大学共同利用機関法人 情報・システム研究機構
国立情報学研究所
National Institute of Informatics

AI時代における品質保証のチャレンジ ～ 機械学習の難しさと（AIによる）テスティング

国立情報学研究所　石川　冬樹
f-ishikawa@nii.ac.jp / @fyufyu
http://research.nii.ac.jp/~f-ishikawa/

ESTIC 2018　2018/07/20

自己紹介

- 国立情報学研究所　准教授
  - ソフトウェア工学および，先端自律・スマートシステムに関する研究・教育
- 電気通信大学　客員准教授
  - 社会人博士学生の指導（在学中も含め計9名）
- 産業界向け教育
  - トップエスイー（形式手法，クラウド等）
  - 日科技連SQiP研究会（要求と仕様のエンジニアリング）
  - 電通大ウェブシステムデザイン（クラウド）

興味：要求や仕様、設計に関する様々なモデルを活用した検証・推論・最適化・自動テスト生成・自己適応など

最近の主な活動

- ERATO-MMSD Project （グループ３リーダー）
  - 物理世界や確率などの連続値や微分方程式を含み複雑なシステムの検証・品質保証（主に車）
  - 形式検証・テスト自動生成・最適化・機械学習などの技術を適宜活用・併用する実用的検証手法の追求
- 機械学習（や広くAI）を含むシステムの品質保証
  - コミュニティ活動，概念や技術の整理
  - QAML Project: 多分野連携による検証・品質保証の研究
- 形式手法における工学的支援・応用
  - RefEng Project: 形式システムモデルの段階的詳細化支援
  - BigClouT Project: IoTにおける自己適応機構

AI・機械学習に関するコミュニティ活動

- 日本ソフトウェア科学会 機械学習工学研究会 主査
  - 2018年4月から（3月までは「有志一同」での活動）
  - 各イベント議論結果や講演資料等公開中
    （5月シンポジウム，7月ワークショップ等）

  https://sites.google.com/view/sig-mlse/

- AIプロダクト品質保証コンソーシアム
  副運営委員長
  - ガイドラインや保証レベルの策定など
  - 2018年4月活動開始

  http://www.qa4ai.jp/

今日の話題

- 「AI」はやめましょうか・・・
  - 技術を話すので
    「どういう特徴・特性」があって，
    「どういう対応アプローチ」を考えていくか？
    に焦点を

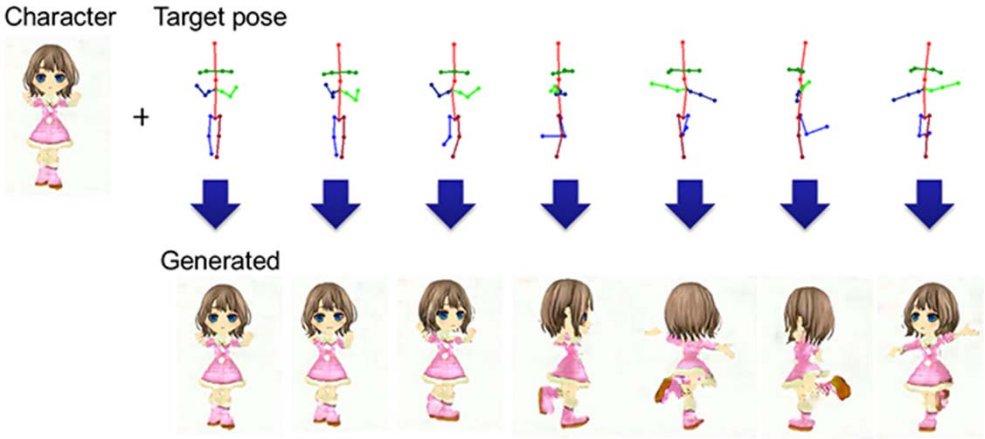
➡ 機械学習 × テスト・品質保証
特に自動車・自動運転関連において

機械学習ヤバい

検索します

技術の進化（ごくごく一例）

- DeNAさん（2018年5月）
  - 画像に対し指定した姿勢の画像を生成可能



画像元：[https://dena.com/intl/anime-generation/images/anime2.png]

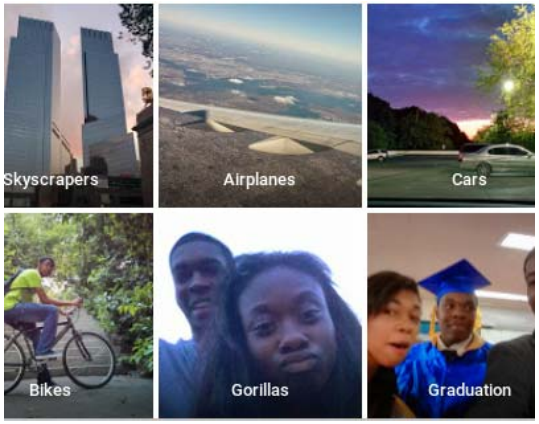
  - 動画まで生成可能で，「始点」から「終点」へ連続的に「変身」させることも（服を変えるなど）

ところで皆さんなら何をどう保証（テスト）しますか？

[https://dena.com/intl/anime-generation/]

社会的影響が大きい例・技術的限界の例

- Googleフォトの画像認識
  - 被写体の自動タグ付け機能
  - 黒人を「ゴリラ」とタグ付け
  - 2年経って本質的には直せていない
    （ゴリラを禁止ワード扱い
    にして対策）



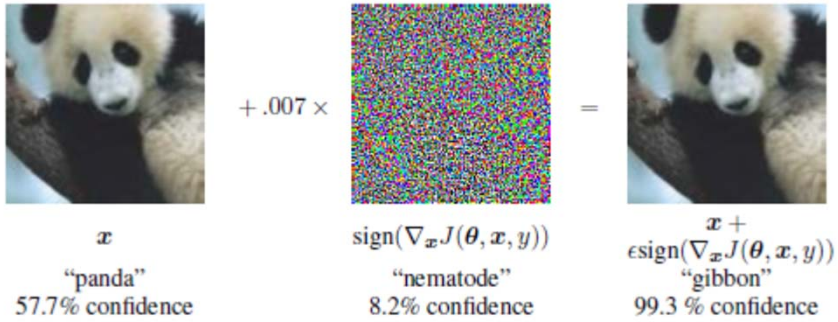
[画像元： https://twitter.com/jackyalcine/status/615329515909156865]

[https://www.theguardian.com/technology/2015/jul/01/google-sorry-racist-auto-tag-photo-app]

[https://www.theguardian.com/technology/2018/jan/12/google-racism-ban-gorilla-black-people]

