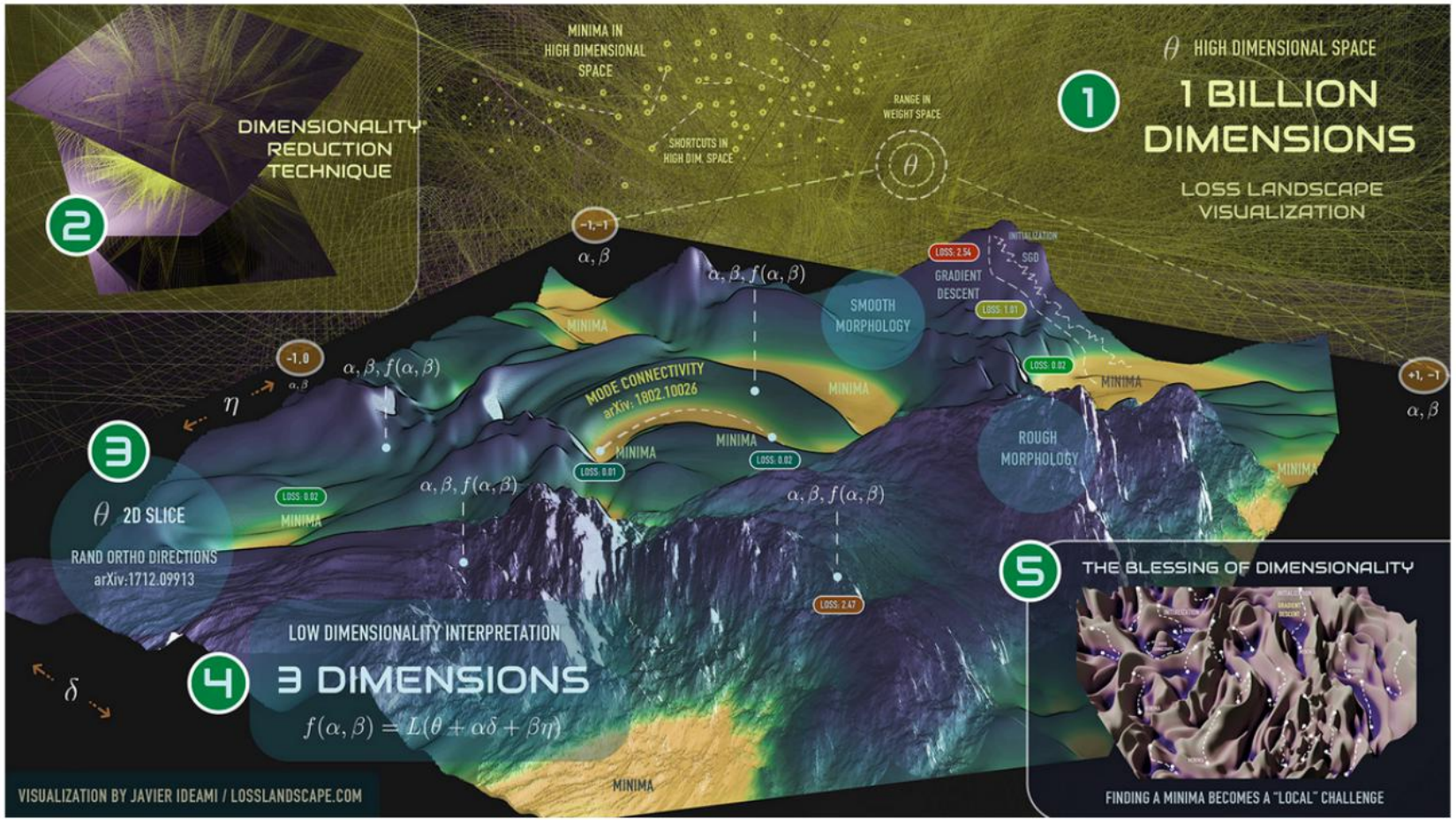
AI VIETNAM
All-in-One Course
(TA Session)

EXERCISE - Optimization Trong Machine Learning



13:53:24

Content

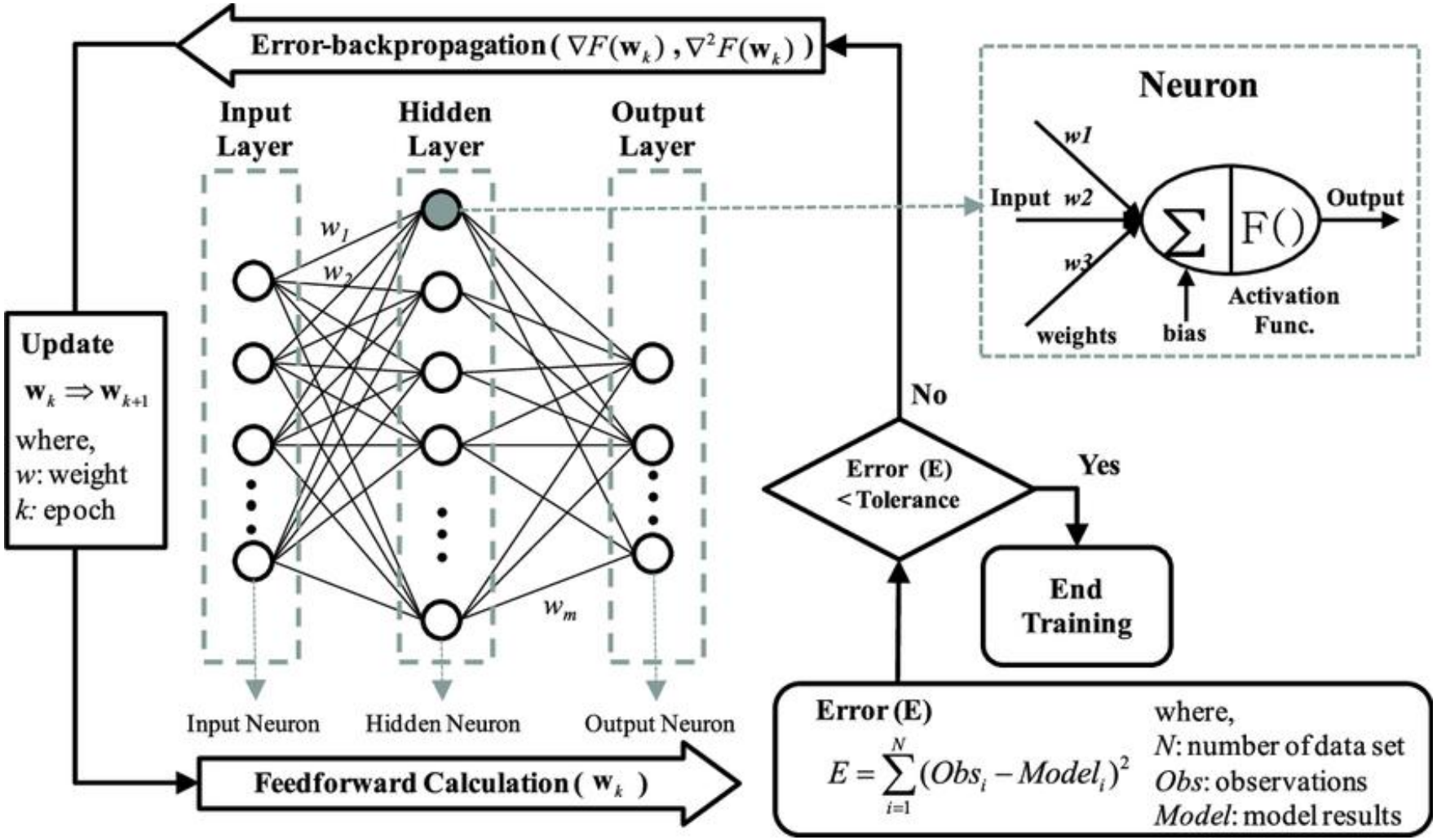
- **Giới Thiệu Optimization Trong Machine Learning**
  - Giới Thiệu Gradient-based Optimization Trong Machine Learning
  - Giới Thiệu Sharpness-Aware Minimization
- **Optimizing Functions of Two Varibales**
  - Giới thiệu vấn đề
  - Exercise1: Gradient Descent
  - Exercise2: Gradient Descent + Momentum
  - Exercise3: RMSProp
  - Exercise4: Adam
- **Vanishing Problem (Optional)**
  - GD, GD + Momentum, RMSProp, và Adam
- **Other Research Papers**

Content

- **Giới Thiệu Optimization Trong Machine Learning**
  - Giới Thiệu Gradient-based Optimization Trong Machine Learning
  - Giới Thiệu Sharpness-Aware Minimization
- **Optimizing Functions of Two Varibales**
  - Giới thiệu vấn đề
  - Exercise1: Gradient Descent
  - Exercise2: Gradient Descent + Momentum
  - Exercise3: RMSProp
  - Exercise4: Adam
- **Vanishing Problem (Optional)**
  - GD, GD + Momentum, RMSProp, và Adam
- **Other Research Papers**

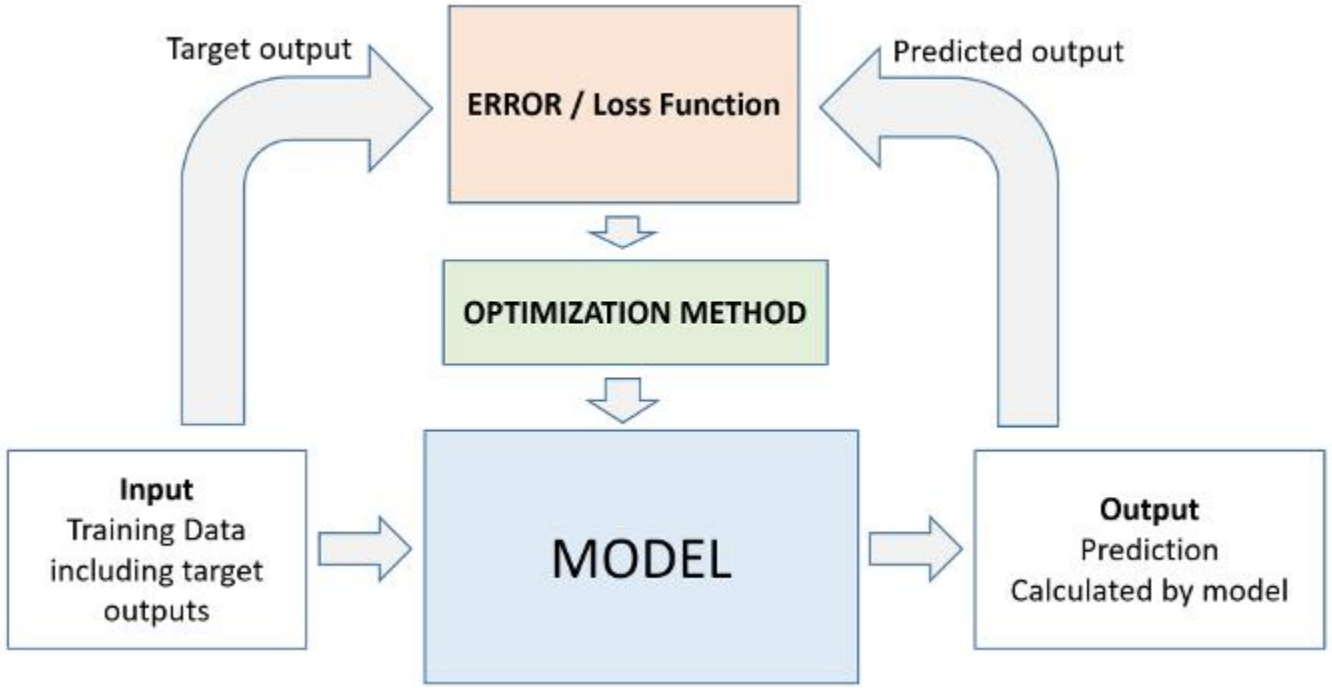
Giới Thiệu Optimization Trong Machine Learning

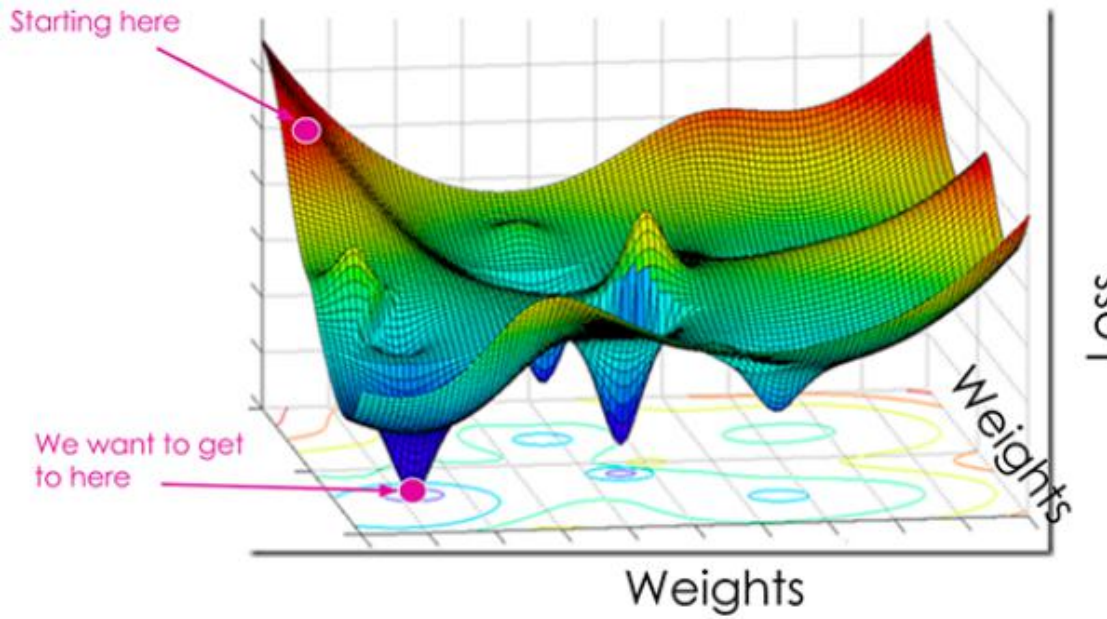
- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**



Giới Thiệu Optimization Trong Machine Learning

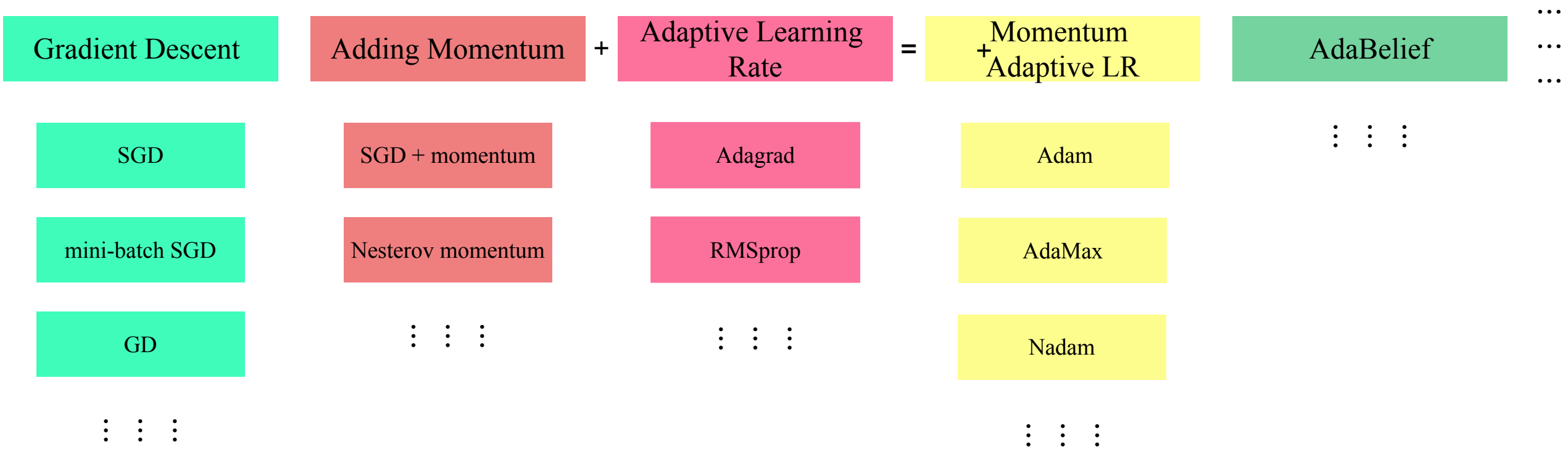
- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**





Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**



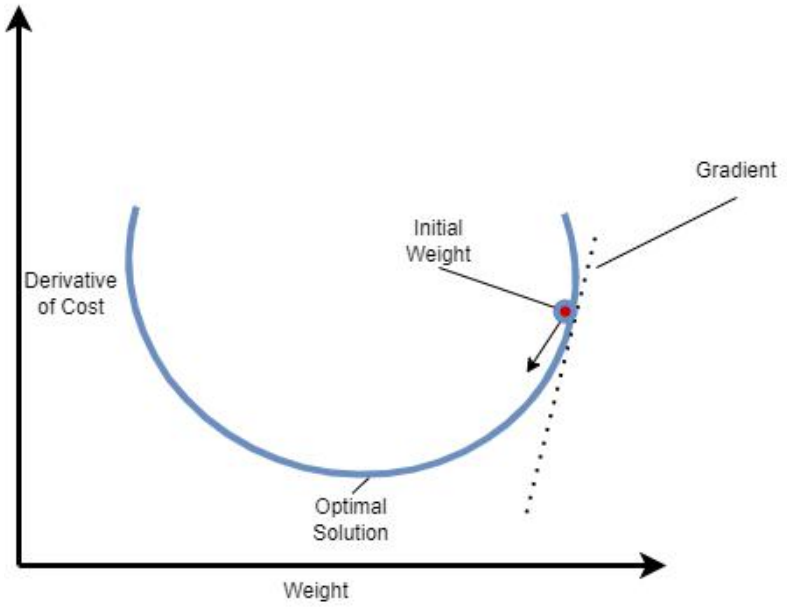
Giới Thiệu Optimization Trong Machine Learning

Gradient Descent

SGD

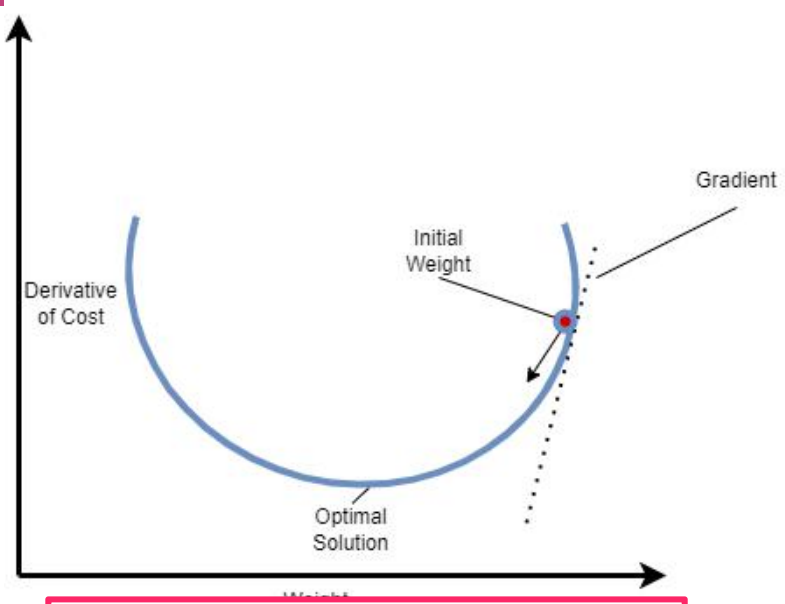
mini-batch SGD

GD



- A Hypothesis: $h_\omega(x) = \omega_0 + \omega_1 x$
- Parameters that need to be updated: ω_0, ω_1
- Cost Function: $J(\omega_0, \omega_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\omega(x^{(i)}) - y^{(i)})^2$
- Aim: Minimization of $J(\omega_0, \omega_1)$ with respect to ω_0, ω_1

Giới Thiệu Optimization Trong Machine Learning



$$w_{i+1} = w_i - a \cdot \nabla_{w_i} J(x^i, y^i; w_i)$$

Algorithm 2: Pseudo-code for SGD

Function `SGD`**:**
- Set epsilon as the limit of convergence
- **for** $i = 1, \ldots, m$ **do**
  - **for** $j = 0, \ldots, n$ **do**
    - **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
      - $\omega_{j+1} := \omega_j - \alpha \cdot (h_\omega(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$;
    - **end**
  - **end**
- **end**

SGD

$$w_{i+1} = w_i - a \cdot \nabla_{w_i} J(w_i)$$

Algorithm 1: Pseudo-code for GD

Function `GD`**:**
- Set epsilon as the limit of convergence
- **for** $j = 1$ *and* $j = 0$ **do**
  - **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
    - $\omega_{j+1} := \omega_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\omega(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$;
  - **end**
- **end**

GD

$$w_{i+1} = w_i - a \cdot \nabla_{w_i} J(x^{i:i+b}, y^{i:i+b}; w_i)$$

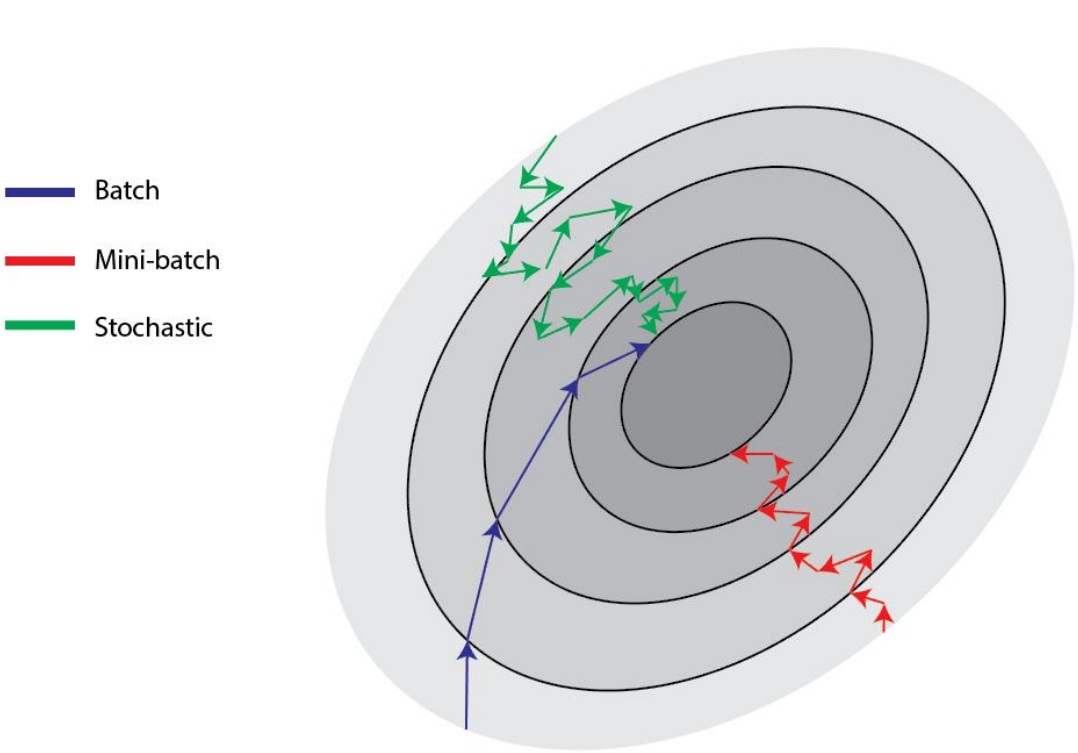
Algorithm 3: Pseudo-code for Mini Batch Gradient Desecent

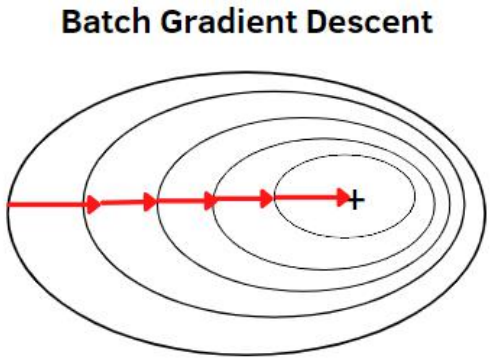
Function `SGD`**:**
- Set epsilon as the limit of convergence
- **for** $i = 1, \ldots, b$ **do**
  - **for** $j = 0, \ldots, n$ **do**
    - **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
      - $\omega_{j+1} := \omega_j - \alpha \cdot \frac{1}{b} \sum_{k=i}^{i+b-1} (h_\omega(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$;
    - **end**
  - **end**
- **end**

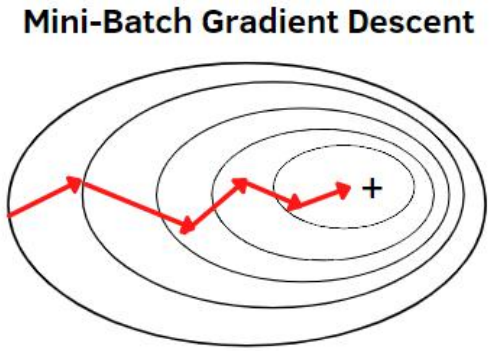
mini-batch SGD

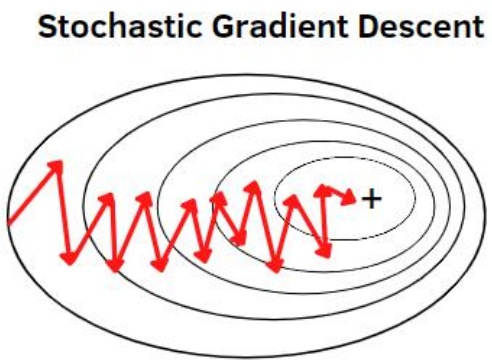
Giới Thiệu Optimization Trong Machine Learning

Gradient Descent









Giới Thiệu Optimization Trong Machine Learning

GD

Advantages

1. Fewer model updates mean that this variant of the steepest descent method is more computationally efficient than the stochastic gradient descent method.
2. Reducing the update frequency provides a more stable error gradient and a more stable convergence for some problems.
3. Separating forecast error calculations and model updates provides a parallel processing-based algorithm implementation.

