|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 1.3

“Tìm hiểu về CNTK”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2022

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 1.3

“Tìm hiểu về CNTK”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2022

MỤC LỤC

[MỤC LỤC 1](#_Toc115870594)

[TÌM HIỂU VỀ CNTK 2](#_Toc115870595)

[1.1. Giới thiệu về CNTK 2](#_Toc115870596)

[1.2. Hoạt động 2](#_Toc115870597)

[1.3. Các tính năng 2](#_Toc115870598)

[1.4. CNTK Python API 3](#_Toc115870599)

[1.4.1. Dense 4](#_Toc115870600)

[1.4.2. Sequential 5](#_Toc115870601)

[1.4.3. For 6](#_Toc115870602)

[1.4.4. Recurrence 6](#_Toc115870603)

[1.4.5. LSTM(), GRU(), RNNStep() 7](#_Toc115870604)

[1.4.6. BatchNormalization(), LayerNormalization(), Stabilizer() 8](#_Toc115870605)

[1.4.7. momentum\_sgd() 9](#_Toc115870606)

[1.4.8. learning\_parameter\_schedule() 11](#_Toc115870607)

[1.4.9. learning\_parameter\_schedule\_per\_sample () 12](#_Toc115870608)

[1.4.10. learning\_rate\_schedule () 12](#_Toc115870609)

[1.4.11. momentum\_schedule () 13](#_Toc115870610)

# TÌM HIỂU VỀ CNTK

## 1.1. Giới thiệu về CNTK

Microsoft Cognitive Toolkit (CNTK) là một bộ công cụ mã nguồn mở dành cho học sâu phân tán cấp thương mại. Nó mô tả mạng nơ-ron như một chuỗi các bước tính toán thông qua một đồ thị có hướng. CNTK cho phép người dùng dễ dàng nhận ra và kết hợp các loại mô hình phổ biến như DNN chuyển tiếp, mạng nơ-ron tích hợp (CNN) và mạng nơ-ron tuần hoàn (RNN / LSTM). CNTK triển khai tính năng học phân tích độ dốc ngẫu nhiên (SGD, lan truyền lỗi) với khả năng phân biệt và song song tự động trên nhiều GPU và máy chủ.

## 1.2. Hoạt động

* Phiên bản mới nhất của CNTK là 2.7
* CNTK có thể được đưa vào như một thư viện trong các chương trình Python, C # hoặc C ++ của bạn hoặc được sử dụng như một công cụ học máy độc lập thông qua ngôn ngữ mô tả mô hình của riêng nó (BrainScript). Ngoài ra, bạn có thể sử dụng chức năng đánh giá mô hình CNTK từ các chương trình Java của mình.
* CNTK hỗ trợ hệ điều hành Linux 64 bit và Windows 64 bit

## 1.3. Các tính năng

**Các thành phần tích hợp**:

* CNTK có các thành phần tích hợp được tối ưu hóa cao để có thể xử lý dữ liệu dày đặc hoặc thưa thớt đa chiều từ Python, C ++ hoặc BrainScript.
* Chúng ta có thể triển khai CNN, FNN, RNN, Batch Normalization và Sequence-to-Sequence với sự chú ý.
* Nó cung cấp cho chúng ta chức năng để thêm các thành phần lõi mới do người dùng xác định trên GPU từ Python.
* Nó cũng cung cấp khả năng điều chỉnh hyperparameter tự động.
* Chúng ta có thể triển khai học tập củng cố, Mạng đối thủ chung (GAN), Học tập có giám sát cũng như Không giám sát.
* Đối với các bộ dữ liệu lớn, CNTK đã tích hợp sẵn các trình đọc tối ưu hóa.

**Sử dụng tài nguyên hiệu quả:**

* CNTK cung cấp cho chúng ta tính năng song song với độ chính xác cao trên nhiều GPU / máy thông qua SGD 1-bit.
* Để phù hợp với các mô hình lớn nhất trong bộ nhớ GPU, nó cung cấp tính năng chia sẻ bộ nhớ và các phương pháp tích hợp khác.

**Dễ dàng thể hiện mạng lưới:**

* CNTK có đầy đủ các API để xác định mạng, người học, người đọc, đào tạo và đánh giá của riêng bạn từ Python, C ++ và BrainScript.
* Sử dụng CNTK, chúng ta có thể dễ dàng đánh giá các mô hình với Python, C ++, C # hoặc BrainScript.
* Nó cung cấp cả API cấp cao và cấp thấp.
* Dựa trên dữ liệu của chúng ta, nó có thể tự động định hình suy luận.
* Nó có các vòng lặp Mạng thần kinh tái diễn (RNN) tượng trưng được tối ưu hóa đầy đủ.