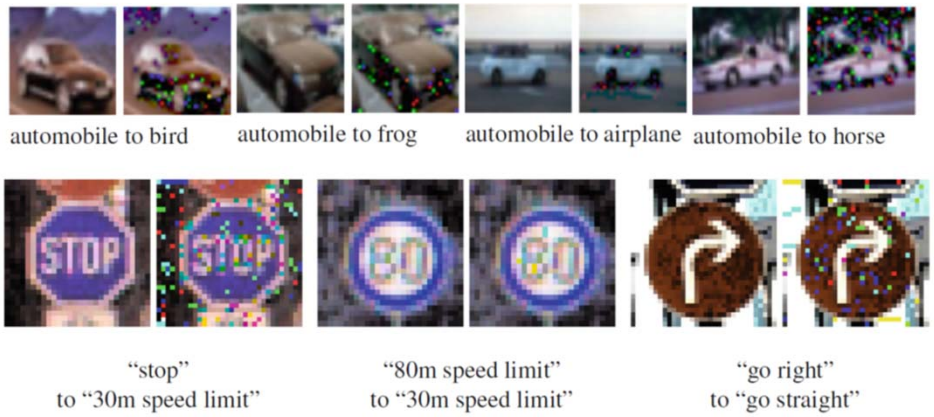
よく知られた課題（の一つ）

- 優れた画像識別器に存在するAdversarial Examples

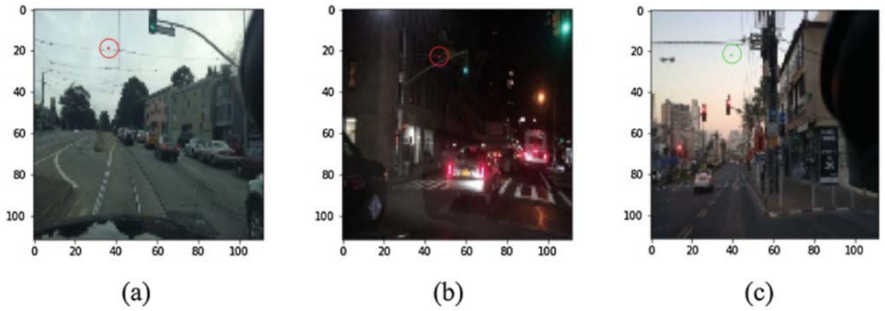


有名な例
「パンダ」が「テナガザル」に

[画像元： Goodfellow et al., Explaining and Harnessing Adversarial Examples, 2015]



[画像元： Huang et al., Safety Verification of Deep Neural Networks, 2017]



1ピクセルの変化で信号色の誤認識発生

[画像元： Wicker et al., Feature-Guided Black-Box Safety Testing of Deep Neural Networks, 2018]

継続的学習に関する例

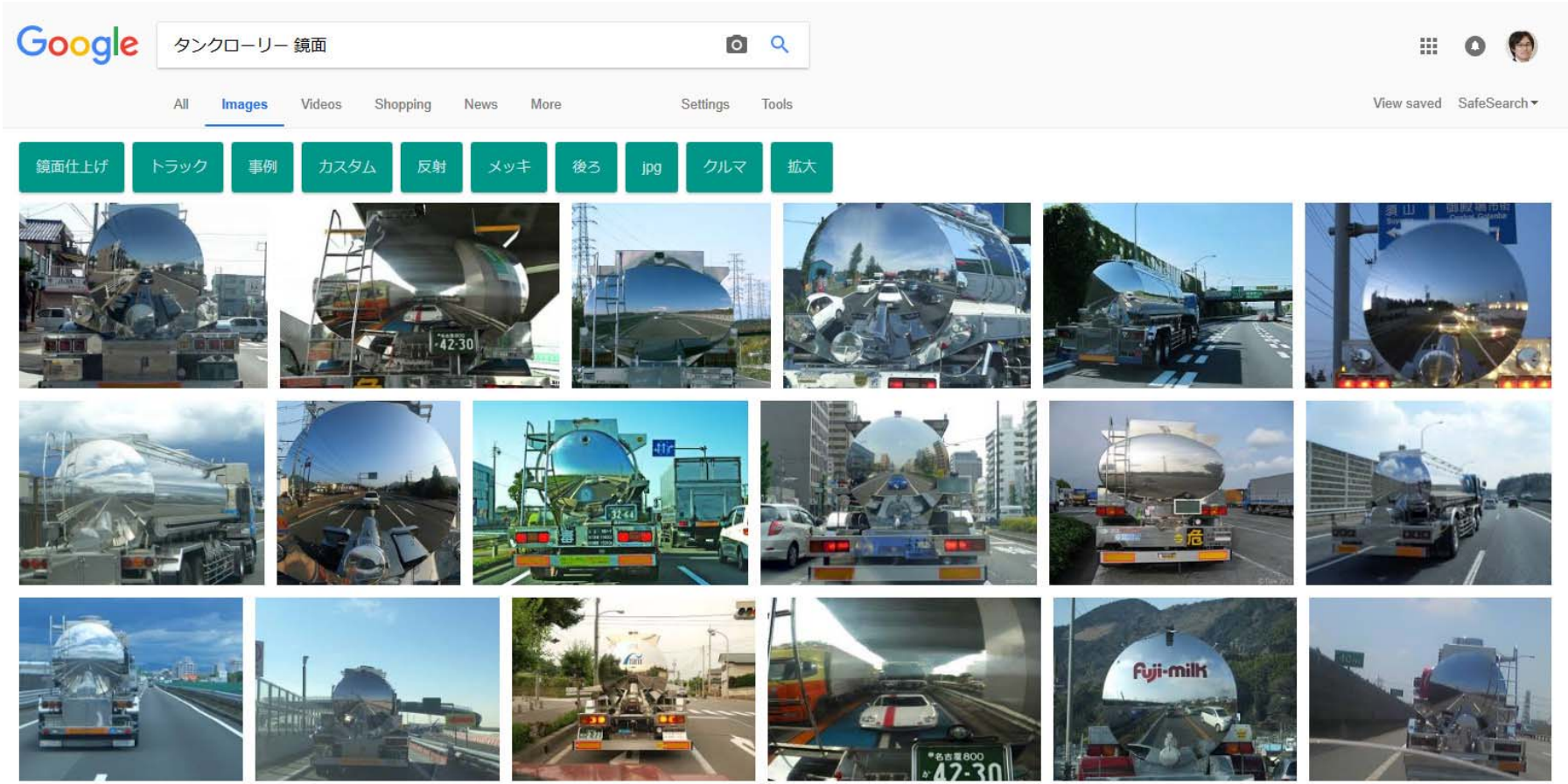
- Twitter Botによる（ユーザを真似た結果の）不適切発言

If you guessed, “It will probably become really racist,” you’ve clearly spent time on the Internet. Less than 24 hours after the bot, @TayandYou, went online Wednesday, Microsoft halted posting from the account and deleted several of its most obscene statements.

The bot, developed by Microsoft’s technology and research and Bing teams, got major assistance in being offensive from users who egged it on. It disputed the existence of the Holocaust, referred to women and minorities with unpublishable words and advocated genocide. Several of the tweets were sent after users commanded the bot to repeat their own statements, and the bot dutifully obliged.

