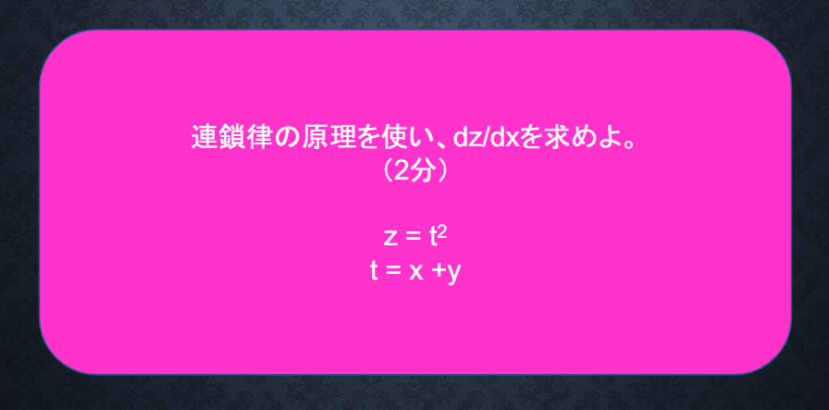
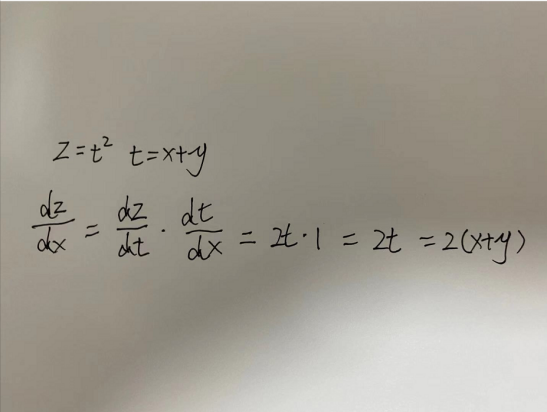
**まとめ要点１：**

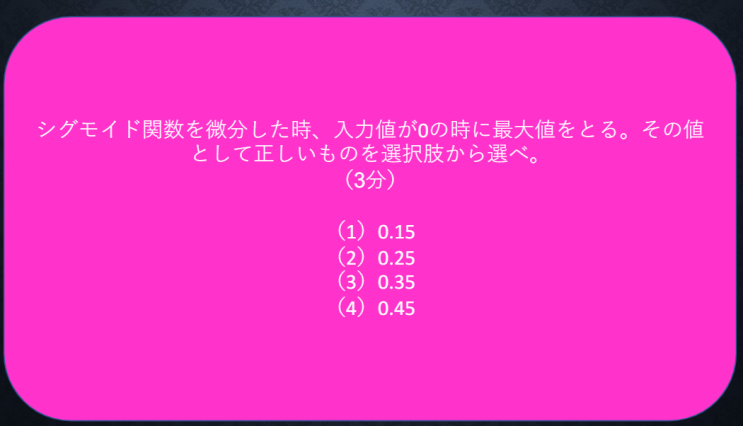
　勾配消失問題について

勾配消失問題は誤差逆伝播法が下位層に進んでいくに連れて、勾配がどんどん緩やかになっていく。そのため、勾配降下法による、更新では下位層のパラメータはほとんど変わらず、訓練は最適値に収束しなくなるという問題。勾配消失問題はsigmoid関数などの活性化関数と関連性は、sigmoid関数は0~1の間で大きな値では出力の変化が微小なため、勾配消失問題を引き起こす事があった。その解決策としてはRELU活性化関数を使用すること。

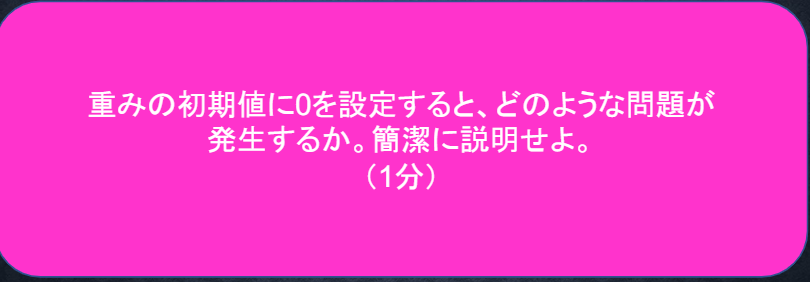
**確認テスト;**



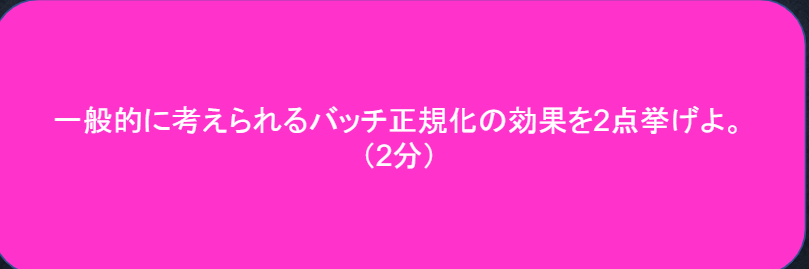
**答え：**



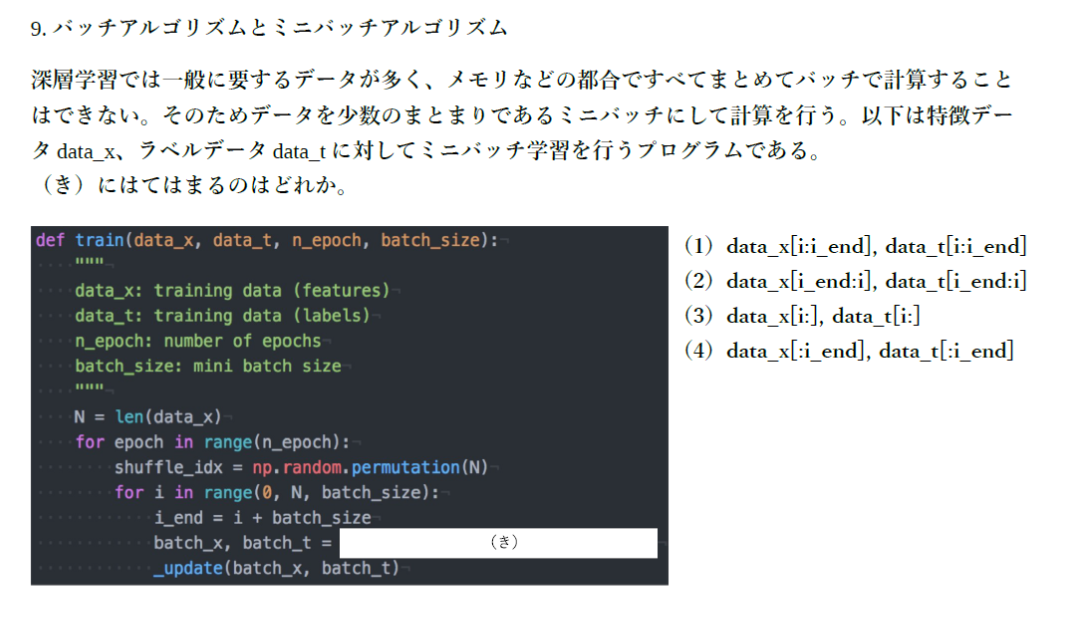
**答え：sigmoid関数を微分した後の式を利用して、０の時、最大値を取ることが出来るので、０を代入して計算すること、0.25、2番を選択することが分かりました。**



**答え**：**全て重みは均一に更新されるため、学習が行えないことになった。**



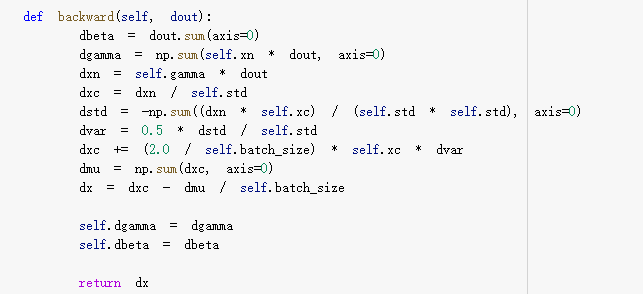
**答え**：**学習を安定化するや計算スビートアップすることができます。**



**答え：バッチサイズだけデータを取り出す処理である、1を選択する。**

**実装確認**







**初期値、バッチサイズなどのハイパーパラメータを調整することで、結果も変化している。**

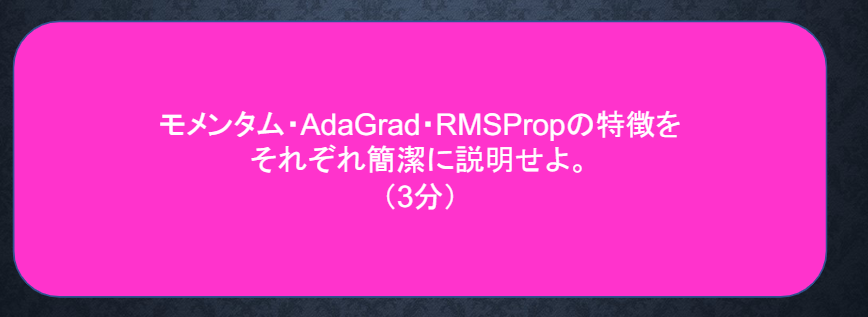
**サマリーと考察：**

**なぜsigmoid関数を使う深層NWの勾配は消えるのでしょうか。その原因は、パラメータを更新するため、活性化関数に対して微分しないといけないところです。もし、微分の結果は1より大きいの場合、中間層が増える時、勾配爆発の問題が起こります、でも1より小さい時は、中間層が増える時、全ての勾配情報は消えていく、それは勾配消失問題の原因だと思います。**

**まとめ要点2：**

　学習率最適化手法について

学習率を最適するための手法はいくつかがあります。よく使う手法として、モメンタムやAdaGradやRMSPropやAdamなど。モメンタム：誤差をパラメータで微分したものと学習率の積を減算した後、現在の重みに前回の重みを減算した値と慣性の積を加算する。AdaGrad：誤差をパラメータで微分したものと再定義した学習率の積を減算する。RMSProp：誤差をパラメータで微分したものと再定義した学習率の積を減算する。Adam：・モメンタムの過去の勾配の指数関数的減衰平均とRMSPropの過去の勾配の2乗の指数関数的減衰平均を組み合わせての最適化手法。



**答え：まとめに書いているので、ここは省略する。**

**実装確認：**

モメンタム：



AdaGrad：



RSMprop：



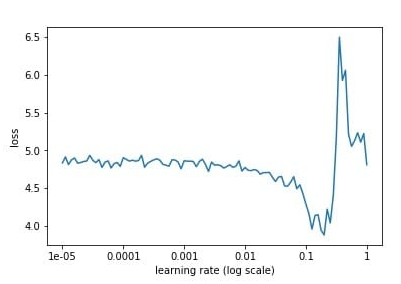
Adam：



**学習率の変更や重みの初期化パラメータ変更、バッチ正規化などの調整により、結果は変わります。**

**サマリーと考察：**

**学習率最適化手法について、お勧めの論文がある。2015年Leslie N. SmithがCyclical Learning Rates for Training Neural Networksを発表した。論文の3.3章には学習率を見つける方法はとても面白いだと思います。簡単に言うと、まず小さい学習率を設定して、毎回バッチ更新後学習率を段々増える、バッチごとのlossを求めて、そして学習率の変化曲線とlossの変化曲線を手に入れる。変化曲線を見ると、学習率が小さいの場合、lossの変化速度は低く、でも学習率が大きいの場合、lossは増大する可能性もあります。でも、その曲線を見ると、全体的に悪くないの初期学習率を見つけるこいが出来ます。(以下の図を見ると、学習率は0.1の時は一番適切)**

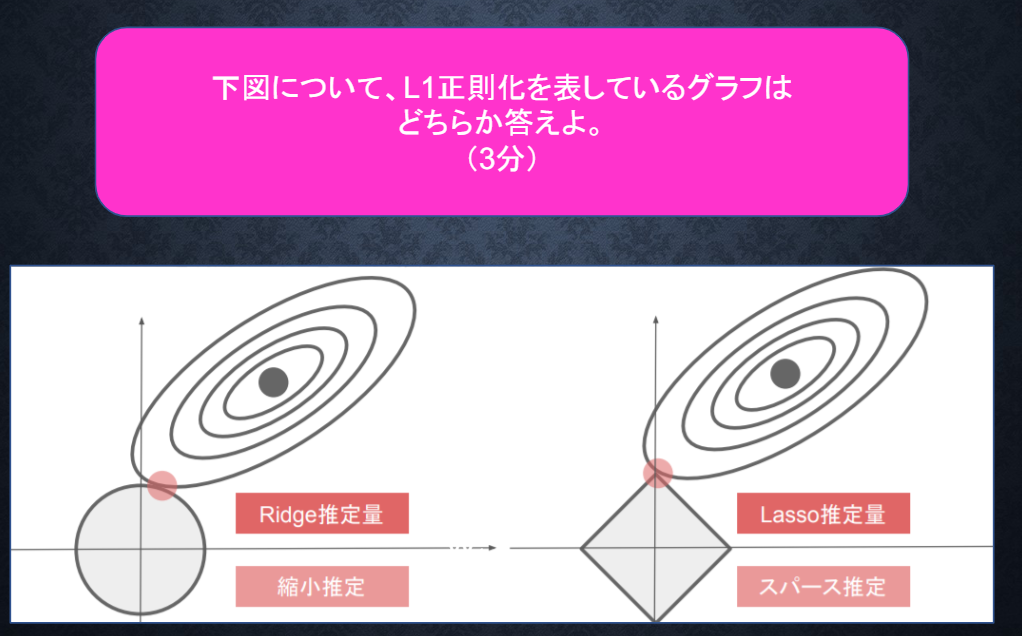


**まとめ要点３：**

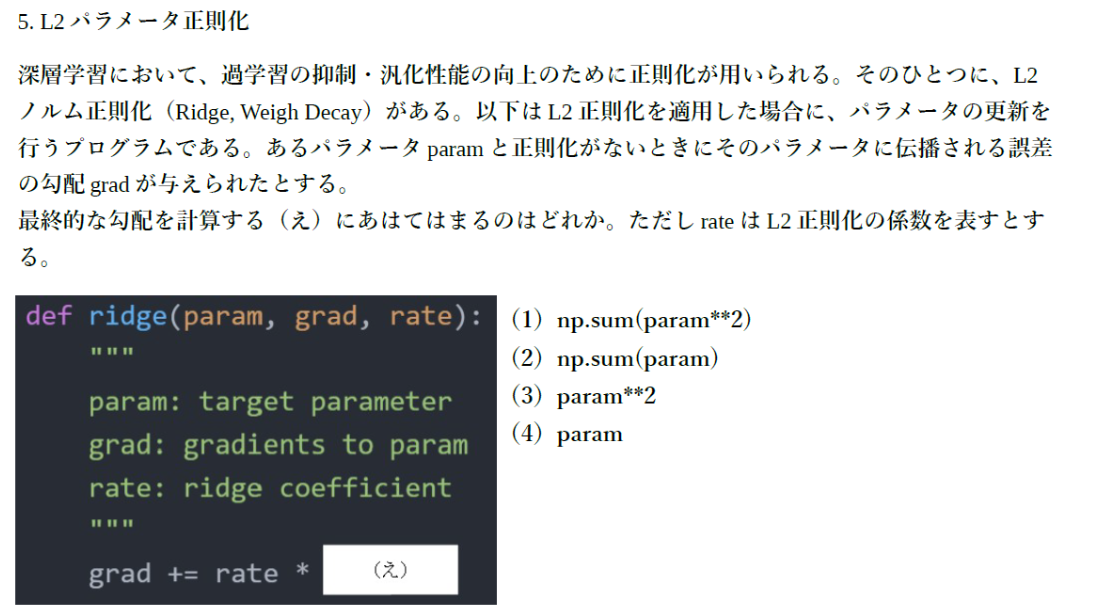
　過学習

過学習の基本概念は既に勉強していたので、説明を省略する。この章は過学習の対処法を紹介している。基本的には、誤差に対して、正則化項を加算することで、重みを抑制する正則化手法(L1、L2)、とランダムにノードを削除して学習させることで過学習、ドロップアウト層の追加して二択あります。

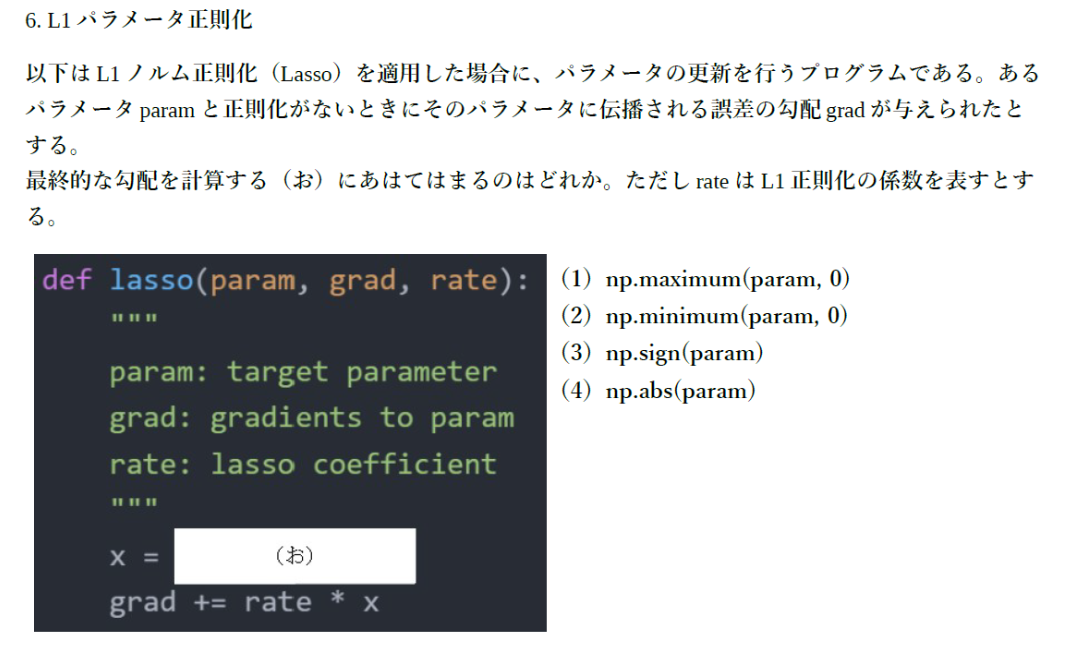
**確認テスト：**



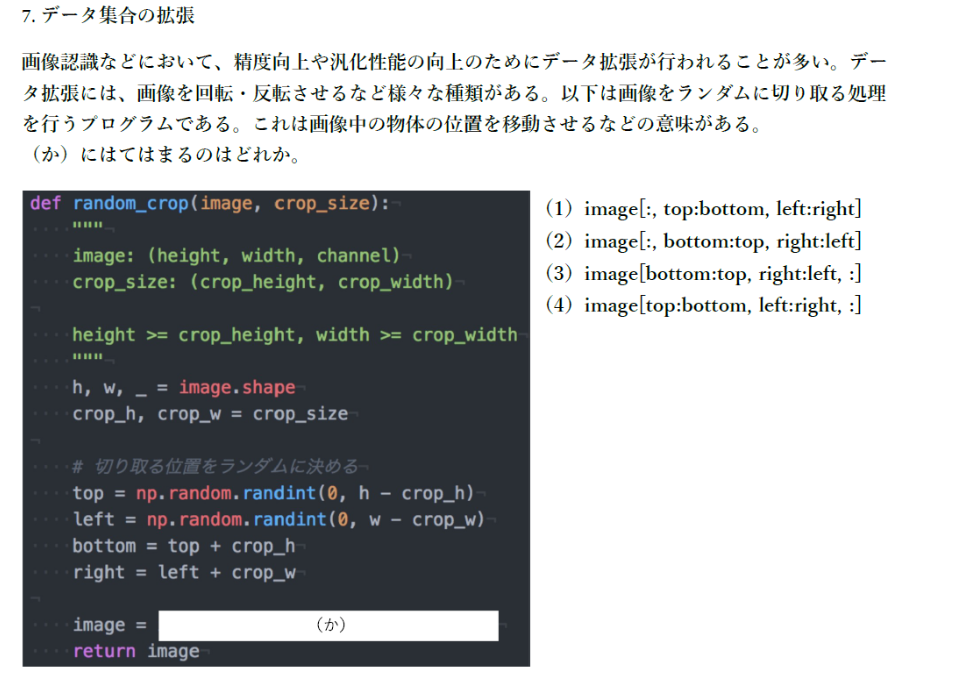
**答え：L1はLasso回帰なので、、右のグラフは正解。**



**答え：Param**



**答え：(３)を選択する、signは符号関数である。**



**答え：imageの形式が(縦幅, 横幅, チャンネル)であるなので、(４)は正解。**

**実装演習：**

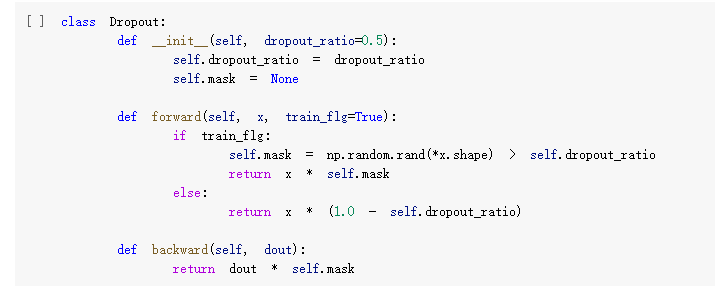
L1：



L2：



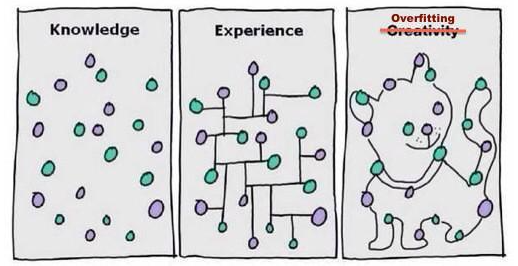
Dropout：



**＊実際に実装する時、正規化やdropoutなどの手法はDLエンジニア(tensorflowなど)に用意されていたので、それを使えば便利かつ間違ないなどのメリットがある。**

**サマリーと考察：**

**面白い画像一枚を見ると、過学習はどんな感じなのか分かると思います。**



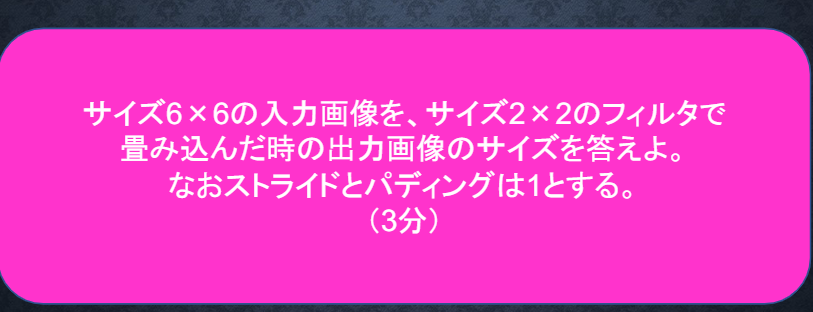
**一言で言うと、やりすぎ。**

**まとめ要点４：**

　畳み込みニューラルネットワークの概念

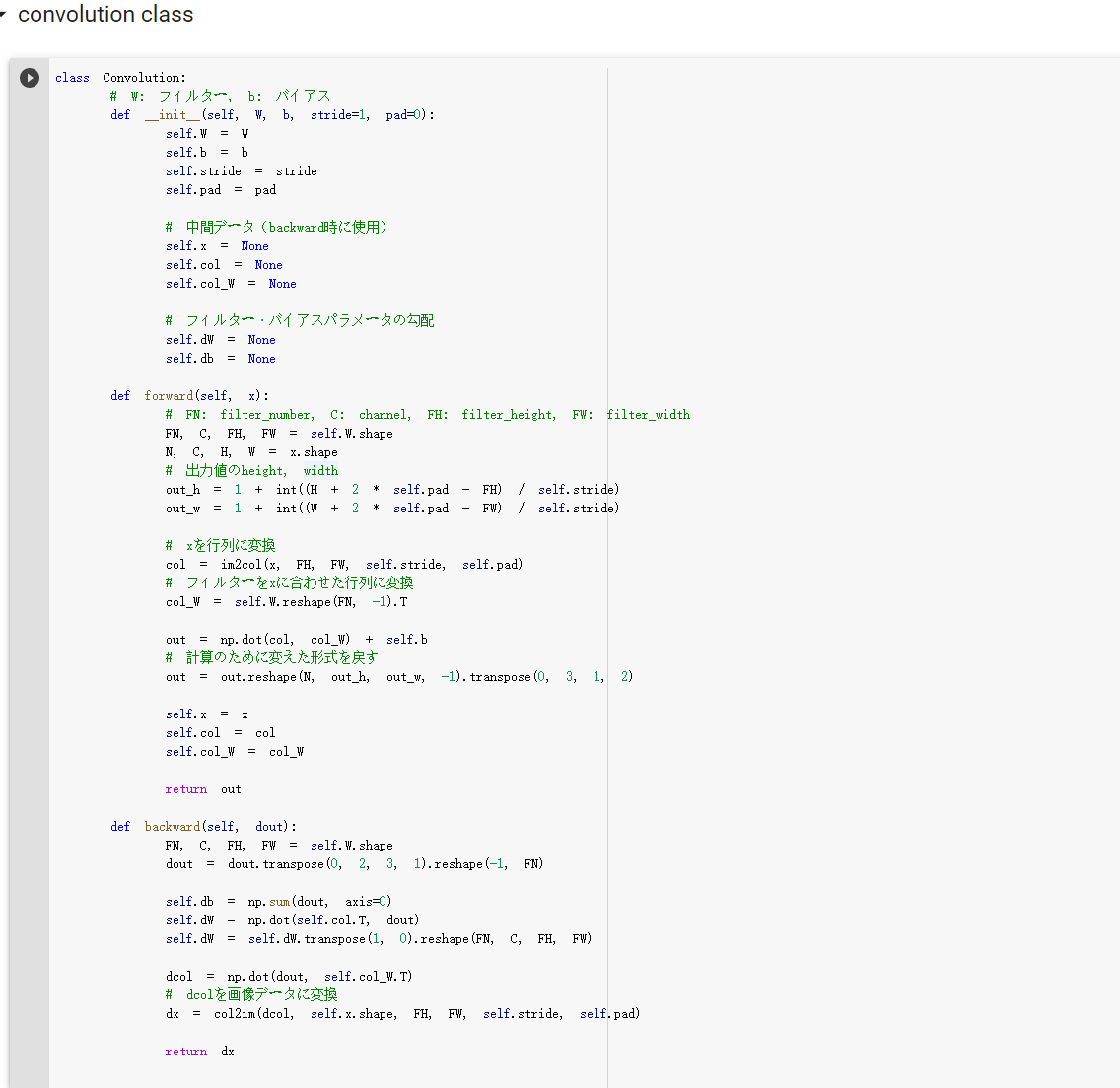
畳み込み層を追加することで、3次元の空間情報も学習できるようになりました。画像の場合、縦、横、チャンネルの3次元のデータをそのまま学習し、次に伝えることができる。数学的計算方式はいつもの結合層と違うところがですが、似てる部分もある。フィルターと畳み込み計算方式を理解することは重要。畳み込み層の後にプーリング層が存在するのは一般的である、違う種類のプーリング層の計算方法を把握する必要がある。

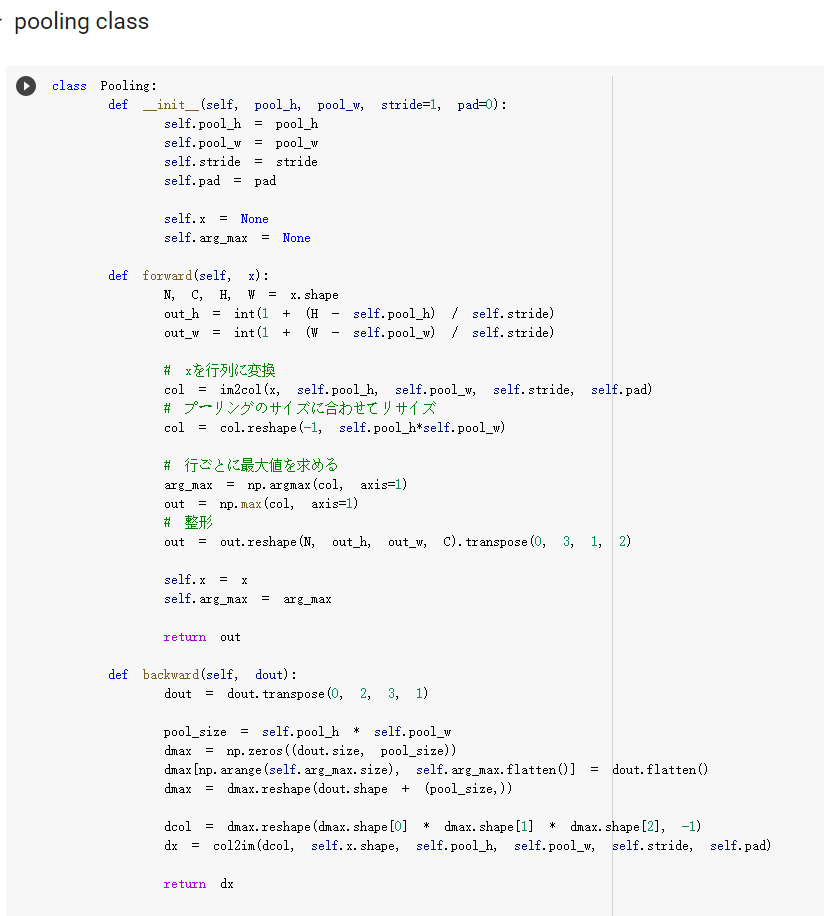
**確認テスト：**

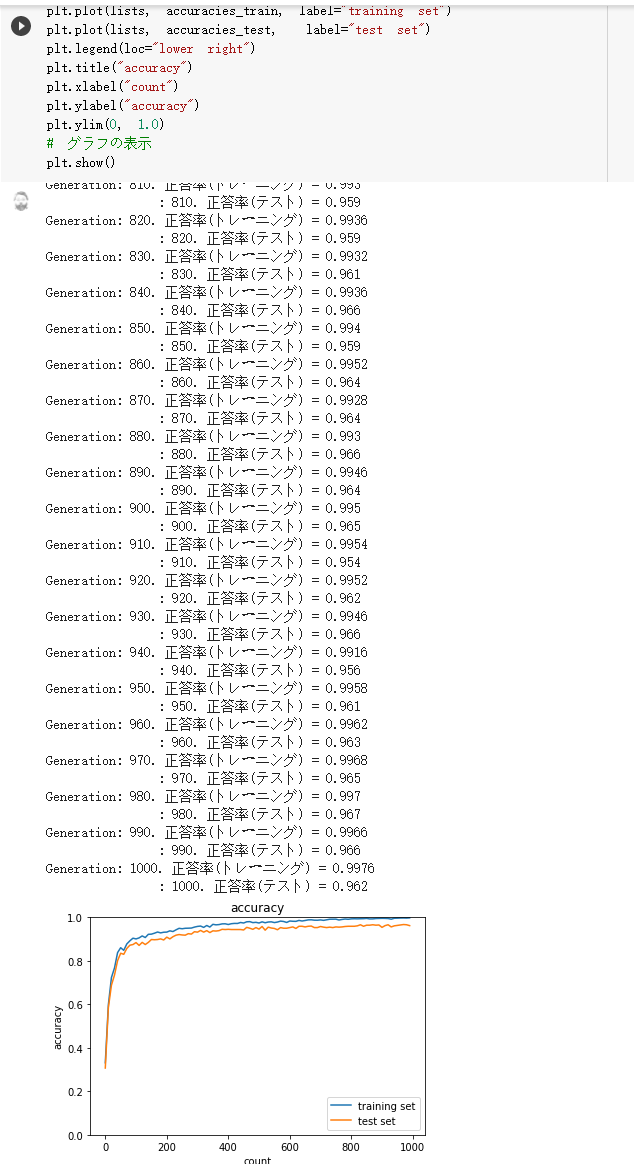


**答え：出力画像のサイズ３ｘ３である。**

**実装演習の確認：**







**サマリーと考察：**

**実際に畳み込み計算は畳み込みニューラルネットワークに使われる前に、既に画像処理領域に使われていた。例えば有名なSobel filterは画面中の輪郭を強調して、出力出来る。一回の畳み込み計算は輪郭を抽出することが可能です、一番重要なのは輪郭は画面の中に何があるのかを判断する根拠になれる。畳み込みニューラルネットワークは複数回の畳み込み計算を行うことで、画像中の特徴を抽出することが出来るを期待できる。**



**まとめ要点５：**

　AlexNetのモデル

AlexNetのモデルの特徴は初の 5 つは畳み込み層で、そのうちのいくつかに max-pooling 層が続き、最後の 3 つは全結合層だった[2]。活性化関数には、非飽和型の ReLU を使用していた。AlexNet は、2012 年 9 月 30 日に開催された ILSVRC 2012[3] に参加した。AlexNet はエラー率 15.3% で優勝し、次点よりも 10.8% 以上低かった。この論文の主な内容は、モデルの深さが高性能には不可欠であるというもので、計算コストは高くなるものの、GPU を用いて学習することで実現した。

**サマリーと考察：**

**AlexNetの新しくポイントについてをまとめていきたいと思います。**

**その一、AlexNetはその時規模は一番大きい畳み込みニューラルネットワークである、その前にLeNet-5は畳み込み層三つ、fully connection層一つしない。**

**その二、GPU計算を利用して、畳み込み計算を高速にした、その後GPUを利用して、畳み込みニューラルネットワークは主流となった。**

**その三、dropoutやRELUやData Augumentationなどの技術を利用して、過学習問題を解決して、モデルをパラメータ数は60milionを超えた場合でも収束できるところ。今でも以上の方法は畳み込みニューラルネットワークを構築する時、欠かせない方法になりました。**