

Sequence Modeling: Recurrent and Recursive Nets

Chapter 10
楊 森 (Sen Yang) 2019/6/23

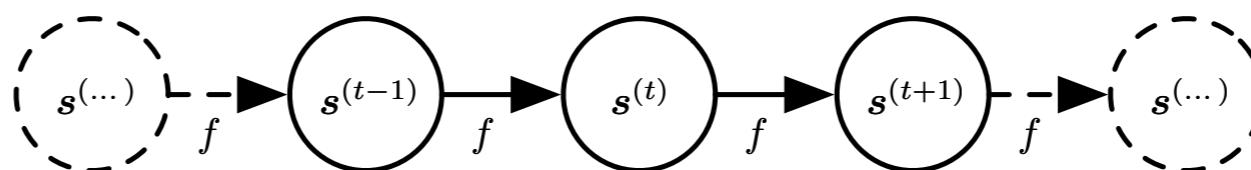
recurrent networks: chain
recursive networks: tree

Unfolding Computational Graphs

伝統的なdynamical systemの式や unfolded computational graph

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

s は状態



伝統的なdynamical system+外部シグナル

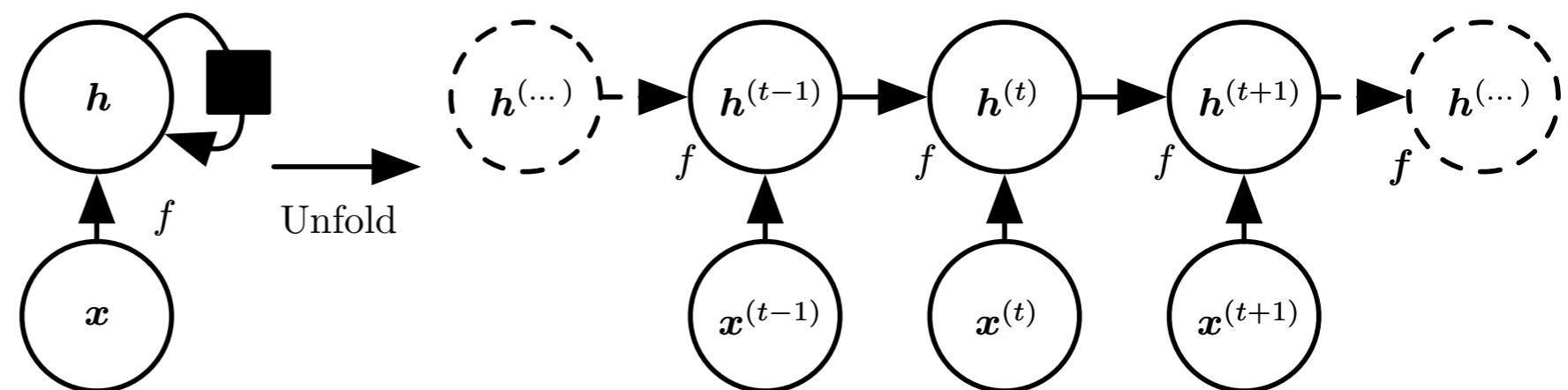
$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$$

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$

h はhidden unit
 θ はshared parameters

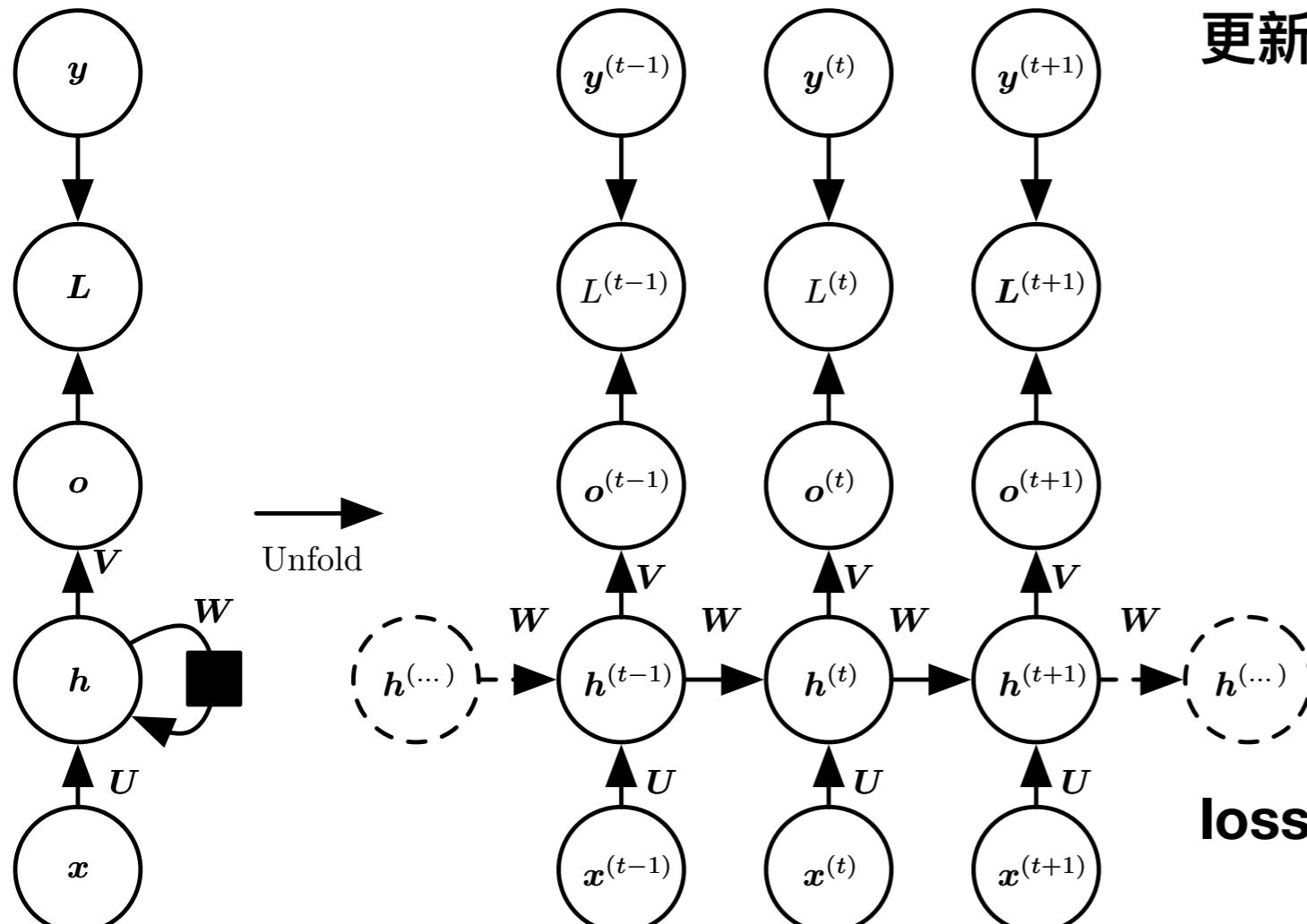
recurrent network用の式やグラフ:

