Recurrent Neural Network(RNN),NLP

HESAM HOSSEINI

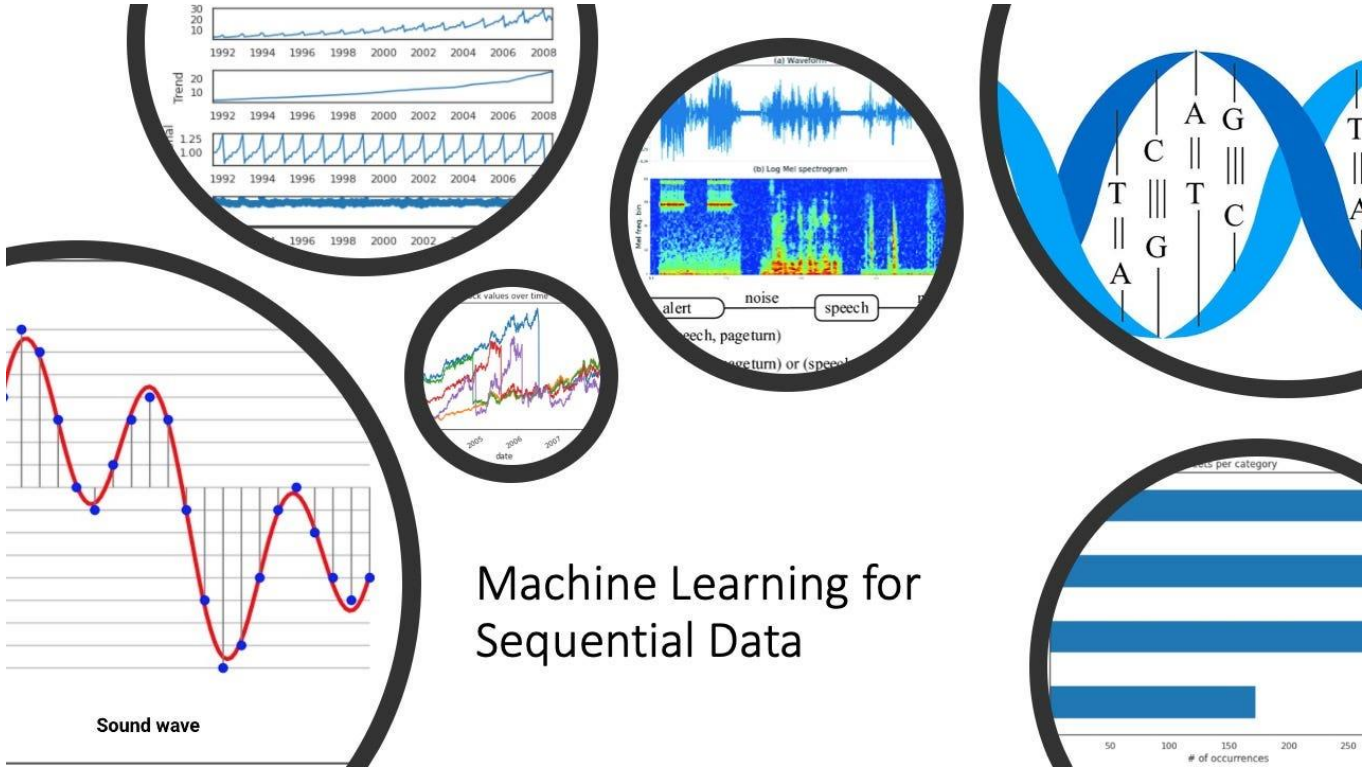
SUMMER 2024

Sequential data analysis

Till now: Static mapping from input to output

RNN: Temporal/Dynamic Systems

- Feedback Connection



Application

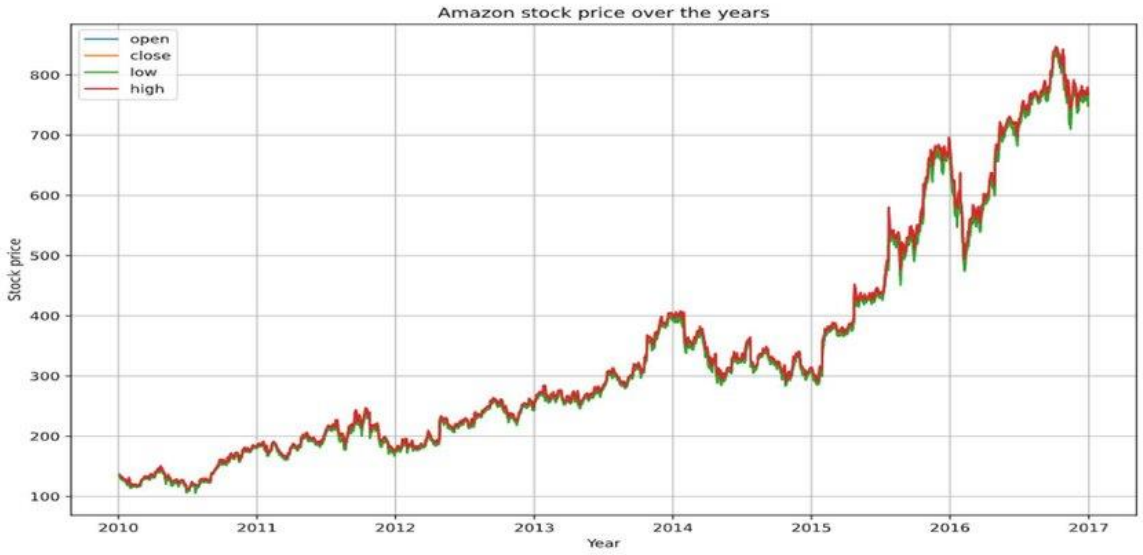
Time series prediction/forecasting

Natural Language Processing

Speech/Signal Processing

Question/Answering Machine

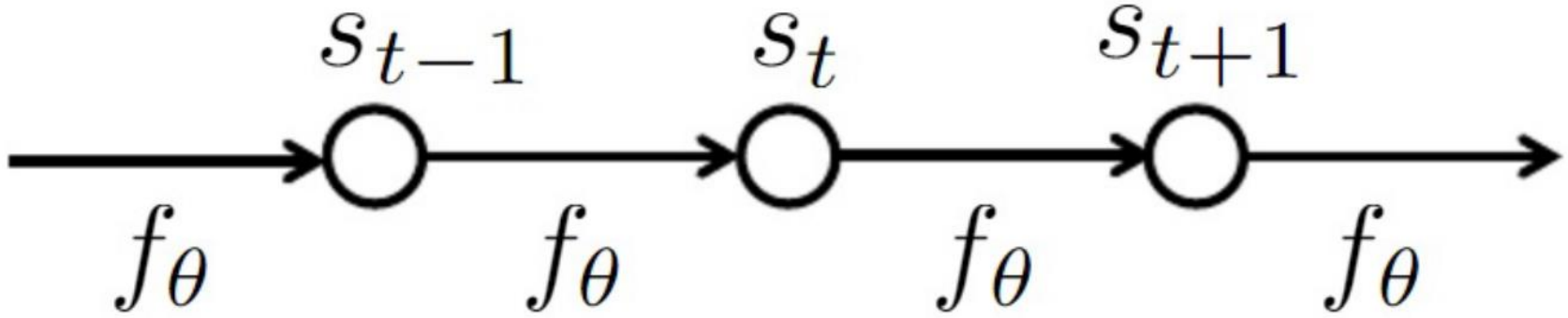
Image Captioning



Dynamic systems

The classical form of a dynamical system:

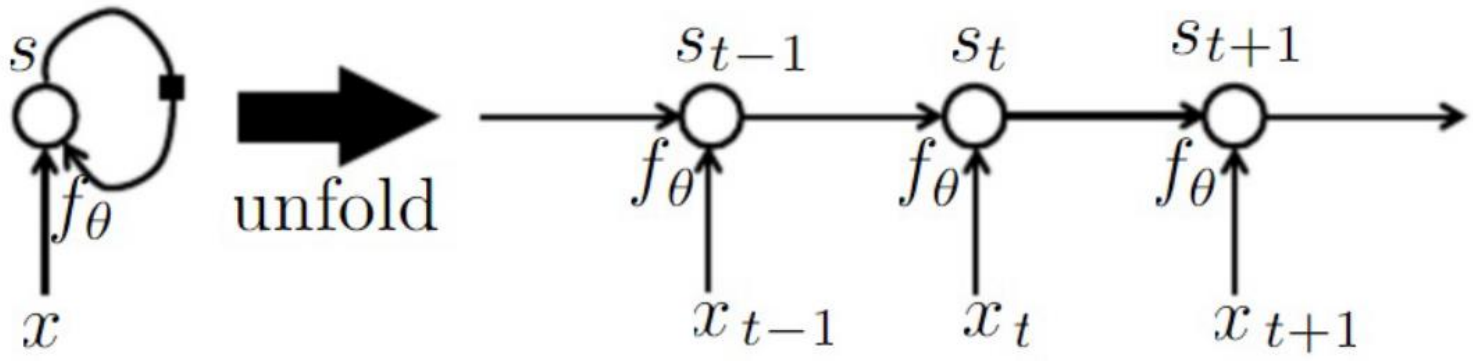
$$s_t = f_\theta(s_{t-1})$$



Dynamic systems

Now consider a dynamic system with an external signal x

$$s_t = f_\theta(s_{t-1}, x_t)$$



The state contains information about the whole past sequence.

$$s_t = g_t(x_t, x_{t-1}, x_{t-s}, \ldots, x_2, x_1)$$

Recurrent Neural Network (RNN)

Base equations

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

h_t is state of network and summarizes information about the entire past

Model directly embeds the memory in the state

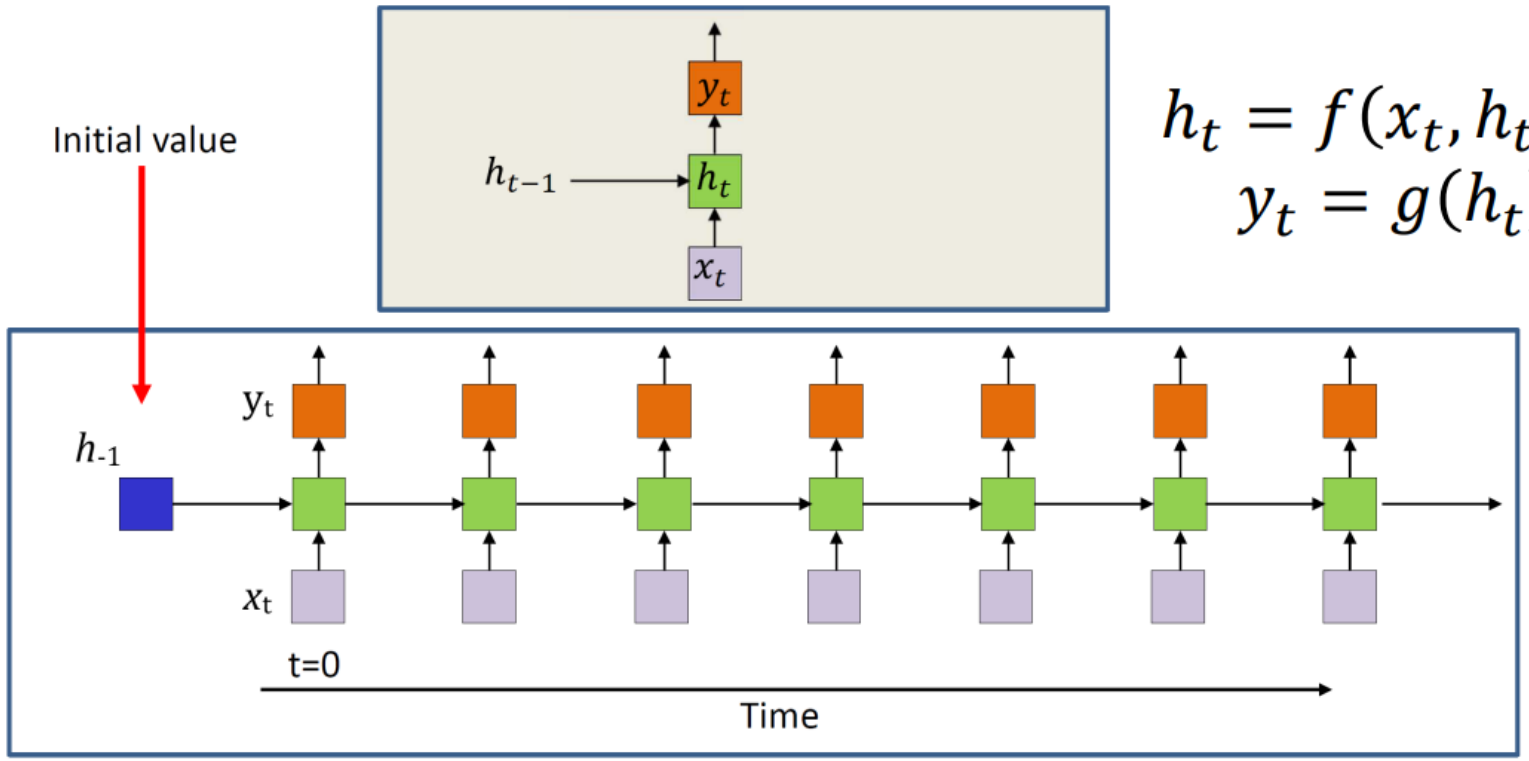
Need to define initial state h_{-1}

This is a fully recurrent neural network or simply a recurrent neural network (RNN)

An input at t = 0 affects outputs forever

Simple RNN

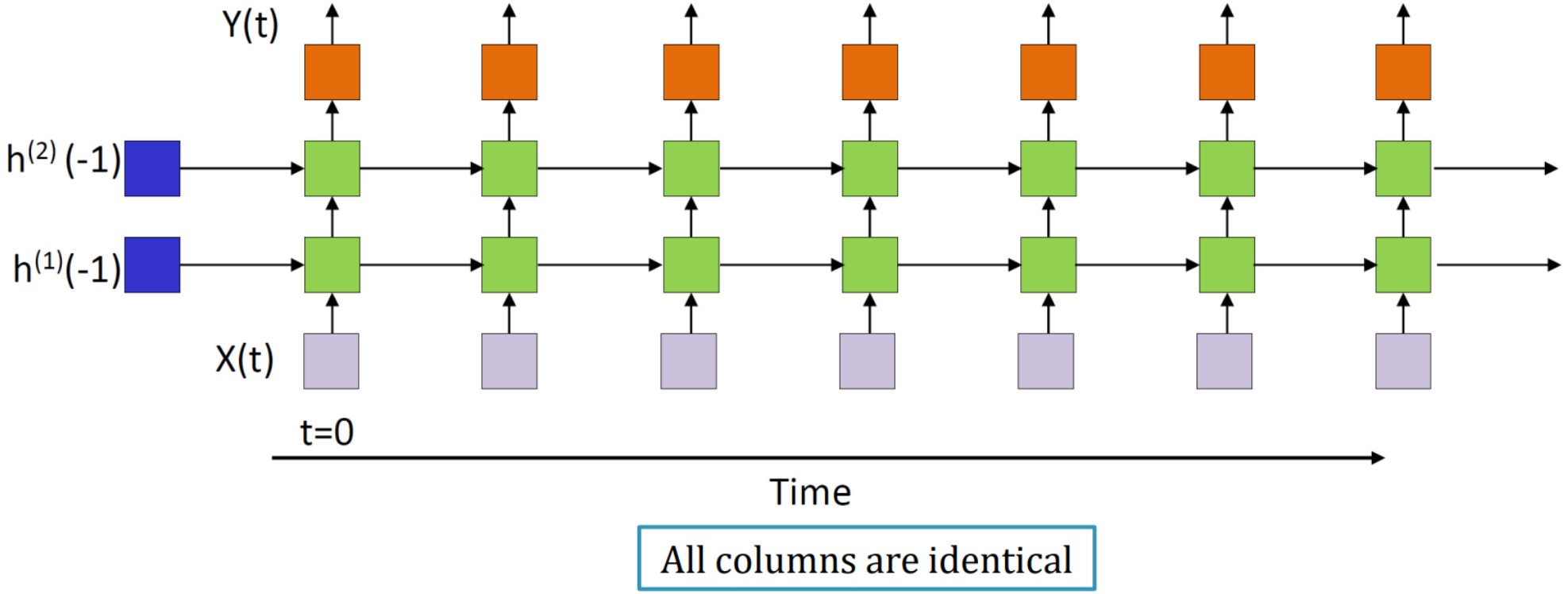
Signal and Computation Flow



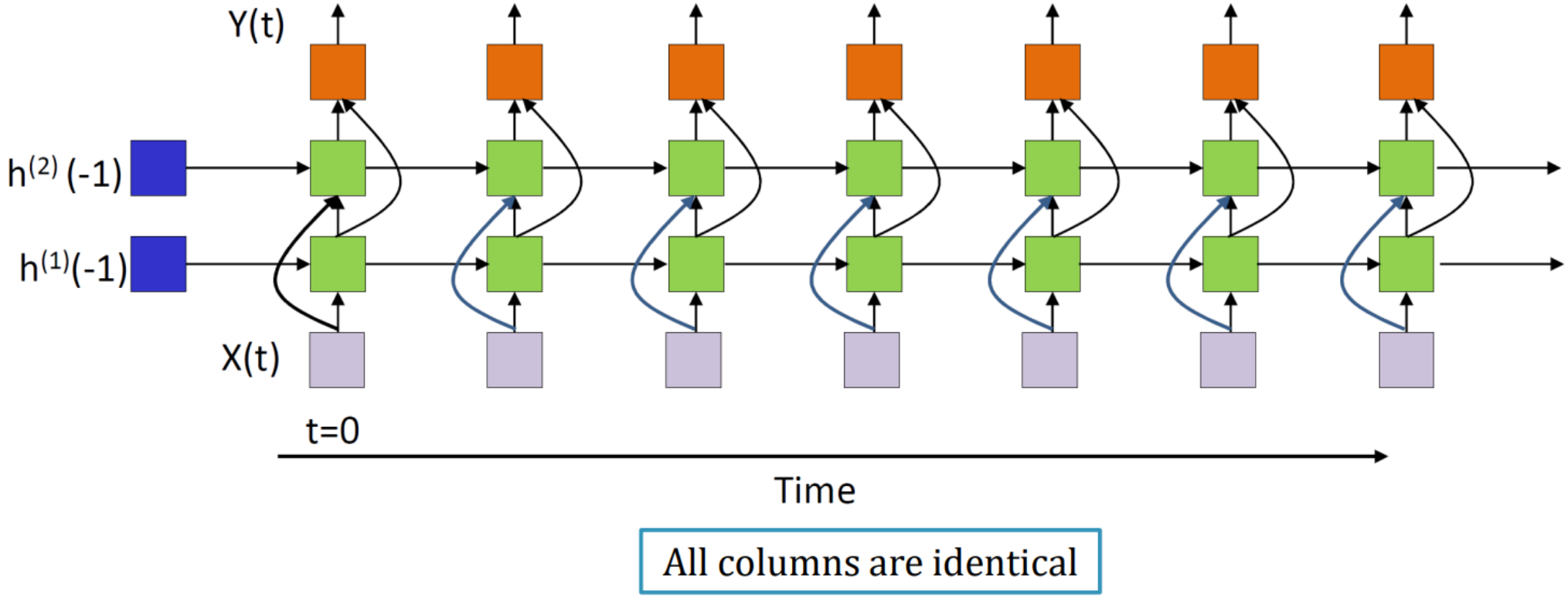
$$h_t = f(x_t, h_{t-1})$$
$$y_t = g(h_t)$$

