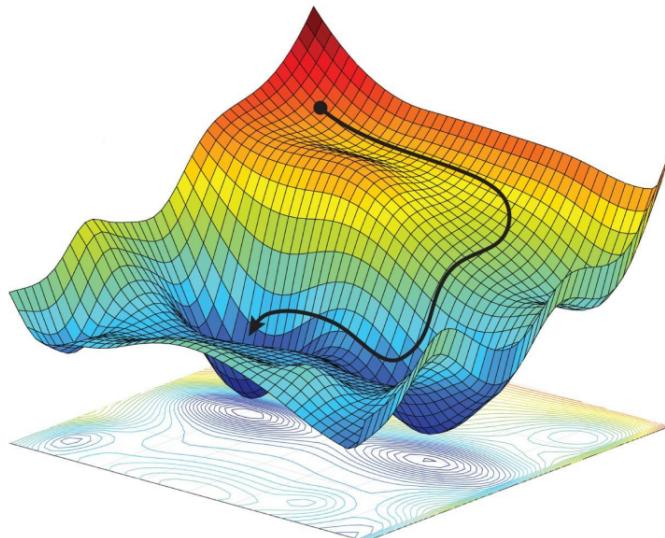
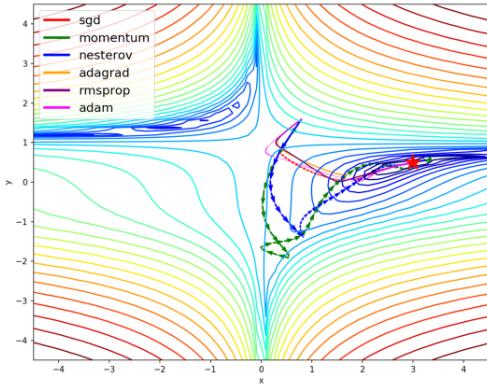
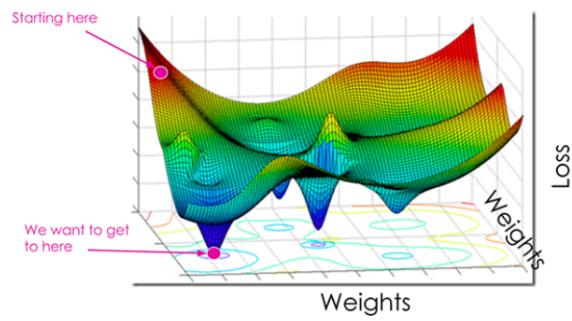
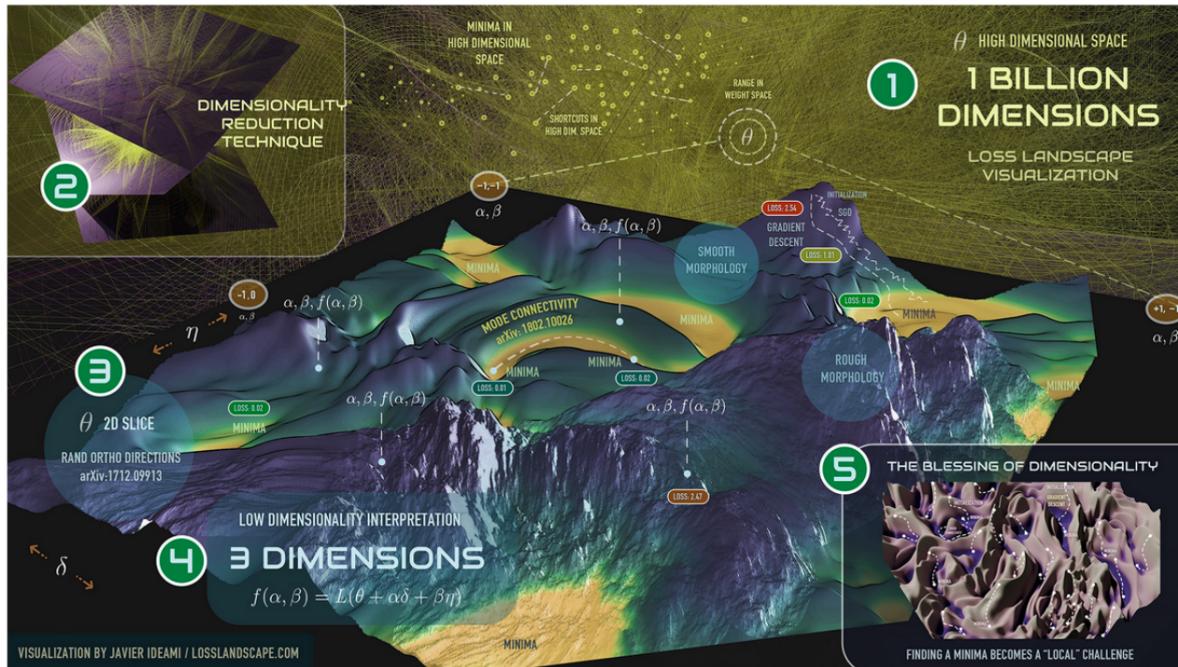


