Gradient descent

Nhắc lại hàm mất mát

– Giá trị dự đoán đầu ra: $\hat{y} = \boldsymbol{\sigma(\theta^T x + b)}$, với $\boldsymbol{\sigma}(z) = \frac{1}{1+e^{-z}}$.

– Hàm mất mát:

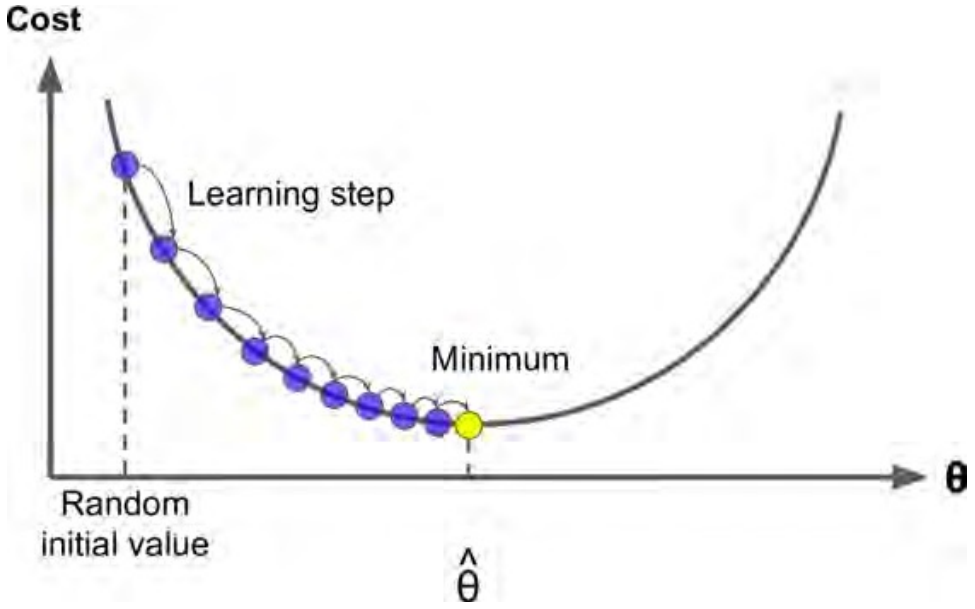
$$\mathcal{L}(\boldsymbol{\theta}, b) = \frac{-1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{y}^{(i)}\right) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})\right]$$

Ý nghĩa hàm chi phí: Đo lường mức độ hiệu quả mà **tham số $\boldsymbol{\theta}$ và b** đang thực hiện trên tập huấn luyện.

➔ Để tối ưu hàm chi phí $\mathcal{L}(\boldsymbol{\theta}, b)$ thì ta phải **hiệu chỉnh giá trị 2 tham số $\boldsymbol{\theta}$ và b**.

Gradient descent

– Gradient descent là một thuật toán tối ưu hoá dùng để tìm điểm tối tiểu cục bộ (**local minimum**) của một hàm số.



Thuật toán gradient descent

— Bước 1: Khởi tạo trọng số $\boldsymbol{\theta}$ và b.

+ W = $\boldsymbol{\theta}_0$.

+ b = b_0.

— Bước 2: Lặp

Ứng với mỗi bước lặp, cập nhật trọng số cho W và b.

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha * \frac{d\mathcal{L}(\boldsymbol{\theta}, b)}{d\boldsymbol{\theta}}$$

$$b = b - \alpha * \frac{d\mathcal{L}(\boldsymbol{\theta}, b)}{db}.$$

α được gọi là learning rate.

