Tài liệu

Recurrent Neural Network (RNN)

Hồ Nguyễn Phú

Ngày 13, tháng 10, năm 2025
Bình Dương cũ

Abstract: Mạng Neural Hồi Quy (RNN) là kiến trúc mạng neural học sâu được sử dụng cho xử lý dữ liệu tuần tự (Sequential Data). RNN có tác dụng trong các tác vụ xử lý ngôn ngữ tự nhiên (NLP), chuỗi thời gian, hình ảnh (e.g., image captioning), etc. Dù hiện nay đã xuất hiện các kiến trúc nổi bật khác như Transformers và Mamba, nắm vững RNN cho phép ta hiểu thêm về lĩnh vực học sâu, cũng như có thêm một công cụ mạnh mẽ và được nghiên cứu kỹ càng. Nội dung tìm hiểu bao gồm: (1) đặc trưng dữ liệu dạng chuỗi, (2) kiến trúc RNN, (3) kiến trúc LSTM, (4) kiến trúc GRU và (5) thực hành với Pytorch.

I. Đặc trưng dữ liệu dạng chuỗi:

Dữ liệu tuần tự bao gồm các quan sát (observations) được sắp xếp theo một thứ tự nhất định. Thứ tự này không nhất thiết phải là thời gian. Một số dạng dữ liệu tuần tự bao gồm:

- Chuỗi thời gian
- Văn bản
- DNA

1. Đặc điểm dữ liệu tuần tự:

Dữ liệu tuần tự có các đặc điểm sau: có thứ tự, phụ thuộc thời gian (temporal dependency), có độ dài thay đổi (variable length).

- ***Có thứ tự***: thay đổi thứ tự các phần tử có thể thay đổi ý nghĩa của cả dữ liệu.
- ***Phụ thuộc thời gian***: các phần tử phụ thuộc vào những phần tử đứng trước nó. Chia làm phụ thuộc ngắn hạn (short–range dependency) và phụ thuộc dài hạn (long–range dependency).
- ***Độ dài thay đổi***: khác với dữ liệu có cấu trúc (dữ liệu dạng bảng), các chuỗi có thể có độ dài bất kỳ. Cách xử lý: padding và masking.

2. Đặc điểm dữ liệu chuỗi thời gian:

Chuỗi thời gian bao gồm các đặc điểm của dữ liệu tuần tự. Ngoài ra còn có các tính chất sau:

- **Systematic pattern**: chuỗi thời gian thường có những pattern mang tính hệ thống mà ta có thể nhận thấy và mô hình hóa. Một số loại chính: trend, seasonality và cyclical patterns.
- **Tần suất**: tức khoảng thời gian (interval) giữa những lần quan sát liên tiếp.
- **Tính stationary / non–stationary**: các chỉ số thống kê (trung bình, trung vị, phương sai) của chuỗi thời gian có thể thay đổi (non–stationary) và gây khó khăn cho việc mô hình hóa.

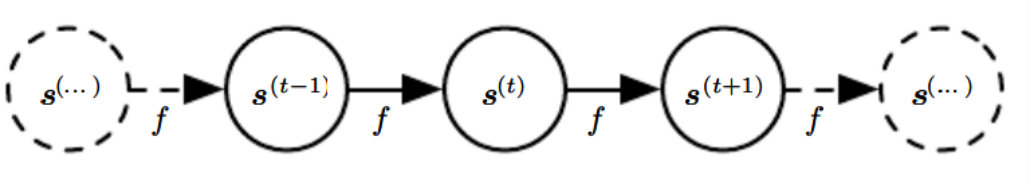
II. Kiến trúc RNN:

Từ đặc trưng trên của dữ liệu, bên cạnh Convolution 1D, ta có kiến trúc RNN cho phép xử lý dữ liệu dạng chuỗi hiệu quả.

1. Ý tưởng thuật toán:

Sách Deep Learning giới thiệu một hệ thống động (dynamic system) với hàm $s^{(t)} = f(s^{(t-1)}; \theta)$, trong đó trạng thái $s^{(t)}$ của step t sẽ được tính dựa trên trạng thái $s^{(t-1)}$ của step (t – 1) ngay trước nó, thông qua hàm $f(x; \theta)$ với tham số θ được sử dụng lại qua các bước thời gian.

