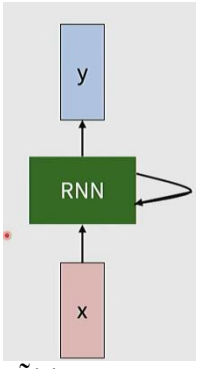
## 1. Recurrent Neural Network




Ý tưởng cốt lõi của RNN là có 1 chuỗi input x và có đầu ra là 1 chuỗi output y, mỗi input phụ thuộc vào quá khứ (gần giống HMM) -> RNN là mạng có hidden state giúp mô hình hoá phụ thuộc theo thời gian bằng cách truyền thông tin từ bước trước sang bước sau

## 2. RNN hidden state update

$$h_t = f_W(h_{t-1}, x_t)$$
$$y_t = f_{W_{hy}}(h_t)$$

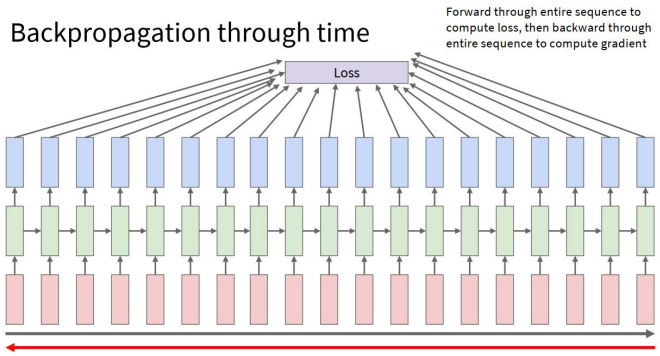
$h_t$ là state mới, $h_{t-1}$ là state cũ

Điểm cốt lõi: cùng một bộ tham số W dùng lại cho mọi timestep (weight sharing)

3. Backpropagation

### 3.1. Backpropagation Through Time

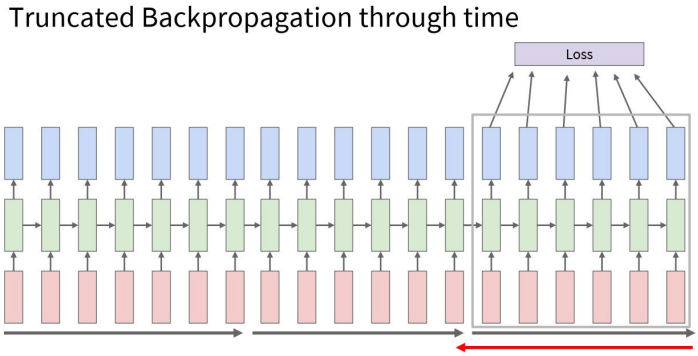
Ý tưởng: Forward qua toàn bộ chuỗi để tính loss từng cái xong rồi cộng tổng nó lại




Điều này đồng nghĩa là sẽ có 1 bộ Weight sẽ được xài xuyên suốt không cập nhật vì khi chạy xong mới cập nhật -> có thể sẽ train nhiều epoch

### 3.2. Truncated Backpropagation Through Time

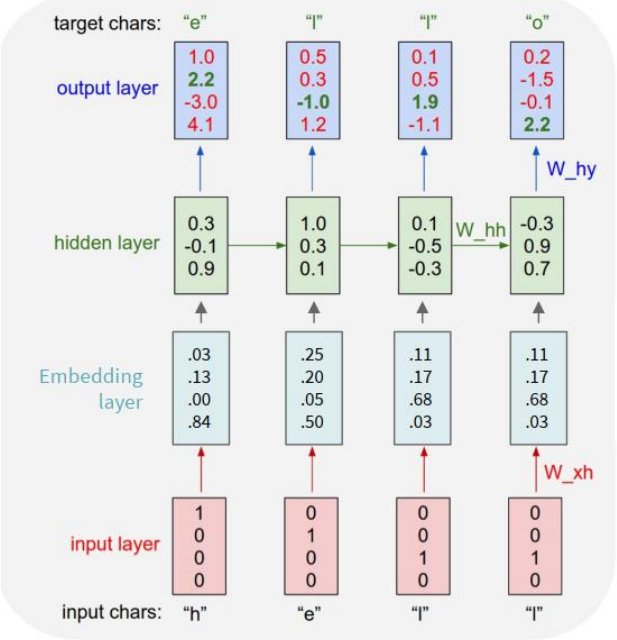
Ý tưởng: fix time window, tức là chọn số lượng các state rồi train và cập nhật weight xong tiếp tới time window tiếp theo -> cập nhật được liên tục hơn




## 4. Ví dụ thực tế: Character-level Language Model + Embedding

Vocabulary ví dụ: [h, e, l, o], training sequence “hello”

Sampling test-time: Mỗi bước softmax ra phân phối ký tự, sample 1 ký tự, feed lại làm input bước sau

