Ghi chép bài giảng: Stanford CS231n (Spring 2025)

Lecture 7: Recurrent Neural Networks (RNNs)

Ho Hong Phuc Nguyen

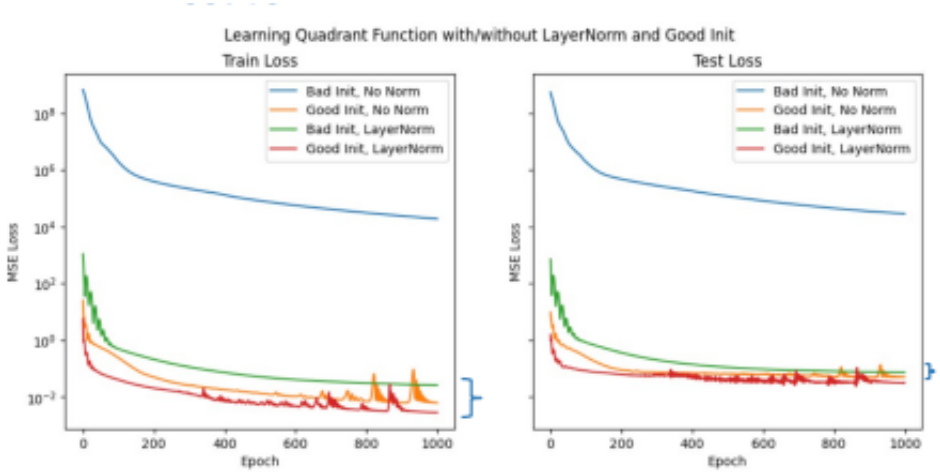
Ngày 18 tháng 2 năm 2026

Mục lục

1 Clarification: Normalization & Weight Initialization

Đầu bài giảng có đề cập đến câu hỏi: *"Liệu Normalization (như LayerNorm) có giải quyết được hoàn toàn vấn đề khởi tạo trọng số kém (bad initialization) không?"*

- **Câu trả lời:** Normalization giúp giảm thiểu vấn đề nhưng **không** giải quyết triệt để.
- **Chi tiết:**
  - Trong ví dụ về phân loại góc phần tư (quadrant classification), LayerNorm giúp mạng hội tụ ngay cả khi khởi tạo xấu.
  - Tuy nhiên, để đạt hiệu suất tối đa (best performance), ta vẫn cần khởi tạo trọng số tốt (Good Initialization).
- **Lưu ý quan trọng về bản chất của Normalization:**
  - **Với bài toán Quadrant Classification:** Mục tiêu chỉ là xác định góc phần tư (dựa trên dấu âm/dương hoặc vị trí tương đối), không cần biết chính xác tọa độ. Do đó, Normalization hoạt động tốt vì nó chuẩn hóa phân phối mà không làm mất đi tính chất phân tách của các góc.
  - **Với bài toán cần tọa độ chính xác (Exact Coordinates):** Normalization thực hiện việc trừ đi giá trị trung bình (centering) và chia cho độ lệch chuẩn (scaling/stretching). Hành động "kéo dãn"và "dịch chuyển"này làm mất thông tin về vị trí không gian tuyệt đối (absolute spatial location). Trong trường hợp này, Normalization sẽ làm **giảm hiệu suất** (hurt performance).



2 Experiment Tracking Tools (Công cụ quản lý thí nghiệm)

Trong quá trình huấn luyện ("babysitting the learning process") và tìm kiếm siêu tham số, việc theo dõi thủ công qua log text rất khó khăn. Giảng viên đề xuất sử dụng các công cụ visualization chuyên dụng.

2.1 Weights & Biases (W&B)

Đây là công cụ được giảng viên đánh giá cao và sử dụng trong hầu hết các dự án thực tế.

- **Chức năng chính:** Theo dõi và trực quan hóa song song nhiều lần chạy (runs) với các cấu hình siêu tham số khác nhau.
- **Parallel Coordinates Plot:** W&B cung cấp các biểu đồ trục song song (như hình minh họa trong bài giảng), giúp dễ dàng nhìn thấy mối tương quan.
- *Ví dụ:* Bạn có thể thấy ngay lập tức rằng các đường màu vàng (tương ứng với Dropout thấp) dẫn đến Accuracy cao hơn, trong khi các đường màu tím (Dropout cao) cho kết quả thấp hơn.
- **Lợi ích:** Giúp tiết kiệm thời gian khi so sánh hàng chục thí nghiệm để chọn ra bộ tham số tốt nhất (Best Hyperparameters).

