

パターン認識と学習

深層学習(3)

管理工学科

篠沢佳久

資料の内容①

- 深層学習(3)
 - 時系列パターンを扱うニューラルネットワーク
 - 時間遅れニューラルネットワーク(TDNN)
 - リカレントネットワーク
 - 単純再帰結合型ネットワーク(エルマンネット)
 - Back propagation through time
 - リカレントネットワークの応用

資料の内容②

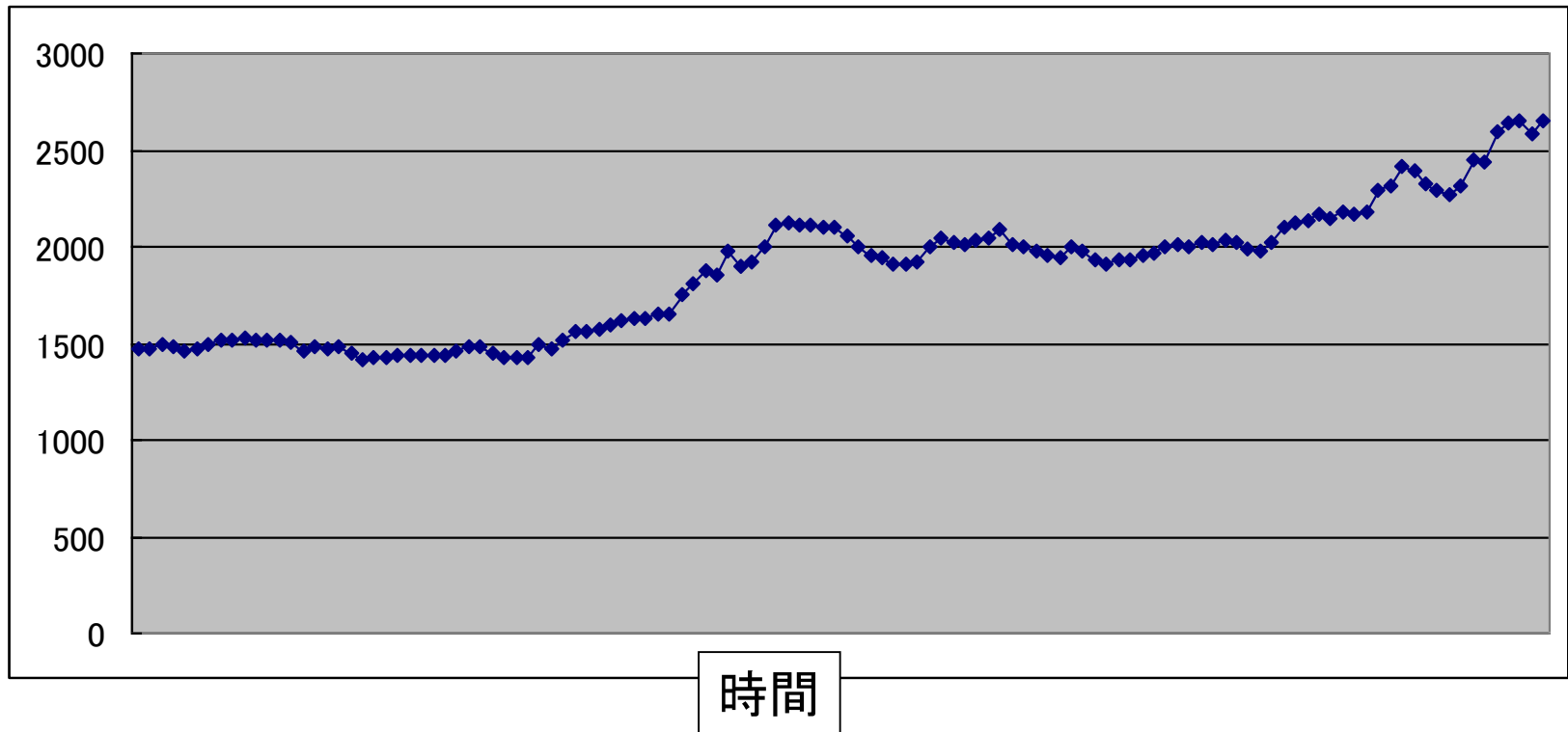
- 実習

- 単純再帰結合型ネットワークによる語系列課題学習

時系列パターン

時系列パターン①

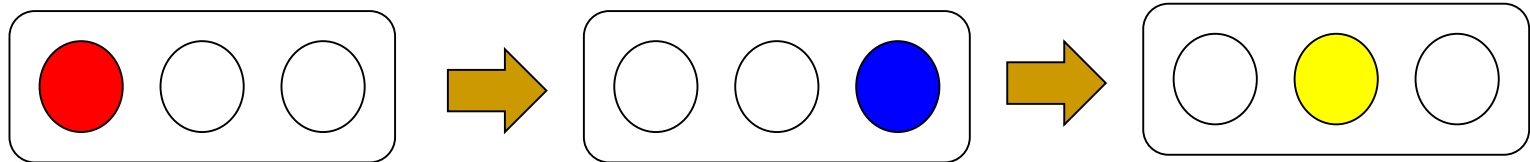
■ 数値データによる時系列パターン



時系列パターン②

■ 視覚による時系列パターン

□ 信号



■ 音声による時系列パターン

□ 言語

□ 楽曲

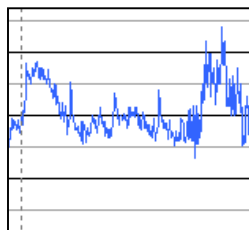
時系列パターンの解析

- 過去のデータより, そのデータがどのクラスに属するのかを予測
 - 音声認識
- 過去のデータより, そのデータの次に現れるデータを予測
 - 株価予測
 - 天気予報

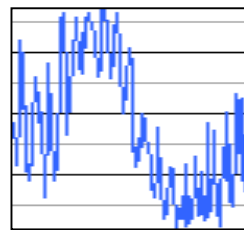
クラスを予測するニューラルネットワーク

時間遅れニューラルネットワーク(TDNN)

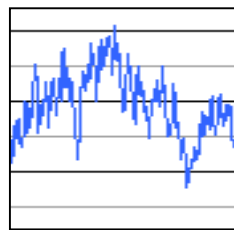
音声認識



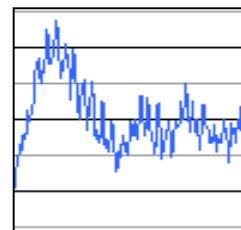
「a」



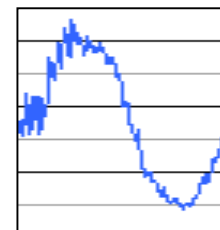
「i」



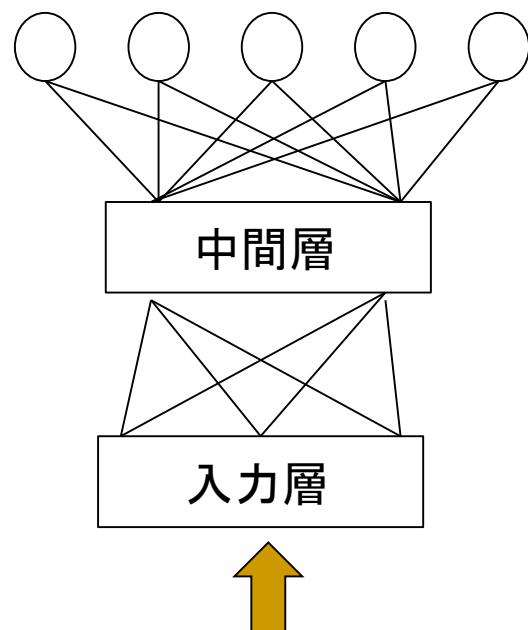
「u」



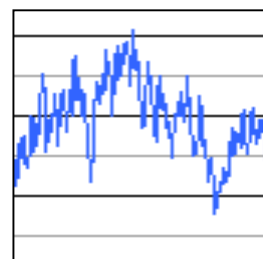
「e」



「o」



出力層 → 5個
中間層 → 任意
入力層 → 音声特徴 $x(0) x(1) \cdots x(N)$



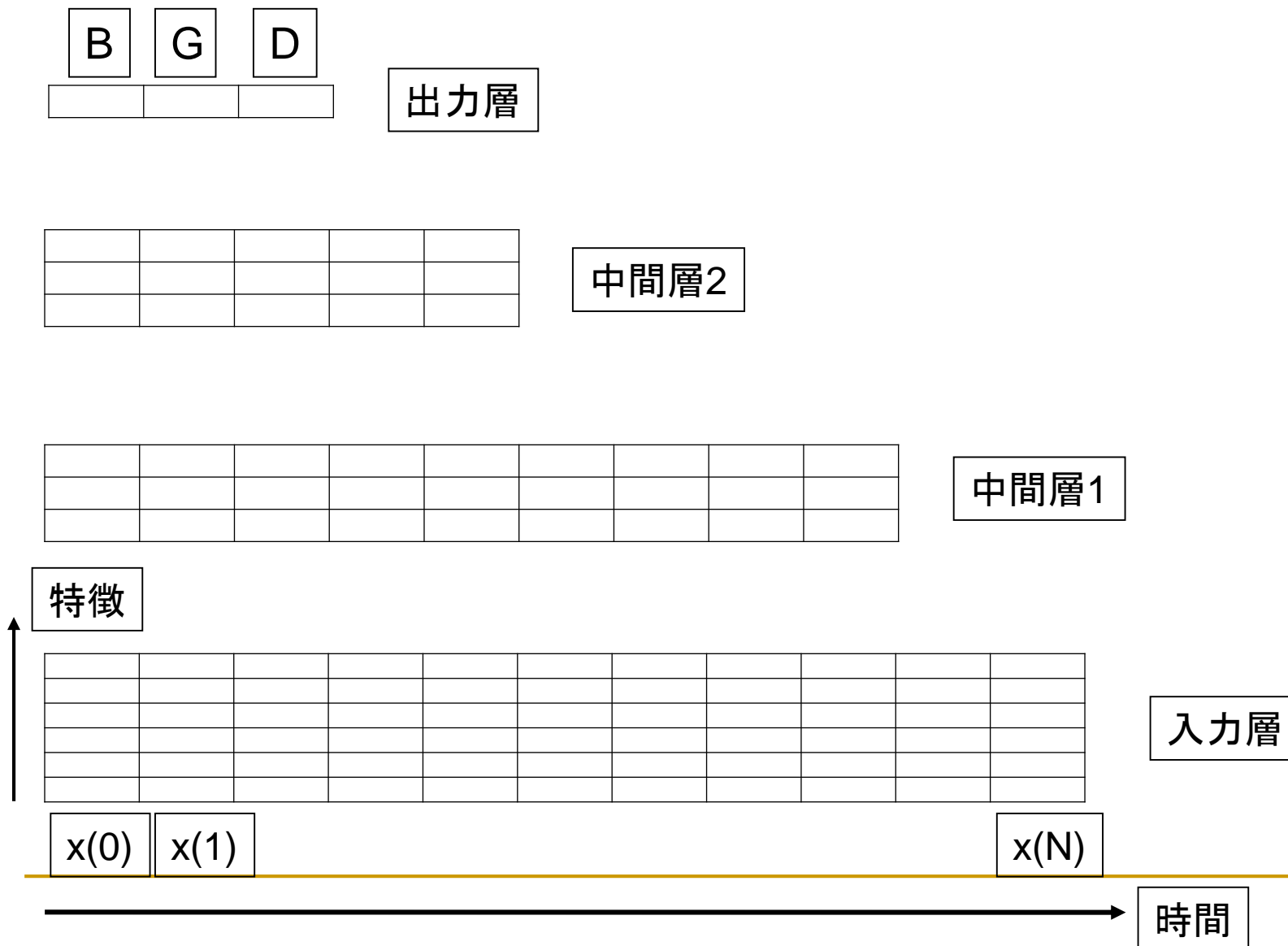
N期に分割

$x(0) x(1) \cdots x(N)$

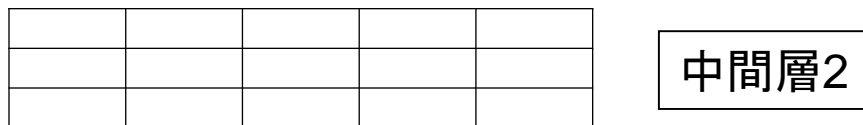
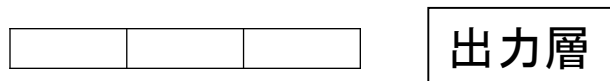
時間遅れニューラルネットワーク

- Time Delay Neural Network (TDNN)
- Alexander Waibel (1989)
- 音声認識 (B, G, D)
 - 入力データは子音と母音 (150ミリ秒)
 - 出力は /B, G, D/
- 階層型ニューラルネットワーク
 - 層間の結合方法を改良 (局所結合)

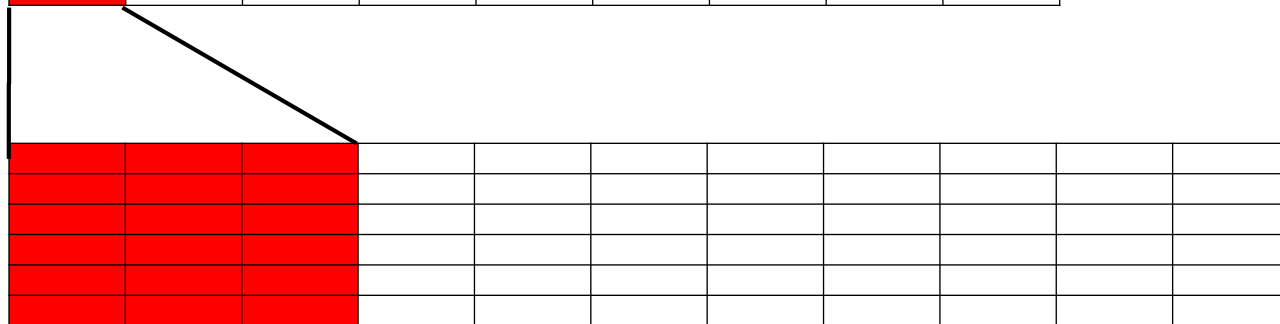
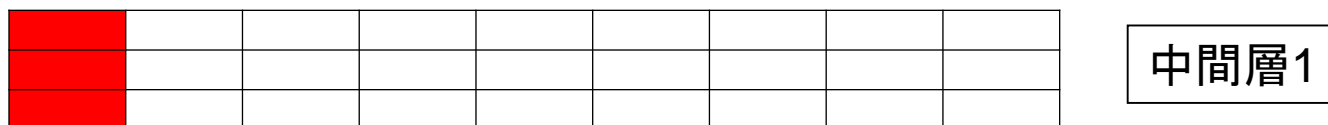
TDNNの構造①



TDNNの構造②



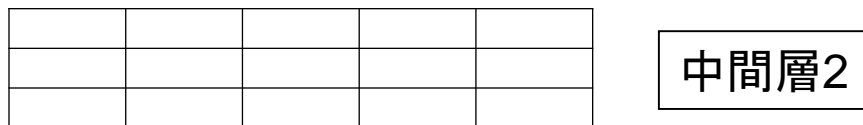
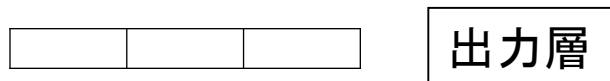
中間層1の赤のニューロンは入力層の赤のニューロンとのみ結合



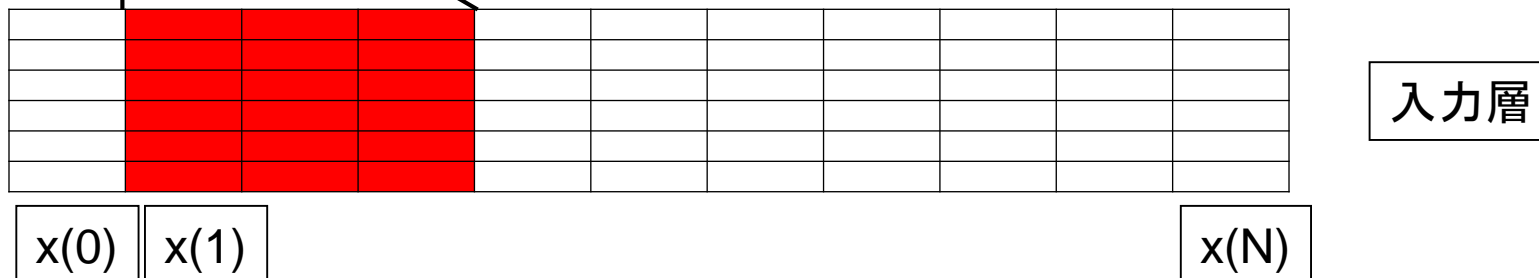
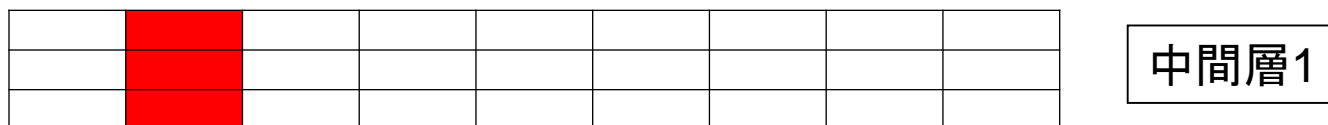
入力層



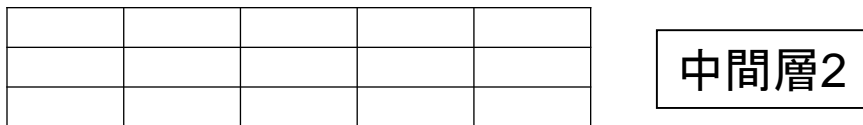
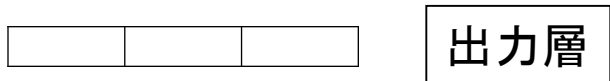
TDNNの構造③



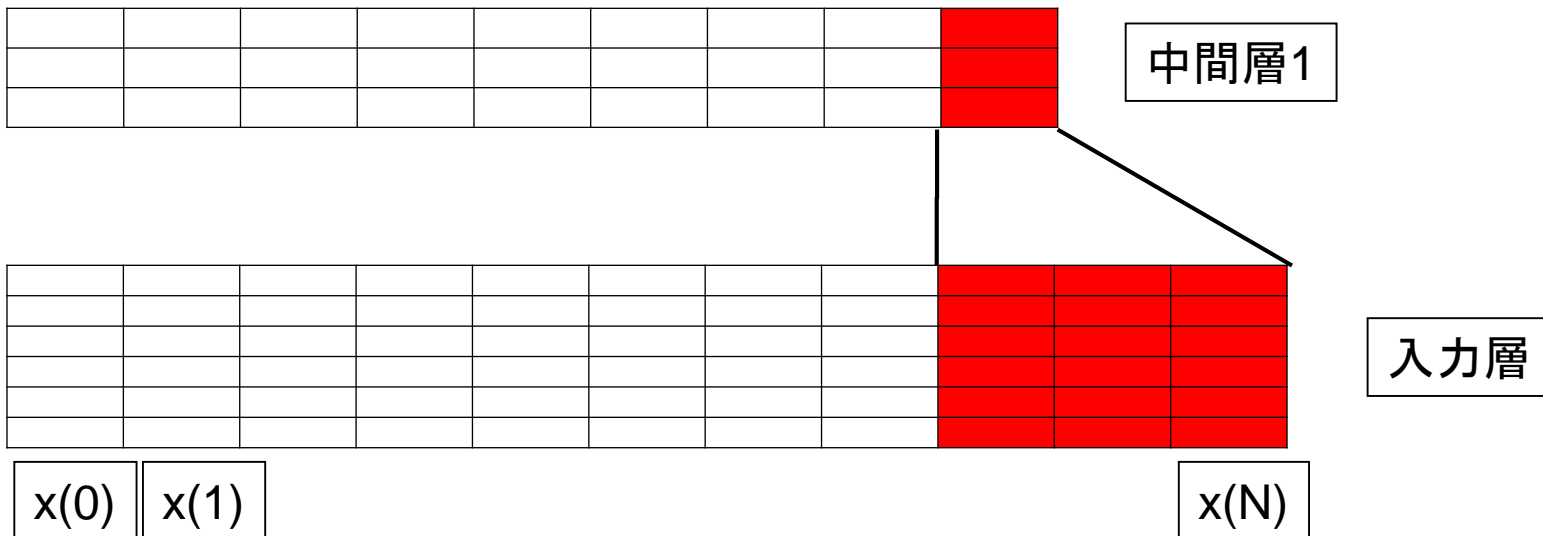
中間層1の赤のニューロンは入力層の赤のニューロンとのみ結合



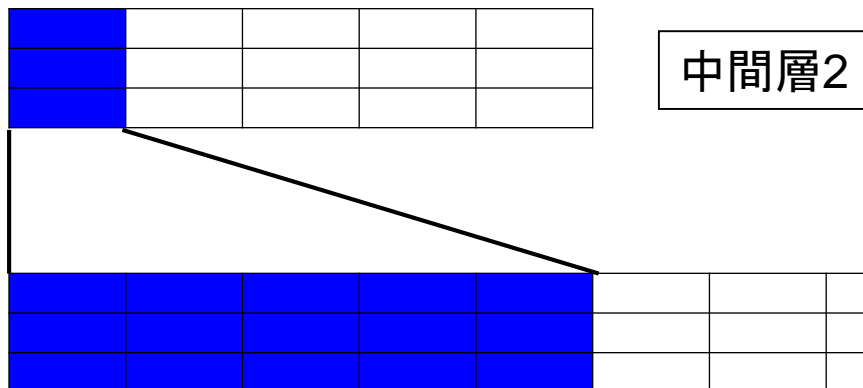
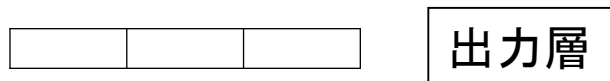
TDNNの構造④



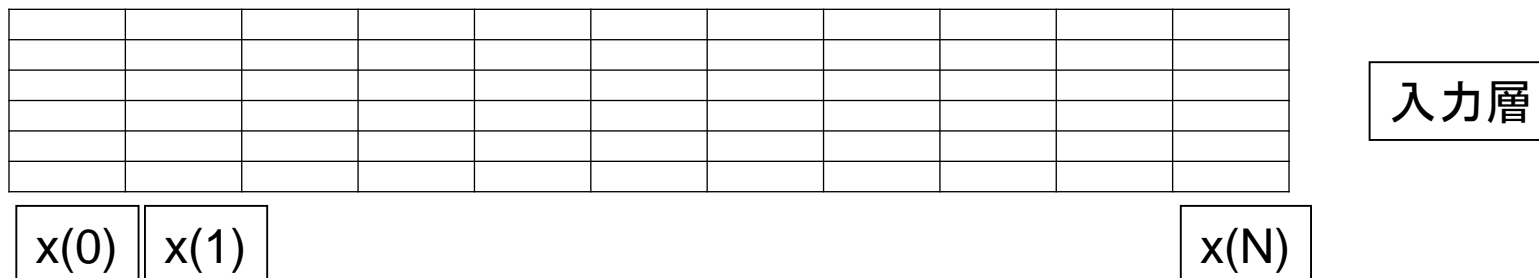
中間層1の赤のニューロンは入力層の赤のニューロンとのみ結合



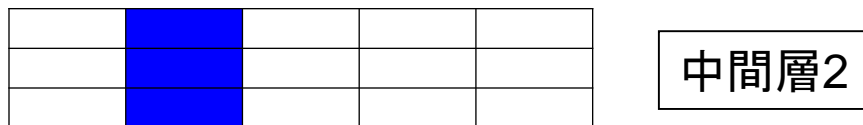
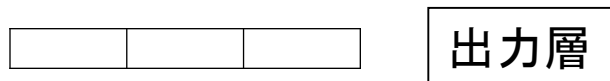
TDNNの構造⑤



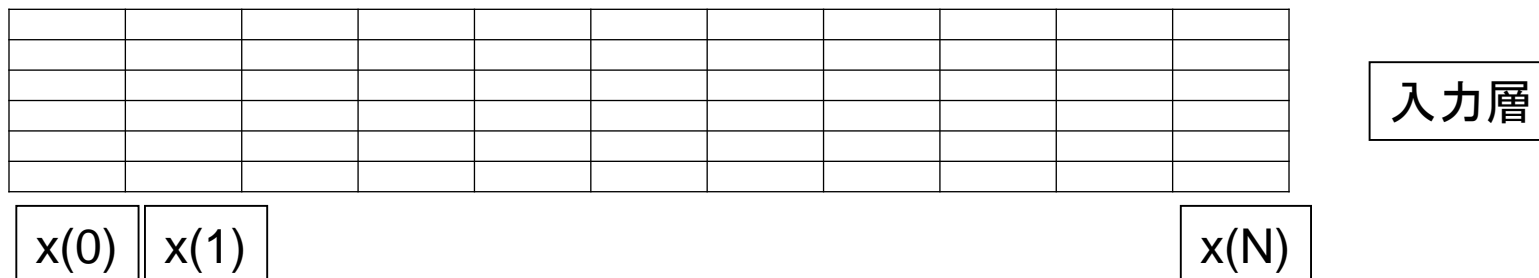
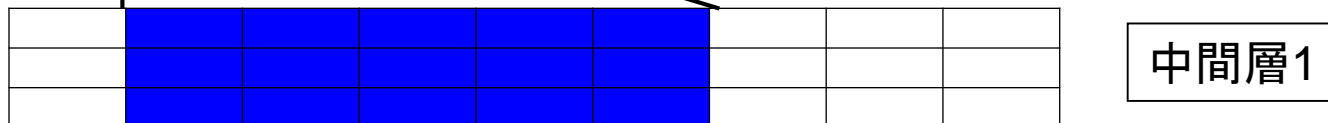
中間層2の青のニューロンは中間層1の青のニューロンとのみ結合