$$W := \boldsymbol{\theta}_0$$

$$b := b_0$$

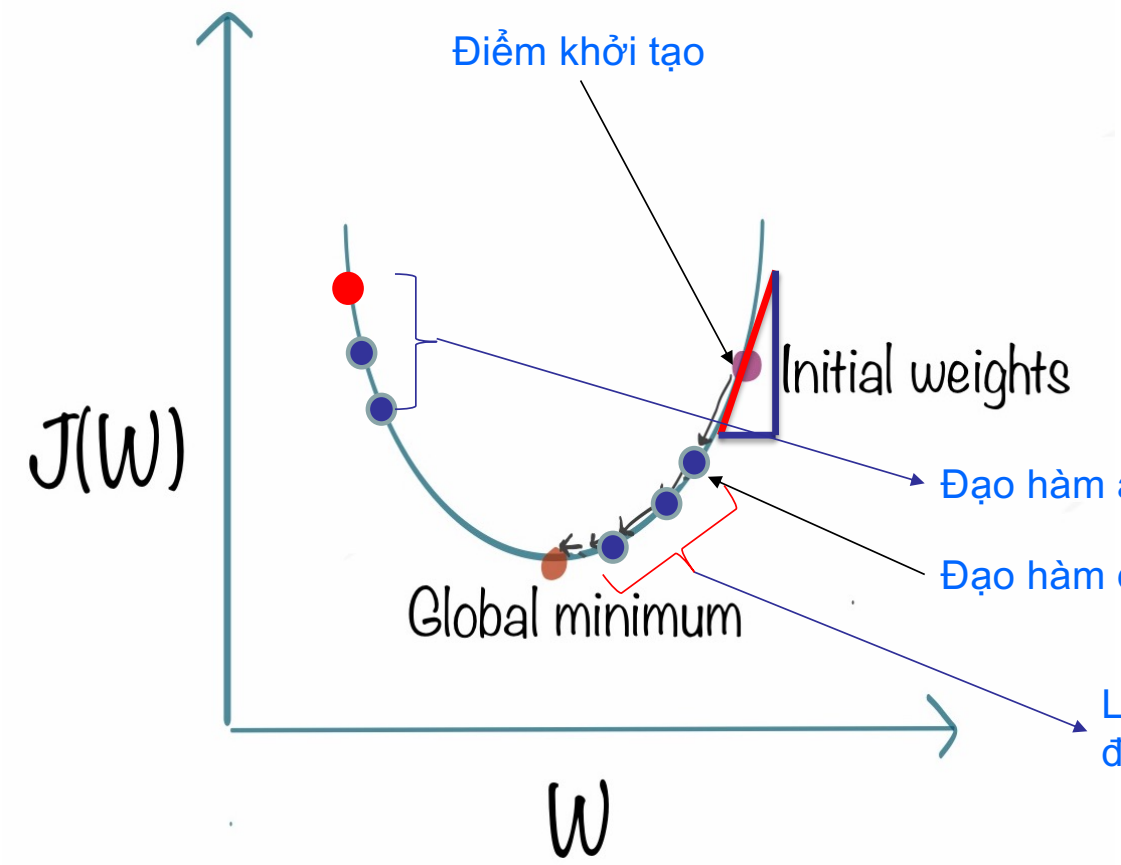
Repeat {

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha * \frac{d\mathcal{L}(\boldsymbol{\theta}, b)}{d\boldsymbol{\theta}}$$

$$\mathbf{b} := \mathbf{b} - \alpha * \frac{d\mathcal{L}(\boldsymbol{\theta}, b)}{db}$$

}

Gradient Descent



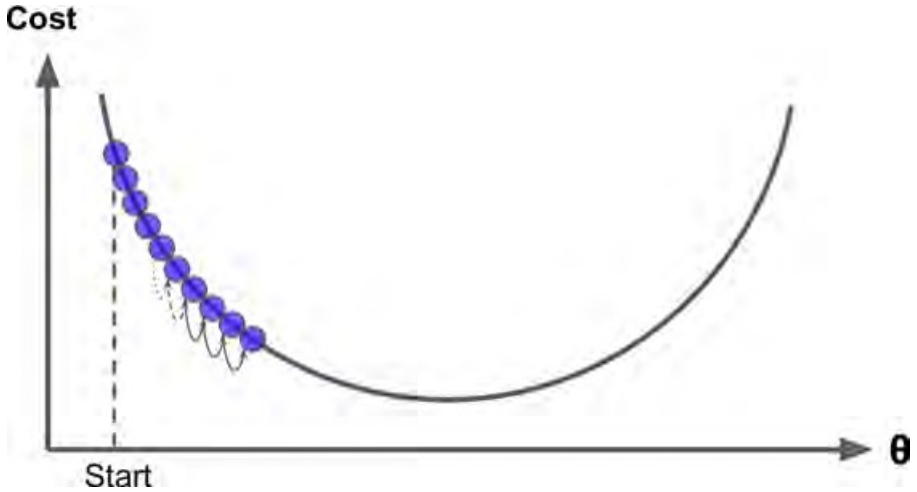
Định nghĩa của đạo hàm: Là độ dốc của hàm ngay tại 1 điểm

Đạo hàm âm ➔ $\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha*(-d\,\boldsymbol{\theta}) = \boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha*d\,\boldsymbol{\theta}$