## Optimization Algorithms in Deep Learning - Exercise

Ngày 21 tháng 10 năm 2023

**Phần I: Mô Tả Lý Thuyết:**

Bài tập tuần này sẽ xoay quanh việc thực hiện các thuật toán optimization cơ bản trong deep learning. Ngoài ra bài tập tuần này cũng giới thiệu một vài nhánh optimization trong deep learning.



Bài tuần này chúng ta sẽ làm quen với 4 thuật toán optimization phổ biến trong deep learning:

**1. Gradient Descent:** Là thuật toán đi tìm điểm minimum của một function dựa trên tính toán gradient của function đó. Function này phải thỏa mãn điều kiện là liên tục và có thể đạo hàm được phần lớn ở mọi nơi. Phương pháp này ta sẽ chọn một điểm xuất phát ngẫu nhiên (1 điểm bất kỳ trên function) sau đó dựa vào độ dốc và hướng của gradient để đi ngược hướng và tiến về nơi có vị trí thấp hơn, việc này được lặp đi lặp lại cho đến khi tìm được điểm minimum hoặc thỏa mãn điều kiện dừng

---

**Algorithm 8.1** Stochastic gradient descent (SGD) update

---

**Require:** Learning rate schedule  $\epsilon_1, \epsilon_2, \dots$

**Require:** Initial parameter  $\theta$

$k \leftarrow 1$

**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient estimate:  $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Apply update:  $\theta \leftarrow \theta - \epsilon_k \hat{\mathbf{g}}$

$k \leftarrow k + 1$

**end while**

---

**2. Gradient Descent + Momentum:** Phương pháp này khắc phục điểm yếu của Gradient Descent là không thể vượt qua local minimum để tiến về điểm local minimum khác thấp hơn. Gradient Descent + Momentum sử dụng exponentially weighted average gradient để tích lũy gradient từ trước đó và kết hợp với gradient hiện tại giúp tạo ra một giá trị đủ lớn vượt qua local minimum hiện tại và tiến về local minimum tốt hơn. Nó hoạt động tương tự như viên bi có khi lăn xuống kết hợp thêm đà (momentum) để vượt qua dốc và rơi vào nơi tốt hơn.

$$\begin{aligned} V_t &= \beta V_{t-1} + (1 - \beta)dW_t \\ W_t &= W_t - \alpha * V_t \end{aligned}$$

### 3. RMSProp:

---

**Algorithm 8.5** The RMSProp algorithm

---

**Require:** Global learning rate  $\epsilon$ , decay rate  $\rho$

**Require:** Initial parameter  $\theta$

**Require:** Small constant  $\delta$ , usually  $10^{-6}$ , used to stabilize division by small numbers

Initialize accumulation variables  $r = 0$

**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ .

    Accumulate squared gradient:  $r \leftarrow \rho r + (1 - \rho)\mathbf{g} \odot \mathbf{g}$ .

    Compute parameter update:  $\Delta\theta = -\frac{\epsilon}{\sqrt{\delta+r}} \odot \mathbf{g}$ . ( $\frac{1}{\sqrt{\delta+r}}$  applied element-wise)

    Apply update:  $\theta \leftarrow \theta + \Delta\theta$ .

**end while**

---

Đây là thuật toán mở rộng từ AdaGrad và cũng sử dụng gradient từ trước để tạo ra một thuật toán có thể adaptive learning rate thay vì learning rate cố định như các thuật toán trước. Ý tưởng chính của RMSProp là sử dụng exponentially weighted average bình phương gradient và chia gradient với căn bậc hai của average này.

4. **Adam:** Có thể xem Adam là thuật toán sử dụng kết hợp ý tưởng của Momentum và RMSProp vì Adam sử dụng exponentially weighted average gradient và exponentially weighted average bình phương gradient. Cách thức Adam hoạt động theo việc thừa hưởng điểm mạnh của momentum và adaptive learning rate, momentum giúp vượt qua các local minimum để tiến đến các local minimum tốt hơn, trong khi adaptive learning rate giúp cho việc hội tụ nhanh hơn thay vì phải mất một khoảng thời gian để dao động xung quanh điểm hội tụ.