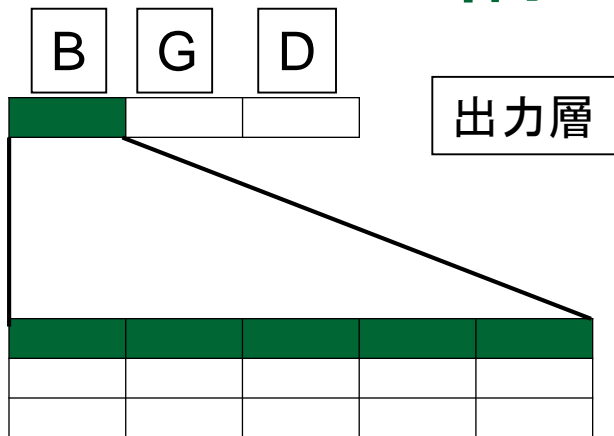
TDNNの構造⑥



中間層2の青のニューロンは中間層1の青のニューロンとのみ結合



TDNNの構造⑦



出力層

出力層の緑のニューロン(B)は中間層2の緑のニューロンとのみ結合

中間層2

中間層1

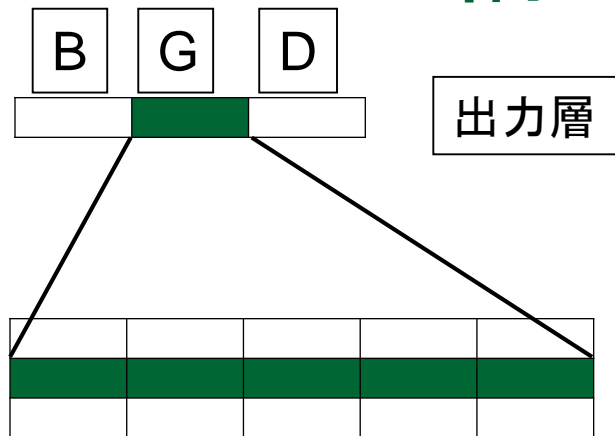
入力層

x(0)

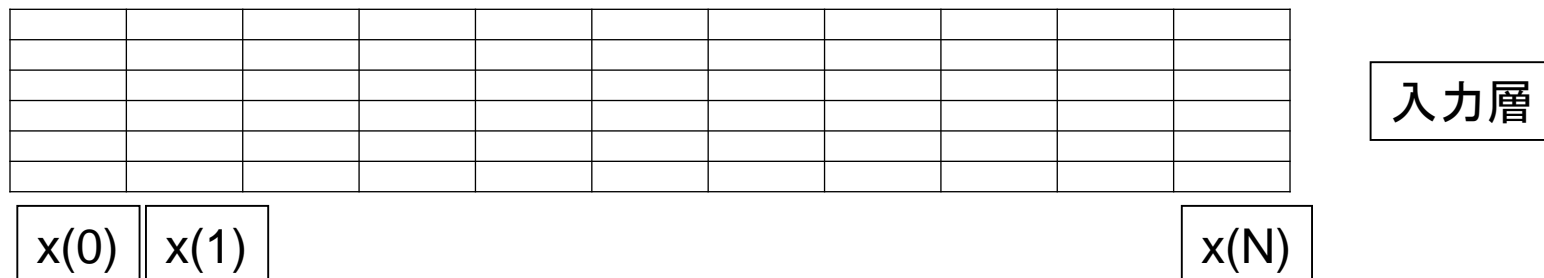
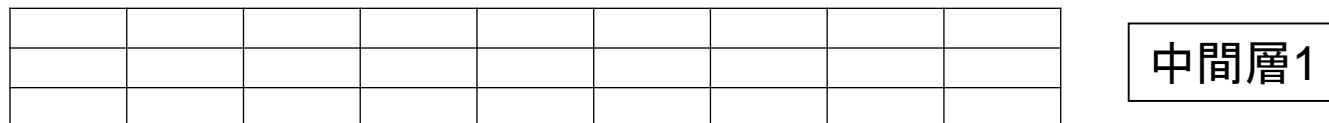
x(1)

x(N)

TDNNの構造⑧



出力層の緑のニューロン(G)は中間層2の緑のニューロンとのみ結合



時間遅れニューラルネットワークの学習

■ 構造上の特徴

- 時間的なずれに対応
- 下層は局所的な特徴を処理
- 上層は大局的な特徴を処理

■ 学習方法

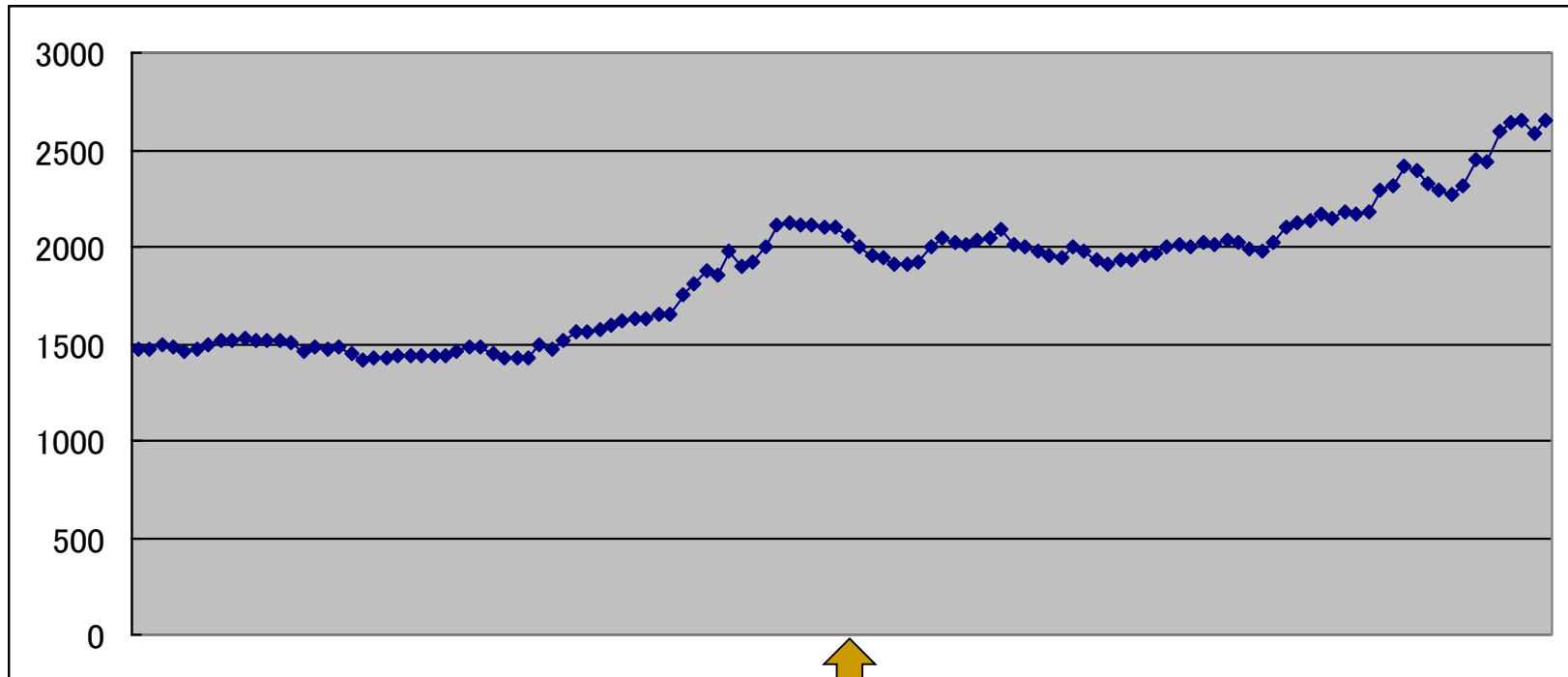
- 誤差逆伝播則が利用可能

次のデータを予測するニューラルネットワーク

リカレントネットワーク

単純再帰結合型ネットワーク(エルマンネット)

時系列データの学習と予測



過去のデータより, どのようにデータが推移してきたかを分析し, 推移過程の規則を見つける(学習)

現在

推移規則が今後も続くと仮定し, 推移規則を用いて, 未来のデータを予測

時系列データの学習①

■ 時系列データ $x_i (i = 0, 1, 2, \dots, N)$

$$x_0, \dots, x_{N-2}, x_{N-1} \rightarrow x_N$$

N-1 期のデータ

時系列データの学習②

■ 時系列データ $x_i (i = 0, 1, 2, \dots, N)$

$$x_0, \dots, x_{m-2}, x_{m-1} \rightarrow x_m$$

m 期のデータ

m: 次数

$$x_1, \dots, x_{m-1}, x_m \rightarrow x_{m+1}$$

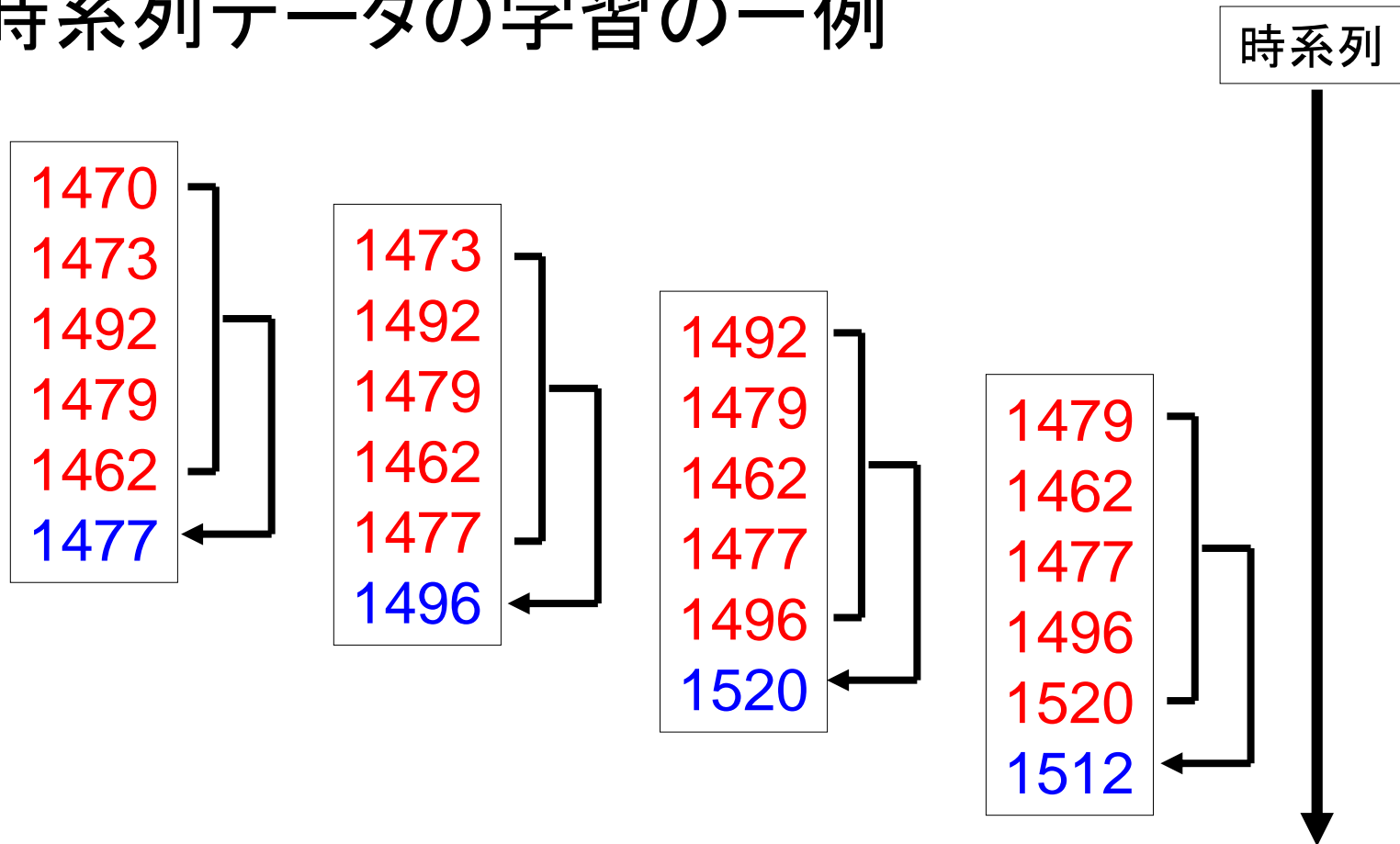
$$x_2, \dots, x_m, x_{m+1} \rightarrow x_{m+2}$$

•
•
•

$$x_{N-m}, \dots, x_{N-2}, x_{N-1} \rightarrow x_N$$

時系列データの学習③

■ 時系列データの学習の一例



過去の情報(赤色)を用いて一期先(青色)を学習

自己回帰モデル①

$$x_0, x_1, \dots, x_{N-1} \rightarrow x_N$$

$$x_{t-N}, \dots, x_{t-2}, x_{t-1} \rightarrow x_t$$

N: 次数

真値

$$\begin{aligned} x_t &= \tilde{a}_1 x_{t-1} + \tilde{a}_2 x_{t-2} + \dots + \tilde{a}_N x_{t-N} \\ &= \sum_{j=1}^N \tilde{a}_j x_{t-j} \end{aligned}$$



推定値

誤差が生じる

$$\begin{aligned} z_t &= a_1 x_{t-1} + a_2 x_{t-2} + \dots + a_N x_{t-N} \\ &= \sum_{j=1}^N a_j x_{t-j} \end{aligned}$$

$$E_t = (x_t - z_t)^2$$

誤差自乗和が最小となる
ように a_j を決める

自己回帰モデル②

$$E_t = (x_t - z_t)^2 = \left(x_t - \sum_{j=1}^N a_j x_{t-j}\right)^2$$
$$= x_t^2 - 2 \sum_{j=1}^N a_j x_t x_{t-j} + \sum_{j=1}^N \sum_{k=1}^N a_j a_k x_{t-j} x_{t-k}$$

最小

$$\frac{\partial E_t}{\partial a_j} = -2x_t x_{t-j} + 2 \sum_{k=1}^N a_k x_{t-j} x_{t-k} = 0$$

$$x_t x_{t-j} = \sum_{k=1}^N a_k x_{t-j} x_{t-k}$$

自己回帰モデル③

$$x_t x_{t-j} = \sum_{k=1}^N a_k x_{t-j} x_{t-k}$$

$$\Phi_j = \sum_{k=1}^N a_k \Phi_{|j-k|}$$

$$\Phi_1 = \sum_{k=1}^N a_k \Phi_{|1-k|} = a_1 \Phi_0 + a_2 \Phi_1 + \cdots + a_N \Phi_{N-1}$$

$$\Phi_2 = \sum_{k=1}^N a_k \Phi_{|2-k|} = a_1 \Phi_1 + a_2 \Phi_0 + \cdots + a_N \Phi_{N-2}$$

\vdots

$$\Phi_N = \sum_{k=1}^N a_k \Phi_{|N-k|} = a_1 \Phi_{N-1} + a_2 \Phi_{N-2} + \cdots + a_N \Phi_0$$

$$x_t^2 = \Phi_0$$

$$x_t x_{t-j} = \Phi_j$$

$$x_{t-j} x_{t-k} = \Phi_{|j-k|}$$

自己回帰モデル④

■ 行列(ベクトル)表記した場合

$$\begin{matrix} \boxed{\mathbf{p}} & & \boxed{\Phi} & & \boxed{\mathbf{a}} \\ \begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ \Phi_N \end{pmatrix} & = & \begin{pmatrix} \Phi_0 & \Phi_1 & \cdots & \Phi_{N-1} \\ \Phi_1 & \Phi_0 & \cdots & \Phi_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_{N-1} & \Phi_{N-2} & \cdots & \Phi_0 \end{pmatrix} & \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} \end{matrix}$$

$$\mathbf{p} = \Phi \mathbf{a}$$



$$\boxed{\mathbf{a} = \Phi^{-1} \mathbf{p}}$$

自己相関行列

自己回帰モデル⑤

■ デルタルールで解いた場合

$$E_t = (x_t - z_t)^2 = (x_t - \sum_{j=1}^N a_j x_{t-j})^2$$

修正方法

$$a_j \leftarrow a_j - \alpha \frac{\partial E_t}{\partial a_j}$$

$$\frac{\partial E_t}{\partial a_j} = -2x_{t-j} \left(x_t - \sum_{k=1}^N a_k x_{t-k} \right)$$

ニューラルネットワークを用いた時系列データの学習①

■ 入力値と教師信号

- 一期前のデータから, 次期のデータを学習(次数が1)
- 0期の入力 $x(0) \rightarrow$ 1期の教師信号 $x(1)$
- 1期の入力 $x(1) \rightarrow$ 2期の教師信号 $x(2)$

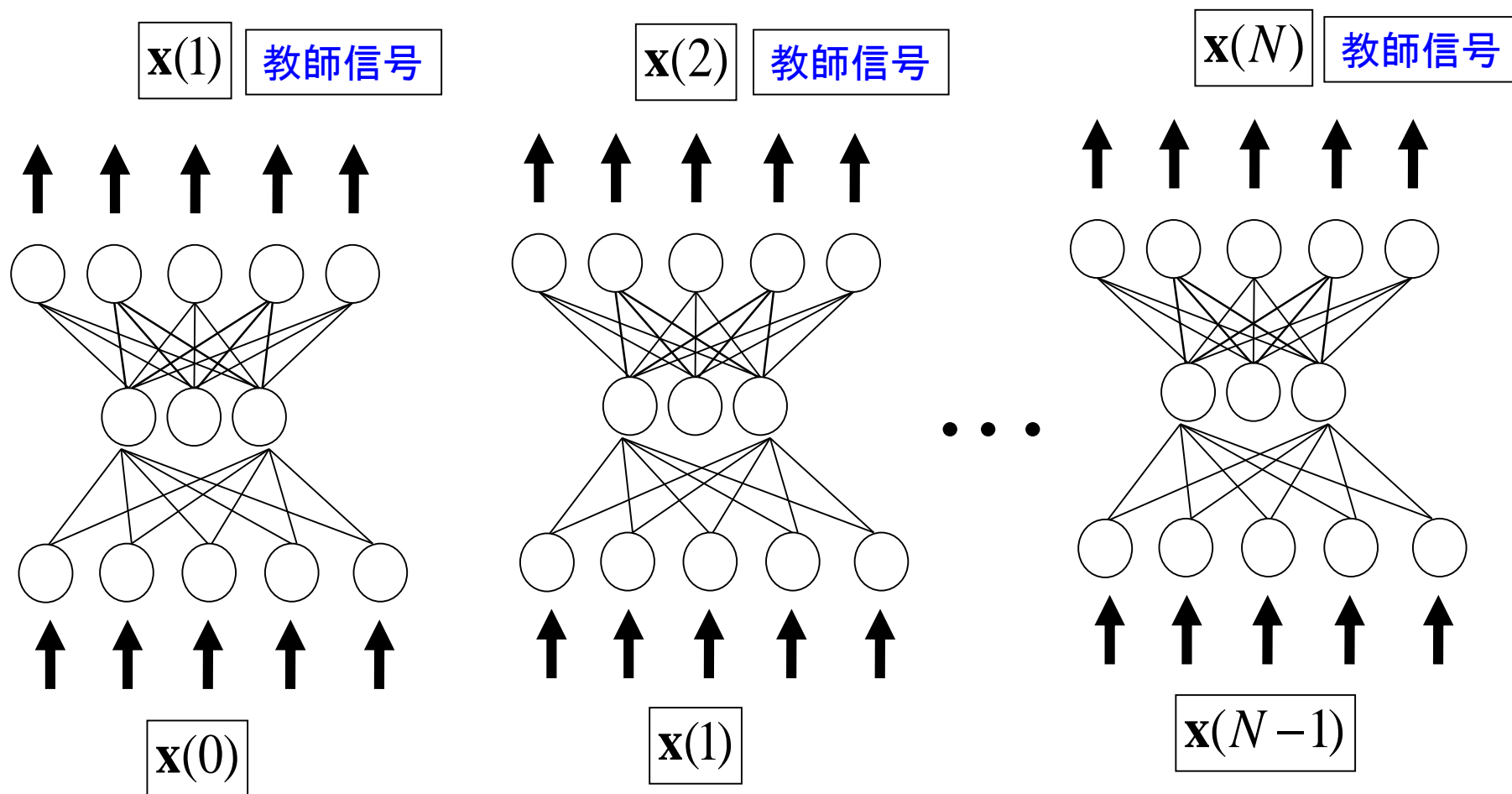
⋮

- N-1期の入力 $x(N-1) \rightarrow$ N期の教師信号 $x(N)$



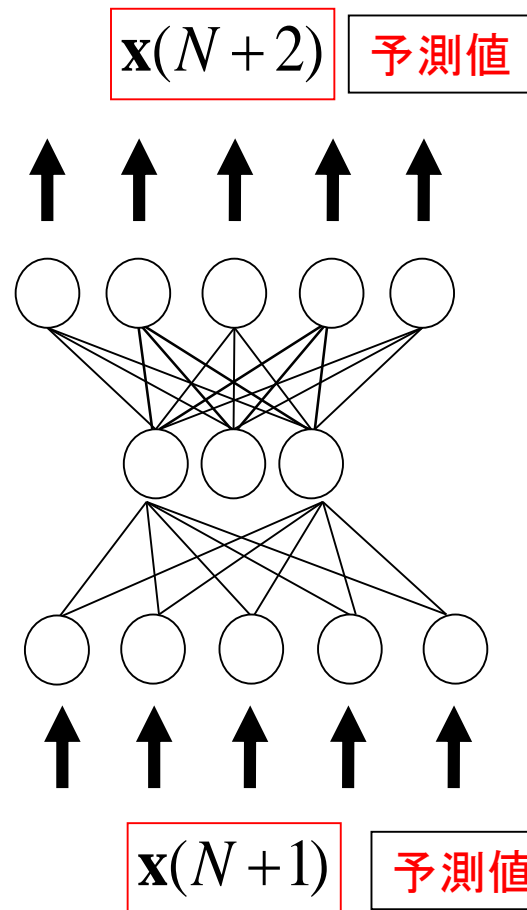
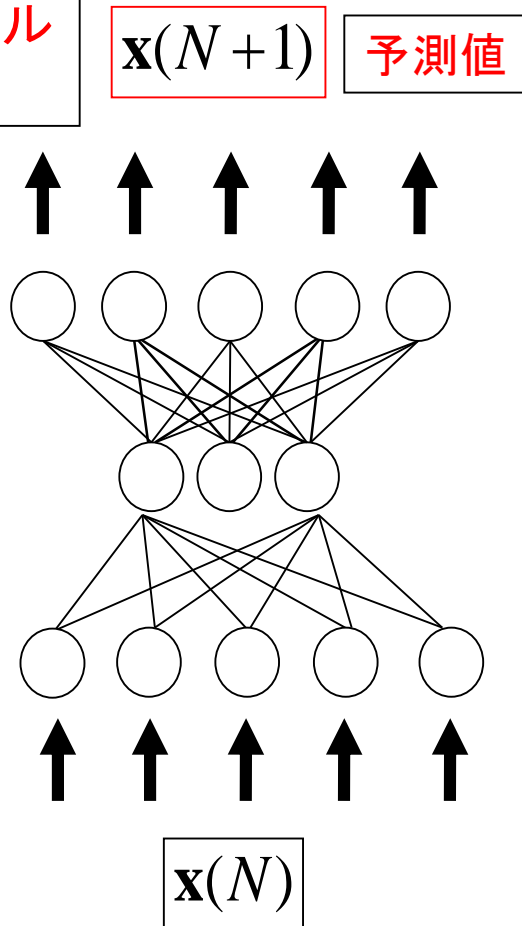
- N期の入力 $x(N) \rightarrow$ N+1期のデータ $x(N+1)$ を予測

ニューラルネットワークを用いた時系列データの学習②



ニューラルネットワークを用いた時系列データの予測

学習後のニューラルネットワーク



ニューラルネットワークを用いた時系列データの学習③

■ 入力値と教師信号

- 一期前のデータから, 次期のデータを学習(次数が1)
- 0期, 1期の入力 $x(0)$, $x(1)$ → 2期の教師信号 $x(2)$
- 1期, 2期の入力 $x(1)$, $x(2)$ → 3期の教師信号 $x(3)$