**Đo lường hiệu suất mô hình:**

* CNTK cung cấp các thành phần khác nhau để đo hiệu suất của mạng nơ-ron mà bạn xây dựng.
* Tạo dữ liệu nhật ký từ mô hình của bạn và trình tối ưu hóa liên quan, chúng ta có thể sử dụng dữ liệu này để theo dõi quá trình đào tạo.

## 1.4. CNTK Python API

[CNTK Python API](https://cntk.ai/pythondocs/) bao gồm các phần tóm tắt để định nghĩa và tính toán mô hình, học thuật toán, đọc dữ liệu và đào tạo phân tán.

* *Tính linh hoạt và tính gọn nhẹ* : Các cấu trúc trừu tượng này tạo ra một cách trực giao cung cấp cả tính linh hoạt và tính ngắn gọn trong định nghĩa và đào tạo các mạng nơ-ron tùy ý.
* *Giao diện dữ liệu hiệu quả* : Các giao diện dữ liệu đơn giản nhưng nhẹ nhàng cho phép người dùng cung cấp dữ liệu một cách hiệu quả dưới dạng các mảng riêng cho công cụ tính toán.
* *Trình đọc dữ liệu* tích hợp: Trình đọc dữ liệu được tối ưu hóa và có thể mở rộng tích hợp của CNTK cho các định dạng dữ liệu HTK hình ảnh, văn bản và giọng nói cũng có sẵn từ các API Python để dễ dàng đào tạo trực tiếp với dữ liệu hiện có mà người dùng không cần phải viết bất kỳ mã đọc dữ liệu nào.
* *Học tập có khả năng mở rộng cao* : API cho thấy khả năng đào tạo phân tán có khả năng mở rộng cao của CNTK (các thuật toán song song hóa như 1-Bit SGD). Ví [dụ đào tạo phân tán](https://github.com/Microsoft/CNTK/tree/release/latest/Examples/Image/Classification/ResNet/Python#trainresnet_cifar10_distributedpy) minh họa API song song đào tạo.
* *Định nghĩa mạng ngắn gọn* : API bao gồm [thư viện các lớp](https://cntk.ai/pythondocs/layerref.html) cấp cao cho phép định nghĩa mạng nơ-ron nâng cao ngắn gọn bao gồm các lần lặp lại tương tự như CNTK V1. Bộ công cụ hỗ trợ biểu diễn các mô hình lặp lại ở dạng biểu tượng dưới dạng các chu trình trong mạng nơ-ron thay vì yêu cầu thao tác gỡ tĩnh các bước lặp lại. Điều này dẫn đến việc biểu diễn và thực thi mạng nơ-ron lặp lại một cách tổng quát, ngắn gọn và hiệu quả hơn nhiều.

Tất cả các bản tóm tắt API tính toán, học tập và đọc dữ liệu cốt lõi trong CNTK Python API đều có thể mở rộng rất dễ dàng từ cả Python và C ++, cho phép người dùng dễ dàng triển khai các toán tử, người học và trình đọc dữ liệu mới có thể tự do biên soạn với các cơ sở tích hợp sẵn của thư viện .

### 1.4.1. Dense

Dense(shape, activation = activation\_identity, init = [init\_glorot\_uniform](https://microsoft.github.io/CNTK-R/reference/init_glorot_uniform.html)(),

input\_rank = NULL, map\_rank = NULL, bias = TRUE, init\_bias = 0,

name = "")

**Arguments**

|  |  |
| --- | --- |
| **shape** | Danh sách số nguyên đại diện cho hình dạng tensor |
| **activation** | (Function) – hàm kích hoạt tùy chọn |
| **init** | (vô hướng hoặc ma trận hoặc trình khởi tạo, mặc định là init\_glorot\_uniform()) –giá trị ban đầu của trọng số W |
| **bias** | (bool) – nếu là TRUE tức là bao gồm bias |
| **init\_bias** | (vô hướng hoặc ma trận hoặc trình khởi tạo, mặc định là 0) – giá trị ban đầu của trọng số b |
| **name** | (string) (optional) tên của hàm trong mạng |

