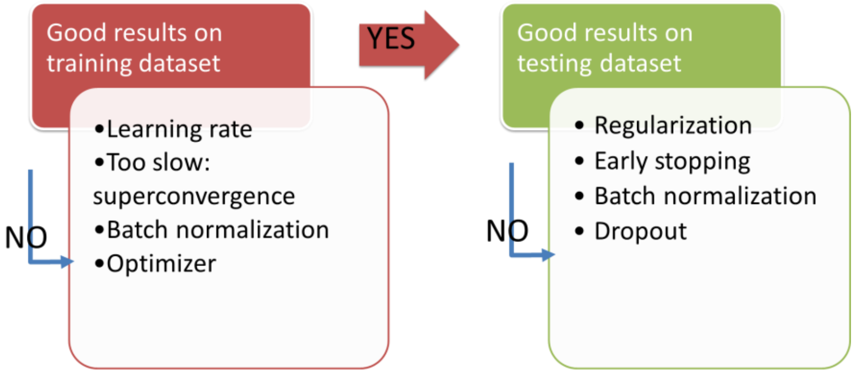
Lecture 4-3 Optimizer: Gradient Descent Optimization

Tian Sheuan Chang



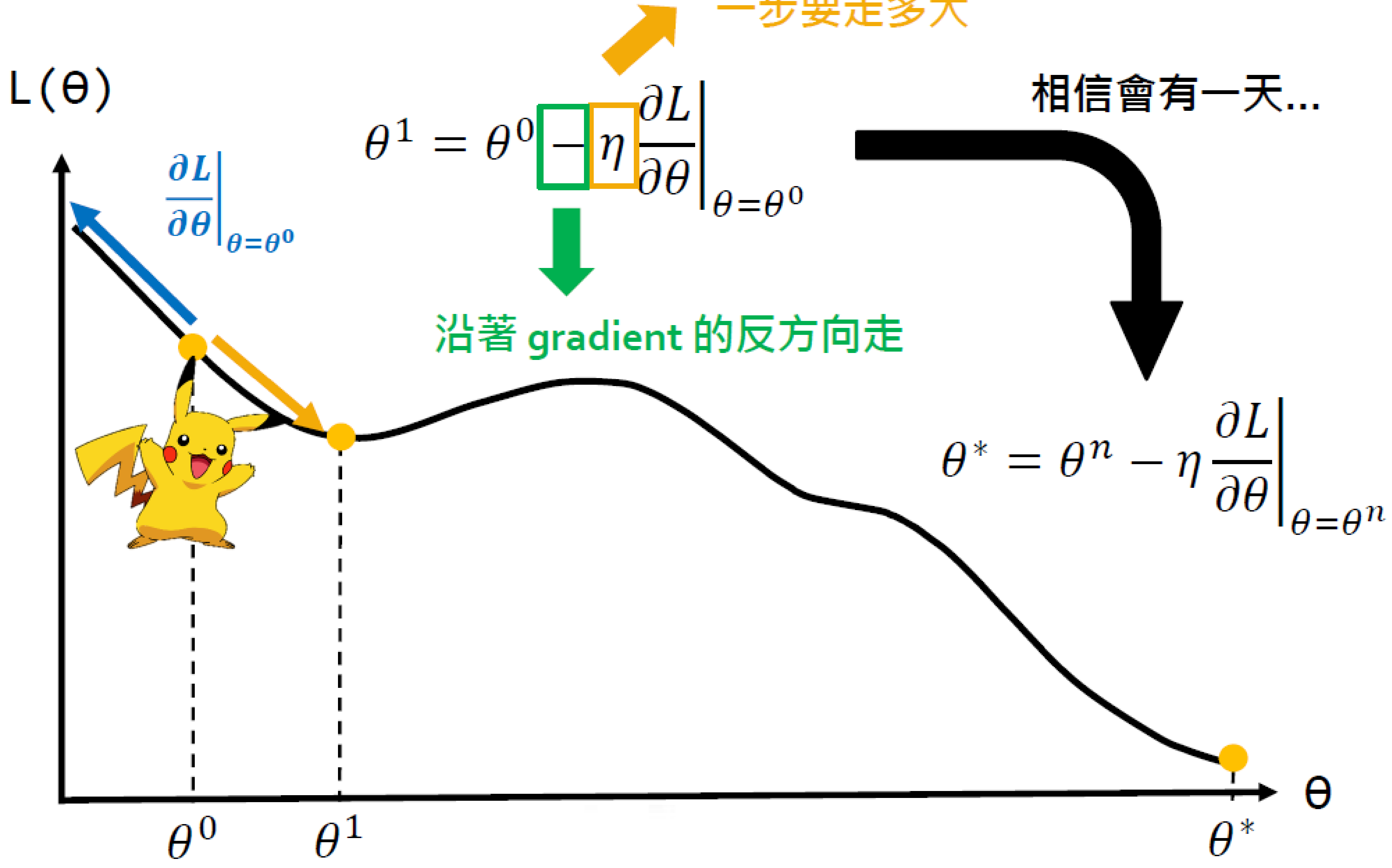
https://ruder.io/optimizing-gradient-descent/index.html#adam

Outline

- Gradient descent variants
  - Batch gradient descent/Stochastic gradient descent/ Mini-batch gradient descent
- Challenges
- Gradient descent optimization algorithms
  - Momentum/Nesterov accelerated gradient
  - Adagrad/Adadelta/RMSprop/
  - Adam/AdaMax//Nadam
  - Visualization of algorithms
  - Which optimizer to choose?

