Recurrent Neural Network for NLP

Sequential NLP Data

Sequential Data

- Characters in words
- Words in sentences
- Sentences in paragraphs
- Paragraphs in documents

NLP is full of **sequential** data

Dependencies in Language

Dependencies: the relationship between two words (it can be semantic or syntax)

- It is equal to the sequential information contained in the sequence data.

Long-distance Dependencies in Languages

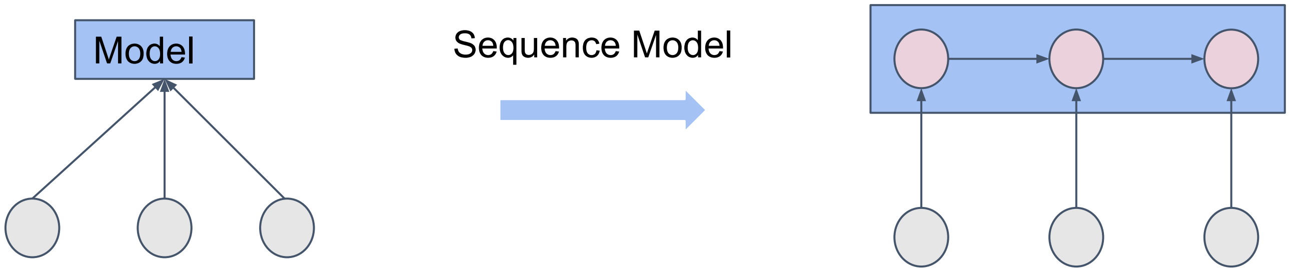
Examples

- He does not have very much confidence in himself

- The rain has lasted as long as the life of clouds

- The *trophy* would not fit in the brown *suitcase* because it was too small

Sequential NLP Data



Machine learning models should capture this kind of sequential information in NLP data.

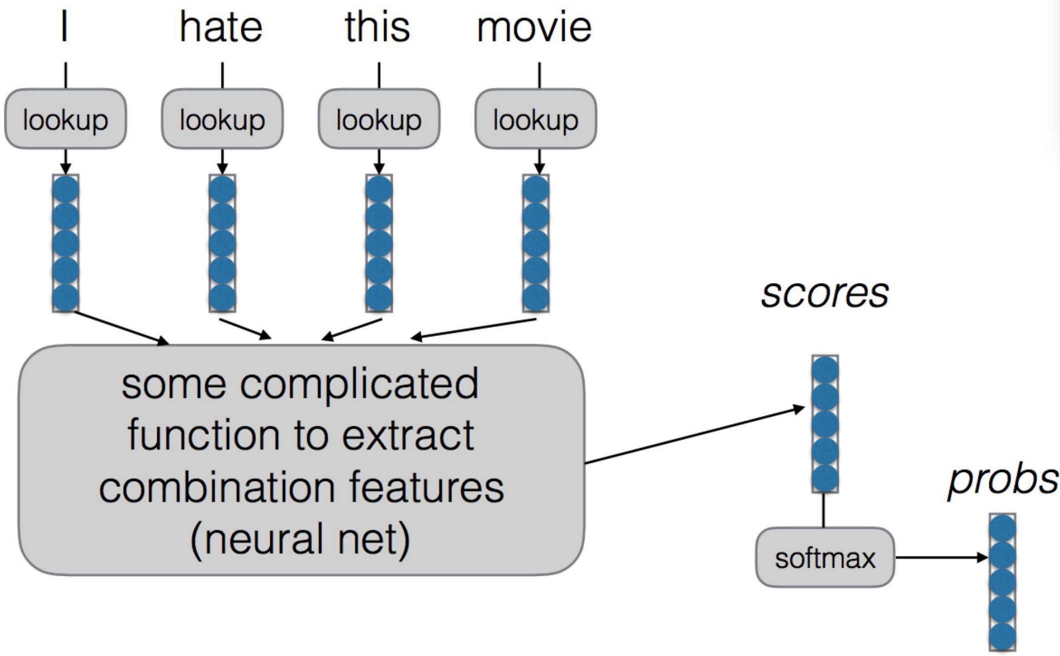
Complex Semantic

1. **Input Text:** a sequence of words;

2. **Through Word Embedding Look-up:**
a sequence of word vectors;

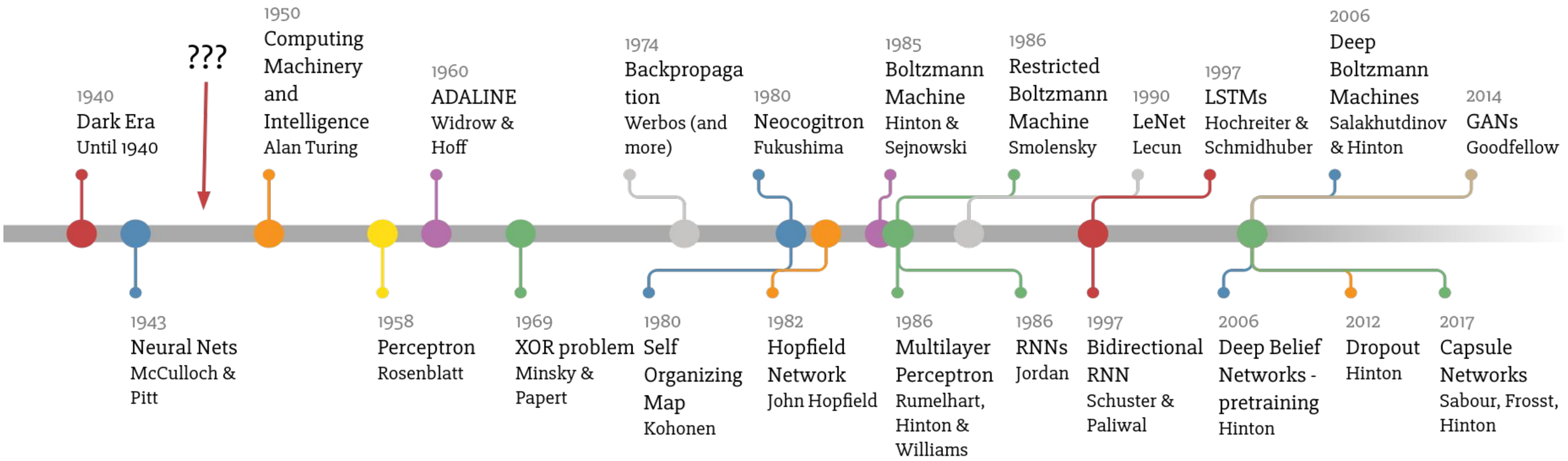
3. Neural networks is applied upon the vector sequences to learn semantic **composition** for final prediction;

➔ Human understand the word meaning firstly, then get the whole sentence meaning by composing these words' meaning together.