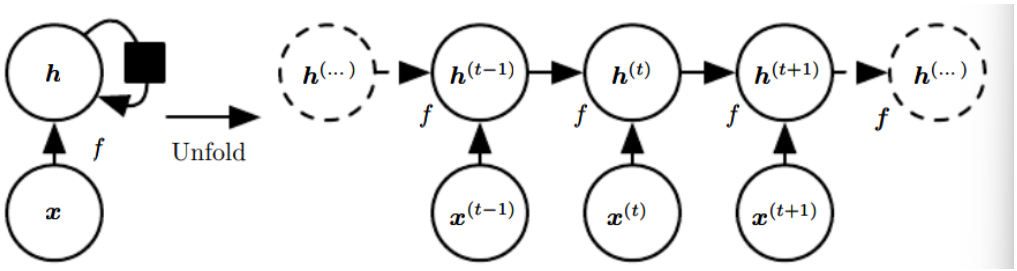
Nếu coi hệ thống động trên là một đồ thị tính toán (computational graph) thì nó sẽ trông như sau (sách Deep Learning):



Tiếp tục mở rộng hệ thống động trên, ta có hệ thống động chịu ảnh hưởng bởi tín hiệu bên ngoài $x^{(t)}$. Với hệ thống này, trạng thái $s^{(t)}$ không chỉ chứa thông tin của các bước trước đó, tức trạng thái $s^{(t-1)}$, mà còn xử lý thông tin đầu vào $x^{(t)}$ tại bước t. Công thức như sau:

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$$

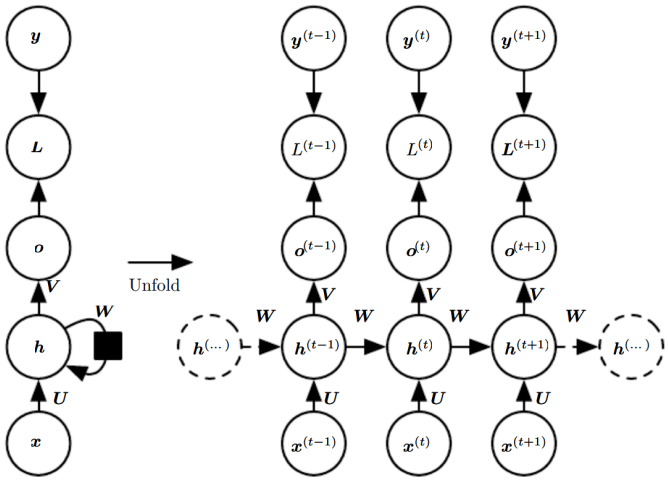
Đồ thị tính toán của hệ thống mới:



Mạng RNN sẽ lấy ý tưởng "graph unrolling" và parameter sharing của mục trên, và được thiết kế tương tự.

2. **Thuật toán:**

Trong mạng RNN, trạng thái ***s*** được biểu diễn dưới dạng một hidden unit (thường là vector ký hiệu ***h***). Đồ thị tính toán của mạng RNN như sau:



Trong đó, ma trận input–hidden U sẽ chuyển đầu vào $x^{(t)}$ thành một candidate. Cùng lúc, ma trận hidden–hidden W xử lý thông tin từ state $h^{(t-1)}$ trước đó, rồi cộng với candidate và đưa vào hàm kích hoạt (tanh) để tạo nên state mới $h^{(t)}$. Từ state $h^{(t)}$ tại bước t, ta tính toán đầu ra $o^{(t)}$ thông qua ma trận trọng số hidden–output V. Trong quá trình huấn luyện, ta sẽ có thêm hàm Loss và giá trị ground–truth trong pipeline để tính loss trong từng bước.

Hai kỹ thuật huấn luyện là Backpropagation Through Time (BPTT), và Teacher Forcing.

- BPTT thực hiện truyền đạo hàm tại Loss ngược về các bước thời gian trước đó thông qua quy tắc chuỗi.
- Ngoài ra, kỹ thuật Teacher Forcing sẽ thực hiện feed trực tiếp ground truth làm đầu vào tại mỗi bước. Teacher Forcing hiệu quả khi ta muốn song song hóa quá trình huấn luyện (do không phụ thuộc vào bước trước đó), hoặc không có ma trận hidden–hidden.

Qua hai kỹ thuật trên, đặc biệt là BPTT, ta thấy một vấn đề quan trọng của RNN là gradient vanishing / exploding. Nguyên nhân là do đạo hàm riêng của hàm loss tại các bước thời gian sẽ tăng theo cấp số mũ (của ma trận hidden–hidden W) khi ta càng lùi về quá khứ. Hệ quả là với giá trị eigen bé hơn 1, đạo hàm riêng của loss sẽ hội tụ về 0 (gradient vanishing); còn với giá trị eigen lớn hơn 1, đạo hàm riêng của loss sẽ phân kỳ (gradient exploding).