メリット: 全ての time stepと sequence lengthでも対応できる



Recurrent Neural Networks①

基本構造



離散の場合のo:

unnormalized log
probabilities of each
possible value of the
discrete variable

更新式:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

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

$$o^{(t)} = c + Vh^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

学習方法: BPTT
(back-propogation
through time)

loss function:

$$L(\{x^{(1)}, \dots, x^{(\tau)}\}, \{y^{(1)}, \dots, y^{(\tau)}\})$$

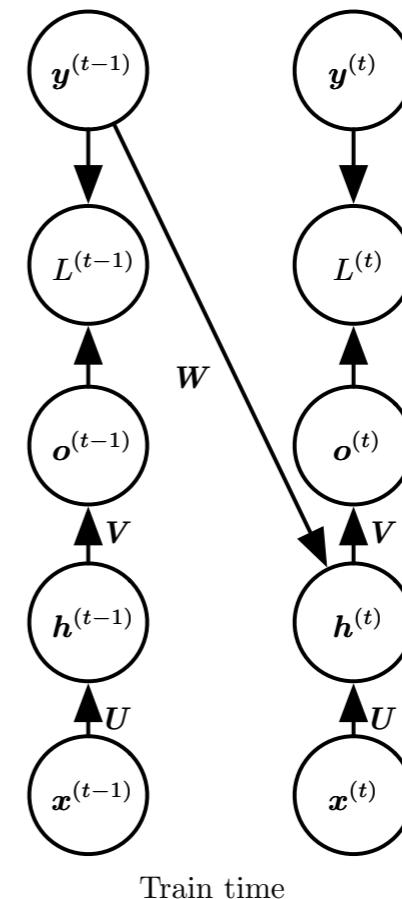
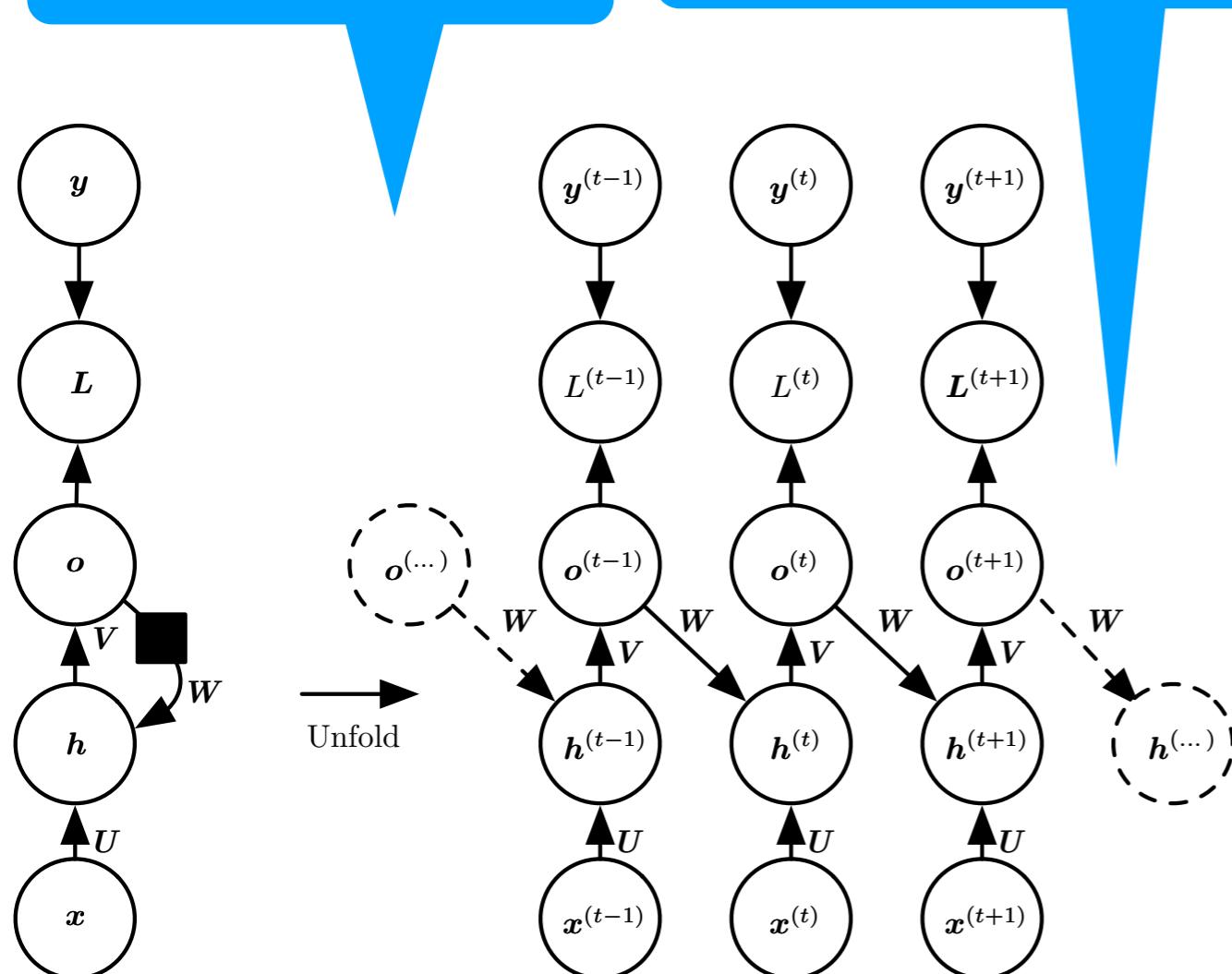
$$= \sum_t L(t) = - \sum_t \log p_{model}(y^{(t)} | \{x^{(1)}, \dots, x^{(t)}\})$$