**Chi tiết:**

* Lớp Dense có thể được áp dụng cho các đầu vào là tensor, không chỉ là vecto. Điều là rất hữu ích, ví dụ, ở trên cùng của tầng xử lý hình ảnh, nơi mà sau nhiều lần lặp lại với các bước đệm, rất khó để biết được kích thước chính xác. Đối với trường hợp này, CNTK có một định nghĩa mở rộng về ma trận, trong đó tensor đầu vào sẽ được coi như thể nó đã tự động làm phẳng
* Dense thể hiện một fully connected layer, tức toàn bộ các unit của layer truóc đó được nối với toàn bộ của các unit của layer hiện tại.Giá trị đầu tiền trọng

### 1.4.2. Sequential

Sequential (arrayOfFunctions)

* **Tham** **số**: arrayOfFunctions: danh sách các hàm có thể là các phiên bản của lớp hoặc các đối số nguyên thủy, ví dụ: [ LinearLayer(1024), sigmoid ]
* **Giá trị trả về**: Hàm này trả về một Hàm khác. Hàm trả về đó nhận một đối số và trả về đó nhận một đối số và trả về kết quả của việc áp dụng tất cả các hàm đã cho theo trình tự đầu vào.
* **Ví dụ**

my\_model **=** **Sequential** ([

Dense(2048, activation**=**sigmoid), *# four hidden layers*

Dense(2048, activation**=**sigmoid),

Dense(2048, activation**=**sigmoid),

Dense(2048, activation**=**sigmoid),

Dense(9000, activation**=**softmax) *# note:last layer is a softmax*

)

features **=** Input(40)

p **=** my\_model(features)

### 1.4.3. For

For (rng, constructor, name=’’)

Lặp lại một lớp nhiều lần

* **Tham số:**
  + rng: số lần lặp lại
  + constructor: một lambda với 0 hoặc 1 đối số tạo ra lớp
* **Giá trị trả lại:** Hàm này trả về một hàm khác. Hàm trả về đó nhận một đối số và trả về kết quả của việc áp dụng các lớp lặp lại cho đầu vào, trong đó mỗi lớp là một thể hiện riêng biệt với một tập các tham số mô hình riêng biệt.
* **Ví dụ:** Sau đây tạo mô hình kiểu VGG 9 lớp ẩn. VGG là một kiến ​​trúc phổ biến để nhận dạng hình ảnh:

**with** default\_options(activation**=**relu):

model **=** **Sequential**([

For(range(3), **lambda** i: [ *# lambda with one parameter*

Convolution((3,3), [64,96,128][i], pad**=True**), *# depth depends on i*

Convolution((3,3), [64,96,128][i], pad**=True**),

MaxPooling((3,3), strides**=**(2,2))

]),

For(range(2), **lambda** : [ *# lambda without parameter*

Dense(1024),

Dropout(0.5)

]),

Dense(num\_classes, activation**=None**)

])

### 1.4.4. Recurrence

Recurrence(step\_function, go\_backwards=default\_override\_or(False), initial\_state=default\_override\_or(0), return\_full\_state=False, name='')

* **Tham số:**
  + step\_function: hàm lặp lại
  + go\_backwards (optinal): nếu đặt là True, hàm sẽ chạy lại ngược lại
  + initial\_state (optional, default 0): giá trị ban đầu của biến ẩn
* **Giá trị trả về:** Recurrence () tạo ra một hàm thực hiện lớp mong muốn áp dụng lặp lại một mô hình, chẳng hạn như LSTM, cho một trình tự đầu vào. Lớp này ánh xạ một chuỗi đầu vào thành một chuỗi các trạng thái ẩn có cùng độ dài.
* Ví dụ

h\_fwd **=** Recurrence(LSTM(150))(e)

h\_bwd **=** Recurrence(LSTM(150), go\_backwards**=True**)(e)

h **=** splice (h\_fwd, h\_bwd)

### 1.4.5. LSTM(), GRU(), RNNStep()

Hàm chức năng tạo LSTM / GRU / RNN không trạng thái, thường được sử dụng với Recurrence().

