課題 4

浦川樹

2022 年 1 月 28 日

1 課題内容

1.1 概要

MNIST のテスト画像 1 枚を入力とし，3 層ニューラルネットワークを用いて，0 から 9 の数字 1 つを出力する．必修課題の内容の他に，活性化関数 ReLU(発展課題 A1) や最適化手法の慣性項 (Momentum) 付き SGD と AdaGrad(発展課題 A4) を実装した．

1.2 誤差逆伝播法の数学的内容

誤差逆伝播法による勾配計算を数学的に記述する．基本的に実装は教科書に記載されている数式に準拠して行ったが，誤差逆伝播法の実装のみ自ら計算した数式に則って実装したため，その式を示すためこの節を設けた．まず順伝播は次のように計算される．入力ベクトルの次元を d, 中間層のユニット数を M，クラス数を C とする．

$$a_{b,j}^{(1)} = \sum_{i=0}^{d} w_{ji}^{(1)} x_{b,i} (j = 1, \cdots, M)$$

$$y_{b,j}^{(1)} = h(a_{b,j}^{(1)}) (j = 1, \cdots, M)$$

$$a_{b,k}^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} y_{b,j}^{(1)} (k = 1, \cdots, C)$$

$$y_{b,k}^{(2)} = \sigma(a_{b,k}^{(2)})$$

ただし $x_{b,0} = y_{b,0}^{(1)} = 0$ と定義する．以降線形和の重み $w_{ji}^{(1)} (j = 1, \ldots, M, i = 0, \ldots, d)$ を (j, i) 成分とする $M \times (d+1)$ 行列を $W^{(1)}$，$w_{kj}^{(2)} (k = 1, \ldots, C, j = 0, \ldots, M)$ を (k, j) 成分とする $C \times (M+1)$ 行列を $W^{(2)}$ と表す．

$$W^{(1)} = \begin{pmatrix} w_{10}^{(1)} & w_{11}^{(1)} & \cdots & w_{1d}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & \cdots & w_{2d}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M0}^{(1)} & w_{M1}^{(1)} & \cdots & w_{Md}^{(1)} \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} w_{10}^{(2)} & w_{11}^{(2)} & \cdots & w_{1M}^{(2)} \\ w_{20}^{(2)} & w_{21}^{(2)} & \cdots & w_{2M}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{C0}^{(2)} & w_{C1}^{(2)} & \cdots & w_{CM}^{(2)} \end{pmatrix}$$

バッチ中のあるデータ b に対するクロスエントロピー誤差 E_b は

$$E_b = \sum_{k=1}^{C} -y_{b,k} \log y_{b,k}^{(2)}$$

で与えられる．よってバッチ全体の誤差 E は

$$E = \frac{1}{B} \sum_{b \in \mathscr{B}} E_b$$

である．ここで $\mathscr{B}$ はバッチのデータ全体の集合を意味する．以上の状況のもとで損失関数の誤差を求めたい．このとき次の式が成り立つ．

$$\frac{\partial E_b}{\partial a_{b,k}^{(2)}} = y_{b,k}^{(2)} - y_{b,k}$$

$$\frac{\partial E}{\partial W^{(2)}} = \frac{1}{B}^t \left(\frac{\partial E_b}{\partial a_{b,k}^{(2)}} \right) \cdot \left(y_{b,j}^{(1)} \right)$$

$$\left(\frac{\partial E_b}{\partial a_{b,j}^{(1)}} \right) = \left(h'(a_{b,j}^{(1)}) \right) \circ \left(\left(\frac{\partial E_b}{\partial a_{b,k}^{(2)}} \right) \cdot \tilde{W}^{(2)} \right)$$

$$\frac{\partial E}{\partial W^{(1)}} = \frac{1}{B}^t \left(\frac{\partial E_b}{\partial a_{b,j}^{(1)}} \right) \cdot (x_{b,i})$$

ここで $\frac{\partial E}{\partial W^{(2)}}$ は (k,j) 成分が $\left(\frac{\partial E}{\partial w_{kj}^{(2)}} \right)$ の $C \times (M+1)$ 行列，$\frac{\partial E}{\partial W^{(1)}}$ は (j,i) 成分が $\left(\frac{\partial E}{\partial w_{ji}^{(1)}} \right)$ の $M \times (d+1)$ 行列，$\left(\frac{\partial E_b}{\partial a_{b,k}^{(2)}} \right)$ は (b,k) 成分で表示された $B \times C$ 行列，$\left(\frac{\partial E_b}{\partial a_{b,j}^{(1)}} \right)$ は (b,j) 成分で表示された $B \times M$ 行列である．$\left(y_{b,j}^{(1)} \right)$ は (b,j) 成分で表示された $B \times (M+1)$ 行列，$(x_{b,i})$ は (b,i) 成分で表示された $B \times (d+1)$ 行列である．また $\left(h'(a_{b,j}^{(1)}) \right)$ は (b,j) 成分で表示された $B \times M$ 行列であり，$\tilde{W}^{(2)}$ は $W^{(2)}$ から 1 列目を取り除いた

$$\tilde{W}^{(2)} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & \cdots & w_{1M}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & \cdots & w_{2M}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{C1}^{(2)} & w_{C2}^{(2)} & \cdots & w_{CM}^{(2)} \end{pmatrix}$$

$C \times M$ 行列である．$\circ$ は行列のアダマール積を表す．これらの式を用いることで損失関数の勾配を計算することができる．これらの式に従って誤差逆伝播法の実装を行っている．

2 プログラムの仕様

2.1 プログラムの構成

ニューラルネットワークの実装の本体である ireco パッケージと各課題の実行プログラムからなる．各課題の実行プログラムは tests/ディレクトリに置かれている．

2.2 ireco パッケージ

ニューラルネットワークの順伝播や逆伝播などの主要部分が実装された neural モジュール，ニューラルネットワークの前処理や後処理で用いる関数を定義した utils モジュール，各最適化手法のクラスを実装した optimizer モジュールからなる．

2.3 各課題の実行プログラム

各課題内容を実現するプログラム kadai$i$.py($i = 1, 2, 3, 4, \mathrm{A1}, \mathrm{A4}$) が tests/ディレクトリに置かれている．

3 工夫点

3.1 パッケージ化

ニューラルネットワークの実装の部分をライブラリとして再利用できるようにパッケージ化した．パッケージの構造や関数の命名は keras を参考にして作成した．他の人が使うときに便利なように，外部から呼び出される関数にはできるだけコメントで関数の説明を書いた．ちなみに Visual Studio Code のインテリセンスでこの説明が表示されるようになっている．

3.2 ベクトル演算の使用

基本的にベクトルや行列の計算をする場合，for 文を使わないことを心がけた．ほとんどの計算は Numpy の提供する演算機能を用いて実装している．計算速度の向上を期待して行った．

3.3 誤差逆伝播法の数学的定式化

誤差逆伝播法の計算は教科書の計算式ではなく，自ら計算した式を用いた．教科書の式よりも簡潔な記述となっている．

4 問題点

4.1 関数で実装したことによる問題

ニューラルネットワークを関数で実装したために，関数を呼び出す際に同じ引数を毎回関数に渡さなければならない．それが少し面倒なのでクラスを用いるべきだった．

4.2 学習にかかる時間

学習にかなり時間がかかる．発展課題 A4 で最適化手法を追加する際に，最適化手法ごとにクラスを作りクラスのメソッドで最適化するように変更したのだが，この変更で格段に遅くなってしまった．もともとはベタ書きで計算式を書いていたので多少遅くなるのはしょうがないと思っていたが，想定よりも遅くなってしまった．クラスの実装方法に少し問題が有ると考えられる．

4.3 関数への予期しない入力

基本的に関数に入力される行列やベクトルが想定した形をしているかの検査はしていない．想定以外の入力に対しては Numpy の演算中にエラーが出ることがほとんどだと考えられるが，エラーが出ずに意味のない値を返すことも考えられる．