UC San Diego
JACOBS SCHOOL OF ENGINEERING

ECE/SIOC 228 Machine Learning for Physical Applications
Lecture 7: Recurrent Neural Networks

Yuanyuan Shi

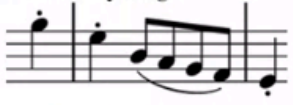
Assistant Professor, ECE

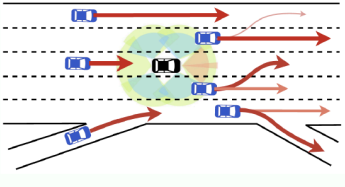
University of California, San Diego

Reference & Acknowledgement: Stanford CS230 Deep Learning Lecture Note, Andrew Ng
Dive into Deep Learning, https://d2l.ai/chapter_recurrent-neural-networks/index.html

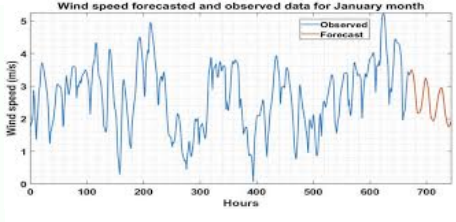
Examples of Sequence Data

UC San Diego

Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	Ø	→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?



trajectory prediction



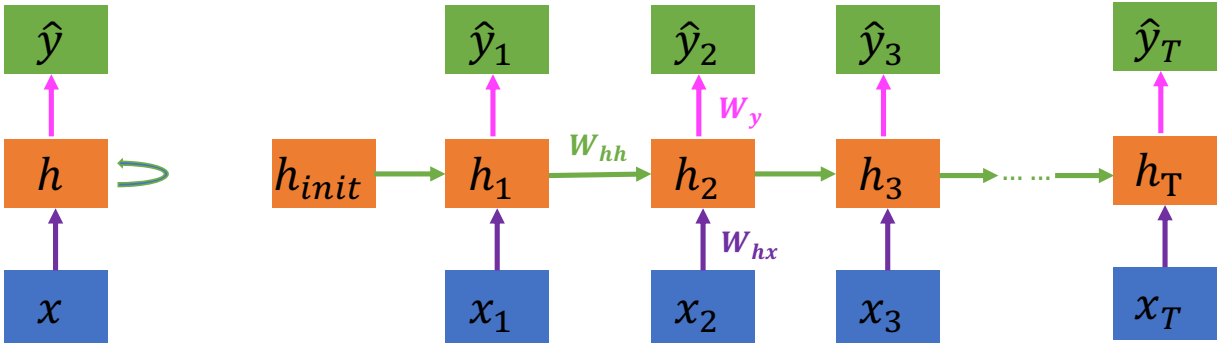
Outline

UC San Diego

- **Recurrent Neural Networks**
- Modern Recurrent Neural Networks
- Attention Mechanism and Transformer

Recurrent Neural Networks

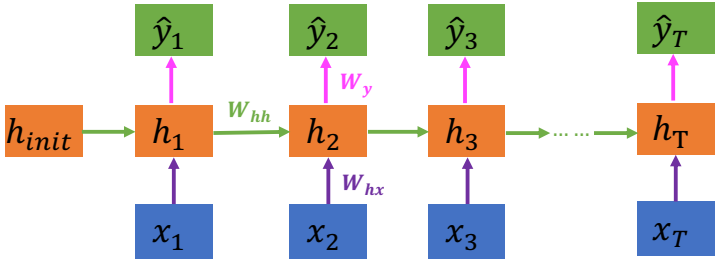
UC San Diego

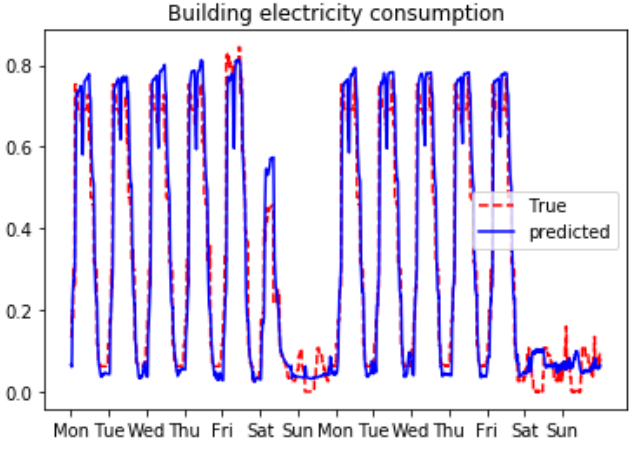


- Computation takes into historical info
- weights are shared across time — each set of colored arrows denotes the same weights
- Process input of any length; model size not increase with size of input

Recurrent Neural Networks: Example

UC San Diego

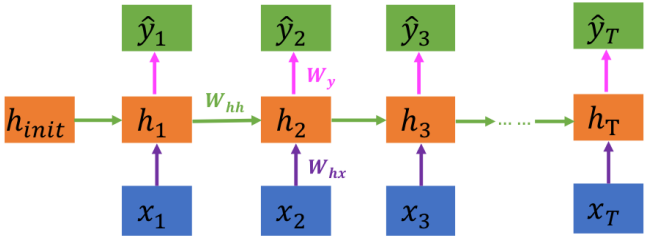




- $\hat{y}_t$ denotes the predicted electricity consumption, at time t
- x_t denotes the features that affects the building energy consumption, e.g., environment features (temperature, humidity, …), occupancy, room temperature setpoint…
- h_t encodes the effect of historical $[x_1, \ldots, x_{t-1}]$

Forward Computation

UC San Diego



for $t = 1, \cdots, T$

$$h_t = \sigma(W_{hh} h_{t-1} + W_{hx} x_t + b_h)$$

$$\hat{y}_t = \sigma(W_y h_t + b_y)$$

Loss function

$$L = \frac{1}{T} \sum_{t=1}^{T} l(\hat{y}_t, y_t)$$

Suppose we have n traing samples:

$x_t \in R^{d \times n}$, $h_t \in R^{h \times n}$, $\hat{y}_t \in R^{q \times n}$

$W_{hx} \in R^{h \times d}$

$W_{hh} \in R^{h \times h}$

$W_y \in R^{q \times h}$

Two simplification: 1) ignore bias b_h, b_y; 2) set activation $\sigma(z) = z$

$$\frac{\partial L}{\partial \hat{y}_t} = \frac{1}{T} \frac{\partial l(y_t, \hat{y}_t)}{\partial \hat{y}_t} \in R^{q \times n}$$

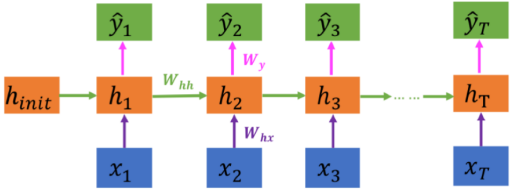
Now we compute gradient w.r.t. W_y (parameter of the output layer)

L depends on W_y via $\hat{y}_1, \hat{y}_2, \cdots \hat{y}_T$

$$\frac{\partial L}{\partial W_y} = \sum_{t=1}^{T} \frac{\partial L}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial W_y} = \sum_{t=1}^{T} \frac{\partial L}{\partial \hat{y}_t} h_t^T$$

