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

BÁO CÁO CHUYÊN ĐỀ SỐ 2.4

“Nghiên cứu một số thuật toán học sâu tiêu biểu: CNN,GAN,LSTM”

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 2.4

“Nghiên cứu một số thuật toán học sâu tiêu biểu: CNN,GAN,LSTM”

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 3](#_Toc115870836)

[DANH MỤC HÌNH ẢNH 4](#_Toc115870837)

[1. CNN: Convolutional neural network 6](#_Toc115870838)

[1.1. Phép tính Convolution [2] 6](#_Toc115870839)

[1.2. Padding 7](#_Toc115870840)

[1.3. Stride 8](#_Toc115870841)

[1.4. Convolution neural network [1] 9](#_Toc115870842)

[1..4.1 Convolutional layer 9](#_Toc115870843)

[1.4.2. Pooling layer 13](#_Toc115870844)

[1.4.3. Fully connected layer 14](#_Toc115870845)

[2. LSTM: Long Short – Term Memory 15](#_Toc115870846)

[2.1. Cấu trúc LSTM [4] 15](#_Toc115870847)

[2.2. Gated Memory Cell 17](#_Toc115870848)

[2.3. Ứng dụng 20](#_Toc115870849)

[3. GAN: Generative Adversarial Networks 21](#_Toc115870850)

[3.1. Tổng quan về GAN 21](#_Toc115870851)

[3.2. Generator 22](#_Toc115870852)

[3.3. Discriminator 23](#_Toc115870853)

[3.4. Loss functions 24](#_Toc115870854)

[3.5. Ứng dụng 25](#_Toc115870855)

[TÀI LIỆU THAM KHẢO 28](#_Toc115870856)

# DANH MỤC HÌNH ẢNH

[Hình 1.1. Phép tính Convolution 6](#_Toc115870799)

[Hình 1.2: padding của ma trận 7](#_Toc115870800)

[Hình 1.3: Các bước thực hiện phép tính convolution cho ma trận X với kernel K 7](#_Toc115870801)

[Hình 1.4: Ma trận X khi thêm viền 0 bên ngoài 7](#_Toc115870802)

[Hình 1.5: stride = 1, padding = 1 8](#_Toc115870803)

[Hình 1.6: padding = 1, stride = 2 8](#_Toc115870804)

[Hình 1.7: Mô hình neural network. 9](#_Toc115870805)

[Hình 1.8: Input layer và hidden layer 1 9](#_Toc115870806)

[Hình 1.9: Phép tính convolution ảnh xám biểu diễn ảnh dạng ma trận 10](#_Toc115870807)

[Hình 1.10: Phép tính convolution trên ảnh màu với k = 3 11](#_Toc115870808)

[Hình 1.11: Tensor X, W 3 chiều được viết dưới dạng 3 matrix [1] 11](#_Toc115870809)

[Hình 1.12: Thực hiện phép tính convolution trên ảnh màu 12](#_Toc115870810)

[Hình 1.13: Convolution layer đầu tiên 12](#_Toc115870811)

[Hình 1.14: Tính toán trong convolutional 12](#_Toc115870812)

[Hình 1.14: max pooling layer với size = (3,3), stride = 1, padding = 0 13](#_Toc115870813)

[Hình 1.15: Sau pooling layer (2\*2) 14](#_Toc115870814)

[Hình 1.16: Ví dụ về pooling layer 14](#_Toc115870815)

[Hình 2.1: Cấu trúc LSTM 15](#_Toc115870816)

[Hình 2.2: Cổng quên 16](#_Toc115870817)

[Hình 2.3: Cổng đầu vào 16](#_Toc115870818)

[Hình 2.4: Cổng ra 17](#_Toc115870819)

[Hình 2.5: Tính toán cổng đầu vào, cổng quên và cổng đầu ra trong mô hình LSTM 18](#_Toc115870820)

[Hình 2. 6:Tính toán ô nhớ candidate trong mô hình LSTM 19](#_Toc115870821)

[Hình 2. 7:Tính toán ô nhớ trong mô hình LSTM 19](#_Toc115870822)

[Hình 2. 8: Tính toán trạng thái ẩn trong mô hình LSTM 20](#_Toc115870823)

[Hình 3.1: Cấu trúc mạng GAN [3] 21](#_Toc115870824)

