TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ───────



**BÀI TẬP LỚN**

Nhập môn Học máy và khai phá dữ liệu

**Đề tài: Nhận dạng chữ viết tay**

**sử dụng mạng CRNN**

Sinh viên thực hiện: Phạm Vũ Dũng - 20183508

Nguyễn Huy Hoàng - 20183592

Nguyễn Nhật Minh -20183592

Lớp: 118596 – Kỳ: 2020.2

Giáo viên hướng dẫn : ThS. Ngô Văn Linh

***Hà Nội, tháng 5 năm 2021***

# Mục lục

[Mục lục 2](#_Toc72870111)

[Danh sách từ viết tắt và thuật ngữ 3](#_Toc72870112)

[Danh sách bảng và hình ảnh 4](#_Toc72870113)

[1. Mở đầu 5](#_Toc72870114)

[1.1. Đặt vấn đề 5](#_Toc72870115)

[1.2. Giải pháp 5](#_Toc72870116)

[2. Kiến thức liên quan 5](#_Toc72870117)

[2.1. Các hàm kích hoạt cơ bản 5](#_Toc72870118)

[2.2. Mạng nơ-ron tích chập CNN 7](#_Toc72870119)

[2.3. Mạng nơ-ron hồi quy RNN 9](#_Toc72870120)

[2.4. Phương pháp Connectionist Temporal Classification 11](#_Toc72870121)

[2.5. Phương pháp tối ưu ADAM 11](#_Toc72870122)

[3. Dữ liệu đầu vào 14](#_Toc72870123)

[4. Mô hình đề xuất 15](#_Toc72870124)

[4.1. Mô hình tổng quát 15](#_Toc72870125)

[4.2. Giải thích thành phần 16](#_Toc72870126)

[4.3. Hàm Loss và Backprob 17](#_Toc72870127)

[5. Thí nghiệm và kết quả 20](#_Toc72870128)

[5.1. Thí nghiệm với epoch 20](#_Toc72870129)

[5.2. Thí nghiệm với số lượng tầng ẩn. 22](#_Toc72870130)

[5.3. Thí nghiệm với số lượng của tập dữ liệu 23](#_Toc72870131)

[6. Kết luận và hướng phát triển 24](#_Toc72870132)

[Danh mục tài liệu tham khảo 25](#_Toc72870133)

# Danh sách từ viết tắt và thuật ngữ

|  |  |
| --- | --- |
| BTL | Bài tập lớn |
| ML | Machine Learning (Học máy) |
| DL | Deep Learning (Học sâu) |
| CNN | Convolutional Neural Network (Mạng nơ-ron tích chập) |
| RNN | Recurrent Neural Network (Mạng nơ-ron hồi quy) |
| CTC | Connectionist Temporal Classification |
| ADAM | Adaptive Moment Estimation |
| LSTM | Long-Short Term Memory |
| BiLSTM | Bidirectional LSTM |
| GD | Gradient Descent |
| SGD | Stochatic Gradient Descent |
| PSD | Prefix Search Decoding |
| MLP | Multilayer Perceptron |

# Danh sách bảng và hình ảnh

[Hình 1. Đồ thị hàm Sigmoid 6](#_Toc72870134)

[Hình 2. Hướng giảm của hàm sigmoid 6](#_Toc72870135)

[Hình 3. Đồ thị hàm tanh 7](#_Toc72870136)

[Hình 4. Minh họa về mạng CNN 8](#_Toc72870137)

[Hình 5. Minh họa về tầng Convolution 8](#_Toc72870138)

[Hình 6. Minh họa về tầng ReLU 9](#_Toc72870139)

[Hình 7. Minh họa về tầng Pooling 9](#_Toc72870140)

[Hình 8. Mô hình RNN tổng quát 9](#_Toc72870141)

[Hình 9. Mô hình kiến trúc LSTM 10](#_Toc72870142)

[Hình 10. Mô hình kiến trúc Bidirectional LSTM 10](#_Toc72870143)

[Hình 11. Minh họa về Gradient Descent 11](#_Toc72870144)

[Hình 12. Minh họa về momentum 12](#_Toc72870145)

[Hình 13. Momentum và cực tiểu địa phương. 12](#_Toc72870146)

[Hình 14. Ví dụ về Bias Correction 14](#_Toc72870147)

[Hình 15. Dữ liệu đầu vào 14](#_Toc72870148)

[Hình 16. Mô hình tổng quát 15](#_Toc72870149)

[Hình 17. Sơ đồ tầng CNN 16](#_Toc72870150)

[Hình 18. Sơ đồ tầng RNN 17](#_Toc72870151)

[Hình 19. Minh họa về đầu vào CTC 18](#_Toc72870152)

[Hình 20. Minh họa về CTC Blank 19](#_Toc72870153)

[Hình 21. Sự phụ thuộc của độ chính xác theo số Epoch 21](#_Toc72870154)

[Hình 22. Sự phụ thuộc của hàm Loss theo số Epoch 21](#_Toc72870155)

[Hình 23. Sự phụ thuộc của độ chính xác theo số tầng CNN 22](#_Toc72870156)

[Hình 24. Sự phụ thuộc của độ chính xác theo số tầng RNN 22](#_Toc72870157)

[Hình 25. Sự phụ thuộc của độ chính xác theo kích thước dữ liệu 23](#_Toc72870158)

[Hình 26. Kết quả trực quan 23](#_Toc72870159)

# Mở đầu

## Đặt vấn đề

Trong quá trình học tập và trao đổi thông tin, nhiều lúc chúng ta sẽ phải ghi chép ra giấy, bảng, hay bất kỳ bề mặt nào bằng chữ viết tay thông thường. Khi muốn số hóa những ký tự hay những từ đó thì thường ta phải nhập bằng tay vào máy vì chưa có cách thông dụng nào thay thế việc đó. Việc nhận dạng các ký tự ở ngoài đời thực (Optical Character Recognition) để xử lý trên máy tính không còn là vấn đề mới nhưng vẫn có tiềm năng phát triển cao và ứng dụng rộng rãi.

## Giải pháp

Đã có rất nhiều bài nghiên cứu sử dụng Deep Learning vào OCR như nhận dạng từ trong ảnh[1], nhận dạng biển số xe[2] và đã có cả nhận dạng chữ viết tay[3]. Bài nghiên cứu này cải tiến từ mô hình đó. Cụ thể là sử dụng kết hợp hai mạng CNN và RNN để tạo ra mô hình CRNN giải quyết bài toán nhận dạng chữ viết tay với độ chính xác cao hơn.

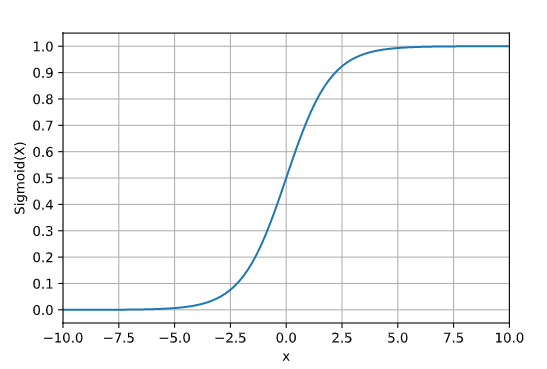
Câu hỏi đặt ra là tại sao không nhận dạng từng ký tự sau đó ghép lại thành từ thay vì nhận dạng cả từ. Việc nhận dạng từng ký tự riêng đòi hỏi chi phí đánh nhãn lớn vì phải vẽ boundary box cho từng ký tự. Ngoài ra chữ viết tay tường không được gọn gàng dẫn đến boundary của các từ bị đè lên nhau hoặc không phân biệt rõ ràng. Do đó nhận dạng từng từ là phương pháp hợp lý và dễ mở rộng cho các dự án sau trong tương lai.

# Kiến thức liên quan

## Các hàm kích hoạt cơ bản

Hàm kích hoạt (activation function) đóng vai định nghĩa giá trị và tỉ lệ đầu ra của một nút trong mạng nơ-ron nhân tạo. Các hàm kích hoạt thường là các hàm phi tuyến, vì nếu chỉ sử dụng hàm tuyến tính thì việc xếp chồng các lớp nơ-ron lên nhau là vô nghĩa.

**Hàm Sigmoid**



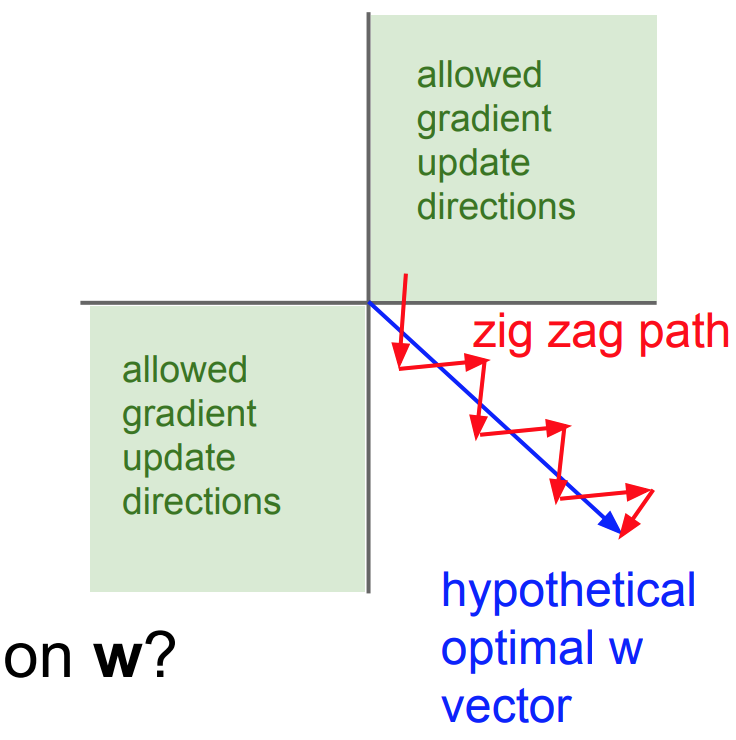
Hình 1. Đồ thị hàm Sigmoid

Hàm Sigmoid nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (0;1) (*Hình 1*). Đầu vào là số thực âm rất nhỏ sẽ cho đầu ra tiệm cận với 0, ngược lại, nếu đầu vào là một số thực dương lớn sẽ cho đầu ra là một số tiệm cận với 1. Trong quá khứ hàm Sigmoid hay được dùng vì có đạo hàm rất đẹp.