Disadvantages

1. A more stable error gradient can cause the model to prematurely converge to a suboptimal set of parameters.
2. End-of-training epoch updates require the additional complexity of accumulating prediction errors across all training examples.
3. The batch gradient descent method typically requires the entire training dataset in memory and is implemented for use in the algorithm.
4. Large datasets can result in very slow model updates or training speeds.
5. Slow and require more computational power.

Giới Thiệu Optimization Trong Machine Learning

SGD

Advantages

1. You can instantly see your model's performance and improvement rates with frequent updates.
2. This variant of the steepest descent method is probably the easiest to understand and implement, especially for beginners.
3. Increasing the frequency of model updates will allow you to learn more about some issues faster.
4. The noisy update process allows the model to avoid local minima (e.g., premature convergence).
5. Faster and require less computational power.
6. Suitable for the larger dataset.

Disadvantages

1. Frequent model updates are more computationally intensive than other steepest descent configurations, and it takes considerable time to train the model with large datasets.
2. Frequent updates can result in noisy gradient signals. This can result in model parameters and cause errors to fly around (more variance across the training epoch).
3. A noisy learning process along the error gradient can also make it difficult for the algorithm to commit to the model's minimum error.

Giới Thiệu Optimization Trong Machine Learning

mini-batch SGD

Advantages

1. The model is updated more frequently than the stack gradient descent method, allowing for more robust convergence and avoiding local minima.
2. Batch updates provide a more computationally efficient process than stochastic gradient descent.
3. Batch processing allows for both the efficiency of not having all the training data in memory and implementing the algorithm.

Disadvantages

1. Mini-batch requires additional hyperparameters “mini-batch size” to be set for the learning algorithm.
2. Error information should be accumulated over a mini-batch of training samples, such as batch gradient descent.
3. It will generate complex functions.

Giới Thiệu Optimization Trong Machine Learning

- **Concerns on SGD**

1. If the loss function changes quickly in one direction and slowly in another, it may result in a high oscillation of gradients making the training progress very slow.

2. If the loss function has a local minimum or a saddle point, it is very possible that SGD will be stuck there without being able to “jump out” and proceed in finding a better minimum. This happens because the gradient becomes zero so there is no update in the weight whatsoever.

3. The gradients are still noisy because we estimate them based only on a small sample of our dataset. The noisy updates might not correlate well with the true direction of the loss function.

4. Choosing a good loss function is tricky and requires time-consuming experimentation with different hyperparameters.

5. The same learning rate is applied to all of our parameters, which can become problematic for features with different frequencies or significance.

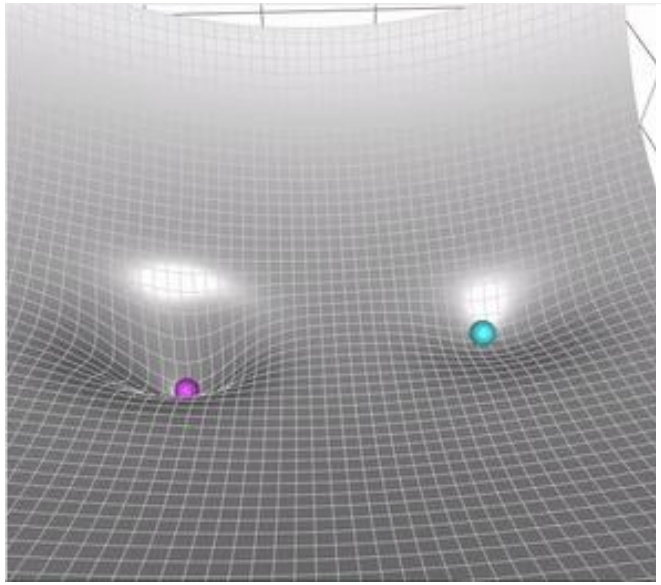
Giới Thiệu Optimization Trong Machine Learning

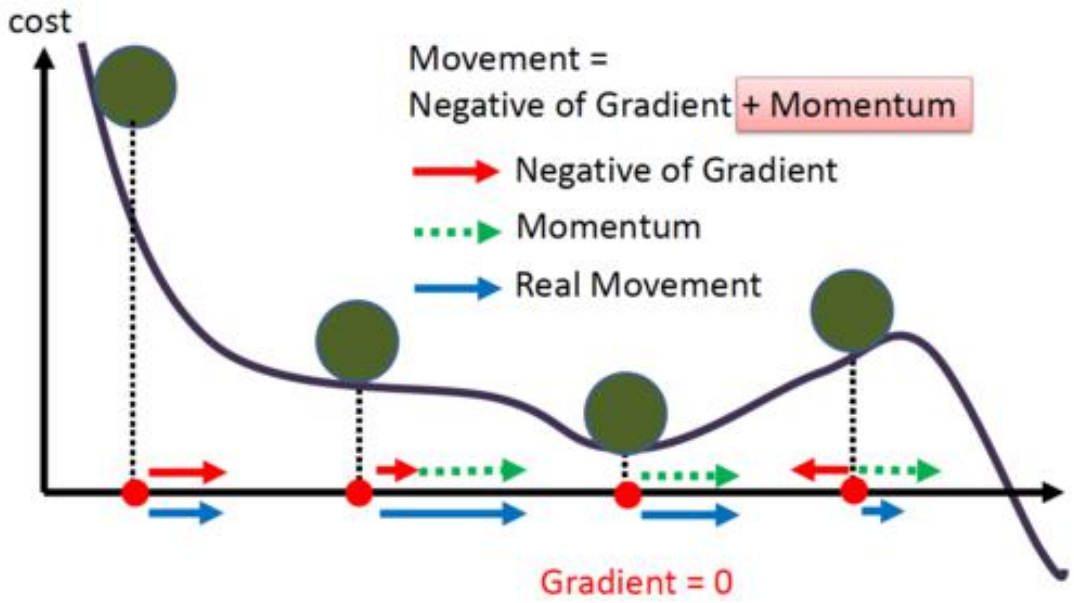
- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Adding Momentum

SGD + momentum

$$v_{dW} = \beta v_{dW} + (1-\beta)dW$$
$$v_{db} = \beta v_{db} + (1-\beta)db$$
$$W = W - \alpha v_{dW},\ b = b - \alpha v_{db}$$





• Giới Thiệu Gradient-based Optimization Trong Machine Learning

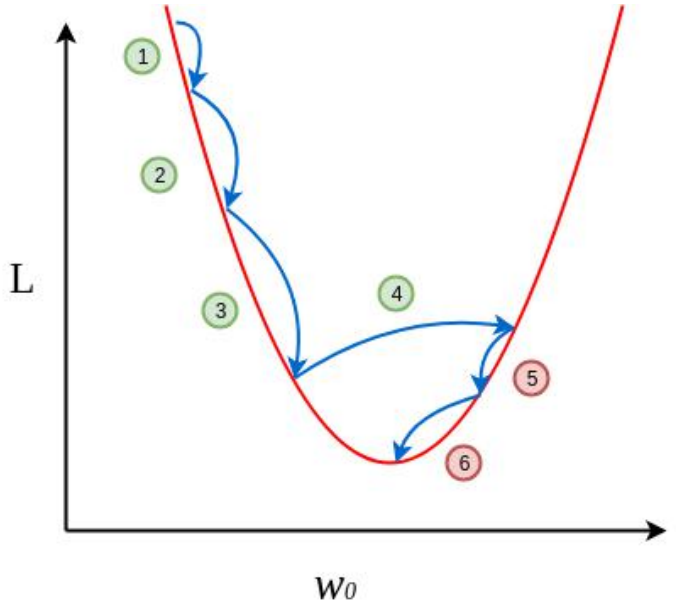
Adding Momentum

SGD + momentum

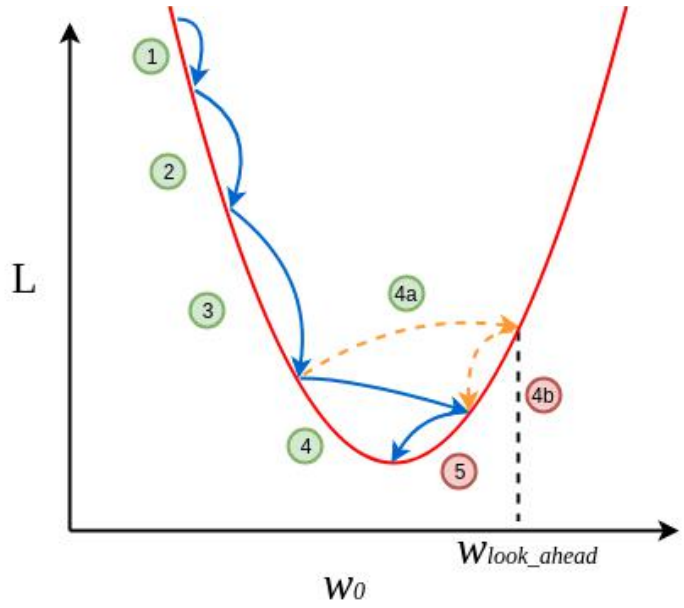
Nesterov momentum

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$



(a) Momentum-Based Gradient Descent



(b) Nesterov Accelerated Gradient Descent

$$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)}$$

$$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$$

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

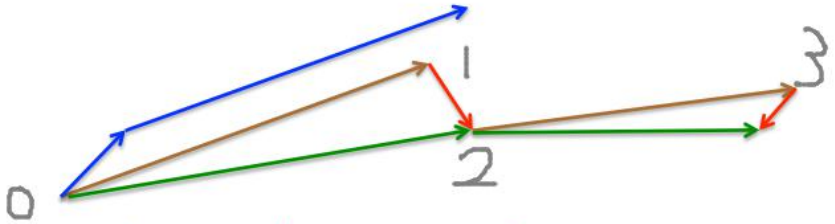
Adding Momentum

SGD + momentum

Nesterov momentum

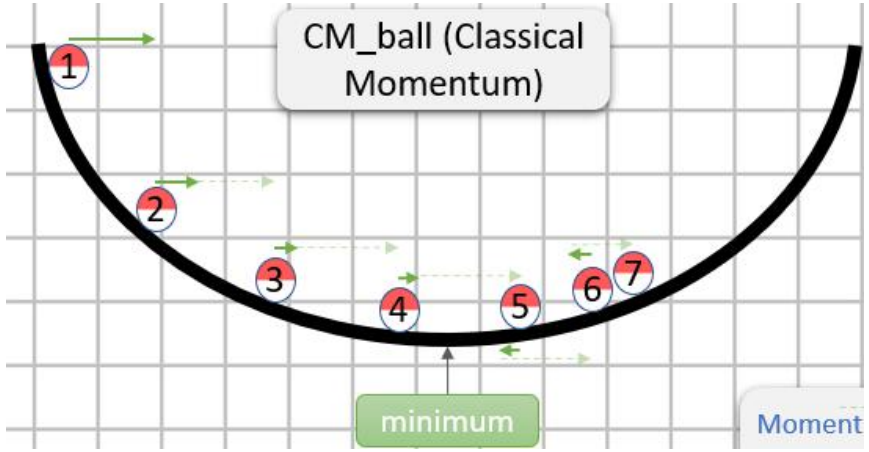
A picture of the Nesterov method

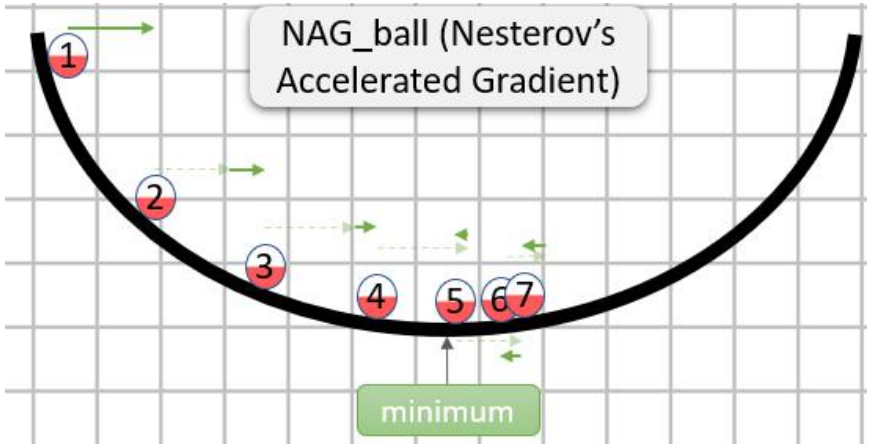
- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction.



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum





Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Adaptive Learning Rate

Adagrad

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

RMSprop

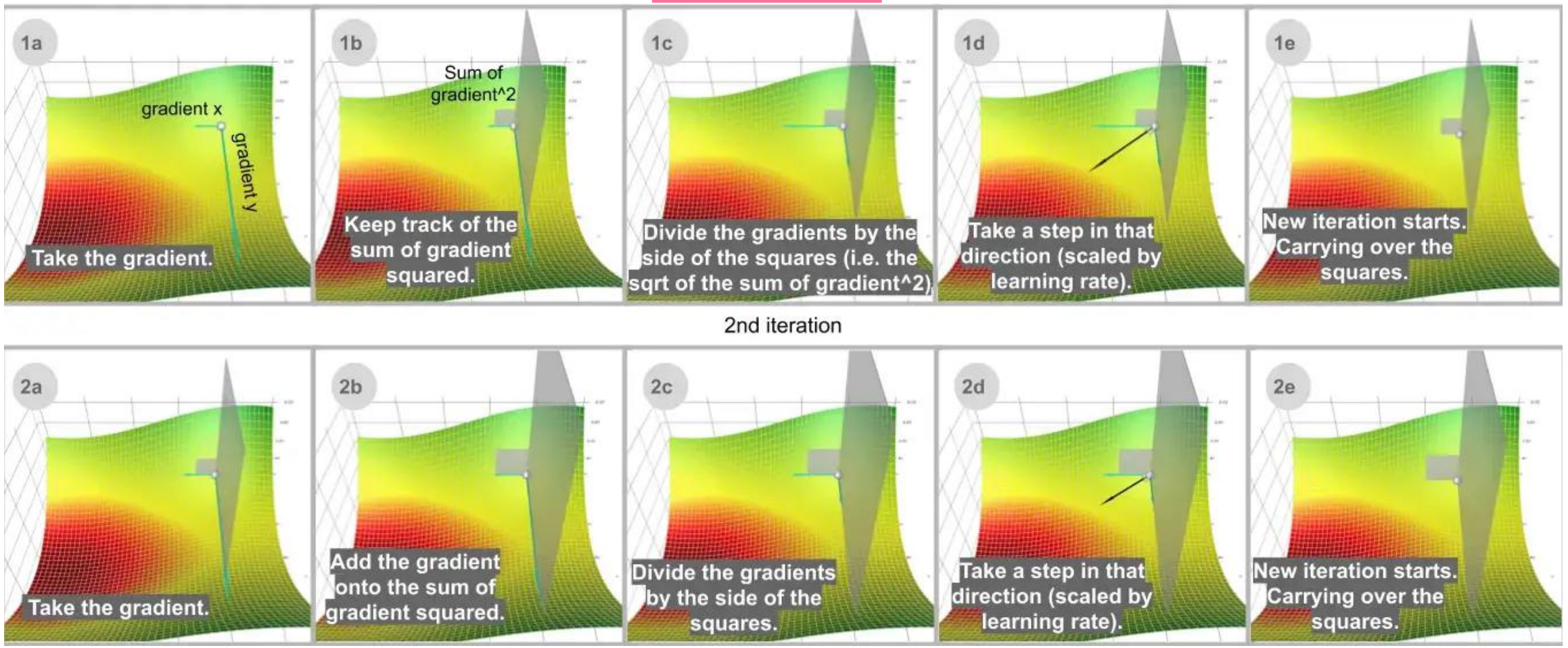
Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

Giới Thiệu Optimization Trong Machine Learning

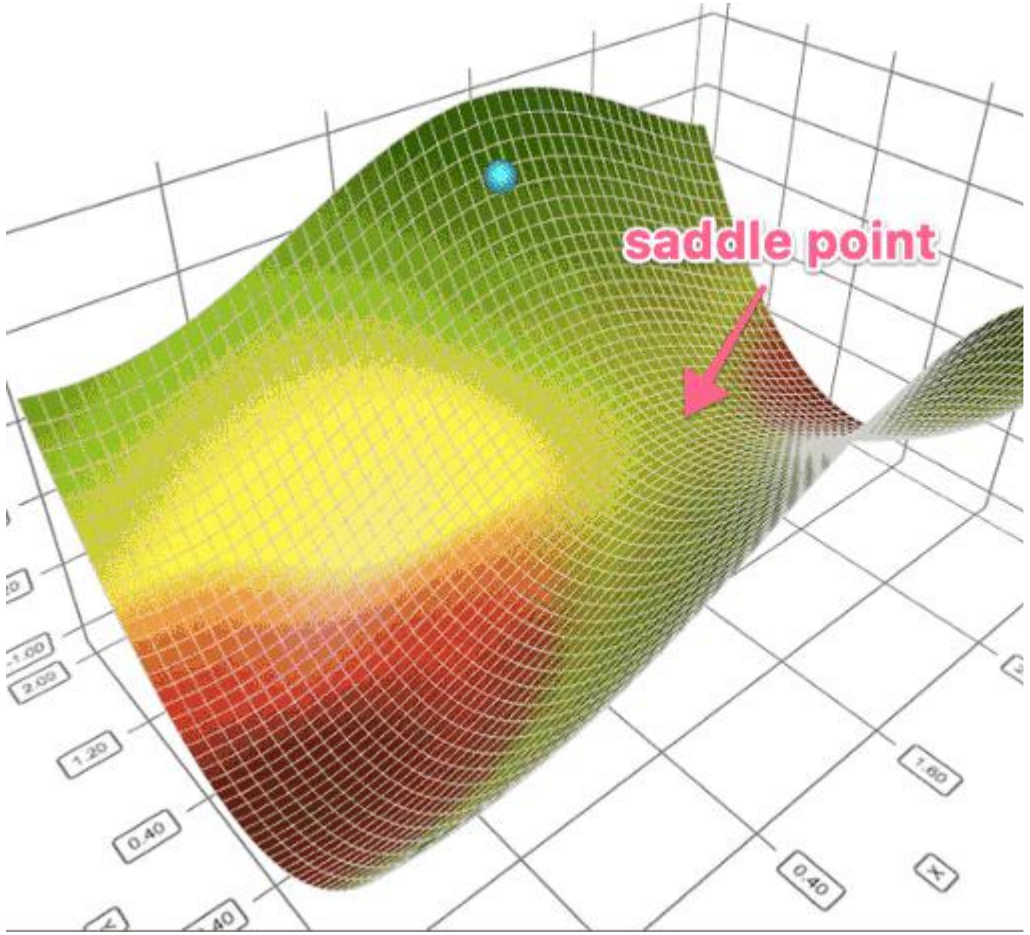
- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Adagrad

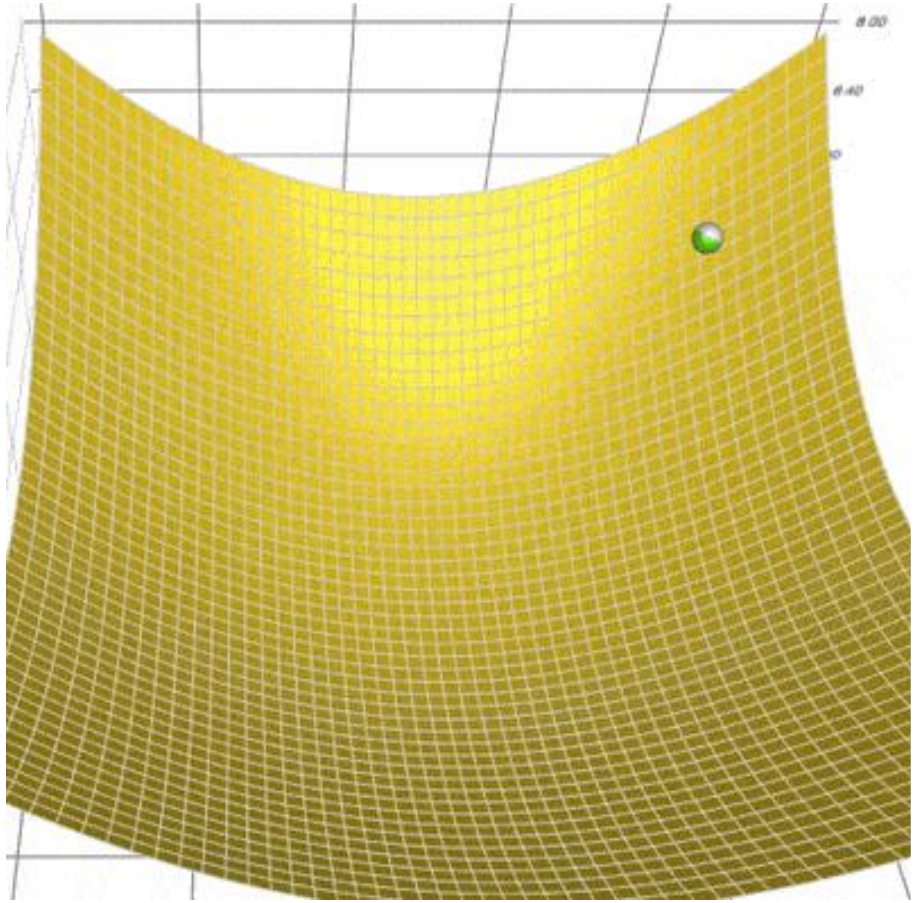


- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

GD vs Adagrad



Adagrad vs RMSProp



Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Momentum
Adaptive LR

Adam

AdaMax

Nadam

⋮ ⋮ ⋮

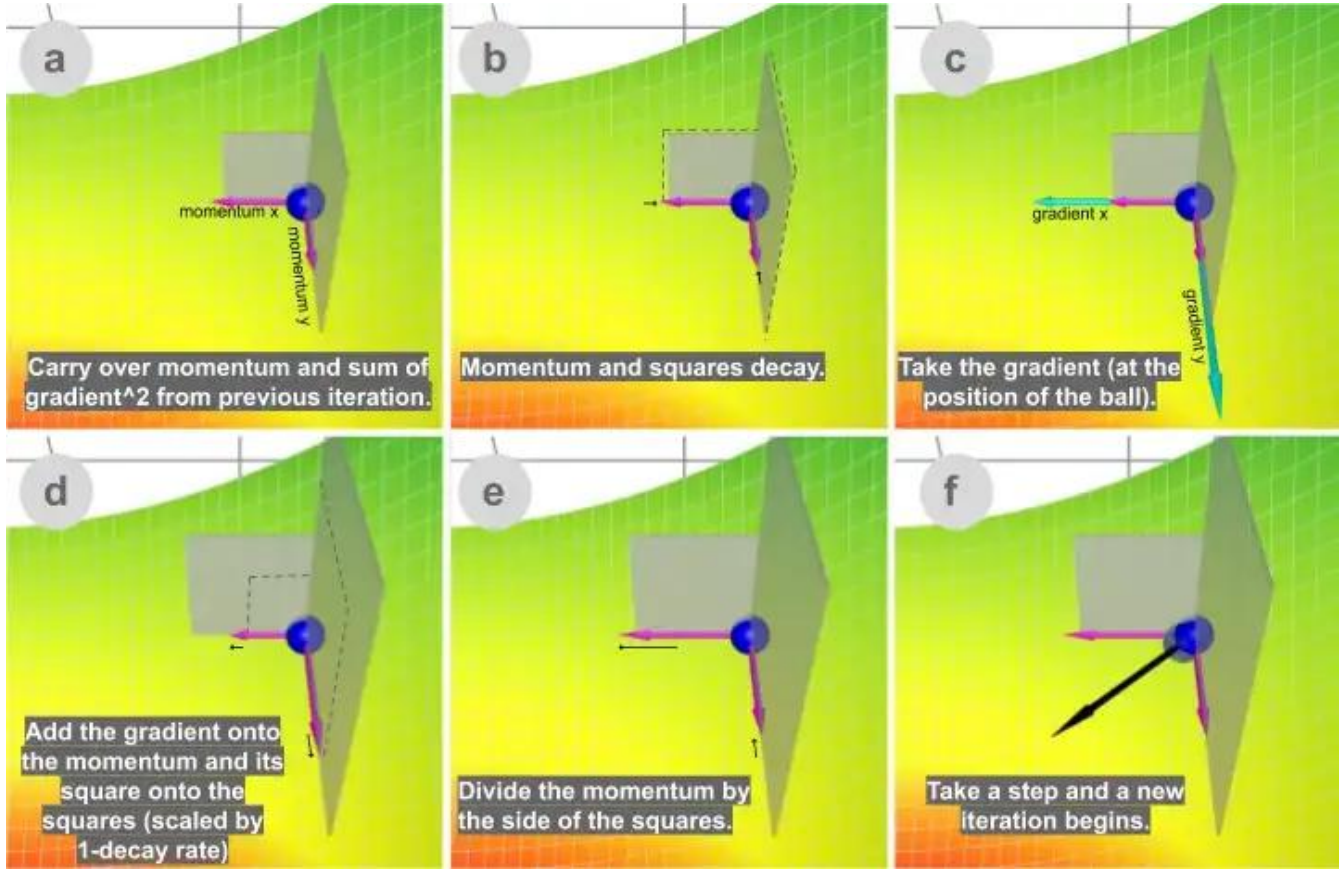
Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Adam

