

# Recurrent Neural Network (RNN)

Deep-people #6

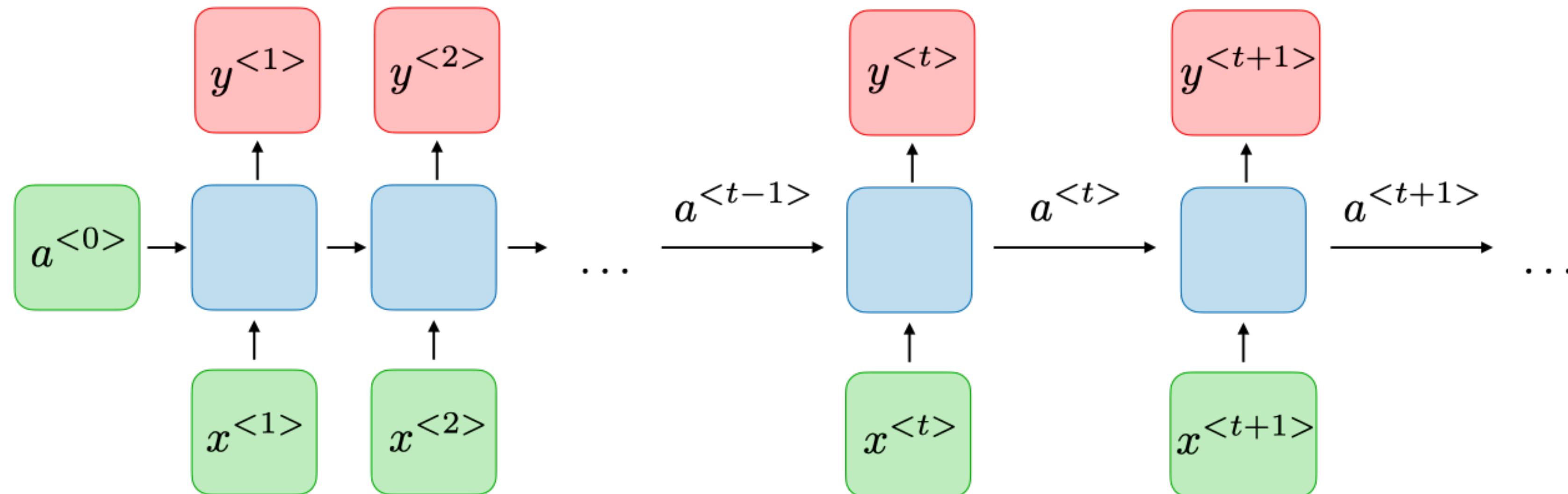
# 前回のおさらい

- **CNN**
  - 局所性と位置不变性
  - 置み込み層, プーリング層, 全結合層
  - 有名モデルの紹介

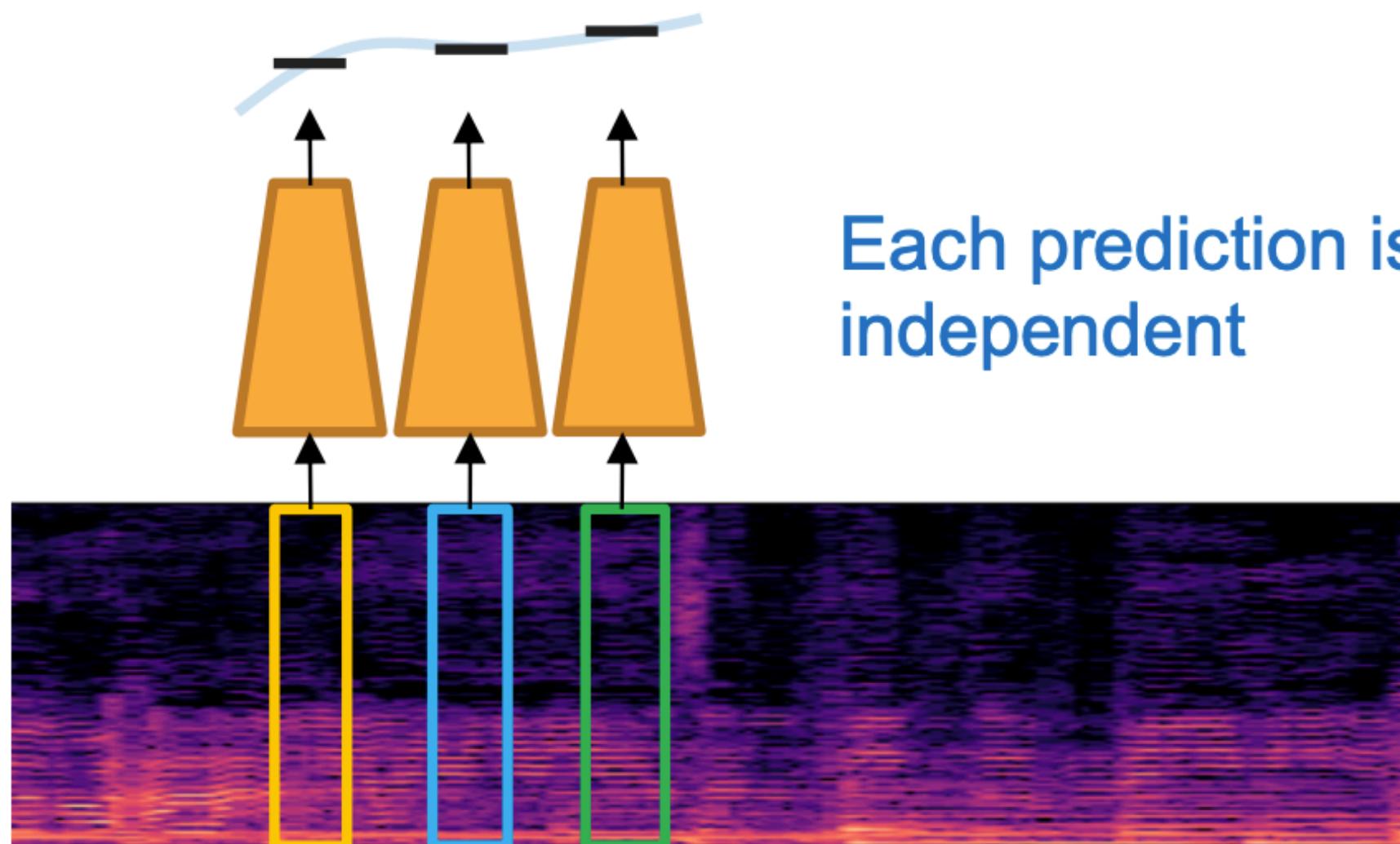
- Juhanの資料
- [https://mac.kaist.ac.kr/~juhan/gct634/Slides/\[week9-1\]%20recurrent%20neural%20network.pdf](https://mac.kaist.ac.kr/~juhan/gct634/Slides/[week9-1]%20recurrent%20neural%20network.pdf)
- Under standing LSTM Networks
- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# 今回のお話

- 再帰型ニューラルネットワーク（以降、RNN）
  - 自然言語処理で成功を遂げたニューラルネットワーク
  - 特に系列データの処理に強いとされている



- 音のデータはしばしば系列データを出力することが求められる
  - CNNも使えるが、あくまで1フレーム分の処理を独立に行う（下図）
  - →他のフレームの影響=コンテキスト情報を活用できないか？

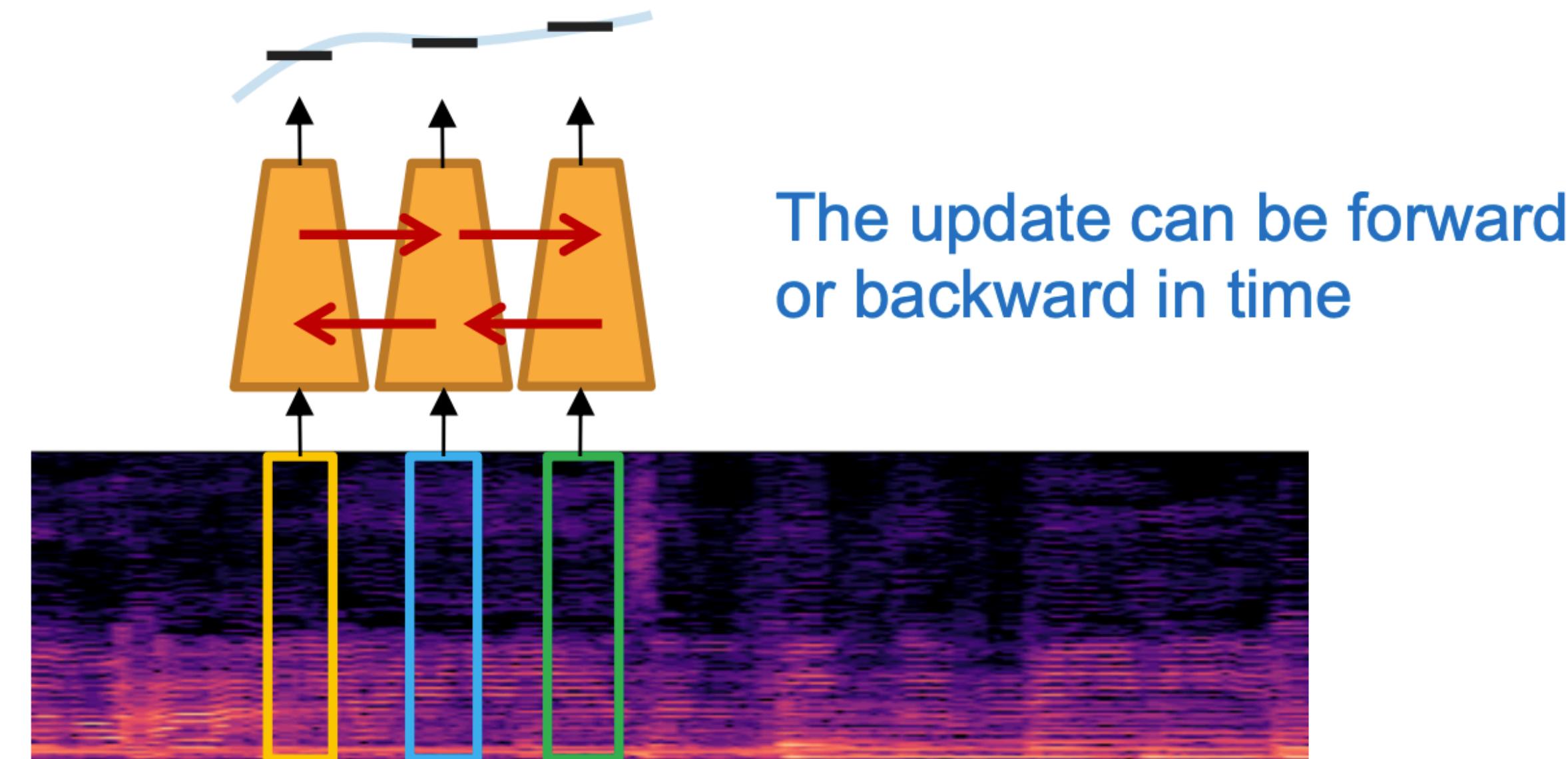


ex. ピッチ推定 by CNN

同じピッチを伸ばしていても、その時点での情報だけでピッチの推定値を決めてしまう  
=他の時点の推定結果を活用できていない

# 他のフレーム情報を繋げる

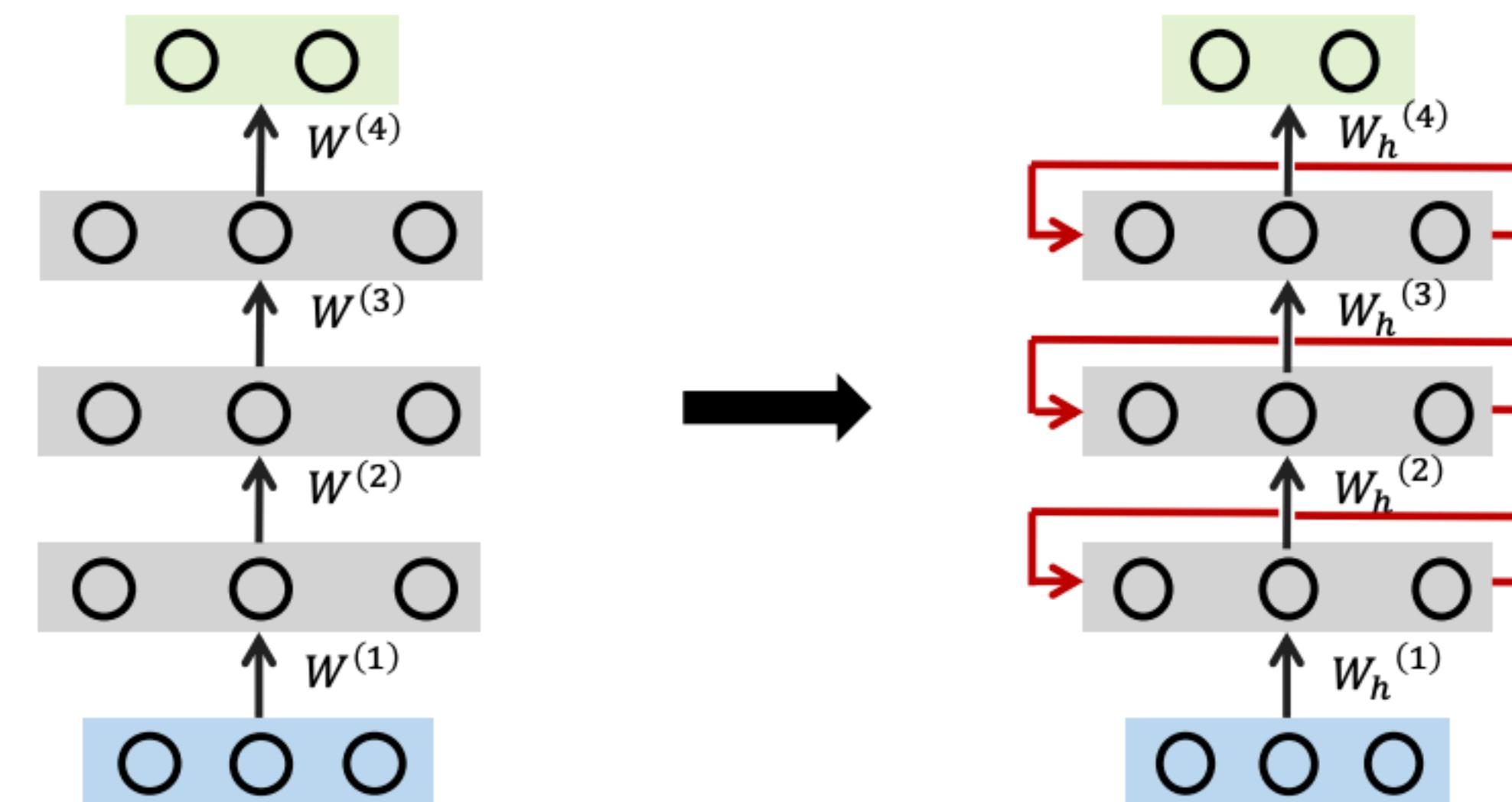
- 着目時点の入力に加え、過去 or 未来の隠れ層の状態を活用する
- 隠れ層の状態を文脈情報として入力する
- →より広い範囲の影響を考慮できるように！



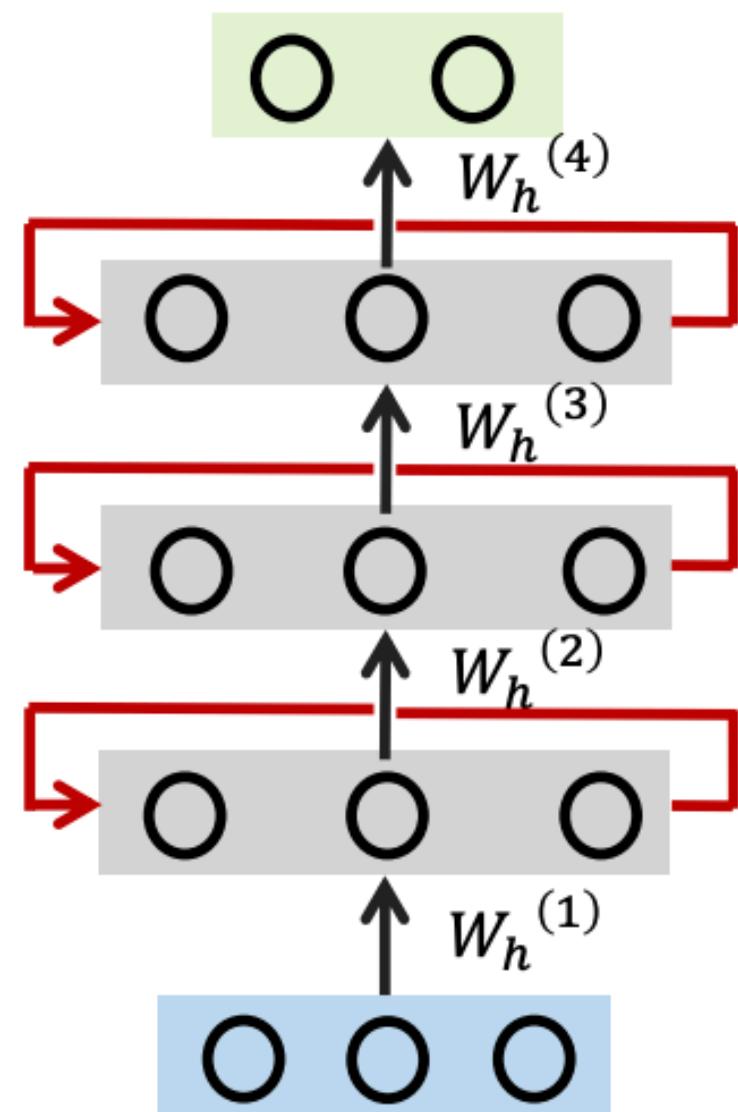
# Recurrent Neural Network (RNN)

7

- 前時刻の隠れ層の状態と、現在の状態を入力するニューラルネットワーク



- いわゆる最もシンプルなRNN
- 前時刻の隠れ状態の重み $W_t$ と現時刻の入力の重み $W_h$ （バイアス $b$ も）の2種類が存在
- 活性化関数 $g(\cdot)$ にはtanhがよく用いられる
- 出力層にはアフィン変換+活性化関数 $f$ をかける（ $g$ と区別しないことも）



$$\hat{y}(t) = f(W_h^{(4)} h^{(3)}(t) + b^{(4)})$$

$$h^{(3)}(t) = g(W_t^{(3)} h^{(3)}(t-1) + W_h^{(3)} h^{(2)}(t) + b^{(3)})$$

$$h^{(2)}(t) = g(W_t^{(2)} h^{(2)}(t-1) + W_h^{(2)} h^{(1)}(t) + b^{(2)})$$

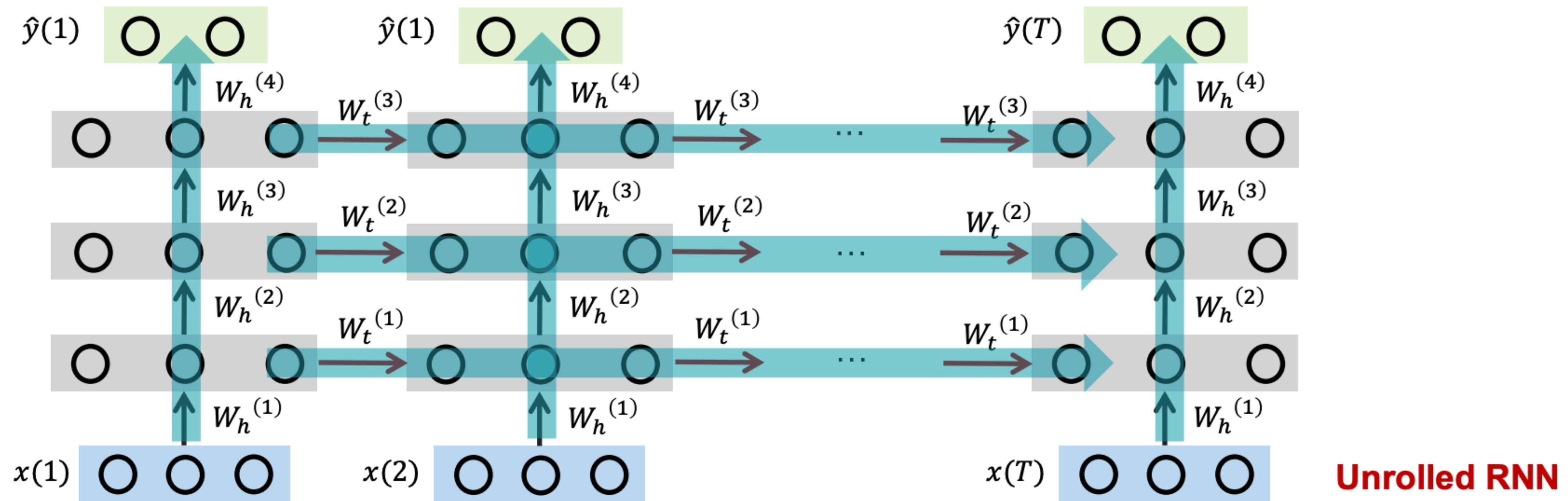
$$h^{(1)}(t) = g(W_t^{(1)} h^{(1)}(t-1) + W_h^{(1)} x(t) + b^{(1)})$$

$t = 0, 1, 2, \dots$

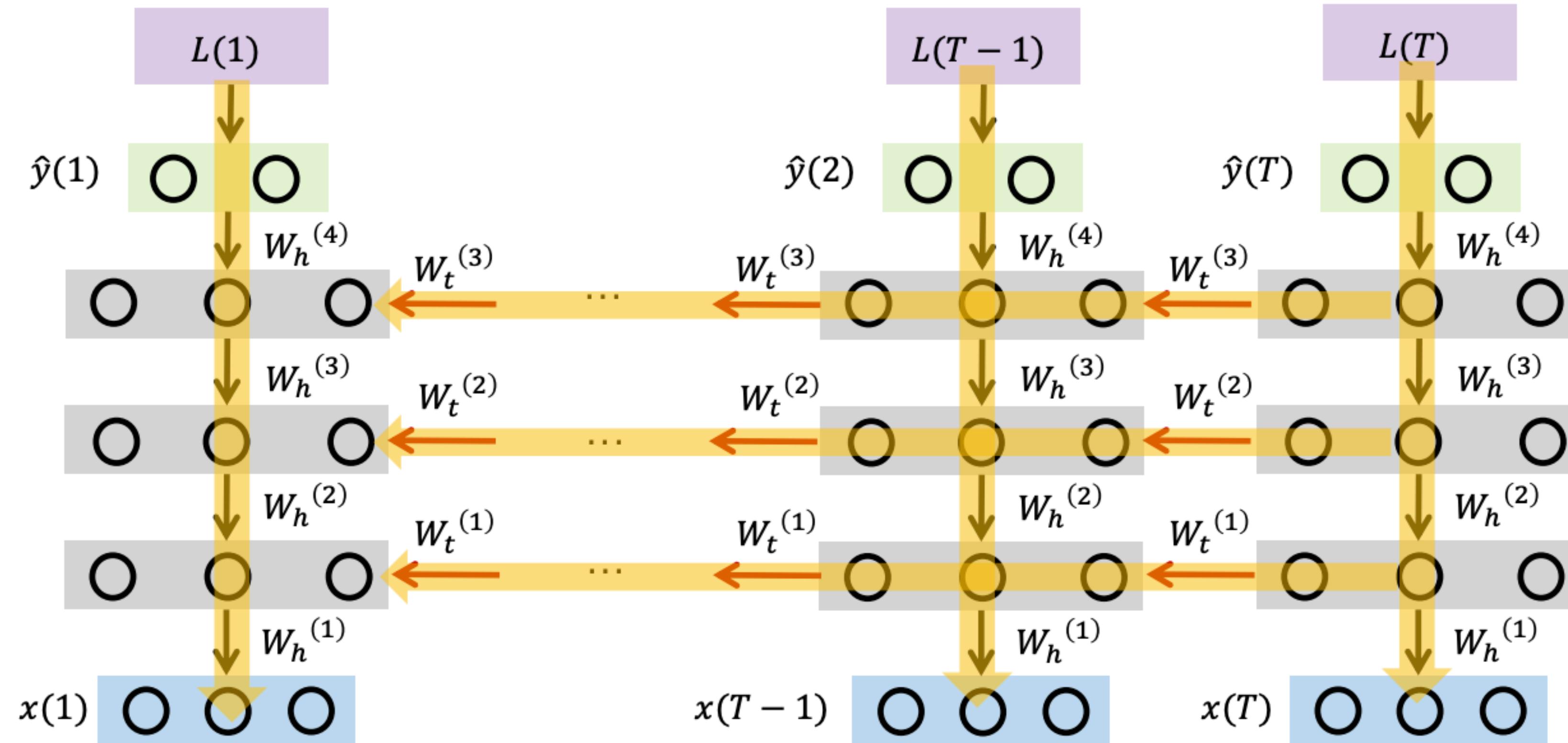
recurrent connections: how much information from  
the previous state is used to update the current state

# 順伝搬 (forward計算)

- 各時刻ごとの状態を横に展開する
  - 各時刻で重み ( $W_h^{(i)}, W_t^{(i)}$ ) を共有する, どでかい1つのNNと捉える



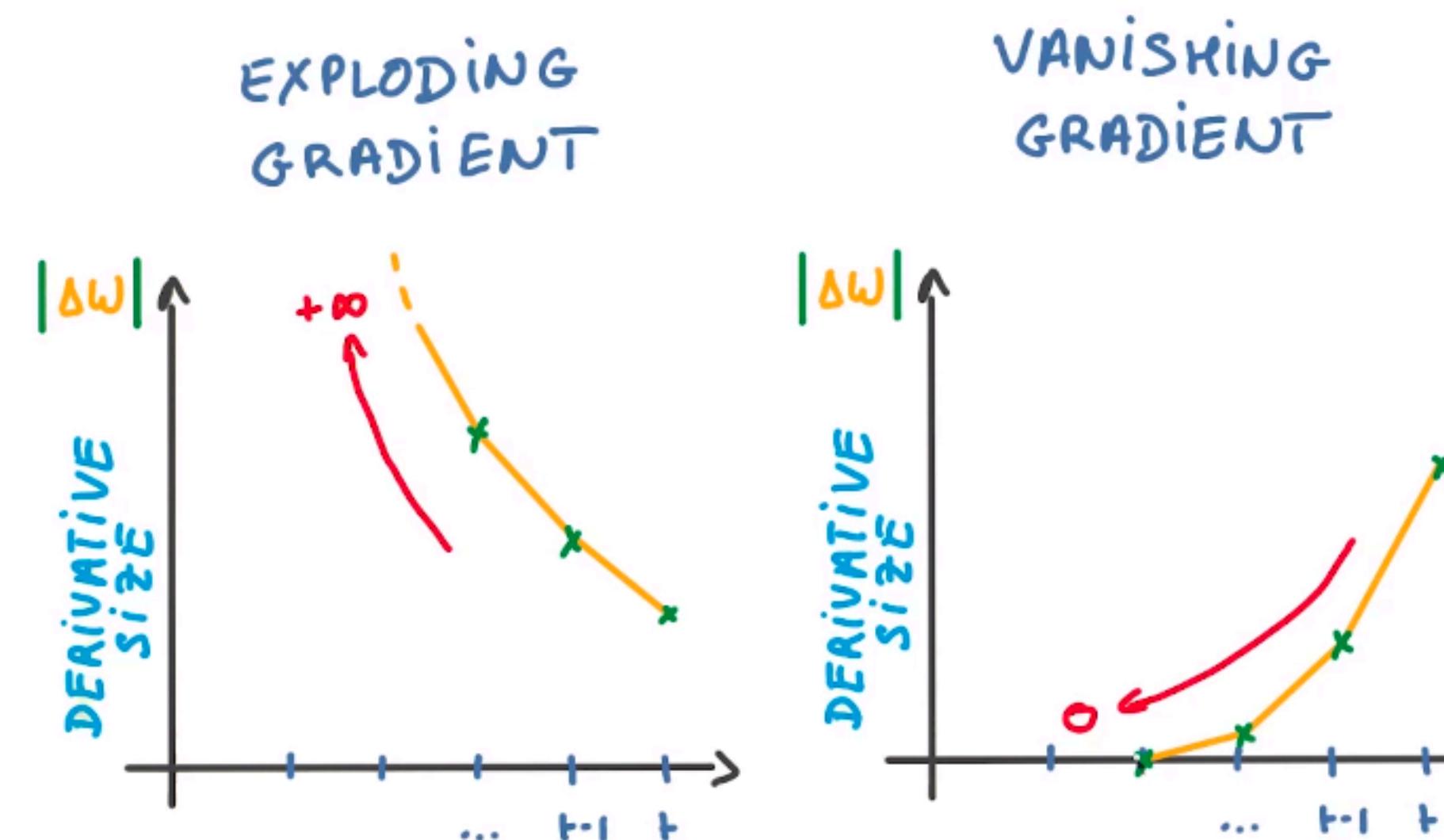
## Back-propagation through time (BPTT)とよばれる



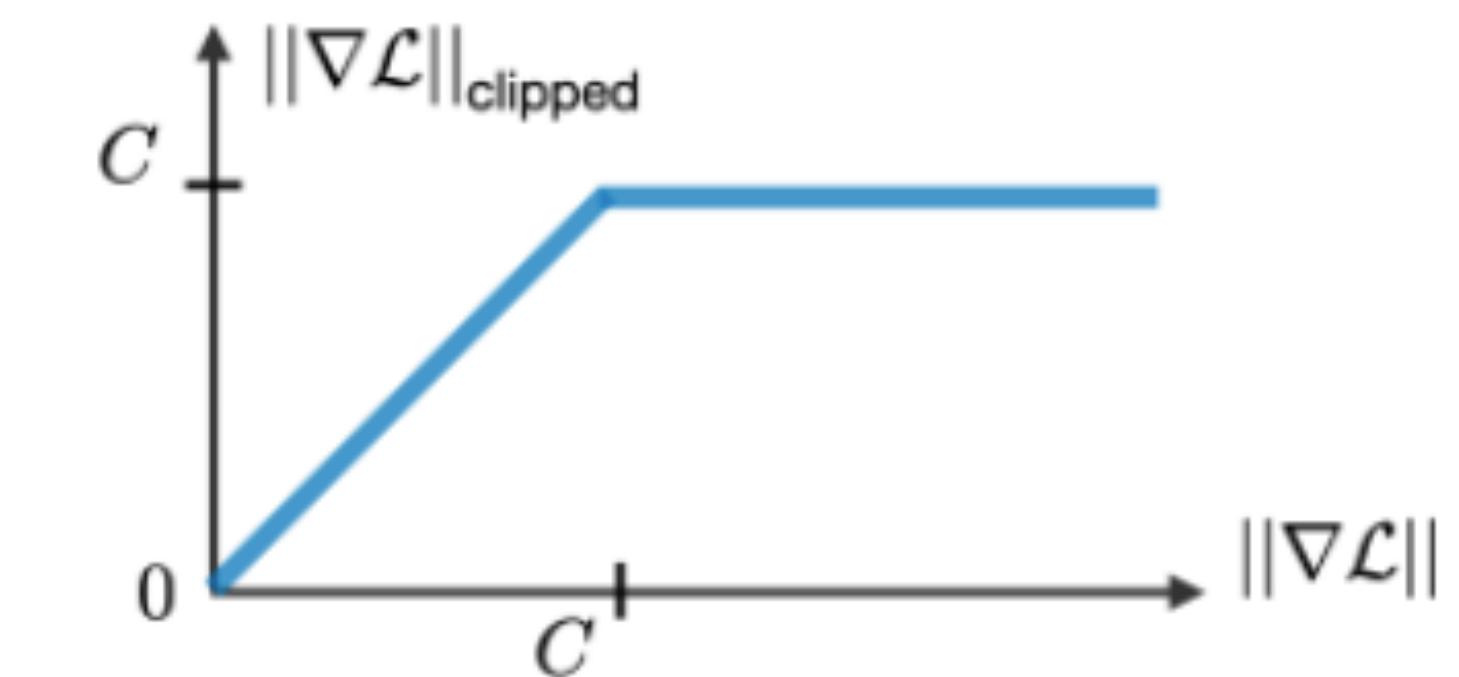
$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^T \left. \frac{\partial \mathcal{L}^{(T)}}{\partial W} \right|_{(t)}$$

時刻が後の方から、時刻が前の方に伝搬させる

- 勾配消失/爆発の問題
  - 系列長が長くなるほど勾配の値が不安定になる
  - 勾配爆発には勾配クリッピング（閾値を超えたならその閾値にする）が使われる
  - 一方、勾配消失には対応できないので、構造自体の改善を考える



<https://medium.com/@ayushch612/vanishing-gradient-and-exploding-gradient-problems-7737c0aa535f>



勾配クリッピング

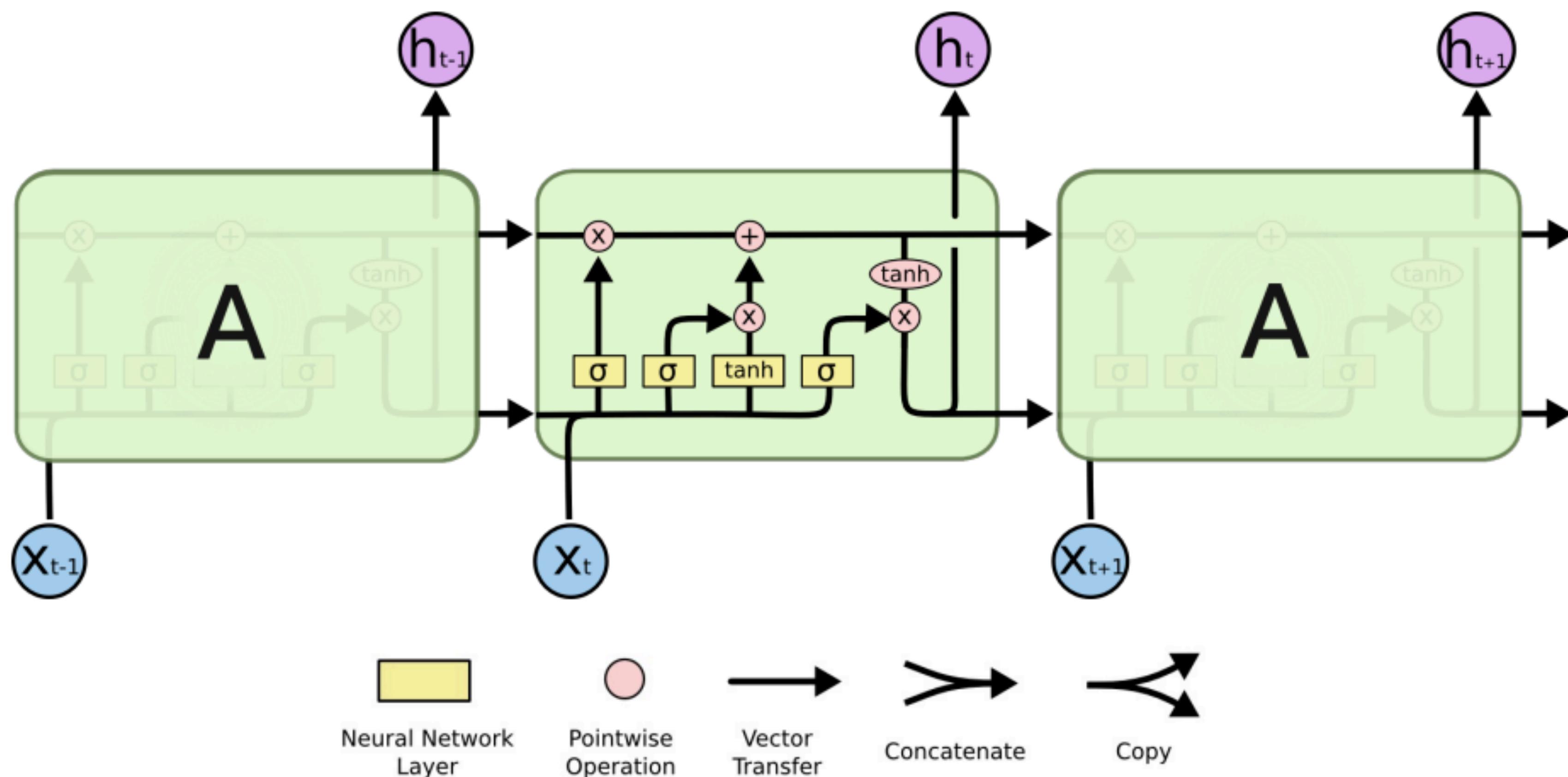
- Gated Recurrent Unit (GRU)と Long Short-Term Memory (LSTM)**

Characterization	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

Remark: the sign  $\star$  denotes the element-wise multiplication between two vectors.

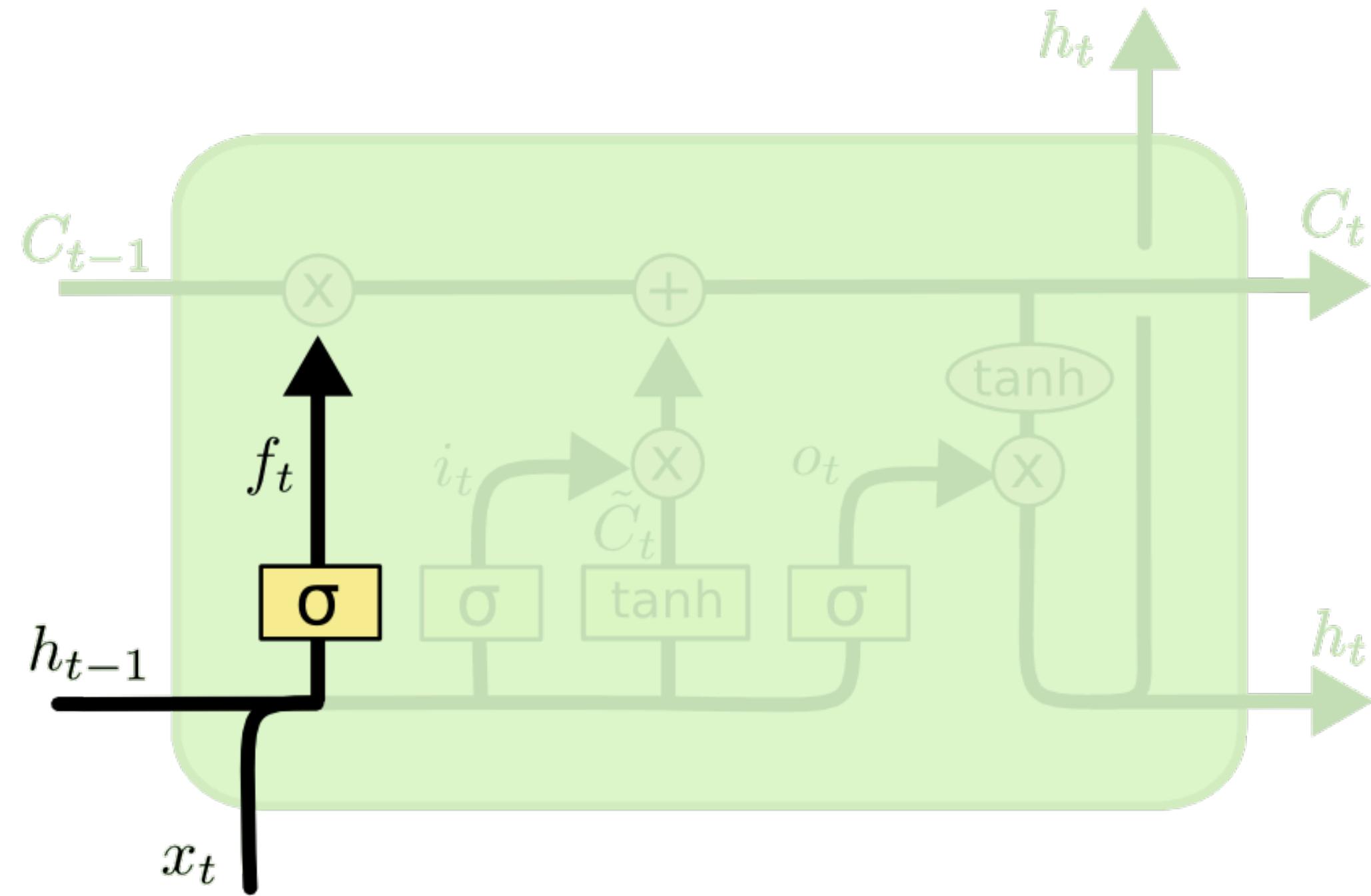
系列長の長いデータでも学習可能

- 4つのモジュールについて解説



## 不要な過去の情報を“忘れる”

- $f_t$  : ここが  $[0,1]$  の範囲をとり、前時刻の記憶セルの影響力を制御
- 0に近いほど前の情報を棄却し、1に近いほど保持する



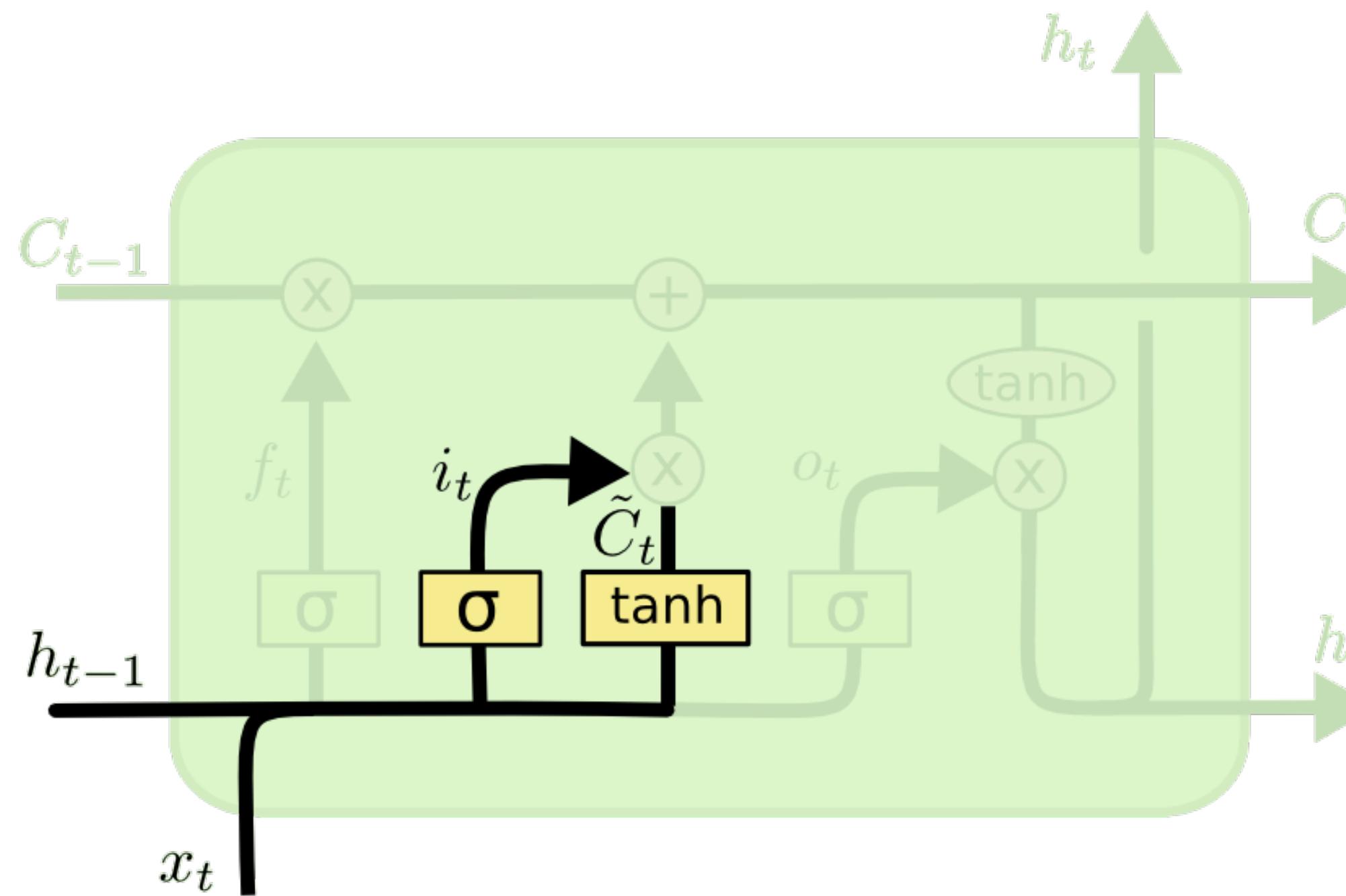
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

## ②現時刻の入力を利用する「新しい記憶セル」

15

現時点の新しい記憶を追加する

- $\tilde{C}_i$  : 現時刻の情報を加える
- $i_t$  :  $\tilde{C}_i$ の情報の価値を判断するための重み



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

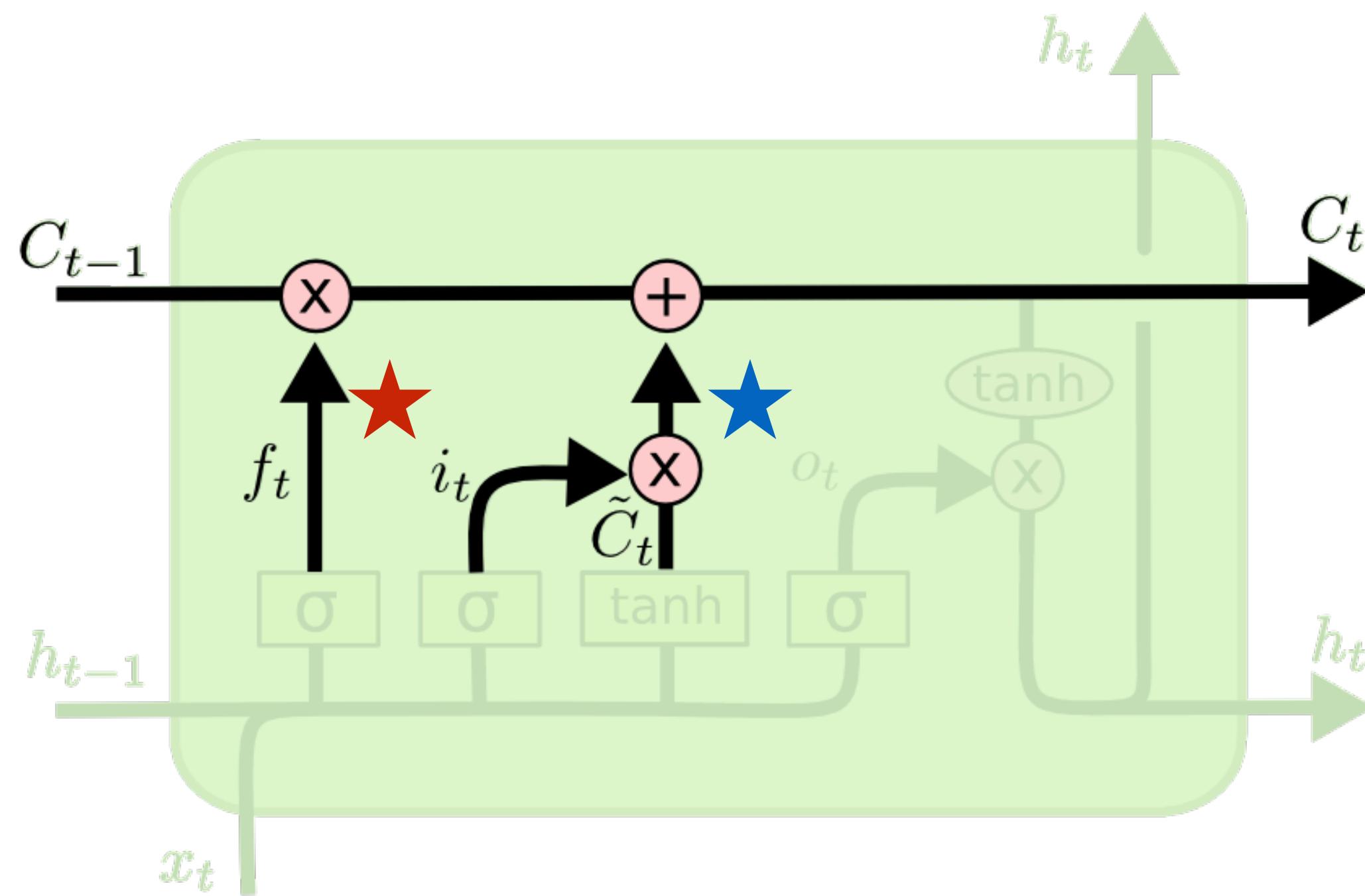
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

### ③記憶セルの更新

16

次の時刻に渡す記憶セルを計算する

- ★：忘却され残った前時刻の記憶
- ★：覚えるべき現時刻の記憶

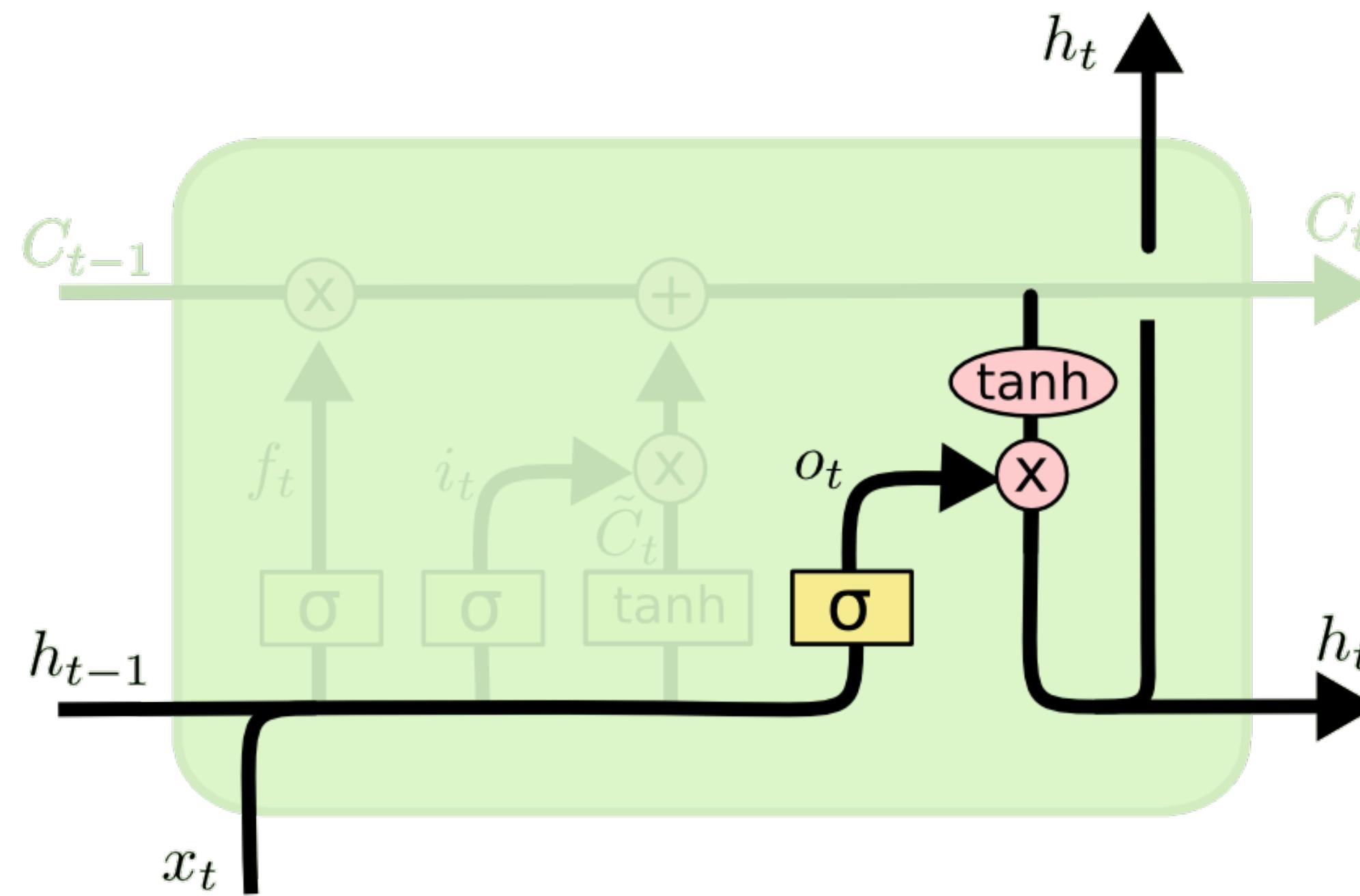


$$C_t = \underbrace{f_t * C_{t-1}}_{\text{忘却}} + \underbrace{i_t * \tilde{C}_t}_{\text{覚えるべき}}$$

\*は要素積

## メイン部分

- $O_t$  : 前時刻の隠れ状態 $h_{t-1}$ と現時刻の入力 $x_t$ のアフィン&活性化関数
- $h_t$  :  $O_t$ と活性化関数を通した記憶セル $C_t$ の要素積 → 次の時刻へ



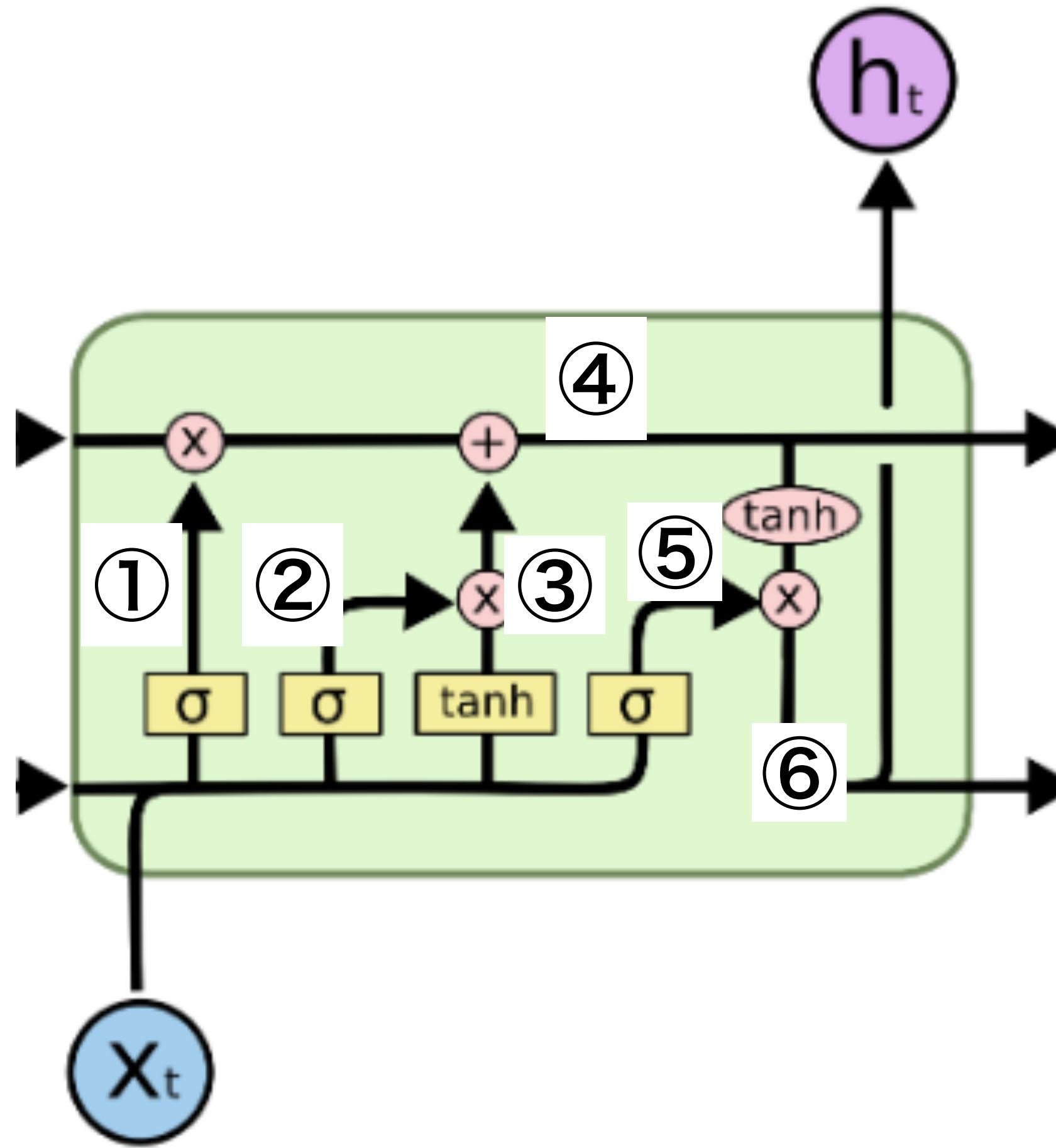
$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

## 全部で6つの式



GRUは省略しますが  
大体おんなじです



忘却ゲート ①  $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

新しい記憶セル  $\begin{cases} ② \quad i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ ③ \quad \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{cases}$

記憶セル更新 ④  $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

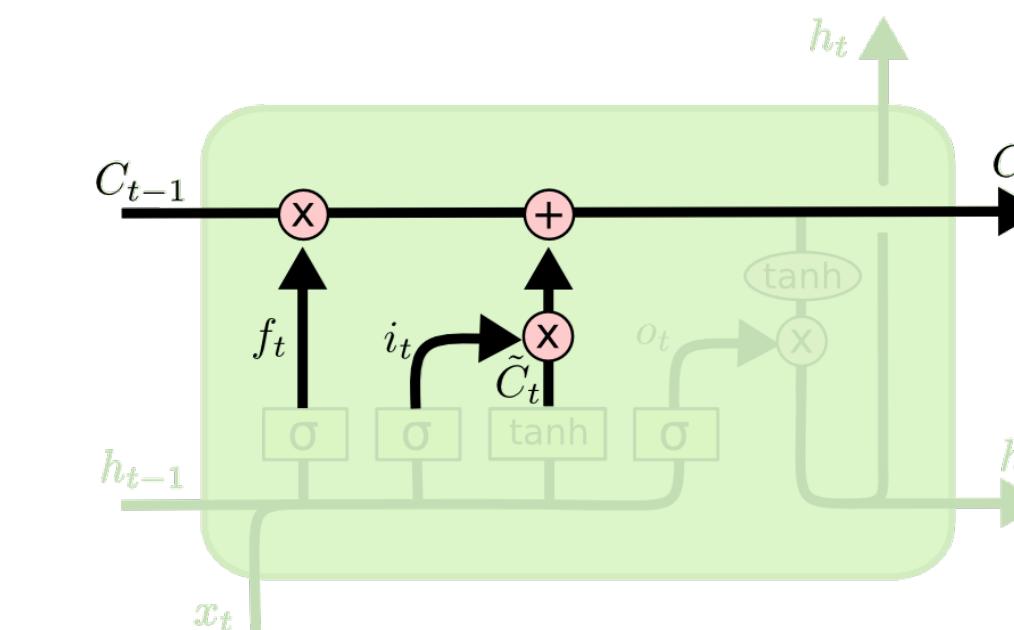
出力  $\begin{cases} ⑤ \quad o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ ⑥ \quad h_t = o_t * \tanh(C_t) \end{cases}$

パラメータ (+バイアス) は4種類

# なぜ勾配消失に強い？

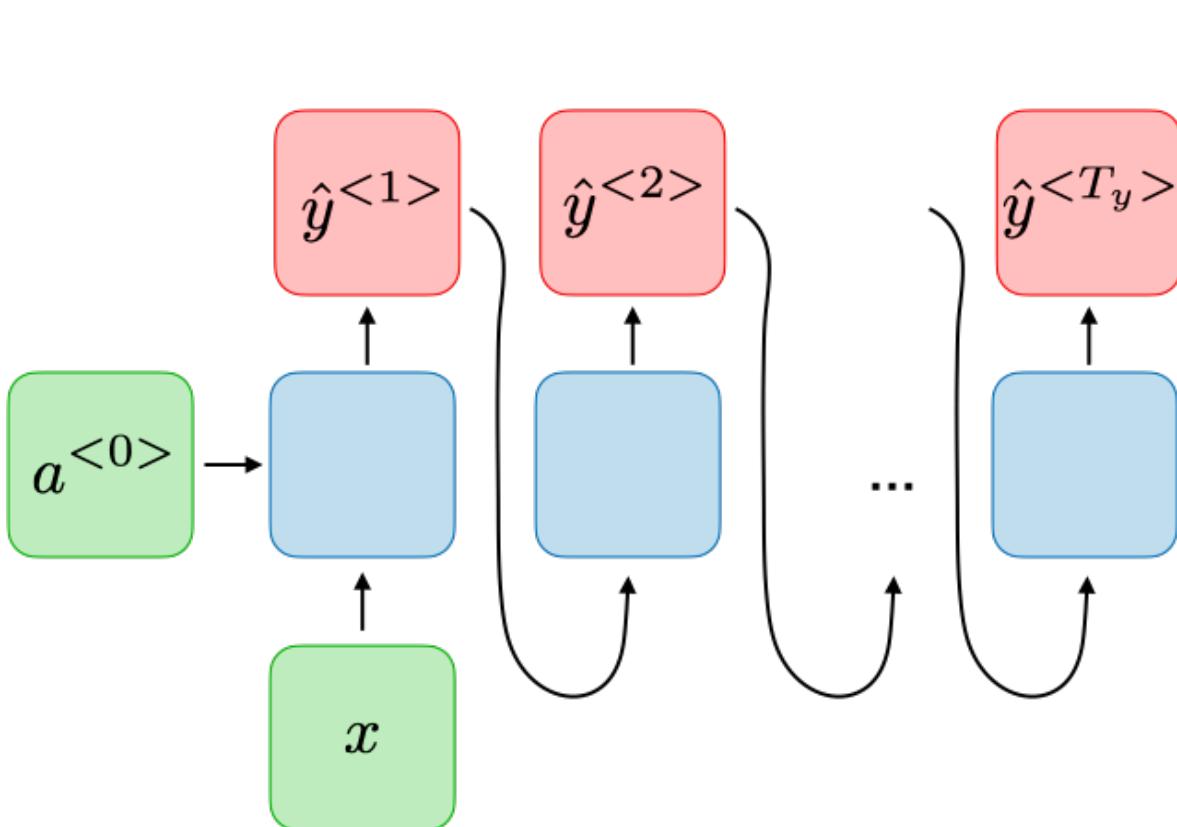
記憶セルが要素積で計算しているから

- RNNは重みを行列積で計算していた
  - →重みは各時刻で共有しているため、その逆伝播が乗算
  - 活性化関数tanhの微分（1以下）も乗算
  - 1以下の重みが時間ごとに乗算されて勾配爆発/消失を起こす
- LSTMでは重みを要素積で計算→要素ごとに勾配を計算
  - tanhもかかっていない
  - その時刻での $f_t$ の値（0,1）がそのまま前時刻の記憶セルに乗算
    - →忘れるべきでない、と判断された要素が小さくならずに長く残る

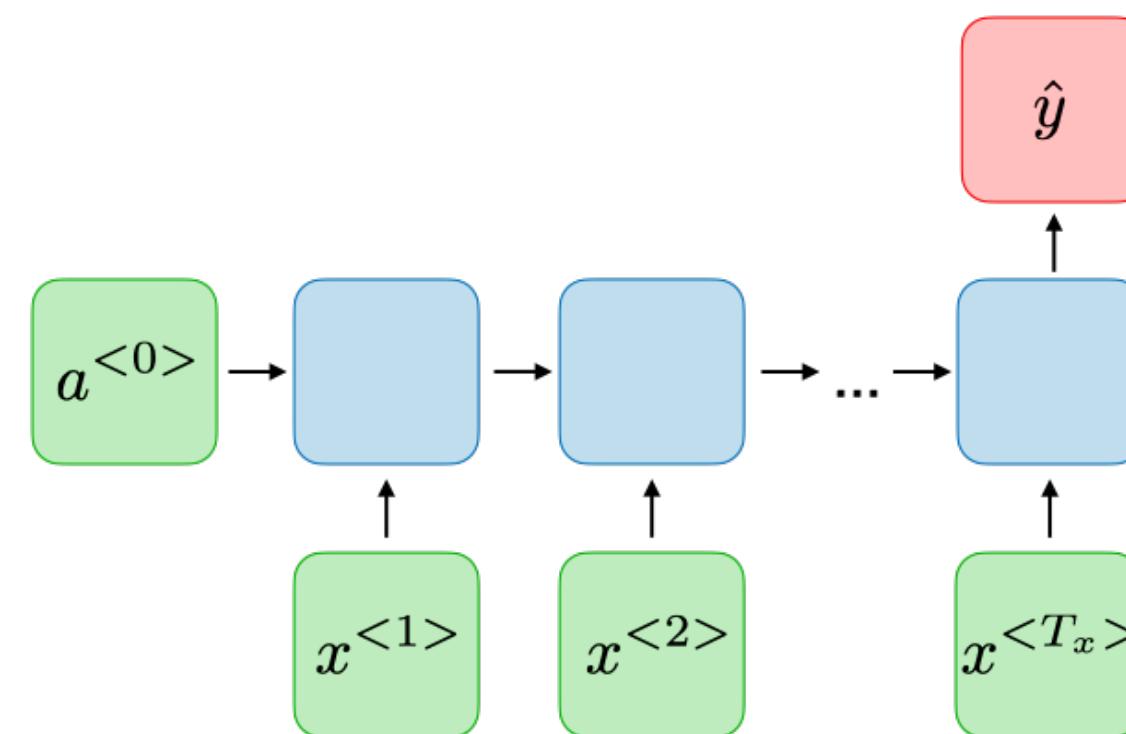


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

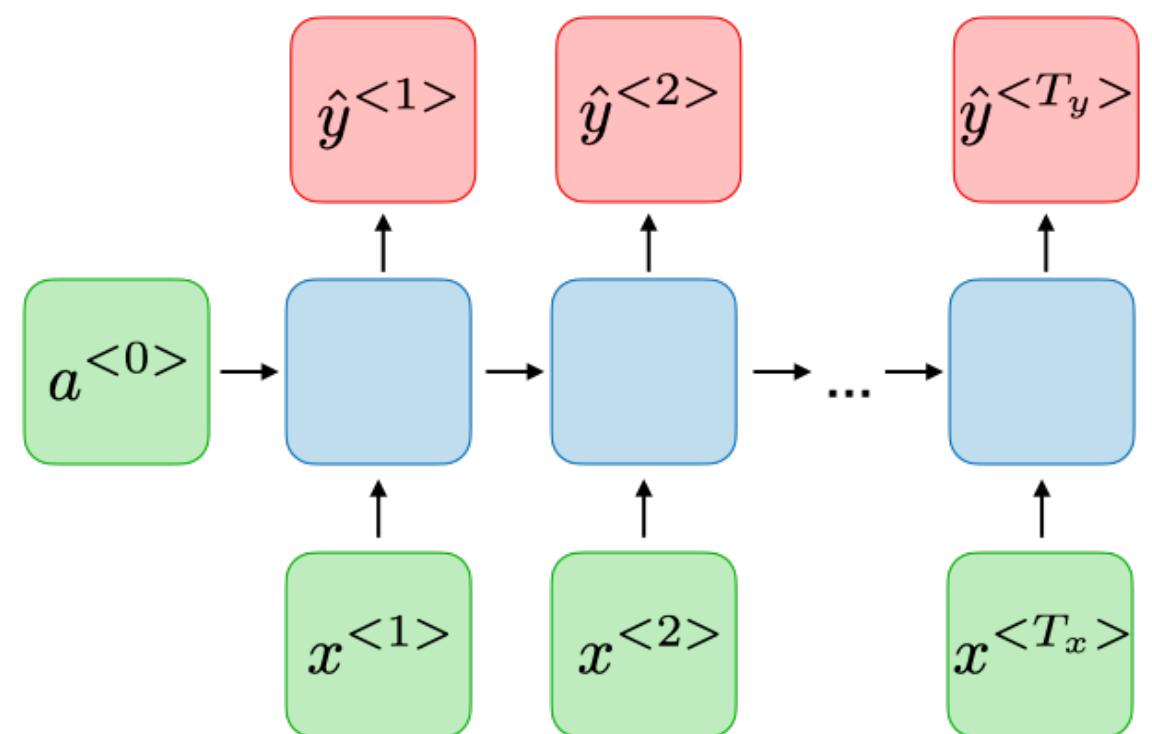
## 問題設定によって変えよう



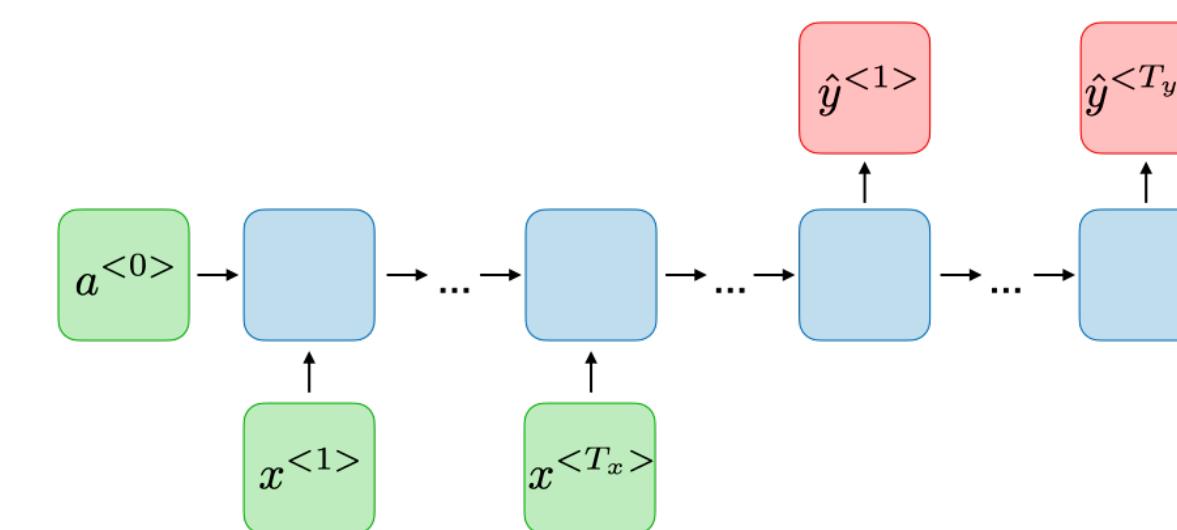
**One-to-Many**  
系列データを0から  
生成する  
(入力 $x$ は開始トークン)  
(音楽生成等)



**Many-to-One**  
時系列のデータから  
グローバルな情報を推定  
(文章感情推定等)



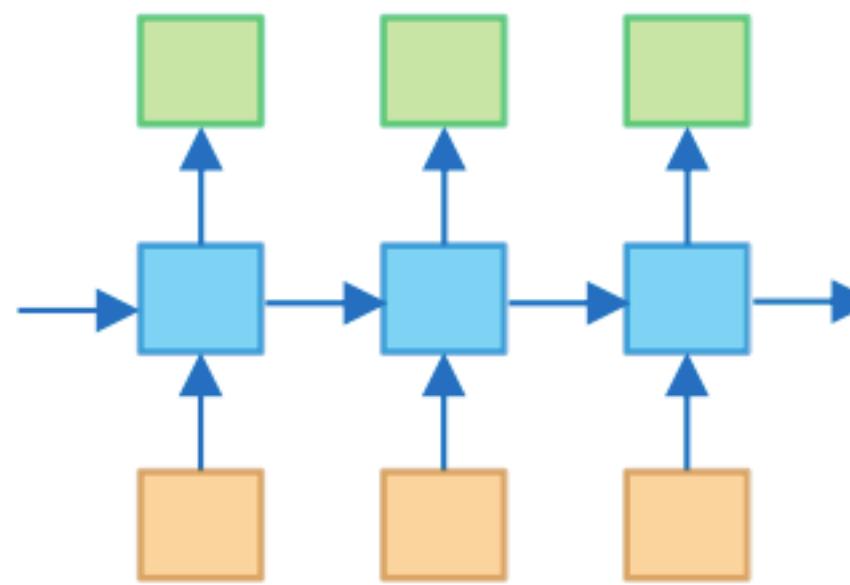
**Many-to-Many (対応あり)**  
入力と出力の時刻的に1対1  
対応する場合の推定/変換  
(単語の品詞推定等)



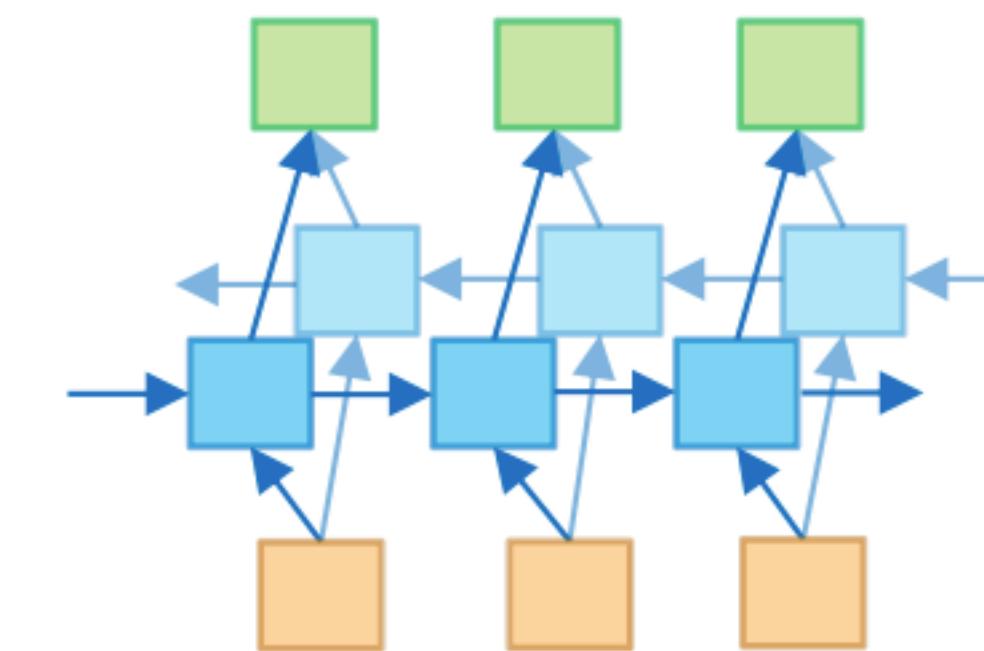
**Many-to-Many (対応なし)**  
いわゆるSeq2Seq.  
異なる2系列間の変換  
(機械翻訳等)

過去→未来に加え、未来→過去の情報伝播も考慮

- 未来の情報が識別に役立つ場合はRNNを双方向にしてもいい
  - 系列の最初から最後までが手に入っていることが前提。  
=リアルタイム処理には使えない
  - 前提を満たさなくとも使ってる論文をたまに見る



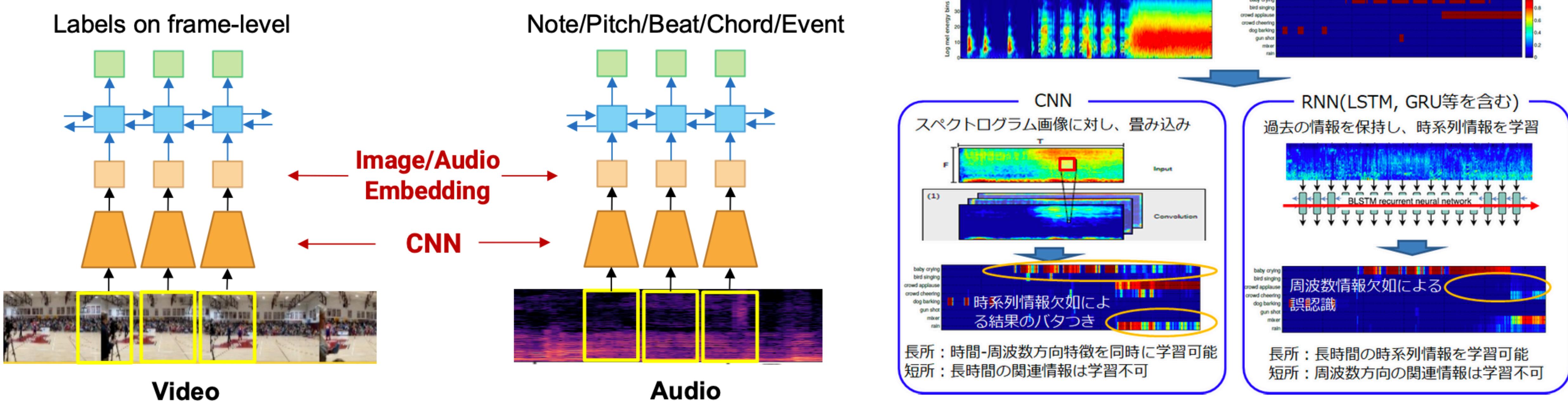
**Uni-directional RNN**  
(use past information  
only)



**Bi-directional RNN**  
(use both past and  
future information)

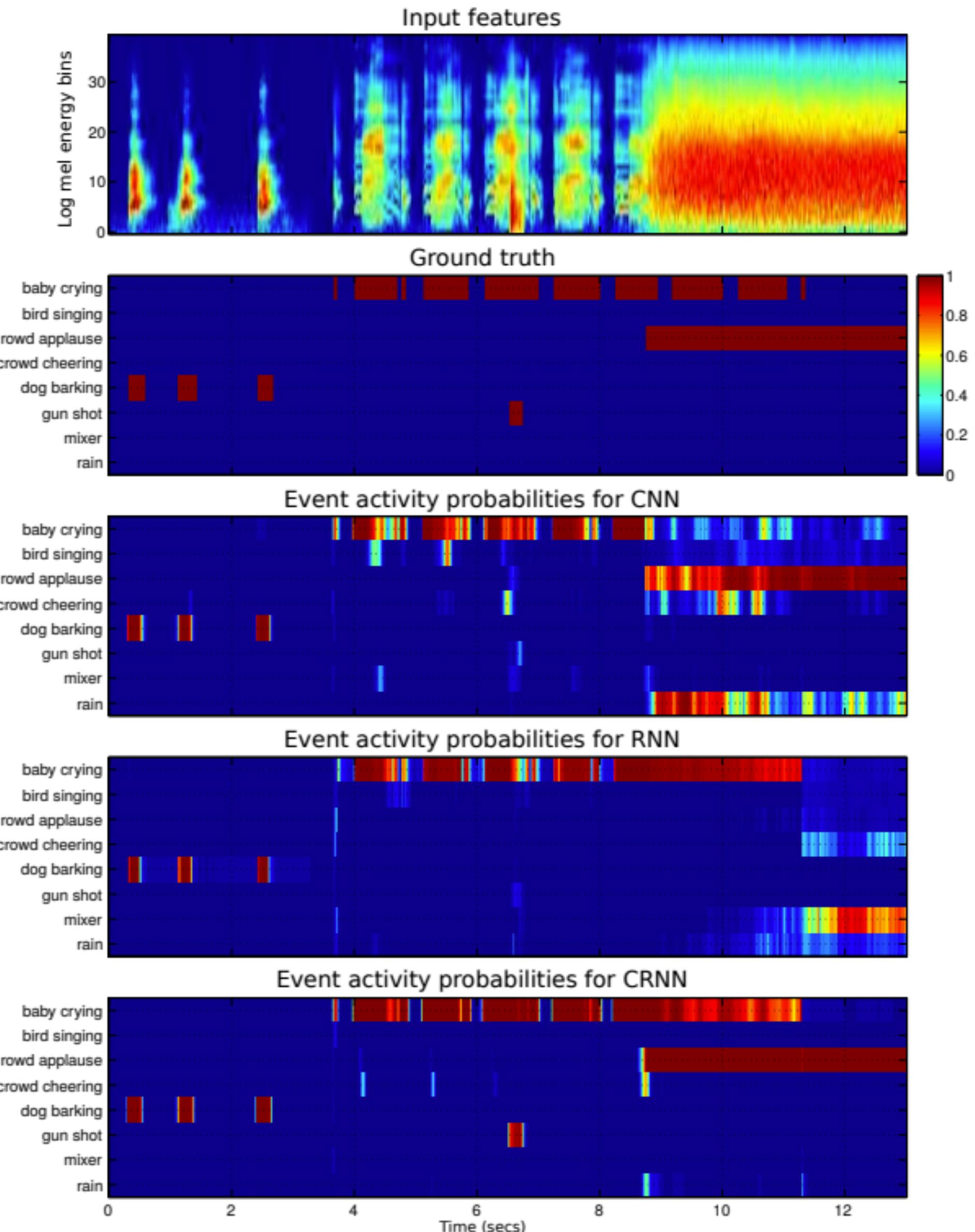
## CNNと組み合わせて音タスクで成功

- 動画・音など、その時刻の特徴抽出をCNNによって行う
- +その時間変化に関する特徴をRNNによって捉える



# CRNNはCNNとRNNのいいとこ取り

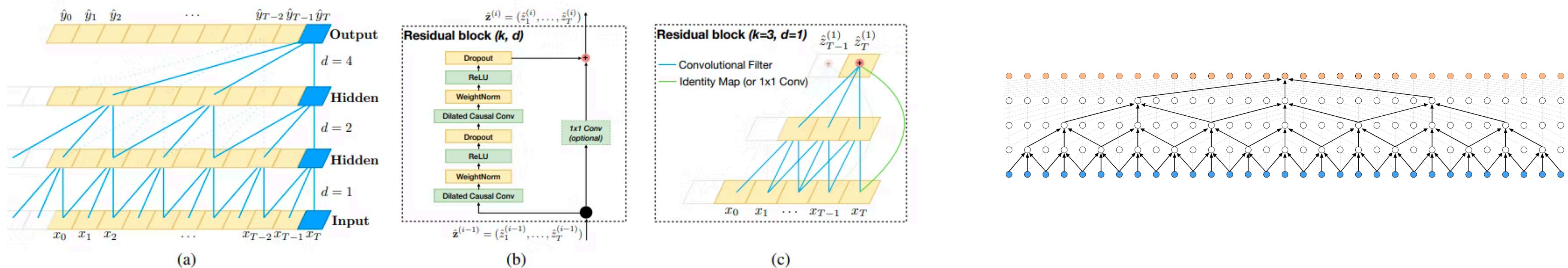
23



- CNNは周波数情報を、RNNは時間情報をよく捉える→**合わせ技**でいいとこ取り
- スペクトログラムを使った処理と相性○

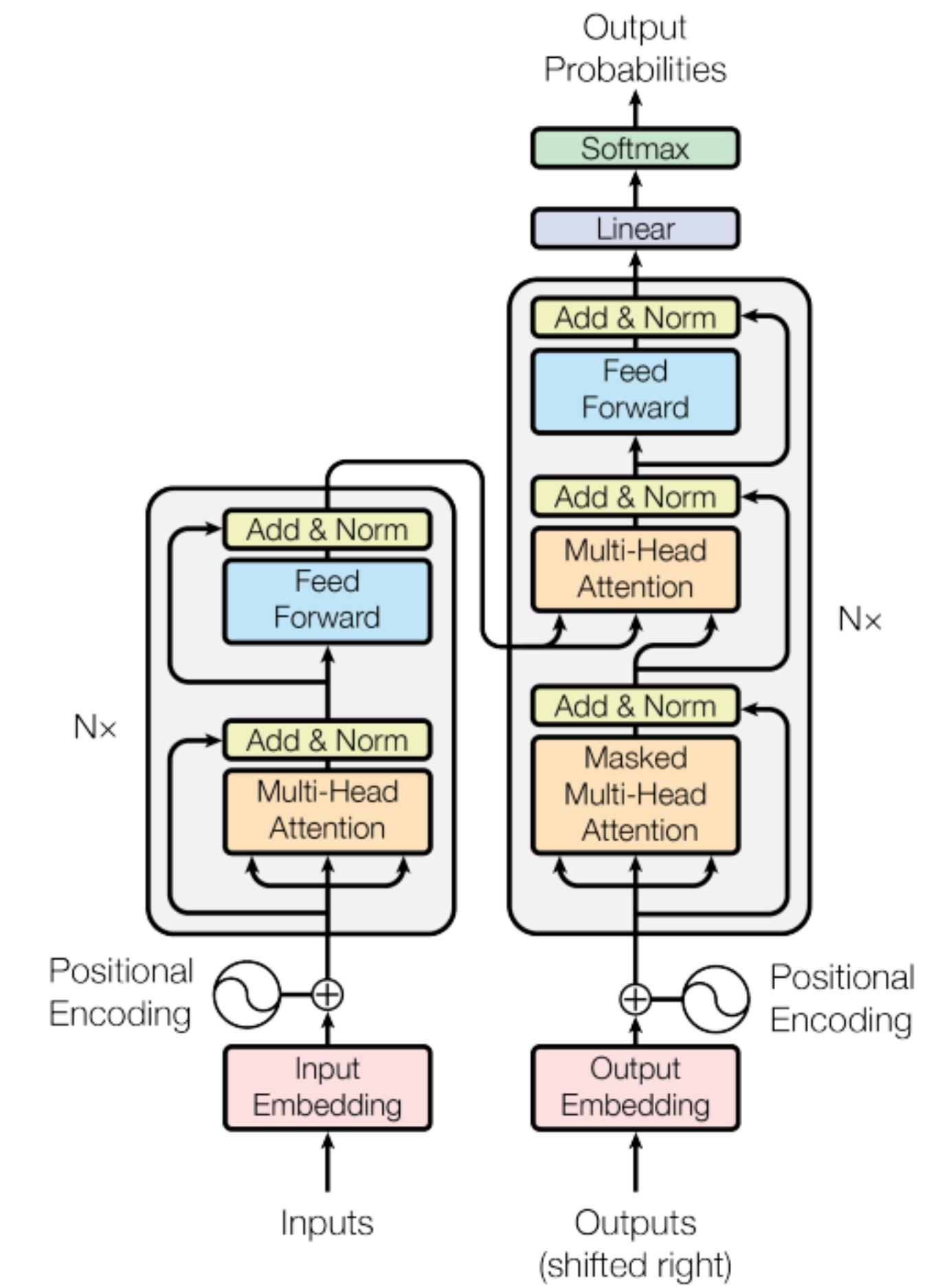
再帰構造をとっているため学習が遅め：速い代替モデルがある

- **Temporal Convolutional Network (TCN)**
- 時刻方向に1次元畳み込みを行う + 受容野をdilatedして拡大
- 学習が速いし、精度もLSTMよりよくなる場合もある（山本の実感）



再帰構造をとっているため学習が遅め：速い代替モデルがある

- **Transformer**
  - 「なんか最近きてるやつ」
  - 注意機構（Attention）に基づくモデル
  - CNNはカーネルの範囲内を、 RNNは系列の一つ一つを逐次に着目するが、 Transformerは系列全てを一気に着目し、 Attentionによって注意するポイントを決める



詳しくは未来のTransformer回に

Vaswani et al. Attention is all you need. NeurIPS 2017

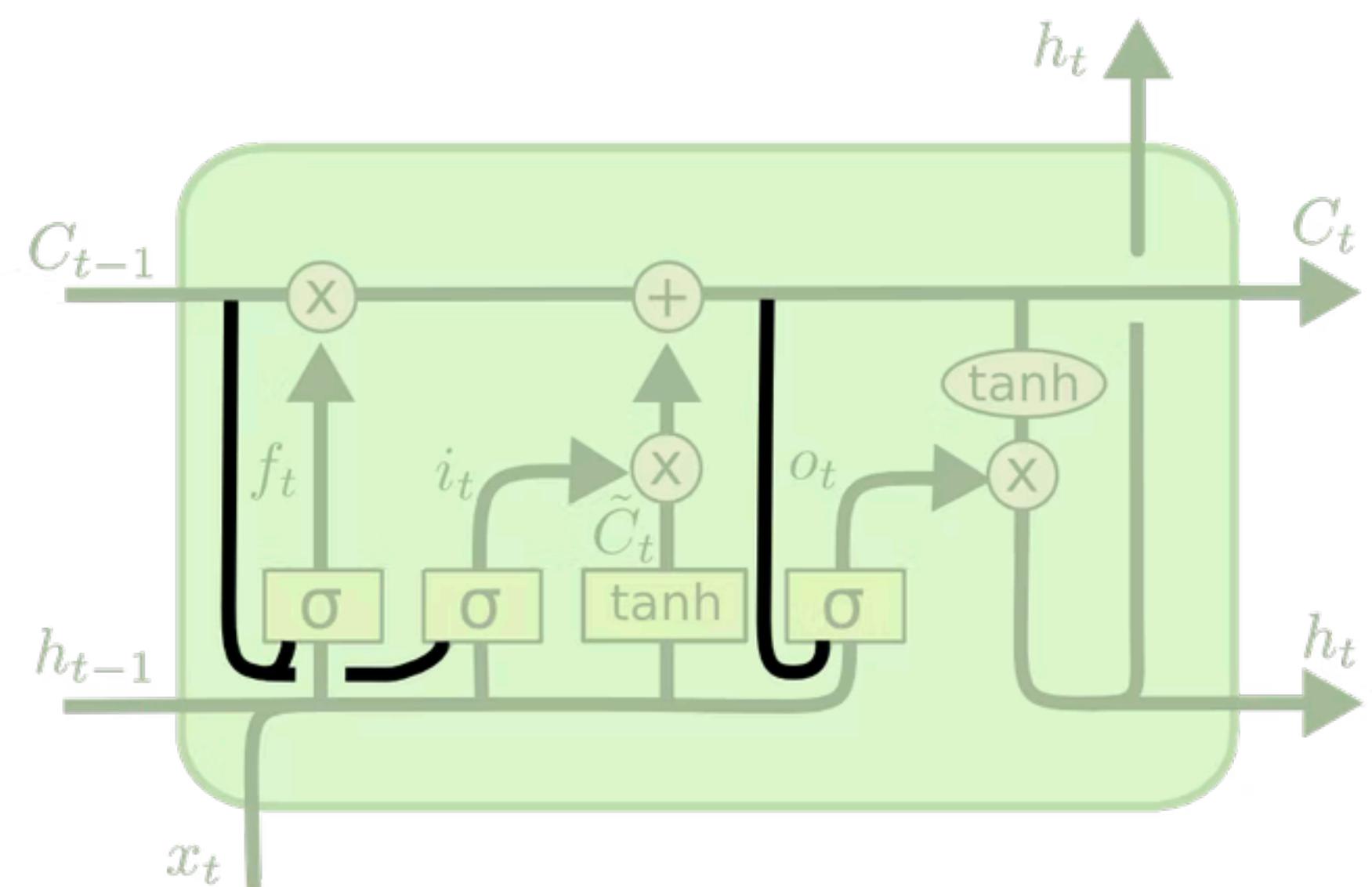
- 系列データにはRNNが効果的
- 長い系列データにおける勾配消失の問題を解決するゲート付きRNN
- 入出力によるRNNのセッティング
- CNNを組み合わせたCRNN
- 最近の系列データ処理でRNNを駆逐しつつあるモデルの紹介

補足

# LSTM の亜種 [Gers & Schmidhuber 00]

28

- Peephole connectionという、 $f, i, o$ に記憶セルの値を入力する機構を追加

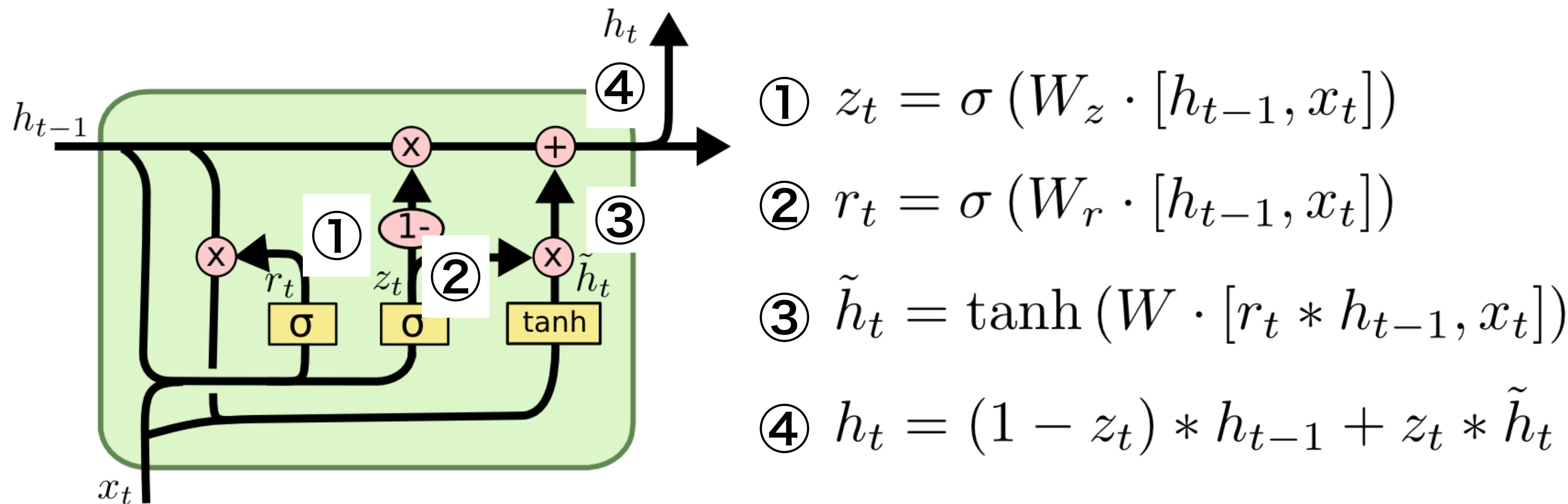


$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

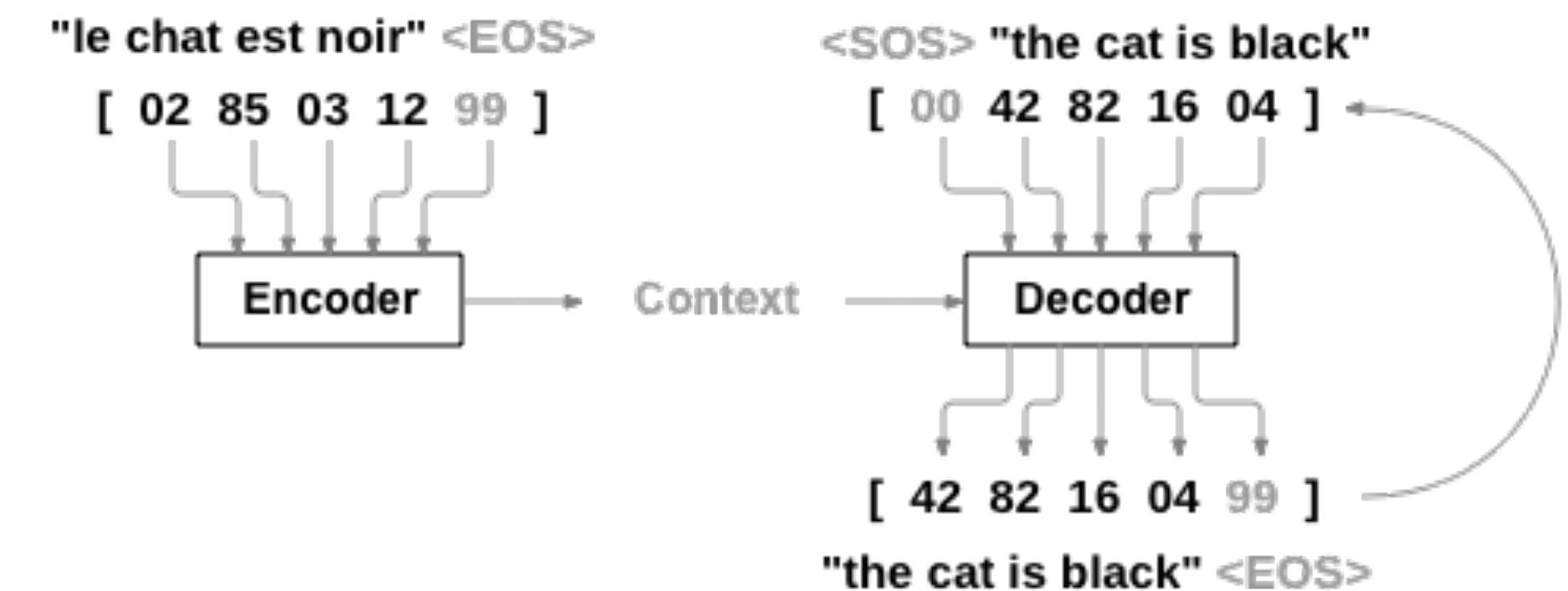
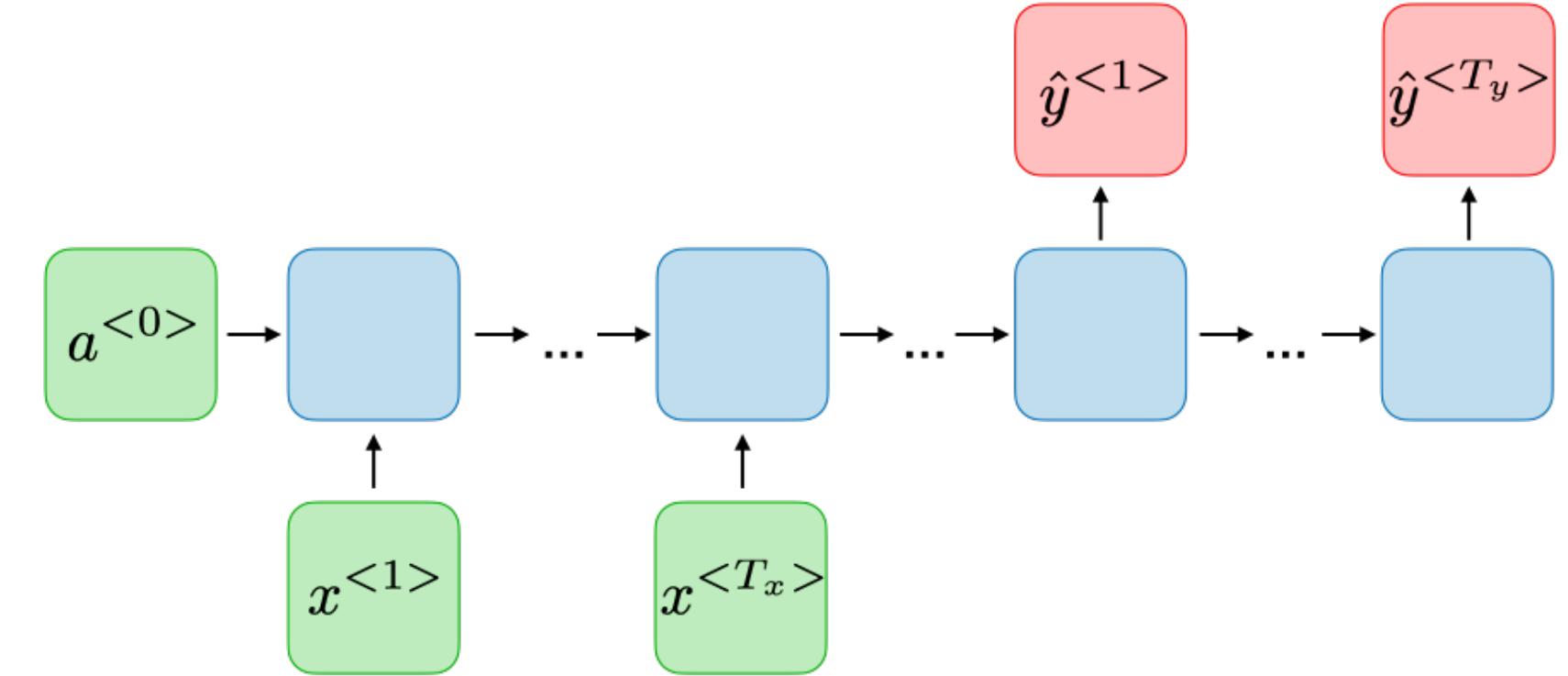
- LSTMの記憶セルを消去
  - ④の第1項がLSTMの忘却ゲート, 第2項が記憶の更新を担う
  - パラメータ数が減る. より小さなデータセットに効果的



# Seq2Seq (Many-to-Many RNN)

30

- 入出力が1対1対応しない系列の変換でのモデル
  - エンコーダー・デコーダーモデル（一般化）
- 実装上では2つの部分に分ける
  - エンコーダー -> 入力からコンテキストを得る
    - many-to-one
    - デコーダー -> コンテキストから出力を得る  
(自己回帰形式)
    - one-to-many

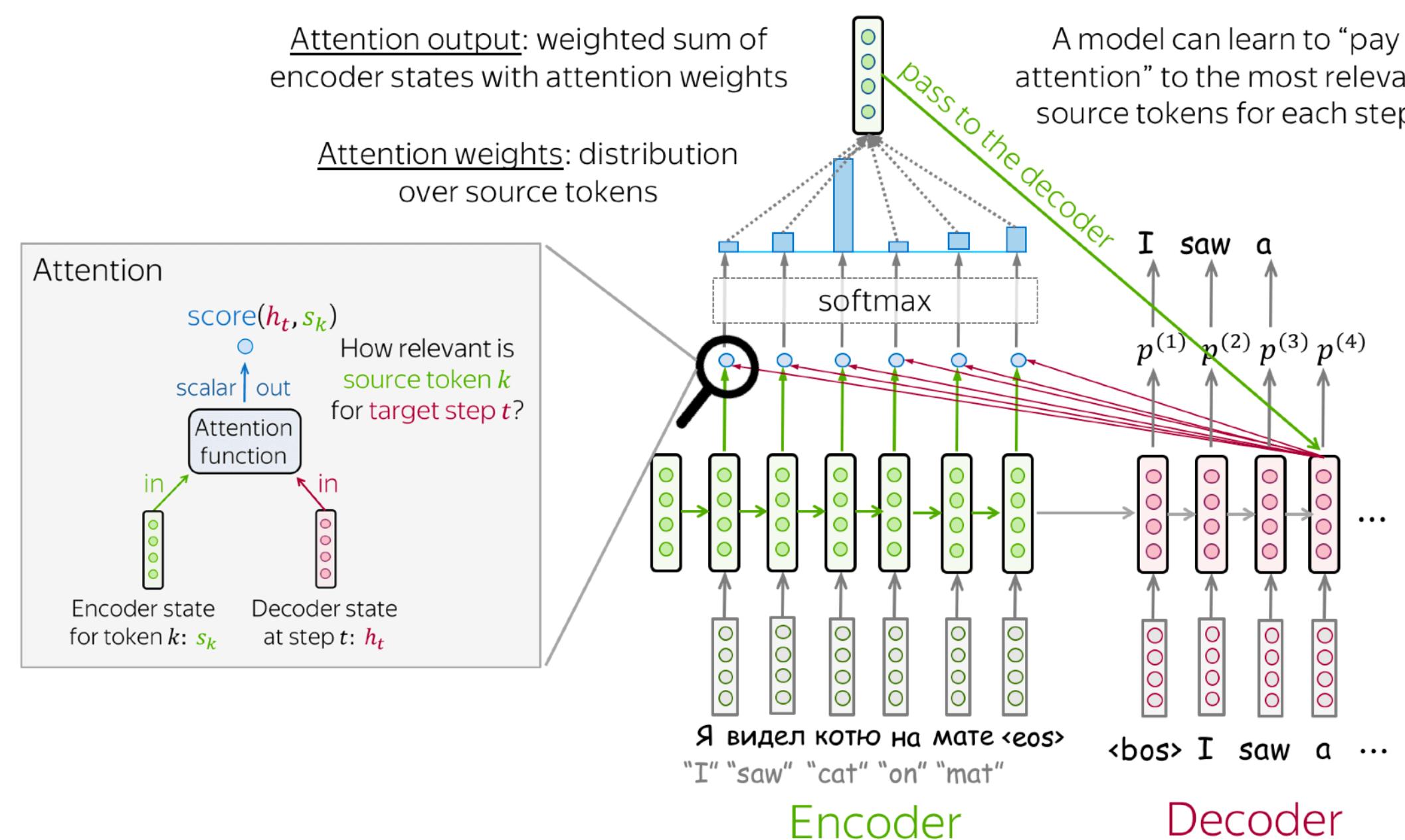


[https://pytorch.org/tutorials/intermediate/seq2seq\\_translation\\_tutorial.html](https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html)

# Attention付きSeq2Seq

31

- Seq2Seqでは入力を全て固定長のベクトルにしていた
- →入力の長さが異なるが全て同じ情報量を保ち,
- →Attention（注意機構）を用いて、入力のどこに着目するかを決定

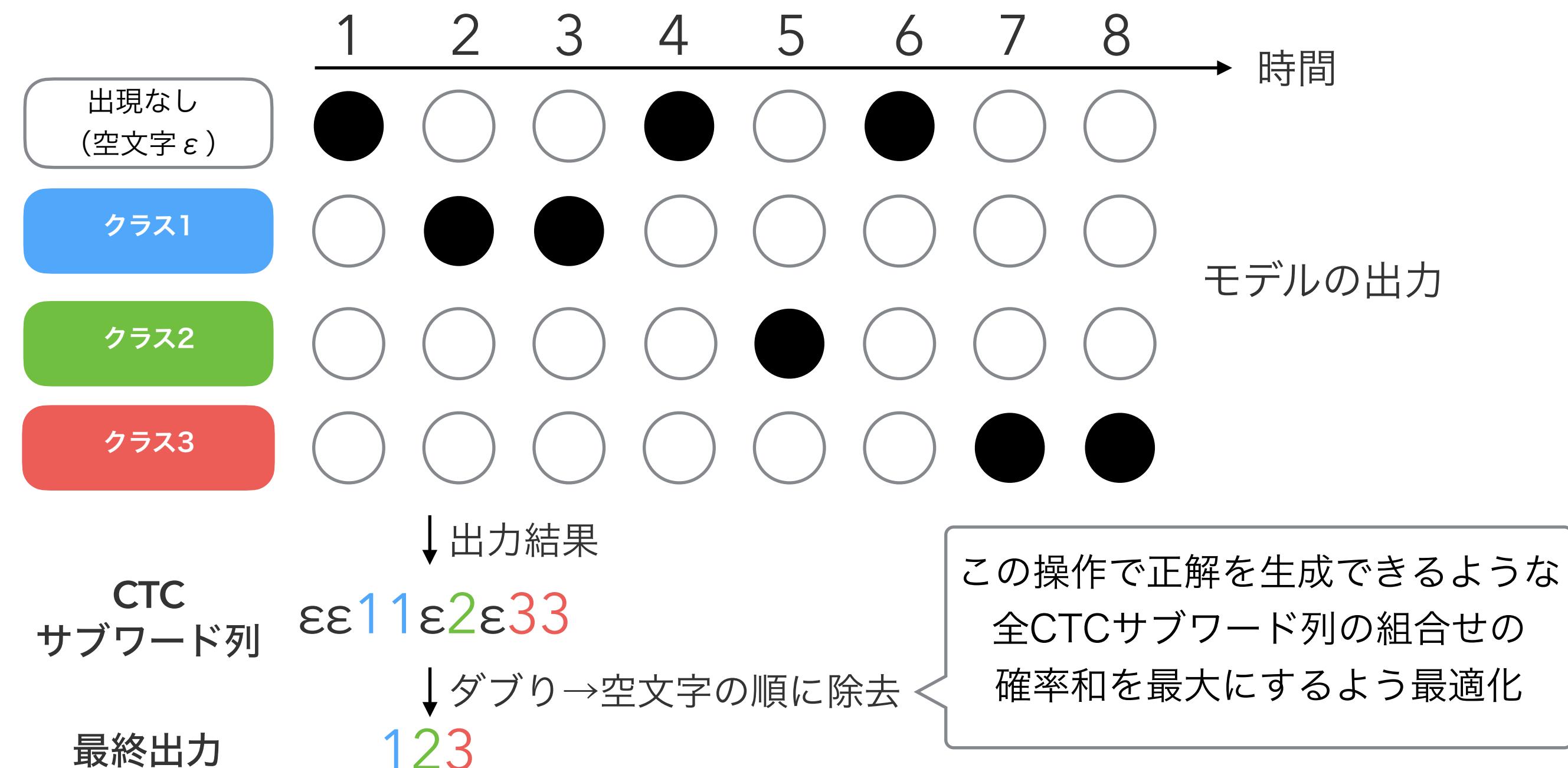


[https://lena-voita.github.io/nlp\\_course/seq2seq\\_and\\_attention.html](https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html)

# CTC : Connectionist Temporal Classification

32

- 長さの異なる時系列間の変換タスクでの、もう一つの手法
- 音声認識、OCRなどで用いられる



- Graves et al. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks.

In Proceedings of the 23rd international conference on Machine learning (pp. 369-376).