Vấn đề của sigmoid là không có trung tâm tại 0 gây khó khăn trong việc hội tụ đến điểm cực tiểu trong quá trình backpropagation. Giả sử đầu vào của L2 và là đầu ra của L1 là x.

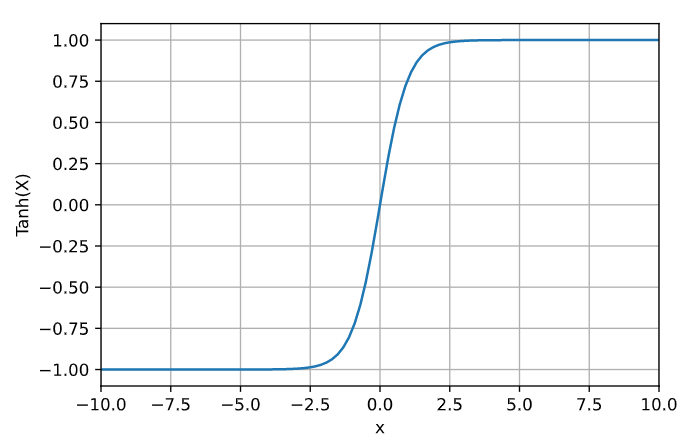
L2 = f()

Có đạo hàm , nếu x là một hàm sigmoid thì x > 0. Điều này dẫn đến các gradient cùng dấu, khi đó các giá trị w sẽ cùng tăng(hướng đông-bắc) hoặc cùng giảm(hướng tây-nam), nếu phải đi theo hướng đông-nam thì sẽ đi theo đường zig-zac như *Hình 2.*



Hình 2. Hướng giảm của hàm sigmoid

**Hàm Tanh**



Hình 3. Đồ thị hàm tanh

Hàm Tanh nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (-1;1) (Hình 1). Cũng như Sigmoid, hàm Tanh bị bão hoà ở 2 đầu (gradient thay đổi rất ít ở 2 đầu). Điều này đồng nghĩa với việc các hệ số tương ứng với unit đang xét sẽ gần như không được cập nhật (còn được gọi là vanishing gradient). Ta giải quyết vấn đề này bằng batch-normalization. Hàm tanh còn khắc phục được vấn đề không tập trung tại điểm 0 của hàm sigmoid.

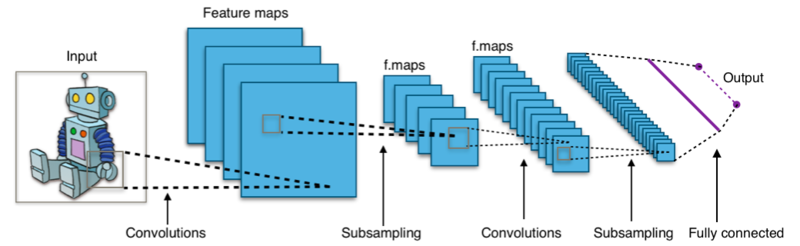
**Hàm Softmax**

Hàm softmax được sử dụng rộng rãi trong các bài toán classification. Hàm softmax mô hình hóa xác suất mỗi phần tử trong một vectơ. Đầu vào là một vectơ các giá trị đặc trưng, đầu ra cũng là một vectơ có kích thước tương đương. Tổng các giá trị đầu ra bằng 1 và luôn dương và quan trọng giá trị đặc trưng càng lớn thì xác suất càng cao.

## Mạng nơ-ron tích chập CNN

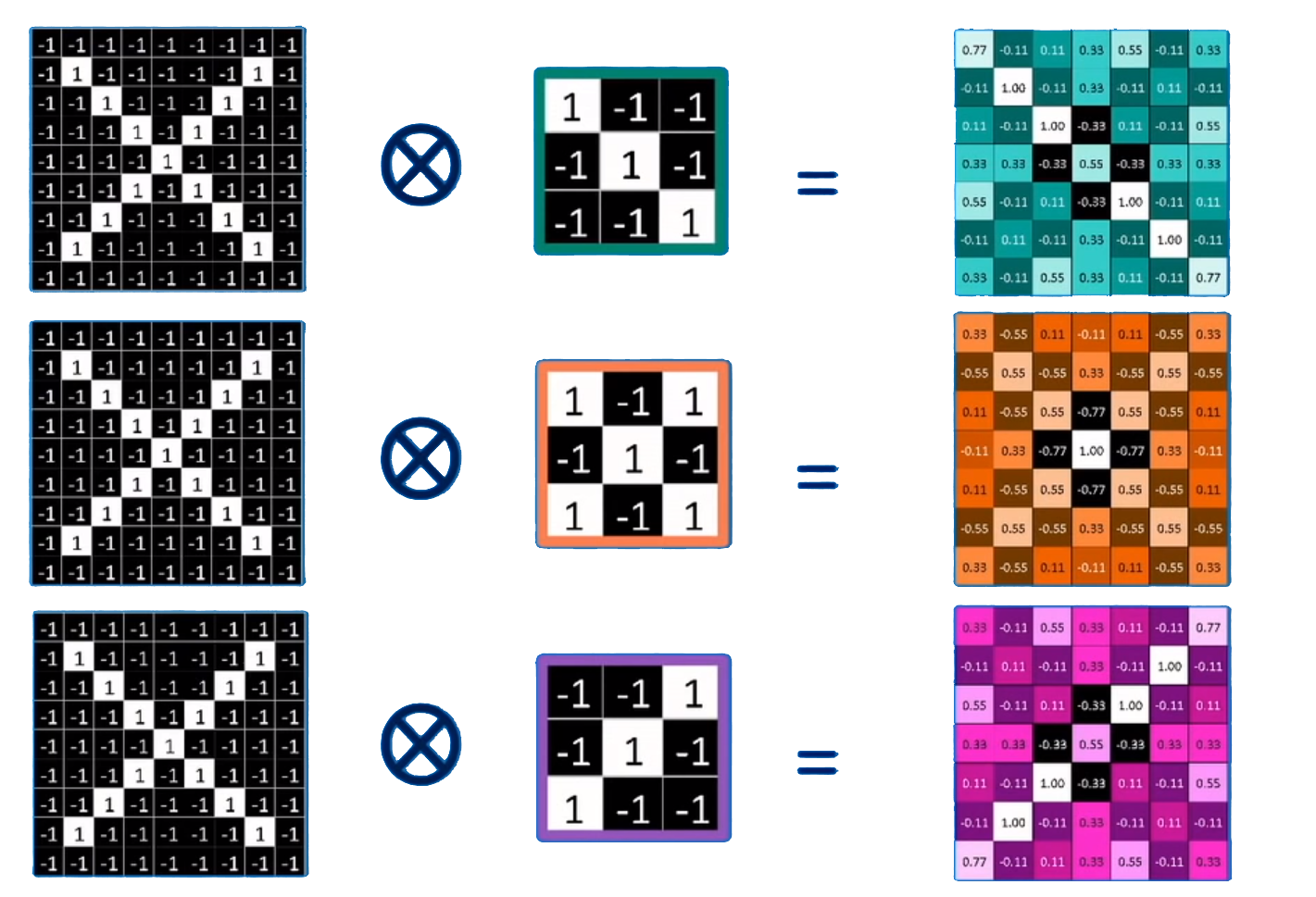
Mạng nơ-ron tích chập (Convolution Neural Network) được dùng để tìm ra các đặc trưng cục bộ của ảnh. Đưa ảnh về ma trận và sử dụng các filter(kernel) để tạo ra để tạo ra các ma trận mới với các đặc tính được tìm ra và đánh giá rõ hơn.

Các tầng ẩn của mạng rơ-ron bao gồm hai loại tầng chính là Convolution và Pooling, có thể thêm sau tầng Convolution một tầng ReLU để chuẩn hóa dữ liệu cho tầng Convolution sau đó. Cuối cùng là một hoặc nhiều tầng fully connected.

