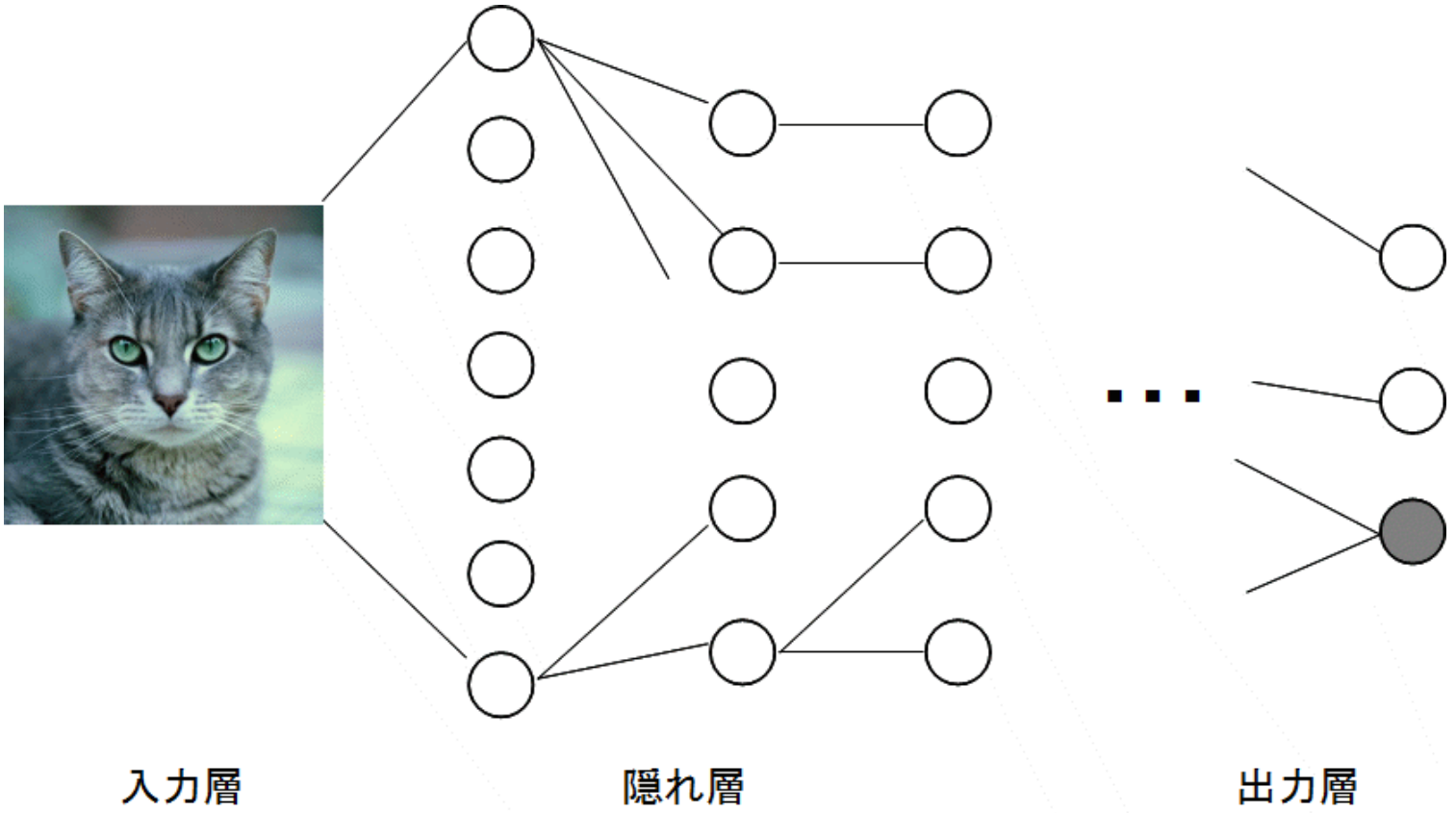
15 章 深層学習

15.1 深層学習とは

- 深層学習の定義のひとつ
  - 表現学習：抽出する特徴も学習する



15.1 深層学習とは

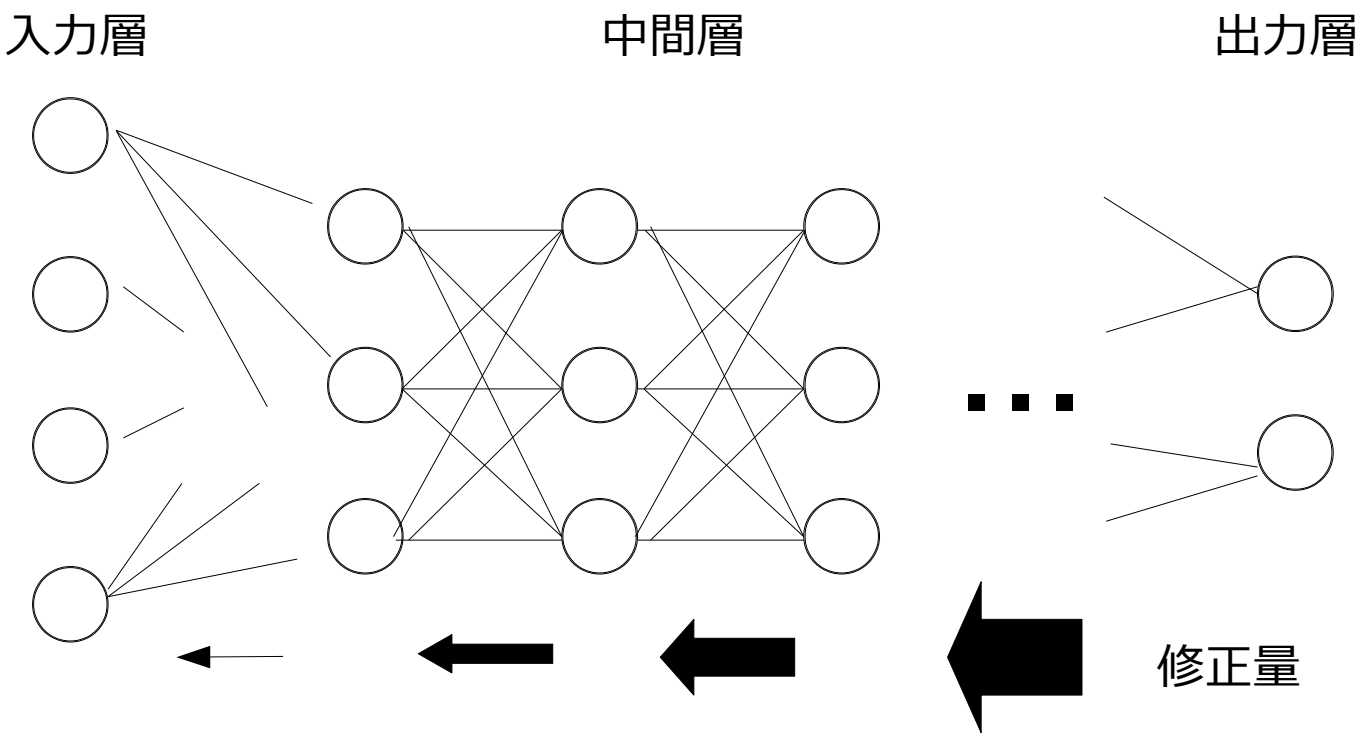
単純なマルチレイヤーパーセプトロンとの違い

- 多階層学習における工夫
  - 事前学習
  - 活性化関数の工夫
  - 過学習の回避：ドロップアウト
- 問題に応じたネットワーク構造の工夫
  - 畳み込みネットワーク
  - リカレントネットワーク

15.2 多階層ニューラルネットワークの学習

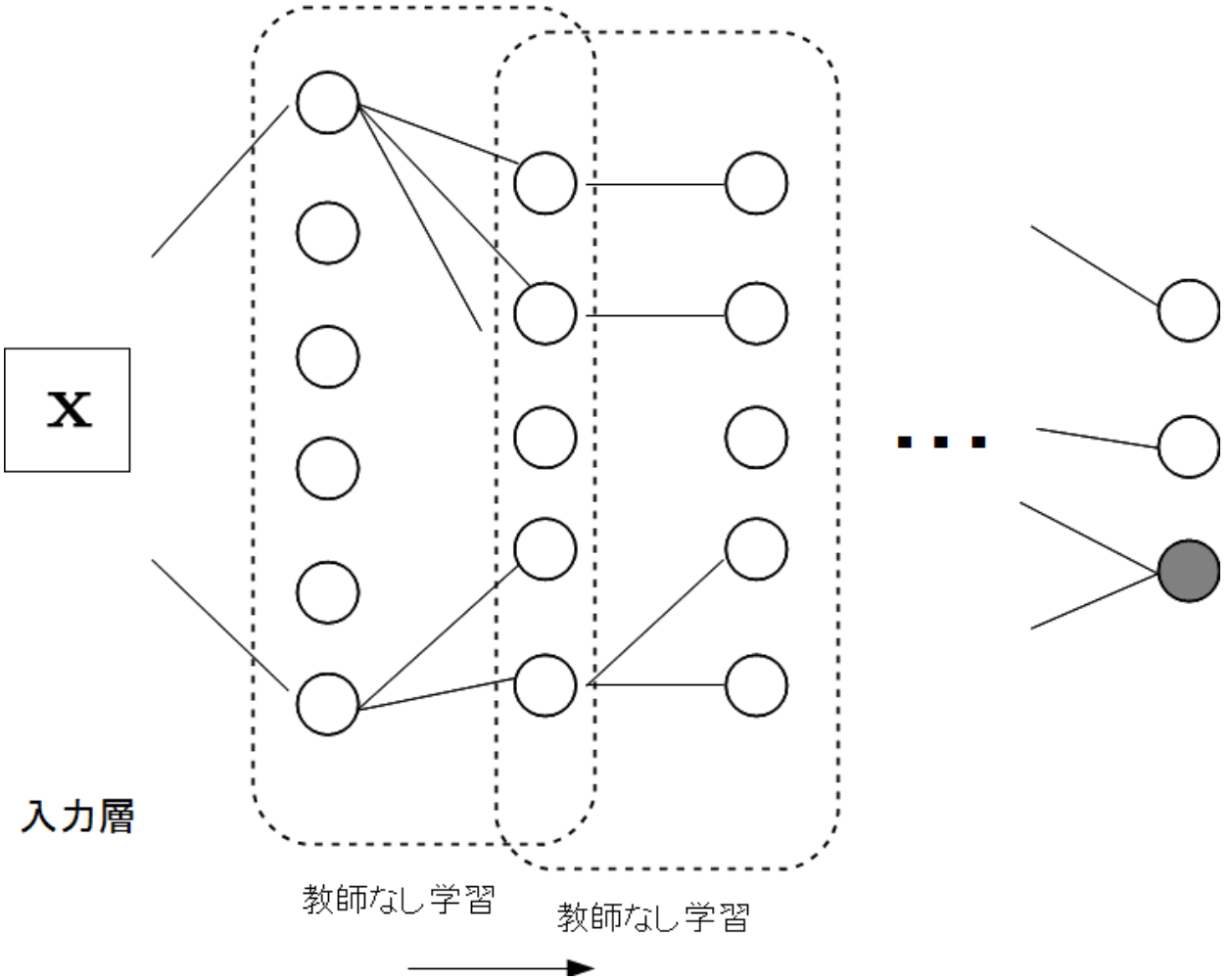
- 多階層における誤差逆伝播法の問題点
  - 修正量が消失／発散する

順方向：非線形　　逆方向：線形



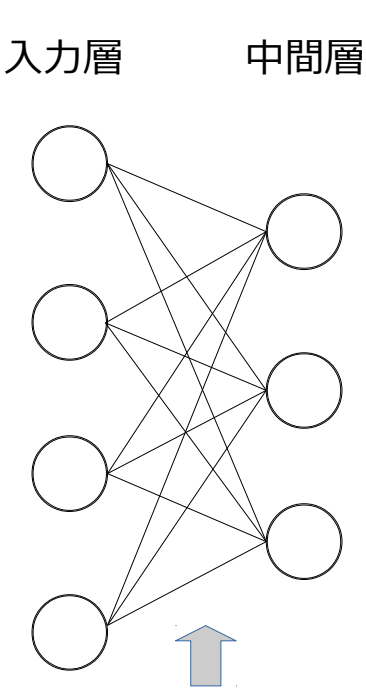
15.2 多階層ニューラルネットワークの学習

- 事前学習法のアイディア
  - 深層学習における初期パラメータ学習

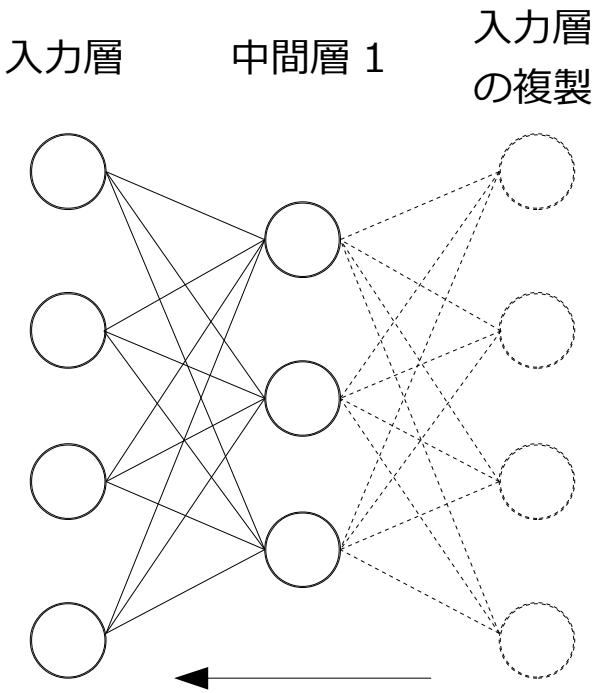


15.3 Autoencoder

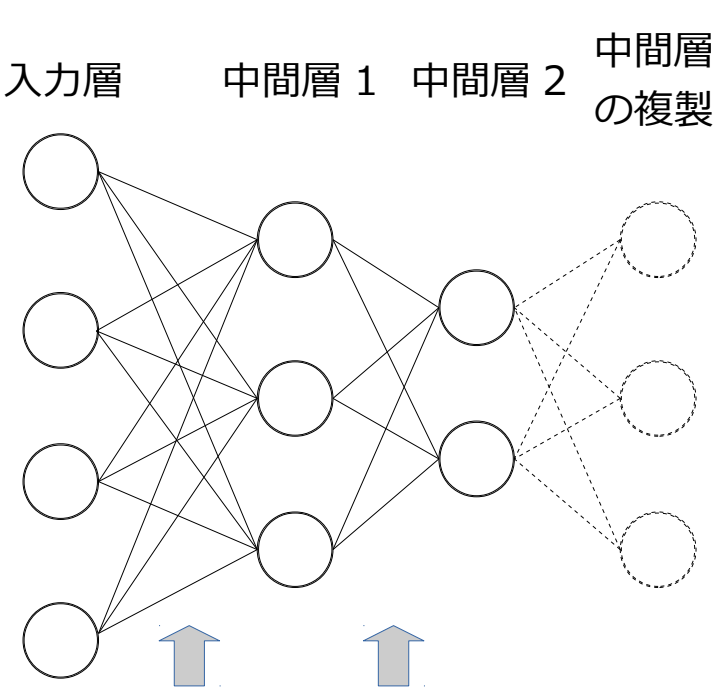
- autoencoder のアイディア：自己写像を行う



(a) 事前調整対象の重み



(b) オートエンコーダによる復元学習



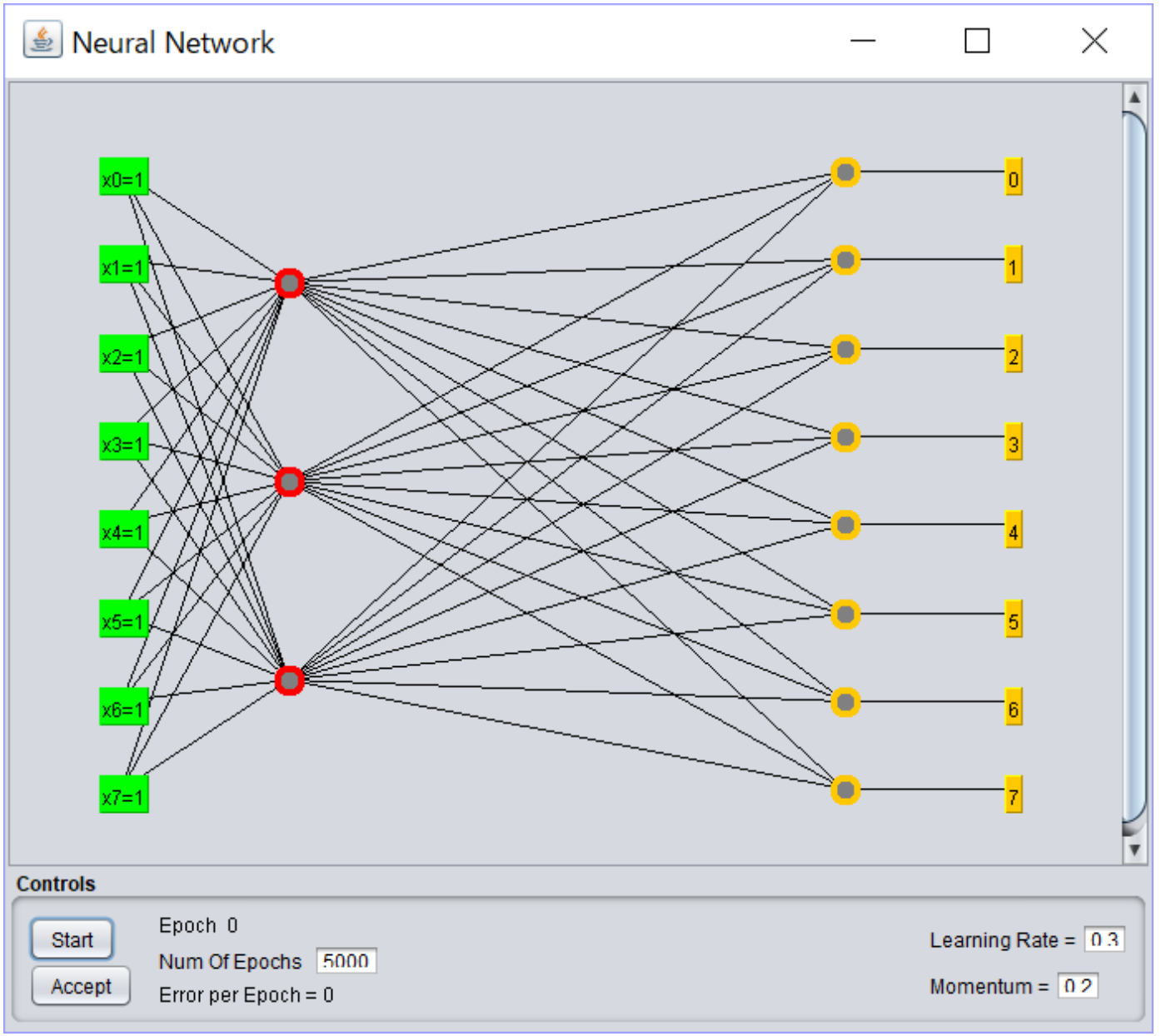
(c) 1 階層上の事前調整

15.3 Autoencoder

- 例題 15.1
  - ２進数の概念を獲得する autoencoder の実現
  - 入力： 0 ～ 7 の数字の one-hot 表現
  - 中間層：ユニット数 3
  - 出力： 0 ～ 7 の 8 クラス
  - 学習回数： 5000 回

```
@relation autoencoder

@attribute x0 {0,1}
@attribute x1 {0,1}
...
@attribute x7 {0,1}
@attribute class
    {0,1,2,3,4,5,6,7}

@data
1,0,0,0,0,0,0,0,0
0,1,0,0,0,0,0,0,1
0,0,1,0,0,0,0,0,2
0,0,0,1,0,0,0,0,3
0,0,0,0,1,0,0,0,4
0,0,0,0,0,1,0,0,5
0,0,0,0,0,0,1,0,6
0,0,0,0,0,0,0,1,7
```

15.3 Autoencoder

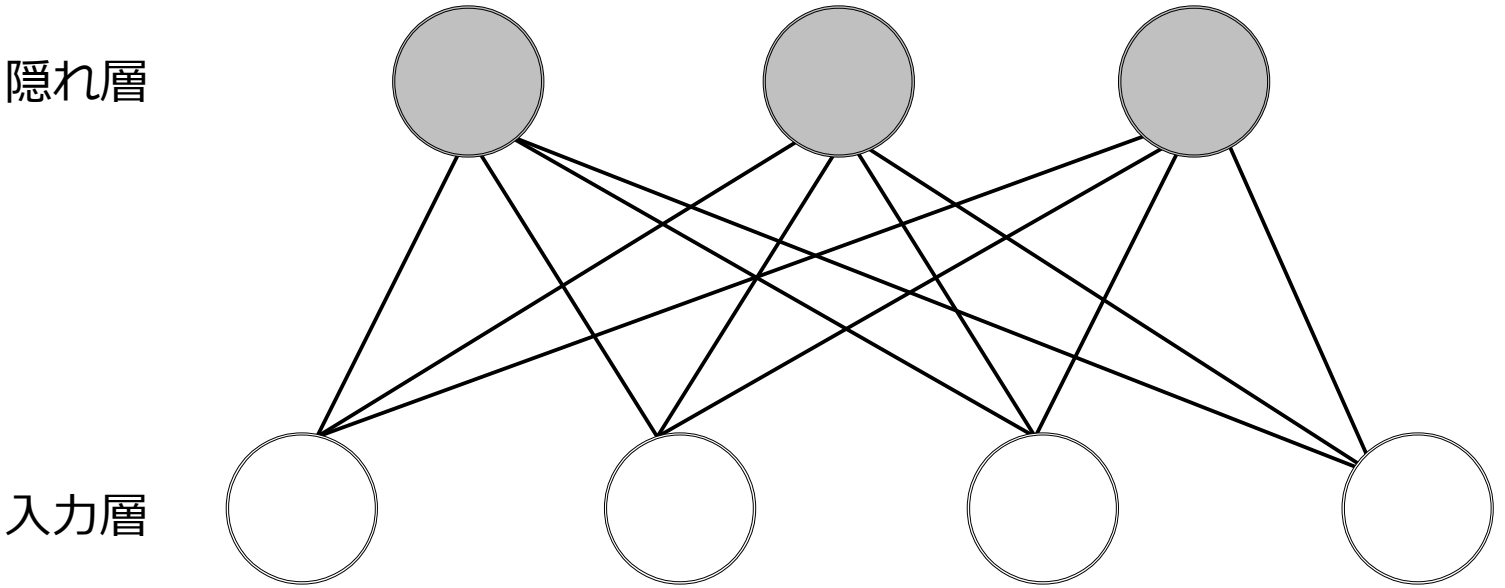


15.4 RBM

- RBM(Restricted Boltzmann Machine)
  - 事前学習の際に利用
  - 生起確率の高い入力 $\boldsymbol{x}$ に対して、エネルギー Φ が高くなるように重み $\boldsymbol{w}$ と閾値 $\boldsymbol{\theta}$ を学習

$$\Phi(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{w}) = -\sum_{i \in \Omega} \theta_i x_i - \sum_{(i,j) \in E} w_{ij} x_i x_j$$

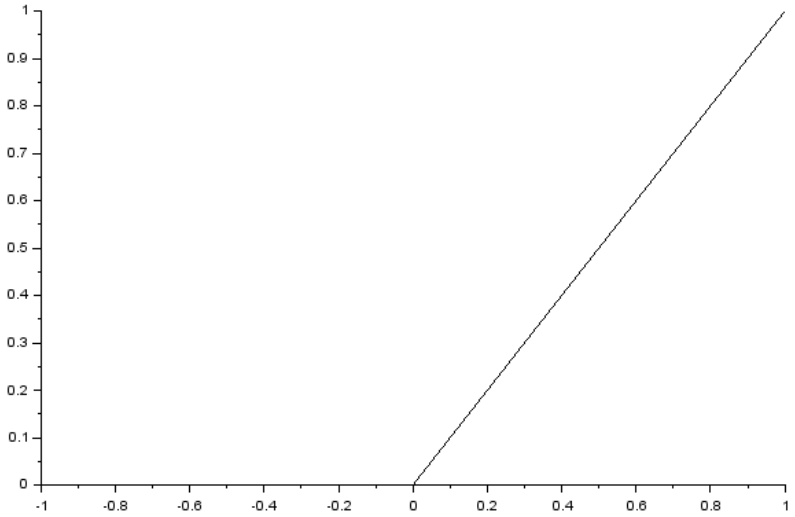
Ω : ノード
E : エッジ



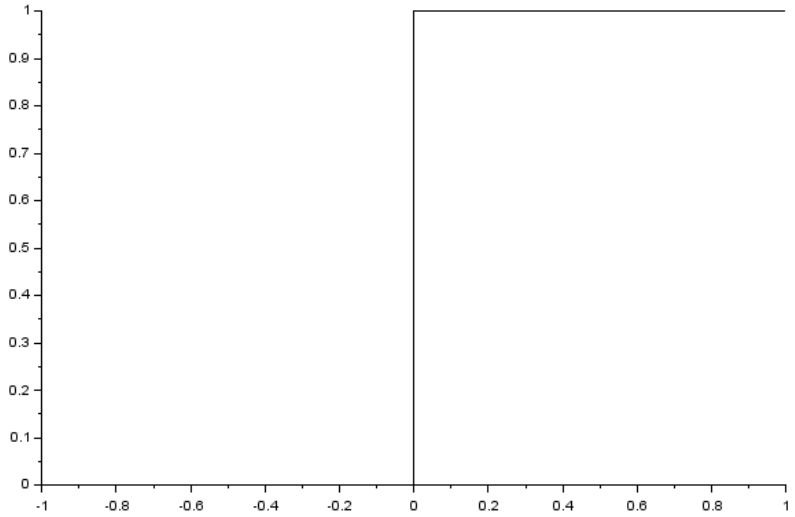
多階層学習における工夫

- 活性化関数を rectified linear 関数に ➡ RELU

$$f(x) = \max(0, x)$$



(a) rectified linear 関数



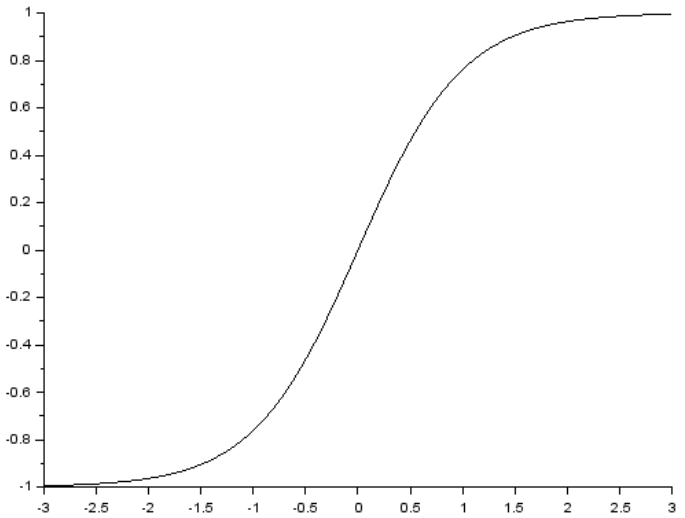
(b) (a) の導関数

- RELU の利点
  - 誤差消失が起こりにくい
  - 0 を出力するユニットが多くなる

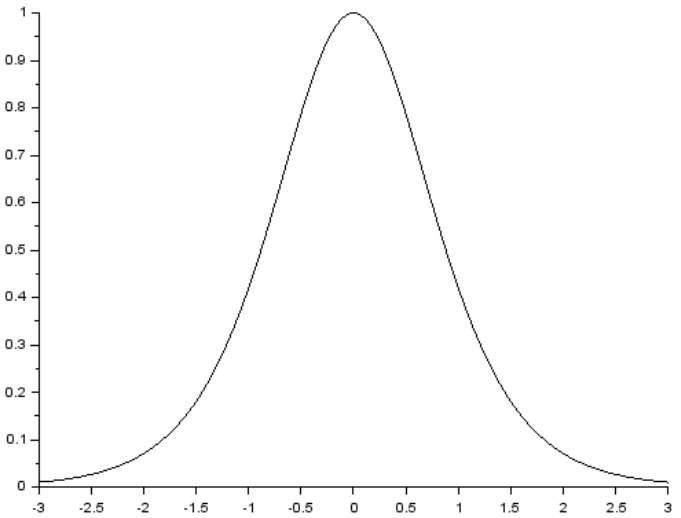
多階層学習における工夫

- 活性化関数を双曲線正接 tanh 関数に

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



(a) tanh 関数



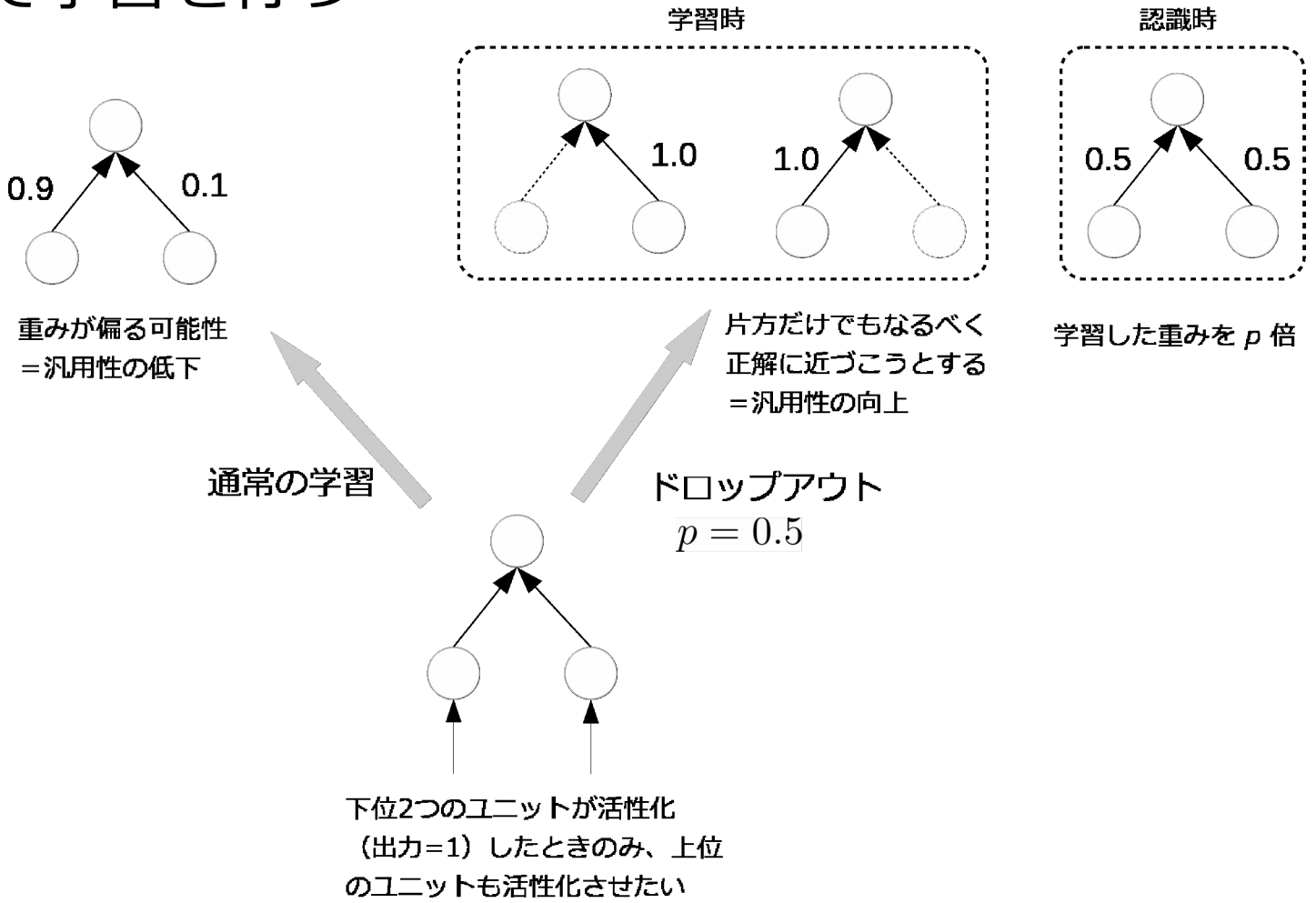
(b) (a) の導関数

- tanh の利点
  - 誤差消失が起こりにくい

    cf) sigmoid は微分係数の最大値が 0.25

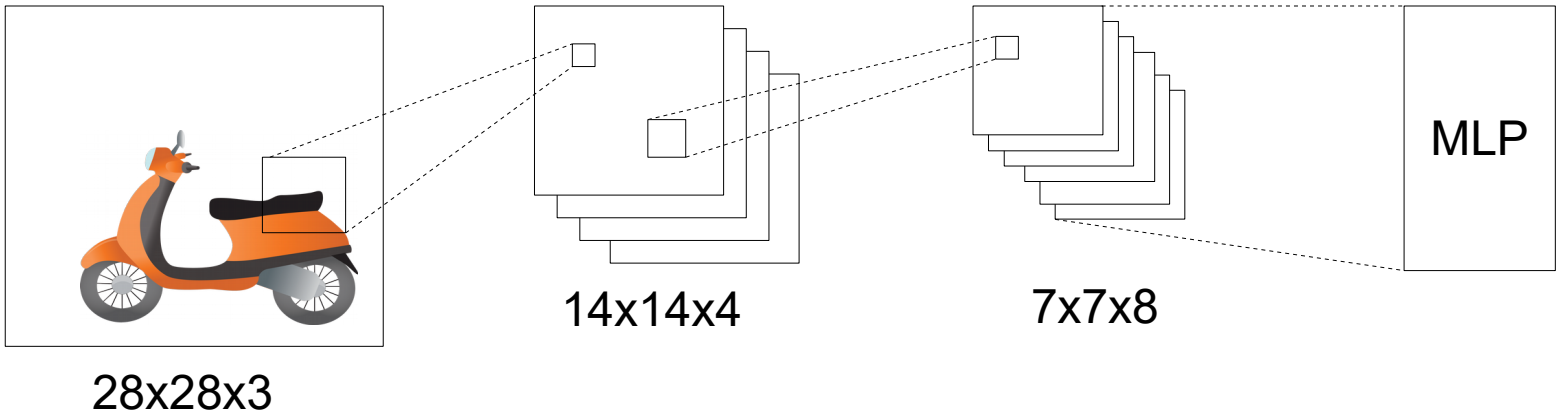
多階層学習における工夫

- 過学習の回避
  - ドロップアウト：ランダムに一定割合のユニットを消して学習を行う



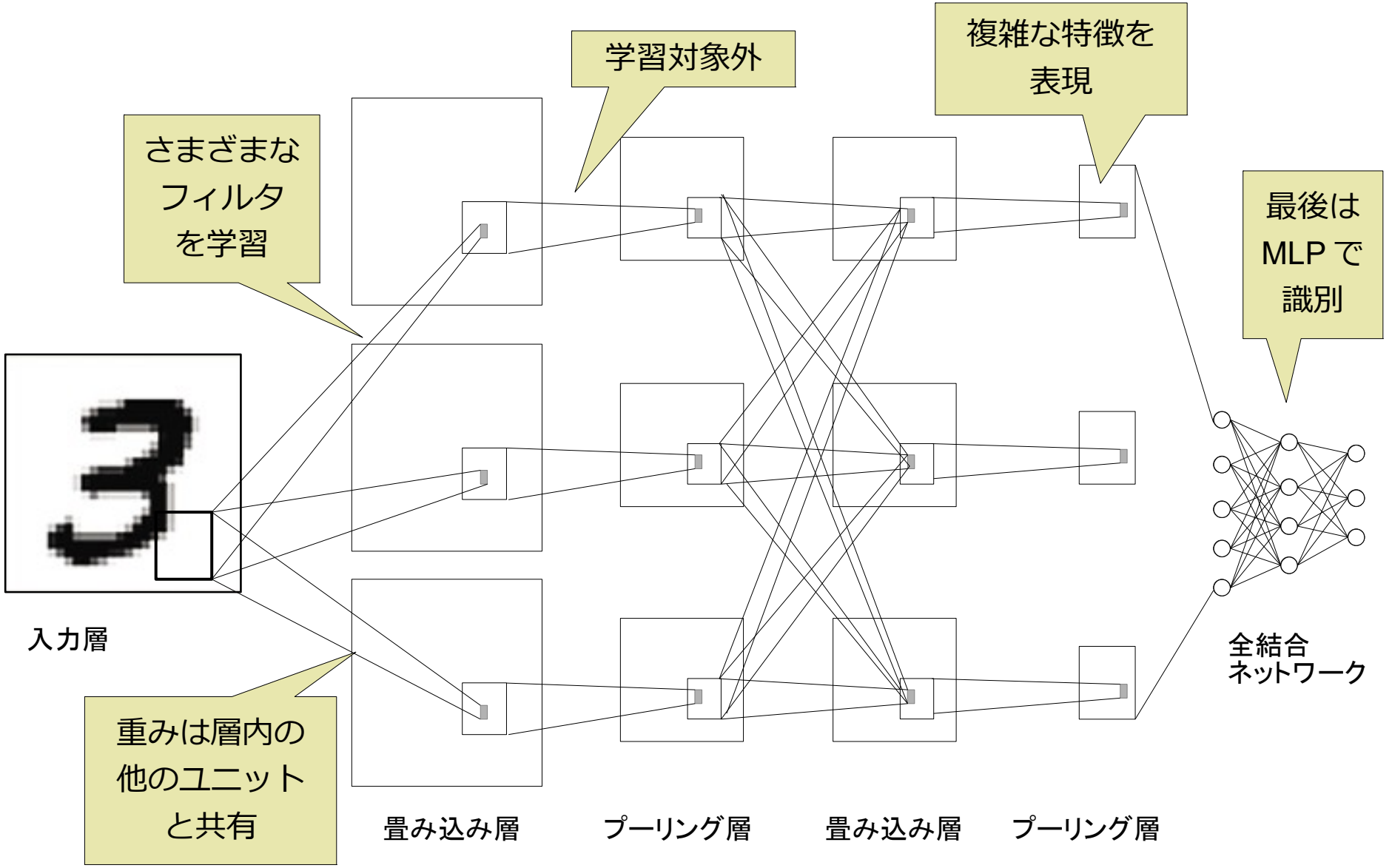
畳み込みニューラルネットワーク

- 畳み込みニューラルネットワークの構造
  - 畳み込み層とプーリング層を交互に重ねる
    - 畳み込み層はフィルタの画素数・ずらす画素数・チャネル数の情報からなる
  - 最後は通常の MLP （ Relu+softmax ）



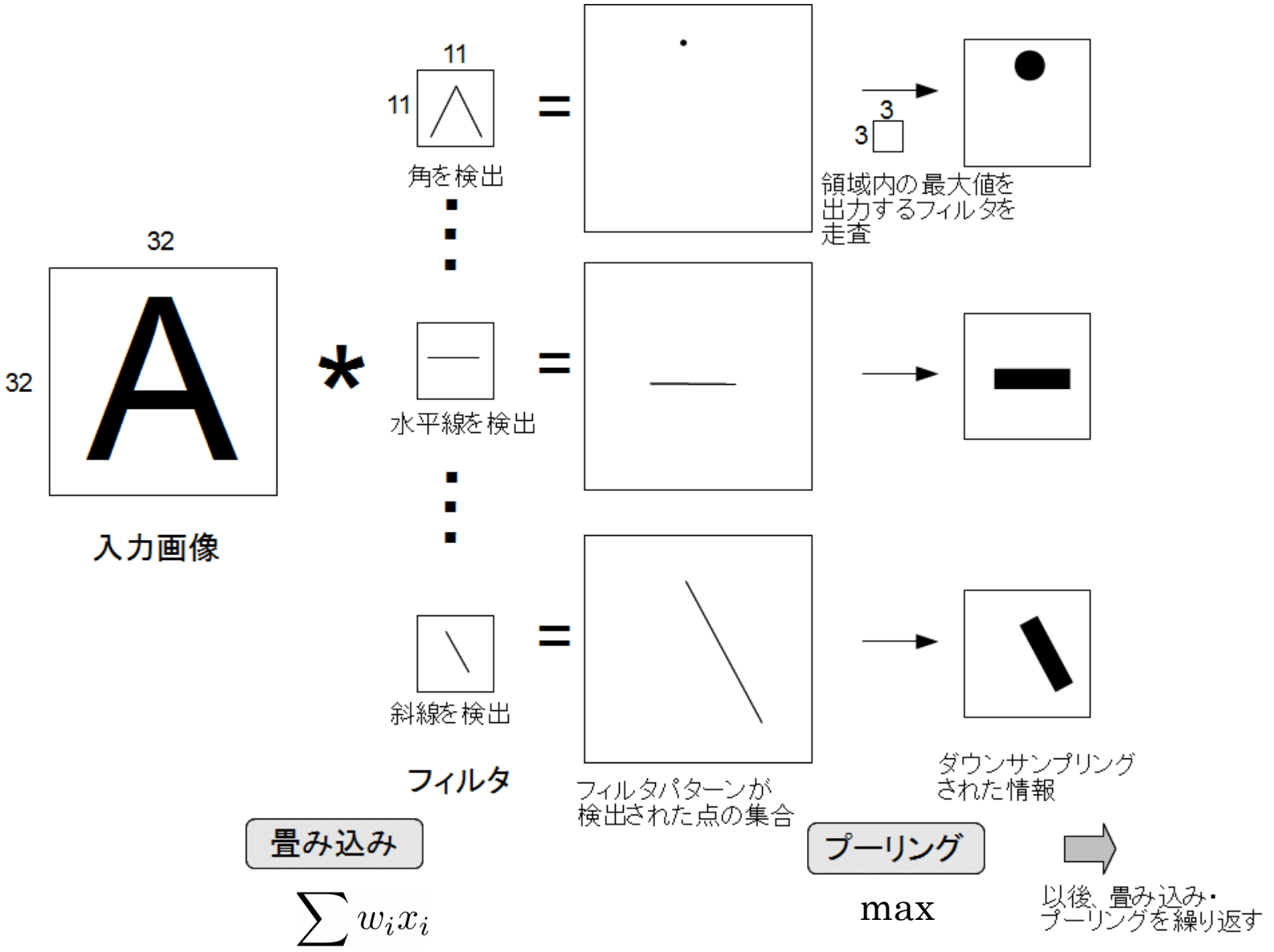
畳み込みニューラルネットワーク

- 畳み込みニューラルネットワークにおける学習



畳み込みニューラルネットワーク

- 畳み込みニューラルネットワークの演算

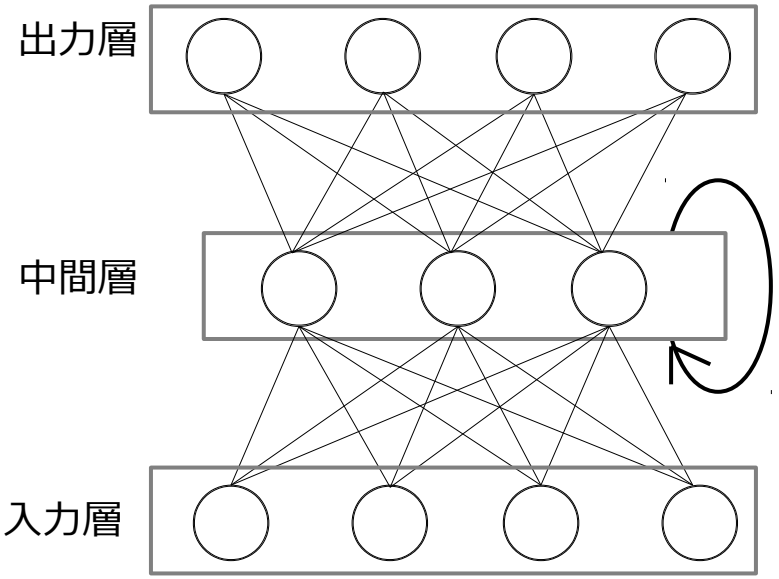


畳み込みニューラルネットワーク

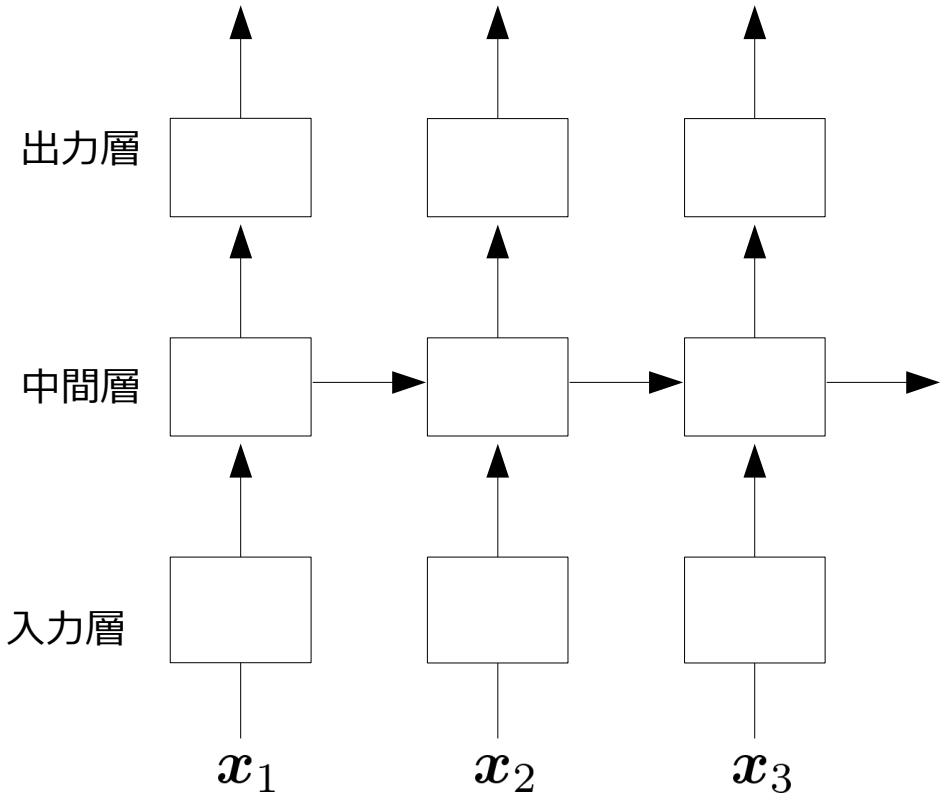
- バッチ標準化の必要性
  - 入力データが標準化されていることは前提
  - 多階層のネットワークで演算を行うと、それぞれの階層の出力が適切な範囲に収まっているとは限らない（たとえば正の大きな値ばかりかもしれない）
- 標準化演算
  - 平均値を引いて標準偏差で割る
  - 1層のネットワークで実現可能

リカレントニューラルネットワーク

- 時系列信号の認識や自然言語処理に適する



(a) リカレントニューラルネットワーク



(b) 帰還路を時間方向に展開

リカレントニューラルネットワーク

- リカレントネットワークの学習
  - 通常の誤差逆伝播法の更新式

    $$w'_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$$

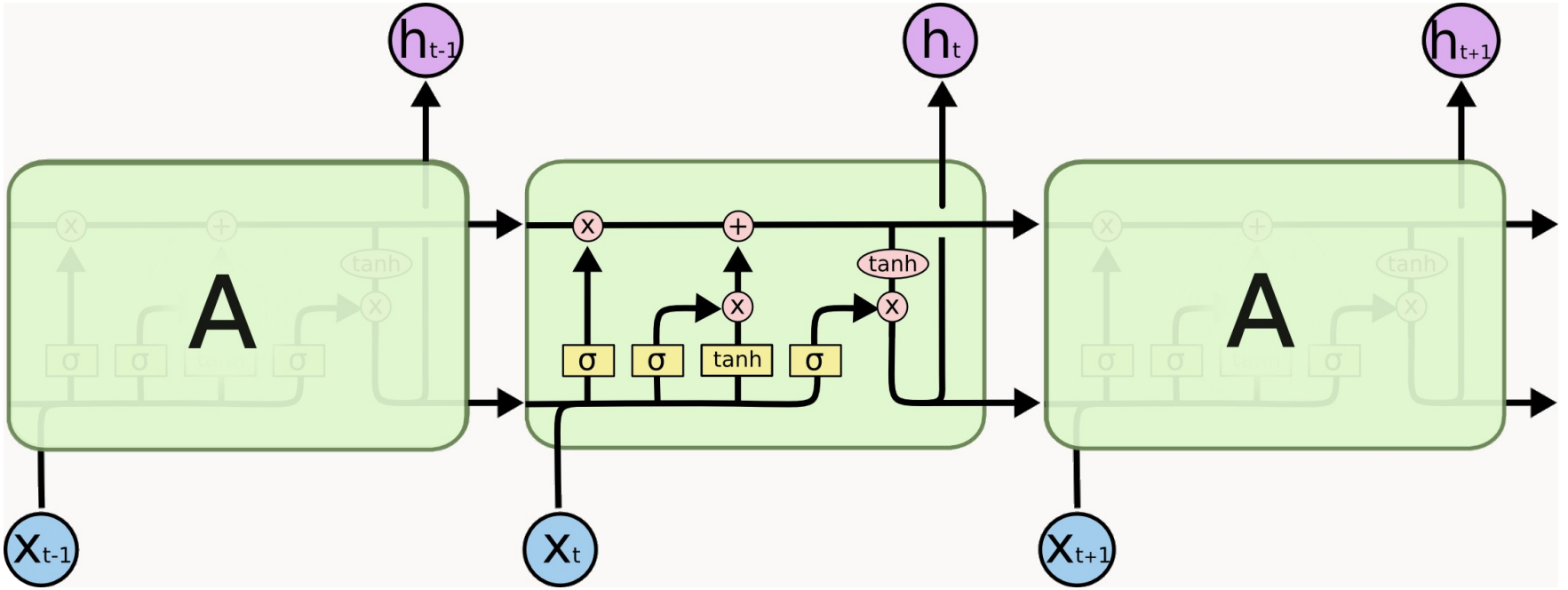
    に対して、時間を遡った更新が必要
  - 時刻 t において、 k 個過去に遡った更新式

    $$w_{ji}(t) \leftarrow w_{ji}(t-1) + \sum_{z=0}^{k} \eta\delta_j(t-z)x_{ji}(t-z-1)$$

  - 勾配消失を避けるため、 k=10 ～ 100 程度とする

リカレントニューラルネットワーク

- LSTM (long short-term memory)
  - いくつかのゲートからなる内部構造をもつユニット
    - ゲート：選択的に情報を通すメカニズム



  - 参考サイト

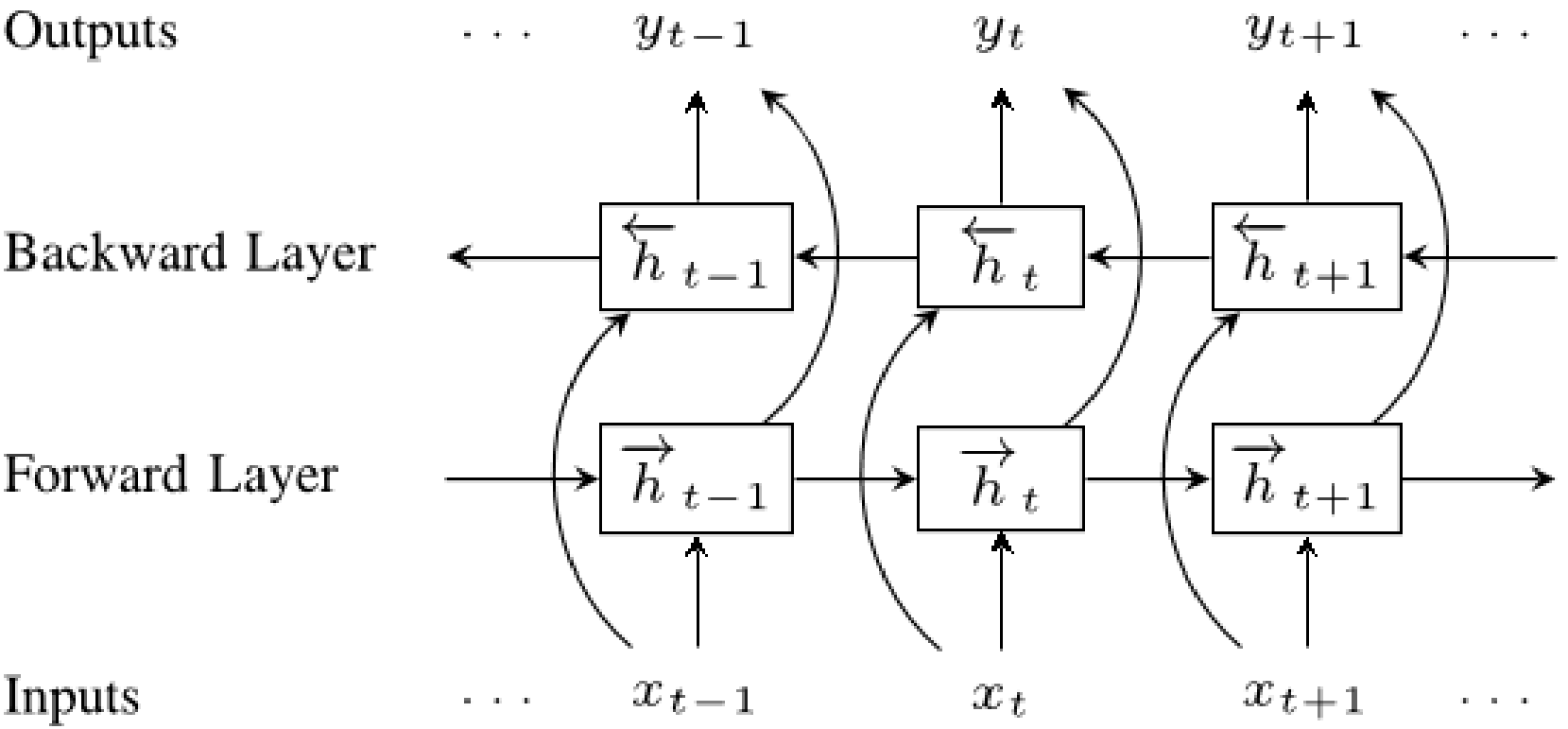
    http://colah.github.io/posts/2015-08-Understanding-LSTMs/

リカレントニューラルネットワーク

- LSTM のゲート
  - 忘却ゲート：セルの内容を捨てるかどうか
    - 例）言語モデルにおいて、新たな主語が現れた場合、古い主語の性別は捨てる
  - 入力ゲート：セルの内容のどの部分を更新するか
    - 例）古い主語の性別を新たな主語の性別で置き換える
  - 出力ゲート：セルの内容のどの部分を出力するか
    - 例）主語に続く動詞の形を決めるために、主語の単複を出力

リカレントニューラルネットワーク

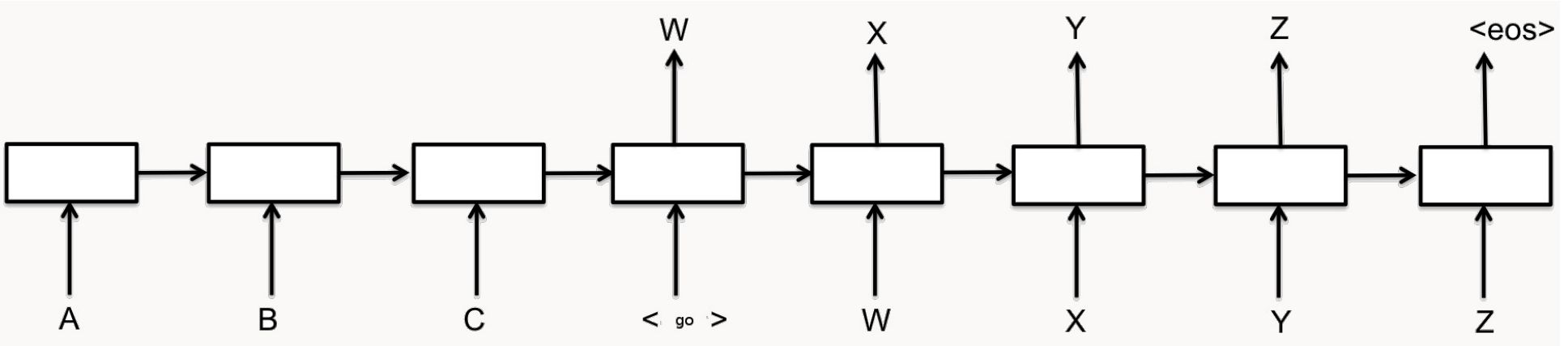
- Bidirectional RNN
  - 過去だけでなく、未来の情報も用いて出力を計算



He, L., Qian, Y., Soong, F.K., Wang, P., & Zhao, H. (2015). A Unified Tagging Solution: Bidirectional LSTM Recurrent Neural Network with Word Embedding. CoRR, abs/1511.00215.

リカレントニューラルネットワーク

- Encoder-Decoder
  - 入力の内容をひとつの表現にまとめて、そこから出力を生成



arXiv:1406.1078

Tensorflow 入門

- 基本的な考え方
- 単純パーセプトロン（ロジステック識別）
- 多層パーセプトロン

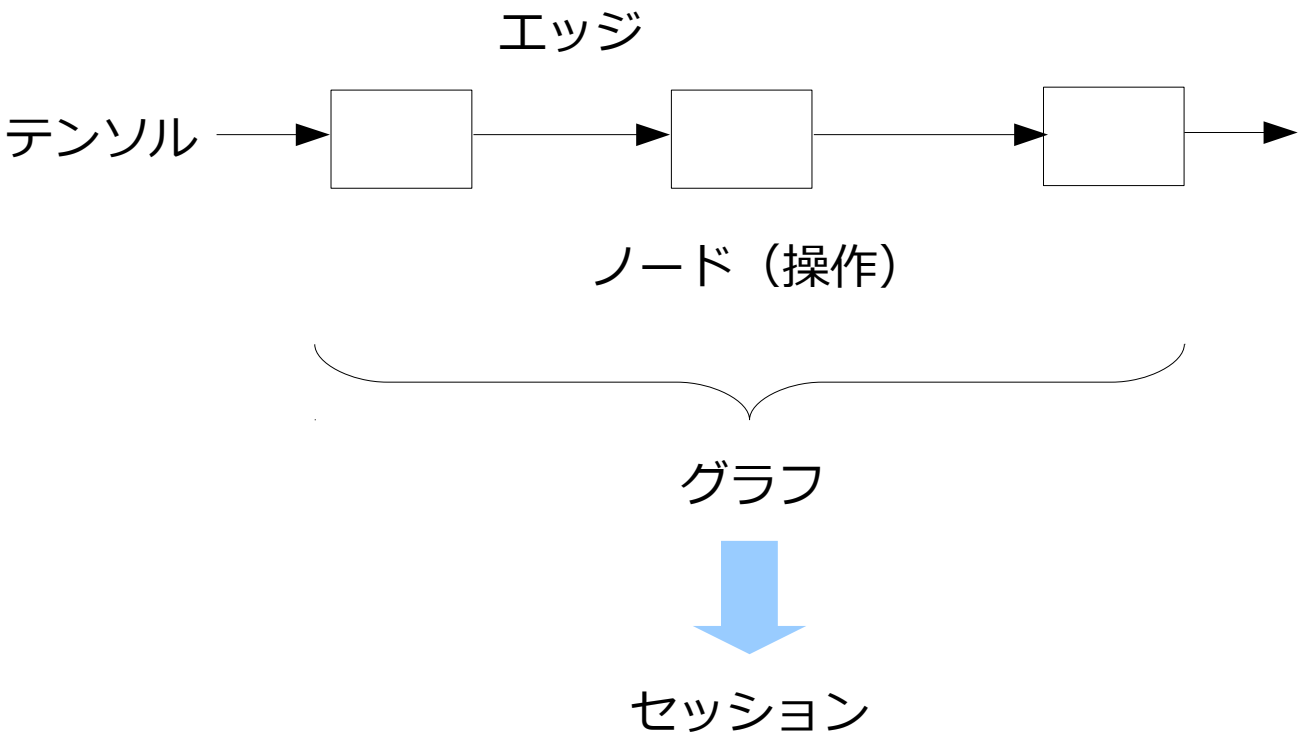
参考資料

Learn TensorFlow and deep learning, without a Ph.D.
https://cloud.google.com/blog/big-data/2017/01/learn-tensorflow-and-deep-learning-without-a-phd

Tensorflow の理解に必要な概念

- テンソル
  - n 次元にデータを並べたもの
    - 1 次元：ベクトル
    - 2 次元：行列
- ノード
  - 入出力と操作を定義されたもの
- エッジ
  - テンソルが流れる標準エッジと制御を示す特殊エッジがある

Tensorflow の理解に必要な概念

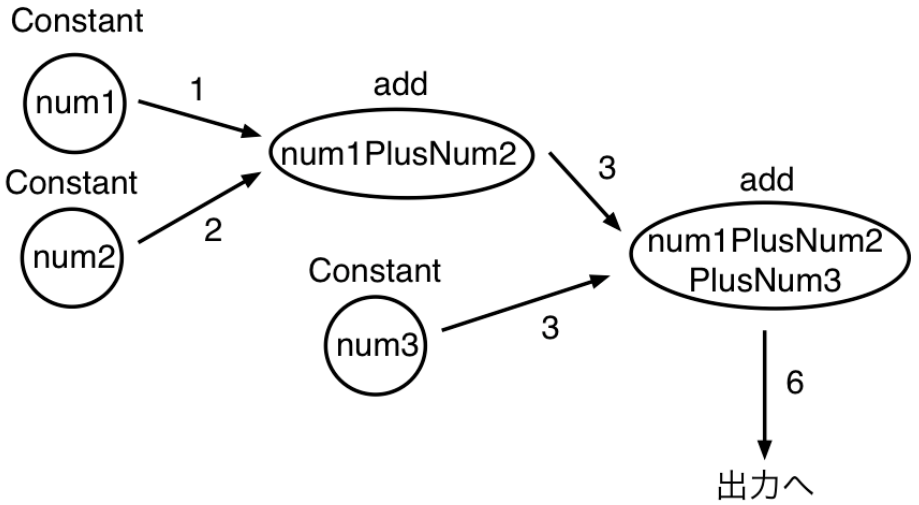


Tensorflow の理解に必要な概念

- グラフによる計算の表現

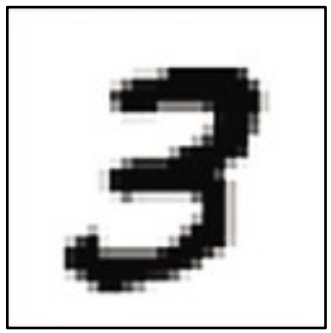
```
import tensorflow as tf
num1 = tf.constant(1)
num2 = tf.constant(2)
num3 = tf.constant(3)
num1PlusNum2 = tf.add(num1,num2)
num1PlusNum2PlusNum3 = tf.add(num1PlusNum2,num3)
sess = tf.Session()
result = sess.run(num1PlusNum2PlusNum3)
print(result)
```

この出力に必要な計算は
自動的に行われる

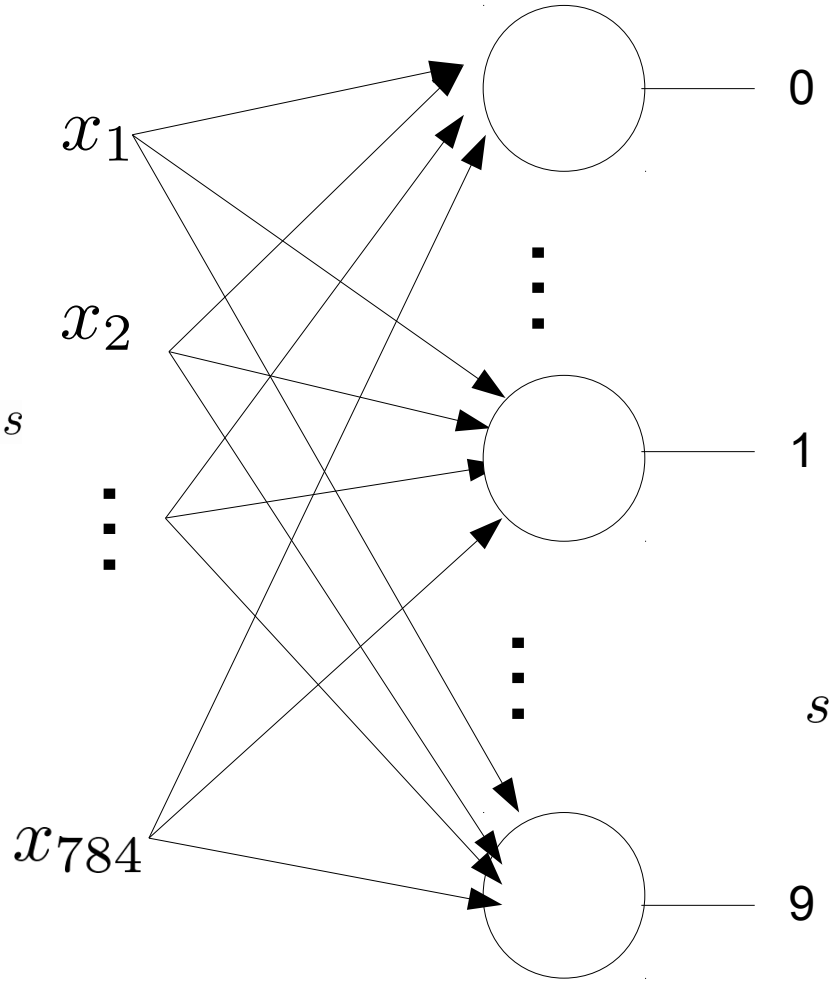


ロジスティック識別

- 中間層なし・ softmax 出力の NN



$28 \times 28\ pixels$



$$L_n = \sum_{i=1}^{784} W_{ni}.X_i + b_n$$

$$softmax(L_n) = \frac{e^{L_n}}{||e^L||}$$

入力層

出力層

softmax 演算

ロジスティック識別

- Tensorflow のコード（準備）

```
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

init = tf.global_variables_initializer()
```

バッチサイズ　入力の次元数

変数の初期化

placeholder : データが格納される予定地

Variable: 変数

ロジスティック識別

- パターン行列による表現

$$\boldsymbol{Y} = softmax(\boldsymbol{X} \cdot \boldsymbol{W} + \boldsymbol{b})$$

- Tensorflow のコード

```
Y = tf.nn.softmax(tf.matmul(X, W) + b)
```

ロジスティック識別

- 誤差関数
  - クロスエントロピー

$$-\sum \boldsymbol{Y}_i' \cdot \log(\boldsymbol{Y}_i)$$

$\boldsymbol{Y}_i'$: one-hot エンコーディングされた教師信号

$\boldsymbol{Y}_i$: NN の出力

正解出力の負の対数値だけが評価されている

  - 二乗誤差
    - 不正解出力の誤差が過剰に評価されており、正解への近さの評価が消えてしまう

参考資料

Why You Should Use Cross-Entropy Error Instead Of Classification Error Or Mean Squared Error For Neural Network Classifier Training
https://jamesmccaffrey.wordpress.com/2013/11/05/

ロジスティック識別

- Tensorflow のコード（モデルと評価値の設定）

```
# model
Y = tf.nn.softmax(tf.matmul(X, W) + b)
# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
```

テンソルをフラットにして和を求める

```
# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))


# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

ロジスティック識別

- Tensorflow のコード（学習の設定）

学習係数

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)
```

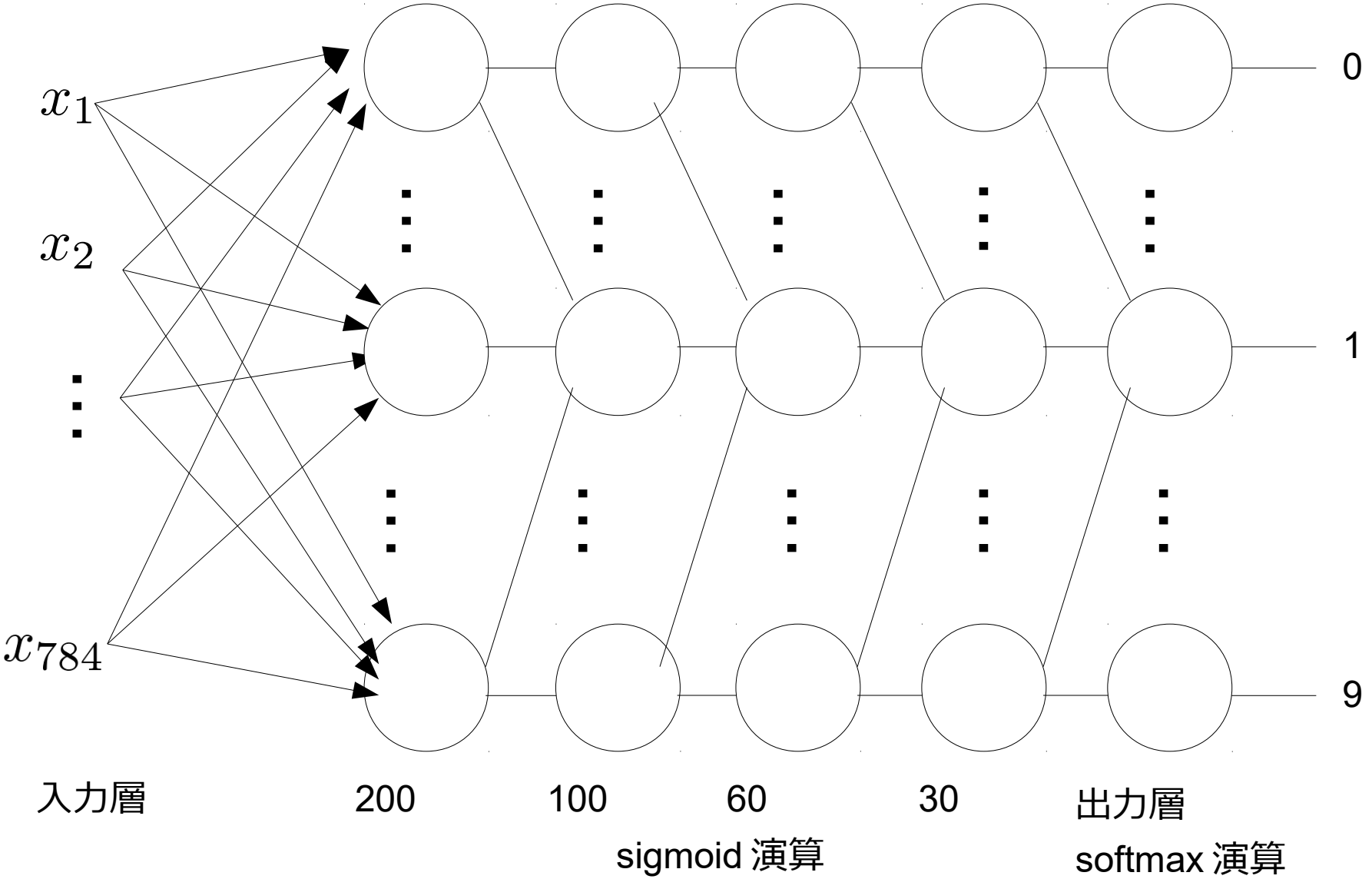
ロジスティック識別

- Tensorflow のコード（学習）

```
sess = tf.Session()
sess.run(init)

for i in range(1000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ?
    a,c = sess.run([accuracy, cross_entropy], feed_dict=train_data)

    # success on test data ?
    test_data={X: mnist.test.images, Y_: mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed_dict=test_data)
```

多層パーセプトロン

- 中間層を 4 層加える



多層パーセプトロン

- Tensorflow のコード（準備）

```
K = 200
L = 100
M = 60
N = 30
```

重みの初期値を乱数で設定

```
W1 = tf.Variable(tf.truncated_normal([28*28,K],stddev=0.1))
B1 = tf.Variable(tf.zeros([K]))

W2 = tf.Variable(tf.truncated_normal([K, L], stddev=0.1))
B2 = tf.Variable(tf.zeros([L]))

W3 = tf.Variable(tf.truncated_normal([L, M], stddev=0.1))
B3 = tf.Variable(tf.zeros([M]))
W4 = tf.Variable(tf.truncated_normal([M, N], stddev=0.1))
B4 = tf.Variable(tf.zeros([N]))
W5 = tf.Variable(tf.truncated_normal([N, 10], stddev=0.1))
B5 = tf.Variable(tf.zeros([10]))
```

多層パーセプトロン

- Tensorflow のコード（モデルの設定）

```
Y1 = tf.nn.sigmoid(tf.matmul(X, W1) + B1)
Y2 = tf.nn.sigmoid(tf.matmul(Y1, W2) + B2)
Y3 = tf.nn.sigmoid(tf.matmul(Y2, W3) + B3)
Y4 = tf.nn.sigmoid(tf.matmul(Y3, W4) + B4)
Y = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
```

活性化関数を Relu とするコード

```
Y1 = tf.nn.relu(tf.matmul(X, W1) + B1)
```

Relu を使うときはバイアス B を正の値に初期化