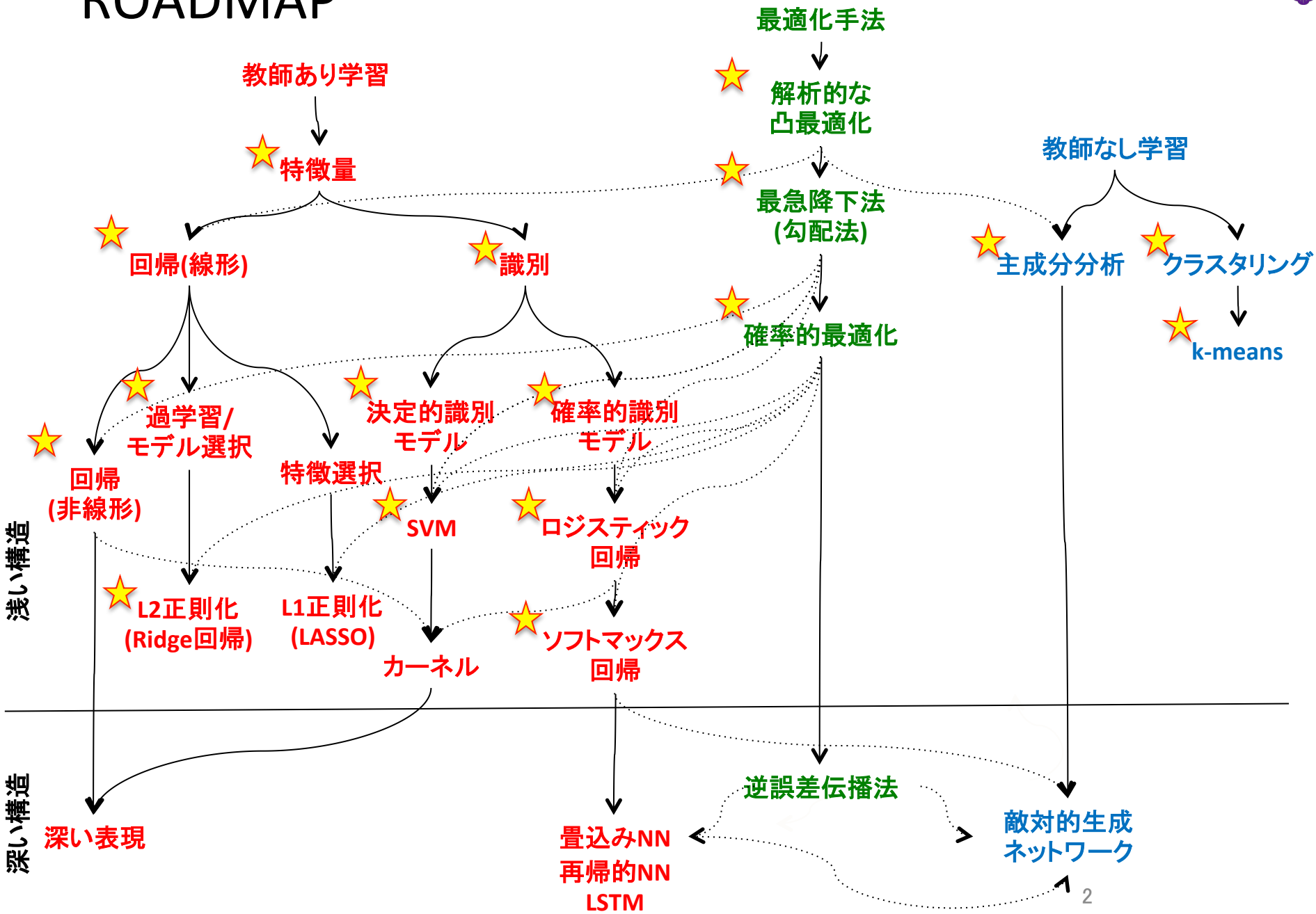
# 機械学習(9)
# ニューラルネットワーク1 基礎とCNN

情報科学類　佐久間　淳

# ROADMAP

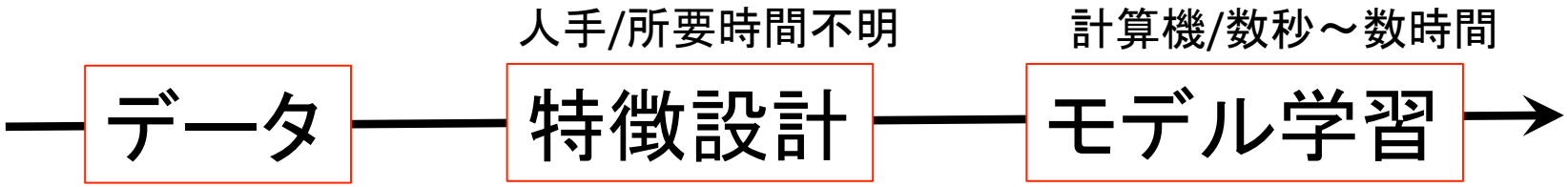

最適化手法
教師あり学習
解析的な
凸最適化
特徴量
教師なし学習
最急降下法
(勾配法)
回帰(線形)
識別
主成分分析
クラスタリング
確率的最適化
k-means
過学習/
モデル選択
決定的識別
モデル
確率的識別
モデル
回帰
(非線形)
特徴選択
SVM
ロジスティック
回帰
浅い構造
L2正則化
(Ridge回帰)
L1正則化
(LASSO)
ソフトマックス
回帰
カーネル
逆誤差伝播法
深い構造
深い表現
畳込みNN
再帰的NN
LSTM
敵対的生成
ネットワーク

# 機械学習から深層学習へ

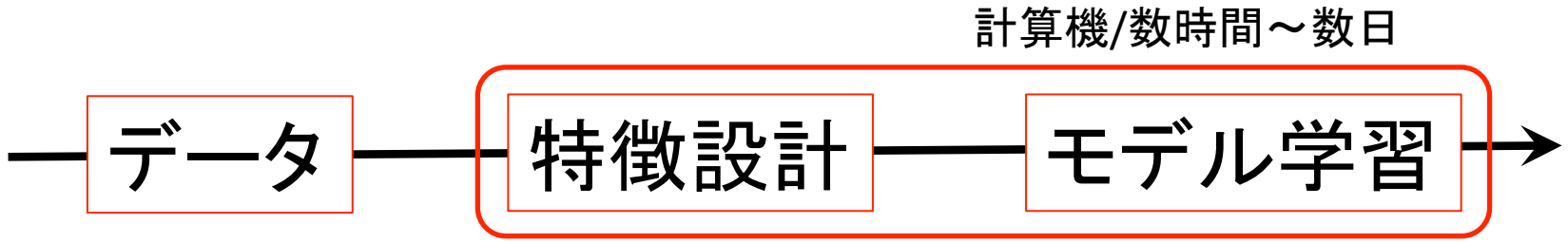
- これまで機械学習は線形モデル $\mathbf{W}^T\mathbf{X}$ をベースにしていた
- また機械学習の前提は「よい特徴量が得られること」であった
- 一方で、「どうすればよい特徴量が得られるか」は画像、音声、自然言語など、各領域の人手による工夫に依存していた
- 深層学習では、「どうすればよい特徴量が得られるか」を、学習ベースで獲得するための試みである
- まだ成功を収めてから若い分野でもあり、理論的裏付けが少ない手法であり、「それはなぜか？」に答えられないことも多い…

# 規模感

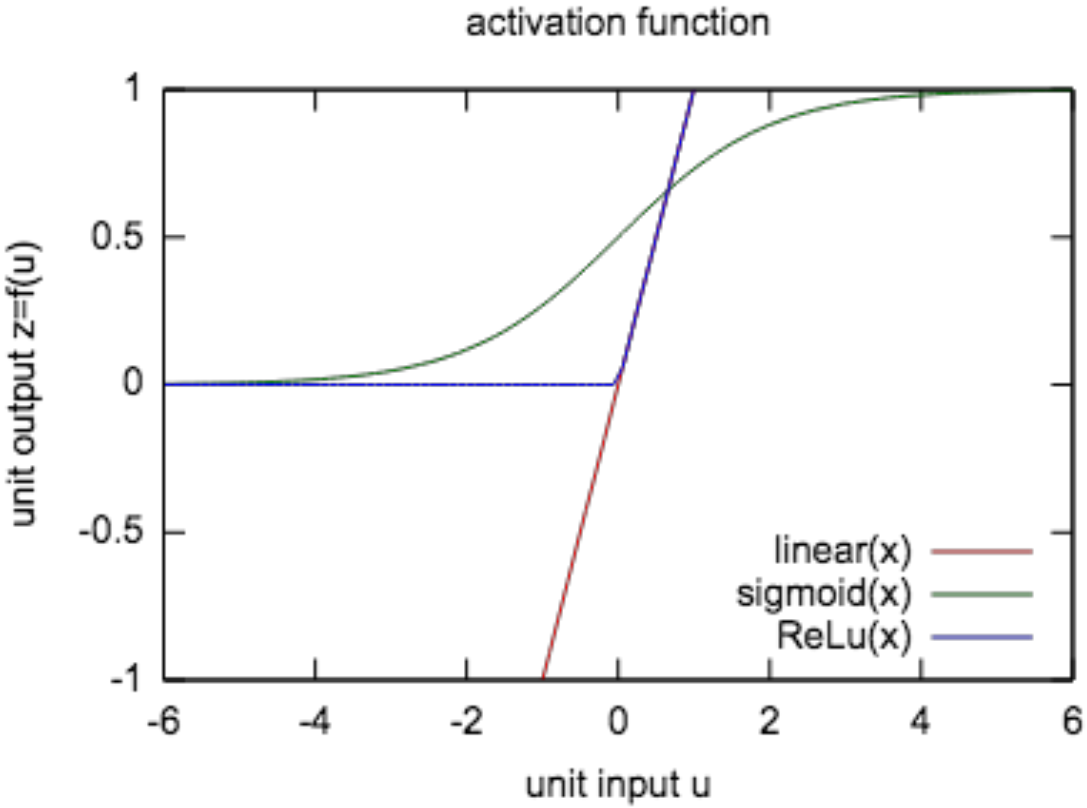
- 機械学習
  - 数千から数万サンプル
  - 通常のCPUで学習時間は<24hrs



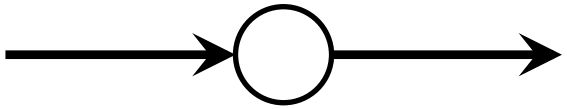

- 深層学習
  - 100万サンプル (e.g, 1000クラス、1000枚/クラス)
  - GPUを多数使って数時間～数日の学習期間



# ユニットと活性化関数




$$z = f(u)$$



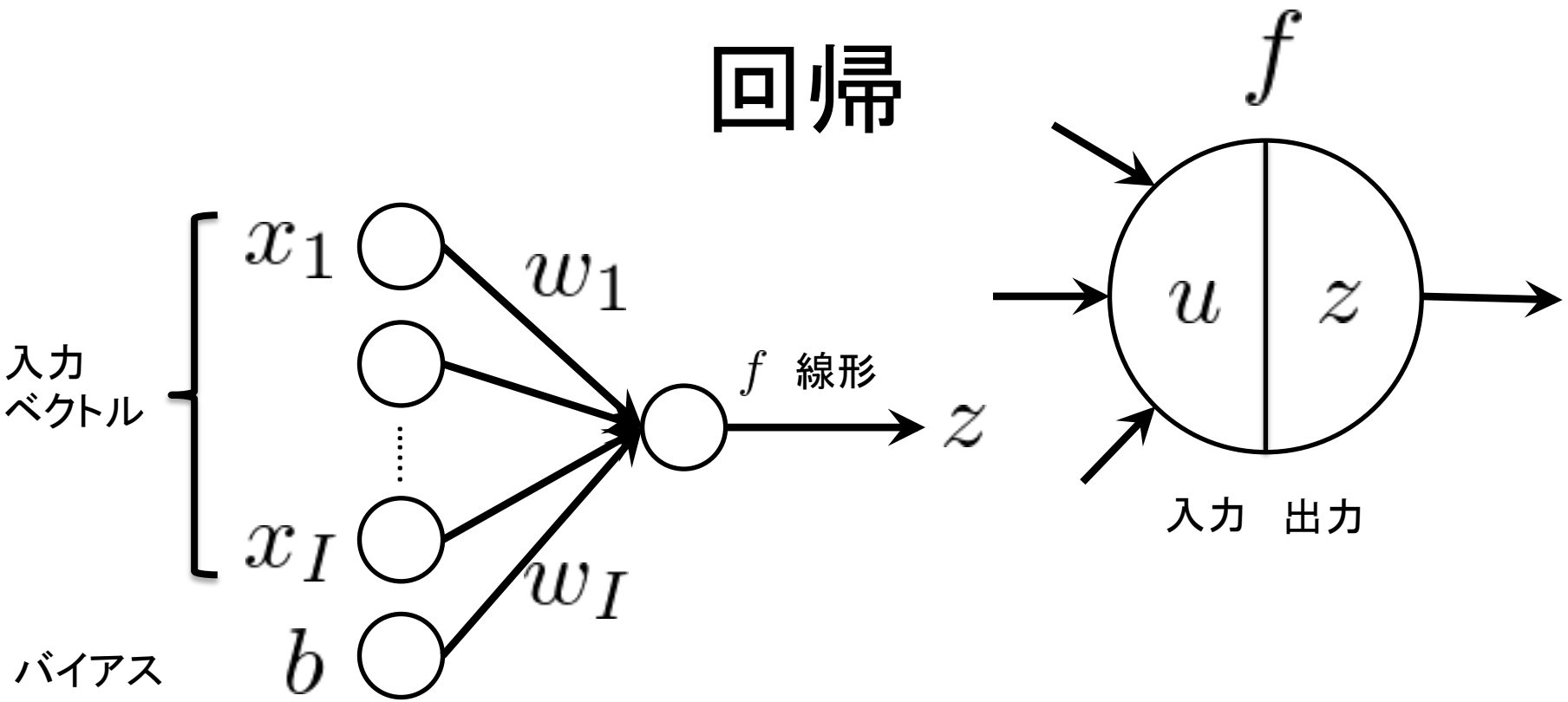
- ユニット: 神経細胞に見立てて、複数の入力信号を受け取り、単一の出力信号を出力する
- 活性化関数: 入力を活性化関数で変換して、その結果を出力する (神経細胞の反応を模倣)

# 活性化関数

- Linear $f(u) = u$
  - 恒等写像
  - 出力は[-∞, ∞], 回帰の最終層に使う
- Sigmoid/softmax $f(u) = \frac{1}{1+e^{-u}}$ $f_k(u) = \frac{e^{u_k}}{\sum_j e^{u_j}}$
  - 微分可能
  - 出力は[0,1]
  - 分類の最終層に使う
- ReLu (rectified liner unit) $f(u) = \max\{0, u\}$
  - 微分不可能点がある
  - 出力は[0,∞]
  - 中間層に使う

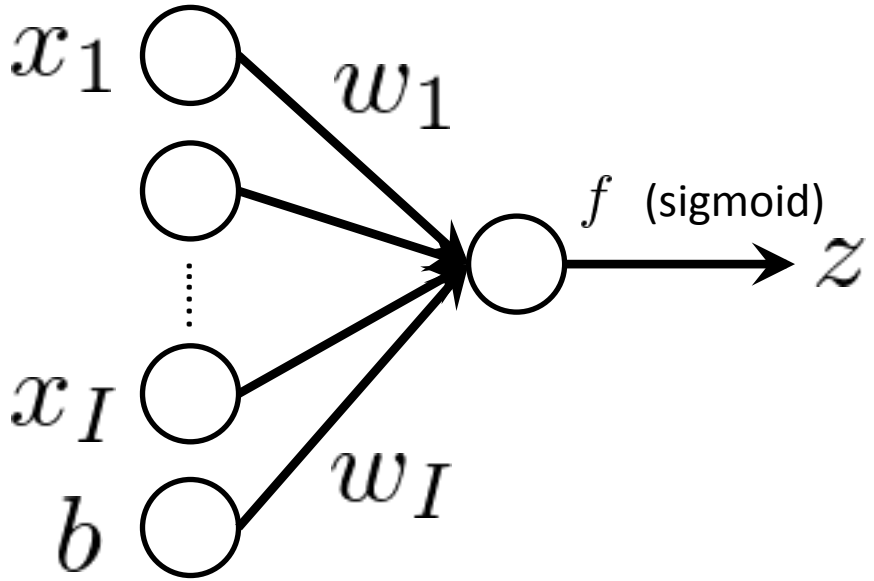
活性化関数は非線形であることが重要→why?

# 回帰

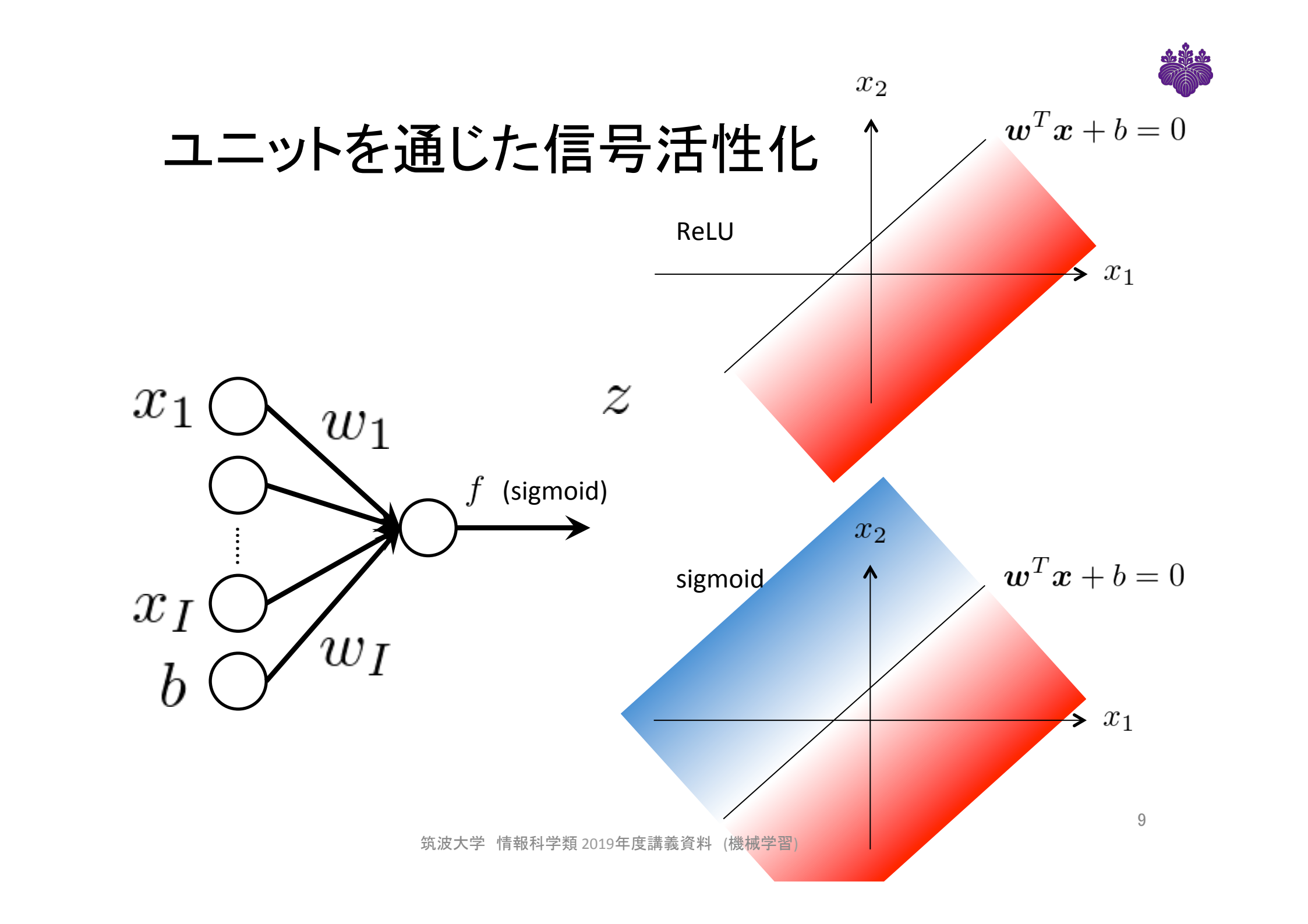



- 入力 $\mathbf{x}^T = (x_1, \ldots, x_I)$
- unit入力 $u = w_1 x_1 + \ldots + w_I x_I + b$
- 出力 $z = f(u)$
- 活性化関数 $u = f(u)$

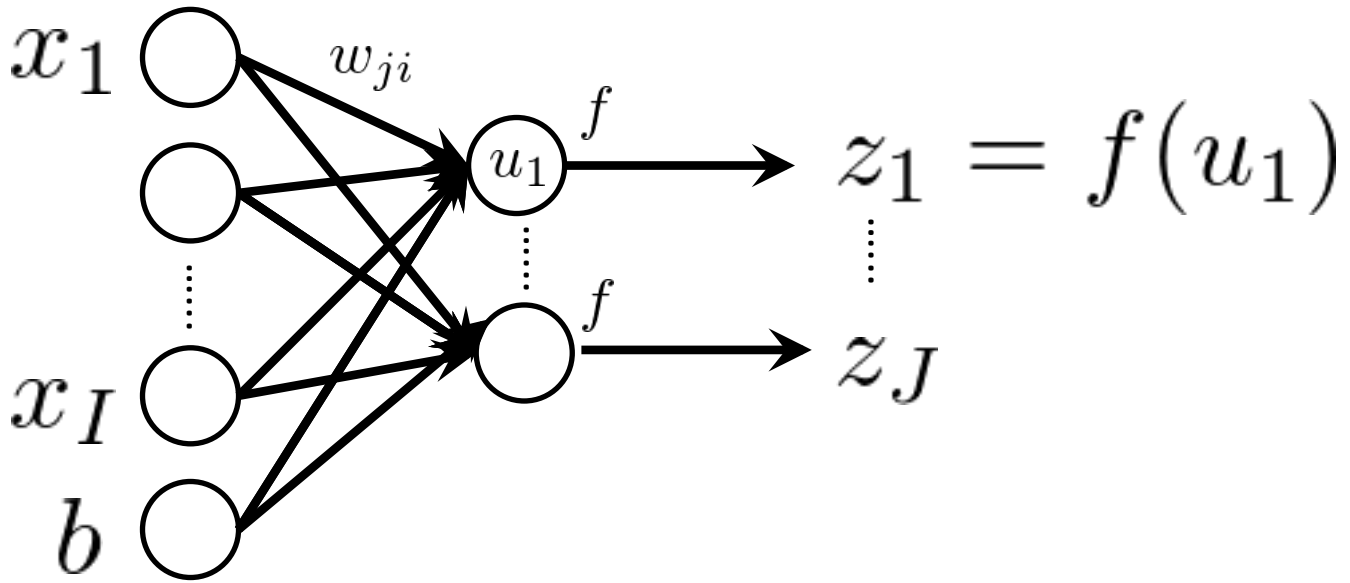
# ロジスティック回帰




- 入力 $\mathbf{x}^T = (x_1, \ldots, x_I)$
- unit入力 $u = w_1 x_1 + \ldots + w_I x_I + b$
- 出力 $z = f(u)$
- 活性化関数 $f(u) = \dfrac{1}{1 + e^{-u}}$

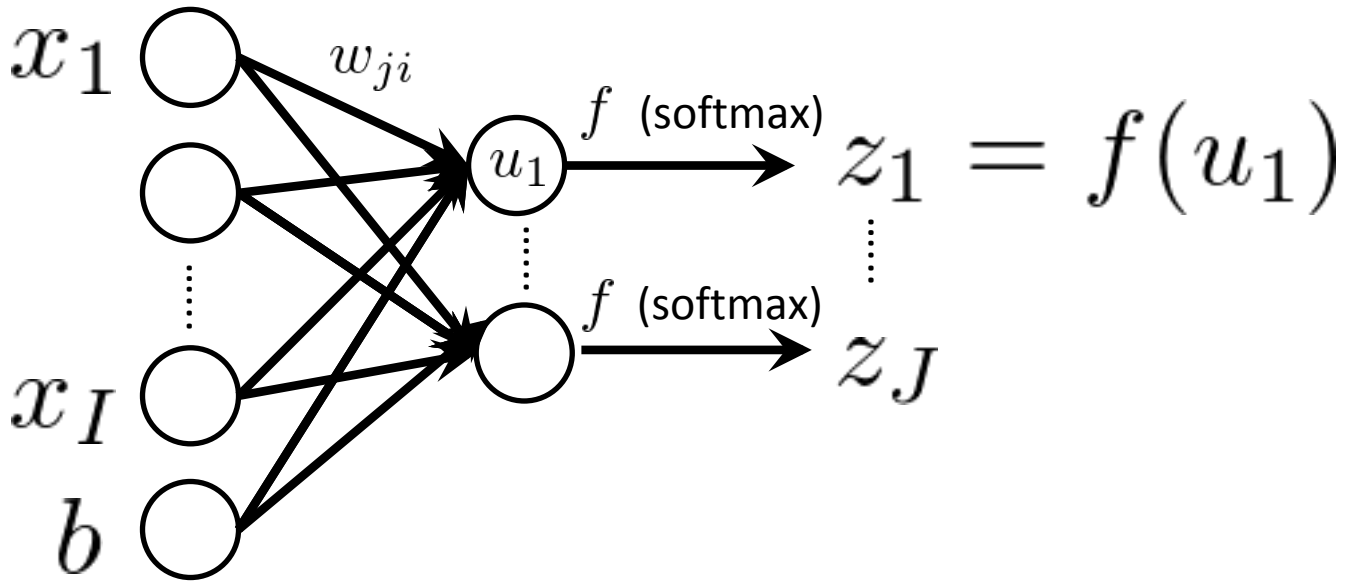

ユニットを通じた信号活性化
$x_1$
$w_1$
$x_I$
$b$
$w_I$
$z$
$f$ (sigmoid)
$x_2$
$\boldsymbol{w}^T\boldsymbol{x}+b=0$
ReLU
$x_1$
$x_2$
sigmoid
$\boldsymbol{w}^T\boldsymbol{x}+b=0$
$x_1$

# ベクトルを出力するネットワーク



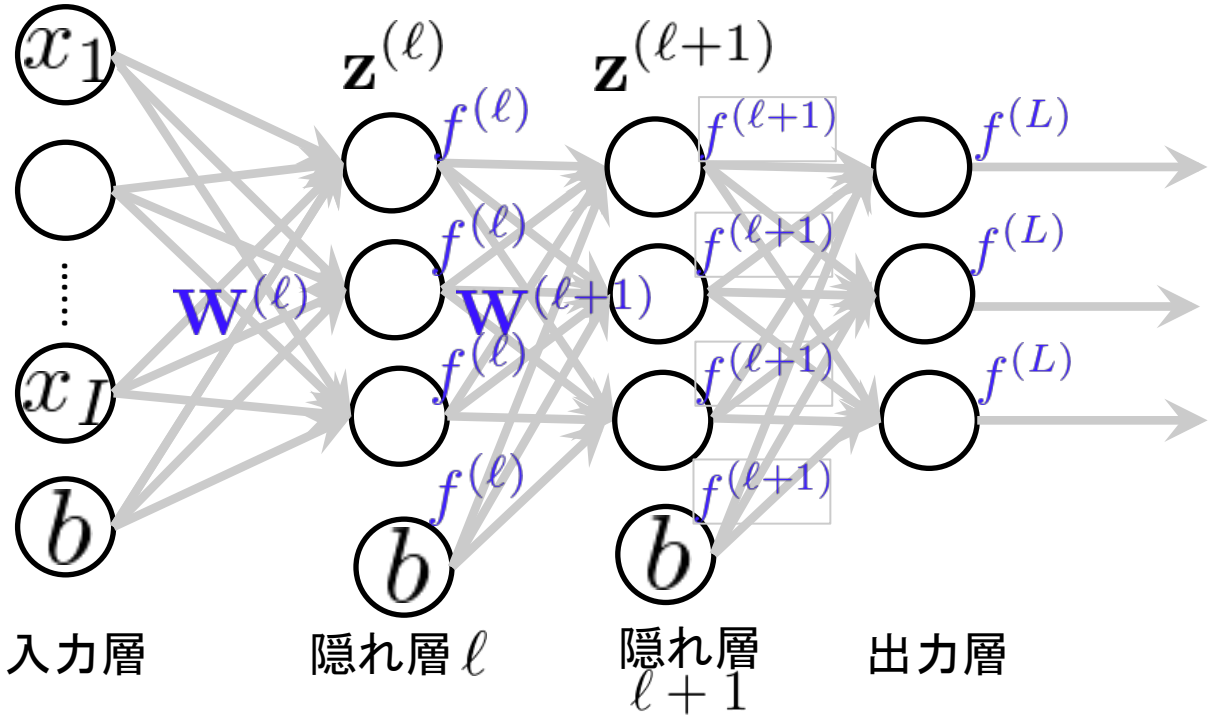

- 入力層第iユニット-第二層第jユニット間の重み $w_{ji}$
- 第二層の計算 $\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$
- 出力 $\mathbf{z} = \mathbf{f}(\mathbf{u})$

# ソフトマックス回帰



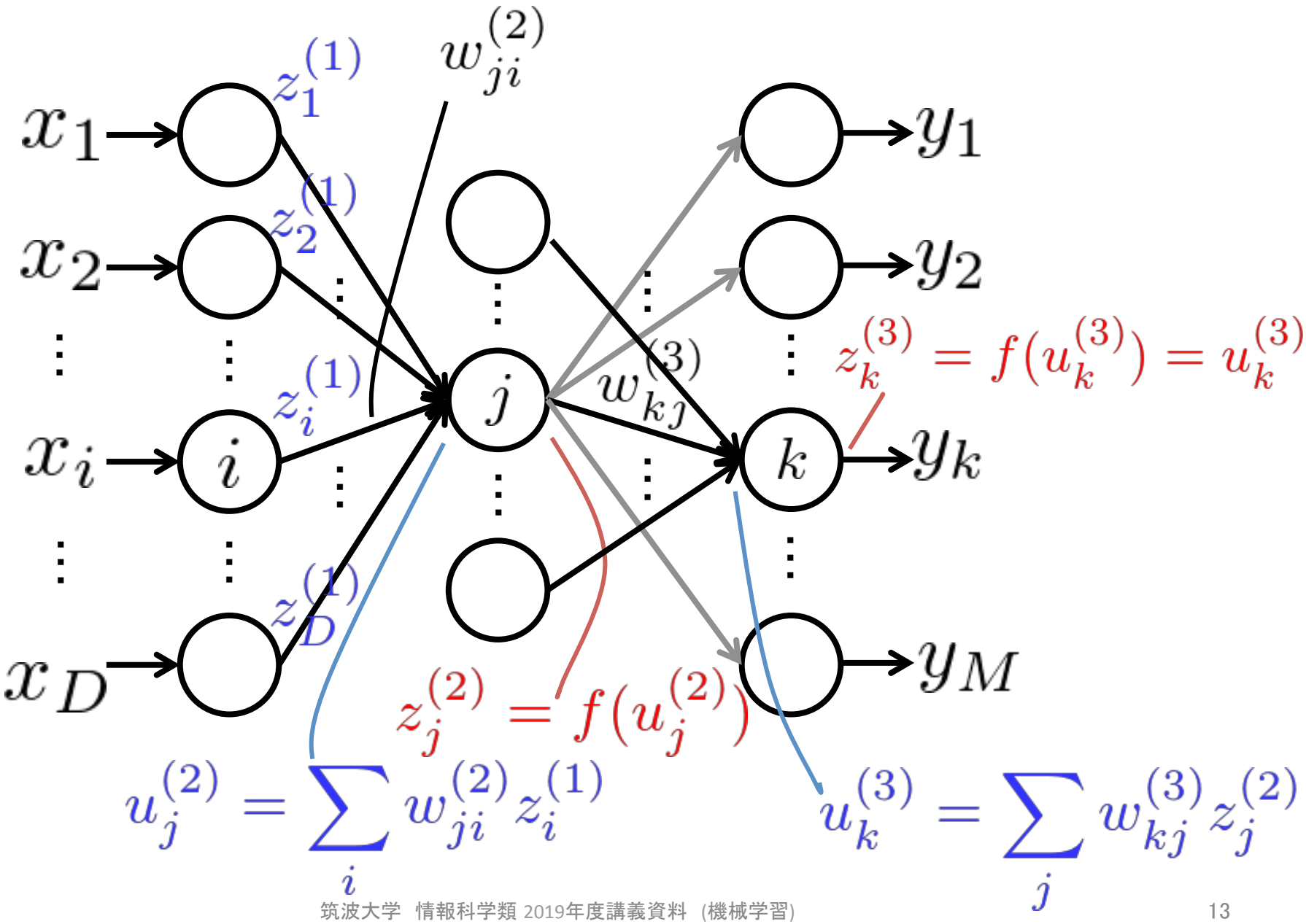

- 入力層第iユニット-第二層第jユニット間の重み$w_{ji}$
- 第二層の計算 $\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$
- 出力 $\mathbf{z} = \mathbf{f}(\mathbf{u})$ $f_k(u) = \dfrac{e^{u_k}}{\sum_j e^{u_j}}$

# 多層ニューラルネットワーク

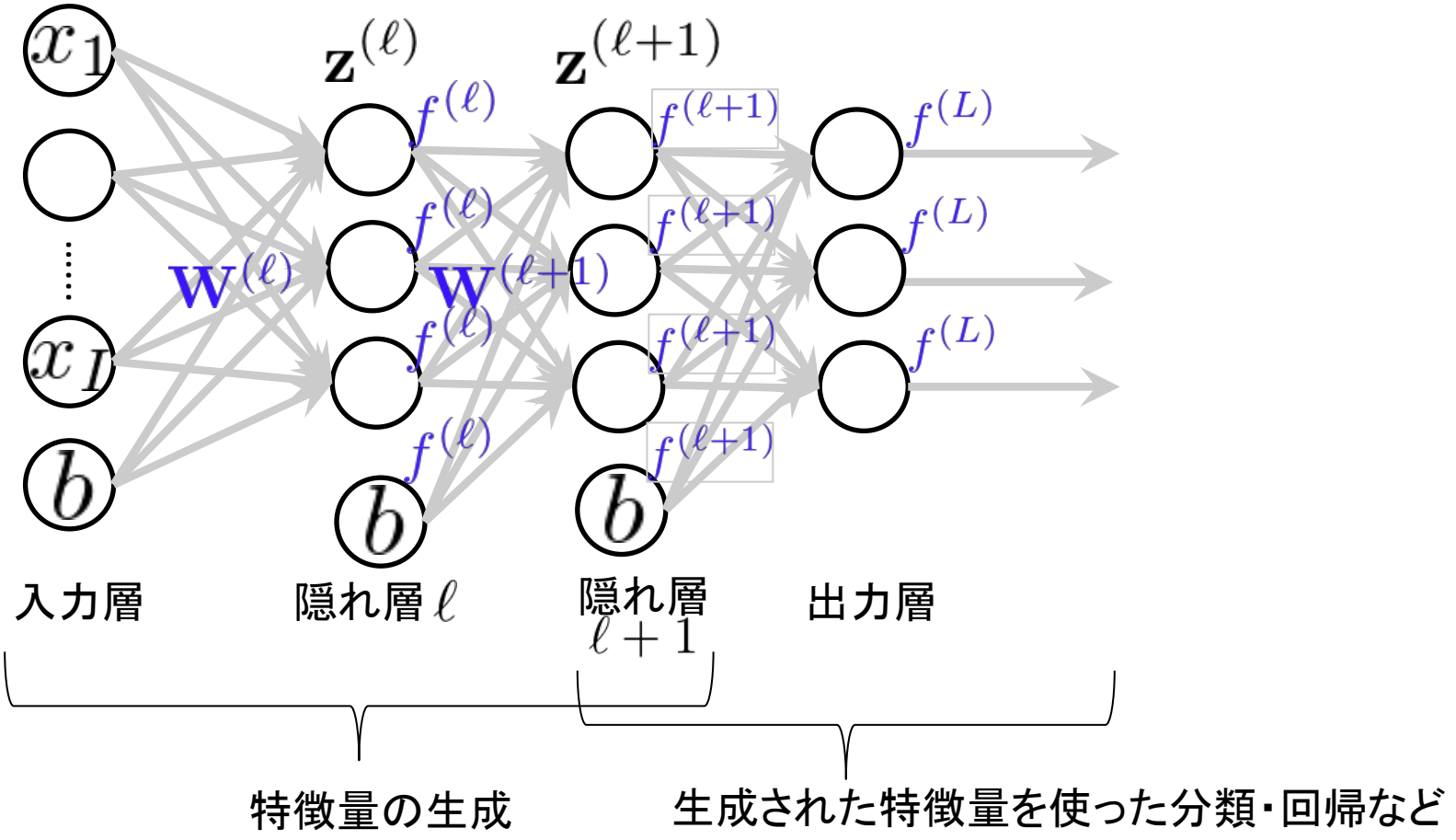



- 入力層( $\ell = 1$), 隠れ層( $\ell = 2 \ldots L-1$ ), 出力層($\ell = L$ )
- 第 $\ell$ 層の重み行列$\mathbf{W}^{(\ell)}$
- 第 $\ell - 1$ 層ノードi と第 層ノードj 間の重み $w_{ji}^{(\ell)}$
- 順伝播:入力層から出力層に向けて、各層に信号を順方向に伝播させ、非線形活性化関数を適用し、出力信号を得る

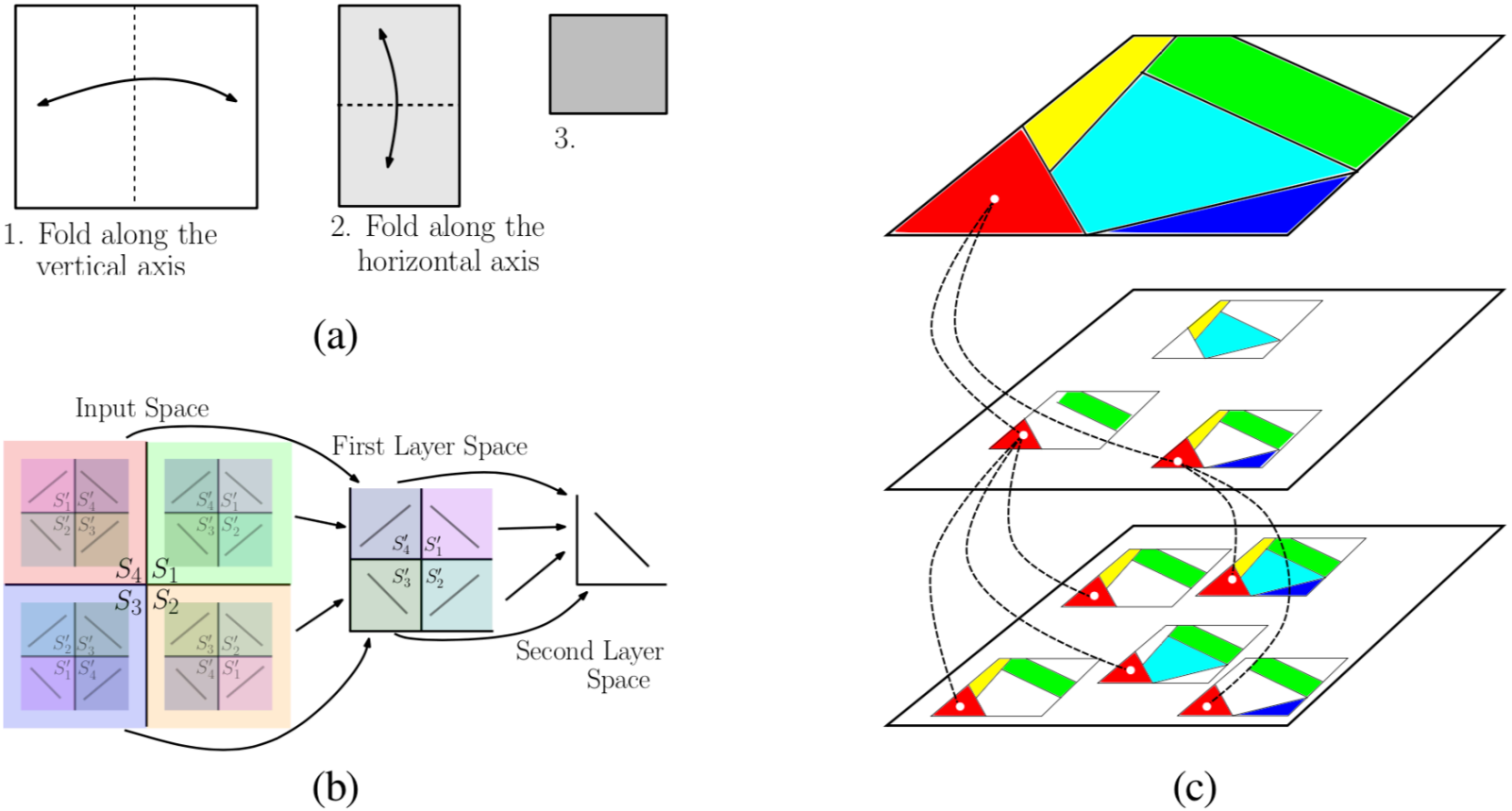
$$\mathbf{z}^{(\ell+1)} = f(\mathbf{W}^{(\ell+1)}\mathbf{z}^{(\ell)} + \mathbf{b}^{\ell+1})$$


$x_1$
$x_2$
$x_i$
$x_D$
$z_1^{(1)}$
$z_2^{(1)}$
$z_i^{(1)}$
$z_D^{(1)}$
$w_{ji}^{(2)}$
$j$
$w_{kj}^{(3)}$
$k$
$y_1$
$y_2$
$y_k$
$y_M$
$z_k^{(3)} = f(u_k^{(3)}) = u_k^{(3)}$
$z_j^{(2)} = f(u_j^{(2)})$
$u_j^{(2)} = \sum_i w_{ji}^{(2)} z_i^{(1)}$
$u_k^{(3)} = \sum_j w_{kj}^{(3)} z_j^{(2)}$

# 多層ニューラルネットワークの構造




- 第一層から最終層手前までは特徴量生成関数
- 最後の一層だけ切り取ると、回帰やsoftmax回帰(分類)




# なぜ多層に？

Figure 2: (a) Space folding of 2-D Euclidean space along the two coordinate axes. (b) An illustration of how the top-level partitioning (on the right) is replicated to the original input space (left). (c) Identification of regions across the layers of a deep model.



On the Number of Linear Regions of Deep Neural Networks, Montufar+, NIPS14

再掲

# 交互作用項を持つ多項式特徴量

- 2次元2次多項式特徴量

$$\boldsymbol{\phi}(\boldsymbol{x})^T = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

- 2次元3次多項式特徴量

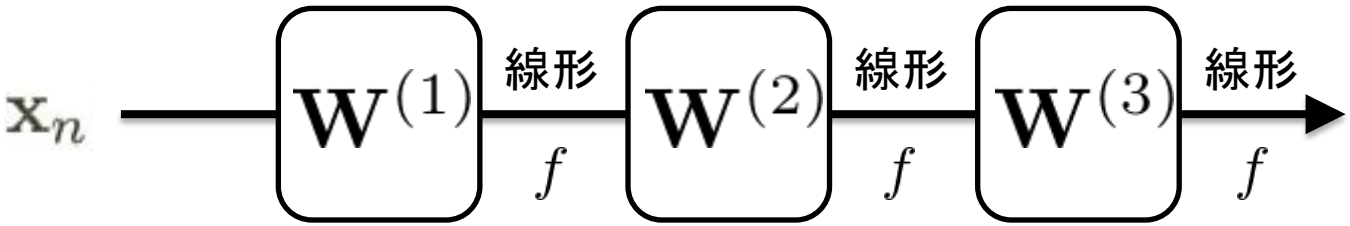
$$\boldsymbol{\phi}(\boldsymbol{x})^T = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1^2 x_2, x_1 x_2^2)$$

- 3次元2次多項式特徴量

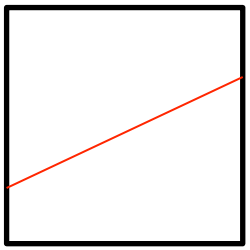
$$\boldsymbol{\phi}(\boldsymbol{x})^T = (1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1 x_2, x_2 x_3, x_1 x_3)$$

- 特徴量同士の積(交互作用項)が含まれることに注意

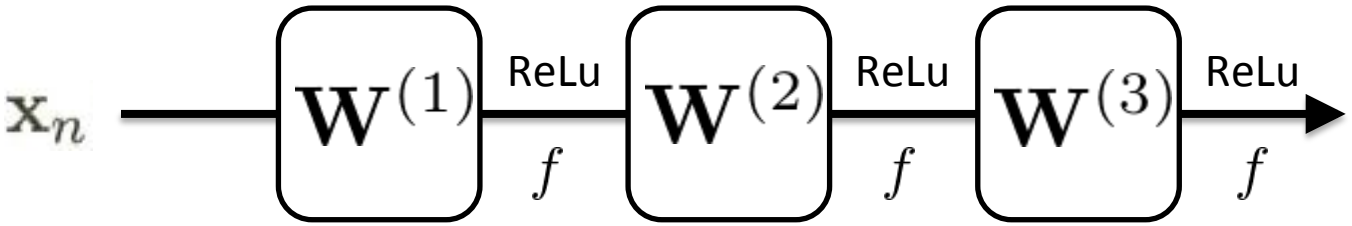
# なぜ非線形な活性化関数?



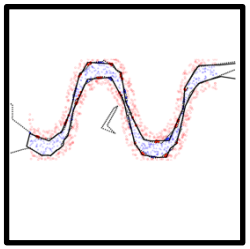

$$\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}\mathbf{x}_n$$



結局表現されるのは線形関数



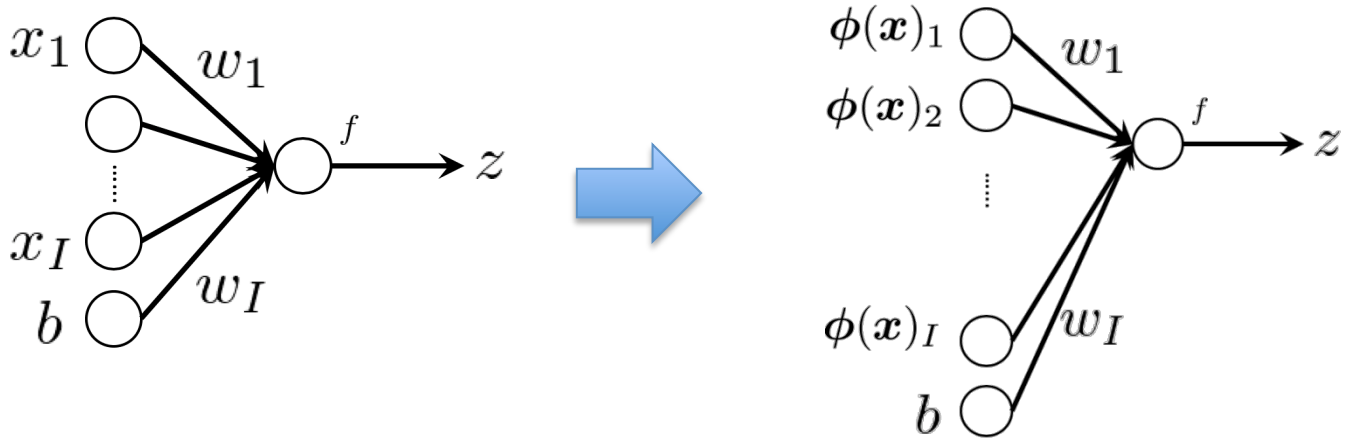

$$f(\mathbf{W}^{(3)}f(\mathbf{W}^{(2)}f(\mathbf{W}^{(1)}\mathbf{x}_n)))$$



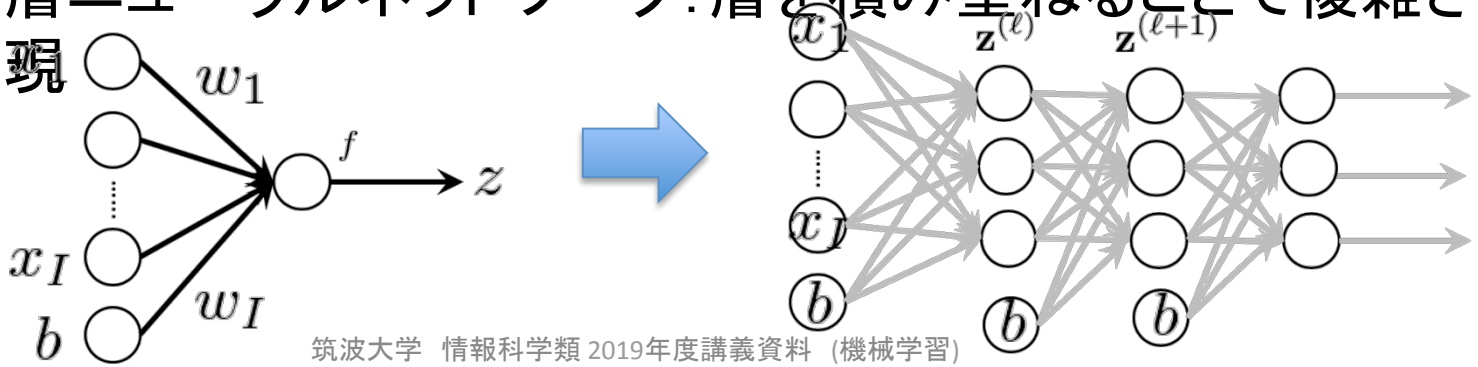
層を増すごとに複雑な関数を表現

# 従来の特徴量と多層NNの違い

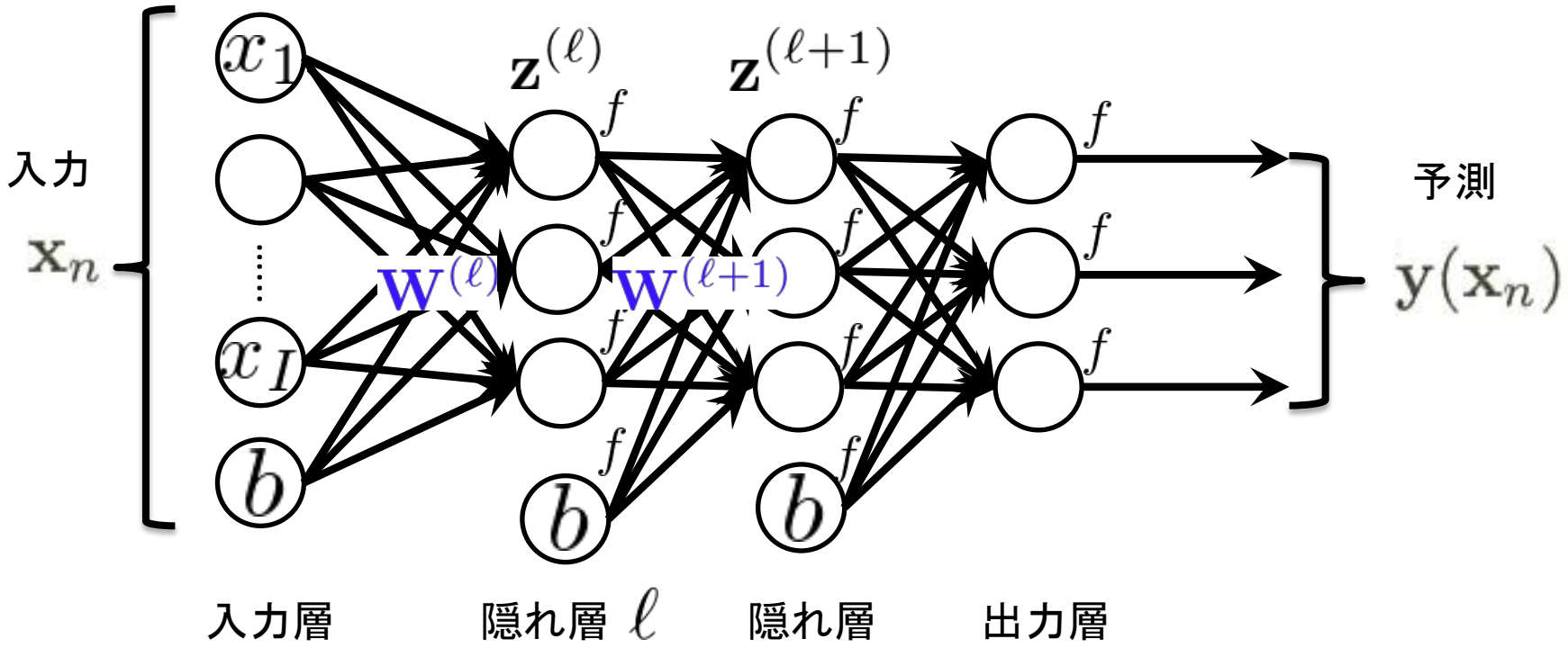
- 従来の特徴量：特徴次元数を高次元化することで複雑さを表現




- 多層ニューラルネットワーク：層を積み重ねることで複雑さを表現



# 多層ニューラルネットの損失関数




- 訓練データ(x,y)と出力y(x)について, 損失関数loss(y, y(x))を最小化するように、各層のwを学習
- 損失関数 (学習対象)
  - 訓練誤差としての二乗誤差、クロスエントロピー、ヒンジ損失、etc
- 評価関数 (性能評価、必ずしも学習対象と一致しない)
  - 訓練誤差あるいはテスト誤差としての二乗誤差、分類の正解率、など

# 損失・評価関数セッティング例 (Keras)

損失関数

```
from keras import losses

model.compile(loss=losses.mean_squared_error, optimizer='sgd')
```

mean_squared_error = 平均二乗損失

評価関数

```
from keras import metrics

model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=[metrics.mae, metrics.categorical_accuracy])
```

mae=mean absolute error (L1誤差)、categorical accuracy (正解率)

# ニューラルネットワークの最適化

- 損失関数の(劣)勾配による最適化

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla E(\mathbf{w}^{(t)})$$

- 確率的(劣)勾配降下法

$$\nabla E_{\boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \mathrm{loss}(\boldsymbol{y}_n, \boldsymbol{y}(\boldsymbol{x}_n))$$

- ミニバッチ

$$\nabla E_{\boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \sum_{n \in D} \mathrm{loss}(\boldsymbol{y}_n, \boldsymbol{y}(\boldsymbol{x}_n))$$

- (劣)勾配の計算→逆誤差伝播法 (あとで)

再掲

# 確率的勾配降下法

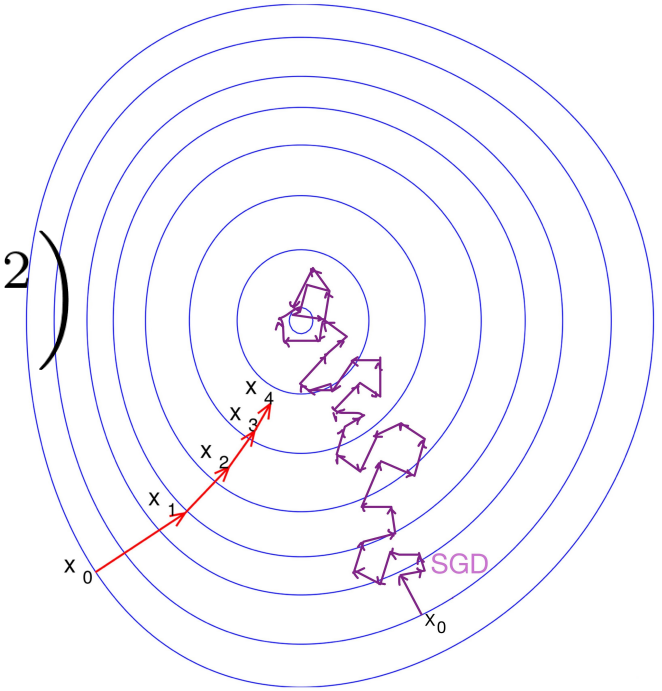
(stochastic gradient method, SDG)

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla E(\mathbf{w}^{(t)})$$
$$t \leftarrow t+1 \text{ until convergence}$$

- ランダムに選んだ一つの事例の訓練誤差について勾配を計算

$$\nabla E(\boldsymbol{w}^{(t)}) = \nabla \left( (\boldsymbol{w}^{(t)})^T \boldsymbol{x}_n - t_n \right)^2$$

- E(w)の勾配という意味では近似的だが、更新あたりの計算量はサンプル数Nについて$O(1)$
- 全データに対する勾配との乖離があり、最適化の過程は不安定(訓練誤差は単調減少しない)




https://commons.wikimedia.org/wiki/File:Gradient_descent.svg

再掲

# ミニバッチ

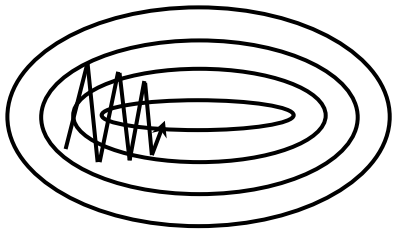
$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla E(\mathbf{w}^{(t)})$$
$$t \leftarrow t+1 \text{ until convergence}$$

- ミニバッチ $D_t$
  - 全データから比較的少数のデータをランダムに選んだ集合
- ランダムに選んだミニバッチの予測誤差について勾配計算

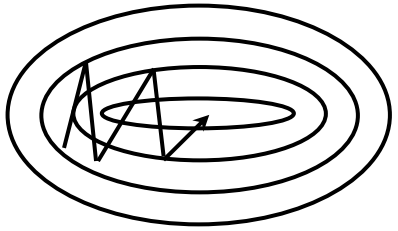
$$\nabla E(\boldsymbol{w}^{(t)}) = \nabla \left( \sum_{(\boldsymbol{x},t) \in D_t} [(\boldsymbol{w}^{(t)})^T \boldsymbol{x} - t]^2 \right)$$

- 更新あたりの計算量はミニバッチサイズに比例
- 勾配はミニバッチについて計算するためSGDほど不安定ではない
- 並列計算器資源の有効活用
  - バッチごとに独立に勾配を計算

# モメンタム



Momentumが無いSGD



Momentumが有るSGD

- ridge付近で最適化が停滞しやすい
- モメンタムのアイディア
  - パラメータ更新量に、前回の更新量の成分を少しブレンド

前回と今回のパラメータ差 $\Delta \mathbf{w}^{(t-1)} \leftarrow \mathbf{w}^{(t-1)} - \mathbf{w}^{(t-2)}$

更新式 $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \nabla E_n + \mu \Delta \mathbf{w}^{(t-1)}$

ここまでなら普通の勾配法

  - μはハイパーパラメータ(要調整)

# 正則化/重み減衰

- 正則化の考え方はこれまでのモデルと同じ
- 損失関数＋正則化項

$$E_t(\mathbf{w}) \equiv \frac{1}{N_t} \sum_{n \in \mathcal{D}_t} E_n(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- 勾配法

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \left( \frac{1}{N_t} \sum \nabla E_n + \underbrace{\lambda \mathbf{w}^{(t)}} \right)$$

この項が、重みの大きな変化を抑制する効果があることから、重み減衰(weight decay)とも呼ばれる

# 最適化Setting例 (Keras)

```
from keras import optimizers

model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

sgd=確率的勾配降下法(stochastic gradient descent)
Lr: 学習率(learning rate=ステップサイズパラメータ)
decay: 重み減衰 (weight decay=正則化パラメータ)
momentum: モメンタムのパラメータ
nesterov: ネステロフの加速法を使うかどうか (授業では扱いません)

そのほかの最適化
RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam

# AdaGrad: ステップサイズパラメータの調整

- (劣)勾配法 $w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta \nabla E_i^{(t)}$
  - i番目のパラメータの更新に着目
- ステップサイズパラメータの決め方は難しい
  - 大きいη　少ない更新回数で大きく改善するが精度が出ない
  - 小さいη　多数の更新を要するが精度が出る
- 初めは大きくし、徐々に小さくすればよいのでは？
  - 例えば$\eta^t \leftarrow \eta^t / \alpha t$　(Robbins-Monro)
  - αのチューニングは必要なことが多い
- AdaGrad[Duchi+11], NNの最適化によく使われている

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} \nabla E_i^{(t')} + \epsilon}} \nabla E_i^{(t)}$$

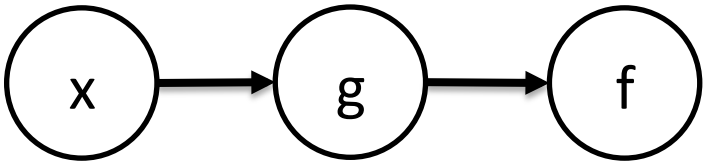
過去の勾配の総和で大きさを調整
これまで更新が少なかったパラメータに大きな学習率を与える
層ごとに学習の進捗が異なるNNの最適化に向いている

# 連鎖率

- 微分法の連鎖律 $$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \frac{dg(x)}{dx}$$

- 多変数関数でも連鎖律は成り立つ

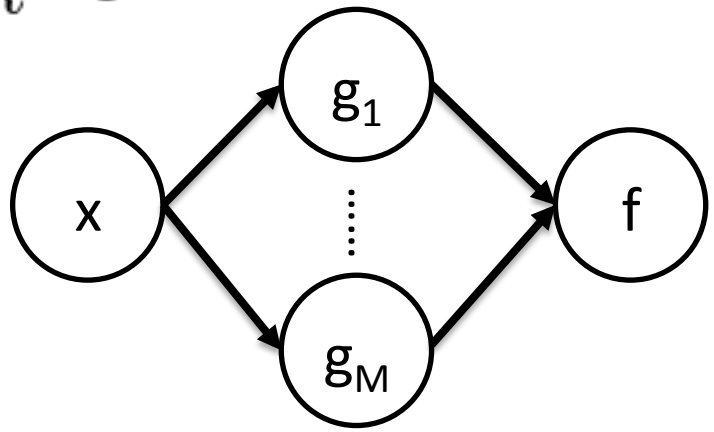
$$\frac{\partial f(g(\boldsymbol{x}))}{\partial \boldsymbol{x}} = \frac{\partial f(g(\boldsymbol{x}))}{\partial g(\boldsymbol{x})} \frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}}$$

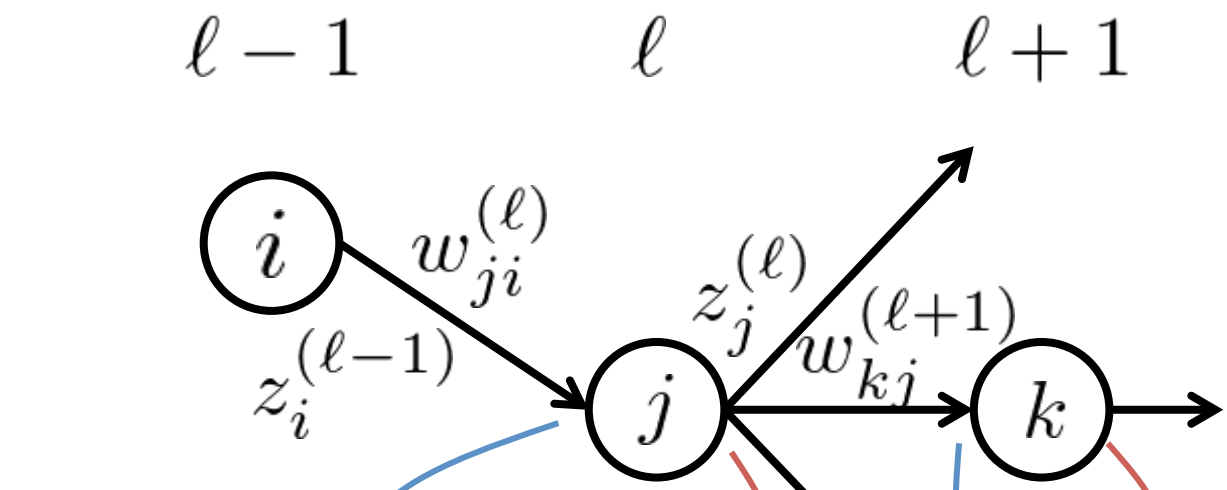


# 多変数関数の連鎖率

- 多変数関数 $f(\mathbf{g}) = f(g_1, g_2, \ldots, g_M)$
- 各$g_k$はxの関数 $g_k(x), k \in \{1, \ldots, M\}$
- このとき

$$\frac{\partial f}{\partial x} = \sum_{k=1}^{M} \frac{\partial f}{\partial g_k} \frac{\partial g_k}{\partial x}$$






$$u_j^{(\ell)} = \sum_i w_{ji}^{(\ell)} z_i^{(\ell-1)}$$

$$z_j^{(\ell)} = f(u_j^{(\ell)})$$

$$u_k^{(\ell+1)} = \sum_j w_{kj}^{(\ell+1)} z_j^{(\ell)}$$

$$z_k^{(\ell+1)} = f(u_k^{(\ell+1)})$$

# 逆誤差伝播法

- 入力:訓練サンプル $\mathbf{x}_n$ および目標ベクトル $\mathbf{d}_n$
- 出力: $\partial E / \partial w_{ji}^{(\ell)}$

1. ネットワークへの入力を $z^{(1)} = \mathbf{x}_n$ とし、各層のユニットの入出力 $u_j^{(\ell)}, z_j^{(\ell)}$ を計算 (順伝播)
2. 出力層において, $\delta_j^{(L)} = z_j - d_j$ を求める
3. 中間層において, $\ell = L-1, L-2, \ldots, 2$ の順に

$$\delta_j^{(\ell)} = \sum_k \delta_k^{(\ell+1)} \left( w_{kj}^{(\ell+1)} f'(u_j^{(\ell)}) \right) \quad \text{(逆伝播)}$$

$$\frac{\partial E_n}{\partial w_{ji}^{(\ell)}} = \delta_j^{(\ell)} z_i^{(\ell-1)}$$

## 9.1　3 層 NN の逆誤差伝播法の導出

訓練サンプル $(\boldsymbol{x}, \boldsymbol{d})$ に対する二乗誤差は以下の通りである。

$$E(\boldsymbol{W}) = \|\boldsymbol{y}(\boldsymbol{x}) - \boldsymbol{d}\|_2^2 \tag{12}$$

ここでは, このニューラルネットワークの確率的勾配降下法を行うために必要な勾配 $\frac{\partial E}{\partial w_{ji}^{(2)}}$ および $\frac{\partial E}{\partial w_{kj}^{(3)}}$ を逆誤差伝播法によって求める。

1. $\frac{\partial E}{\partial w_{kj}^{(3)}}$ を求めよ
2. $\frac{\partial E}{\partial w_{ji}^{(2)}}$ を求めよ

## 9.2　一般的な NN の逆誤差伝播法の導出

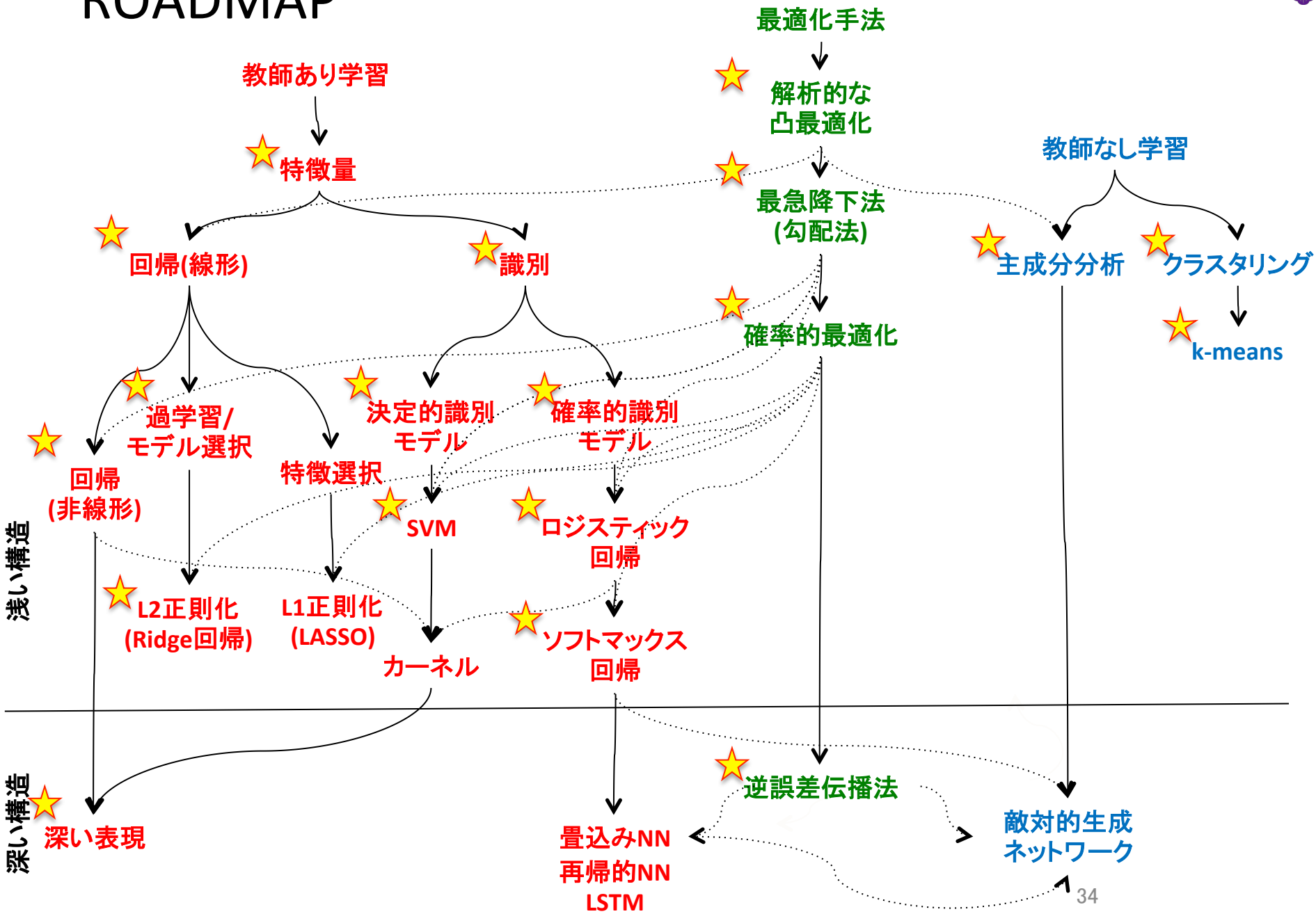
訓練サンプルに対する二乗誤差は以下の通りである。

$$E(\boldsymbol{W}) = \|\boldsymbol{y}(\boldsymbol{x}) - \boldsymbol{d}\|_2^2 \tag{13}$$

ここでは, このニューラルネットワークの確率的勾配降下法を行うために必要な勾配 $\frac{\partial E}{\partial w_{ji}^{(L)}}$ および $\frac{\partial E}{\partial w_{kj}^{(\ell)}}$ を求める。

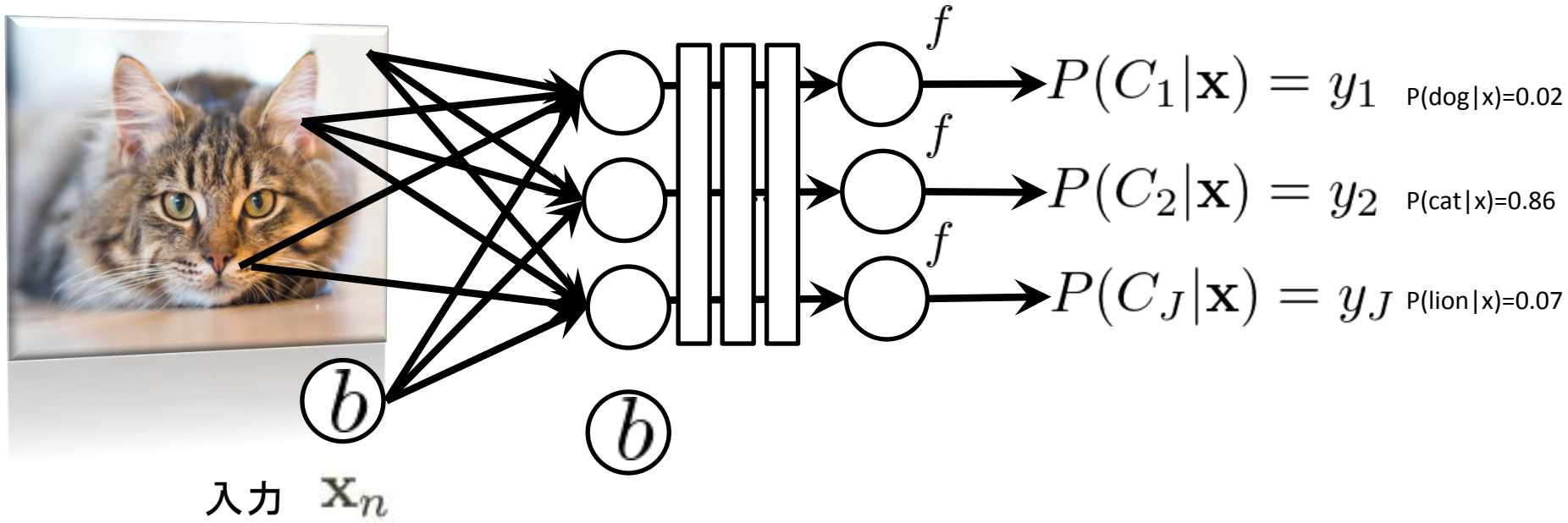
1. $\frac{\partial E}{\partial w_{ji}^{(L)}}$ を求めよ
2. $\frac{\partial E}{\partial u_j^{(\ell)}} \equiv \delta_j^{(\ell)}$ とする。$\frac{\partial E}{\partial w_{ji}^{(\ell)}}$ を $\delta_k^{(\ell+1)}, k = 1, 2, \ldots,$ の式で求めよ

# ROADMAP


最適化手法
教師あり学習
解析的な
凸最適化
特徴量
最急降下法
(勾配法)
教師なし学習
回帰(線形)
識別
主成分分析
クラスタリング
確率的最適化
k-means
過学習/
モデル選択
決定的識別
モデル
確率的識別
モデル
回帰
(非線形)
特徴選択
SVM
ロジスティック
回帰
浅い構造
L2正則化
(Ridge回帰)
L1正則化
(LASSO)
ソフトマックス
回帰
カーネル
逆誤差伝播法
深い構造
深い表現
畳込みNN
再帰的NN
LSTM
敵対的生成
ネットワーク

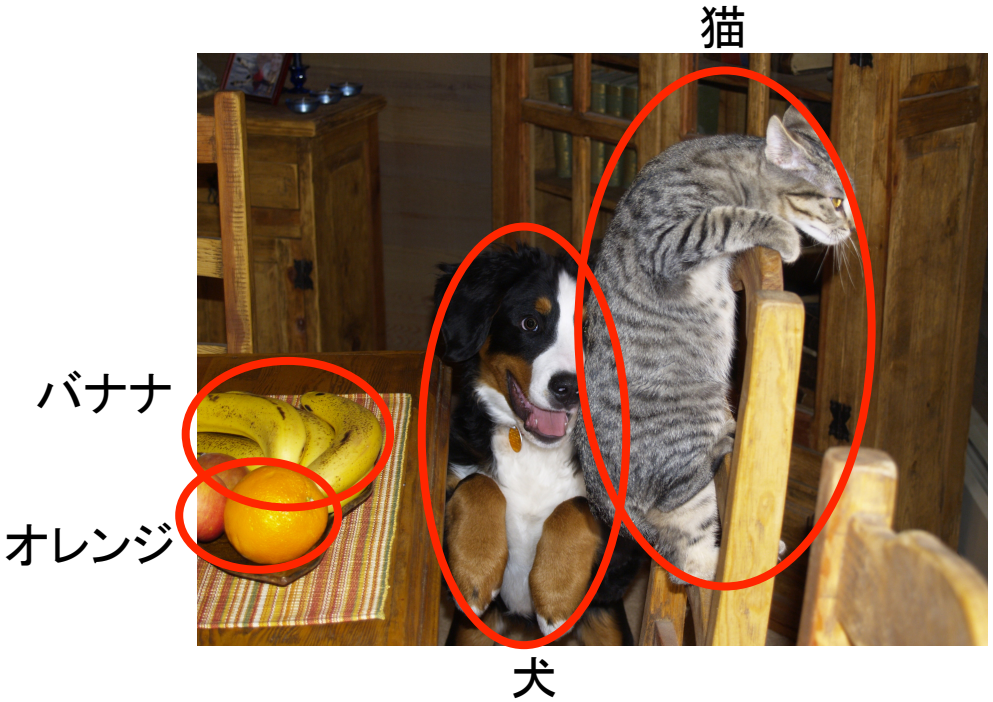
# 畳み込みNNによる画像認識

200*200の画像、40Kユニット
→2Bパラメータ




- 実際の画像認識には局所的な理解で足るはず
  - 目があって、耳があって、ヒゲがあって…
- 全結合は不要
- 画像理解に役立つネットワークの構造を考える

# 一般物体認識





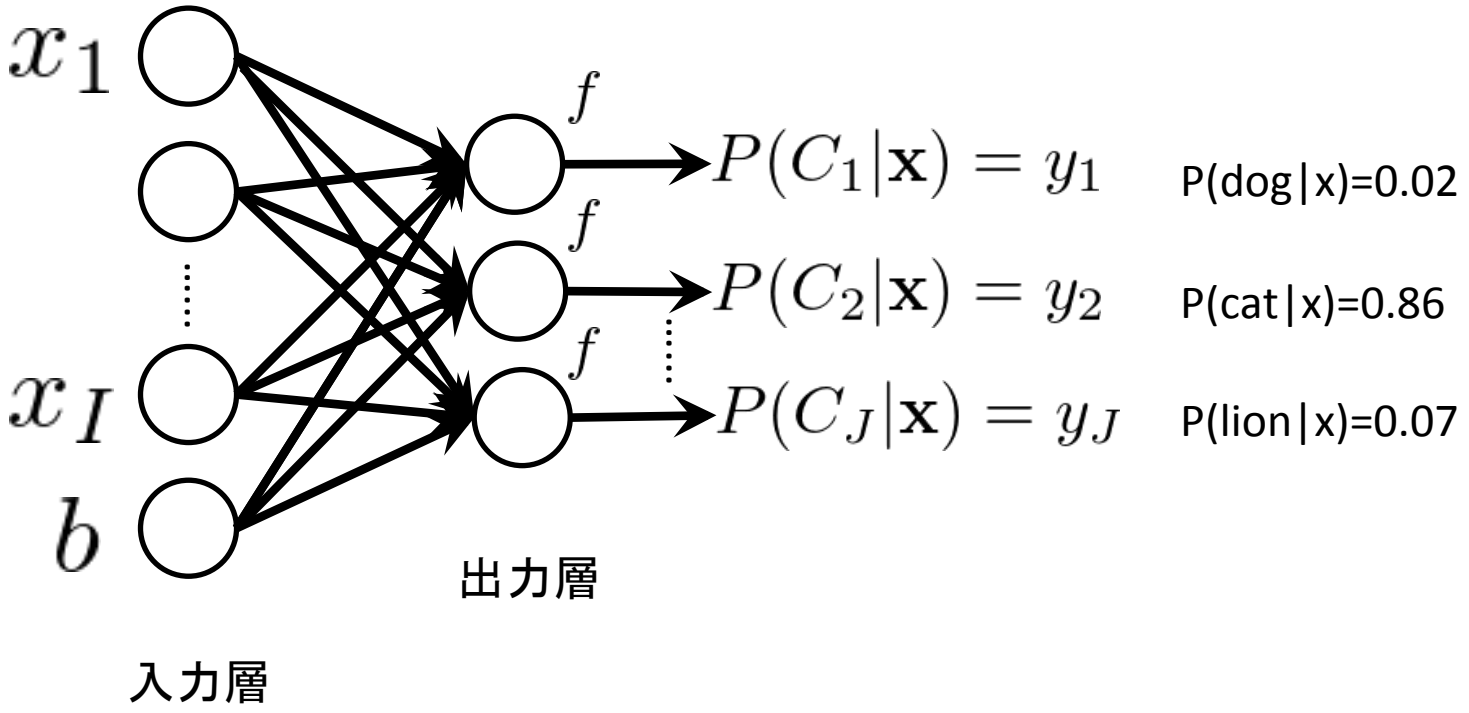
猫


きのこ

- そこに写っているものが何かを当てる
  1. 領域の切り分け
  2. 各領域の物体認識
  3. シーンの理解
- かなり複雑なタスク(水草だけみても何かわからない)
- ここでは2だけ考える→とてもクラス数の多い分類

# 多クラス分類(softmax回帰)

- Softmax関数 (cf sigmond関数)= $\frac{\exp(u_k)}{\sum_{j=1}^{K}\exp(u_j)}$



再掲

# softmax回帰

- 訓練データの目標値
  - E.g. 手書き数字 0,1,…,9 の10クラス
  - 1-ok-k表記 (1 hot vector) $\boldsymbol{t}_n = [0\,0\,1\,0\,0\,0\,0\,0\,0\,0]^T$
- 予測 $\hat{t}_{nk} = \Pr(k|\boldsymbol{x}_n) = \dfrac{\exp(\boldsymbol{w}_k^T \boldsymbol{x}_n)}{\sum_{k=1}^{K} \exp(\boldsymbol{w}_k^T \boldsymbol{x}_n)}$
- 訓練データ集合に対する尤度

$$L(\boldsymbol{w}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \Pr(k|\boldsymbol{x}_n)^{t_{n,k}} = \prod_{n=1}^{N} \prod_{k=1}^{K} (\hat{t}_{n,k})^{t_{n,k}}$$

- 誤差関数(負の対数尤度)は交差エントロピー

$$E(\mathbf{w}) = -\ln L(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln \hat{t}_{nk}$$

再掲

# Softmax回帰と交差エントロピー

- 2クラス分類の場合

ロジスティック回帰 $\hat{t}_n = \sigma(\boldsymbol{w}^T \boldsymbol{x}_n)$

$$E(\boldsymbol{w}) = -\ln L(\boldsymbol{w}) = -\sum_{n=1}^{N} \{t_n \ln \hat{t}_n + (1 - t_n) \ln(1 - \hat{t}_n)\}$$

- Kクラス分類の場合 同様にニュートン法などで最小化する

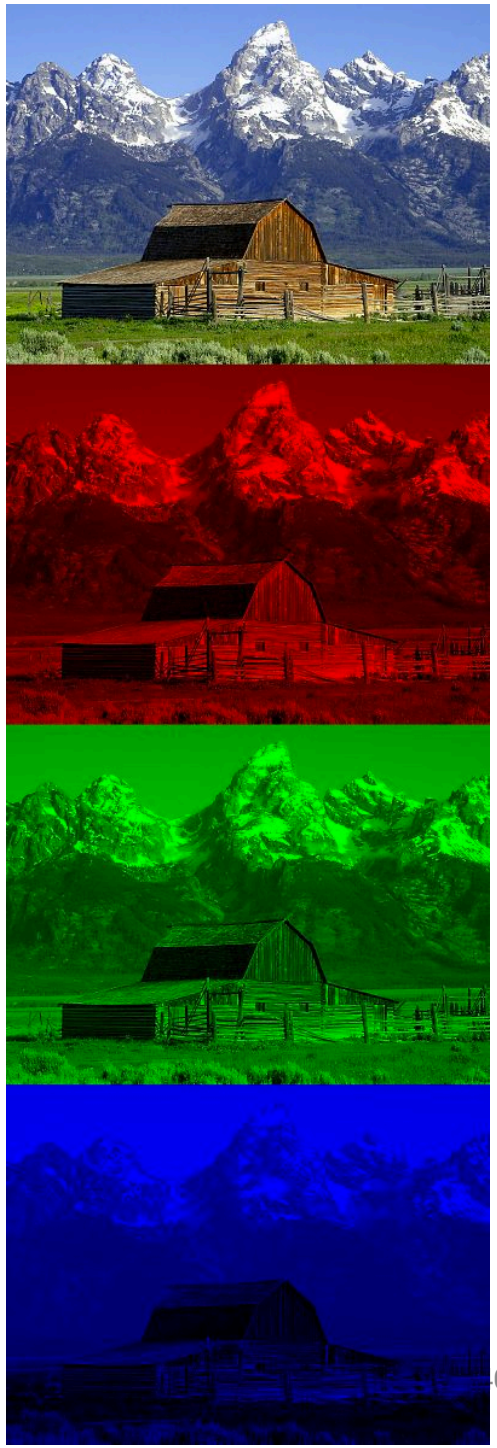
$$E(\mathbf{w}) = -\ln L(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln \hat{t}_{nk}$$

softmax回帰

$$\hat{t}_{nk} = \Pr(k|\boldsymbol{x}_n) = \frac{\exp(\boldsymbol{w}_k^T \boldsymbol{x}_n)}{\sum_{k=1}^{K} \exp(\boldsymbol{w}_k^T \boldsymbol{x}_n)}$$

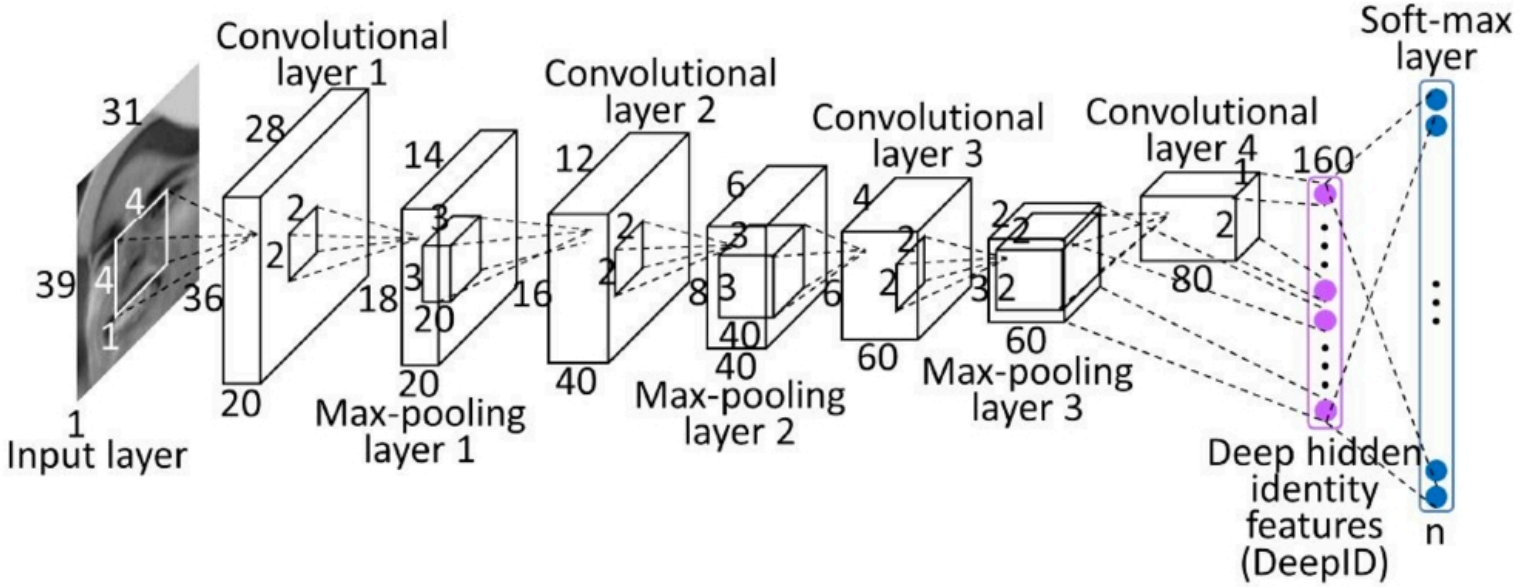
# 画像

- 画像サイズ　W*H
- チャンネル
  - モノクロ画像 1チャンネル
  - カラー画像 3チャンネル (RGB)



Wikipedia/public domain

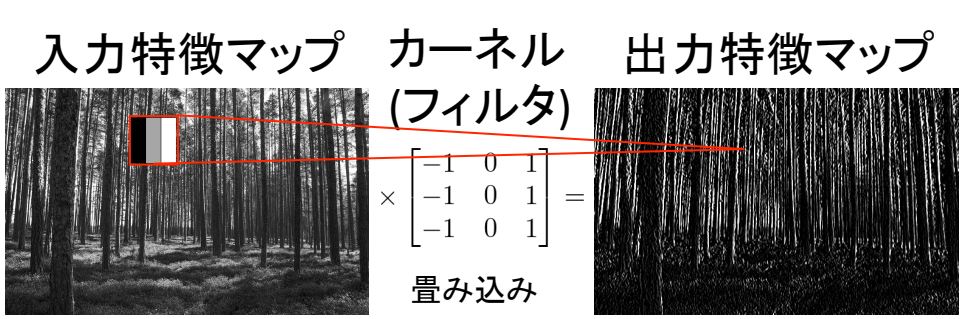
# Convolutional Neural Net

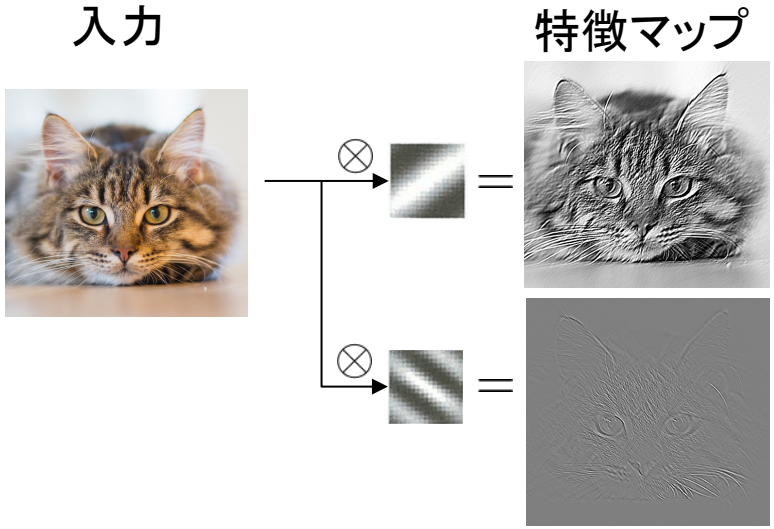



- 畳み込み層(conv)、プーリング層(pool)、全結合層(FC)からなる(後述)
- 用語
  - フィルタ or カーネル: 画像内で検出対象となる局所的なパターン
  - 特徴マップ:画像内の局所領域の、特定のフィルタに対する活性度

Sun, Yi, Xiaogang Wang, and Xiaoou Tang. "Deep learning face representation from predicting 10,000 classes." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014.

# 畳み込み層 (conv)



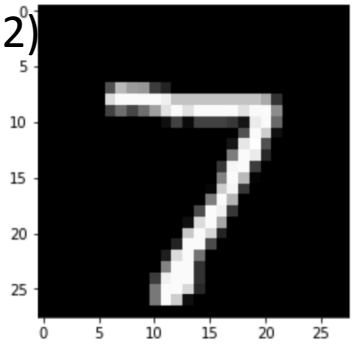

- フィルタが表すパターンが、入力特徴マップのどこにあるかを検出し出力特徴マップに表す
- 畳み込みの計算
  - H:フィルタサイズ
  - s: ストライド

$$u_{ij} = \sum_{p=0}^{H-1} \sum_{q=0}^{H-1} x_{si+p,sj+q} h_{pq}$$

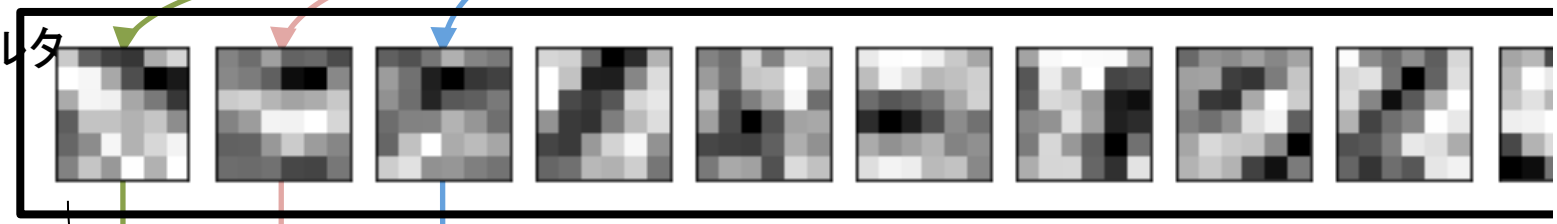
- 畳み込み後、特徴マップは縮小する
- フィルタ自身が学習対象

# 畳み込み層の処理(conv2d_1 (6, 6, 1, 32))

入力画像
28*28



畳み込みフィルタ
Conv filter
6*6

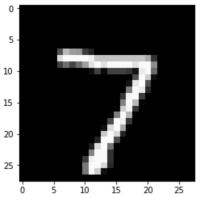


32 filters

特徴マップ
Feature map
23*23

32 feature maps

畳み込み層の処理(conv2d_2 (6, 6, 32, 64))



First layer

32 feature maps

特徴マップ
Feature map
23*23*32

64 filters

畳み込みフィル
Conv filter
6*6*32*64

64 feature maps

特徴マップ
Feature map
18*18*64

# プーリング層(pool)

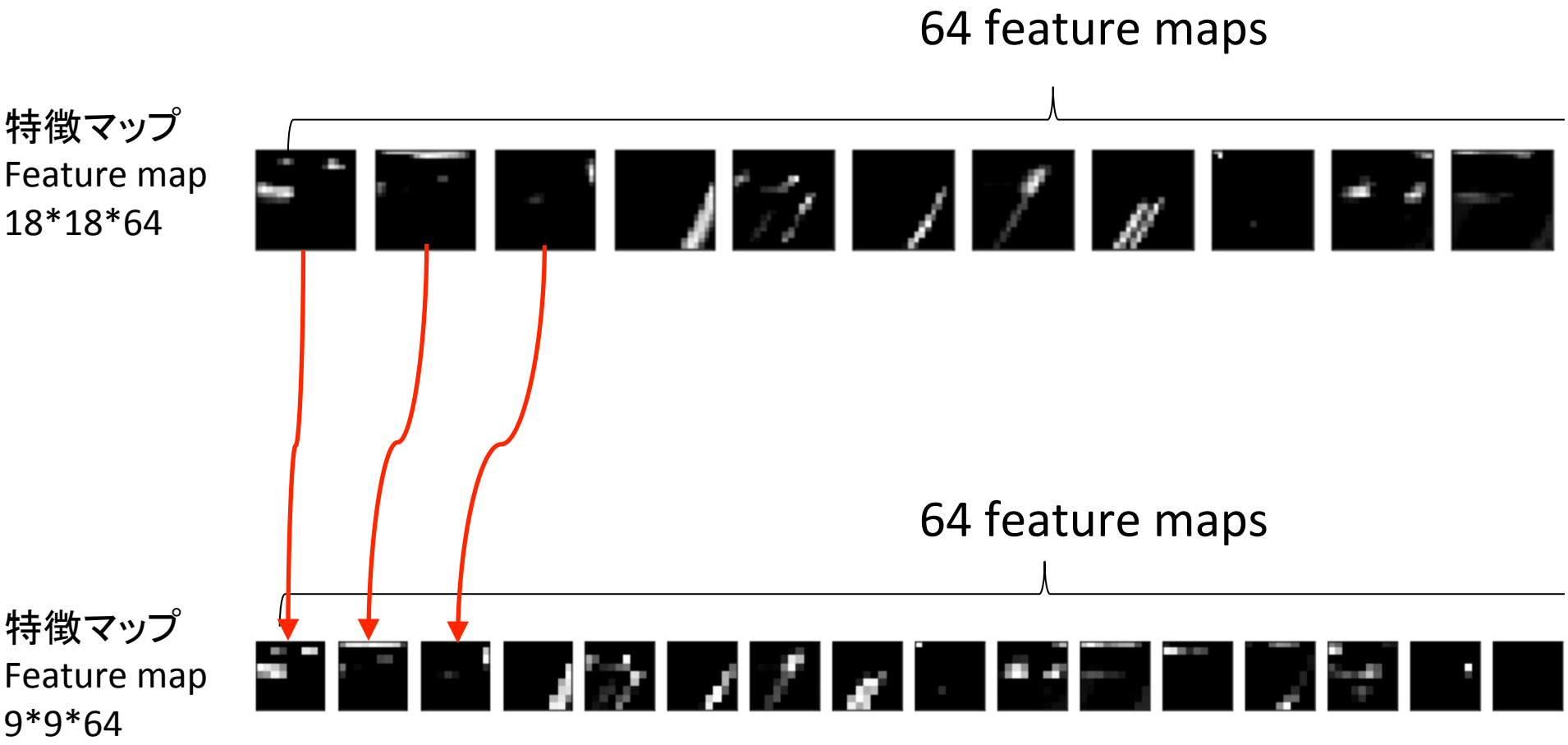
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 62 | 71 | 72 | 69 | 65 | 71 | 79 | 107 |
| | 73 | 79 | 80 | 81 | 79 | 79 | 98 | 128 |
| | 76 | 82 | 81 | 79 | 75 | 81 | 114 | 132 |
| | 77 | 85 | 83 | 77 | 72 | 99 | 144 | 145 |
| | 74 | 79 | 77 | 77 | 79 | 112 | 155 | 142 |
| | 74 | 73 | 71 | 73 | 89 | 142 | 162 | 137 |
| | 69 | 73 | 73 | 77 | 110 | 160 | 166 | 134 |
| | 60 | 67 | 68 | 78 | 124 | 154 | 148 | 116 |

Max pooling

| | | |
|---|---|---|
| 79 | 81 | 128 |
| 85 | 83 | 155 |
| 74 | 124 | 166 |

- conv層で検出した特徴の位置依存性を低下させ、検出をロバストにする
- 単に対象領域の最大値を取るだけ $u_{ijk} = \max_{(p,q)\in P_{ij}} z_{pqk}$
- バイアスだけが学習対象となる

# MaxPooling層の処理



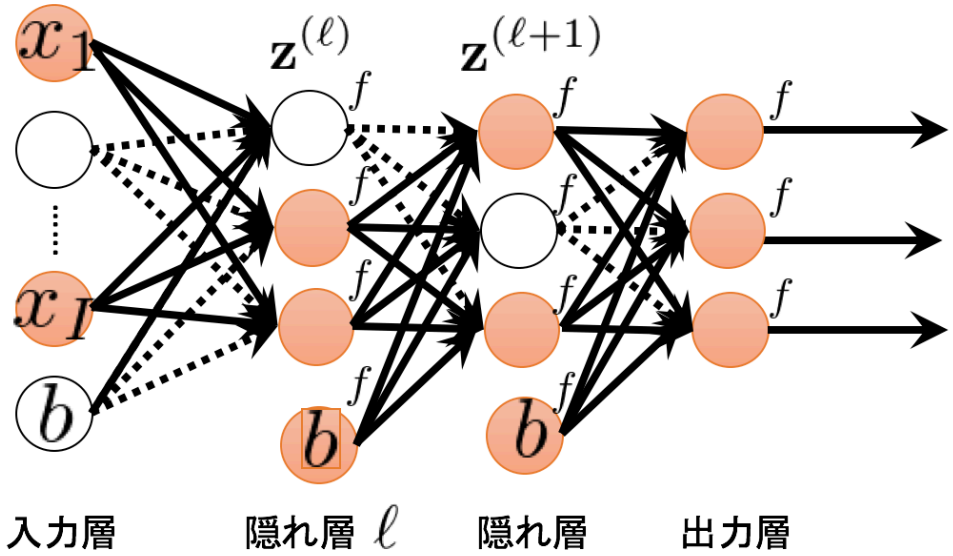
# conv+pool

- Conv
  - 各フィルタに特徴付けれらた特徴を検出
- Pool
  - 位置の微小変化に左右されない普遍性
- Conv+pool
  - 局所的な変異に依存しない安定した特徴抽出

…

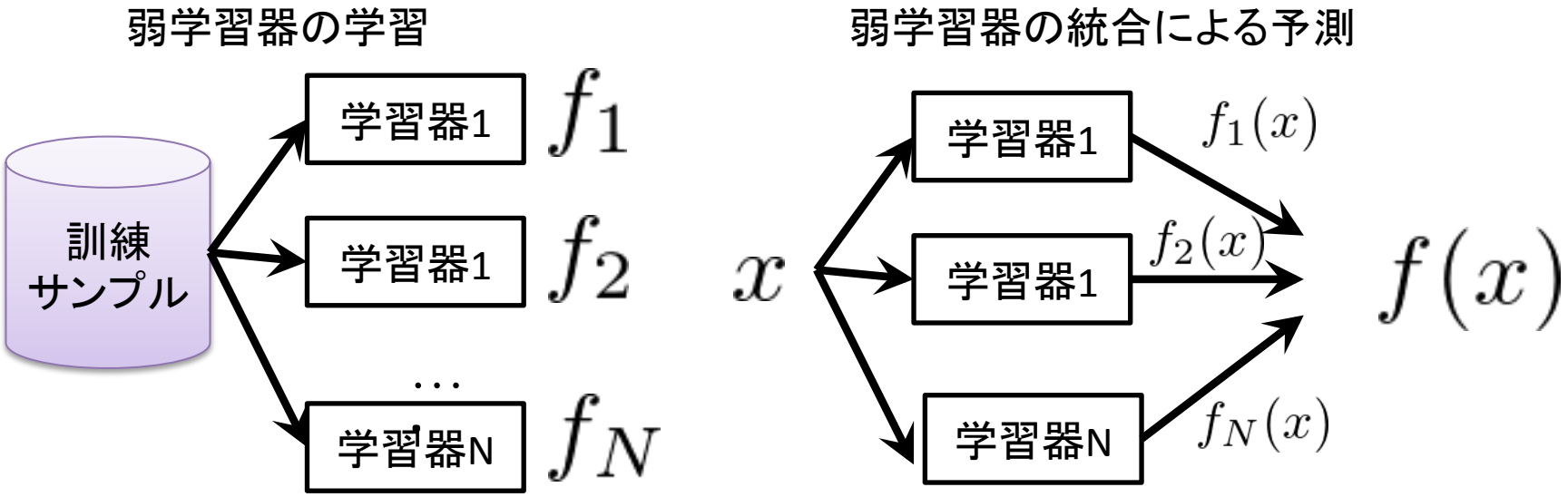
# 過学習の緩和 Dropout

- 学習時
  1. 入力、中間ノードを確率pでランダムに選択
  2. 選択されたノードだけで重みを更新




- 予測時
  - 各重みをp倍して順伝播し、dropoutで割り引かれた分を補償する

# アンサンブル学習とdropout




- アンサンブル学習(e.g., boosting, bagging)
  - 複数の異なる弱学習器を同一サンプルから学習
  - それぞれの予測結果を統合(強学習器)
  - 予測結果は単一の弱学習器よりも良いことが多い
- Dropout
  - 複数の異なる構造を持つNNを学習
  - 予測値はそれぞれの構造での確率的平均
  - Dropoutでの学習はアンサンブル学習に類似し、過学習を防ぎ、汎化誤差が改善すると言われる

# CNNモデルSetting例 (Keras)

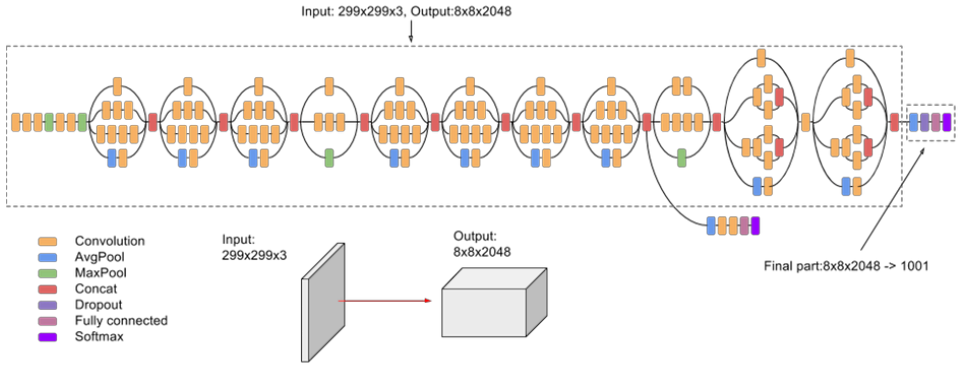
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(6, 6),    //6*6フィルタ, 32チャンネル
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))   //3*3フィルタ, 64チャンネル
model.add(MaxPooling2D(pool_size=(2, 2)))          //2*2 max pooling
model.add(Dropout(0.25))  //dropout, λ=0.25
model.add(Flatten())  //テンソルからベクトルへ
model.add(Dense(128, activation='relu')) //128次元密結合
model.add(Dropout(0.5))   //dropout, λ=0.5
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
model.save_weights(WEIGHTS_FNAME)
model.summary()
```

# 様々なCNN

VGG16 VGG19

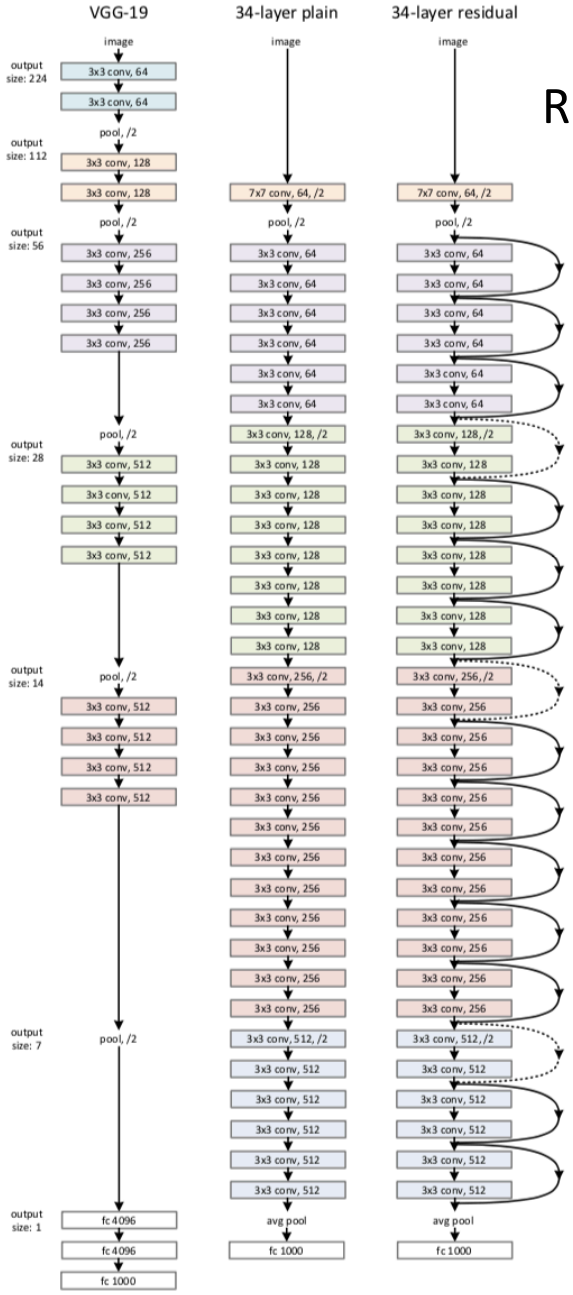
| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64 **LRN** | conv3-64 **conv3-64** | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 **conv3-128** | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 |
| maxpool | | | | | |
| conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 **conv1-256** | conv3-256 conv3-256 **conv3-256** | conv3-256 conv3-256 **conv3-256** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 **conv1-512** | conv3-512 conv3-512 **conv3-512** | conv3-512 conv3-512 **conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Inception V3




https://cloud.google.com/tpu/docs/inception-v3-advanced

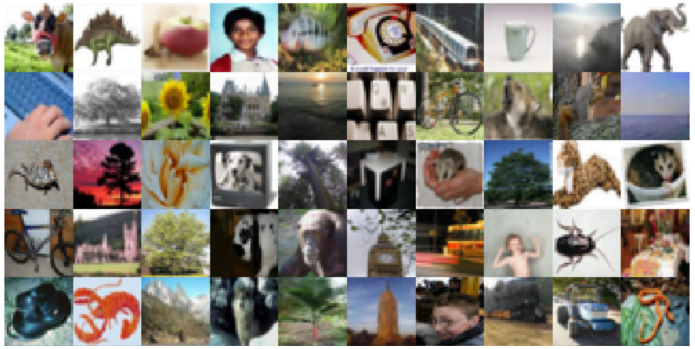
ResNet




Deep Residual Learning for Image Recognition, He+'15

演習9.4

# CIFAR100データセット



- 100クラス画像分類用データセット
  - サイズ32x32, 3チャンネル(RGB)
  - 100クラス、各クラス600画像
  - 20の上位クラス、各上位クラスに5の下位クラス

| Superclass | Classes |
|---|---|
| aquatic mammals | beaver, dolphin, otter, seal, whale |
| fish | aquarium fish, flatfish, ray, shark, trout |
| flowers | orchids, poppies, roses, sunflowers, tulips |
| food containers | bottles, bowls, cans, cups, plates |
| fruit and vegetables | apples, mushrooms, oranges, pears, sweet peppers |
| household electrical devices | clock, computer keyboard, lamp, telephone, television |
| household furniture | bed, chair, couch, table, wardrobe |
| insects | bee, beetle, butterfly, caterpillar, cockroach |
| ..... | |

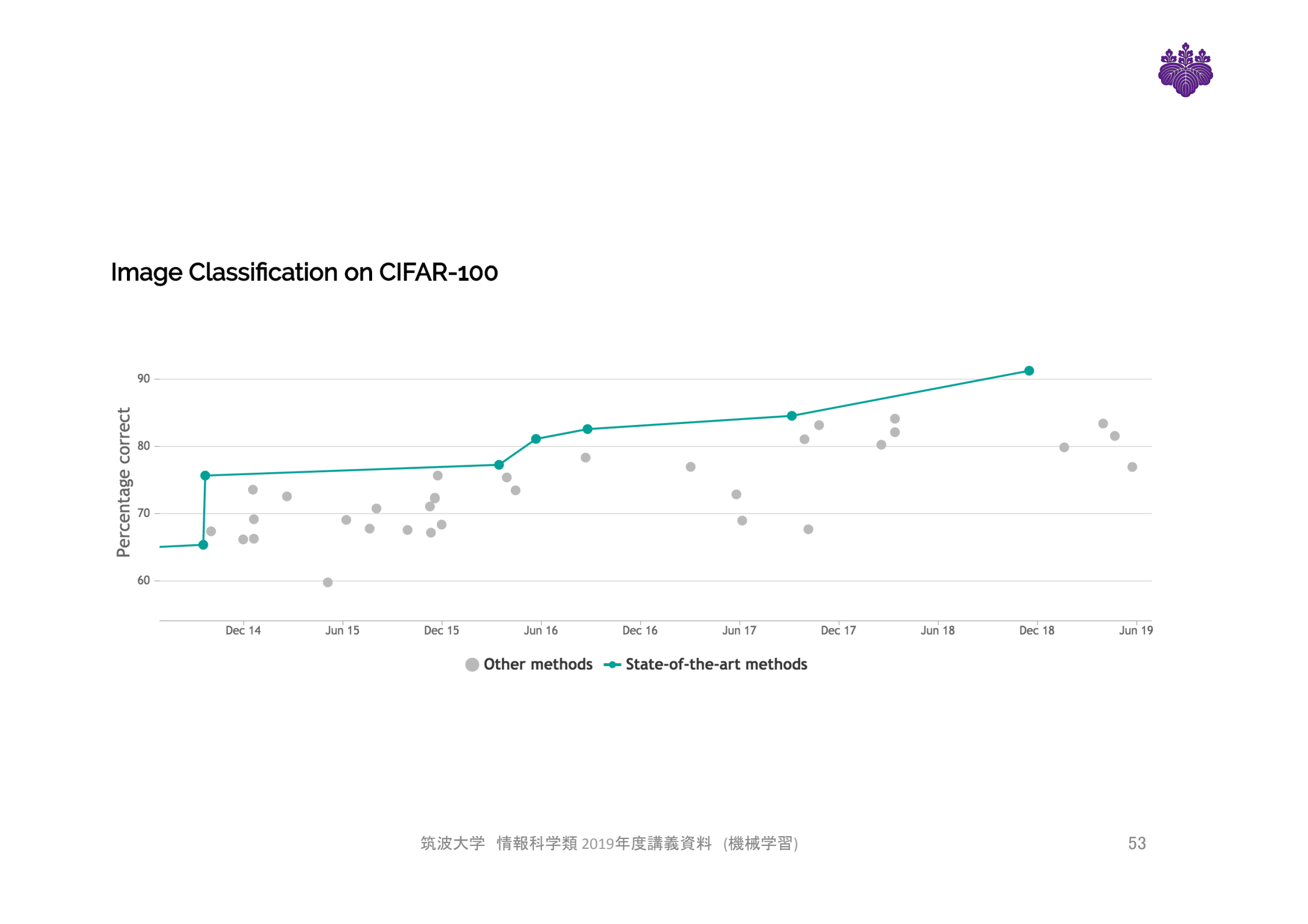
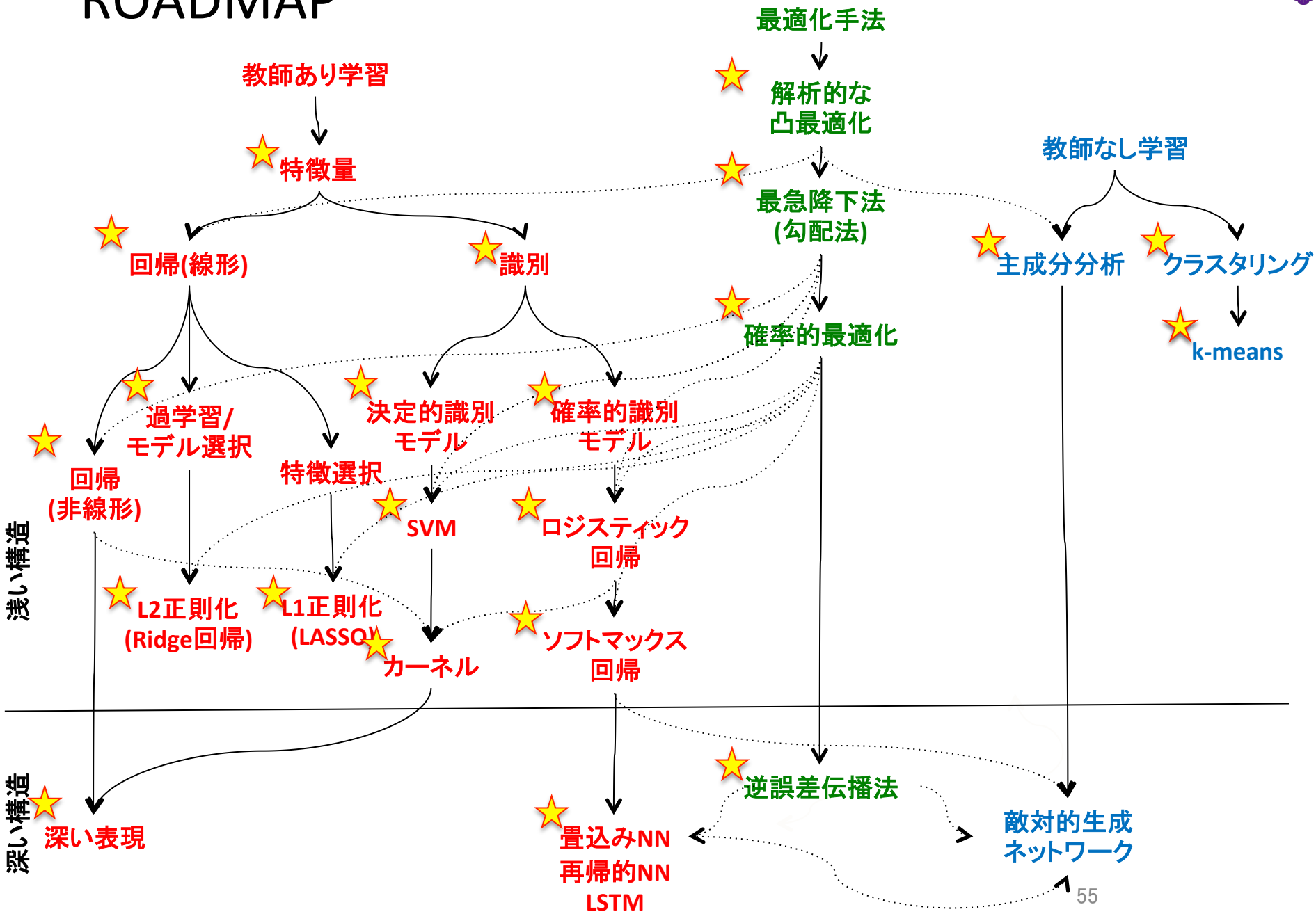


Image Classification on CIFAR-100
Percentage correct
90
80
70
60
Dec 14
Jun 15
Dec 15
Jun 16
Dec 16
Jun 17
Dec 17
Jun 18
Dec 18
Jun 19
Other methods
State-of-the-art methods

# まとめ

- 多層NN
  - 情報の線型結合と非線形活性化関数の繰り返しによる特徴生成
  - 生成された特徴量を回帰・分類
- 多層NNの学習
  - 確率的勾配降下法(ミニバッチ)
  - 勾配は逆誤差伝播法で求める
- CNN=画像分類のための多層NN
  - 畳み込み層、max-poolingの繰り返し(特徴生成)
  - Softmax回帰(分類)

# ROADMAP


最適化手法
教師あり学習
解析的な
凸最適化
特徴量
最急降下法
(勾配法)
教師なし学習
回帰(線形)
識別
主成分分析
クラスタリング
確率的最適化
k-means
過学習/
モデル選択
決定的識別
モデル
確率的識別
モデル
回帰
(非線形)
特徴選択
SVM
ロジスティック
回帰
浅い構造
L2正則化
(Ridge回帰)
L1正則化
(LASSO)
カーネル
ソフトマックス
回帰
逆誤差伝播法
深い構造
深い表現
畳込みNN
再帰的NN
LSTM
敵対的生成
ネットワーク