---

**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  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

---

**Require:**  $\alpha$ : Stepsize  
**Require:**  $\beta_1, \beta_2 \in [0, 1]$ : Exponential decay rates for the moment estimates  
**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$   
**Require:**  $\theta_0$ : 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  $\theta_t$  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)
     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)
     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)
     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)
end while
return  $\theta_t$  (Resulting parameters)

```

---

## Phần II: Bài Tập và Gợi Ý:

### 1) Gợi Ý Các Bước Làm Bài:

**1. Giới thiệu sơ lược về bài tập:** Bài tập tuần này sẽ có gồm có các dạng chính như tính toán và trình bày những bước trong các thuật toán optimization, thực hiện code bằng numpy cho các thuật toán optimization. Ngoài ra các bài tập dạng optional sẽ thực hiện thay đổi các thuật toán optimization trên Pytorch và quan sát hiện tượng vanishing được khắc phục qua các thuật toán (Chi tiết xem phần [Yêu cầu bài tập](#))

Các công thức và gợi ý dưới đây được rút gọn và làm đơn giản hơn để các bạn thuận tiện làm bài. Bài tập 1,2,3 và 4 sẽ optimize cho function gồm 2 biến như sau :

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

#### 2. Bài 1 Gradient Descent:

$$W = W - \alpha * dW \quad (1.1)$$

(a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent (tìm  $w_1$  và  $w_2$  sau 2 epoch) với epoch = 2.

- 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:** Dùng công thức Gradient Descent (1.1) để cập nhật  $w_1$  và  $w_2$ . Hoàn thành epoch = 1
- **STEP3:** epoch = 2 ta thực hiện tương tự với STEP1 và STEP2 như trên với  $w_1$  và  $w_2$  đã được cập nhật từ epoch = 1

(b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent (tìm  $w_1$  và  $w_2$ ) với epoch = 30

```

1 def df_w(W):
2     """
3     Thực hiện tính gradient của dw1 và dw2
4     Arguments:
5     W -- np.array [w1, w2]
6     Returns:
7     dW -- np.array [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
8     """
9     ##### YOUR CODE HERE #####
10
11
12     dW =
13     #####
14
15     return dW

1 def sgd(W, dW, lr):
2     """
3     Thực hiện thuật toán Gradient Descent để update w1 và w2
4     Arguments:
5     W -- np.array: [w1, w2]
6     dW -- np.array: [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
7     lr -- float: learning rate
8     Returns:
9     W -- np.array: [w1, w2] w1 và w2 sau khi đã update
10    """
11   ##### YOUR CODE HERE #####
12
13
14     W =
15     #####
16     return W

```

Hình 1: Function thực hiện tính gradient và function thực hiện thuật toán GD

