DeepLearning 入門

〜『ゼロから作る Deep Learning』を読んで〜

著：博ノ助

目次

1. 本書で用いる表現について

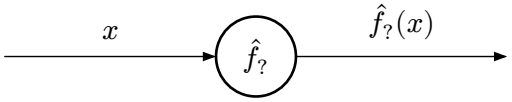
x X	スカラー
$\boldsymbol{x}$	行ベクトル（小文字・太文字） $1 \times N$ の行列と同一視する
x_j	ベクトル $\boldsymbol{x}$ の j 成分（特に指定がない場合）
$\mathbf{1}_n$	要素数 n で成分が全て 1 の列ベクトル
$\boldsymbol{X}$	行列（大文字・太文字）
x_{ij} $x_j^{(i)}$	行列 $\boldsymbol{X}$ の i, j 成分（特に指定がない場合）
$[\boldsymbol{X}]_{ij}$	行列 $\boldsymbol{X}$ の i, j 成分
$\boldsymbol{X}^\top$	行列 $\boldsymbol{X}$ の転置
$[\boldsymbol{X}]_{i*}$ $\boldsymbol{x}^{(i)}$	行列 $\boldsymbol{X}$ の i 行を表す行ベクトル
$[\boldsymbol{X}]_{*j}$	行列 $\boldsymbol{X}$ の j 列を表す列ベクトル
$\boldsymbol{x} \cdot \boldsymbol{y}$	ベクトル $\boldsymbol{x}$ とベクトル $\boldsymbol{y}$ の標準内積
$\boldsymbol{x} \odot \boldsymbol{y}$	ベクトル $\boldsymbol{x}$ とベクトル $\boldsymbol{y}$ のアダマール積
$\frac{\boldsymbol{x}}{\boldsymbol{y}}$	ベクトル $\boldsymbol{x}$ とベクトル $\boldsymbol{y}$ のアダマール除算
$\boldsymbol{x}^{\circ n}$	ベクトル $\boldsymbol{x}$ 自身とのアダマール積を n 回行ったもの 各要素を n 乗したものと一致する
$\boldsymbol{X} \cdot \boldsymbol{Y}$	行列 $\boldsymbol{X}$ と行列 $\boldsymbol{Y}$ のドット積
$\boldsymbol{X} \odot \boldsymbol{Y}$	行列 $\boldsymbol{X}$ と行列 $\boldsymbol{Y}$ のアダマール積
$\frac{X}{\boldsymbol{Y}}$	行列 $\boldsymbol{X}$ と行列 $\boldsymbol{Y}$ のアダマール除算
$\sqrt{\boldsymbol{X}}$	行列 $\boldsymbol{X}$ の各成分の平方根をとった行列 $\left[\sqrt{\boldsymbol{X}}\right]_{ij} = \sqrt{[\boldsymbol{X}]_{ij}}$
$\frac{\partial}{\partial x}$	スカラー x に関する偏微分作用素
$\frac{\partial}{\partial \boldsymbol{x}}$	$\left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, ..., \frac{\partial}{\partial x_n}\right)$
$\frac{\partial}{\partial \boldsymbol{X}}$	$\left[\frac{\partial}{\partial \boldsymbol{X}}\right]_{ij} = \frac{\partial}{\partial x_{ij}}$
$\mathbb{E}[f(x_1, x_2, ..., x_n)]$	確率変数 $x_1, x_2, ..., x_n$ に依存する関数 f の期待値
$\mathbb{E}_p[f(x_1, x_2, ..., x_n)]$	確率変数 p に依存する関数 f の期待値
$\mathcal{N}(\mu, \sigma^2)$	平均 μ、分散 σ^2 の正規分布
$\mathrm{Bi}(p)$	期待値 p のベルヌーイ分布
$x \sim \mu$	μ が確率分布を表すとき x はその分布に従うことを表す しばしば、分布 μ から x をサンプリングすることを表す

2. 多層パーセプトロン（MLP）

この章では、DeepLearning の基礎である多層パーセプトロン（MLP）について学ぶ。最小単位ニューロンの構造を紹介し、その具体的な利用法と、限界について触れることでのちの章で出現する様々な要素を導入するモチベーションを与える。

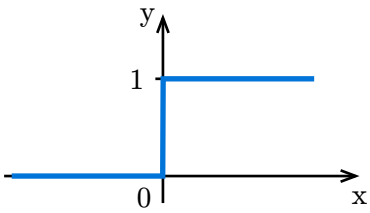
2.1. ニューロンの始まり

ニューロンは生物の神経細胞を模した情報処理の単位である。入力信号を受け取り、それに基づいて出力信号を生成する。



この際、入力の強さによって信号を出力するかどうかを決めることにしてみよう。これは生物的に行われていることであるが、これを数学的に表現するとステップ関数を用いることができる。

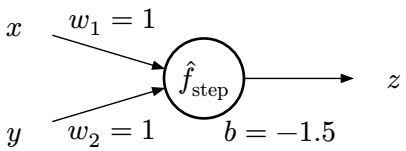
$$f_{\text{step}}(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



この関数では入力 x が 0 以上で発火することになる。しかしこれでは、 0 か 1 しか伝播しないため、帰納的にすべてのニューロンが発火してしまう。そこで、ニューロン同士の結合の強さ w と、発火する閾値を調整する b というパラメータを導入することでニューロンの結合は意味を持つ。また、複数の入力に対してもその結合を考慮することができるように、ニューロンの出力は以下のように定義される。

$$\hat{f}_{\text{step}}(\boldsymbol{x}) =: f_{\text{step}}\left(\sum_i w_i x_i + b\right) = \begin{cases} 1 & \text{if} \quad \sum_i w_i x_i \geq -b \\ 0 & \text{otherwise} \end{cases}$$

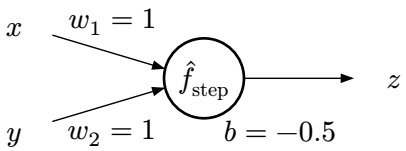
w_i は i 番目の入力 x_i に対する結合強度を表す。このようにしてニューロンは複数の入力を受け取り,それらの結合強度と閾値を考慮して出力を決定することができる。これをパーセプトロンという。このパーセプトロンによる実例を見てみよう。



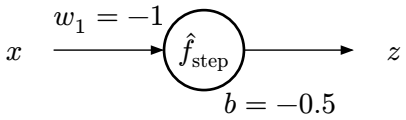
$$z = f_{\text{step}}(x + y - 1.5) = \begin{cases} 1 & \text{if} \quad x + y \geq 1.5 \\ 0 & \text{otherwise} \end{cases}$$

上のパーセプトロンは AND ゲートを模している。 0 か 1 しかとらない入力 x と y に対して、ともに 1 のときのみ出力 z が 1 になり、それ以外のときは 0 になる。このことは上の式からわかるだろう。

次に OR ゲートを模したパーセプトロンを見てみよう。



上のパーセプトロンは OR ゲートを模している。 0 か 1 しかとらない入力 x と y に対して、少なくとも片方が 1 のときのみ出力 z が 1 になり、それ以外のときは 0 になる。同じように入力 x に対して 0 ならば 1 、 1 ならば 0 を出力する NOT ゲートを模したパーセプトロンを実装できる。



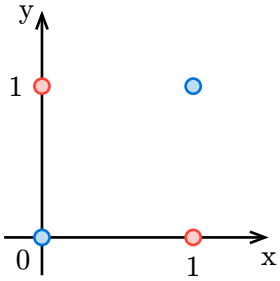
2.2. 多層パーセプトロン

では,XOR ゲートはどうだろうか。XOR ゲートは 0 か 1 しかとらない入力 x と y に対して、片方が 1 のときのみ出力 z が 1 になり、それ以外のときは 0 になる関数である。しかし、XOR ゲートは上のようなパーセプトロンでは実装できない。右下のようなグラフを書いてみる。

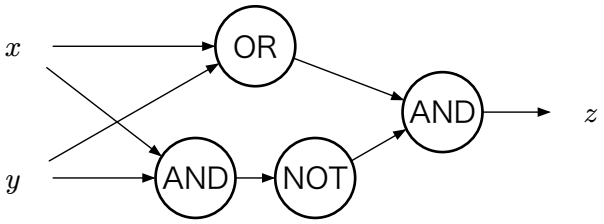
二つの入力を x 軸 y 軸として、出力を色分けした。青色は 0 、赤色は 1 を表している。ニューロンは

$$\hat{f}_{\text{step}}(w_1 x + w_2 y + b) = \begin{cases} 1 & \text{if} \quad w_1 x + w_2 y \geq -b \\ 0 & \text{otherwise} \end{cases}$$

と表されるのであるが、これは直線 $w_1 x + w_2 y + b = 0$ を境界に xy 平面を分けることを意味する。



しかし、グラフを見ればわかるように直線 1 つで青と赤の点を分けることができない。これを、線形分離不可能という。XOR ゲートは線形分離不可能な関数である。 したがって、XOR ゲートは 1 つのニューロンでは実装できない。しかし、XOR ゲートはいくつかの他の論理ゲートを組み合わせることで実装できることが知られている。



このように、XOR ゲートは OR ゲートと AND ゲートと NOT ゲートを組み合わせることで実装できる。複数のパーセプトロンをまるで層を重ねるように組み合わせることで、より複雑な関数を実装することができる。これを多層パーセプトロン（MLP）とよぶ。データの入力を受け付ける層を入力層、出力を表す層を出力層とよぶ。その間にある層を隠れ層とよぶ。入力層と出力層をみればデータの入出力の形だけはわかる。

2.3. 活性化関数

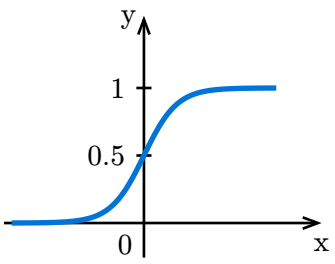
多層パーセプトロンは、複数のニューロンを組み合わせることでより複雑な関数を実装することができる。しかし、上のようなステップ関数 f_{step} では、出力が離散的で値の表現が乏しい。そのため、連続的な値を出力する関数を用いる。これら、ニューロンの出力を決定する関数を活性化関数とよぶ。

また、のちに多層パーセプトロンはいわゆる深層学習に使われるが、その際、微分が大きな役割を果たす。そのため、微分可能であるような関数が現在の活性化関数として用いられている。活性化関数の進化によって、ニューロンの「発火」という性質は薄れていった。一方で、パラメータ w, b は意味を変えつつもそのまま残っており、重み及びバイアスとよばれるようになった。

活性化関数には様々な種類があるが、ここでは代表的なものを紹介する。

2.3.1. シグモイド関数

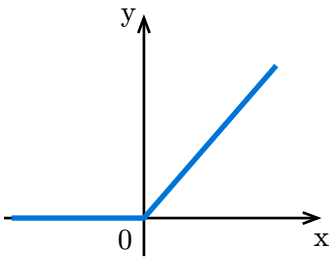
$$f_{\text{sigmoid}}(x) = \frac{1}{1+\exp(-x)}$$



シグモイド関数は、入力 x に対して 0 から 1 の値を出力する微分可能な関数である。これは、ニューロンの出力を確率として解釈することができる。ステップ関数と形が似ているため、ステップ関数の滑らかな近似としてよく用いられていた。しかし、入力の絶対値が十分大きい場合に、微分係数が 0 に近くなってしまうため、学習が進まなくなるという問題[1]がある。

2.3.2. ReLU 関数

$$f_{\text{ReLU}}(x) = \begin{cases} x & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases} = \max(0, x)$$



ReLU 関数は、入力 x に対して 0 以上の値を出力する微分可能な関数である。これは、入力が 0 以上のときはそのまま出力し、 0 未満のときは 0 を出力する。シグモイド関数と比べて、勾配消失問題が起こりにくいという利点があるため、現在ではデフォルトで用いられている。面白いことに、ReLU 関数はステップ関数を積分したものになっている。

2.3.3. Softmax 関数

$$f_{\text{softmax}}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

[1] これを勾配消失問題という。

Softmax 関数は、他の活性化関数とは異なり入力全体のベクトル $\boldsymbol{x}$ を用いる. i 番目の要素 x_i に対して、 x_i の全体に占める割合に似たものを出力する.出力は 0 以上の値を持ち、全体の和は 1 になる。確率分布を表すために用いられる。

Softmax 関数を実装する際の注意点として、入力 x_i の値が大きいときに、 $\exp(x_i)$ がオーバーフローしてしまうことがある。これを防ぐために、全体の最大値を引いてから計算する。値が小さくても $\exp(x_i)$ は 0 に近づくだけなので問題ない。入力を定数分だけ引いても出力は変わらないので、この操作で出力は変化しない。

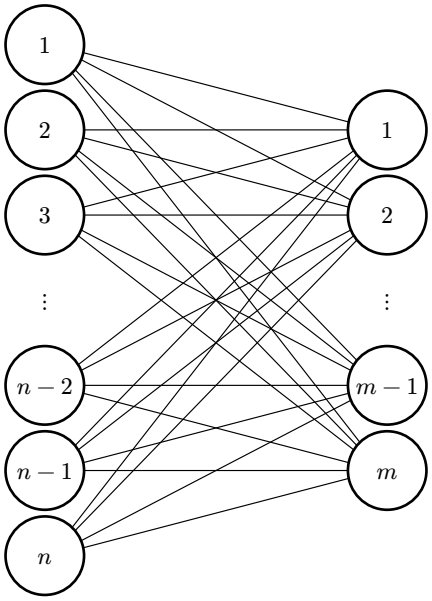
$$
\begin{aligned}
&C = \text{const.} \\
&\frac{\exp(x_i - C)}{\sum_j \exp\bigl(x_j - C\bigr)} = \frac{e^{-C}\exp(x_i)}{\sum_j e^{-C}\exp\bigl(x_j\bigr)} \\
&\qquad = \frac{e^{-C}\exp(x_i)}{e^{-C}\sum_j \exp\bigl(x_j\bigr)} \\
&\qquad = \frac{\exp(x_i)}{\sum_j \exp\bigl(x_j\bigr)}
\end{aligned}
$$

2.4. 行列表現と Affine 層

ここでは、層構造をもつ多層パーセプトロンについて、効率的な計算をおこなうために行列計算に帰着できることを示す。

2.4.1. Affine 層

Affine 層は、入力 x に対して、重み W とバイアス b を用いて出力 y を計算する層である。活性化関数については一度目を瞑ることにする。最も基本的な形は以下の図で示されるようなものである。



いくつかのニューロンが並列に配置されており、これを層あるいはレイヤという。レイヤ内のニューロンの数を層の大きさとよび、上図ではサイズ n のレイヤとサイズ m のレイヤが結合されている。また、上図では省略されているが、各入力は個別の重みとバイアスを持つ。

注意すべきなのが、各ニューロンの各入力について重みがあるという点である。例えば、右側レイヤの 1 番目のニューロンは左側レイヤの 1 番目のニューロンからの入力に対して重み $w_{1\leftarrow 1}$ を持つ。加えて、右側レイヤの 2 番目のニューロンは左側レイヤの 1 番目のニューロンからの入力に先ほどとは異なる重み $w_{2\leftarrow 1}$ を持つのである。ただし、バイアスはニューロンごとに 1 つしか持たない。

今現在、右側レイヤに注目する。左側レイヤの i 番目のニューロンの出力を x_i 、右側レイヤの j 番目がもつ x_i に対する重みを $w_{j\leftarrow i}$ と書くことにする。右側レイヤの j 番目のニューロンのバイアスを b_j 、出力を y_j とすると、以下のように表すことができる。

$$y_j = \sum_{1\leq i\leq n} w_{j\leftarrow i} x_i + b_j$$

ここで、以下のような重み行列 $\boldsymbol{W}$ とバイアスベクトル $\boldsymbol{b}$ を定義する。

$$\begin{aligned}\boldsymbol{W} &:= \begin{pmatrix} w_{1\leftarrow 1} & w_{2\leftarrow 1} & \cdots & w_{m\leftarrow 1} \\ w_{1\leftarrow 2} & w_{2\leftarrow 2} & \cdots & w_{m\leftarrow 2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1\leftarrow n} & w_{2\leftarrow n} & \cdots & w_{m\leftarrow n} \end{pmatrix} \quad \in \mathbb{M}^{n\times m} \\ \boldsymbol{b} &:= \begin{pmatrix} b_1 & b_2 & \cdots & b_m \end{pmatrix}\end{aligned}$$

$\boldsymbol{W}$ の i,j 成分 w_{ij} は $w_{j\leftarrow i}$ に一致している.この行列とベクトルを用いると、左側レイヤの出力 $\boldsymbol{x}$ に対して、右側レイヤの出力 $\boldsymbol{y}$ は以下のように表すことができる。

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

簡単な行列計算で済むのである.

2.4.2. バッチ付き Affine 層

機械学習では複数のデータを同時に扱うことで計算を効率化することが一般的である。ここでいう、データとは $\boldsymbol{x}$ や $\boldsymbol{y}$ を 1 単位としている。時に、何億というデータを扱うこともあるため、コンピュータの性能が許す限り、一度に多くのデータを扱いたいのである。これをバッチ処理とよび、いくつかのデータをまとめたものをバッチとよぶ。また、一度に扱うデータの数をバッチサイズとよび、 $B\in\mathbb{N}$ と書くことにする。

Affine 層では、単に入出力をベクトルから行列に拡張するだけでバッチ処理を実現できる。次の $\boldsymbol{X},\boldsymbol{Y}$ に入出力を改める。またバイアスも $\boldsymbol{B}$ に改める。ただし、バッチ内の i 番目のデータを $\boldsymbol{x}_i,\boldsymbol{y}_i$ とする。

$$\begin{aligned}\boldsymbol{X} &:= \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_B \end{pmatrix}^\top && \in \mathbb{M}^{B\times n} \\ \boldsymbol{Y} &:= \begin{pmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_B \end{pmatrix}^\top && \in \mathbb{M}^{B\times m} \\ \boldsymbol{B} &:= \begin{pmatrix} \boldsymbol{b} & \boldsymbol{b} & \cdots & \boldsymbol{b} \end{pmatrix}^\top && \in \mathbb{M}^{B\times m} \\ &= \boldsymbol{1}_B \cdot \boldsymbol{b}\end{aligned}$$

$\mathbf{1}_n$ は成分がすべて 1 で n 次元列ベクトルである。このベクトルと、任意の行ベクトル $\boldsymbol{b}$ のドット積は、$\boldsymbol{b}$ の成分を n 行分だけコピーした行列になる。これをベクトルのブロードキャストとよぶ。さて、これは先ほどと同様に行列の積で計算できる。

$$\boldsymbol{Y} = \boldsymbol{X} \cdot \boldsymbol{W} + \boldsymbol{B}$$

これは以下のように正当化される。

$$\begin{aligned}
\boldsymbol{X} \cdot \boldsymbol{W} + \boldsymbol{B} &= (\boldsymbol{x}_1 \ \ \boldsymbol{x}_2 \ \ \ldots \ \ \boldsymbol{x}_B)^\top \cdot \boldsymbol{W} + \boldsymbol{B} \\
&= (\boldsymbol{x}_1 \cdot \boldsymbol{W} \ \ \boldsymbol{x}_2 \cdot \boldsymbol{W} \ \ \ldots \ \ \boldsymbol{x}_B \cdot \boldsymbol{W})^\top + (\boldsymbol{b} \ \ \boldsymbol{b} \ \ \ldots \ \ \boldsymbol{b})^\top \\
&= (\boldsymbol{x}_1 \cdot \boldsymbol{W} + \boldsymbol{b} \ \ \boldsymbol{x}_2 \cdot \boldsymbol{W} + \boldsymbol{b} \ \ \ldots \ \ \boldsymbol{x}_B \cdot \boldsymbol{W} + \boldsymbol{b})^\top \\
&= (\boldsymbol{y}_1 \ \ \boldsymbol{y}_2 \ \ \ldots \ \ \boldsymbol{y}_B)^\top \\
&= \boldsymbol{Y}
\end{aligned}$$

2.5. 行列表現での活性化関数の扱い

活性化関数は Affine 層とは別と考えることができる。実際の深層学習フレームワークでは一緒に利用できるように実装されているが、ここでは別々に考えることにする。ReLU やシグモイドなどの関数は、行列の個々の要素に対して適用すれば良い。

$$[\boldsymbol{Y}]_{ij} = f_{\mathrm{ReLU}}\big([\boldsymbol{X}]_{ij}\big)$$
$$[\boldsymbol{Y}]_{ij} = f_{\mathrm{sigmoid}}\big([\boldsymbol{X}]_{ij}\big)$$

Softmax 関数などのデータ全体を用いる関数は、バッチ内のデータごとに適用すれば良い。行列表現では行ごとということになる。

$$[\boldsymbol{Y}]_{ij} = \frac{\exp\big([\boldsymbol{X}]_{ij}\big)}{\sum_k \exp([\boldsymbol{X}]_{ik})}$$

3. ニューラルネットワーク

多層パーセプトロンを実装するだけでは、意図した出力を得ることはできない。適正なパラメータを与えなければならない。しかし、人間が正確なパラメータを与えることはおおよその場合、不可能である。このパラメータを自動的に調整する方法が必要である。深層学習とはつまるところ、パラメータを自動的に調整することである。

多層パーセプトロンに与えるデータを $\boldsymbol{x}$ 、それを与えた場合に出力されるべきデータを $\boldsymbol{t}$ 、実際に出力されたデータを $\boldsymbol{y}$ とする。 $\boldsymbol{t}$ と $\boldsymbol{y}$ は大きさ C のベクトルとする。

3.1. 損失関数

損失関数 $\mathcal{L}$ は、実際に出力されるデータ $\boldsymbol{t}$ と与えられたデータ $\boldsymbol{y}$ の差を表す関数である。これは、パラメータを調整するための指標となる。多層パーセプトロンは数学的には $\boldsymbol{x}$ だけでなく、用いるパラメータ $\boldsymbol{p}$ すべてを引数にとる多変数関数とみなせる。なので、その出力を引数に持つ、損失関数もそれらを引数に持つ多変数関数とみなせる。ここで、 $\boldsymbol{t}$ は決められた定数とみなす。

$$\mathcal{L} : \mathbb{R}^{|\boldsymbol{x}|} \times \mathbb{R}^{|\boldsymbol{p}|} \to \mathbb{R}$$

いくつかの損失関数があるが、ここでは代表的なものを紹介する。

3.1.1. 平均二乗誤差

平均二乗誤差は、 $\boldsymbol{y}$ と $\boldsymbol{t}$ の個々の差を二乗して足し合せて平均をとったものである。これは、出力の値が連続的な場合に用いられる。

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{C}||\boldsymbol{y} - \boldsymbol{t}||^2$$

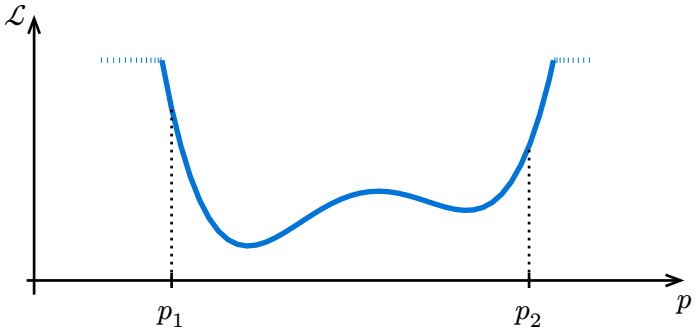
3.1.2. クロスエントロピー誤差

クロスエントロピー誤差は、 $\boldsymbol{y}$ と $\boldsymbol{t}$ を確率分布とみなして、その確率分布の誤差を表すものである。分類問題において使われることが多い。Softmax 関数と組み合わせて用いることが多い。

$$\mathcal{L}_{\mathrm{CrossEntropy}} = -\sum_{1 \leq j \leq C} t_j \log(y_j)$$

3.2. 確率的勾配降下法（SGD）

損失を最小化することが、深層学習の目標である。あるパラメータ p を更新することを考えよう。損失関数 $\mathcal{L}$ は先述した通り, p を引数に持つ。 p についての $\mathcal{L}$ のグラフを書くと以下のようになる。



実際は p に対する $\mathcal{L}$ の形はわからない。しかし、 p での傾きを知ることはできる。これは後述する。 p_1 のように傾きが負のときは、 p を増やすことで損失が減る。 p_2 のように傾きが正のときは、 p を減らすことで損失が減る。したがって、 p を更新する際には、傾きに応じて p を増減させれば良い。これを勾配降下法という。つまり、 p をの更新は以下のように行う。

$$p \leftarrow p - \eta \frac{\partial \mathcal{L}}{\partial p}$$

ここで、 η は学習率とよばれる定数である。これは、 p の更新幅を決めるものである。 η が大きすぎると、最適なパラメータを通り過ぎてしまうことがある。逆に小さすぎると、最適なパラメータにたどり着くまでに時間がかかる。この更新方法を確率的勾配降下法（SGD）とよぶ。

現在、パラメータの更新には SGD に様々な工夫を加えたものが用いられている。しかし、偏微分を考えるという部分は変わらない。

3.3. 誤差逆伝播法

さて、SGD を用いるためには、 $\mathcal{L}$ の偏微分を計算する必要がある。 $\mathcal{L}$ は多層パーセプトロンの出力を引数に持つ多変数関数である。したがって、 $\mathcal{L}$ の偏微分は連鎖律を用いて計算できる。

3.3.1. 数値微分

連鎖律を用いる以外には、数値微分を用いる方法がある。本書では、連鎖律を用いることを前提としているので、詳しくは述べないが紹介だけしておく。数値微分は、 $\mathcal{L}$ の入力を少しだけ変えて、出力の変化をみることで偏微分を求める方法である。 p に関しての偏微分は以下のように表される。そもそも、偏微分の定義は以下のようなものであった。

$$\frac{\partial \mathcal{L}}{\partial p} = \lim_{\delta \to 0} \frac{\mathcal{L}(p+\delta) - \mathcal{L}(p)}{\delta}$$

これを簡単に求められない、つまり四則演算を用いて計算することができないのは δ を 0 に近づけるためである。そこで、 δ を極めて小さな値にし、

$$\frac{\mathcal{L}(p+\delta)-\mathcal{L}(p)}{\delta}$$

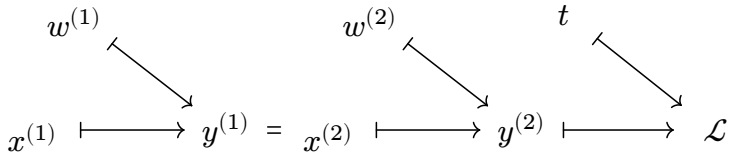
を計算することで、近似的に偏微分を求めることができる。このように、δ を用いて偏微分を近似的に求める方法を数値微分とよぶ。上の式は、δ が正なら前方差分、δ が負なら後方差分とよばれる。しかし、求めたいのは p における偏微分であるのでこれらの計算は誤差が大きい。したがって、以下のように p を中心にした差分を計算することが多い。これを中心差分とよぶ。中心差分は前方差分や後方差分よりも精度がいいことは解析的に示すことができる。

$$\frac{\mathcal{L}(p+\delta)-\mathcal{L}(p-\delta)}{2\delta}$$

すべてのパラメータに対して、δ を用いて偏微分を近似的に求めることができればいいが、パラメータの数は膨大であるため、計算量が膨大になってしまう。また、より精度のいい 5 点公式や 7 点公式などもある。

3.3.2. 逆伝播法における連鎖律

改めて、今の目的は $\mathcal{L}$ のあるパラメータ p に関する偏微分を求めることである。以下のような計算を考えてみる。



$w^{(1)}$ および $w^{(2)}$ は学習可能なパラメータで、 $y^{(1)}$ および $y^{(2)}$ は $w^{(1)}, x^{(1)}$ および $w^{(2)}, x^{(2)}$ で何かしらの計算が行われる。 $y^{(1)}$ は $x^{(2)}$ と同じである。 $\mathcal{L}$ は $y^{(2)}$ と t を用いて計算される。

たとえば、 $w^{(2)}$ による偏微分を求めることを考えてみよう。 $w^{(2)}$ は $y^{(2)}$ に影響を与えており（これを寄与しているという）、 $y^{(2)}$ は $\mathcal{L}$ に寄与している。したがって、 $w^{(2)}$ の偏微分は連鎖律をもって以下のように表される。

$$\frac{\partial \mathcal{L}}{\partial w^{(2)}} = \frac{\partial \mathcal{L}}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial w^{(2)}}$$

ここで、 $\mathcal{L}$ は $y^{(2)}$ と t を用いて計算されているので、当然 $y^{(2)}$ に関する偏微分 $\frac{\partial \mathcal{L}}{\partial y^{(2)}}$ は $y^{(2)}$ と t を用いて計算可能であり、既知である。同様に、 $y^{(2)}$ は $w^{(2)}$ と $x^{(2)}$ を用いて計算されているので、 $w^{(2)}$ に関する偏微分 $\frac{\partial y^{(2)}}{\partial w^{(2)}}$ も既知である。したがって、 $w^{(2)}$ に関する偏微分 $\frac{\partial \mathcal{L}}{\partial w^{(2)}}$ は求めることができる。

次に、 $w^{(1)}$ による偏微分を求める。 $w^{(1)}$ は $y^{(1)}$ に寄与しており、 $y^{(1)}$ は $x^{(2)}$ と同一である。 $x^{(2)}$ は $y^{(2)}$ に寄与しており、 $y^{(2)}$ は $\mathcal{L}$ に寄与している。したがって、 $w^{(1)}$ の偏微分は連鎖律をもって以下のように表される。

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}} = \frac{\partial \mathcal{L}}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial x^{(2)}} \frac{\partial y^{(1)}}{\partial w^{(1)}}$$

これらはすべて既知であるので、 $w^{(1)}$ に関する偏微分 $\frac{\partial \mathcal{L}}{\partial w^{(1)}}$ は求めることができる。ここで重要なのは、学習可能なパラメータではない $x^{(2)}$ に関する偏微分を用意しておく必要があるのである。それ自体は学習に使うわけではないが、より上位の層に伝播するために必要である。これを逆伝播とよぶ。

つまり、各層ではより下位（順伝播では次に値する層）からの偏微分係数を受け取り、連鎖律を用いて学習可能なパラメータの勾配を求め、上位の層に伝播するために順伝播では入力に値するものの偏微分係数を求めるように実装すれば、繰り返し行うことですべてのパラメータに関する偏微分を求めることができる。では、各層の逆伝播の計算を順伝播の計算を元に求めることを次に考える。

3.3.3. 実数の逆伝播

行列で逆伝播を考える前に、実数の逆伝播法を考える。例えばある実数を返す関数 $\mathcal{L}$ [2]に関してある出力 y による偏微分係数 $\frac{\partial \mathcal{L}}{\partial y}$ が既知であるとする。そして入力 x に関して、

$$y = wx + b$$

が成立するとする。 w, b は更新対象のパラメータである。勾配降下法を用いるためにはこれらのパラメータに関する偏微分係数が必要である。そのためにまず、 y の偏微分を求める。

$$\frac{\partial y}{\partial x} = w \qquad \frac{\partial y}{\partial w} = x \qquad \frac{\partial y}{\partial b} = 1$$

次にこれらの関係をまとめると、

$$\begin{aligned} x &\mapsto y \mapsto \mathcal{L}(y) \\ w &\mapsto y \mapsto \mathcal{L}(y) \\ b &\mapsto y \mapsto \mathcal{L}(y) \end{aligned}$$

であるので、連鎖律を用いて以下のように表すことができる。

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x} &= \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x} = w \frac{\partial \mathcal{L}}{\partial y} \\ \frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial w} = x \frac{\partial \mathcal{L}}{\partial y} \\ \frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial b} = \frac{\partial \mathcal{L}}{\partial y} \end{aligned}$$

[2]表現からも分かる通り、損失関数を念頭に置いている。

ここで、$\frac{\partial \mathcal{L}}{\partial y}$ は既知であるので,入力とパラメータに関する偏微分係数を求めることができた。$\frac{\partial \mathcal{L}}{\partial x}$ はこの前の層にとっては既知とされた $\frac{\partial \mathcal{L}}{\partial y}$ に対応する。

3.3.4. Affine 層の逆伝播

ある行列 $\boldsymbol{X} \in \mathbb{M}^{n \times m}$ について $\frac{\partial}{\partial \boldsymbol{X}}$ を次のように定義する。

$$\frac{\partial}{\partial \boldsymbol{X}} := \begin{pmatrix} \frac{\partial}{\partial x_{11}} & \frac{\partial}{\partial x_{12}} & \cdots & \frac{\partial}{\partial x_{1m}} \\ \frac{\partial}{\partial x_{21}} & \frac{\partial}{\partial x_{22}} & \cdots & \frac{\partial}{\partial x_{2m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_{n1}} & \frac{\partial}{\partial x_{n2}} & \cdots & \frac{\partial}{\partial x_{nm}} \end{pmatrix}$$

$\frac{\partial}{\partial \boldsymbol{X}}$ 自身も形式上は $n \times m$ の行列であり、ドット積は形式的に行える。左側から行われた場合は、例えば $\frac{\partial}{\partial x_{11}} y$ のように書くことができるが、y を x_{11} で偏微分するという意味になる。

これを踏まえて、バッチ付き Affine 層の逆伝播を考える。Affine 層の出力 $\boldsymbol{Y}$ についての実数関数 $\mathcal{L}$ の偏微分 $\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}$ は既知であるとする。なお、要素を書き下すと以下のようになることに注意されたい。

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} := \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial y_{11}} & \frac{\partial \mathcal{L}}{\partial y_{12}} & \cdots & \frac{\partial \mathcal{L}}{\partial y_{1m}} \\ \frac{\partial \mathcal{L}}{\partial y_{21}} & \frac{\partial \mathcal{L}}{\partial y_{22}} & \cdots & \frac{\partial \mathcal{L}}{\partial y_{2m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial y_{n1}} & \frac{\partial \mathcal{L}}{\partial y_{n2}} & \cdots & \frac{\partial \mathcal{L}}{\partial y_{nm}} \end{pmatrix}$$

Affine 層は以下のような演算を行うことは既に述べた通りである。

$$\boldsymbol{Y} = \boldsymbol{X} \cdot \boldsymbol{W} + \boldsymbol{B}$$

y_{ij} について以下のように書き下せる。

$$y_{ij} = \sum_{1 \leq \alpha \leq n} x_{i\alpha} w_{\alpha j} + b_j$$

つぎに、両辺を w_{pq} と x_{pq} で偏微分すると、

$$\frac{\partial y_{ij}}{\partial w_{pq}} = \begin{cases} x_{ip} & \text{if} \quad j = q \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial y_{ij}}{\partial x_{pq}} = \begin{cases} w_{qj} & \text{if} \quad i = p \\ 0 & \text{otherwise} \end{cases}$$

となる. X の行と W の列は固定されているので、式にそれらが出現しないと偏微分係数は 0 になる。そして、 $\mathcal{L}$ がどのように w_{pq} に依存しているかを示すと、

$$w_{pq} \quad \mapsto \quad y_{ij}\,(1 \leq \forall i \leq n, 1 \leq \forall j \leq m) \quad \mapsto \quad \mathcal{L}$$

となっているので、

$$\begin{aligned}\left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}\right]_{pq} = \frac{\partial \mathcal{L}}{\partial w_{pq}} &= \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} \frac{\partial \mathcal{L}}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial w_{pq}} \\ &= \sum_{1 \leq i \leq n} \frac{\partial \mathcal{L}}{\partial y_{iq}} x_{ip} \\ &= \sum_{1 \leq i \leq n} \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}\right]_{iq} [\boldsymbol{X}]_{ip} \\ &= \sum_{1 \leq i \leq n} [\boldsymbol{X}^\top]_{pi} \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}\right]_{iq}\end{aligned}$$

が成立する。これはまさに、行列のドット積の定義であるので

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \boldsymbol{X}^\top \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}$$

が成立する。同様に

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} \cdot \boldsymbol{W}^\top$$

も成立する。 $\boldsymbol{X}, \boldsymbol{W}, \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}$ は既知であるので、重みの偏微分係数を求めることができた。 さて、バイアスについては、以下のような依存関係がある。

$$b_j \quad \mapsto \quad y_{ij}\,(1 \leq \forall i \leq n) \quad \mapsto \quad \mathcal{L}$$

したがって、 b_j について偏微分すると

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_j} &= \sum_{1 \leq i \leq n} \frac{\partial \mathcal{L}}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial b_j} \\ &= \sum_{1 \leq i \leq n} \frac{\partial \mathcal{L}}{\partial y_{ij}}\end{aligned}$$

となる。b_j は y_{ij} に依存しているので、j 番目の列の和をとることになる。したがって、逆伝播は以下のように表せる。

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} = \sum_{1 \leq i \leq n} \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} \right]_{i,*}$$

3.3.5. ReLU 関数の逆伝播

ReLU 関数は非常に簡単に逆伝播できる。ReLU 関数は以下のように定義されていた。

$$y_{ij} = \begin{cases} x_{ij} & \text{if} \quad x_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

これを x_{pq} について偏微分すれば

$$\frac{\partial y_{ij}}{\partial x_{pq}} = \begin{cases} 1 & \text{if} \quad i = p \wedge j = q \wedge x_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

となる。よって、

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_{pq}} &= \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} \frac{\partial \mathcal{L}}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{pq}} \\ &= \begin{cases} 1 & \text{if } x_{pq} \geq 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

となる。ReLU 関数自体が、行列の各要素ごとに適応されるため、その逆伝播も行列の各要素ごとに適応される。したがって、ReLU 関数の逆伝播は行列の計算を伴わず非常に簡単である。

3.3.6. Softmax 関数の逆伝播

Softmax 関数は、行列の各行ごとに適応される。したがって、Softmax 関数の逆伝播も行列の各行ごとに考える。この節に限り、入力を $\boldsymbol{x}$、出力を $\boldsymbol{y}$ とする。ともに大きさ C である。すると、Softmax 関数は以下のように定義されていた。

$$y_j = \frac{\exp(x_j)}{\sum_k \exp(x_k)}$$

右辺の分母を S とすると、y_j を x_q で偏微分すると

$$
\begin{aligned}
\frac{\partial y_j}{\partial x_q} &= \begin{cases} \dfrac{\exp(x_j)S - \exp(x_j)\{0 + \exp(x_j)\}}{S^2} & \text{if} \quad j = q \\ -\dfrac{\exp(x_j)\{0 + \exp(x_j)\}}{S^2} & \text{otherwise} \end{cases} \\
&= \begin{cases} \dfrac{\exp(x_j)}{S} - \dfrac{\exp(x_j)^2}{S^2} & \text{if} \quad j = q \\ -\dfrac{\exp(x_j)^2}{S^2} & \text{otherwise} \end{cases} \\
&= \begin{cases} y_j - y_j^2 & \text{if} \quad j = q \\ -y_j y_q & \text{otherwise} \end{cases} \qquad = y_j(\delta_{jq} - y_q)
\end{aligned}
$$

となる。ここで、δ_{jq} は Kronecker のデルタである。$j = q$ のときは 1、それ以外は 0 である。依存関係は以下のように表せる。

$$
x_q \quad \mapsto \quad y_j \, (1 \leq \forall j \leq C) \quad \mapsto \quad \mathcal{L}
$$

連鎖律を用いれば以下のように求められる。

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_q} &= \sum_{1 \leq j \leq C} \frac{\partial \mathcal{L}}{\partial y_j} \frac{\partial y_j}{\partial x_q} \\
&= \sum_{\substack{1 \leq j \leq C \\ j \neq q}} \frac{\partial \mathcal{L}}{\partial y_j}(-y_j y_q) + \frac{\partial \mathcal{L}}{\partial y_q}(y_q - y_q^2) \\
&= \sum_{1 \leq j \leq C} \frac{\partial \mathcal{L}}{\partial y_j}(-y_j y_q) + \frac{\partial \mathcal{L}}{\partial y_q} y_q \\
&= \sum_{1 \leq j \leq C} \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}\right]_j \left[\begin{pmatrix} y_1 & & & \\ & y_2 & & \\ & & \ddots & \\ & & & y_C \end{pmatrix}\right]_{jq} - \sum_{1 \leq j \leq C} \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}\right]_j \left[\begin{pmatrix} y_1 y_1 & y_1 y_2 & \cdots & y_1 y_C \\ y_2 y_1 & y_2 y_2 & \cdots & y_2 y_C \\ \vdots & \vdots & \ddots & \vdots \\ y_C y_1 & y_C y_2 & \cdots & y_C y_C \end{pmatrix}\right]_{jq}
\end{aligned}
$$

やや強引な式変形を行うと、簡単に求めることができる行列を用いることができる。以上から、

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \left\{ \begin{pmatrix} y_1 & & & \\ & y_2 & & \\ & & \ddots & \\ & & & y_C \end{pmatrix} - \begin{pmatrix} y_1 y_1 & y_1 y_2 & \cdots & y_1 y_C \\ y_2 y_1 & y_2 y_2 & \cdots & y_2 y_C \\ \vdots & \vdots & \ddots & \vdots \\ y_C y_1 & y_C y_2 & \cdots & y_C y_C \end{pmatrix} \right\} \\
&= \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \{\mathrm{diag}(\boldsymbol{y}) - \boldsymbol{y}^\top \cdot \boldsymbol{y}\}
\end{aligned}
$$

ここで、$\mathrm{diag}(\boldsymbol{y})$ は $\boldsymbol{y}$ の対角行列を表す。numpy などのライブラリには diag が実装されていることが多い。

3.3.7. クロスエントロピー誤差の逆伝播

レイヤと同じように損失関数にも逆伝播が必要である。逆伝播の開始位置ともいえるだろう。クロスエントロピー誤差は以下のように定義されていた。ただし、モデルの出力を $\boldsymbol{x}$ 、教師データを $\boldsymbol{t}$ とする。

$$\mathcal{L} = -\frac{1}{C}\sum_{1\leq j\leq C} t_j \log(x_j))$$

Softmax 関数と同様に行ごとに適応される。逆伝播はすぐに求めることができる。

$$\frac{\partial \mathcal{L}}{\partial y_j} = -\frac{1}{C}\frac{t_j}{x_j}$$

3.3.8. Softmax 関数とクロスエントロピー誤差の組み合わせ

Softmax 関数とクロスエントロピー誤差は、組み合わせて用いることが多い。というのも、Softmax 関数は確率分布を表し、クロスエントロピー誤差は確率分布の誤差を表すからである。Softmax 関数とクロスエントロピー誤差を組み合わせて逆伝播を求めてみる。ただし、Softmax 関数への入力を $\boldsymbol{x}$ 、出力を $\boldsymbol{y}$ クロスエントロピー誤差の教師データを $\boldsymbol{t}$ とする。

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_q} &= \sum_{1\leq j\leq C} \frac{\partial \mathcal{L}}{\partial y_j}\frac{\partial y_j}{\partial x_q} \\
&= \sum_{1\leq j\leq C} \frac{\partial \mathcal{L}}{\partial y_j}(-y_j y_q) + \frac{\partial \mathcal{L}}{\partial y_q} y_q \\
&= \sum_{1\leq j\leq C} \left(-\frac{1}{C}\frac{t_j}{y_j}\right)(-y_j y_q) + \left(-\frac{1}{C}\frac{t_q}{y_q}\right) y_q \\
&= \frac{y_q}{C}\left(\sum_{1\leq j\leq C} t_j - t_q\right) \\
&= \frac{y_q - t_q}{C}
\end{aligned}$$

ここで、 t_j はクロスエントロピー誤差の教師データである。 t_j は正規化されているので、 $\sum_{1\leq j\leq C} t_j = 1$ である。このように、Softmax 関数とクロスエントロピー誤差を組み合わせることで、非常に簡単に逆伝播を求めることができる。加えて、Softmax 関数もクロスエントロピー誤差も行列計算不可能にも関わらず、この逆伝播はバッチ付きの行列計算で表現できる。

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} = \frac{1}{C}(\boldsymbol{Y} - \boldsymbol{T})$$

4. ニューラルネットワークの工夫

前章までで、ニューラルネットワークの基本的な部分を学んだ。ここからは、ニューラルネットワークをより効率的に学習させるための工夫を紹介する。

4.1. Optimizer

Optimizer は、パラメータを更新するためのアルゴリズムである。SGD は Optimizer の一つである。ここでは、SGD にさまざまな工夫を加えた Optimizer を紹介する。この章ではあるパラメータ $\mathbb{P}$ を更新することを考える。

4.1.1. Momentum

Momentum は、SGD の更新に過去の更新を引き継いだ慣性のようなものを加えたものである。前回までの更新を $\boldsymbol{M}$ に保存し、新たな更新では勾配と $\boldsymbol{M}$ の両方を用いる。

$$\boldsymbol{M} \leftarrow \alpha \boldsymbol{M} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}}$$
$$\boldsymbol{P} \leftarrow \boldsymbol{P} + \boldsymbol{M}$$

4.1.2. RMSProp

RMSProp は、SGD の更新に対して学習率を調整する機能を付け加えたものである。更新分が一定だと、最小に近づくにつれ、更新幅が必要な幅より大きくなってしまう問題を解決するため、更新幅の分だけ学習率を割り引く。更新幅は指数移動平均を用いて計算し、ある程度過去の更新幅を引き継ぐ。

$$\boldsymbol{G} \leftarrow \beta \boldsymbol{G} + (1-\beta) \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}} \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}}$$
$$\boldsymbol{P} \leftarrow \boldsymbol{P} - \frac{\eta}{\sqrt{\boldsymbol{G}}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}}$$

パラメータごとに学習率を調整するために、アダマール積とアダマール除算を用いている。

4.1.3. Adam

Adam は、Momentum と RMSProp を組み合わせたものである。

$$\boldsymbol{M} \leftarrow \beta_1 \boldsymbol{M} + (1-\beta_1) \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}}$$
$$\boldsymbol{G} \leftarrow \beta_2 \boldsymbol{G} + (1-\beta_2) \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}} \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}}$$
$$\widehat{\boldsymbol{M}} = \frac{\boldsymbol{M}}{1-\beta_1^{t+1}} \qquad \hat{\boldsymbol{G}} = \frac{\boldsymbol{G}}{1-\beta_2^{t+1}}$$
$$\boldsymbol{P} \leftarrow \boldsymbol{P} - \frac{\eta}{\sqrt{\hat{\boldsymbol{G}}}} \widehat{\boldsymbol{M}}$$
$$t \leftarrow t+1$$

4.1.4. AdamW

AdamW は、Adam に L2 正則化を加えたものである。L2 正則化と AdamW の構造についてはペナルティ項の追加にて後述する。

4.2. レイヤ

Affine 層や活性化関数層以外の、レイヤを追加することで学習を加速あるいは、精度を上げることができる。ここでは、代表的なレイヤを紹介する。

4.2.1. Batch Normalization 層

4.2.1.1. 順伝播

学習時はバッチを用いて複数のデータを同時に学習する。この際、バッチ内のデータに偏りが存在する場合があり、学習が進まないことがある。また、途中の層の出力が極端になってしまうことがある。これを防ぐために、バッチ内のデータについて、各特徴量を正則化する。その後、学習可能なパラメータを用いて、正則化したデータを変換する。これを Batch Normalization（以下、BN）とよぶ。入力が $\boldsymbol{X} \in \mathbb{M}^{B \times C}$ の行列である場合、正則化を行うのは列ごとであることに注意されたい。

ここでは $\boldsymbol{X}$ の i 行目ベクトルを $\boldsymbol{x}^{(i)}$ とする。ここでは、要素 $x_j^{(i)}$ がどのように正則化されるかを考える。その際、 j 列目のことのみを考えればよい。 j 行目の平均を μ_j 、分散を σ_j とし、中間出力を $\boldsymbol{z}^{(i)}$ 、最終出力を $\boldsymbol{y}^{(i)}$ とする。すると、BN は以下のように行われる。

$$
\begin{aligned}
\mu_j &= \frac{1}{B} \sum_{1 \leq k \leq B} x_j^{(k)} \\
\sigma_j^2 &= \frac{1}{B} \sum_{1 \leq k \leq B} \left(x_j^{(k)} - \mu_j \right)^2 \\
z_j^{(i)} &= \frac{x_j^{(i)} - \mu_j}{\sigma_j} \\
y_j^{(i)} &= \gamma_j z_j^{(i)} + \beta_j
\end{aligned}
$$

ここで、 γ_j と β_j は学習可能なパラメータで、行ごとに別の値が適応される。正則化後のデータを適切にスケールするためのものである。 μ_j 、 σ_j は推論時に列ごとに計算される。つまり、実際には $\gamma, \beta, \boldsymbol{\mu}, \boldsymbol{\sigma}$ はベクトルである。次に、推論時では平均と標準偏差は batch 内のデータを用いて計算されない。なぜなら、推論時には学習時にくらべて少ないデータが入力されるからである。したがって、学習時に平均と分散をある程度計算しておく必要がある。 毎ステップ、上で計算したバッチの平均と分散を用いて推論時に用いる平均 $\boldsymbol{\mu}_{\text{running}}$ と分散 $\boldsymbol{\sigma}_{\text{running}}^{\circ 2}$ を更新する。 α は係数である。

$$
\begin{aligned}
\boldsymbol{\mu}_{\text{running}} &\leftarrow \alpha * \boldsymbol{\mu}_{\text{running}} + (1 - \alpha) \boldsymbol{\mu} \\
\boldsymbol{\sigma}_{\text{running}}^{\circ 2} &\leftarrow \alpha * \boldsymbol{\sigma}_{\text{running}}^{\circ 2} + (1 - \alpha) \boldsymbol{\sigma}^{\circ 2}
\end{aligned}
$$

推論時には $\boldsymbol{\mu}$ と $\boldsymbol{\sigma}$ の代わりに $\boldsymbol{\mu}_{\text{running}}$ と $\boldsymbol{\sigma}_{\text{running}}$ を用いる。

4.2.1.2. 逆伝播

次に、学習可能なパラメータである $\boldsymbol{\gamma}$ と $\boldsymbol{\beta}$ と、 $\boldsymbol{X}$ の逆伝播を考える。ここでも、 $\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}$ は既知とする。まずは、 $\boldsymbol{Y}$ から $\boldsymbol{Z}$ までの逆伝播を考える。まず、 $\gamma_j, \beta_j, z_j^{(i)}$ の損失関数 $\mathcal{L}$ に対する依存関係を整理しよう。

$$
\begin{aligned}
\gamma_j &\mapsto y_j^{(k)}\ (1 \leq \forall k \leq B) \mapsto \mathcal{L} \\
\beta_j &\mapsto y_j^{(k)}\ (1 \leq \forall k \leq B) \mapsto \mathcal{L} \\
z_j^{(i)} &\mapsto y_j^{(i)} \mapsto \mathcal{L}
\end{aligned}
$$

つまり、

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial z_j^{(i)}} &= \frac{\partial \mathcal{L}}{\partial y_j^{(i)}} \frac{\partial y_j^{(i)}}{\partial z_j^{(i)}} = \frac{\partial \mathcal{L}}{\partial y_j^{(i)}} \gamma_j \\
\frac{\partial \mathcal{L}}{\partial \gamma_j} &= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial \gamma_j} \\
&= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial y_j^{(k)}} z_j^{(k)} \\
\frac{\partial \mathcal{L}}{\partial \beta_j} &= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial \beta_j} \\
&= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial y_j^{(k)}}
\end{aligned}
$$

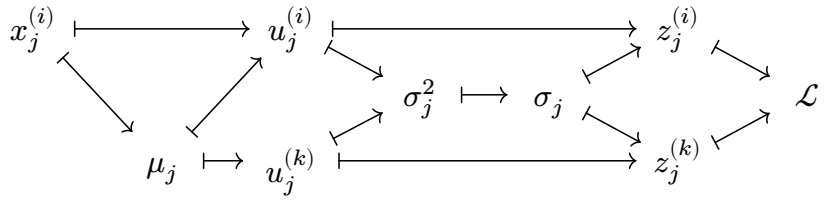
である。つまり、ベクトル表現を用いると

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\gamma}} &= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(k)}} \odot \boldsymbol{z}^{(k)} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} &= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(k)}}
\end{aligned}
$$

となる。次に、 $\boldsymbol{Z}$ から $\boldsymbol{X}$ までの逆伝播を考える。注意すべき点は $x_j^{(i)}$ が μ や σ に寄与しており、それがすべての $z_j^{(k)} (1 \leq \forall k \leq B)$ に寄与していることである。計算しやすさのため、 $u_j^{(i)} := x_j^{(i)} - \mu_j$ を定義する。すると、 σ_j は

$$
\sigma_j = \frac{1}{B} \sum_{1 \leq k \leq B} \left(u_j^{(k)} \right)^2
$$

と表される。 $u_j^{(i)}$ を用いて関係をまとめると以下のようになる。ただしこの図で k は $1 \leq k \leq B \wedge k \neq i$ を満たすすべての k である。



$u_j^{(i)}$ に対しては、$x_j^{(i)}$ が μ_j を介してだけではなく直接作用するので、$u_j^{(i)}$ と $z_j^{(i)}$ を他の $u_j^{(k)}$ や $z_j^{(k)}$ と区別する必要がある。逆に、$u_j^{(k)}$ までは区別する必要はないのでこれからは k は i も含めた任意の添字として扱う。まずは、σ_j について考える。

$$
\begin{aligned}
z_j^{(k)} = \frac{u_j^{(k)}}{\sigma_j} \quad \Rightarrow \quad \frac{\partial z_j^{(k)}}{\partial \sigma_j} &= -\frac{u_j^{(k)}}{\sigma_j^2} \\
\frac{\partial \sigma_j}{\partial \sigma_j^2} &= \frac{1}{2\sigma_j} \\
\therefore \frac{\partial \mathcal{L}}{\partial \sigma_j^2} &= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial \sigma_j} \frac{\partial \sigma_j}{\partial \sigma_j^2} \\
&= -\frac{1}{2\sigma_j^3} \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial z_j^{(k)}} u_j^{(k)}
\end{aligned}
$$

次に、$u_j^{(k)}$ について考えると、

$$
\sigma_j = \frac{1}{B} \sum_{1 \leq k \leq B} \left(u_j^{(k)} \right)^2 \quad \Rightarrow \quad \frac{\partial \sigma_j^2}{\partial u_j^{(k)}} = \frac{2}{B} u_j^{(k)}
$$

である。また、$z_j^{(k)} = \frac{u_j^{(k)}}{\sigma_j}$ から、$u_j^{(i)}$ の σ_j に対する依存を一旦無視すると

$$
\left. \frac{\partial z_j^{(k)}}{\partial u_j^{(k)}} \right|_{\text{direct}} = \frac{1}{\sigma_j}
$$

であるので、

$$
\begin{aligned}
\frac{\partial z_j^{(k)}}{\partial u_j^{(k)}} &= \left. \frac{\partial z_j^{(k)}}{\partial u_j^{(k)}} \right|_{\text{direct}} + \frac{\partial z_j^{(k)}}{\partial \sigma_j^2} \frac{\partial \sigma_j^2}{\partial u_j^{(k)}} \\
&= \frac{1}{\sigma_j} + \frac{2}{B} u_j^{(k)} \frac{\partial z_j^{(k)}}{\partial \sigma_j^2} \\
\therefore \frac{\partial \mathcal{L}}{\partial u_j^{(k)}} &= \frac{\partial \mathcal{L}}{\partial z_j^{(k)}} \frac{1}{\sigma_j} + \frac{2}{B} u_j^{(k)} \frac{\partial \mathcal{L}}{\partial \sigma_j^2}
\end{aligned}
$$

となる。最後に $u_j^{(k)}$ に対する $x_j^{(i)}$ への逆伝播を考える。しかし、 μ_j がすべての $u_j^{(k)}$ に寄与していることを考慮する必要がある。

$$
\begin{aligned}
u_j^{(k)} = x_j^{(k)} - \mu_j \quad \Rightarrow \quad & \frac{\partial u_j^{(k)}}{\partial \mu_j} = -1 \\
\therefore \frac{\partial \mathcal{L}}{\partial \mu_j} &= \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial u_j^{(k)}} \frac{\partial u_j^{(k)}}{\partial \mu_j} \\
&= - \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial u_j^{(k)}} \\
&= - \sum_{1 \leq k \leq B} \left[\frac{\partial \mathcal{L}}{\partial z_j^{(k)}} \frac{1}{\sigma_j} + \frac{2}{B} u_j^{(k)} \frac{\partial \mathcal{L}}{\partial \sigma_j^2} \right] \\
&= -\frac{1}{\sigma_j} \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial z_j^{(k)}} - \frac{2}{B} \frac{\partial \mathcal{L}}{\partial \sigma_j^2} \sum_{1 \leq k \leq B} u_j^{(k)}
\end{aligned}
$$

ここで、 $\sum_{1 \leq k \leq B} u_j^{(k)}$ について考えてみる。

$$
\begin{aligned}
\sum_{1 \leq k \leq B} u_j^{(k)} &= \sum_{1 \leq k \leq B} \left(x_j^{(k)} - \mu_j \right) \\
&= \sum_{1 \leq k \leq B} x_j^{(k)} - B\mu_j \\
&= \sum_{1 \leq k \leq B} x_j^{(k)} - \sum_{1 \leq k \leq B} x_j^{(k)} \\
&= 0
\end{aligned}
$$

したがって、

$$
\frac{\partial \mathcal{L}}{\partial \mu_j} = -\frac{1}{\sigma_j} \sum_{1 \leq k \leq B} \frac{\partial \mathcal{L}}{\partial z_j^{(k)}}
$$

そして、 $x_j^{(i)}$ は直接 $u_j^{(i)}$ に寄与している他、 μ_j を通して $u_j^{(k)}$ に寄与している。つまり、

$$\mu_j = \frac{1}{B}\sum_{1\leq k\leq B} x_j^{(k)} \quad \Rightarrow \quad \frac{\partial \mu_j}{\partial x_j^{(i)}} = \frac{1}{B}$$

$$\therefore \frac{\partial \mathcal{L}}{\partial x_j^{(i)}} = \frac{\partial \mathcal{L}}{\partial u_j^{(i)}}\frac{\partial u_j^{(i)}}{\partial x_j^{(i)}}\bigg|_{\text{direct}} + \frac{\partial \mathcal{L}}{\partial \mu_j}\frac{\partial \mu_j}{\partial x_j^{(i)}}$$

$$= \frac{\partial \mathcal{L}}{\partial u_j^{(i)}} + \frac{1}{B}\frac{\partial \mathcal{L}}{\partial \mu_j}$$

$$= \frac{\partial \mathcal{L}}{\partial u_j^{(i)}} - \frac{1}{B\sigma_j}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}$$

となる。第 1 項について詳しく考える。 $z_j^{(k)} = \frac{u_j^{(k)}}{\sigma_j}$ を念頭に考えると、

$$\frac{\partial \mathcal{L}}{\partial u_j^{(i)}} = \frac{\partial \mathcal{L}}{\partial z_j^{(i)}}\frac{1}{\sigma_j} + \frac{2}{B}u_j^{(i)}\frac{\partial \mathcal{L}}{\partial \sigma_j^2}$$

$$= \frac{\partial \mathcal{L}}{\partial z_j^{(i)}}\frac{1}{\sigma_j} + \frac{2}{B}u_j^{(i)}\left(-\frac{1}{2\sigma_j^3}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}u_j^{(k)}\right)$$

$$= \frac{\partial \mathcal{L}}{\partial z_j^{(i)}}\frac{1}{\sigma_j} - \frac{1}{\sigma_j}\frac{1}{B}\frac{u_j^{(i)}}{\sigma_j}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}\frac{u_j^{(k)}}{\sigma_j}$$

$$= \frac{\partial \mathcal{L}}{\partial z_j^{(i)}}\frac{1}{\sigma_j} - \frac{1}{\sigma_j}\frac{z_j^{(i)}}{B}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}z_j^{(k)}$$

つまり、以上をまとめると

$$\frac{\partial \mathcal{L}}{\partial x_j^{(i)}} = \frac{\partial \mathcal{L}}{\partial z_j^{(i)}}\frac{1}{\sigma_j} - \frac{1}{\sigma_j}\frac{z_j^{(i)}}{B}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}z_j^{(k)}$$

$$= \frac{\partial \mathcal{L}}{\partial z_j^{(i)}}\frac{1}{\sigma_j} - \frac{1}{\sigma_j}\frac{z_j^{(i)}}{B}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}z_j^{(k)} - \frac{1}{B\sigma_j}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}$$

$$= \frac{1}{\sigma_j}\left[\frac{\partial \mathcal{L}}{\partial z_j^{(i)}} - \frac{z_j^{(i)}}{B}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}z_j^{(k)} - \frac{1}{B}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial z_j^{(k)}}\right]$$

$$= \frac{\gamma_j}{\sigma_j}\left[\frac{\partial \mathcal{L}}{\partial y_j^{(i)}} - \frac{z_j^{(i)}}{B}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial y_j^{(k)}}z_j^{(k)} - \frac{1}{B}\sum_{1\leq k\leq B}\frac{\partial \mathcal{L}}{\partial y_j^{(k)}}\right]$$

となる。最後の等号は、 $\frac{\partial \mathcal{L}}{\partial z_j^{(i)}} = \gamma_j\frac{\partial \mathcal{L}}{\partial y_j^{(i)}}$ を用いている。さて、 γ_j や β_j の勾配を思い出すと、

$$\frac{\partial \mathcal{L}}{\partial x_j^{(i)}} = \frac{\gamma_j}{\sigma_j}\left[\frac{\partial \mathcal{L}}{\partial y_j^{(i)}} - \frac{z_j^{(i)}}{B}\frac{\partial \mathcal{L}}{\partial \gamma_j} - \frac{1}{B}\frac{\partial \mathcal{L}}{\partial \beta_j}\right]$$

となる。この式はすべての j 列に対して成立するのでベクトル表現を用いれば以下のように整理できる。

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{(i)}} = \frac{\boldsymbol{\gamma}}{\boldsymbol{\sigma}} \odot \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}^{(i)}} - \frac{1}{B} \left\{ \boldsymbol{z}^{(i)} \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{\gamma}} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \right\} \right]$$

さらに成分がすべて 1 で大きさ B の列ベクトル $\mathbf{1}_B$ を用いて行列表現に拡張すると以下のようになる。

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}} = \left(\mathbf{1}_B \cdot \frac{\boldsymbol{\gamma}}{\boldsymbol{\sigma}} \right) \odot \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} - \frac{1}{B} \left\{ \boldsymbol{Z} \odot \left(\mathbf{1}_B \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{\gamma}} \right) + \mathbf{1}_B \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \right\} \right]$$

4.2.2. Dropout 層

Dropout は、学習時にランダムにノードを無効化することで、過学習を防ぐ手法である。そもそも背景として、構造の異なる複数のモデルを学習させ、それらの結果を平均化することで、より精度の高いモデルを得るという方法がある。これをアンサンブル学習と呼ぶ。Dropout はミニバッチごとにある一定確率でニューロン（ノード）を無効化することで、アンサンブル学習に似た効果を得る手法である。Dropout は、学習時にのみ適応される。推論時はすべてのニューロンを有効化する。また、推論時と同様のスケールに保つため、学習時は Dropout を適応した後にスケールを合わせる必要がある。この Dropout をレイヤとして実装することを考える。

4.2.2.1. 順伝播

全結合層の順伝播でニューロンを無効化するということは、ニューロンの出力を 0 にするということである。無効化を行う層の単一の出力を $\boldsymbol{y}$ とする。これは Affine 層の出力に関して活性化関数を作用させたものである。もし、 j 番目のニューロンを無効化するならば $y_j = 0$ とするということである。つまり、Dropout 層は Dropout を適応する全結合層の後に適応するように設計するのが適当である。

まず一つのニューロンが無効化される確率を p とする。すると全体で n 個のニューロンがあった時に、k 個のニューロンが無効化される確率 $P^{-}(k)$ は

$$P^{-}(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

である。つまりその期待値は

$$\begin{aligned}
E^{-} &= \sum_{0 \leq k \leq n} k P^{-}(k) \\
&= \sum_{1 \leq k \leq n} \frac{n!}{(k-1)!(n-k)!} p^k (1-p)^{n-k} \\
&= np \sum_{0 \leq k \leq n-1} \frac{(n-1)!}{k!(n-k-1)!} p^k (1-p)^{n-k-1} \\
&= np(p + (1-p))^{n-1} = np
\end{aligned}$$

つまり全体のうち、割合 p だけのニューロンが無効化される。つまり、 $1-p$ の割合のニューロンが有効化される。推論時に比べ、 $1-p$ のスケールが学習時に出力される。これを推論時と同じスケールにするために $1-p$ で割る。また、無効化される成分を 0 、そうでない成分を 1 とするベクトル $\boldsymbol{r}$ を用いる。すると、 $\boldsymbol{r}$ は p の確率で 0 、 $1-p$ の確率で 1 となる。これは $\boldsymbol{r}$ の各成分がベルヌーイ分布に従うことを意味する。つまり、Dropout 層の順伝播は以下のように表現できる。

$$
\begin{aligned}
r_j &\sim Bi(1-p) \quad (1 \leq \forall j \leq n) \\
\hat{\boldsymbol{y}} &= \frac{\boldsymbol{r} \odot \boldsymbol{y}}{1-p}
\end{aligned}
$$

さて、バッチ学習においてもこれらは基本的に同じである。どのニューロンを無効化するかは、バッチごとではなくデータごとに決定される。つまり、全結合層の出力 $\boldsymbol{Y} \in \mathbb{M}^{B \times C}$ に対して、Dropout を適応した出力 $\hat{\boldsymbol{Y}} \in \mathbb{M}^{B \times C}$ は

$$
\begin{aligned}
r_{ij} &\sim Bi(1-p) \quad (1 \leq \forall i \leq B, 1 \leq \forall j \leq C) \\
\hat{\boldsymbol{Y}} &= \frac{\boldsymbol{Y} \odot \boldsymbol{R}}{1-p}
\end{aligned}
$$

で計算される。

4.2.2.2. 逆伝播

Dropout 層には学習可能なパラメータは存在しない。つまり、 $\dfrac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}$ のみを考えればいい。

$$
\begin{aligned}
\hat{y}_{ij} &= \frac{y_{ij} r_{ij}}{1-p} \\
\frac{\partial \hat{y}_{ij}}{\partial y_{ij}} &= \frac{r_{ij}}{1-p}
\end{aligned}
$$

より、

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial y_{ij}} &= \frac{\partial \mathcal{L}}{\partial \hat{y}_{ij}} \frac{\partial \hat{y}_{ij}}{\partial y_{ij}} \\
&= \frac{\partial \mathcal{L}}{\partial \hat{y}_{ij}} \frac{r_{ij}}{1-p} \\
\therefore \frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} &= \frac{1}{1-p} \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{Y}}} \odot \boldsymbol{R}
\end{aligned}
$$

となる。

4.3. ペナルティ項の追加

ペナルティ項とは過学習を防ぐために損失関数に追加する項である。過学習の状況ではパラメータが汎用性を失うため、重みといったパラメータが非常に複雑な値をとることがある。これを防ぐために、パ

ラメータの「大きさ」をペナルティとして追加する。パラメータの「大きさ」を求める手法でいくつかの違いが存在する。ここでは、L1 正則化と L2 正則化を紹介する。一方で、実装においては損失関数にペナルティ項を追加するのではなく、Optimizer などの更新時に用いる勾配を変更することで実装することがある。あるペナルティ項を $\mathcal{P}$ とすると、ペナルティ項を追加した損失関数は

$$\mathcal{L}_{\text{new}} = \mathcal{L} + \mathcal{P}$$

である。これを用いてパラメータ p の勾配を考えると、

$$\frac{\partial \mathcal{L}_{\text{new}}}{\partial p} = \frac{\partial \mathcal{L}}{\partial p} + \frac{\partial \mathcal{P}}{\partial p}$$

となる。 $\frac{\partial \mathcal{L}}{\partial p}$ は元の損失関数を用いて今まで通り計算を行った勾配である。つまり、その元の勾配の代わりにそれに $\frac{\partial \mathcal{P}}{\partial p}$ を加えてものを用いることでペナルティ項を追加することができる。

4.3.1. L1 正則化（Lasso：Least Absolute Shrinkage and Selection Operator）

L1 正則化は、パラメータの「大きさ」として L1 ノルムを用いる手法である。L1 ノルムは、パラメータの絶対値の和である。つまり、新たな損失関数 $\mathcal{L}_{\text{L1}}$ は

$$\mathcal{L}_{\text{L1}} = \mathcal{L} + \lambda \left\| \boldsymbol{P} \right\|_1 = \mathcal{L} + \lambda \sum_i |p_i|$$

となる。つまり、新たな勾配との SGD における更新式は

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{L1}}}{\partial p_i} &= \frac{\partial \mathcal{L}}{\partial p_i} + \lambda \frac{\partial \mathcal{P}}{\partial p_i} \\
&= \frac{\partial \mathcal{L}}{\partial p_i} + \lambda \ \text{sign}(p_i) \\
p_i &\leftarrow p_i - \eta \frac{\partial \mathcal{L}}{\partial p_i} - \eta \lambda \ \text{sign}(p_i)
\end{aligned}$$

となる。 $\text{sign}(p_i)$ は p_i の符号を表す関数である。つまり、 p_i が正ならば 1 、負ならば -1 、 0 ならば 0 となる。L1 正則化は、比較的強くパラメータを 0 に近づける。なぜなら、パラメータが 0 でなければ、そのスケールに関わらず勾配のペナルティ項の絶対値は 1 である。すると、パラメータが 0 でない限りそれを 0 に近づけようとする力は常に一定で働く。これは結果として、必要最低限のパラメータを取捨選択することにつながり、必要最低限で非常に疎なパラメータを得ることができる。これをスパース性という。

4.3.2. L2 正則化（Ridge）

L2 正則化は、パラメータの「大きさ」として L2 ノルムを用いる手法である。しかし、諸々の理由で L2 ノルムそのものではなく、その 2 乗を用いる。新たな損失関数 $\mathcal{L}_{\text{L2}}$ は

$$\mathcal{L}_{\mathrm{L2}} = \mathcal{L} + \frac{\lambda}{2} \left\| \boldsymbol{P} \right\|_2^2 = \mathcal{L} + \frac{\lambda}{2} \sum_i p_i^2$$

である。ここで、勾配計算の簡単さから $\frac{1}{2}$ を掛けている。つまり、新たな勾配と SGD における更新式は

$$\frac{\partial \mathcal{L}_{\mathrm{L2}}}{\partial p_i} = \frac{\partial \mathcal{L}}{\partial p_i} + \lambda p_i$$
$$p_i \leftarrow p_i - \eta \frac{\partial \mathcal{L}}{\partial p_i} - \eta \lambda p_i$$

となる。L2 正則化は、すべてのパラメータを 0 に近づけるように働く。しかし、一方で L1 正則化とは異なり、パラメータを 0 に近づける力はパラメータの大きさに依存しているので、 0 に近づけるにつれその力は弱まる。つまり、パラメータはそこまで 0 にはならない。一方で、パラメータが複雑になることを防ぐことができる。

4.3.3. L1L2 正則化（Elastic Net）

L1L2 正則化は、L1 正則化と L2 正則化を純粋に足し合わせたものである。新たな損失関数 $\mathcal{L}_{\mathrm{L1L2}}$ は

$$\mathcal{L}_{\mathrm{L1L2}} = \mathcal{L} + \lambda_1 \sum_i |p_i| + \frac{\lambda_2}{2} \sum_i p_i^2$$

となる。つまり、新たな勾配と SGD における更新式は

$$\frac{\partial \mathcal{L}_{\mathrm{L1L2}}}{\partial p_i} = \frac{\partial \mathcal{L}}{\partial p_i} + \lambda_1 \ \mathrm{sign}(p_i) + \lambda_2 p_i$$
$$p_i \leftarrow p_i - \eta \frac{\partial \mathcal{L}}{\partial p_i} - \eta \lambda_1 \ \mathrm{sign}(p_i) - \eta \lambda_2 p_i$$

となる。

4.3.4. L2 正則化の問題点と AdamW

L2 正則化を Adam と組み合わせて学習を行うと、置き換えたのちの勾配を用いて Adam の更新式を用いることになる。つまり、L2 正則化を行うと、Adam の性質よりそのペナルティ項も含めて更新式を計算することになり、ペナルティ項が Adam でのスケーリングの影響を受けてしまう。これを防ぐために、AdamW という手法が提案された。AdamW は、L2 正則化のペナルティ項を含めない勾配を用いてスケーリングを行ったのち、更新時にペナルティ項を加える手法である。つまり、L2 正則化を Optimizer で担うのである。式は以下のようになる。

$$
\begin{aligned}
\boldsymbol{M} &\leftarrow \beta_1 \boldsymbol{M} + (1-\beta_1)\frac{\partial \mathcal{L}}{\partial \boldsymbol{P}} \\
\boldsymbol{G} &\leftarrow \beta_2 \boldsymbol{G} + (1-\beta_2)\frac{\partial \mathcal{L}}{\partial \boldsymbol{P}} \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}} \\
\widehat{\boldsymbol{M}} &= \frac{\boldsymbol{M}}{1-\beta_1^{t+1}} \qquad \hat{\boldsymbol{G}} = \frac{\boldsymbol{G}}{1-\beta_2^{t+1}} \\
\boldsymbol{P} &\leftarrow \boldsymbol{P} - \frac{\eta}{\sqrt{\hat{\boldsymbol{G}}}}\widehat{\boldsymbol{M}} - \eta\lambda\boldsymbol{P} \\
t &\leftarrow t+1
\end{aligned}
$$

Adam と見比べてみると、 $\boldsymbol{P}$ の更新式にペナルティ項が追加されていることがわかる。