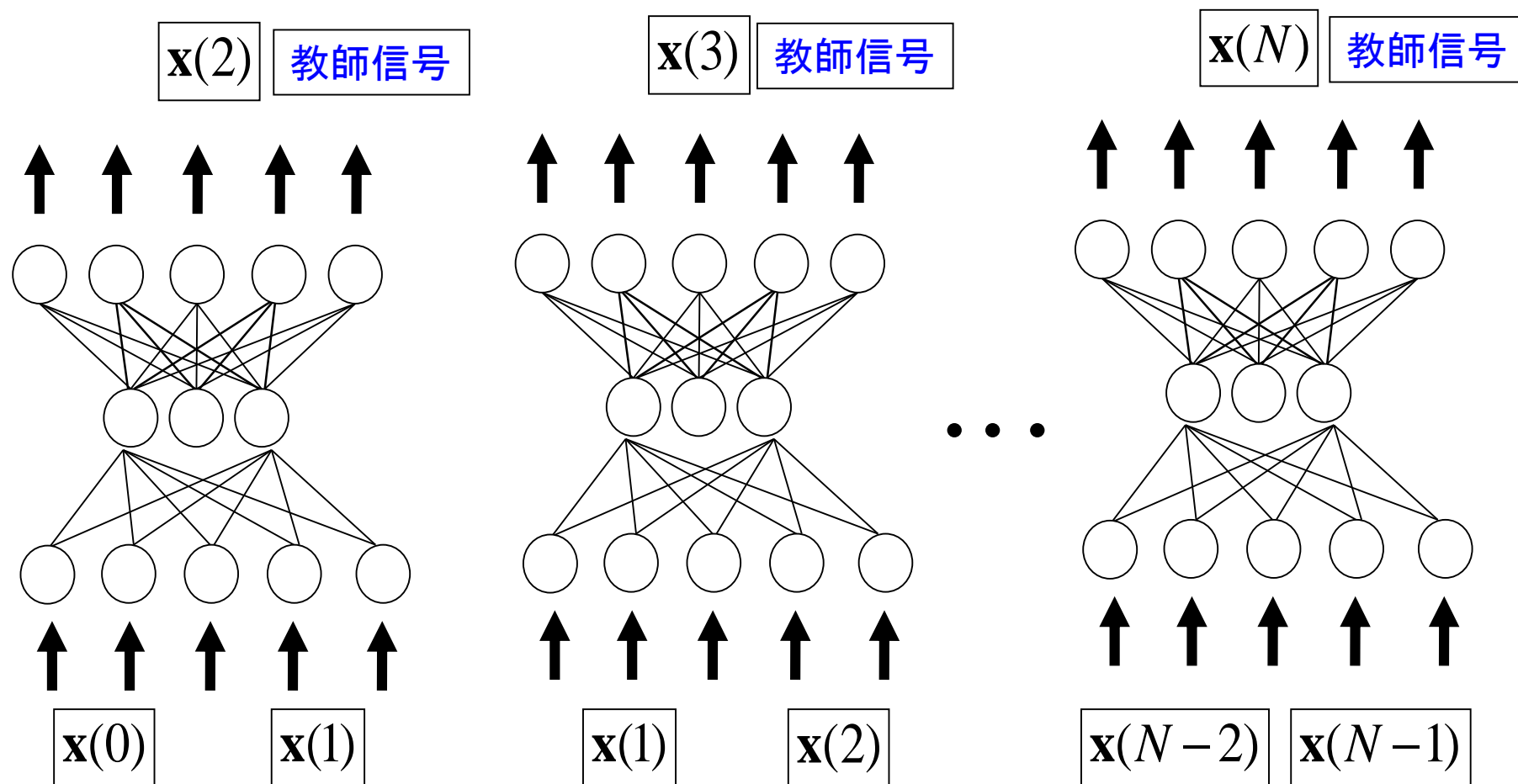
⋮

- N-2期, N-1期の入力 $x(N-2)$, $x(N-1)$ → N期の教師信号 $x(N)$



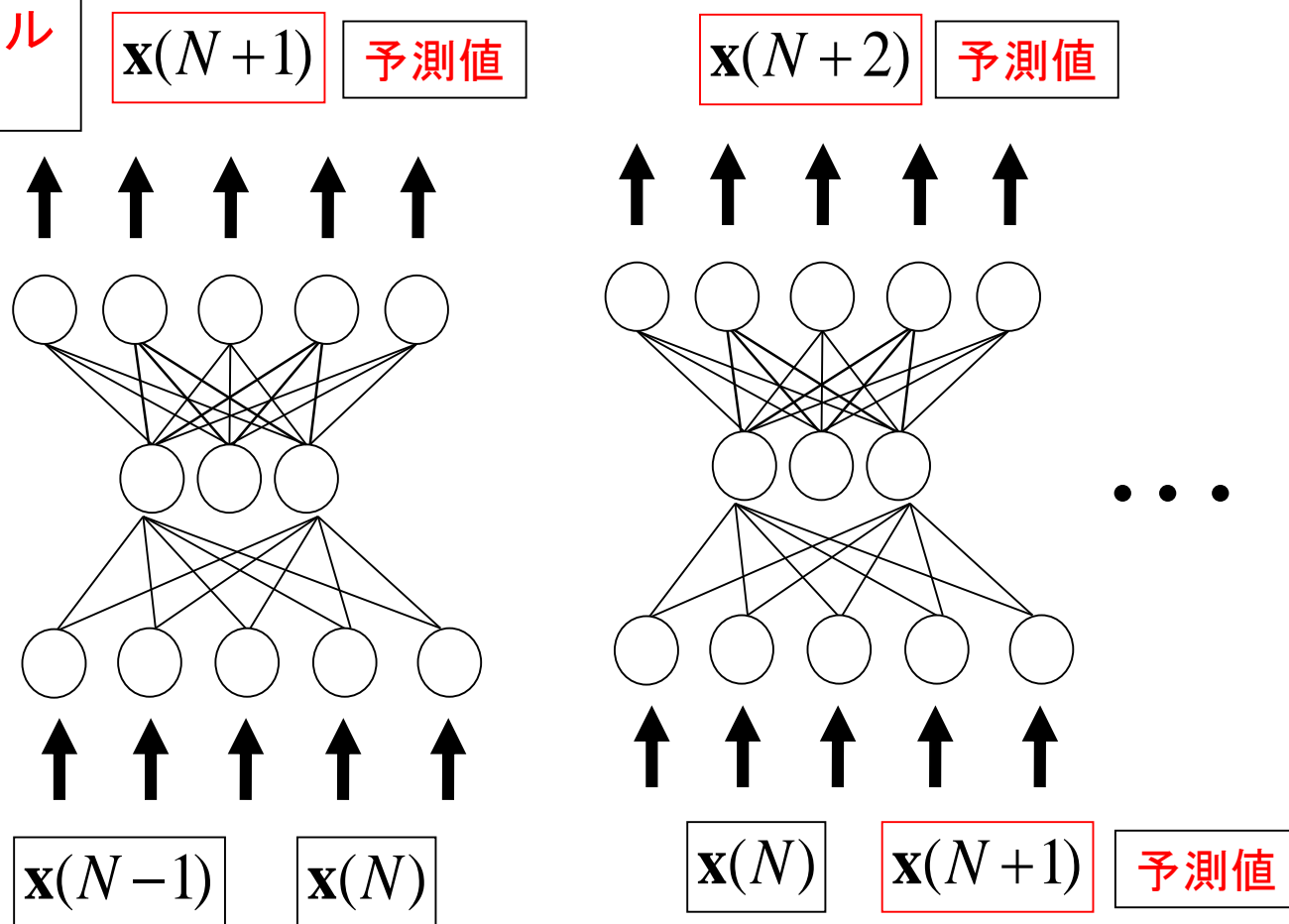
- N期の入力 $x(N-1)$ $x(N)$ → N+1期のデータ $x(N+1)$ を予測

ニューラルネットワークを用いた時系列データの学習④




ニューラルネットワークを用いた時系列データの予測

学習後のニューラルネットワーク

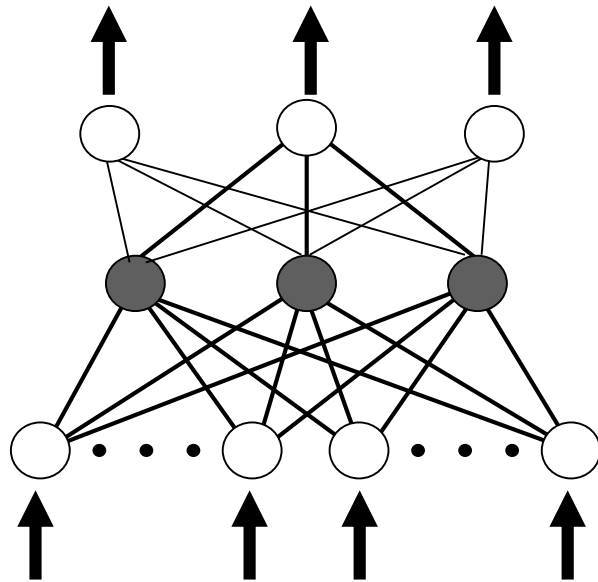


時系列パターンを扱うニューラルネットワーク

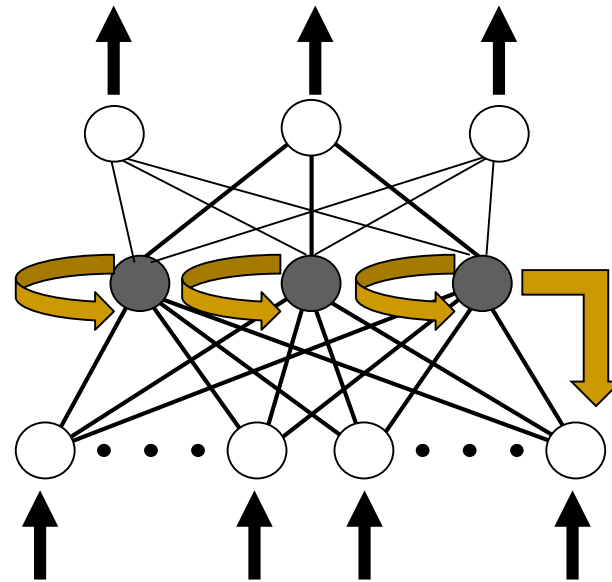
- 時系列パターンを予測する上で,
 - 過去入力したデータの中に、次に現れるデータを予測する上で必要となる情報(規則, パターン)が残っているはず(と仮定)
 - 過去の入力データがネットワーク内部に蓄積されるように構造を改良
- 
- リカレントネットワーク (Recurrent Neural Network, RNN)

リカレントネットワーク

階層型ネットワーク



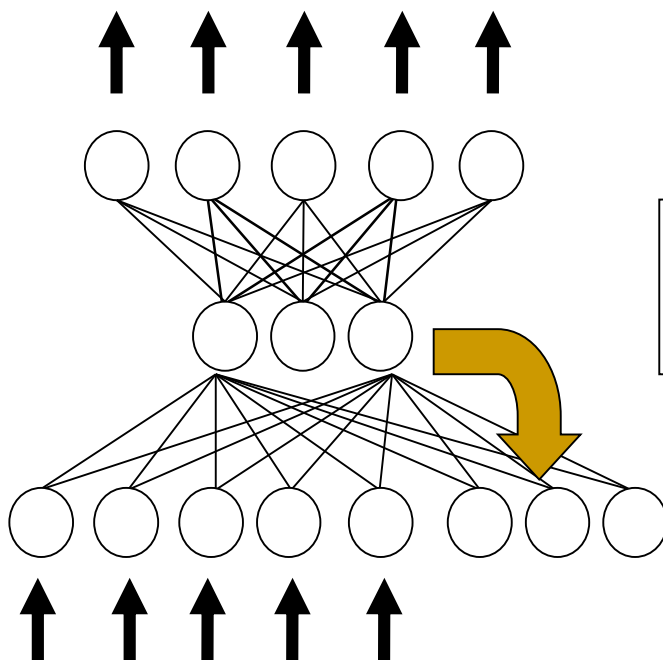
リカレントネットワーク



前の層, あるいは自分自身に対してフィードバックを持つネットワーク

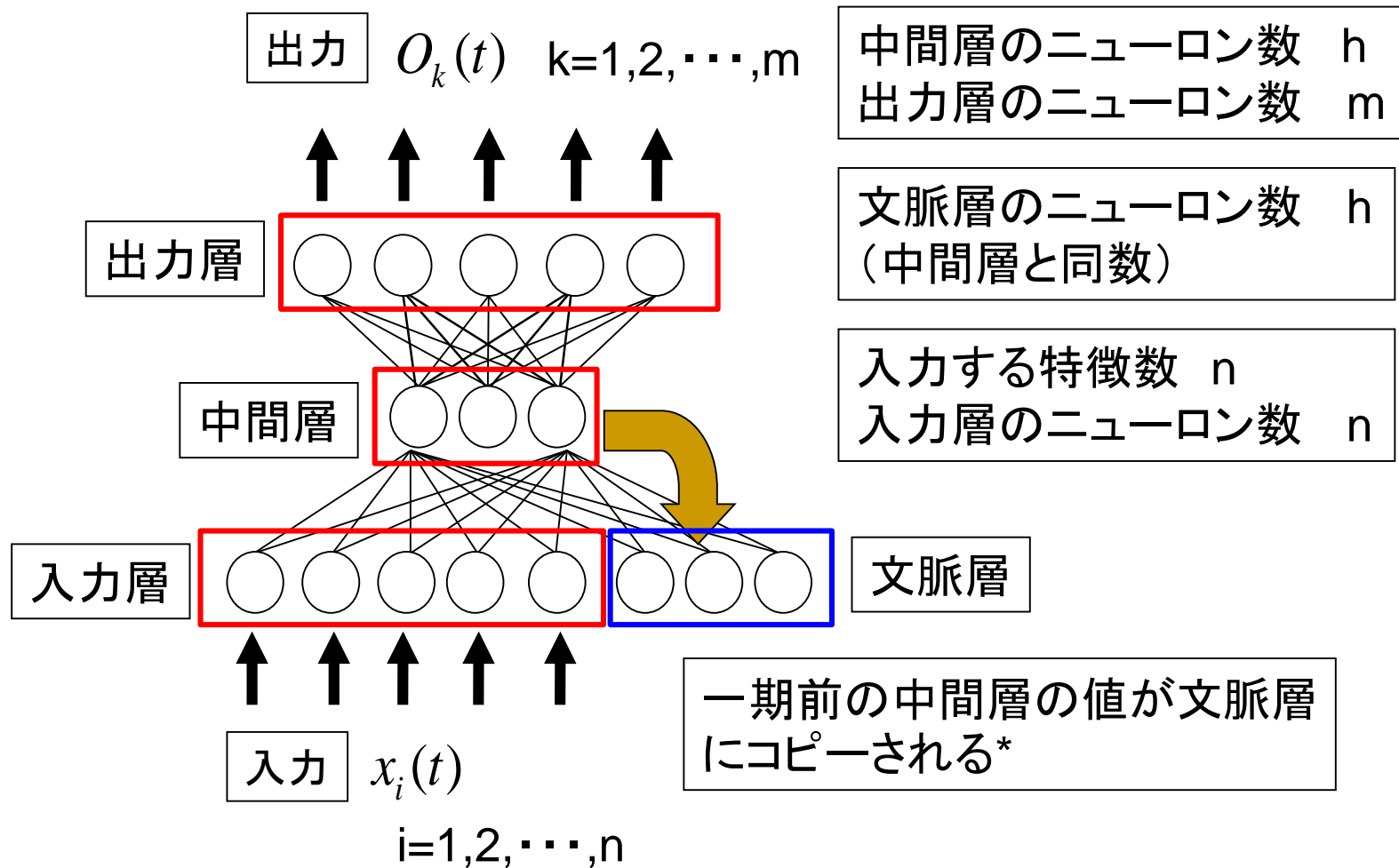
単純再帰結合型ネットワーク

- Jeffrey L. Elman (1990)
- Simple Recurrent Network (略してSRN)
- エルマンネットとも呼ばれる



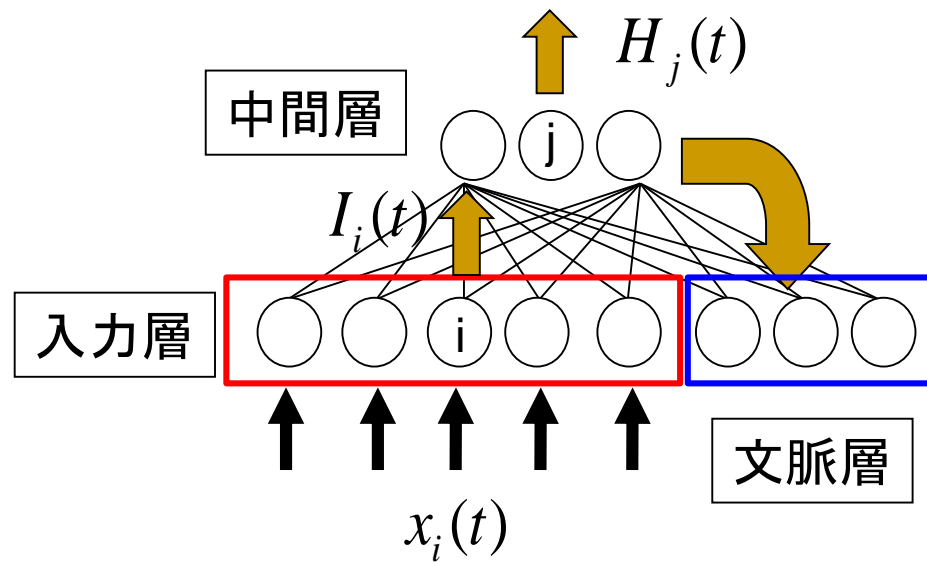
中間層からのフィード
バックを持つ

エルマンネットの構造



*開始時は文脈層の値は0とするのが一般的です

エルマンネットの動作①



中間層の出力

$$H_j(t) = f\left(\sum_{i=1}^{n+h} W_{ji} I_i(t) - \theta_j\right)$$

θ_j 中間層のニューロン j のしきい値

W_{ji}
中間層のニューロン j と入力層のニューロン i との結合係数

$i=1 \sim n$ 入力層からの出力

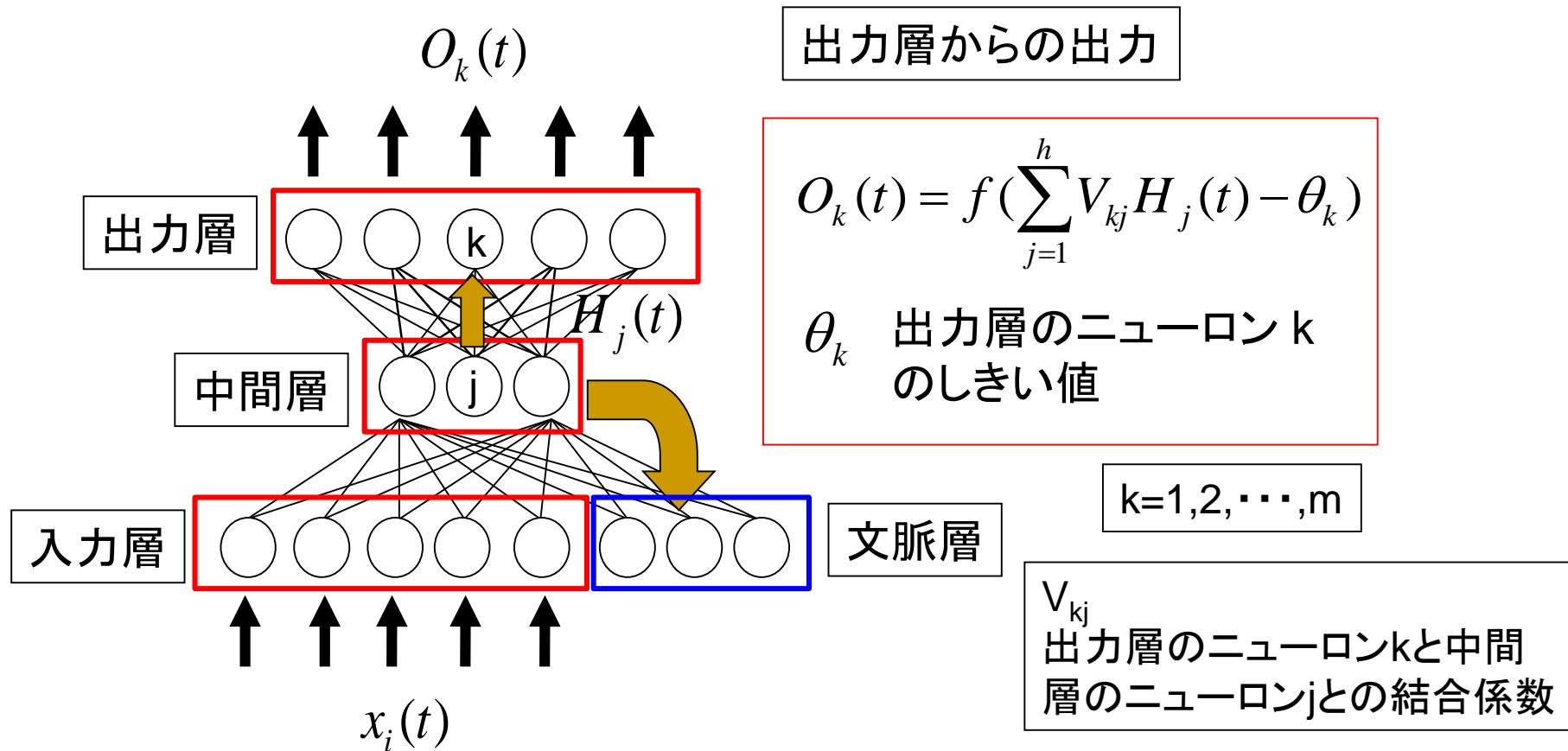
$$I_i(t) = x_i(t)$$

$i=n+1 \sim n+h$ 文脈層からの出力

$$I_i(t) = H_j(t-1)$$

一期前の中間層の値
 $j=1 \sim h$

エルマンネットの動作②



エルマンネットの学習①

■ 入力値と教師信号

- (例) 一期前のデータから, 次期のデータを学習

- 0期の入力 $\rightarrow \mathbf{x}(0)$ 1期の教師信号 $\rightarrow \mathbf{x}(1)$

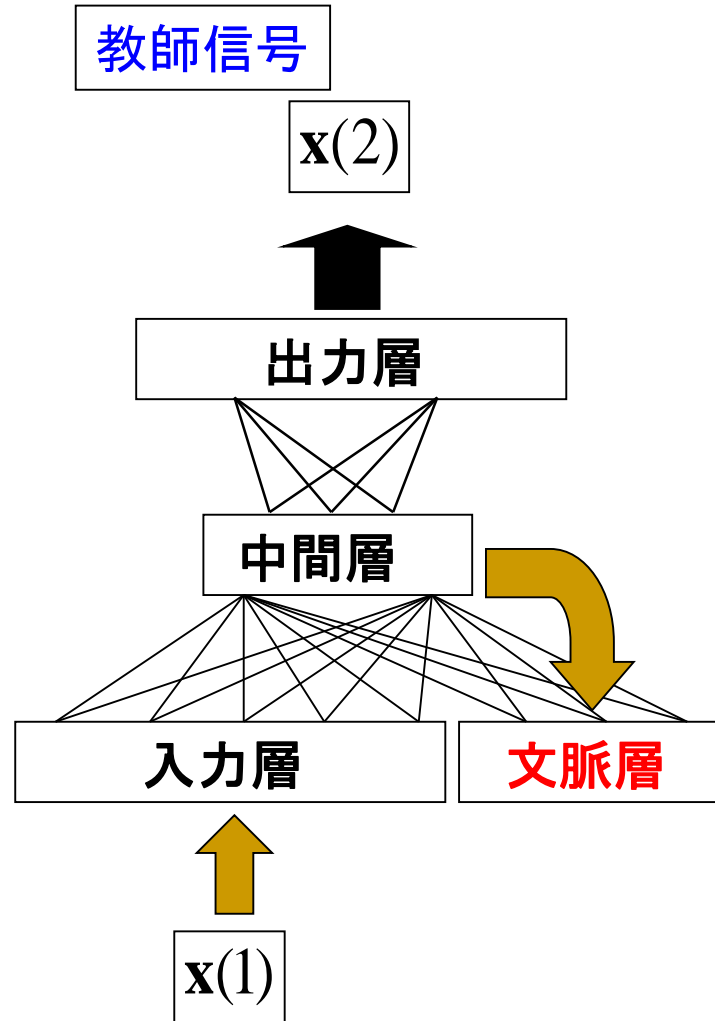
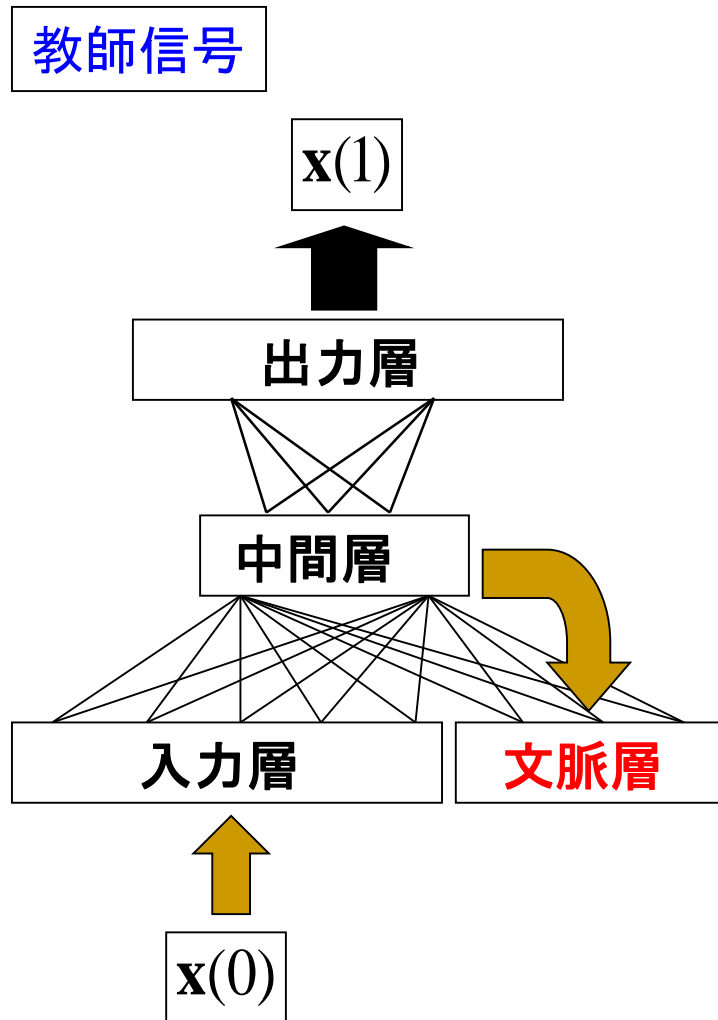
- 1期の入力 $\rightarrow \mathbf{x}(1)$ 2期の教師信号 $\rightarrow \mathbf{x}(2)$