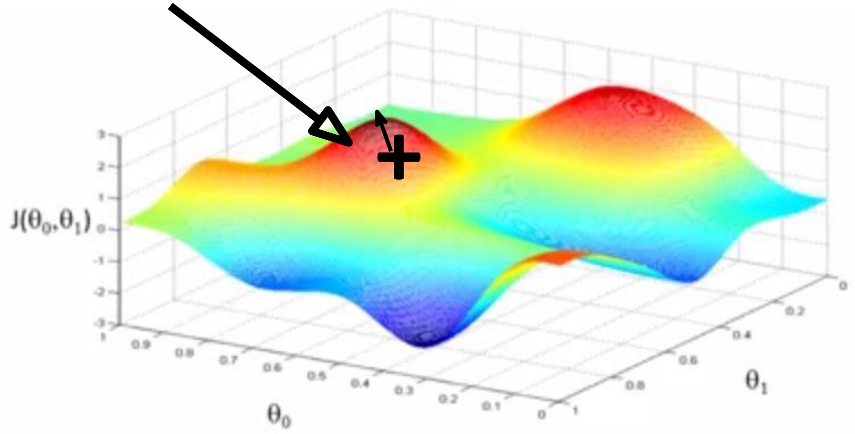
INTRODUCTION

Gradient Descent

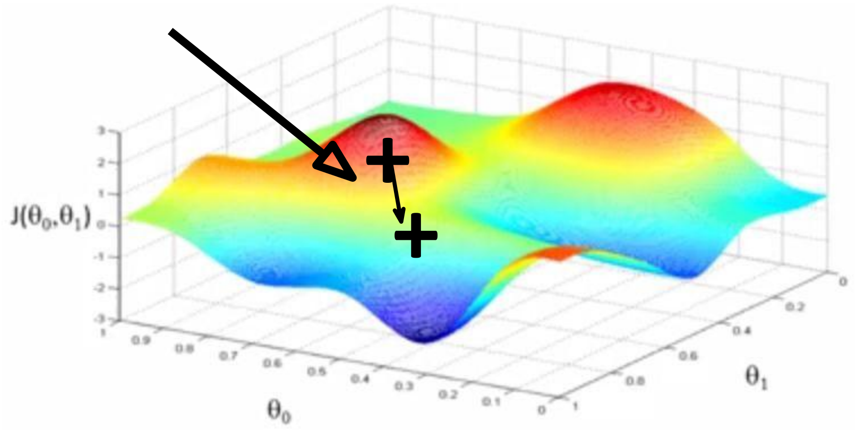


Gradient Descent (梯度下降法)

計算梯度 $\dfrac{\partial L}{\partial \theta}$



沿著 gradient 的反方向走



Initialize θ randomly

For N Epochs

- For each training example (x, y):
  - Compute Loss Gradient: $\dfrac{\partial L}{\partial \theta}$
  - Update θ with update rule:

$$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$$

沿著 gradient 的反方向走

Learning rate

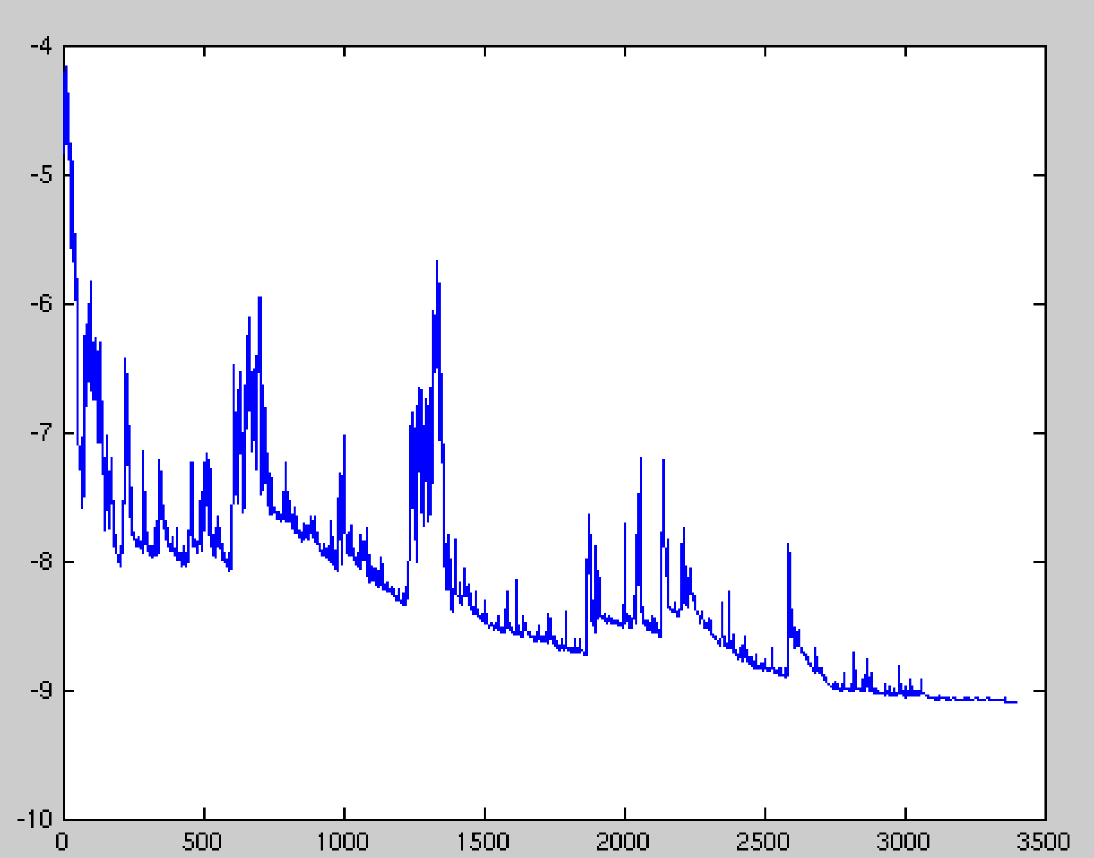
Source: Nick Locascio, intro to deep learning,

Three Variants

- Batch gradient descent
  - Vanilla, original
  - Compute loss over whole training dataset at a time
- Stochastic gradient descent
  - Loss measured per example & label
- Mini-batch gradient descent
  - "Best of both worlds"
  - Loss computed over subset of training dataset

Stochastic Gradient Descent

- SGD performs frequent updates
  - high variance
  - Causes objective function to fluctuate heavily

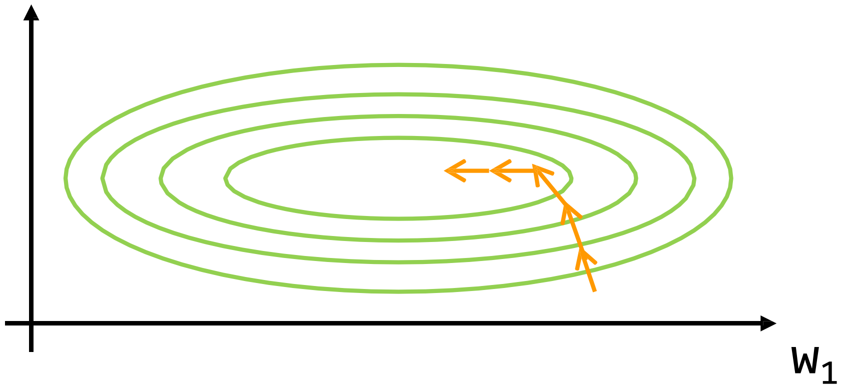


Stochastic Gradient Descent

- Other advantages of SGD vs. batch?
  - enables it to jump to new and potentially better local minima
- Side-effect of SGD's approach to finding minima?
  - Complicates convergence: SGD will keep overshooting
- Solution for this?
  - Slowly decrease learning rate
- SGD shows the same convergence behaviour as batch gradient
  - Almost certainly converging to a local minimum for non-convex optimisation
  - Almost certainly converging to a global minimum for convex optimisation

Challenges of Vanilla mini-batch gradient descent

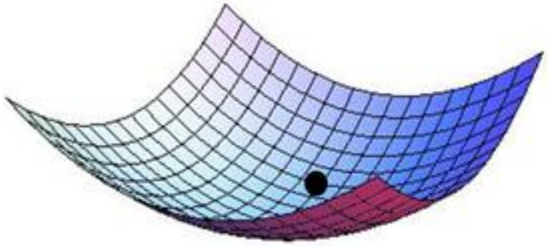
- No guarantee good convergence
- Choosing a proper learning rate can be difficult
  - Too small: very slow convergence
  - Too high:
    - Damages convergence
    - Loss rate can fluctuate or even diverge