```

1 def train_p1(optimizer, lr, epochs):
2     """
3     Thực hiện tìm điểm minimum của function (1) dựa vào thuật toán
4     được truyền vào từ optimizer
5     Arguments:
6     optimize : function thực hiện thuật toán optimization cụ thể
7     lr -- float: learning rate
8     epoch -- int: số lượng lần (epoch) lặp để tìm điểm minimum
9     Returns:
10    results -- list: list các cặp điểm [w1, w2] sau mỗi epoch (mỗi lần cập nhật)
11    """
12
13    # initial point
14    W = np.array([-5, -2], dtype=np.float32)
15    # list of results
16    results = [W]
17    ##### YOUR CODE HERE #####
18    # Tạo vòng lặp theo số lần epochs
19    # tìm gradient dW gồm dw1 và dw2
20    # dùng thuật toán optimization cập nhật w1 và w2
21    # append cặp [w1, w2] vào list results
22
23
24    #####
25    return results

```

Hình 2: Function thực hiện train tìm điểm minimum theo thuật toán GD

- **STEP1:** Tại function `df_w`, các bạn tìm đạo hàm của function (1) theo  $w_1$  và  $w_2$  (partial derivative). Tiếp theo các bạn dùng  $w_1$ ,  $w_2$  lấy từ input  $W$  thế vào hai function vừa đạo hàm sẽ thu được  $dw_1$  và  $dw_2$ . Cuối cùng đưa  $[dw_1, dw_2]$  vào một array và gán vào biến  $W$  để return.
- **STEP2:** Xây dựng hàm `sgd` theo công thức Gradient Descent (1.1) để cập nhật  $w_1$  và  $w_2$ . Sau khi thu được  $w_1$  và  $w_2$  mới, ta sẽ gán vào biến  $W$  (array chứa  $[w_1, w_2]$  mới) để return
- **STEP3:** Xây dựng hàm `train_p1` nhận 3 input function optimize, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với  $w_1 = -5$  và  $w_2 = -2$  (line 14, hình 2), tạo list `results` chứa các điểm được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo) (line 16, hình 2). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function `df_w` để tìm gradient, rồi gọi hàm `sgd` để thực hiện cập nhật  $w_1$  và  $w_2$ . Cuối cùng append cặp  $w_1$  và  $w_2$  sau mỗi lần cập nhật vào list `results`

### 3. Bài 2 Gradient Descent + Momentum:

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

$$W_t = W_t - \alpha * V_t \quad (2.2)$$

- (a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent + Momentum (tìm  $w_1$  và  $w_2$  sau 2 epoch) với epoch = 2.
- 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  $dw_2$  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
- (b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent + Momentum (tìm  $w_1$  và  $w_2$ ) với epoch = 30. Các bạn nên tham khảo file hint trước.
- **STEP1:** Tương tự như STEP1 của Bài 1 Gradient Descent. Tại function `df_w`, các bạn tìm đạo hàm của function (1) theo  $w_1$  và  $w_2$  (partial derivative) và return về  $W$  là array chứa  $[dw_1, dw_2]$
  - **STEP2:** Xây dựng hàm `sgd_momentum` theo công thức (2.1) và (2.2) Gradient Descent + Momentum để cập nhật  $v_1$ ,  $v_2$ ,  $w_1$  và  $w_2$ . Sau đó ta sẽ gán vào biến  $V$  (array chứa  $[v_1, v_2]$  mới) và  $W$  (array chứa  $[w_1, w_2]$  mới) để return
  - **STEP3:** Xây dựng hàm `train_p1` nhận 3 input function optimize, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với  $w_1 = -5$  và  $w_2 = -2$ , khởi tạo  $v_1 = 0$  và  $v_2 = 0$ , tạo list `results` chứa các điểm  $[w_1, w_2]$  được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function `df_w` để tìm gradient, rồi gọi hàm `sgd_momentum` để thực hiện cập nhật  $v_1$ ,  $v_2$ ,  $w_1$  và  $w_2$ . Cuối cùng append cặp  $w_1$  và  $w_2$  sau mỗi lần cập nhật vào list `results`

#### 4. Bài 3 RMSProp:

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

$$W_t = W_t - \alpha * \frac{dW}{\sqrt{S_t + \epsilon}} \quad (3.2)$$

- (a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán RMSProp (tìm  $w_1$  và  $w_2$  sau 2 epoch) với epoch = 2.

- 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

- (b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán RMSProp (tìm  $w_1$  và  $w_2$ ) với epoch = 30. Các bạn nên tham khảo file hint trước.

- **STEP1:** Tương tự như STEP1 của Bài 1 Gradient Descent. Tại function `df_w`, các bạn tìm đạo hàm của function (1) theo  $w_1$  và  $w_2$  (partial derivative) và return về  $W$  là array chứa  $[dw_1, dw_2]$