Recurrent Neural Networks②

ほかの構造

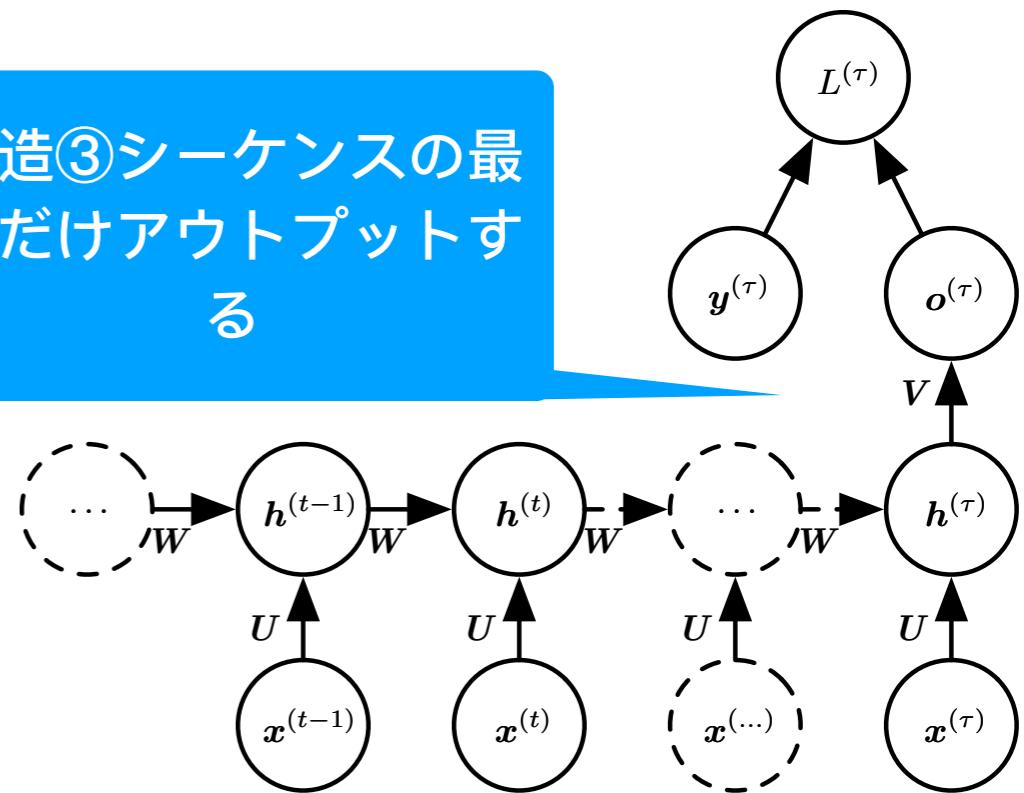
構造②隠れユニットの間のconnectionを切って、アウトプットから feedback connection を追加する

○は過去からの大重要な情報をほとんど持っていないので、弱くなるが、学習しやすい (teacher forcingで並列学習できる)



学習方法:
Teacher Forcing

構造③シーケンスの最後だけアウトプットする



Recurrent Neural Networks③

gradientの計算 (BPTT用)

$$\frac{\partial L}{\partial L^{(t)}} = 1. \quad (10.17)$$

$$(\nabla_{o^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}} \quad (10.18)$$

$$\nabla_{h^{(t)}} L = \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}} \right)^\top (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}} \right)^\top (\nabla_{o^{(t)}} L) \quad (10.20)$$

$$= \mathbf{W}^\top (\nabla_{h^{(t+1)}} L) \text{diag}\left(1 - (h^{(t+1)})^2\right) + \mathbf{V}^\top (\nabla_{o^{(t)}} L), \quad (10.21)$$

$$(tanh(x))' = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}} \right)' = \frac{(e^x - e^{-x})(e^x + e^{-x}) - (e^x + e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2}$$

$$= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - (\tanh(x))^2$$

$$L^{(t)} = -\log(\text{softmax}(o_i^{(t)}))$$

$$= -\log\left(\frac{e^{o_i^{(t)}}}{\sum e^{o^{(t)}}}\right)$$

$$= \log\left(\sum e^{o^{(t)}}\right) - o_i^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \frac{1}{\sum e^{o^{(t)}}} e^{o_i^{(t)}} - 1$$