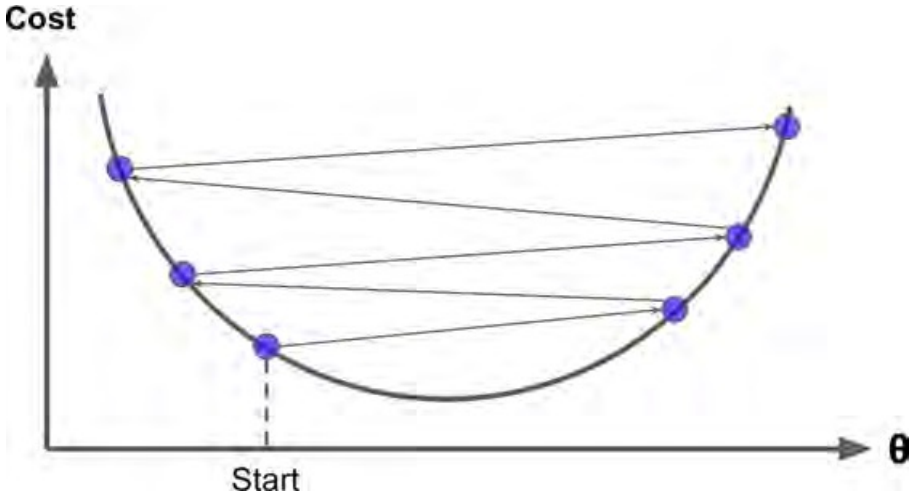
Đạo hàm dương ➔ $\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha*d\,\boldsymbol{\theta}$

Lặp đi lặp lại nhiều lần liên tiếp, W sẽ tiến tới điểm cực tiểu

Giá trị của learning rate



learning rate nhỏ: cần nhiều thời gian để đạt tới cực tiểu (minimum)



learning rate lớn: giá trị sẽ bị dao động, dẫn tới hiện tượng bị "phân tán" (diverge), không hội tụ được về điểm cực tiểu (minimum)

Batch Gradient descent

Batch gradient descent (BGD)

- Thuật toán này sẽ tìm ra tham số tối ưu cho mô hình trên toàn bộ tập dữ liệu huấn luyện (gọi là 1 batch).
- Mỗi bước cập nhật tham số được mô tả bằng toán học như sau:

$$\boldsymbol{\theta^{next\_step} \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta)}$$

Trong đó:

α: hệ số học (learning rate).

$\nabla_{\theta}\mathcal{L}(\theta)$: gradient vectors. Chứa tất cả các giá trị đạo hàm riêng (parital derivatives) của hàm mất mát theo *từng tham số* θ_{i}.

Đạo hàm riêng

- Hàm loss của Linear regression: $\mathcal{L}(\theta) = \frac{1}{2\mathrm{m}}\sum_{\mathrm{i}=1}^{\mathrm{m}}\left(\theta^{\mathrm{T}} \cdot x^{(i)} - \mathrm{y}^{(\mathrm{i})}\right)^2$
- Đạo hàm riêng của hàm loss $\mathcal{L}$ theo tham số θ_j được mô tả như sau: $\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{x}_{\mathrm{j}}^{(\mathrm{i})}\left(\theta^{\mathrm{T}} \cdot \mathrm{x}_{\mathrm{j}}^{(i)} - \mathrm{y}^{(\mathrm{i})}\right)$.
- Viết lại $\frac{\partial \mathcal{L}}{\partial \theta_j}$ dưới dạng vector hoá, ta được:

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{1}{\mathrm{N}}\left(\theta^{\mathrm{T}} \cdot \mathrm{X} - \mathrm{y}\right)\mathrm{X}^{\mathrm{T}}$$

Tính đạo hàm riêng

Hàm loss: $\mathcal{L}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\theta^T \cdot x^{(i)} - y^{(i)}\right)^2$

Đạo hàm riêng của hàm loss $\mathcal{L}$ theo tham số θ_j

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{1}{2m}\sum_{i=1}^{m} 2\left(\theta^T \cdot x^{(i)} - y^{(i)}\right) * \frac{\partial \mathcal{L}}{\partial \theta_j}\theta^T \cdot x^{(i)}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{1}{2m}\sum_{i=1}^{m} 2\left(\theta^T \cdot x^{(i)} - y^{(i)}\right) * x^{(i)}$$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot x^{(i)} - y^{(i)}\right) * x^{(i)}$$

Đạo hàm của hàm hợp:
$(u^\alpha)' = \alpha u^{\alpha-1} * u'$

Gradient descent vector

– Vector gradient đạo hàm riêng theo tham số θ_j được thể hiện như sau:

$$\nabla_\theta \mathcal{L}(\theta) = \begin{bmatrix} \frac{\partial \mathcal{L}(\theta)}{\partial \theta_0} \\ \frac{\partial \mathcal{L}(\theta)}{\partial \theta_1} \\ \cdots \\ \frac{\partial \mathcal{L}(\theta)}{\partial \theta_n} \end{bmatrix} = \frac{1}{N} \boldsymbol{X^T}(\boldsymbol{\theta^T} \cdot \boldsymbol{X} - \boldsymbol{y})$$

– Thế vector gradient đạo hàm riêng cho từng tham số θ_j vào công thức cập nhật tham số của gradient descent, ta được như sau:

$$\theta^{next\_step} \leftarrow \theta - \alpha \frac{1}{\mathrm{N}} \mathrm{X}^{\mathrm{T}}(\theta^{\mathrm{T}} \cdot \mathrm{X} - \mathrm{y})$$