Intro to RNN

Deep Learning Timeline



Made by Favio Vázquez

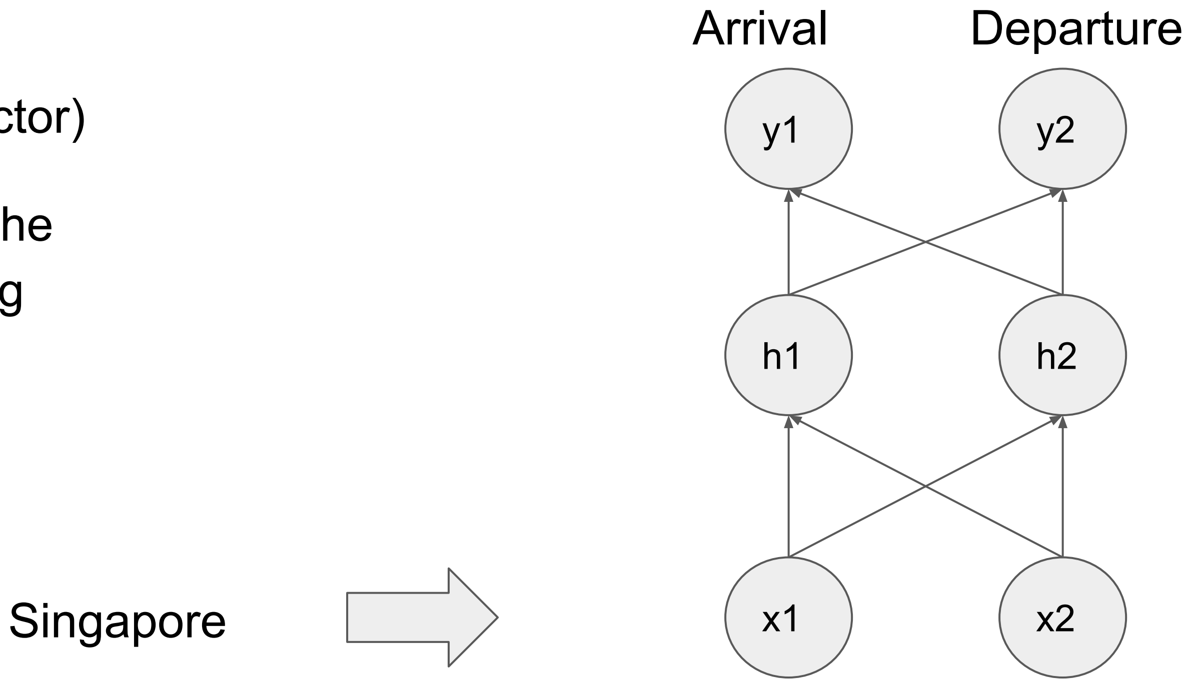
Examples

- Tagging Task: Assuming we have a predefined set of tags, we assign a tag to each word in input sentences.
- Given an input sentence: I would like to arrive at Singapore on Mar. 29.

Singapore -> Dest. Place
Mar. 29 -> Time of Arrival

Feed-forward Network

- Input: Each word (word vector)
- Output: Prob. Scores that the input word belong to the tag



Confusing Case

- Input Sentence 1:

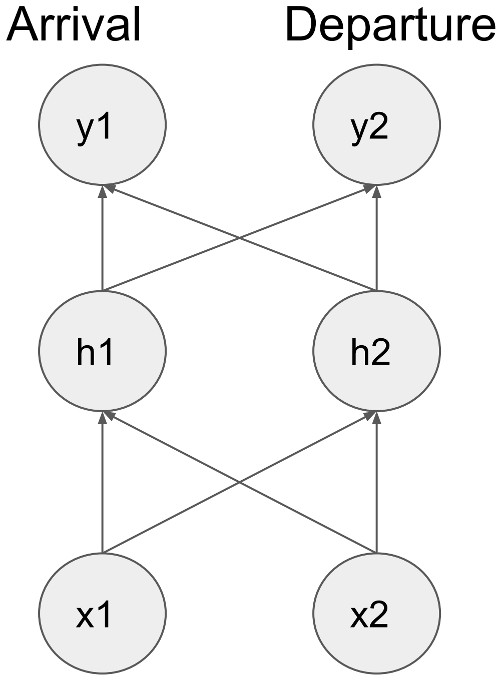
  I would like to arrive at Singapore on Mar. 29.

- Input Sentence 2:

  I would like to leave Singapore on Mar. 29.

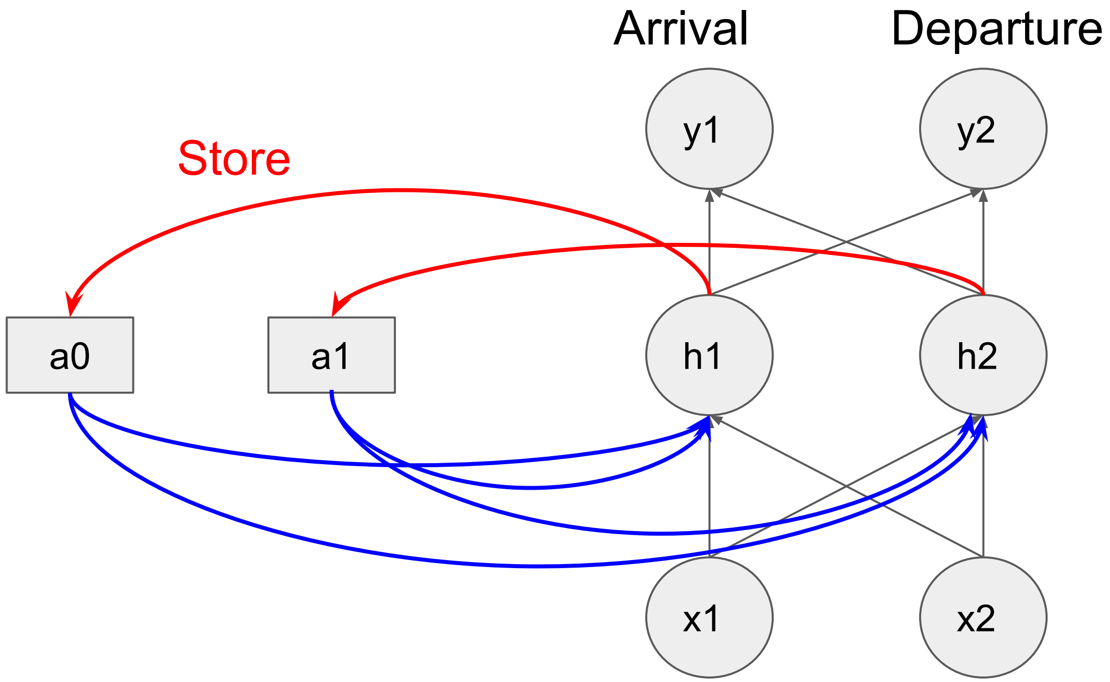
Neural Network needs Memory

Singapore



Neural Network with Memory

The outputs of hidden layer are stored in the memory.

