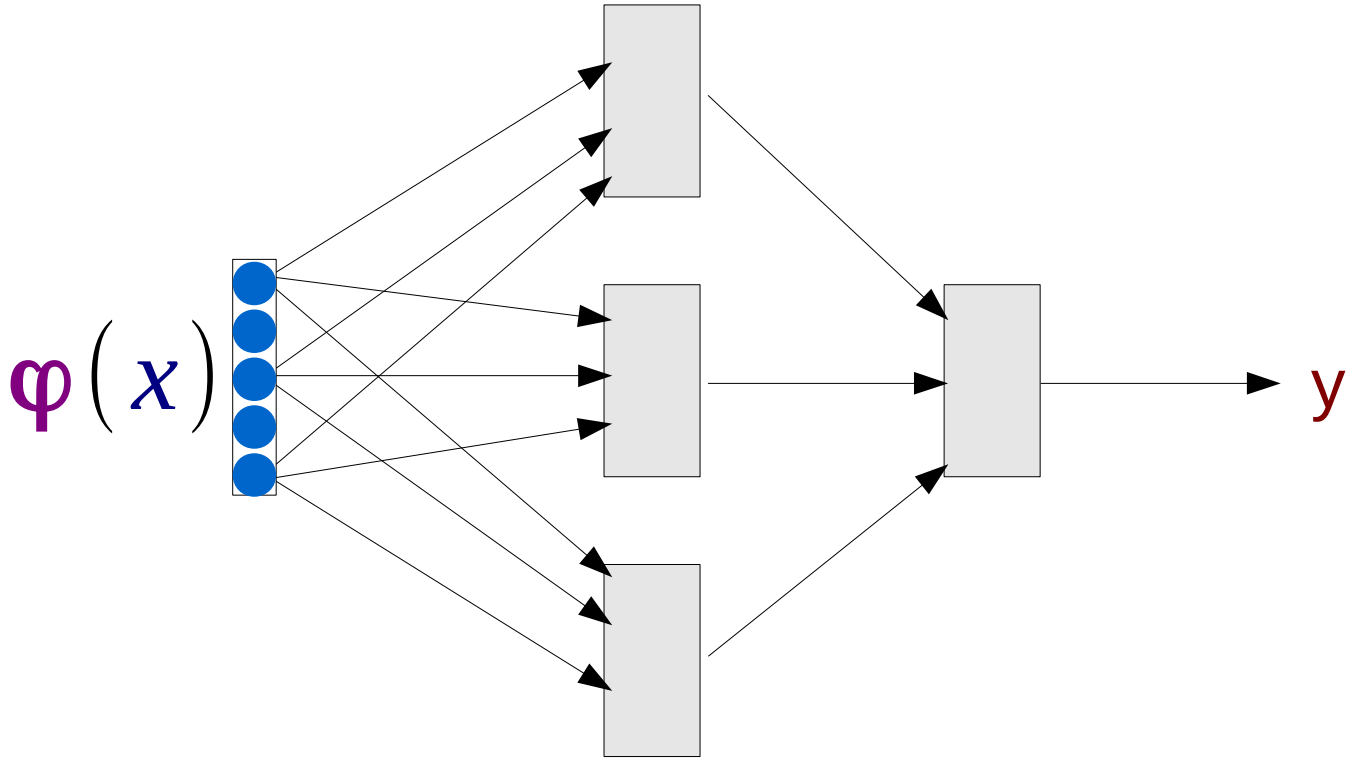
自然言語処理プログラミング勉強会 8
リカレントニューラルネット

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

フィードフォーワード ニューラルネット

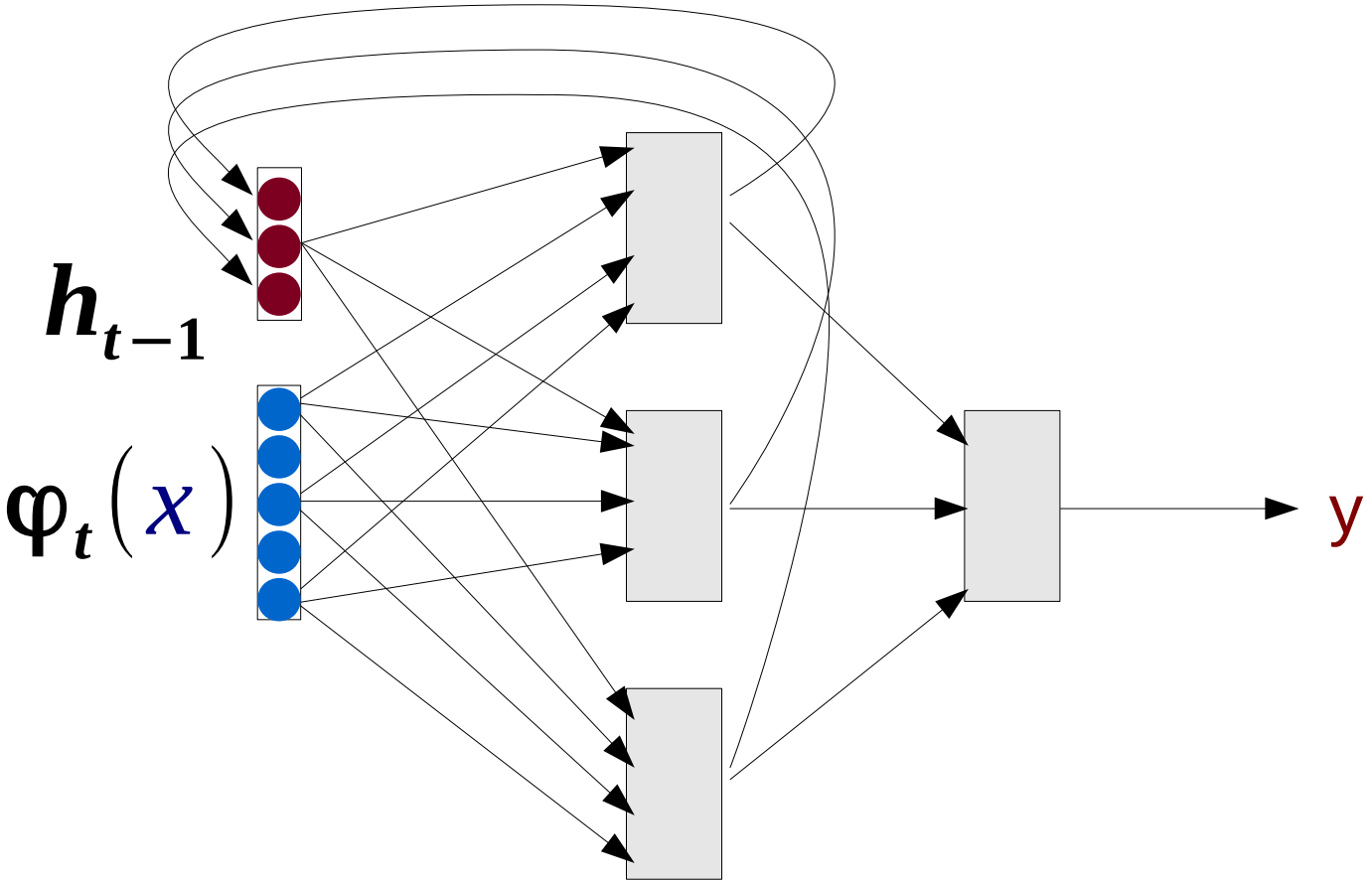
- すべての線が前方向のネット



- (有効非巡回グラフ DAG になっている)