[画像元： https://www.nytimes.com/2016/03/25/technology/microsoft-created-a-twitter-bot-to-learn-from-users-it-quickly-became-a-racist-jerk.html]

「いろいろな」想定環境に関する例



[画像元： https://www.google.com/ （「タンクローリー 鏡面」で検索・7/8）]

何を考えますか？

- これらすべて「バグ」か？潰さなければならないのか？
- そもそも潰せるものなのか？それとも技術的限界？
- どうやってこれらに気づくのか？他に対処すべき「同種」や「類似」の状況は列挙できるのか？どうやって？
- そもそも何を持って「品質保証」「完成」とするのか？「仕様」は何なのか？事前に見積・合意できるのか？
- 修正内容をどう決めてどれだけの頻度で更新するのか？
- 顧客やユーザには何をどうやって説明するのか？
- こんな挙動の原因を把握したり，予測し踏まえてテストしたりできるのか？どうやって？開発者ならわかるものなのか？外部品質保証者は何ができるのか？
- ・・・

さらに・・・保守もこれまでよりさらに難しい

- 従来のソフトウェアとはだいぶ性質が異なる
  「技術的負債」の存在

Googleでの経験から14個の負債を紹介
けっこう刺激的なタイトル

[Sculley et al., Machine Learning: The High-Interest Credit Card of Technical Debt, 2014]

品質の観点・指標やその担保施策が大きく異なる

- CASE： Changing Anything Changes Everything
- （コードではなく）データに対する依存性分析や保守・管理などのエンジニアリング
- 統計のパラドックス，フィードバックループの影響など固有の留意点

➡ *「何とかVer. 1.0リリース」は借金の始まり・・・*

本質的な違い：
振る舞いの帰納的な決定

参考：演繹と帰納

- 演繹
  - 諸前提から論理の規則にしたがって必然的に結論を導き出すこと。普通、一般的原理から特殊な原理や事実を導くことをいう。

- 帰納
  - 個々の特殊な事実や命題の集まりからそこに共通する性質や関係を取り出し、一般的な命題や法則を導き出すこと。

[大辞林（三省堂） via https://kotobank.jp/]

演繹的システム開発と帰納的システム開発

- 演繹的システム開発（従来）
  - 計算や判断を行うための知識・規則（モデル・アルゴリズム）を，人が決めてプログラムという形で書き下す
- 帰納的システム開発（というか機械学習）
  - 計算や判断を行うための知識・規則（モデル・アルゴリズム）を訓練データから獲得し生成する
  - それを行うプログラムを人が書き下す

※広く「AI」というだけだとどちらの作り方もありうるが，後者の場合，開発・運用・品質保証が大きく変わる

例（PFN丸山さん資料より）

- 同じ場所で摂氏・華氏両方を計ることを大量に行う
  という手段でも，「摂氏・華氏変換」の実装はできる
  （嬉しいかどうかはともかく）

[丸山宏，PPL 2018，演繹から帰納へ〜新しいシステム開発パラダイム〜
https://www.slideshare.net/pfi/20180305ppl2018]

別の例

この線引きを訓練データから作るイメージ

テナガザル

35 24 210	20 121 24	122 81 20
211 54 42	12 222 90	88 79 116
24 36 98	98 181 31	66 31 198

13 83 33	13 45 94	75 74 111
111 8 73	192 1 221	237 31 1
74 35 122	93 76 244	73 211 45

0 245 210	20 12 114	84 99 100
11 86 99	121 88 91	18 0 77
46 87 121	70 76 122	122 14 94

パンダ

254 32 67	222 88 1	108 76 14
12 86 222	98 75 122	111 74 74
198 87 33	188 173 4	68 176 83

77 81 123	122 158 6	76 63 42
3 3 78	19 183 84	76 63 123
98 83 111	123 7 99	253 48 91

0 24 31	20 21 124	12 101 50
21 54 242	112 22 90	8 79 214
124 56 85	98 99 141	166 1 198

[http://free-photos.gatag.net/]

機械学習（帰納的システム開発）

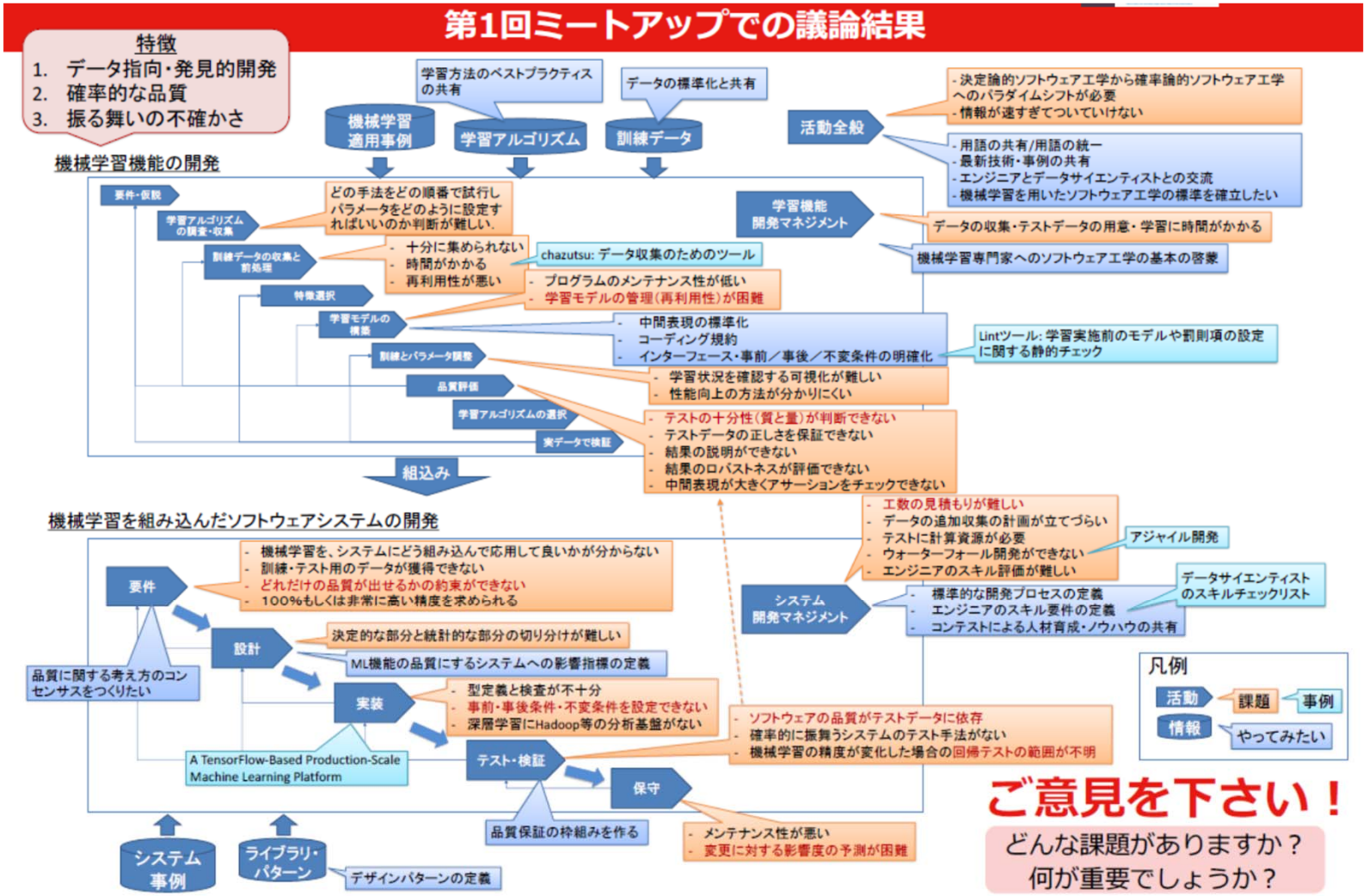
- すごいこと
  - 人が明確に規則として書き出せないことも，訓練データが大量にあれば判断・予測などの計算ができる
    - 「この画像はパンダの画像だ」
    - 「本Aを買った人は，本Bも買う可能性が高い」
  - 訓練データを更新していけば，新しいことに対応できる
    - 今月デビューした芸能人の画像も判別可能に
    - ユーザごとのクセがある音声指示も認識可能に