All columns are identical

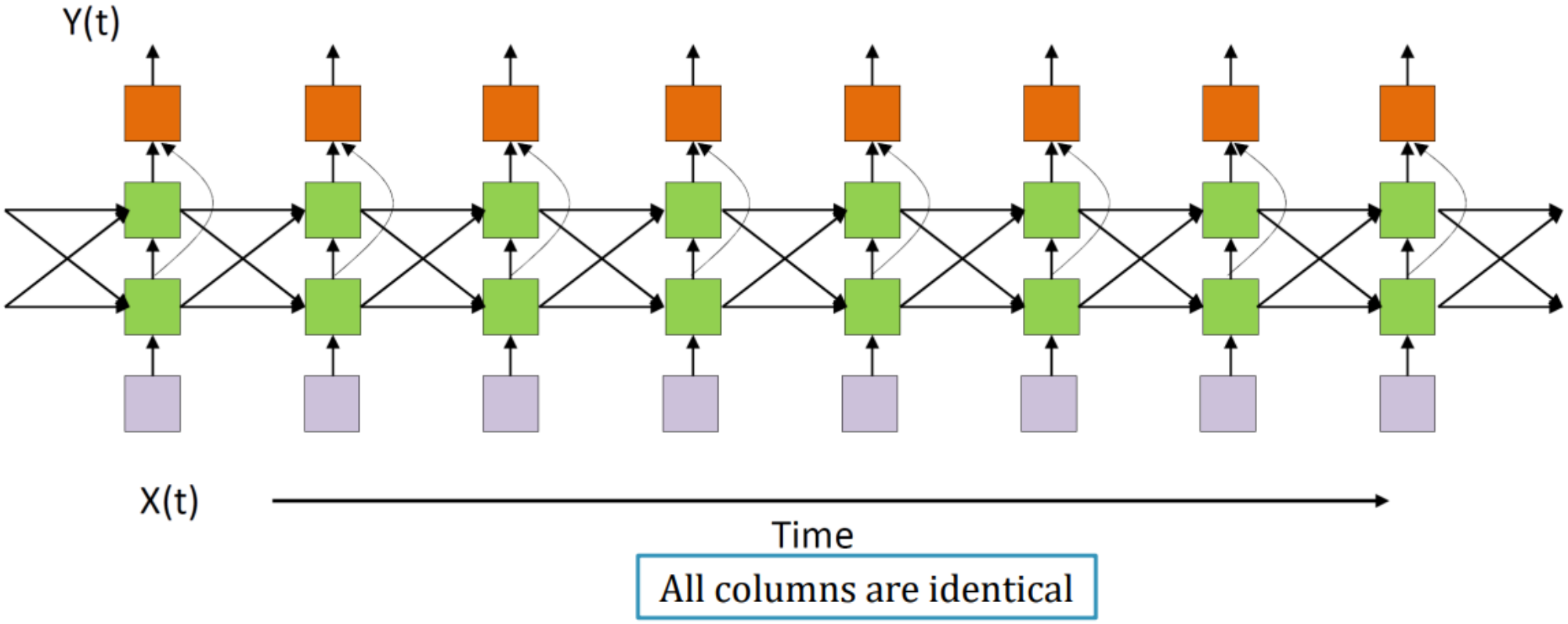
Multiple Recurrent Layer RNN



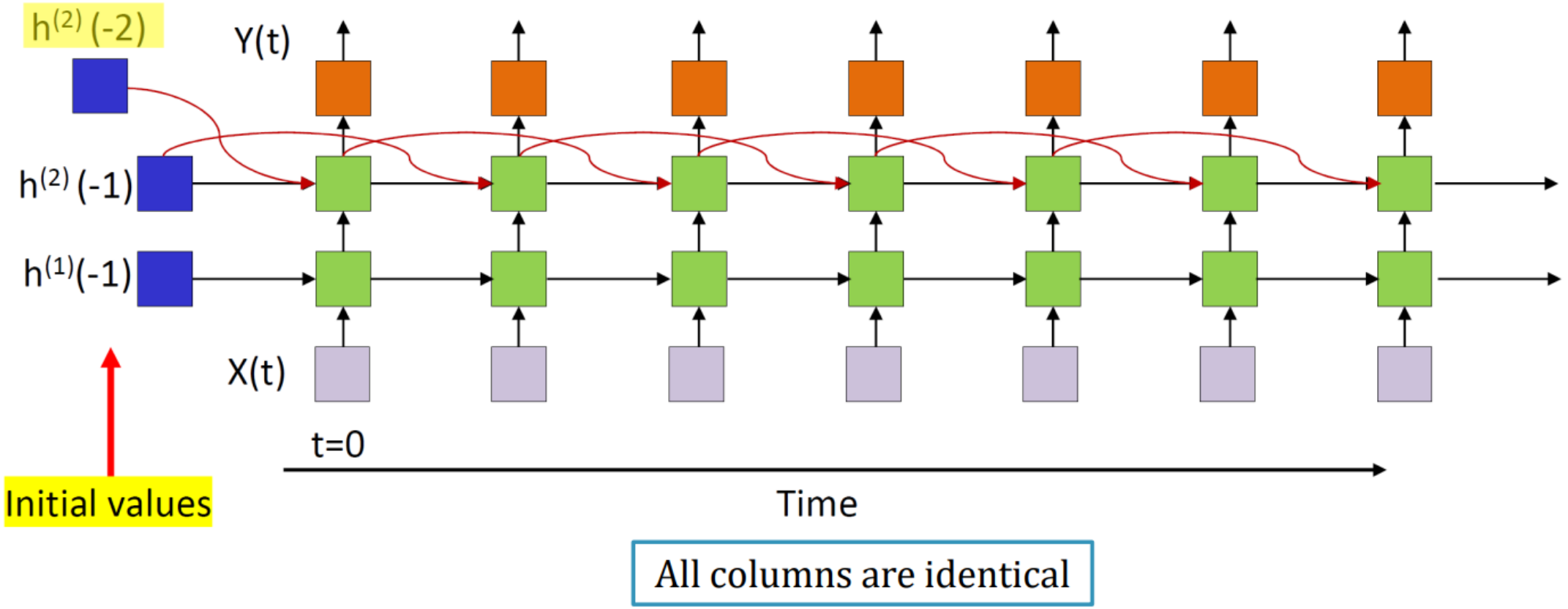
More #1 Complex RNN (Skip Connection)



More #2 Complex RNN



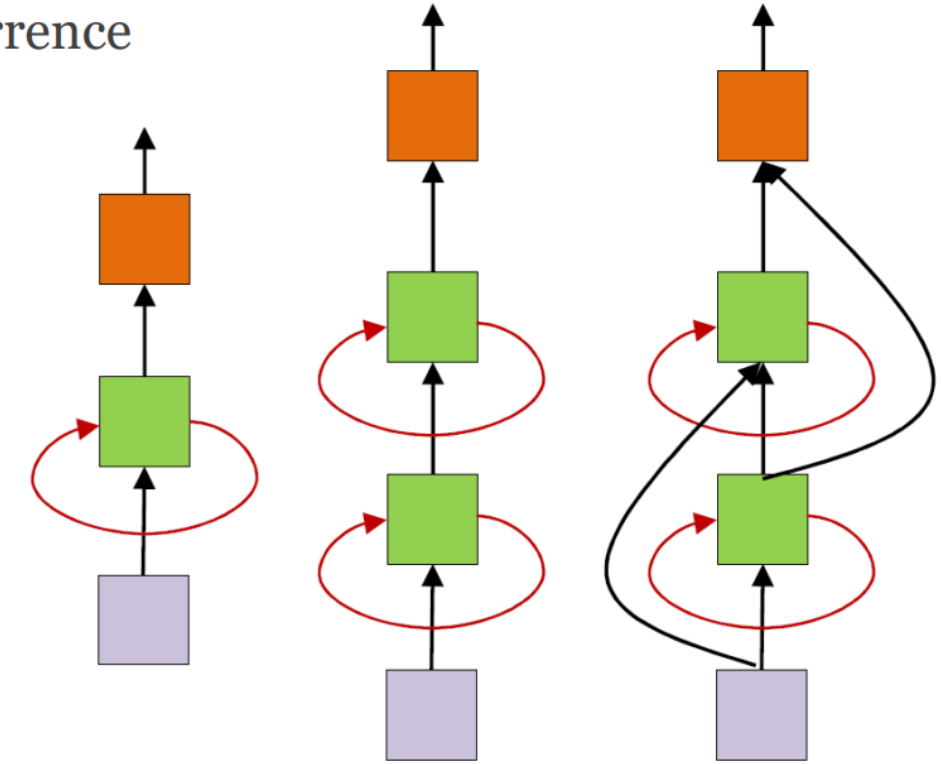
More #3 Complex RNN



Simplified Illustration

Simplified models often drawn

The loops imply recurrence

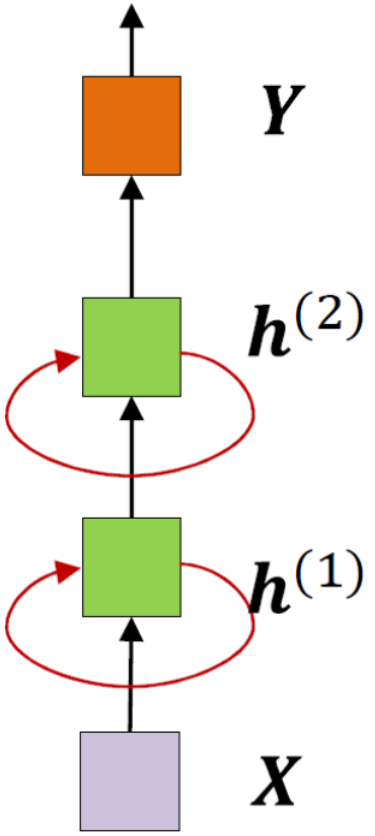


RNN Equations

$$\boldsymbol{h}^{(1)}(t) = f_1\left(\boldsymbol{W}^{(1)}\boldsymbol{X}(t) + \boldsymbol{W}^{(11)}\boldsymbol{h}^{(1)}(t-1) + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{h}^{(2)}(t) = f_2\left(\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)}(t) + \boldsymbol{W}^{(22)}\boldsymbol{h}^{(2)}(t-1) + \boldsymbol{b}^{(2)}\right)$$

$$\boldsymbol{Y}(t) = f_3\left(\boldsymbol{W}^{(3)}\boldsymbol{h}^{(2)}(t) + \boldsymbol{b}^{(3)}\right)$$



Superscript indicates layer of network from which inputs are obtained

Assuming vector function at output, e.g. softmax.

The state node activation, $f1\cdot$ and $f2\cdot$ is typically tanh(·)

RNN variants

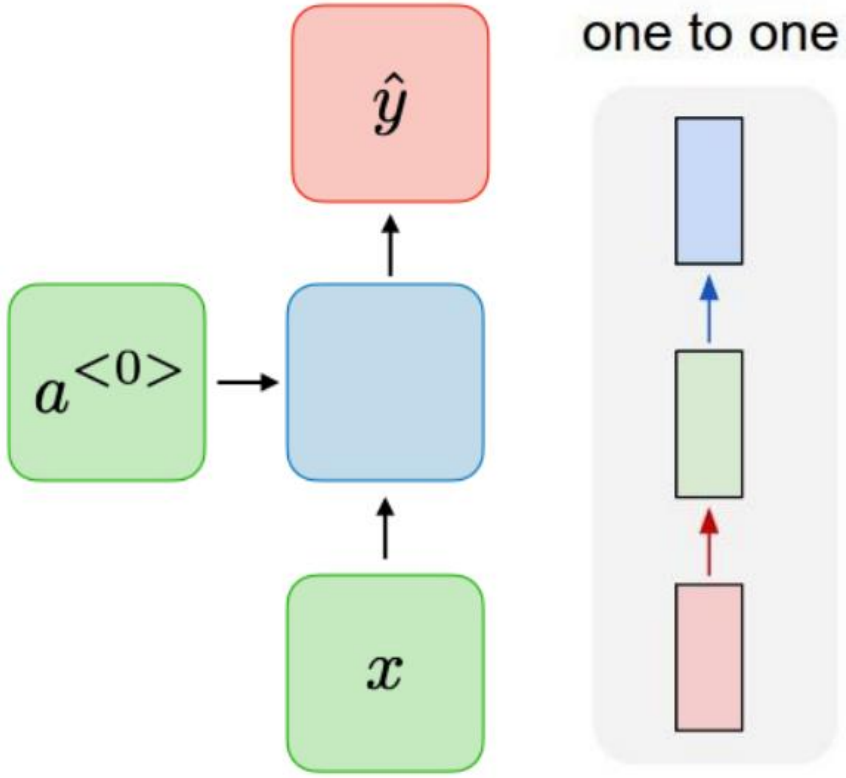
Type of RNNs based on number of input(s)-output(2):

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many

Type of RNNs based on signal flow:

- Deep RNN (DRNN)
- Bidirectional (BRNN)

One-to-One RNN



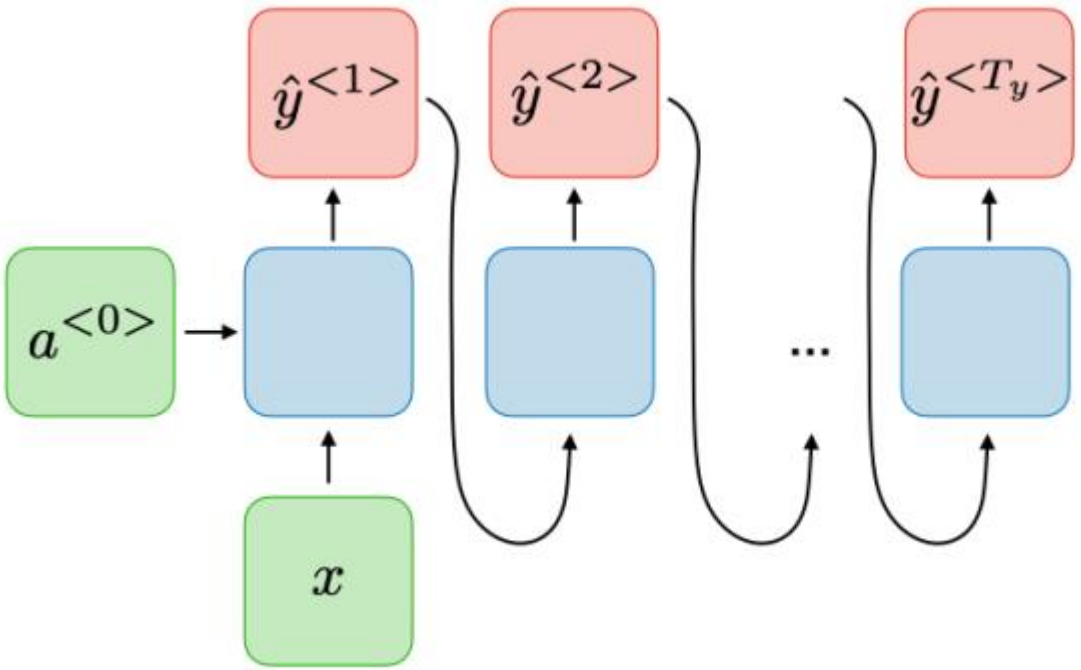
One-to-Many RNN

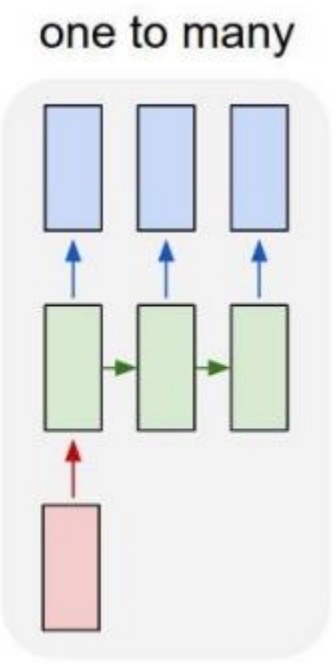
Sequence generation

$Tx = 1, Ty > 1$

Applications:

- Image Captioning
- Music generation





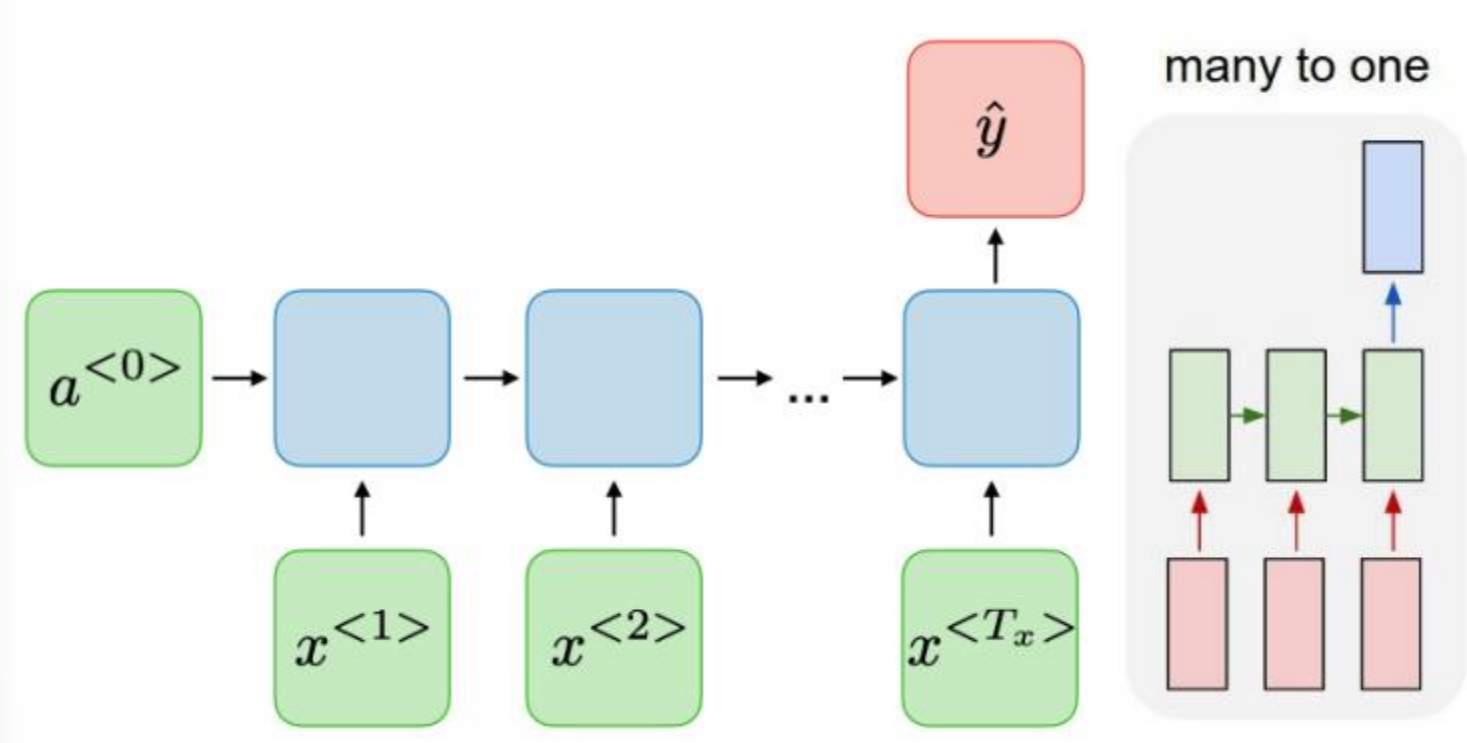
Many-to-One RNN

Sequence based prediction or classification

$Tx > 1, Ty = 1$

Applications:

- speech Recognition
- Action Prediction
- Sentiment Classification
- Emotion Classification



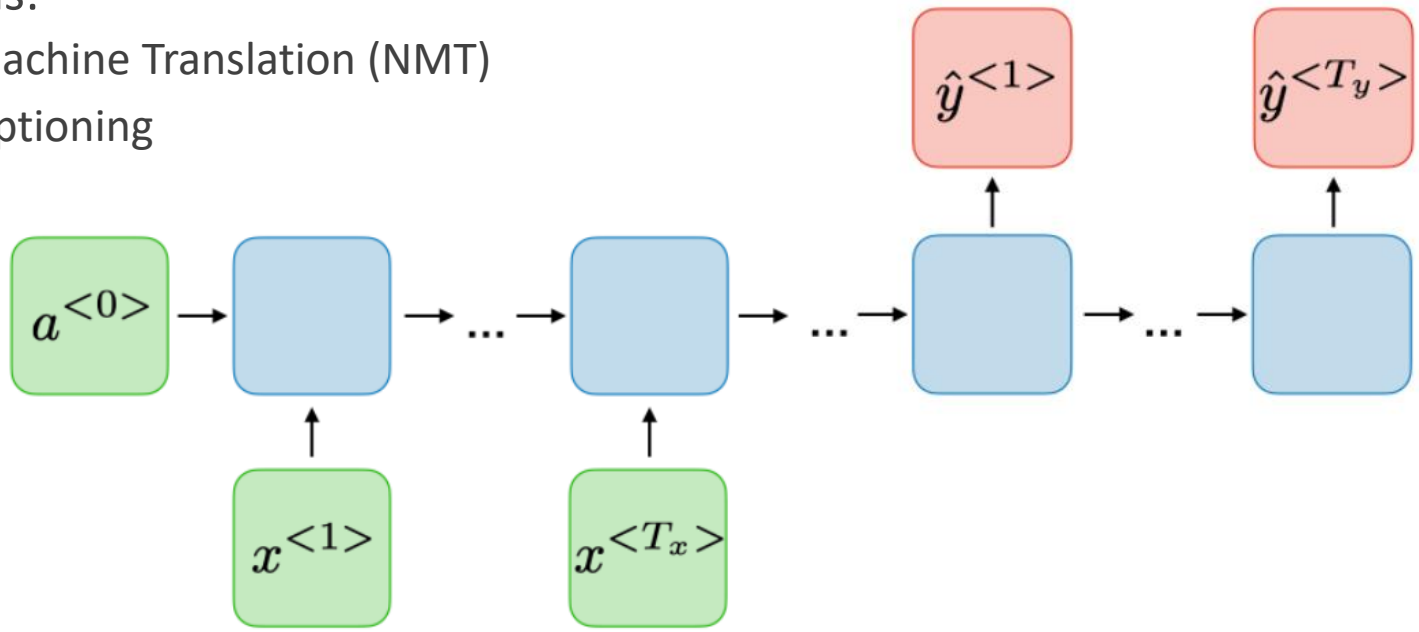
Many-to-Many RNN

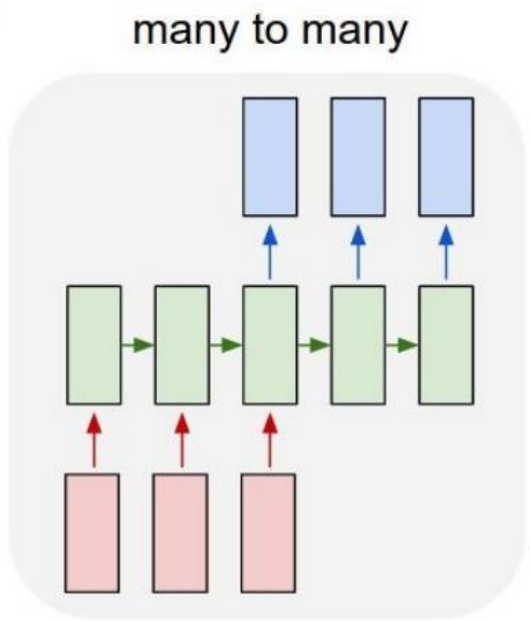
Delayed sequence to sequence

$Tx \neq Ty > 1$

Applications:

- Neural Machine Translation (NMT)
- Video Captioning





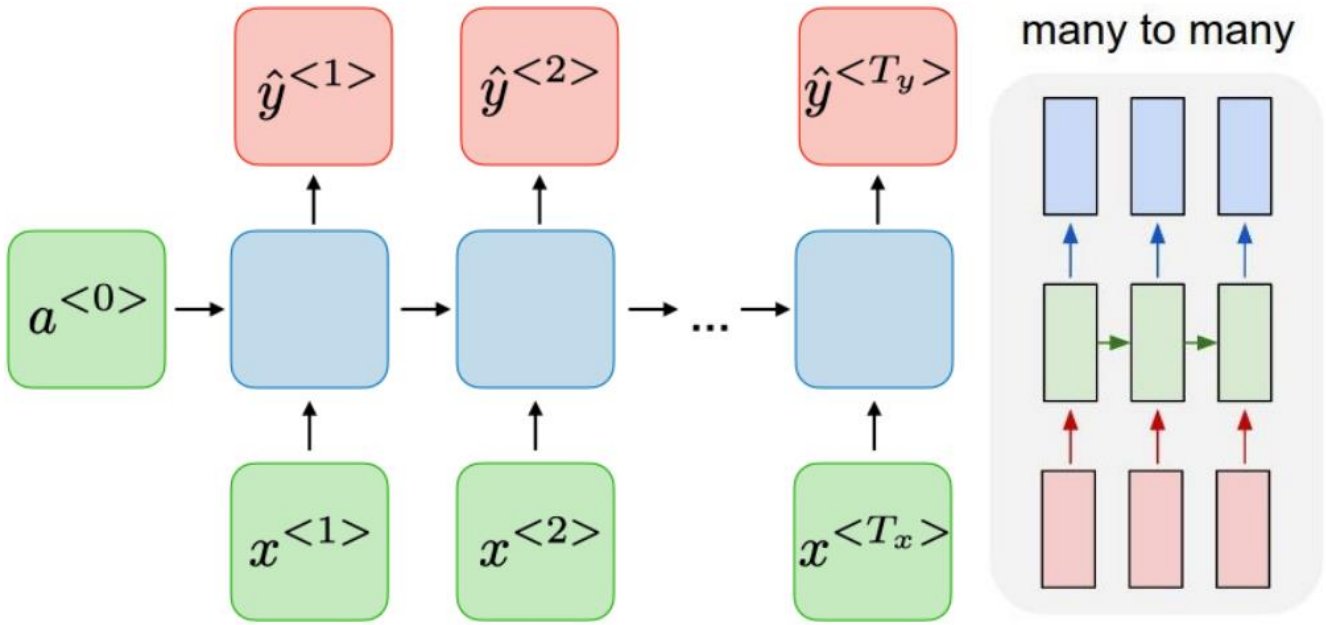
Many-to-Many RNN

Sequence to sequence

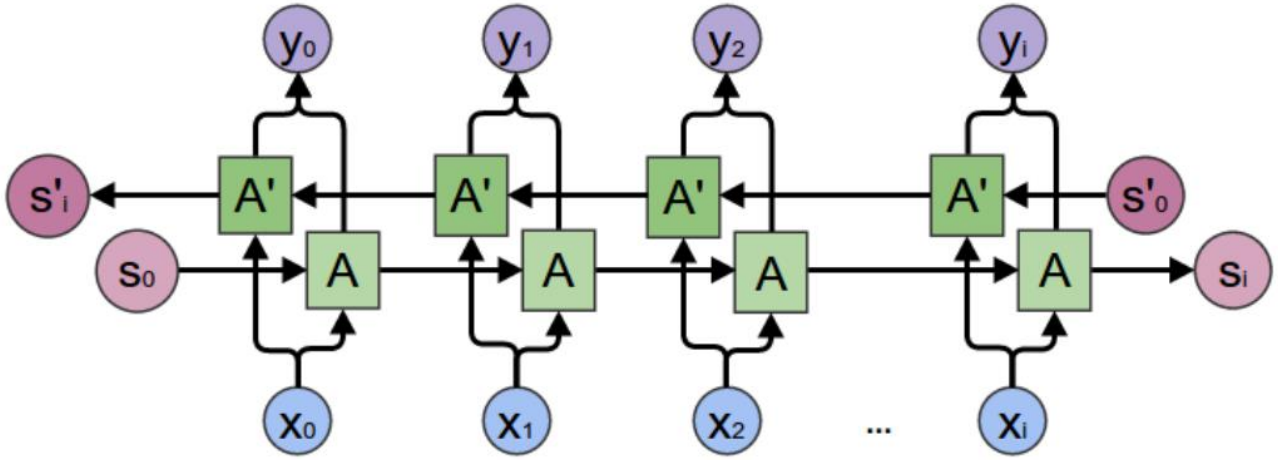
$Tx = Ty > 1$

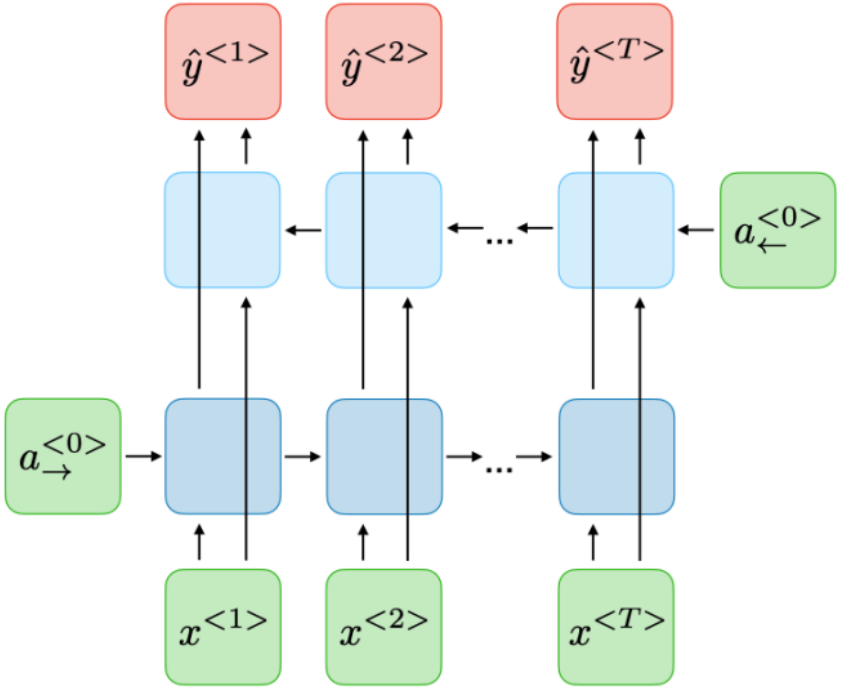
Applications:

- Stock problem
- Label prediction
- Video classification on frame level
- Named entity recognition



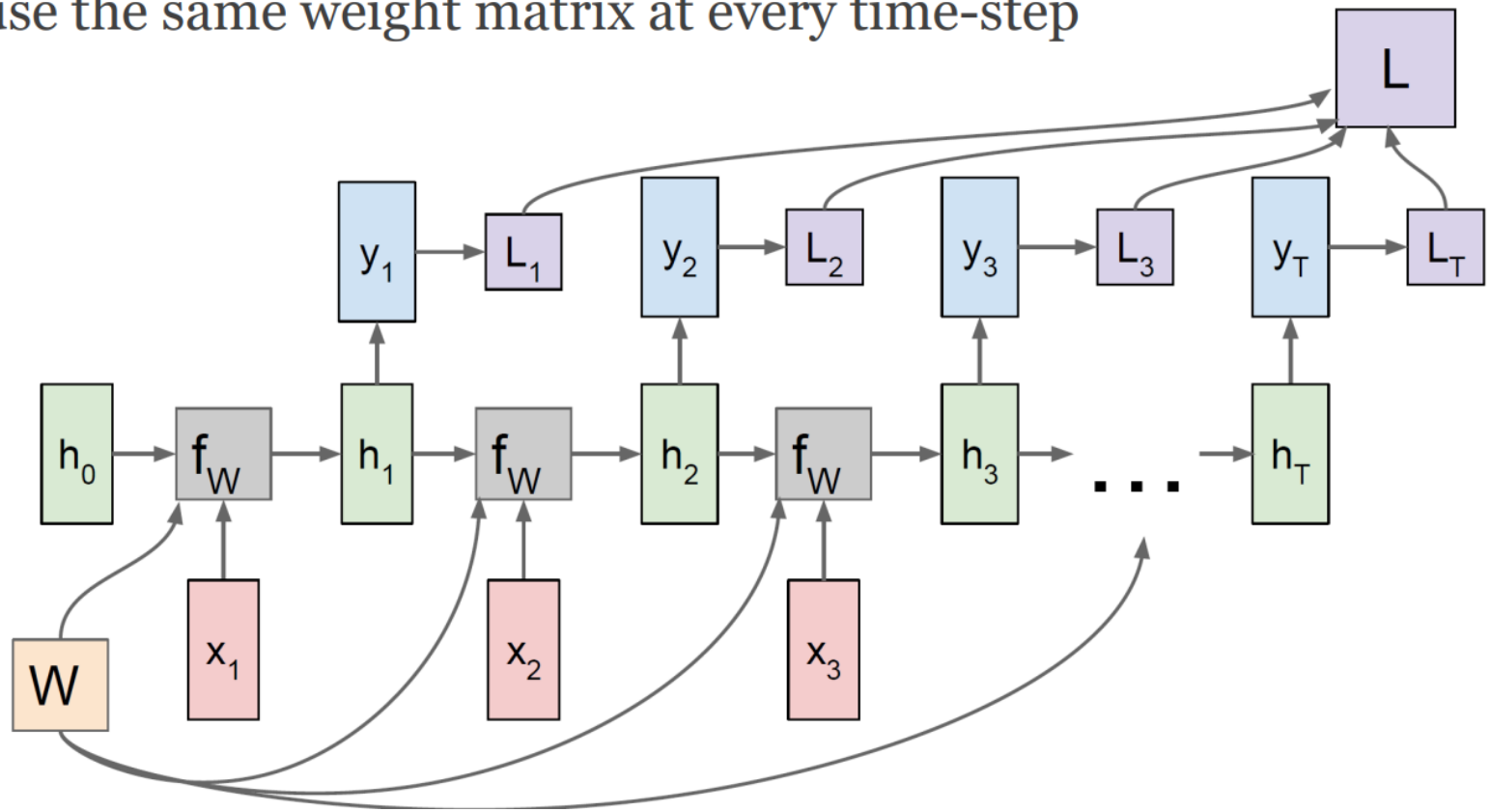
Bidirectional RNN





Computational Graph

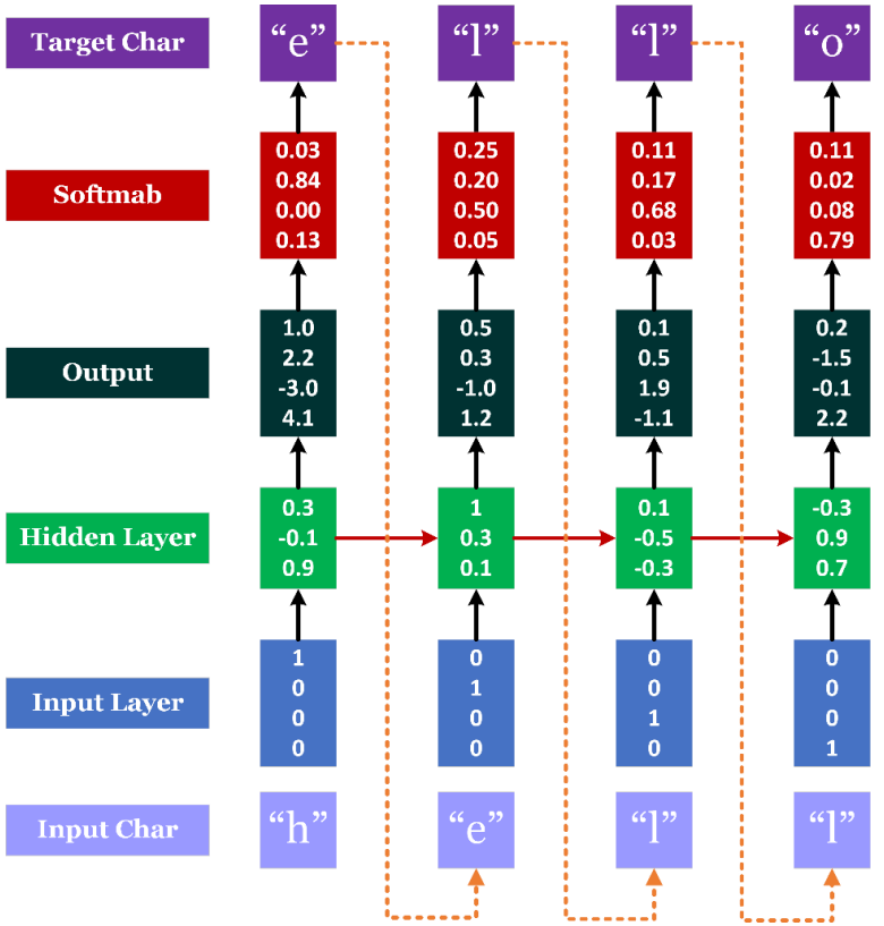
Re-use the same weight matrix at every time-step