- **STEP2:** Xây dựng hàm RMSProp theo công thức (3.1) và (3.2) để cập nhật  $s_1, s_2, w_1$  và  $w_2$ . Sau đó ta sẽ gán vào biến  $S$  (array chứa  $[s_1, s_2]$  mới) và  $W$  (array chứa  $[w_1, w_2]$  mới) để return
- **STEP3:** Xây dựng hàm `train_p1` nhận 3 input function optimize, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với  $w_1 = -5$  và  $w_2 = -2$ , khởi tạo  $s_1 = 0$  và  $s_2 = 0$ , tạo list `results` chứa các điểm  $[w_1, w_2]$  được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function `df_w` để tìm gradient, rồi gọi hàm RMSProp để thực hiện cập nhật  $s_1, s_2, w_1$  và  $w_2$ . Cuối cùng append cặp  $w_1$  và  $w_2$  sau mỗi lần cập nhật vào list `results`

#### 5. Bài 4 Adam:

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

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

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

$$S_{corr} = \frac{S_t}{1 - \beta_2^t} \quad (4.4)$$

$$W_t = W_t - \alpha * \frac{V_{corr}}{\sqrt{S_{corr}} + \epsilon} \quad (4.5)$$

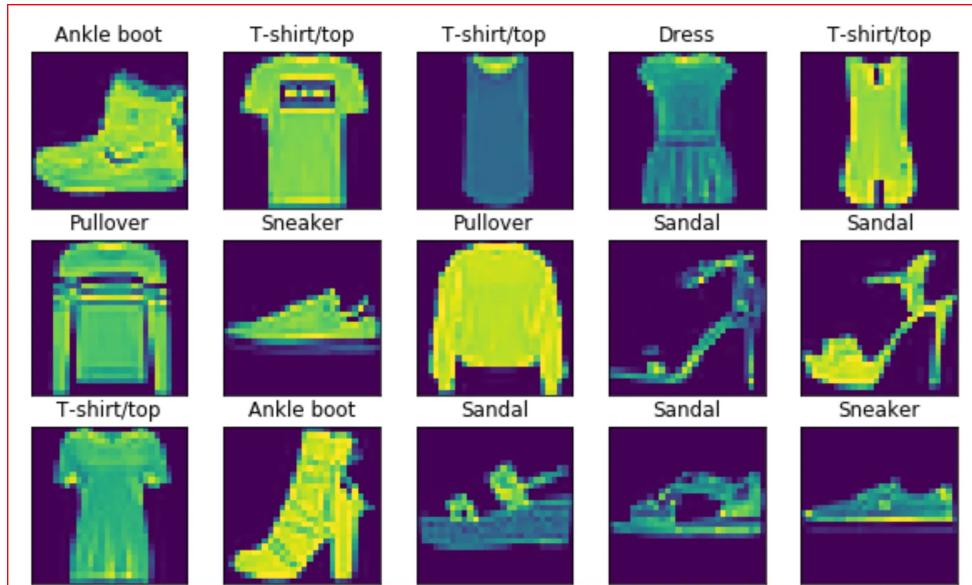
- (a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Adam (tìm  $w_1$  và  $w_2$  sau 2 epoch) với epoch = 2.

- 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
- (b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Adam (tìm  $w_1$  và  $w_2$ ) với epoch = 30. Các bạn nên tham khảo file hint trước.
- **STEP1:** Tương tự như STEP1 của Bài 1 Gradient Descent. Tại function df\_w, các bạn tìm đạo hàm của function (1) theo  $w_1$  và  $w_2$  (partial derivative) và return về W là array chứa  $[dw_1, dw_2]$
  - **STEP2:** Xây dựng hàm Adam theo công thức (4.1), (4.2), (4.3), (4.4) và (4.5) để cập nhật  $v_1, v_2, s_1, s_2, w_1$  và  $w_2$ . Sau đó ta sẽ gán vào biến V (array chứa  $[v_1, v_2]$  mới), S (array chứa  $[s_1, s_2]$  mới) và W (array chứa  $[w_1, w_2]$  mới) để return
  - **STEP3:** Xây dựng hàm train\_p1 nhận 3 input function optimize, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với  $w_1 = -5$  và  $w_2 = -2$ , khởi tạo  $v_1 = 0$  và  $v_2 = 0$ , khởi tạo  $s_1 = 0$  và  $s_2 = 0$ , tạo list results chứa các điểm  $[w_1, w_2]$  được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function df\_w để tìm gradient, rồi gọi hàm Adam để thực hiện cập nhật  $v_1, v_2, s_1, s_2, w_1$  và  $w_2$ . Cuối cùng append cặp  $w_1$  và  $w_2$  sau mỗi lần cập nhật vào list results

**6. Bài 5 Vanishing Problem (Optional):** Các bạn sẽ nhận được 1 file code để train một MLP model phân loại 10 class thời trang trên tập FashionMNIST. Model đã được thiết kế để xảy ra vấn đề Vanishing. Nhiệm vụ của các bạn là sẽ thay thế các optimizer đã làm ơ trên quan sát và đánh giá performance của các optimizer với Vanishing như thế nào

FashionMNIST là một tập dữ liệu hình ảnh bài viết của Zalando, được giới thiệu như một lựa chọn thách thức hơn so với tập dữ liệu MNIST truyền thống để đánh giá các thuật toán học máy. Nó được thiết kế để phục vụ như một sự thay thế trực tiếp cho MNIST gốc. Mỗi sample trong tập dữ liệu FashionMNIST là một hình ảnh xám 28x28, liên kết với một trong 10 class thời trang như áo sơ mi/áo phông, ... Có 60.000 hình ảnh cho tập train và 10.000 hình ảnh cho tập test.



Hình 3: Sample FashionMNIST Data

**Load Data:**

- Cell 1

- Dòng 1: Khởi tạo biến batch\_size với giá trị 512. Đây chỉ định số lượng mẫu dữ liệu sẽ được đưa qua mạng trong mỗi lần cập nhật trọng số.
- Dòng 2: Khởi tạo biến num\_epochs với giá trị 300.
- Dòng 3: Khởi tạo biến lr (learning rate) với giá trị 0.01. Đây là một tham số quan trọng chỉ định bước học hoặc tốc độ mà model cập nhật trọng số của nó dựa trên gradient của hàm mất mát.

- Cell 2:

- Dòng 1: Khởi tạo tập dữ liệu train\_train\_dataset sử dụng tập dữ liệu FashionMNIST. Các hình ảnh sẽ được lưu trữ trong thư mục './data', tải về nếu chưa có và biến đổi thành tensor.
- Dòng 3: Khởi tạo train\_loader, một bộ tải dữ liệu, sử dụng train\_dataset với batch size đã chỉ định và có tùy chọn xáo trộn dữ liệu.
- Dòng 4: Khởi tạo tập test test\_dataset tương tự như train\_dataset nhưng đặt tham số train thành False để chỉ định rằng đây là tập test.
- Dòng cuối: Khởi tạo test\_loader sử dụng test\_dataset với batch size đã chỉ định và không có tùy chọn xáo trộn dữ liệu (mặc định là shuffle=False).