※ なぜ最近この点が持ち上げられるようになったのか，関連してディープラーニングとは何なのか，は省略（技術進歩があった，と思えばよい）

機械学習（帰納的システム開発）

- 大変なこと
  - 大量かつ「適切な」訓練データが必要
    - 正面写真しか見せなければ，横からの写真を判別できるようにはならない
    - Twitterbotによる「暴言訓練」の例
  - 訓練データがあればうまくいくとは限らない
  - 何がなぜ起きるのかは説明できないことが多い
  - 訓練データ外のデータでどう振る舞うかは未知
    - 想定していた種類だが別のデータ（パンダの例）
    - そもそも想定していない種類のデータ（鏡面タンクローリー）
    - Googleのゴリラの例

機械学習について皆さんから集まった意見



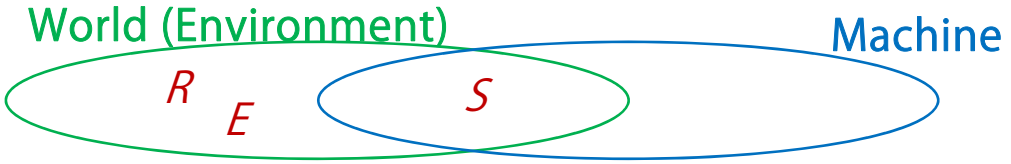
[吉岡ら, SEチャレンジ: 機械学習 x ソフトウェア工学 = 機械学習工学, 2017]
[http://research.nii.ac.jp/~f-ishikawa/work/fose17/]

さらに難しく：
実世界アプリケーション

「システムが何をするか」のそもそも

■Zave/Jacksonのモデル

$$S,\ E \models R$$



マシン（開発の成果物）の仕様 S は，

要求 R を満たす

[Zave et al., Four Dark Corners of Requirements Engineering, 1997]

■例：

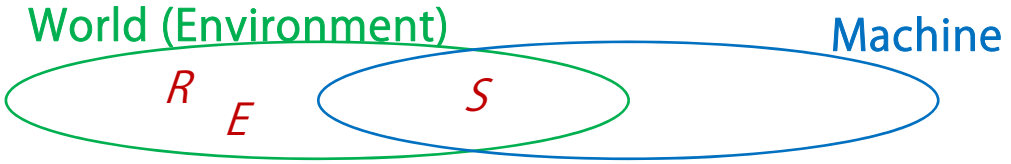
R	• enter 発生回数 ≦ pay 発生回数
S	• pay の発生を検知したら，unlock する • push の発生を検知したら， lock する

[http://www.fujitaka.com/]

「システムが何をするか」のそもそも

■Zave/Jacksonのモデル

$$S,\ E \models R$$



マシン（開発の成果物）の仕様 S は，

想定環境（ドメインの性質） E の下で，要求 R を満たす

[Zave et al., Four Dark Corners of Requirements Engineering, 1997]

■例：

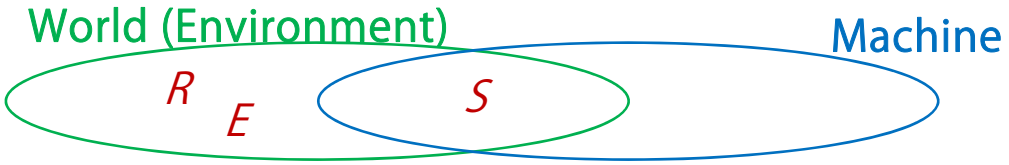
[http://www.fujitaka.com/]

R	• enter 発生回数 ≦ pay 発生回数
S	• pay の発生を検知したら，unlock する • push の発生を検知したら， lock する
E	• push と enter は交互にしか起きないと仮定 • lock が起きてから unlock が起きるまでは push は起こせないと仮定

消える境界

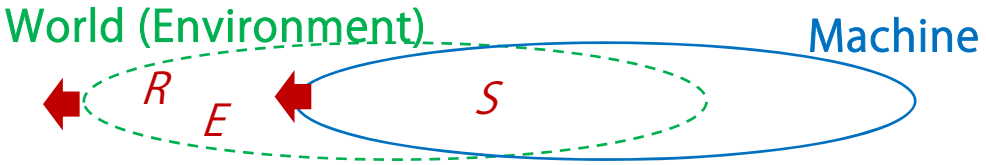
- 先の例は古典的なもの（20年前）
  - しかし，ソフトウェアづくりにおいて，そこまで
    実世界における真の R と向きあっていなかったかも