Backpropagation

UC San Diego



for $t = 1, \cdots, T$

$h_t = \sigma(W_{hh} h_{t-1} + W_{hx} x_t + b_h)$

$\hat{y}_t = \sigma(W_y h_t + b_y)$

Loss function

$L = \frac{1}{T} \sum_{t=1}^{T} l(\hat{y}_t, y_t)$

Now, we look at gradient w.r.t. h_t

$$\frac{\partial L}{\partial h_T} = \text{prod}\left(\frac{\partial L}{\partial \hat{y}_T}, \frac{\partial \hat{y}_T}{\partial h_T}\right) = W_y^T \frac{\partial L}{\partial \hat{y}_T}$$

For $t < T$, we want to compute $\frac{\partial L}{\partial h_t}$ suppose $\frac{\partial L}{\partial h_{t+1}}$ and $\frac{\partial L}{\partial \hat{y}_t}$ are computed

$$\frac{\partial L}{\partial h_t} = \text{prod}\left(\frac{\partial L}{\partial h_{t+1}}, \frac{\partial h_{t+1}}{\partial h_t}\right) + \text{prod}\left(\frac{\partial L}{\partial \hat{y}_t}, \frac{\partial \hat{y}_t}{\partial h_t}\right)$$

$$= W_{hh}^T \frac{\partial L}{\partial h_{t+1}} + W_y^T \frac{\partial L}{\partial \hat{y}_t}$$

$$\frac{\partial L}{\partial h_t} = W_{hh}^T \frac{\partial L}{\partial h_{t+1}} + W_y^T \frac{\partial L}{\partial \hat{y}_t}$$

$$= W_{hh}^T \left(W_{hh}^T \frac{\partial L}{\partial h_{t+2}} + W_y^T \frac{\partial L}{\partial \hat{y}_{t+1}} \right) + W_y^T \frac{\partial L}{\partial \hat{y}_t}$$

$$= (W_{hh}^T)^2 \cdot \frac{\partial L}{\partial h_{t+2}} + W_{hh}^T W_y^T \frac{\partial L}{\partial \hat{y}_{t+1}} + W_y^T \frac{\partial L}{\partial \hat{y}_t}$$

$$= (W_{hh}^T)^2 \cdot \left[W_{hh}^T \frac{\partial L}{\partial y_{t+3}} + W_y^T \frac{\partial L}{\partial \hat{y}_{t+2}} \right] + W_{hh}^T W_y^T \frac{\partial L}{\partial \hat{y}_{t+1}} + W_y^T \frac{\partial L}{\partial \hat{y}_t}$$

$$= \cdots = \sum_{i=t}^{T} (W_{hh}^T)^{T-i} W_y^T \frac{\partial L}{\partial \hat{y}_{T+t-i}}$$

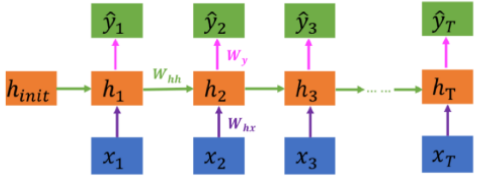
$$\frac{\partial L}{\partial h_t} = \sum_{i=t}^{T} (W_{hh}^T)^{T-i} W_y^T \frac{\partial L}{\partial \hat{y}_{T+t-i}}$$

- It involves (potentially) very large power of W_{hh}^T
- If $\text{eigen}(W_{hh}^T) > 1$, diverge (gradient exploding)
- If $\text{eigen}(W_{hh}^T) < 1$, very small (gradient vanishing)

☆ Truncate the time steps $\frac{\partial L}{\partial h_t} = \sum_{i=t}^{t+H} (W_{hh}^T)^{t+H-i} W_y^T \frac{\partial L}{\partial \hat{y}_{t+H+t-i}}$

Backpropagation

UC San Diego



for $t = 1, \cdots, T$

$h_t = \sigma(W_{hh} h_{t-1} + W_{hx} x_t + b_h)$

$\hat{y}_t = \sigma(W_y h_t + b_y)$

Loss function

$L = \frac{1}{T} \sum_{t=1}^{T} l(\hat{y}_t, y_t)$

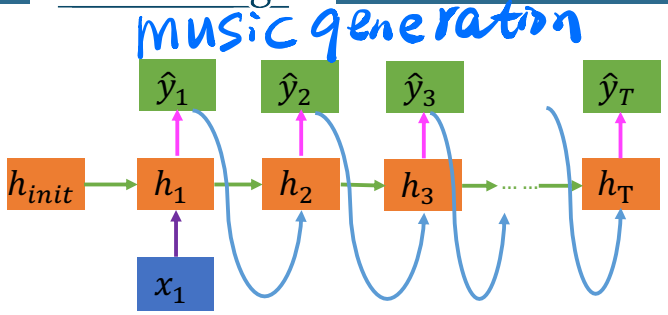
$$\frac{\partial L}{\partial W_{hx}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial h_t}, \frac{\partial h_t}{\partial W_{hx}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} x_t^T$$

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial h_t}, \frac{\partial h_t}{\partial W_{hh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} h_{t-1}^T$$

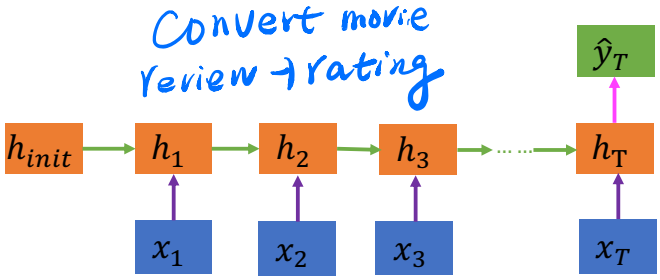
- $\frac{\partial L}{\partial h_t}$ also affect the gradient magnitude $\frac{\partial L}{\partial W_{hx}}, \frac{\partial L}{\partial W_{hh}}$
- Traing RNN alternates forward & backward propogation through time

Different Types of RNNs

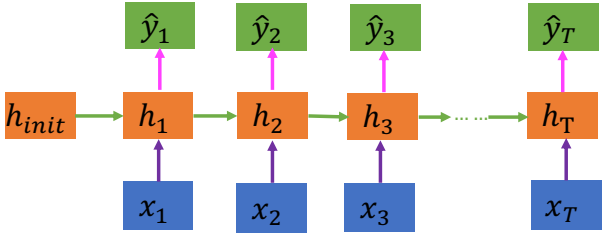
UC San Diego



One to Many

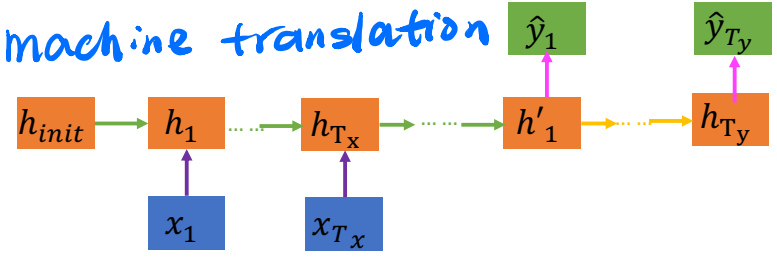


Many to One



Many to Many

building energy prediction



Many to Many (input and output have different length)