Seq2Seg Example

Character-Level Language Models:

Vocabulary: [h,e,l,o]



RNN – Loss

Cumulative loss and update rules:

$$L\left(\{\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(T)}\},\{\boldsymbol{y}^{(1)},\dots,\boldsymbol{y}^{(T)}\}\right)=\sum_{t}L^{(t)}=-\sum_{t}logp_{model}\left(\boldsymbol{y}^{(t)}\middle|\{\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(t)}\}\right)$$

For t = 1 to t = T, we apply the following update equation:

$$\boldsymbol{a}^{(t)}=\boldsymbol{b}+\boldsymbol{W}\boldsymbol{h}^{(t-1)}+\boldsymbol{U}\boldsymbol{x}^{(t)}$$

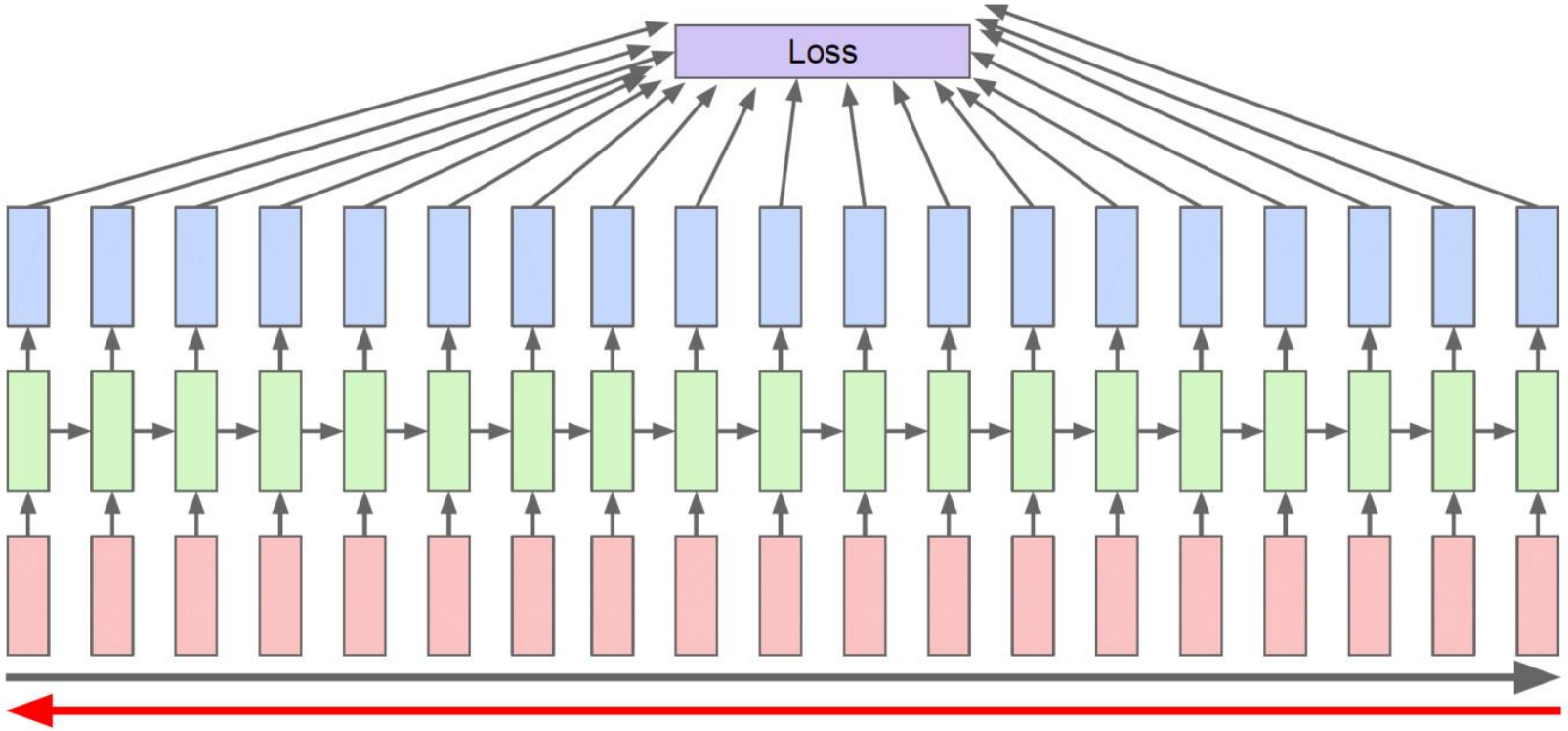
$$\boldsymbol{h}^{(t)}=tanh\left(\boldsymbol{a}^{(t)}\right)$$

$$\boldsymbol{o}^{(t)}=\boldsymbol{c}+\boldsymbol{V}\boldsymbol{h}^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)}=softmax\left(\boldsymbol{o}^{(t)}\right)$$

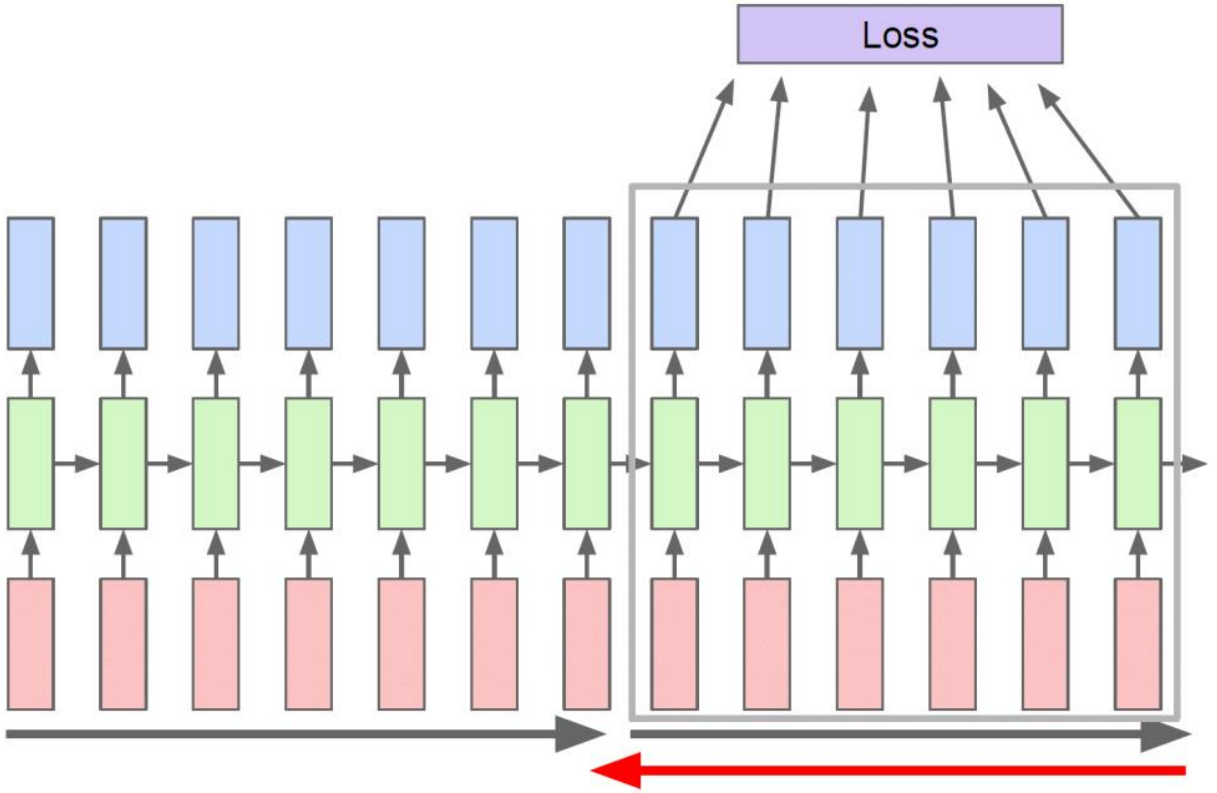
BPTT (Back Propagation Through Time)

BTPP: Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



Truncated Backpropagation Through Time

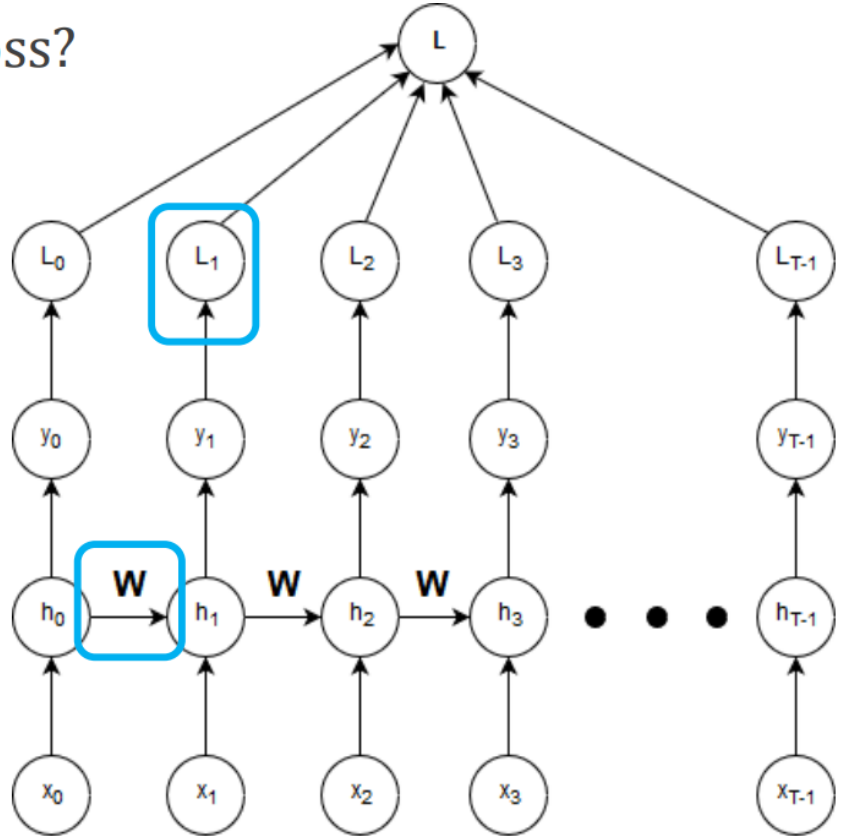
T-BPTT : Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps



BPTT

How Many path from weight(s) to temporal Loss?

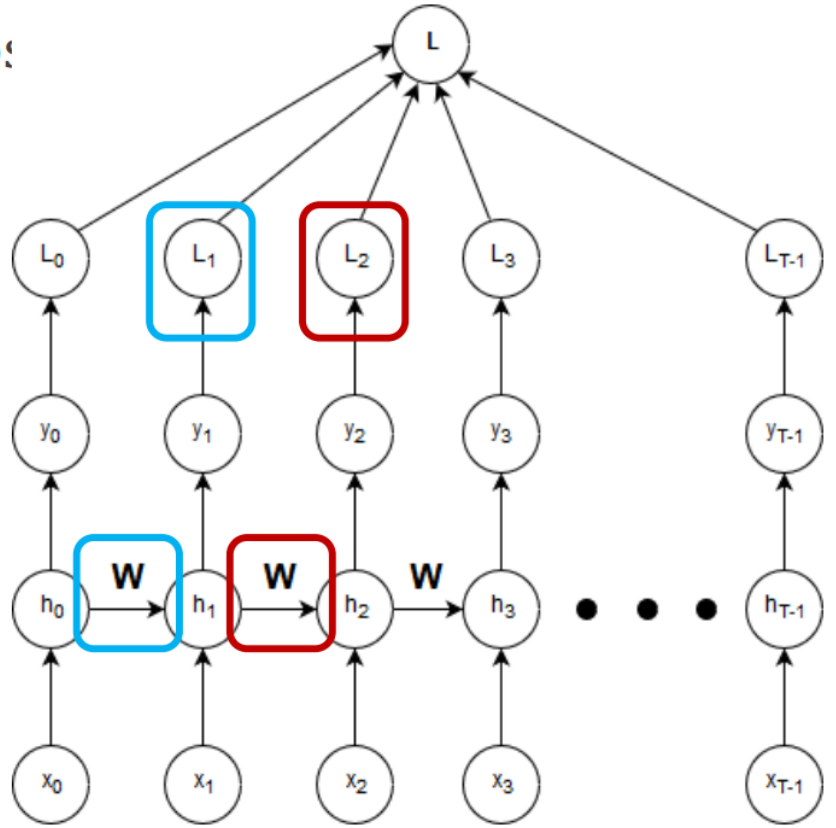
- Path #1 originate at h_0



BPTT

How Many path from weight(s) to temporal Los

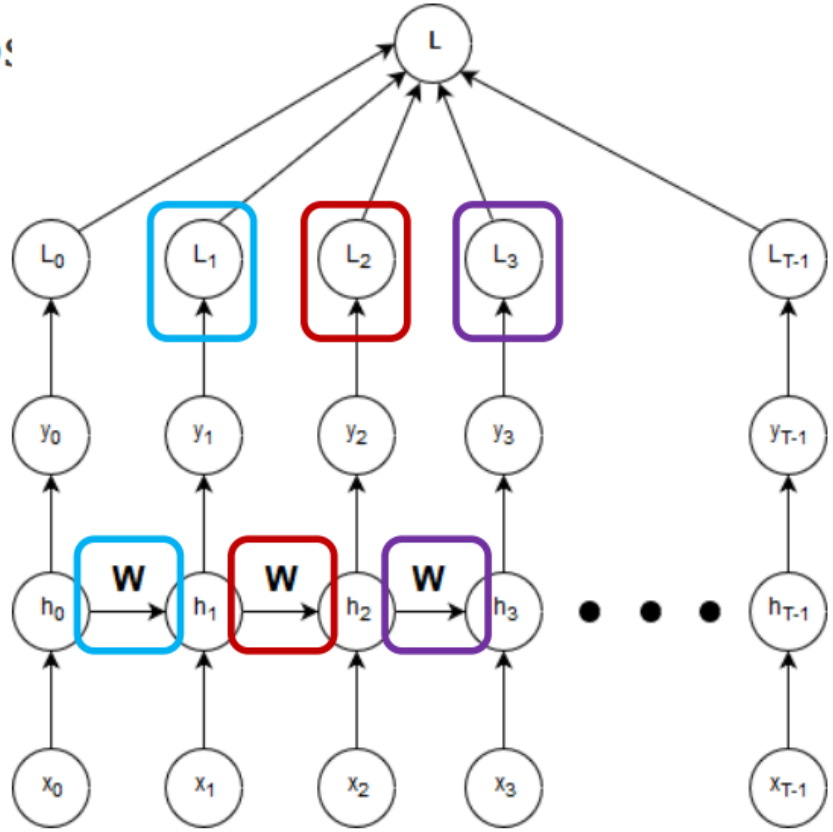
- Path #2 originate at h_0 and h_1



BPTT

How Many path from weight(s) to temporal Lo:

- Path #3 originate at h_0 , h_1 and h_2

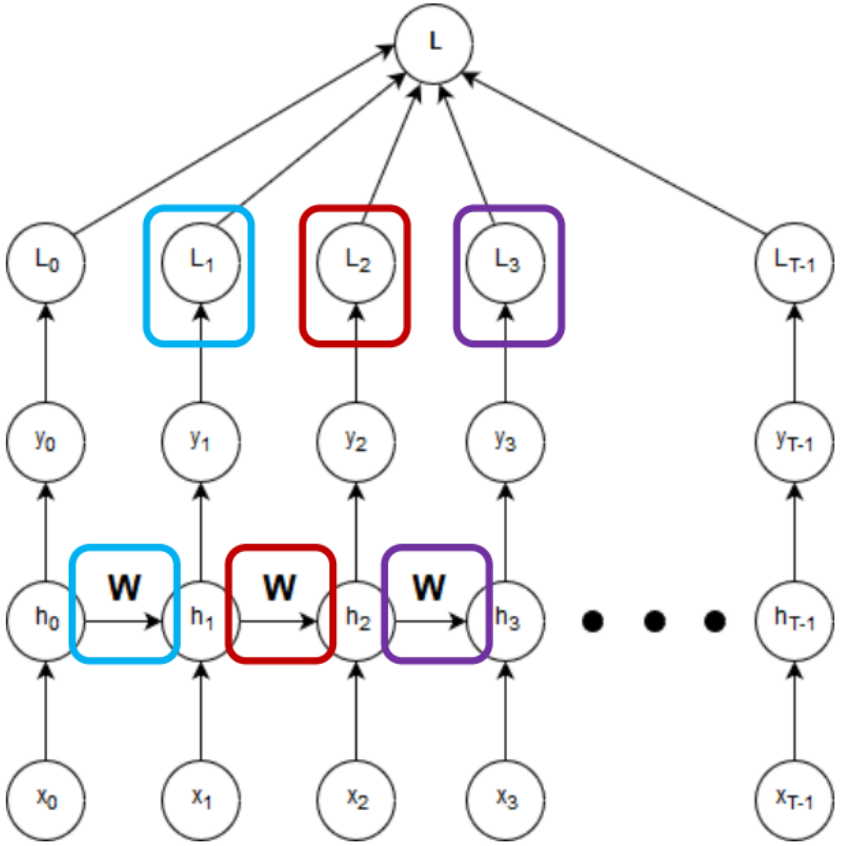


BPTT

For EBP, we need

There are two Summations:

- Sum of temporal Losses
- Sum of gradient path(s)



BPTT

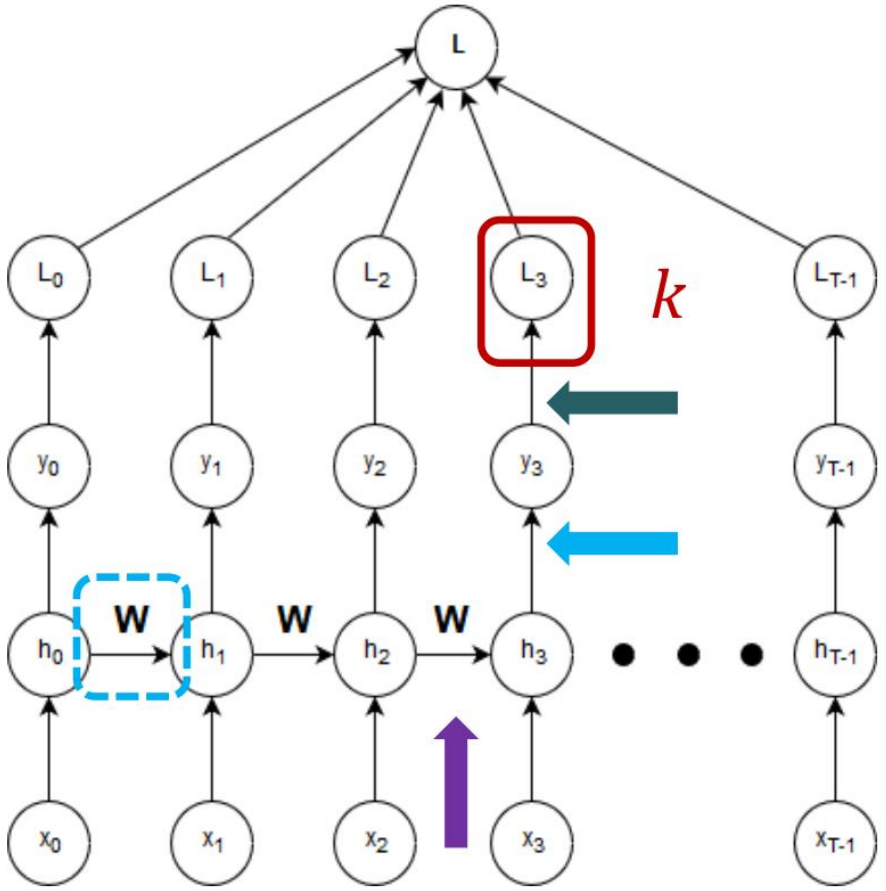
Gradient Summation

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial W}$$

$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Jacobian:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$



BPTT

$$h_m = f(\boldsymbol{W}_h h_{m-1} + \boldsymbol{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \boldsymbol{W}_h^T diag\left(f'(\boldsymbol{W}_h h_{m-1} + \boldsymbol{W}_x x_m)\right)$$

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \boldsymbol{W}_h^T diag\left(f'(\boldsymbol{W}_h h_{m-1} + \boldsymbol{W}_x x_m)\right)$$

Long Term Dependency → Repeated Matrix Multiplication

$$\boldsymbol{Q}^{-1}\boldsymbol{\Lambda}^n\boldsymbol{Q}$$

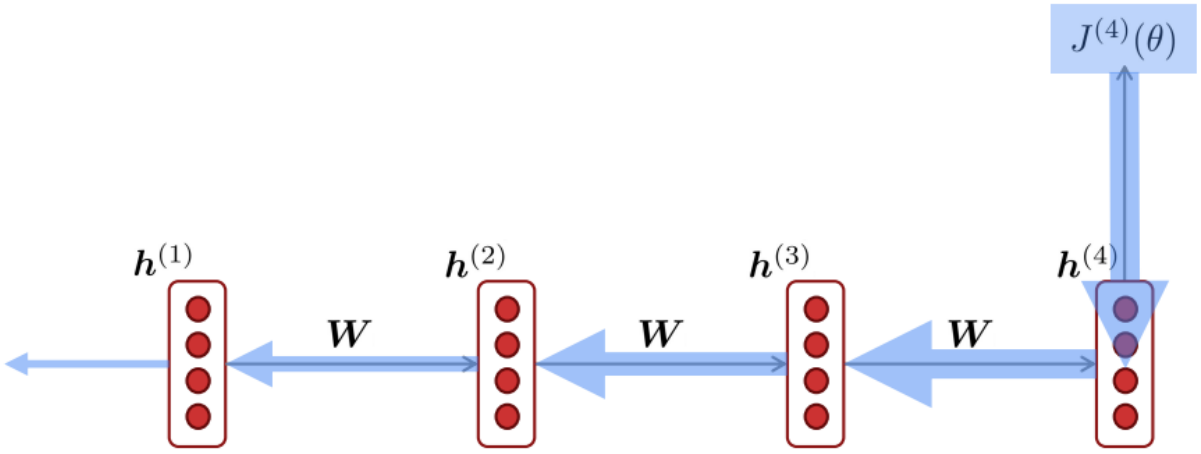
Gradient Vanishing and Exploding

A numerical example

$$\left(\boldsymbol{W}_h^T\right)^n = \boldsymbol{Q}^{-1}\boldsymbol{\Lambda}^n\boldsymbol{Q}$$

$$\boldsymbol{\Lambda} = \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix} \Rightarrow \boldsymbol{\Lambda}^{10} = \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{10} = \begin{bmatrix} 0.00098 & 0 \\ 0 & 57.665039 \end{bmatrix}$$

Vanishing gradient intuition

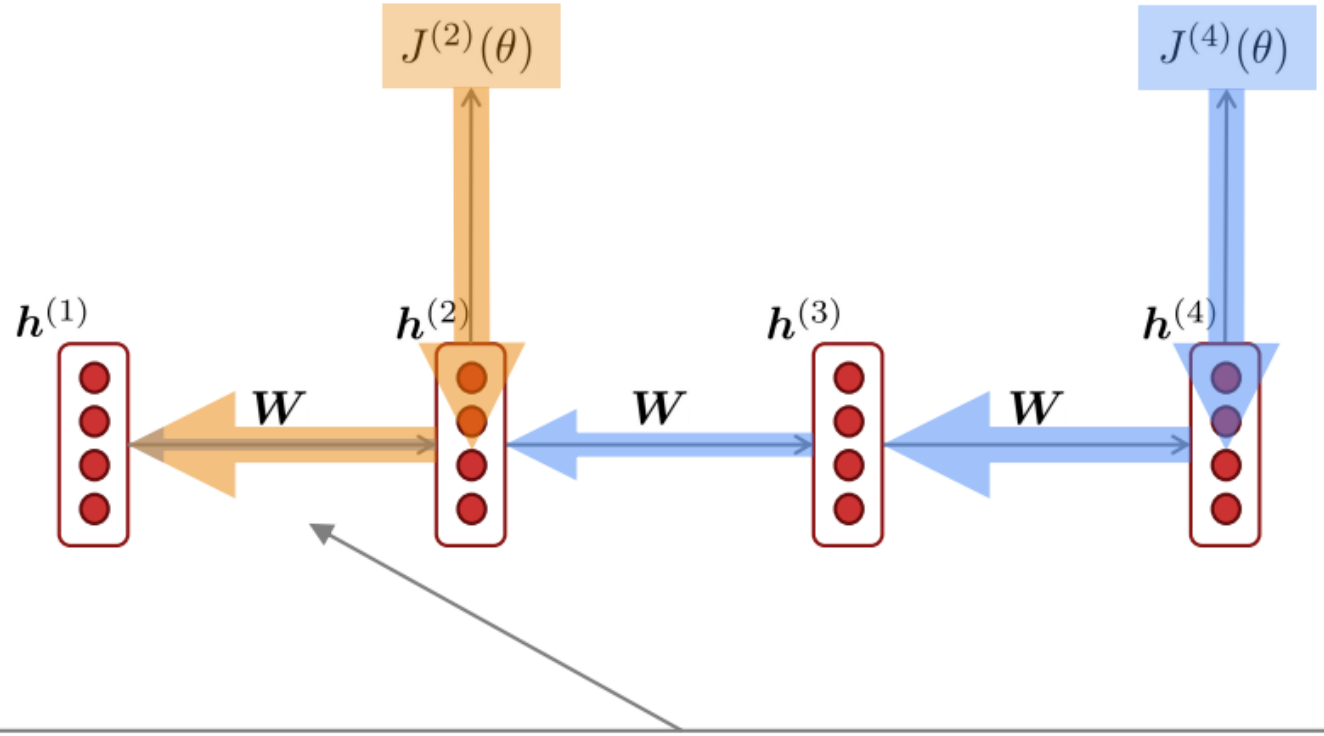


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Why is vanishing gradient a problem?



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are updated only with respect to near effects, not long-term effects.

solution

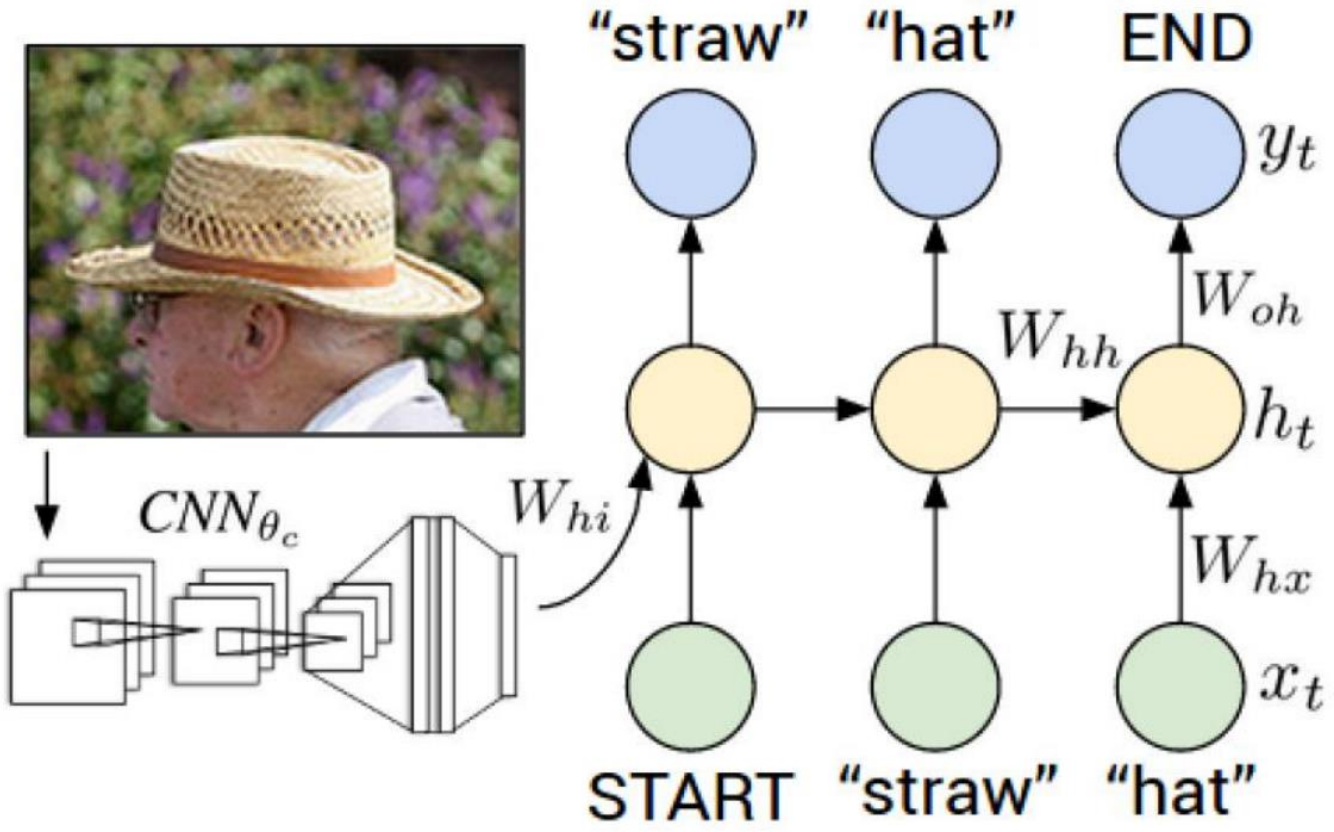
Solution #1 (Gradient Clipping):

- $\hat{\boldsymbol{g}} \leftarrow \frac{\partial Loss}{\partial \boldsymbol{W}}$
- if $\|\hat{\boldsymbol{g}}\| > threshold$ then $\hat{\boldsymbol{g}} \leftarrow \frac{threshold}{\|\hat{\boldsymbol{g}}\|}\hat{\boldsymbol{g}}$

Solution #2 (unity feedback gain, ResBlock idea):

- $\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + F(\boldsymbol{x}_t) \Longrightarrow \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = 1$

Example – Image Captioning



RNN Advantages and Drawbacks

Advantages:

- Possibility of processing input of any length
- Model size not increasing with size (time course) of input
- Computation takes into account historical information
- Weights are shared across time

Drawbacks:

- Computation being slow
- Difficulty of accessing information from a long time ago
- Cannot consider any future input for the current state

Long Short-Term Memory(LSTM)

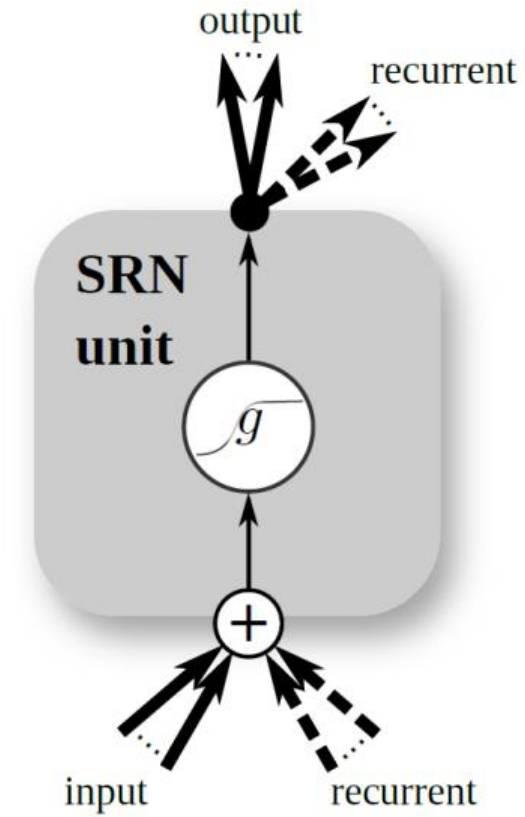
Motivation: Memory Block Operation!

- Read, How much?
- Write, How much?
- Reset , How much?