[Hình 3.2: Mô hình Generative Adversarial Netword 22](#_Toc115870825)

[Hình 3.3: MNIST dataset [7] 23](#_Toc115870826)

[Hình 3.4: Mô hình generator 23](#_Toc115870827)

[Hình 3.5: Ví dụ về reshape 23](#_Toc115870828)

[Hình 3.6: Mô hình đối xứng lại với Generator 24](#_Toc115870829)

[Hình 3.7: Mô hình Loss function 24](#_Toc115870830)

[Hình 3.8: Ảnh mặt GAN sinh ra qua các năm 25](#_Toc115870831)

[Hình 3.9: StarGAN 26](#_Toc115870832)

[Hình 3.10: AgeGAN 26](#_Toc115870833)

[Hình 3.11: Ảnh anime được sinh ra 26](#_Toc115870834)

[Hình 3.12: Ví dụ ảnh draft sang ảnh màu 27](#_Toc115870835)

# 1. CNN: Convolutional neural network

## 1.1. Phép tính Convolution [2]

Để cho dễ hình dung tôi sẽ lấy ví dụ trên ảnh xám, tức là ảnh được biểu diễn dưới dạng ma trận A kích thước m\*n.

Ta định nghĩa kernel là một ma trận vuông kích thước k\*k trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9... Ví dụ kernel kích thước 3\*3.

W =

Kí hiệu phép tính convolution (⊗), kí hiệu Y = X ⊗W

Với mỗi phần tử xi j trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử xi j làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tính element-wise của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y.

Chart, scatter chart

Description automatically generated

Hình 1.1. Phép tính Convolution

Ví dụ khi tính tại x22 (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W, có x22 làm trung tâm có màu nền da cam như trong hình. Sau đó tính y11 = sum(A⊗W) = x11 ∗w11 +x12 ∗w12 + x13 ∗w13 +x21 ∗w21 +x22 ∗ w22 +x23 ∗w23 +x31 ∗w31 +x32 ∗w32 +x33 ∗w33 = 4. Và làm tương tự với các phần tử còn lại trong ma trận.

Thế thì sẽ xử lý thế nào với phần tử ở viền ngoài như x11? Bình thường khi tính thì sẽ bỏ qua các phần tử ở viền ngoài, vì không tìm được ma trận A ở trong X.

A picture containing chart

Description automatically generated

Hình 1.2: padding của ma trận

Nên bạn để ý thấy ma trận Y có kích thước nhỏ hơn ma trận X. Kích thước của ma trận Y là (m-k+1) \* (n-k+1).

A picture containing diagram

Description automatically generated

Hình 1.3: Các bước thực hiện phép tính convolution cho ma trận X với kernel K

## 1.2. Padding

Như ở trên thì mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận Y đều nhỏ hơn X. Tuy nhiên giờ ta muốn ma trận Y thu được có kích thước bằng ma trận X => Tìm cách giải quyết cho các phần tử ở viền => Thêm giá trị 0 ở viền ngoài ma trận X.

A picture containing different

Description automatically generated

Hình 1.4: Ma trận X khi thêm viền 0 bên ngoài

Rõ ràng là giờ đã giải quyết được vấn đề tìm A cho phần tử x11 , và ma trận Y thu được sẽ bằng kích thước ma trận X ban đầu. Phép tính này gọi là convolution với padding=1. Padding=k nghĩa là thêm k vector 0 vào mỗi phía (trên, dưới, trái, phải) của ma trận.

1.3. Stride

Như ở trên ta thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.

A picture containing shelf

Description automatically generated

Hình 1.5: stride = 1, padding = 1

Tuy nhiên nếu stride=k (k > 1) thì ta chỉ thực hiện phép tính convolution trên các phần tử x1+i\*k, 1+j\*k.  Ví dụ k = 2

Text, table

Description automatically generated

Hình 1.6: padding = 1, stride = 2

Hiểu đơn giản là bắt đầu từ vị trí x11 sau đó nhảy k bước theo chiều dọc và ngang cho đến hết ma trận X. Kích thước của ma trận Y là 3\*3 đã giảm đi đáng kể so với ma trận X. Công thức tổng quát cho phép tính convolution của ma trận X kích thước m\*n với kernel kích thước k\*k, stride = s, padding = p ra ma trận Y kích thước ( + 1) ∗ ( +1).