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)



Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

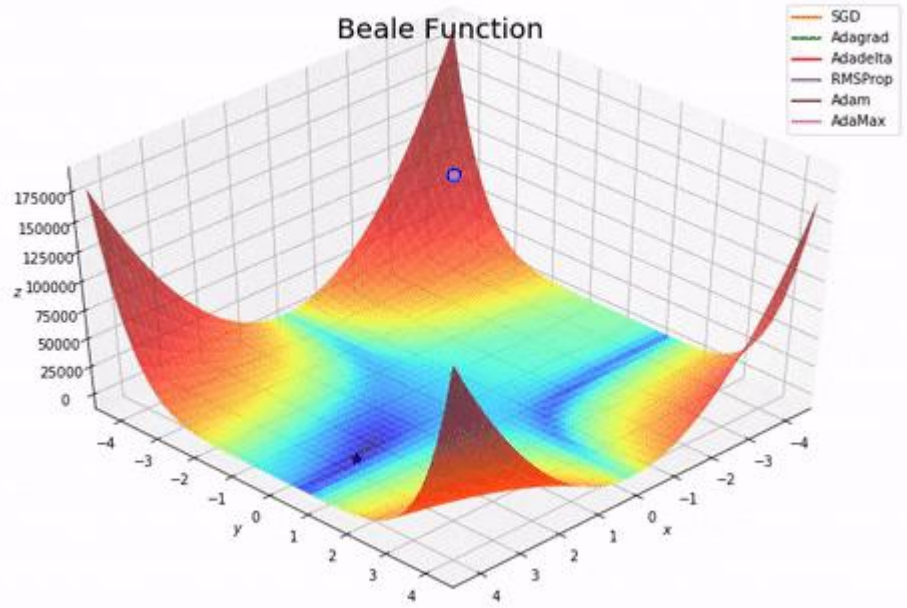
AdaMax

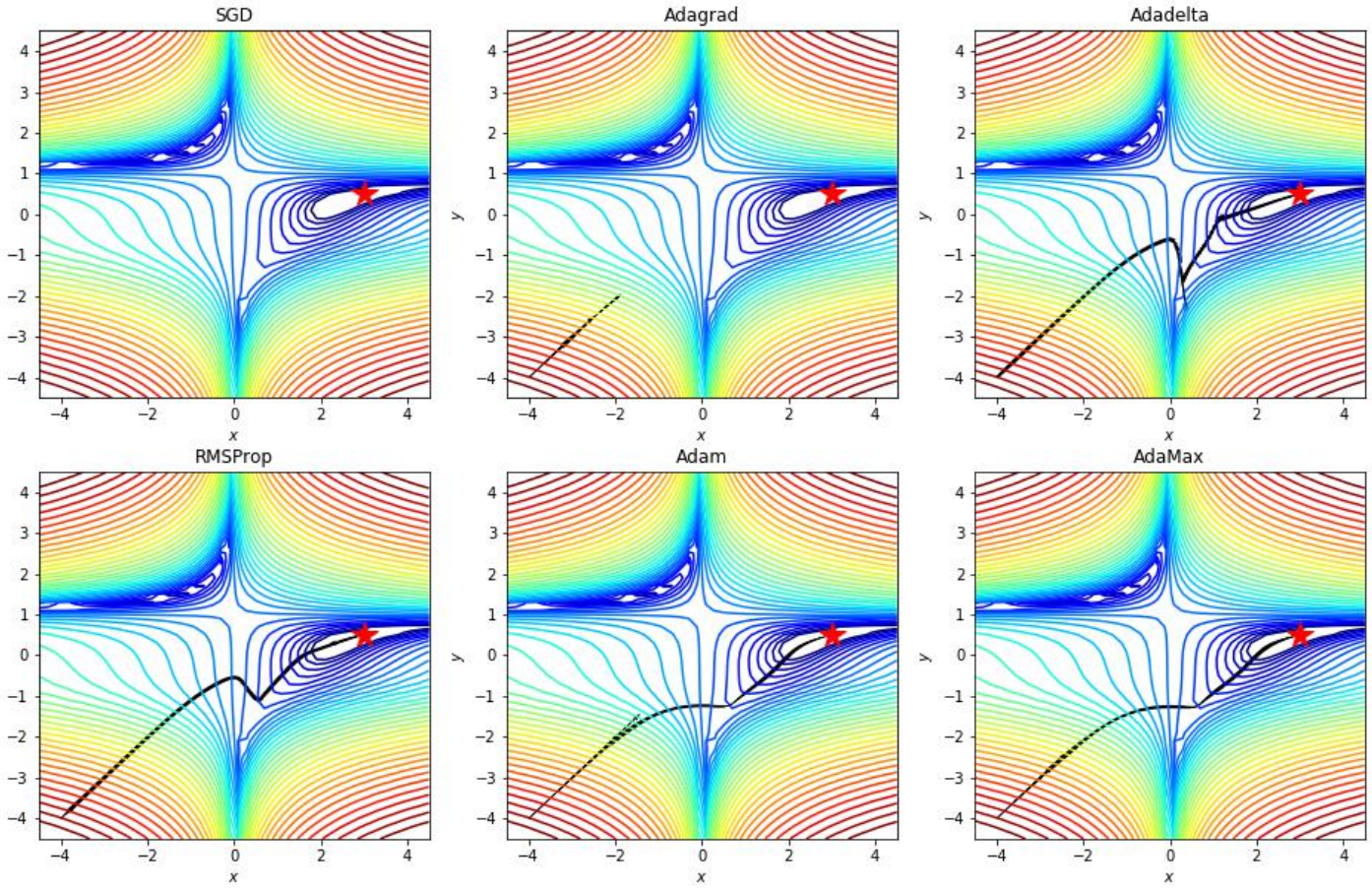
Algorithm 2: *AdaMax*, a variant of Adam based on the infinity norm. See section 7.1 for details. Good default settings for the tested machine learning problems are $\alpha = 0.002$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. With β_1^t we denote β_1 to the power t. Here, $(\alpha/(1-\beta_1^t))$ is the learning rate with the bias-correction term for the first moment. All operations on vectors are element-wise.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$u_0 \leftarrow 0$ (Initialize the exponentially weighted infinity norm)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$u_t \leftarrow \max(\beta_2 \cdot u_{t-1}, |g_t|)$ (Update the exponentially weighted infinity norm)
$\theta_t \leftarrow \theta_{t-1} - (\alpha/(1-\beta_1^t)) \cdot m_t/u_t$ (Update parameters)
end while
return θ_t (Resulting parameters)

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

AdaMax





- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Nadam

Algorithm 3 Nesterov's accelerated gradient

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta\mu\mathbf{m}_{t-1})$$
$$\mathbf{m}_t \leftarrow \mu\mathbf{m}_{t-1} + \mathbf{g}_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta\mathbf{m}_t$$

Algorithm 7 NAG rewritten

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\mathbf{m}_t \leftarrow \mu_t\mathbf{m}_{t-1} + \mathbf{g}_t$$
$$\bar{\mathbf{m}}_t \leftarrow \mathbf{g}_t + \mu_{t+1}\mathbf{m}_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta\bar{\mathbf{m}}_t$$

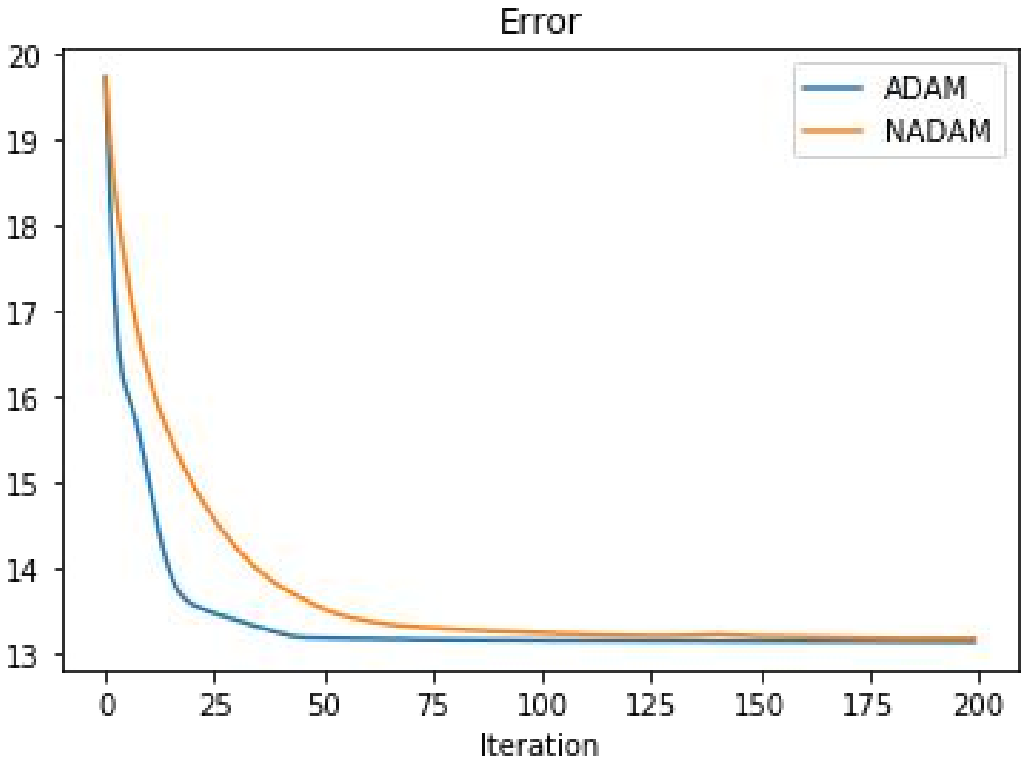
Algorithm 8 Nesterov-accelerated adaptive moment estimation

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\hat{\mathbf{g}} \leftarrow \frac{\mathbf{g}_t}{1-\prod_{i=1}^{t}\mu_i}$$
$$\mathbf{m}_t \leftarrow \mu\mathbf{m}_{t-1} + (1-\mu)\mathbf{g}_t$$
$$\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1-\prod_{i=1}^{t+1}\mu_i}$$
$$\mathbf{n}_t \leftarrow \nu\mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2$$
$$\hat{\mathbf{n}}_t \leftarrow \frac{\mathbf{n}_t}{1-\nu^t}$$
$$\bar{\mathbf{m}}_t \leftarrow (1-\mu_t)\hat{\mathbf{g}}_t + \mu_{t+1}\hat{\mathbf{m}}_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta\frac{\bar{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{n}}_t}+\varepsilon}$$

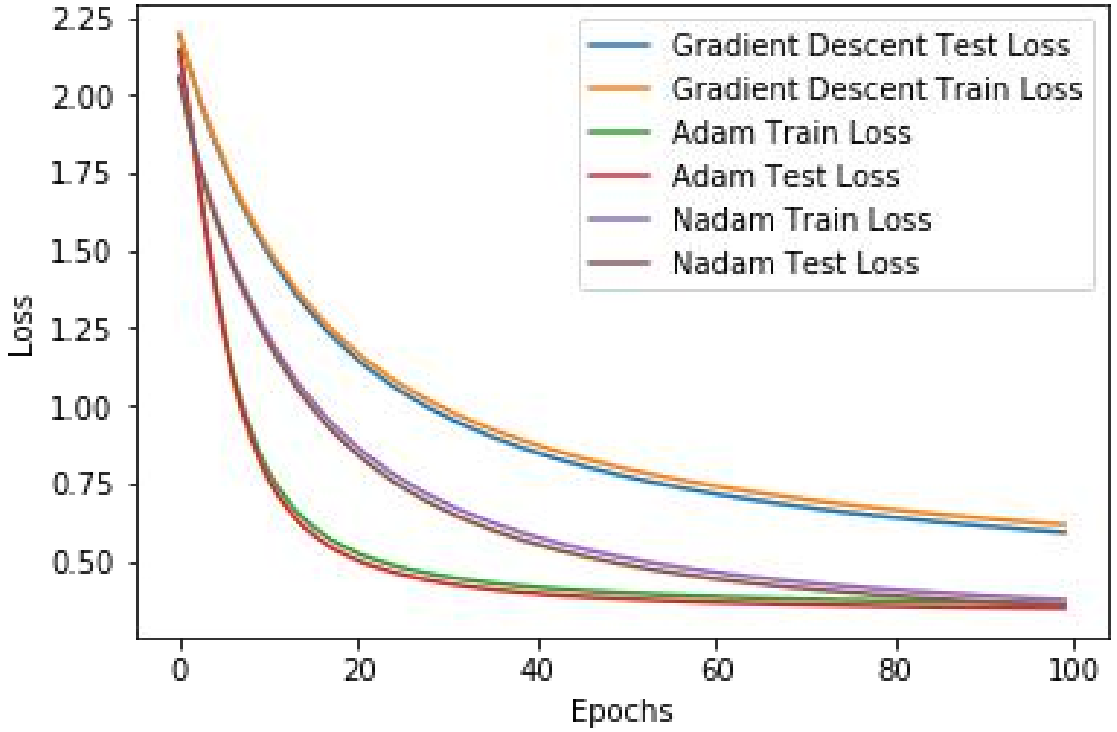
Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Linear Regression



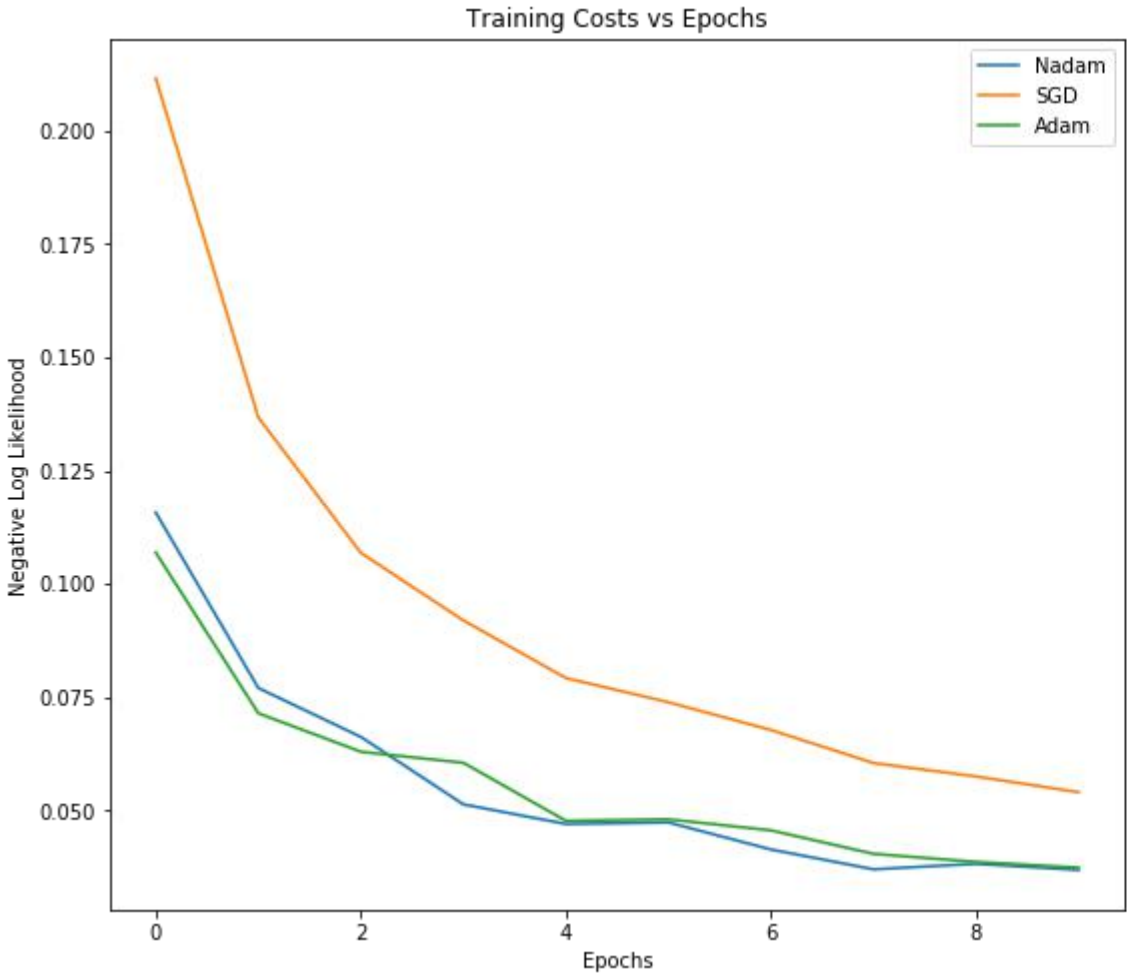
Logistic Regression MNIST

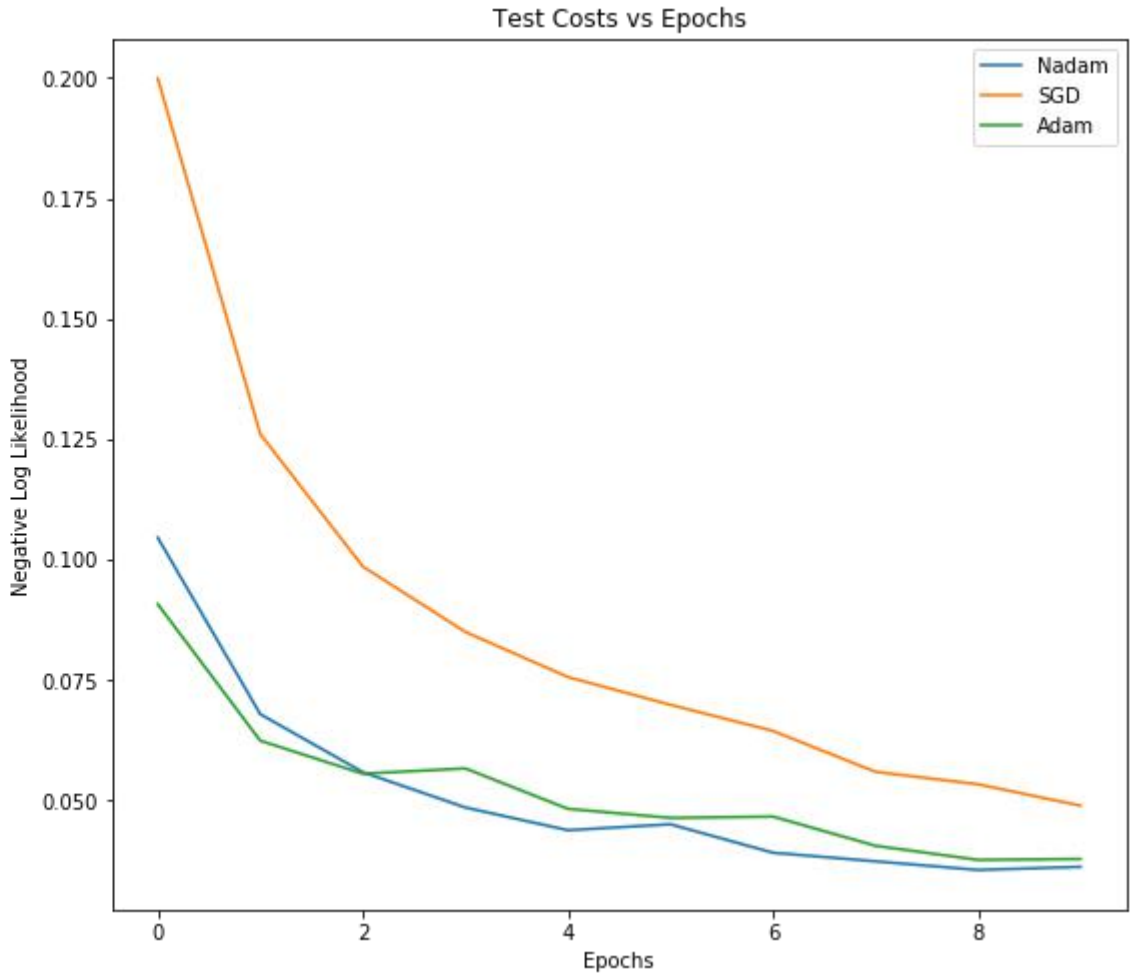


Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

MNIST Digit Classification with Convolutional Neural Networks





Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

AdaBelief

Algorithm 1: Adam Optimizer

Initialize θ_0, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$

While θ_t not converged

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

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

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

Bias Correction

$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}, \widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$

Update

$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{v_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \right)$

Algorithm 2: AdaBelief Optimizer

Initialize θ_0, $m_0 \leftarrow 0$, $s_0 \leftarrow 0$, $t \leftarrow 0$

While θ_t not converged

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

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

$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2 + \epsilon$

Bias Correction

$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}, \widehat{s_t} \leftarrow \frac{s_t}{1-\beta_2^t}$

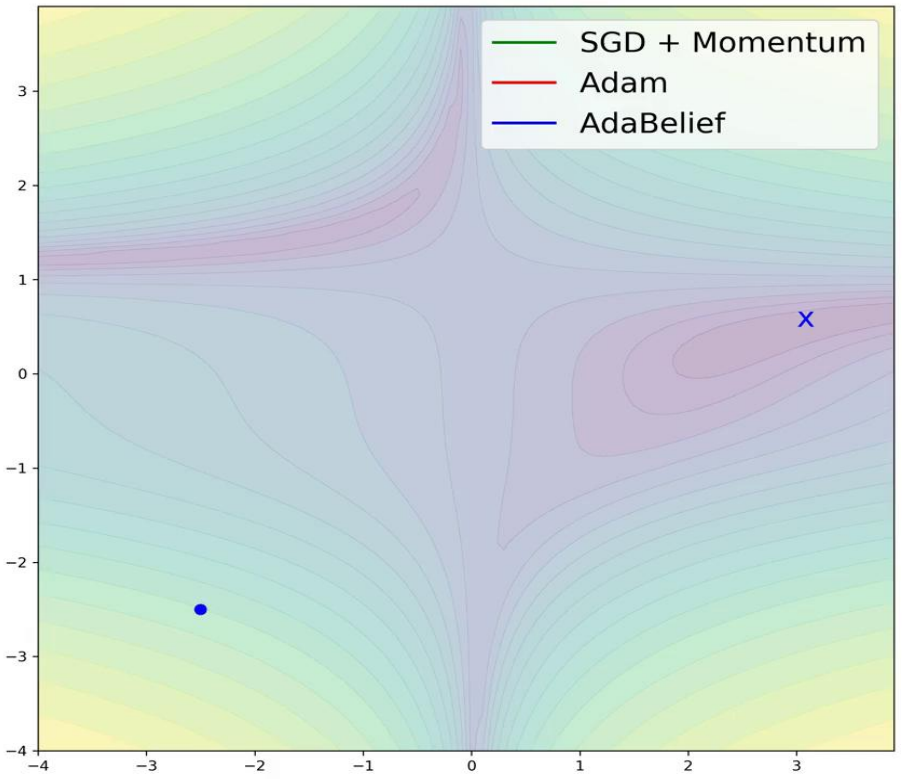
Update

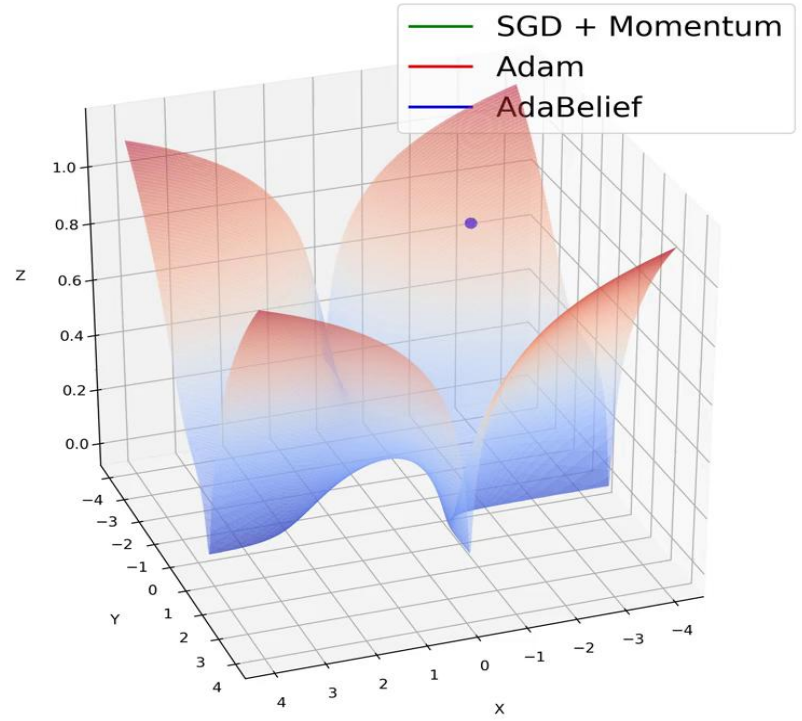
$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{s_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{s_t}} + \epsilon} \right)$

