Recurrent Neural Networks

Tirtharaj Dash

Dept. of CS & IS and APPCAIR
BITS Pilani, Goa Campus

October 28, 2021

Where we are

Summary of the previous lecture:

- Data with temporal-dependency
- Learning without *latent* variables
- Role of *summary*
- Modelling joint-distribution
- The idea of recurrence

We now have fair idea of what is recurrence in modelling sequences. We will do the following:

- RNNs with recurrence
- Different varieties of sequence learning problems
- Forward propagation in a RNN
- Backpropagation in a RNN (BPTT)*

*If time permits

Limitations of MLP or CNN:

- They accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes). [Too constrained]
- These models perform this mapping using a fixed amount of computational steps (e.g. the number of layers in the model). [Expressivity]
- Not effective data with temporal dependencies (e.g. time-series, sentences) [Problem-specific limitation]

*** We will need network structures that allow us to operate over sequences of vectors.

State and Recurrence I

- We call a hidden representation of a network as a 'state', denoted as $\mathbf{h}$.
- We will be attaching a time information (t) to this:

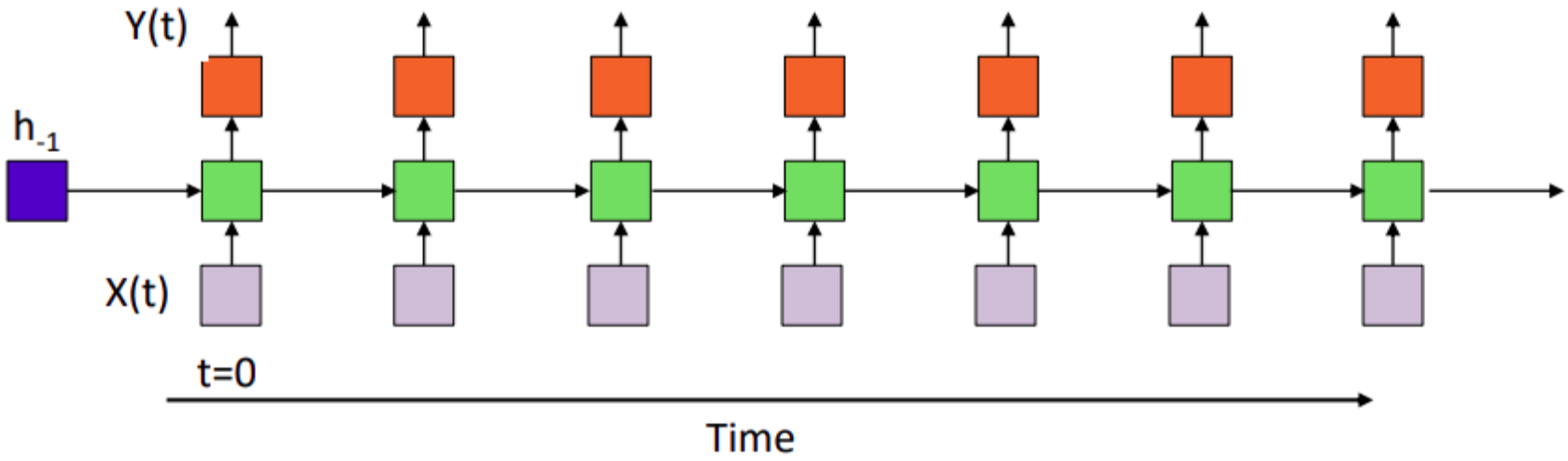
$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

and,

$$\mathbf{y}_t = g(\mathbf{h}_t)$$

- An initial state has to be defined: $\mathbf{h}_0$ (or, sometimes, people use $\mathbf{h}_{-1}$)
- State summarises information about the entire past.
- The neural network architecture that implements the above is a fully recurrent neural network.