- 先の例では，その気になれば，V&Vは可能
  SとEがRに対して十分であるかの正当性検証
  Eの仮定の下考えることが現実的かの妥当性確認



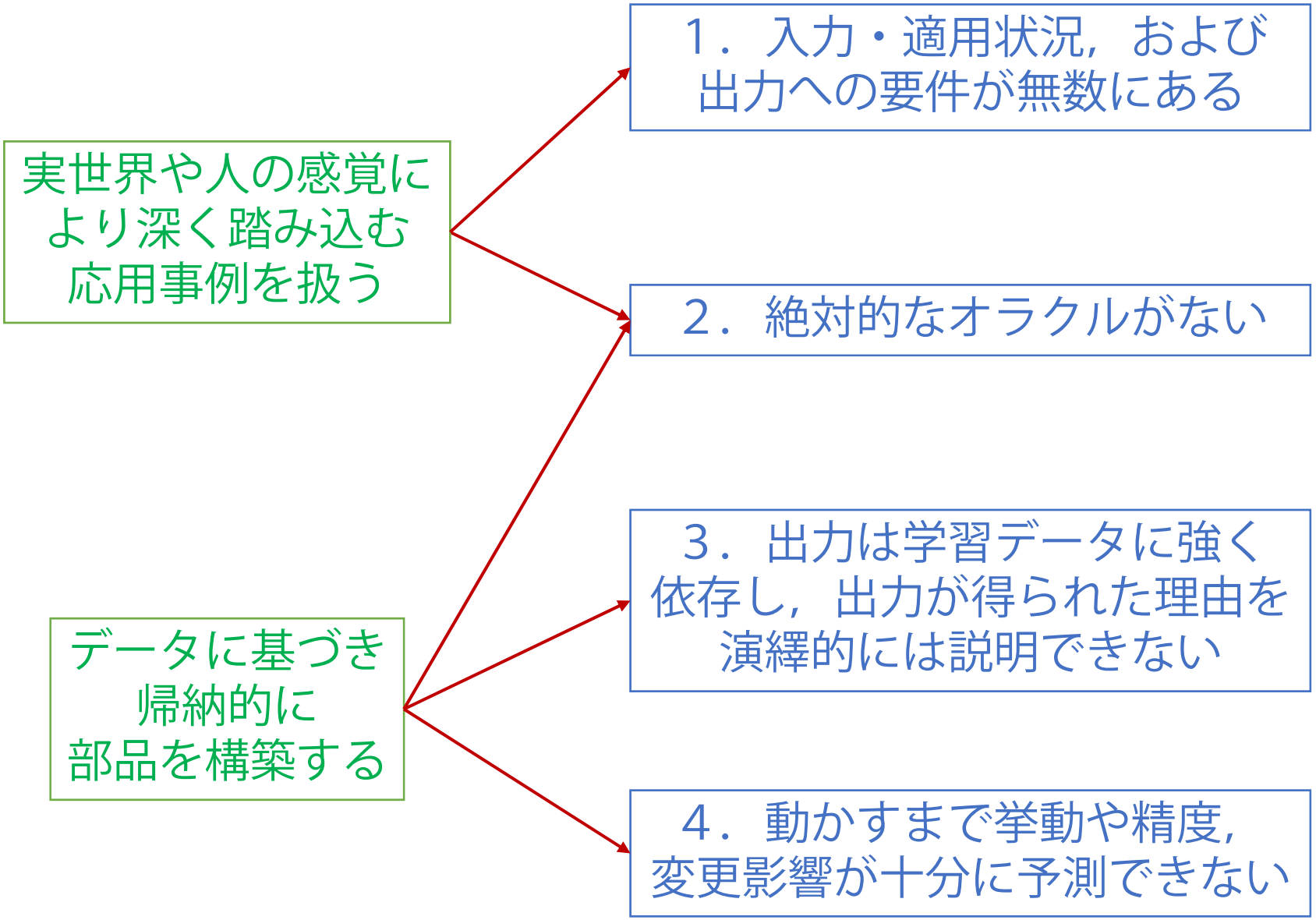
消える境界

- 実世界に踏み込むCPS，IoT（自動運転が象徴的）
  - しかし，ソフトウェアづくりにおいて，そこまで
    実世界における真のRと向きあっていなかったかも
  - ➡ *ソフトウェア制御の範囲（Sの範囲）が広がり，*
    実世界における真のRを直接扱うことが可能に・必要に
  - 先の例では，その気になれば，V&Vは可能
    SとEがRに対して十分であるかの正当性検証
    Eの仮定の下考えることが現実的かの妥当性確認
  - ➡ *RとEにおいて「扱う範囲の境界」が不明確で，*
    さらにRやEの完全・網羅的な把握や予測は不可能