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

AdaBelief

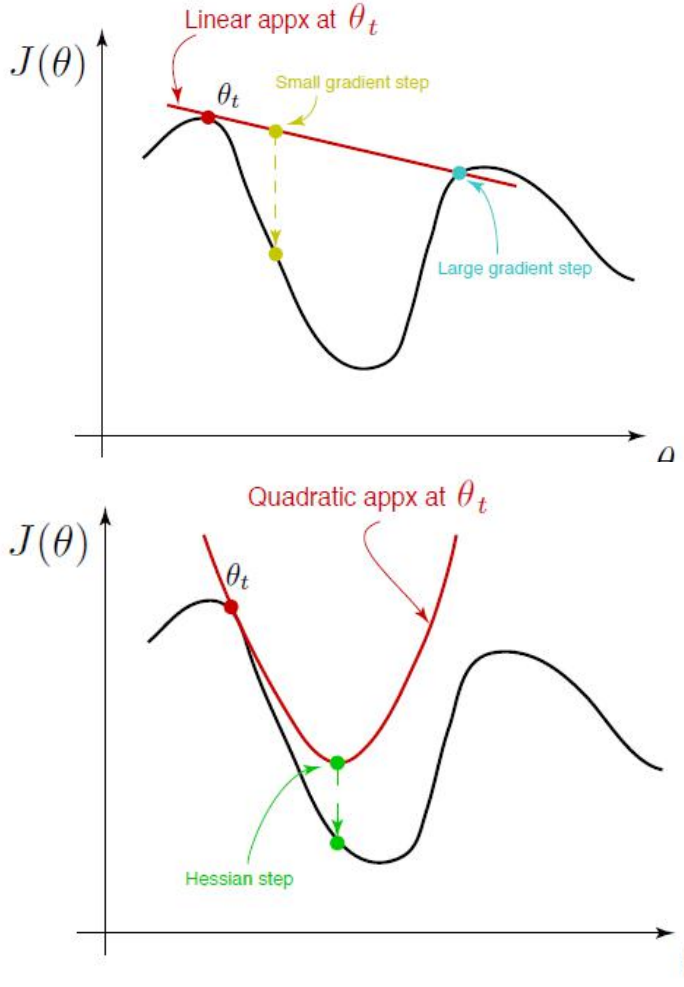




Giới Thiệu Optimization Trong Machine Learning

• Giới Thiệu Gradient-based Optimization Trong Machine Learning

Second-Order Derivatives Methods



- *Notation:*

objective function: $f : \mathbb{R}^n \to \mathbb{R}$

gradient vector: $\nabla f(x) = \left[\frac{\partial}{\partial_x} f(x)\right]^\top \in \mathbb{R}^n$

Hessian (symmetric matrix):

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2}{\partial_{x_1}\partial_{x_1}} f(x) & \frac{\partial^2}{\partial_{x_1}\partial_{x_2}} f(x) & \cdots & \frac{\partial^2}{\partial_{x_1}\partial_{x_n}} f(x) \\ \frac{\partial^2}{\partial_{x_1}\partial_{x_2}} f(x) & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial^2}{\partial_{x_n}\partial_{x_1}} f(x) & \cdots & \cdots & \frac{\partial^2}{\partial_{x_n}\partial_{x_n}} f(x) \end{pmatrix} \in \mathbb{R}^{n\times n}$$

Taylor expansion:

$$f(x') = f(x) + (x' - x)^\top \nabla f(x) + \frac{1}{2}(x' - x)^\top \nabla^2 f(x)\ (x' - x)$$

- *Problem:*

$$\min_x f(x)$$

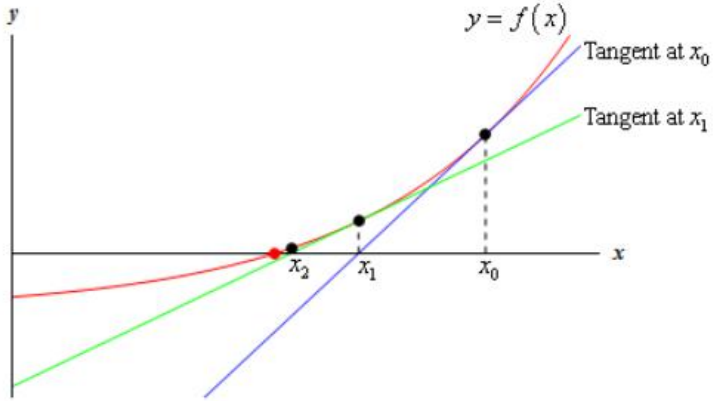
where we can evaluate $f(x)$, $\nabla f(x)$ and $\nabla^2 f(x)$ for any $x \in \mathbb{R}^n$

Giới Thiệu Optimization Trong Machine Learning

• Giới Thiệu Gradient-based Optimization Trong Machine Learning

Second-Order Derivatives Methods

- For finding roots (zero points) of $f(x)$



$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

- For finding optima of $f(x)$ in 1D:

$$x \leftarrow x - \frac{f'(x)}{f''(x)}$$

For $x \in \mathbb{R}^n$:

$$x \leftarrow x - \nabla^2 f(x)^{-1} \nabla f(x)$$

Algorithm 8.8 Newton's method with objective $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$

Require: Initial parameter $\boldsymbol{\theta}_0$
Require: Training set of m examples
while stopping criterion not met **do**
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute Hessian: $\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute Hessian inverse: $\boldsymbol{H}^{-1}$
  Compute update: $\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Second-Order Derivatives Methods

Table 1. Summary of second-order derivative method.

Method	Summary
Newton [5]	● Quadratic convergence only if the searching point is close to the minimum solution. ● Requires high computing cost to compute inverse Hessian at each iteration. ● Requires to stores large Hessian matrix to compute the updates for each iteration.
Conjugate gradient [7]	● Utilizes conjugate direction to replace the needs of costly inverse Hessian computation. ● Requires line-search approach to approximate step size therefore only suitable for batch learning.
Quasi-Newton [8]	● Utilizes BFGS for inverse Hessian estimation to reduce computational cost. ● Requires decent amount of memory to store gradient and update direction information from previous iteration.

[5] Nocedal J and Wright S 2006 Numerical Optimization Springer Series in Operations Research and Financial Engineering (Springer New York) ISBN 9780387303031
[7] Møller M F 1993 Neural Networks 6 525 – 533 ISSN 0893-6080 URL http://www.sciencedirect.com/science/article/pii/S0893608005800565
[8] Sohl-Dickstein J, Poole B and Ganguli S 2013 CoRR abs/1311.2115 (Preprint 1311.2115) URL http://arxiv.org/abs/1311.2115

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Gradient-based Optimization Trong Machine Learning**

Second-Order Derivatives Methods

Gauss-Newton [4]	● Simplify the Hessian computaiton by using squared Jacobian. ● Does not requires to store information from previous iteration to compute the update direction. ● Struggle with indefinite Hessian and may halt the training without making significant training progress.
Levenberg-Marquardt [10]	● Solves the indefinite Hessian problem with regularization parameter. ● The choice of regularization parameter is heuristic.
Hessian-free [14]	● Compute approximate Hessian with finite differences of gradient evaluation. ● Requires conjugate gradient to complete the optimization.
Approximate Greatest Descent [13]	● Utilizes two-phase spherical optimization strategy to compute trajectory based on control theory. ● The step size is controlled by the relative step length derived based on the constructed spherical regions. ● It is an adaptive method.

[4] LeCun Y, Bottou L, Orr G B and M¨uller K R 1998 Efficient BackProp (Berlin, Heidelberg: Springer Berlin Heidelberg) pp 9–50 ISBN 978-3-540-49430-0
[10] Wilamowski B M and Yu H 2010 IEEE Transactions on Neural Networks 21 930–937 ISSN 1045-9227
[13] Goh B S 2012 Latest Advances in Systems Science and Computational Intelligence 25–30
[14] Martens J 2010 Proceedings of the 27th International Conference on International Conference on Machine Learning 735–742 URL http://dl.acm.org/citation.cfm?id=3104322.3104416

Giới Thiệu Optimization Trong Machine Learning

• Giới Thiệu Sharpness-Aware Minimization

- Training dataset $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ drawn i.i.d. from distribution $\mathscr{D}$
- Neural network with weights $\boldsymbol{w} \in \mathcal{W} \subseteq \mathbb{R}^d$;
- Per-data-point loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$,
- Training loss $L_S(\boldsymbol{w}) \triangleq \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)$ which approximates $L_{\mathscr{D}}(\boldsymbol{w}) \triangleq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D}[l(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})]$.

Table: Train and test accuracy for a convolutional network trained on CIFAR10, for different batch sizes (reproducing an experiment from [SMN+16])

batch size	train accuracy	test accuracy	train loss
1	100.0 (100.0 - 100.0)	77.2 (77.7 - 76.4)	0.00 (0.00 - 0.00)
8	100.0 (100.0 - 100.0)	76.5 (76.7 - 75.9)	0.00 (0.00 - 0.00)
256	100.0 (100.0 - 100.0)	63.2 (63.4 - 61.3)	0.00 (0.00 - 0.00)
2048	100.0 (100.0 - 99.8)	60.2 (60.6 - 58.6)	0.00 (0.02 - 0.00)

$\Longrightarrow$ Train the network to get $\boldsymbol{w}$ having low population loss $L_{\mathscr{D}}(\boldsymbol{w})$.

Giới Thiệu Optimization Trong Machine Learning

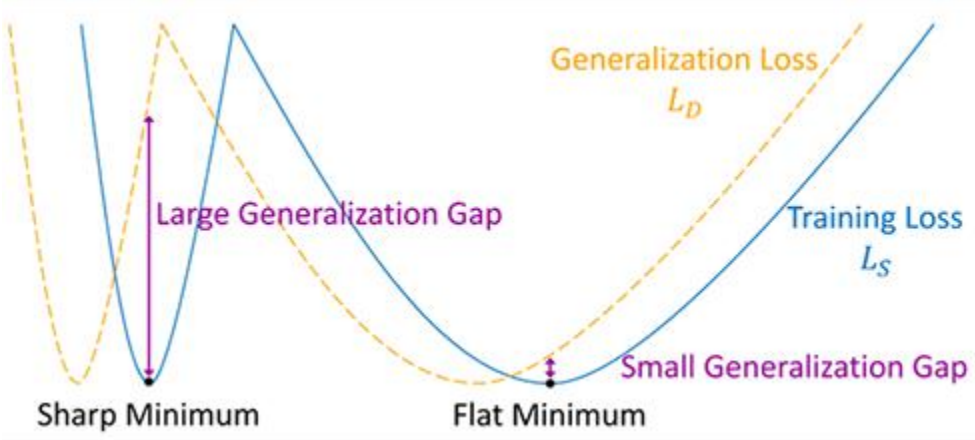
- **Giới Thiệu Sharpness-Aware Minimization**

How to leverage those theoretical findings to design better algorithms?

- Step 1: Find a quantity that is "correlated" to generalization (margin, some norm on the weights, sharpness, etc...).
- Step 2: Design an efficient algorithm to maximize this quantity.

Why sharpness?

- Strong empirical "correlation" with generalization in deep networks ([JNM+19], [SMN+16])
- Successful in the past (SWA: [IPG+18])



[JNM+19]: Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio, Fantastic Generalization Measures and Where to Find Them, arXiv e-prints (2019), arXiv:1912.02178.
[SMN+16]:Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang, On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, arXiv e-prints (2016), arXiv:1609.04836.
[IPG+18]: Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson, Averaging Weights Leads to Wider Optima and Better Generalization, arXiv e-prints (2018), arXiv:1803.05407.

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**

Theorem

For any $\rho > 0$, with high probability over training set $\mathcal{S}$ generated from distribution $\mathcal{D}$,

$$L_{\mathcal{D}}(\boldsymbol{w}) \leq \max_{\|\epsilon\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \epsilon) + h(\|\boldsymbol{w}\|_2^2 / \rho^2),$$

where $h : \mathbb{R}^+ \to \mathbb{R}^+$ is a strictly increasing function (under some technical conditions on $L_{\mathcal{D}}(\boldsymbol{w})$).

$$[\max_{\|\epsilon\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \epsilon) - L_{\mathcal{S}}(\boldsymbol{w})] + L_{\mathcal{S}}(\boldsymbol{w}) + h(\|\boldsymbol{w}\|_2^2 / \rho^2)$$

This gives us the SAM objective:

$$\min_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) + \lambda ||\boldsymbol{w}||_2^2 \qquad \text{where} \qquad L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\epsilon||_p \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \epsilon),$$

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**

SAM is an optimization algorithm

- Minimizes loss value AND sharpness.
- Is efficient and easy to implement.
- Strongly improves generalization (SOTA on Imagenet, CIFAR, SVHN, and others).
- Is robust to label noise.

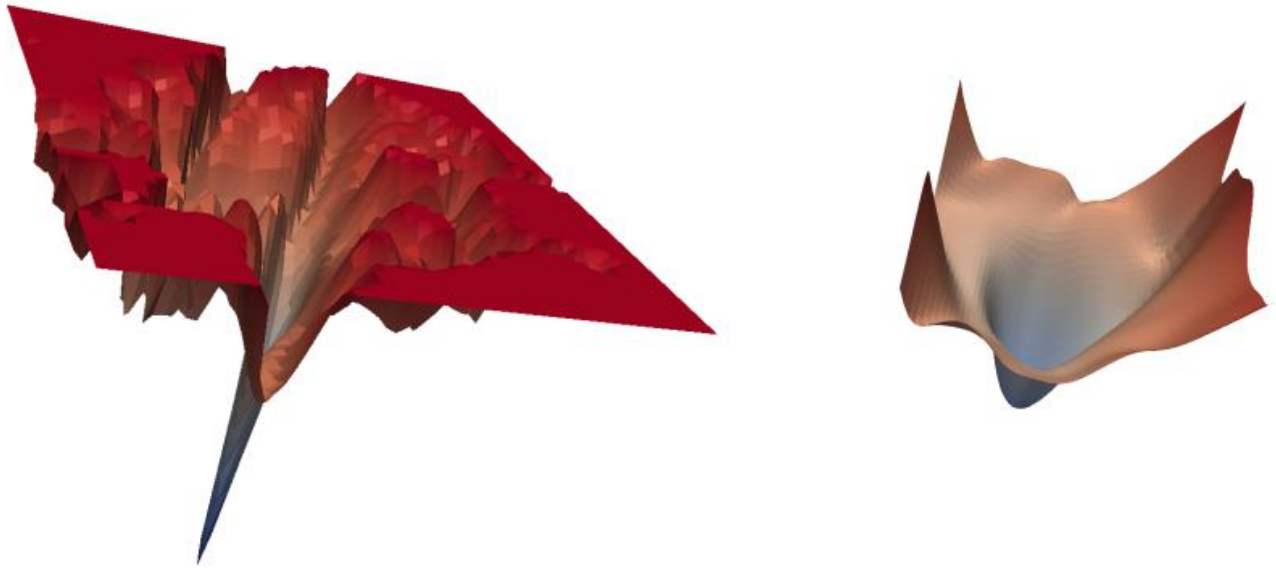


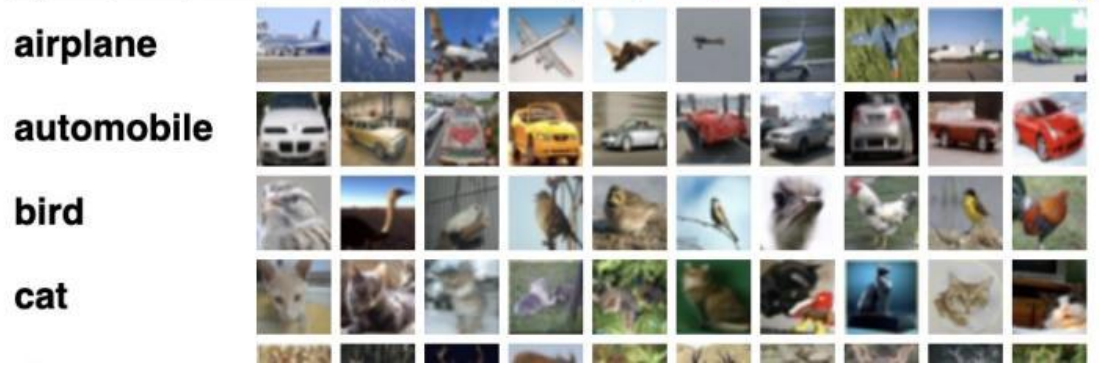
Figure: (left) A sharp minimum to which a ResNet trained with SGD converged. (right) A wide minimum to which the same ResNet trained with SAM converged.

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**

paperswithcode.com/sota/image-classification-on-cifar-100

Rank	Model	Percentage correct	PARAMS	Extra Training Data	Paper	Code
1	EffNet-L2 (SAM)	96.08		✓	Sharpness-Aware Minimization for Efficiently Improving Generalization	
2	Swin-L + ML-Decoder	95.1		✓	ML-Decoder: Scalable and Versatile Classification Head	
3	ViT-H/14	94.55±0.04		✓	An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale	
4	ViT-B-16 (ImageNet-21K-P pretrain)	94.2		✓	ImageNet-21K Pretraining for the Masses	
5	CvT-W24	94.09		✓	CvT: Introducing Convolutions to Vision Transformers	
6	ViT-L/16	93.90±0.05		✓	An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale	
7	BiT-L	93.51		✓	Big Transfer (BiT): General Visual Representation	



paperswithcode.com/sota/image-classification-on-svhn

Rank	Model	Percentage error	Paper	Code	Result	Year
1	WRN28-10 (SAM)	0.99	Sharpness-Aware Minimization for Efficiently Improving Generalization			2020
2	E2E-M3	1.0	Rethinking Recurrent Neural Networks and Other Improvements for Image Classification			2020
3	Wide-ResNet-28-10 (Fast AA)	1.1	Fast AutoAugment			2019
4	Colornet	1.11	ColorNet: Investigating the importance of color spaces for image classification			2019
5	PBA [ho2019pba]	1.2	Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules			2019
6	Cutout	1.30	Improved Regularization of Convolutional Neural Networks with Cutout			2017
7	PyramidNet + AA (AMP)	1.35	Regularizing Neural Networks via Adversarial Model Perturbation			2020

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**

Input: Training set $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, Loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, Batch size b, Step size $\eta > 0$, Neighborhood size $\rho > 0$.

Output: Model trained with SAM

Initialize weights $\boldsymbol{w}_0$, $t = 0$;

while *not converged* **do**

  Sample batch $\mathcal{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ...(\boldsymbol{x}_b, \boldsymbol{y}_b)\}$;

  $\delta(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})$;

  $\hat{\epsilon} = \frac{\delta(\boldsymbol{w}_t)}{\|\delta(\boldsymbol{w}_t)\|}$;

  $\boldsymbol{w}_{\text{adv}} = \boldsymbol{w}_t + \hat{\epsilon}$;

  $\boldsymbol{g} = \delta(\boldsymbol{w}_{\text{adv}})$;

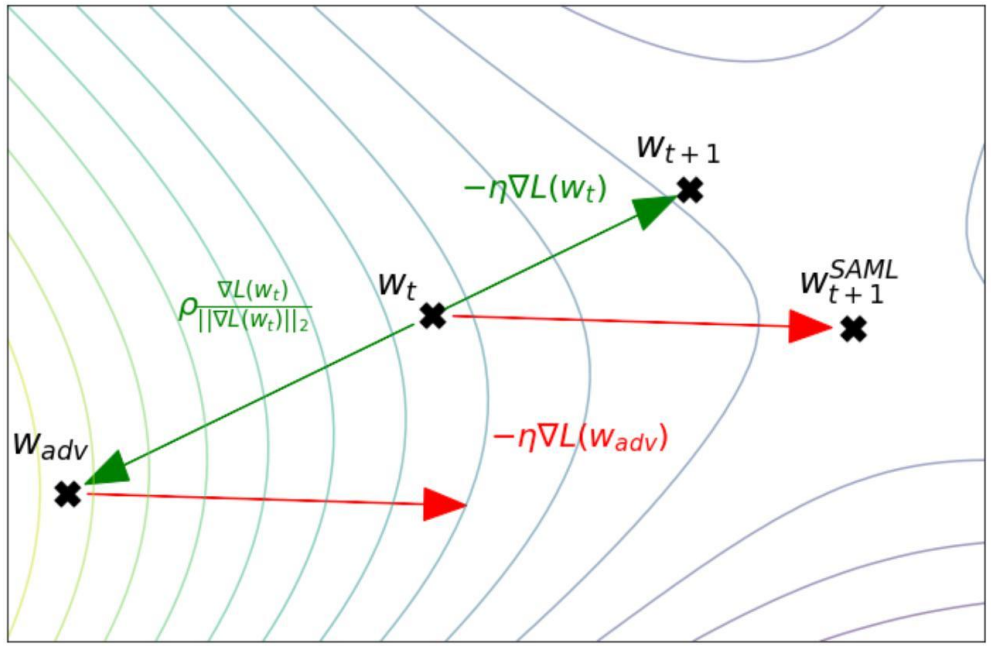
  $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}$;

  $t = t + 1$;

end

return $\boldsymbol{w}_t$

One update of SAM against one update of plain gradient descent.



Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**

		CIFAR-10		CIFAR-100	
Model	Augmentation	SAM	SGD	SAM	SGD
WRN-28-10 (200 epochs)	Basic	**2.7**$_{\pm 0.1}$	3.5$_{\pm 0.1}$	**16.5**$_{\pm 0.2}$	18.8$_{\pm 0.2}$
WRN-28-10 (200 epochs)	Cutout	**2.3**$_{\pm 0.1}$	2.6$_{\pm 0.1}$	**14.9**$_{\pm 0.2}$	16.9$_{\pm 0.1}$
WRN-28-10 (200 epochs)	AA	**2.1**$_{\pm <0.1}$	2.3$_{\pm 0.1}$	**13.6**$_{\pm 0.2}$	15.8$_{\pm 0.2}$
WRN-28-10 (1800 epochs)	Basic	**2.4**$_{\pm 0.1}$	3.5$_{\pm 0.1}$	**16.3**$_{\pm 0.2}$	19.1$_{\pm 0.1}$
WRN-28-10 (1800 epochs)	Cutout	**2.1**$_{\pm 0.1}$	2.7$_{\pm 0.1}$	**14.0**$_{\pm 0.1}$	17.4$_{\pm 0.1}$
WRN-28-10 (1800 epochs)	AA	**1.6**$_{\pm 0.1}$	2.2$_{\pm <0.1}$	**12.8**$_{\pm 0.2}$	16.1$_{\pm 0.2}$
Shake-Shake (26 2x96d)	Basic	**2.3**$_{\pm <0.1}$	2.7$_{\pm 0.1}$	**15.1**$_{\pm 0.1}$	17.0$_{\pm 0.1}$
Shake-Shake (26 2x96d)	Cutout	**2.0**$_{\pm <0.1}$	2.3$_{\pm 0.1}$	**14.2**$_{\pm 0.2}$	15.7$_{\pm 0.2}$
Shake-Shake (26 2x96d)	AA	**1.6**$_{\pm <0.1}$	1.9$_{\pm 0.1}$	**12.8**$_{\pm 0.1}$	14.1$_{\pm 0.2}$
PyramidNet	Basic	**2.7**$_{\pm 0.1}$	4.0$_{\pm 0.1}$	**14.6**$_{\pm 0.4}$	19.7$_{\pm 0.3}$
PyramidNet	Cutout	**1.9**$_{\pm 0.1}$	2.5$_{\pm 0.1}$	**12.6**$_{\pm 0.2}$	16.4$_{\pm 0.1}$
PyramidNet	AA	**1.6**$_{\pm 0.1}$	1.9$_{\pm 0.1}$	**11.6**$_{\pm 0.1}$	14.6$_{\pm 0.1}$
PyramidNet+ShakeDrop	Basic	**2.1**$_{\pm 0.1}$	2.5$_{\pm 0.1}$	**13.3**$_{\pm 0.2}$	14.5$_{\pm 0.1}$
PyramidNet+ShakeDrop	Cutout	**1.6**$_{\pm <0.1}$	1.9$_{\pm 0.1}$	**11.3**$_{\pm 0.1}$	11.8$_{\pm 0.2}$
PyramidNet+ShakeDrop	AA	**1.4**$_{\pm <0.1}$	1.6$_{\pm <0.1}$	**10.3**$_{\pm 0.1}$	10.6$_{\pm 0.1}$