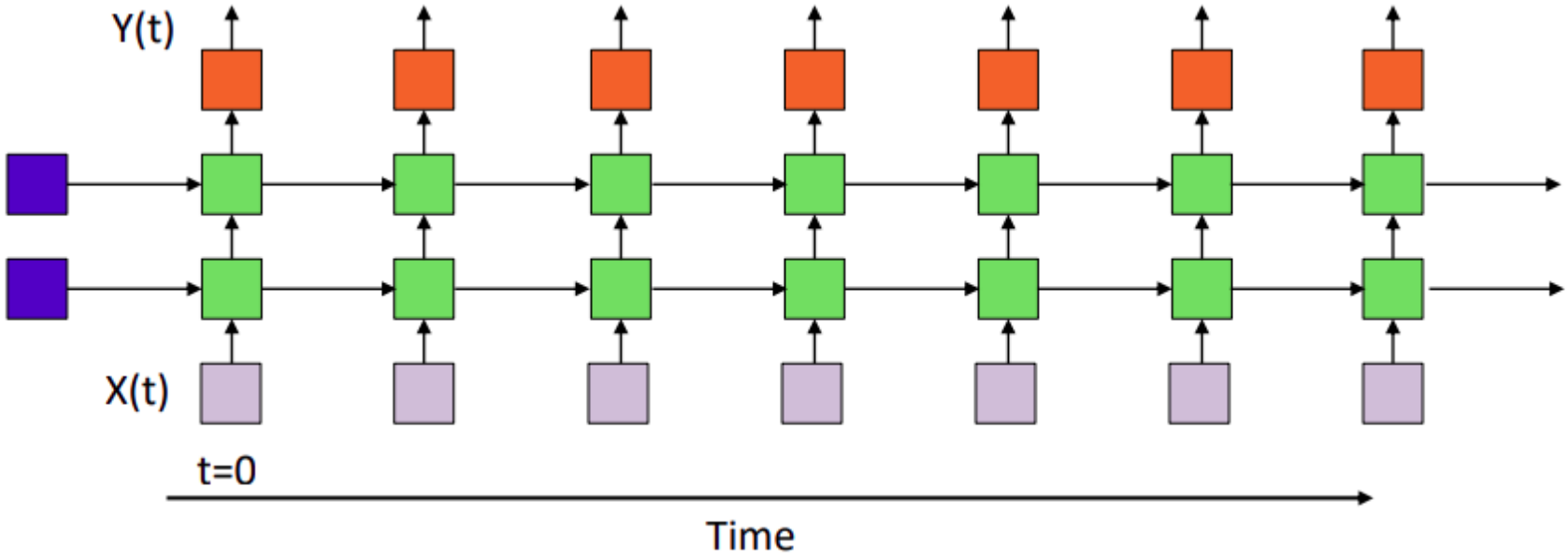
State and Recurrence II



- A state at any time is determined by the input at that time, and the previous state.
- This is a typical recurrent neural network (RNN), with one hidden layer.
- Also note that this is a generic case: where, input and output lengths are same. In other cases, we just remove some of the inputs or outputs or both.

