【要点まとめ】 勾配消失問題とは、誤差逆伝播法により出力層に近づくにつれて微分の連鎖式が膨大になるが 微分結果が0に近くなり更新がほとんど行われない（収束しない）状況のことである。

シグモイド関数は勾配消失問題を引き起こす活性化関数の代表例である。

解決方法は大きく3つ

- 別の活性化関数を使用する →活性化関数はシグモイド関数以外にも無数にある（たとえばRelu関数）
- 重み(w)の初期値を再検討 →用途（活性化関数）に合わせて、XavierやHeを使い分けないと、中間層の意味が薄くなる
- バッチ処理をする →バッチ正規化を行うことで ・中間層の重みの更新が安定化する ・学習が速くなる ・過学習を抑えることができる（正規化を行うことでデータのばらつきが抑えられるため）

【実装演習】

```
# Xavier or He で重みの初期値を設定している箇所を抜粋して写経
def __init_weight(self, weight_init_std):
    all_size_list = [self.input_size] + self.hidden_size_list +
[self.output_size]
    for idx in range(1, len(all_size_list)):
        scale = weight_init_std # 今回は初期値0.01
        if str(weight_init_std).lower() in ('relu', 'he') # Relu関数、Heで初
期化の場合
            scale = np.sqrt(2.0 / all_size_list[idx - 1])
        elif str(weight_init_std).lower() in ('sigmoid', 'xavier') # シグモ
イド関数、Xavierで初期化の場合
            scale = np.sqrt(1.0 / all_size_list[idx - 1])

        self.params['W' + str(idx)] = scale *
np.random.randn(all_size_list[idx - 1], all_size_list[idx])
        self.params['b' + str(idx)] = np.zeros(all_size_list[idx])
```

※ 参考資料：2_2_2_vanishing_gradient_modified.ipynb

【実装演習考察】 重みの初期値は適当な値を置いておけばよいと考えていましたが、使用する活性化関数に応じた 適切な手法（XavierやHe）を検討し、実装することが大切だと実感しました。 なぜこれらはsqrtを使用するのか難しいところもありましたが、動画と実際の出力グラフで理解を深めることができました。