●
●
●

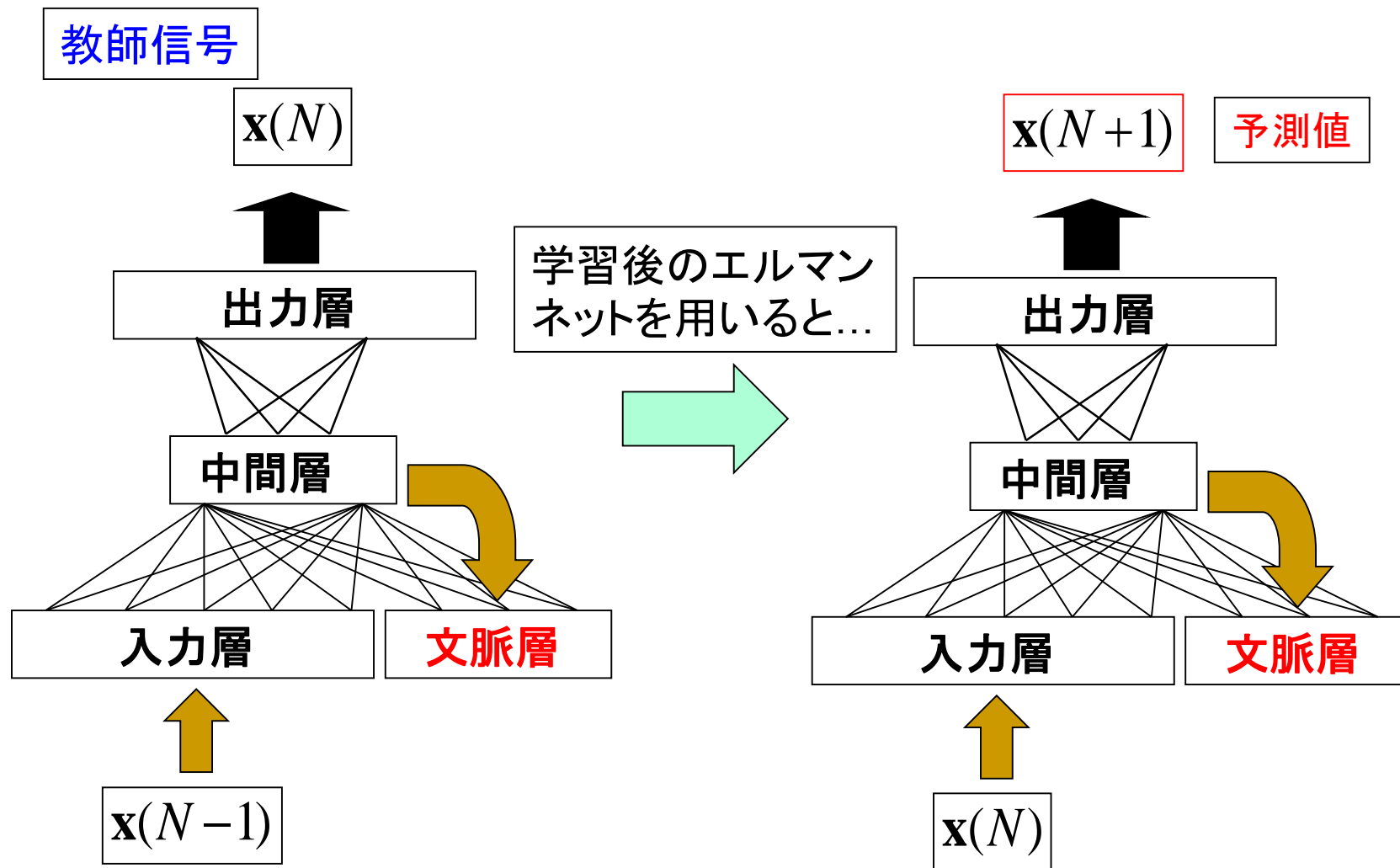
- N-1期の入力 $\rightarrow \mathbf{x}(N-1)$ N期の教師信号 $\rightarrow \mathbf{x}(N)$

■ ネットワークの学習には誤差逆伝播則が利用可能

エルマンネットの学習②

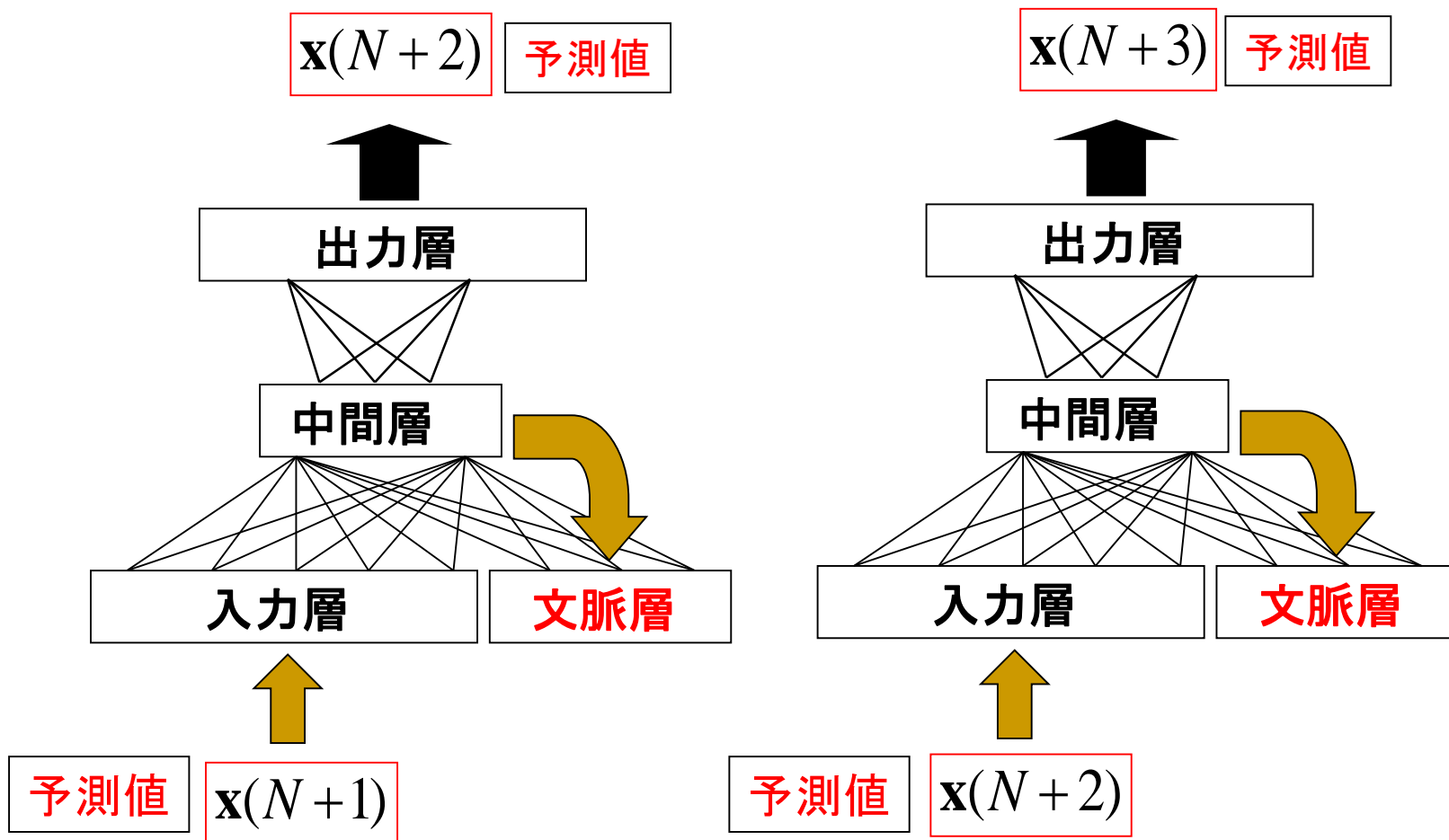


エルマンネットの学習③



エルマンネットの学習④

学習後のエルマンネット



語系列予測課題学習 (Elman, 1990)

■ 現時点の単語から次単語を予測する学習

現在の単語

次の単語

- Mary → sees
- Mary sees → John
- Mary sees John → who
- Mary sees John who → feeds
- Mary sees John who feeds → dogs
- Mary sees John who feeds dogs → .

文頭から順次単語を提示
次単語を予測

直前の単語情報のみしか利用しない

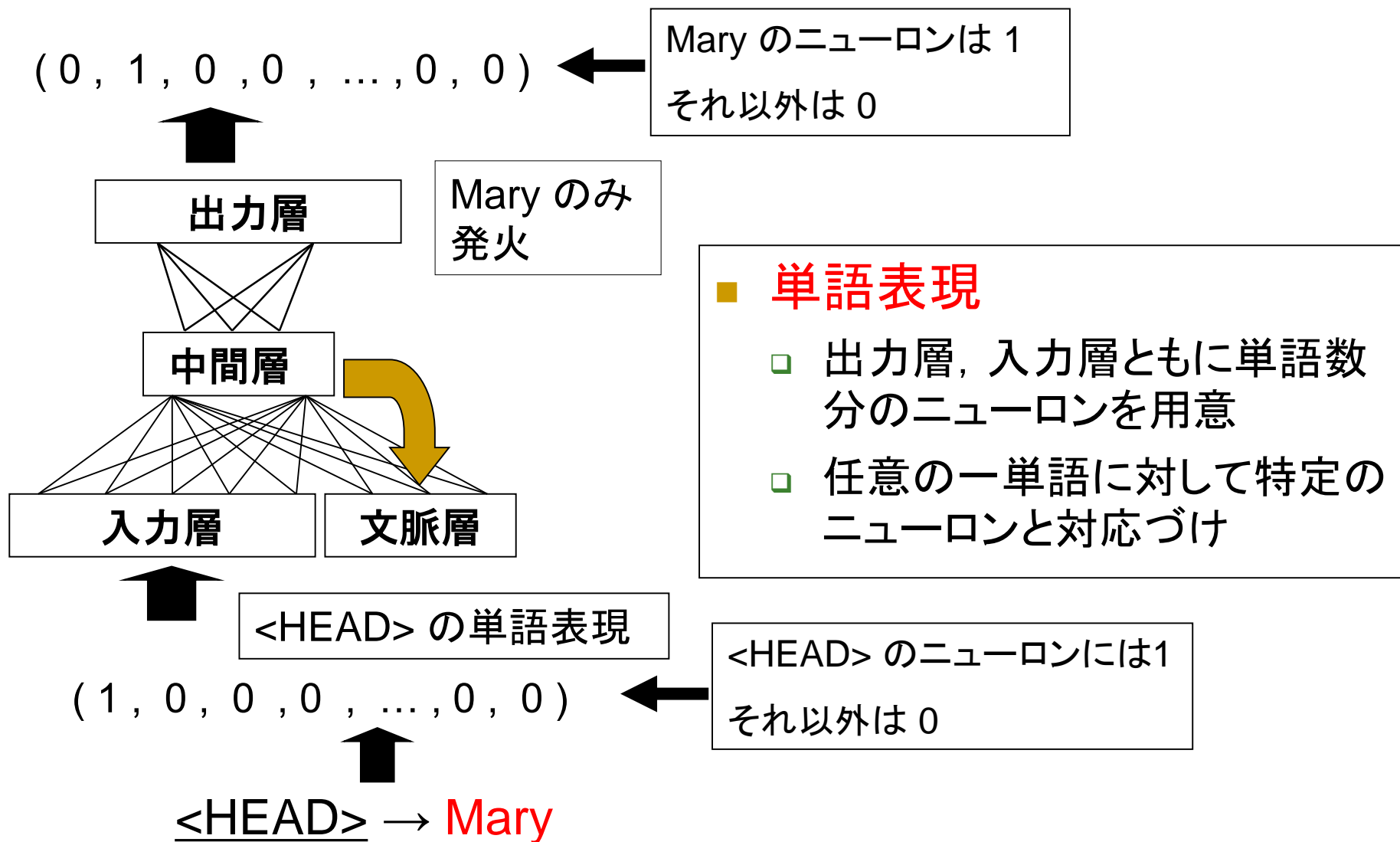
語系列予測課題学習

■ 言語獲得学習

- Elman(1990)

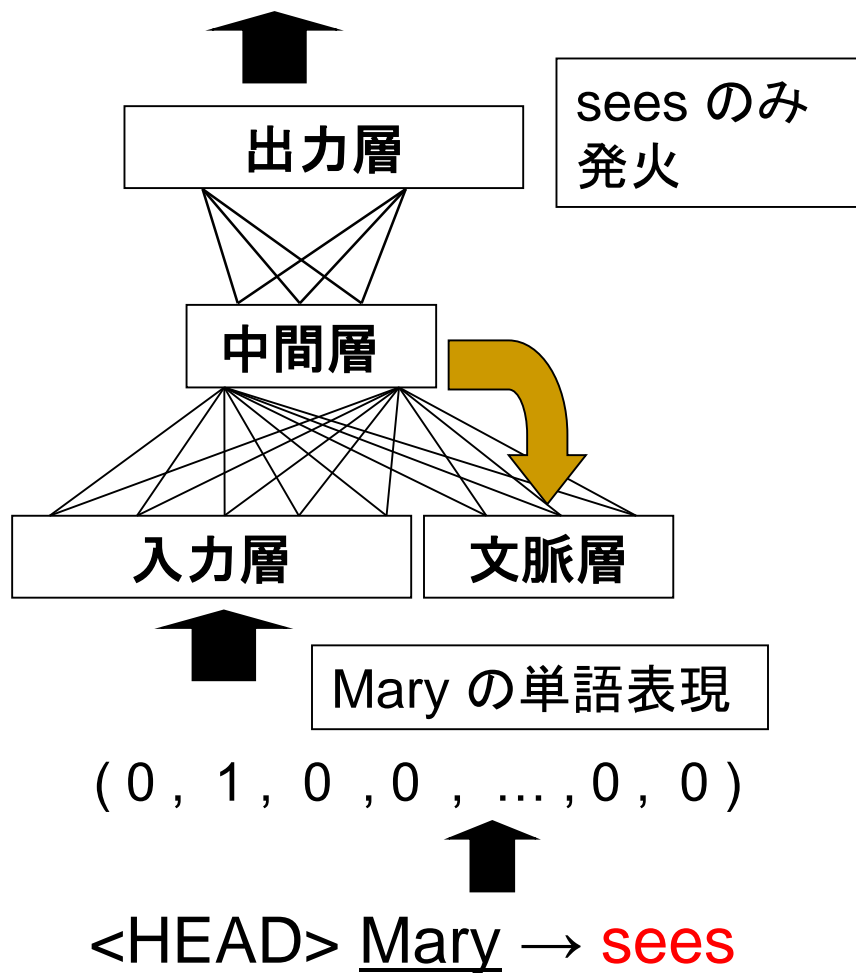
- 幼児が外界(例えば両親)から言葉を聞き, 言語(文法)を獲得していく過程をモデル化

エルマンネットによる語系列予測課題学習①

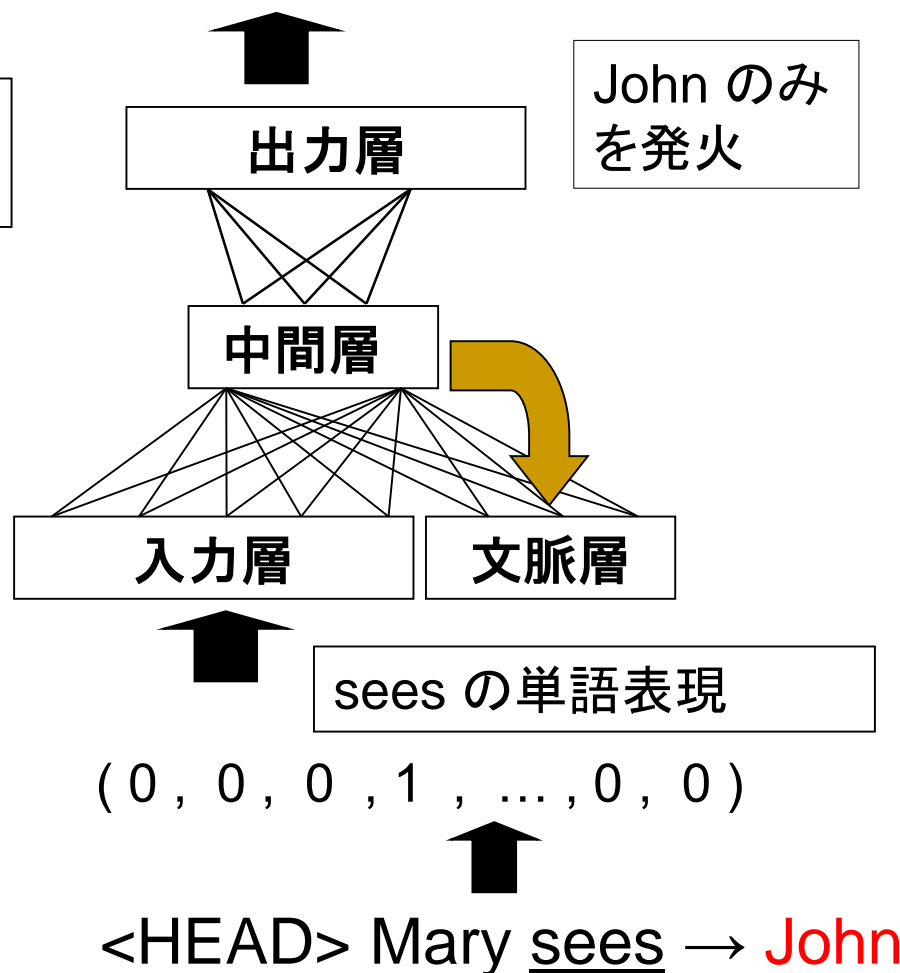


エルマンネットによる語系列予測課題学習②

$(0, 0, 0, 1, \dots, 0, 0)$

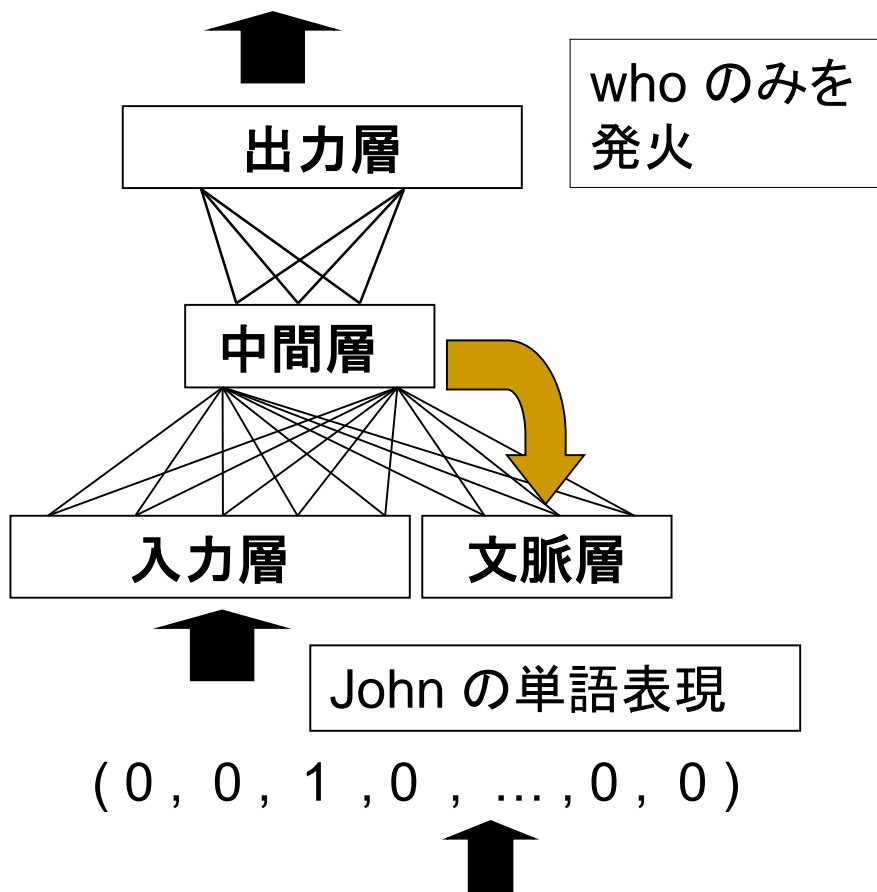


$(0, 0, 1, 0, \dots, 0, 0)$



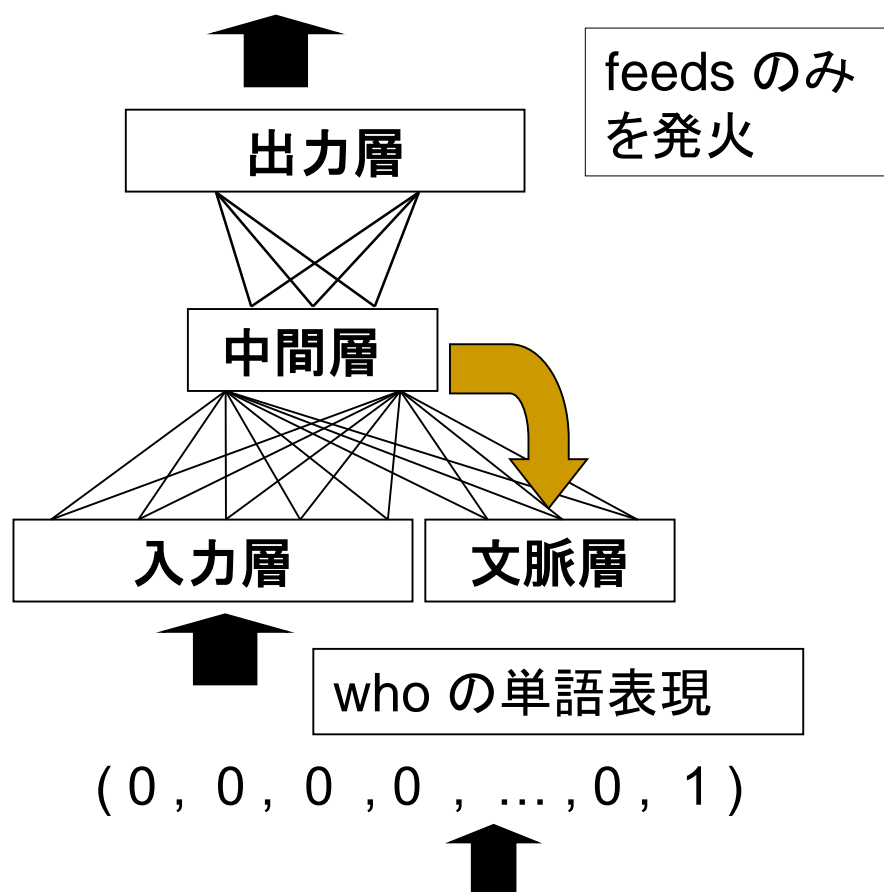
エルマンネットによる語系列予測課題学習③

$(0, 0, 0, 0, \dots, 0, 1)$



<HEAD> Mary sees John →
who

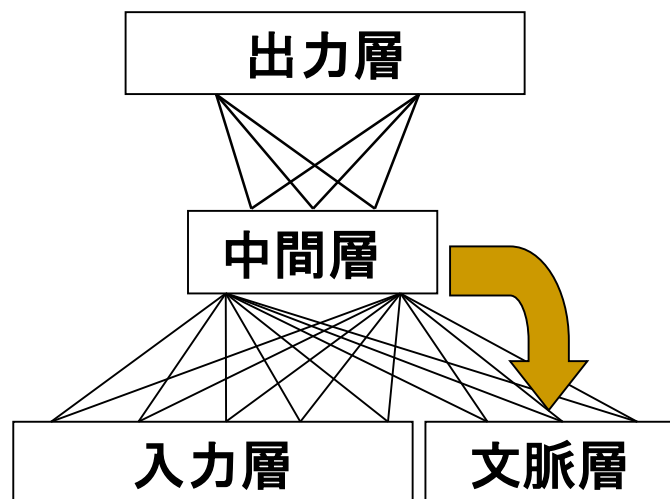
$(0, 0, 0, 0, \dots, 1, 0)$



Mary sees John who →
feeds

学習後のエルマンネットの動作①

学習後のネットワーク



出力結果

sees 0.00128
feeds 0.00115
walks 0.00112
see 0.00002
feed 0.00002
walk 0.00002
Mary 0.00001
John 0.00001

文法的に正しい次単語
の出現確率に近似

who の単語表現

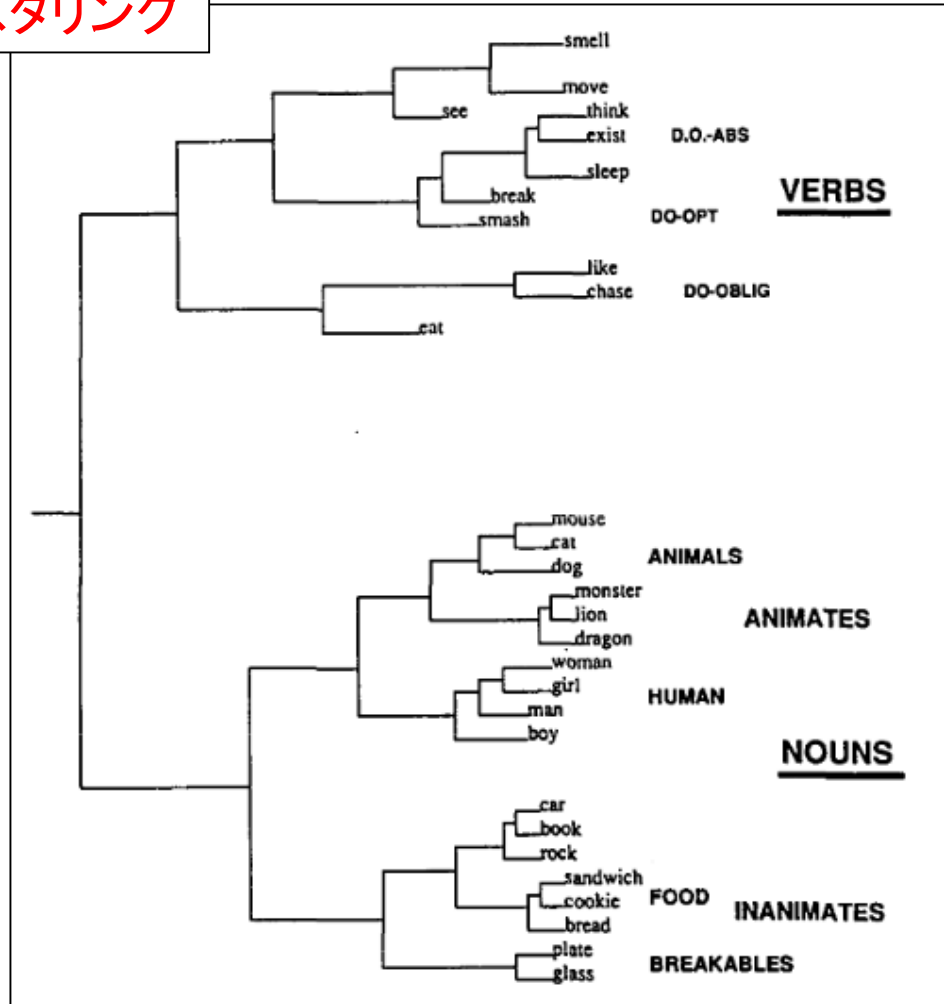
(0, 0, 0, 0, ..., 0, 1)

who のニューロンには1
それ以外は 0

Mary sees John who → ???