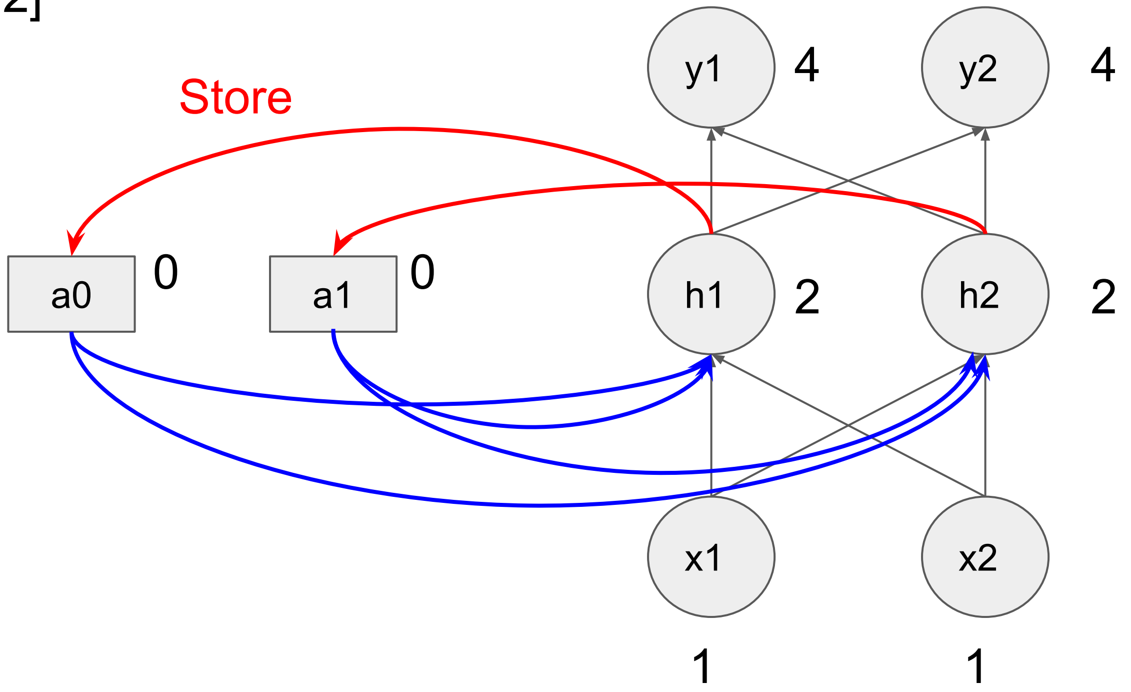


Memory can be considered as another input

Neural Network with Memory

Input Seq　:　[1, 1]　[1, 1]　[2, 2]
Output Seq:　[4, 4]

Given initial values

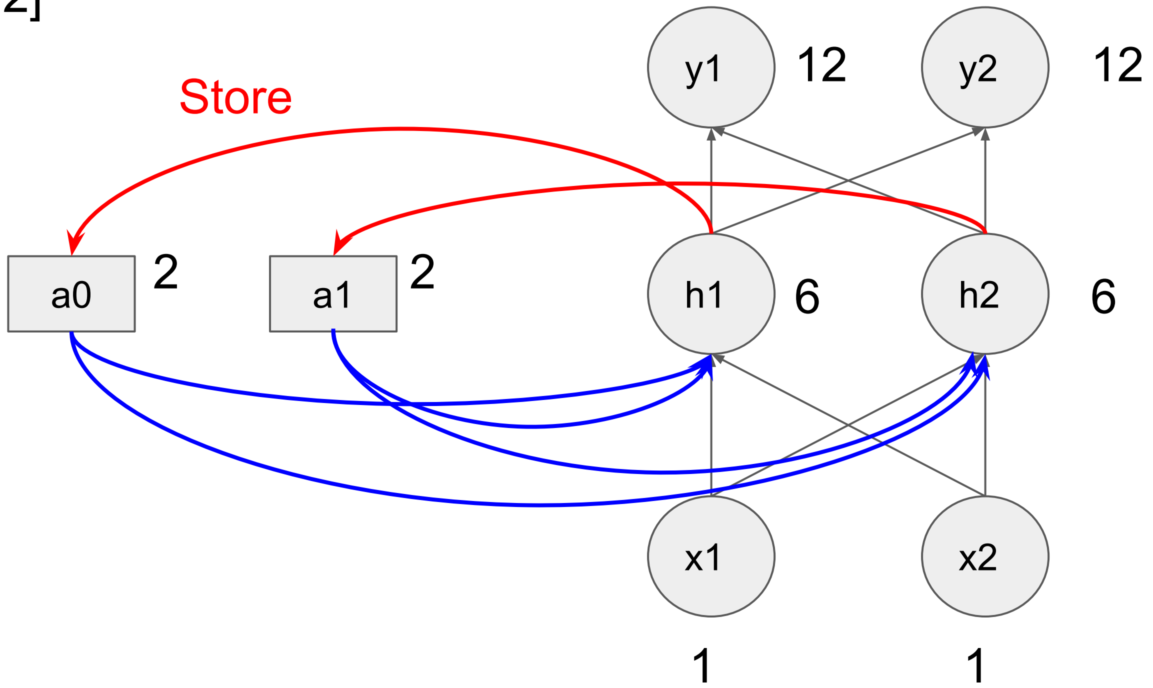


All weights are 1, no bias, linear activation

Neural Network with Memory

Input Seq : [1, 1] [1, 1] [2, 2]
Output Seq: [4, 4] [12, 12]