Stride thường dùng để giảm kích thước của ma trận sau phép tính convolution.

## 1.4. Convolution neural network [1]

### 1..4.1 Convolutional layer

Diagram

Description automatically generated

Hình 1.7: Mô hình neural network.

Mỗi hidden layer được gọi là fully connected layer, tên gọi theo đúng ý nghĩa, mỗi node trong hidden layer được kết nối với tất cả các node trong layer trước. Cả mô hình được gọi là fully connected neural network (FCN).

Vấn đề của fully connected neural network với xử lý ảnh Như bài trước về xử lý ảnh, thì ảnh màu 64\*64 được biểu diễn dưới dạng 1 tensor 64\*64\*3. Nên để biểu thị hết nội dung của bức ảnh thì cần truyền vào input layer tất cả các pixel (64\*64\*3 = 12288). Nghĩa là input layer giờ có 12288 nodes.

Diagram, schematic

Description automatically generated

Hình 1.8: Input layer và hidden layer 1

Giả sử số lượng node trong hidden layer 1 là 1000. Số lượng weight W giữa

input layer và hidden layer 1 là 12288\*1000 = 12288000, số lượng bias là 1000 => tổng số parameter là: 12289000. Đấy mới chỉ là số parameter giữa input layer và hidden layer 1, trong model còn nhiều layer nữa, và nếu kích thước ảnh tăng, ví dụ 512\*512 thì số lượng parameter tăng cực kì nhanh => Cần giải pháp tốt hơn!!!

**Nhận xét:**

* Trong ảnh các pixel ở cạnh nhau thường có liên kết với nhau hơn là những pixel ở xa. Ví dụ như phép tính convolution trên ảnh ở bài trước. Để tìm các đường trong ảnh, ta áp dụng sobel kernel trên mỗi vùng kích thước 3\*3. Hay làm nét ảnh ta áp dụng sharpen kernel cũng trên vùng có kích thước 3\*3.
* Với phép tính convolution trong ảnh, chỉ 1 kernel được dùng trên toàn bộ bức ảnh. Hay nói cách khác là các pixel ảnh chia sẻ hệ số với nhau. => Áp dụng phép tính convolution vào layer trong neural network ta có thể giải quyết được vấn đề lượng lớn parameter mà vẫn lấy ra được các đặc trưng của ảnh.
* **Convolutional layer đầu tiên**

Bài trước phép tính convolution thực hiện trên ảnh xám với biểu diễn ảnh dạng ma trận

Shape

Description automatically generated with low confidence

Hình 1.9: Phép tính convolution ảnh xám biểu diễn ảnh dạng ma trận

Tuy nhiên ảnh màu có tới 3 channels red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều. Nên ta cũng sẽ định nghĩa kernel là 1 tensor 3 chiều kích thước k\*k\*3.

Diagram, engineering drawing

Description automatically generated

Hình 1.10: Phép tính convolution trên ảnh màu với k = 3

Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám

Box and whisker chart

Description automatically generated

Hình 1.11: Tensor X, W 3 chiều được viết dưới dạng 3 matrix [1]

Khi biểu diễn ma trận ta cần 2 chỉ số hàng và cột: i và j, thì khi biểu diễn ở dạng tensor 3 chiều cần thêm chỉ số độ sâu k. Nên chỉ số mỗi phần tử trong tensor là xijk. y11 = b + (x111 ∗ w111 + x121 ∗ w121 + x131 ∗ w131 + x211 ∗ w211 + x221 ∗ w221 + x231 ∗ w231 + x311 ∗ w311+x321 ∗w321+x331 ∗w331)+ (x112 ∗w112+x122 ∗w122+x132 ∗w132+x212 ∗w212+x222 ∗w222+ x232 ∗w232 +x312 ∗w312 +x322 ∗w322 +x332 ∗w332)+ (x113 ∗w113 +x123 ∗w123 +x133 ∗w133 +x213 ∗ w213 +x223 ∗w223 +x233 ∗w233 +x313 ∗w313 +x323 ∗w323 +x333 ∗w333) = −25

* Nhận xét:
* Output Y của phép tính convolution trên ảnh màu là 1 matrix.
* Có 1 hệ số bias được cộng vào sau bước tính tổng các phần tử của phép tính element-wise.