学習後のエルマンネットの動作②

文脈層のクラスタリング



学習後のエルマンネットの動作③

- 文法情報は与えず, 前後の単語情報のみを提示
- 文法情報
 - 文脈層の分散表現としてニューラルネットの内部に獲得
- ニューラル言語モデルの基礎

エルマンネットの原理①

$$h(t) = f(wI(t) + vh(t-1))$$

$w=1$ とする

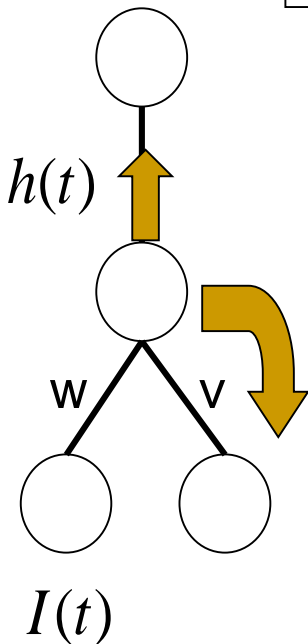
$$h(t) = f(I(t) + vh(t-1))$$

$$h(t-1) = f(I(t-1) + vh(t-2))$$

$$h(t) = f(I(t) + vf(I(t-1) + vh(t-2)))$$

$$h(t-2) = f(I(t-2) + vh(t-3))$$

$$h(t) = f(wI(t) + vf(I(t-1) + vf(I(t-2) + vh(t-3))))$$



エルマンネットの原理②

■ 特徴

$$h(t) = f(wI(t) + vf(I(t-1) + vf(I(t-2) + \cdots vf(I(t-r) + vh(t-r-1))))$$

- 一期前の中間層の値をフィードバック



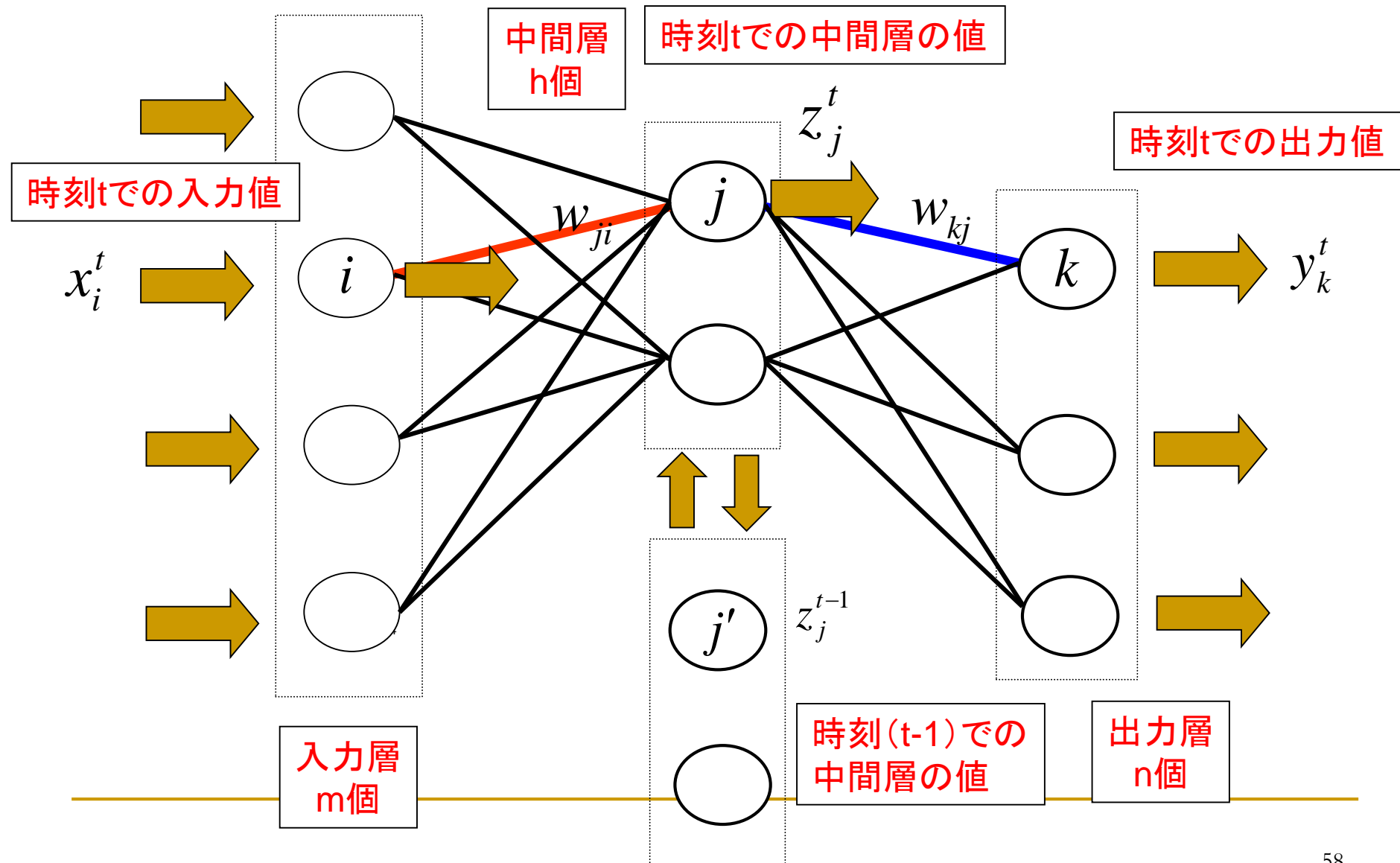
- 過去に入力された全ての情報を(何らかの形で)ネットワーク内部(中間層)で保存

Back propagation through time

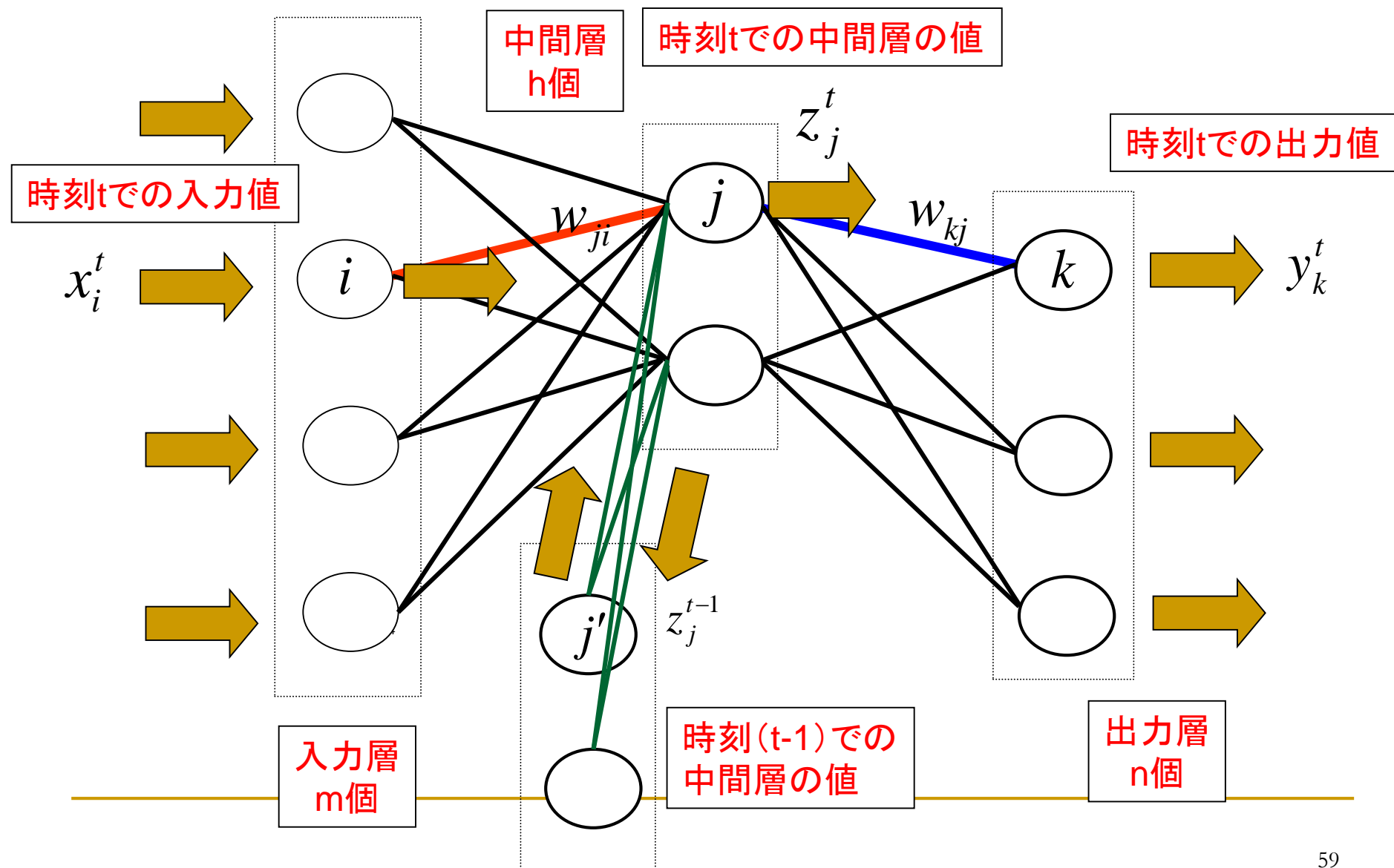
RNNにおける誤差逆伝播

- x_t を入力し, x_{t+1} を教師信号として誤差逆伝播を用いて, RNNの学習は可能
- x_T から x_{T-1} , x_{T-2} , \dots , x_1 と順に遡って, 学習はできない
→ Back propagation through time

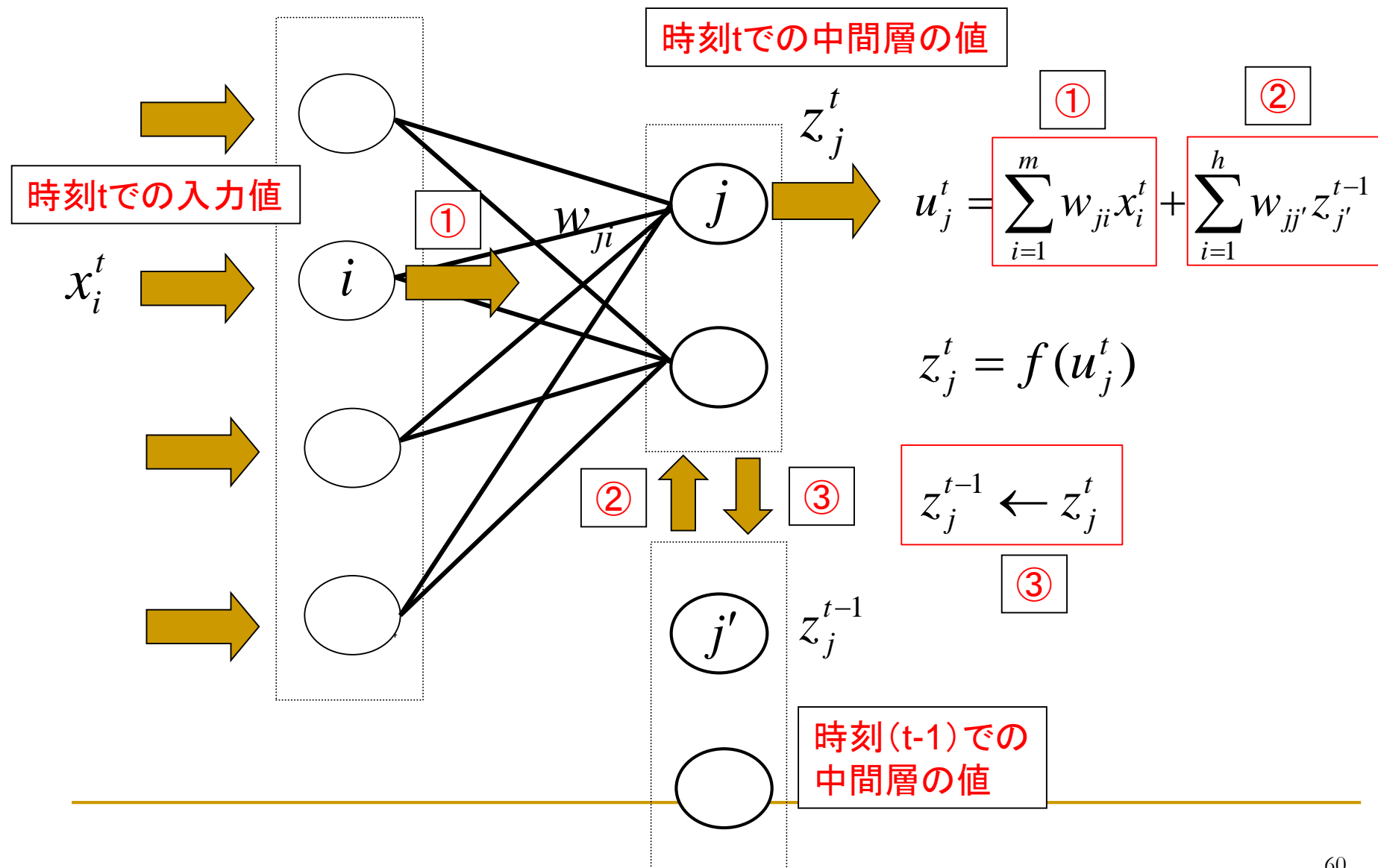
リカレントネットワークの表記



中間層と中間層の結合

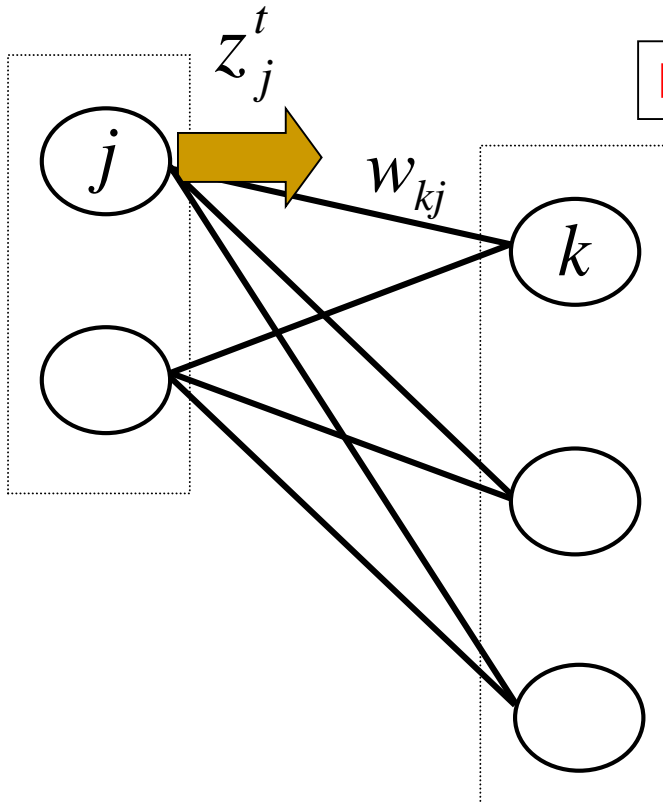


中間層からの出力値の計算



出力層からの出力値の計算

時刻 t での中間層の値

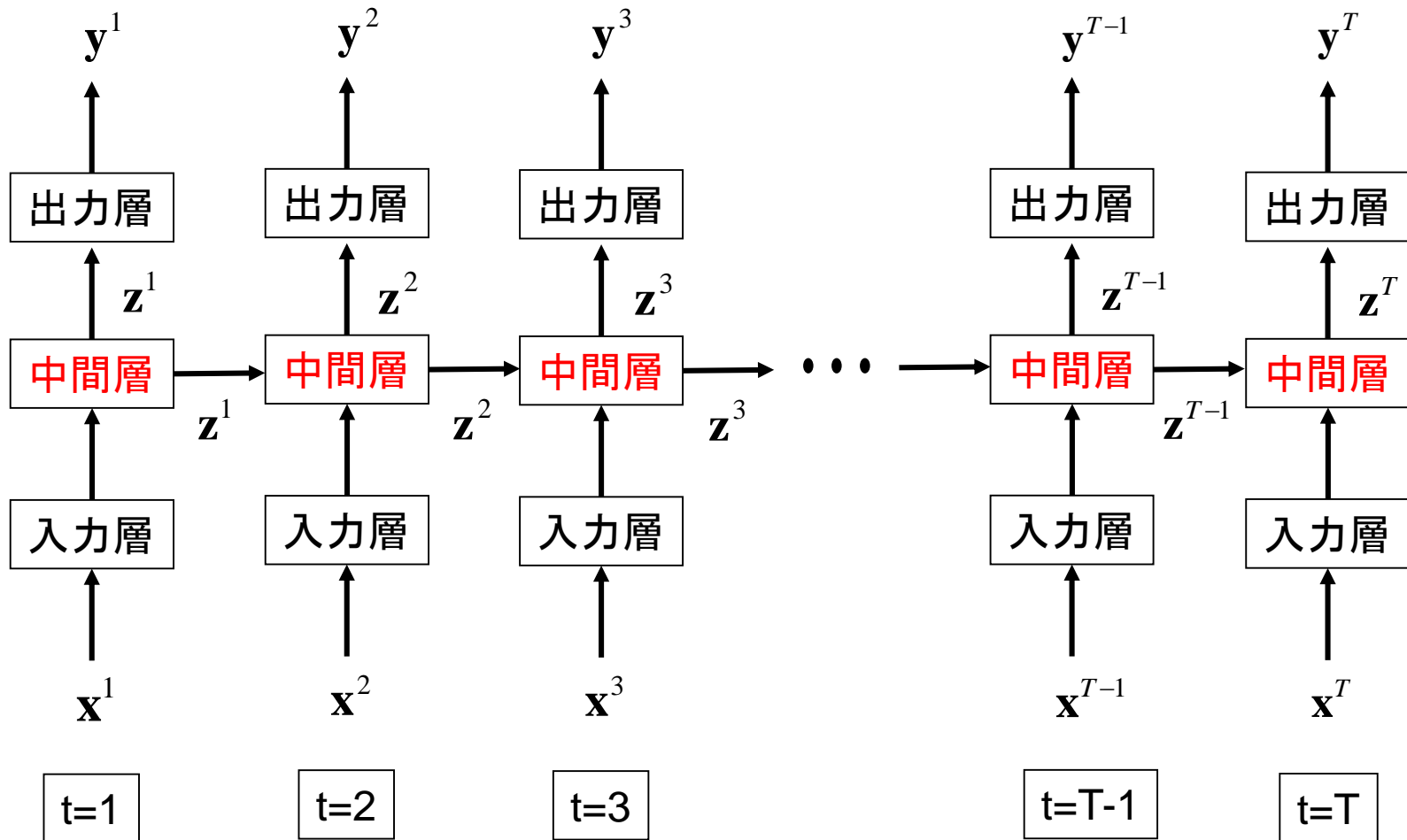


時刻 t での出力値

$$v_k^t = \sum_{j=1}^h w_{kj} z_j^t$$

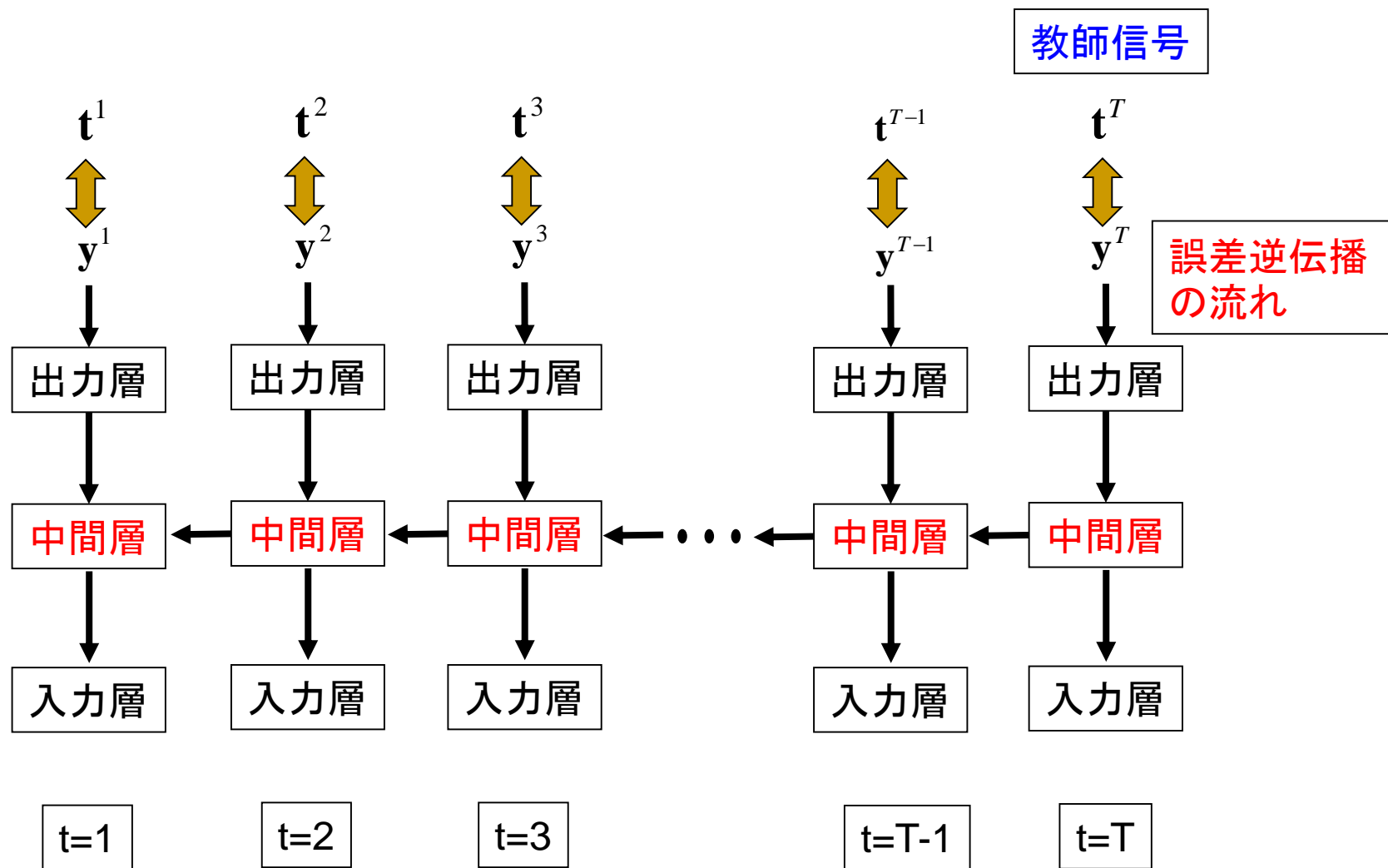
$$y_k^t = f(v_k^t)$$

リカレントネットワークの動作



* \mathbf{z}^0 は0として開始するのが一般的です

学習（誤差逆伝播）の流れ



Back-Propagation through time (BPTT)

■ 損失関数

$$E = \sum_{t=1}^T \sum_{k=1}^n (t_k^t - y_k^t)^2$$

■ 学習

$$w_{kj} \leftarrow w_{kj} - \alpha \frac{\partial E}{\partial w_{kj}}$$

出力層と中間層の結合係数の修正

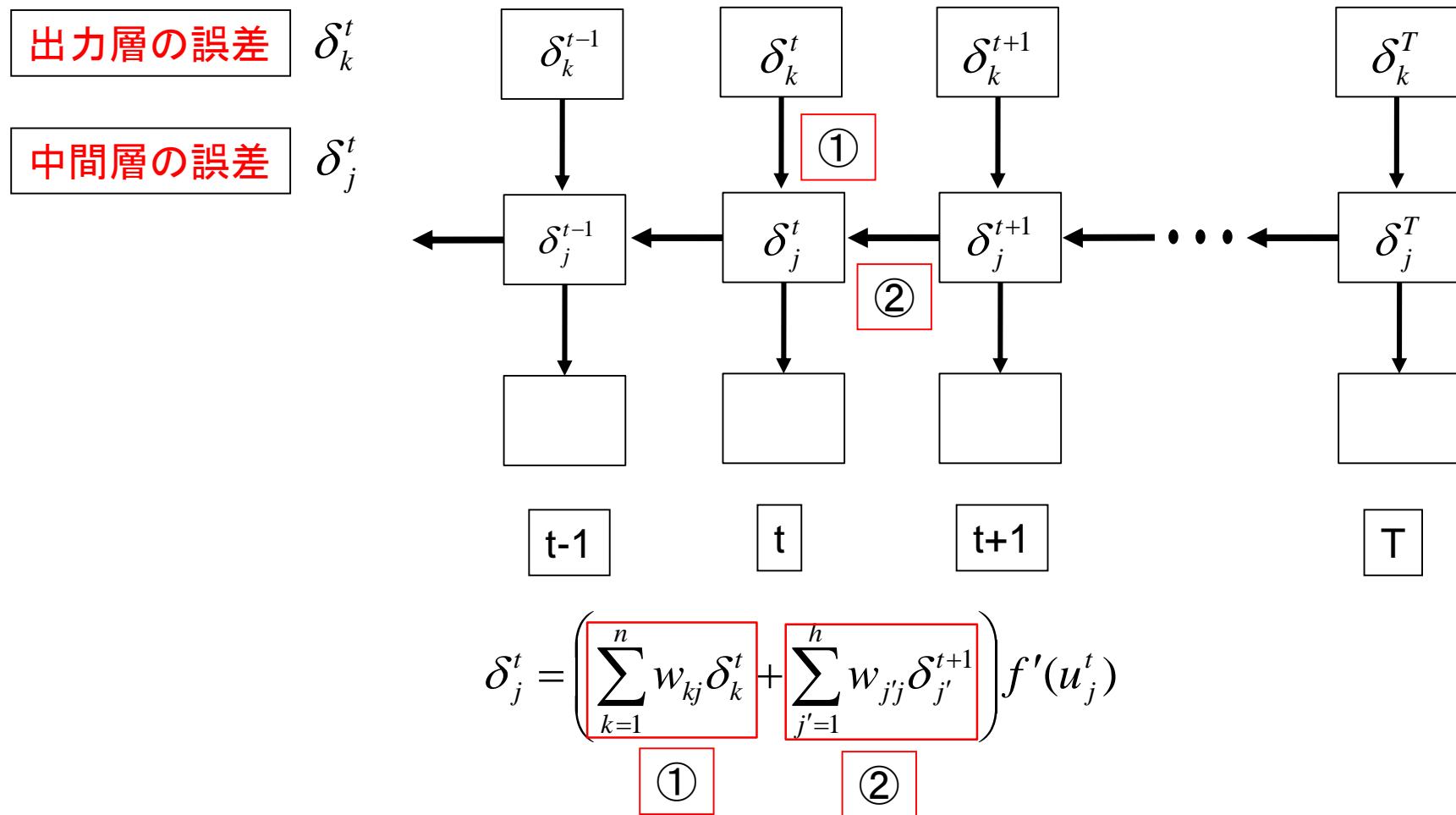
$$w_{ji} \leftarrow w_{ji} - \alpha \frac{\partial E}{\partial w_{ji}}$$

