FIXSTARS Amplify

量子コンピュータ時代のプログラミングセミナー

～ブラックボックス最適化による機械学習の特徴量選択・コスト削減・精度向上～

本日の予定

第一部

- 本セミナーのゴール
- 会社紹介
- Fixstars Amplify の紹介
- 組合せ最適化事例
- ワークショップ事前準備

第二部

- 組合せ最適化の基本
  - 数の分割ハンズオン

第三部

- ブラックボックス最適化とは
- FMQAの概要とフロー
- 機械学習の特徴量選択ハンズオン
  - 問題の説明
  - FM
  - FMQA
- まとめ

質問は随時 Zoom の Q&A へお願いします

本セミナーのゴール

組合せ最適化に関する次の項目を実感していただく

- 身の回りに組合せ最適化問題は多い。一方で、複雑で非線形な物理・社会現象の場合、組合せ最適化に必要な定式化が困難なものもある。
- 機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化により、複雑現象に対する最適化を実施可能。
- ブラックボックス最適化を Fixstars Amplify により実施できる。

質問は随時 Zoom の Q&A へお願いします

会社紹介

フィックスターズグループの基本情報

会社名	株式会社フィックスターズ
本社所在地	東京都港区芝浦3-1-1 msb Tamachi 田町ステーションタワーN 28階
設立	2002年8月
上場区分	東証プライム（証券コード：3687）
代表取締役社長	三木　聡

資本金	5億5,446万円
社員数（連結）	292名（2023年9月現在）
主なお客様	キオクシア株式会社 ルネサスエレクトロニクス株式会社 トヨタグループ（トヨタ自動車株式会社・ 豊田通商株式会社・株式会社デンソー） みずほ証券株式会社 キヤノン株式会社

グループ会社

Fixstars Solutions, Inc.
完全子会社
米国での営業及び開発を担当

(株) Fixstars Autonomous Technologies
株式会社ネクスティ エレクトロニクスとのJV
自動運転向けソフトウェアを開発

2021/10/1 設立

(株) Fixstars Amplify
完全子会社
最適化と量子コンピューティングのクラウド事業を運営

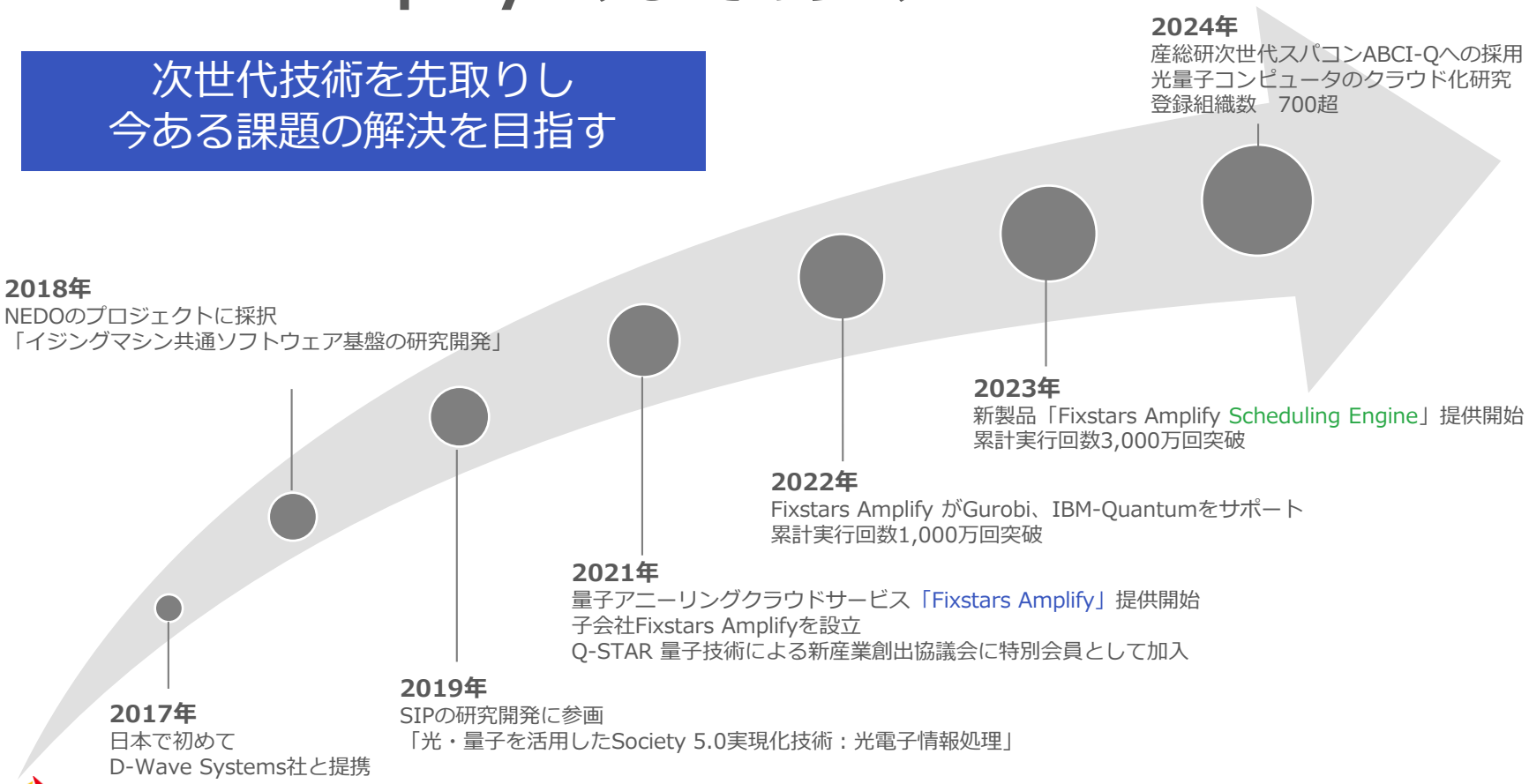
(株) Sider
完全子会社
開発支援SaaS「Sider」を運営

(株) Smart Opinion
連結子会社
乳がんAI画像診断支援事業を運営

オスカーテクノロジー (株)
連結子会社
ソフトウェア自動並列化サービスを提供

Fixstars Amplify: 今までの歩み

次世代技術を先取りし
今ある課題の解決を目指す



最適化と量子技術、Fixstars Amplify

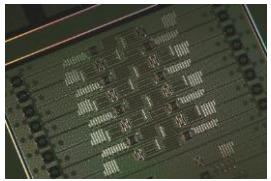
最適化と量子技術、Fixstars Amplifyの対応領域

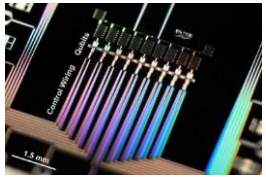
1. 量子コンピュータ

量子ゲート方式

古典汎用コンピュータの上位互換。量子力学の重ね合わせ状態を制御する量子ゲートを操作し、特定の問題を汎用的かつ高速に処理する。

QAOAにより組合せ最適化問題（**QUBO**）を取り扱うことが可能。





2. 量子アニーリング方式

量子焼きなまし法

イジングマシンの一種であり、量子焼きなまし法の原理に基づいて動作する。量子イジング模型を物理的に搭載したプロセッサで実現する。自然計算により低エネルギー状態が出力される。組合せ最適化問題（**QUBO**）を扱う専用マシン。

3. イジングマシン

二値二次多項式模型

二次の多変数多項式で表される目的関数の組合せ最適化問題（**QUBO**）を扱う専用マシン。

変数は0,1または±1。統計物理学におけるイジング模型（磁性体の性質を表す模型）に由来。様々な実装により実現されている。

Amplify AE

組合せ最適化

数理最適化問題

- 連続最適化問題
  - 決定変数が連続値（実数など）
- 決定変数が離散値 (整数など)
  - 整数計画問題 (決定変数が整数)
  - 0-1整数計画問題 (決定変数が二値)

量子アニーリング・イジングマシン

Quadratic	二次形
Unconstrained	制約条件なし
Binary	0-1整数 (二値)
Optimization	計画（最適化）

QUBO目的関数 (0-1整数二次計画問題)

$$f(\boldsymbol{q}) = \sum_{i<j} Q_{ij} q_i q_j + \sum_{i} Q_{ii} q_i$$

$f(\boldsymbol{q})$を最小化するような $\boldsymbol{q}$ を求める

f: 目的関数　q: 決定変数　Q: 係数

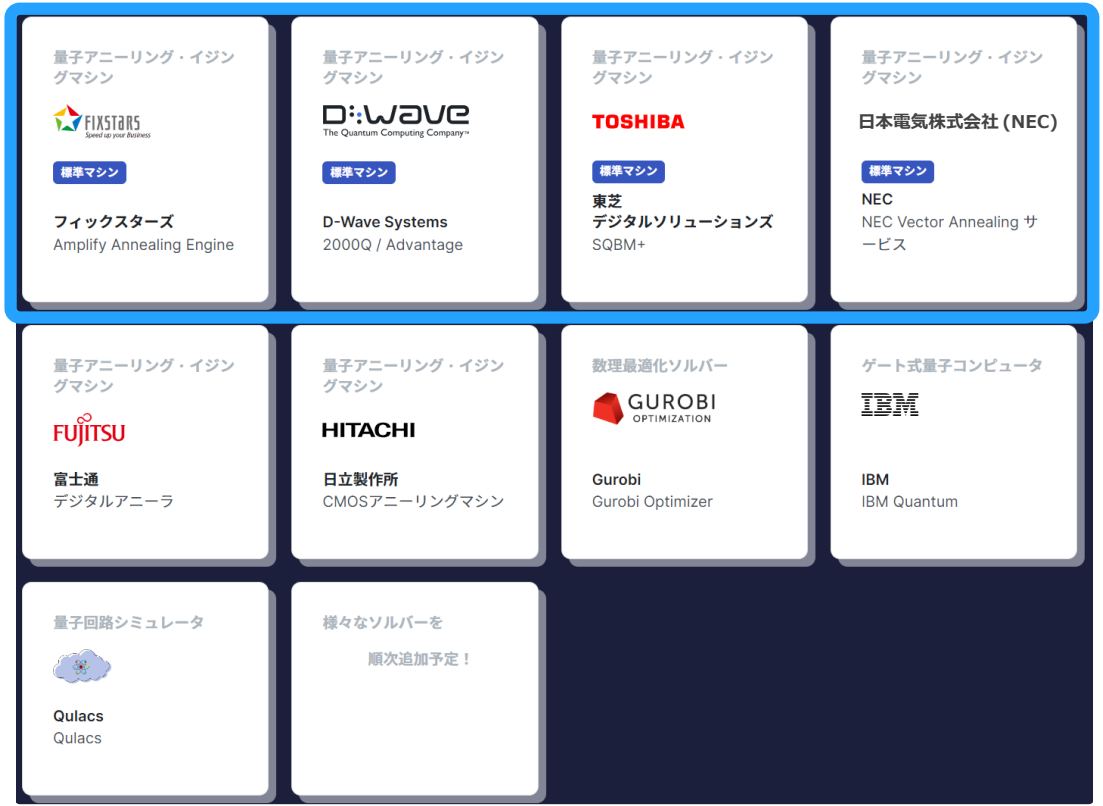
クラウドサービス：**Fixstars Amplify**

Fixstars Amplify とは

- **いつでも**　開発環境 と 実行環境 がセット
  すぐにプログラミングと実行が出来る
- **誰でも**　ハードウェアや専門的な知識が不要
  無料で開発がスタート可能
  多くの解説、サンプルコード
- **高速に**　26万ビットクラスの大規模問題の
  高速処理と高速実行が可能
- **あらゆる**　一般に公開されている全てのイジング
  マシンを利用可能