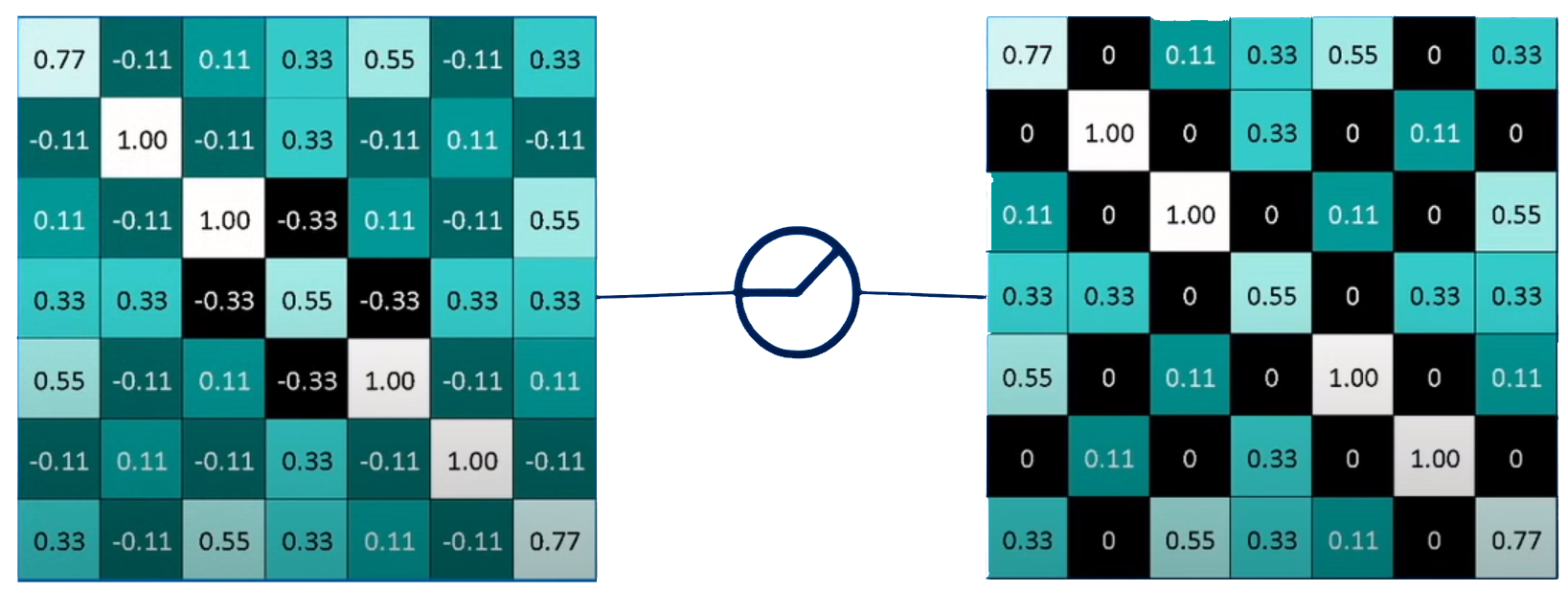


Hình 4. Minh họa về mạng CNN

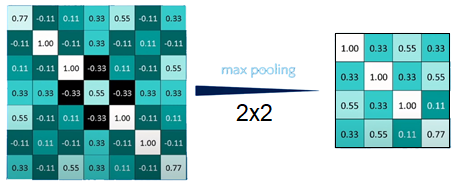
Lặp lại các tầng Convolution và Pooling ta sẽ có được mạng CNN. Qua quá trình học mạng sẽ tối ưu các trọng số được sử dụng. Mạng CNN được áp dụng rộng trong xử lý ảnh, video, xử lý ngôn ngữ tự nhiên và dữ liệu có tính cấu trúc cao. Tầng convolutional mô hình hóa các đường nét, độ đậm nhạt của ảnh và nhiều chi tiết khác, càng nhiều tầng convoluitional thì càng khó cho con người có thể hiểu. Tầng Pooling để giảm kích thước của dữ liệu giảm thiểu overfiting mà không làm mất mát nhiều thông tin.



Hình 5. Minh họa về tầng Convolution



Hình 6. Minh họa về tầng ReLU

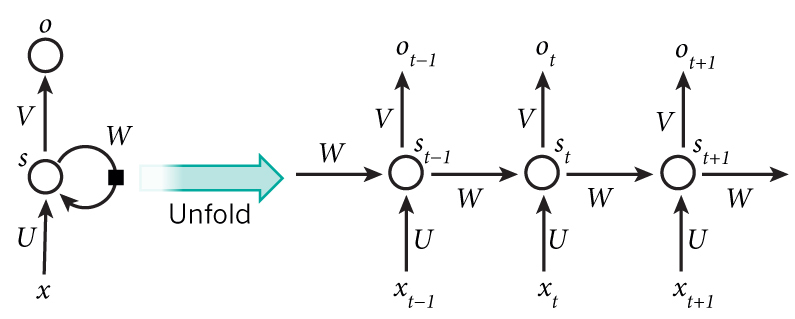


Hình 7. Minh họa về tầng Pooling

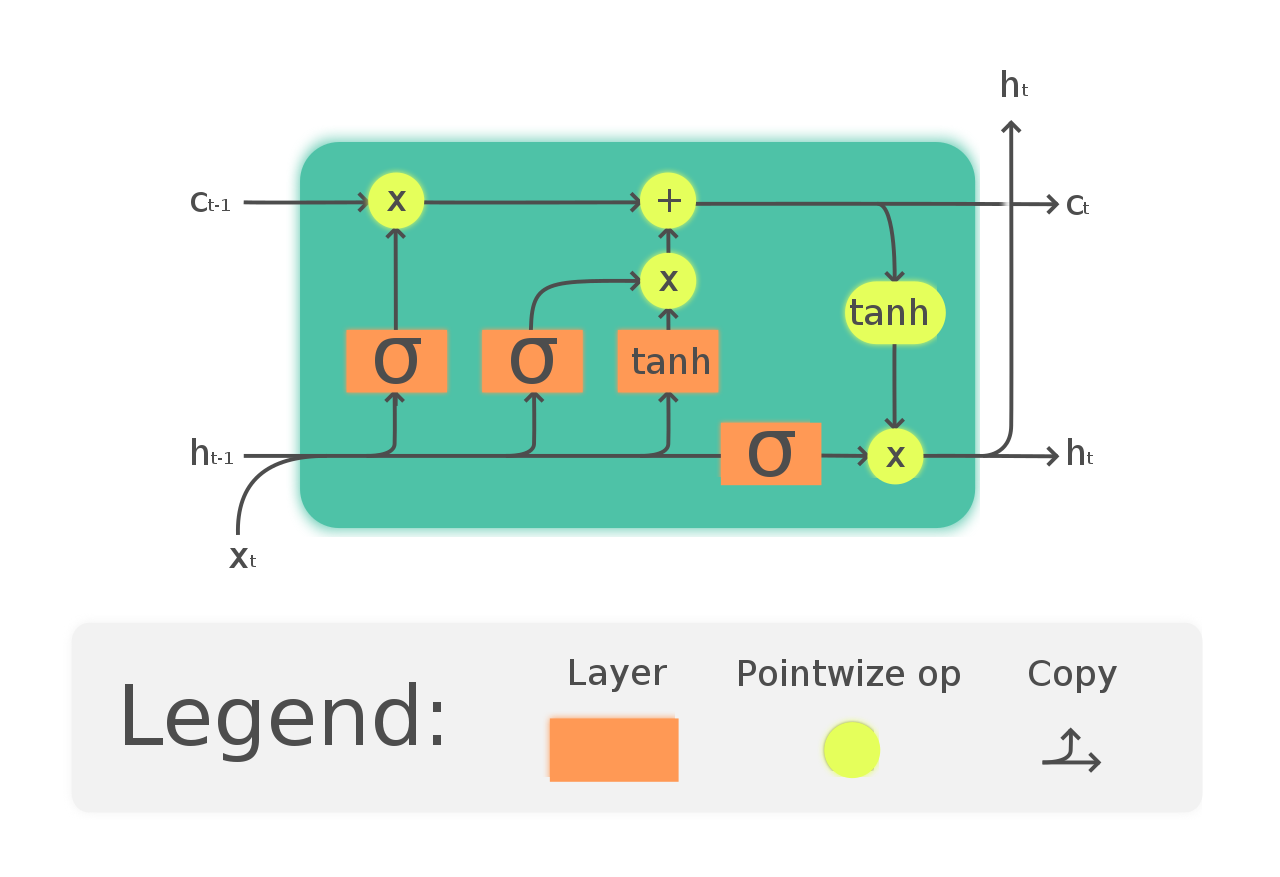
## Mạng nơ-ron hồi quy RNN

Mạng RNN là một phiên bản mở rộng của ANN sử dụng đồ thị có hướng giữa các nút áp dụng cho việc xử lý dữ liệu đầu vào có dạng chuỗi. Mạng trích xuất được các đặc tính trình tự liên tục, mỗi phần tử của chuỗi nằm ở một bước thời gian(time step) và có quan hệ về mặt thứ tự với nhau. Mục đích của mạng RNN là mô hình hóa quan hệ thứ tự giữa các phần tử trong chuỗi.

Mỗi nút trong tầng ẩn sẽ có 1 đầu vào từ input ở bước hiện tại và 1 đầu ra tương ứng, ngoài ra còn có thêm 1 đầu vào từ nút liền trước nó trong đồ thị có hướng.



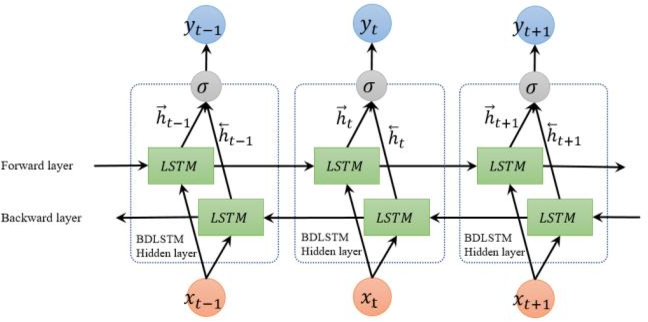
Hình 8. Mô hình RNN tổng quát



Hình 9. Mô hình kiến trúc LSTM

Kiến trúc LSTM được đề xuất để xử lý vấn đề vanishing gradient, phù hợp với các bài toán phân lớp, xử lý và đưa ra lựa chọn dựa trên dữ liệu dạng chuỗi, đặc biệt là khi có sự xuất hiện các sự kiện nhiễu trong chuỗi thời gian. Mỗi nút của LSTM sẽ có 3 luồng vào và 3 luồng ra được phân làm 3 cổng với 1 memory cell. Cổng Forget có nhiệm vụ chọn lọc (quên) lượng thông tin từ nút trước, cổng Input chọn lượng thông tin từ input vào nút, memory cell của nút hiện tại sẽ tổng hợp thông tin từ 2 cổng Forget và Input với memory cell của nút trước, cuối cùng thông tin được tổng hợp lại và đưa ra cổng Output.

Vấn đề của directed graph trong LSTM là chúng ta chỉ có thể biết được dữ liệu trong nút thời gian ở quá khứ nhưng lại không biết được trong tương lai. Trong một chuỗi có thứ tự chặt chẽ thì phần tử ở mỗi nút sẽ liên quan đến cả trước và sau nó. Giải quyết vấn đề này bằng Bidirectional LSTM, thêm một luồng directed graph nữa ngược chiều, như vậy sẽ có 2 luồng Forward layer và Backward layer. Thông tin output sẽ được tổng hợp từ cả 2 luồng qua một hàm sigmoid.



