Применение различных методов оптимизации для моделей суммаризации кода

Антон Праздничных

НИУ ВШЭ СПб, Машинное обучение и анализ данных

Научный руководитель:
Михаил Евтихиев
JetBrains Research, ML4SE Lab

GitHub repo: https://github.com/JetBrains-Research/optimization-methods/tree/prazdnichnykh_experiments

Стандартная задача оптимизации

Пусть x_i-- i-ый вектор данных, y_i -- соответствующая ему метка;

$f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ -- модель, θ -- вектор весов модели;

$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(x_i))$ -- оптимизируемая функция потерь (ex. $\ell(y, \hat{y}) = \|y - \hat{y}\|^2$)

Базовая идея оптимизации функции $L(\theta)$:

Batch GD:

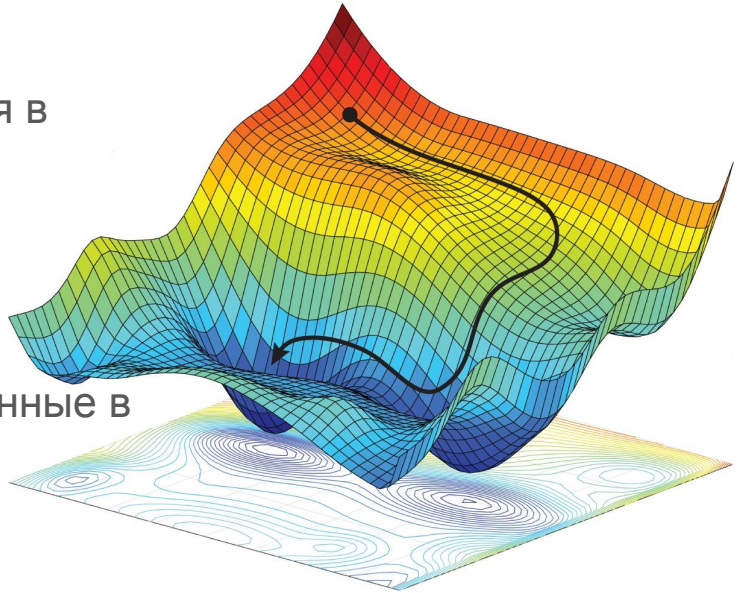
$$\theta_t = \theta_{t-1} - \eta_t \nabla_\theta L(\theta_{t-1}) = \theta_{t-1} - \frac{\eta_t}{n}\sum_{i=1}^{n} \nabla_\theta \ell_i(\theta_{t-1}),$$ где η_t-- learning rate

Stochastic (Minibatch) GD:

shuffle data; for ($x_{i:i+n}$, $y_{i:i+n}$) in shuffled data: $\theta_t = \theta_{t-1} - \eta_t \nabla_\theta L(\theta_{t-1}; x_{i:i+b}, y_{i:i+b})$

Мотивация

- Размерность пространства весов нейронной сети ~10M, поверхность функции потерь может иметь очень сложный ландшафт
- Различные методы оптимизации могут сходиться в разные локальные минимумы, соответствующие моделям разного качества
- В недавнее время предложены множество модификаций стандартных методов
- Мало информации о том, как результаты полученные в оригинальных статьях обобщаются на другие задачи/архитектуры сетей/датасеты



Цели и задачи

Цель: получить информацию о ранжировании исследуемых методов оптимизации на задаче суммаризации кода в имя метода (глобально, на нескольких задачах и датасетах ML4SE)

```
fun [redacted]() {
    val jsonString = Json.encodeToString(fileToId)
    File(ProjectConfig.FILE_ID_PATH).writeText(jsonString)
}
```

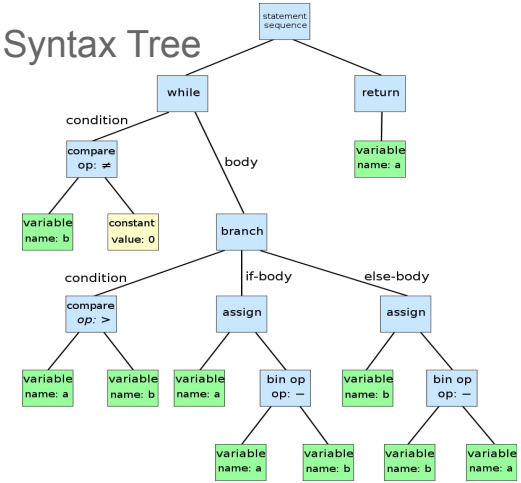
Задачи:

- Выбрать из литературы несколько интересных методов оптимизации для глубокого обучения
- Добавить реализации этих методов в код обучения исследуемой модели и обучить модель, с помощью разных методов
- Оценить (статистическую) значимость улучшений исследуемых модификаций для задачи суммаризации кода

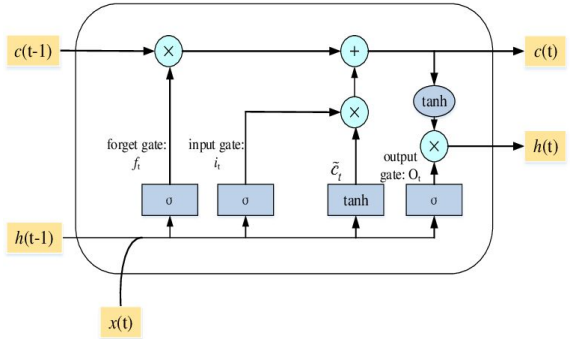
Модель

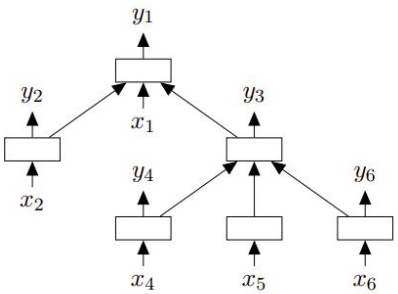
```
while b ≠ 0
  if a > b
    a := a − b
  else
    b := b − a
return a
```