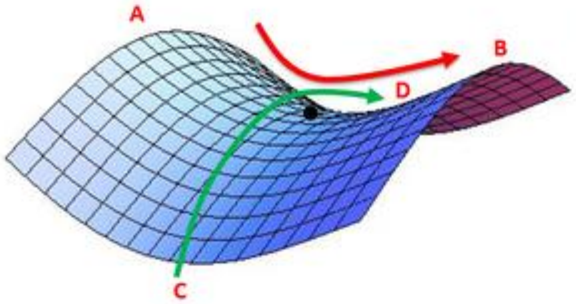
- Learning rate decay should be defined in advance (time or threshold)
  - unable to adapt to a dataset's characteristics
- Same learning rate for all parameters updates
  - Not all features have the same variance
  - Bad for sparse data and features with different frequencies (e.g. need larger update for rarely occurring features)

- Saddle points are a challenge
  - Neural nets tend to have many local minima
  - These tend to meet in saddle points
  - Equates to a plateau of the same error
  - Hard to escape using gradient-based method, as gradient → 0 in all dimensions

$$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$$



Minima- Black dot placed on the PES shows a minimum energy point. Note how a PES resembles a well around the minimum point.

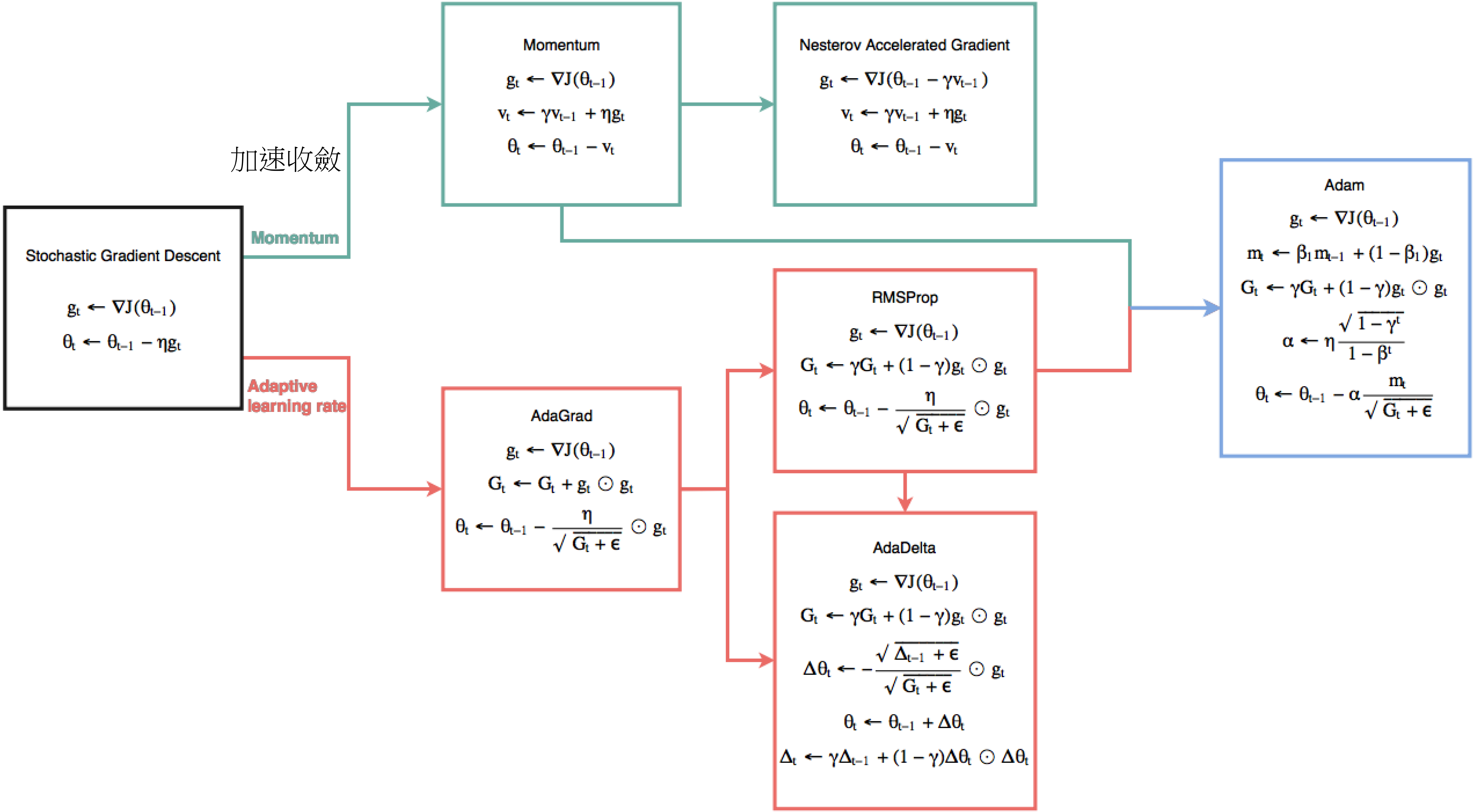


Saddle Point- Black dot placed on the PES shows a minima along path A-B and a maxima along path C-D. It represents a transition state along path C-D which, in this case, is the reaction coordinate.

Solutions: adaptive learning rate strategy + momentum

Source: https://en.wikipedia.org/wiki/Energy_profile_(chemistry)

FROM SGD TO ADAM AND ITS VARIANTS


Stochastic Gradient Descent
$g_t \leftarrow \nabla J(\theta_{t-1})$
$\theta_t \leftarrow \theta_{t-1} - \eta g_t$
加速收斂
Momentum
Adaptive learning rate
Momentum
$g_t \leftarrow \nabla J(\theta_{t-1})$
$v_t \leftarrow \gamma v_{t-1} + \eta g_t$
$\theta_t \leftarrow \theta_{t-1} - v_t$
Nesterov Accelerated Gradient
$g_t \leftarrow \nabla J(\theta_{t-1} - \gamma v_{t-1})$
$v_t \leftarrow \gamma v_{t-1} + \eta g_t$
$\theta_t \leftarrow \theta_{t-1} - v_t$
AdaGrad
$g_t \leftarrow \nabla J(\theta_{t-1})$
$G_t \leftarrow G_t + g_t \odot g_t$
$\theta_t \leftarrow \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
RMSProp
$g_t \leftarrow \nabla J(\theta_{t-1})$
$G_t \leftarrow \gamma G_t + (1-\gamma) g_t \odot g_t$
$\theta_t \leftarrow \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
AdaDelta
$g_t \leftarrow \nabla J(\theta_{t-1})$
$G_t \leftarrow \gamma G_t + (1-\gamma) g_t \odot g_t$
$\Delta\theta_t \leftarrow -\frac{\sqrt{\Delta_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} \odot g_t$
$\theta_t \leftarrow \theta_{t-1} + \Delta\theta_t$
$\Delta_t \leftarrow \gamma \Delta_{t-1} + (1-\gamma) \Delta\theta_t \odot \Delta\theta_t$
Adam
$g_t \leftarrow \nabla J(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
$G_t \leftarrow \gamma G_t + (1-\gamma) g_t \odot g_t$
$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$
$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{m_t}{\sqrt{G_t + \epsilon}}$