Các quy tắc đối với padding và stride hoàn toàn tương tự như ở bài trước

Với mỗi kernel khác nhau ta sẽ học được những đặc trưng khác nhau của ảnh, nên trong mỗi convolutional layer ta sẽ dùng nhiều kernel để học được nhiều thuộc tính của ảnh. Vì mỗi kernel cho ra output là 1 matrix nên k kernel sẽ cho ra k output matrix. Ta kết hợp k output matrix này lại thành 1 tensor 3 chiều có chiều sâu k.

Calendar

Description automatically generated

Hình 1.12: Thực hiện phép tính convolution trên ảnh màu

Diagram

Description automatically generated

Hình 1.13: Convolution layer đầu tiên

**Convolutional layer tổng quát**

Giả sử input của 1 convolutional layer tổng quát là tensor kích thước H \* W \* D. Kernel có kích thước F \* F \* D (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.

Convolutional layer áp dụng K kernel=> Output của layer là tensor 3 chiều có kích thước: + 1) \* ( + 1) \* K

Diagram

Description automatically generated

Hình 1.14: Tính toán trong convolutional

* **Lưu ý:**
* Output của convolutional layer sẽ qua hàm non-linear activation function trước khi trở thành input của convolutional layer tiếp theo.
* Tổng số parameter của layer: Mỗi kernel có kích thước F\*F\*D và có 1 hệ số bias, nên tổng parameter của 1 kernel là F\*F\*D + 1. Mà convolutional layer áp dụng K kernel => Tổng số parameter trong layer này là K \* (F\*F\*D + 1).

### 1.4.2. Pooling layer

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Việc giảm kích thước dữ liệu giúp giảm các phép tính toán trong model.

Bên cạnh đó, với phép pooling kích thước ảnh giảm, do đó lớp convolution học được các vùng có kích thước lớn hơn. Ví dụ như ảnh kích thước 224\*224 qua pooling về 112\*112 thì vùng 3\*3 ở ảnh 112\*112 tương ứng với vùng 6\*6 ở ảnh ban đầu. Vì vậy qua các pooling thì kích thước ảnh nhỏ đi và convolutional layer sẽ học được các thuộc tính lớn hơn.

Gọi pooling size kích thước K\*K. Input của pooling layer có kích thước H\*W\*D, ta tách ra làm D ma trận kích thước H\*W. Với mỗi ma trận, trên vùng kích thước K\*K trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.

A screenshot of a game

Description automatically generated with low confidence

Hình 1.14: max pooling layer với size = (3,3), stride = 1, padding = 0

Nhưng hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên.

A picture containing text, clock, screenshot

Description automatically generated

Hình 1.15: Sau pooling layer (2\*2)

Diagram

Description automatically generated

Hình 1.16: Ví dụ về pooling layer

Trong một số model người ta dùng convolutional layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.

### 1.4.3. Fully connected layer

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,...) thì tensor của output của layer cuối cùng, kích thước H\*W\*D, sẽ được chuyển về 1 vector kích thước (H\*W\*D, 1)

Graphical user interface

Description automatically generated

Hình 1.17: chuyển feature map sang flatten

Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model.

# 2. LSTM: Long Short – Term Memory

Long Short Term Memory là một loại mạng nơ-ron tuần hoàn. Trong RNN, đầu ra từ bước cuối cùng được cung cấp dưới dạng đầu vào trong bước hiện tại. LSTM được thiết kế bởi Hochreiter & Schmidhuber. Nó giải quyết vấn đề phụ thuộc dài hạn của RNN trong đó RNN không thể dự đoán từ được lưu trữ trong bộ nhớ dài hạn nhưng có thể đưa ra dự đoán chính xác hơn từ thông tin gần đây. Khi độ dài khe hở tăng RNN không cho hiệu suất hiệu quả. Theo mặc định, LSTM có thể giữ lại thông tin trong một khoảng thời gian dài. Nó được sử dụng để xử lý, dự đoán và phân loại trên cơ sở dữ liệu chuỗi thời gian.