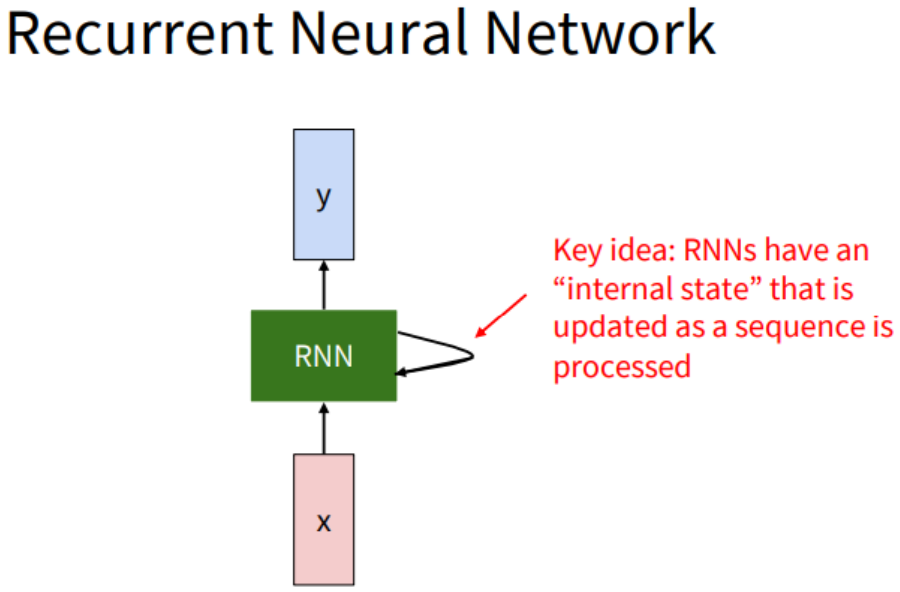
2.2 TensorBoard

Một công cụ tiêu chuẩn khác cũng được nhắc đến. Tuy giảng viên thích W&B hơn, nhưng TensorBoard vẫn là công cụ rất mạnh mẽ (thường đi kèm mặc định với TensorFlow/PyTorch) để theo dõi Loss, Accuracy và visualize đồ thị tính toán (Computational Graph).

3 Recurrent Neural Networks (RNN)

3.1 Định nghĩa

RNN là mạng nơ-ron được thiết kế để xử lý dữ liệu dạng chuỗi (sequence modeling) có độ dài thay đổi (variable length). Điểm đặc trưng của RNN là tính chất **hồi quy (recurrent)**: mạng duy trì một trạng thái ẩn (hidden state) được cập nhật qua từng bước thời gian.



3.2 Cơ chế hoạt động

Tại mỗi bước thời gian t, mạng nhận đầu vào x_t và trạng thái ẩn trước đó h_{t-1} để tính toán trạng thái ẩn mới h_t.

Công thức hồi quy (Recurrence Formula):

$$h_t = f_W(h_{t-1}, x_t) \tag{1}$$

Trong đó:

- h_t: Trạng thái ẩn tại thời điểm t (New state).
- h_{t-1}: Trạng thái ẩn tại thời điểm $t-1$ (Old state).

- x_t: Vector đầu vào tại thời điểm t.
- f_W: Hàm với tham số W. **Quan trọng:** Cùng một bộ trọng số W được sử dụng (share weights) cho mọi bước thời gian.