Abstract Syntax Tree



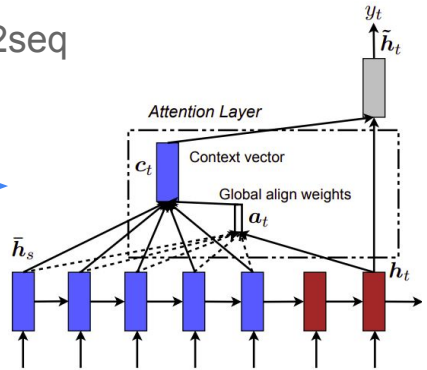
LSTM cell





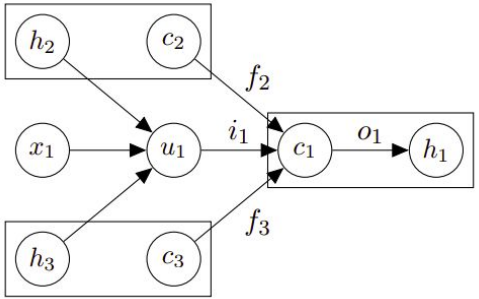
TreeLSTM encoder

tree2seq



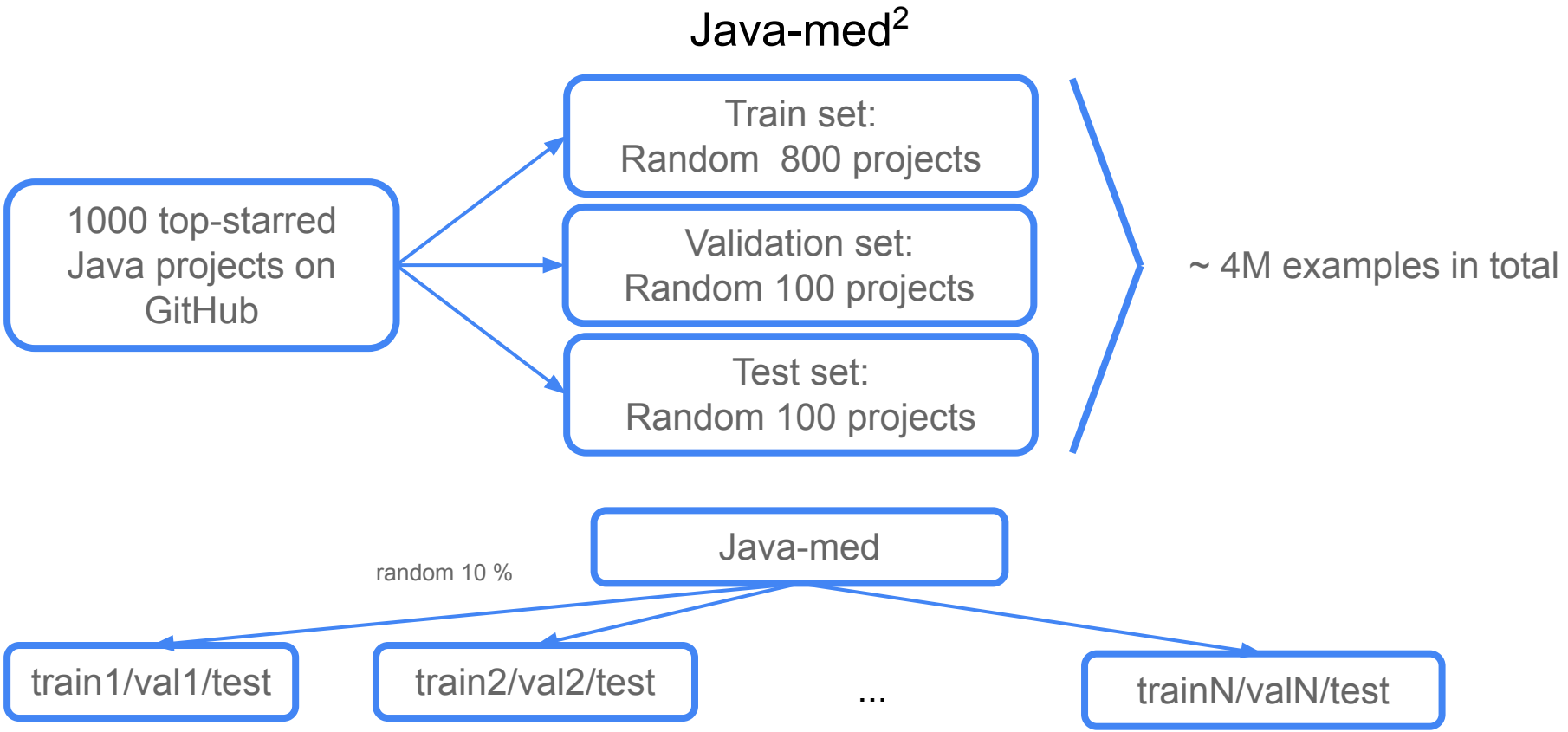
LSTM with attn on subtrees decoder

TreeLSTM[1]



[1] Tai et al., “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks”, arXiv:1503.00075, 2015

Датасет



[2] https://github.com/tech-srl/code2seq#datasets

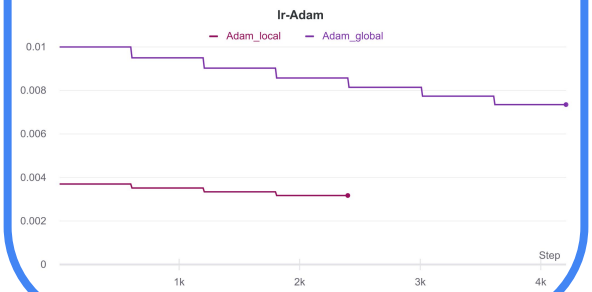
Этапы проекта

Отбор методов

- **SGD** (nesterov accelerated): momentum = 0.95
- **Adam**: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$
- **RAdam**: as Adam
- **Lamb**: as Adam
- **Lookahead** + SGD/Adam/RAdam as base optimizer: k = 5, $\alpha = 0.5$
- **SWA**: cyclic lr, cycle len=epoch, min lr = 0.001, max lr = 0.0074
- **Locals**: globals as locals, SVRG, SWA, BB

Проведение численных экспериментов

- Batch size = 512
- Base lr = 0.01 (g) / 0.0037 (l)
- Lr schedule = base*0.95^{epochs}
- #epochs = 7 (g) \ 4 (l)
- Fixed initialization
- No weight decay and fixed dropout



Проверка гипотез о ранжировании методов

Как сравнивать?

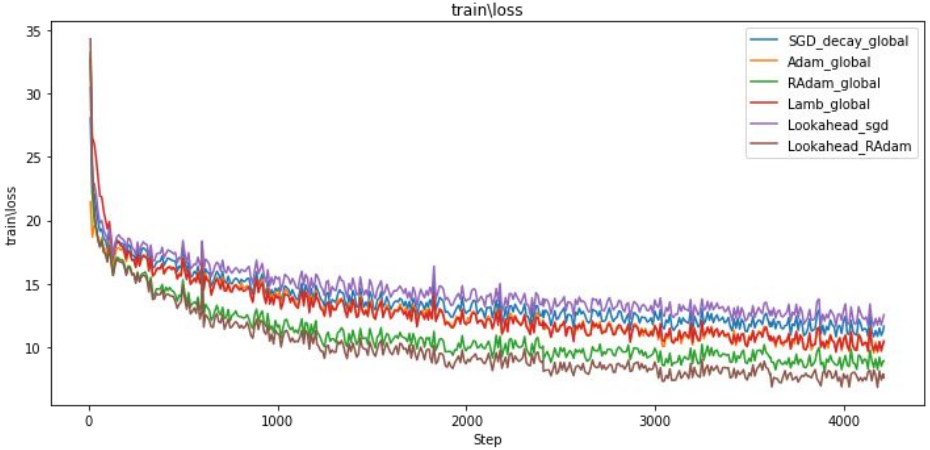
- ROUGE-1/2/L
- BLEU
- Meteor
- BERTScore

Значимо ли различие?

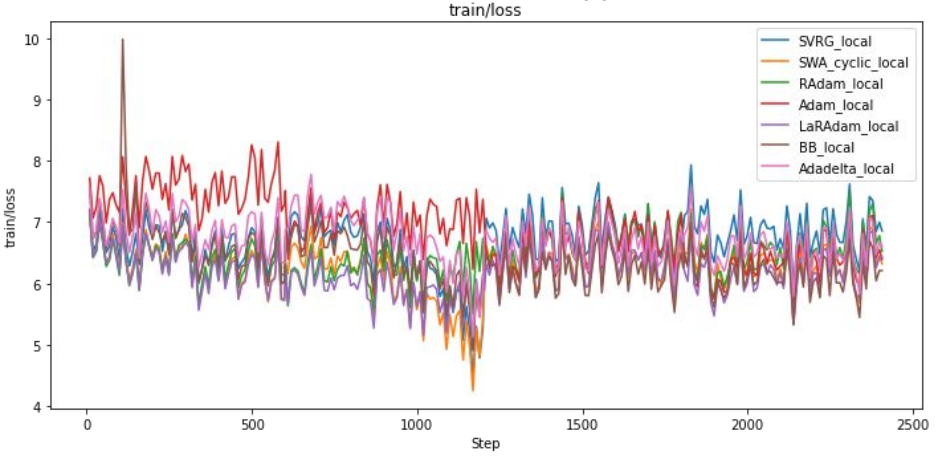
- Wilcoxon signed-rank test
- Mann–Whitney U-test

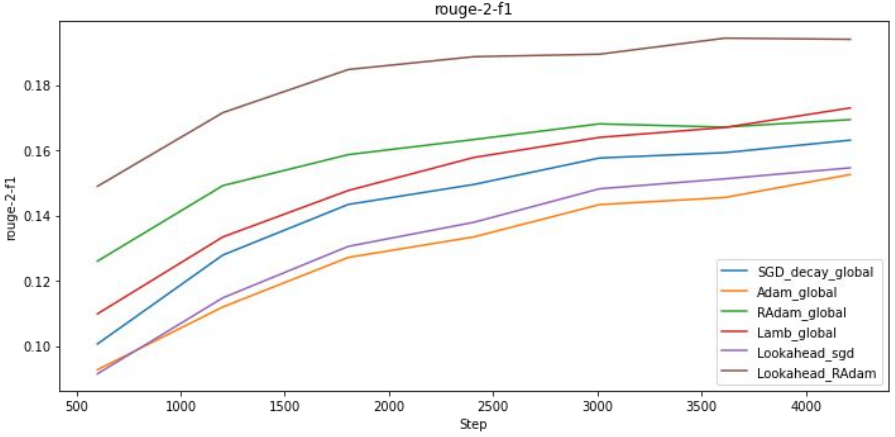
Результаты экспериментов. Training curves