## 2.1. Cấu trúc LSTM [4]

LSTM có cấu trúc chuỗi chứa bốn mạng nơ-ron và các khối bộ nhớ khác nhau được gọi là **ô**

Diagram

Description automatically generated

Hình 2.1: Cấu trúc LSTM

Thông tin được giữ lại bởi các ô và các thao tác trong bộ nhớ được thực hiện bởi các cổng.Có ba cổng:

1. **Cổng quên:** Thông tin không còn hữu ích ở trạng thái ô sẽ bị loại bỏ bằng cổng quên. Hai đầu vào *xt* (đầu vào tại thời điểm cụ thể) và *ht-1* (đầu ra ô trước đó) được đưa vào cổng và nhân với ma trận trọng số, sau đó là phép cộng độ lệch. Kết quả được chuyển qua một hàm kích hoạt để đưa ra đầu ra nhị phân. Nếu đối với một trạng thái ô cụ thể, đầu ra là 0, phần thông tin sẽ bị quên và đối với đầu ra 1, thông tin được giữ lại để sử dụng trong tương lai.

A screenshot of a computer

Description automatically generated with low confidence

Hình 2.2: Cổng quên

1. **Cổng đầu vào:** Việc bổ sung thông tin hữu ích cho trạng thái ô được thực hiện bởi cổng đầu vào. Đầu tiên, thông tin được điều chỉnh bằng cách sử dụng hàm sigmoid và lọc các giá trị cần ghi nhớ tương tự như cổng quên sử dụng đầu vào *ht-1* và *xt*. Sau đó, một vectơ được tạo bằng cách sử dụng hàm *tanh* cho đầu ra từ -1 đến +1, chứa tất cả các giá trị có thể có từ ht-1 và *x\_t*. Cuối cùng, các giá trị của vectơ và các giá trị được điều chỉnh được nhân lên để thu được thông tin hữu ích.

A picture containing text, clock

Description automatically generated

Hình 2.3: Cổng đầu vào

1. **Cổng ra:** Nhiệm vụ trích xuất thông tin hữu ích từ trạng thái ô hiện tại để trình bày dưới dạng đầu ra được thực hiện bởi cổng xuất. Đầu tiên, một vectơ được tạo ra bằng cách áp dụng hàm tanh trên ô. Sau đó, thông tin được điều chỉnh bằng cách sử dụng hàm sigmoid và lọc theo các giá trị được ghi nhớ bằng cách sử dụng đầu vào *ht-1* và *xt*. Cuối cùng, các giá trị của vectơ và các giá trị được điều chỉnh được nhân lên để được gửi dưới dạng đầu ra và đầu vào cho ô tiếp theo.

Diagram

Description automatically generated

Hình 2.4: Cổng ra

## 2.2. Gated Memory Cell

Có thể cho rằng thiết kế của LSTM được lấy cảm hứng từ các cổng logic của máy tính. LSTM giới thiệu một ô nhớ (hay gọi tắt là ô) có hình dạng giống như trạng thái ẩn (một số tài liệu coi ô nhớ là một loại trạng thái ẩn đặc biệt), được thiết kế để ghi thêm thông tin. Để điều khiển ô nhớ, chúng ta cần một số cổng. Một cổng là cần thiết để đọc các mục nhập từ ô. Chúng tôi sẽ gọi đây là cổng đầu ra. Một cổng thứ hai là cần thiết để quyết định thời điểm đọc dữ liệu vào ô. Chúng tôi gọi đây là cổng đầu vào. Cuối cùng, chúng ta cần một cơ chế để thiết lập lại nội dung của ô, được quản lý bởi một cổng quên. Động lực cho một thiết kế như vậy cũng giống như các GRU, cụ thể là có thể quyết định khi nào cần nhớ và khi nào bỏ qua các đầu vào ở trạng thái ẩn thông qua một cơ chế chuyên dụng. Hãy để chúng tôi xem điều này hoạt động như thế nào trong thực tế.

**Input Gate, Forget Gate and Output Gate [5]**

Cũng giống như trong GRU, dữ liệu cấp vào cổng LSTM là đầu vào ở bước thời gian hiện tại và trạng thái ẩn của bước thời gian trước đó. Chúng được xử lý bởi ba lớp được kết nối đầy đủ với chức năng kích hoạt sigmoid để tính toán các giá trị của đầu vào. và các cổng đầu ra. Kết quả là, giá trị của ba cổng nằm trong khoảng (0, 1).