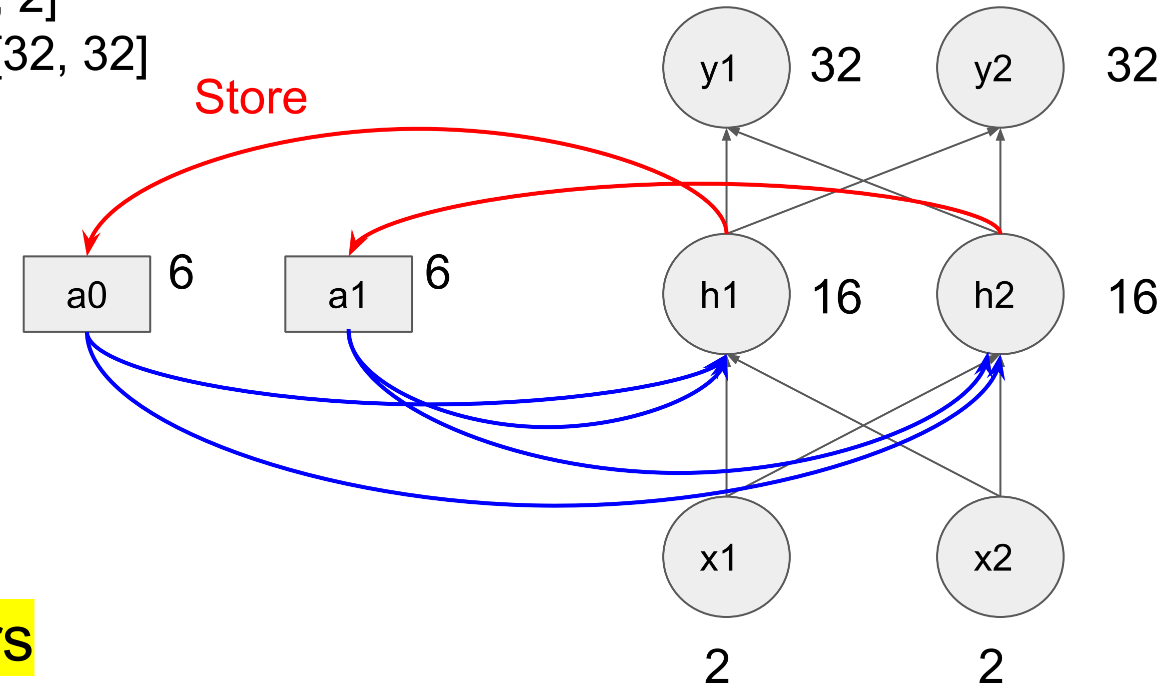
Using previous hidden output



All weights are 1, no bias, linear activation

Neural Network with Memory

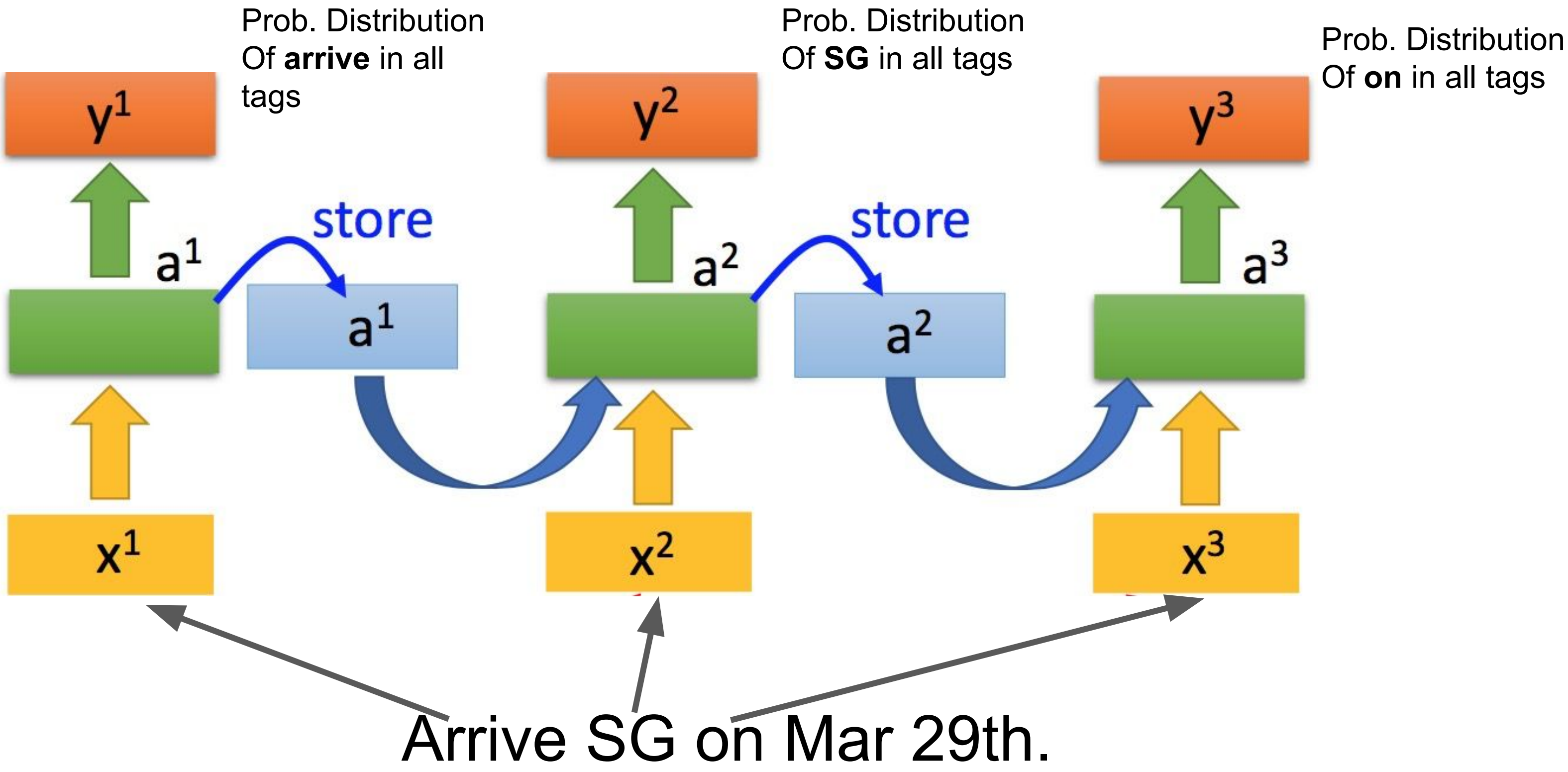
Input Seq :   [1, 1]   [1, 1]   [2, 2]
Output Seq: [4, 4]   [12, 12] [32, 32]



Try [1, 1] [2, 2], [1,1]