$$= \text{softmax}(o_i^{(t)}) - 1$$

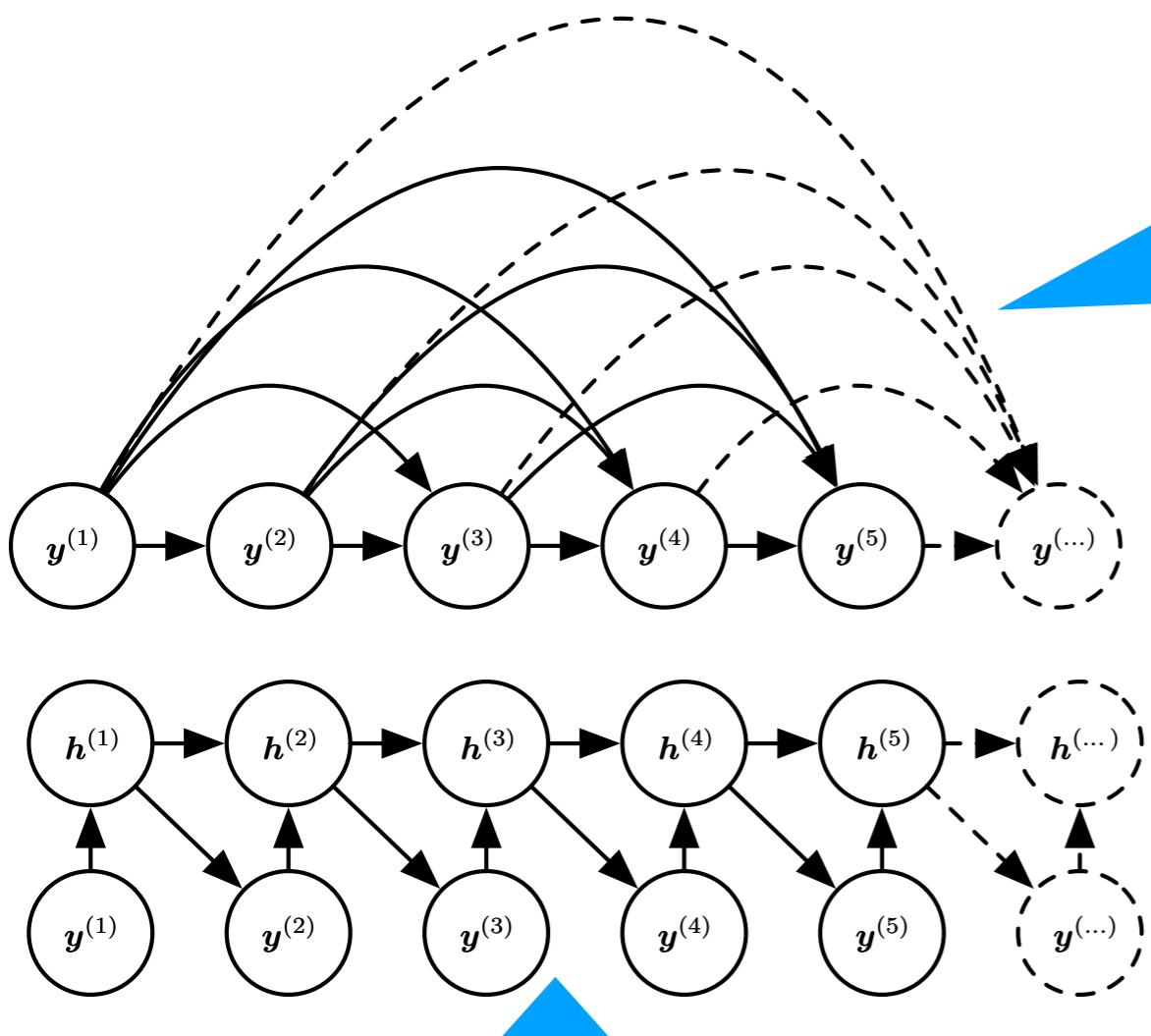
$$= \hat{y}_i^t - 1$$

$$\nabla_c^L, \nabla_b^L, \nabla_V^L, \nabla_W^L, \nabla_U^L$$

は \sum_t 必要、省略

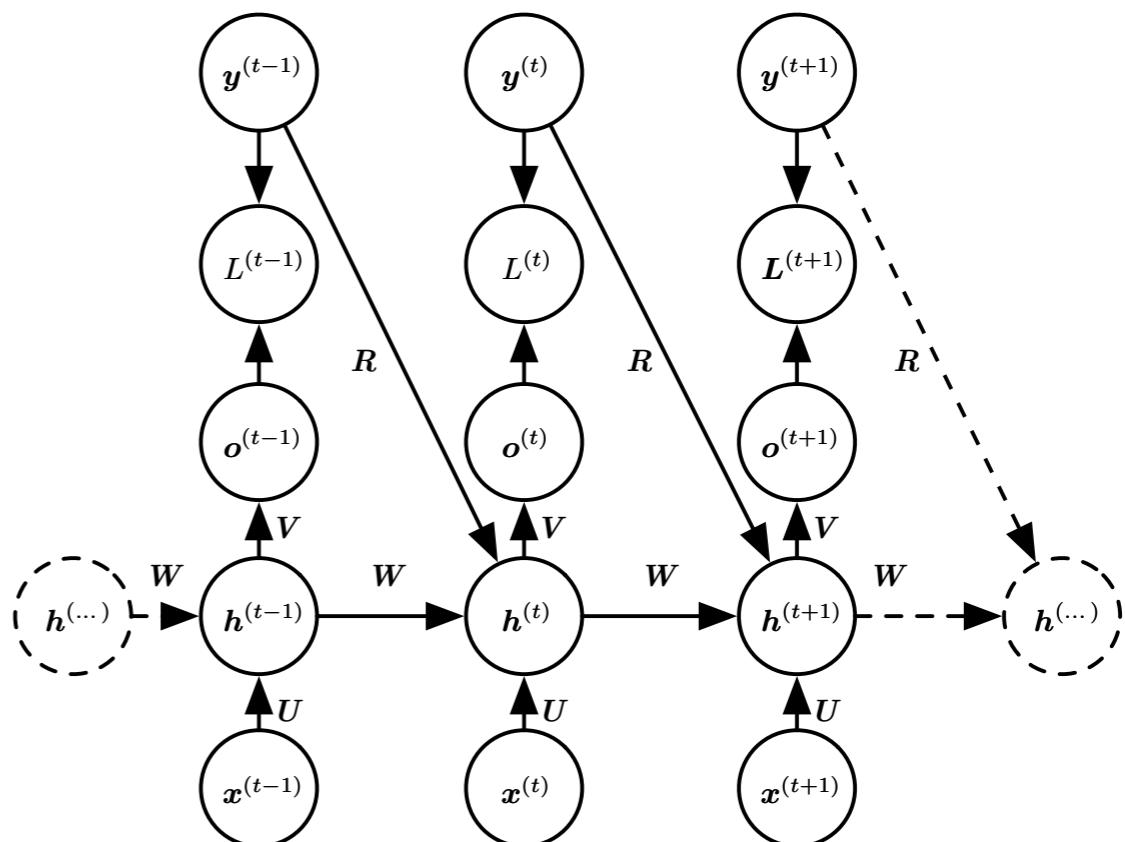
Recurrent Neural Networks④

Directed (有向) Graphical ModelとしてのRNN



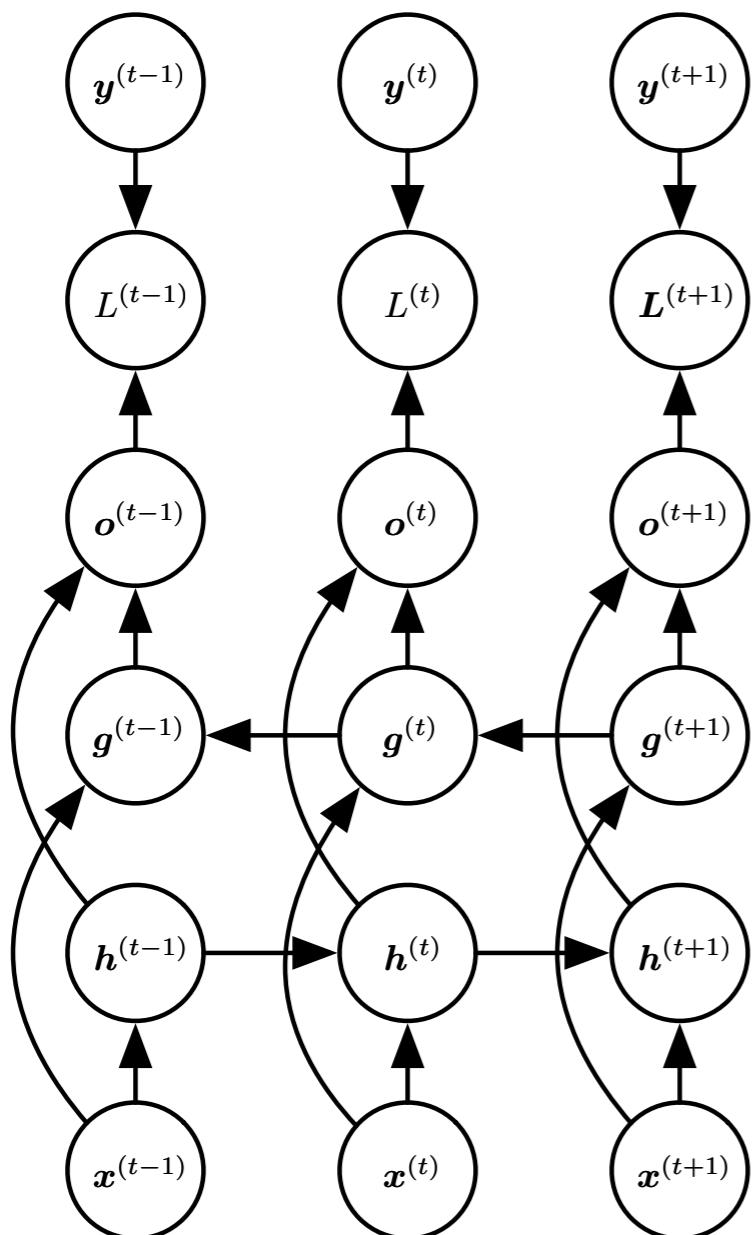
状態変数（隠れユニット）を導入したRNNグラフモデル：全結合グラフモデルのメリットである $y(i) \rightarrow y(j)$ の通路も保留している（ h 経由）し、 $y(t) \rightarrow h(t)$, $h(t) \rightarrow y(t+1)$, $h(t) \rightarrow h(t+1)$ の遷移でパラメタ共有によって効率的になる。