根本的なスタンスの変化

本質的な難しさの要因（の一部）（１）



本質的な難しさの要因（の一部）（２）

1．入力・適用状況，および出力への要件が無数にある

2．絶対的なオラクルがない

3．出力は学習データに強く依存し，出力が得られた理由を演繹的には説明できない

4．動かすまで挙動や精度，変更影響が十分に予測できない

実行時に想定外のことが発生することが原則になる

テスト入力を多数用意しても成否判定が困難・高コストで，明らかなコーディングミス等すら気づかない可能性がある

何が実際実現できているのか把握・説明できず，問題の原因同定・修正も難しい

要求の実現可否・実現コストや変更の影響を事前に把握しての分析・意思決定は難しい

本質的な難しさの要因（の一部）（３）

実行時に想定外のことが
発生することが原則になる

テスト入力を多数用意しても
成否判定が困難・高コストで，
明らかなコーディングミス等
すら気づかない可能性がある

何が実際実現できているのか
把握・説明できず，問題の
原因同定・修正も難しい

要求の実現可否・実現コストや
変更の影響を事前に把握しての
分析・意思決定は難しい

ん？全部「不確かさ」の
話じゃないか！

要求と評価指標，
入力，実装，出力が不確か

自然に必要となるだろうスタンス

実行時に想定外のことが
発生することが原則になる

より上位のシステム全体で
失敗の対応策を打つ

テスト入力を多数用意しても
成否判定が困難・高コストで，
明らかなコーディングミス等
すら気づかない可能性がある

実行時に継続的な監視・評価を
行いフィードバックする

システムの上位の目的
（ユーザ体験や安全性）の観点
からフィードバックを得る

何が実際実現できているのか
把握・説明できず，問題の
原因同定・修正も難しい

疑似オラクルを用意し
大量に（ゆえに自動）テストをする

構築過程から部分部分に対し
確信が持てる構築方法を用いたり
検証手段を設けたりする
（想定する性質のアサーション等）

要求の実現可否・実現コストや
変更の影響を事前に把握しての
分析・意思決定は難しい

探索的・試験的に
仮説・分析・評価を繰り返す

不確かさへの対応

ソフトウェア工学と不確かさ

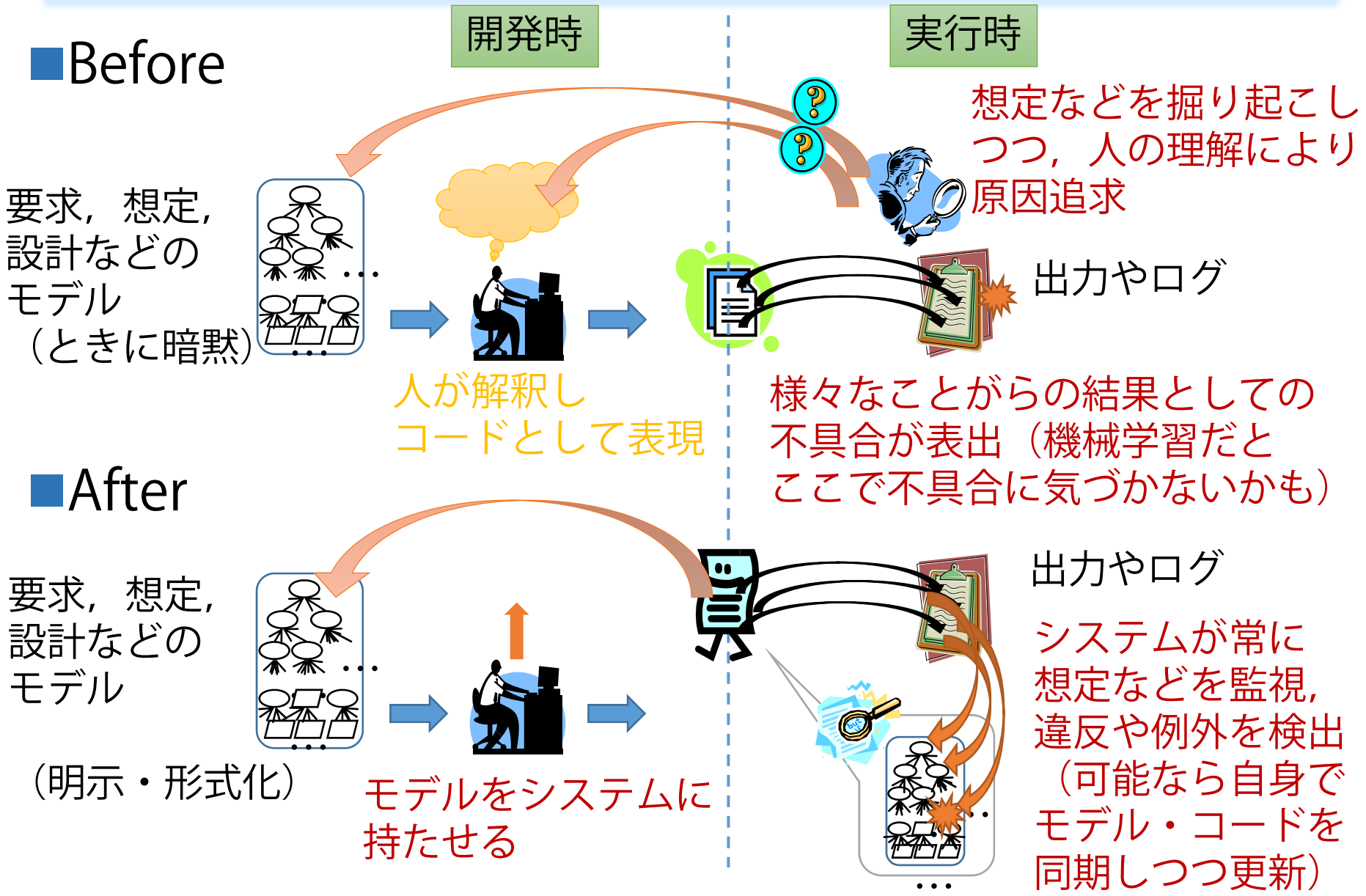
- Models@run.timeアプローチが代表的

要求，仮定・想定や，設計に関する意思決定の論理
などのモデルを明示化し，システムが実行時にも活用

➡ *現実とモデルのずれを検出・同定（実行時検証）*
➡ *意思決定の論理をたどり直すことにより，*
実行中の再設計なども可能に（自己適応）

- 自己適応のためのソフトウェア工学研究において
  この10年ほど盛んに取り組まれている一つの方向性
- ただし帰納的な振る舞いを主眼においてはこなかった

Model@run.time



機械学習の先端企業から出ている原則・指針

仮定・想定を「テスト」せよ，は方向性の一つ

（実装の正しさだけではなく）

- ベストプラクティス集
  - 例： データの統計を追跡するなどして，問題として顕在化しない失敗を見張れ

[Zinkevich, Rules of Machine Learning: Best Practices for ML Engineering, 2016]

- どれだけ「テスト」できているかの評価スコア
  - 例： 個々のfeatureの入力値範囲や分布が予想と合うか
  - 例： 直接的なメトリクス（精度等）と実影響のあるメトリクス（クリック率等）との相関はどうか，例えばあえて前者を悪くしてA/Bテストすると何が起きるか

[Breck, What's your ML Test Score? A rubric for ML production systems, 2016]

と呼んだら
買ってくれますか？

（AI的？）自動テスティング

（他にもいろいろあるが）一つの大きな課題

- 無数の入力・環境に対するテストを考える
  - 連続値の様々な値をとるパラメータが多数
  - 機械学習モデル，物理挙動のSimulinkモデル，それらを連動するようなシステムモデルなど

➡ ***どう立ち向かうか？***

脱属人化・職人芸（勘と経験）

一定の保証（という外部への説明）

費用対効果

  - ベテランの経験・勘？
  - 大量のランダムテスティング？
  - 網羅性のある形式検証？（スケールする抽象モデル？）

サーチベースドテスティング

- サーチベースドテスティング（Search-based）
  - 最適化技術（特に進化計算・メタヒューリスティック）により「欲しいもの」を表すスコアを最大化するようなテストスイートやテストケースを生成
    - 例：「車が人に最も近づくような危ういテスト」ケースを生成
    - 例：「前バージョンと比べ出力が大きく変わる」入力を発見
    - 例：「高カバレッジで数が小さい」回帰テストスイートを生成

[S. Ali et al., Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation, 2010]
[http://www.evosuite.org/]

サーチベースドテスティングのレベル感

- この10年ほど盛ん
  - というか，テスト生成に限らず，プログラム修復なども
    やってしまう「サーチベースドソフトウェア工学」
- 2018年のJava Unit Testing Competition
  - 比較用に複数ツールを組み合わせた「最強ツール」は，
    人が作ったものよりよいテストスイートを10秒で生成
  - ここでの評価基準は，コードカバレッジとミューテー
    ションスコア（人工バグの検出率），テストケース数
  - 例えば，コード（の意図）が読めないテストケースが
    出るかも（理解容易性や自然さもホットトピック）

[https://github.com/PROSRESEARCHCENTER/junitcontest/blob/master/README.md#6th-junit-contest]
[Molina et al, Java Unit Testing Tool Competition - Sixth Round]

CPSのサーチベースドテスティング研究例

- Simulinkモデル対象
  - スコア付けをヒートマップと機械学習によりアタリを付け，効率的に「悪いケース」を探す

    [R. Matinnejad, et al., MiL Testing of Highly Configurable Continuous Controllers: Scalable Search Using Surrogate Models, 2014]
  - 全く異なる多様な出力を含むようなテストスイートを生成（人目での確認用）

    [R. Matinnejad, et al., Automated Test Suite Generation for Time-continuous Simulink Models, 2016]
  - 形式検証によるパス生成も組み合わせて「悪いケース」を効率的に探す

    [S. Kobuna, et al., Validation of Control Software by Search-Based Testing Using Formal Methods, 2016]

※ 機械学習関連は後述

おまけ： 最初の一歩

- 小さめSimulinkモデルでお試し実装してみた例
  - ユーザ操作（アクセルペダル踏み込み等）に対する歩行者への衝突有無・衝突速度が焦点
  - 進化計算の既存フレームワーク上で100行いかない程度
  - 結果の一例： 10回の試行で，衝突速度が一定以上となるケースを常に5000シミュレーション以内に発見（ランダムだと10回中2回）
  - あとは，計算資源にコストをかけて並列化してぶん回す，ドメイン知識による高速化を入れる，などすればよい（それらをしやすい・した方がよいのが進化計算・サーチベースドテスティング，ある意味「力業」）