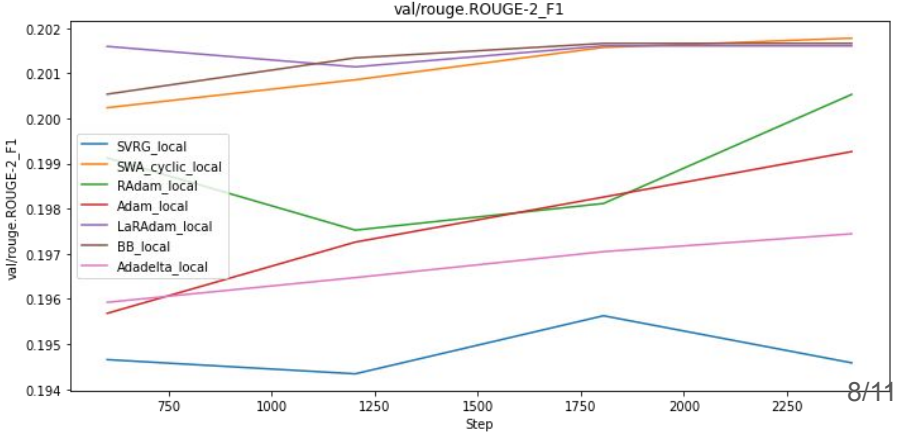
Глобальные методы



Локальные методы







Результаты экспериментов. Test Metrics

Global methods test

Method	BLEU	meteor	rouge1	rouge2	rougeL
SGD	0.2777	0.2647	0.3839	0.1597	0.3940
Adam	0.2474	0.2443	0.3556	0.1416	0.3651
RAdam	0.3108	0.2784	0.3879	0.1781	0.4010
Lamb	0.3158	0.2853	0.3980	0.1809	0.4095
LaSGD	0.2451	0.2475	0.3689	0.1424	0.3792
LaRAdam	**0.3647**	**0.3151**	**0.4310**	**0.2080**	**0.4428**
Upside SGD-LaRadam, %	31.3	19.1	12.3	30.3	12.4

Local methods test

Method	BLEU	meteor	rouge1	rouge2	rougeL
Adadelta	0.3705	0.3191	0.4317	0.2126	0.4430
Barzelai-Borwein	**0.3863**	**0.3268**	**0.4414**	**0.2172**	**0.4531**
LaRAdam (local)	0.3843	0.3260	0.4400	0.2171	0.4511
SVRG	0.3663	0.3172	0.4306	0.2107	0.4419
SWA	0.3851	0.3263	0.4407	0.2173	0.4523
Upside LaRAdam(global)-BB	5.9	3.7	2.4	4.4	2.3

Результаты

1. На нашей задаче суммаризации кода в имя метода почти все исследуемые исследуемые модификации стандартных методов показывают **улучшение** результатов, при этом разница между ранее использовавшимся методом и лучшим из модификаций достигает **31%** (в зависимости от метрики)
2. Лучшим из глобальных методов является **Lookahead** с **RAdam** в качестве базового оптимизатора
3. Лучшими из локальных методов являются **SWA** и **Barzilai-Borwein**, которые показывают практически одинаковые результаты и в среднем дают **улучшение ~1%** по всем метрикам относительно стартовой точки

Дальнейшая работа

1. Прогнать лучший и худший методы на полном Java-med, получив тем самым более точную оценку метрик, и проверить гипотезу о значимости улучшений
2. Провести аналогичные эксперименты на других архитектурах моделей (CNN, Transformer) и задачах SE (например, суммаризация кода в docstring)
3. Попробовать другие функции потерь кроме кросс-энтропии
4. Поэкспериментировать с различными методами регуляризации
5. Исследовать влияние инициализации на обучение, проверить гипотезу о существовании удалённых “глобальных” минимумов

Спасибо за внимание!

TreeLSTM

$$\tilde{h}_j = \sum_{k \in C(j)} h_k,$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right),$$

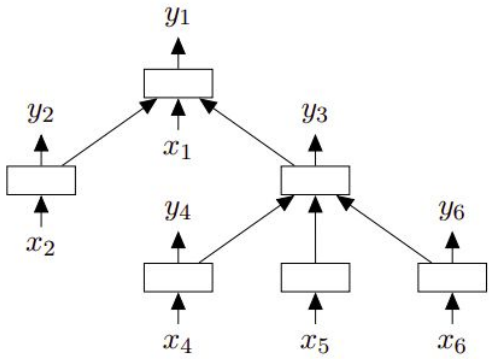
$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right),$$

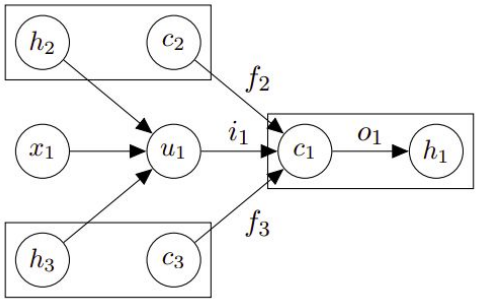
$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right),$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right),$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k,$$

$$h_j = o_j \odot \tanh(c_j),$$





Tree2Seq

Code of method

```
@Override protected void preHead(Page.HTML<_> html) {
  commonPreHead(html);
}
```

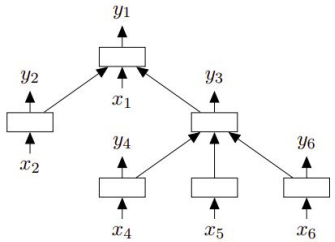
Abstract Syntax Tree


```
{"label":"pre|head","AST":[{"node":"METHOD","children":[1,2,3,4,13],"token":"<EMPTY>"},-
{"node":"MODIFIER_LIST|PROTECTED_KEYWORD","token":"protected"},{"node":"TYPE|-
VOID_KEYWORD","token":"void"},{"node":"IDENTIFIER","token":"<MN>"},{"node":"PARAMETER_LIST|-
PARAMETER","children":[5,12],"token":"<EMPTY>"},{"node":"TYPE|JAVA_CODE_REFERENCE","children":-
[6,7,8],"token":"<EMPTY>"},{"node":"JAVA_CODE_REFERENCE|IDENTIFIER","token":"page"},-
{"node":"IDENTIFIER","token":"html"},{"node":"REFERENCE_PARAMETER_LIST","children":-
[9,10,11],"token":"<EMPTY>"},{"node":"LT","token":"<"},{"node":"TYPE|JAVA_CODE_REFERENCE|-
IDENTIFIER","token":"_"},{"node":"GT","token":">"},{"node":"IDENTIFIER","token":"html"},-
{"node":"CODE_BLOCK","token":"<EMPTY>"}]}
```

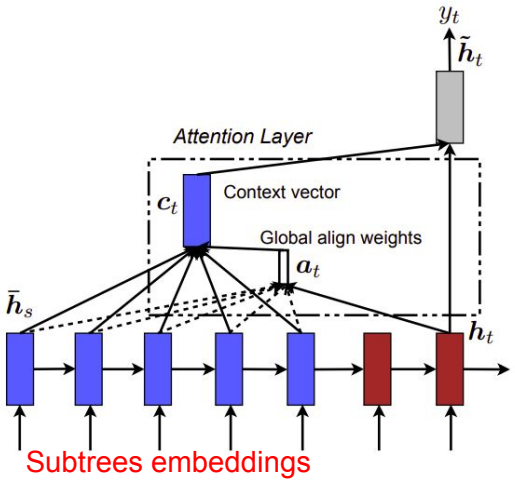

Node Embeddings

{“node”: …, “token”: ...} → token_embedding + node_embedding $\in \mathbb{R}^{\text{emb\_size}}$

TreeLSTM Encoder



LSTM Decoder



$$\alpha_i = \text{softmax}\left(h_t^T W_a h_i^s\right) \qquad c_t = \sum_{i=1}^{n} \alpha_i h_i^s$$