Gradient descent optimization

- Momentum based improvement
  - Momentum
  - Nesterov Momentum
- Per-parameter adaptive learning rate
  - Adagrad
  - Adadelta
  - RMSprop
- Combination of above two
  - ADAM

SGD

$$\mathbf{g}_t \leftarrow \nabla J_i(\boldsymbol{\theta}_{t-1})$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t$$

Minibatch SGD

$$\mathbf{g}_t \leftarrow \frac{1}{n}\sum_{i}^{n} \nabla J_i(\boldsymbol{\theta}_{t-1})$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t$$

Momentum

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$

$$\mathbf{v}_t \leftarrow \underline{\gamma \mathbf{v}_{t-1}} + \eta \mathbf{g}_t$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

梯度是過去梯度的加權累積

Keep accumulated momentum to escape local minimum

Nesterov Momentum

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1} - \underline{\gamma \mathbf{v}_{t-1}})$$

$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

Calculate gradient based on predicted next value

Weight 先用預測值假裝更新，再算梯度，做真的更新，會比較準

SGD

$$\mathbf{g}_t \leftarrow \nabla J_i(\boldsymbol{\theta}_{t-1})$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \mathbf{g}_t$$

AdaGrad

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$

$$G_t \leftarrow G_t + \underline{\mathbf{g}_t \odot \mathbf{g}_t}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\underline{\sqrt{G_t + \epsilon}}} \odot \mathbf{g}_t$$

Use $\sum_{i=1}^{t} \mathbf{g}_{i,j}^2$ to judge how often Weight$_i$ have been updated
(Arithmetic average)

Larger Gt-> freq. update
-> smaller learning rate
Small Gt -> seldom update
-> larger learning rate
Note. Gt is monotonically increase
-> learning rate => 0

RMSProp

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$

$$G_t \leftarrow \underline{\gamma G_t + (1-\gamma)}\mathbf{g}_t \odot \mathbf{g}_t$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

Degraded Gt effect (Exponential average)
Avoid learning rate becomes zero

RMSProp

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$

$$G_t \leftarrow \underline{\gamma G_t} + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

AdaDelta

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$

$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$

$$\Delta\boldsymbol{\theta}_t \leftarrow -\frac{\underline{\sqrt{\Delta_{t-1} + \epsilon}}}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

估計learning rate

Automated learning rate instead of fixed one

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t$$

利用之前的步長們估計下一步的步長

$$\underline{\Delta_t \leftarrow \gamma \Delta_{t-1} + (1-\gamma)\Delta\boldsymbol{\theta}_t \odot \Delta\boldsymbol{\theta}_t}$$

可能問題; Gt 是固定時間窗口內的累積，隨著時間窗口的變化，
遇到的數據可能發生巨變，使得Gt可能會時大時小，不是單調變化。這就可能在
訓練後期引起學習率的震盪，導致模型無法收斂

Momentum

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \mathbf{v}_t$$

RMSProp

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$
$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

ADAM

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$ Adam做一階/二階動量估計

$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$$ Momemtum

$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$ RMSProp

$$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$ Bias correction term

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t + \epsilon}}$$

AdamW (Adam with decoupled weight decay)

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$

initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$

repeat

- $t \leftarrow t + 1$
- $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
- $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
- $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
- $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
- $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
- $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
- $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
- $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) +\lambda\boldsymbol{\theta}_{t-1} \right)$

until *stopping criterion is met*

return optimized parameters $\boldsymbol{\theta}_t$

為什麼要「Decoupled Weight Decay」AdamW 介紹

- 避免overfitting，兩種常用的正規化策略
  - L2 regularization in loss function
    - final_loss = loss + wd * all_weights.pow(2).sum() / 2
    - 沒有weight decay, $w = w - lr * \nabla L$
    - 對SGD，$w = w - lr * \nabla L_{final} = w - lr * (\nabla L + wd * w) = w(1 - wd) - lr * \nabla L$
  - Weight decay
    - w = w - lr * w.grad - lr * wd * w = (1-wd) *w - lr * w.grad

$$\mathbf{g}_t \leftarrow \nabla J(\boldsymbol{\theta}_{t-1})$$

$$\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$$

$$G_t \leftarrow \gamma G_t + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t$$

$$\alpha \leftarrow \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{G_t+\epsilon}}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon} \underbrace{-\alpha \cdot \mathrm{wd} \cdot \theta_t}_{\text{decoupled weight decay}}$$

$\nabla L_{final} = (\nabla L + \lambda * \theta)$

用 L2 $regularization\ g_t = g_{t-1} + \lambda\theta$

用 weight decay $\theta = \theta - \alpha \nabla L_{final} = (1-\lambda)\,\theta - \alpha\nabla L$

對 **SGD** 這等價於 **weight decay**（每步把 θ乘上 1−λ）。

且對 **Adam**，因為會除以 sqrt(Gt) 也被自適應縮放，導致每一維的衰減「不再等比例」，正則化力道被梯度統計**扭曲**，使得「L2≠weight decay」

AdamW 的關鍵：把衰減從梯度路徑中**拿出來獨立處理**這樣衰減量 與 v^t**無關**，回到「真正的 weight decay」（各維等比例收縮），正則效果更可控、與學習率/排程的互動也更直觀。

。

Lion (Evolved Sign Momentum)

- Algorithm searched optimizer
- Compared to Adam
  - Lower memory usage (only keep track of the momentum)
- Different from adaptive optimizers,
  - its update has the same magnitude for each parameter calculated through the sign operation
- Performance
  - Up to 2% increase for ViT
  - Reduce training time by up to 2.3X for diffusion models

$$\text{Lion} := \begin{cases} \boldsymbol{u}_t = \text{sign}\left(\beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\, \boldsymbol{g}_t\right) \\ \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t(\boldsymbol{u}_t + \lambda_t \boldsymbol{\theta}_{t-1}) \\ \boldsymbol{m}_t = \beta_2 \boldsymbol{m}_{t-1} + (1-\beta_2)\, \boldsymbol{g}_t \end{cases}$$