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**

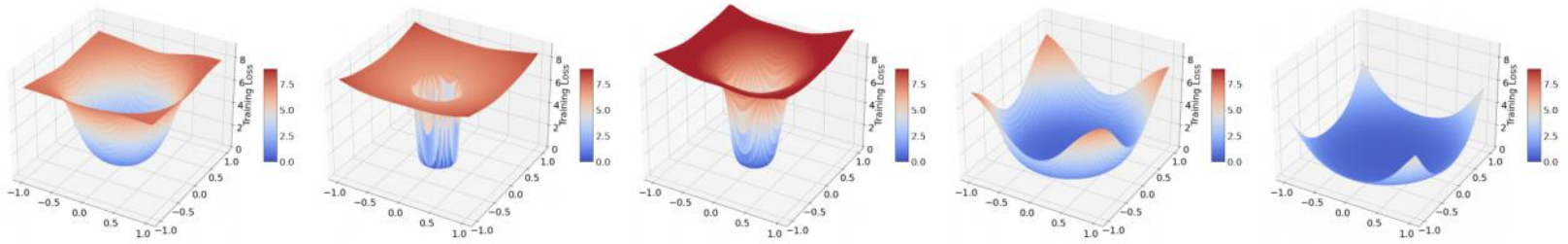
Model	Epoch	SAM		Standard Training (No SAM)	
		Top-1	Top-5	Top-1	Top-5
ResNet-50	100	**22.5**$_{\pm 0.1}$	6.28$_{\pm 0.08}$	22.9$_{\pm 0.1}$	6.62$_{\pm 0.11}$
	200	**21.4**$_{\pm 0.1}$	5.82$_{\pm 0.03}$	22.3$_{\pm 0.1}$	6.37$_{\pm 0.04}$
	400	**20.9**$_{\pm 0.1}$	5.51$_{\pm 0.03}$	22.3$_{\pm 0.1}$	6.40$_{\pm 0.06}$
ResNet-101	100	**20.2**$_{\pm 0.1}$	5.12$_{\pm 0.03}$	21.2$_{\pm 0.1}$	5.66$_{\pm 0.05}$
	200	**19.4**$_{\pm 0.1}$	4.76$_{\pm 0.03}$	20.9$_{\pm 0.1}$	5.66$_{\pm 0.04}$
	400	**19.0**$_{\pm <0.01}$	4.65$_{\pm 0.05}$	22.3$_{\pm 0.1}$	6.41$_{\pm 0.06}$
ResNet-152	100	**19.2**$_{\pm <0.01}$	4.69$_{\pm 0.04}$	20.4$_{\pm <0.0}$	5.39$_{\pm 0.06}$
	200	**18.5**$_{\pm 0.1}$	4.37$_{\pm 0.03}$	20.3$_{\pm 0.2}$	5.39$_{\pm 0.07}$
	400	**18.4**$_{\pm <0.01}$	4.35$_{\pm 0.04}$	20.9$_{\pm <0.0}$	5.84$_{\pm 0.07}$

Dataset	EffNet-b7 + SAM	EffNet-b7	Prev. SOTA (ImageNet only)	EffNet-L2 + SAM	EffNet-L2	Prev. SOTA
FGVC_Aircraft	6.80$_{\pm 0.06}$	8.15$_{\pm 0.08}$	**5.3** (TBMSL-Net)	**4.82**$_{\pm 0.08}$	5.80$_{\pm 0.1}$	5.3 (TBMSL-Net)
Flowers	**0.63**$_{\pm 0.02}$	1.16$_{\pm 0.05}$	0.7 (BiT-M)	**0.35**$_{\pm 0.01}$	0.40$_{\pm 0.02}$	0.37 (EffNet)
Oxford_IIIT_Pets	**3.97**$_{\pm 0.04}$	4.24$_{\pm 0.09}$	4.1 (Gpipe)	**2.90**$_{\pm 0.04}$	3.08$_{\pm 0.04}$	4.1 (Gpipe)
Stanford_Cars	5.18$_{\pm 0.02}$	5.94$_{\pm 0.06}$	**5.0** (TBMSL-Net)	4.04$_{\pm 0.03}$	4.93$_{\pm 0.04}$	**3.8** (DAT)
CIFAR-10	**0.88**$_{\pm 0.02}$	0.95$_{\pm 0.03}$	1 (Gpipe)	**0.30**$_{\pm 0.01}$	0.34$_{\pm 0.02}$	0.63 (BiT-L)
CIFAR-100	**7.44**$_{\pm 0.06}$	7.68$_{\pm 0.06}$	7.83 (BiT-M)	**3.92**$_{\pm 0.06}$	4.07$_{\pm 0.08}$	6.49 (BiT-L)
Birdsnap	**13.64**$_{\pm 0.15}$	14.30$_{\pm 0.18}$	15.7 (EffNet)	**9.93**$_{\pm 0.15}$	10.31$_{\pm 0.15}$	14.5 (DAT)
Food101	7.02$_{\pm 0.02}$	7.17$_{\pm 0.03}$	7.0 (Gpipe)	**3.82**$_{\pm 0.01}$	3.97$_{\pm 0.03}$	4.7 (DAT)
ImageNet	15.14$_{\pm 0.03}$	15.3	**14.2** (KDforAA)	**11.39**$_{\pm 0.02}$	11.8	11.45 (ViT)

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**

ViT (and MLP-Mixer) converge to much sharper local optima than what ResNet does. Consequently, SAM can substantially improve their accuracy (+5.3%, +11.0%) and robustness (+9.9%, +16.9%) [CHG22].



(a) ResNet (b) ViT (c) Mixer (d) ViT-SAM (e) Mixer-SAM

Figure 1: Cross-entropy loss landscapes of ResNet-152, ViT-B/16, and Mixer-B/16. ViT and MLP-Mixer converge to sharper regions than ResNet when trained on ImageNet with the basic Inception-style preprocessing. SAM, a sharpness-aware optimizer, significantly smooths the landscapes.

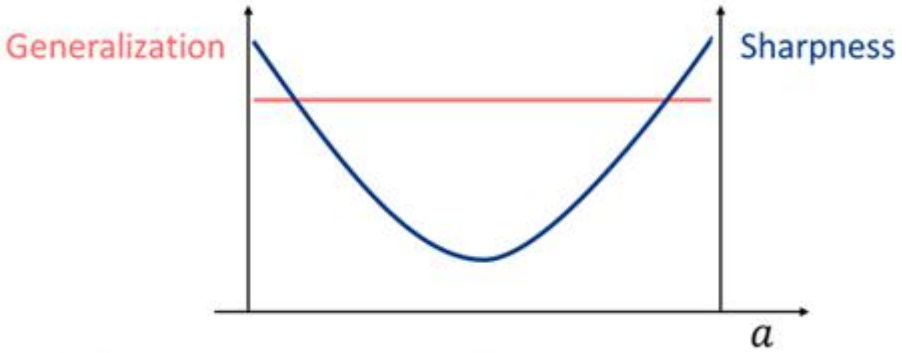
Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Sharpness-Aware Minimization**
  - **ASAM**: Adaptive version of SAM which is scale-invariant
  - **ESAM**: A more efficient version of SAM
  - **WASAM**: Weight Averaging SAM
  - **GSAM**: A Gap Guided Sharpness-Aware Minimization
  - **rwSAM**: Reweighted SAM for Imbalance Data
  - **LookSAM**: Faster SAM for training Vision Transformers
  - **SAMQ** & **SAQ**: SAM for quantized models
  - **SALR**: Sharpness-Aware Learning Rate Scheduler
  - δ-**SAM**: SAM with Dynamic Reweighting
  - Generalized Version of SAM
  - Understanding SAM

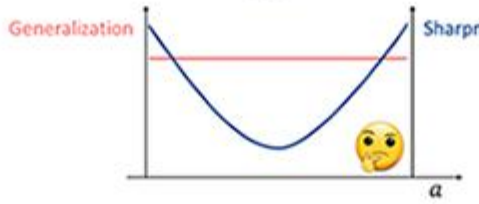
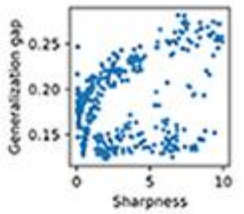
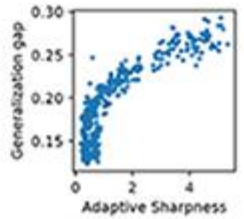
Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Adaptive Sharpness-Aware Minimization**

EX) $f(x;a) = \frac{1}{a} w_1 ReLU(a w_2 x)$

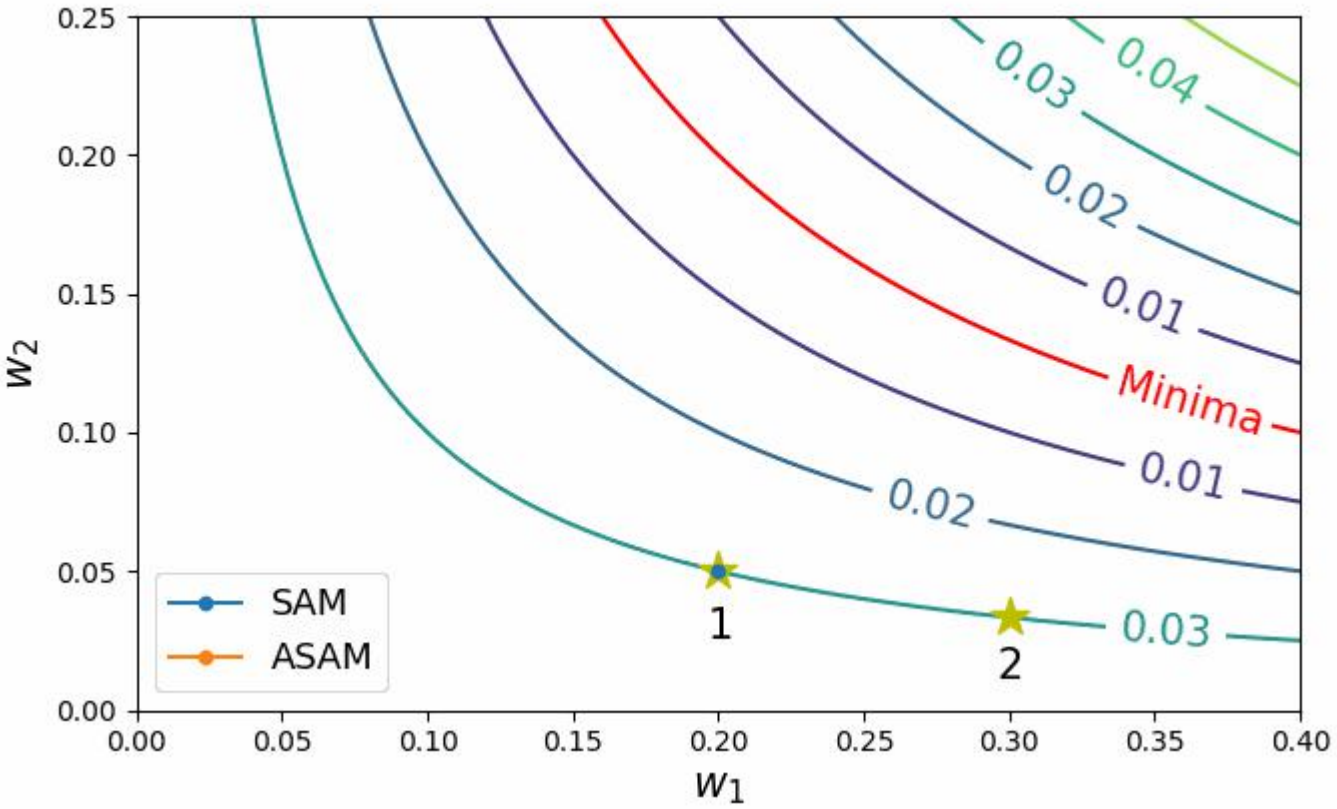


$$\max_{\|T_{\mathbf{w}}^{-1}\epsilon\|_p \leq \rho} L_S(\mathbf{w}+\epsilon) - L_S(\mathbf{w})$$

	Maximization region	Robustness to weight scaling	Scatter plots	Correlation
Sharpness (Foret et al., 2021)	$\|\epsilon\|_p \leq \rho$, $p = 2$: Sphere	No			$\psi = 0.316$, $\tau = 0.257$
Adaptive Sharpness (Ours)	$\|T_{\mathbf{w}}^{-1}\epsilon\|_p \leq \rho$, $p = 2$: Ellipsoid	Yes			$\psi = 0.626$, $\tau = 0.616$ (Higher is better)

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Adaptive Sharpness-Aware Minimization**



Converged to a minimum?

	SAM	ASAM
Point 1	O	O
Point 2	X	O

Giới Thiệu Optimization Trong Machine Learning

- **Giới Thiệu Adaptive Sharpness-Aware Minimization**

Model	CIFAR-10			CIFAR-100		
	SGD	SAM	ASAM	SGD	SAM	ASAM
DenseNet-121	$91.00_{\pm 0.13}$	$92.00_{\pm 0.17}$	$\mathbf{93.33}_{\pm 0.04}$	$68.70_{\pm 0.31}$	$69.84_{\pm 0.12}$	$\mathbf{70.60}_{\pm 0.20}$
ResNet-20	$93.18_{\pm 0.21}$	$93.56_{\pm 0.15}$	$\mathbf{93.82}_{\pm 0.17}$	$69.76_{\pm 0.44}$	$71.06_{\pm 0.31}$	$\mathbf{71.40}_{\pm 0.30}$
ResNet-56	$94.58_{\pm 0.20}$	$95.18_{\pm 0.15}$	$\mathbf{95.42}_{\pm 0.16}$	$73.12_{\pm 0.19}$	$75.16_{\pm 0.05}$	$\mathbf{75.86}_{\pm 0.22}$
VGG19-BN	$93.87_{\pm 0.09}$	94.60	$\mathbf{95.07}_{\pm 0.05}$	$71.80_{\pm 1.35}$	$73.52_{\pm 1.74}$	$\mathbf{75.80}_{\pm 0.27}$
ResNeXt29-32x4d	$95.84_{\pm 0.24}$	$96.34_{\pm 0.30}$	$\mathbf{96.80}_{\pm 0.06}$	$79.76_{\pm 0.23}$	$81.48_{\pm 0.17}$	$\mathbf{82.30}_{\pm 0.11}$
WRN-28-2	$95.13_{\pm 0.16}$	$95.74_{\pm 0.08}$	$\mathbf{95.94}_{\pm 0.05}$	$75.28_{\pm 0.17}$	$77.25_{\pm 0.35}$	$\mathbf{77.54}_{\pm 0.14}$
WRN-28-10	$96.34_{\pm 0.12}$	$96.98_{\pm 0.04}$	$\mathbf{97.28}_{\pm 0.07}$	$81.56_{\pm 0.13}$	$83.42_{\pm 0.04}$	$\mathbf{83.68}_{\pm 0.12}$
PyramidNet-272	$98.44_{\pm 0.08}$	$98.55_{\pm 0.05}$	$\mathbf{98.68}_{\pm 0.08}$	$88.91_{\pm 0.12}$	$89.36_{\pm 0.20}$	$\mathbf{89.90}_{\pm 0.13}$

Table 2 Maximum test accuracies on CIFAR-{10, 100}.

	SGD	SAM	ASAM
Top1	$75.79_{\pm 0.22}$	$76.39_{\pm 0.03}$	$\mathbf{76.63}_{\pm 0.18}$
Top5	$92.62_{\pm 0.04}$	$92.97_{\pm 0.07}$	$\mathbf{93.16}_{\pm 0.18}$

Table 3 ImageNet using ResNet-50.

Content

- **Giới Thiệu Optimization Trong Machine Learning**
  - Giới Thiệu Optimization
  - Giới Thiệu Gradient-based Optimization Trong Machine Learning
  - Giới Thiệu Sharpness-Aware Minimization
- **Optimizing Functions of Two Varibales**
  - Giới thiệu vấn đề
  - Exercise1: Gradient Descent
  - Exercise2: Gradient Descent + Momentum
  - Exercise3: RMSProp
  - Exercise4: Adam
- **Vanishing Problem (Optional)**
  - GD, GD + Momentum, RMSProp, và Adam

Optimizing Functions of Two Varibales

- **Giới thiệu vấn đề**

Cho một function 2 biến (1). Sử dụng các thuật toán optimizationđể tìm điểm minimum của function:

(a) Trình bày chi tiết từng thuật toán với 2 lần lặp (latex, doc, ...)

(b) Thực hiện code với 30 lần lặp (sử dụng thư viện Numpy)

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \qquad (1)$$

Optimizing Functions of Two Varibales

- **Exercise1: Gradient Descent**

1. **Gradient Descent:** Dựa vào thuật toán Gradient Descent các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w1 = -5$, $w2 = -2$, $\alpha = 0.4$:

   (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.

   (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \qquad (1)$$

Gradient Descent: $W = W - \alpha * dW$ (1.1)

Optimizing Functions of Two Varibales

- **Exercise1: Gradient Descent**

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

Gradient Descent: $W = W - \alpha * dW$ (1.1)

1. **Gradient Descent:** Dựa vào thuật toán Gradient Descent các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w1 = -5$, $w2 = -2$, $\alpha = 0.4$:

   (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.

   (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).

- epoch=1:
  - $dw_1 = 0.2w_1 = 0.2 * (-5) = -1$
  - $dw_2 = 4w_2 = 4 * (-2) = -8$
  - $w_1 = w_1 - \alpha * dw_1 = -5 - 0.4 * (-1) = -4.6$
  - $w_2 = w_2 - \alpha * dw_2 = -2 - 0.4 * (-8) = 1.2$

- epoch=2:
  - $dw_1 = 0.2w_1 = 0.2 * (-4.6) = -0.92$
  - $dw_2 = 4w_2 = 4 * 1.2 = 4.8$
  - $w_1 = w_1 - \alpha * dw_1 = -4.6 - 0.4 * (-0.92) \approx -4.232$
  - $w_2 = w_2 - \alpha * dw_2 = 1.2 - 0.4 * 4.8 = -0.72$

Optimizing Functions of Two Varibales

- **Exercise1: Gradient Descent**

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

Gradient Descent: $W = W - \alpha * dW$ (1.1)

```python
def df_w(W):
    """
    Thực hiện tính gradient của dw1 và dw2
    Arguments:
    W -- np.array [w1, w2]
    Returns:
    dW -- np.array [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
    """
    ################### YOUR CODE HERE ###################


    dW =
    ######################################################

    return dW
```

FUNCTION df_w(W)
    w1, w2 = W
    dw1 = 0.2*w1
    dw2 = 4*w2
    dW = array([dw1, dw2])
    RETURN dW
ENDFUNCTION

```python
def sgd(W, dW, lr):
    """
    Thực hiện thuật tóa Gradient Descent để update w1 và w2
    Arguments:
    W -- np.array: [w1, w2]
    dW -- np.array: [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
    lr -- float: learning rate
    Returns:
    W -- np.array: [w1, w2] w1 và w2 sau khi đã update
    """
    ################### YOUR CODE HERE ###################


    W =
    ######################################################
    return W
```

FUNCTION sgd(W, dW, lr)
    W = W - lr*dW
    RETURN W
ENDFUNCTION

Optimizing Functions of Two Varibales

- **Exercise1: Gradient Descent**

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

Gradient Descent: $W = W - \alpha * dW$ (1.1)

```python
def train_p1(optimizer, lr, epochs):
    """
    Thực hiện tìm điểm minimum của function (1) dựa vào thuật toán
    được truyền vào từ optimizer
    Arguments:
    optimize : function thực hiện thuật toán optimization cụ thể
    lr -- float: learning rate
    epoch -- int: số lượng lần (epoch) lặp để tìm điểm minimum
    Returns:
    results -- list: list các cặp điểm [w1, w2] sau mỗi epoch (mỗi lần cập nhật)
    """

    # initial point
    W = np.array([-5, -2], dtype=np.float32)
    # list of results
    results = [W]
    ##################### YOUR CODE HERE ####################
    # Tạo vòng lặp theo số lần epochs
    # tìm gradient dW gồm dw1 và dw2
    # dùng thuật toán optimization cập nhật w1 và w2
    # append cặp [w1, w2] vào list results


    ##########################################################
    return results
```

FUNCTION train_p1(optimizer, lr, epochs)
    W = array([-5, 2])
    results = [W]
    FOR e start at 0 **TO** epochs-1
        dW = df_w(W)
        W = optimizer(W, dW, lr)
        results.append(W)
    RETURN results
ENDFUNCTION

Optimizing Functions of Two Varibales

- **Exercise2: Gradient Descent + Momentum**

2. **Gradient Descent + Momentum:** Dựa vào thuật toán Gradient Descent và momentum các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w1 = -5$, $w2 = -2$, $v_1 = 0$, $v_2 = 0$, $\alpha = 0.6$, $\beta = 0.5$:

   (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent + Momentum với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.

   (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent + Momentum với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \qquad (1)$$

Gradient Descent + Momentum:

$$V_t = \beta V_{t-1} + (1-\beta) dW_{t-1} \qquad (2.1)$$

$$W_t = W_{t-1} - \alpha * V_t \qquad (2.2)$$

Optimizing Functions of Two Varibales

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

- **Exercise2: Gradient Descent + Momentum**

Gradient Descent + Momentum:

$$V_t = \beta V_{t-1} + (1-\beta) dW_{t-1} \quad (2.1)$$

$$W_t = W_{t-1} - \alpha * V_t \quad (2.2)$$

- **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
- **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 tìm được ở step 1 (công thức (2.1)).
- **STEP3:** Dùng công thức (2.2) Gradient Descent + Momentum để cập nhật w_1 và w_2. Hoàn thành epoch = 1
- **STEP4:** epoch = 2 ta thực hiện tương tự với STEP1, STEP2 và STEP3 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

- epoch=1:
  - $dw_1 = 0.2w_1 = 0.2 * (-5) = -1$
  - $dw_2 = 4w_2 = 4 * (-2) = -8$
  - $v_1 = \beta * v_1 + (1-\beta) * dw_1 = 0.5 * 0 + (1-0.5) * (-1) = -0.5$
  - $v_2 = \beta * v_2 + (1-\beta) * dw_2 = 0.5 * 0 + (1-0.5) * (-8) = -4$
  - $w_1 = w_1 - \alpha * dw_1 = -5 - 0.6 * (-0.5) = -4.7$
  - $w_2 = w_2 - \alpha * dw_2 = -2 - 0.6 * (-4) = 0.4$

Optimizing Functions of Two Varibales

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

- **Exercise2: Gradient Descent + Momentum**

Gradient Descent + Momentum:

$$V_t = \beta V_{t-1} + (1-\beta) dW_{t-1} \quad (2.1)$$

$$W_t = W_{t-1} - \alpha * V_t \quad (2.2)$$

- **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
- **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 tìm được ở step 1 (công thức (2.1)).
- **STEP3:** Dùng công thức (2.2) Gradient Descent + Momentum để cập nhật w_1 và w_2. Hoàn thành epoch = 1
- **STEP4:** epoch = 2 ta thực hiện tương tự với STEP1, STEP2 và STEP3 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

- epoch=2:
  - $dw_1 = 0.2w_1 = 0.2 * (-4.7) = -0.94$
  - $dw_2 = 4w_2 = 4 * 0.4 = 1.6$
  - $v_1 = \beta * v_1 + (1-\beta) * dw_1 = 0.5 * (-0.5) + (1-0.5) * (-0.94) = -0.72$
  - $v_2 = \beta * v_2 + (1-\beta) * dw_2 = 0.5 * (-4) + (1-0.5) * 1.6 = -1.2$
  - $w_1 = w_1 - \alpha * dw_1 = -4.7 - 0.6 * (-0.72) = -4.268$
  - $w_2 = w_2 - \alpha * dw_2 = 0.4 - 0.6 * (-1.2) = 1.12$

Optimizing Functions of Tw

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

- **Exercise2: Gradient Descent + Momentum**

Gradient Descent + Momentum:

$$V_t = \beta V_{t-1} + (1-\beta) dW_{t-1} \quad (2.1)$$

$$W_t = W_{t-1} - \alpha * V_t \quad (2.2)$$

```python
def df_w(w):
    """
    Thực hiện tính gradient của dw1 và dw2
    Arguments:
    W -- np.array [w1, w2]
    Returns:
    dW -- np.array [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
    """
    ################### YOUR CODE HERE ###################


    dW =
    ######################################################

    return dW
```

FUNCTION df_w(W)
  w1, w2 = W
  dw1 = 0.2*w1
  dw2 = 4*w2
  dW = array([dw1, dw2])
  RETURN dW
ENDFUNCTION

```python
def sgd_momentum(W, dW, lr, V, beta):
    """
    Thực hiện thuật tóan Gradient Descent + Momentum để update w1 và w2
    Arguments:
    W -- np.array: [w1, w2]
    dW -- np.array: [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
    lr -- float: learning rate
    V -- np.array: [v1, v2] Exponentially weighted averages gradients
    beta -- float: hệ số long-range average
    Returns:
    W -- np.array: [w1, w2] w1 và w2 sau khi đã update
    V -- np.array: [v1, v2] Exponentially weighted averages gradients sau khi đã cập nhật
    """
    ################### YOUR CODE HERE ###################

    V =
    W =
    ######################################################
    return W, V
```

FUNCTION sgd_momentum(W, dW, lr, V, beta)
  V = beta*V + (1-beta)*dW
  W = W - lr*V
  RETURN W, V
ENDFUNCTION

Optimizing Functions of Two Varibales

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

- **Exercise2: Gradient Descent + Momentum**

Gradient Descent + Momentum:

$$V_t = \beta V_{t-1} + (1-\beta) dW_{t-1} \quad (2.1)$$

$$W_t = W_{t-1} - \alpha * V_t \quad (2.2)$$

```python
def train_p1(optimizer, lr, epochs):
    """
    Thực hiện tìm điểm minimum của function (1) dựa vào thuật toán
    được truyền vào từ optimizer
    Arguments:
    optimize : function thực hiện thuật toán optimization cụ thể
    lr -- float: learning rate
    epochs -- int: số lượng lần (epoch) lặp để tìm điểm minimum
    Returns:
    results -- list: list các cặp điểm [w1, w2] sau mỗi epoch (mỗi lần cập nhật)
    """
    # initial
    W = np.array([-5, -2], dtype=np.float32)
    V = np.array([0, 0], dtype=np.float32)
    results = [W]
    ##################### YOUR CODE HERE ####################
    # Tạo vòng lặp theo số lần epochs
    # tìm gradient dW gồm dw1 và dw2
    # dùng thuật toán optimization cập nhật w1, w2, v1, v2
    # append cặp [w1, w2] vào list results


    ##########################################################
    return results
```

FUNCTION train_p1(optimizer, lr, epochs)
    W = array([-5, 2])
    V = array([0, 0])
    results = [W]
    FOR e start at 0 **TO** epochs-1
        dW = df_w(W)
        W, V = optimizer(W, dW, lr, V, beta=0.5)
        results.append(W)
    RETURN results
ENDFUNCTION

Optimizing Functions of Two Varibales

- **Exercise3: RMSProp**

3. **RMSProp:** Dựa vào thuật toán RMSProp các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w1 = -5$, $w2 = -2$, $s_1 = 0$, $s_2 = 0$, $\alpha = 0.3$, $\gamma = 0.9$, $\epsilon = 10^{-6}$:

   (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán RMSProp với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.