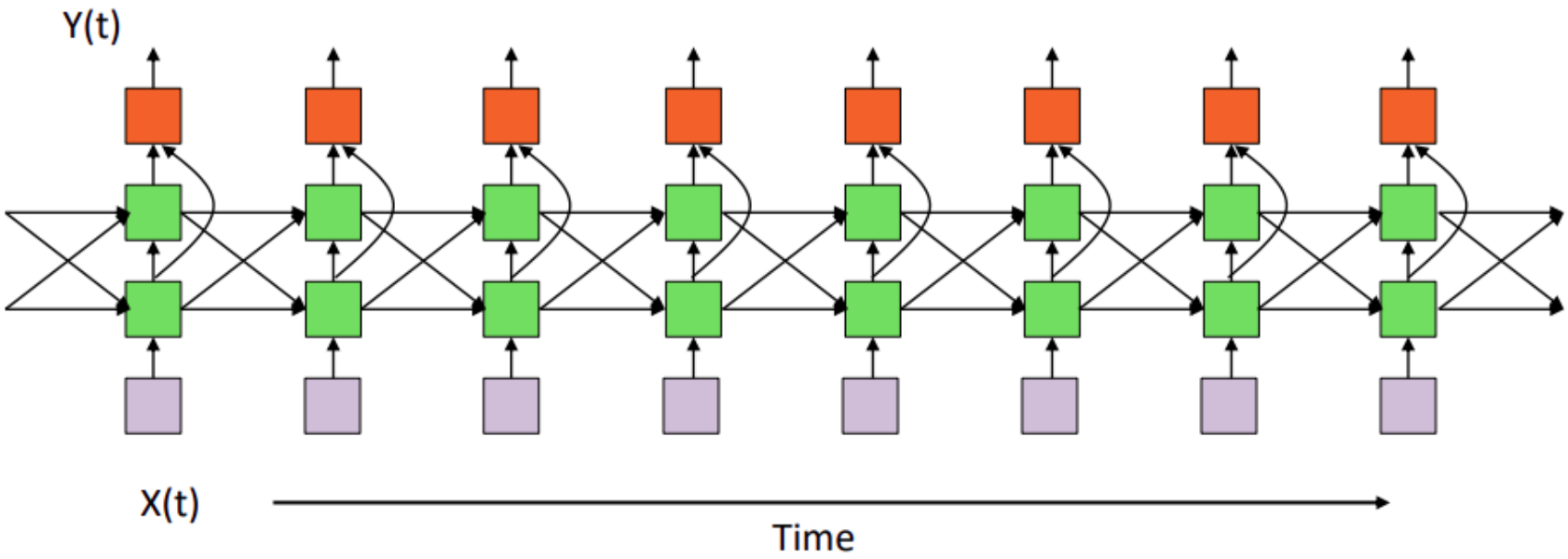
State and Recurrence III

- The state can be arbitrarily complex. Like this:

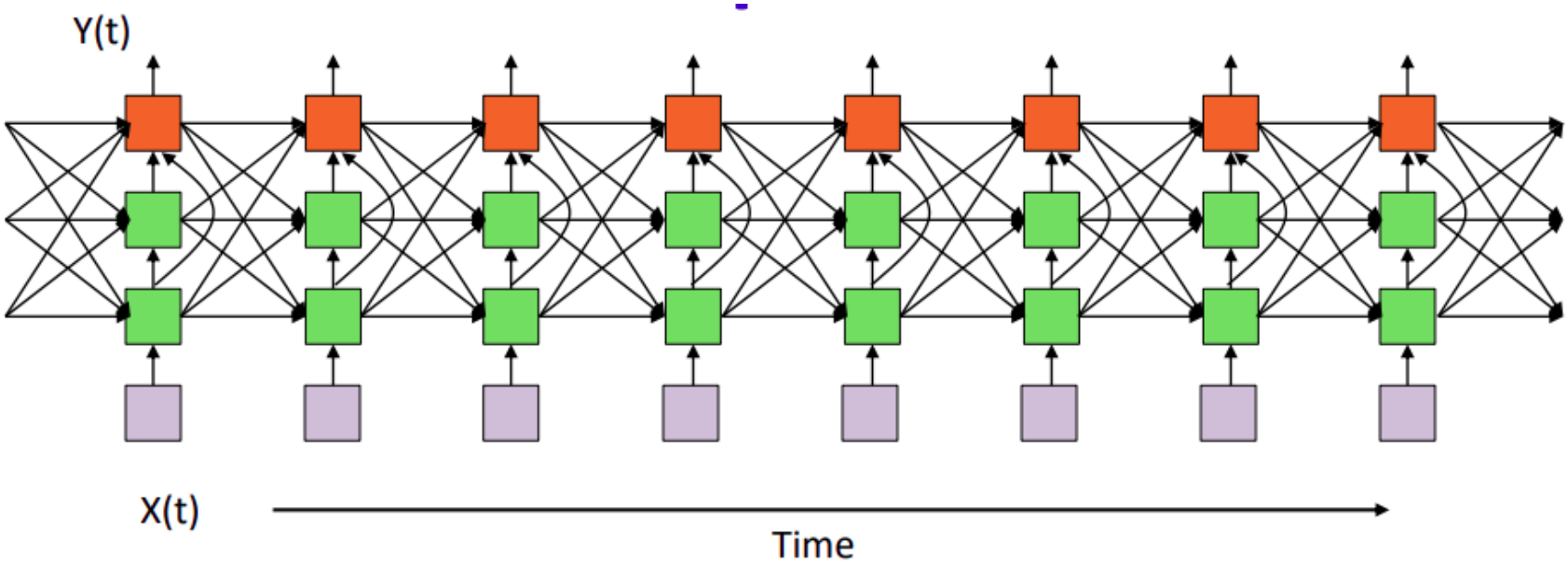


State and Recurrence IV

or, these:

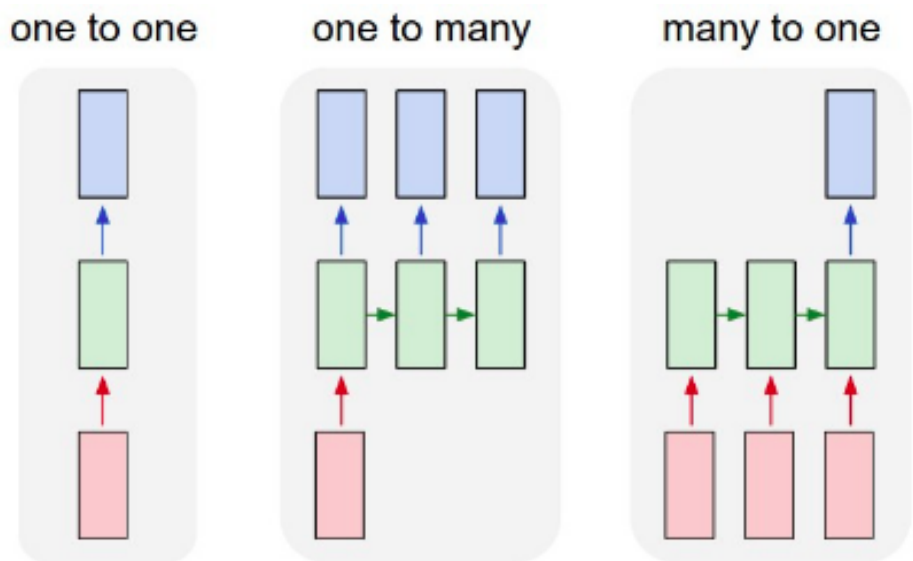


State and Recurrence V



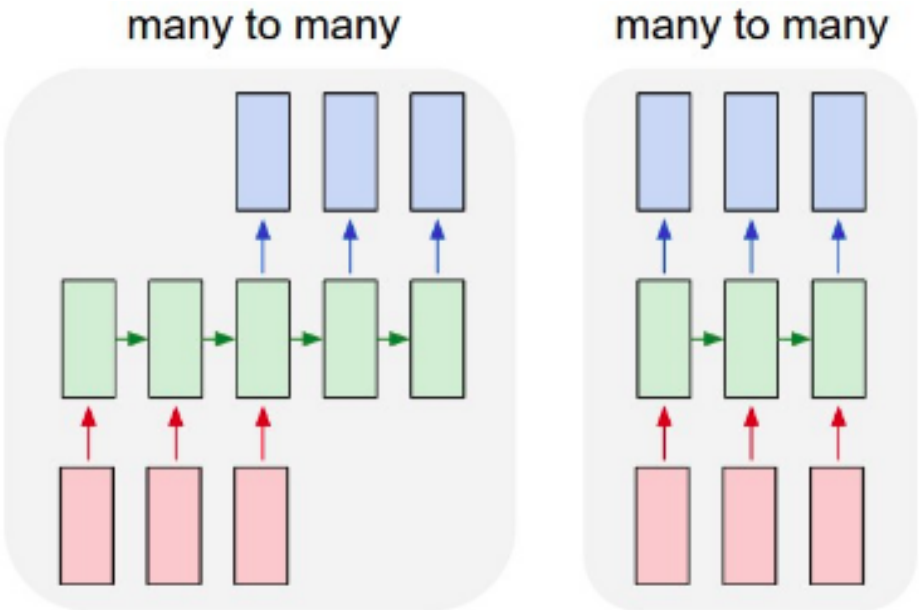
Variants of architectures I

- (1) One-to-one: Vanilla network; from fixed-sized input to fixed-sized output (e.g. image classification).
  (2) One-to-many: Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
  (3) Many-to-one: Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).