Hình 10. Mô hình kiến trúc Bidirectional LSTM

## Phương pháp Connectionist Temporal Classification

Khi xử lý tín hiệu có dạng chuỗi, đặc biệt là dữ liệu có sự xuất hiện của các sự kiện trống hay sự kiện nhiễu, kết quả output chúng ta thu được thường không được mượt và bị lặp từ đối với ký tự hay bị kéo dài đối với dạng âm thanh. Do đó chúng ta cần phải căn chỉnh lại đầu ra, đây được gọi là *alignment problem.*

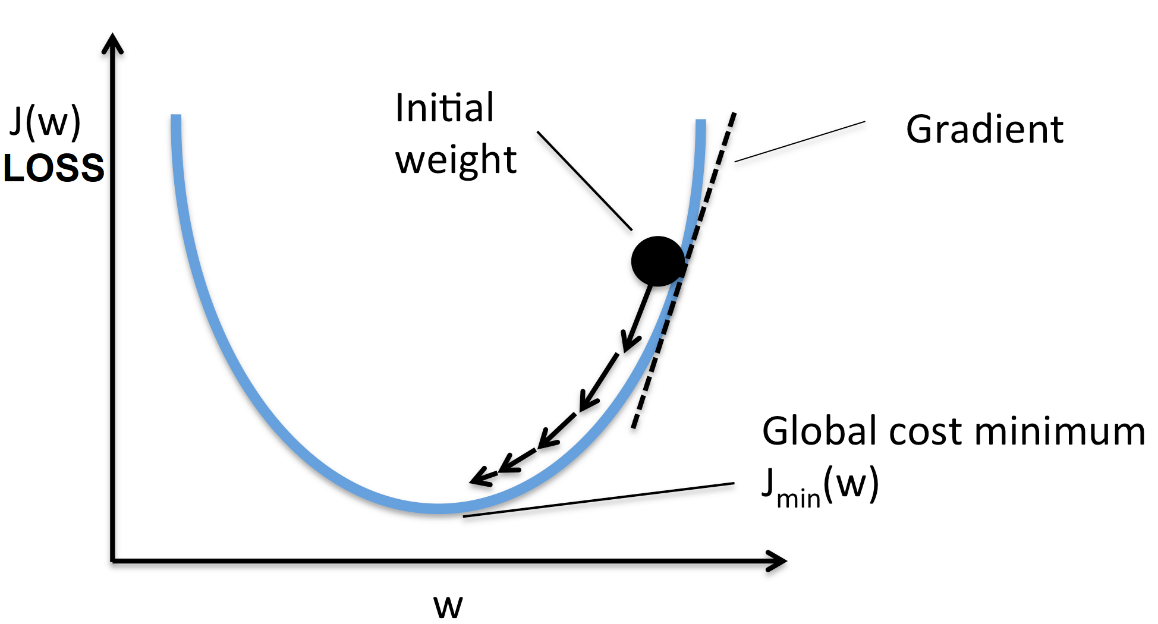
Ta có tập dữ liệu S bao gồm các cặp chuỗi (x, y), chuỗi y là nhãn của chuỗi x. Ta gọi một hàm temporal calssifier h: X → Z để xử lý chuỗi đầu vào (input sequence) x = (x1, x2, …, xT) và đưa ra chuỗi đầu ra z = (z1, z2,…, zU) với U ≤ T. Nhiệm vụ đặt ra là phải huấn luyện h để tối thiểu hóa một lượng mất mát nào đó giữa *z* và *y*.

Đầu vào của CTC là kết quả của lớp softmax có số unit tương đương với số instant ta định nghĩa, mỗi unit chứa xác suất xuất hiện của instant tại vị trí đó. Cụ thể về cách hoạt động và hàm Loss sẽ trình bày ở phần **4.3.**

## Phương pháp tối ưu ADAM

ADAM viết tắt của Adaptive Moment Estimation, là phương pháp tối ưu hàm Loss được xem là state-of-the-art. ADAM là sự kết hợp của Momentum và RMSProb.

Gradient Descent là phương pháp tìm hướng giảm của hàm số, từ đó thay đổi giá trị đầu vào để hàm số tiến tới cực trị địa phương. Nếu tìm cực tiểu thì phải đi ngược hướng GD, tức là ngược hướng đạo hàm.



Hình 11. Minh họa về Gradient Descent

**Công thức của Gradient Descent:**

w = w - *lr* . ∇wJ(w)

w : trọng số của mô hình

*lr* : learning rate – tốc độ học, đại diện cho độ lớn của các bước nhảy.

∇wJ(w) : vectơ gradient của hàm loss J(w) với w.

Dấu “-“ thể hiện việc đi ngược hướng với vectơ gradient.

Nếu sử dụng GD thường thì ∇wJ(w) khá cồng kềnh khi mà nó được tính bởi toàn bộ epoch, nếu sử dụng Stochatic Gradient Descent thì thời gian tính toán khá lâu vì phải update lại trọng số mỗi khi xử lý một input. Ta sử dụng mini-batch GD để khắc phục cả 2 nhược điểm trên.

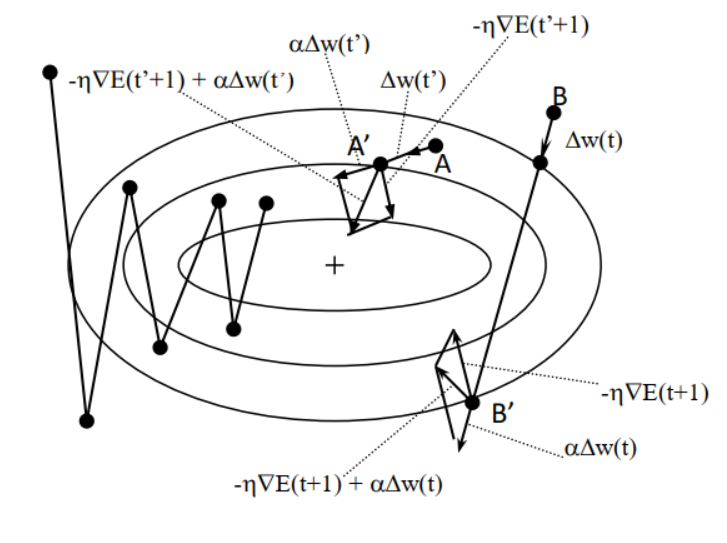
Vấn đề của mini-batch GD là hướng giảm sẽ bị nhiễu nhiều và dao động mạnh. Ta đưa vào một thành phần momentum để lưu các giá trị gradent của các bước nhảy trước, từ đó triệu tiêu được giao động và đưa vectơ hướng thẳng về điểm cực tiểu.

**Công thức của momentum:**

V = α.V + (1-α). ∇wJ(w)

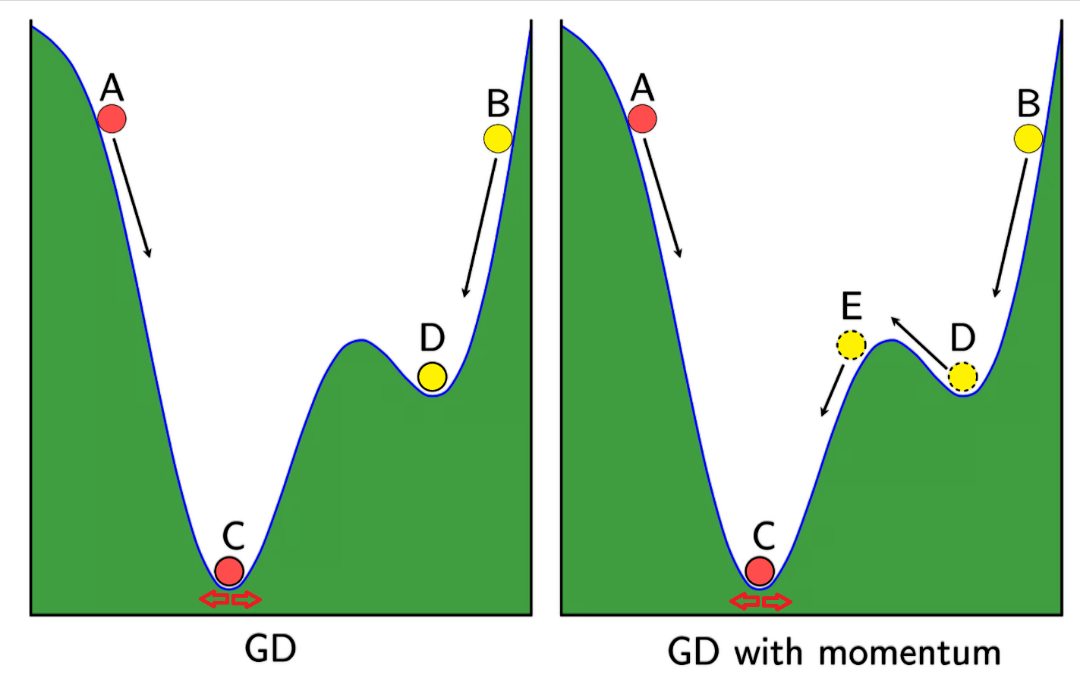
w = w – lr . V

α : tham số momentum (thường lấy α = 0.9)



Hình 12. Minh họa về momentum

Khi cho thêm vào momentum, hướng giảm sẽ có quán tính, giúp ích trong việc vượt qua các đồi nhỏ, bỏ qua cực tiểu địa phương và tìm được phương án tối ưu hơn. Vì có quán tính nên khi tìm được cực tiểu toàn cục sẽ mất vài lần dao động trước khi dừng lại hẳn.



Hình 13. Momentum và cực tiểu địa phương.

Vấn đề thứ hai của mini-batch GD là tốc độ học cố định. Chiến thuật ban đầu của ta sẽ là thay đổi tốc độ học từ nhanh dần sau đó giảm dần khi tiến gần đến cực tiểu.

**Công thức của GD theo learning-rate giảm dần:**

S = S + (∇wJ)2

**w = w -** ∇wJ(w)

Ta thấy sau càng nhiều vòng lặp, S càng tăng, khi đó learning-rate sẽ càng giảm việc lấy bình phương của ∇wJ để được một số luôn dương khiến cho S tăng dần. Tham số 𝜀 được thêm vào để tránh trường hợp lỗi chia cho 0 (thường lấy 𝜀=10-8). Nhưng vấn đề ở đây là learning-rate sẽ giảm dần và tiến về 0, còn gọi là hiện tượng vanishing gradient. Để khắc phục thì chúng ta sử dụng Exporential Weight Average, sử dụng tham số β như với momentum.

