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

自動（運転）車システムのための AI的自動テスト生成

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/

AI4SEセミナー　　2018/12/14

自己紹介

- ソフトウェア工学と先端自律・スマートシステム
  - 形式手法，テスティング，自己適応，
    要求やディペンダビリティの分析・論証
  - サービス指向（Web・IoT），CPS（特に車）
- 産業界向け教育・応用研究
  - トップエスイー，日科技連SQiP，電通大社会人博士
- 最近（１）：自動（運転）車ソフトのテスト
- 最近（２）：機械学習ベースAIのテスト・品質

MLSE 機械学習工学研究会
MACHINE LEARNING SYSTEMS ENGINEERING

ERATO MMSD

QAI
ＡＩプロダクト
品質保証
コンソーシアム

自動車システムへの応用

ERATO-MMSD プロジェクト

led by Ichiro Hasuo (NII)
(2016-2022)

Group 0: Metamathematical Integration

$T + e \rightarrow T(e)$

Category Theory

Group 1: Heterogenous Formal Methods

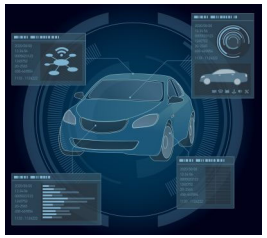
$T_1 + e_1 \rightarrow T_1(e_1)$
$T_2 + e_2 \rightarrow T_2(e_2)$
…

Computer Science

Control Theory

Group 2: Formal Methods in Industry

Advanced setting in autonomous driving



Automotive Industry

Practical setting to improve present practices

Group 3: Formal Methods and Intelligence

Heuristics, Evolutionary, Search-based approaches

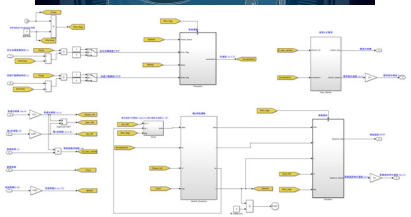
Optimization

Software Engineering

Machine Learning

再掲：（ものすごく簡単な）問題イメージ

Simulinkモデル：
自動ブレーキ付の車の挙動



入力

ユーザの挙動
- アクセルペダル
- ブレーキペダル

環境
- 初期速度
- 歩行者の位置・動き
- 路面状況
- …

出力
- 衝突有無
- 衝突速度

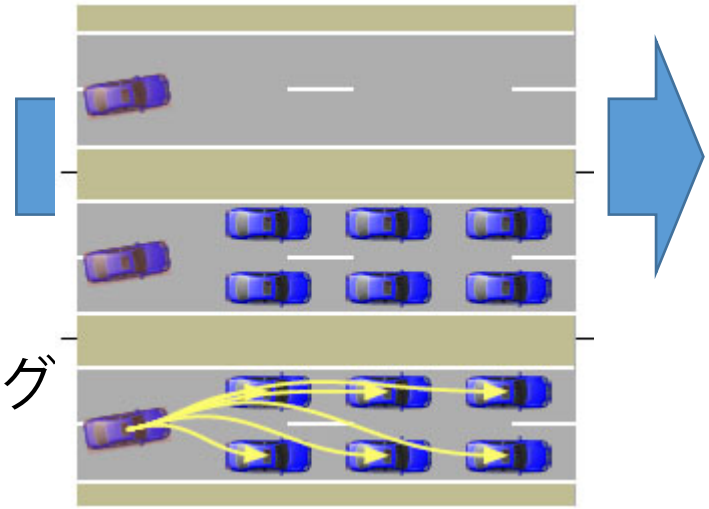
- *微分方程式が入る・連続的挙動（数学的には，ハイブリッドオートマトンで表現できるようなもの）*
- *実行可能なものがあるだけのことも（シミュレーションモデルやコード，実機）*

別の問題イメージ

Cコード：
自動運転コントローラ

入力

シミュレーション
設定
- 歩行者配置
- 道路の構造
- 信号変化タイミング
- 他の車の動き
- ・・・



出力
（のスコア付け）

- 安全性
（様々な種類）
- 快適性
- 法令遵守
- ・・・

[図： McNaughton, Parallel algorithms for real-time motion planning, 2011]

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]
  - 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]
  - 全く異なる多様な出力シグナル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]