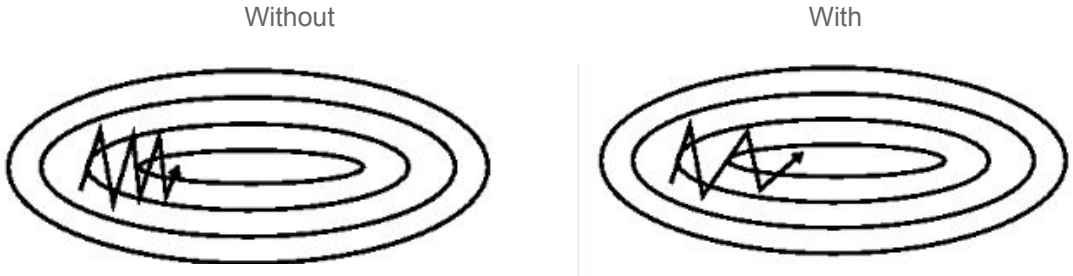
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \qquad p(y_t|y_{<t}, x) = \text{softmax}(\boldsymbol{W_s}\tilde{\boldsymbol{h}}_t)$$

[2] Luong et. al, “Effective Approaches to Attention-based Neural Machine Translation”, arXiv:1508.04025

Some improvements of SGD

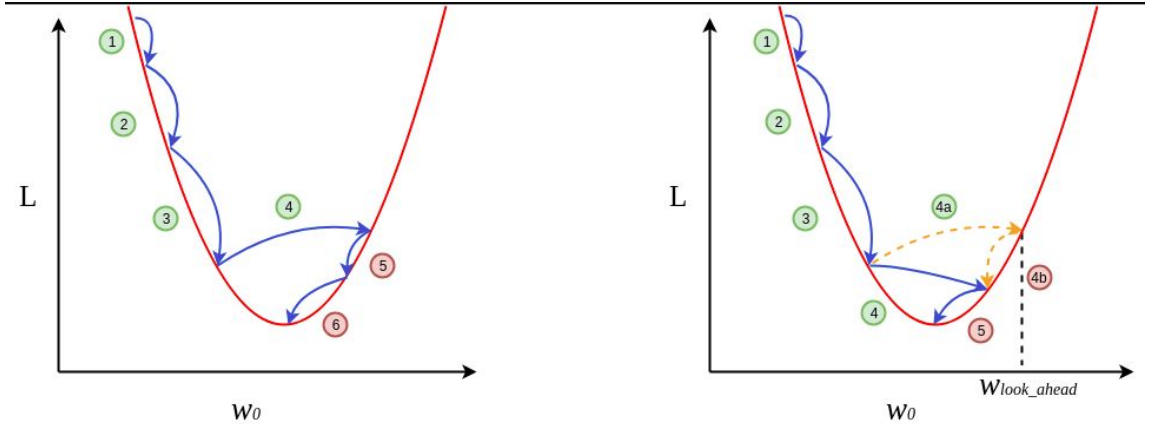
- Momentum:

$$m_t = \beta m_{t-1} + \eta_t \nabla L(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$



- Nesterov acceleration: $m_t = \beta m_{t-1} + \eta_t \nabla L(\theta_{t-1} - \beta m_{t-1})$

Classic Momentum



Nesterov acceleration

[3] Sutskever et al., “On the importance of initialization and momentum in deep learning”, JMLR: W&CP volume 28, 2013

Adaptive methods. Adam

[3] Kingma et al. “Adam: a Method for Stochastic Optimization. ICLR 1–13, 2015

Идея: Обновлять параметры в зависимости от того, насколько много значимо среднее направление их импульса относительно второго момента

Шаг обновления:

Вычисляем градиент:

$$g_t = \nabla L(\theta_{t-1})$$

Накапливаем импульс:

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

Оцениваем частоту обновлений:

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

Корректируем смещение:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Обновляем параметры:

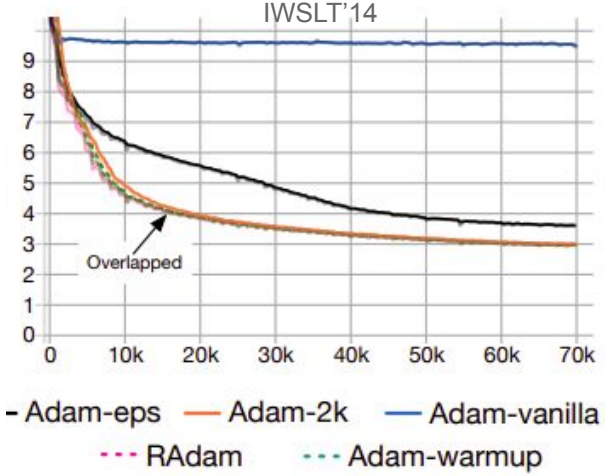
$$\theta_t = \theta_{t-1} - \frac{\eta_t}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$$

~ signal-noise ratio

Adaptive methods disadvantage. Warm-up

Есть проблема: На начальных шагах обучения дисперсия адаптивного шага может быть очень большой → можем делать очень большие шаги на основе маленького количества данных и попадать в плохие локальные минимумы

Train loss vs #iterations for Transformers on De-En IWSLT'14



Проблема:

$$\psi_t = \frac{1}{\sqrt{v_t} + \varepsilon}$$

Наивные решения

- Adam-warmup: $\eta_t = t\eta_0, \text{ if } t < T_{warmup}$
- Adam-eps: $\varepsilon \sim 10^{-4}$ (стандартный $\varepsilon \sim 10^{-8}$)
- Adam-2k: первые 2k итераций обновляем только ψ_t

[4] Liu et al. "On The Variance Of The Adaptive Learning Rate And Beyond", arXiv:1908.03265, 2020

RAdam

Идея: давайте явно ограничим дисперсию адаптивного множителя

Адаптивный множитель (Adam): $\psi(g_1, \dots, g_t) = \sqrt{\frac{1-\beta_2^t}{(1-\beta_2)\sum_{i=1}^t \beta_2^{t-i} g_i^2}}$

Его дисперсия*: $D\psi(.) = \tau^2 \left(\frac{\rho}{\rho-2} - \frac{\rho 2^{2\rho-5}}{\pi} \mathcal{B}\left(\frac{\rho-1}{2}, \frac{\rho-1}{2}\right)\right) \approx \frac{\rho}{2(\rho-2)(\rho-4)\sigma^2}$

Добавляем множитель, минимизирующий дисперсию: $D[r_t \psi(g_1, \dots, g_t)] = D_{\min}, \quad r_t = \sqrt{\frac{(\rho_t-2)(\rho_t-4)\rho_\infty}{(\rho_\infty-2)(\rho_\infty-4)\rho_t}}$

*Немного математики:

Пусть $g_i \sim N(0, \sigma^2)$, тогда $\psi(.) \approx \sqrt{\frac{t}{\sum_{i=1}^t g_i^2}} \Rightarrow \psi^2(.) \sim \text{Scale-inv-}\chi^2(t, 1/\sigma^2)$

На самом деле $\psi^2(.) \sim \text{Scale-inv-}\chi^2(\rho, 1/\sigma^2)$, где $\rho = \rho(t) = \frac{2}{1-\beta_2} - 1 - \frac{2t\beta_2^t}{1-\beta^t} = \rho_\infty - \frac{2t\beta_2^t}{1-\beta^t}$

В этом случае при $\rho > 4$ верно $D\psi(.) = \tau^2 \left(\frac{\rho}{\rho-2} - \frac{\rho 2^{2\rho-5}}{\pi} \mathcal{B}\left(\frac{\rho-1}{2}, \frac{\rho-1}{2}\right)\right) \approx \frac{\rho}{2(\rho-2)(\rho-4)\sigma^2}$

RAdam

Algorithm 2: Rectified Adam. All operations are element-wise.

Input: $\{\alpha_t\}_{t=1}^{T}$: step size, $\{\beta_1, \beta_2\}$: decay rate to calculate moving average and moving 2nd moment, θ_0: initial parameter, $f_t(\theta)$: stochastic objective function.