全結合グラフモデル：理想的だが、非効率（全結合層と似ている）パラメタ数はシーケンスの長さと関係ある。



さらに、 x シーケンスインプットを入れる。 $y(t) \rightarrow h(t+1)$ は、 x に対する y のconditional independence仮定を消せる。

Bidirectional (双向) RNNs

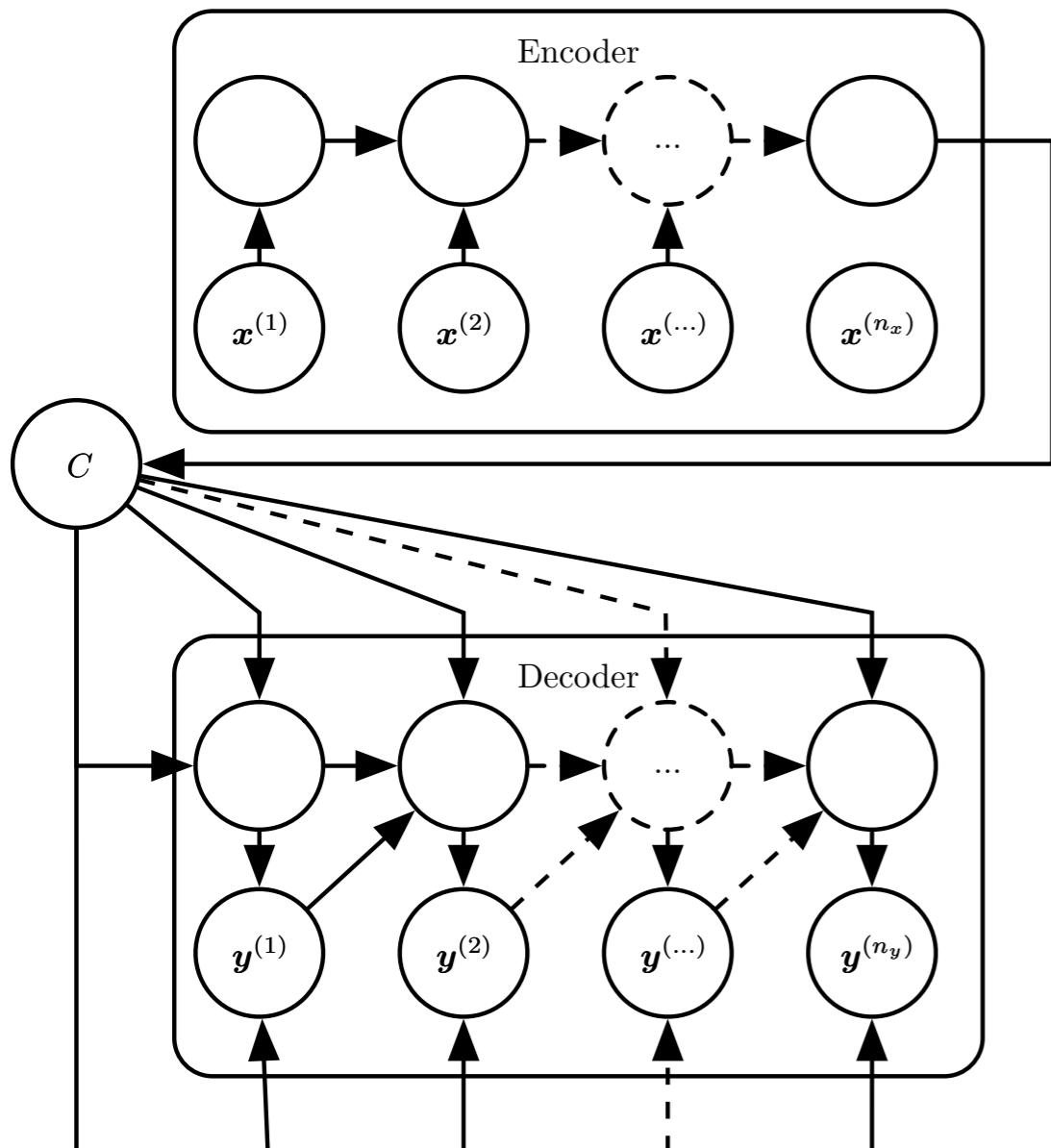


$y(t)$ の予測が将来の x シーケンス（コンテクスト）も依頼する場合。例えば、音声認識。普通のRNNは、 $y(t)$ を予測する時のインプットは、 $x(1), \dots, x(t-1), x(t)$ のみです。もしくはプラス $y(1), \dots, y(t-1)$ です。

前方へ伝播する h 、後方へ伝播する g
recurrence。
 $x(t)$ は、 $h(t)$ や $g(t)$ 両方のインプット。
 $o(t)$ を計算する時、 $h(t)$, $g(t)$ 両方を使う。

自然に画像のような2Dインプットにも適用できる。上、下、左、右の4つ方向。そうすると $o(i,j)$ は各方向のローカル情報をキャプチャできる。しかも遠く離れる所の情報も依頼できる。