Momentum 的更新放在weight更新之後

β1=0.9,β=0.99

Lion的更新量u每個分量的絕對值都是1，這通常比AdamW要大，所以學習率要縮小10倍以上

$$\text{AdamW} := \begin{cases} \boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\, \boldsymbol{g}_t \\ \boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\, \boldsymbol{g}_t^2 \\ \hat{\boldsymbol{m}}_t = \boldsymbol{m}_t \left/ \left(1-\beta_1^t\right)\right. \\ \hat{\boldsymbol{v}}_t = \boldsymbol{v}_t \left/ \left(1-\beta_2^t\right)\right. \\ \boldsymbol{u}_t = \hat{\boldsymbol{m}}_t \left/ \left(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon\right)\right. \\ \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t(\boldsymbol{u}_t + \lambda_t \boldsymbol{\theta}_{t-1}) \end{cases}$$

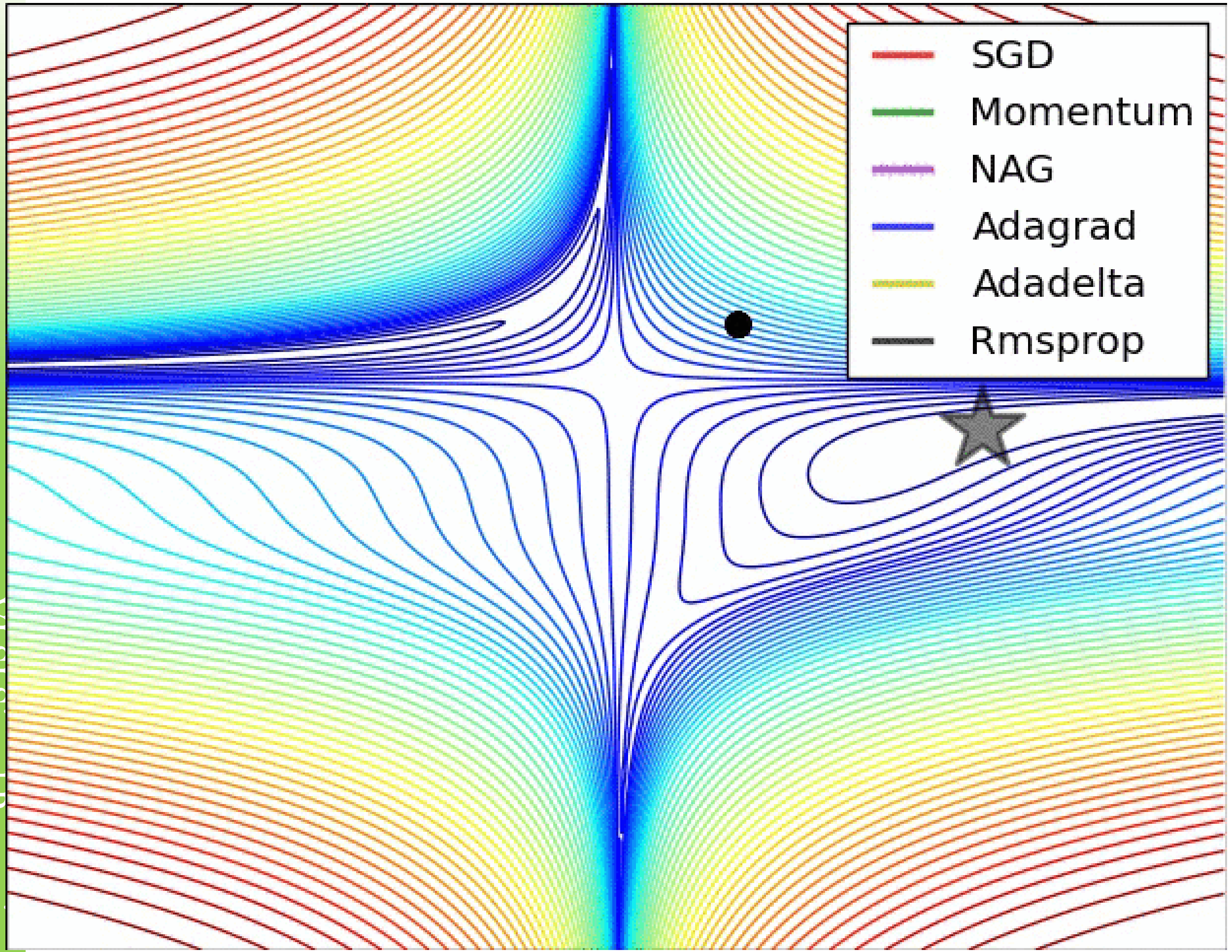
Table 7: The performance of various optimizers to train ViT-S/16 and ViT-B/16 on ImageNet (with RandAug and Mixup). Lion is still the best performing one, and there is no clear winner amongst the baselines.

Model	Task	AdamW	RAdam	NAdam	AdaBelief	AMSGrad	$\text{Ablation}_{0.9}$	$\text{Ablation}_{0.99}$	Lion
ViT-S/16	ImageNet	78.89	78.59	78.91	78.71	79.01	78.23	78.19	**79.46**
	ReaL	84.61	84.47	84.62	84.56	85.01	84.28	84.17	**85.25**
	V2	66.73	66.39	66.02	66.35	66.82	66.13	65.96	**67.68**
ViT-B/16	ImageNet	80.12	80.26	80.32	80.29	79.85	79.54	79.90	**80.77**
	ReaL	85.46	85.45	85.44	85.48	85.16	85.10	85.36	**86.15**
	V2	68.14	67.76	68.46	68.19	68.48	68.07	68.20	**69.19**