```
[ ] 1 batch_size = 512
[ ] 2 num_epochs = 300
[ ] 3 lr = 0.01

[ ] 1 train_dataset = FashionMNIST(root='./data', train=True,
[ ]                                download=True, transform=transforms.ToTensor())
[ ] 2 train_loader = DataLoader(train_dataset, batch_size, shuffle=True)
[ ] 3 test_dataset = FashionMNIST(root='./data', train=False,
[ ]                                download=True, transform=transforms.ToTensor())
[ ] 4 test_loader = DataLoader(test_dataset, batch_size)
```

Hình 4: Load Data

## Build và Compile Model

- Cell 1

- Trong cell này, chúng ta định nghĩa một lớp MLP kế thừa từ nn.Module, một lớp cơ bản trong thư viện PyTorch dùng để xây dựng các model học sâu.
- Trong phương thức `__init__`, chúng ta khởi tạo mạng với: Một lớp đầu vào layer1 chuyển đổi từ số chiều `input_dims` đến `hidden_dims`. Bốn lớp ẩn layer2, layer3, layer4, và layer5 mỗi lớp chuyển đổi từ số chiều `hidden_dims` đến `hidden_dims`. Một lớp đầu ra output chuyển đổi từ `hidden_dims` đến `output_dims`. Hàm kích hoạt sigmoid được sử dụng sau mỗi lớp.
- Phương thức forward xác định cách dữ liệu được truyền qua mạng. Đầu tiên, dữ liệu được làm phẳng bằng `nn.Flatten()`, sau đó truyền qua mỗi lớp và hàm kích hoạt sigmoid. Cuối cùng, dữ liệu được truyền qua lớp đầu ra và kết quả trả về.

- Cell 2

- Khởi tạo model MLP với số chiều đầu vào là 784 (tương đương với hình ảnh 28x28 pixel), số chiều của lớp ẩn là 128, và số chiều đầu ra là 10 (đại diện cho 10 lớp trong tập dữ liệu FashionMNIST).
- Khởi tạo hàm mất mát criterion (loss) sử dụng hàm mất mát CrossEntropyLoss, thường được sử dụng cho các bài toán phân loại nhiều lớp.
- Khởi tạo thuật toán tối ưu hóa optimizer sử dụng SGD (Stochastic Gradient Descent) với tốc độ học lr và các tham số của model.

**NOTE:** Ở đây sử dụng optimizer là SGD các bạn sẽ thay đổi các optimizer theo yêu cầu của đề bài