**Công thức của RMSProb:**

S = β.S + (1- β). (∇wJ)2

**w = w -** ∇wJ(w)

Khi này learning-rate không chỉ giảm mà còn có thể tăng để thích nghi với việc tính toán, thường lấy β=0.99. Như vậy kết hợp 2 phương pháp trên ta sẽ có Adaptive Moment Estimation.

**Công thức của ADAM:**

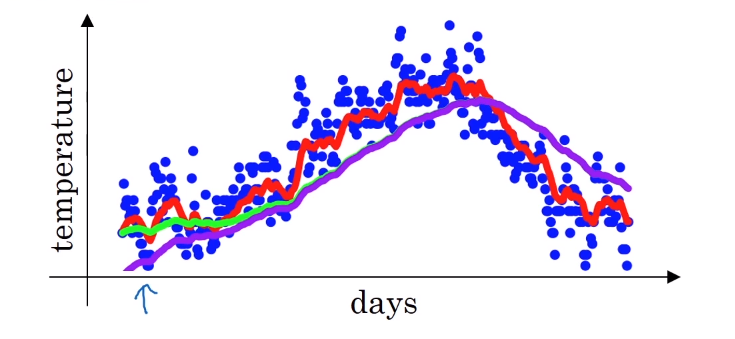
V = α.V + (1-α). ∇wJ(w)

Vcorrect =

S = β.S + (1- β). (∇wJ)2

**w = w -** Vcorrect

Trong đó Vcorrect là tham số của Bias Correction, tránh cho việc các giá trị ở các vòng lặp đầu quá bị phụ thuộc vào giá trị khởi tạo. Như hình dưới đây, nếu không dùng Bias Correction thì ta sẽ được đường màu tím còn nếu dùng Bias Correction thì ra sẽ thu được đường màu xanh lá. Qua vài vòng lặp đầu thì tiến dần về 0, khi đó V­correct ≈ V.

****

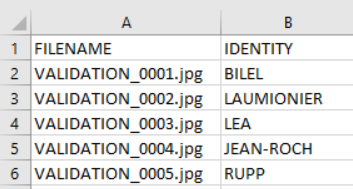
Hình 14. Ví dụ về Bias Correction

# Dữ liệu đầu vào

Tập dữ liệu được tải về từ bộ dữ liệu [Handwriting Recognition | Kaggle](https://www.kaggle.com/landlord/handwriting-recognition):

<https://www.kaggle.com/landlord/handwriting-recognition>

Bộ dữ liệu gồm các bức ảnh về chữ viết tay tên người gồm các chữ cái alphabet, trong đó train Data gồm 300,000 ảnh và validation gồm 30,000 ảnh. Nhãn của ảnh là tên gồm chữ cái latin được lưu trong các file .csv, mỗi ảnh tương đương với 1 nhãn tên.



Hình 15. Dữ liệu đầu vào

Dưới đây là một số bức ảnh trong tập validation.

C:\Users\hoang\Desktop\python\OCR_with_CRNN\input\validation_v2\validation\VALIDATION_0001.jpgVALIDATION\_0001.jpg:

C:\Users\hoang\Desktop\python\OCR_with_CRNN\input\validation_v2\validation\VALIDATION_0002.jpg

VALIDATION\_0002.jpg:

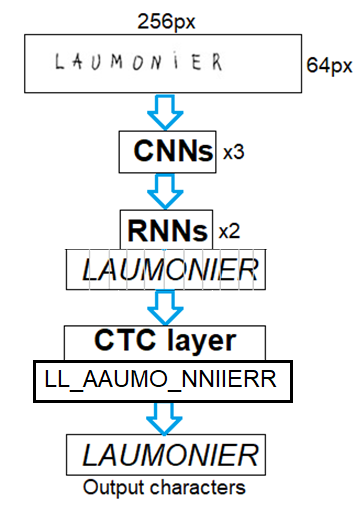
C:\Users\hoang\Desktop\python\OCR_with_CRNN\input\validation_v2\validation\VALIDATION_0003.jpg

VALIDATION\_0003.jpg:

# Mô hình đề xuất

## Mô hình tổng quát

Mô hình dưới đây sử dụng 3 tầng CNN nối tiếp sau là 2 tầng BiLSTM, cuối dùng là sử dụng 1 tầng fully connected để xử lý CTC. Tầng CNN tối ưu hơn Multilayer perceptron (MLP) vì lấy ra được đặc trưng theo chiều của ảnh và MLP là mạng fully connected dễ dẫn đến overfiting. Tầng CNN sẽ trích xuất ra các đặc trưng bên trong của ảnh như các nét của từ, độ đậm nhạt và qua nhiều tầng sẽ có được nhiều đặc trưng khác nhau khó có thể để cho con người hiểu được. Tầng RNN hay tầng BiLSTM sẽ mô hình hóa quan hệ chuỗi của từng phần tử. Từng nút trong LSTM nhận từ đầu vào từng cột được tổng mợp trong ma trận các khung hình của CNN. Tác dụng của batch-normalization là chuẩn hóa giá trị đầu ra sau khi đi qua lớp activation, tránh việc rơi vào điểm bão hòa của các hàm phi tuyến. Việc giá trị không phân phối trong khoảng ổn định (quá to hoặc quá nhỏ) sẽ ảnh hưởng đến quá trình tối ưu của mạng. Batch-normalization còn giảm thiểu overfiting, không còn phải sử dụng dropout quá nhiều gây mất mát thông tin.

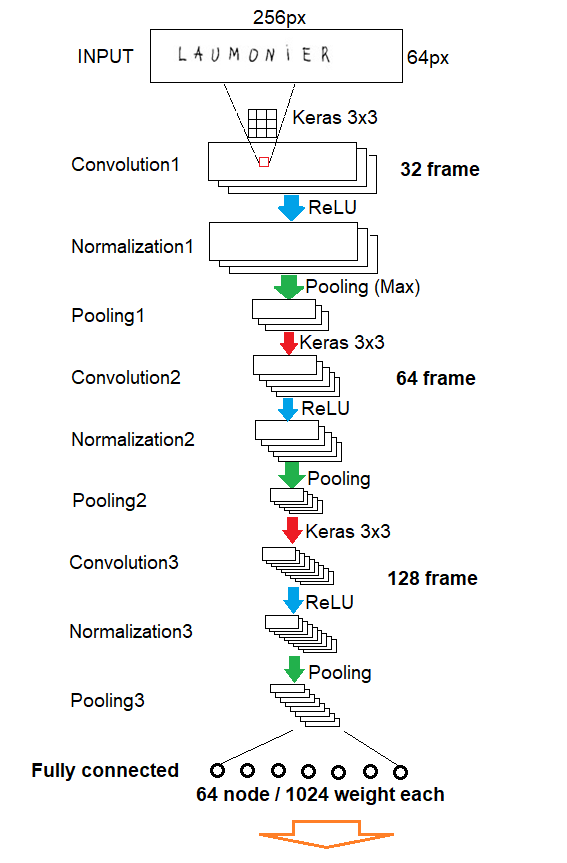


Hình 16. Mô hình tổng quát

## Giải thích thành phần

Ảnh đen trắng (1 chiều) sẽ được cắt đưa về kích thước 256x64 pixel, sau đó được chuẩn hóa về ma trận kích thước 256x64.

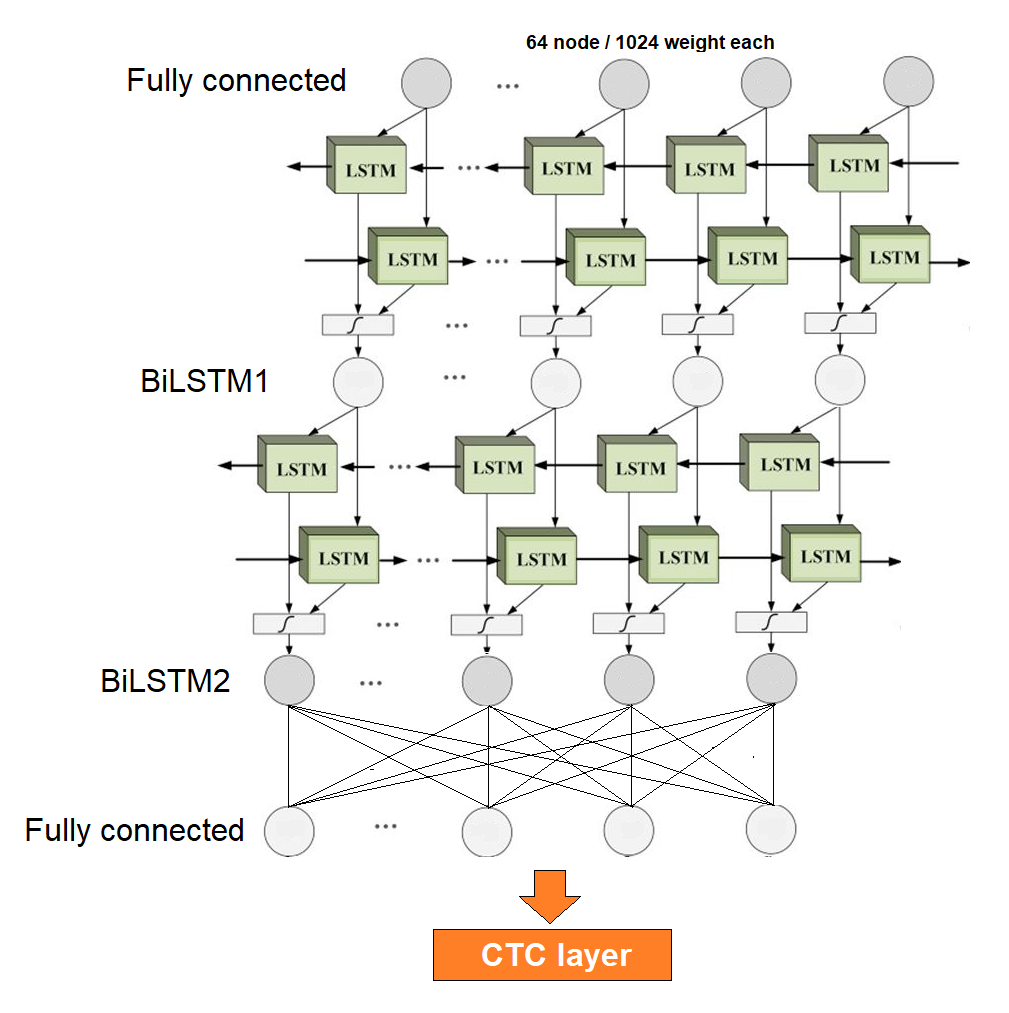
**Tầng Convolution Neural Network:**



Hình 17. Sơ đồ tầng CNN

Kích thước của frame sau khi Pooling1 còn 128x32, sau khi Pooling2 còn 64x16, sau khi Pooling3 còn 64x8. Trước khi chuyển sang RNN thì ra tạo một tầng fully connected để trải dữ liệu ra thành chuỗi. Sau mỗi tầng pooling mô hình sẽ drop 30% trọng số để tránh vấn đề overfiting.