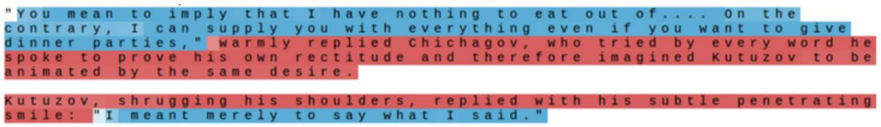



Embedding layer = bảng tra (lookup table) biến mỗi token/ký tự (ID rời rạc) -> vector dense D chiều để NN/RNN xử lý được -> giảm chiều dữ liệu xuống D thay vì phải V(số vocab)

## 5. Searching for interpretable cells

Khi train RNN/LSTM trên chuỗi dài (nhất là code), một số cell sẽ học ra biến trạng thái dễ hiểu (in_string, in_comment, depth, line_pos, …) -> interpretable cells vì activation của chúng tương ứng rõ ràng với một khái niệm

Ví dụ có tìm câu nói



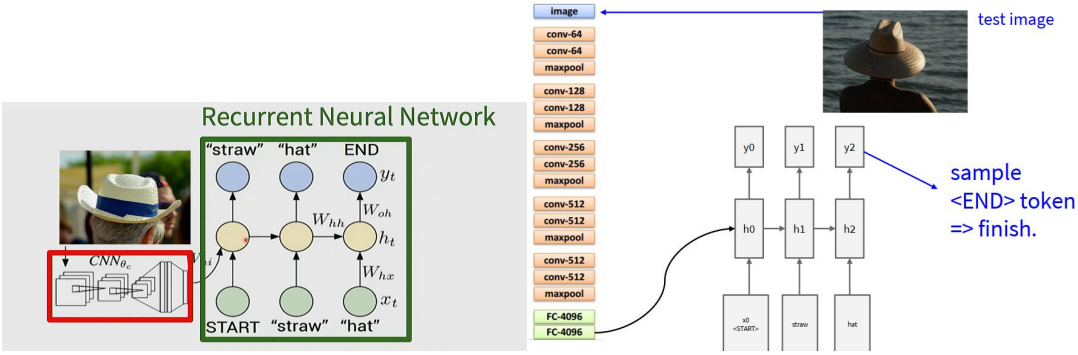

quote detection cell

## 6. RNN tradeoffs

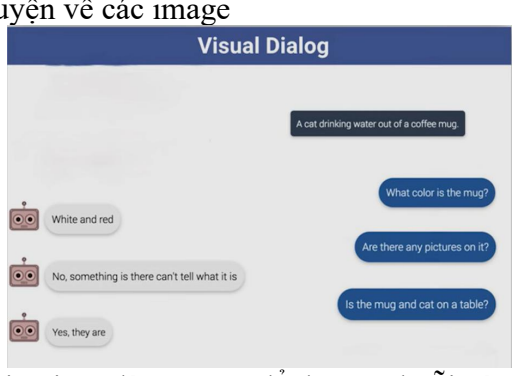
- Ưu điểm
  + RNN xử lý được chuỗi có độ dài bất kỳ
  + RNN có thể dùng thông tin rất xa trong quá khứ
  + Tham số của RNN không phụ thuộc độ dài chuỗi
  + Weight sharing theo thời gian
- Nhược điểm
  + RNN chậm vì phụ thuộc tuần tự giữa các timestep
  + Thực tế RNN khó học long-range dependency vì nếu đi xa thì các gradient trước đó rất dễ bị vanishing

### 6.1. Một vài application

- Image captioning: thường sẽ kết hợp với CNN (không dùng last layer vì last layer để classify) để tạo ra mảng matrix W trước rồi mới dùng RNN



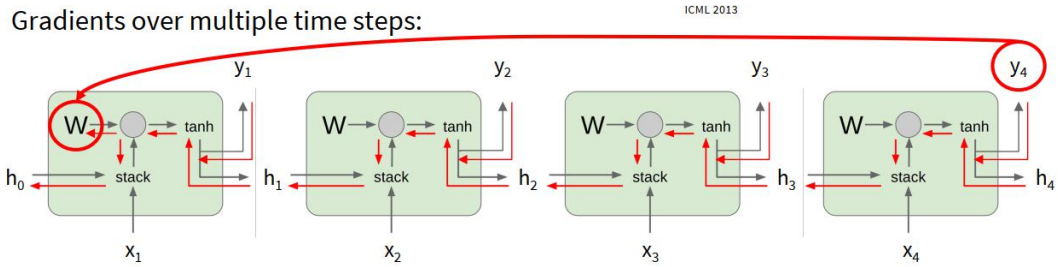

- Visual question answering (VOA)
- Visual dialog: nói chuyện về các image




- Visual Language Navigation: dùng RNN để đưa ra chuỗi các bước đi với input là cảnh thay đổi với mỗi move

## 7. Vannila RNN Gradient Flow

Gradients over multiple time steps:




$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_T}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_t}\left(\prod_{t=2}^{T} \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Mà với hàm tanh đạo hàm thì hầu như sẽ bé hơn 1 tức là nếu T càng lớn thì tích các số bé hơn 1 càng nhỏ -> vanishing gradient

Còn nếu không áp dụng non-linear:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_T}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_t} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

W lớn hơn 1 thì exploding còn bé hơn 1 thì vanishing
Đối với exploding thì đơn giản ta scale gradient với norm
Còn với vanishing thì đổi kiến trúc (LSTM)