LSTM(shape, cell\_shape**=None**, activation**=**default\_override\_or(tanh), use\_peepholes**=**default\_override\_or(**False**),

init**=**default\_override\_or(glorot\_uniform()), init\_bias**=**default\_override\_or(0),

enable\_self\_stabilization**=**default\_override\_or(**False**),

name**=**'')

GRU(shape, cell\_shape**=None**, activation**=**default\_override\_or(tanh),

init**=**default\_override\_or(glorot\_uniform()), init\_bias**=**default\_override\_or(0),

enable\_self\_stabilization**=**default\_override\_or(**False**),

name**=**'')

RNNStep(shape, cell\_shape**=None**, activation**=**default\_override\_or(sigmoid),

init**=**default\_override\_or(glorot\_uniform()), init\_bias**=**default\_override\_or(0),

enable\_self\_stabilization**=**default\_override\_or(**False**),

name**=**'')

* Tham số:
  + shape: chiều của đầu ra
  + cell\_shape (optional): chiều của ô LSTM. Nếu được đặt là **None,**  hình dạng ô giống với **shape**. Nếu được chỉ định, một phép chiều tuyến tính bổ sung sẽ được chèn để chiếu từ kích thước ô đến hình dạng đầu ra.
* **Giá trị trả về:** Một Function

### 1.4.6. BatchNormalization(), LayerNormalization(), Stabilizer()

Tạo các lớp để chuẩn hóa hàng loạt, chuẩn hóa lớp và tự ổn định.

BatchNormalization(map\_rank**=**default\_override\_or(**None**), # if given then normalize only over this many dimensions. E.g. pass 1 to tie all (h,w) in a (C, H, W)-shaped input

init\_scale**=**1,

normalization\_time\_constant**=**default\_override\_or(5000),

blend\_time\_constant**=**0,

epsilon**=**default\_override\_or(0.00001),

use\_cntk\_engine**=**default\_override\_or(**False**),

name**=**'')

LayerNormalization(initial\_scale**=**1, initial\_bias**=**0, epsilon**=**default\_override\_or(0.00001), name**=**'')

Stabilizer(steepness**=**4, enable\_self\_stabilization**=**default\_override\_or(**True**), name**=**'')

* **Tham số:**
  + BatchNormalization:
    - map\_rank :
    - normalization\_time\_constant(mặc định 5000): hằng số thời gian trong các mẫu của bộ lọc thông thấp bậc nhất được sử dụng để tính toán thống kê trung bình / phương sai để sử dụng trong suy luận.
    - initial\_scale: giá trị ban đầu của tham số tỷ lệ
    - epsilon: giá trị nhỏ được thêm vào ước tính phương sai khi tính toán nghịch đảo
    - use\_cntk\_engine: nếu **True**, sử dụng cài đặt gốc của CNTK. Nếu **False**, hãy sử dụng triển khai cuDNN (chỉ GPU).
    - disable\_regularization: nếu **True** thì vô hiệu hóa chính quy trong BatchNormalization.
  + LayerNormalization:
    - initial\_scale: giá trị ban đầu của tham số tỷ lệ
    - initial\_bias: giá trị ban đầu của tham số bias
  + Stabilizer:
    - steepness: độ sắc nét của đầu gối của hàm softplus
* **Giá trị trả về:** Một hàm thực thi một lớp thực hiện hoạt động chuẩn hóa.
* **Ví dụ:**

**def** **my\_convo\_layer**(x, depth, init):

c **=** Convolution(depth, (5,5), pad**=True**, init**=**init)(x)

b **=** BatchNormalization(map\_rank**=**1)(c)

r **=** relu(b)

p **=** MaxPooling((3,3), strides**=**(2,2))(r)

**return** p

### 1.4.7. momentum\_sgd()

Tạo một Momentum SGD learner

momentum\_sgd(parameters, lr, momentum, unit\_gain=default\_unit\_gain\_value(), l1\_regularization\_weight=0.0, l2\_regularization\_weight=0, gaussian\_noise\_injection\_std\_dev=0, gradient\_clipping\_threshold\_per\_sample=np.inf, gradient\_clipping\_with\_truncation=True)