3. **Công thức:**

Kí hiệu của sách Deep Learning hơi khó, do đó ta sẽ sử dụng kí hiệu của paper [Recurrent Neural Networks (RNNs): A gentle Introduction and Overview]. Đặt:

W_{xh} : ma trận trọng số input–hidden

W_{hh} : ma trận trọng số hidden–hidden

W_{ho} : ma trận trọng số hidden–output

h_t : vector trạng thái / hidden tại bước thời gian t

x_t : vector đầu vào tại bước thời gian t

a) Tính trạng thái tại bước thời gian t:

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

Trong đó:

$W_{hh}h_{t-1}$: vector thông tin từ các bước thời gian trước đó (dimension ***h***)

$W_{xh}x_t$: vector thông tin input (dimension từ ***d*** → ***h***)

b : bias

Với bước thời gian đầu tiên, trạng thái $h_{t-1} = 0$.

b) Tính đầu ra tại bước thời gian t:

$$o_t = W_{ho}h_t + c$$

Trong đó:

$W_{ho}h_t$: vector chứa thông tin đầu ra (dimension phụ thuộc vào đầu ra)

c : bias

Bên cạnh đó, ta có thể có hàm softmax, sigmoid để đưa ra đầu ra phù hợp.

III. Kiến trúc LSTM:

1. **Bối cảnh**: vấn đề vanishing gradient và cảm hứng từ long / short term memory của con người.

2. **Ý tưởng:**

Nhìn chung là một gated recurrent unit. Trong đó, ta tạo ra những đường đi cho gradient mà không bị vanishing hay exploding. Ở LSTM, đó là những gate I, O và F giúp xử lý thông tin đầu vào và ra, cũng như quyết định "quên" đi bao nhiêu thông tin cũ.

Theo đó, LSTM về cơ bản là một mạng RNN nhưng có thêm cơ chế lưu thông tin dựa vào cấu trúc gọi là cell. Cell được đọc và bổ sung thông tin thông qua các gate: input gate I_t để đọc thông tin đầu vào, O_t để đọc thông tin trên cell (và tính trạng thái hidden), và F_t để reset thông tin trước đó của cell.

3. **Thuật toán và công thức**: (xem thêm **Backpropagation cho LSTM** và Constant Error Carousel)

Ký hiệu dùng trong tính toán như sau:

i_t : vector đầu vào (input gate)

o_t : vector đầu ra (output gate)

f_t : vector "quên" (forget gate)

$\overline{c_t}$: vector candidate tính từ đầu vào

c_t : vector candidate dùng để tính trạng thái hidden

a) Tính toán các gate I, O và F:

Gate I: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$

Gate O: $i_o = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$

Gate F: $i_f = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$

b) Tính toán thông tin tại cell:

Cell là đơn vị lưu thông tin của mô hình LSTM. Dựa trên các gate I, O và F ta sẽ cập nhật thông tin cell đó.

Trước hết, khi có input, ta sẽ tính toán candidate memory cell $\overline{C}_t$. **Candidate memory cell** này có thể coi như là **thông tin mới** dựa trên đầu vào tại bước hiện tại (x_t) và trạng thái tại bước trước đó (h_{t-1}). Ta sử dụng hàm $tanh(x)$ để giới thiệu tính phi tuyến vào mô hình. Công thức:

$$\overline{c}_t = tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c)$$

Ở bước kế tiếp, ta dùng $\overline{c}_t$ để cập nhật nội dung memory trước đó c_{t-1}. Chi tiết hơn, forget gate F sẽ quyết định bao nhiêu phần của c_{t-1} được giữ lại; còn input gate I sẽ quyết định bao nhiêu phần của $\overline{c}_t$ được dùng để cập nhật trạng thái. Các vector f và i có cùng kích thước với các vector cell c sẽ điều chỉnh cell thông qua phép nhân element–wise $\odot$ (tích Hadamard). Công thức:

$$c_t = f_t \odot c_{t-1} + i_t \odot \overline{c}_t$$

c) Tính toán trạng thái hidden từ giá trị cell c_t:

Như vậy, ta đã tính được giá trị cell c_t. Để chuyển đổi từ cell thành trạng thái hidden, ta sử dụng output gate O kết hợp với hàm $tanh(c_t)$ để tạo tính phi tuyến. Công thức:

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

IV. Kiến trúc GRU:

1. **Bối cảnh**:

- Vấn đề vanishing / exploding gradient của RNN.
- Các mô hình biến thể của RNN hoạt động mạnh mẽ trên tác vụ Machine Translation.
- Mạng LSTM có độ phức tạp tính toán cao.

2. **Ý tưởng**:

Có các gating unit điều chỉnh dòng chảy thông tin như LSTM, nhưng không có một cell riêng biệt. Ở đây, thay vì có cả ba gate gốc của LSTM, GRU chỉ có update gate (tạm gọi Z) và reset gate (tạm gọi R). Thông tin trước đây phải xử lý trên cell sẽ được xử lý trực tiếp trên hidden state.