"How much?" implementation

- Read: input gate (if closed, overwrite by new input is not possible)
- Write: output gate
- Reset: clear saved memory

Simple Recurrent Network



LSTM

LSTM as used in the hidden layers of a RNN:

x: input

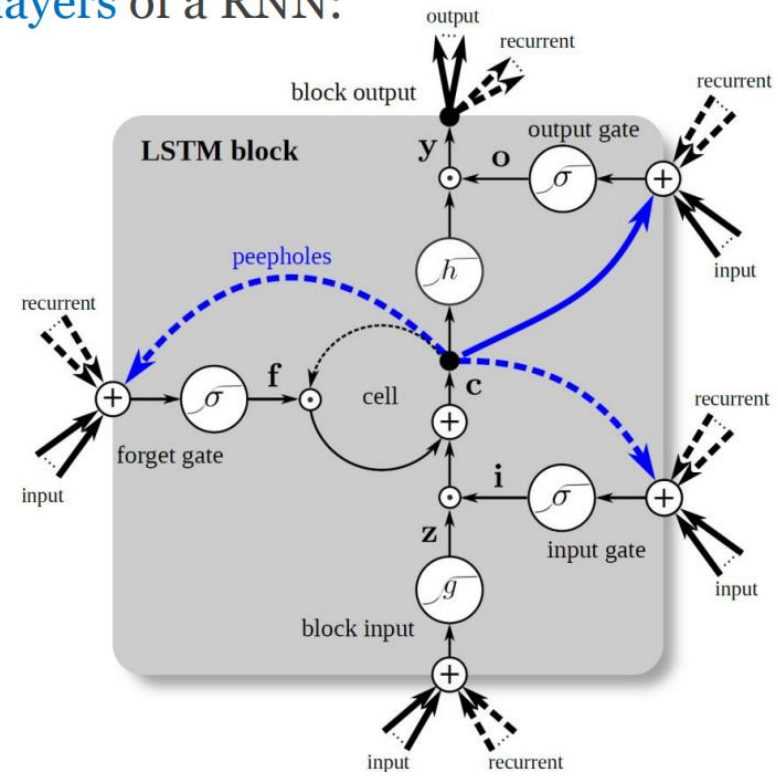
y: output

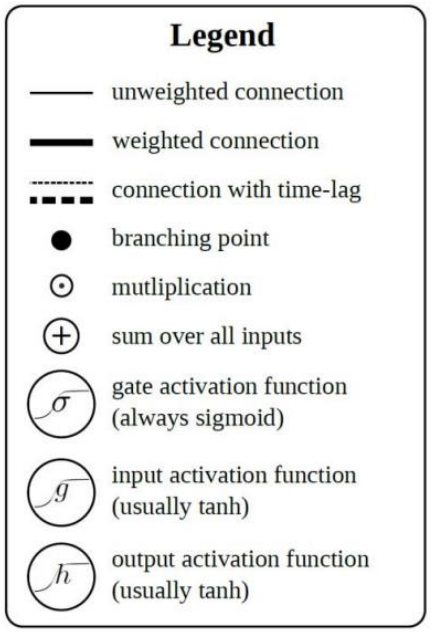
i: input gate

o: output gate

c: memory cell

f: forget gate

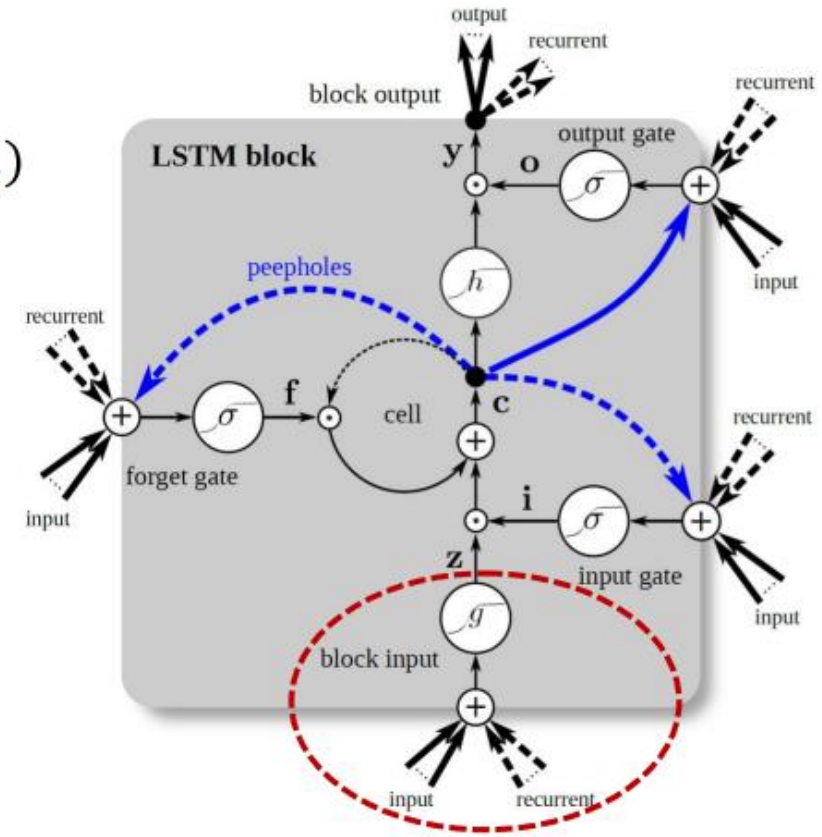


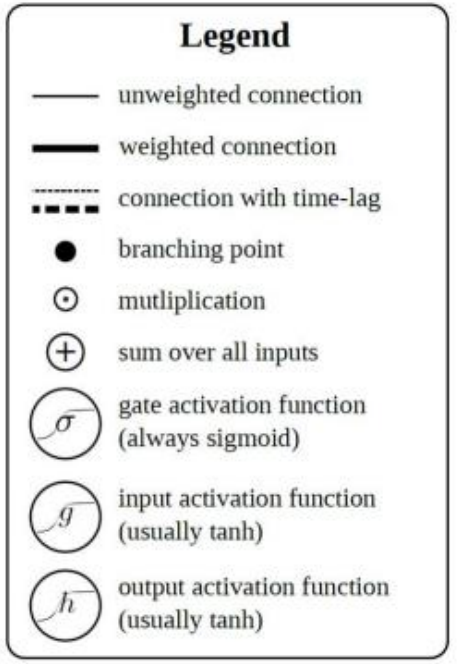


LSTM

- Input Block:

$$\mathbf{z}^t = g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z)$$
$$\mathbf{z}^t = tanh(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z)$$

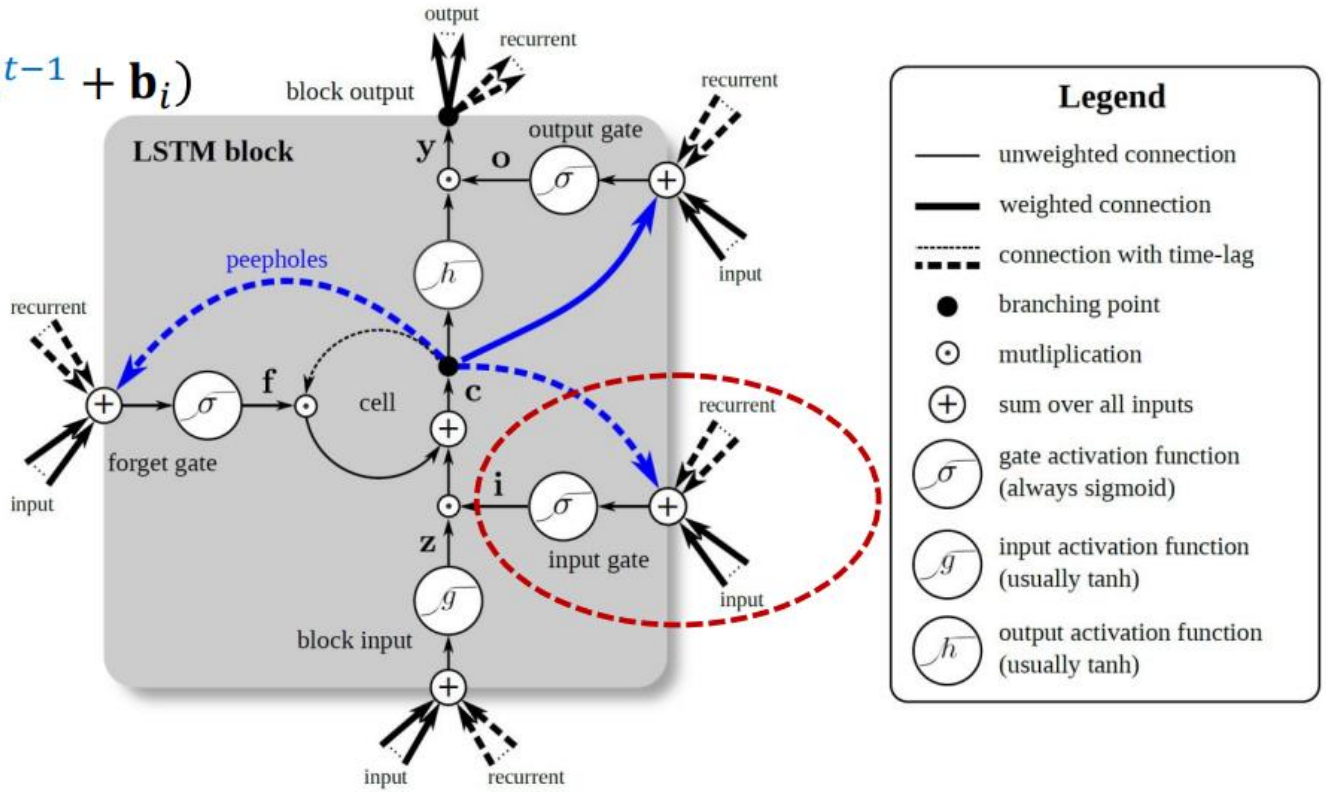




LSTM

- Input Gait:

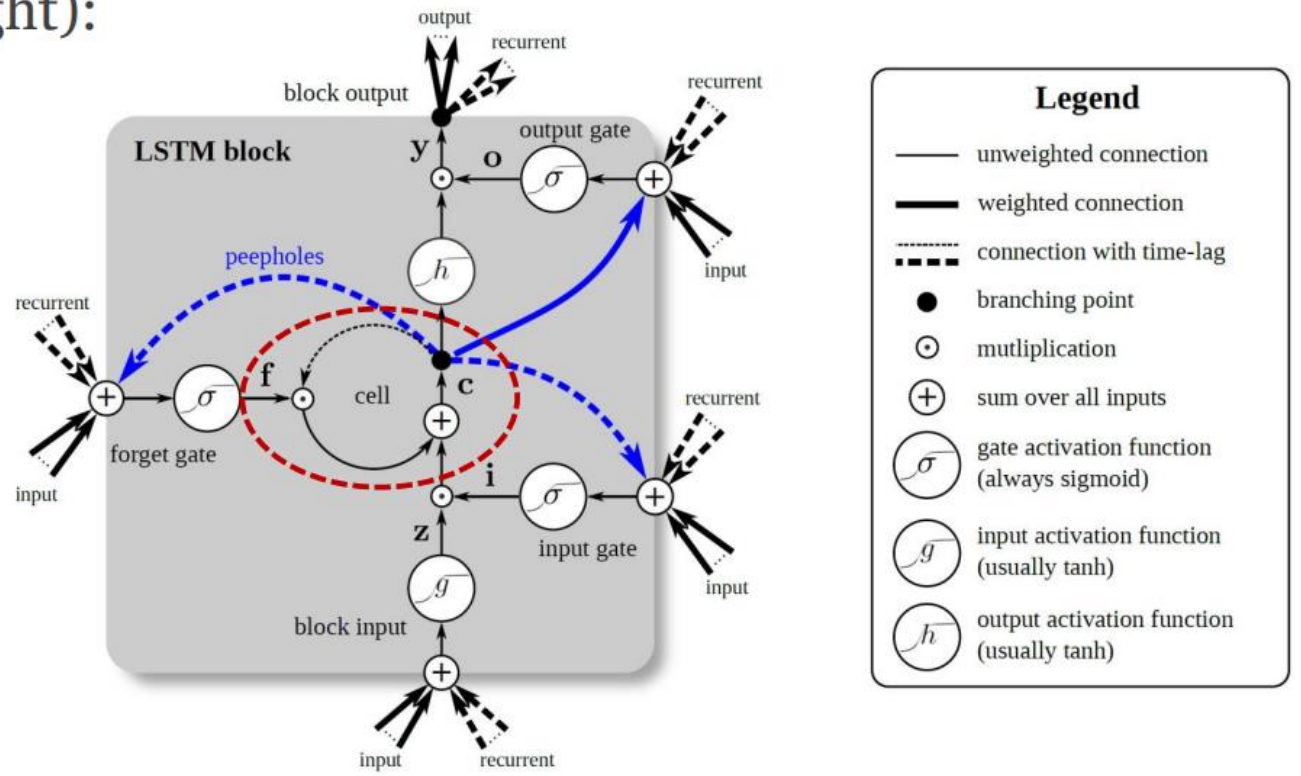
$$\mathbf{i}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i)$$



LSTM

Memory Cell (unity weight):

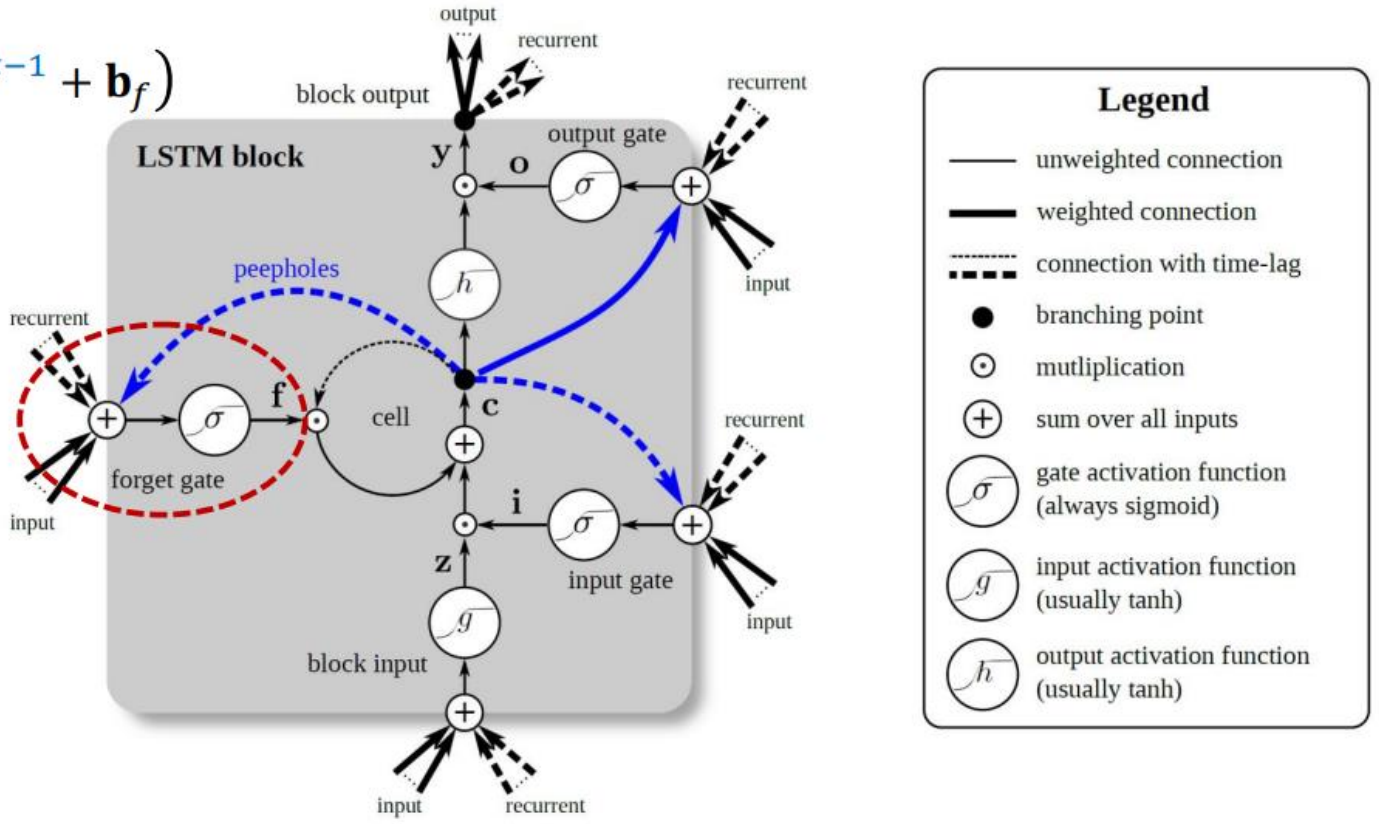
$$\mathbf{c}^t = \boldsymbol{z}^t \odot \boldsymbol{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t$$



LSTM

- Forget Gate:

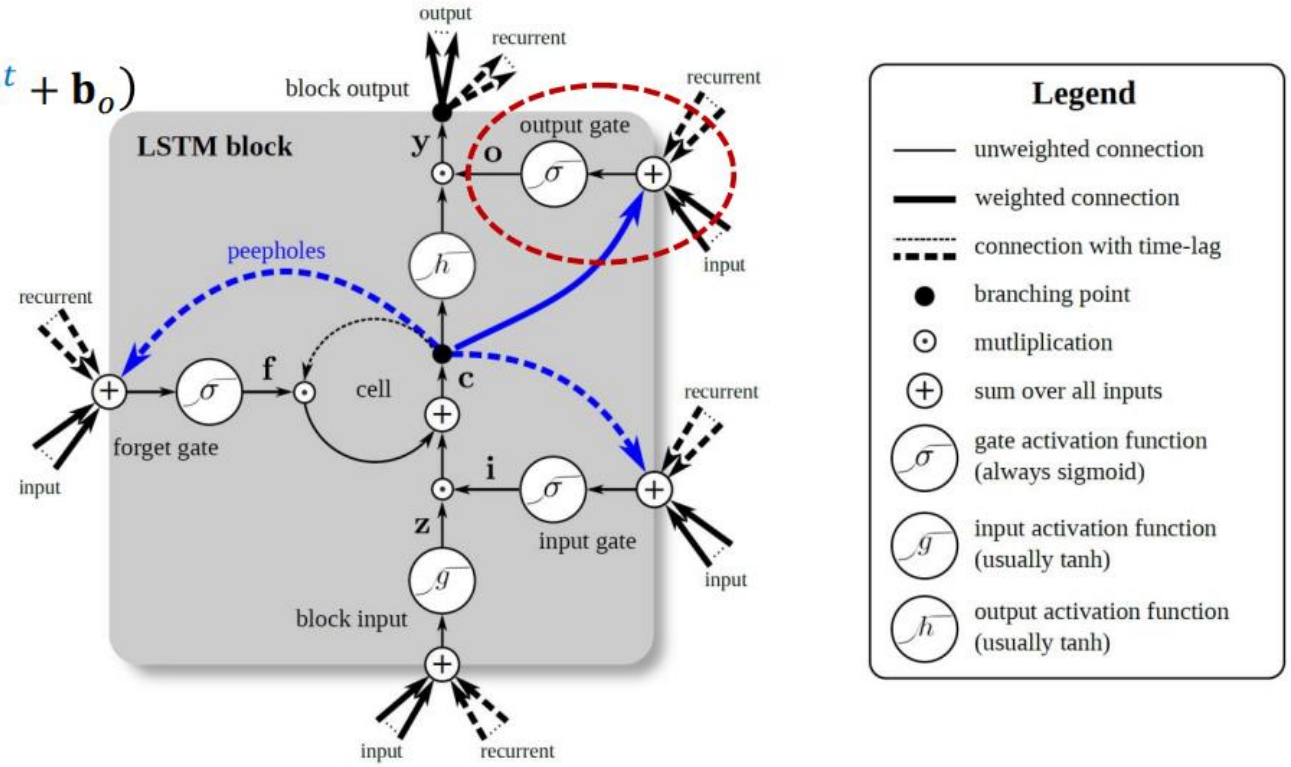
$$\mathbf{f}^t = \sigma\left(\mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f\right)$$