**Tầng Recurrent Neural Network:**

****

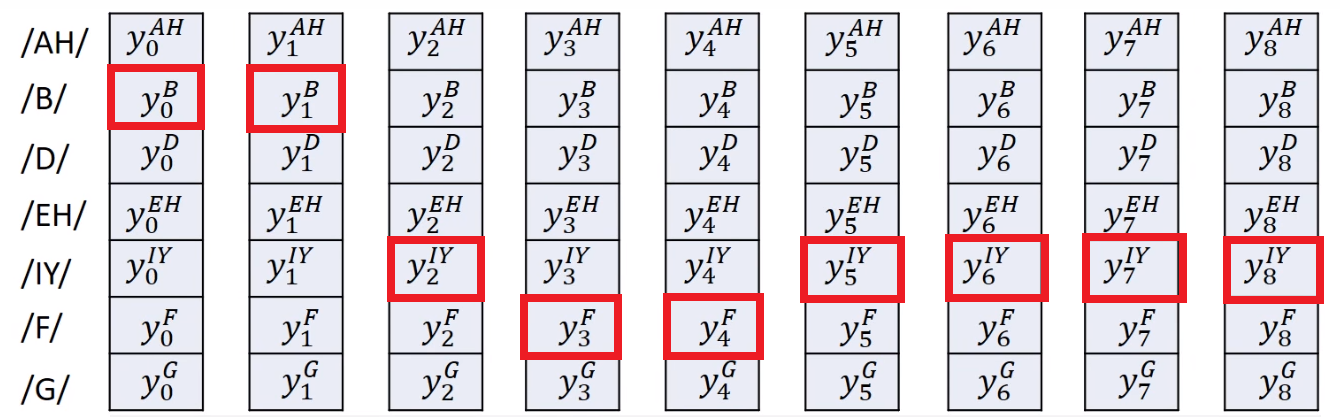
Hình 18. Sơ đồ tầng RNN

Cả tầng fully connected và LSTM đều có số nút là 64, số unit được chọn trong mỗi nút LSTM là 256, kết quả output của mỗi tầng là một hàm sigmoid. Ở tầng fully connected cuối cùng có 30 nút, tượng trưng cho 30 ký tự, mỗi nút sẽ có 64 unit tương đương với 64 vị trí cắt dọc theo ảnh. 30 ký tự bao gồm 26 ký tự alphabet, tự gạch ngang ” - “, ký tự khoảng trống “ “, ký tự nháy đơn” ‘ “ và ký tự CTC blank. Kết quả sau đó được xử lý CTC.

## Hàm Loss và Backprob

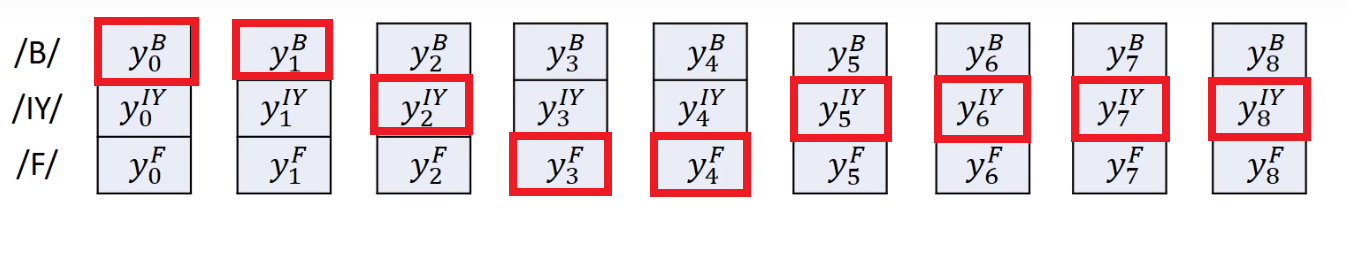
Phần này sẽ nối tiếp phần 2.3.

Đầu vào mỗi time step sẽ có 30 unit tương đương với 30 ký tự. Như vậy ta có ma trận 64x30, mỗi giá trị yt,k trong ma trận tượng trưng cho xác suất xuất hiện từ k tại vị trí t. Như vậy xác suất để từ 0 đến t thu được một chuỗi 𝜆 có độ dài t là: p(𝜆|x) = . Sử dụng Prefix search decoding (PSD) để tính hàm phân loại h(x), phương pháp này dựa trên thuật toán forwarding-backwarding .

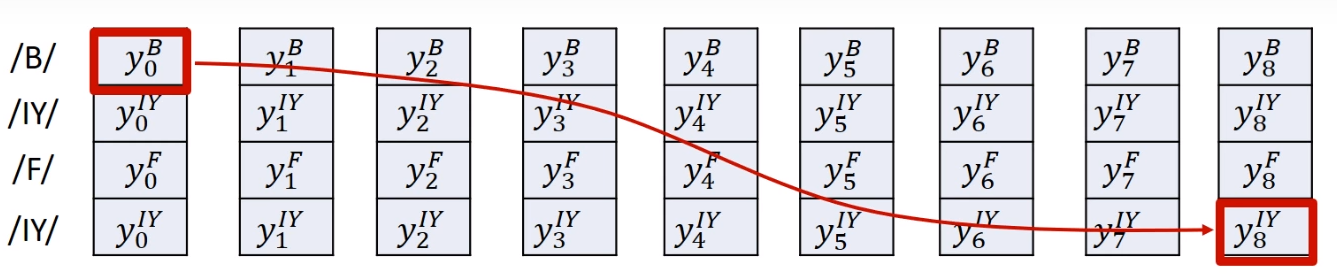


Hình 19. Minh họa về đầu vào CTC

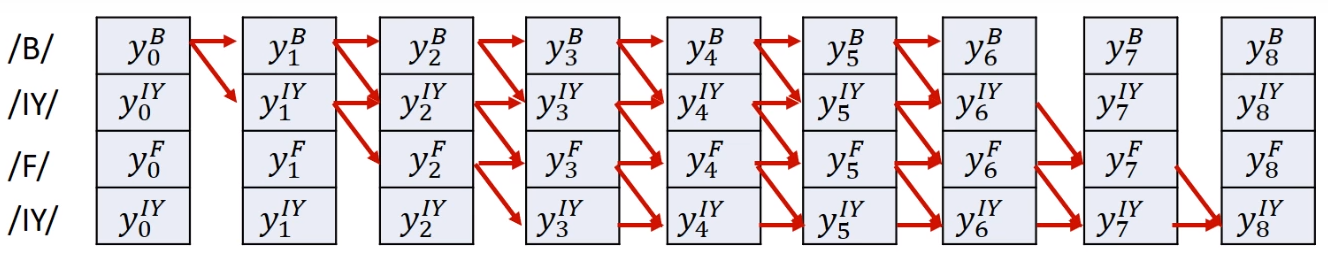
Hình trên cho ta ví dụ về input của CTC với 8 ký tự và T=9 time step, các ô tô được đánh dấu trên mỗi time step là từ có khả năng xuất hiện cao nhất tại vị trí đó.



Ta lấy ra các từ có khả năng xuất hiện trong chuỗi.



Ta lặp lại các hàng đối với từ xuất hiện nhiều lần, để sao cho khi nối các ô được đánh dấu, ta được đường đi từ ô trên cùng bên trái đến ô dưới cùng bên phải, ta gọi là path.



Từ mỗi ô, ta chỉ có thể đi xuống dưới để sang ký tự tiếp theo hoặc đi sang ngang tương đương với ký tự bị lặp.

Khi forwarding, định nghĩa hàm α [t][k] là tổng xác suất để chuỗi tiền tố có độ dài t (từ 0 đến t) kết thúc bằng ký tự k. Dùng quy hoạch động để tính α.

α[t][k] = (α[t-1][k] + α[t-1][k-1]) . yt,k

Khi backwarding, định nghĩa hàm β[t][k] là tổng xác suất để chuỗi hậu tố độ dài T-t (từ t đến T) bắt đầu bằng ký tự k. Dùng quy hoạch động để tính β.