Output: θ_t: resulting parameters

$m_0, v_0 \leftarrow 0, 0$ (Initialize moving 1st and 2nd moment)

$\rho_\infty \leftarrow 2/(1-\beta_2) - 1$ (Compute the maximum length of the approximated SMA)

while $t = \{1, \cdots, T\}$ **do**

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)

  $v_t \leftarrow 1/\beta_2 v_{t-1} + (1-\beta_2) g_t^2$ (Update exponential moving 2nd moment)

  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$ (Update exponential moving 1st moment)

  $\widehat{m_t} \leftarrow m_t / (1-\beta_1^t)$ (Compute bias-corrected moving average)

  $\rho_t \leftarrow \rho_\infty - 2t\beta_2^t/(1-\beta_2^t)$(Compute the length of the approximated SMA)

  if *the variance is tractable, i.e.,* $\rho_t > 4$ **then**

    $l_t \leftarrow \sqrt{(1-\beta_2^t)/v_t}$ (Compute adaptive learning rate)

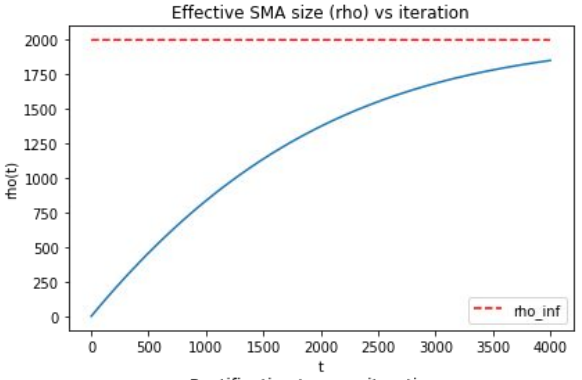
    $r_t \leftarrow \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}}$ (Compute the variance rectification term)

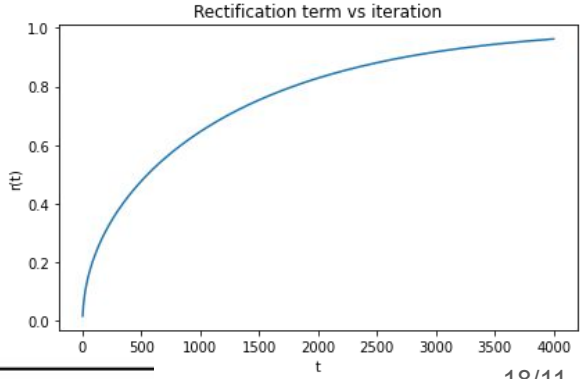
    $\theta_t \leftarrow \theta_{t-1} - \alpha_t r_t \widehat{m_t} l_t$ (Update parameters with adaptive momentum)

  else

    $\theta_t \leftarrow \theta_{t-1} - \alpha_t \widehat{m_t}$ (Update parameters with un-adapted momentum)

return θ_T





Lamb

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

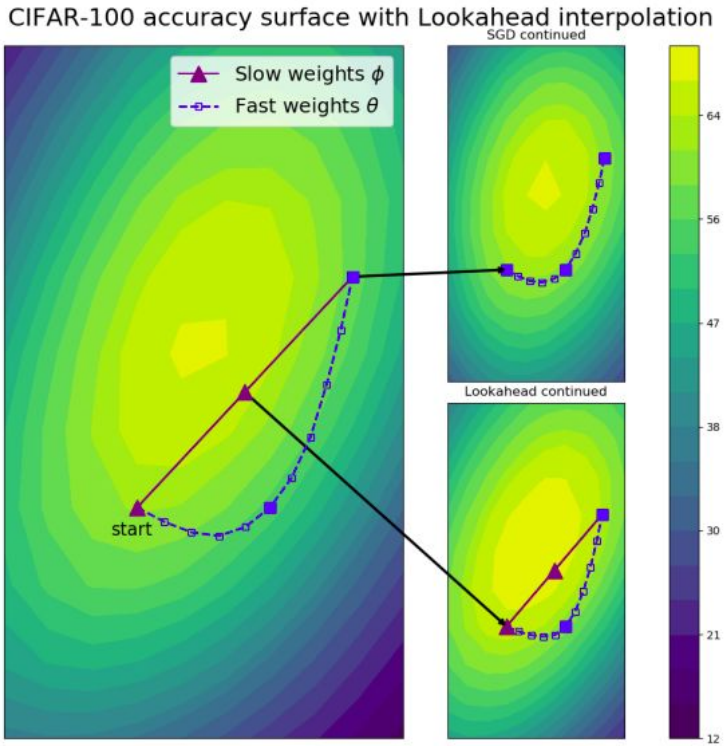
$x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|r_t^{(i)} + \lambda x_t^{(i)}\|} (r_t^{(i)} + \lambda x_t^{(i)})$ $\quad i \in \{0, 1, \ldots, h\}$ -- layer, $\quad \phi(z) = \min(\max(z, \gamma_l), \gamma_u)$

end for

You et al. “Large Batch Optimization For Deep Learning: Training Bert In 76 Minutes”, 2020, arXiv:1904.00962

Lookahead

Идея: разбиваем обучение на “быстрые” и “медленные” веса. Это поможет снизить дисперсию обновлений и сделать обучение более стабильным и устойчивым к изменению гиперпараметров



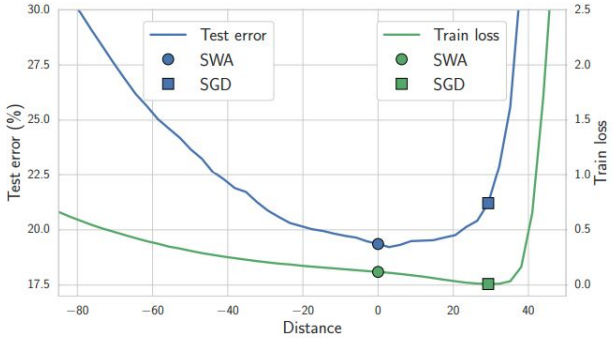
Algorithm 1 Lookahead Optimizer:

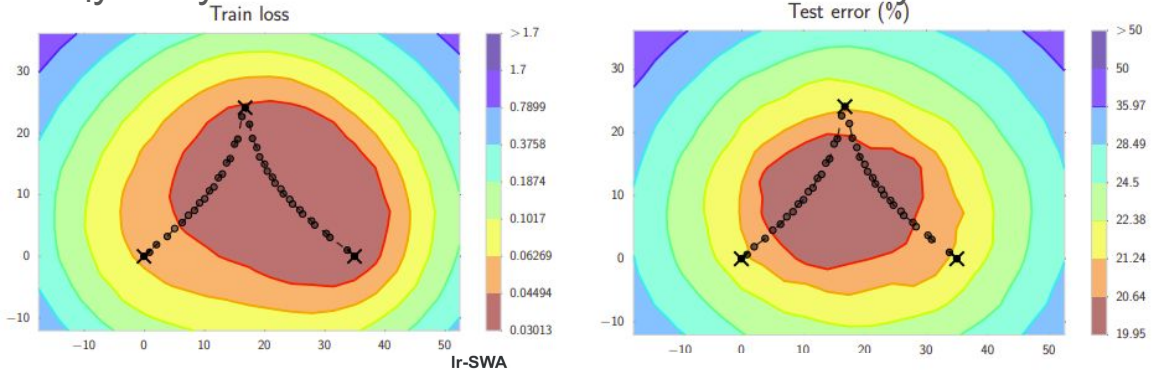
Require: Initial parameters ϕ_0, objective function L
Require: Synchronization period k, slow weights step size α, optimizer A
for $t = 1, 2, \ldots$ **do**
  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
  for $i = 1, 2, \ldots, k$ **do** K ~ 5..10
    sample minibatch of data $d \sim \mathcal{D}$
    $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
  end for
  Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
end for
return parameters ϕ

[5] Zhang et al., “Lookahead Optimizer: k steps forward, 1 step back”, NeurIPS 2019

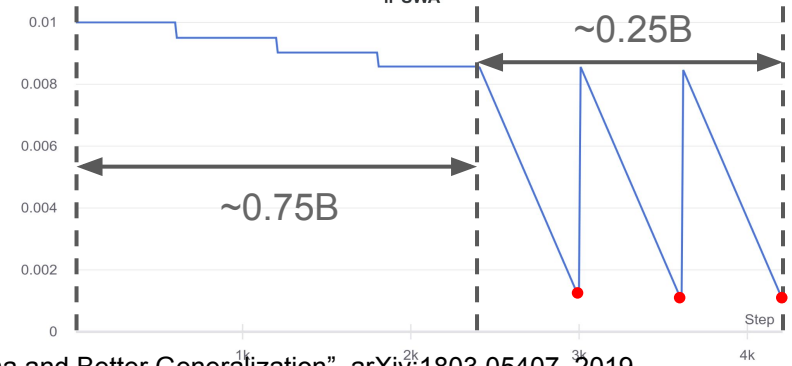
Stochastic Weight Averaging

Идея: SGD склонен сходиться к точкам, расположенным близко к границам локальных минимумов. Давайте возьмем несколько точек с конца траектории обучения SGD, усредним их и получим точку, лежащую глубоко в области локального минимума