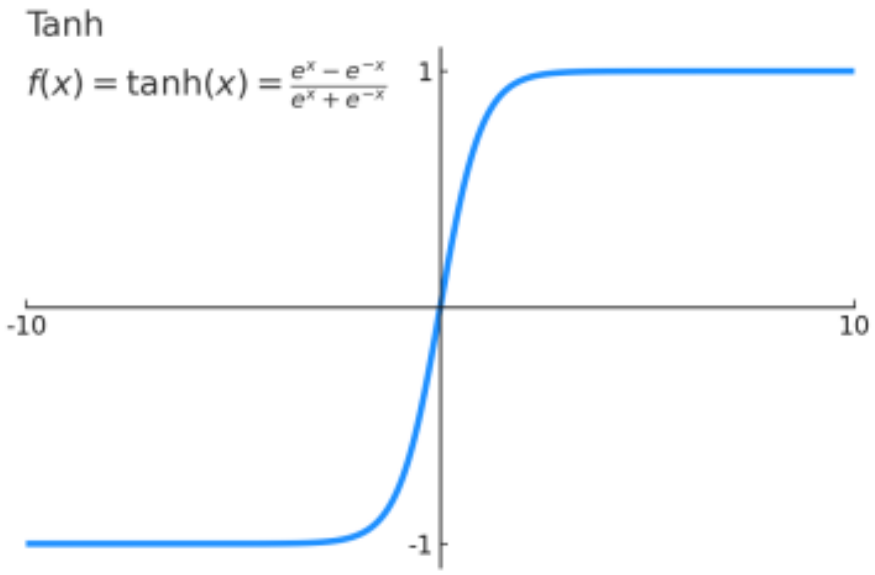
Vanilla RNN (Cụ thể):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{2}$$

$$y_t = W_{hy}h_t \tag{3}$$

Trong đó:

- W_{hh}: Trọng số chuyển đổi từ hidden state sang hidden state.
- W_{xh}: Trọng số chuyển đổi từ input sang hidden state.
- W_{hy}: Trọng số chuyển đổi từ hidden state sang output.
- tanh: Hàm kích hoạt (giá trị từ -1 đến 1), giúp kiểm soát độ lớn của trạng thái ẩn qua nhiều bước.



4 Sequence Modeling Scenarios (Các mô hình chuỗi)

Computational Graph của RNN thay đổi tùy thuộc vào bài toán:

4.1 One-to-Many

- **Input:** Kích thước cố định (ví dụ: 1 tấm ảnh).
- **Output:** Chuỗi có độ dài thay đổi (ví dụ: caption mô tả ảnh).
- **Ứng dụng:** Image Captioning.

4.2 Many-to-One

- **Input:** Chuỗi (ví dụ: video frames, chuỗi văn bản).
- **Output:** Kích thước cố định (ví dụ: nhãn phân loại).
- **Ứng dụng:** Video Classification, Sentiment Analysis.

4.3 Many-to-Many

Có hai dạng chính:

1. **Synced (Đồng bộ):** Input và Output có độ dài bằng nhau ($T_{in} = T_{out}$).
   - Ví dụ: Video classification per frame (phân loại từng frame trong video).
2. **Unsynced (Không đồng bộ):** Input và Output có độ dài khác nhau. Mạng đọc hết input rồi mới sinh output.
   - Ví dụ: Machine Translation (Dịch máy).

5 Training RNNs & Computational Graph

5.1 Backpropagation Through Time (BPTT)

Để huấn luyện RNN, ta lan truyền ngược gradient qua từng bước thời gian.

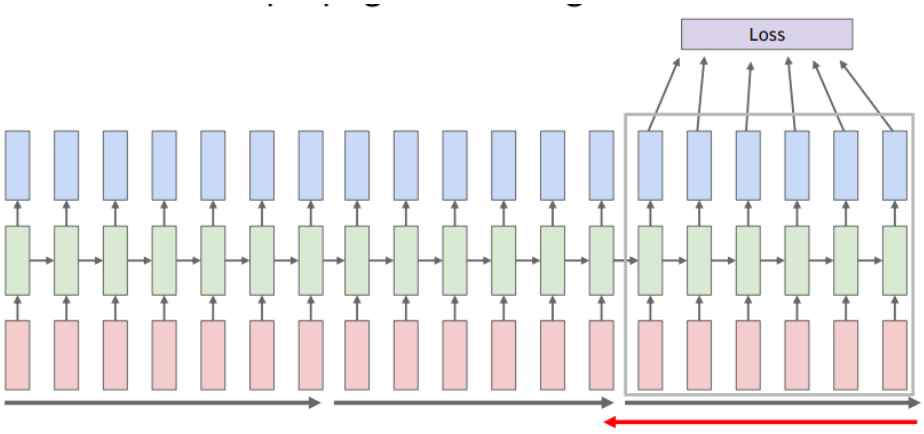
- Ta tính Loss tại từng bước (nếu có) và cộng tổng lại: $L = \sum L_t$.
- Gradient của W là tổng gradient tại mỗi bước thời gian (do W được dùng chung).