```

1 class MLP(nn.Module):
2     def __init__(self, input_dims, hidden_dims, output_dims):
3         super(MLP, self).__init__()
4         self.layer1 = nn.Linear(input_dims, hidden_dims)
5         self.layer2 = nn.Linear(hidden_dims, hidden_dims)
6         self.layer3 = nn.Linear(hidden_dims, hidden_dims)
7         self.layer4 = nn.Linear(hidden_dims, hidden_dims)
8         self.layer5 = nn.Linear(hidden_dims, hidden_dims)
9         self.output = nn.Linear(hidden_dims, output_dims)
10        self.sigmoid = nn.Sigmoid()
11
12    def forward(self, x):
13        x = nn.Flatten()(x)
14        x = self.layer1(x)
15        x = self.sigmoid(x)
16        x = self.layer2(x)
17        x = self.sigmoid(x)
18        x = self.layer3(x)
19        x = self.sigmoid(x)
20        x = self.layer4(x)
21        x = self.sigmoid(x)
22        x = self.layer5(x)
23        x = self.sigmoid(x)
24        out = self.output(x)
25        return out
[ ] 1 model = MLP(input_dims=784, hidden_dims=128, output_dims=10).to(device)
2 criterion = nn.CrossEntropyLoss()
3 optimizer = optim.SGD(model.parameters(), lr=lr)

```

Hình 5: Build và Compile Model

### Train và Evaluate:

- Khởi tạo list: Bốn list train\_losses, train\_acc, val\_losses, và val\_acc được tạo ra để lưu trữ giá trị mất mát và độ chính xác của model trên tập huấn luyện và tập kiểm thử sau mỗi kỳ (epoch).
- Train: Vòng lặp for chạy qua số lượng epoch đã xác định trước (num\_epochs). Trong mỗi epoch: model được chuyển sang chế độ huấn luyện (model.train()).
  - Một vòng lặp bên trong chạy qua tất cả các batch dữ liệu trong train\_loader.
  - Gradient của các tham số model được đặt về 0.
  - Model tiến hành dự đoán và mất mát được tính toán.
  - Gradient được tính toán thông qua phương thức .backward().
  - Các tham số model được cập nhật bằng thuật toán của optimizer.
  - Giá trị mất mát và độ chính xác của model trên tập huấn luyện được log lại.
- Evaluate: Sau khi huấn luyện xong trên tất cả các batch:
  - model được chuyển sang chế độ đánh giá (model.eval()).
  - Test data được đưa qua model.