B -- общий бюджет обучения



[7] Izmailov et al., “Averaging Weights Leads to Wider Optima and Better Generalization”, arXiv:1803.05407, 2019

Stochastic Variance Reduced Gradients (SVRG)

Procedure SVRG

Parameters update frequency m and learning rate η
Initialize $\tilde{w}_0$
Iterate: for $s = 1, 2, \ldots$
  $\tilde{w} = \tilde{w}_{s-1}$
  $\tilde{\mu} = \frac{1}{n} \sum_{i=1}^{n} \nabla \psi_i(\tilde{w})$
  $w_0 = \tilde{w}$
  Iterate: for $t = 1, 2, \ldots, m$
    Randomly pick $i_t \in \{1, \ldots, n\}$ and update weight
      $w_t = w_{t-1} - \eta(\nabla \psi_{i_t}(w_{t-1}) - \nabla \psi_{i_t}(\tilde{w}) + \tilde{\mu})$
  end
  option I: set $\tilde{w}_s = w_m$
  option II: set $\tilde{w}_s = w_t$ for randomly chosen $t \in \{0, \ldots, m-1\}$
end

"In order to apply SVRG to nonconvex problems such as neural networks, it is useful to start with an initial vector w_0 that is close to a local minimum (which may be obtained with SGD), and then the method can be used to accelerate the local convergence rate of SGD (which may converge very slowly by itself). If the system is locally (strongly) convex, then Theorem 1 can be directly applied, which implies local geometric convergence rate with a constant learning rate."

Theorem 1. *Consider SVRG in Figure 1 with option II. Assume that all ψ_i are convex and both (5) and (6) hold with $\gamma > 0$. Let $w_* = \arg\min_w P(w)$. Assume that m is sufficiently large so that*

$$\alpha = \frac{1}{\gamma\eta(1 - 2L\eta)m} + \frac{2L\eta}{1 - 2L\eta} < 1,$$

then we have geometric convergence in expectation for SVRG:

$$\mathbb{E}\, P(\tilde{w}_s) \leq \mathbb{E}\, P(w_*) + \alpha^s [P(\tilde{w}_0) - P(w_*)]$$

Barzilai-Borwein

Разложим лосс в окрестности текущей точки w_k до 2ого порядка:

$$L(w_k + \Delta w) \approx L(w_k) + \nabla L(w_k)^T \Delta w + \frac{1}{2} \Delta w^T H(w_k) \Delta w$$

Связь градиентов и гессиана:

$$\nabla L(w_k + \Delta w) = \nabla L(w_k) + H(w_k)\Delta w \Leftrightarrow \nabla L(w_k + \Delta w) - \nabla L(w_k) = H(w_k)\Delta w \quad (1)$$

Оптимальное обновление:

$$\Delta w_k = -H(w_k)^{-1} \nabla L(w_k)$$

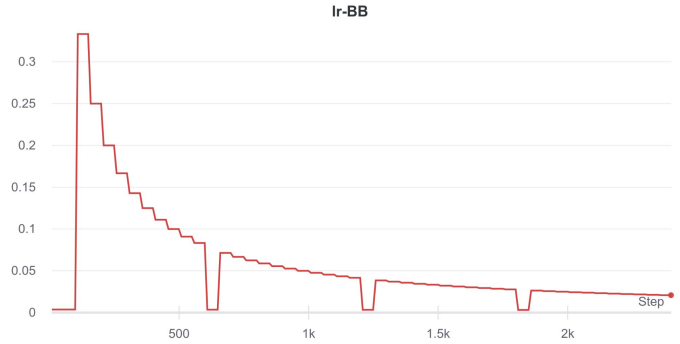
В случае нейронных сетей гессиан просто не влезет в память, давайте на каждом шаге пытаться приблизить $H(w_k)^{-1}$ с помощью $(\alpha_k^{-1}\mathbf{I})$

- Least-squares problem: (let $\beta = \alpha^{-1}$)

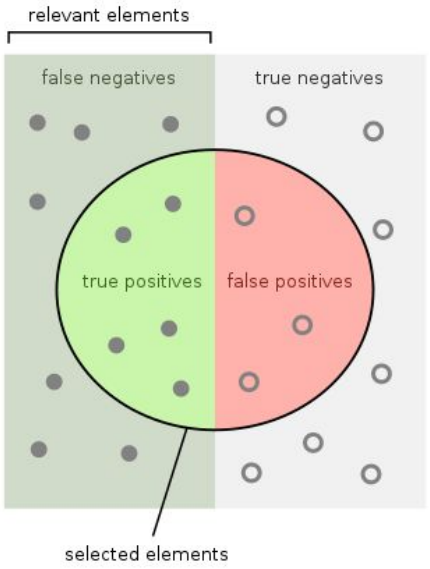
$$\alpha_k^{-1} = \arg\min_{\beta} \frac{1}{2} \| \boldsymbol{s}^{(k-1)} \beta - \boldsymbol{y}^{(k-1)} \|^2 \implies \alpha_k^1 = \frac{(\boldsymbol{s}^{(k-1)})^T \boldsymbol{s}^{(k-1)}}{(\boldsymbol{s}^{(k-1)})^T \boldsymbol{y}^{(k-1)}}$$

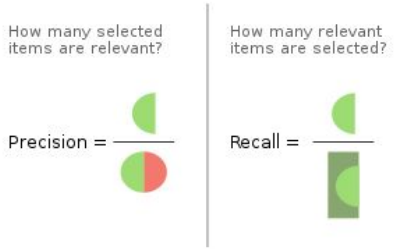
- Alternative Least-squares problem:

$$\alpha_k = \arg\min_{\alpha} \frac{1}{2} \| \boldsymbol{s}^{(k-1)} - \boldsymbol{y}^{(k-1)} \alpha \|^2 \implies \alpha_k^2 = \frac{(\boldsymbol{s}^{(k-1)})^T \boldsymbol{y}^{(k-1)}}{(\boldsymbol{y}^{(k-1)})^T \boldsymbol{y}^{(k-1)}}$$



Metrics





$$precision = \frac{\text{number of n-grams found in output and reference}}{\text{number of n-grams in } \textbf{output}}$$

$$recall = \frac{\text{number of n-grams found in output and reference}}{\text{number of n-grams in } \textbf{reference}}$$

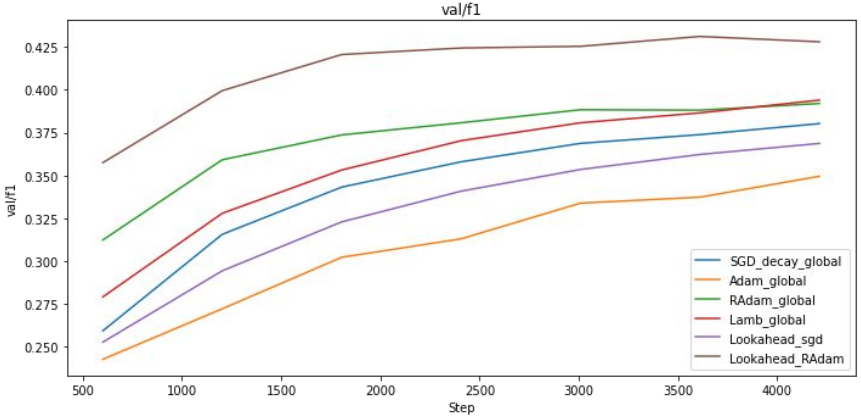
$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

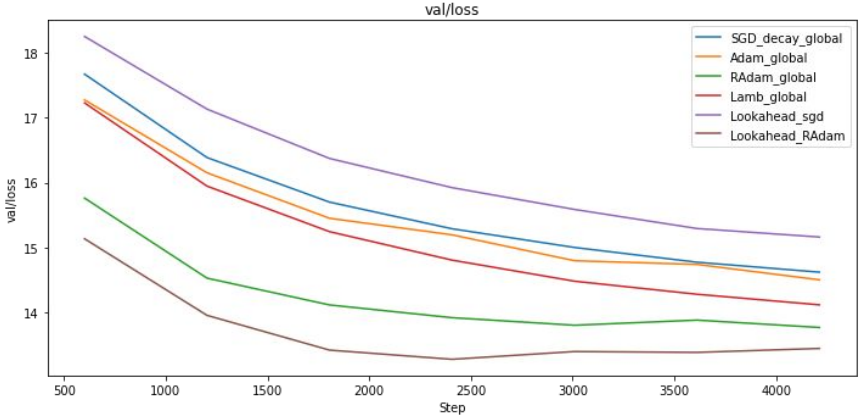
$$BLEU = \min\left(1, \frac{\text{output len}}{\text{reference len}}\right)\left(\prod_{i=1}^{3} \text{precision}_i\right)^{\frac{1}{3}}$$