Variants of architectures II

- (4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
  (5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

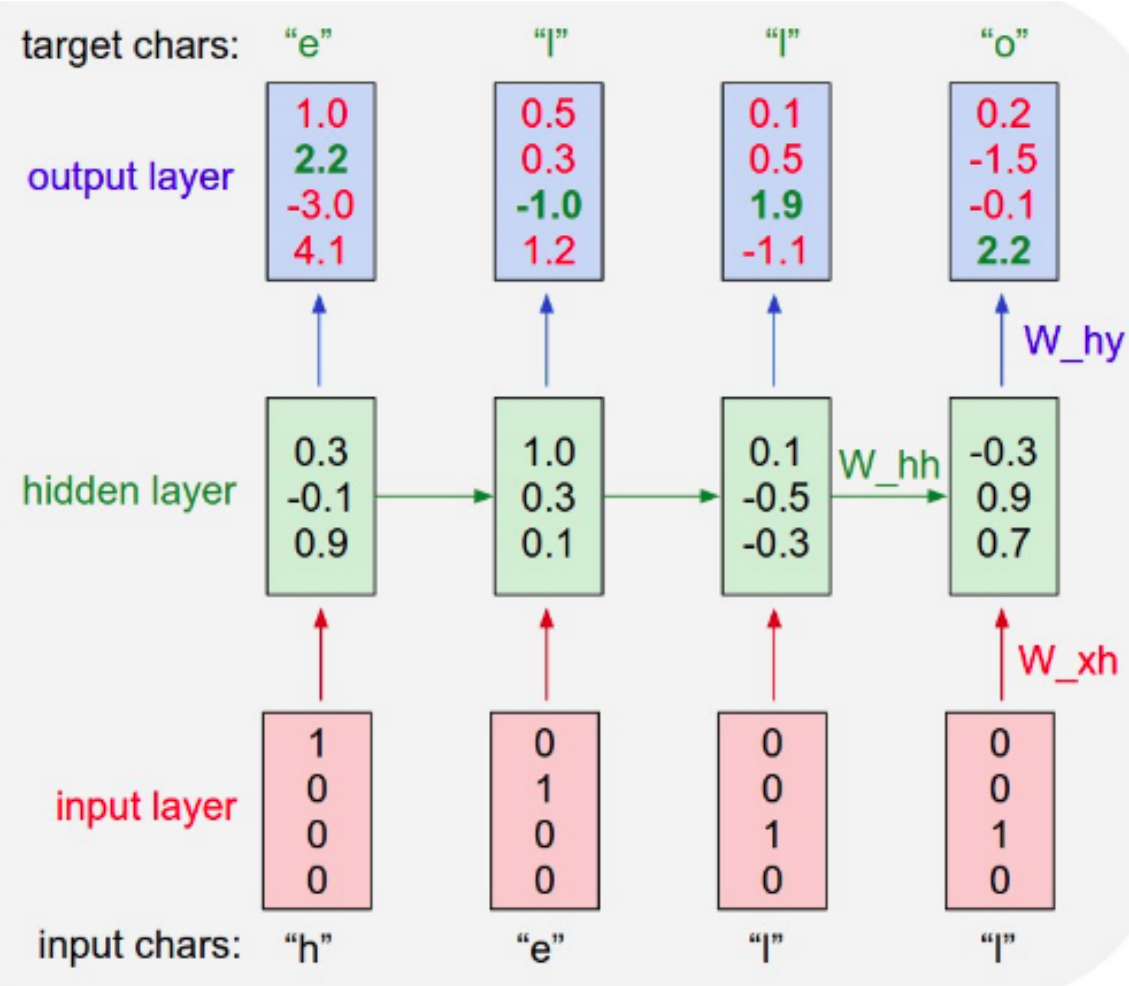


Ref: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

A simple example I

- Character-level language model (Ref: Karpathy):
  - Input: a huge chunk of text
  - Models: probability distribution of the next character in the sequence given a sequence of previous characters.
  - Usage: It will allow us to generate new text one character at a time.
- Suppose we only had a vocabulary of 4 possible letters "helo", and wanted to train an RNN on the training sequence "hell".
  - The probability of "e" should be likely given the context of "h"
  - "l" should be likely in the context of "he"
  - "l" should also be likely given the context of "hel"
  - "o" should be likely given the context of "hell"

A simple example II



- State allows the network to understand a "context".
- It uses its recurrent connections to keep track of the context.

Gradient analysis in RNN I

- Let's consider a simplified model (general case). Making the notations also simple.
  - h_t: hidden state
  - x_t: input
  - o_t: output at time-step t
- Let's denote some weights:
  - w_{hx}: $x \rightarrow h$
  - w_{hh}: $h \rightarrow h$
  - w_o: $h \rightarrow o$
- For convenience of explanation (and, also done in practice), we concatenate h_{t-1} and x_t; and the weights are then stacked. Let's call this weight as w_h.