5.2 Problem: Memory Efficiency

Với chuỗi đầu vào quá dài (ví dụ: hàng nghìn token), việc lưu trữ toàn bộ computational graph để tính gradient sẽ gây tràn bộ nhớ (OOM) và tính toán rất chậm.

5.3 Solution: Truncated Backpropagation Through Time

Giải pháp thực tế là **Truncated BPTT**:



1. Chia chuỗi dài thành các đoạn nhỏ (chunks/batches) có kích thước cố định.
2. Thực hiện Forward pass và Backward pass chỉ trong phạm vi chunk đó.
3. **Quan trọng:** Trạng thái ẩn (h_t) cuối cùng của chunk trước được dùng làm đầu vào cho chunk sau, NHƯNG gradient **không** được truyền ngược về chunk trước (stop gradient/detach).

6 Interpretability & Visualizing RNNs

Một điểm thú vị là RNN có thể tự học các đặc trưng có thể giải thích được (interpretable) mà không cần gán nhãn cụ thể. Các nhà nghiên cứu (như Karpathy) đã visualize các tế bào (cells) trong hidden state và phát hiện:

- **Quote Detection Cell:** Tự động kích hoạt khi gặp dấu mở ngoặc kép và tắt khi gặp dấu đóng.
- **Line Length Cell:** Theo dõi độ dài dòng để dự đoán khi nào cần xuống dòng ($\backslash n$).

- **If-statement Cell:** Kích hoạt bên trong khối lệnh if.
- **Code Depth Cell:** Theo dõi mức độ thụt đầu dòng (indentation) trong code.

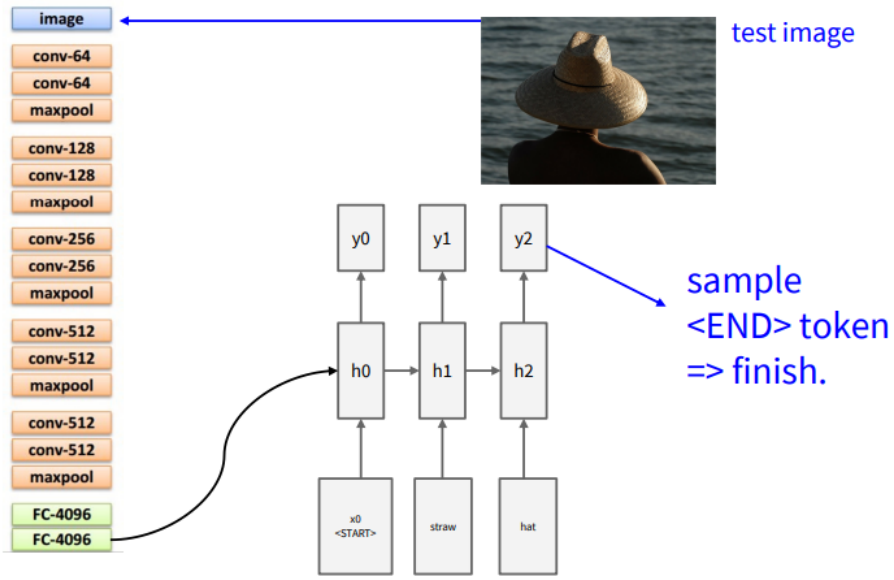
7 RNN Tradeoffs (Ưu Nhược điểm)

Ưu điểm (Pros)	Nhược điểm (Cons)
• Xử lý được đầu vào có độ dài bất kỳ (không bị giới hạn context window như Transformer). • Kích thước mô hình (số lượng tham số) không tăng theo độ dài đầu vào. • Về lý thuyết có thể nhớ thông tin từ rất xa trong quá khứ. • Tính đối xứng: áp dụng cùng một quy tắc cập nhật tại mọi bước.	• Tính toán tuần tự (Sequential computation): Chậm, khó song song hóa (không tận dụng tốt GPU như CNN/Transformer). • **Vanishing/Exploding Gradients:** Khó huấn luyện với chuỗi dài. • Trong thực tế, khả năng nhớ thông tin dài hạn rất kém (do giới hạn bottleneck của vector hidden state).

