深層学習による系列モデル

概念と実装

Graham Neubig

カーネギーメロン大学 言語技術研究所

Language
Technologies
Institute

2017年6月16日

音響学会技術講習会

ニューラルネットの現状

- ニューラルネットはもはや欠かせない存在
- と同時にモデルは徐々に複雑化
- 実装したい機能をなるべく簡単にバグのないように実装したい
- 本講習では、音声・言語に関連するモデルと、その実装例を紹介

第１部：

ニューラルネットの実装

- ニューラルネットと計算グラフ
- DyNetツールキットの紹介
- 多層パーセプトロンによる文字認識の例
- 学習のパラメータ：最適化手法、ミニバッチ化、ネットの大きさ
- 畳み込みニューラルネット

第２部：
系列データの扱い

- 可変長データの分類問題

- bag of words

- リカレントニューラルネット、
  再帰的ニューラルネット

- タグの予測（例：品詞推定、音響モデル）

- 次の単語の予測（例：言語モデル）

第３部：
系列変換モデル

- encoder-decoder
- 解の探索
- attention (注意機構)

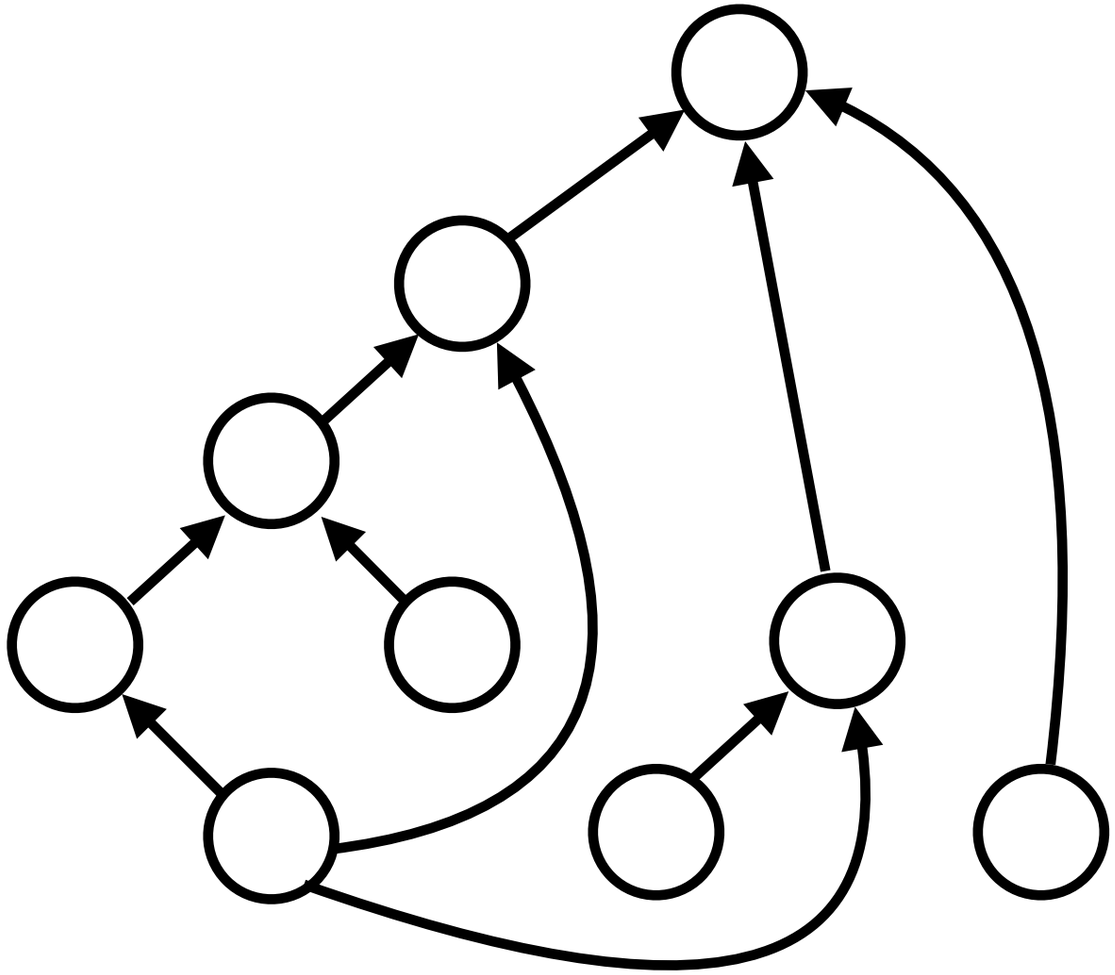
計算グラフ

深層学習の共通語

計算グラフ

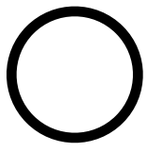
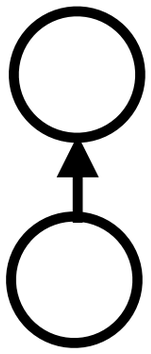
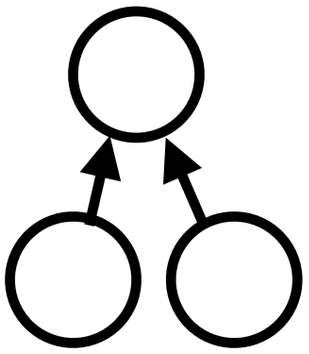
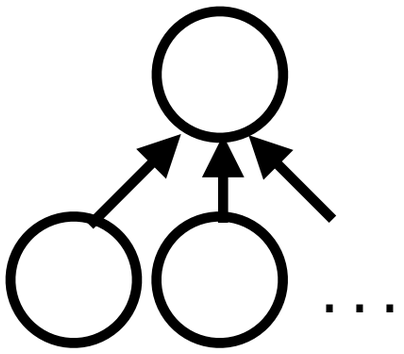
- **計算**を表す**有向非巡回グラフ**(DAG)

- **頂点**（ノード）と**辺**（エッジ）から構成される



頂点と辺

- **頂点**は**関数**とその**計算結果の値**を同時に表す
- **辺**は**関数の引数**と**頂点の依存関係**を同時に表す

ゼロ項演算	一項演算	二項演算	n項演算
			
パラメータ	$f(\mathbf{u}) = -\mathbf{u}$	$f(\mathbf{u},\mathbf{v}) = \mathbf{u}+\mathbf{v}$	
データ	$f(\mathbf{u}) = \mathbf{u}^{\mathsf{T}}$	$f(U,V) = UV$	

数式：

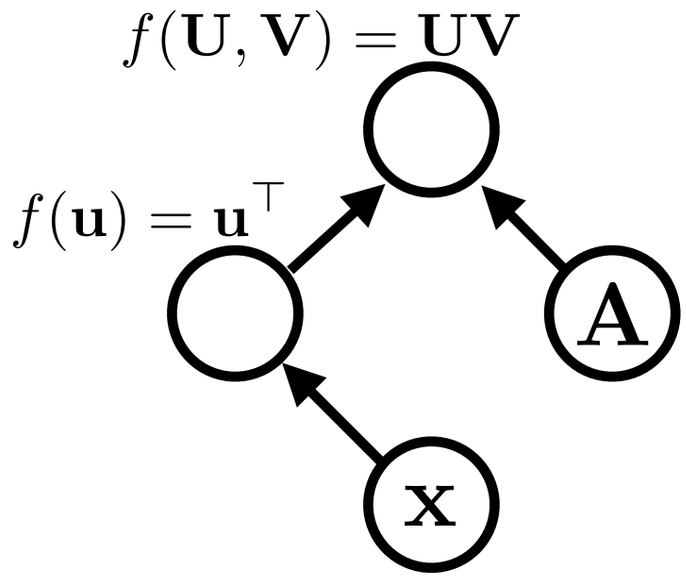
$$x$$

グラフ：

数式：

$$\mathbf{x}^\top \mathbf{A}$$

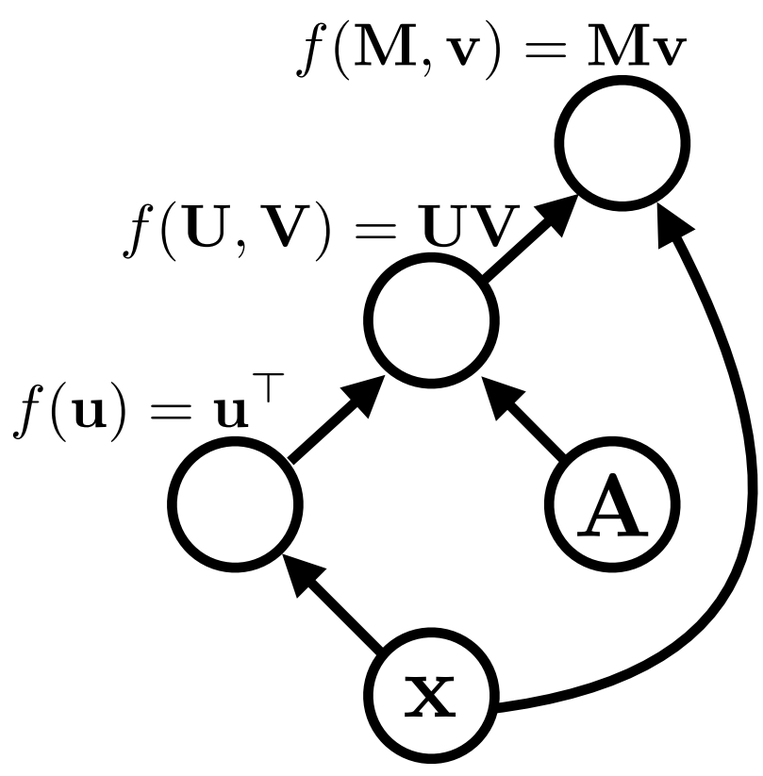
グラフ：



数式：

$$\mathbf{x}^\top \mathbf{A} \mathbf{x}$$

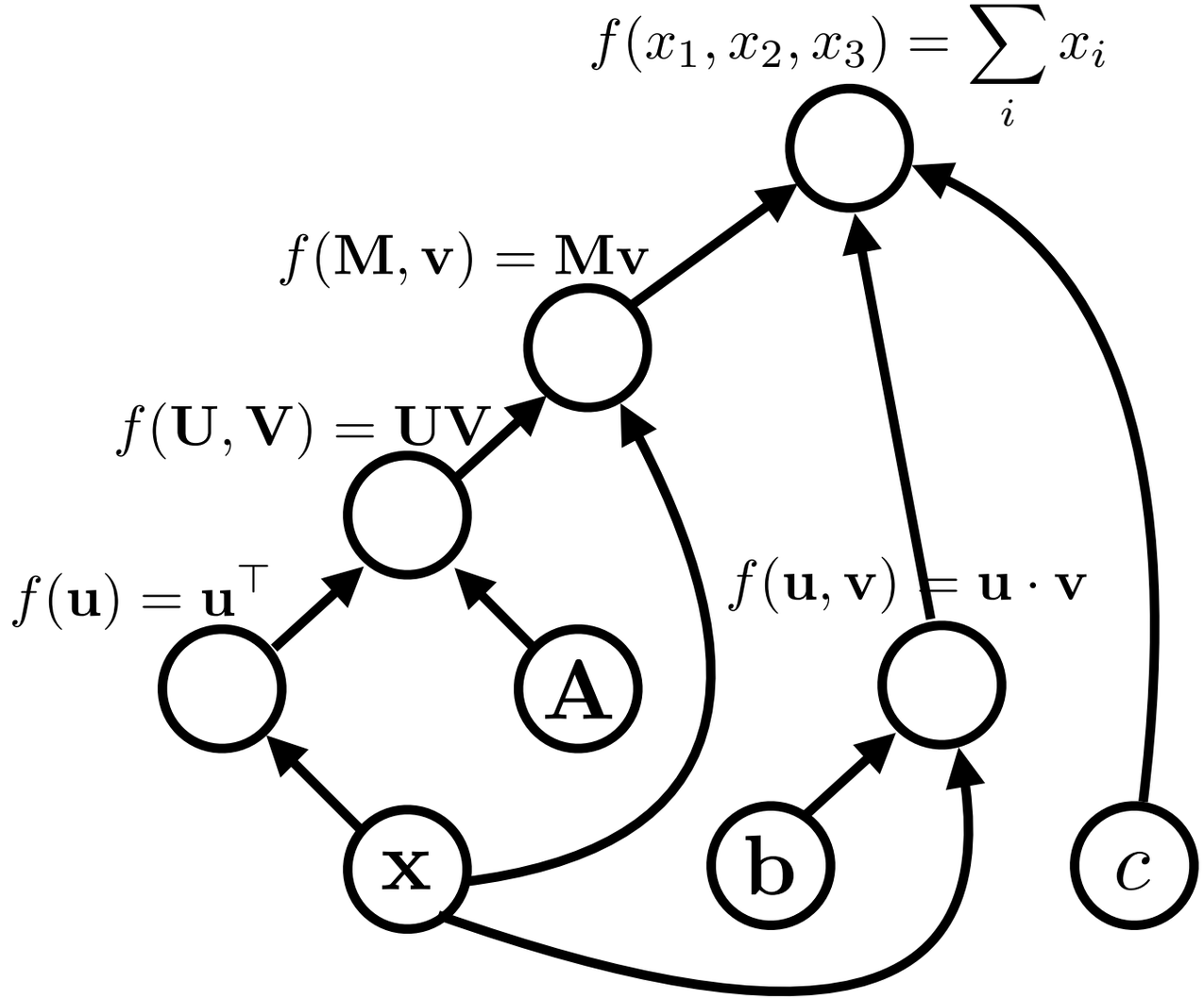
グラフ：



数式：

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$

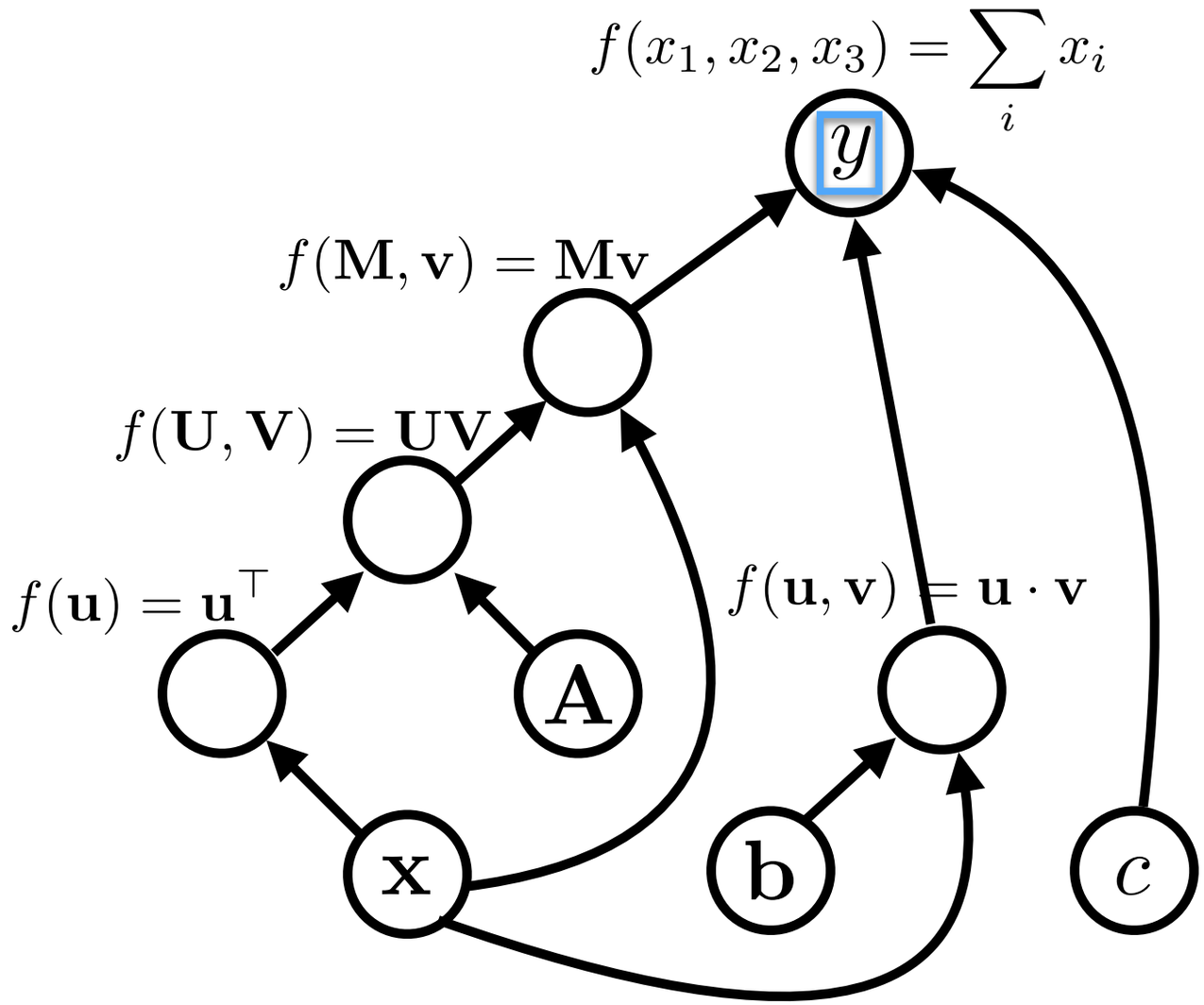
グラフ：



数式：

$$\boxed{y =} \mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$$

グラフ：



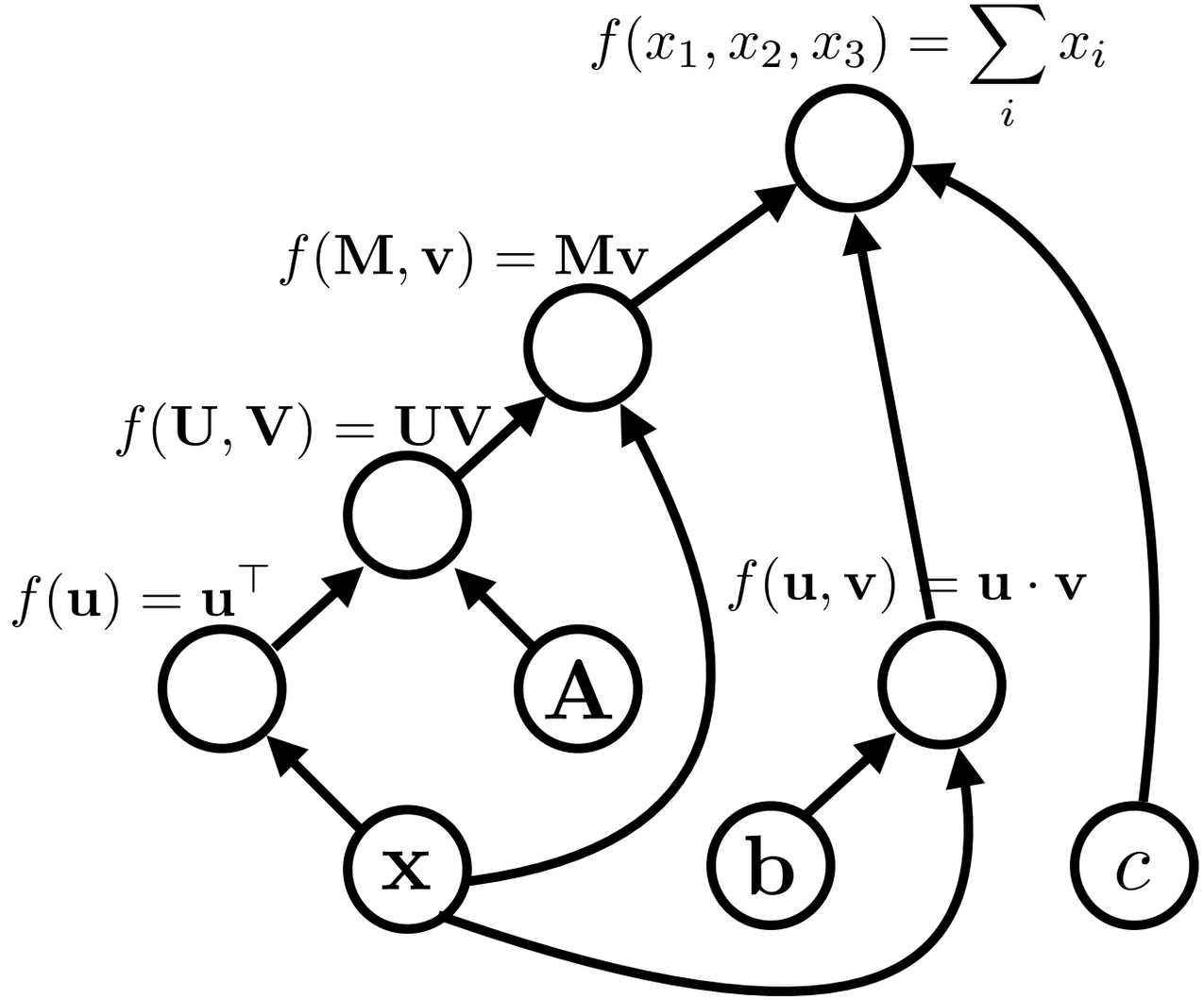
変数はただの頂点へのポインター

アルゴリズム

- **計算グラフの構築**
- **前向き計算**
    - 頂点をトポロジカル順に訪問し、値を計算していく
    - テスト時：「入力が与えられて、予測をする」
      学習時：「入力と正解が与えられて、損失関数を計算する」
- **後ろ向き計算**
    - 頂点を逆トポロジカル順に訪問し、勾配を計算していく
    - 「入力に対して少し変更を加えれば、出力がどう変わるか？」
- **（パラメータの更新）**

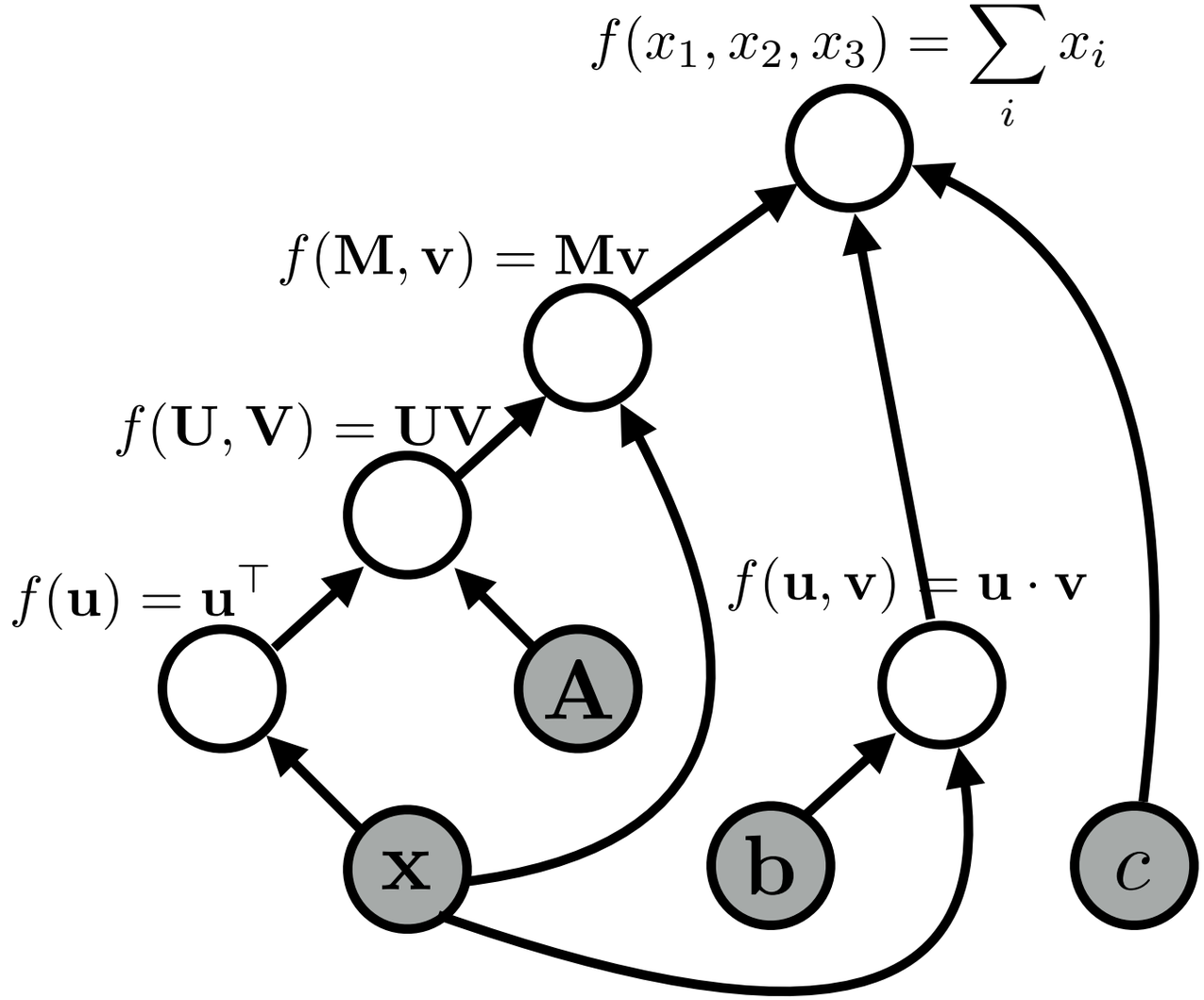
前向き計算

グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
A
x
b
c

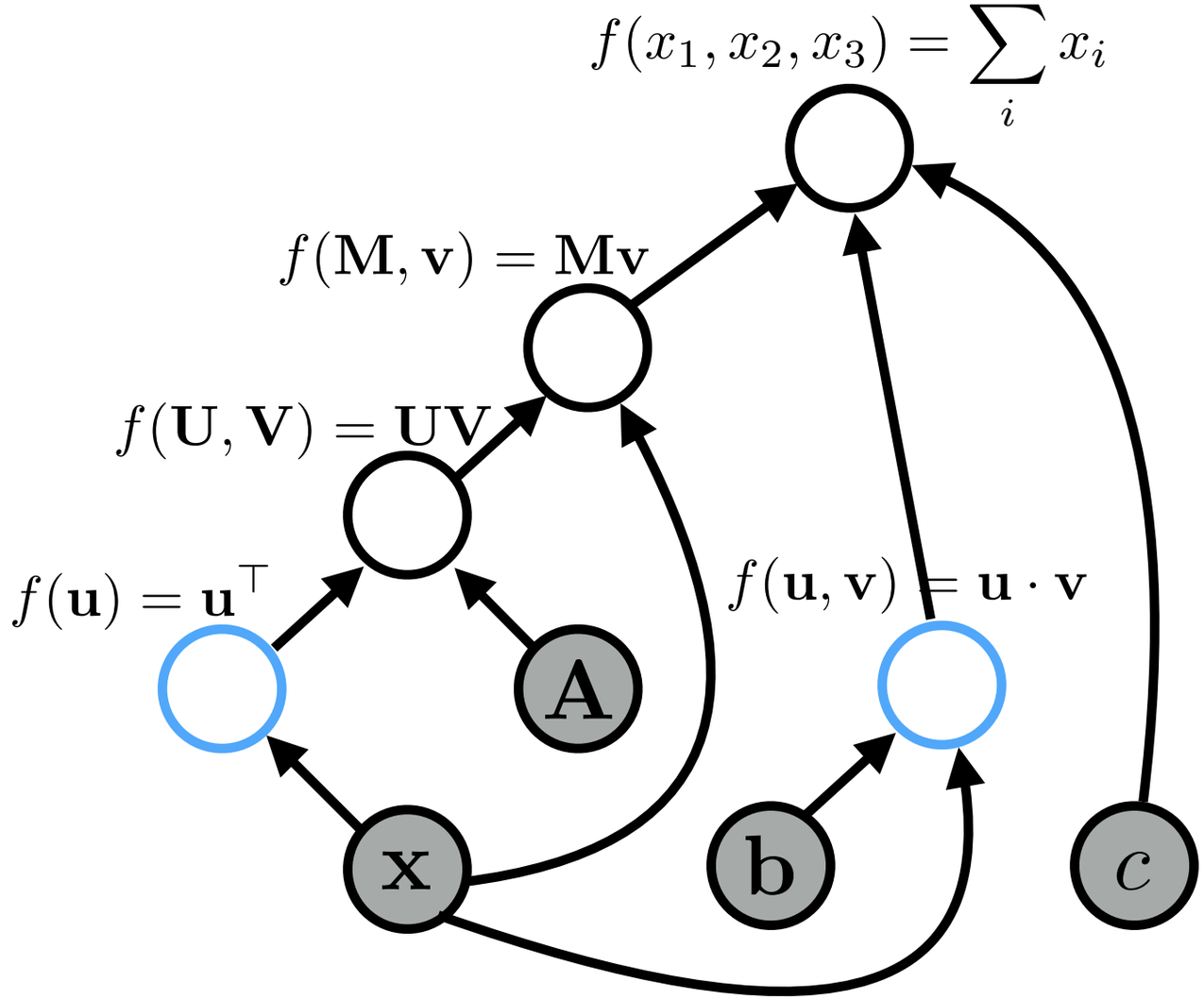
前向き計算

グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
A
x
b
c

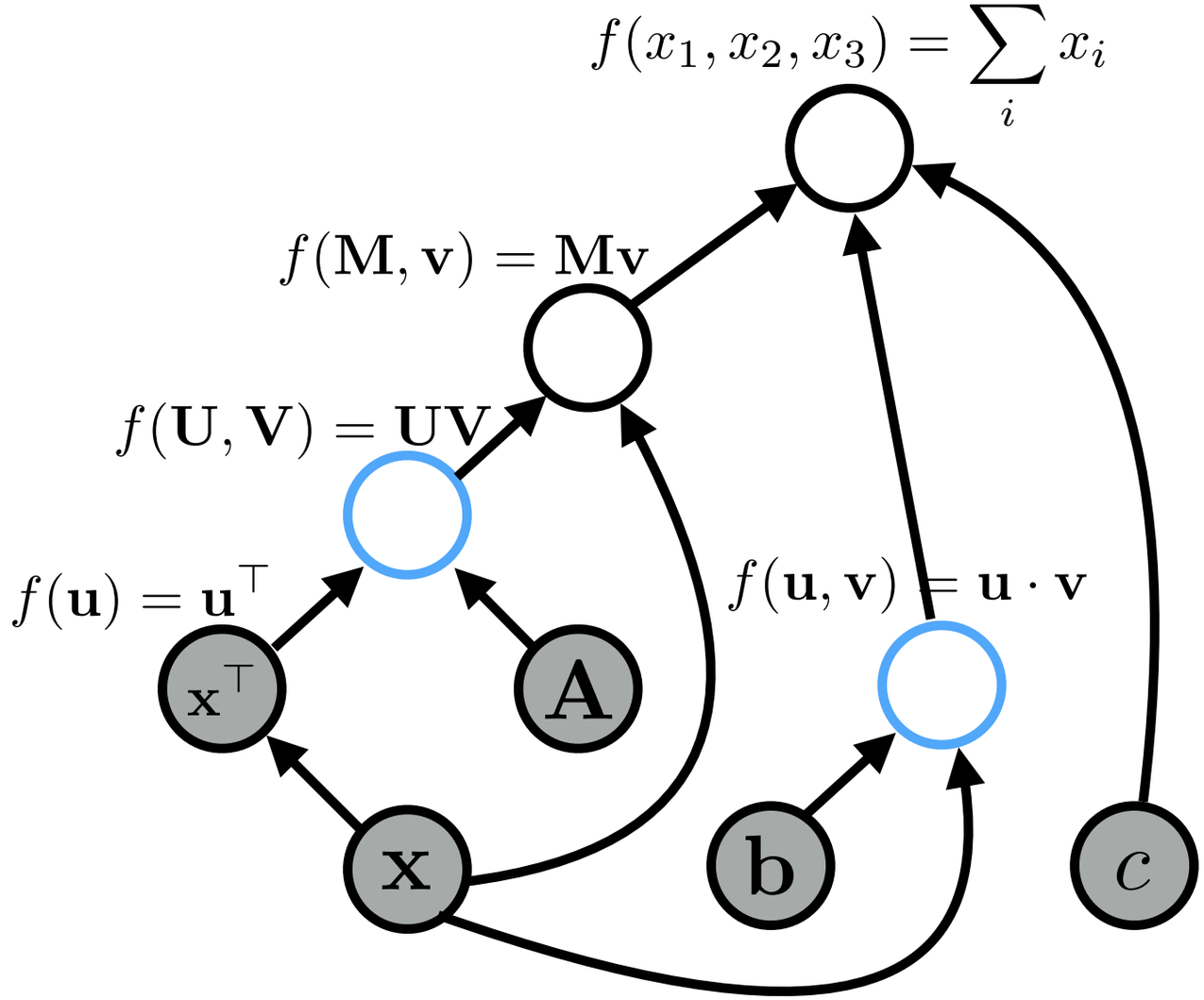
前向き計算

グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
A
x
b
c

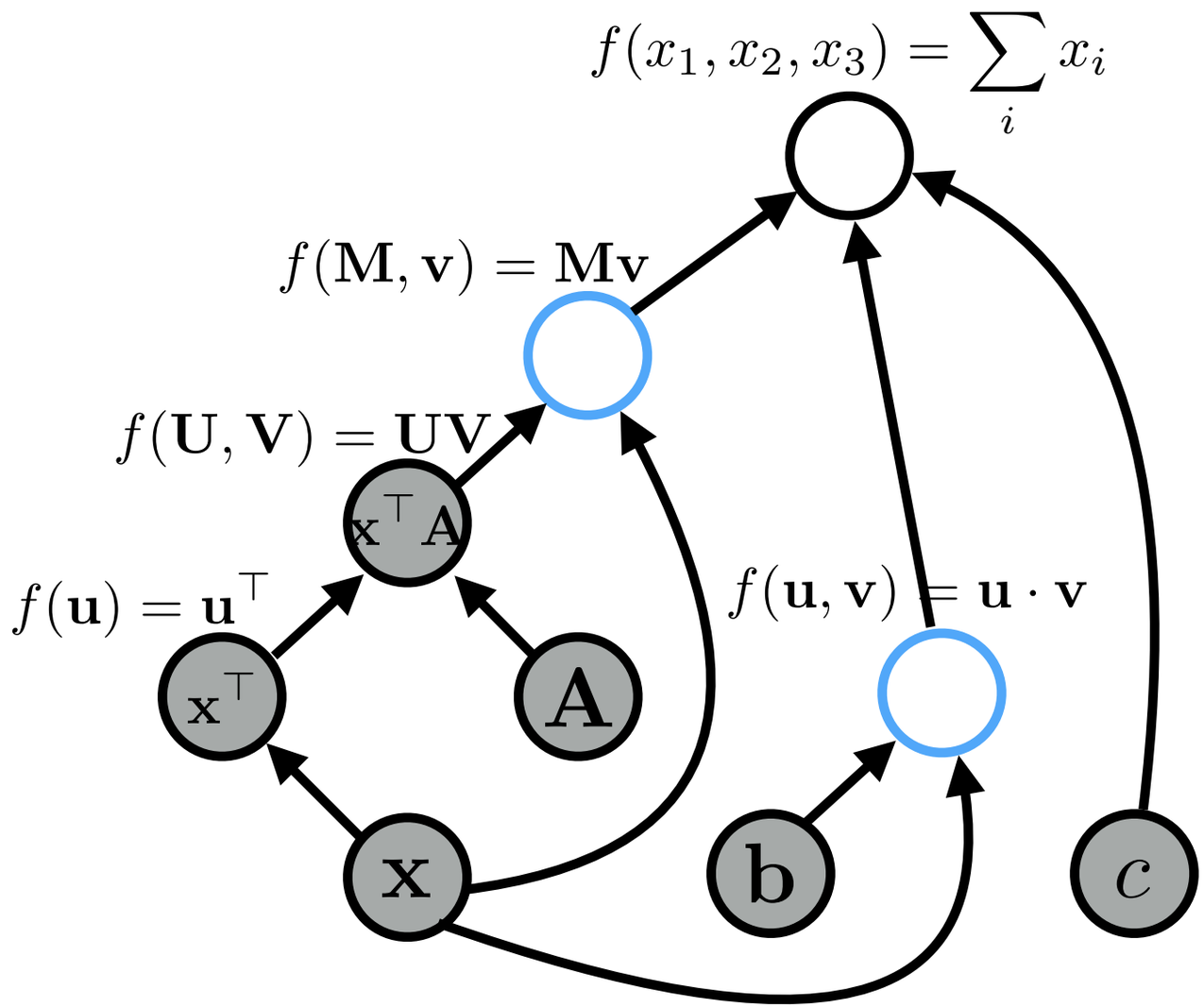
前向き計算

グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
$\mathbf{x}^\top$
$\mathbf{A}$
$\mathbf{x}$
$\mathbf{b}$
c

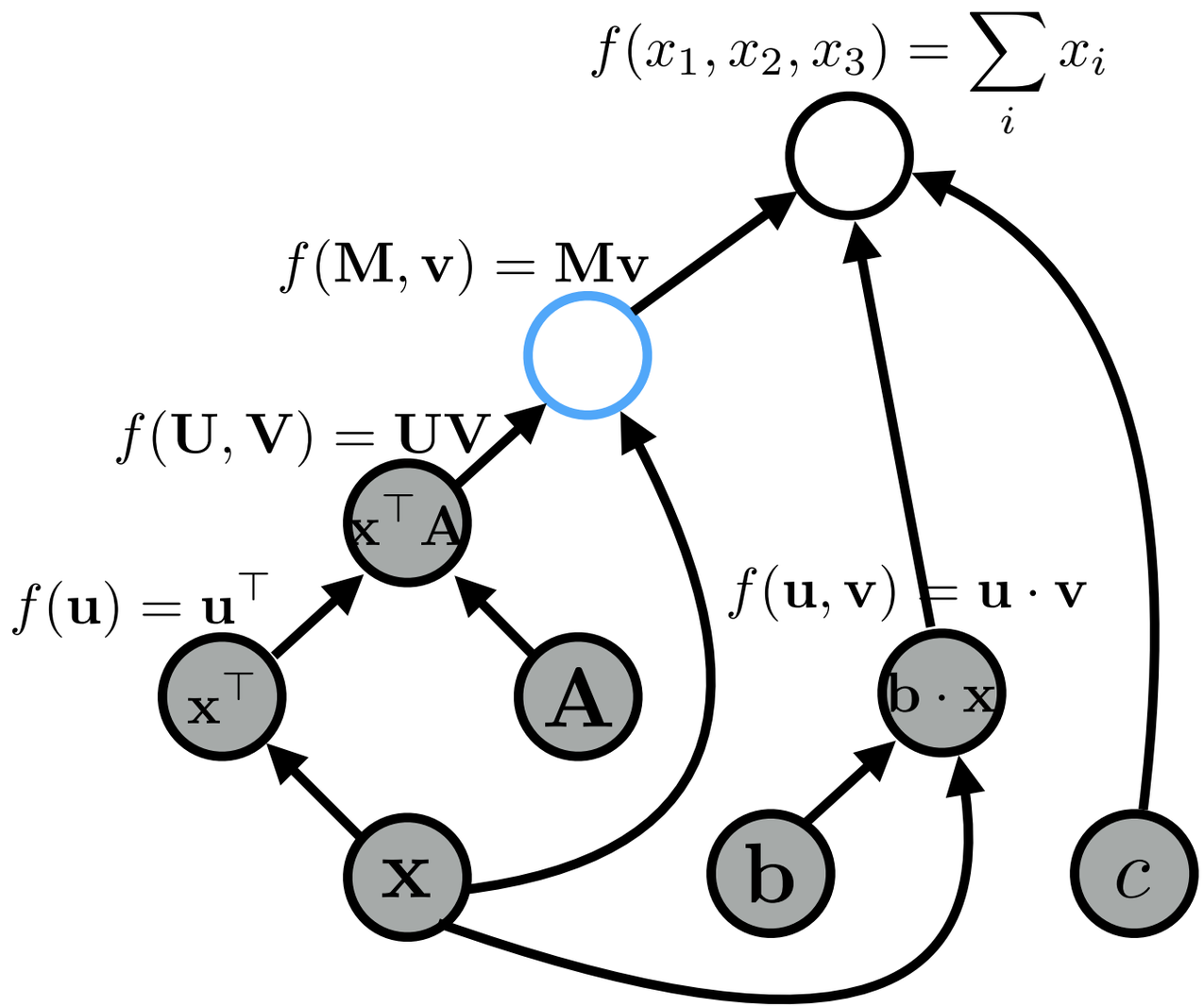
前向き計算

グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$
$\mathbf{x}^\top \mathbf{A}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
$\mathbf{x}^\top$
$\mathbf{A}$
$\mathbf{x}$
$\mathbf{b}$
c

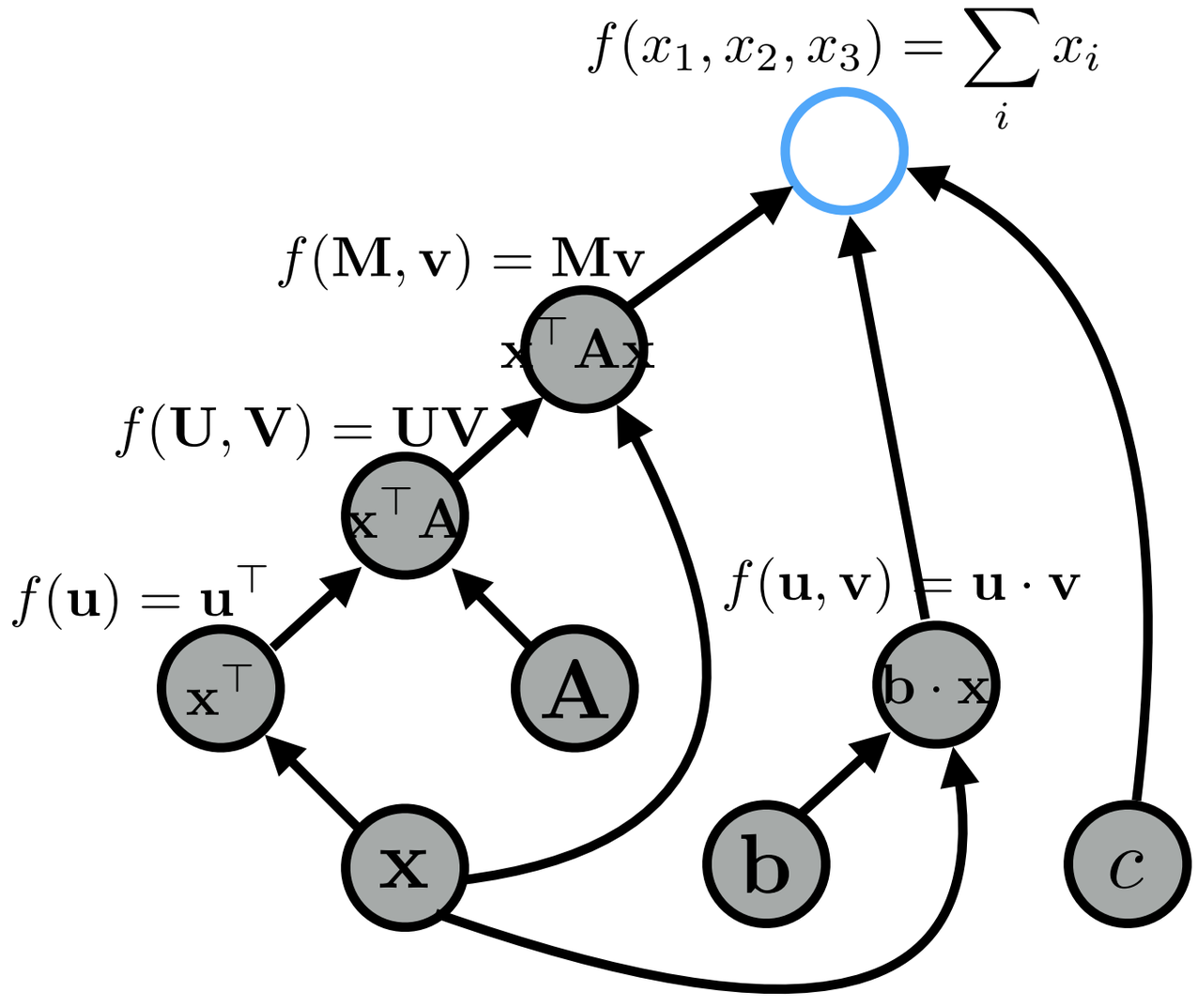
前向き計算

グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$
$\mathbf{x}^\top \mathbf{A}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
$\mathbf{x}^\top$
$\mathbf{A}$
$\mathbf{b} \cdot \mathbf{x}$
$\mathbf{x}$
$\mathbf{b}$
c

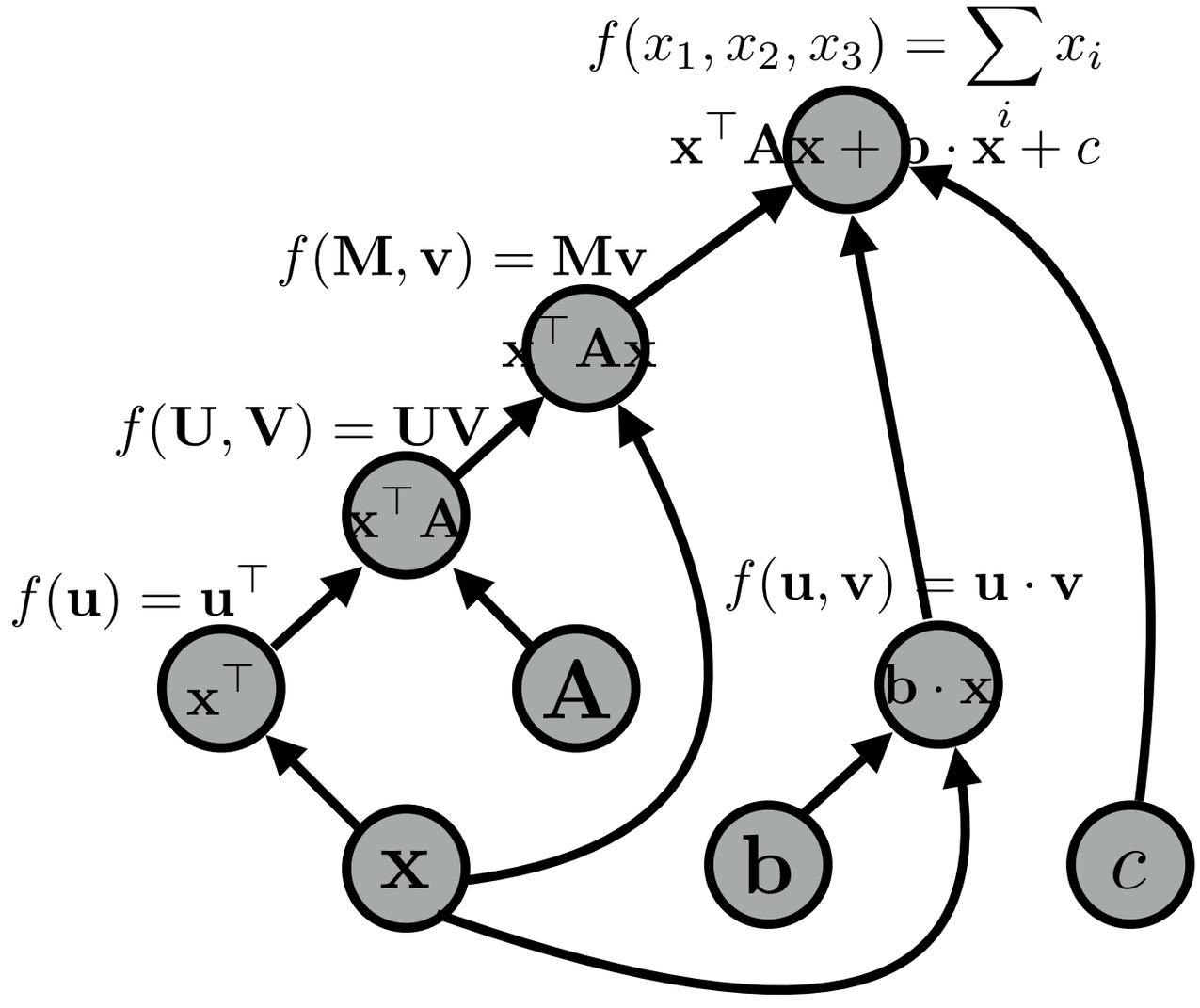
前向き計算

グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{Mv}$
$\mathbf{x}^\top \mathbf{Ax}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{UV}$
$\mathbf{x}^\top \mathbf{A}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
$\mathbf{x}^\top$
$\mathbf{A}$
$\mathbf{b} \cdot \mathbf{x}$
$\mathbf{x}$
$\mathbf{b}$
c

前向き計算

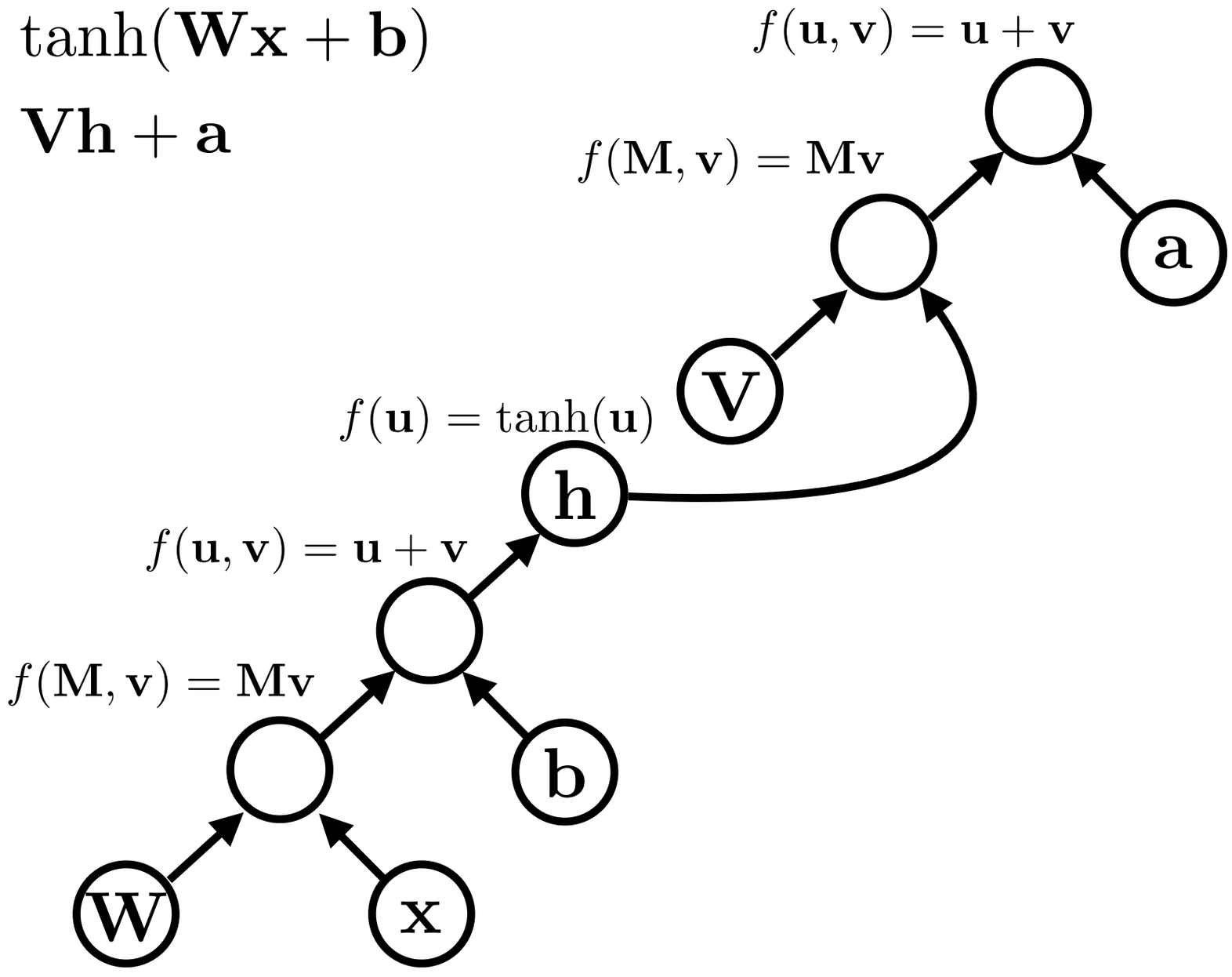
グラフ：


$f(x_1, x_2, x_3) = \sum_i x_i$
$\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c$
$f(\mathbf{M}, \mathbf{v}) = \mathbf{M}\mathbf{v}$
$\mathbf{x}^\top \mathbf{A}\mathbf{x}$
$f(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}$
$\mathbf{x}^\top \mathbf{A}$
$f(\mathbf{u}) = \mathbf{u}^\top$
$f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$
$\mathbf{x}^\top$
$\mathbf{A}$
$\mathbf{b} \cdot \mathbf{x}$
$\mathbf{x}$
$\mathbf{b}$
c

多層パーセプトロン

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$



グラフの構築法

計算したい関数をどう定義するか

手法1: 静的グラフ

(Tensorflow, Theano)

- **グラフの定義**

- 各データに対して：
  - **前向き計算**
  - **後ろ向き計算**
  - **パラメータ更新**

静的グラフの長所短所

- **長所**
  - グラフ定義時に最適化が可能
  - データのGPUへの送信、並列化の最適化などが可能
- **短所**
  - 複雑な形状（可変長、木やグラフを利用）や複雑な制御（計算結果によって動作が変わる）を必要とするネットには適応が困難
  - 上記の例に対応するため、APIが膨大となり取得が困難

手法２：動的グラフ＋即時評価 (PyTorch, Chainer)

- 各データに対して：
  - **グラフの定義**と同時に**前向き計算**
  - **後ろ向き計算**
  - **パラメータ更新**

動的グラフ＋即時評価の長所短所

- **長所**
  - 複雑な形状や制御が比較的楽
  - APIは普通のPython/C++などに近い
- **短所**
  - グラフ定義時に最適化が不可
  - データのGPUへの送信、並列化の最適化などが困難

手法３：動的グラフ＋遅延評価 (DyNet)

- 各データに対して：
  - **グラフの定義**
  - **前向き計算**
  - **後ろ向き計算**
  - **パラメータ更新**

動的グラフ＋遅延評価の長所短所

- **長所**
  - 複雑な形状や制御が比較的楽
  - APIは普通のPython/C++などに近い
  - グラフ定義時に最適化が可能
- **短所**
  - データのGPUへの送信、並列化の最適化などが困難

DyNetでの実装

作ってみよう！

重要な概念

- ComputationGraph
- Expression
- Parameter
- Model (Parameterの集合)
- Trainer

ComputationGraph, Expression

```
import dynet as dy

dy.renew_cg() # create a new computation graph

v1 = dy.inputVector([1,2,3,4])
v2 = dy.inputVector([5,6,7,8])
# v1 and v2 are expressions

v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1

v6 = dy.concatenate([v1,v2,v3,v5])

print v6
print v6.npvalue()
```

ComputationGraph, Expression

```
import dynet as dy

dy.renew_cg() # create a new computation graph

v1 = dy.inputVector([1,2,3,4])
v2 = dy.inputVector([5,6,7,8])
# v1 and v2 are expressions

v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1

v6 = dy.concatenate([v1,v2,v3,v5])

print v6 expression 5/1
print v6.npvalue()
```

ComputationGraph, Expression

```
import dynet as dy

dy.renew_cg() # create a new computation graph

v1 = dy.inputVector([1,2,3,4])
v2 = dy.inputVector([5,6,7,8])
# v1 and v2 are expressions

v3 = v1 + v2
v4 = v3 * 2
v5 = v1 + 1

v6 = dy.concatenate([v1,v2,v3,v5])

print v6
print v6.npvalue()
```

```
array([  1.,   2.,   3.,   4.,   2.,   4.,   6.,   8.,   4.,   8.,  12.,  16.])
```

ComputationGraph, Expression

- 入力やパラメータのExpressionを作成
- 計算で組み合わせる
- Expression**定義時に計算が行われない**
- 下記の関数を呼ぶとき計算が行われ、値が返される

```
.value()
.npvalue()
.scalar_value()
.vec_value()
.forward()
```

Model, Parameters

- **Parameters**は学習したいパラメータの{スカラー、ベクトル、テンソル}を表す

- **Model** はParametersの集合を表す

- ComputationGraphは1回限り
  Parametersはプログラム実行時ずっと

Model, Parameters

```
model = dy.Model()

pW = model.add_parameters((20,4))
pb = model.add_parameters(20)
```

```
dy.renew_cg()
x = dy.inputVector([1,2,3,4])
W = dy.parameter(pW) # convert params to expression
b = dy.parameter(pb) # and add to the graph

y = W * x + b
```

Parameterの初期化

```
model = dy.Model()

pW = model.add_parameters((4,4))

pW2 = model.add_parameters((4,4), init=dy.GlorotInitializer())
```

```
pW3 = model.add_parameters((4,4), init=dy.NormalInitializer(0,1))

pW4 = model.parameters_from_numpu(np.eye(4))
```

Trainerと後ろ向き計算

- **Trainer**はパラメータを最適化（Modelに対して定義）
- `expr.backward()`で勾配の後ろ向き計算を行う
- `trainer.update()`で勾配にもとづいてパラメータを更新

Trainerと後ろ向き計算

```
model = dy.Model()

trainer = dy.SimpleSGDTrainer(model)

p_v = model.add_parameters(10)

for i in xrange(10):
    dy.renew_cg()

    v = dy.parameter(p_v)
    v2 = dy.dot_product(v,v)
    v2.forward()

    v2.backward()  # compute gradients

    trainer.update()
```

Trainerと後ろ向き計算

```
model = dy.Model()

trainer =

p_v = mode

for i in x
    dy.rer

    v = dy
    v2 = d
    v2.for

    v2.backward()  # compute gradients

    trainer.update()
```

```
dy.SimpleSGDTrainer(model,...)
dy.MomentumSGDTrainer(model,...)
dy.AdagradTrainer(model,...)
dy.AdadeltaTrainer(model,...)
dy.AdamTrainer(model,...)
```

DyNetでの学習の全体像

- **Parameter/Modelの定義**

- 各データに対して：
  - **グラフの定義**
  - **前向き計算**
  - **後ろ向き計算**
  - **パラメータ更新**

例:XOR用の多層パーセプトロン

- データ

$$\mathrm{xor}(0,0) = 0$$
$$\mathrm{xor}(1,0) = 1$$
$$\mathrm{xor}(0,1) = 1$$
$$\mathrm{xor}(1,1) = 0$$

$\mathbf{x}$ y

- モデル

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

- 損失関数

$$\ell = \begin{cases} -\log \hat{y} & y = 1 \\ -\log(1 - \hat{y}) & y = 0 \end{cases}$$

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

```
import dynet as dy
import random

data =[ ([0,1],0),
        ([1,0],0),
        ([0,0],1),
        ([1,1],1) ]


model = dy.Model()
pU = model.add_parameters((4,2))
pb = model.add_parameters(4)
pv = model.add_parameters(4)

trainer = dy.SimpleSGDTrainer(model)
closs = 0.0

for ITER in xrange(1000):
    random.shuffle(data)
    for x,y in data:
        ....
```

```
for ITER in xrange(1000):
  for x,y in data:
```

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      x = dy.inputVector(x)
      # predict
      yhat = dy.logistic(dy.dot_product(v,dy.tanh(U*x+b)))
      # loss
      if y == 0:
          loss = -dy.log(1 - yhat)
      elif y == 1:
          loss = -dy.log(yhat)

      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      x = dy.inputVector(x)
      # predict
      yhat = dy.logistic(dy.dot_product(v,dy.tanh(U*x+b)))
      # loss
      if y == 0:
          loss = -dy.log(1 - yhat)
      elif y == 1:
          loss = -dy.log(yhat)

      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      x = dy.inputVector(x)
      # predict
      yhat = dy.logistic(dy.dot_product(v,dy.tanh(U*x+b)))
```

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

```
      # loss
      if y == 0:
          loss = -dy.log(1 - yhat)
      elif y == 1:
          loss = -dy.log(yhat)
```

$$\ell = \begin{cases} -\log \hat{y} & y = 1 \\ -\log(1 - \hat{y}) & y = 0 \end{cases}$$

```
      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      x = dy.inputVector(x)
      # predict
      yhat = dy.logistic(dy.dot_product(v,dy.tanh(U*x+b)))
      # loss
      if y == 0:
          loss = -dy.log(1 - yhat)
      elif y == 1:
          loss = -dy.log(yhat)
```

$$\ell = \begin{cases} -\log \hat{y} & y = 1 \\ -\log(1 - \hat{y}) & y = 0 \end{cases}$$

```
      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      x = dy.inputVector(x)
      # predict
      yhat = dy.logistic(dy.dot_product(v,dy.tanh(U*x+b)))
      # loss
      if y == 0:
          loss = -dy.log(1 - yhat)
      elif y == 1:
          loss = -dy.log(yhat)

      closs += loss.scalar_value() # forward
```

$$\ell = \begin{cases} -\log \hat{y} & y = 1 \\ -\log(1 - \hat{y}) & y = 0 \end{cases}$$

```
      if ITER > 0 and ITER % 100 == 0:
              print "Iter:",ITER,"loss:", closs/400
              closs = 0
```

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      x = dy.inputVector(x)
      # predict
      yhat = dy.logistic(dy.dot_product(v,dy.tanh(U*x+b)))
      # loss
      if y == 0:
          loss = -dy.log(1 - yhat)
      elif y == 1:
          loss = -dy.log(yhat)

      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

コードの整理をしよう！

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      x = dy.inputVector(x)
      # predict
      yhat = dy.logistic(dy.dot_product(v,dy.tanh(U*x+b)))
      # loss
      if y == 0:
          loss = -dy.log(1 - yhat)
      elif y == 1:
          loss = -dy.log(yhat)

      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

コードの整理をしよう！

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()

      x = dy.inputVector(x)
      # predict
      yhat = predict(x)
      # loss
      loss = compute_loss(yhat, y)
      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()

      x = dy.inputVector(x)
      # predict
      yhat = predict(x)
      # loss
      loss = compute_loss(yhat, y)
      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

$$\hat{y} = \sigma(\mathbf{v} \cdot \tanh(\mathbf{U}\mathbf{x} + \mathbf{b}))$$

```
def predict(expr):
      U = dy.parameter(pU)
      b = dy.parameter(pb)
      v = dy.parameter(pv)
      y = dy.logistic(dy.dot_product(v,dy.tanh(U*expr+b)))
      return y
```

```
for ITER in xrange(1000):
  for x,y in data:
      # create graph for computing loss
      dy.renew_cg()

      x = dy.inputVector(x)
      # predict
      yhat = predict(x)
      # loss
      loss = compute_loss(yhat, y)
      closs += loss.scalar_value() # forward
      loss.backward()
      trainer.update()
```

```
def compute_loss(expr, y):
      if y == 0:
            return -dy.log(1 - expr)
      elif y == 1:
            return -dy.log(expr)
```

$$\ell = \begin{cases} -\log \hat{y} & y = 1 \\ -\log(1 - \hat{y}) & y = 0 \end{cases}$$

注意点

- 各事例に対してグラフを作成
- グラフはExpressionの組み合わせで作成
- 関数はExpressionを入力とし、Expressionを返す

実例：手書き文字認識

深層学習のHello World

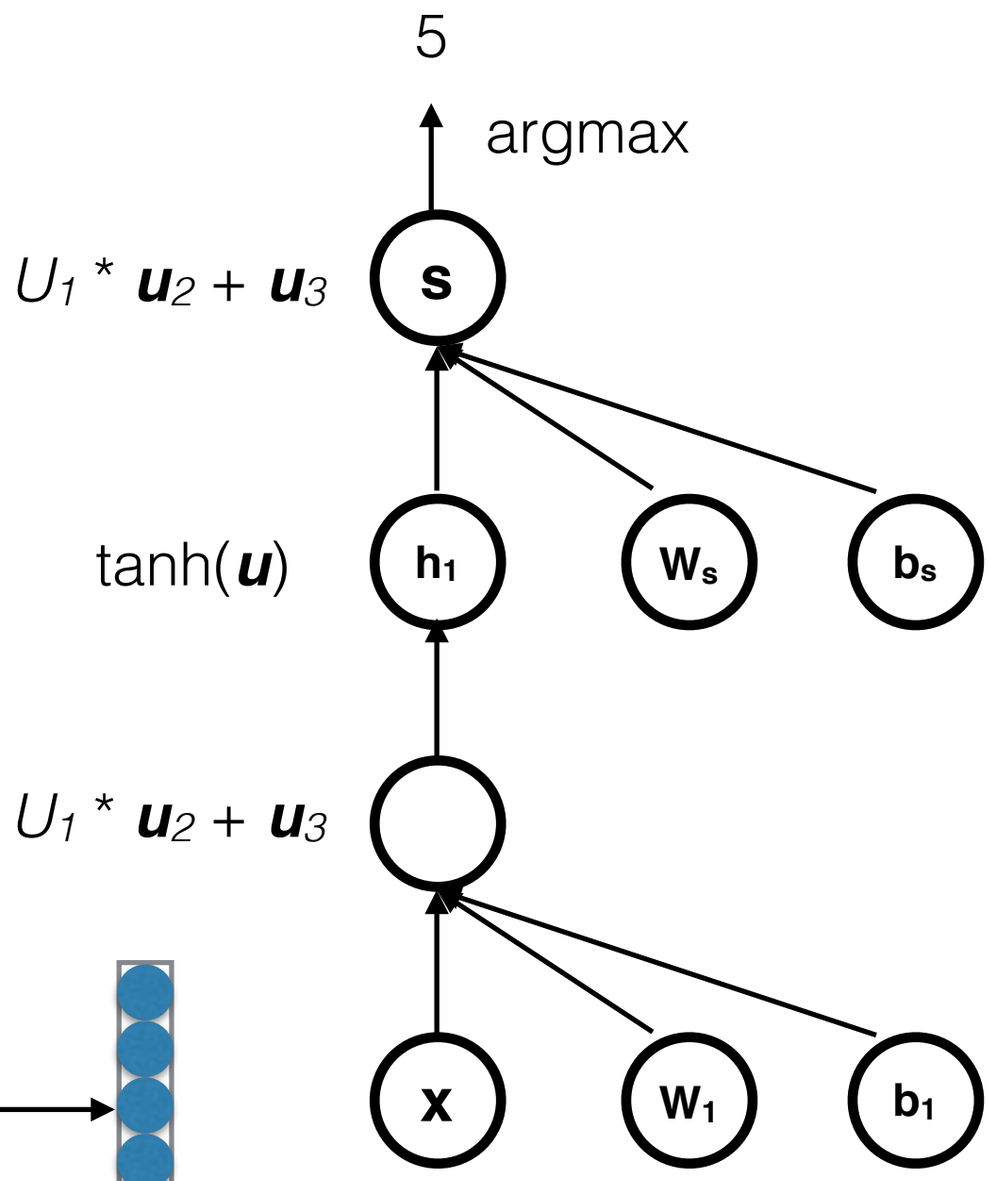
MNIST

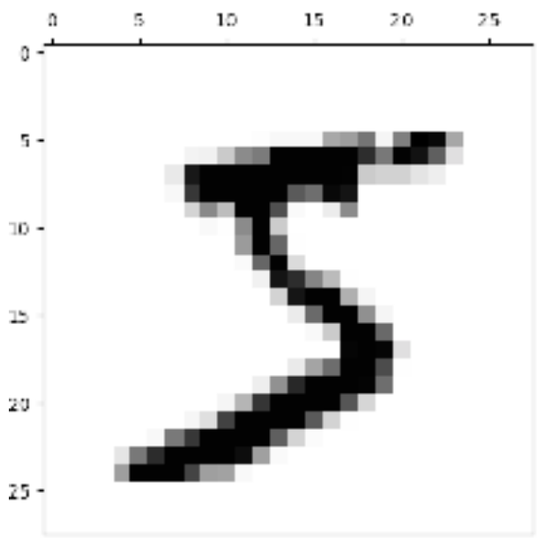
なぜ文字認識なのか？

- 入出力の**大きさが固定**：画素数 → ラベル数
  （テキスト・音声はそうでもない）

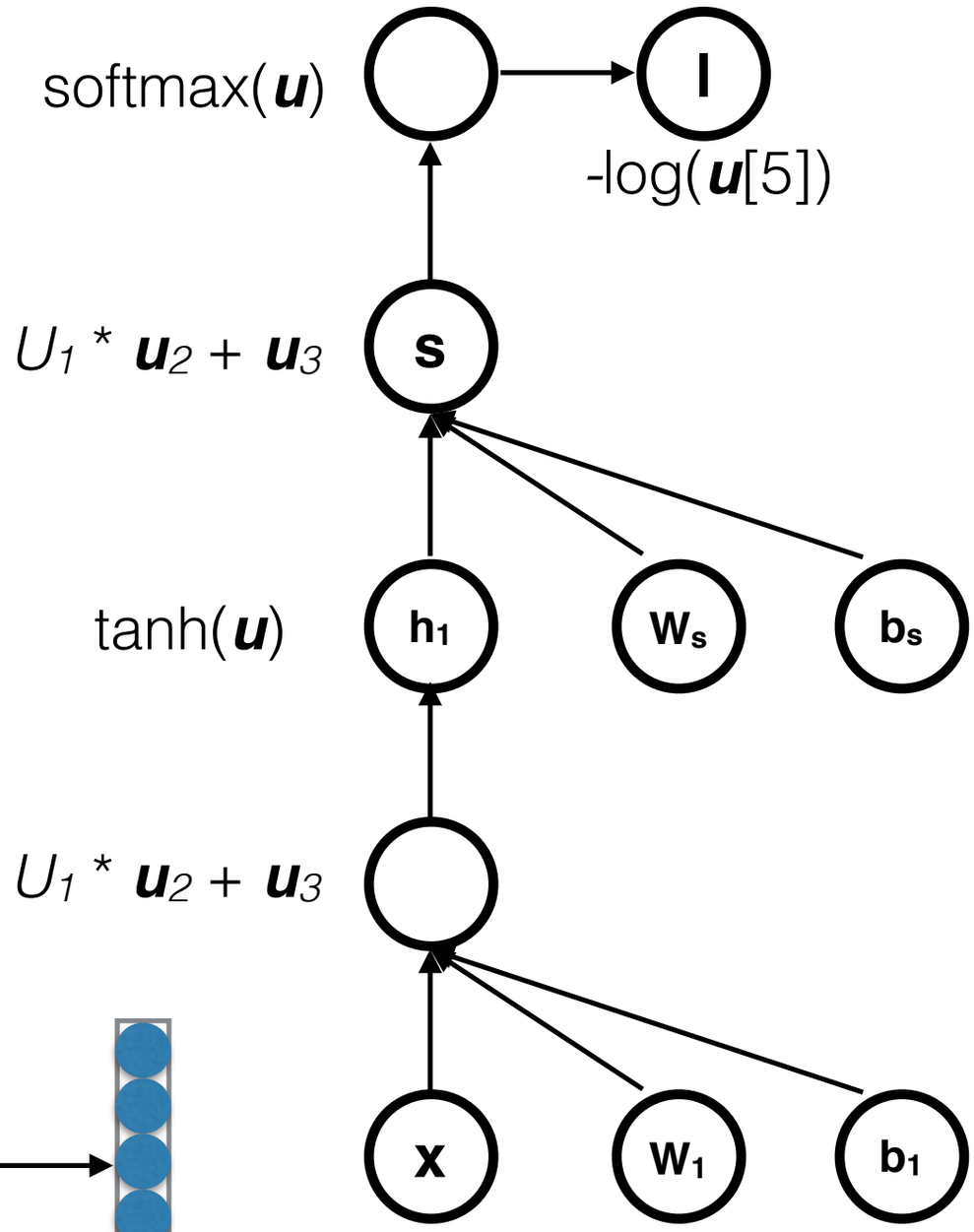
- 入力が**連続値**
  （テキストはそうでもない）

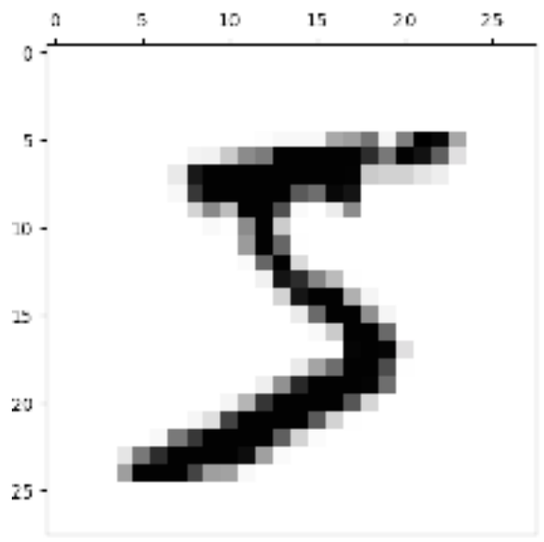
識別してみよう





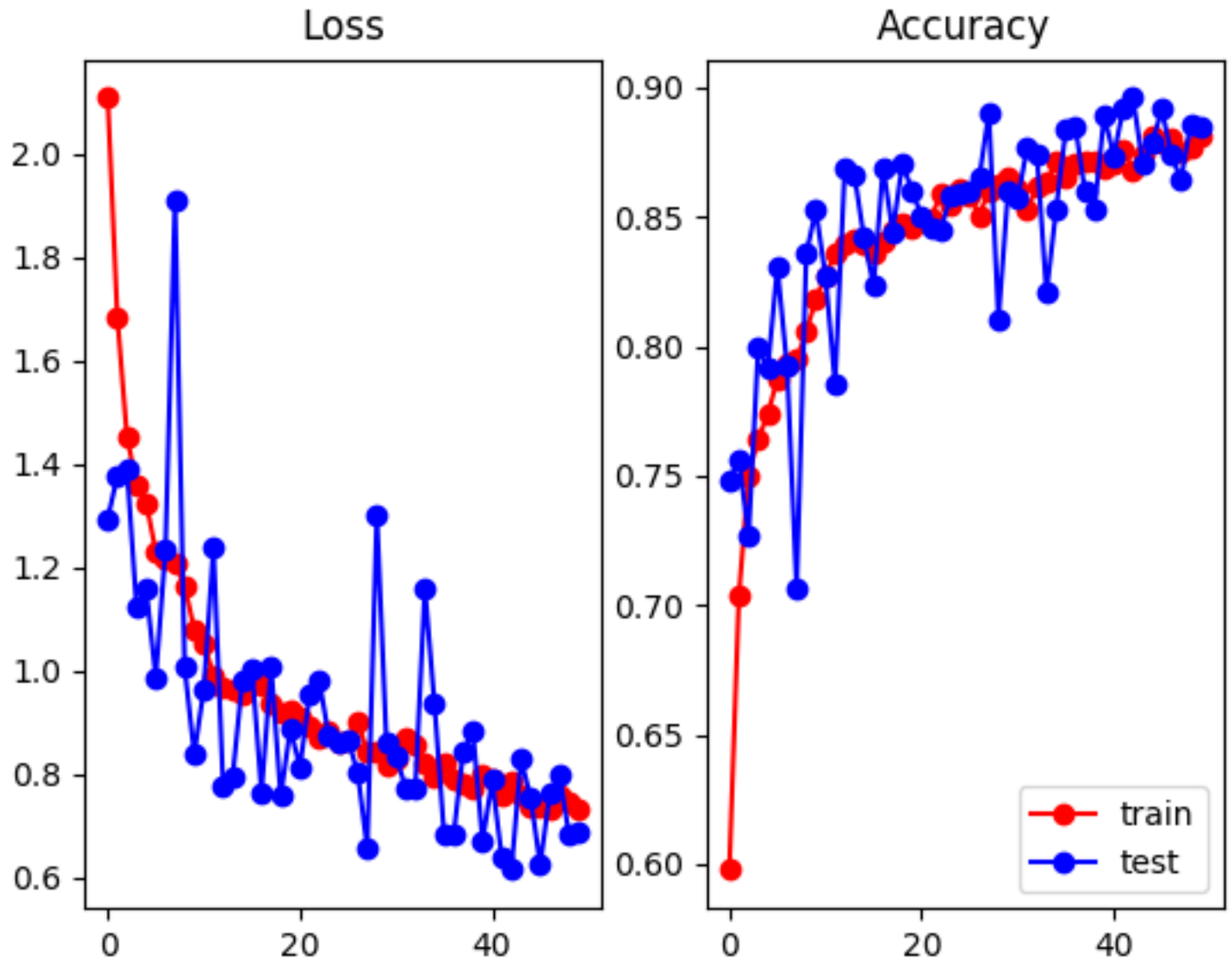
学習してみよう


softmax(**u**)
l
-log(**u**[5])
U_1 * $\mathbf{u}_2$ + $\mathbf{u}_3$
s
tanh(**u**)
h1
Ws
bs
U_1 * $\mathbf{u}_2$ + $\mathbf{u}_3$
x
W1
b1



MNISTの実装例

- 01-mnist-mlp.py



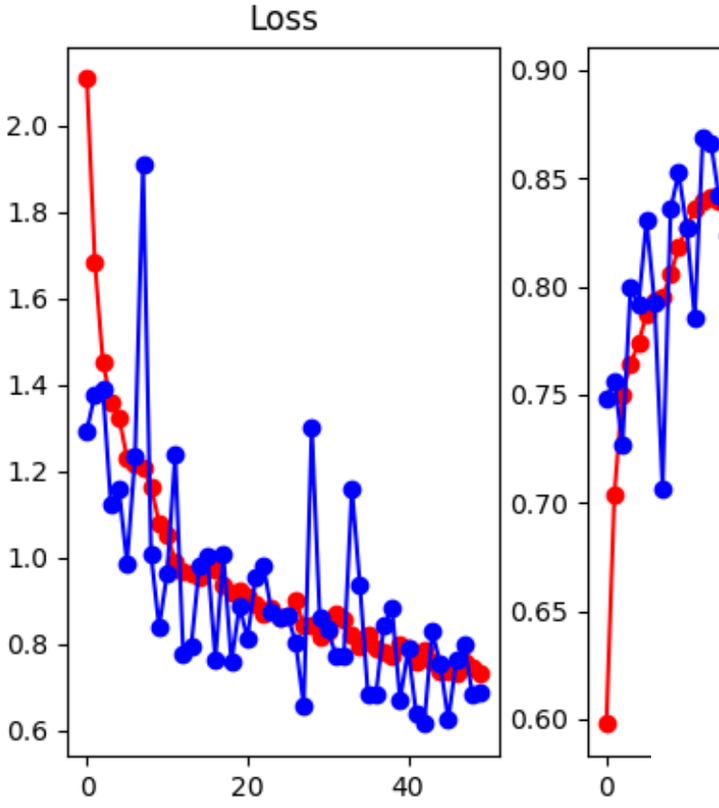
実装上の工夫

学習率

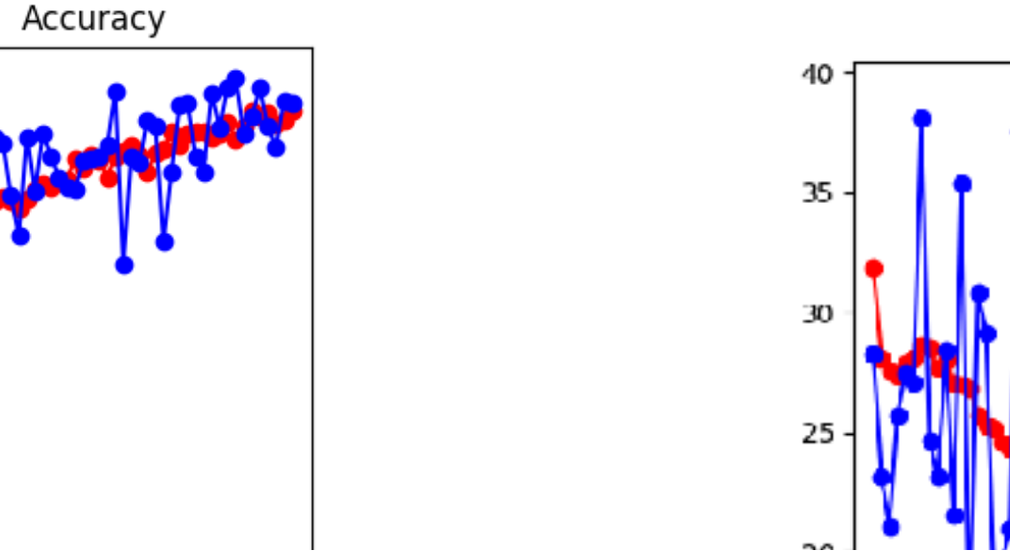
- 学習率が
  - 高い：すぐに収束し、安定しない
  - 低い：時間がかかり、局所解に陥りやすい

学習率の例

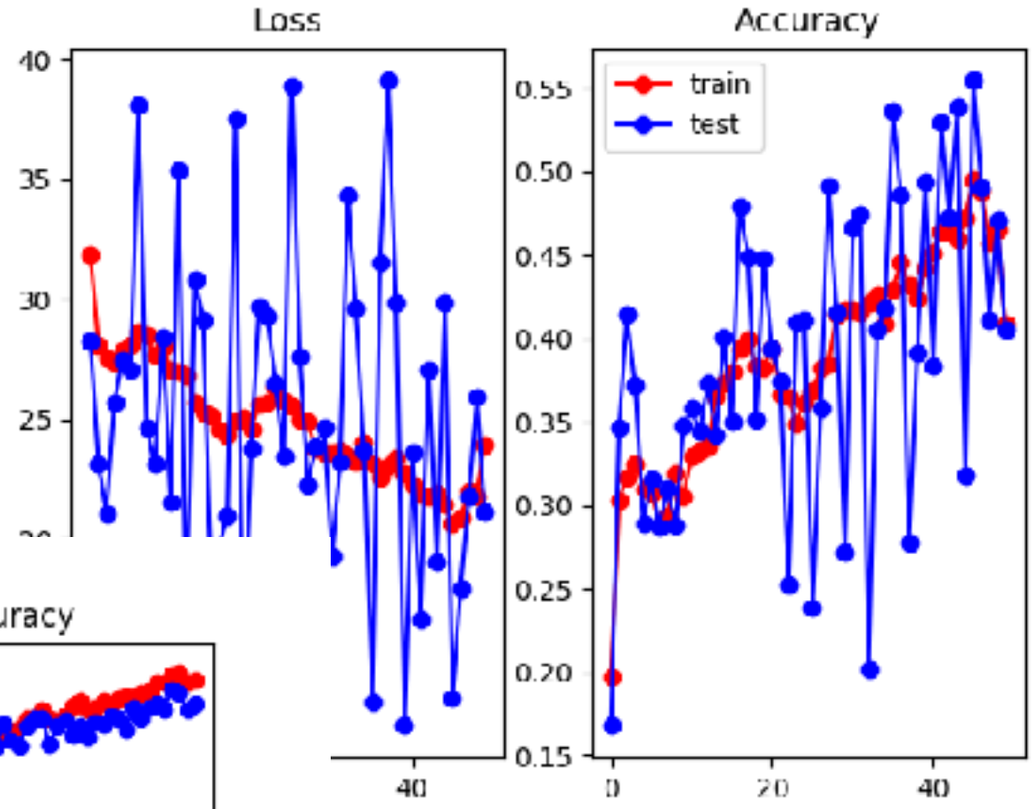
- 02-mnist-learningrate.py



lr=0.1



lr=0.01



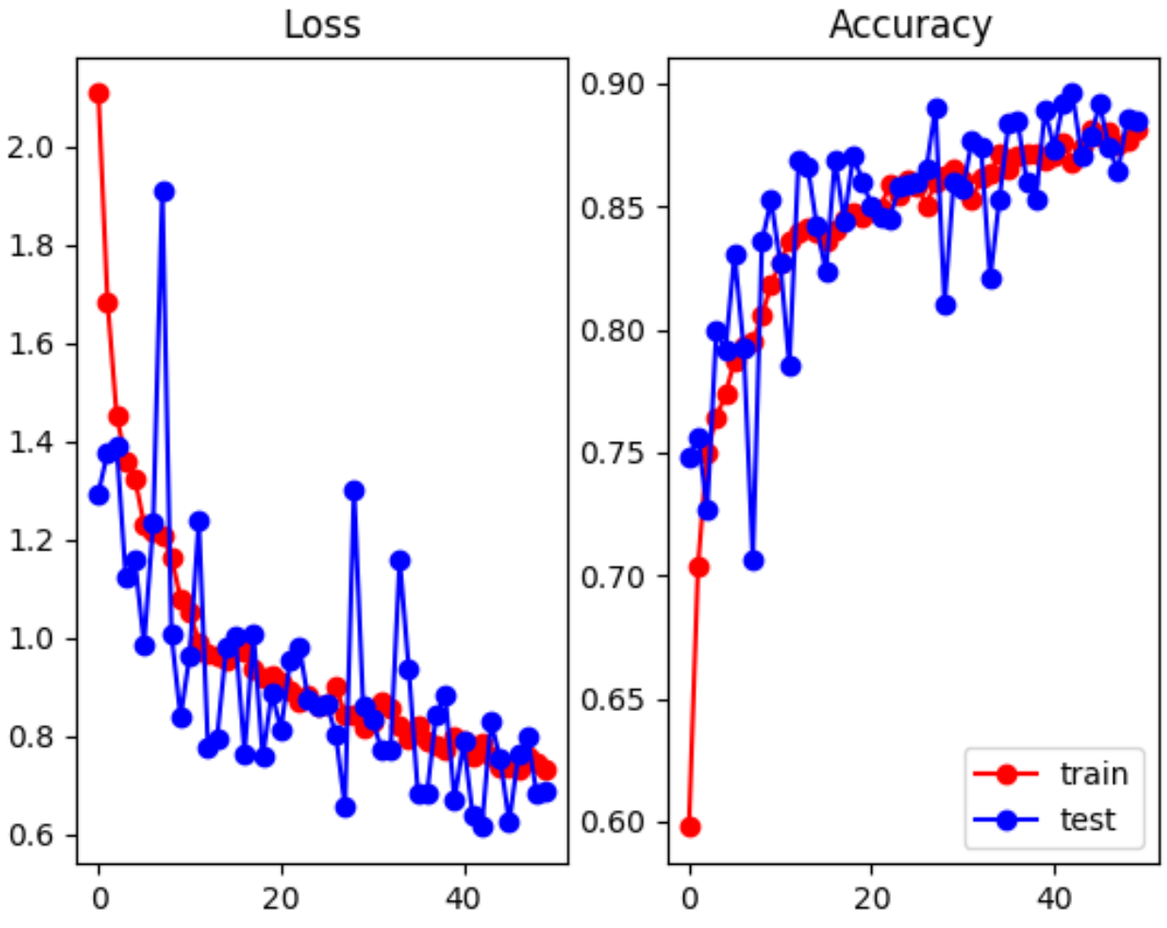
lr=1.0

最適化手法

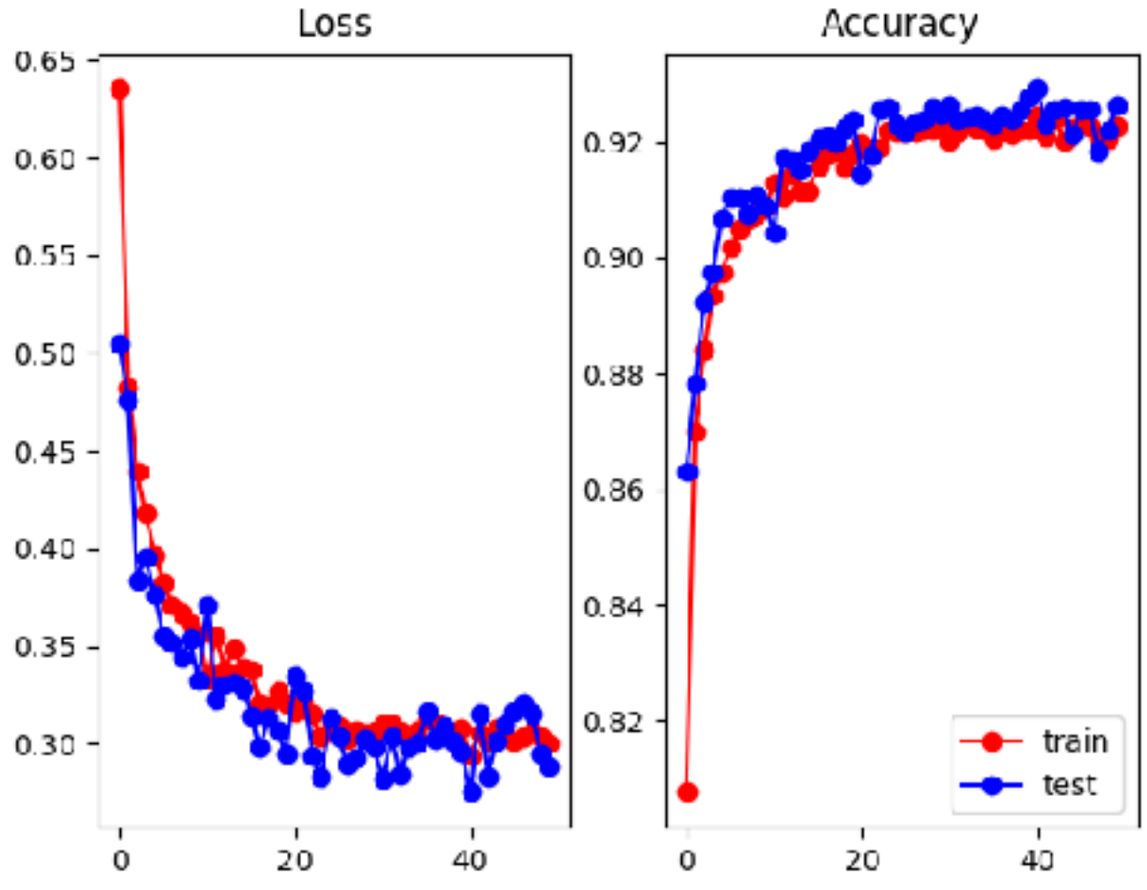
- **Simple SGD：**単純に勾配に従う。遅い
- **AdaGrad：**各パラメーターごとに学習率を調整。よく更新されるパラメータの更新幅を小さく
- **AdaDelta：**AdaGradで学習率が小さくなっていく問題を修正、移動平均を用いる
- **Adam：**移動平均と勾配の平均・分散を考慮。現在よく用いられる
- ただ、時間を気にしない場合はSimple SGDは過学習しにくく強い

最適化手法の例

- 03-mnist-adam.py



SGD (lr=0.1)



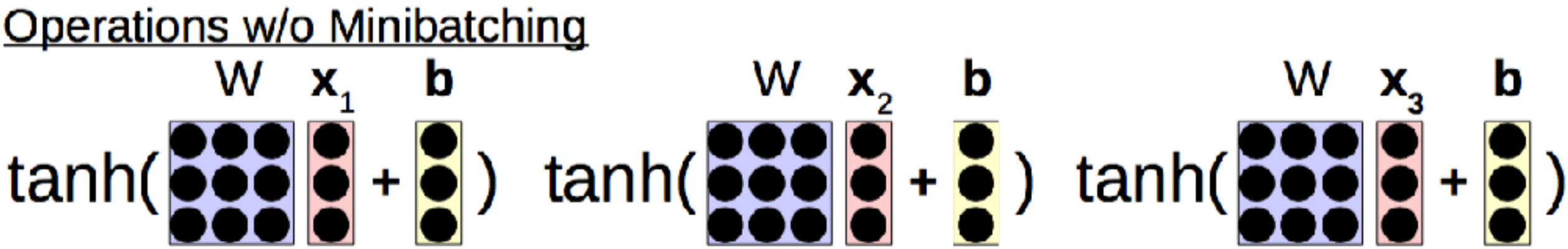
Adam (lr=0.001)

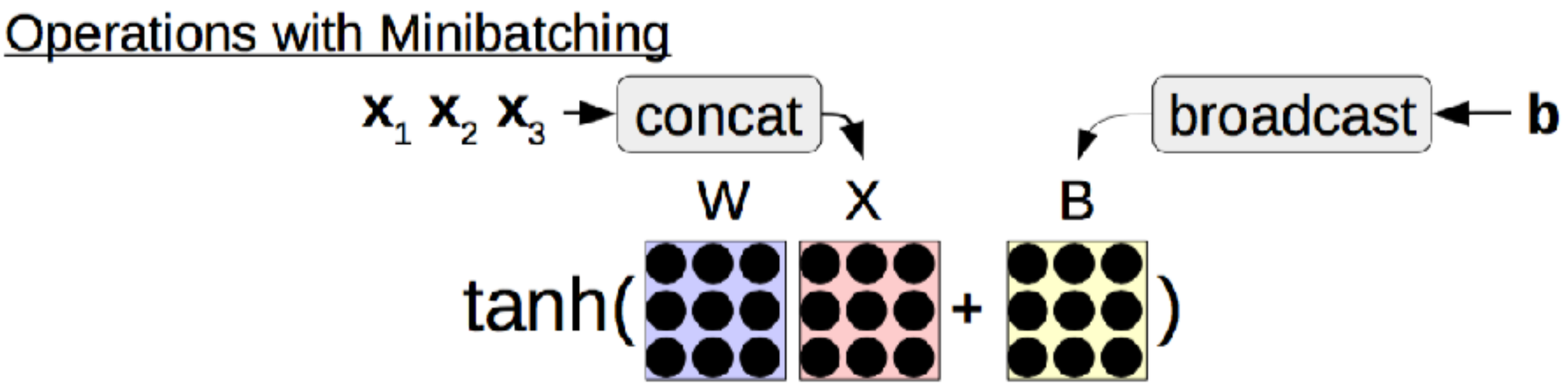
ミニバッチ化

- 最近のハードウェアではサイズ１の演算を１０回行うよりはサイズ10の演算を１回行う方が**ずっと早い**

- ミニバッチ化は複数の小さい演算を１つの大きな演算に組み合わせ

ミニバッチ化の例


Operations w/o Minibatching
W x1 b
tanh(+)
W x2 b
tanh(+)
W x3 b
tanh(+)

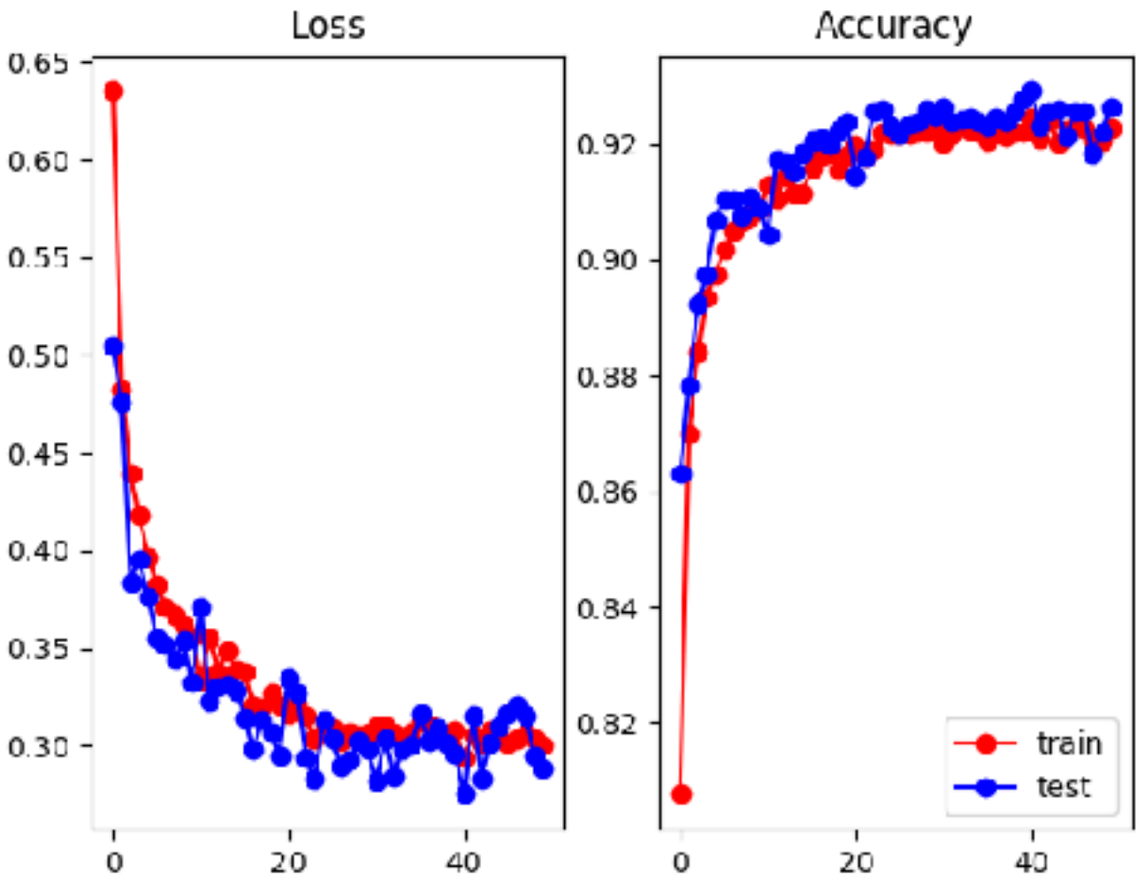

Operations with Minibatching
x1 x2 x3
concat
broadcast
b
W X B
tanh(+)

DyNetにおけるミニバッチ化

- DyNetにおけるミニバッチ化は比較的簡単
- 変更は２つのみ：
  - 入力時に各事例の入力をグループ化
  - 損失の計算などで各事例のラベルを同時に渡す

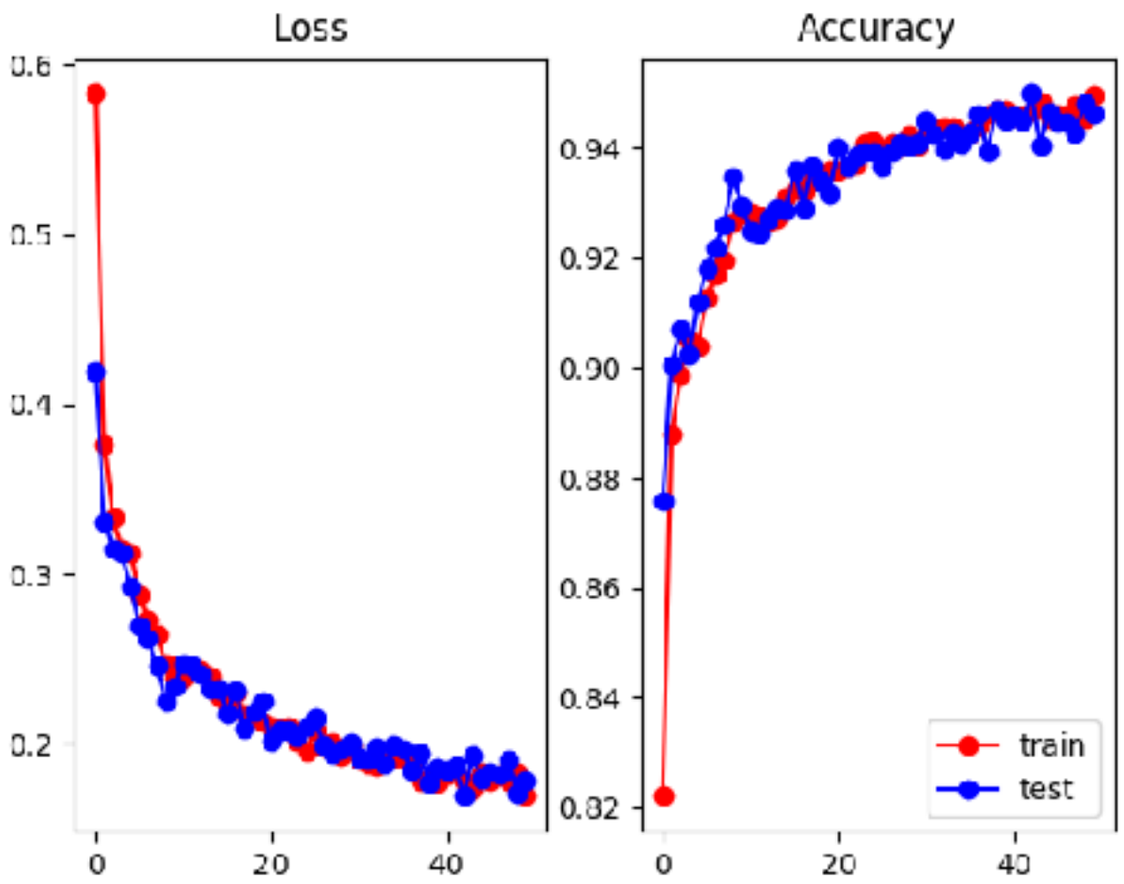
ミニバッチ化の例

- 04-mnist-minibatch.py



ミニバッチ：1文

1エポック3分



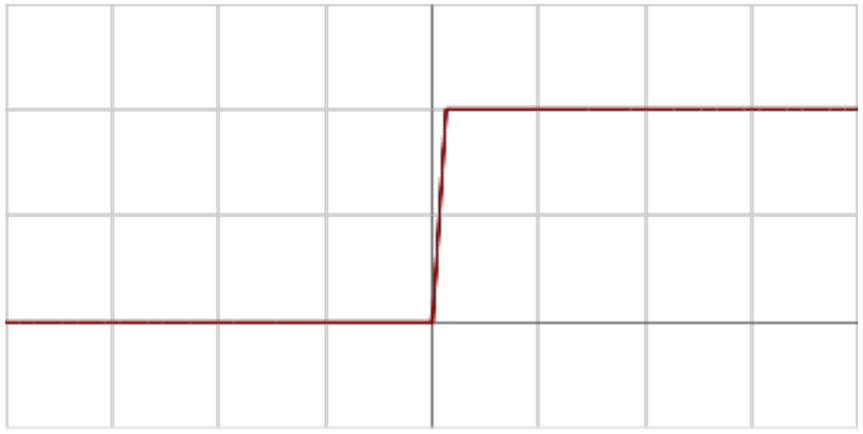
ミニバッチ：50文

1エポック10秒

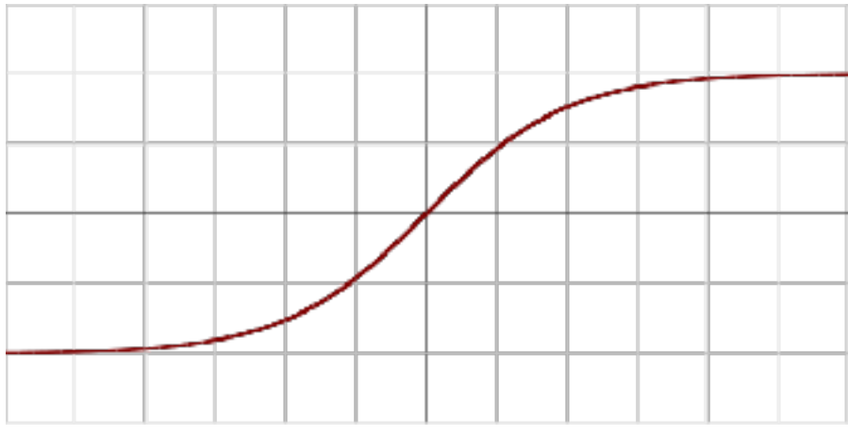
活性化関数（非線形関数）の調整

- 何かの非線形な関数を利用する必要がある
  - （そうでなければ、結局線形モデルと同じ表現力）

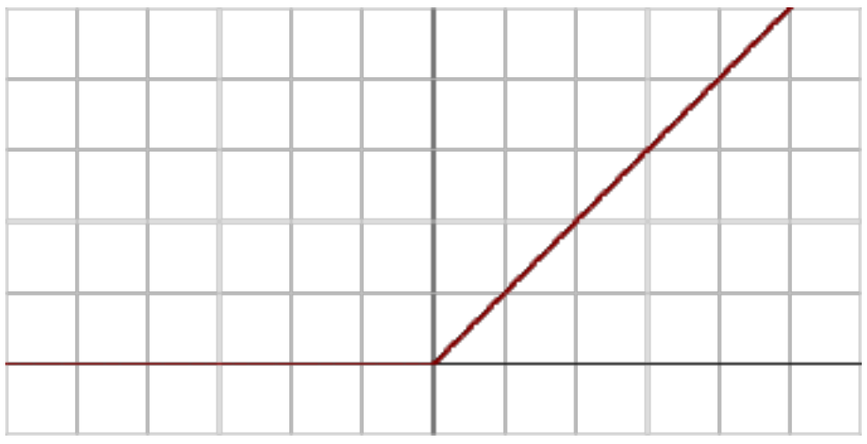
step



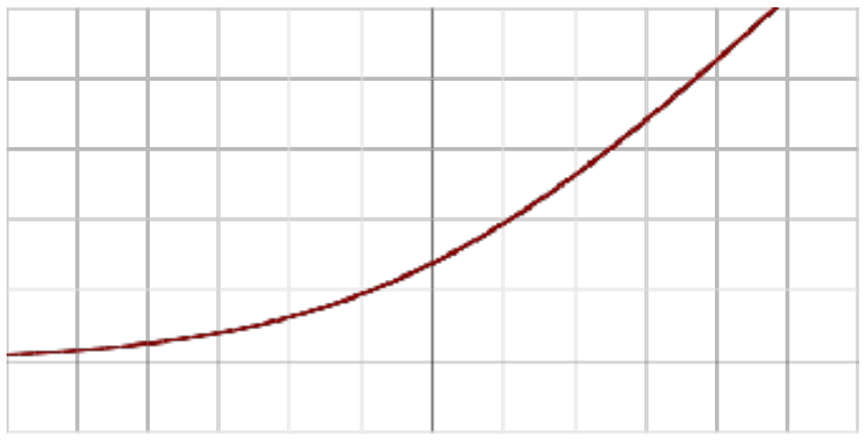
tanh



rectifier (RelU)



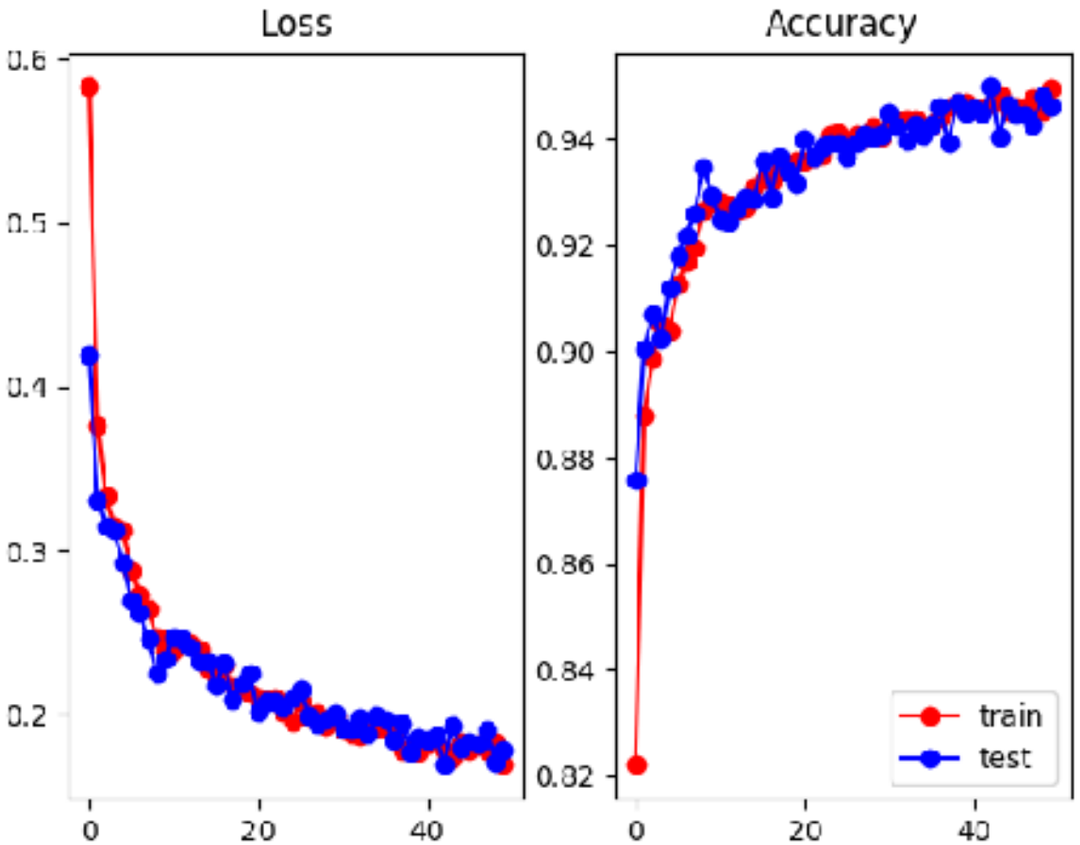
soft plus



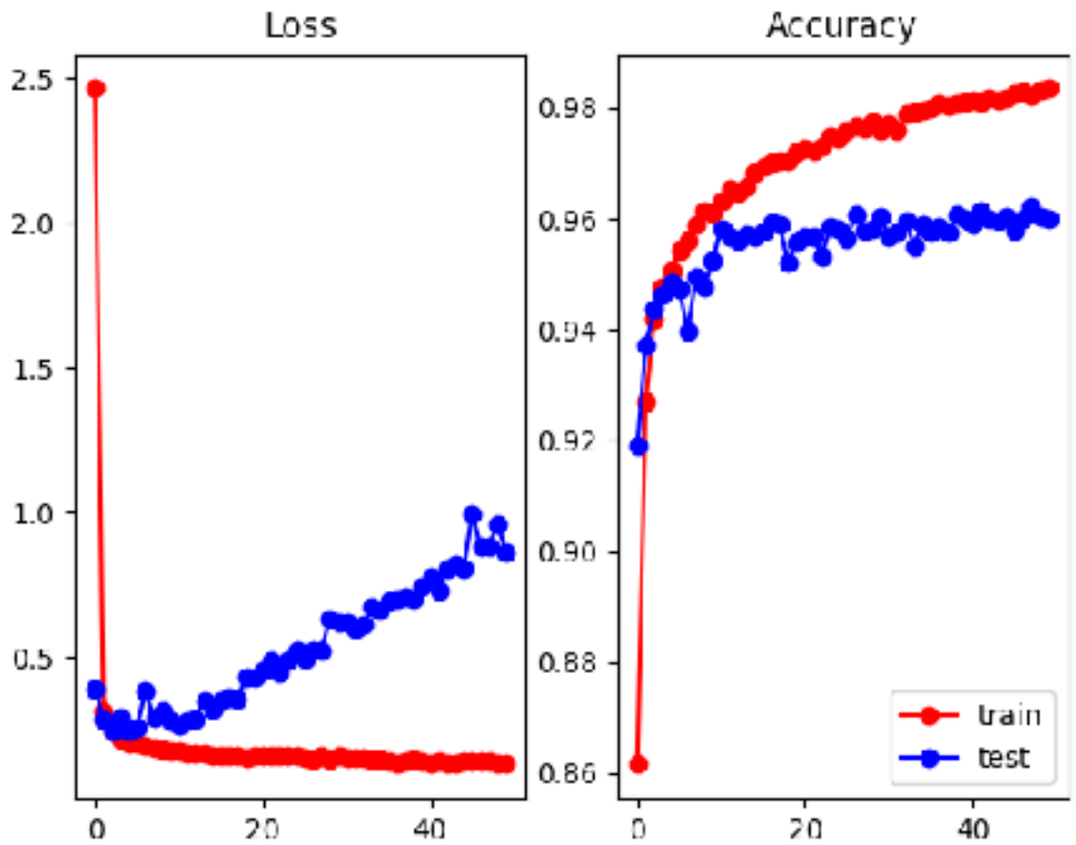
画像：Wikipedia

非線形関数の調整例

- 05-mnist-activation.py



tanh



relU

ネットの幅・深さ調整

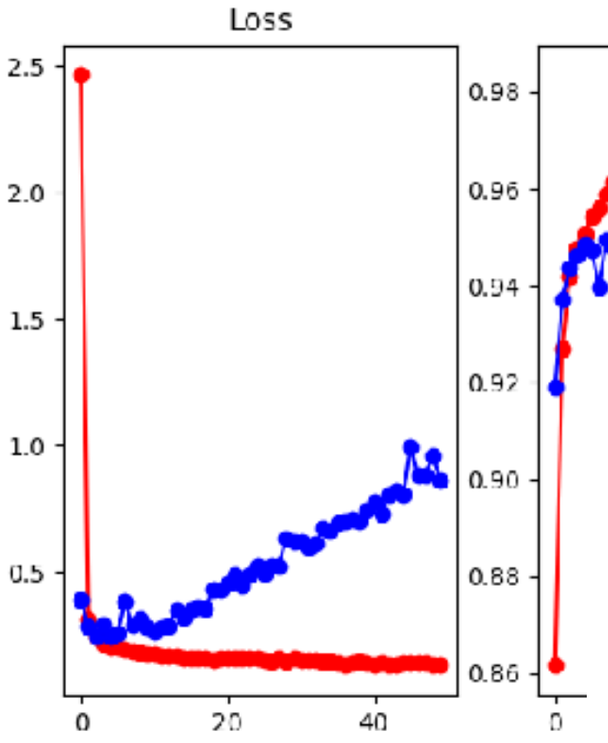
- **ネットの幅：**広いほど覚える要領が増えるが、過学習が起きやすくなる

- **ネットの深さ：**深いほど複雑な素性が学習可能であるが、たくさんの層を通して学習する必要があるため学習がしにくい

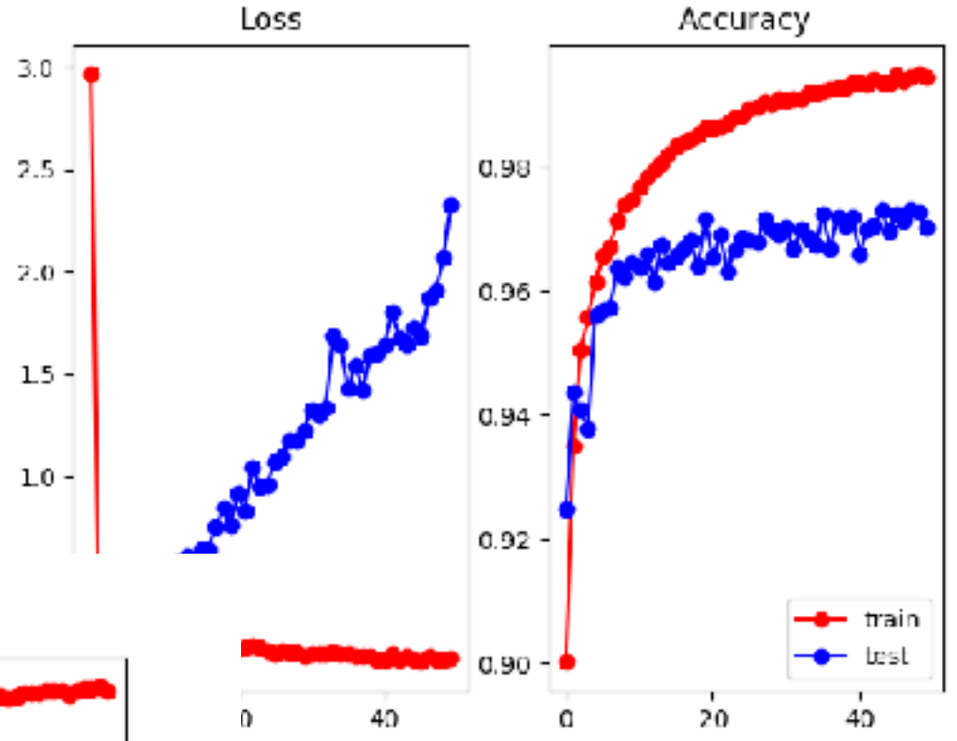
ネットの幅・深さ調整

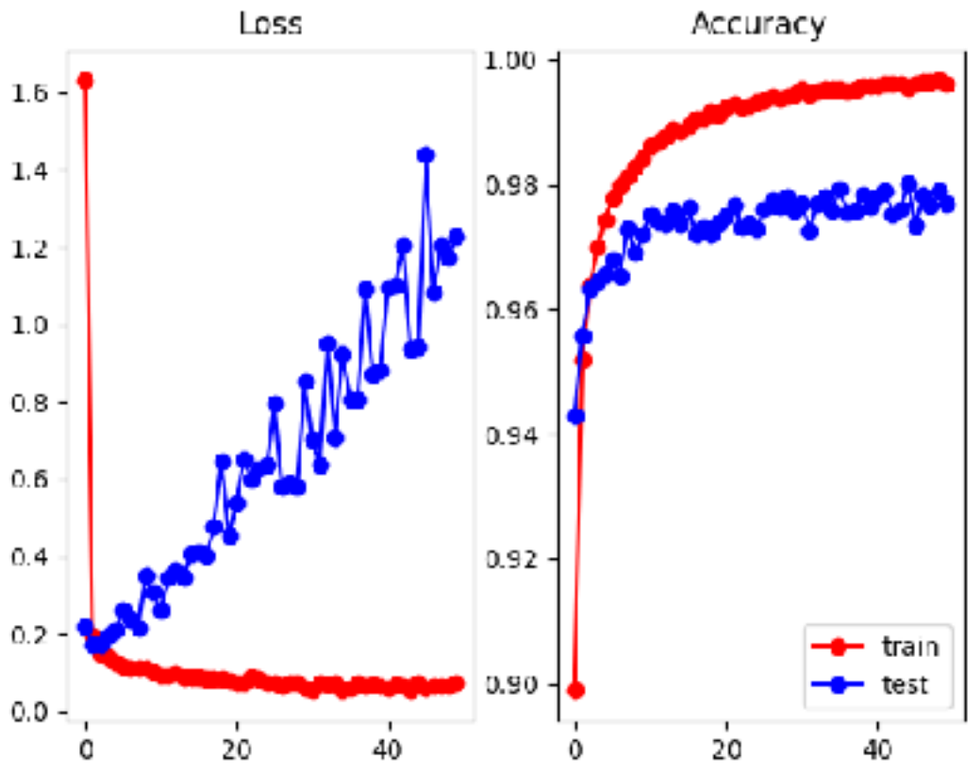
- 06-mnist-widerdeeper.py

隠れ層1層、幅128



隠れ層1層、幅512





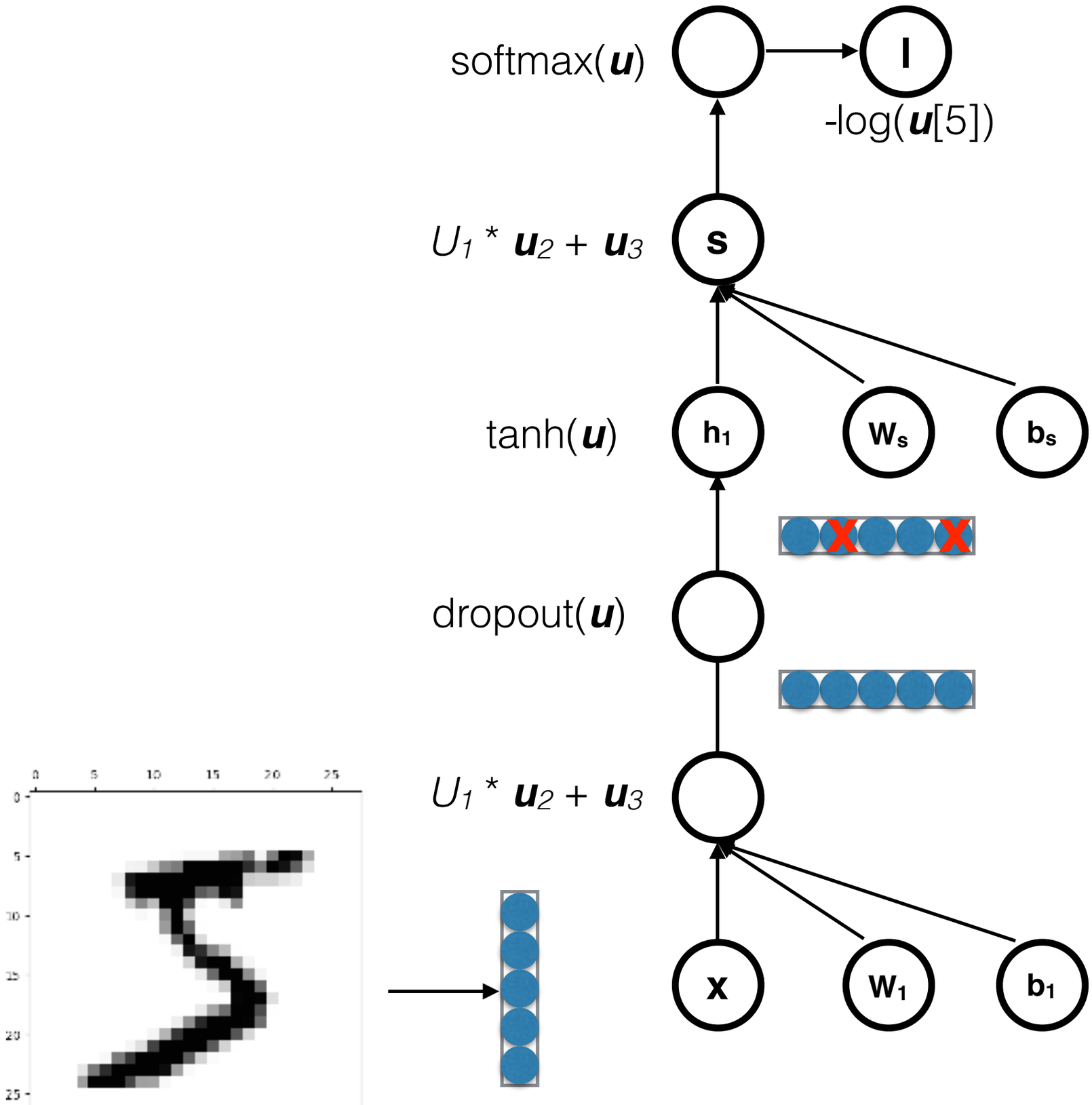
隠れ層2層

幅512

過学習を防ぐDropout

- ニューラルネットは学習事例に比べてたくさんのパラメータが存在→**過学習がしやすい**
- Dropoutのアイデア：
  - 学習時に毎回ランダムに隠れ層の１部をゼロに
  - １つの値に過剰に頼ることを防ぐ
  - テスト時は通常通り、ネットを全部使う

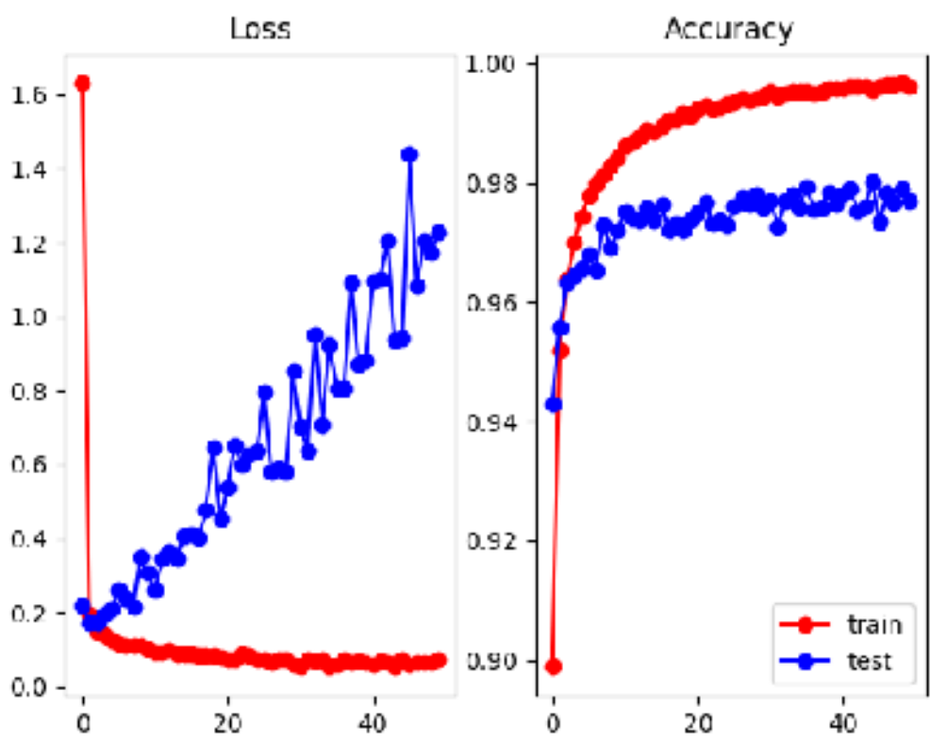
Dropoutの仕組み



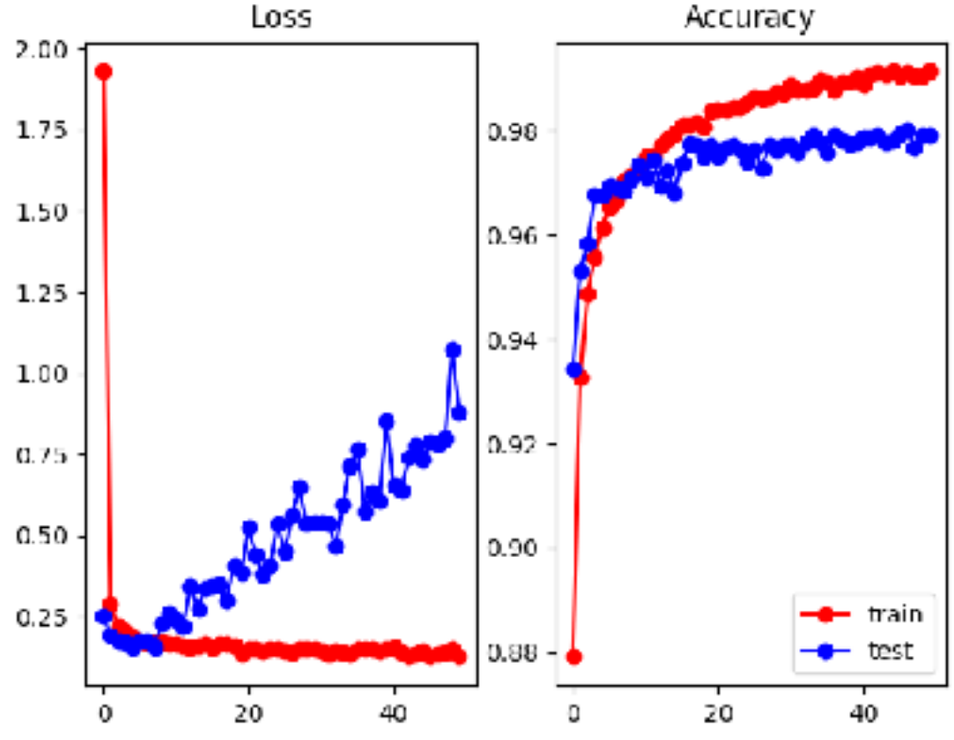
Dropoutの例

- 07-mnist-dropout.py

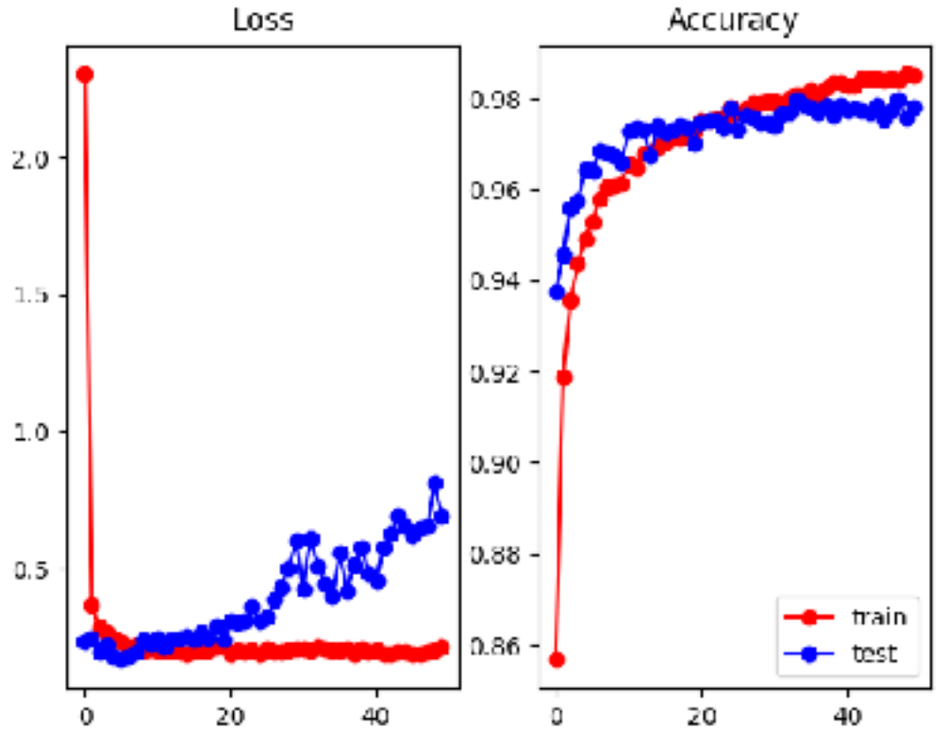
do=0.0



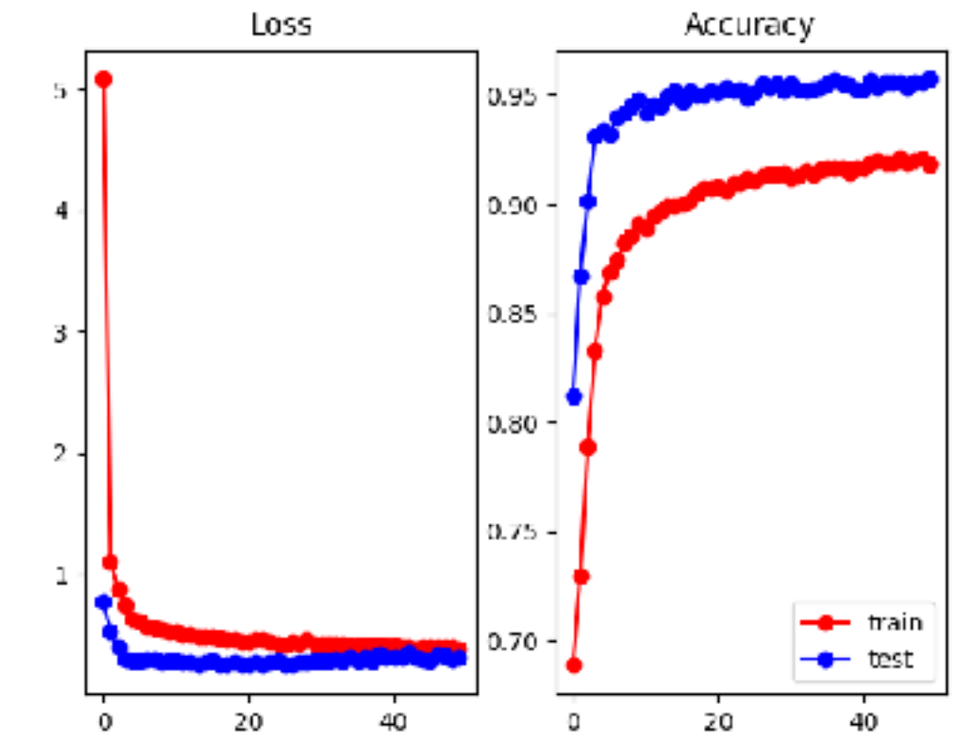
do=0.1



do=0.2



do=0.5

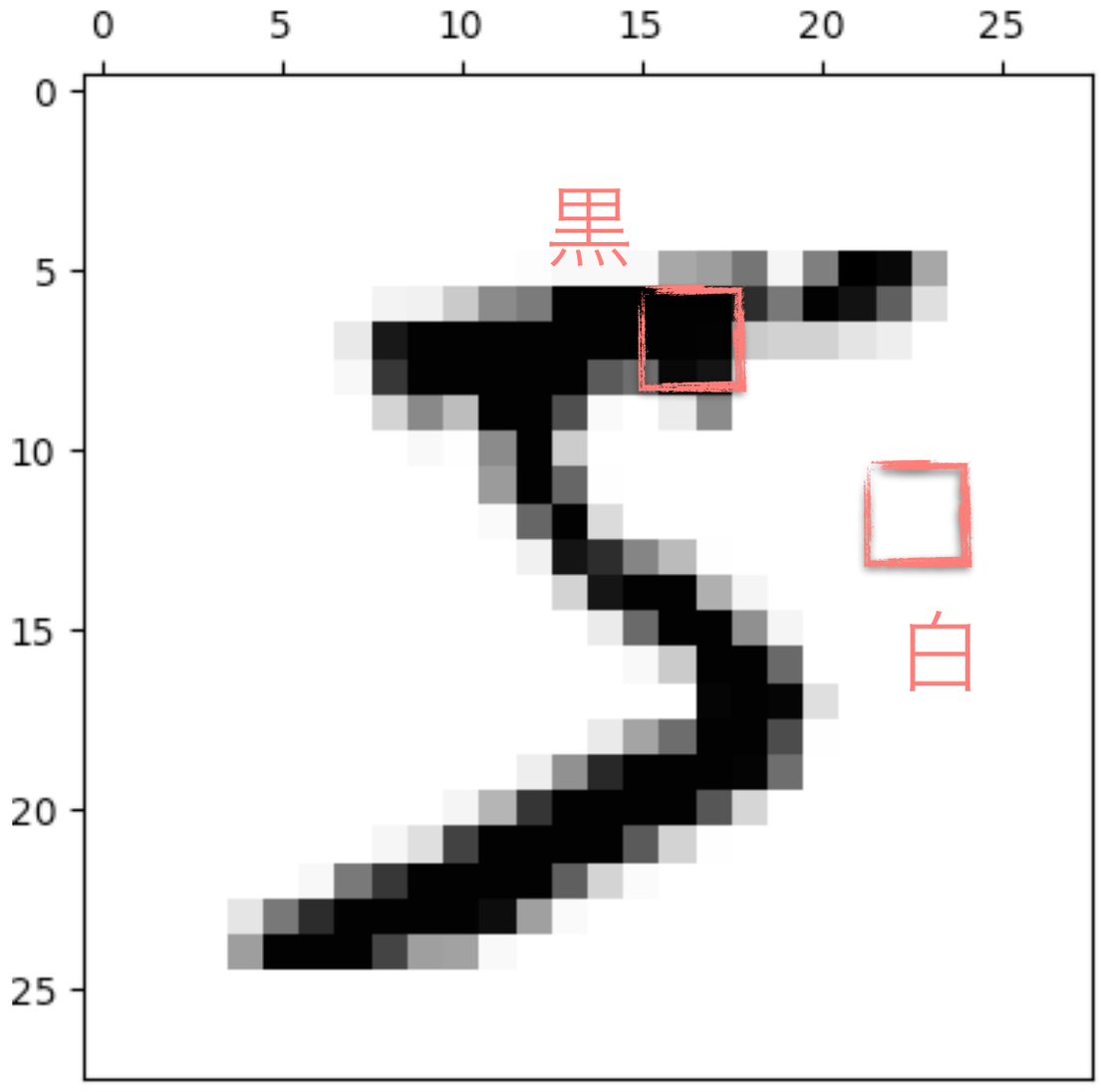


畳み込みニューラルネット

(Convolutional Neural Net; CNN)

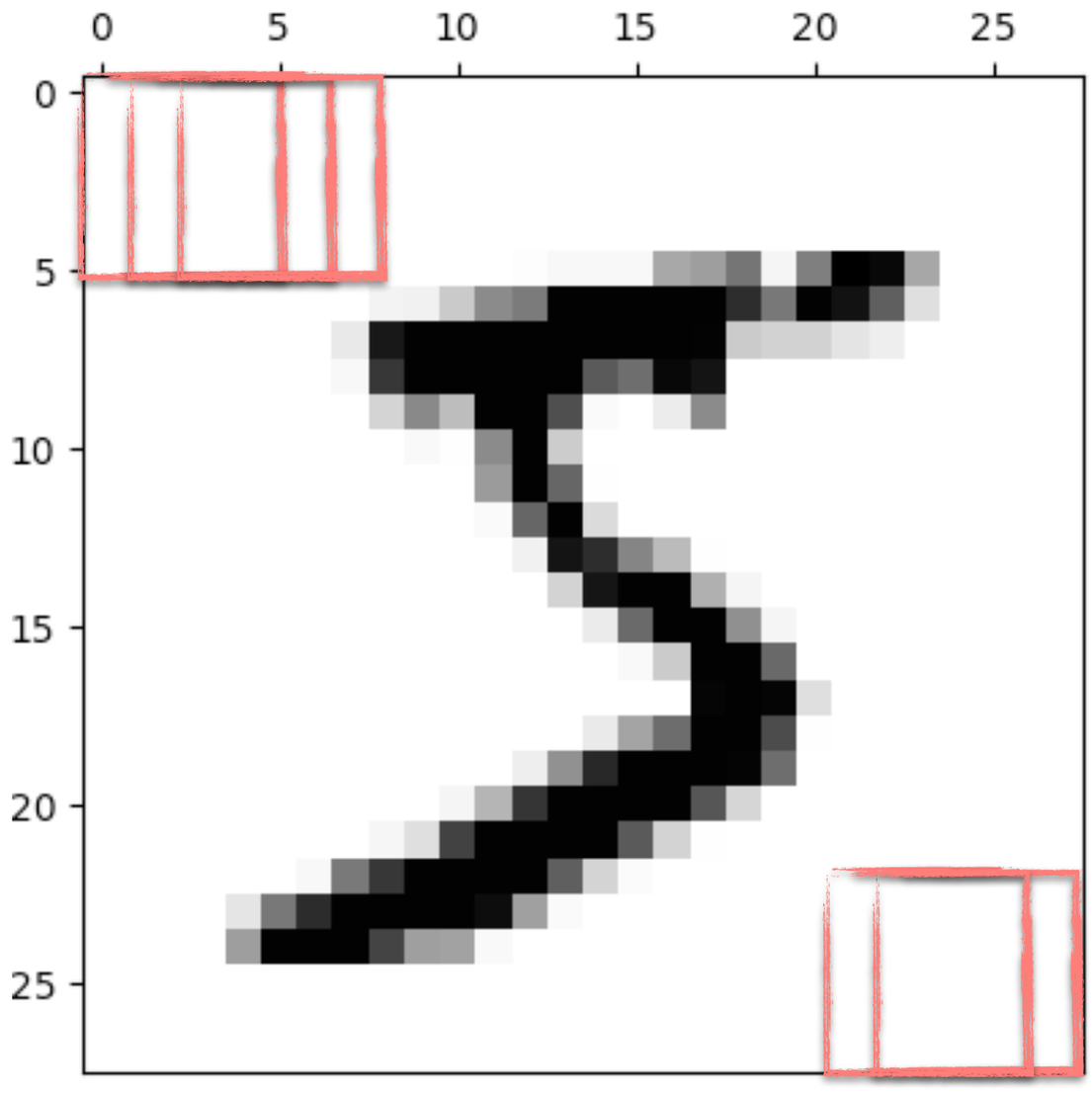
モチベーション：

画像等の局所的な一貫性



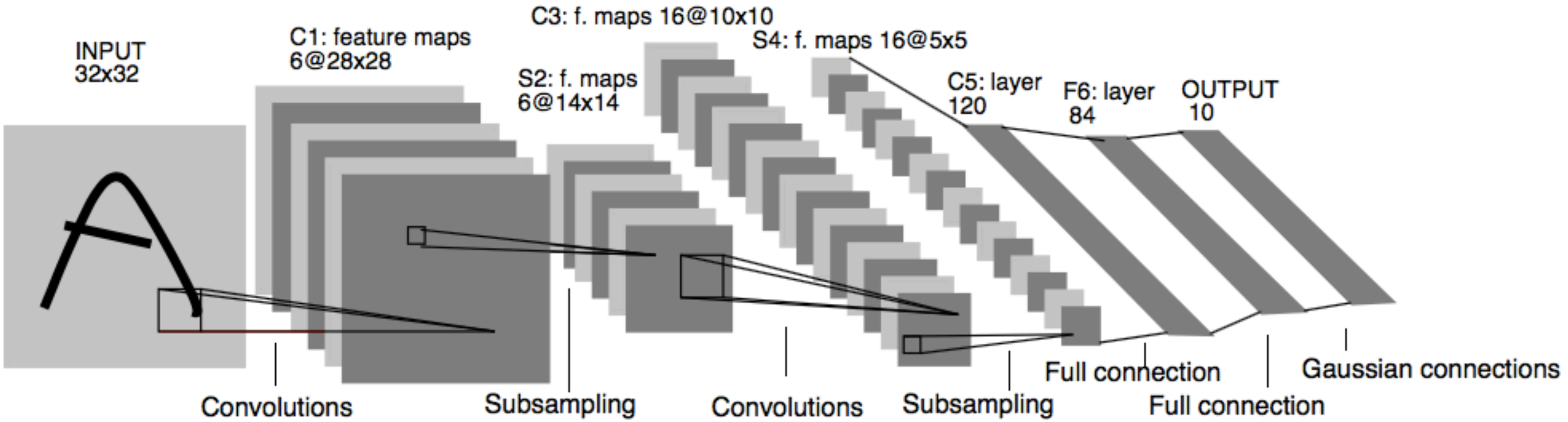
畳み込みニューラルネット

- 同じ場所にあるものを同時に処理



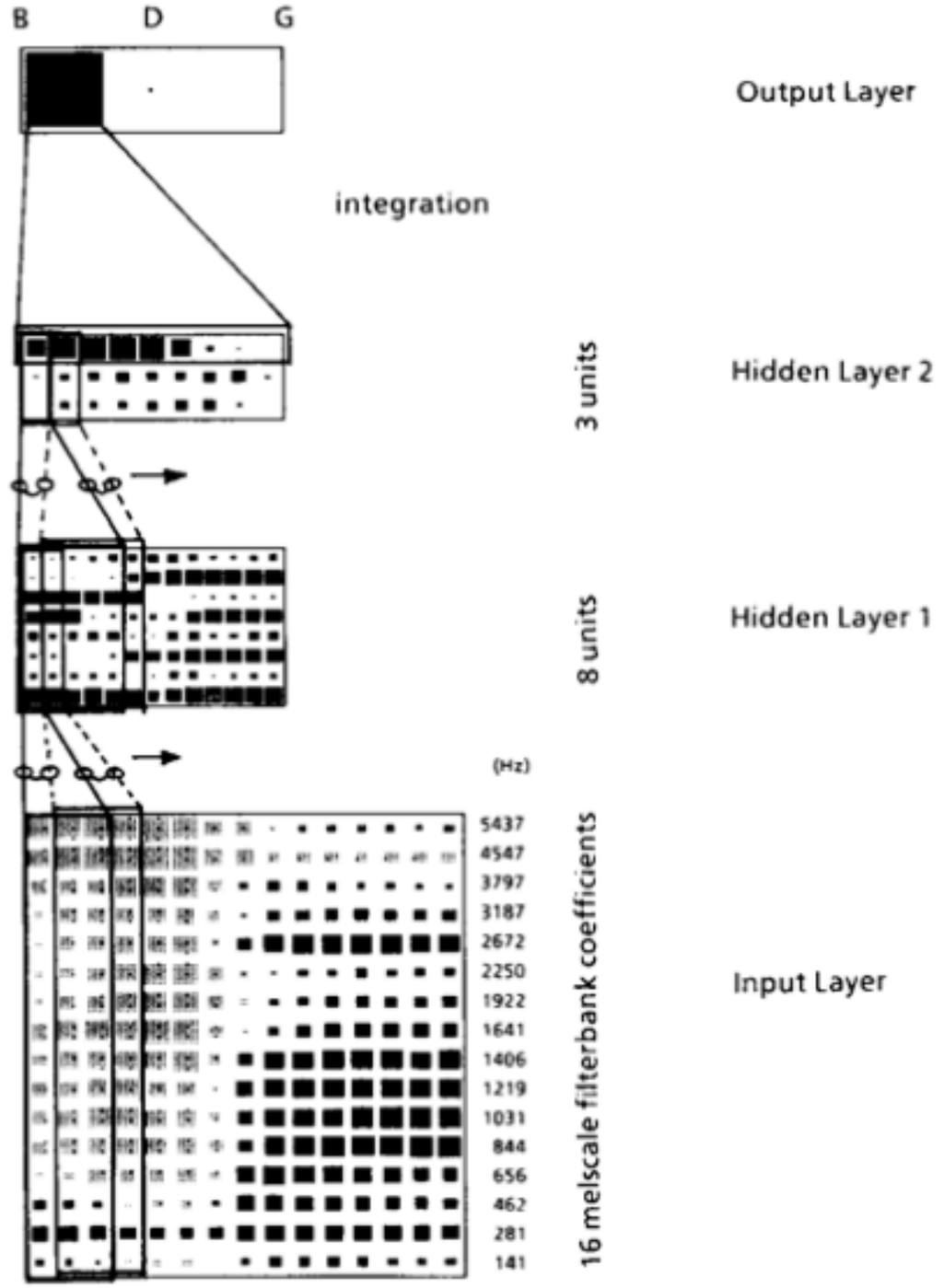
- 場所が少しずれても、特徴を抽出可能

例：文字認識 (Lecun 98)



例：音素認識 (Waibel+89)

- Time Delay Neural Networks



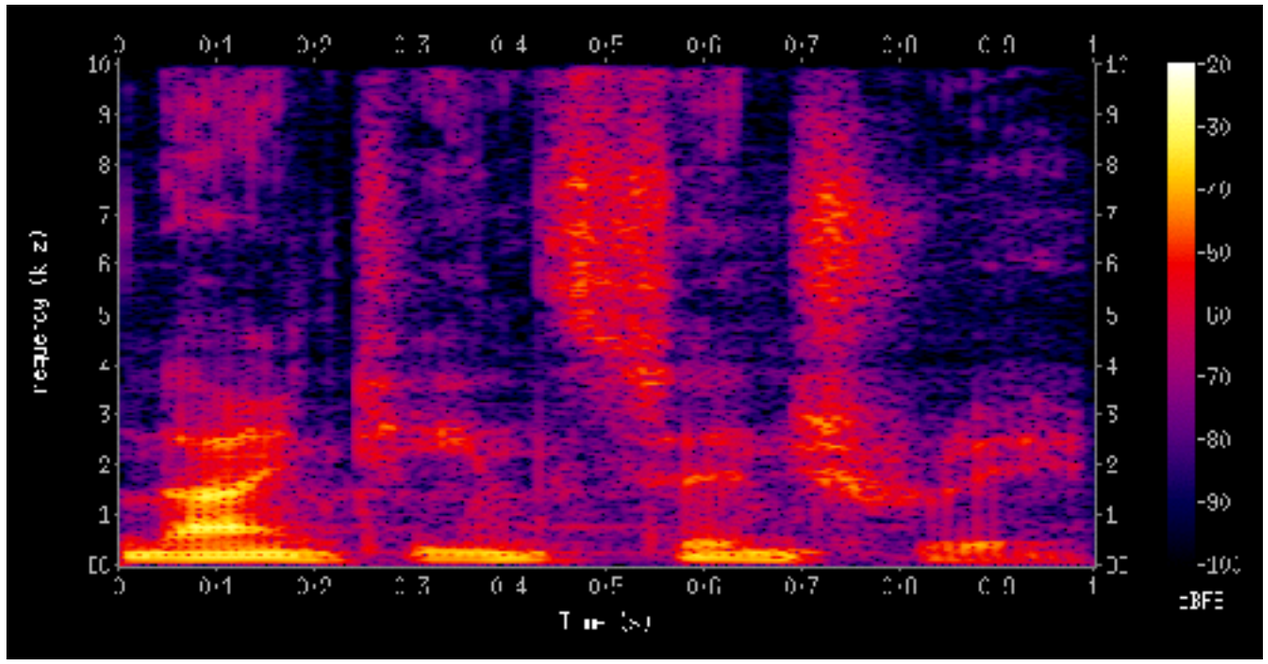
畳み込みネットの例

- 08-mnist-cnn.py

可変長な系列の扱い

可変長な系列

- 音声

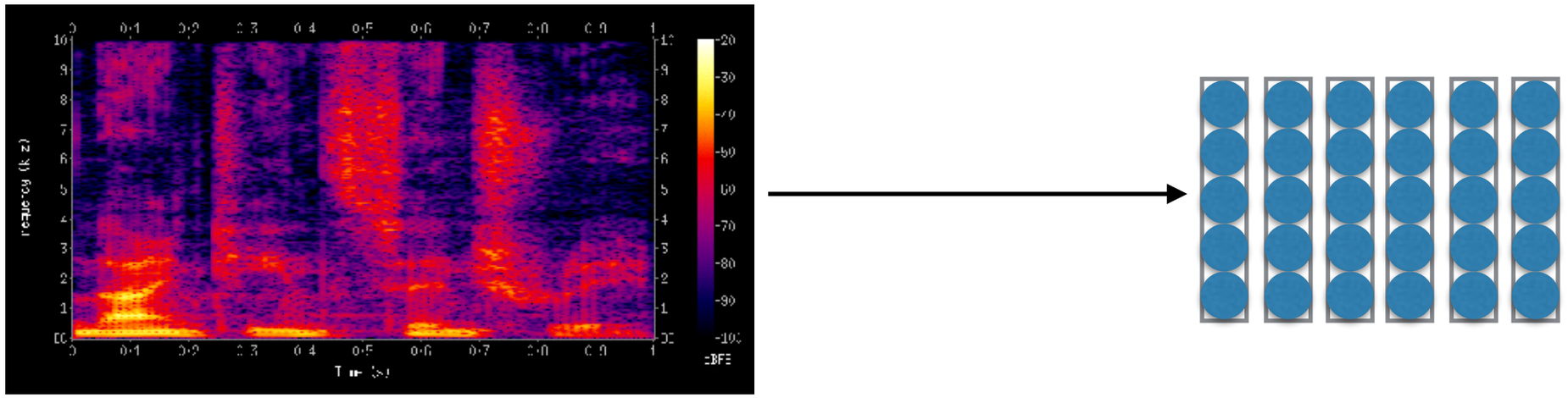


- テキスト

this is a pen
i really like this pen

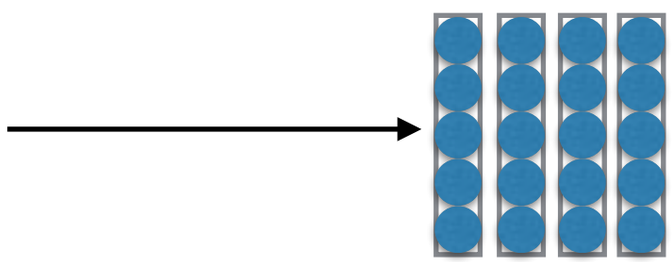
系列からベクトルへ

- 音声はMFCCなどの特徴量抽出

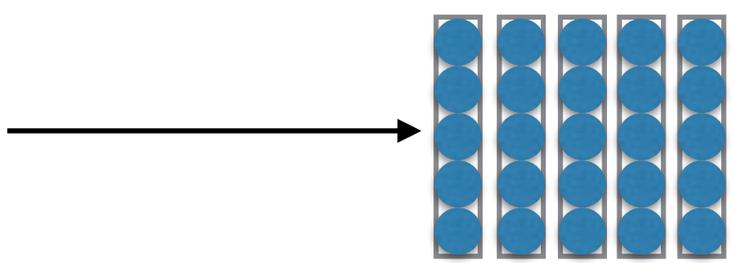


- テキストは単語ベクトル

this is a pen



i really like this pen



単語ベクトルと LookupParameters

- 言語に対するネットの応用で、各単語がベクトルで表現
- 事前学習による計算が可能（word2vec）
- ただし、ほとんどの場合はモデルと一緒に学習

単語ベクトルと LookupParameters

- DyNetでは単語ベクトルを `LookupParameters`として実装

```
vocab_size = 10000
emb_dim = 200

E = model.add_lookup_parameters((vocab_size, emb_dim))
```

単語ベクトルと LookupParameters

- DyNetでは単語ベクトルを `LookupParameters`として実装

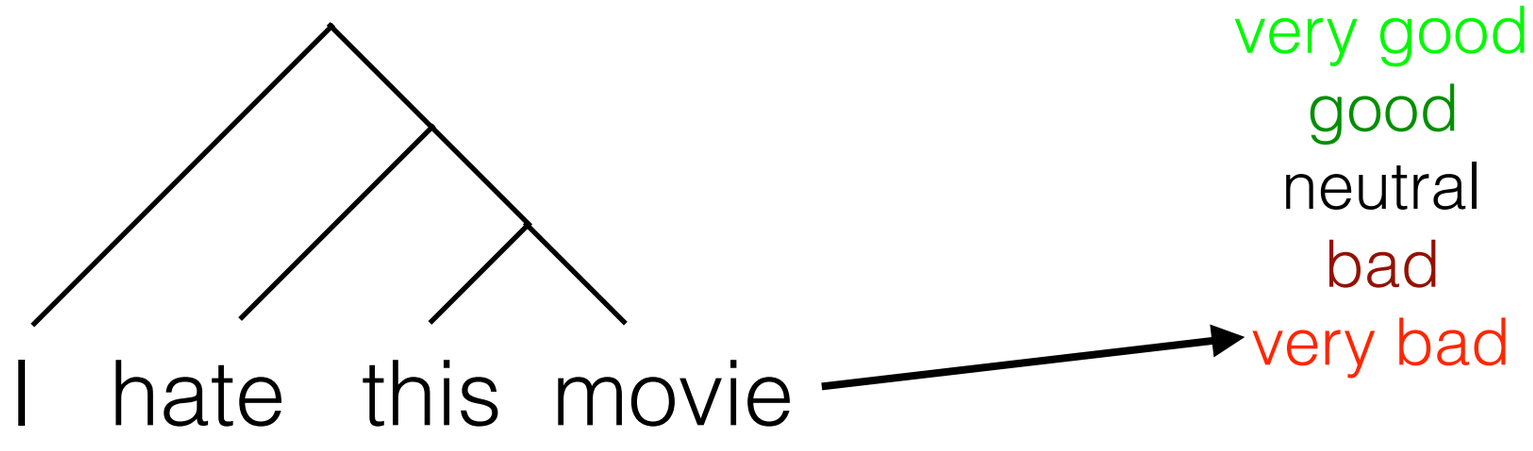
```
vocab_size = 10000
emb_dim = 200

E = model.add_lookup_parameters((vocab_size, emb_dim))


dy.renew_cg()
x = dy.lookup(E, 5)
# or
x = E[5]
# x is an expression
```

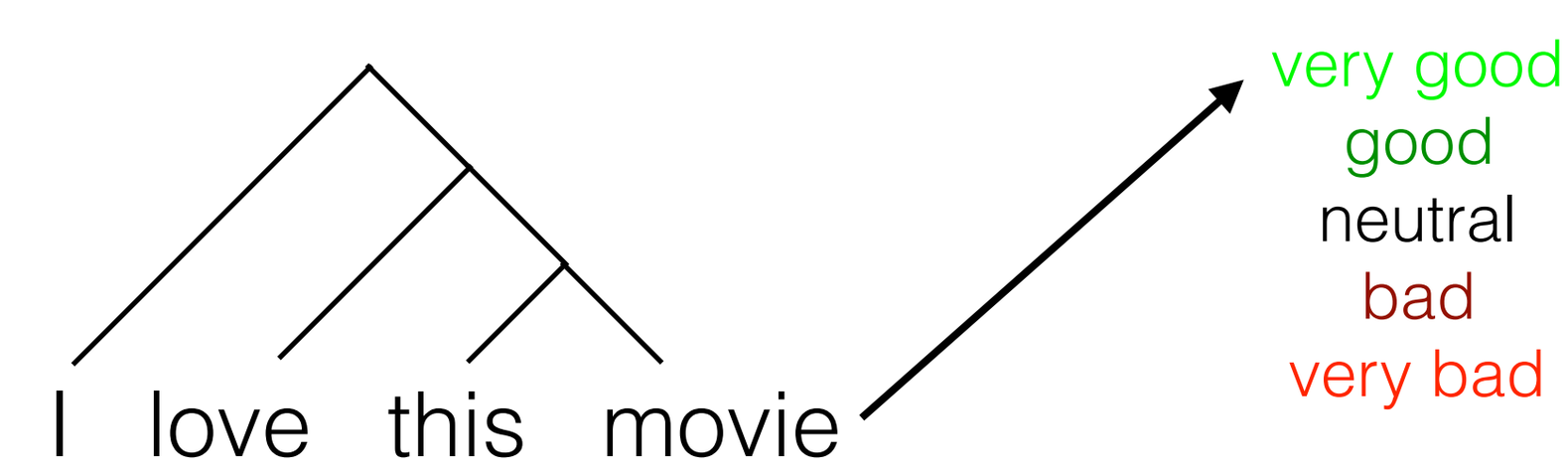
系列の分類：事例

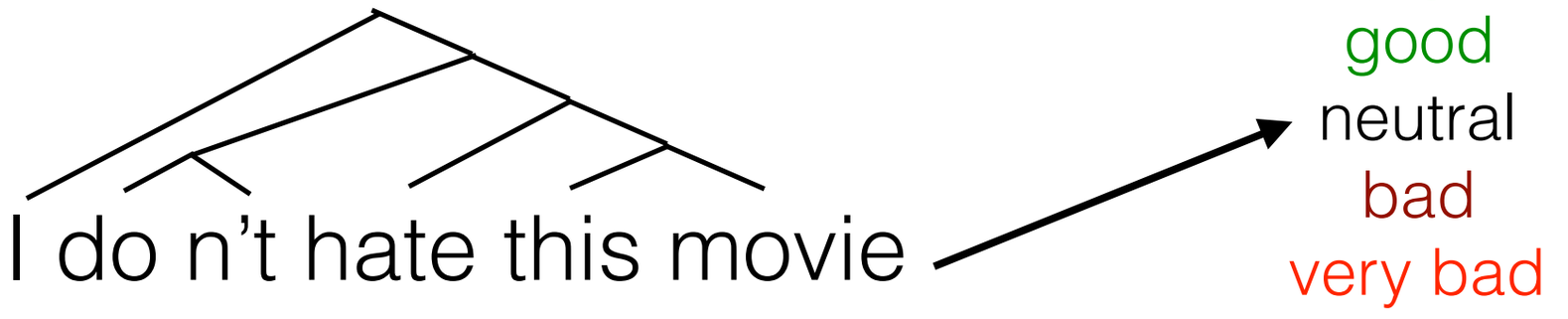
なぜ系列を分類するのか？

- テキストの分類
  - トピック
  - 感情極性分類
- 音声の分類
  - 話者分類
  - 感情認識

感情極性分類


I hate this movie
very good
good
neutral
bad
very bad


I love this movie
very good
good
neutral
bad
very bad

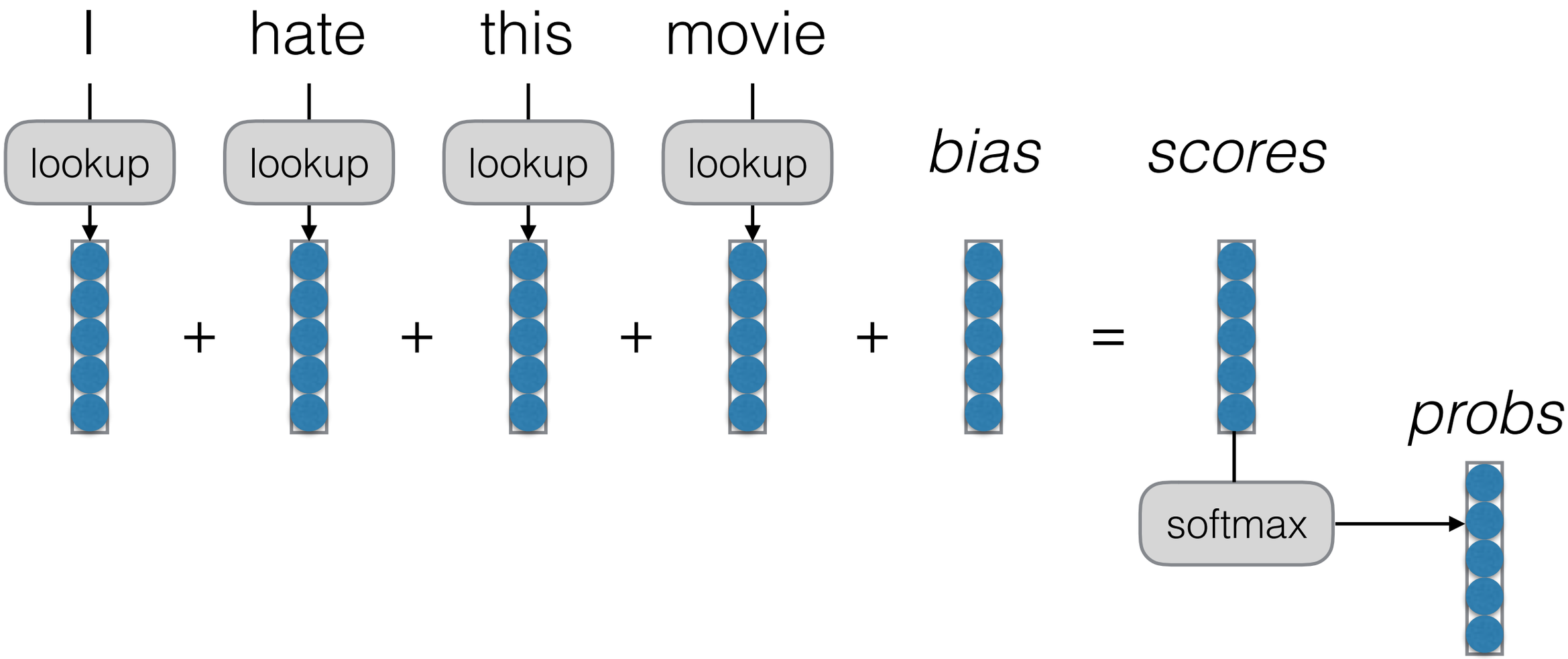

I do n’t hate this movie
very good
good
neutral
bad
very bad

足し算によるモデル

足し算による系列モデル

- 単純にベクトルを足すだけ
- ベクトルの平均が何か情報を持つ場合
  - 音声の男女分類：周波数の平均を取る
  - 感情極性分類：単語ベクトルの平均を取る

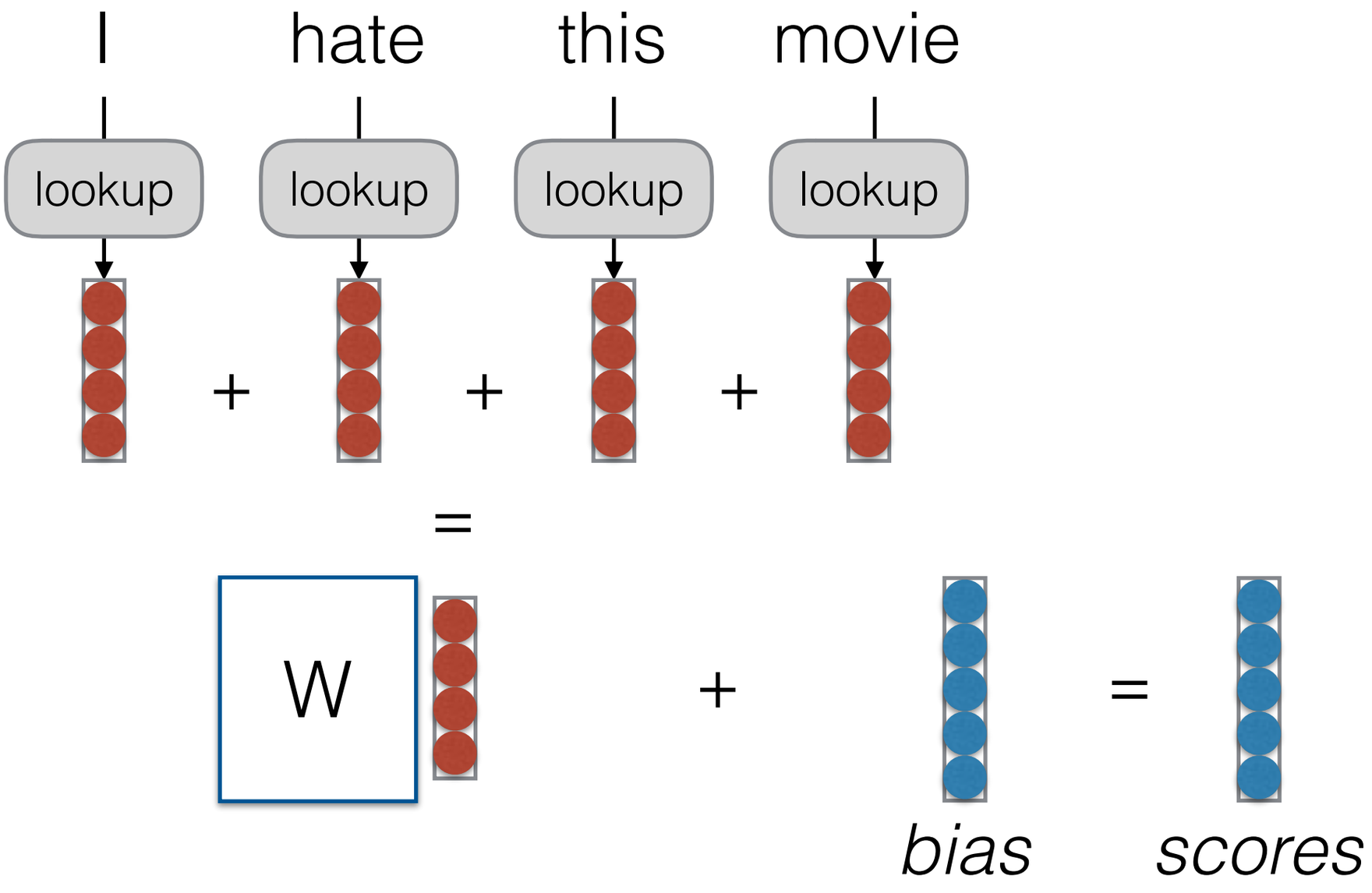
Bag of Words (BOW)



Bag of Wordsの例

- 10-sentiment-bow.py

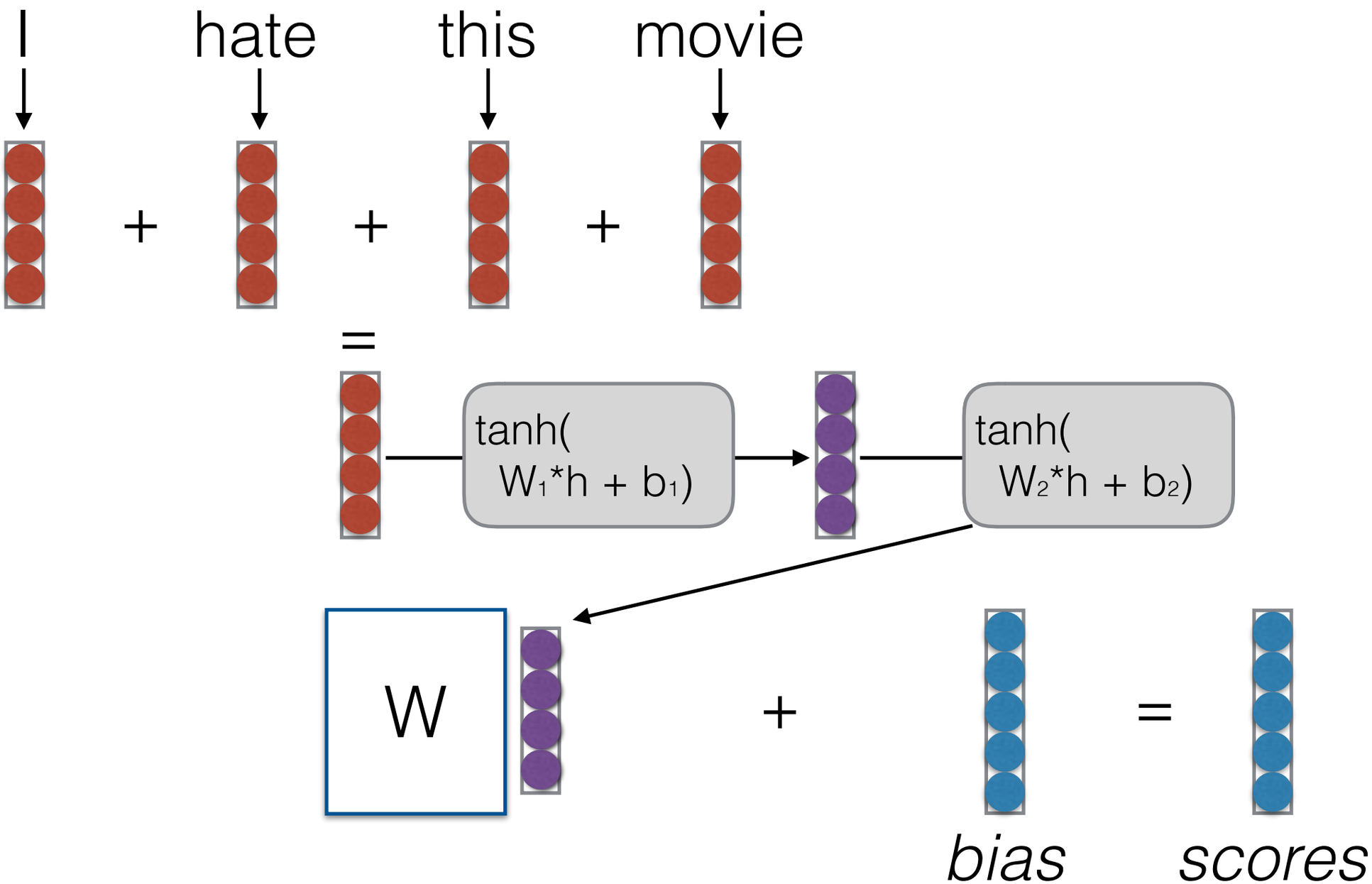
Continuous Bag of Words (CBOW)



CBOWの例

- 11-sentiment-cbow.py

Deep CBOW



Deep CBOWの例

- 12-sentiment-deepcbow.py

リカレントニューラルネット

リカレントニューラルネット

(RNN)

- NLP is full of sequential data
  - Words in sentences
  - Characters in words
  - Sentences in discourse
  - …
- **How do we represent an arbitrarily long history?**

リカレントニューラルネット

(RNN)

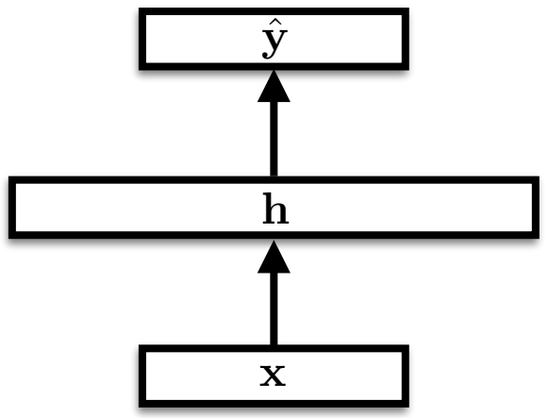
- NLP is full of sequential data
  - Words in sentences
  - Characters in words
  - Sentences in discourse
  - …
- **How do we represent an arbitrarily long history?**
  - we will train neural networks to build a representation of these arbitrarily big sequences

リカレントニューラルネット

通常のfeed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

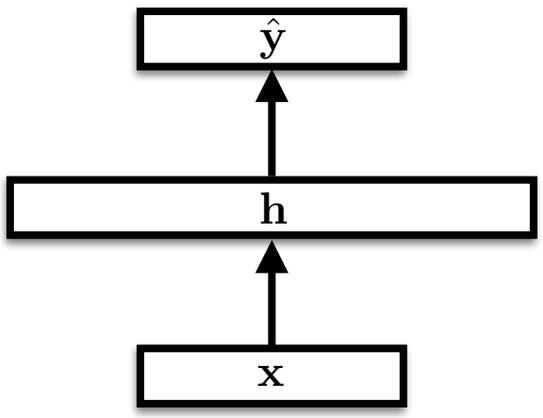


リカレントニューラルネット

通常のfeed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

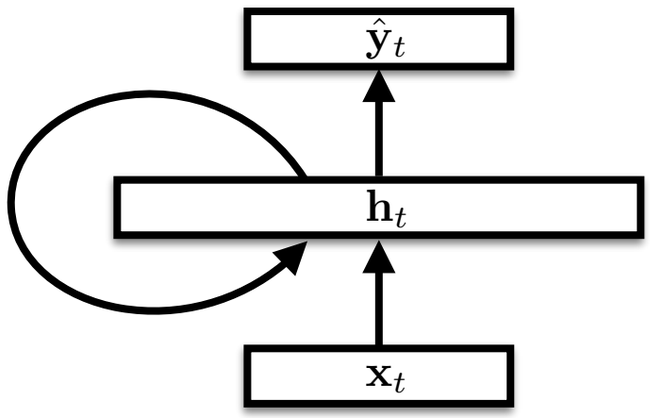
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent NN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

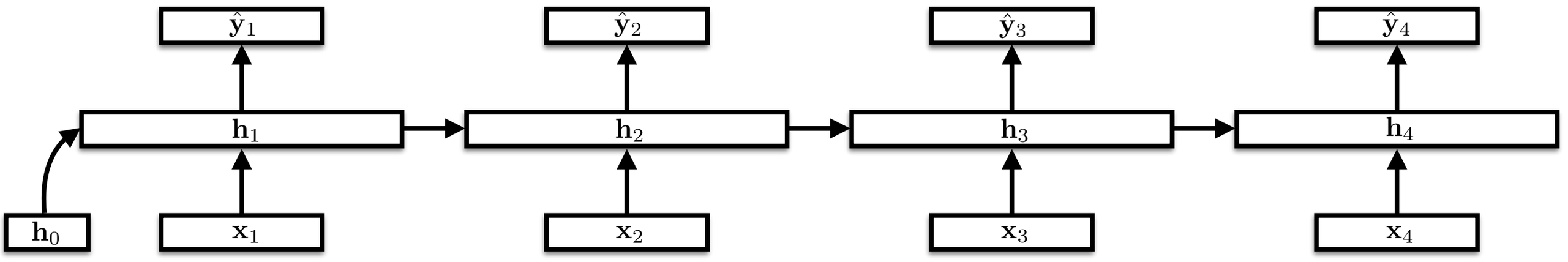
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



時間ごとに展開されたRNN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

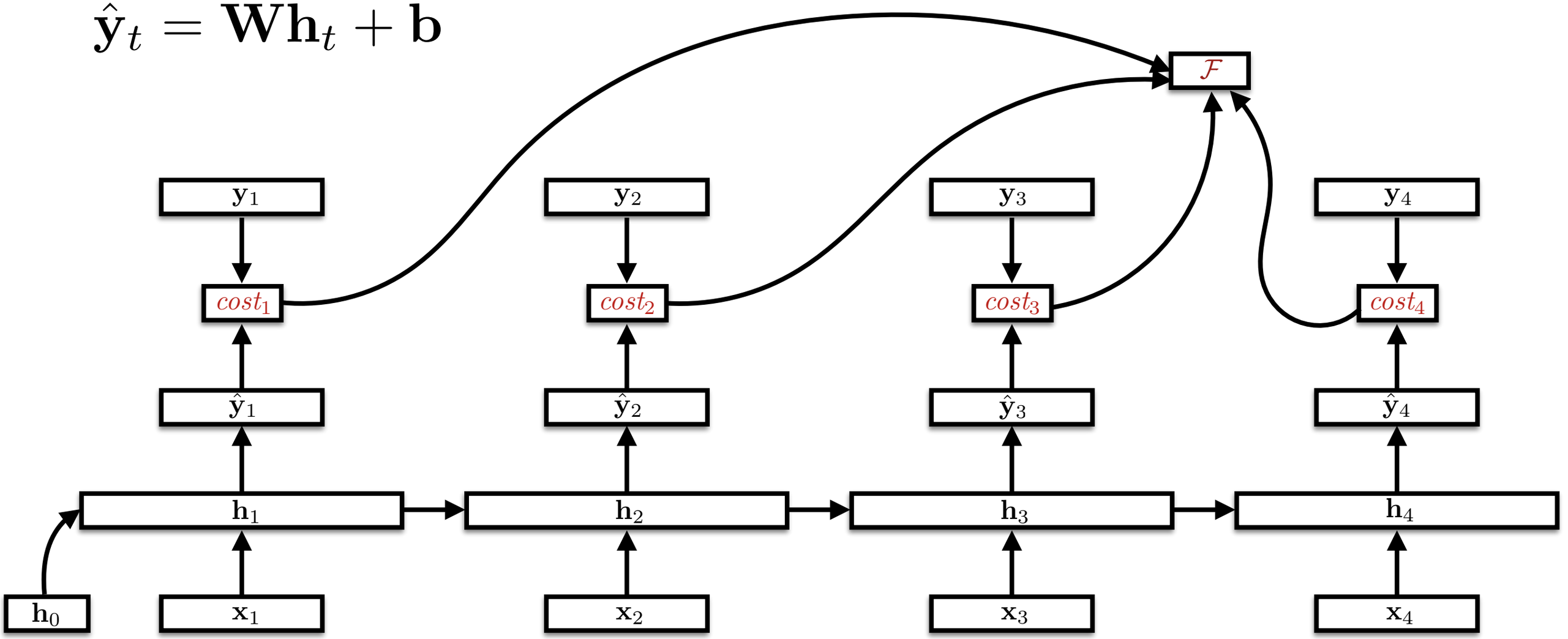
RNNのパラメータをどう学習するか？



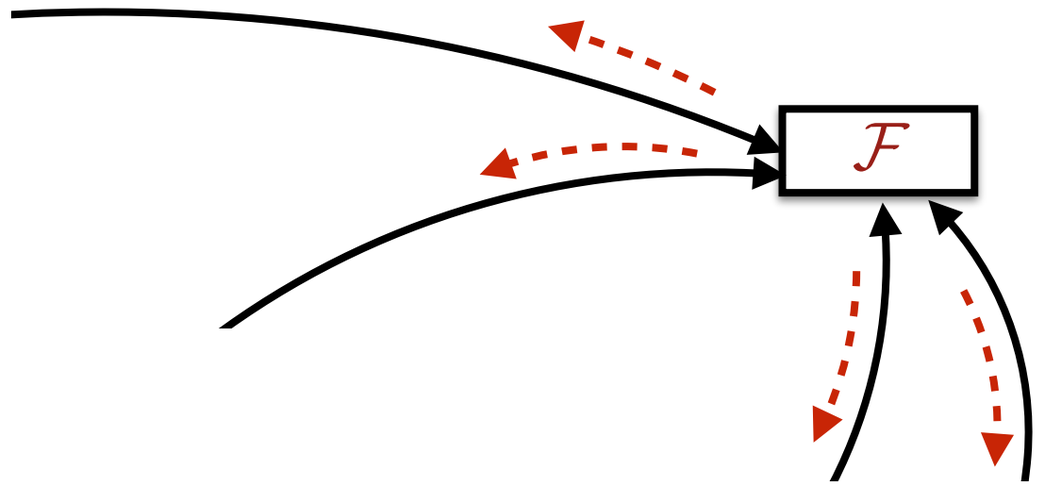
時間ごとに展開されたRNN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



RNNの学習

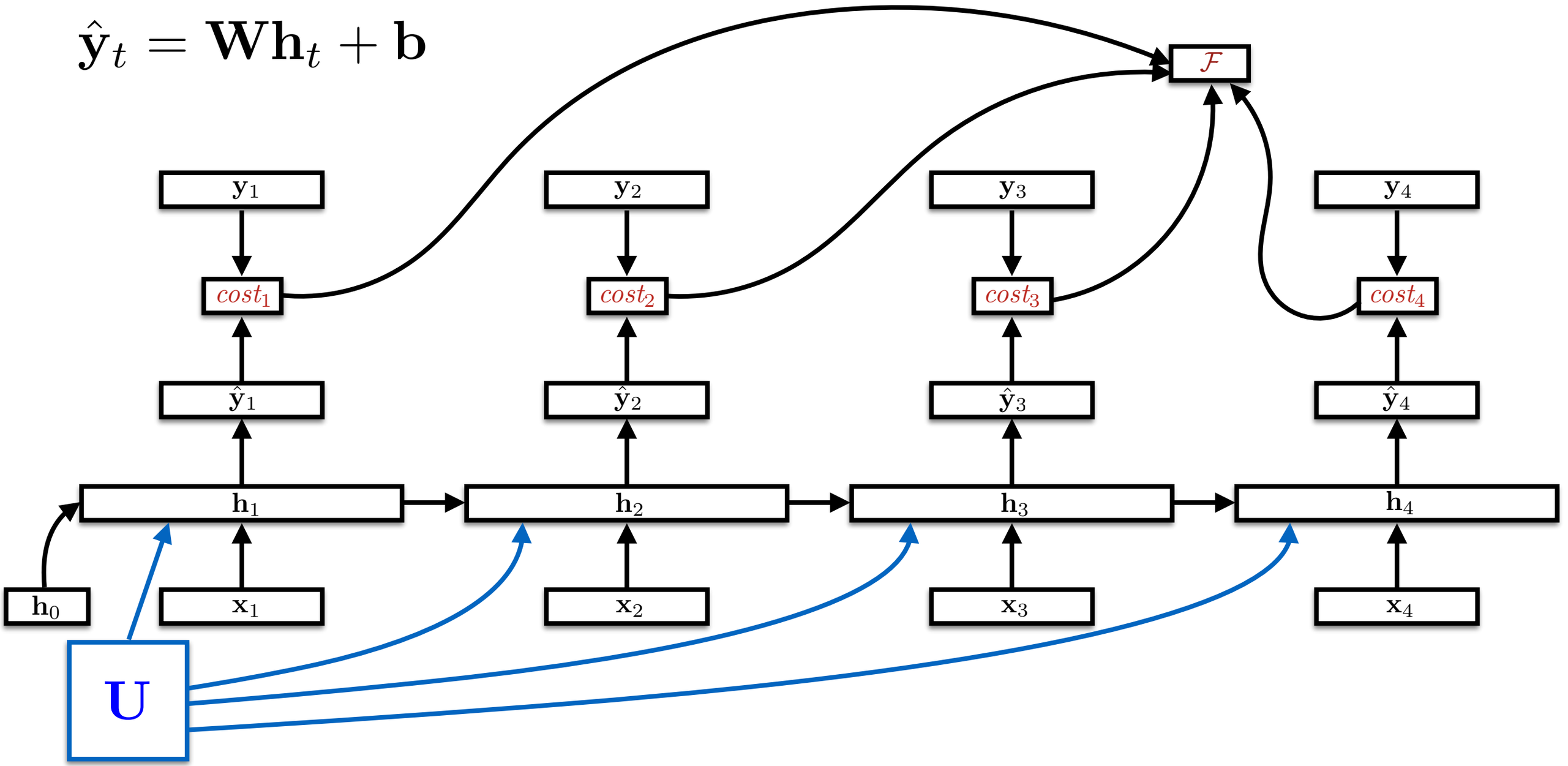


- 時間ごとに展開されたRNNは普通の有向比巡回グラフ：通常のNNと同じように勾配の計算が可能
  - パラメータは複数の時間で共有：勾配は各時間による勾配の累積
  - “backpropagation through time” (BPTT)とも

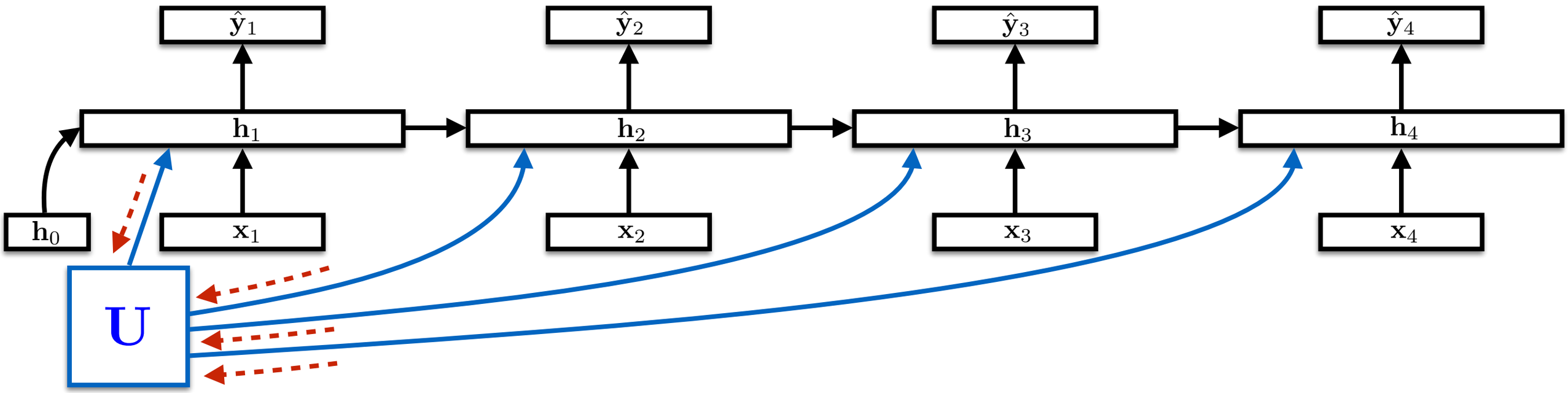
パラメータの共有

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



パラメータの共有



$$\frac{\partial \mathcal{F}}{\partial \mathbf{U}} = \sum_{t=1}^{4} \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}} \frac{\partial \mathcal{F}}{\partial \mathbf{h}_t}$$

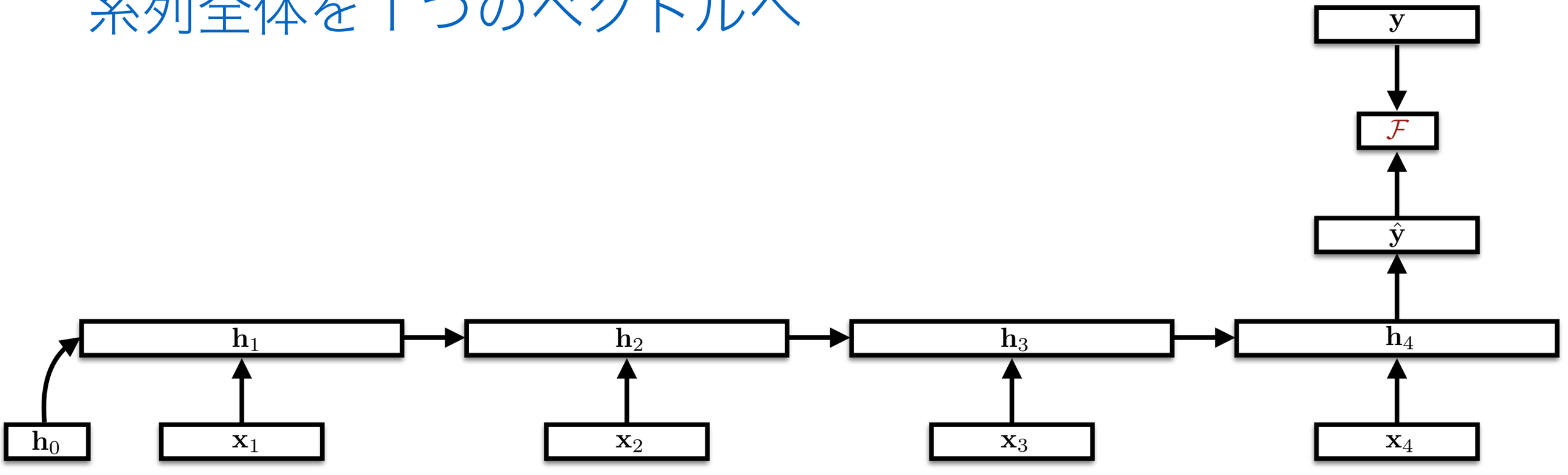
他の応用：

系列全体の表現を計算

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$

系列全体を１つのベクトルへ



DyNetのRNN

- “*Builder”クラスを利用 (*=SimpleRNN/LSTM)
- パラメータをモデルへ追加（学習開始時１回のみ）

```
# LSTM (layers=1, input=64, hidden=128, model)
RNN = dy.LSTMBuilder(1, 64, 128, model)
```

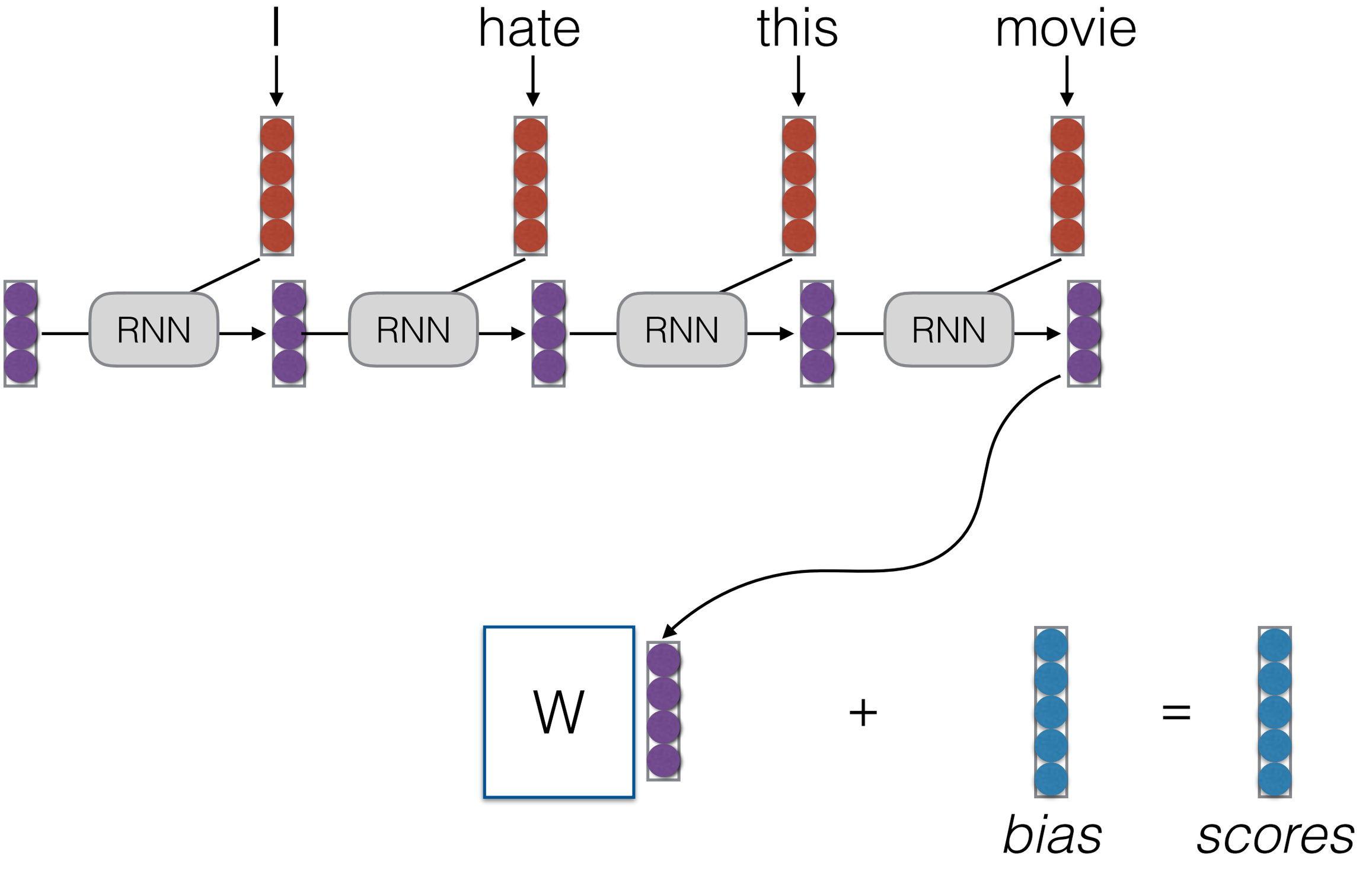
- パラメータをグラフへ追加し、開始状態を獲得(系列ごと)

```
s = RNN.initial_state()
```

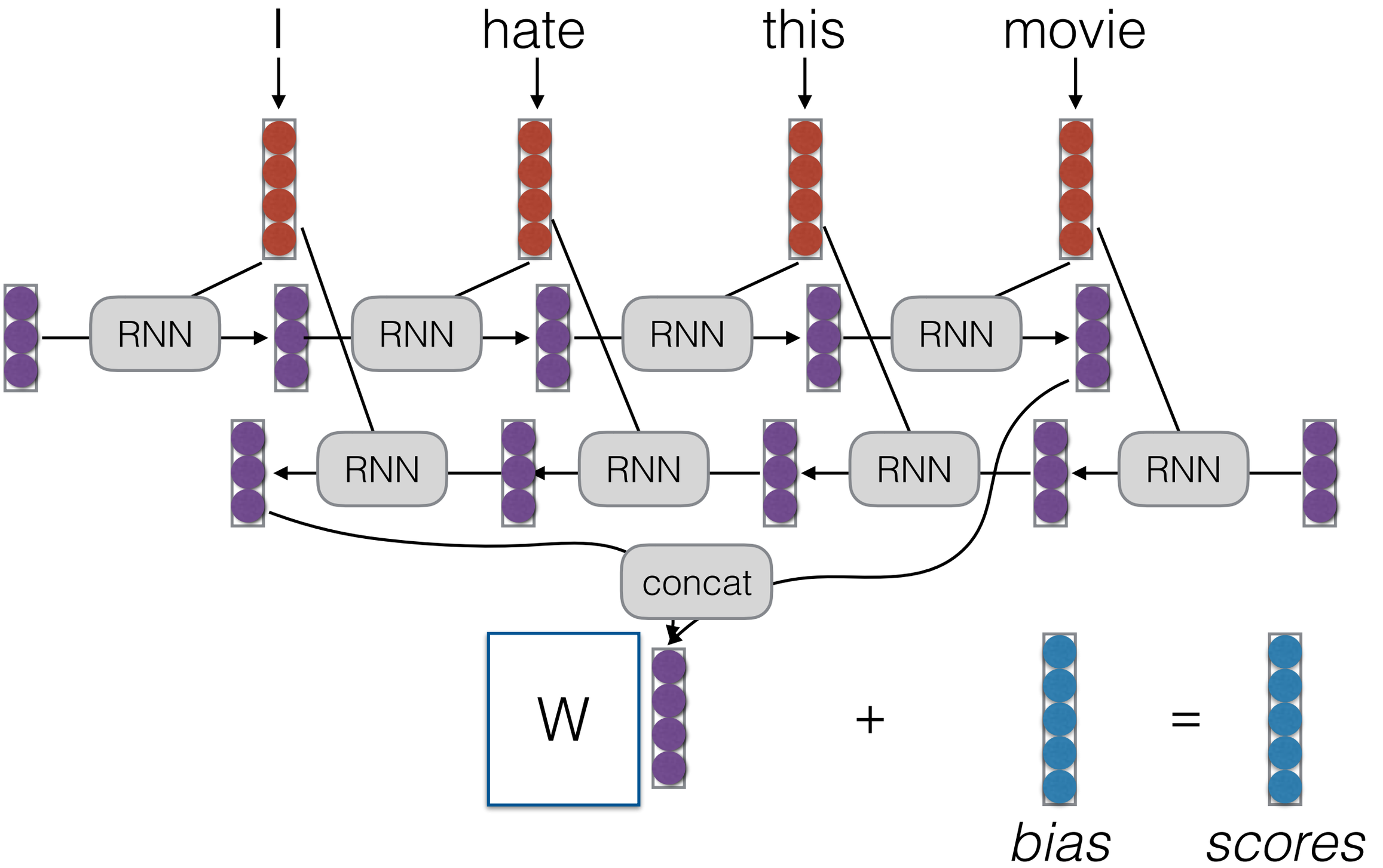
- 状態を更新し、獲得（入力ごと）

```
s = s.add_input(x_t)
h_t = s.output()
```

RNNで文を読んで判断



両方向RNN

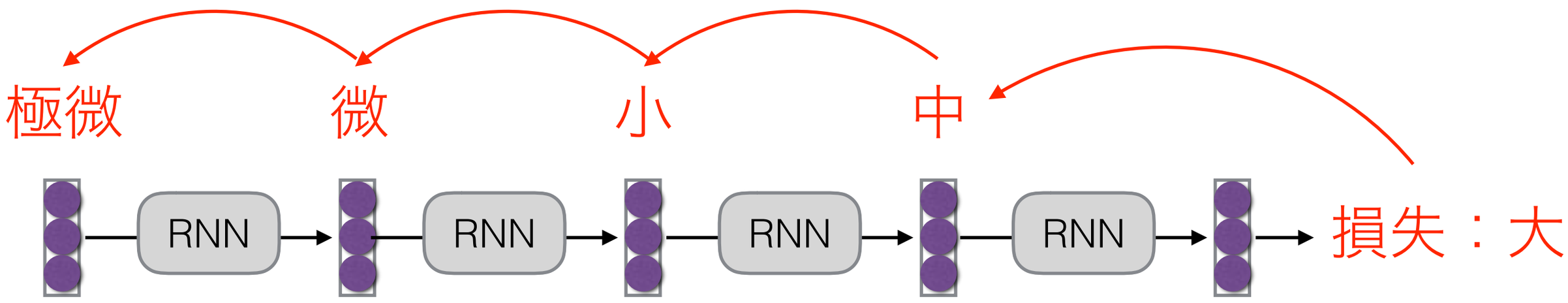

I
hate
this
movie
RNN
RNN
RNN
RNN
RNN
RNN
RNN
RNN
concat
W
+
bias
=
scores

RNNによる系列分類の例

- 13-sentiment-rnn.py

勾配消失の問題

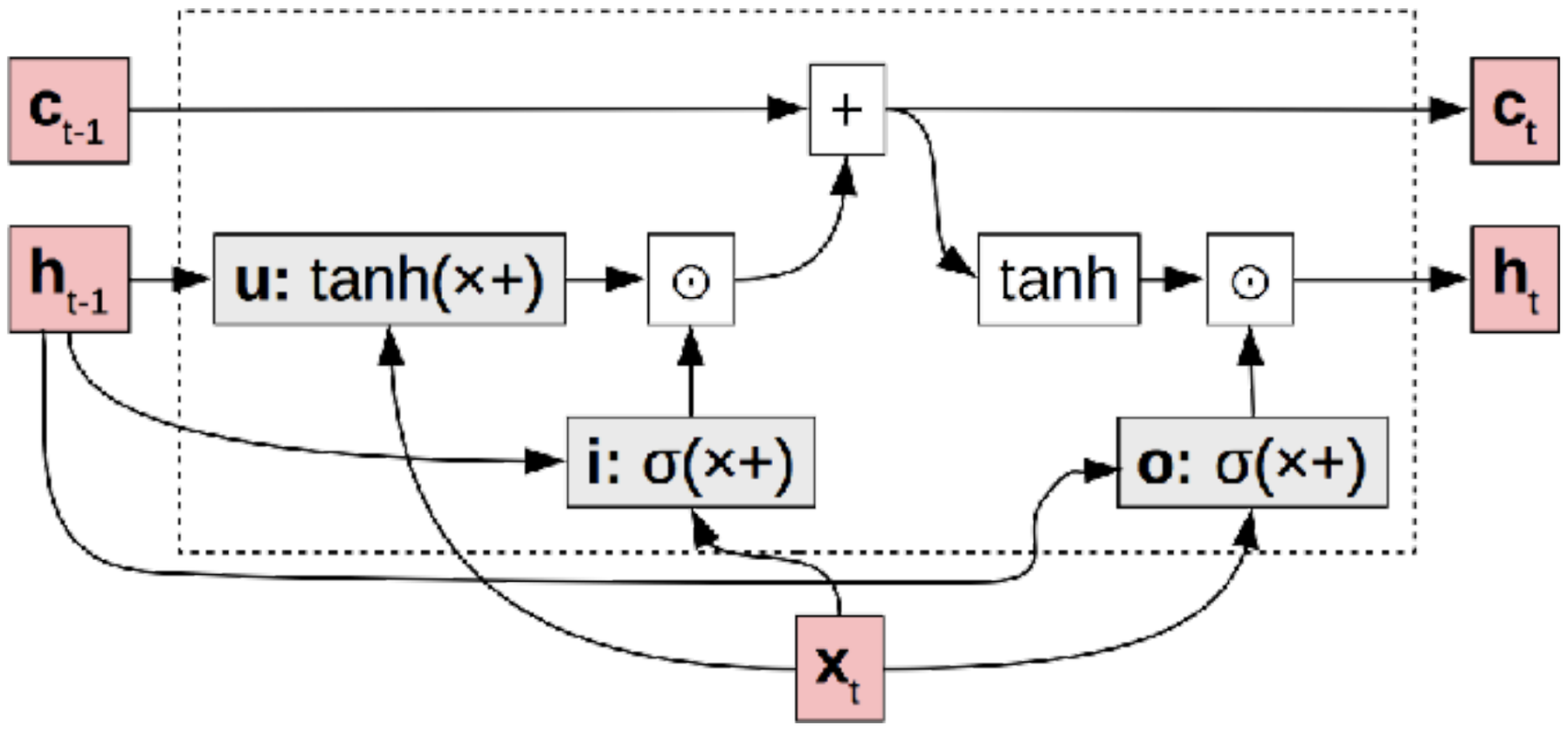
- 文の最後に損失が出る場合



- なぜ起こるのか？活性関数の微分が１以下なため
  （１以上の場合は逆に勾配が爆発）

RNNの他の選択肢

- Long short-term memory (LSTMs; Hochreiter and Schmidthuber, 1997)



- Gated recurrent units (GRUs; Cho et al., 2014)

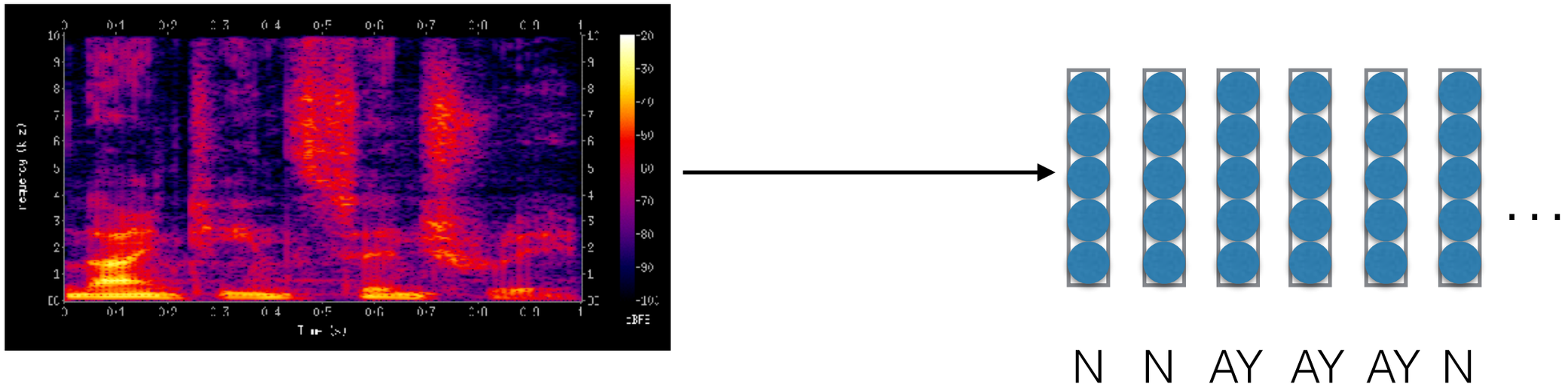
LSTMによる系列分類の例

- 15-sentiment-lstm.py

タグ推定の問題

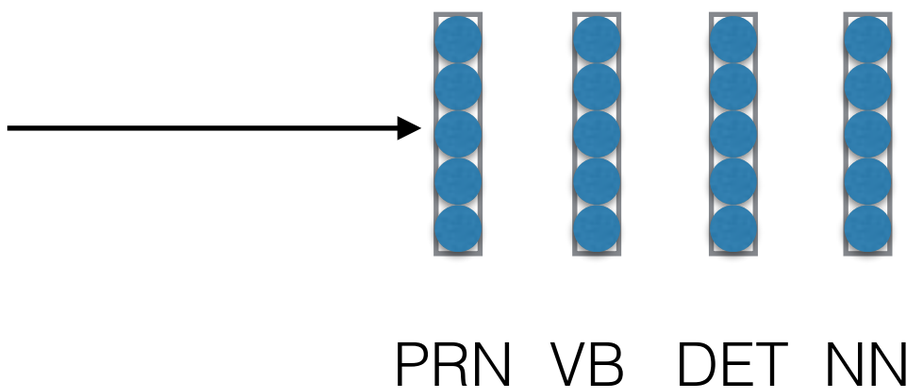
タグ推定の応用

- 音素認識

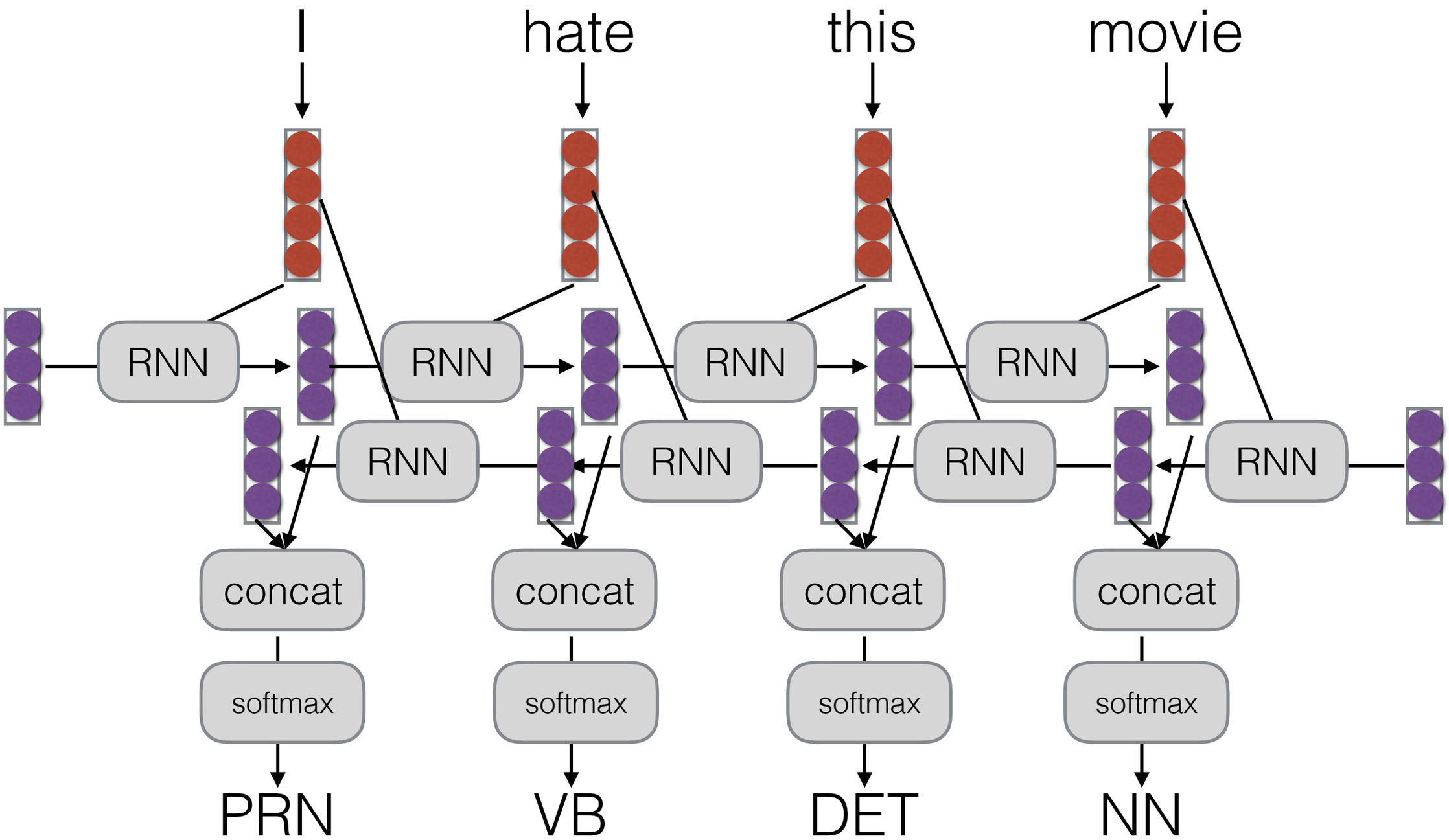


- 品詞推定

this is a pen



両方向RNNによるタグ推定



タグ推定の例

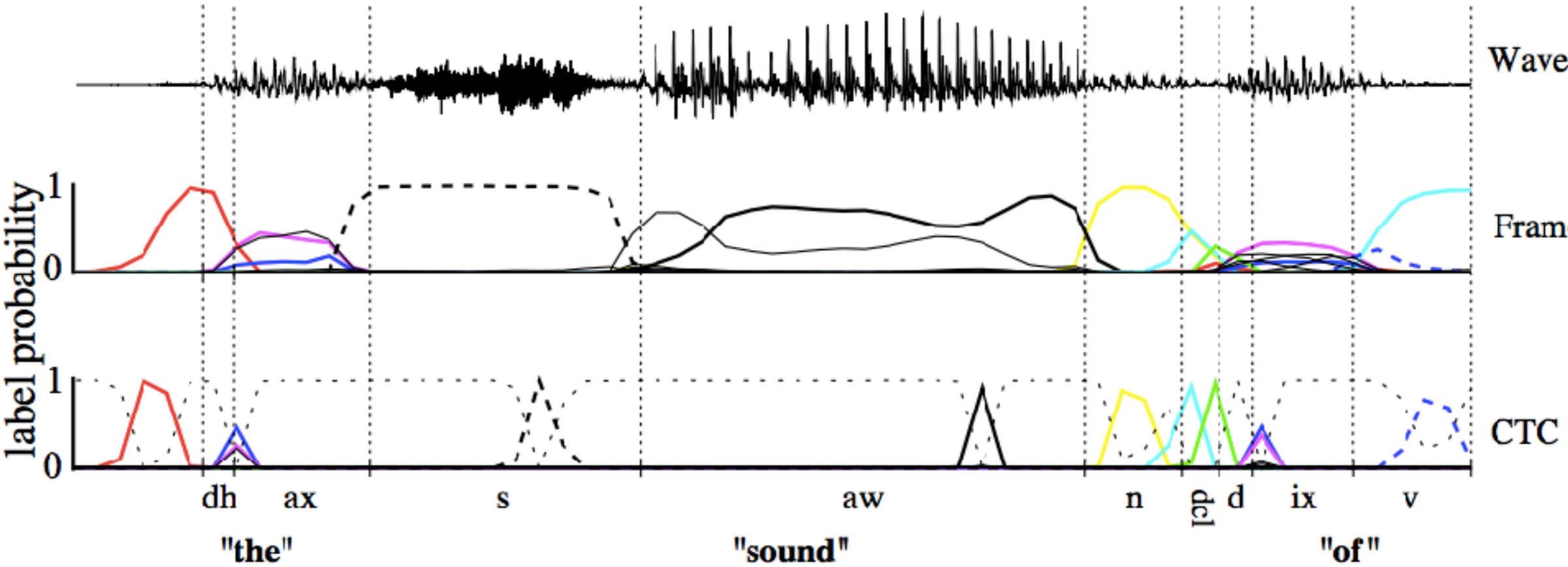
- 20-tagging-bilstm.py

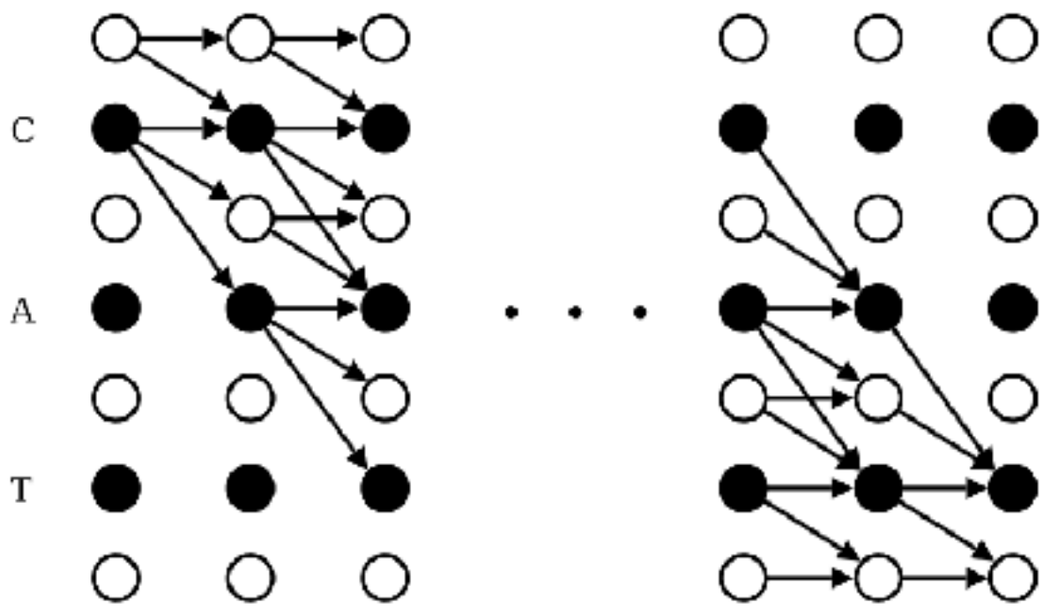
CTC
(Connectionist Temporal Classification)

- 入力とタグの対応が自明ではない時がある
    - 音響モデルで音素列が分かるがアライメントが分からない

- CTCは動的計画法で、どのフレームがどの音素に対応するかを推定しながら学習

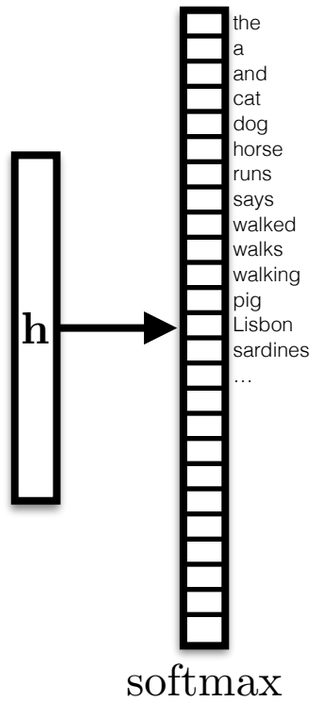
CTCの概念図 (Graves et al. 2006)





次の単語の予測
(言語モデル)

例：言語モデル



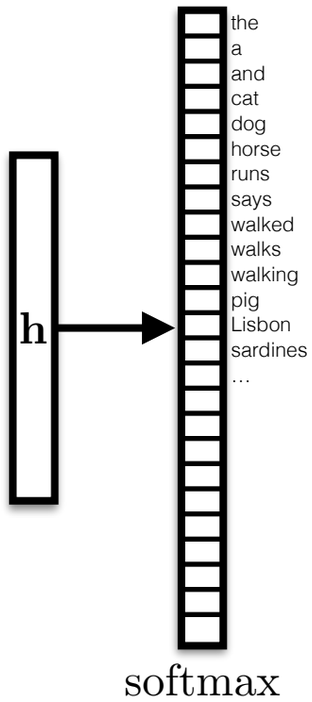
$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$$\mathbf{h} \in \mathbb{R}^d$$

$$|V| = 100,000$$

例：言語モデル



$$\mathbf{u} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

$$p_i = \frac{\exp u_i}{\sum_j \exp u_j}$$

$$\mathbf{h} \in \mathbb{R}^d$$

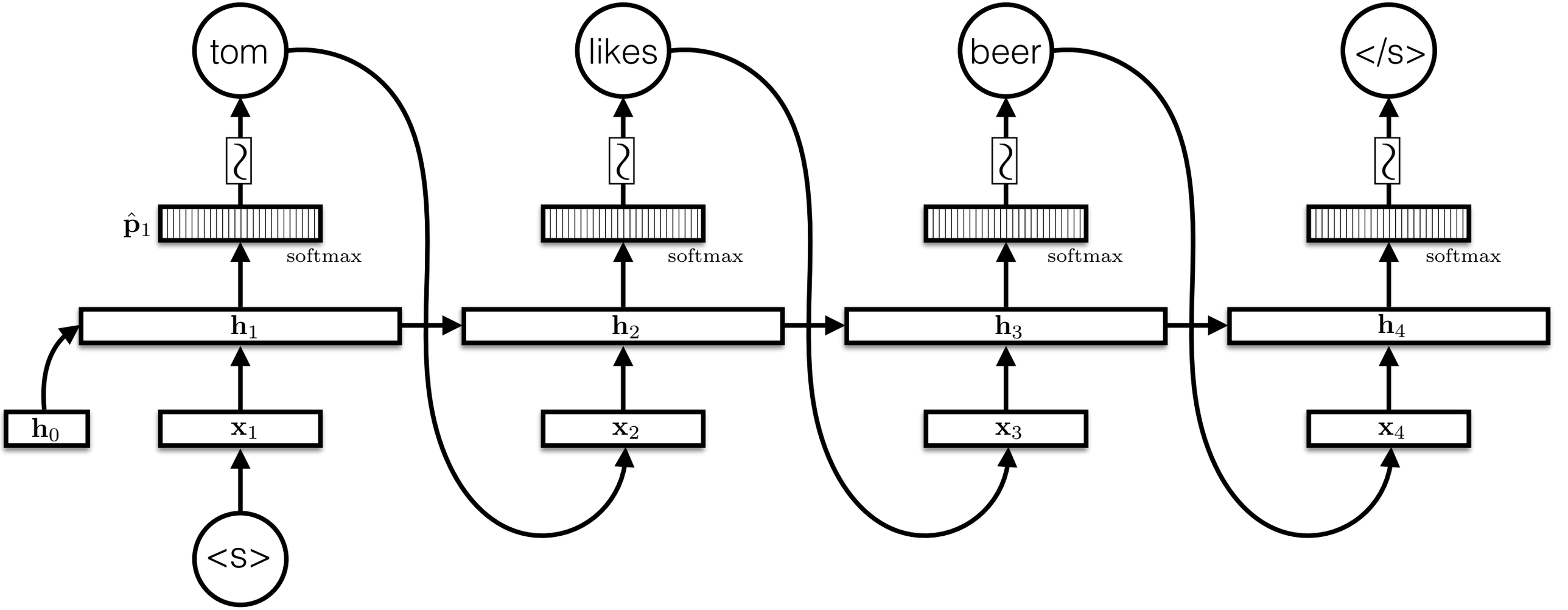
$$|V| = 100,000$$

$$p(\boldsymbol{e}) = p(e_1) \times$$
$$p(e_2 \mid e_1) \times$$
$$p(e_3 \mid e_1, e_2) \times$$
$$p(e_4 \mid e_1, e_2, e_3) \times$$
$$\dots$$

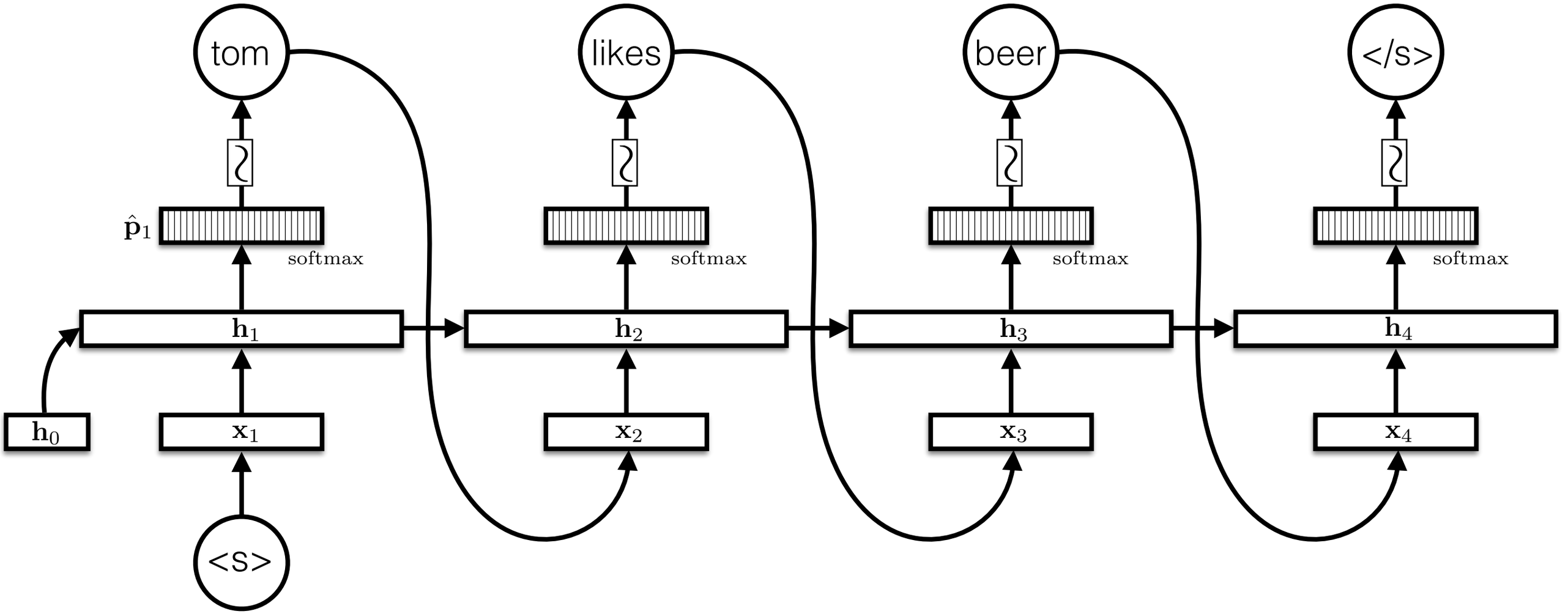
$\mathbf{h}$ 今までに生成した単語で条件付け

例：言語モデル

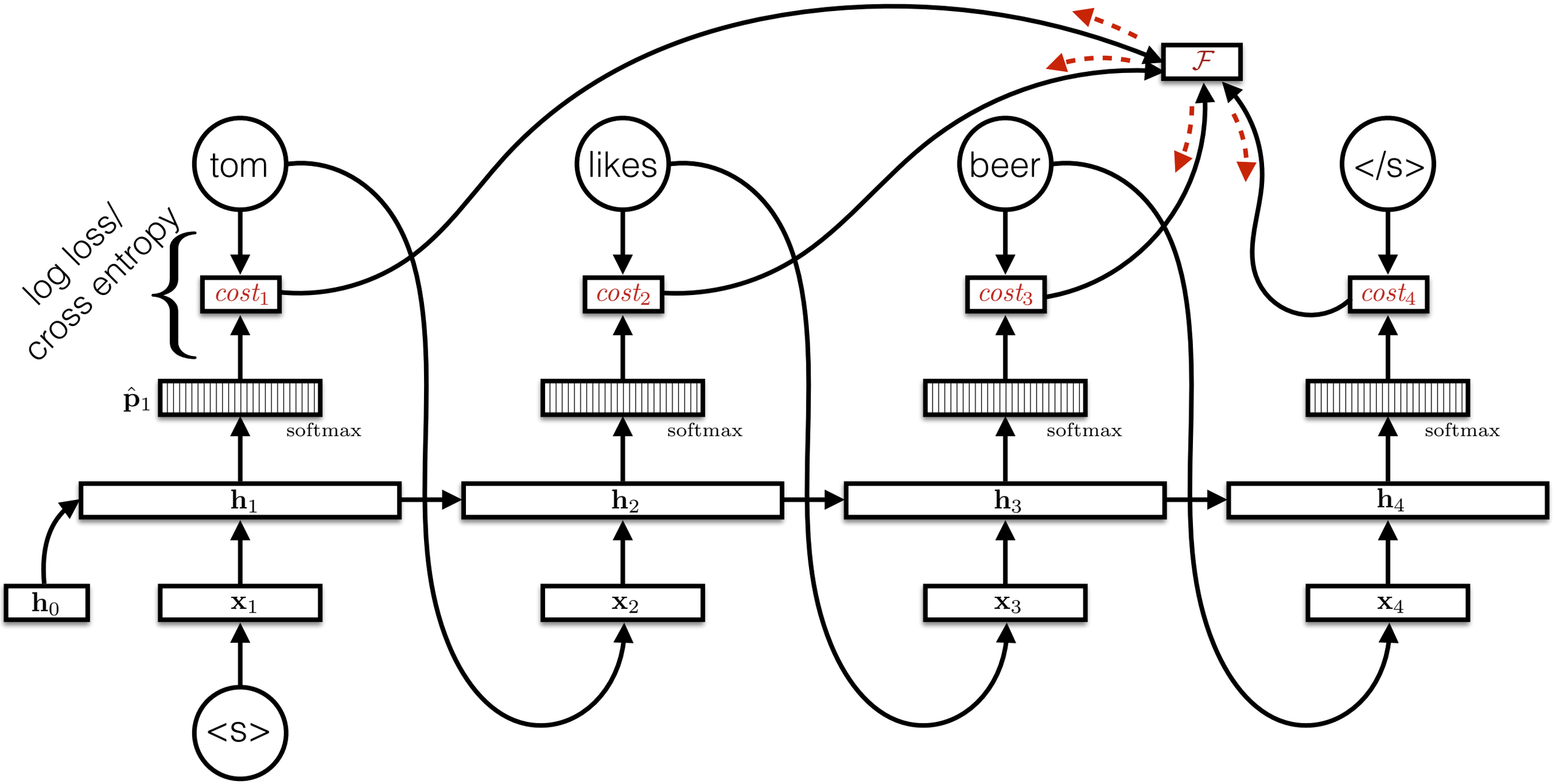
$$p(tom \mid \langle \mathtt{s} \rangle) \times p(likes \mid \langle \mathtt{s} \rangle, tom) \times p(beer \mid \langle \mathtt{s} \rangle, tom, likes) \times p(\langle /\mathtt{s} \rangle \mid \langle \mathtt{s} \rangle, tom, likes, beer)$$



言語モデルの学習


tom
likes
beer
</s>
$\hat{\mathbf{p}}_1$
softmax
softmax
softmax
softmax
$\mathbf{h}_0$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
<s>

言語モデルの学習


tom
likes
beer
</s>
<S>
log loss/
cross entropy
$\mathcal{F}$
$cost_1$
$cost_2$
$cost_3$
$cost_4$
$\hat{\mathbf{p}}_1$
softmax
softmax
softmax
softmax
$\mathbf{h}_0$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$

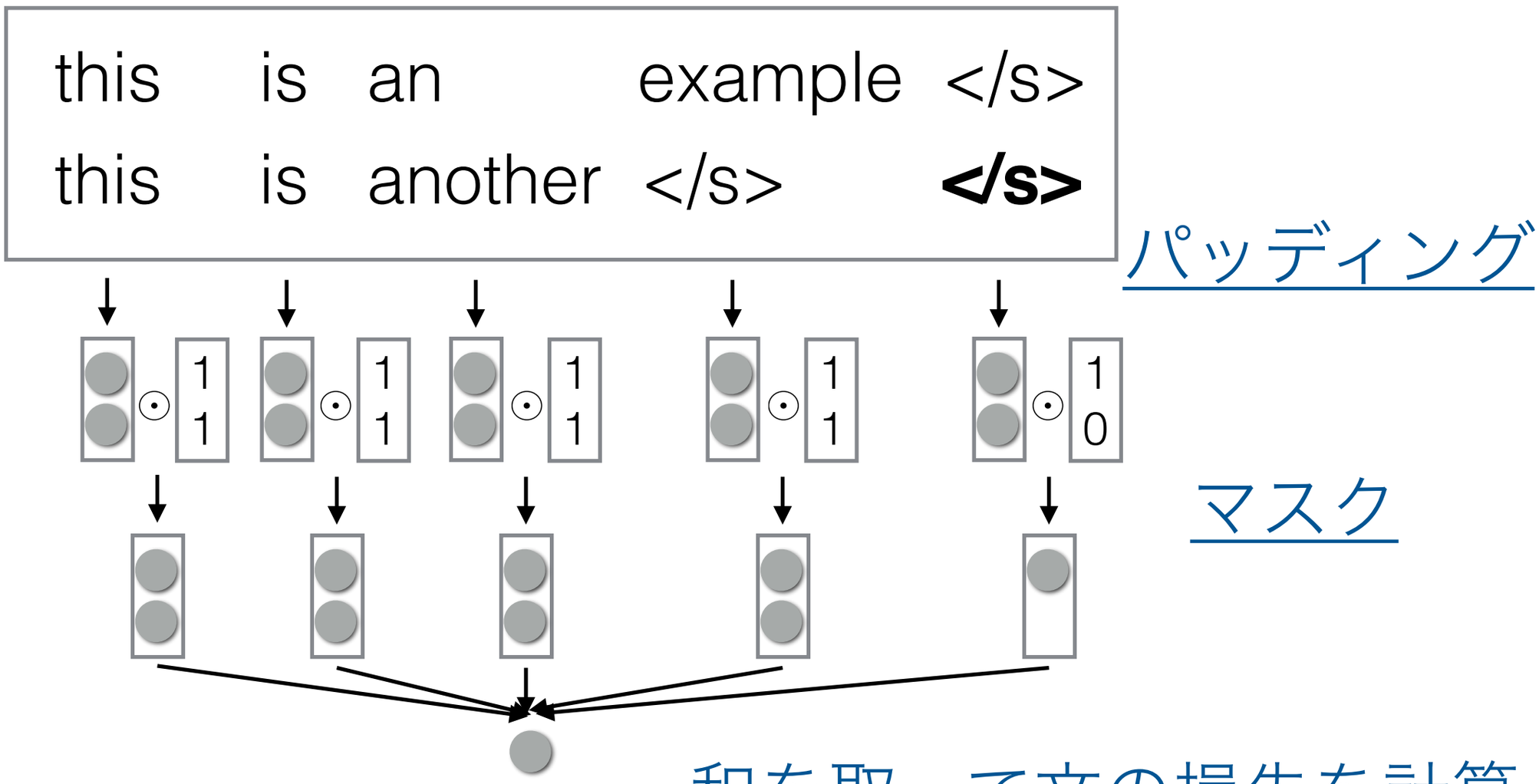
言語モデルの例

- 30-lm-lstm.py

系列モデルの高速化

系列上のミニバッチ化

- 複数の長さの系列をどう扱うか？

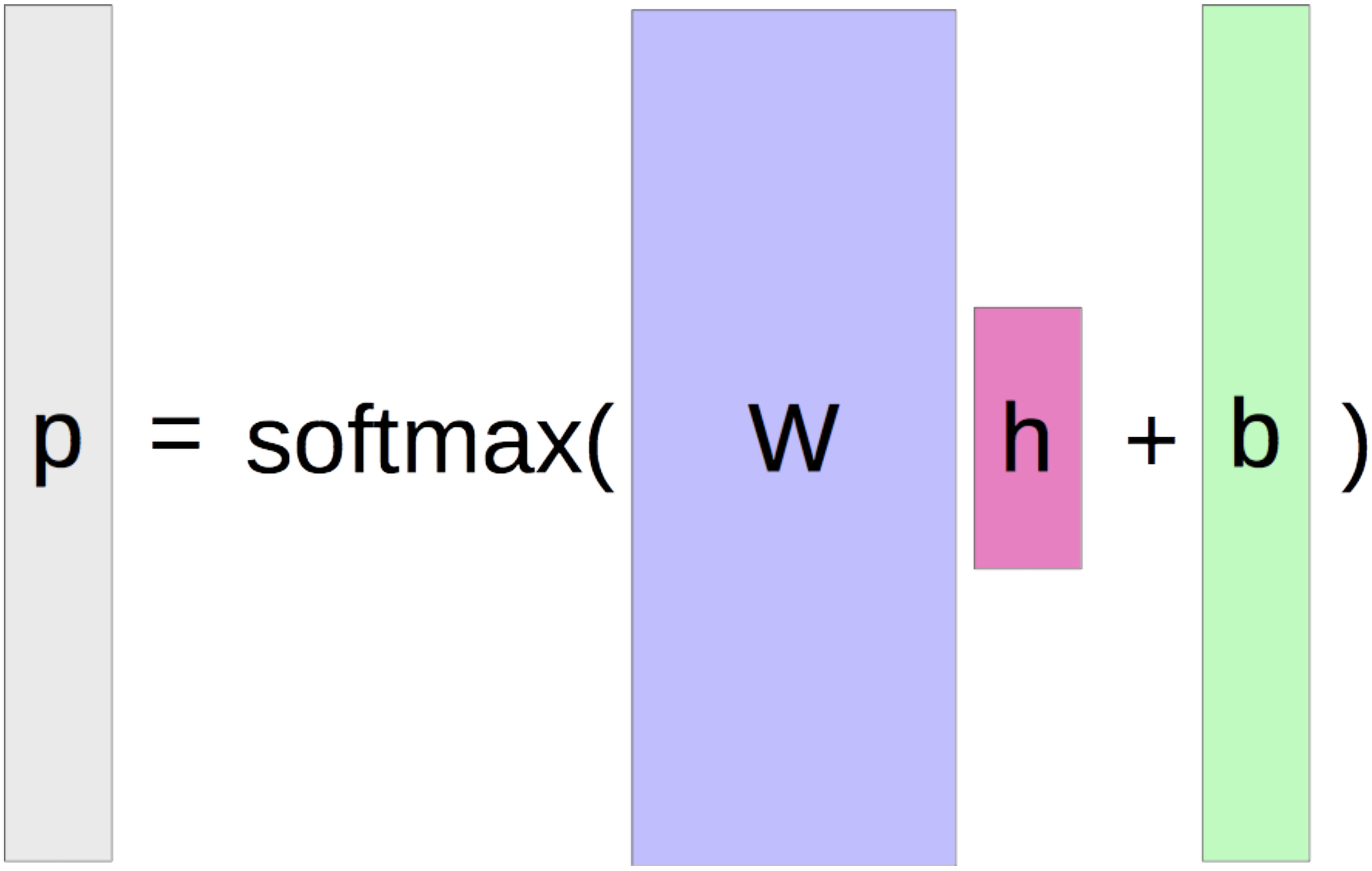


softmax計算の効率化

- 計算量としてもう１つのネックは大規模な語彙に対するsoftmax計算

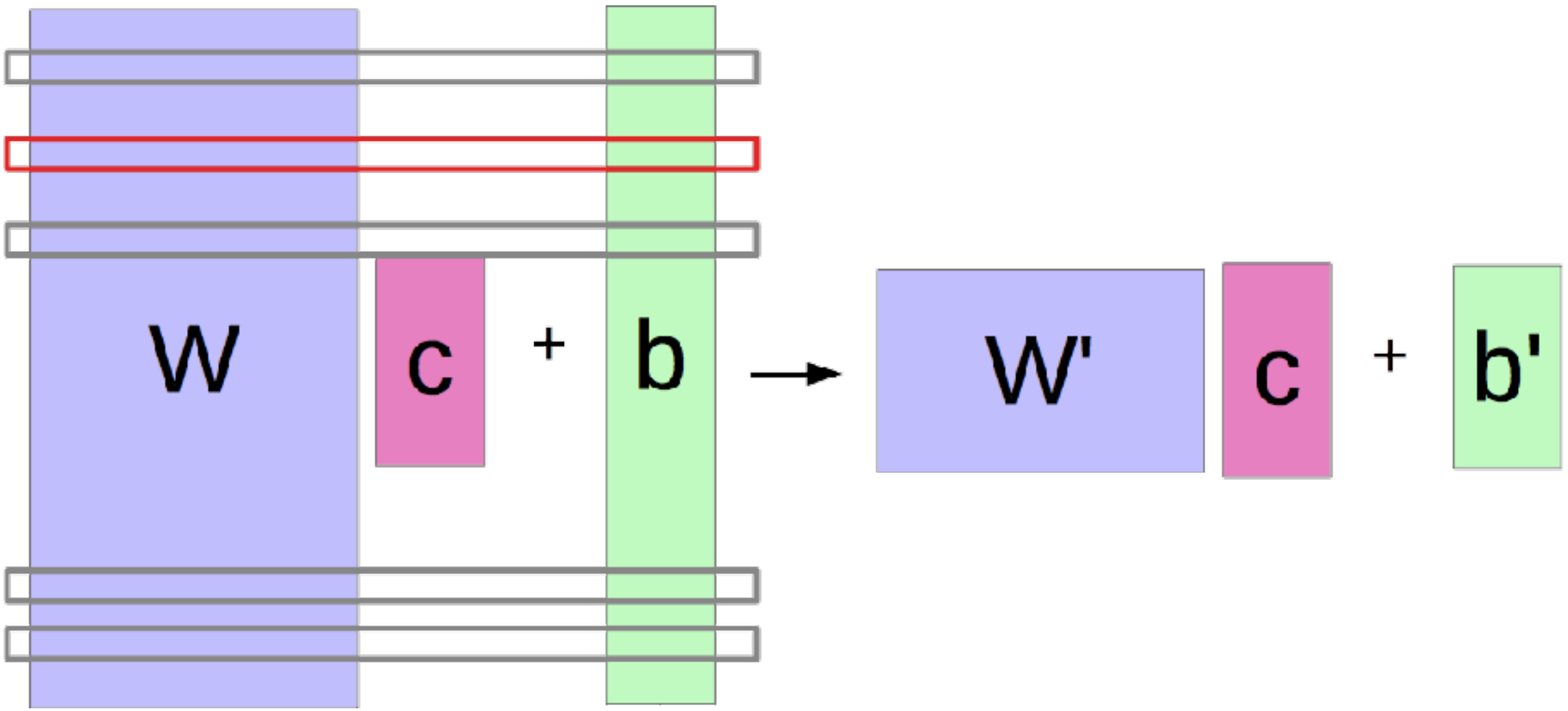
- 様々な効率化方法：階層化、サンプリング、バイナリコード

Softmaxの計算例



サンプリング/Noise Contrastive Estimationによる近似

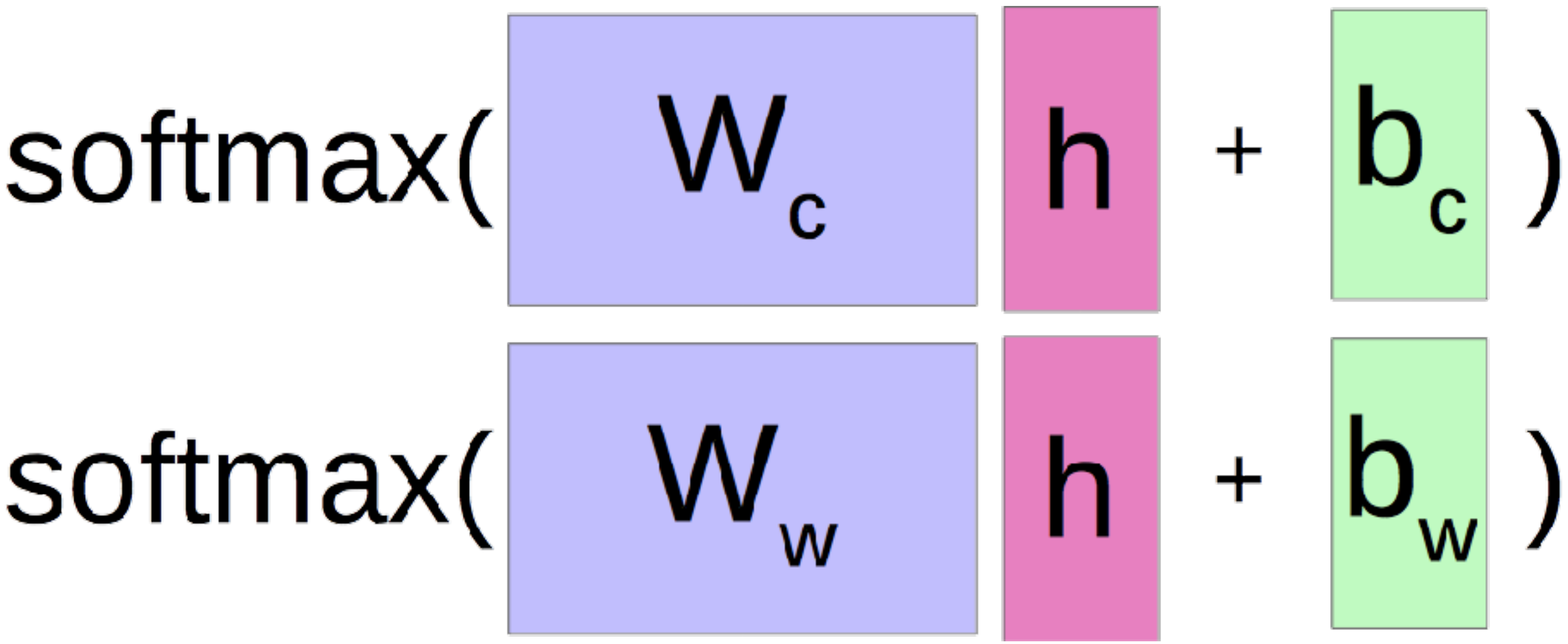
- 語彙の部分集合に対する計算



DyNetでrnnlm-batch-nceという例

クラスに基づくsoftmax

- まず単語のクラスを推定し、そのあと単語を推定

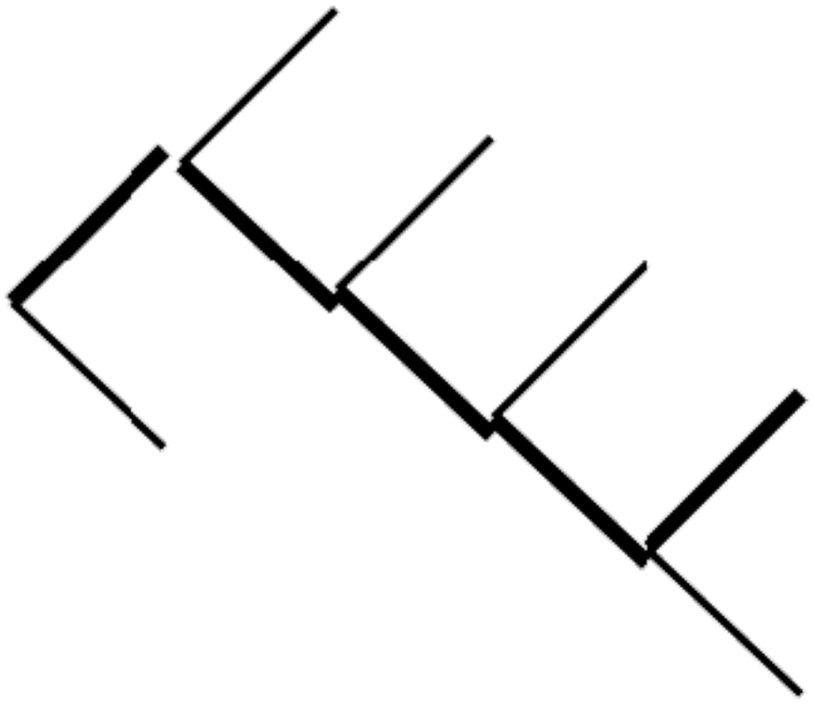


DyNetのClassFactoredSoftmaxBuilder

Goodman 2001. Classes for Fast Maximum Entropy Training

階層的softmax

- 単語を木構造で徐々に推定

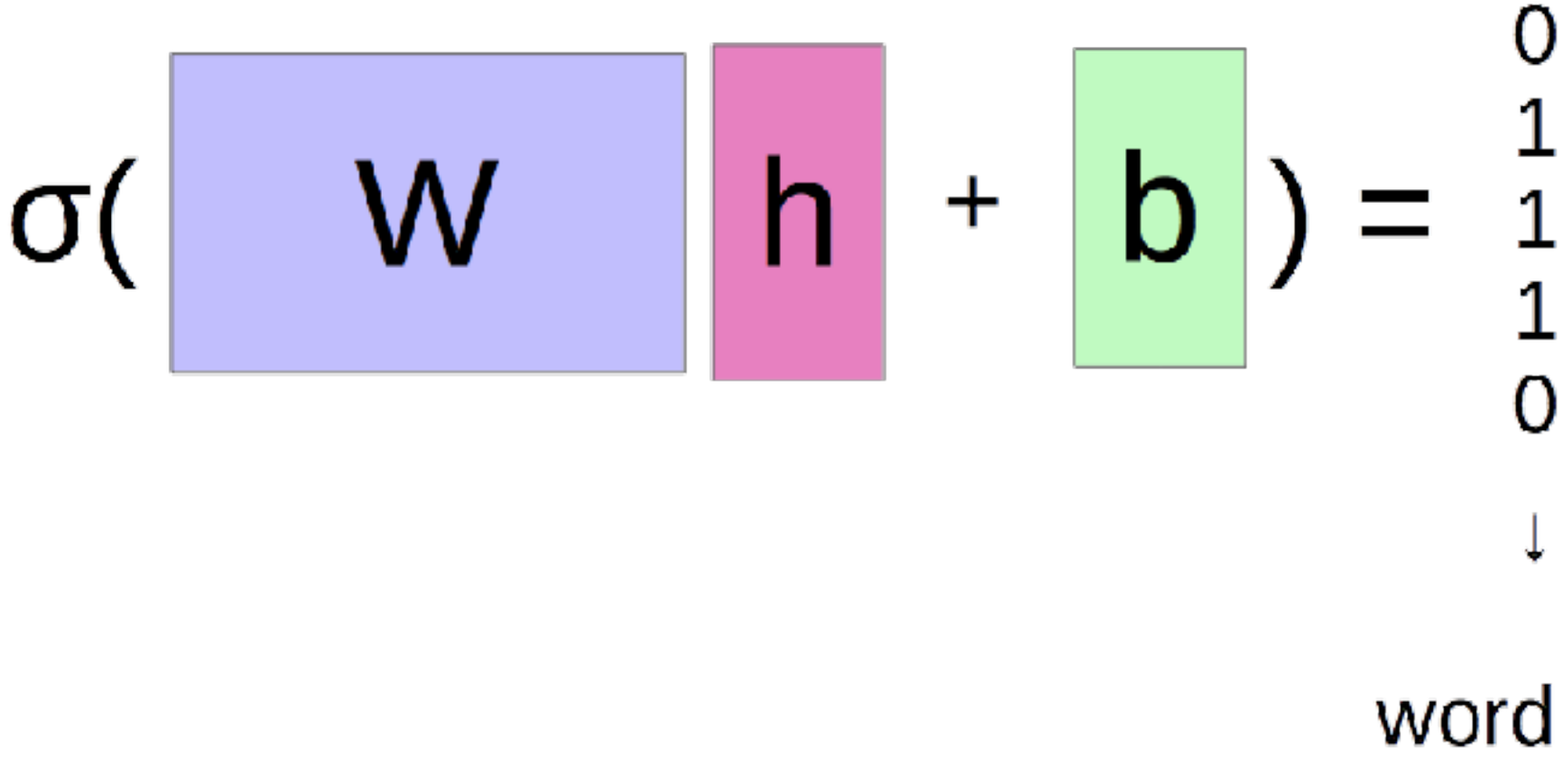


0　1　1　1　0　　→ word 14

Morin and Bengio 2005: Hierarchical Probabilistic NNLM

バイナリコード

- 単語のバイナリを直接推定



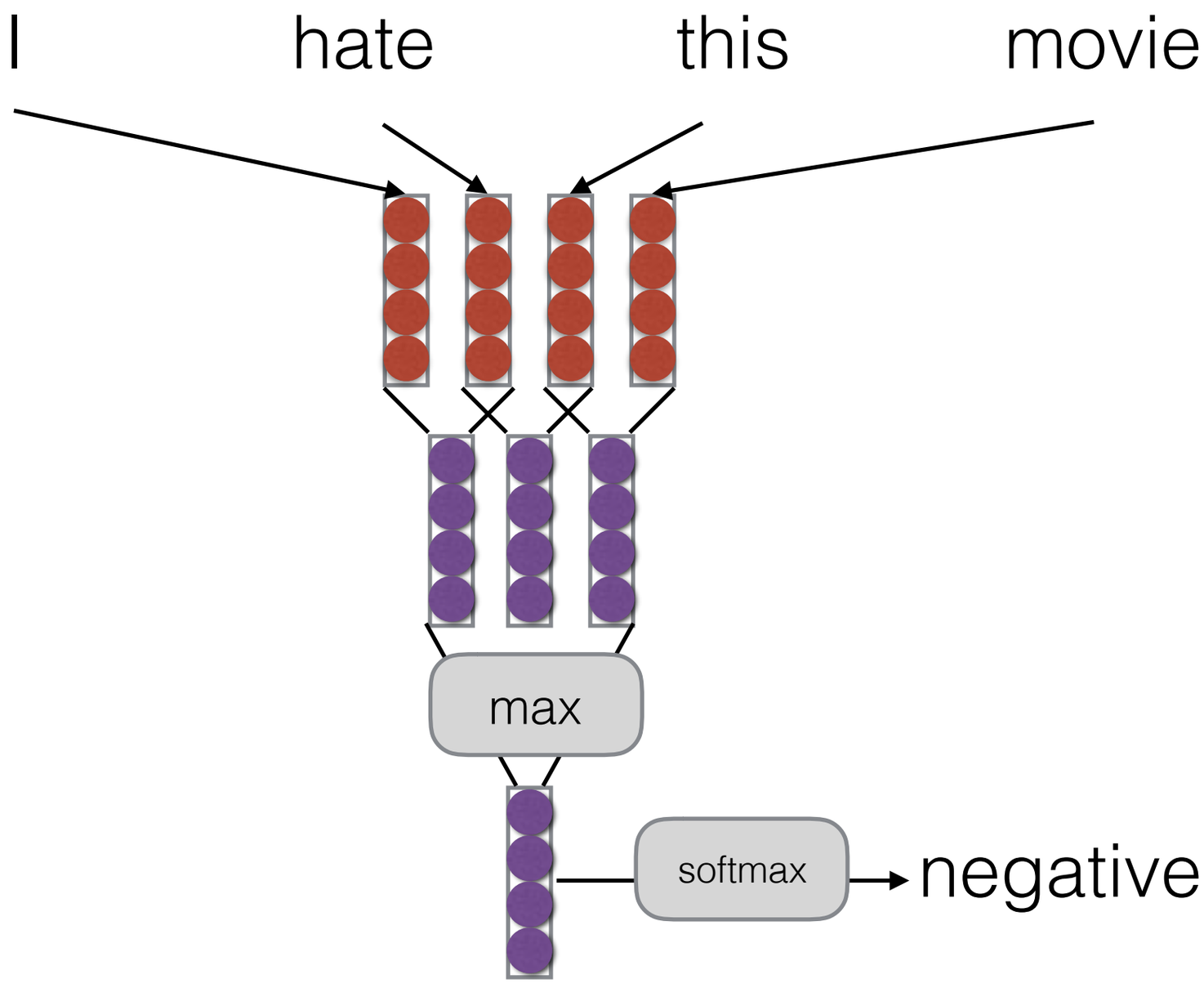
Oda et al. 2017. Neural Machine Translation via Binary Code Prediction.

おまけ：

その他の系列に対するニューラルネット

系列に対するCNN (TDNN)

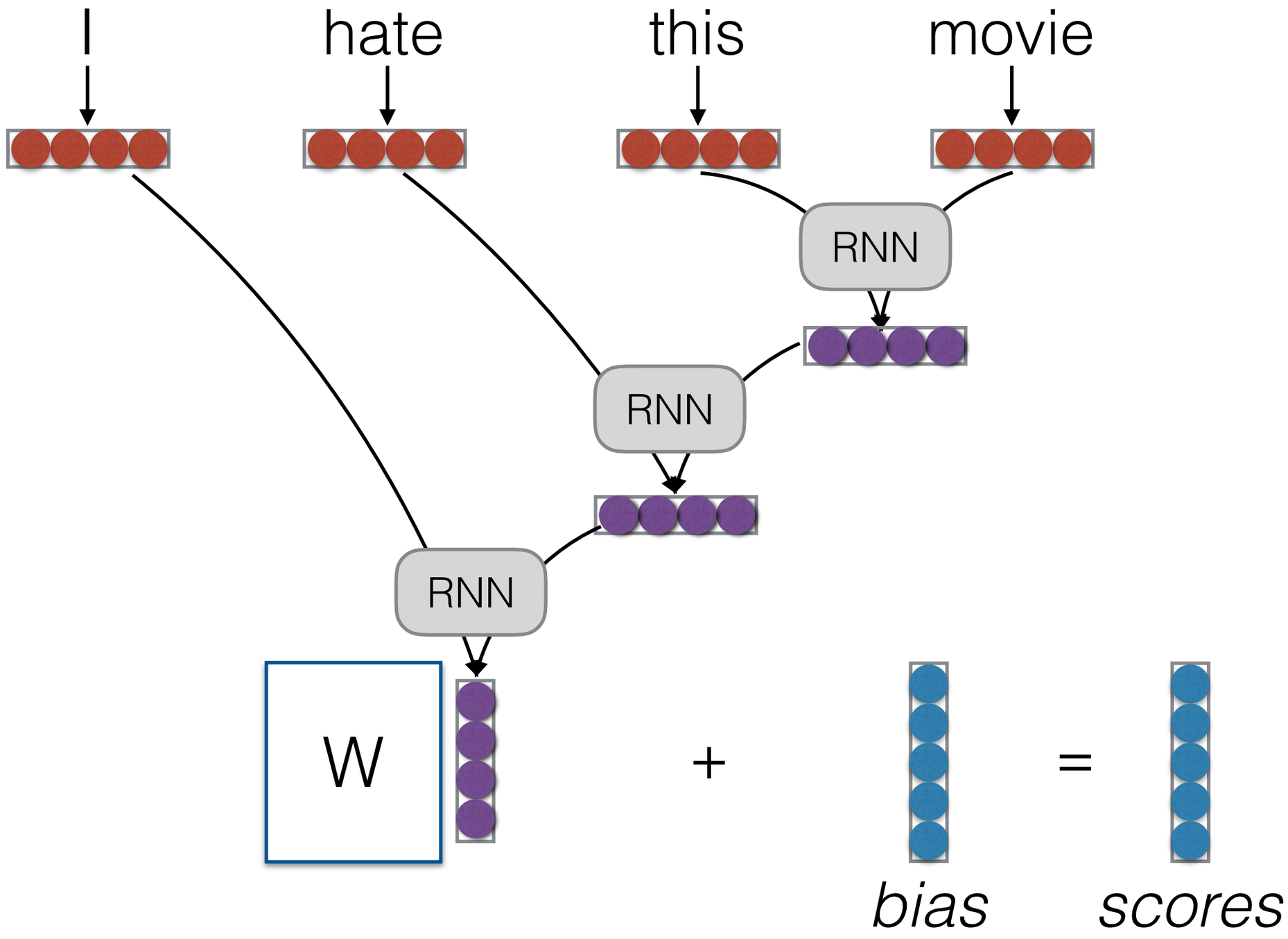
- 長い系列内の短い系列から特徴量を抽出



CNNによる系列モデルの例

- 14-sentiment-cnn.py

Tree-structured RNN/LSTM



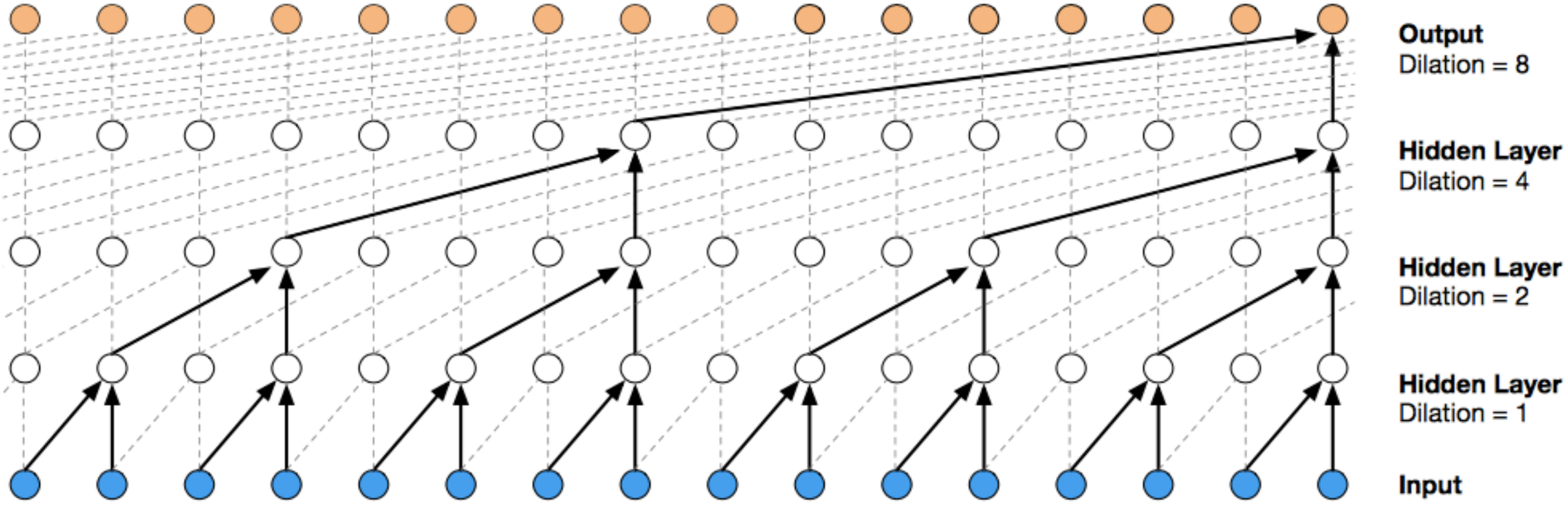
Tree-LSTMの例

- 15-sentiment-treenn.py

Dilated Convolution

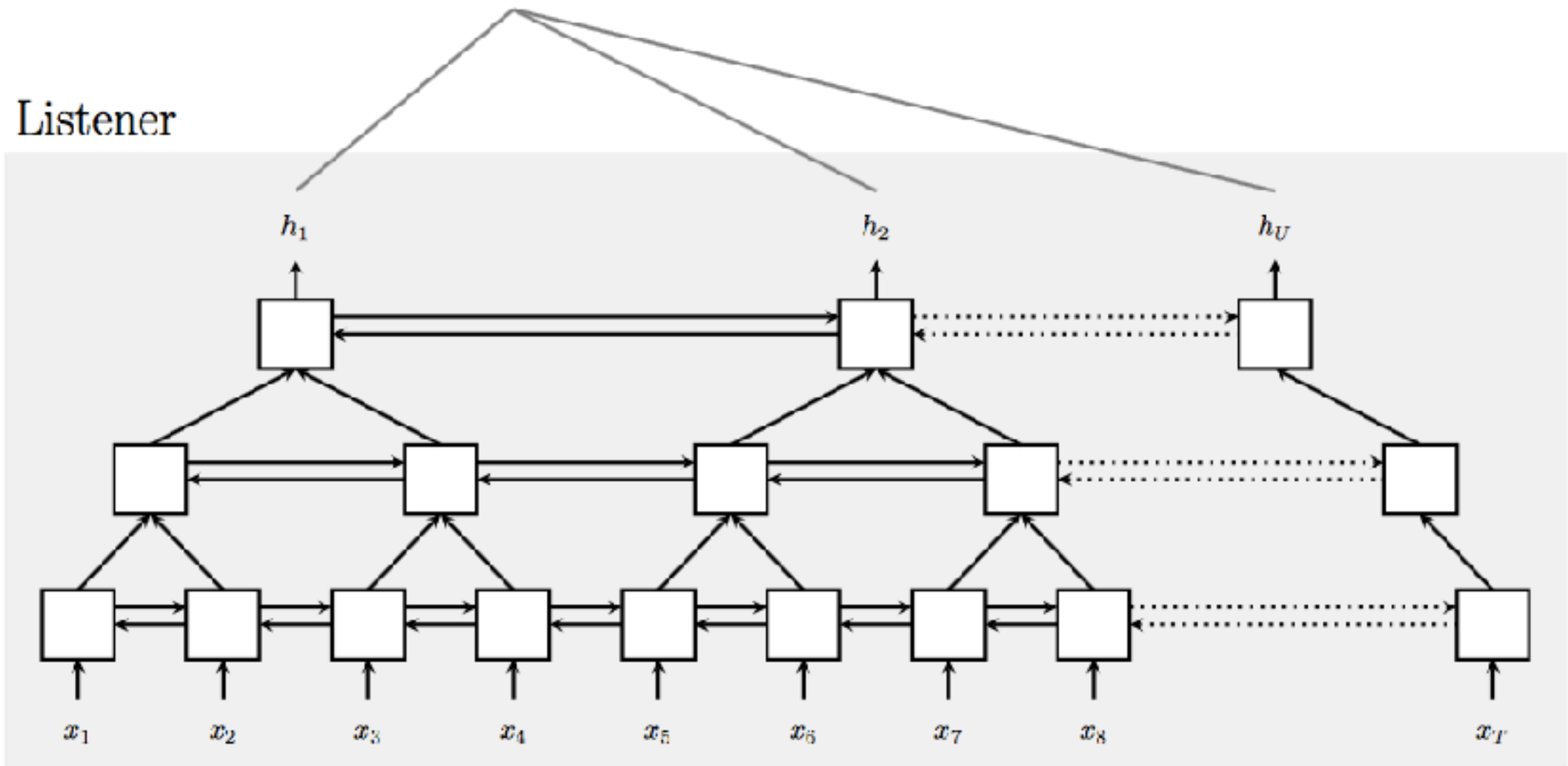
- 徐々に粒度を荒くしていくネット

- 例：WaveNet (van den Oord 2016)



Pyramidal RNN

- 同じく粒度を徐々に荒く
- 例：Listen, Attend, and Spell (Chan et al. 2015)



系列変換モデル

系列変換モデル

機械翻訳

he ate an apple → 彼はりんごを食べた

タグ推定

he ate an apple → PRN VBD DET NN

対話

彼はりんごを食べた → 良かった！最近ラーメンばっかり食べてて心配になってた。

音声認識

 → 彼はりんごを食べた

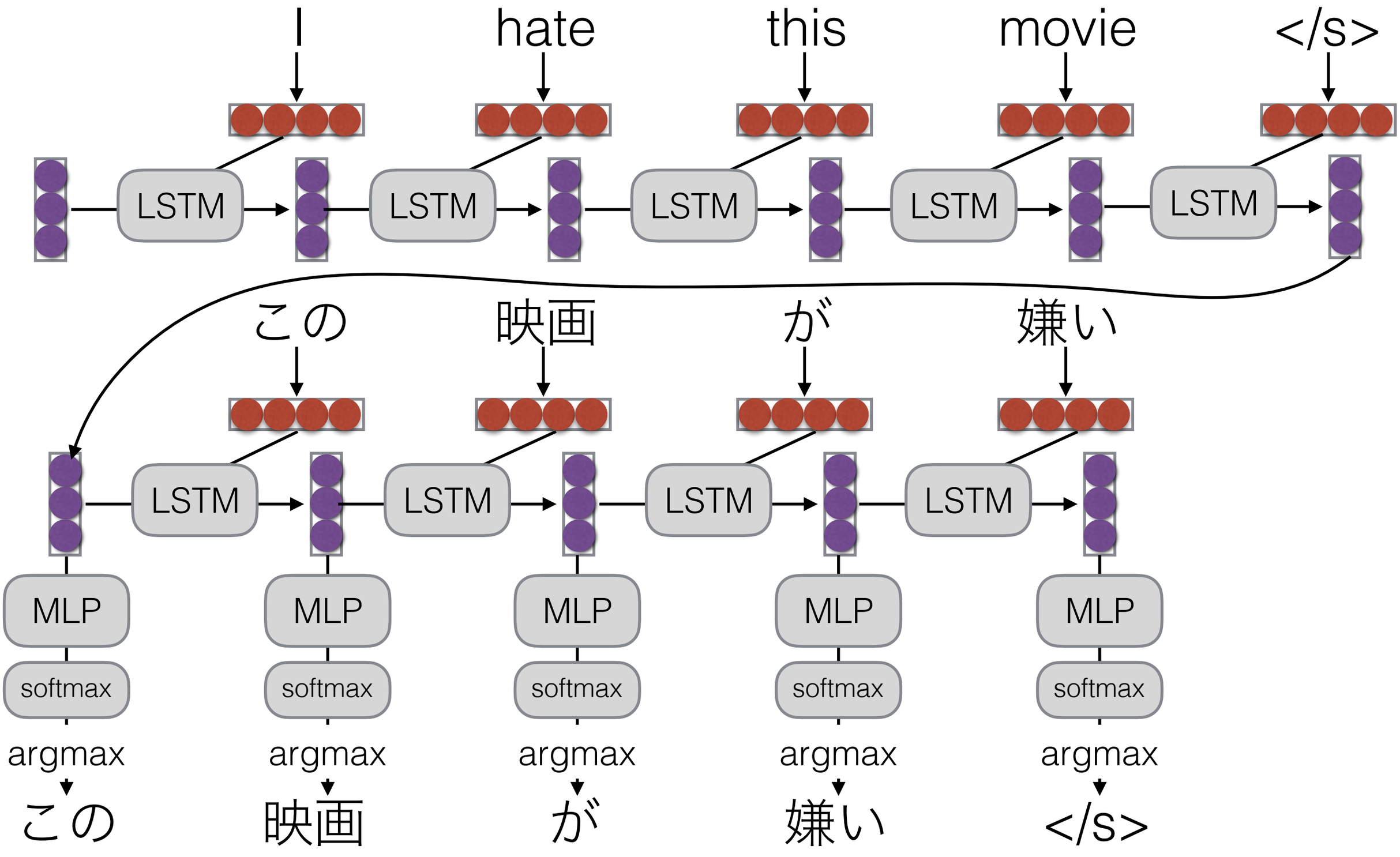
その他

010001110010 → 110110010111110

Encoder-Decoder

- Encoder：入力Fを読み込みベクトルを生成
- Decoder：出力Eの次の単語の確率を推定
- 出力の友達

Encoder-Decoder



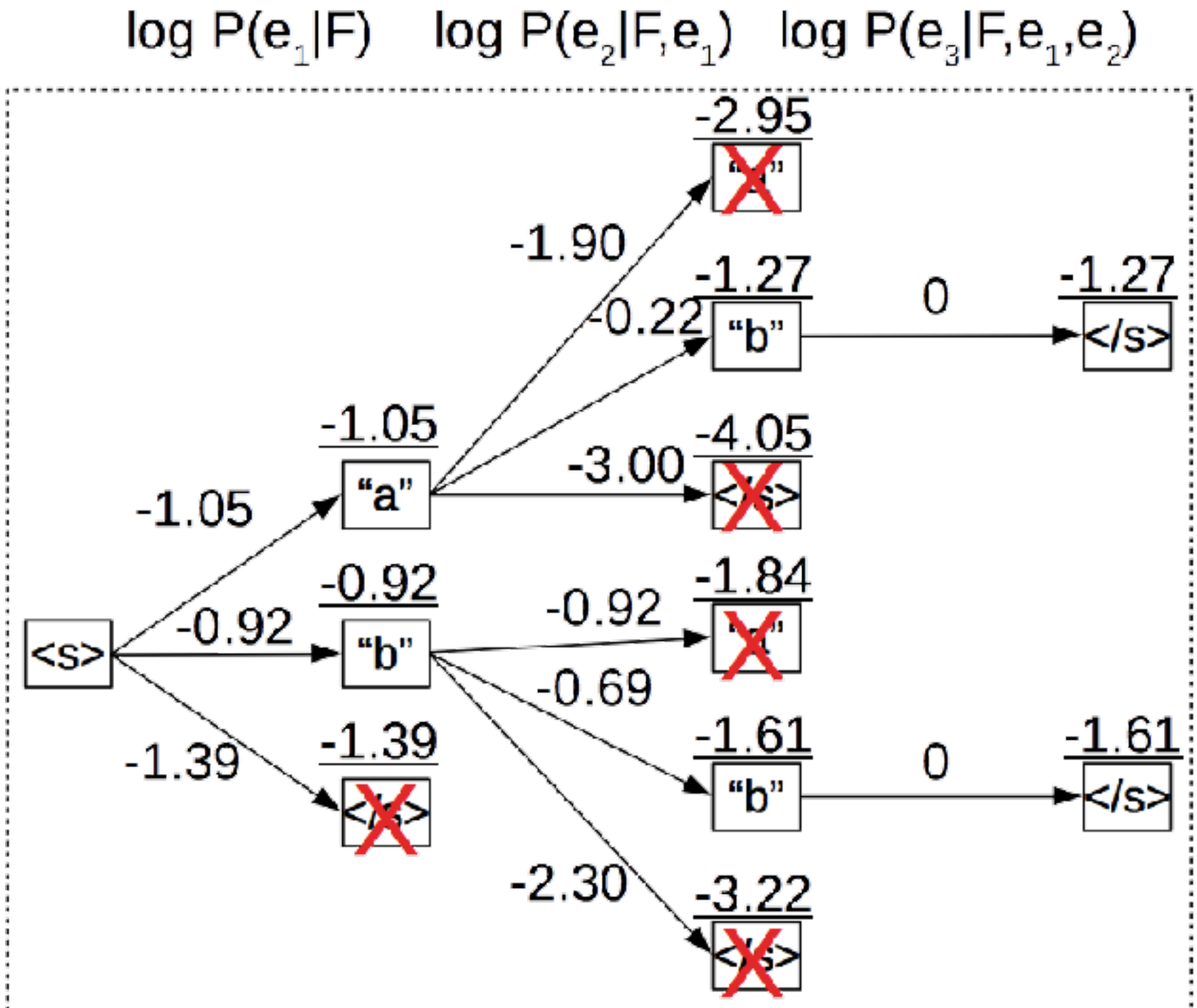
Encoder-Decoderでの生成（貪欲法）

- Encoderでベクトルを計算
- 文末記号</s>が生成されるまで
  - 次の単語の確率を計算
  - 最も確率の高い単語を生成

Encoder-Decoderの例

- 40-translation-encdec.py

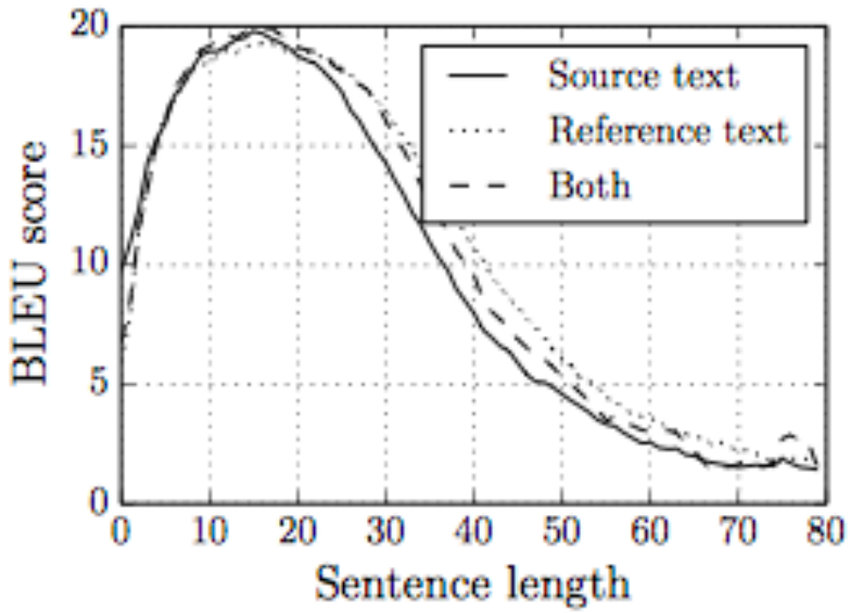
Encoder-Decoderでの生成（ビーム探索）



Attention

Encoder-Decoderの問題

- 最後の隠れ状態で入力文のすべての情報を覚えていないといけない
  - ネットが小さい：
    精度が下がる（特に長い文）

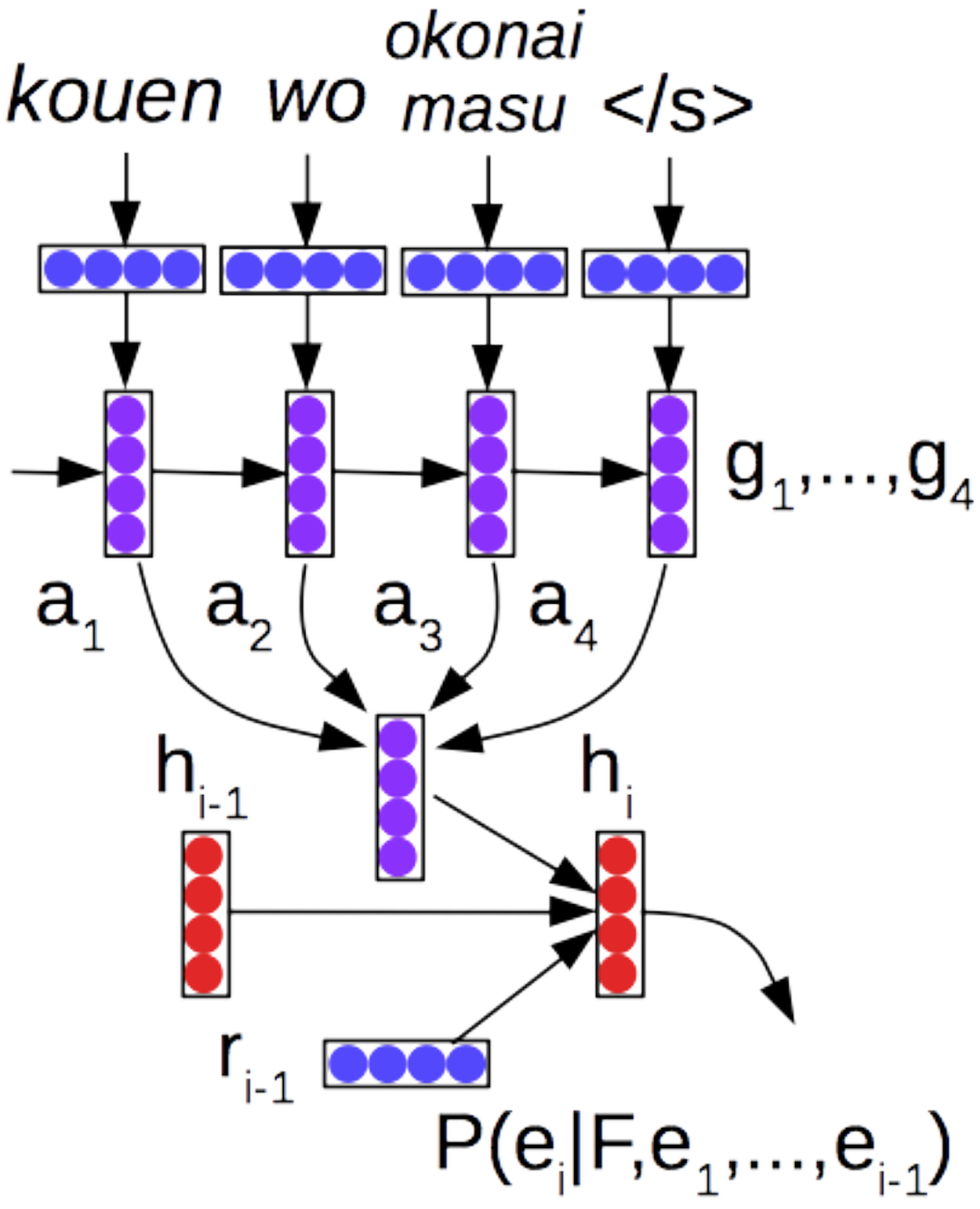


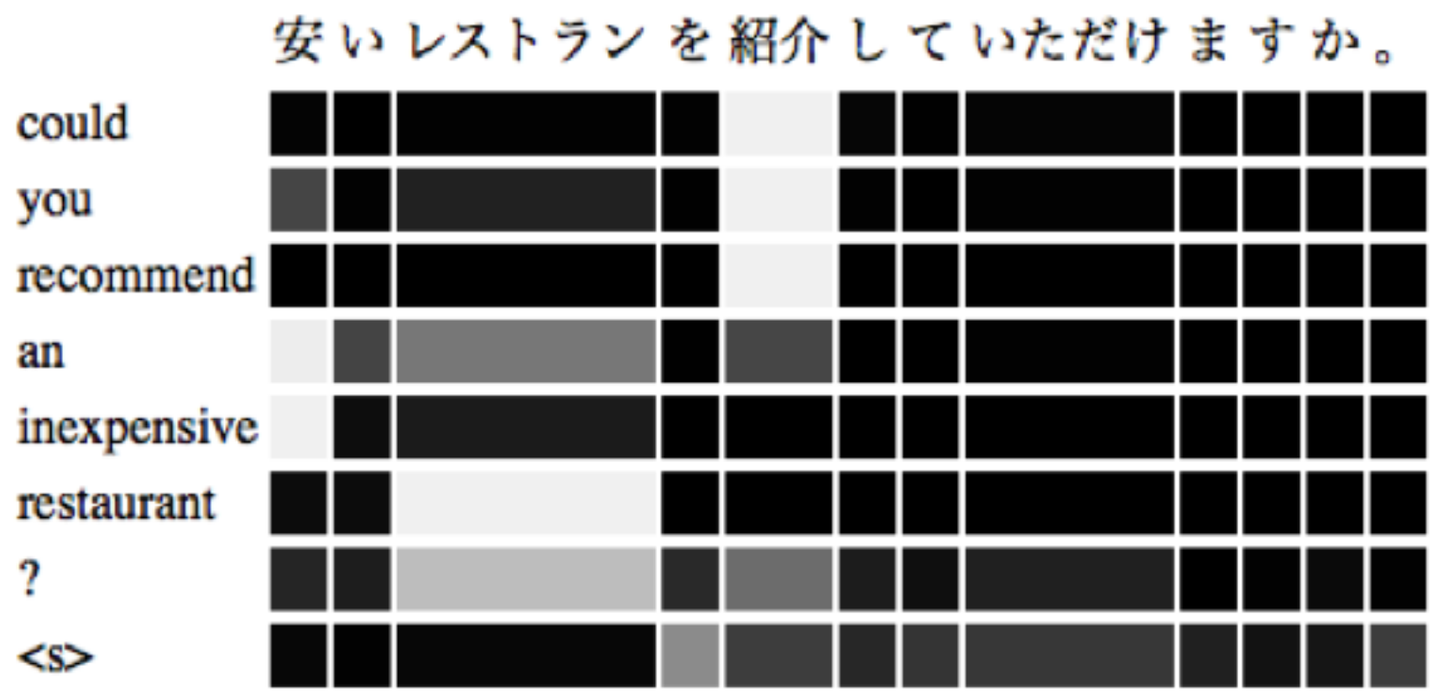
(a) RNNenc without segmentation

- ネットが大きい：余計な計算量、学習が大変

Attention (注意機構)

- **基本的なアイデア：**次の単語を生成するときに、入力側のどの単語に注目するかを決定





Attentionの計算

- 入力側のすべてのベクトルg_i、出力側の現在のベクトルf_jに基づき
- それぞれに対してスコアを計算

  $s_i = \mathrm{score}(g_i, f_j)$

- スコアを足して1になるようにsoftmaxをかける

  $\boldsymbol{a} = \mathrm{softmax}(\boldsymbol{s})$

Attentionの例

- 41-translation-attention.py

系列変換モデルの
課題・解決法

未知語の扱い

- モデルの語彙に含まれない語彙は生成不可
  - そもそもデータに存在しない単語
  - 計算量のために語彙から省いた単語

- 解決策：珍しい単語を部分文字列に分ける

Thank you . I'm just looking .

▁Thank ▁you ▁. ▁I ' m ▁just ▁look ing ▁.

おすすめソフトウェア：

sentencepiece (https://github.com/google/sentencepiece/)

単語の繰り返し・訳し抜け

- 単語が繰り返されるか、訳し抜けが多い

入力: どのファンデーションが私の肌の色に近いですか。
正解: which foundation comes close to my natural skin color ?
EncDec: which foundation is my favorite foundation with a foundation ?

解決策：どの単語が既に訳されたかを記録するモデルの利用

Mi et al. 2015. Coverage Embeddings for Neural Machine Translation.

近い単語への置き換え

- 意味的に近い単語の単語ベクトルは近い→
  生成で誤って生成する可能性が高い

Input: I come from Tunisia.
Reference: チュニジア の 出身です。
Chunisia no shusshindesu.
(I'm from Tunisia.)
System: ノルウェー の 出身です。
Noruue- no shusshindesu.
(I'm from Norway.)

- 離散的な翻訳辞書の導入

  Arthur et al. Incorporating Discrete Translation Lexicons into Neural Machine Translation.

高頻度語・高頻度文重視

- 対話などの応用では、
  「よくある応答」ばかりを
  生成しがち

Input: What are you doing?			
−0.86	I don't know.	−1.09	Get out of here.
−1.03	I don't know!	−1.09	I'm going home.
−1.06	Nothing.	−1.09	Oh my god!
−1.09	Get out of the way.	−1.10	I'm talking to you.
Input: what is your name?			
−0.91	I don't know.	...	
−0.92	I don't know!	−1.55	My name is Robert.
−0.92	I don't know, sir.	−1.58	My name is John.
−0.97	Oh, my god!	−1.59	My name's John.
Input: How old are you?			
−0.79	I don't know.	...	
−1.06	I'm fine.	−1.64	Twenty-five.
−1.17	I'm all right.	−1.66	Five.
−1.17	I'm not sure.	−1.71	Eight.

- 解決法：よくある応答にペナルティをかける

  Li et al. 2015. A Diversity-promoting Objective for Neural Conversation Models

まとめ

今回の内容

- 識別（最適化手法、ミニバッチ、ネットの調整、dropout）
- 可変長系列の扱い（BOW、CNN、RNN等）
- 系列変換（encoder-decoder、attention）

今回扱わなかった内容

- 複数のモデルのアンサンブル
- モデルの並列化
- GPU周りの話

終