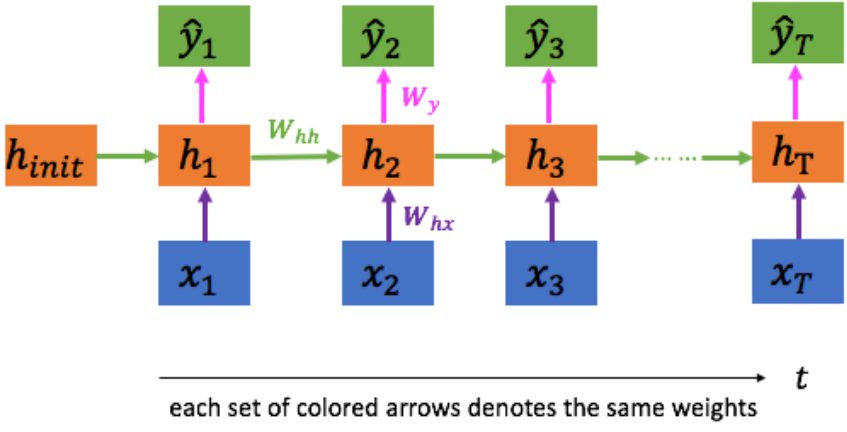
Outline

UC San Diego

- Recurrent Neural Networks
- **Modern Recurrent Neural Networks**
- Attention Mechanism and Transformer

Vanishing Gradient with RNN

UC San Diego

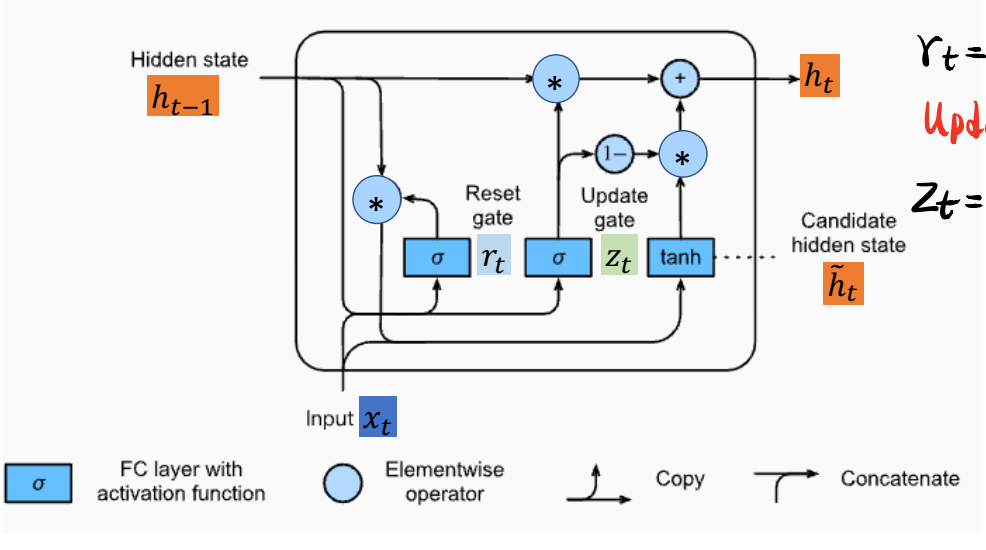


- Long products of matrices can lead to vanishing or exploding gradients

$$\frac{\partial L}{\partial h_t} = \sum_{i=t}^{T} (W_{hh}^T)^{T-i} W_y^T \frac{\partial L}{\partial \hat{y}_{T+t-i}}$$

Gated Recurrent Units (GRU)

UC San Diego



Reset gate:

$$r_t = \text{sigmoid}(W_{rx} x_t + W_{rh} h_{t-1} + b_r)$$

Update gate:

$$z_t = \text{sigmoid}(W_{zx} x_t + W_{zh} h_{t-1} + b_z)$$

$x_t \in \mathbb{R}^{d \times n}$, $h_{t-1} \in \mathbb{R}^{h \times n}$,

$r_t, z_t \in \mathbb{R}^{h \times n}$

$W_{rx}, W_{zx} \in \mathbb{R}^{h \times d}$

$W_{rh}, W_{zh} \in \mathbb{R}^{h \times h}$

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling: https://arxiv.org/pdf/1412.3555.pdf

Gated Recurrent Units (GRU)

UC San Diego

Reset gate: $r_t = \text{sigmoid}(W_{rx} x_t + W_{rh} h_{t-1} + b_r)$

Update gate: $z_t = \text{sigmoid}(W_{zx} x_t + W_{zh} h_{t-1} + b_z)$

proposal / candidate hidden state:

$$\tilde{h}_t = \tanh(W_{hx} x_t + W_{hh}(r_t * h_{t-1}) + b_h)$$

element-wise

- If $r_t \approx 1$, recover RNN
- If $r_t \approx 0$, $\tilde{h}_t$ only depends on x_t, resetting hidden states

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling: https://arxiv.org/pdf/1412.3555.pdf

Gated Recurrent Units (GRU)

UC San Diego

Reset gate: $r_t = \text{sigmoid}(W_{rx} x_t + W_{rh} h_{t-1} + b_r)$

Update gate: $z_t = \text{sigmoid}(W_{zx} x_t + W_{zh} h_{t-1} + b_z)$

Proposal: $\tilde{h}_t = \tanh(W_{hx} x_t + W_{hh}(r_t * h_{t-1}) + b_h)$

Hidden state: $h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t$

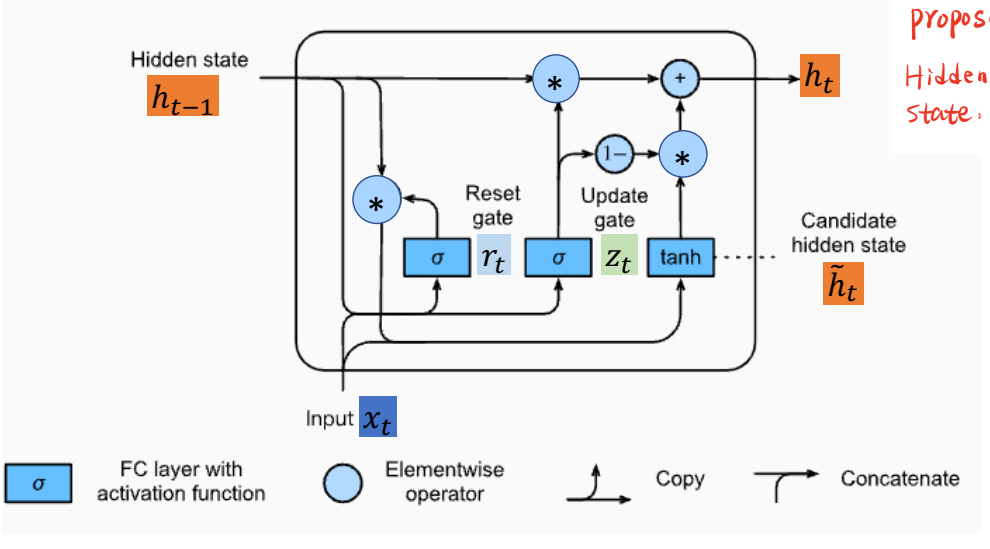
($*$: elementwise)