Diagram

Description automatically generated

Hình 2.5: Tính toán cổng đầu vào, cổng quên và cổng đầu ra trong mô hình LSTM

Về mặt toán học, giả sử rằng có các đơn vị ẩn h, kích thước lô là n và số lượng đầu vào là d. Do đó, đầu vào là Xt Rn.d và trạng thái ẩn của bước thời gian trước đó là Ht-1 Rn.h . Tương ứng, các cổng tại bước thời gian t được xác định như sau: cổng vào là It Rn.h , cổng quên là Ft Rn.h và cổng ra là Ot Rn.h . Chúng được tính như sau:

It = ᵟ(XtWxi + Ht-1Whi + bi)

Ft = ᵟ(XtWxf + Ht-1Whf + bf)

Ot = ᵟ(XtWxo + Ht-1Who + bo)

\* Candidate Memory Cell

Tiếp theo chúng ta thiết kế ô nhớ. Vì chúng tôi chưa xác định rõ hoạt động của các cổng khác nhau, nên trước tiên chúng tôi giới thiệu ô nhớ candidate  . Tính toán của nó tương tự như ba cổng được mô tả ở trên, nhưng sử dụng một hàm tanh có phạm vi giá trị (-1, 1) làm hàm kích hoạt. Điều này dẫn đến phương trình sau tại bước thời gian t:



Wxc  Rd.hvà Whc Rh.hlà các thông số trọng lượng và bc R1.h là một tham số thiên vị

Diagram

Description automatically generated

Hình 2. 6:Tính toán ô nhớ candidate trong mô hình LSTM

**Memory Cell**

Trong GRU, chúng ta có một cơ chế để quản lý việc nhập và quên (hoặc bỏ qua). Tương tự, trong LSTM, chúng ta có hai cổng dành riêng cho các mục đích như vậy: cổng đầu vào It điều chỉnh lượng dữ liệu mới mà chúng ta tính đến  và forget gate Ft giải quyết lượng nội dung ô nhớ cũ Ct-1 Rn.h mà chúng ta giữ lại. Sử dụng cùng một thủ thuật nhân theo chiều kim như trước đây, chúng ta đi đến phương trình cập nhật sau: 

Nếu cổng quên luôn xấp xỉ 1 và cổng vào luôn xấp xỉ 0, các ô nhớ trong quá khứ Ct-1 sẽ được lưu theo thời gian và chuyển sang bước thời gian hiện tại. Thiết kế này được giới thiệu để giảm bớt vấn đề gradient biến mất và để nắm bắt tốt hơn sự phụ thuộc phạm vi dài trong các chuỗi.

Diagram

Description automatically generated

Hình 2. 7:Tính toán ô nhớ trong mô hình LSTM

**Hidden State**

Cuối cùng, chúng ta cần xác định cách tính toán trạng thái ẩn Ht Rn.h . Đây là nơi mà cổng đầu ra phát huy tác dụng. Trong LSTM, nó chỉ đơn giản là một phiên bản kiểm soát tanh của ô nhớ. Điều này đảm bảo rằng các giá trị của Ht luôn nằm trong khoảng



Bất cứ khi nào cổng đầu ra xấp xỉ 1, chúng tôi chuyển tất cả thông tin bộ nhớ qua bộ dự đoán một cách hiệu quả, trong khi đối với cổng ra gần bằng 0, chúng tôi chỉ giữ lại tất cả thông tin trong ô nhớ và không thực hiện thêm xử lý nào.

Diagram

Description automatically generated

Hình 2. 8: Tính toán trạng thái ẩn trong mô hình LSTM

## 2.3. Ứng dụng

* Mô hình hóa ngôn ngữ
* Dịch máy
* Chú thích hình ảnh
* Thế hệ viết tay
* Chatbots trả lời câu hỏi

# 

# 3. GAN: Generative Adversarial Networks

## 3.1. Tổng quan về GAN

* **Khái niệm**

Ảnh mặt GAN sinh ra qua các năm, Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation, 2018

GAN là mô hình có khả năng sinh ra dữ liệu mới, dữ liệu sinh ra nhìn như thật nhưng không phải thật.