中間層と入力層の結合係数の修正

$$w_{j'j} \leftarrow w_{j'j} - \alpha \frac{\partial E}{\partial w_{j'j}}$$

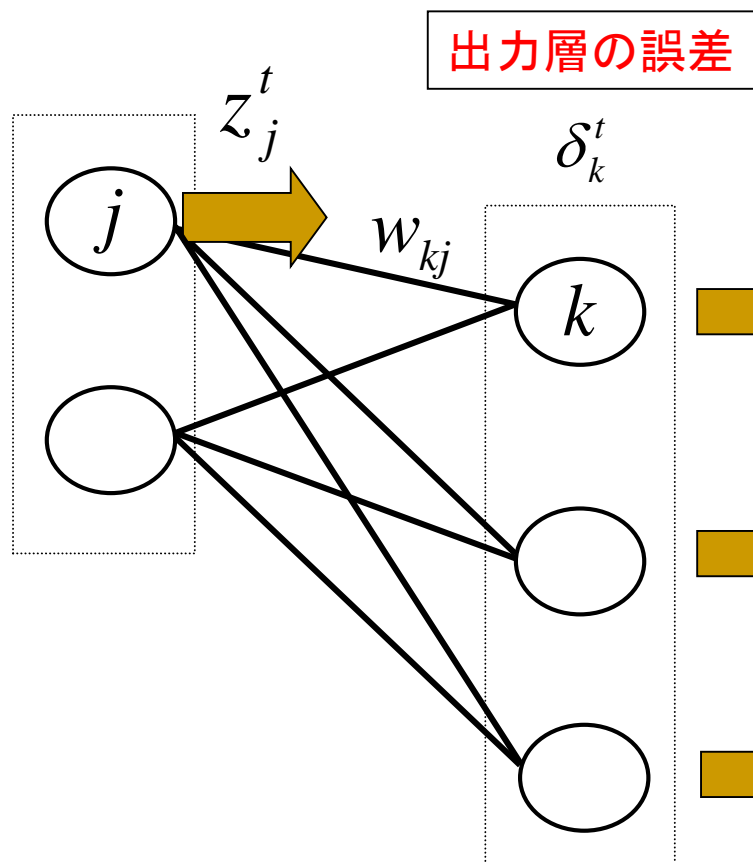
中間層と中間層の結合係数の修正

誤差の逆伝播の流れ



* δ_j^{T+1} は0として開始するのが一般的です

出力層と中間層との結合係数の修正

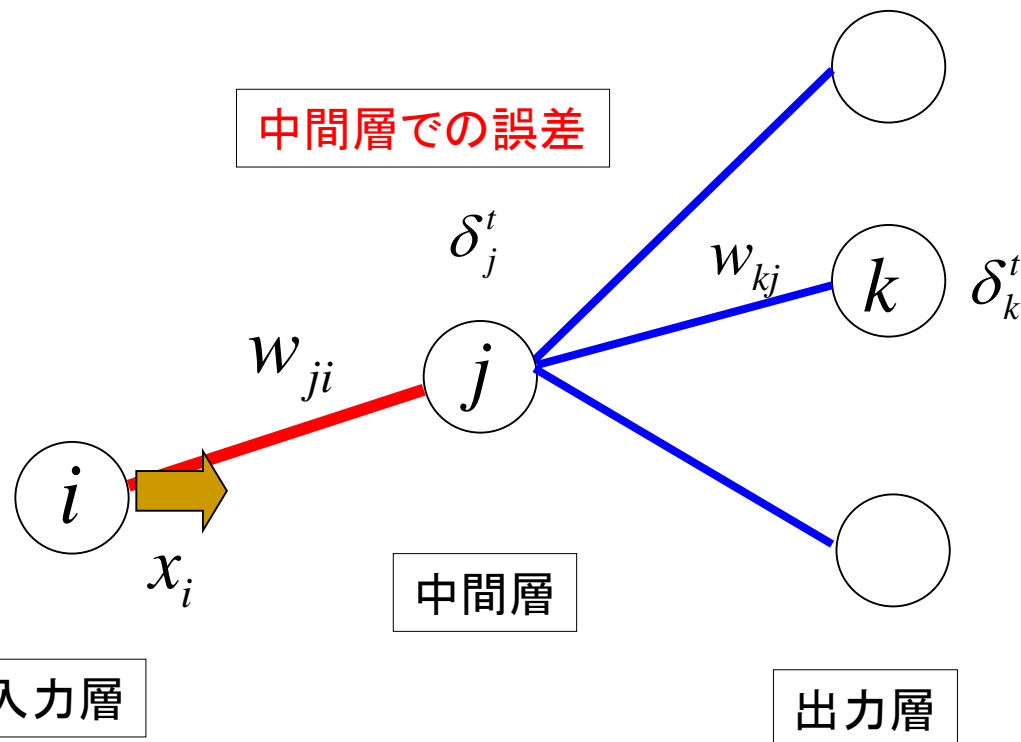


$$v_k^t = \sum_{k=1}^h w_{kj} z_j^t$$

$$y_k^t = f(v_k^t)$$

$$\begin{aligned} \frac{\partial E}{\partial w_{kj}} &= \sum_{t=1}^T \frac{\partial E}{\partial y_k^t} \frac{\partial y_k^t}{\partial v_k^t} \frac{\partial v_k^t}{\partial w_{kj}} \\ &= \sum_{t=1}^T (t_k^t - y_k^t) f'(v_k^t) z_j^t \\ &= \sum_{t=1}^T \delta_k^t z_j^t \end{aligned}$$

入力層と中間層との結合係数の修正



$$u_j^t = \sum_{i=1}^m w_{ji} x_i^t + \sum_{i=1}^h w_{jj'} z_{j'}^{t-1}$$

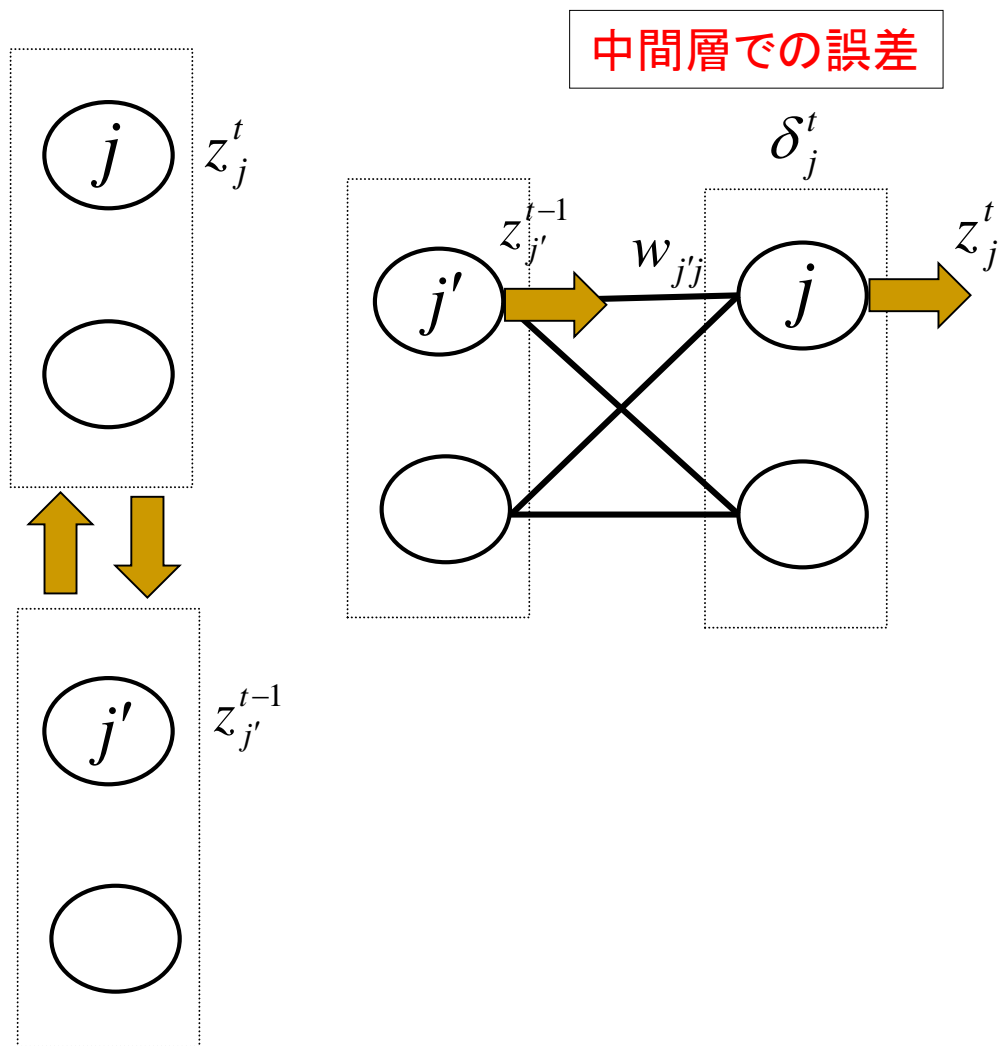
$$z_j^t = f(u_j^t)$$

$$\delta_j^t = \left(\sum_{k=1}^n w_{kj} \delta_k^t + \sum_{j'=1}^h w_{jj'} \delta_{j'}^{t+1} \right) f'(u_j^t)$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_{t=1}^T \frac{\partial E}{\partial u_j^t} \frac{\partial u_j^t}{\partial w_{ji}^t}$$

$$= \sum_{t=1}^T \delta_j^t x_i^t$$

中間層と中間層との結合係数の修正



$$\delta_j^t = \left(\sum_{k=1}^n w_{kj} \delta_k^t + \sum_{j'=1}^h w_{jj'} \delta_{j'}^{t+1} \right) f'(u_j^t)$$

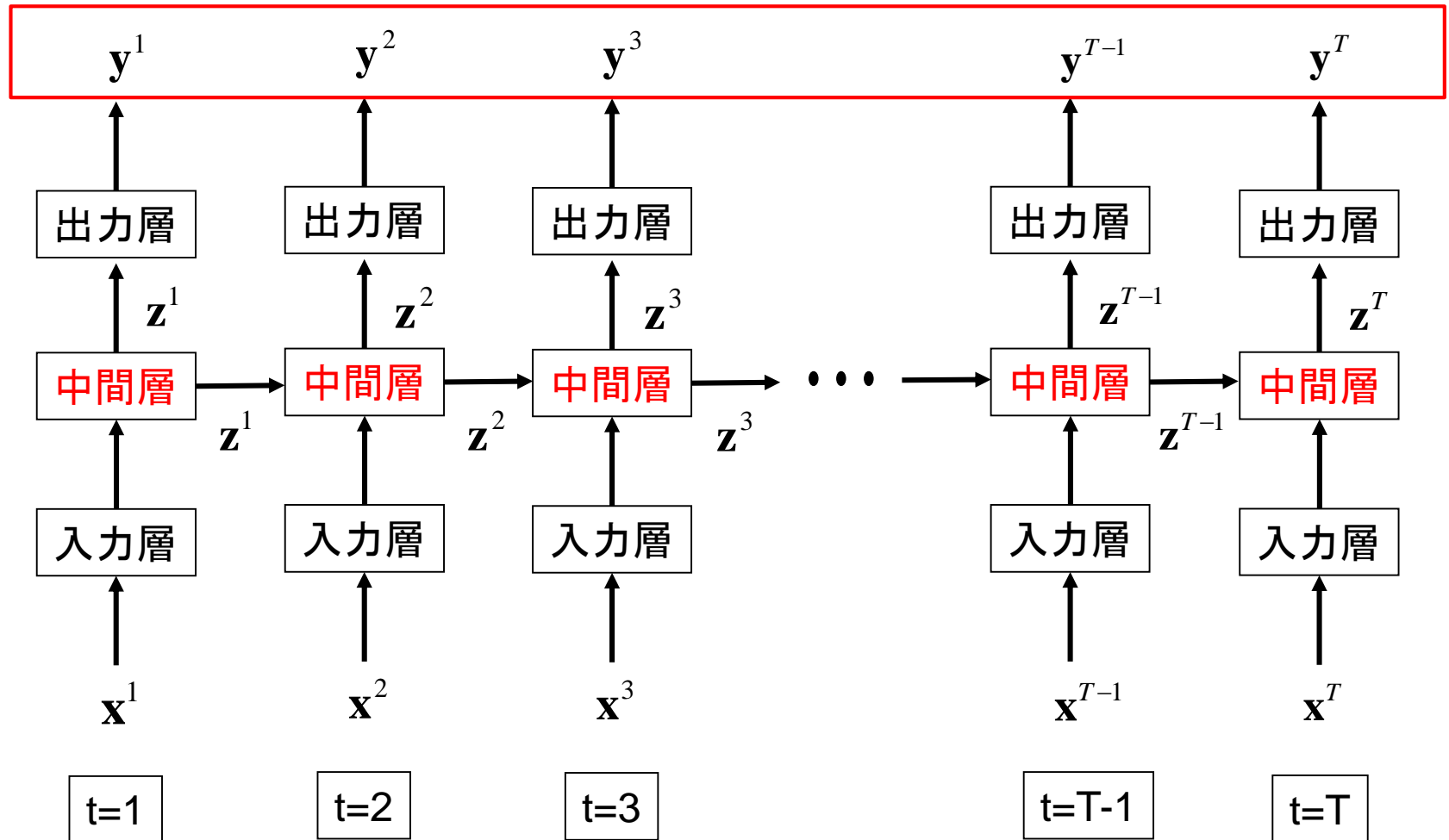
$$u_j^t = \sum_{i=1}^m w_{ji} x_i^t + \sum_{i=1}^h w_{jj'} z_{j'}^{t-1}$$

$$z_j^t = f(u_j^t)$$

$$\begin{aligned} \frac{\partial E}{\partial w_{jj'}} &= \sum_{t=1}^T \frac{\partial E}{\partial u_j^t} \frac{\partial u_j^t}{\partial w_{jj'}} \\ &= \sum_{t=1}^T \delta_j^t z_{j'}^{t-1} \end{aligned}$$

BPTTの流れ(出力値の計算)①

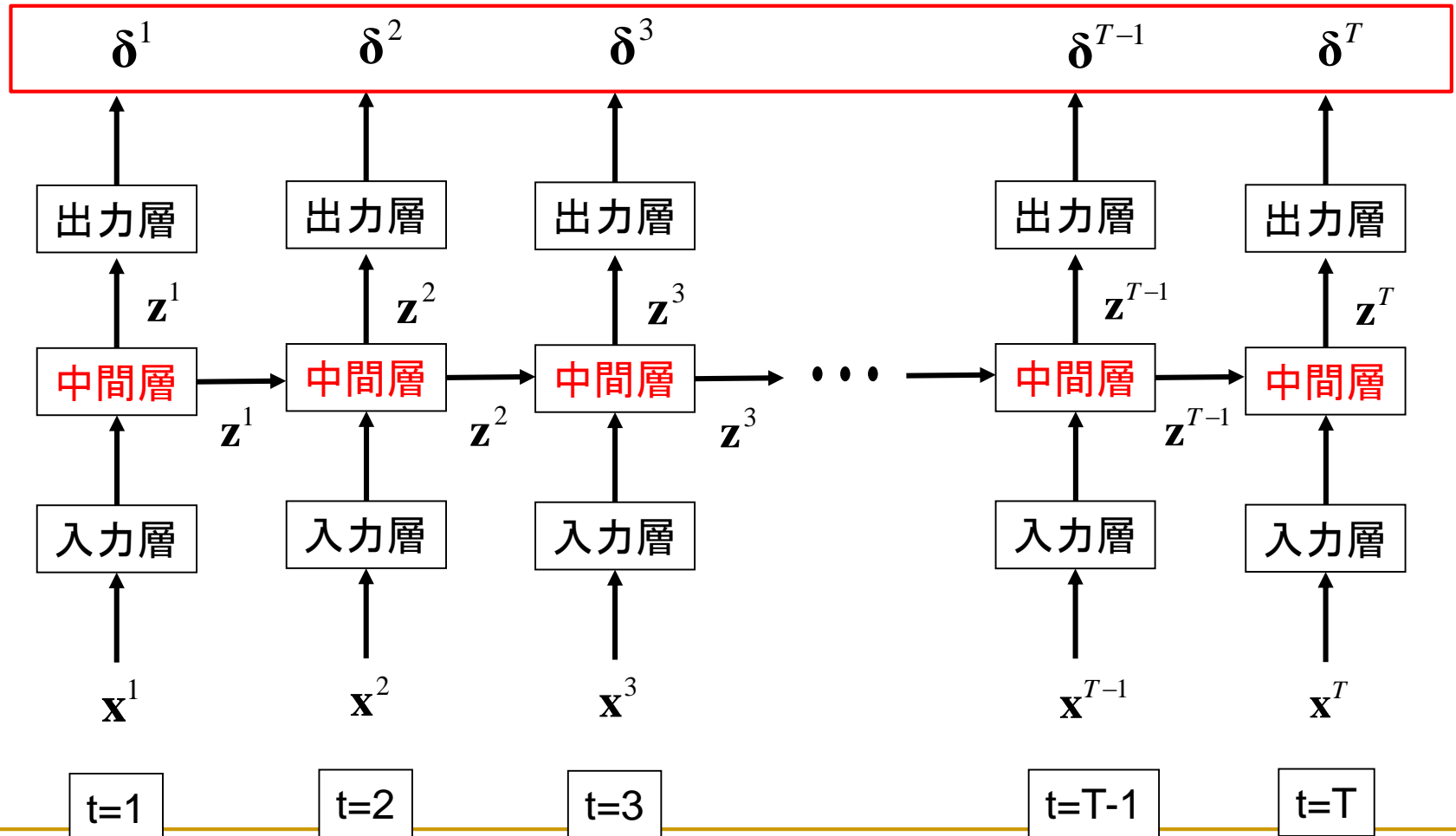
出力値の計算



BPTTの流れ(出力層の誤差の計算)②

出力層の誤差の計算

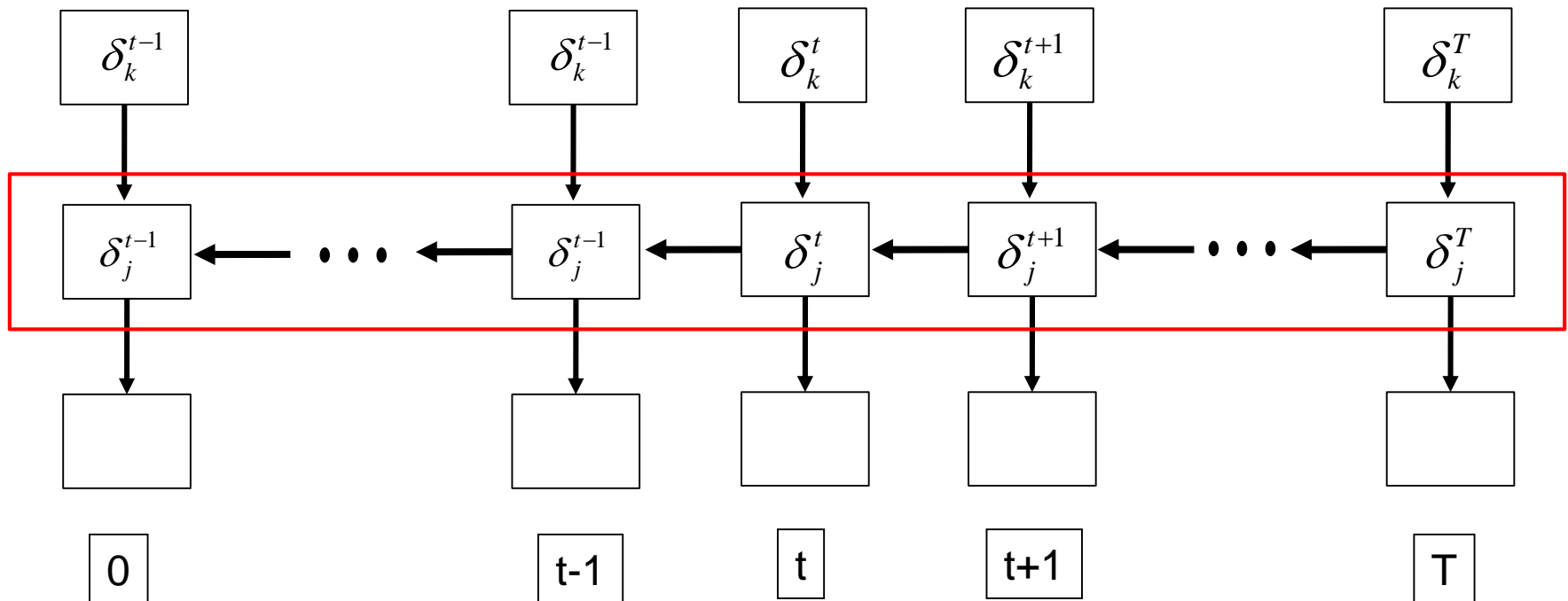
$$\delta_j^t = (t_k^t - y_k^t) f'(v_k^t)$$



BPTTの流れ(中間層の誤差の計算)③

中間層の誤差の計算

$$\delta_j^t = \left(\sum_{k=1}^n w_{kj} \delta_k^t + \sum_{j'=1}^h w_{jj'} \delta_{j'}^{t+1} \right) f'(u_j^t)$$



BPTTの流れ(結合係数の修正)④

- 出力層と中間層との結合係数の修正

$$\frac{\partial E}{\partial w_{kj}} = \sum_{t=1}^T \delta_k^t z_j^t$$

- 中間層と入力層との結合係数の修正

$$\frac{\partial E}{\partial w_{ji}} = \sum_{t=1}^T \delta_j^t x_i^t$$

- 中間層と中間層との結合係数の修正

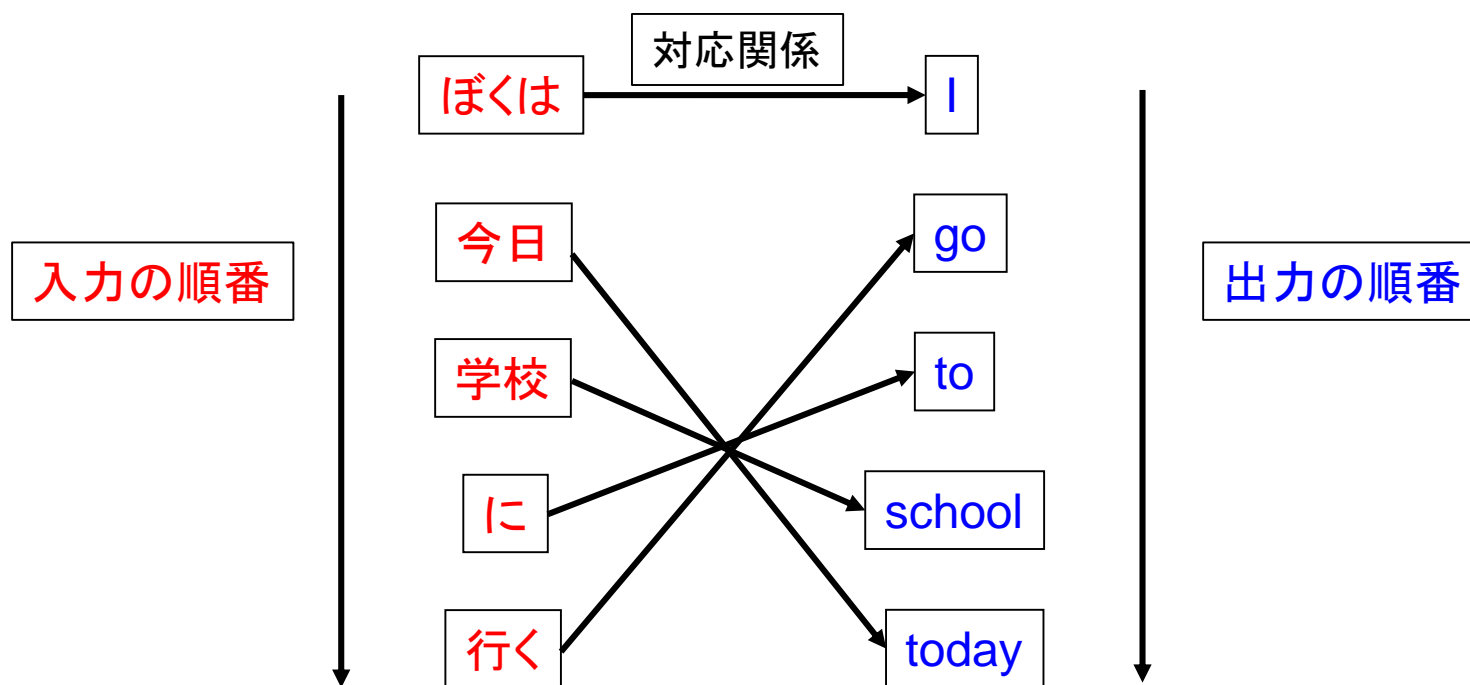
$$\frac{\partial E}{\partial w_{j'j}} = \sum_{t=1}^T \delta_j^t z_{j'}^{t-1}$$

リカレントネットワークの応用

機械翻訳

■ 日英翻訳

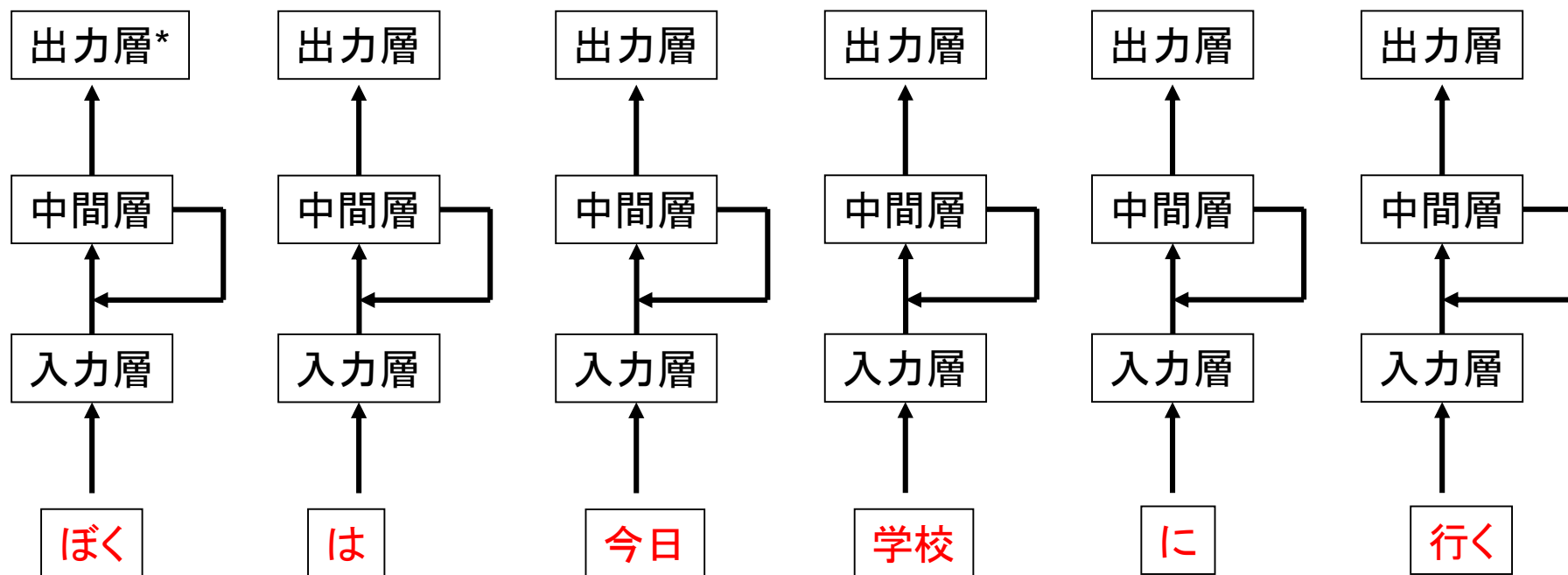
- ぼくは, 今日, 学校に行く. → I go to school today.