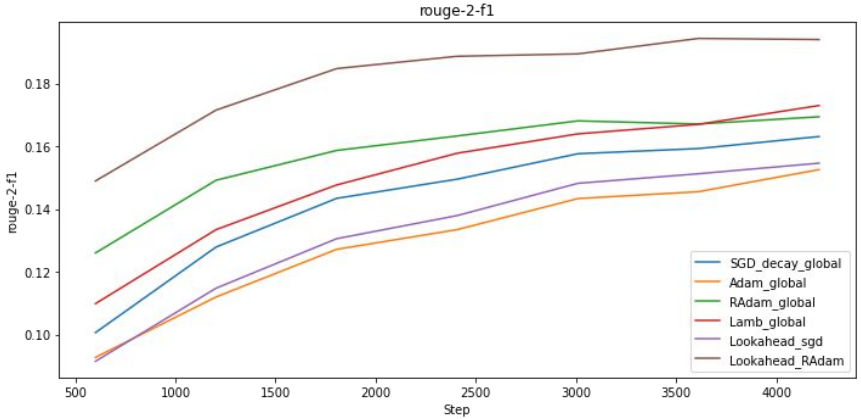
Meteor

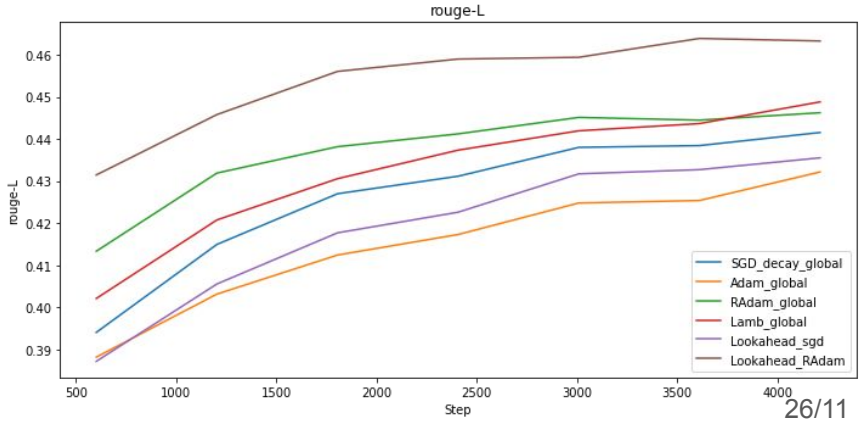
Reference	the cat sat on the mat
Hypothesis	on the mat sat the cat
Score	$0.5000 = \underset{\text{Fmean}}{1.0000} \times (1 - \underset{\text{Penalty}}{0.5000})$
Fmean	$1.0000 = 10 \times \underset{\text{Precision}}{1.0000} \times \frac{\overset{\text{Recall}}{1.0000}}{\underset{\text{Recall}}{1.0000} + 9 \times \underset{\text{Precision}}{1.0000}}$
Penalty	$0.5000 = 0.5 \times \underset{\text{Fragmentation}}{1.0000^3}$
Fragmentation	$1.0000 = \frac{\overset{\text{Chunks}}{6.0000}}{\underset{\text{Matches}}{6.0000}}$
Reference	the cat sat on the mat
Hypothesis	the cat sat on the mat
Score	$0.9977 = \underset{\text{Fmean}}{1.0000} \times (1 - \underset{\text{Penalty}}{0.0023})$
Fmean	$1.0000 = 10 \times \underset{\text{Precision}}{1.0000} \times \frac{\overset{\text{Recall}}{1.0000}}{\underset{\text{Recall}}{1.0000} + 9 \times \underset{\text{Precision}}{1.0000}}$
Penalty	$0.0023 = 0.5 \times \underset{\text{Fragmentation}}{0.1667^3}$
Fragmentation	$0.1667 = \frac{\overset{\text{Chunks}}{1.0000}}{\underset{\text{Matches}}{6.0000}}$