\* Cấu trúc [9]

Diagram

Description automatically generated

Hình 3.1: Cấu trúc mạng GAN [3]

Ta có thể chia Gans ra làm hai phần:

* Generator: nhận nhiệm vụ fake data sao cho giống thật nhất có thể (ma trận số, audio, hình nude…) và cố gắng lừa Discriminator để nó ko nhận ra.
* Discriminator: Kiểm định dữ liệu mà thằng Genarator đổ vào, phân biệt nó là fake hay real.

Generator và Discriminator đều là mạng neural network và cùng “cạnh tranh” nhau, Generator cố gắng tạo ra dữ liệu giống thật nhất có thể , còn Discriminator cố gắng phân biệt dữ liệu mà Genarator đổ vào và chứng minh nó là hàng fake sau đó thông báo lại cho Generator để nó cải thiện, cứ như thế quá trình này lặp đi lặp lại để generator có thể tạo ra sample hoàn hảo nhất mà Discriminator không thể phân biệt được.

Diagram

Description automatically generated

Hình 3.2: Mô hình Generative Adversarial Netword

Discriminative models: Các mô hình phân biệt ước lượng trực tiếp xác suất có điều kiện P (y | X) của nhãn y, với các giá trị đặc trưng trong X. Một ví dụ về mô hình Discriminative là hồi quy logistic. Các mô hình Discriminative chỉ có thể được sử dụng trong cài đặt được giám sát

Generative models: Các mô hình tổng quát ước tính xác suất chung P (X, y), là xác suất chung của một cá thể dữ liệu. Lưu ý rằng xác suất chung có thể được sử dụng để ước tính xác suất có điều kiện của y cho trước X bằng cách sử dụng quy tắc Bayes như sau:

A picture containing text, watch, clock

Description automatically generated

Mô hình Generative được sử dụng trong cả cài đặt được giám sát và không được giám sát.

## 3.2. Generator

Generator là mạng sinh ra dữ liệu, tức là sinh ra các chữ số giống với dữ liệu trong MNIST dataset. Generator có input là noise (random vector) là output là chữ số.

Tại sao input là noise? Vì các chữ số khi viết ra không hoàn toàn giống nhau. Ví dụ số 0 ở hàng đầu tiên có rất nhiều biến dạng nhưng vẫn là số 0. Thế nên input của Generator là noise để khi ta thay đổi noise ngẫu nhiên thì Generator có thể sinh ra một biến dạng khác của chữ viết tay. Noise cho Generator thường được sinh ra từ normal distribution hoặc uniform distribution.

Shape, arrow

Description automatically generated

Hình 3.3: MNIST dataset [7]

Input của Generator là noise vector 100 chiều. Sau đấy mô hình neural network được áp dụng với số node trong hidden layer lần lượt là 256, 512, 1024

Chart, box and whisker chart

Description automatically generated

Hình 3.4: Mô hình generator

Output layer có số chiều là 784, vì output đầu ra là ảnh giống với dữ liệu MNIST, ảnh xám kích thước 28\*28 (784 pixel). Output là vector kích thước 784\*1 sẽ được reshape về 28\*28 đúng định dạng của dữ liệu MNIST.

A picture containing chart

Description automatically generated

Hình 3.5: Ví dụ về reshape

## 3.3. Discriminator

Discriminator là mạng để phân biệt xem dữ liệu là thật (dữ liệu từ dataset) hay giả (dữ liệu sinh ra từ Generator). Trong bài toán này thì discriminator dùng để phân biệt chữ số từ bộ MNIST và dữ liệu sinh ra từ Generator. Discriminator có input là ảnh biểu diễn bằng 784 chiều, output là ảnh thật hay ảnh giả. Đây là bài toán binary classification, giống với logistic regression.

Chart, box and whisker chart

Description automatically generated

Hình 3.6: Mô hình đối xứng lại với Generator

## 3.4. Loss functions

Kí hiệu z là noise đầu vào của generator, x là dữ liệu thật từ bộ dataset. Kí hiệu mạng Generator là G, mạng Discriminator là D. G(z) là ảnh được sinh ta từ Generator. D(x) là giá trị dự đoán của Discriminator xem ảnh x là thật hay không, D(G(z)) là giá trị dự đoán xem ảnh sinh ra từ Generator là ảnh thật hay không. Vì ta có 2 mạng Generator và Discriminator với mục tiêu khác nhau, nên cần thiết kế 2 loss function cho mỗi mạng. Discriminator thì cố gắng phân biệt đâu là ảnh thật và đâu là ảnh giả.