* Tham số
* parameters (danh sách các tham số) : danh sách các tham số mạng cần điều chỉnh. Chúng có thể được lấy bởi các tham số của toán tử gốc.
* Ir (float, list, output of learning\_parameter\_schedule()) – a learning rate in float, or a learning rate schedule
* gamma (float) - Hệ số cân bằng cho các gradient hiện tại và trước đó. Giá trị phổ biến là 0,95. Phải nằm trong phạm vi (0.0 -> 1.0)
* inc ( float ) - Hệ số tăng khi cố gắng điều chỉnh tốc độ học hiện tại. Phải lớn hơn 1
* dec ( float ) - Hệ số giảm khi cố gắng điều chỉnh tốc độ học hiện tại. Phải nằm trong phạm vi (0,0, 1,0)
* max ( float ) - Tỷ lệ tối đa được phép cho learning\_rate ban đầu. Phải lớn hơn 0 và tối thiểu
* min ( float ) - Thang đo tối thiểu được phép cho learning\_rate ban đầu. Phải lớn hơn 0
* need\_ave\_multiplier (bool, mặc định **True**).
* l1\_regulification\_ weight ( float , tùy chọn ) - trọng lượng chính quy L1 trên mỗi mẫu, mặc định là 0.0
* l2\_regulification\_ weight ( float , tùy chọn ) - trọng lượng chính quy L2 trên mỗi mẫu, mặc định là 0.0
* gaussian\_noise\_injection\_std\_dev ( float , tùy chọn ) - độ lệch chuẩn của nhiễu - Gaussian được thêm vào thông số sau khi cập nhật, mặc định là 0.0
* gradient\_clipping\_threshold\_per\_sample ( float , tùy chọn ) - ngưỡng cắt cho mỗi mẫu, mặc định là vô cùng
* gradient\_clipping\_with\_truncation (bool, mặc định **True**) - sử dụng gradient clipping với sự cắt ngắn
* use\_mean\_gradient ( bool , tùy chọn ) - sử dụng gradient trung bình làm đầu vào cho learner.
* Sử dụng tham số **minibatch\_size** để chỉ định kích thước **minibatch** tham chiếu.
  + minibatch\_size (int, default **None**) - Kích thước minibatch mà các tham số của người học được thiết kế hoặc điều chỉnh trước. Kích thước này thường được đặt bằng với kích thước của nguồn dữ liệu minibatch. CNTK sẽ thực hiện chia tỷ lệ tự động của các tham số để cho phép thực hiện cập nhật tham số mô hình hiệu quả trong khi ước tính hành vi của các tham số được thiết kế trước và điều chỉnh trước. Trong trường hợp kích thước minibatch\_size đó không được chỉ định, CNTK sẽ kế thừa kích thước minibatch từ lịch trình tỷ lệ học tập; nếu lịch trình tốc độ học tập không chỉ định minibatch\_size, CNTK sẽ đặt nó thành **IGNORE**. Đặt minibatch\_size thành **IGNORE** sẽ yêu cầu người học áp dụng vì nó ngăn cản CNTK thực hiện bất kỳ tỷ lệ siêu tham số nào.
  + epoch\_size ( tùy chọn , int ) - số lượng mẫu làm đơn vị lập lịch cho tốc độ học.
* **Giá trị trả về: Learner instance** có thể được chuyển cho **Trainer.**

### 1.4.8. learning\_parameter\_schedule()

Create a learning parameter schedule.

learning\_parameter\_schedule(schedule, minibatch\_size=None, epoch\_size=None)

* **Tham số:**
  + schedule( float hoặc list ) - nếu float, là lịch tham số được sử dụng cho tất cả các mẫu. Trong trường hợp danh sách [p\_1, p\_2, .., p\_n], tham số thứ i p\_i trong danh sách được sử dụng làm giá trị từ mẫu thứ ( **epoch\_size**\* (i-1) + 1) đến mẫu thứ ( **epoch\_size**\* i). Nếu danh sách chứa cặp, tức là [(num\_epoch\_1, p\_1), (num\_epoch\_n, p\_2), .., (num\_epoch\_n, p\_n)], thì tham số thứ i được sử dụng làm giá trị từ mẫu thứ ( **epoch\_size**\* (num\_epoch\_0 + ... + num\_epoch\_2 + ... + num\_epoch\_ (i-1) + 1) đến mẫu thứ ( **epoch\_size**\* num\_epoch\_i) (lấy num\_epoch\_0 = 0 làm khởi tạo đặc biệt).
  + minibatch\_size ( int ) - một số nguyên để chỉ định kích thước minibatch mà lịch biểu được thiết kế. CNTK sẽ mở rộng lịch trình trong nội bộ để mô phỏng hành vi của lịch trình nhiều nhất có thể để phù hợp với hiệu quả đã thiết kế. Nếu nó không được chỉ định, CNTK sẽ đặt thành giá trị đặc biệt **IGNORE**.
  + epoch\_size ( tùy chọn , int ) - số lượng mẫu làm đơn vị lập lịch. Các thông số trong lịch trình thay đổi giá trị của chúng mỗi **epoch\_size** mẫu. Nếu **epoch\_size** không được cung cấp, tham số này được thay thế bằng kích thước của toàn bộ quá trình quét dữ liệu, trong trường hợp đó, đơn vị lập lịch là toàn bộ quá trình quét dữ liệu (như được chỉ ra bởi MinibatchSource) và các tham số thay đổi giá trị của chúng trên cơ sở quét từng bước được chỉ định bởi **schedule**.
* **Giá trị trả về:** learning parameter schedule