```

1 train_losses = []
2 train_acc = []
3 val_losses = []
4 val_acc = []
5 for epoch in range(num_epochs):
6     model.train()
7     t_loss = 0
8     t_acc = 0
9     cnt = 0
10    for X, y in train_loader:
11        X, y = X.to(device), y.to(device)
12        optimizer.zero_grad()
13        outputs = model(X)
14        loss = criterion(outputs, y)
15        loss.backward()
16        optimizer.step()
17        t_loss += loss.item()
18        t_acc += (torch.argmax(outputs, 1) == y).sum().item()
19        cnt += len(y)
20    t_loss /= len(train_loader)
21    train_losses.append(t_loss)
22    t_acc /= cnt
23    train_acc.append(t_acc)
24
25    model.eval()
26    v_loss = 0
27    v_acc = 0
28    cnt = 0
29    with torch.no_grad():
30        for X, y in test_loader:
31            X, y = X.to(device), y.to(device)
32            outputs = model(X)
33            loss = criterion(outputs, y)
34            v_loss += loss.item()
35            v_acc += (torch.argmax(outputs, 1)==y).sum().item()
36            cnt += len(y)
37    v_loss /= len(test_loader)
38    val_losses.append(v_loss)
39    v_acc /= cnt
40    val_acc.append(v_acc)

```

Hình 6: Train và Evaluate

- Mắt mát và độ chính xác trên tập test được log lại.
- Lưu trữ và in kết quả: Giá trị mắt mát trung bình và độ chính xác trung bình cho tập huấn luyện và tập kiểm thử được thêm vào các list tương ứng.

## 2) Yêu Cầu Bài Tập:

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  $w_1 = -5, w_2 = -2, \alpha = 0.4$ :
  - (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).
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  $w_1 = -5, w_2 = -2, v_1 = 0, v_2 = 0, \alpha = 0.6, \beta = 0.5$ :
  - (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).
3. **RMSProp:** Dựa vào thuật toán RMSProp các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo  $w_1 = -5, w_2 = -2, s_1 = 0, s_2 = 0, \alpha = 0.3, \gamma = 0.9, \epsilon = 10^{-6}$  :
  - (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).
4. **Adam:** Dựa vào thuật toán Adam các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo  $w_1 = -5, w_2 = -2, \text{initial } v_1 = 0, v_2 = 0, s_1 = 0, s_2 = 0, \alpha = 0.2, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-6}$ :
  - (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).
5. **Vanishing Problem (Optional):** Bài tập này yêu cầu các bạn thay đổi các optimizer và quan sát sự giảm thiểu vấn đề vanishing của từng loại thuật toán. Một model được xây dựng theo các yêu cầu như sau: weight được khởi tạo random theo normal distribution ( $\mu = 0, \sigma = 0.05$ ), Loss = Cross entropy, Optimizer = SGD, Hidden layers = 5, Nodes mỗi layer = 128, Activation = Sigmoid. Các bạn sẽ sử dụng các thuật toán như sau với model trên:
  - (a) Gradient Descent
  - (b) Gradient Descent + Momentum
  - (c) RMSProp
  - (d) Adam

## Phần III: Trắc Nghiệm:

1. Vấn đề của Gradient Descent trong không gian có nhiều chiều là gì?:
 

(A). Không thể tối ưu hóa	(B). Luôn tối ưu hóa nhanh chóng
(C). Bị mắc kẹt ở cực tiểu địa phương	(D). Không liên quan đến số lượng chiều
  
2. Momentum trong Gradient Descent giúp giảm thiểu?:
 

(A). Overfitting	(B). Sự dao động khi cập nhật trọng số
(C). Learning rate	(D). Underfitting
  
3. Mục đích chính của RMSProp là giải quyết vấn đề gì?:
 

(A). Tăng tốc độ hội tụ	(B). Giảm sự dao động trong quá trình cập nhật
(C). Điều chỉnh learning rate dựa trên gradient	(D). Tăng learning rate
  
4. So với các thuật toán tối ưu hóa khác như Gradient Descent và RMSProp, lợi ích chính của Adam là gì?:
 

(A). Nó luôn đạt được hiệu suất tốt nhất trên mọi tập dữ liệu	(B). Kết hợp cả momentum và điều chỉnh tỷ lệ học dựa trên độ lớn của gradient.
(C). Nó chỉ sử dụng momentum mà không quan tâm đến độ lớn của gradient	(D). Không cần đến việc điều chỉnh learning rate.;
  
5. Với bài tập **Gradient Descent**: câu (a) tại **epoch = 2** sau khi áp dụng Gradient Descent ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):
 

(A). -4.232, -0.72	(B). -4.232, -0.83
(C). -4.232, -0.94	(D). -4.232, -0.75
  
6. Với bài tập **Gradient Descent**: câu (b) tại **epoch cuối cùng** sau khi áp dụng Gradient Descent ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):
 

(A). -3.098e-01, -4.521e-07	(B). -3.098e-01, -4.421e-07
(C). -4.098e-01, -4.521e-07	(D). -4.098e-01, -4.421e-07
  
7. Với bài tập **Gradient Descent + Momentum**: câu (a) tại **epoch = 2** sau khi áp dụng Gradient Descent + Momentum ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):
 

(A). -4.468, 1.22	(B). -4.368, 1.32
(C). -4.268, 1.12	(D). -4.568, 1.42
  
8. Với bài tập **Gradient Descent + Momentum**: câu (b) tại **epoch cuối cùng** sau khi áp dụng Gradient Descent + Momentum ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):
 

(A). -7.1e-03, 6.45e-05	(B). -6.1e-03, 7.45e-06
(C). -7.1e-02, 7.45e-06	(D). -6.1e-02, 6.45e-05

9. Mục đích chính của việc sử dụng thuật toán tối ưu hóa trong deep learning là gì?  
(A). Điều chỉnh kiến trúc của mạng neural  
(B). Điều chỉnh mạng để ngăn chặn overfitting  
(C). Cập nhật weight của mạng để tối thiểu hóa hàm loss  
(D). Tăng kích thước dữ liệu đầu vào cho mô hình

10. Cho gradient 1 chiều  $g_t$  tại thời điểm  $t$  có giá trị 2 và sử dụng momentum với  $\beta = 0.9$ , giá trị moving average  $m_t$  sẽ là bao nhiêu nếu  $m_{t-1}$  là 1??  
(A). 1.1  
(B). 1.9  
(C). 2.9  
(D). 0.01

11. Với bài tập **RMSProp**: câu (a) tại **epoch = 2** sau khi áp dụng RMSProp ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):  
(A). -3.436, -0.791  
(B). -4.436, -0.791  
(C). -4.436, -0.591  
(D). -3.436, -0.591

12. Với bài tập **RMSProp**: câu (b) tại **epoch cuối cùng** sau khi áp dụng RMSProp ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):  
(A). 3.00577e-04, -3.005e-18  
(B). 3.00577e-03, -3.005e-17  
(C). 3.00577e-02, -3.005e-16  
(D). 3.00577e-01, -3.005e-15

13. Với bài tập **Adam**: câu (a) tại **epoch = 2** sau khi áp dụng Adam ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):  
(A). -4.6002546, -1.6008245  
(B). -5.6002546, -2.6008245  
(C). -6.6002546, -3.6008245  
(D). -7.6002546, -4.6008245

14. Với bài tập **Adam**: câu (b) tại **epoch cuối cùng** sau khi áp dụng Adam ta thu được  $w_1$  và  $w_2$  là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):  
(A). -0.71, 0.0679  
(B). -0.51, 0.0679  
(C). -0.11, 0.0679  
(D). -0.31, 0.0679