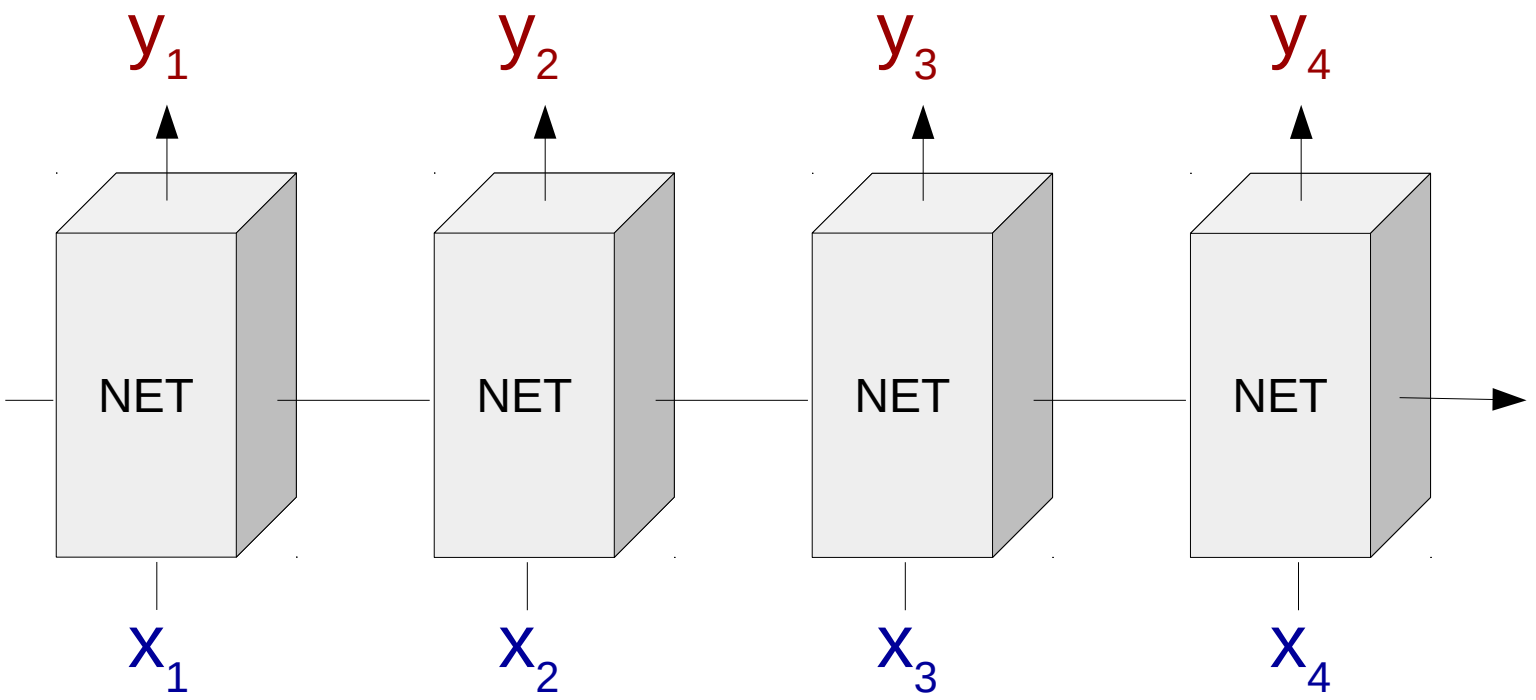
リカレントニューラルネット (RNN)

- ノードの一部の出力が入力として戻ってくる

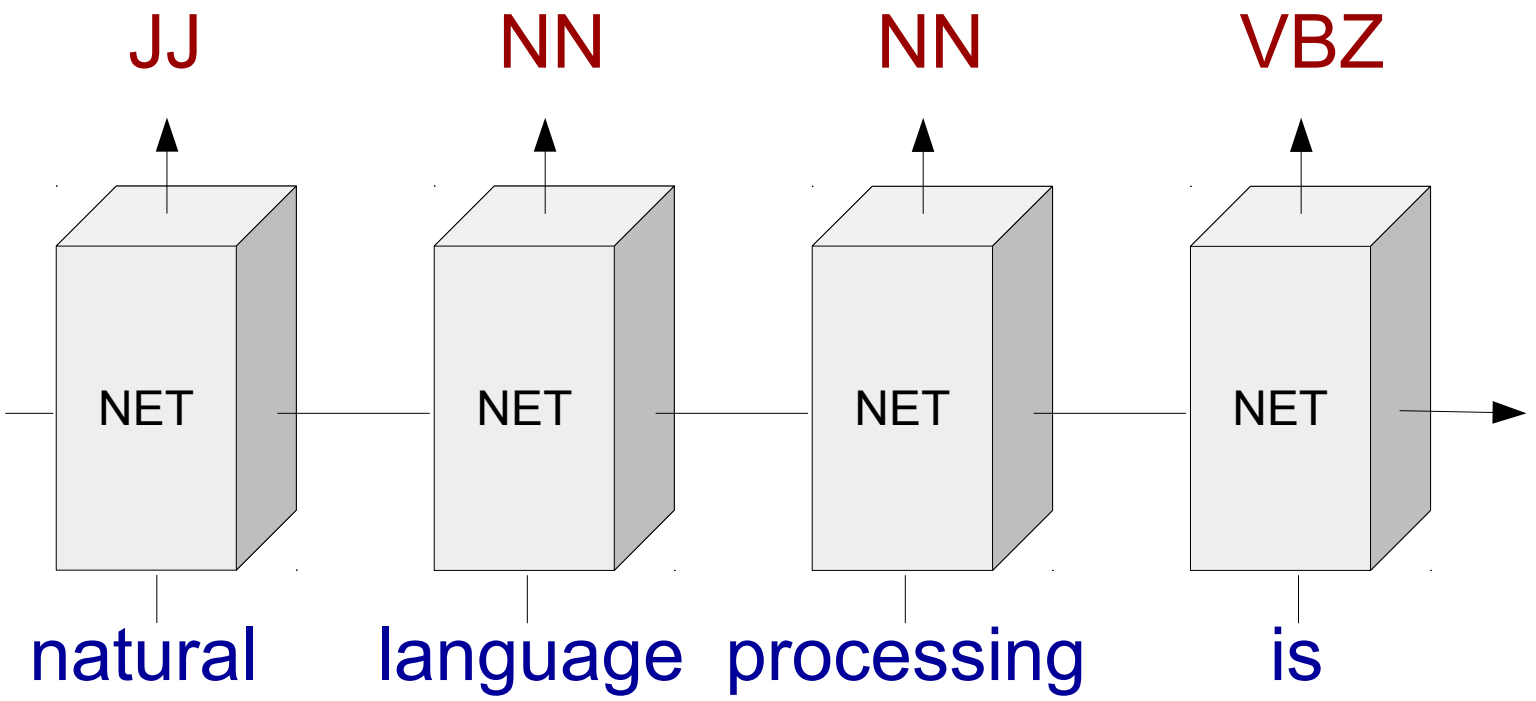


- 理由：「記憶」が可能

系列モデルとしての RNN



例：品詞推定



ニューラルネットを使った多クラス予測

復習：予測問題

x が与えられた時　y を予測する

本のレビュー	「良い」評価なのか？
Oh, man I love this book!	yes
This book is so boring...	no

２値予測
(選択肢が２つ)

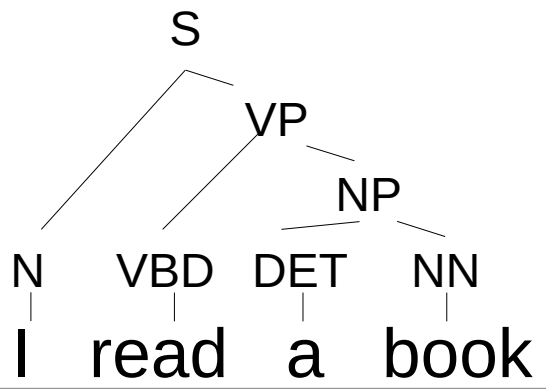
ツイート	書かれた言語
On the way to the park!	English
公園に行くなう！	Japanese

多クラス予測
(選択肢が数個)

文

I read a book

構文木

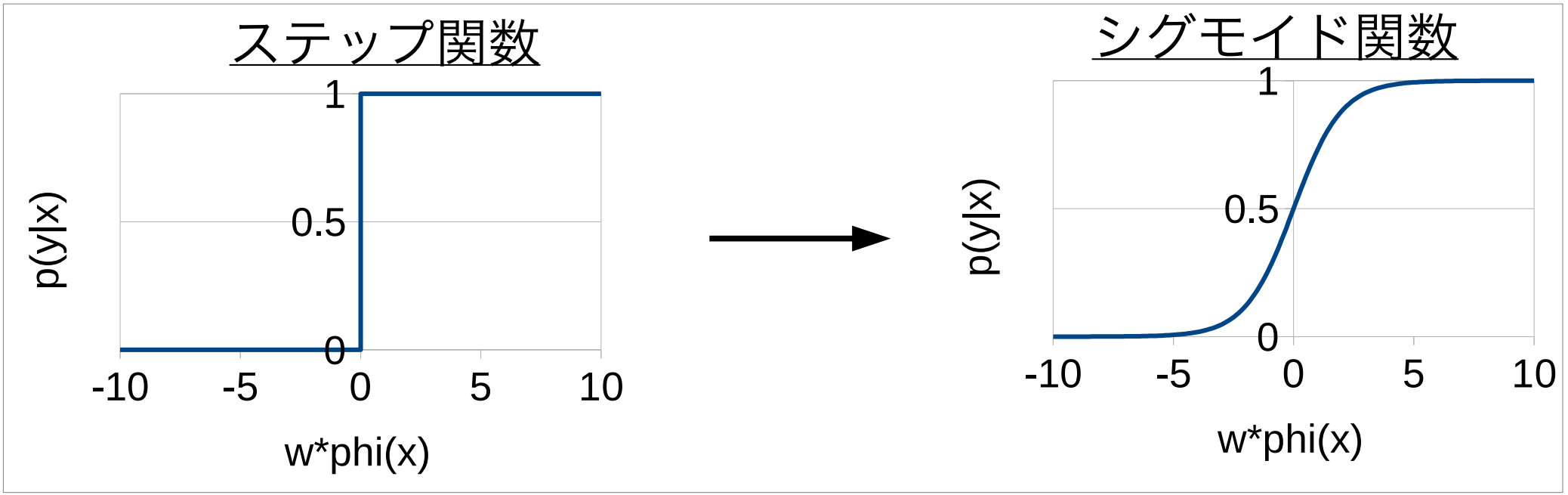


構造化予測
(選択肢が膨大)

復習：シグモイド関数

- シグモイド関数はステップ関数を柔らかくしたもの

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$



softmax 関数

- シグモイド関数を複数のクラスに拡張したもの

$$P(y|x) = \frac{e^{\boldsymbol{w}\cdot\boldsymbol{\varphi}(x,y)}}{\sum_{\tilde{y}} e^{\boldsymbol{w}\cdot\boldsymbol{\varphi}(x,\tilde{y})}}$$

← 今のクラス

← すべてのクラスの和

- ベクトル・行列の掛け算で表現

$$\boldsymbol{r} = \exp\left(\boldsymbol{W}\cdot\boldsymbol{\varphi}(x)\right)$$

$$\boldsymbol{p} = \boldsymbol{r} / \sum_{\tilde{r}\in\boldsymbol{r}} \tilde{r}$$

確率分布から最大値の選択

- 確率が一番高いインデックス *y* を探す

```
find_best(p):
    y = 0
    for each element i in 1 .. len(p)-1:
        if p[i] > p[y]:
            y = i
    return y
```

softmax 関数の勾配

- 正解の確率分布からモデル推定の確率分布を引いた値

$$-d\,\mathrm{err}/d\,\varphi_{out} = \boldsymbol{p}' - \boldsymbol{p}$$

- 正解 **p'** は y 番目の要素だけが 1 のベクトル (one-hot ベクトル)

$$\boldsymbol{p}' = \{0, 0, \ldots, 1, \ldots, 0\}$$

one-hot ベクトルの作成

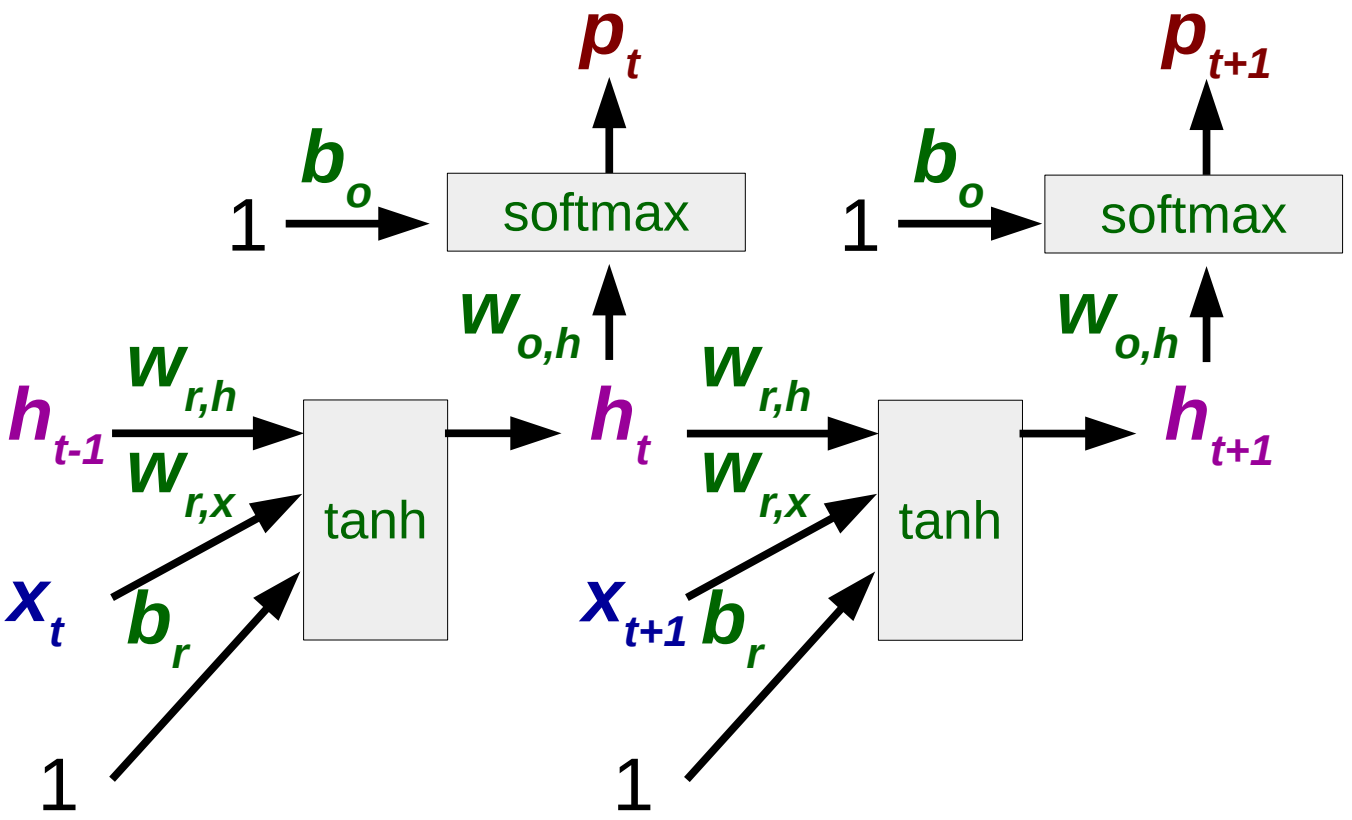
```
create_one_hot(id, size):
    vec = np.zeros(size)
    vec[id] = 1
    return vec
```

リカレントニューラルネットの伝搬

復習：ニューラルネットの計算

```
forward_nn(network, φ_0)
    φ= [ φ_0 ] #  各層の値
    for each layer i in 0 .. len(network)-1:
        w, b = network[i]
        #  前の層の値に基づいて値を計算
        φ[i] = np.tanh( np.dot( w, φ[i-1] ) + b )
    return φ #  各層の結果を返す
```

RNN 計算式



$$h_t = \tanh\left(w_{r,h} \cdot h_{t-1} + w_{r,x} \cdot x_t + b_r\right)$$

$$p_t = \mathrm{softmax}\left(w_{o,h} \cdot h_t + b_o\right)$$

RNN の前向き計算

```
forward_rnn(w_r,x, w_r,h, b_r, w_o,h, b_o, x)
    h = [ ] #  隠れ層の値（各時間 t において）
    p = [ ] #  出力の確率分布の値（各時間 t において）
    y = [ ] #  出力の確率分布の値（各時間 t において）
    for each time t in 0 .. len(x)-1:
        if t > 0:
            h[t] = tanh(w_r,x x[t] + w_r,h h[t-1] + b_r)
        else:
            h[t] = tanh(w_r,x x[t] + b_r)
        p[t] = tanh(w_o,h h[t] + b_o)
        y[t] = find_max(p[t])
    return h, p, y
```

復習：
フィードフォーワードネットの逆伝搬

確率的勾配降下法 (stochastic gradient descent; SGD)

- ロジスティック回帰を含む確率モデルのための学習アルゴリズム

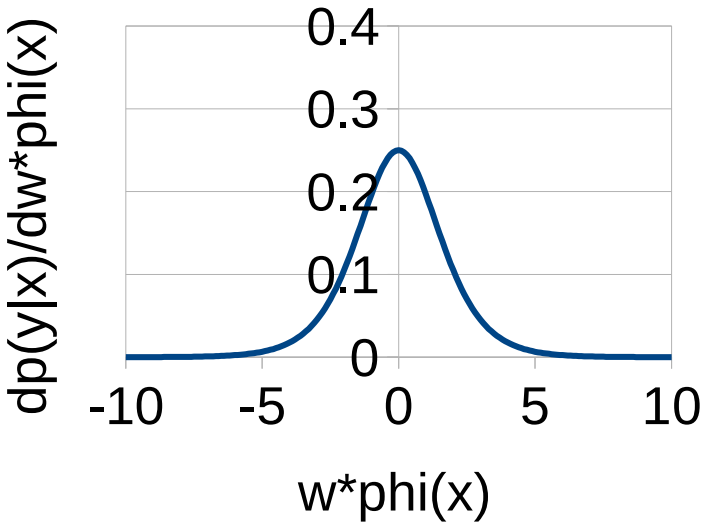
```
w = 0
for / iterations
    for each labeled pair x, y in the data
        w += α * dP(y|x)/dw
```

- つまり
  - 各学習例に対して勾配を計算（y の確率が上がる方向）
  - その方向へ、学習率 α をかけた分だけ動かす

シグモイド関数の勾配

- 確率の微分

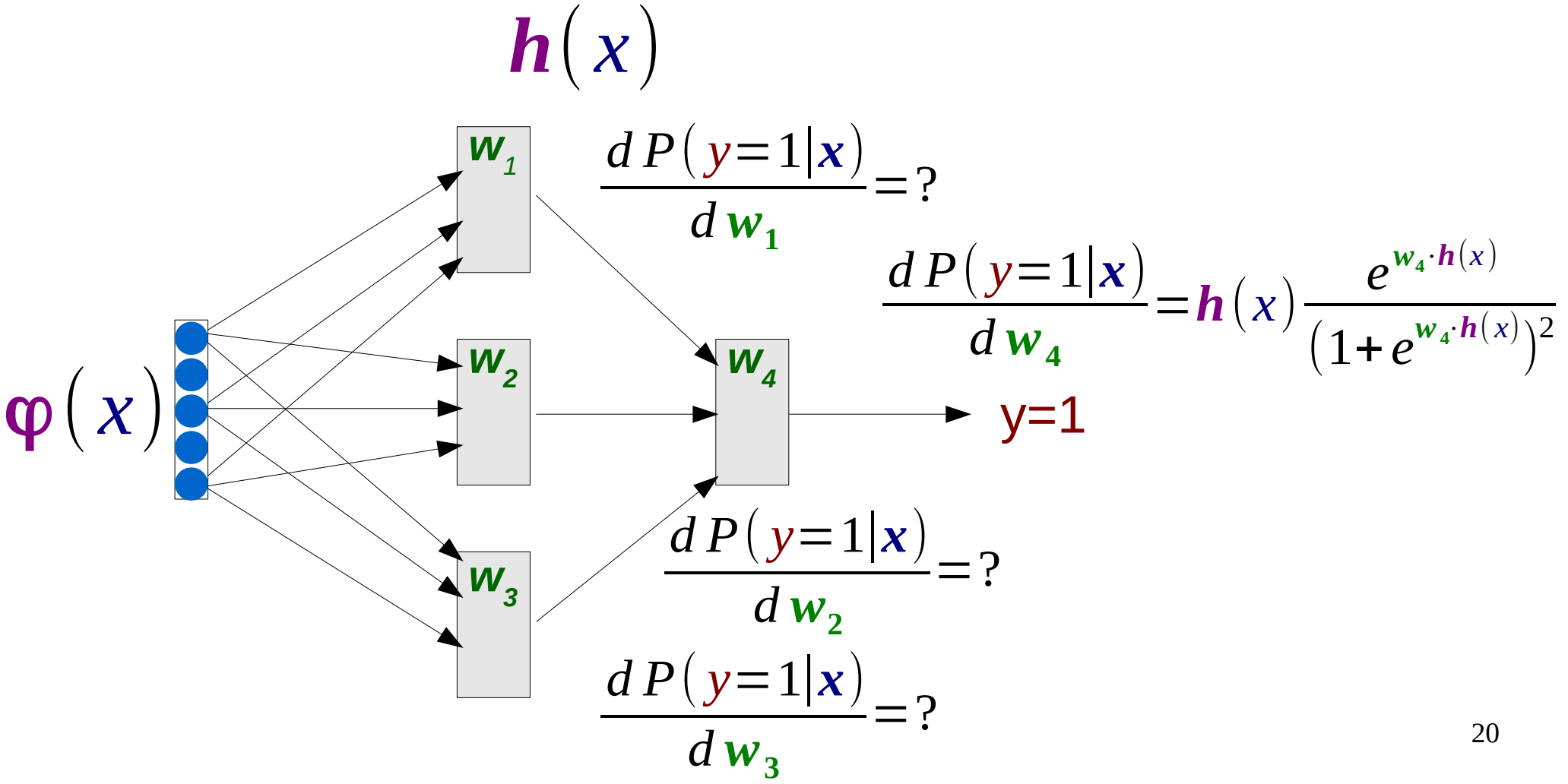
$$\frac{d}{dw}P(y=1|x) = \frac{d}{dw}\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}} = \varphi(x)\frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$



$$\frac{d}{dw}P(y=-1|x) = \frac{d}{dw}\left(1-\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}\right) = -\varphi(x)\frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$

学習：隠れ層の勾配が分からない

- 出力層のタグしか分からない



逆伝搬 (back propagation)

- 連鎖律を使って計算

$$\frac{dP(y=1|x)}{dw_1} = \frac{dP(y=1|x)}{dw_4 h(x)} \frac{dw_4 h(x)}{dh_1(x)} \frac{dh_1(x)}{dw_1}$$

$$\frac{dP(y=1|x)}{dw_4 h(x)} = \frac{e^{w_4 \cdot h(x)}}{\left(1+e^{w_4 \cdot h(x)}\right)^2}$$

次の層のエラー (δ_4)

$$\frac{dh_1(x)}{dw_1} = w_{1,4}$$

重み

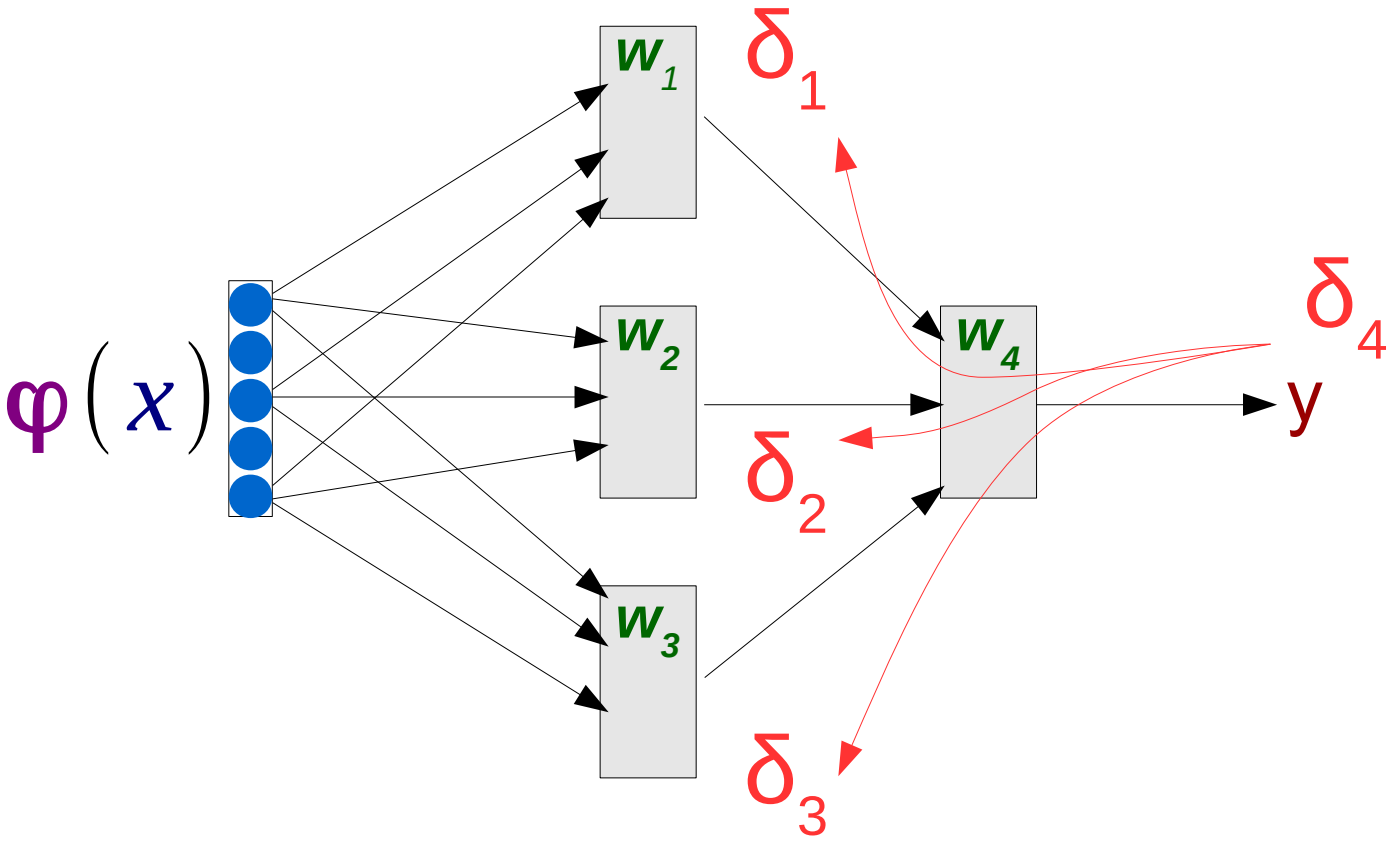
勾配

一般には
iを次の層の
jのエラーで計算

$$\frac{dP(y=1|x)}{w_i} = \frac{dh_i(x)}{dw_i} \sum_j \delta_j w_{i,j}$$

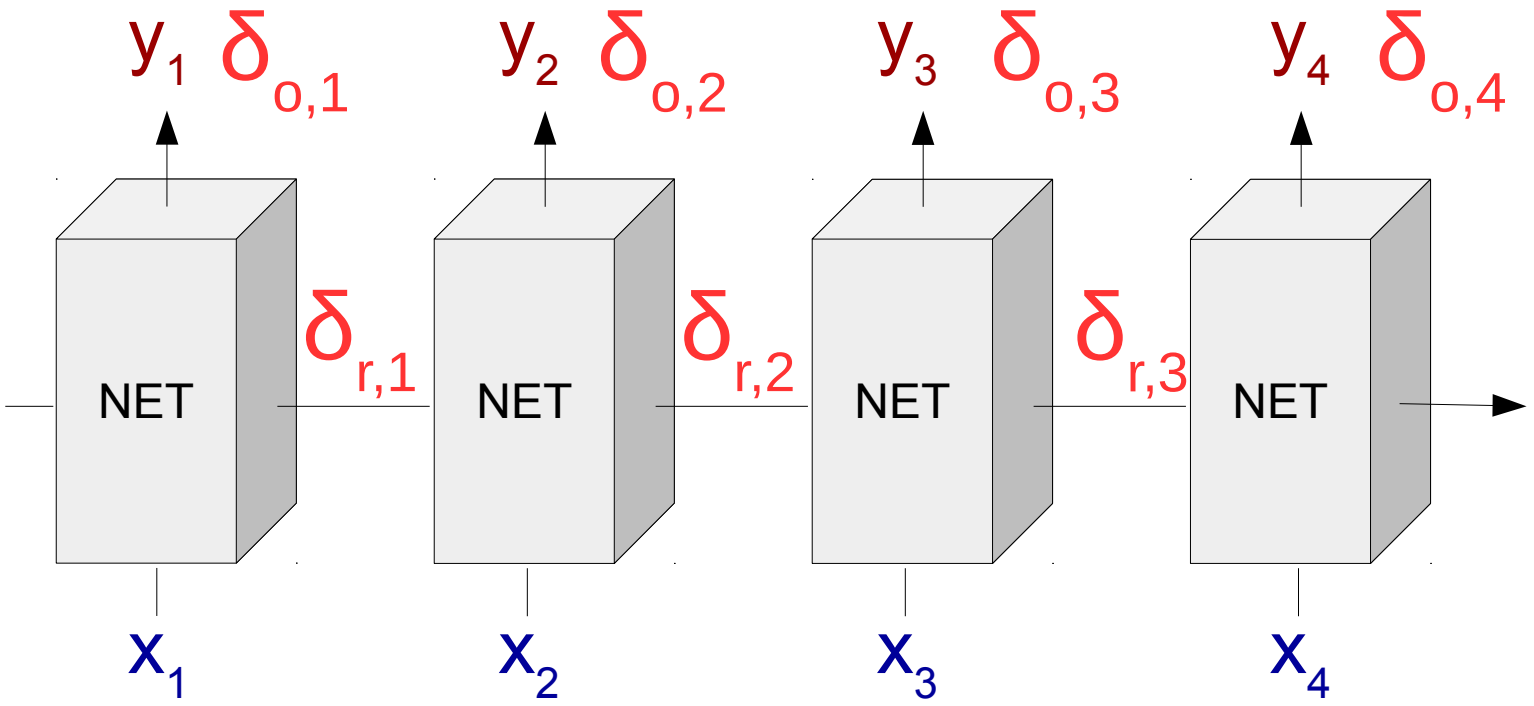
逆伝搬の概念図

- エラーを逆方向に伝搬



リカレントネットの逆伝搬

計算できるエラーは？

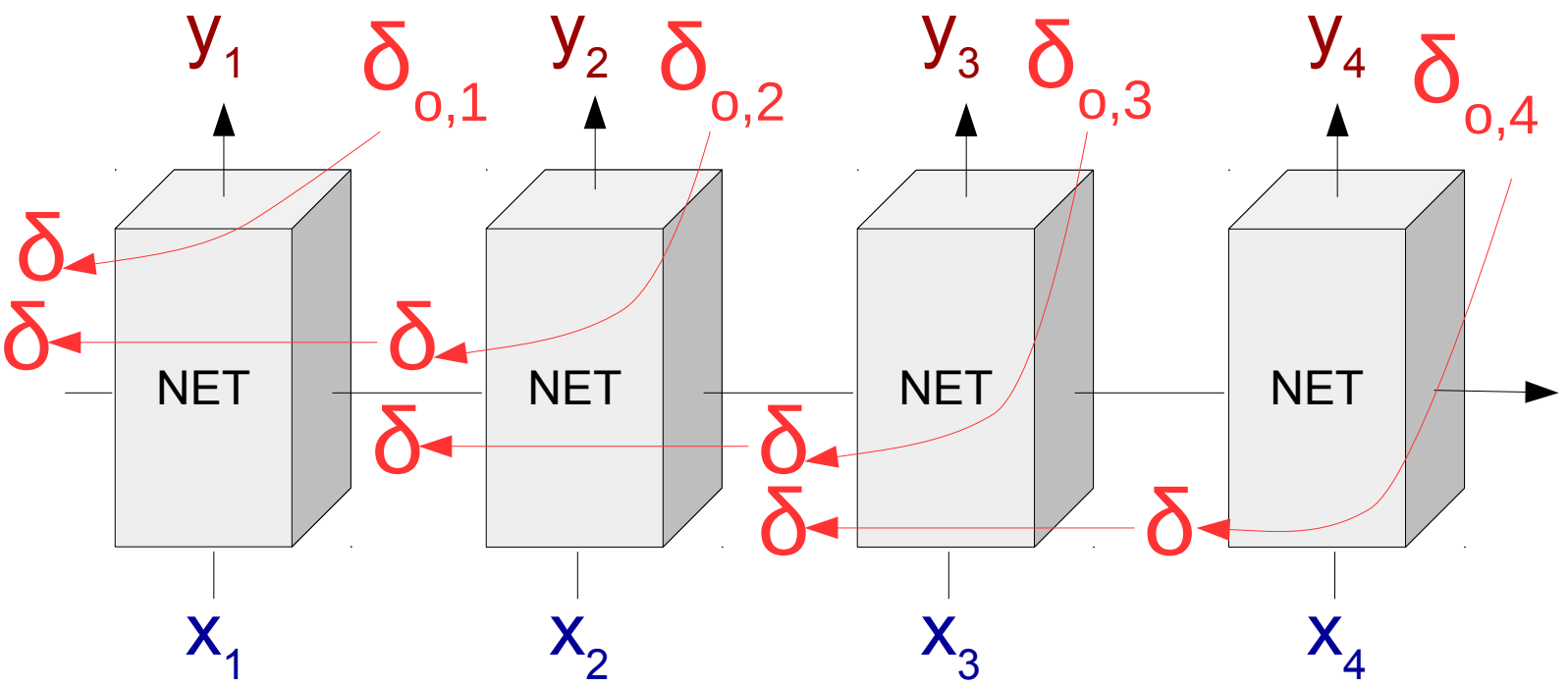


- 出力エラー δ_o は簡単に計算可能
- 次の時間からのリカレントエラー δ_r は逆伝搬

逆伝搬の方法

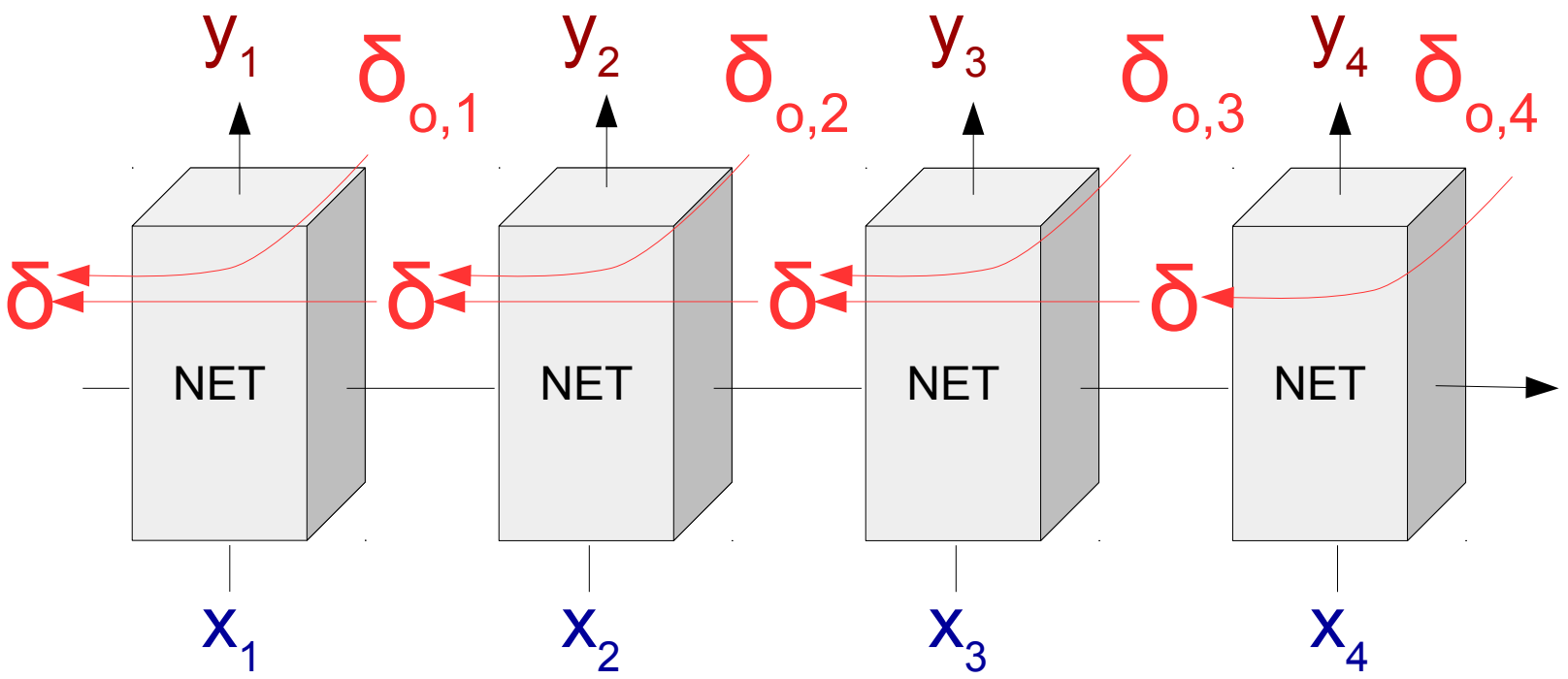
- 通時的逆伝播 (back propagation through time; BPTT)
  - 各 δ_o に対して n 回（時間）の逆伝搬を行う
- 完全勾配計算 (full gradient calculation)
  - 動的計画法に基づいて、系列全体の勾配を計算

通時的逆伝搬



- 1 つの出力エラーだけを考慮
- n 回の計算だけを行う (ここでは n=2)

完全勾配計算

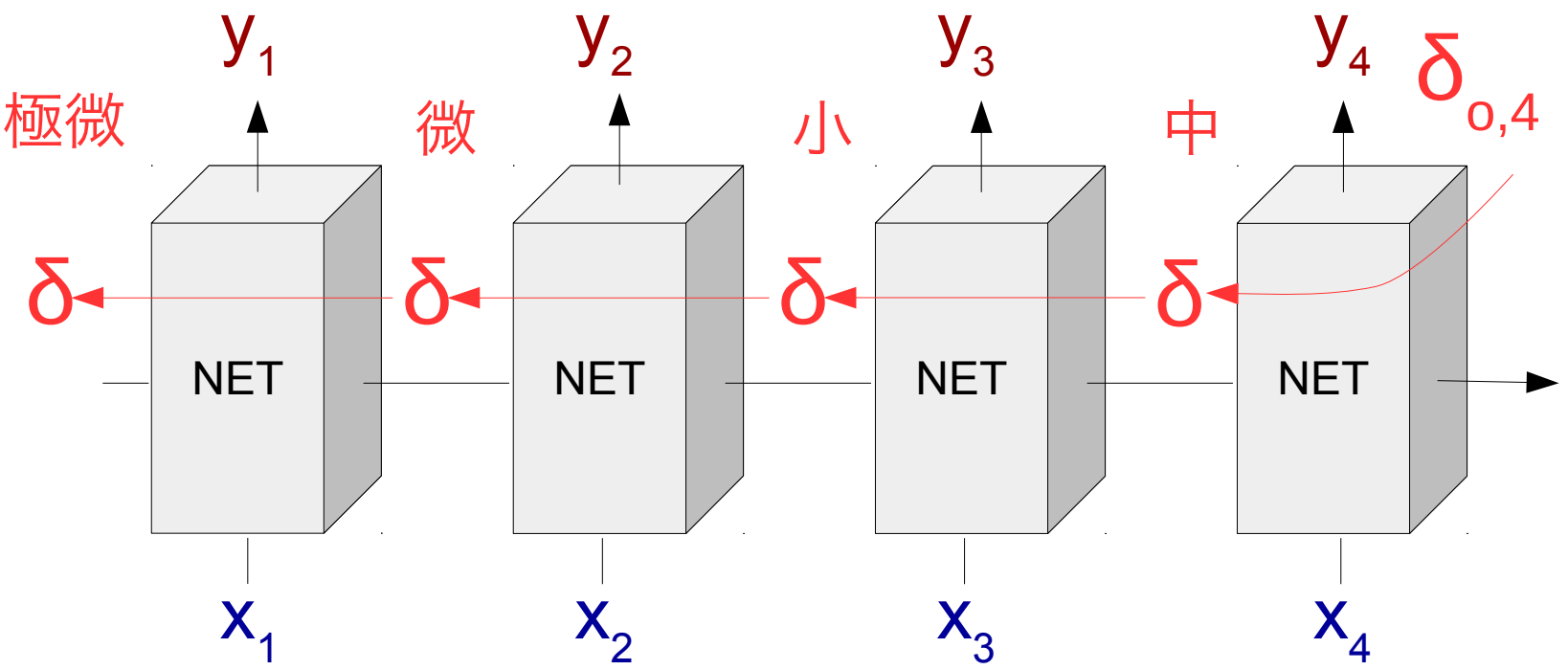


- まず系列のネット結果全体を計算
- 後ろからエラーを計算

BPTT? 完全勾配計算？

- 完全勾配計算
  - + 計算量が少ない、ステップ数の指定がない
  - - 系列全体の計算結果を保存するのが必要
- BPTT:
  - + メモリ量は系列の長さではなくステップ数に比例
  - - 計算量が多い、長距離の情報が捉えにくい

ニューラルネットにおける消える勾配



- 「Long Short Term Memory (LSTM)」などでこの問題に対処

RNN の完全勾配計算

```
gradient_rnn(w_r,x, w_r,h, b_r, w_o,h, b_o, x, h, p, y')
    initialize Δw_r,x, Δw_r,h, Δb_r, Δw_o,h, Δb_o
    δ_r' = np.zeros(len(b_r))  # 次の時間から伝搬するエラー
    for each time t in len(x)-1 .. 0:
        p' = create_one_hot(y'[t])
        δ_o' = p' – p                                          # 出力層エラー
        Δw_o,h += np.outer(h[t], δ_o'); Δb_o += δ_o'           # 出力層重み勾配
        δ_r = np.dot(δ'_r, w_r,h) + np.dot(δ'_o, w_o,h)        # 逆伝搬
        δ'_r = δ_r * (1 – h[t]^2)                              # tanh の勾配
        Δw_r,x += np.outer(x[t], δ_r'); Δb_r += δ_r'           # 隠れ層重み勾配
        if t != 0:
            Δw_r,h += np.outer(h[t-1], δ_r');
    return Δw_r,x, Δw_r,h, Δb_r, Δw_o,h, Δb_o
```

重み更新

update_weights($w_{r,x}$, $w_{r,h}$, b_r, $w_{o,h}$, b_o, $\Delta w_{r,x}$, $\Delta w_{r,h}$, Δb_r, $\Delta w_{o,h}$, Δb_o, λ)

$w_{r,x} \mathrel{+}= \lambda * \Delta w_{r,x}$

$w_{r,h} \mathrel{+}= \lambda * \Delta w_{r,h}$

$b_r \mathrel{+}= \lambda * \Delta b_r$

$w_{o,h} \mathrel{+}= \lambda * \Delta w_{o,h}$

$b_o \mathrel{+}= \lambda * \Delta b_o$

学習の全体像

```
# 素性を作り、ネットワークをランダムな値で初期化
create map x_ids, y_ids, array data
for each labeled pair x, y in the data
    add (create_ids(x, x_ids), create_ids(y, y_ids) ) to data
initialize net randomly

# 学習を行う
for I iterations
    for each labeled pair x, y' in the feat_lab
        h, p, y = forward_rnn(net, φ0)
        Δ= gradient_rnn(net, x, h, y')
        update_weights(net, Δ, λ)

print net to weight_file
print x_ids, y_ids to id_file
```

演習問題

演習問題

- 系列ラベリングのための RNN を実装
- 学習 train-rnn とテスト test-rnn
- テスト：品詞推定のものを利用
  - 入力： test/05-{train,test}-input.txt
  - 正解： test/05-{train,test}-answer.txt
- data/wiki-en-train.norm_pos を使ってモデルを学習し、 data/wiki-en-test.norm に対して品詞推定を行う
- 品詞推定の性能を評価し、 HMM と比較：
  script/gradepos.pl data/wiki-en-test.pos my_answer.pos