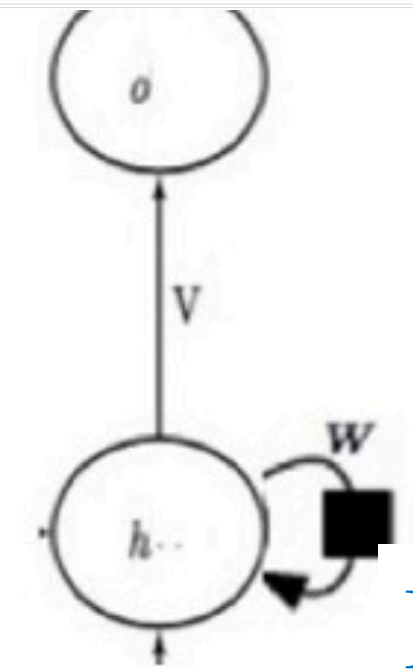
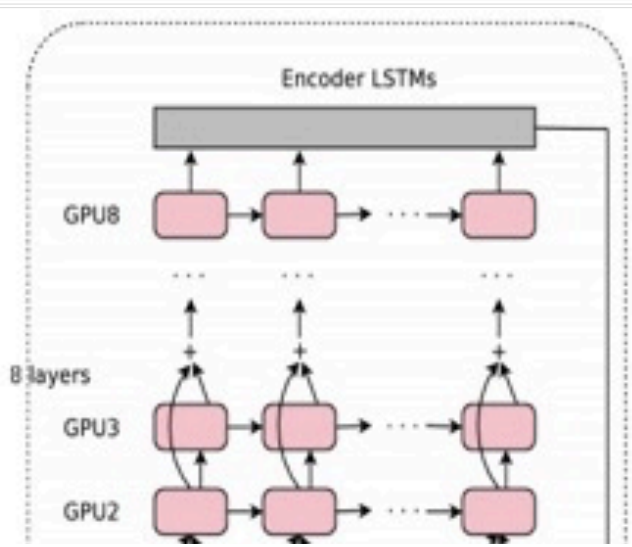
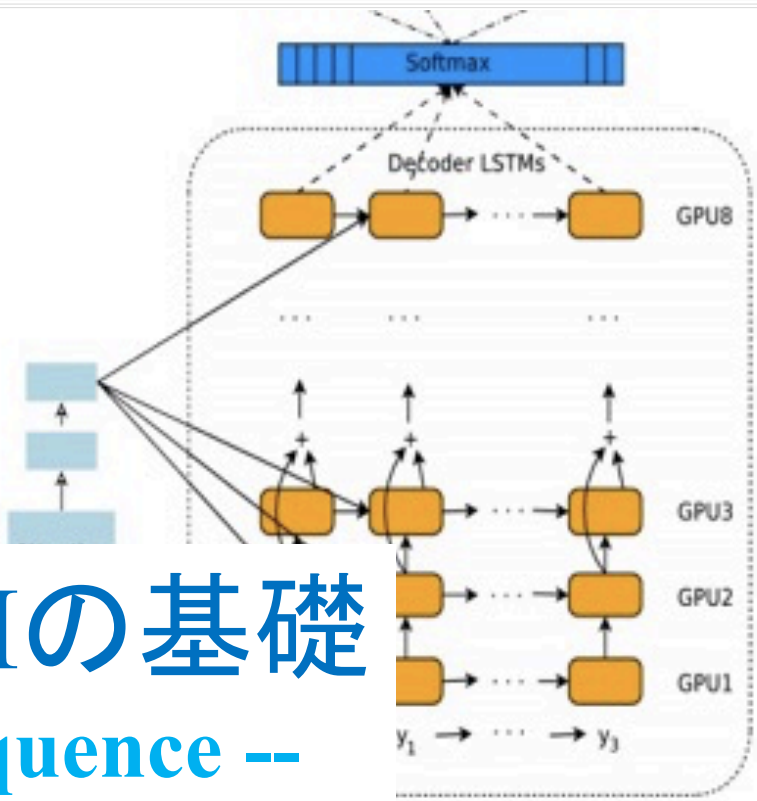
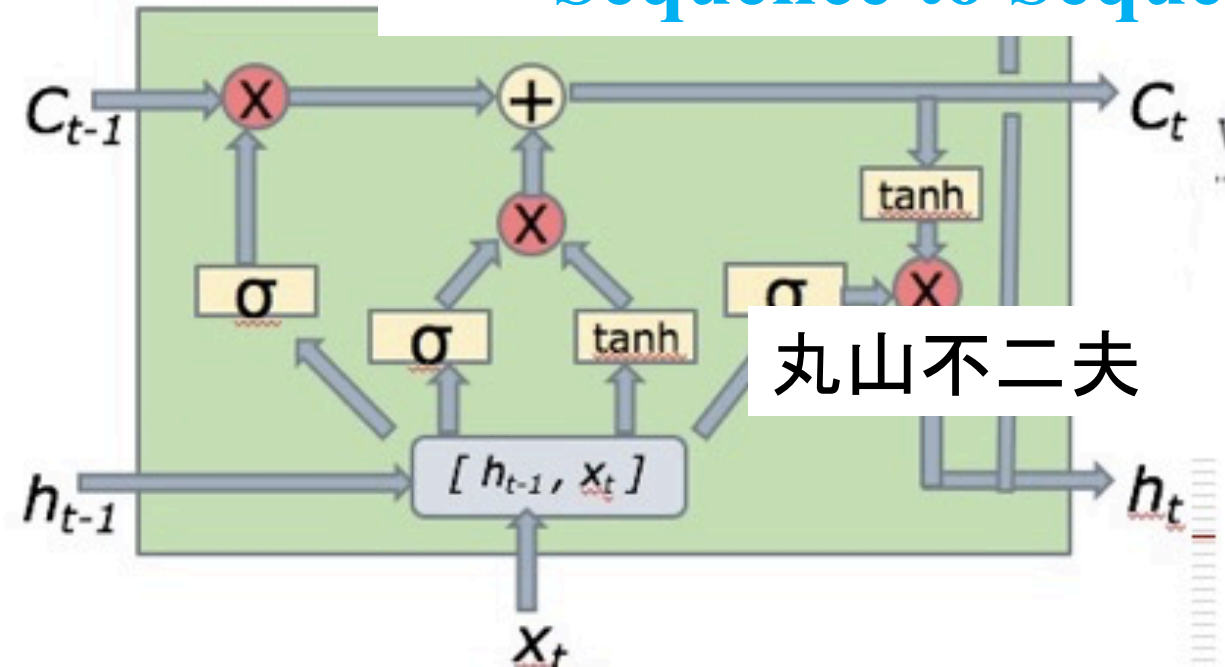


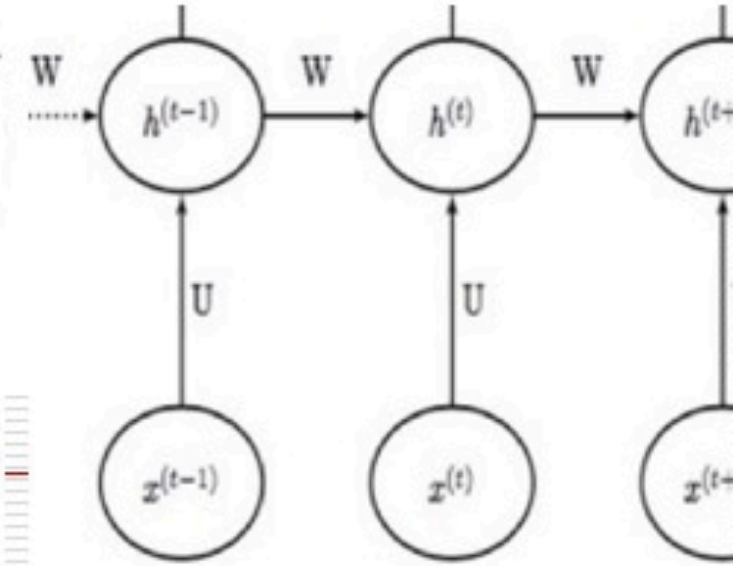




RNNとLSTMの基礎

-- Sequence to Sequence --





丸山不二夫

I didn’t pay attention to it at all, to be perfectly honest. Having been trained as a computer scientist in the 90s, everybody knew that AI didn’t work. People tried it, they tried neural nets and none of it worked.

World Economic Forum’s Annual Meeting in Davos 2017 Sergey Brin

“English is not a regular language”
As for context-free languages,
“I do not know whether or not English is itself literally outside the range of such analyses”

-- Chomsky

We believe that this is possible only because our shared architecture enables the model to learn an interlingua between all these languages.

-- Google Neural Machine Translation System

はじめに

- 科学でも技術でも、その発展には、飛躍がある。もちろん、飛躍にも小さいものから大きいものまでいろいろあるのだが。面白いことに、何かが共鳴するように、いろんな飛躍が、ある時期に集中して起きることがある。
- 人工知能の分野では、2012年がそうした時期であった。ImageNetの画像認識の国際的コンテストで、CNNを使ったAlexNetが圧勝し、Googleの巨大なシステム DistBeliefが、教師なしでも猫の認識に成功する。それが、2012年だ。
- この時から、人工知能ブームが起き、技術的には、ニューラル・ネットワークを用いたDeep Learning技術が花形となる。そして、そうした技術的な達成は、ただちに、ビジネスの世界で応用を見つけ出していく。

はじめに

- 今また、2012年のブレークスルーに匹敵する目覚ましい飛躍が、AIの世界で起きようとしている。昨年11月にサービスが始まった、Googleの「ニューラル機械翻訳」がそのさきがけである。そこで使われている技術は、RNN/LSTMと呼ばれるものである。
- Googleの新しい機械翻訳のシステムは、LSTMのお化けのような巨大なシステムなのだが、 RNN/LSTMの基本原理を理解することは、誰にでもできると思う。また、そうした理解が、新しいAI技術にキャッチアップしようとするIT技術者には、必要だと考えている。
- 今回は、図を多用した。IT技術者の中で、 RNN/LSTM の基礎の理解が進むことを願っている。

Agenda

- Part I　RNNの驚くべき能力について
- Part II　RNNとは何か？
- Part III　LSTMの基礎
- Appendix　各Frameworkでの実装を見る

Part I

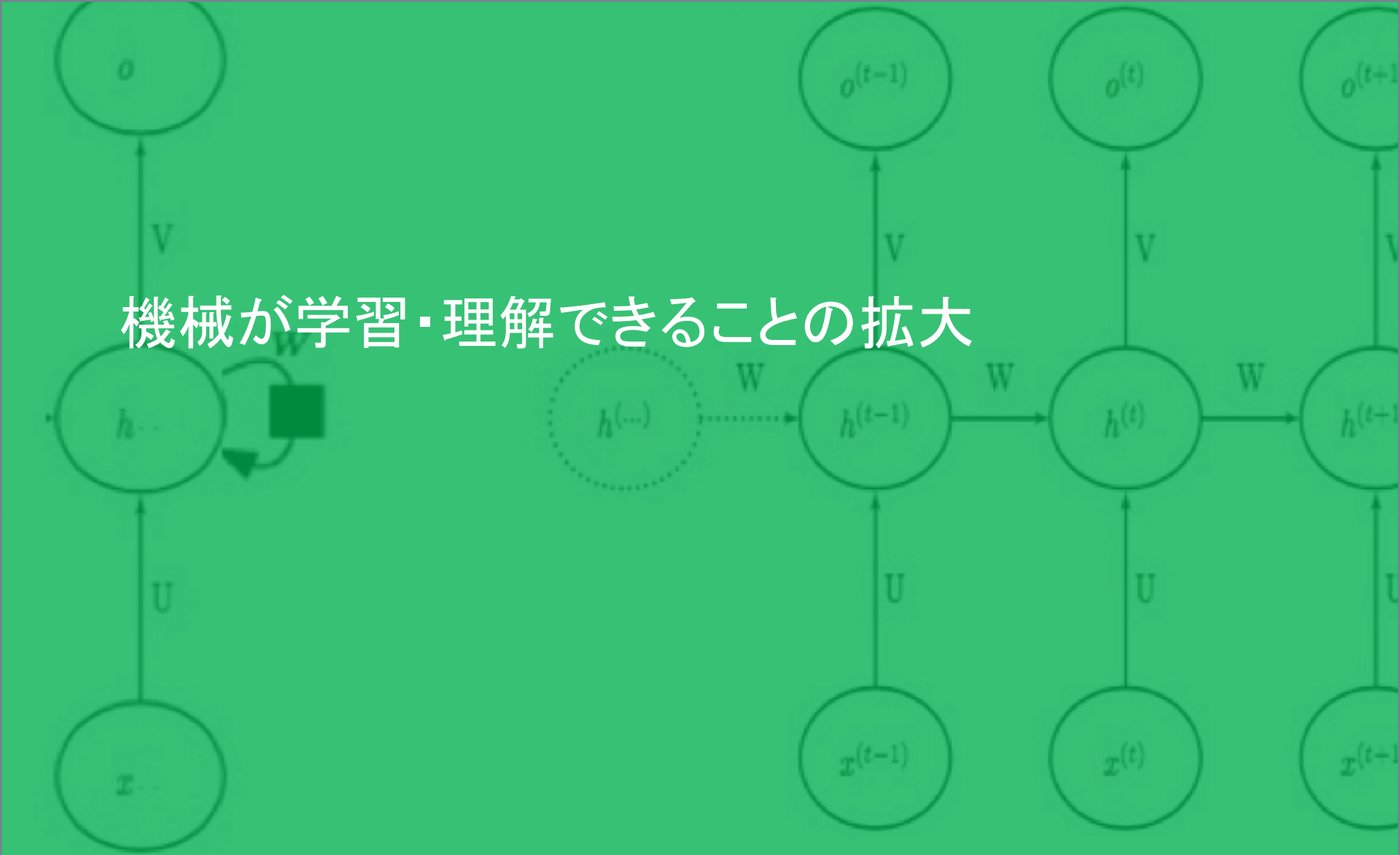
RNNの驚くべき能力について

Part I

RNNの驚くべき能力について　Agenda

- 機械が学習・理解できることの拡大
- RNNの能力について Sepp Hochreiter
- RNNによる文の生成 Ilya Sutskever
- RNNの驚くべき能力　Andrej Karpathy
- 文法の階層性 -- Chomsky Hierarchyについて
- Google ニューラル機械翻訳

機械が学習・理解できることの拡大



機械は、「何」を理解できるか？

- 機械は、「何」を理解・学習できるのか？　画像認識や自動運転技術のような、生物の感覚・運動系の能力の機械での対応物については、機械の能力の評価は、比較的容易である。
- 問題は、我々の認識能力が、具体的な感覚・運動系の能力にはとどまらないということ。人工知能研究の大きなターゲットの一つは、人間の言語能力である。この分野で、機械は、「何」を理解・学習できるようになったのだろうか？

機械は、「文法」を理解できるか？

- 言語能力の領域には、「意味の理解」をはじめとして、未解明の問題がたくさんある。ここでは、問題を限定して、機械は、自然言語のものであれ、人工的に構成されたものであれ、言語の構文規則である「文法」を、理解できるのかという問題を考えてみよう。
- 既に1990年代に、Hochreiterは、有限状態オートマトンで記述される構文規則を、 RNNが、理解・学習できることを発見していた。
- 2011年、 Ilya Sutskever らは、5億文字ものテキストをRNNに学習させ、学習したテキストに近い文体の文章を生成して見せた。文法にはかなっているように見えるが、意味をなさない文章だったが。

LaTexやCの文法は、理解・学習できた
自然言語の文法理解へ

- 2015年、Andrej Karpathy は、Stack Theoryの膨大なドキュメントをRNNに学習させ、一見すると数学の論文に見える文書を生成してみせる。彼は、また、 Linuxの膨大なソースコードをRNNに学習させ、Cのコード（に見えるもの）を生成して見せた。これは、機械が、文脈自由文法であるLaTexやC言語の構文規則を、理解・学習できることを示すものだ。
- 2016年の、Googleニューラル機械翻訳は、これらの先行する機械の文法理解の能力に関する仕事を、さらに一歩進めるもののように、僕は考えている。
- ここでは、こうした流れを振り返ってみよう。

RNNの能力について -- LSTM原論文を読む

"Long Short Term Memory"

Sepp Hochreiter et al.

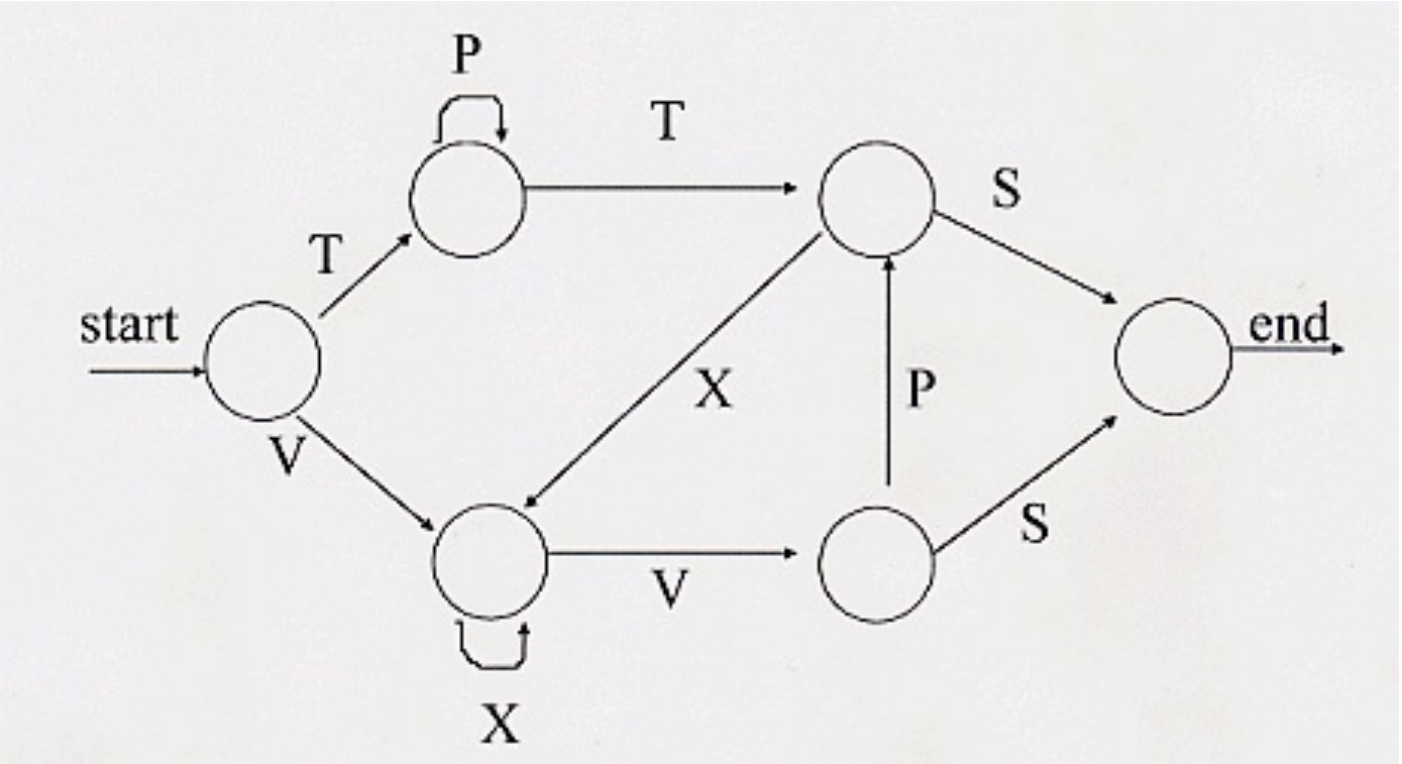
https://goo.gl/sDAq81

1997年

Hochreiterの90年代の発見

- LSTMの提案者であるHochreiterは、1990年代に、既に RNNが、極めて高い、認識能力を持つことを発見していた。LSTMの原論文に、詳細に、多数の実験結果が報告されている。ここでは、その中から、三つほどの例を紹介する。
- 一つは、有限オートマトンの生成する文字列の規則性を、LSTMが認識できるという実験結果である。これは、認知心理学の分野で、人間が持つ「人工的文法の学習（Artificial grammar learning）」能力として研究されていたものである。 https://goo.gl/VgvboK
- もう一つは、LSTMが、掛け算を学習できるという実験である。ニューラルネットが、任意の関数の近似が可能であることは知られていたのだが、これはこれで興味ふかい実験である。
- 最後は、あるルールに従って、時間的に離れたところで起きるイベントのパターンを、LSTMが学習できるという実験である。これは、人間でも手こずりそうかもしれない。
- いずれの実験も、膨大な学習の繰り返しが必要である。

人工的な文法の学習
実験 1: REBER GRAMMAR



例えば、次の文字列、**VXVS, TPTXVS** は、文法にあっているが、
VXXXS, TPTPS は文法的ではない。

実験 1: Embedded Reber grammar

LSTMは、この文法を学習できる！

method	hidden units	# weights	learning rate	% of success	success after
RTRL	3	≈ 170	0.05	"some fraction"	173,000
RTRL	12	≈ 494	0.1	"some fraction"	25,000
ELM	15	≈ 435		0	>200,000
RCC	7-9	≈ 119-198		50	182,000
LSTM	4 blocks, size 1	264	0.1	100	39,740
LSTM	3 blocks, size 2	276	0.1	100	21,730
LSTM	3 blocks, size 2	276	0.2	97	14,060
LSTM	4 blocks, size 1	264	0.5	97	9,500
LSTM	3 blocks, size 2	276	0.5	100	8,440

掛け算の学習
実験5: MULTIPLICATION PROBLEM

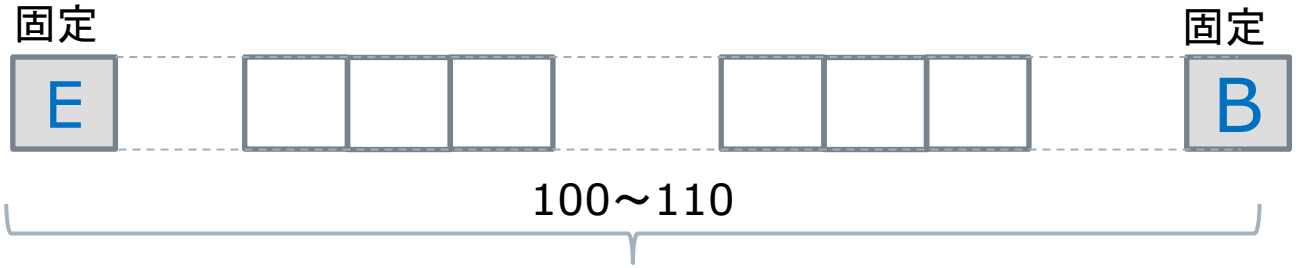
- **Task**. Like the task in Section 5.4, except that the first component of each pair is a real value randomly chosen from the interval [0; 1]. In the rare case where the rst pair of the input sequence gets marked, we set X1 to 1.0. The target at sequence end is the product X1 x X2.

T	minimal lag	# weights	n_{seq}	# wrong predictions	MSE	Success after
100	50	93	140	139 out of 2560	0.0223	482,000
100	50	93	13	14 out of 2560	0.0139	1,273,000

LSTMは、足し算（実験4）も掛け算も学習できる！

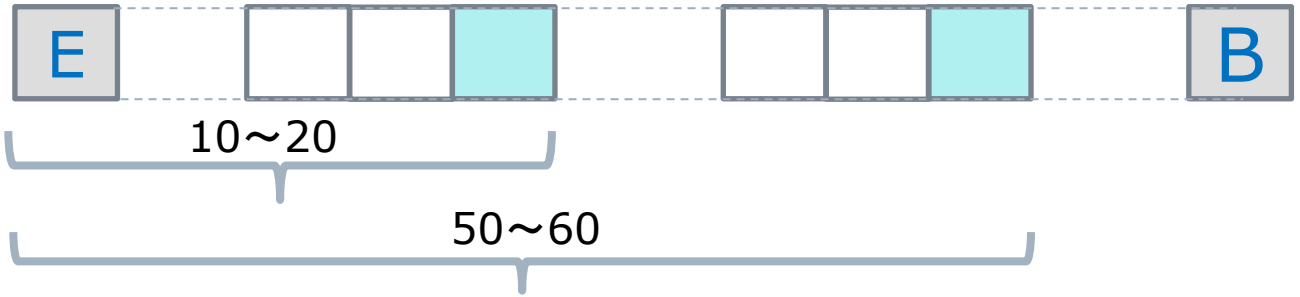
実験6:TEMPORAL ORDER

先頭にE、最後にBが入っている（固定）、長さ100～110（可変）の文字列がある。

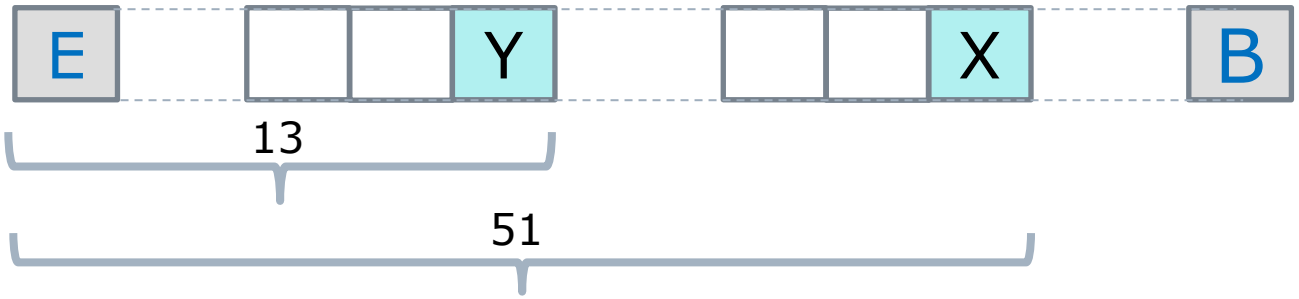


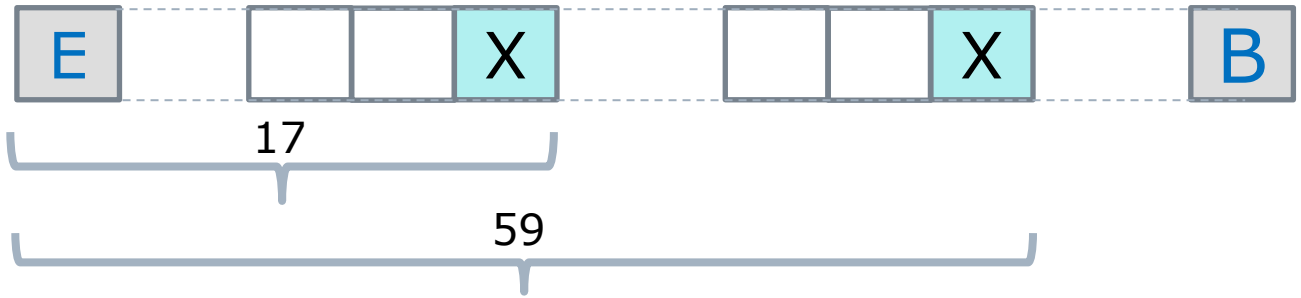
先頭のE、最後のB以外の場所には、a, b, c, d の文字がランダムに入っている。

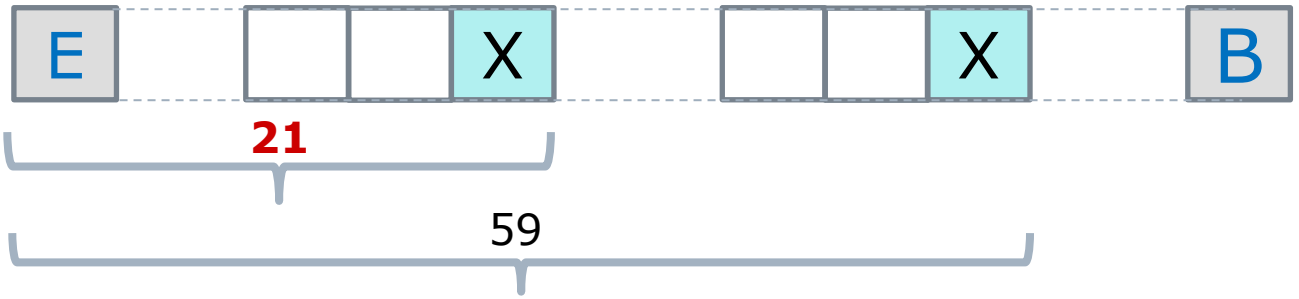
ただし、先頭から10～20番目と50～60番目のランダムに選ばれたな二箇所に、XまたはYの文字が入っている。



例えば、次の二例は、条件を満たすが、三番目の例は、条件を満たしていない。







この時、範囲指定された二箇所に、

X,Xが入っているならQ、
X,Yが入っているならR、
Y,Xが入っているならS、
Y,Yが入っているならU

として、与えられた文字列を分類する。

先の第一例はS、第二例はQ、
第三例は、そのいずれでもないことになる。

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Task6bは、三箇所版。
X,X,X -> Q; X,X,Y -> R; X,Y, X -> S; X,Y,Y -> U;
Y,X,X -> V ; Y,X,Y -> A; Y,Y,X -> B; Y,Y,Y -> C
の8分類。

RNNによる文の生成

"Generating Text with Recurrent Neural Networks"

Ilya Sutskever et al.

http://goo.gl/vHRHSn

2011年

ニューラルネットによる文章の生成

- Googleの Ilya Sutskeverは、文字数が5億文字にものぼるテキストを長い時間をかけてRecurrent Neural Nets に学習させ、次のページのような文章を生成することができた。

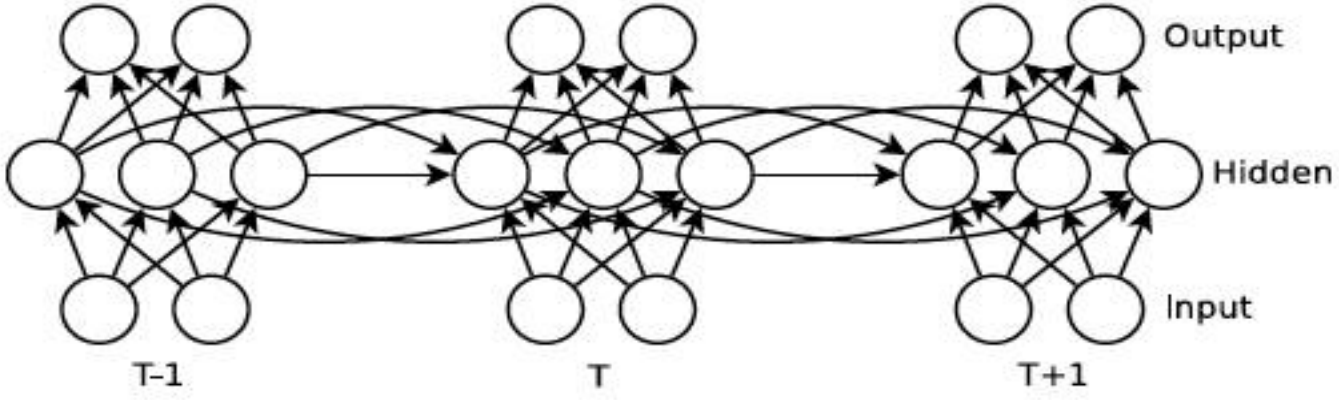


Figure 1. A Recurrent Neural Network is a very deep feedforward neural network whose weights are shared across time. The non-linear activation function used by the hidden units is the source of the RNN's rich dynamics.

“An example of what recurrent neural nets can now do” WikiPedia で学習したもの

- The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. In the show’s agreement unanimously resurfaced. The wild pasteured with consistent street forests were incorporated by the 15th century BE. In 1996 the primary rapford undergoes an effort that the reserve conditioning, written into Jewish cities, sleepers to incorporate the .St Eurasia that activates the population.

http://goo.gl/vHRHSn

“An example of what recurrent neural nets can now do” New York Timesで学習

- while he was giving attention to the second advantage of school building a 2-for-2 stool killed by the Cultures saddled with a halfsuit defending the Bharatiya Fernall ’s office . Ms . Claire Parters will also have a history temple for him to raise jobs until naked Prodiena to paint baseball partners , provided people to ride both of Manhattan in 1978 , but what was largely directed to China in 1946 , focusing on the trademark period is the sailboat yesterday and comments on whom they obtain overheard within the 120th anniversary , where .....

http://goo.gl/vHRHSn

RNNの驚くべき能力

"The Unreasonable Effectiveness of Recurrent Neural Networks"

Andrej Karpathy

http://goo.gl/mNqwCv

2015年

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M^\bullet} = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longrightarrow (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets.

RNNが産み出した数学論文モドキ

Stack Theory の教科書を「学習」させたもの

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

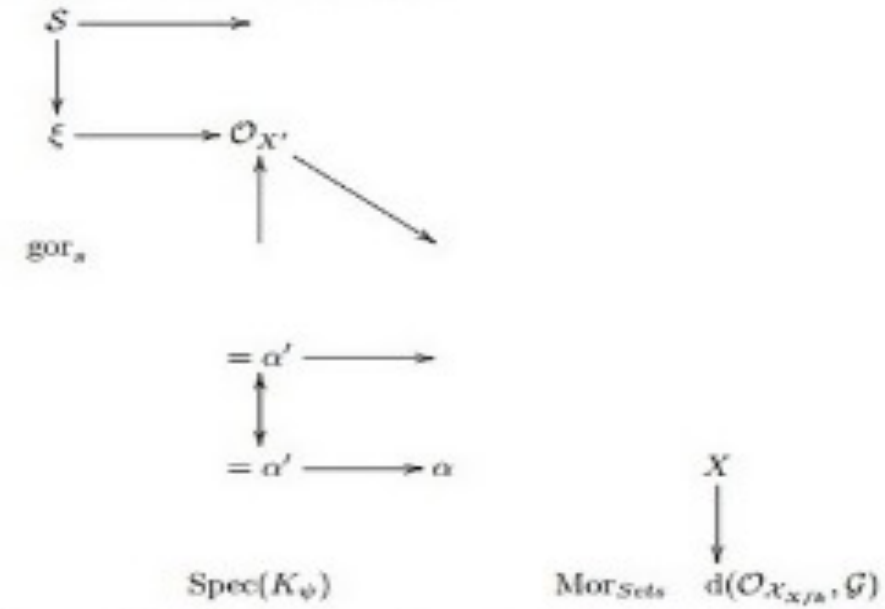
(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ finite type.

RNNが産み出した数学論文モドキ

Stack Theory の教科書を「学習」させたもの

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U.

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The fu
"field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_i}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\nu}^{\overline{x}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type o
If $\mathcal{F}$ is a scheme theoretic image points.

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??
sequence of $\mathcal{F}$ is a similar morphism.

RNNが産み出した
数学論文モドキ

Stack Theory の教科書を
「学習」させたもの

RNNは、Latexの構文規則を学習し、
こうしたシーケンスを構成できる。

```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on $\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \to \mathcal{I}$
is an injective and let $\mathfrak q$ be an abelian sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$ be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on $\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module. Then
$\mathcal{F}$ is an abelian catenary over $\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{lemma}
```

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
 int error;
 if (fd == MARN_EPT) {
  /*
   * The kernel blank will coeld it to userspace.
   */
  if (ss->segment < mem_total)
   unblock_graph_and_set_blocked();
  else
   ret = 1;
  goto bail;
 }
 segaddr = in_SB(in.addr);
 selector = seg / 16;
 setup_works = true;
 for (i = 0; i < blocks; i++) {
  seq = buf[i++];
  bpf = bd->bd.next + i * search;
  if (fd) {
   current = blocked;
  }
```

RNNが産み出した
Cプログラム・モドキ

Linuxのソースコードを
「学習」させたもの

```
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}

static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
  return 0;
}
```

RNNが産み出した
Cプログラム・モドキ

Linuxのソースコードを
「学習」させたもの

Generating Baby Names from 8000 baby names

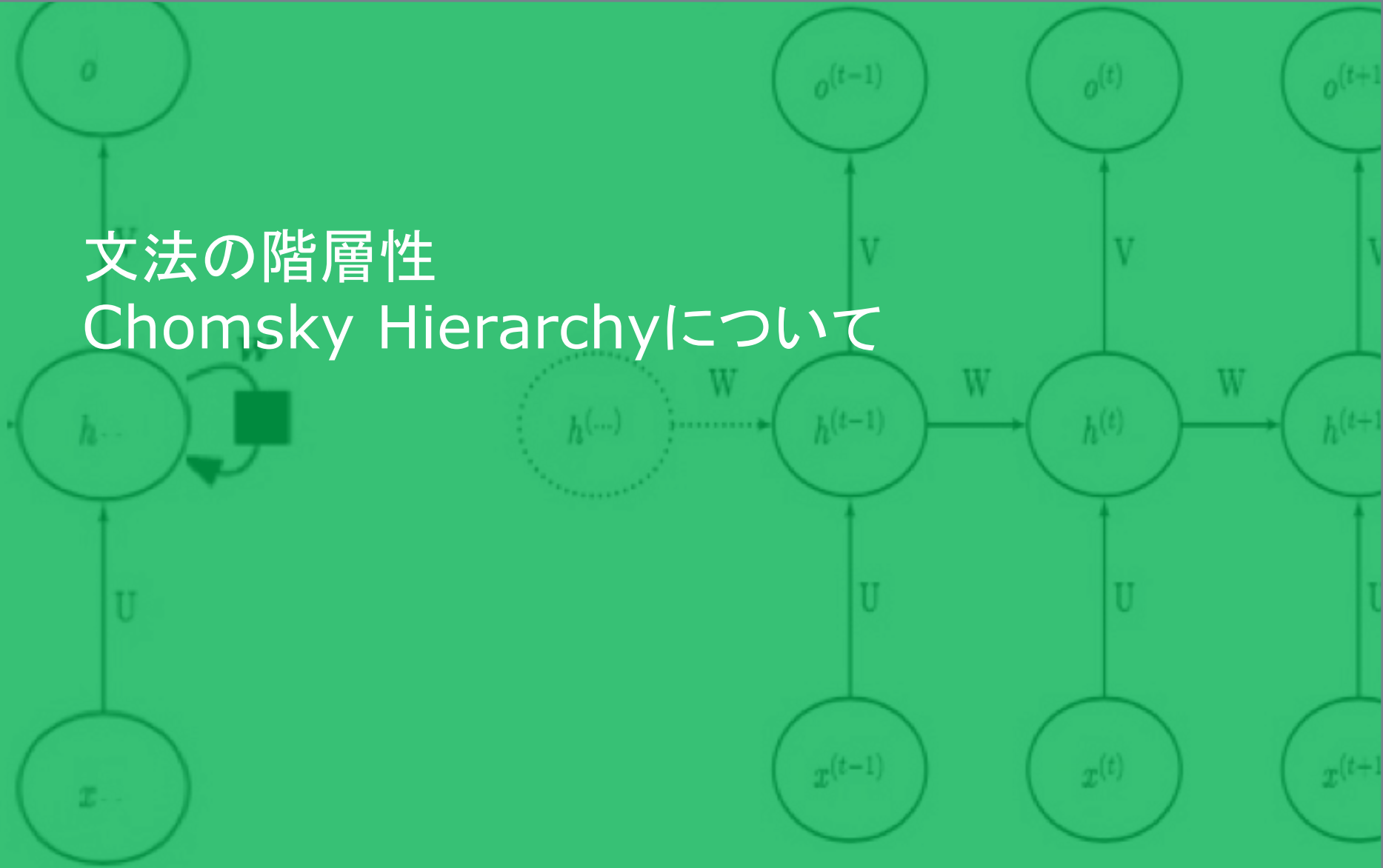
- *Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina*


タモリの「四カ国麻雀」
「ハナモゲラ日本語」は、
RNNで真似できる。
（多分）

マシンは、簡単な文法を理解できる
でも、意味はどこに？

- これらの取り組みは、マシンが、例えば、C言語の構文や、LaTexの構文は、ほぼ完璧に学習していること示しており、興味深いものだ。ただし、自然言語の生成では、いくつかの破綻が見られる。
- このことは、プログラム言語の文法規則が、基本的には「文脈自由文法（レベル2）」で、自然言語の文法規則である「文脈依存文法(レベル1)」よりも単純であることの表れとして理解できる。より単純な、有限オートマトンで表わされる「正規文法（レベル3）」の構造を、その出力から推定する問題が、膨大な計算を必要とするように、ここでは、膨大な計算が行われている。
- もちろん、このアプローチの最大の問題は、文の「意味」を捉え損ねていることである。文字通り「意味がない」のだ。

文法の階層性
Chomsky Hierarchyについて



Chomsky Hierarchy

- 形式的言語の形式的文法は、次のような階層をなすことが知られている。これをChomsky Hierarchyと呼ぶ。
  - タイプ-0 文法は、全ての形式文法を包含する。
  - タイプ-1 文法は、文脈依存言語を生成する。
  - タイプ-2 文法は、文脈自由言語を生成する。
  - タイプ-3 文法は、正規言語を生成する。
- 機械が、この階層の中に位置付けられるどの形式的な文法を理解・学習できたかを考えることができる。

https://goo.gl/lR960c

Chomsky Hierarchy

Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	$\alpha \to \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \to \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \to \gamma$
Type-3	Regular	Finite state automaton	$A \to a$ and $A \to aB$

https://en.wikipedia.org/wiki/Chomsky_hierarchy

Chomsky Hierarchyと Deep Learningでの文法理解の取り組み

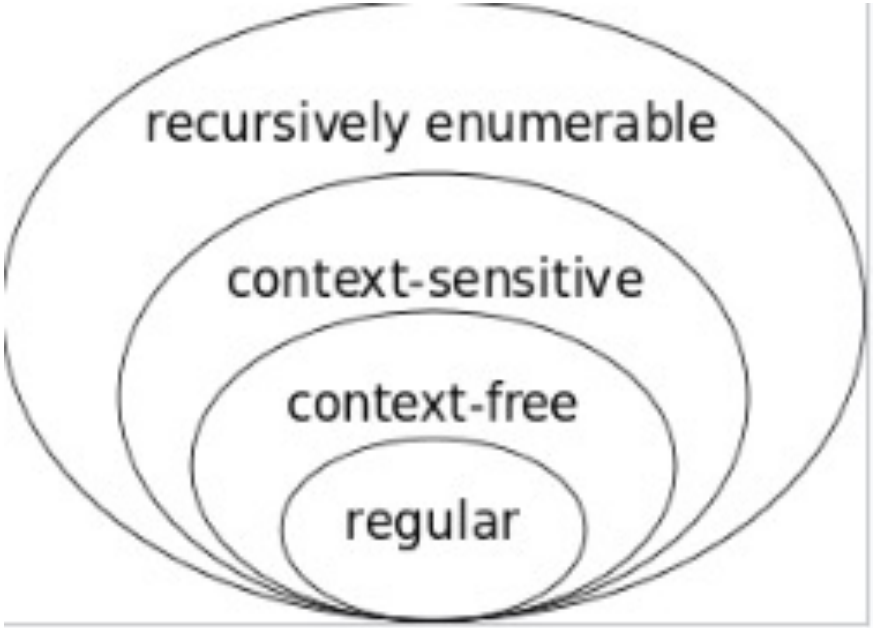
	文法	例	Deep Learning
Type-0	帰納的可算	チューリングマシン	
Type-1	文脈依存文法	自然言語*	?
Type-2	文脈自由文法	プログラム言語	Karpathy **2015年**
Type-3	正規文法	有限オートマトン	Hochreiter **1997年**

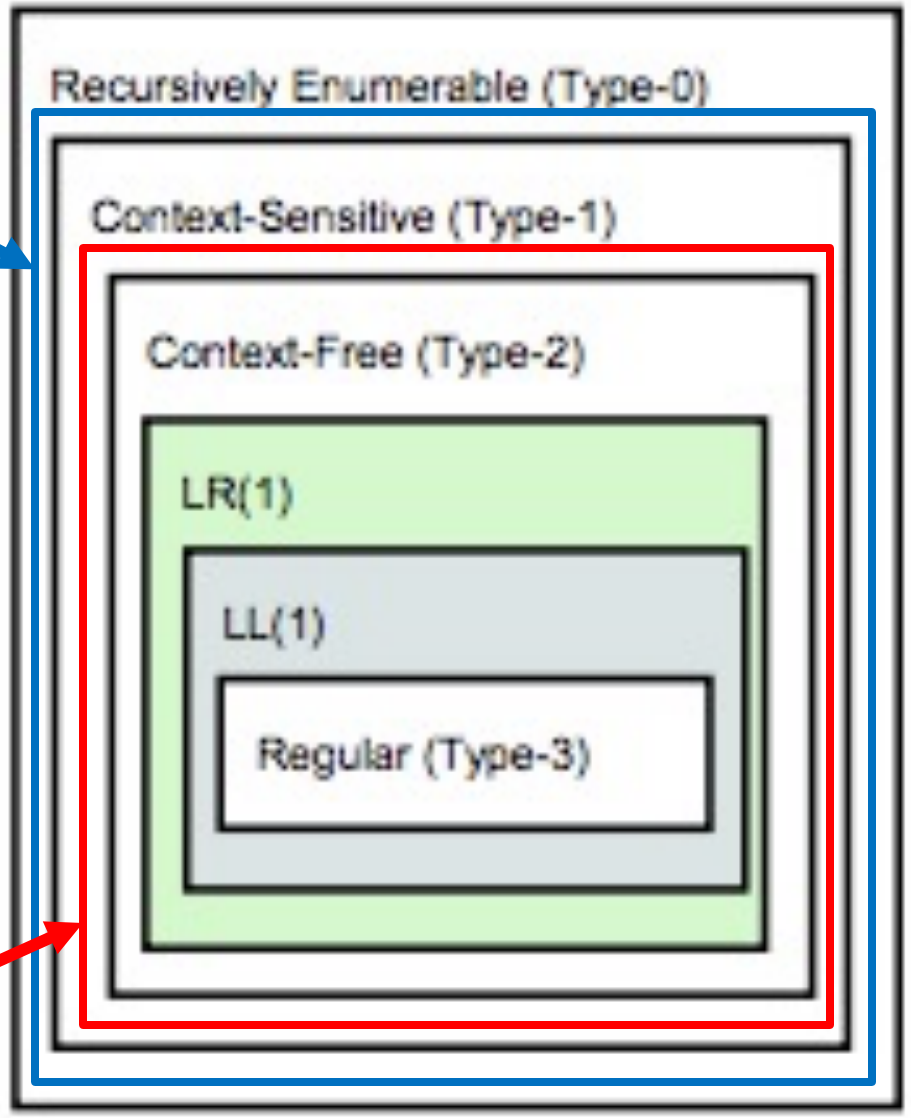
自然言語の多くの文法は、Contex-Freeで記述できる。
Mildly Context Sensitive Languages',proposed by Aravind Joshi.

Chomsky Hierarchyと Deep Learningでの文法理解の取り組み

	文法	例	Deep Learning
Type-0	帰納的可算	チューリングマシン	
Type-1	文脈依存文法	自然言語	Google Neural Machine Translation System ？ **2016年**
Type-2	文脈自由文法	プログラム言語	Karpathy **2015年**
Type-3	正規文法	有限オートマトン	Hochreiter **1997年**

Recursive Language
“Merge” is recursive

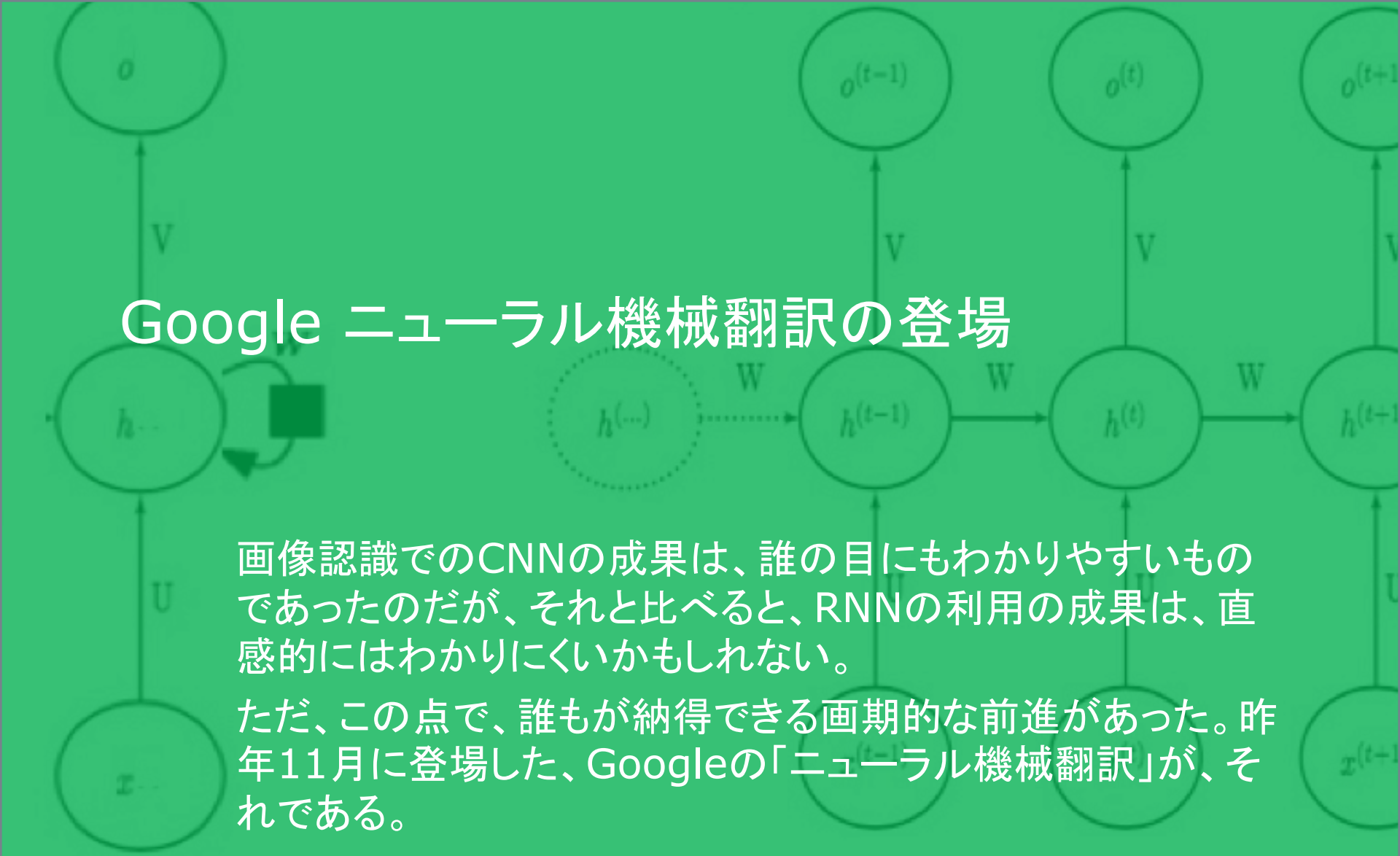




自然言語？

Mildly Context Sensitive Languages

Google ニューラル機械翻訳の登場



画像認識でのCNNの成果は、誰の目にもわかりやすいものであったのだが、それと比べると、RNNの利用の成果は、直感的にはわかりにくいかもしれない。

ただ、この点で、誰もが納得できる画期的な前進があった。昨年11月に登場した、Googleの「ニューラル機械翻訳」が、それである。

英語と日本語の差異から見るGoogle翻訳

- 感覚的には明らかなのだが、Google翻訳が実現した「飛躍」が、どのようなものかを正確に述べるのは、意外と難しい。翻訳の評価でよく利用されるBLEU等のスコアは、翻訳改善の重要な目安にはなるのだが、それは、あくまで量的なものだ。質的な「飛躍」は、その数字には、間接的にしか反映していない。
- ここでは、英語と日本語の文法の差異に注目して、その差異が、Google翻訳では、どのように埋められているかを、いくつかの具体例で見てみよう。それらは、日本語・英語翻訳の中心的課題であるにもかかわらず、以前の機械翻訳技術では、うまく扱えなかったものである。
- 英語と日本語の文法の差異については、Chomskyの以前の”Principles and Parameters” 理論を援用した。

https://en.wikipedia.org/wiki/Principles_and_parameters

Head-directionality parameter

日英文法の差異

head-initial / head-final

Haed(文法的な補語・修飾語の対象)が、先に来るか後に来るかという違い

- **head-initial** English
  - ***eat*** *an apple*
  - ***a person*** *happy about her work*
  - *I live* ***in*** *Takasu village.*
  - ***any*** *book*
  - *We saw* ***that*** *Mary did not swim*
- **head-final** Japanese
  - *リンゴを食べる*
  - *ジョンの昨日のニューヨークでの講義*
  - *僕が、高須村に住んでいる*
  - *誰も*
  - *マリーが泳がなかったと*

Null-subject Parameter

主語の省略を許すか許さないかの違い

- Null-subject Parameter (+) Japanese
  Null-subject Parameter (-) English

- 私達は買い物をした。後でご飯を食べた。
- *We went shopping. Afterwards, **we** ate dinner.*

- 今日はゲームの発売日なんだけど、買おうかどうか迷っている。
- *The game comes out today, but **I** can't decide whether or not to buy **it**.*

Pro-drop Parameter

代名詞の省略を許すか許さないかの違い

- Pro-drop Parameter (+) Japanese
  Pro-drop Parameter (-) English

- このケーキは美味しい。誰が焼いたの？
- *This cake is tasty. Who baked **it**?*

- 知らない。気に入った？
- ***I** don't know. Did **you** like **it**?*

Google翻訳と他の翻訳の比較

- 今日はゲームの発売日なんだけど、買おうかどうか迷っている。
  - Today is the release date of the game, but I'm wondering if I should buy it. (Google翻訳)
  - Wondering whether today is the release date of the game, but I'm buying.（その他の翻訳）

- 買い物をした。後でご飯を食べた。
  - I did some shopping. I ate rice later. (Google翻訳)
  - With the shopping. After eating.（その他の翻訳）

- このケーキは美味しい。誰が焼いたの？
  - This cake is tasty. Who baked it? (Google翻訳)
  - This cake is delicious. Who baked them?（その他の翻訳）

- 知らない。気に入った？
  - Do not know. favorite? (Google翻訳)
  - Don't know. Into your mind?（その他の翻訳）

Google翻訳と他の翻訳の比較

- ジョンの昨日のニューヨークでの講義
  - Lecture by John yesterday in New York (Google翻訳)
  - In New York yesterday by John's lecture（その他の翻訳）
- 風邪がひどい
  - My cold is bad. (Google翻訳)
  - Terrible cold（その他の翻訳）
- 私はうなぎ
  - I am eel (Google翻訳)
  - I am an eel（その他の翻訳）
- 私はうなぎかカツ丼
  - I like eels or cutlet on rice (Google翻訳)
  - I am the eel or katsudon（その他の翻訳）

Google翻訳は、意味を理解しているのか？

- 注意すべきなのは、Googleの「ニューラル機械翻訳」で、RNNが、言語の意味理解の能力を獲得したわけではないだろうということ。
- ある言語Aを母語とするある人が、ある「意味」を込めた文S_Aを発話したとする。もちろんこの文は、A言語の文法にかなっている。機械翻訳システムは、このS_Aを他の言語Bの文S_Bに変換する。もしこの文S_Bが、B言語の文法にかなっていれば、B言語を母語とする人は、この文S_Bに「意味」を見い出す。
- 翻訳システムは、S_Aの意味をS_Bの意味に翻訳したように見えるのだが、そう見えるのは、送り手と受け手の双方の「人間」が、それぞれの文の意味の解釈を、行なっているからである。

Google翻訳は、意味を理解しているのか？

- 翻訳システムが行なっているのは、A言語の文法的な文字列S_Aを、「対応する」B言語の文法的な文字列S_Bに、書き換えているだけである。そこには、「意味」の介在は必要ではない。
- 問題は、「対応」の中身だが。一番、自然な解釈は、A言語の文法とB言語の文法の「対応」である。
- 更に言えば、もしも、このシステムが、基本的には同一の構成のままで、A言語、B言語だけでなく、多言語間の翻訳が可能であるなら、このシステムは、多くの言語の文法の対応付けを可能とする、より深い共通の文法構造を理解していると考えることができる。これは、Chomskyの言う、**Universal Grammer** 普遍文法だと思うと、興味深い。

Googleのニューラル機械翻訳（1）

"Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation"

Yonghui Wu et al.

https://goo.gl/YqlAAW

2016年

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

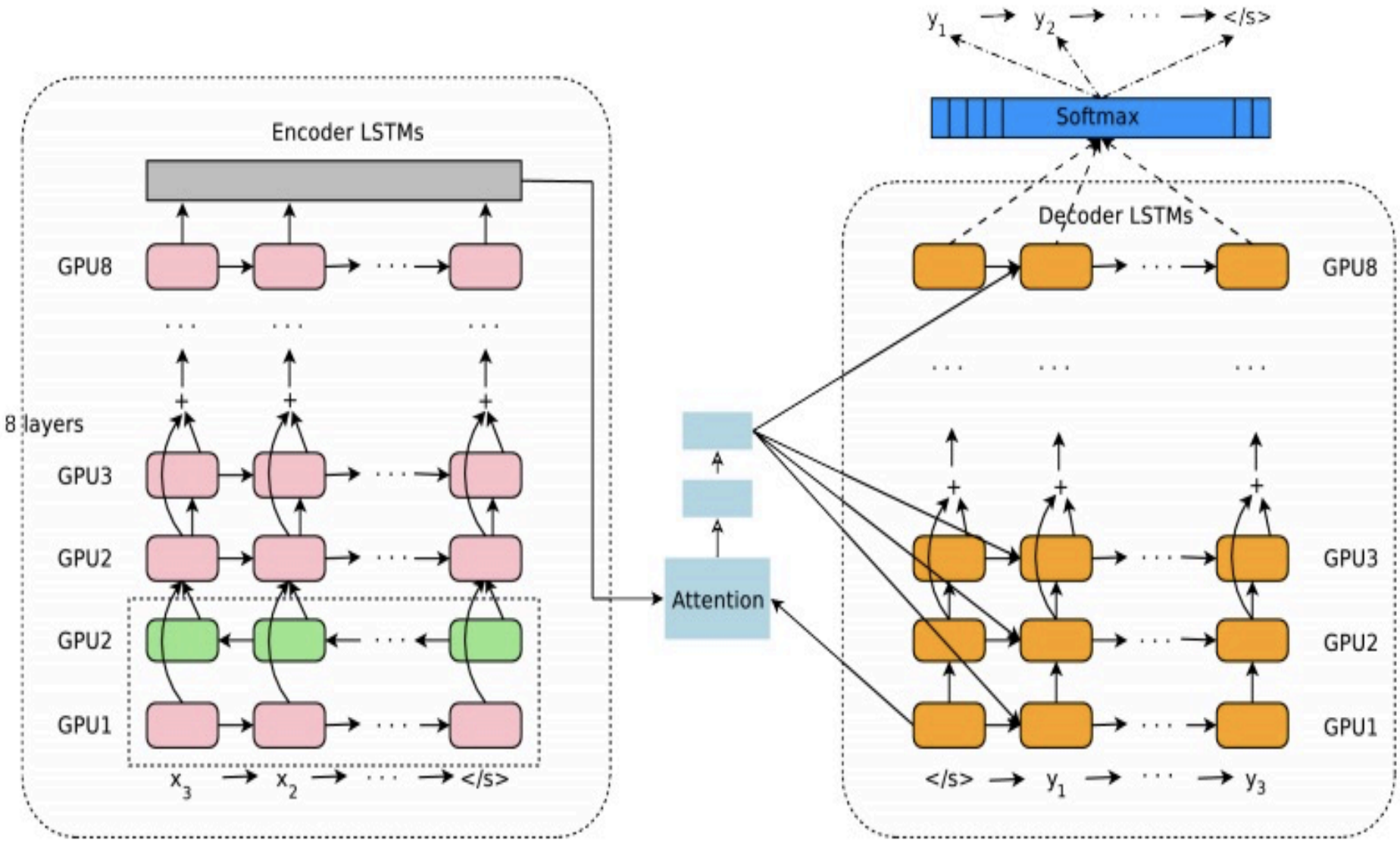
Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

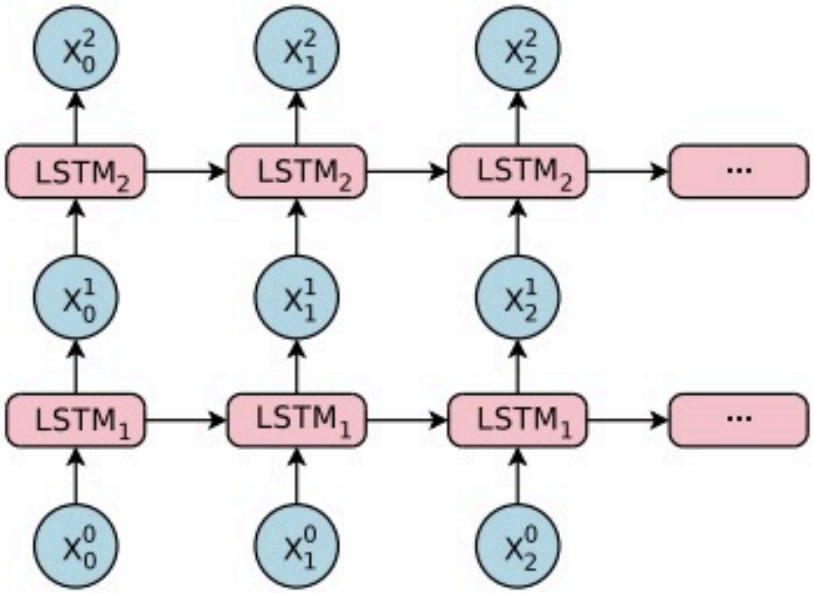
Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

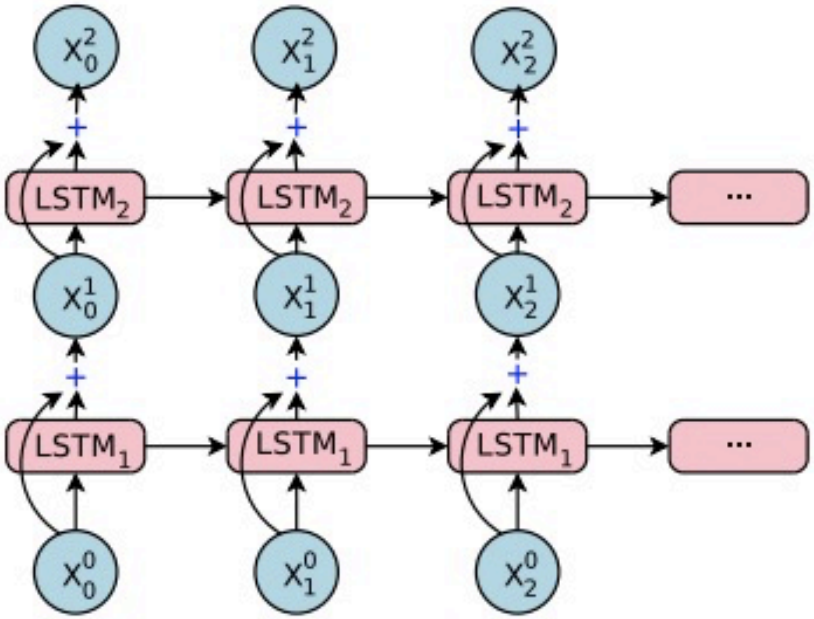
Abstract

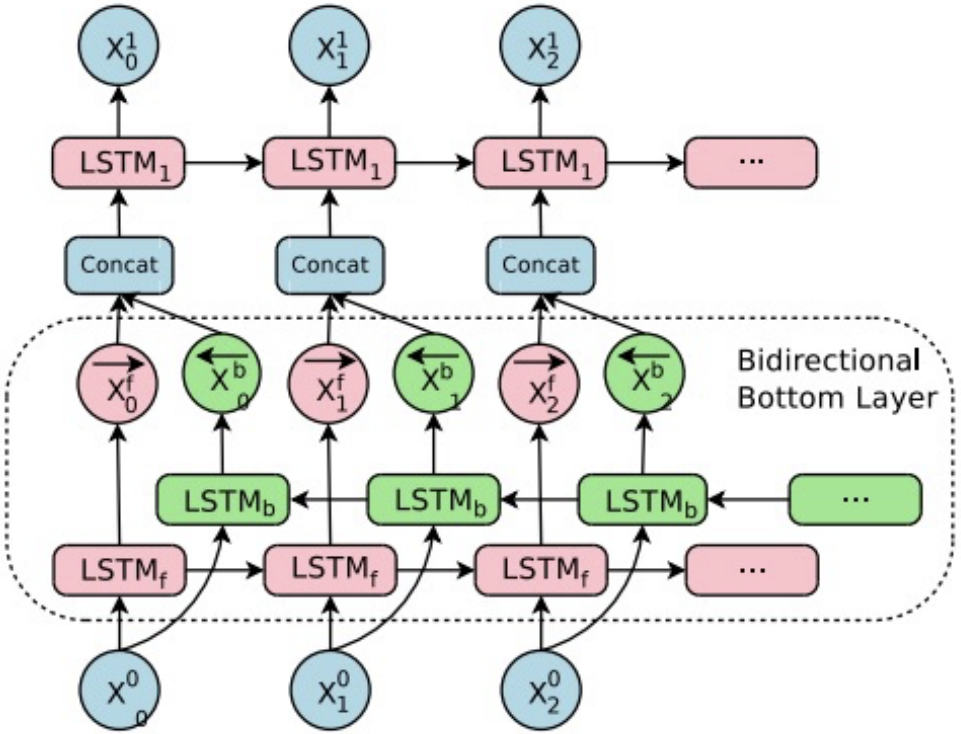
Neural Machine Translation (NMT) is an end-to-end learning approach for automated translation, with the potential to overcome many of the weaknesses of conventional phrase-based translation systems. Unfortunately, NMT systems are known to be computationally expensive both in training and in translation inference – sometimes prohibitively so in the case of very large data sets and large models. Several authors have also charged that NMT systems lack robustness, particularly when input sentences contain rare words. These issues have hindered NMT's use in practical deployments and services, where both accuracy and speed are essential. In this work, we present GNMT, Google's Neural Machine Translation system, which attempts to address many of these issues. Our model consists of a deep LSTM network with 8 encoder and 8 decoder layers using residual connections as well as attention connections from the decoder network to the encoder. To improve parallelism and therefore decrease training time, our attention mechanism connects the bottom layer of the decoder to the top layer of the encoder. To accelerate the final translation

https://arxiv.org/pdf/1609.08144.pdf


Encoder LSTMs
GPU8
8 layers
GPU3
GPU2
GPU2
GPU1
x3
x2
</s>
Attention
y1
y2
</s>
Softmax
Decoder LSTMs
GPU8
GPU3
GPU2
GPU1
</s>
y1
y3


x_0^2 x_1^2 x_2^2
LSTM$_2$ LSTM$_2$ LSTM$_2$ ...
x_0^1 x_1^1 x_2^1
LSTM$_1$ LSTM$_1$ LSTM$_1$ ...
x_0^0 x_1^0 x_2^0


x_0^2 x_1^2 x_2^2
+
LSTM$_2$ LSTM$_2$ LSTM$_2$ ...
x_0^1 x_1^1 x_2^1
+
LSTM$_1$ LSTM$_1$ LSTM$_1$ ...
x_0^0 x_1^0 x_2^0


x_0^1 x_1^1 x_2^1
LSTM$_1$ LSTM$_1$ LSTM$_1$ ...
Concat Concat Concat
$\overrightarrow{x_0^f}$ $\overleftarrow{x_0^b}$ $\overrightarrow{x_1^f}$ $\overleftarrow{x_1^b}$ $\overrightarrow{x_2^f}$ $\overleftarrow{x_2^b}$
Bidirectional Bottom Layer
LSTM$_b$ LSTM$_b$ LSTM$_b$ ...
LSTM$_f$ LSTM$_f$ LSTM$_f$ ...
x_0^0 x_1^0 x_2^0

Googleのニューラル機械翻訳（2）

"Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation"

Melvin Johnson et al.

https://goo.gl/islUXa

2016年

Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu,
Zhifeng Chen, Nikhil Thorat
melvinp,schuster,qvl,krikun,yonghui,zhifengc,nsthorat@google.com

Fernanda Viégas, Martin Wattenberg, Greg Corrado,
Macduff Hughes, Jeffrey Dean

Abstract

We propose a simple, elegant solution to use a single Neural Machine Translation (NMT) model to translate between multiple languages. Our solution requires no change in the model architecture from our base system but instead introduces an artificial token at the beginning of the input sentence to specify the required target language. The rest of the model, which includes encoder, decoder and attention, remains unchanged and is shared across all languages. Using a shared wordpiece vocabulary, our approach enables Multilingual NMT using a single model without any increase in parameters, which is significantly simpler than previous proposals for Multilingual NMT. Our method often improves the translation quality of all involved language pairs, even while keeping the total number of model parameters constant. On the WMT'14 benchmarks, a single multilingual model achieves comparable performance for English→French and surpasses state-of-the-art results for English→German. Similarly, a single multilingual model surpasses state-of-the-art results for French→English and German→English on WMT'14 and WMT'15 benchmarks respectively. On production corpora, multilingual models of up to twelve language pairs allow for better translation of many individual pairs. In addition to improving the translation quality of language pairs that the model was trained with, our models can also learn to perform implicit bridging between language pairs never seen explicitly during training, showing that transfer learning and zero-shot translation is possible for neural translation. Finally, we show analyses that hints at a universal interlingua representation in our models and show some interesting examples when mixing languages.

https://arxiv.org/pdf/1611.04558.pdf

Universal interlingua representation !

- In addition to improving the translation quality of language pairs that the model was trained with, our models can also learn to perform implicit bridging between language pairs never seen explicitly during training, showing that transfer learning and zero-shot translation is possible for neural translation. Finally, we show analyses that hints at a **universal interlingua representation** in our models and show some interesting examples when mixing languages.

interlingua between all these languages.

- The most interesting observation is that both Model 1 and Model 2 can perform zero-shot translation with reasonable quality (see (d) and (e)). It should be noted that Model 2 outperforms Model 1 by close to 3 BLEU points. In other words, the addition of Spanish on the source side and Portuguese on the target side helps Portuguese→Spanish zero-shot translation. We believe that this is possible only because our shared architecture enables the model to learn an **interlingua** between all these languages. We explore this hypothesis in more detail in Section 5

4.6 Zero-Shot Translation

- An interesting benefit of our approach is that we can perform zero-shot translation between a language pair for which **no explicit training data** has been seen. To demonstrate this we will use two multilingual models — a model trained with examples from two different language-pairs, Portuguese→English and English→Spanish (Model 1), and a model trained with examples from four different language-pairs, English↔Portuguese and English↔Spanish (Model 2). We show that both of these models can generate reasonably good quality Portuguese→Spanish translations without ever having seen Portuguese→Spanish data during training.

5.1 Evidence for an Interlingua

mixing languages

- **Japanese:** 私は東京大学の学生です。 → I am a student at Tokyo University.
- **Korean:** 나는 도쿄 대학의 학생입니다. → I am a student at Tokyo University.
- **Mixed Japanese/Korean:** 私は東京大学학생입니다. → I am a student of Tokyo University.

Googleニューラル機械翻訳については、詳しくは、次回のマルレク 3/27 で！

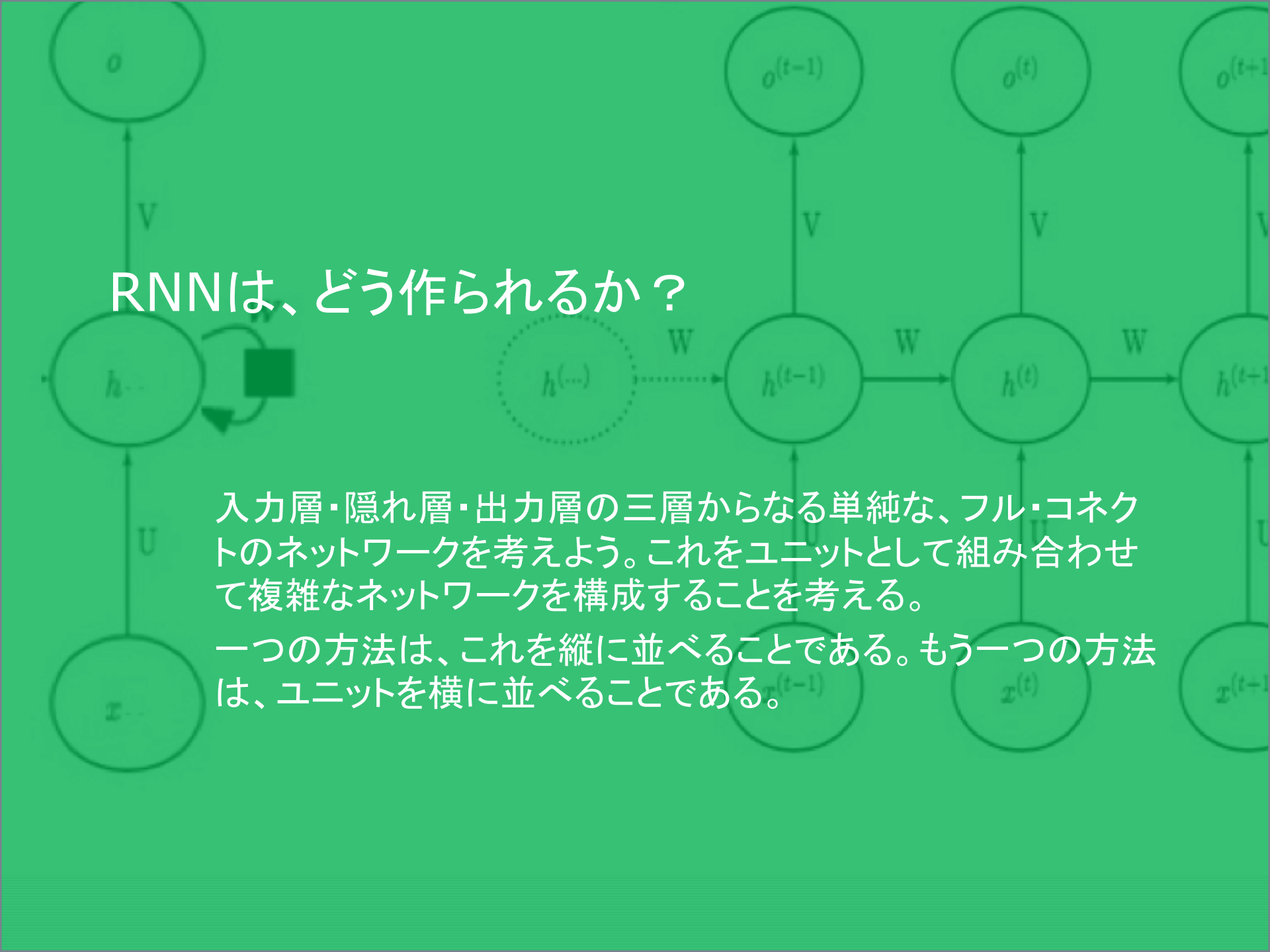
Part II

RNNとは何か

Part II

RNNとは何か　Agenda

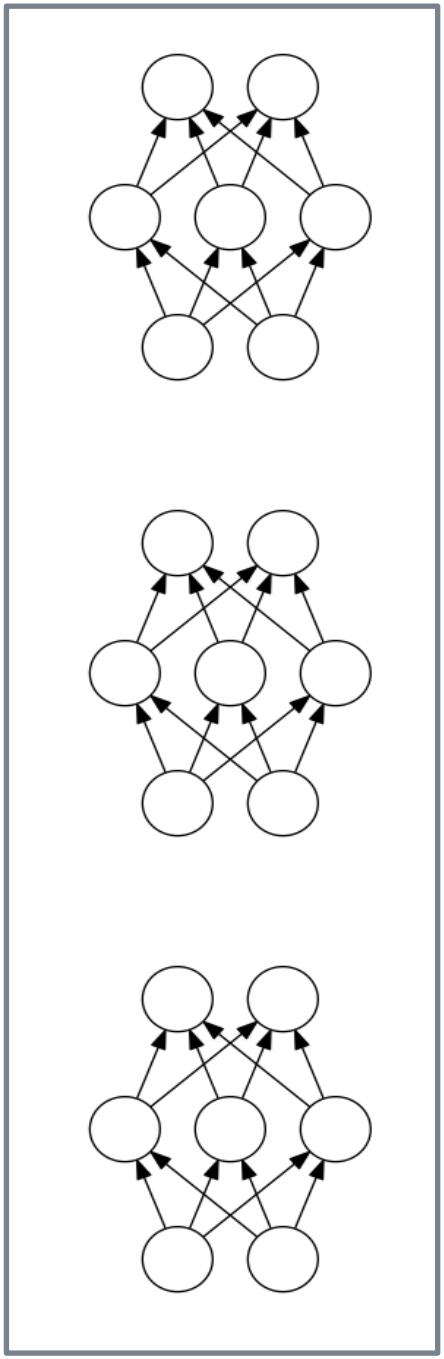
- RNNは、どう作られるか？
- RNNをグラフで表す-- 展開形と再帰形
- Sequence to Sequence
- RNNでの φ（ WX + b ）の応用
- 20数年前、いったん放棄されたRNN
- RNNの復活



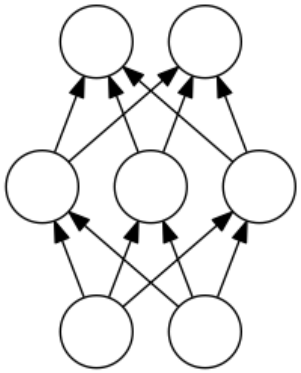
RNNは、どう作られるか？

入力層・隠れ層・出力層の三層からなる単純な、フル・コネクトのネットワークを考えよう。これをユニットとして組み合わせて複雑なネットワークを構成することを考える。

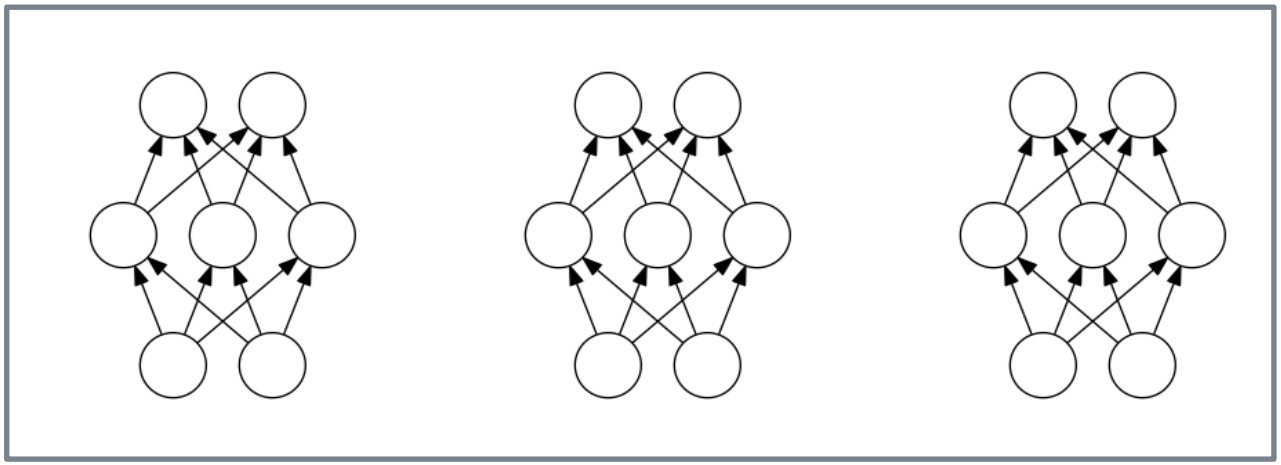
一つの方法は、これを縦に並べることである。もう一つの方法は、ユニットを横に並べることである。



縦方向に
並べる

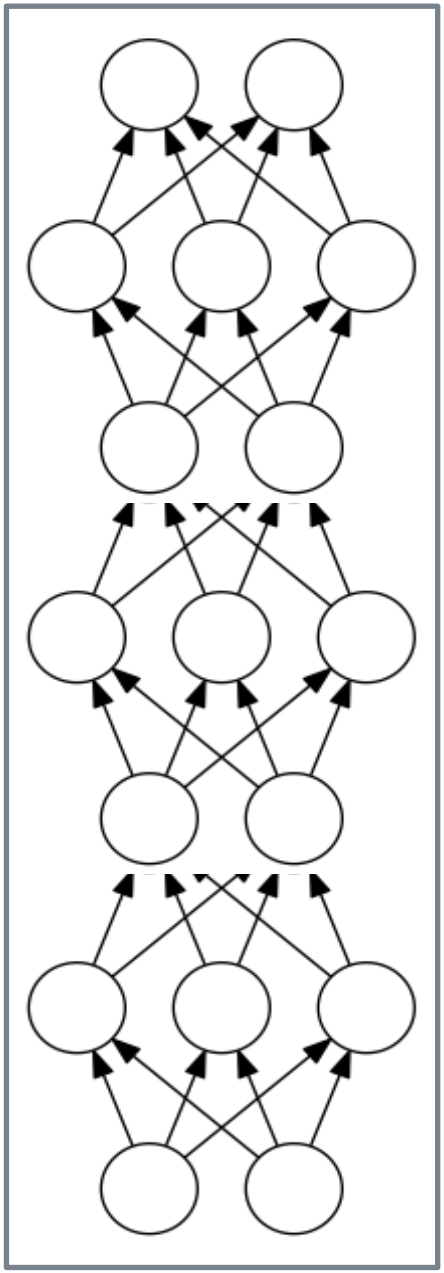


横方向に
並べる

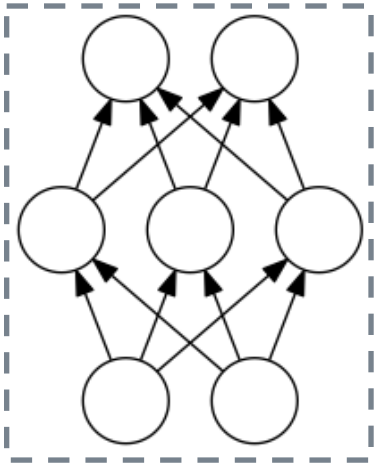


DNN（Full Connect Feed Forward Network)

- 縦に並べる場合、一つ上に積み上げるごとに、上のユニットの入力層のノードの数を、すぐ下のユニットの出力層のノードの数とを同じにして、重ね合わせれば、どんどん縦に積み重ねることができる。しかも、出来上がったグラフは、自然にフル・コネクトのネットワークになる。
- これがDNNである。（正式には、Full ConnectのFeed Forward Network である。）



縦方向に
並べる

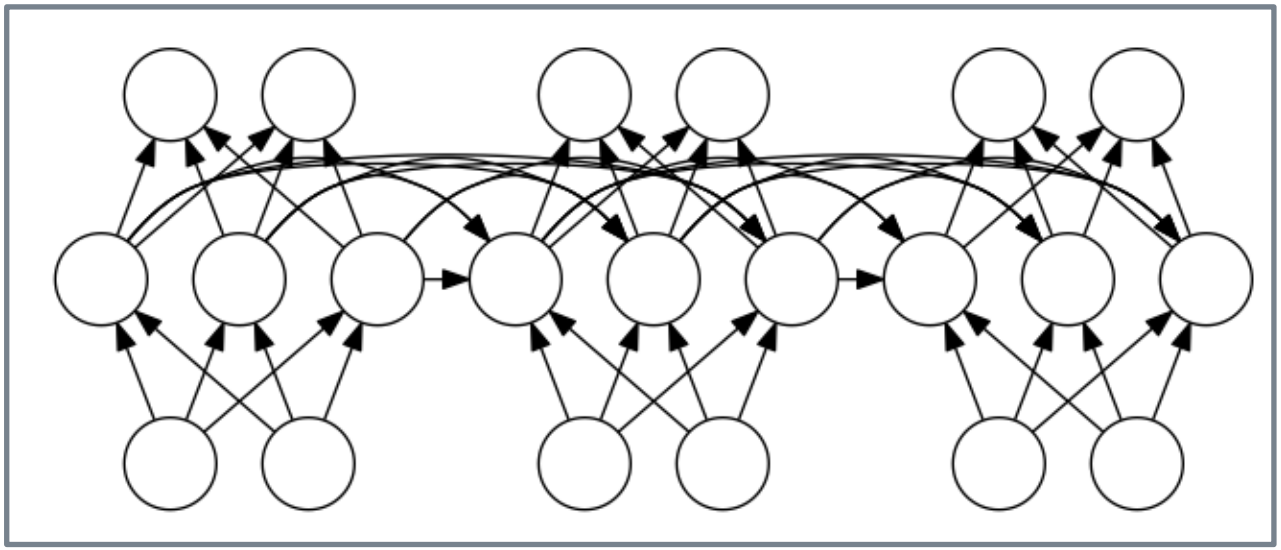


単純なネットワーク

Full Connect
DNN

横方向に
並べる

RNN



RNNは、Recurrentなネットワーク

- 横に並べる場合、ただ、横に並べただけでは、一つのネットワークにはならない。そこで、隣り合うユニットの隠れ層のノード同士を、フル・コネクトでつなぐこととする。
- これが、RNNである。と言いたいところだが、RNNには、ユニットを横につなげる時に、大きな条件が課せられている。それは、横につながるユニットは、同じ形のユニット（それぞれの入力層・隠れ層・出力層のノード数が同じ）でなければならないという条件である。要するに、RNNとは、同じ形のユニットが、横に「繰り返される」形のネットワークなのである。
- ただ、それだけではない。RNNには、もっと強い条件がある。横に繰り返されるのは、「同じ形」のユニットではなく、全く「同じ」ユニット（各ノードの重み、バイアスといったパラメーターも同一）の再帰的繰り返しであるという条件である。そのことで、RNNは、“Recurrent” Neural Networkと呼ばれる。

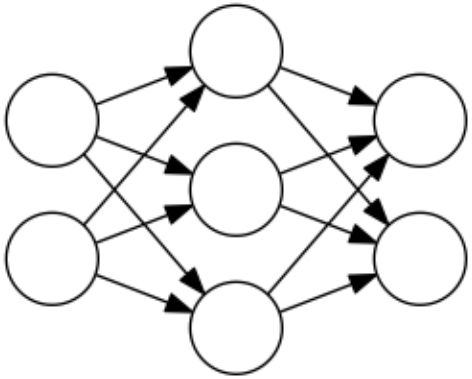
RNNをグラフで表す-- 展開形と再帰形

ここでは、RNNのネットワークの構成を、グラフで表してみよう。二つの表し方がある。展開形と再帰形である。

RNNのグラフの表記

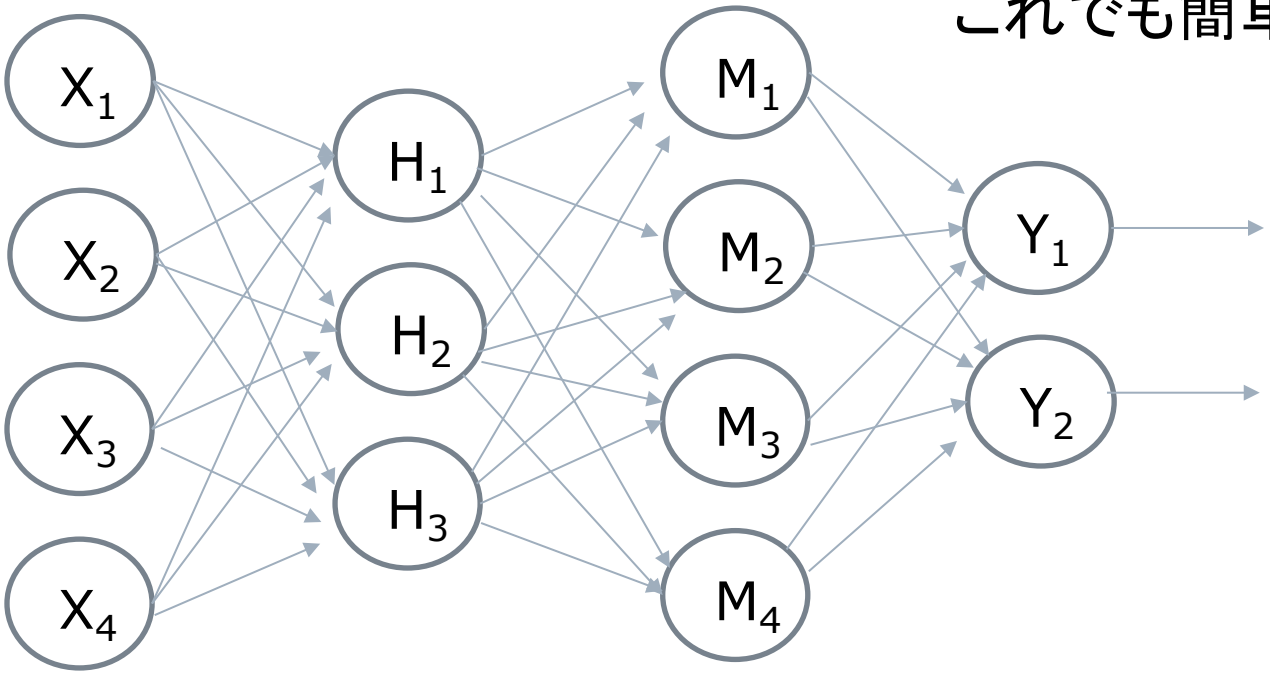
- 入力層・隠れ層・出力層の三層からなる単純なネットワークを考えよう（次の図の上）。これを、TensorFlowのグラフで書けば次のようになる（次の図の中）。ここでは、ニューラル・ネットワークの基本的な計算式 $\varphi(WX + b)$ が、明示的に図示されている。
- 新しい表記は、このグラフの細部を省略したものだ（次の図の下）。重みのパラメーターのU, Vのみを残して、二つのバイアスbも二つの活性化関数φも表記上からは消えている。ただ、入力ベクトルxが、パラメーターUの作用を受けて隠れ層のベクトルhを形成し、隠れ層ベクトルhが、パラメーターVの作用を受けて、出力ベクトルoに変換されることは、きちんと表現されている。

グラフの表記について

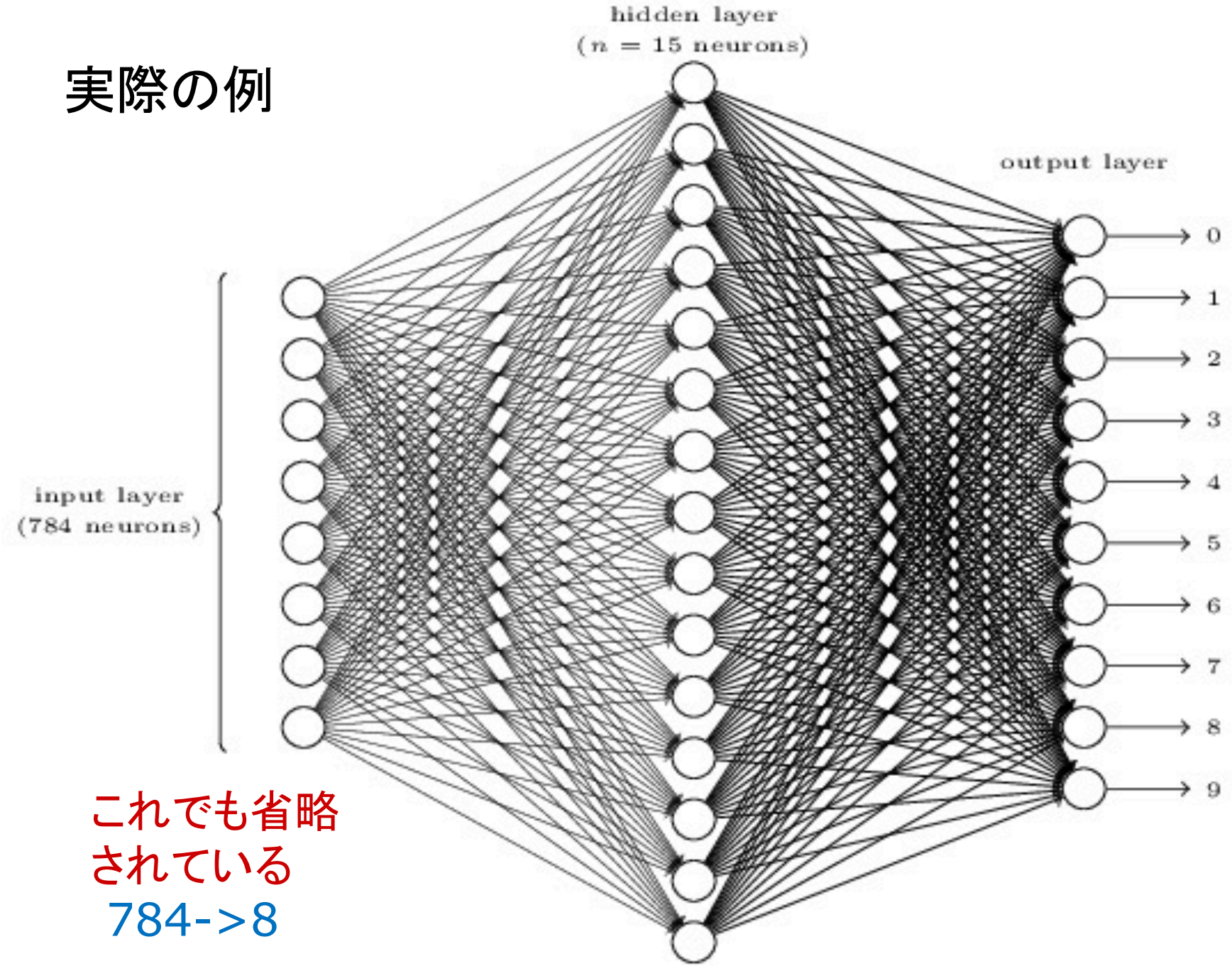


ニューロンが
2 + 3 + 2 = 7個
これは、簡単すぎる。

これでも簡単すぎる。



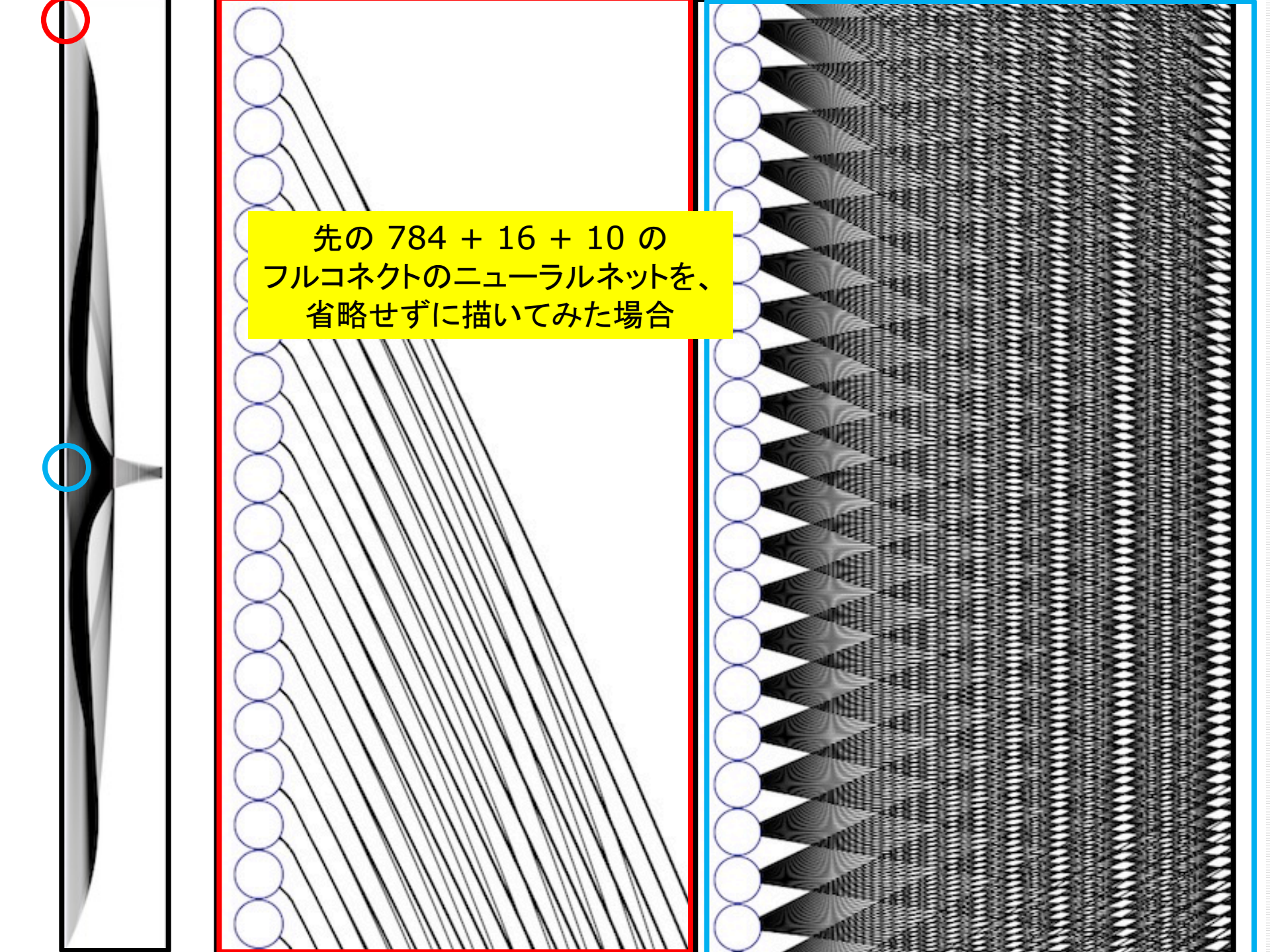
実際の例



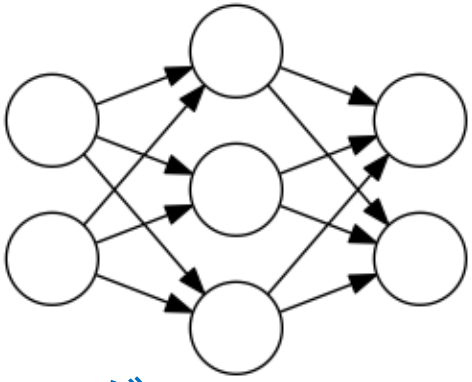
これでも省略
されている
784->8

先の 784 + 16 + 10 の
フルコネクトのニューラルネットを、
省略せずに描いてみた場合

？


先の 784 + 16 + 10 の
フルコネクトのニューラルネットを、
省略せずに描いてみた場合

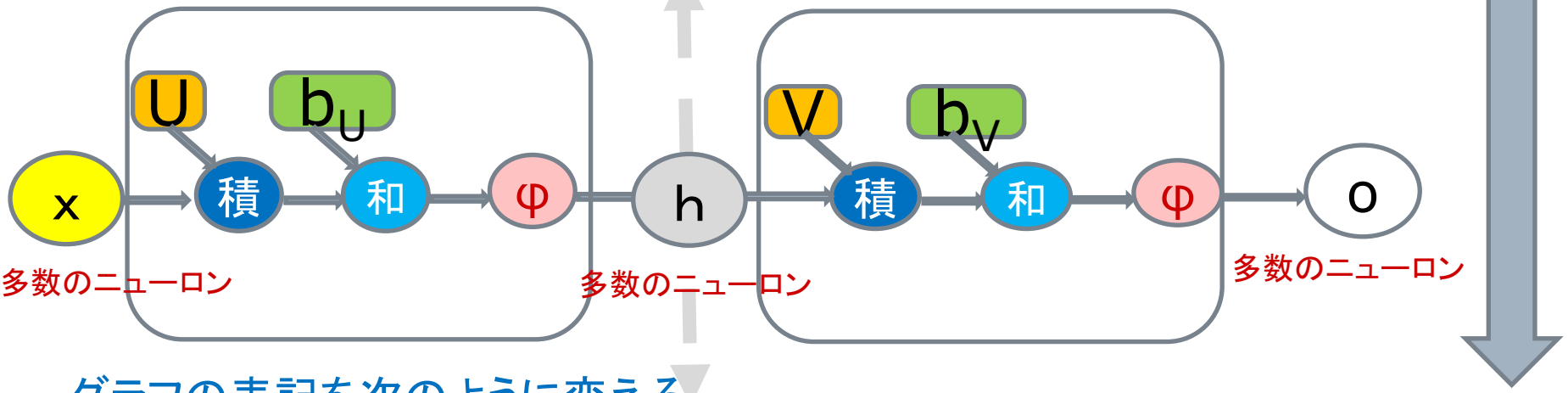
グラフの表記を変える



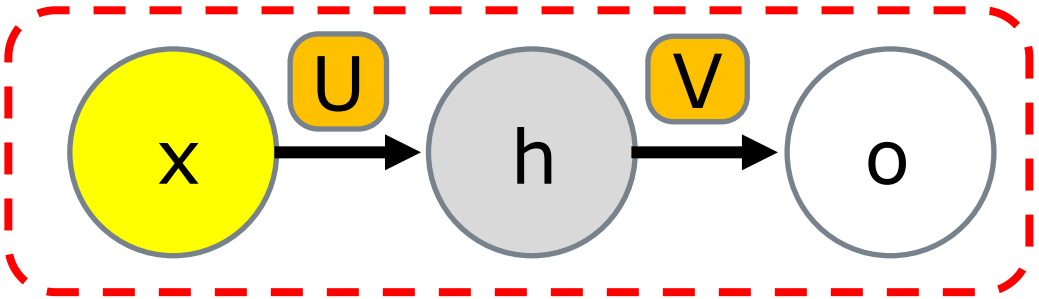
入力層・隠れ層・出力層からなる単純なネットワークがあったとしよう。

TensorFlowでのDNNのグラフ

TensorFlowのグラフで書けば次のようになる。

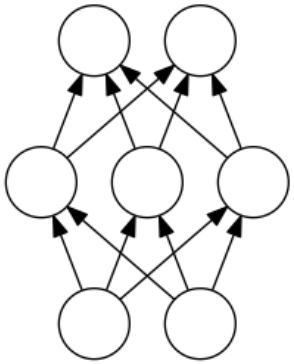


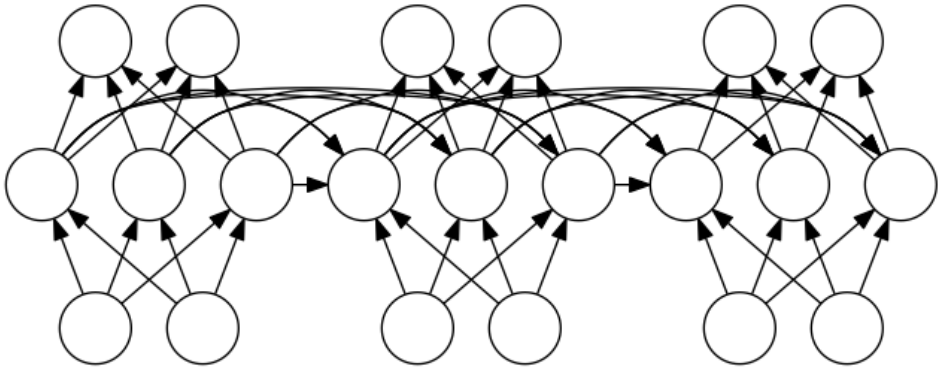
グラフの表記を次のように変える



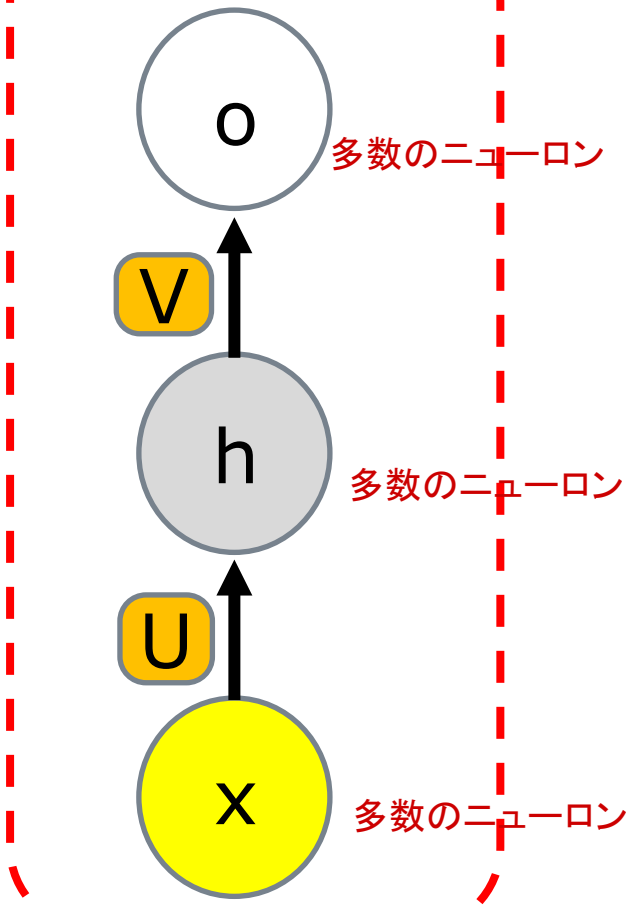
上のグラフの細部を省略してU,Vのみを残して、入力ベクトルxが、出力ベクトルoに変換されることを左のように表す。

- こうした表記（ただし、今度は、入力層を下に、出力層を上にしている（次の図の左））を用いると、単純な三層のニューラル・ネットワークを横に並べ、隣り合う隠れ層のノードをフル・コネクトで結んだネットワーク（次の図の右上）は、次の図の右下のように表されることになる。
- この図では、パラメーターの***U1, U2, U3*** も、***V1, V2, V3*** も、***W1, W2*** も、それぞれ異なっていることに注意しよう。これは、横につなげられたニューラル・ネットワークが、それぞれ異なったパラメーターを持つ、異なったネットワークであることを表している。

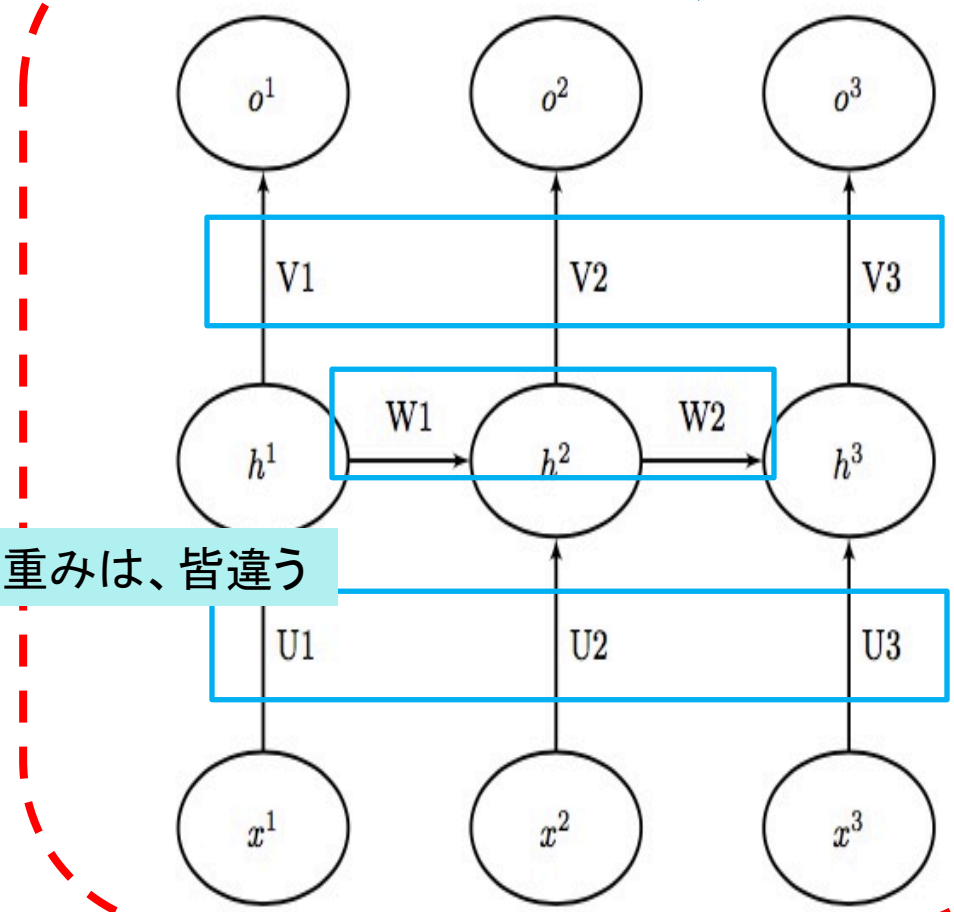




単純なネットの表記



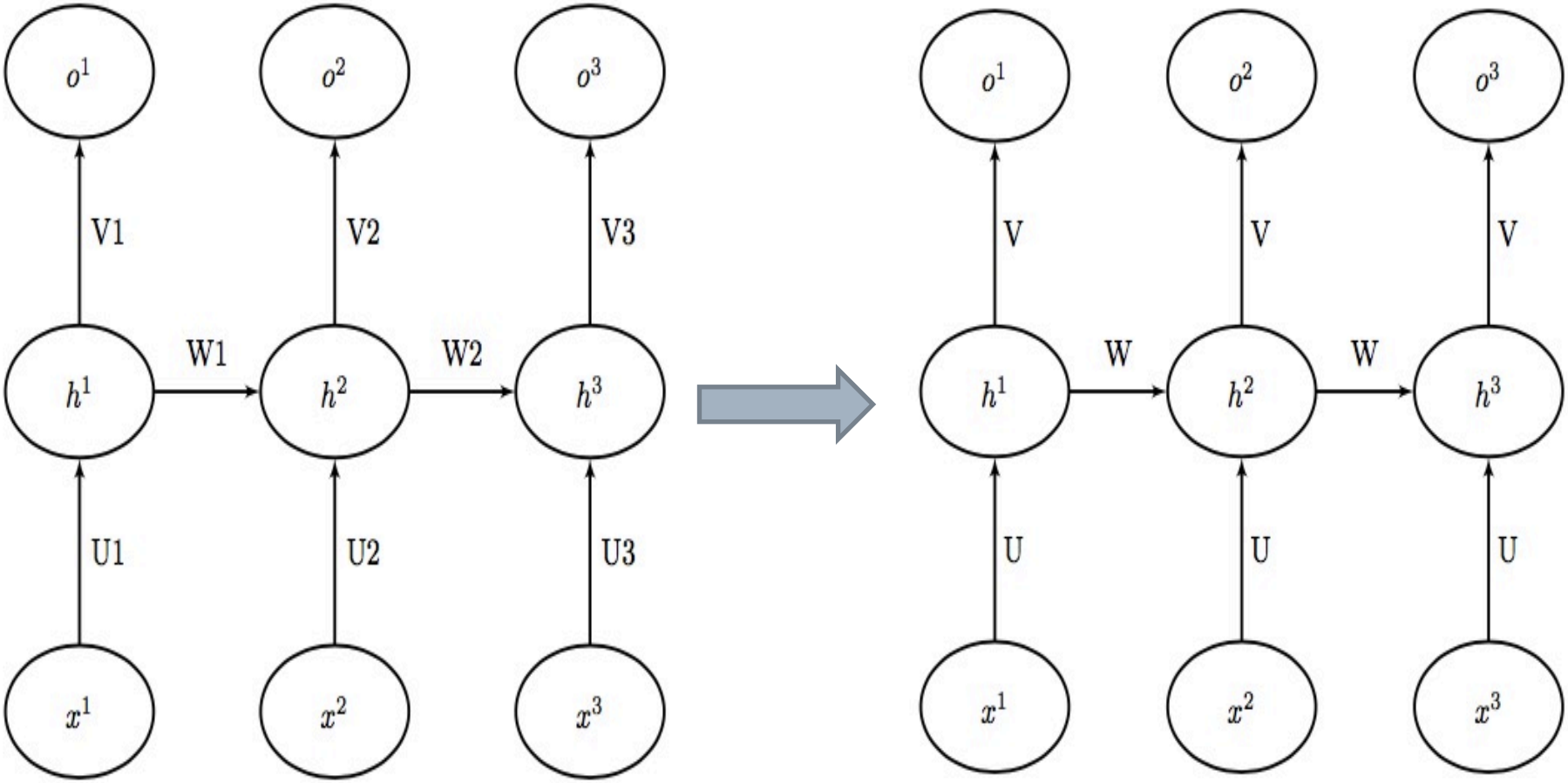
上のネットの表記



重みは、皆違う

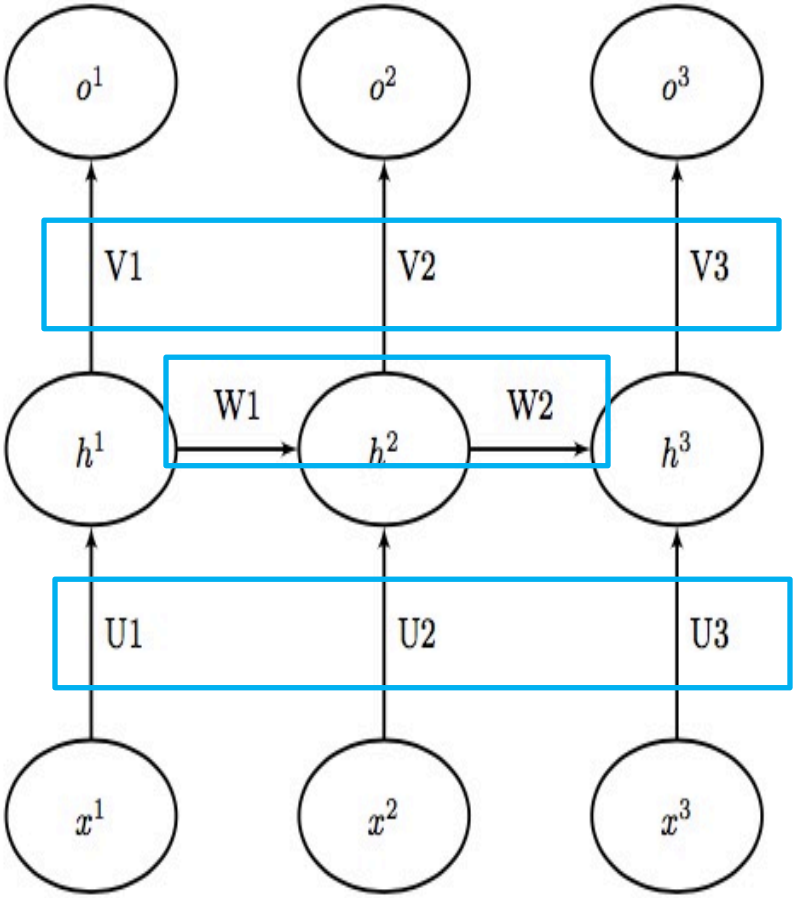
- ところで、前にも述べたように、単純なニューラルネットを横につなげたものがRNNではないのだ。
- 次の図の右の方を見て欲しい。そこには、パラメーターは、***U, V, W*** の三つしかない。
- 左の図の ***U1=U2=U3=U, V1=V2=V3=V, W1=W2=W*** としたものが、右のグラフである。これが、RNNのグラフになる。
- RNNは、同じパラメーターを持つ、同じニューラル・ネットワークを横につなげたものだ。この図では、三つのニューラルネットが横につながっているように表記されているが、この三つのニューラルネットは、同じパラメーターを共有している同一のネットワークである。

パラメーターを共有する

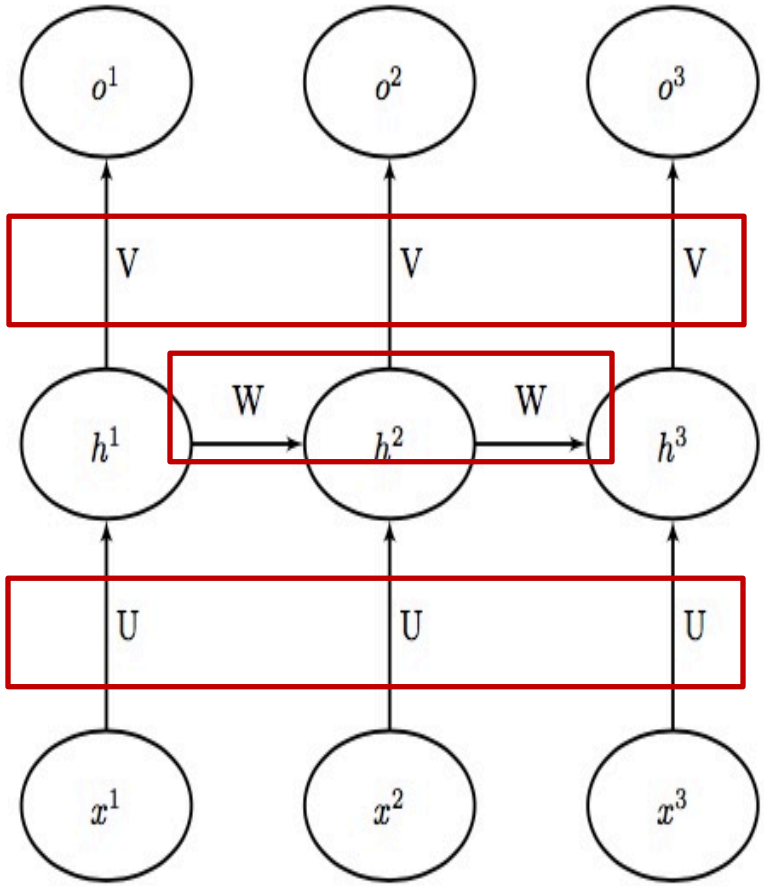


パラメーターを共有する

重みは、皆違う



重みは、皆同じ

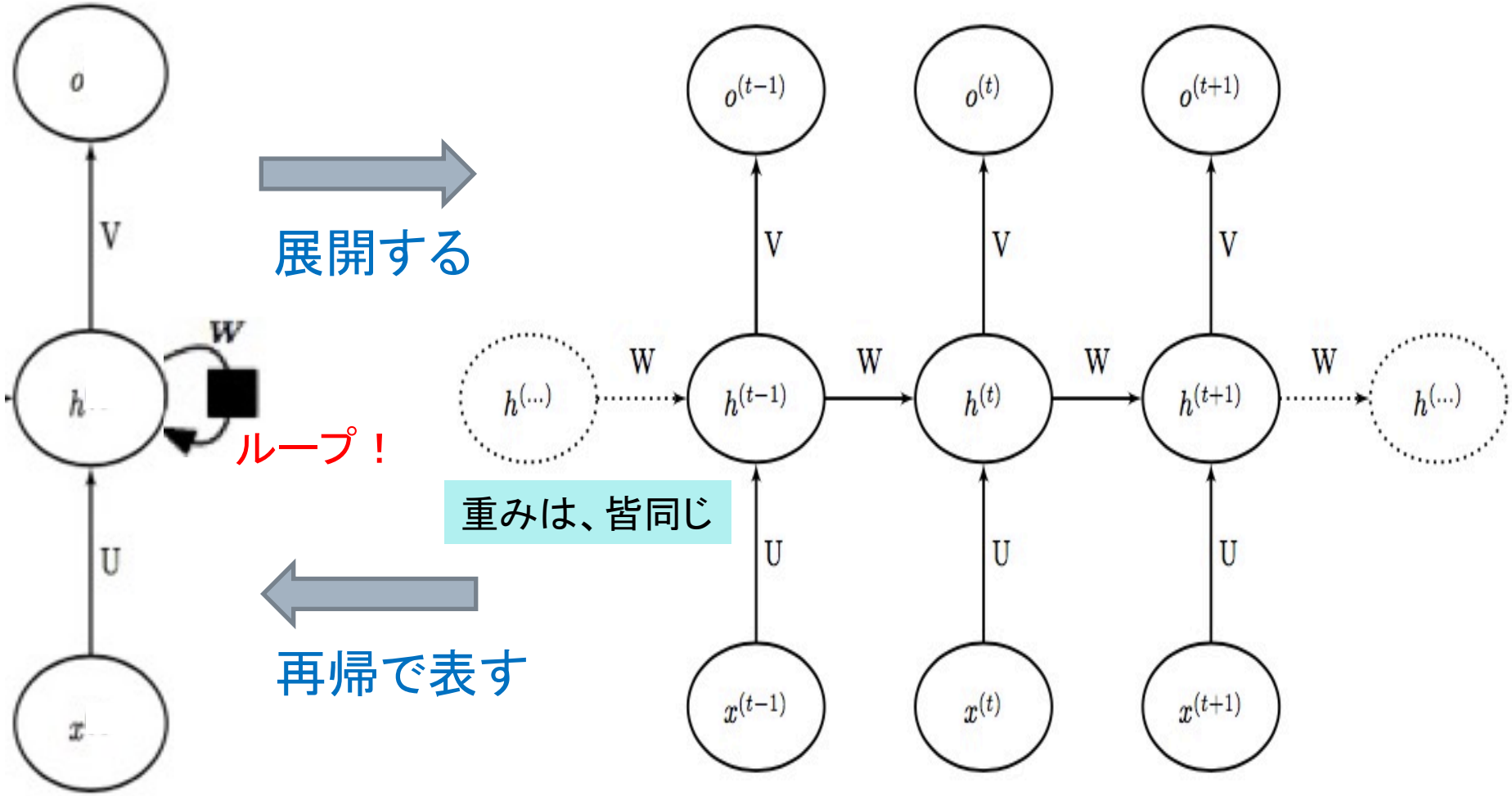


こちらが、RNN

見方を変えてみよう。再帰形の表記

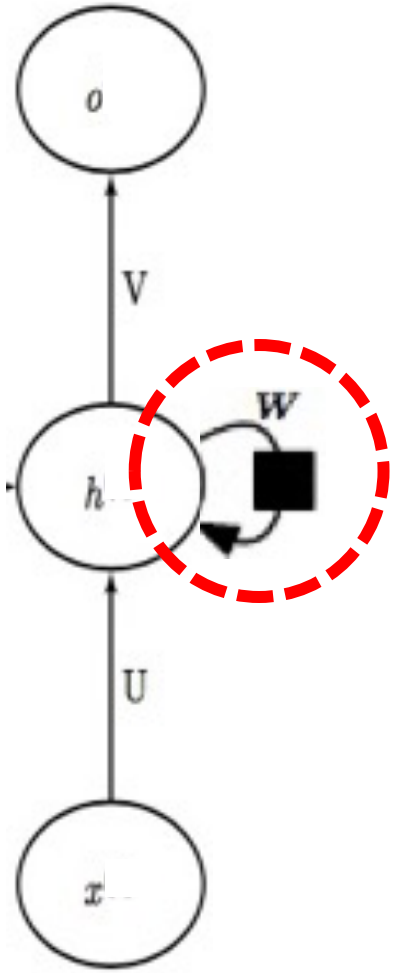
- もしも、先の図で、入力層からの三つの入力が「同時」に与えられることがなく、「別々」に与えられるなら、この三つのネットワークは、一つのネットワークの別々の場合の状態を表していると考えることができる。
- 次の図の左を見て欲しい。これが、RNNの一つの表示である。一つのネットワークだ。これを再帰形という。
- 注目して欲しいのは、隠れ層***h***の出力は二つに分かれていて、出力層***o***に向かうものと、隠れ層***h*自身**に向かうものがあるということ。ループがあるのだ。Full ConnectのFeed Forwardのネットワークには、ループは存在しなかった。このことは、RNNの大きな特徴の一つである。（Back Propargationは、Feed Back Loopと考えていいのだが。）

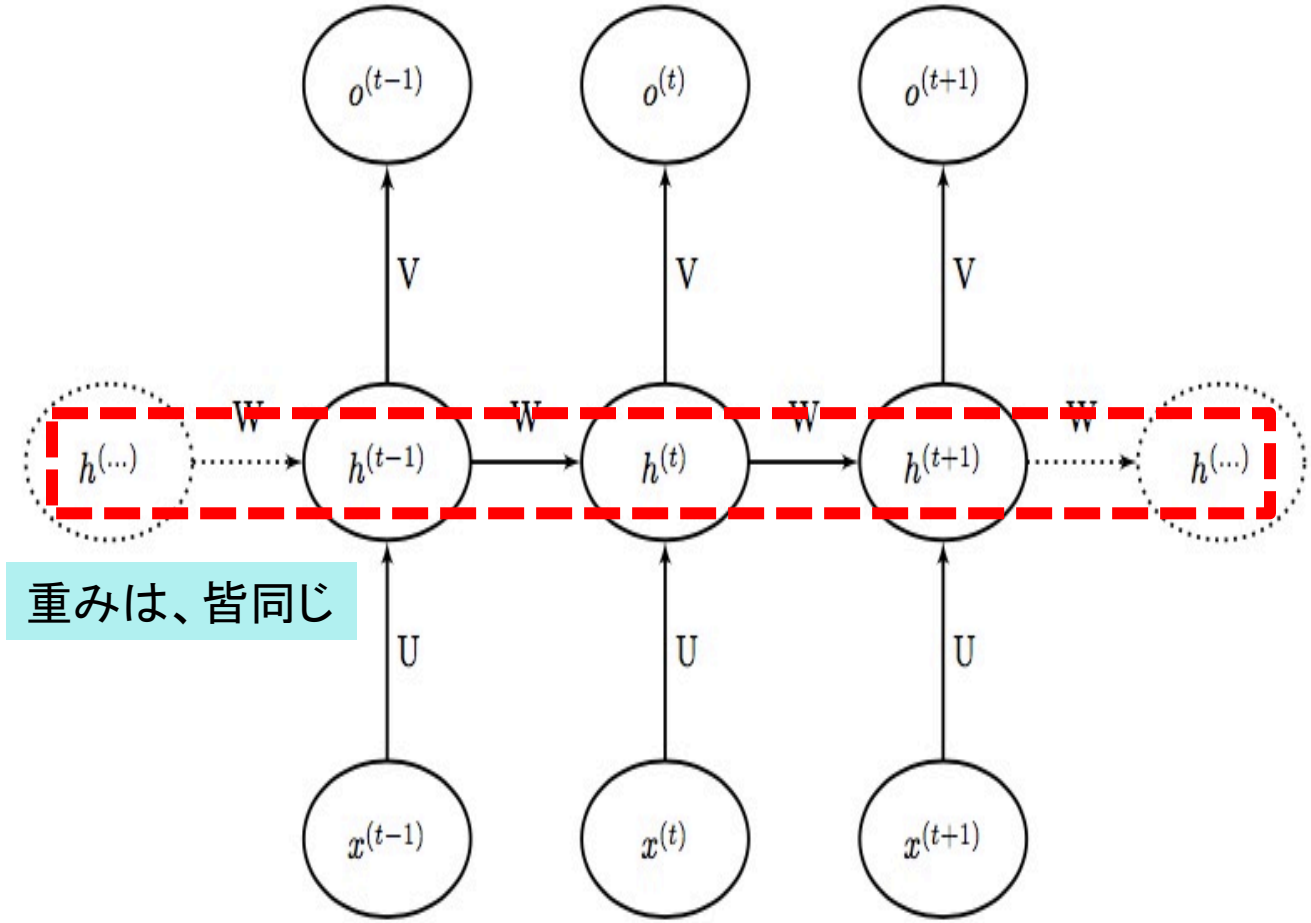
再帰のループを、繰り返しの形に展開する



繰り返しの形を、再帰で表現する

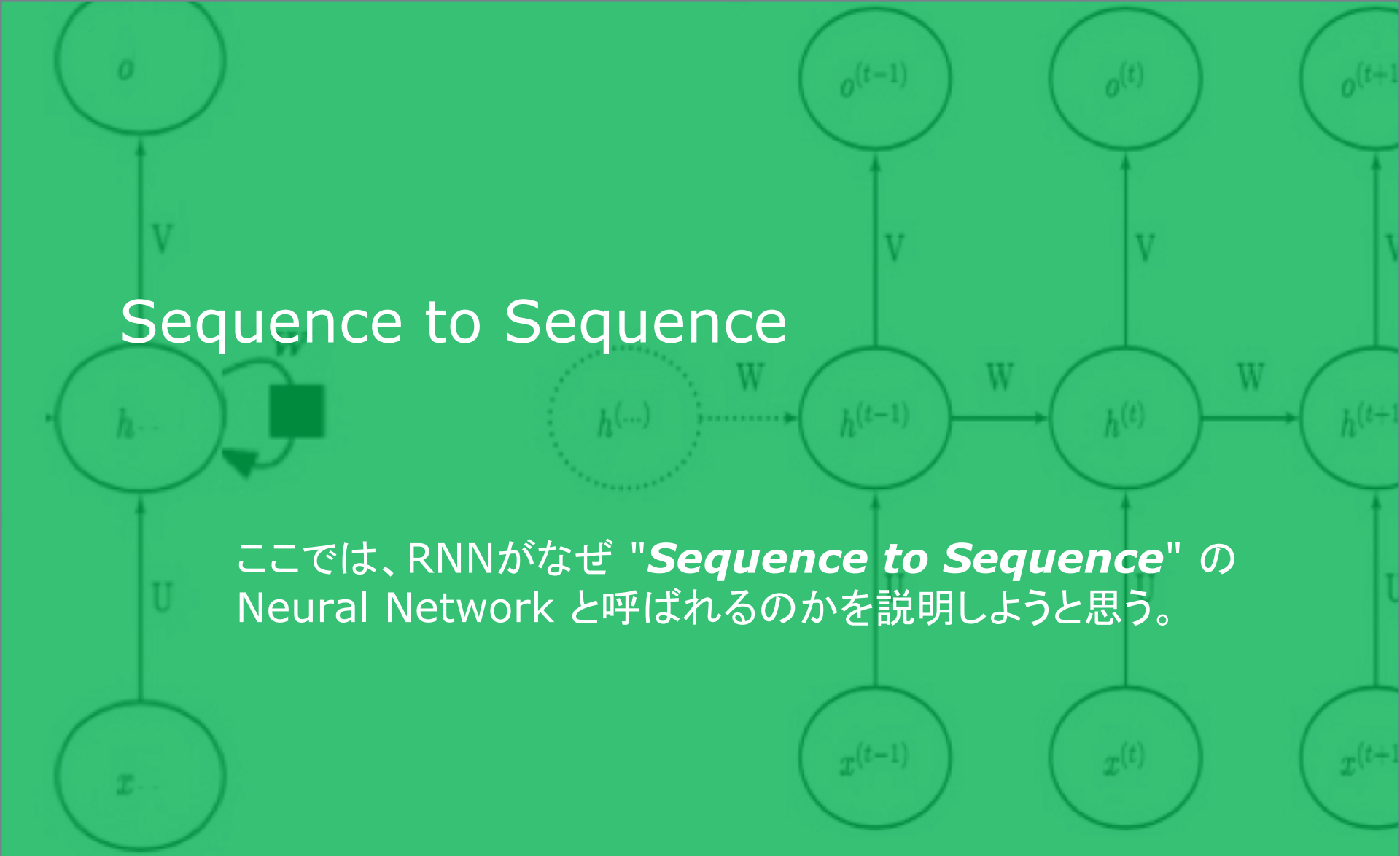
再帰形の「ループ」は、展開形の「横串」に対応する





重みは、皆同じ

Sequence to Sequence



ここでは、RNNがなぜ "***Sequence to Sequence***" の Neural Network と呼ばれるのかを説明しようと思う。

展開形と再帰形、二つの表示は同じものである

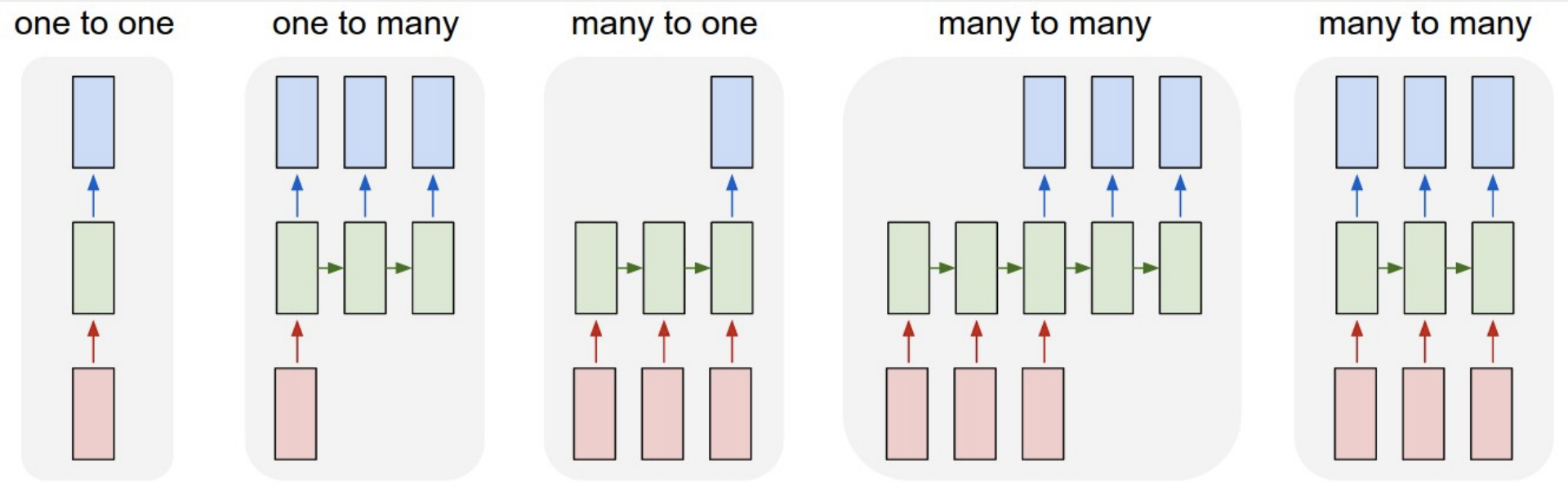
- このー つのネットワークに「別々」に、というか、次々と入力が与えられると、このー つのネットワークの状態は、次々に変化していく。こうした状態の変化を、入力ごとに表現したものが、今まで見てきた右側の図だと考えればいい。これらは、基本的に、同じものを表現している。
- 確かに、右の表示でも、隠れ層***h***の出力は二つに分かれていて、出力層***o***に向かうものと、隠れ層***h***（ただし、隣の）に向かうものがある。もちろん、それは、RNNを、隣り合うネットワークの隠れ層同士を結ぶものとして導入してきたので、当然なのだが。ただ、そのイメージは、RNNの実装を考える時には、捨てたほうがいい。一方で、RNNの動作をイメージするには、こうした展開形が、ずっとわかりやすい。

Sequence – 順序を持つ継起

- 先にも触れたが、左の表示と右の表示が、同じものになるためには、右側の表示の入力が、「同時」には与えられないことが、前提である。
- 入力と状態の変化が、時間上で起きると考えるのは自然である。先の図でも、添え字に用いられている ***(t-1) -> (t) -> (t+1)*** は、時間の変化に対応していると考えることもできる。ただ、本当に重要なことは、この変化が、あとさきの「順序」を持って継起することである。時間は、順序を持つ継起の一つの身近な例にすぎない。
- こうして、RNNは、順番を持つ入力 ***x1, x2, x3, ...*** を、出力 ***o1, o2, o3, ...*** に変換するのだが、この出力 ***o1, o2, o3, ...*** も、当然、順番を持ち、順番を保つ。Sequence ***x1, x2, x3, ...*** は、Sequence ***o1, o2, o3, ...*** に変換されることになる。

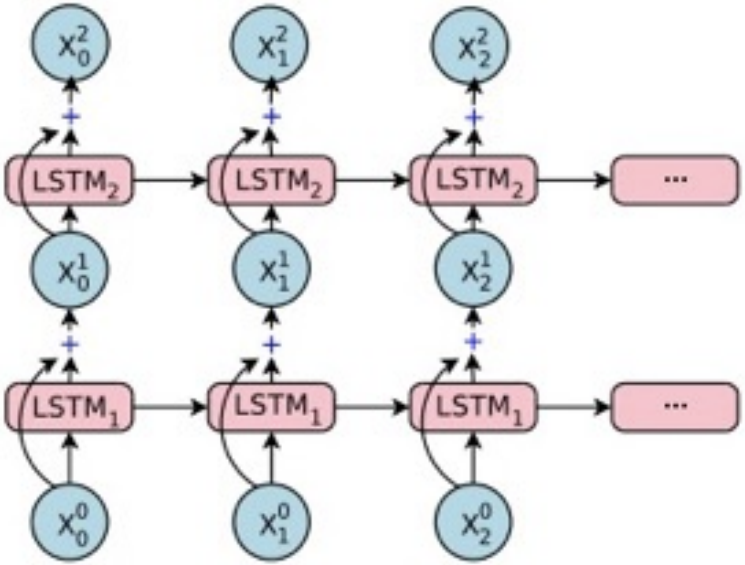
RNNのバリエーション

- RNNには、下図に見るように、様々なバリエーションがある。
  これまで見た”Sequence to Sequence”は、ここで
  の”many to many” の一例である。

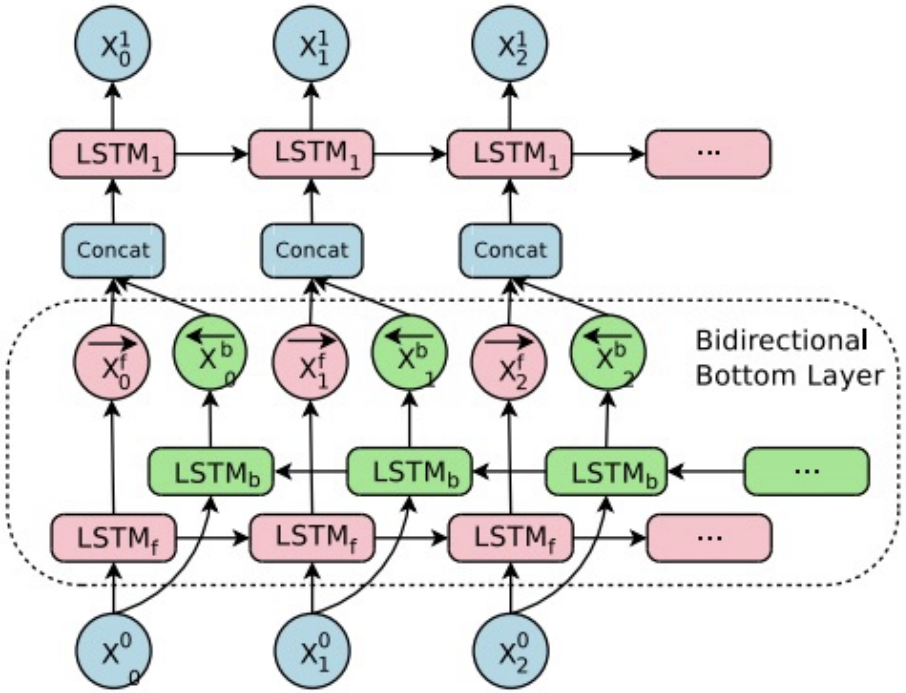


RNNのバリエーション

LSTMを上下二段に重ねたもの



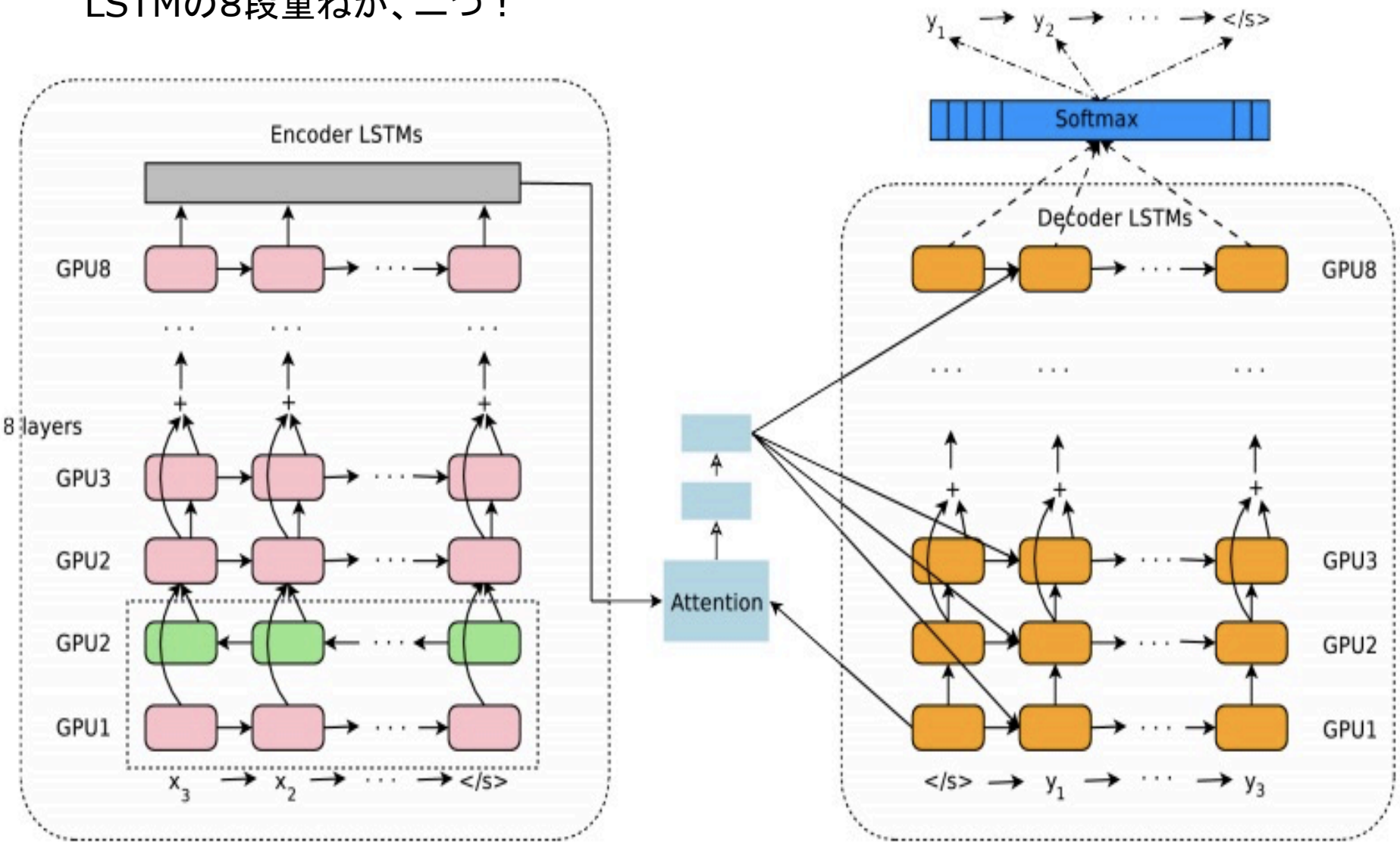
こちらは、LSTMを上下三段に重ねたもの
しかも、下二段は、逆方向に情報が流れる。



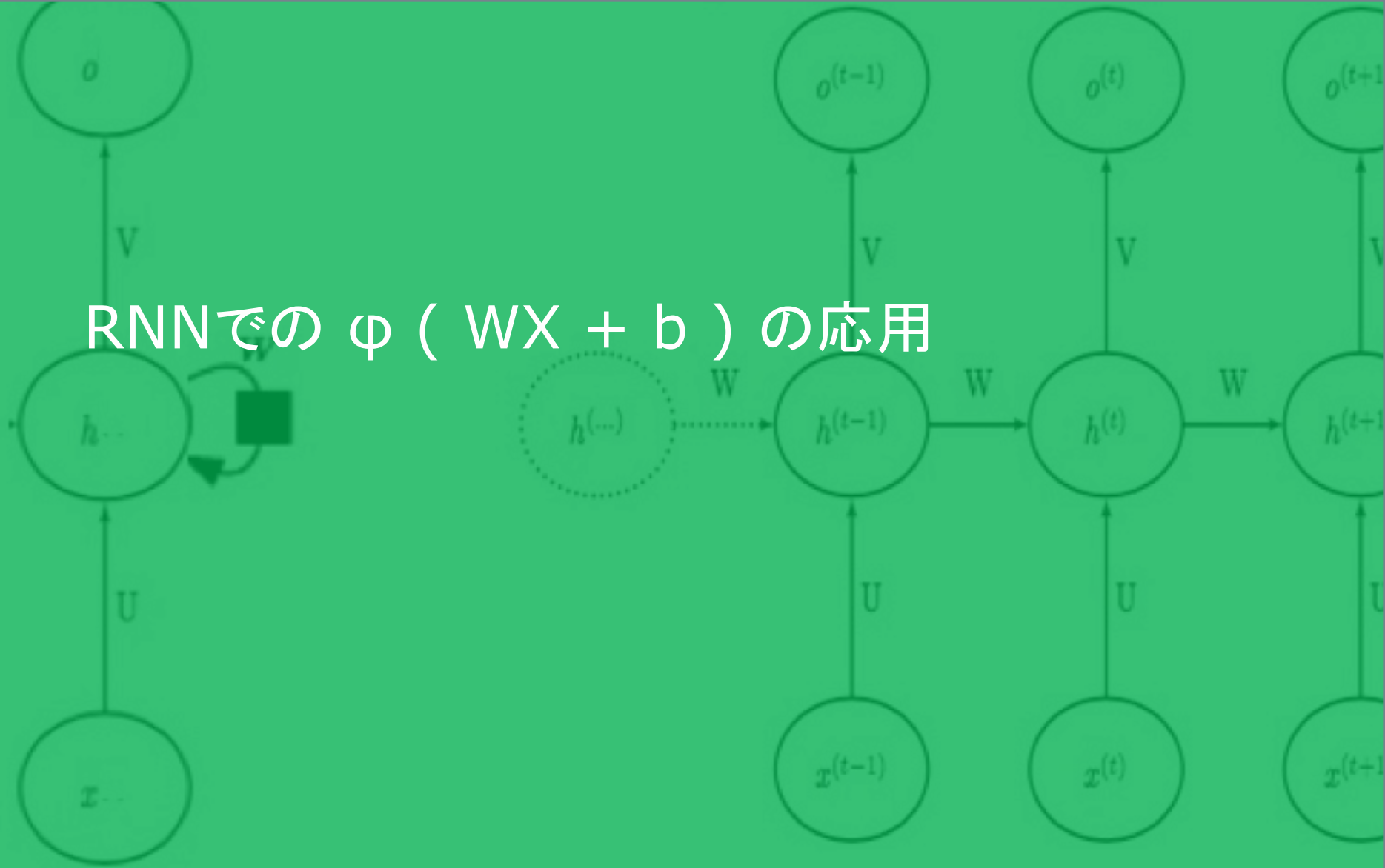
RNNのバリエーション

これがGoogle翻訳の
アーキテクチャーだ

LSTMの8段重ねが、二つ！



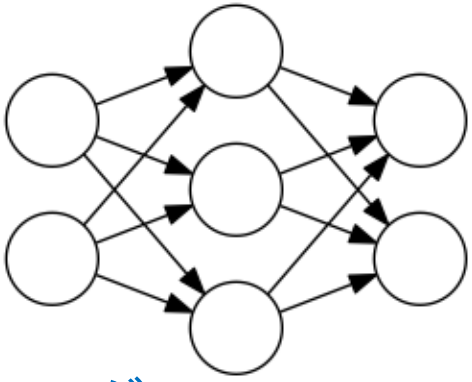
RNNでの φ（ WX + b ）の応用



基本式　φ（ WX ＋ b ）

- 先に、次のようなグラフの表記法を導入した。（繰り返しになるが、次図の一番下）
- この表記では、大きな流れは分かりやすくなるのだが、実際に、どのような処理がなされているのかは、この表記ではわからない。その点では、TensorFlowでのグラフ表記の方が、細かいところまでよくわかった。
- このTensorFlowのグラフが表しているのは、フルコネクトのニューラルネットのある層の出力は、***φ (WX + b)*** という形で表せるということである。ここに***φ*** は、その層の活性化関数、***W*** はその層の重み、***b*** はその層のバイアス、***X*** はその層への入力である。
- この計算式 ***φ (WX + b)*** は、とても基本的なものである。

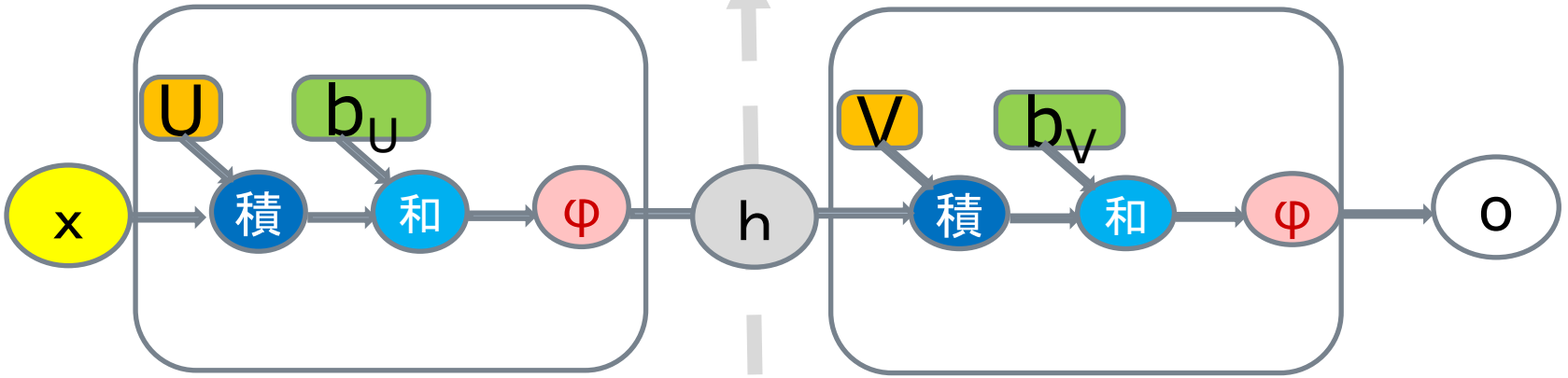
グラフの表記を変える



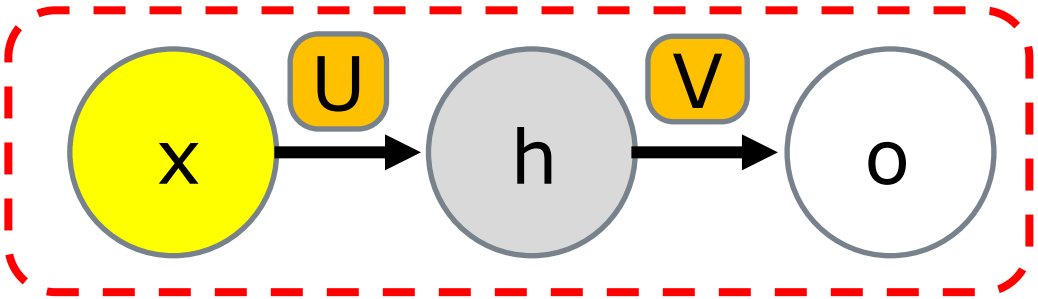
入力層・隠れ層・出力層からなる単純なネットワークがあったとしよう。

TensorFlowでのDNNのグラフ

TensorFlowのグラフで書けば次のようになる。



グラフの表記を次のように変える



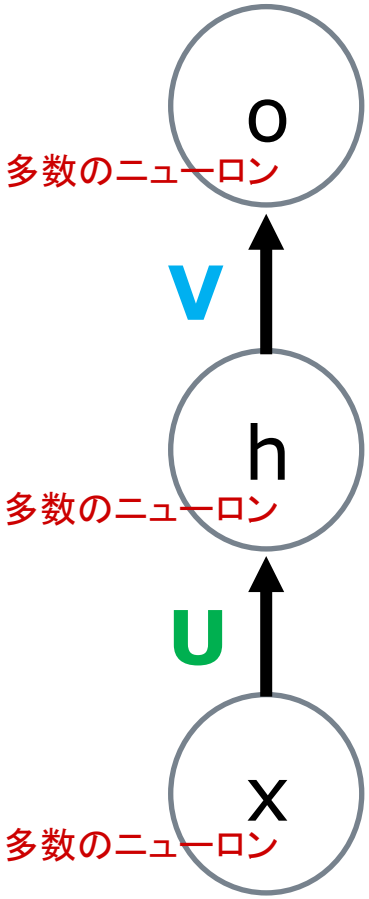
上のグラフの細部を省略してU,Vのみを残して、入力ベクトルxが、出力ベクトルoに変換されることを左のように表す。

新しいグラフ表記を式で表す

- ここでは、復習を兼ねて、三層からなる単純なネットワークの新しいグラフ表記を、この式で表してみよう。それが、次の図である。
- 内容的には、先の図で、TensorFlowのグラフが表しているものと同じである。ただ、グラフが表しているものを式で表すと、細かなところまではっきりとわかる。
- これは大事なことで、RNNのような複雑なネットワークでは、グラフだけすますと、肝心のところがわからなくなる。グラフと式を併用するのがいい。もちろん、基本的な情報は、式の方にある。

単純なネットワークでの
$\varphi (WX + b)$ の復習

φ：活性化関数
W：重み
b：バイアス
X：入力



$$o = \varphi_o(Vh + b_o)$$

o ：出力層のベクトル
φ_o：出力層の活性化関数
V ：出力層の重み
b_o：出力層のバイアス
h ：隠れ層のベクトル

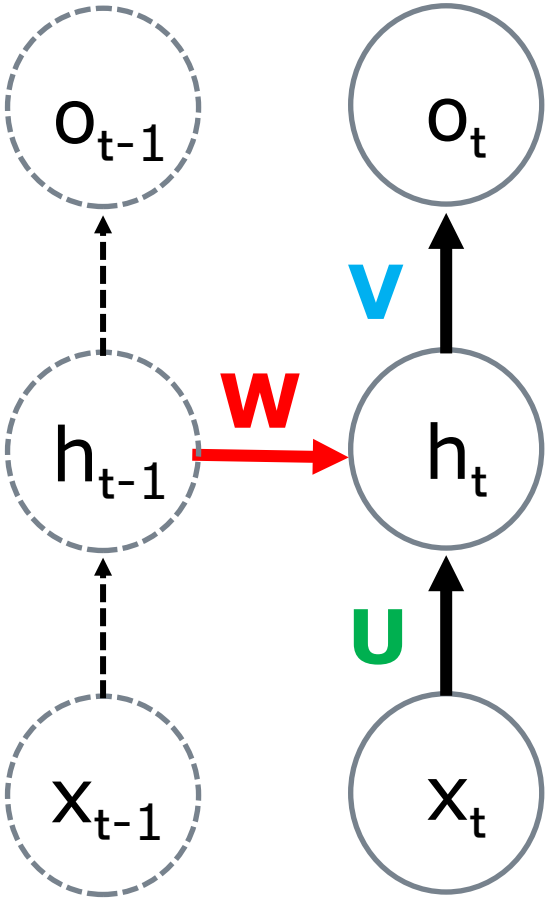
$$h = \varphi_h(Ux + b_h)$$

h ：隠れ層のベクトル
φ_h：隠れ層の活性化関数
U ：隠れ層の重み
b_h：隠れ層のバイアス
x ：入力層のベクトル

RNNのグラフ表記を式で表す

- 出力層の式は同じだが、隠れ層の式に、ちょっとした変化がある。それは、RNNの隠れ層が、入力層からだけでなく、隣の隠れ層からの入力も受けるので当然のことである。
- 上の図で、赤い字で書かれた部分が、それである。
- RNNでは、隣り合う隠れ層は、フルコネクトでつながっている。隠れ層同士のつながりの重みを***W***とすれば、隣の隠れ層 ***h(t-1)***から受け取るベクトルは、***W***と***h(t-1)***をかけたものになる。それは、隠れ層が、重み***U*** でフルコネクトでつながっている入力層 ***x(t)***から受け取るベクトルが、***U***と***x(t)*** をかけたものになるのと同じことである。
- こうして、RNNの隠れ層が外から受け取るベクトルは、入力層からくるベクトルに、隠れ層からくるベクトルを足したものだと考えればいいことになる。

RNN(展開形)での
$\varphi (WX + b)$ の応用



$$o_t = \varphi_o(Vh_t + b_o)$$

$$h_t = \varphi_h(Ux_t + Wh_{t-1} + b_h)$$

新たに追加された項。
隣の隠れ層の寄与分。
もちろん、重みxベクトルの形をしている。

Minimal character-level language model with a Vanilla Recurrent Neural Network

```
37    for t in xrange(len(inputs)):
38      xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39      xs[t][inputs[t]] = 1
40      hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41      ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
42      ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43      loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
```

$$hs_t = tanh(Wxh \cdot xs_t + Whh \cdot hs_{t-1} + bh)$$
$$ys_t = Why \cdot hs_t$$

https://goo.gl/5npVuw

Back Propagation

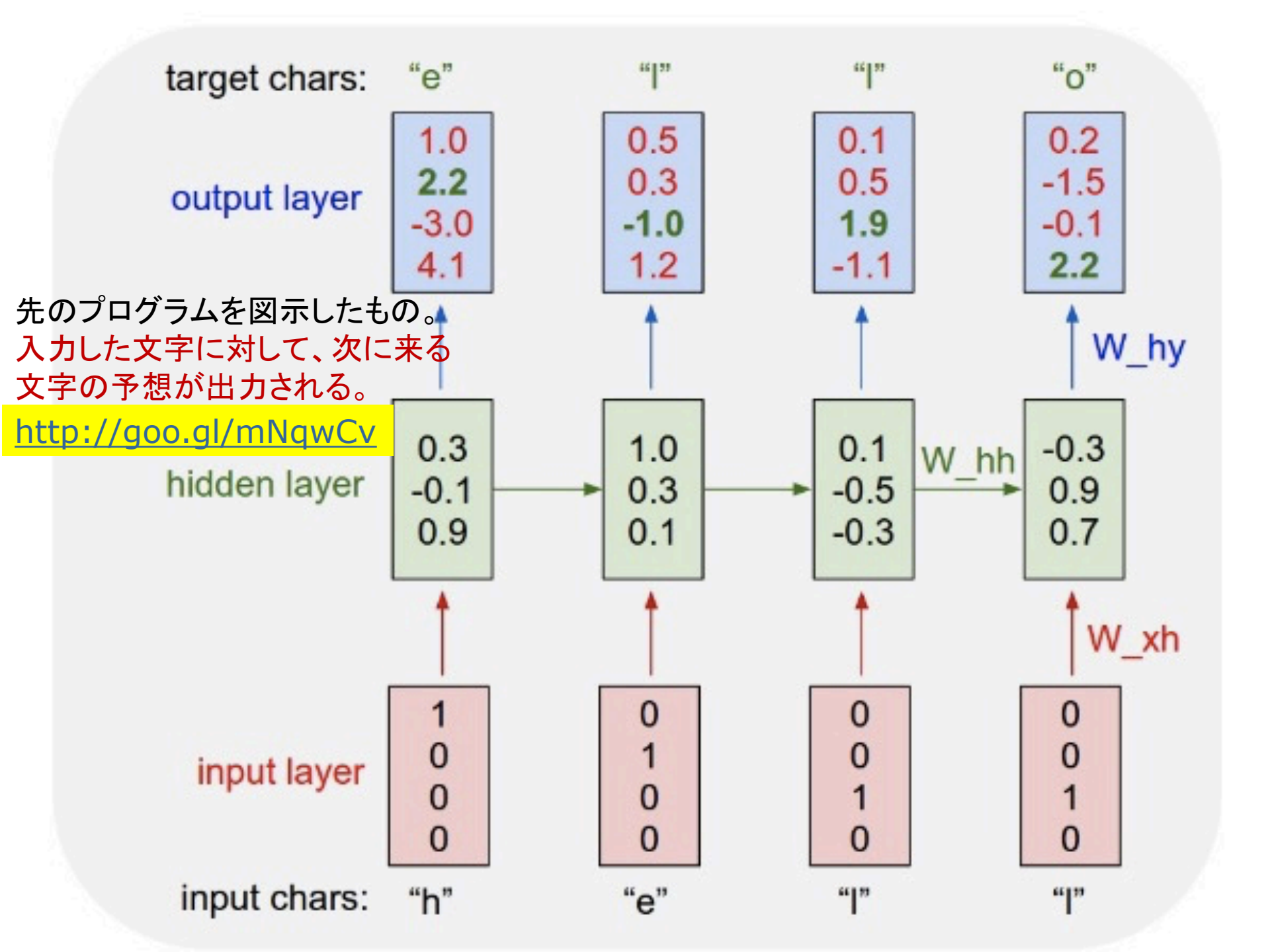
$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad L_i = -\log(p_{y_i})$$

$$\frac{\partial L_i}{\partial f_k} = p_k - \mathbb{1}(y_i = k)$$

```
48    for t in reversed(xrange(len(inputs))):
49      dy = np.copy(ps[t])
50      dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-ca
51      dWhy += np.dot(dy, hs[t].T)
52      dby += dy
53      dh = np.dot(Why.T, dy) + dhnext # backprop into h
54      dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
55      dbh += dhraw
56      dWxh += np.dot(dhraw, xs[t].T)
57      dWhh += np.dot(dhraw, hs[t-1].T)
58      dhnext = np.dot(Whh.T, dhraw)
59    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
60      np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

$$\frac{\mathrm{d}}{\mathrm{d}x}\tanh(x) = 1 - \tanh^2(x)$$

http://cs231n.github.io/neural-networks-case-study/#grad

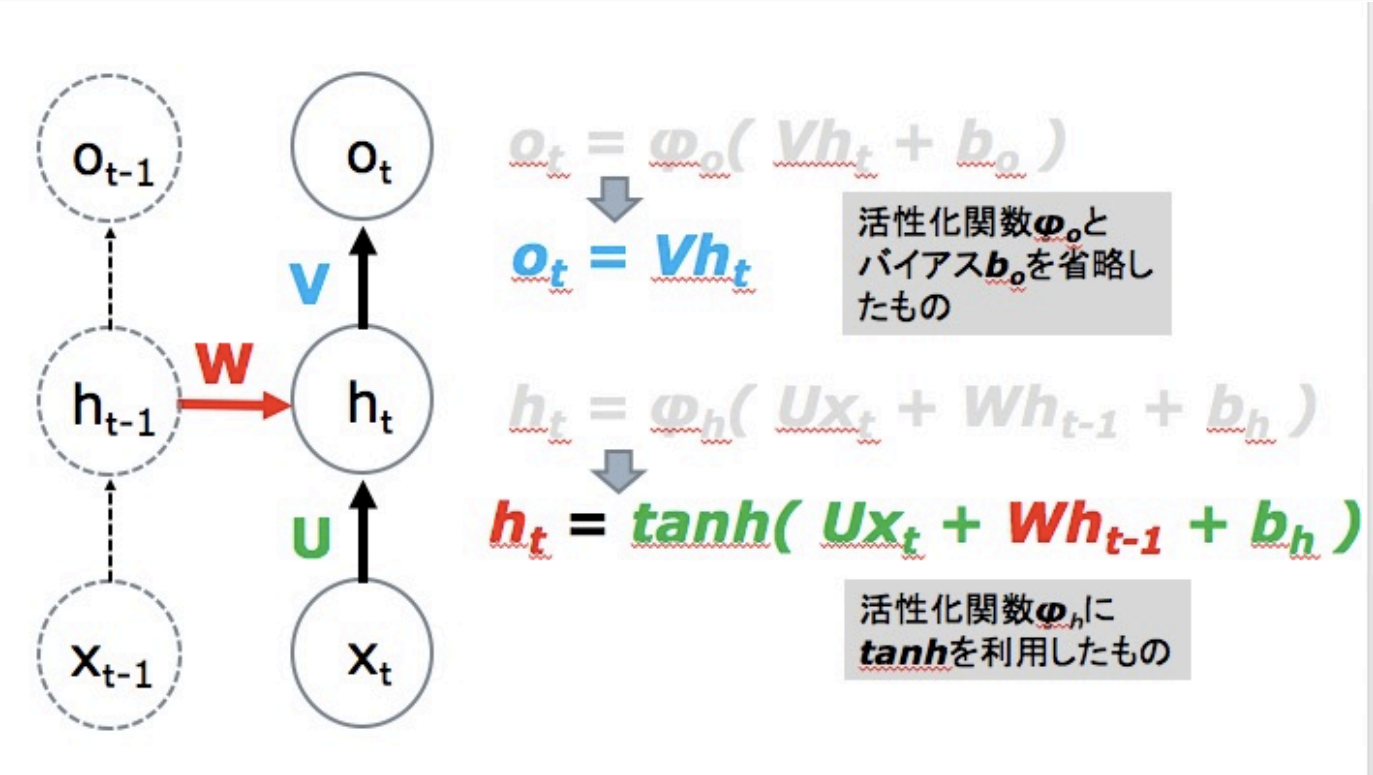

target chars:
"e"
"l"
"l"
"o"
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
先のプログラムを図示したもの。
入力した文字に対して、次に来る
文字の予想が出力される。
http://goo.gl/mNqwCv
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
"h"
"e"
"l"
"l"

20数年前、いったん放棄されたRNN

- テキストの文字列や音素のつらなりである人間の発話、あるいは音楽の音やビデオのフレームのようなSequentialなデータは、我々の周りに多数存在している。こうしたSequentialなデータを、単純なニューラル・ネットを横に連ねたニューラル・ネットで解析しようとするRNNは、当初、大きな関心を集めたようだ。今から、20数年前のことだ。
- 隠れ層をつなぐ接続を同じ重みのパラメーター $\boldsymbol{W}$ を共有し、同一のネットワークを繰り返し利用するというアイデアで、実装の負担も抑えられそうに見えた。

かつてのRNN

- 当時のRNNは、次のような形をしていたらしい。先のコードと同じである。



この式に問題があったわけではない。この構成のRNNは、繰り返しの少ない短い系列なら、ちゃんと動くのだ。問題は、他のところにあった。

勾配の消失（あるいは、爆発）の問題

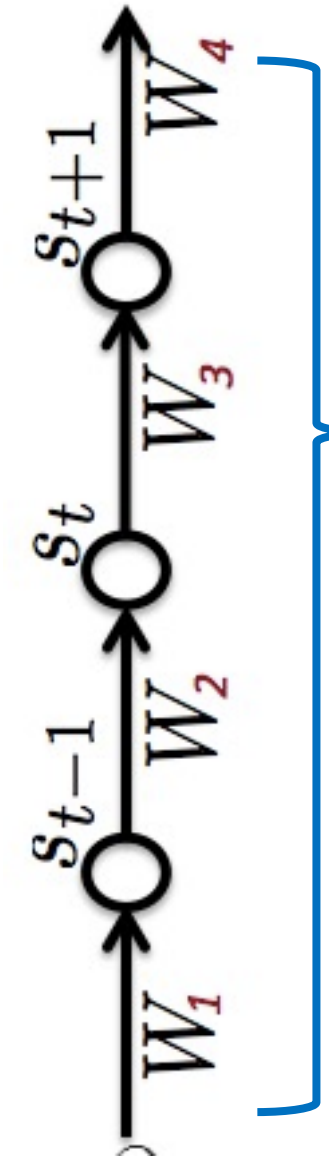
- ところが、この試みはうまくいかなかった。RNNでは、Back Propagation / Gradient Descent を使った学習アルゴリズムがうまく働かないのだ。横に並ぶネットワークの数が増えるにつれて、パラメーターを修正する勾配が、どんどんゼロに近づいてゆくのだ（時には、爆発的に増大することもあった）。「勾配の消失（あるいは、爆発）の問題」という。

DNNとRNNの違い

- Full ConnectのDNNとRNNとでは、単純なニューラルネットを、縦に並べる、横に並べるだけではない違いがある。
- Full ConnectのDNNでは、各層ごとの重みのパラメーター $\boldsymbol{W_i}$ は、一般には異なっている。RNNでは、隠れ層を結ぶ重みのパラメーター $\boldsymbol{W}$ は、常に同じである。DNNの場合には、勾配の消失が起きそうになったら、一つ一つの層の「学習率」を個別に修正して、問題に対応できる。これは、いささか姑息な手段だが、DNNでは有効である。しかし、この方法は、常に全く同じ形でパラメーターを適用するRNNには、適用できない。
- さらに、一般的には、DNNでの縦への積み重ねより、RNNでの横の繰り返しは長くなる傾向がある。50段重ねのDNNは、あまり見たことないが、50文字以上の文字列を扱うRNNを作ろうとするのは自然である。

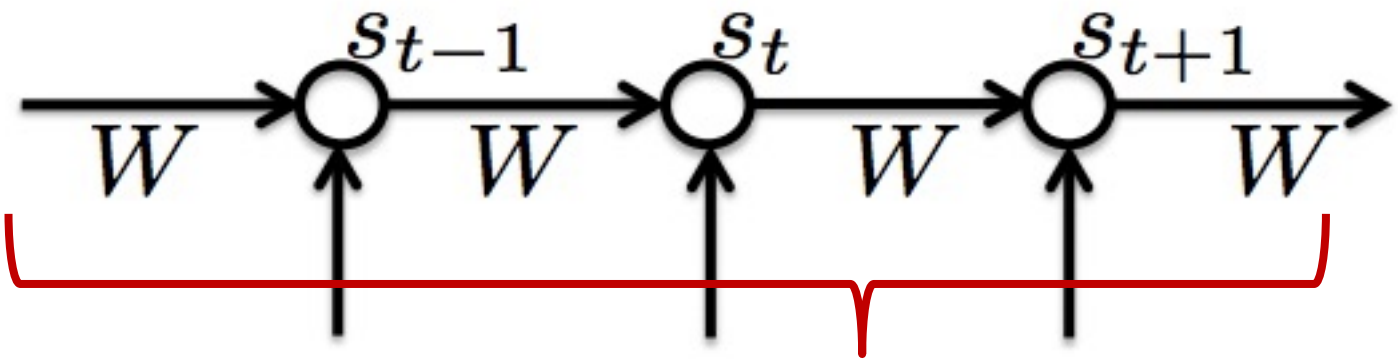
Full Connect

DNN



パラメーターW_iが違う

RNN



パラメーターが同じW

損失関数は、先行する状態の再帰的な関数

$$L = L(s_T(s_{T-1}(\ldots s_{t+1}(s_t, \ldots))))$$

$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T}\frac{\partial s_T}{\partial s_{T-1}}\cdots\frac{\partial s_{t+1}}{\partial s_t}$$

損失関数の微分
(合成関数の微分)

この積が、どんどん長くなる

Bengioの論文

- いろいろやって見たらしいが、実験結果は、思わしくなかった。
- そこにある論文が登場する。1994年、Bengioらが、"Learning Long-Term Dependencies with Gradient Descent is Difficult" という論文を発表する。この論文は、長いRNNでの学習の難しさを、理論的に証明したものだった。
- 実は、それより早く、1991年に、Hochreiter がドイツ語で書いた修士論文で、同じ証明を与えていたことが、現在では知られている。
- これらの論文で、RNNの学習の困難の理論的根拠が明らかにされたことを受けて、当時のニューラルネットの世界では、RNNの探求は、事実上、放棄されることになった。

RNNに内在する困難

"Learning Long-Term Dependencies with Gradient Descent is Difficult"

Yoshua Bengio et al.

https://goo.gl/lGBWsP

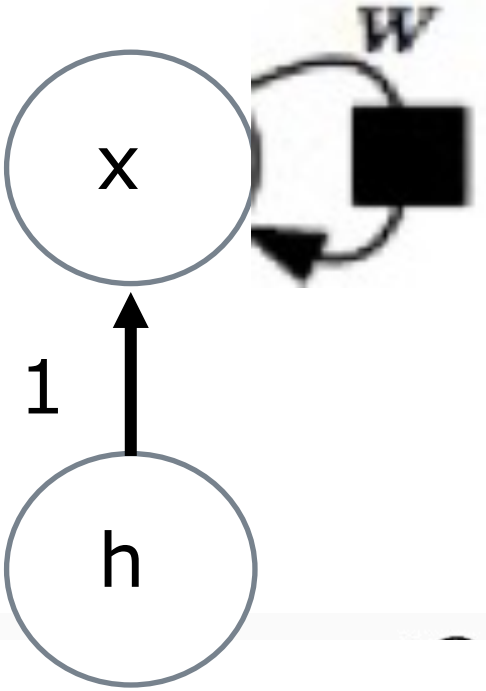
1994年

D. Discrete Error Propagation

The analysis of Section IV could suggest that the root of the problem lies in the essentially discrete nature of the process of storing information for an indefinite amount of time. Indeed, the gradient backpropagated through time vanishes when the system stays in the same stable state for several time steps. Intuitively, we would like to recover some error information at the time when the input made the system reach that stable state. Instead of propagating a gradient through differentiable units, the algorithm presented here was explicitly designed to propagate discrete error information through units that compute a non-differentiable function, such as a hard threshold. In this way we hope to find algorithms that directly address the problem of propagating error backward in time, even though the process of robustly storing information appears to have a discrete nature.

$x_t = \tanh(1 \cdot h_t + W \cdot x_{t-1})$

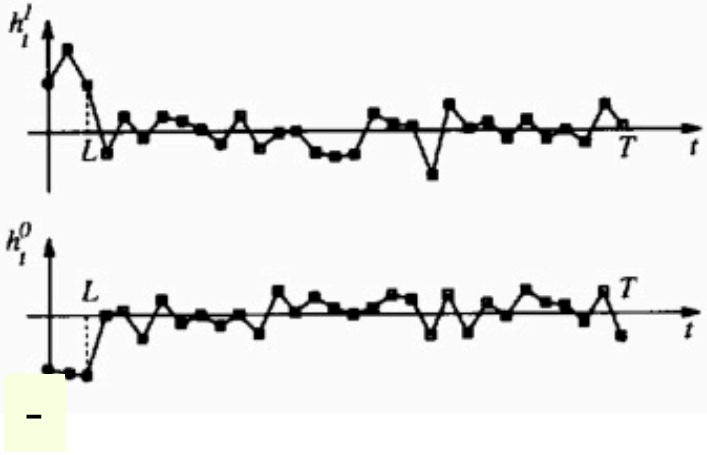
次のような単純化したRNNを考える。

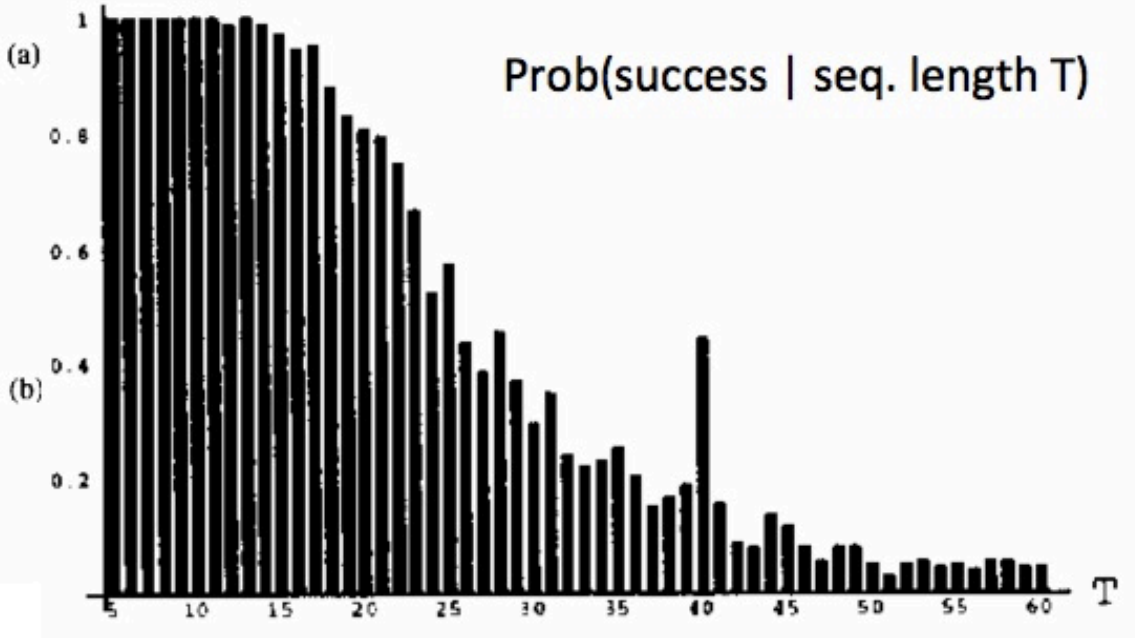


初期の入力の符号（+-）で与えられた情報を、このユニットは、記憶できるか？

Tが大きくなるにつれて、それは、極端に難しくなる。

次のような入力をあたえる。





長期的な依存関係は短期依存性と比較して、
指数関数的に小さな重みを持つ

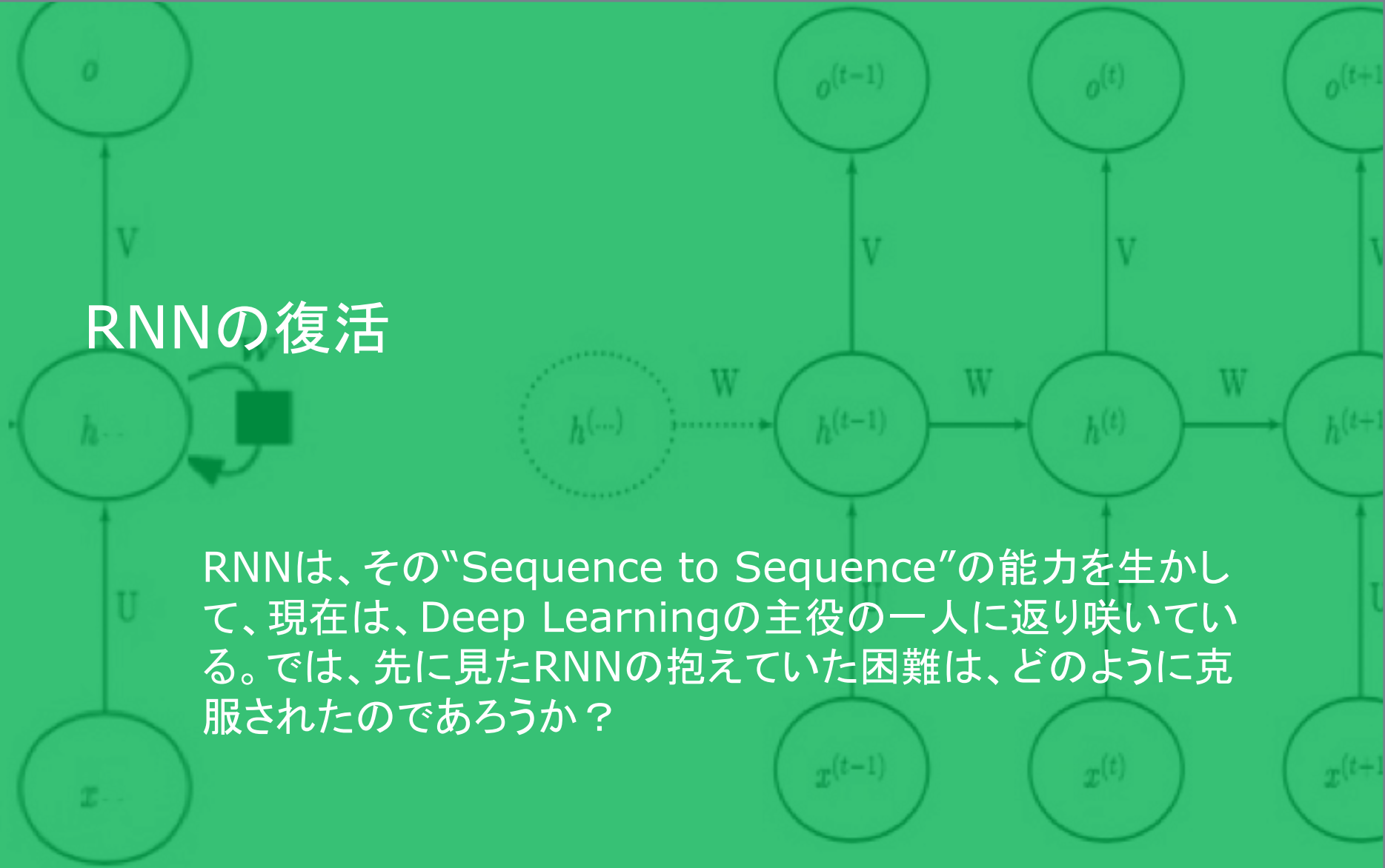
$$\frac{\partial C_t}{\partial W} = \sum_{\tau \leq t} \frac{\partial C_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W} = \sum_{\tau \leq t} \frac{\partial C_t}{\partial a_t} \frac{\partial a_t}{\partial a_\tau} \frac{\partial a_\tau}{\partial W}$$

スペクトル半径＜1の時、
この部分は、より長い
時間差については、指数
関数的に小さくなる

これは、a_Tが、他のより良い値にジャンプすることを可能にするWの変化が存在しても、Wに関するコストの勾配はこの可能性を反映しないことを意味する。

これは、Wの小さな変化は、ほとんどの場合、近い過去（tに近いT）で感じられるためである。

RNNの復活



RNNは、その“Sequence to Sequence”の能力を生かして、現在は、Deep Learningの主役の一人に返り咲いている。では、先に見たRNNの抱えていた困難は、どのように克服されたのであろうか？

- RNNの復活に大きな役割を果たしたのは、皮肉なことに、RNNの抱えている困難を原理的に明らかにし、いわば、かつてRNNに「引導」を渡した二人、Hochreiter とBengioであることに気づく。
- それは、本当は、皮肉なことではないのかもしれない。問題の難しさを一番よく理解している人が、問題の解決に一番近いところにいるのは、ある意味自然なことであるのだから。

- 1997年に、Hochreiter と Schmidhuberが論文 “Long Short Term Memory ”で導入したLSTMは、現在のRNN技術の中核となった。Googleの「ニューラル機械翻訳」もLSTMベースである。LSTMについては、後で、詳しく見てみたいと思う。
- 去年 2016年の6月のICML (International Conference on Machine Learning)で、Bengioが、面白いタイトルの講演をしている。タイトルは、"Learning Long-Term Dependencies with Gradient Descent is Difficult" 。そう、1994年に、彼がRNNの原理的な困難を明らかにした論文と同じタイトルである。

Back to the Future

- ポイントは、この講演が、このカンファレンスの “Neural Nets Back to the Future” というワークショップで行われていること。 このワークショップの目的は、次のようなものだ。
- 「ディープ・ラーニングの研究が今日非常に活発なので、私たちは一歩踏み込んでその基礎を調べることができるでしょう。私たちは、ニューラルネットワークに関する過去の研究を批判的に見て、今日の研究との違いをよりよく理解しようと提案します。過去の取り組みは、従うべき有望な方向、回避すべき落とし穴、再考すべきアイディアや前提を、私たちに示すことができます。同様に、今日の進歩は、何が今後も研究されるべきか、何が解決されたのかを、批判的検討することを可能にします…」
- こうした趣旨からいえば、1994年のBengioの論文は、取り上げられてしかるべきものの一つだろう。

Learning Long-Term Dependencies with Gradient Descent is Difficult



Y. Bengio, P. Simard & P. Frasconi, IEEE Trans. Neural Nets, **1994**

June 23, 2016, ICML, New York City
Back-to-the-future Workshop

Yoshua Bengio
Montreal Institute for Learning Algorithms
Université de Montréal

PLUG: **Deep Learning**, MIT Press book in press, Chapters will remain online

Université de Montréal

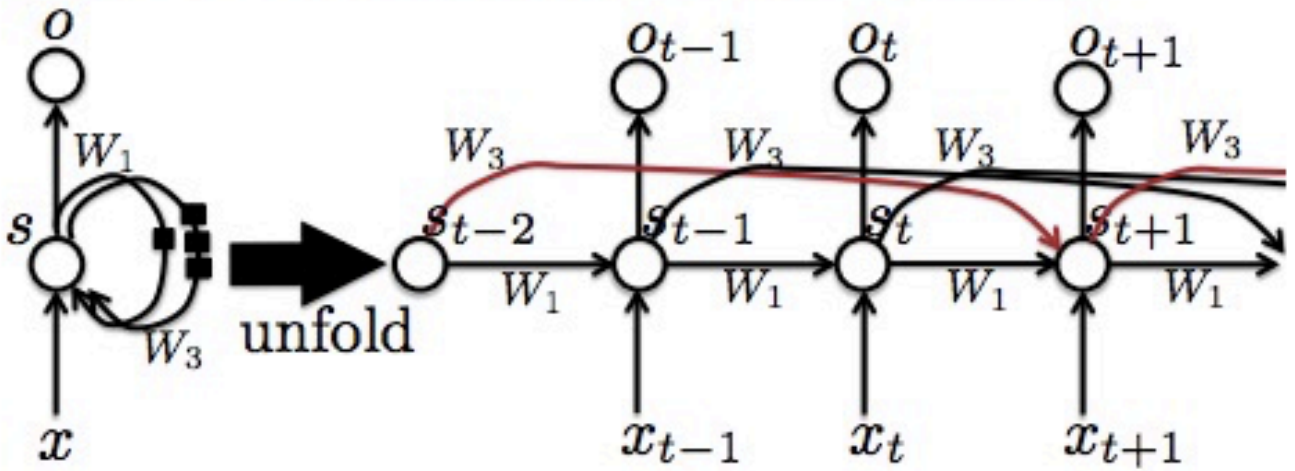
CIFAR | ICRA

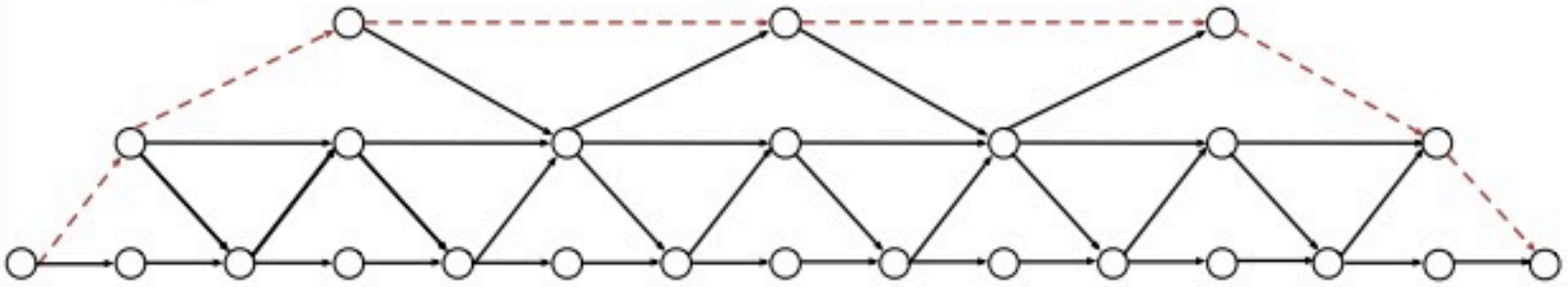
MILA

https://goo.gl/RB3mu9

Bypassing nonlinearities to learn longer term dependencies

- Delays *(Lin et al & Giles 1995)*
- Multiple time scales *(Elhihi & Bengio NIPS 1995)*



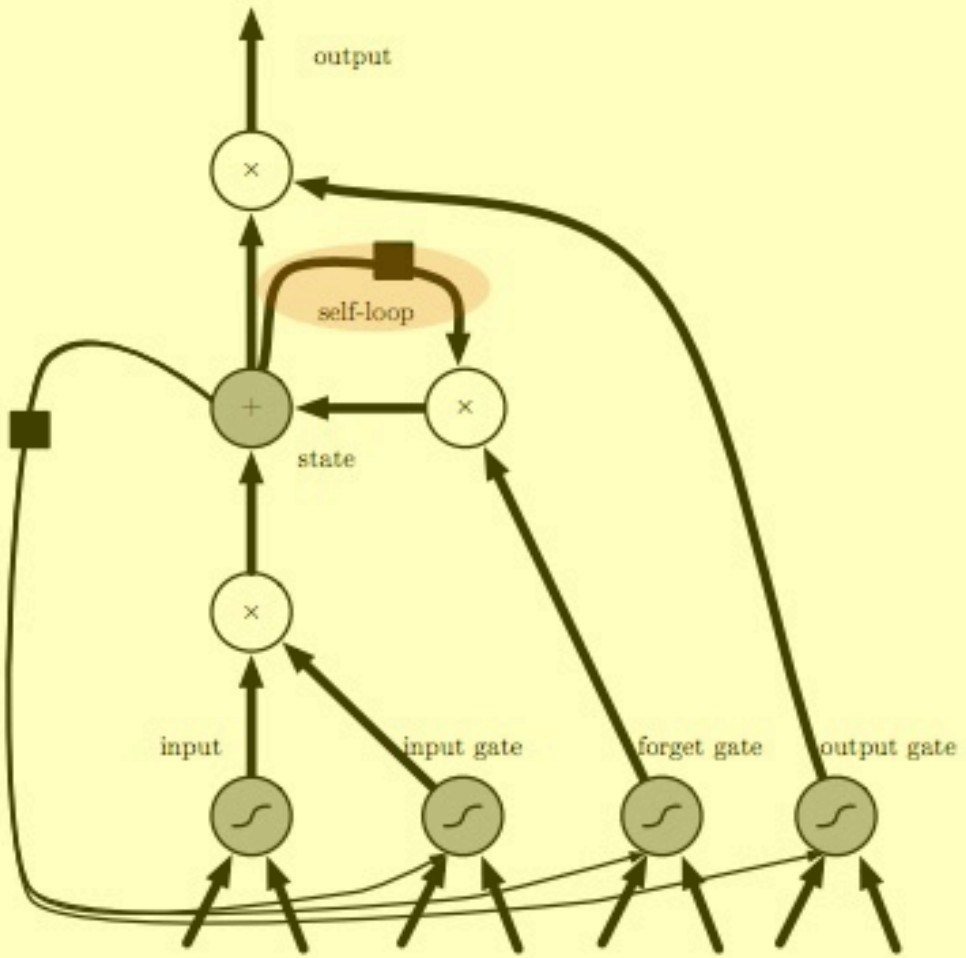


Fighting the vanishing gradient: LSTM & GRU

(Hochreiter 1991); first version of the LSTM, called Neural Long-Term Storage with self-loop

- Create a path where gradients can flow for longer with a self-loop
- Corresponds to an eigenvalue of Jacobian slightly less than 1
- LSTM is now **heavily used** *(Hochreiter & Schmidhuber 1997)*
- GRU light-weight version *(Cho et al 2014)*

LSTM: (Hochreiter & Schmidhuber 1997)



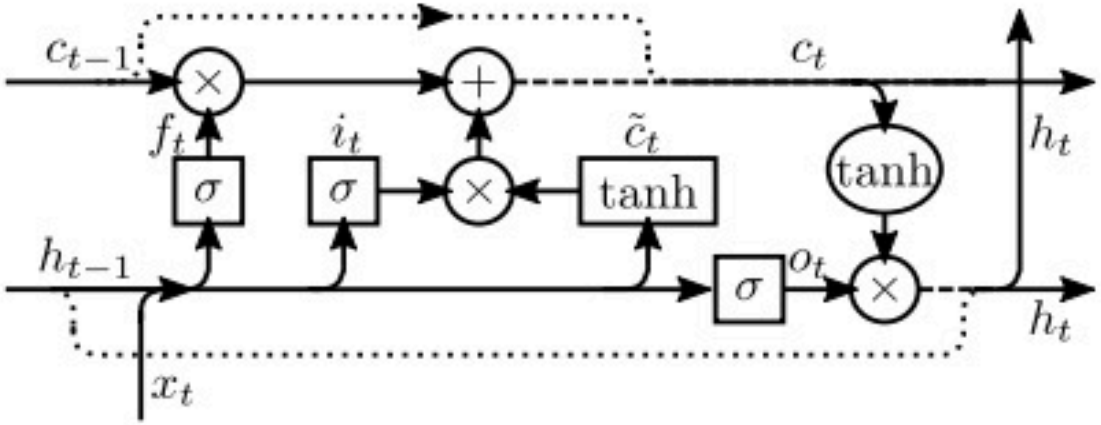
New Ideas to Help Information Propagation

- Unitary matrices: all e-values of matrix are 1

(Arjowski, Amar & Bengio ICML 2016)

$$\mathbf{W} = \mathbf{D}_3\mathbf{R}_2\mathcal{F}^{-1}\mathbf{D}_2\mathbf{\Pi}\mathbf{R}_1\mathcal{F}\mathbf{D}_1$$

- Zoneout: randomly choose to simply copy the state unchanged



(Krueger et al 2016, submitted)

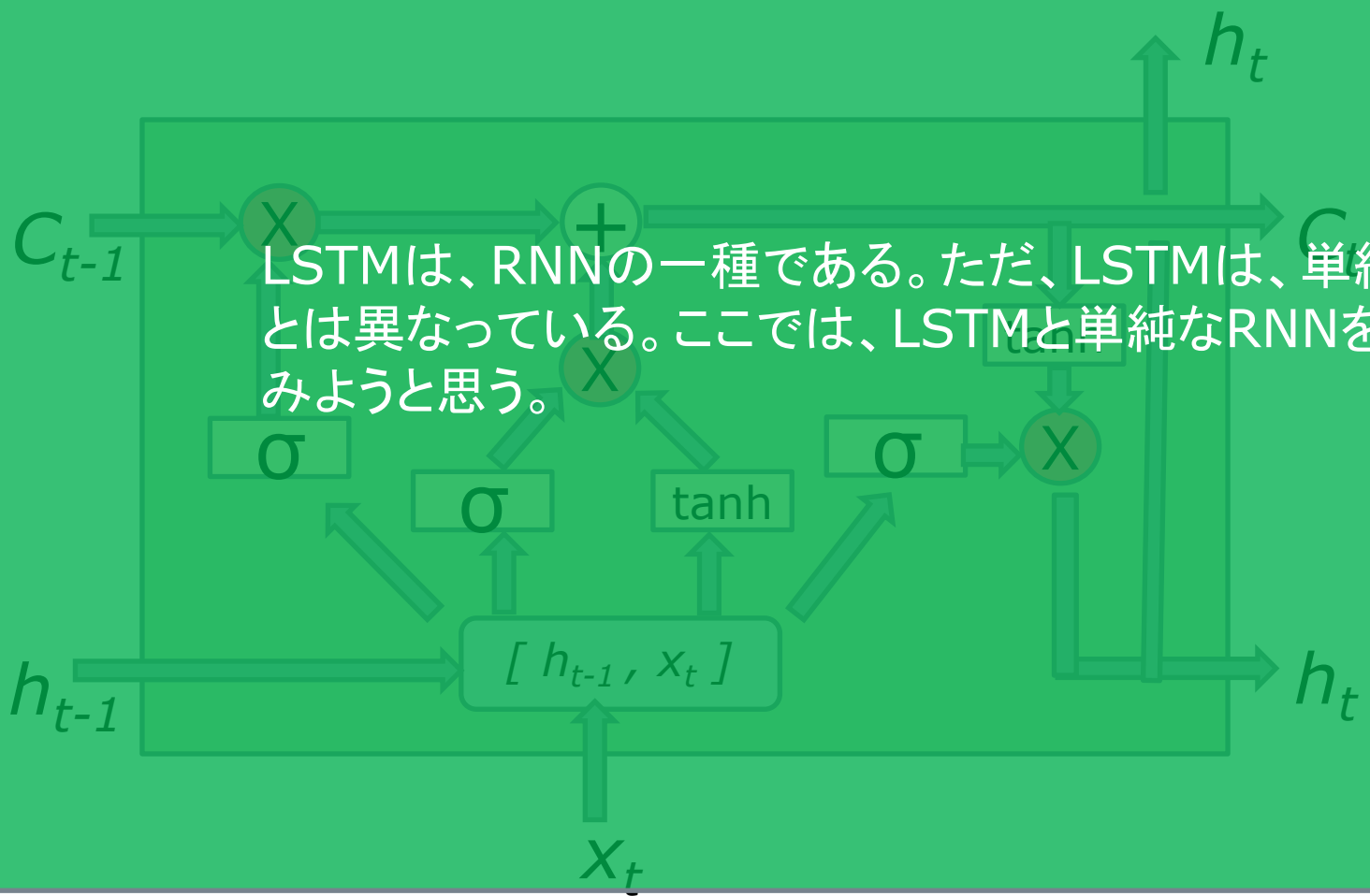
Part III

LSTMの基礎

Part III
LSTMの基礎 Agenda

- LSTMをRNNと比較する
- LSTMは「記憶」を持つ
- LSTMの振る舞いを可視化する
- LSTM -- Gateを持つRNN
- LSTMの構成を、別のスタイルで概観する
- LSTMの働きを詳細に見る
  - **Input Unit と Input Gateの働き**
  - **Memory Unit と Forget Gateの働き**
  - **Status Unit とOutput Gateの働き**
  - **LSTM の働きを式で表す（まとめ）**

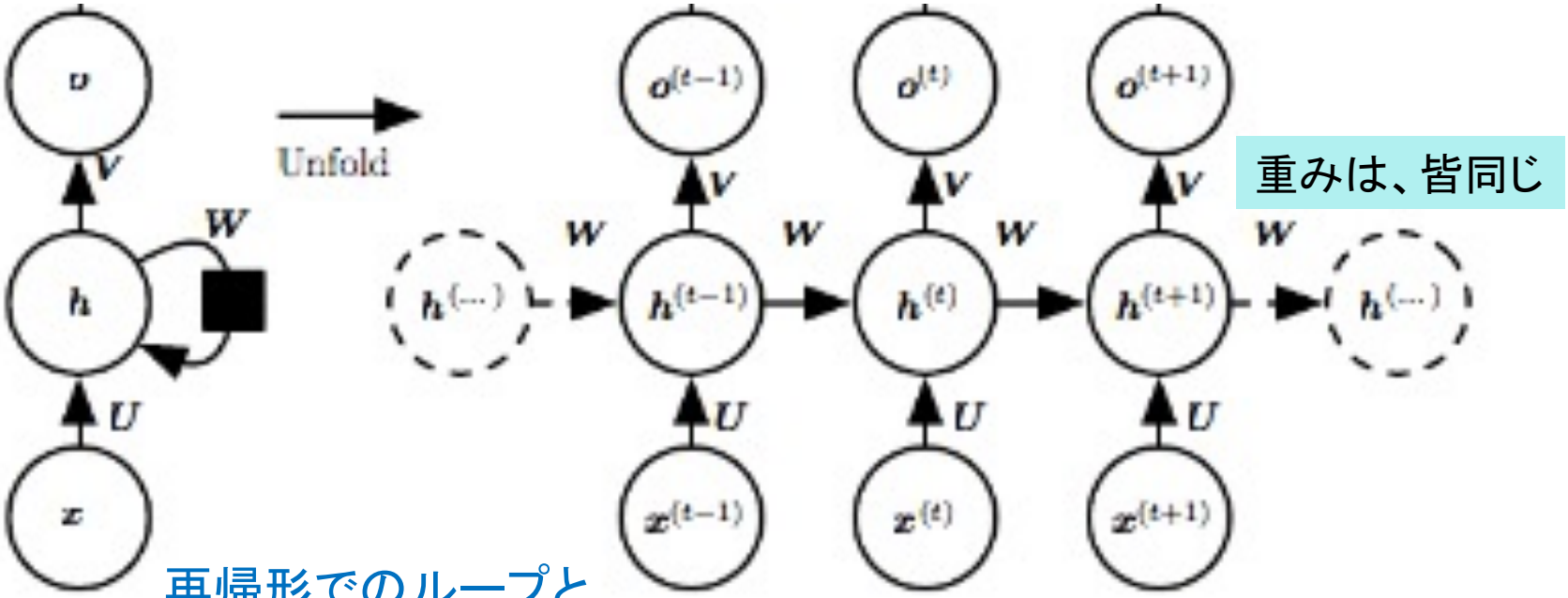
LSTMをRNNと比較する



LSTMは、RNNの一種である。ただ、LSTMは、単純なRNNとは異なっている。ここでは、LSTMと単純なRNNを比較してみようと思う。

RNN

- かつてのRNN(以下、単にRNNと呼ぶことにする)は、次のような形をしていた。



再帰形でのループと、
展開形での横串は、
対応する

RNNでの情報の流れ

- RNNでの情報の流れは、次の三つからなる。

1. 入力から隠れ層へ。 $x_t \rightarrow h_t$
2. 前の隠れ層から現在の隠れ層へ。$h_{t-1} \rightarrow h_t$
3. 隠れ層から出力へ。$h_t \rightarrow o_t$

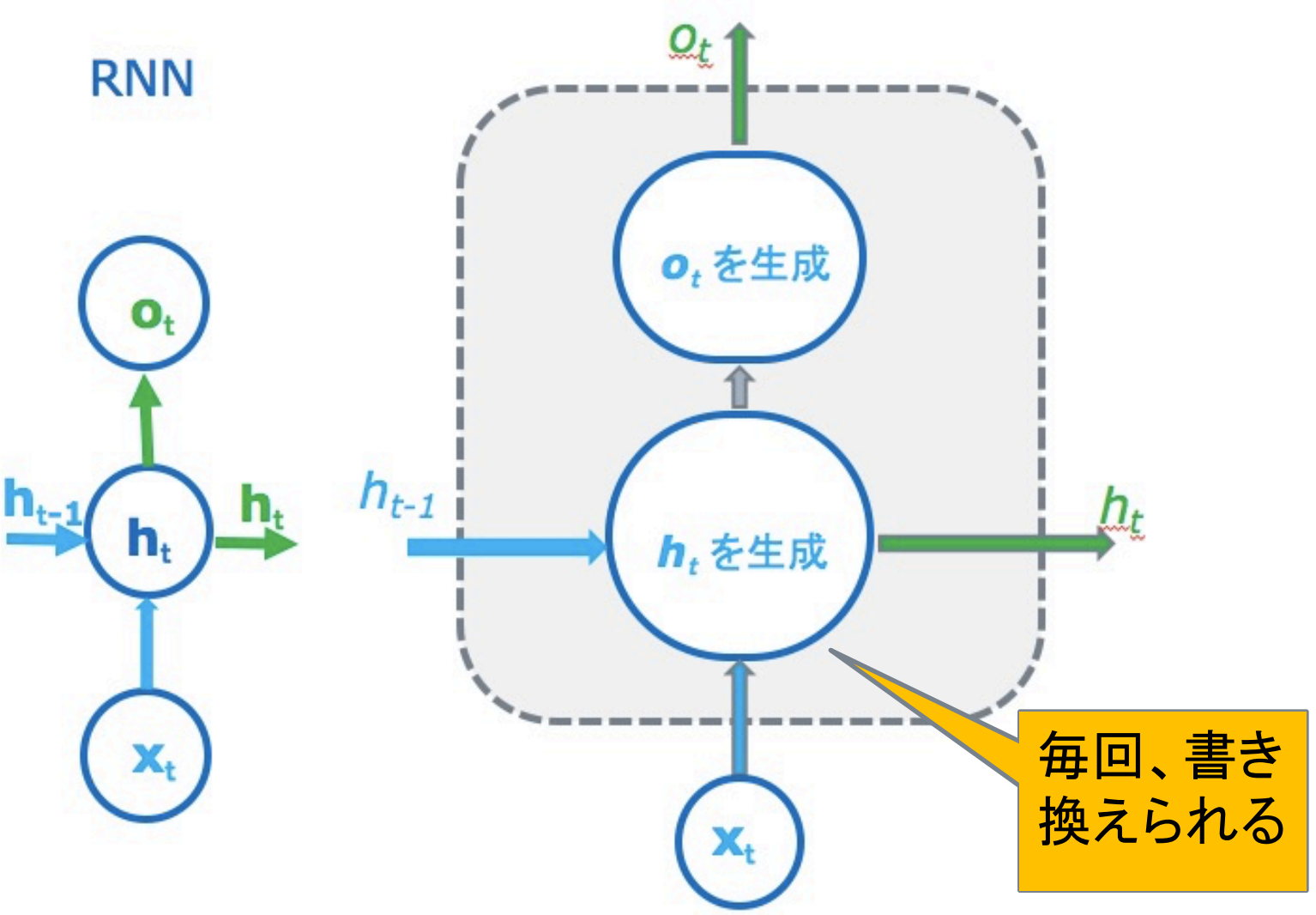
（2. の部分が、RNNで新しく付け加わったもの。）

- 前の層の状態と現在の入力から、新しく現在の状態が生成される。RNNの状態は、前の層の状態と現在の入力で、毎回、書き換えられることになる。
- 単純なRNNは、次の式で表すことができた。

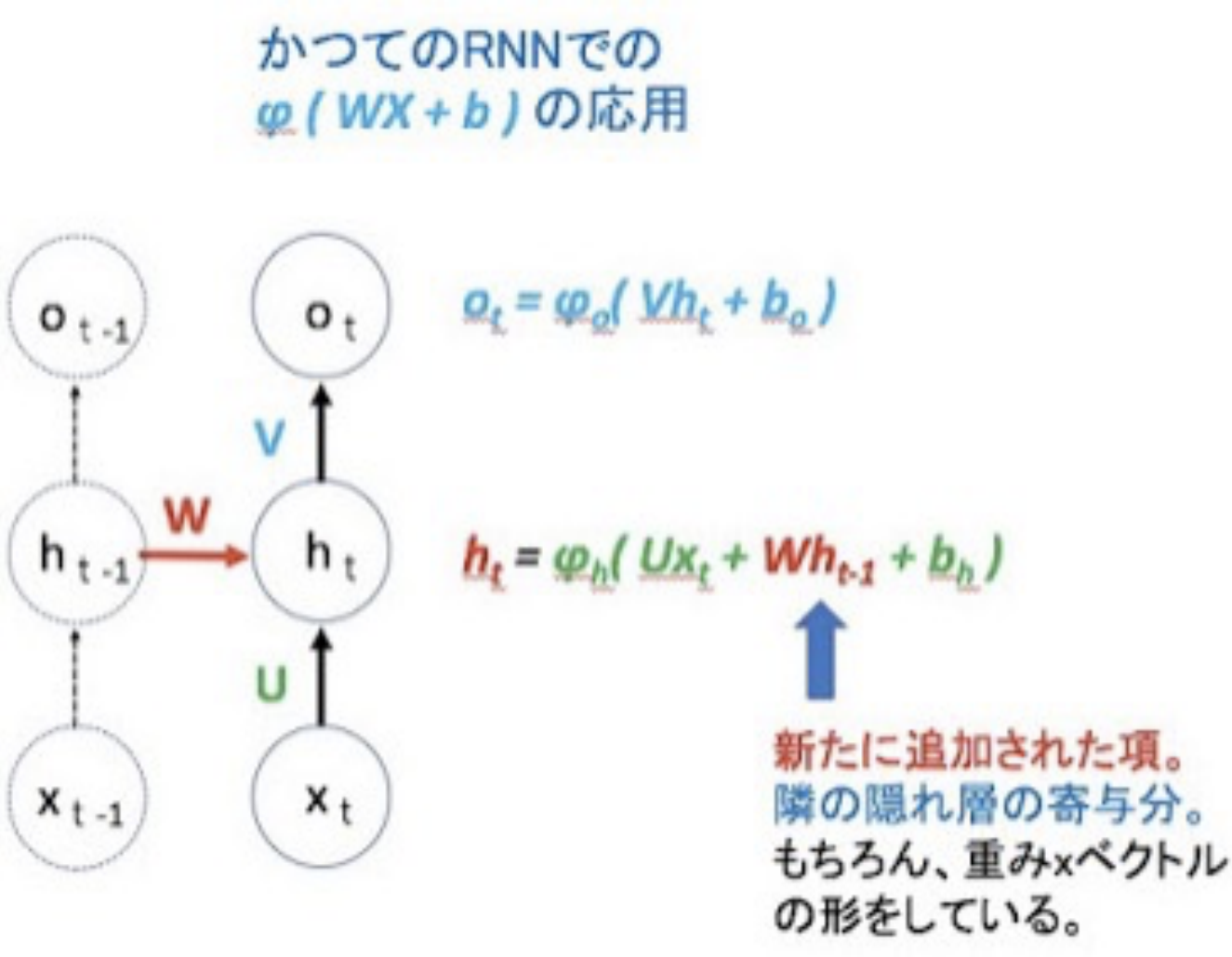
$$h_t = Wh_{t-1} + Ux_t + b$$

$$o_t = tahn\,(h_t)$$

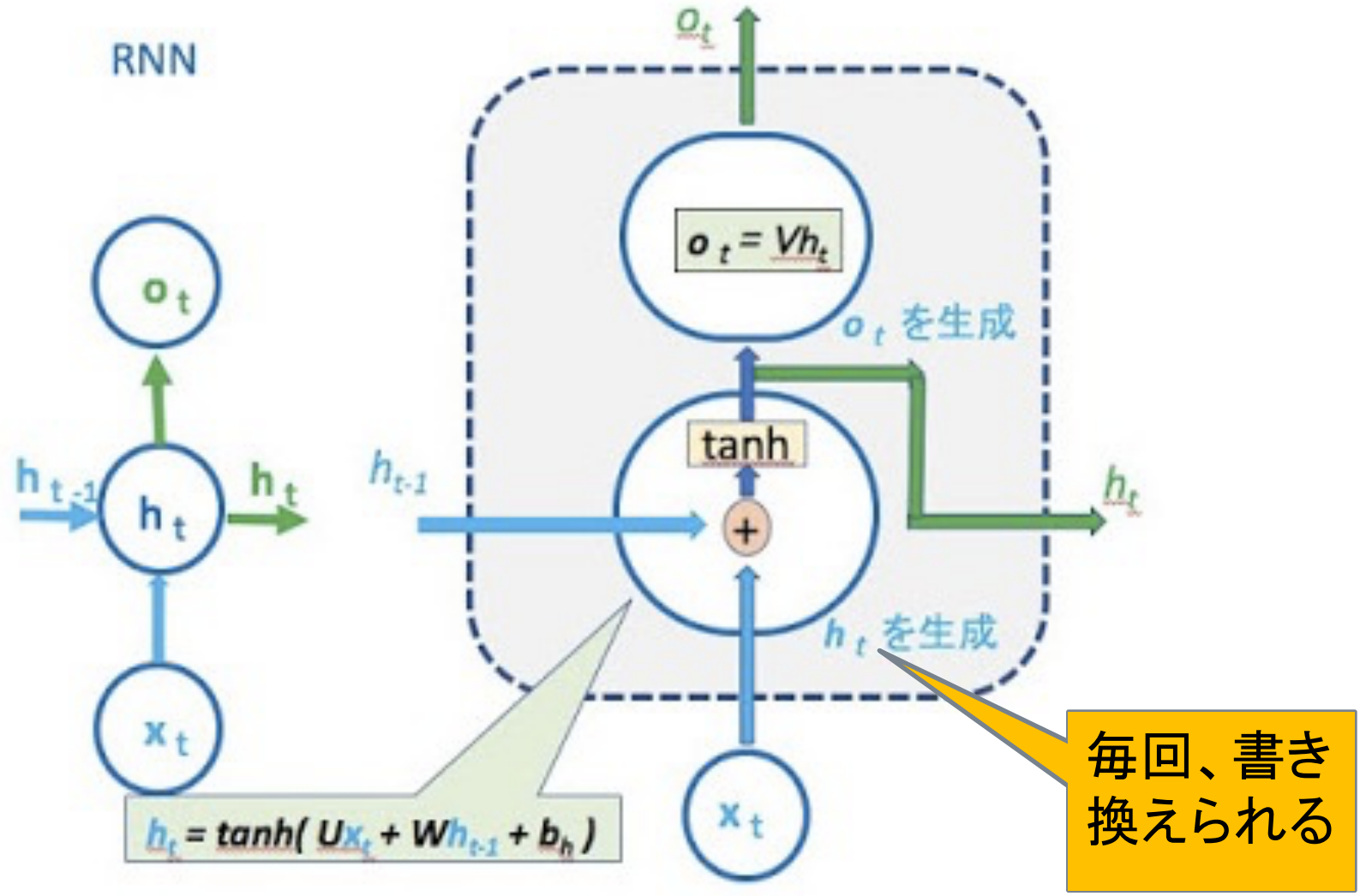
RNNの内部を、簡単に見てみよう。RNNは、h_t を生成する層と、O_t を生成する層の二つからできている。



もう少し詳しく、RNNの内部を見てみよう。
かつての単純なRNNは、次の式で、h層の計算をしていた。



単純なRNNの関係式を、先のRNNのグラフに書き込んでみよう。次の図が、それである。出力層では、活性化関数とバイアスが省略されている。



LSTM

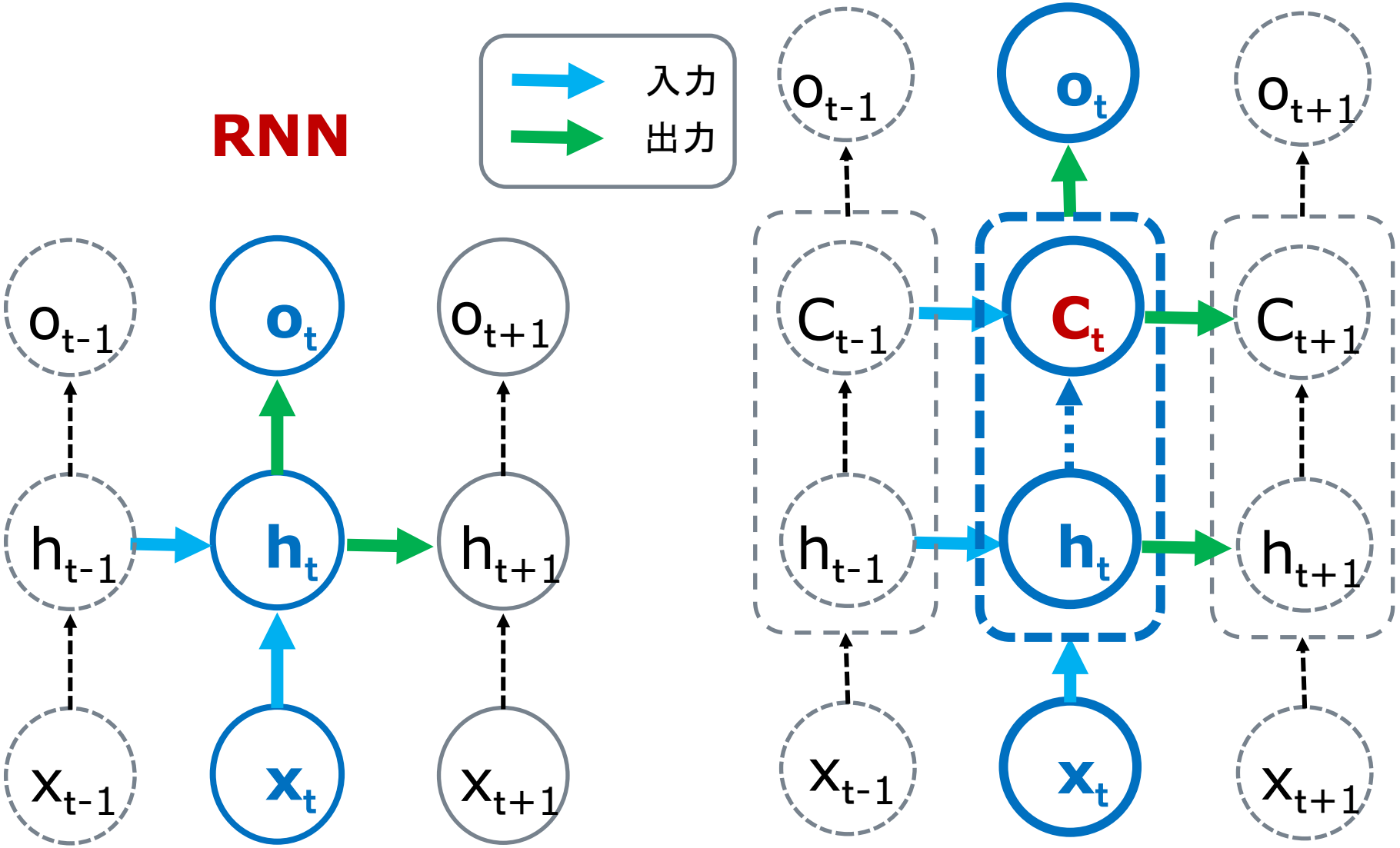
- RNNでは、横の繋がり（展開形で）は、h層の一本だけだったのだが、LSTMでは、もう一本多く横のつながりを持つ。このc層は、LSTMで新たに付け加えられたものだ。LSTMでは、この層が独自の大事な働きをすることになる。（Memory Cell）
- この図では、入力は水色、出力は緑色で表されているのだが、RNNは、二つの入力 h_{t-1} 、x_t を持ち、二つの出力 h_t 、o_t を持つ。それに対して、LSTMは、三つの入力 h_{t-1} 、c_{t-1}、x_t を持ち、三つの出力 h_t 、c_t、 o_t を持つ。

LSTMに導入された、もう一つの「状態」

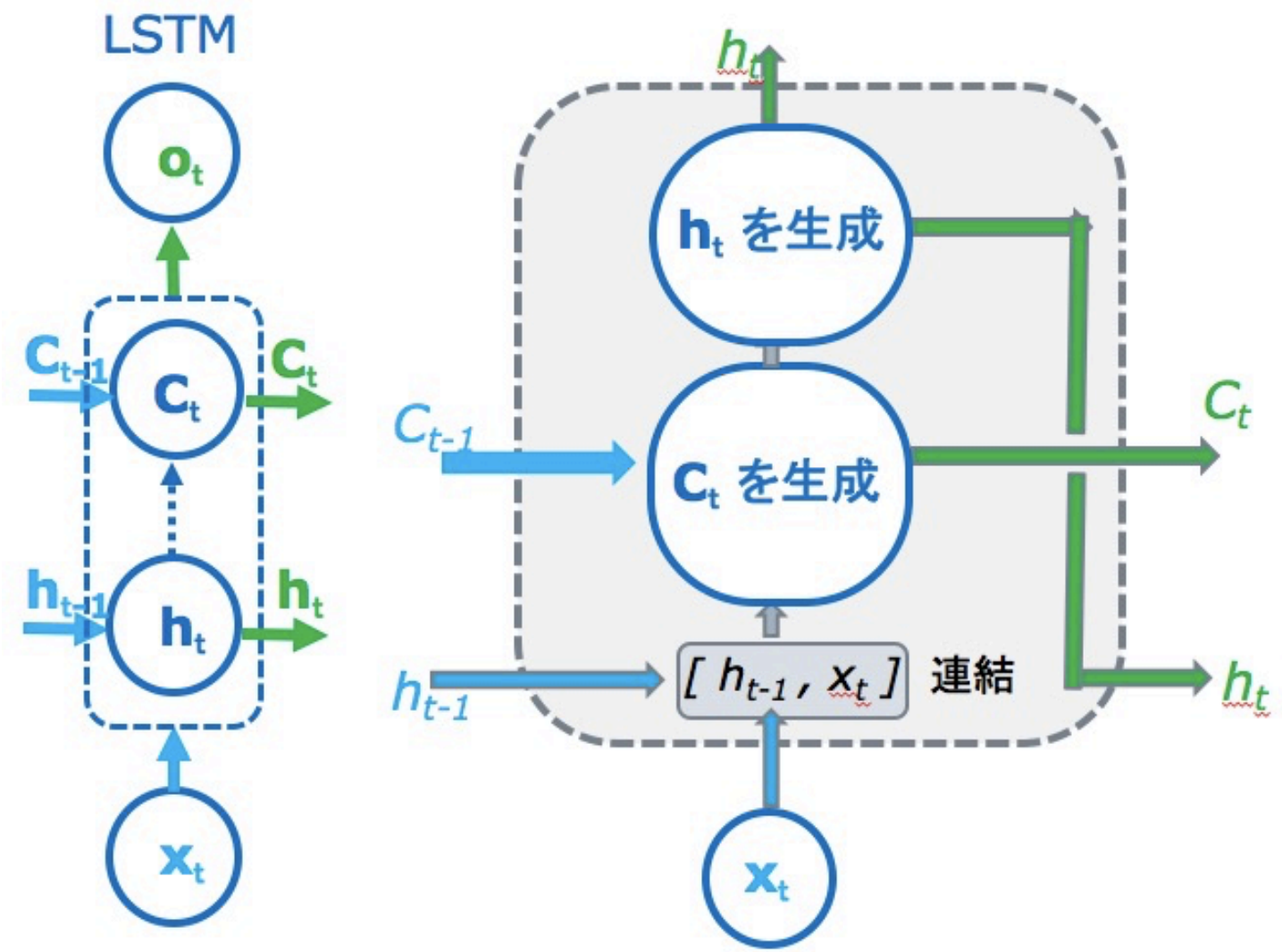
- LSTMでは、RSSの状態 h_t に加えて、もう一つの状態 C_t が追加されている。ここが、単純なRSSと大きく違うところだ。
- この状態 C_t は、一つ前の状態 C_{t-1} と、一つ前のLSTMの状態 h_{t-1} と、現在の入力 o_t から作られる。
- 一つ前の状態 h_{t-1} と、現在の入力 o_t に依存するという点では、RSSの状態 h_t と同じである。
- LSTMの状態 h_t は、この状態 C_t から作られる。

この状態 C_t を、**「記憶（Memory）」**と呼ぶ。

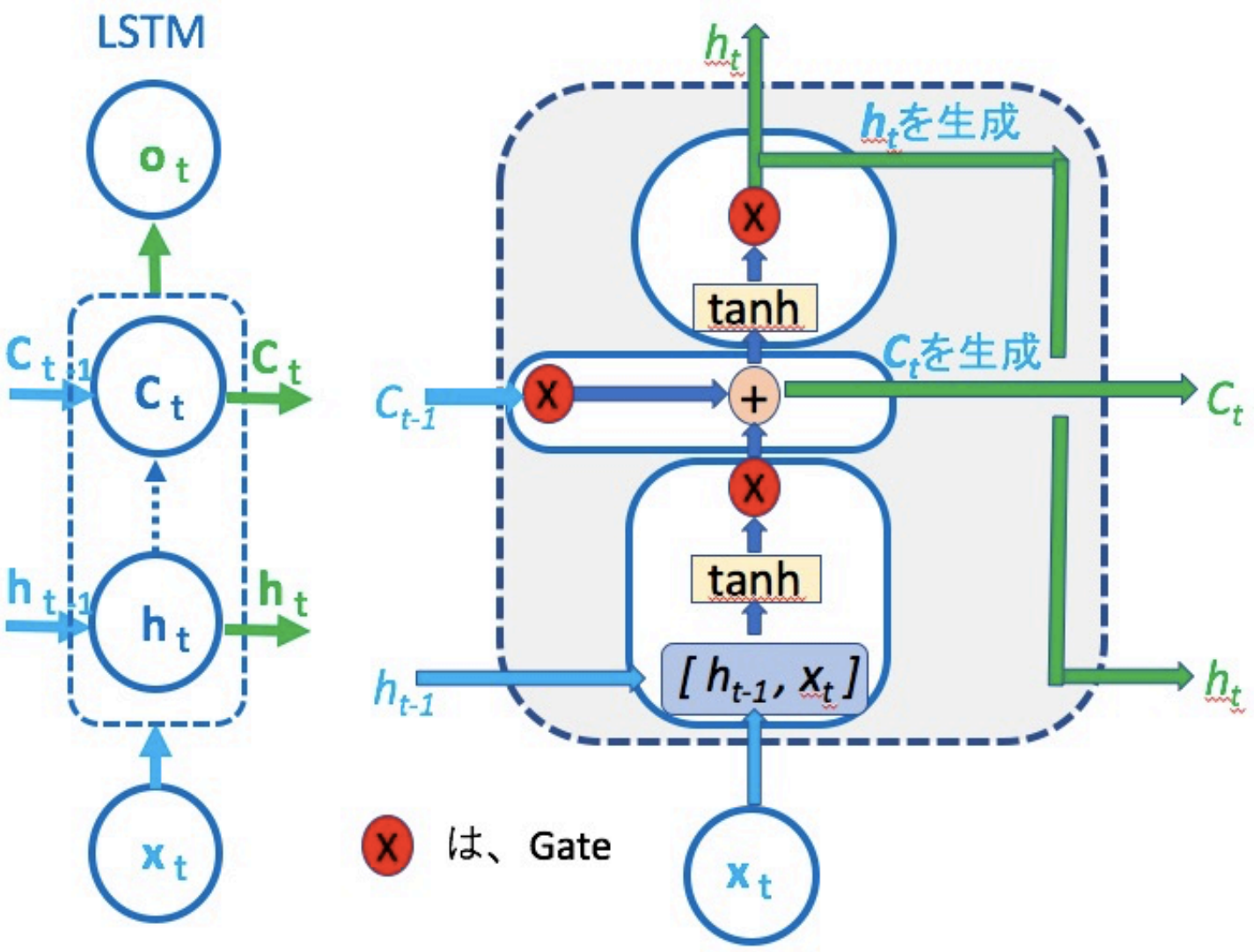
RNNとLSTMの比較



LSTMの内部を、簡単に見てみよう。（詳細な説明は、後にする。）
基本的には、左の図に対応して、LSTMは、 状態 h_t を生成する層と、c_t を生成する層の二つからできている。



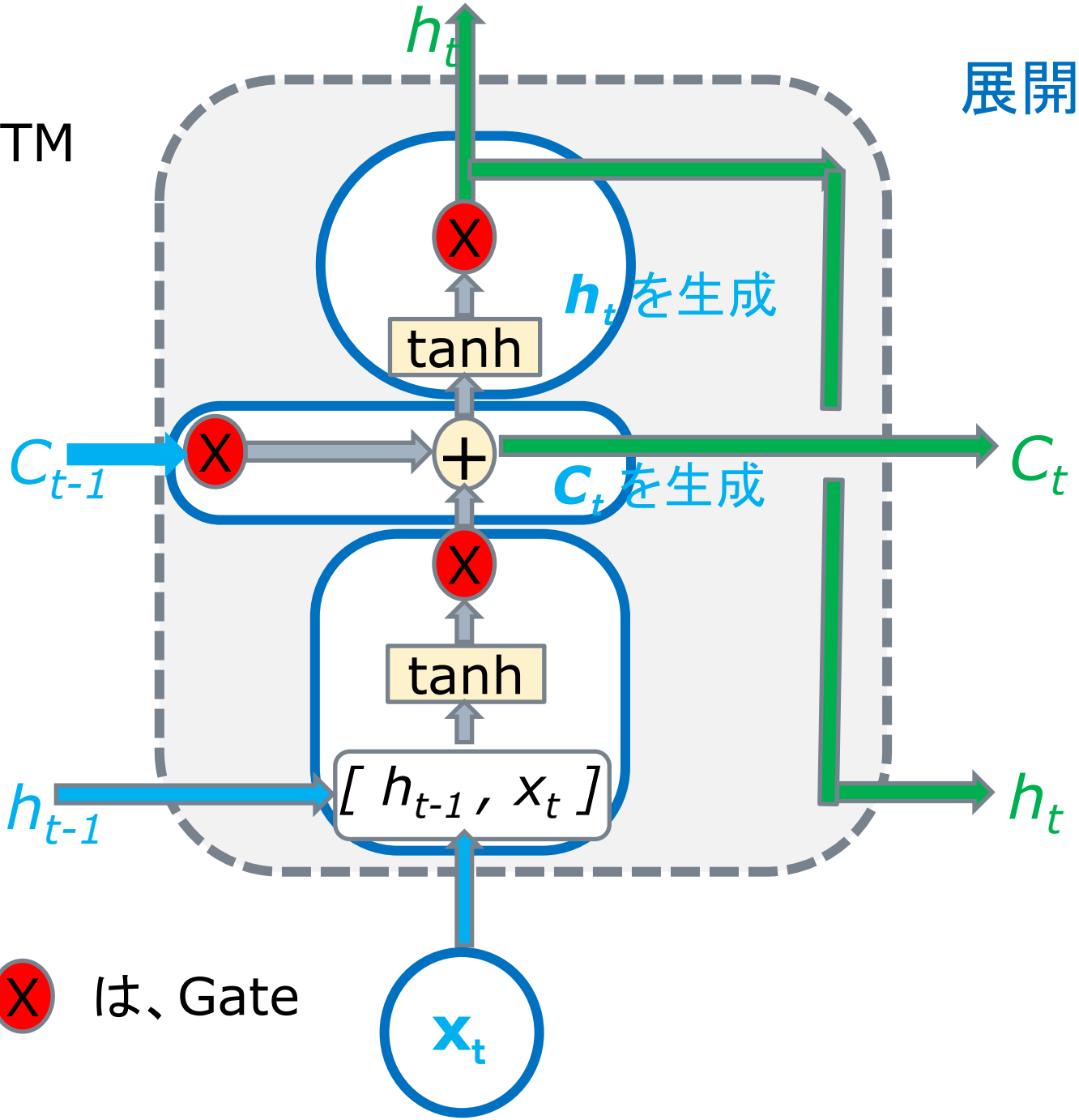
もう少し詳しくみてみよう。



LSTMでは、次のことに注意しよう。

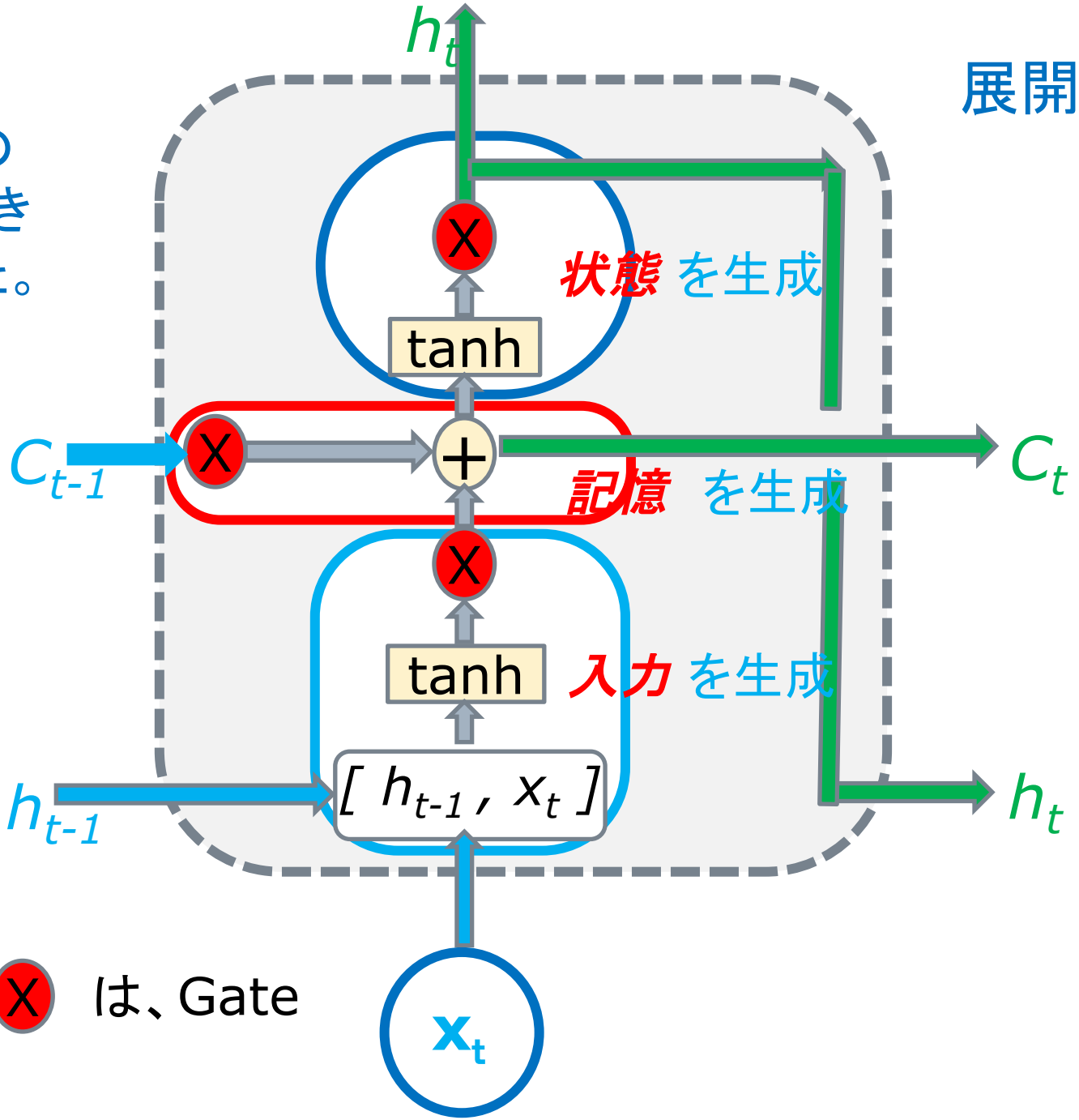
- 第一に、LSTMの出力 o_t は、h_tと同じものである。先ほど、三つの出力があると書いたが、三つのうち二つは、同じ h_t である（右の図で、上方向と横方向の出力の h_t ）。右と左の図を比べて、c層の位置とh層の位置が、上下反対のように見えるのは、h層が出力を行う層だからである。また、右図の大きな回り道は、h_t が、二箇所で利用されているからである。
- 第二に、それでは、情報の流れは、右の図の大きなノード間の、c層からh層への流れであって、左の図にあるような h層からc層の流れは、存在しないのであろうか？　そうではない。h層への入力である h_{t-1} は、入力 x_t と合流して、それがc層に流れている。ここでは、二つの入力を合流させるのに、二つのベクトルを横に繋げて連結するという方法を取っている。（右図の下の方）

これが、LSTM



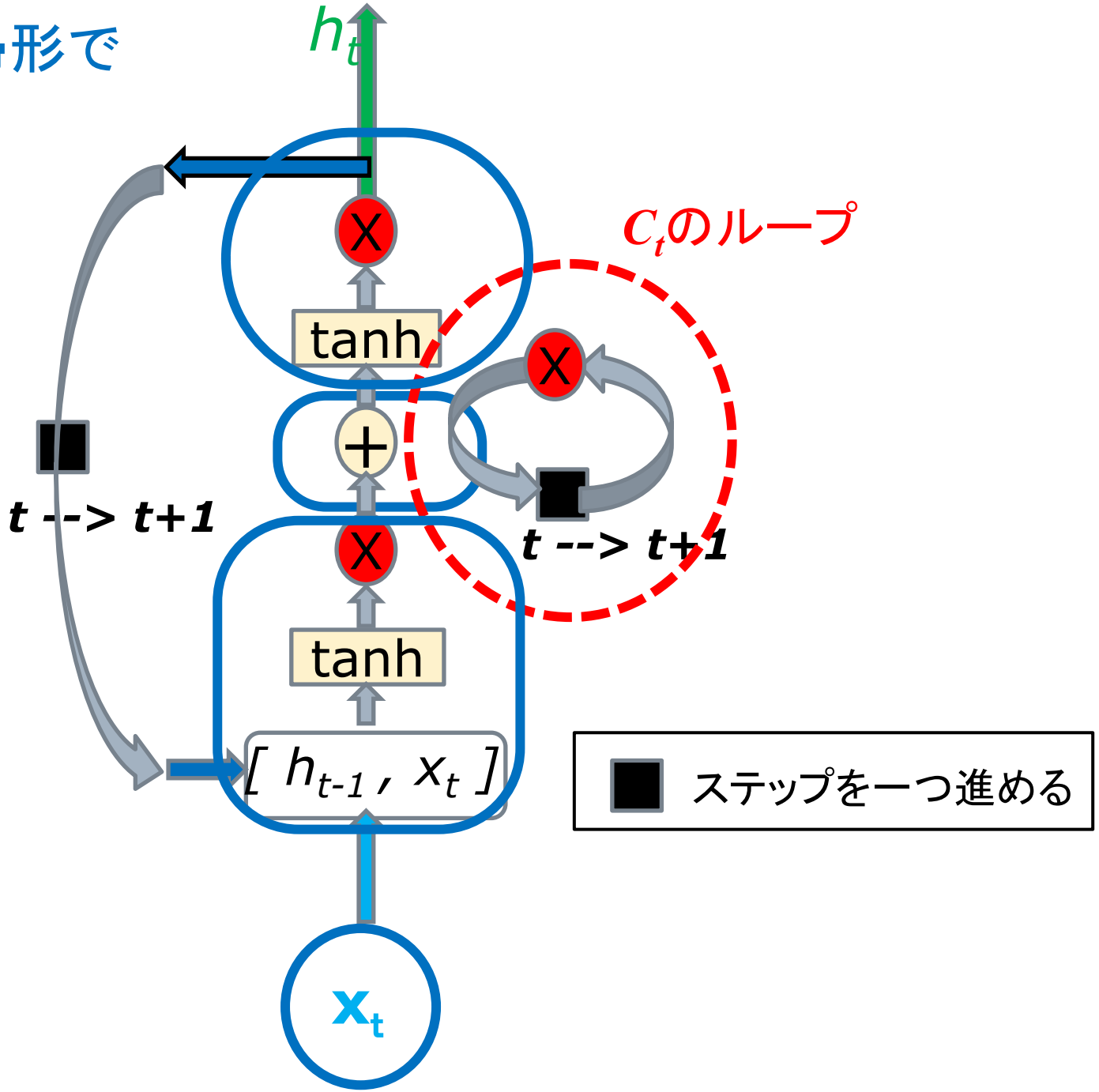
X は、Gate

それぞれの
働きを、書き
込んで見た。

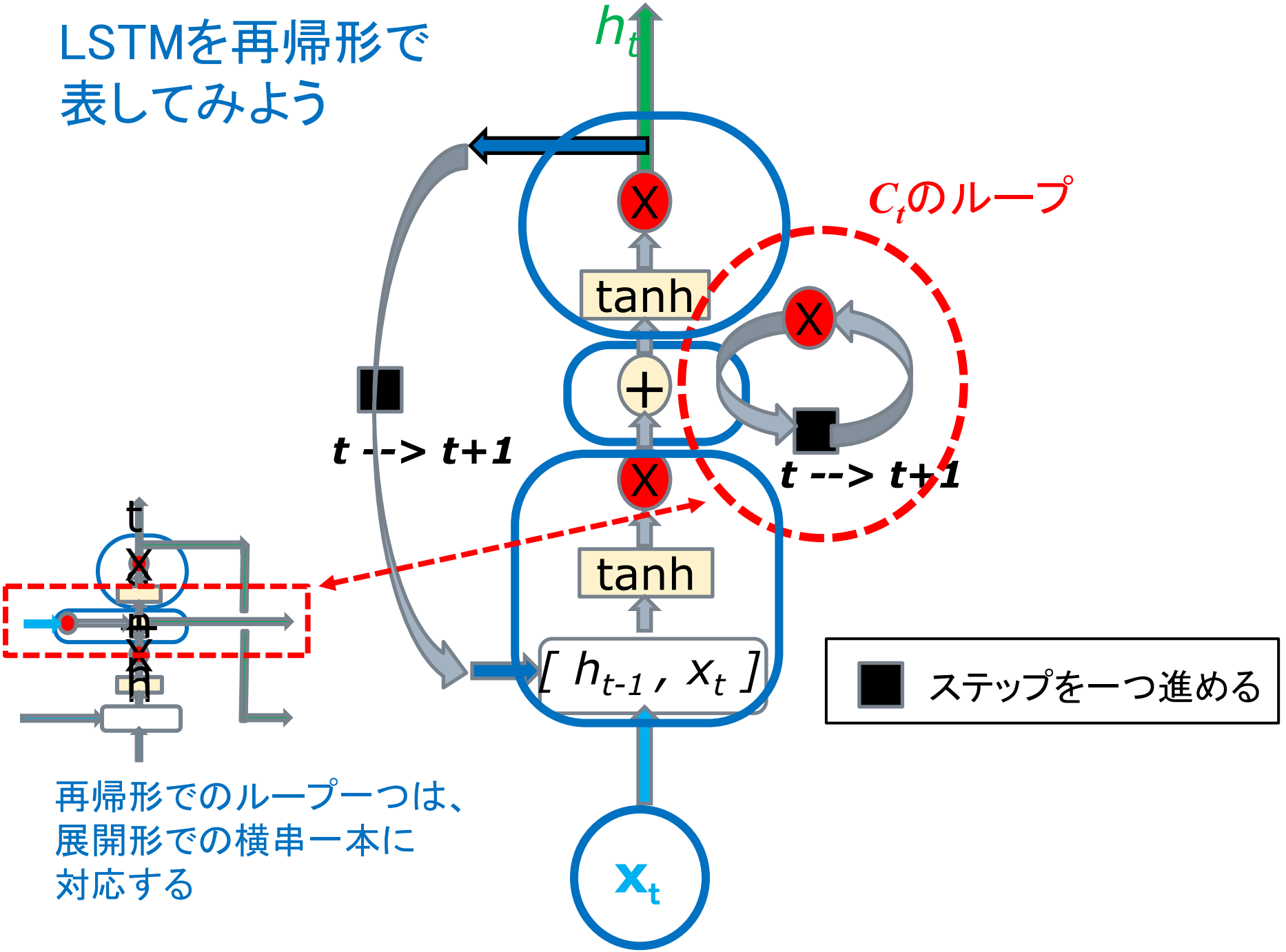


X は、Gate

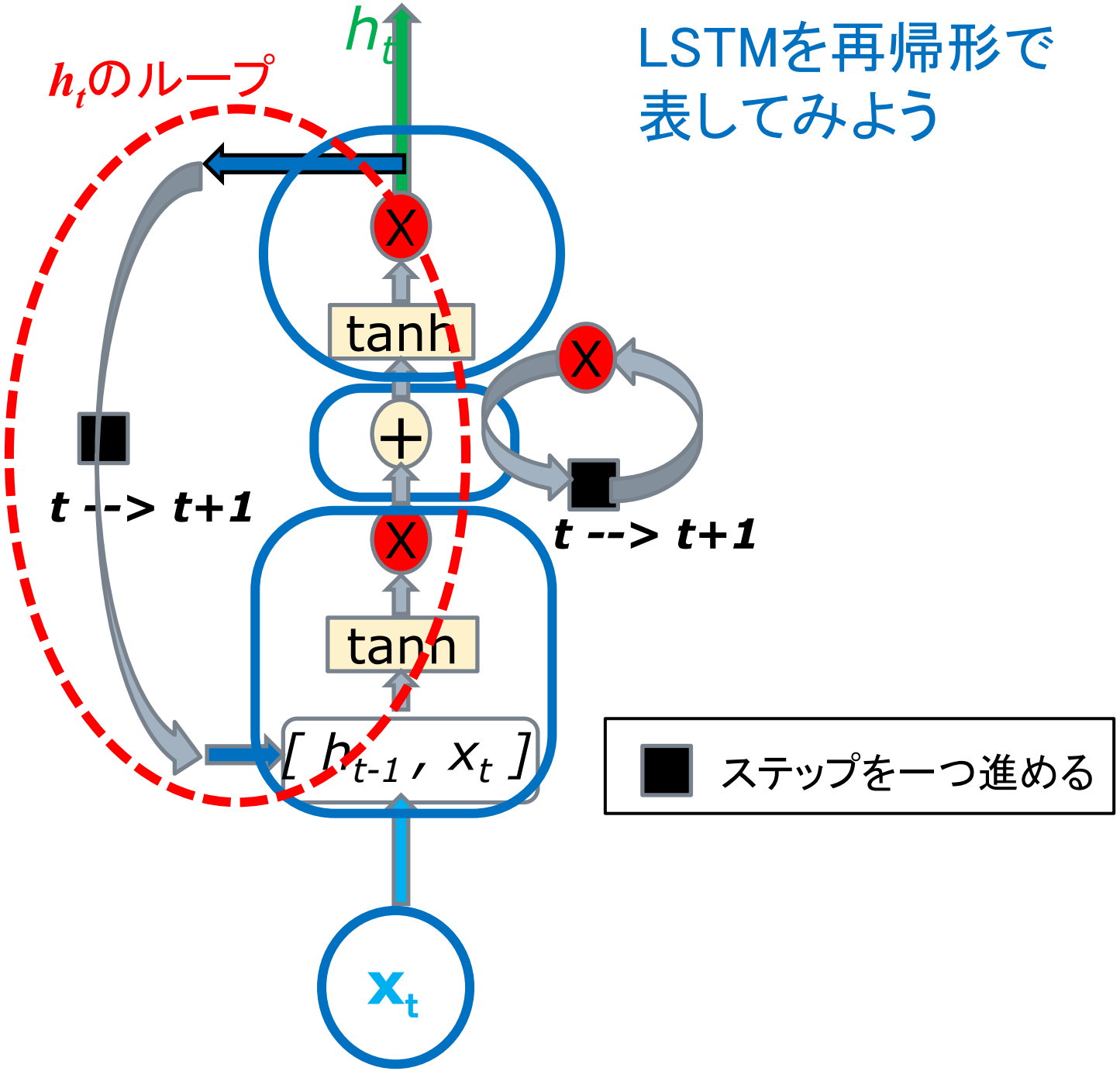
LSTMを再帰形で
表してみよう

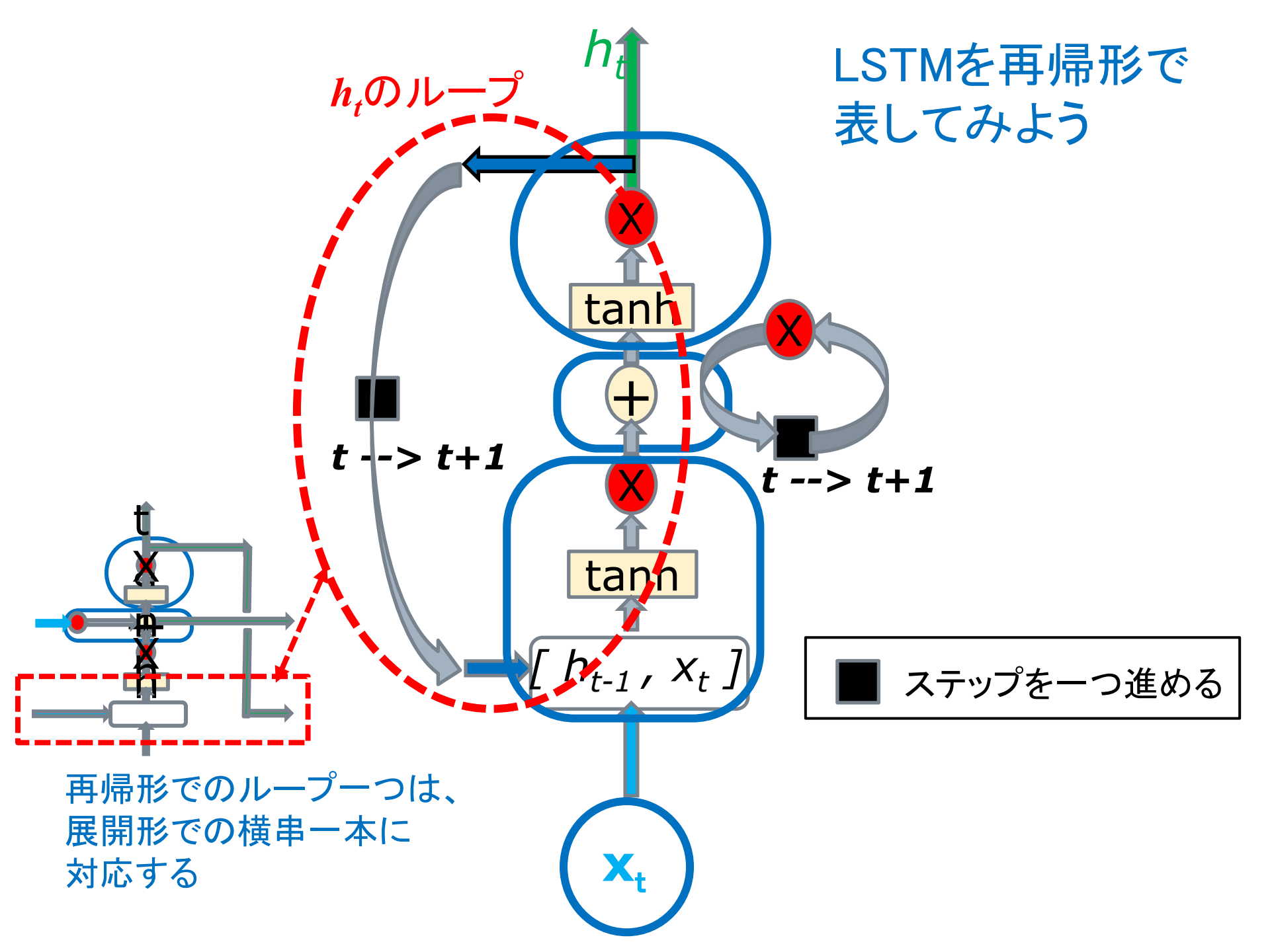


LSTMを再帰形で
表してみよう



再帰形でのループ一つは、
展開形での横串一本に
対応する


LSTMを再帰形で
表してみよう
h_t
h_tのループ
X
tanh
X
+
X
t --> t+1
t --> t+1
tanh
[h_{t-1} , x_t]
x_t
ステップを一つ進める


LSTMを再帰形で
表してみよう
h_t
h_tのループ
tanh
tanh
t --> t+1
t --> t+1
[h_{t-1} , x_t]
x_t
ステップを一つ進める
再帰形でのループ一つは、
展開形での横串一本に
対応する

LSTMは「記憶」を持つ

ここでは、LSTMが、「記憶」を持つことを述べようと思う。

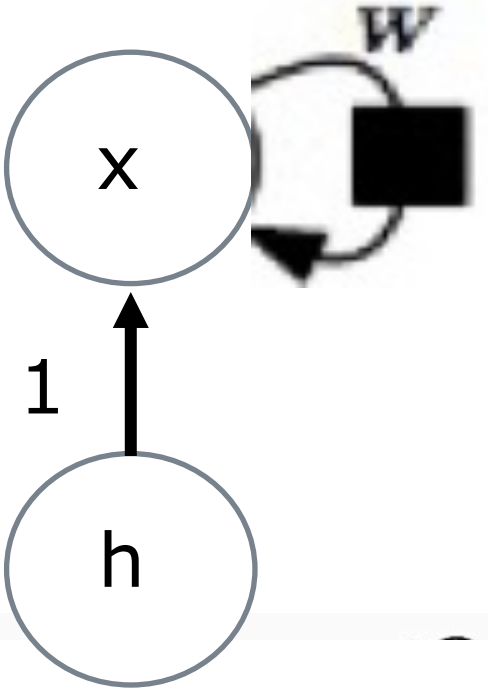
LSTMでは、RNNに新たに追加された $\boldsymbol{C_t}$ ユニットが、「記憶」を担う。これを ***Memory Cell*** と呼ぶことがある。
（小論では、Memory Unit と呼んでいる）

LSTMが「記憶」を持つメカニズム
Constant Error CarouselとForget Gate

- かつて放棄されたRNNは、長い「記憶」を持つことができなかった。その主な理由は、RNNの状態が、毎ステップごとに、ループの重みWによって書き換えられるからである。
- LSTMでは、Memory Cellのループの重みを1に固定する。このループは、Back Propagationによって学習される重みを持たないと考えればいい。これによって、何回ループを回っても、Memory Cell は書き換えられることがなくなる。「記憶」は、持続する。これを**Constant Error Carousel**という。
- LSTMでの、もう一つの工夫は、ループの途中に、前の記憶を「忘れる」装置を取り付けたことだ。これを **Forget Gate** と呼ぶ。これは、ある時には、ループの重みがゼロになることに相当する。
- これによって、LSTMは、前の「記憶」を忘れるだけでなく、新しい「記憶」に、「記憶」を更新できるようになった。

$x_t = \tanh(1 \cdot h_t + W \cdot x_{t-1})$

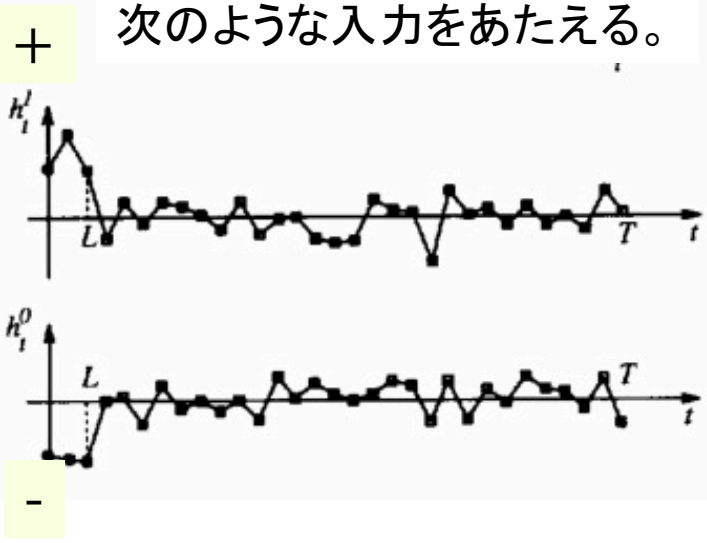
RNNでの記憶の困難

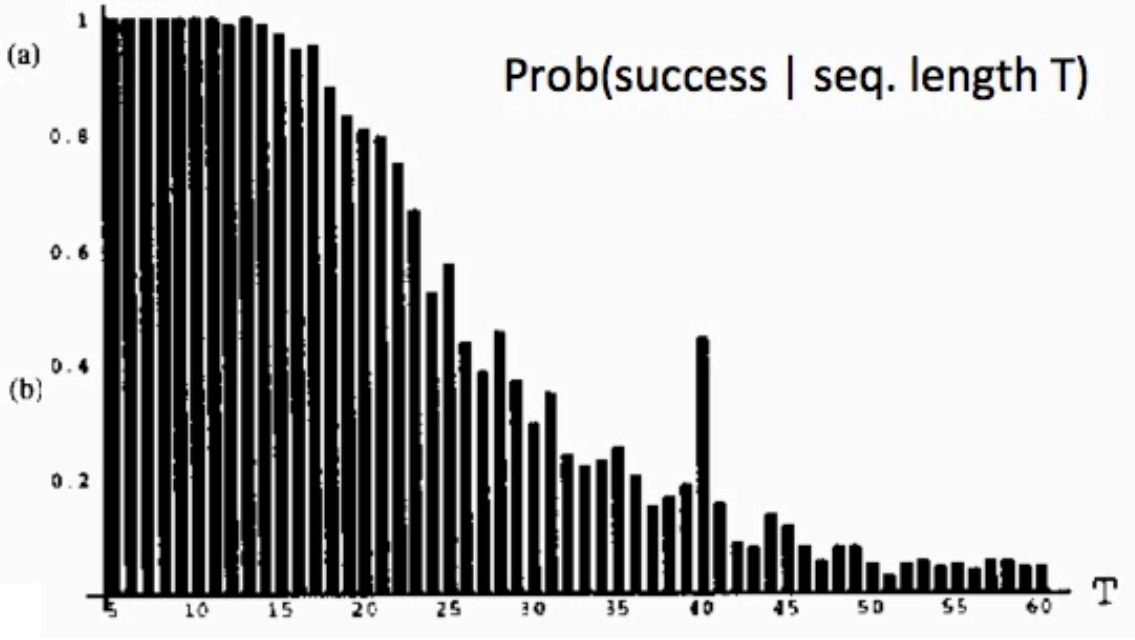


初期の入力の符号（+-）で与えられた情報を、このユニットは、記憶できるか？

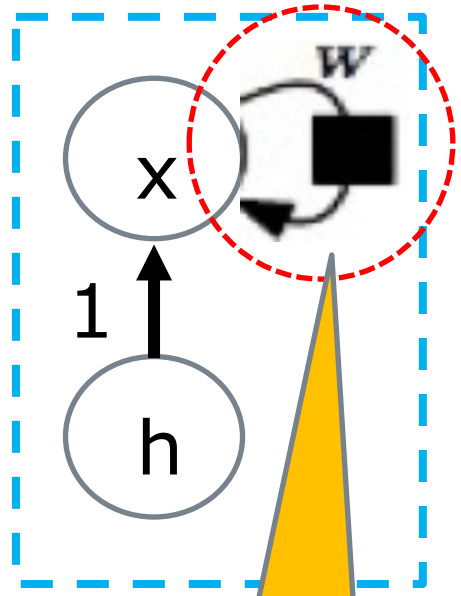
Tが大きくなるにつれて、それは、極端に難しくなる。

次のような入力をあたえる。

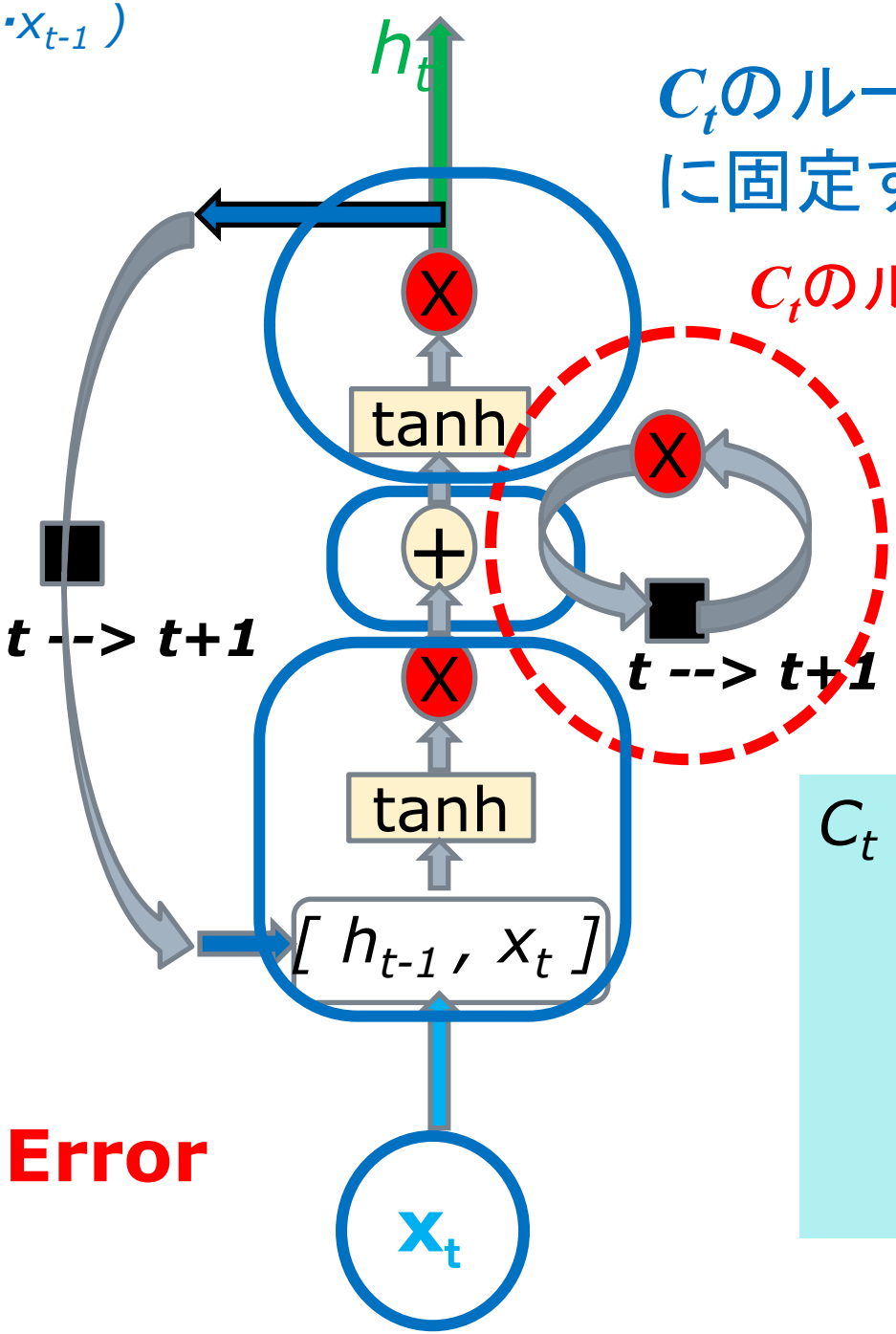




$x_t = \tanh(1 \cdot h_t + W \cdot x_{t-1})$



xは、Wによって毎回、書き換えられる

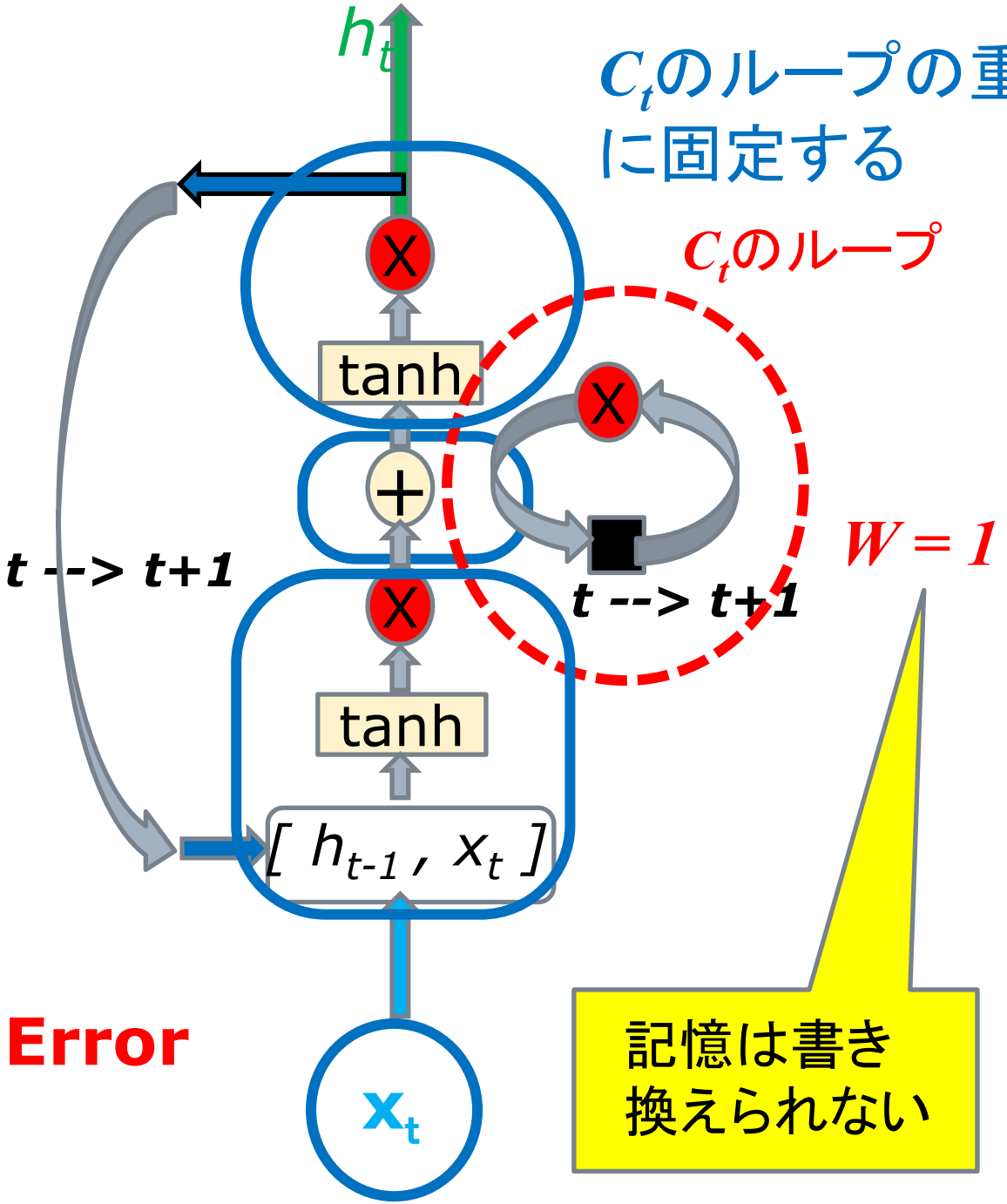


C_tのループの重みを1に固定する

C_tのループ

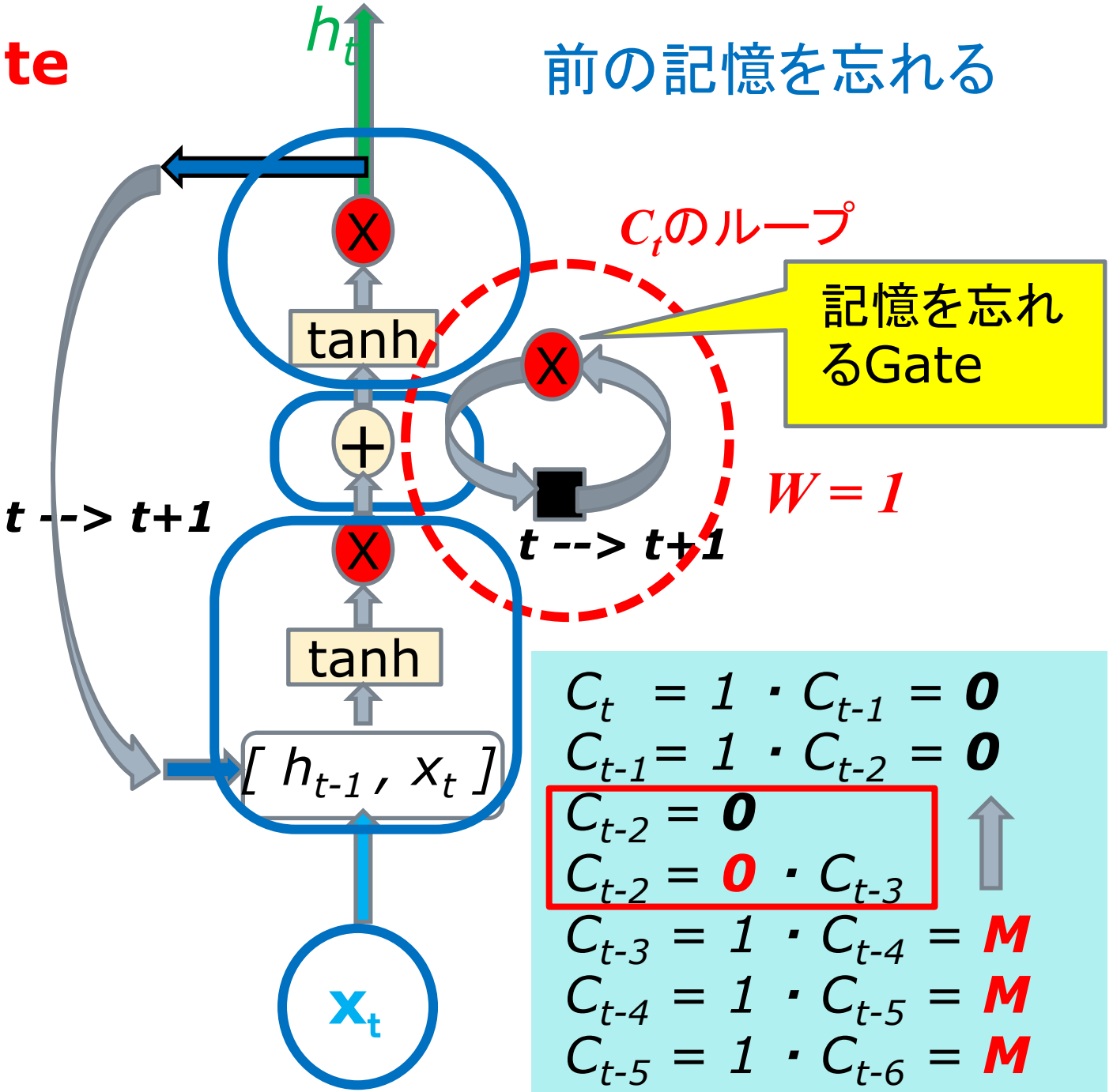
$$
\begin{aligned}
C_t &= 1 \cdot C_{t-1} \\
&= 1 \cdot C_{t-2} \\
&= 1 \cdot C_{t-3} \\
&\cdots\cdots \\
&= 1 \cdot C_2 \\
&= C_1
\end{aligned}
$$

Constant Error Carousel


h_t
C_tのループの重みを1に固定する
C_tのループ
X
tanh
X
+
X
t --> t+1
t --> t+1
W = 1
X
tanh
[h_{t-1} , x_t]
x_t
記憶は書き換えられない
Constant Error Carousel

Forget Gate

前の記憶を忘れる



$C_t = 1 \cdot C_{t-1} = \mathbf{0}$

$C_{t-1} = 1 \cdot C_{t-2} = \mathbf{0}$

$C_{t-2} = \mathbf{0}$

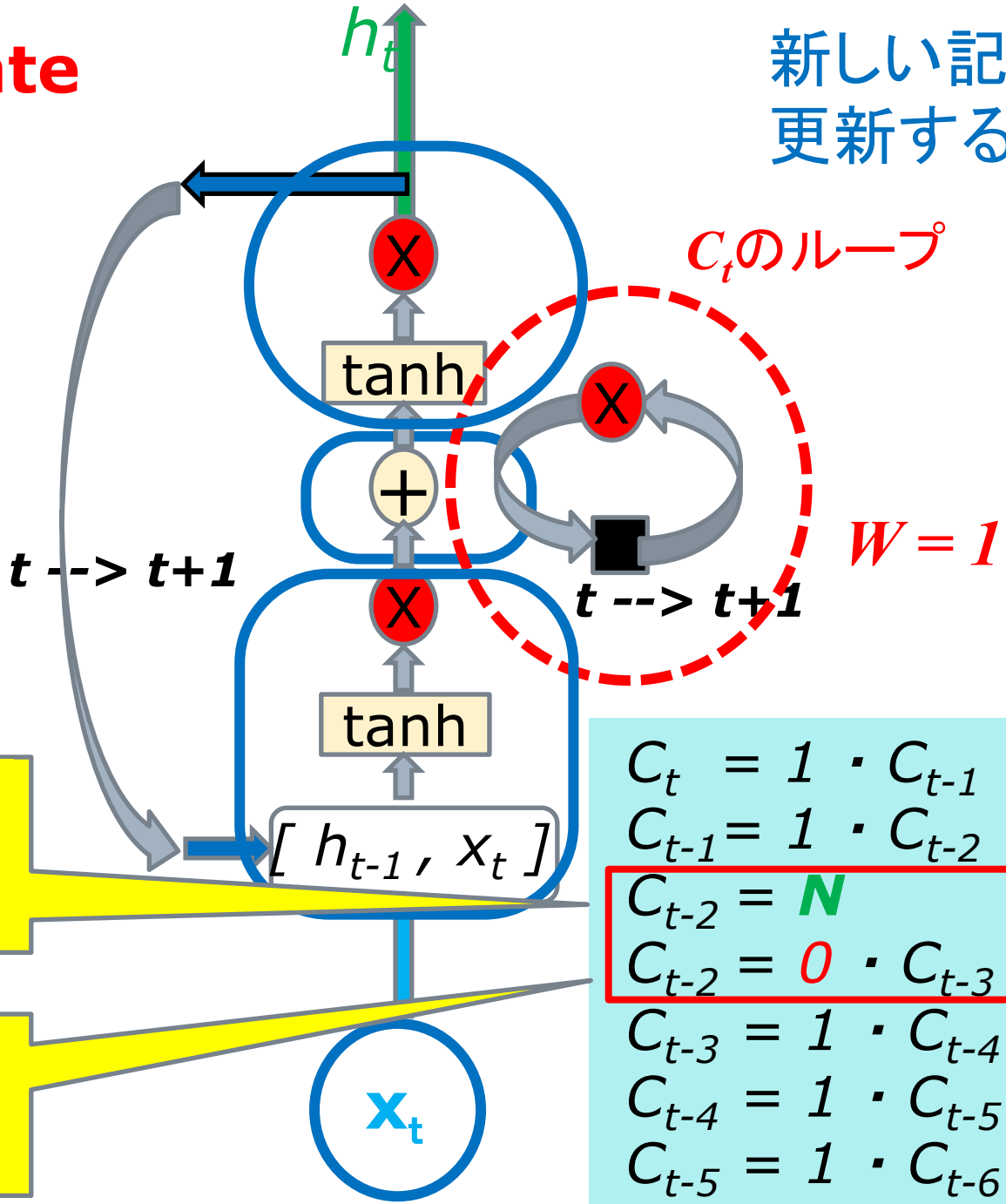
$C_{t-2} = \mathbf{0} \cdot C_{t-3}$

$C_{t-3} = 1 \cdot C_{t-4} = \mathbf{M}$

$C_{t-4} = 1 \cdot C_{t-5} = \mathbf{M}$

$C_{t-5} = 1 \cdot C_{t-6} = \mathbf{M}$

Forget Gate



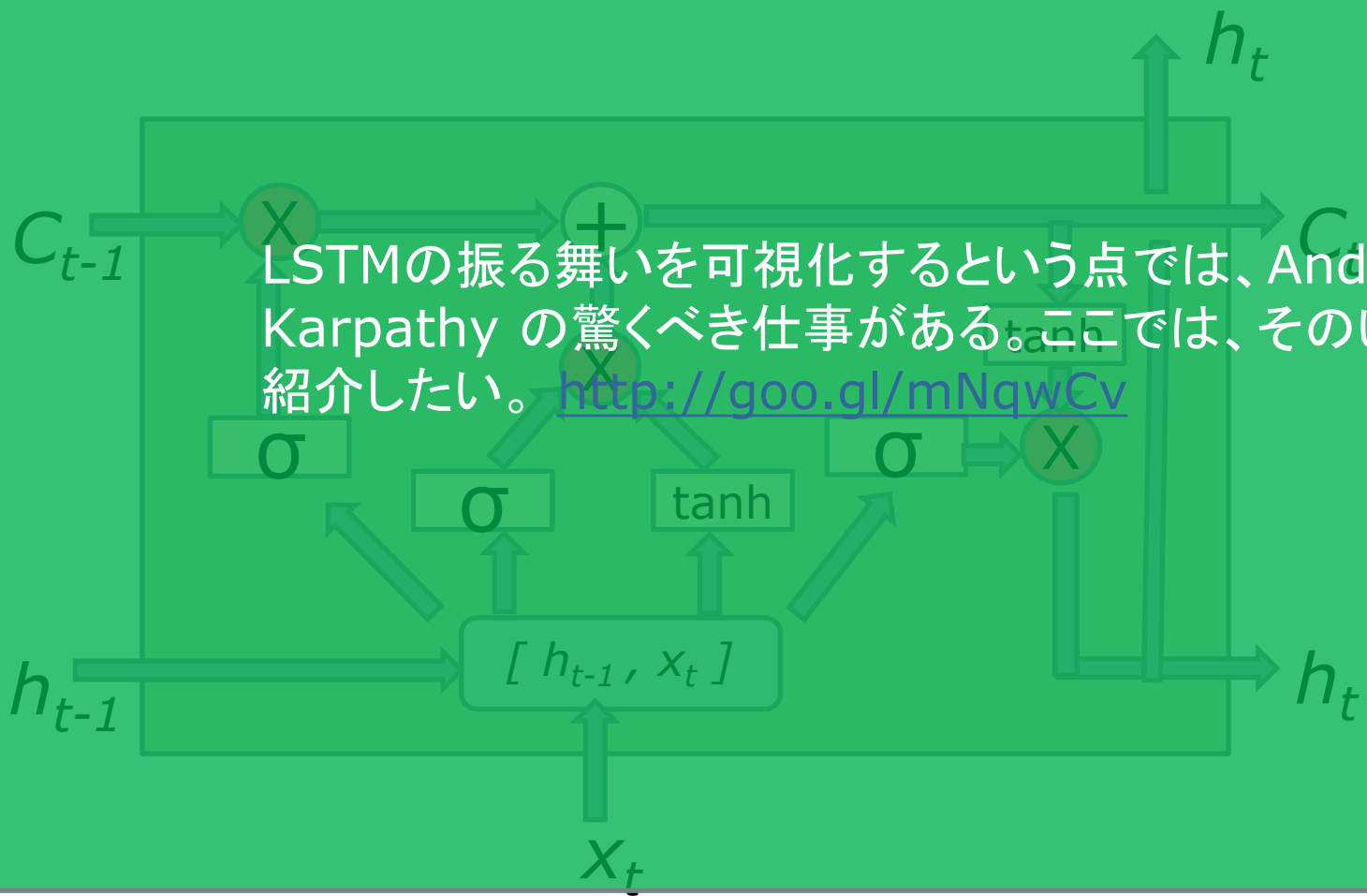
新しい記憶に更新する

新しい記憶**N**に更新する

前の記憶**M**を消す

$C_t = 1 \cdot C_{t-1} = N$
$C_{t-1} = 1 \cdot C_{t-2} = N$
$C_{t-2} = N$
$C_{t-2} = 0 \cdot C_{t-3}$
$C_{t-3} = 1 \cdot C_{t-4} = M$
$C_{t-4} = 1 \cdot C_{t-5} = M$
$C_{t-5} = 1 \cdot C_{t-6} = M$

LSTMの振る舞いを可視化する

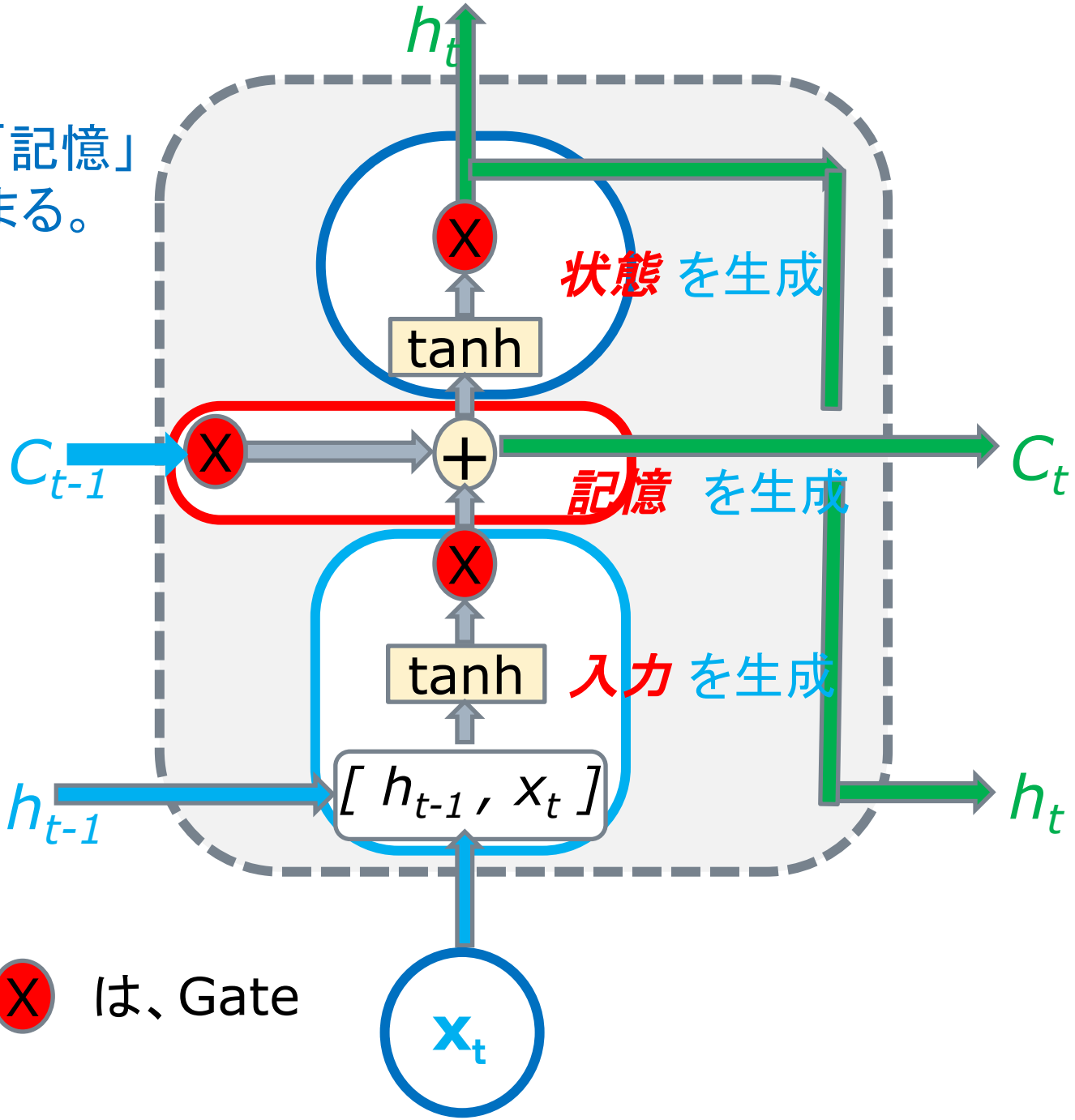


LSTMの振る舞いを可視化するという点では、Andrej Karpathy の驚くべき仕事がある。ここでは、そのいくつかを紹介したい。http://goo.gl/mNqwCv

LSTMの振る舞いを可視化する

- LSTMの振る舞いを決めているのは、LSTMの「状態」である。RSSでは、RSSの「状態」は、直前のRSSの「状態」と、現在の「入力」で決定されていた。（それゆえ、毎回、更新される）
- LSTMの「状態」を決めているのは、直前のLSTMの「状態」でも、現在の「入力」でもない。それを直接に決めているのは、LSTMの「記憶」である。
- 先に見たように、LSTMの「記憶」は、一定期間持続し、ある時には、別の「記憶」に書き換えられ、また持続する。
- LSTMの「記憶」の変化に対応して、LSTMの「状態」が変化するので、LSTMの「状態」は、一定期間持続し、ある時には、別の「状態」に書き換えられ、また持続する。
- こうして、LSTMの「状態」の変化に応じて、LSTMの振る舞いは、変化することになる。

「状態」は、「記憶」
によって決まる。



X は、Gate

非常に興奮

興奮していない

上位５つの候補
次の文字の予測

非常に興奮

上位５つの候補
次の文字の予測

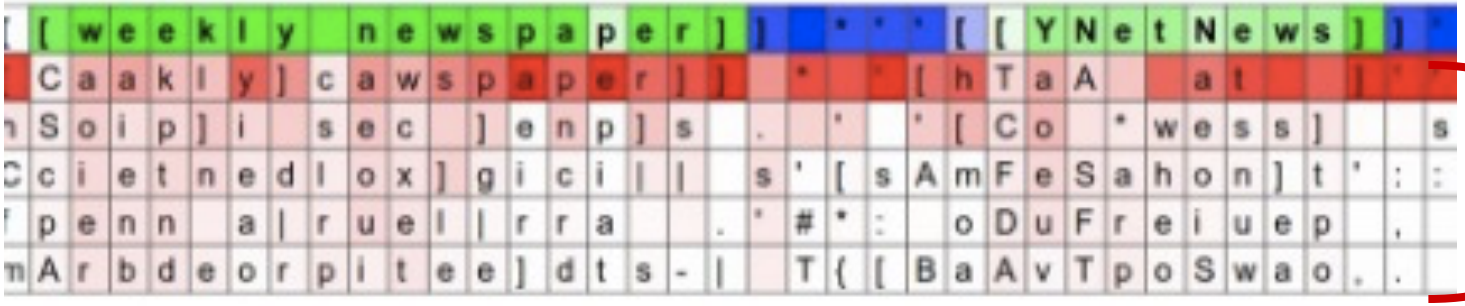
非常に興奮

URLの内部にいるという記憶

興奮していない　非常に興奮　興奮していない

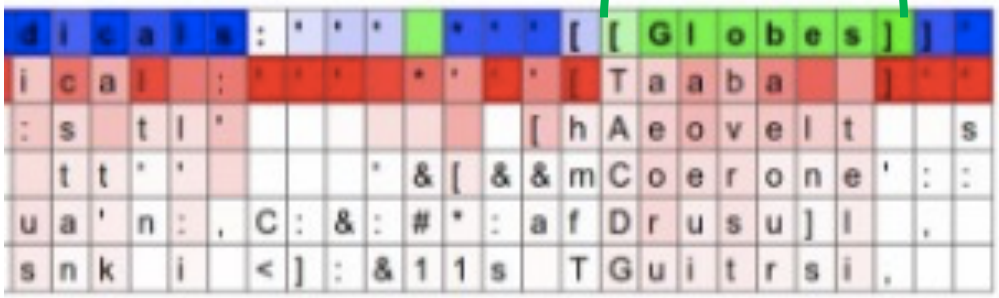
上位５つの候補
次の文字の予測

非常に興奮　興奮していない　非常に興奮



上位５つの候補
次の文字の予測

興奮していない　非常に興奮



マークダウンの
[[…]]の
内部にいると
いう記憶

文末に反応するCell

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

引用符""の内側のみに反応するCell

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

if 文の内側のみに反応するCell

```
static int __dequeue_signal(struct sigpending *pending, sigset
    siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

式の深さに反応するCell

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

コメントと引用符に反応するCell

```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                       struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
            df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

改行と ')' に反応するCell

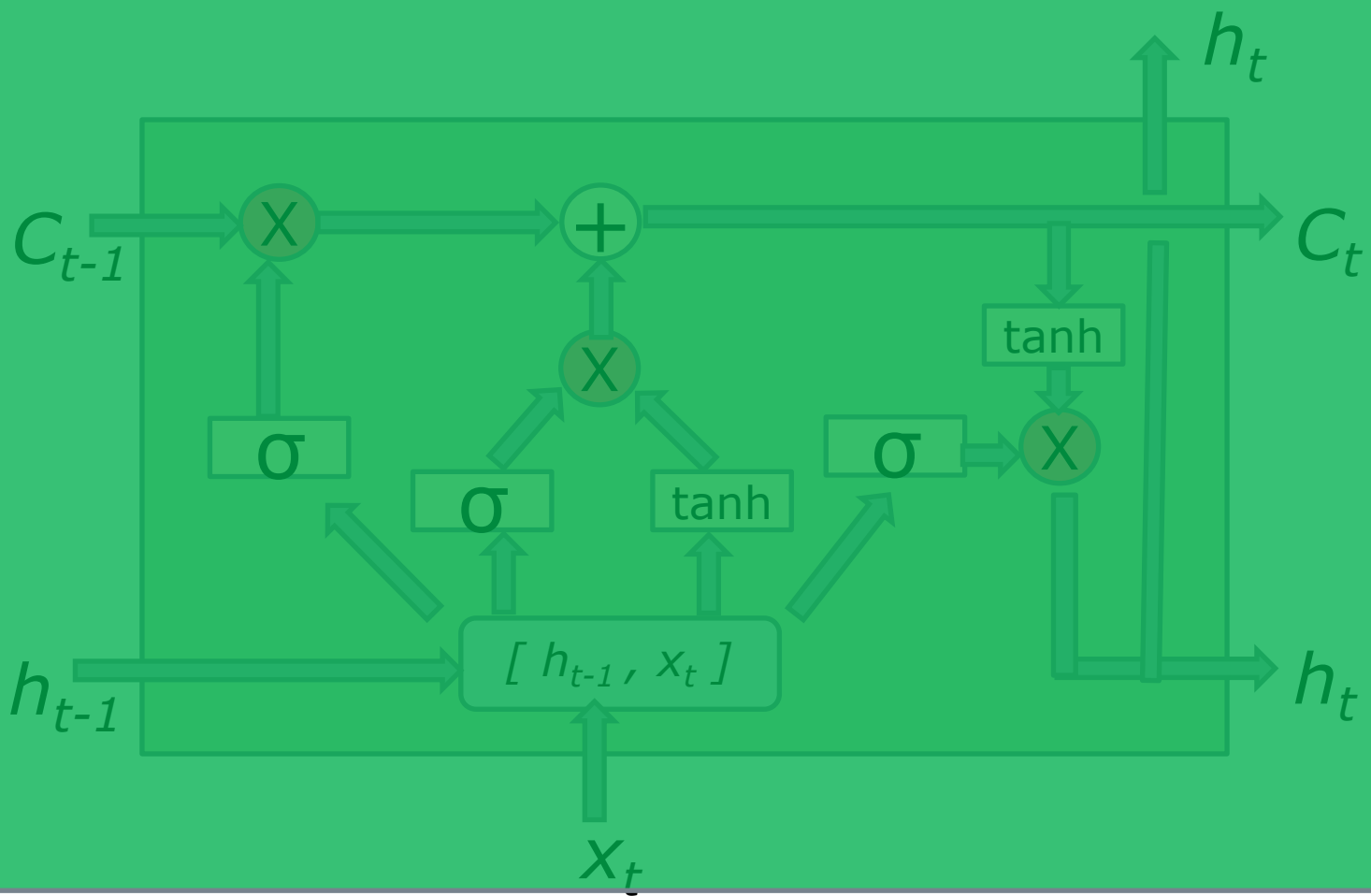
```
char *audit_unpack_string(void **bufp, size_t *remain, si
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
  if (len > PATH_MAX)
    return ERR_PTR(-ENAMETOOLONG);
  str = kmalloc(len + 1, GFP_KERNEL);
  if (unlikely(!str))
    return ERR_PTR(-ENOMEM);
  memcpy(str, *bufp, len);
  str[len] = 0;
  *bufp += len;
  *remain -= len;
  return str;
}
```

しかし、ほとんど大部分のCellの反応は、解釈不能である。

こんな感じ。

```
/*  Unpack a filter field's string representation from user-space
 *  buffer.  */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /*  Of the currently implemented string fields, PATH_MAX
   *  defines the longest valid length.
   */
```

LSTM -- Gateを持つRNN



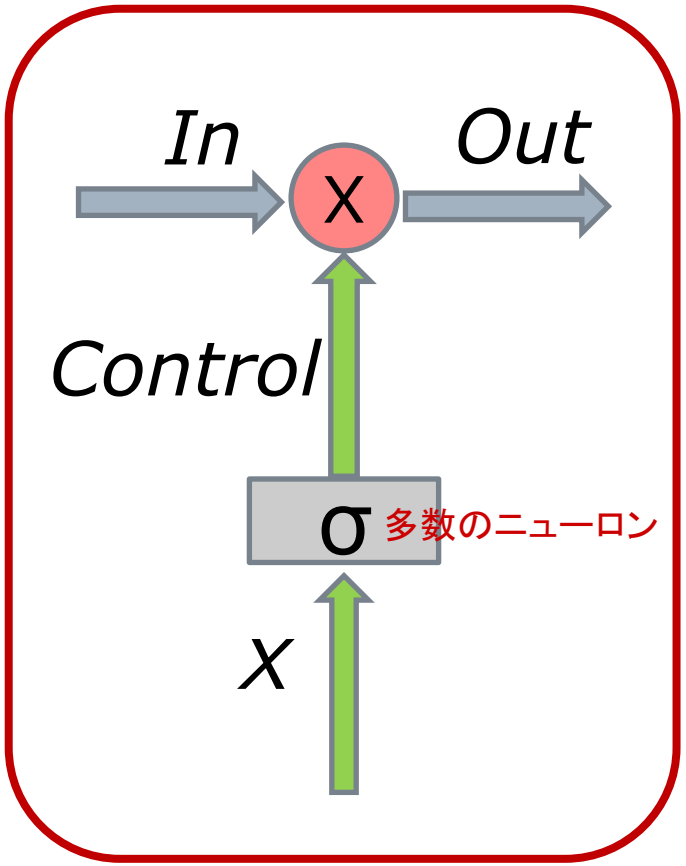
- RNNは、基本的には、単純な三層構造を持つネットワークをユニットとして、その隣り合う隠れ層同士をフルコネクトで横につなげたものだ。ただし、Feed ForwardのDNNのように、実際に、ユニットを積み重ねるのではなく、隠れ層を結合する重みのパラメーターを共有し、再帰的にシステムを記述する。
- RNNの発展系LSTMも、こうしたRNNの基本的なアイデアを踏襲している。ただ、組み合わせの基本となるユニットが少し複雑な構成をしている。LSTMのユニットに導入された新しいアイデアの中心にあるのが、今回、取り上げるGateである。
- LSTMの働きを理解するのに、Gateの働きの理解は必須であるのだが、同時に、それは、LSTMの働きを理解する、最も早い近道でもある。
- ここでは、RNNやLSTMの文脈を離れて、ニューラル・ネットワーク上のGate回路について見てみようと思う。
- 次の図の左側が、Gateの構成をグラフで示したものである。Gateは、入力（ In ）を出力（ Out ）に渡すのだが、その流れをGateに与えられる第三の情報（ X ）でコントロールするのだ。

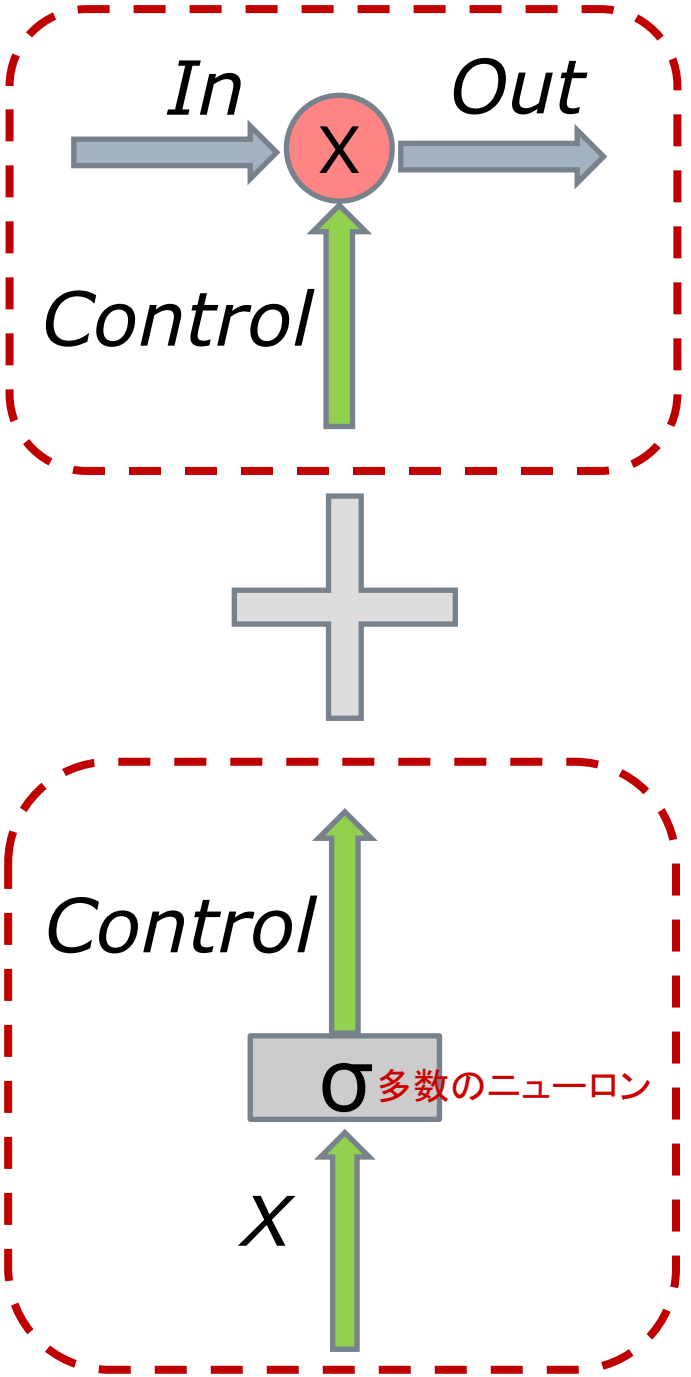
Gateを構成している二つの回路

- 一つの回路（図右下）は、Gateに与えられる情報 Xを、Gateを直接コントロールする情報 Controlに変える。もう一つの回路は、Control 情報の元で、入力 In を出力 Out に変える。
- Gate内部で、XからControlを生成する回路（図右下）は、Sigmoid関数を活性化関数とする一層のフルコネクトのニューラル・ネットワークである。図中の四角の中に書かれたσは、この層の活性化関数がSigmoidであることを表している。ただし、この層の重み W やバイアス b は、この図では省略されている。
- ContorolとInから、Outを生成する回路（図右上）が行なっているのは、簡単な演算である。二つの量を掛け合わせるだけ。ただし、掛け合わせは、ベクトルの要素ごとに行われる。これは、Hadamard積と呼ばれるものだ。

Gateのグラフ

矢印に添えられているのは、そこを流れるデータである





X ベクトルの要素ごとの積

σ Sigmoidを活性化関数とした
フルコネクトのニューロンの層

要素ごとの積（Hadamard 積）

$$(x_1, x_2, x_3, \ldots\ldots, x_n) \cdot (y_1, y_2, y_3, \ldots\ldots, y_n)$$
$$= (x_1y_1, x_2y_2, x_3y_3, , \ldots\ldots, x_ny_n)$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \\ x_n \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \\ y_n \end{pmatrix} = \begin{pmatrix} x_1y_1 \\ x_2y_2 \\ x_3y_3 \\ \\ x_ny_n \end{pmatrix}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \circ \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} & a_{13}\,b_{13} \\ a_{21}\,b_{21} & a_{22}\,b_{22} & a_{23}\,b_{23} \\ a_{31}\,b_{31} & a_{32}\,b_{32} & a_{33}\,b_{33} \end{pmatrix}$$

- こうして、Gateが行なっている働きは、次の二つの式で表すことができる。

$$Out = Control \cdot In$$

$$Control = \sigma(W \cdot X + b)$$

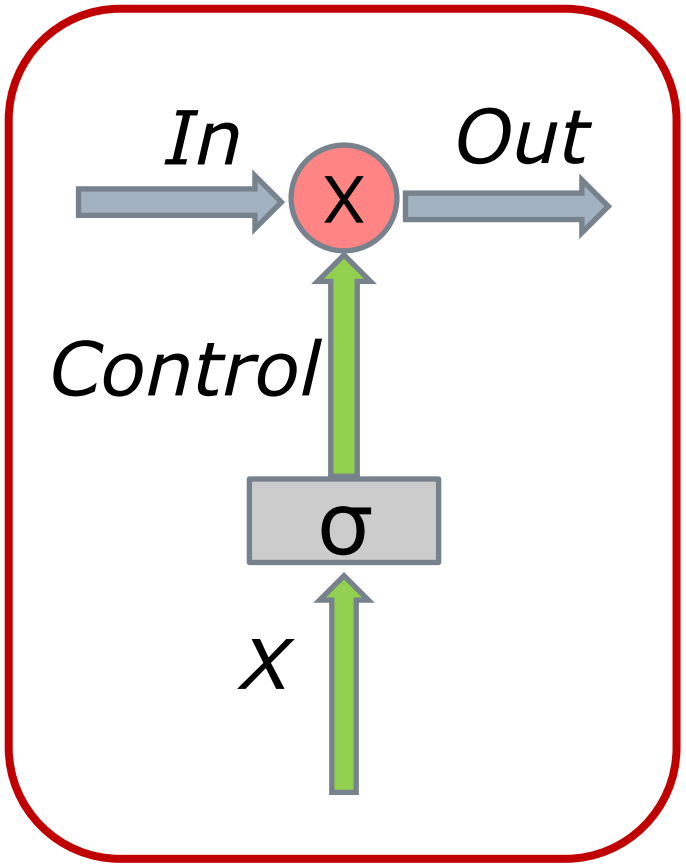
ただし、前者の積(・)は要素ごとのHadamard積、後者の積(・)は、行列とベクトルの積である。

Controlは、典型的には(σの値を、0か1としている)、次のような形のベクトルである。

$[\ 0, 1, 1, 0, 1, \ldots , 1, 1, 0\ ]$

これが、***In*** のベクトルに、要素ごとに、掛け合わされる。

Gateのグラフ



X　ベクトルの要素ごとの積

σ　Sigmoidを活性化関数とした
フルコネクトのニューロンの層

Gateの式

$$Out = Control \otimes In$$

$$Control = \sigma (WX + b)$$

Wは、この層の「重み」、Xは「入力」、
bは「バイアス」。この図では、W, bは
省略されている。

Sigmoid関数の特徴

- Control信号を生成するのに、Sigmoid関数σが使われているのには理由がある。σ(x)は、0と1の間の値をとるのだが、xが正の時には、ほとんどのxについてσ(x)は、ほぼ 1となり、xが負の時には、ほとんどのxについてσ(x)は、ほぼ 0となる。

$\sigma(x)$は。
0と1の間の
値をとる。

大部分のxに
対して、 $\sigma(x)$は、
0または1の値を
取る。

Controlは、入力に対して要素ごとに作用する

- こうして、(ほぼ)0,1からなるベクトルControlとベクトルIn との、対応するi番目の要素ごとの積を考えると。

$Control_i$が 0の時、Gateの出力のi番目の要素は、

$$Out_i = Control_i \cdot In_i = 0 \cdot In_i = 0$$

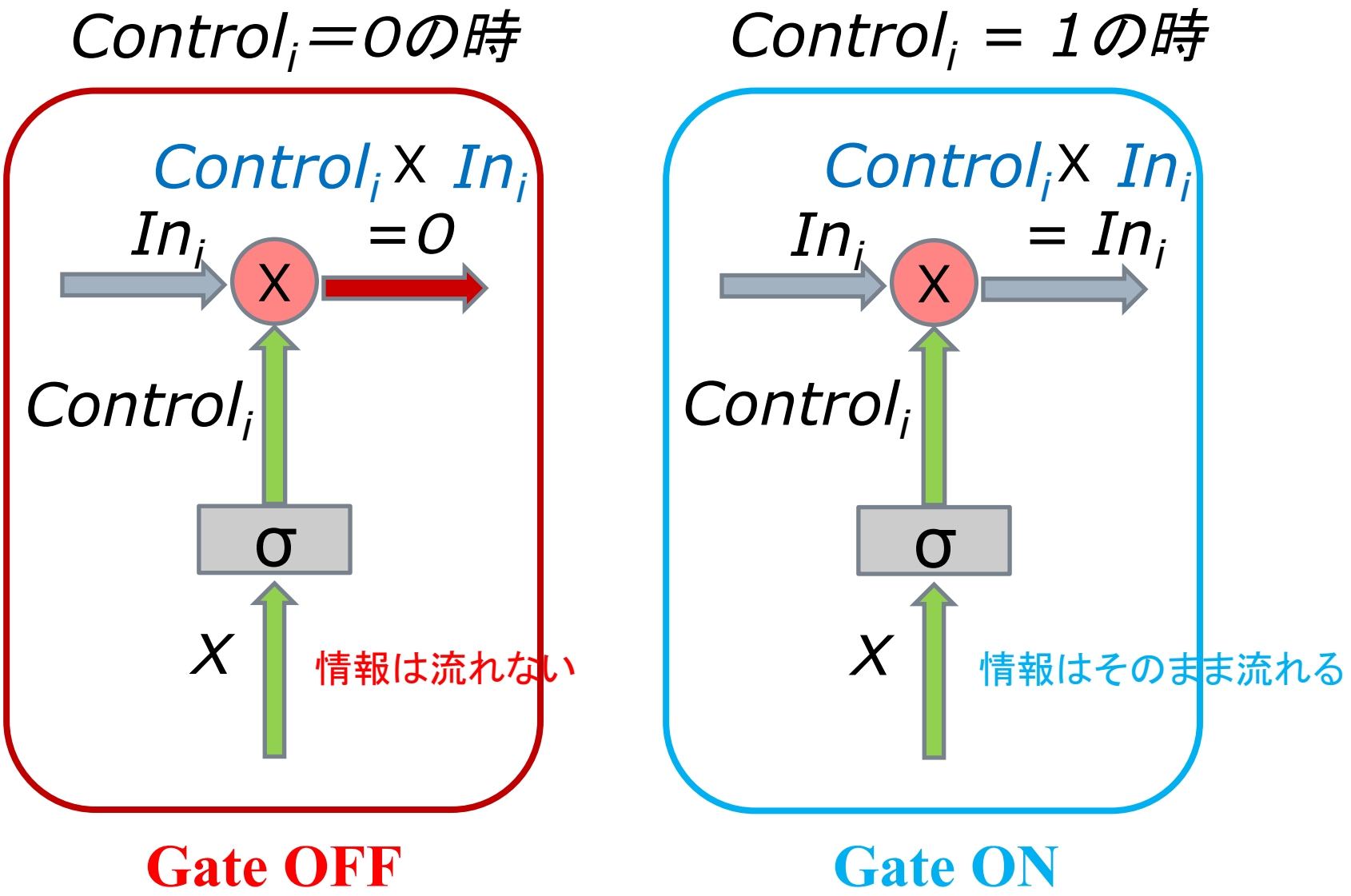
となり、情報は流れず、

$Control_i$が 1の時、Gateの出力i番目の要素は、

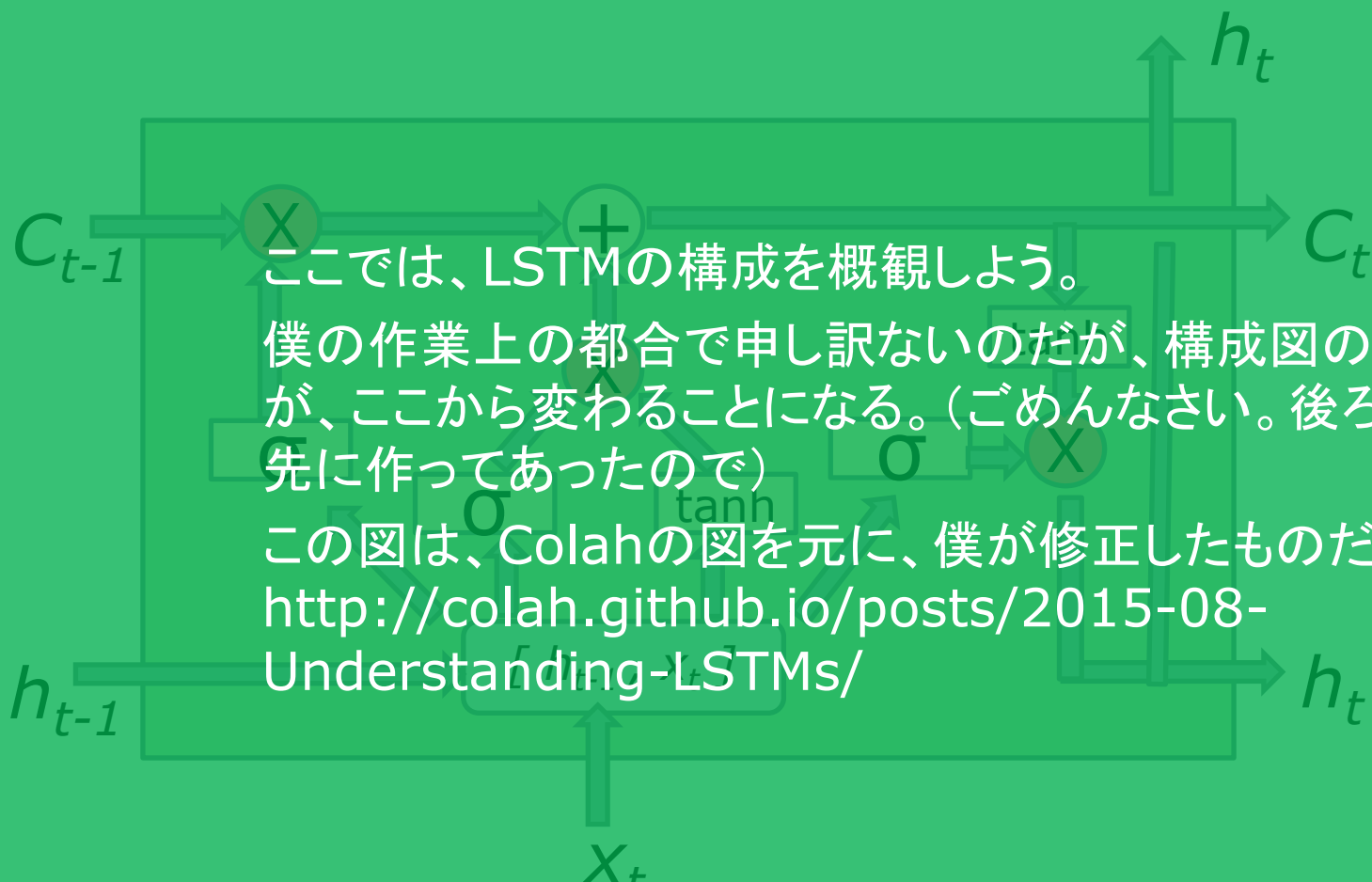
$$Out_i = Control_i \cdot In_i = 1 \cdot In_i = In_i$$

となり、情報はそのまま流れることになる。

Gateの要素ごとの振る舞い



LSTMの構成を、別のスタイルで概観する

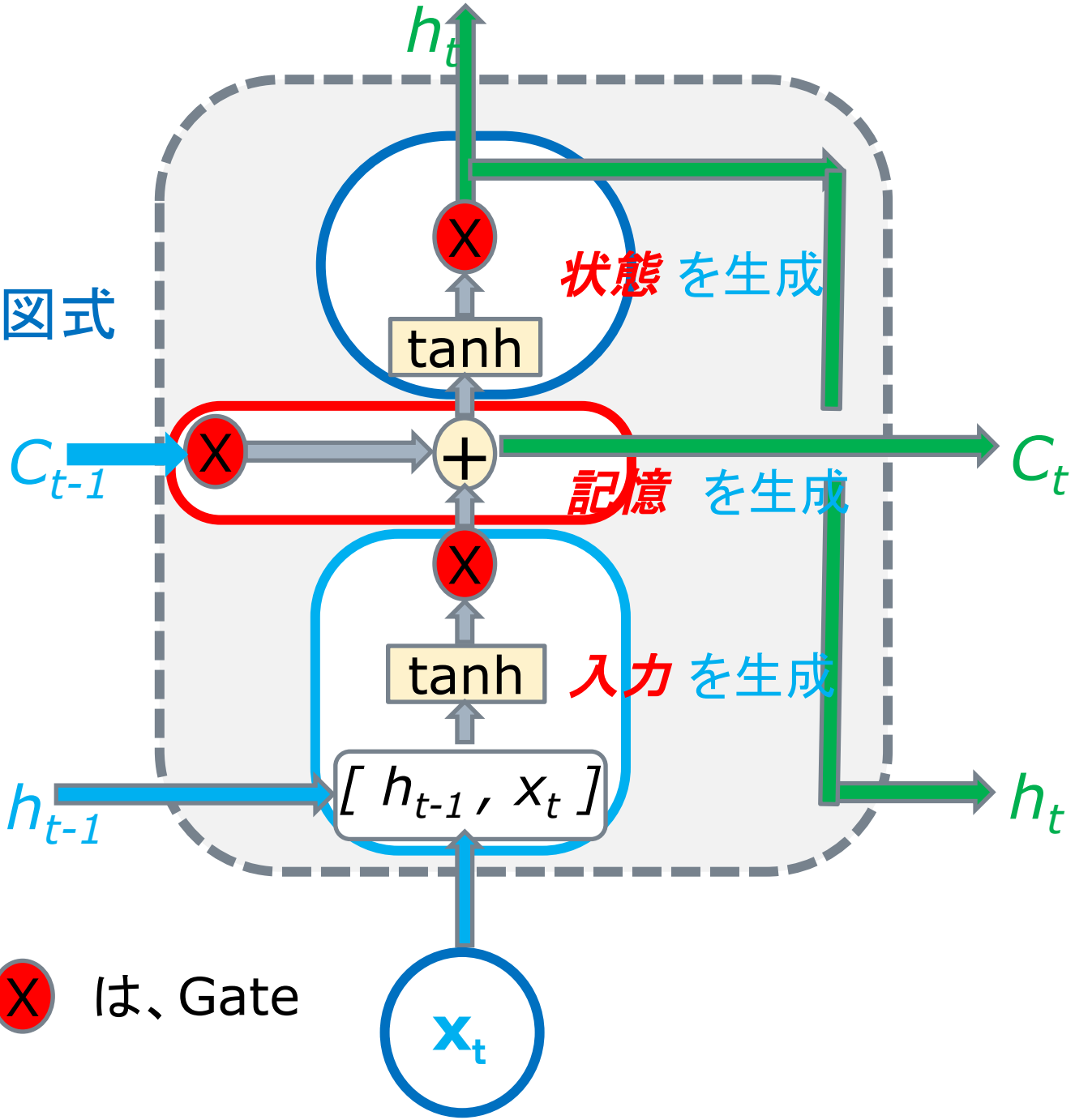


ここでは、LSTMの構成を概観しよう。

僕の作業上の都合で申し訳ないのだが、構成図のスタイルが、ここから変わることになる。（ごめんなさい。後ろの部分を先に作ってあったので）

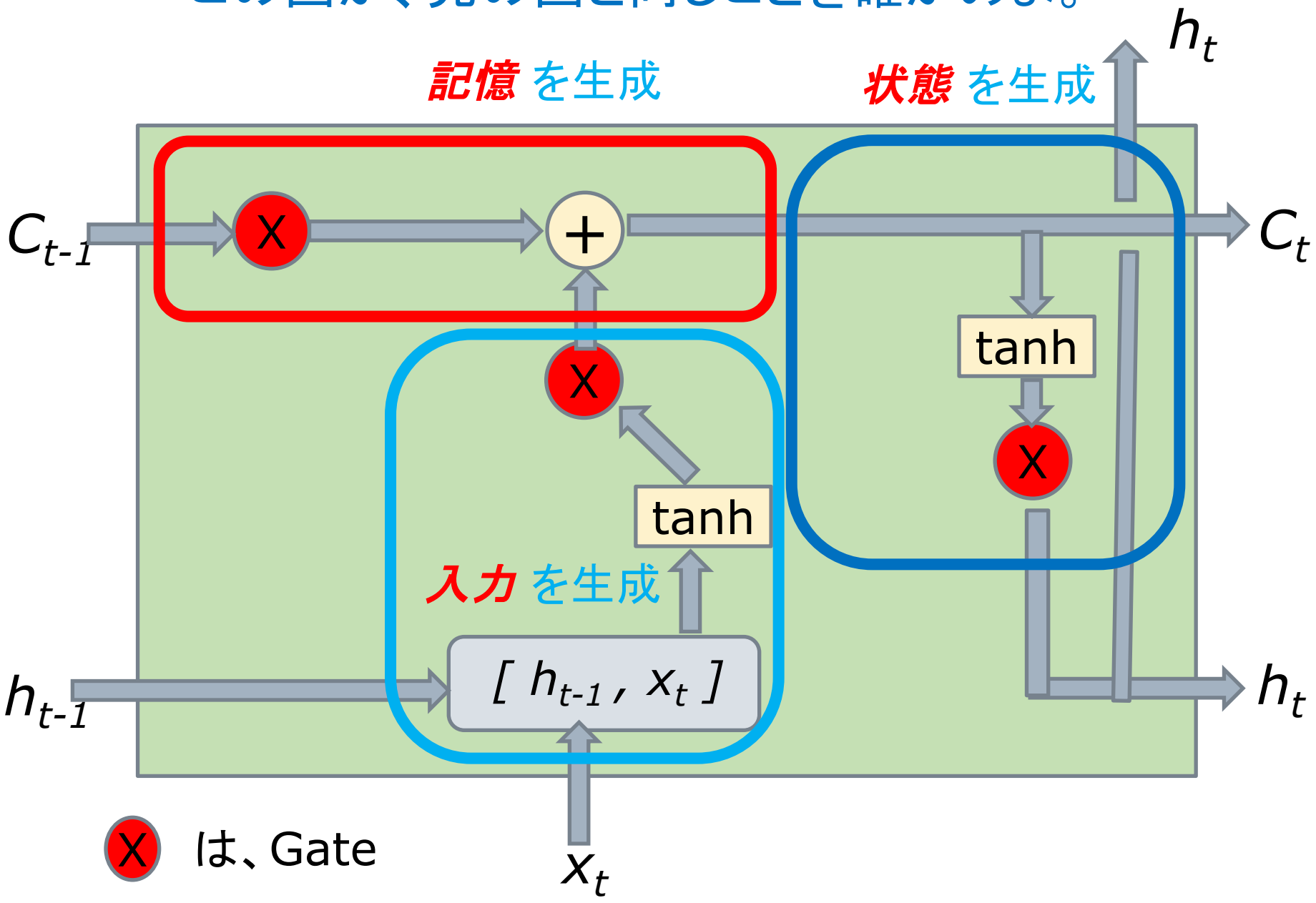
この図は、Colahの図を元に、僕が修正したものだ。
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

これまでの図式

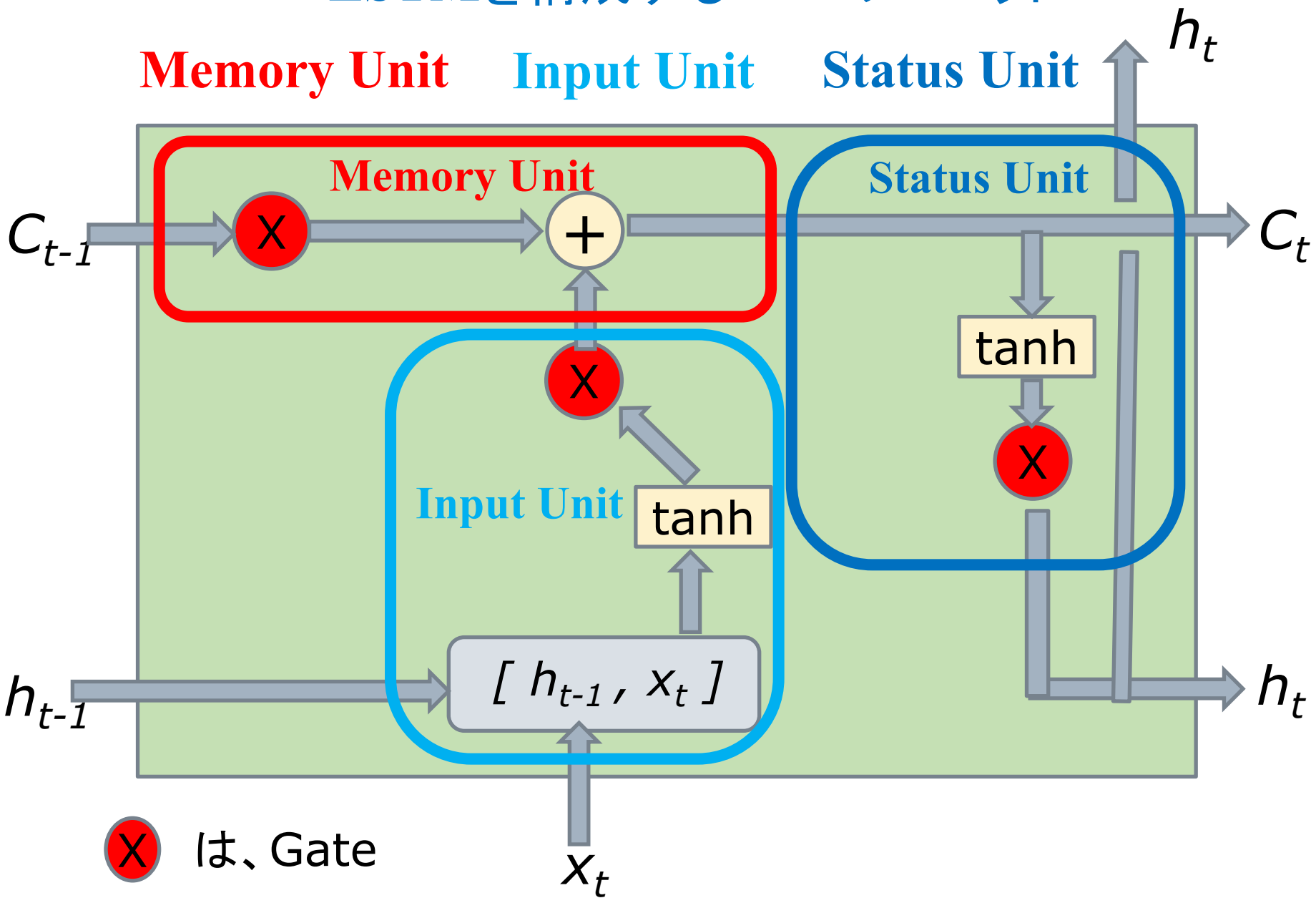


X は、Gate

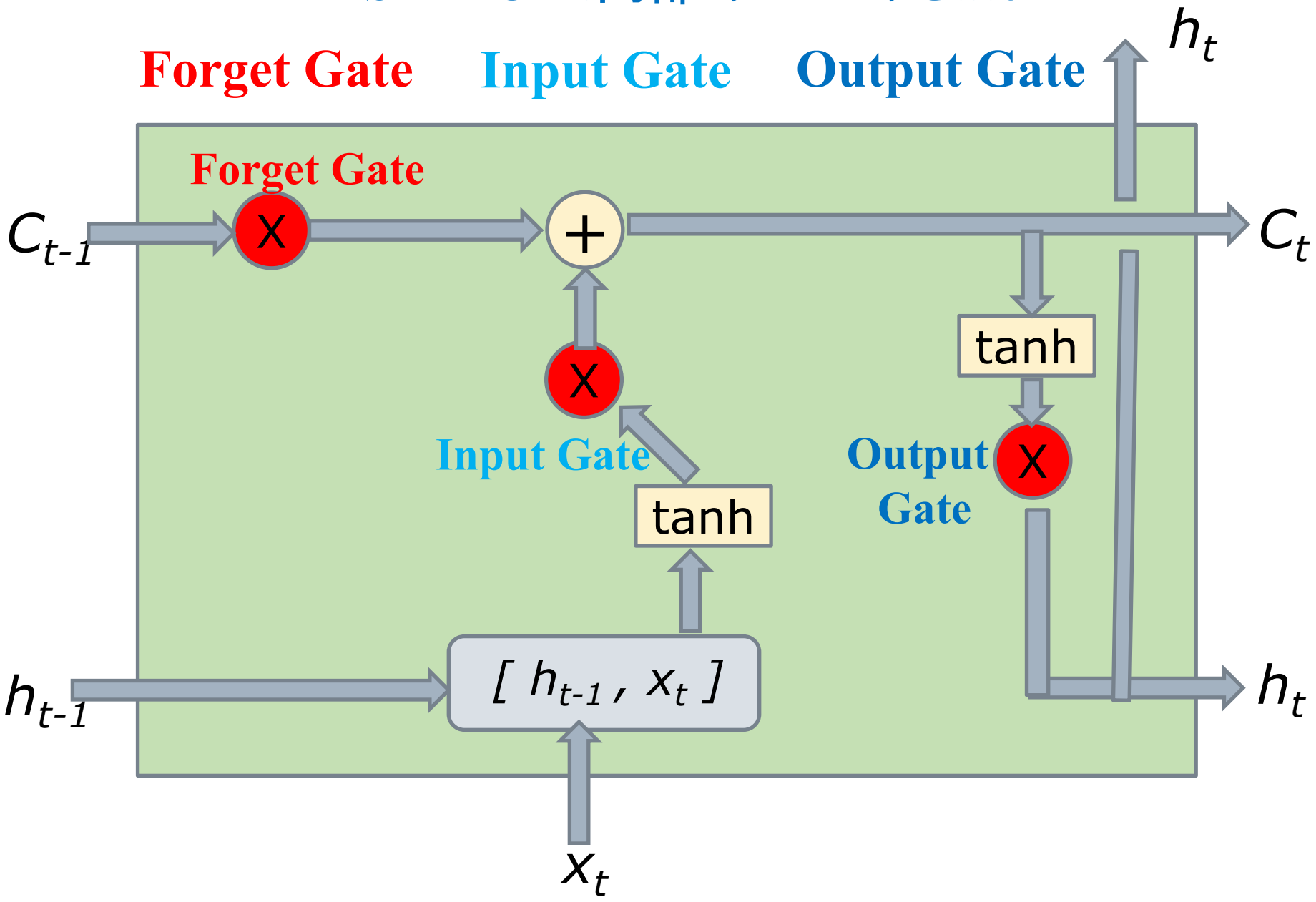
この図が、先の図と同じことを確かめよ。



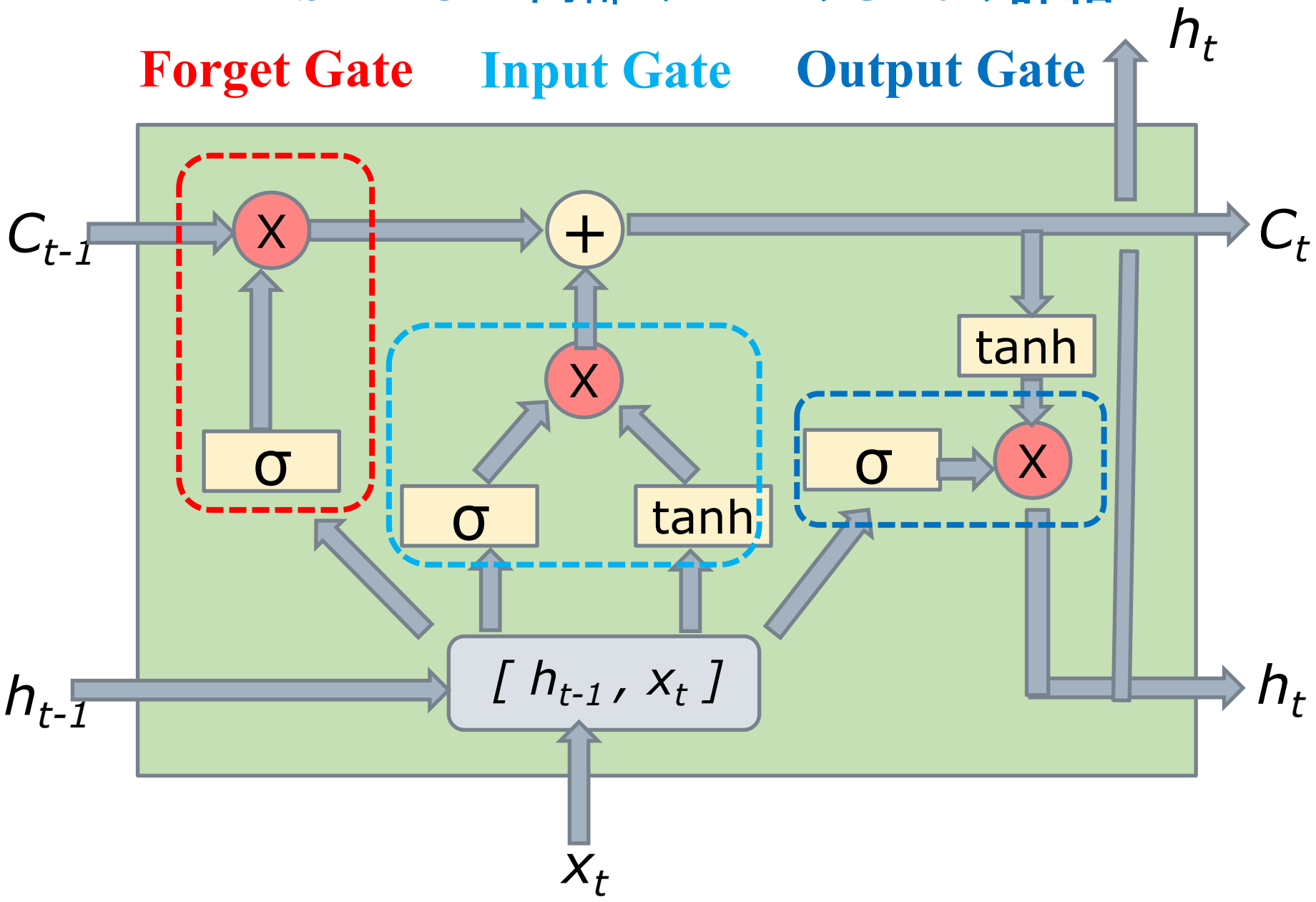
LSTMを構成する三つのユニット



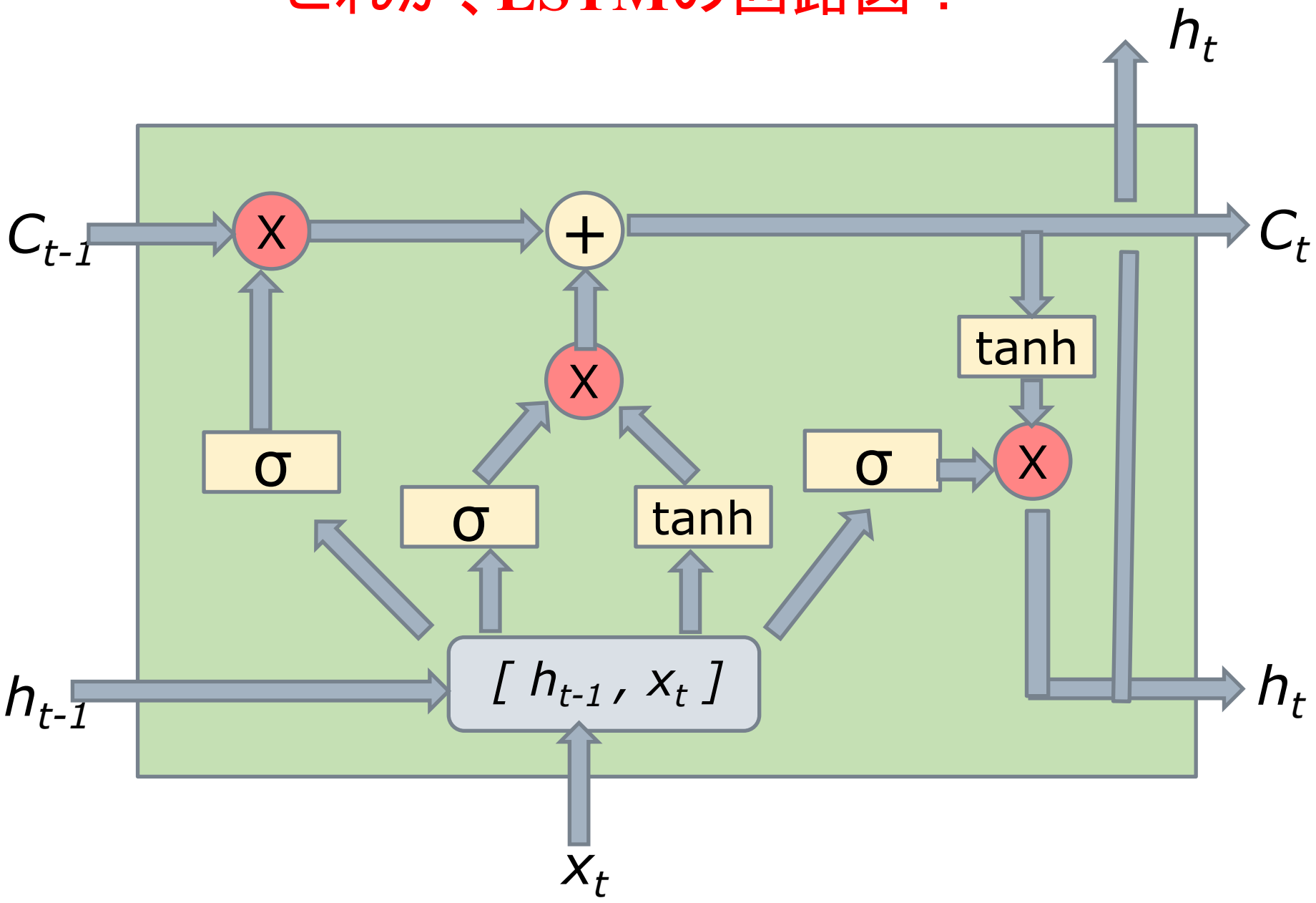
LSTM Unit内部の三つのGate



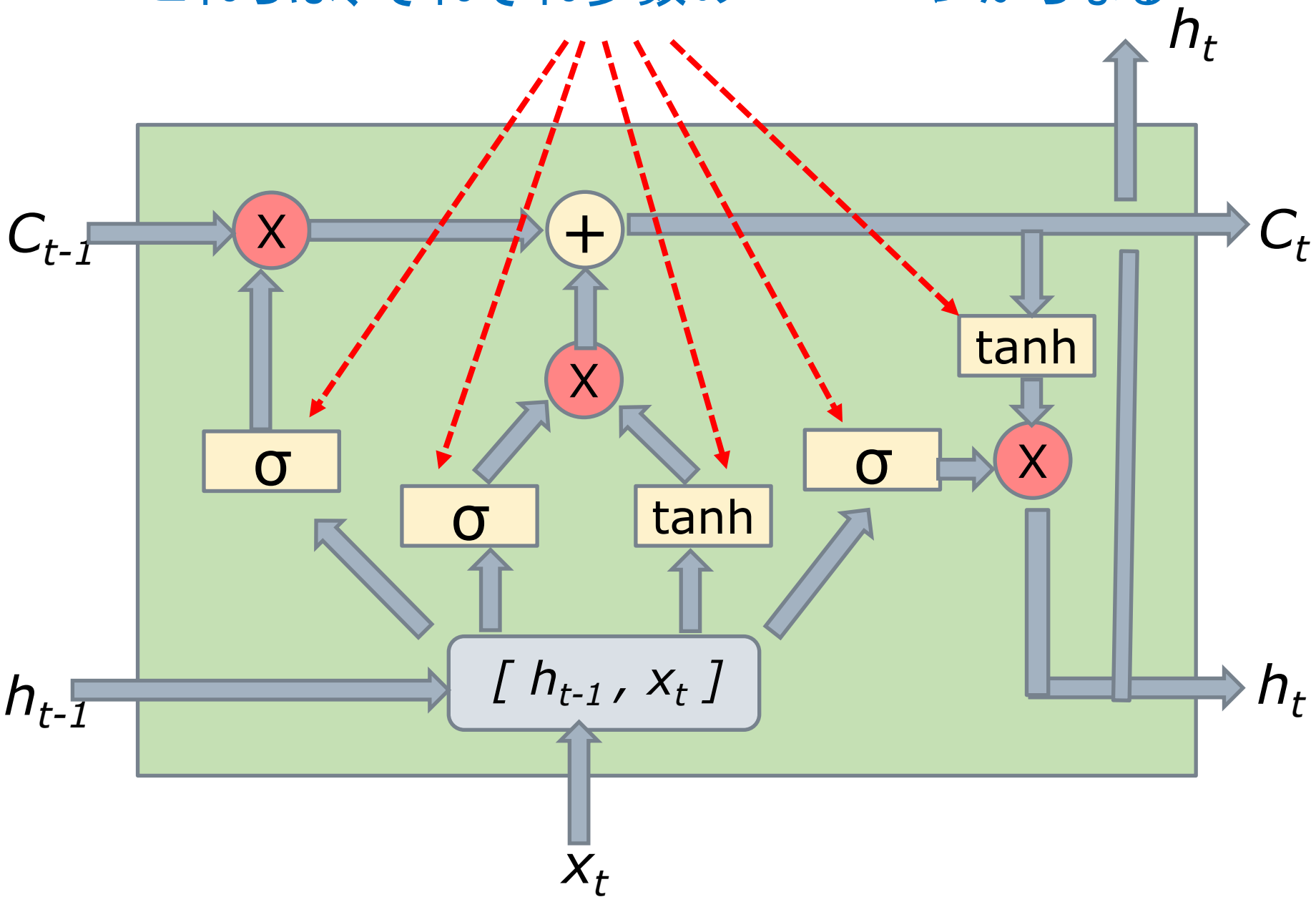
LSTM Unit内部の三つのGateの詳細



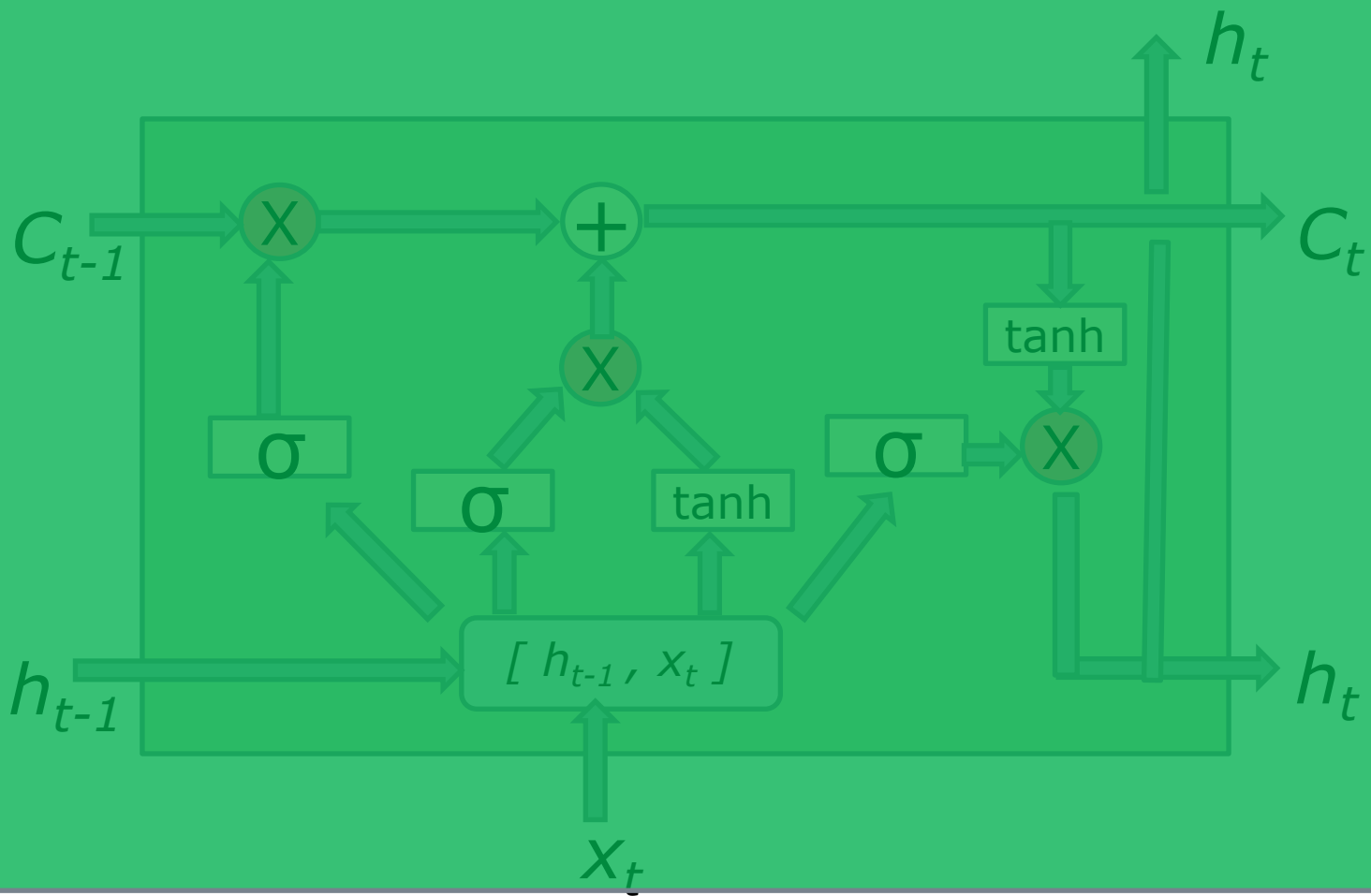
これが、LSTMの回路図！



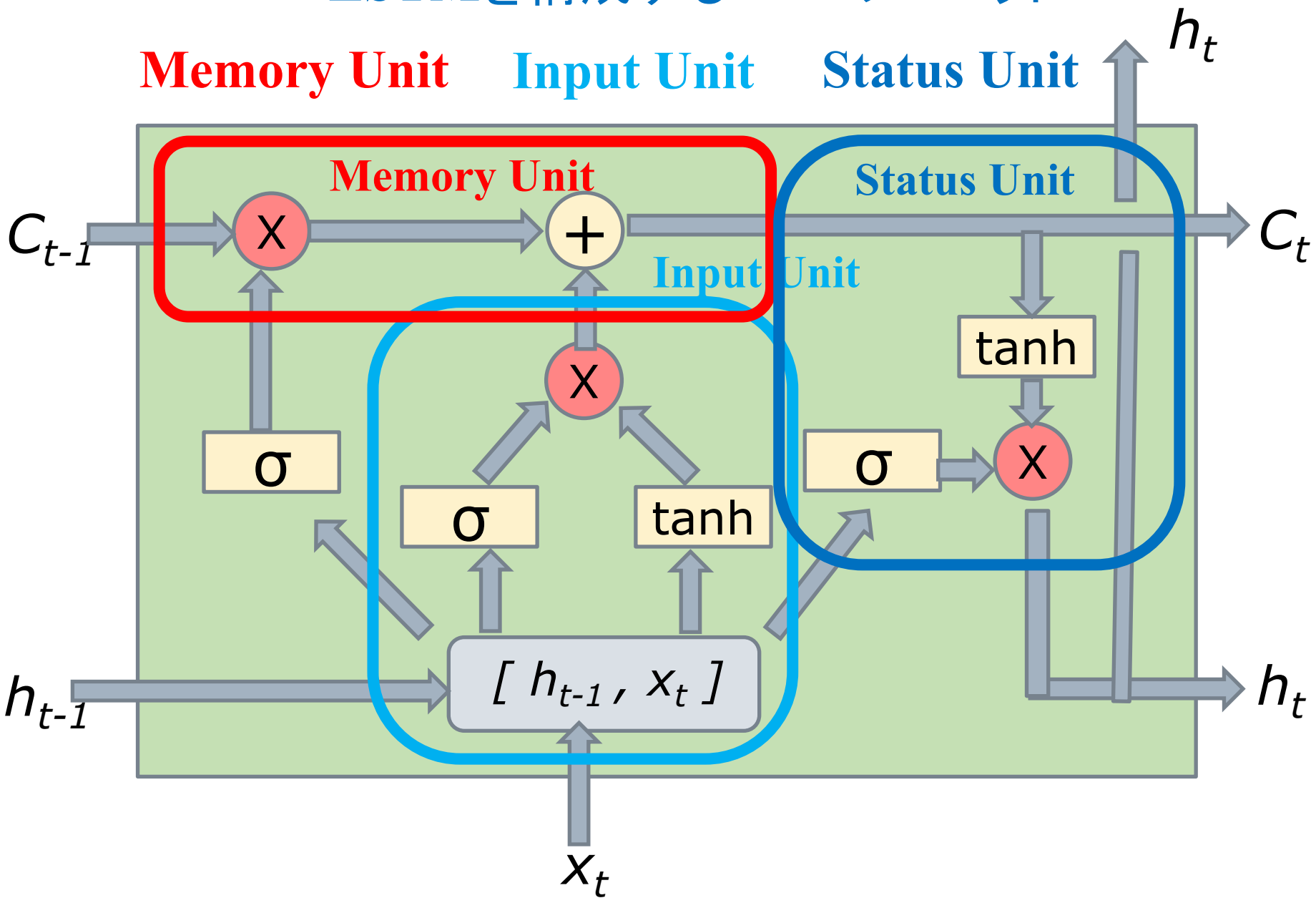
これらは、それぞれ多数のニューロンからなる



LSTMの働きを詳細に見る



LSTMを構成する三つのユニット



LSTMを構成する三つのユニットの働き

- **Input Unit**

  入力 x_t と、前の層の状態 h_{t-1} から、Memory Unitへの入力を生成する。

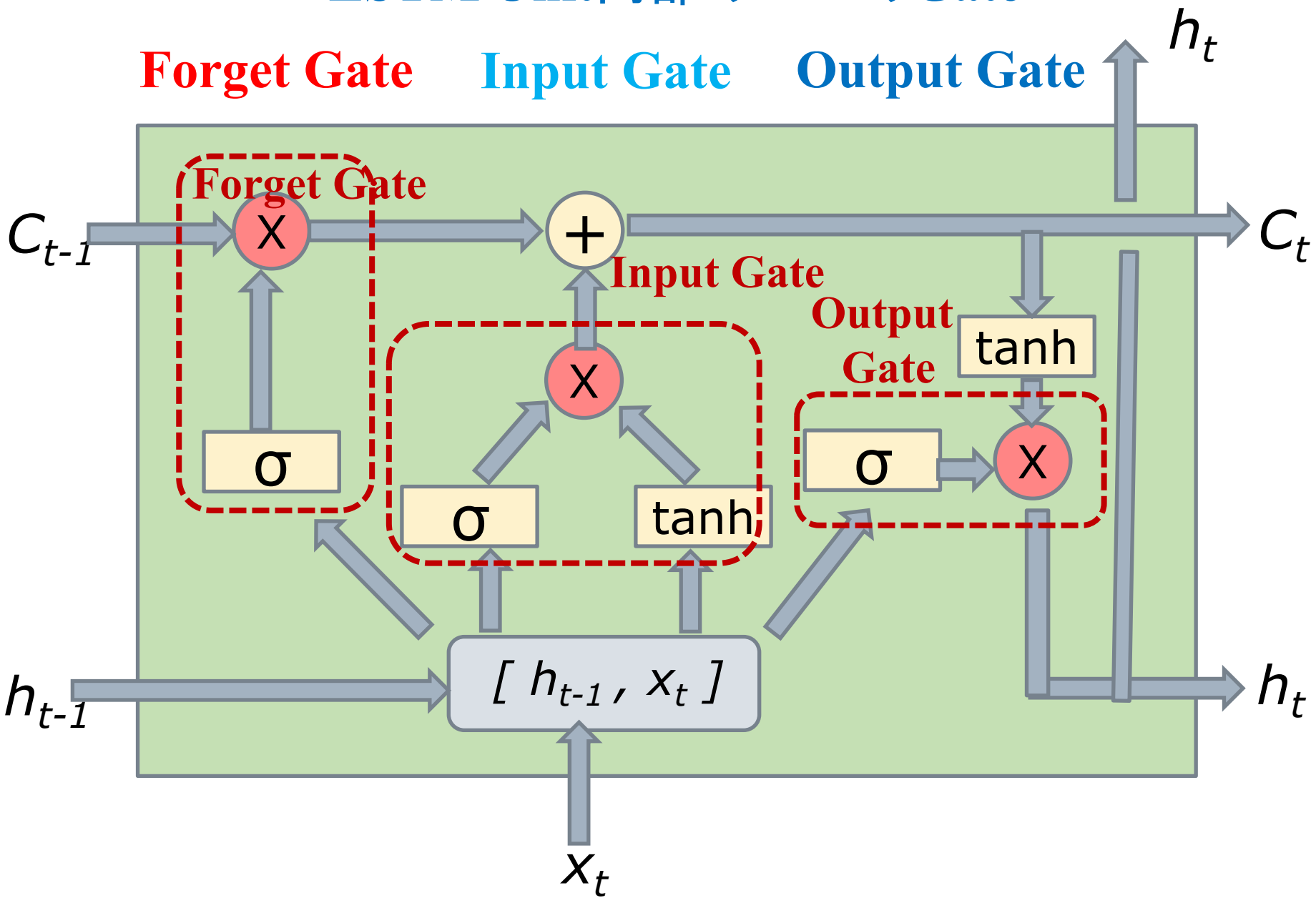
- **Memory Unit**

  Input Unitが生成した入力と、前の層の記憶 C_{t-1} から、記憶 C_t を生成・更新し、Status Unitへ渡す。

- **Status Unit**

  層の記憶 C_t から、状態 h_t を生成・更新し、出力する。

LSTM Unit内部の三つのGate



LSTMを構成する三つのGateの働き

- **Input Gate**

  入力 x_t と前の層の状態 h_{t-1} から、この情報をMemory Unitに渡すか否かをコントロールする。

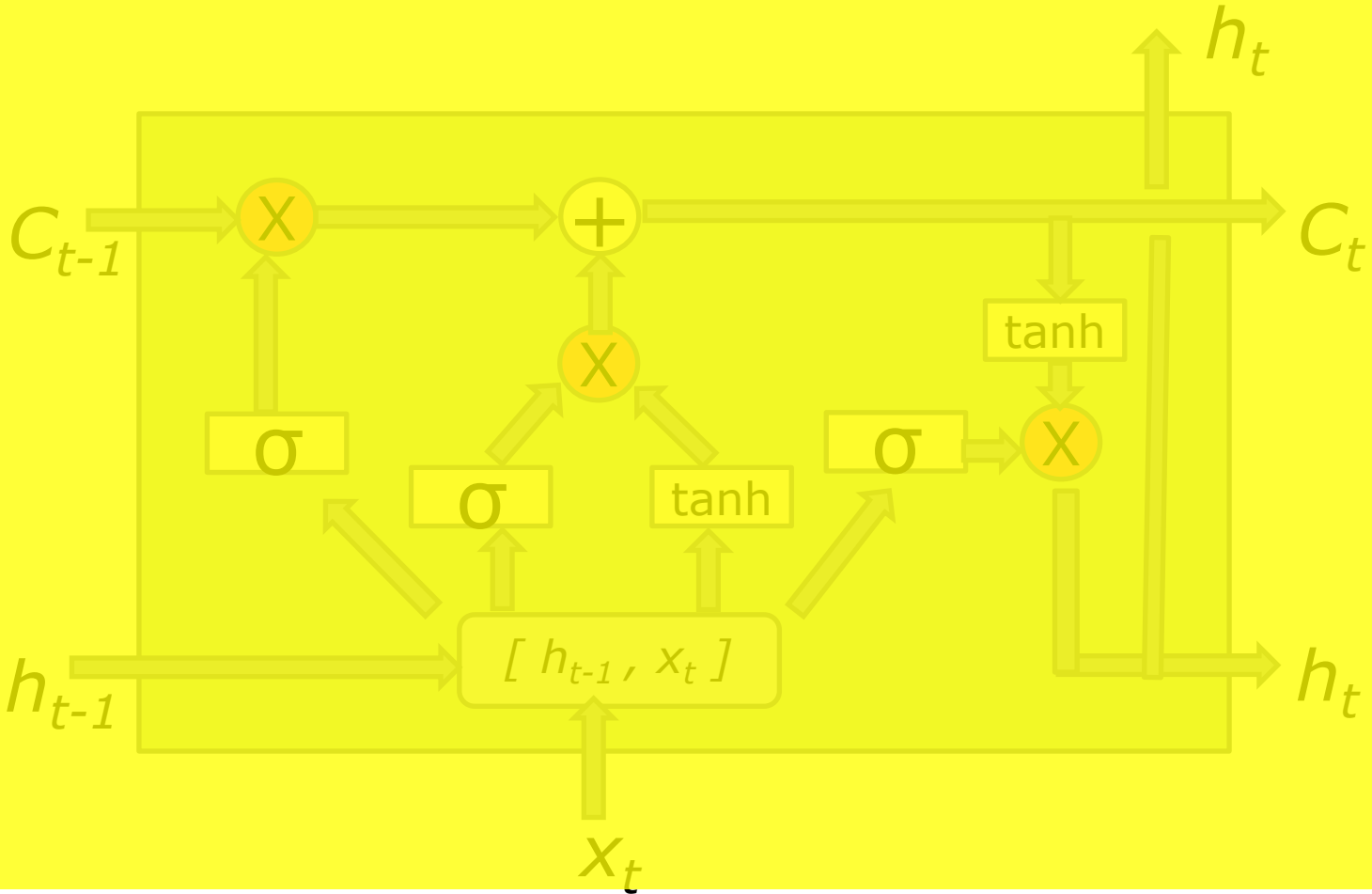
- **Forget Gate**

  入力 x_t と前の層の状態 h_{t-1} から、前の層の記憶 C_{t-1} を、Memory Unitに渡すか否かをコントロールする。

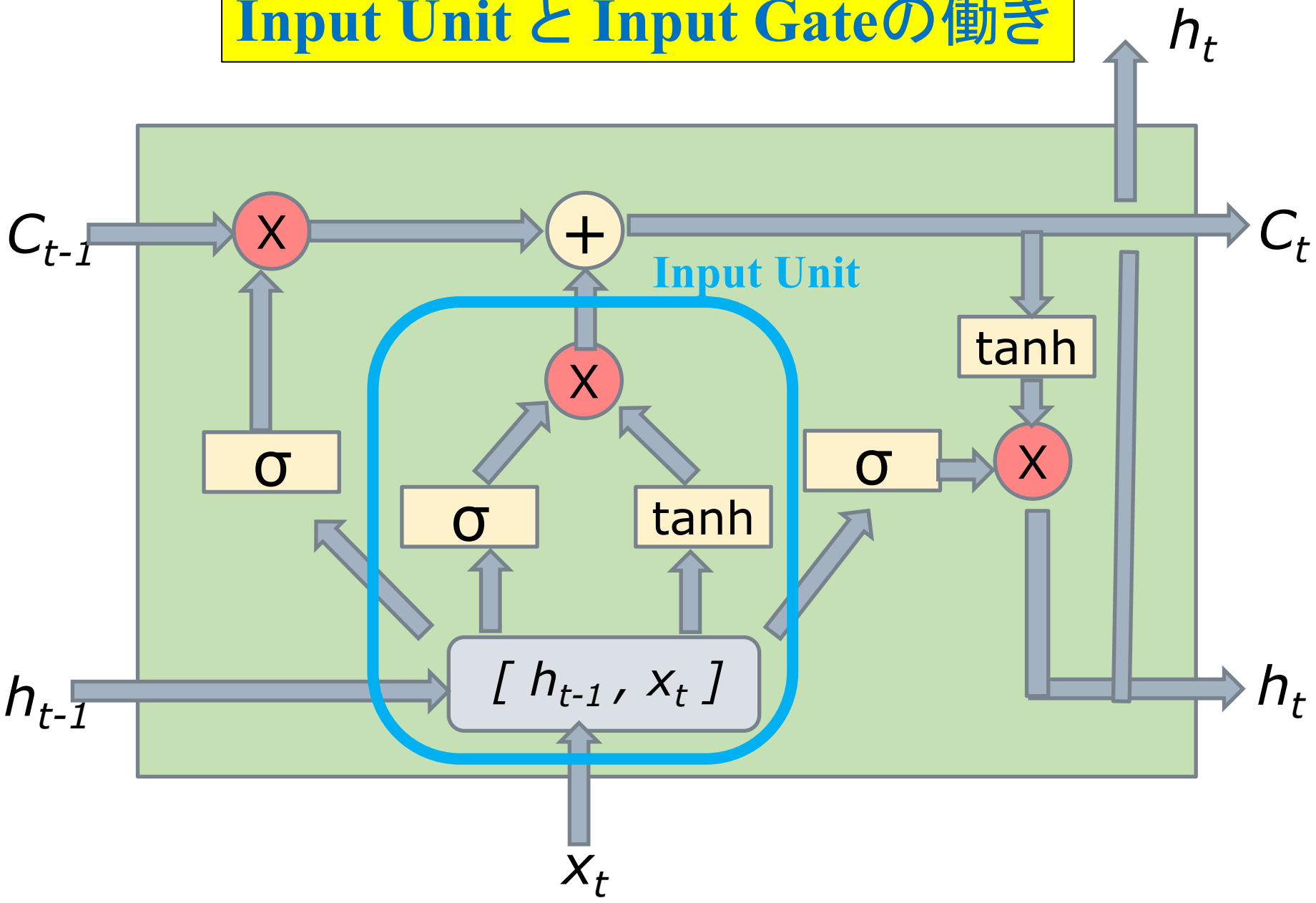
- **Output Gate**

  入力 x_t と前の層の状態 h_{t-1} から、生成・更新された状態 h_t を出力するか否かをコントロールする。

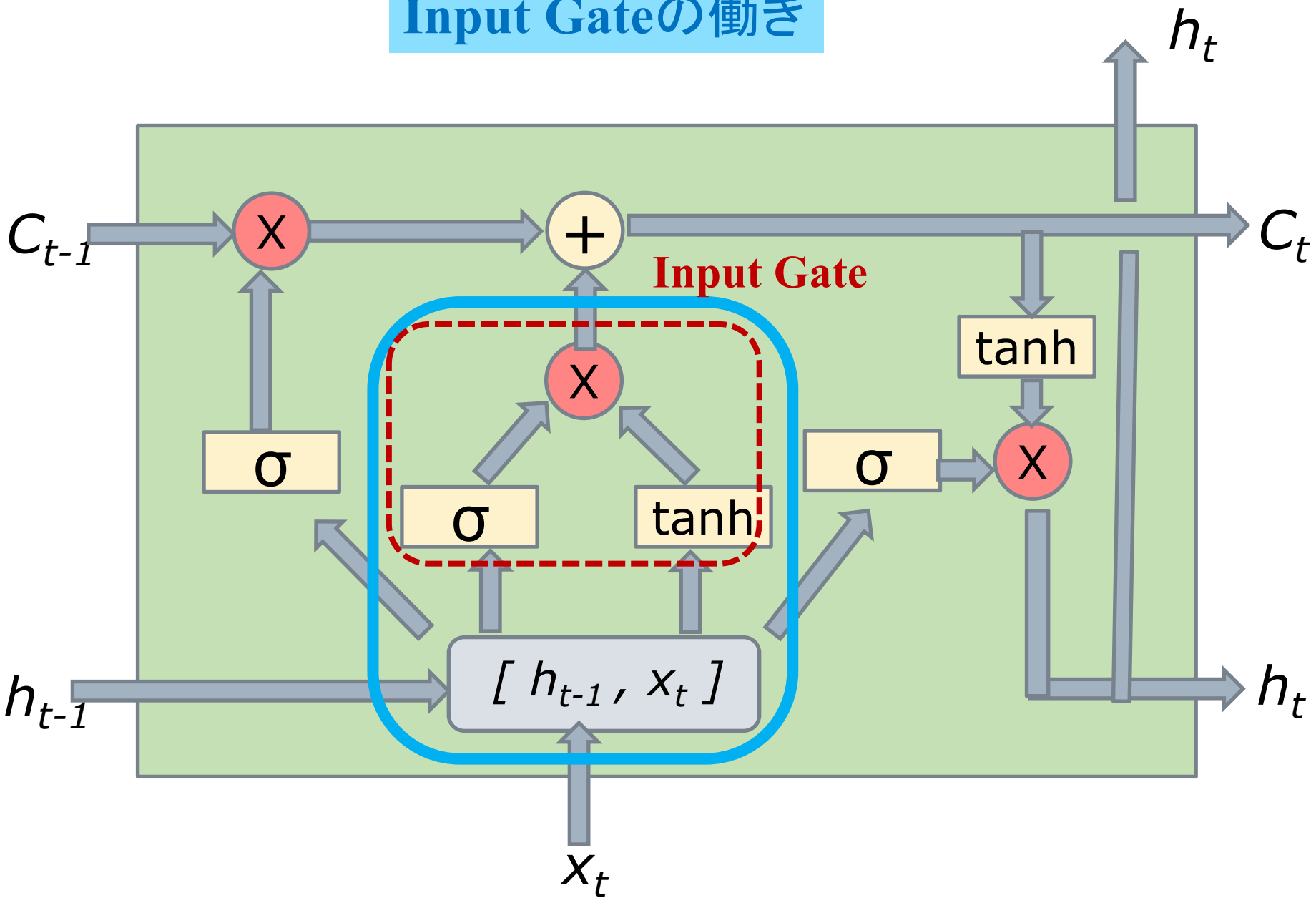
Input Unit と Input Gateの働き

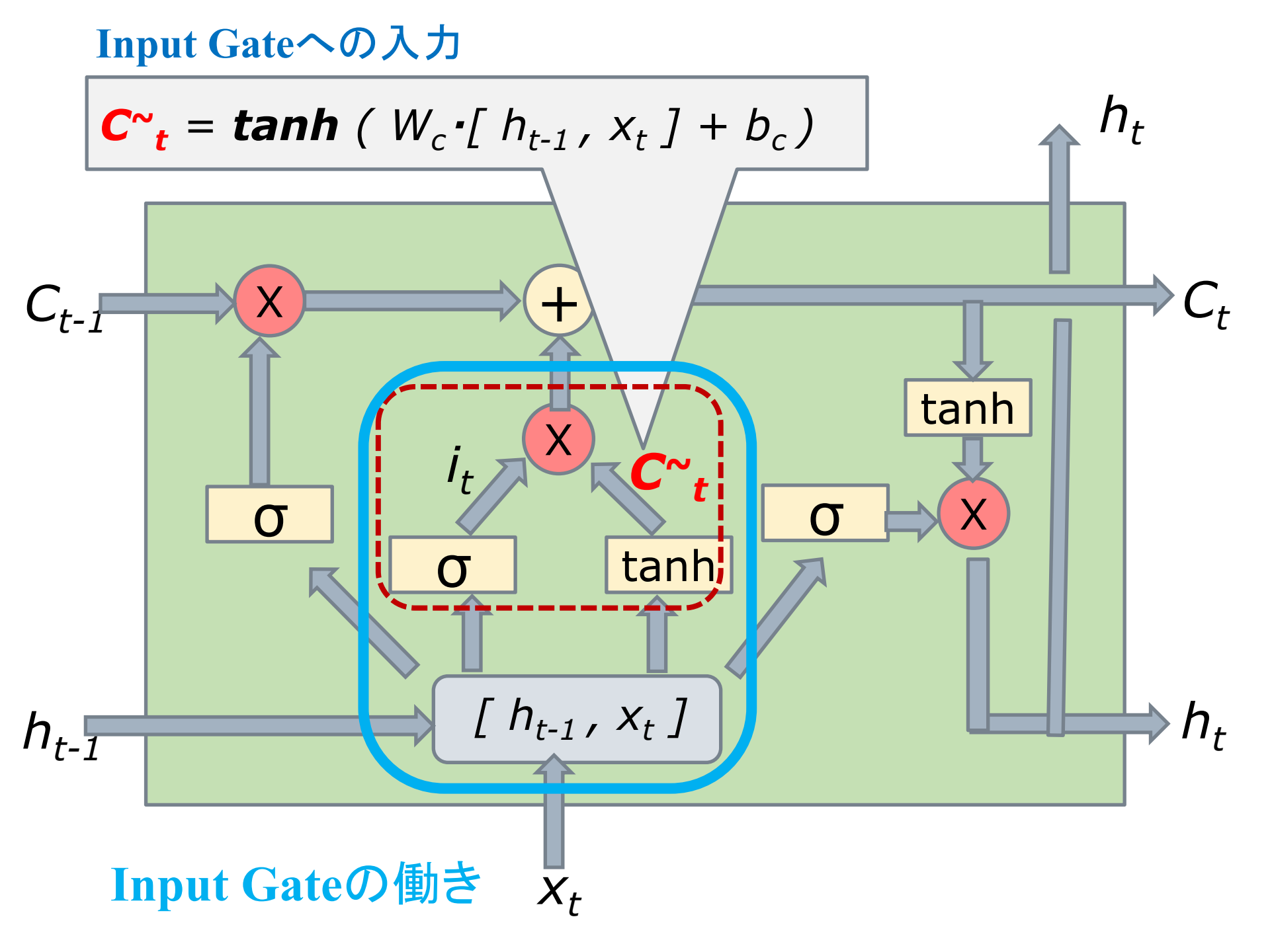


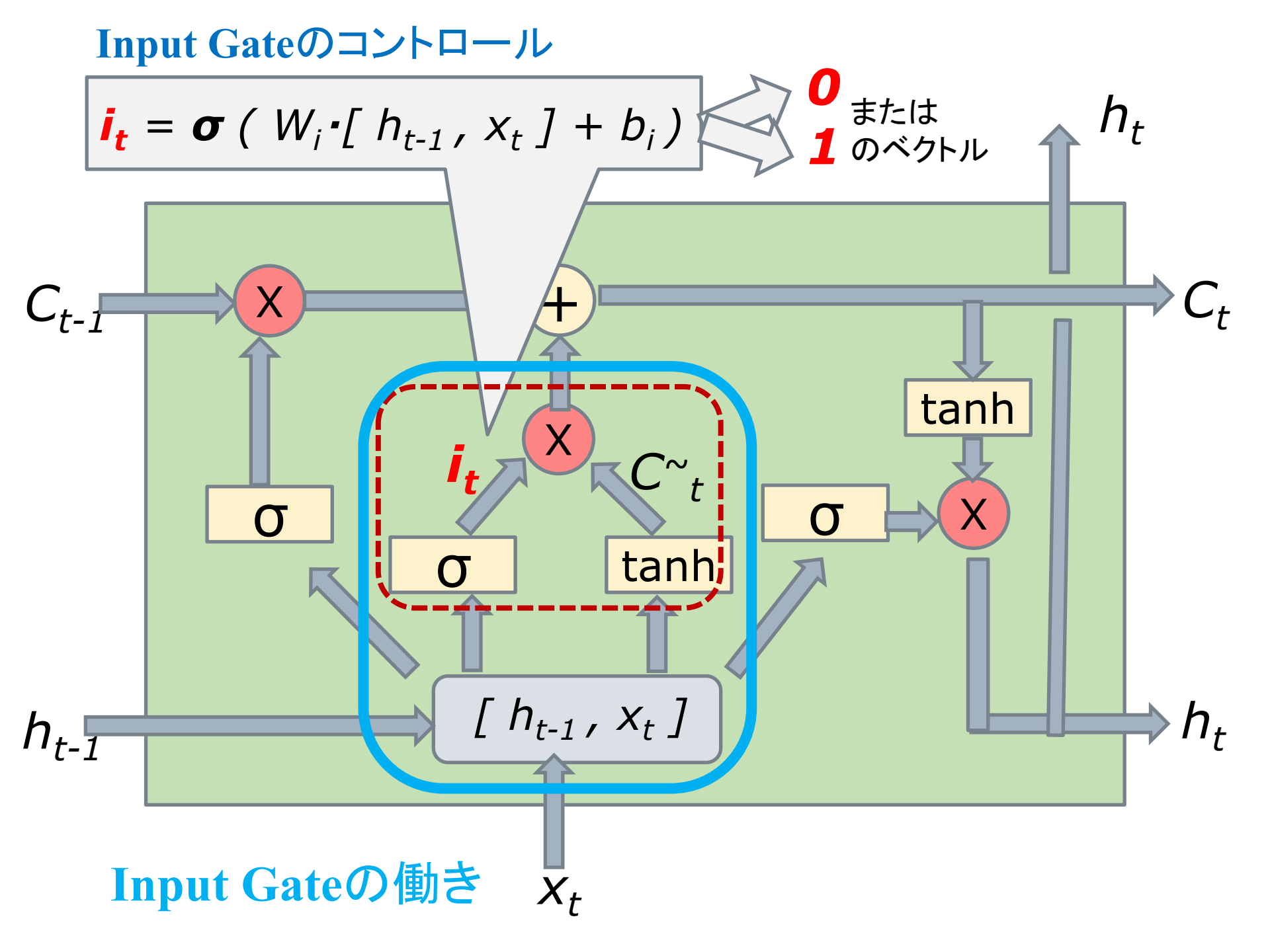
Input Unit と Input Gateの働き

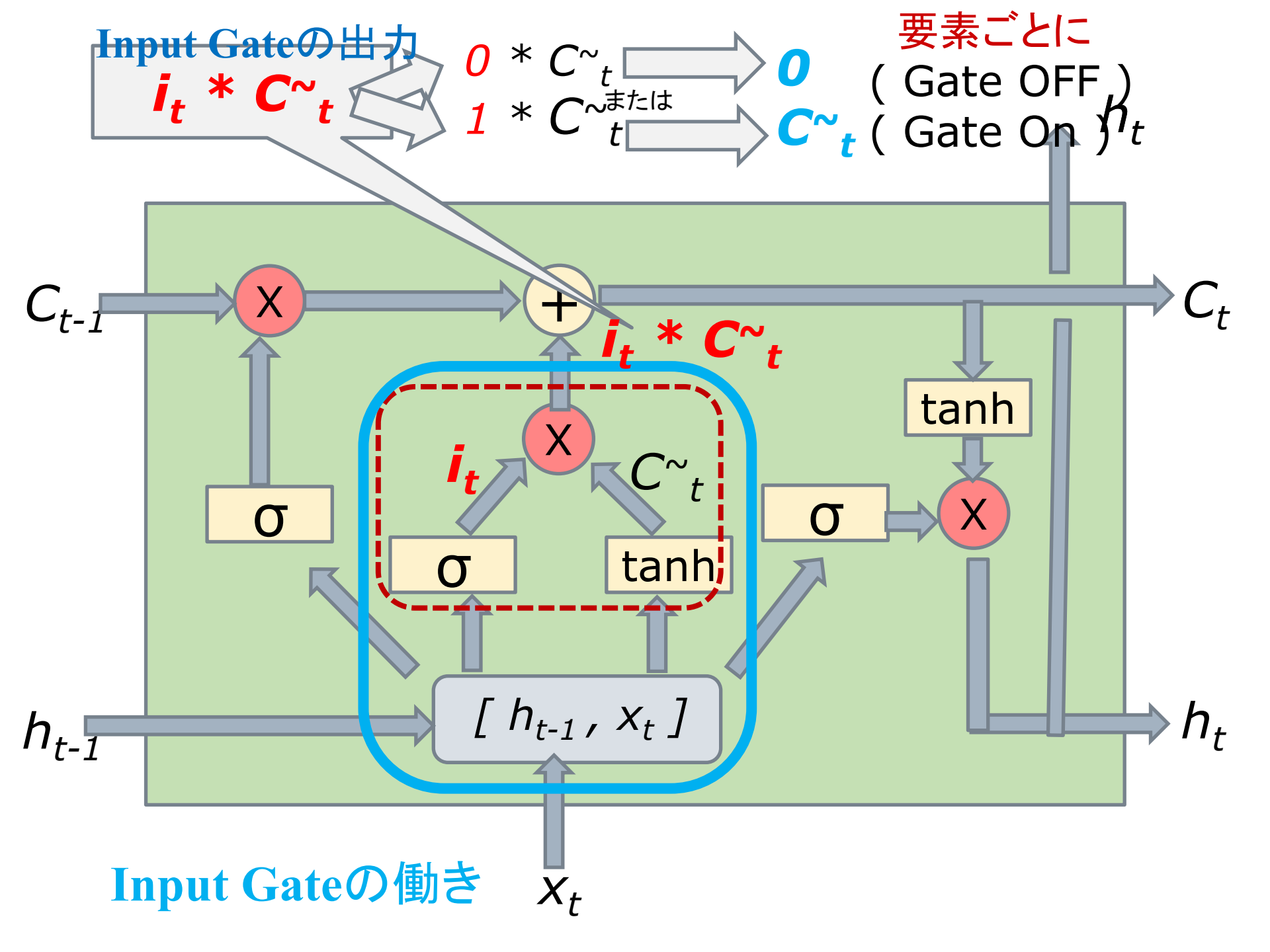


Input Gateの働き



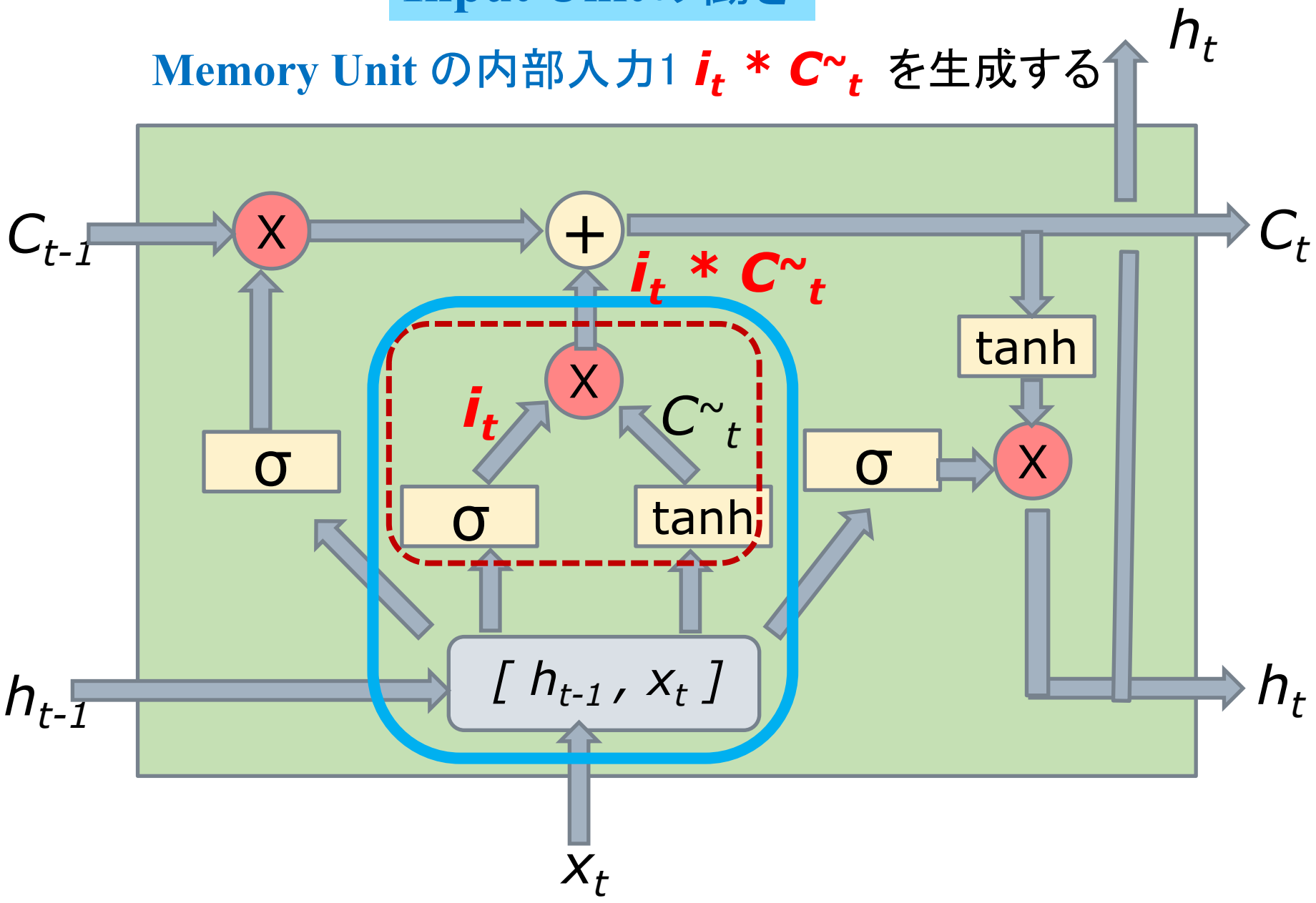

Input Gateへの入力
$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$
h_t
C_{t-1}
X
+
C_t
tanh
X
i_t
$\tilde{C}_t$
σ
σ
tanh
σ
X
$[h_{t-1}, x_t]$
h_{t-1}
h_t
x_t
Input Gateの働き


Input Gateのコントロール
$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$
0
1
またはのベクトル
h_t
C_{t-1}
C_t
X
+
tanh
i_t
X
$C\tilde{}_t$
σ
σ
tanh
σ
X
$[h_{t-1}, x_t]$
h_{t-1}
h_t
x_t
Input Gateの働き


Input Gateの出力
$i_t * C\tilde{}_t$
$0 * C\tilde{}_t$
$1 * C\tilde{}_t$
または
0
$C\tilde{}_t$
要素ごとに
(Gate OFF)
(Gate On)
h_t
C_{t-1}
X
+
$i_t * C\tilde{}_t$
C_t
tanh
X
i_t
$C\tilde{}_t$
σ
σ
tanh
σ
X
h_{t-1}
$[h_{t-1}, x_t]$
h_t
x_t
Input Gateの働き

Input Unitの働き

Memory Unit の内部入力1 $i_t * C\tilde{}_t$ を生成する



Input Unit と Input Gateの働きを式で表す

- **Input Gateへの入力:**

$$\tilde{C}_t = \tanh\left(W_c \cdot [\, h_{t-1}, x_t \,] + b_c \right)$$

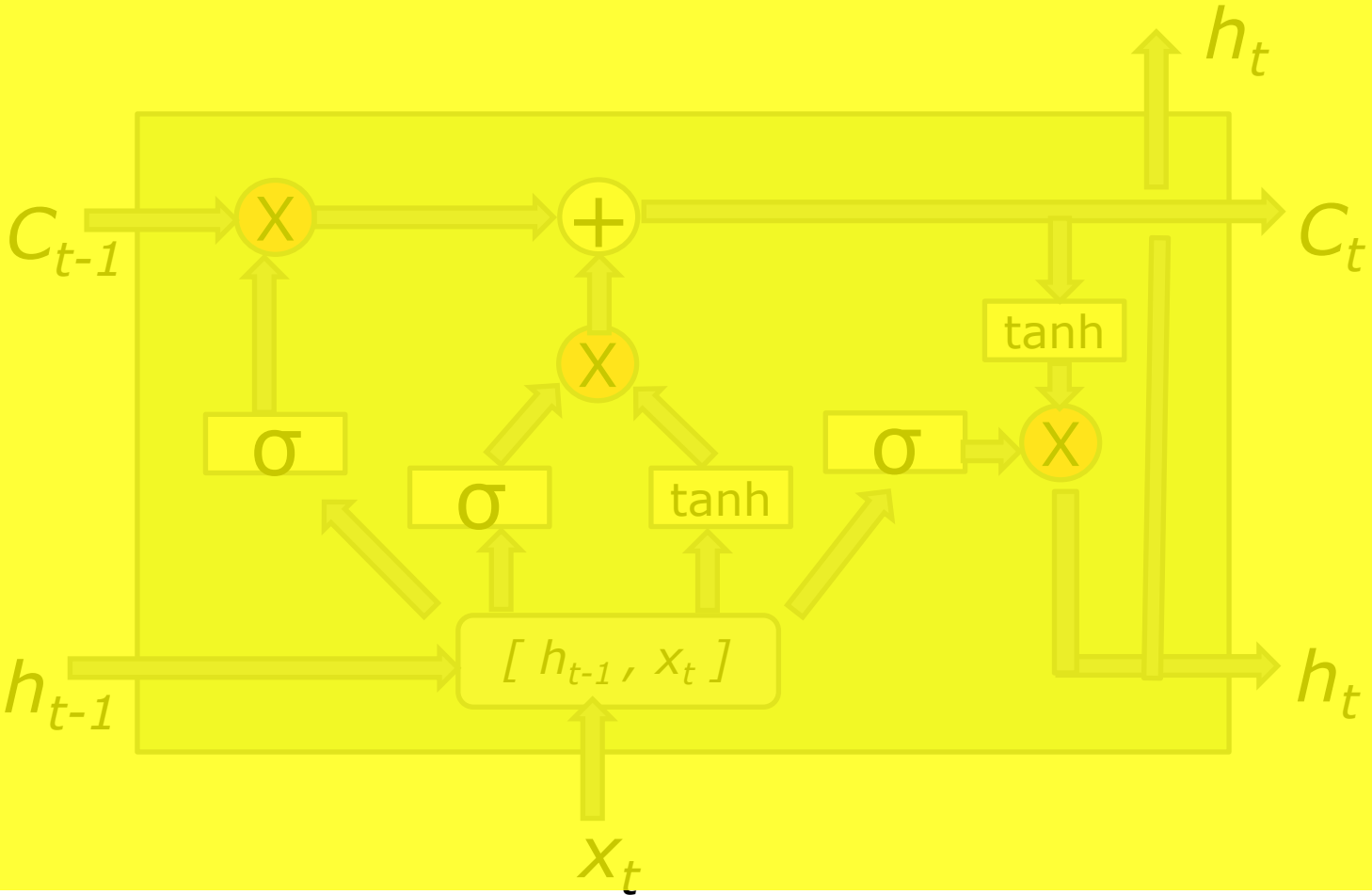
- **Input Gateのコントロール:**

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

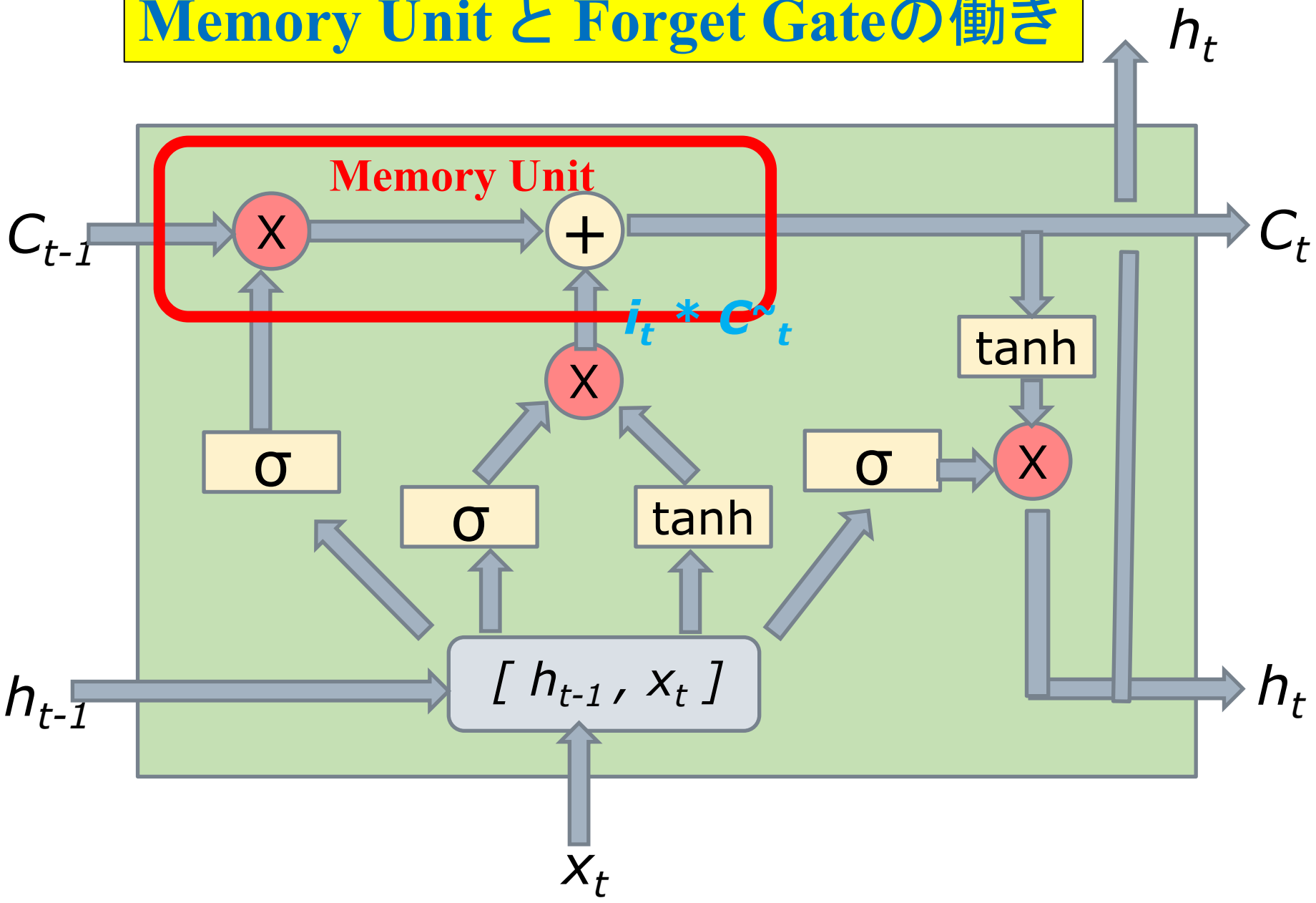
- **Memory Unitへの入力1**

$$Memory_1 = i_t * \tilde{C}_t$$

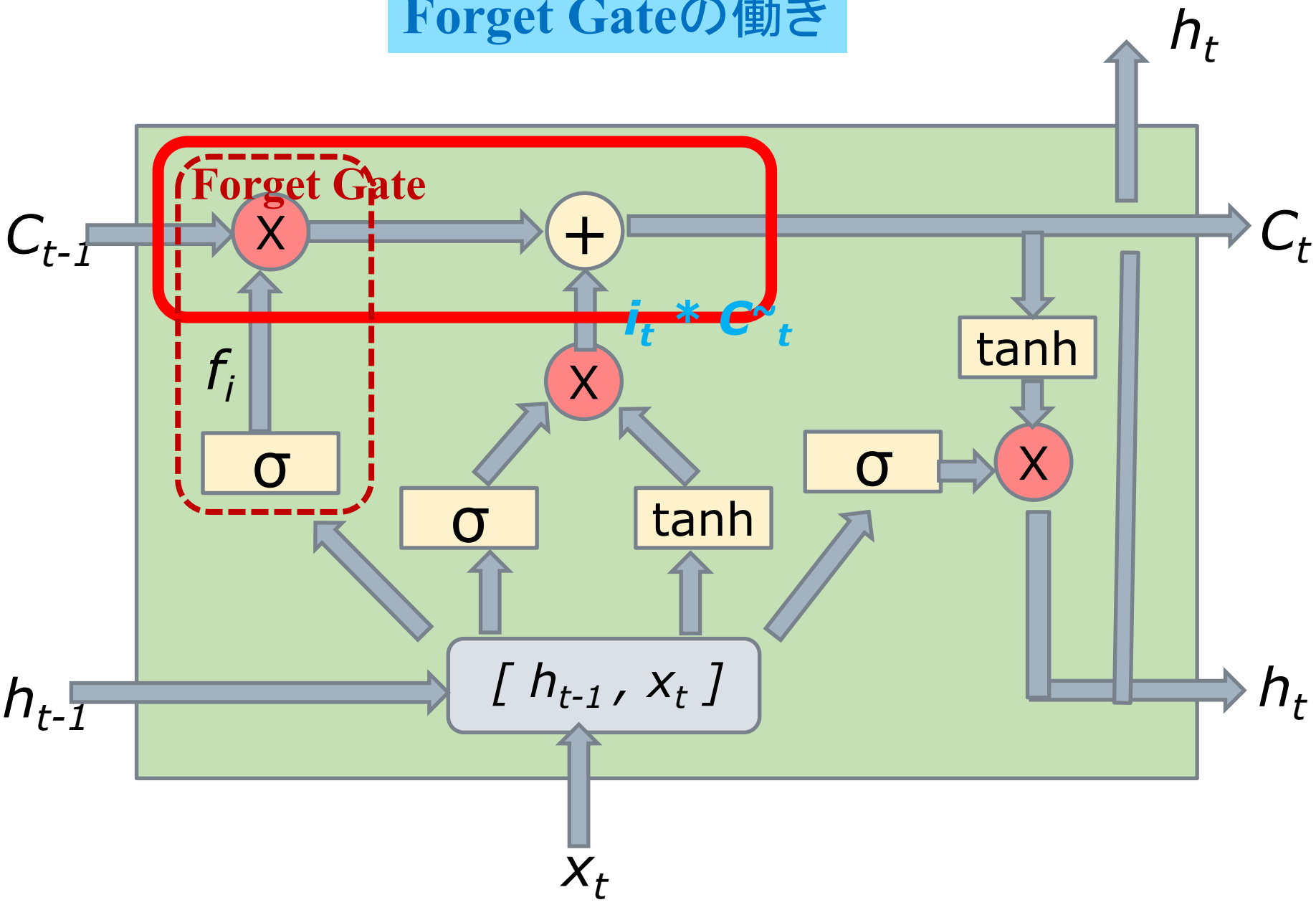
Memory Unit と Forget Gateの働き

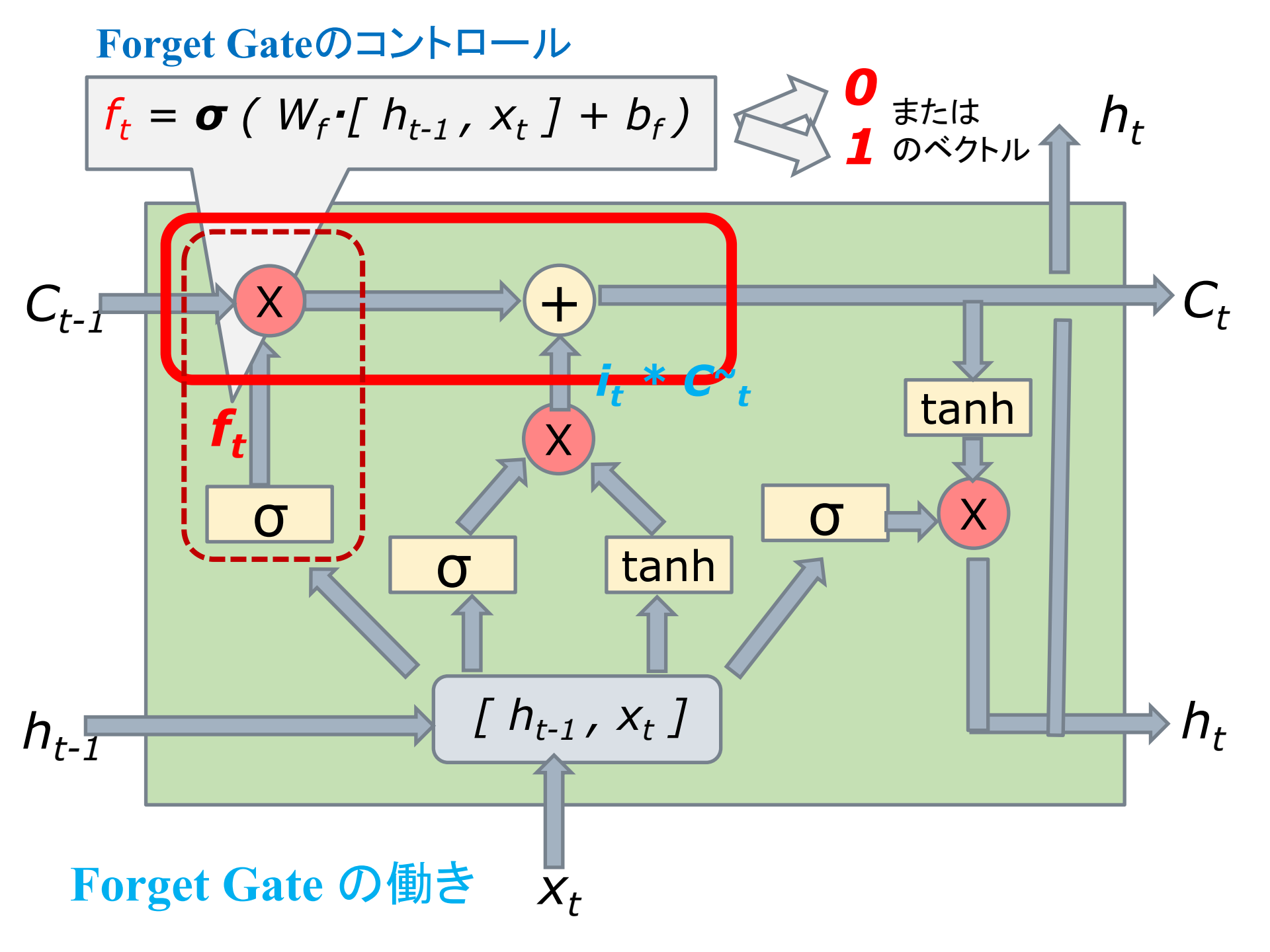


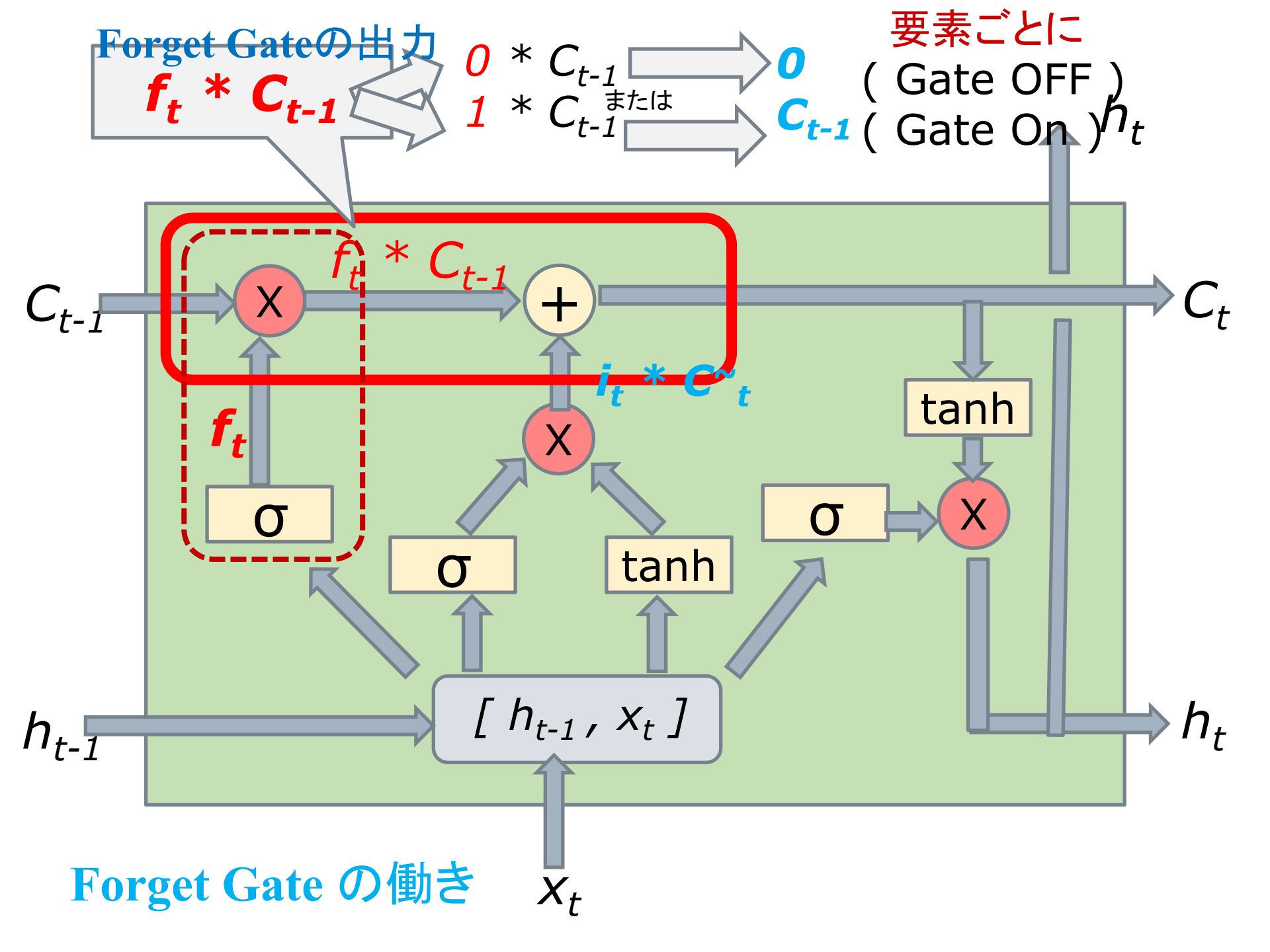
Memory Unit と Forget Gateの働き



Forget Gateの働き

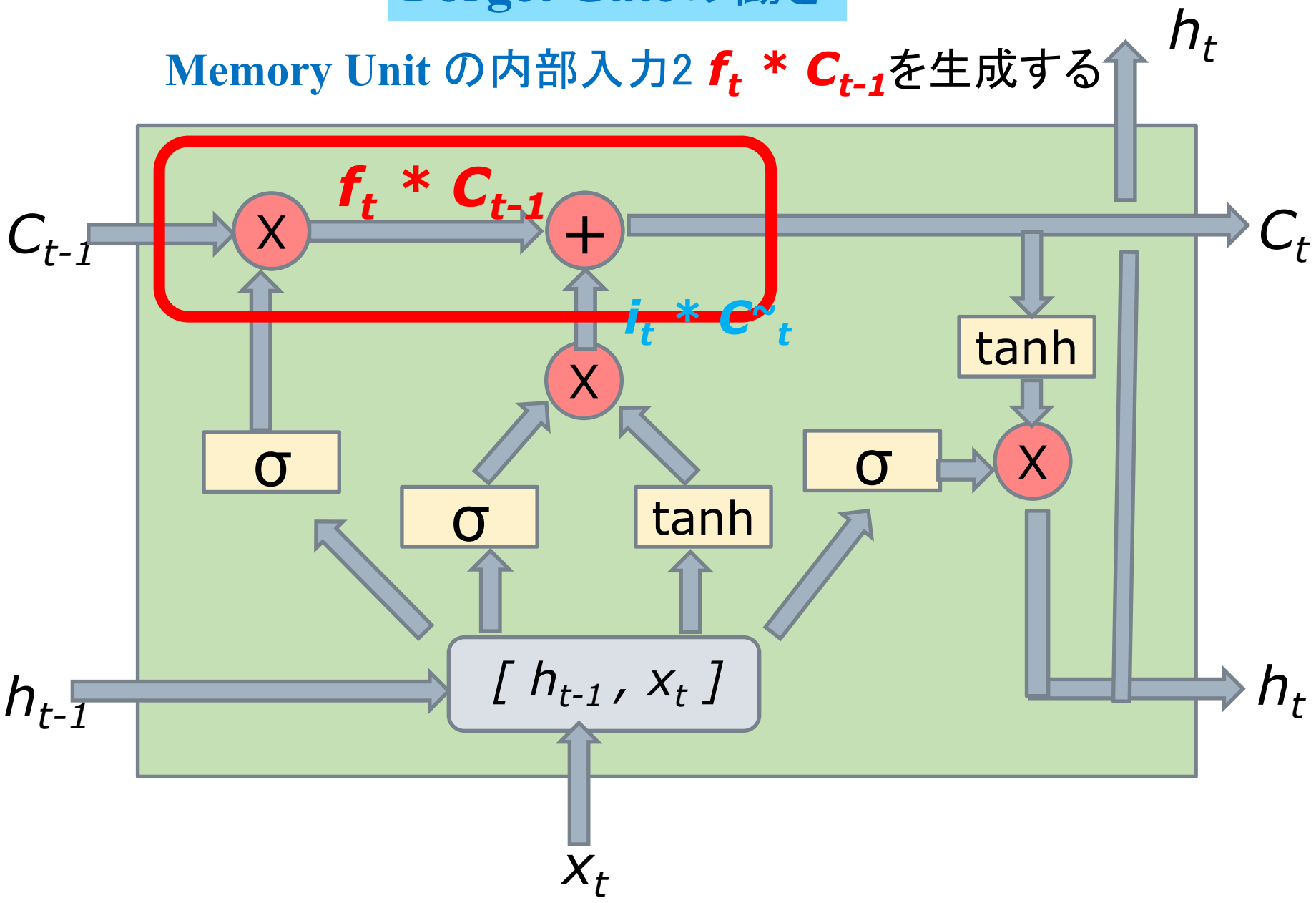



Forget Gateのコントロール
$f_t = \sigma (W_f \cdot [h_{t-1} , x_t] + b_f)$
0
1
または
のベクトル
h_t
C_{t-1}
C_t
X
+
$i_t * \tilde{C}_t$
f_t
X
tanh
σ
σ
tanh
σ
X
$[h_{t-1} , x_t]$
h_{t-1}
h_t
x_t
Forget Gate の働き


Forget Gateの出力
$f_t * C_{t-1}$
$0 * C_{t-1}$
$1 * C_{t-1}$
または
0
C_{t-1}
要素ごとに
(Gate OFF)
(Gate On)
h_t
$f_t * C_{t-1}$
$i_t * \tilde{C}_t$
C_{t-1}
C_t
f_t
σ
σ
tanh
σ
tanh
$[h_{t-1}, x_t]$
h_{t-1}
h_t
x_t
Forget Gate の働き

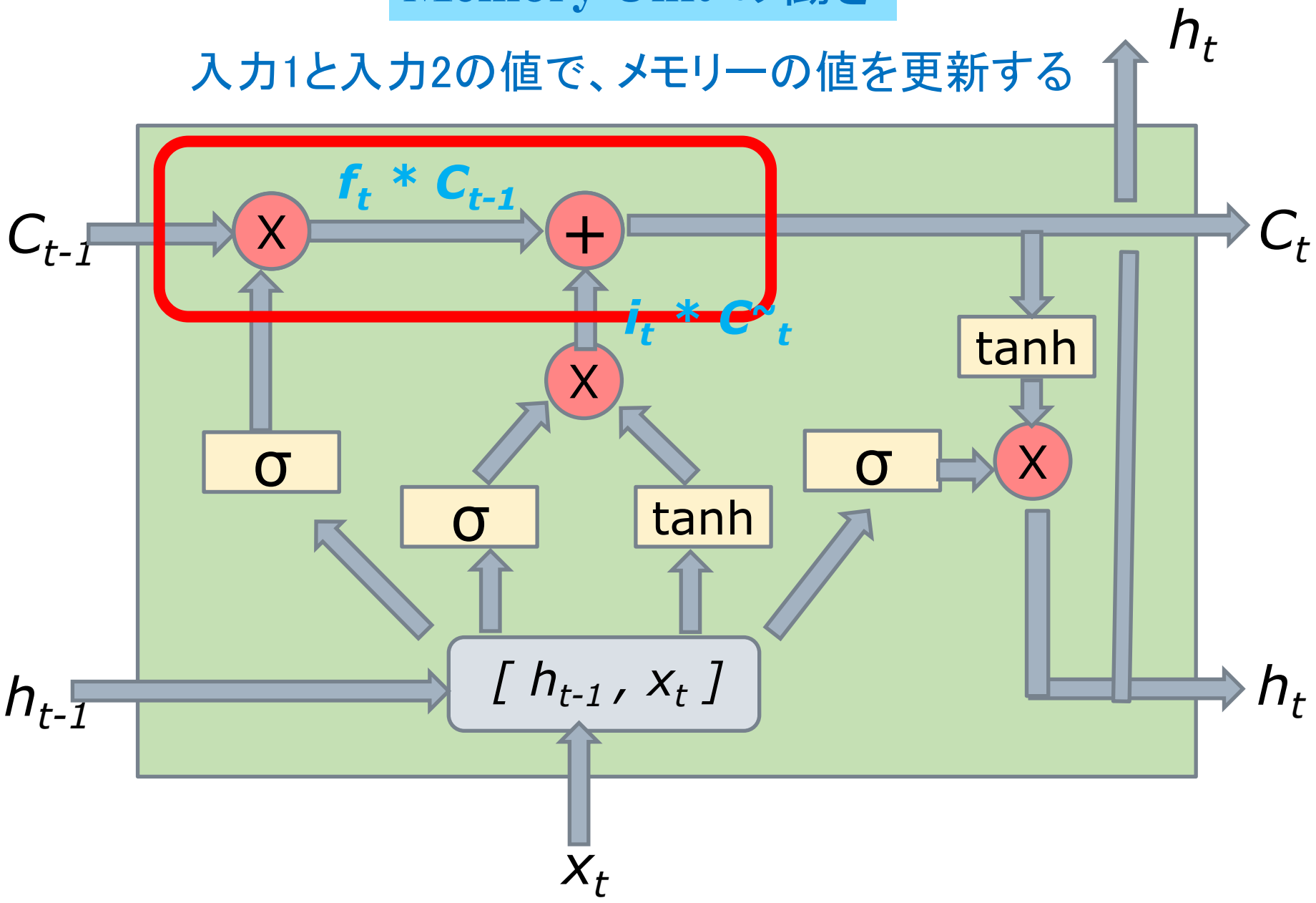
Forget Gateの働き

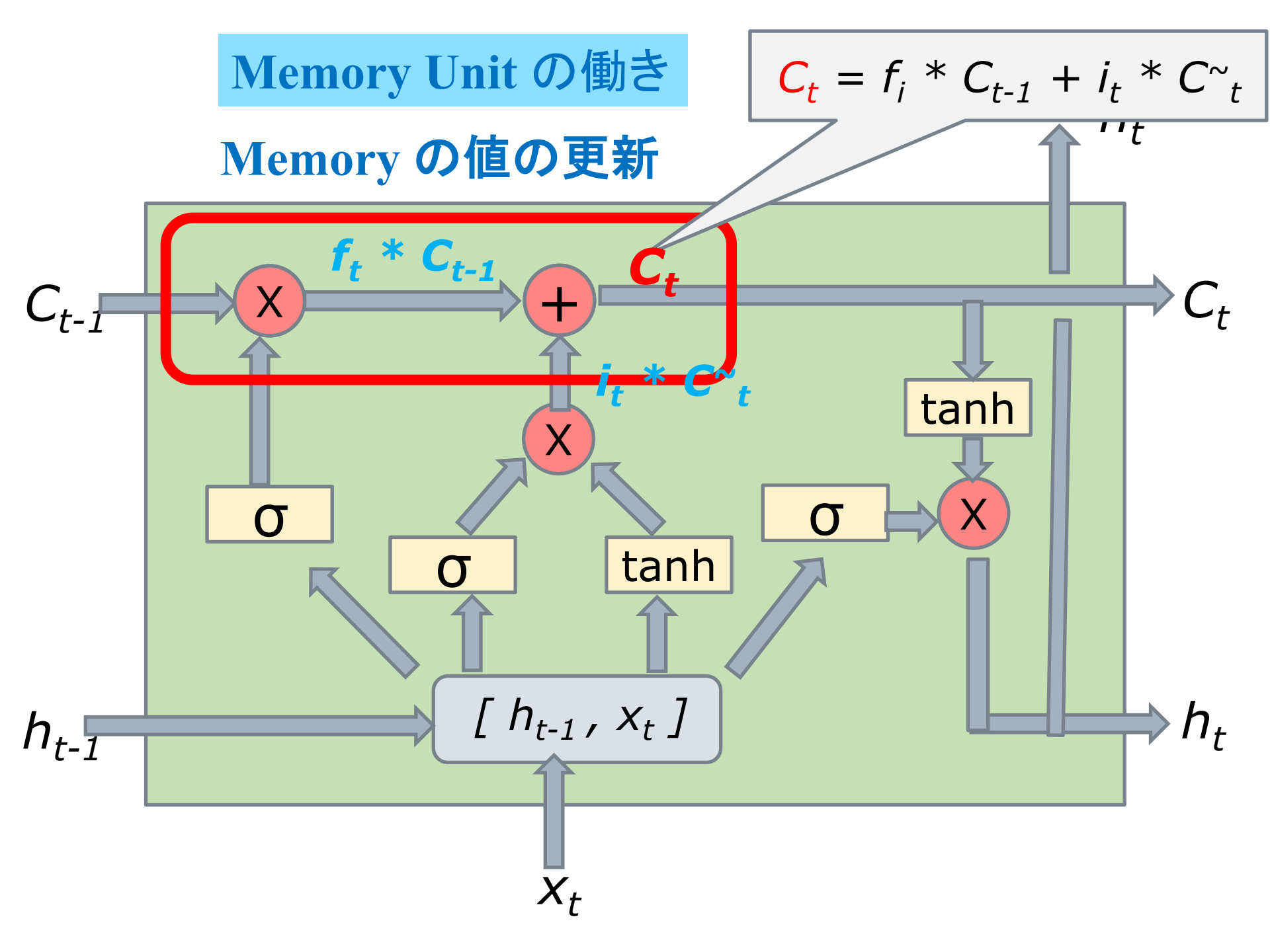
Memory Unit の内部入力2 $f_t * C_{t-1}$を生成する



Memory Unit の働き

入力1と入力2の値で、メモリーの値を更新する




Memory Unit の働き
Memory の値の更新
$C_t = f_i * C_{t-1} + i_t * C\tilde{}_t$
$f_t * C_{t-1}$
C_t
$i_t * C\tilde{}_t$
C_{t-1}
C_t
X
+
X
tanh
σ
σ
tanh
σ
X
$[\ h_{t-1}, x_t\ ]$
h_{t-1}
h_t
x_t

Memory Unit と Forget Gateの働きを式で表す

- Forget Gateのコントロール:

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

- Memory Unitへの入力1 （Input Gateから）:

$$Memory_1 = i_t * C\tilde{}_t$$

- Memory Unit への入力2 （Forget Gateから）:

$$Memory_2 = f_t * C_{t-1}$$

- Memory の値の更新:

$$C_t = Memory_1 + Memory_2$$
$$= f_i * C_{t-1} + i_t * C\tilde{}_t$$

Memory Cellの式 $C_t = f_i * C_{t-1} + i_t * \tilde{C}_t$ の意味

- Forget Gateが閉じられていれば(すなわち $f_i = 0$ の時)、$C_t = i_t * \tilde{C}_t$ となる。
  - Input Gateが閉じられていれば(すなわち $i_i = 0$ の時)、Memory Cellの値は、ゼロ・クリアされ、
  - Input Gateが開いていれば(すなわち $i_i = 1$ の時)、Memory Cellの値は、Input Unitが生成した値 $\tilde{C}_t$ に更新される。
- Forget Gateが開いていれば(すなわち $f_i = 1$ の時)、$C_t = C_{t-1} + i_t * \tilde{C}_t$ となる。
  - Input Gateが閉じられていれば(すなわち $i_i = 0$ の時)、Memory Cellの値は、$C_t = C_{t-1}$ となり、メモリーの値は、保持される。
  - Input Gateが開いていれば(すなわち $i_i = 1$ の時)、Memory Cellの値は、$C_t = C_{t-1} + \tilde{C}_t$ に更新される。

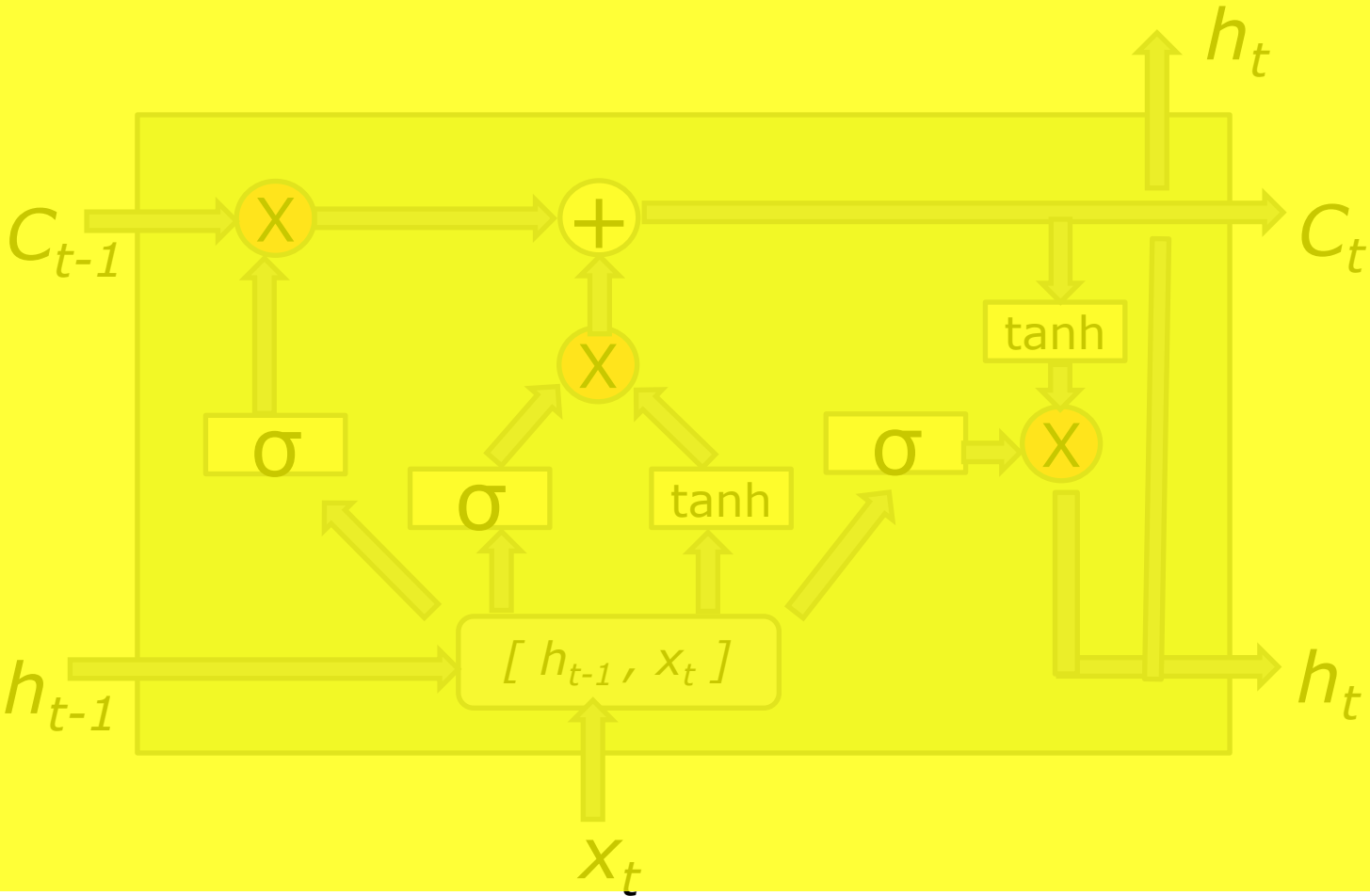
2つのGateとMemory Cellの動作

先の記述をまとめると、次のようになる。

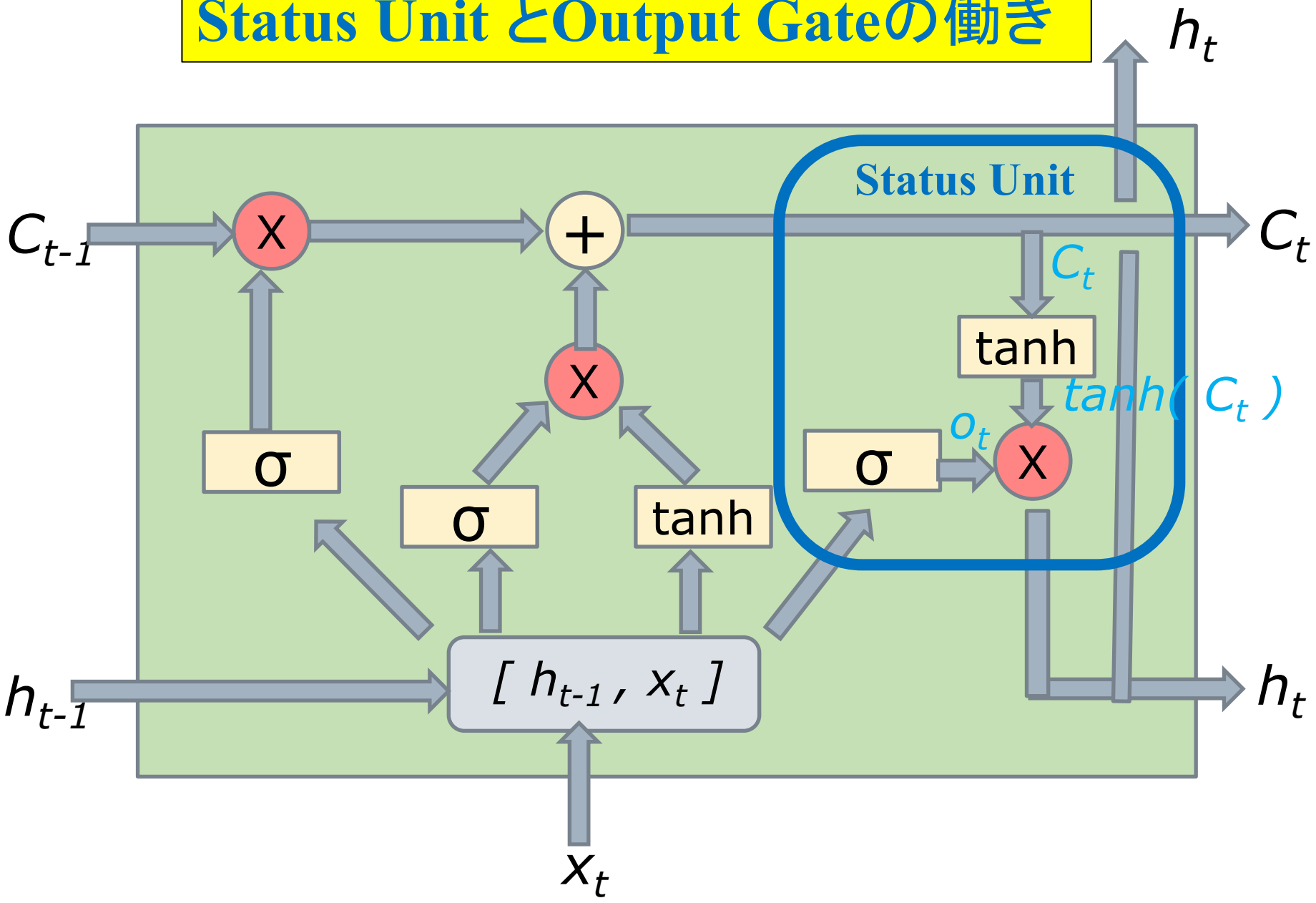
	Forget Gate OFF	Forget Gate ON
Input Gate OFF	メモリークリア	メモリー保持
Input Gate ON	メモリー更新	メモリー更新（混合）

ただ、Gateのコントローラの値は、完全に、0 または 1 の値を取るとは限らないので（Sigmoid関数の出力だから）、この表は、基本的な傾向を表していると考えた方が、いいかもしれない。

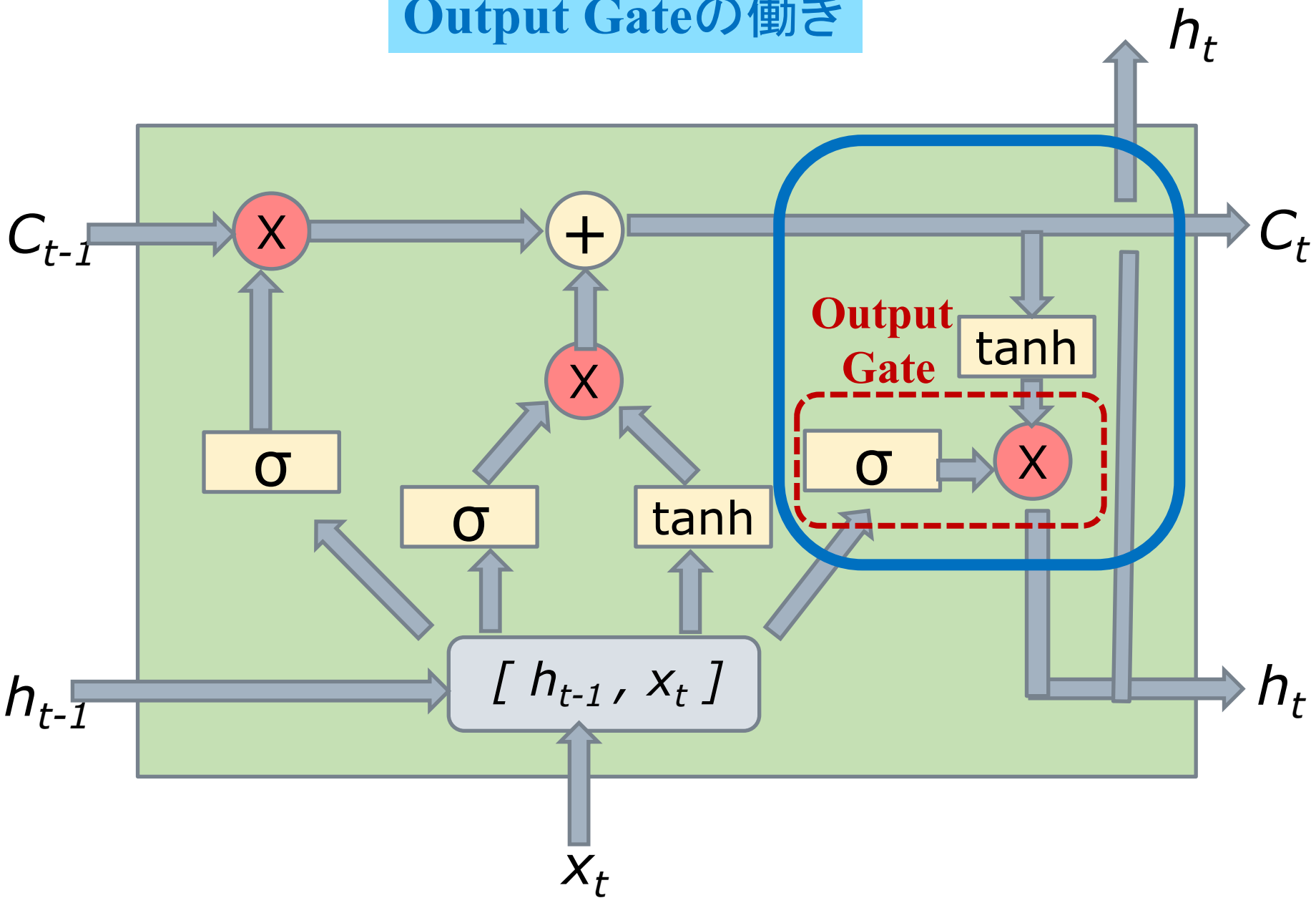
Status Unit とOutput Gateの働き

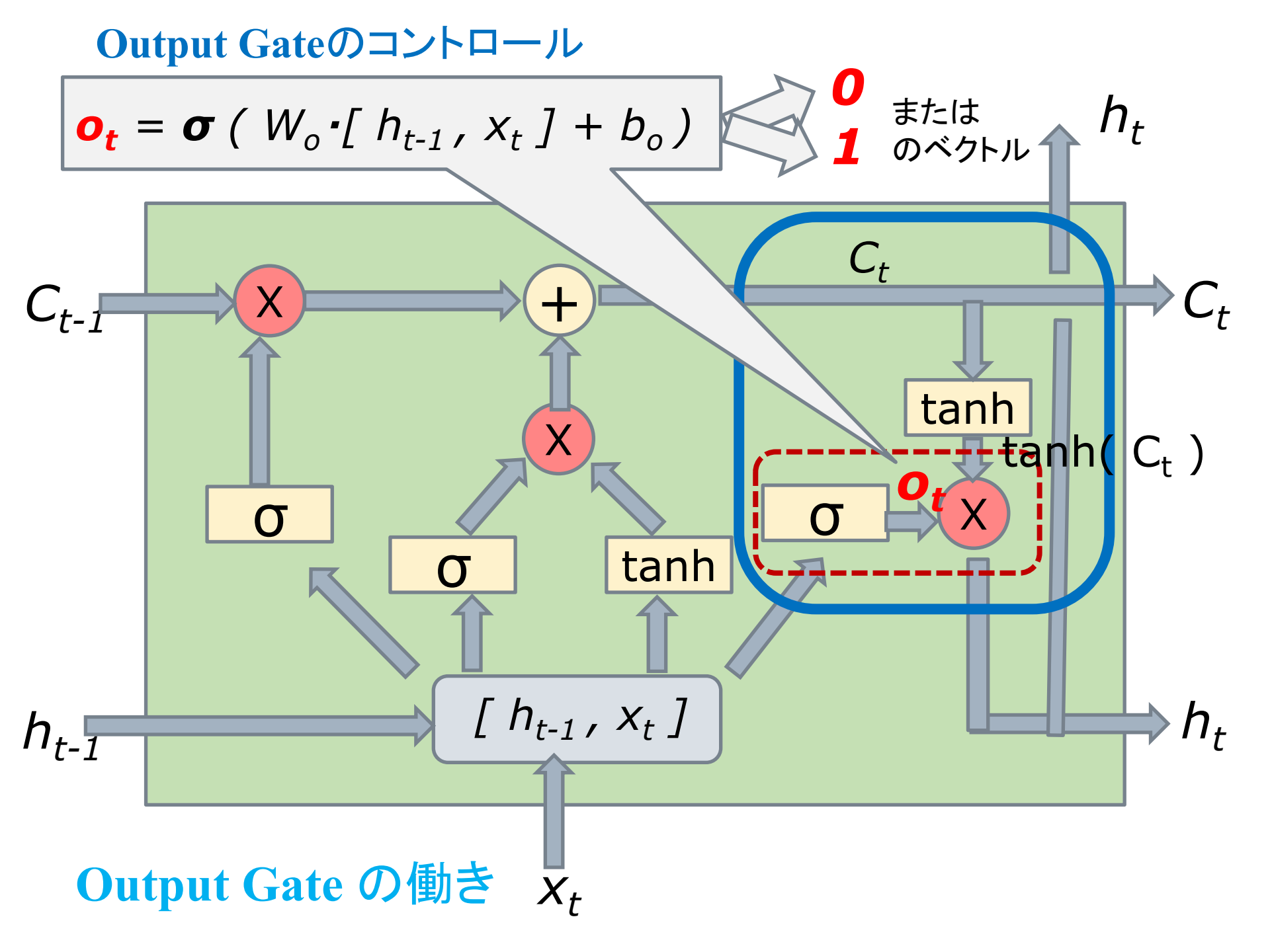


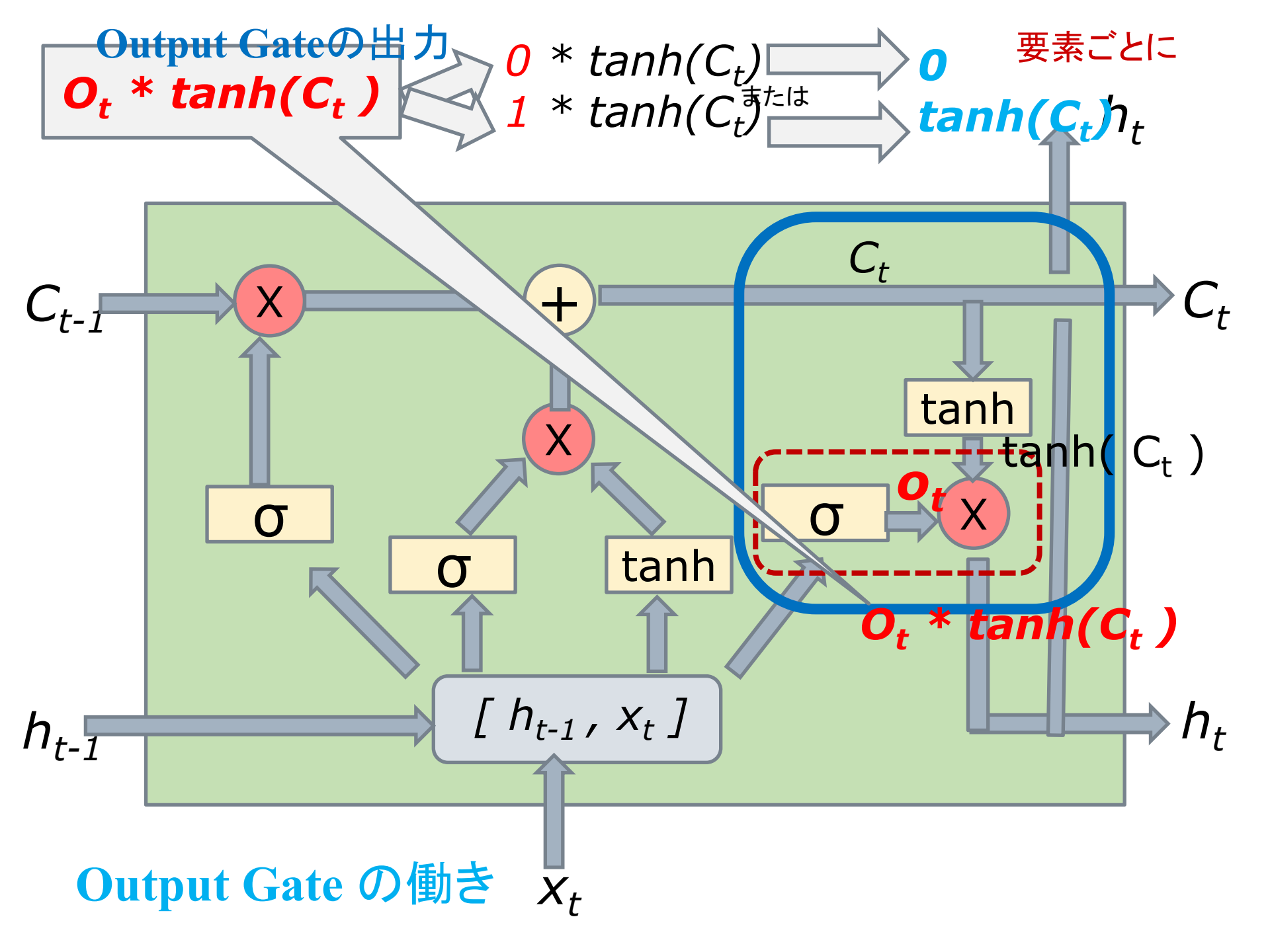
Status Unit とOutput Gateの働き



Output Gateの働き

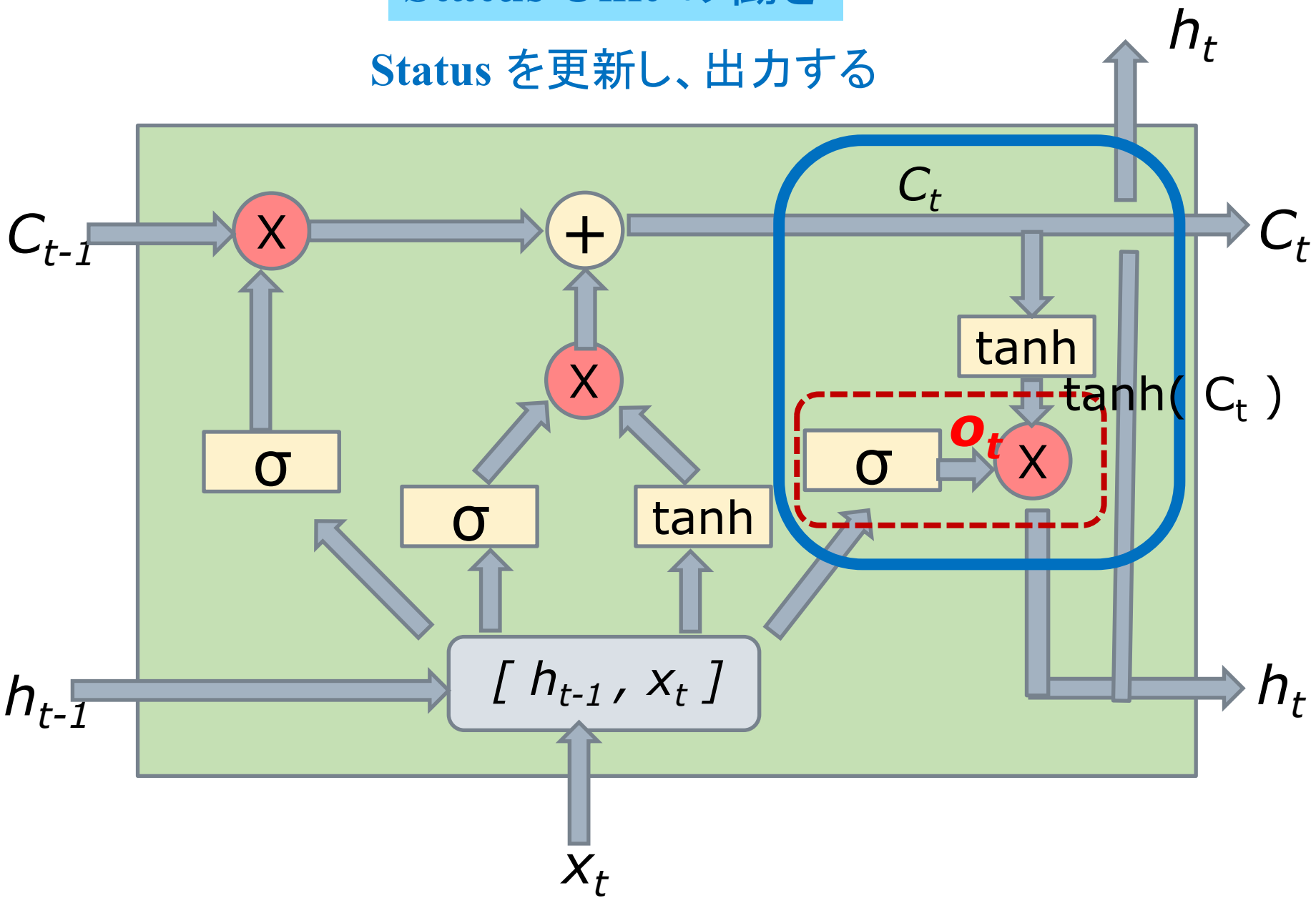


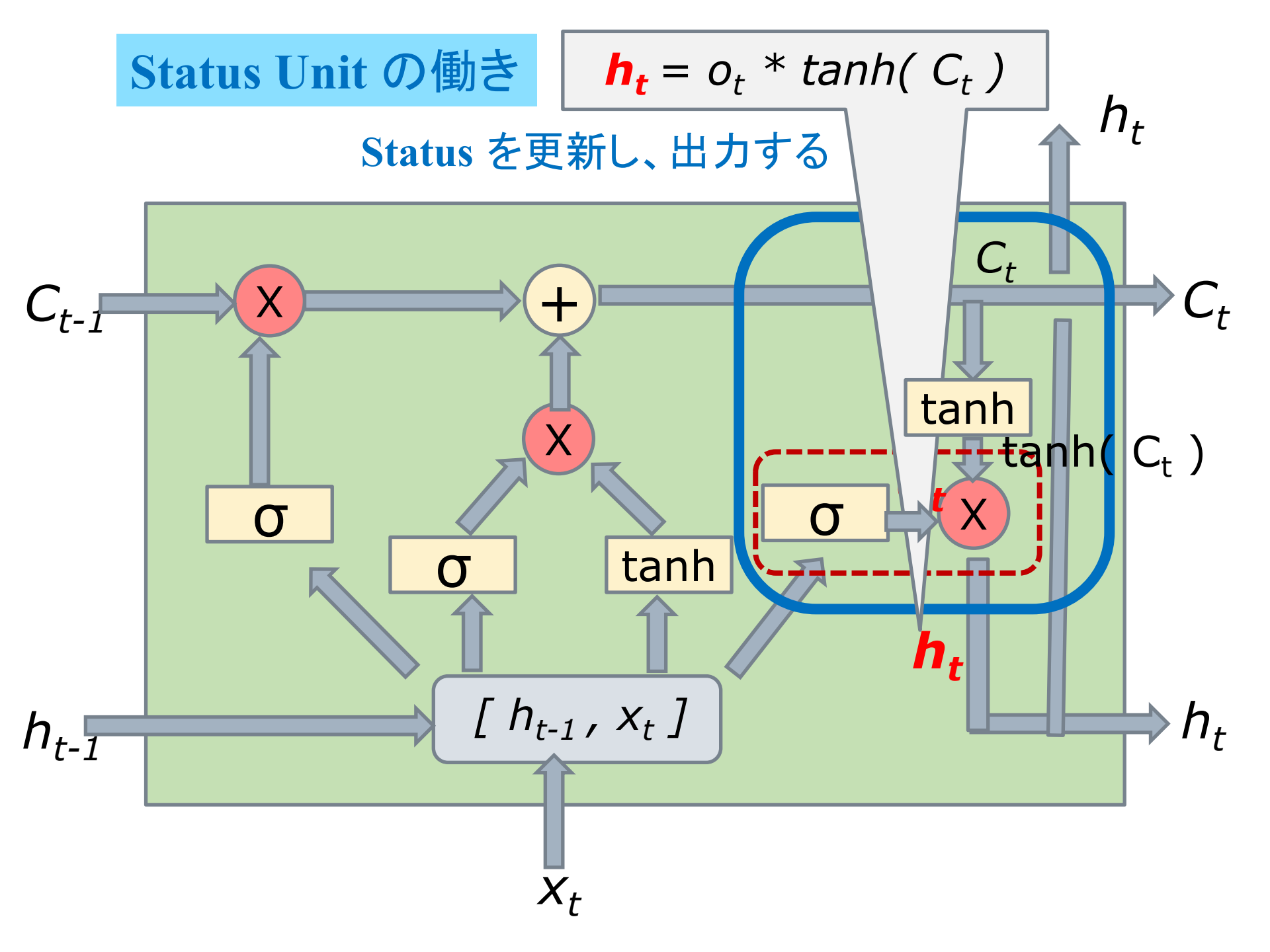

Output Gateのコントロール
$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
0
1
またはのベクトル
h_t
C_t
C_{t-1}
X
+
X
σ
σ
tanh
tanh
tanh(C_t)
o_t
σ
X
$[h_{t-1}, x_t]$
h_{t-1}
h_t
x_t
Output Gate の働き


Output Gateの出力
$O_t * \tanh(C_t)$
$0 * \tanh(C_t)$
$1 * \tanh(C_t)$
または
0
$\tanh(C_t)$
h_t
要素ごとに
C_t
C_{t-1}
X
+
C_t
tanh
$\tanh(C_t)$
X
O_t
σ
σ
tanh
σ
X
$O_t * \tanh(C_t)$
$[h_{t-1}, x_t]$
h_{t-1}
h_t
x_t
Output Gate の働き

Status Unit の働き

Status を更新し、出力する




Status Unit の働き
$h_t = o_t * tanh(C_t)$
Status を更新し、出力する
h_t
C_t
C_{t-1}
X
+
C_t
tanh
tanh(C_t)
X
σ
σ
tanh
σ
X
h_t
h_{t-1}
[h_{t-1} , x_t]
h_t
x_t

Status Unit とOutput Gateの働きを式で表す

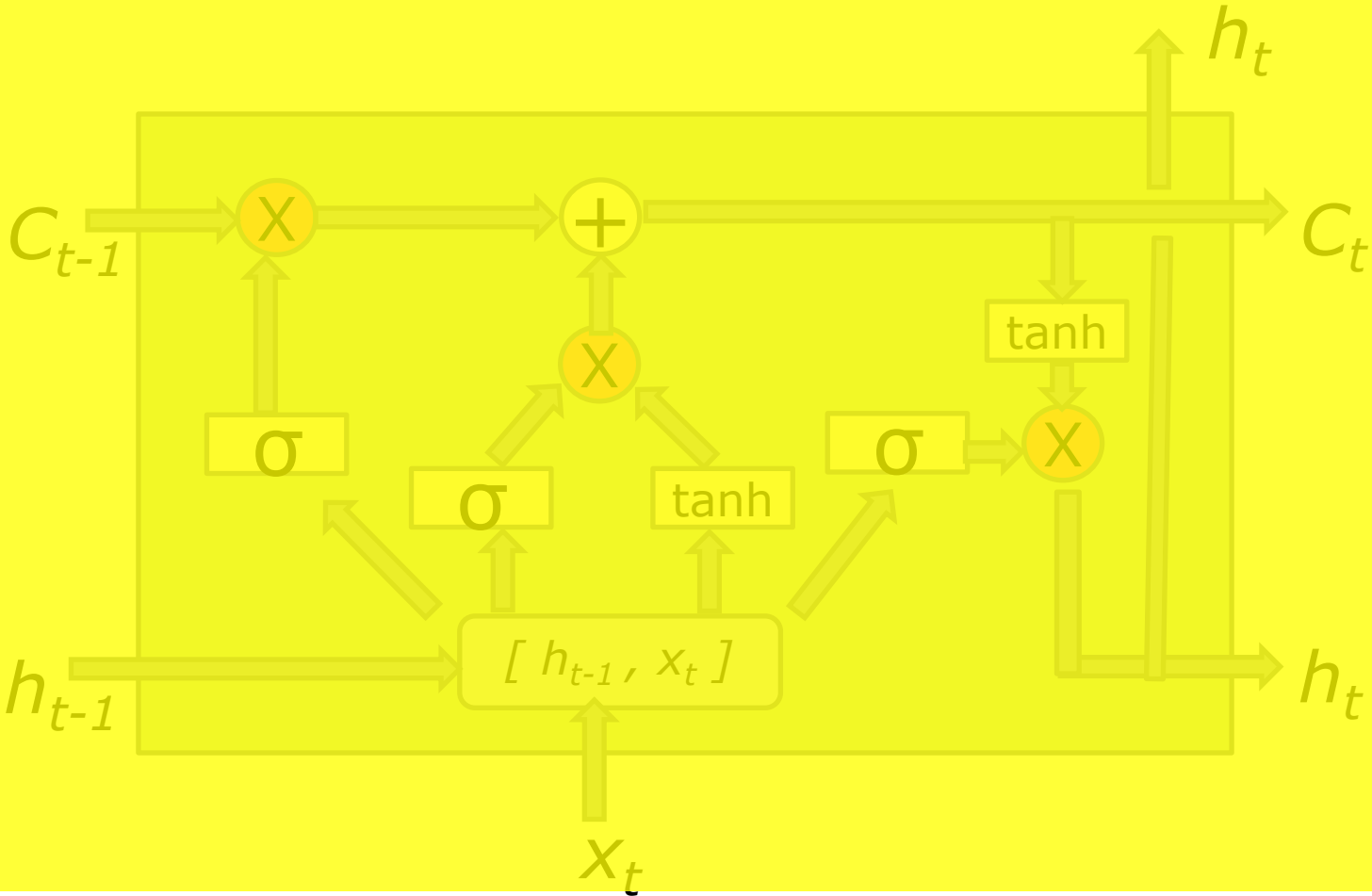
- Output Gateのコントロール:

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

- Output Gateの出力:

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

LSTM の働きを式で表す（まとめ）



3つのGate

- **Input Gateのコントロール:**
  $i_t = \sigma (W_i \cdot [h_{t-1} , x_t] + b_i)$
- **Forget Gateのコントロール:**
  $f_t = \sigma (W_f \cdot [h_{t-1} , x_t] + b_f)$
- **Output Gateのコントロール:**
  $o_t = \sigma (W_o \cdot [h_{t-1} , x_t] + b_o)$

3つのユニット

Input Unit

入力: $[\ h_{t-1}\ ,\ x_t\ ]$

出力: $\boldsymbol{i_t} * \tilde{C}_t$

(ただし、$\tilde{C}_t = tanh\ (\ W_c \cdot [\ h_{t-1}\ ,\ x_t\ ]\ +\ b_c\ )$)

Memory Unit

入力1: $\boldsymbol{i_t} * \tilde{C}_t$

入力2: $\boldsymbol{f_t} * C_{t-1}$

出力: $\boldsymbol{i_t} * \tilde{C}_t + \boldsymbol{f_t} * C_{t-1}$

Status Unit

入力: $tanh(\ C_t\ )$

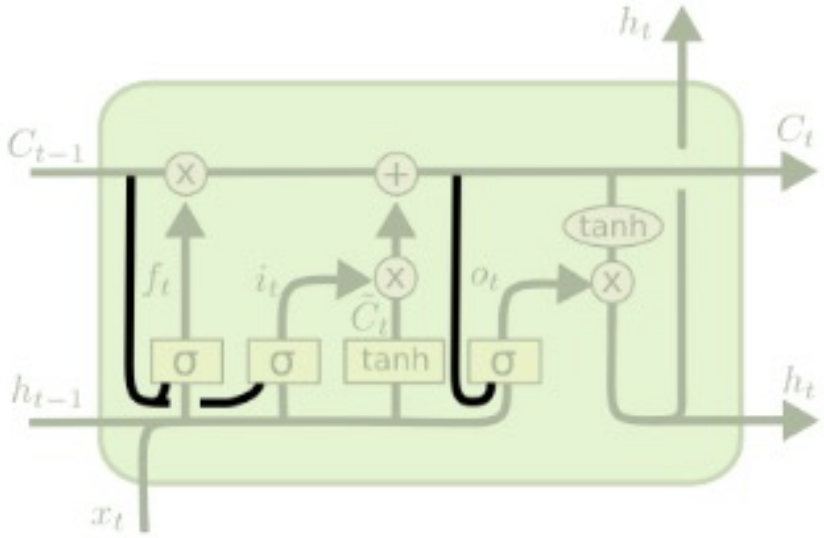
出力: $h_t = \boldsymbol{o_t} * tanh(\ C_t\ )$

LSTM のバリアント

LSTMには、いくつかのバリアントがある。ここは、先のColahのblogに依拠した。

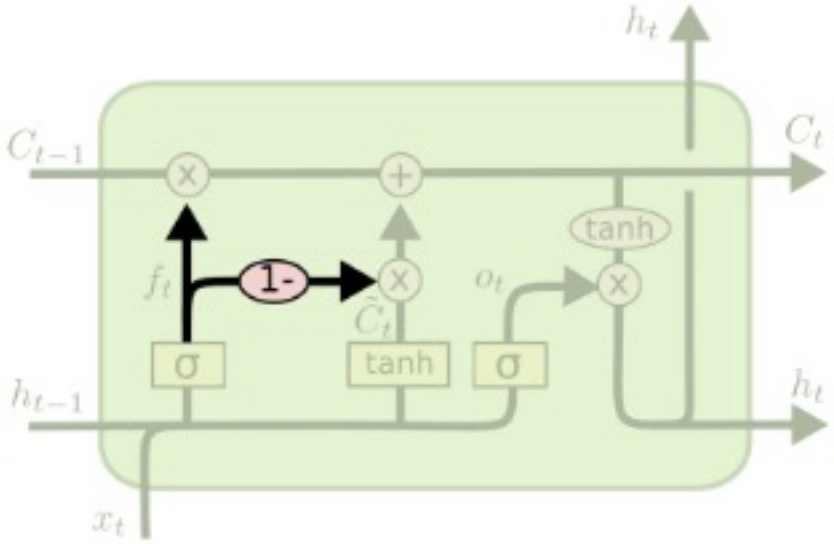
Peephole

Gateが、メモリーの状態を利用できるようにしたもの



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

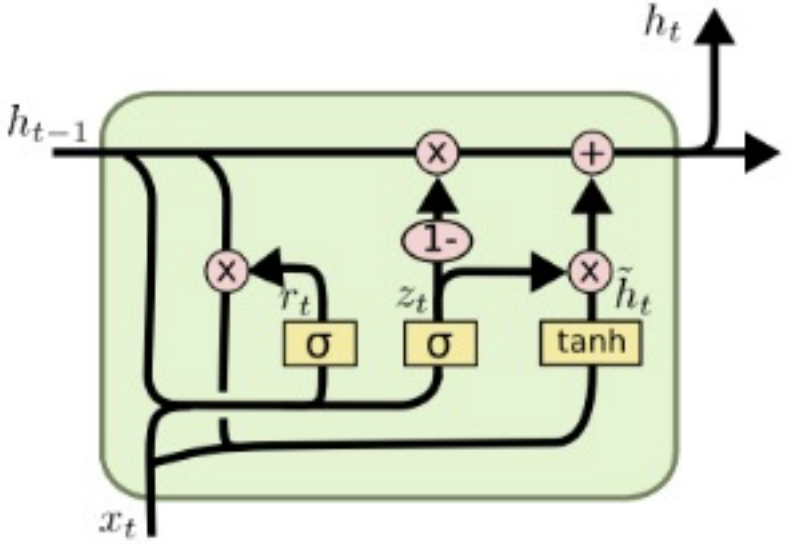
Forget Gate と Input Gate の結合



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

GRU(Gated Recurrent Unit)

Forget Gate と Input Gateを一つにまとめ（Update Gate）
「状態」と「記憶」をマージする。その他の改良を加える。
元のLSTMより、構造が単純になる。最近、よく使われている。



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

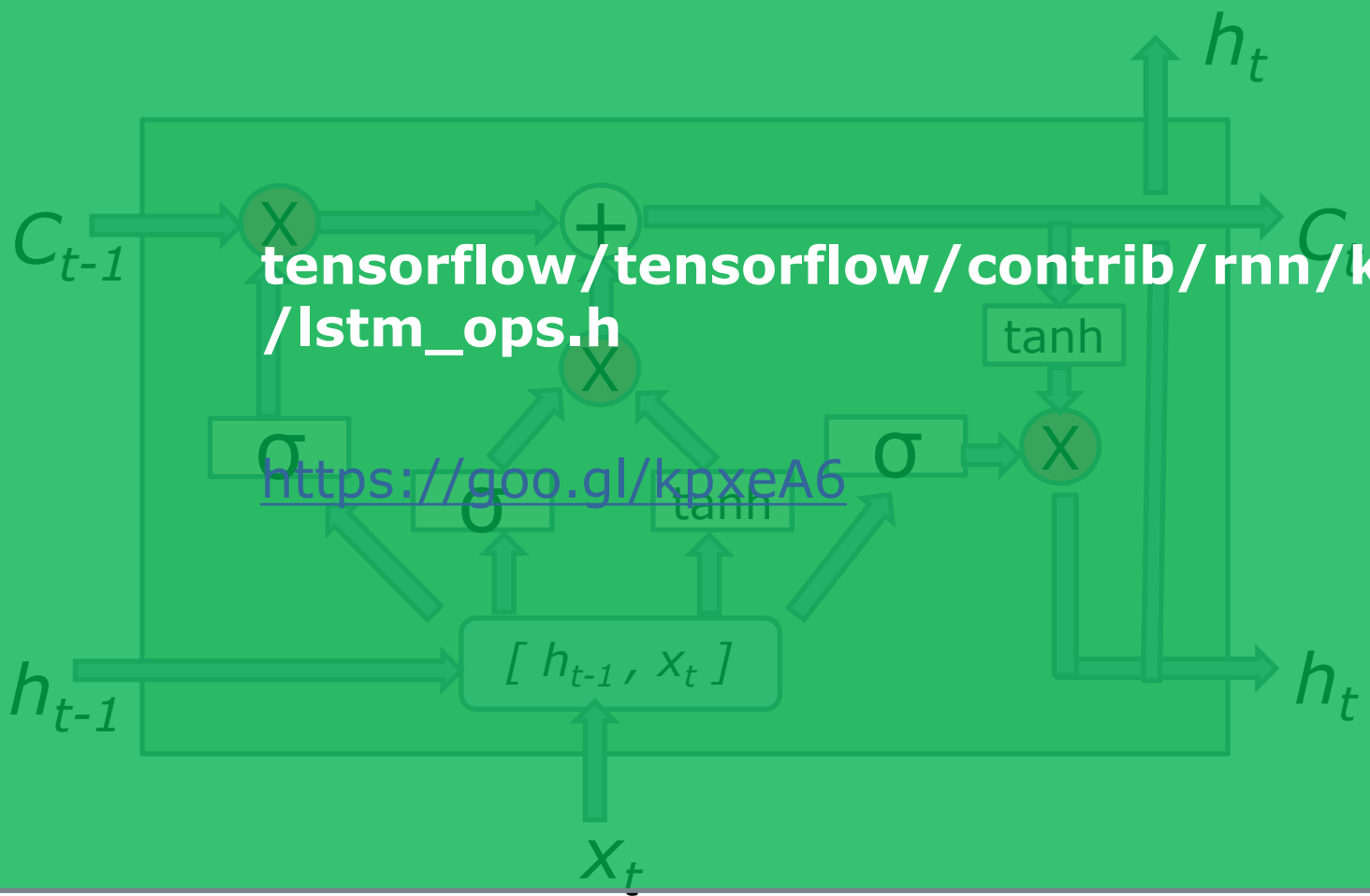
Appendix

参考資料

Appendix
Agenda

- TensorFlowでのLSTMの定義
- TensorFlowでのLSTMの定義(Python)
- CNTKでのLSTMの定義
- MXNetでのLSTMの定義
- TensorFlow FoldでのLSTMの記述

TensorFlowでのLSTMの定義



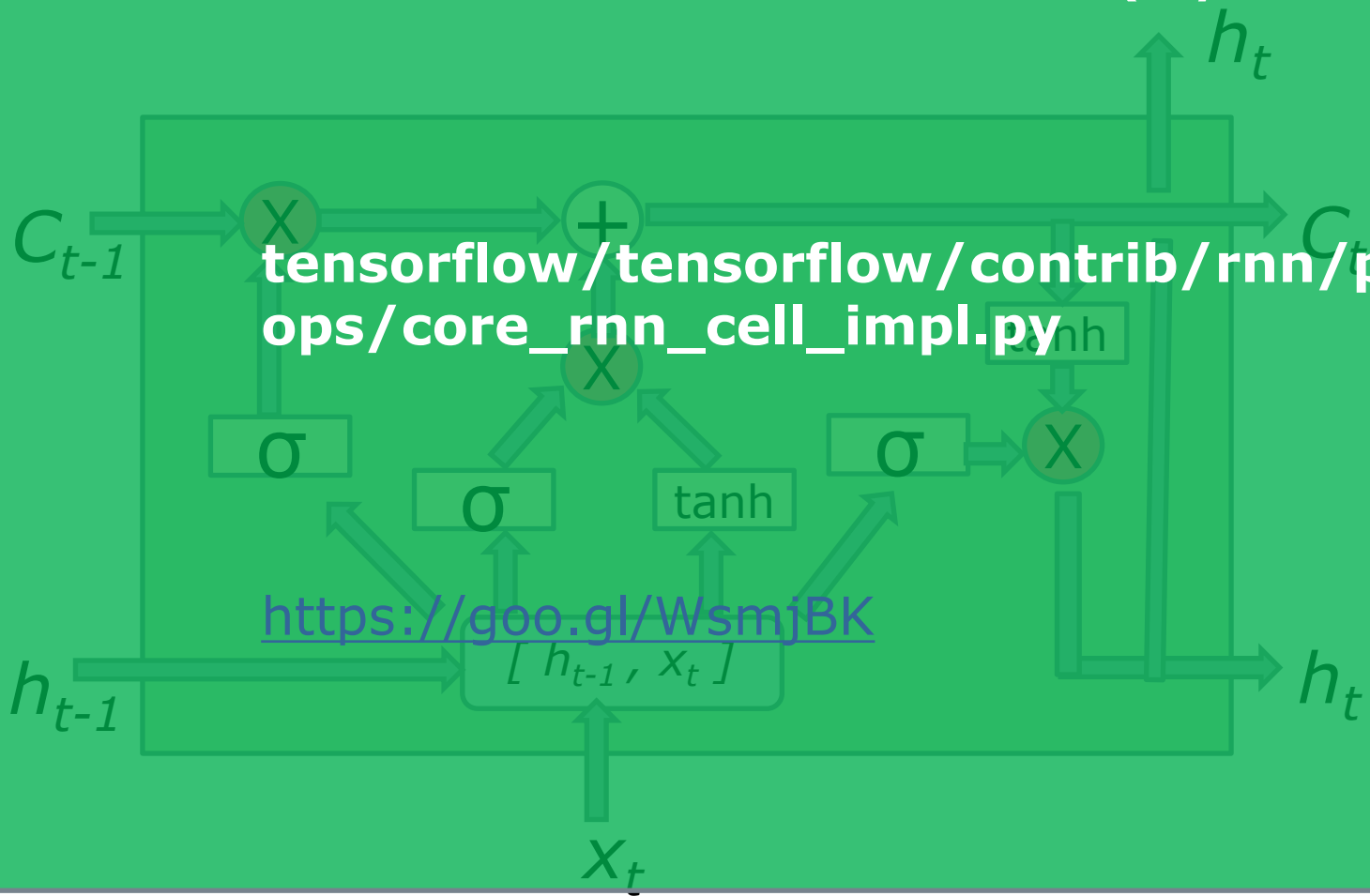
tensorflow/tensorflow/contrib/rnn/kernels/lstm_ops.h

https://goo.gl/kpxeA6

```
150     void operator()(
151         OpKernelContext* ctx, const Device& d, const T forget_bias,
152         const T cell_clip, bool use_peephole, typename TTypes<T>::ConstMatrix x,
153         typename TTypes<T>::ConstMatrix cs_prev,
154         typename TTypes<T>::ConstMatrix h_prev, typename TTypes<T>::ConstMatrix w,
155         typename TTypes<T>::ConstVec wci, typename TTypes<T>::ConstVec wcf,
156         typename TTypes<T>::ConstVec wco, typename TTypes<T>::ConstVec b,
157         typename TTypes<T>::Matrix xh, typename TTypes<T>::Matrix i,
158         typename TTypes<T>::Matrix cs, typename TTypes<T>::Matrix f,
159         typename TTypes<T>::Matrix o, typename TTypes<T>::Matrix ci,
160         typename TTypes<T>::Matrix co, typename TTypes<T>::Matrix icfo,
161         typename TTypes<T>::Matrix h) {
162       // Concat xh = [x, h].
163       xh.slice(xh_x_offsets(), xh_x_extents()).device(d) = x;
164       xh.slice(xh_h_offsets(), xh_h_extents()).device(d) = h_prev;
165
166       // states1 = xh * w + b
167       typename TTypes<T>::ConstMatrix const_xh(xh.data(), xh.dimensions());
168       TensorBlasGemm<Device, T, USE_CUBLAS>::compute(ctx, d, false, false, T(1),
169                                                      const_xh, w, T(0), icfo);
170       Eigen::array<Eigen::DenseIndex, 2> b_shape({1, b.dimensions()[0]});
171       Eigen::array<Eigen::DenseIndex, 2> broadcast_shape({batch_size_, 1});
172       icfo.device(d) += b.reshape(b_shape).broadcast(broadcast_shape);
173
```

```
174     Eigen::array<Eigen::DenseIndex, 2> p_shape({1, cell_size_});
175     Eigen::array<Eigen::DenseIndex, 2> p_broadcast_shape({batch_size_, 1});
176
177     // Input gate.
178     if (use_peephole) {
179       auto i_peep = cs_prev * wci.reshape(p_shape).broadcast(p_broadcast_shape);
180       i.device(d) =
181           (icfo.slice(icfo_i_offsets(), cell_extents()) + i_peep).sigmoid();
182     } else {
183       i.device(d) = icfo.slice(icfo_i_offsets(), cell_extents()).sigmoid();
184     }
185
186     // Cell input.
187     ci.device(d) = icfo.slice(icfo_c_offsets(), cell_extents()).tanh();
188
189     // Forget gate (w/ bias).
190     if (use_peephole) {
191       auto f_peep = cs_prev * wcf.reshape(p_shape).broadcast(p_broadcast_shape);
192       f.device(d) = (icfo.slice(icfo_f_offsets(), cell_extents()) +
193                      f.constant(forget_bias) + f_peep)
194                         .sigmoid();
195     } else {
196       f.device(d) = (icfo.slice(icfo_f_offsets(), cell_extents()) +
197                      f.constant(forget_bias))
198                         .sigmoid();
199     }
```

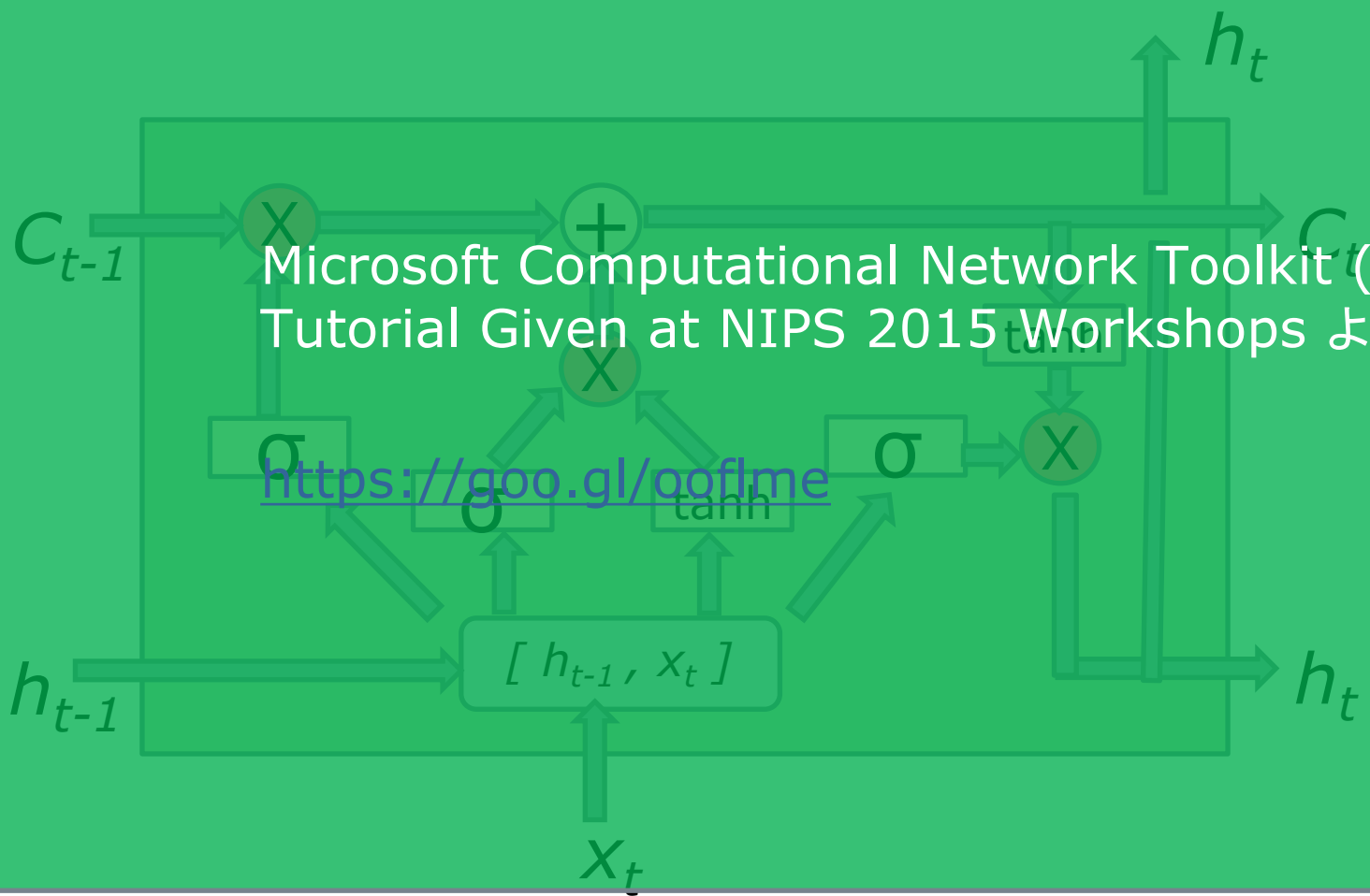
TensorFlowでのLSTMの定義(Python)



tensorflow/tensorflow/contrib/rnn/python/ops/core_rnn_cell_impl.py

https://goo.gl/WsmjBK

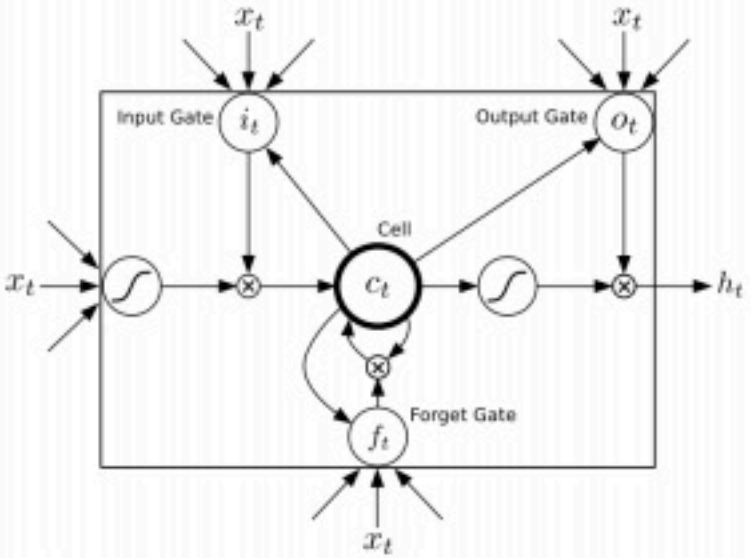
CNTKでのLSTMの定義



Microsoft Computational Network Toolkit (CNTK) A Tutorial Given at NIPS 2015 Workshops より

https://goo.gl/oofIme

Network Definition with NDL (One Way)



これは、CNTKのサンプルだが、グラフだけでは、全容を把握するのは難しい。

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}^{(xi)}\mathbf{x}_t + \mathbf{W}^{(hi)}\mathbf{h}_{t-1} + \mathbf{W}^{(ci)}\mathbf{c}_{t-1} + \mathbf{b}^{(i)}\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}^{(xf)}\mathbf{x}_t + \mathbf{W}^{(hf)}\mathbf{h}_{t-1} + \mathbf{W}^{(cf)}\mathbf{c}_{t-1} + \mathbf{b}^{(f)}\right) \\
\mathbf{c}_t &= \mathbf{f}_t \bullet \mathbf{c}_{t-1} + \mathbf{i}_t \bullet \tanh\left(\mathbf{W}^{(xc)}\mathbf{x}_t + \mathbf{W}^{(hc)}\mathbf{h}_{t-1} + \mathbf{b}^{(c)}\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}^{(xo)}\mathbf{x}_t + \mathbf{W}^{(ho)}\mathbf{h}_{t-1} + \mathbf{W}^{(co)}\mathbf{c}_t + \mathbf{b}^{(o)}\right) \\
\mathbf{h}_t &= \mathbf{o}_t \bullet \tanh\left(\mathbf{c}_t\right),
\end{aligned}
$$

Network Definition with NDL (One Way)

```
LSTMComponent(inputDim, outputDim, inputVal) = [
    Wxo = Parameter(outputDim, inputDim)
    Wxi = Parameter(outputDim, inputDim)
    Wxf = Parameter(outputDim, inputDim)
    Wxc = Parameter(outputDim, inputDim)

    bo = Parameter(outputDim, 1, init=fixedvalue, value=-1.0)
    bc = Parameter(outputDim, 1, init=fixedvalue, value=0.0)
    bi = Parameter(outputDim, 1, init=fixedvalue, value=-1.0)
    bf = Parameter(outputDim, 1, init=fixedvalue, value=-1.0)

    Whi = Parameter(outputDim, outputDim)
    Wci = Parameter(outputDim , 1)
    Whf = Parameter(outputDim, outputDim)
    Wcf = Parameter(outputDim , 1)
    Who = Parameter(outputDim, outputDim)
    Wco = Parameter(outputDim , 1)
    Whc = Parameter(outputDim, outputDim)
```

Wrapped as a macro and can be reused

parameters

Network Definition with NDL (One Way)

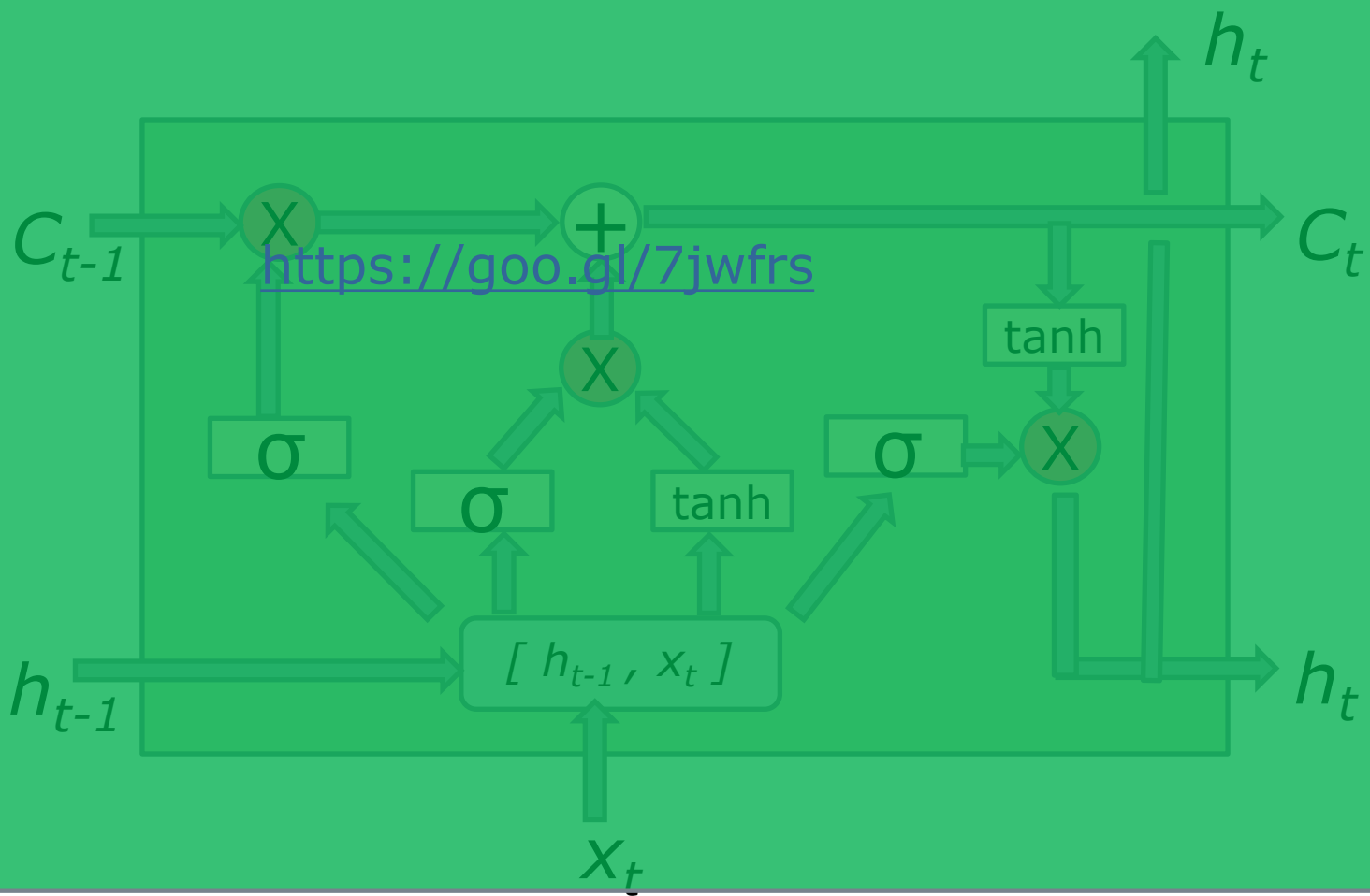
```
delayH = PastValue(outputDim, output, timeStep=1)
delayC = PastValue(outputDim, ct, timeStep=1)

WxiInput = Times(Wxi, inputVal)
WhidelayHI = Times(Whi, delayH)
WcidelayCI = DiagTimes(Wci, delayC)

it = Sigmoid (Plus ( Plus (Plus (WxiInput, bi), WhidelayHI),
      WcidelayCI))

WhfdelayHF = Times(Whf, delayH)
WcfdelayCF = DiagTimes(Wcf, delayC)
Wxfinput = Times(Wxf, inputVal)

ft = Sigmoid( Plus (Plus (Plus(Wxfinput, bf), WhfdelayHF),
      WcfdelayCF))
```

recurrent nodes

$$\mathbf{i}_t = \sigma\left(\mathbf{W}^{(xi)}\mathbf{x}_t + \mathbf{W}^{(hi)}\mathbf{h}_{t-1} + \mathbf{W}^{(ci)}\mathbf{c}_{t-1} + \mathbf{b}^{(i)}\right)$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}^{(xf)}\mathbf{x}_t + \mathbf{W}^{(hf)}\mathbf{h}_{t-1} + \mathbf{W}^{(cf)}\mathbf{c}_{t-1} + \mathbf{b}^{(f)}\right)$$

MXNetでのLSTMの定義



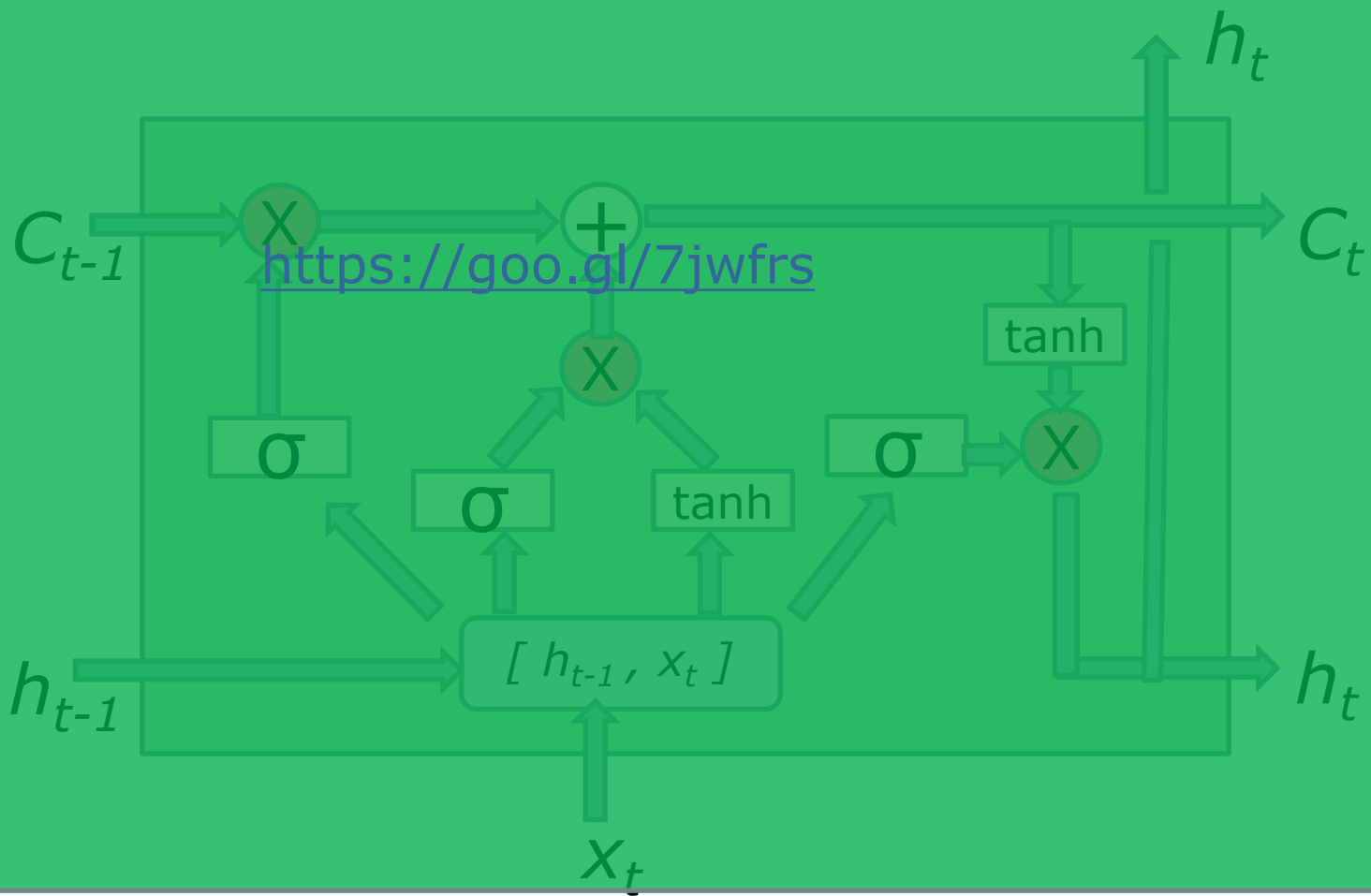
```
# lstm cell symbol
lstm <- function(num.hidden, indata, prev.state, param, seqidx, layeridx, dropout=0) {
    if (dropout > 0)
        indata <- mx.symbol.Dropout(data=indata, p=dropout)
    i2h <- mx.symbol.FullyConnected(data=indata,
                                    weight=param$i2h.weight,
                                    bias=param$i2h.bias,
                                    num.hidden=num.hidden * 4,
                                    name=paste0("t", seqidx, ".l", layeridx, ".i2h"))
    h2h <- mx.symbol.FullyConnected(data=prev.state$h,
                                    weight=param$h2h.weight,
                                    bias=param$h2h.bias,
                                    num.hidden=num.hidden * 4,
                                    name=paste0("t", seqidx, ".l", layeridx, ".h2h"))
    gates <- i2h + h2h
    slice.gates <- mx.symbol.SliceChannel(gates, num.outputs=4,
                                          name=paste0("t", seqidx, ".l", layeridx, ".slice"))

    in.gate <- mx.symbol.Activation(slice.gates[[1]], act.type="sigmoid")
    in.transform <- mx.symbol.Activation(slice.gates[[2]], act.type="tanh")
    forget.gate <- mx.symbol.Activation(slice.gates[[3]], act.type="sigmoid")
    out.gate <- mx.symbol.Activation(slice.gates[[4]], act.type="sigmoid")
    next.c <- (forget.gate * prev.state$c) + (in.gate * in.transform)
    next.h <- out.gate * mx.symbol.Activation(next.c, act.type="tanh")

    return (list(c=next.c, h=next.h))
}
```

```r
# unrolled lstm network
lstm.unroll <- function(num.lstm.layer, seq.len, input.size,
                        num.hidden, num.embed, num.label, dropout=0.) {

    embed.weight <- mx.symbol.Variable("embed.weight")
    cls.weight <- mx.symbol.Variable("cls.weight")
    cls.bias <- mx.symbol.Variable("cls.bias")

    param.cells <- lapply(1:num.lstm.layer, function(i) {
        cell <- list(i2h.weight = mx.symbol.Variable(paste0("l", i, ".i2h.weight")),
                     i2h.bias = mx.symbol.Variable(paste0("l", i, ".i2h.bias")),
                     h2h.weight = mx.symbol.Variable(paste0("l", i, ".h2h.weight")),
                     h2h.bias = mx.symbol.Variable(paste0("l", i, ".h2h.bias")))
        return (cell)
    })
    last.states <- lapply(1:num.lstm.layer, function(i) {
        state <- list(c=mx.symbol.Variable(paste0("l", i, ".init.c")),
                      h=mx.symbol.Variable(paste0("l", i, ".init.h")))
        return (state)
    })

    # embeding layer
    label <- mx.symbol.Variable("label")
    data <- mx.symbol.Variable("data")
    embed <- mx.symbol.Embedding(data=data, input_dim=input.size,
                                 weight=embed.weight, output_dim=num.embed, name="embed")
    wordvec <- mx.symbol.SliceChannel(data=embed, num_outputs=seq.len, squeeze_axis=1)
```

```
    last.hidden <- list()
    for (seqidx in 1:seq.len) {
        hidden <- wordvec[[seqidx]]
        # stack lstm
        for (i in 1:num.lstm.layer) {
            dp <- ifelse(i==1, 0, dropout)
            next.state <- lstm(num.hidden, indata=hidden,
                               prev.state=last.states[[i]],
                               param=param.cells[[i]],
                               seqidx=seqidx, layeridx=i,
                               dropout=dp)
            hidden <- next.state$h
            last.states[[i]] <- next.state
        }
        # decoder
        if (dropout > 0)
            hidden <- mx.symbol.Dropout(data=hidden, p=dropout)
        last.hidden <- c(last.hidden, hidden)
    }
    last.hidden$dim <- 0
    last.hidden$num.args <- seq.len
    concat <-mxnet:::mx.varg.symbol.Concat(last.hidden)
    fc <- mx.symbol.FullyConnected(data=concat,
                                   weight=cls.weight,
                                   bias=cls.bias,
                                   num.hidden=num.label)
```

以下略

TensorFlow FoldでのLSTMの記述



hierarchical LSTM

```
# Create RNN cells using the TensorFlow RNN library
char_cell = td.ScopedLayer(tf.contrib.rnn.BasicLSTMCell(num_units=16), 'char_cell')
word_cell = td.ScopedLayer(tf.contrib.rnn.BasicLSTMCell(num_units=32), 'word_cell')

# character LSTM converts a string to a word vector
char_lstm = (td.InputTransform(lambda s: [ord(c) for c in s]) >>
             td.Map(td.Scalar('int32') >>
                    td.Function(td.Embedding(128, 8))) >>
             td.RNN(char_cell))
# word LSTM converts a sequence of word vectors to a sentence vector.
word_lstm = td.Map(char_lstm >> td.GetItem(1)) >> td.RNN(word_cell)
```

LSTM Cell

this definition for the illustrative purposes only

```
# The input to lstm_cell is (input_vec, (previous_cell_state, previous_output_vec))
# The output of lstm_cell is (output_vec, (next_cell_state, output_vec))
lstm_cell = td.Composition()
with lstm_cell.scope():
  in_state = td.Identity().reads(lstm_cell.input[1])
  bx = td.Concat().reads(lstm_cell.input[0], in_state[1])     # inputs to gates
  bi = td.Function(td.FC(num_hidden, tf.nn.sigmoid)).reads(bx)  # input gate
  bf = td.Function(td.FC(num_hidden, tf.nn.sigmoid)).reads(bx)  # forget gate
  bo = td.Function(td.FC(num_hidden, tf.nn.sigmoid)).reads(bx)  # output gate
  bg = td.Function(td.FC(num_hidden, tf.nn.tanh)).reads(bx)     # modulation
  bc = td.Function(lambda c,i,f,g: c*f + i*g).reads(in_state[0], bi, bf, bg)
  by = td.Function(lambda c,o: tf.tanh(c) * o).reads(bc, bo)    # final output
  out_state = td.Identity().reads(bc, by)   # make a tuple of (bc, by)
  lstm_cell.output.reads(by, out_state)
```