データサイエンス
第10回

~ニューラルネットワーク・深層学習入門~

情報理工学系研究科
創造情報学専攻
中山 英樹

本日の内容

- 前回のやり残し
  - カーネル法、Explicit feature maps
- ニューラルネットワーク、深層学習概要
  - 歴史
  - 誤差逆伝播法の実装

レポート課題 1

- お疲れさまでした（提出132名）
  - 提出者一覧を確認してください
- ネタばらし
  - Facebook comment volume prediction
    https://archive.ics.uci.edu/ml/datasets/Facebook+Comment+Volume+Dataset

    K.Singh, R. Kaur, and D. Kumar, " Comment Volume Prediction Using Neural Networks and Decision Trees", In Proc. 17th UKSIM-AMSS International Conference on Modelling and Simulation, 2015.
- 最終結果

制限あり

Rank	Name	Mean ABS Error
1	z	3.7507134880708
2	zukkyun	3.7785930350644
3	amaotone	3.8785407265772
4	scarlet	3.8968
5	Hugh	3.921

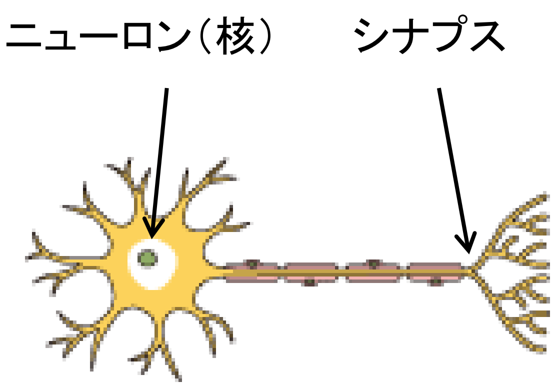
制限なし

Rank	Name	Mean ABS Error
1	zukkyun	3.5861741273348
2	amaotone	3.6572861451587
3	YI	3.7196
4	shinkyo	3.7471082293018
5	nagano	3.7521

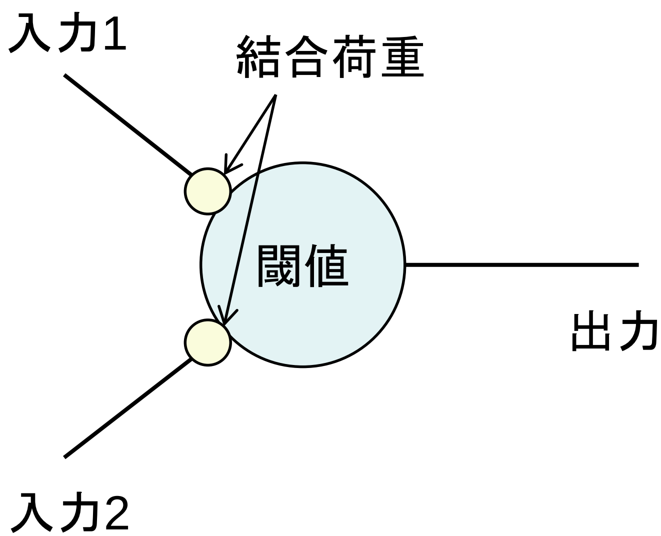
（人工）ニューラルネットワーク

- 脳神経系を模した数学モデル
- ネットワークを形成する多数の人工ニューロンのシナプス結合強度を変化させて問題解決能力を獲得する

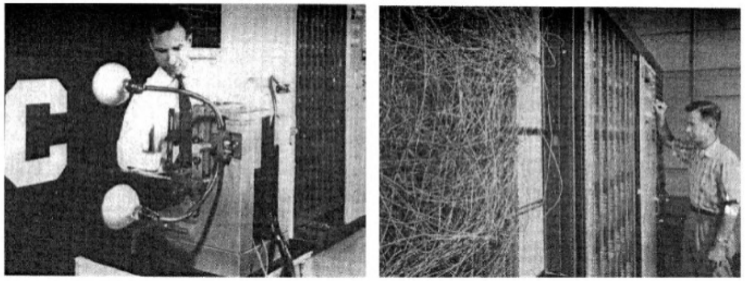
神経細胞（ニューロン）



ニューロンモデル



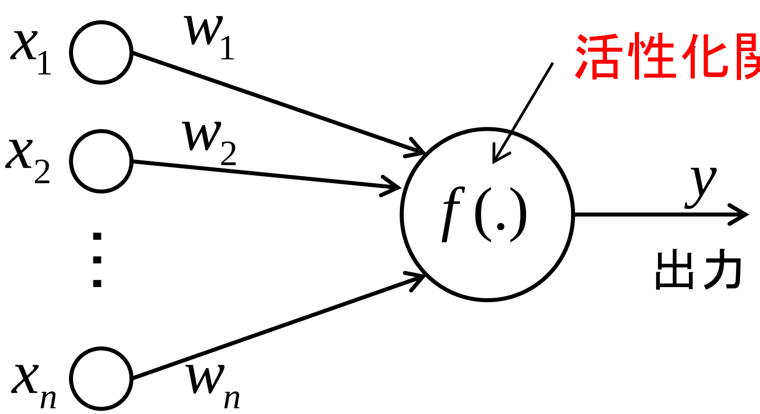
第一世代



Rosenblatt のMark 1 perceptron hardware (1960)

- 単純パーセプトロン (1960s)
  - 入力値の線形結合 + 活性化関数による非線形変換

入力



活性化関数

$$y = f(v)$$

$$v = \underline{\mathbf{w}}^T \mathbf{x} - \underline{\theta}$$

結合重み　バイアス

- 例)

$$f(\eta) = \begin{cases} 1 & \eta > 0 \\ 0 & \text{otherwise} \end{cases}$$

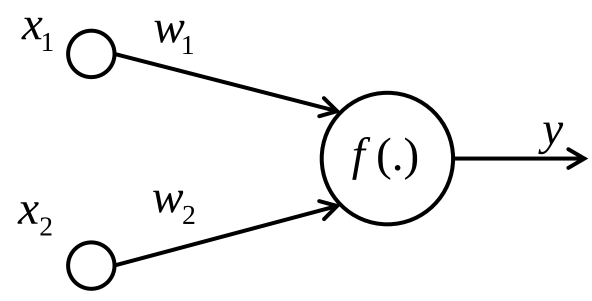
ステップ関数
→ **McCulloch & Pitts モデル (1943)**

$$f(\eta) = \frac{1}{1 + \exp(-\eta)}$$

シグモイド関数
実用上便利で最もよく用いられてきた
（ロジスティック回帰と等価）

単純パーセプトロンの学習

- 線形識別平面を作ることに対応
- 訓練サンプルが正しく識別されるように少しずつパラメータを更新



i番目の訓練サンプル　$\{\mathbf{x}_i, y_i\},\ y_i \in \{0, 1\}$

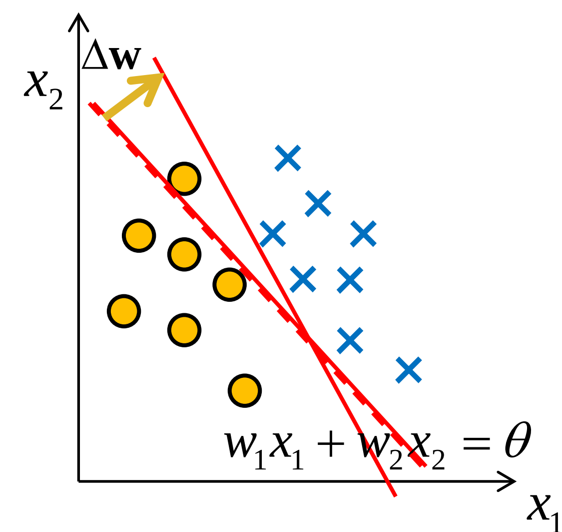
$$\frac{\partial L}{\partial \mathbf{w}} = \frac{1}{2} \cdot \frac{\partial (y_i - \hat{y}_i)^2}{\partial \mathbf{w}}$$

（二乗誤差基準の場合）

$$= -(y_i - \hat{y}_i) f'(\mathbf{w}^T \mathbf{x}_i - \theta) \mathbf{x}_i$$

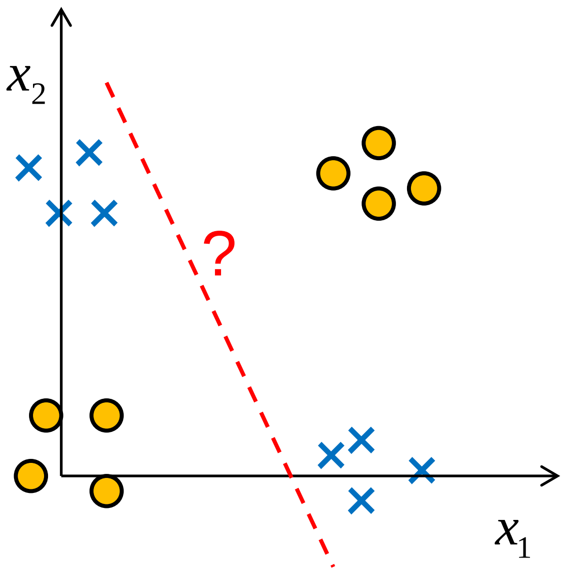
$$\mathbf{w}_{new} = \mathbf{w}_{old} + \underline{\eta} \underline{(y_i - \hat{y}_i)} \underline{f'(\mathbf{w}_{old}^T \mathbf{x}_i - \theta_{old})} \mathbf{x}_i$$

エラーに説明変数をかけたもの



単純パーセプトロンの限界

- 非線形な識別は原理的に不可能
  - 例： XOR

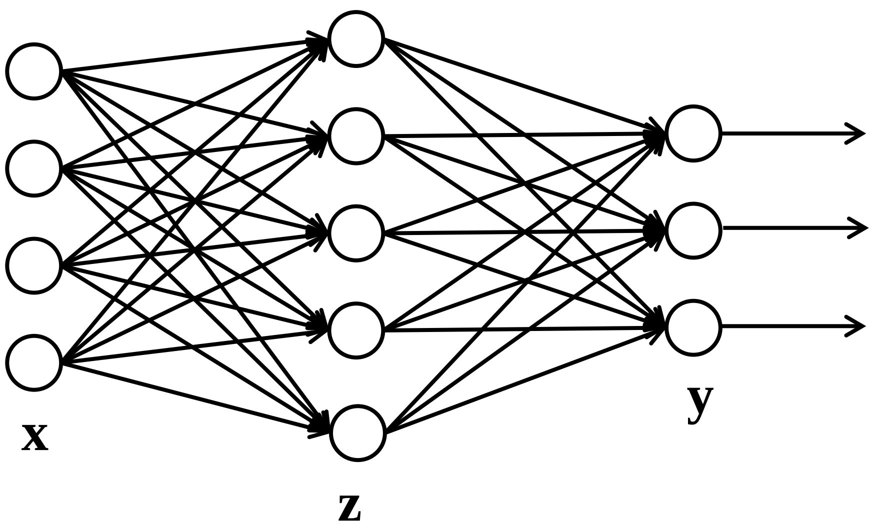


以降、第一次ニューラルネットブームは急速に下火に…

第二世代

- 多層パーセプトロン (1980s～1990s)
- 多数の単純パーセプトロンを階層的に組み合わせる
  - パラメータ最適化はNP困難だが、誤差逆伝播法で局所解へ収束

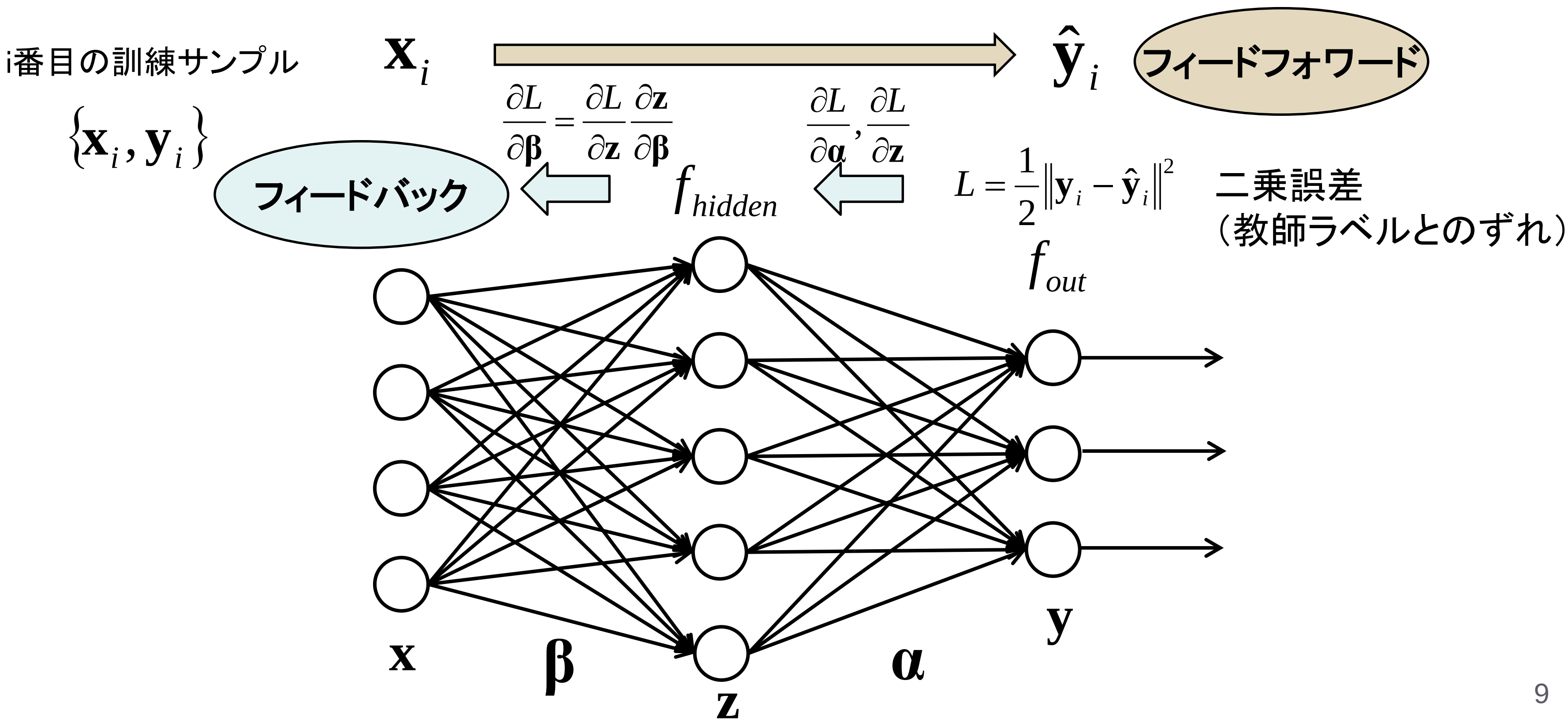
- 十分な数のニューロンが隠れ層にあれば、任意の非線形連続関数は３層のネットワークで近似できる



隠れ層
（陽に観測されない）

誤差逆伝播法

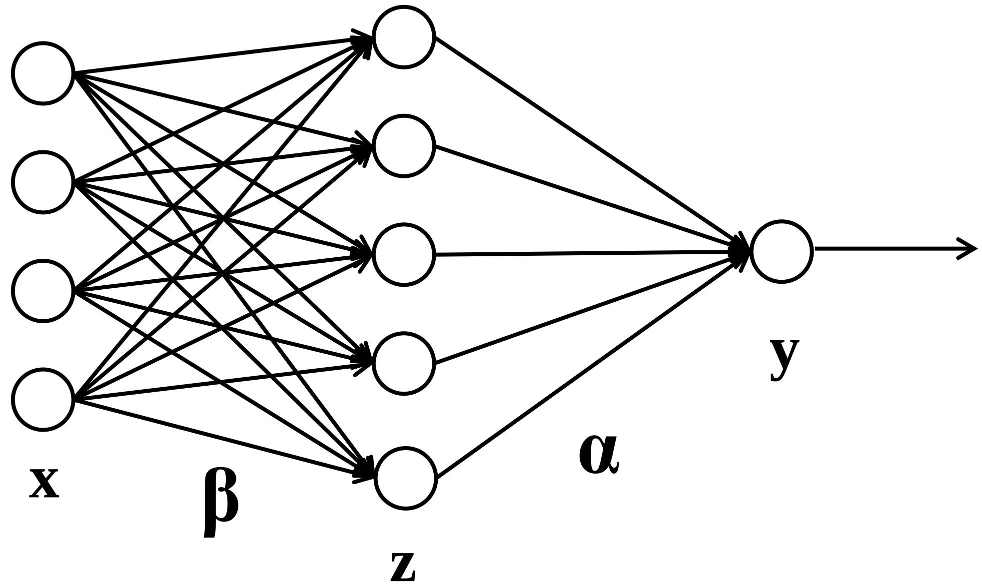
- やること自体は単純パーセプトロンと同じ
- 訓練サンプルをフィードフォワードし、得られた出力誤差を小さくする方向へパラメータを更新
- 上層からのchain ruleで誤差を順に低層へフィードバック



誤差逆伝播法

$\mathbf{x}_i$

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \frac{\partial L}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \boldsymbol{\beta}} \qquad \frac{\partial L}{\partial \boldsymbol{\alpha}}, \frac{\partial L}{\partial \mathbf{z}} \qquad L = \left(f(\mathbf{x}_i) - y_i\right)^2$$



評価関数（二乗誤差）：

$$L = \left(f(\mathbf{x}_i) - y_i\right)^2$$

隠れ層の活性化関数を
シグモイド関数とすると

$$\frac{\partial L}{\partial \alpha_j} = 2\sigma_{i,j}\delta_i$$

$$\delta_i = f(\mathbf{x}_i) - y_i$$

$$\sigma_{i,j} = \frac{1}{1 + \exp\left(-\boldsymbol{\beta}_j^T \mathbf{x}_i\right)}$$

Chain rule

$$\frac{\partial L}{\partial \beta_j^{(k)}} = \boxed{\left(\frac{\partial L}{\partial \mathbf{z}}\right)^T \frac{\partial \mathbf{z}}{\partial \beta_j^{(k)}}} = 2\alpha_j \delta_i \sigma'_{i,j} x_i^{(k)}$$

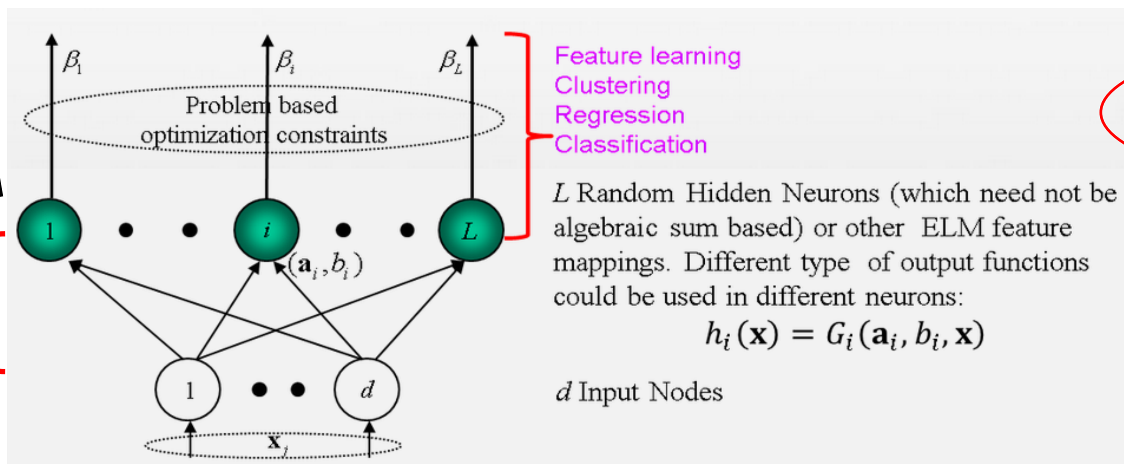
発生した誤差に対し、前層での出力が強いニューロンが大きく更新される

ちなみに…

- かつては入力層から中間層への結合はランダム
  - それでも理論上はOK。（ただし中間層のノードが指数的に必要になるので非現実的 by Minsky）
- Extreme learning machine [Huang et al., 2011]

活性化関数

Random projection



in MNIST OCR dataset

Learning Methods	Testing Accuracy	Training Time
ELM Auto Encoders	99.03%	<7.5 mins
Deep Belief Networks (DBN)	98.87%	5.7 hours
Deep Boltzmann Machines (DBM)	99.05%	19 hours
SAE	98.6%	> 17 hours
SDAE	98.72%	> 17 hours

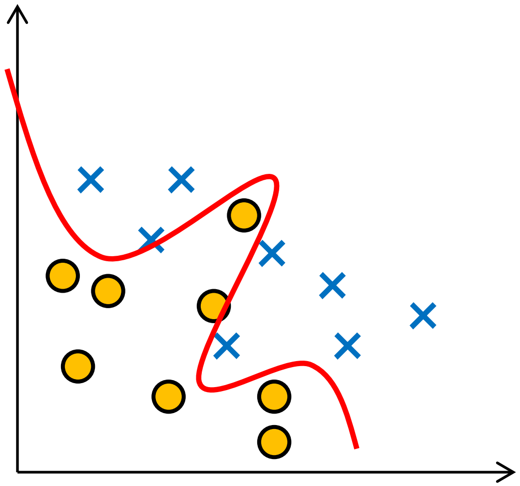
  - Random Fourier features と等価
    - 活性化関数がcos、sinの場合

多層パーセプトロンの限界（当時）

- 誤差逆伝播学習は実際にはあまりうまく働かず…

- 問題点
  - 入力に近い層の学習が遅い（層を遡る過程で誤差が多数のニューロンへ拡散されてしまい、パラメータがほとんど更新されない）

  - 過学習しやすい（訓練データのみに過度に適応する現象）



ニューラルネット冬の時代

- ある論文の冒頭 [Simard et al., ICDAR 2003]

  After being extremely popular in the early 1990s, neural networks have fallen out of favor in research in the last 5 years. In 2000, it was even pointed out by the organizers of the Neural Information Processing System (NIPS) conference that the term “neural networks” in the submission title was negatively correlated with acceptance. In contrast, positive correlations were made with support vector machines (SVMs), Bayesian networks, and variational methods.

第三世代：Deep learning (深層学習)

- Deep learning (深層学習) (2006～)
  [Hinton and Salakhutdinov, Science, 2006]

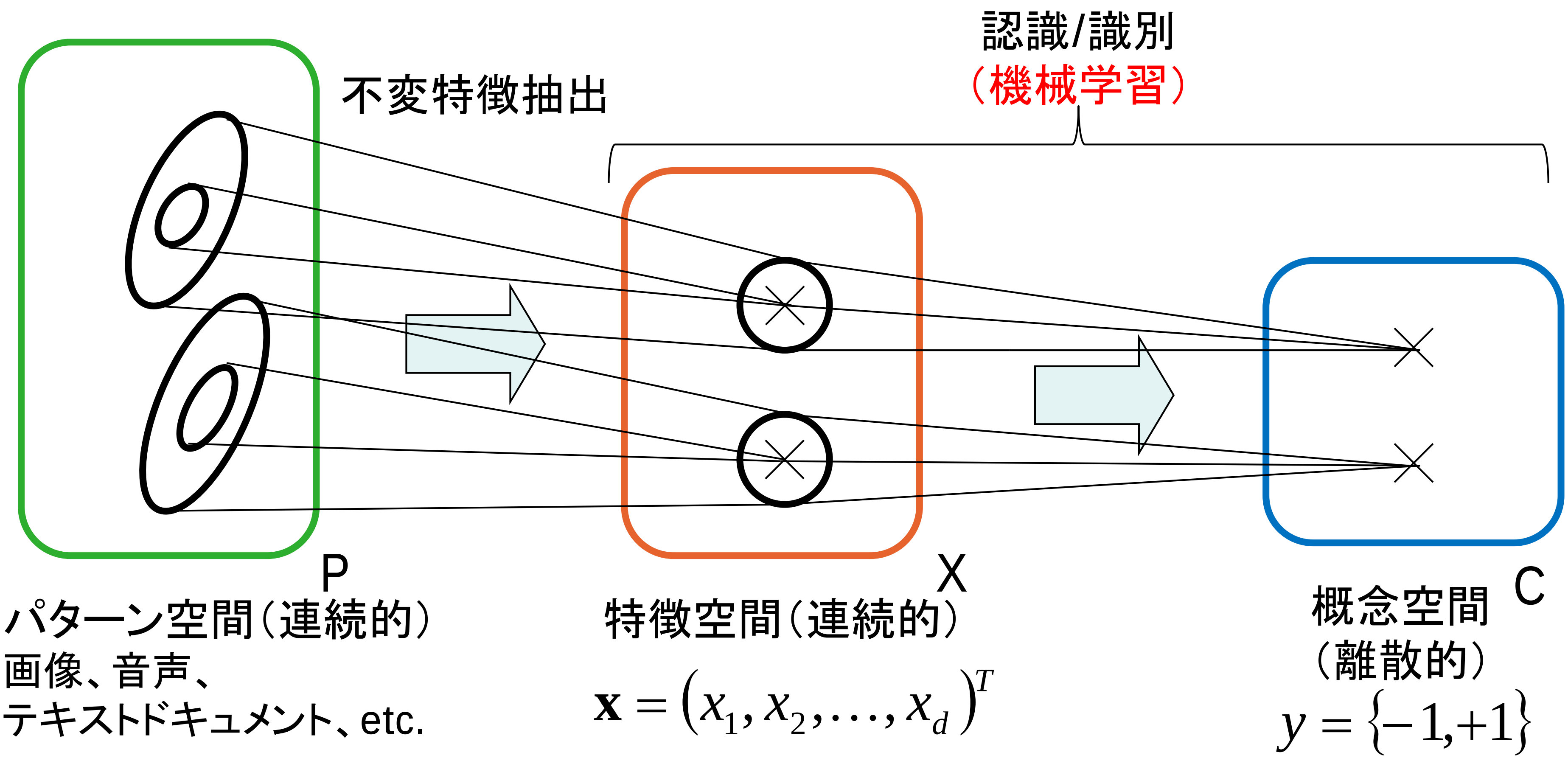
- 従来扱われてきたよりもさらに多層のニューラルネット
  - 2010年頃で7～8層程度。現在は150層以上のものも！

- 音声認識・画像認識・自然言語処理などさまざまな分野で圧倒的な性能を達成

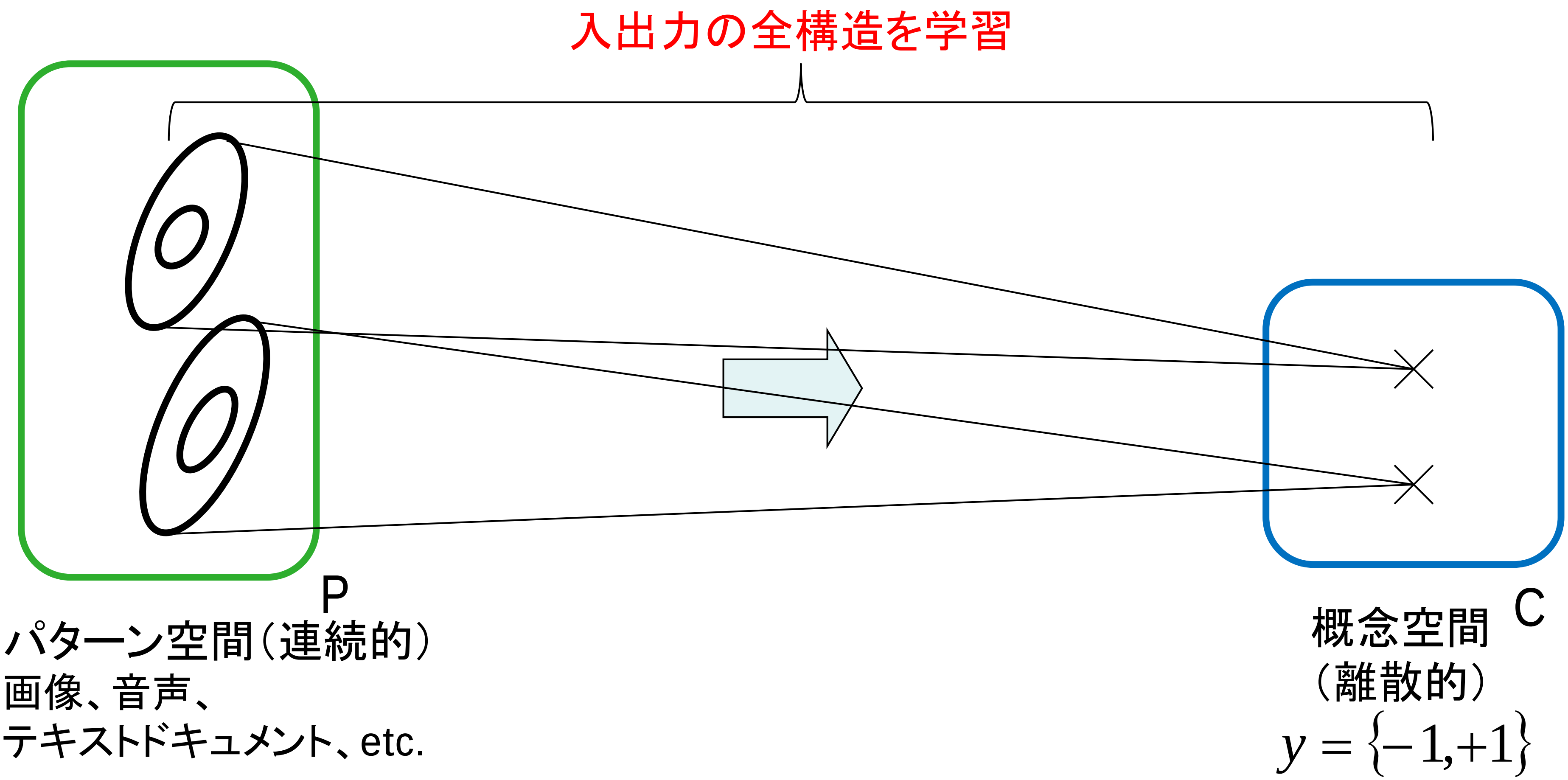
- 生データから目的変数に至るend-to-endの構造を学習
  - 従来の特徴量（に相当する構造）も自動的に獲得
  - パターン認識の文脈では表現学習（representation learning)とほぼ同義で扱われることも

パターン認識のパイプライン

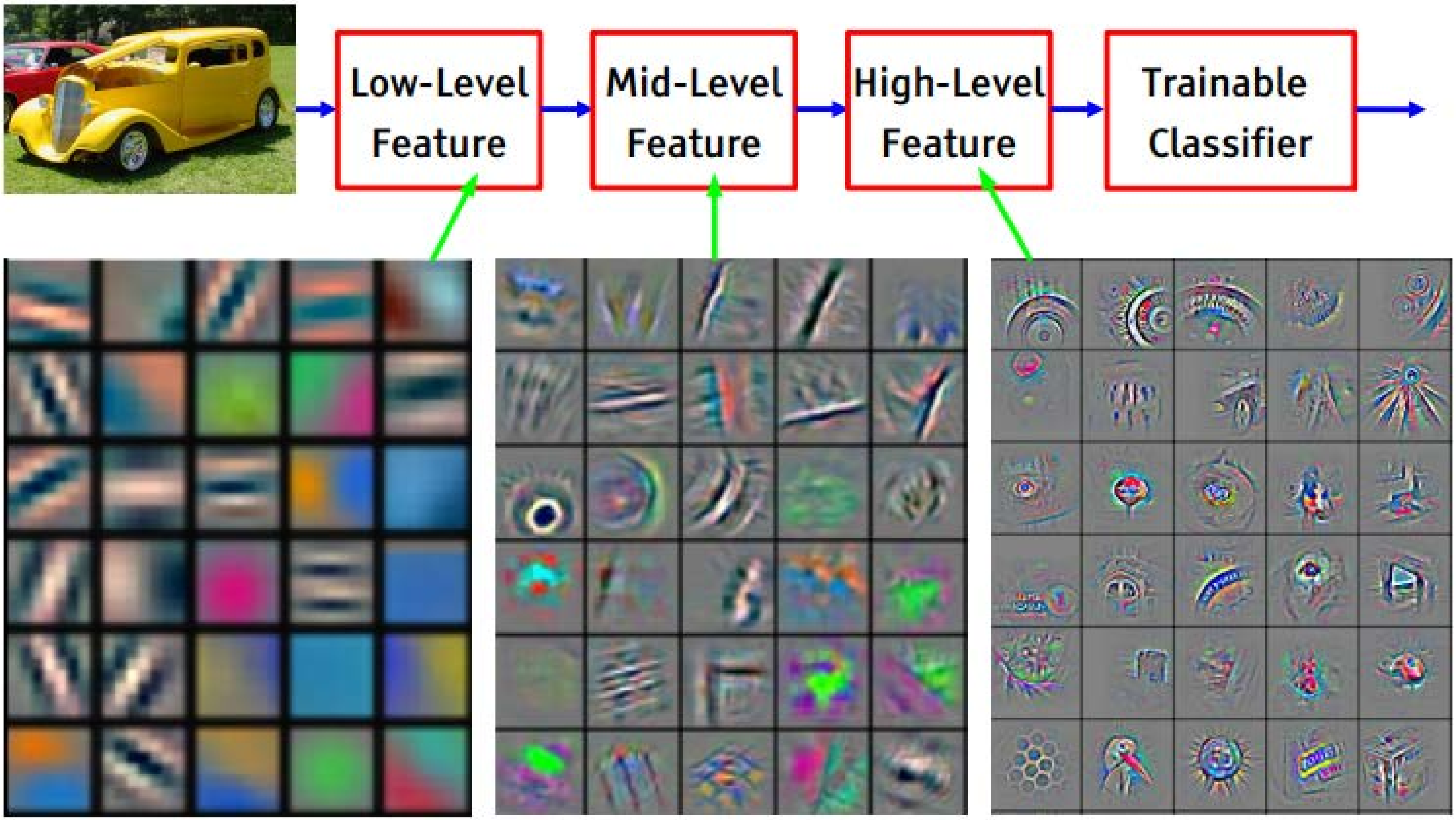
- よい特徴量（説明変数）を抽出・選択することは極めて重要
  - Garbage in garbage out



こうなりつつある



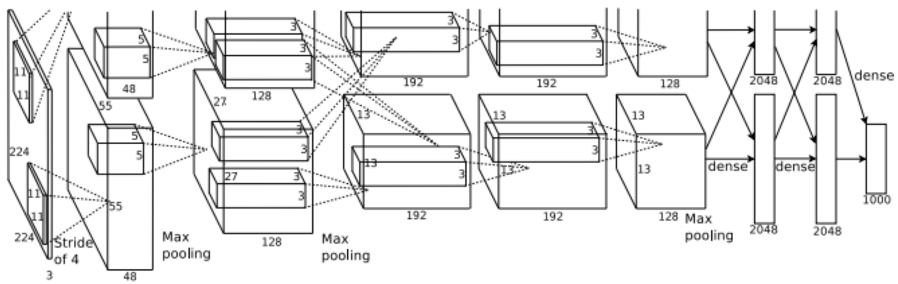
画像認識の例



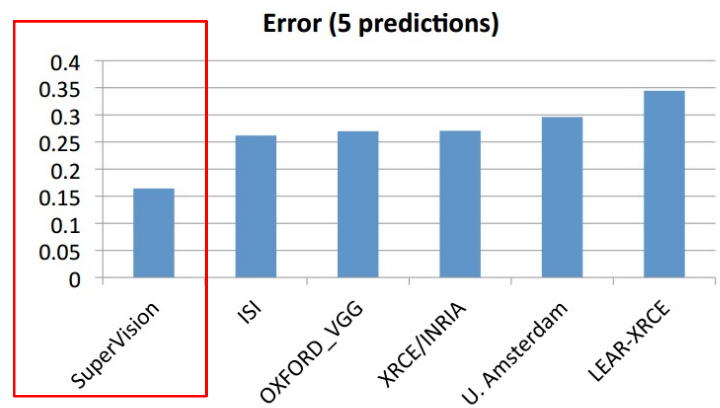
http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

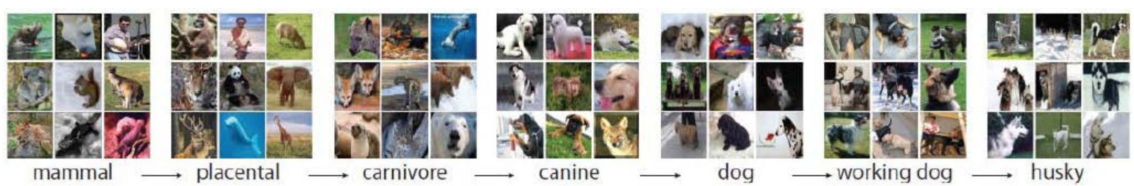
画像認識の例

- ImageNet large-scale visual recognition challenge 2012
  - 1000カテゴリの画像認識コンペティション



[Krizhevsky et al., 2012]

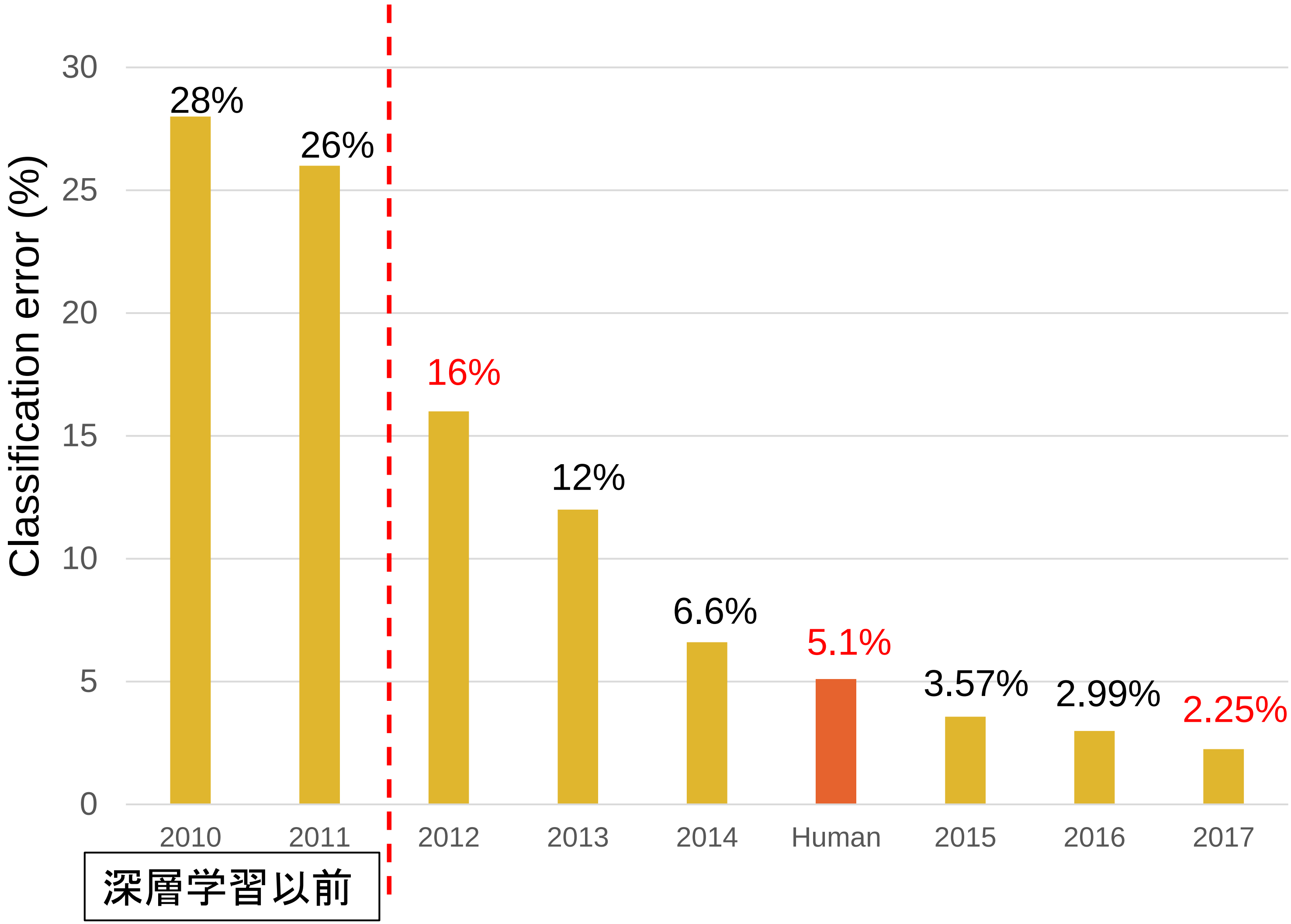




- 他、文字認識、道路標識認識などの主要なデータセットで
  人間に以上の識別精度を達成

圧倒的な性能向上

- エラー率が 16% (2012) → 2.3% (2017)



ニューラルネットワークの学習

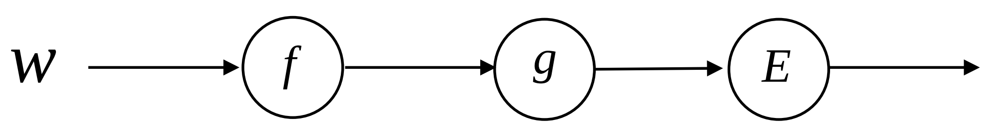
- 基本は勾配降下法

$$\mathbf{w} \Leftarrow \mathbf{w} - \alpha \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

- 連鎖率(chain rule)を用いて微分を計算
  - 合成関数の微分

$$E'(w) = E'(g(f(w)))g'(f(w))f'(w)$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial g}\frac{\partial g}{\partial f}\frac{\partial f}{\partial w}$$



- NNとは、要するに一次微分可能な関数のつながり

最適化手法の発達（?）

- 確率的勾配降下法 (stochastic gradient descent)

  ※深層学習のために出てきたものではない

  - 1データごとに目的関数の勾配を出し、重みを更新
  - 学習が圧倒的に高速化

  （注意）学習サンプルはシャッフルしておくこと

- 更新式
  - 最急降下法

$$\mathbf{w} \Leftarrow \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} \left\{ \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}_i, y_i) \right\}$$

  - 確率的勾配降下法

$$\mathbf{w} \Leftarrow \mathbf{w} - \alpha \frac{\partial L(\mathbf{x}_i, y_i)}{\partial \mathbf{w}}$$

ミニバッチによるSGD

- ある程度サンプルを束ねて更新

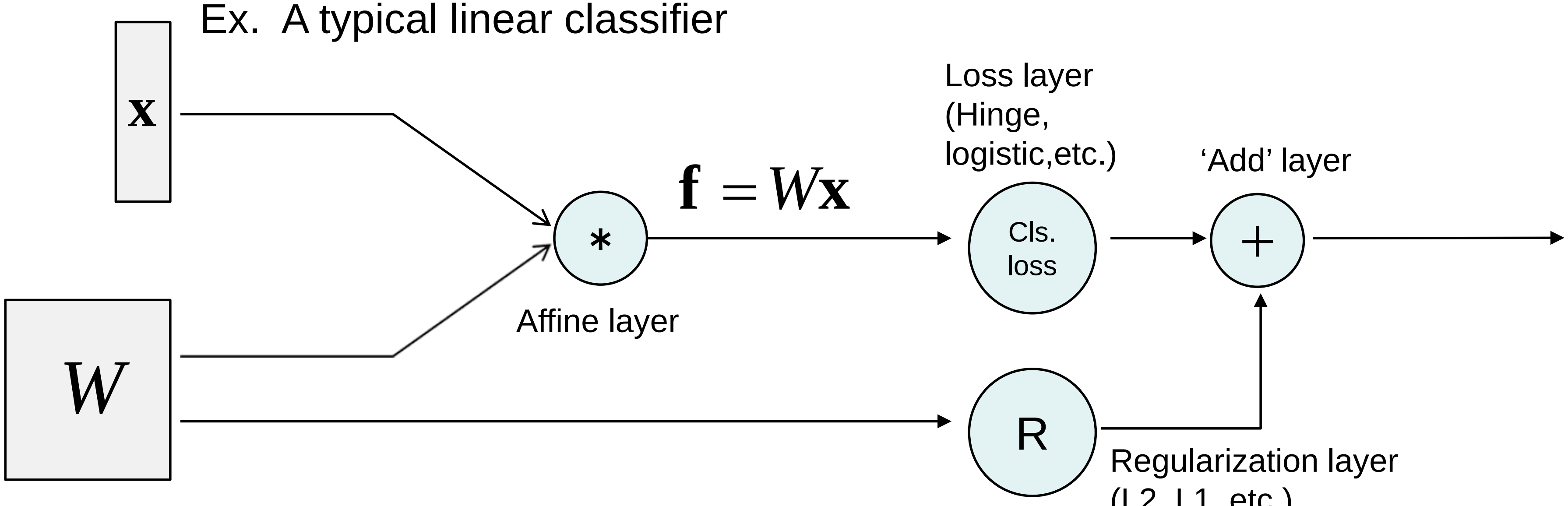
$$\mathbf{w} \Leftarrow \mathbf{w} - \gamma \frac{\partial}{\partial \mathbf{w}} \left\{ \frac{1}{B} \sum_{i=1}^{B} L(\mathbf{x}_i, y_i) \right\}$$

- バッチ内のデータの評価は並列化可能
  - 一般にSGDの並列化は難しいが、GPUの実装法まで含めて研究が進められている
    Coates et al., “Deep learning with COTS HPC systems”, ICML’13

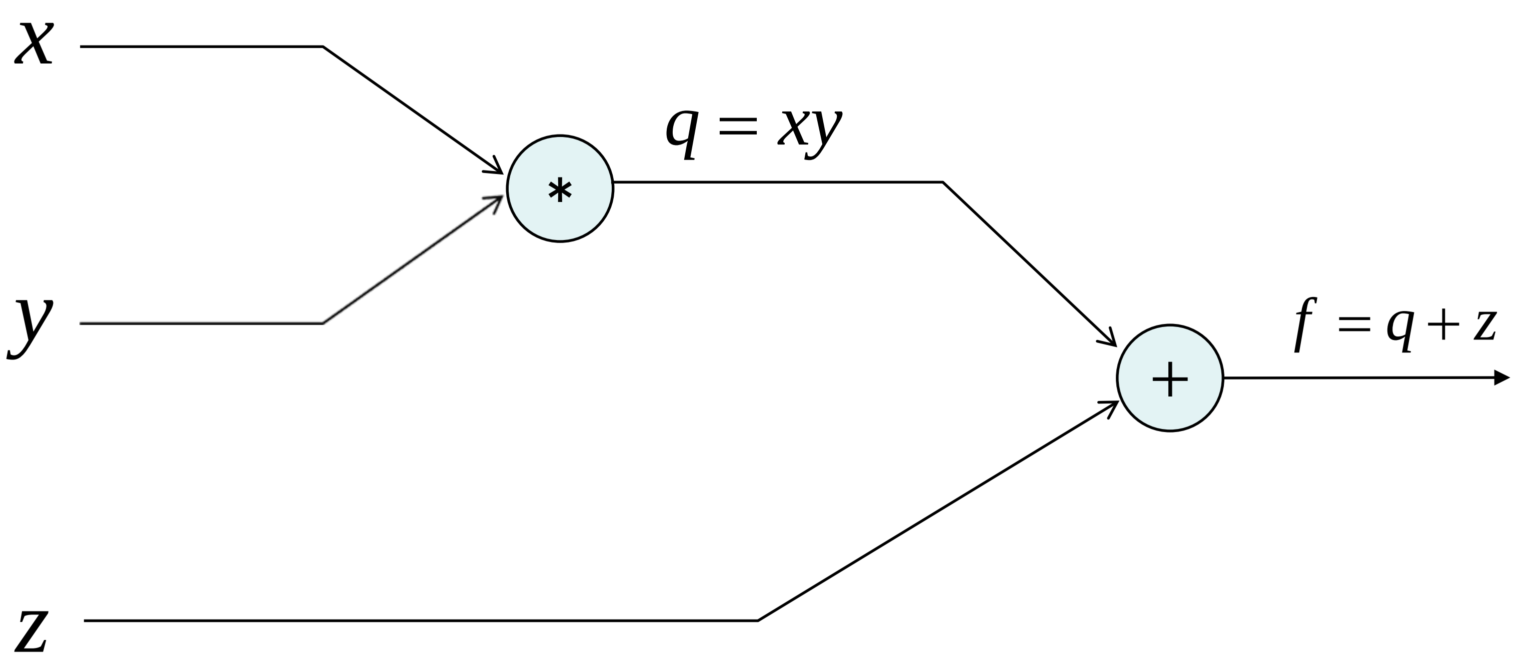
誤差逆伝播法

- 計算グラフ
  - ノードが演算、エッジが渡される値
  - 全てのノードはレイヤーとして、順伝播・逆伝播がそれぞれの中で定義される
  - 計算が局所に分解される（自在に入れ替え可能）

Ex. A typical linear classifier

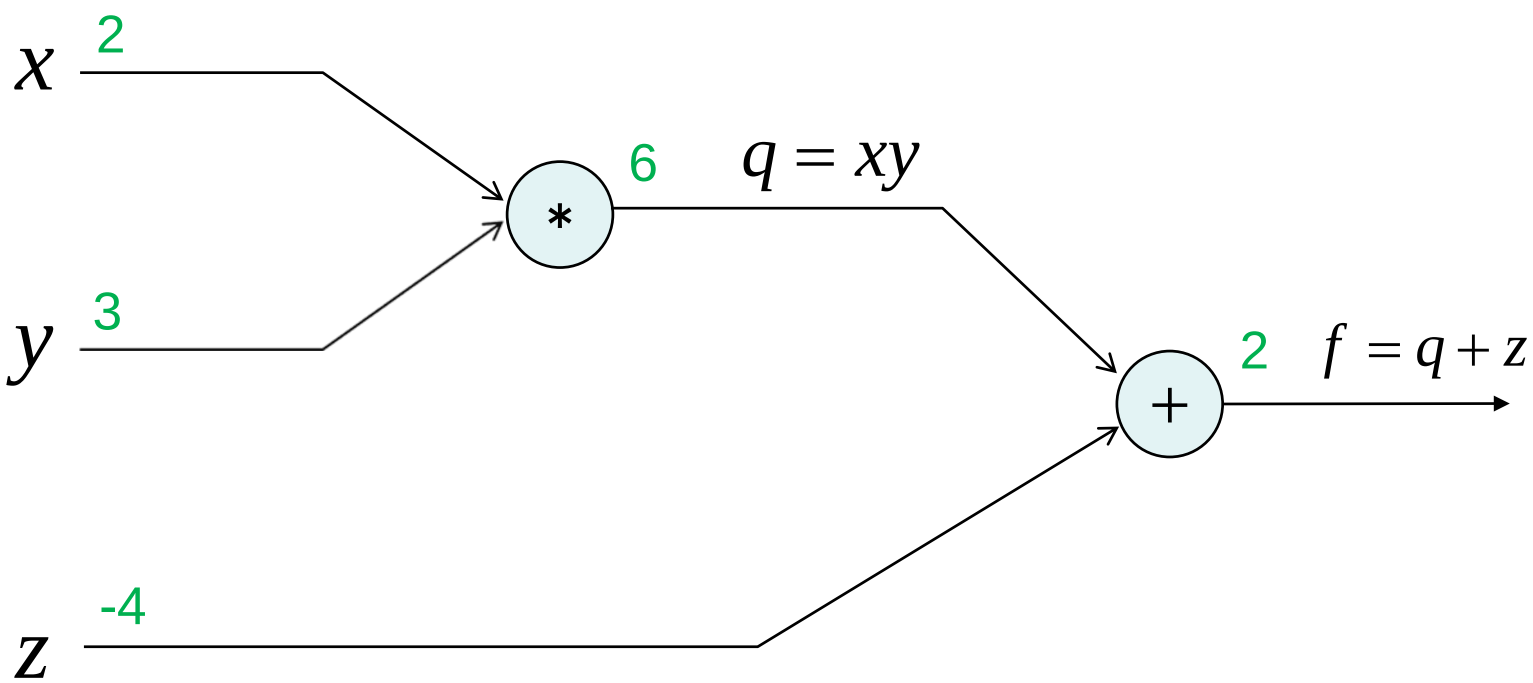


簡単な例（全部スカラー）$f = xy + z$



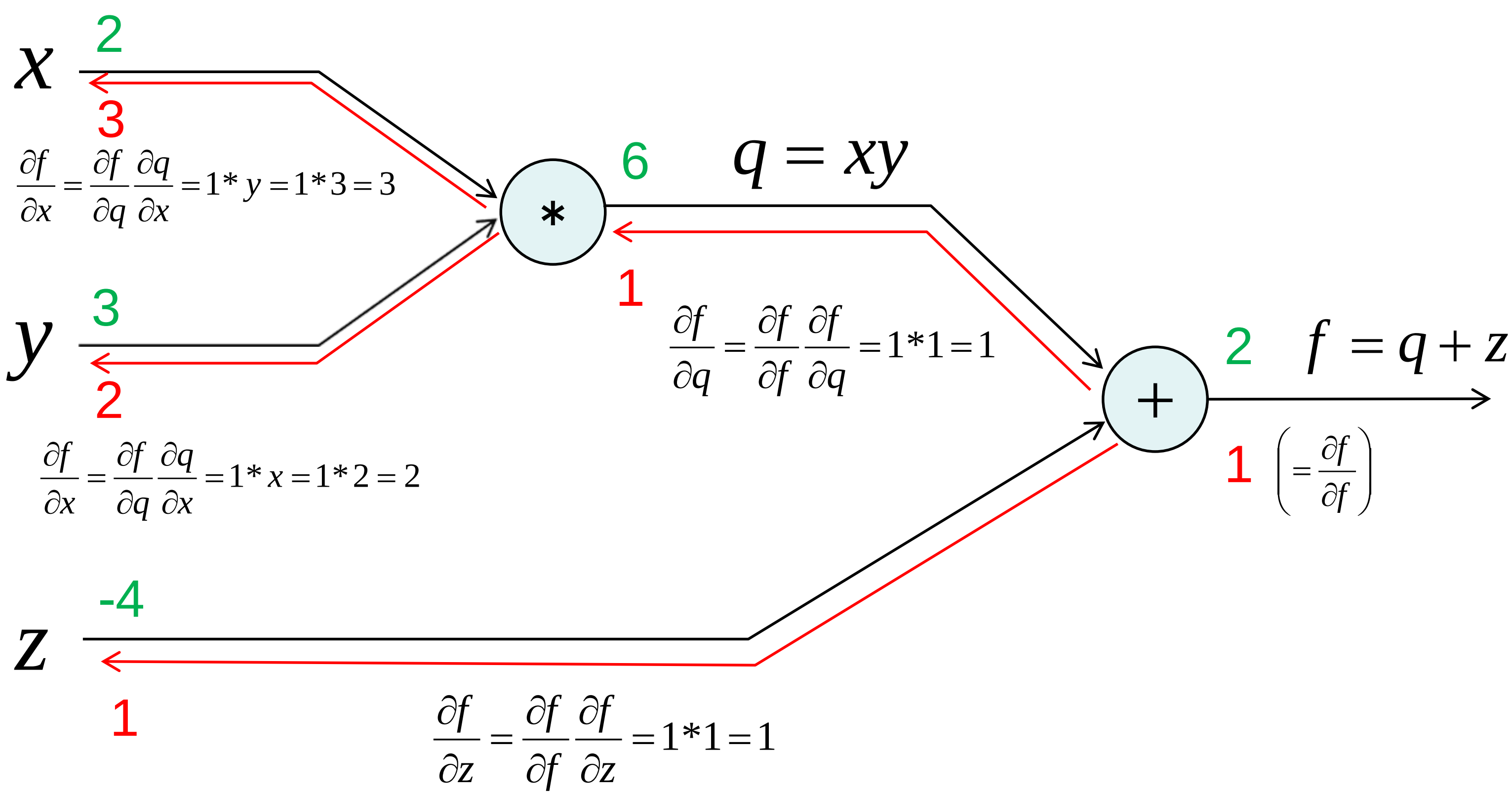
簡単な例（全部スカラー） $f = xy + z$

Feed forward:

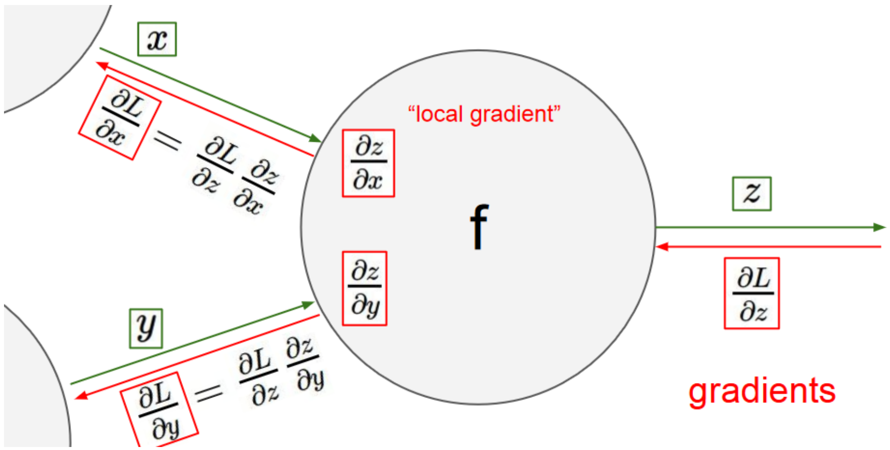


簡単な例（全部スカラー） $f = xy + z$

Backward:



All we need is local gradient!

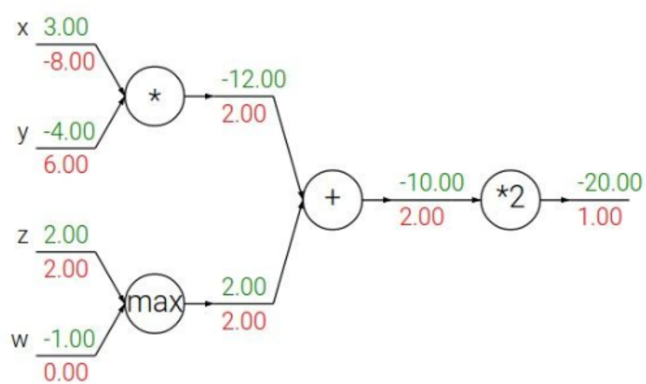


Patterns in backward flow

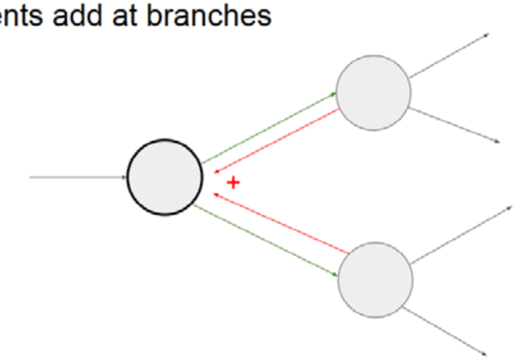
add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher



Gradients add at branches



Stanford lecture CS231nの
講義スライドより引用

実装

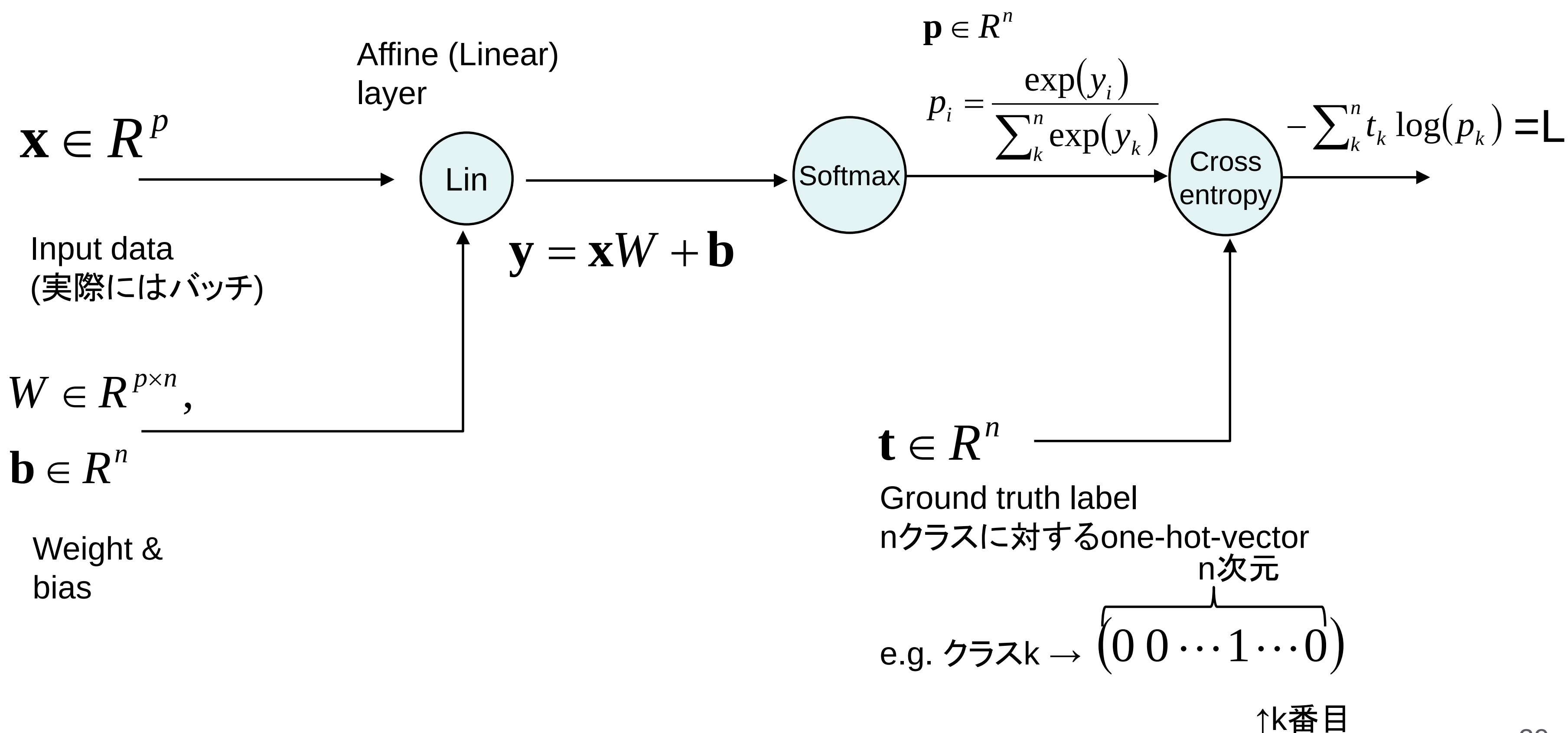
```python
class AddLayer(object):
    def forward(self,x,y):
        z = x+y
        return z
    def backward(self,dz): #dz represents dL/dz
        dx = dz #dL/dz * dz/dx
        dy = dz #dL/dz * dz/dy
        return [dx,dy]
```

- 順伝播(forward)、逆伝播(backward)を定義
- 順伝播で計算した内容を保存しておくとよい場合が多い（逆伝播で再利用する場合）
- 逆伝播の実装は間違いやすい…
  - 数値微分と比較して確認

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h} \cong \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon} \quad (\varepsilon << 1)$$

ロジスティック回帰の実装
(データは行ベクトルで与える)

- Softmax: 多クラスの識別モデル
  - シグモイド関数の一般化（二値の場合は一致）



ベクトル、行列の場合

- データは普通は多次元実数値（ベクトル）
- ミニバッチを前提とすると行列

- ベクトル、行列のchain ruleはどうなる？
  - 一般に、入力 $\mathbf{x}$ と勾配 $\frac{\partial L}{\partial \mathbf{x}}$ は同じサイズになる

準備：ベクトル、行列の微分

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix} \qquad X = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}$$

①ベクトル、行列をスカラーで微分

$$\frac{\partial \mathbf{x}}{\partial a} = \begin{pmatrix} \partial x_1 / \partial a \\ \vdots \\ \partial x_p / \partial a \end{pmatrix} \qquad \frac{\partial X}{\partial a} = \begin{pmatrix} \partial x_{11} / \partial a & \cdots & \partial x_{1n} / \partial a \\ \vdots & \ddots & \\ \partial x_{m1} / \partial a & \cdots & \partial x_{mn} / \partial a \end{pmatrix}$$

②スカラーをベクトル、行列で微分

$$\frac{\partial a}{\partial \mathbf{x}} = \begin{pmatrix} \partial a / \partial x_1 \\ \vdots \\ \partial a / \partial x_p \end{pmatrix} \qquad \frac{\partial a}{\partial X} = \begin{pmatrix} \partial a / \partial x_{11} & \cdots & \partial a / \partial x_{1n} \\ \vdots & \ddots & \\ \partial a / \partial x_{m1} & \cdots & \partial a / \partial x_{mn} \end{pmatrix}$$

③ベクトルをベクトルで微分

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \partial y_1 / \partial x_1 & \cdots & \partial y_q / \partial x_1 \\ \vdots & \ddots & \\ \partial y_1 / \partial x_p & \cdots & \partial y_q / \partial x_p \end{pmatrix}$$

ベクトル：element-wise operation

- Ex. $\mathbf{y} = \mathbf{x}**2$ (element wise)

p x p matrix (Jacobian matrix)

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \frac{\partial L}{\partial \mathbf{y}} = \boxed{\begin{pmatrix} \partial y_1 / \partial x_1 & \cdots & \partial y_q / \partial x_1 \\ \vdots & \ddots & \\ \partial y_1 / \partial x_p & \cdots & \partial y_q / \partial x_p \end{pmatrix}} \boxed{\frac{\partial L}{\partial \mathbf{y}}}$$

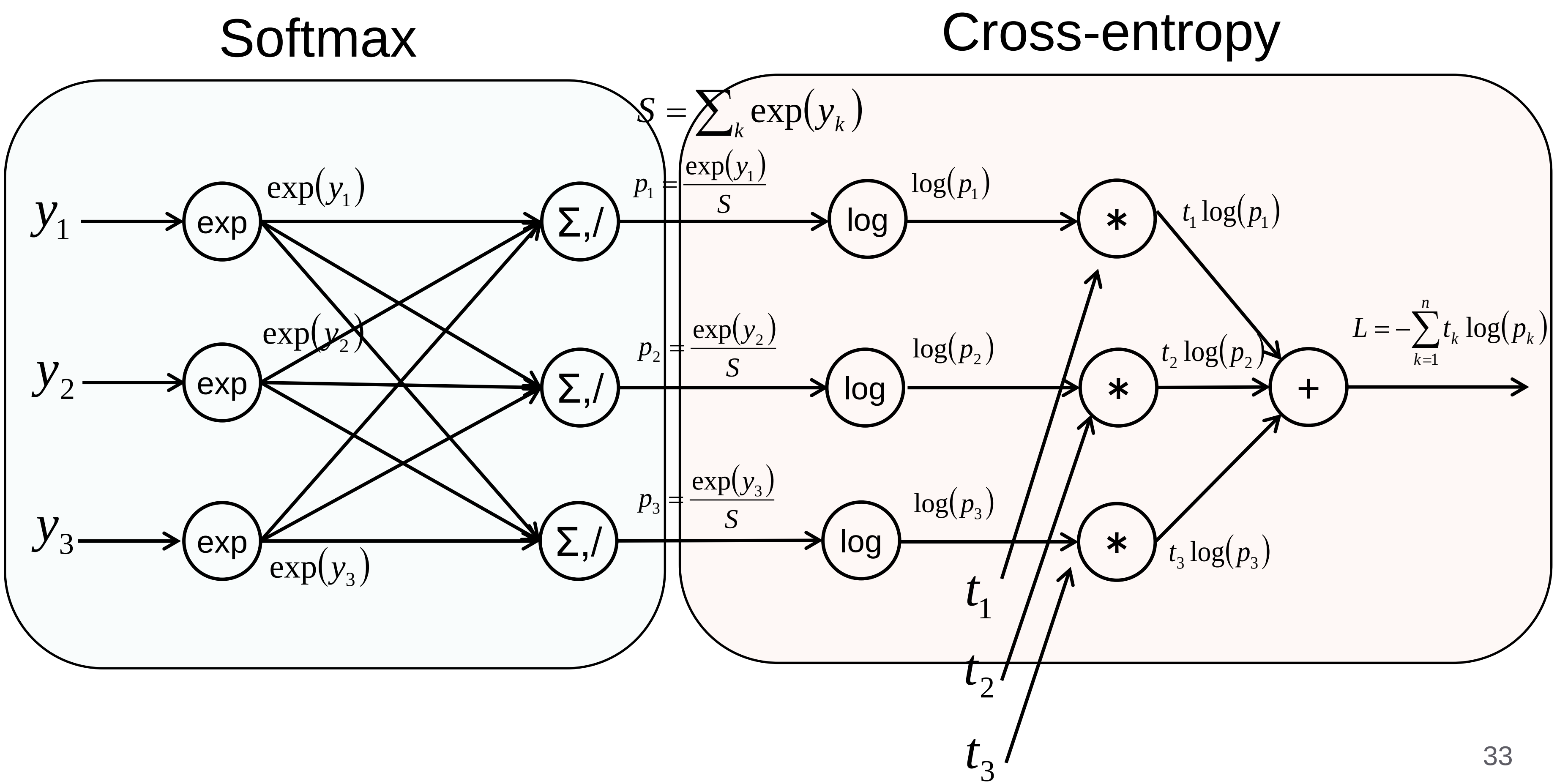
p-dim vector

$$= \begin{pmatrix} 2x_1 & \cdots & 0 \\ \vdots & \ddots & \\ 0 & \cdots & 2x_p \end{pmatrix} \frac{\partial L}{\partial \mathbf{y}} = \boxed{\begin{pmatrix} 2x_1 \\ \vdots \\ 2x_p \end{pmatrix} * \frac{\partial L}{\partial \mathbf{y}}}$$

Element-wise mult

Softmax & cross entropy

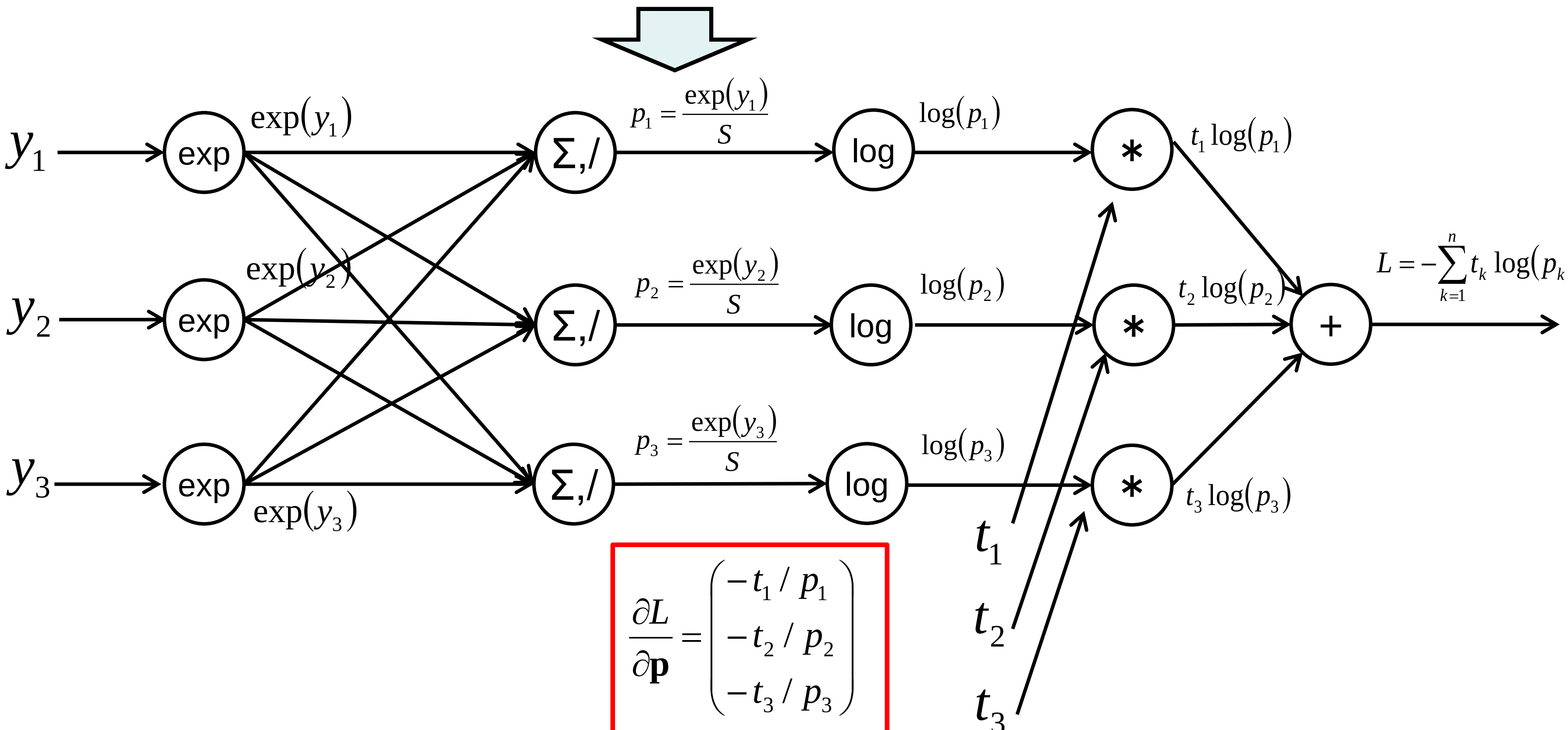
- 通常は一つのレイヤで実装（効率がよい）



Softmax & cross entropy

- Cross-entropy部分は簡単 (element-wise)

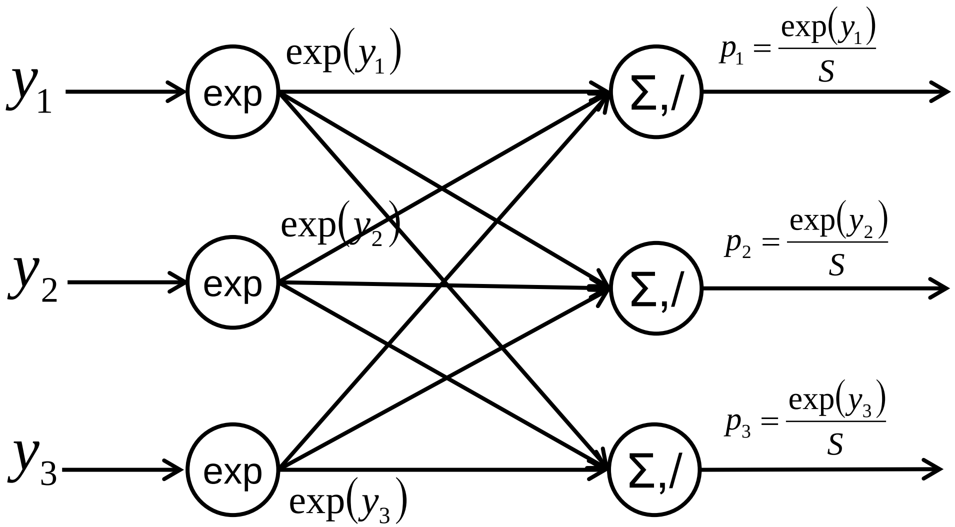
$$\boxed{\frac{\partial L}{\partial p_i} = \frac{-\sum_k t_k \log(p_k)}{\partial p_i} = \frac{-t_i \log(p_i)}{\partial p_i} = \frac{-t_i}{p_i}}$$



$$\boxed{\frac{\partial L}{\partial \mathbf{p}} = \begin{pmatrix} -t_1 / p_1 \\ -t_2 / p_2 \\ -t_3 / p_3 \end{pmatrix}}$$

Softmax & cross entropy

- Softmax



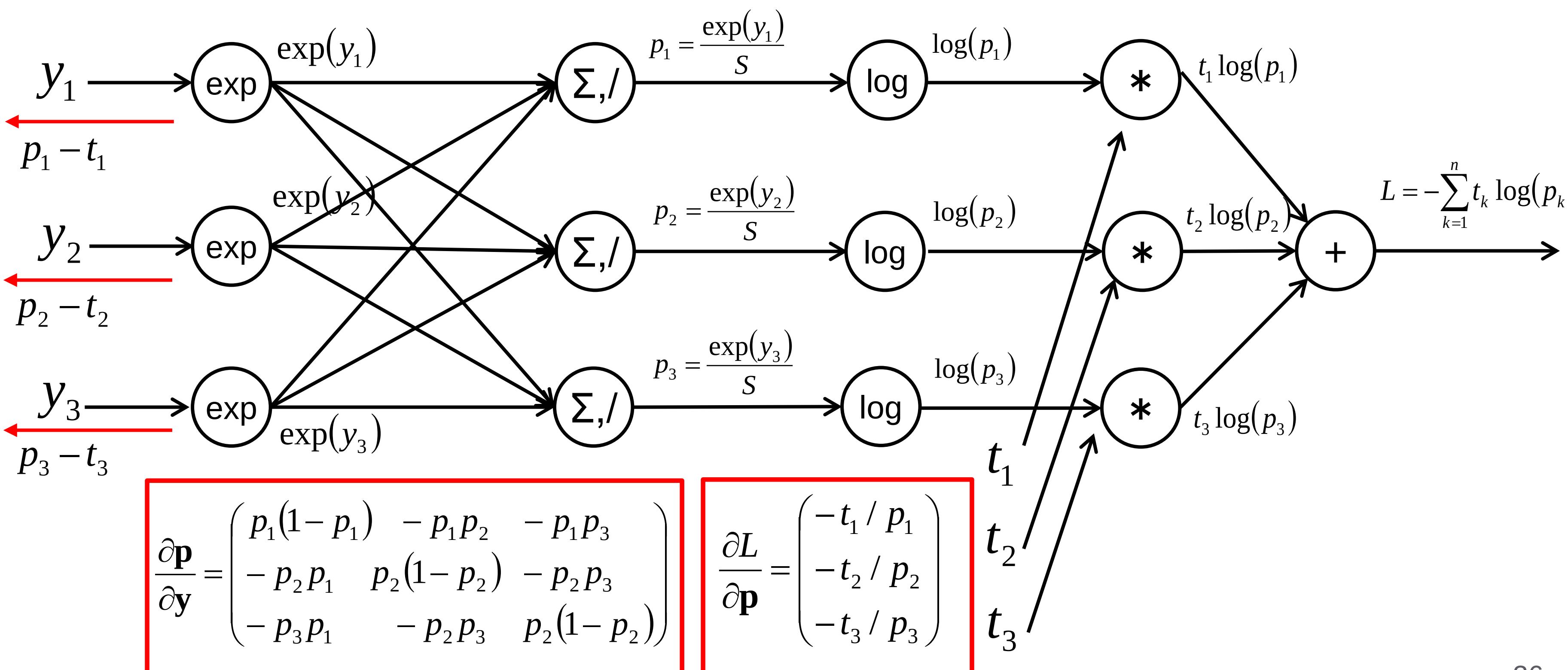
$$\frac{\partial \mathbf{p}}{\partial \mathbf{y}} = \begin{pmatrix} \partial p_1 / \partial y_1 & \cdots & \partial p_q / \partial y_1 \\ \vdots & \ddots & \\ \partial p_1 / \partial y_p & \cdots & \partial p_q / \partial y_p \end{pmatrix}$$

$$\frac{\partial p_j}{\partial y_i} = \begin{cases} p_i (1 - p_i) & \text{if} \quad i = j \\ - p_i p_j & \text{otherwise} \end{cases}$$

Softmax & cross entropy

- 以上まとめると… (element-wise)

$$\frac{\partial L}{\partial \mathbf{y}} = \frac{\partial \mathbf{p}}{\partial \mathbf{y}}\frac{\partial L}{\partial \mathbf{p}} = \begin{pmatrix} -t_1 + (t_1+t_2+t_3)p_1 \\ -t_2 + (t_1+t_2+t_3)p_2 \\ -t_3 + (t_1+t_2+t_3)p_3 \end{pmatrix} = \begin{pmatrix} p_1 - t_1 \\ p_2 - t_2 \\ p_3 - t_3 \end{pmatrix}$$



$$\frac{\partial \mathbf{p}}{\partial \mathbf{y}} = \begin{pmatrix} p_1(1-p_1) & -p_1p_2 & -p_1p_3 \\ -p_2p_1 & p_2(1-p_2) & -p_2p_3 \\ -p_3p_1 & -p_2p_3 & p_2(1-p_2) \end{pmatrix}$$

$$\frac{\partial L}{\partial \mathbf{p}} = \begin{pmatrix} -t_1/p_1 \\ -t_2/p_2 \\ -t_3/p_3 \end{pmatrix}$$

Affine (Linear) layer

$$Y = XW + \begin{pmatrix} (& \mathbf{b} &) \\ (& \mathbf{b} &) \\ & \vdots & \\ (& \mathbf{b} &) \end{pmatrix}$$

$$\frac{\partial L}{\partial X} = \left(\frac{\partial L}{\partial Y} \right) W^T$$

$X \in R^{b \times p}$ 入力バッチ（行ベクトルベース）
バッチサイズ b

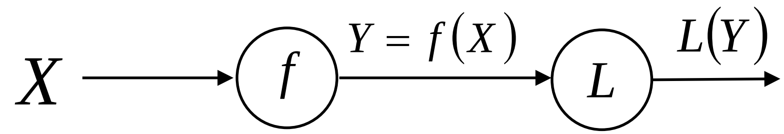
$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{i=1}^{b} \left(\frac{\partial L}{\partial \mathbf{y}_i} \right)$$

$W \in R^{p \times n}$ 重み行列

$\mathbf{b} \in R^{n}$ バイアスベクトル

$Y \in R^{b \times n}$ 出力バッチ

Matrix backpropagation



- 欲しいのは、合成関数の微分 $\dfrac{\partial L\circ f}{\partial X}$

- 行列関数のテイラー展開

$$L(Y+dY)-L(Y)=\frac{\partial L}{\partial Y}:dY+O\left(\|dY\|^2\right)$$

$$L\circ f(X+dX)-L\circ f(X)=\frac{\partial L\circ f}{\partial X}:dX+O\left(\|dX\|^2\right)$$

$A:B\equiv Tr\left(A^T B\right)$ （行列の内積）

- (少なくとも)一次の項は等しくなる

$\dfrac{\partial L}{\partial Y}:dY=\dfrac{\partial L\circ f}{\partial X}:dX$ を満たす $\dfrac{\partial L\circ f}{\partial X}$ が求めるもの

例) Affine layer

$Y = XW$ より $dY = (dX)W$

$$\frac{\partial L}{\partial Y} : dY = \frac{\partial L \circ f}{\partial X} : dX$$

$$\frac{\partial L}{\partial Y} : dY = Tr\left(\left(\frac{\partial L}{\partial Y}\right)^T (dX)W\right)$$

$$= Tr\left(W\left(\frac{\partial L}{\partial Y}\right)^T (dX)\right)$$

$$= Tr\left(\left(\left(\frac{\partial L}{\partial Y}\right)W^T\right)^T (dX)\right)$$

$$= \left(\frac{\partial L}{\partial Y}\right)W^T : dX$$ よって $$\frac{\partial L \circ f}{\partial X} = \left(\frac{\partial L}{\partial Y}\right)W^T$$

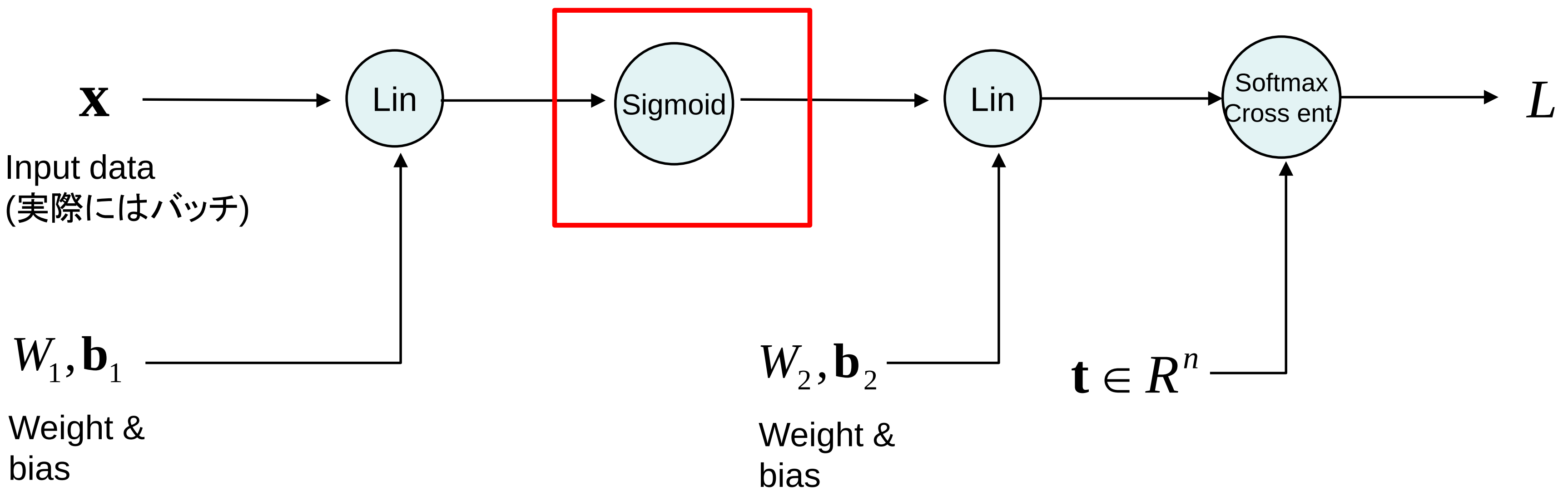
三層パーセプトロン

活性化関数（とても大事）

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

例）シグモイド関数
(element-wise)

$$\frac{\partial \sigma}{\partial x} = \sigma(x)(1-\sigma(x))$$



補足：Softmaxとシグモイド関数

- Softmaxモデル (線形変換も含む)

$$P(C_i \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_i^T \mathbf{x})}{\sum_k^n \exp(\mathbf{w}_k^T \mathbf{x})}$$

- n=2 (2クラス識別)の場合

$$P(C_0 \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_0^T \mathbf{x})}{\exp(\mathbf{w}_0^T \mathbf{x}) + \exp(\mathbf{w}_1^T \mathbf{x})} = \frac{\exp((\mathbf{w}_0 - \mathbf{w}_1)^T \mathbf{x})}{\exp((\mathbf{w}_0 - \mathbf{w}_1)^T \mathbf{x}) + 1} = \frac{\exp(-\mathbf{w}^T \mathbf{x})}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$P(C_1 \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_1^T \mathbf{x})}{\exp(\mathbf{w}_0^T \mathbf{x}) + \exp(\mathbf{w}_1^T \mathbf{x})} = \frac{1}{\exp((\mathbf{w}_0 - \mathbf{w}_1)^T \mathbf{x}) + 1} = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

であるから、$\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$ と考えればシグモイド関数に帰着