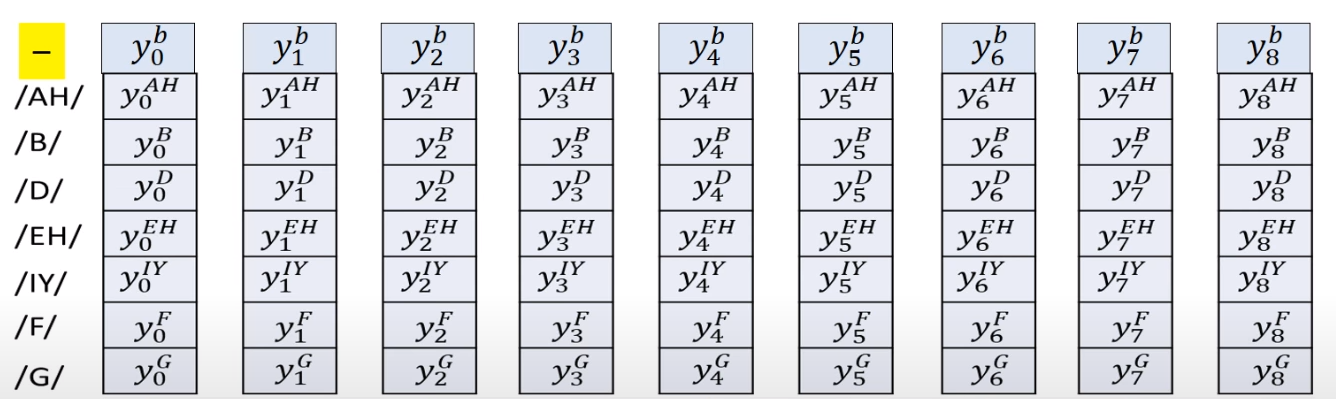
β[t][k] = ( β[t+1][k] + β[t+1][k+1]) . yt,k

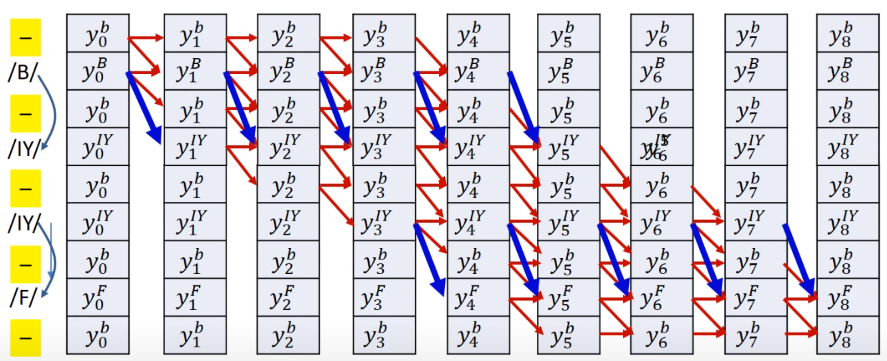
Như vậy xác suất để ký tự k suất hiện tại vị trí t trong một path hay một path đi qua ô t,k sẽ là:

Để tránh việc vanishing về 0 ta đưa về phương trình log likelyhood:

α[t][k] = log(

Vấn đề tiếp theo là làm thế nào để đưa một chuỗi lặp thành một chuỗi tốt hơn. Ta thêm vào ký tự CTC blank để phân cách các ký tự. Sau đó trong bảng lặp hàng ta thêm ký tự CTC blank vào giữa các ký tự.





Hình 20. Minh họa về CTC Blank

Chuỗi cuối dùng thu được bằng cách bỏ các ký tự lặp rồi sau đó bỏ kí tự CTC blank. Định nghĩa một many-to-one map B để thực hiện công việc trên: B(“hhelll-lo”) = B(“hhe-l-l-o”) = “hello”. Với mỗi chuỗi z sao cho B-1(y) = z, trong đó y là nhãn của ảnh đầu vào x, B-1 là hàm ngược của B, ta tính xác suất của chuỗi z như sau.

p(z|x) =

Đầu ra của bộ phân loại h(x) sẽ là label có xác suất cao nhất:

h(x) =B

Xác suất của nhãn y với x như sau:

q(y|x) =

Từ đó định nghĩa làm Loss (Negative Maximum Likelyhood):

OML(S,Nw) =

Gọi là maximum vì muốn cực đại hàm log từ đó cực hiểu hàm OML. Để

áp dụng ADAM, ta phải tính đạo hàm của OML theo yt,k

=

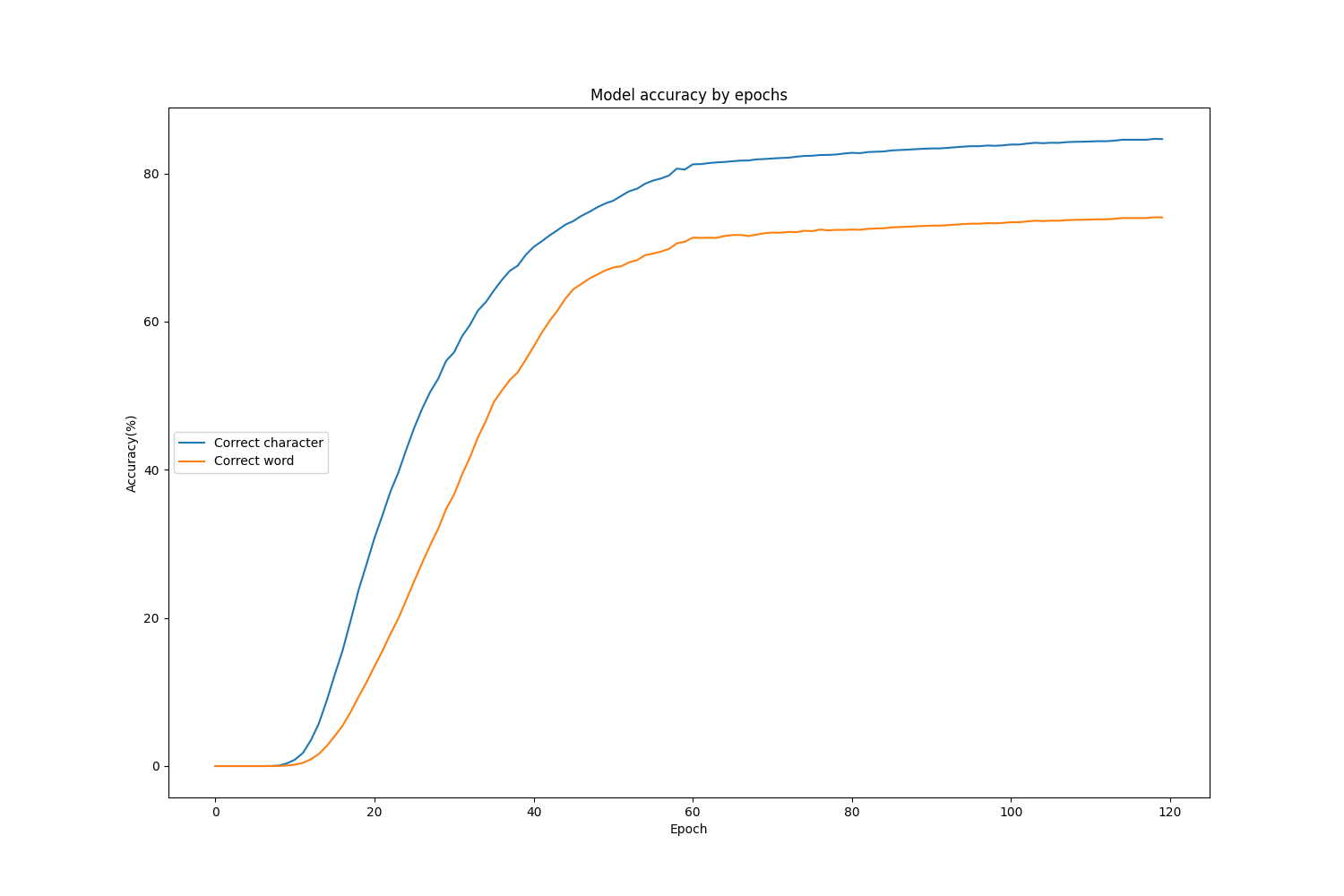
Vấn đề của mô hình CRNN sử dụng hàm mục tiêu CTC Loss là số lượng ký tự tối đa có thể dự đoán là cố định và khai báo từ đầu, do đó cần phải cẩn thận điều chỉnh mô hình để có thể dự đoán được số ký tự phù hợp với dataset. Hướng giải quyết là sử dụng mô *hình AttentionOCR* hoặc *TransformerOCR*, nhưng bài nghiên cứu sẽ không đề cập đến hai mô hình này.

# Thí nghiệm và kết quả

Công thức để đánh giá về độ chính xác của mô hình:

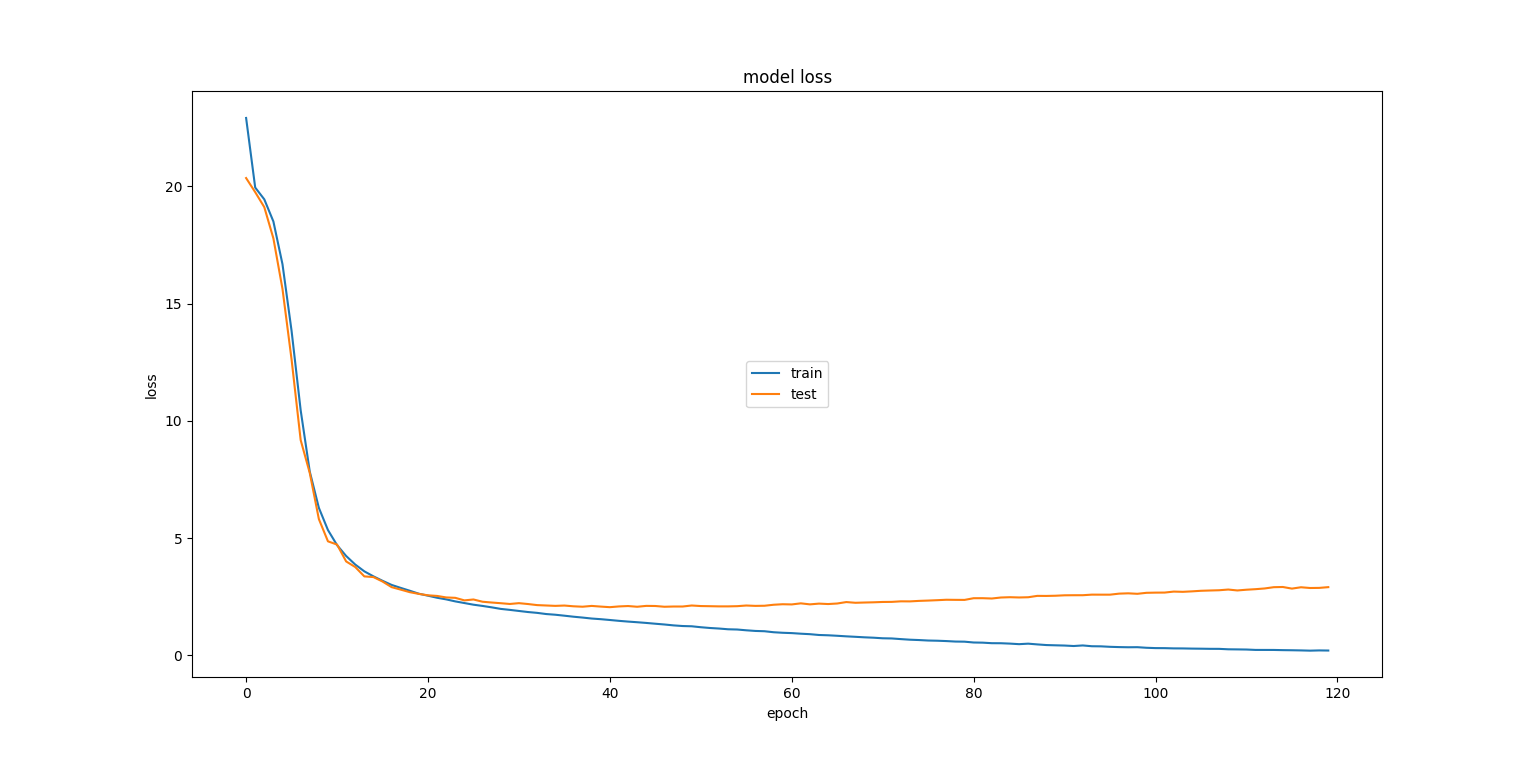
## Thí nghiệm với epoch

Một ký tự trong chuỗi predict được gọi là đúng nếu vị trí tương ứng của nó trong chuỗi ground truth cũng là ký tự đó. Một từ gọi là đúng nếu nó giống hoàn toàn từ ground truth. Với tập train gồm 30000 ảnh, tập valid lấy 3000 ảnh. Kết quả thu được lần lượt với 120 epoch như sau:



Hình 21. Sự phụ thuộc của độ chính xác theo số Epoch

Trong vài Epoch đầu độ chính xác bằng 0%, từ Epoch 10 trở đi độ chính xác tăng rất nhanh, cho đến Epoch 60 thì độ chính xác không còn tăng nhiều.

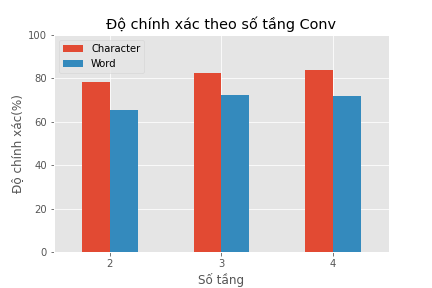


Hình 22. Sự phụ thuộc của hàm Loss theo số Epoch

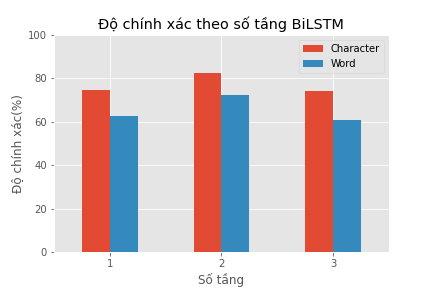
Đường màu cam là loss trên validation và màu xanh là loss trên tập train. Bắt đầu từ epoch 30 validation loss không còn biến đổi nhiều và đến epoch thứ 60 thì bão hòa. Trong các mô hình thử nghiệm tiếp theo sẽ chỉ lấy epoch bằng 60.

## Thí nghiệm với số lượng tầng ẩn.

Với mô hình 3 tầng CNN, 2 tầng RNN thì kết quả của độ chính xác về ký tự là 82,32% và chính xác về từ là 72,11%. Ta thử nghiệm các mô hình khác khi thay đổi số lượng tầng CNN và số lượng tầng RNN(BiLSTM).



Hình 23. Sự phụ thuộc của độ chính xác theo số tầng CNN



Hình 24. Sự phụ thuộc của độ chính xác theo số tầng RNN

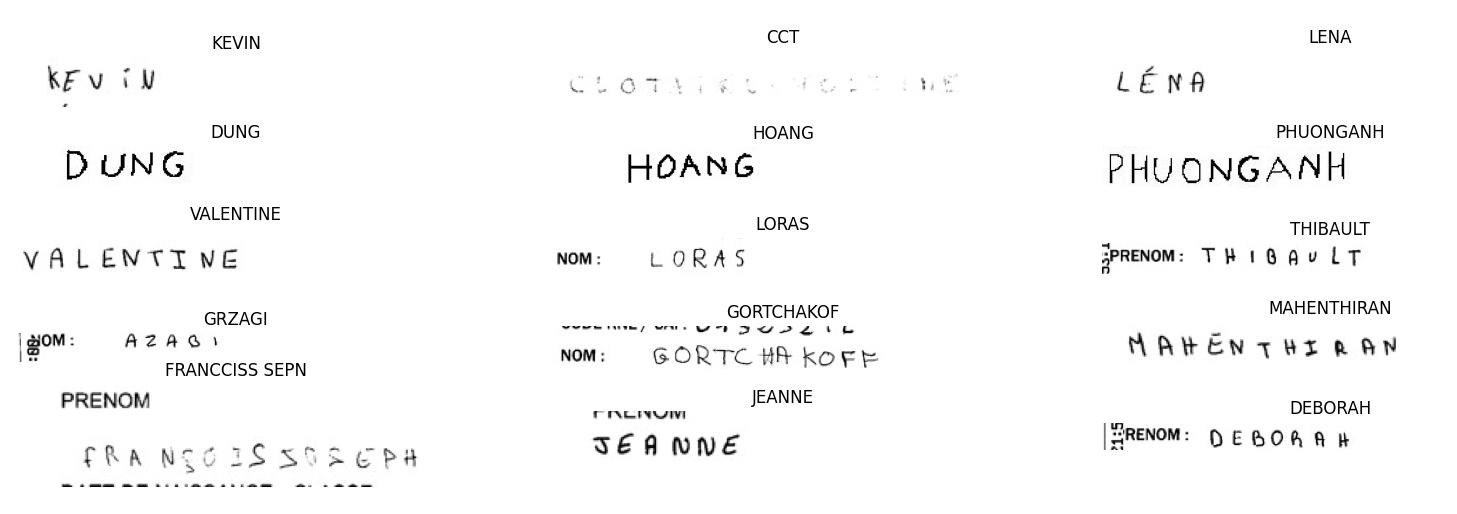
Như vậy 3 tầng CNN và 2 tầng BiLSTM cho được mô hình ổn định và hiệu quả về thời gian huấn luyện.

## Thí nghiệm với số lượng của tập dữ liệu

Mới chỉ có 10% dữ liệu của dataset được sử dụng. Ta sẽ thử đánh giá về độ chính xác theo kích thước của tập train.

Hình 25. Sự phụ thuộc của độ chính xác theo kích thước dữ liệu

Như vậy cách tốt nhất để tăng độ chính xác chính là tăng kích thước của tập train. Dưới đây là một số hình ảnh predict trực quan:



Hình 26. Kết quả trực quan

# Kết luận và hướng phát triển

Bài nghiên cứu này không đề xuất ra phương pháp mới cho tác vụ nhận dạng chữ viết tay (handwriting regconition) mà nghiên cứu về phương pháp đã được đề xuất[3]. Bài nghiên cứu chỉ ra các thành phần, công cụ và các thuật toán liên quan trong mô hình và đi từ dữ liệu đầu vào cho đến kết quả đầu ra. Ảnh được đưa vào tầng CNN để lấy ra các đặc trưng cục bộ sau đó đưa vào tầng RNN để mô hình hóa quan hệ giữa các phần tử trong vị trí của ảnh. Phần thí nghiệm đã chỉ ra tính vững chắc của mô hình với 3 tầng CNN và 2 tầng BiLSTM. Cách tốt nhất để tăng độ chính xác là tăng kích thước của tập học.

Hướng phát triển tiếp theo là sử dụng mô hình để trích xuất thông tin ghi bằng chữ viết tay trong các tờ đơn, tờ phiếu, văn bản hành chính. Ngoài ra mô hình còn có thể cải tiến để áp dụng với các ký tự tượng hình như Tiếng Trung, Tiếng Nhật hoặc nhận dạng các phép tính toán học.

# Danh mục tài liệu tham khảo

1. Chirag Patel, Atul Patel, PhD., Dharmendra Patel. [Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study (researchgate.net)](https://www.researchgate.net/profile/Chirag-Patel-12/publication/235956427_Optical_Character_Recognition_by_Open_source_OCR_Tool_Tesseract_A_Case_Study/links/00463516fa43a64739000000/Optical-Character-Recognition-by-Open-source-OCR-Tool-Tesseract-A-Case-Study.pdf), 2012.
2. Muhammad Tahir Qadri, Muhammad Asif. [Automatic Number Plate Recognition System for Vehicle Identification Using Optical Character Recognition,](http://www.emmersion.com.au/wp-content/uploads/2015/05/AUTOMATIC-NUMBER-PLATE-RECOGNITION-SYSTEM-FOR-VEHICLE1.pdf) 2009.
3. Jebastin Nadar, Handwriting Recognition using CRNN in Keras, 2020.
4. Jason Brownlee, [Gentle Introduction to the Adam Optimization Algorithm for Deep Learning](https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/) , machinelearningmastery.com - 2017
5. Alex Graves1, Santiago Fern´andez, Faustino Gomez, J¨urgen Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. *SNF grants 200021-111968/1 and 200020-107534/1.*
6. [*http://deeplearning.cs.cmu.edu/slides.spring19/lec14.CTC.pdf*](http://deeplearning.cs.cmu.edu/slides.spring19/lec14.CTC.pdf)
7. [*How do Convolutional Neural Networks work? (e2eml.school)*](https://e2eml.school/how_convolutional_neural_networks_work.html)
8. [*Homepage | End to End Machine Learning (teachable.com)*](https://end-to-end-machine-learning.teachable.com/)
9. [*MIT Deep Learning 6.S191 (introtodeeplearning.com)*](http://introtodeeplearning.com/)
10. [*How Deep Neural Networks Work | End to End Machine Learning (teachable.com)*](https://end-to-end-machine-learning.teachable.com/p/how-deep-neural-networks-work)
11. [*Các hàm kích hoạt (activation function) trong neural network | AI Curious*](https://aicurious.io/posts/2019-09-23-cac-ham-kich-hoat-activation-function-trong-neural-networks/#1-sigmoid)