Tạo dữ liệu huấn luyện

– Dữ liệu huấn luyện:

```
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
X_b = np.c_[np.ones((100, 1)), X]
```

– Dữ liệu kiểm tra:

```
X_new = np.array([[0], [2]])
X_new_b = np.c_[np.ones((2, 1)), X_new]
```

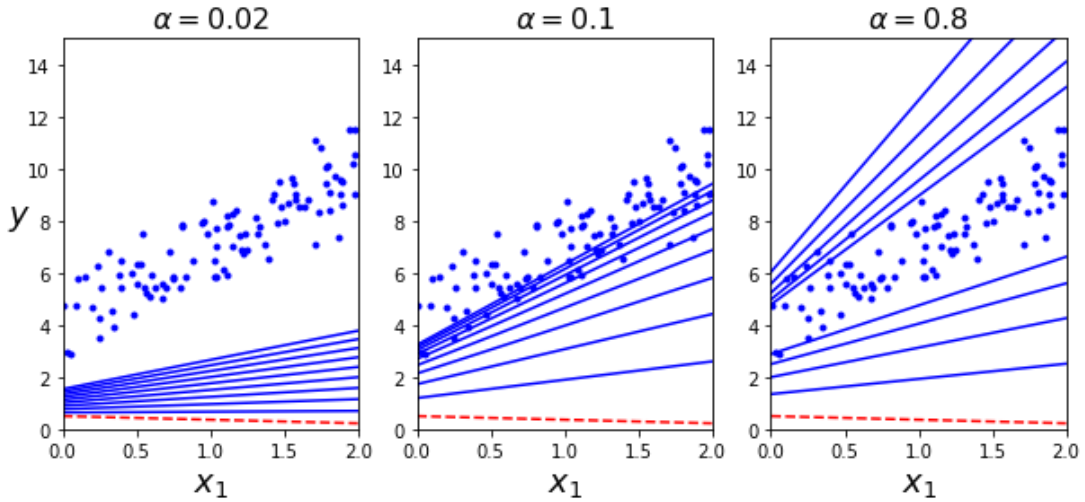
Minh hoạ

– Thực hiện gradient descent với 1000 iteration, learning rate là 0.1 và N = 100

```
1. import numpy as np
2. alpha = 0.1 # learning rate
3. n_iterations = 1000
4. N = 100
5. theta = np.random.randn(2,1) # random initialization
6. for iteration in range(n_iterations):
7.    gradients = 1/N * X_b.T.dot(X_b.dot(theta) - y)
8.    theta = alpha - eta * gradients
```

Ảnh hưởng của learning rate

iteration = 1000



đường màu đỏ: giá trị khởi tạo ban đầu

- Với $\alpha = 0.02$: đường hồi quy đạt đến giá trị minimum rất chậm.
- Với $\alpha = 0.1$: đường hồi quy đạt đến giá trị minimum.
- Với $\alpha = 0.8$: đường hồi quy bị nhảy ra xa giá trị minimum.

Nhận xét

- Batch gradient descent sẽ tính đạo hàm trên toàn bộ tập dữ liệu huấn luyện. Tuy nhiên, nếu tập dữ liệu huấn luyện rất lớn (khoảng vài tỷ điểm dữ liệu) thì việc tính toán sẽ tốn rất nhiều tài nguyên.
- Không hiệu quả với online learning:
  + Dữ liệu cập nhật liên tục.
  + Thời gian đáp ứng phải nhanh.

Stochastic Gradient descent

Stochastic Gradient descent (SGD)

- Stochastic Gradient descent (SGD) sẽ cập nhật lại tập tham số θ bằng cách lấy ngẫu nhiên 1 điểm dữ liệu (instance) từ tập huấn luyện.
- Mục tiêu:
  + Tăng tốc quá trình huấn luyện: do chỉ tính đạo hàm cho 1 điểm dữ liệu thay vì tính cho toàn bộ tập dữ liệu huấn luyện.
  + Có khả năng huấn luyện đối với tập dữ liệu huấn luyện lớn và rất lớn: việc load 1 điểm dữ liệu vào bộ nhớ sẽ khả thi hơn là load một tập dữ liệu rất lớn (vài chục đến vài trăm gigabyte).