Encoder-Decoder Sequence-to-Sequence Architectures



xシーケンス長さとyシーケンス長さが違う場合。普通のRNNは、x,yの長さが同じ。

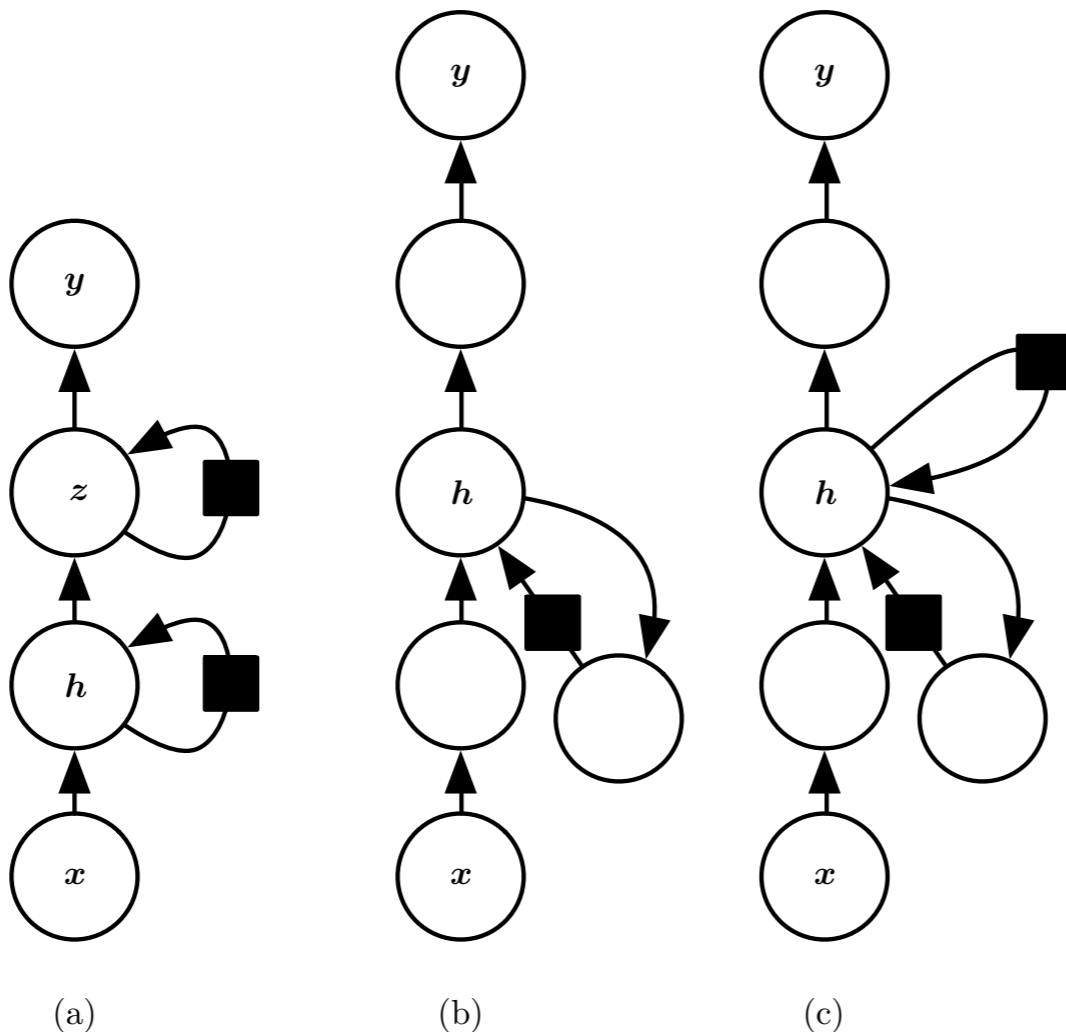
Encoderはxシーケンスから固定サイズのコンテクストCを計算する。（ページ3構造③を参照してください）

DecoderはCからny長さのyシーケンスを計算する。

制限： Cのサイズが小さいと、長いシーケンスをまとめられない。

Deep Recurrent Networks

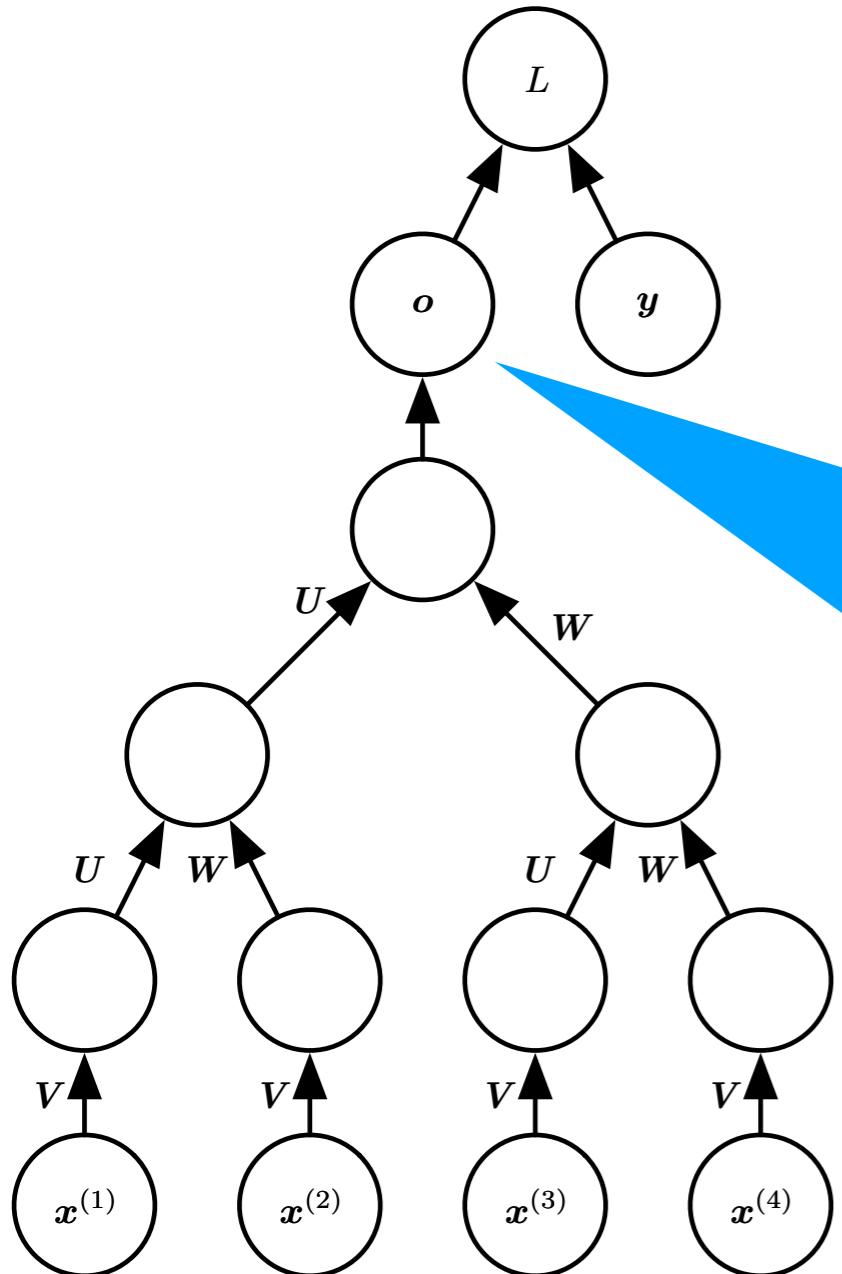
隠れユニット
を増やす。



(b)によってパスが長くなる課題は、Skip Connectionで和らげられる。Skip Connectionで長いパスを経由しない。

input -> hidden state, previous hidden state -> next hidden state, hidden state -> outputにもっと深い計算（など MLP）を入れる。

Recursive (再帰的) Neural Networks



Encoderと同じく、可変長 x に対して、固定サイズのアウトプットを出力する。
Recurrent Networkと同じく、 U, V, W はパラメタ共有です。

Recurrent NetworkのChainと違って
Recursive Networkの形はTree。
メリットは、 x の長さ τ に対する深さは
 $O(\log \tau)$ なので、長期依頼問題を解決でき
るかも。Chainだと、深さは $O(\tau)$ 。

未解決の問題は、どうやっ
てベストなTreeを構造でき
る？

The Challenge of Long-Term Dependencies (Chapter8ページ9最適化挑戦⑤)

$$h^{(t)} = Wh^{(t-1)} = (W^t)^T h^{(0)}$$

固有分解: $W = Q \Lambda Q^T$

$$h^{(t)} = Q^T \Lambda^t Q h^{(0)}$$

長期インタラクションの依頼を学習する時の課題:

- ①勾配が爆発するor消えると、長期依頼を表せない。 $(h(0)$ の情報を保留できない)
- ②爆発しなく、消えなくても、長期インタラクションの勾配が、短期インタラクションの勾配より、**exponentially**小さい。つまり、学習しにくい。 $(h(0)$ の情報を保留できるけど、非常に弱くなってしまう)

ステップ毎に同じWをかけるので、RNNには特に長期依頼問題がある。

ステップ毎に違うWを使うと、勾配爆発や消失問題を対応できる。例えば、スカラー w の場合、 $\Pi w(t)$ の variance が v^* になってほしい場合、 $w(t)$ の variance を $\sqrt[n]{v^*}$ に設定すれば、nステップ後爆発も消失しない。

10.8 Echo State Networksは略ですが、 $h(t-1) \rightarrow h(t)$ の重み W を学習しなくて、設定する RNN です。 W のスペクトル半径を 3 にして、過去情報が前方に伝播されて、 \tanh が $(-1, 1)$ に制限するから爆発しない。Echo State Networks、また Liquid State Machines は、reservoir (貯水池) computing ネットワークです。

Leaky (漏れある) Units and Other Strategies for Multiple Time Scales

長期依頼問題の一つ解決策は、違うtime scaleで動作できるモデルを作ること。
Fine-grained time scaleでディテールをキャプチャーする。Coarse time scaleで離れた過去の情報を流す。

1.Skip Connection

2.Remove Connection

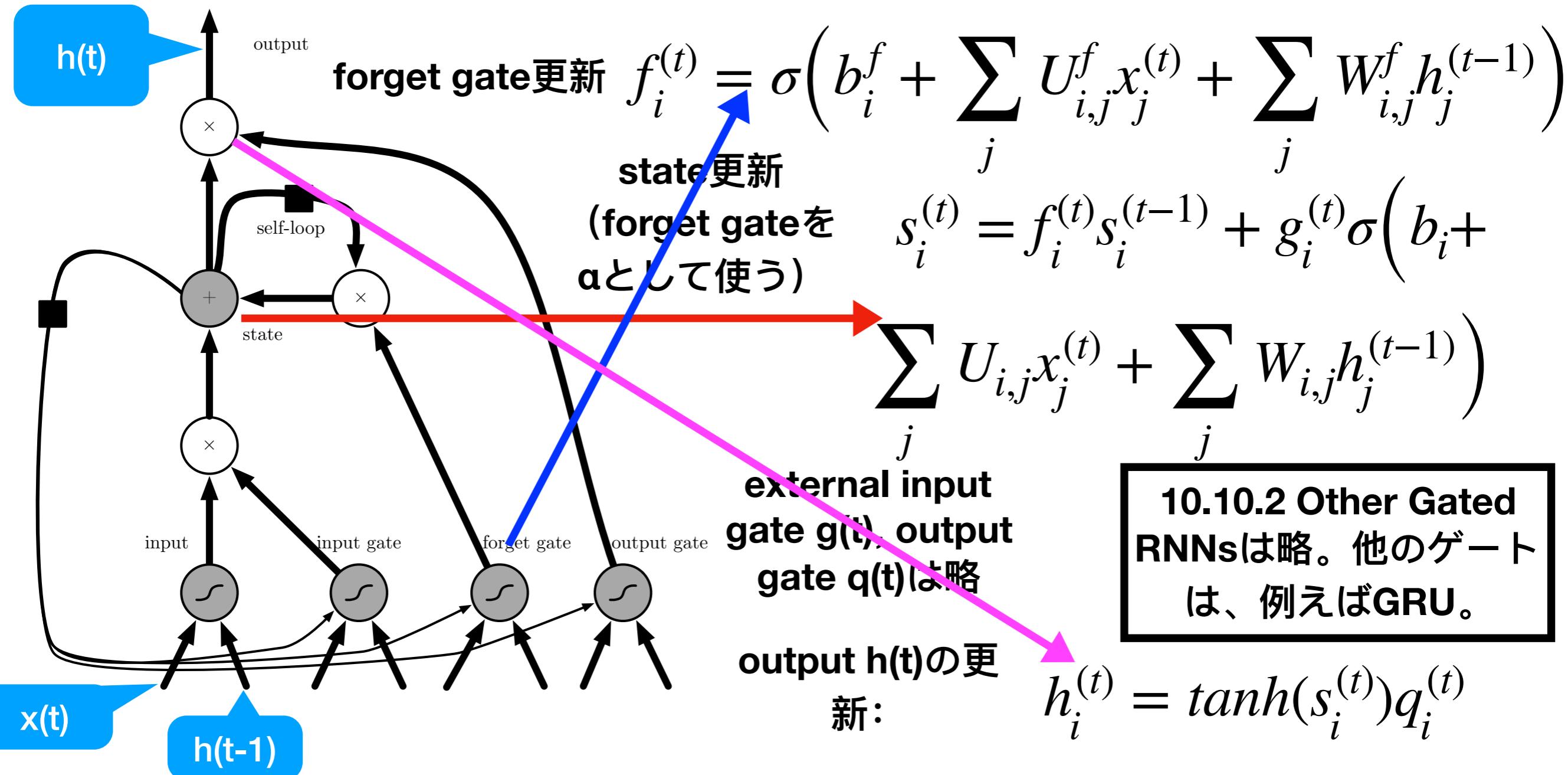
3.Leaky Units

$$\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1 - \alpha)v^{(t)}$$

α が1に近い場合、過去の情報を覚える。
 α が0に近い場合、過去の情報を捨てる。
 α を調整して、どんなtime scaleで過去の情報を覚えるかをコントロールできる。
skip connectionだと、skip lengthを指定しないといけない。

The Long Short-Term Memory

Leaky Unitのディメリットは、 α が固定であること。LSTMは、ステップ毎に α を変えられるようにしたい。（ α もモデルの出力の一部、forget gateという）。微分が消失も爆発もしない時間上のパスを作つて、長期依頼を学習できる。