反例探索（Falsification）

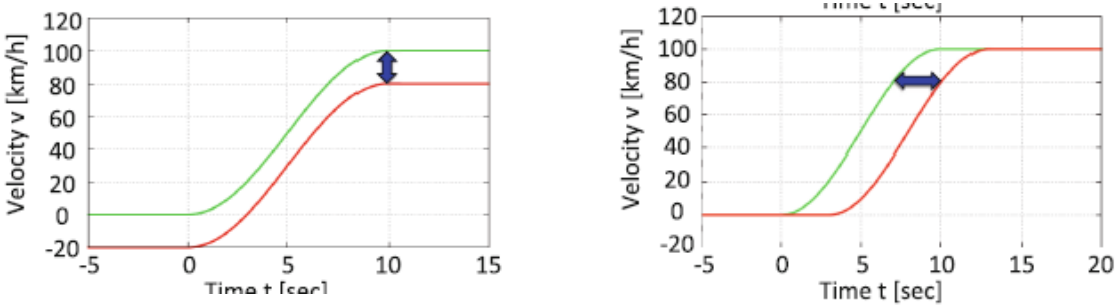
一方，形式検証のコミュニティでも似たことが・・・

- 仕様は定量化された時相論理により形式的に記述

  「速度は，イベントBが発生してから10秒以内に80以上になる」

  - 充足？どれだけロバスト？（9.9秒後 対 6秒後）
  - 違反？どれだけ遠い？（10秒後に78 対 10秒後に50）

➡最適化により「悪いケース」を探索



画像元：[Akazaki et al., Time Robustness in MTL and Expressivity in Hybrid System Falsification, 2015]

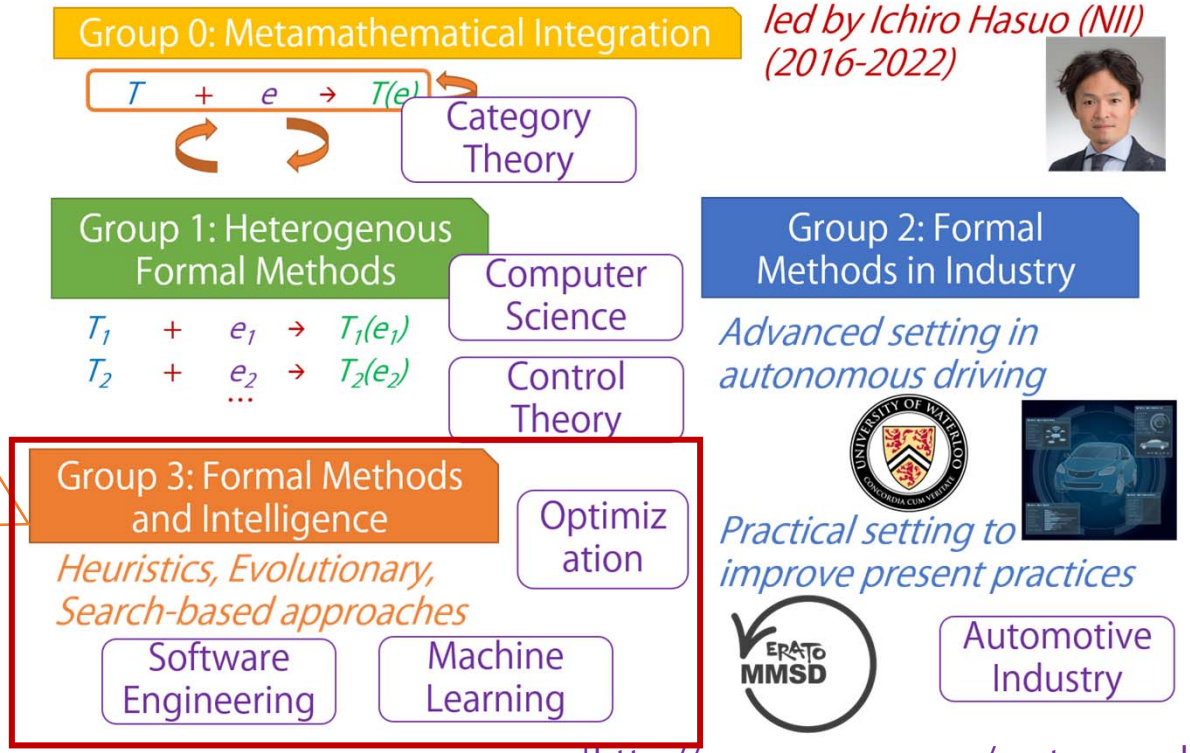
おまけ： ERATO-MMSD Project

- といった技術に強力な体制で取り組んでいます！
  - 問題投げ込み大歓迎
  - プロジェクトの別の軸は，「数学強い人たちによる，理論の『横展開』」で，理論的成果とも連動

成果の例：

「モデルファミリー」の反例探索の仕方を学習，多少モデルが変わっても効率よく反例検出可能

これは「AIによるテスティング」と言える



[http://group-mmm.org/eratommsd/]

石川はココ

機械学習システムの（AI的？）テスティング研究

機械学習モデルのテスティング

- 機械学習モデル（画像識別器や進路判断器等）のサーチベースドテスティング
  - 既存画像に雨や覆い等を加えることで様々な入力を作る
  - 「ニューロンカバレッジ」を最大化するテストスイートを出す，つまり「ホワイトボックス・テスティング」
  - これによりテストスイートの多様性・網羅性を上げつつ，「悪いケース」を探す
    - 同じ入力を別バージョンの実装に入れたときと比べて，より大きく出力結果が変わるケース
    - メタモルフィック関係が満たされない不具合ケース（後述）