Vì là bài toán binary classification nên loss function dùng giống với binary cross-entropy loss của bài sigmoid. d.compile(loss='binary\_crossentropy', optimizer=adam, metrics=['accuracy']) Giá trị output của model qua hàm sigmoid trong (0, 1) nên Discriminator sẽ được train để input ảnh ở dataset thì output gần 1, còn input là ảnh sinh ra từ Generator thì output gần 0, hay D(x) -> 1 còn D(G(z)) -> 0.

Diagram

Description automatically generated

Hình 3.7: Mô hình Loss function

Hay nói cách khác là loss function muốn maximize D(x) và minimize D(G(z)). Ta có minimize D(G(z)) tương đương với maximize (1 - D(G(z))). Do đó loss function của Discriminator trong paper gốc được viết lại thành.

Graphical user interface

Description automatically generated

**Loss Discriminator [6]**

E là kì vọng, hiểu đơn giản là lấy trung bình của tất cả dữ liệu, hay maximize D(x) với x là dữ liệu trong traning set.

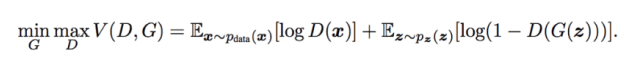
Generator sẽ học để đánh lừa Discriminator rằng số nó sinh ra là số thật, hay D(G(z)) -> 1. Hay loss function muốn maximize D(G(z)), tương đương với minimize (1 - D(G(z)))

Graphical user interface

Description automatically generated with medium confidence

Loss Generator [12]

Do đó ta có thể viết gộp lại loss của mô hình GAN



**Loss GAN**

Từ hàm loss của GAN có thể thấy là việc train Generator và Discriminator đối nghịch nhau, trong khi D cố gắng maximize loss V thì G cố gắng minimize V. Quá trình train GAN kết thúc khi model GAN đạt đến trạng thái cân bằng của 2 models, gọi là Nash equilibrium.

## 3.5. Ứng dụng

**Tạo ra khuôn mặt của con người**

Graphical user interface, application

Description automatically generated

Hình 3.8: Ảnh mặt GAN sinh ra qua các năm

**Chỉnh sửa hình ảnh**

Chắc mọi người vẫn nhớ tới FaceApp làm mưa làm gió trong thời gian vừa qua. Nó là một ứng dụng của GAN để sửa các thuộc tính của khuôn mặt như màu tóc, da, giới tính, cảm xúc hay độ tuổi

A collage of a person

Description automatically generated with medium confidence

Hình 3.9: StarGAN

A collage of a person's face

Description automatically generated with medium confidence

Hình 3.10: AgeGAN

**Tạo các nhân vật Anime**

Việc thuê các họa sĩ thiết kế các nhân vật anime rất đắt đỏ thế nên GAN được sử dụng để tự động sinh ra các nhân vật anime

A collage of a person's face

Description automatically generated

Hình 3.11: Ảnh anime được sinh ra

**Chuyển đồi hình ảnh sang hình ảnh**



Hình 3.12: Ví dụ ảnh draft sang ảnh màu

# TÀI LIỆU THAM KHẢO

1. Nguyễn Anh Tuấn, “Deep learning cơ bản”, <https://drive.google.com/file/d/1lNjzISABdoc7SRq8tg-xkCRRZRABPCKi/view>
2. Deep Learning based Vehicle Detection in Aerial, <https://library.oapen.org//bitstream/handle/20.500.12657/53149/9783731511137.pdf?sequence=1&isAllowed=y>
3. <https://viblo.asia/p/generative-adversarial-networksgan-va-ung-dung-cua-no-trong-deepfakes-jvElagwdKkw>
4. <https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/>
5. <https://d2l.ai/chapter_recurrent-modern/lstm.html>
6. Generative Adversarial Nets (neurips.cc), <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>
7. MNIST Reborn, Restored and Expanded: Additional 50K Training Samples | Synced (syncedreview.com), <https://syncedreview.com/2019/06/19/mnist-reborn-restored-and-expanded-additional-50k-training-samples/>
8. <https://machinelearningcoban.com/2017/04/09/smv/>