RNNによる機械翻訳

■ 日英翻訳

□ ぼくは、今日、学校に行く。 → I go to school today.

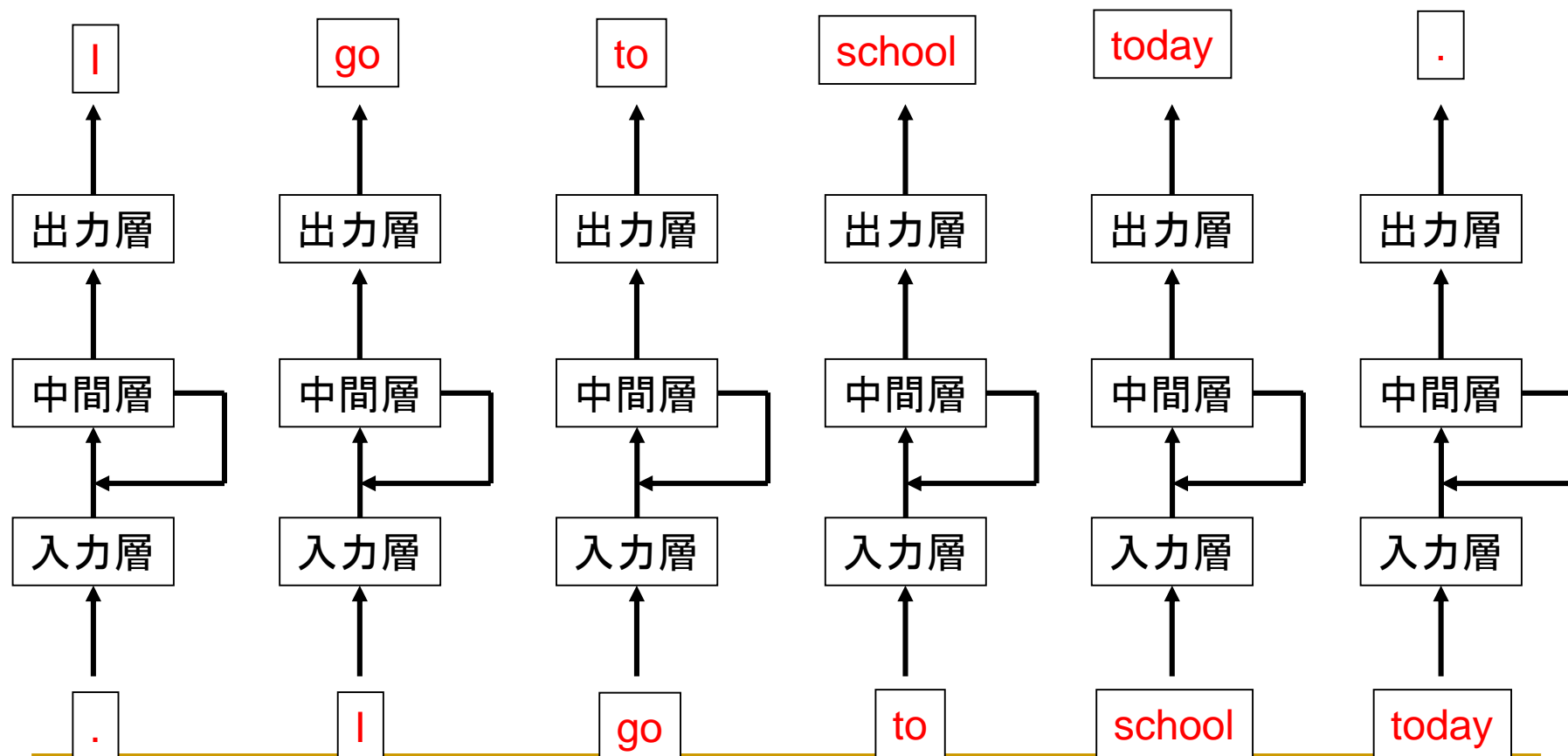


*出力層からは出力はなくてよい

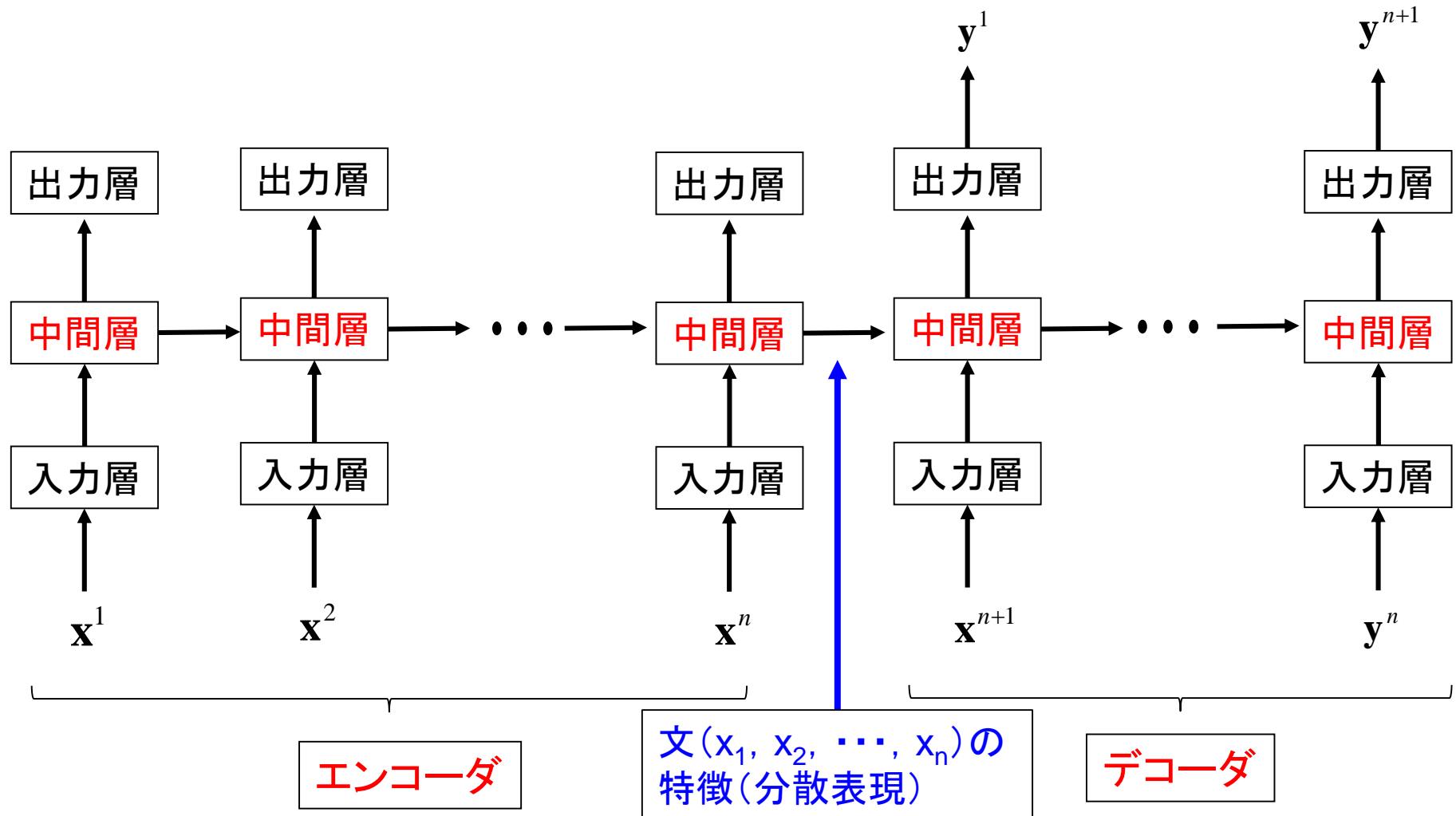
RNNによる機械翻訳

■ 日英翻訳

□ ぼくは、今日、学校に行く。 → I go to school today.



エンコーダ・デコーダ型

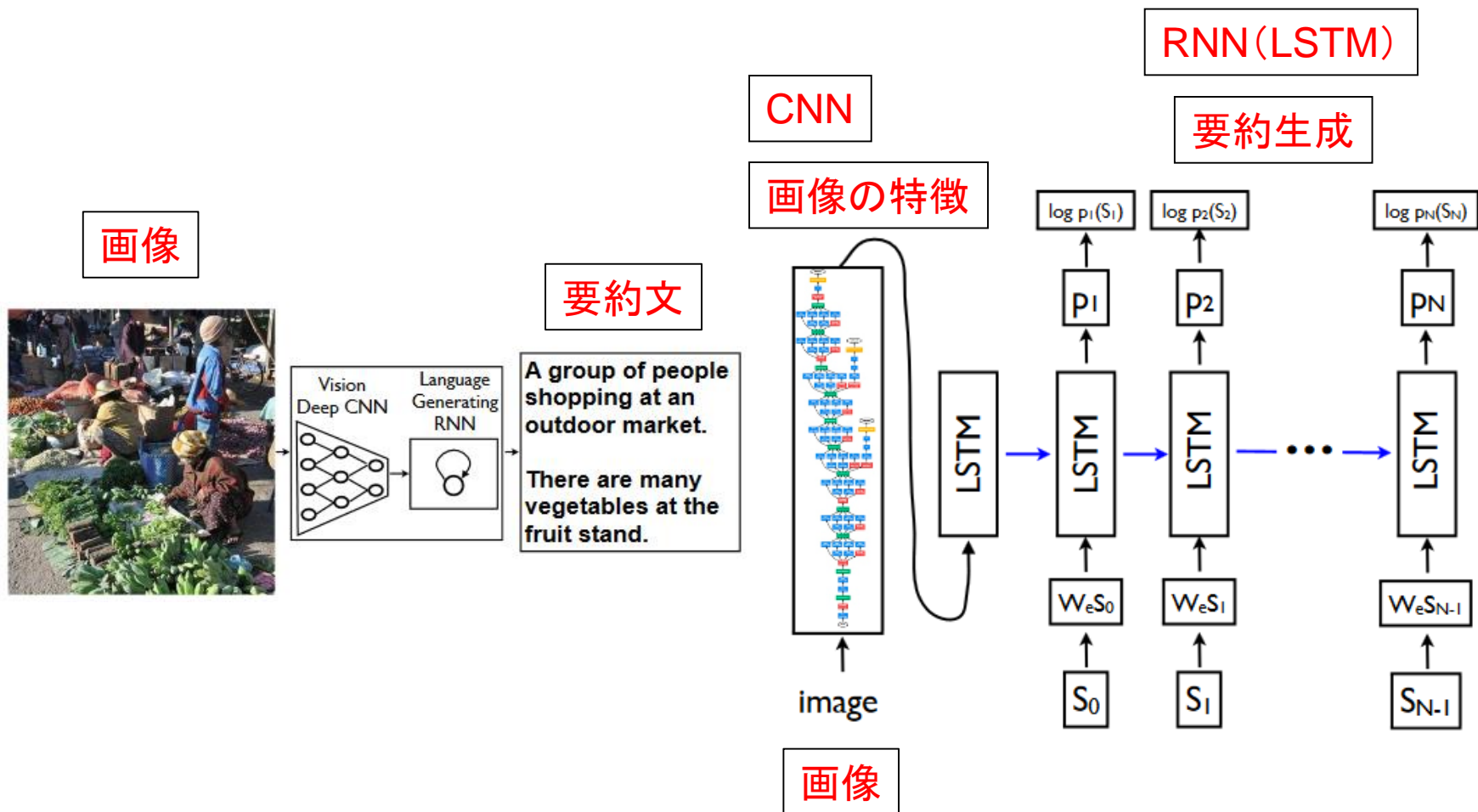


*Seq2Seq (Sequence to Sequence) と呼ばれます

Seq2Seq型の応用

- 機械翻訳
 - 対話, 質問応答
 - 文書生成
 - 文書要約
-
- LSTM
 - 長期依存性により対応が可能なRNNとして, LSTM(Long Short-term Memory)が近年, よく利用されている

要約文の生成



実習(単純再帰結合型ネットワーク)

語系列課題学習

エルマンネット (Elman.py)

- 語系列課題学習 (corpus-200.txt, corpus-1000.txt)
- 「dat」というフォルダーを作成して下さい

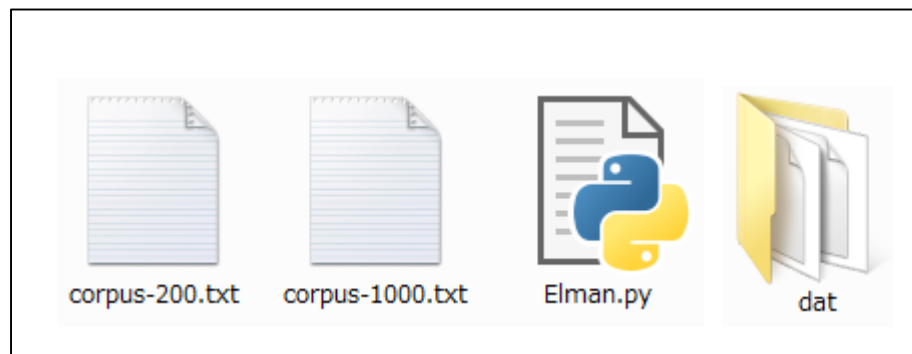
- 実行方法

- 学習

- > python Elman.py t

- 認識

- > python Elman.py p



引数をつけて下さい

コーパス

- SCoRE

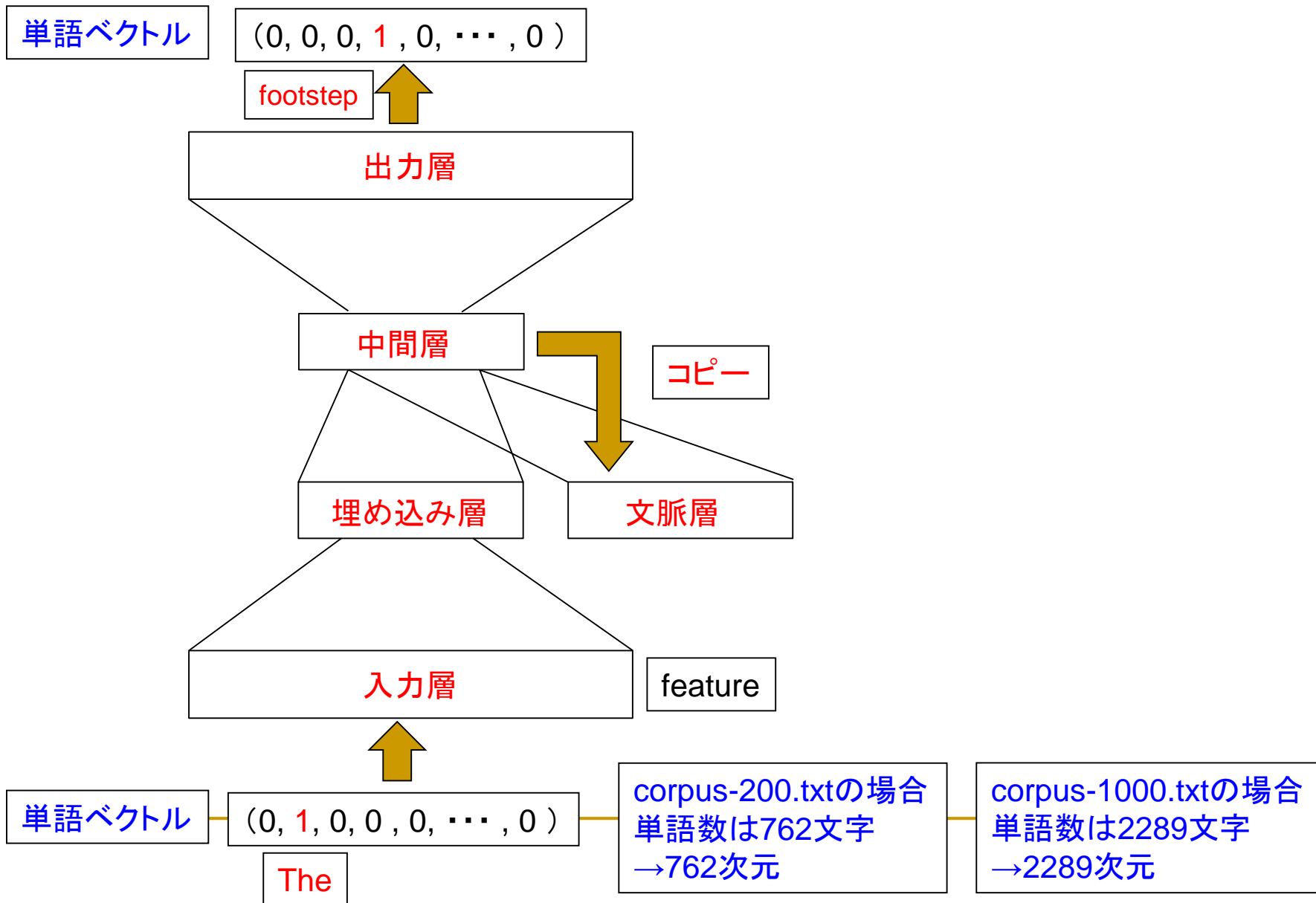
- <http://www.score-corpus.org/>

The footstep was that of a teacher . <eos>

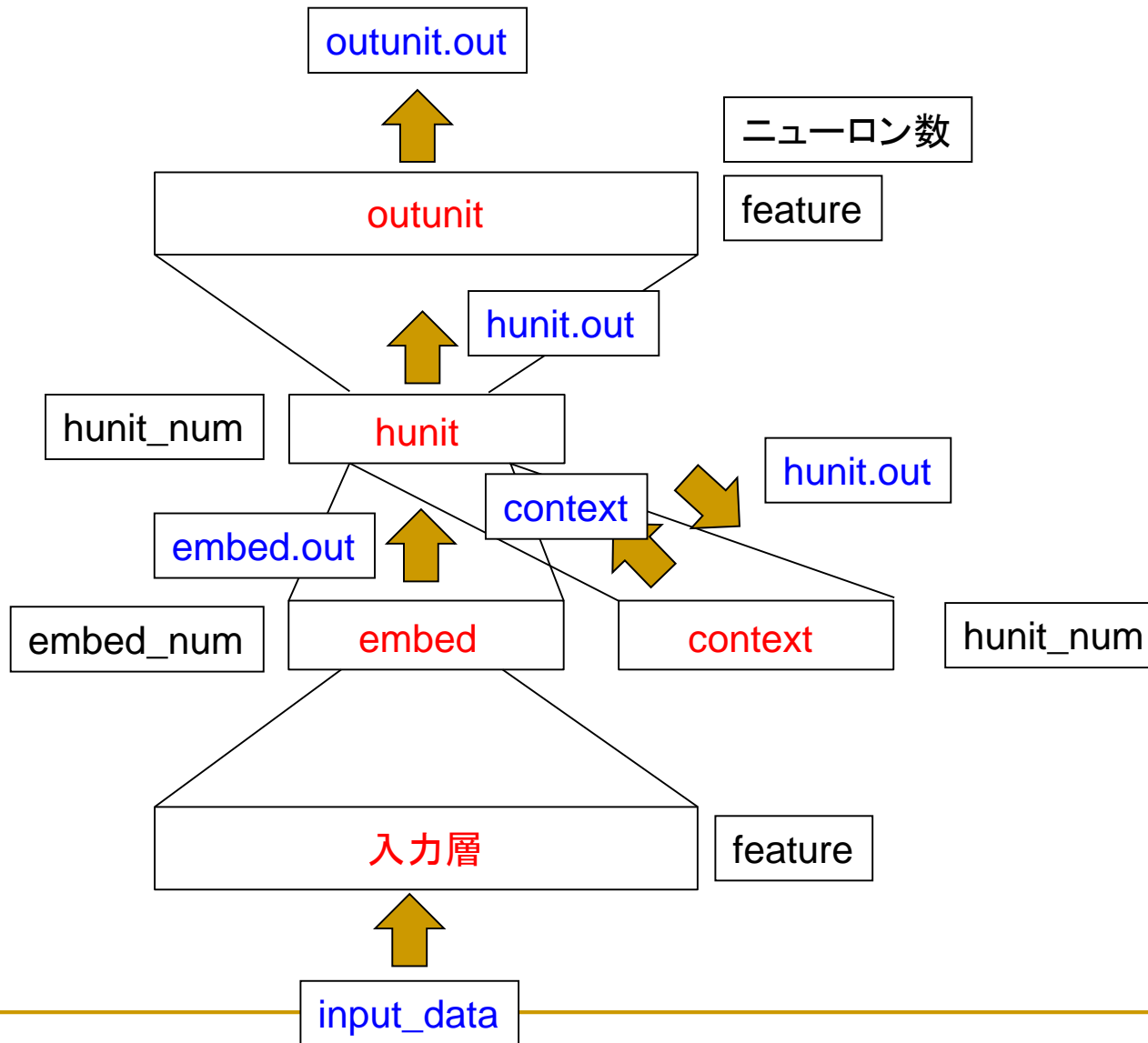
文の終了

- corpus-200.txt (200文)
 - corpus-1000.txt (1,000文)

ニューラルネットワークの構造①



ニューラルネットワークの構造②(変数名)



ニューラルネットワークの構造③

- 出力層
 - 損失関数: 交差エントロピー
 - 活性化関数: ソフトマックス関数
 - 個数: 単語数(feature)
- 中間層
 - 活性化関数: シグモイド関数
 - 個数: `hunit_num`(100個)
- 入力層
 - 個数: 単語数(feature)

ニューラルネットワークの構造④

■ 埋め込み層

- 目的: 入力された単語ベクトルの次元数の削減
- 個数: `embed_num`(100個)

■ 文脈層

- 個数: 中間層と同数, `hunit_num`(100個)

埋め込み層①

(中間層, 入力層はこれまでのプログラムと同一です)

```
class Embed:
```

```
    def __init__(self, n, m):
```

```
        # 重み
```

```
        self.w = np.random.uniform(-0.5,0.5,(n,m))
```

```
        # 閾値
```

```
        self.b = np.random.uniform(-0.5,0.5,m)
```

```
    def Propagation(self, x):
```

```
        self.x = x
```

```
        # 内部状態
```

```
        self.u = np.dot(self.x, self.w) + self.b
```

```
        # 出力値(恒等関数)
```

```
        self.out = self.u
```

n: 入力層の個数
m: 埋め込み層の個数

重み: (n × m) の行列
→ 0.5から0.5の乱数で初期化

閾値: m次元のベクトル
→ 0.5から0.5の乱数で初期化

$$\mathbf{u}_p = \mathbf{x}_p W + \mathbf{b}_p$$

次元数の削減が目的のため
恒等関数(線形結合)を用いる

$$\mathbf{o}_p = \mathbf{u}_p$$

埋め込み層②

```
def Error(self, p_error):
```

```
    # 誤差
```

p_error: 上位の層から伝播する誤差

```
    f_ = 1
```

```
    delta = p_error * f_
```

```
    # 重み, 閾値の修正値
```

```
    self.grad_w = np.dot(self.x.T, delta)
```

$$\partial W = \mathbf{x}^t \Delta$$

```
    self.grad_b = np.sum(delta, axis=0)
```

$$\partial \mathbf{b} = \mathbf{1}^t \Delta$$

```
    # 前の層に伝播する誤差
```

```
    self.error = np.dot(delta, self.w.T)
```

$$\Delta W^t$$

```
def Update_weight(self):
```

```
    # 重み, 閾値の修正
```

```
    self.w -= alpha * self.grad_w
```

$$W' = W - \alpha \partial W$$

```
    self.b -= alpha * self.grad_b
```

$$\mathbf{b}' = \mathbf{b} - \alpha \partial \mathbf{b}$$

埋め込み層②

```
def Save(self, filename):
```

```
    # 重み, 閾値の保存
```

```
    np.savez(filename, w=self.w, b=self.b)
```

```
def Load(self, filename):
```

```
    # 重み, 閾値のロード
```

```
    work = np.load(filename)
```

```
    self.w = work['w']
```

```
    self.b = work['b']
```

np.savez

numpy形式のデータの保存(バイナリ)

np.savez(ファイル名, 変数名)

→ ファイル名.npzとして保存

重み→キー「w」

閾値→キー「b」

np.load

numpy形式のデータのロード

np.load(ファイル名)

キー「w」→重み

キー「b」→閾値

メインメソッド

```
if __name__ == '__main__':  
    # コーパスの読み込み  
    word, sentence = Read_Corpus()  
    eos_id = word[ '<eos>' ]  
    print( eos_id )  
  
    # 埋め込み層の個数  
    embed_num = 100  
  
    # 中間層の個数  
    hunit_num = 100  
  
    # 特徴数(総単語数)  
    feature = len( word )
```

コーパスの読み込み
Read_Corpus()

単語辞書
word{'単語'} → 単語id

全文
sentence
{単語id, 単語id, 単語id, ..., eos_id,
単語id, 単語id, 単語id, ..., eos_id}

文の区切り
eos_idで区切られている

埋め込み層, 中間層のコンストラクター

embed = Embed(feature , embed_num)

埋め込み層の個数: embed_num
一つ前の層(入力層)の個数: feature

hunit = Hunit(embed_num+hunit_num , hunit_num)

出力層のコンストラクター

outunit = Outunit(hunit_num , feature)

中間層の個数: hunit_num
一つ前の層(埋め込み層+文脈層)の個数:
embed_num+hunit_num

argvs = sys.argv

出力層の個数: feature
中間層の個数: hunit_num

引数がtの場合

if argvs[1] == "t":

学習

Train()

elif argvs[1] == "p":

予測

Predict()

コーパスの読み込み①

```
def Read_Corpus():
```

```
# ファイルのオープン
```

```
work = []
```

```
with open('corpus-200.txt', "r", encoding="utf-8-sig") as f:
```

```
    for line in f:
```

```
        # 改行を削除
```

```
        work.append( line.rstrip("\n") )
```

```
# 文字列に変換
```

```
work1 = " ".join(work)
```

```
# 空白で分割
```

```
text = work1.strip().split()
```

1,000文の場合
corpus-1000.txt

The footstep was that of a teacher . <eos>

改行を削除

work1

The footstep was that of a teacher . <eos> His voice
is louder than that of mine . <eos> ...

text

空白で分割

'The' , 'footstep' , 'was' , 'that' , 'of' , 'a' , 'teacher'
, '.' , '<eos>' , 'His' , 'voice' , 'is' , 'louder' , 'than' ,
'that' , 'of' , 'mine' , '.' , '<eos>' , ...