### 1.4.9. learning\_parameter\_schedule\_per\_sample ()

Tạo một lịch trình tham số học tập như thể tham số được áp dụng cho các minibatch có kích thước 1. CNTK sẽ chia tỷ lệ các tham số tương ứng với kích thước minibatch thực tế.

learning\_parameter\_schedule\_per\_sample(schedule, epoch\_size=None)

* Tham số:
  + schedule( float hoặc list ) - nếu float, là lịch tham số được sử dụng cho tất cả các mẫu. Trong trường hợp danh sách [p\_1, p\_2, .., p\_n], tham số thứ i p\_i trong danh sách được sử dụng làm giá trị từ mẫu thứ ( **epoch\_size**\* (i-1) + 1) đến mẫu thứ ( **epoch\_size**\* i). Nếu danh sách chứa cặp, tức là [(num\_epoch\_1, p\_1), (num\_epoch\_n, p\_2), .., (num\_epoch\_n, p\_n)], thì tham số thứ i được sử dụng làm giá trị từ mẫu thứ ( **epoch\_size**\* (num\_epoch\_0 + ... + num\_epoch\_2 + ... + num\_epoch\_ (i-1) + 1) đến mẫu thứ ( **epoch\_size**\* num\_epoch\_i) (lấy num\_epoch\_0 = 0 làm khởi tạo đặc biệt).
  + epoch\_size ( tùy chọn , int ) - số lượng mẫu làm đơn vị lập lịch. Các thông số trong lịch trình thay đổi giá trị của chúng mỗi **epoch\_size** mẫu. Nếu **epoch\_size** không được cung cấp, tham số này được thay thế bằng kích thước của toàn bộ quá trình quét dữ liệu, trong trường hợp đó, đơn vị lập lịch là toàn bộ quá trình quét dữ liệu (như được chỉ ra bởi MinibatchSource) và các tham số thay đổi giá trị của chúng trên cơ sở quét từng bước được chỉ định bởi **schedule**.
  + **Giá trị trả về:** learning parameter schedule as if it is applied to minibatches of size 1.

### 1.4.10. learning\_rate\_schedule ()

Tạo lịch trình tốc độ học tập (sử dụng ngữ nghĩa giống như training\_parameter\_schedule ()).

learning\_rate\_schedule(lr, unit, epoch\_size=None)

* **Tham số:**
  + lr ( float hoặc list ) - xem tham số **schedule** trong training\_parameter\_schedule()
  + unit ( UnitType) - xem tham số **unit** trong training\_parameter\_schedule().
  + epoch\_size ( int ) - xem tham số epoch\_size trong training\_parameter\_schedule()
* **Giá trị trả về**: learning rate schedule

### 1.4.11. momentum\_schedule ()

Khởi tạo một lập lịch momentum (sử dụng giống như *learning\_parameter\_schedule()*) p dụng phân rã động lượng mỗi N mẫu trong đó N được chỉ định bởi đối số minibatch\_size.training\_parameter\_schedule ()).

momentum\_schedule(momentum, epoch\_size=None, minibatch\_size=None)

* **Tham số:**
  + momentum ( float hoặc list ) - xem tham số **schedule** trong training\_parameter\_schedule()
  + epoch\_size ( int ) - xem tham số epoch\_size trong training\_parameter\_schedule().
  + minibatch\_size (int) - một số nguyên để chỉ định kích thước minibatch tham chiếu; CNTK sẽ chia tỷ lệ xung lượng bên trong để mô phỏng sự phân rã xung lượng của kích thước minibatch được chỉ định trong khi kích thước minibatch thực tế của dữ liệu được cung cấp có thể khác nhau. Bằng cách này, các giá trị động lượng có thể được cung cấp theo cách bất khả tri với kích thước minibatch (phân rã bằng nhau trên mỗi mẫu). Nếu minibatch\_size là None (mặc định), động lượng được áp dụng cho toàn bộ minibatch bất kể kích thước minibatch thực tế là gì (không theo cách bất khả tri đối với kích thước minibatch).
* **Giá trị trả về:** momentum schedule