Stochastic Gradient descent (SGD)

– Hàm cập nhật tham số cho SGD được mô tả như sau:

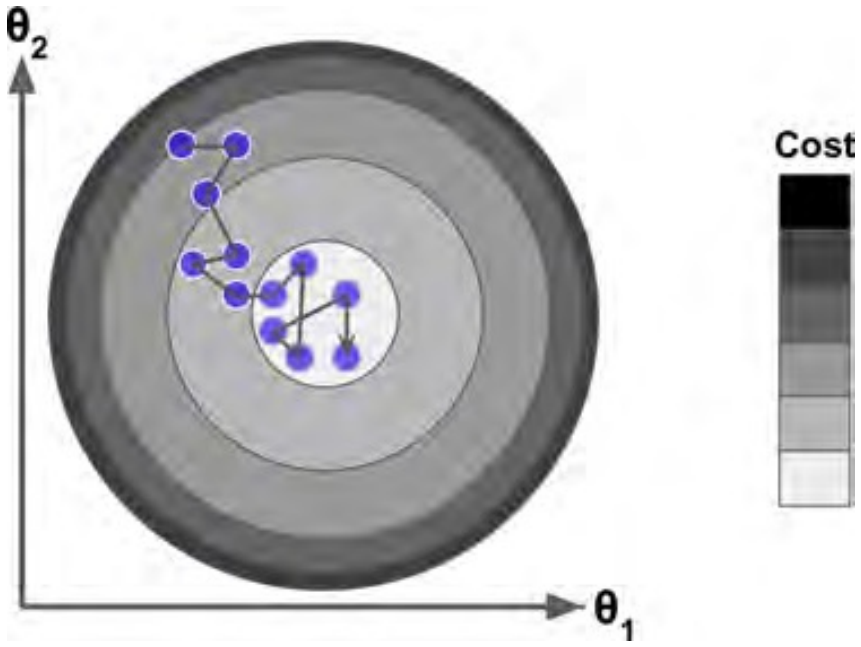
$$\boldsymbol{\theta}^{next\_step} \leftarrow \boldsymbol{\theta} - \boldsymbol{\alpha}\boldsymbol{\nabla}_{\boldsymbol{\theta}}\boldsymbol{\mathcal{L}}(\boldsymbol{\theta}; x_i; y_i)$$

$\boldsymbol{\mathcal{L}}(\boldsymbol{\theta}; x_i; y_i)$: hàm mất mát với **1 điểm dữ liệu** $(x_i; y_i)$

– Một lần duyệt qua dữ liệu huấn luyện để cập nhật lại tham số $\boldsymbol{\theta}$ được gọi là một epoch.

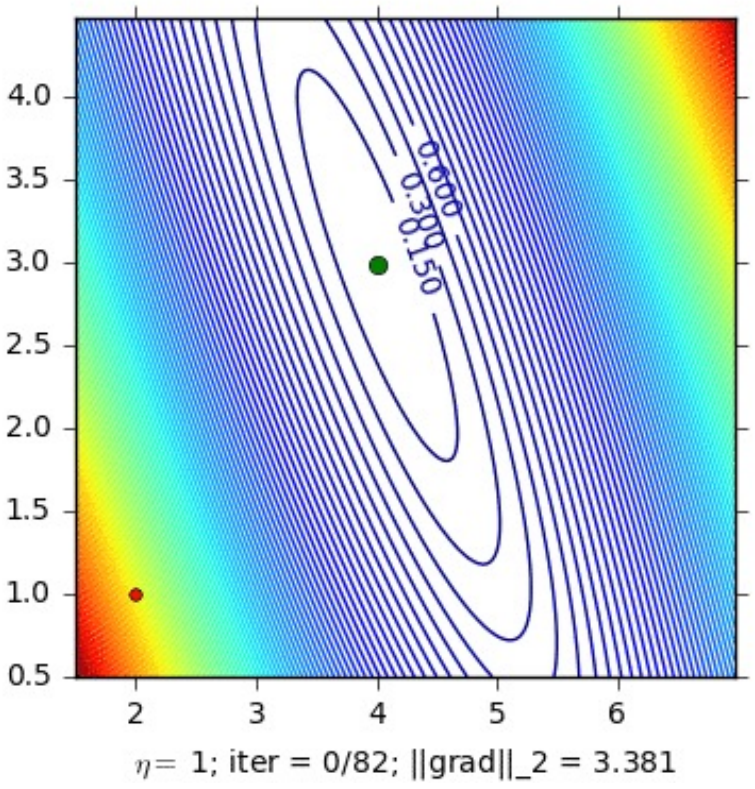
– Với SGD, sẽ có tổng cộng N epoch ứng với N điểm dữ liệu trong tập huấn luyện.

Stochastic gradient descent



- Do ảnh hưởng của quá trình lấy ngẫu nhiên, giá trị của hàm loss sẽ dao động lên xuống (nhiễu).
- Vì giá trị loss dao động lên xuống, giá trị cuối cùng tìm được sẽ chỉ gần với giá trị minimum chứ không *thực sự đạt đến giá trị minimum*.