Sequence Order Matters

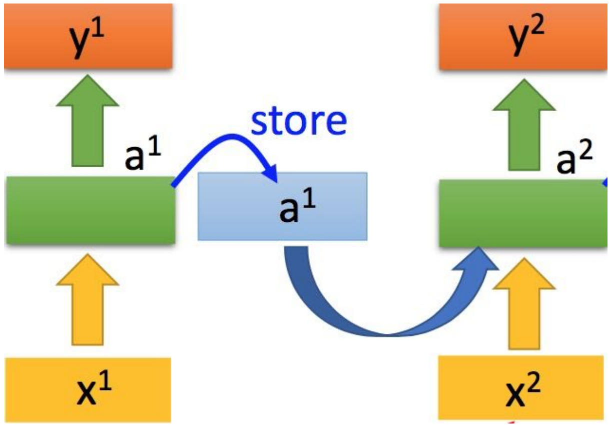
Back to Our Case



Back to Our Case

Prob. Distribution Of **arrive** in all tags

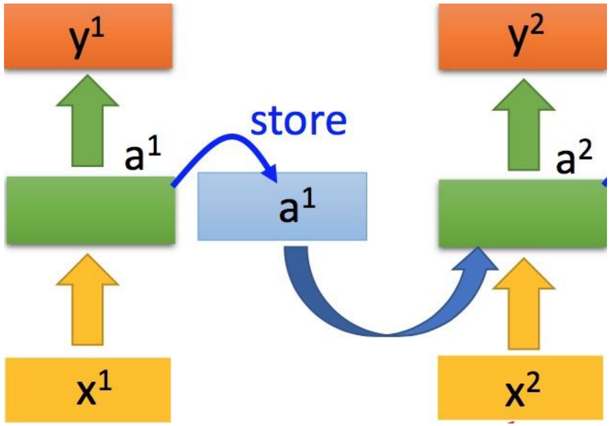
Prob. Distribution Of **SG** in all tags



Arrive SG

Prob. Distribution Of **leave** in all tags

Prob. Distribution Of **SG** in all tags

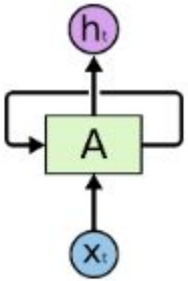


Leave SG

Are these two probs. distribution of SG same ? And Why?

Recurrent Neural Network (Elman 1990)

- Recurrent neural network is proposed to utilize information from **previous** time steps and **current** information to make reasoning at the current step



- A is the neural network that take x_t in and generate h_t

Image credits to Colah.

- **A loop** allows information to be passed from one step of the network to the next.

Allows information to persist

Neuron computation of RNN

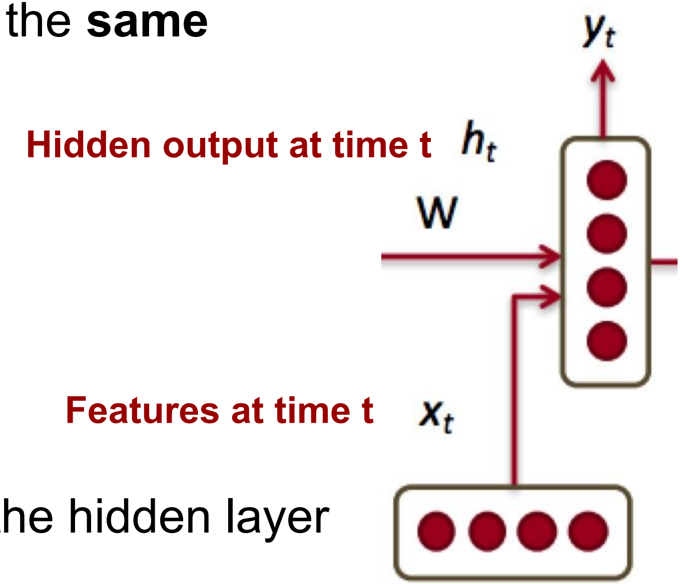
- Model parameters are tied across all time steps (run the **same** RNN cell)

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

$$\hat{y}_t = \text{softmax}\left(W^{(S)} h_t\right)$$

- We need $h_0 \in \mathbb{R}^{D_h}$ as the initialization vector for the hidden layer at time step **0**.

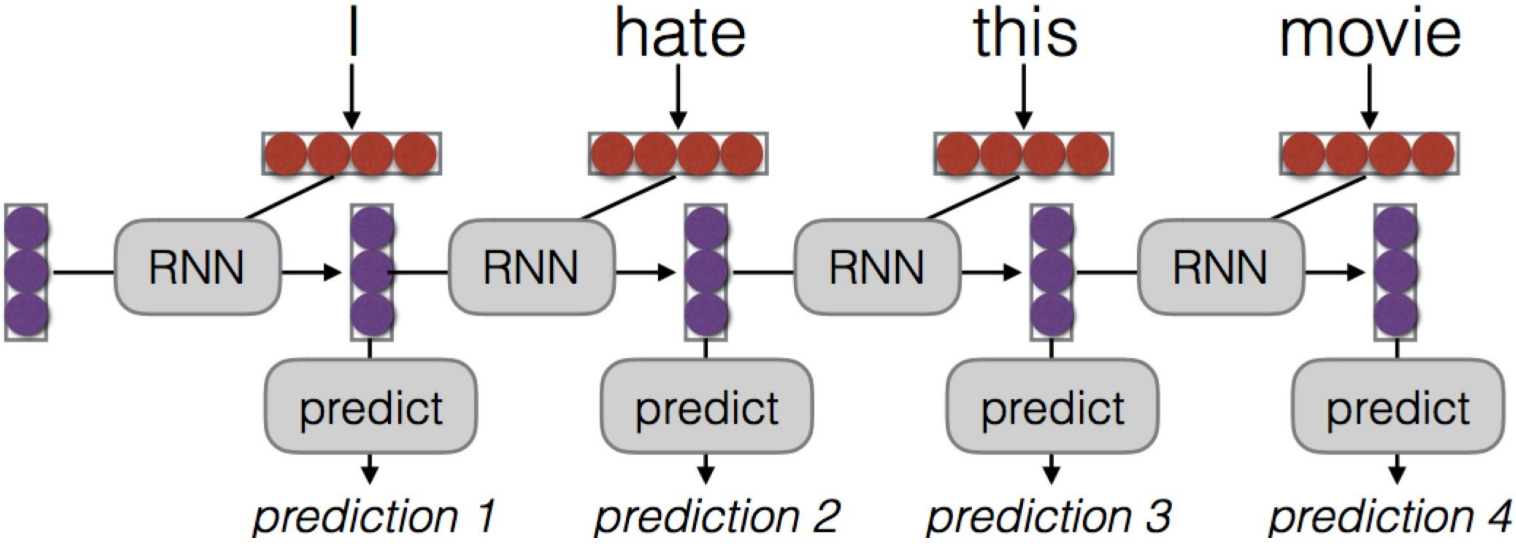
- Inputs enter and move forward at each time step



Focus on certain time step

Unrolling in Time

- Process the NLP sequence data



RNN’s Bottleneck

- RNN is not suitable for parallel computation.