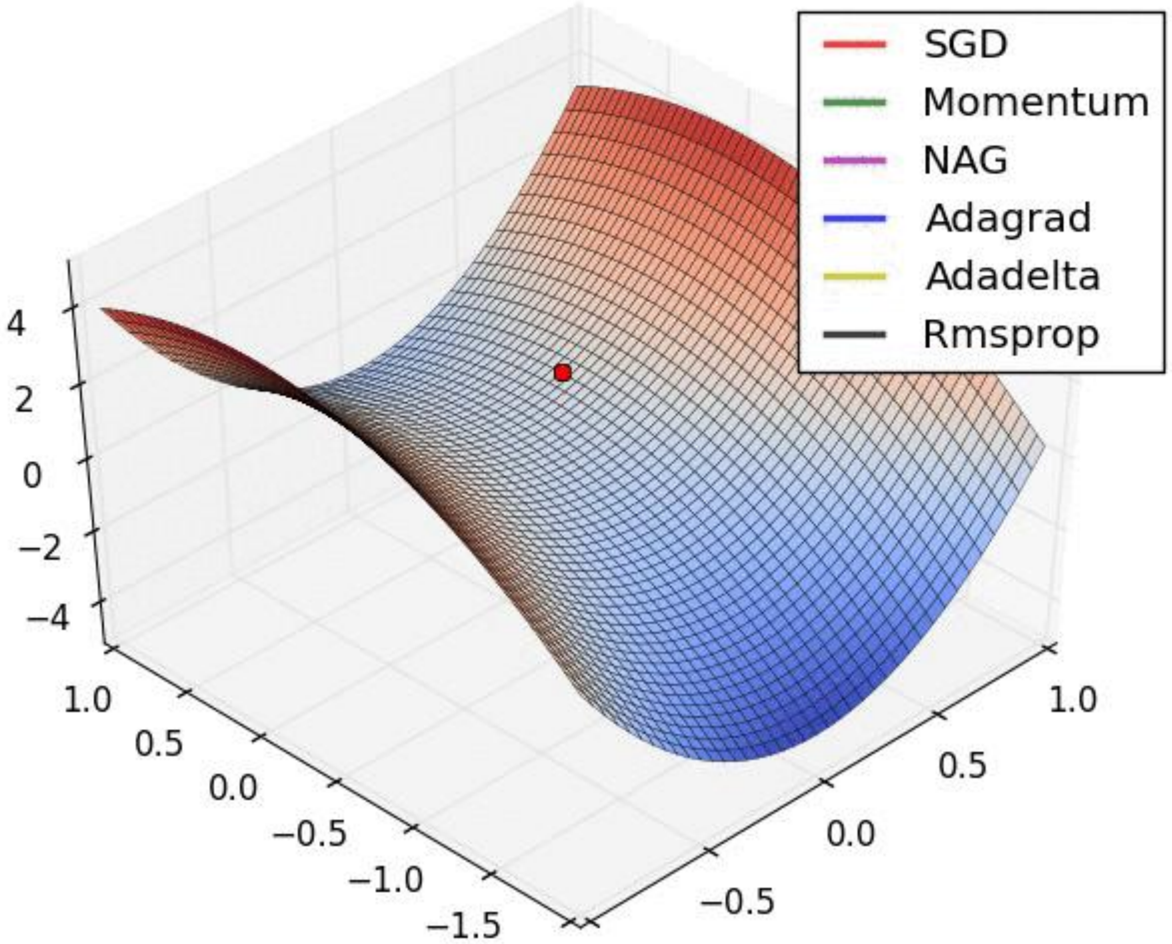
Table 5: One-shot evaluation averaged over three NLG and 21 NLU tasks. The results of GPT-3 (Brown et al., 2020) and PaLM (Chowdhery et al., 2022) are included for reference. The LLMs trained by Lion have better in-context learning ability. See Table 11 (in the Appendix) for detailed results on all tasks.

Task	1.1B		2.1B		7.5B		6.7B	8B
	Adafactor	Lion	Adafactor	Lion	Adafactor	Lion	GPT-3	PaLM
#Tokens	300B						300B	780B
Avg NLG	11.1	**12.1**	15.6	**16.5**	24.1	**24.7**	23.1	23.9
Avg NLU	53.2	**53.9**	56.8	**57.4**	61.3	**61.7**	58.5	59.4

Visualization



SGD optimization on loss surface contours



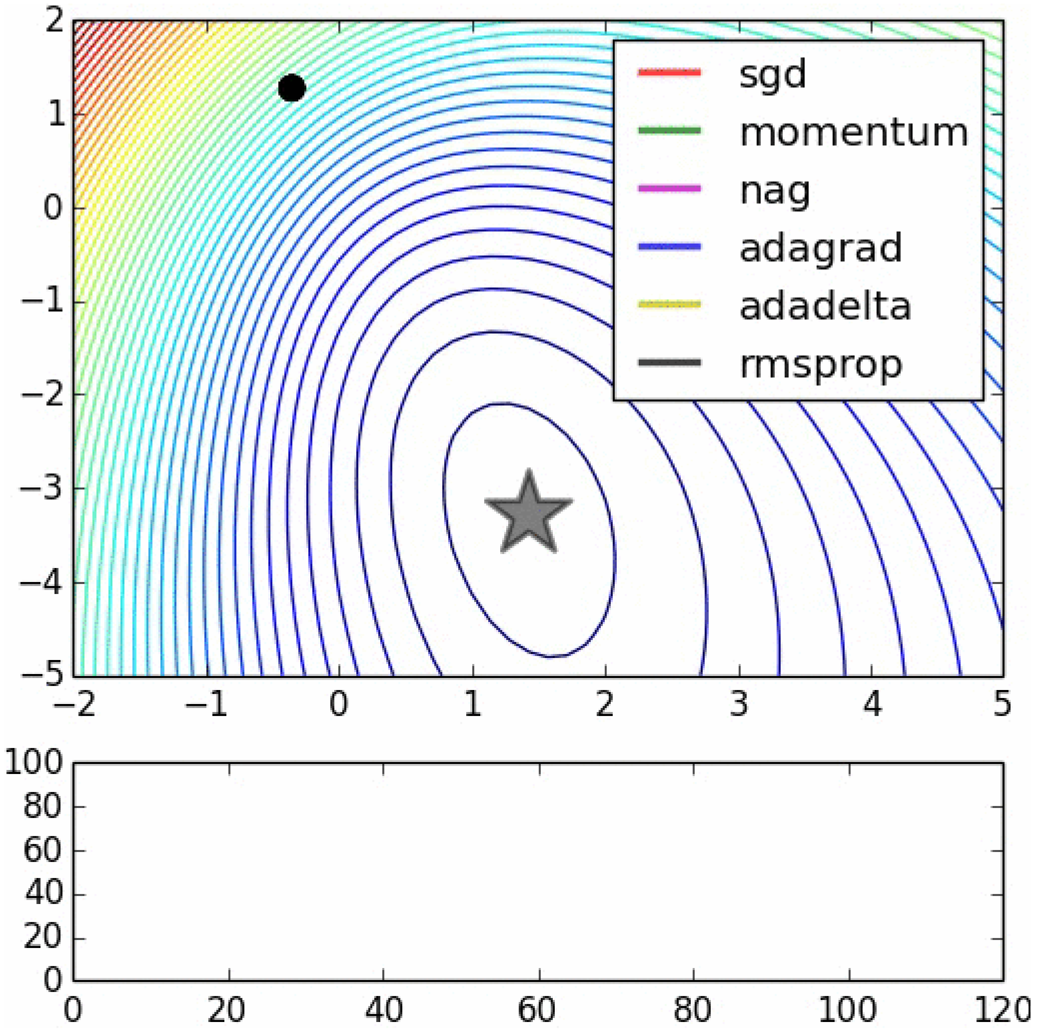
SGD optimization on saddle point

- Behaviour on the countours of a loss surface
  - Adagrad, Adadelta, and RMSprop almost immediately head off in the right direction and converge similarly fast,
  - Momentum and NAG are led off-track, evoking the image of a ball rolling down the hill.
  - NAG, however, is quickly able to correct its course due to its increased responsiveness by looking ahead and heads to the minimum
  - Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill

Behaviour on the saddle points

- SGD, Momentum, and NAG find it difficulty to break symmetry, although the two latter eventually manage to escape the saddle point
- Adagrad, RMSprop, and Adadelta quickly head down the negative slope.

