FUNAI 輪読会

ゼロから作るDeepLearning

5章 誤差逆伝播法 2/2

多田 瑛貴
公立はこだて未来大学 システム情報科学部
複雑系知能学科 複雑系コース 2年

前章で学んだこと: 逆伝播への理解

- 「計算グラフ」を用いて計算過程を「局所的な計算」の連なりとして表現
  - これを用いて順伝播・逆伝播を説明
  - 順伝播: 通常の計算を行う
  - 逆伝播: 微分(勾配)を求める
- 誤差逆伝播法のモチベーションを理解
  - 計算過程の途中で勾配計算(逆伝播)の結果を保持し
    他のパラメータに対する勾配計算に再利用することができる
  - **効率的な勾配計算を実現**

今回学ぶこと: 誤差逆伝播法の実装

- 様々なノードの順伝播・逆伝播を理解し
  Pythonを用いて実際に実装する
  - ニューラルネットワークを構成する「層（レイヤ）」として実装
    - 乗算レイヤと加算レイヤ
    - 活性化レイヤ （ReLU、Sigmoid）
    - Affine、Softmaxレイヤ
  - これらを用いて、 **ニューラルネットワークでの学習を実践する**
- 誤差逆伝播法の実装に誤りがないかの検証方法
  - 誤差逆伝播法は実装が複雑であり、ミスしやすい
  - 数値微分と比較して行う手法を紹介

5.4 単純なレイヤの実装

補足: 「クラス」とは

誤解を恐れずに説明すると…

変数の型であり

- いろんな種類の値を、複数保持できる
  - ＝ たくさんの変数を中に持っている
- 関数を紐付けられる (保持している値を直接利用できる)

適切な捉え方ではないが、とりあえず今はこの理解でOK

クラスのコンセプトは、オブジェクト指向プログラミングと同時に学ぶことになるはず

補足: 「クラス」の例

```
class Human: # □□□□□
    def __init__(self, name):                    # □□□□□□□□□□□
        self.name = name                         # □□□"name"□□□

    def greet(self):                             # □□□□□□□□
        print("Hello, I'm " + self.name)

peruki = Human("Teruki TADA")   # □□□□□□□peruki□□□□(□□□□□□□)□□□□□
print(peruki.name)              # > "Teruki TADA"
peruki.greet()                  # > "Hello, I'm Teruki TADA"
```

Pythonによる実装の方針

各レイヤは、pythonのクラスとして記述される

```
# □□□□□□□□□□□□□□□□
class Layer:
    def __init__(self):              # □□□□□□□□
        ...
    def forward(self, x, y):         # □□□
        ...
    def backward(self, dout):        # □□□
        ...
```

以下のようにインスタンスを作成
このインスタンスを使って、ノードの入出力を表現

```
# □□□□□□□□□□□□□□□
layer = Layer()
```

順伝播の実装の方針

2つの入力`x` `y`から順伝播し、`output`に代入

```
output = layer.forward(x, y)
```

例 (乗算レイヤの場合)

```
mul_layer = MulLayer()
apple = 100
apple_num = 2
output = mul_layer.forward(apple, apple_num)
print(output) # -> 200
```

逆伝播の実装の方針

入力`dout`から逆伝播し、`output1` `output2`に代入

```
output1, output2 = layer.backward(dout)
```

例 (乗算レイヤの場合)

```
dapple, dapple_num = mul_layer.backward(dout)
print(dapple, dapple_num) # -> 2.2, 110
```

乗算レイヤの実装

```
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout):
        dx = dout * self.y
        dy = dout * self.x
        return dx, dy
```

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=PHEIJTYaknW3

加算レイヤの実装

```
class AddLayer:
    def __init__(self):
        pass                        # □□□□□

    def forward(self, x, y):
        out = x + y

        return out

    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1

        return dx, dy
```

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=XmMRRTavkvxk

りんごの値段の問題を解いてみる

太郎くんはスーパーで1個100円のりんごを2個買った。支払う金額を求めよ。ただし、消費税は10%とする。

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=mUkVOzF3k7Re&line=1&uniqifier=1

5.5 活性化関数レイヤの実装

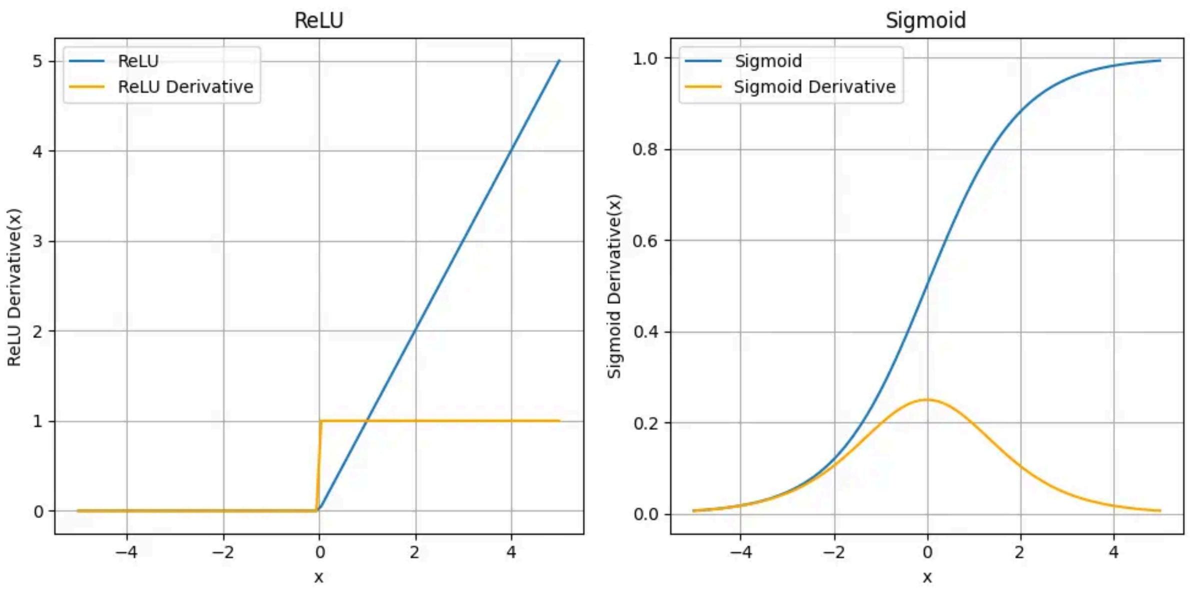
活性化関数とは

入力された値の総和を調整し、出力に変換する関数のこと

3-5

参照: 3.2 活性化関数

ReLU関数や**Sigmoid**関数がある



ReLUレイヤ

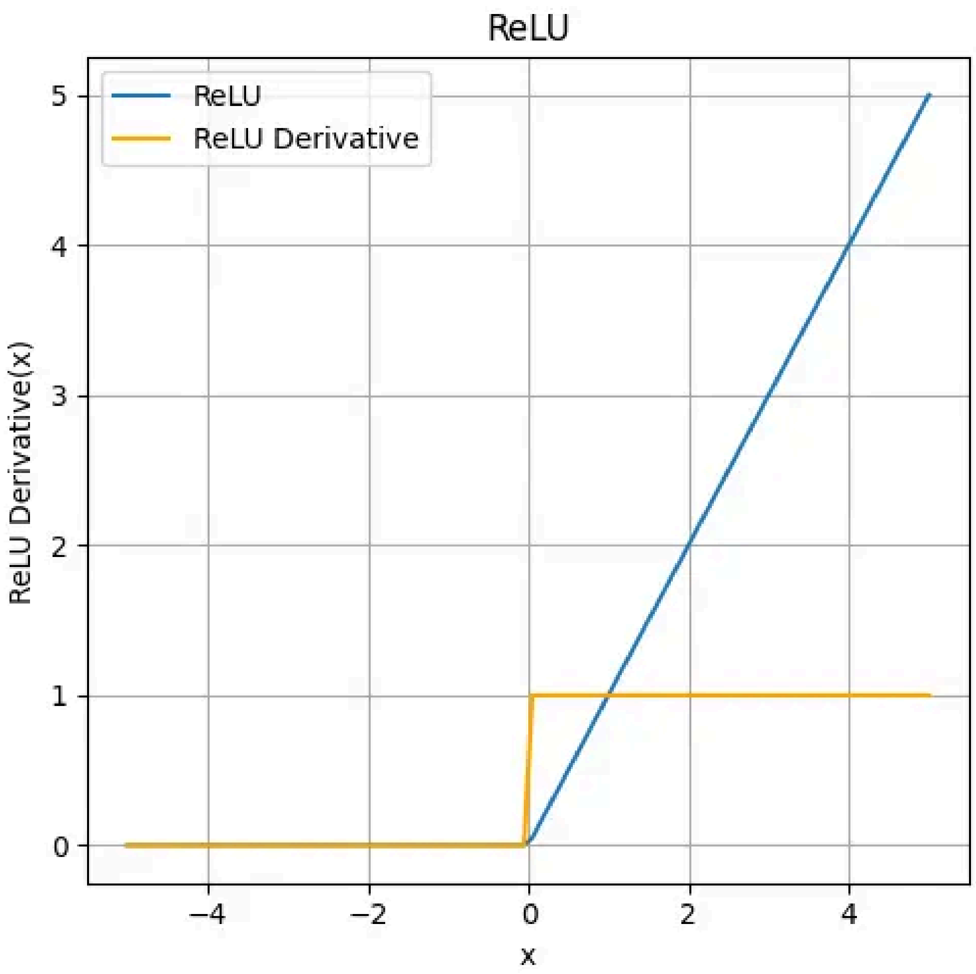
ReLU(Rectified Linear Unit)関数は以下の式で表される

$$y = \begin{cases} x(x > 0) \\ 0(x \leq 0) \end{cases}$$

グラフは右図の青線のようになる

橙線はその微分(後述)

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=U_uV2H99l1xq



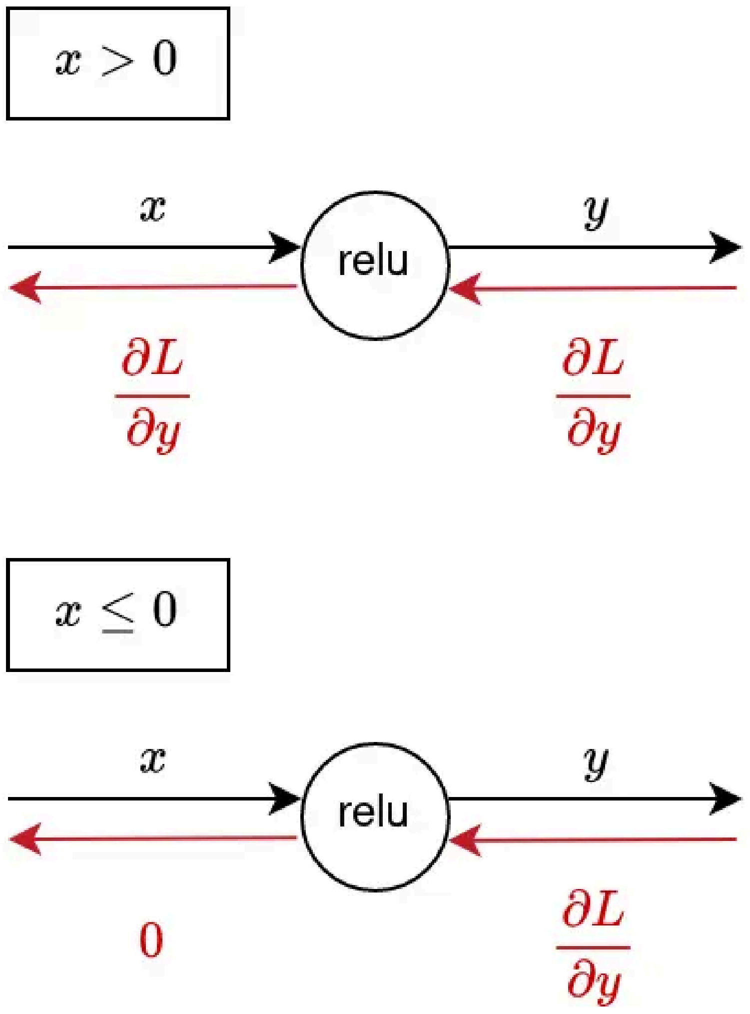
ReLUレイヤの勾配

ここで、xに関するyの微分は

$$\frac{\partial y}{\partial x} = \begin{cases} 1 (x > 0) \\ 0 (x \leq 0) \end{cases}$$

のようになる。つまり

- 順伝播時のxが0より大きければ
  逆伝播では上流の値をそのまま下流に返す
- xが0より小さければ、何も流さない
  (信号をストップする)



ReLUレイヤの実装

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=_vja_aBs3cwS

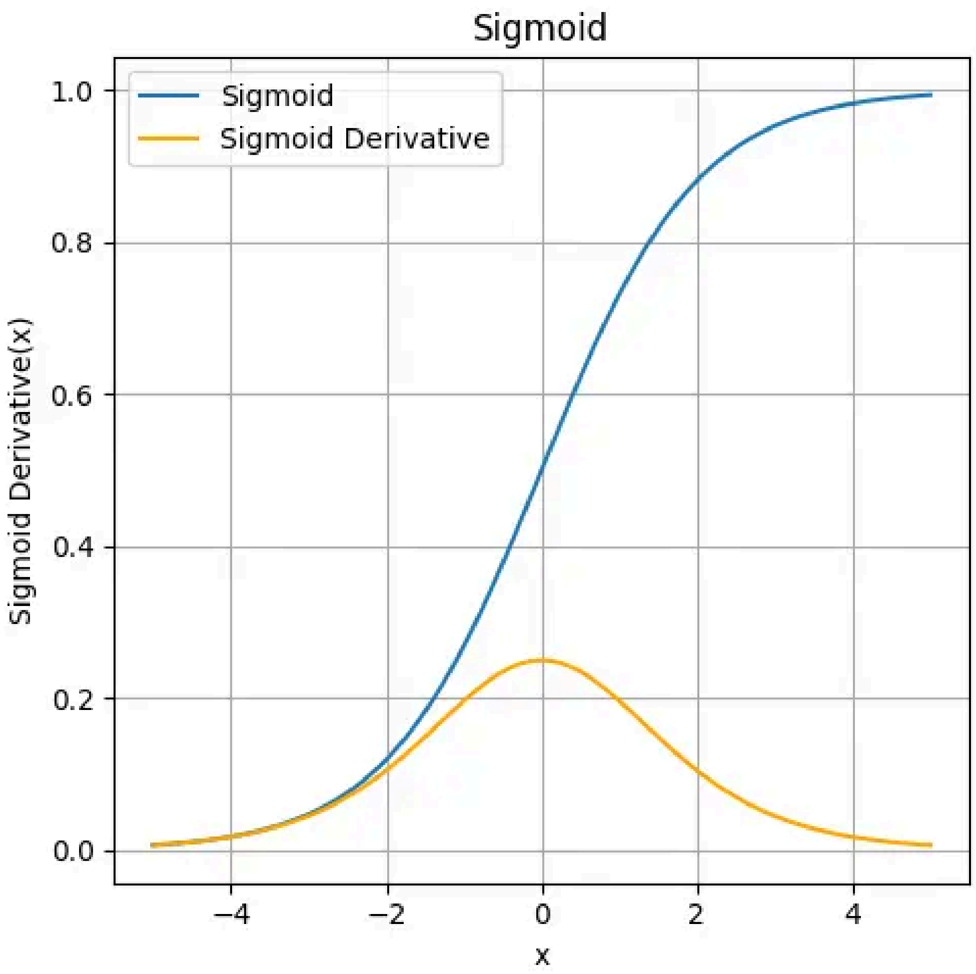
Sigmoidレイヤ

$$y = \frac{1}{1 + \exp(-x)}$$

グラフは右図の青線のようになる

橙線はその微分(後述)

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=2cMOXieflvWN

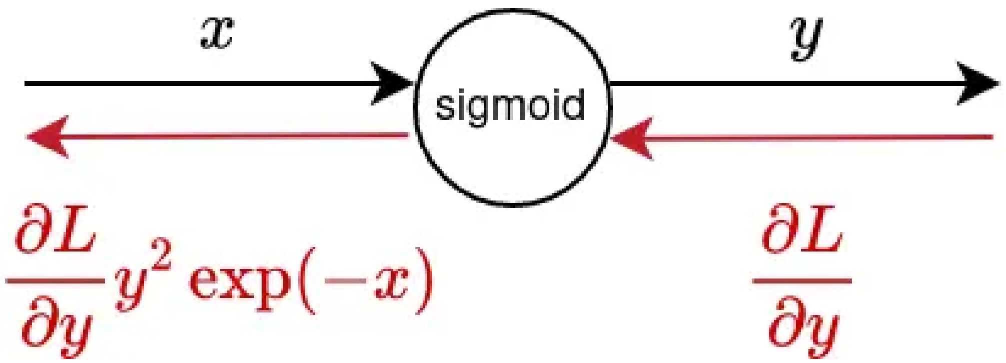


Sigmoidレイヤの勾配

ここで、xに関するyの微分は以下のようになる

$$\frac{\partial y}{\partial x} = y^2 \exp\left(-x\right)$$

導出は省略



Sigmoidレイヤの勾配の簡略化

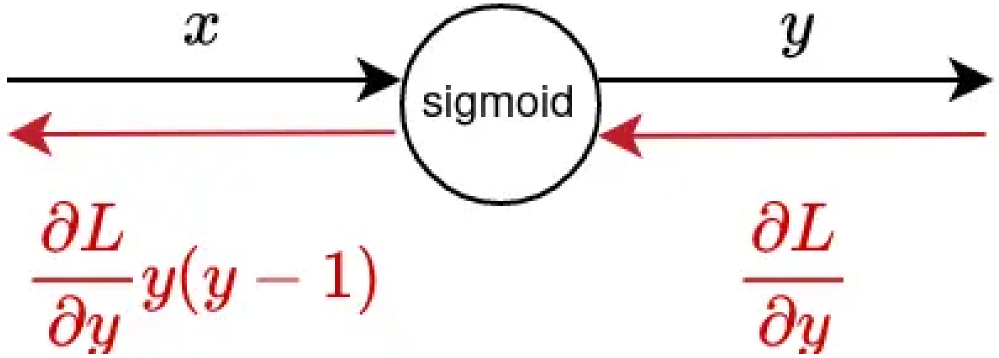
微分した式は、もっと簡略化にできる

$$
\begin{aligned}
y^2 \exp(-x) &= \frac{1}{(1+\exp(-x))^2} \exp(-x) \\
&= \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)} \\
&= y(1-y)
\end{aligned}
$$

Sigmoidレイヤの勾配

$$\frac{\partial y}{\partial x} = y(y-1)$$

最終的に
順伝播の出力(y)だけを使って
逆伝播の結果を計算することができる



Sigmoidレイヤの実装

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=nAUucqwRmNbj

5.6 Affine/Softmaxレイヤの実装

重み付きの和の計算

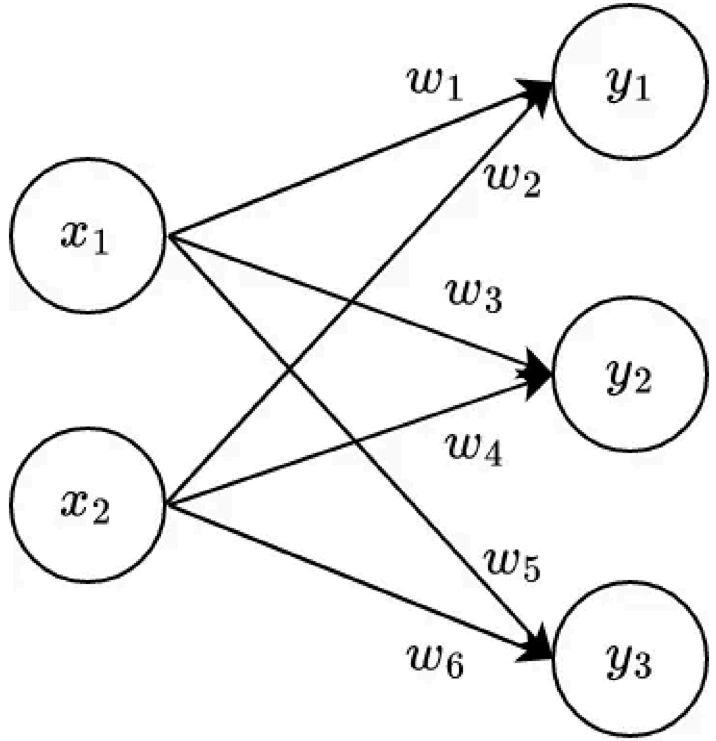
ニューラルネットワークの順伝播では「重み付きの信号の総和」を求める必要がある

参照: 3.3 多次元配列の計算

ここで用いられる行列計算を行うレイヤを**Affineレイヤ**と呼ぶことにする

いわゆるAffine変換と直接の関係はない、ただやっている計算が同じというだけ

$\mathbf{X}$: 入力、$\mathbf{W}$: 重みの行列、$\mathbf{Y}$: 出力



$$\mathbf{W} = \begin{pmatrix} w_1 & w_3 & w_5 \\ w_2 & w_4 & w_6 \end{pmatrix}$$

$$\begin{array}{cccc} \mathbf{X} & \mathbf{W} & = & \mathbf{Y} \\ 2 & 2 \times 3 & & 3 \end{array}$$

Affineレイヤの逆伝播

計算グラフは□5-24のようになる

この計算グラフを用いて逆伝播を導くと、以下のようになる

行列の各要素を書き下せば、行列でも今までのスカラー値と同じように求められる

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^T$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

導出方法をより詳しく知りたい方

Qiita: Affineレイヤの逆伝播を地道に成分計算する

https://qiita.com/yuyasat/items/d9cdd4401221df5375b6

「バッチ版」Affineレイヤについて

先程の実装では、入力するデータ$\mathbf{X}$は
1つのデータ(1次元のベクトル)だった

図5-25の示すように
N個のデータ (=バッチ) でも同様に扱うことができる

このとき、$\mathbf{X}$と $\mathbf{Y}$はデータのベクトルを結合した行列になる

バッチ版Affineレイヤの実装

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=UVqN02cG3vHT

Softmax関数について

Softmax関数ってなんだったっけ

入力された値を正規化して出力

正規化: 出力の総和が1になるように変形

この計算を行うレイヤを**Softmaxレイヤ**と呼ぶことにする

```
□□: [4.96578994 4.60163166 3.22858034 4.39713805 0.3080219 ]
□□: [0.40873582 0.28398197 0.07194194 0.23146233 0.00387793]
```

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=qoe2pPIXpvAj

Softmaxレイヤ導入のモチベーション

ニューラルネットワークで行う処理には「推論」「学習」の2つのフェーズがある

- **推論**: 答えを出す処理
- **学習**: 答えを出すためのパラメータを調整する処理

推論の段階では、出力の最大値にだけ興味があるので、正規化は求められない
→Softmaxレイヤは使わず、Affineレイヤの出力(スコア)で処理を終了

Softmaxレイヤは**学習の段階**で求められる
→損失関数を適用し、性能を評価するため

そういえばMNISTを使った学習では、教師データとして

t = [0,0,0,1,0,...,0,0]のようにone-hotエンコーディングされたベクトルが用いられていた

Softmax-with-Lossレイヤ

損失関数である**交差エントロピー誤差**を含めたレイヤとして実装

5-29

非常に複雑なため、書籍では省略されている (**付録A**を参照)
ここでは結果のみを紹介

Softmax-with-Lossレイヤの逆伝播

簡略化し、`図5-30`のように考えると

入力を$\mathbf{A}(a_1, a_2, a_3 \dots)$
$\mathbf{A}$を正規化したものを$\mathbf{Y}(y_1, y_2, y_3 \dots)$とすると
逆伝播は教師データ$\mathbf{T}(t_1, t_2, t_3 \dots)$に対して

$$\frac{\partial L}{\partial \mathbf{A}} = \mathbf{Y} - \mathbf{T}$$

つまり$y_i - t_i$を求めれば良い

非常に簡単になった！

Softmax-with-Lossレイヤの逆伝播

交差エントロピー誤差は、Softmax関数の損失関数として
都合の良いように設計されている (偶然の産物ではない)

一方で、もう一つの損失関数である2乗和誤差は、恒等関数に対して都合が良い

Softmax-with-Lossレイヤの実装

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=YKTA7C3D3xdS

5.7 誤差逆伝播法の実装

ニューラルネットワークの学習の全体図

前提: 学習とは

ニューラルネットワークには「重み」「バイアス」があり
それらを訓練データに適応するように調整していく

ニューラルネットワークの学習の全体図

学習のステップ

1. **ミニバッチ**
   訓練データの中からランダムに1つデータを選び出す
2. **勾配の算出**
   各重みパラメータに関する損失関数の勾配を求める
   誤差逆伝播法はこのステップで、勾配を効率的に求めるために用いられる
3. **パラメータの更新**
   重みパラメータを勾配方向に微少量だけ更新する
4. **繰り返し**
   ステップ1,2,3を繰り返す

誤差逆伝播法に対応したニューラルネットワーク

全体のレイヤの構造は5-28のようになる

実装は以下リンクのようになる

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=V_By3sHX3Rtl

誤差逆伝播法の勾配確認

誤差逆伝播法は数値微分と比べ、実装が複雑であり、ミスも起こりやすい

そこで、数値微分による勾配の計算結果と比較し、実装に誤りがないかを確認する(このような作業を**勾配確認**という)

誤差逆伝播法で学習を行ってみる

勾配の正しさを確認し、実際にMNISTを使った学習を行う

勾配確認

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=Gn7VREiD3USR

学習

https://colab.research.google.com/drive/1aGsxpOVapW9NOHPsOXsjTmKFxdC62Z96#scrollTo=7NPbvzGR3dnK

まとめ

- 乗算・加算だけでなく、ニューラルネットワークの実装に求められる
  複合的な関数や行列計算にも
  計算グラフの考え方に基づいて逆伝播を適用できる
- ニューラルネットワークの構成要素をレイヤとして実装することで、
  逆伝播の適用により勾配の計算を効率的に求められる
- 誤差逆伝播法は実装が難しいことから、数値微分と計算結果と比較することで
  実装の正しさを検証する「勾配確認」が行われる

次章について

ニューラルネットワークのよりよい学習に重要なテクニックを学ぶ

- パラメータの更新方法の再検討
  今までの方法(SGD)の課題と、
  その解決策となるMomentum, AdaGrad, Adamについて理解
- 重みの初期値の設定の再検討
  - 重みの初期値による学習結果への影響を観察し、重要性を理解
  - Xavierの初期値やHeの初期値などの手法について理解

次章について

ニューラルネットワークのよりよい学習に重要なテクニックを学ぶ

- Batch Normalizationの導入による学習の改善
- 過学習を抑制する手法
  - Weight Decay, Dropoutについて理解
- 重み・バイアス以外のパラメータ(ハイパーパラメータ)の調整方法