LSTM

- Output Gait:

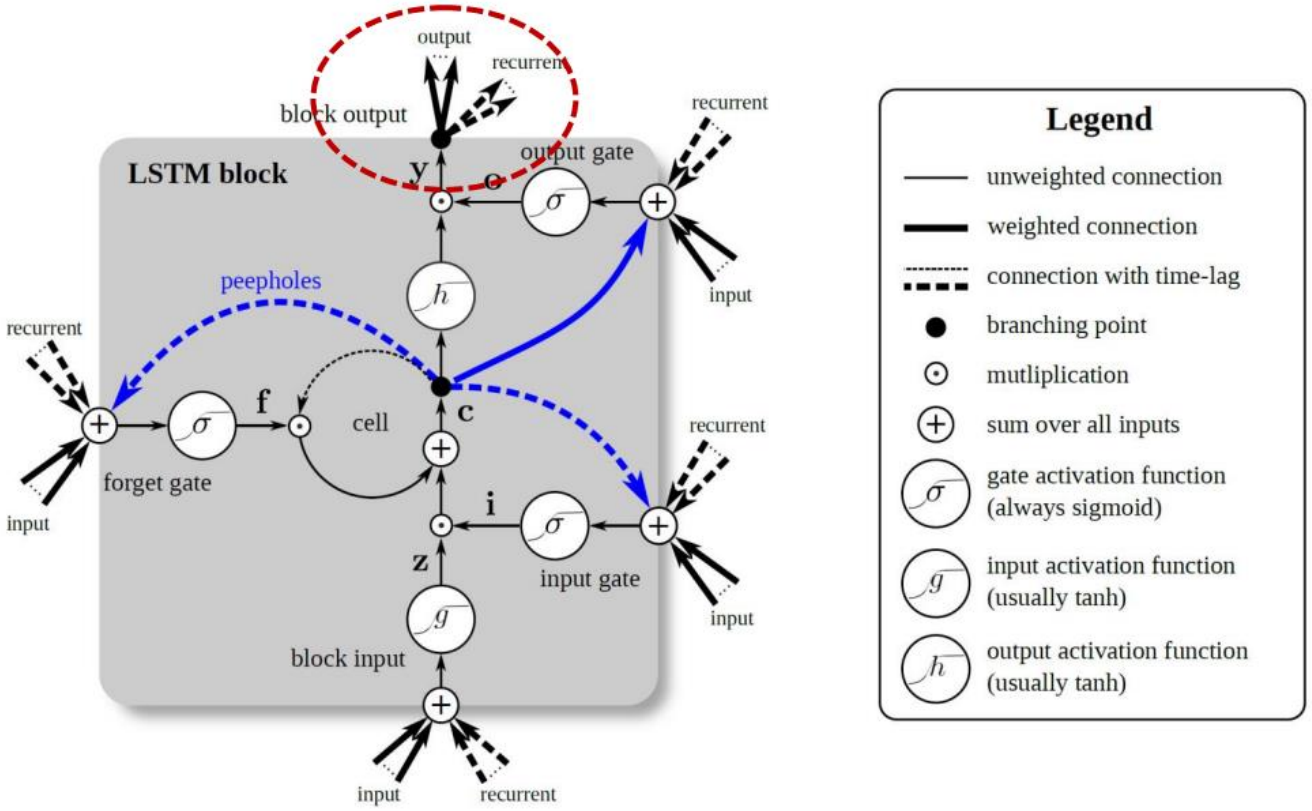
$$\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o)$$



LSTM

Output Block:

$$\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t$$
$$\mathbf{y}^t = tanh(\mathbf{c}^t) \odot \mathbf{o}^t$$



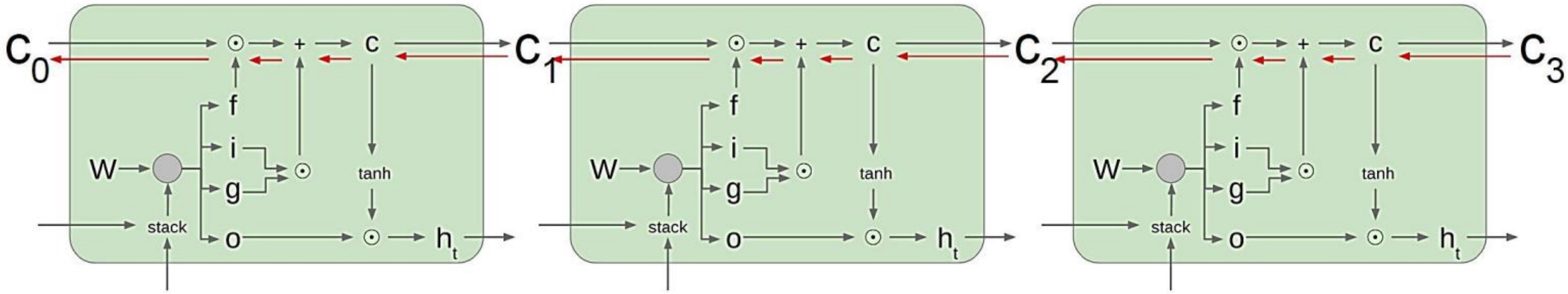
LSTM

- Details:
  - $\mathbf{x}^t \in \mathbb{R}^M$: input vector at time t
  - N: Number of LSTM blocks
  - M: Number of inputs
- Learnable Parameters
  - Input weights: $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{N \times M}$
  - Recurrent weights: $\mathbf{R}_z, \mathbf{R}_i, \mathbf{R}_f, \mathbf{R}_o \in \mathbb{R}^{N \times \mathrm{N}}$
  - Peephole weights: $\mathbf{p}_i, \mathbf{p}_f, \mathbf{p}_o \in \mathbb{R}^N$
  - Bias weights: $\mathbf{b}_z, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^N$

LSTM

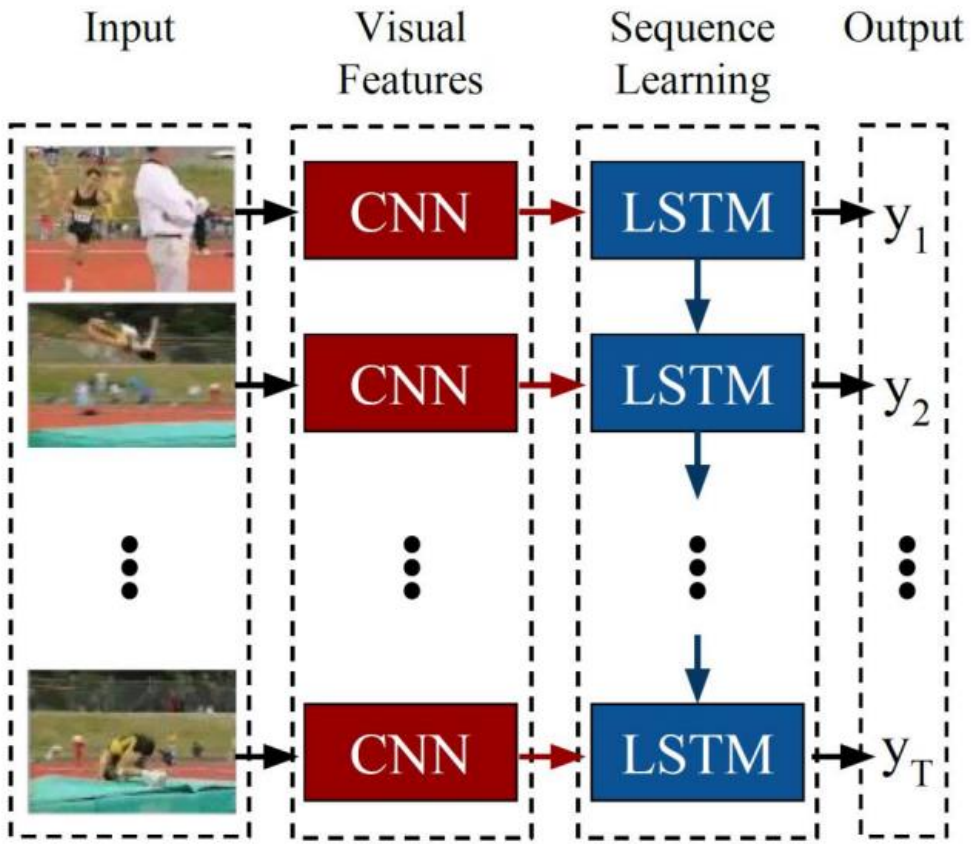
- Forward pass:
  - $\mathbf{z}^t = g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z) = tanh(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z)$
  - $\mathbf{i}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i)$
  - $\mathbf{f}^t = \sigma\left(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f\right)$
  - $\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t$
  - $\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o)$
  - $\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t = tanh(\mathbf{c}^t) \odot \mathbf{o}^t$

LSTM - Unfolding



LSTM Application

Visual Recognition and Description



LSTM application

A large clock mounted to the side of a building.

A bunch of fruit that are sitting on a table.

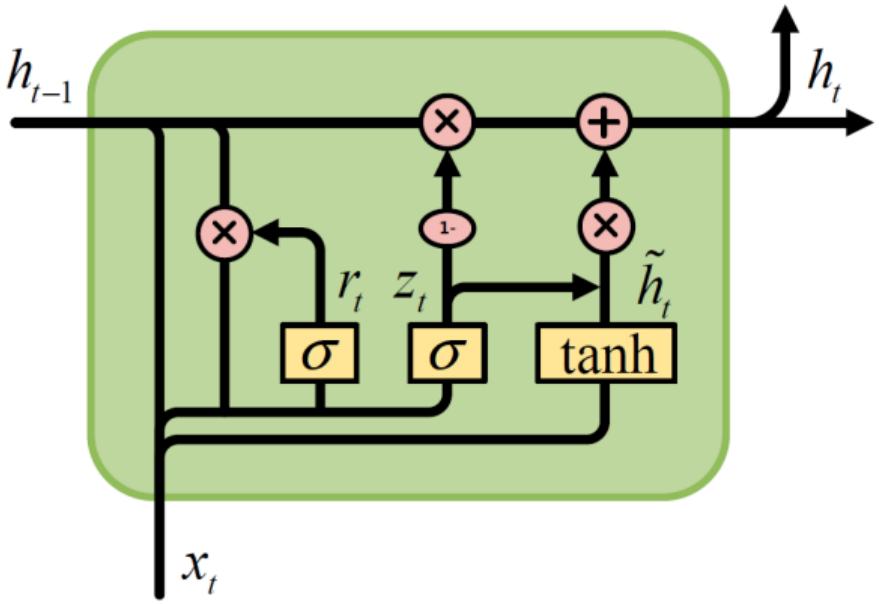
A toothbrush holder sitting on top of a white sink.

Gate Recurrent Unit (GRU)

The GRU is like a long short-term memory (LSTM) with a forget gate but:

Has fewer parameters than LSTM, as it lacks an output gate.

- Fully Gated Unit



- Equation:
- $\mathbf{z}_t = \sigma_g(\mathbf{W}_{xz}\mathbf{x}_t + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_z)$
- $\mathbf{r}_t = \sigma_g(\mathbf{W}_{xr}\mathbf{x}_t + \mathbf{W}_{hr}\boldsymbol{h}_{t-1} + \mathbf{b}_r)$
- $\hat{\mathbf{h}}_t = \phi_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hr}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h)$
- $\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \hat{\mathbf{h}}_t$
- σ_g: sigmoid function
- ϕ_h: tanh function

GRU vs LSTM

GRU: "update" and "reset" gates

LSTM: "input", "output", and "forget" gates

No internal memory state c in GRU

No nonlinearity (sigmoid) before the output gate in GRU

NLP

Tokenization

Word Embedding

Model Generation

Tokenization

The process of converting a sequence of text into smaller parts, known as tokens. These tokens can be as small as characters or as long as words.

Consider: Chatbots based on deep learning are awesome, why not use?

Techniques:

- Character level tokenization:

  ["C", "h", …, “?"]

- Word level tokenization:

  ["Chatbots", "based", …, "awesome,", …, “use?"]

- Sub-word level tokenization:

  ["Chat", "bots", "based", …, "learn", "ing“, "awe ", "some", ".", …]

- Sentence level tokenization:

  ["Chatbots based on deep learning are awesome", "why not use?"]

Word Embedding

Definition: A representation of a word (token).

Typically, the representation is a real-valued vector that encodes the meaning of the word in such a way that words that are closer in the vector space are expected to be similar in meaning.

Example:

$$Deep \longrightarrow \begin{bmatrix} 1.1 \\ -2 \\ 15 \\ 9 \end{bmatrix}$$

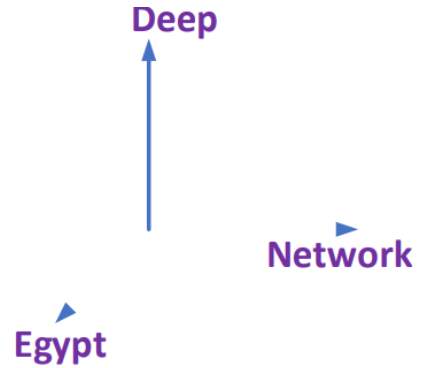
Naïve approach: one-hot coding

Naïve approach: one-hot coding

$$Deep \longrightarrow \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, Network \longrightarrow \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, Egypt \longrightarrow \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

Problems:

- Sparse and Low Capacity
- High-dimensional
- Hard-coded (new words new dimension)
- No similarity information ($Deep \perp Network \perp Egypt$)



Word Embedding

Word Embedding using embedding matrix

$$E_{word} = \mathbf{W} \times O_{word}$$

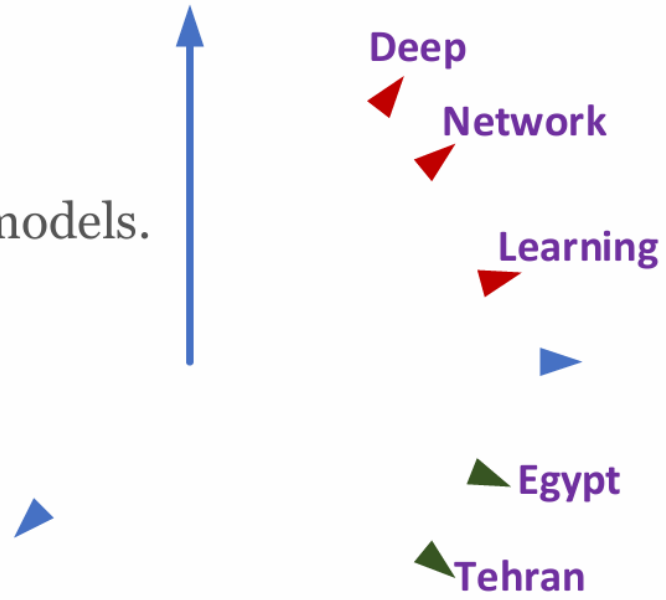
$O_{word} \in \{0,1\}^V$ one-hot vectors

- Typical value of V: 783M, 1.6B, 6B

$E_{word} \in \mathbb{R}^N, N \ll V$

- Typical value of E: 100, 300, 600, 1000

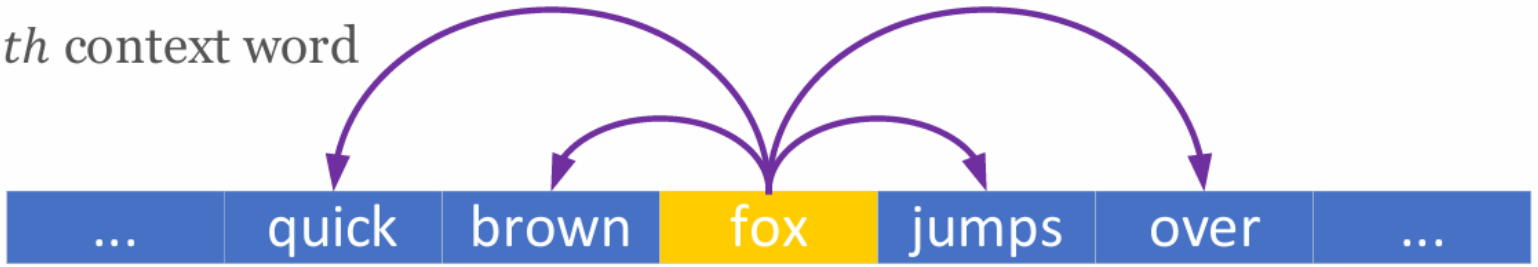
$\mathbf{W} \in \mathcal{R}^{N \times V}$: can be learned using target/context likelihood models.