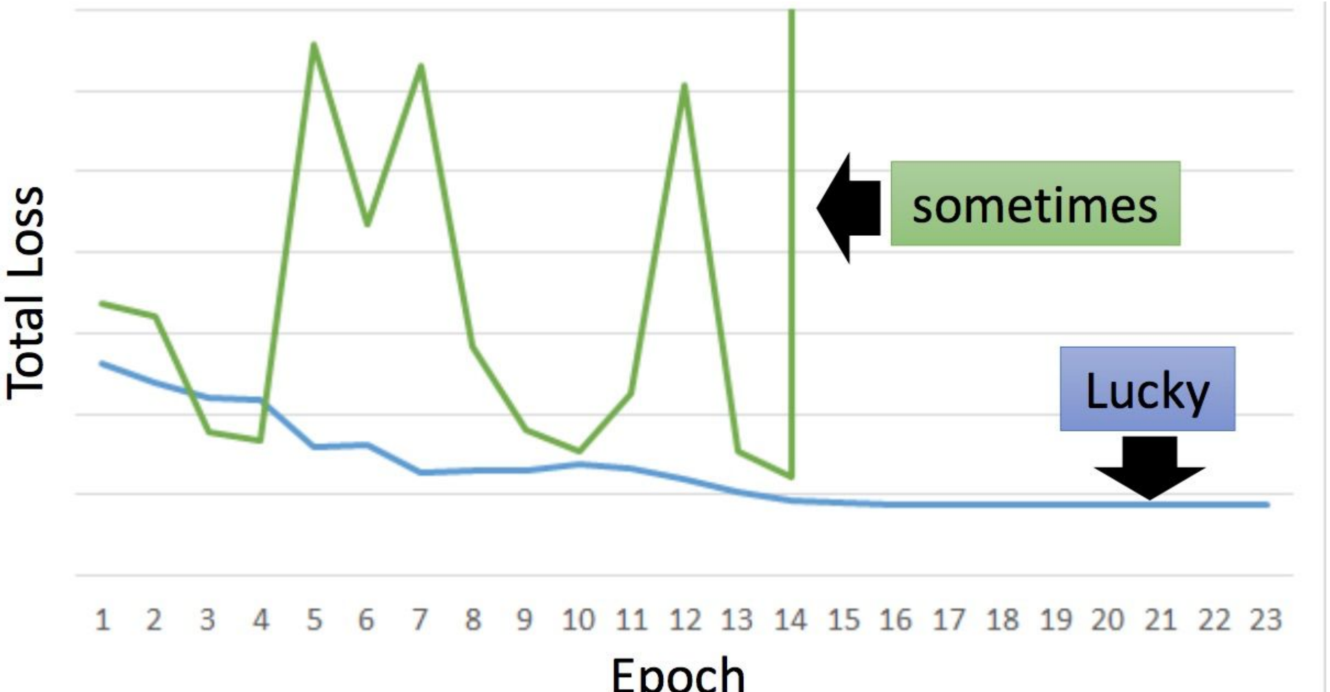
- RNN’s training is not easy
    - Gradient Vanishing
    - Gradient Exploding
    - Reminder on gradient descent algorithms:

$$W = W - \alpha \, dL/dW$$

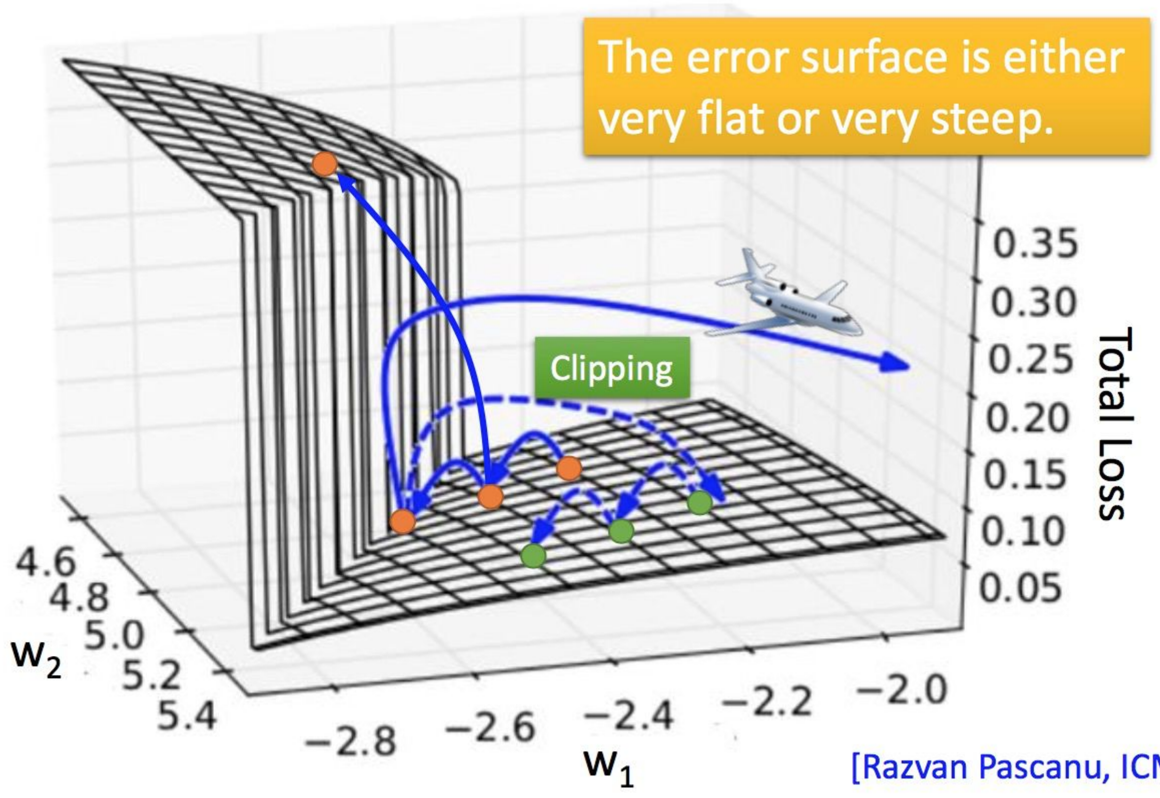
W is the layer weights;
L is the loss function;
Alpha is the learning rate, which represents how aggressive that our model parameters are updated.