Visualize Stochastic Gradient Descent



Simulated annealing

- Điều chỉnh learning_rate theo từng epoch: giá trị learning rate ban đầu sẽ lớn, sau đó giá trị sẽ giảm dần sau mỗi epoch.
- Mục tiêu:
  + Giá trị learning_rate ban đầu lớn ➔ giúp cho quá trình học nhanh hơn (đạt đến cực tiểu nhanh hơn).
  + Giá trị learning_rate giảm dần dần ➔ tạo sự ổn định cho thuật toán (giảm nhiễu).
- Quá trình này được gọi là ***simulated annealing*** (có thể đọc thêm về tối ưu hoá – optimization) để biết thêm.

Minh hoạ

```
n_epochs = 50
t0, t1 = 5, 50 # learning schedule hyperparameters
alpha = 0.1    # learning rate
N = 100

def learning_schedule(t):
    return t0/(t+t1)
```

Điều chỉnh giá trị learning rate theo từng epoch

Minh hoạ

— Hiện thực thuật toán SGD:

```
1. theta = np.random.randn(2,1) # random initialization
2. for epoch in range(n_epochs):
3.    for i in range(N):
4.          random_index = np.random.randint(N)
5.          xi = X_b[random_index:random_index+1]
6.          yi = y[random_index:random_index+1]
7.          gradients = xi.T.dot(xi.dot(theta) - yi)
8.          alpha = learning_schedule(epoch * N + i)
9.          theta = theta - alpha * gradients
```

Minh hoạ với sklearn

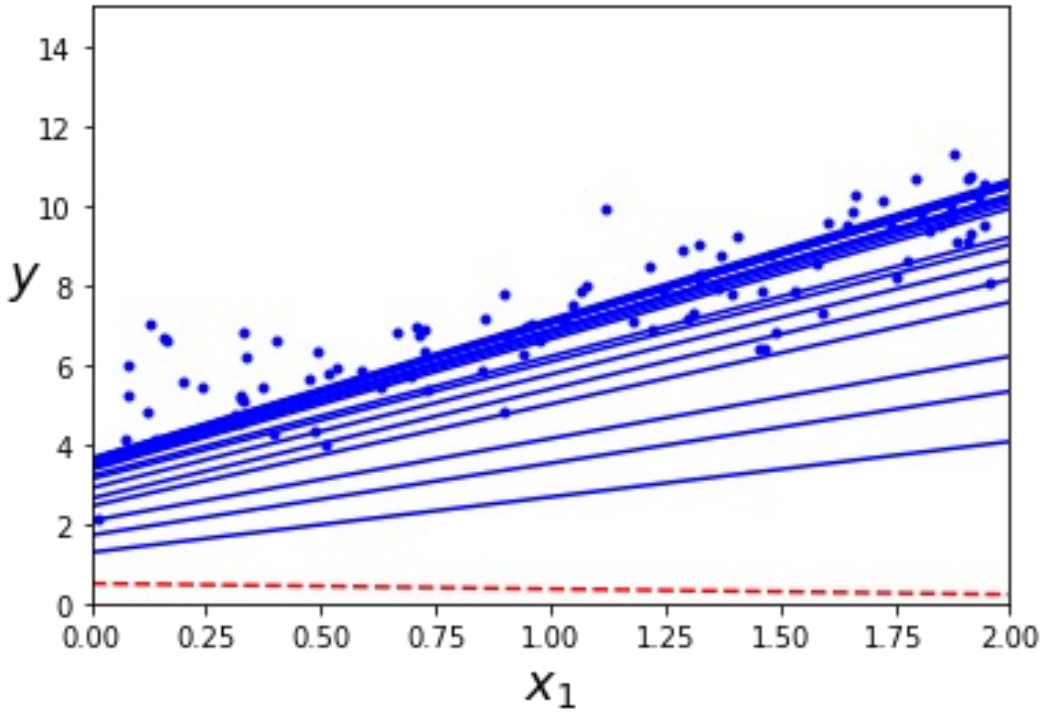
– Sử dụng `SGDRegressor` trong sklearn:

```
1. from sklearn.linear_model import SGDRegressor


2. sgd_reg = SGDRegressor(max_iter=50, penalty=None, eta0=0.1)
3. sgd_reg.fit(X, y.ravel())
```

– Hàm *`ravel()`* trong numpy sẽ trả về mảng 1 chiều đã được làm phẳng (flattened).

Kết quả thực hiện SGD với 10 step đầu



đường màu đỏ: giá trị khởi tạo ban đầu

Nhận xét

- Tốc độ hội tụ đến điểm cực trị của SGD nhanh hơn so với BGD. Tuy nhiên, đường giá trị loss của SGD không mượt như BGD (có nhiều giá trị nhiễu).
- Cần xáo trộn dữ liệu để đảm bảo tính ngẫu nhiên.
- Tuy nhiên, quá trình ngẫu nhiên trong SGD khiến cho quá trình học không ổn định, khó thực sự đạt được giá trị cực tiểu (minimum).