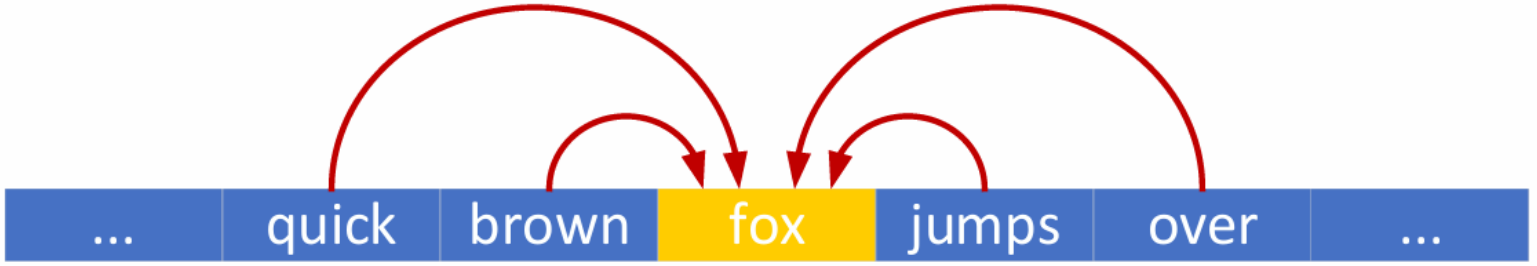
Word2vec – Two Policies

Estimate $P\{w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c\}$

- w_c : center word
- w_{c-j} : jth context word



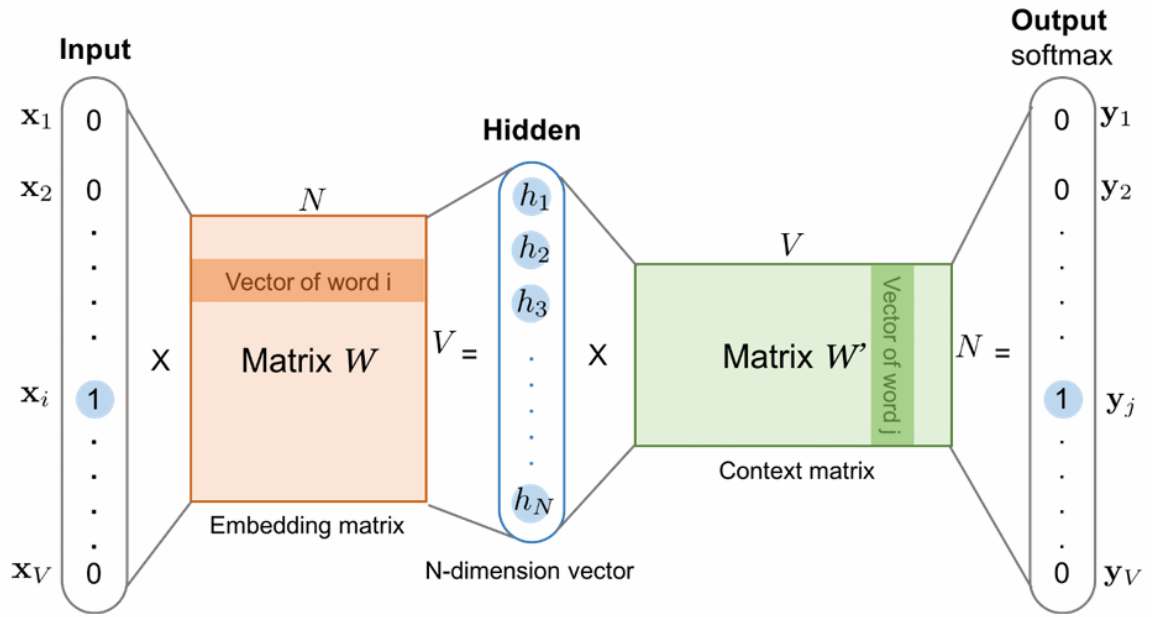
Estimate $P\{w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}\}$



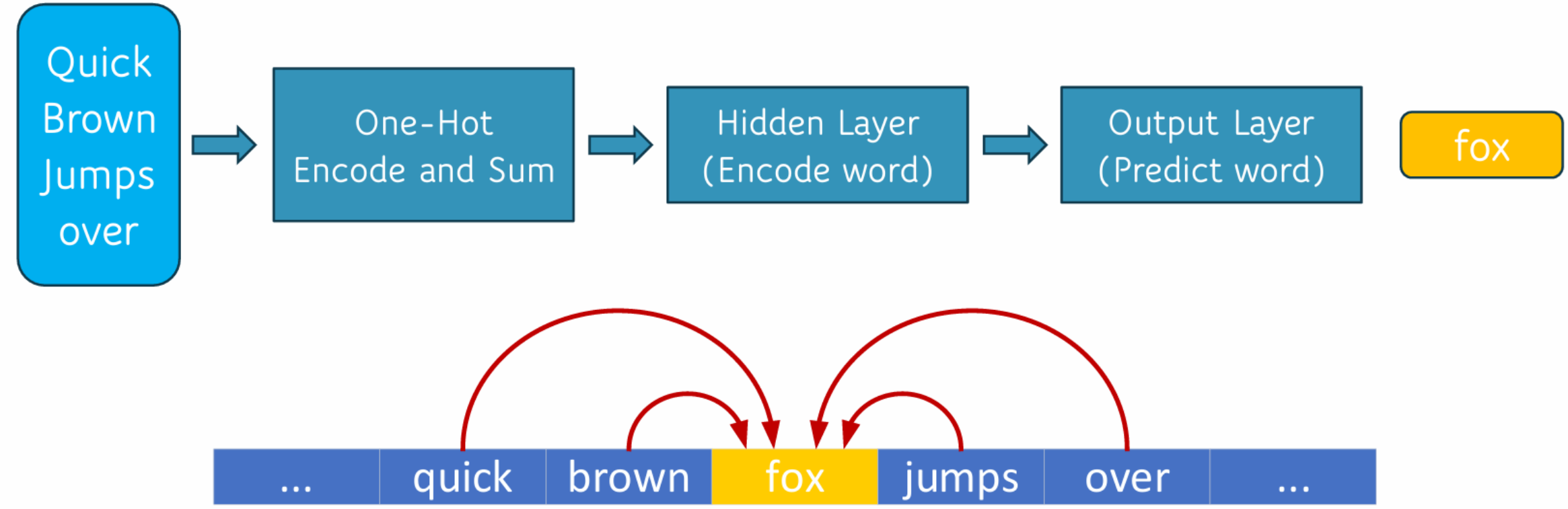
Word2vec – Network Architecture

Architecture is too simple but HUGE:

- Input layer (one-hot), $O_{word} \in \{0,1\}^V, V = \mathcal{O}(10^{6-8})$
- Linear hidden layer, $E_{word} \in \mathbb{R}^N, N = \mathcal{O}(10^{2-3})$
- Output Layer (SoftMax), $P_{word} \in [0,1]^V, V = \mathcal{O}(10^{6-8})$



Word2vec – Learning Algorithms



Word2vec

An interesting arithmetic properties:

$vector('Rome') \approx vector('Paris') - vector('France') + vector('Italy')$

$vector('Queen') \approx vector('King') - vector('Man') + vector('Woman')$

Machine Translation

Machine Translation (MT) is the task of translating a sentence x from one language (the source language) to a sentence y in another language (the target language)

x: *L'homme est né libre, et partout il est dans les fers*

y: *Man is born free, but everywhere he is in chains*

1990s-2010s: Statistical Machine Translation

Core idea : Learn a probabilistic model from data

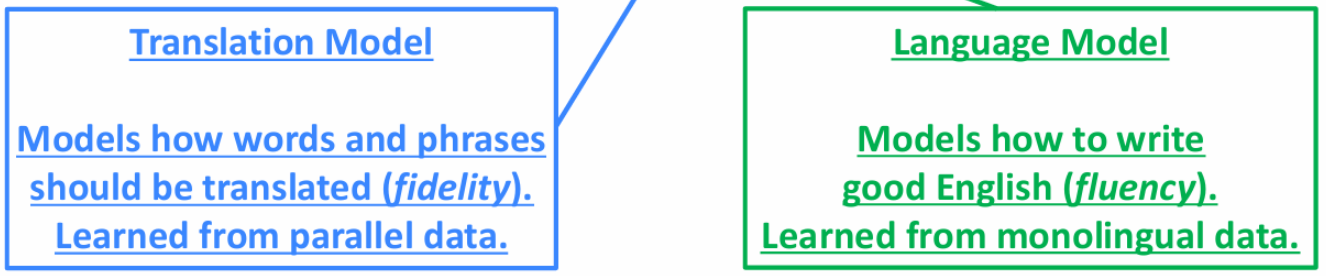
Suppose we're translating French → English.

We want to find best English sentence y, given French sentence x

$$\text{argmax}_y P(y|x)$$

Use Bayes Rule to break this down into two components to be learned separately:

$$= \text{argmax}_y P(x|y)P(y)$$



1990s–2010s: Statistical Machine Translation

SMT was a huge research field

The best systems were extremely complex

- Hundreds of important details

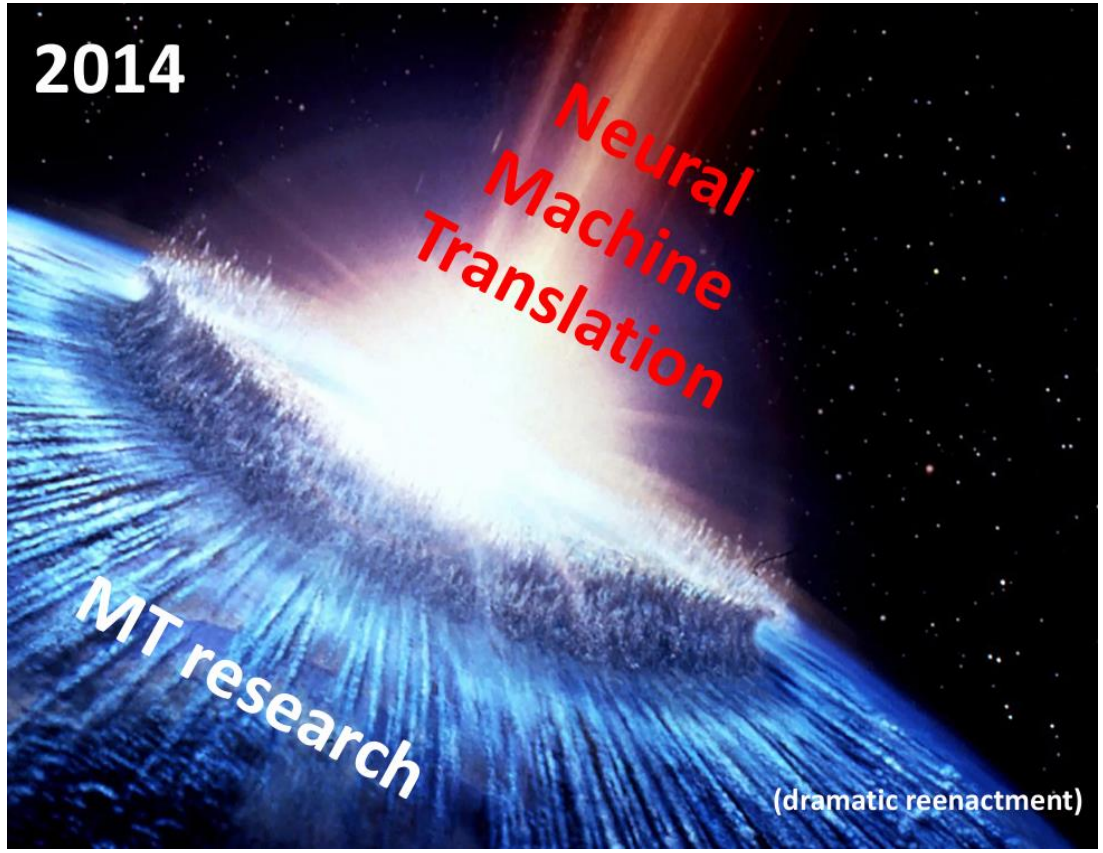
Systems had many separately-designed subcomponents

- Lots of feature engineering
  - Need to design features to capture particular language phenomena
- Required compiling and maintaining extra resources
  - Like tables of equivalent phrases
- Lots of human effort to maintain
  - Repeated effort for each language pair!

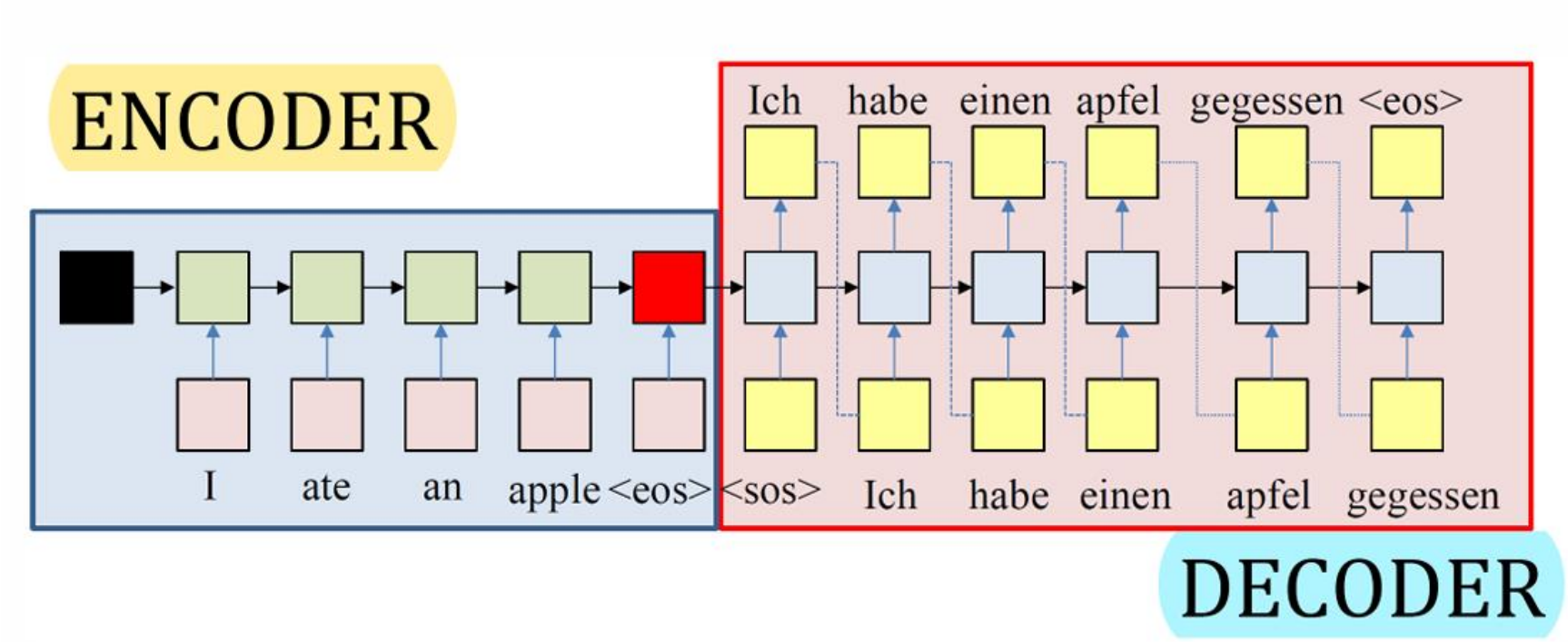
Neural Machine Translation

Neural Machine Translation (NMT) is a way to do Machine Translation with a single end-to-end neural network

The neural network architecture is called a sequence-to-sequence model (aka seq2seq) and it involves two RNNs

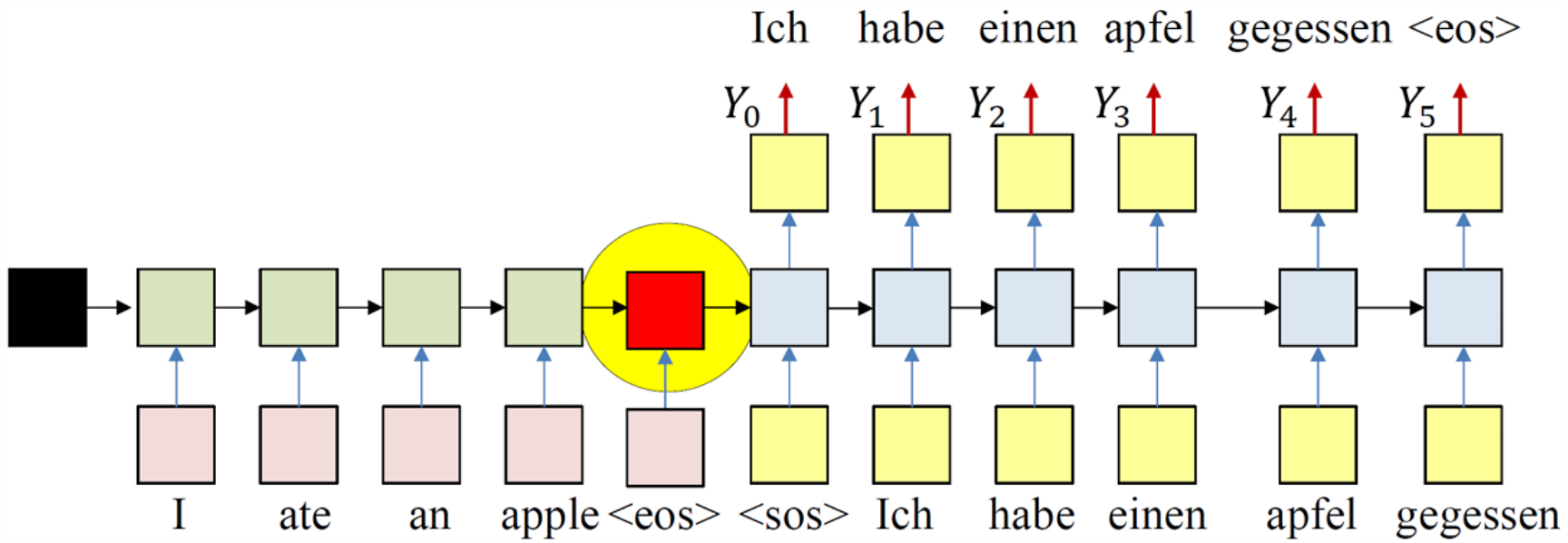


Seq2Seq Architecture



Simple Seq2Seq Architecture

All the information about the input sequence is embedded into a single vector, the last hidden neuron “overloaded” with information (Particularly for the long sequences)



Sequence-to-sequence is versatile!

The general notion here is an encoder-decoder model

- One neural network takes input and produces a neural representation
- Another network produces output based on that neural representation
- If the input and output are sequences, we call it a seq2seq model

Sequence-to-sequence is useful for more than just MT

Many NLP tasks can be phrased as sequence-to-sequence:

- Summarization (long text → short text)
- Dialogue (previous utterances → next utterance)
- Parsing (input text → output parse as sequence)
- Code generation (natural language → Python code)

the first big success story of NLP Deep Learning

Neural Machine Translation went from a fringe research attempt in 2014 to the leading standard method in 2016

2014: First seq2seq paper published [Sutskever et al. 2014]

2016: Google Translate switches from SMT to NMT – and by 2018 everyone had

This was amazing!

SMT systems, built by hundreds of engineers over many years, were outperformed by NMT systems trained by small groups of engineers in a few months