RNN Training is Hard

- Real experiments on Language Models



Rough Error Surface of RNN



Exploding/Vanishing Gradient

[Razvan Pascanu, ICML'13]

Toy Example

$w = 1 \Rightarrow y_{1000} = 1$

$w = 1.01 \Rightarrow y_{1000} = 20000$

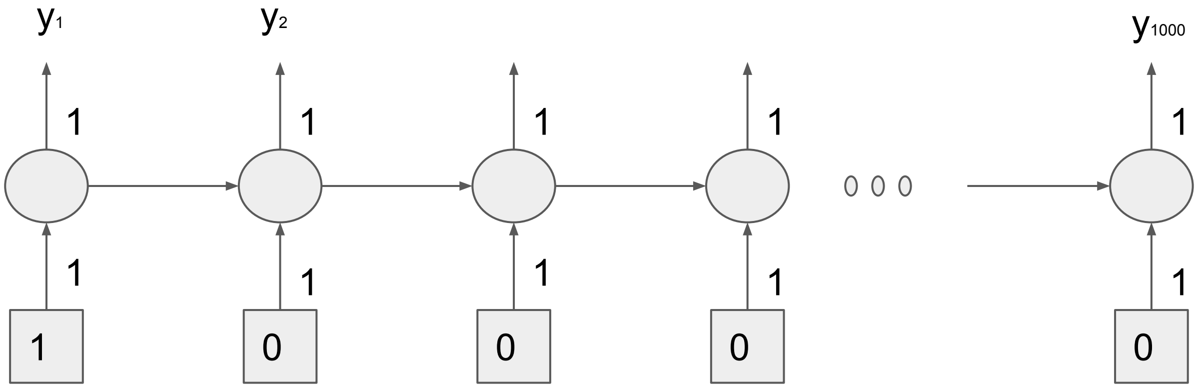
$\Rightarrow$ Large Gradient: dL/dw

$w = 0.99 \Rightarrow y_{1000} \sim= 0$

$w = 0.01 \Rightarrow y_{1000} \sim= 0$

$\Rightarrow$ Small Gradient: dL/dw

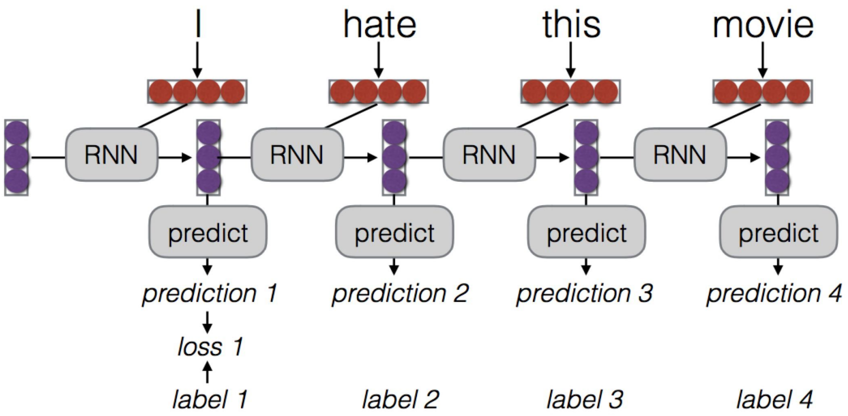
Irregular Error Surface

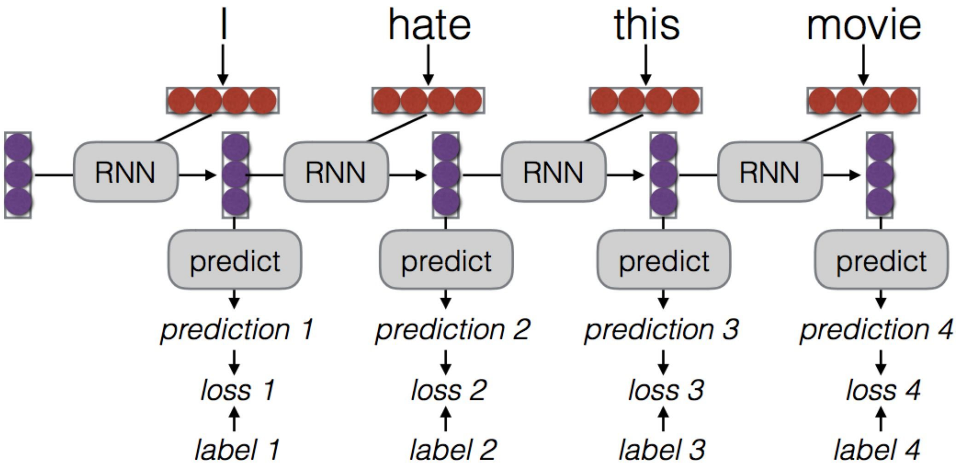


All weights are 1,
no bias,
linear activation

Loss Computation

- Total Loss
  - Total loss is the sum of loss computed on each time step.

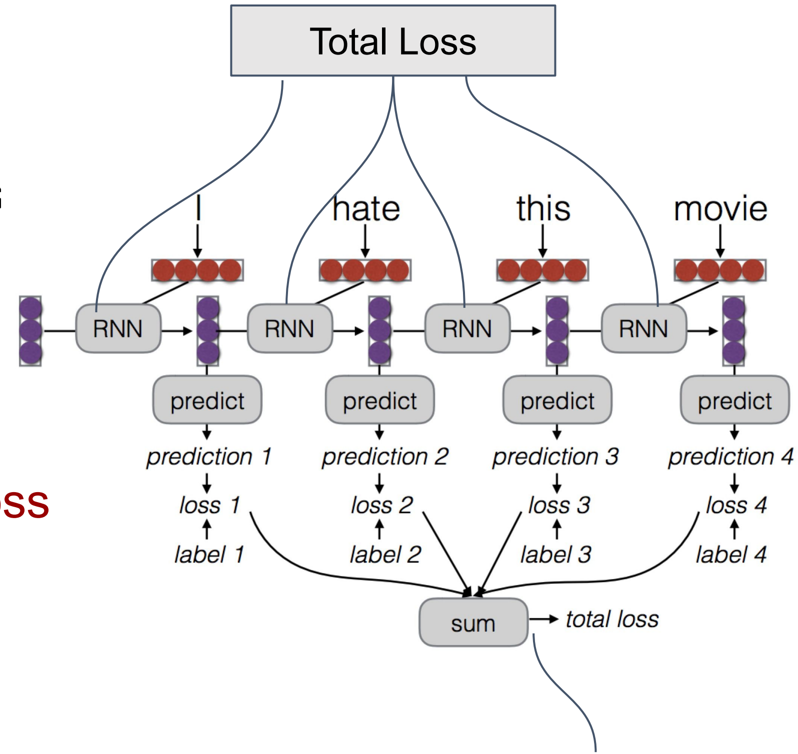




BPTT

- For RNN gradient computation, Backpropagation Through Time
- Weight is the same for all the time steps
- Inputs from many time step ago can modify output

1. Shared Model Parameters across time steps
2. Accumulated Derviatives



BPTT Equation

- RNN formulation

$$h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right)$$
$$\hat{y}_t = \mathrm{softmax}\left(W^{(S)}h_t\right)$$