- If $z_t \approx 1$, $h_t = h_{t-1}$; (help cope the vanishing gradient)
- If $z_t \approx 0$, $h_t = \tilde{h}_t$ (incorporate new info)

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling: https://arxiv.org/pdf/1412.3555.pdf

Gated Recurrent Units (GRU)

UC San Diego



Reset gate: $r_t = \text{sigmoid}(W_{rx} x_t + W_{rh} h_{t-1} + b_r)$

Update gate: $z_t = \text{sigmoid}(W_{zx} x_t + W_{zh} h_{t-1} + b_z)$

Proposal: $\tilde{h}_t = \tanh(W_{hx} x_t + W_{hh}(r_t * h_{t-1}) + b_h)$

Hidden state: $h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t$

($*$: elementwise)

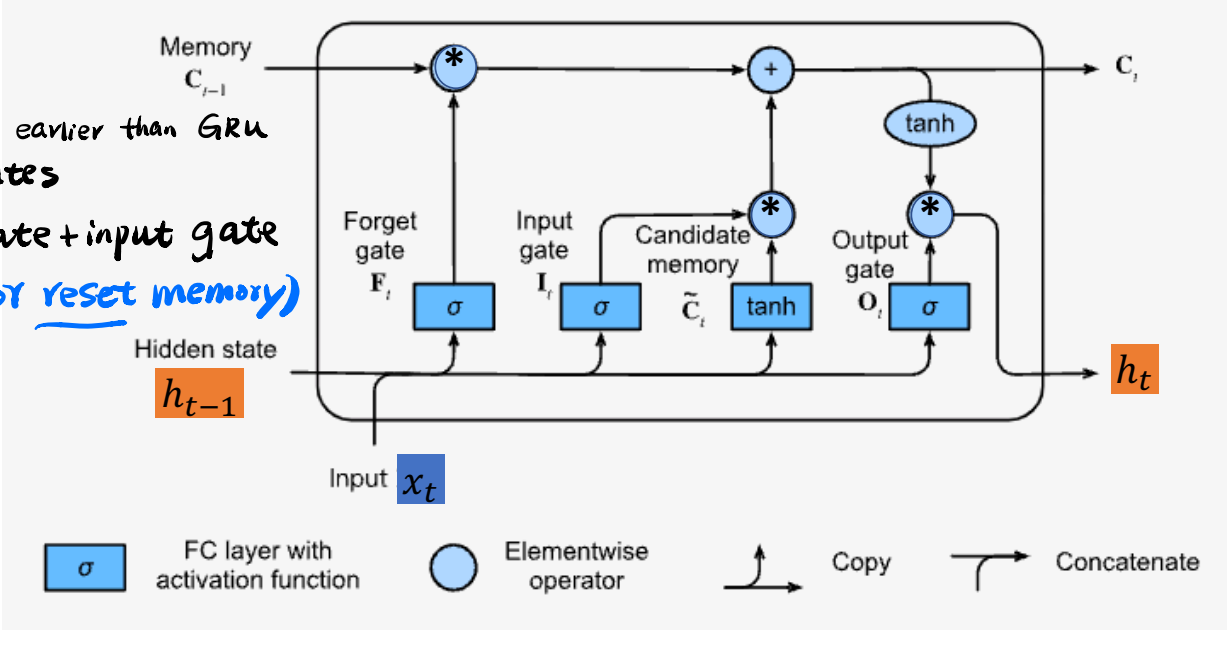
- Reset gates help capture short-term dependencies
- Update gates help capture long-term dependencies

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling: https://arxiv.org/pdf/1412.3555.pdf

Long Short-Term Memory (LSTM)

UC San Diego

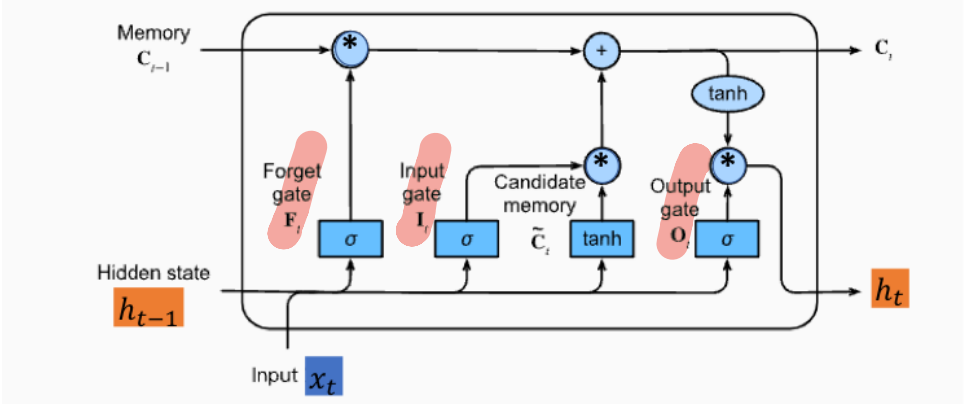
- ~20 years earlier than GRU
- three-gates
- Forget gate + input gate (retain or reset memory)
- Output gate



Long short-term memory, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf

Long Short-Term Memory (LSTM)

UC San Diego



Input Gate: $I_t = \text{sigmoid}(W_{ix} x_t + W_{ih} h_{t-1} + b_i)$

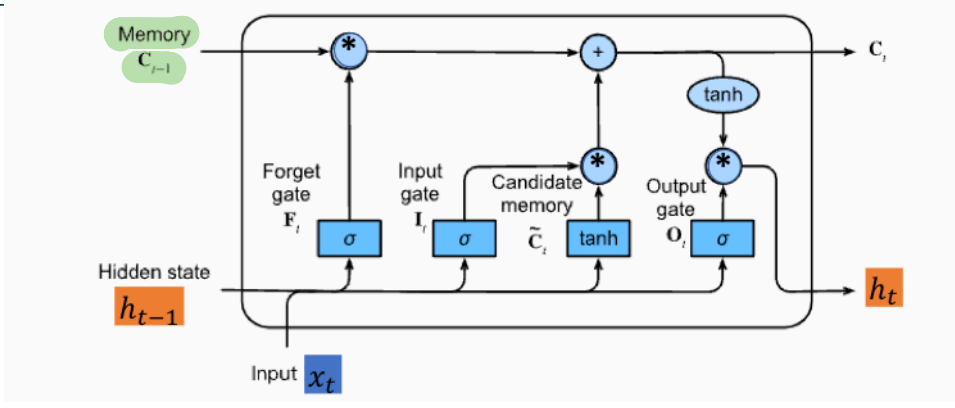
Forget Gate: $F_t = \text{sigmoid}(W_{fx} x_t + W_{fh} h_{t-1} + b_f)$

Output Gate: $O_t = \text{sigmoid}(W_{ox} x_t + W_{oh} h_{t-1} + b_o)$

Long short-term memory, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf

Long Short-Term Memory (LSTM)

UC San Diego



- I_t: how much we take x_t into account in computing $\tilde{C}_t$
- F_t: how much C_{t-1} is used