FIXSTARS Amplify

Fixstars Amplify の対応マシンの一例



標準マシン は、

- ベンダ各社と個別マシン利用契約なし、
- 評価・検証用ベーシックプランなら無料、

で利用可能！←「いつでも」、「誰でも」

今後も幅広い対応マシンの追加が続々と行われる予定です！←「あらゆる」

Fixstars Amplify の内容と特徴

- 開発環境：Amplify SDK
- 実行環境：Amplify Annealing Engine (AE)

開発環境：Fixstars Amplify SDK

Fixstars Amplify SDK ならアニーリングのプログラミングが圧倒的に短縮されます

通常のプログラミング

1. **課題を定式化**
   マシンのSDKやAPI仕様に合わせて物理モデルをデータ化
2. **論理モデルへ変換**
   目的関数をマシンの動作モデルで再定義
3. **物理モデルへ変換**
   マシン仕様や制約を考慮した物理モデルに再変換
4. **マシンにデータを入力**
   マシンのSDKやAPI仕様に合わせて物理モデルをデータ化
5. **マシンを実行**
   特定マシンのみで実行可能

Fixstars Amplifyを用いたプログラミング

1. **課題を定式化**
   定式化された数式をプログラムコードで表現
2. **マシンを実行**
   複数マシンの中から選択可能

SDKが提供するAPIが、自動で各マシンに合った形式へ多段変換をして入力。実行結果は逆変換をして、ユーザーにとって結果の解釈が容易な形式で返却されます。

開発環境インストール

```
$ pip install amplify
```

最適化コード例

```python
from amplify import VariableGenerator, FixstarsClient, solve

# 入力モデルの構築
q = VariableGenerator().array("Binary", 2)
f = 1 - q[0] * q[1]

# 実行マシンの設定
client = FixstarsClient()
client.parameters.timeout = 1000

# アニーリングの実行
result = solve(f, client)

# 結果の解釈
solution = q.evaluate(result.best.values)

print(f"result: {q} = {solution}")
# result: [q_0, q_1] = [1. 1.]
```

FIXSTARS Amplify

実行環境：Fixstars Amplify Annealing Engine (AE)

NVIDIA GPU V100/A100 で動作

- 独自の並列化シミュレーテッド
  アニーリングアルゴリズム

WEB経由で計算機能を提供

- 社会実装・PoC・検証が加速
- Amplify SDK の実装を直ぐに実行可能

商用マシンでは最大規模・最高速レベル

- 120,000 ビット（全結合）
- 260,000 ビット超（疎結合）

	標準マシン Fixstars Amplify AE	標準マシン D-Wave 2000Q/Advantage	標準マシン 東芝 SQBM+	日立 CMOS Annealing	富士通 Digital Annealer
装置型式	GPU	量子回路	GPU	デジタル回路	デジタル回路
最大ビット数	262,144以上	2,048 (16x16x8)/ 5,760 (16x15x24)	100,000 (SQBM+)/ 10,000 (SBM PoC 版)	61,952 (352x176)	8,192 (DA2)/ 100,000 (DA3)
係数パラメータ	デジタル (32/64bit)	アナログ (5bit程度)	デジタル (32bit)	デジタル (3bit)	デジタル (16/64 bit)
結合グラフ	全結合	キメラグラフ/ ペガサスグラフ	全結合	キンググラフ	全結合
全結合換算ビット数	131,072	64/124	31,000程度 (SQBM+)(*1)/ 1,000 (SBM PoC 版)	176	8,192 (DA2)/ 100,000 (DA3)
APIエンドポイント	Fixstars Amplify	D-Wave Leap	Fixstars Amplify / AWS	Annealing Cloud Web	DA Cloud

※評価・検証用途では無償提供

FIXSTARS Amplify

オンラインデモ & チュートリアル

Amplify デモ 検索

https://amplify.fixstars.com/ja/demo

デモアプリケーション

ピクロスパズルの求解

プログラミング難易度

複雑な定式化の例として、数字で与えられるヒントを元にマスを塗り、絵を完成させるパズルゲーム、ピクロスを解くアプリを開発します。

デモアプリ サンプルコード

チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度

複雑で未知な目的関数にも適用可能な、機械学習と組み合わせ最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超電導を実現する材料探索を取り扱います。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度

化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と化学反応に関する物理シミュレーションを用います。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度

流体機器設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚抗比を最大化するように翼型の探索を行います。

サンプルコード

デモアプリケーション

容量制約つき運搬経路問題 (CVRP)

プログラミング難易度

運送業における効率的な配送計画の策定やごみ収集や道路清掃における訪問順序の最適化等での応用が期待される容量制約つき運搬経路問題（CVRP）を取り扱います。

デモアプリ サンプルコード

チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度

ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するような信号機群の最適制御を実施します。最適化の実施及び実証には、マルチ・エージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

チュートリアル応用編

定式化による交通信号機の最適化

プログラミング難易度

都市における渋滞を最小化するために、刻一刻と変化する交通状況に応じ、組合せ最適化を用いてリアルタイムに信号機の最適制御を実施します。また、その様な信号機制御を実施した際の都市の交通量をシミュレーションします。

サンプルコード

チュートリアル応用編

10. 整数長ジョブスケジューリング問題

プログラミング難易度

あらかじめ決まった数のジョブとマシンがあり、それぞれのジョブにかかる時間が分かっているとします。それぞれのジョブをいずれかのマシンに割り当てます。ジョブスケジューリング問題では、最も早く全ジョブが完了するような割り当て方を求めます。

サンプルコード

チュートリアル基礎編

画像のノイズ除去

プログラミング難易度

画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード

チュートリアル応用編

会議室割当問題

プログラミング難易度

制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード

チュートリアル応用編

タクシーマッチング問題

プログラミング難易度

目的関数と制約条件を用いて定式化するアプリケーションの例としてタクシーマッチング問題のアプリケーションを開発します。

サンプルコード

デモアプリケーション

グラフ彩色問題

プログラミング難易度

Fixstars Amplifyによる、グラフ彩色問題の定式化を体験します。

デモアプリ サンプルコード

デモアプリケーション

巡回セールスマン問題

プログラミング難易度

Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ サンプルコード

デモアプリケーション

数独

プログラミング難易度

Fixstars Amplifyによる、数独の定式化を体験します。

デモアプリ サンプルコード

デモアプリケーション

ライドシェア

プログラミング難易度

集合型ライドシェアの最適化アプリケーションを体験します。

デモアプリ サンプルコード

デモアプリケーション

タスク割当問題

プログラミング難易度

店舗とタスクに従業員を割り当てる組合せ最適化問題のアプリケーションを体験します。

デモアプリ サンプルコード

デモアプリケーション

ポートフォリオ最適化

プログラミング難易度

リスクとリターンを考慮した株式ポートフォリオの最適化アプリケーションを体験します。

デモアプリ サンプルコード

様々な分野で利用が拡大しています

豊田通商株式会社 TOYOTA TSUSHO CORPORATION

NTT DATA

NRI 野村総合研究所 Nomura Research Institute

住友商事株式会社

IPA Better Life with IT

登録社・組織数: 約900

累計実行回数: 7,000万回超

Fusic Co.,Ltd.

慶應義塾 Keio University

早稲田大学

株式会社ネクスティ エレクトロニクス

豊田中央研究所

Fixstars Amplify ご利用プラン

料金のご紹介

https://amplify.fixstars.com/ja/pricing

	ベーシック 評価・検証用の無料プラン 使い始める	スタンダード 実用レベルの計算環境 見積りを依頼	プレミアム 高性能な計算環境 見積りを依頼	Sプレミアム 最高性能の計算環境 見積りを依頼
利用料金	無料	月額10万円（1名） （税込11万円） 月額20万円（最大5名） （税込22万円）	月額20万円（1名） （税込22万円） 月額60万円（最大5名） （税込66万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）
計算環境	スモール	ミディアム	ラージ	スーパーラージ
D-Waveマシンの無料実行	3分/月	3分/月	3分/月	3分/月
SQBM+オプション	無料	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）	月額30万円（1名） （税込33万円） 月額90万円（最大5名） （税込99万円）
サポート	ベーシック	スタンダード	プレミアム	プレミアム
評価・検証フェーズでの利用	✓	✓	✓	✓
実運用フェーズでの利用		✓	✓	✓

定式化や実装を
手厚く
支援します！

開発支援サービス(個別見積り)

コンサル・システム開発等
数百万円～数千万円

＋

月額利用料
百万円～

セミナー・トレーニングのご紹介

https://amplify.fixstars.com/ja/news/seminar

お客様の実際の課題解決をご支援するために、**無料セミナー**や**有償トレーニング**を提供しています。

無料セミナー・ワークショップ

ビジネス向け、エンジニア向けに分けて
開催しています！

ビジネス向け

製造業向け量子コンピュータ時代のDXセミナー

見える化、予測・分析、その先の最適化へ

組合せ最適化問題や量子アニーリング・イジングマシンの概要をご紹介したのち、製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化や生産ラインのシフト最適化などの事例とデモをご紹介いたします。「Fixstars Amplify」を通じて量子アニーリング・イジングマシンを活用することで、どのようなビジネス上の効果が期待できるのかを感じていただきたいと思います。

エンジニア向け

製造業向け量子コンピュータ時代のDXセミナー

最適化の中身を覗いてみよう

製造業における組合せ最適化を活用したDX推進の一例として、生産計画最適化、勤務シフト最適化などの事例を用いて、問題設定の考え方、目的関数や制約条件の定式化、実装のポイントなどを実際のコードを見ながら解説します。また、サンプルコードを用いて、ご自身の環境で実際に量子アニーリング・イジングマシンを動かす体験をしていただきます。

企業向けプライベートトレーニング

お客様が抱える実際の課題やデータを使った
カスタムメイド のトレーニングです！

全4回のレクチャーとお客様に実施いただく「課題」を含む約1.5か月のコースです。コースの前半では、量子アニーリング・イジングマシン専用の開発／実行環境であるFixstars Amplifyを用いてPython言語による組合せ最適化アプリケーション開発方法を学びます。後半では、お客様が抱える実際の課題やデータを使ったトレーニングを実施します。量子アニーリング・イジングマシンを使って実課題の解決に取り組んでみたい方に最適なコースです。

ワークショップ
事前準備（事前メールの内容）

ワークショップの事前準備（1）

- 【事前メールに記載】ご自身のPC (ブラウザ上) で Python プログラミングを行います。Google Colaboratory を使うので、事前にログイン出来ることを確認をお願いします（要 Google アカウント）

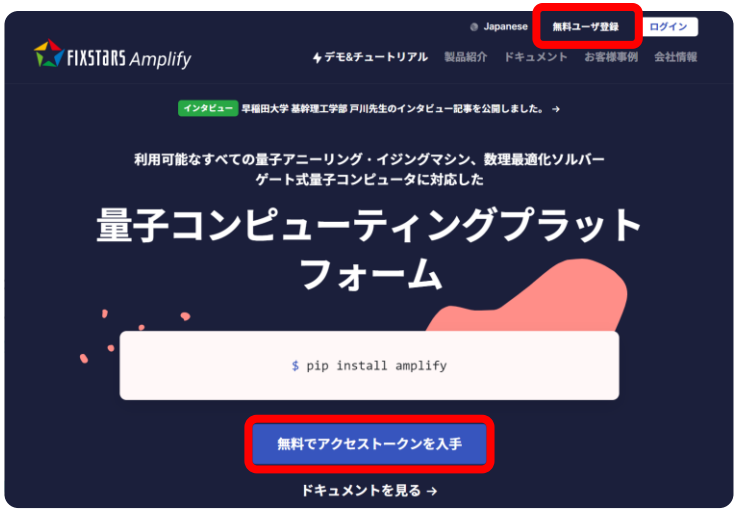
Google Colab 検索

https://colab.research.google.com/

- 【事前メールに記載】Fixstars Amplify ホームページより
  ユーザ登録の上、無料トークンの取得をお願いします
  （1分で終わります）

Fixstars Amplify 検索

https://amplify.fixstars.com/



質問は随時 Zoom の Q&A へお願いします

ワークショップの事前準備 (2)

【事前メールに記載】

- 取得されたトークンを用いて、トークンチェック用サンプルコードが動くか確認をお願いします。

  https://colab.research.google.com/drive/1bg2Ql3McJck_Sto8uvxtmPUMWtRFhf7a （※URLはZoomのチャット欄を参照）

- サンプルコードは閲覧のみ可能な状態です。「ファイル」→「ドライブにコピーを保存」の上、ご自身のトークンを入力してください。その後、Shift + Enterで実行下さい。

```
! pip install amplify

token = "AE/**************************" # ご自身のトークンを入力
```

- ご自身のトークン番号は、Amplifyウェブページ → よりご確認いただけます。

- 実行後、以下の結果が出力されればOKです。

```
result: [q_0, q_1] = [1. 1.] (f = 0.0)
```

ワークショップの事前準備 (3)

- 決定変数を増やしてみる

```
q = VariableGenerator().array("Binary", 10)
```

- 目的関数を変更してみる

```
f = 1 - q[0] * q[1] - 2 * q[0] + q[1]
f
```

```
import random

f = 1
for i in range(1, 10):
    f -= random.randint(-2, 2) * q[i] * q[i - 1]
f
```

```
from amplify import VariableGenerator, solve, Fixstar

# バイナリー変数q_0, q_1を定義
q = VariableGenerator().array("Binary", 2)
print(q)

# 目的関数 1 - q_0 * q_1 を定義
f = 1 - q[0] * q[1]

# クライアントの設定
client = FixstarsClient()  # Fixstars Amplify AE
client.parameters.timeout = 1000  # タイムアウト1秒
client.token = token

# マシンを実行
result = solve(f, client)  # 問題を入力してマシンを実行

# 解の取得
values = result.best.values  # 解を格納
q_solutions = q.evaluate(values)  # デコード

print(f"result: {q} = {q_solutions} (f = {result.best
```

FIXSTARS Amplify

ワークショップ

通常の組合せ最適化

（ブラックボックス最適化への導入）

サンプルコード

サンプルコードを開き、「ファイル」→「ドライブにコピー」の上、トークンを入力し実行して下さい。

- 数の分割サンプルコード
  https://colab.research.google.com/drive/1bl6BGPwnpd3N0AveoDMGb8egprEdQMqU
  （※URLはZoomのチャット欄を参照）

質問は随時 Zoom の Q&A へお願いします

数の分割問題（概要）

- 与えられた n 個の整数$a_0, \cdots, a_{n-1}$を二つの集合に分ける。
  集合内の数の和が、もう一方の集合内の数の和と等しくなるようできるか?
  - NP完全問題: とても難しい問題として知られている → 全通り試すしか方法は無い
  - 問題のバリエーション
    - 判定問題: 完全に等しく出来るか？または等しい組合せは何か？
    - 最適化問題: 完全に等しいか、または最も惜しい組合せは何か？

数の分割問題（具体例と解法の方針）

具体例

`{2,10,3,8,5,7,9,5,3,2}` の10個の数の完璧な分割は見つけられるか？

答え

- 存在する
  - `{2,3,5,7,10}` と `{2,3,5,8,9}`
  - どちらも和は 27
- 分割方法は 23 通り存在する (対称を除く)

どうやって解くか？

- ひとつの『数』がどちらの集合に分割されるか全通り試す → $2^{10} = 1024$通り
- 効率のよい厳密な方法は知られていない・・・(もし発見されたら大騒ぎ)

数の分割問題（定式化）

最適化問題：数の分割において最も惜しい組合せは何か？

- **目的関数**

  {集合1の和} – {集合2の和} の絶対値を最小化

- **決定変数**

  数 a_i がどちらの集合に属するかを s_i で表す

  - a_i = { 2,10, 3, 8, 5, 7, 9, 5, 3, 2}
  - s_i = {-1, 1,-1, 1,-1,-1, 1, 1, 1, 1}

数理モデル

- 目的関数

$$f = \left| \sum_{i=0}^{N-1} s_i a_i \right| \quad (s_i \in \{-1, +1\})$$

$\sum s_i a_i$ は、自然と
{『1』の集合の和} – {『-1』の集合の和}
となる！

数の分割問題（バイナリへの式変形）

- 0-1整数二次計画問題への変換
  - Quadratic Unconstrained Binary Optimization (QUBO) 式

$$f = \left| \sum_{i=0}^{N-1} s_i a_i \right| \quad (s_i \in \{-1, +1\})$$

$\sum s_i a_i$ は、自然と
{〖1〗の集合の和} – {〖-1〗の集合の和}
となる！

$$\rightarrow \left(\sum_{i=0}^{N-1} s_i a_i \right)^2 \quad (s_i \in \{-1, +1\})$$

絶対値を二次式で表す

$$\rightarrow \left(\sum_{i=0}^{N-1} (2q_i - 1) a_i \right)^2 \quad (q_i \in \{0, +1\})$$

±1をバイナリで表す

数の分割問題（定式化の具体例）

問題

- a_i ={2,10,3,8,5,7,9,5,3,2} の10個の数の完璧な分割は見つけられるか？

決定変数

- $q_i = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\} (q_i \in \{0,1\})$ で集合0又は集合1、どちらに所属するかを表す

目的関数

$$f = \left(\sum_{i=1}^{N} (2q_i - 1) a_i \right)^2$$

目的関数を展開

$$f = \begin{pmatrix} 2(2q_0 - 1) + 10(2q_1 - 1) + 3(2q_2 - 1) + 8(2q_3 - 1) + 5(2q_4 - 1) \\ +7(2q_5 - 1) + 9(2q_6 - 1) + 5(2q_7 - 1) + 3(2q_8 - 1) + 2(2q_9 - 1) \end{pmatrix}^2$$

数の分割問題（プログラムコード）

1. 定式化

- 問題の定義と決定変数生成器による決定変数の生成

```
a = [2, 10, 3, 8, 5, 7, 9, 5, 3, 2]
q = amplify.VariableGenerator().array("Binary", len(a))
```

- 目的関数、$f = \left(\sum_{i=1}^{N}(2q_i - 1)a_i\right)^2$、の定式化（①②③は同等）

①
```
f = ((2 * q - 1) * a).sum() ** 2
```

②
```
f = 0
for i in range(len(a)):
    f += (2 * q[i] - 1) * a[i]
f **= 2
```

③
```
f = amplify.sum((2 * q - 1) * a) ** 2
```

色々な書き方が出来る

2. 実行

```
result = amplify.solve(f, client)
```

3. 結果

```
q = [1, 1, 1, 0, 1, 1, 0, 0, 0, 0], f = 0.0, w = 27
```

得られた目的関数の値0

各集合の合計値27

各数字に対して、集合0か、集合1か

オンラインデモ & チュートリアル

Amplify デモ　検索

https://amplify.fixstars.com/ja/demo

デモアプリケーション

ピクロスパズルの求解

プログラミング難易度 ★★★★

複雑な定式化の例として、数字で与えられるヒントを元にマスを塗り、絵を完成させるパズルゲーム、ピクロスを解くアプリを開発します。

デモアプリ　サンプルコード

チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度 ★★★★

複雑で未知な目的関数にも適用可能な、機械学習と組み合わせ最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超電導を実現する材料探索を取り扱います。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度 ★★★★

化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と化学反応に関する物理シミュレーションを用います。

サンプルコード

チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★

流体機器設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚抗比を最大化するように翼型の探索を行います。

サンプルコード

デモアプリケーション

容量制約つき運搬経路問題 (CVRP)

プログラミング難易度 ★★★★

運送業における効率的な配送計画の策定やごみ収集や道路清掃における訪問順序の最適化等での応用が期待される容量制約つき運搬経路問題（CVRP）を取り扱います。

デモアプリ　サンプルコード

チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★

ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するような信号機群の最適制御を実施します。最適化の実施及び実証には、マルチ・エージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

チュートリアル応用編

定式化による交通信号機の最適化

プログラミング難易度 ★★★★

都市における渋滞を最小化するために、刻一刻と変化する交通状況に応じ、組合せ最適化を用いてリアルタイムに信号機の最適制御を実施します。また、その様な信号機制御を実施した際の都市の交通量をシミュレーションします。

サンプルコード

チュートリアル応用編

10. 整数長ジョブスケジューリング問題

プログラミング難易度 ★★★

あらかじめ決まった数のジョブとマシンがあり、それぞれのジョブにかかる時間が分かっているとします。それぞれのジョブをいずれかのマシンに割り当てます。ジョブスケジューリング問題では、最も早く全ジョブが完了するような割り当て方を求めます。

サンプルコード

チュートリアル基礎編

画像のノイズ除去

プログラミング難易度 ★★★

画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード

チュートリアル応用編

会議室割当問題

プログラミング難易度 ★★★

制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード

チュートリアル応用編

タクシーマッチング問題

プログラミング難易度 ★★★

目的関数と制約条件を用いて定式化するアプリケーションの例としてタクシーマッチング問題のアプリケーションを開発します。

サンプルコード

デモアプリケーション

グラフ彩色問題

プログラミング難易度 ★★★

Fixstars Amplifyによる、グラフ彩色問題の定式化を体験します。

デモアプリ　サンプルコード

デモアプリケーション

巡回セールスマン問題

プログラミング難易度 ★★★★

Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ　サンプルコード

デモアプリケーション

数独

プログラミング難易度 ★★★

Fixstars Amplifyによる、数独の定式化を体験します。

デモアプリ　サンプルコード

デモアプリケーション

ライドシェア

プログラミング難易度 ★★★★★

集合型ライドシェアの最適化アプリケーションを体験します。

デモアプリ　サンプルコード

デモアプリケーション

タスク割当問題

プログラミング難易度 ★★★★★

店舗とタスクに従業員を割り当てる組合せ最適化問題のアプリケーションを体験します。

デモアプリ　サンプルコード

デモアプリケーション

ポートフォリオ最適化

プログラミング難易度 ★★★★

リスクとリターンを考慮した株式ポートフォリオの最適化アプリケーションを体験します。

デモアプリ　サンプルコード

組合せ最適化と
ブラックボックス最適化

組合せ最適化とブラックボックス最適化

通常の組合せ最適化

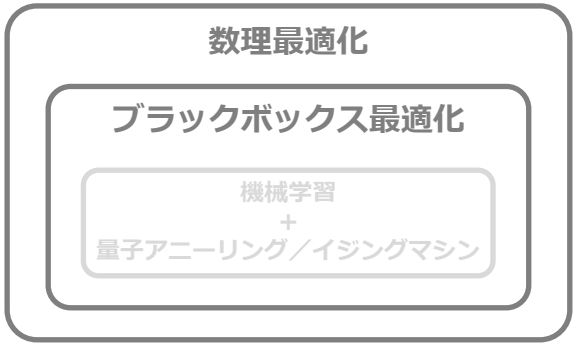
- 課題に対し、2次定式化 (QUBO) を実施
  - 数の分割
    $f = [\Sigma(2q_i - 1)a_i]^2$
  - 生産計画最適化
    $f = \Sigma\Sigma\left(\Sigma a_i p_i q_{m,i,t} + \Sigma\Sigma b_i s_i q_{m,i,t} q_{m,i-1,f}\right) \cdots$
  - TSP問題（お遍路巡り最短経路探索）
    $f = \Sigma\Sigma\Sigma d_{i,j} q_{n,i} q_{n+1,j} \cdots$
- QUBO式に対し、直接アニーリングを行う
  （量子アニーリング・イジングマシン）

ブラックボックス最適化

- 課題が複雑で直接の定式化が不可能。
  例）最も計測精度が高くなるセンサー群の配置は？
  ➔ 実験かシミュレーション（コスト大）
- シミュレーションや実験の試行回数の削減
  ➔ シミュレーションや実験が楽になる
- 逆問題に対するアプローチとしても適用化
  例）ある計測値を実現する実験条件を見つける

FMQAの紹介

ブラックボックス最適化



目的関数値 $f(x)$ のみ観測可能

- 目的関数の形状や勾配などは何も分からない。

目的関数の評価回数に限りがある

- 入力集合 Ω が有限集合であっても全検索できない。

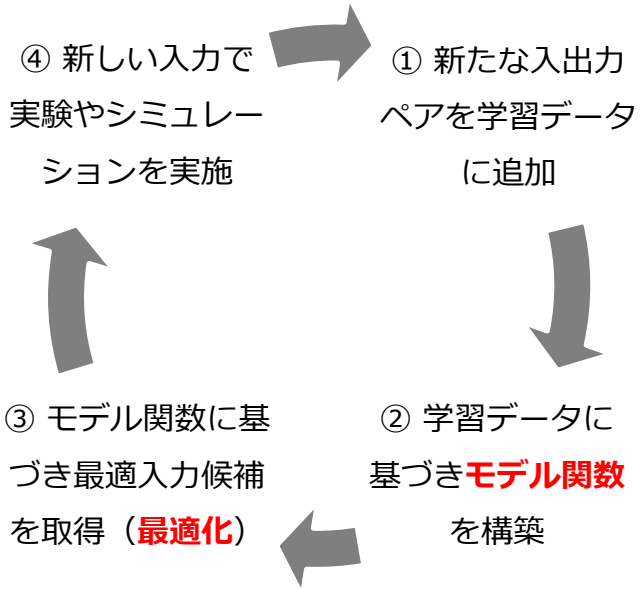
イメージ

- 5種類の材料から、いくつかの材料を選択・合成し、最も電気抵抗の小さな物質を作る。
  - 決定変数（材料の選択肢）x ・・・ 5桁の01ビットでどの材料を選択するかを記述

    $x = [1,0,0,0,0],\ [1,1,0,0,0],\ [1,0,1,0,0],\ \cdots,\ [1,1,1,1,1]$ → $(2^5 - 1)$ 通り
  - 目的関数 $f(x)$ = 合成物質の電気抵抗（実験又は数値シミュレーション）

ブラックボックス最適化のフロー

- 逐次最適化：最適化サイクルの実施



FMQA Kitai, et al., *Phys. Rev. Res.* (2020)

- **モデル関数** ➔ **FM**
- **最適化** ➔ **QA**

*Quadratic-optimization Annealing

FMQAの特徴：

- 活用のみ・探索なし
- 過学習の低減
- QA最適化

次元の呪いに強い！

モデル関数としての Factorization Machine (FM)

- モデル関数 $g(\boldsymbol{x})$ に機械学習モデルの一種である Factorization Machine (FM) を用いると、次のように変数 $\boldsymbol{x}$ に対する2次式での記述ができる。

$$g(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{v}) = w_0 + \langle \boldsymbol{w},\boldsymbol{x}\rangle + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \boldsymbol{v}_i,\boldsymbol{v}_j\rangle\, x_i\, x_j$$

$$= w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{if}\, x_i\right)^2 - \sum_{i=1}^{n} v_{if}^2\, x_i^2\right)$$

➔ **QUBO式**

- k はハイパーパラメータ、$\boldsymbol{w}$ 及び $\boldsymbol{v}$ は FM 学習後に取得される FM パラメータ。
- FM パラメータ数は k に依存。$k = n$ のときは QUBO の相互作用項と同じ自由度がある一方、k を小さくすることでパラメータ数を減らし過学習を抑制する効果
- このようなブラックボックス最適化手法を **FMQA** と呼ぶ場合がある。

K. Kitai, et al., Phys. Rev. Res. (2020).
T. Inoue, et al., Opt. Express (2022).

モデル関数としての Factorization Machine (FM)

ブラックボックス最適化の精度を向上する方法。
なぜQUBOの2次関数で複雑な現象を近似できるのか？

- モデル関数 $g(\boldsymbol{x})$ に機械学習モデルの一種
  に変数 $\boldsymbol{x}$ に対する2次式での記述ができる

$$g(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{v}) = w_0 + \langle \boldsymbol{w},\boldsymbol{x}\rangle + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \boldsymbol{v}_i,\boldsymbol{v}_j\rangle\, x_i\, x_j$$

$$= w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{if}\, x_i\right)^2 - \sum_{i=1}^{n} v_{if}^2\, x_i^2\right)$$

➔ **QUBO式**

- k はハイパーパラメータ、$\boldsymbol{w}$ 及び $\boldsymbol{v}$ は FM 学習後に取得される FM パラメータ。
- FM パラメータ数は k に依存。$k = n$ のときは QUBO の相互作用項と同じ自由度がある一方、k を小さくすることでパラメータ数を減らし過学習を抑制する効果
- このようなブラックボックス最適化手法を **FMQA** と呼ぶ場合がある。

K. Kitai, et al., Phys. Rev. Res. (2020).
T. Inoue, et al., Opt. Express (2022).

ブラックボックス最適化 活用例

材料分野に限らず、幅広い分野へ適用可能

FMQA: 活用例 (Amplify サンプルプログラム)

Amplify demo 検索

チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度 ★★★★☆

複雑で未知な目的関数にも適用可能な、機械学習と組み合わせ最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード

FMQA導入

チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★☆

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超電導を実現する材料探索を取り扱います。

サンプルコード

材料最適化

FMQA
×
物理モデル

チュートリアル応用編

ブラックボックス最適化 (3)

プログラミング難易度 ★★★★☆

化学プラントにおける生産量を最大化するための運転条件最適化を行います。最適化には、機械学習モデルに基づくブラックボックス最適化と化学反応に関する物理シミュレーションを用います。

サンプルコード

化学プラント運転条件最適化

FMQA
×
化学シミュレーション

チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★☆

流体機器設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚抗比を最大化するように翼型の探索を行います。

サンプルコード

翼形状最適化

FMQA
×
流体シミュレーション

チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★☆

ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するような信号機群の最適制御を実施します。最適化の実施及び実証には、マルチ・エージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

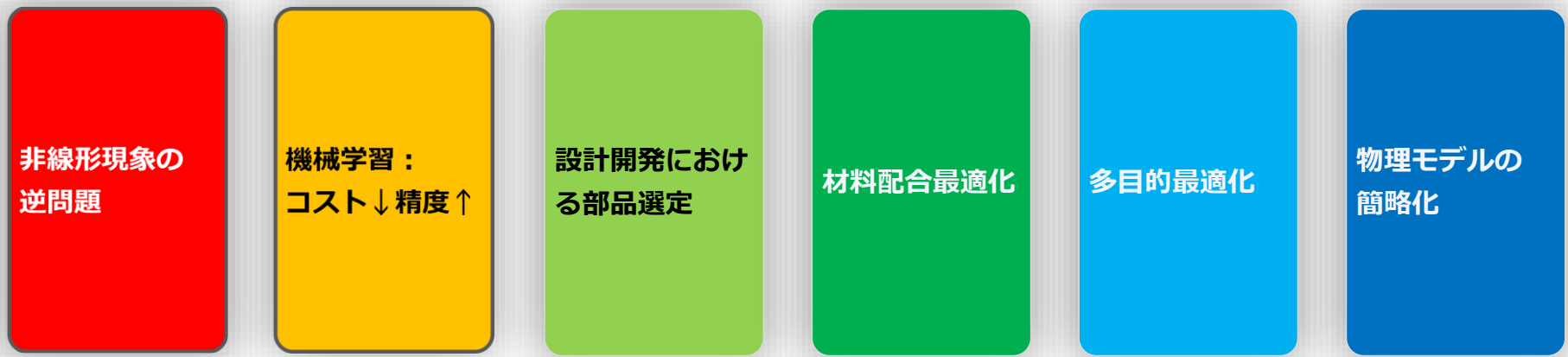
信号制御最適化

FMQA
×
マルチ・エージェント・シミュレーション

FIXSTARS Amplify

FMQA: 活用例 (Amplify ユーザー)

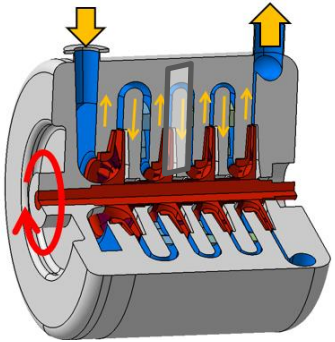
- **活用領域**
  - 化学、 創薬、 食品、 自動車、 電機、 通信、 重工、 エネルギー、 ・・・



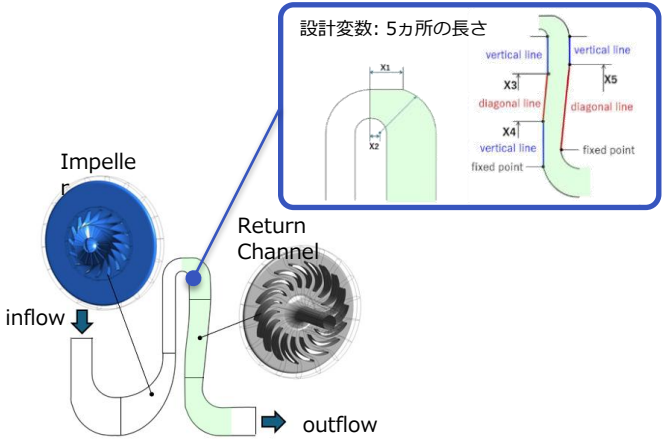
事例: ターボ機械の形状最適化（川崎重工業様）

- 従来より遺伝的アルゴリズム (GA) により形状最適化を行うことが多かった。規模が大きくなると求解までに時間がかかり、開発期間が長期化するといった課題があった
- FMQA により、従来手法と比べ、**同じ計算回数でもより優れた解が得られる**ことを確認。

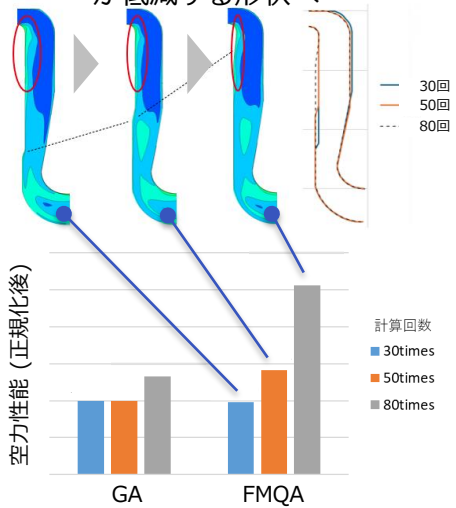
遠心圧縮機



流路形状の最適化
(圧縮機全体の空力性能の最大化)



最適化が進むごとに損失発生領域
が低減する形状へ



事例: 車両設計最適化（マツダ様）

Amplify マツダ　検索

- 車体設計の複数車種同時最適化問題。実数変数200以上の大規模問題。
- **車体の軽量化**と**共通部品数の最大化**の実現（**衝突性能・製造制約などの制約**を満たした上で）
- 国内外研究グループ*1,2 により様々な手法が試されてきており、**1～3万回程度の試行**により、ある程度良い解が見つけられることは確認されていた

- FMQA により、**1,000回程度の試行**で、従来手法と同等以上の解を見つけることに成功！

*1 進化計算コンペティション2017開催報告
*2 Multi-objective Bayesian optimization over high-dimensional search spaces

事例: 車両設計最適化（マツダ様）

Amplify マツダ　検索

複数の要件を満たすような多目的最適化の場合だとどのように
ブラックボックス最適化を適用するのか紹介してほしいです

- 車体設計の複数車種同時最適化問題。実
- **車体の軽量化**と**共通部品数の最大化**の実現（**衝突性能・製造制約などの制約**を満たした上で）
- 国内外研究グループ*1,2 により様々な手法が試されてきており、**1～3万回程度の試行**により、ある程度良い解が見つけられることは確認されていた

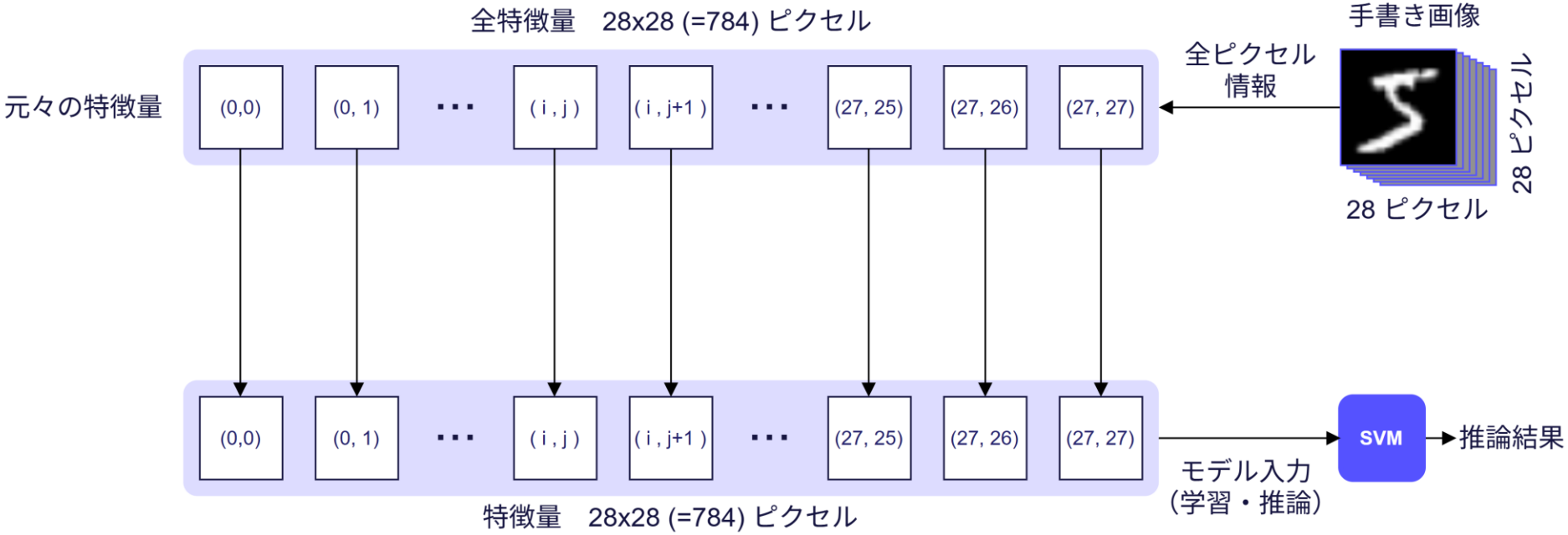
- FMQA により、**1,000回程度の試行**で、従来手法と同等以上の解を見つけることに成功！

*1 進化計算コンペティション2017開催報告
*2 Multi-objective Bayesian optimization over high-dimensional search spaces

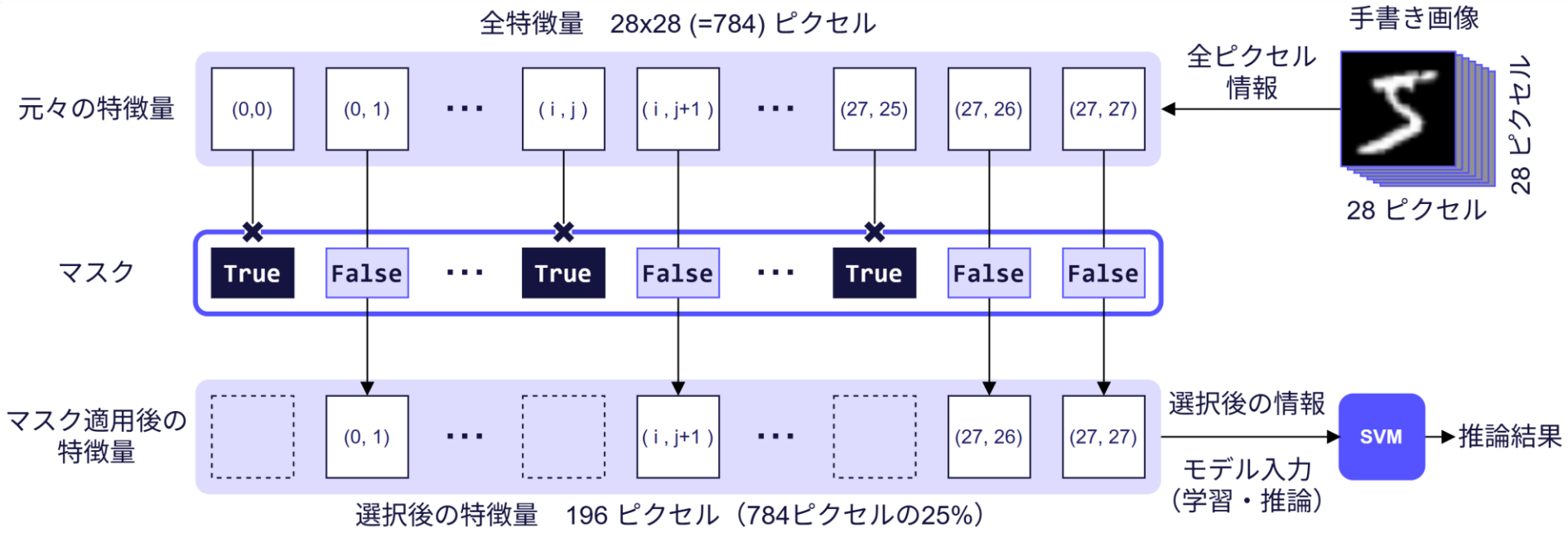
ブラックボックス最適化

1. 機械学習 (FM) ハンズオン
2. FMQA ハンズオン

SVMによるMNIST画像認識モデル



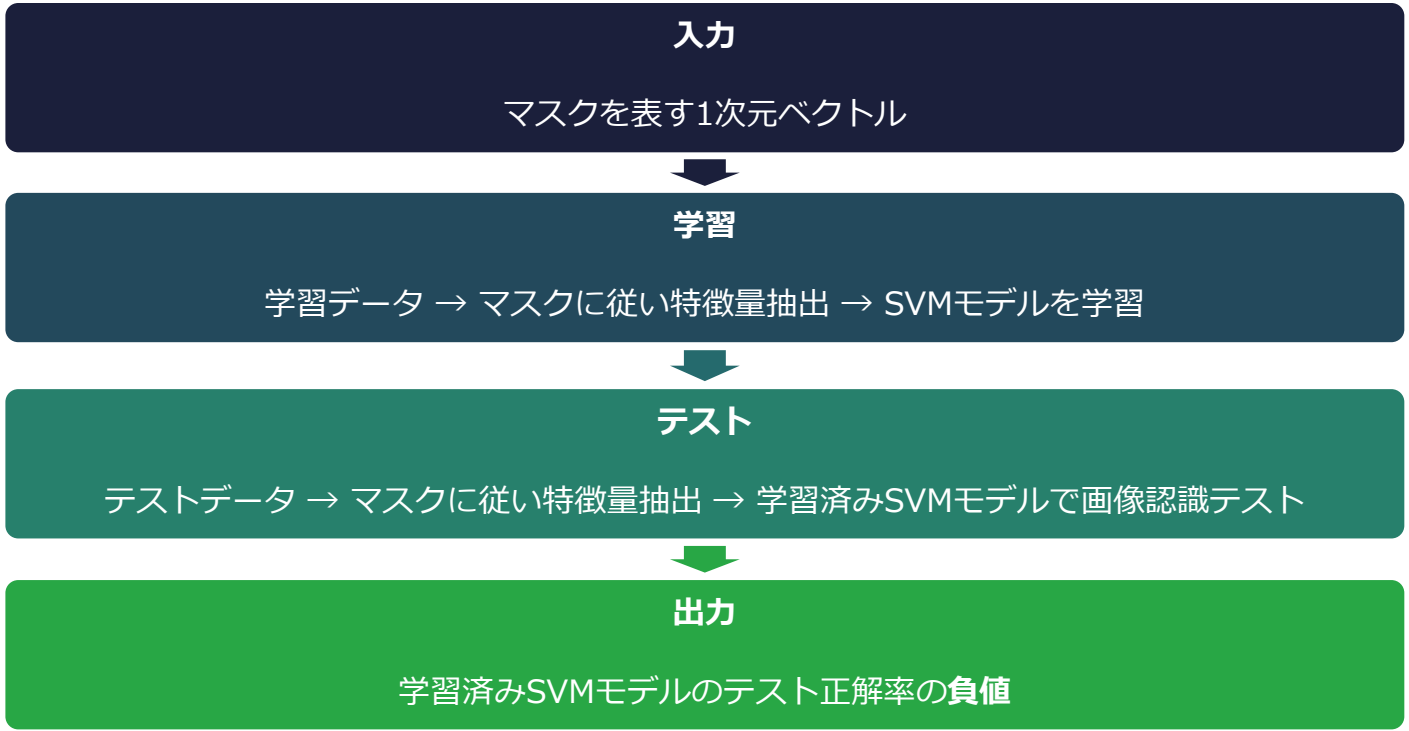
SVM画像認識モデルの最適特徴量選択



最適なマスクを探索し、軽く、正解率が高いSVMを目指したい！

画像認識モデルの特徴量選択・最適化フロー

- ブラックボックス目的関数



ブラックボックス最適化のデモプログラム：機械学習 (FM)

『マスク』を入力とし、『マスクに従って削減された特徴量に基づく SVM モデルの精度』を予測するFMモデル

https://colab.research.google.com/drive/1WmXIWVAQ-2MasS7XU9EE4rXXDb1loqnm

問題設定

マスクを $\boldsymbol{x}$ とすると、ブラックボックス関数 $f(\boldsymbol{x})$ は、

$f(\boldsymbol{x}) =$

最適化サイクル

1. 学習データからモデル関数 $g(\boldsymbol{x})$ を構築 (FM)
2. モデル関数 $g(\boldsymbol{x})$ が最小となる点 $\hat{\boldsymbol{x}}$ を推定 (QA)
3. 評価結果 $(\hat{\boldsymbol{x}}, f(\hat{\boldsymbol{x}}))$ を学習データに追加

↑のサイクルにより、最適化点近傍におけるFMの予測精度が向上し、組合せ最適化により、より良い $\hat{\boldsymbol{x}}$ の推定が期待される

ブラックボックス最適化のデモプログラム ― 機械学習 (FM)

『マスク』を入力とし、『マスクに従って削減された特徴量に基づく SVM モデルの精度』を予測するFMモデル

https://colab.research.google.com/drive/1WmXIWVAQ-2MasS7XU9EE4rXXDb1loqnm

問題設定

マスクを $\boldsymbol{x}$ とすると、ブラックボックス関数 $f(\boldsymbol{x})$ は、

$f(\boldsymbol{x}) =$

1. 学習データに基づき、（入力 $\boldsymbol{x}$ でマスクされない入力特徴量のみを考慮して）SVM 学習
2. テストデータに基づき、（入力 $\boldsymbol{x}$ でマスクされない入力特徴量のみを考慮して）学習済みSVMモデルをテスト
3. SVMモデルの評価結果（正答率の負値）を返却

最適化サイクル

1. 教師データから獲得関数 $g(\boldsymbol{x})$ を構築 (FM)
2. 獲得関数 $g(\boldsymbol{x})$ が最小となる点 $\hat{\boldsymbol{x}}$ を推定 (QA)
3. 評価結果 $(\hat{\boldsymbol{x}}, f(\hat{\boldsymbol{x}}))$ を教師データに追加

↑のサイクルにより、最適化点近傍におけるFMの予測精度が向上し、組合せ最適化により、より良い $\hat{\boldsymbol{x}}$ の推定が期待される

FMデモプログラム 1/6（SVMクラスの実装）

```python
PIX_SIZE = 28  # 画像1辺のピクセルサイズ

class MnistSvm:

    def __init__(
        self, mask_ratio: float, train_size=1000, test_size=1000, seed=0
    ) -> None:
        """与えられたマスク率に応じてクラスを初期化"""
        self._data, self._label = datasets.fetch_openml("mnist_784", version=1, return_X_y=True, parser="auto")

        ...

        self._num_features = len(self._data.columns)  # 特徴量の数
        self._num_masked_features = int(self._num_features * mask_ratio)  # マスクする特徴量の数

    def generate_random_mask(self) -> np.ndarray[bool, Any]:
        """マスク率を満たすマスクをランダムに生成"""
        mask = [True] * self.num_masked_features + [False] * (self._num_features - self.num_masked_features)
        self._rng.shuffle(mask)
        return np.array(mask)

    def train_eval(self, mask: np.ndarray[bool, Any]) -> float:
        """マスク適用後のデータに基づき機械学習を行い、学習後のモデル評価を実施、モデルの正答率を返却"""
        self._mask = mask
        train_data, test_data, train_label, test_label = self._fetch_masked_data()
        self._train(train_data, train_label)
        pred = self._eval(test_data, test_label)
        return pred
```

- `__init__`
  - 引数のマスク率に応じてクラスを初期化
  - `_num_features`：特徴量の総数（28x28=784個）
  - `_num_masked_features`：マスクする特徴量の数
- `generate_random_mask`
  - マスク率を満たすマスクをランダムに生成する関数
- `train_eval`
  - 与えられたマスクに基づく特徴量で学習を行い、得られたモデルを評価し、モデルの正答率を返却する関数。

FM デモプログラム 2/6（ブラックボックス関数の定義）

```python
# 問題サイズ（＝最適化対象のマスクの長さ）
problem_size = PIX_SIZE * PIX_SIZE


def make_blackbox_func(mnist_svm: MnistSvm) -> Callable[[np.ndarray], float]:

    def blackbox(x: np.ndarray) -> float:
        assert x.shape == (problem_size,)  # x は要素数 28*28 の一次元配列
        return -mnist_svm.train_eval(
            x
        )  # ブラックボックス関数として、モデルの正解率の負値を返却

    return blackbox
```

関数の利用例：

```python
# マスク率 75% を考慮する MnistSvm クラス
mnist_svm = MnistSvm(mask_ratio=0.75)

blackbox_func = make_blackbox_func(mnist_svm=mnist_svm)

# [True, ..., True, False, ... False] であるようなマスク率 75%のマスク
test_mask = np.array([True] * 588 + [False] * 196)

# 上記のマスクを与えた場合のSVMモデルの正解率（の負値）
print(f"[blackbox_func(test_mask) = ]")
```

- `make_blackbox_func`: ブラックボックス関数を生成する関数
  - `blackbox_func`: `MnistSvm.train_eval` から返却される正答率を(-1)倍して返却

    ➔つまり、ブラックボックス関数からの返却値は、学習済みSVMモデルのテストデータに対する正答率の負値

- マスク率75%を考慮する場合のブラックボックス関数
  - 特徴量588個をマスク、それ以外 (196個)をSVMで考慮

FM デモプログラム 3/6（FMの定義）

```python
class TorchFM(nn.Module):
    def __init__(self, d: int, k: int):
        """モデルを構築する

        Args:
            d (int): 入力ベクトルのサイズ
            k (int): パラメータ k
        """
        super().__init__()
        self.d = d
        self.v = nn.Parameter(torch.randn((d, k)))
        self.w = nn.Parameter(torch.randn((d,)))
        self.w0 = nn.Parameter(torch.randn(()))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """入力 x を受け取って y の推定値を出力する

        Args:
            x (torch.Tensor): (データ数 × d) の 2 次元 tensor

        Returns:
            torch.Tensor: y の推定値 の 1次元 tensor (サイズはデータ数)
        """
        out_linear = torch.matmul(x, self.w) + self.w0
        out_1 = torch.matmul(x, self.v).pow(2).sum(1)
        out_2 = torch.matmul(x.pow(2), self.v.pow(2)).sum(1)
        out_quadratic = 0.5 * (out_1 - out_2)

        out = out_linear + out_quadratic
        return out
```

- FM モデルを PyTorch で定義する

$$f(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{v}) = \underbrace{w_0 + \sum_{i=1}^{d} w_i x_i}_{\texttt{out\_linear}} + \underbrace{\frac{1}{2}\left[\underbrace{\sum_{f=1}^{k}\left(\sum_{i=1}^{d} v_{if}x_i\right)^2}_{\texttt{out\_1}} - \underbrace{\sum_{f=1}^{k}\sum_{i=1}^{d} v_{if}^2 x_i^2}_{\texttt{out\_2}}\right]}_{\texttt{out\_quadratic}}$$

FM デモプログラム 4/6（FMの学習）

```python
def train(x: np.ndarray, y: np.ndarray, model: TorchFM, plot_learning_curve=False):
    # イテレーション数
    epochs = 2000
    # モデルの最適化関数
    optimizer = torch.optim.AdamW([model.v, model.w, model.w0], lr=0.3)
    # 学習率スケジューラ
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=200, gamma=0.9)
    # 損失関数
    loss_func = nn.MSELoss()
    # データセットの用意
    x_tensor, y_tensor = (
        torch.from_numpy(x).float(),
        torch.from_numpy(y).float(),
    )
    dataset = TensorDataset(x_tensor, y_tensor)
    train_set, valid_set = random_split(dataset, [0.8, 0.2])
    batch_size = 8
    train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
    valid_loader = DataLoader(valid_set, batch_size=batch_size, shuffle=True)

    # 学習の実行
    min_loss = 1e18  # 損失関数の最小値を保存
    max_corrcoef = -1e18  # 相関係数の最大値を保存
    losses: list[float] = [0.0] * epochs
    corrcoefs: list[float] = [0.0] * epochs
    best_state = model.state_dict()  # 最も良いモデルのパラメータを保存するための変数
    for i in range(epochs):
        # 学習フェイズ
        for x_train, y_train in train_loader:
            optimizer.zero_grad()
            pred_y = model(x_train)
            loss = loss_func(pred_y, y_train)
            loss.backward()
```

FM 学習は、通常の機械学習と同様に進める。教師データを学習・検証データに分割し、ミニバッチ学習。

- **x, y**: 教師データ
- **model**: FM モデル（TorchFM）
- **epochs**: エポック（繰り返し）の数
- **lr**: （初期）学習率
- **scheduler**: 学習率スケジューラー

学習後、最も良いモデルに対し、次のような評価を実施

```python
# 教師データの入力値に基づきモデルを評価
y_pred = model(x_tensor)
print(f"corrcoef: {torch.corrcoef(torch.stack([y_tensor, y_pred]))[0
print(f"RMS error: {min_loss:.2f}")
```

FM デモプログラム 5/6（教師データ作成）

```python
def init_training_data(n0: int, mnist_svm: MnistSvm, blackbox_func: Callable[[np.ndarray], float]):
    """n0 組の初期教師データを作成する"""
    assert n0 < 2**problem_size

    # n0 個のマスクを乱数を用いて作成
    x = np.array([mnist_svm.generate_random_mask() for _ in range(n0)])

    # マスクの重複が発生していたらランダムに値を変更して回避する
    x = np.unique(x, axis=0)
    while x.shape[0] != n0:
        x = np.vstack((x, mnist_svm.generate_random_mask()))
        x = np.unique(x, axis=0)

    # blackbox 関数を評価して入力マスクに対応する n0 個の出力を得る
    y = np.zeros(n0) + 1e10
    for i in range(n0):
        start = time.perf_counter()
        y[i] = blackbox_func(x[i])
        print(
            f"Random process {i}: found y={y[i]:.4f}, current best={np.min(y):.4f}, "
            f"cycle elapsed time={time.perf_counter()-start:.2f}s"
        )
    return x, y
```

教師データを乱数により生成

- **mnist_svm**：MnistSvmクラスインスタンス
- **n0**：初期教師データのサンプル数
- **blackbox**：ブラックボックス関数（実験又はシミュレーション） $f(\boldsymbol{x})$

FM デモプログラム 6/6（メイン部分）

```
n0 = 20  # 初期教師データの数
x, y = init_training_data(n0, mnist_svm, blackbox_func)

# 機械学習モデルの作成
model = TorchFM(problem_size, k=20)
train(x, y, model, plot_learning_curve=True)
```

実際にサンプルプログラムを実行してみましょう。デフォルトの条件から、

- FMのハイパーパラメータ (k)
- エポック数 (epochs)

を変更した場合、真値と予測値の相関係数及びRMS誤差はどのように変化するでしょうか？

ブラックボックス最適化

1. 機械学習 (FM) ハンズオン
2. FMQA ハンズオン

ブラックボックス最適化のデモプログラム：FMQA

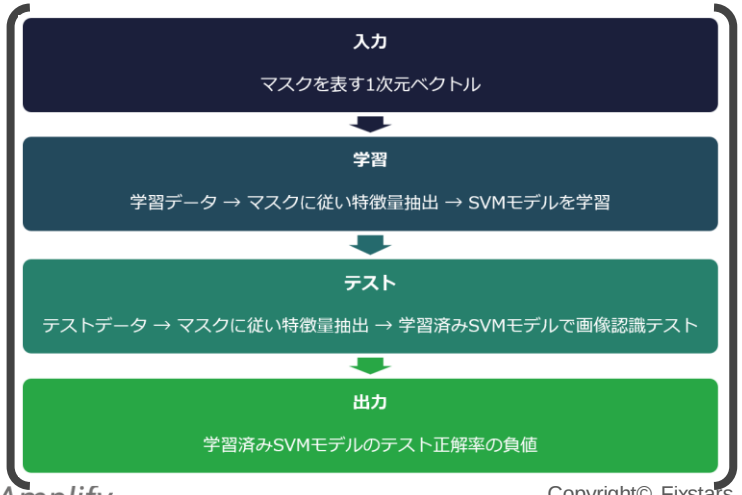
FMQAによる画像分類モデルのための最適特徴量選択

https://colab.research.google.com/drive/12vElYQoIgOJJhDucoUqAs8ktJzolnUK7

問題設定

マスクを $\boldsymbol{x}$ とすると、ブラックボックス関数 $f(\boldsymbol{x})$ は、

$f(\boldsymbol{x}) =$



最適化サイクル

1. 学習データからモデル関数 $g(\boldsymbol{x})$ を構築 (FM)
2. モデル関数 $g(\boldsymbol{x})$ が最小となる点 $\hat{\boldsymbol{x}}$ を推定 (QA)
3. 評価結果 $(\hat{\boldsymbol{x}}, f(\hat{\boldsymbol{x}}))$ を学習データに追加

↑のサイクルにより、最適化点近傍におけるFMの予測精度が向上し、組合せ最適化により、より良い $\hat{\boldsymbol{x}}$ の推定が期待される

FMQA デモプログラム 1/3 （アニーリング部分）

```python
# ソルバークライアントを Amplify AE に設定
client = FixstarsClient()
client.parameters.timeout = timedelta(milliseconds=5000)
client.token = token  # ローカル環境等で実行する場合はコメントを外して Amplify AEのアクセストークンを

def anneal(
    torch_model: TorchFM,
    constraint: Callable[[PolyArray], Constraint],
) -> np.ndarray:
    """受け取った FM モデルの最小値を与える x を求める"""

    # 長さ d のバイナリ変数の配列を作成
    gen = VariableGenerator()
    x = gen.array("Binary", torch_model.d)

    # TorchFM からパラメータ v, w, w0 を取得
    v, w, w0 = torch_model.get_parameters()

    # 目的関数を作成
    out_linear = w0 + (x * w).sum()
    out_1 = ((x[:, np.newaxis] * v).sum(axis=0) ** 2).sum()  # type: ignore
    out_2 = ((x[:, np.newaxis] * v) ** 2).sum()
    objective: Poly = out_linear + (out_1 - out_2) / 2

    # 組合せ最適化モデルを構築
    amplify_model = Model(objective, constraint(x))

    # 最小化を実行（構築したモデルと、始めに作ったソルバークライアントを引数として渡す）
    result = solve(amplify_model, client)
    if len(result.solutions) == 0:
        raise RuntimeError(f"No solution was found.")

    # モデルを最小化する入力ベクトルを返却
    return x.evaluate(result.best.values).astype(int)
```

学習済みFMモデルに基づき $\hat{\boldsymbol{x}}$ を推定する関数

- 決定変数配列の作成
- 学習済みFMモデルからパラメータ（重み・バイアス）を取得
- パラメータに基づき目的関数のサロゲートモデル $g(\boldsymbol{x})$ を作成
- サロゲートモデルを制約条件（後述）と合わせて `amplify.Model`を作成
  - 単に、`objective + constraint(x)` でも良い
- `amplify.solve` によりアニーリングを実行
- このサイクルにおける $\hat{\boldsymbol{x}}$ を返却

FMQA デモプログラム 2/3 （制約条件の設定）

```python
def get_constraint(q: PolyArray) -> Constraint:
    """
    マスクされる特徴量は mnist_svm.num_masked_features 個という制約条件を
    返却する関数。制約条件の重みとして 2 を選択。制約条件の重みについては、
    https://amplify.fixstars.com/ja/docs/amplify/v1/constraint.html#id8 を参照。
    """
    return 2 * equal_to(q.sum(), mnist_svm.num_masked_features)
```

- `get_constraint(q)`

  最適マスクを決定する決定変数 $\boldsymbol{x}$ に基づく以下の制約条件を返却する関数

  制約条件

  - $\sum_0^n x_i$ = `mnist_svm.num_masked_features`

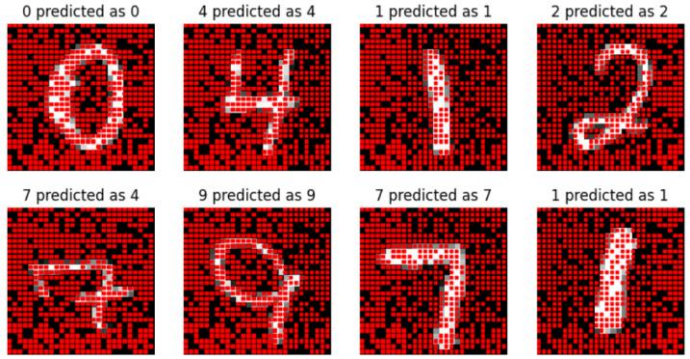
FMQA デモプログラム 3/3（メイン部分）

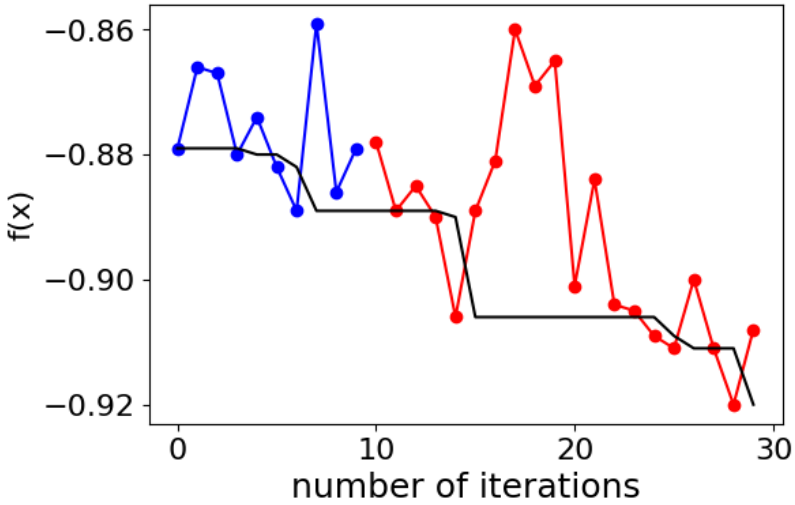
```
# N 回のイテレーションを実行
for i in range(n):
    print(f"FMQA cycle {i}")
    # 機械学習モデルの作成
    model = TorchFM(problem_size, k=20)
    corrcoef = train(x, y, model)
    print(f"- corrcoef: {corrcoef:.2f}")

    # 学習済みモデルの最小値を与える入力ベクトルの値を取得
    x_hat = anneal(model, get_constraint)
    # x_hat が重複する場合、それに近い解を乱数に基づき生成する
    while (x_hat == x).all(axis=1).any():
        idx_0 = rng.choice(np.arange(problem_size))
        idx_1 = rng.choice(np.arange(problem_size))
        x_hat[idx_0], x_hat[idx_1] = x_hat[idx_1], x_hat[idx_0]

    # 推定された入力ベクトルを用いてブラックボックス関数を評価
    y_hat = blackbox_func(x_hat)

    # 評価した値をデータセットに追加
    x = np.vstack((x, x_hat))
    y = np.append(y, y_hat)
    print(f"- found y = {y_hat:.4f}, current best = {np.min(y):.4f}")
```

1. `train()` による FM モデルの学習
2. `anneal()` によるアニーリング
   - FM モデル値を最小化するような入力値（マスク）を探索
3. 得られた入力値（マスク）が既存の場合の処理
4. 得られた入力値（マスク）によるブラックボックス関数の評価
   - マスク適用後の学習データを使ってSVMモデル学習
   - マスク適用後のテストデータを使ってSVMモデル評価
   - 正答率の負値を返却
5. 新しい入出力ペアをデータセットに追加

デモプログラムの実行

- 全特徴量 (784個) を考慮した場合のモデル評価
  - `accuracy_score=0.905`
- FMQA の基本条件
  - 初期データ個数：10個
  - FMQAの最適化サイクル数：20回
  - 特徴量：784個→196個（75%減）
- 実行時間
  - およそ5分





（上）最終的に得られたマスクをいくつかの手書き画像に重ねた例と（下）最適化の履歴

FMQA デモプログラムの実務での活用方法

```python
def make_blackbox_func(mnist_svm: MnistSvm) -> Callable[[np.ndarray], float]:

    def blackbox(x: np.ndarray) -> float:
        assert x.shape == (problem_size,)  # x は要素数 28*28 の一次元配列
        return -mnist_svm.train_eval(
            x
        )  # ブラックボックス関数として、モデルの正解率の負値を返却

    return blackbox
```

```python
def make_blackbox_func(d: int) -> Callable[[np.ndarray], float]:
    # 必要に応じて最適化対象以外の条件設定や初期設定などをここで実施

    def blackbox(x: np.ndarray) -> float:
        # x の組合せを元に実験を実施
        # または、シミュレーションを実施
        # 実験・シミュレーション結果を後処理し、最小化したい目的関数値を取得
        objective = hogehoge
        return objective  # type: ignore

    return blackbox
```

基本的に、`blackbox()` を変更する。必要に応じて、現在の教師データの出力などを追加。

- 例①：`blackbox()` 内で特徴量選択ではなく、枝刈りやデータ選択、ハイパラチューニング等を実施し、学習済みモデルの評価結果が最適になる様に手法を調整
- 例②：`blackbox()` 内でシミュレーションを呼び出し、後処理、その戻り値を最小化するように最適化。
- 例③：`blackbox()` 内で実験を行う。つまり、1回のFMQA で推定された探索候補 $\hat{\boldsymbol{x}}$ を対象に実験し、結果を教師データに追加、次のFMQA 試行を行う。
- $h(\boldsymbol{x})$ を最大にするような入力 $\boldsymbol{x}$ を推定する場合は、$1/h(\boldsymbol{x})$ や $-h(\boldsymbol{x})$ などを目的関数 $f(\boldsymbol{x})$ とする。

今後について

ぜひ、デモ・チュートリアルにあるサンプルコードにも挑戦してみてください！

一般的な組合せ最適化問題

目的関数のみで定式化

チュートリアル基礎編

画像のノイズ除去

プログラミング難易度 ★★☆☆☆

画像のノイズ除去を行うアプリケーションを開発します。

サンプルコード

制約条件のみで定式化

チュートリアル応用編

会議室割当問題

プログラミング難易度 ★★★☆☆

制約条件を用いて定式化するアプリケーションの例として会議室割当問題のアプリケーションを開発します。

サンプルコード

目的関数 + 制約条件

デモアプリケーション

巡回セールスマン問題

プログラミング難易度 ★★★★☆

Fixstars Amplifyによる、巡回セールスマン問題の定式化を体験します。

デモアプリ　サンプルコード

デモアプリケーション

容量制約つき運搬経路問題 (CVRP)

プログラミング難易度 ★★★★☆

運送業における効率的な配送計画の策定やごみ収集や道路清掃における訪問順序の最適化等での応用が期待される容量制約つき運搬経路問題（CVRP）を取り扱います。

デモアプリ　サンプルコード

ブラックボックス最適化問題

概要

チュートリアル応用編

ブラックボックス最適化 (1)

プログラミング難易度 ★★★★☆

複雑で未知な目的関数にも適用可能な、機械学習と組み合わせ最適化を組み合わせたブラックボックス最適化手法を紹介し、Amplifyを用いて実装します。

サンプルコード

材料探索

チュートリアル応用編

ブラックボックス最適化 (2)

プログラミング難易度 ★★★★☆

機械学習と量子アニーリング・イジングマシンを活用するブラックボックス最適化の適用例として、疑似的な高温超電導を実現する材料探索を取り扱います。

サンプルコード

翼型最適化

チュートリアル応用編

ブラックボックス最適化 (4)

プログラミング難易度 ★★★★☆

流体機器設計に不可欠な翼型の最適化問題を取り上げます。最適化には、組み合わせ最適化及び機械学習に基づくブラックボックス最適化と流体シミュレーションを用い、翼の揚抗比を最大化するように翼型の探索を行います。

サンプルコード

信号機制御

チュートリアル応用編

ブラックボックス最適化 (5)

プログラミング難易度 ★★★★☆

ブラックボックス最適化により、商業施設による交通集中が発生し得る都市における、交通渋滞を低減するような信号機群の最適制御を実施します。最適化の実施及び実証には、マルチ・エージェント・シミュレーションによる交通シミュレーションを用います。

サンプルコード

困った時はドキュメンテーションを！

https://amplify.fixstars.com/docs/amplify/v1/index.html

今後のセミナーの予定

今後も定期的に無料セミナーを開催します！

2025/4/24
「ブラックボックス最適化（機械学習）」

- はじめに
- 会社紹介
- Fixstars Amplifyの紹介
- ブラックボックス最適化のワークショップ
- Wrap Up
  - 事例のご紹介
  - 今後の進め方
  - Q&A

2025/5/14
「経路最適化」
(AnnealingEngine)

- はじめに
- 会社紹介
- Fixstars Amplifyのご紹介
- 経路最適化のワークショップ
- Wrap Up
  - 事例のご紹介
  - 今後の進め方
  - Q&A

2026/6/19（予定）
「BBO技術解説」
(AnnealingEngine)

- はじめに
- 会社紹介
- Fixstars Amplifyのご紹介
- BBO技術解説
- Wrap Up
  - 事例のご紹介
  - 今後の進め方
  - Q&A

ご質問・ご不明点がありましたら、お問い合わせフォームでご連絡下さい
https://amplify.fixstars.com/ja/contact

FIXSTARS Amplify

Q&A