3. **Thuật toán**:

a) Tính các gate:

Đầu tiên, ta tính các update gate và reset gate. Update gate thay thế cho input và forget gate của LSTM, còn reset gate được dùng để quyết định bao nhiêu trạng thái cũ được xét đến khi tạo candidate input cho bước thời gian mới (ở LSTM, bước này không tồn tại). Output gate của LSTM không cần dùng đến do ta không còn lưu thông tin trên cell state mà xử lý trực tiếp trên hidden state.

Công thức tính update gate	Công thức tính forget gate
$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1})$	$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1})$

Nhìn chung, điểm khác biệt chính trong công thức giữa hai gate chỉ là ma trận trọng số để xử lý đầu vào x_t và trạng thái hidden h_{t-1}.

b) Tính hidden state hiện tại:

Sau khi có forget gate r_t, candidate activation $\overline{h}_t$ được tính dựa trên đầu vào x_t và trạng thái h_{t-1}. Như đã nói bên trên, ở bước này ta sẽ quyết định bao nhiêu thông tin từ trạng thái cũ h_{t-1} sẽ được lưu trữ trong candidate của input – thông qua forget gate r_t. Công thức tính:

$$\overline{h}_t = tanh[x_t + W_{hh}(r_t \odot h_{t-1})]$$

Từ candidate input $\overline{h}_t$ và trạng thái hidden cũ h_{t-1}, ta tính trạng thái hidden mới. Công thức tính như sau:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \overline{h}_t$$

Ở đây, update gate nằm trong khoảng $[0; 1]$ nên công thức tính trạng thái hidden h_t là một phép nội suy tuyến tính. Tức, giá trị h_t sẽ nằm trên đường thẳng nối liền hai tọa độ của vector h_{t-1} và vector $\overline{h}_t$.

V. Thảo luận bên lề:

1. **Tại sao lại dùng phép cộng trong lúc cập nhật state đối với LSTM và GRU?** (nguồn)

- (1) Phép cộng giúp lưu trữ thông tin lâu hơn: các đặc trưng quan trọng không bị ghi đè (như RNN gốc) mà được giữ nguyên.
- (2) Đạo hàm ngược dễ hơn: phép cộng tạo ra các shortcut paths, giảm vanishing gradient và tránh đạo hàm tăng theo cấp số mũ của ma trận trọng số khi lùi về các bước thời gian xa hơn.

2. LSTM khác gì so với RNN?

- **Không trực tiếp lưu hidden state**: Thay vào đó, LSTM tính toán giá trị cell và tính hidden state thông qua output gate.
- **Xử lý thông tin qua các gate**: input gate, output gate, forget gate.
- **"Quên" thông tin**: Có cơ chế quên thông tin trước đó thông qua input gate và forget gate.
- **Đạo hàm truyền về dễ hơn thông qua**:
  + Constant Error Carousel (CES): các gate sẽ học cách điều chỉnh lượng error truyền về, giảm thiểu gradient vanishing và exploding (**paper gốc trang 1745–1746** và **trang Github**); với forget gate bằng 1, đạo hàm truyền về theo hàm identity (**thảo luận**).
  + Phép cộng các giá trị: tránh đạo hàm tăng tương ứng với cấp số mũ của ma trận hidden–hidden.
- **Độ phức tạp tính toán**: cao hơn RNN.

3. GRU khác gì so với LSTM?

- **Không lưu giá trị cell riêng biệt**: GRU xử lý trực tiếp trên trạng thái hidden.
- **Các gate khác biệt**: bao gồm *update gate* và *reset gate*.
    + *Input gate* và *forget gate* của LSTM được thay thế bởi *update gate*.
    + *Output gate* không cần sử dụng do xử lý trực tiếp trên trạng thái hidden.
    + *Reset gate*: dùng để scale đối với trạng thái hidden khi tính toán candidate của input (LSTM không có bước này).
- **Chỉ dùng update gate khi cập nhật trạng thái hidden mới**:
    + Dùng nội suy tuyến tính giữa hai giá trị trạng thái hidden cũ và candidate input mới.
    + *Update gate* thay thế *input gate* và *forget gate*.
- **Độ phức tạp tính toán**: thấp hơn LSTM.

VI. Thực hành với Pytorch:

Xem file đính kèm.

VII. Tài liệu tham khảo:

GeeksForGeeks: Sequential Data Analysis in Python (code, khá sâu về time–series)

ApX: Characteristics of Sequence Data (khóa học đi sâu vào Sequence Data)

ApX: Characteristics of Time Series Data (khóa học đi sâu vào Time Series Data)

Recurrent Neural Networks (RNNs): A gentle Introduction and Overview (giới thiệu RNN, BPTT, Teacher Forcing và LSTM)

Long Short-Term Memory (Paper gốc LSTM)

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling (công thức GRU)

Cuộc thảo luận trên Data Science Stack Exchange (Constant Error Carousel)