深層学習 Part 2

東京海洋大学
竹縄知之

目次

1　ニューラルネットワークの順伝播

本節では，ニューラルネットワークは入力データ $\mathbf{x} = [x_0, x_1, \cdots, x_{D-1}]$ に対して，いくつかの層での計算を経由して予測値 $\hat{y}$ を出力するネットワークモデルであること，各層での計算は基本的には線形変換（正確にはアファイン変換）と活性化関数によって行われることを学びます．

1.1　アファイン層の順伝播

ニューラルネットワークの各層は複数のニューロン（グラフの頂点）からなり，第 i 層の l 番目のニューロンにおける演算は，線形変換（正確には定数を加える操作もあるのでアファイン変換）してから非線形変換を施したもの：

$$\left\{\begin{array}{rcl} a_l^{(i)} & = & x_0^{(i-1)} w_{0l}^{(i)} + x_1^{(i-1)} w_{1l}^{(i)} + \cdots + x_{K-1}^{(i-1)} w_{K-1,l}^{(i)} + b_l^{(i)} \\ x_l^{(i)} & = & f\left(a_l^{(i)}\right) \end{array}\right.$$

になります．ここで，$\mathbf{x}^{(i-1)} = (x_0^{(i-1)}, \cdots, x_{K-1}^{(i-1)})$ は前の層の出力値，$w_{kl}^{(i)}$ および $b_l^{(i)}$ はモデルの重みパラメータと呼ばれる値．また，$f(a)$ は活性化関数と呼ばれる非線形関数です．

上式は (K, L) 次元行列 $W^{(i)} = \left(w_{kl}^{(i)}\right)$ および L 次元ベクトル $\mathbf{b}^{(i)} = \left(b_l^{(i)}\right)$ を用いると

$$\left\{\begin{array}{rcl} \mathbf{a}^{(i)} & = & \mathbf{x}^{(i-1)} W^{(i)} + \mathbf{b}^{(i)} \\ \mathbf{x}^{(\mathbf{i})} & = & f\left(\mathbf{a}^{(i)}\right) \end{array}\right.$$

と書けます．ただし，ベクトルは横ベクトルであり，関数 f は $\mathbf{a}^{(i)}$ の各成分に作用するものとします（このような作用のさせ方をブロードキャストといいます）．

ミニバッチの場合

N 個のデータによるミニバッチ学習の場合について考えましょう．前の層の出力値の集合を

(N, K) 型配列 $X^{(i-1)} = \begin{bmatrix} x_{0,0}^{(i-1)} & \cdots & x_{0,K-1}^{(i-1)} \\ \vdots & \ldots & \vdots \\ x_{N-1,0}^{(i-1)} & \cdots & x_{N-1,K-1}^{(i-1)} \end{bmatrix}$ とするとき，アファイン層の順伝播は

$$\begin{cases} A^{(i)} &= X^{(i-1)}W^{(i)} + \mathbf{b}^{(i)} \\ X^{(i)} &= f\left(A^{(i)}\right) \end{cases}$$

と書けます．ただし，

$W^{(i)}$：(K, L) 型配列，$\mathbf{b}^{(i)}$：$(L,)$ 型配列

であることは変わりません．このとき，$A^{(i)}$ や $X^{(i)}$ は (N, L) 型配列になります．

NumPy では

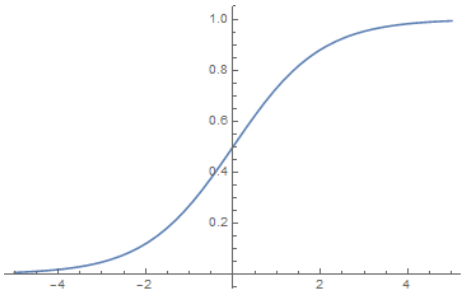
$$(N, L) \text{ 型配列} + (L,) \text{ 型配列} = (N, L) \text{ 型配列}$$

とブロードキャストされる（第 1 項のすべての行に第 2 項を行ベクトルと見たものが加えられる）ことに注意しましょう．また，活性化関数は成分ごとに作用します．

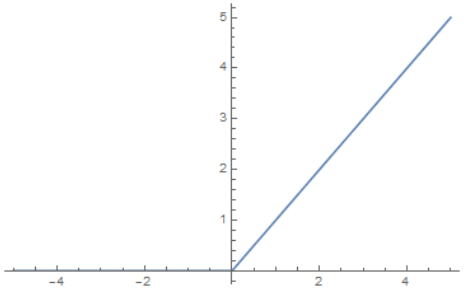
1.2 活性化関数

活性化関数としては，以下の 2 つが典型的です．

シグモイド関数 $\sigma(x) = \dfrac{1}{1+e^{-x}}$



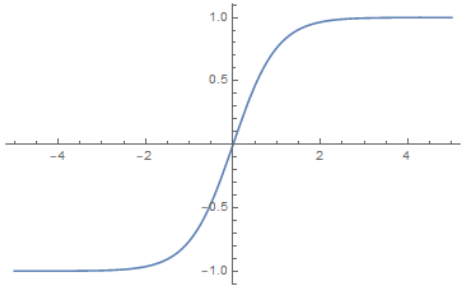
ReLU **関数** $\mathrm{ReLU}(x) = \max\{0, x\}$



その他に以下の関数なども状況に応じて用いられます．

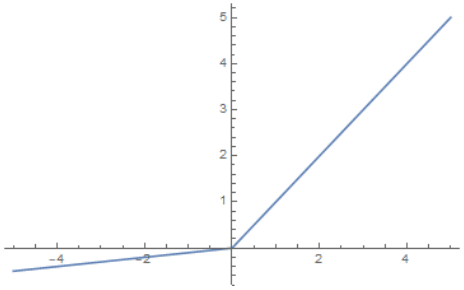
ハイパボリックタンジェント

$\tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$



Leaky ReLU **関数**

$$f(x) = \begin{cases} x & (x > 0) \\ 0.1x & (x \leq 0) \end{cases}$$



2　損失関数と出力層の設計

　損失関数あるいはコスト関数とは機械学習のモデルの「悪さ」を表す指標です．損失関数は基本的には予測の精度を測るための誤差関数と同じものですが，ときにはパラメータの大きさなどをペナルティとして損失関数に取り込むこともあります．しかし，しばらくは簡単のため，損失関数としては誤差関数のみを考えることにします．

　ニューラルネットワークを学習させるためには，誤差関数を適切に設計する必要があります．まず，当然ですが誤差関数は予測の精度が低いときは大きく，高いときは小さくなければなりません．また，学習が可能になるためには，モデルのパラメータで微分できなければなりません．教師あり学習の場合，誤差関数は正解値 y と予測値 $\hat{y}$ の関数となります．

　本節では回帰問題及び分類問題の教師あり学習における典型的な出力層および損失関数（誤差関数）を紹介します．

2.1　回帰問題のとき

出力値を $\hat{y}$，正解ラベルを y とするとき，2 乗誤差（の $\frac{1}{2}$）

$$\frac{1}{2}(\hat{y}-y)^2$$

を誤差関数とするのが一般的です．出力層では活性化関数を用いません．

N 個のデータによるミニバッチ学習の場合は，出力値の集合を $\hat{Y}=[\hat{y}^{(0)},\cdots,\hat{y}^{(N-1)}]$，正解ラベルの集合を $Y=[y^{(0)},\cdots,y^{(N-1)}]$ とするとき，平均 2 乗誤差（の $\frac{1}{2}$）

$$\frac{1}{2N}\sum_{n=0}^{N-1}\left(\hat{y}^{(n)}-y^{(n)}\right)^2$$

を用います．

2.2　0-1 の 2 値分類問題のとき

正解値が 0 または 1 をとる 2 値問題のときは，出力層の活性化関数としてシグモイド関数 $\hat{y} = \sigma(x) = \dfrac{1}{1+e^{-x}}$ を施して，$P(y=1) = \hat{y}$，$P(y=0) = 1-\hat{y}$ とすることで，ベルヌーイ分布（1 が出る確率 p，0 が出る確率 $1-p$ とする確率分布 ）を出力しているとみなせます．

この出力値 $\hat{y}$ の正解ラベル y に対する交差エントロピー

$$H(y, \hat{y}) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

を誤差関数とするのが一般的です．

N 個のデータによるミニバッチ学習の場合は，出力値の集合を $\hat{Y} = [\hat{y}^{(0)}, \cdots, \hat{y}^{(N-1)}]$，正解ラベルの集合を $Y = [y^{(0)}, \cdots, y^{(N-1)}]$ とするときの平均交差エントロピー

$$H(Y, \hat{Y}) = -\frac{1}{N} \sum_{n=0}^{N-1} \left(y^{(n)} \log \hat{y}^{(n)} + (1-y^{(n)}) \log(1-\hat{y}^{(n)}) \right)$$

を誤差関数とします．

2.3 K 種類に分類するとき

出力層の活性化関数としてソフトマックス関数

$$\begin{bmatrix}\hat{y}_0 & \cdots & \hat{y}_{K-1}\end{bmatrix} = \mathrm{softmax}\left(\begin{bmatrix}x_0 & \cdots & x_{K-1}\end{bmatrix}\right) = \frac{1}{e^{x_0}+\cdots+e^{x_{K-1}}}\begin{bmatrix}e^{x_0} & \cdots & e^{x_{K-1}}\end{bmatrix}$$

を施して，各成分が 0 以上で，合計が 1 になるようにすると，K 個の根元事象をもつカテゴリカル分布（排反な K 個の事象に対して，各事象が出る確率 $p_0, \cdots, p_{K-1}$ が与えられている分布）を出力しているとみなせます．

この出力値 $\hat{\mathbf{y}} = \begin{bmatrix}\hat{y}_0 & \cdots & \hat{y}_{K-1}\end{bmatrix}$ の正解ラベル $\mathbf{y} = \begin{bmatrix}y_0 & \cdots & y_{K-1}\end{bmatrix}$（ワンホット表現，すなわち正解が l のとき，$y_l = 1$，$y_k = 0$（$k \neq l$））に対する交差エントロピー

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=0}^{K-1} y_k \log \hat{y}_k$$

を誤差関数とするのが一般的です．N 個のデータによるミニバッチ学習の場合は，平均交差エントロピー

$$H(Y, \hat{Y}) = -\frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1} y_k^{(n)} \log \hat{y}_k^{(n)}$$

を誤差関数とします．

交差エントロピーについての復習

情報量

事象 A の起こる確率を $P(A)$ とします．ある施行において事象 A が起こったときの情報量を

$$I(A) = -\log P(A)$$

と定めます．

交差エントロピー

全事象が排反な K 個の事象 $A_0, A_1, \cdots, A_{K-1}$ からなり，2 つの確率分布 $P(A_k)$ と $Q(A_k)$ （$k = 0, 1, \cdots, K-1$）が与えられているとします．このとき，

$$H(P, Q) = -\sum_{k=0}^{K-1} P(A_k) \log Q(A_k)$$

を（P に対する Q の）交差エントロピー）といいます．交差エントロピーは確率分布 P に関する確率分布 Q についての情報量の期待値ということができます．

3　誤差逆伝播法

　ニューラルネットワークの（例えば確率的勾配降下法による）学習のためには誤差関数のパラメータに関する微分を計算する必要があります．微分の計算は一つのパラメータについてだけなら，数値微分（パラメータを少しずらして誤差がどう変化するか観察する）を用いれば簡単にできるのですが，この方法だとすべての変数についてそれぞれ計算しなければいけないので大変時間がかかります．ニューラルネットワークでは，微分の連鎖律（チェインルール）を利用して効率よく行います．この計算はネットワークを誤差から逆に辿っていくので，誤差逆伝播法と呼ばれます．

3.1　確率的勾配降下法と誤差の微分

パラメータ θ を持つニューラルネットワーク $\hat{\mathbf{y}} = f(\mathbf{x}, \theta)$ の誤差を $E(\mathbf{x}, \theta)$ と書くとき，確率的勾配降下法による θ の更新は

$$\theta \leftarrow \theta - \lambda \frac{\partial E}{\partial \theta}$$

（$\lambda > 0$ は学習係数とよばれるハイパーパラメータ），ただし，

$$\frac{\partial E}{\partial \theta} = \frac{1}{N} \sum_{n=0}^{N-1} \frac{\partial E(\mathbf{x}^{(n)}, \theta)}{\partial \theta}$$

（$\mathbf{x}^{(0)}, \cdots, \mathbf{x}^{(N-1)}$ はミニバッチ内の入力データ）で与えられるので，パラメータに関する誤差の偏微分を計算する必要があります．連鎖律を用いてこれを効率よく計算する方法が誤差逆伝播法です．

3.2　微分の連鎖律

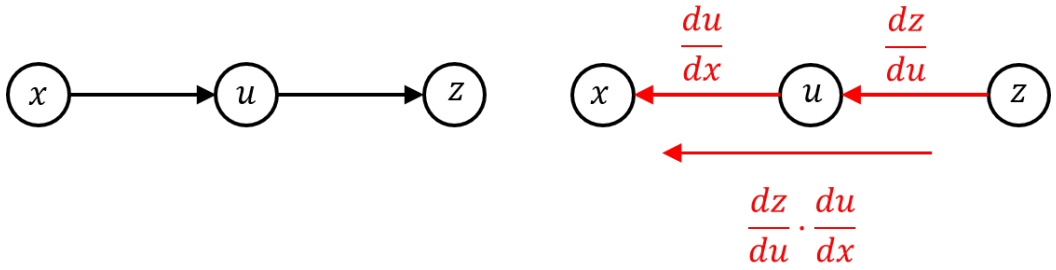
記号の節約のため，関数 $y = f(x)$ を $y = y(x)$ などと書きます．

１変数関数の合成関数の微分

関数 $z = z(u)$，$u = u(x)$ がともに微分可能なとき，

$$\frac{dz}{dx} = \frac{dz}{du}\frac{du}{dx} = z'(u)\,u'(x)$$

が成り立ちます．変数の依存関係を



という図式で表すとき，合成関数の微分公式は矢印に対応する微分をかけ合わせたものと解釈できます．

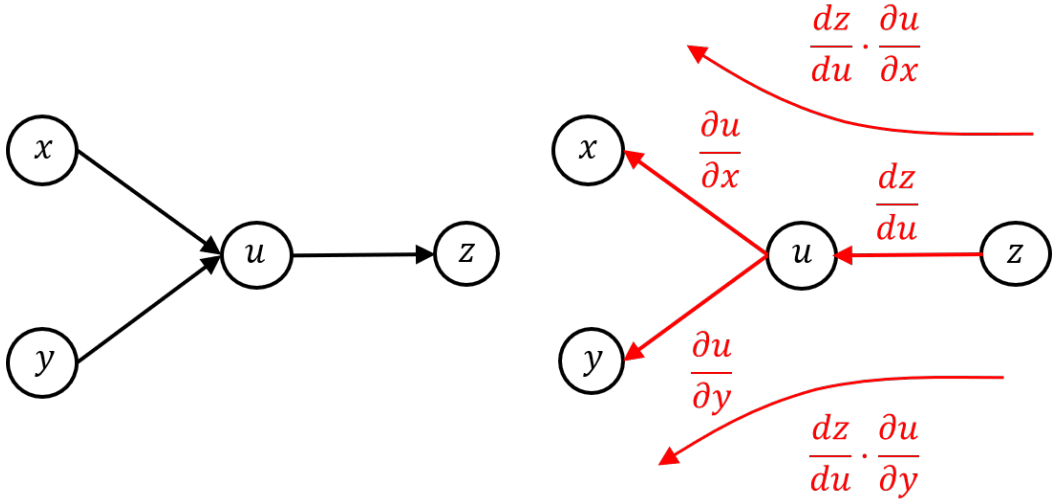
２変数関数の合成関数の微分

パターン１：

関数 $z = z(u)$ **が微分可能かつ** $u = u(x, y)$ **が偏微分微分可能なとき，**

$$\frac{\partial z}{\partial x} = \frac{dz}{du}\frac{\partial u}{\partial x} = z'(u)\,u_x(x, y) \quad \text{および} \quad \frac{\partial z}{\partial y} = \frac{dz}{du}\frac{\partial u}{\partial y} = z'(u)\,u_y(x, y)$$

が成り立ちます．変数の依存関係は



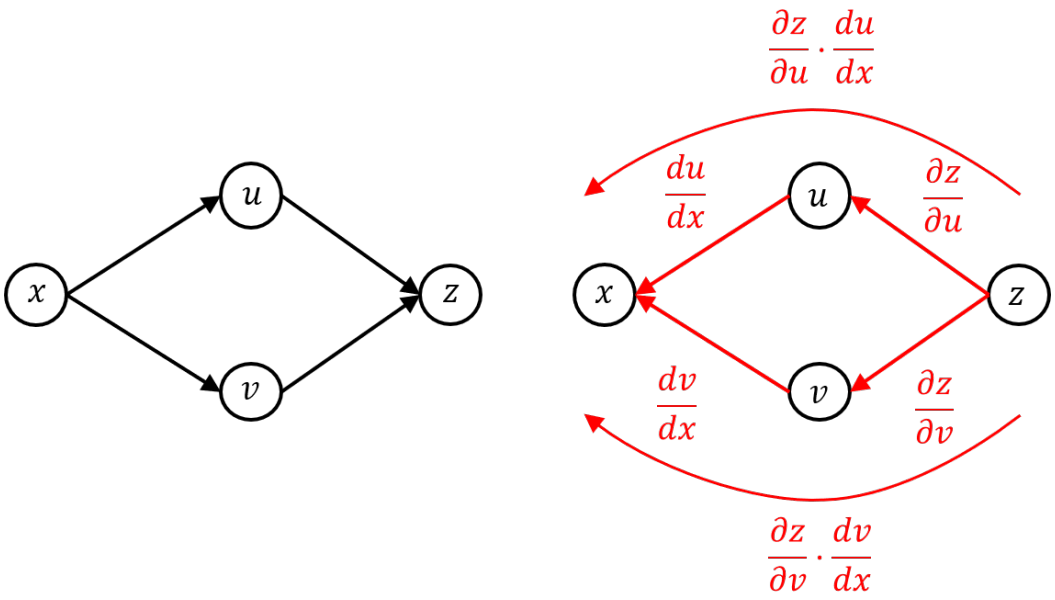
という図式で表されます．この場合も合成関数の微分公式は矢印に沿って微分をかけ合わせたものと解釈できます．

パターン 2：

関数 $z = z(u, v)$ が全微分可能かつ $u = u(x)$ および $v = v(x)$ が微分微分可能なとき，

$$\frac{dz}{dx} = \frac{\partial z}{\partial u}\frac{du}{dx} + \frac{\partial z}{\partial v}\frac{dv}{dx} = z_u(u, v)\, u'(x) + z_v(u, v)\, v'(x)$$

が成り立ちます．変数の依存関係は



という図式で表されます．この場合，矢印をたどって x から z に行くルートは２つありますが，合成関数の微分公式はそれぞれのルートについて矢印に沿って微分をかけたものを足し合わせたものと解釈できます．

パターン 3：

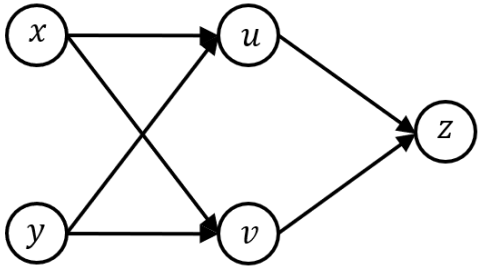
関数 $z = z(u, v)$ が全微分可能かつ $u = u(x, y)$ および $v = v(x, y)$ が偏微分微分可能なとき，

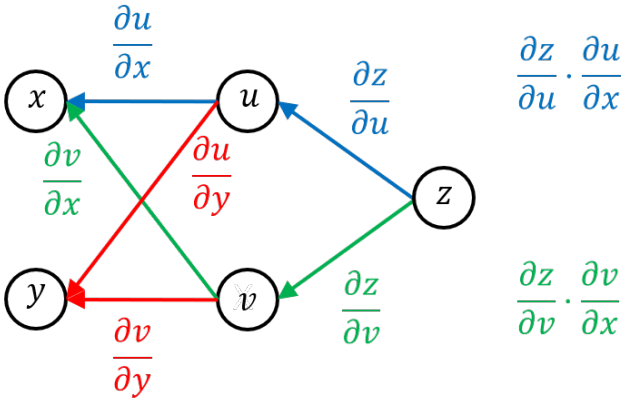
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial z}{\partial v}\frac{\partial v}{\partial x} = z_u(u, v)\, u_x(x, y) + z_v(u, v)\, v_x(x, y)$$

および

$$\frac{\partial z}{\partial y} = \frac{\partial z}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial z}{\partial v}\frac{\partial v}{\partial y} = z_u(u, v)\, u_y(x, y) + z_v(u, v)\, v_y(x, y)$$

が成り立ちます．変数の依存関係は





という図式で表されます．この場合，矢印をたどって x や y から z に行くルートは２つずつありますが，合成関数の微分公式はやはり，それぞれのルートについて矢印に沿って微分をかけたものを足し合わせたものと解釈できます．

3.3　連鎖律に関する例題 ♯

（余裕があったらやって下さい）

(1) $z = z(u),\ u = \dfrac{1}{1+e^{-x}}$ **（シグモイド関数）のとき，** $\dfrac{dz}{dx}$ **を** $\dfrac{dz}{du}$ **を用いて表してください．**

(2) (1) **で** $z = -t\log u - (1-t)\log(1-u)$**，（0-1 値の交差エントロピー）のとき，** $\dfrac{dz}{dx}$ **を** t, u **を用いて表してください．**

(3) $z = z(u,v),\ u = 1+e^{x}$**，** $v = 1+e^{-x}$**(入力** x **が** u**，** v **に分離してから** z **で合流) のとき，** $\dfrac{dz}{dx}$ **を** $\dfrac{\partial z}{\partial u}$ **および** $\dfrac{\partial z}{\partial v}$ **を用いて表してください．**

(4) $z = z(u,v),\ u = \dfrac{e^{x}}{e^{x}+e^{y}}$**，** $v = \dfrac{e^{y}}{e^{x}+e^{y}}$**(２変数のソフトマックス関数) のとき，** $\dfrac{\partial z}{\partial x}$ **および** $\dfrac{\partial z}{\partial y}$ **を** $\dfrac{\partial z}{\partial u}$ **および** $\dfrac{\partial z}{\partial v}$ **を用いて表してください．**

(5) (4) **で** $z = -s\log u - t\log v$**，ただし，** $s+t=1$ **かつ** $u+v=1$**（2 変数の交差エントロピー）のとき，** $\dfrac{\partial z}{\partial x}$ **および** $\dfrac{\partial z}{\partial y}$ **を** s, t, u, v **を用いて表してください．**

(6) **実数** w_{00}**，** w_{10}**，** w_{01}**，** w_{11}**，** b_0**，** b_1 **に対して，** $z = z(y_0, y_1),\ (y_0, y_1) = (x_0w_{00} + x_1w_{10} + b_0,\ x_0w_{01} + x_1w_{11} + b_1)$ **のとき，** $\dfrac{\partial z}{\partial x_0}$ **および** $\dfrac{\partial z}{\partial x_1}$ **を** $\dfrac{\partial z}{\partial y_0}$ **および** $\dfrac{\partial z}{\partial y_1}$ **を用いて表してください．**

(解答)

(1) $\dfrac{dz}{dx} = \dfrac{e^{-x}}{(1+e^{-x})^2}\dfrac{dz}{du} = u(1-u)\dfrac{dz}{du}$

(2) (1) の結果に $\dfrac{\partial z}{\partial u} = -\dfrac{t}{u} + \dfrac{1-t}{1-u}$ を代入して，$\dfrac{dz}{dx} = -t(1-u) + (1-t)u = u - t$

(3) $\dfrac{dz}{dx} = e^{x}\dfrac{\partial z}{\partial u} - e^{-x}\dfrac{\partial z}{\partial v}$

(4) $\dfrac{\partial u}{\partial x} = \dfrac{\partial v}{\partial y} = \dfrac{e^{x+y}}{(e^x+e^y)^2} = uv$，$\dfrac{\partial u}{\partial y} = \dfrac{\partial v}{\partial x} = \dfrac{e^{x+y}}{(e^x+e^y)^2} = -uv$ より，

$\dfrac{\partial z}{\partial x} = uv\left(\dfrac{\partial z}{\partial u} - \dfrac{\partial z}{\partial v}\right)$，$\dfrac{\partial z}{\partial y} = uv\left(-\dfrac{\partial z}{\partial u} + \dfrac{\partial z}{\partial v}\right)$

(5) (4) の結果に $\dfrac{\partial z}{\partial u} = -\dfrac{s}{u}$，$\dfrac{\partial z}{\partial v} = -\dfrac{t}{v}$ を代入して，$s+t=1$，$u+v=1$ を用いると，

$\dfrac{\partial z}{\partial x} = -vs + ut = -(1-u)s + u(1-s) = u - s$

$\dfrac{\partial z}{\partial y} = vs - ut = v(1-t) - (1-v)t = v - t$

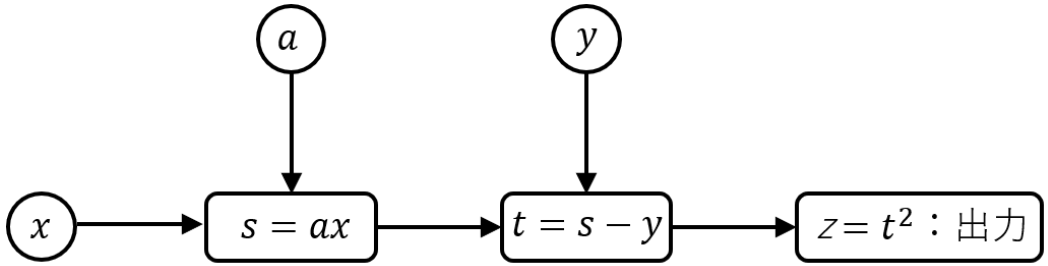
(6) $\dfrac{\partial z}{\partial x_0} = w_{00}\dfrac{\partial z}{\partial y_0} + w_{01}\dfrac{\partial z}{\partial y_1}$，$\dfrac{\partial z}{\partial x_1} = w_{10}\dfrac{\partial z}{\partial y_0} + w_{11}\dfrac{\partial z}{\partial y_1}$

3.4　計算グラフ

　複雑な関数であっても，単純な関数を何度も合成して表すことにより，その微分を連鎖律を用いて計算できます．このとき，その合成の仕方を表すグラフを計算グラフといいます．計算グラフの各頂点（ノード）は関数（およびその出力）を表し，矢印は元の頂点からの出力が先の頂点の入力として受け渡されることを表します．Tensorflow や PyTorch などの深層学習フレームワークでは計算グラフを用いて自動的に連鎖律を適用する機能（自動微分）が提供されています．

計算グラフの簡単な例

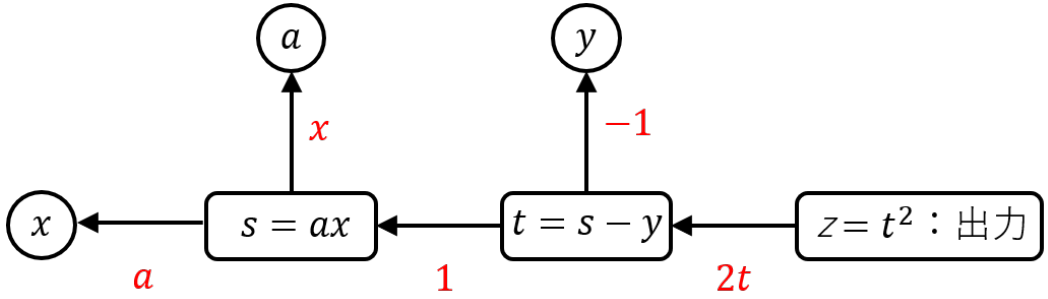
3 つの実数 x，a，y の関数 $z = (ax - y)^2$ は，



という計算グラフに分解できます．ここで，

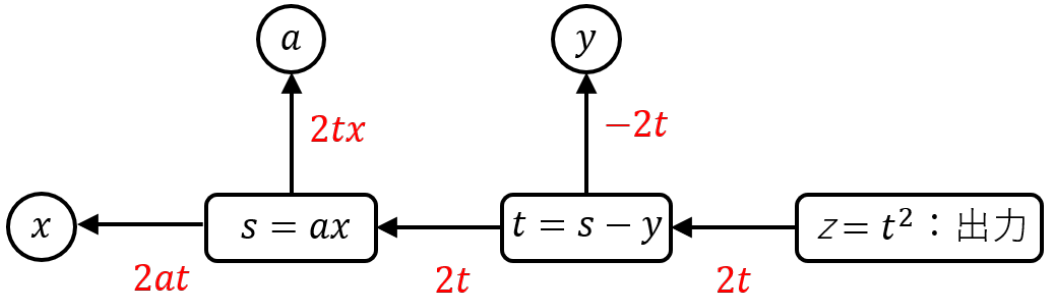
$$\frac{dz}{dt} = 2t, \quad \frac{\partial t}{\partial y} = -1, \quad \frac{\partial t}{\partial s} = 1, \quad \frac{\partial s}{\partial a} = x, \quad \frac{\partial s}{\partial x} = a$$

より，



という逆向きのグラフを得ます．矢印の横の式は，矢印の元の変数を矢印の先の変数で微分したものです．（次ページにつづく）

チェインルールによれば，これらを矢印をたどるごとに順次かけていけば各変数に関する出力 z の微分を得るはずです．実際にやってみると，



となり，これより，

$$\frac{\partial z}{\partial x} = 2at = 2a(s-y) = 2a(ax-y)$$
$$\frac{\partial z}{\partial a} = 2xt = 2x(s-y) = 2x(ax-y)$$
$$\frac{\partial z}{\partial y} = -2t = -2(s-y) = -2(ax-y)$$

と正しい結果を得ます．

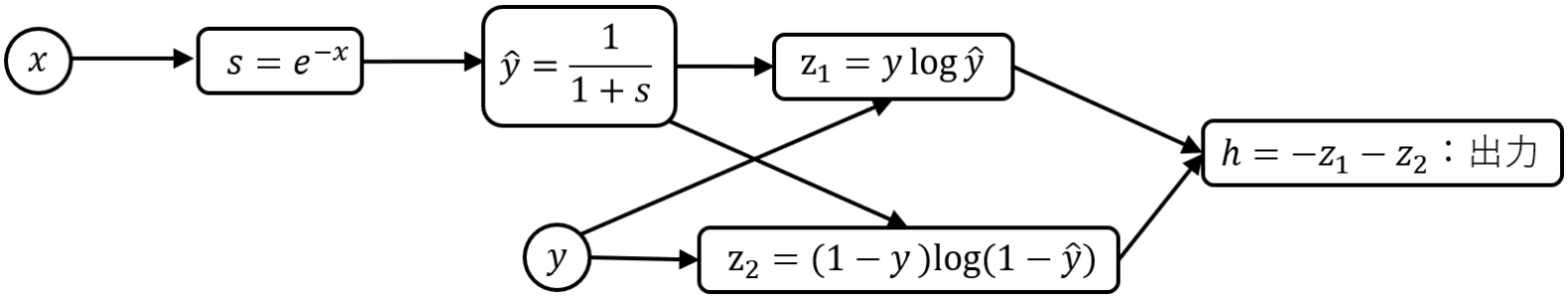
このように，計算グラフを用いることで出力に対する微分が元のグラフを逆にたどって伝わっていきます．

計算グラフのやや複雑な例

2 つの実数 x，y（ただし，$0 \leq y \leq 1$）に対してシグモイド関数の出力との（2 値の）交差エントロピー

$$h = -y \log \hat{y} - (1-y) \log(1-\hat{y}), \text{（ただし，} \hat{y} = \frac{1}{1+e^{-x}}\text{）}$$

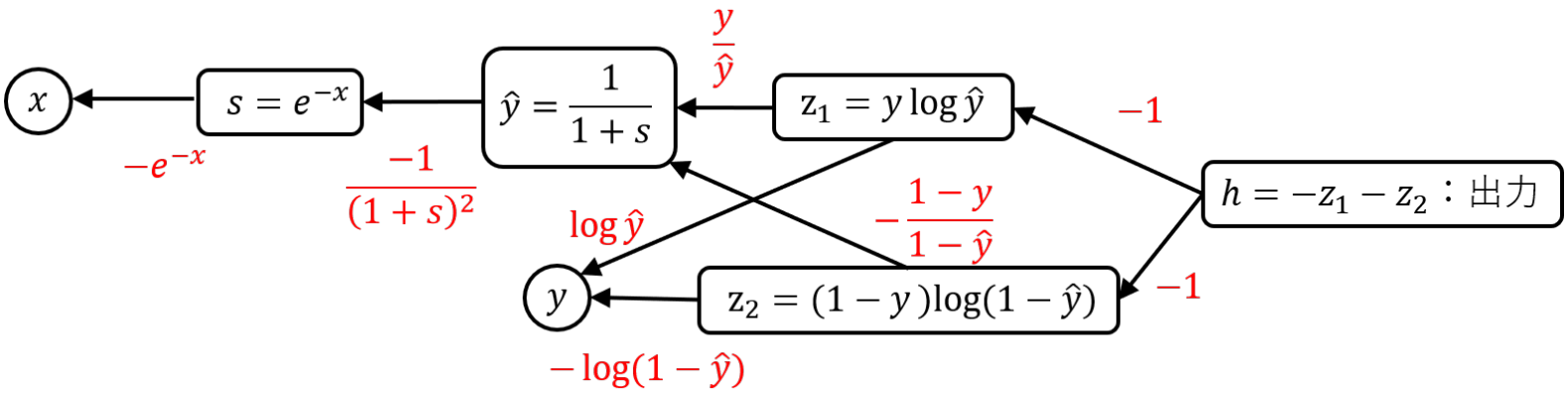
を考えてみましょう．この関数は



という計算グラフに分解できます（もっと細かく分解できますが，その分グラフが大きくなります）．ここで，

$$\frac{\partial h}{\partial z_1} = \frac{\partial h}{\partial z_2} = -1, \quad \frac{\partial z_1}{\partial y} = \log \hat{y}, \quad \frac{\partial z_1}{\partial \hat{y}} = \frac{y}{\hat{y}}, \quad \frac{\partial z_2}{\partial y} = -\log(1-\hat{y}),$$
$$\frac{\partial z_2}{\partial \hat{y}} = -\frac{1-y}{1-\hat{y}}, \quad \frac{d\hat{y}}{ds} = \frac{-1}{(1+s)^2}, \quad \frac{ds}{dx} = -e^{-x}$$

より，



という逆向きのグラフを得ます．チェインルールによれば，矢印が合流する場合は加えれば良いので，（$\hat{y} = \frac{1}{1+s}$ より，$s = \frac{1-\hat{y}}{\hat{y}}$ を用いると）

$$\frac{\partial h}{\partial y} = -\log \hat{y} + \log(1-\hat{y}) = \log \frac{1-\hat{y}}{\hat{y}} = \log s = \log e^{-x} = -x$$

$$\frac{\partial h}{\partial x} = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \cdot \frac{-1}{(1+s)^2} \cdot (-e^{-x}) = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \cdot \frac{s}{(1+s)^2}$$

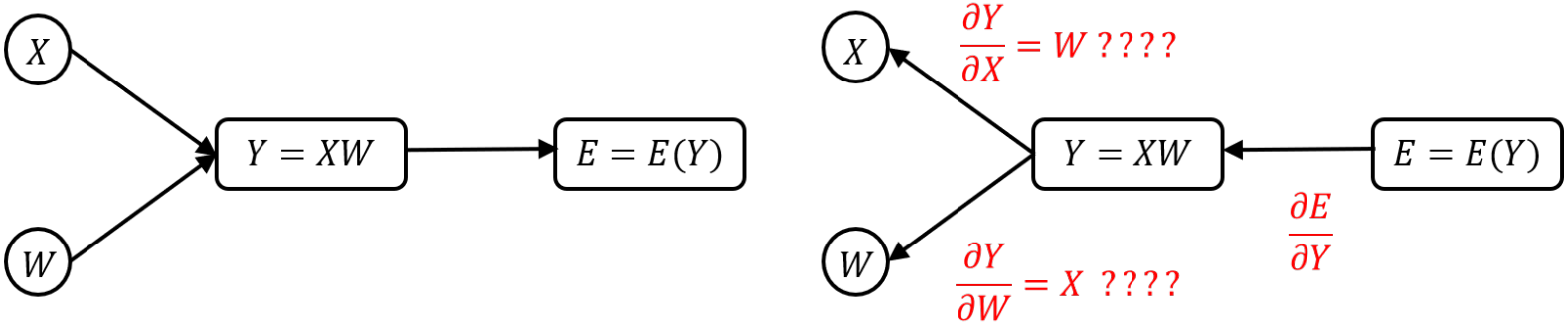
$$= \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \cdot \hat{y}(1-\hat{y})$$（この変形はやや技巧的）

$$= \hat{y} - y$$

と計算できます．

3.5　ニューラルネットワークにおける計算グラフ

ニューラルネットワークでは層全体を計算グラフのノード，つまり頂点として考えることができます．この場合，計算グラフのノードへの入出力は一つの数ではなく，高階の配列となります．このようにまとめて考えることで誤差逆伝播の実装が容易になりますが，その代わりにノードでのチェインルールの適用は複雑になります．



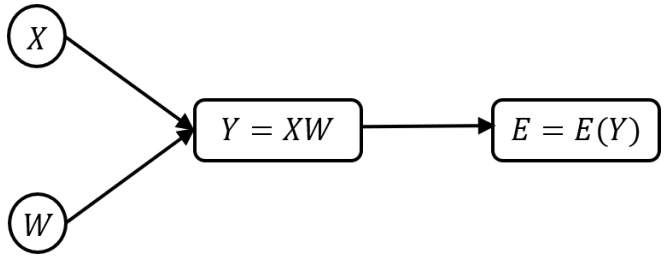
X，W は行列で E はスカラーとするとき，$\frac{\partial E}{\partial X}$ や $\frac{\partial E}{\partial W}$ を計算したいとします．

しかし，$\frac{\partial E}{\partial Y}$ のようにスカラーの行列に関する勾配を考えることはできますが，

$\frac{\partial Y}{\partial X}$ や $\frac{\partial Y}{\partial W}$ のように行列の行列に関する勾配は（通常は）考えられません．

ニューラルネットワークにおける基本的な例

例として，行列の積のノード



を考えてみましょう．ここで，X，W，Y は行列であり，$E = E(Y)$ は最終的なモデルの誤差関数で１次元であるとします．誤差逆伝播法では，$\dfrac{\partial E}{\partial Y}$ を与えたときに，$\dfrac{\partial E}{\partial X}$ および $\dfrac{\partial E}{\partial W}$ を求めることが目的です．

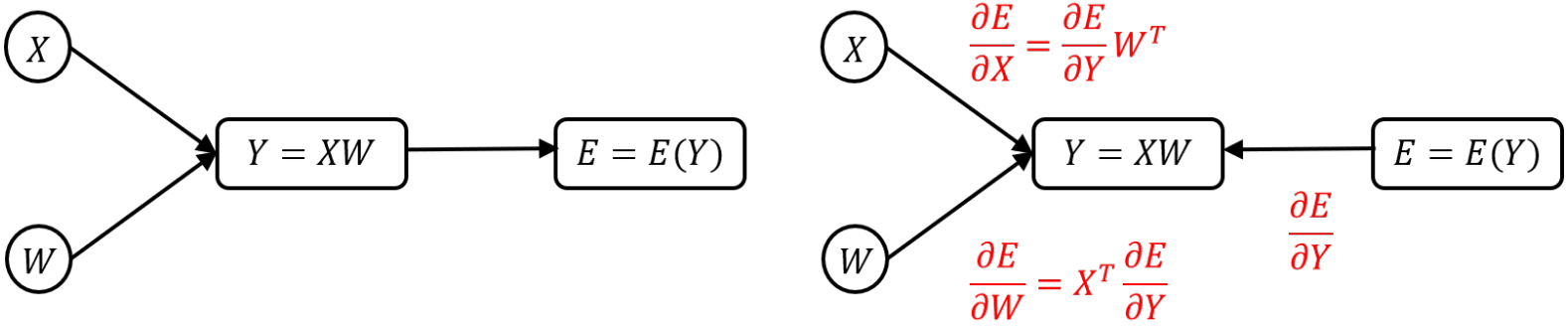
※記号：行列 X と関数 $f(X)$ に対して勾配 $\dfrac{\partial f}{\partial X} = \nabla_X f$ は X と同じ型で，(i,j) 成分を $\dfrac{\partial f}{\partial x_{ij}}$ とする行列を表します．X がベクトルやテンソルのときも同様です．

結論は，

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} W^T, \qquad \frac{\partial E}{\partial W} = X^T \frac{\partial E}{\partial Y}$$

となります．後ほどこの公式の導出を行います．

次元を利用した覚え方



という公式を導きました．この公式は大変覚えにくいのですが，実は積の微分は $\dfrac{\partial E}{\partial Y}$ と X や W の積になるということさえ覚えておけば，あとは次元の計算だけで自動的に決まります．

実際，X を (N, K) 型行列，W を (K, L) 型行列とすると，Y は (N, L) 型であり，

$$\frac{\partial E}{\partial X} : (N, K) \text{ 型}, \quad \frac{\partial E}{\partial W} : (K, L) \text{ 型}, \quad \frac{\partial E}{\partial Y} : (N, L) \text{ 型}$$

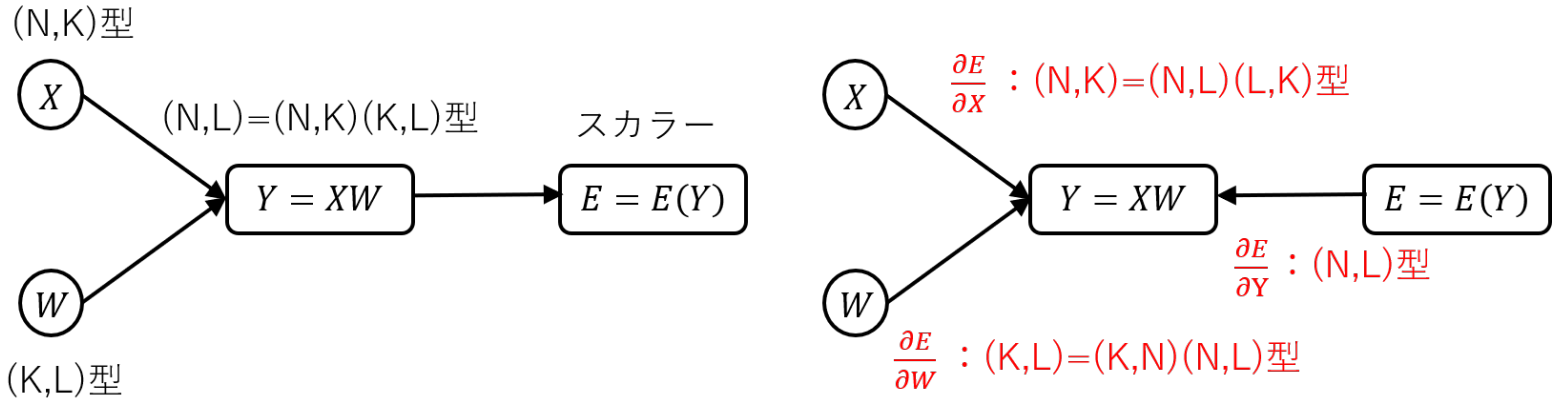
です．

次元を利用した覚え方（つづき）

例えば，$\frac{\partial E}{\partial Y} : (N, L)$ 型と積をとって $\frac{\partial E}{\partial X}$ の (N, K) 型にするには，

$$(N, K) = (N, L) \times (L, K)$$

より，右から (L, K) 型の変数をかけなければなりませんが，このようなものは W^T しかありません．



問題 上記の次元の計算の手法を用いて，$\frac{\partial E}{\partial W} = X^T \frac{\partial E}{\partial Y}$ を導いてください．

公式の導出

一般のサイズでやるとややこしいので，すべての行列が $(2,2)$ 型の場合に計算してみましょう．

$$\begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{bmatrix} = \begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{bmatrix} = \begin{bmatrix} x_{00}w_{00} + x_{01}w_{10} & x_{00}w_{01} + x_{01}w_{11} \\ x_{10}w_{00} + x_{11}w_{10} & x_{10}w_{01} + x_{11}w_{11} \end{bmatrix}$$

より，

$$\begin{aligned} \frac{\partial E}{\partial X} &= \begin{bmatrix} \frac{\partial E}{\partial x_{00}} & \frac{\partial E}{\partial x_{01}} \\ \frac{\partial E}{\partial x_{10}} & \frac{\partial E}{\partial x_{11}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_{00}}w_{00} + \frac{\partial E}{\partial y_{01}}w_{01} & \frac{\partial E}{\partial y_{00}}w_{10} + \frac{\partial E}{\partial y_{01}}w_{11} \\ \frac{\partial E}{\partial y_{10}}w_{00} + \frac{\partial E}{\partial y_{11}}w_{01} & \frac{\partial E}{\partial y_{10}}w_{10} + \frac{\partial E}{\partial y_{11}}w_{11} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial E}{\partial y_{00}} & \frac{\partial E}{\partial y_{01}} \\ \frac{\partial E}{\partial y_{10}} & \frac{\partial E}{\partial y_{11}} \end{bmatrix} \begin{bmatrix} w_{00} & w_{10} \\ w_{01} & w_{11} \end{bmatrix} = \frac{\partial E}{\partial Y} W^T \end{aligned}$$

となります．よって $Y = XW$ に対して，

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} W^T$$

を得ます．また，同様に計算することにより，

$$\frac{\partial E}{\partial W} = X^T \frac{\partial E}{\partial Y}$$

を得ます．

3.6　誤差逆伝播に関する例題

1. $X = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix}$，$W = \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 4 \end{bmatrix}$，$Y = XW$ とします．$E = E(Y)$（E は 1 変数）とします．

   (1) 正解 $(t_1, t_2) = (1, 2)$ に対する誤差が $E = \dfrac{1}{2}\left((y_1 - t_1)^2 + (y_2 - t_2)^2\right)$ のとき，$\dfrac{\partial E}{\partial X}$，$\dfrac{\partial E}{\partial W}$ を求めて下さい．

   (2) 正解 $(t_1, t_2) = (1, 0)$ に対する誤差が $E = -t_1 \log\left(\dfrac{e^{y_1}}{e^{y_1} + e^{y_2}}\right) - t_2 \log\left(\dfrac{e^{y_2}}{e^{y_1} + e^{y_2}}\right)$ のとき，$\dfrac{\partial E}{\partial X}$，$\dfrac{\partial E}{\partial W}$ を求めて下さい．

2. $X = \begin{bmatrix} 1 & 3 \\ 2 & -1 \\ 0 & 1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 2 & 1 \end{bmatrix}$, $Y = X + \mathbf{b} = \begin{bmatrix} 1+2 & 3+1 \\ 2+2 & -1+1 \\ 0+2 & 1+1 \end{bmatrix}$ (ブロードキャスト計算)

とします．$E = E(Y)$（E は 1 変数）とします．$\dfrac{\partial E}{\partial Y} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix}$ のとき，$\dfrac{\partial E}{\partial X}$，$\dfrac{\partial E}{\partial \mathbf{b}}$ を

求めて下さい．

(解答)

1. (1) $Y = \begin{bmatrix} 3 & 3 \end{bmatrix}$ **より，** $\dfrac{\partial E}{\partial Y} = \begin{bmatrix} y_1 - t_1 & y_2 - t_2 \end{bmatrix} = \begin{bmatrix} 2 & 1 \end{bmatrix}$ **なので，**

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} W^T = \begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 4 \end{bmatrix}$$

$$\frac{\partial E}{\partial W} = X^T \frac{\partial E}{\partial Y} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ -2 & -1 \\ 2 & 1 \end{bmatrix}$$

(2) **??連鎖律に関する例題** (5) **より，** $\dfrac{\partial E}{\partial Y} = \begin{bmatrix} \frac{e^{y_1}}{e^{y_1}+e^{y_2}} - t_1 & \frac{e^{y_2}}{e^{y_1}+e^{y_2}} - t_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$ **なので，**

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} W^T = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 4 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 2 \end{bmatrix}$$

$$\frac{\partial E}{\partial W} = X^T \frac{\partial E}{\partial Y} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

]

2. $\begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \\ y_{20} & y_{21} \end{bmatrix} = \begin{bmatrix} x_{00} + b_0 & x_{01} + b_1 \\ x_{10} + b_0 & x_{11} + b_1 \\ x_{20} + b_0 & x_{21} + b_1 \end{bmatrix}$ より，

$$\frac{\partial E}{\partial X} = \begin{bmatrix} \frac{\partial E}{\partial x_{00}} & \frac{\partial E}{\partial x_{01}} \\ \frac{\partial E}{\partial x_{10}} & \frac{\partial E}{\partial x_{11}} \\ \frac{\partial E}{\partial x_{20}} & \frac{\partial E}{\partial x_{21}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_{00}} & \frac{\partial E}{\partial y_{01}} \\ \frac{\partial E}{\partial y_{10}} & \frac{\partial E}{\partial y_{11}} \\ \frac{\partial E}{\partial y_{20}} & \frac{\partial E}{\partial y_{21}} \end{bmatrix} = \frac{\partial E}{\partial Y} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix}$$

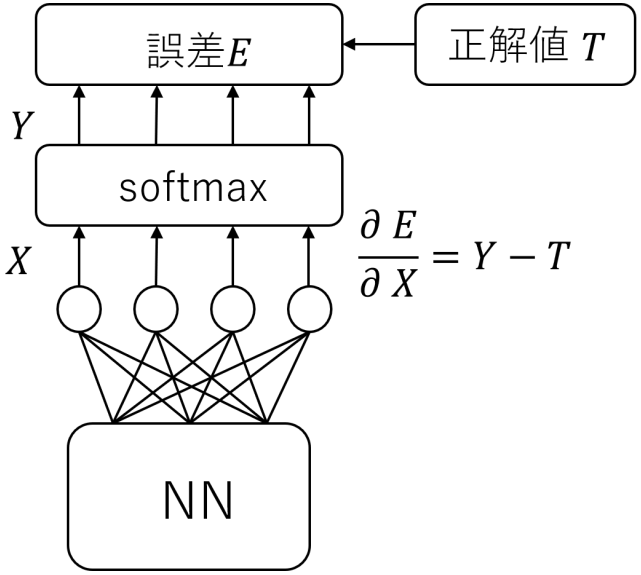
$$\begin{aligned} \frac{\partial E}{\partial \mathbf{b}} &= \begin{bmatrix} \frac{\partial E}{\partial b_0} & \frac{\partial E}{\partial b_1} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_{00}} + \frac{\partial E}{\partial y_{10}} + \frac{\partial E}{\partial y_{20}} & \frac{\partial E}{\partial y_{01}} + \frac{\partial E}{\partial y_{11}} + \frac{\partial E}{\partial y_{21}} \end{bmatrix} \\ &= \sum_{i=0}^{2} \begin{bmatrix} \left(\frac{\partial E}{\partial Y}\right)_{i0} & \left(\frac{\partial E}{\partial Y}\right)_{i1} \end{bmatrix} = \mathrm{sum}\left(\begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix}, \mathrm{axis} = 0 \right) = \begin{bmatrix} 2 & 0 \end{bmatrix} \end{aligned}$$

ここで，配列 X に対して $\mathrm{sum}(X, \mathrm{axis} = 0)$ は，NumPy の関数と同様に第 0 インデックス（つまり，$X = (x_{ij})$ のときは i）に関する和とします．

3.7　逆伝播公式のまとめ

データ一つのとき

- 回帰問題の出力層：2 乗誤差の $\frac{1}{2}$：$E = \frac{1}{2}(y-t)^2$ に対して $\frac{\partial E}{\partial y} = y-t$
  （t：正解値，y：予測値）
- 0-1 値 2 クラス分類の出力層（シグモイド + 交差エントロピー）：
  $\frac{\partial E}{\partial x} = y - t$
- K クラス分類の出力層（ソフトマックス + 交差エントロピー）：
  $\frac{\partial E}{\partial \mathbf{x}} = \mathbf{y} - \mathbf{t}$（ t：正解値をワンホット表現したベクトル）



Softmax with loss の逆伝播

- アファイン変換層 $E = E(\mathbf{y})$，$\mathbf{y} = \mathbf{x}W + \mathbf{b}$ に対して，

$$\frac{\partial E}{\partial \mathbf{x}} = \frac{\partial E}{\partial \mathbf{y}} W^T, \quad \frac{\partial E}{\partial W} = \mathbf{x}^T \frac{\partial E}{\partial \mathbf{y}}, \quad \frac{\partial E}{\partial \mathbf{b}} = \frac{\partial E}{\partial \mathbf{y}}$$

- $E = E(\mathbf{y}), \mathbf{y} = \frac{1}{1+e^{-\mathbf{x}}}$（シグモイド関数） のとき，$\frac{\partial E}{\partial \mathbf{x}} = \mathbf{y} * (1-\mathbf{y}) * \frac{\partial E}{\partial \mathbf{y}}$
- $E = E(\mathbf{y}), \mathbf{y} = \max\{0, \mathbf{x}\}$（ReLU 関数） のとき，$\frac{\partial E}{\partial x_k} = \begin{cases} \frac{\partial E}{\partial y_k} & (x_k > 0 \text{ のとき}) \\ 0 & (x_k \leq 0 \text{ のとき}) \end{cases}$

データ数 N のミニバッチのとき—これだけ憶えておけば NN が実装できます

- 回帰問題の出力層：2 乗誤差の和 $\frac{1}{2}$：$E = \frac{1}{2}\sum_{n=0}^{N-1}(y_n - t_n)^2$ に対して $\frac{\partial E}{\partial \mathbf{y}} = \mathbf{y} - \mathbf{t}$
- 0-1 値 2 クラス分類の出力層（シグモイド + 交差エントロピーの和）：$\frac{\partial E}{\partial \mathbf{x}} = \mathbf{y} - \mathbf{t}$
- K クラス分類の出力層（ソフトマックス + 交差エントロピーの和）：$\frac{\partial E}{\partial X} = Y - T$

  (ミニバッチに対して通常は誤差の和ではなく平均をとるので，上の 3 つを N で割ります．)

- アファイン変換層 $E = E(Y)$，$Y = XW + \mathbf{b}$ に対して，

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y}W^T, \quad \frac{\partial E}{\partial W} = X^T\frac{\partial E}{\partial Y}, \quad \frac{\partial E}{\partial \mathbf{b}} = \mathrm{sum}\left(\frac{\partial E}{\partial Y}, \mathrm{axis} = 0\right)$$

- $E = E(Y)$, $Y = \frac{1}{1+e^{-X}}$（シグモイド関数） のとき，$\frac{\partial E}{\partial X} = Y * (1 - Y) * \frac{\partial E}{\partial Y}$
- $E = E(Y)$, $Y = \max\{0, X\}$（ReLU 関数） のとき，$\frac{\partial E}{\partial x_k^{(n)}} = \begin{cases} \frac{\partial E}{\partial y_k^{(n)}} & (x_k^{(n)} > 0 \text{ のとき}) \\ 0 & (x_k^{(n)} \leq 0 \text{ のとき}) \end{cases}$

3.8　PyTorch による計算グラフの可視化に関する演習

使用するノートブック: 2-1 自動微分

PyTorch は Facebook の開発した Torch を基にした深層学習のライブラリです（Chainer の設計思想を継承して高速化させたものと言われています）．2-1 では，PyTorch による計算グラフの生成と，自動微分を行います．特に行列計算と SoftmaxWithLoss 層の計算グラフを作成し，自動微分を行ってみましょう．

- 計算グラフのノードと辺
- PyTorch と Numpy の類似性と違い
- 連鎖律
- PyTorch で自動微分を行うためには，計算グラフの最後の出力は 1 次元でなければならないこと

※最後の点は，例えば X，$Y = f(X)$ がともに行列のとき，$\dfrac{\partial Y}{\partial X}$ が 4 階の配列となってしまうことに起因します．

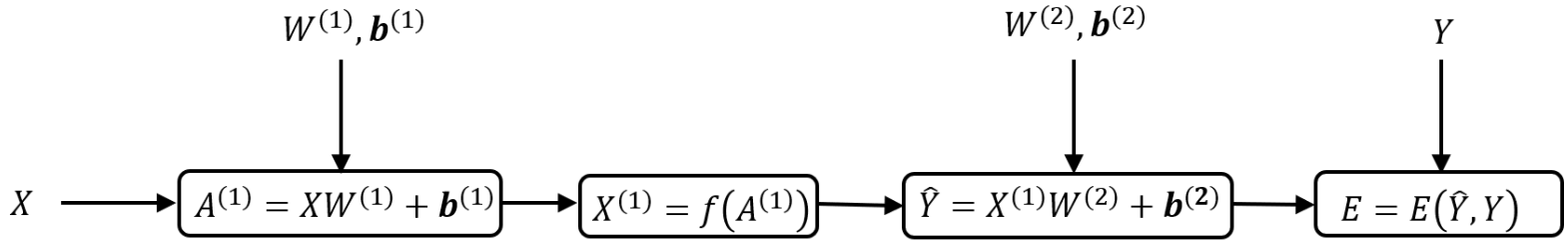
4　2 層のニューラルネットワーク

入力層（第 0 層），隠れ層（第 1 層），出力層（第 2 層）からなるニューラルネットワークを 2 層のニューラルネットワークといいます．3 層という数え方もありますが，入力層では演算を施さないので実質的な層の数を数えると 2 層となります．

2 層のニューラルネットワークは最も基本的なニューラルネットワークなので，一からコードが書けるように練習しましょう．

4.1　2 層のニューラルネット（回帰）

教師ありの回帰問題において，入力データ（データ数 N のミニバッチ）を X，対応する予測値を $\hat{Y}$ とすると，2 層のニューラルネットワークは



と表せます．ただし，活性化関数を ReLU にするとき，

$$X^{(1)} = f(A^{(1)}) = \mathrm{relu}(A^{(1)}) = \max\{A^{(1)}, 0\}$$

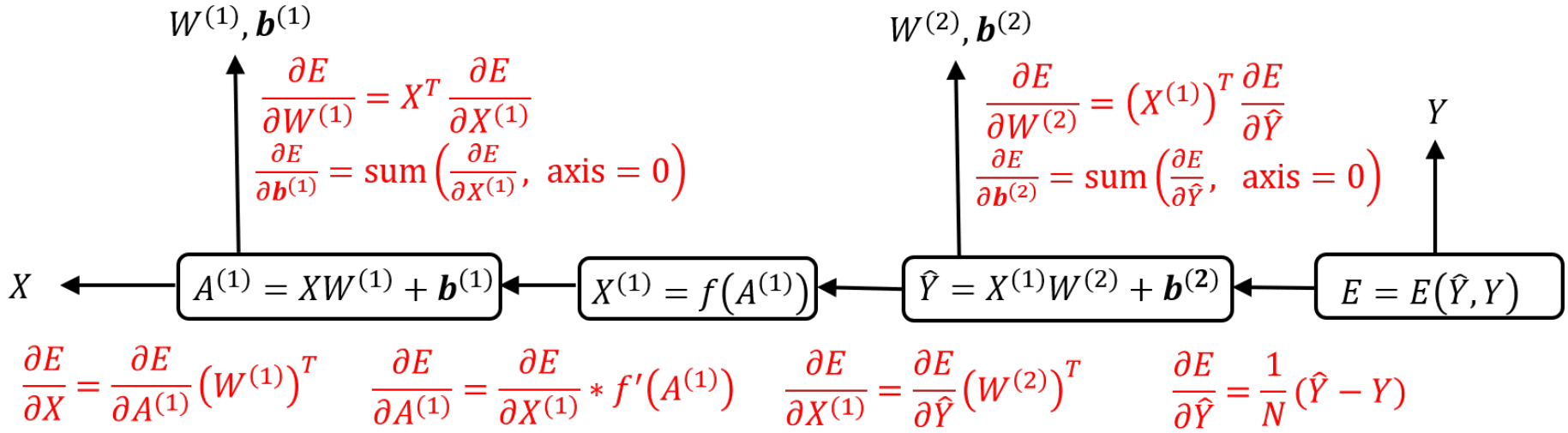
となり，予測 Y との平均 2 乗誤差は

$$E = \frac{1}{2N} \sum_{n=0}^{N-1} (\hat{Y}_n - Y_n)^2$$

となります．

2 層のニューラルネット（回帰）（つづき）

一方，逆方向の伝播は前節の公式により



となることが分かります．ただし，活性化関数を ReLU にするとき $\dfrac{\partial E}{\partial A^{(1)}}$ は

$$\frac{\partial E}{\partial A^{(1)}} = \frac{\partial E}{\partial X^{(1)}} * (\text{mask of } A^{(1)}), \qquad \left(\text{mask of } A^{(1)} = \textbf{論理式} : A^{(1)} > 0\right)$$

で与えられます．この mask は Python では以下のように書けます．

```
# forward のとき
mask1 = (a1 > 0)
x1 = mask1 * a1
```

```
#backward のとき
da1 = dx1 * mask1
```

確率的勾配降下法

パラメータの更新には確率的勾配降下法を用います．

確率的勾配降下法（Stochastic Gradient Decent）のアルゴリズム（再掲）：

1. パラメータ W の値を適当にとる
2. 正解付きの入力データの集合から，いくつかのサンプル（ミニバッチ）をランダムに選ぶ
3. 入力データそれぞれに対して予測値を計算する
4. 正解との誤差のミニバッチに関する平均 E のパラメータ W に関する微分 $\dfrac{\partial E}{\partial W}$ を計算
5. 実定数 $\lambda > 0$ に対して，パラメー W を

$$W \leftarrow W - \lambda \frac{\partial E}{\partial W}$$

と更新して 2 に戻る

4.2　Numpy によるシンプルな 2 層のニューラルネットワークに関する演習

使用するノートブック：

- 2-2-1_シンプルな 2 層のニューラルネット（回帰）
- 2-2-2_シンプルな 2 層のニューラルネット（2 値分類）
- 2-2-3_シンプルな 2 層のニューラルネット（分類）

次の点を意識して，Numpy によるシンプルな 2 層のニューラルネットワークに関する演習をおこなってください．

- 回帰問題・2 値分類問題・分類問題それぞれにおける出力層設計
- 誤差逆伝播
- クラスとそのパラメータ，メソッドの使い方
- シグモイドと ReLU の実装
- pandas の使用方法
- 訓練データに対する精度とテストデータに対する精度

Numpy によるシンプルな 2 層のニューラルネットワークに関する演習その 2

前ページのノートブックを参考にして

2-2-4_Simple2Layers_YearPrediction_問題

を解いてください．これは本講義の理解のためには欠かすことの出来ない演習問題です．必ずやりましょう

5　多層のニューラルネットワーク

　前節では 2 層のニューラルネットワークを定義し，確率的勾配降下法を用いて学習を行いました．本節では，層の数を増やします．

5.1 層のクラス

- 各層を Python のクラスとして実装します．
- 活性化関数は独立した層とします．
- ここで定義するのは以下のものです．
  - アファイン層
  - シグモイド層
  - ReLU 層
  - 回帰誤差の層
  - シグモイドと交差エントロピー誤差の層
  - ソフトマックスと交差エントロピー誤差の層

アファイン層

パラメータ：　行列 W，ベクトル $\mathbf{b}$

順伝播：

入力：行列 X

出力：

$$Y = XW + \mathbf{b}$$

逆伝播：

入力：行列 $\mathrm{dout} = \dfrac{\partial E}{\partial Y}$

出力：

$$\frac{\partial E}{\partial W} = X^T \cdot \mathrm{dout}$$

$$\frac{\partial E}{\partial \mathbf{b}} = \mathrm{dout} \text{ の行（0 番目のインデックス）に関する和}$$

$$\frac{\partial E}{\partial X} = \mathrm{dout} \cdot W^T$$

この式の一行目で X という変数を使わなければならないので，順方向の計算のときにこれを `self.x` として格納しておきます．上の二つは `self.dW` および `self.db`　というパラメータとして格納し，一番下のものを返り値とします．

シグモイド層

パラメータ：　なし

順伝播：

入力：行列 X

出力：

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

逆伝播：

入力：行列 $\mathrm{dout} = \dfrac{\partial E}{\partial Y}$

出力：

$$\frac{\partial E}{\partial X} = \mathrm{dout} * Y * (1 - Y)$$

逆伝播の計算には出力 Y **が必要なので，順方向の計算のときにこれをパラメータ** `self.y` **として格納しておきます．**

ReLU 層

パラメータ：　なし

順伝播：

入力：行列 X

出力：

$$Y = \max\{0, X\} = \begin{cases} 0 & (X \leq 0) \\ X & (X > 0) \end{cases}$$

コード：ブロードキャストを利用して以下のように書けます

```
self.mask = (x > 0) # x>0 なら True，x<=0 なら False
y = x * self.mask
```

逆伝播：

入力：行列 $\mathrm{dout} = \dfrac{\partial E}{\partial Y}$

出力：

$$\frac{\partial E}{\partial X} = \begin{cases} 0 & (X \leq 0) \\ \frac{\partial E}{\partial Y} & (X > 0) \end{cases}$$

コード：

```
dx = dout * self.mask
```

回帰誤差の層

パラメータ：　なし

順伝播：

入力：予測値 Y，正解値 T

出力：

$$E = \frac{1}{2N}\sum_n (y_n - t_n)^2 \quad (N \text{ はバッチサイズ})$$

コード：

```
batch_size = y.shape[0]
loss = np.sum((y-t) ** 2)/(2 * batch_size)
```

逆伝播：

入力：なし

出力：

$$\frac{\partial E}{\partial Y} = \frac{1}{N}(Y - T)$$

コード：　`dy = ( self.y - self.t )/batch_size`

シグモイドと交差エントロピー誤差の層

パラメータ：　なし

順伝播：

入力：正規化する前の予測値 X，正解値 T（データ一つの次元は 1）

出力：

$$Y = \mathrm{sigmoid}(X)$$
$$E = \frac{1}{N}(-T\log(Y) - (1-T)\log(1-Y)) \quad (N \text{ はバッチサイズ})$$

逆伝播：

入力：なし

出力：

$$\frac{\partial E}{\partial X} = \frac{1}{N}(Y - T)$$

ソフトマックスと交差エントロピー誤差の層

パラメータ：　なし

順伝播：

入力：正規化する前の予測値 X，正解値 T

出力：

$$Y = \mathrm{softmax}(X)$$

$$E = -\frac{1}{N}\sum_n \sum_k t_{n,k} \log y_{n,k} \quad (N \text{ はバッチサイズ})$$

コード：

```
y = softmax(x)
batch_size = y.shape[0]
loss =  -np.sum(t * np.log(y))/batch_size
```

逆伝播：

入力：なし

出力：

$$\frac{\partial E}{\partial X} = \frac{1}{N}(Y - T)$$

5.2　層のクラスに関する演習

使用するノートブック:

1. 3-1-1_アファイン層.ipynb
2. 3-1-2_SoftmaxWithLoss 層_演習問題.ipynb
3. 3-1-3_その他の層.ipynb

ノートブック 1 はアファイン層のクラスを定義し，数値微分を用いて確認をしています．ノートブック 2 はソフトマックスと交差エントロピー誤差の層のクラスを定義し，数値微分を用いて確認するためのものですが，一部のコードが欠けているのでコードを補って完成させてください．ノートブック 3 ではその他の層のクラスを定義しています．

なお，ソフトマックス関数の実装ではオーバーフロー対策として，

$$\frac{1}{e^{a_0}+\cdots+e^{a_{K-1}}}\left(e^{a_0},\cdots,e^{a_{K-1}}\right) \qquad \text{の代わりに}$$

$$\frac{1}{e^{a_0-c}+\cdots+e^{a_{K-1}-c}}\left(e^{a_0-c},\cdots,e^{a_{K-1}-c}\right) \qquad (\text{ただし，} c=\max\{a_0,\cdots,a_{K-1}\})$$

を用いています．このように変更しても関数の値は変わりません．

数値微分についての注意

　前ページの演習では，各層の誤差逆伝播を数値微分と比較して実装の正しさを確かめました．
ということは

そもそも数値微分をすれば勾配は求まるのでは？

と思われるかもしれません．答えはその通りです．しかし，数値微分は重みパラメータの成分一つ一つに対してそれぞれ順伝播を計算しなくてはならないので，多くのパラメータを持つモデルに対しては計算機への負荷が非常に高くなり，実装のテスト以外では使われません．

5.3　多層のニューラルネットワークについての演習

使用するノートブック:　2-4_多層 NN(分類).ipynb

このノートブックでは多層のニューラルネットワークのクラスを定義して，MNIST の手書き数字データセットを用いて学習を行います．以下のことに注意して演習を行ってください．

- 層の格納方法：層オブジェクトをリストで持ちます
- （順序付き）辞書の使い方：重み付きパラメータは順序付き辞書で持ちます
- 順伝播
- 逆伝播
- 層を深くするとどうなるか

深層モデルのための最適化に関する工夫

前ページの演習では，単純に層を深くしても学習が進まなくなってしまうことを学びました．このことはニューラルネットワークが 1970 年代には発明されていたにも関わらず長い間実用的ではないとされてきた要因となりました．しかし，その間にも少しずつ改良が進められてきました．

特に以下のものが基本的：

- 最適化法の改良：SGD に代わる最適化法
- 重みパラメータの初期値の取り方
- バッチ正則化：各層の出力をバッチに関して正規化する手法

続く３つの節ではこれらについて学び，層が深くなっても学習ができるようにして行きます．

6　最適化法の改良

本節では，確率的勾配降下法（SGD）を収束がしやすくなるように改良した最適化手法について学びます．

- モメンタム：「運動量（モメンタム）」を導入する手法
- ネステロフのモメンタム：勾配法の「加速アルゴリズム」に基づく方法
- AdaGrad：学習係数を勾配に対して適応的に変化させる手法
- RMSProp：AdaGrad を改良した手法
- Adam：AdaGrad および RMSProp をさらに改良した手法（最近良く用いられる）

このうち講義ではモメンタムと Adam について紹介します．

SGD の復習

1. 正解付きの入力データの集合から，N 個のサンプル（ミニバッチ）$\mathbf{x}^{(0)}, \cdots, \mathbf{x}^{(N-1)}$ をランダムに選び，$y^{(0)}, \cdots, y^{(N-1)}$ を対応する正解値とする
2. コスト関数の平均 $E = \dfrac{1}{N} \displaystyle\sum_{n=0}^{N-1} E\left(f(\mathbf{x}^{(n)}; \theta),\ y^{(n)}\right)$ のパラメータ θ に関する勾配

$$\nabla_\theta E = \frac{\partial E}{\partial \theta}$$

   を計算
3. パラメータ θ を

$$\theta \leftarrow \theta - \lambda \frac{\partial E}{\partial \theta}$$

   と更新

1 に関してはどの最適化法でも同じで，2，3 が変わります．

注：[Goodfellow] 8.3 節ではコスト関数を $L(f(\mathbf{x}; \theta),\ y)$ と対数尤度と同じ記号使っているので非常に紛らわしいですが，コスト関数は（正則化項などを無視すれば）対数尤度の符号を逆にしたものです． パラメータの最適化はコストを最小化 $\fallingdotseq$ 尤度を最大化する方向に進めます．

学習係数の大きさに関する注意

学習係数 λ は大きすぎると学習が不安定になったり発散したりしてしまい，一方，小さすぎると学習に時間がかかったり局所最適解から抜け出せなくなったりしてしまいます．

実践的には学習が進むに連れて減衰させるのが一般的ですが，これは始めは荒い網目で大局的に見てコスト関数が小さくなるように進み，最後に細かい網目で局所的に見ることでより良いパラメータを探すことに相当すると考えられます．

どのような最適化戦略をとるにしろ，勾配が 0 に収束したり発散してしまっては学習できません．このことに関しては次節で考察します．

6.1　モメンタム

モメンタムのアルゴリズム

α：$0 \leq \alpha < 1$ を満たすハイパーパラメータ

$\mathbf{v}$：θ と同じ型をもつ配列で初期値は 0.「運動量（モメンタム）」

終了条件を満たすまで以下を実行：

1. SGD と同じ
2. SGD と同じ
3. パラメータ θ およびモメンタム $\mathbf{v}$ を

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \lambda\frac{\partial E}{\partial \theta}$$
$$\theta \leftarrow \theta + \mathbf{v}$$

と更新

3 をまとめて書くと，

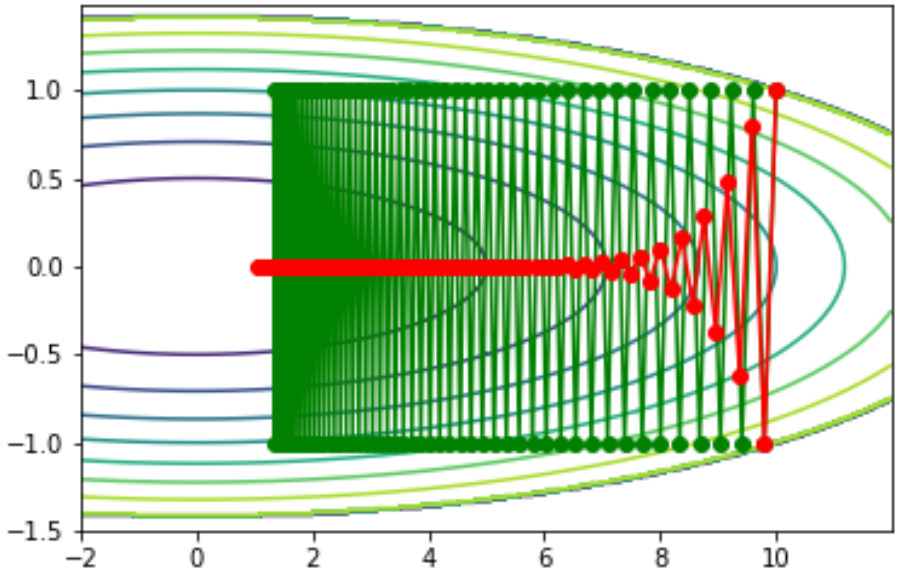
$$\theta \leftarrow \theta + \alpha\mathbf{v} - \lambda\frac{\partial E}{\partial \theta}$$

なので，$\alpha = 0$ のときは SGD と同じです．

勾配降下法とモメンタムの時間発展の比較

$$f(x, y) = x^2 + 100y^2$$

に対して，初期値 $(x, y) = (10, 1)$ から（確率的ではなく普通の）勾配降下法とモメンタム（のアルゴリズムのうち 2,3）を適用した結果



GD とモメンタムの時間発展の比較

緑：GD，赤：モメンタム

運動方程式との関係—なぜ「モメンタム」と呼ばれるのか

モメンタムはアルゴリズムの 1 ステップを時刻の 1 ステップとみなすとき，θ が時刻 t での物体の位置を，$\mathbf{v}$ が運動量を，$1-\alpha$ が摩擦係数を，$-\lambda\dfrac{\partial E}{\partial \theta}$ が力を表すと考えられます． 実際，

$$\begin{aligned}\theta(t+1)-\theta(t) &= \mathbf{v}(t+1)\\ \mathbf{v}(t+1)-\mathbf{v}(t) &= -(1-\alpha)\mathbf{v}(t)-\lambda\frac{\partial E}{\partial \theta}\end{aligned}$$

と，運動方程式

$$\begin{aligned}\dot{\theta}(t) &= \mathbf{v}(t)\\ \dot{\mathbf{v}}(t) &= -(1-\alpha)\mathbf{v}(t)-\lambda\frac{\partial E}{\partial \theta}\end{aligned}$$

の類似性は明らかでしょう．

6.2 Adam

学習係数の λ を，パラメータの成分ごとに学習の進み方に対して適応的に（adaptively）変化させる AdaGrad，RMSProp といったアルゴリズムがあります．

Adam (adaptive momentum) [Kingma-Ba, 2014] はモメンタムに RMSProp の考え方を適用し，さらに改良を施したものと考えられます．

ただし，経験上は Adam であっても途中で初期の学習係数を強制的に変更した方が良い場合が多いです．

Adam のパラメータ更新アルゴリズム (オリジナル)

$0 < \beta_1 < 1$，$0 < \beta_2 < 1$：ハイパーパラメータ（例えば，$\beta_1 = 0.9$，$\beta_2 = 0.999$）

$\mathbf{s}$, $\mathbf{r}$：パラメータ θ と同じ型の配列で初期値は 0

$$\mathbf{s} \leftarrow \beta_1 \mathbf{s} + (1 - \beta_1) \frac{\partial E}{\partial \theta}$$

$$\mathbf{r} \leftarrow \beta_2 \mathbf{r} + (1 - \beta_2) \frac{\partial E}{\partial \theta} * \frac{\partial E}{\partial \theta}$$

$$\tilde{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \beta_1^t}$$

$$\tilde{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \beta_2^t} \qquad (t \text{ はそれまでの更新回数})$$

$$\theta \leftarrow \theta - \alpha \frac{\tilde{\mathbf{s}}}{\sqrt{\tilde{\mathbf{r}}} + \epsilon}$$

ややこしいので意味付けはしづらい

実装しやすいように書き換えた更新アルゴリズム

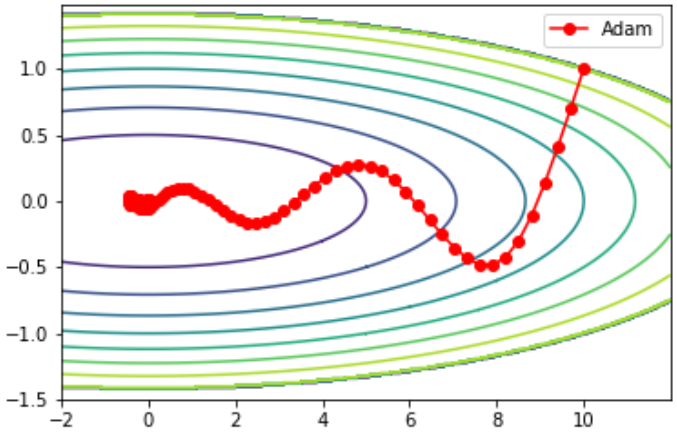
λ の初期値は 0.001 などとします．

$$\mathbf{s} \leftarrow \beta_1 \mathbf{s} + (1-\beta_1)\frac{\partial E}{\partial \theta}$$

$$\mathbf{r} \leftarrow \beta_2 \mathbf{r} + (1-\beta_2)\frac{\partial E}{\partial \theta} * \frac{\partial E}{\partial \theta}$$

$$\lambda \leftarrow \lambda \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \quad (t \text{ はそれまでの更新回数})$$

$$\theta \leftarrow \theta - \lambda \frac{\mathbf{s}}{\sqrt{\mathbf{r}} + \epsilon}$$



時間発展の様子

6.3　最適化法の実装に関する演習

使用するノートブック:

1. 2-5-1_最適化法.ipynb
2. 2-5-2_最適化法の NN への適用.ipynb

ノートブック 1 は最適化手法のクラスを定義し，2 変数 2 次関数の最小値を求めます．ノートブック 2 はそれを多層のニューラルネットワークに適用します．

ノートブック 1 には最後に演習問題があるのでそれを解いてください．また，以下のことに注意して各ノートブックを実行してください．

- パラメータの保管を配列を要素とする辞書形式で行っていること：NN への適用のため
- NN へ適用した場合の振る舞いの違い

7　重みパラメータの初期値の取り方の工夫

ニューラルネットワークは多くの学習パラメータを持ち，それらの学習における初期値は予めランダムに生成されなければなりません．例えばアファイン変換

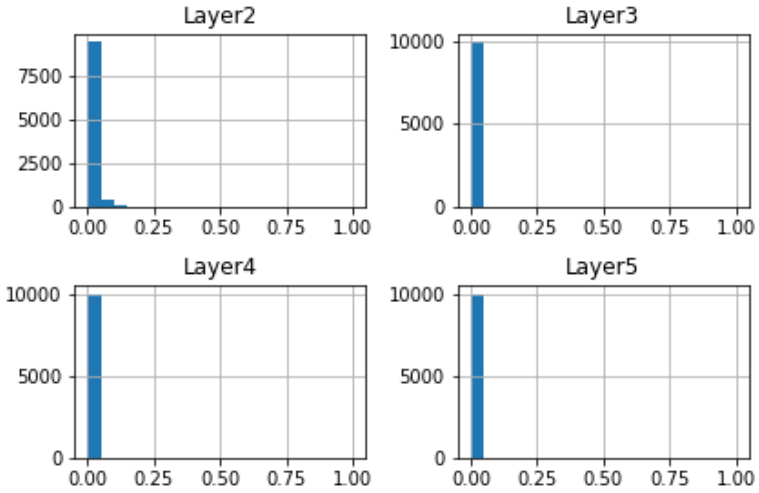
$$\mathbf{y} = \mathbf{x}W + \mathbf{b}$$

における行列 W とバイアスベクトル $\mathbf{b}$ の場合，通常，W の各成分は平均 0 分散 σ^2 の正規分布に従ってとり，$\mathbf{b}$ の各成分は 0 とします．

しかし，階層がある程度深くなる（およそ 5 層以上）と分散 σ^2 を適当にとるだけでは，深い層におけるデータの値が発散してしまったり，0 に近くなってしまったりして（勾配も同様に収束したり発散）学習がうまく進まなくなります．これを回避するためには，重みパラメータの分散をネットワークの構造に応じて適切にとる必要があります．

7.1　中間層におけるデータの収束・発散

例えば，次に行う演習のように，5 層のネットワークにおいて，10000 件のデータに対する第 2 層から第 5 層までの入力値を棒グラフで表したとき，



となったとします．これは最初に発生させた重みパラメータの分布の標準偏差が小さすぎるために層が進むにつれて値が 0 に収束してしまっている状態です．逆に，パラメータの標準偏差を大きくしすぎると値が発散してしまいます．

どのように標準偏差をとるとちょうど良くなるでしょうか？

7.2　パラメータの初期化戦略

He の初期値

発見者の名前をとって He の初期値と呼ばれるとり方では，活性化関数として ReLU を用いる場合，前の層のニューロンの数を N_{l-1} とするとき，第 l 層の W の分散を

$$\sigma_l^2 = \frac{2}{N_{l-1}}$$

とします．

注：畳み込み層の場合は N_{l-1} の代わりに，前の層のうち第 l 層の一つのニューロンへの入力に使うニューロンの数とします．

Glorot-Bengio の初期値

Glorot-Bengio の初期値と呼ばれるとり方では，活性化関数としてシグモイド関数を用いる場合，第 l 層の分散を $\sigma_l^2 = \dfrac{2}{N_{l-1} + N_l}$ とします．

なぜこのようにとると良いのか？

重みパラメータ W_l が平均 0，分散 V_l の分布に従うとき，第 l 層によってデータの分散が

約 $N_{l-1} \times V_l$ 倍される

ので，

$$V_l \coloneqq \frac{1}{N_{l-1}}$$

ととれば分散が変わらずに次の層に伝わるから，と言えます．

7.3　パラメータの初期値の取り方に関する演習

使用するノートブック:　2-6_初期値の取り方.ipynb

このノートブックでは，層を伝わるとともにデータがどのように伝わるのか実験を行い，これまでに作成したニューラルネットワークのクラスに He の初期値を組み込みます．

以下のことに注意して各ノートブックを実行してください．

- Pandas によるヒストグラムの作成
- 層を伝わるとともにデータがどのように伝わるのか
- データの伝わり方と学習の関係
- 初期化戦略の実装方法

8　バッチ正規化

バッチ正規化（Batch normalization）[Ioffe, S., Szegedy, C., 2015] **はミニバッチに対して各層のニューロンごとにデータを正規化することによって勾配収束・爆発を防ぐ方法です．前節では重みパラメータの初期値の取り方を工夫しましたが，バッチ正規化では層の出力を正規化します．バッチ正規化は比較的最近提案された手法にも関わらず，現在広く用いられています．**

8.1　データセットの正規化

実はこれまでにも何度も使っていますが，データセットの正規化について改めてまとめておきましょう．

あるデータセット

$$X = [x_0, x_1, x_2, \cdots, x_{N-1}]$$

があるとき，平均 μ，分散 $V = \sigma^2$ は

$$\mu = \frac{1}{N}\sum_{n=0}^{N-1} x_n$$

$$V = \frac{1}{N}\sum_{n=0}^{N-1} (x_n - \mu)^2$$

$$\sigma = \sqrt{V}$$

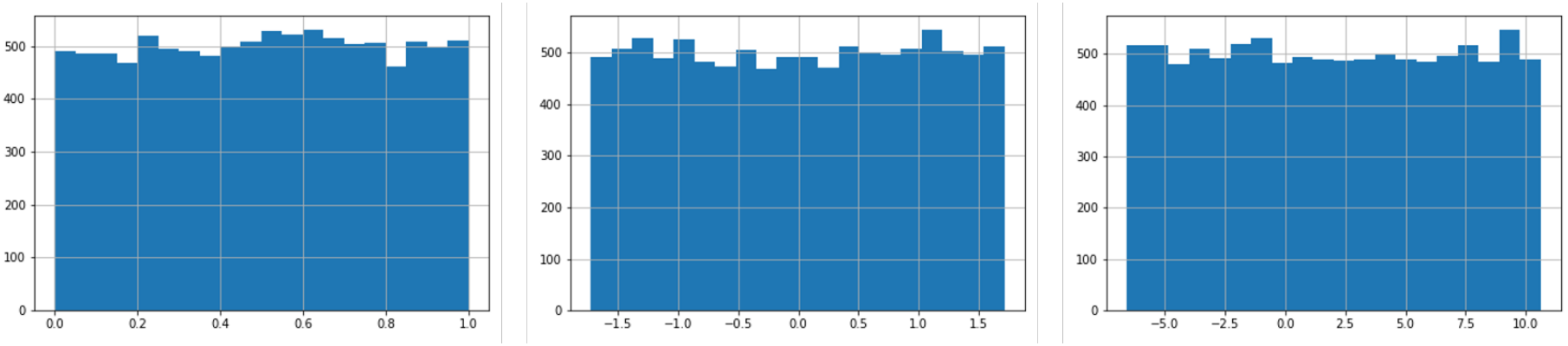
と計算できます．これを平均 0，分散 1 のデータセットに変換する操作は以下のものになります．

$$\tilde{X} = \frac{X - \mu}{\sigma}$$

さらに平均 β，分散 γ^2 に変換したい場合は，

$$\tilde{\tilde{X}} = \gamma * \tilde{X} + \beta$$

とすればできます．



一様分布に従う 10,000 件のサンプルデータ

左：オリジナルの $[0, 1]$ 区間のもの，　中：正規化したもの，　右：平均 2，分散 $5^2 = 25$ に変換したもの

8.2　バッチ正規化のアルゴリズム

X： 対象となる層の出力，$N \times K$ 型（ミニバッチのデータ数 N，層のニューロン数 K）

β_k：$(K,)$ 型配列，新たなデータセットの平均値，初期値の成分は 0

γ_k：$(K,)$ 型配列，新たなデータセットの標準偏差，初期値の成分は 1

順伝播の計算

$$
\begin{aligned}
\mu_k &= \frac{1}{N}\sum_{n=0}^{N-1} x_{nk} \\
v_k &= \frac{1}{N}\sum_{n=0}^{N-1} (x_{nk} - \mu_k)^2 \\
\tilde{x}_{nk} &= \frac{x_{nk} - \mu_k}{\sqrt{v_k + \epsilon}} \\
y_{nk} &= \gamma_k \, \tilde{x}_{kn} + \beta_k
\end{aligned}
$$

ただし，ϵ は発散を防ぐための微小量で $e = 10^{-8}$ などとします．

この変換を NumPy 風に書くと

$$
\begin{aligned}
\mu &= \mathrm{mean}(X, \mathrm{axis} = 0) \\
\mathbf{v} &= \mathrm{mean}\big((X - \mu)^2, \mathrm{axis} = 0\big) \\
\tilde{X} &= \frac{X - \mu}{\sqrt{\mathbf{v} + \epsilon}} \\
Y &= \gamma * \tilde{X} + \beta
\end{aligned}
$$

となります．β，γ は K 次元のベクトルであることに注意します．

バッチ正規化は評価するデータが少ないと正規化がうまく働きません（例えばデータ数が１のとき，$Y = \beta$ となってしまいます）．特に予測時はデータ数 1 となるのは普通のことです．この問題については，訓練時の平均や分散のデータをテスト時に使用することで対処します．

逆伝播

誤差逆伝播の計算はチェインルールや計算グラフを用いて行うことができますが．非常にややこしいので，結果のみ記すと以下のようになります．

$$\frac{\partial E}{\partial \beta} = \mathrm{sum}\left(\frac{\partial E}{\partial Y},\ \mathrm{axis}=0\right), \quad \frac{\partial E}{\partial \gamma} = \mathrm{sum}\left(\tilde{X} * \frac{\partial E}{\partial Y},\ \mathrm{axis}=0\right)$$

および

$$\mathbf{c} = \mathrm{mean}\left(\frac{\partial E}{\partial Y},\ \mathrm{axis}=0\right), \quad \mathbf{d} = \mathrm{mean}\left(\tilde{X} * \frac{\partial E}{\partial Y},\ \mathrm{axis}=0\right)$$

$$\frac{\partial E}{\partial X} = \frac{\gamma}{\sigma}\left(\frac{\partial E}{\partial Y} - \mathbf{c} - \tilde{X} * \mathbf{d}\right)$$

となります．ここで，$\mathbf{c}$ ，$\mathbf{d}$ と $\tilde{X}$，Y では配列の型が違いますが，ブロードキャストで計算できます．

これは誤差 E の Y に関する勾配 $\frac{\partial E}{\partial Y}$ を用いて $\frac{\partial E}{\partial X}$ などを計算する式になっています．

8.3　バッチ正規化に関する演習

使用するノートブック:　2-7_バッチ正規化.ipynb

このノートブックでは，データセットの正規化に関する演習を行い，バッチ正規化を実装します．

以下のことに注意してノートブックを実行してください．

- NumPy によるデータセットの正規化
- バッチ正規化では何をやっているのか
- バッチ正規化は層の一つであること
- 学習時とテスト時の違いについてどのように実装しているのか

8.4　深層モデルのための最適化に関する演習

使用するノートブック:　2-8_深層モデルのための最適化.ipynb

このノートブックではこれまでに実装してきた最適化手法，パラメータの初期値戦略，バッチ正規化をニューラルネットワークに組み込み，実験を行います．

以下のことに注意して各ノートブックを実行してください．

- それぞれの手法の効果
- どのくらいまで深くできるか

9　深層モデルのための正則化

ニューラルネットワークのモデルを深くしていくと，過学習が起こりやすくなります．過学習を回避するための手法としては以下のようなものがあります．

- パラメータノルムペナルティー：重みパラメータのノルムによる正則化
- データ集合の拡張：入力データを模倣したデータを作ってデータを増やす
- 出力値に対するノイズ注入：出力値を丸めて頑健性を上げる
- 半教師あり学習：教師なしデータを学習に利用する
- 早期終了：評価データに対する精度の向上具合によって学習をやめる
- アンサンブル学習：複数のモデルによる予測の平均をとる
- ドロップアウト：ニューロンの出力を一定の確率で次の層で用いない

本節ではこれらのうち，パラメータノルムペナルティーとドロップアウトについて解説を行います．なお，データ集合の拡張については節を改めて解説します．

9.1　パラメータノルムペナルティー

教師あり機械学習において，モデルの正則化は多くの場合，パラメータに対する何らかのノルム Ω を用いて，問題の本来のコスト関数 J を

$$\tilde{J} = J + \alpha\Omega$$

（ただし，$\alpha > 0$）と変更することによって実現されます．

ニューラルネットワークの場合，L 層のニューラルネットワークにおける第 l 層の係数行列 $W^{(l)}$，バイアスベクトル $\mathbf{b}^{(l)}$ に対して，$W^{(l)}$ のみを Ω の計算に用いるのが一般的です．つまり，

$$\tilde{J} = J + \alpha\Omega\left(W^{(1)}, \cdots, W^{(L)}\right)$$

とします．

α を層ごとに変えることも可能ですが，ハイパーパラメータの探索範囲が増えてしまうので，モデル全体で共通の α を使う方が一般的です．

L2 パラメータ正則化

係数行列の L2 ノルム

$$\Omega = \sum_{l=1}^{L} ||W^{(l)}||^2 = \sum_{l=1}^{L} W^{(l)} \left(W^{(l)}\right)^T = \mathrm{sum}(W^{(l)} * W^{(l)})$$

を用いて行う正則化

$$\tilde{J} = J + \alpha \sum_{l=1}^{L} ||W^{(l)}||^2$$

を L2 パラメータ正則化（あるいは単に L2 正則化）といいます．他分野では，L2 正則化のことをリッジ正則化，チホノフ正則化と呼ぶこともあります．

Ω の $W^{(l)}$ に関する偏微分は

$$\frac{\partial \Omega}{\partial W^{(l)}} = 2\alpha W^{(l)}$$

と簡単に計算できます．

L1 パラメータ正則化

係数行列の L1 ノルム

$$\Omega = \sum_{l=1}^{L} \left| W^{(l)} \right|$$

（ただし，$\left| W^{(l)} \right|$ は行列 $W^{(l)}$ のすべての成分の絶対値の和）を用いて行う正則化

$$\tilde{J} = J + \alpha \sum_{l=1}^{L} \left| W^{(l)} \right|$$

を L1 パラメータ正則化（あるいは単に L1 正則化）といいます．L1 正則化を用いた線形回帰は LASSO (Least Absolute Shrinkage and Selection Operator) とも呼ばれます．

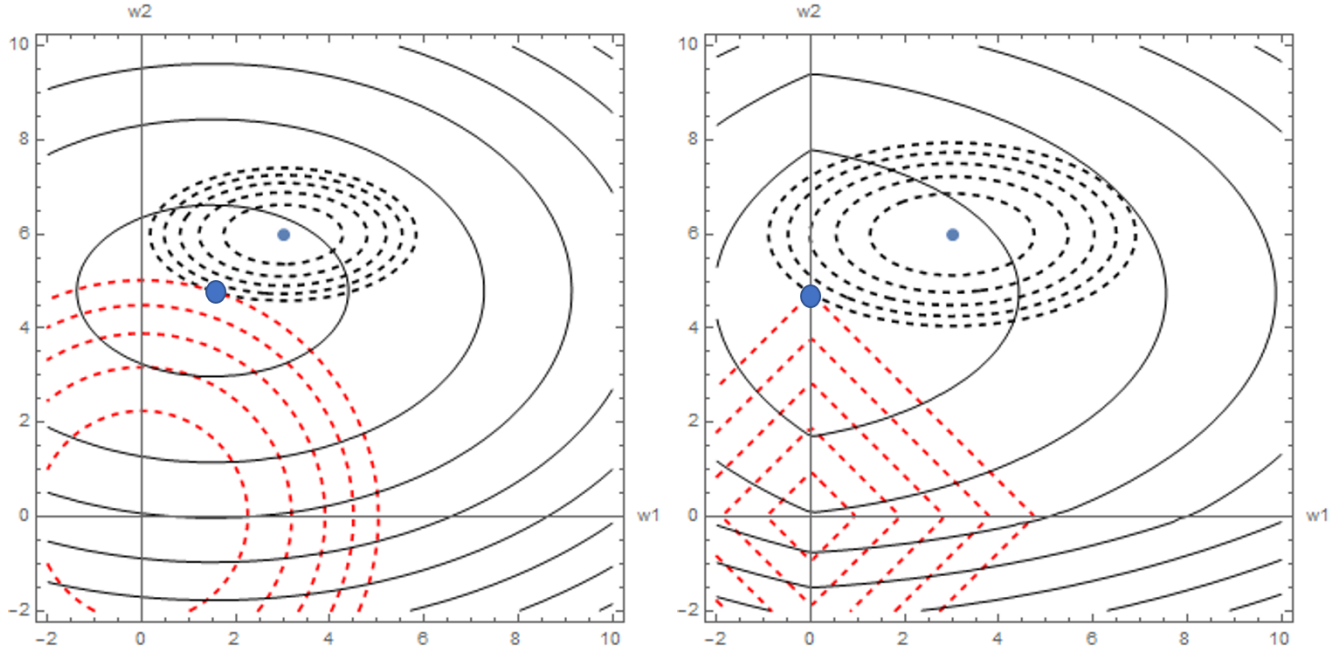
Ω の $W^{(l)}$ に関する偏微分は

$$\frac{\partial \Omega}{\partial W^{(l)}} = \alpha \ \mathrm{sign}\ W^{(l)}$$

（sign x は x の符号をとる関数，x が行列のときは成分それぞれに対して適用する）と計算できます．

L1 正則化のスパース効果

ある値の集合がスパースであるとは，値のうちの多くが 0 であることです．L2 正則化と L1 正則化を較べると，L1 正則化にはパラメータをスパースにする効果があります．下図は目的関数 $f(w_1, w_2) = (w_1 - 3)^2 + 4(w_2 - 6)^2$ の最小化を行うモデルに対して，L2 正則化と L1 正則化を行ったものです．L1 正則化の解は $w_1 = 0$ となっています．



L2 正則化と L1 正則化

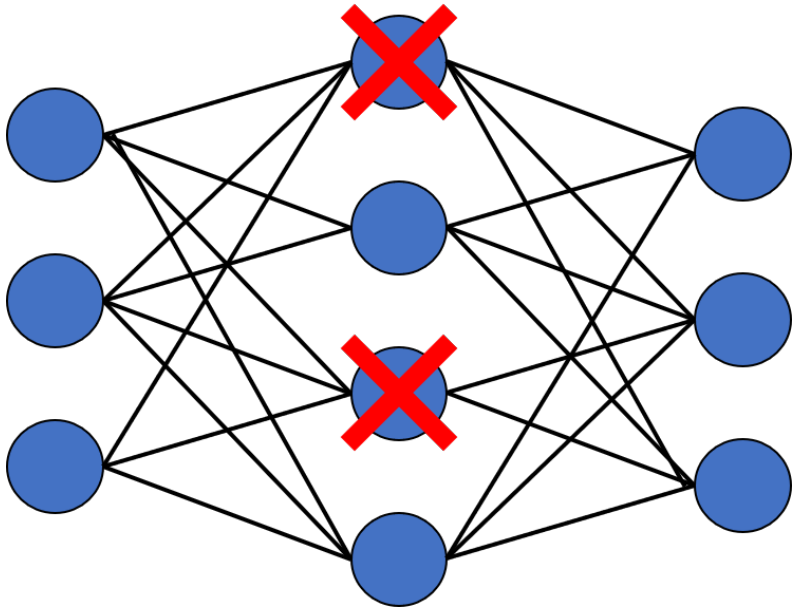
左図：正則化項として $w_1^2 + w_2^2$ を付け加えた場合，　右図：$10(|w_1| + |w_2|)$ を付け加えた場合

黒破線：f の等高線，　赤破線：正則化項の等高線，　黒実線：正則化された関数の等高線

小さい点：元の最小点，　大きい点：正則化項を加えた場合の最小点

9.2　ドロップアウト

ドロップアウト [Srivastava, N. et al., 2014] **は，ニューラルネットワークの学習を改良する一手法であり，学習時にはドロップアウトを適用する各層において，データ毎にニューロンを与えられた割合** p **で消去（マスク）し，推測時にはマスクはかけない代わりにドロップアウトを適用していた各層の出力値に** $1-p$ **を乗じるという手法です．データごとにニューロンの消去の仕方を変えることでアンサンブル学習を模倣していると考えられ，汎化性能の向上のために広く用いられています．**



ドロップアウトのアルゴリズム

p：出力値を 0 にする確率，ハイパーパラメータの一つ

入力：層の出力 $X_{N\times K}$ （N：バッチサイズ，K：ニューロンの数）

順伝播

マスク M： X と同じ型の配列で各成分が独立に，確率 p で 0，確率 $1-p$ で 1 をとる行列

$$Y=\begin{cases}X*M & (\text{学習時})\\ X*(1-p) & (\text{予測時})\end{cases}$$

逆伝搬

$$\frac{\partial E}{\partial X}=\frac{\partial E}{\partial Y}*\frac{\partial Y}{\partial X}=\frac{\partial E}{\partial Y}*M$$

9.3　ドロップアウトに関する演習

使用するノートブック:　2-9_ドロップアウト.ipynb

このノートブックでは，ドロップアウトに関する演習を行います．

以下のことに注意してノートブックを実行してください．

- マスク M の実装の仕方：各成分は確率 p で 0 をとり，$1-p$ で 1 をとる
- 逆伝播の計算
- 予測時の計算

9.4　正則化に関する総合演習

使用するノートブック: 2-10_正則化に関する演習.ipynb

このノートブックではこれまでのまとめとして，バッチ正規化，ドロップアウト，L2 正則化，L1 正則化を組み込んだニューラルネットワークに関する演習を行います．

以下のことに注意してノートブックを実行してください．

1. どのようにネットワークを設計すると精度が上がるのか（これは難問）
2. 学習係数を途中で小さくすると変えるとなぜ精度が上がるのか

2 について：始めは荒い網目で大局的に見てコスト関数が小さくなるように進み，最後に細かい網目で局所的に見ることで良いパラメータを探すことに相当すると考えられます．