【要点まとめ】 ニューラルネットワークのパラメータ量は膨大である。 それらを全て直接的に求めようとすると、膨大な資源と時間がかかってしまう。 そこで、誤差を出力層の側から順に微分し、入力層の側に向かって伝播する手法が生み出された。 それが誤差逆伝播法である。

【実装演習】

```
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（平均二乗誤差の導関数）
    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) * function.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)

    print_vec("偏微分_W1", grad["W1"])
    print_vec("偏微分_W2", grad["W2"])
    print_vec("偏微分_b1", grad["b1"])
    print_vec("偏微分_b2", grad["b2"])

    return grad
```

【実装演習考察】 各層を遡って順次微分していく動きは理解しやすく、サンプルソースではエポックを重ねる毎に勾配が収束していく様子が理解できました。 ※とはいえ写経をしましたが、実装となると難しい部分があるため（特にデルタ回り）別途書籍など利用しての復習が必要だと感じました。