   (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán RMSProp với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \qquad (1)$$

RMSProp:

$$S_t = \gamma S_{t-1} + (1-\gamma){dW^2}_{t-1} \qquad (3.1)$$

$$W_t = W_{t-1} - \alpha * \frac{dW_t}{\sqrt{S_t + \epsilon}} \qquad (3.2)$$

Optimizing Functions of Two Variables

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

RMSProp:

$$S_t = \gamma S_{t-1} + (1-\gamma) {dW^2}_{t-1} \quad (3.1)$$

$$W_t = W_{t-1} - \alpha * \frac{dW_t}{\sqrt{S_t + \epsilon}} \quad (3.2)$$

- **Exercise3: RMSProp**

  - Bắt đầu với epoch = 1
  - **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
  - **STEP2:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (3.1)).
  - **STEP3:** Dùng công thức (3.2) để cập nhật w_1 và w_2. Hoàn thành epoch = 1
  - **STEP4:** epoch = 2 ta thực hiện tương tự với STEP1, STEP2 và STEP3 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

  - epoch=1:
    - $dw_1 = 0.2w_1 = 0.2 * (-5) = -1$
    - $dw_2 = 4w_2 = 4 * (-2) = -8$
    - $s_1 = \gamma * s_1 + (1-\gamma) * dw_1^2 = 0.9 * 0 + (1-0.9) * (-1)^2 = 0.1$
    - $s_2 = \gamma * s_2 + (1-\gamma) * dw_2^2 = 0.9 * 0 + (1-0.9) * (-8)^2 = 6.4$
    - $w_1 = w_1 - \alpha * \frac{dw_1}{\sqrt{s_1 + \epsilon}} = -5 - 0.3 * \frac{-1}{\sqrt{0.1 + 10^{-6}}} \approx -4.051$
    - $w_2 = w_2 - \alpha * \frac{dw_2}{\sqrt{s_2 + \epsilon}} = -2 - 0.3 * \frac{-8}{\sqrt{6.4 + 10^{-6}}} \approx -1.051$

Optimizing Functions of Two Variables

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

RMSProp:

$$S_t = \gamma S_{t-1} + (1-\gamma) {dW^2}_{t-1} \quad (3.1)$$

$$W_t = W_{t-1} - \alpha * \frac{dW_t}{\sqrt{S_t + \epsilon}} \quad (3.2)$$

- **Exercise3: RMSProp**

  - Bắt đầu với epoch = 1
  - **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
  - **STEP2:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (3.1)).
  - **STEP3:** Dùng công thức (3.2) để cập nhật w_1 và w_2. Hoàn thành epoch = 1
  - **STEP4:** epoch = 2 ta thực hiện tương tự với STEP1, STEP2 và STEP3 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

  - epoch=2:
    - $dw_1 = 0.2w_1 = 0.2 * (-4.051) = -0.8102$
    - $dw_2 = 4w_2 = 4 * (-1.051) = -4.204$
    - $s_1 = \gamma * s_1 + (1-\gamma) * dw_1^2 = 0.9 * 0.1 + (1-0.9) * (-0.8102)^2 \approx 0.156$
    - $s_2 = \gamma * s_2 + (1-\gamma) * dw_2^2 = 0.9 * 6.4 + (1-0.9) * (-4.204)^2 \approx 7.527$
    - $w_1 = w_1 - \alpha * \frac{dw_1}{\sqrt{s_1 + \epsilon}} = -4.051 - 0.3 * \frac{-0.8102}{\sqrt{0.156 + 10^{-6}}} \approx -3.436$
    - $w_2 = w_2 - \alpha * \frac{dw_2}{\sqrt{s_2 + \epsilon}} = -1.051 - 0.3 * \frac{-4.204}{\sqrt{7.527 + 10^{-6}}} \approx -0.591$

Optimizing Functions of Two Variables

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

RMSProp:

$$S_t = \gamma S_{t-1} + (1-\gamma) {dW^2}_{t-1} \quad (3.1)$$

$$W_t = W_{t-1} - \alpha * \frac{dW_t}{\sqrt{S_t + \epsilon}} \quad (3.2)$$

- **Exercise3: RMSProp**

```python
def df_w(w):
    """
    Thực hiện tính gradient của dw1 và dw2
    Arguments:
    W -- np.array [w1, w2]
    Returns:
    dW -- np.array [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
    """
    ################### YOUR CODE HERE ###################


    dW =
    ######################################################

    return dW
```

FUNCTION df_w(W)
    w1, w2 = W
    dw1 = 0.2*w1
    dw2 = 4*w2
    dW = array([dw1, dw2])
    RETURN dW
ENDFUNCTION

```python
def RMSProp(W, dW, lr, S, gamma):
    """
    Thực hiện thuật toán RMSProp để update w1 và w2
    Arguments:
    W -- np.array: [w1, w2]
    dW -- np.array: [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
    lr -- float: learning rate
    S -- np.array: [s1, s2] Exponentially weighted averages bình phương gradients
    gamma -- float: hệ số long-range average
    Returns:
    W -- np.array: [w1, w2] w1 và w2 sau khi đã update
    S -- np.array: [s1, s2] Exponentially weighted averages bình phương gradients
    sau khi đã cập nhật
    """
    epsilon = 1e-6
    ################### YOUR CODE HERE ###################

    S =

    W =
    ######################################################
    return W, S
```

FUNCTION RMSProp(W, dW, lr, S, gamma)
    epsilon = 1e-6
    S = gamma*S + (1-gamma)*dW**2
    adapt_lr = lr/sqrt(S + epsilon)
    W = W - adapt_lr*dW
    RETURN W, S
ENDFUNCTION

Optimizing Functions of Two Variables

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

RMSProp:

$$S_t = \gamma S_{t-1} + (1-\gamma){dW^2}_{t-1} \quad (3.1)$$

$$W_t = W_{t-1} - \alpha * \frac{dW_t}{\sqrt{S_t + \epsilon}} \quad (3.2)$$

- **Exercise3: RMSProp**

```python
def train_p1(optimizer, lr, epochs):
    """
    Thực hiện tìm điểm minimum của function (1) dựa vào thuật toán
    được truyền vào từ optimizer
    Arguments:
    optimize : function thực hiện thuật toán optimization cụ thể
    lr -- float: learning rate
    epochs -- int: số lượng lần (epoch) lặp để tìm điểm minimum
    Returns:
    results -- list: list các cặp điểm [w1, w2] sau mỗi epoch (mỗi lần cập nhật)
    """
    # initial
    W = np.array([-5, -2], dtype=np.float32)
    S = np.array([0, 0], dtype=np.float32)
    results = [W]
    ##################### YOUR CODE HERE ####################
    # Tạo vòng lặp theo số lần epochs
    # tìm gradient dW gồm dw1 và dw2
    # dùng thuật toán optimization cập nhật w1, w2, s1, s2
    # append cặp [w1, w2] vào list results


    ##########################################################
    return results
```

FUNCTION train_p1(optimizer, lr, epochs)
  W = array([-5, 2])
  S = array([0, 0])
  results = [S]
  FOR e start at 0 **TO** epochs-1
    dW = df_w(W)
    W, S = optimizer(W, dW, lr, S, beta=0.9)
    results.append(W)
  RETURN results
ENDFUNCTION

Optimizing Functions of Two Varibales

- **Exercise4: Adam**

4. **Adam:** Dựa vào thuật toán Adam các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w1 = -5$, $w2 = -2$, initial $v_1 = 0$, $v_2 = 0$, $s_1 = 0$, $s_2 = 0$, $\alpha = 0.2$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$:

   (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Adam với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.

   (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Adam với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \qquad (1)$$

Adam:

$$V_t = \beta_1 V_{t-1} + (1-\beta_1) dW_{t-1} \qquad (4.1)$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) dW^2{}_{t-1} \qquad (4.2)$$

$$V_{coor} = \frac{V_t}{1-{\beta_1}^t} \qquad (4.3)$$

$$S_{coor} = \frac{S_t}{1-{\beta_1}^t} \qquad (4.4)$$

$$W_t = W_{t-1} - \alpha * \frac{V_{coor}}{\sqrt{S_{coor}+\epsilon}} \qquad (4.5)$$

Optimizing Functions of Two Varibales

• Exercise4: Adam

- Bắt đầu với epoch = 1
- **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
- **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức ở (4.1)).
- **STEP3:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (4.2)).
- **STEP4:** Thực hiện bias-correction cho V và S để thu được V_coor và S_coor. Sau khi áp dụng bias_correction (4.3) và (4.4) ta sẽ thu được v_{coor1}, v_{coor2}, s_{coor1} và s_{coor2}
- **STEP5:** Dùng công thức (4.5) để cập nhật w_1 và w_2. Hoàn thành epoch = 1
- **STEP6:** epoch = 2 ta thực hiện tương tự với STEP1 - STEP5 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

Adam:

$$V_t = \beta_1 V_{t-1} + (1-\beta_1) dW_{t-1} \quad (4.1)$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) {dW^2}_{t-1} \quad (4.2)$$

$$V_{coor} = \frac{V_t}{1-{\beta_1}^t} \quad (4.3)$$

$$S_{coor} = \frac{S_t}{1-{\beta_1}^t} \quad (4.4)$$

$$W_t = W_{t-1} - \alpha * \frac{V_{coor}}{\sqrt{S_{coor}}+\epsilon} \quad (4.5)$$

- epoch=1:
  - $dw_1 = 0.2w_1 = 0.2 * (-5) = -1$
  - $dw_2 = 4w_2 = 4 * (-2) = -8$
  - $v_1 = \beta_1 * v_1 + (1-\beta_1) * dw_1 = 0.9 * 0 + (1-0.9) * (-1) = -0.1$
  - $v_2 = \beta_1 * v_2 + (1-\beta_1) * dw_2 = 0.9 * 0 + (1-0.9) * (-8) = -0.8$
  - $s_1 = \beta_2 * s_1 + (1-\beta_2) * dw_1^2 = 0.999 * 0 + (1-0.999) * (-1)^2 = 0.001$
  - $s_2 = \beta_2 * s_2 + (1-\beta_2) * dw_2^2 = 0.999 * 0 + (1-0.999) * (-8)^2 = 0.064$

Optimizing Functions of Two Varibales

• Exercise4: Adam

- Bắt đầu với epoch = 1
- **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
- **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức ở (4.1)).
- **STEP3:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (4.2)).
- **STEP4:** Thực hiện bias-correction cho V và S để thu được V_coor và S_coor. Sau khi áp dụng bias_correction (4.3) và (4.4) ta sẽ thu được v_{coor1}, v_{coor2}, s_{coor1} và s_{coor2}
- **STEP5:** Dùng công thức (4.5) để cập nhật w_1 và w_2. Hoàn thành epoch = 1
- **STEP6:** epoch = 2 ta thực hiện tương tự với STEP1 - STEP5 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

Adam:

$$V_t = \beta_1 V_{t-1} + (1-\beta_1) dW_{t-1} \quad (4.1)$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) dW^2{}_{t-1} \quad (4.2)$$

$$V_{coor} = \frac{V_t}{1-{\beta_1}^t} \quad (4.3)$$

$$S_{coor} = \frac{S_t}{1-{\beta_1}^t} \quad (4.4)$$

$$W_t = W_{t-1} - \alpha * \frac{V_{coor}}{\sqrt{S_{coor}}+\epsilon} \quad (4.5)$$

- $v_{corr1} = \dfrac{v_1}{1-\beta_1^t} = \dfrac{-0.1}{1-0.9^1} = -1$
- $v_{corr2} = \dfrac{v_2}{1-\beta_1^t} = \dfrac{-0.8}{1-0.9^1} = -8$
- $s_{corr1} = \dfrac{s_1}{1-\beta_2^t} = \dfrac{0.001}{1-0.999^1} \approx 1$
- $s_{corr2} = \dfrac{s_2}{1-\beta_2^t} = \dfrac{0.064}{1-0.999^1} \approx 64$

- $w_1 = w_1 - \alpha * \dfrac{v_{corr1}}{\sqrt{s_{corr1}+\epsilon}} = -5 - 0.2 * \dfrac{-1}{\sqrt{1}+10^{-6}} \approx -4.8$
- $w_2 = w_2 - \alpha * \dfrac{v_{corr2}}{\sqrt{s_{corr2}+\epsilon}} = -2 - 0.2 * \dfrac{-8}{\sqrt{64}+10^{-6}} \approx -1.8$

Optimizing Functions of Two Varibales

• Exercise4: Adam

- Bắt đầu với epoch = 1
- **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
- **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức ở (4.1)).
- **STEP3:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (4.2)).
- **STEP4:** Thực hiện bias-correction cho V và S để thu được V_coor và S_coor. Sau khi áp dụng bias_correction (4.3) và (4.4) ta sẽ thu được v_{coor1}, v_{coor2}, s_{coor1} và s_{coor2}
- **STEP5:** Dùng công thức (4.5) để cập nhật w_1 và w_2. Hoàn thành epoch = 1
- **STEP6:** epoch = 2 ta thực hiện tương tự với STEP1 - STEP5 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

Adam:

$$V_t = \beta_1 V_{t-1} + (1-\beta_1) dW_{t-1} \quad (4.1)$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) d{W^2}_{t-1} \quad (4.2)$$

$$V_{coor} = \frac{V_t}{1-{\beta_1}^t} \quad (4.3)$$

$$S_{coor} = \frac{S_t}{1-{\beta_1}^t} \quad (4.4)$$

$$W_t = W_{t-1} - \alpha * \frac{V_{coor}}{\sqrt{S_{coor}}+\epsilon} \quad (4.5)$$

- epoch=2:
  - $dw_1 = 0.2w_1 = 0.2 * (-4.8) = -0.96$
  - $dw_2 = 4w_2 = 4 * (-1.8) = -7.2$
  - $v_1 = \beta_1 * v_1 + (1-\beta_1) * dw_1 = 0.9 * (-0.1) + (1-0.9) * (-0.96) = -0.186$
  - $v_2 = \beta_1 * v_2 + (1-\beta_1) * dw_2 = 0.9 * (-0.8) + (1-0.9) * (-7.2) = -1.44$
  - $s_1 = \beta_2 * s_1 + (1-\beta_2) * dw_1^2 = 0.999 * 0.001 + (1-0.999) * (-0.96)^2 \approx 0.0019206$
  - $s_2 = \beta_2 * s_2 + (1-\beta_2) * dw_2^2 = 0.999 * 0.064 + (1-0.999) * (-7.2)^2 \approx 0.115776$

Optimizing Functions of Two Varibales

• Exercise4: Adam

- Bắt đầu với epoch = 1
- **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
- **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức ở (4.1)).
- **STEP3:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (4.2)).
- **STEP4:** Thực hiện bias-correction cho V và S để thu được V_coor và S_coor. Sau khi áp dụng bias_correction (4.3) và (4.4) ta sẽ thu được v_{coor1}, v_{coor2}, s_{coor1} và s_{coor2}
- **STEP5:** Dùng công thức (4.5) để cập nhật w_1 và w_2. Hoàn thành epoch = 1
- **STEP6:** epoch = 2 ta thực hiện tương tự với STEP1 - STEP5 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1

$$f(w_1, w_2) = 0.1{w_1}^2 + 2{w_2}^2 \quad (1)$$

Adam:

$$V_t = \beta_1 V_{t-1} + (1-\beta_1) dW_{t-1} \quad (4.1)$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2) dW^2{}_{t-1} \quad (4.2)$$

$$V_{coor} = \frac{V_t}{1-{\beta_1}^t} \quad (4.3)$$

$$S_{coor} = \frac{S_t}{1-{\beta_1}^t} \quad (4.4)$$

$$W_t = W_{t-1} - \alpha * \frac{V_{coor}}{\sqrt{S_{coor}}+\epsilon} \quad (4.5)$$

$$- \ v_{corr1} = \frac{v_1}{1-\beta_1^t} = \frac{-0.186}{1-0.9^2} \approx -0.9789474$$

$$- \ v_{corr2} = \frac{v_2}{1-\beta_1^t} = \frac{-1.44}{1-0.9^2} \approx -7.5789474$$

$$- \ s_{corr1} = \frac{s_1}{1-\beta_2^t} = \frac{0.0019206}{1-0.999^2} \approx 0.9607804$$

$$- \ s_{corr2} = \frac{s_2}{1-\beta_2^t} = \frac{0.115776}{1-0.999^2} \approx 57.9169585$$

$$- \ w_1 = w_1 - \alpha * \frac{v_{corr1}}{\sqrt{s_{corr1}+\epsilon}} = -4.8 - 0.2 * \frac{-0.9789474}{\sqrt{0.9607804}+10^{-6}} \approx -4.6002546$$

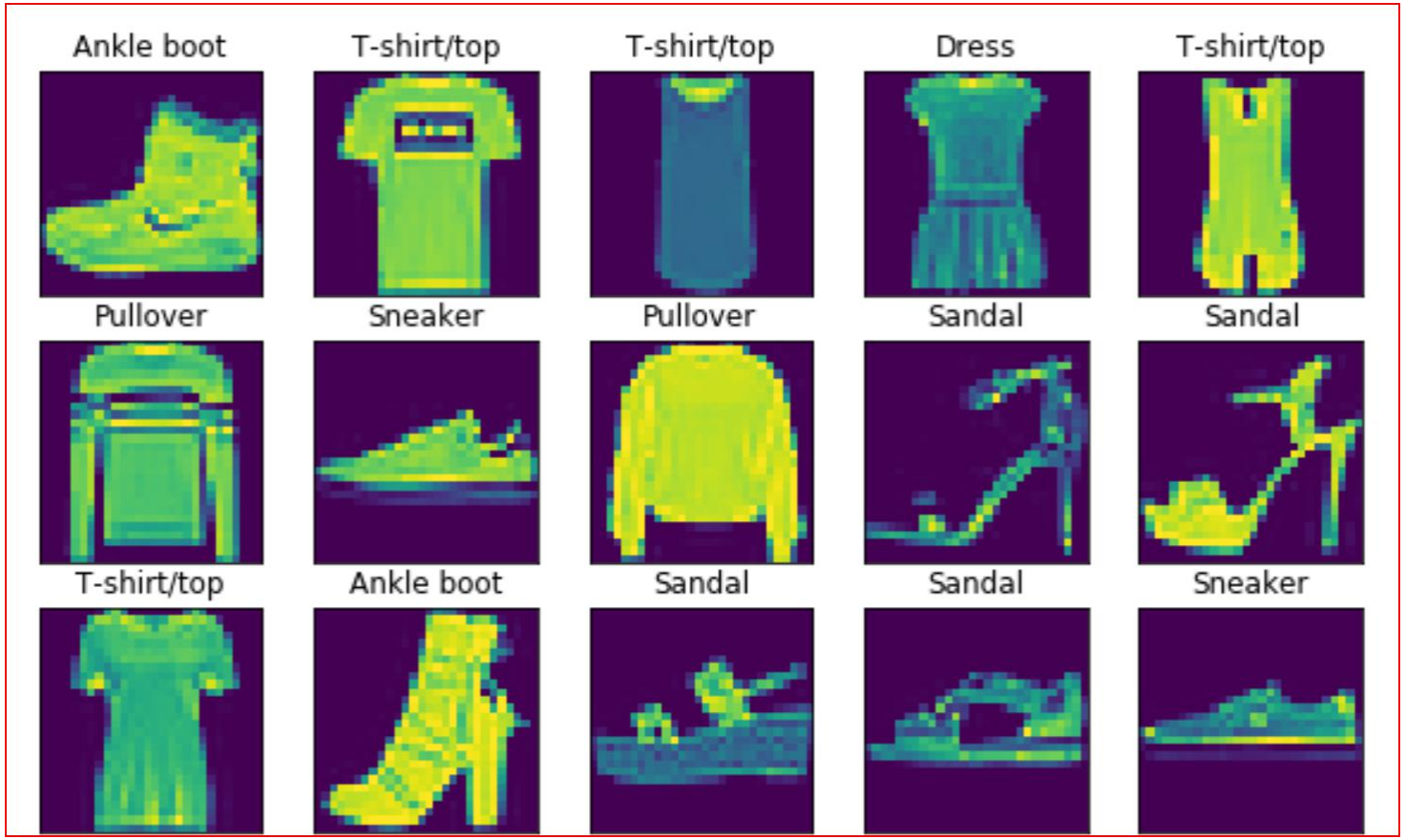
$$- \ w_2 = w_2 - \alpha * \frac{v_{corr2}}{\sqrt{s_{corr2}+\epsilon}} = -1.8 - 0.2 * \frac{-7.5789474}{\sqrt{57.9169585}+10^{-6}} \approx -1.6008245$$

Content

- **Giới Thiệu Optimization Trong Machine Learning**
  - Giới Thiệu Optimization
  - Giới Thiệu Gradient-based Optimization Trong Machine Learning
  - Giới Thiệu Sharpness-Aware Minimization
- **Optimizing Functions of Two Varibales**
  - Giới thiệu vấn đề
  - Exercise1: Gradient Descent
  - Exercise2: Gradient Descent + Momentum
  - Exercise3: RMSProp
  - Exercise4: Adam
- **Vanishing Problem (Optional)**
  - GD, GD + Momentum, RMSProp, và Adam

Vanishing Problem (Optional)

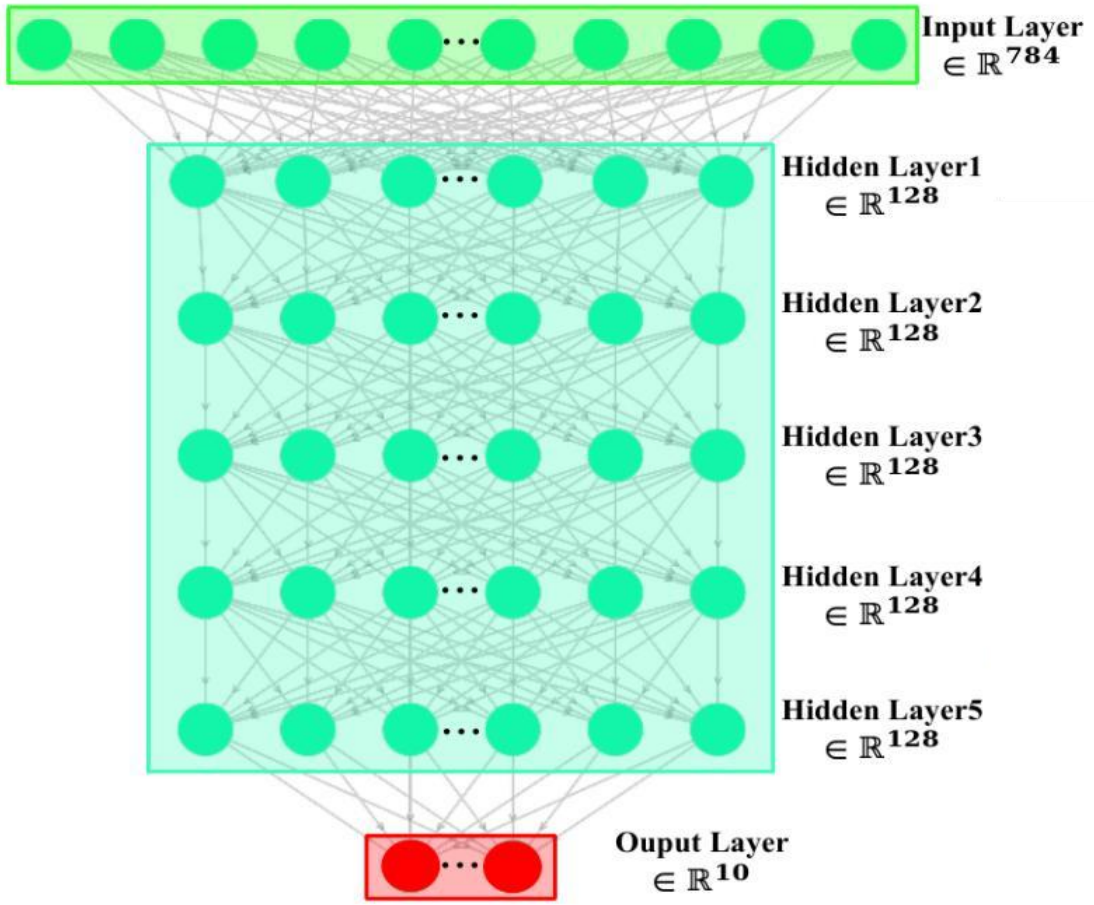
- **Giới thiệu vấn đề**
  - Fashion MNIST dataset
    - **Train**: 60,000 samples
    - **Test**: 10,000 samples
    - **Classes**: 10
    - **Size**: 28x28
    - **Image type**: grayscale



https://github.com/zalandoresearch/fashion-mnist
https://miro.medium.com/max/1838/1*6YhvuUHE0LPHEsqU_Cis9w.png

