

機械学習 (5)

概要

1. 機械学習の基礎

- 概論
 - 教師あり学習
 - 教師なし学習
 - 強化学習

2. 教師あり学習

- ランダムフォレスト
- サポートベクターマシン

3. 教師なし学習と確率モデリング

- クラスタリング
- ナイーブベイズモデル
- 混合ガウスモデル
- クロスバリデーション・モデル選択

4. データ構造と機械学習アルゴリズム

- テーブルデータ
- 行列データ
- 時系列データ
- グラフデータ

補助資料：<http://small-island.work/trial/>

5. 深層学習

ニューラルネットワークの基礎

深層学習の基礎

画像処理・自然言語処理におけるモデル

まず始めに

深層学習（ディープラーニング）

今後データ分析を行う上で避けては通れない技術

いつでも利用可能というわけではないが、
さまざまなデータに適用可能な強力な手法
特に、画像・自然言語・音声などではよく使う

深層学習の勉強方法

どんどん新しくなっていく技術に対応するために、
きちんとした基礎と基本的な手法を押さえておくことが必要
(必要になった時に最新のものをすぐに学べるようにしておくことが重要)

深層学習に関しては経験的な技術やライブラリの使い方などの占める割合も多いのである程度の実践も大事

この講義の目的

目的

1. ニューラルネットワークが何をしているかを学ぶ
2. 深層学習の基本的な技術を学ぶ
3. 有名なモデルにこういったものがあるかの紹介

人工知能の歴史

計算と人工知能

人工知能

1950年：チューリングテスト

1956年：ダートマス会議

論理・探索

第1次AIブーム: 推論と探索

1956年－1974年

1972年：Prolog

第2次AIブーム:
知識

(エキスパートシステム)

1980年－1987年

機械学習 機械学習

1967年：k-means法
(クラスタリング)

1989年：Q学習
(強化学習)

1992年：非線形SVM
(教師あり学習)

1990年～：統計的機械学習

- ・ 確率モデル
- ・ 汎化誤差理論
- ・ ベイズモデル

ニューラルネットワーク

1958年：パーセプトロン

1969年：パーセプトロンの限界

1986年：誤差逆伝搬法

深層学習

2006年：オートエンコーダー

第3次AIブーム:
深層学習
2006年－

「深層学習（ディープラーニング）」 という言葉の広がり(1/3)

← → ↻ 保護された通信 | https://books.google.com/ngrams/graph?content=Deep+learning,machine+learning,Artificial+intelligence&case_insensitive=on&year_...

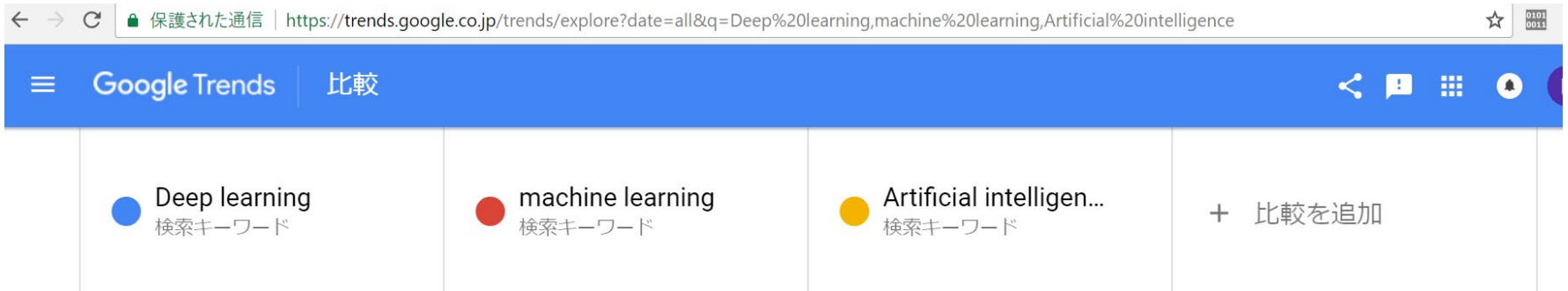
Google Books Ngram Viewer

Graph these comma-separated phrases: case-insensitive

between and from the corpus with smoothing of

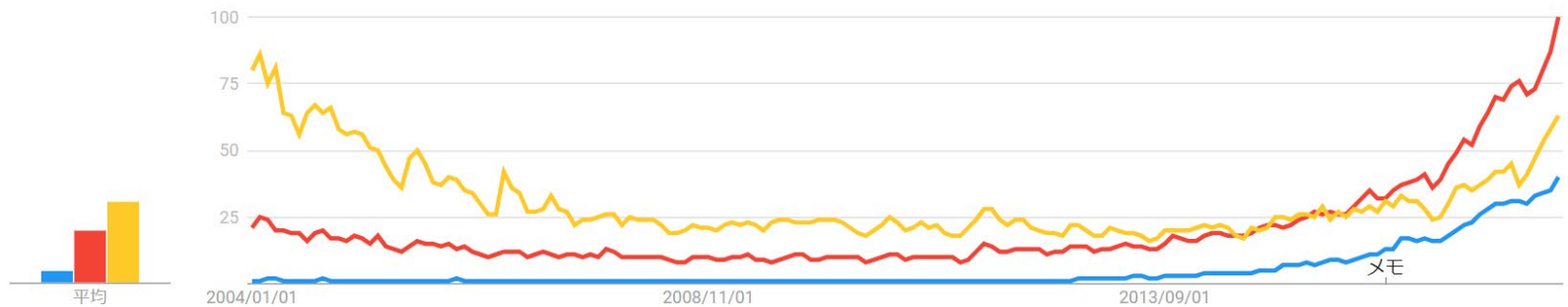


「深層学習（ディープラーニング）」 という言葉の広がり(2/3)



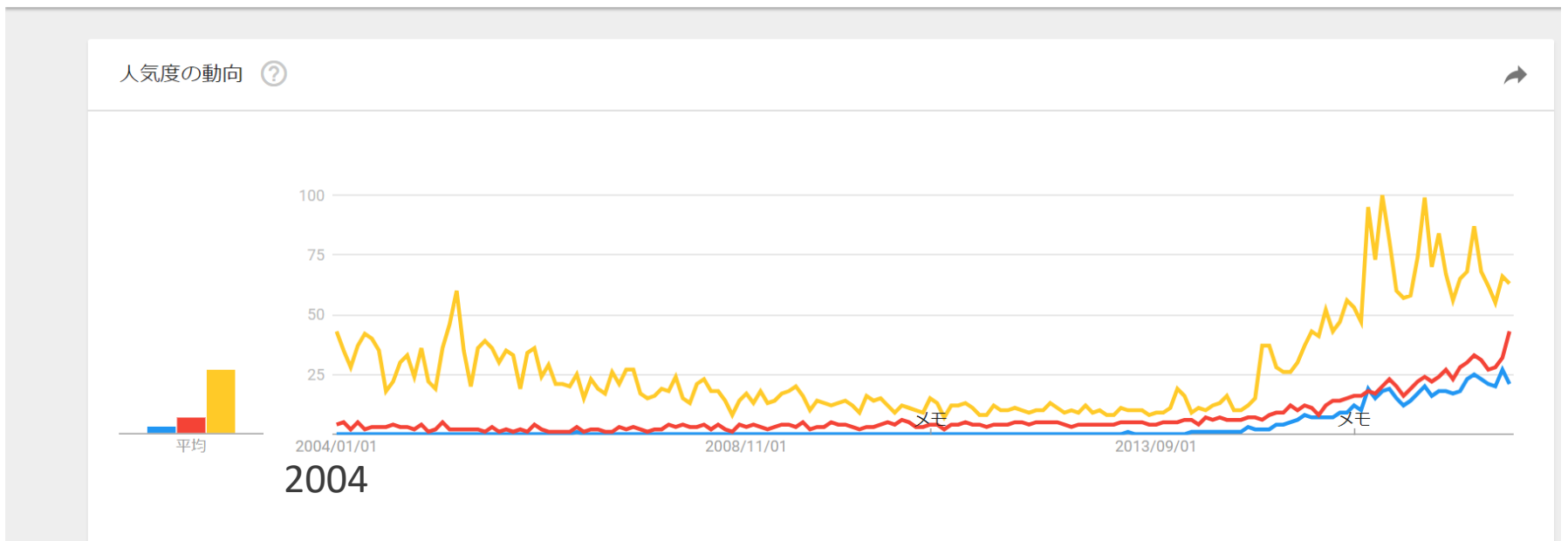
すべての国 ▼ 2004 - 現在 ▼ すべてのカテゴリ ▼ ウェブ検索 ▼

人気度の動向 ⓘ



AlphaGo

「深層学習（ディープラーニング）」 という言葉の広がり(3/3)



人工知能の歴史

計算と人工知能

人工知能

1950年：チューリングテスト
1956年：ダートマス会議

論理・探索

第1次AIブーム: 推論と探索
1956年－1974年

1972年：Prolog

第2次AIブーム:
知識
(エキスパートシステム)
1980年－1987年

機械学習 機械学習

1967年：k-means法
(クラスタリング)

1989年：Q学習
(強化学習)

1992年：非線形SVM
(教師あり学習)

1990年～：統計的機械学習
・確率モデル
・汎化誤差理論
・ベイズモデル

ニューラルネットワーク

1958年：パーセプトロン

1969年：パーセプトロンの限界

1986年：誤差逆伝搬法

深層学習

2006年：オートエンコーダー

第3次AIブーム:
深層学習
2006年－

ニューラルネットワークの基本構造 (1/3)

パーセプトロン(perceptron) [Rosenblatt58]

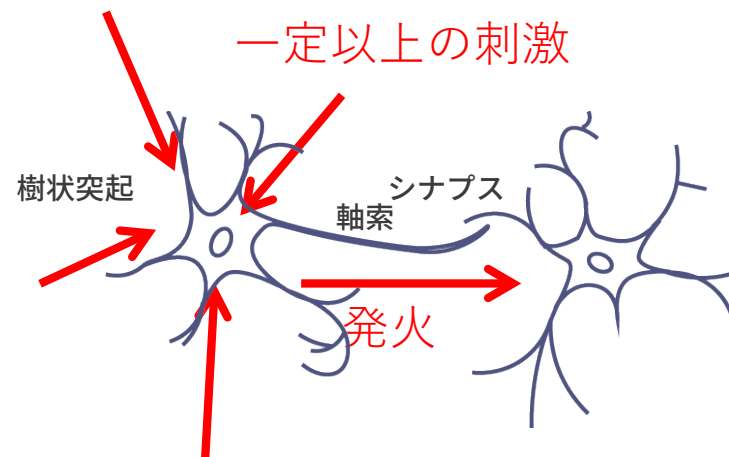
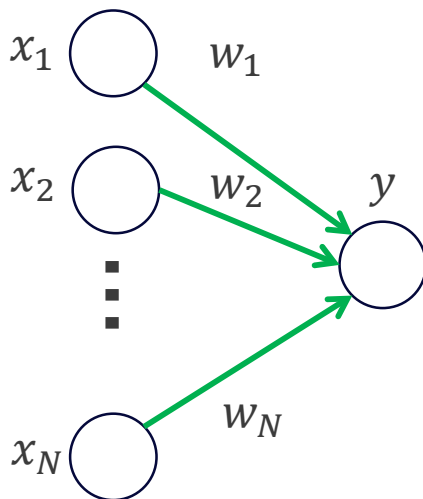
$$y = \sigma \left(\sum_{n=1}^N w_n x_n + b \right)$$

入力： x_1, x_2, \dots, x_N

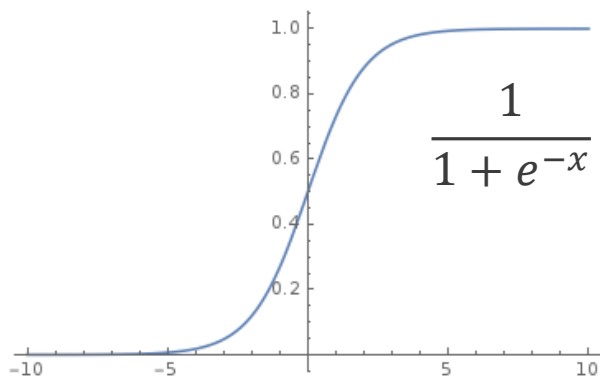
重み： w_1, w_2, \dots, w_N, b

出力： y

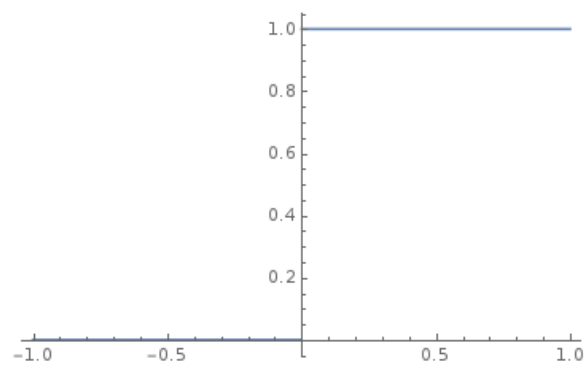
活性化関数： $\sigma(x)$



$\sigma(x)$ ：シグモイド関数



$\sigma(x)$ ：ステップ関数



ニューラルネットワークの基本構造 (2/3)

パーセプトロンの例

$$y = \sigma(w_1x_1 + w_2x_2 + b)$$

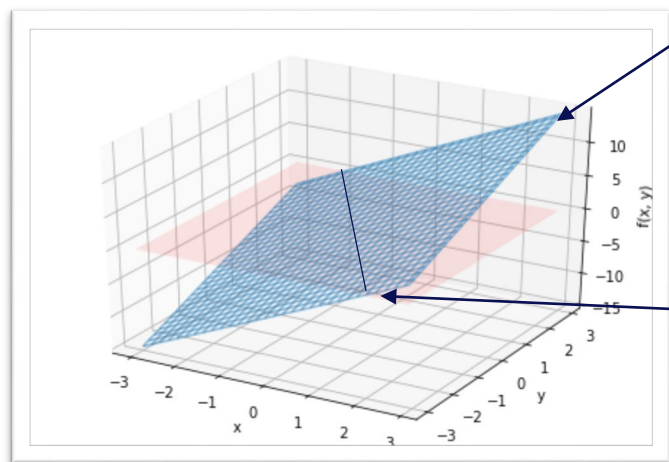
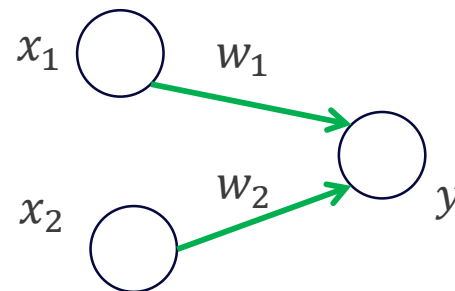
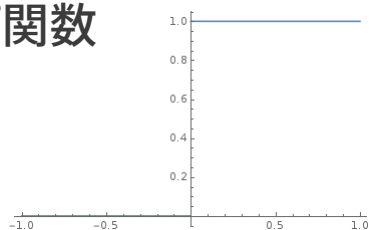
入力： x_1, x_2

重み： w_1, w_2, b

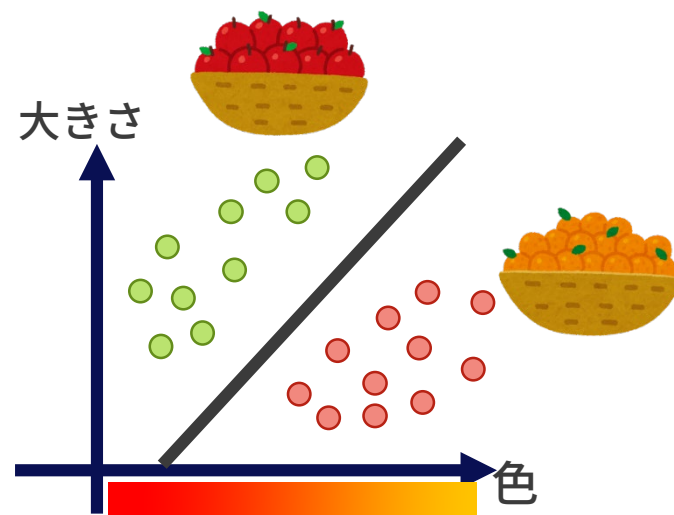
出力： y

活性化関数： $\sigma(x)$

ステップ関数



境界は直線になる



ニューラルネットワークの基本構造 (2/3)

パーセプトロンの例

$$y = \sigma(w_1x_1 + w_2x_2 + b)$$

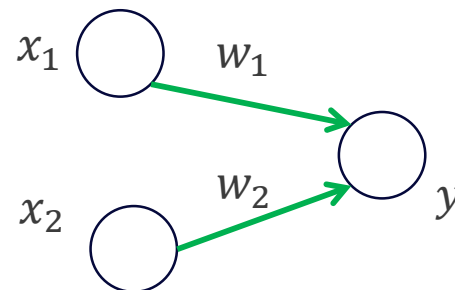
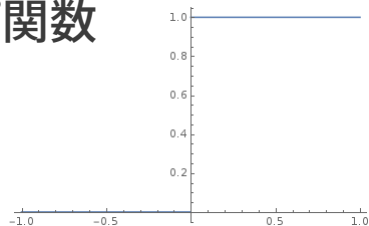
入力： x_1, x_2

重み： w_1, w_2, b

出力： y

活性化関数： $\sigma(x)$

ステップ関数

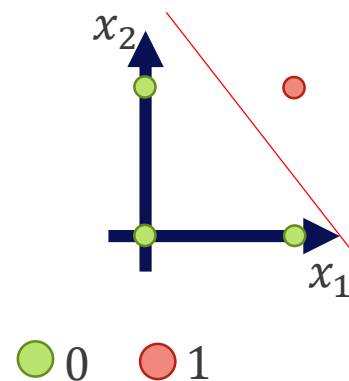


x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



いい感じに分かれるように
線を引きたい
= w_1, w_2, b を決めたい

$$\begin{aligned}w_1 &= 1 \\w_2 &= 1 \\b &= -1.5\end{aligned}$$



ニューラルネットワークの基本構造 (3/3)

パーセプトロン $y = \sigma(w_1x_1 + w_2x_2 + b)$

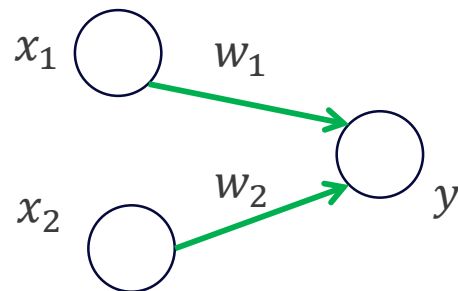
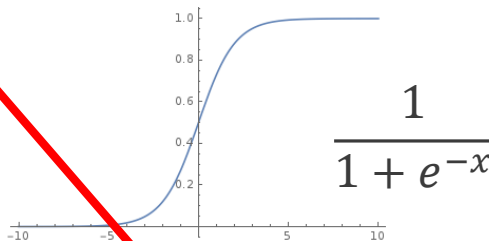
入力: x_1, x_2

重み: w_1, w_2, b

出力: y

活性化関数: $\sigma(x)$

シグモイド関数



パーセプトロンの学習

問題: 左の表を満たす w_1, w_2, b を決定せよ

x_1	x_2	\hat{y}
0	0	0
0	1	0
1	0	0
1	1	1

$$\min_{w_1, w_2, b} \frac{1}{2} \sum_i e_i^2$$

$$e_i = y(x_1^{(i)}, x_2^{(i)}) - \hat{y}^{(i)}$$

i : データのインデックス

二乗誤差の最小化問題

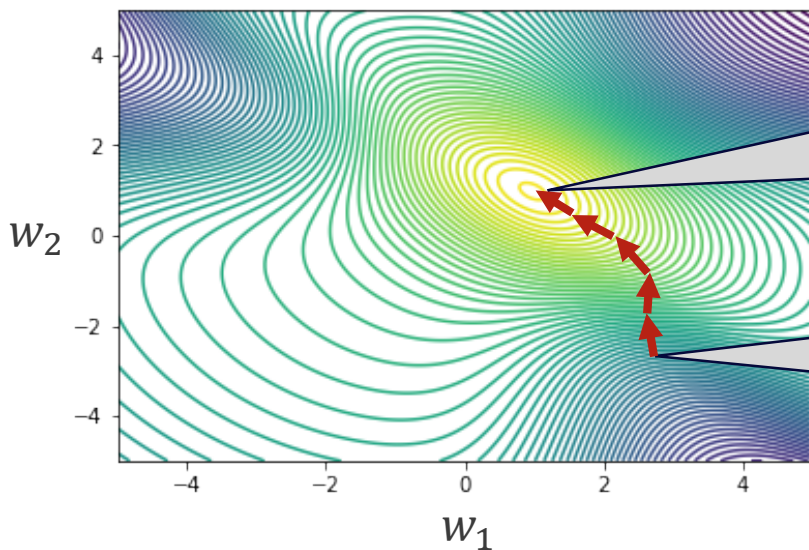
→ 活性化関数があるので単純には解けない

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

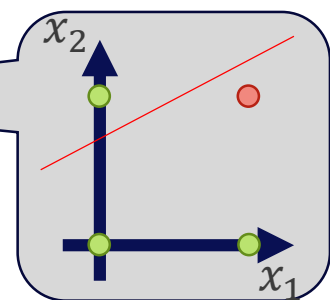
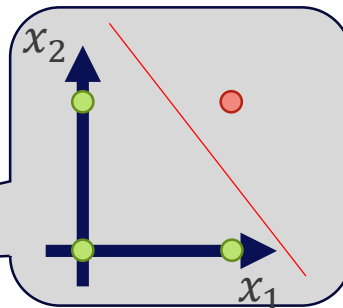
勾配降下法

$$\min_{w_1, w_2, b} \frac{1}{2} \sum_i e_i^2$$

$$e = y - \hat{y}$$
$$y = \sigma(w_1 x_1 + w_2 x_2 + b)$$



$$w_1 x_1 + w_2 x_2 + b$$



● 0
● 1

各変数について以下の更新を何回も行うと最小値に到達できる：

$$w_1 \leftarrow w_1 - \alpha \frac{\partial e^2}{\partial w_1}$$
$$w_2 \leftarrow w_2 - \alpha \frac{\partial e^2}{\partial w_2}$$

α : 学習率 (Learning rate)
小さすぎると学習が進まない
大きすぎると収束しない

人工知能の歴史

計算と人工知能

人工知能

1950年：チューリングテスト
1956年：ダートマス会議

論理・探索

第1次AIブーム: 推論と探索
1956年－1974年

1972年：Prolog

第2次AIブーム:
知識
(エキスパートシステム)
1980年－1987年

機械学習 機械学習

1967年：k-means法
(クラスタリング)

1989年：Q学習
(強化学習)

1992年：非線形SVM
(教師あり学習)

1990年～：統計的機械学習

- ・確率モデル
- ・汎化誤差理論
- ・ベイズモデル

ニューラルネットワーク

1958年：パーセプトロン

1969年：パーセプトロンの限界

1986年：誤差逆伝搬法

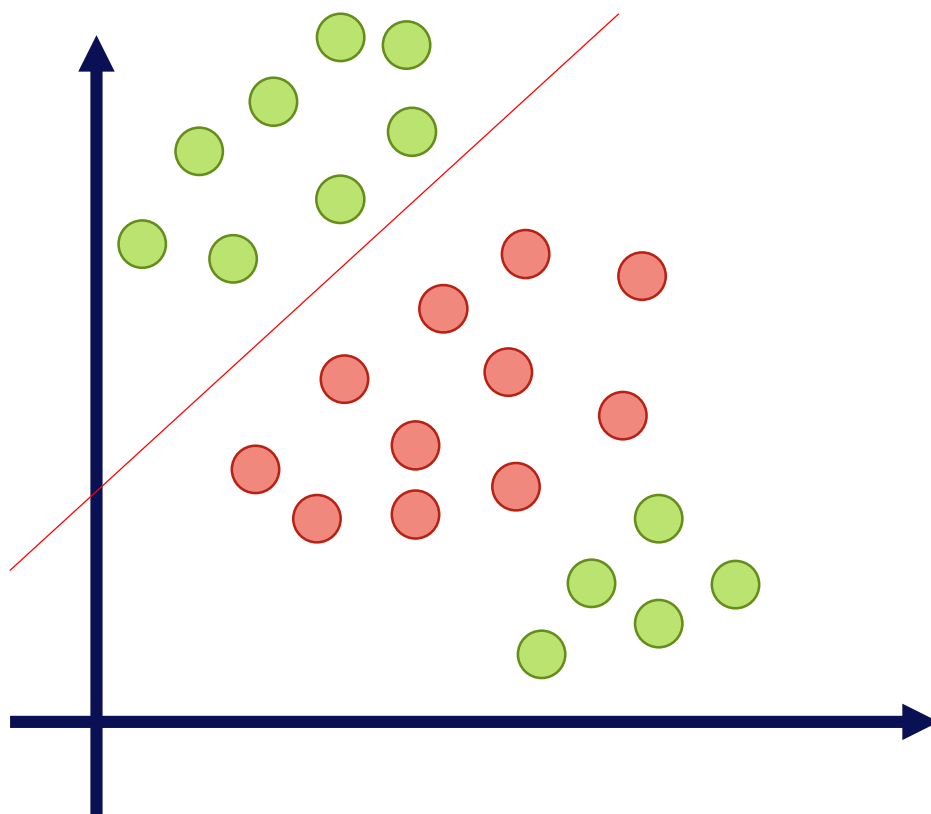
深層学習

2006年：オートエンコーダー

第3次AIブーム:
深層学習
2006年－

パーセプトロンの限界

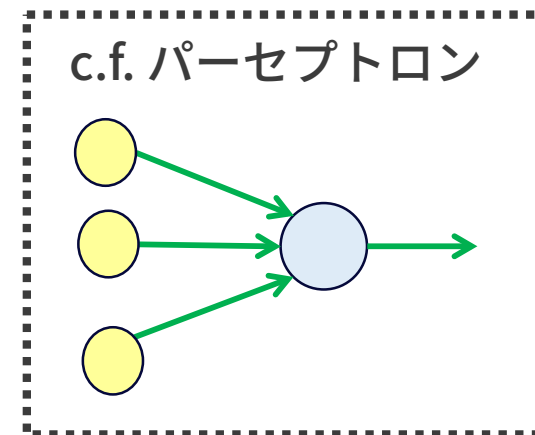
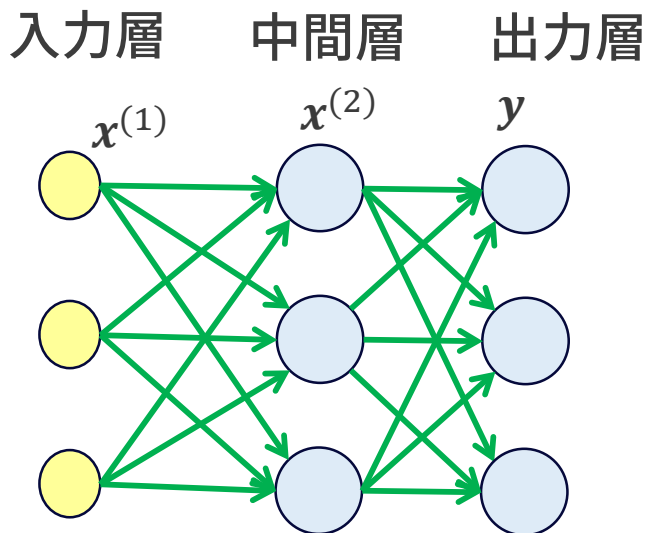
直線で切れられないような分離はできない




多層パーセプトロン (1/2)

(Multilayer perceptron:MLP) [Rumelhart+ 86]

パーセプトロンを多段に積むことでより複雑な曲線・曲面で分離できるようにする



ネットワークの要素  や  をノードと呼ぶ

数式的には行列を使って以下のように書ける

$$\mathbf{x}^{(2)} = \sigma(\mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)})$$

$$\mathbf{y} = \sigma(\mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)})$$

入力ベクトル： $\mathbf{x}^{(1)}$

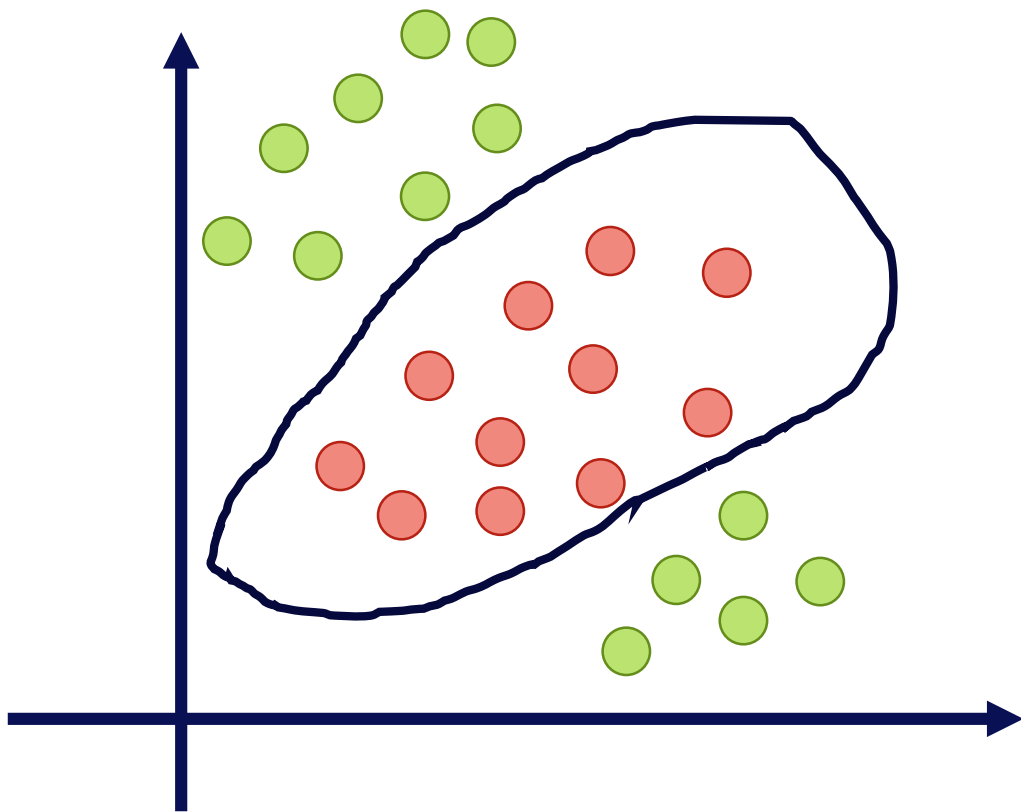
重み： $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}$

出力ベクトル： \mathbf{y}

活性化関数： $\sigma(x)$

多層パーセプトロン

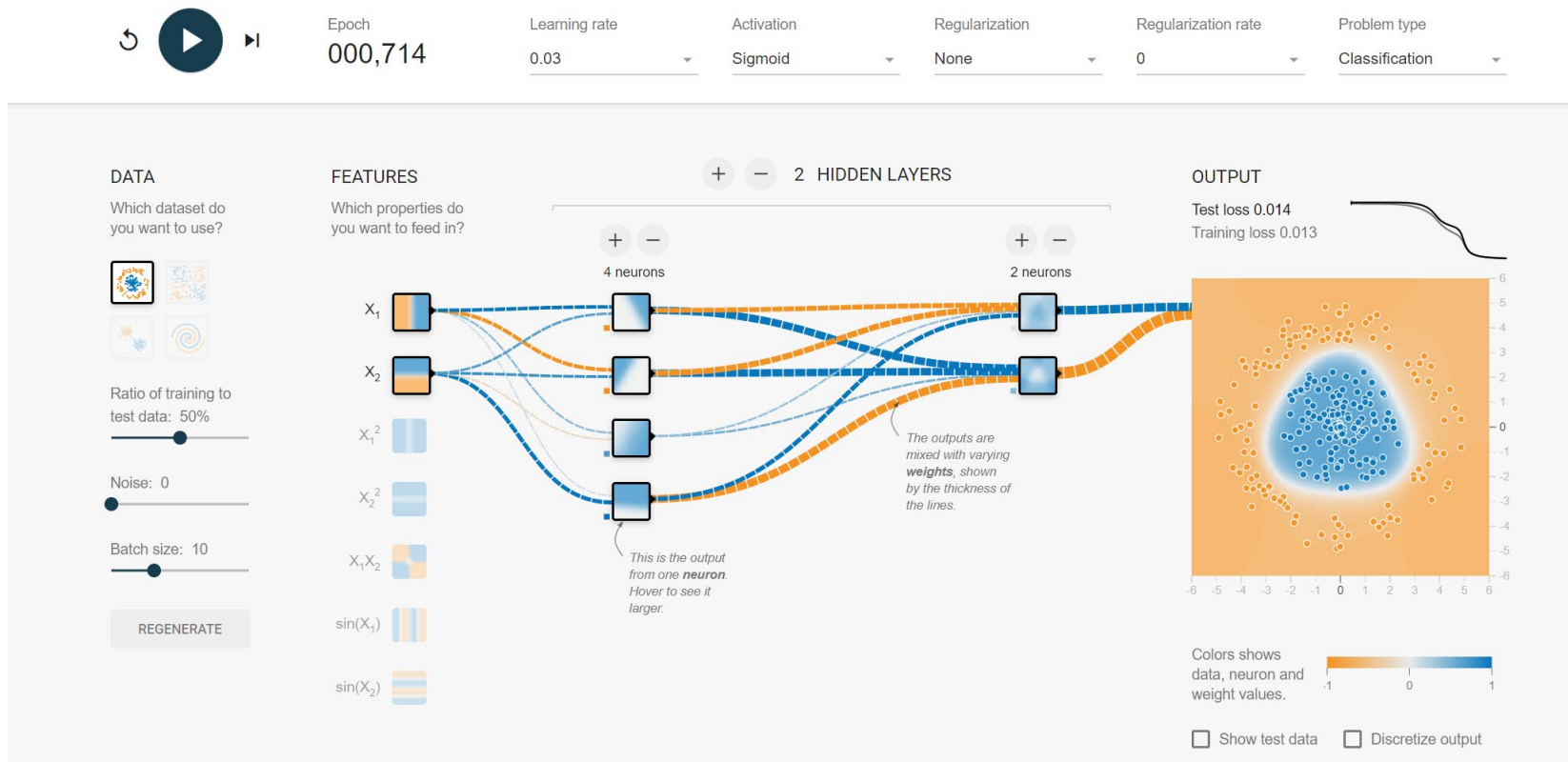
多層パーセプトロンは中間層のノード数を増やせば複雑な関数も作れるので下のような境界を作ることができる



実際に回してみることができるサイト

A neural network playground

<https://playground.tensorflow.org/>



多層パーセプトロンの学習(1/4)

誤差逆伝播法(Back propagation) [Rumelhart+ 86]

$$\min_{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{W}^{(3)}, \mathbf{b}^{(3)}} \frac{1}{2} e^2$$

- $e = \hat{y} - y$
- \hat{y} : 正解データ

$$x^{(2)} = \sigma(\mathbf{W}^{(1)}x^{(1)} + \mathbf{b}^{(1)})$$

$$x^{(3)} = \sigma(\mathbf{W}^{(2)}x^{(2)} + \mathbf{b}^{(2)})$$

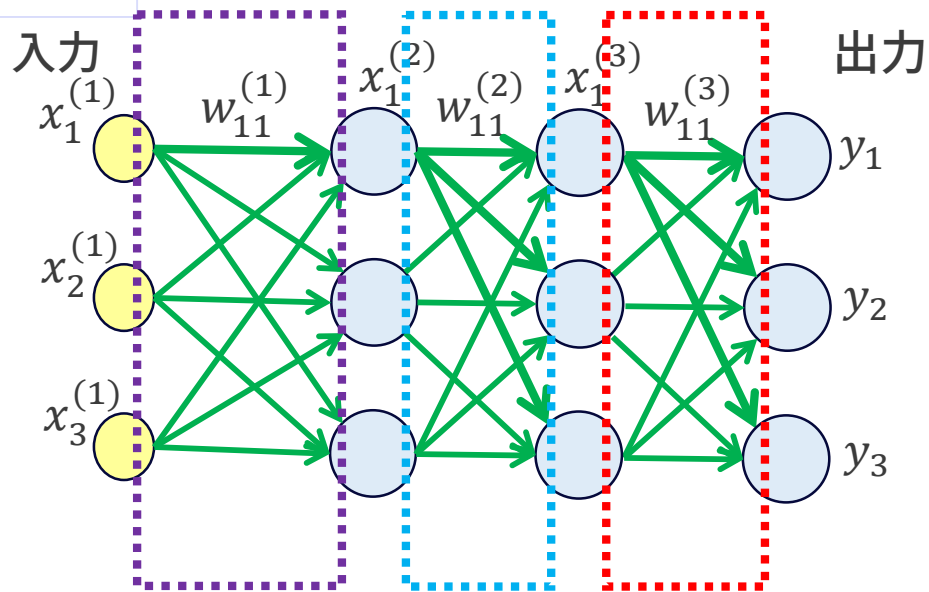
$$y = \sigma(\mathbf{W}^{(3)}x^{(3)} + \mathbf{b}^{(3)})$$

勾配降下法：

$$w_{11}^{(3)} \leftarrow w_{11}^{(3)} + \alpha \frac{1}{2} \frac{\partial e^2}{\partial w_{11}^{(3)}}$$

$$w_{11}^{(2)} \leftarrow w_{11}^{(2)} + \alpha \frac{1}{2} \frac{\partial e^2}{\partial w_{11}^{(2)}}$$

$$w_{11}^{(1)} \leftarrow w_{11}^{(1)} + \alpha \frac{1}{2} \frac{\partial e^2}{\partial w_{11}^{(1)}}$$



w_{11} は W の一行一列目の要素、
それ以外の要素も同様にアップデート可能

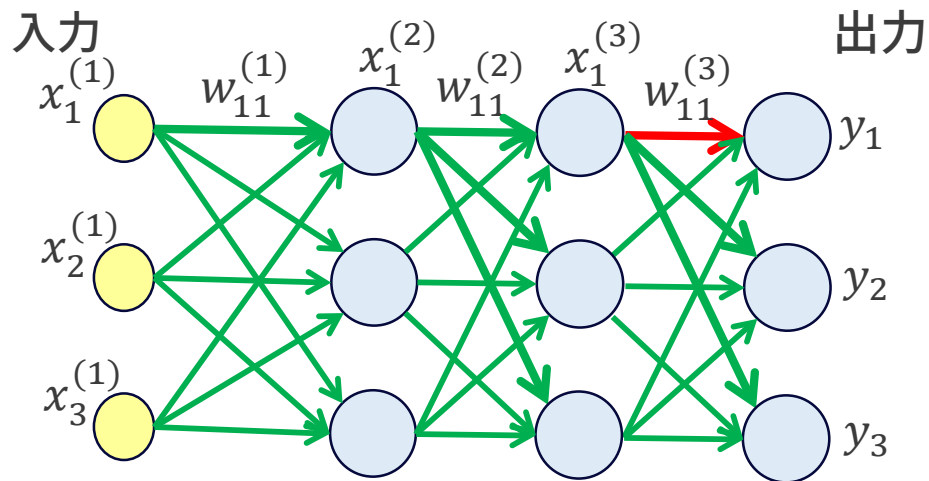
多層パーセプトロンの学習(2/4)

誤差逆伝播法(Back propagation) [Rumelhart+ 86]

$$\min_{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{W}^{(3)}, \mathbf{b}^{(3)}} \frac{1}{2} e^2$$

- $e = \hat{y} - y$
- \hat{y} : 正解データ

$$\frac{\partial e_1^2}{\partial w_{11}^{(3)}} = e_1 \frac{\partial y_1}{\partial w_{11}^{(3)}}$$



$$\mathbf{x}^{(2)} = \sigma(\mathbf{W}^{(1)} \mathbf{x}^{(1)} + \mathbf{b}^{(1)})$$

$$\mathbf{x}^{(3)} = \sigma(\mathbf{W}^{(2)} \mathbf{x}^{(2)} + \mathbf{b}^{(2)})$$

$$\mathbf{y} = \sigma(\mathbf{W}^{(3)} \mathbf{x}^{(3)} + \mathbf{b}^{(3)})$$

多層パーセプトロンの学習(3/4)

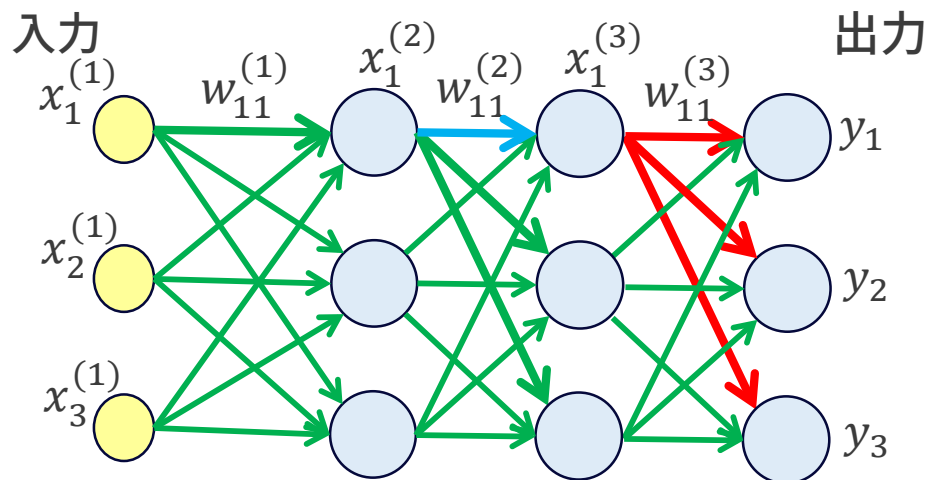
誤差逆伝播法(Back propagation) [Rumelhart+ 86]

$$\min_{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{W}^{(3)}, \mathbf{b}^{(3)}} \frac{1}{2} e^2$$

- $e = \hat{y} - y$
- \hat{y} : 正解データ

$$\frac{\partial e_1^2}{\partial w_{11}^{(3)}} = e_1 \frac{\partial y_1}{\partial w_{11}^{(3)}}$$

$$\frac{\partial e^2}{\partial w_{11}^{(2)}} = \left(\sum_i e_i \frac{\partial y_i}{\partial x_1^{(3)}} \right) \frac{\partial x_1^{(3)}}{\partial w_{11}^{(2)}}$$



$$\mathbf{x}^{(2)} = \sigma(\mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)})$$

$$\mathbf{x}^{(3)} = \sigma(\mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)})$$

$$\mathbf{y} = \sigma(\mathbf{W}^{(3)}\mathbf{x}^{(3)} + \mathbf{b}^{(3)})$$

多層パーセプトロンの学習(4/4)

誤差逆伝播法(Back propagation) [Rumelhart+ 86]

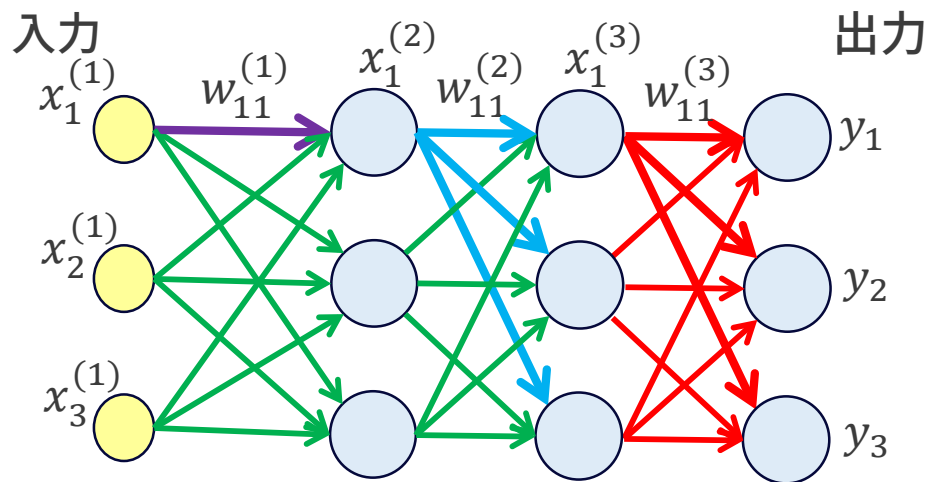
$$\min_{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{W}^{(3)}, \mathbf{b}^{(3)}} \frac{1}{2} e^2$$

- $e = \hat{y} - y$
- \hat{y} : 正解データ

$$\frac{\partial e_1^2}{\partial w_{11}^{(3)}} = e_1 \frac{\partial y_1}{w_{11}^{(3)}}$$

$$\frac{\partial e^2}{\partial w_{11}^{(2)}} = \left(\sum_i e_i \frac{\partial y_i}{\partial x_1^{(3)}} \right) \frac{\partial x_1^{(3)}}{\partial w_{11}^{(2)}}$$

$$\frac{\partial e^2}{\partial w_{11}^{(1)}} = \left(\sum_j \left(\sum_i e_i \frac{\partial y_i}{\partial x_j^{(3)}} \right) \frac{\partial x_j^{(3)}}{\partial x_1^{(2)}} \right) \frac{\partial x_1^{(2)}}{\partial w_{11}^{(1)}}$$



$$\begin{aligned} \mathbf{x}^{(2)} &= \sigma(\mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)}) \\ \mathbf{x}^{(3)} &= \sigma(\mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)}) \\ \mathbf{y} &= \sigma(\mathbf{W}^{(3)}\mathbf{x}^{(3)} + \mathbf{b}^{(3)}) \end{aligned}$$

ニューラルネットワークのUniversality

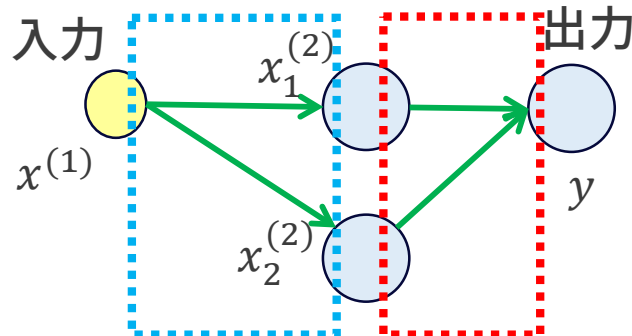
Universality : 任意の連続関数を十分近似できるか [Cybenko+ 1989]

入出力1次元、中間層2次元の
ニューラルネットワークを考える

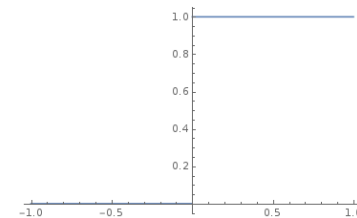
$$x_1^{(2)} = \sigma(w_1^{(1)}x^{(1)} + b_1^{(1)})$$

$$x_2^{(2)} = \sigma(w_2^{(1)}x^{(1)} + b_2^{(1)})$$

$$y = w_1^{(2)}x_1^{(2)} + w_2^{(2)}x_2^{(2)} + b^{(2)}$$



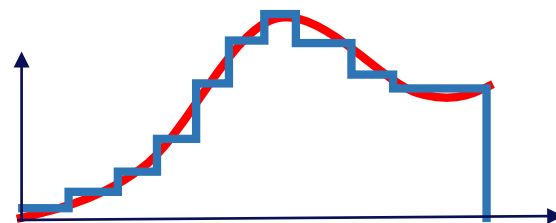
活性化関数 $\sigma(x)$
ステップ関数



$$w_1^{(2)} = 0.5, w_2^{(2)} = 0.5, b^{(2)} = -0.5$$

中間層2個
ノードがあ
れば短冊が
作れる

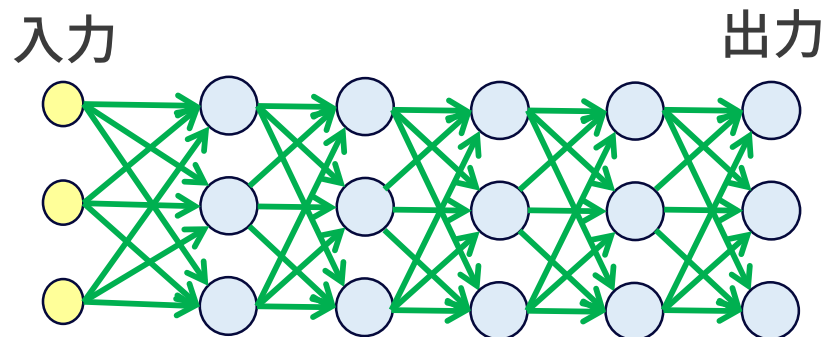
たくさん中間層のノードがあればwとbを
うまく設定すれば任意の関数を近似できる



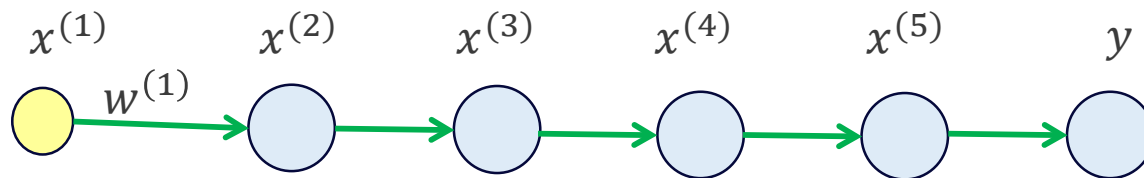
ニューラルネットワークの多層化

3層以上のネットワークであっても
同様に誤差逆伝播法で学習可能

- 階層化した方がより少ないノード数で関数を近似することができる



勾配消失問題



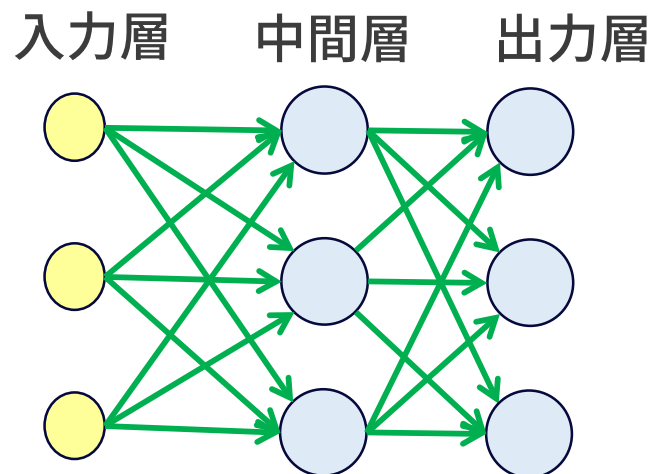
$$\frac{\partial e^2}{\partial w^{(1)}} = e \frac{\partial y}{\partial x^{(5)}} \frac{\partial x^{(5)}}{\partial x^{(4)}} \frac{\partial x^{(4)}}{\partial x^{(3)}} \frac{\partial x^{(3)}}{\partial x^{(2)}} \frac{\partial x^{(2)}}{\partial w^{(1)}}$$

$$\frac{\partial x^{(\ell)}}{\partial x^{(\ell-1)}} = \sigma'(x^{(\ell)}) w^{(\ell-1)} \text{は通常 } 1 \text{ より小さい}$$

シグモイド関数の微分 < 0.25

ここまでのニューラルネットワーク

- 多層パーセプトロン自体は様々な関数近似に利用することができるが、よく用いられるのは識別（教師あり学習）
 - 入力：画像（SIFT,SURFなどの特徴量）
 - 出力：ラベル
- 勾配消失問題のため3層（入力層・中間層・出力層）の多層パーセプトロンが用いられていた（頑張れば中間層2層の4層も学習できた）
- 他の機械学習手法に比べ性能はそれほど良くなかった



人工知能の歴史

計算と人工知能

人工知能

1950年：チューリングテスト
1956年：ダートマス会議

論理・探索

第1次AIブーム: 推論と探索
1956年－1974年

1972年：Prolog

第2次AIブーム:
知識
(エキスパートシステム)
1980年－1987年

機械学習 機械学習

1967年：k-means法
(クラスタリング)

1989年：Q学習
(強化学習)

1992年：非線形SVM
(教師あり学習)

1990年～：統計的機械学習
・確率モデル
・汎化誤差理論
・ベイズモデル

ニューラルネットワーク

1958年：パーセプトロン

1969年：パーセプトロンの限界

1986年：誤差逆伝搬法

深層学習

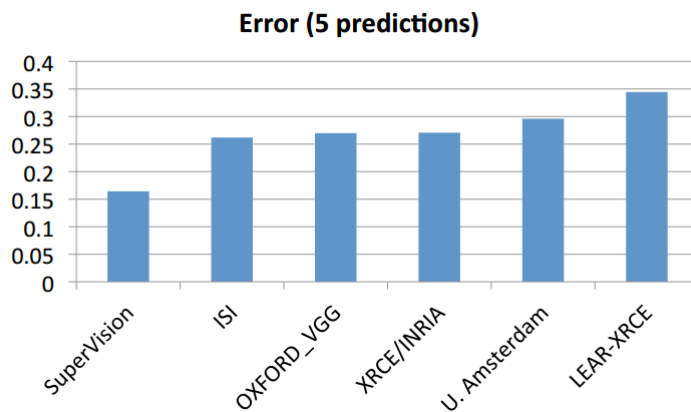
2006年：オートエンコーダー

第3次AIブーム:
深層学習
2006年－

深層学習の勃興(1/2)

RBM(restricted Boltzmann machine) を用いた”pre-training”が提案される
[Hinton06]

ILSVRC2012 (ImageNet Large Scale Visual Recognition Challenge) でHintonらのグループが優勝



2位とのエラー率の差10%以上の差をつけて性能を示した！
(識別エラー率16%)

深層学習の勃興(2/2)

- 音声認識、化合物の活性予測でもDNNが最高精度を達成！
- 自然言語処理では特に機械翻訳で高精度を達成

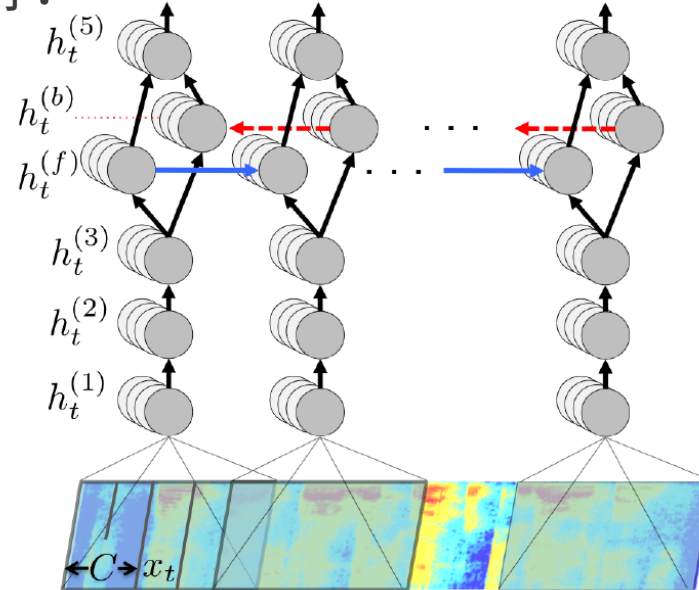
主にデータが豊富にある分野で大きな成功を収めている

(基本的に深層学習はデータ数が多い時に有効)

特に、**end-to-end** (入力から欲しいものを直接出力するような構成方法) 型のニューラルネットワークが高精度 (従来は様々な前処理等を行っていたが、それらなしで高精度を出した)

音声認識の例：

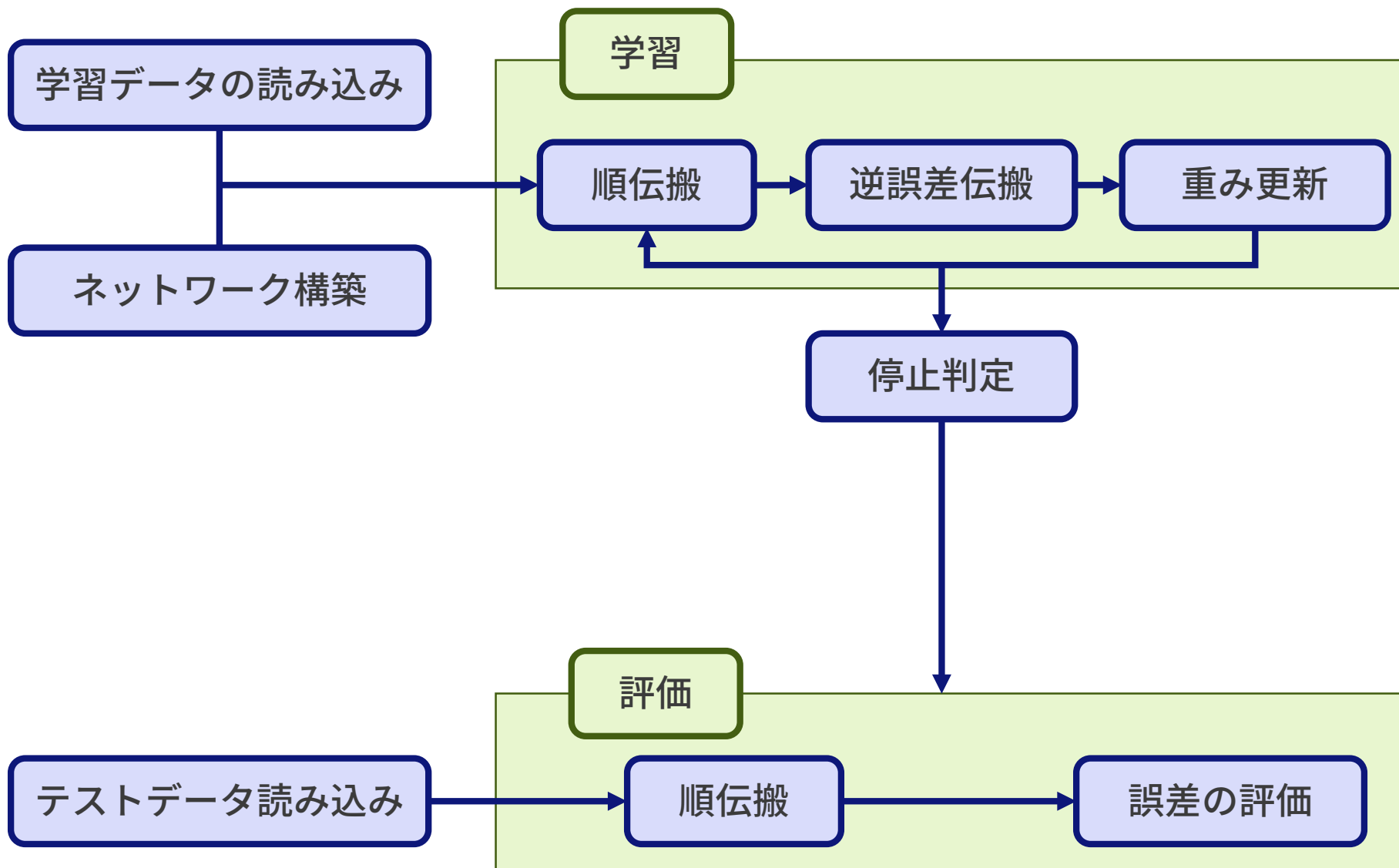
出力：文字



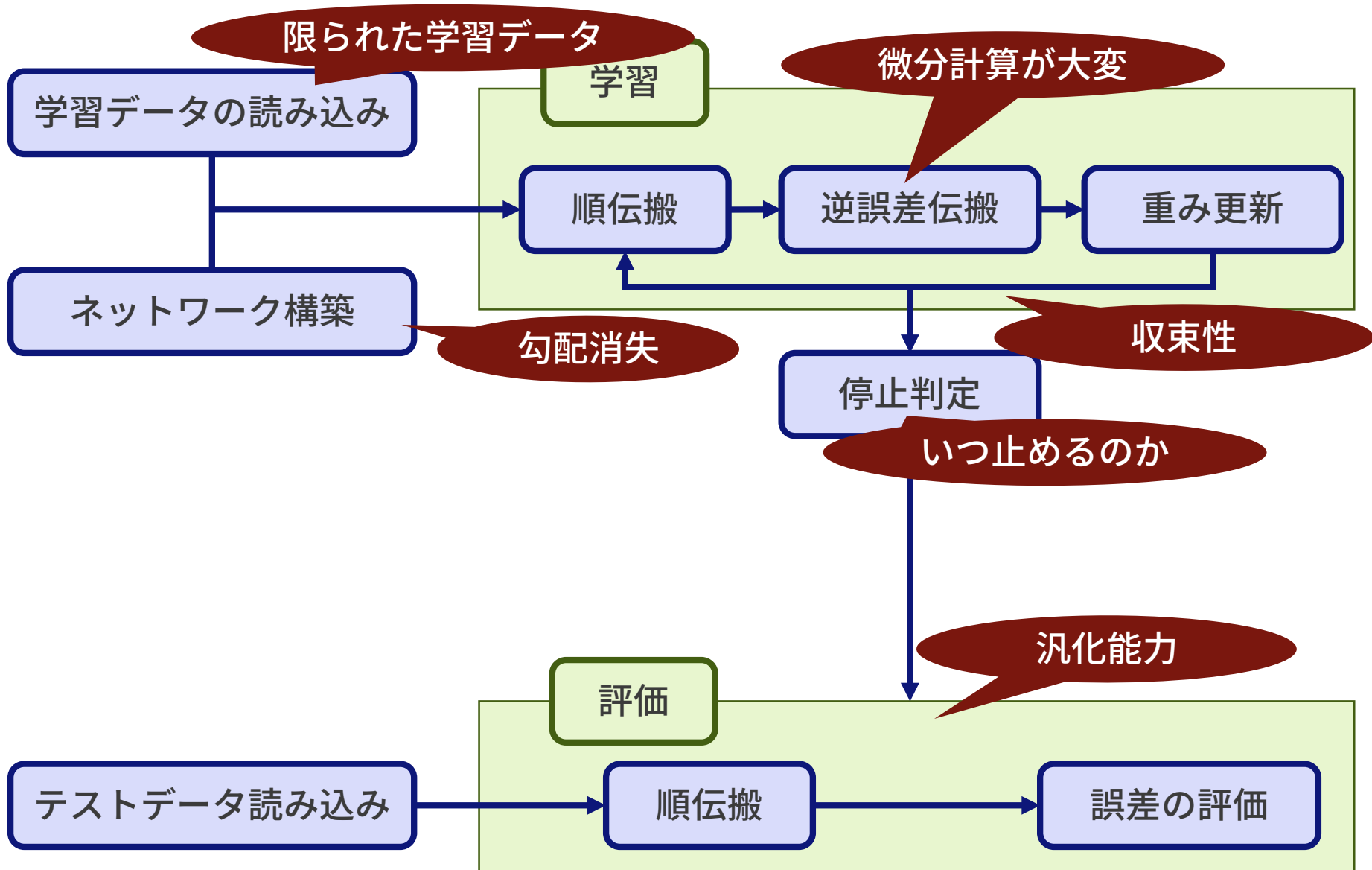
入力：音声スペクトログラム

Deep Speech [Hannun+ 14]

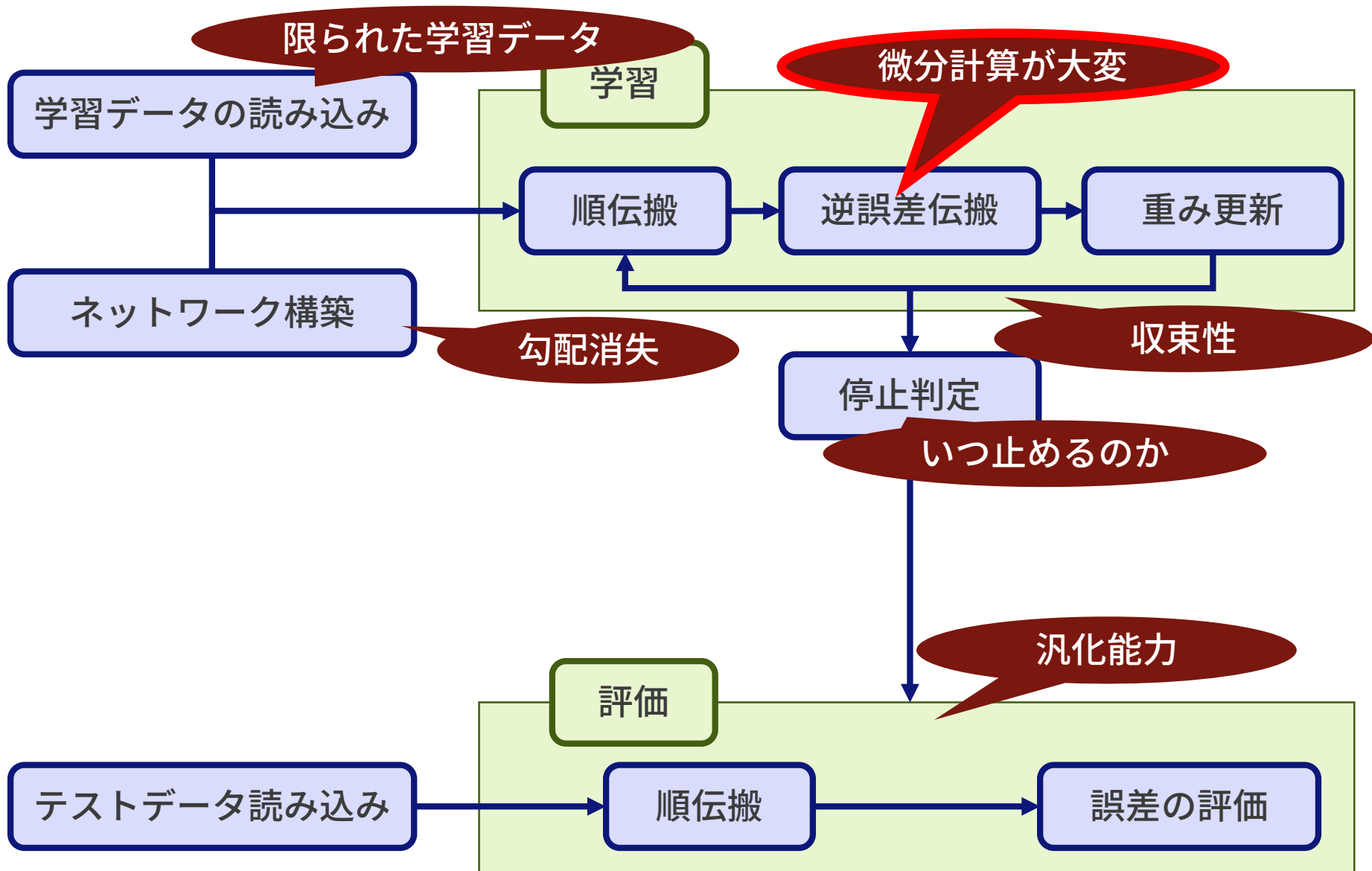
ニューラルネットワーク学習・評価の流れ



ニューラルネットワーク学習・評価の流れ



ニューラルネットワーク学習・評価の流れ

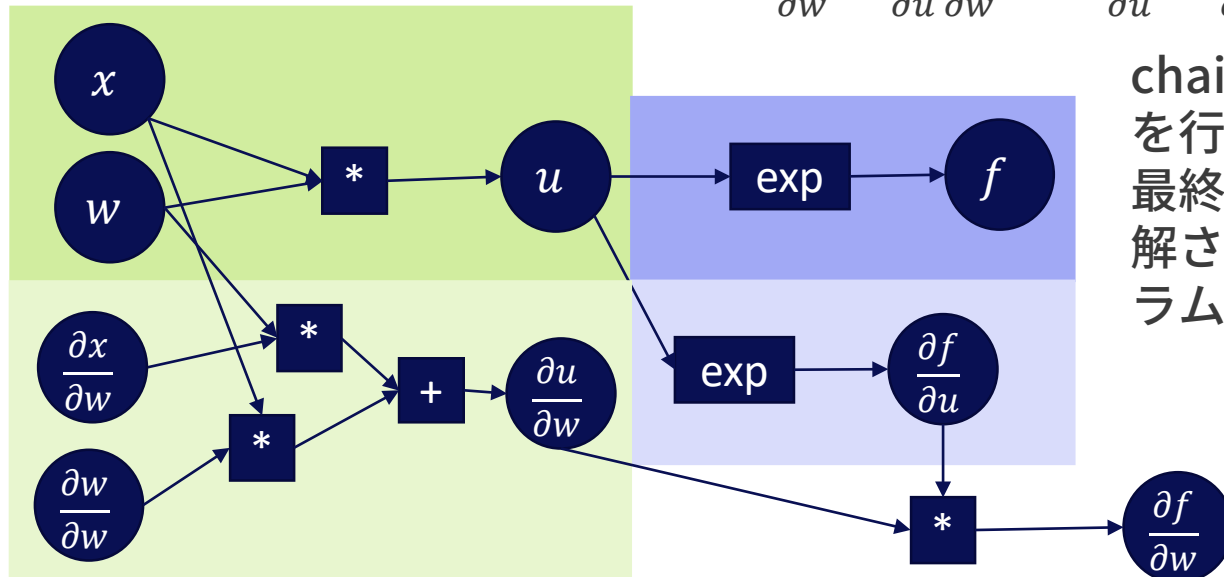


自動微分(Automatic differentiation)

多くの深層学習ライブラリはプログラムを実行することで、計算グラフと呼ばれるものを構築し、その後その計算グラフ上で微分計算を行う (reverse mode)

例: $f(x, w) = \exp(xw)$

を w に関して偏微分する $\frac{\partial f}{\partial w} = \frac{\partial f}{\partial u} \frac{\partial u}{\partial w} = \frac{\partial \exp(u)}{\partial u} \frac{\partial wx}{\partial w}$



chain rule を用いて微分計算を行う
最終的に基本的な計算まで分解されて、あらかじめプログラムされた微分計算になる

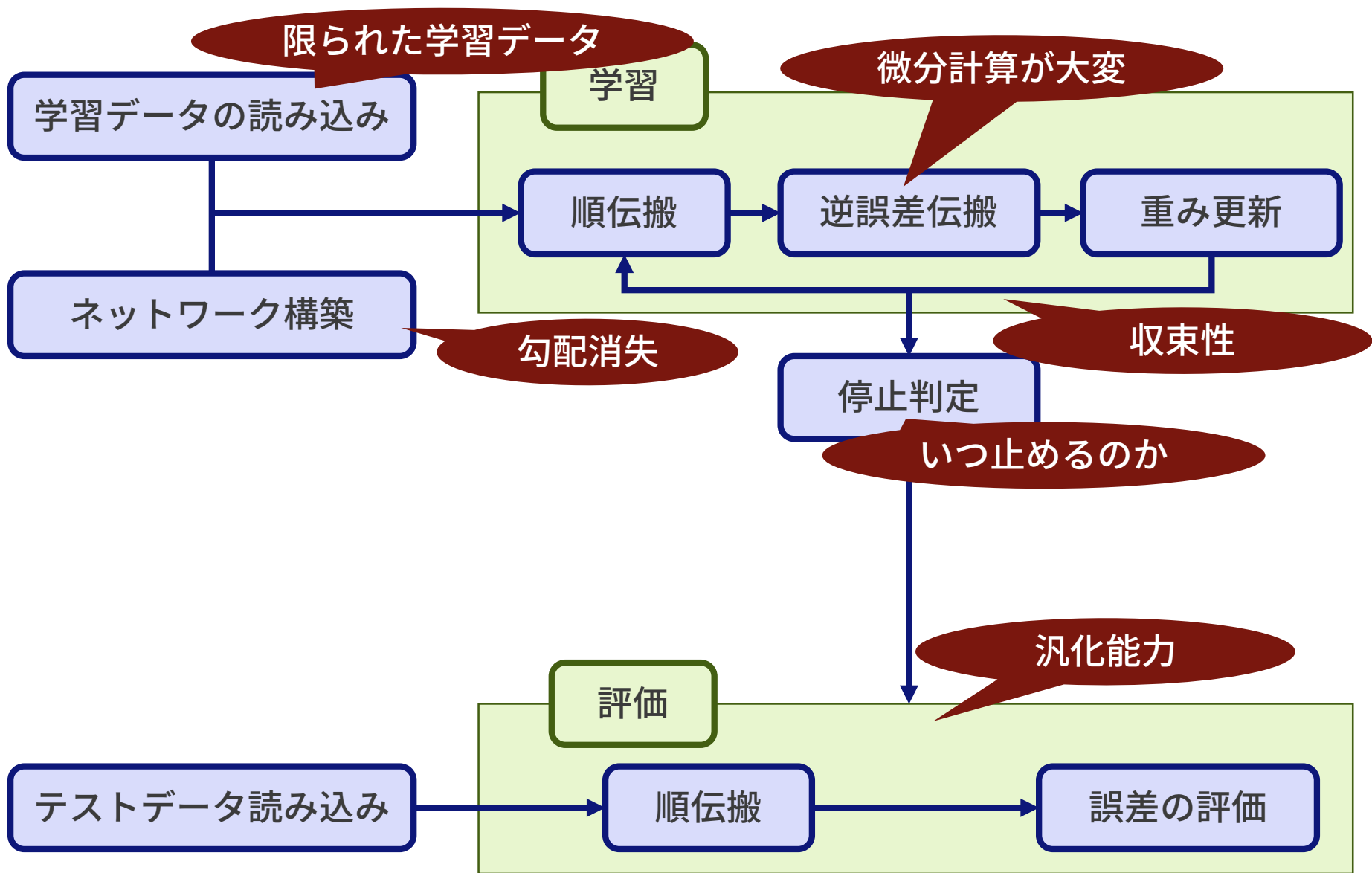
各種ライブラリ

- Tensorflow (<https://www.tensorflow.org/>)
- Keras (<https://keras.io/>)
- pyTorch (<https://pytorch.org/>)
- chainer (<https://docs.chainer.org/>)

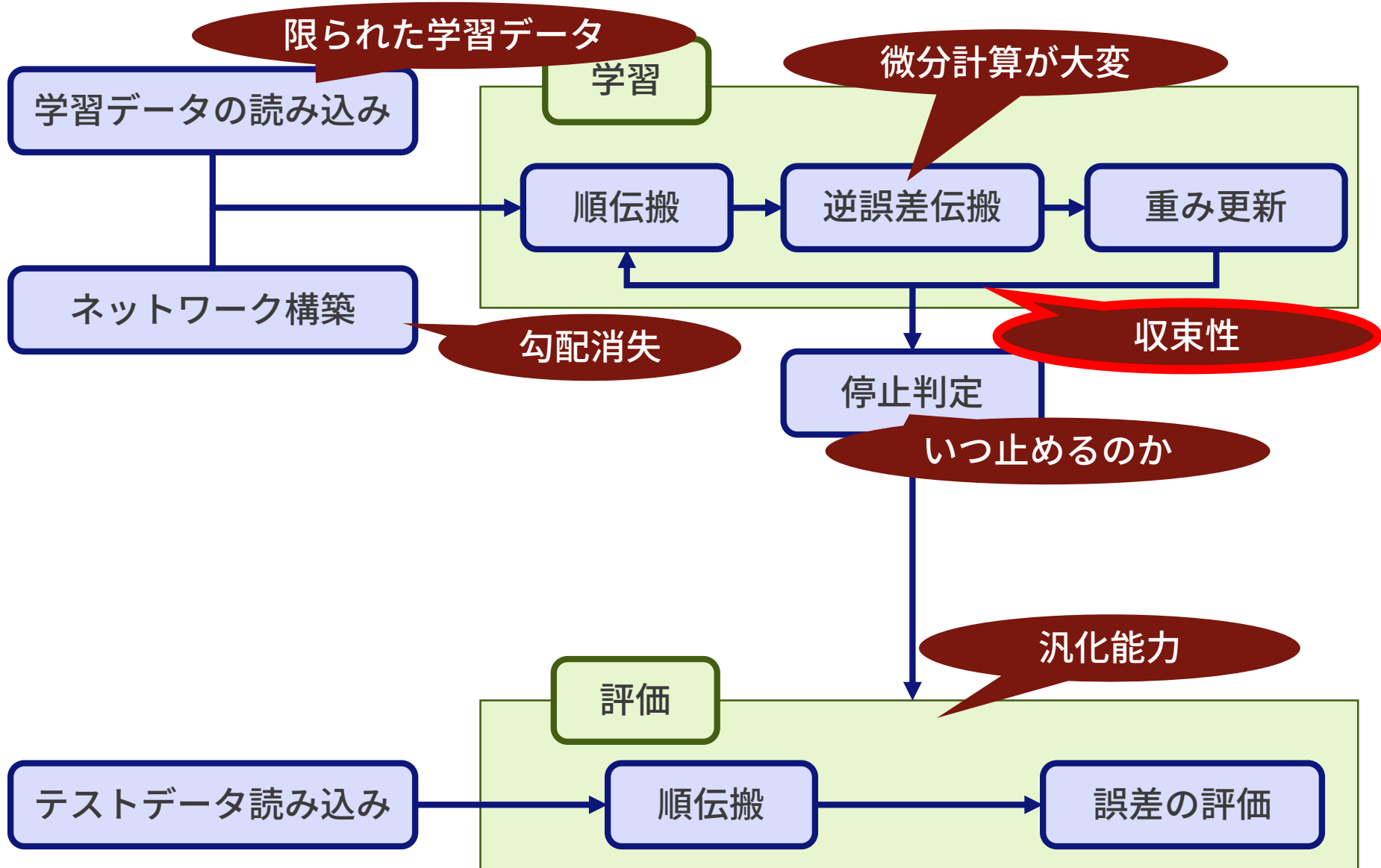
関連技術

- Numerical differentiation
微小変化を数値的に計算する
- Symbolic differentiation
数式を直接操作して微分を計算する

ニューラルネットワーク学習・評価の流れ



ニューラルネットワーク学習・評価の流れ



確率的勾配降下法(Stochastic Gradient Decent : SGD)

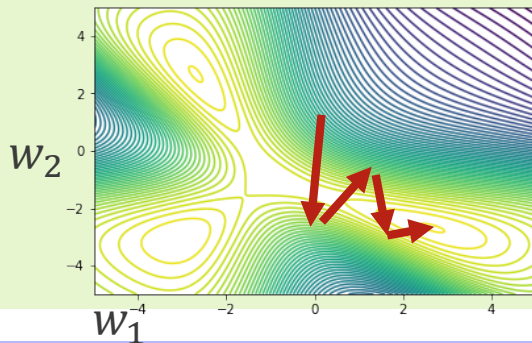
データ数 D 時のコスト関数 :

$$J = \sum_{d=1}^D J_d \quad \text{二乗誤差の場合は、} J_d = e^2$$

確率的勾配降下法(SGD)

$$w \leftarrow w + \alpha \nabla J_d$$

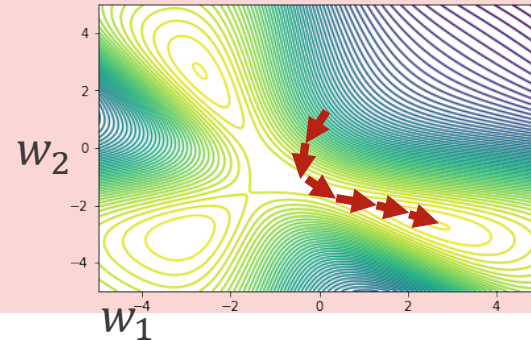
データを1つずつ使って更新



勾配降下法

$$w \leftarrow w + \alpha \nabla J$$

データをすべて使って更新



実際は、間をとって、データを分割して m 個ずつ使う ミニバッチ法 を使うことが多い

さらに、Adam や AdaDelta といった適応的に α を決める方法も併用される

最適化手法

- Momentum SGD † * † †
- Nesterov Accelerate Gradient (1983) †
- Rprop(1992)+
- L-BFGS(1995)+
- ProximalAdagrad(2009) †
- ProximalGD(2009) †
- AdaGrad (2011) † * † †
- AdaGrad+DualAveraging(2010) †
- FTRL(2013) †
- RMSProp (2012) † * † †
- AdaDelta (2012) † * † †
- Adam (2014) † * † †
- RMSpropGraves (2014) †

- SMORMS3 (2015) †
- AdaMax (2015)*+
- Nadam (2016)*
- Eve (2016)
- Santa (2016)
- GD by GD (2016)
- AdaSecant (2017)

各種ライブラリの対応状況

† tensorflow v1.13

https://www.tensorflow.org/api_docs/python/tf/train/
(tensorflow probability 以下の最適化は除く)

*keras

<https://keras.io/optimizers/>

+pyTorch v1.10

<https://pytorch.org/docs/stable/optim.html#algorithms>

‡ chainer v7.0

<https://docs.chainer.org/en/stable/reference/optimizers.html>

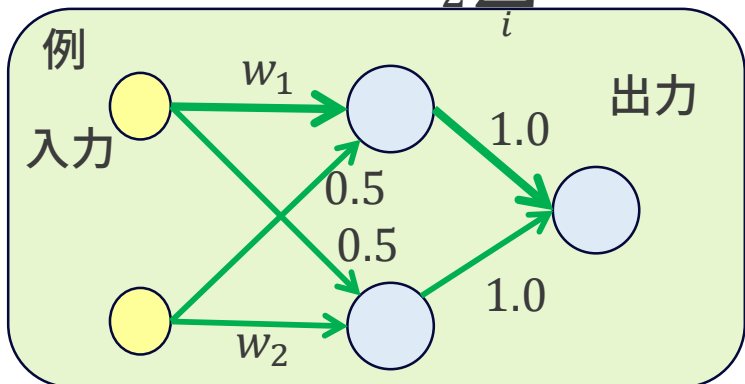
正則化 (reguralization)

ニューラルネットの出力の誤差に罰則化項を追加する

$$J = J_D + J_R$$

$$J_D = \frac{1}{2} \sum_i e_i^2$$

$$J_R = \lambda \|W\|^2$$



J_D

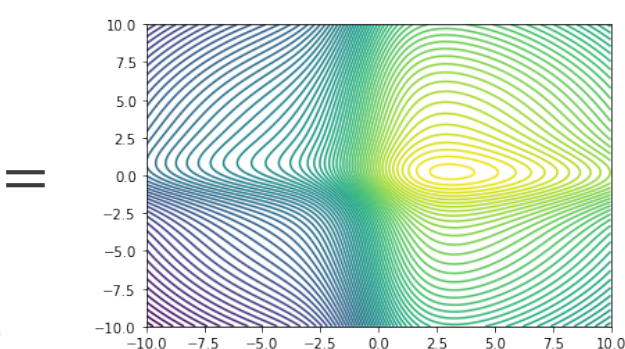
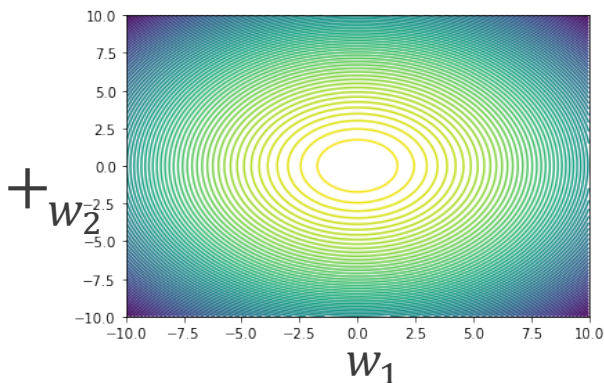
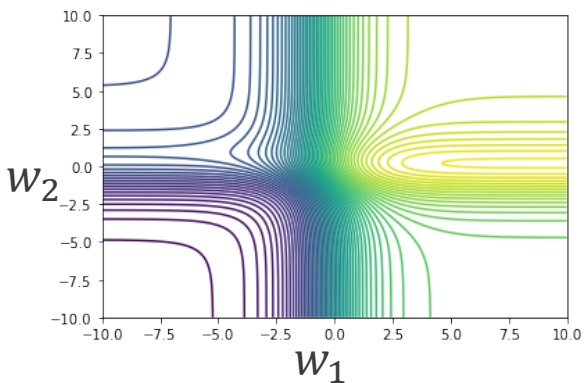
J_R

$e_i = \hat{y}_i - y_i$
 \hat{y}_i : i 番目のデータの正解
 y_i : i 番目のデータの出力

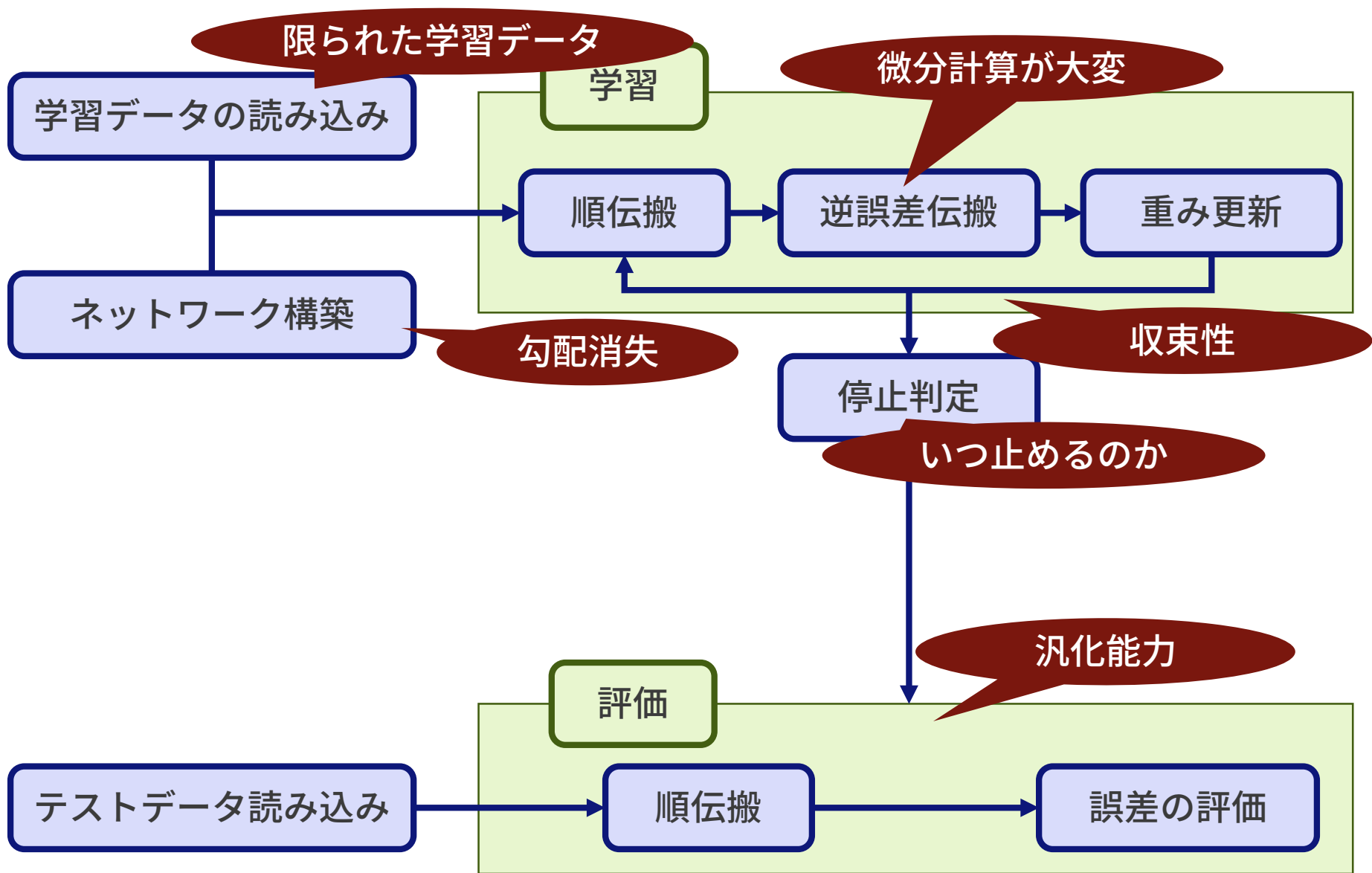
$$x^{(2)} = \sigma(W^{(1)}x^{(1)} + b^{(1)})$$

$$y = \sigma(W^{(2)}x^{(2)} + b^{(2)})$$

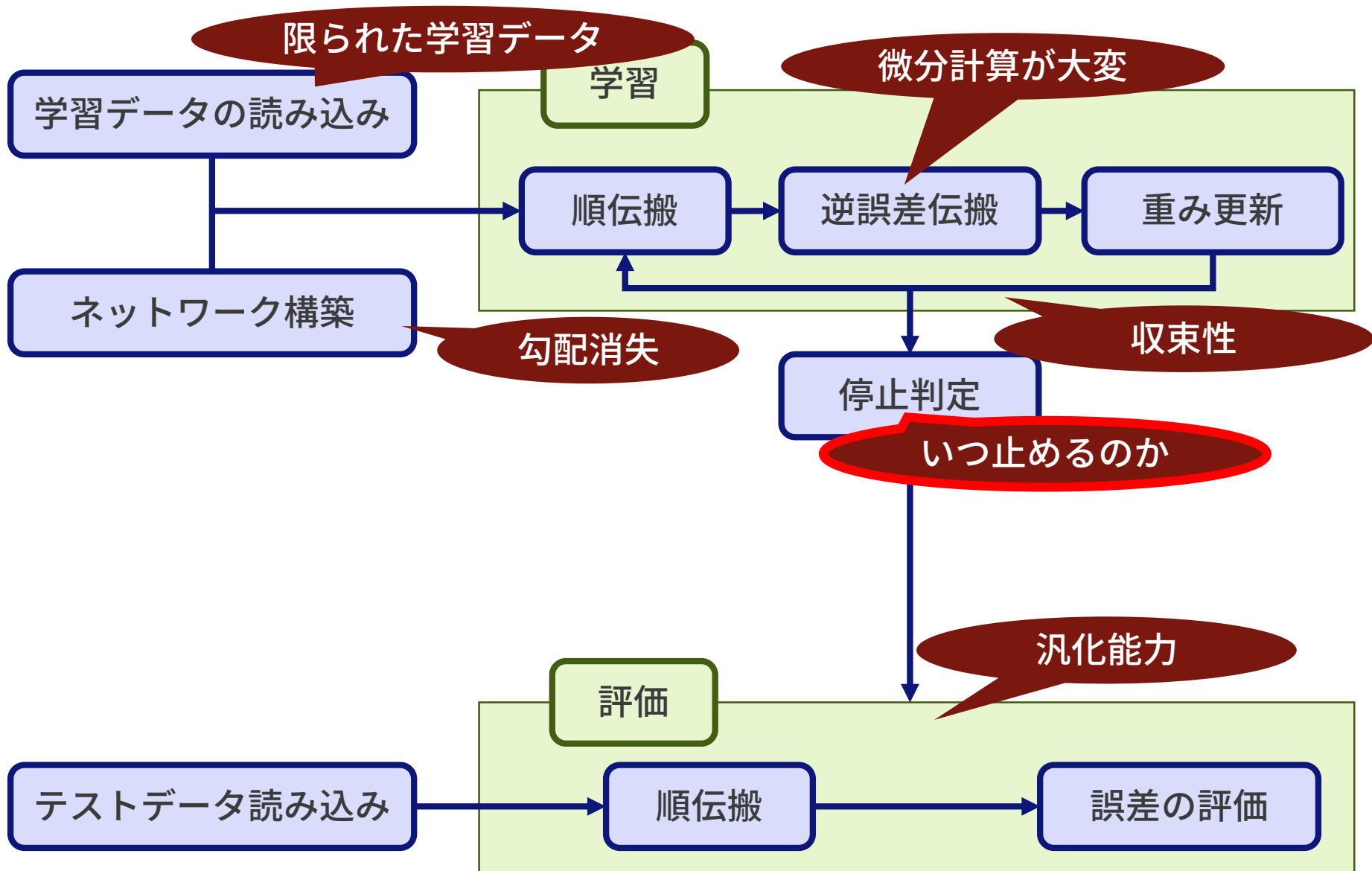
入力: $x^{(1)}$
 重み: $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$
 出力: y
 活性化関数: $\sigma(x)$



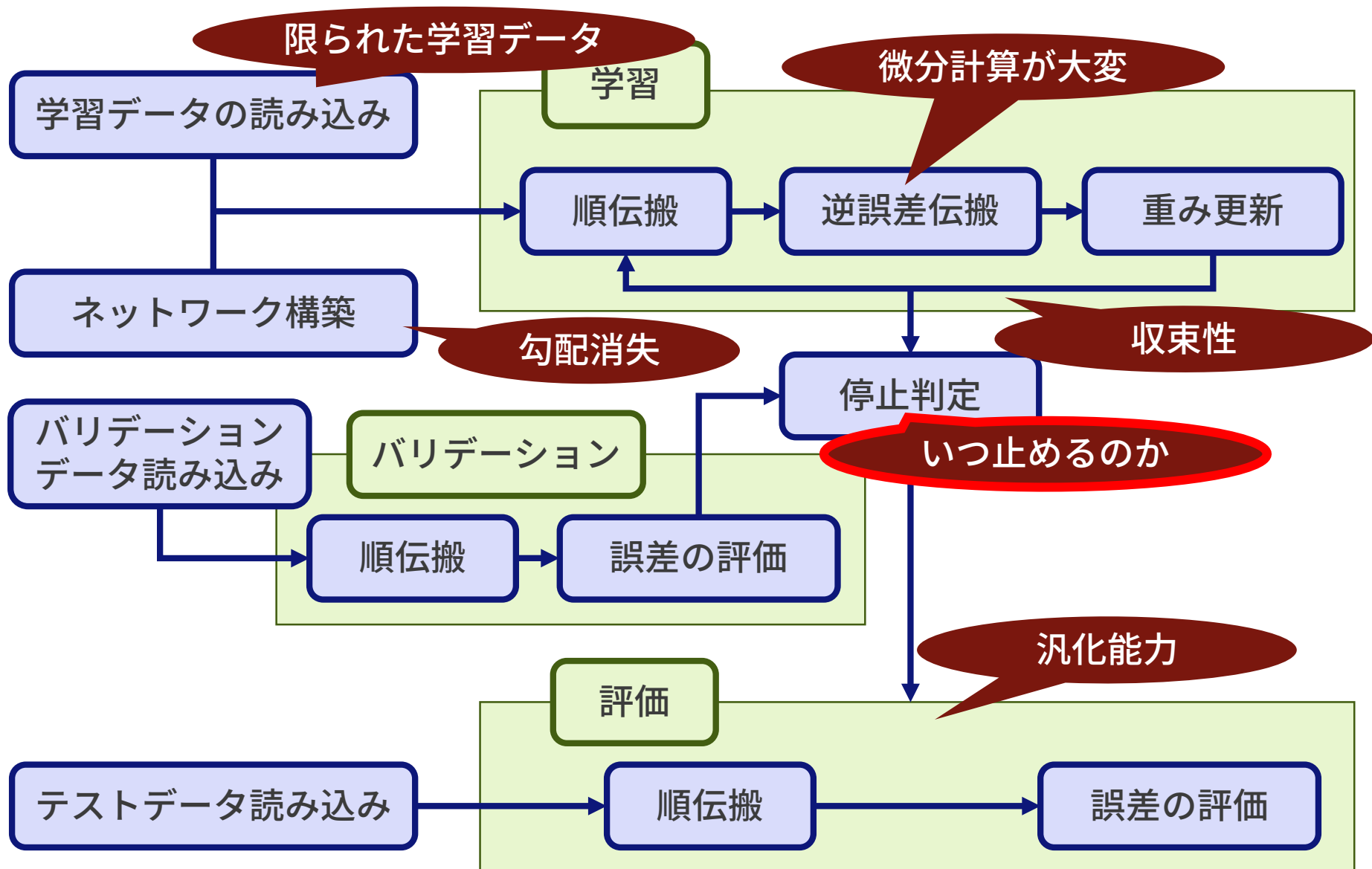
ニューラルネットワーク学習・評価の流れ



ニューラルネットワーク学習・評価の流れ



ニューラルネットワーク学習・評価の流れ



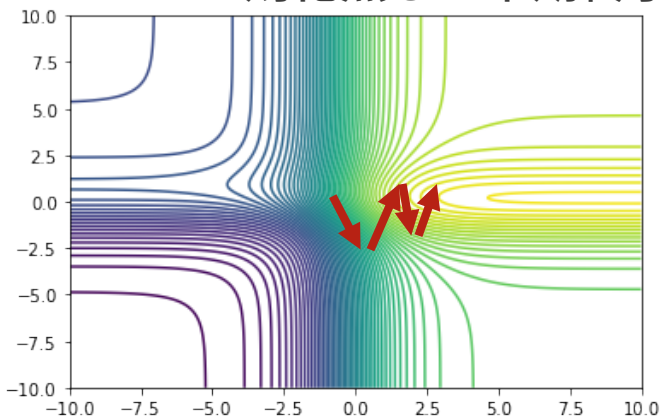
早期終了 (Early stopping)

バリデーションデータの誤差が下がらなくなった時点で学習を停止する

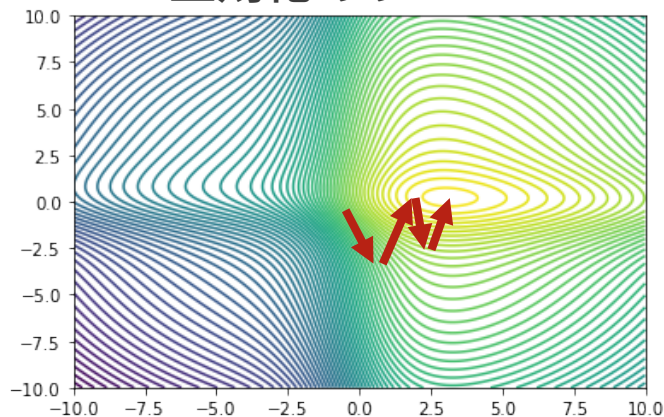
バリデーションデータは学習データから抜いて一部を使ったり、あらかじめ分けておいたりする

データを0付近で初期化していた場合、正則化と同様の効果が得られる

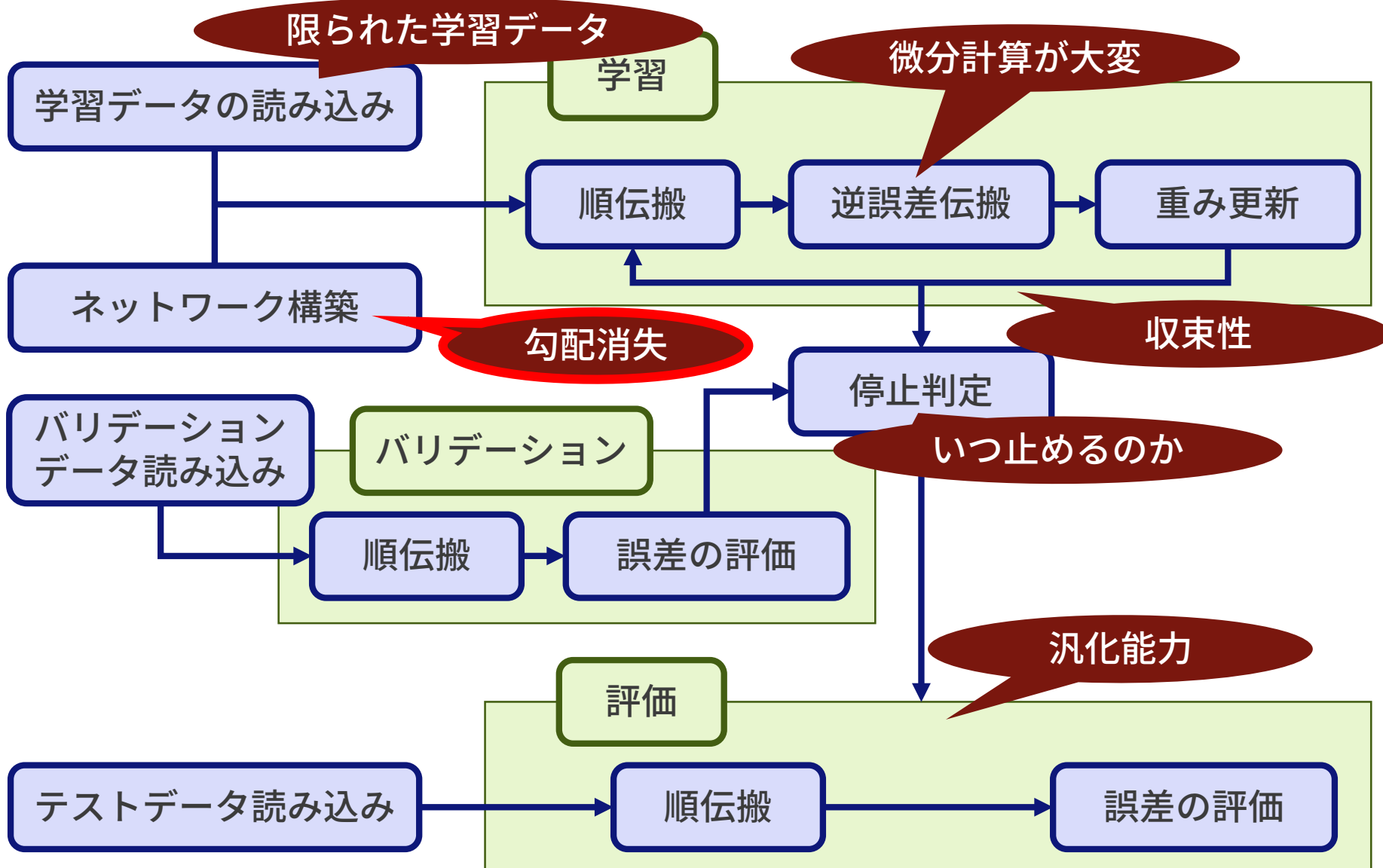
正則化無し+早期終了



正則化あり



ニューラルネットワーク学習・評価の流れ



活性化関数

Sigmoid関数

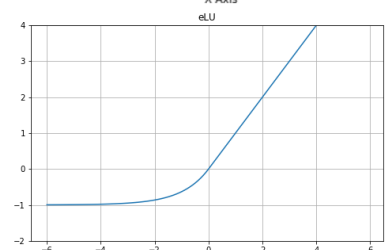
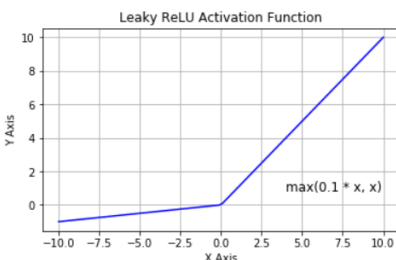
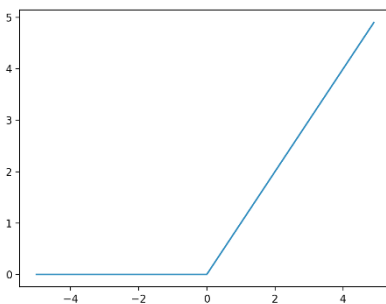
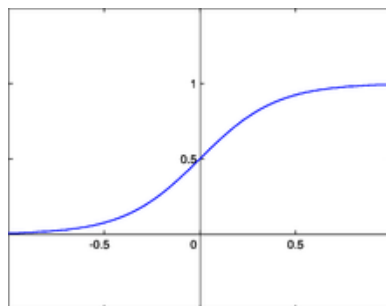
tanh関数：sigmoid関数を引き延ばしたものともみなせる

ReLU(Rectifier Linear Unit)関数

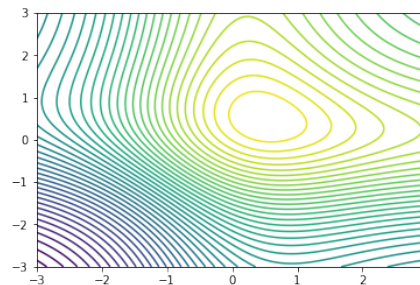
LeakyReLU関数

eLU関数

関数の形

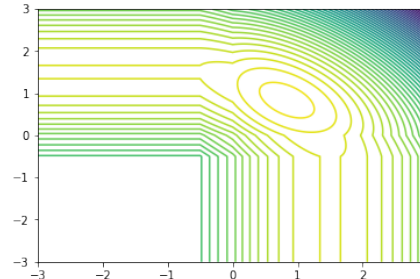


3層ニューラルネットワークの目的関数の形

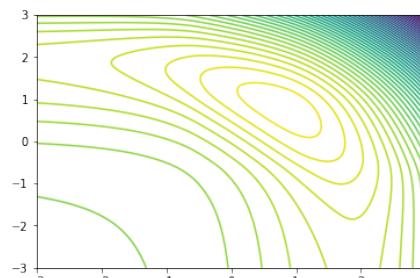
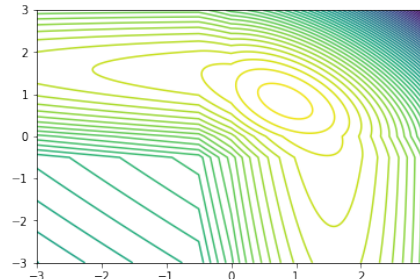


勾配がなだらか
(Sigmoid関数

の微分 <0.25) なので勾配消失の問題がある



勾配が0になってしまう場所がある (dead neurons)



*すべての活性化関数で最適地付近が円のように見える部分は二乗誤差を用いているため

活性化関数の選び方まとめ

- Sigmoid/tanhなどは勾配消失に注意
- Reluなどは中間層にのみ使う
- Reluなどは発散に注意（クリッピングが必要な場合もある）
- Reluは小規模なニューラルネットでは特にdead neuronsに注意：
LeakyReluなどを利用する

その他の活性化関数 (pytorch 1.9.0 の関数)

Hardshrink

Hardsigmoid

Hardtanh

Hardswish

LogSigmoid

MultiheadAttention

PReLU

RReLU

SELU

CELU

GELU

SiLU(swish)

Mish

Softplus

Softshrink

Softsign

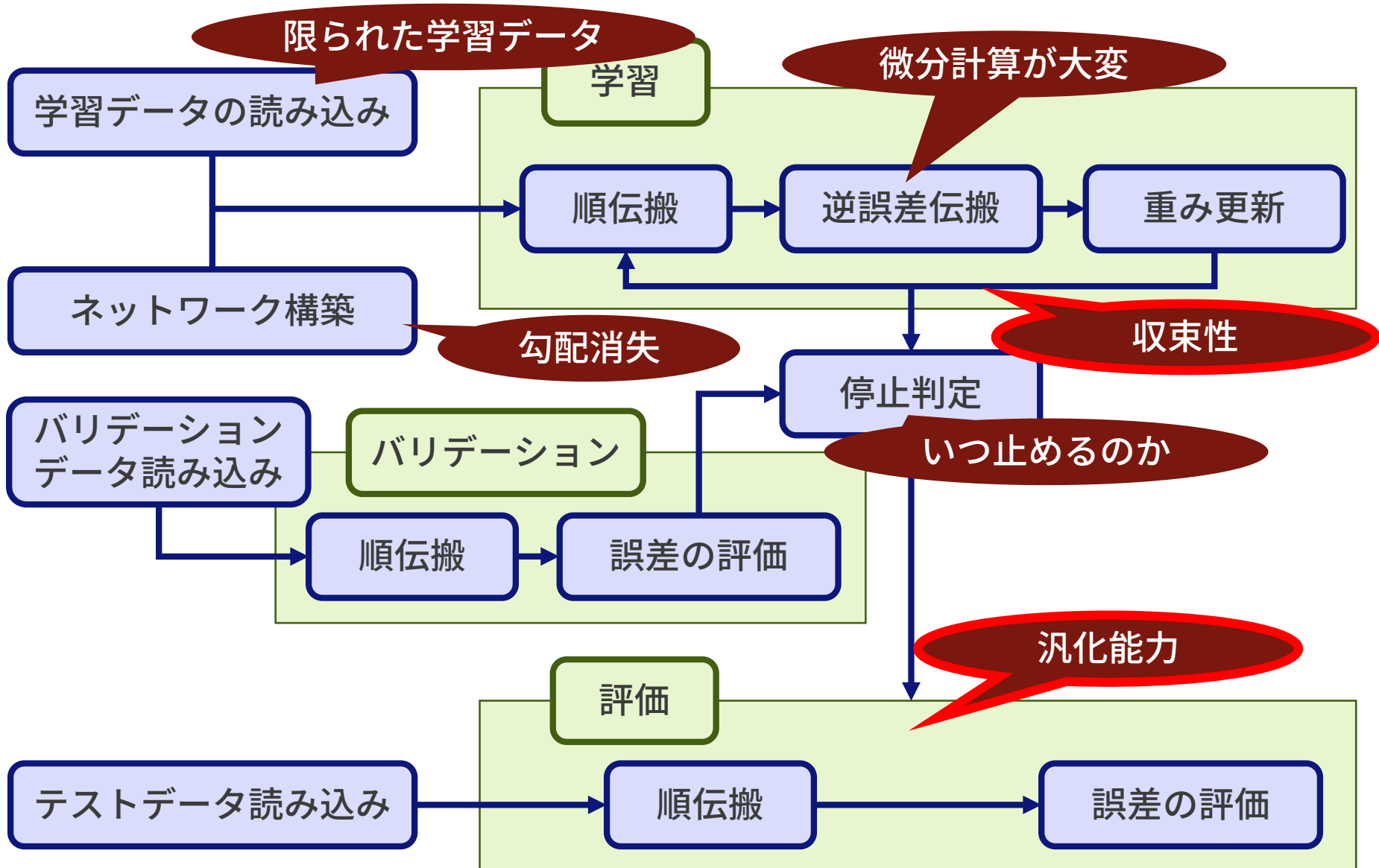
Tanhshrink

Threshold

Softmin

Softmax

ニューラルネットワーク学習・評価の流れ

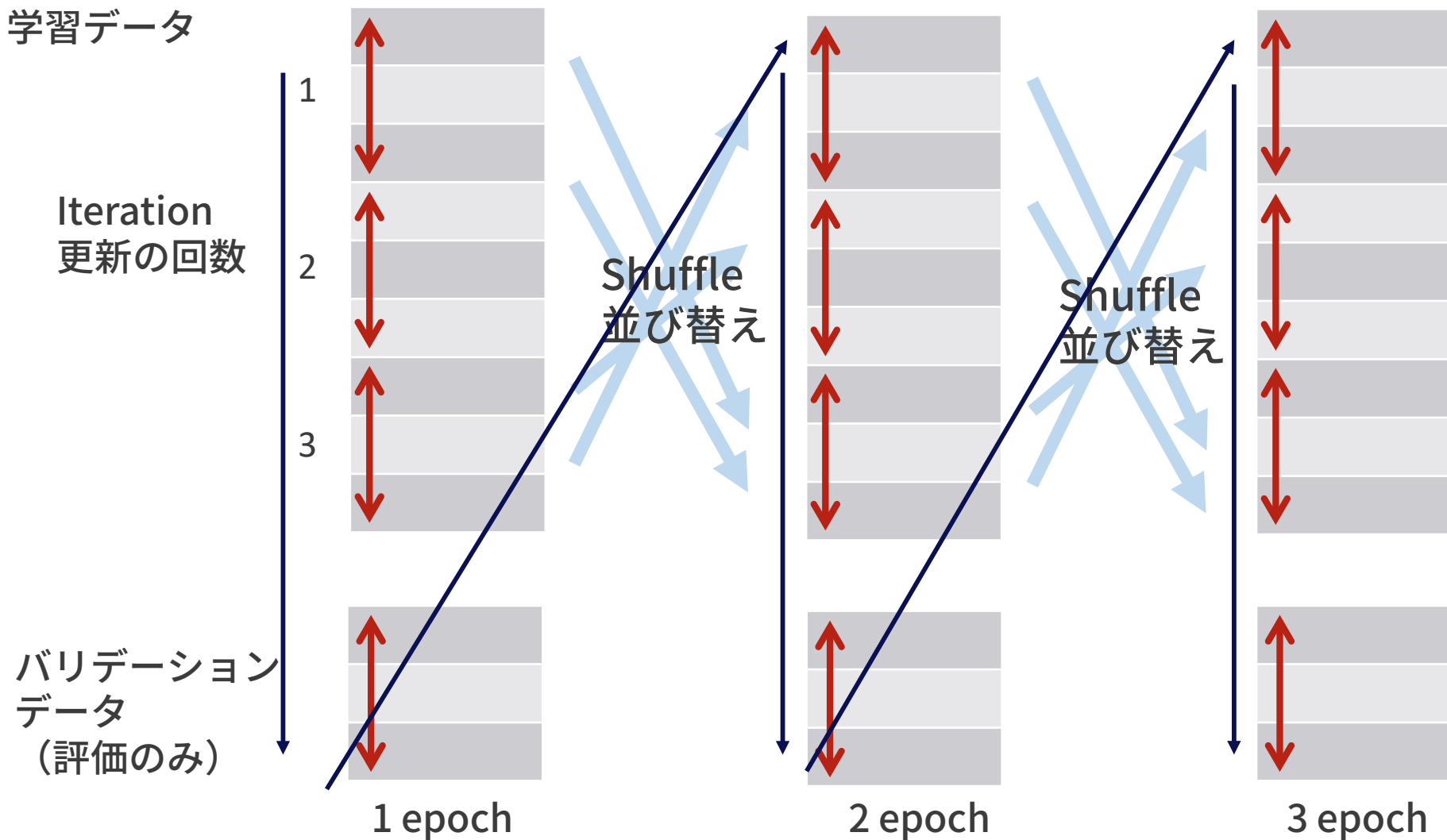


ミニバッチ法の詳細

ミニバッチ内の誤差

$$J = \sum_{d=1}^m J_d$$

m : バッチサイズ



epoch : データ全体を何回ループしたか

バッチノーマライゼーション

D 個のデータを m 個ごとのミニバッチに分割して、勾配を計算するミニバッチ法を使う場合に、ニューラルネットワークの中間層でデータをノーマライズする方法

前の層からの入力

$$\mathbf{x} = (x_1, x_2, \dots, x_i, \dots, x_m)$$



$$\begin{aligned} \mu &= \text{mean}(\mathbf{x}) & \hat{x}_i &= \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \\ \sigma^2 &= \text{var}(\mathbf{x}) & y_i &\leftarrow \gamma \hat{x}_i + \beta \\ i &= 1 \sim m \\ m &: \text{ミニバッチの内のデータ数} \end{aligned}$$

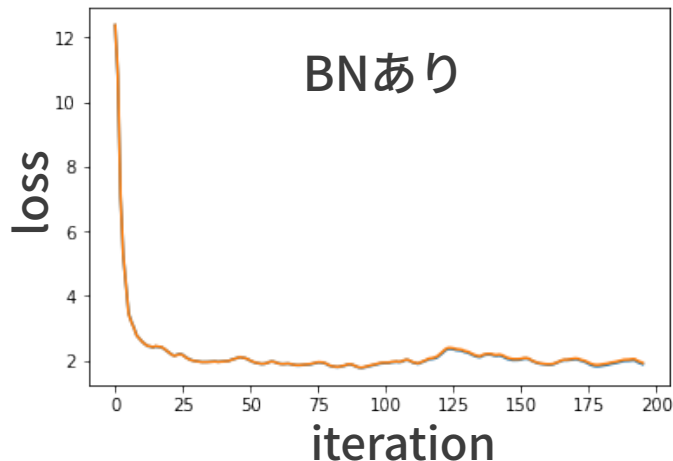
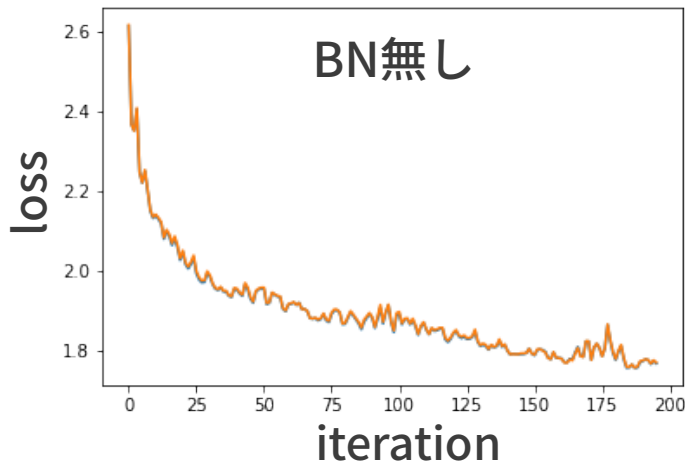
次の層への出力

$$\mathbf{y} = (y_1, y_2, \dots, y_i, \dots, y_m)$$

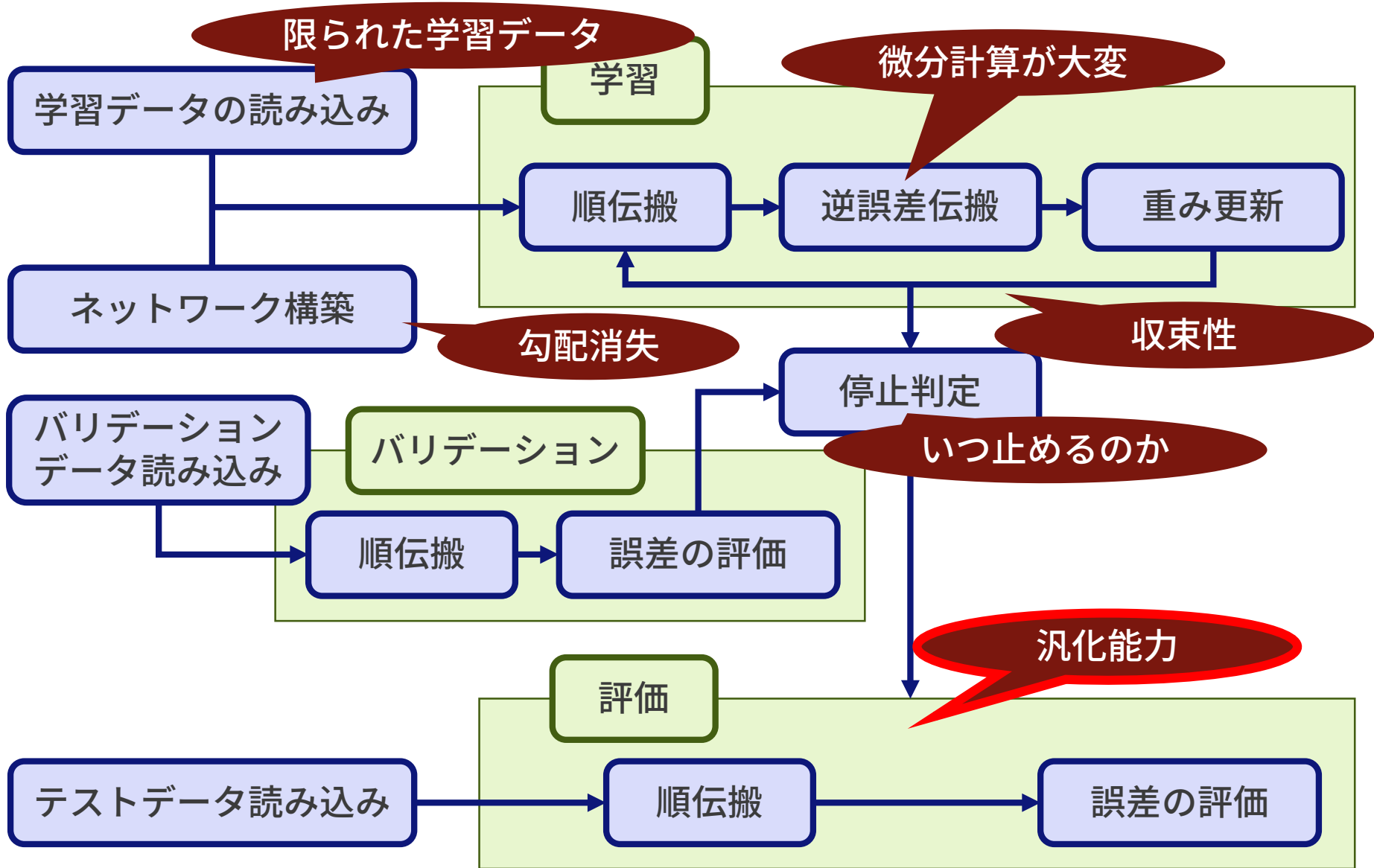


*テスト時には学習データ全体の平均・分散を $\mu \cdot \sigma^2$ として使うために移動平均を学習時に計算しておく

4層ニューラルネットワークの学習の例



ニューラルネットワーク学習・評価の流れ

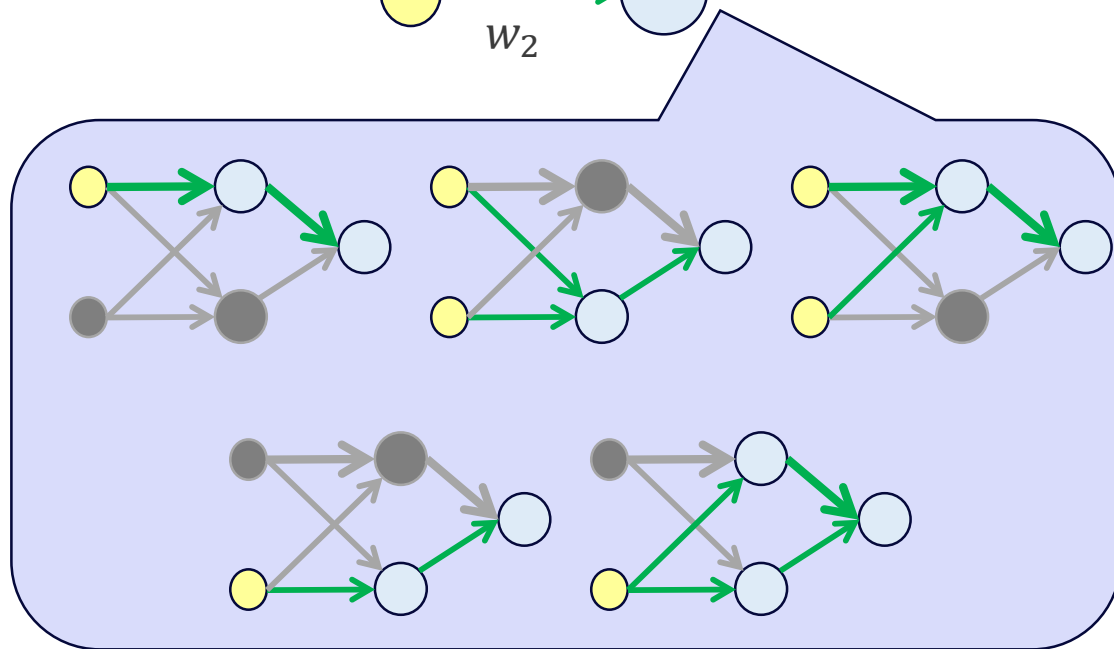
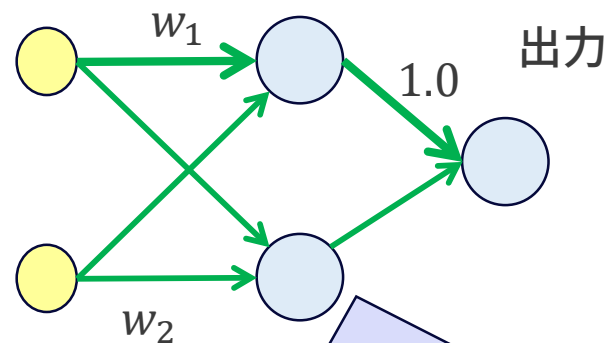


ドロップアウト

学習時にランダムにニューラルネットのノードを0にする

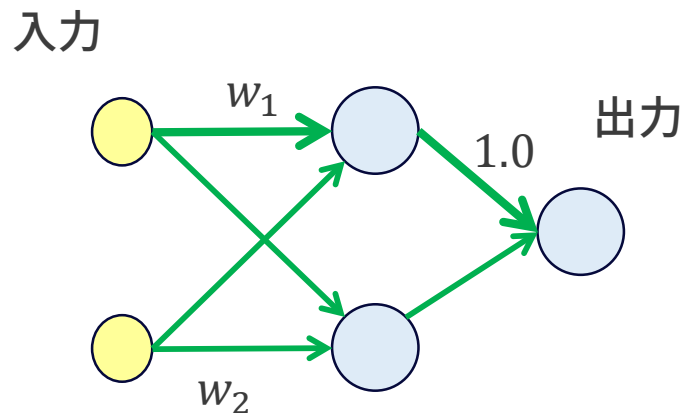
誤差逆伝搬時にも0になった部分は学習されない

入力



ドロップアウトの効果

ドロップアウトはある種のベイジアンニューラルネットワークにおける推論と等価である



ドロップアウト無し : $y = \sigma(\mathbf{W}x + \mathbf{b})$

ドロップアウトあり : $y = \sigma(\mathbf{W}(x \odot \boldsymbol{\mu}) + \mathbf{b})$

$\boldsymbol{\mu}$: binary vector

$$y = \sigma(\hat{\mathbf{W}}x + \mathbf{b})$$

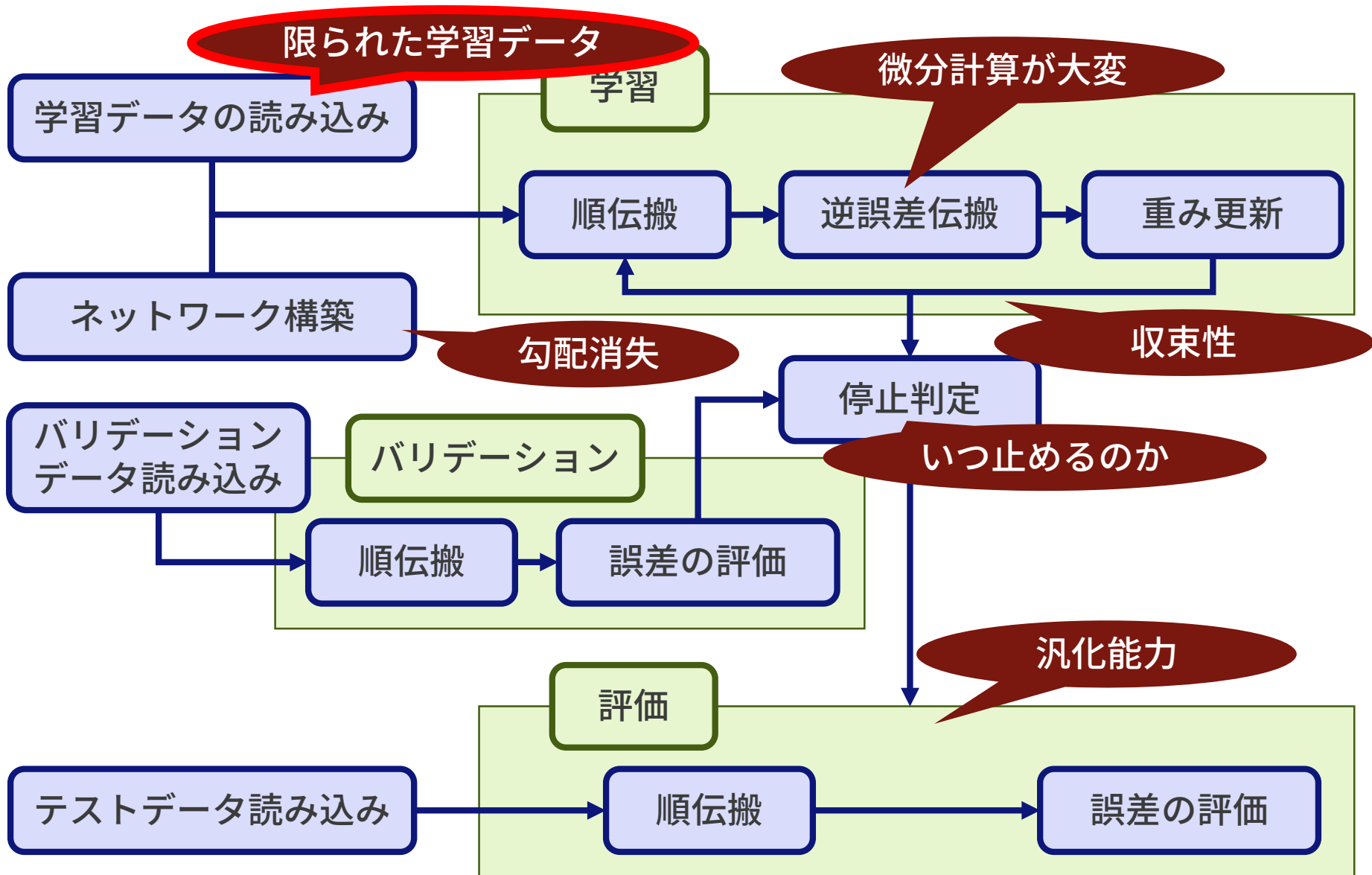
$$\hat{\mathbf{W}} = \mathbf{W} \text{diag}(\boldsymbol{\mu})$$

ランダムにノードを0にすることが、 $p(\hat{\mathbf{W}})$ という事前分布を考慮したサンプリングに対応

\mathbf{W} の正則化に関する項は変分推論における事前分布と変分分布のKL-divergenceの項に対応

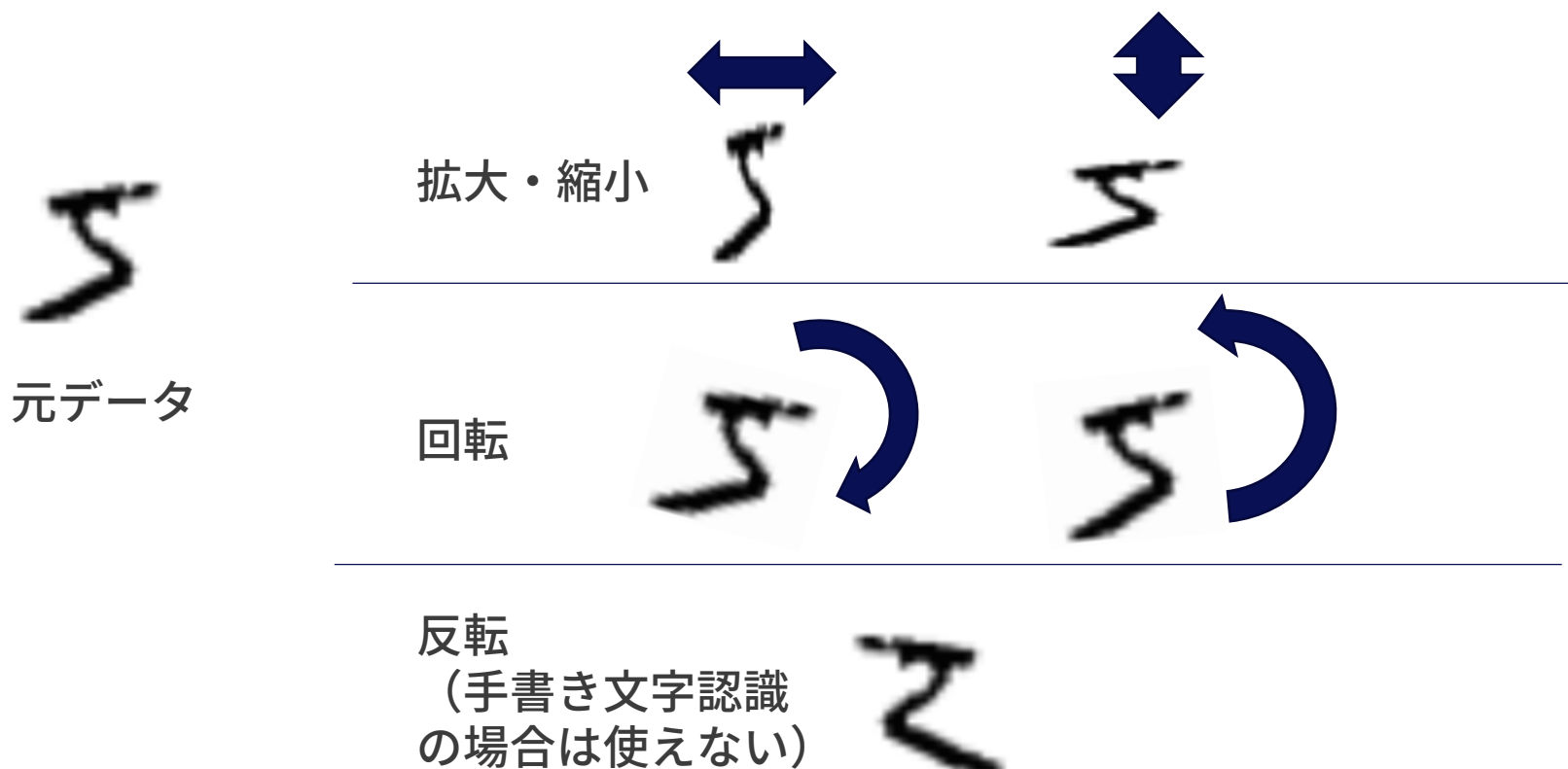
このことを使って、予測時にもランダムにノードの値をゼロにすることで、予測値の分散を計算することもできる

ニューラルネットワーク学習・評価の流れ



Data Augmentation

- データに変換・摂動を加えてデータを増やす手法
- 加えた変換によってラベルが変化しないという不変性を持たせられる



不変性を考慮した正則化項とみなすことができる
画像などの場合には使いやすい

代表的な構造と各種データに対する手法の紹介

画像

- Convolution
- Pooling
- 画像でよく使われるモデル

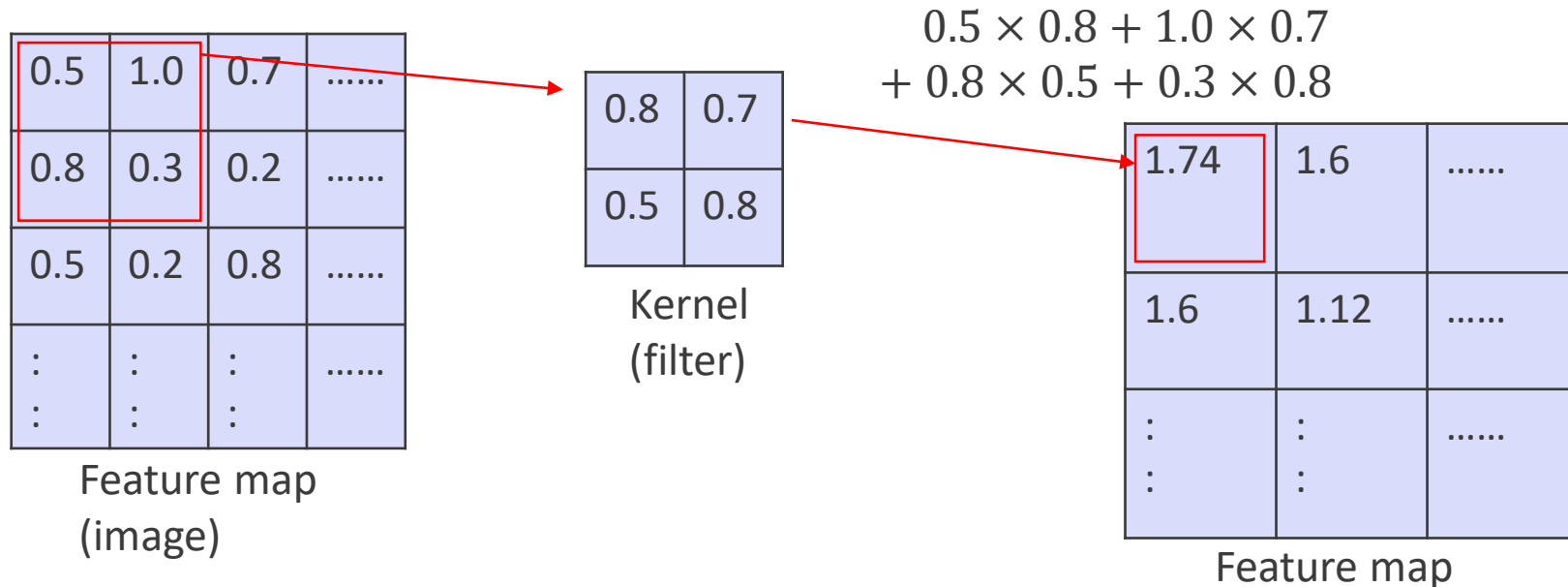
自然言語処理・時系列

- Embedding
- RNN
- LSTM
- Attention
- 画像でよく使われるモデル

Convolutional Neural Network : CNN

畳み込みニューラルネットワークは画像的特徴量に特化したニューラルネットワーク [Fukushima 79, LeCun+ 98]

- 畳み込み層とプーリング層を重ねて、最後にMLPを入れることで識別学習を行う



実際にはチャンネル方向にもう1次元あるので、（画像はRGBに対応）
Feature mapは3次元配列、Kernelは4次元配列になっている

フィルタリング

0.5	1.0	0.7
0.8	0.3	0.2
0.5	0.2	0.8
⋮	⋮	⋮
⋮	⋮	⋮

Image

0.8	0.7
0.5	0.8

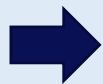
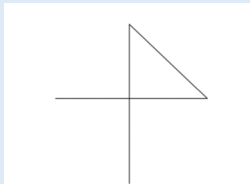
filter

$$0.5 \times 0.8 + 1.0 \times 0.7 + 0.8 \times 0.5 + 0.3 \times 0.8$$

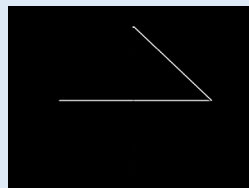
1.74	1.6
1.6	1.12
⋮	⋮
⋮	⋮

Image

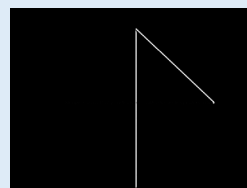
例 1



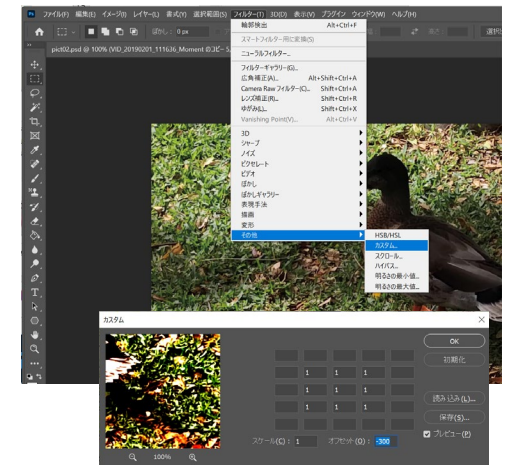
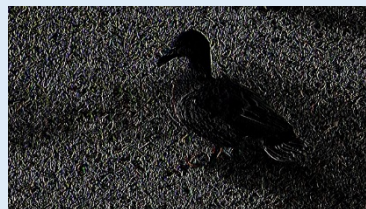
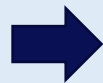
1	2	1
0	0	0
-1	-2	-1



-1	0	1
-2	0	2
-1	0	1



例 2



フィルタリング

0.5	1.0	0.7
0.8	0.3	0.2
0.5	0.2	0.8
⋮	⋮	⋮
⋮	⋮	⋮

Image

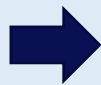
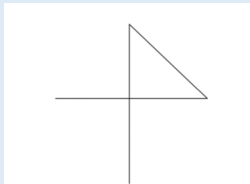
0.8	0.7
0.5	0.8

filter

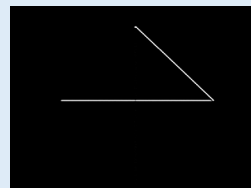
1.74	1.6
1.6	1.12
⋮	⋮
⋮	⋮

Image

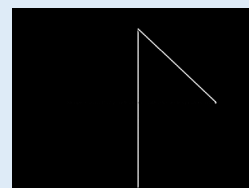
例 1



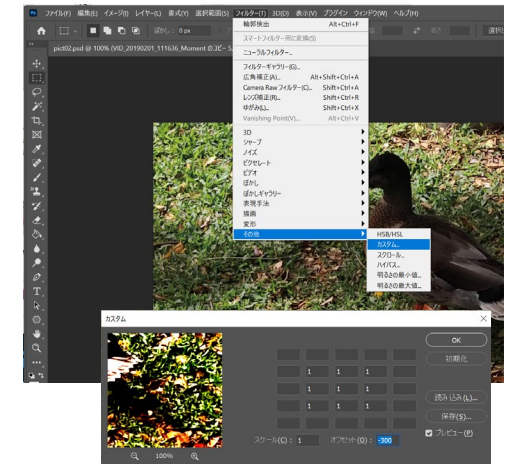
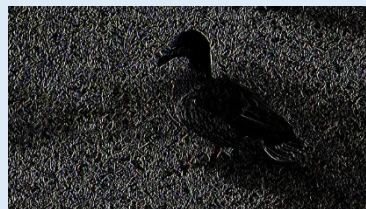
1	2	1
0	0	0
-1	-2	-1



-1	0	1
-2	0	2
-1	0	1



例 2



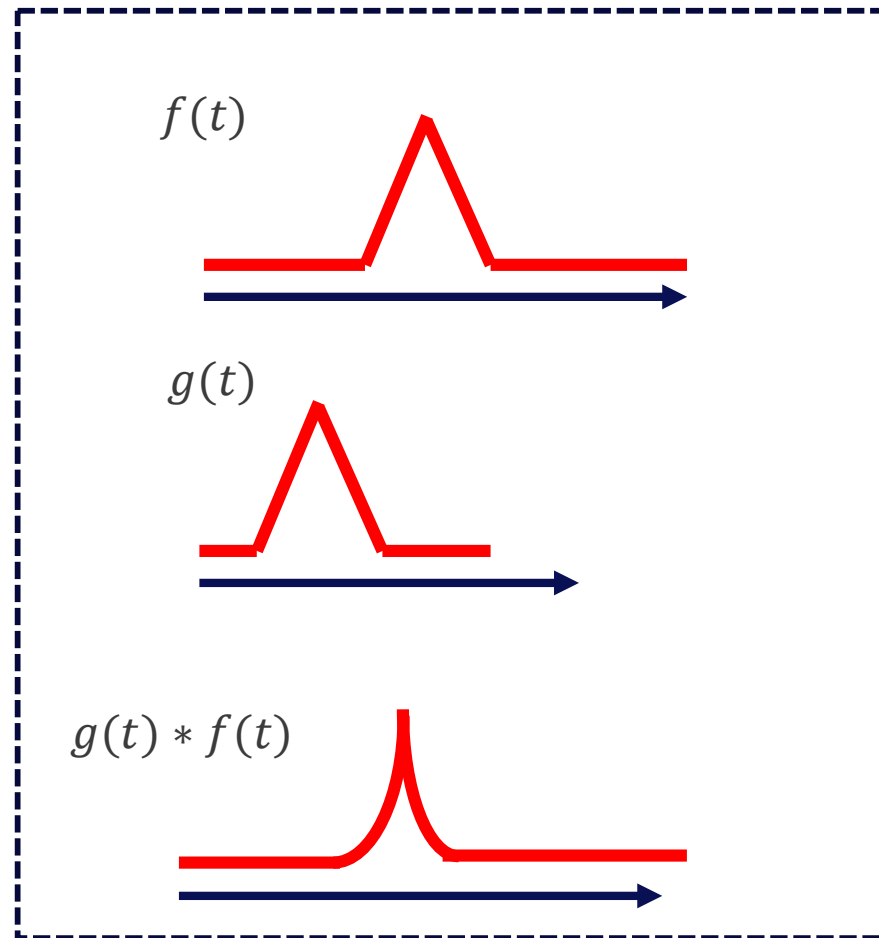
Convolution 演算 (1/3)

畳み込み計算により特定のパターンにのみ値が大きくなるような関数を構成することができる

畳み込み計算

$$g(t) * f(t) = \sum_{\tau} g(\tau) f(t - \tau)$$

特に信号処理の分野では既知の信号パターンを抽出する技術として類似した計算である相互相関関数を用いる



Convolution 演算 (2/3)

$$g(t) * f(t) = \sum_{\tau=0}^N g(\tau)f(t - \tau)$$

$$= (g(0), g(1), \dots, g(N)) (f(t - 0), f(t - 1), \dots, f(t - \tau))^T$$
$$= \mathbf{w}^T \mathbf{x}(t)$$

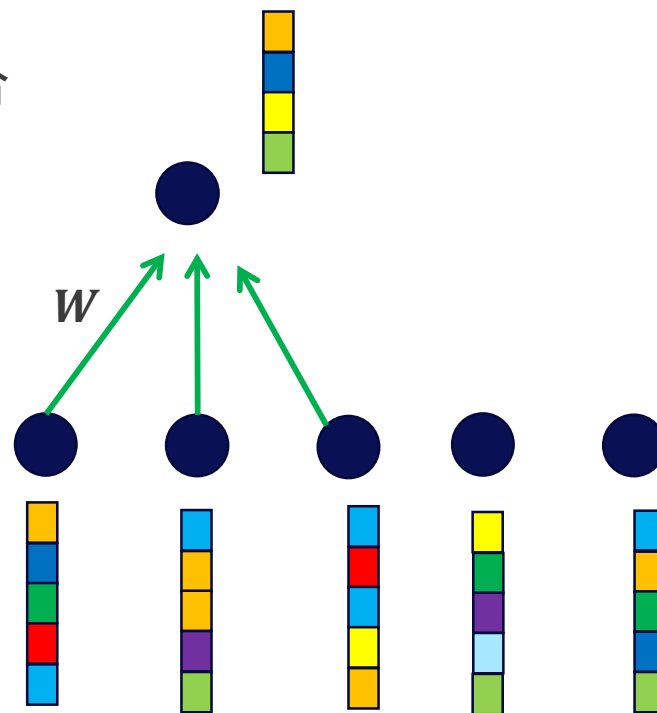
↓ 複数のフィルタを適用する場合

$$W\mathbf{x}(t)$$

↓ 入力がベクトル系列の場合

$$W\mathbf{X}(t)$$

Convolution は重みを共有し、かつローカルな接続のみを持つニューラルネットワークとみなせる



Convolution はある画素の周辺のベクトルに対してフィルタを掛けて、足し合わせるという処理を行っている。（行列表現では W を掛けるという操作）

Convolution 演算 (3/3)

$$g(t) * f(t) = \sum_{\tau=0}^N g(\tau)f(t - \tau)$$

$$= (g(0), g(1), \dots, g(N)) (f(t - 0), f(t - 1), \dots, f(t - \tau))^T$$
$$= \mathbf{w}^T \mathbf{x}(t)$$

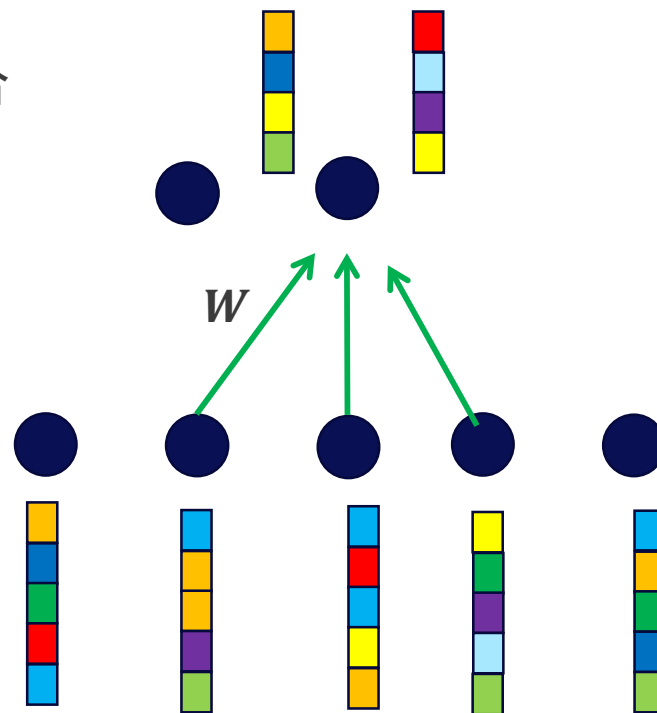
↓ 複数のフィルタを適用する場合

$$W\mathbf{x}(t)$$

↓ 入力がベクトル系列の場合

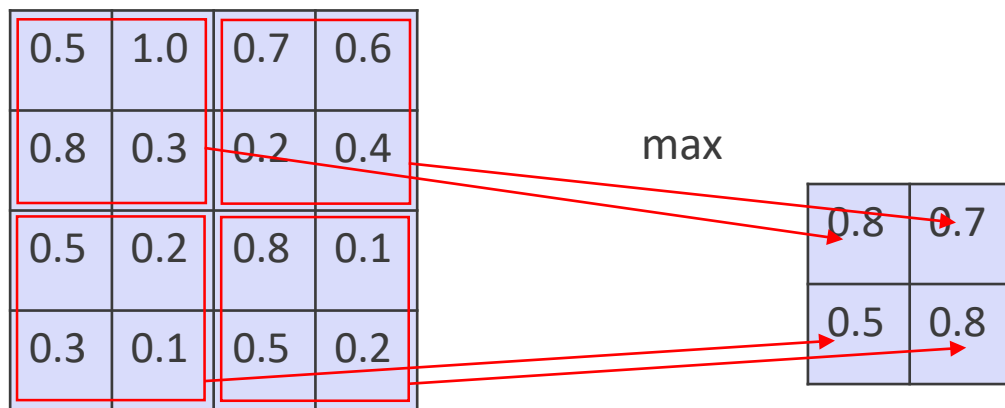
$$W\mathbf{X}(t)$$

Convolution は重みを共有し、かつローカルな接続のみを持つニューラルネットワークとみなせる



Convolution はある画素の周辺のベクトルに対してフィルタを掛けて、足し合わせるという処理を行っている。（行列表現では W を掛けるという操作）

Pooling層



Max pooling

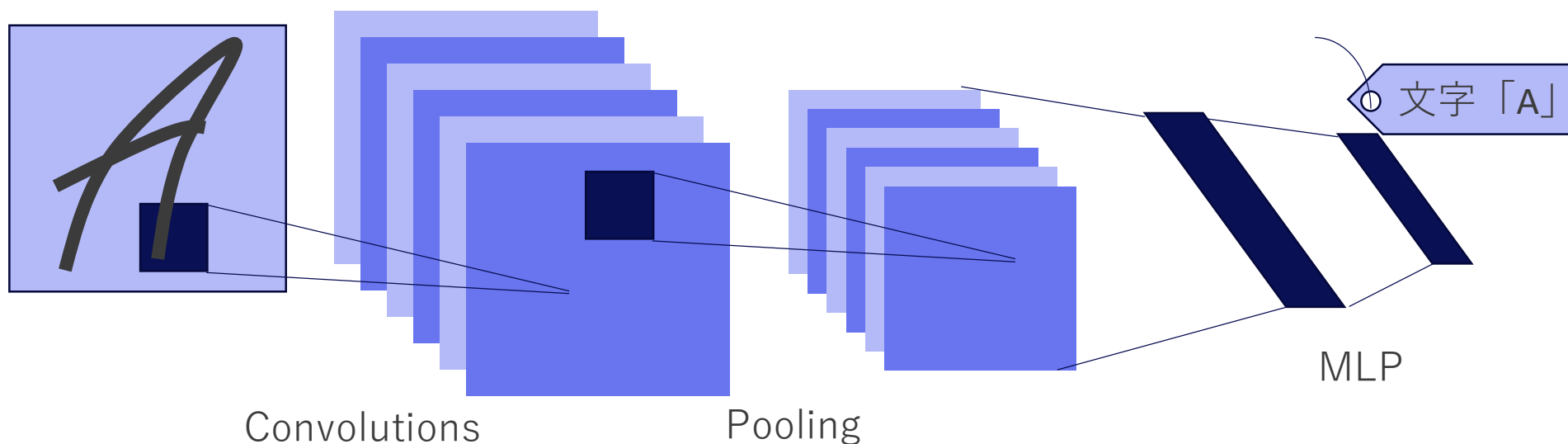
Maxをとる代わりに平均をとるAverage pooling などもある
実際にはチャンネル方向にもう1次元あるので、すべてのチャンネルに
対して同様の処理を行う。

Pooling層では画像中での少しの変化で出力が変わらないようにする効果がある

Convolutional Neural Network : CNN

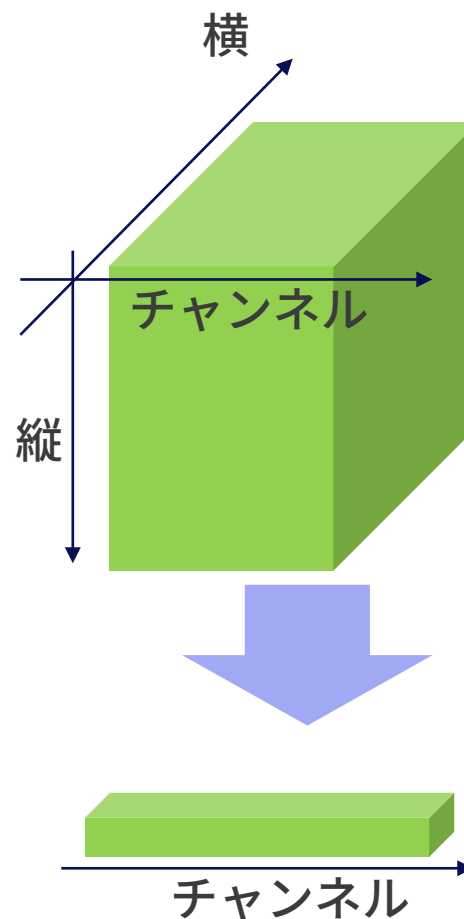
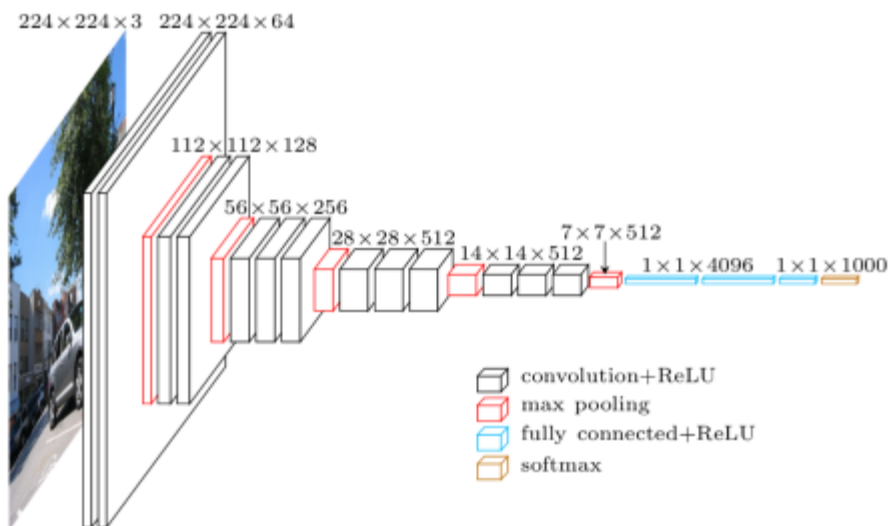
畳み込みニューラルネットワークは画像的特徴量に特化したニューラルネットワーク [LeCun+ 98]

- 畳み込み層とプーリング層を重ねて、最後にすべての画像をベクトルになるように並べて、MLPを入れることで識別学習を行う。



GlobalAveragePooling

VGG-16 [Simonyan+ 2015]

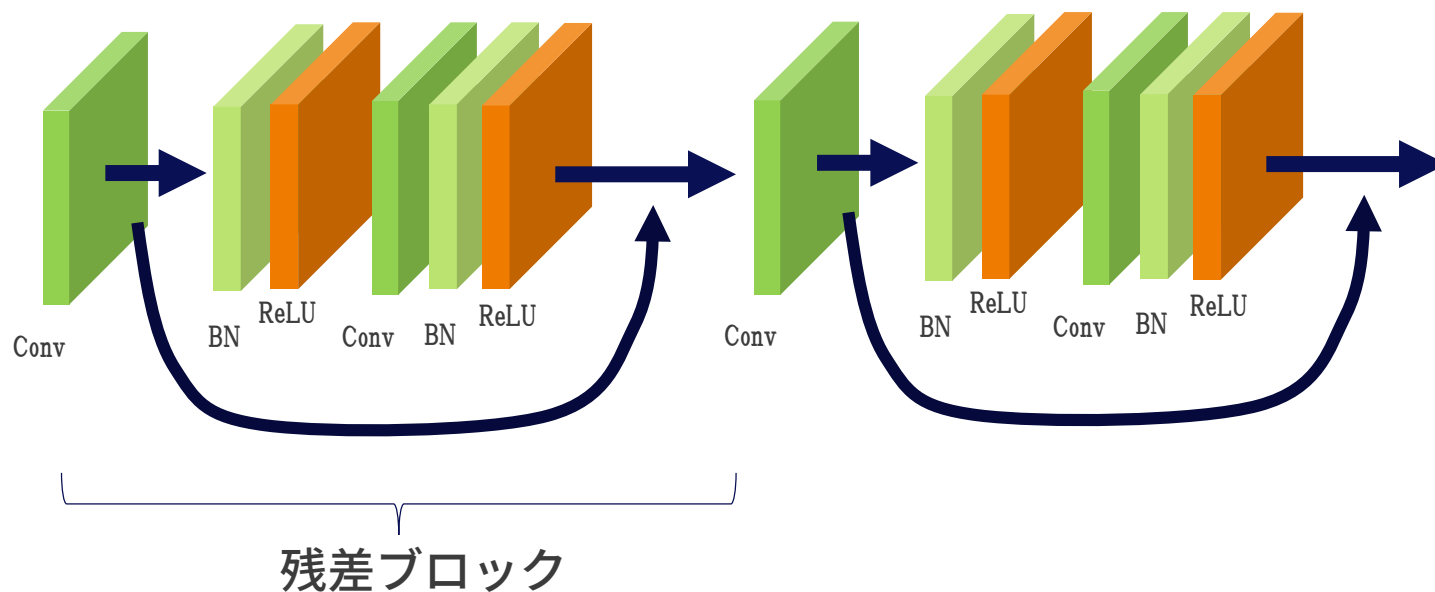


- 最終層でチャンネルごとに画像平均をとって1次元ベクトルにする
- 平均の代わりにMaxをとるGlobalMaxPoolingなどもある

ResNet

残差を学習するようなニューラルネットワークの構成方法

勾配消失の影響を受けにくいため非常に深いニューラルネットワークを構成できる



残差ブロックの式は以下のような形になる

$$y = x + f(x)$$

残差 ←

ResNet-50, ResNet-101, ResNet-152 のようなバージョンがあり、
152層のニューラルネットワークの学習にも成功した

画像関連の有名なニューラルネットワーク

～NetとかDeep～とかいろいろある <https://paperswithcode.com/area/computer-vision>

物体認識(Image classification)

- LeNet
- AlexNet
- VGG(Visual Geometry Group) (2014)
- GoogLeNet(Inception-v1…v4)
- ResNet(Residual Network)(2015)

airplane

automobile

bird

cat

deer

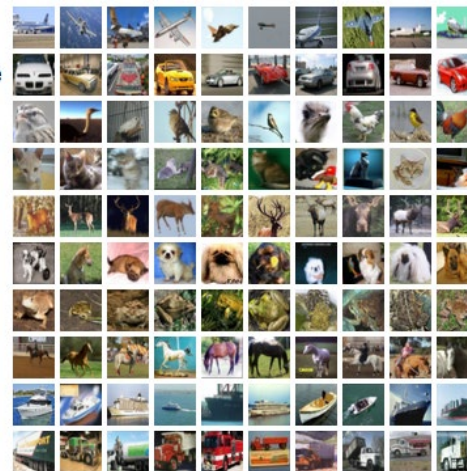
dog

frog

horse

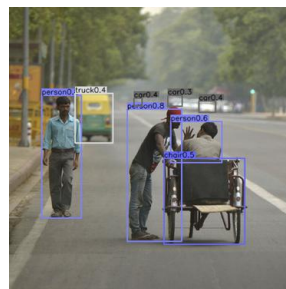
ship

truck



物体検出(Object Detection)

- Faster RCNN
- YOLO(You only look once: v1…v3)
- SSD



セマンティックセグメンテーション

- U-Net



代表的な構造と各種データに対する手法の紹介

画像

- Convolution
- Pooling
- 画像でよく使われるモデル

自然言語処理・時系列

- Embedding
- RNN
- LSTM
- Attention
- 画像でよく使われるモデル

Embedding

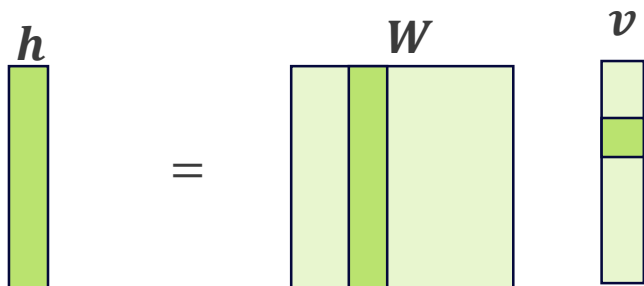
One-hotベクトル

入力の単語数×辞書サイズの行列
なので、辞書サイズが大きい時には直接保持が難しい

疎な行列なので

Embedding

v が one-hot ベクトルの時、 $h = Wv$ のニューラルネットの計算は行列 W の単語の ID 番目の行を取り出すことと同じ



リンゴ：単語ID 0
みかん：単語ID 3
メロン：単語ID 2

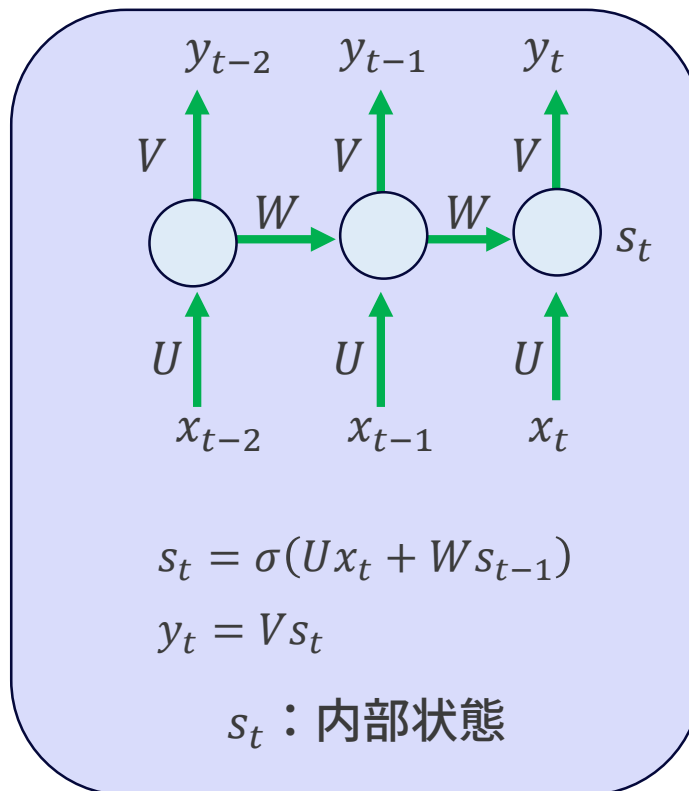
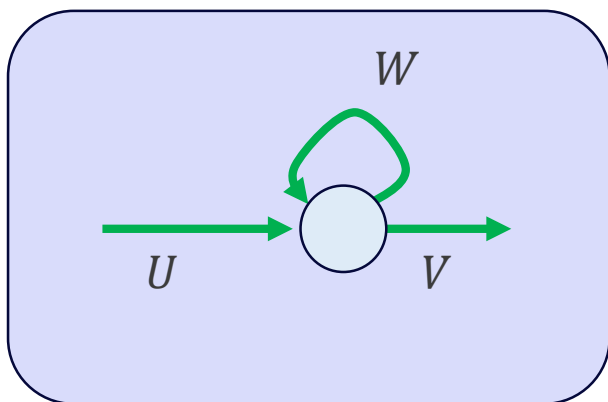
単語IDから直接 h を計算することで、one-hotベクトルを経由せずに単語をニューラルネットワークに入力できる

	リンゴ	みかん	メロン	...
リンゴ	1	0	0	0, ...
みかん	0	0	1	0, ...
メロン	0	0	1	0, ...

RNN (Recurrent Neural Network) (1/2)

再帰構造を持ったニューラルネットワーク

右のように展開したニューラルネットワークとみなすことができる



利点

- ・ 時系列を扱うことができる

課題

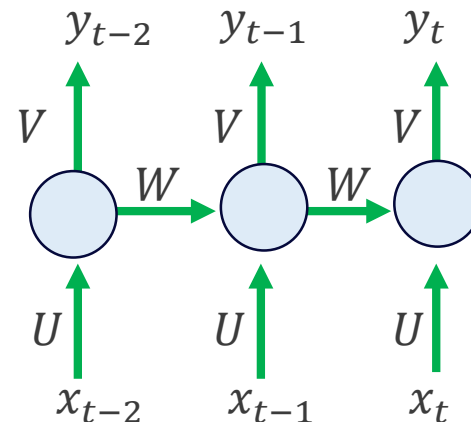
- ・ 勾配消失・勾配爆発の問題
- ・ 長時間の時間変化は学習が難しい

ライブラリ等のRNNは使いやすさのために s_t をそのまま出力するようになっていることが多いので注意

RNN (Recurrent Neural Network) (2/2)

RNNの誤差逆伝播

- $\frac{\partial E}{\partial w} = \sum_{t=1}^T \frac{\partial E_t}{\partial w}$
- $\frac{\partial E_t}{\partial w} = \sum_{k=1}^t \frac{\partial E_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial w}$
- $\frac{\partial s_t}{\partial s_k} = \prod_{i=k+1}^t \frac{\partial s_i}{\partial s_{i-1}}$



この部分が深いニューラルネットと同じ状況になっている

$$\frac{\partial s_5}{\partial s_0} = \frac{\partial s_5}{\partial s_4} \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

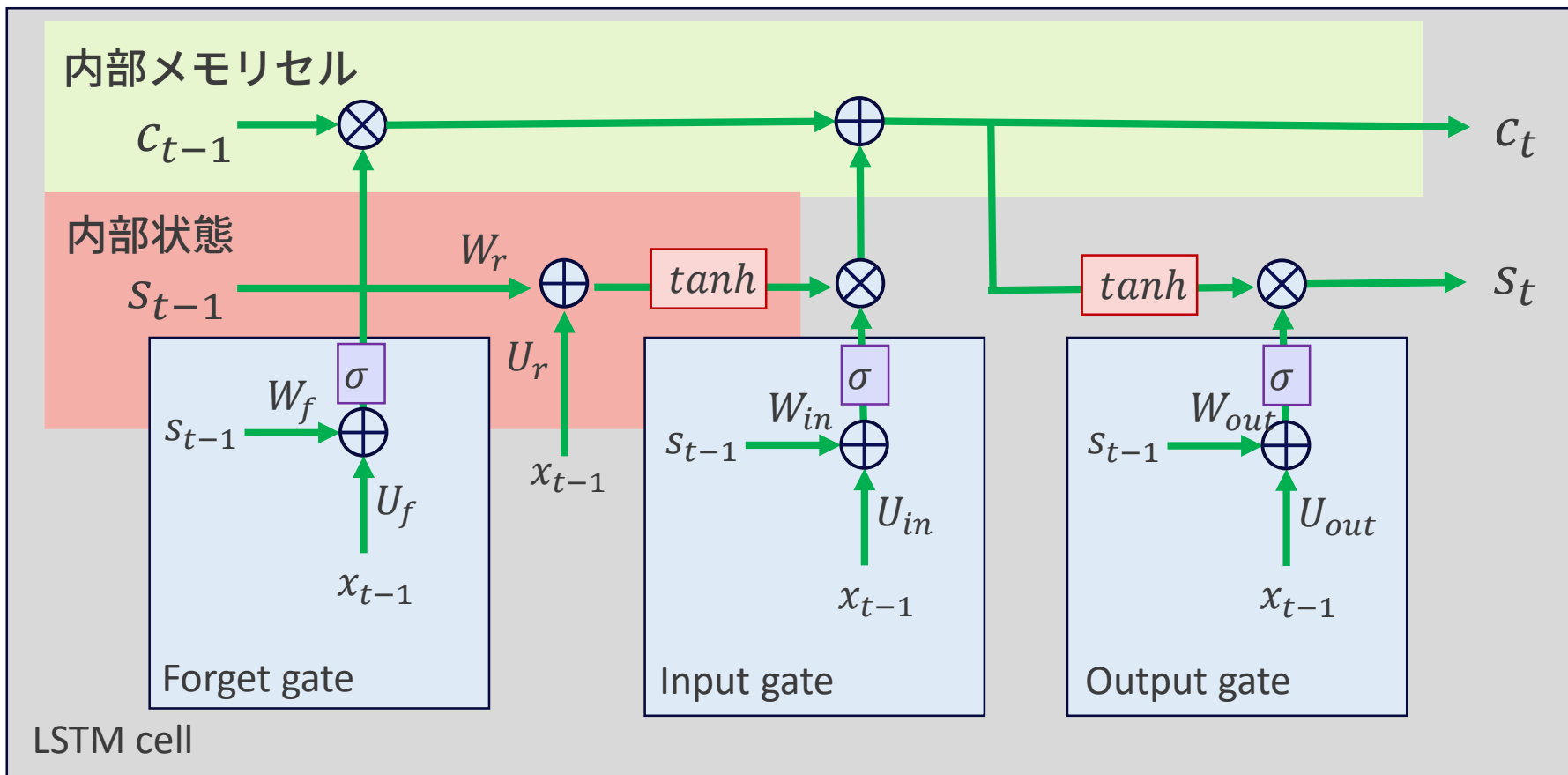
$$\frac{\partial s_t}{\partial s_{t-1}} = \sigma'(Ux_t + Ws_{t-1})W \text{ が}$$

- 1より小さい：勾配消失
- 1より大きい：勾配爆発

ただし、深いMLPと異なり、深い部分と浅い部分でWを共有しているため、単に勾配消失するだけでなく、勾配爆発も起こりやすい

LSTM(Long-term short-term memory)

RNNの内部状態に加えてメモリセルと呼ばれる変数を追加し、それらをgateで制御するような構造を持ったニューラルネットワーク

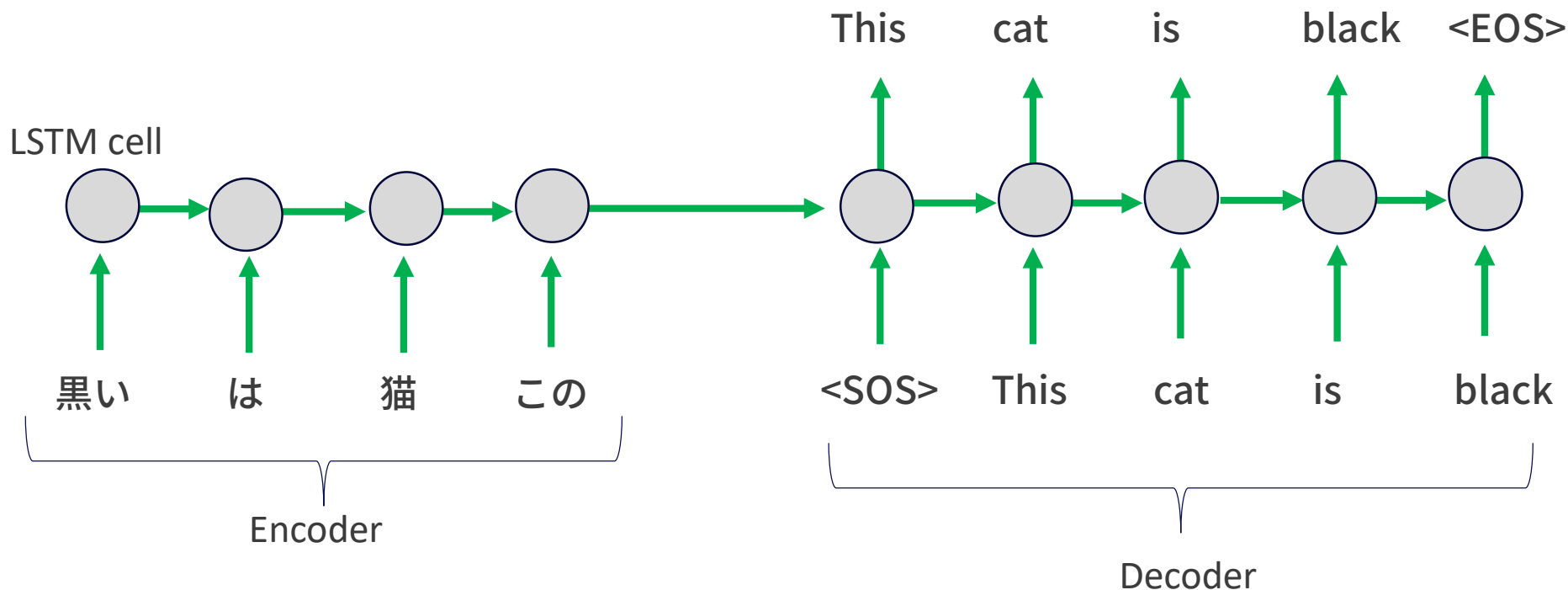


メモリセルの部分は安定して情報を伝達できる(Constant Error Carousel (CEC) : Forget gateがなければ係数1、ただし、Forget gateのない場合だと情報を更新するのに過去の蓄積よりも大きい入力をメモリセルに入れる必要があった)

LSTM(Long-term short-term memory)

LSTMを用いたニューラルネットワークの例

- 自然言語の対話や翻訳によく使われている encoder-decoder model (sequence to sequence model)のパーツとして利用できる



Attention(1/2)

Attentionはプログラミングにおけるqueryとkey-valueの計算をニューラルネットで行うものと解釈できる。

Price dictionary={
"リンゴ": 100,
"みかん": 40,
"メロン": 800,
}

辞書サイズ: $M = 3$
クエリ数: $L = 1$

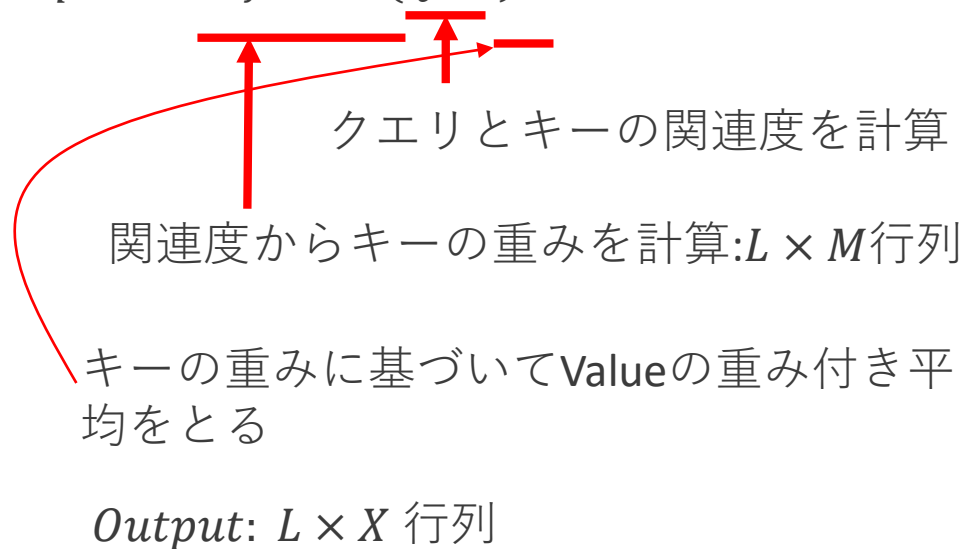
Key Value

q = "みかん"
v = Price_dictionary[q]

Query q
Key k
Value v

Query Q $L \times D$
Key K $M \times D$
Value V $M \times X$

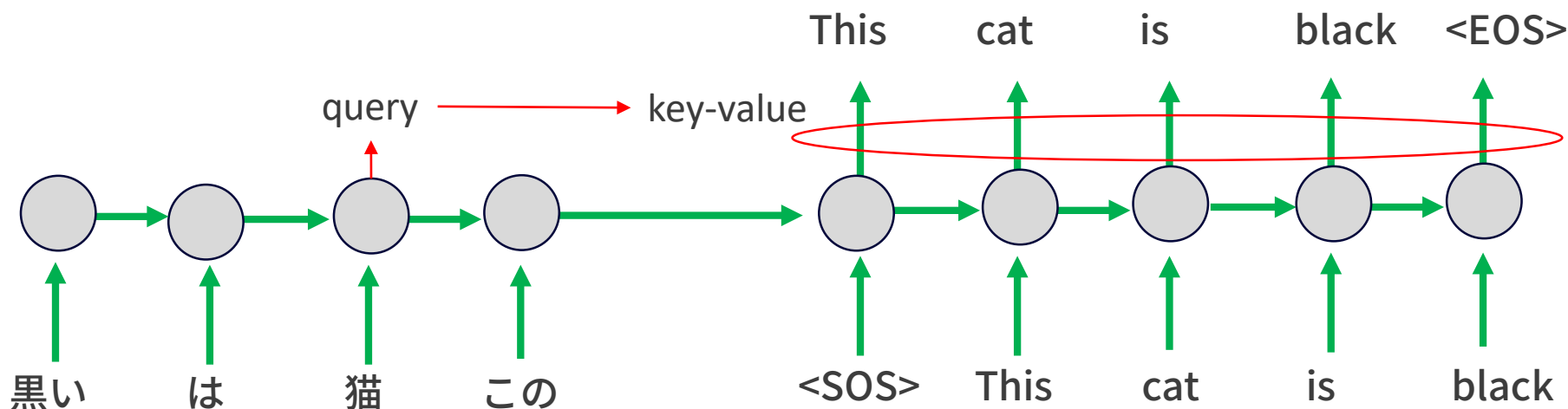
$$Output = \text{Softmax}(QK^T)V$$



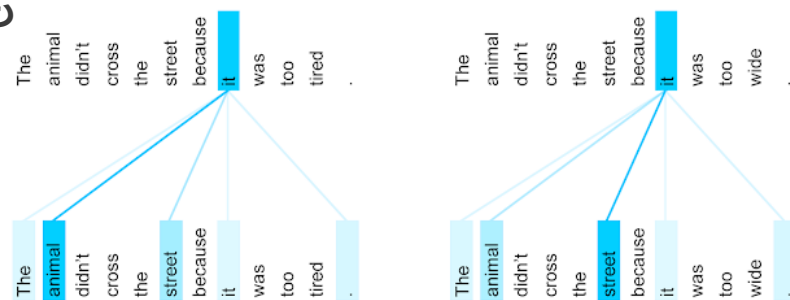
Attention(2/2)

Attentionはencoder-decoder model と合わせて利用することが可能

特に、キーの重みを可視化することでqueryに最も関連したkeyを見つけられる



Attentionの応用範囲は広く文が一つの場合にも自分自身をqueryとkey-valueを同じ文にすることもできる (self-attention) また、複数のAttentionを同時に使うMulti-head attentionなどの改良もある。



自然言語処理関連の有名なモデル

- GNMT (Google's NeuralMachine Translation) [Wu+ 2016]
Encoder-decoder model: 8-layer LSTM + residual connection
- Elmo[Matthew+ 2018]
- Transformer [Vaswani+ 2017]
Attentionのみを使用したニューラル翻訳モデル
- BERT[Devlin+2018]
BERTは、大規模コーパスから事前学習（単語補間のタスクと隣接文予測のタスク）させたTransformerモデルに、目的のタスクごとに再度学習をしたら多くのタスクで高い得点を出した
- GPT-2[Radford+2019]
 - Transformer ベースの48層NNを800万文書で学習
 - 特に生成で有名

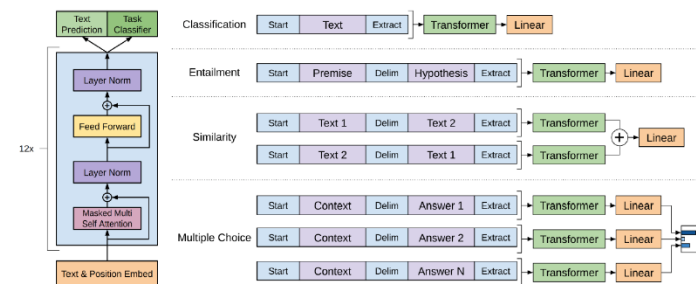
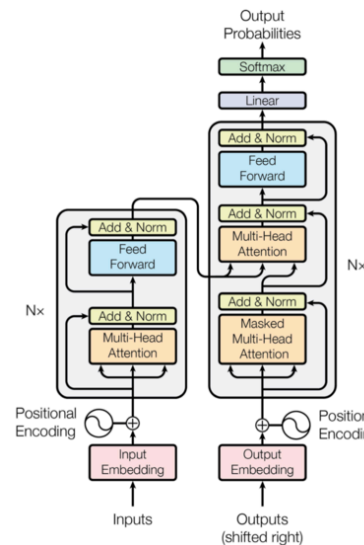


Figure 1: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

代表的な構造と各種データに対する手法の紹介

画像

- Convolution
- Pooling
- 画像でよく使われるモデル

自然言語処理・時系列

- Embedding
- RNN
- LSTM
- Attention
- 自然言語でよく使われるモデル

まとめ

- ニューラルネットの基礎
- 深層学習の基本的なテクニック
- 最近の画像認識・自然言語処理のモデルと関連技術の紹介

今回話さなかったこと

- 深層学習の解釈・説明可能性
- GANやVAEなどの生成モデル
- Adversarial exampleなどの話題