Proposal/candidate. $\tilde{C}_t = \tanh(W_{cx} x_t + W_{ch} h_{t-1} + b_c)$

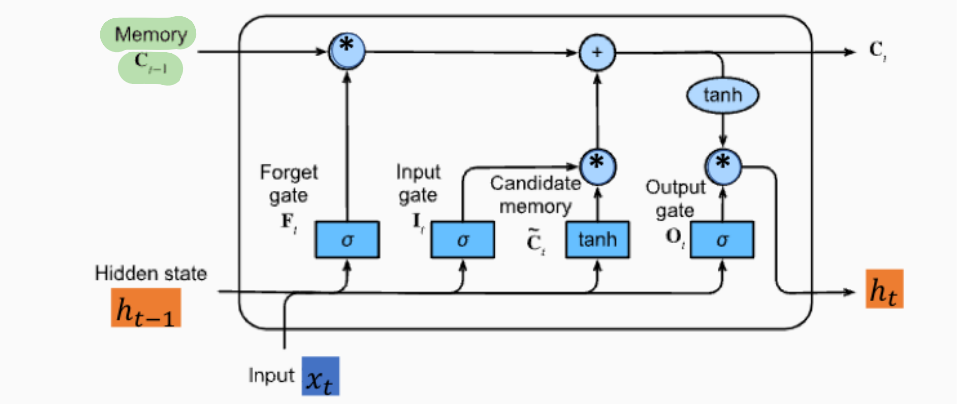
$C_t = F_t * C_{t-1} + I_t * \tilde{C}_t$, * (element-wise multiplication)

Long short-term memory, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf

(GRU: $h_t = z_t * h_{t-1} + (1 - z_t) \tilde{h}_t$)

Long Short-Term Memory (LSTM)

UC San Diego



$h_t = O_t * \tanh(C_t)$

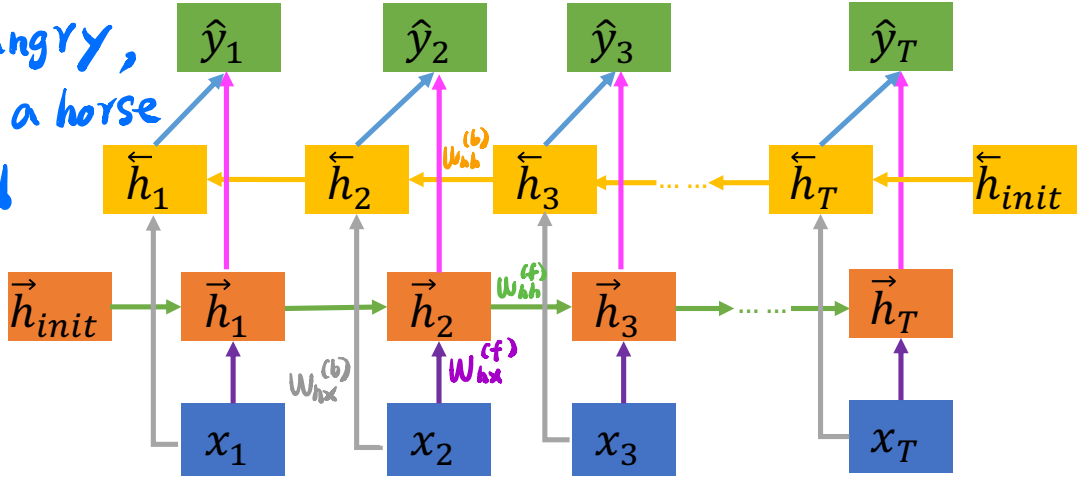
$Y_t = W_y h_t + b_y$ (+ activation func if needed)

Long short-term memory, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf

Bidirectional RNN

UC San Diego

I am _ hungry,
I can eat a horse
I just had dinner



$\vec{h}_t = \sigma(W_{hh}^{(f)} \vec{h}_{t-1} + W_{hx}^{(f)} x_t + b_h^{(f)})$ $\quad x_t \in R^{d \times n}, W_{hx} \in R^{h \times d}, W_{hh} \in R^{h \times h}$

$\overleftarrow{h}_t = \sigma(W_{hh}^{(b)} \overleftarrow{h}_{t+1} + W_{hx}^{(b)} x_t + b_h^{(b)})$ $\quad h_t = [\vec{h}_t, \overleftarrow{h}_t] \in R^{2h \times n}$

$\hat{y}_t = \sigma(W_y h_t + b_y)$, $\quad W_y \in R^{q \times 2h}$, $b_y \in R^{q \times 1}$

Outline

UC San Diego

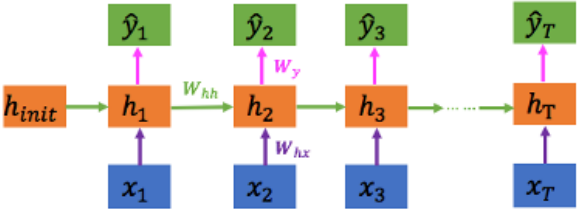
- Recurrent Neural Networks
- Modern Recurrent Neural Networks
- **Attention Mechanism and Transformer**