Learning rate v.s. Gradient Optimization



Appendix

IMPROVE ADAM

WE WILL ONLY GO THROUGH PART OF THIS

Improvement

- +weight decay or L2 regularization
  - ADAMW: use weight decay
- Initial warmup for training stability
  - RADAM: rectified ADAM, 避免起始model 不穩定，根據方差分散度，動態地打開或者關閉自適應學習率，並且提供了一種不需要可調參數學習率預熱的方法。
    - 對起始warmup 有用，後期類似ADAM
  - AdaMod: ADAM + long term memory of learning rate (long term average of the adaptive learning rates)
- 減少超參數調整
  - Lookahead optimizer: fast and slow weight
  - Ranger: RADAM + Lookahead
- Better convergence
  - Gradient centralization: zero mean gradient
  - diffGrad: 控制收斂時梯度，避免overshooting

AdamW: Adam + weigh decay regularization

- Weight decay or L2 regularization?

```
final_loss = loss + wd * all_weights.pow(2).sum() / 2

w = w - lr * w.grad - lr * wd * w
```

  - These two are equivalent for vanilla SGD,
  - But become different with momentum

```
//L2 regularization update
moving_avg = alpha * moving_avg + (1-alpha) * (w.grad + wd*w)
//weight decay's update
moving_avg = alpha * moving_avg + (1-alpha) * w.grad
w = w - lr * moving_avg - lr * wd * w
```

  - Suggest to use weight decay version (ADAMW)

Adafactor (Adaptive Learning Rates with Factorization)

針對NLP (LLM) 應用，省記憶體用量

$$\begin{cases} g_t = \nabla_\theta L(\theta_{t-1}) \\ \boxed{m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t} \\ v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \\ \hat{m}_t = m_t \,/ (1-\beta_1^t) \\ \hat{v}_t = v_t \,/ (1-\beta_2^t) \\ \theta_t = \theta_{t-1} - \alpha_t \hat{m}_t \,/ \left(\sqrt{\hat{v}_t} + \epsilon\right) \end{cases}$$

再加入正確的weight decay 和layer adaptive

對NLP， adaptive learning 比較重要，拋棄動量=> 類RMSprop，變數直接減少了一半

$$\begin{cases} g_t = \nabla_\theta L(\theta_{t-1}) \\ v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \\ \hat{v}_t = v_t \,/ (1-\beta_2^t) \\ \theta_t = \theta_{t-1} - \alpha_t g_t \,/ \sqrt{\hat{v}_t + \epsilon} \end{cases}$$

$$\begin{cases} g_{i,j;t} = \nabla_\theta L(\theta_{i,j;t-1}) \\ \hat{\beta}_{2,t} = 1 - t^{-c} \\ v_{i;t}^{(r)} = \hat{\beta}_{2,t} v_{t-1;i}^{(r)} + \left(1-\hat{\beta}_{2,t}\right) \sum_j \left(g_{i,j;t}^2 + \epsilon_1\right) \\ v_{j;t}^{(c)} = \hat{\beta}_{2,t} v_{t-1;j}^{(c)} + \left(1-\hat{\beta}_{2,t}\right) \sum_i \left(g_{i,j;t}^2 + \epsilon_1\right) \\ \hat{v}_{i,j;t} = v_{i;t}^{(r)} v_{j;t}^{(c)} \Big/ \sum_j v_{j;t}^{(c)} \\ \boxed{\begin{array}{l} u_t = g_t \,/ \sqrt{\hat{v}_t} \\ \hat{u}_t = u_t \,/ \max\left(1, RMS(u_t) / d\right) \times \max\left(\epsilon_2, RMS(\theta_{t-1})\right) \end{array}} \\ \theta_t = \theta_{t-1} - \alpha_t \hat{u}_t \end{cases}$$

把變數v的參數量再壓縮。用矩陣的低秩分解

$$\begin{cases} g_{i,j;t} = \nabla_\theta L(\theta_{i,j;t-1}) \\ \boxed{\begin{array}{l} v_{i;t}^{(r)} = \beta_2 v_{t-1;i}^{(r)} + (1-\beta_2) \sum_j \left(g_{i,j;t}^2 + \epsilon\right) \\ v_{j;t}^{(c)} = \beta_2 v_{t-1;j}^{(c)} + (1-\beta_2) \sum_i \left(g_{i,j;t}^2 + \epsilon\right) \\ v_{i,j;t} = v_{i;t}^{(r)} v_{j;t}^{(c)} \Big/ \sum_j v_{j;t}^{(c)} \end{array}} \\ \hat{v}_t = v_t \,/ (1-\beta_2^t) \\ \theta_t = \theta_{t-1} - \alpha_t g_t \,/ \sqrt{\hat{v}_t} \end{cases}$$

https://arxiv.org/pdf/1804.04235

Lion: EvoLved Sign Momentum

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
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ ← Lion 省了二階
bias correction
$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ← Lion 省了二階
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

與 AdamW 和各種自適應優化器需要同時保存一階和二階矩相比，Lion 只需要動量，將**額外的記憶體佔用減半**。這在訓練大型模型和/或大批量時很有用。小batch_size（小於64）的時候效果不如AdamW

https://arxiv.org/abs/2302.06675v4

Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training

Algorithm 1 Hutchinson(θ)

Input: parameter θ.
Compute mini-batch loss $L(\theta)$.
Draw u from $\mathcal{N}(0, \mathrm{I}_d)$.
return $u \odot \nabla(\langle \nabla L(\theta), u \rangle)$.

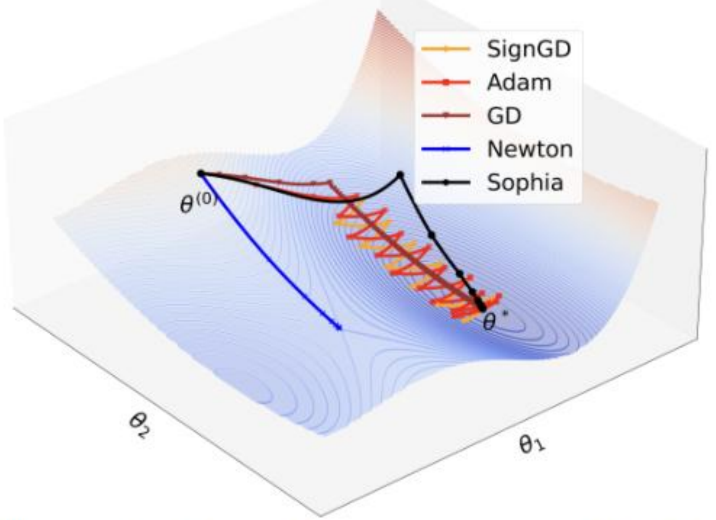


Figure 2: The motivating toy example. $\theta_{[1]}$ is the sharp dimension and $\theta_{[2]}$ is the flat dimension. GD's learning rate is limited by the sharpness in θ_1, and makes slow progress along $\theta_{[2]}$. Adam and SignGD bounce along $\theta_{[1]}$ while making slow progress along $\theta_{[2]}$. Vanilla Newton's method converges to a saddle point. Sophia makes fast progress in both dimensions and converges to the minimum with a few steps.

