深層学習 Day1

秋葉洋哉

2024 年 8 月 4 日

1　ニューラルネットワーク

1.1　識別モデルと生成モデル

識別モデルとは、データを目的のクラスに分類するための方法である。一方で、生成モデルとは、特定のクラスのデータを生成するための方法である。例えば、犬という画像を入力で与えた場合にそれが犬である確率を出力するのが識別モデルであり、犬というクラスを与えた場合に犬の画像を生成するのが生成モデルである。

識別モデルは、基本的に高次元 (=データ量の多いもの) から低次元 (=データ量の少ないもの) へと変換することに長けており、画像認識等で用いられる。一方で、生成モデルでは、低次元から高次元へと変換することに長けており、画像の生成等で用いられる。

識別モデルでは、決定木・ロジスティック回帰・SVM・ニューラルネットワークなどがある。生成モデルでは、隠れマルコフモデル・ベイジアンネットワーク・変分オートエンコーダー (VAE)・敵対的生成ネットワーク (GAN) などがある。

生成モデルは、ベイズの定理を用いて識別器を作成する方法である。入力が生まれる確率と、クラスが生まれる確率を与え、それを用いてクラスを推定する。つまり、

$$P(x|C_k)P(C_k) \tag{1}$$

によって各クラスの生起確率を求め、最大のクラスを選択するというものである。生成モデルではデータを人工的に生成できることが特徴である。ベイズの定理は、

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)} \tag{2}$$

で表される。

1.2　万能近似定理

万能近似定理とは、任意の連続関数をニューラルネットワークで近似できるという定理である。この定理により、ニューラルネットワークは、非常に高い表現力を持つことがわかる。この定理により、ニューラルネットワークは、多くの分野で利用されている。

1.3 ニューラルネットワークの概要

ニューラルネットワークは、脳の神経細胞を模倣したモデルである。ニューラルネットワークは、入力層・中間層・出力層から構成される。入力層は、入力データを受け取る層であり、中間層は、入力データを変換する層であり、出力層は、データを出力する層である。多層パーセプトロン (MLP)・畳み込みニューラルネットワーク (CNN)・再帰型ニューラルネットワーク (RNN) といった種類を有する。

確認テスト

Q: ディープラーニングは、結局なにをやろうとしているか 2 行以内で述べよ。また、最適化する値は何か。

A: ディープラーニングは、多層のニューラルネットワークを用いて、高度な特徴を抽出し、複雑な問題を解決することを目的としている。最適化する値は、誤差関数を最小化する重み w とバイアス b である。

Q: 入力層 2 ノード 1 層・中間層 3 ノード 2 層・出力層 1 ノード 1 層のネットワークを書け。

A: 図 1 を参照。

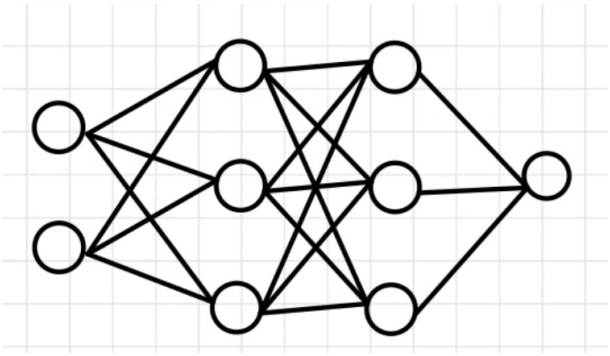


図 1: 入力層 2 ノード 1 層・中間層 3 ノード 2 層・出力層 1 ノード 1 層のネットワーク

ニューラルネットワークは、自動売買・チャットボット・画像認識・音声認識・自然言語処理・囲碁将棋 AI など、多くの分野で利用されている。

ニューラルネットワークの 1 つのノードに対して、入力 x_i, 重み w_i, バイアス b, 総入力 u, 出力 z, 活性化関数 f とすると、

$$u = \sum_{i=1}^{n} w_i x_i + b \tag{3}$$

$$z = f(u) \tag{4}$$

で表される。この出力は次のノードの入力として用いられることになる。

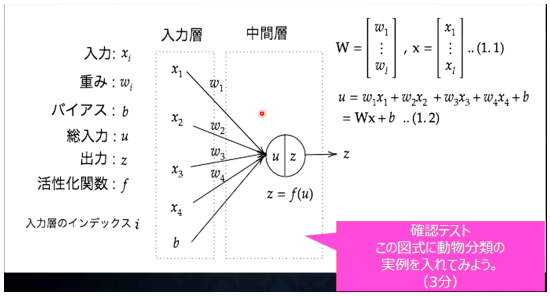
確認テスト

Q: 図 2a に動物分類の実例を入れろ。

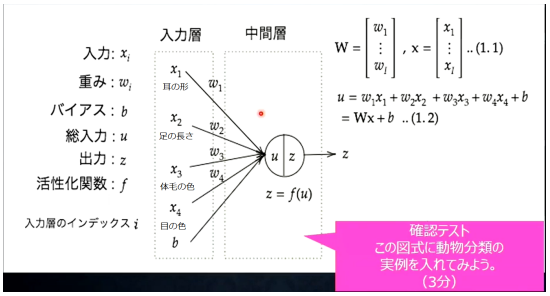
A: 図 2b を参照

Q: 図 3a の数式を書け。

A: 図 3b を参照

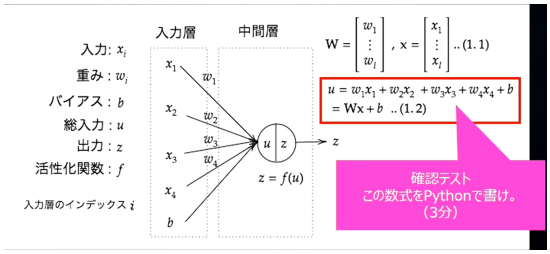


(a)



(b)

図 2



(a)

```python
x = np.array([109, 455, 322, 504]) #入力値
W = np.array([[0.12, 0.20, 0.24, 0.55],
              [0.34, 0.42, 0.14, 0.95],
              [0.41, 0.03, 0.94, 0.22],
              [0.40, 0.22, 0.41, 0.80]]) #重み
b = np.array(5.51) #バイアス
u = np.dot(x, W) + b #総入力
print(u)

```

```
[506.91 338.95 604.69 971.75]
```

(b)

図 3

中間層と出力層で用いられる活性化関数に違いがある。中間層では、ReLU 関数、シグモイド関数、ステップ関数といった関数が用いられることが多い。出力層では、ソフトマックス関数 (多値分類)、シグモイド関数 (二値分類)、恒等関数 (回帰) といった関数が用いられることが多い。出力層において用いる活性化関数と誤差関数は、扱う問題に応じて選択する必要がある。

1. 回帰問題の場合 → 恒等関数・二乗和誤差
2. 2 値分類問題の場合 → シグモイド関数・交差エントロピー誤差
3. 多クラス分類問題の場合 → ソフトマックス関数・交差エントロピー誤差

1.4 活性化関数

活性化関数とは、入力を出力に変換する関数である。活性化関数には、ステップ関数・シグモイド関数・ReLU 関数・恒等関数・ソフトマックス関数などがある。

1.4.1 ステップ関数

閾値を超えたら発火する関数で、0 か 1 を示す。Python のサンプルコードを示す。

```
def step_function(x):
    if x > 0:
        return 1
    else:
        return 0
```

1.4.2 シグモイド関数

勾配消失問題がある関数で、0 から 1 の値を示す。Python のサンプルコードを示す。

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

1.4.3 ReLU 関数

勾配消失問題を回避し、スパース化によって過学習を防ぐ関数である。Python のサンプルコードを示す。

```
def relu(x):
    return np.maximum(0, x)
```

1.4.4 恒等関数

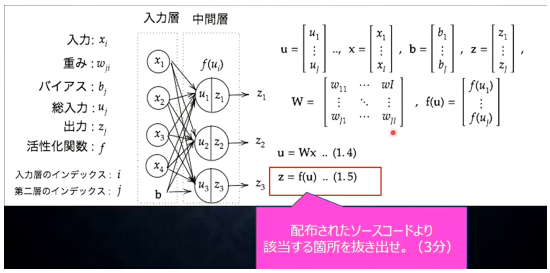
入力をそのまま出力する関数である。Python のサンプルコードを示す。

```
def identity_function(x):
    return x
```

確認テスト

Q: 図 4a の赤枠に該当する箇所をソースコードから抜き出せ。

A: 図 4b を参照



(a)

```
30 # 中間層出力
31 z = functions.sigmoid(u)
```

(b)

図 4

1.4.5　ソフトマックス関数

出力の総和が 1 になる関数である。数式で表すと、以下の通りとなる。

$$f(i, u) = \frac{\exp(u_i)}{\sum_{k=1}^{K} \exp(u_k)} \tag{5}$$

Python のサンプルコードを示す。

```
def softmax(x):
    return np.exp(x) / np.sum(np.exp(x))
```

確認テスト

Q: 図 5a の丸 1 から丸 3 までの数式に該当するソースコードを示し、1 行ずつ処理の説明をせよ。

A: 図 5b を参照。それぞれの説明は以下の通り。

①. 2 層目の総出力 u_2 を入力として、ソフトマックス関数を適用する。

②. ソフトマックス関数の分子を計算する。

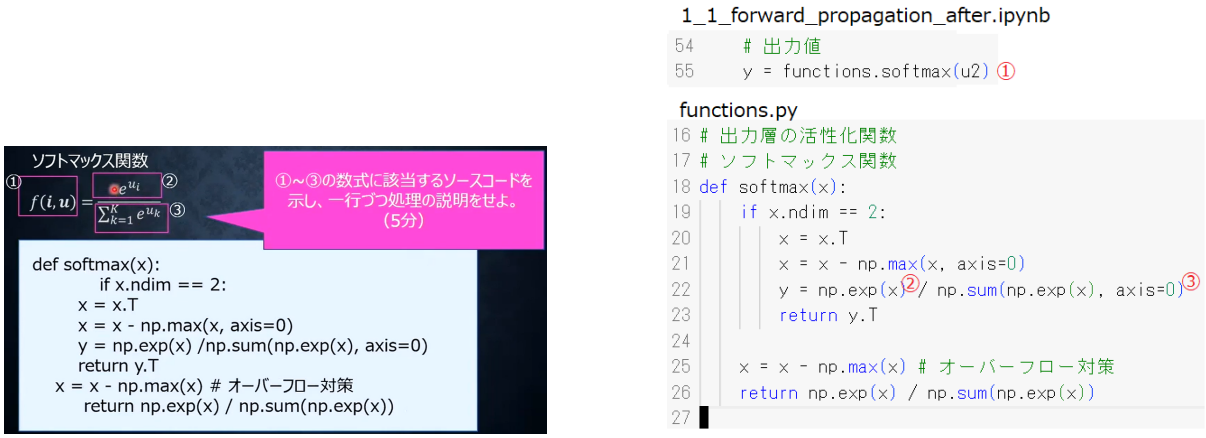
③. ソフトマックス関数の分母を計算する。

1.5　誤差関数

1.5.1　二乗和誤差

二乗和誤差とは、予測値と正解値の差の 2 乗和を 2 で割ったものである。数式で表すと、以下の通りとなる。

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{w} (y_i - t_i)^2 \tag{6}$$



(a) (b)

図 5

確認テスト

Q: 二乗和誤差は、なぜ単なる引き算ではなく二乗するのか述べよ。

A: 二乗和誤差は、予測値と正解値の差を取ることで、正解値との誤差を表す。二乗することで、正解値との誤差が大きい場合に誤差が大きくなり、正解値との誤差が小さい場合に誤差が小さくなるようにするためである。

Q : 二乗和誤差の $\frac{1}{2}$ はどういう意味を持つか述べよ。

A: $\sum_{i=1}^{w}(y_i - t_i)^2$ の微分値は以下の通りである。

$$\frac{\partial E}{\partial y_i} = \frac{\partial}{\partial y_i}\left(\sum_{i=1}^{n}(y_i - t_i)^2\right) \tag{7}$$

$$= 2(y_i - t_i) \tag{8}$$

二乗和誤差の定義式で $\frac{1}{2}$ を掛けるのは、前に出た 2 を消し、微分値を $(y_i - t_i)$ として、計算を簡単にするためである。

平均二乗和誤差と二乗和誤差

機械学習でよく用いられる平均二乗誤差 (Mean Squared Error: MSE) は、二乗和誤差の分母 2 をデータ数に変えて計算したものである。数式で表すと、以下の通りとなる。

$$E(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{w}(y_i - t_i)^2 \tag{9}$$

平均二乗和誤差は、データ数によって誤差のスケールが変わるため、データ数が異なる場合でも誤差を比較することができる。一方で、二乗和誤差は、データ数によって誤差のスケールが変わらないため、データ数が異なる場合は誤差を比較することができないが、計算が簡単という特徴がある。

1.5.2 交差エントロピー誤差

交差エントロピー誤差とは、2 つの確率分布の違いを表す指標である。数式で表すと、以下の通りとなる。

$$E(\mathbf{w}) = -\sum_{i=1}^{w} t_i log(y_i) \tag{10}$$

ただし、t_i は正解ラベル、y_i は予測値を表す。この式を例を示しながら説明する。写真に写っている動物が犬であるとする。分類器は、犬・猫・ネズミを見分けることができるとし、いずれかのクラスの確率を出力する。今、犬・猫・ネズミのそれぞれの確率を 0.8, 0.1, 0.1 であると予測した場合、交差エントロピー誤差は、

$$E = -1.0 * \log(0.8) - 0.0 * \log(0.1) - 0.0 * \log(0.1) \tag{11}$$

$$= 0.223 \tag{12}$$

となる。次に、犬・猫・ネズミのそれぞれの確率を 0.4, 0.3, 0.3 であると予測した場合、交差エントロピー誤差は、

$$E = -1.0 * \log(0.4) - 0.0 * \log(0.3) - 0.0 * \log(0.3) \tag{13}$$

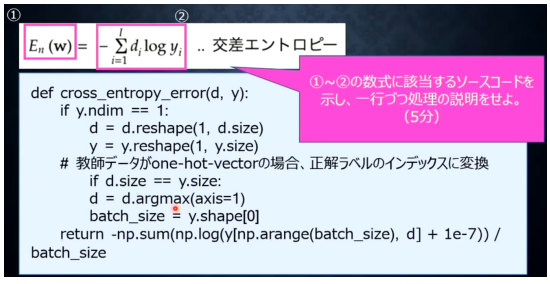
$$= 0.52 \tag{14}$$

となる。このように、交差エントロピー誤差は、予測が正解に近いほど小さくなり、予測が不正解に近いほど大きくなる。これは直観的にも誤差関数として交差エントロピー誤差が適していることが分かる。

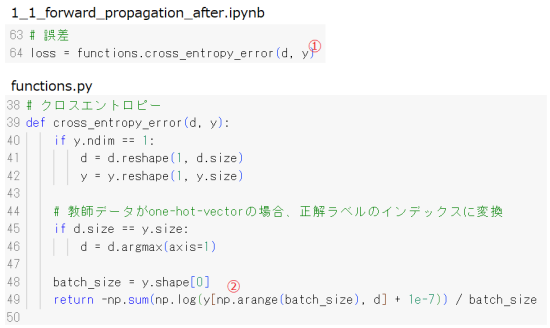
確認テスト

Q: 図 6a の丸 1 から丸 2 までの数式に該当するソースコードを示し、1 行ずつ処理の説明をせよ。
A: 図 6b を参照。それぞれの説明は以下の通り。

①. 正解値 d, 予測値 y を入力を与えて、交差エントロピー誤差を呼び出す。
②. $\log y^d = d \log y$ として、numpy の log 関数を用いて計算する。1.0d-7 を加えるのは、log(0) を防ぐためである。



(a)



(b)

図 6

1.5.3 カテゴリカル交差エントロピー誤差

カテゴリカル交差エントロピー誤差とは、多クラス分類問題において用いられる誤差関数である。数式で表すと、以下の通りとなる。

$$E = -\sum_{j=1}^{L-1}\sum_{i=1}^{M-1} t_{ji}\log(y_{ji}) \tag{15}$$

ただし、t_i は正解ラベル、y_i は予測値、L はクラス数、M はクラス数を表す。交差エントロピー誤差をクラス間で拡張し、すべてを足し合わせている。

1.6 勾配降下法

勾配降下法とは、導関数を用いて数値的に方程式の解を求めるための方法である。深層学習では、学習を通して誤差関数を最小にするパラメータ $\mathbf{w}$ を求めることが目的であり、勾配降下法はそのための手法の一つとして用いられる。勾配降下法は、以下の手順で行われる。

1. パラメータ $\mathbf{w}$ を初期化する
2. 誤差関数 $E(\mathbf{w})$ を最小化するために、$\mathbf{w}$ を更新する
3. 収束するまで 2 を繰り返す

数式で表すと、以下の通りとなる。

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon E(\mathbf{w}) \tag{16}$$

ここで、ϵ は学習率を表す。学習率は、パラメータの更新量を調整するためのハイパーパラメータであり、大きすぎると発散し、小さすぎると収束が遅くなるため、適切な値を設定する必要がある。勾配降下法には様々な種類があり、Momentum・AdaGrad・Adadelta・Adam などがある。

確認テスト

Q: 式 (16) に該当するソースコードを示せ。
A: 図 7 を参照

1_3_stochastic_gradient_descent.ipynb

```
110 # 勾配降下の繰り返し
111 for dataset in random_datasets:
112     x, d = dataset['x'], dataset['d']
113     z1, y = forward(network, x)
114     grad = backward(x, d, z1, y)
115     # パラメータに勾配適用
116     for key in ('W1', 'W2', 'b1', 'b2'):
117         network[key]  -= learning_rate * grad[key]
118
```

図 7

1.7 確率的勾配降下法 (SGD)

確率的勾配降下法とは、勾配降下法の一種であり、毎回違う学習データをランダムに選んで行う方法である。計算コストを下げたり、局所解に陥ることを防ぐために用いられる。勾配降下法を用いた深層学習モデルの学習データの選び方は、

1. バッチ学習
2. ミニバッチ学習
3. オンライン学習

の 3 つが挙げられる。

バッチ学習は、エポック毎に全てのデータを使って傾きを逐次求め、最適解を見つける方法である。学習結果が安定しやすいという特徴を持つ。新たな訓練データが追加された場合に、再度全データを用いて計算を行う必要があるため、計算コストが高い。

ミニバッチ学習は、すべてのデータの中から一部を取り出して学習する方法で、局所最適解を回避しやすいという特徴を持つ。バッチ学習とオンライン学習の中間の特徴を持つ。すべてのデータで学習し終えた時に 1 エポックが終了し、重みが更新される。

オンライン学習は、すべてのデータの中から 1 つを取り出して学習する方法で、ミニバッチ学習よりも局所最適解を回避しやすいという特徴を持つ。計算コストが低いが、学習結果が不安定であるという特徴を持つ。

深層学習のモデリングにおいては、ミニバッチ学習が最も一般的に用いられる。その理由として、確率的勾配法のメリットを損なわずに、計算資源を有効利用することができるからである。例えば、1000 万件の学習データがあった場合、それをバッチ学習で学習しようとすると、1000 万件のデータを一度に計算する必要がある。現代のコンピュータでは、1000 万件のデータの計算には莫大な時間を要してしまう。そのため 1000 件ずつのミニバッチを作成し、それを 10000 回繰り返す際に、他の資源でも同時平行で計算することで時間を短縮させる。これには、CPU のスレッド並列化や GPU の SIMD 並列化という技術が用いられる。

確認テスト

Q: $\mathbf{w^{(t+1)}} = \mathbf{w^{(t)}} - \epsilon \nabla E_t$ の数式の意味を図に書いて説明せよ。
A: 図 8 を参照

1.8 誤差逆伝播法

勾配降下法を用いて、重みとバイアスを更新する際に用いられる方法が誤差逆伝播法である。誤差逆伝播法は、出力層から入力層に向かって、誤差を逆伝播させることで、各層の重みとバイアスを更新する方法である。誤差逆伝播法は、以下の手順で行われる。

1. 順伝播
2. 誤差逆伝播
3. 重みとバイアスの更新

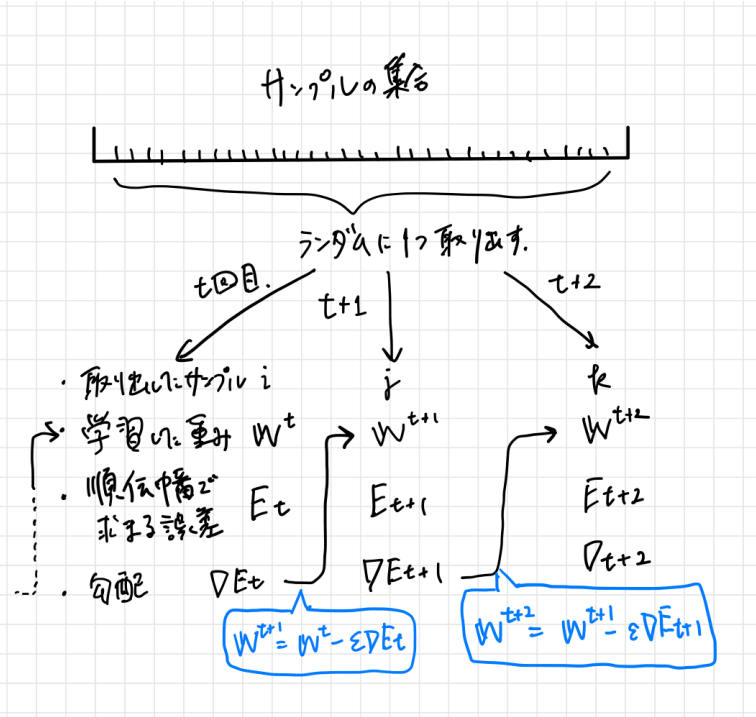


図 8

数式で表すと、以下の通りとなる。

$$w \leftarrow w - \epsilon \frac{\partial E}{\partial w} \tag{17}$$

$$b \leftarrow b - \epsilon \frac{\partial E}{\partial b} \tag{18}$$

ただし、w は重み、b はバイアス、ϵ は学習率、E は誤差関数を表す。勾配を求めることができれば、上式を用いて重みとバイアスを更新することができる。

以下では、ノード i における前ノード j との間の重み w_{ij} の勾配を求める。まず、$\frac{\partial E}{\partial w_{ij}}$ を求める。

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial u_i} \frac{\partial u_i}{\partial w_{ij}} \tag{19}$$

u_i は、ノード i の総入力を表し、

$$u_i = \sum_{j=1}^{m} w_{ij} y_j + b_i \tag{20}$$

であるから、$\frac{\partial u_i}{\partial w_{ij}}$ は、

$$\frac{\partial u_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{j=1}^{m} w_{ij} y_j + b_i \right) \tag{21}$$

$$= y_j \tag{22}$$

となる。また、$\frac{\partial E}{\partial u_i}$ は、

$$\frac{\partial E}{\partial u_i} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial u_i} \tag{23}$$

で表される。ここで、E が二乗和誤差の場合、

$$\frac{\partial E}{\partial y_i} = \frac{\partial}{\partial y_i}\left(\frac{1}{2}\left(\sum_{i=1}^{n}(y_i - t_i)^2\right)\right) \tag{24}$$

$$= y_i - t_i \tag{25}$$

となる。また、y_i が恒等関数の場合、

$$\frac{\partial y_i}{\partial u_i} = \frac{\partial}{\partial u_i}u_i \tag{26}$$

$$= 1 \tag{27}$$

となる。以上より、$\frac{\partial E}{\partial u_i}$ は、

$$\frac{\partial E}{\partial u_i} = (y_i - t_i) \times 1 = y_i - t_i \tag{28}$$

となる。よって、$\frac{\partial E}{\partial w_{ij}}$ は、

$$\frac{\partial E}{\partial w_{ij}} = (y_i - t_i)y_j \tag{29}$$

となる。同様に、バイアス b_i についても、

$$\frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial u_i}\frac{\partial u_i}{\partial b_i} = y_i - t_i \tag{30}$$

となる。以上より、重みとバイアスの更新式は、

$$w_{ij} \leftarrow w_{ij} - \epsilon(y_i - t_i)y_j \tag{31}$$

$$b_i \leftarrow b_i - \epsilon(y_i - t_i) \tag{32}$$

となる。これを、ノード i におけるすべての前ノード j について計算するために、行列表現で表すと、

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon\mathbf{y}^T(y_i - t_i) \tag{33}$$

$$\mathbf{b} \leftarrow \mathbf{b} - \epsilon(y_i - t_i) \tag{34}$$

となる。ここで、$\mathbf{y}$ を転置しているのは、$\mathbf{w}$ と形状を合わせるためである。

図 9 に、誤差逆伝播の概要を示す。

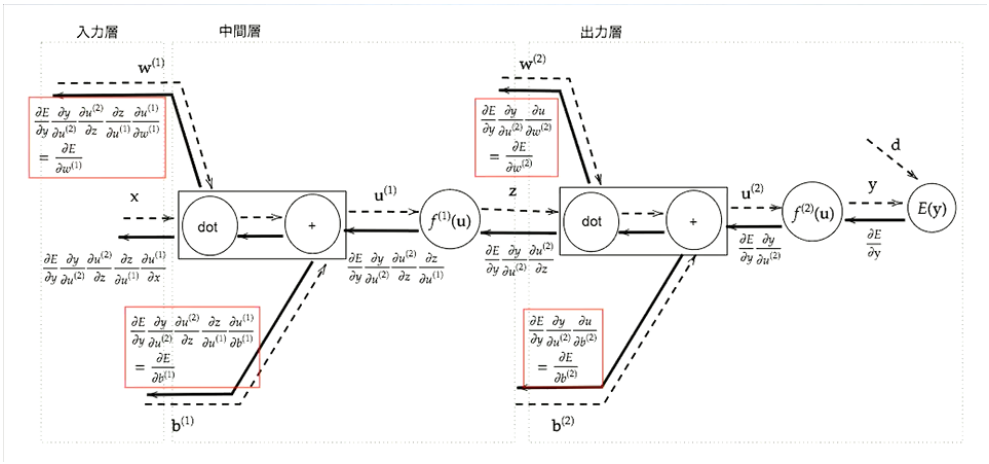


図 9: 誤差逆伝播法における勾配

確認テスト

Q: 誤差逆伝搬法では不要な再帰的処理を避けることができる。すでに行った計算結果を保持しているソースコードを抽出せよ。

A: 図 10a を参照

Q: $\frac{\partial E}{\partial y}\frac{\partial y}{\partial u}$ と、$\frac{\partial E}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial w_{ji}^{(2)}}$ に該当するソースコードを探せ。

A: 図 10b を参照

```python
# 誤差逆伝播
def backward(x, d, z1, y):
    # print("¥n##### 誤差逆伝播開始 #####")

    grad = {}

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    # 出力層でのデルタ
    delta2 = functions.d_mean_squared_error(d, y)

    # b2の勾配
    grad['b2'] = np.sum(delta2, axis=0)

    # W2の勾配
    grad['W2'] = np.dot(z1.T, delta2)

    # 中間層でのデルタ
    #delta1 = np.dot(delta2, W2.T) * functions.d_relu(z1)

    ## 試してみよう
    delta1 = np.dot(delta2, W2.T) * functions.d_sigmoid(z1)

    delta1 = delta1[np.newaxis, :]
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    x = x[np.newaxis, :]
    # W1の勾配
    grad['W1'] = np.dot(x.T, delta1)
```

(a)

```python
# 誤差逆伝播
def backward(x, d, z1, y):
    # print("¥n##### 誤差逆伝播開始 #####")

    grad = {}

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    # 出力層でのデルタ
    delta2 = functions.d_mean_squared_error(d, y)

    # b2の勾配
    grad['b2'] = np.sum(delta2, axis=0)

    # W2の勾配
    grad['W2'] = np.dot(z1.T, delta2)

    # 中間層でのデルタ
    #delta1 = np.dot(delta2, W2.T) * functions.d_relu(z1)

    ## 試してみよう  ∂E/∂y ∂y/∂u に対応
    delta1 = np.dot(delta2, W2.T) * functions.d_sigmoid(z1)

    delta1 = delta1[np.newaxis, :]
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    x = x[np.newaxis, :]
    # W1の勾配  ∂E/∂y ∂y/∂u ∂u/∂w_ji^(2) に対応
    grad['W1'] = np.dot(x.T, delta1)
```

(b)

図 10

計算グラフ

計算グラフとは計算の過程をグラフで視覚化したものである。これを用いることで、複雑な偏微分の連なりを分解し、出力へ影響を与えるノードを絞りこむことができるようになる。計算グラフは複数の「ノード」と「エッジ」から構成される。例えば、a,b,c,d,e,f をノードとし、

$$e = ab \tag{35}$$
$$f = cd \tag{36}$$
$$g = e + f \tag{37}$$

を計算グラフで表すと、図 11 の青字ようになる。これは順伝播を表す。
ここで、最終的な出力である g が、それぞれのノードの影響をどの程度受けるか知りたいとする。偏微分について以下が成立することは明白である。

$$f(a,b) = ab \tag{38}$$
$$\frac{\partial f}{\partial a} = b \tag{39}$$
$$\frac{\partial f}{\partial b} = a \tag{40}$$
$$g(e,f) = e + f \tag{41}$$
$$\frac{\partial g}{\partial e} = 1 \tag{42}$$
$$\frac{\partial g}{\partial f} = 1 \tag{43}$$

e, f は、値の大きさに依らずそれぞれ 1 の影響を g に与える。つまり、e の値が 1 変わったら g が 1 変わるという意味である。a, b, c, d は、e, f に対してそれぞれ 2, 3, 5, 4 の影響を与える。これらは同様に、a, b, c, d の値が 1 変わったら、e, f がそれぞれ 2, 3, 5, 4 変わるという意味である。さらに e, f は g に等倍の影響であるため、a, b, c, d が 1 変わったら、g が $2 + 3 + 5 + 4 = 14$ 変わることがわかる。これらの処理の流れは、図 11 の黄色で示されており、逆伝播を表す。
ここでは、計算グラフを用いることで、g に与える影響が最も大きいノードは c であることが分かった。

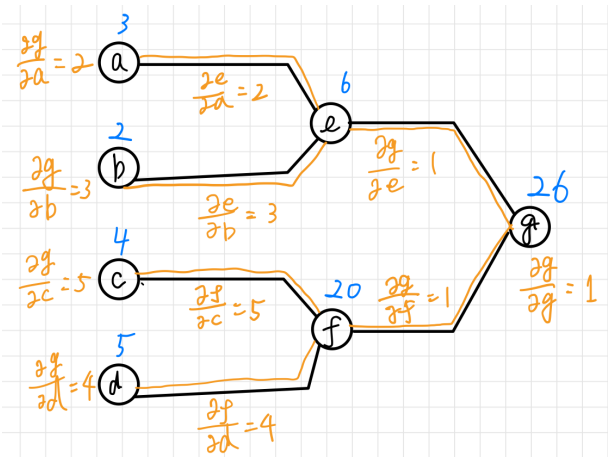


図 11: 計算グラフ

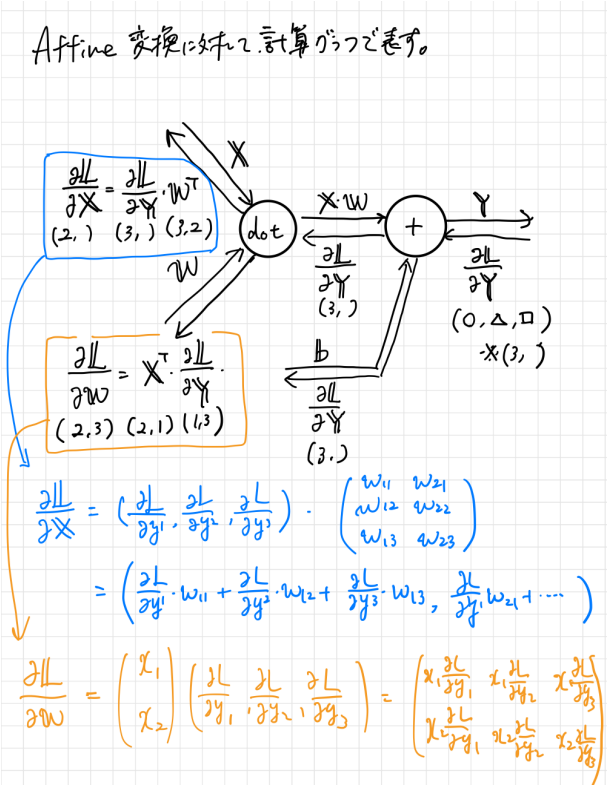


図 12: Affine 変換を計算グラフで表す

■参考文献

1. 岡谷貴之/深層学習 改訂第 2 版 [機械学習プロフェッショナルシリーズ]/ 講談社サイエンティフィク/ 2022-01-17
2. DeepLearning- 計算グラフについて理解する/@edo_m18(Kazuya Hiruma) https://qiita.com/edo_m18/items/7c95593ed5844b5a0c3b
3. 誤差逆伝播法を計算グラフを使って分かりやすく解説する https://deepage.net/deep_learning/2017/04/02/backpropagation.html
4. 交差エントロピー誤差をわかりやすく説明してみる/@kenta1984(Kenta Sasaki) https://qiita.com/kenta1984/items/59a9ef1788e6934fd962