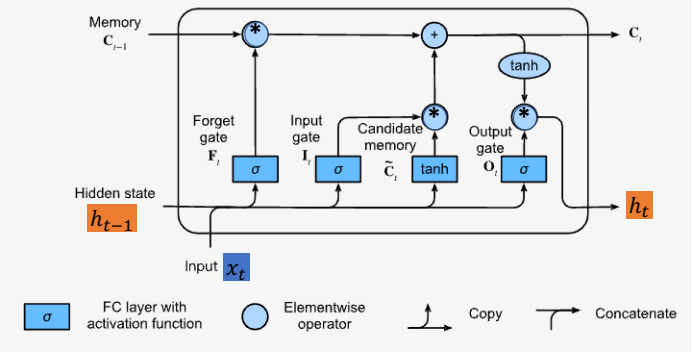
Motivation

UC San Diego

Increased complexity, sequential

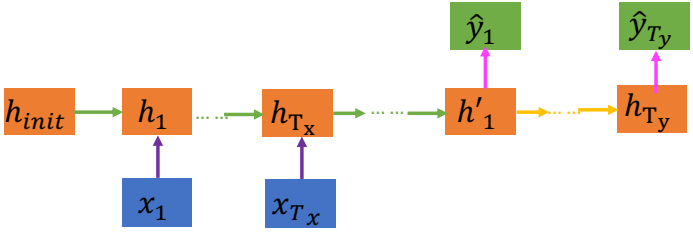
RNN GRU LSTM



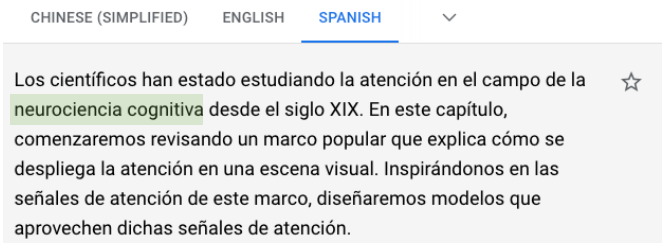


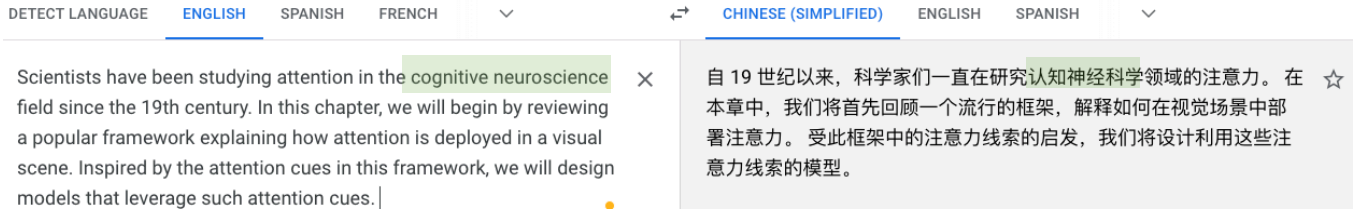
Attention Mechanism

UC San Diego



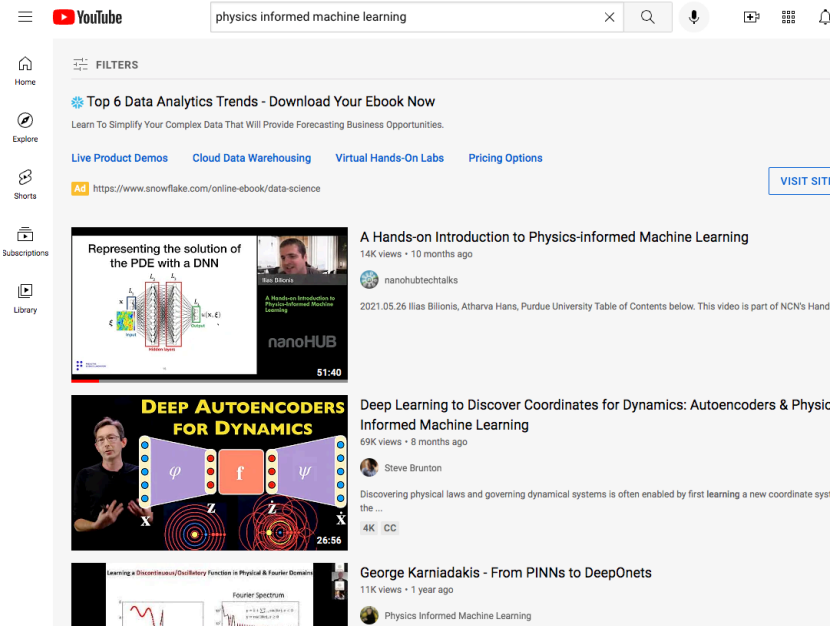
The problem of long sequence in machine translation





Attention Mechanism

UC San Diego



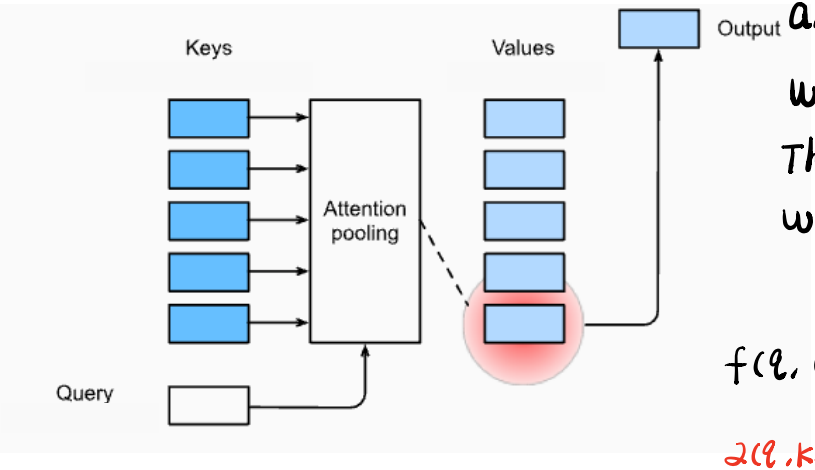
- **Query, Key, Value**

Example: Youtube video retrieval

- Query: "physics informed machine learning"
- Key: each video has a set of keys (video title, description, viewing count…)
- Values: video

Attention Mechanism

UC San Diego



Mathematically, suppose we have a query $q \in R^{d_q}$ and m key-value pairs $(k_1, v_1), \ldots, (k_m, v_m)$ where $k_i \in R^k$ and each value $v_i \in R^v$.

The output of the query can be written as:

Weight: amount of attention q should pay to item (k_i, v_i)

$$f(q, (k_1, v_1), \ldots (k_m, v_m)) = \sum_{i=1}^{m} \alpha(q, k_i) v_i$$

$$\alpha(q, k_i) = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{m} \exp(a(q, k_j))} \in R$$

a is called attention score function

UC San Diego

Different attention score function:

- Query and key are vector of same length: $d_q = d_k$

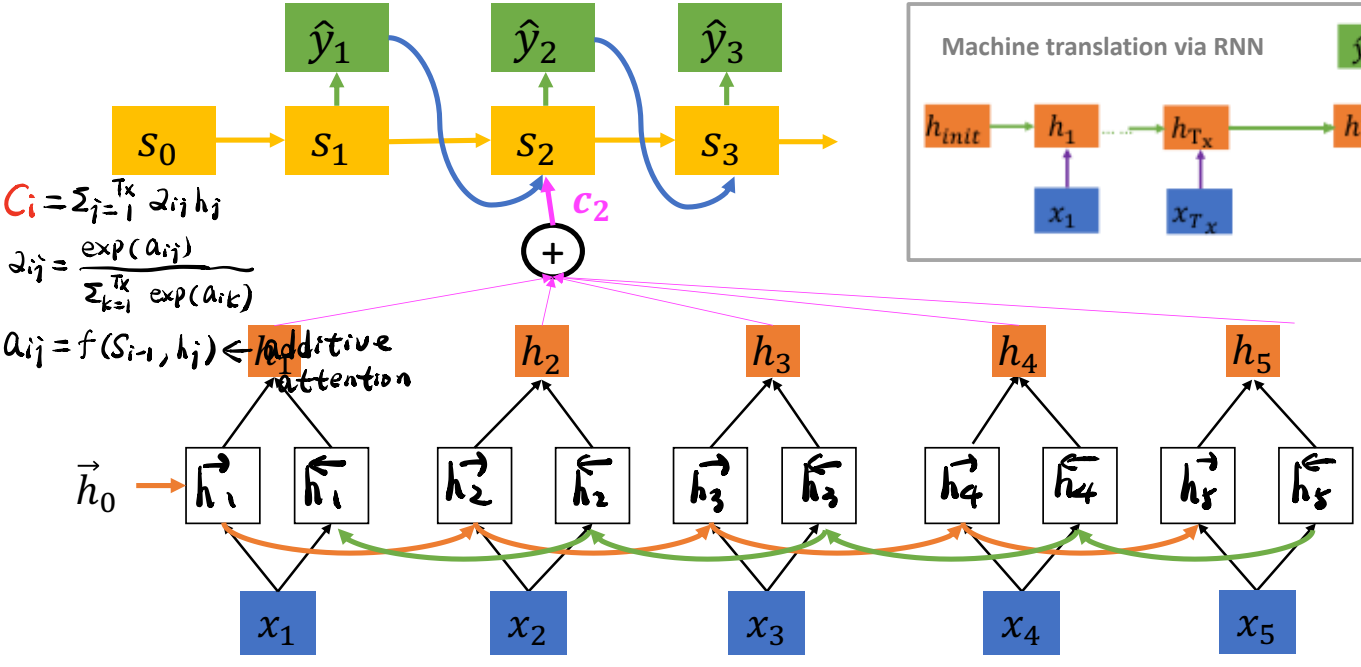
$$a(q, k_i) = \frac{q^T k_i}{\sqrt{d_k}} = \frac{k_i^T q}{\sqrt{d_k}}$$ (Scaled dot-product attention)