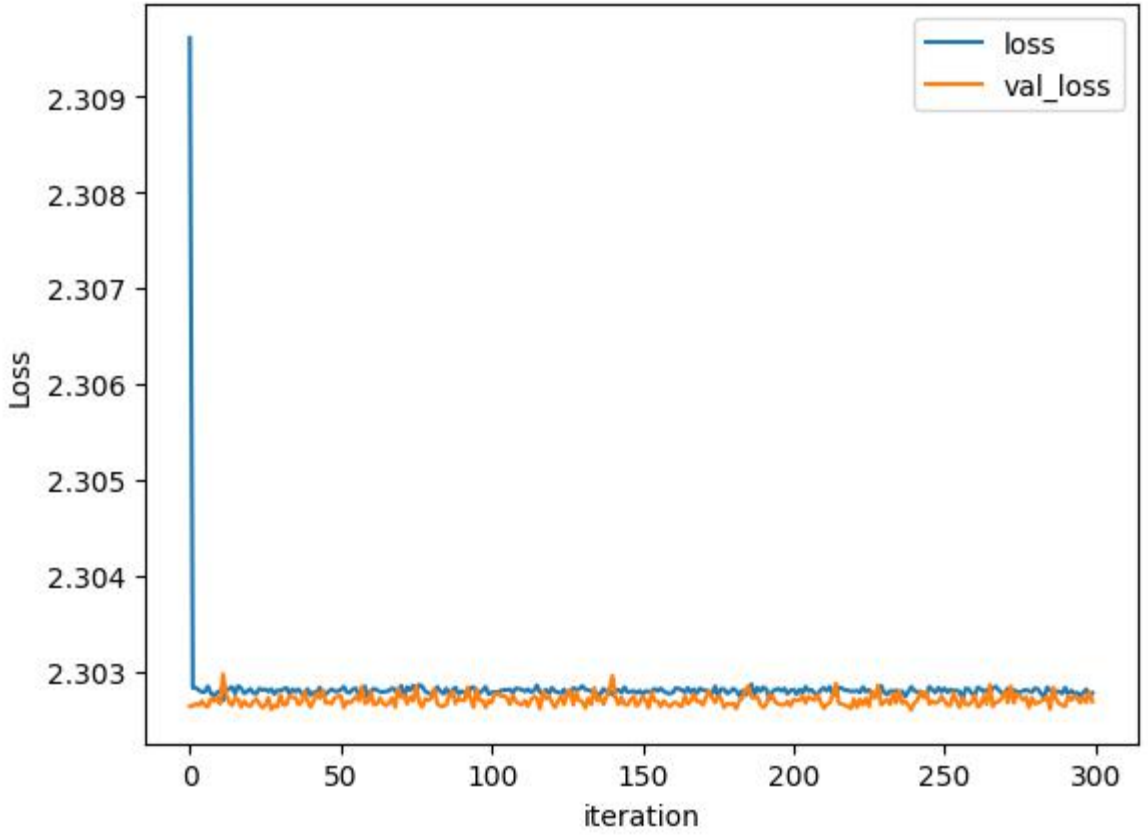
Vanishing Problem (Optional)

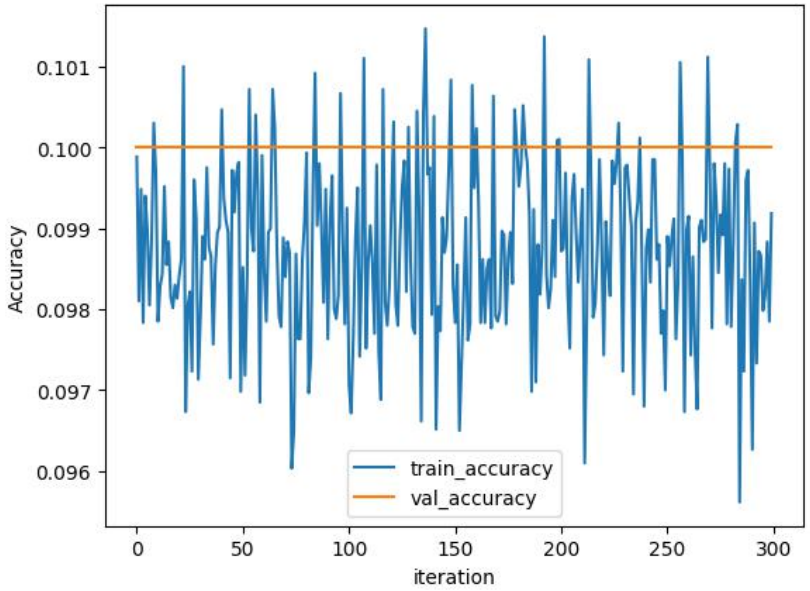
- **Giới thiệu vấn đề**
  - Model:
    - **Hidden Layers**: 5 layers
    - **Activation**: sigmoid
    - **Nodes**: 128
    - **Loss**: CE
    - **LR:** 0.01



Vanishing Problem (Optional)

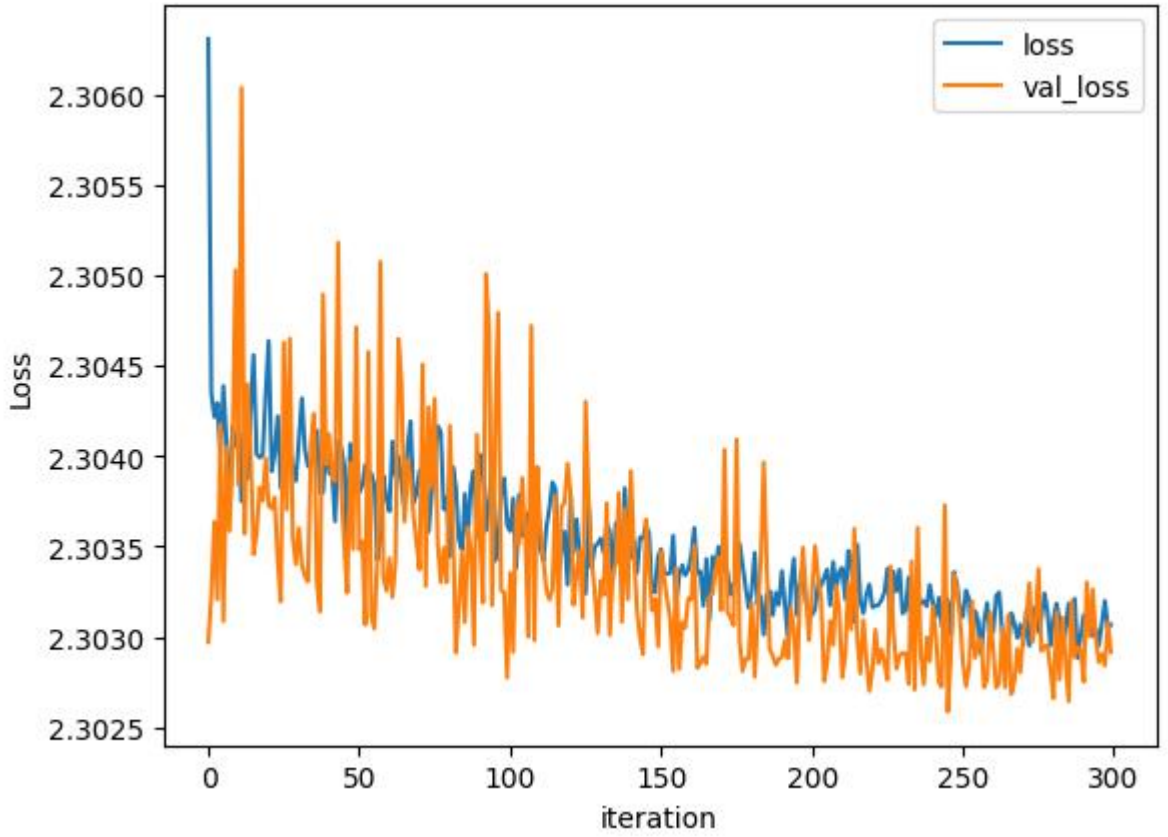
- **GD**
  - Model:
    - **Hidden Layers**: 5 layers
    - **Activation**: sigmoid
    - **Nodes**: 128
    - **Loss**: CE
    - **Optimizer**: sgd

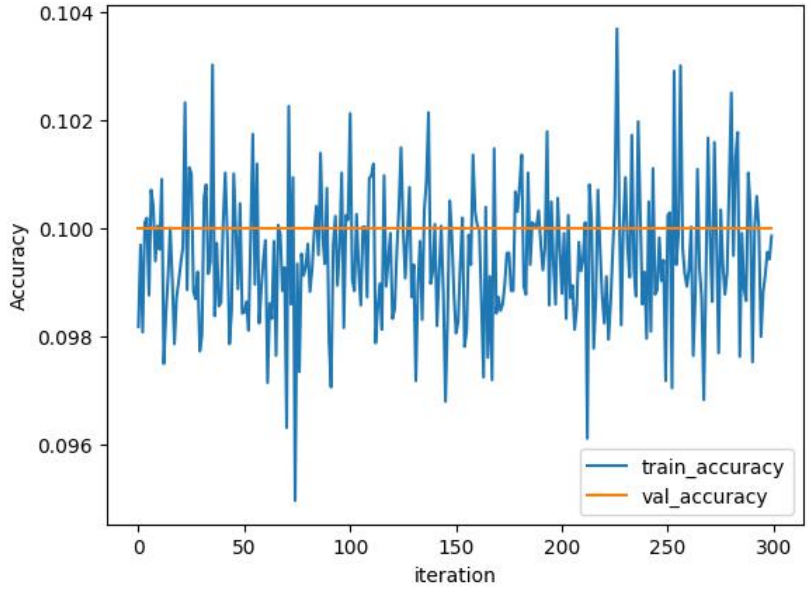




Vanishing Problem (Optional)

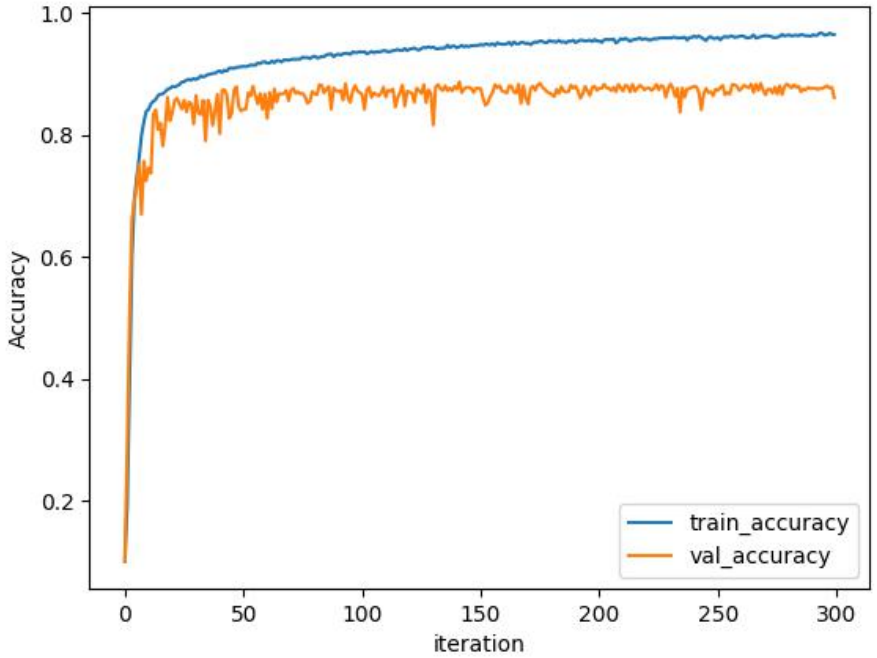
- **GD + Momentum**
  - Model:
    - **Hidden Layers**: 5 layers
    - **Activation**: sigmoid
    - **Nodes**: 128
    - **Loss**: CE
    - **Optimizer**: sgd + momentum (0.9)

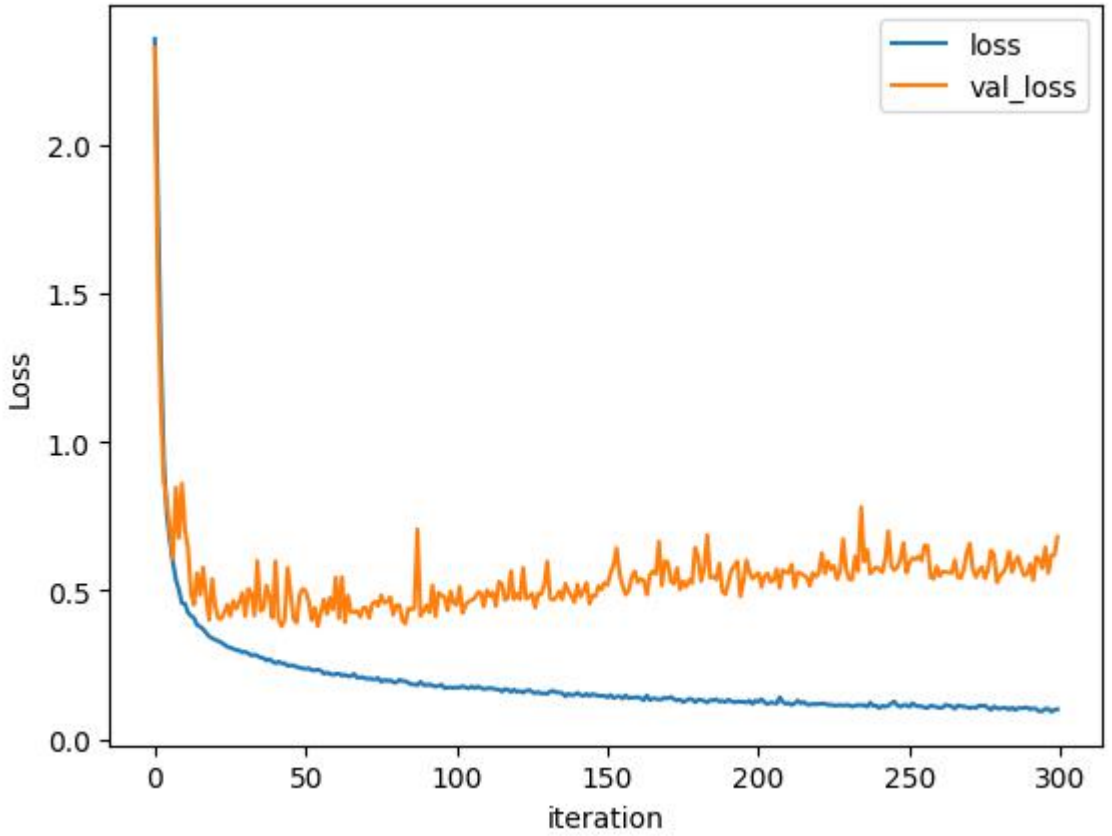




Vanishing Problem (Optional)

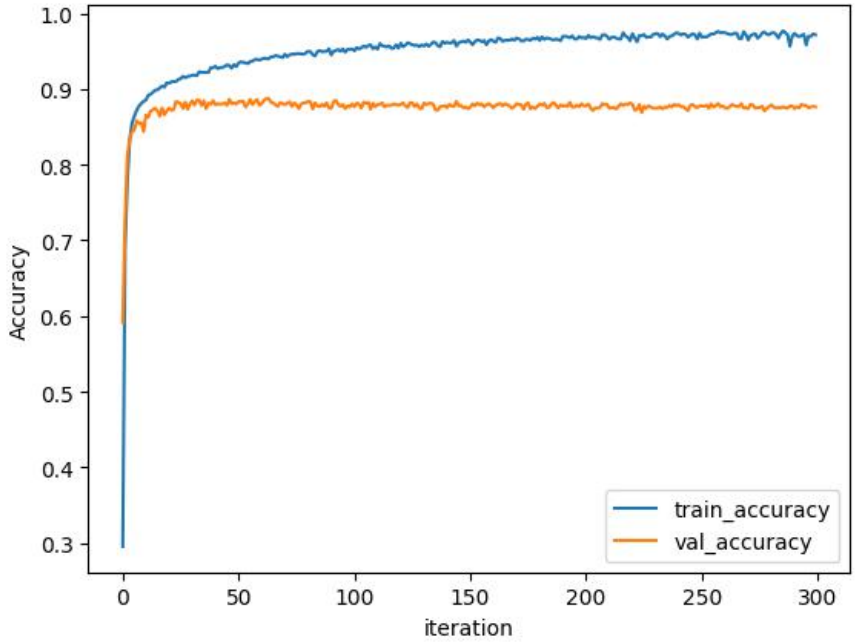
- **RMSProp**
  - Model:
    - **Hidden Layers**: 5 layers
    - **Activation**: sigmoid
    - **Nodes**: 128
    - **Loss**: CE
    - **Optimizer**: RMSProp

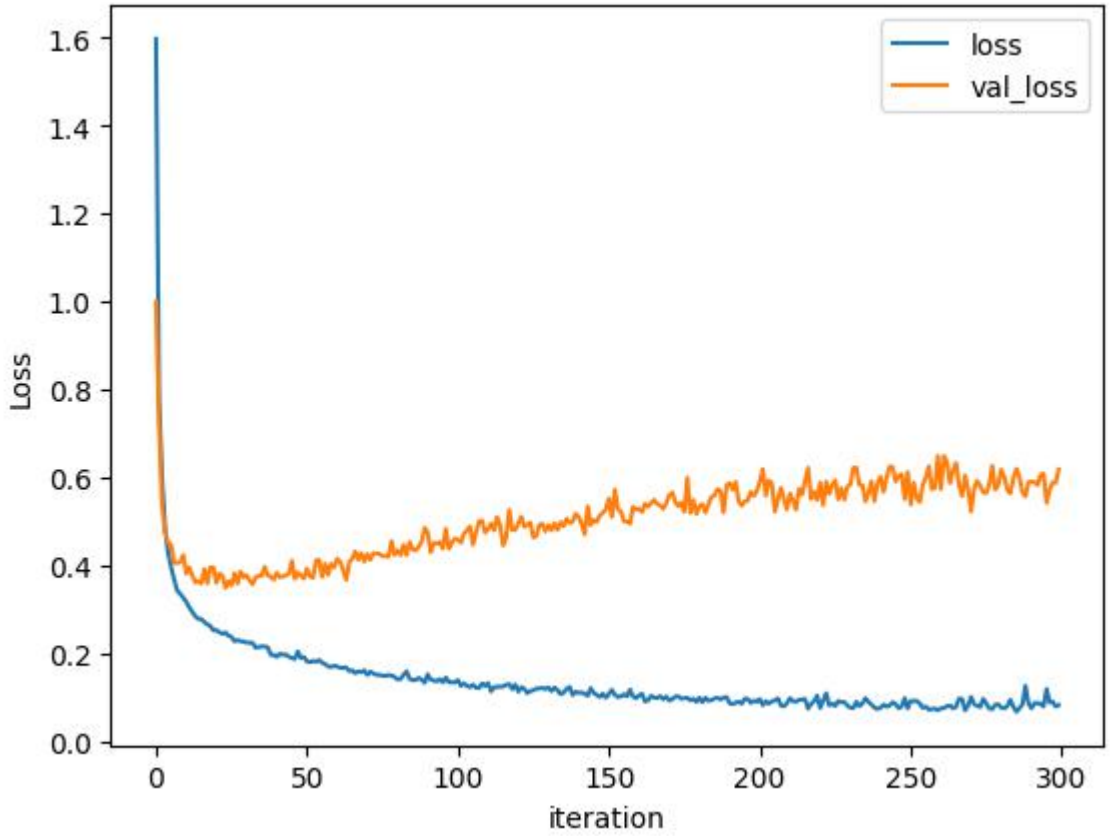




Vanishing Problem (Optional)

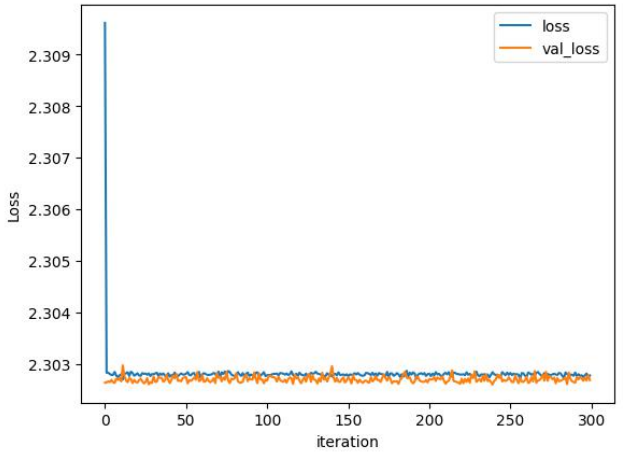
- **Adam**
  - Model:
    - **Hidden Layers**: 5 layers
    - **Activation**: sigmoid
    - **Nodes**: 128
    - **Loss**: CE
    - **Optimizer**: Adam $\beta_1 = 0.9, \beta_2 = 0.999$

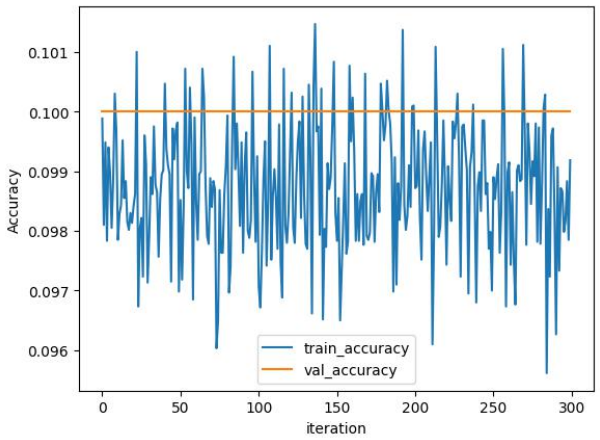




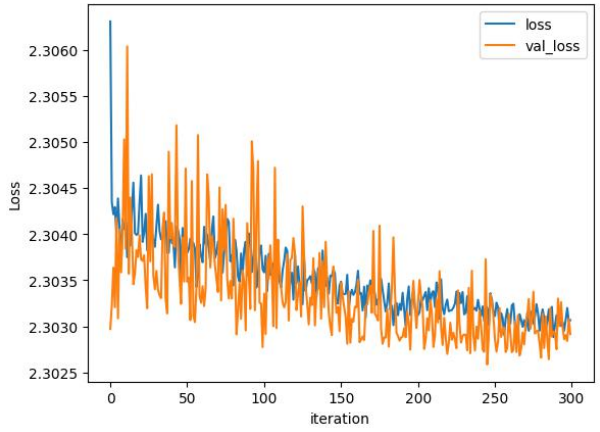
Vanishing Problem (Optional)

