ディープラーニングの特徴

- 理論が単純（多クラスロジスティック回帰の延長）
- モデルを柔軟に変えることができる
- 汎用性が高く、どんな分野にも応用できる（万能近似器）
- 脳の構造に似ているので、脳の研究にもつながる可能性がある？

ディープラーニングの能力

2013　Ponanza （山本、ロジスティック回帰）
将棋で佐藤天彦に勝利

一つの大きな技術の発展ではなく、
多くの小さな技術の積み重ね

2016　AlphaGo（Google DeepMind、ディープラーニング）
囲碁でイ・セドルに勝利

囲碁の難しさ:
実現可能局面数が多い
評価関数を作るのが困難

オセロ（6×6）：10^{12} 完全解析済み
オセロ（8×8）：10^{28}
将棋：10^{70}
囲碁：10^{170}
（人の細胞数: 60兆~ 10^{13}）
（宇宙全体の星の数: 10^{26}）

駒の価値

駒	Ver.2016
歩	93
香車	322
桂馬	416
銀	528
金	567
角	951
飛車	1087
玉	∞
と金	598
成香	567
成桂	569
成銀	582
馬	1101
龍	1550

https://ja.wikipedia.org/wiki/Ponanza

ディープラーニングを支える主な技術

多層化・深層化に関する技術（勾配消失問題の解決、残差ネットワーク）

その他のアルゴリズムの発展（畳み込み、バッチ正規化、埋め込み、Transfomer）

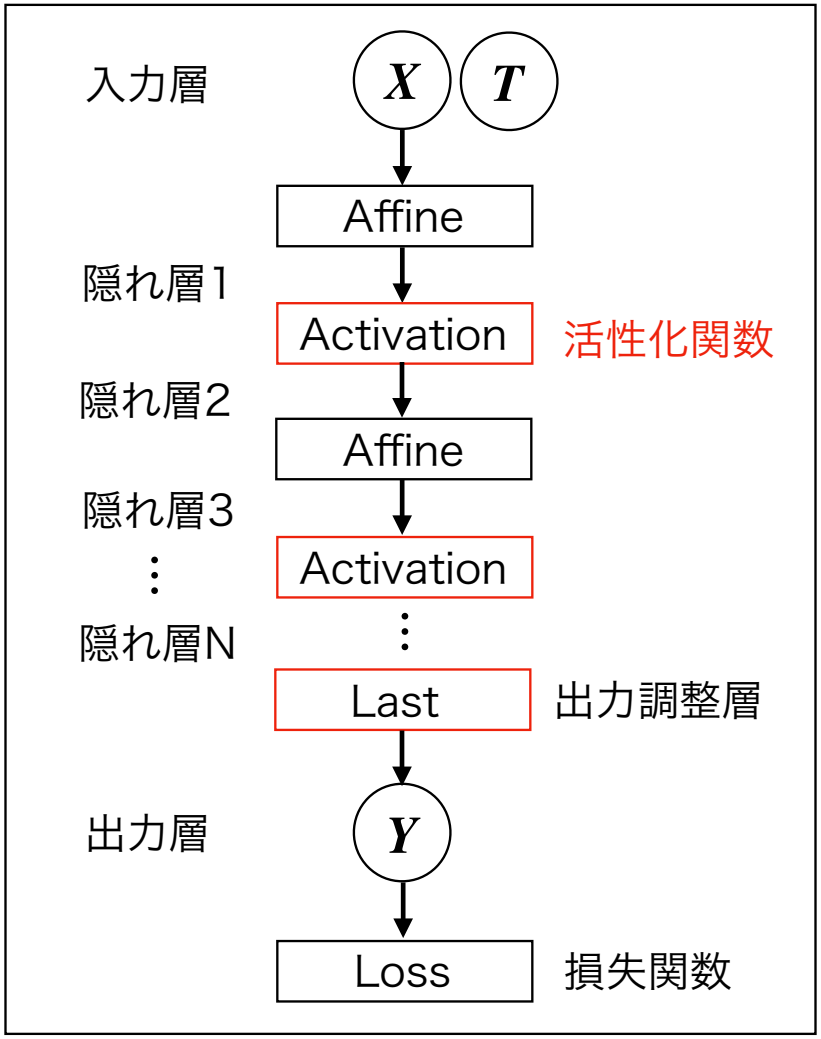
ソフト・ハードの性能向上（DLフレームワーク、GPU）

Pythonで使える主なDLフレームワーク:

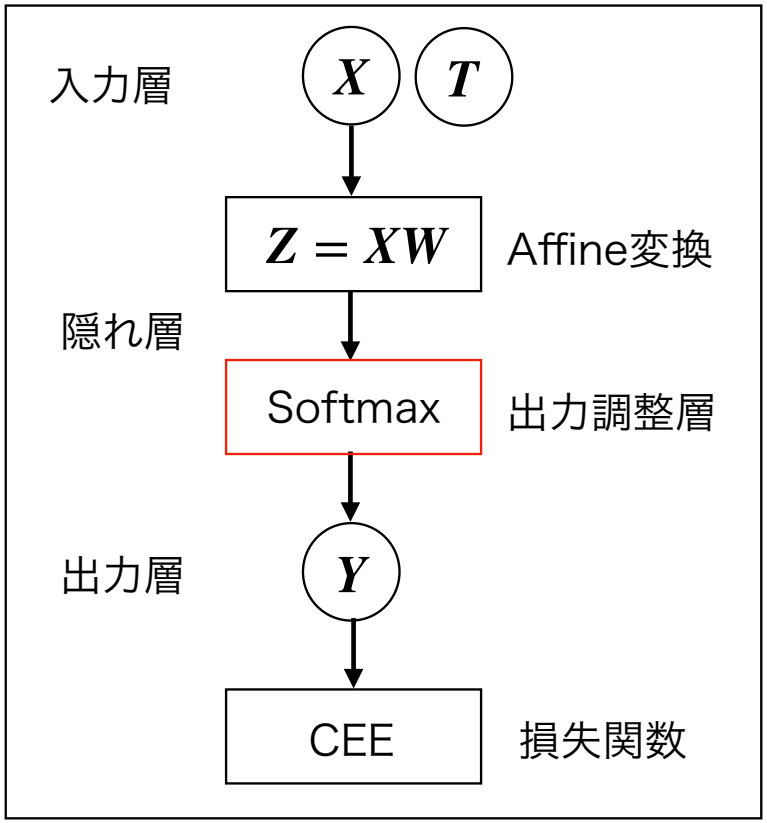
- TensorFlow（Google）
- Keras（Google）
- PyTorch (Facebook)

ディープラーニングの基本構造（アーキテクチャ）

ディープラーニング



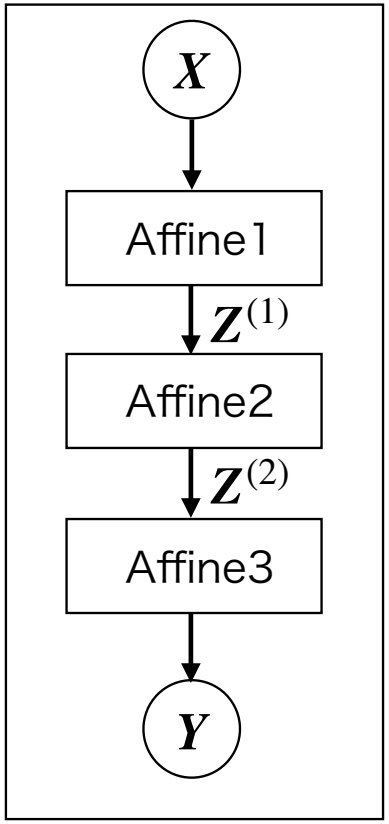
多クラスロジスティック回帰



多クラスロジスティック回帰は隠れ層が1つのディープラーニングとみなせる。
隠れ層によりモデルの表現力が大幅に向上。

活性化関数の必要性

愚直な多層化

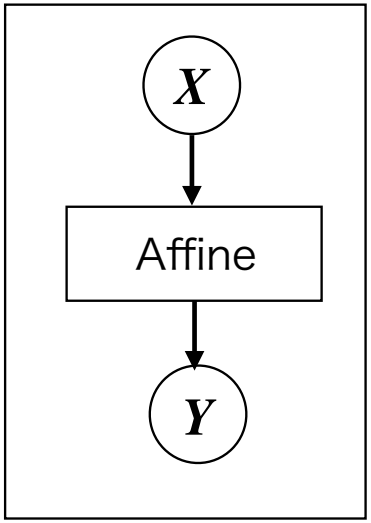


$$Z^{(1)} = XW^{(1)}$$

$$Z^{(2)} = Z^{(1)}W^{(2)}$$

$$\begin{aligned} Y &= Z^{(2)}W^{(3)} \\ &= Z^{(1)}W^{(2)}W^{(3)} \\ &= XW^{(1)}W^{(2)}W^{(3)} \end{aligned}$$

等価モデル



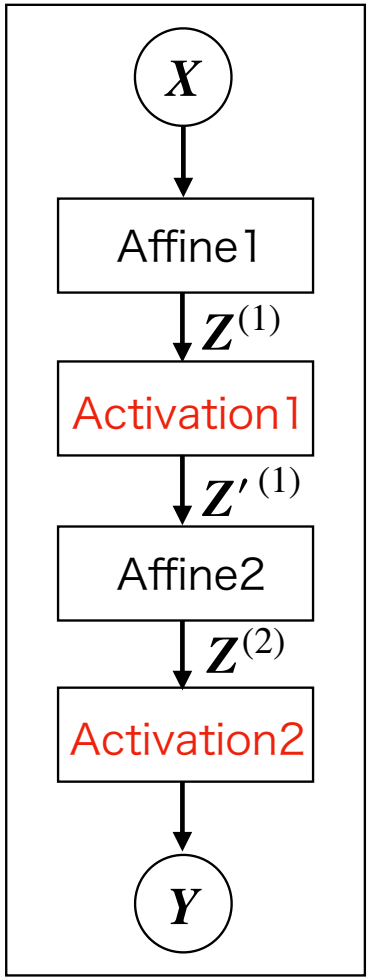
$$W \equiv W^{(1)}W^{(2)}W^{(3)}$$

$$Y = XW$$

多層化して表現力を上げるつもりが、Affine 変換を一回行うだけのモデルとなってしまう。

多層化に意味を持たせるためには、活性化関数を入れてモデルを非線形にする必要がある

活性化関数の必要性



$$Z^{(1)} = XW^{(1)}$$

$$Z'^{(1)} = A^{(1)}\left(Z^{(1)}\right)$$

$$Z^{(2)} = Z'^{(1)}W^{(2)}$$

$$Y = A^{(2)}\left(Z^{(2)}\right)$$

$$\begin{aligned} Y &= A^{(2)}\left(Z^{(2)}\right) \\ &= A^{(2)}\left(Z'^{(1)}W^{(2)}\right) \\ &= A^{(2)}\left(A^{(1)}\left(Z^{(1)}\right)W^{(2)}\right) \\ &= A^{(2)}\left(A^{(1)}\left(XW^{(1)}\right)W^{(2)}\right) \end{aligned}$$

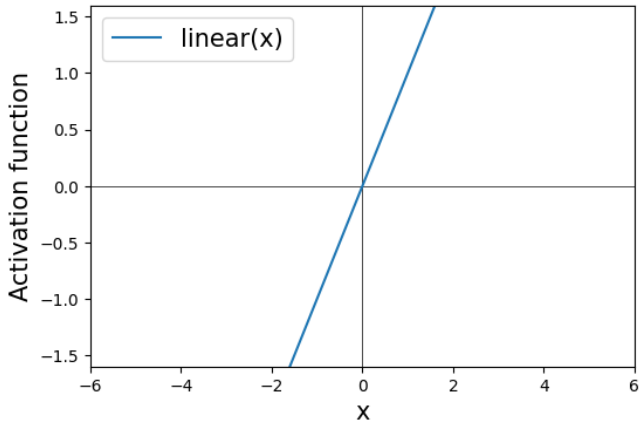
これは一つのAffine変換では表せない

→ 多層化することに意味がある

→ 増えたウェイトの分だけ表現力が向上

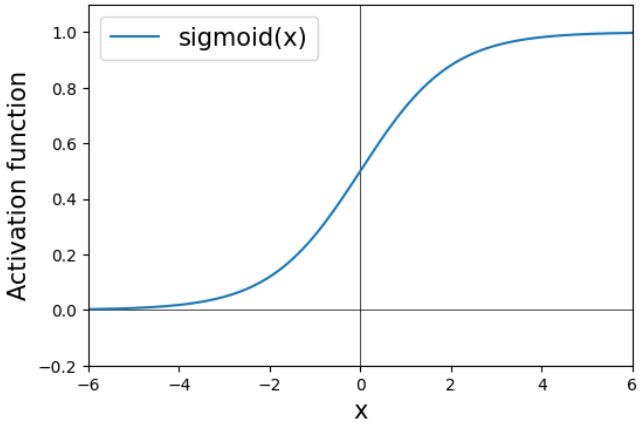
活性化関数の例（飽和系）

$$\mathrm{linear}(x) = x$$

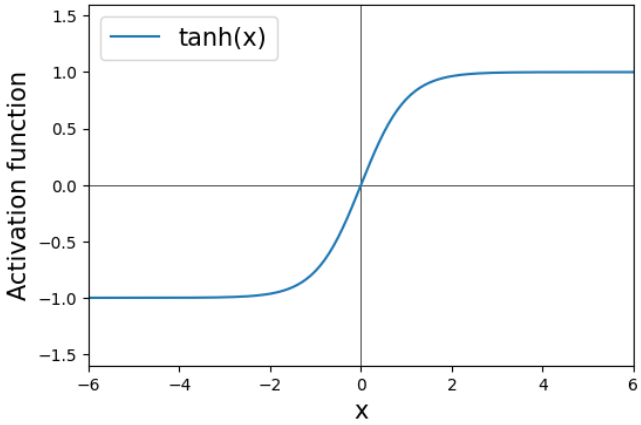


線形関数でなければなんでも良い。

$$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



$$\tanh(x)$$



多層化（深層化）

多層化（深層化）の利点と問題点

利点:

モデルの表現力を効率的に上げることができる。
（モデルを大きくするよりも深くした方が良い）

必要なウェイト数の削減
分解による問題の簡素化
階層化による学習の高度化
（概念的特徴量の獲得）

問題点:

層が多くなりモデルが複雑化するため、ウェイト更新のための勾配計算が大変
→ 自動微分（誤差逆伝播法）

多層化により勾配が0になりやすくなる（勾配消失問題）
→ 活性化関数としてReLU関数などを用いる

多層化によって精度の劣化問題が起きる（恒等変換を学習できないことが原因）
→ 残差ネットワーク（スキップ接続）

線形回帰とロジスティック回帰の手動微分

$$L = \frac{1}{2}\sum_{i=1}^{M}\left(t_i - y_i\right)^2 = \frac{1}{2}\sum_{i=1}^{M}\left(t_i - \sum_{j=0}^{N} x_{ij}w_j\right)^2$$

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{M}\frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial w_j} = -\sum_{i=1}^{M}(t_i - y_i)x_{ij}$$

$$L = -\sum_{i=1}^{M}\left[t_i \ln y_i + (1 - t_i)\ln(1 - y_i)\right]$$

$$\frac{\partial L}{\partial w_j} = \sum_{i}^{M}\frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial w_j} = \sum_{i}^{M}\frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial z_i}\frac{\partial z_i}{\partial w_j}$$

$$= -\sum_{i}^{M}\frac{t_i - y_i}{y_i(1 - y_i)}y_i(1 - y_i)x_{ij} = -\sum_{i}^{M}(t_i - y_i)x_{ij}$$

層が増えると手動微分は大変

二重の場合の連鎖律の一般形

$$
\begin{aligned}
z_1 &= z_1(x_1, x_2, \ldots, x_N) \\
z_2 &= z_2(x_1, x_2, \ldots, x_N) \\
&\vdots \\
z_Q &= z_Q(x_1, x_2, \ldots, x_N)
\end{aligned}
\qquad \longrightarrow \qquad
\boxed{\frac{\partial y}{\partial x_i} = \sum_q^Q \frac{\partial z_q}{\partial x_i} \frac{\partial y}{\partial z_q}}
$$

$$
y = y(z_1, z_2, \ldots, z_Q)
$$

多重の場合の連鎖律の一般形（1/6）

*N*重の場合

$$z_{q_1}^{(1)} = z_{q_1}^{(1)}\left(z_1^{(0)}, z_2^{(0)}, \ldots, z_{q_0}^{(0)}, \ldots, z_{Q_0}^{(0)}\right)$$

$$z_{q_2}^{(2)} = z_{q_2}^{(2)}\left(z_1^{(1)}, z_2^{(1)}, \ldots, z_{q_1}^{(1)}, \ldots, z_{Q_1}^{(1)}\right)$$

$$\vdots$$

$$z_{q_{N-1}}^{(N-1)} = z_{q_{N-1}}^{(N-1)}\left(z_1^{(N-2)}, z_2^{(N-2)}, \ldots, z_{q_{N-2}}^{(N-2)}, \ldots, z_{Q_{N-2}}^{(N-2)}\right)$$

$$z_{q_N}^{(N)} = z_{q_N}^{(N)}\left(z_1^{(N-1)}, z_2^{(N-1)}, \ldots, z_{q_{N-1}}^{(N-1)}, \ldots, z_{Q_{N-1}}^{(N-1)}\right)$$

ベクトル表現

$$z_i \equiv \left[z_1^{(i)}, z_2^{(i)}, \ldots, z_{q_i}^{(i)}, \ldots, z_{Q_i}^{(i)}\right]^T$$

$$z_1 = z_1\left(z_0\right)$$

$$z_2 = z_2\left(z_1\right)$$

$$\vdots$$

$$z_{N-1} = z_{N-1}\left(z_{N-2}\right)$$

$$z_N = z_N\left(z_{N-1}\right)$$

多重の場合の連鎖律の一般形（2/6）

$$\frac{\partial z_{q_N}^{(N)}}{\partial z_{q_0}^{(0)}} = \sum_{q_{N-1}}^{Q_{N-1}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_0}^{(0)}} \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}} = \sum_{q_{N-1}}^{Q_{N-1}} \left(\sum_{q_{N-2}}^{Q_{N-2}} \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_0}^{(0)}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} \right) \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}$$

$$= \sum_{q_{N-1}}^{Q_{N-1}} \sum_{q_{N-2}}^{Q_{N-2}} \cdots \sum_{q_2}^{Q_2} \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}$$

$$\frac{\partial z_{q_N}^{(N)}}{\partial z_{q_0}^{(0)}} = \sum_{q_{N-1}}^{Q_{N-1}} \sum_{q_{N-2}}^{Q_{N-2}} \cdots \sum_{q_2}^{Q_2} \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}$$

$$= \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \sum_{q_2}^{Q_2} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \sum_{q_{N-2}}^{Q_{N-2}} \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} \sum_{q_{N-1}}^{Q_{N-1}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}$$

並び替え

多重の場合の連鎖律の一般形（3/6）

$$\frac{\partial z_{q_N}^{(N)}}{\partial z_{q_0}^{(0)}} = \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \sum_{q_2}^{Q_2} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \sum_{q_{N-2}}^{Q_{N-2}} \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} \sum_{q_{N-1}}^{Q_{N-1}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}$$

$$= \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \sum_{q_2}^{Q_2} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \sum_{q_{N-2}}^{Q_{N-2}} \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} \sum_{q_{N-1}}^{Q_{N-1}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} g_{N-1}^{(N-1)}$$

$$= \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \sum_{q_2}^{Q_2} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \sum_{q_{N-2}}^{Q_{N-2}} \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} g_{N-2}^{(N-2)}$$

$$= \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} g_{q_1}^{(1)}$$

$$g_{q_{N-1}}^{(N-1)} \equiv \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}$$

逆伝播公式

$$g_{q_{i-1}}^{(i-1)} \equiv \sum_{q_i}^{Q_i} \frac{\partial z_{q_i}^{(i)}}{\partial z_{q_{i-1}}^{(i-1)}} g_{q_i}^{(i)} \quad \left(= \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{i-1}}^{(i-1)}} \right)$$

下層の勾配がわかれば、上層の勾配が計算可能

多重の場合の連鎖律の一般形（4/6）

局所微分の（ヤコビ）行列による表現

$$\left[\boldsymbol{J}_i\right]_{q_i,q_{i-1}} \equiv \frac{\partial z_{q_i}^{(i)}}{\partial z_{q_{i-1}}^{(i-1)}}$$

$$\boldsymbol{J}_i = \begin{bmatrix} \frac{\partial z_1^{(i)}}{\partial z_1^{(i-1)}} & \frac{\partial z_1^{(i)}}{\partial z_2^{(i-1)}} & \cdots & \frac{\partial z_1^{(i)}}{\partial z_{Q_{i-1}}^{(i-1)}} \\ \frac{\partial z_2^{(i)}}{\partial z_1^{(i-1)}} & \frac{\partial z_2^{(i)}}{\partial z_2^{(i-1)}} & \cdots & \frac{\partial z_2^{(i)}}{\partial z_{Q_{i-1}}^{(i-1)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_3^{(i)}}{\partial z_1^{(i-1)}} & \frac{\partial z_3^{(i)}}{\partial z_2^{(i-1)}} & \cdots & \frac{\partial z_3^{(i)}}{\partial z_{Q_{i-1}}^{(i-1)}} \end{bmatrix}$$

勾配ベクトルの定義

$$\boldsymbol{g}_i \equiv \left[g_1^{(i)}, g_2^{(i)}, \ldots, g_{q_i}^{(i)}, \ldots, g_{Q_i}^{(i)}\right]^T = \left[\frac{\partial z_{q_N}^{(N)}}{\partial z_1^{(i)}}, \frac{\partial z_{q_N}^{(N)}}{\partial z_2^{(i)}}, \ldots, \frac{\partial z_{q_N}^{(N)}}{\partial z_{Q_i}^{(i)}}\right]^T = \frac{\partial z_{q_N}^{(N)}}{\partial \boldsymbol{z}_i}$$

多重の場合の連鎖律の一般形（5/6）

$$\frac{\partial z_{q_N}^{(N)}}{\partial z_{q_0}^{(0)}} = \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} g_{q_1}^{(1)}$$

$$\rightarrow \quad \frac{\partial z_{q_N}^{(N)}}{\partial \boldsymbol{z}_0} = \boldsymbol{g}_0 = \boldsymbol{J}_1^T \boldsymbol{g}_1$$

$$= \boldsymbol{J}_1^T \boldsymbol{J}_2^T \boldsymbol{g}_2$$

$$\vdots$$

$$= \boldsymbol{J}_1^T \boldsymbol{J}_2^T \cdots \boldsymbol{J}_{N-2}^T \boldsymbol{J}_{N-1}^T \boldsymbol{g}_{N-1}$$

逆伝播公式

$$\boldsymbol{g}_{i-1} = \boldsymbol{J}_i^T \boldsymbol{g}_i$$

多重の場合の連鎖律の一般形（6/6）

連鎖律の一般形

$$\frac{\partial z_{q_N}^{(N)}}{\partial z_{q_0}^{(0)}} = \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} g_{q_1}^{(1)} = \sum_{q_1}^{Q_1} \frac{\partial z_{q_1}^{(1)}}{\partial z_{q_0}^{(0)}} \sum_{q_2}^{Q_2} \frac{\partial z_{q_2}^{(2)}}{\partial z_{q_1}^{(1)}} \cdots \sum_{q_{N-2}}^{Q_{N-2}} \frac{\partial z_{q_{N-2}}^{(N-2)}}{\partial z_{q_{N-3}}^{(N-3)}} \sum_{q_{N-1}}^{Q_{N-1}} \frac{\partial z_{q_{N-1}}^{(N-1)}}{\partial z_{q_{N-2}}^{(N-2)}} \frac{\partial z_{q_N}^{(N)}}{\partial z_{q_{N-1}}^{(N-1)}}$$

行列表現

$$\boldsymbol{g}_0 = \boldsymbol{J}_1^T \boldsymbol{g}_1 = \boldsymbol{J}_1^T \boldsymbol{J}_2^T \cdots \boldsymbol{J}_{N-2}^T \boldsymbol{J}_{N-1}^T \boldsymbol{g}_{N-1}$$

勾配の逆伝播公式

$$\boldsymbol{g}_{i-1} = \boldsymbol{J}_i^T \boldsymbol{g}_i$$

勾配

$$\boldsymbol{g}_i \equiv \frac{\partial z_{q_N}^{(N)}}{\partial \boldsymbol{z}_i}$$

i 層目の変数による
N 層目の出力の偏微分

局所微分

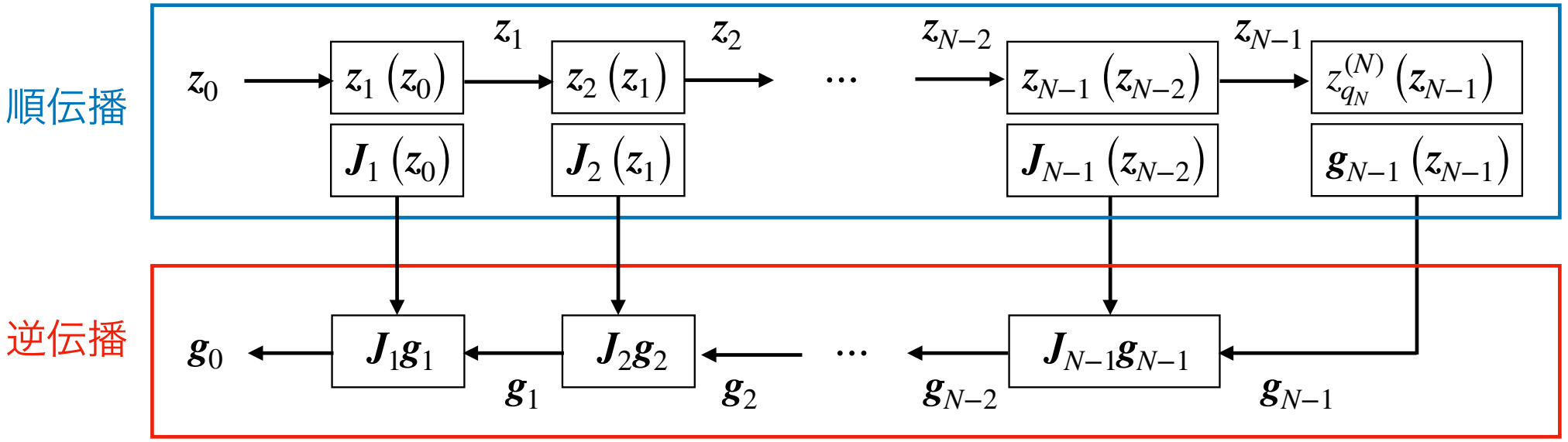
$$\left[\boldsymbol{J}_i\right]_{q_i, q_{i-1}} \equiv \frac{\partial z_{q_i}^{(i)}}{\partial z_{q_{i-1}}^{(i-1)}}$$

$i-1$ 層目の変数による
i 層目の出力の偏微分

自動微分

各関数 $z_{i+1}\left(z_i\right)$ の定義の際、予め局所微分 $\boldsymbol{J}_{i+1}\left(z_i\right)$ の形も定義しておく。そして、データ z_i を順伝播させる時に、ついでに $\boldsymbol{J}_{i+1}\left(z_i\right)$ も計算して値を記憶しておき、それを勾配の計算の際に利用するというのが自動微分の特徴。

順伝播の計算量は少し増えるが、逆伝播での各勾配の計算は $\boldsymbol{J}_i$ と $\boldsymbol{g}_i$ の単なる内積なので、全ての変数の勾配を高速、且つ、厳密に行うことができる。（実際には、計算量や必要なメモリを減らすために、勾配を知る必要のない変数に関する部分では局所微分の計算をスキップさせる。）



数値微分の場合

$$\boldsymbol{h}_{q_0} \equiv \underbrace{[0,0,...,h,\ldots,0]^T}_{q_0\text{ 番目}} \qquad z_0^{q_0\pm} \equiv z_0 \pm \boldsymbol{h}_{q_0} = \left[z_1^{(0)}, z_2^{(0)}, \ldots, z_{q_0}^{(0)} \pm h, \ldots, z_{Q_0}^{(0)}\right]^T \qquad h \sim 10^{-4}$$

$$z_0^{q_0+} \equiv z_0 + \boldsymbol{h} \longrightarrow \boxed{z_1\left(z_0^{q_0+}\right)} \xrightarrow{z_1^{q_0+}} \boxed{z_2\left(z_1^{q_0+}\right)} \xrightarrow{z_2^{q_0+}} \cdots \xrightarrow{z_{N-2}^{q_0+}} \boxed{z_{N-1}\left(z_{N-2}^{q_0+}\right)} \xrightarrow{z_{N-1}^{q_0+}} \boxed{z_{q_N}^{(N)}\left(z_{N-1}^{q_0+}\right)}$$

$$z_0^{q_0-} \equiv z_0 - \boldsymbol{h} \longrightarrow \boxed{z_1\left(z_0^{q_0-}\right)} \xrightarrow{z_1^{q_0-}} \boxed{z_2\left(z_1^{q_0-}\right)} \xrightarrow{z_2^{q_0-}} \cdots \xrightarrow{z_{N-2}^{q_0-}} \boxed{z_{N-1}\left(z_{N-2}^{q_0-}\right)} \xrightarrow{z_{N-1}^{q_0-}} \boxed{z_{q_N}^{(N)}\left(z_{N-1}^{q_0-}\right)}$$

$$\frac{\partial z_{q_N}^{(N)}}{\partial z_{q_0}^{(0)}} \sim \frac{z_{q_N}^{(N)}\left(z_{N-1}^{q_0+}\right) - z_{q_N}^{(N)}\left(z_{N-1}^{q_0-}\right)}{2h}$$

実装は簡単だが、打切（離散）誤差や丸め誤差があるため厳密な値ではない。

関数が Q_i 個の引数を取る場合、順方向の計算を $2Q_i$ 回行わなければならない。

勾配を知りたい変数が多い場合、計算が大変。（勾配チェックには使える）

計算グラフによる自動微分の表現（順伝播）

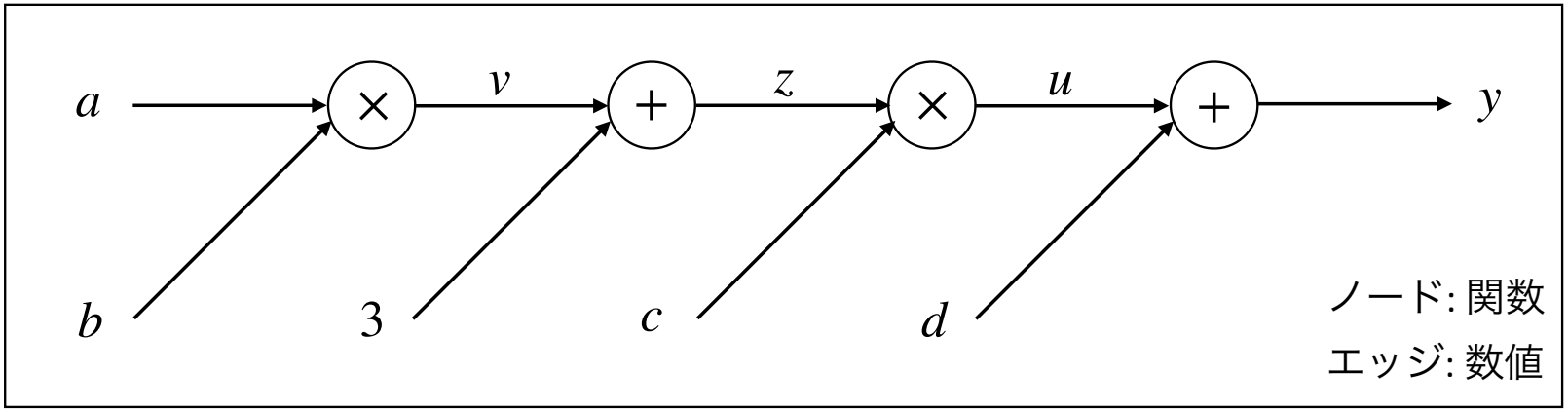
$$z(a,b) = ab + 3$$
$$y(z,c,d) = cz + d$$

演算の分解 →

$$v(a,b) = a \times b$$
$$z(v) = v + 3$$
$$u(z,c) = c \times z$$
$$y(u,d) = u + d$$

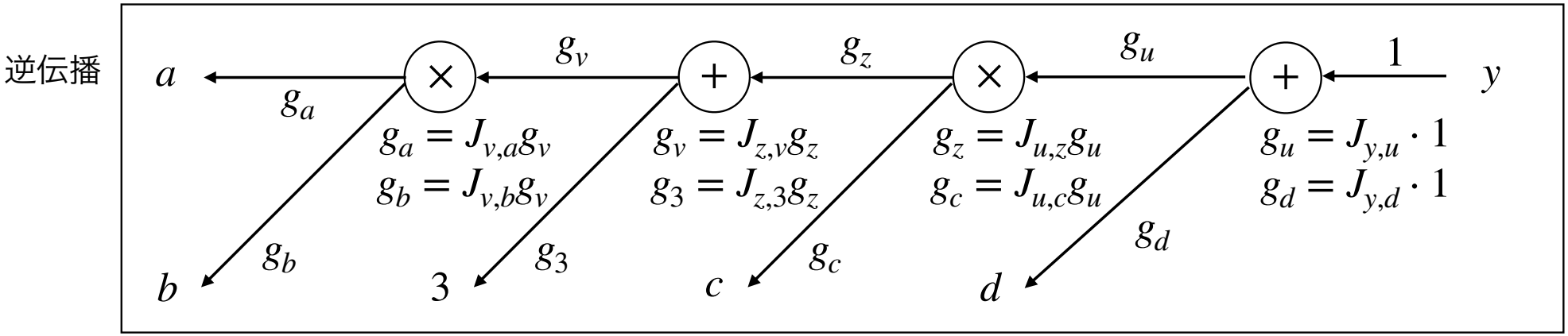
mul (a,b)	add $(v,3)$	mul (z,c)	add (u,d)
$J_{v,a} = b$	$J_{z,v} = 1$	$J_{u,z} = c$	$J_{y,u} = 1$
$J_{v,b} = a$	$J_{z,3} = 1$	$J_{u,c} = z$	$J_{y,d} = 1$

順伝播



計算グラフによる自動微分の表現（逆伝播）

mul (a, b)	add $(v,3)$	mul (z, c)	add (u, d)
$J_{v,a} = b$	$J_{z,v} = 1$	$J_{u,z} = c$	$J_{y,u} = 1$
$J_{v,b} = a$	$J_{z,3} = 1$	$J_{u,c} = z$	$J_{y,d} = 1$



例えば $g_b = J_{v,b} J_{z,v} J_{u,z} J_{y,u} \cdot 1 = a \cdot 1 \cdot c \cdot 1 \cdot 1 = ac$

オペレーション・オーバーロードによる自動微分の実装例（python）

「Variable」クラスの実装

```
class Variable:
    def __init__(self,value,parents=None):
        self.value   = value
        self.grad    = 0
        self.parents = parents or []

    def backward(self,grad=1):
        self.grad += grad
        for parent,lgrad in self.parents:
            parent.backward( lgrad * grad )

    def __add__(self,other):
        if type(other) != Variable:  other = Variable(other)
        value = self.value + other.value
        lgrad_byself  = 1
        lgrad_byother = 1
        parents = [(self,lgrad_byself),(other,lgrad_byother)]
        return Variable(value,parents)

    def __mul__(self,other):
        if type(other) != Variable:  other = Variable(other)
        value = self.value * other.value
        lgrad_byself  = other.value
        lgrad_byother = self.value
        parents = [(self,lgrad_byself),(other,lgrad_byother)]
        return Variable(value,parents)
```

$z^{(i)}$

$g_i = \partial z^{(N)} / \partial z^{(i)}$

再帰的処理

$g_{(i-1)} = J_i g_i$

J_i

J_i

__radd__()、__rmul__() なども定義しておく。

引き算、割り算も同様に定義する。

「Variable」クラスの使い方

```
a = Variable(3)
b = Variable(5)
c = Variable(7)
d = Variable(11)

z = a*b + 3
y = c*z + d

print(f"{b.value=}, {b.grad=}")
y.backward()
print(f"{b.value=}, {b.grad=}")
```

$z(a,b) = ab + 3$

$y(z,c,d) = cz + d$

出力結果

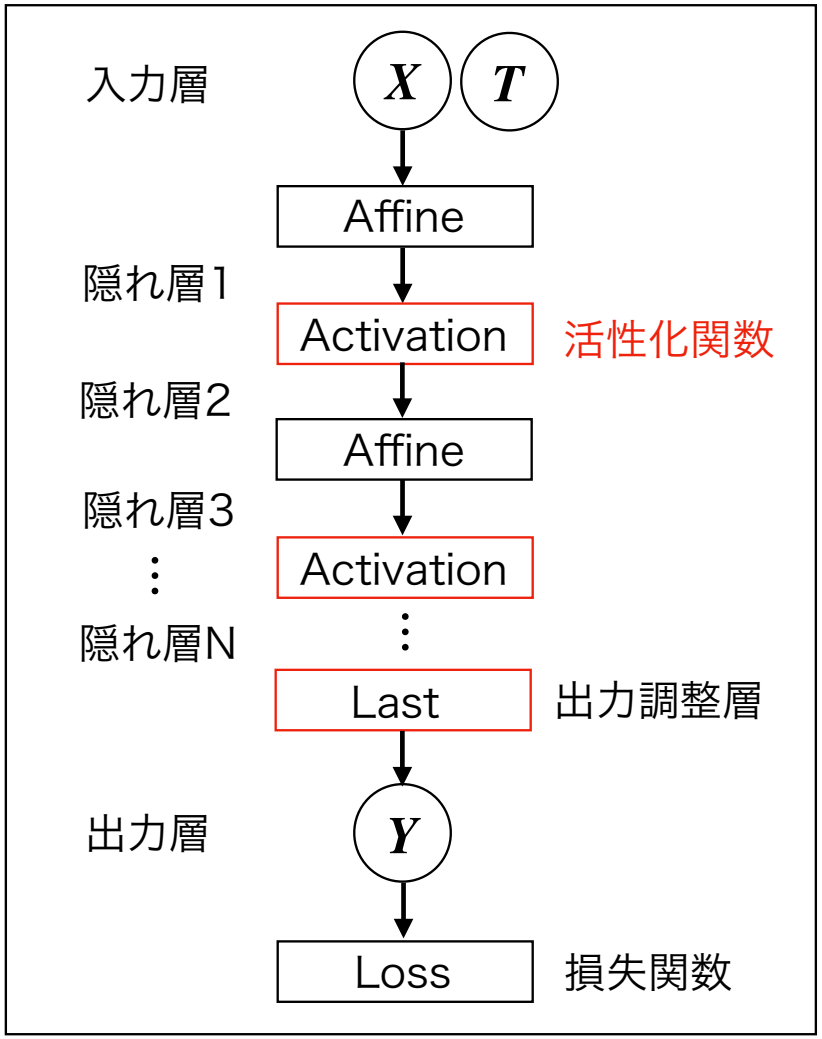
```
b.value=5, b.grad=0
b.value=5, b.grad=21
```

$g_b = \dfrac{\partial y}{\partial b} = \dfrac{\partial z}{\partial b}\dfrac{\partial y}{\partial z} = ac$

勾配: 変数の変化が最終出力値に及ぼす影響の大きさ。

（b を 1 大きくすると y の値は 21 大きくなるということ）

誤差逆伝播法（逆伝播公式の応用）



$$\boldsymbol{g}_l = \frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{J}_{l-1}^T \boldsymbol{g}_{l-1}$$

（活性化関数にはウェイトはない）

$$\boldsymbol{g}_N = \frac{\partial L}{\partial \boldsymbol{Z}^{(N)}} = \boldsymbol{J}_{\boldsymbol{Y}}^T \boldsymbol{g}_{\boldsymbol{Y}} = -(\boldsymbol{T} - \boldsymbol{Y})$$ 誤差（残差）

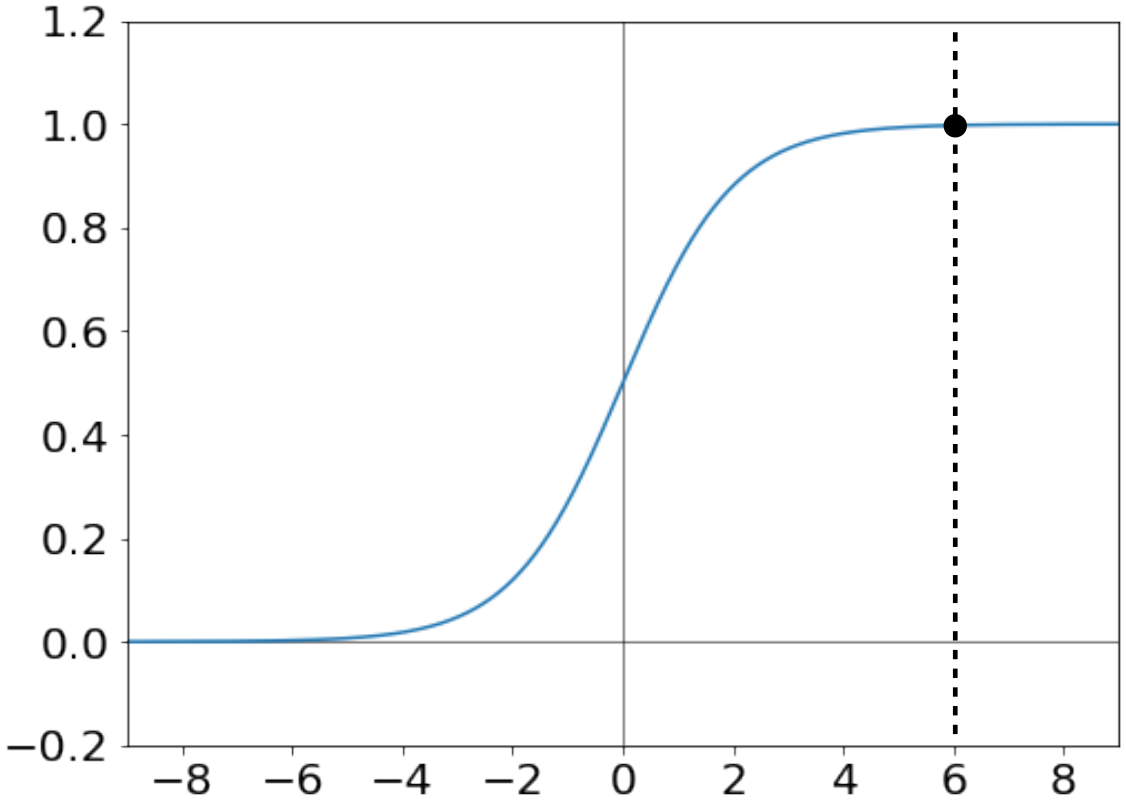
$$\boldsymbol{g}_{\boldsymbol{Y}} = \frac{\partial L}{\partial \boldsymbol{Y}} = \boldsymbol{J}_L^T \mathbf{1}$$

線形回帰やロジスティック回帰の場合、最終層の一つ前の変数に関する損失関数の勾配は誤差（残差）となる。

誤差を逆向きに伝播させて各ウェイトに関する勾配を求めていく様子から、誤差逆伝播法と呼ばれる。

勾配消失問題

$$a(z) = \mathrm{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$



$$z = xw$$

$$a(z = 6) = 0.9975$$

$$J_a = \frac{\partial a}{\partial z} = a(1 - a) = 0.0025$$

$$g_w = \frac{\partial L}{\partial w} = J_a g_a$$

活性化関数の手前の層の出力 z（の絶対値）が大きい時、局所微分 J_a はとても小さくなる。多層ネットワークの誤差逆伝播において、そのような箇所があれば、それより上層に伝播される勾配は全てとても小さなものとなり、ウェイトが更新されなくなる。これは逆伝播の問題ではなく、活性化関数の形の問題である。

非飽和系活性化関数

非飽和な活性化関数を使うことで、
勾配消失問題を回避できる。
これにより多層化が一気に進んだ。

$$\mathrm{softplus}(x) = \ln(1 + e^{x})$$

$$\mathrm{relu}(x) = \max(0,\, x)$$

ResNet
AlphaGo

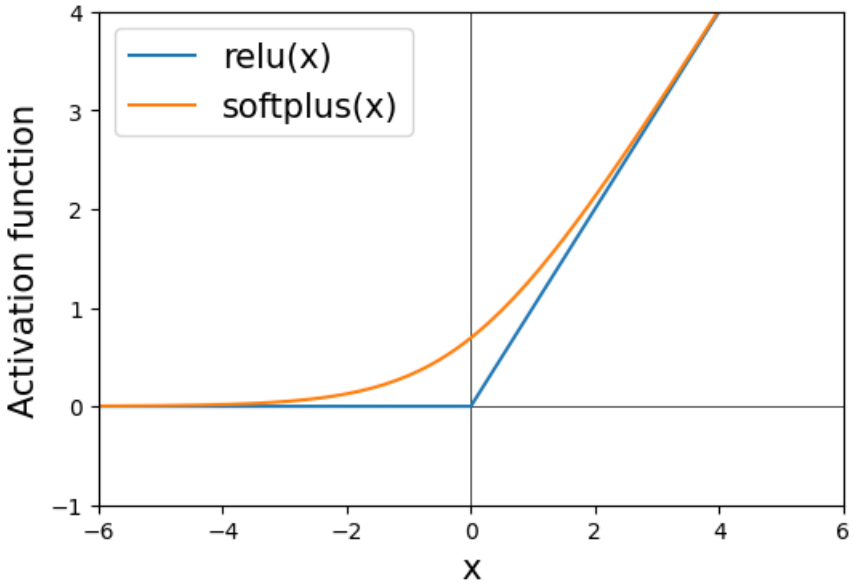
$$\mathrm{gelu}(x) = \frac{x}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

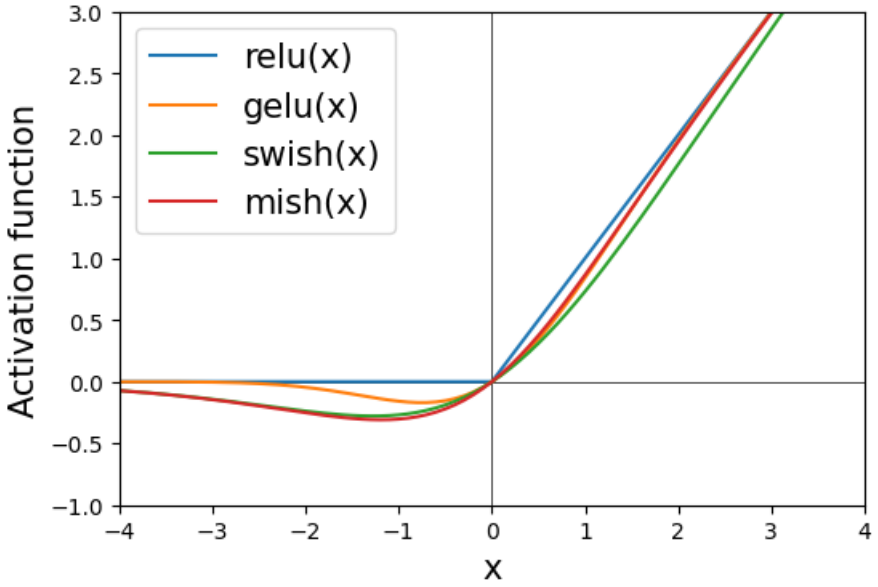
GPT
BERT

$$\mathrm{swish}(x) = x \cdot \mathrm{sigmoid}(\beta x)$$

2018 SOTA

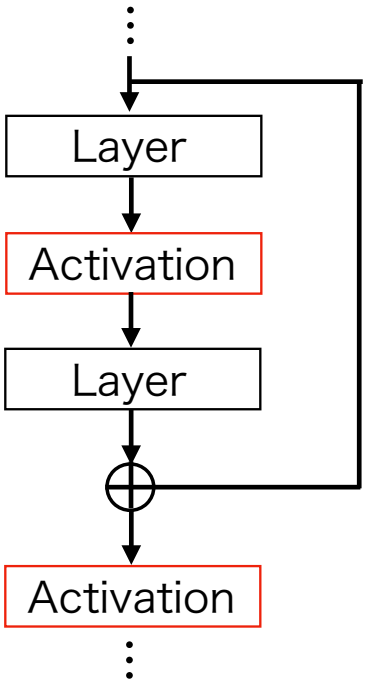
$$\mathrm{mish}(x) = x \cdot \tanh\left[\mathrm{softplus}(x)\right]$$





残差ネットワーク（スキップ接続） 2015 ResNet (Microsoft)

深層化による劣化問題を解決



劣化問題:
数層で十分な学習が可能な問題である場合に、余分な層を追加すると、かえって性能が悪くなってしまう問題。これはディープラーニングが恒等変換（インプットに何も手を加えず出力すること）を学習することが苦手なことに起因する。

スキップ接続により、何もしないことが最善である場合にはウェイトをゼロにすれば良いことになり、学習が簡単になる。

また、勾配消失問題も軽減することが期待される。

畳み込みニューラルネットワーク（CNN）

CNNの歴史

1998 LeNet（Sigmoid）

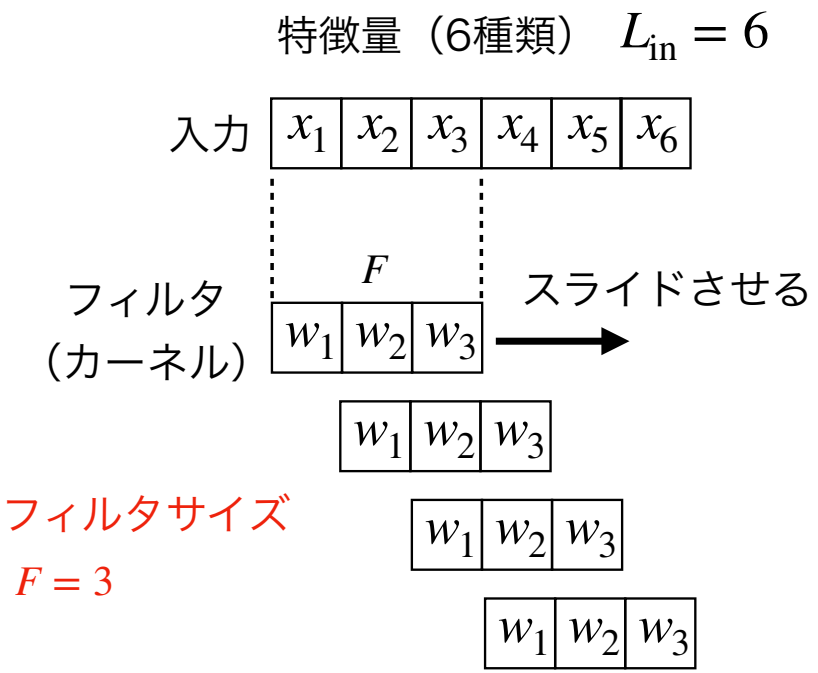
2012 AlexNet（ReLU、ドロップアウト）

2014 VGG（小さなフィルタによる連続畳み込みとプーリング）

2015 GoogLeNet（インセプション構造）

2015 ResNet（スキップ接続）← 画像認識で人間を超える

1次元畳み込み演算



$L_{\text{out}} = 4$

$$L_{\text{out}} = L_{\text{in}} - F + 1$$

出力 z_1 z_2 z_3 z_4

$$z_1 = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$z_2 = x_2 w_1 + x_3 w_2 + x_4 w_3$$

$$\ddots$$

$$z_l = \sum_{j=1}^{F} x_{l+j-1} w_j$$

近くの連続した特徴量だけの内積をとる（入力の粗視化の方法を学習することに相当）。

特徴量の並び順に意味がある場合、その情報をある程度保持することができる。

1次元畳み込み演算の実装（Affine 化）

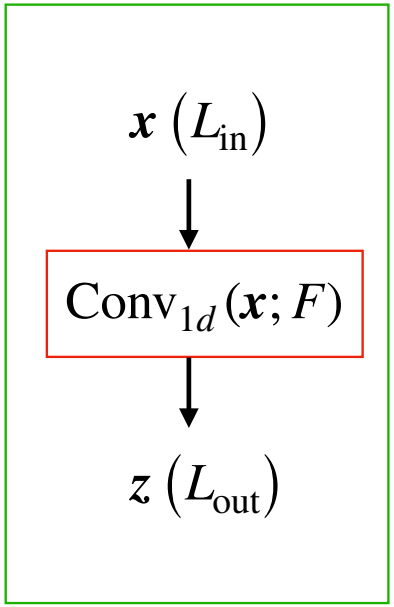
入力

$$\underset{L_{\text{in}}}{\boldsymbol{x}} = [x_1, x_2, x_3, x_4, x_5, x_6]$$

Expand

$$\boldsymbol{X}_{\text{expand}} = \underset{(L_{\text{out}}, F)}{\begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \\ x_4 & x_5 & x_6 \end{bmatrix}}$$

余分な計算が必要で、使用メモリも増えるが、畳み込み演算が単なる Affine 変換となるため、全体としてはとても高速になる。



$$L_{\text{out}} = L_{\text{in}} - F + 1$$

$$\underset{\text{Affine}}{\boldsymbol{X}_{\text{expand}}\boldsymbol{w}} = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \\ x_4 & x_5 & x_6 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \xrightarrow{\text{Transpose}} \underset{L_{\text{out}}}{\boldsymbol{z} = [z_1, z_2, z_3, z_4]}$$

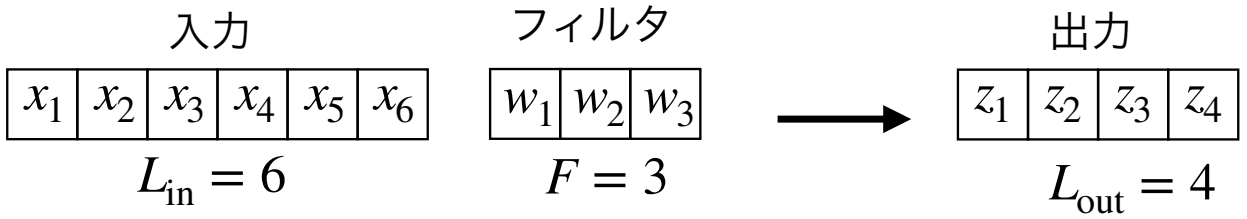
畳み込み層と全結合層の違い

畳み込み層（Convolution）　　ウェイト数が入力サイズに依存しない

・ウェイト数（フィルタサイズ）を自由に設定できる。

・出力サイズは公式を使って求める必要がある。

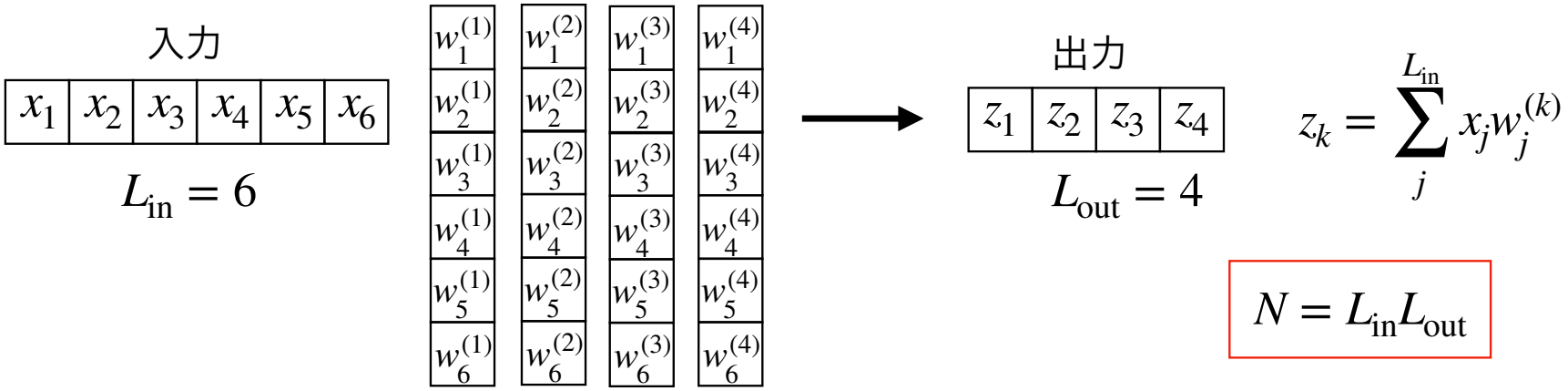
$$L_{\text{out}} = L_{\text{in}} - F + 1$$



全結合層（Dense, Affine）

・出力サイズを設定する。

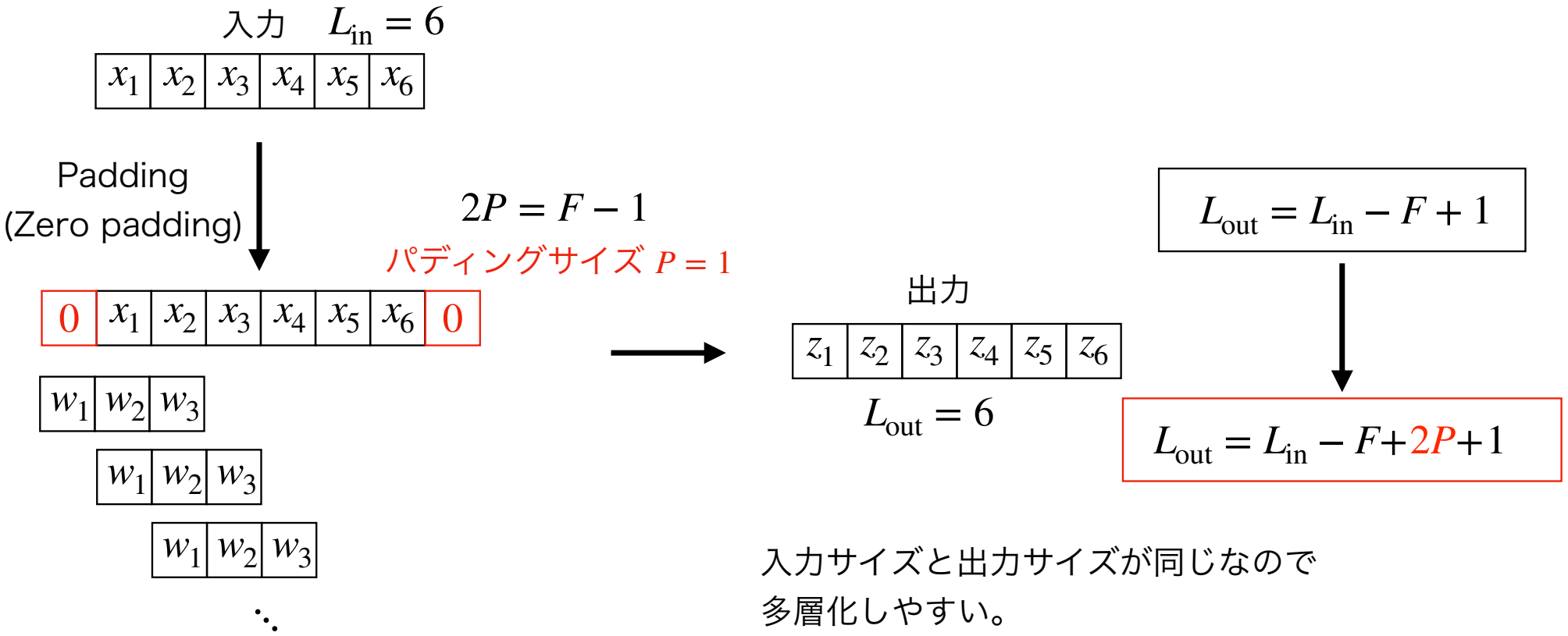
・ウェイトの数は $N = L_{\text{in}} L_{\text{out}}$ となる。



$$N = L_{\text{in}} L_{\text{out}}$$

パディング

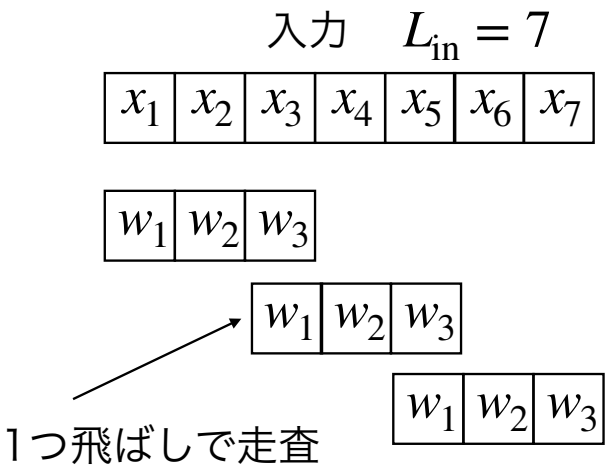
畳み込み層で出力サイズがだんだん小さくなってしまう問題を解決



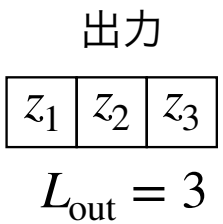
入力サイズと出力サイズが同じなので
多層化しやすい。

ストライド

入力を粗く走査したい時はストライド（歩幅）を大きくする。
（ストライドを大きくすると出力サイズは小さくなる点に注意）



ストライド $S = 2$



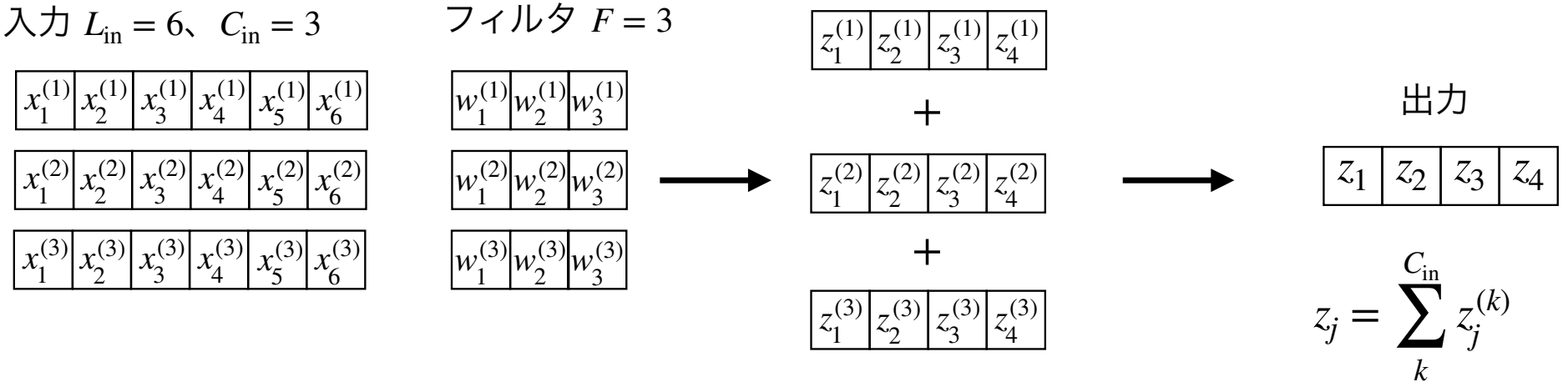
$$L_{\text{out}} = L_{\text{in}} - F + 2P + 1$$

$$L_{\text{out}} = \left(L_{\text{in}} - F + 2P\right) // S + 1$$

$$\begin{aligned} L_{\text{out}} &= \left(L_{\text{in}} - F + 2P\right) // S + 1 \\ &= (7 - 3 + 2 \cdot 0) // 2 + 1 \\ &= 4 // 2 + 1 \\ &= 3 \end{aligned}$$

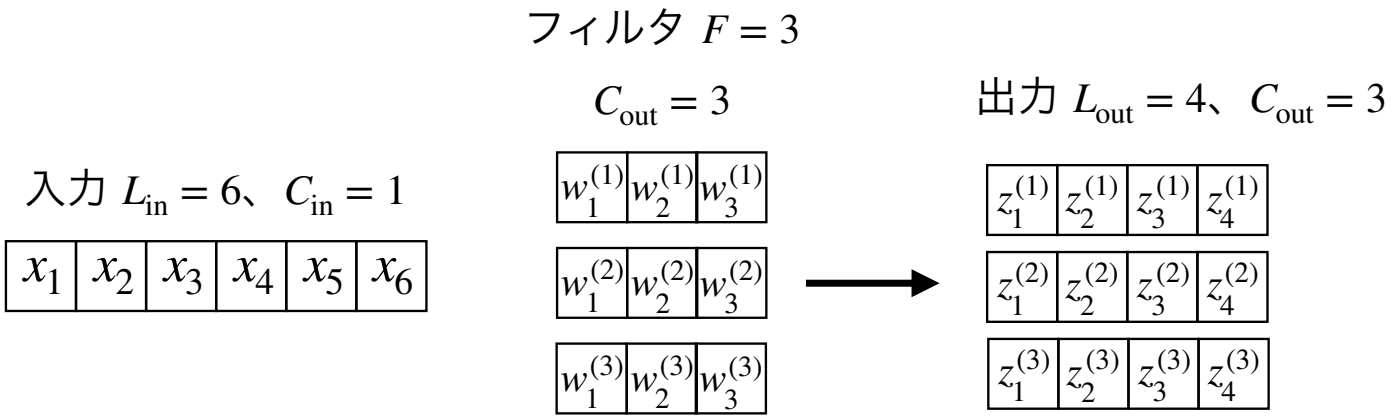
多チャンネル**入力**の場合の畳み込み演算

フィルタを入力チャンネルの数だけ用意し、それぞれ独立に畳み込み演算を行い、
結果を**入力チャンネル方向に足し合わせる。**

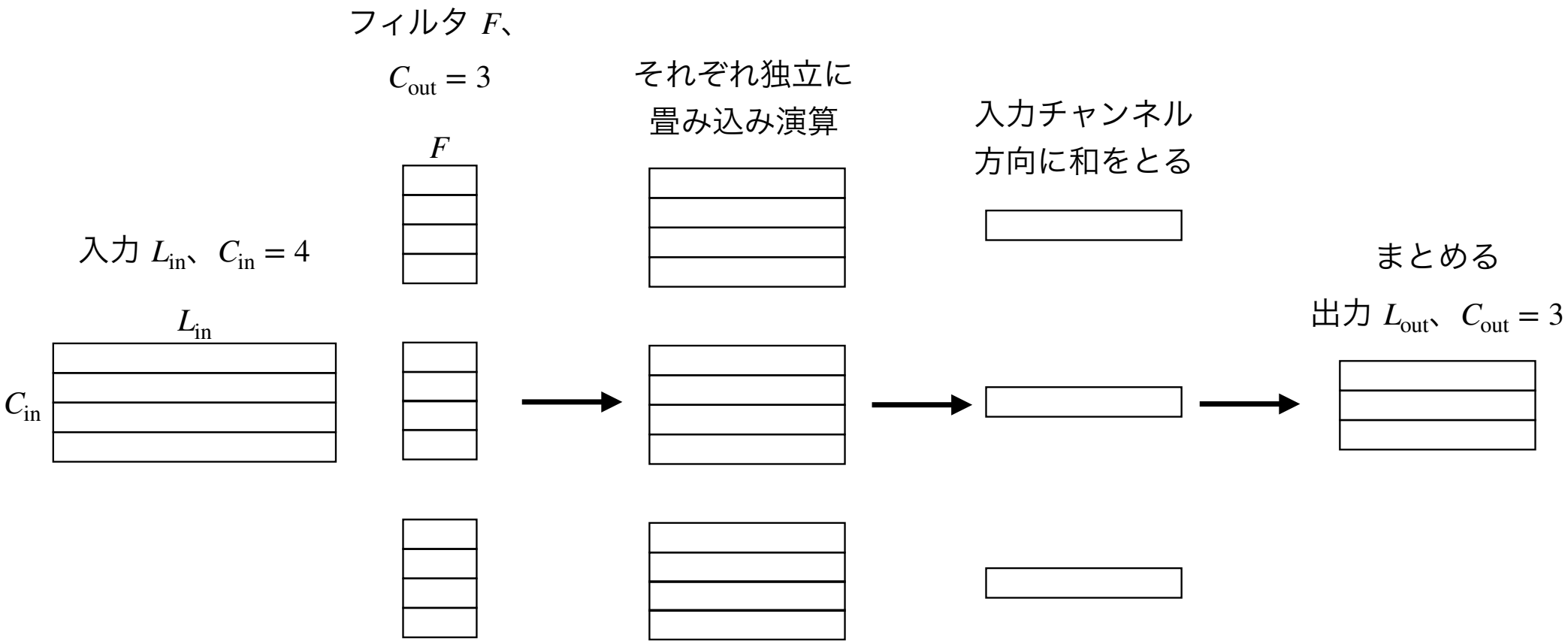


多チャンネル**出力**の場合の畳み込み演算

出力を多チャンネルにしたい場合は、フィルタを出力チャンネルの数だけ用意し、
それぞれ独立に畳み込み演算を行うだけで良い。

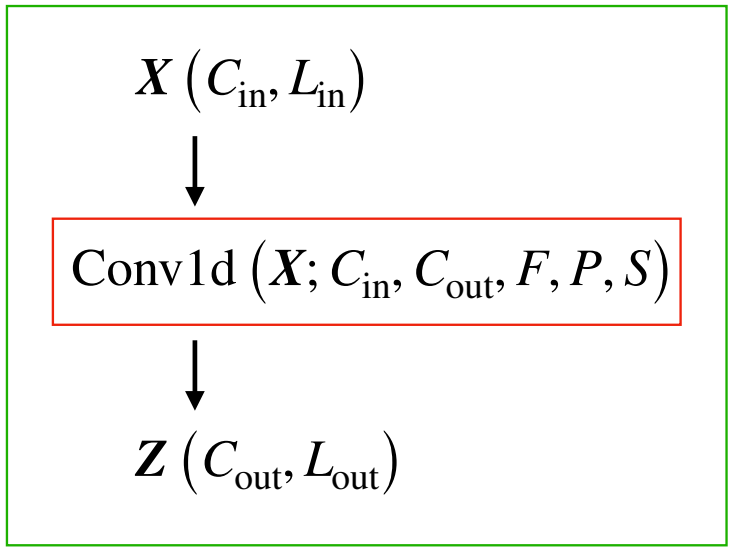


1次元畳み込み演算のまとめ



$$L_{\text{out}} = \left(L_{\text{in}} - F + 2P\right) // S + 1$$

1次元畳み込み演算のまとめ



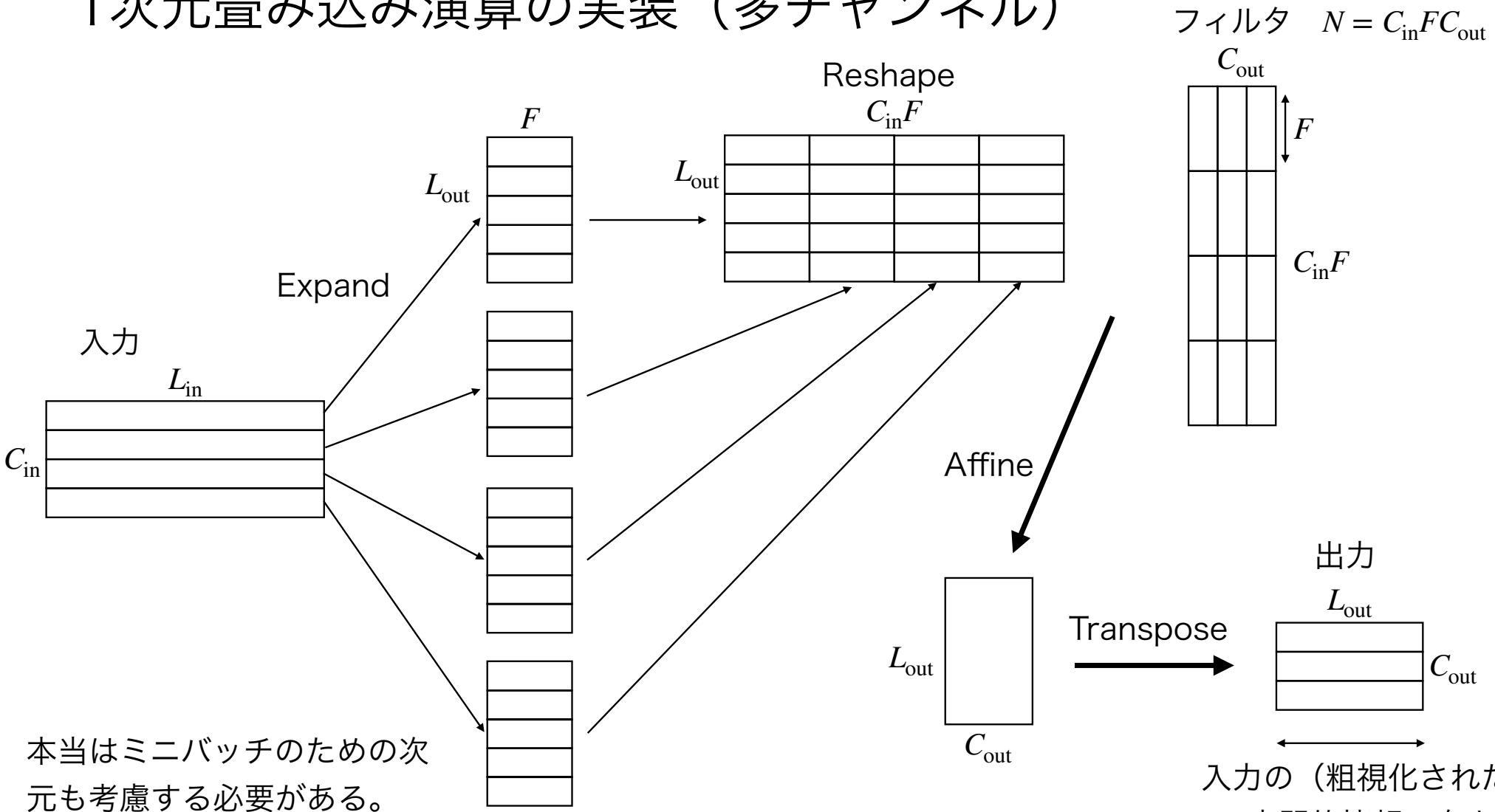
ウェイト数

$$N = C_{\text{in}} F C_{\text{out}}$$

入力サイズ数 L_{in} に依存しない

$$L_{\text{out}} = \left(L_{\text{in}} - F + 2P\right) // S + 1$$

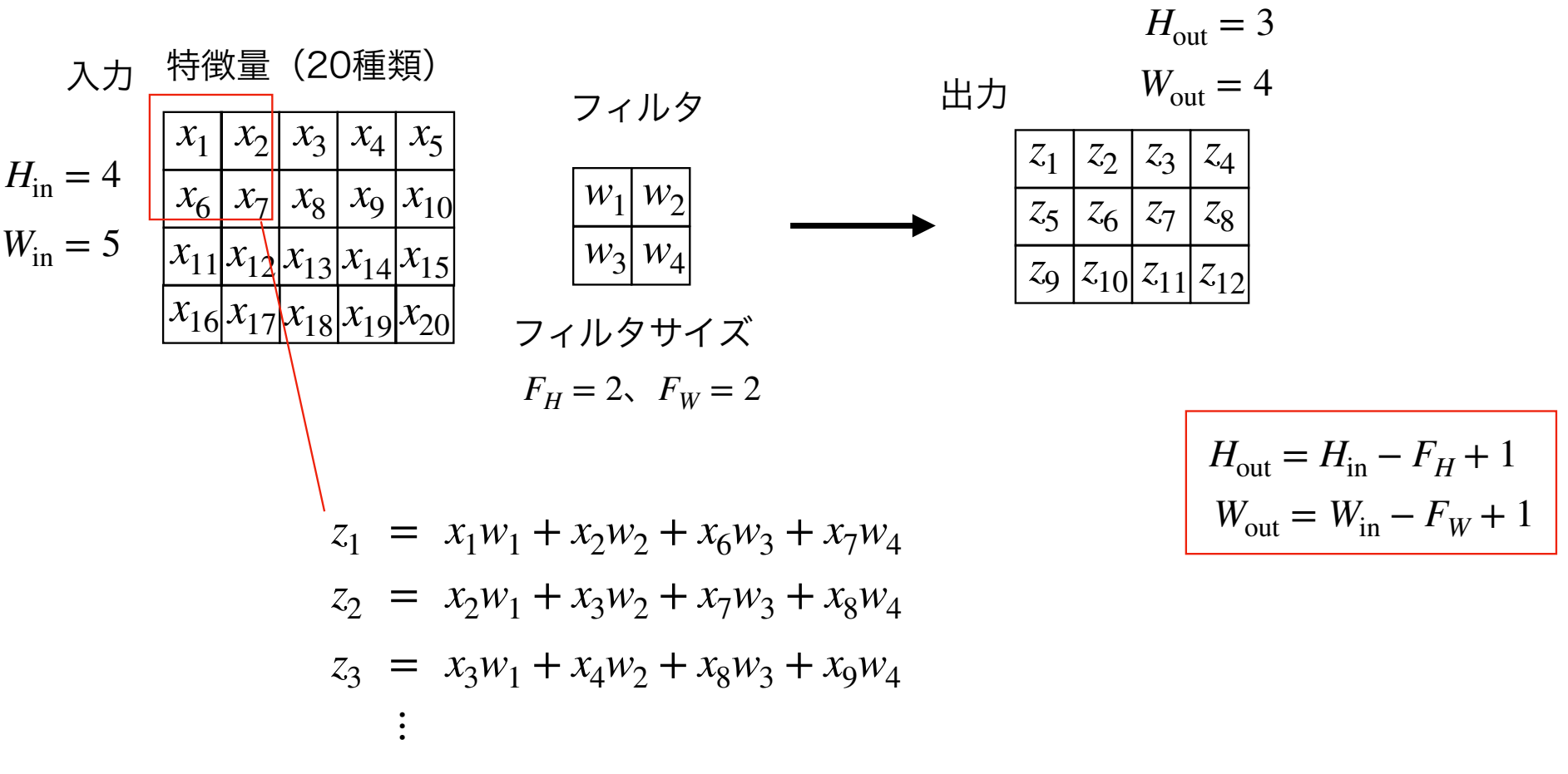
1次元畳み込み演算の実装（多チャンネル）



Expandの実装例

```
def expand1d(z,fl,pd,st):
    c,l,n = z.shape
    lo = (l + 2*pd - fl)//st + 1
    z_pad = np.pad(z,[(0,0),(pd, pd),(0,0)])
    z_exd = np.zeros([c,fl,lo,n])
    for i in range(fl):
        z_exd[:,i,:,:] = z_pad[:,i:i+st*lo:st,:]
    return z_exd,lo
```

2次元畳み込み演算



$$
\begin{aligned}
z_1 &= x_1 w_1 + x_2 w_2 + x_6 w_3 + x_7 w_4 \\
z_2 &= x_2 w_1 + x_3 w_2 + x_7 w_3 + x_8 w_4 \\
z_3 &= x_3 w_1 + x_4 w_2 + x_8 w_3 + x_9 w_4 \\
&\vdots
\end{aligned}
$$

$$
\begin{aligned}
H_{\text{out}} &= H_{\text{in}} - F_H + 1 \\
W_{\text{out}} &= W_{\text{in}} - F_W + 1
\end{aligned}
$$

2次元畳み込み演算の実装

入力

$$\boldsymbol{X}\left(H_{\text{in}}, W_{\text{in}}\right) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 & x_9 & x_{10} \\ x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{16} & x_{17} & x_{18} & x_{19} & x_{20} \end{bmatrix}$$

Expand

$$\boldsymbol{X}_{\text{expand}}\left(H_{\text{out}}W_{\text{out}}, F_H F_W\right) = \begin{bmatrix} x_1 & x_2 & x_6 & x_7 \\ x_2 & x_3 & x_7 & x_8 \\ x_3 & x_4 & x_8 & x_9 \\ x_4 & x_5 & x_9 & x_{10} \\ x_6 & x_7 & x_{11} & x_{12} \\ x_7 & x_8 & x_{12} & x_{13} \\ x_8 & x_9 & x_{13} & x_{14} \\ x_9 & x_{10} & x_{14} & x_{15} \\ x_{11} & x_{12} & x_{16} & x_{17} \\ x_{12} & x_{13} & x_{17} & x_{18} \\ x_{13} & x_{14} & x_{18} & x_{19} \\ x_{14} & x_{15} & x_{19} & x_{20} \end{bmatrix}$$

$$\boldsymbol{W} = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \xrightarrow{\text{Reshape}} \boldsymbol{W} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

Affine

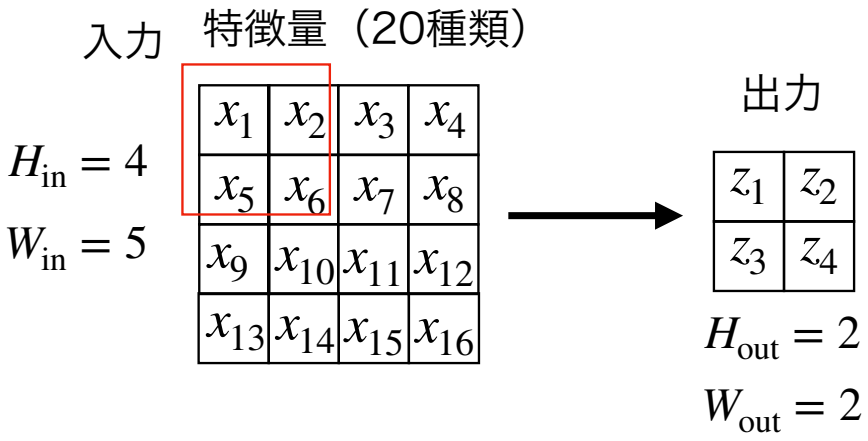
$$\boldsymbol{X}_{\text{expand}}\boldsymbol{W} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{12} \end{bmatrix} \xrightarrow{\text{Reshape}} \boldsymbol{Z} = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 \\ z_5 & z_6 & z_7 & z_8 \\ z_9 & z_{10} & z_{11} & z_{12} \end{bmatrix}$$

（チャンネル、ミニバッチの次元を合わせて入力は一般に4次元テンソルで、展開する際には6次元テンソルとなる）

$$H_{\text{out}} = H_{\text{in}} - F_H + 1$$
$$W_{\text{out}} = W_{\text{in}} - F_W + 1$$

プーリング層

冗長な特徴量の除去、粗視化、ノイズ除去、
入力のズレ（平行移動）の吸収



プーリングサイズ

$F_H = 2$、$F_W = 2$

ストライドと同じにする

MaxPooling:　データにメリハリが出る

$$z_1 = \text{Max}\left(x_1, x_2, x_5, x_6\right)$$

AveragePooling:　ノイズが除去される

$$z_1 = \frac{1}{F_{\text{H}} F_{\text{W}}}\left(x_1 + x_2 + x_5 + x_6\right)$$

バッチ正則化層

Batch Normalizarion 層

$$X(N, L, C)$$

$$\mu_c = \frac{1}{NL} \sum_{i=1}^{N} \sum_{j=0}^{L} x_{i,j,c}$$

$$\sigma_c^2 = \frac{1}{NL} \sum_{i=1}^{N} \sum_{j=0}^{L} \left(x_{i,j,c} - \mu_c \right)^2$$

$$\hat{x}_{i,j,c} \leftarrow \frac{x_{i,j,c} - \mu_c}{\sqrt{\sigma_c^2 + \varepsilon}}$$

$$y_{i,j,c} \leftarrow \gamma_c \hat{x}_{i,j,c} + \beta_c$$

Trainable parameters ($2C$) : γ_c, β_c

Non-trainable parameters ($2C$) : μ_c, σ_c^2

2024/07/11 問題

文章に最も関連のある単語を、後に続くカッコ内から**2つ**ずつ選べ。

問1.
データからルールを自動的に抽出するアルゴリズム、あるいは、そのようなアルゴリズムを研究する分野。
（a. エキスパートシステム / b. 機械学習 / c. 人工知能 / d. 推論エンジン / e. ルールベースAI）

問2.
主成分分析のように、特徴量データのより良い表現方法を学習。
（a. 強化学習 / b. 教師なし学習 / c. 転移学習 / d. 基底変換 / e. クラスタリング）

問3.
教師データとは特徴量と何のセットのことか。
（a. 目的変数 / b. 記述子 / c. 属性値 / d. 説明変数 / e. 正解の値）

問4.
モデルパラメータの内、学習によって直接値が更新されないパラメータ。
（a. ウェイト / b. ハイパーパラメータ / c. 回帰係数 / d. 学習率 / e. バイアス）

問5.
ロジスティック回帰の特徴。
（a. 回帰問題 / b. 分類問題 / c. 最適化問題 / d. 線形モデル / e. 非線形モデル）

2024/07/11 問題

文章に最も関連のある単語を、後に続くカッコ内から**2つ**ずつ選べ。

問6.
学習の際、計算効率を上げるために教師データを分割して学習する方法。
（a. ミニバッチ学習 / b. 学習率減衰 / c. K分割クロスバリデーション / d. モーメンタム / e. 確率的勾配降下法）

問7.
モデルの性能の指標となるだけでなく、学習の際の目標ともなる値を与える関数。
（a. 活性化関数 / b. 損失関数 / c. 決定係数 / d. 相関関数 / e. コスト関数）

問8.
目的変数とモデルの出力との差。
（a. 勾配 / b. 予測残差 / c. 標準誤差 / d. 近似誤差 / e. 残差）

問9.
多クラスロジスティック回帰のモデルの形とよく使われる損失関数。
（a. ソフトマックス関数 / b. 交差エントロピー誤差 / c. シグモイド関数 / d. 平均2乗誤差 / e. 活性化関数）

問10.
カテゴリ変数を機械学習で扱える数値に変換すること。
（a. 埋め込み / b. ラベルエンコーディング / c. デコーディング / d. ワンホットエンコーディング / e. オートエンコーダ）

2024/07/18 問題

文章に最も関連のある単語を、後に続くカッコ内から**1つ**選べ。

問1.
ディープラーニングの多層化に関する技術。
（a. 畳み込み / b. ドロップアウト / c. ReLU）

問2.
ディープラーニングには必ずあるが線形回帰にはないもの。
（a. Affine層 / b. Softmax関数 / c. 活性化関数）

問3.
多層化・深層化のメリット。
（a. 高度な特徴量の獲得 / b. 過学習の抑制 / c. 劣化問題の緩和）

問4.
誤差逆伝播法で使われる技術。
（a. 形式微分 / b. 数値微分 / c. 自動微分）

問5.
ウェイトを持つ層。
（a. プーリング層 / b. 畳み込み層 / c. 出力調整層）

2024/07/18 問題

文章に最も関連のある単語を、後に続くカッコ内から**1つ**選べ。

問6.
入力サイズを大きくすると畳み込み層のウェイトの数はどうなるか。
（a. 少なくなる / b. 多くなる / c. 変わらない）

問7.
フィルタサイズを増やすと畳み込み層のウェイトの数はどうなるか。
（a. 少なくなる / b. 多くなる / c. 変わらない）

問8.
パディングサイズを大きくすると畳み込み層の出力サイズはどうなるか。
（a. 小さくなる / b. 大きくなる / c. 変わらない）

問9.
ストライドを大きくすると畳み込み層の出力サイズはどうなるか。
（a. 小さくなる / b. 大きくなる / c. 変わらない）

問10.
プーリングサイズを大きくすると畳み込み層の出力サイズはどうなるか。
（a. 小さくなる / b. 大きくなる / c. 変わらない）