https://arxiv.org/abs/2305.14342v4

Algorithm 2 Gauss-Newton-Bartlett(θ)

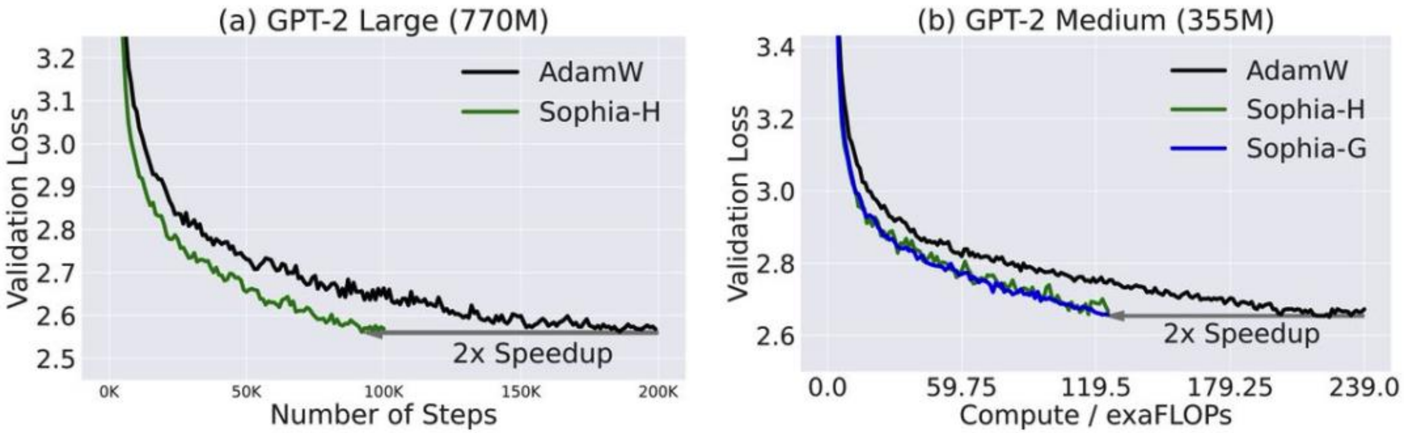
Input: parameter θ.
Draw a mini-batch of input $\{x_b\}_{b=1}^{B}$.
Compute logits on the mini-batch: $\{f(\theta, x_b)\}_{b=1}^{B}$.
Sample $\hat{y}_b \sim \mathrm{softmax}(f(\theta, x_b)), \forall b \in [B]$.
Calculate $\hat{g} = \nabla(1/B \sum \ell(f(\theta, x_b), \hat{y}_b))$.
return $B \cdot \hat{g} \odot \hat{g}$.

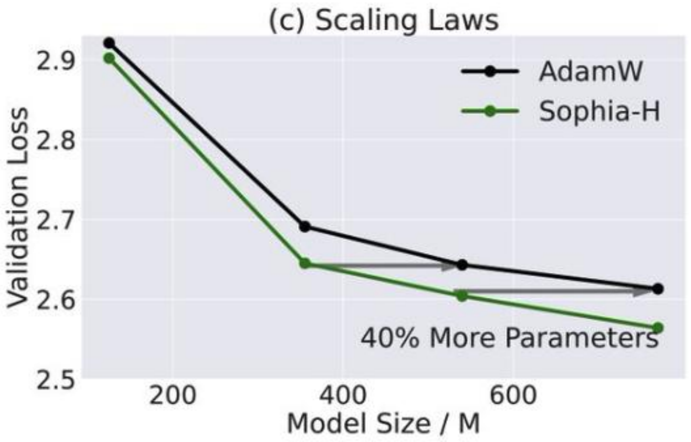
Algorithm 3 Sophia

Input: θ_1, learning rate $\{\eta_t\}_{t=1}^{T}$, hyperparameters $\lambda, \gamma, \beta_1, \beta_2, \epsilon$, and estimator choice Estimator $\in$ {Hutchinson, Gauss-Newton-Bartlett}
Set $m_0 = 0$, $v_0 = 0$, $h_{1-k} = 0$
for $t = 1$ **to** T **do**
  Compute minibach loss $L_t(\theta_t)$.
  Compute $g_t = \nabla L_t(\theta_t)$.
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  if $t \bmod k = 1$ **then**
    Compute $\hat{h}_t = \mathrm{Estimator}(\theta_t)$.
    $h_t = \beta_2 h_{t-k} + (1 - \beta_2)\hat{h}_t$
  else
    $h_t = h_{t-1}$
  $\theta_t = \theta_t - \eta_t \lambda \theta_t$ (weight decay)
  $\theta_{t+1} = \theta_t - \eta_t \cdot \mathrm{clip}(m_t / \max\{\gamma \cdot h_t, \epsilon\}, 1)$

Sophia 優化器使用隨機估計值作為Hessian矩陣對角線的pre-conditioner，並採用剪切（clipping）機制來控制最壞情況下的參數大小更新。在像GPT-2這樣的預訓練語言模型上，Sophia 與Adam 相比，在減少了50% step 數量的情況下實現了相同的驗證預訓練損失。

Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training





Gauss-Newton-Bartlett's diag Hess estimator

1. $\widehat{L}(\theta) \triangleq$ CE loss on labels sampled from θ.
2. diag Hessian estimate $\propto \nabla\widehat{L}(\theta) \odot \nabla\widehat{L}(\theta)$.

Sophia

1. Estimate Hess every $k = 10$ steps
2. $\theta_{t+1} = \theta_t - \eta_t \cdot \text{clip}\left(\frac{\text{EMA of gradients}}{\text{EMA of diag Hessian estimate}}, \rho\right)$

https://arxiv.org/abs/2305.14342v4

Optim	類型	額外狀態	特色	何時試
AdamW	一階自適應 + **decoupled wd**	m,v	預設穩、調參直覺	Transformer/LLM/ViT 基線 (arXiv)
Adafactor	省記憶體（行列分解二階）	低	近 Adam 效果、顯存省	大模型/顯存緊（T5 等）(arXiv)
Lion	符號+動量（無 v）	m	省記憶體；大 batch 友好	ViT/對比學習/微調可試 (arXiv)
Sophia	輕量二階（Hessian 對角估計+裁剪）	m, diag-H	步數減少報告（LLM）	預訓練省步數場景 (arXiv)

LEARNING RATE VS BATCH SIZE

Learning rate vs Batch Size

- 增加batchsize為原來的N倍時，
  - 要保證經過同樣的樣本後更新的權重相等，按照線性縮放規則，學習率應該增加為原來的N倍[5]。
  - 但是如果要保證權重的方差不變，則學習率應該增加為原來的sqrt(N)倍[7]

Goyal P, Dollar P, Girshick R B, et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.[J]. arXiv: Computer Vision and Pattern Recognition, 2017.
Hoffer E, Hubara I, Soudry D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks[C]//Advances in Neural Information Processing Systems. 2017: 1731-1741.