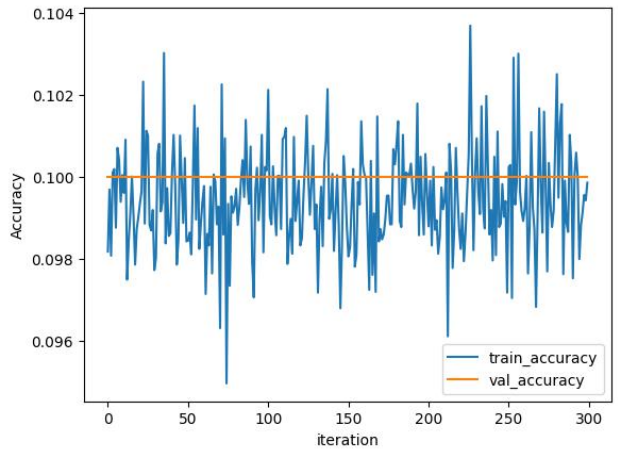
GD



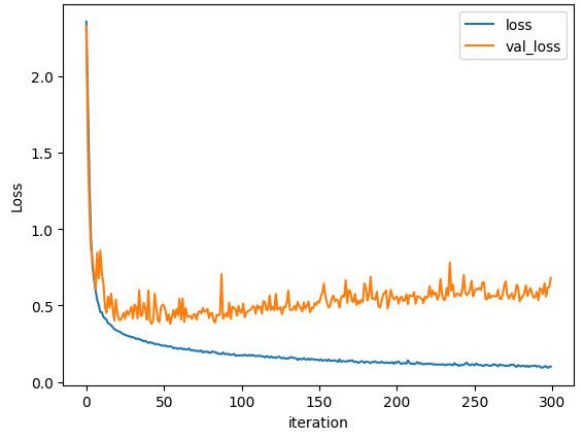


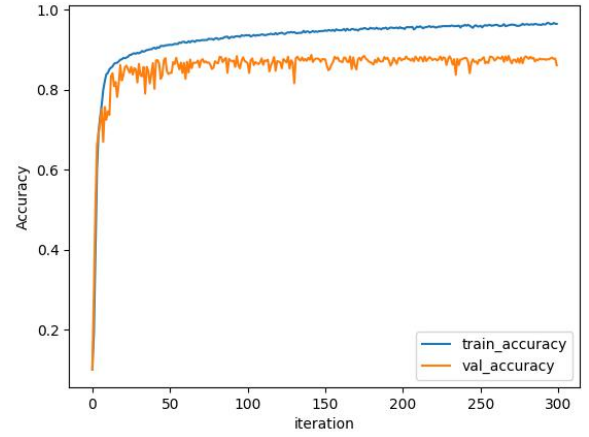
GD + Momentum



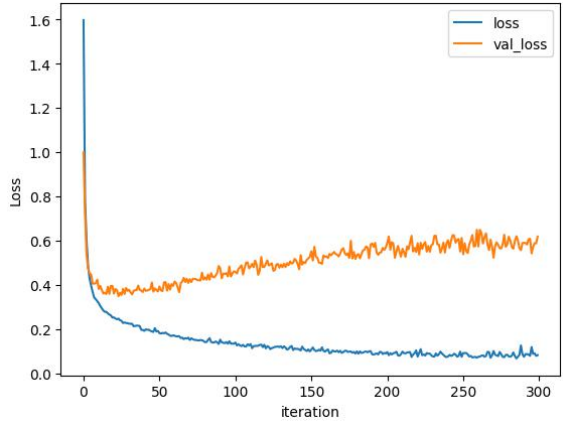


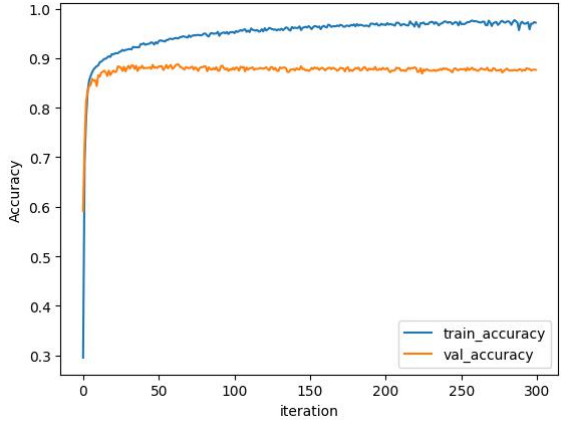
RMSProp





Adam

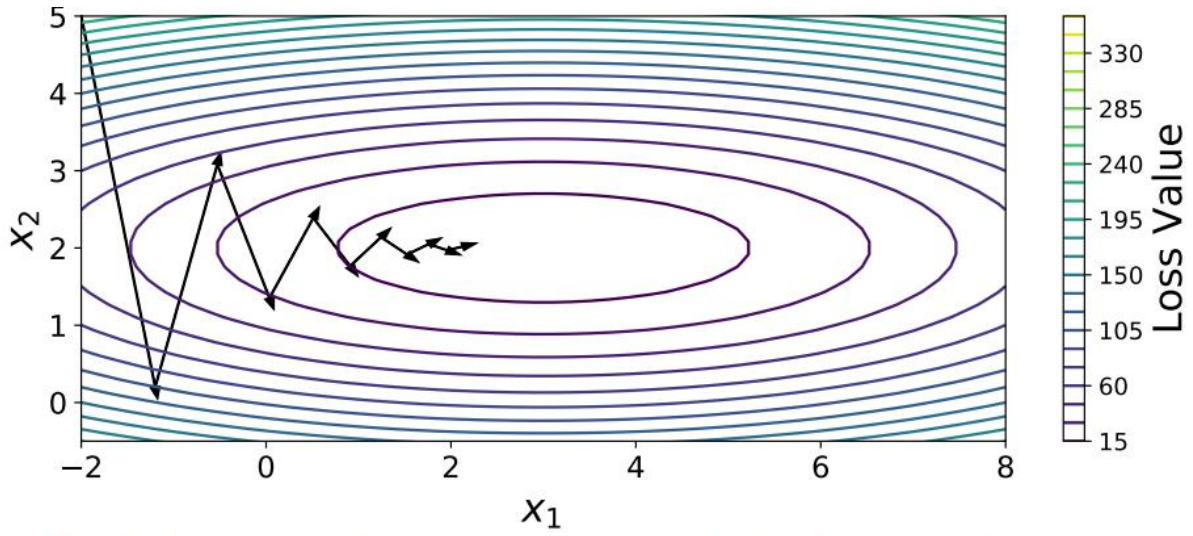




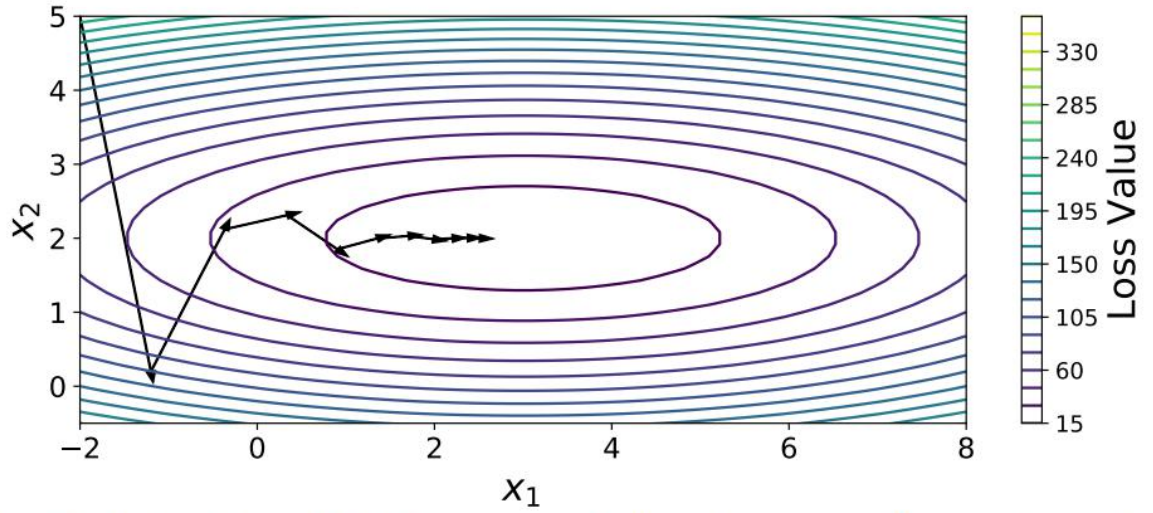
13:53:27

Other Research Papers

AdaSmooth: An Adaptive Learning Rate Method based on Effective Ratio



(a) Optimization without Momentum. A higher learning rate may result in larger parameter updates in dimension across the valley (direction of x_2) which could lead to oscillations back as forth across the valley.



(b) Optimization with Momentum. Though the gradients along the valley (direction of x_1) are much smaller than the gradients across the valley (direction of x_2), they are typically in the same direction and thus the momentum term accumulates to speed up movement, dampen oscillations and cause us to barrel through narrow valleys, small humps and (local) minima.

Figure 1. A 2-dimensional convex function $L(\boldsymbol{x}) = 2(x_1 - 3)^2 + 20(x_2 - 2)^2 + 5$ and $\frac{\partial L(\boldsymbol{x})}{\partial \boldsymbol{x}} = (4x_1 - 12, 8x_2 - 16)^\top$. Starting point to descent is $(-2, 5)^\top$.

Other Research Papers

AdaSmooth: An Adaptive Learning Rate Method based on Effective Ratio

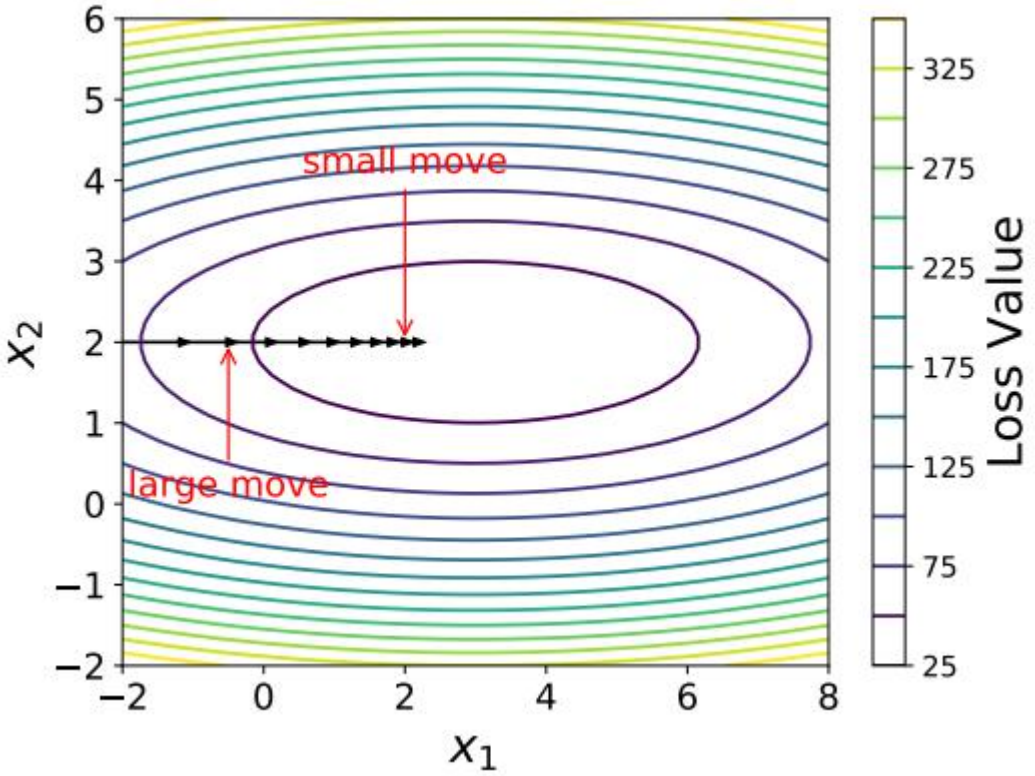


Figure 2. Demonstration of how the effective ratio works. Stochastic optimization tends to move a large step when it is far from the (local) minima; and a relatively small step when it is close to the (local) minima.

Effective Ratio (ER)

$$e_t = \frac{s_t}{n_t} = \frac{h_t - h_{t-M}}{\sum_{i=0}^{M-1} |h_{t-i} - h_{t-1-i}|} = \frac{\text{Total move for a period}}{\text{Sum of absolute move for each bar}}, \tag{14}$$

$$\boldsymbol{e}_t = \frac{\boldsymbol{s}_t}{\boldsymbol{n}_t} = \frac{|\boldsymbol{x}_t - \boldsymbol{x}_{t-M}|}{\sum_{i=0}^{M-1} |\boldsymbol{x}_{t-i} - \boldsymbol{x}_{t-1-i}|} = \frac{|\sum_{i=0}^{M-1} \Delta \boldsymbol{x}_{t-1-i}|}{\sum_{i=0}^{M-1} |\Delta \boldsymbol{x}_{t-1-i}|}, \tag{15}$$

Other Research Papers

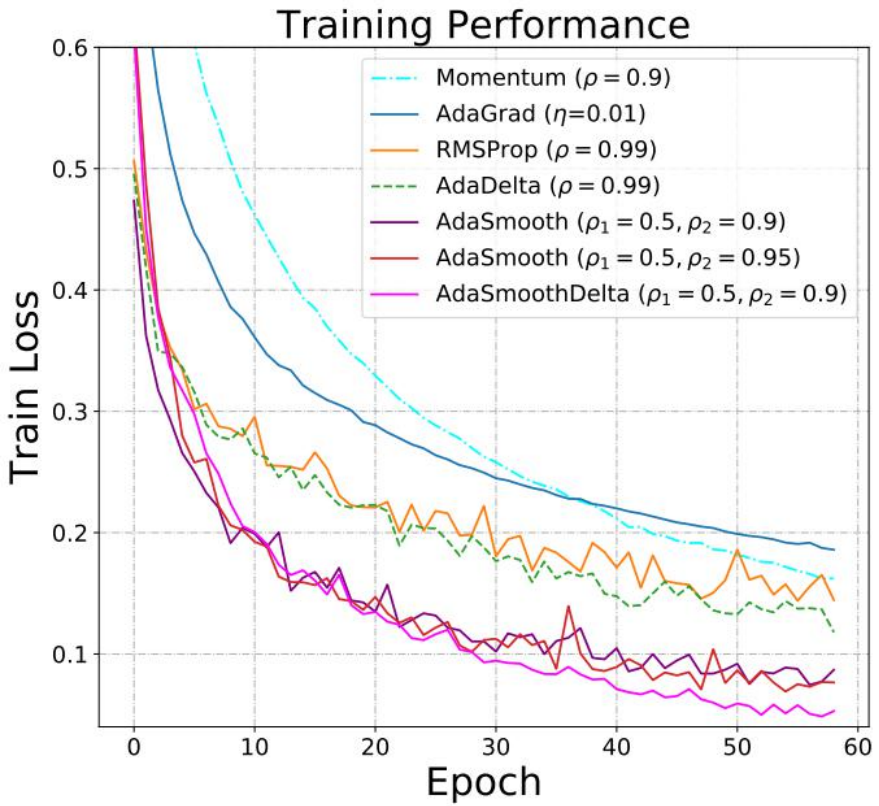
AdaSmooth: An Adaptive Learning Rate Method based on Effective Ratio

Algorithm 1 Computing AdaSmooth at iteration t: the proposed AdaSmooth algorithm. All operations on vectors are element-wise. Good default settings for the tested tasks are $\rho_1 = 0.5, \rho_2 = 0.99, \epsilon = 1e{-}6, \eta = 0.001$; see Section 3.2 or Eq (8) for a detailed discussion on the explanation of the decay constants' default values. Empirical study in Section 4 shows that the AdaSmooth algorithm is not sensitive to the hypermarater ρ_2, while $\rho_1 = 0.5$ is relatively a lower bound in this setting. The AdaSmoothDelta iteration can be calculated in a similar way.

Input: initial parameter $\boldsymbol{x}_1$, Constant ϵ;
Input: global learning rate η, by default $\eta = 0.001$;
Input: fast decay constant ρ_1, slow decay constant ρ_2;
Input: assert $\rho_2 > \rho_1$, by default $\rho_1 = 0.5$, $\rho_2 = 0.99$;
for $t = 1 : T$ **do**
  Compute gradient $\boldsymbol{g}_t = \nabla L(\boldsymbol{x}_t)$;
  Compute ER $\boldsymbol{e}_t = \frac{|\boldsymbol{x}_t - \boldsymbol{x}_{t-M}|}{\sum_{i=0}^{M-1} |\Delta \boldsymbol{x}_{t-1-i}|}$;
  Compute smoothing $\boldsymbol{c}_t = (\rho_2 - \rho_1) \times \boldsymbol{e}_t + (1 - \rho_2)$;
  Compute normalization term:

$$E[\boldsymbol{g}^2]_t = \boldsymbol{c}_t^2 \odot \boldsymbol{g}_t^2 + \left(1 - \boldsymbol{c}_t^2\right) \odot E[\boldsymbol{g}^2]_{t-1};$$

  Compute step $\Delta \boldsymbol{x}_t = -\frac{\eta}{\sqrt{E[\boldsymbol{g}^2]_t + \epsilon}} \odot \boldsymbol{g}_t$;
  Apply update $\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \Delta \boldsymbol{x}_t$;
end for
Return: resulting parameters $\boldsymbol{x}_t$, and the loss $L(\boldsymbol{x}_t)$.



(a) MNIST training Loss

Other Research Papers

AdaSmooth: An Adaptive Learning Rate Method based on Effective Ratio

- **Background**:
  - Traditional optimization methods require tedious manual tuning of hyper-parameters.
  - AdaSmooth introduces a novel per-dimension learning rate method for gradient descent.
- **Key Feature**:
  - Insensitivity to hyper-parameters, eliminating the need for manual tuning akin to Momentum, AdaGrad, and **AdaDelta methods**.
- **Objective**:
  - To increase optimization efficiency, out-of-sample accuracy, and reduce memory requirements.
- **Advantages**:
  - No Manual Tuning: Simplifies the optimization process.
  - Increased Efficiency: Promising results in optimization compared to other methods.
  - Memory Efficiency: Designed for handling large-scale machine learning tasks with lesser memory requirements.
  - Combines advantages of AdaGrad, RMSProp, and AdaDelta for a robust optimization approach.

Other Research Papers

Other Research Papers

Gradients without Backpropagation

Forward gradients are unbiased estimators of the gradient $\nabla f(\theta)$ for a function $f : \mathbb{R}^n \to \mathbb{R}$, given by $g(\theta) = \langle \nabla f(\theta), v \rangle v$.

Here $v = (v_1, \ldots, v_n)$ is a random vector, which must satisfy the following conditions in order for $g(\theta)$ to be an unbiased estimator of $\nabla f(\theta)$

- $v_i \perp v_j$ for all $i \neq j$
- $\mathbb{E}[v_i] = 0$ for all i
- $\mathbb{V}[v_i] = 1$ for all i

3.1. Forward Gradients

Definition 1. *Given a function $f : \mathbb{R}^n \to \mathbb{R}$, we define the "forward gradient" $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}^n$ as*

$$\boldsymbol{g}(\boldsymbol{\theta}) = (\nabla f(\boldsymbol{\theta}) \cdot \boldsymbol{v})\, \boldsymbol{v} \,, \tag{1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^n$ is the point at which we are evaluating the gradient, $\boldsymbol{v} \in \mathbb{R}^n$ is a perturbation vector taken as a multivariate random variable $\boldsymbol{v} \sim p(\boldsymbol{v})$ such that $\boldsymbol{v}$'s scalar components v_i are independent and have zero mean and unit variance for all i, and $\nabla f(\boldsymbol{\theta}) \cdot \boldsymbol{v} \in \mathbb{R}$ is the directional derivative of f at point $\boldsymbol{\theta}$ in direction $\boldsymbol{v}$.

Other Research Papers

Gradients without Backpropagation

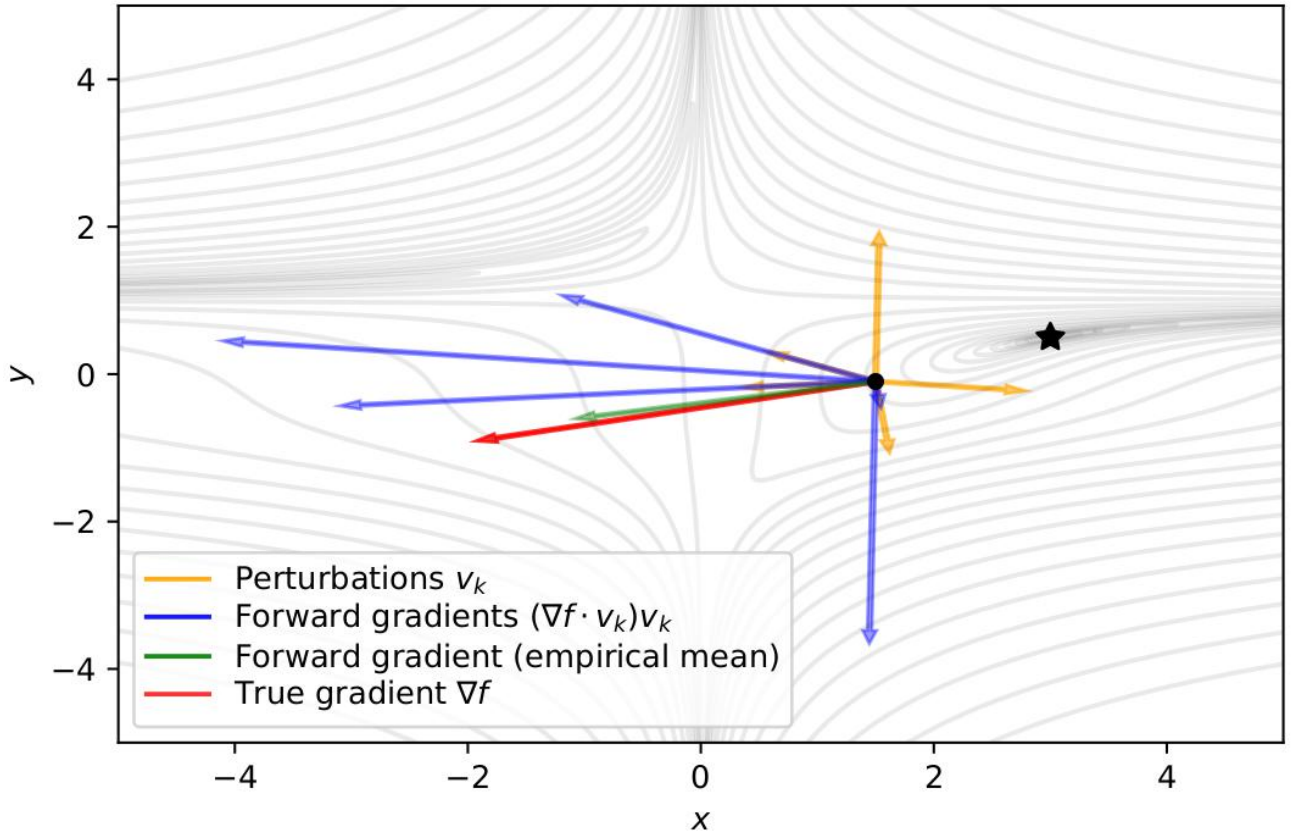
Algorithm 1 Forward gradient descent (FGD)

Require: η: learning rate
Require: f: objective function
Require: $\boldsymbol{\theta}_0$: initial parameter vector

$t \leftarrow 0$ ▷ Initialize
while $\boldsymbol{\theta}_t$ not converged **do**
$t \leftarrow t + 1$
$\boldsymbol{v}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ ▷ Sample perturbation

Note: *the following computes* f_t *and* d_t *simultaneously and without having to compute* ∇f *in the process*
$f_t,\ d_t \leftarrow f(\boldsymbol{\theta}_t),\ \nabla f(\boldsymbol{\theta}_t) \cdot \boldsymbol{v}$ ▷ Forward AD (Section 3.1)

$\boldsymbol{g}_t \leftarrow \boldsymbol{v}_t d_t$ ▷ Forward gradient
$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\, \boldsymbol{g}_t$ ▷ Parameter update
end while
return $\boldsymbol{\theta}_t$



Other Research Papers

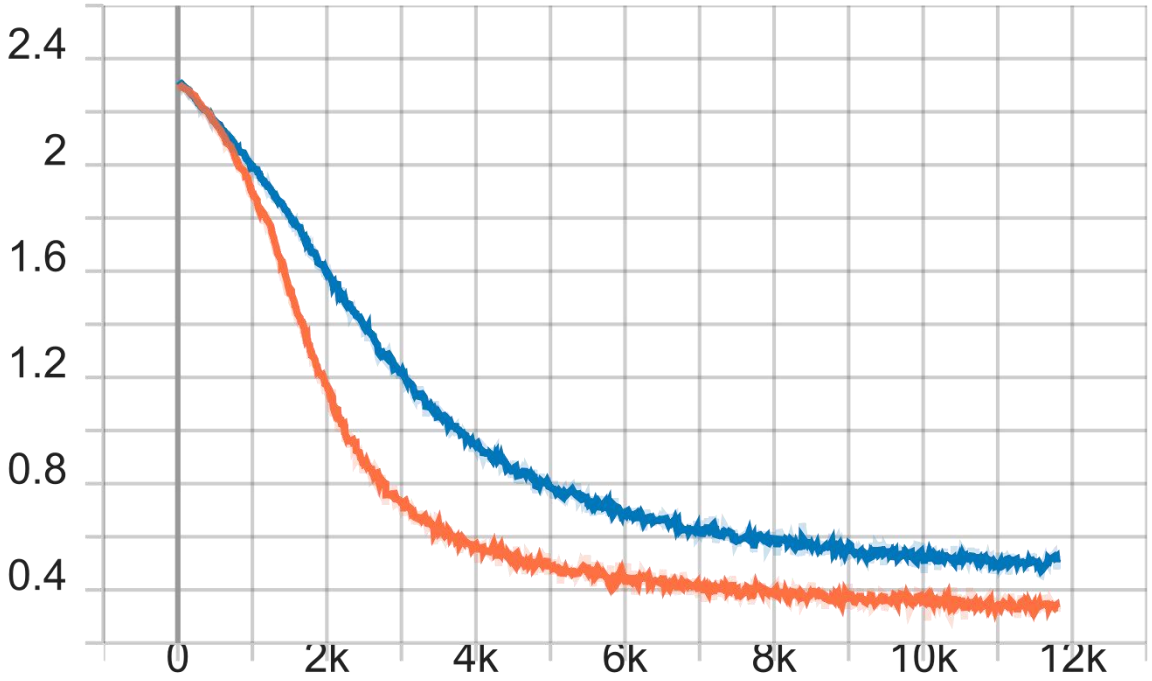
Gradients without Backpropagation

Algorithm 1 Forward gradient descent (FGD)

Require: η: learning rate
Require: f: objective function
Require: $\boldsymbol{\theta}_0$: initial parameter vector
$t \leftarrow 0$ ▷ Initialize
while $\boldsymbol{\theta}_t$ not converged **do**
$t \leftarrow t + 1$
$\boldsymbol{v}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ ▷ Sample perturbation

Note: *the following computes* f_t *and* d_t *simultaneously and without having to compute* ∇f *in the process*
$f_t,\ d_t \leftarrow f(\boldsymbol{\theta}_t),\ \nabla f(\boldsymbol{\theta}_t) \cdot \boldsymbol{v}$ ▷ Forward AD (Section 3.1)

$\boldsymbol{g}_t \leftarrow \boldsymbol{v}_t d_t$ ▷ Forward gradient
$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta\, \boldsymbol{g}_t$ ▷ Parameter update
end while
return $\boldsymbol{\theta}_t$



The accuracies on the test set are:

	Test accuracy
Backprop	0.8839
Fwdgrad	0.9091

Other Research Papers

Gradients without Backpropagation

- **Background**:
  - Traditionally, backpropagation is used to compute gradients in machine learning.
  - These gradients are crucial for optimizing a model's parameters to minimize loss.
- **Problem**:
  - Backpropagation can be computationally intensive, impacting the efficiency of training pipelines.
- **Solution**:
  - The paper introduces a novel method called "**forward gradient**," eliminating the need for backpropagation.
- **Advantages**:
  - Substantial savings in computation, with training being up to twice as fast in some cases.
  - Potential to reduce time and energy costs in ML training pipelines.
  - Opens discussions on ML hardware design and the biological plausibility of backpropagation.