1.1 original　　1.2 with added rain

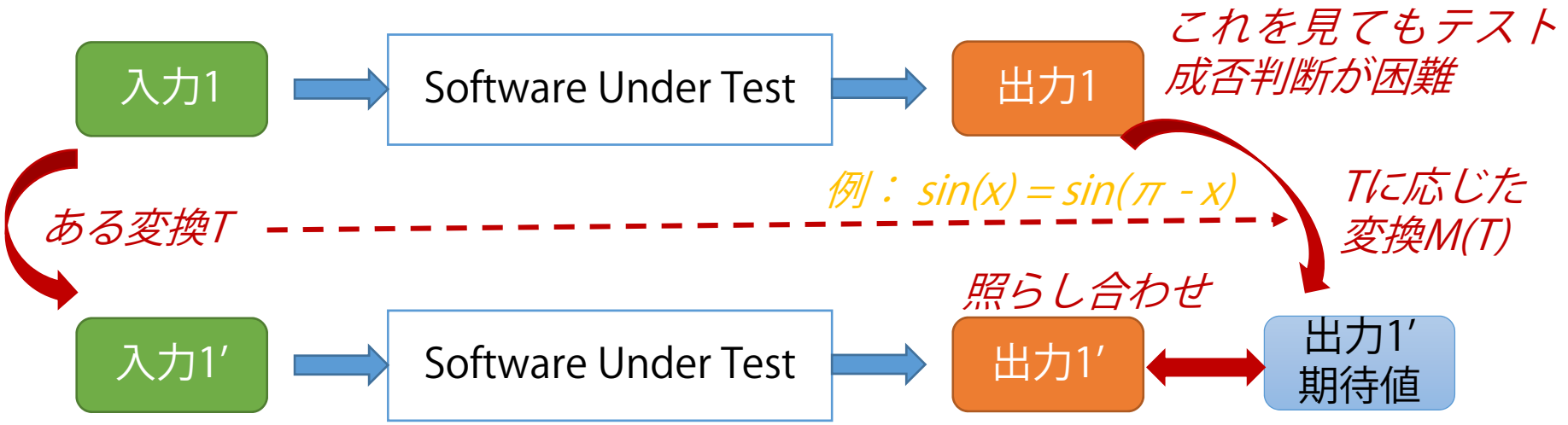
[Pei et al., DeepXplore: Automated Whitebox Testing of Deep Learning Systems, 2017]

画像元：[Tian et al., DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars, 2018]

補足：メタモルフィックテスティング

- 「正解がない or 作るのが大変」問題への対応
  - 多数の入力を用いてテストをしたくても，各出力の正しさ（テスト成否）を判定するのが困難または高コスト

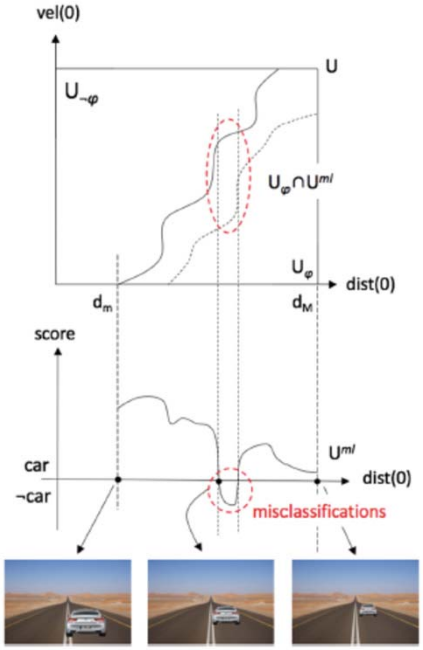
➡ 「入力を変えると出力はこう変わるはず」という関係を検証，既存テストケースから多数のテストケースを生成



[Segura et al., A Survey on Metamorphic Testing, 2016]
[Jarman et al., Metamorphic Testing for Adobe Data Analytics Software, 2017]
[Lindvall et al., Metamorphic Model-based Testing of Autonomous Systems, 2017]

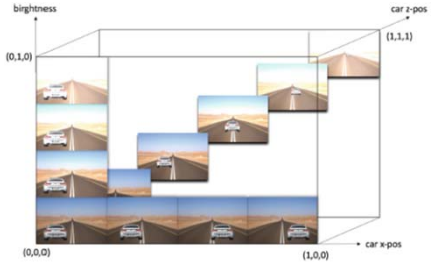
機械学習システムのテスティング

- システム全体の要求を踏まえるのが重要
  - 機械学習部品（生成されたモデル）の精度だけむやみに突き詰めてもしょうがない
  - 例： 遠くの物体を誤認識しても衝突には至らないかも



例題：自動ブレーキシステム（人の操作や先行車の位置が入力，「先行車との距離が一定以上ある」ことが要求）

1．完璧な機械学習部品を用いると要求を満たすが，常に失敗する機械学習部品を使うとそうならないような入力パラメータ領域を絞り込む



2．その領域に限定して機械学習部品が誤認識するような画像を探す

3．その画像を用い要求を満たさないケースを反例探索

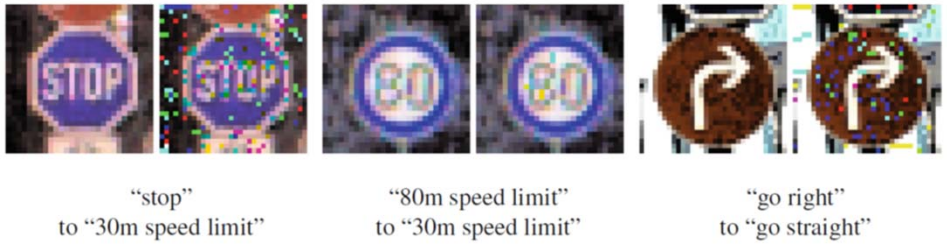
[Dreossi et al., Compositional Falsification of Cyber-Physical Systems with Machine Learning Components, 2017]

その他の話題

- メタモルフィックテスティングの適用
  - 時系列分析で入力に一定の幾何学変換をかける
  - ドローンシステムで地図を回転など変化させる
  - より分類が難しいコーナーケースを生成する

[Jarman et al., Metamorphic Testing for Adobe Data Analytics Software, 2017]
[Lindvall et al., Metamorphic Model-based Testing of Autonomous Systems, 2017]
[中島,データセット多様性のソフトウェア・テスティング, 2017]

- 形式検証技術の応用
  - 画像認識において，「一定範囲の画像操作」を行っても認識結果が変わらないかどうかを網羅的に検証



"stop" to "30m speed limit"　"80m speed limit" to "30m speed limit"　"go right" to "go straight"

画像元：[Huang et al., Safety Verification of Deep Neural Networks, 2017]
[Mirman et al., Differentiable Abstract Interpretation for Provably Robust Neural Networks, 2018]

おわりに

機械学習関係なく考えていただきたいこと

- やっていけるだろうか？
  - 同じやり方・「自分たちの考えたベスト」だけで
    今後の5年・10年立ち向かっていく
  - 「難しい」技術は「うちにはムリ」・人手と工数で
    （「難しい」のは求められているレベルでは？）
- 機械学習の人たちのやり方
  - 論文の新技術をすぐさま実装
    （事前評価が難しいからこそ作って実投入してみる）
  - 研究チームと開発チームが密に連動する（Google）

[https://www.slideshare.net/hamadakoichi/dena-ai-service-development]
[Sculley et al., Machine Learning: The High-Interest Credit Card of Technical Debt, 2014]

おわりに

※ これも機械学習（AI）に限らず

今までソフトウェア屋さんがサボってきた・やりきれてなかったことが突きつけられているのかも

- 画面の外の実世界での意義や安全性，ユーザの感性的な満足の追求や評価，探索的・試験的な投資や問題解決，仮説と検証による科学的な判断や評価，常に測定，数学の理解・活用，Correctness-by-Construction，現実問題の数学的問題（最適化等）への帰着，実行時検証やテスト自動化（入力生成・疑似オラクル），といったことへの本気のパラダイムシフト，・・・

楽しんで切り拓いていきましょう！