- Query and key are vector of different length (additive attention)

$$a(q, k_i) = \underset{R^{d_a}}{W_a^T} \tanh\left(\underset{R^{d_a \times d_q}}{W_q} q + \underset{R^{d_a \times d_k}}{W_k} k_i\right) \in R$$

RNN with Attention Mechanism

UC San Diego



Neural Machine Translation by Jointly Learning to Align and Translate, https://arxiv.org/abs/1409.0473

Transformer

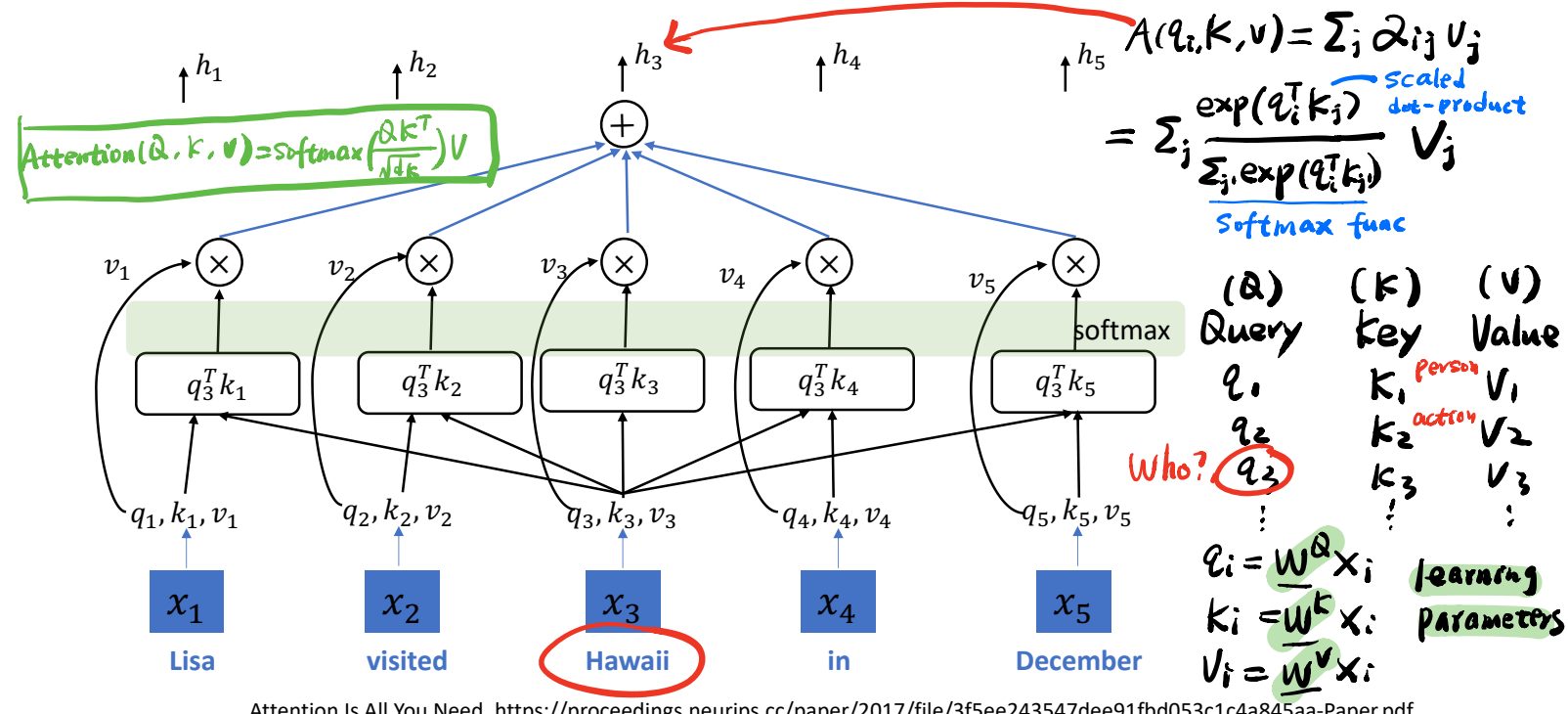
UC San Diego

- Self Attention
- Multi-head Attention

Attention Is All You Need, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer: Self Attention

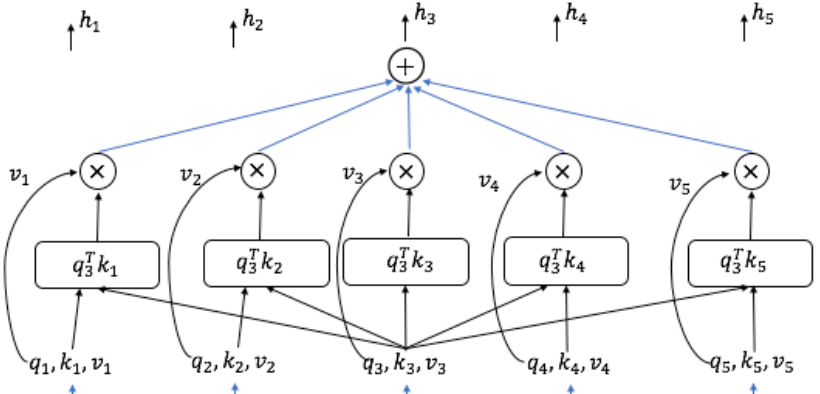
UC San Diego

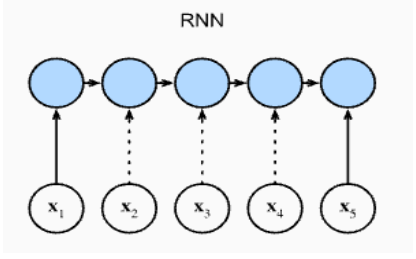


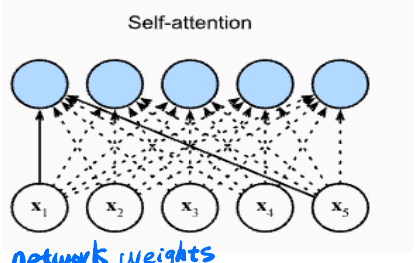
Attention Is All You Need, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer: Self Attention vs RNN

UC San Diego







network weights depend on input

Attention Is All You Need, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer: Multi-Head Attention

UC San Diego

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\begin{bmatrix} head^{(1)} \\ head^{(2)} \\ \vdots \\ head^{(N)} \end{bmatrix} W_o$$

