深層学習 Day1 提出レポート

■ 動画講義メモ

Section1:入力層～中間層

入力データに対して重みをかけあわせたものが、中間層に入力される。（重みは、入力層と中間層の各ユニット毎に値が設定されている）
入力データの設定は、特徴が抽出出来そうなものを意図的に選ぶ。

Section2:活性化関数

それまでのパーセプトロンなどでは線形しか扱えなかったため、表現力に限界があった。
非線形の活性化関数を適用することで、表現力が上がった事によりニューラルネットワークが実用的なものとなり注目されることになった。
活性化関数の例として、シグモイド関数、ReLU 関数等がある。
ReLU 関数の方が結果が良い事が比較的多く、多く利用されている傾向ではあるが、場合によってはシグモイド関数の方が良い事もある。

Section3:出力層

誤差関数について、二乗誤差とクロスエントロピーが利用される。
二乗誤差は訓練データ（正解データ）と出力データとの差を元に求める。（以降、説明の都合上二乗誤差中心に展開していく）

活性化関数について、中間層とは目的が異なるため、利用される活性化関数も異なる。
回帰問題の場合、出力の値をそのまま利用する事になるので恒等写像を利用する。誤差は二乗誤差でも止める。
ロジスティック回帰問題の場合シグモイド関数を利用し、クラス分類問題の場合ソフトマックス関数を利用し、誤差関数はどちらも交差エントロピーとなる。

Section4:勾配降下法

勾配降下法では、一回のパラメータ更新を行うために訓練データ全てを利用して計算を行うため、計算量が問題となり、学習率の設定方法により大変な時間を要する事になる。

確率的勾配降下法(SGD)では、ランダムに抽出したサンプルの誤差を求める。データ計算量の低減、局所極小解に陥るリスクの低減が可能、またオンライン学習も可能。
ミニバッチ勾配降下法では、ランダムに分割したデータの集合に対する誤差を求める。CPU のスレッドを活用することや GPU を利用することが可能となるため、計算時間の短縮が可能となる。

Section5:誤差逆伝播法

出力と正解の誤差をパラメータで偏微分した導関数を求め、これを利用し出力側から入力側に順にパラメータの更新を行っていく。
（導関数を利用するため、解析的に求められるものに限定される事になる）
不要な再帰的な計算を避け、微分を逆算することでパラメータ更新を行うことが出来る。

■ 確認テスト

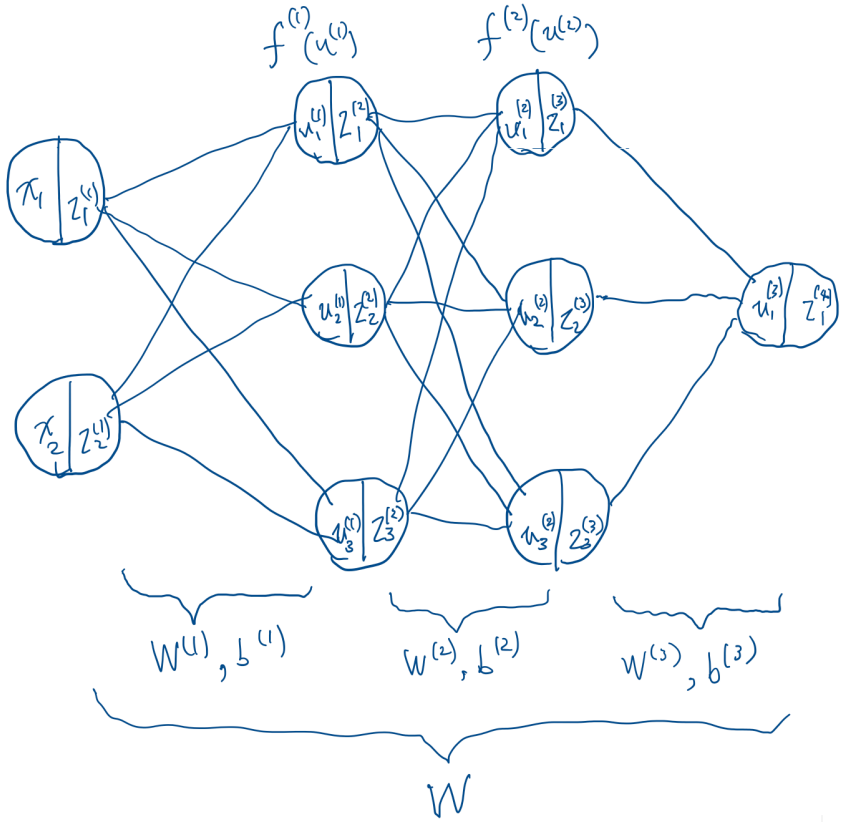
P.10 確認テスト
[Q] ディープラーニングは、結局何をやろうとしているか 2 行以内で述べよ。 また、次の中のどの値の最適化が最終目的か。 全て選べ。
①入力値[X] ②出力値[Y] ③重み[W] ④バイアス[b] ⑤総入力[u] ⑥中間層入力[z] ⑦学習率[ρ]
[A]
学習を通して誤差を最小にするネットワークを作成する。
具体的には、勾配降下法など（最小二乗法等）を利用してパラメータ(重み W,バイアス b)を学習により、誤差を最小化するパラメータを発見する事を目的とする。

最適化の最終目的は：③重み[W]、④バイアス[b]

P.12 確認テスト
次のネットワークを紙に書け。

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

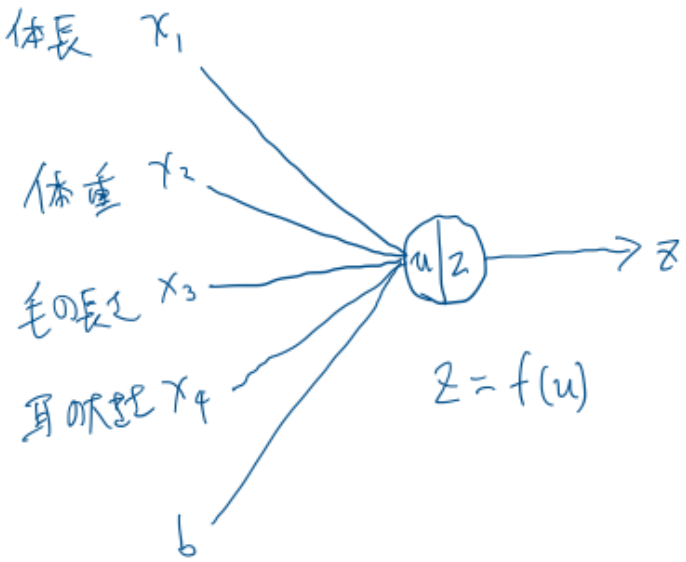
[A]



P.19 確認テスト

[Q]この図式に動物分類の 実例を入れてみよう。

[A]



P.21 確認テスト

[Q] 確認テスト この数式を Python で書け。

[A]

u1 = np.dot(x,W1) + b1

P.23 確認テスト

[Q] 1-1 のファイルから 中間層の出力を定義しているソースを抜き出せ。

[A]

中間層出力

z = functions.relu(u)

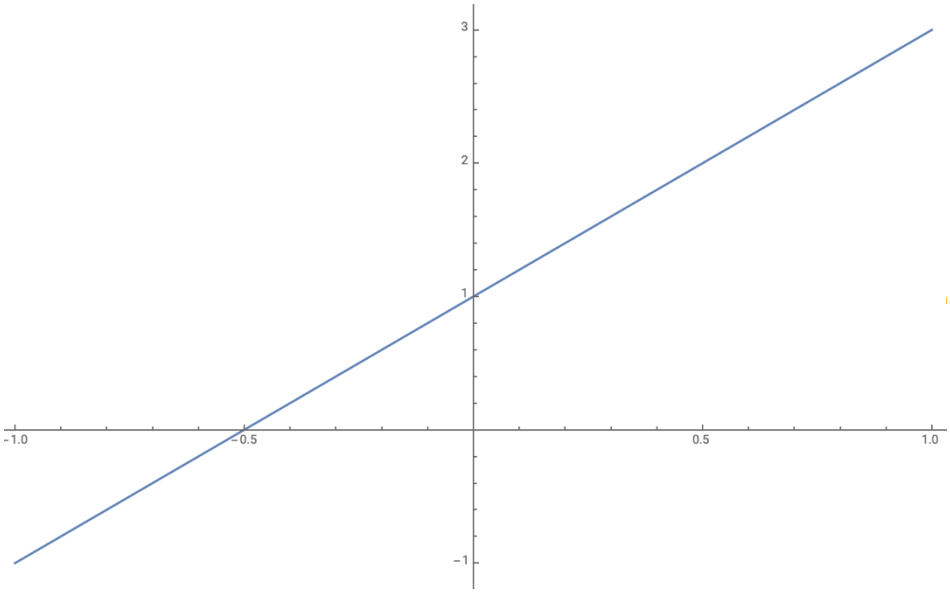
P.26

[Q] 線形と非線形の違いを図にかいて 簡易に説明せよ。

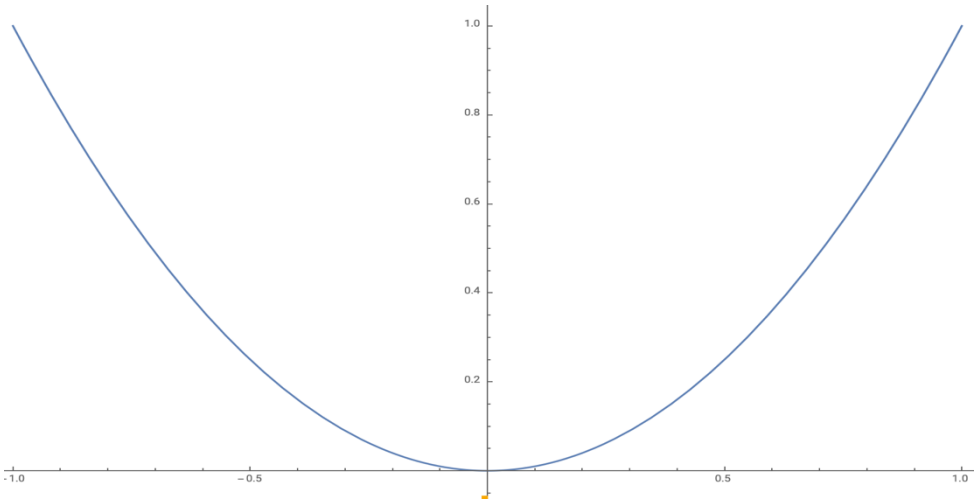
[A]

グラフが直線形になる関数が線形関数で、そうでない関数が非線形関数。

線形：y=2x+1 のグラフ



非線形：y=x^2 のグラフ



P.33 確認テスト

[Q] 配布されたソースコードより該当する箇所を抜き出せ。

[A]

z1 = functions.sigmoid(u)

P.44 確認テスト

[Q]

1. なぜ、引き算でなく二乗するか述べよ

2.下式の 1/2 はどういう意味を持つか述べよ

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{J}(y_j - d_j)^2 = \frac{1}{2}\|(\mathbf{y}-\mathbf{d})\|^2$$

[A]

1.絶対値の代わりに、微分が可能な二乗で代用している

2.上記で二乗したものを微分した際に相殺し、計算簡略化を図るため。

P.51 確認テスト
[Q] 該当するソースコードを示せ。
[A]

```
# ソフトマックス関数
def softmax(x):
    #【説明】x が二次元の場合の対応
    if x.ndim == 2:
        x = x.T
     #【説明】np.exp で高次元を指定した場合のオーバーフロー対策のため、x の最大値(行方向の最大値)で全ての値を減算しておく
        x = x - np.max(x, axis=0)
        #【説明】ソフトマックスの数式通りの実装
        # 出力 y として、各入力 x について np.exp(x) を x の行方向で np.exp(x) の結果を足し合わせたもので割ったものを出力する
        y = np.exp(x) / np.sum(np.exp(x), axis=0)
        return y.T

    #【説明】x が一次元の場合の対応
    x = x - np.max(x) # オーバーフロー対策
    return np.exp(x) / np.sum(np.exp(x))
```

P.56 確認テスト
[Q] 該当するソースコードを示せ。
[A]

```
# クロスエントロピー
def cross_entropy_error(d, y):
    #【説明】出力データ配列が一次元の場合、出力データ、正解データ共に 1 x データ配列要素数 の行列へ変換
    if y.ndim == 1:
        d = d.reshape(1, d.size)
        y = y.reshape(1, y.size)

    # 教師データが one-hot-vector の場合、正解ラベルのインデックスに変換
    if d.size == y.size:
        #【説明】正解データ配列から正解ラベルの最大値インデックスの配列へ変換
```

```
        d = d.argmax(axis=1)

    #【説明】batch_size = y のレコード数
    batch_size = y.shape[0]

    #【説明】交差エントロピーの数式通りの実装
  # np.arange(batch_size) = 0~batch_size -1 までの配列を等差数列で作成する
[0,1,… batch_size -1 ]
    # y[np.arange(batch_size), d] = y[[0,1,… batch_size -1 ], [正解データ配列
1,4,3,3,2....]]となり、[y[0,1], y[1,4], y[2, 3],… となり、結果 1,5,5,… のような配列が
得られる
    # これらの結果をそれぞれ log を取ったものを全て足し合わせ batch_size で割
っている
    # また、微小量 1e-7 を加える事で np.log に 0 を与える事が事がないようにし
ている(log0=-inf となるため)
    return -np.sum(np.log(y[np.arange(batch_size), d] + 1e-7)) / batch_size
```

P.56 確認テスト

[Q] 以下それぞれに、該当するソースコードを示せ。
[A]

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E$$

重み更新（1 世代前の重みから学習率 * 勾配を減算していく）
network[key]　-= learning_rate * grad[key]

$$\nabla E = \frac{\partial E}{\partial \mathbf{w}} = \left[\frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_M} \right]$$

誤差逆伝播による勾配計算
grad = backward(x, d, z1, y)

P.65 確認テスト

[Q] オンライン学習とは何か、二行でまとめよ

[A]

バッチ学習（学習対象となるデータが全て学習開始前に事前に揃っている状態で学習する）の対となり、オンライン学習では事前データ無しで新たなデータが発生次第都度学習を行う。

オンライン学習の学習方として、例えばロジスティック回帰であれば都度発生した入力データに対する出力と、正解データで損失関数を計算したものの微分値で重み更新を行う確率的勾配法(SGD)による学習が行われる。

P.68 確認テスト

[Q] この数式の意味を図に書いて説明せよ

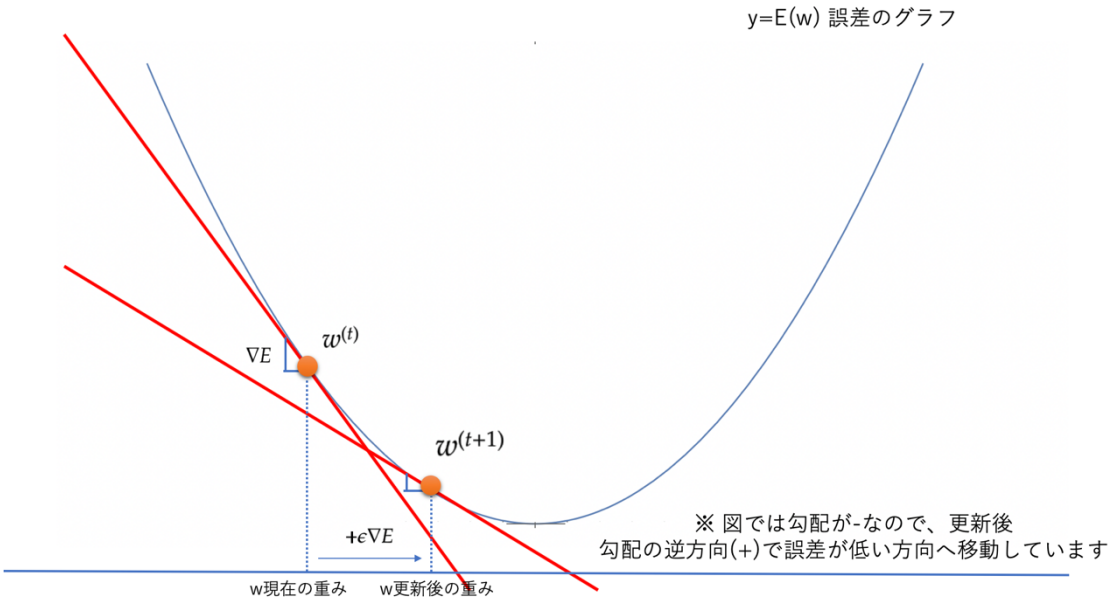
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E_t$$

[A]

重みの更新について、

更新後の重み w(t+1)は、現時点の重み w(t)を正解データとの誤差 E の勾配に学習係数ϵを掛け合わしたものを引いたもので更新していく事になる。

（勾配がプラスであればマイナス方向へ、勾配がマイナスの場合はプラス方向とすることで誤差が小さくなる点を見つける）



P.79 確認テスト
[Q] 誤差逆伝播法では不要な再帰的処理を避ける事が出来る。
既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]

```
# 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
    # 【説明】誤差（誤差関数　sig+クロスエントロピー)を微分を計算、結果として
delta2 を求める
    delta2 = functions.d_mean_squared_error(d, y)

  # 【説明】delta2 を利用し微分を逆算することで再帰的処理を回避
    # b2 の勾配
    grad['b2'] = np.sum(delta2, axis=0)
    # W2 の勾配
    grad['W2'] = np.dot(z1.T, delta2)

  # 【説明】delta2 を利用し微分を逆算し delta1 を求めている
    # 中間層でのデルタ
    delta1 = np.dot(delta2, W2.T) * functions.d_relu(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)
```

return grad

P.83 確認テスト

[Q] 2 つの空欄に該当するソースコードを探せ。

[A]

$$\frac{\partial E}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}}$$

delta2 = functions.d_mean_squared_error(d, y)

$$\frac{\partial E}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial w_{ji}^{(2)}}$$

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