Mini-batch gradient descent

Batch and Mini Batch

– Dữ liệu huấn luyện:

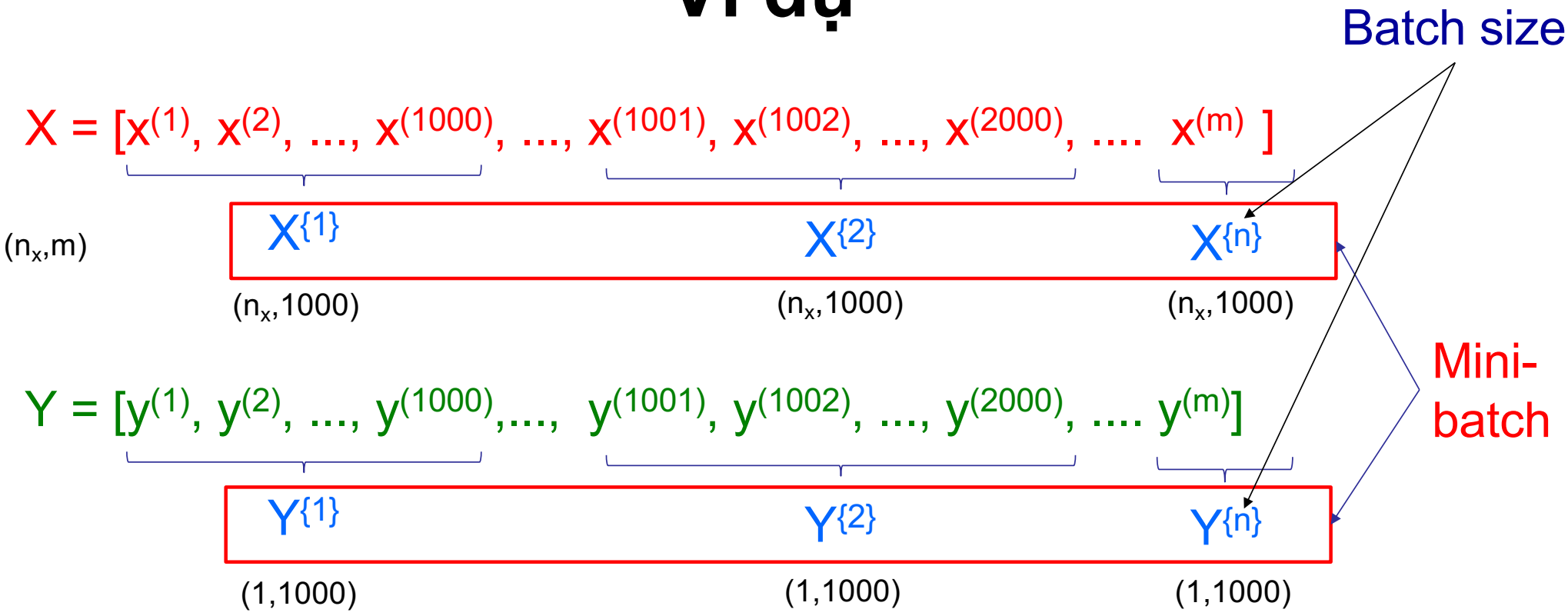
$X = [x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(m)}]$

$Y = [y^{(1)}, y^{(2)}, y^{(3)}, ..., y^{(m)}]$

Nếu như dữ liệu có kích thước quá lớn (>50M) thì việc huấn luyện một lần hết cả bộ dữ liệu không khả thi (không đủ tài nguyên, ...).

➔ Chia dữ liệu ra thành từng đoạn nhỏ, gọi là batch, lần lượt huấn luyện các batch nhỏ trên.

Ví dụ



Mỗi mini-batch chứa 1000 điểm dữ liệu (batch size: n=1000)

Mini-batch gradient descent (MGD)

– Thuật toán này sẽ cập nhật lại tập tham số θ dựa trên n điểm dữ liệu từ tập dữ liệu huấn luyện (n < N).

➔Kết hợp giữa BGD và SGD.

– Hàm cập nhật tham số cho MGD được mô tả như sau:

$$\boldsymbol{\theta}^{\boldsymbol{next\_step}} \leftarrow \boldsymbol{\theta} - \boldsymbol{\alpha}\boldsymbol{\nabla}_{\boldsymbol{\theta}}\boldsymbol{\mathcal{L}}(\boldsymbol{\theta}; x_{i:i+n}; y_{i:i+n})$$

$x_{i:i+n}$: dữ liệu từ vị trí i đến vị trí i+n - 1 trong tập dữ liệu. Tương tự cho $y_{i:i+n}$.

