JDLA E資格認定プログラム

全人類がわかる ディープラーニングコース DAY 3

全人類がわかる統計学 Presents

https://to-kei.net

目次

1. 学習と最適化計算
2. 基本最適化アルゴリズム
3. パラメータ初期化戦略
4. 応用最適化アルゴリズム
5. 最適化戦略とメタアルゴリズム

全コース紹介

DAY 1 順伝播型	DAY 2 前処理と工夫	DAY 3 データの学習
DAY 4 CNN	DAY 5 RNN	DAY 6 応用モデル演習

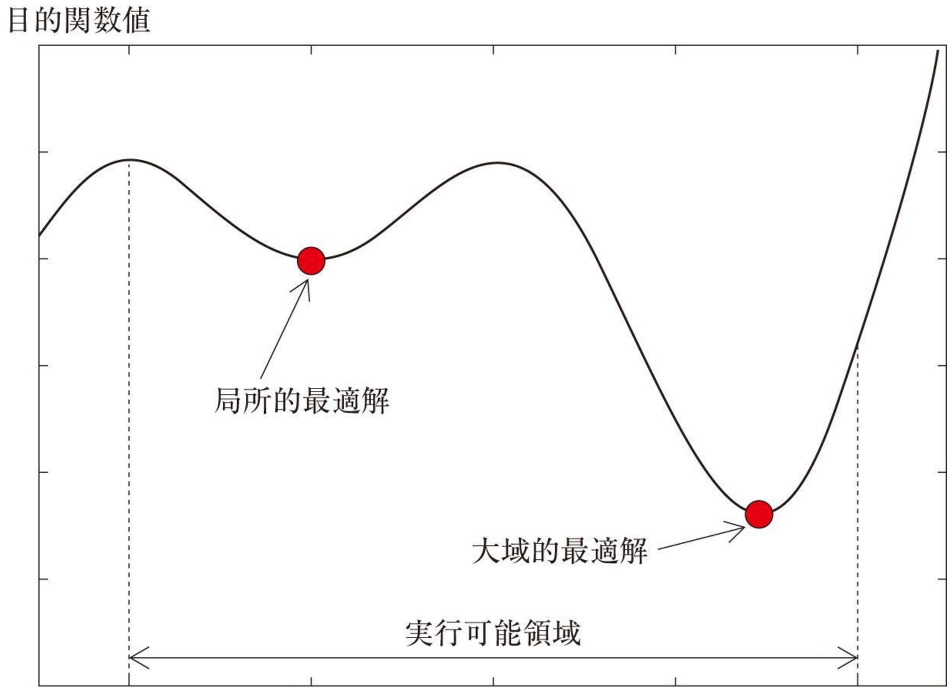
©全人類がわかる統計学

ディープラーニング体系講座 DAY3

学習と最適化計算

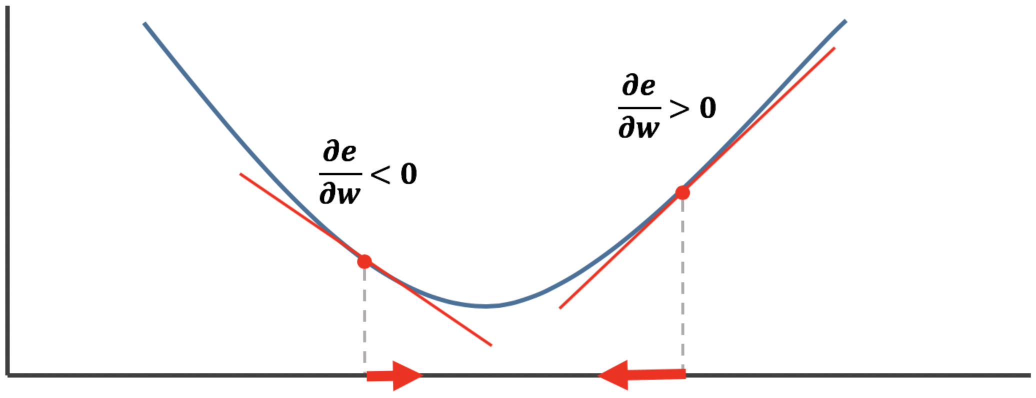
最適化とは

- 目的関数を最大（最小）にするようなパラメータをみつけること



勾配降下法

- 勾配の情報を使って目的関数を減らす方向にパラメータを更新していく



深層学習と最適化

- 深層学習の最適化計算は、純粋な最適化とは少し異なる
- 深層学習では最小化したい目的関数自体を計算できないので、代理の目的関数を設定する

一般的なコスト関数

$$J^*(\theta) = E_{(x,y)\sim p_{data}} L(f(x;\ \theta), y) \rightarrow minimize$$

深層学習でのコスト関数

$$J(\theta) = E_{(x,y)\sim \hat{p}_{data}} L(f(x;\ \theta), y) \rightarrow minimize$$

深層学習と最適化

- 深層学習の最適化計算は、純粋な最適化とは少し異なる
- 深層学習では最小化したい目的関数自体を計算できないので、代理の目的関数を設定する

一般的なコスト関数

データの生成分布
(入力として想定される
あらゆるデータ)

$$J^*(\theta) = E_{(x,y)\sim p_{data}} L(f(x;\theta),y) \rightarrow minimize$$

深層学習でのコスト関数

入力xの時の予測出力

$$J(\theta) = E_{(x,y)\sim \hat{p}_{data}} \underbrace{L(f(x;\theta),y)}_{\text{実例当たりの損失関数}} \rightarrow minimize$$

データの経験分布
(実際に学習に使用
したデータ)

実例当たりの
損失関数

経験損失最小化

$$J(\theta) = E_{(x,y)\sim\hat{p}_{data}} L(f(x;\ \theta), y) \rightarrow minimize$$

- この損失関数を各訓練データについて書き下すと

$$E_{(x,y)\sim\hat{p}_{data}}[L(f(x;\ \theta), y)] = \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)};\ \theta\right), y^{(i)}\right) \rightarrow minimize$$

m：訓練事例数

経験損失最小化

$$E_{(x,y)\sim\hat{p}_{data}}[L(f(x;\ \theta),y)] = \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)};\ \theta\right), y^{(i)}\right) \rightarrow minimize$$

- 経験損失と呼ばれる
- この経験損失を正確に計算しようとすると、訓練データ全てに対して損失を計算してようやく一回しか更新できない
- 深層学習や機械学習における最適化は、厳密な更新をゆっくり行うことよりも、近似してでも素早い更新を何度も繰り返すことが重要

→確率的勾配降下法・ミニバッチアルゴリズムが用いられる

バッチとミニバッチ

- 機械学習の最適化…目的関数が訓練事例の和で表せる
- 例えば最尤推定

$$\theta_{ML} = argmax_{\theta} \sum_{i=1}^{m} \log_{p_{model}}(x^{(i)}, y^{(i)}, \theta) \rightarrow maximize$$

- 全訓練事例（バッチ）での損失を最小化するという問題は、訓練データ集合をある程度のまとまり（ミニバッチ）に分割し、各ミニバッチでの損失を最小化するという問題として考えられる

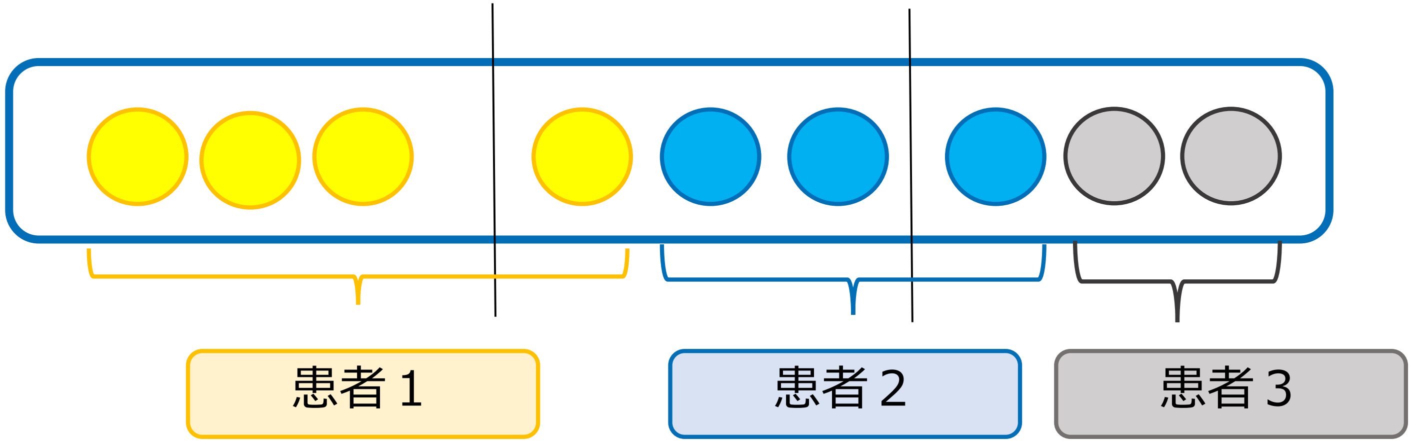
$$X = [\underbrace{\underbrace{x^{(1)}, x^{(2)}, \ldots, x^{(1000)}}_{\text{ミニバッチ } x^{\{m\}}}, x^{(1001)}, \ldots\ x^{(2000)}, \ldots \qquad \ldots, x^{(M)}}_{\text{バッチ}}]$$

;

$$Y = [y^{(1)}, y^{(2)}, \ldots, y^{(1000)}, y^{(1001)}, \ldots\ y^{(2000)}, \ldots \qquad \ldots, y^{(M)}]$$

ミニバッチの選択

- ミニバッチは選択前にシャッフルされる必要がある
- Ex) 血液検査のデータセット



- シャッフルされずに事例を順番に抽出すると
  計算時のバイアスが大きくなる

ミニバッチのサイズ

ミニバッチのデータ数のことをバッチサイズという

(問) バッチサイズが大きい場合と小さい場合にどのようなことが起こり得るか？

ミニバッチのサイズ

ミニバッチのデータ数のことをバッチサイズという

(問) バッチサイズが大きい場合と小さい場合にどのようなことが起こり得るか？

(答) バッチサイズが大きい場合

計算時間がかかる、必要メモリ量が増える

バッチサイズが小さい場合

パラメータの更新方向が安定しない（非常に小さい場合）

ミニバッチのサイズ

- バッチサイズが小さいと勾配の方向が安定しないのではないか？

(問) 各データで計算した損失の標準偏差をσとして、バッチサイズnのミニバッチに分割した後、各ミニバッチについて計算した損失の期待値をバッチでの損失の期待値として推定した場合について、標準偏差はどれほどになるか？

ミニバッチのサイズ

- バッチサイズが小さいと勾配の方向が安定しないのではないか？

(問) 各データで計算した損失の標準偏差をσとして、バッチサイズnのミニバッチに分割した後、各ミニバッチについて計算した損失の期待値をバッチでの損失の期待値として推定した場合について、標準偏差はどれほどになるか？

(答) $\frac{\sigma}{\sqrt{n}}$ →勾配推定の精度は、バッチサイズに対して線形以下

バッチサイズを10000倍にすると、計算量は10000倍だが精度は100倍しか改善しない

ミニバッチのサイズ

- ハードウェアの限界（メモリ量）や、勾配推定の安定性などを考慮して決める重要なハイパーパラメータ
- GPU計算を行う場合 32,64,128,256 が一般的（たまに16）

ミニバッチ確率的勾配降下法（詳細は後節）

入力x、出力yがともに離散的である場合、汎化誤差は

$$J^*(\boldsymbol{\theta}) = \sum_x \sum_y p_{data}(\boldsymbol{x}, y)\, L(f(\boldsymbol{x};\ \boldsymbol{\theta}), y) \to minimize$$

上式の勾配は

$$g = \nabla_\theta J^*(\boldsymbol{\theta}) = \sum_x \sum_y p_{data}(\boldsymbol{x}, y)\, \nabla_\theta L(f(\boldsymbol{x};\ \boldsymbol{\theta}), y)$$

ミニバッチを用いると

$$\hat{g} = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}))$$

θを$\hat{g}$の方向へ更新するとSGDが実行される

©全人類がわかる統計学

本節のまとめ

- 深層学習では、直接データの生成分布についての損失を最適化するわけではなく、他の指標（経験損失）を最適化することで精度の改善を見込む
- 訓練事例をいくつかのデータセットに分ける
ミニバッチも非常によくつかわれる手段
- 確率的勾配降下法は最も基本的なアルゴリズムであり、
後節で学習する

ディープラーニング体系講座 DAY3

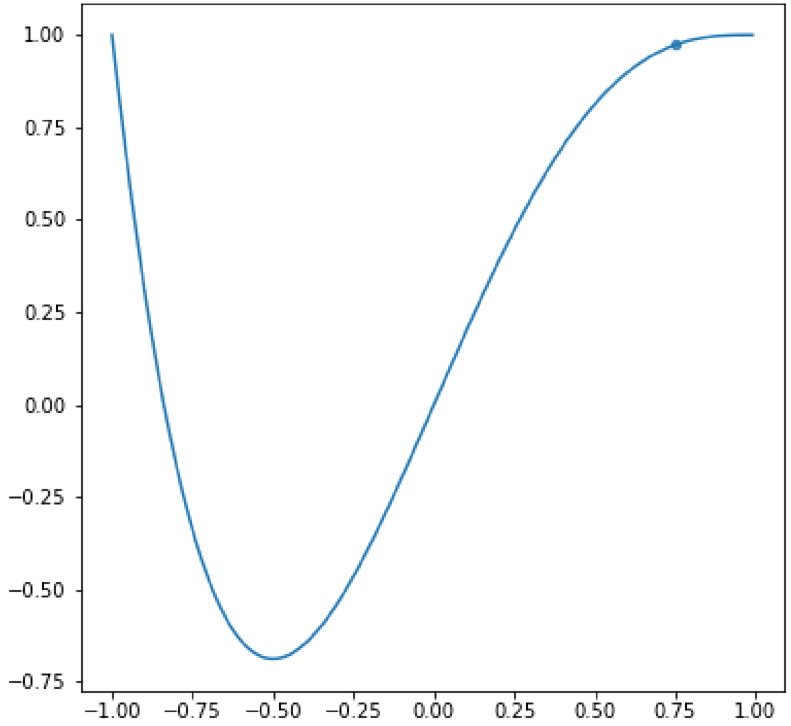
アルゴリズム（基本）

確率的勾配降下法 (Stochastic Gradient Descent)

- 反復法を用いて重みw やバイアス b を更新する

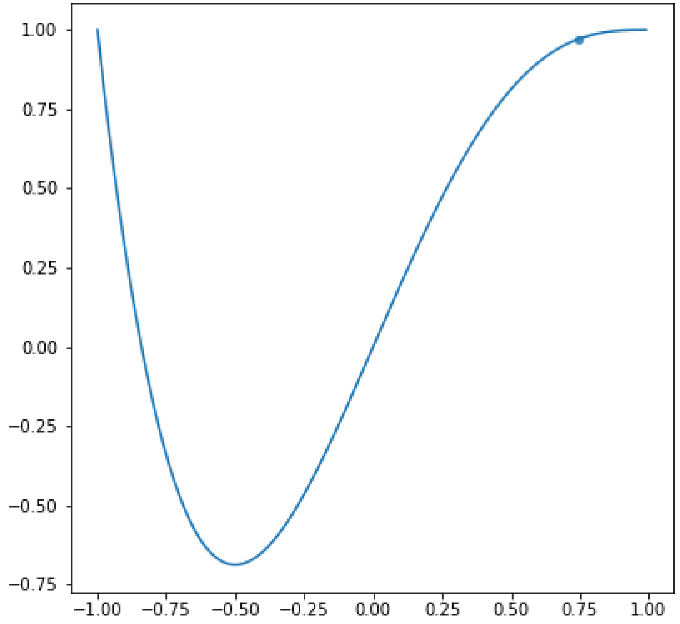
$$\theta \leftarrow \theta - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right) \right)$$

- ϵ を学習率と言い、機械学習において最も重要なハイパーパラメータの一つ
- $\dfrac{de}{dw}$ は e を wで微分したものであり、グラフにおける傾きと考えることができる

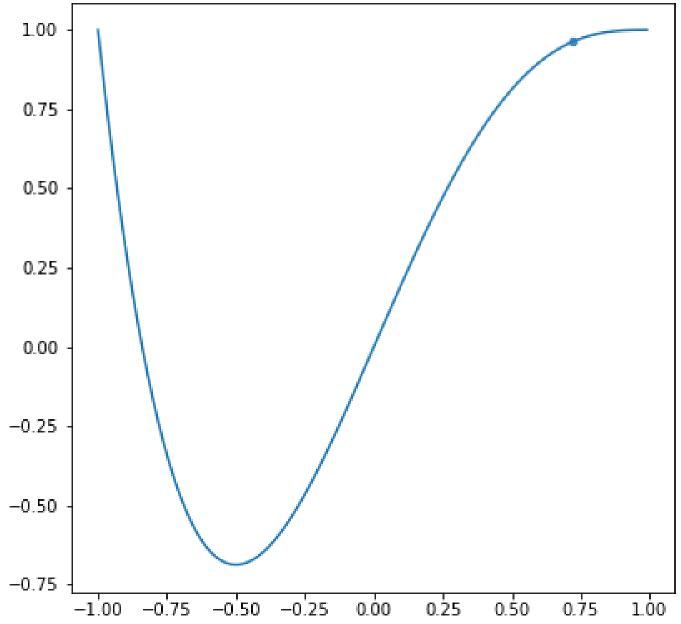


学習率 ϵ による違い

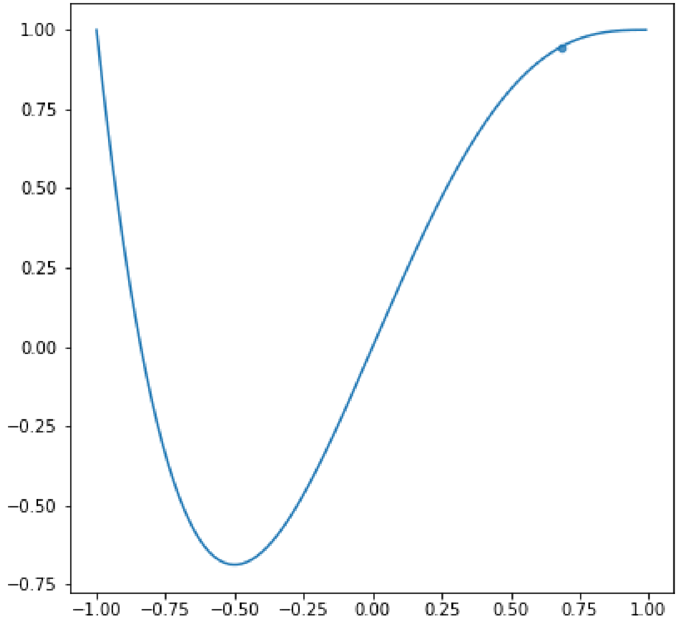
$\epsilon = 0.01$ 過小



$\epsilon = 0.1$ 適度



$\epsilon = 0.2$ 過大



学習率 ϵ

- SGDは以下の条件で収束

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \qquad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

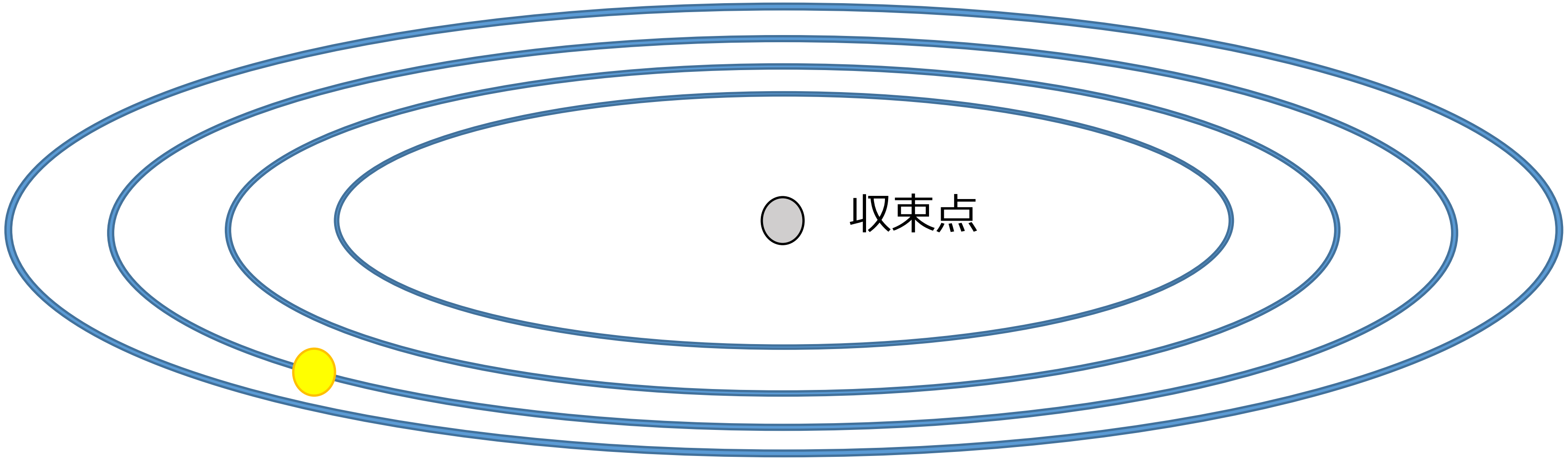
- よくつかわれる学習率(δ は定数)

$$\epsilon_k = \frac{\epsilon}{1 + \delta t} \qquad \epsilon_k = \frac{\epsilon}{t^{\delta}} \ (0.5 < \delta \leq 1)$$

ϵ の設定は学習データ、NNの構造によりけり、
今の技術では経験と試行錯誤的に決める場合がほとんど

練習問題

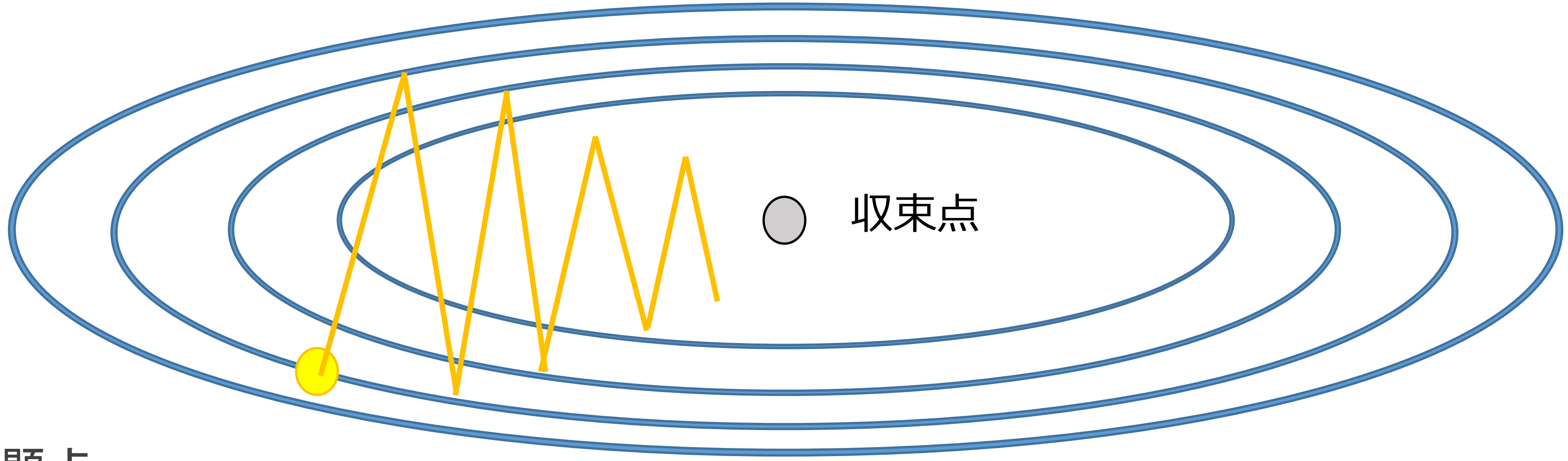
1. 以下のような損失関数のマップがあった場合、
SGDはどのような挙動を示すか？



2. 問1から考えられる、SGDの問題点は何か
3. SGDの核となる部分のプログラムを書いてみよう（勾配計算の関数は逆伝播の実装なので省略して良い）

練習問題の解答

1. 以下のような損失関数のマップがあった場合、
SGDはどのような挙動を示すか？



2. 問題点

損失関数の勾配が小さい方向に学習がなかなか進まないので、
ジグザグに進んでいき、学習時間がとても遅くなる

練習問題の解答2

3. SGDの中心部のコード

```
while True: # 適当な条件を設定する
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

モメンタム

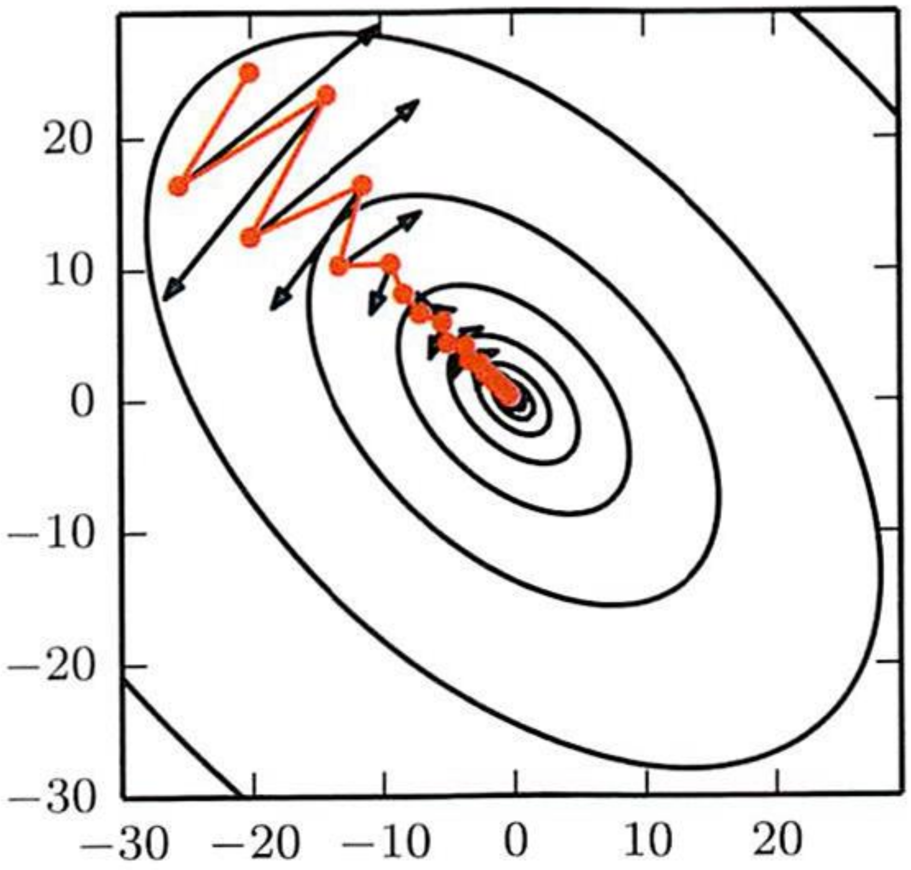
- SGDはよく使用されるが、学習が遅い場合がある
→学習を高速化するためのアルゴリズム

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)};\ \theta), y^{(i)}) \right)$$

$$\theta \leftarrow \theta + v$$

- ※ $\alpha \in [0, 1)$はハイパーパラメータ
典型的な値としては0.5, 0.9, 0.99など

- 前までの更新の勢いに引っ張られるイメージ



ネステロフのモメンタム

- モメンタムアルゴリズムの派生形

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L\left(f\left(x^{(i)};\ \theta + \alpha v\right), y^{(i)}\right) \right)$$

$$\theta \leftarrow \theta + v$$

ネステロフのモメンタム

- モメンタムアルゴリズムの派生形

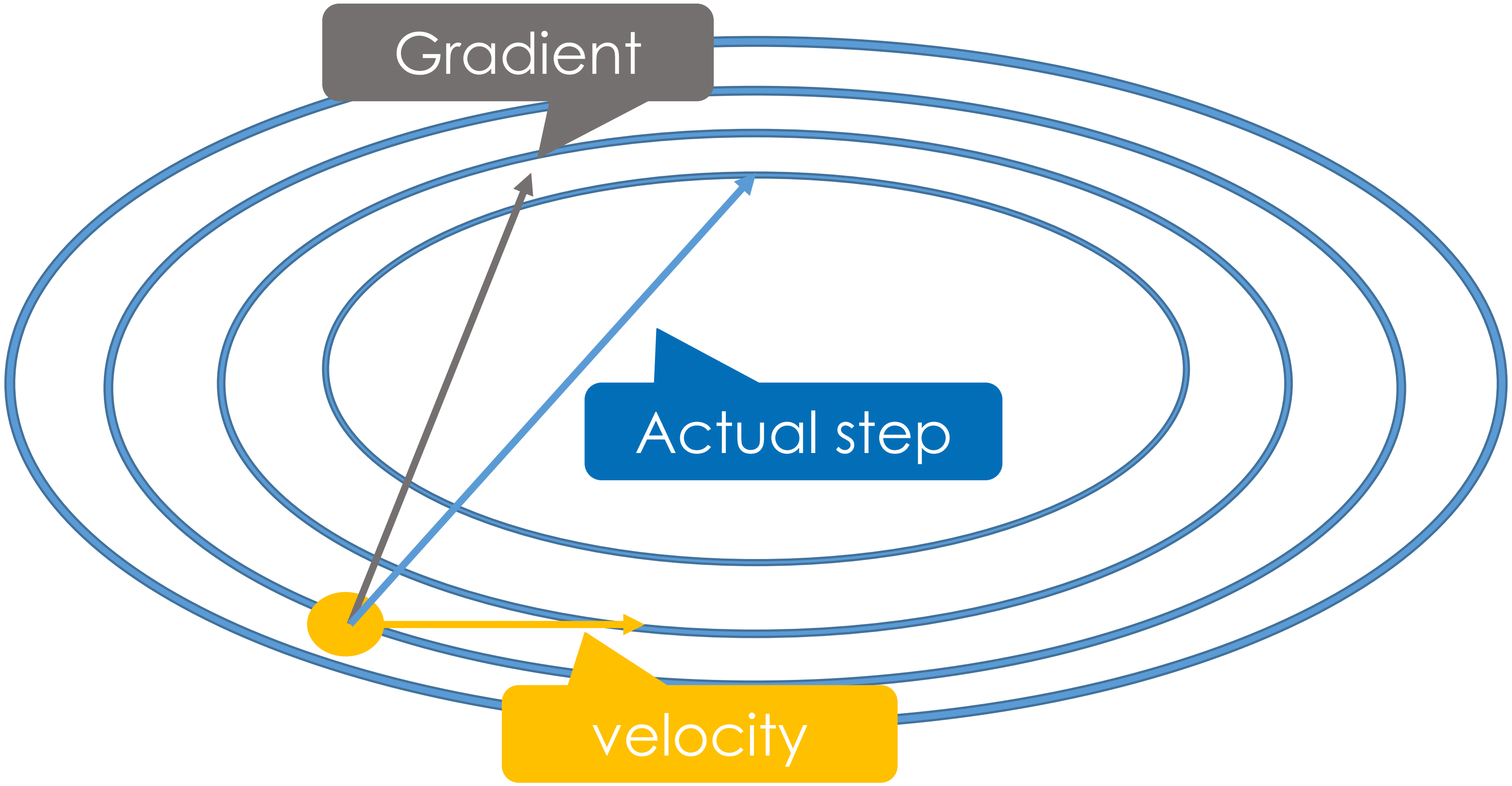
$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L\left(f\left(x^{(i)}; \boxed{\theta + \alpha v}\right), y^{(i)}\right) \right)$$

$$\theta \leftarrow \theta + v$$

- 現在の速度が適用された移動先での勾配の評価

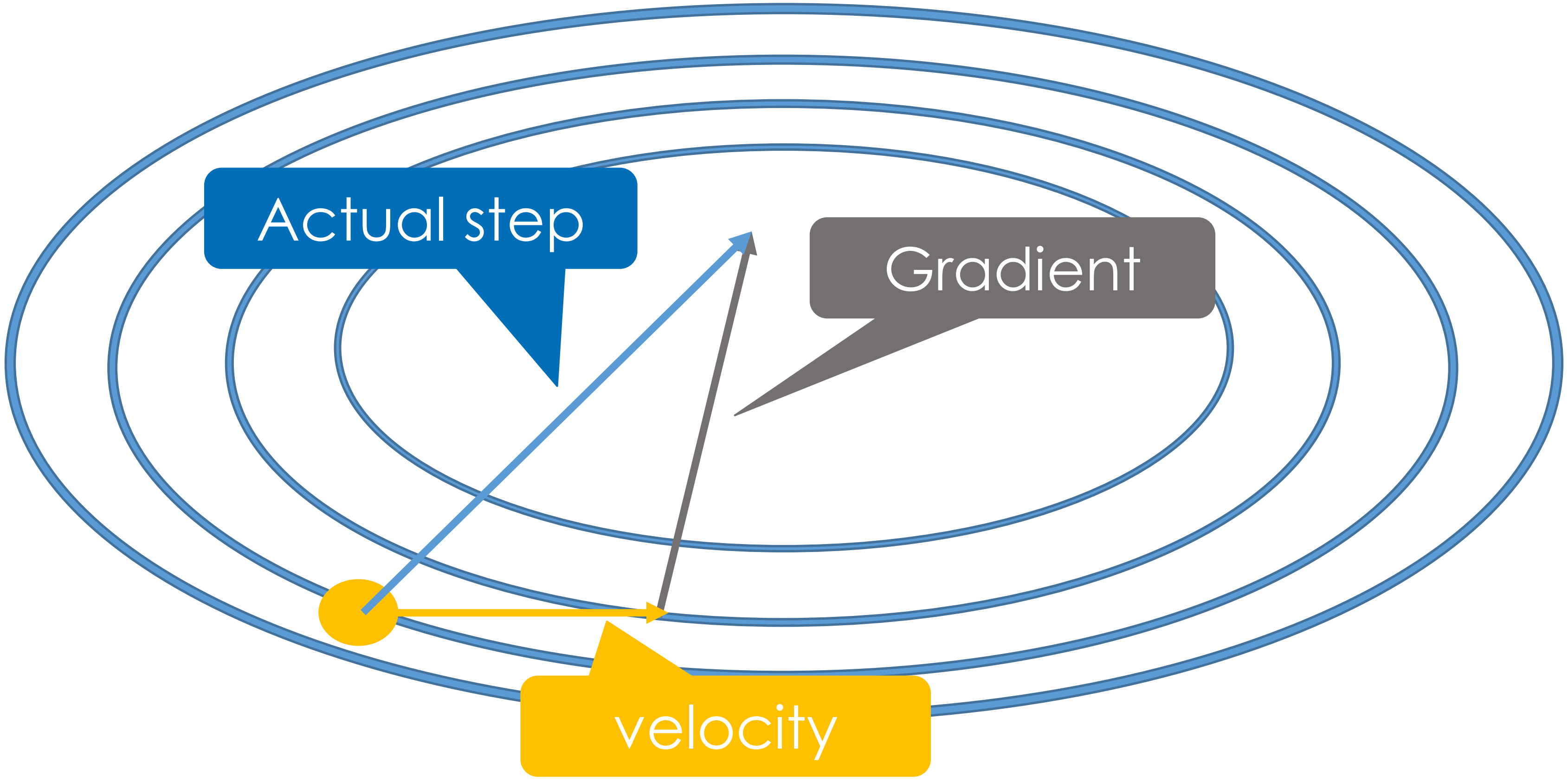
モメンタムの種類(イメージ)

- 一般的なモーメンタム



モメンタムの種類(イメージ)

- ネステロフのモーメンタム



練習問題

- 一般的なモーメンタムアルゴリズムとネステロフ型モーメンタムアルゴリズムを実装せよ

練習問題のヒント

- 一般的なモーメンタム

$$
\begin{aligned}
v_{t+1} &= \rho v_t - \alpha \nabla f(x_t) \\
x_{t+1} &= x_t + v_{t+1}
\end{aligned}
$$

- ネステロフモーメンタム

$$
\begin{aligned}
v_{t+1} &= \rho v_t - \alpha \nabla f(x_t + \rho v_t) \\
x_{t+1} &= x_t + v_{t+1}
\end{aligned}
$$

$\tilde{x}_t = x_t + \rho v_t$
により変数変換

$$
v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)
$$

$$
\begin{aligned}
\tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1} \\
&= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)
\end{aligned}
$$

練習問題解答

- 一般的なモーメンタム

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
# Code for Momentum
vx = 0
rho = 0.9
while True:
    dx = compute_gradient(x)
    vx = rho * vx - learning_rate * dx
    x += vx
```

- ネステロフモーメンタム

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\begin{aligned}\tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)\end{aligned}$$

```
# Code for Nesterov Momentum
while True:
    dx = compute_gradient(x)
    old_v = v
    v = rho * v - learning_rate * dx
    x += -rho * old_v + (1 + rho) * v
```

本節のまとめ

- ニューラルネットワークの学習に用いられる基本的な
  アルゴリズムとして、確率的勾配降下法がある

- しかしながら、確率的勾配降下法では学習が遅くなる場合がある

- そこで前回の勾配の速度を用いて早く最適化を
  終了させようとするアルゴリズムが
  モーメンタムアルゴリズムである。

ディープラーニング体系講座 DAY3

パラメータの初期化戦略

初期化戦略の難しさ

- 深層モデルの学習…非常に難しいタスクの一種、
  学習がうまくいくかは初期値に依存する
- この初期値の設定については試行錯誤で行われている
- 設定した初期値が最適化にはよくても、
  汎化性能に悪影響を与える場合もある

- 重みパラメータの初期化
  …同一の活性化関数をもつユニット間では、
  初期パラメータによってその「対称性を破る」必要がある
  ex) 正規化された初期化など経験則
- バイアスパラメータの初期化

ディープラーニング体系講座 DAY3

アルゴリズム(応用)

AdaGrad

- 学習率を各パラメータ（方向）で可変にする
- モーメンタムアルゴリズムではvelocityが追加されていたのに対し、AdaGradではその方向の勾配の大きさの指標が追加される
- つまり、
  その方向の勾配が大　→　小さな変化
  その方向の勾配が小　→　大きな変化

AdaGradのアルゴリズム

- アルゴリズム

1. 訓練集合からm個の事例をミニバッチサンプリング
2. 勾配の計算

$$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$$

3. 勾配の二乗を蓄積（行列積ではなく要素積で計算する）

$$r \leftarrow r + g \odot g$$

4. 更新

$$\Delta\theta \leftarrow \frac{\varepsilon}{\delta + \sqrt{r}} \odot g$$

練習問題

1. タイムステップが十分経過したとき、
   AdaGradの挙動はどうなるか考えよ。

- （ヒント）AdaGradは前のアルゴリズムの1-4を収束するまで繰り返すので、十分回数繰り返した際に何が起こりそうかを考えてみればよい

2. AdaGradを実装するようなコードを考えてみよ。

練習問題の解答

1. 十分時間がたつと勾配がどんどん小さくなっていく問題
   最適解に辿り着く前に止まってしまう可能性

2. コードとしてはこのようになる

```
delta = 1e-7
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + delta)
```

RMSProp

- AdaGradの修正版
- AdaGradでは勾配が時間とともに減少していたが、それを解消する方策
- 減衰率を用いることで、現在のタイムステップの勾配の影響を抑える
- 累積二乗和ではなく、移動平均を使用する

RMSPropのアルゴリズム

AdaGradのアルゴリズム

```
delta = 1e-7
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + delta)
```

RMSPropのアルゴリズム

```
delta = 1e-7
grad_squared = 0
decay_rate = 0.9
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1-decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + delta)
```

Adam (Adaptive Momentum)

- モメンタムアルゴリズムとAdaGrad/RMSPropの組み合わせ
- かなり広く使用されているアルゴリズムである

Adam (Adaptive Momentum)のアルゴリズム(仮)

```
# Code for Adam
first_moment = 0
second_moment = 0
beta1, beta2 = 0.9, 0.999
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + delta)
```

問. このアルゴリズムを行うと初期段階の
挙動はどうなるか考察せよ

Adam (Adaptive Momentum)のアルゴリズム(仮)

```
# Code for Adam
first_moment = 0
second_moment = 0
beta1, beta2 = 0.9, 0.999
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + delta)
```

問. このアルゴリズムを行うと初期段階の挙動はどうなるか考察せよ
答え．初期段階では、second_moment=0のため
変動がとても大きくなり不安定になる

Adam (Adaptive Momentum)のアルゴリズム(完全)

```
# Code for Adam
first_moment = 0
second_moment = 0
beta1, beta2 = 0.9, 0.999
for i in range(num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + delta)
```

ここで調整

本節のまとめ

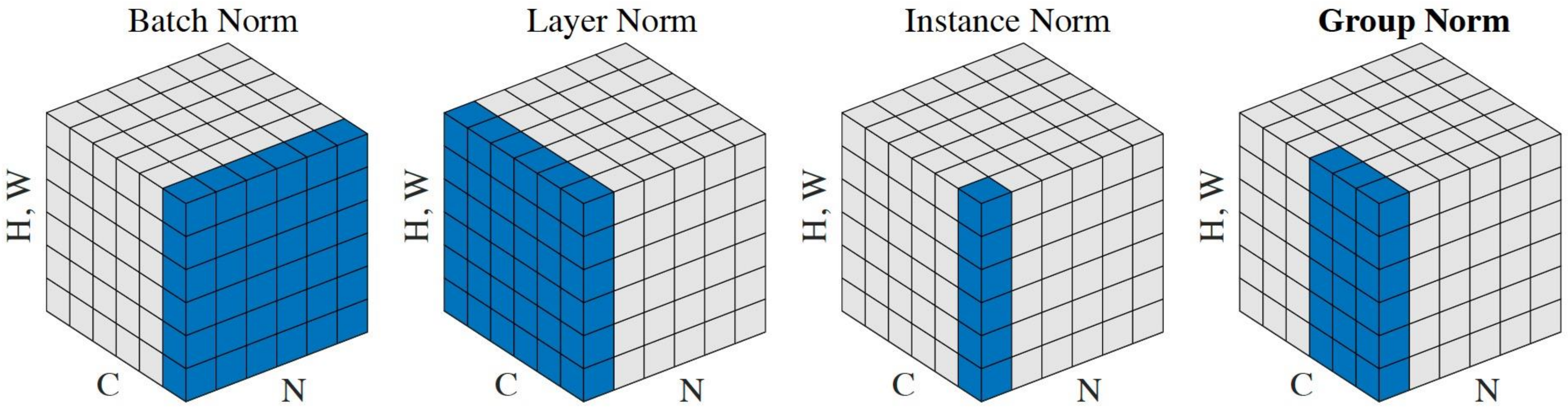
- 学習をすすめるにあたって行う損失の最適化には、前節で紹介したような確率的勾配降下法以外にも、AdaGradやRMSProp、Adamなどが使われる。
- 特にAdamはオーバーシュート（飛び越え）も少なく、比較的学習時間も短いため非常によく用いられる。

ディープラーニング体系講座 DAY3

最適化戦略と
メタアルゴリズム

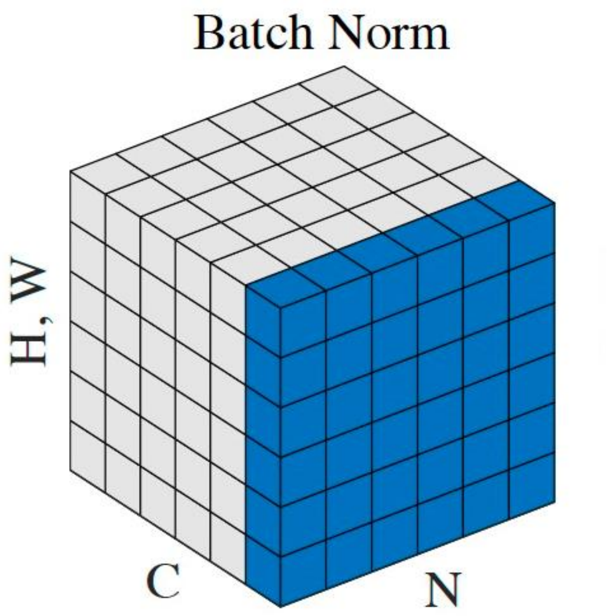
様々なNormalization

- 深いモデルの訓練に関するデータの学習の難しさを解決する手法
- 主に画像処理の分野で使用されることが多い



引用 : https://arxiv.org/abs/1803.08494

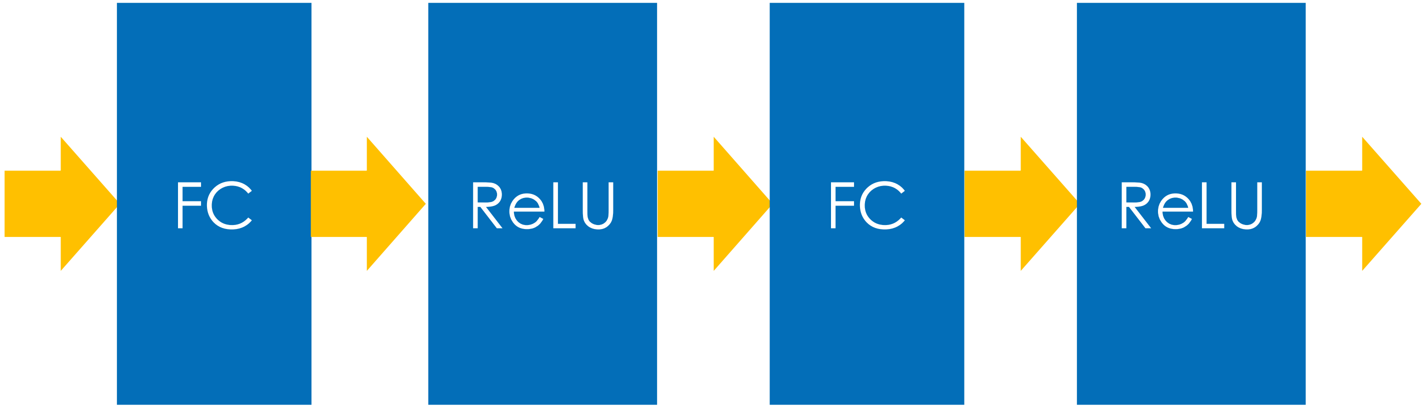
Batch Normalization



引用 : https://arxiv.org/abs/1803.08494

- 学習データに(N, C, HW) の
  3 次元からなる多次元配列を考える
- 例えば CNN の中間層の出力だと考えると、バッチサイズ N, チャンネル数 C, 画像の縦幅 H, 画像の横幅 W と捉えることができる。
- このとき、Batch Normalizationは青色の部分の入力を正規化することに同義である

Batch Normalization



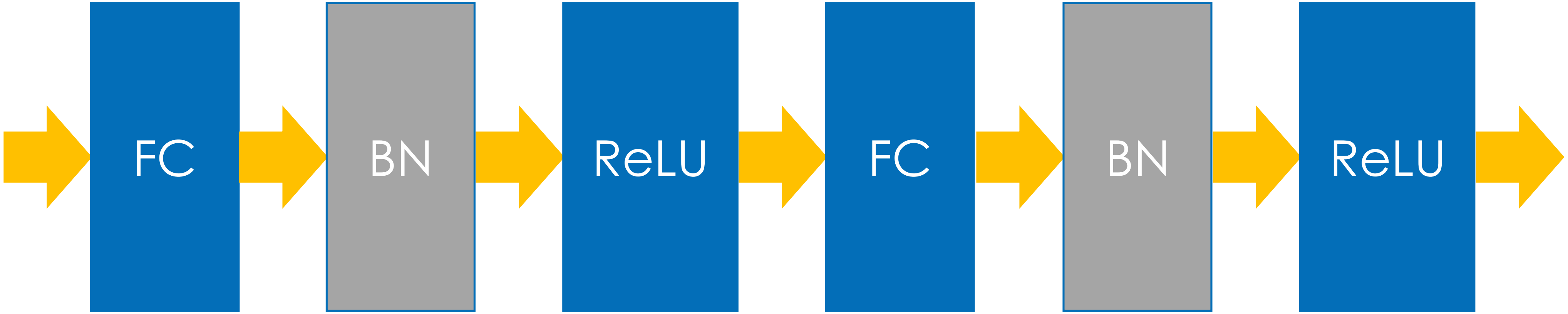
- パラメータの勾配は偏微分で、すなわち他のパラメータが変わらない前提で損失を減らす方向を計算している
- ネットワークが深いと、非線形変換を何度も繰り返すため各層の入力データの分布が大きく変わる（内部の共変量シフト）

→　各ノードの値をミニバッチ単位で正規化することで、
　　パラメータのスケールを揃える（バッチ正規化）

©全人類がわかる統計学

Batch Normalization 層

- 各線形変換と非線形変換の間に入れることが多い



- バッチ正規化によりパラメータの初期値に注意を払う必要が低下する
- 正則化としても機能するため、L2正則化やDropoutの必要性が低下
- バッチサイズが小さすぎると機能しにくい

Batch Normalization

- バッチ正規化の具体的なアルゴリズム
- 各入力次元に対して、
  ミニバッチ内での
  平均と分散を計算する

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

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

- 各入力次元に対して
  正規化を行う

$$\hat{x}^{(k)} \leftarrow \frac{x^{(k)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- スケーリング・シフトを
  行う

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \equiv BN_{\gamma,B}(x_i)$$

©全人類がわかる統計学

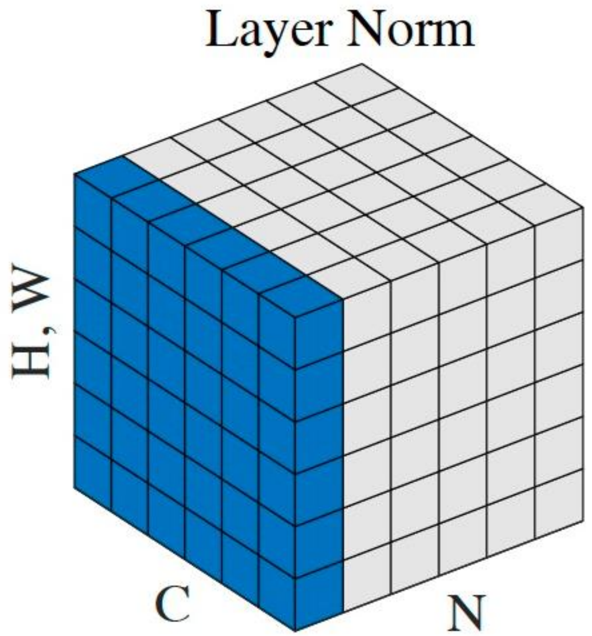
Batch Normalization

- テスト時はどうするか？
- テストデータに対して平均・分散を求めて正規化してしまうと、1つだけの事例に対して予測をしたい場合などにうまくいかない
- 全訓練データ（フルバッチ）の平均・分散を使用して正規化を行えば良い
- ミニバッチでの平均・分散を使って移動平均によりフルバッチでの平均・分散を推定することで、全てミニバッチでの処理にすることができる

$$\mu_{new} = m\mu_{old} + (1-m)\mu$$

$$\sigma^2_{new} = m\sigma^2_{old} + (1-m)\sigma^2$$

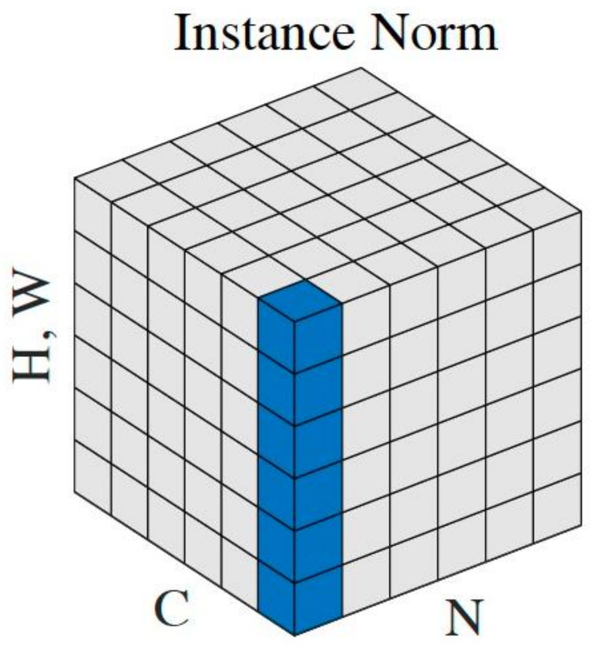
Layer Normalization



- 同じ層のニューロン間で正規化を施す手法
- 図の例では全てのチャンネルにまたがり平均・分散をとる
- Batch Normalizationと異なり、トレーニングとテストで同じ計算を実行する。
- オンライン学習やRNNに拡張可能

引用 : https://arxiv.org/abs/1803.08494

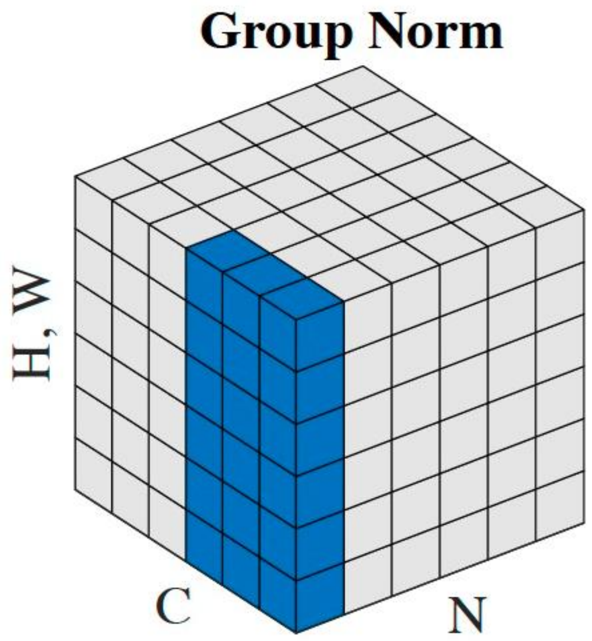
Instance Normalization



引用 : https://arxiv.org/abs/1803.08494

- 各チャンネルで独立に画像の縦横方向についてのみ
  平均・分散をとる手法
- バッチサイズが十分に確保されている条件下では Batch Normalizationに比べて劣っており、あまり主流ではない手法である
- RNNなど画像以外の分野にはあまり拡張できない
- 画像生成などの分野ではバッチ正規化の代わりとして注目されている

Group Normalization



引用 : https://arxiv.org/abs/1803.08494

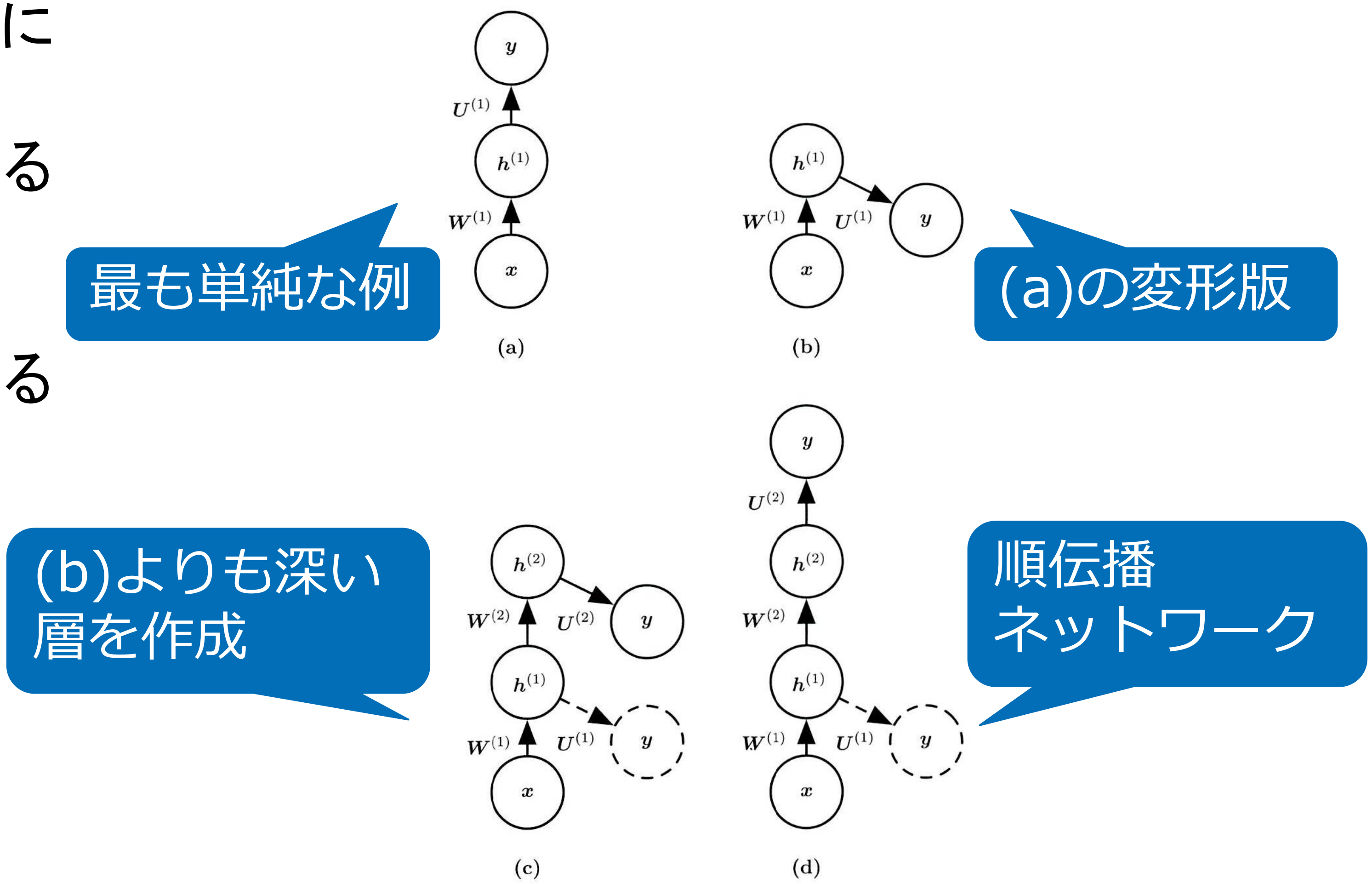
- チャンネルをG個にグルーピングし Layer Normalizationと Instance Normalization の中間的な処理を行う
- 任意に定義されたグループごとのチャンネルを正規化し入力として使用する
- 物体検出やセグメンテーションの分野において COCOやKineticsなどの一部のデータセットで Batch Normalizationよりも強力な手法であることが報告されている

教師あり事前学習

- モデルが複雑な場合に、比較的単純なモデルを始めに訓練してそれを徐々に複雑にしていく方法
- この方が効果的にモデルを作成できる可能性がある
- 貪欲法が代表例

教師あり事前学習 –貪欲法 (Greedy Algorithm)-

- 貪欲法は問題を多くの要素に分割後、その要素ごとに個別に最適な要素を決定する
- 計算コストが低く、最適解でなくとも許容できる精度になることが多い



まとめ

- ニューラルネットワークの学習のための最適化には、いくつか代表的なテクニックがある
- バッチ正規化はその一例であり、入力データを整形し、訓練を行いやすくする
- 事前に単純な問題を設定し、その後そのモデルを複雑にしていく事前学習もその一例であり、代表的なものとしては、貪欲法によるものがあげられる

ディープラーニング体系講座 DAY3

深層学習ライブラリ

Framework1

- TensorFlow(https://www.tensorflow.org/)
  - Googleが主導して開発がすすめられている最も有名なオープンソースライブラリ
  - コードを書くためにはある程度のディープラーニングの知識が必要
  - 静的ネットワーク構築（define and run）
- Chainer(https://chainer.org/)
  - 日本発のディープラーニング用フレームワークで、国内でも多くの企業が採用
  - Preferred Networks社という日本企業が開発
  - 動的ネットワーク構築（define by run）
- PyTorch(http://pytorch.org/)
  - Facebook社の人工知能研究グループが開発したオープンソースの機械学習用ライブラリ
  - 動的ネットワーク構築（define by run）

Framework2

- MXNet(https://mxnet.incubator.apache.org/)
  - AWSで公式サポートされているフレームワーク
  - 動的ネットワーク構築と静的ネットワーク構築を併用
- CNTK(https://www.microsoft.com/en-us/cognitive-toolkit/)
  - マイクロソフトが主導しているフレームワーク
  - 元々音声処理用なのでRNNに強い
  - 静的ネットワーク構築（define and run）
- Theano(http://deeplearning.net/software/theano/)
  - ほぼ最古のライブラリで一時代を築いたが今は開発終了
  - 静的ネットワーク構築（define and run）
- Caffe2(https://caffe2.ai/)
  - Facebookが主導して開発を行っているフレームワーク
  - 静的ネットワーク構築（define and run）

Higher API

- ディープラーニングフレームワークを抽象化して、
  簡単にかけるようにしたライブラリ

Keras

- Keras(https://keras.io/ja/)
  - 主にTensorFlowを抽象化し、より簡単にDNNを扱えるようにしたライブラリ
  - TensorFlow, Theano，CNTKなどのライブラリを扱える

GLUON

- Gluon(https://gluon.mxnet.io/)
  - AWSとMSが開発
  - MXNetが主なターゲット

アンケートのお願い/SNSなど

今日の講座をより良くするため、以下のURLからアンケートの協力をお願いしております。
https://seminar.to-kei.net/qt/

LINE@ イベントや講座の優先告知、事務的な質問等をすぐに返します。
https://avilen.co.jp/contact/

twitter 世の中のAIニュースをビジネス視点で紹介するアカウント
https://twitter.com/toukei_net

Thank you