10.10.2 Other Gated RNNsは略。他のゲートは、例えばGRU。

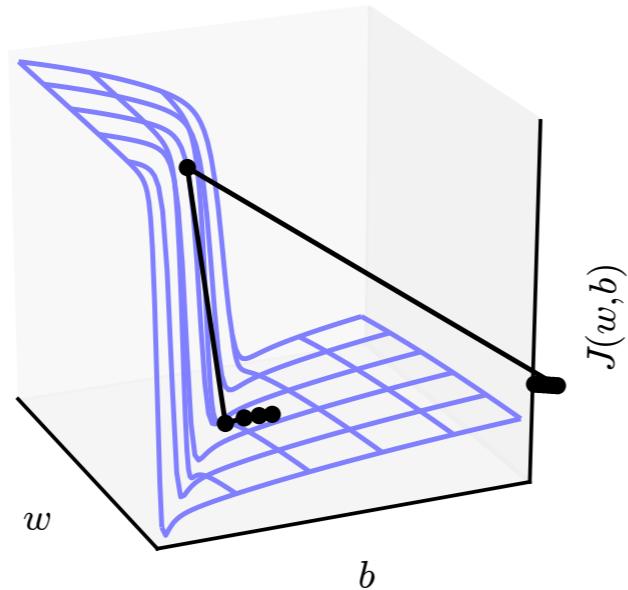
Optimization for Long-Term Dependencies

1. Clipping Gradients

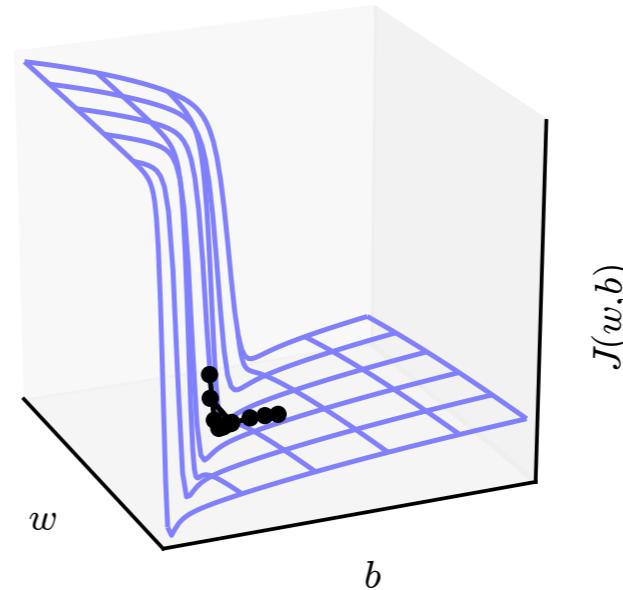
Chapter 8 ページ8で紹介したCliffでGradient Descentが飛んちゃう問題。解決方法: パラメータ更新前ステップを制限する。

例えば、 $\text{if } \|g\| > v, g \leftarrow \frac{gv}{\|g\|}$

Without clipping



With clipping



2. Regularizing to Encourage Information Flow

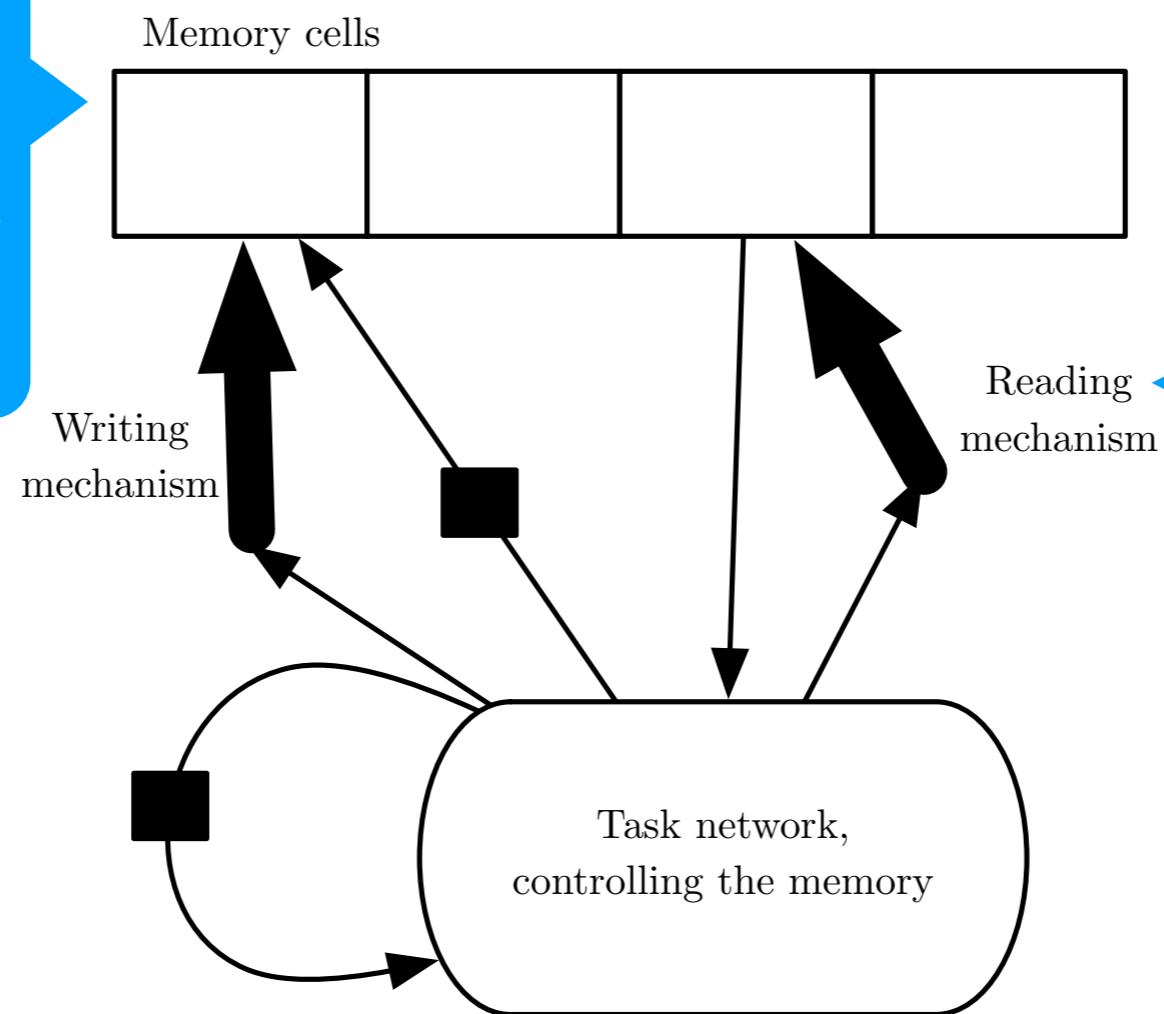
考え方: 正則化で微分 1 のパス (information flow) を誘導する

$$\Omega = \sum_t \left(\frac{\left\| (\nabla_{h^{(t)}} L) \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \right\|}{\|\nabla_{h^{(t)}} L\|} - 1 \right)^2$$

Explicit Memory (顯在メモリ)

Neural Networkは潜在知識（例えば、犬と猫がどう違う）を学習し保存するのが得意、しかし事実（顯在記憶、例えば、犬は動物だ）を覚えるのが苦手。だから、**memory cells**を導入して、時間上に長く情報を保存すれば、長期依頼を学習できる。

一つのMemory cellはベクトルを保存できる。そうすると、内容ベースのアドレッシングもできる。



ネットワークがどこから読み込むか、どこに書き込むかという内部状態をアウトプットする。