Experiment results. Global methods

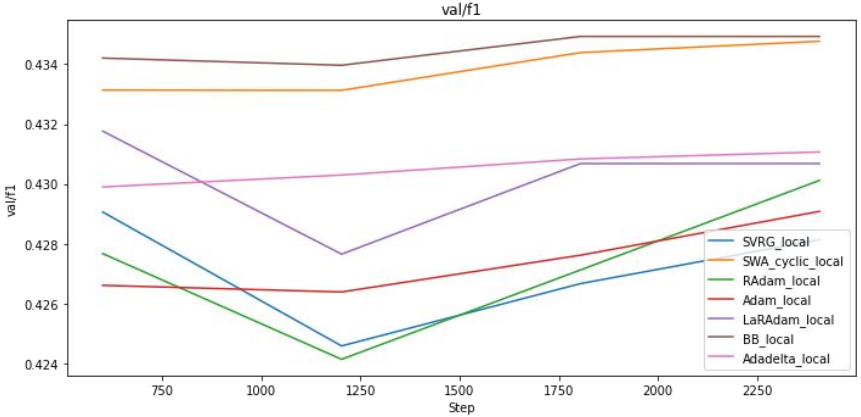


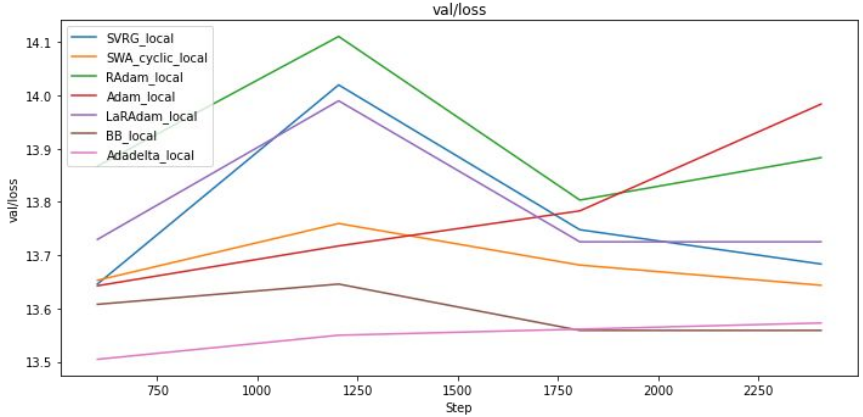


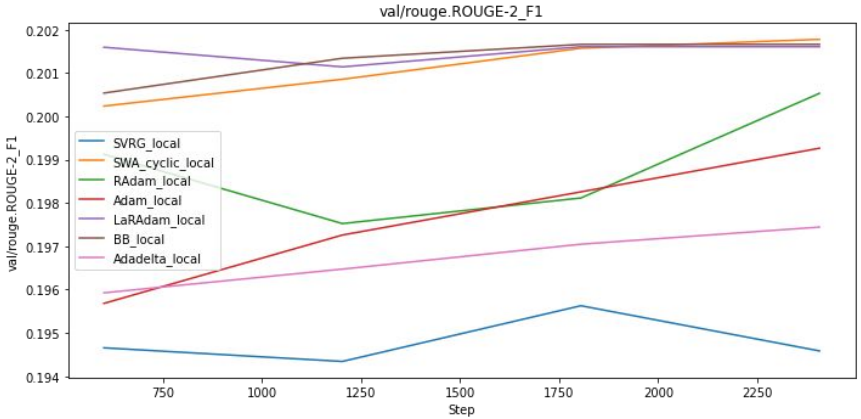


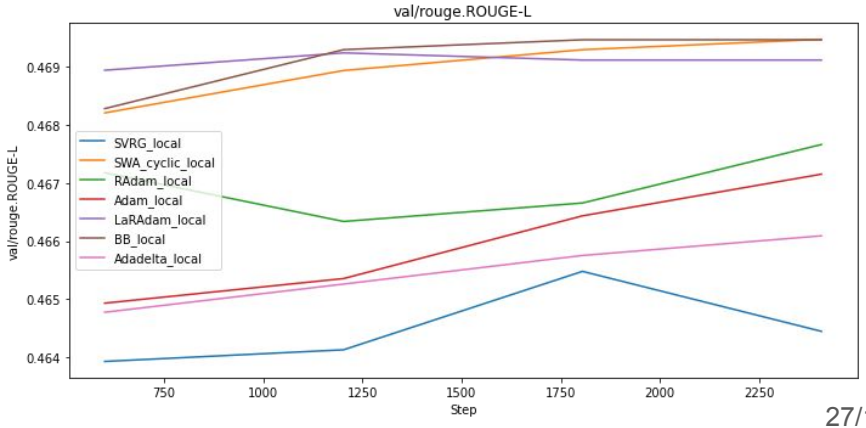


Experiment results. Local methods









RAdam performance

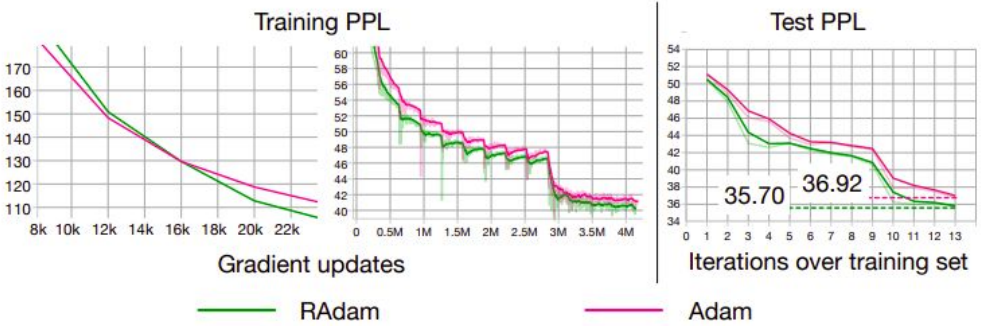


Figure 4: Language modeling (LSTMs) on the One Billion Word.

Table 1: Image Classification

	Method	Acc.
CIFAR10	SGD	91.51
	Adam	90.54
	RAdam	91.38
ImageNet	SGD	69.86
	Adam	66.54
	RAdam	67.62

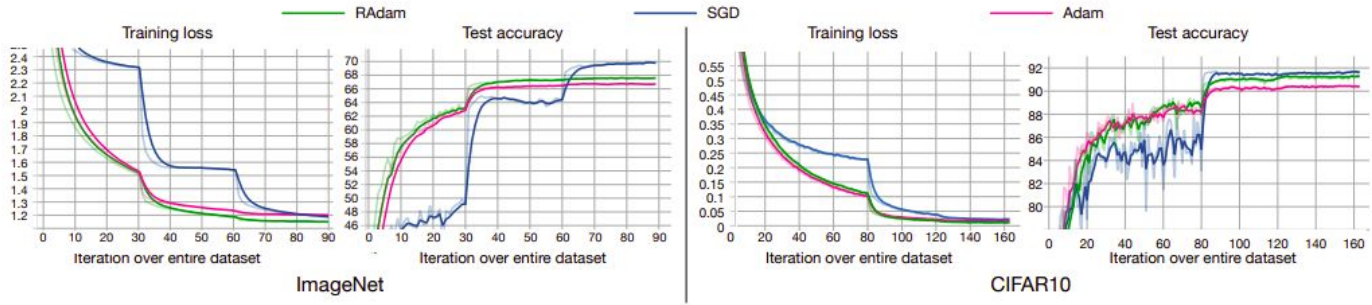


Figure 5: Training of ResNet-18 on the ImageNet and ResNet-20 on the CIFAR10 dataset.

SWA performance

Table 1: Accuracies (%) of SWA, SGD and FGE methods on CIFAR-100 and CIFAR-10 datasets for different training budgets. Accuracies for the FGE ensemble are from Garipov et al. [2018].

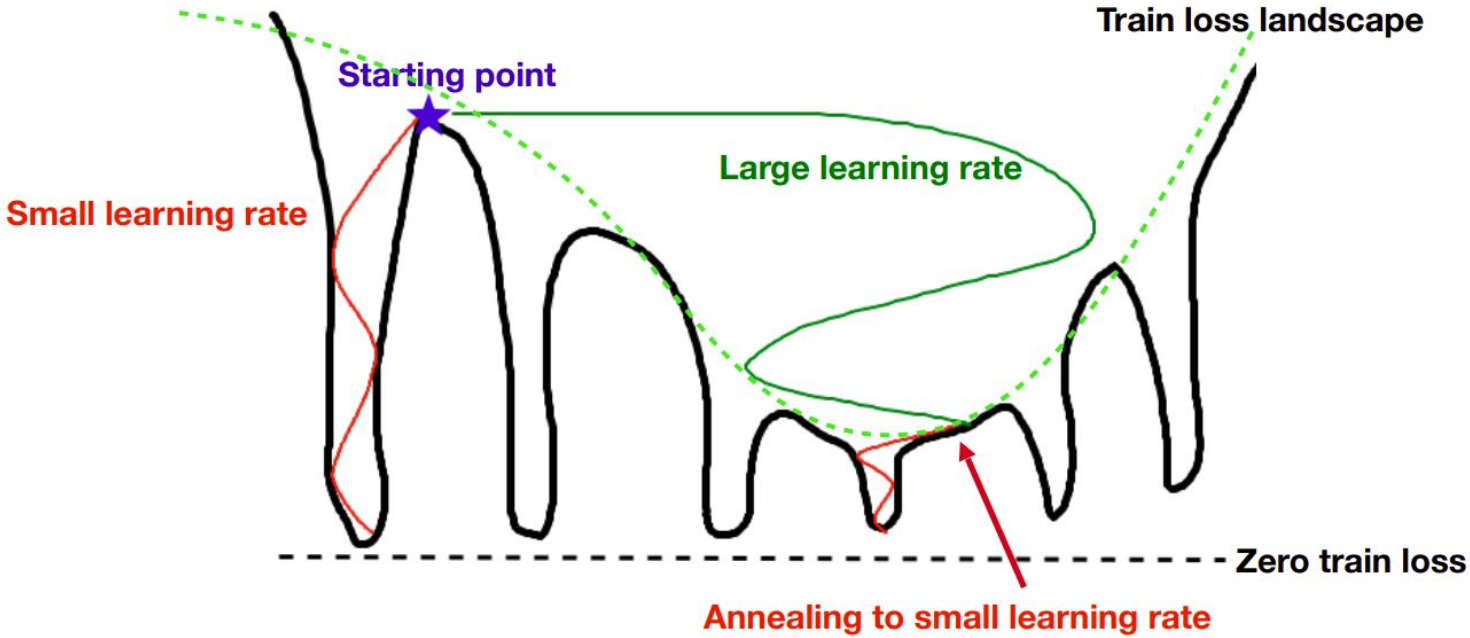
DNN (Budget)	SGD	FGE (1 Budget)	SWA		
			1 Budget	1.25 Budgets	1.5 Budgets
CIFAR-100					
VGG-16 (200)	72.55 ± 0.10	74.26	73.91 ± 0.12	74.17 ± 0.15	74.27 ± 0.25
ResNet-164 (150)	78.49 ± 0.36	79.84	79.77 ± 0.17	80.18 ± 0.23	80.35 ± 0.16
WRN-28-10 (200)	80.82 ± 0.23	82.27	81.46 ± 0.23	81.91 ± 0.27	82.15 ± 0.27
PyramidNet-272 (300)	83.41 ± 0.21	–	–	83.93 ± 0.18	84.16 ± 0.15
CIFAR-10					
VGG-16 (200)	93.25 ± 0.16	93.52	93.59 ± 0.16	93.70 ± 0.22	93.64 ± 0.18
ResNet-164 (150)	95.28 ± 0.10	95.45	95.56 ± 0.11	95.77 ± 0.04	95.83 ± 0.03
WRN-28-10 (200)	96.18 ± 0.11	96.36	96.45 ± 0.11	96.64 ± 0.08	96.79 ± 0.05
ShakeShake-2x64d (1800)	96.93 ± 0.10	–	–	97.16 ± 0.10	97.12 ± 0.06

LAMB performance

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

Batch Size	512	1K	2K	4K	8K	16K	32K
LARS	90.717	90.369	90.748	90.537	90.548	89.589	diverge
LAMB	91.752	91.761	91.946	91.137	91.263	91.345	91.475

Hypothesis


Train loss landscape
Starting point
Large learning rate
Small learning rate
Zero train loss
Annealing to small learning rate