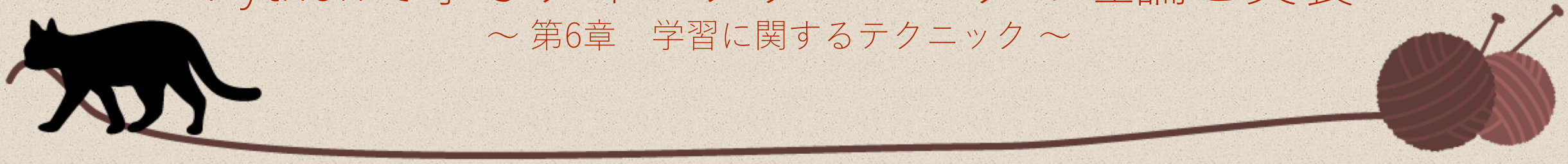
# ゼロから作るDeep Learning
## Pythonで学ぶディープラーニングの理論と実装
～ 第6章　学習に関するテクニック ～



浅井香奈江
(お茶の水女子大学)

# 内容

**1節　パラメータの更新**
**2節　重みの初期値**
**3節　Batch Normalization**
4節　正則化
5節　ハイパーパラメータの検証

**今日の話**

# 前章まで使用してきたパラメータの更新手法

・ニューラルネットワークの学習の目的：損失関数の値をできるだけ小さくする最適なパラメータを見つける
→これまでは勾配をヒントにより良いパラメータを探していた
・確率的勾配降下法(SGD：stochastic gradient descent)：単純に勾配方向へある一定の距離を進む、ということを繰り返し、パラメータの更新する手法

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

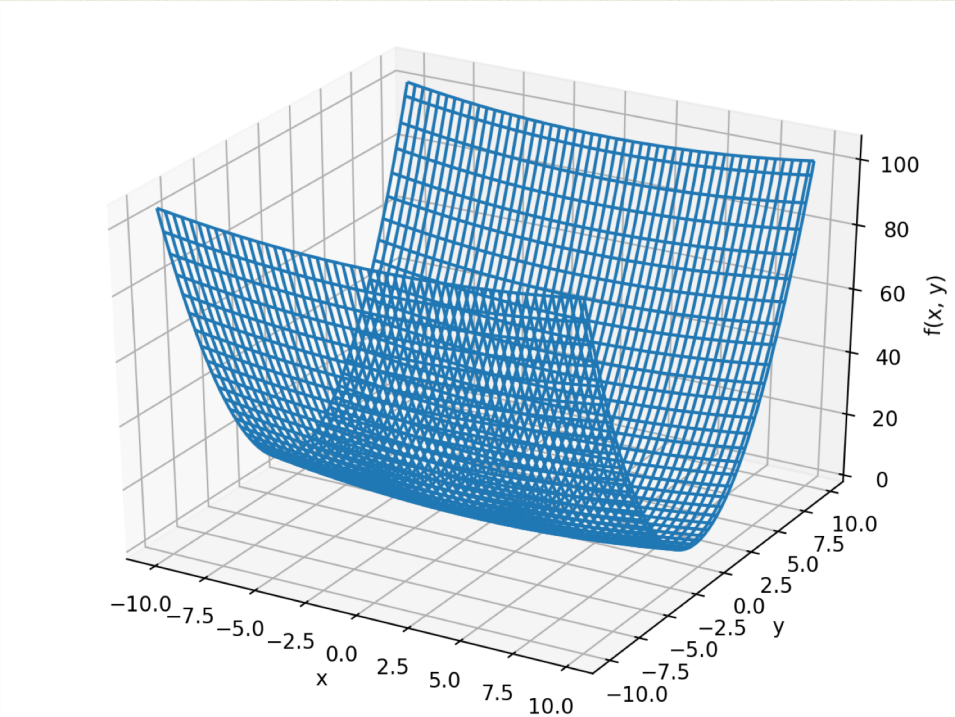
$W$：更新する重みパラメータ, $\frac{\partial L}{\partial W}$：$W$に関する損失関数の勾配, $\eta$：学習係数

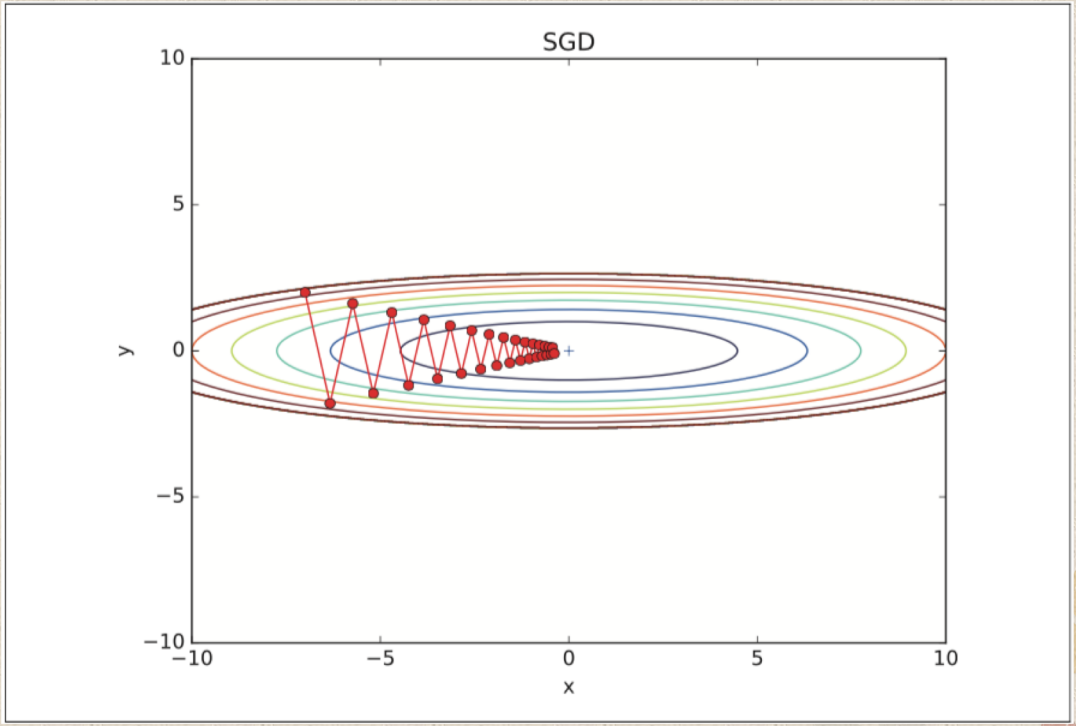
```
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr # lr：学習率

    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]
            # params[key]：パラメータ毎の値, grads[key]：パラメータ毎の損失関数の微分
```

# SGDの欠点

・関数の形状が等方的でないとパラメータの更新が非効率

・例：$f(x,y)=\frac{x^2}{20}+y^2$ → 勾配はx軸方向が極端に小さくy軸方向に大きい

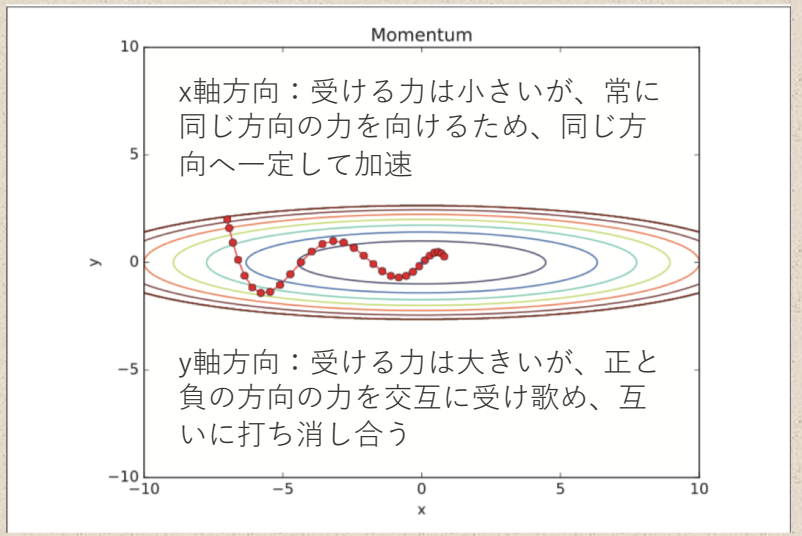



パラメータの更新経路がy軸方向にジグザグし
非効率な経路でパラメータを探索

# SGDの欠点を改善した手法

## Momentum

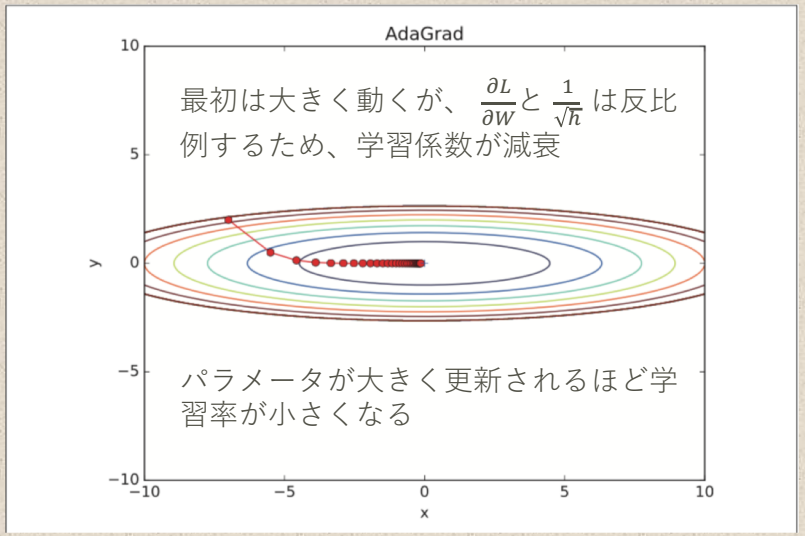



$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$
$$W \leftarrow W + v$$

$v$：重みパラメータの更新量(速度),
$\alpha$：抵抗のようなもの

**抵抗を加え、ジグザグの動きを軽減**

## AdaGrad

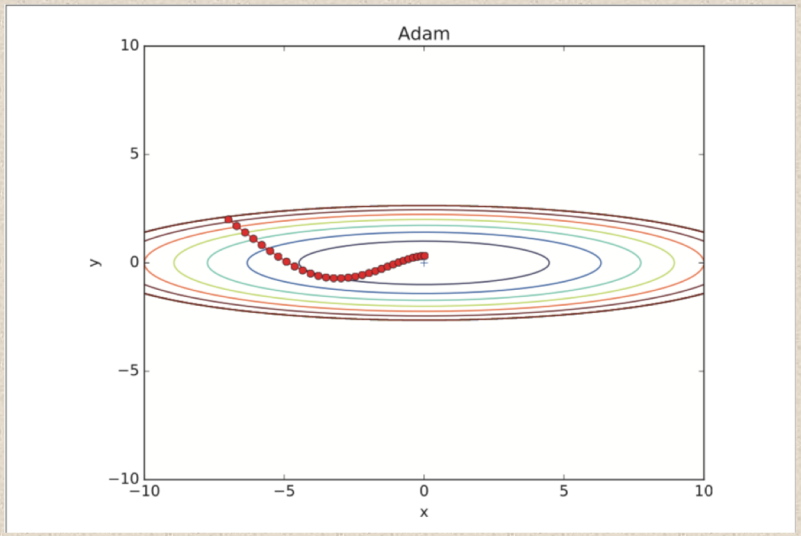



$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$
$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

$\frac{1}{\sqrt{h}}$を乗算することで学習係数を減衰

**最初は大きく学習し、次第に小さく学習**

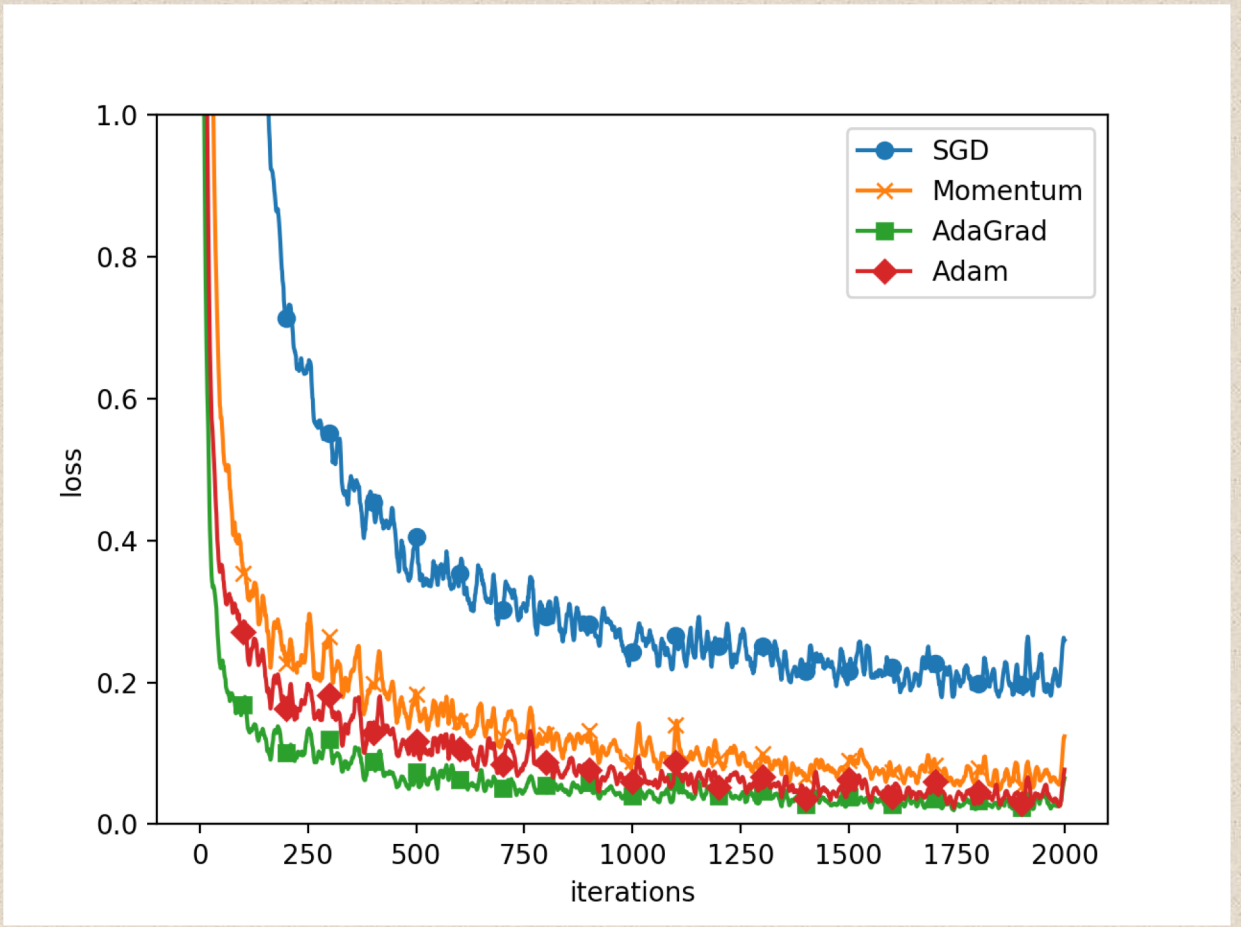
## Adam




MomentumとAdaGradの融合
詳細は原著論文を参照

解くべき問題、ハイパーパラメータ(学習係数など)の設定値によって結果が変わる
それぞれに特徴があり、得意・不得意な問題がある（けど主に使用されるのはSGDやAdamらしい）

# 更新手法の比較




手書き数字認識を対象にSGD, Momentum, AdaGrad, Adamを比較

結果：

・SGDよりも他の手法が速く学習できている
・AdaGrad の学習が少しだけ一番速い

※学習係数のハイパーパラメータや、ニューラルネットワークの構造(何層の深さか等)によって結果は変化

一般に、

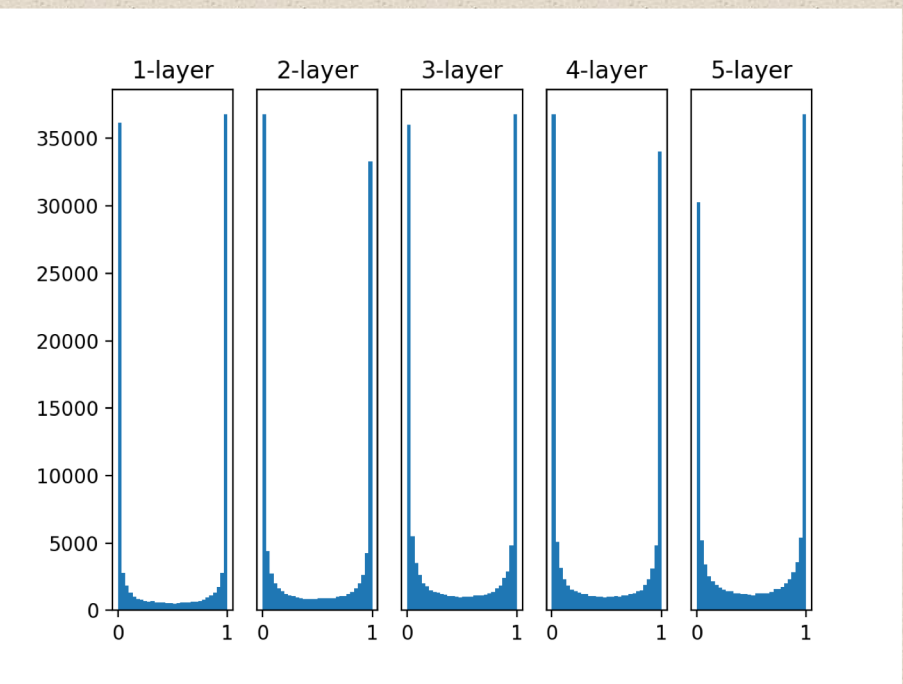
**SGDよりも他3つの手法の方が速く学習でき、時には最終的な認識性能も高くなる**

ことが知られている

# 重みの初期値

重みパラメータの初期値は小さくすることで過学習を防げる

Q. 0にするとどうなる？
A. 各層での誤差逆伝播法の出力が均等になってしまうため、沢山の重みをもつ意味がなくなる
　→ 0 (正確には重みを均一な値に設定すること)は避けるべき




重みの初期値によって隠れ層のアクティベーション（活性化関数の後の出力データ）がどのように変化するかを調べる

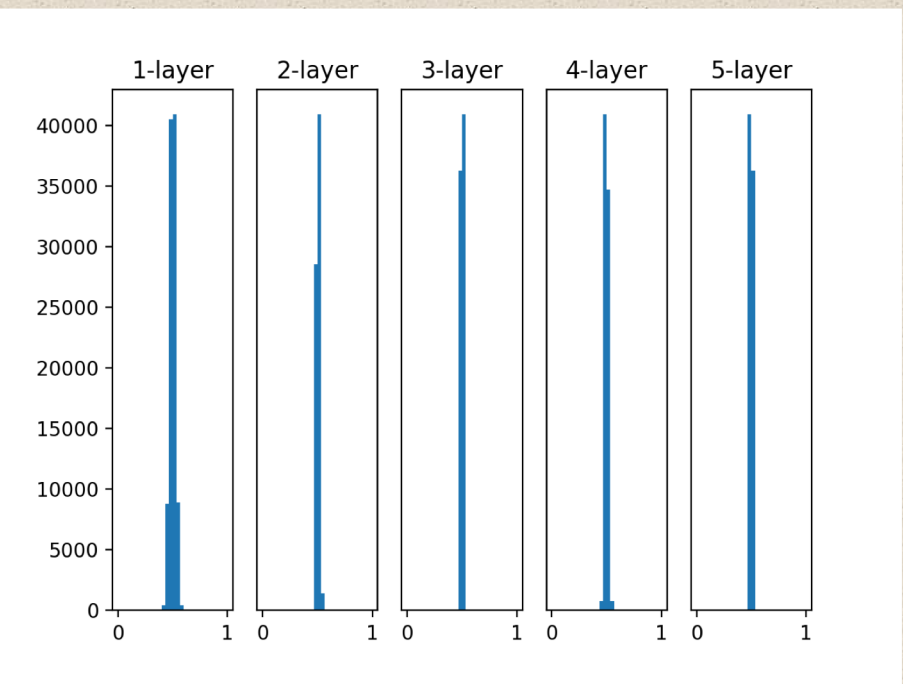
① 重みの初期値: 標準偏差が1のガウス分布
・5 つの層があり、それぞれの層は 100 個のニューロンを持つ
・活性化関数にsigmoid関数を使用

結果：
　・各層のアクティベーションは0と1に偏っている
　→ 0と1に偏ったデータ分布では、逆伝播での勾配の値がどんどん小さくなって消えてしまう(**勾配消失(gradient vanishing)**問題)

# 重みの初期値

重みパラメータの初期値は小さくすることで過学習を防げる

Q. 0にするとどうなる？
A. 各層での誤差逆伝播法の出力が均等になってしまうため、沢山の重みをもつ意味がなくなる
　→ 0 (正確には重みを均一な値に設定すること)は避けるべき




重みの初期値によって隠れ層のアクティベーション（活性化関数の後の出力データ）がどのように変化するかを調べる

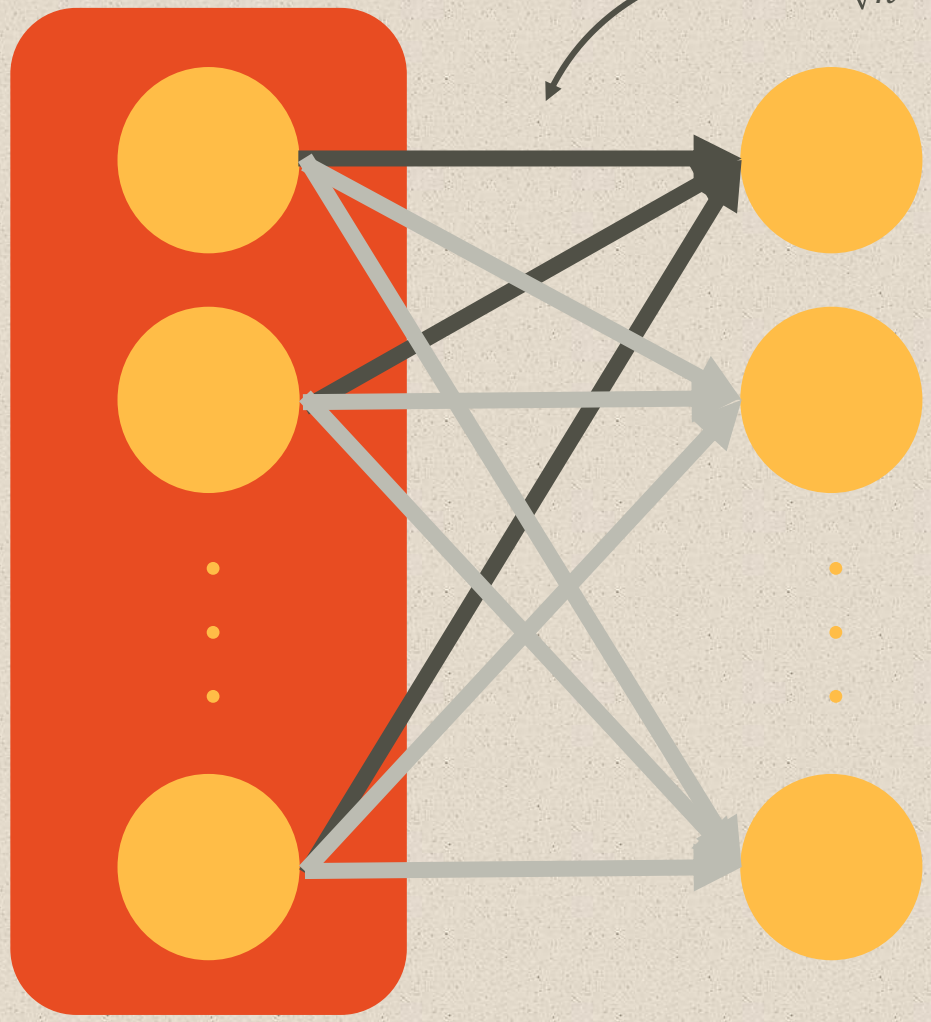
② 重みの初期値: 標準偏差が0.01のガウス分布
・5 つの層があり、それぞれの層は 100 個のニューロンを持つ
・活性化関数にsigmoid関数を使用

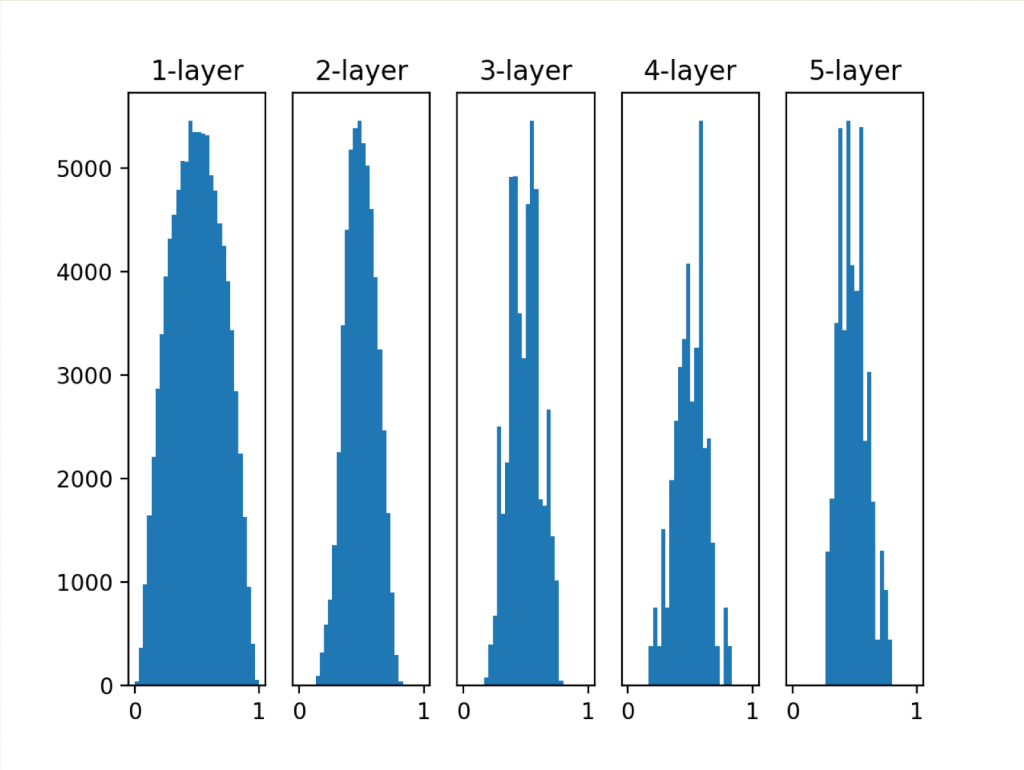
結果：
　・各層のアクティベーションは0.5付近に集中する
　→ 0と1の偏りがないため、勾配消失の問題が起きないが、
　　アクティベーションに偏りがある
　　ほとんど同じ値を出力するとすれば、複数のニューロンが存在する意味がなくなってしまう....

# Xavierの初期値

n個のノード

$\frac{1}{\sqrt{n}}$の標準偏差をもつガウス分布で初期化



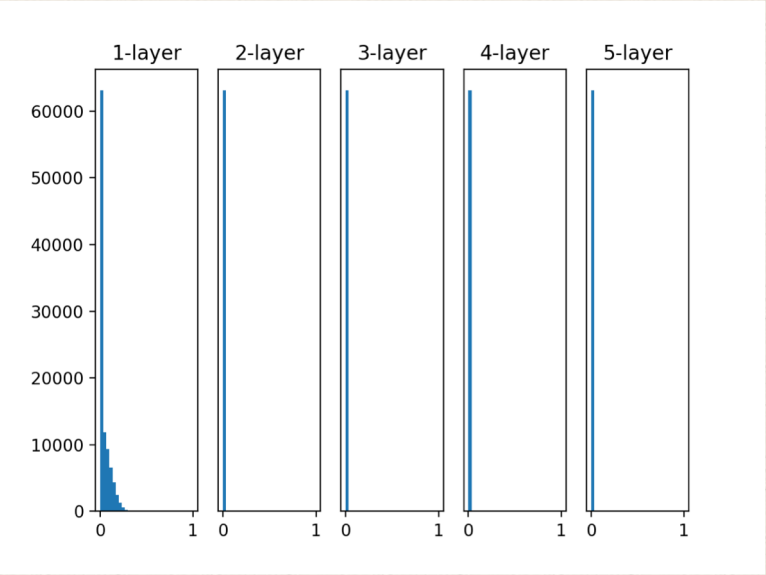



これまでよりも、広がりを持った分布になる
→ 効率的に学習が行える

# ReLUの場合の重みの初期値

・Xavierの初期値は、活性化関数が線形であることを前提に導いた結果
　→ sigmoid関数やtanh関数は左右対称で中央付近が線形関数として見なせ、Xavierの初期値が適している
・ReLUを用いる場合は、ReLUに特化した初期値を用いることが推奨されている

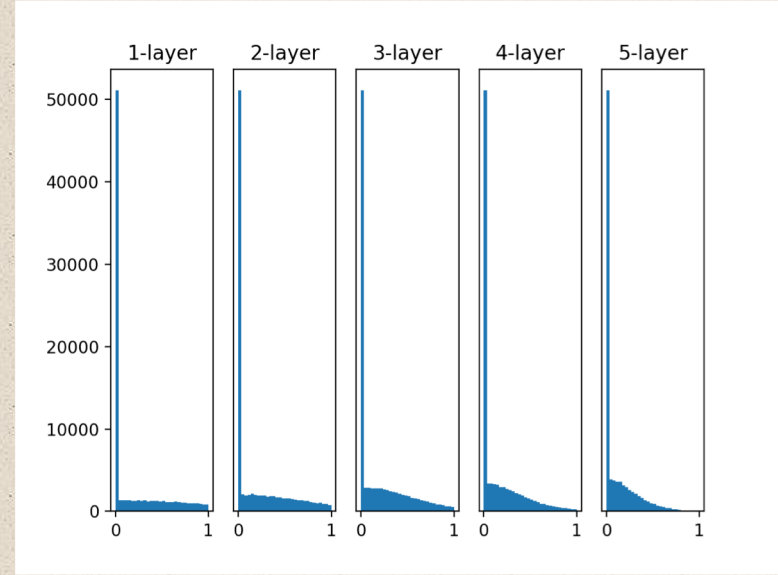
0.01の標準偏差をもつガウス分布




どの層のアクティベーションも小さい
学習が全く進まない

**Xavierの初期値**

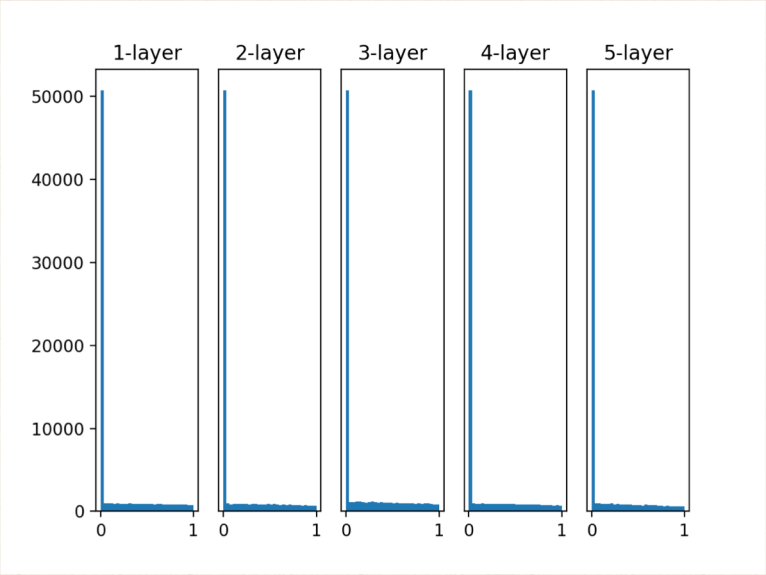
$\frac{1}{\sqrt{n}}$の標準偏差をもつガウス分布




層がディープになるにつれて
偏りが大きくなる
勾配消失問題

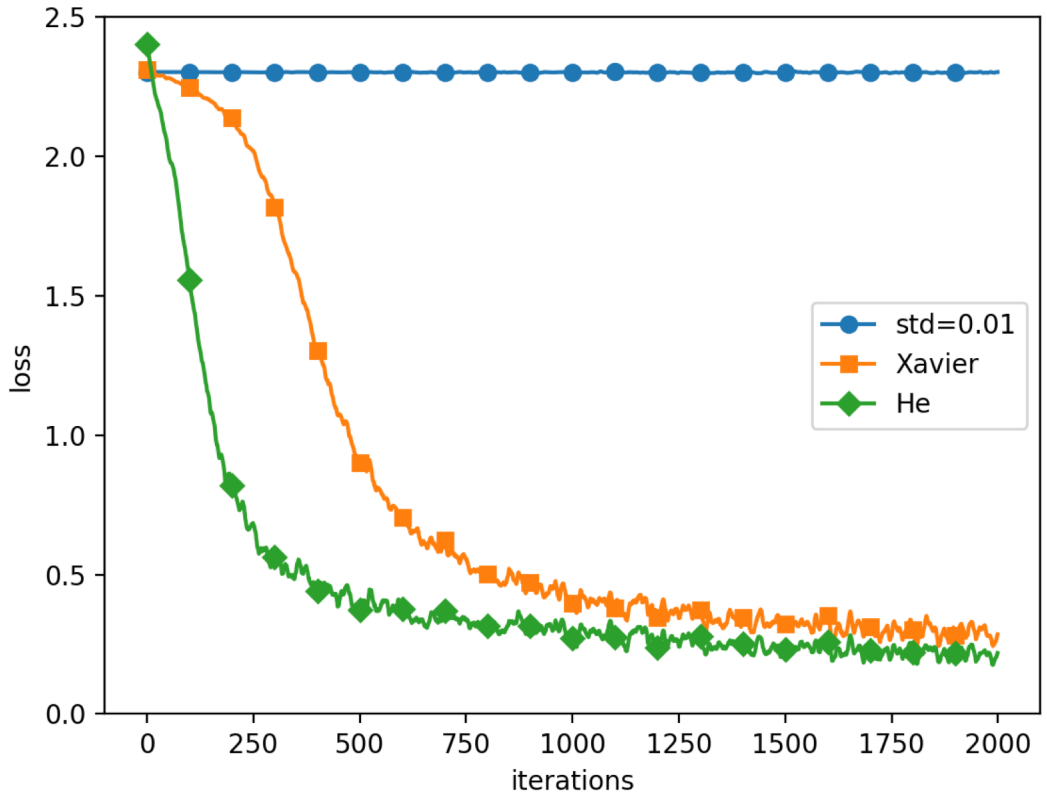
**Heの初期値**

$\frac{2}{\sqrt{n}}$の標準偏差をもつガウス分布




層をディープにしても
分布の広がりが均一
逆伝播の際も適切な値が流れると期待

# 重みの初期値の比較




重みの初期値の与え方の違いによって、学習効率の比較
・5層のニューラルネットワーク(各層100 個のニューロン)
・活性化関数としてReLUを使用

結果：
・std=0.01の場合はほとんど学習が進まない
・He、Xavierはサクサク学習が進んでいる
・Heの初期値の方が、より学習の進みが速い

→ **初期値の問題はとても重要**

# Batch Normalization

Batch Normalization：各層のアクティベーションの分布が適度な広がりを持つように”強制的”にアクティベーションの調整を行う

利点：

・学習を早く進行させることができる（学習係数を大きくすることができる）
・初期値にそれほど依存しない（初期値に対してそこまで神経質にならなくて良い）
・過学習を抑制する（Dropoutなどの必要性を減らす）

Batch Normalizationアルゴリズム：

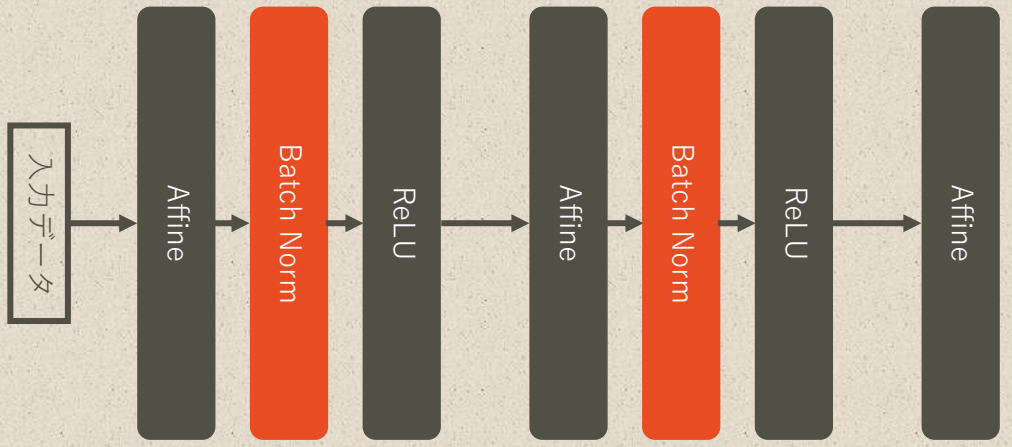
・各層のアクティベーションの分布が適度な広がりを持つように調整する
→ データ分布の正規化(平均が0、分散が1の分布）を行うレイヤーをニューラルネットワークに挿入する

# Batch Normalization

各層のアクティベーションの分布が適度な広がりを持つように強制的にアクティベーションの調整

利点：

・学習を早く進行させることができる（学習係数を大きくすることができる）
・初期値にそれほど依存しない（初期値に対してそこまで神経質にならなくて良い）
・過学習を抑制する（Dropoutなどの必要性を減らす）




Batch Normalizationアルゴリズム：

・各層のアクティベーションの分布が適度な広がりを持つように調整

→ データ分布の正規化(平均が0、分散が1の分布)を行うレイヤをニューラルネットワークに挿入

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

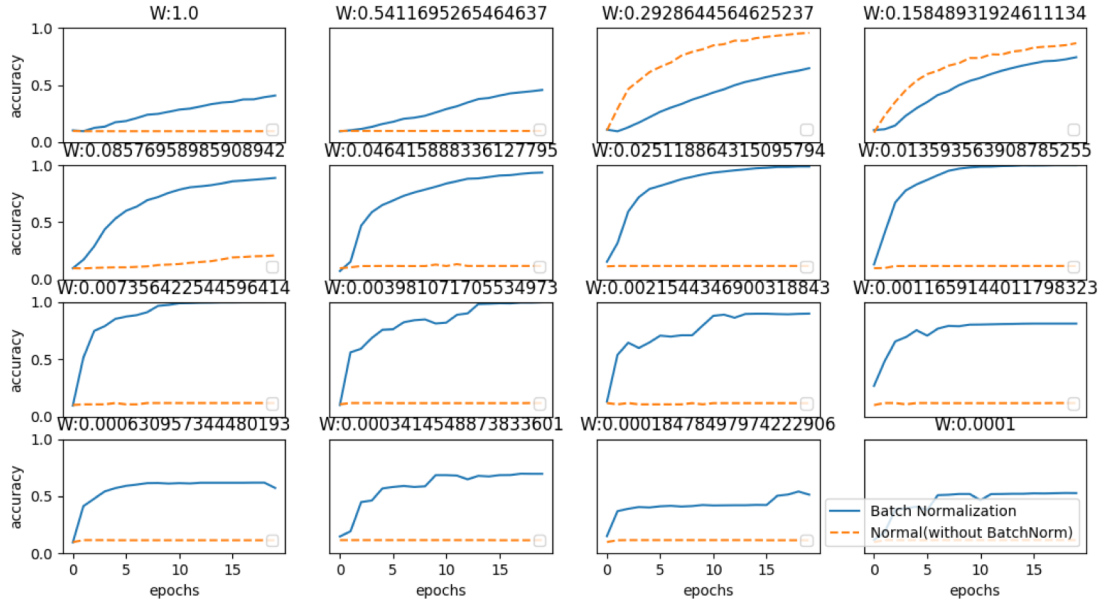
・Bath Normレイヤは、この正規化されたデータに対して、固有のスケールとシフトで変換

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

$\gamma, \beta$はパラメータで、最初は$\gamma = 1, \beta = 0$ $\beta$=0からスタートして、学習によって適した値に調整されていく

# Batch Normalizationの評価

重みの初期値の標準偏差をさまざまな値に
変えたときの学習経過のグラフ




実線：Batch Norm を使用した場合の結果
点線：Batch Norm を使用しなかった場合の結果
図のタイトル：重みの初期値の標準偏差

・ほとんどすべてのケースで、Batch Normを使用したほうが学習の進みが速い
・Batch Norm を用いない場合は、良い初期値のスケールを与えないと、まったく学習が進まない

↓

・Batch Norm を使用することで、学習の進行を促進させることができる
・重みの初期値にそれほど依存しなくなる

# まとめ

**1節　パラメータの更新**

・勾配法の種類はSGD, Momentum, AdaGrad, Adamがある（AdaGrad, Adamが学習スピードが早い）
・重みパラメータの初期値は小さくすることで過学習を防げる

**2節　重みの初期値**

・重みを均一な値にすると各層での誤差逆伝播法の出力が均等になってしまうので避ける
・各隠れ層のアクティベーションが、0と1に傾くと勾配損失が発生し、層を深くする意味がなくなる
・アクティベーションを分散させるには重みパラメータのスケールにHeの定数、Xavierの定数を適用
　・活性化関数が線形(sigmoid, tanh等)：Xavierの定数
　・活性化関数がReLU：Heの定数

**3節　Batch Normalization**

・強制的にアクティベーションの分布を制御するには活性化関数の前にBatch Normalization(正規化)
・正規化をしない場合、適切なスケールを設定しないといつまでたってもしっかりと学習されない
・正規化を使用すると重みの初期値に依存しない