– Giá trị n được gọi là **mini-batch size** (đôi khi còn gọi là **batch size**), thường được chọn là khoảng từ 50 đến 100.

Minh hoạ

– Dữ liệu huấn luyện:

```
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
X_b = np.c_[np.ones((100, 1)), X]
```

– Dữ liệu kiểm tra:

```
X_new = np.array([[0], [2]])
X_new_b = np.c_[np.ones((2, 1)), X_new]
```

Minh hoạ

– Thực hiện MGD với mini-batch size là 20:

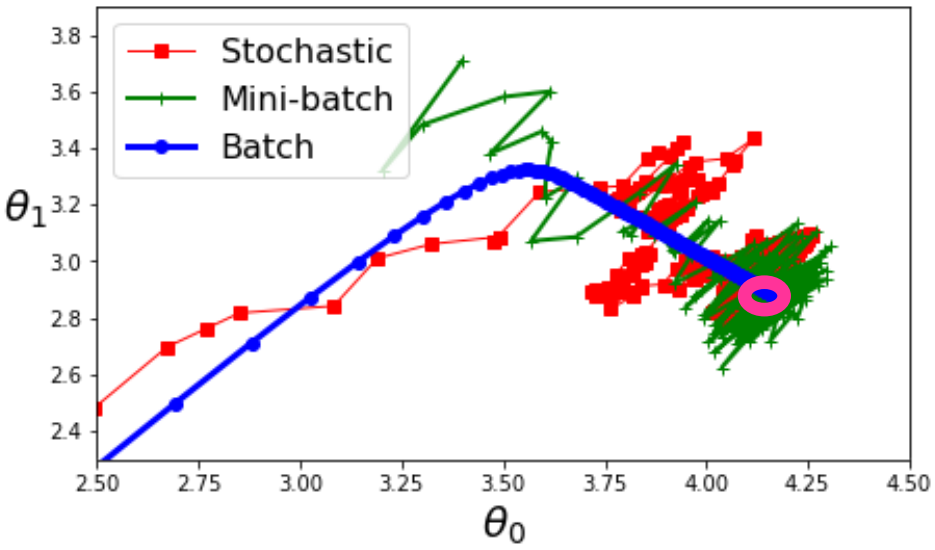
```
1. n_epochs = 50
2. t0, t1 = 5, 50    # learning schedule hyperparameters
3. alpha = 0.1       # learning rate
4. N = 100
5. minibatch_size = 20

6. def learning_schedule(t):
7.    return t0/(t+t1)
```

Minh hoạ

```
1. theta = np.random.randn(2,1) # random initialization
2. for epoch in range(n_epochs):
3.     shuffled_indices = np.random.permutation(N)
4.     X_b_shuffled = X_b[shuffled_indices]
5.     y_shuffled = y[shuffled_indices]
6.     for i in range(N):
7.         random_index = np.random.randint(N)
8.         xi = X_b_shuffled[i:i+minibatch_size]
9.         yi = y_shuffled[i:i+minibatch_size]
10.        gradients = xi.T.dot(xi.dot(theta) - yi)
11.        alpha = learning_schedule(epoch * N + i)
12.        theta = theta - alpha * gradients
```

So sánh giữa các thuật toán gradient descent



theta_best = (4.17795097, 2.87092976)

- BGD: **N = m** (m: kích thước tập dữ liệu). Thuật toán huấn luyện khá chậm khi kích thước dữ liệu lớn.
- SGD: **N = 1**. Thuật toán huấn luyện khá nhanh chóng, tuy nhiên khả năng xảy ra nhiễu cao.
- MGD: **N = k (k < m)**. Thuật toán kết hợp điểm mạnh của cả 2. Thường được dùng trong các mô hình Deep learning. k thường trong khoảng [50, 100].

TỔNG KẾT

- Gradient descent là thuật toán dùng để tìm được tập tham số tối ưu của một mô hình máy học dựa trên hàm mất mát (loss) và dữ liệu huấn luyện.
- Các biến thể của thuật toán gradient descent:
  + Batch gradient descent (BGD): huấn luyện trên toàn bộ tập dữ liệu.
  + Stochastic gradient descent (SGD): huấn luyện trên 1 điểm dữ liệu ngẫu nhiên trong tập dữ liệu.
  + Mini-batch gradient descent (MGD): huấn luyện trên n điểm dữ liệu ngẫu nhiên trong tập dữ liệu.

TÀI LIỆU THAM KHẢO

1. Chương 4 của sách: *Hands-on Machine Learning with ScikitLearn, Keras & TensorFlow.*
2. *Vũ Hữu Tiệp, Machine Learning cơ bản – Chương Gradient Descent, NXB Khoa học và Kỹ thuật (2018).*
3. *Blog: https://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent*