ニューラルネット勉強会（LSTM編）

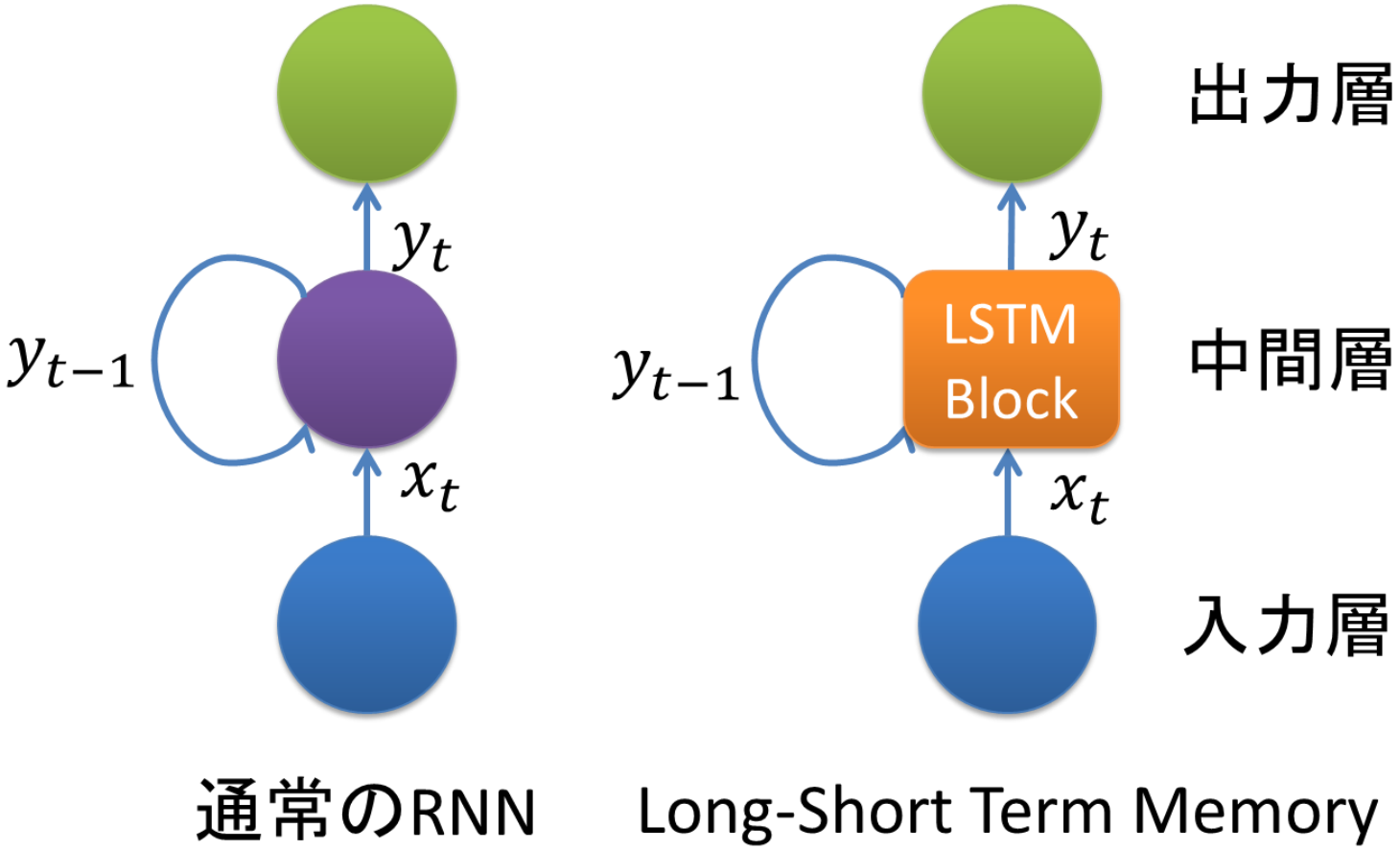
2016/10/25

Seitaro Shinagawa

主にニューラルネットワークの基礎については既に分かっていて、最近LSTMを触り始めた人向けの解説です。

LSTMの仕組み、知っておくべき関連知識、chainerにおけるtipsについて紹介します。

1．RNNからLSTMへ



わかるLSTM ～ 最近の動向と共に　から引用
(http://qiita.com/t_Signull/items/21b82be280b46f467d1b)

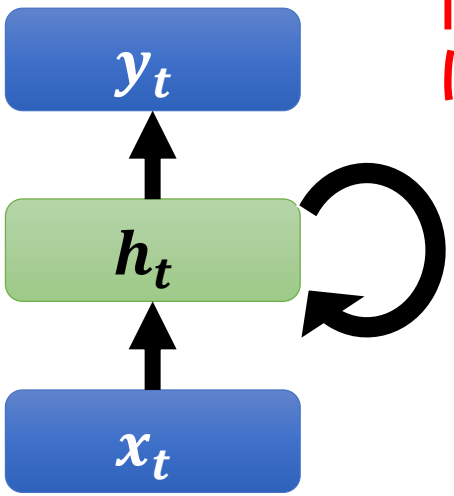
LSTMビギナーの学生からよく出るQ&A

A君

LSTMってRNNの一種ですよね？
僕の知ってるのと書き方が違う気がする・・・

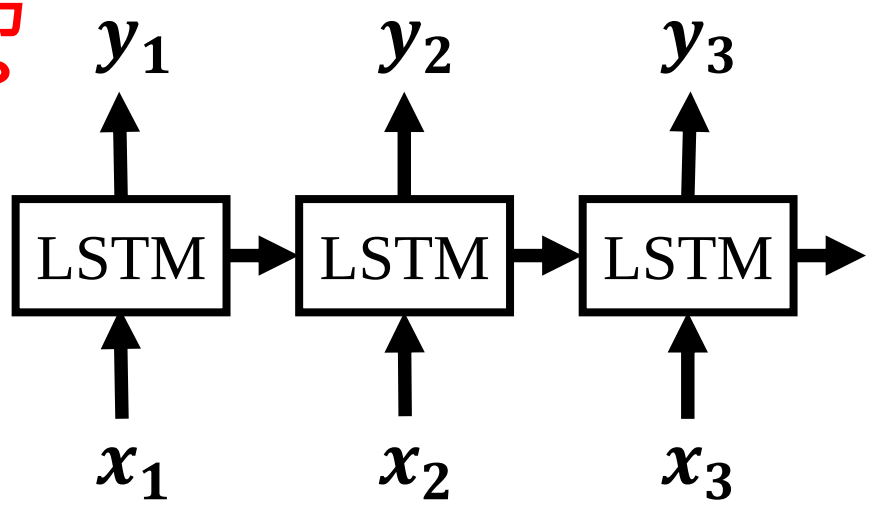
基本同じ！
数式を追えばきっと分かるよ！

ニューラルな
ツキノワグマ



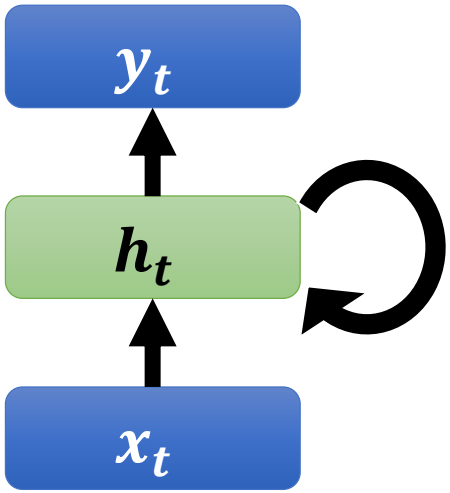
A君の知っているRNN

両者の書き方は何が違う？



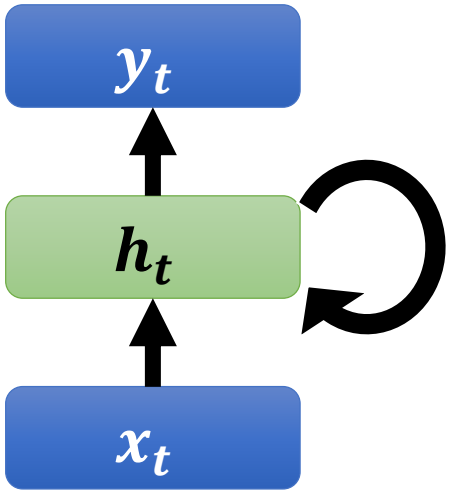
A君が良く見るLSTM

RNNから展開してみる



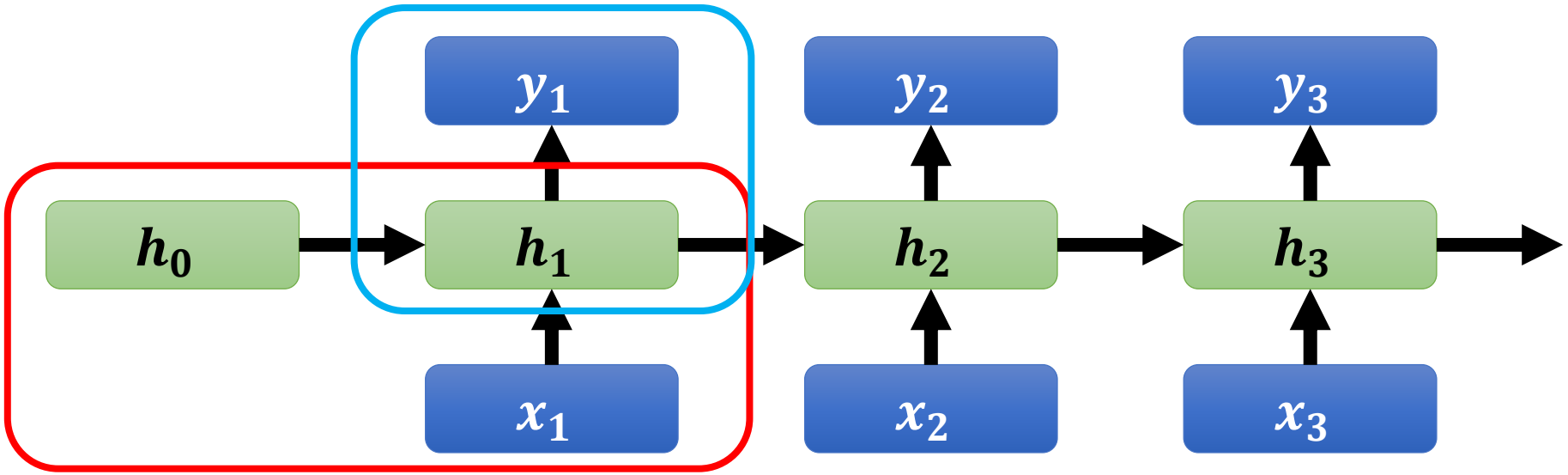
RNNから展開してみる

時間展開してみる



RNNから展開してみる

時間展開してみる



ここまではRNNでよく見ますね！

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})$$

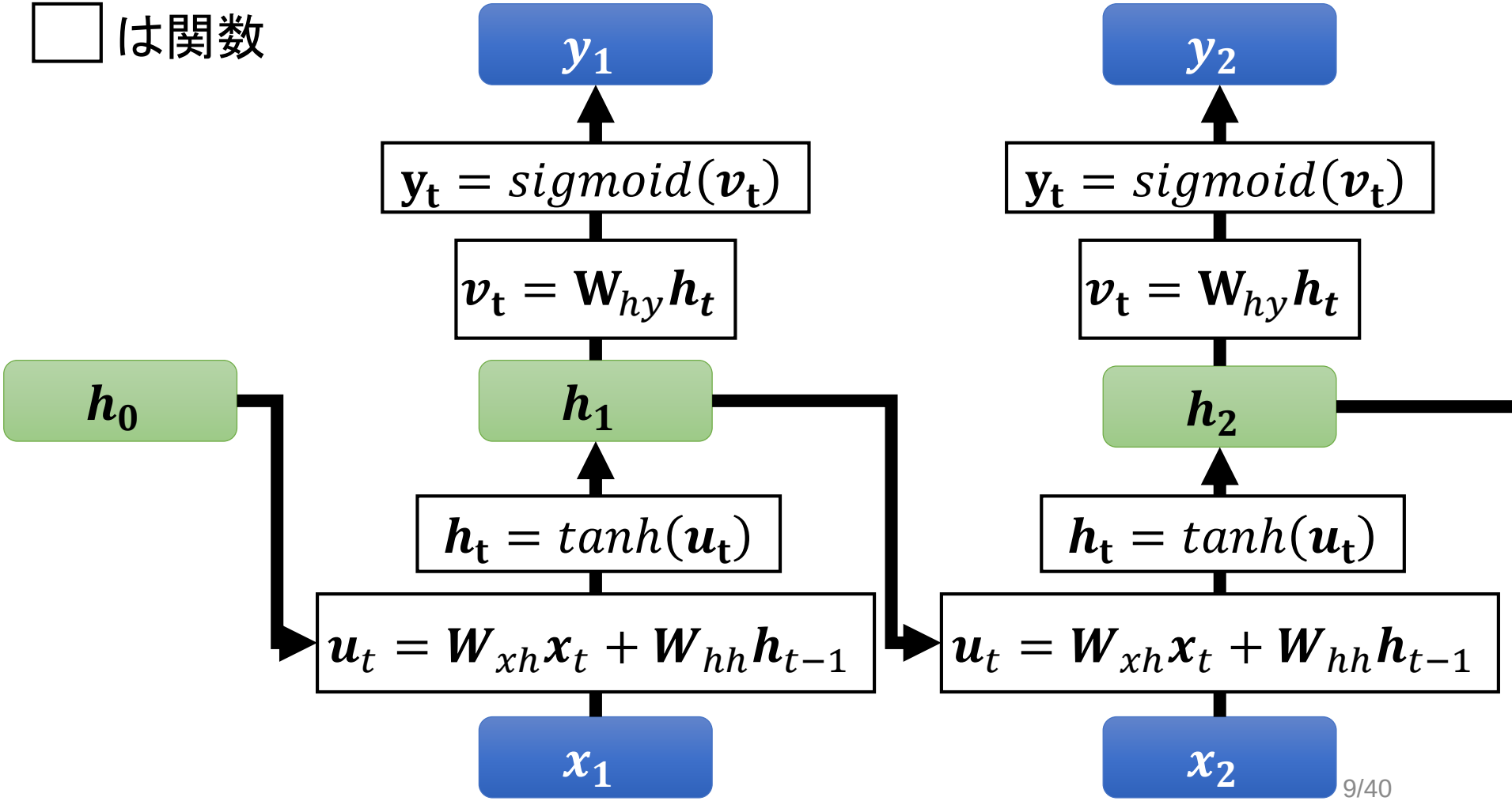
$$\boldsymbol{y}_t = sigmoid(\boldsymbol{W}_{hy}\boldsymbol{h}_t)$$

だから、変数に注目してその関係性を記述している図だと言えるね。

もっと処理に注目した図にしてみる

RNNを引数x_t, h_{t-1}戻り値y_t, h_tの大きな関数としてみてみると・・・

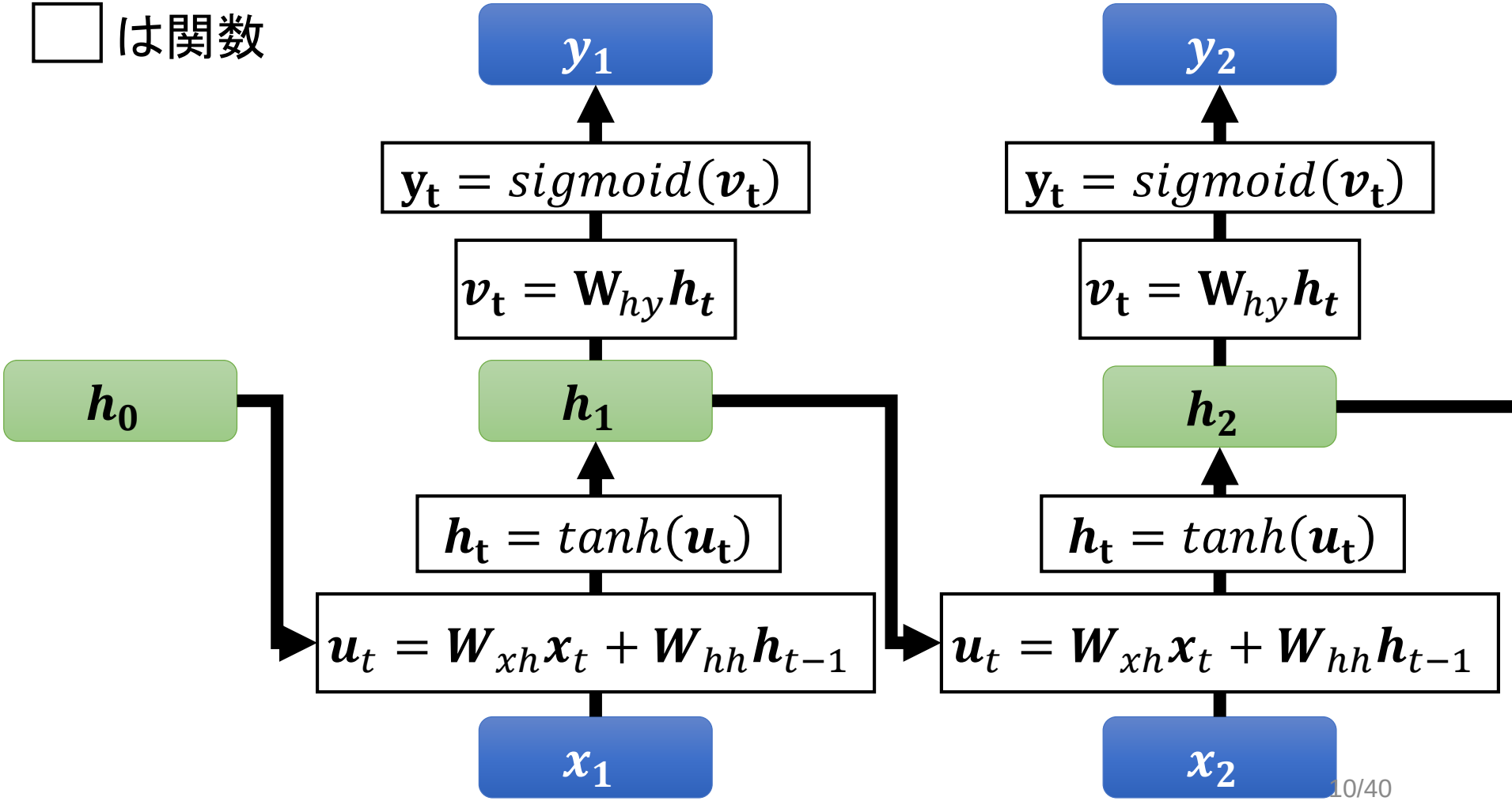
□は関数



もっと処理に注目した図にしてみる

RNNを引数x_t, h_{t-1}戻り値y_t, h_tの大きな関数としてみてみると・・・

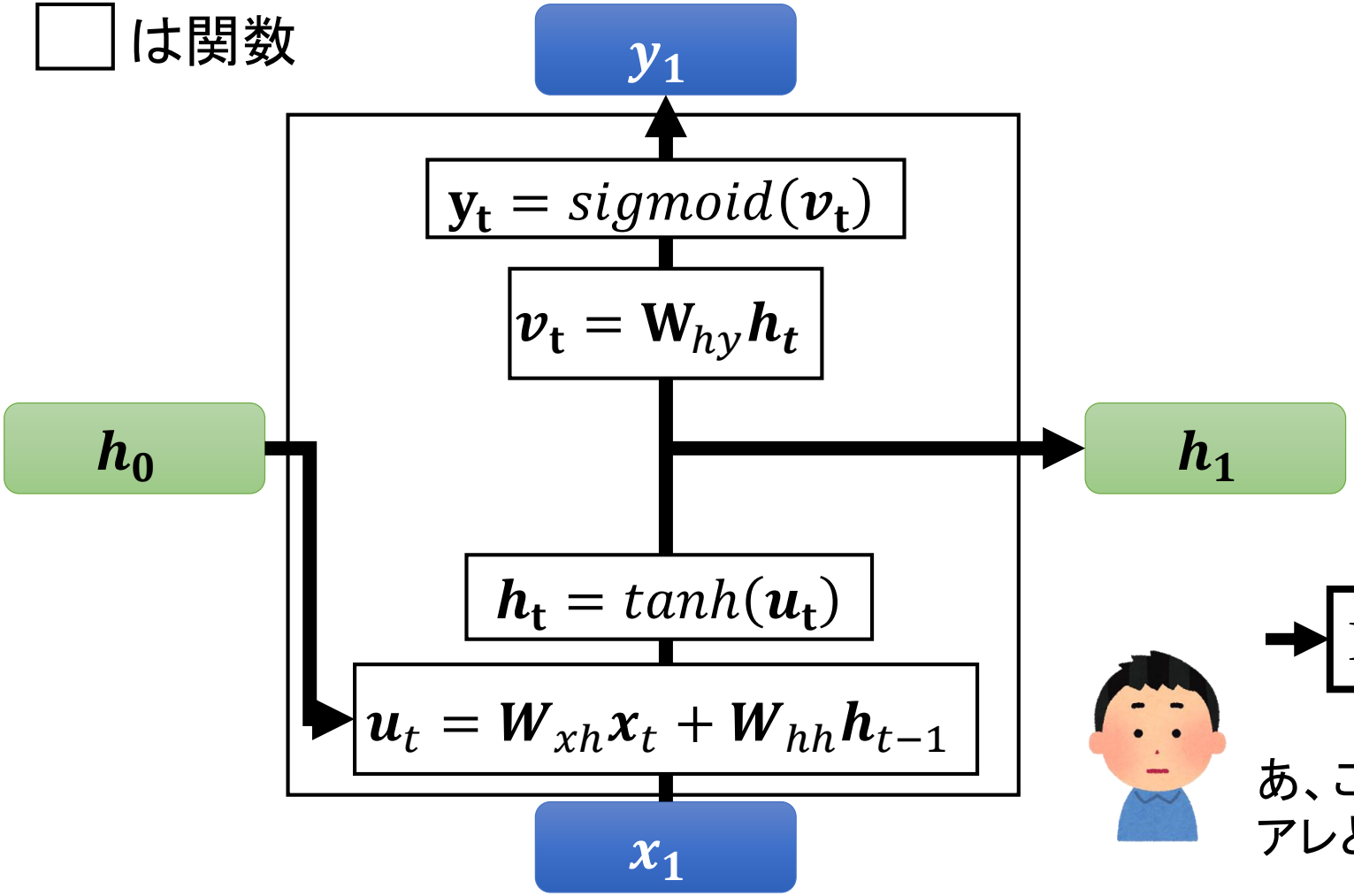
□は関数

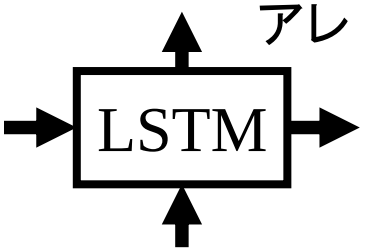


もっと処理に注目した図にしてみる

RNNを引数x_t, h_{t-1}戻り値y_t, h_tの大きな関数としてみてみると・・・

□は関数



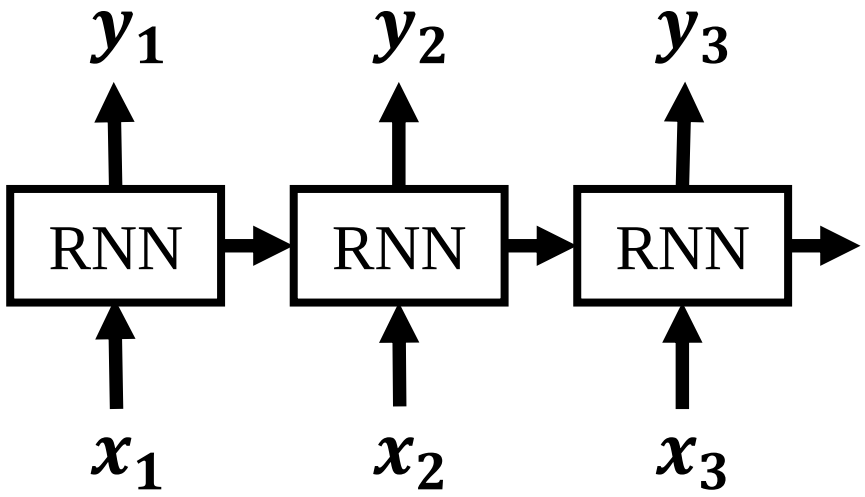


あ、これLSTMのアレと同じだ・・・

両者の書き方のまとめ

LSTMだけの特別な書き方でもなかったんですね！

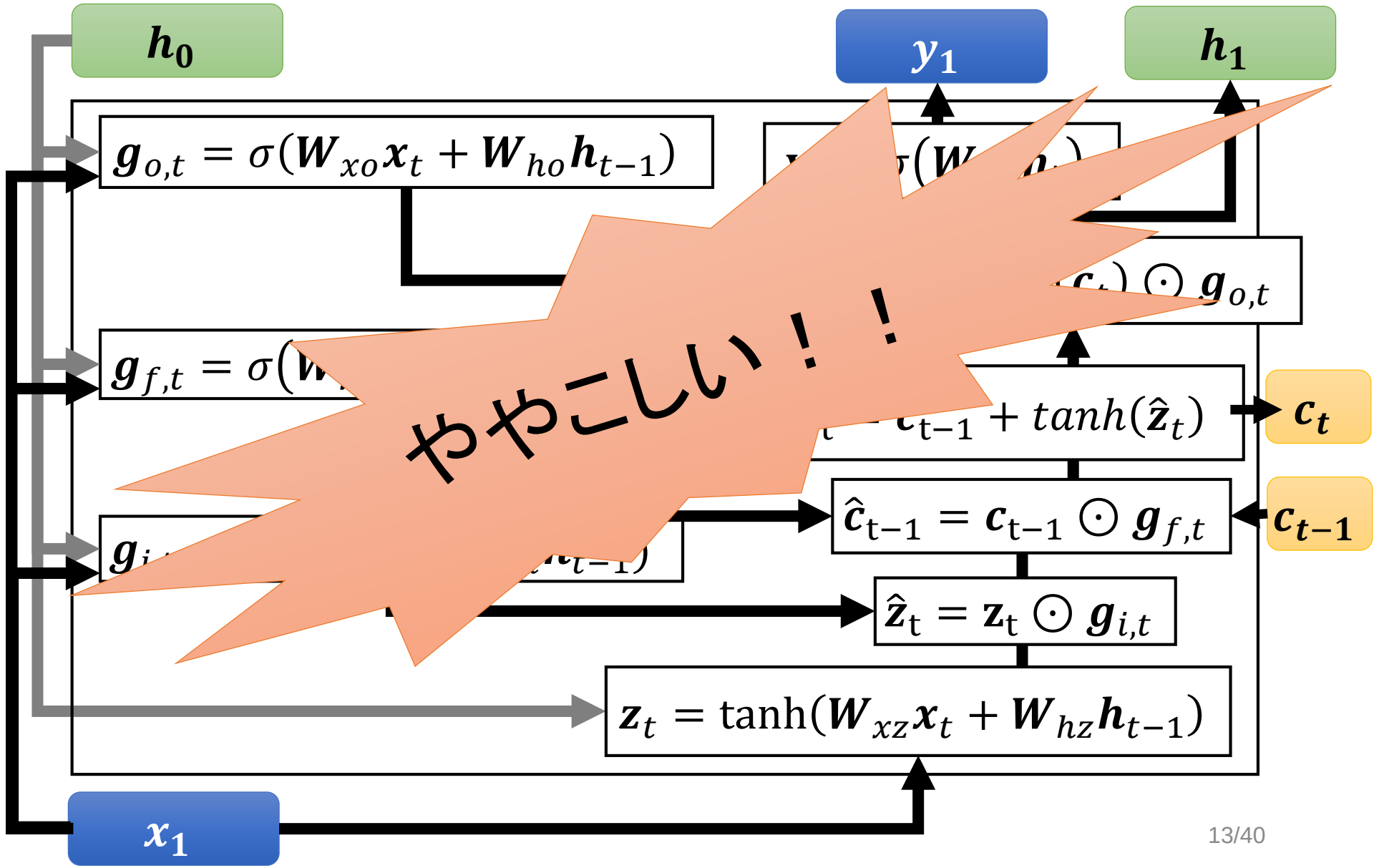
そうだね。ちなみに隠れ層の初期値h_0は下のように省略されることが多いよ。



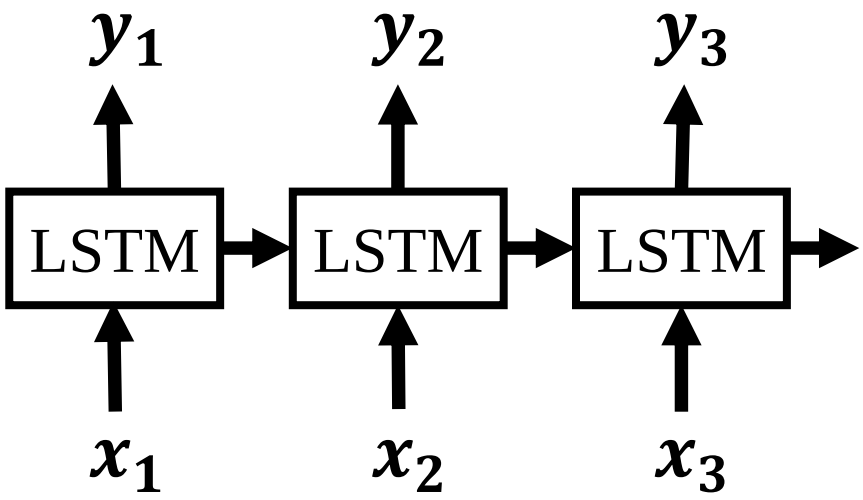
ちなみにRNNの部分がLSTMだと、本当はcellの値も次の時間に渡すけど、大体省略されるよ。

ちなみに、LSTMの中身をちゃんと書くとこうなります

$(\sigma(\cdot) = sigmoid(\cdot)$とする)



LSTMの図をみたら中身はあんな感じだったなとぜひ思い出してみてください



LSTMでよくある質問

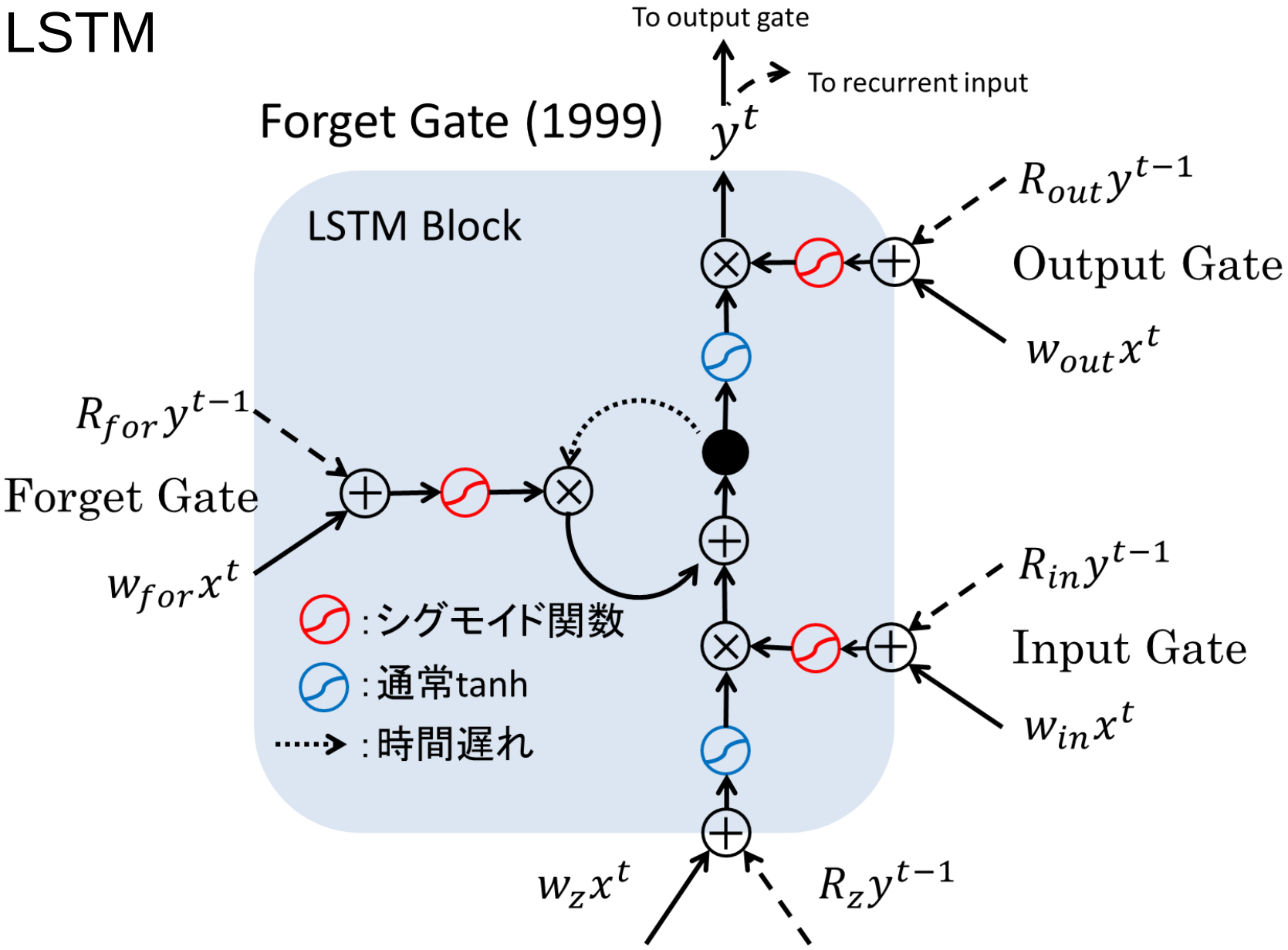
Q. RNNとLSTMの違いは？

- Constant Error Carousel(CEC, いわゆるcell)の導入
- input gate, forget gate, outputの導入
  - Input gate: 入力を取り込むか選ぶ
  - Forget gate: cellに保持している情報をリセットするか選ぶ
  - Output gate:　次の時刻にどの程度情報を伝えるか選ぶ

Q. LSTMが勾配消失問題を解決してるってどういうこと？三行でよろ

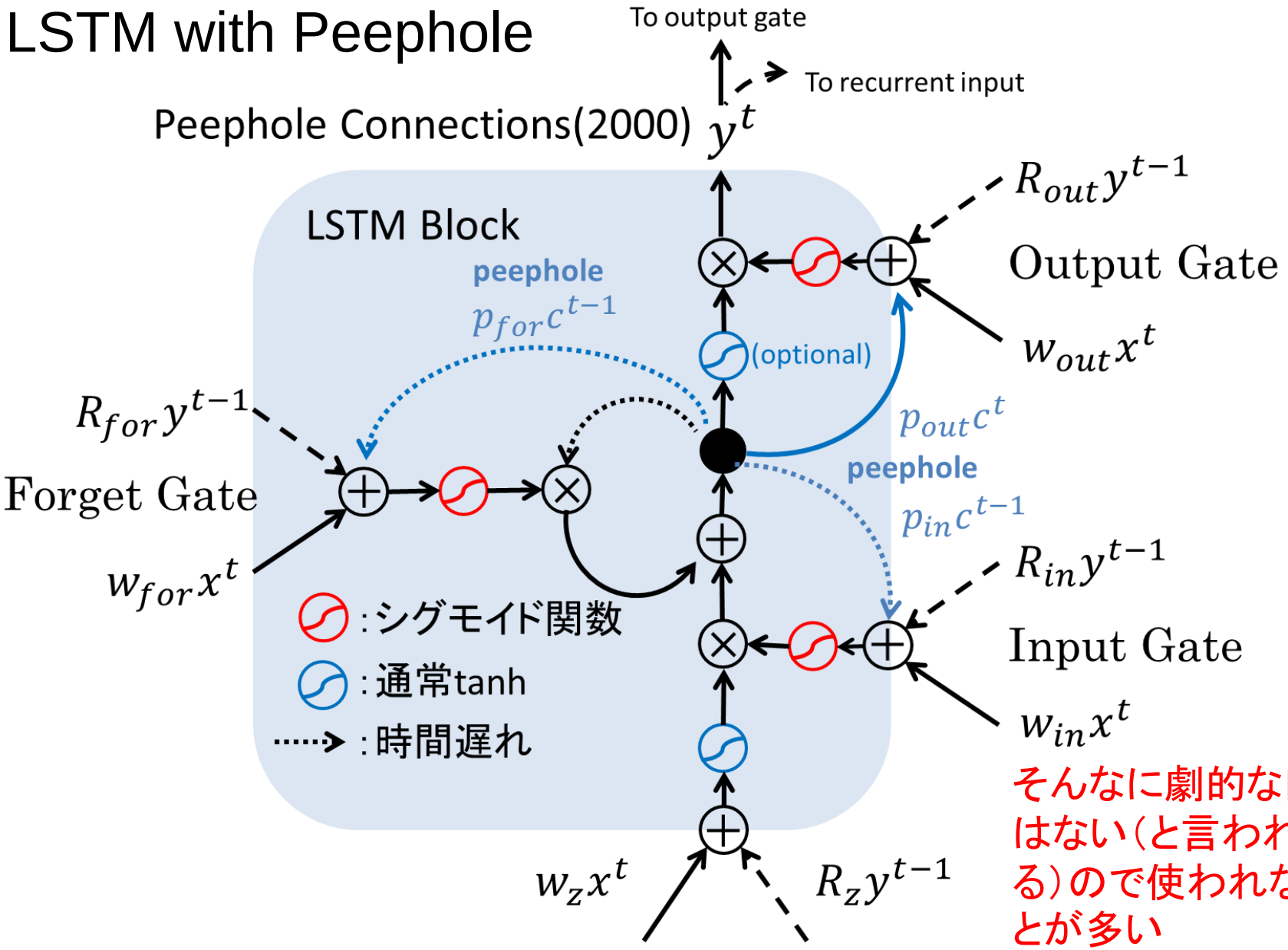
1. BPはシグモイド関数の微分の乗算を繰り返すので勾配消失する
2. RNNは時系列の情報が隠れ層に埋め込まれるので影響を受ける
3. LSTMはcellに前の系列が線形和で保持されているので影響を受けない（覚えられる系列には当然限界がある）

LSTM



わかるLSTM ～ 最近の動向と共に　から引用
(http://qiita.com/t_Signull/items/21b82be280b46f467d1b)

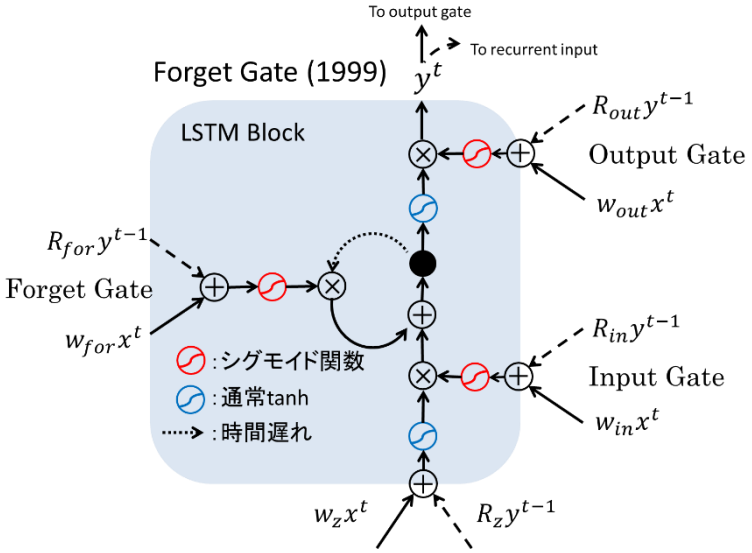
LSTM with Peephole



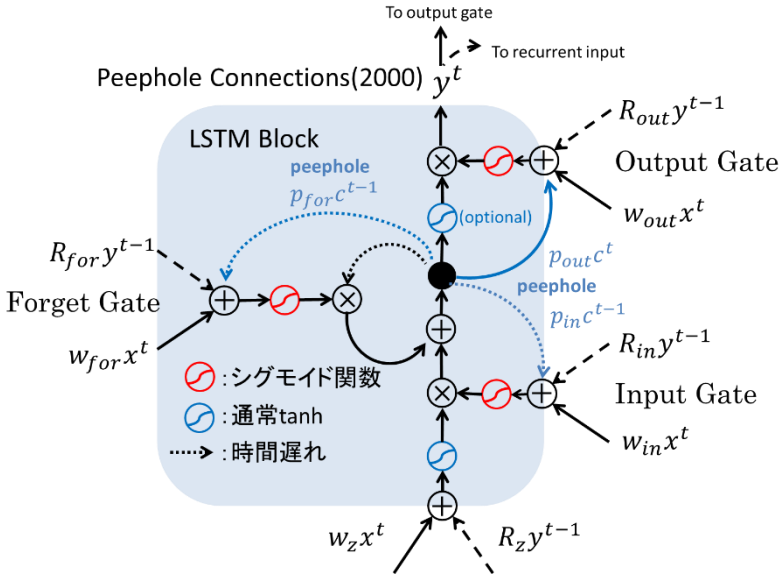
そんなに劇的な向上はない(と言われている)ので使われないことが多い

わかるLSTM ～ 最近の動向と共に から引用
(http://qiita.com/t_Signull/items/21b82be280b46f467d1b)

Chainerではどうなってる？



Peepholeなし(こちらが標準)
chainer.links.LSTM



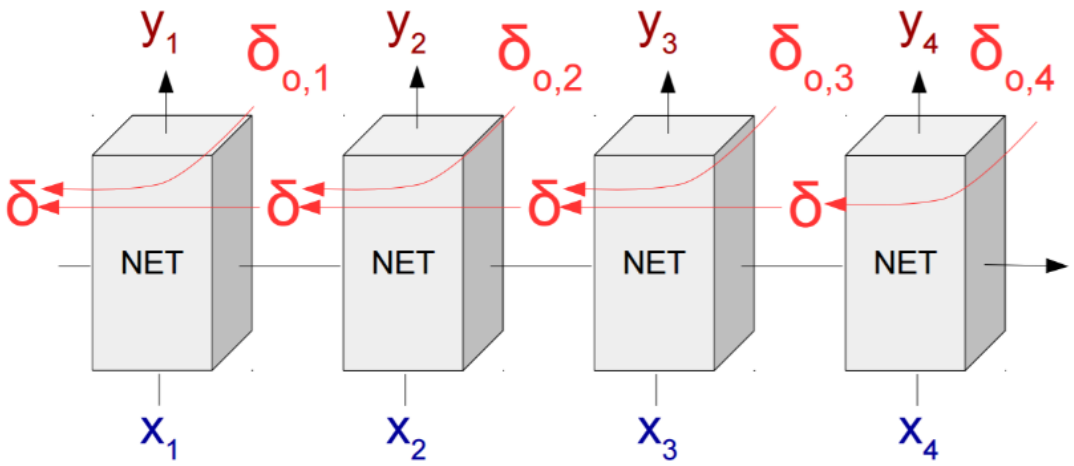
Peepholeあり
chainer.links.StatefulPeepholeLSTM

ちなみに、この部分はhをローカル変数として外に出すかどうかを表している(※)

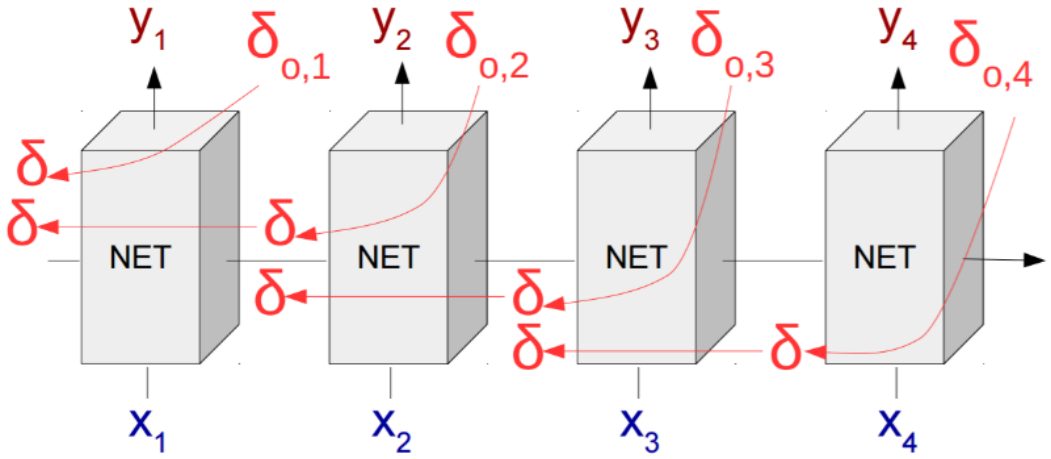
Stateful〇〇	Stateless〇〇
stateful_lstm(x1)	h = init_state()
stateful_lstm(x2)	h = stateless_lstm(h, x1)
	h = stateless_lstm(h, x2)

(※) https://groups.google.com/forum/#!topic/chainer-jp/bJ9IQWtsef4

2. LSTMの学習手法



Full BPTT



Truncated BPTT

（BPTTはBack Propagation Trough Timeの略）

Graham Neubig NLPプログラミング勉強会8- リカレントニューラルネットワーク　から引用
http://www.phontron.com/slides/nlp-programming-ja-08-rnn.pdf

ChainerでTruncated BPTT

unchain_backward による Truncated BPTT

- Truncated BPTT は `unchain_backward()` を使うことで実装できる
  - Variable から逆向きに計算グラフをたどったときに通る辺をすべて計算グラフから取り除く
  - Python 変数に保持している Variable オブジェクトはそのまま残る

```
accum_loss = 0
for i, x in enumerate(batches):
    loss, h = forward_on_step(*x)  # forward
    accum_loss += loss
    if i % 30 == 0:
        optimizer.zero_grads()
        accum_loss.backward()  # backward
        accum_loss.unchain_backward()  # truncate graph
        optimizer.update()
        accum_loss = 0
```

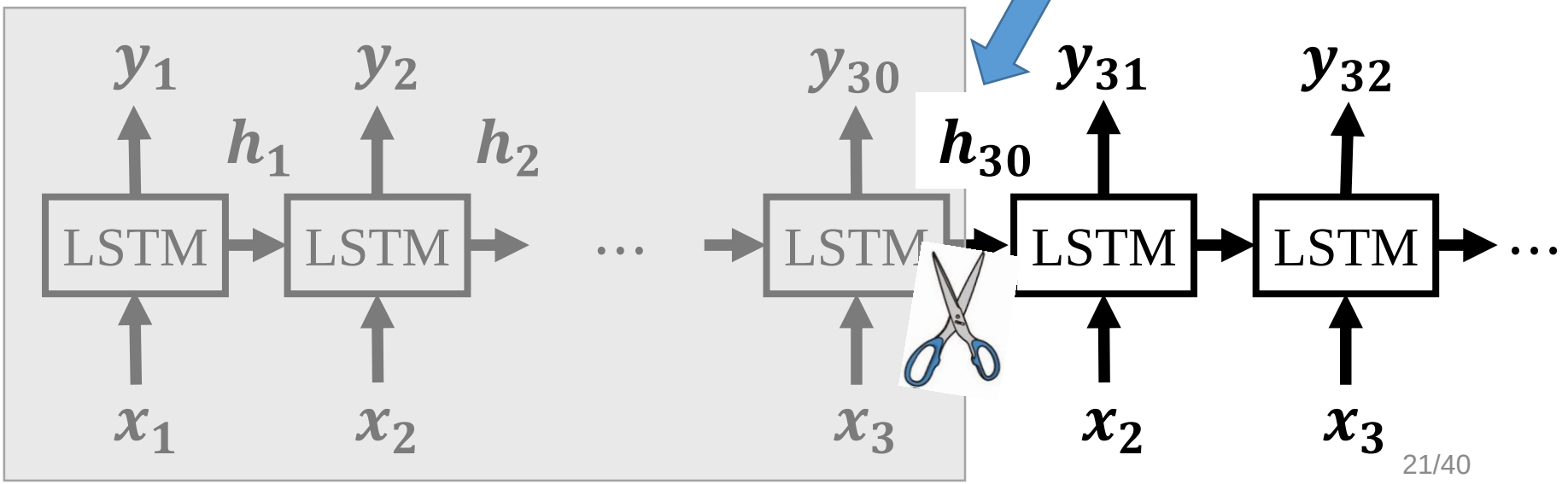
Chainerの使い方と自然言語処理への応用　から引用
http://www.slideshare.net/beam2d/chainer-52369222

ChainerでTruncated BPTT

```
accum_loss = 0
for i, x in enumerate(batches):
    loss, h = forward_on_step(*x)
    accum_loss += loss
    if i % 30 == 0:
        optimizer.zero_grads()
        accum_loss.backward()   # backwa
        accum_loss.unchain_backward()      uncate graph
        optimizer.update()
        accum_loss = 0
```

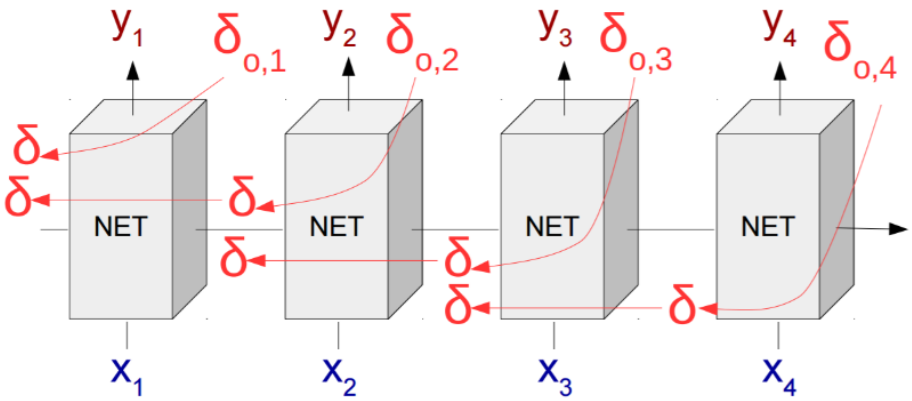
この実装ではbackward()の後に必ずupdate()をしている
前の系列の学習を諦めているというよりは、勾配更新を時系列の塊に区切って更新していると思った方が良い

$i = 60$ではここまでBPTTを行う



あれ？でもこのTruncated BPTTってこの図と違くないですか？

ソウデスネ・・・



この図はBPTTとRTRL（※）がミックスされてると解釈できる？

順伝播を計算しながらあらかじめ計算済みの順伝播の逆伝播を並列に計算できる（オンライン学習に向いている）

※詳しくはこちら　ニューラルネットワークで時系列データの予測を行う
http://qiita.com/icoxfog417/items/2791ee878deee0d0fd9c

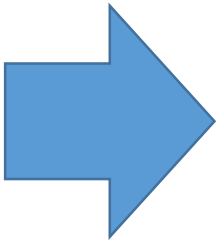
ミニバッチ計算への対応

ミニバッチにして並列計算したいんですけど、
長さが揃ってない可変長データはどうやって
ミニバッチにするんですか？

基本は末尾を適当に埋めます

例: 最後は必ず0で終わるとする

1 2 0
1 3 3 2 0
1 4 2 0



1 2 0 0 0
1 3 3 2 0
1 4 2 0 0

Zero paddingとか
呼んでます

ミニバッチ計算への対応

このままだとモデルが「**0が一度出たら0を出し続ける**」よう
に学習しなくてはならなくなるためモデルが冗長になります。

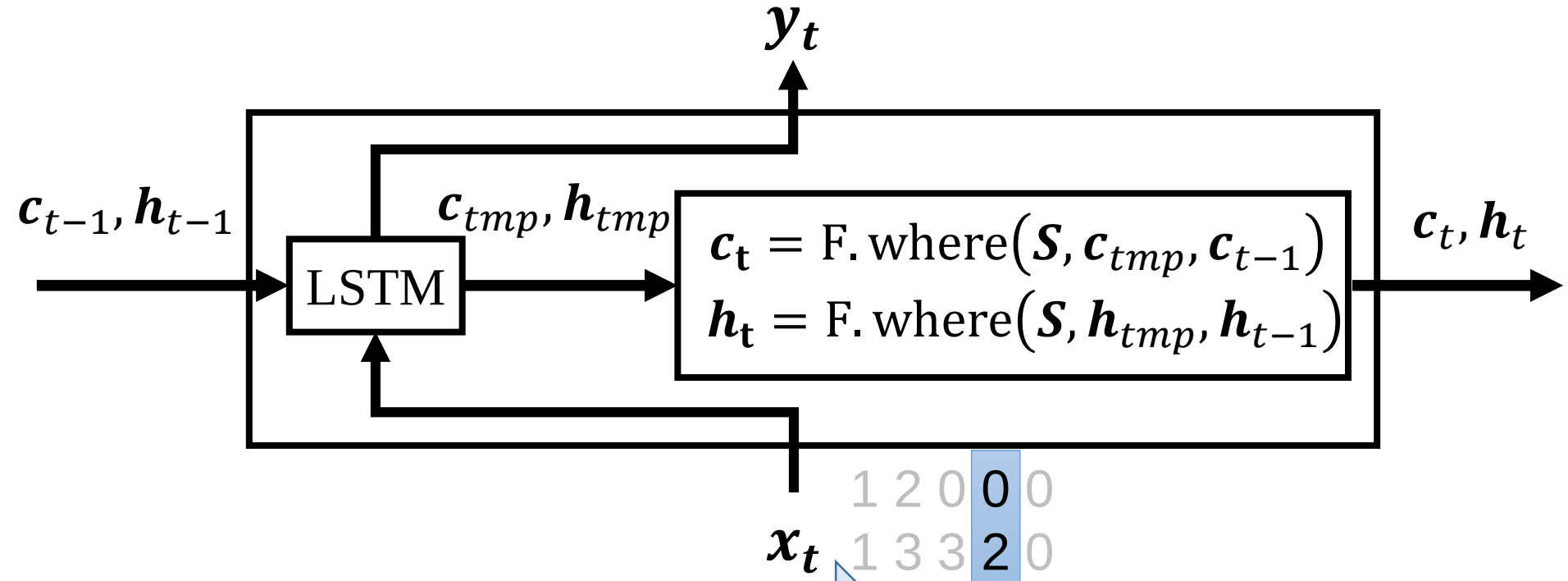
ルールで制約をかけると解決します。

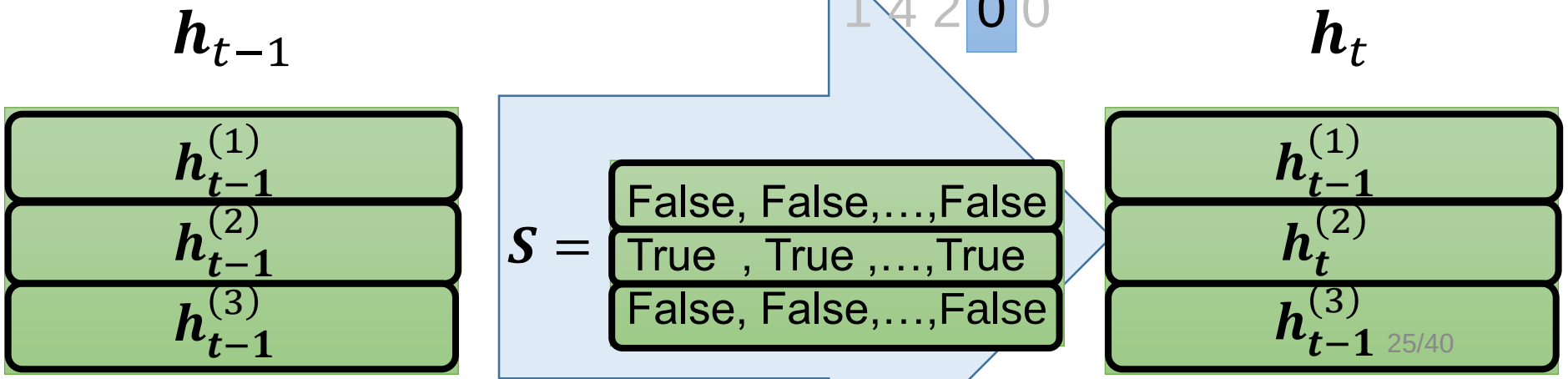
うまくやる方法は2つ

- chainer.functions.whereによるルール記述
- NStepLSTM(v1.16.0以降)を使う

chainer.functions.whereによるルール記述

要素ごとにTrueなら左、Falseなら右を選ぶ





NStepLSTM(v1.16.0以降)を使う

先ほどまでの内容をchainerが自動でやってくれる

ただ、cudnnをdropoutと一緒に使うとバグるらしい（※）
~~BugはfixされてV1.18.0でアップデートか~~
10/25 マスターにマージされた模様
https://github.com/pfnet/chainer/pull/1804

F.whereの話いらなかったのでは

本当につらい

（※）ChainerのNStepLSTMでニコニコ動画のコメント予測。
http://www.monthly-hack.com/entry/2016/10/24/200000

Gradient Clippingによる勾配爆発への対処

LSTMではしばしば勾配が爆発します（詳しくは※）
勾配に制約をかけることで回避できます。

※で提案されている方法

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

全ての勾配のノルムが閾値以上なら、その閾値にノルムを合わせるようにスケーリングする

Chainerではoptimizer.add_hook(Gradient_Clipping(threshold))で簡単に使える

※ On the difficulty of training recurrent neural networks
http://www.jmlr.org/proceedings/papers/v28/pascanu13.pdf

LSTMへのDropOutの適用について

DropOutは正則化手法として強力ですが、
何でもDropOutすればいいわけではありません。

※の論文によると、
1. LSTMの隠れ層の再帰部分にDropOutをかけた場合
2. LSTMのcellにDropOutをかけた場合
3. LSTMのinput gateにDropOutをかけた場合
で3. が一番良い性能であるそうです。

基本的に再帰している部分にはDropOutをかけず、forwardな
部分にだけDropOutをかけるのが良さそうです

※ Recurrent Dropout without Memory Loss
https://arxiv.org/abs/1603.05118

LSTMへのBatch Normalizationについて

Batch Normalizationとは？ http://jmlr.org/proceedings/papers/v37/ioffe15.pdf
　activation(入力と重みの積和)を活性化関数に通す前に平均0分散1になるようにスケーリングして学習高速化、精度向上を狙う手法

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Activation x に対してのBatch Normalization

本来は全学習データについてノーマライズするべきだが、
ミニバッチ計算のときは計算を簡単にするためミニバッチ内でBatch Normalizationを行う

LSTMへのBatch Normalizationについて

しかし、RNNについてはあまり効果がないと言われてきた（※）

- hidden-to-hiddenへの適用はスケーリングの繰り返しで勾配爆発が起きてしまい、学習がうまく進まない
- input-to-hiddenは学習が速くなるものの、汎化性は向上しない

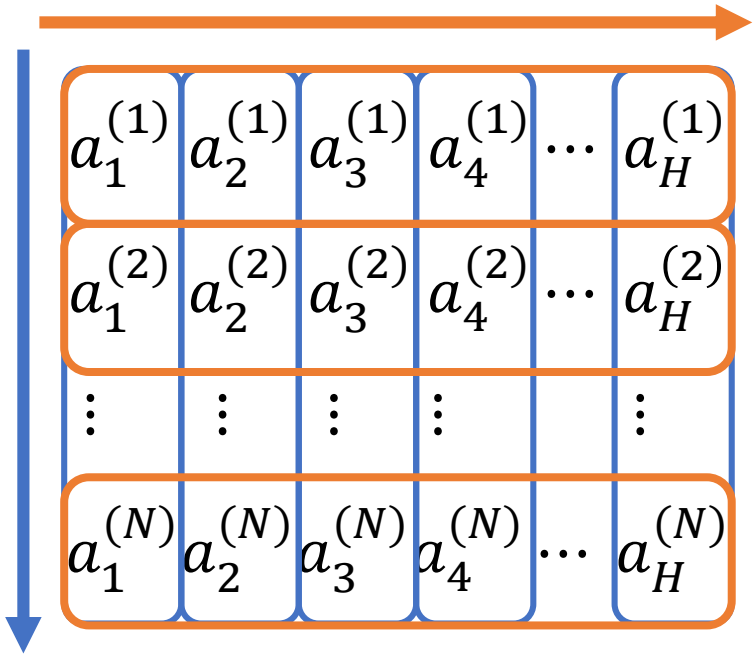
3つの提案がある（提案された順）

- (Weight Normalization) https://arxiv.org/abs/1602.07868
- (Recurrent Batch Normalization) https://arxiv.org/abs/1603.09025
- Layer Normalization https://arxiv.org/abs/1607.06450

※Batch Normalized Recurrent Neural Networks
https://arxiv.org/abs/1510.01378

Batch Normalization と Layer Normalization

x->hの間の activation $\boldsymbol{a}$ $(a_i^{(n)} = \Sigma_j w_{ij} x_j^{(n)}, \mathrm{h}_i^{(n)} = a_i^{(n)})$を考える



横方向、つまり
サンプルごとのニューロンについて正則化するのがLayer Normalization

縦方向、つまり
ニューロンごとのサンプルについて正則化するのがBatch Normalization

情報幾何的側面から勾配の爆発に対して分散σが大きくなり、全体の入出力が変わらないように正則化される（詳しくは元論文参照）

初期化のtips（時間なくて読めませんでした・・・）

- Exact solutions to the nonlinear dynamics of learning in deep linear neural networks
  https://arxiv.org/abs/1312.6120v3

- A Simple Way to Initialize Recurrent Networks of Rectified Linear Units
  https://arxiv.org/abs/1504.00941v2

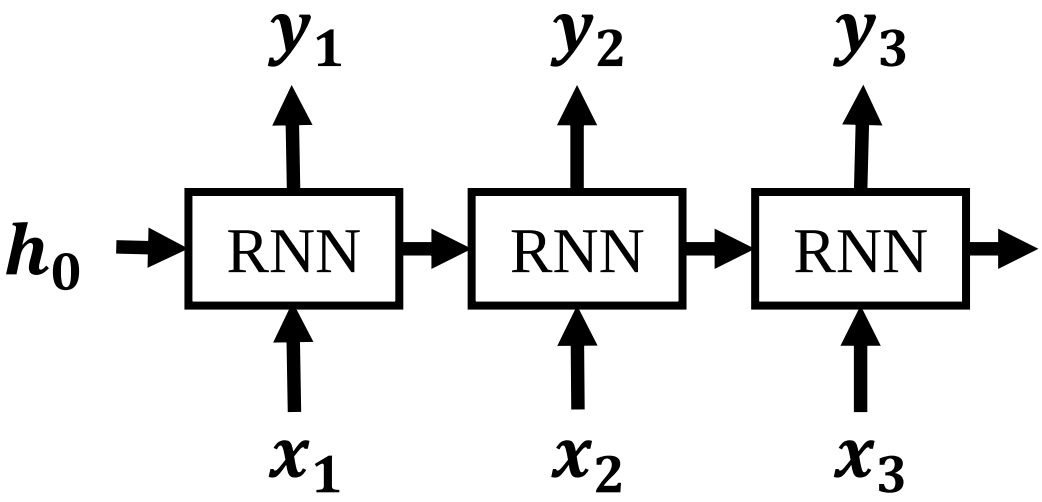
RNNにReLUを使う、回帰結合を単位行列で初期化することでLSTMと同程度の性能が出せる（らしい）

おまけ

RNNのモデルの色々

Encoder-Decoder
Bidirectional LSTM
Attention model

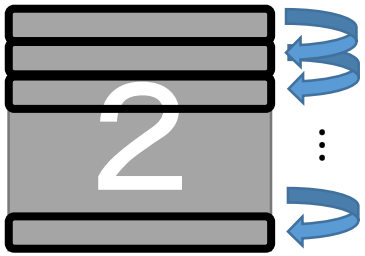
RNNの隠れ層の初期値に注目する



RNNは隠れ層の初期値h_0によって出力する内容が変わる

h_0もBPで学習できる
Encoderを使って適当な情報を紐づけることもできる
⇒言語モデルなどへの応用

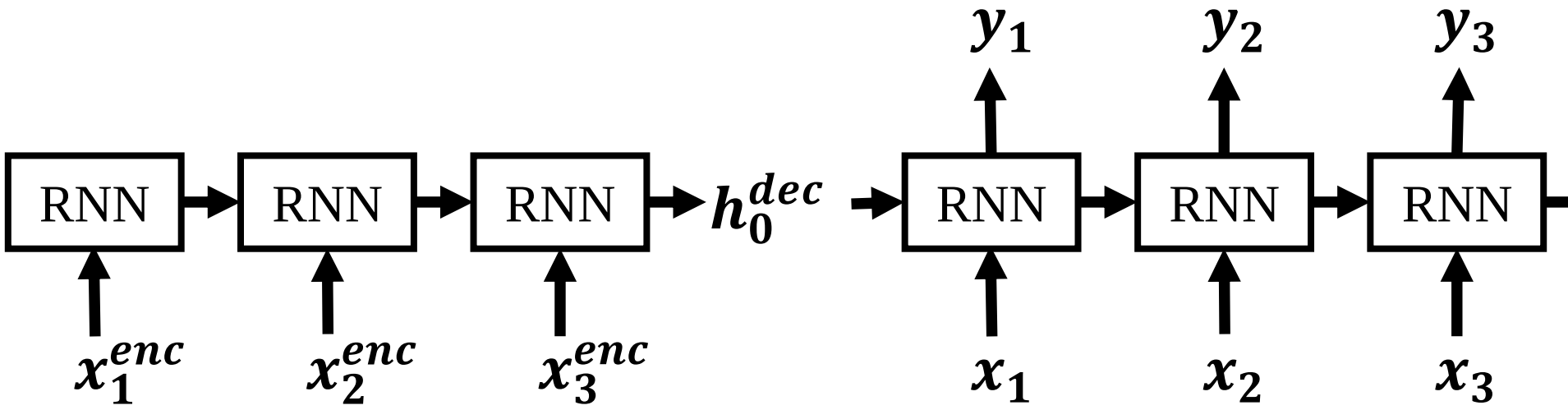
最初のスライスは大体どれも真っ黒だが、ちゃんと生成できる



original
学習したh_0から生成
ランダムh_0からの生成

RNNスライスpixel生成

Encoder-Decoder model

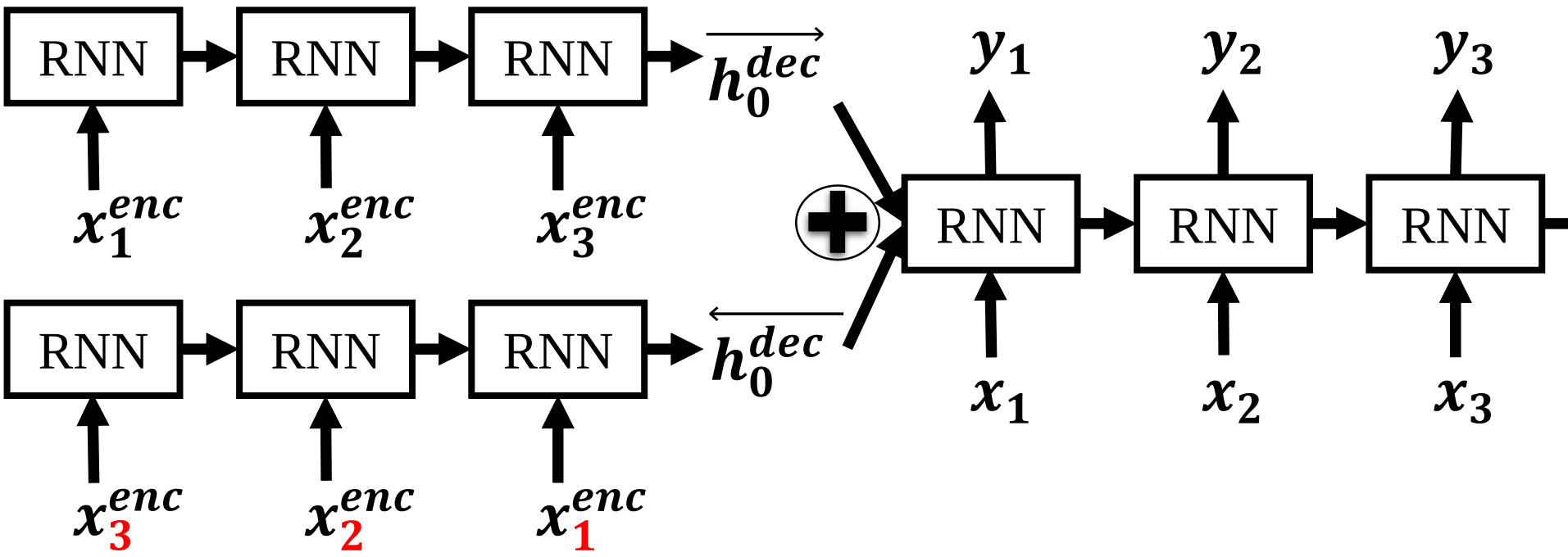


Encoderが可変長の入力を扱う場合には、こちらもRNNを使って入力を順番に埋め込んでいく

最後に出てきたh_0^{dec}が対応する情報を出力する理想的なDecoderの初期値となるようにEncoder側とDecoder側を同時に学習する

Decoderにはビームサーチを使うことが多い

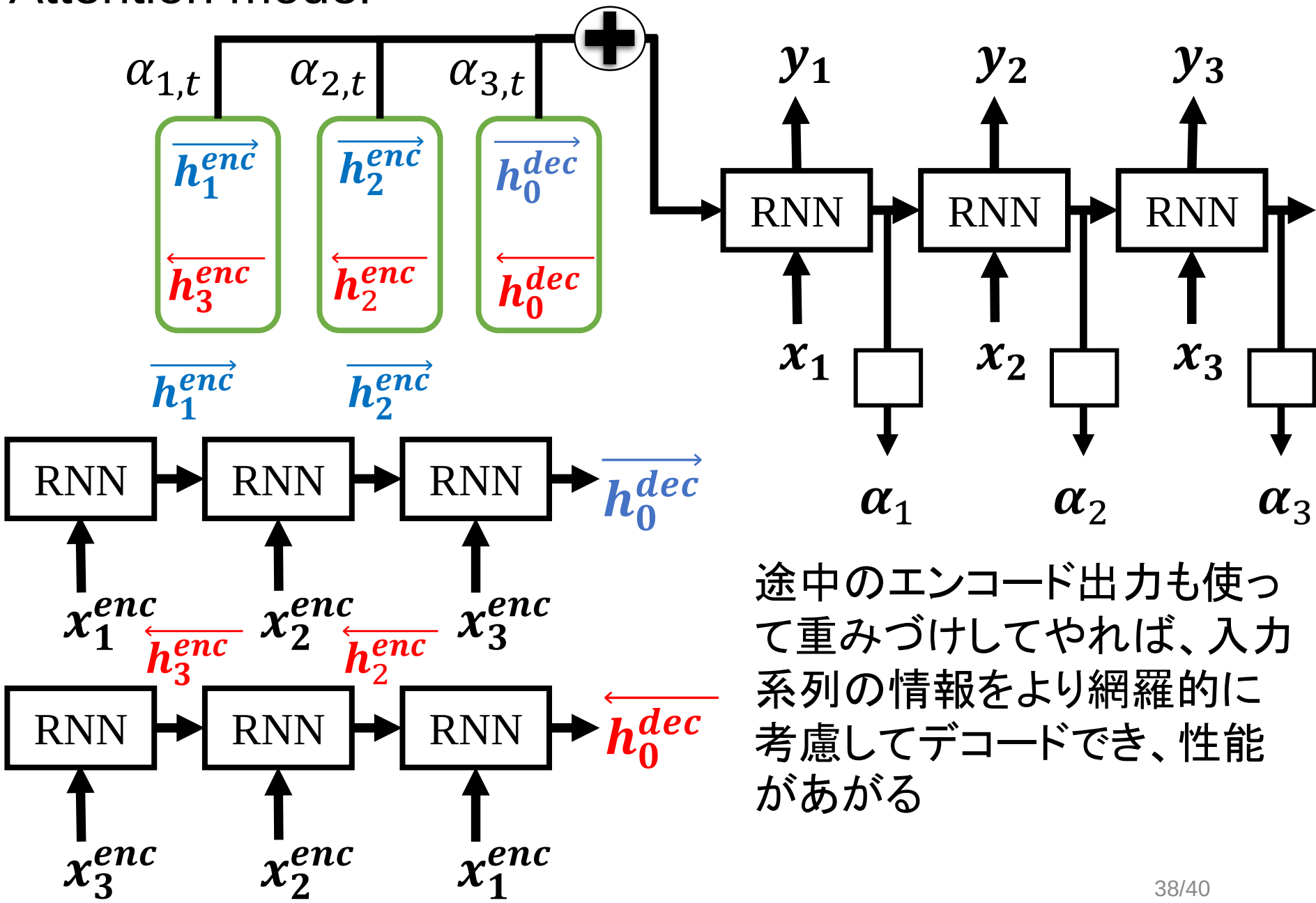
Bidirectional LSTM



LSTMを使えど、入力長が長い場合には前の系列の情報がh_0^{dec}に保持されなくなる（勾配消失問題による）

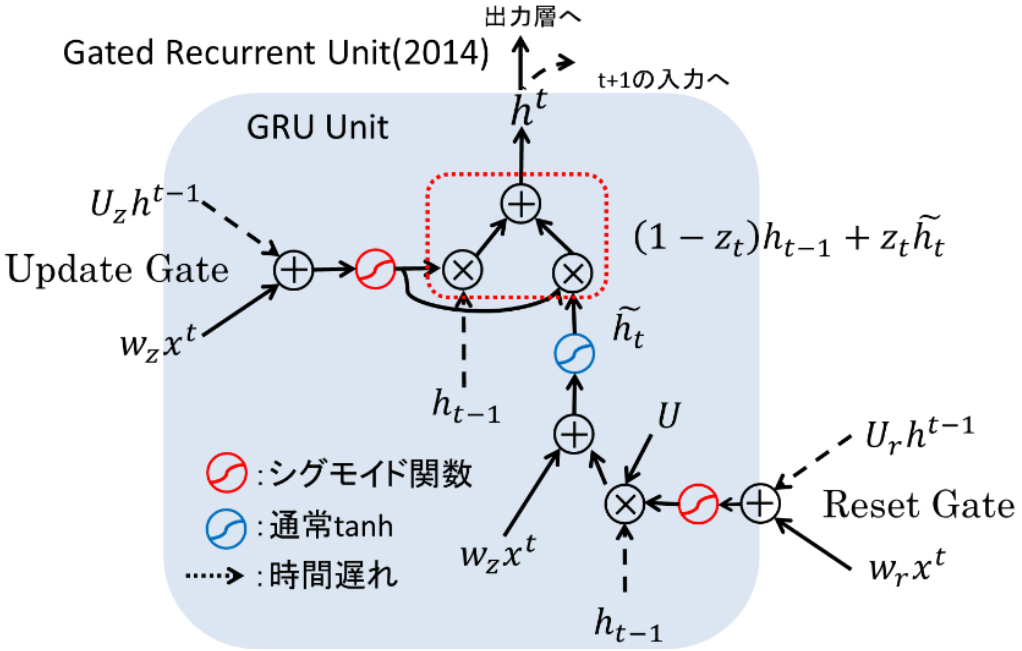
逆方向にした系列を入力するEncoderも用意することで前の方の情報も考慮することができ性能が上がる

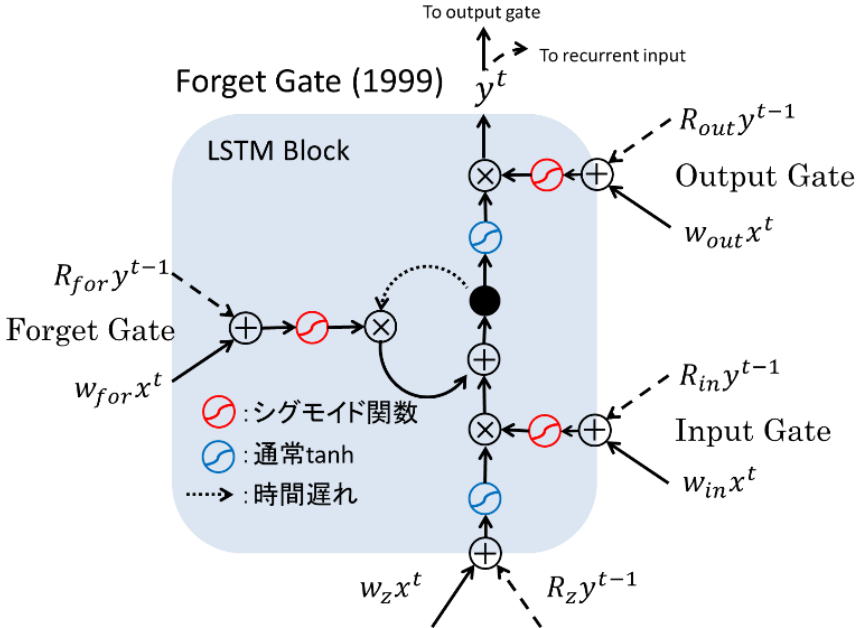
Attention model



途中のエンコード出力も使って重みづけしてやれば、入力系列の情報をより網羅的に考慮してデコードでき、性能があがる

Gated Recurrent Unit (GRU)

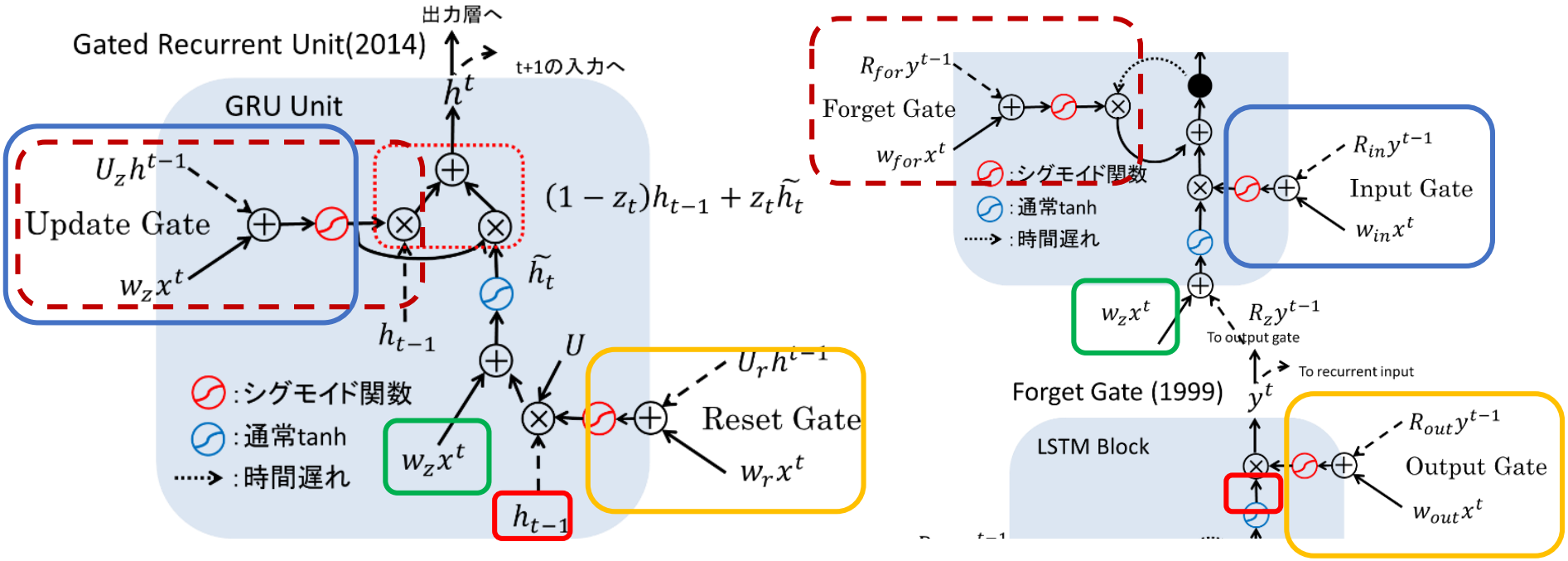




LSTMの亜種
- cellを排除
- ゲートも2つ

の2点が大きな違い
性能的にはタスクによる
機械翻訳、対話で使われてるのをよく見る

余談



ちなみにLSTMをcellの出力のところでぶった切って組み替えるとGRUと非常によく似た形になる（と個人的には思う）が、関係性は不明

cellがなくなることで勾配消失問題が起きる気がするが、性能は悪くない。なぜか？