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

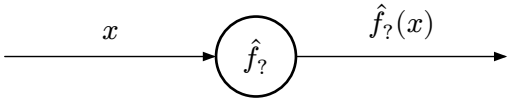
- ベクトルは太文字の小文字で表す。($\boldsymbol{x}$)
- 行列は太文字の大文字で表す。($\boldsymbol{X}$)
- スカラーは通常の小文字で表す。(x)
- 行列の要素は定義のうえ、通常の小文字と添字で表す。添字は行、列の順に表記し、わかりにくい場合を除き区切り文字をつけない。(x_{ij} $x_{i+1,j}$)
- 行列の要素について断りがない時、行列名 $\boldsymbol{X}$ を用いて $[\boldsymbol{X}]_{ij}$ と表すことがある。添字は同様。
- N 次元ベクトルは大きさ $1 \times N$ の行列とみなし、いわゆる行ベクトルとする。
- 列ベクトルの要素表示は通常、行ベクトルと転置記号 $\top$ を用いて表す。わかりやすさのため、列ベクトルそのものを書くこともある。

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。

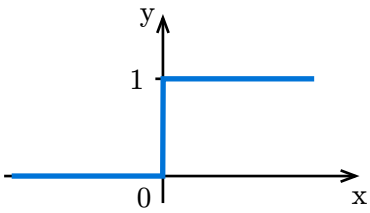
2.1. ニューロンの始まり

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。

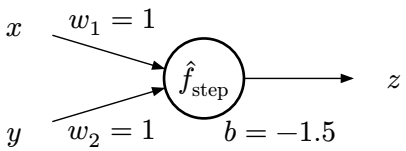
$$f_{\text{step}}(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



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。

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

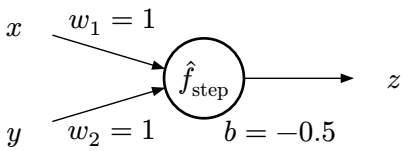
w_i は i 番目の入力 x_i に対する結合強度を表す。このようにしてニューロンは複数の入力を受け取り,それらの結合強度と閾値を考慮して出力を決定することができる。これをパーセプトロンという。このパーセプトロンによる実例を見てみよう。



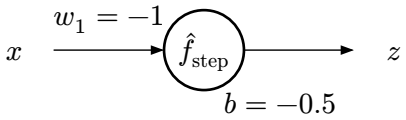
$$z = f_{\text{step}}(x + y - 1.5) = \begin{cases} 1 & \text{if} \quad x + y \geq 1.5 \\ 0 & \text{otherwise} \end{cases}$$

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。



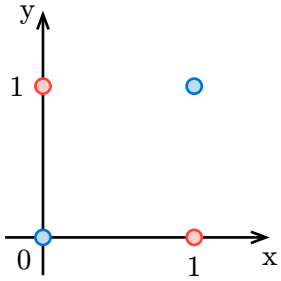
2.2. 多層パーセプトロン

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。

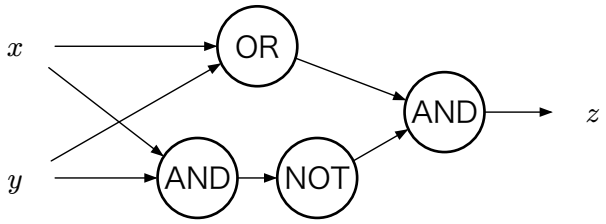
二つの入力を x 軸 y 軸として、出力を色分けした。青色は 0 、赤色は 1 を表している。ニューロンは

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

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

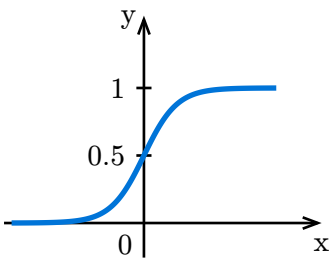
多層パーセプトロンは、複数のニューロンを組み合わせることでより複雑な関数を実装することができる。しかし、上のようなステップ関数 f_{step} では、出力が離散的で値の表現が乏しい。そのため、連続的な値を出力する関数を用いる。これら、ニューロンの出力を決定する関数を活性化関数とよぶ。

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

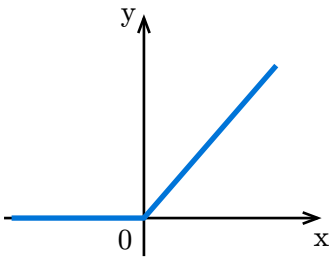
$$f_{\text{sigmoid}}(x) = \frac{1}{1+\exp(-x)}$$



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。

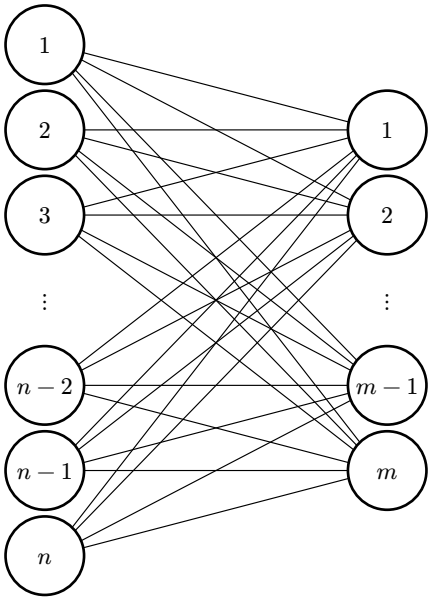
$$
\begin{aligned}
&C = \text{const.} \\
&\frac{\exp(x_i - C)}{\sum_j \exp\bigl(x_j - C\bigr)} = \frac{e^{-C}\exp(x_i)}{\sum_j e^{-C}\exp\bigl(x_j\bigr)} \\
&\qquad = \frac{e^{-C}\exp(x_i)}{e^{-C}\sum_j \exp\bigl(x_j\bigr)} \\
&\qquad = \frac{\exp(x_i)}{\sum_j \exp\bigl(x_j\bigr)}
\end{aligned}
$$

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。

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

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。

$$\begin{aligned}
\boldsymbol{X} \cdot \boldsymbol{W} + \boldsymbol{B} &= \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_{|b|} \end{pmatrix}^\top \cdot \boldsymbol{W} + \boldsymbol{B} \\
&= \begin{pmatrix} \boldsymbol{x}_1 \cdot \boldsymbol{W} & \boldsymbol{x}_2 \cdot \boldsymbol{W} & \dots & \boldsymbol{x}_{|b|} \cdot \boldsymbol{W} \end{pmatrix}^\top + \boldsymbol{B} \\
&= \begin{pmatrix} \boldsymbol{x}_1 \cdot \boldsymbol{W} + \boldsymbol{b} & \boldsymbol{x}_2 \cdot \boldsymbol{W} + \boldsymbol{b} & \dots & \boldsymbol{x}_{|b|} \cdot \boldsymbol{W} + \boldsymbol{b} \end{pmatrix}^\top \\
&= \begin{pmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_{|b|} \end{pmatrix}^\top \\
&= \boldsymbol{Y}
\end{aligned}$$

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 $\boldsymbol{x}^{(\mathrm{train})}$ 、それを与えた場合に出力されるべきデータを $\boldsymbol{y}^{(\mathrm{train})}$ 、実際に出力されたデータを $\boldsymbol{y}^{(\mathrm{pred})}$ とする。 $\boldsymbol{y}^{(\mathrm{train})}$ と $\boldsymbol{y}^{(\mathrm{pred})}$ 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 $\boldsymbol{y}^{(\mathrm{train})}$ と与えられたデータ $\boldsymbol{y}^{(\mathrm{pred})}$ 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 $\boldsymbol{x}^{(\mathrm{train})}$ だけでなく、用いるパラメータ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、 $\boldsymbol{y}^{(\mathrm{train})}$ は決められた定数とみなす。

$$\mathcal{L} : \mathbb{R}^{|\boldsymbol{x}^{(\mathrm{train})}|} \times \mathbb{R}^{\mathrm{parameters}} \to \mathbb{R}$$

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、 $\boldsymbol{y}_{\mathrm{pred}}$ と $\boldsymbol{y}_{\mathrm{train}}$ 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。

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{C}||\boldsymbol{y}^{(\mathrm{pred})} - \boldsymbol{y}^{(\mathrm{train})}||^2$$

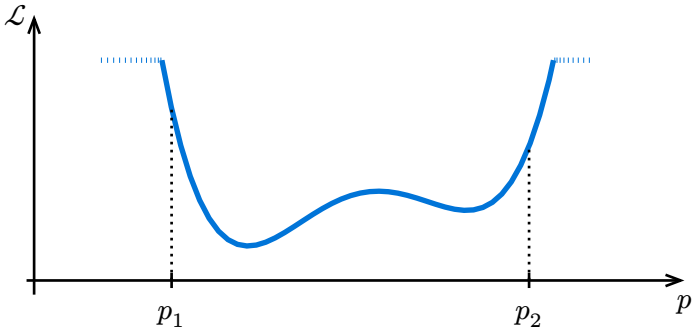
3.1.2. クロスエントロピー誤差

クロスエントロピー誤差は、 $\boldsymbol{y}_{\mathrm{pred}}$ と $\boldsymbol{y}_{\mathrm{train}}$ 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。

$$\mathcal{L}_{\mathrm{CrossEntropy}} = -\sum_{1 \leq j \leq C} y_i^{(\mathrm{train})} \log\left(y_i^{(\mathrm{pred})}\right)$$

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. 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。

[2]表現からも分かる通り、損失関数を念頭に置いている。

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x} = w\frac{\partial \mathcal{L}}{\partial y}$$
$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial w} = x\frac{\partial \mathcal{L}}{\partial y}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial b} = \frac{\partial \mathcal{L}}{\partial y}$$

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。 $\frac{\partial \mathcal{L}}{\partial x}$ はなぜ求めたのだろうか。これは x が前の層の出力であるからである。つまり、 $\frac{\partial \mathcal{L}}{\partial x}$ はこの前の層にとっては既知とされた $\frac{\partial \mathcal{L}}{\partial y}$ に対応するのだ。

3.3.3. 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。SGD の更新式もそのまま使える。

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$$

バイアスについては、同様に計算すると

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{B}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}}$$

であるが、バイアスは各ニューロンに 1 つしかないので、 $\boldsymbol{B}$ の各行は同じ値を持つ。そのために、列方向に総和を取る。したがって、以下のように更新幅を決めることにする。

$$[\boldsymbol{B}]_{ij} \leftarrow [\boldsymbol{B}]_{ij} - \eta \sum_{1 \leq k \leq n} \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{Y}} \right]_{kj}$$

場合によっては、 η の代わりに $\frac{\eta}{n}$ を用いることもある。

3.3.4. 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。

3.3.5. 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。

3.3.6. 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。

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

3.3.7. 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。

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