Review of machine learning

Naoto Mizuno
2019/4/22 地震発生論セミナー

目次

今学期のセミナーで出てくる手法を中心に紹介

- K-means
- MinHash
- Random Forest
- Hidden Markov Model
- Neural Network

教科書

- パターン認識と機械学習
- ゼロから作るDeep Learning
- はじめてのパターン認識

- scikit-learn Tutorials

  https://scikit-learn.org/stable/tutorial/index.html
- Chainer Tutorial

  https://tutorials.chainer.org/ja/tutorial.html

概論

機械学習とは

- きっちりとした定義はない
  時代によって全然違うので注意

- “データから特徴を抽出する”
- ルールを明示的に与えない（↔ルールベース）

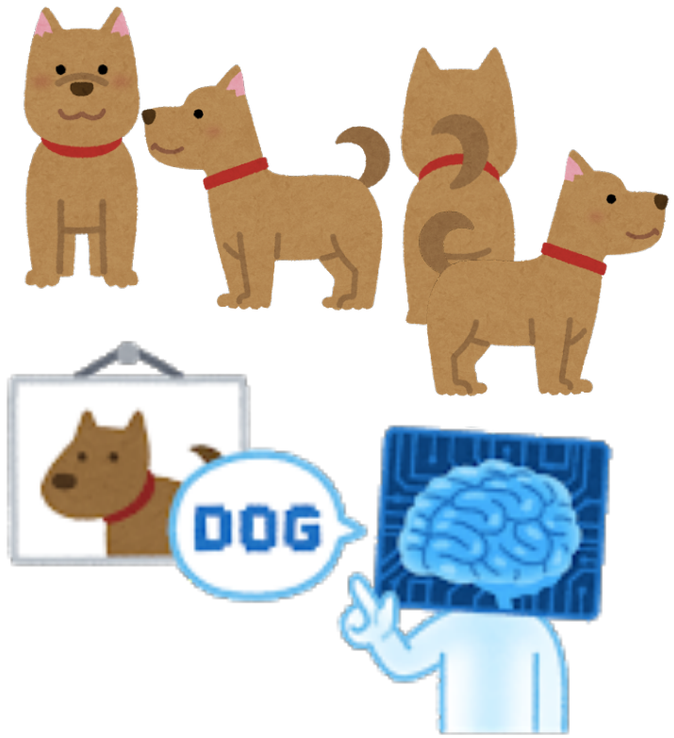
ルールベースのイメージ

- 犬とは？
  - 鼻が高い
  - 耳が尖っている
  -
  - …

- 人手で網羅するのは難しい

機械学習のイメージ

- 犬の画像を大量に与える
  - 犬の特徴を機械が自動で学習



- 基本的に精度は100%にならない

機械学習の分類

- 教師あり学習
- 教師なし学習
- 半教師あり学習
- 強化学習

教師あり学習

- データとラベルのペアが与えられる
- データからラベルを予測する

- 例：分類問題・回帰問題
  - 「このデータはシグナルかノイズか？」（分類）
  - 「余震が起こる確率は？」（回帰）

教師なし学習

- データのみが与えられてモデルを学習する

- 例：クラスタリング
  - 「同じ地域で起きた地震をまとめる」

半教師あり学習

- ラベルのあるデータと、ラベルのないデータがある
  - ラベルありのデータは少ない（ことが多い）

- 現実的にはラベルのついているデータは少数
  - 例：「画像」はたくさんある

    「犬のラベルが付いた画像」は少ない

- ラベルの付いていないデータからも情報を得たい
  - Pseudo labeling / 分布を利用など

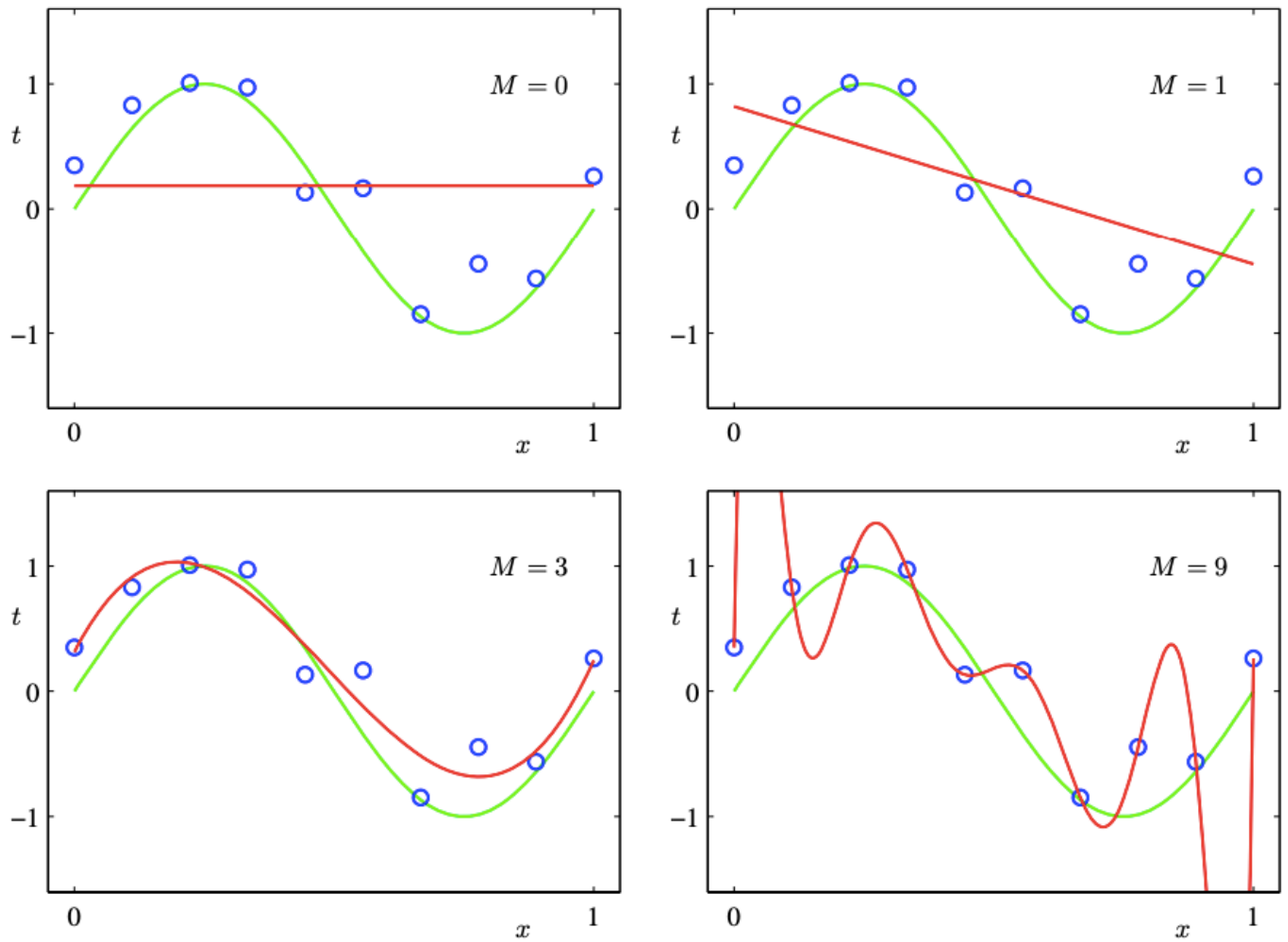
強化学習

- 環境を観測することで最適な行動を学習する
  - 行動による報酬がフィードバックされる

- 例：ボードゲーム（チェスなど）
  - 指す（=行動）と勝敗（=報酬）が分かる（=観測）

汎化

- 未知のデータに対して正確に判断できる ↔ 過学習
- 訓練データで性能が出ても、実際に性能が出るとは限らない



(PRML)

汎化

1. training data と validation data を作る
2. training data だけで学習させる
3. validation data でも性能が出たらOK

汎化

1. training data と validation data を作る
2. training data だけで学習させる
3. validation data でも性能が出たらOK

…とは限らないことに注意！

Validation の失敗例

「モデルを100個作ってvalidationが一番良かったものを使おう！」

- 手動で validation data に対して学習させている
  - validation が機能していない

- データを training と validation と test に分ける

汎化

1. training data と validation data と test data を作る
2. training data だけで学習させる
3. validation data でも性能が出る
4. 最後に test dataでも性能が出ることを確認

…これでもダメなケースがある！

test の失敗例

- training data と test data の中身に重複がある
  - 完全に一致していなくても関連があるだけでダメ

- できる限り関連のないデータを使う
  - 観測期間を変える / 観測点を変える等々

- 汎化性能の確認は細心の注意が必要

近年の機械学習の状況

最近の潮流：大規模化

- 大量のデータ（ビッグデータ）
- 大量のパラメータ
- GPGPU / ハイパーコンピューティング

機械学習によくある誤解

❌ 理論的背景がない・動けばOK

- なぜ上手く動くのかという研究も活発
- 統計学の知識は必須

❌ 解釈性がない・ブラックボックス

- ある程度は解釈/説明可能
- 何をもって“説明”とするかが難しい

K-means

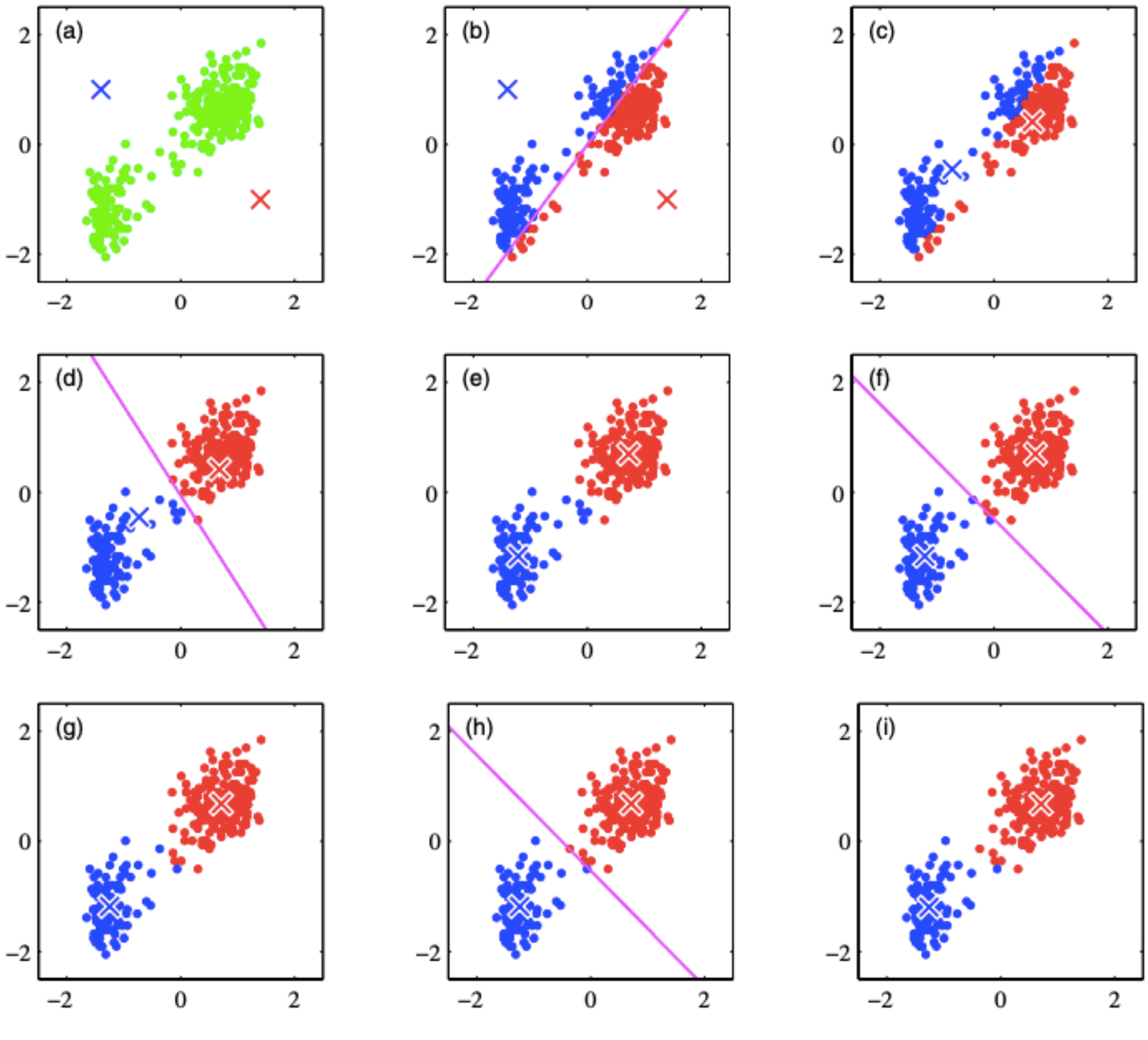
K-means

データをK個のクラスタに分ける

アルゴリズム：

1. K個のクラスタの中心をランダムに決める
2. データを最も近いクラスタに割り当てる
3. クラスタの中心を重心に決め直す
4. 2~3を繰り返す

K-meansの例



(PRML)

K-meansの理論的背景

- 目的関数　　$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| x_n - \mu_k \|^2$$

- r_{nk} はデータ点 x_n がクラスタ k に属している時1、

  そうでないなら0

- 「データを最も近いクラスタに割り当てる」

  → r_{nk} について最小化

- 「クラスタの中心を重心に決め直す」

  → μ_k について最小化

K-meansの理論的背景

- 目的関数 $$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2$$

- 最小二乗法が正規分布のlog-likelihoodから導出されることを思い出すと、

$$J' = \sum_{n=1}^{N} \sum_{k=1}^{K} \log N(x_n \mid \mu_k, I)$$

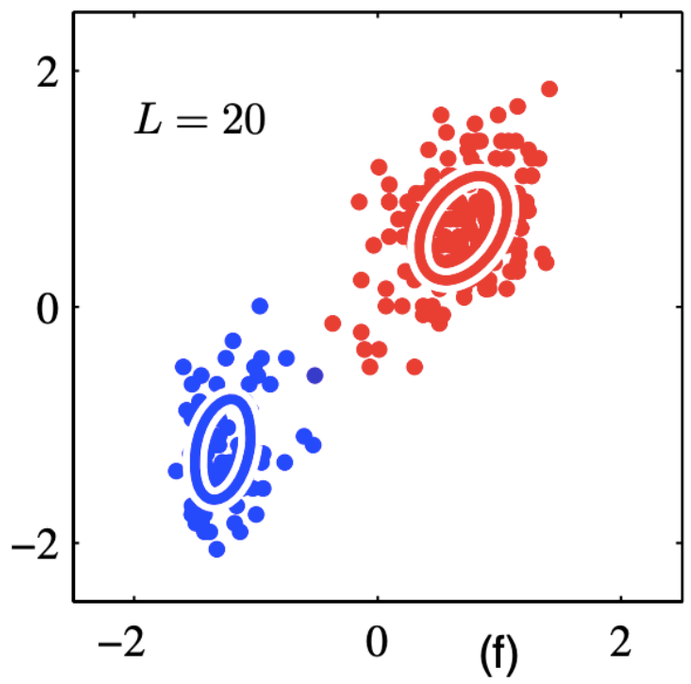
の最大化をしているようなものだとみなせる

（分散を0に近づける極限をとると一致する）

K-means に関連するアルゴリズム

- X-means
  - BICを基準に最適なKの値を決定

- 混合ガウス分布
  - 標準正規分布ではないようなデータ
  - 「ソフト」なクラス分け



(PRML)

MinHash

MinHash

- 似たデータのペアを探すアルゴリズム
  - Similarity Search の一種

- 扱うデータは集合
  - 同じ要素が多い時「似ている」

- FASTの場合、01のビット列を集合とみなす
  - 例：(0, 1, 1, 0) → {2, 3}

MinHash

- 似ている

  S_1 : {**10**, 20, **30**, **40**, **50**}

  S_2 : {**10**, 15, **30**, **40**, **50**, 60}

- 似ていない

  S_1' : {10, 20, 30, 40, 50}

  S_2' : {11, 21, 31, 41, 51}

Jaccard 係数

- 集合の類似度：Jaccard 係数

$$J = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

- 例

$S_1 = \{10, 20, 30, 40, 50\}$

$S_2 = \{10, 15, 30, 40, 50, 60\}$

$$J = \frac{|S_1 \cap S_1|}{|S_1 \cup S_2|} = \frac{|\{10,30,40,50\}|}{|\{10,15,20,30,40,50,60\}|} = \frac{4}{7}$$

Hash

MinHash は Jaccard 係数の近似を高速に計算する

…その前にHashを説明

- Hashとは “入力の代用となる整数への写像”
- 高速に一致判定を行える
- 連想配列の実装方法の一つ

  : ハッシュテーブル

```
>>> hash(0)
0
>>> hash(0.123)
2836186901332843 52
>>> hash(1000000000000000000000)
8487506038111601 07
>>> hash('hasseiron')
-57530363102380 1467
```

Pythonの例

MinHash

- 集合のそれぞれの要素に対してハッシュを計算する
- それらの最小値を集合のハッシュとする

例 :

$S_1 = \{10, 20, 30, 40, 50\}$

$H(S_1) = \min(H(10), H(20), H(30), H(40), H(50))$

$H(S_1) = \min(31, 24, 14, 6, 8) = 6$

H : ハッシュ関数

- このハッシュが一致する確率が Jaccard 係数と一致

MinHash と Jaccard 係数

- $S_1 \cup S_2$ からランダムに要素を一つ選び最小値とする
- これが $S_1 \cap S_2$ に含まれるならば $H(S_1) = H(S_2)$
  - 逆も成り立つ

- ハッシュ関数をたくさん用意すると Jaccard 係数の近似が得られる
  - ハッシュ関数のパラメータを変更するなど

MinHash に関連するアルゴリズム

- Nearest Neighbor
  - 最も似ているデータを探したい（距離関数は色々）
    - 愚直に計算すると$O(N^2)$
    - 工夫すると$O(N \log N)$（ただし高次元だと遅い）

- Approximate Nearest Neighbor
  - 近似解で良いことにすると高次元でも速くなる
  - Billion-scaleでも動作 (Johnson et al., 2017)

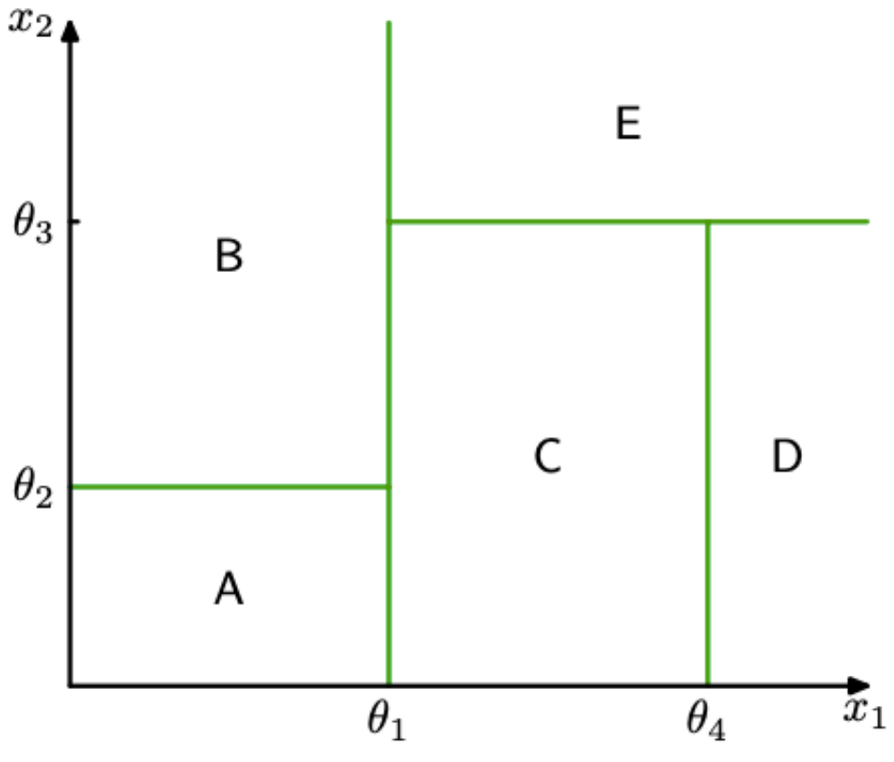
Random Forest

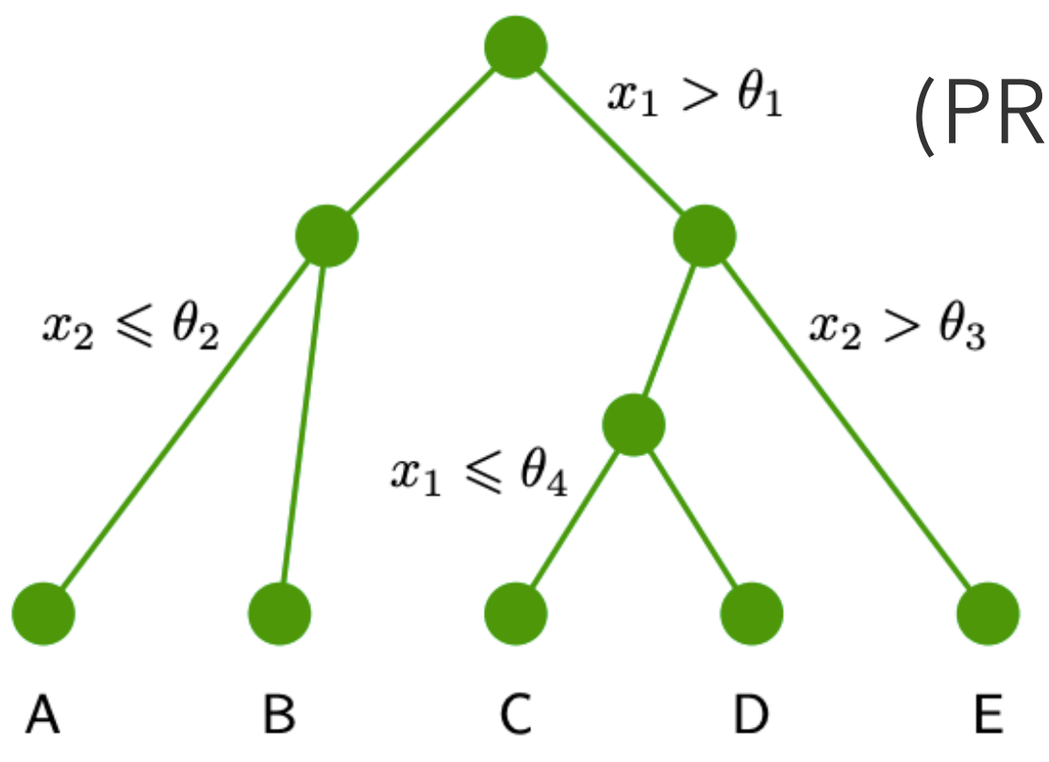
Random Forest

Random Forestは決定木をたくさん足し合わせたもの

…その前に決定木を説明

- 決定木：領域を木構造で分割する





(PRML)

決定木

- 木構造の作り方

1. ある関数（後述）を最小化する分割を探す
2. 分割された領域で、関数を最小化する分割を探す
3. 基準を満たしたら分割を終了
    - 要素数/残差など
4. データに対応する領域の平均値や最頻値を出力
    - 回帰にも分類にも使える

領域の分割基準

- 二乗和残差（回帰）

$$\Sigma(x_i - y)^2$$

- エントロピー / 平均情報量（分類）

$$-\Sigma p(x_i)\log p(x_i)$$

- Gini 係数（分類）

$$\Sigma p(x_i)(1 - p(x_i))$$

アンサンブル学習

- 決定木はデータの細部に敏感
  - 訓練データの分布に過学習しやすい

- アンサンブル学習
  - 性能がそこそこのモデル（弱学習器）をたくさん集めて合わせると性能が上がる
  - モデルのバリエーションの付け方は、データのサンプリング・特徴量のサンプリング・アルゴリズムの種類等々

ランダムフォレスト

- 決定木をたくさん作って多数決で決める
  - 「木」の集合なので「森」

1. データをランダムにサンプリング
   - データの分布に過学習するのを抑える
2. 分割するときに使う特徴量をサンプリング
   - 同じような決定木を学習するのを抑える

特徴量の重要度

1. ある特徴量をランダムにシャッフルして性能がどれだけ下がるかを見る

2. 分割基準の関数を小さくするのにどれだけ寄与しているかを見る

- scikit-learnはこっち

Random Forest に関連するアルゴリズム

- Bagging
  - サンプリングでデータセットをたくさん作って

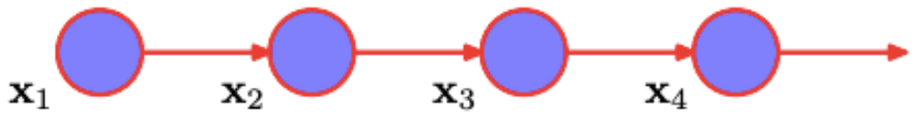
    アンサンブルする
  - Random Forest は Bagging + 特徴量のサンプリング
- Boosting
  - 今までに作った弱学習器で正しく学習できなかった

    データの重みを増やす

Hidden Markov Model

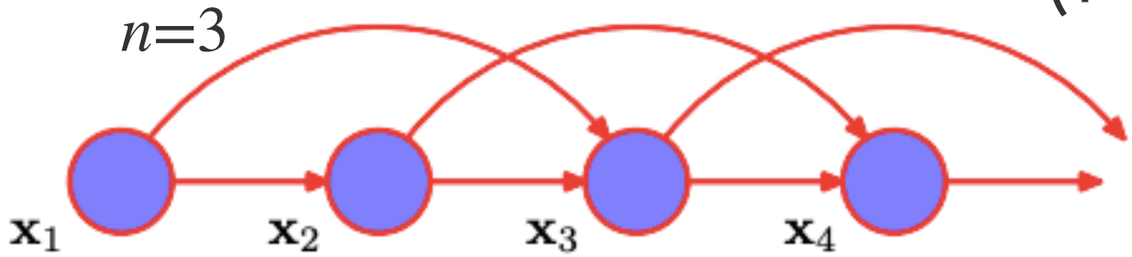
Hidden Markov Model

- 系列データに対する学習
  - 音声, 為替, オンライン手書き文字等々
- Markov chain
  - 一番シンプルな場合は、一つ前の観測値のみに依存
- $p(x_1, \ldots, x_N) = p(x_1) \prod_{n=2}^{N} p(x_n | x_{n-1})$



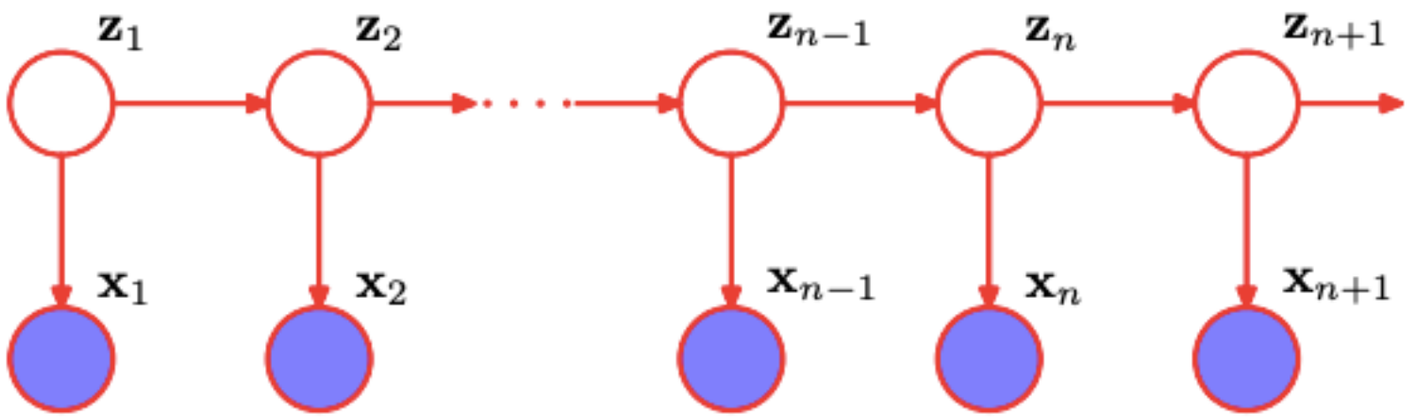
- もうちょっと複雑にしたい
- $p(x_1, \ldots, x_N) = p(x_1) p(x_2 | x_1) \prod_{n=3}^{N} p(x_n | x_{n-1}, x_{n-2})$
- もっと複雑に…？

(PRML)



Hidden Markov Model

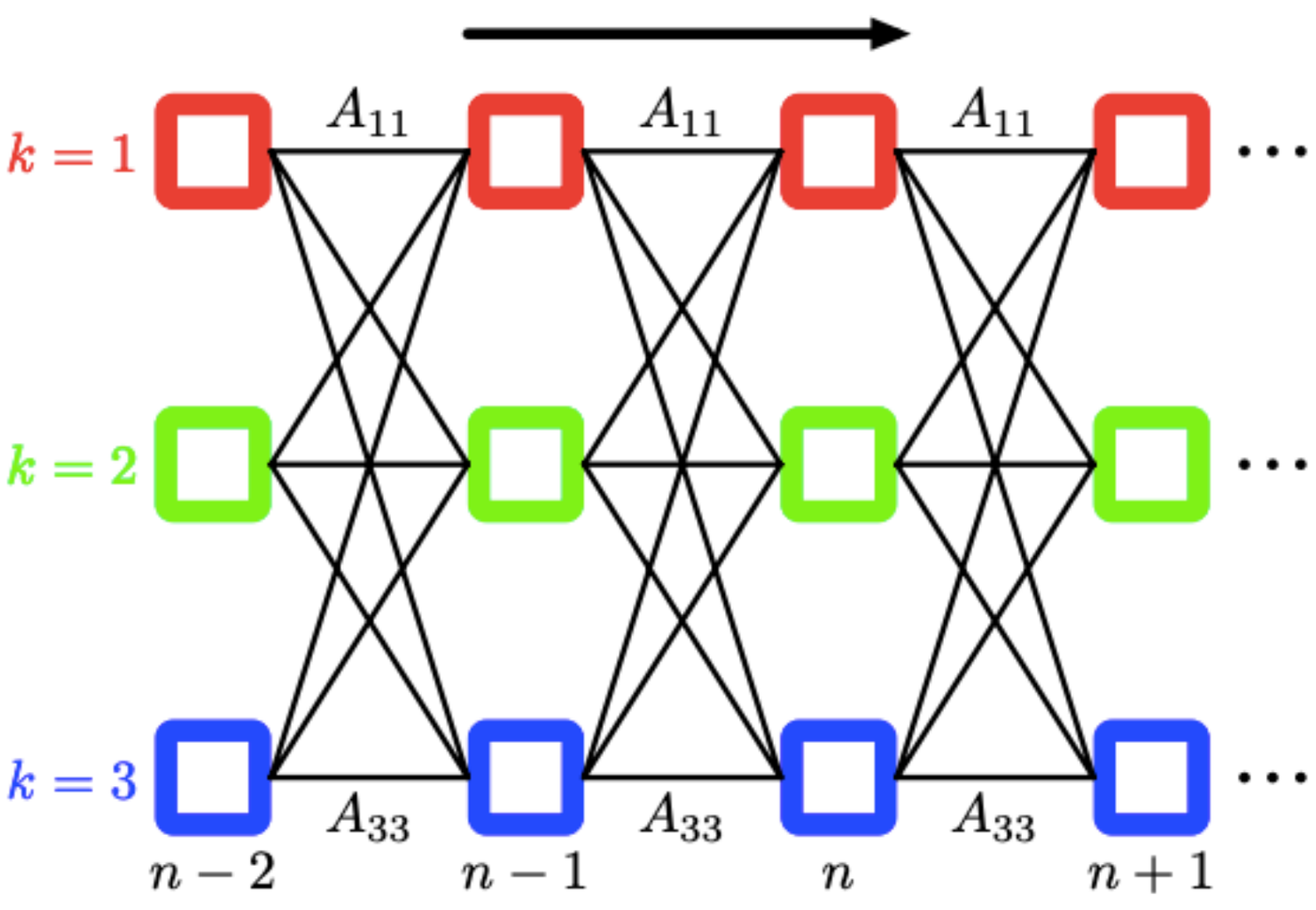
- 潜在変数を導入する
  - 潜在変数は直接的には分からない
  - 観測値だけが得られる
  - 例：大気の状態は分からないが降水量は分かる場合



(PRML)

Hidden Markov Model

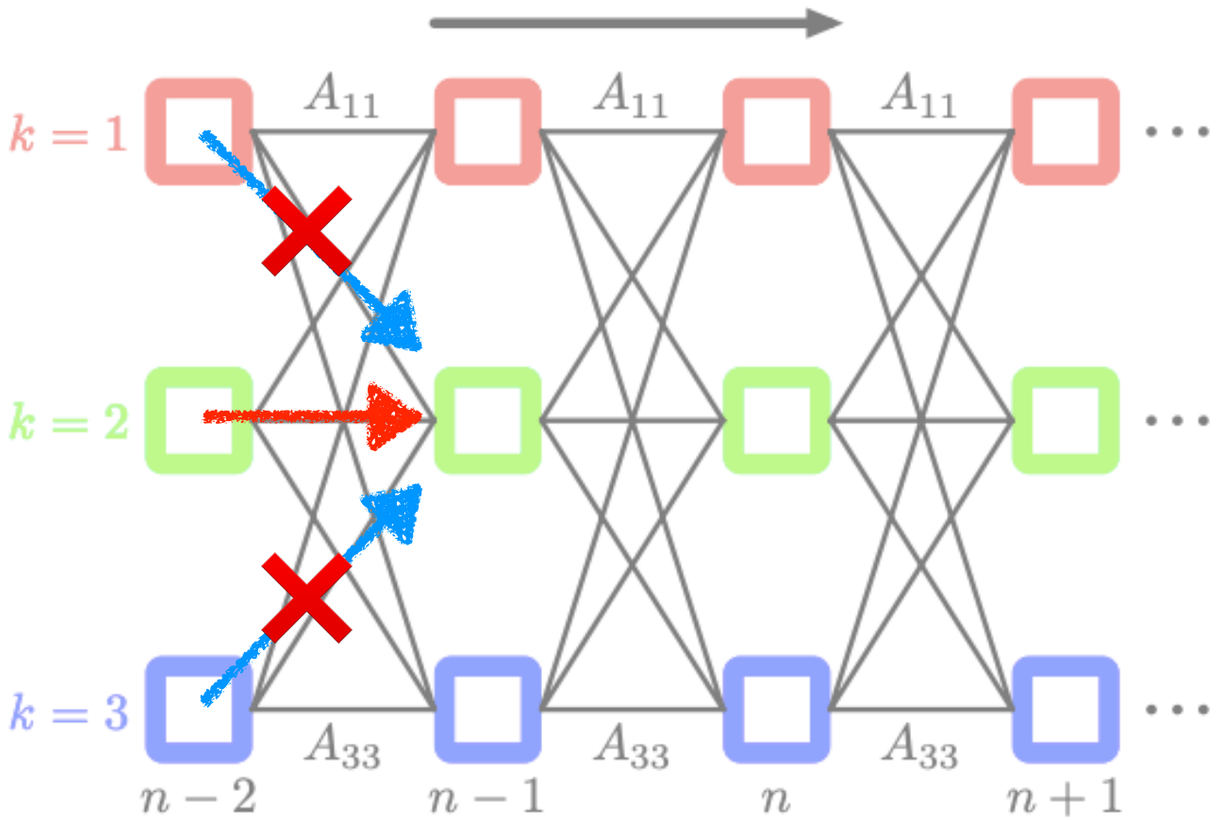
- 潜在変数が離散であるものを隠れマルコフモデルという
- 潜在変数だけ見るとMarkov chainになっている



(PRML)

Viterbi Algorithm

- 潜在変数を推定したい
  - 観測値を出力する確率が最も高い系列を選ぶ
- 系列は指数個あるが、確率が最大のものだけを
  各ステップで保持すれば良い



Hidden Markov Model に関連するアルゴリズム

- Recurrent Neural Network
  - 系列データをニューラルネットワークで扱う（後述）

Neural Network

Deep Learning

Neural Network の歴史

- 元々は"知能"の研究：Artificial Neural Network
  - 人工的に脳の機能を再現できるか？
- モデルの大規模化が困難：Neural Network 冬の時代

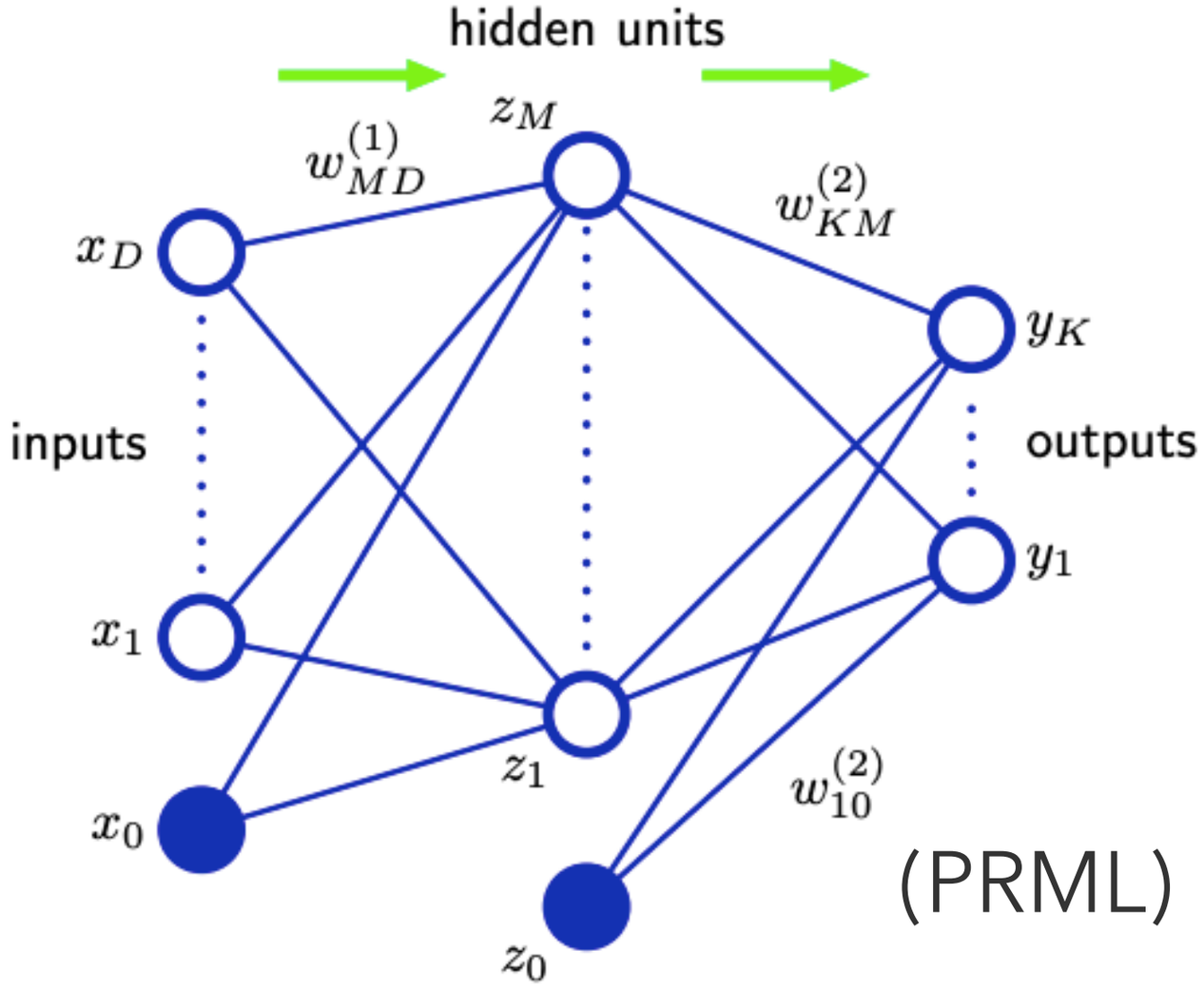
- 2012年に画像認識でブレイクスルー
  - Krizhevsky, Sutskever, and Hinton, 2012
- 様々な分野で高性能が出ることが知られてくる
- 画像、音声、自然言語処理、医療、…

Neural Network

- ニューロンの活動をモデル化

- 情報が順に流れる
- 入力層→隠れ層→出力層
  - 重みを学習する

- 隠れ層を増やすと複雑な関数を表現しやすい
  - ディープラーニング



(PRML)

Neural Network

- 行列を使って書くと

$$z = f(W_1 x)$$

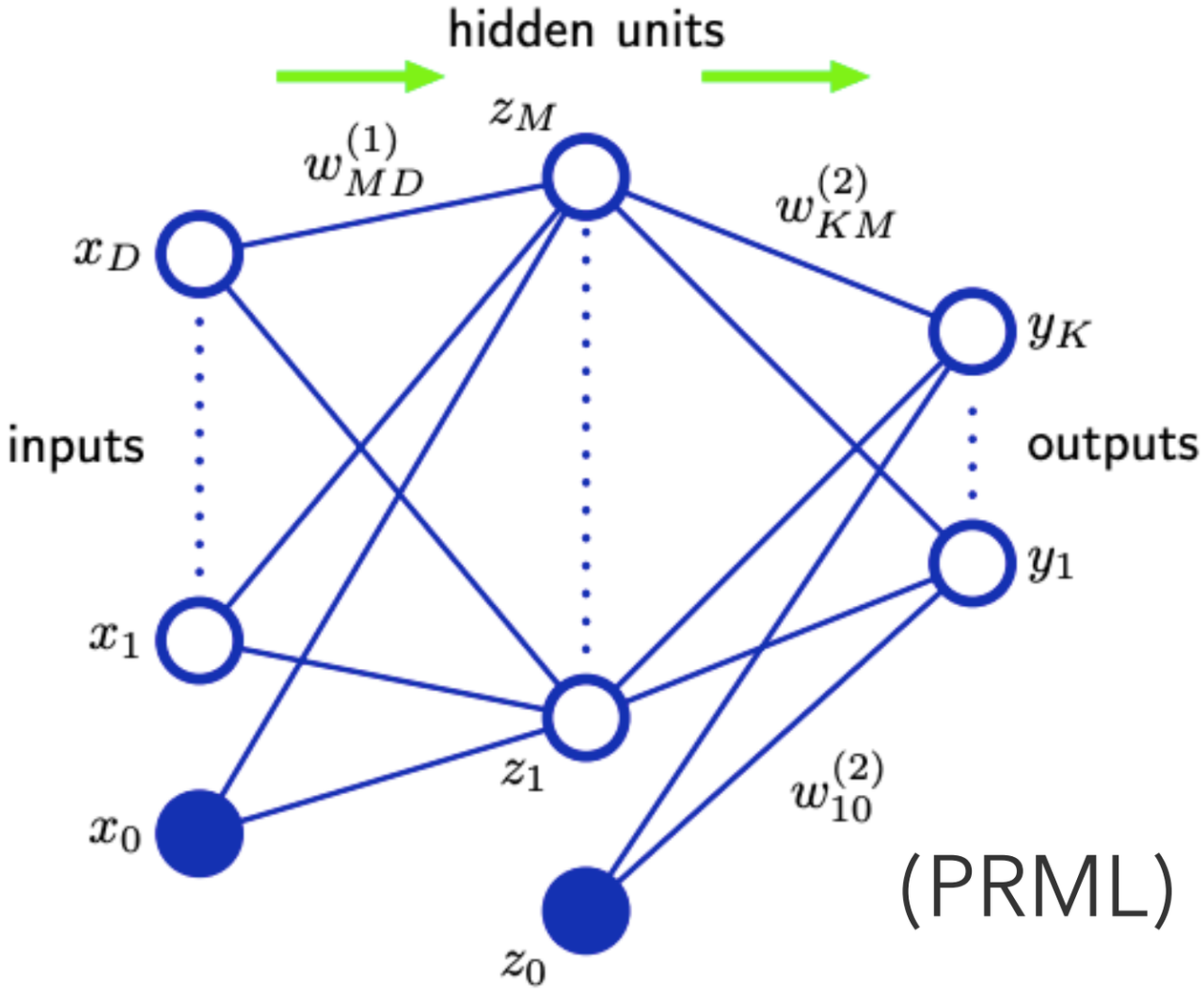
$$y = f(W_2 z)$$

f は何らかの非線形関数

- 活性化関数
- 最近はLeRU関数を使う

ことが多い

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

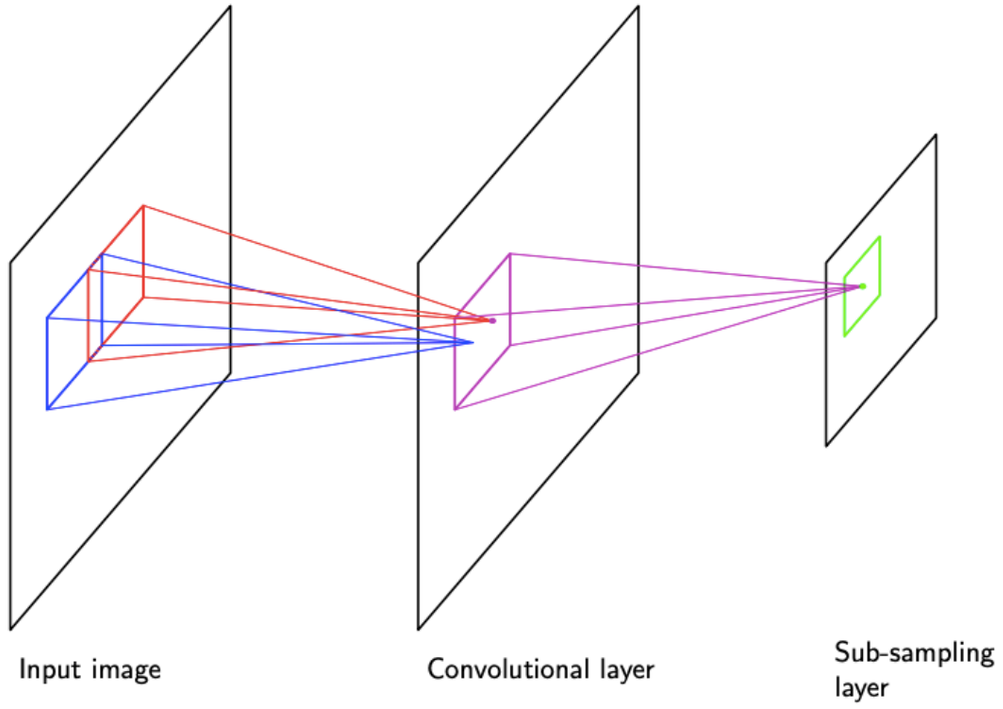


(PRML)

Convolutional Neural Network

- データの形状を利用したモデル
  - 主に画像分野で良く使われる
  - 平行移動してもデータの内容は変わらない

- “Convolution”
  - 部分領域に対するフィルター
- “Pooling”
  - 部分領域の情報をまとめる



(PRML)

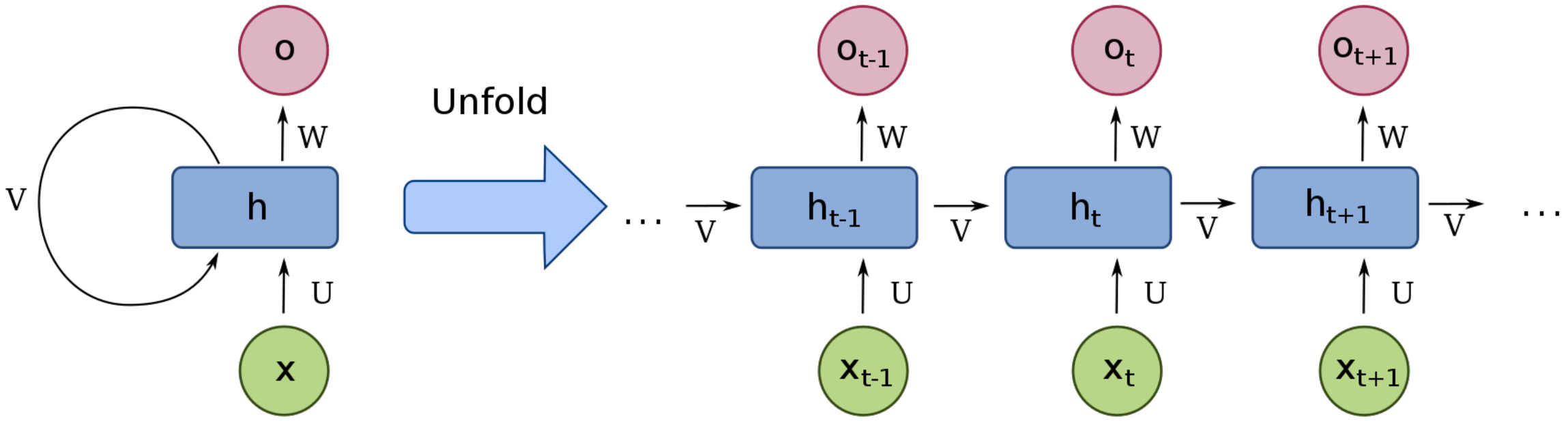
Convolutional Neural Network

- 普通のニューラルネットワーク（全結合）と比べて、
  パラメータの数を減らすことができる

- 形状の情報を利用することで性能が向上する
  - CNNの関数形が強い事前知識になっている
    （Ulyanov et al., 2017）

Recurrent Neural Network

- 系列データにニューラルネットワークを適用する
- 再帰的にニューラルネットワークにデータを入れる



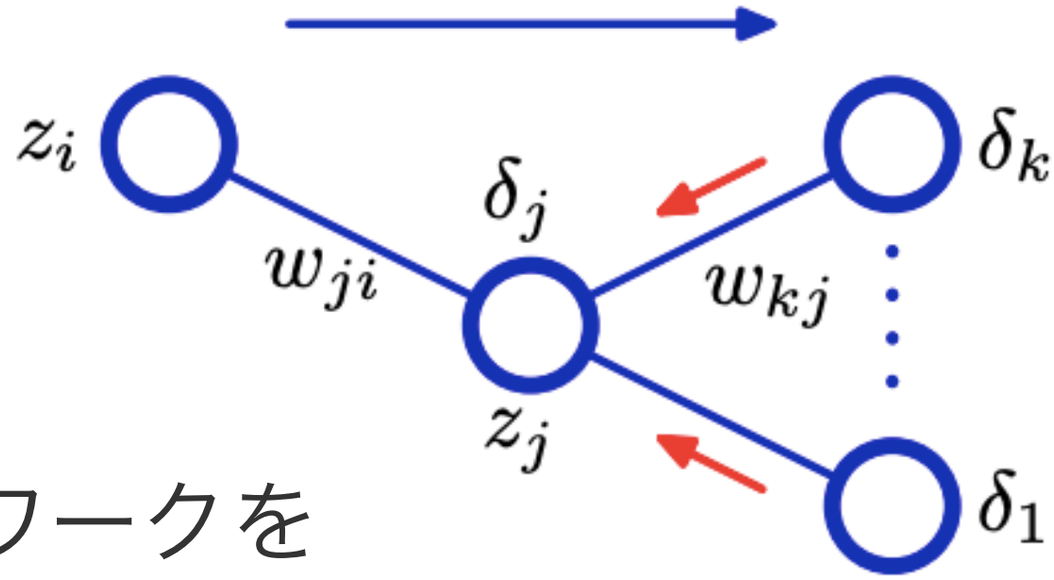
(François Deloche, Wikipedia)

- Long Short-Term Memory
  - 長距離でも情報を伝えられる構造を入れる

Back propagation

- 重みを学習するために勾配を求める
  - 誤差逆伝播法
  - 要するにchain rule

(PRML)



- ディープラーニングフレームワークを使えば自動で計算される
  - 自動微分

Deep Learning

- 隠れ層をたくさん並べると勾配が指数的に消失 / 発散
  - 勾配消失 / 勾配発散

- 様々な工夫により克服
  - 活性化関数, 正規化, ネットワーク構造等々

- 数10層のネットワークはよく用いられる
  - 100層を超えるのも一般的

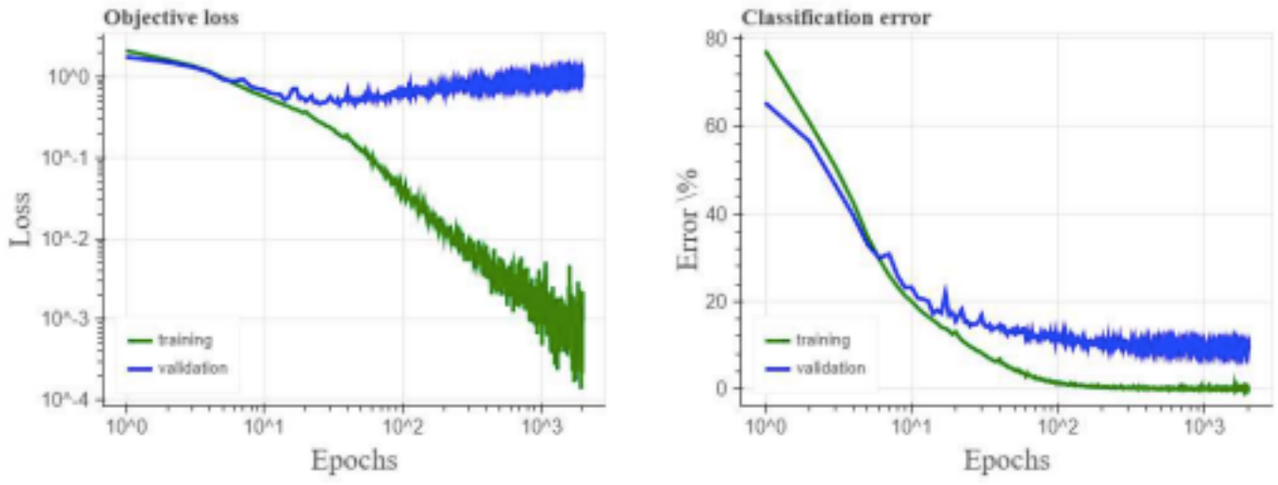
Stochastic Gradient Descent + Mini-batch

- Gradient Descent（最急降下法）
  - 全てのデータについての勾配方向に動かす
  - 局所解に陥りやすい
- Stochastic Gradient Descent（確率的勾配降下法）
  - ランダムに選んだ1つのデータに対して勾配を計算
  - 計算機の並列性を活かせない/外れ値に弱い
- Mini-batch SGD
  - データの部分集合（mini-batch）を使う
  - GDとSGDの中間的なアルゴリズム

Neural Network の学習の理論的研究

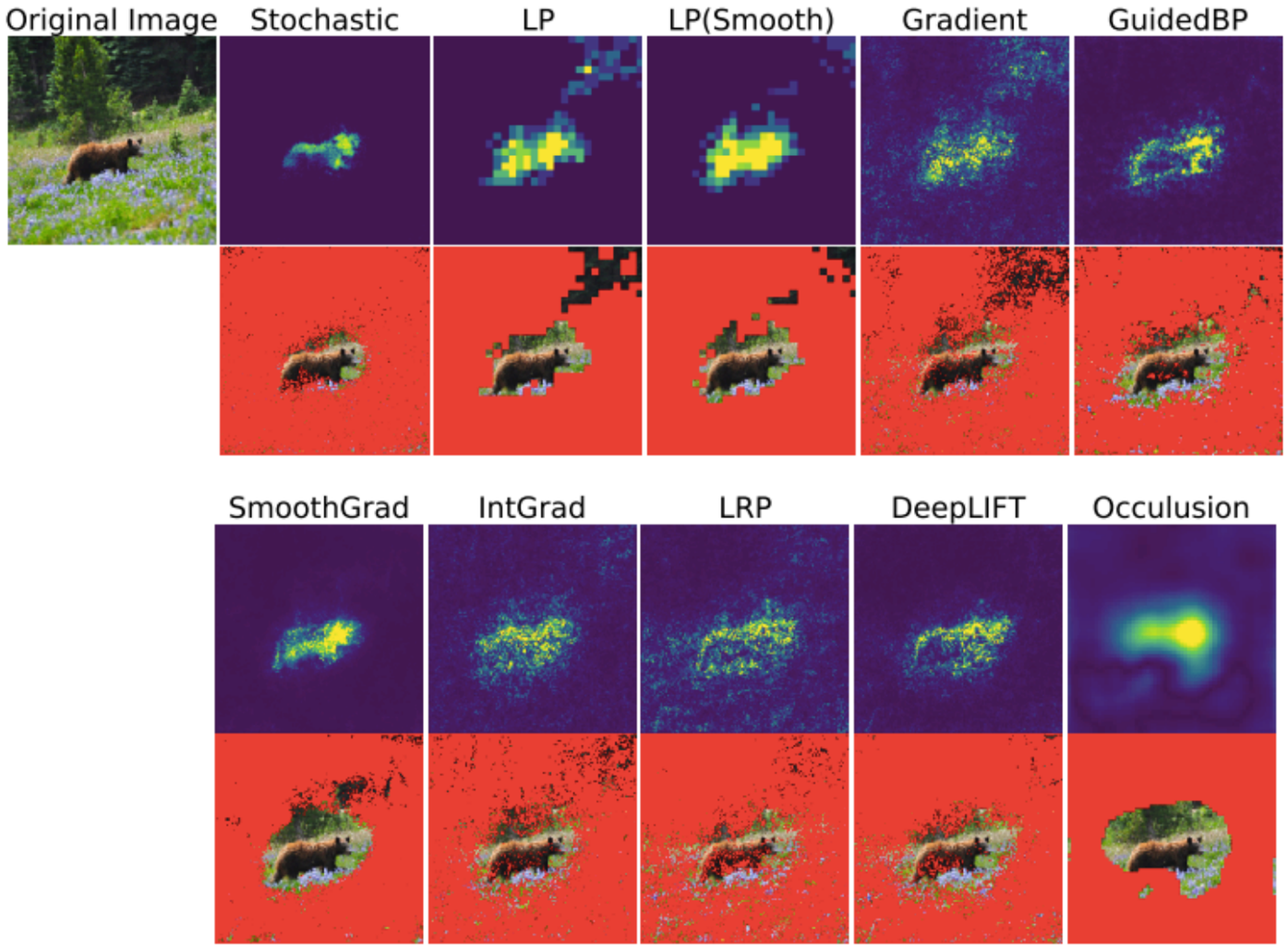
- 過学習しにくい
  - パラメータが多くなると過学習しやすそう？
  - NN においては成り立たない
    - SGDが過学習を防ぐ（Advani and Saxe, 2017）

- validation loss が増えても学習を続けると汎化する
  - Soudry et al., 2017



Sensitivity map, 可視化

- ニューラルネットワークが入力のどの部分を重視するか
  - 入力層の勾配の大きさを使うのが基本



(Ikeno and Hara, 2018)

分散深層学習

- HPCで高速に学習させる
- ミニバッチのサイズを大きくしたい
- 大きすぎると最急降下法と同じように局所解に落ちる
  - バッチサイズと学習率を比例させると上手くいく

- Goyal et al., 2017

TABLE I
TRAINING TIME AND TOP-1 VALIDATION ACCURACY WITH RESNET-50 ON IMAGENET

	Batch Size	Processor	DL Library	Time	Accuracy
He et al. [1]	256	Tesla P100 × 8	Caffe	29 hours	75.3 %
Goyal et al. [2]	8,192	Tesla P100 × 256	Caffe2	1 hour	76.3 %
Smith et al. [3]	8,192 → 16,384	full TPU Pod	TensorFlow	30 mins	76.1 %
Akiba et al. [4]	32,768	Tesla P100 × 1,024	Chainer	15 mins	74.9 %
Jia et al. [5]	65,536	Tesla P40 × 2,048	TensorFlow	6.6 mins	75.8 %
Ying et al. [6]	65,536	TPU v3 × 1,024	TensorFlow	1.8 mins	75.2 %
Mikami et al. [7]	55,296	Tesla V100 × 3,456	NNL	2.0 mins	75.29 %
This work	**81,920**	**Tesla V100 × 2,048**	**MXNet**	**1.2 mins**	**75.08%**

(Yamazaki et al., 2019)

ニューラルネットワークの応用例

- 様々な分野への応用を紹介
- 詳細は省略します

Generative Adversarial Network

- 敵対的生成ネットワーク
- 生成モデルと識別モデルを同時に学習する

- 生成モデルは識別モデルを騙しやすいデータを生成
- 識別モデルは本物のデータと生成されたデータを識別

- 精緻な画像を生成できることで有名に
- 安定して学習させるのが難しい
  - e.g. 高解像度

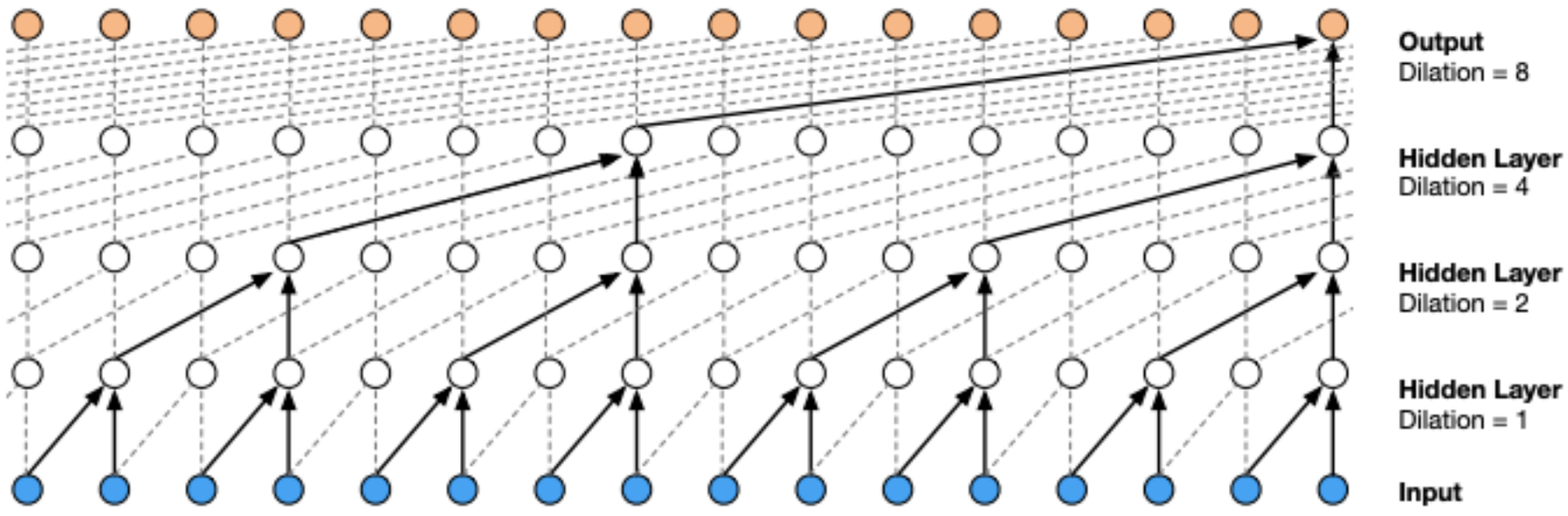
Generative Adversarial Network の例

(Karras et al., 2018)

(Wang et al., 2018)

音声合成

- Wavenet（van den Oord et al., 2016）
- “Dilated causal convolution layers” によって長期の関係性を学習



深層強化学習

- AlphaZero（David et al., 2017）
  - チェスと囲碁と将棋を同じモデルで学習
  - 全てでSOTA（最高性能）を達成
  - 人間の棋譜を一切用いていない

- 最善手の予測と評価関数をまとめる
- モンテカルロ木探索
- 大量の計算資源

自然言語処理

- BERT（Devlin et al., 2018）
- Transformer というモデルの拡張
  - “Attention is all you need”, Vaswani et al., 2017
- 双方向 Transformerを提案
- 様々な自然言語処理の問題でSOTAを達成

- “Pre-training”：事前学習
  - それぞれの問題に対して微調整 “Fine-tuning”

まとめ

- 様々な手法が提案されている
  - データの性質を暗に仮定していたりするので、適切な手法を使い分けられることが重要

- 機械学習の結果を安易に信用するのは危険
  - 過学習, バイアス等々

- 性能向上以外にも様々な研究がある
  - 理論的研究, 可視化, 高速化等々