コーパスの読み込み②

辞書の作成

word{"単語"} -> 単語id

sentence = [単語id , 単語id , ... , 単語id]

word = {}

sentence = np.ndarray((len(text),), dtype=np.int32)

全文の単語数分, 確保

for i, w in enumerate(text):

if w not in word:

word[w] = len(word)

単語idの確定

sentence[i] = word[w]

sentenceに単語idを代入

return word , sentence

全文
sentence

```
C:\Windows\system32\cmd.exe
C:\home\shino\prml-2018\12-24\program\program-10>python3 Elman.py p
[ 0 1 2 ... 761 7 8]
```

単語id

単語辞書

単語

単語id

単語辞書

word{'単語'} → 単語id

```
C:\Windows\System32\cmd.exe
{'The': 0, 'footstep': 1, 'was': 2, 'that': 3, 'of': 4, 'a': 5, 'teacher': 6, '': 7, 'ed': 20, 'to': 21, "Linda's": 22, 'singer': 23, 'smell': 24, "mother's": 25, 'pie': 37, 'China': 38, 'Japan': 39, 'climate': 40, 'here': 41, 'very': 42, 'from': 43, 'craftsmen': 56, 'style': 57, 'painting': 58, 'similar': 59, 'Picasso': 60, 'poli': 72, 'almost': 73, 'identical': 74, "Tasha's": 75, 'laugh': 76, 'twin': 77, "b": 89, 'Suzy': 90, 'extremely': 91, 'delicious': 92, 'and': 93, 'five': 94, 'star': 107, 'listening': 108, 'him': 109, 'equivalent': 110, 'professional': 111, 'p': 122, 'loud': 123, 'sound': 124, 'coming': 125, 'next': 126, 'door': 127, "Tom's": 140, '10': 141, 'years': 142, 'old': 143, 'artwork': 144, 'refined': 145, 'u': 156, 'hop': 157, 'At': 158, 'first': 159, 'opinion': 160, 'always': 161, 'siblin': 173, 'got': 174, 'Linda': 175, 'built': 176, 'quickly': 177, 'She': 178, 'want': 191, 'needed': 192, 'in': 193, 'order': 194, 'be': 195, 'friends': 196, 'w': 208, 'asleep': 209, 'awake': 210, 'shelf': 211, 'kitchen': 212, 'Can': 213, 'for': 226, 'They': 227, 'all': 228, 'cheap': 229, 'except': 230, 'which': 231, '242, 'local': 243, 'restaurants': 244, 'NYC': 245, 'celebrities': 246, 'often': 259, 'never': 260, 'afford': 261, 'Not': 262, "I'm": 263, 'pretty': 264, 'n': 275, 'States': 276, 'Most': 277, 'companies': 278, 'conference': 279, 'or': 292, 'loves': 292, 'action': 293, 'movies': 294, 'out': 295, 'know': 296, 'love': 297, 'get': 310, 'some': 311, 'restaurant': 312, 'school': 313, 'popular': 314, 'busy': 326, 'player': 327, 'team': 328, 'skills': 329, 'lesson': 330, 'cost': 331, '$50': 343, 'sport': 344, 'its': 345, 'pros': 346, 'cons': 347, 'must': 348, 'talk': 360, 'sent': 361, 'someone': 362, 'represent': 363, 'parking': 364, 'space': 376, 'beautiful': 377, 'sapphire': 378, 'necklace': 379, 'gift': 380, 'In': 392, 'think': 392, 'positively': 393, 'day': 394, 'new': 395, 'full': 396, 'opportunities': 407, 'according': 408, 'major': 409, 'box': 410, 'contained': 411, 'something': 412
```

単語idから単語を返すメソッド

単語id -> 単語

```
def get_key_from_value(d, val):  
    keys = [k for k, v in d.items() if v == val]  
    if keys:  
        return keys[0]  
    return None
```

get_key_from_value(word, 単語id) → 単語

学習

```
def Train():
```

```
    # エPOCH数
```

```
    EPOCH = 100
```

```
    for e in range(EPOCH):
```

```
        # 文脈層の初期化
```

```
        context = np.zeros( (1,hunit_num) )
```

```
        # 誤差二乗和
```

```
        error = 0.0
```

```
        for s in range(len(sentence)):
```

```
            if sentence[s] == eos_id:
```

```
                # 文脈層の初期化
```

```
                context = np.zeros( (1,hunit_num) )
```

```
                continue
```

```
            # 入力層への入力 (one-hot vector)
```

```
            input_data = np.zeros( (1,feature) )
```

```
            input_data[0][ sentence[s] ] = 1
```

eos_id
文の区切り

eos_idが現れた場合,
次の文の学習
文脈層を初期化(重要です)

sentence[s]
現在の単語id

(0, 1, 0, 0, 0, ..., 0)

埋め込み層への入力

```
embed.Propagation( input_data )
```

埋め込み層と文脈層の値を結合

```
hunit_in = np.hstack((embed.out,context))
```

中間層への入力

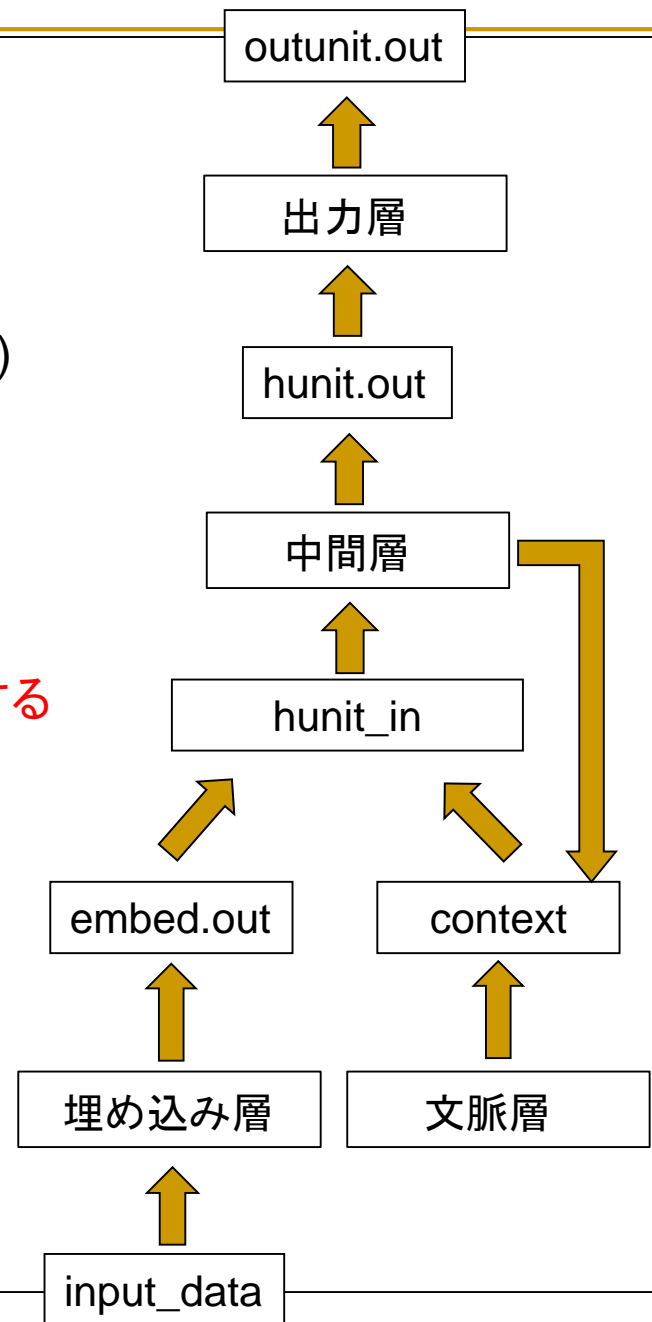
```
hunit.Propagation( hunit_in )
```

現在の中間層の値を次の文脈層の値とする

```
context = hunit.out
```

出力層の入力

```
outunit.Propagation( hunit.out )
```



教師信号 (one-hot vector)

```
teach = np.zeros( (1,feature) )  
teach[0][ sentence[s+1] ] = 1
```

(0, 0, 0, **1**, 0, ..., 0)

sentence[s+1]
次の単語id

誤差の計算

```
outunit.Error( teach )
```

出力層

```
hunit.Error( outunit.error )
```

中間層

```
work = np.reshape( hunit.error[0,0:embed_num] , (1,embed_num) )
```

```
embed.Error( work )
```

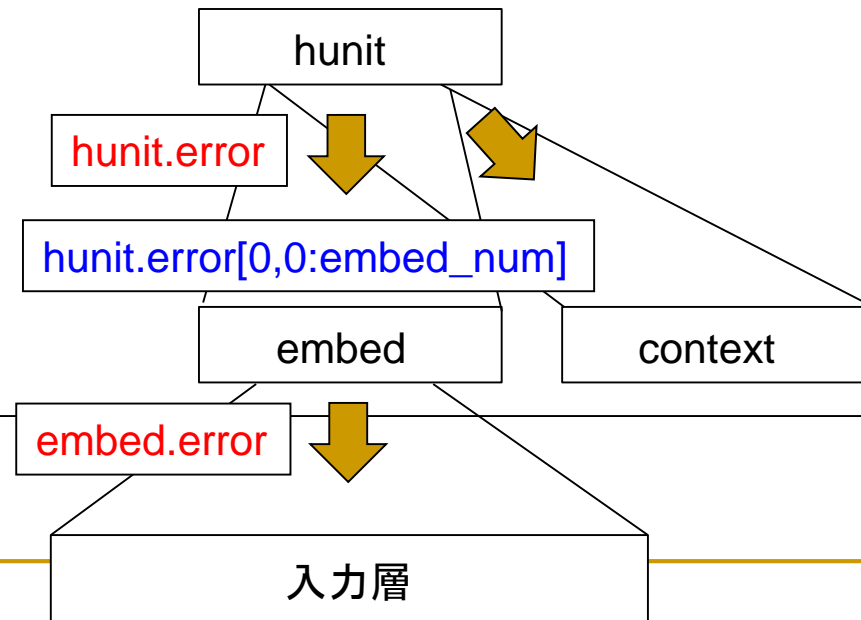
埋め込み層

重みの修正

```
outunit.Update_weight()
```

```
hunit.Update_weight()
```

```
embed.Update_weight()
```



学習

誤差二乗和の計算

```
error += np.dot( ( outunit.out[0] - teach[0] ) , ( outunit.out[0] - teach[0] ) )  
print( e , "->" , error )
```

重みの保存

```
outunit.Save( "dat/Elman-out.npz" )  
hunit.Save( "dat/Elman-hunit.npz" )  
embed.Save( "dat/Elman-embed.npz" )
```

出力層

中間層

埋め込み層

予測

```
def Predict():
```

```
    # 重みのロード
```

```
    outunit.Load( "dat/Elman-out.npz" )
```

出力層

```
    huntunit.Load( "dat/Elman-hunit.npz" )
```

中間層

```
    embed.Load( "dat/Elman-embed.npz" )
```

埋め込み層

```
    # 文脈層の初期化
```

```
    context = np.zeros( (1,hunit_num) )
```

```
    for s in range(len(sentence)):
```

```
        if sentence[s] == eos_id:
```

eos_id
文の区切り

```
            print( "-----" )
```

eos_idが現れた場合,
次の文の学習
文脈層を初期化

```
            # 中間層の初期化
```

```
            context = np.zeros( (1,hunit_num) )
```

```
            continue
```

入力層への入力

```
input_data = np.zeros( (1,feature) )  
input_data[0][ sentence[s] ] = 1
```

sentence[s]
現在の単語id

(0, 1, 0, 0, 0, ..., 0)

埋め込み層への入力

```
embed.Propagation( input_data )
```

埋め込みと文脈層の値を結合

```
hunit_in = np.hstack((embed.out,context))
```

中間層への入力

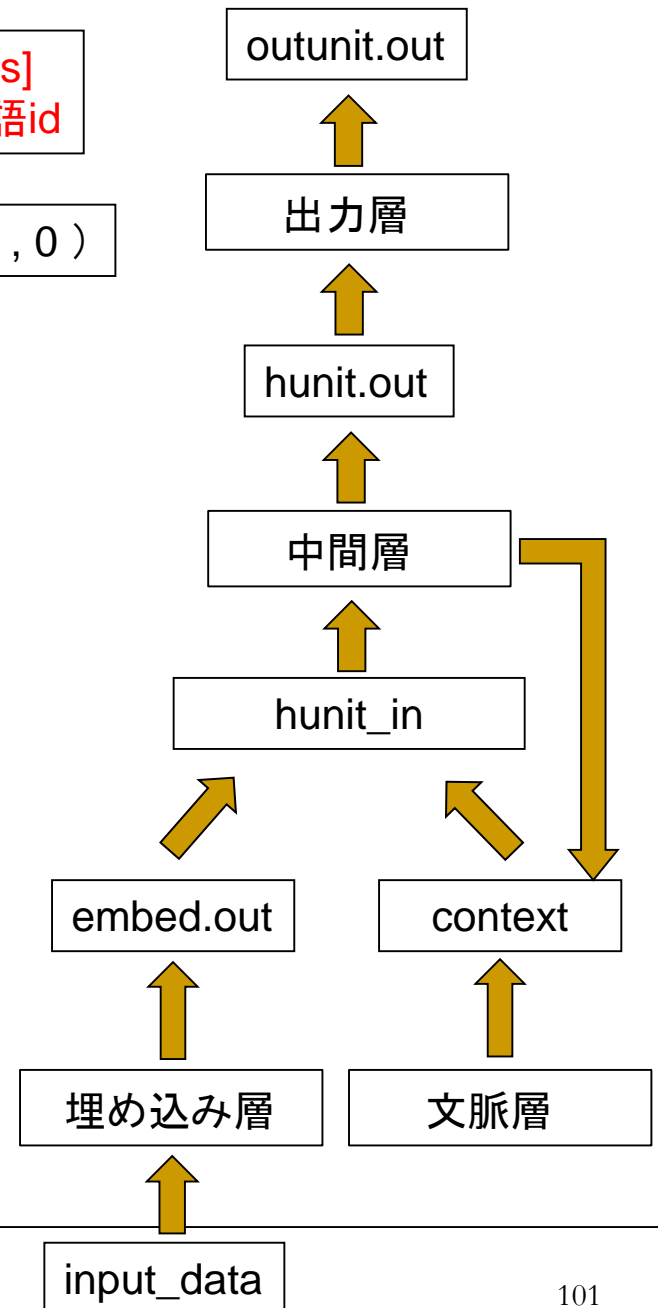
```
hunit.Propagation( hunit_in )
```

現在の中間層の値を次の文脈層の値とする

```
context = hunit.out
```

出力層への入力

```
outunit.Propagation( hunit.out )
```



予測

予測

```
predict = np.argmax( outunit.out[0] )
```

予測結果の出力

```
print( get_key_from_value(word, sentence[s] ) , "->" ,  
       get_key_from_value(word, sentence[s+1] ) ,  
       "[" , get_key_from_value(word, predict) + " ]" )
```

`get_key_from_value(word, 単語id) → 単語`

実行①(学習)

> python Elman.py t

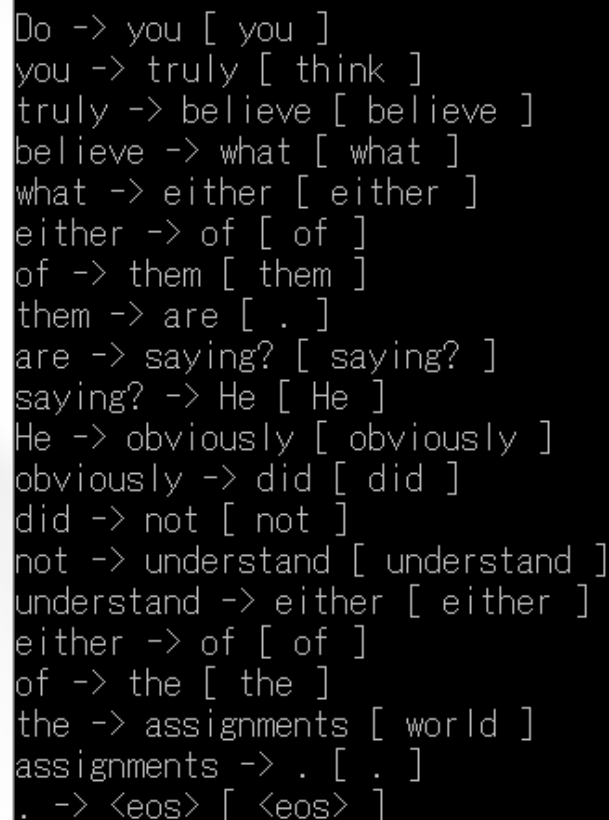


```
C:\windows\system32\cmd.exe
73 -> 759.229220435981
74 -> 748.3625079322416
75 -> 737.7879375601882
76 -> 727.4914545347976
77 -> 717.4597378219287
78 -> 707.6802383905368
79 -> 698.1411857653411
80 -> 688.8315721994144
81 -> 679.7411221908728
82 -> 670.8602535715478
83 -> 662.1800350773157
84 -> 653.69214408232
85 -> 645.3888269580896
86 -> 637.2628633195469
87 -> 629.3075343584844
88 -> 621.5165946816235
89 -> 613.8842466507681
90 -> 606.4051161563405
91 -> 599.0742289500557
92 -> 591.8869869877237
93 -> 584.8391445707406
94 -> 577.9267843447365
95 -> 571.146293382053
96 -> 564.494339641862
97 -> 557.9678490894995
98 -> 551.5639836932628
99 -> 545.2801204302971
```

誤差二乗和

実行②(予測*)

> python Elman.py p



```
-----  
Do -> you [ you ]  
you -> truly [ truly ]  
truly -> believe [ believe ]  
believe -> what [ what ]  
what -> either [ either ]  
either -> of [ of ]  
of -> them [ them ]  
them -> are [ . ]  
are -> saying? [ saying? ]  
saying? -> He [ He ]  
He -> obviously [ obviously ]  
obviously -> did [ did ]  
did -> not [ not ]  
not -> understand [ understand ]  
understand -> either [ either ]  
either -> of [ of ]  
of -> the [ the ]  
the -> assignments [ world ]  
assignments -> . [ . ]  
. -> <eos> [ <eos> ]
```

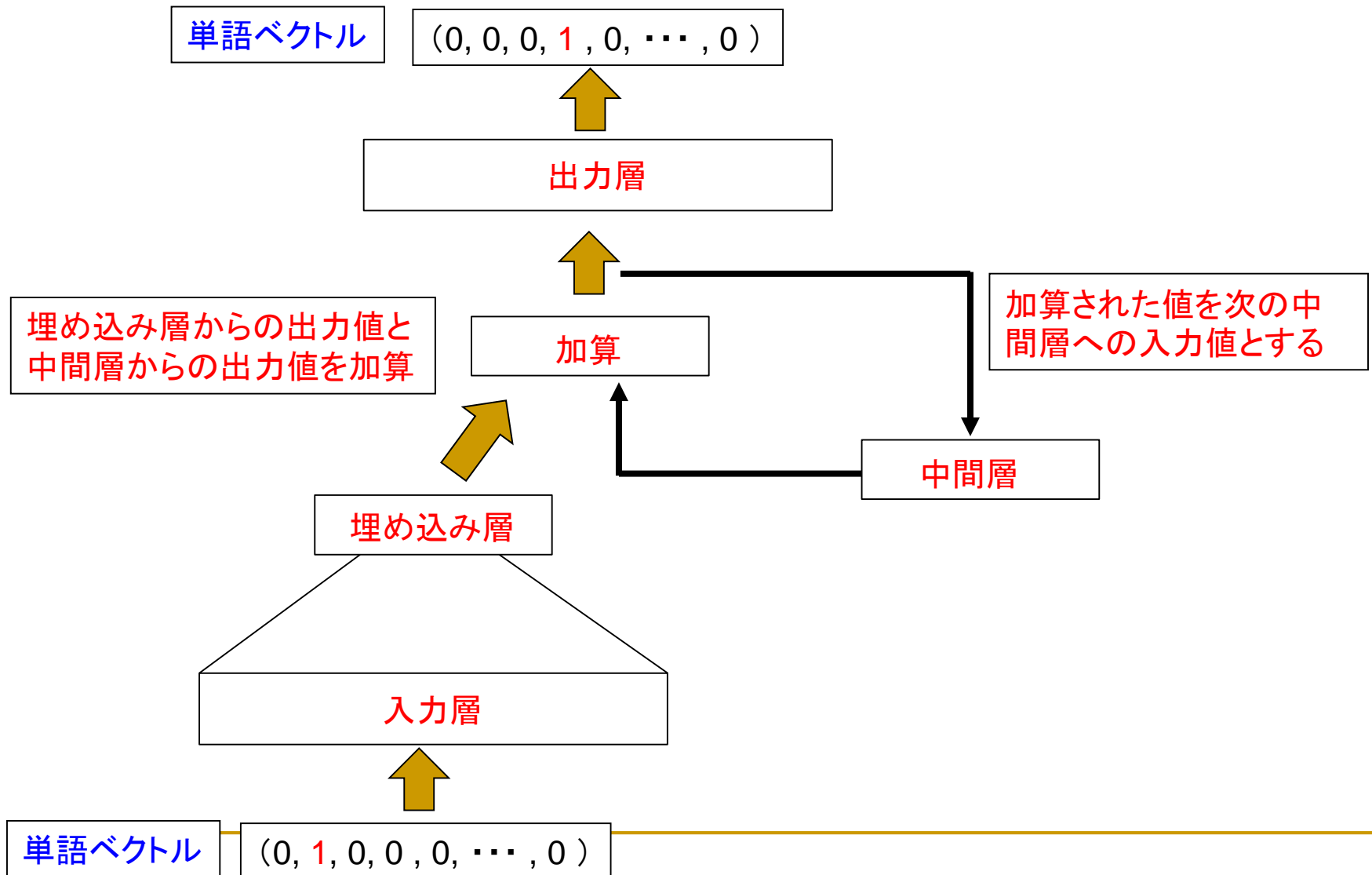
現在の単語→次の単語 [予測した単語]

*学習データの予測です. テストデータの予測についてはより工夫しなければ予測できません

宿題⑪

- 次頁に示す構造のリカレントネットワークを用いて、語系列課題学習を行なって下さい。
- 埋め込み層からの出力値と中間層からの出力値を加算し、出力層に入力します。また加算された値を次の中間層への入力値とします。
- 語系列課題学習を行ない、学習データに対して、次単語が予測できているか確認して下さい。

宿題⑪



参考文献

- J.デイホフ:ニューラルネットワークアーキテクチャ入門, 森北出版(1992)
- P.D.Wasserman:ニューラル・コンピューティング, 理論と実際, 森北出版(1993)
- 都築誉史編:高次認知のコネクショニストモデル, 共立出版(2005)
- 岡谷貴之:深層学習, 講談社(2015)
- 瀧雅人:これならわかる深層学習入門, 講談社(2017)
- 坪井祐太他:深層学習による自然言語処理, 講談社(2017)