8 Applications (Computer Vision Focus)

8.1 Image Captioning

Mô hình kết hợp CNN + RNN:



- **Encoder:** Sử dụng CNN (thường bỏ lớp fully connected cuối, lấy features map hoặc vector đặc trưng) để mã hóa ảnh.
- **Decoder:** RNN nhận vector đặc trưng từ ảnh (thường khởi tạo h_0 hoặc đưa vào bước đầu tiên) và sinh ra chuỗi từ ngữ.
- **Vấn đề:** Mô hình thường dựa vào xác suất đồng xuất hiện (co-occurrence) trong dữ liệu train dẫn đến "hallucination"(ví dụ: thấy biển thì đoán có ván lướt sóng dù không có).

8.2 Visual Question Answering (VQA)

Input là Ảnh + Câu hỏi → Output là Câu trả lời.

- RNN dùng để mã hóa câu hỏi.
- CNN dùng để mã hóa ảnh.
- Kết hợp hai vector này để phân loại ra câu trả lời (Many-to-One).

9 Gradient Flow Vanilla RNN Issues

Khi thực hiện Backprop qua nhiều bước thời gian, gradient của loss L đối với h_0 bao gồm tích của rất nhiều ma trận Jacobians:

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(...) \cdot W_{hh} \tag{4}$$

9.1 Vanishing Gradients (Biến mất đạo hàm)

- **Nguyên nhân:** Hàm tanh có đạo hàm luôn < 1. Nếu singular value lớn nhất của $W_{hh} < 1$, tích của chuỗi dài các số nhỏ hơn 1 sẽ tiến về 0 cực nhanh.
- **Hậu quả:** Các bước ở xa (đầu chuỗi) không nhận được tín hiệu cập nhật, mạng không học được sự phụ thuộc dài hạn (long-term dependencies).
- **Giải pháp:** Thay đổi kiến trúc (LSTM, GRU).

9.2 Exploding Gradients (Bùng nổ đạo hàm)

- **Nguyên nhân:** Nếu singular value lớn nhất của $W_{hh} > 1$, gradient tăng theo cấp số nhân.
- **Giải pháp: Gradient Clipping** (Cắt ngọn gradient). Nếu norm của gradient vượt quá ngưỡng (threshold), ta scale nó lại.

10 Architecture Variants (Bổ sung)

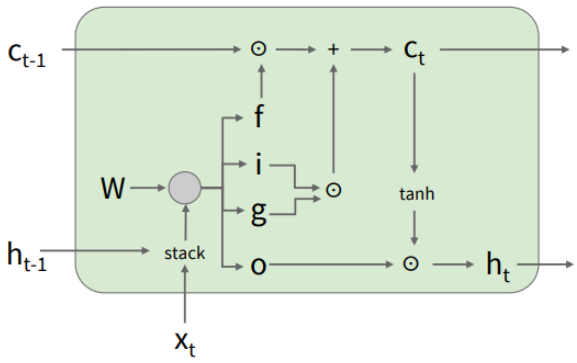
10.1 Multilayer RNNs

Tương tự như CNN, ta có thể chồng nhiều lớp RNN lên nhau để tăng khả năng biểu diễn.

- Output h_t của lớp 1 sẽ là Input x_t của lớp 2.
- Thường sâu khoảng 2-4 lớp (ít khi sâu như CNN).

10.2 Long Short-Term Memory (LSTM)

Được thiết kế năm 1997 để giải quyết Vanishing Gradients. LSTM tách biệt bộ nhớ tế bào (Cell State c_t) và trạng thái ẩn (Hidden State h_t).



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Cấu trúc gồm 4 cổng (gates) dùng hàm Sigmoid (σ) và Tanh:

1. **Forget Gate (f):** Quyết định quên bao nhiêu thông tin từ c_{t-1}.
2. **Input Gate (i):** Quyết định thêm bao nhiêu thông tin mới vào.
3. **Gate Gate (g):** Ứng cử viên thông tin mới (dùng tanh).
4. **Output Gate (o):** Quyết định xuất ra gì cho h_t.

Ý tưởng chủ đạo: LSTM tạo ra một "Gradient Superhighway"(thông qua Cell State) cho phép gradient chảy ngược về quá khứ mà không bị nhân liên tục với W hay đi qua hàm kích hoạt phi tuyến, giảm thiểu Vanishing Gradient. Tương tự ý tưởng Skip Connection trong ResNet.