Gradient analysis in RNN II

- These are the forward computation of hidden layer and output at every time step t:

$$h_t = f(x_t, h_{t-1}, w_h)$$

$$o_t = g(h_t, w_o)$$

  where, f and g are activations at hidden and output layer respectively.
- We have now a chain of values that are dependend on each other due to recurrence:

$$\{\ldots, (x_{t-1}, h_{t-1}, o_{t-1}), (x_t, h_t, o_t), \ldots\}$$

Gradient analysis in RNN III

- Now, for finishing up with the forward computation we need to *loop* through each of the triplets at every time-step: $t = 1, \ldots, T$ and compute a total loss:

$$L(x_1, \ldots, x_T, y_1, \ldots, y_T, w_h, w_o) = \frac{1}{T} \sum_{t=1}^{T} \ell(y_t, o_t)$$

Gradient analysis in RNN IV

- Recall that we need to use backprop for learning the parameters of the network. This will require
  - Gradient of L w.r.t. w_o (easy)
  - Gradient of L w.r.t. w_h (tricky)

Gradient analysis in RNN V

- The gradient of L w.r.t. w_h:

$$\begin{aligned}\frac{\partial L}{\partial w_h} &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial \ell(y_t, o_t)}{\partial w_h} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial \ell(y_t, o_t)}{\partial o_t}\frac{\partial o_t}{\partial h_t}\frac{\partial h_t}{\partial w_h} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{\partial \ell(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_h)}{\partial h_t}\frac{\partial h_t}{\partial w_h}\end{aligned}$$

  First and second term above: easy to compute

- The term $\frac{\partial h_t}{\partial w_h}$ is tricky to compute due to recurrence.

Gradient analysis in RNN VI

$\frac{\partial h_t}{\partial w_h}$:

- Recurrently compute the effect of w_h on h_t.
- h_t depends on both w_h and h_{t-1}; again h_{t-1} depends on w_h.
- Therefore,

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}$$

- Do you see a chain there in the red term above?

Gradient analysis in RNN VII

Let's now assume that we have 3 sequences: $\{a_t\}$, $\{b_t\}$, and $\{c_t\}$.

- Let these sequences satisfy: $a_0 = 0$ and $a_t = b_t + c_t a_{t-1}$ for $t = 1, 2, \ldots$
- For any $t \geq 1$, we can show that:

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} c_j \right) b_i$$

(Taken from d2l.ai)

Gradient analysis in RNN VIII

This is what we had:

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} c_j \right) b_i$$

Okay!!! Let's now substitute:

$$a_t = \frac{\partial h_t}{\partial w_h}$$

$$b_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}$$

$$c_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}$$

Gradient analysis in RNN IX

Therefore,

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

We are doing backprop through time (BPTT).

Do you see some potential problems:

- Q What will happen if t is large?
- Q What will happen if the bracketted term is < 0 or large?

Gradient analysis in RNN X

Strategies to deal with some problems in computing (note that full computation of the equation is not desirable in practice):

$$\sum_{i=1}^{t-1}\left(\prod_{j=i+1}^{t}\frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}}\right)\frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

Trucating time-steps (Jaeger, 2002):

- Truncate the sum after τ steps: $\frac{\partial h_{t-\tau}}{\partial w_h}$ (leads to an approximation of the true gradient)
- In practice, this works quite well. It is referred to as truncated backpropgation through time.
- The model focuses primarily on short-term influence rather than long-term consequences.
- Outcome is a simple and stable model.

Gradient analysis in RNN XI

Read:

- Randomized Truncation (Tallec & Ollivier, 2017) that uses a random variable to replace $\frac{\partial h_t}{\partial w_h}$.
- Gradient clipping: limits the magnitude of the gradient. (Textbook 2: Chapter 10 on RNNs)

RNNs are good for handling short-term dependencies. To deal with long-term dependencies, the structure of the RNN needs change. (e.g. LSTM, GRU cells)

Gradient analysis in RNN XII

While explaining this, in the lecture you will see the following example:

```
The input is the characters from a C program.  The system
will tell whether it is a syntactically correct program.  A
syntactically correct program should have a valid number of
braces and parentheses....
```

This is a bad example. You DO NOT need machine learning for the above problem. A simple regex program could do the job. Just because you are doing machine learning, don't just use it where you don't need it.