【要点まとめ】 NNの最終目的の一つは、誤差関数の値を最小化すること。 重み(w)とバイアス(b)について最適な値を見つける必要があるが、パラメータ数が膨大である場合は 解析的に求めるのはほぼ不可能である。 そのために勾配降下法と呼ばれる、微分を利用して極小を求める手法が利用される。

- 勾配降下法 一般的、全てのデータを使用して一度にパラメータを更新するため、膨大なメモリを使用する可能性がある。
- 確率的勾配降下法 ランダムに抽出したサンプルを使用して、少しずつパラメータを更新していく。 オンライン学習が可能であるため、メモリの節約、データの逐次投入が可能となる。
- ミニバッチ勾配降下法 分割されたデータからサンプルを抽出し、誤差をとっていく。 並列に処理を実行することが可能であるため、CPU・GPU資源の有効活用ができる。

【実装演習】

- ここでは学習率にフォーカスした実装を写経する

```
# 学習率
learning_rate = 0.07

# 勾配降下を繰り返す
for dataset in random_datasets:
    x, d = dataset['x'], dataset['d']
    z1, y = forward(network, x)  # 順伝播
    grad = backward(x, d, z1, y) # 誤差逆伝播

    # パラメータに勾配降下を適用
    for key in ('W1','W2','b1','b1'):
        network[key] -= learning_rate * grad[key]

    # 誤差を計算
    loss = functions.mean_squared_error(d, y)
    losses.append(loss)
```

【実装演習考察】 勾配降下法を実装、使用する際は学習率に注意を払う必要があることを学びました。 学習率が大きすぎると、いつまで経っても極小に辿り着けません。 学習率が小さすぎると、極小に辿り着くまでにとても時間がかかります。（関数が波打っている場合は極小に辿り着けない場合すらあります）