- **Total error** is the sum of each error at time steps **t**

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

- Apply **chain rule** for error at certain time step **t**

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

Exploding Gradient Solutions

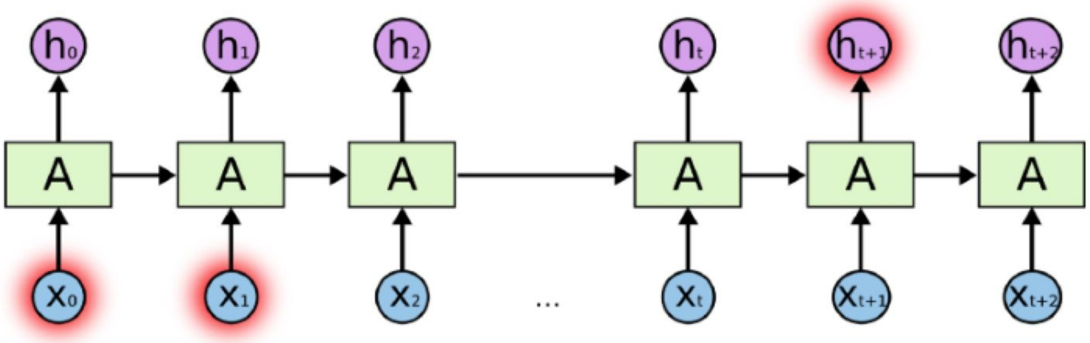
- Truncated BPTT
- Clip gradients at threshold
- RMSprop to adjust learning rate

Vanishing Gradient Problem

- The error at a time step **ideally** can tell a previous time step from many steps away to change during backprop
- **Can not capture long-term dependency**

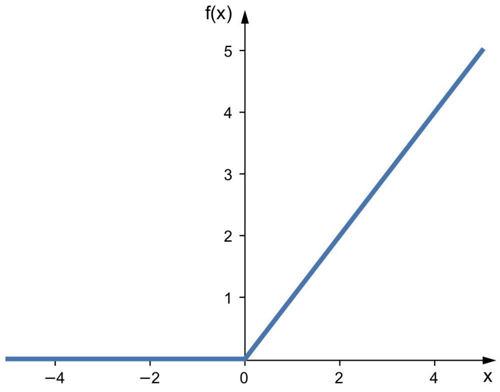
- The representation from time steps 0 and t can not travel to influence the time step t+1

- **Harder to detect**



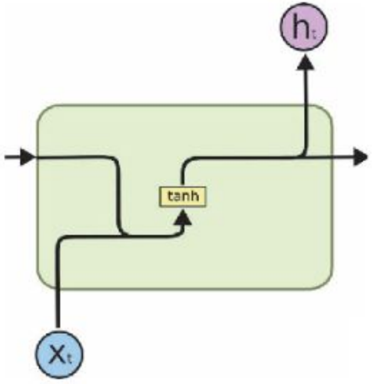
Vanishing Gradient Solutions

- RMSprop
- LSTM, GRUs (gated RNN)
- ReLu activation functions

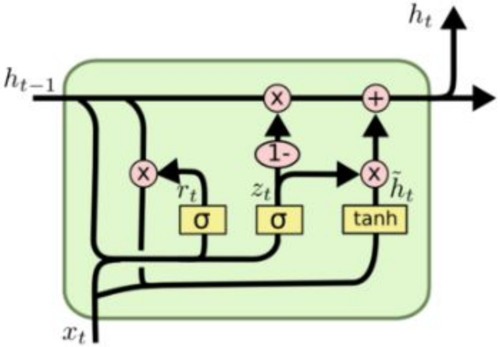


LSTM/GRU

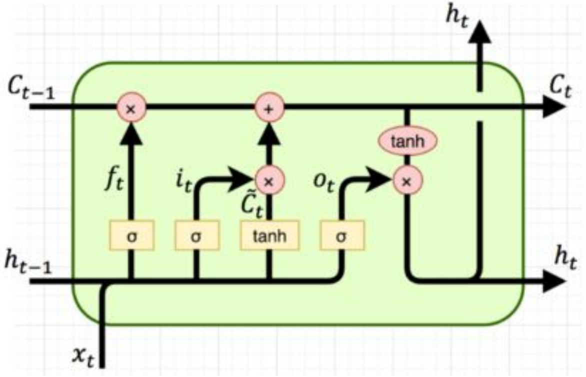
- LSTM/GRU are using gates in cell computation to control information flow



RNN



GRU



LSTM

RNN for Language Model

Recurrent Neural Network

- Recurrent neural network works in a chain way
- The method is in naturally suitable for processing sequences data
- A broad applications:
    - Speech Recognition
    - Time series Prediction
    - Language Modeling
    - Machine Translation

Language Models

- A language model computes **a probability for a sequence of words**:
    - P(w1,..., wT)
- Useful for machine translation/Question Answering
    - Word Ordering
        - p(the cat is small) > p(small is the cat)
    - Word Choice
        - p(walking home after school) > p(walking house after school)

Traditional Methods

- Probability is usually conditioned on **window of *n*** previous words
- An incorrect but practical **Markov** Assumption
- To estimate probabilities, compute for unigram and bigrams

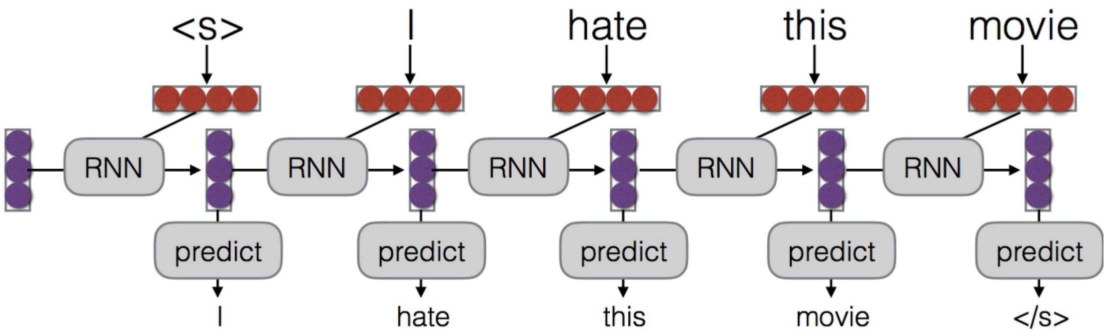
$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \qquad p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

Traditional Methods

- A lot of n-grams and extremely large combinations
    - Requires large RAM requirements
- Use one machine with 140GB RAM for 2.8days to built a model on 126 billion tokens.

RNN-based Language Model

- Language Modelling here is one of **tagging** task
- Each label/tag is the next word!



At a single time step:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

$$\hat{y}_t = \mathrm{softmax}\left(W^{(S)} h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$

RNN-based Language Model

- This task is for sequential labelling, i.e., each time step has its own target value
- At each time step, the learned hidden representation are fed into two layers:
    - Predict the target value (softmax or linear regression)
    - Compute the hidden features at the next time step