N: # of head

h_3

$head^{(1)} = \text{Attention}(W_1^Q Q, W_1^K K, W_1^V V)$

$W_1^Q q_1, W_1^K k_1, W_1^V v_1$ | $W_1^Q q_2, W_1^K k_2, W_1^V v_2$ | $W_1^Q q_3, W_1^K k_3, W_1^V v_3$ | $W_1^Q q_4, W_1^K k_4, W_1^V v_4$ | $W_1^Q q_5, W_1^K k_5, W_1^V v_5$

q_1, k_1, v_1 | q_2, k_2, v_2 | q_3, k_3, v_3 | q_4, k_4, v_4 | q_5, k_5, v_5

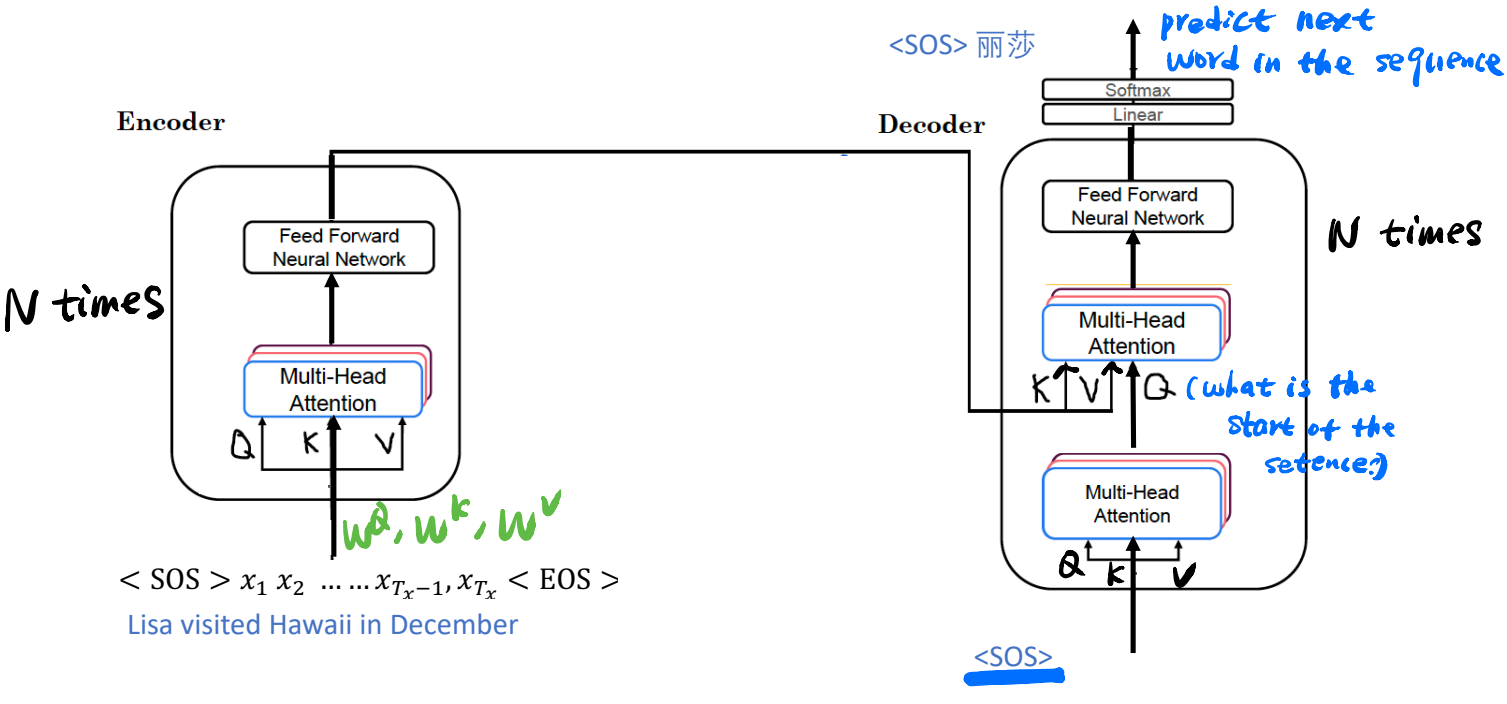
$q_i = W^Q x_i$, $k_i = W^K x_i$, $v_i = W^V x_i$ (learning parameters)

x_1	x_2	x_3	x_4	x_5
Lisa	visited	Hawaii	in	December

Attention Is All You Need, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer

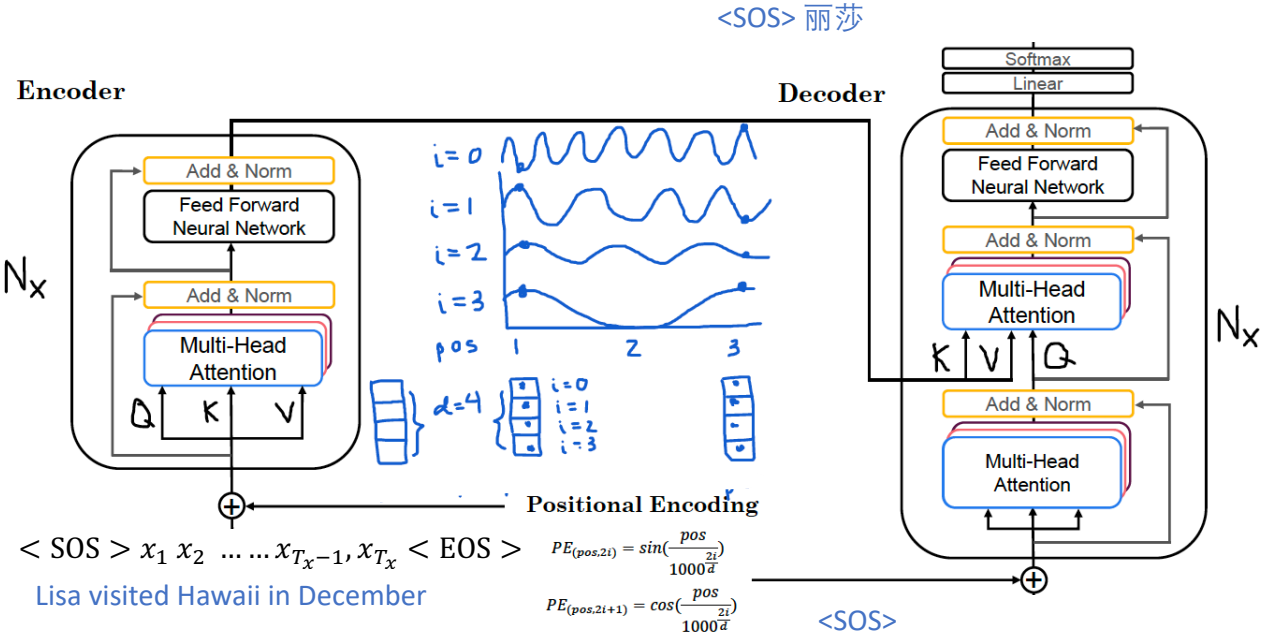
UC San Diego



Attention Is All You Need, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer Details

UC San Diego



Attention Is All You Need, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer vs RNN

UC San Diego

Transformer

- ☆ Convinient parellel computation
- ☆ Can capture (very) long range dependency & not suffer from vanishing gradient issue
- ☆ work well when you have BIG data (e.g. NLP tasks)

RNN / LSTM / GRU

- ☆ better capture temporal sequence structure
- ☆ RNN/LSTM/GRU can still work very well (or even better) in common tasks with moderate # of data and not hyper-long sequences