最初の一歩はどんなもん？（1）

- 先の自動ブレーキの例でお試し実装してみた例
  - 進化計算の既存フレームワークが利用できる
  - 「今回突きたいポイント」のコードがメイン

```java
/**
 * Generic implementation of the evaluation step: running simulation according
 * to the number of samples in a solution and the number of model files and
 * delegating the fitness evaluation to
 * {@link #evaluateSimulationResult(double[], double[], Solution)}
 *
 * @see org.moeaframework.core.Problem#evaluate(org.moeaframework.core.Solution)
 */
@Override
public final void evaluate(Solution solution) {
        /*
         * Input array is interpreted as: sample0-param0, sample0-param1, ... then
         * sample1-param0, sample1-param1, ... (sample-param order)
         */
        double[] encodedInputs = EncodingUtils.getReal(solution);
        double[][] inputs = new double[numberOfSamplesInSolution][INPUT_VARIABLES.length];
        double[][] outputs = new double[targetModelFiles.length][numberOfSamplesInSolution];

        for (int i = 0; i < targetModelFiles.length; i++) {
                for (int j = 0; j < numberOfSamplesInSolution; j++) {
                        for (int k = 0; k < INPUT_VARIABLES.length; k++) {
                                inputs[j][k] = encodedInputs[INPUT_VARIABLES.length * j + k];
                        }
                        outputs[i][j] = runSimulator(targetModelFiles[i], inputs[j]);
                }
        }
        solution.setObjectives(evaluateSimulationResult(inputs, outputs, solution));
}
```

シミュレーション回すコード
（歩行者との最小距離などが計算される）

```java
@Override
protected double[] evaluateSimulationResult(double[] inputs, double output, Solution solution) {
        double[] ret = new double[6];
        if (output > 0) {
                ret[0] = 0;
                ret[1] = 0;
                ret[2] = 0;
                ret[3] = 0;
                ret[4] = 0;
                ret[5] = output;
        } else {
                double[] scores = [redacted]ASILScorer.calculateASILScoreDetails(inputs, output);
                int minEidx = (int) scores[6];
                for (int i = 0; i < 4; i++) {
                        if (i + 2 == minEidx) {
                                ret[i] = -scores[i + 2];
                        } else {
                                ret[i] = 0;
                        }
                }
                ret[4] = -scores[7];
                ret[5] = -scores[8];
                solution.setAttribute("s", scores);
        }
        return ret;
}
```

状況の非レア度と衝突の重大さなど
複数観点でスコア付け

最初の一歩はどんなもん？（2）

- 続
  - 一番簡単な「悪い1ケースを探す」という設定での結果例：
    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）
  - 他には，「非レア度」と「衝突重大度」などのパレートフロント生成，バージョンアップの影響が大きく出るしなりの生成など，ごく簡単に作れた
  - 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，などが必要

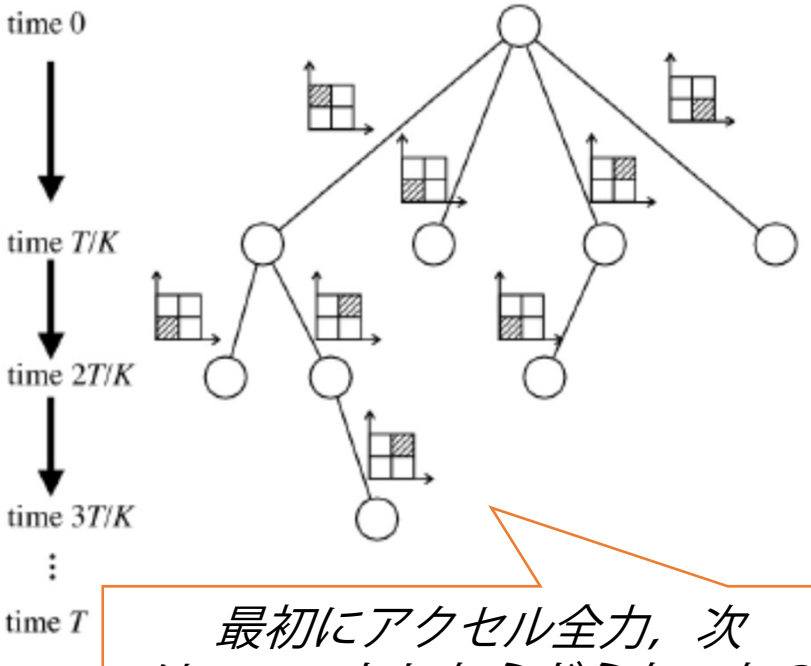
やっている研究（一例）：動機

- 局所最適の問題
  - 「あら探し」意識が強すぎると，たまたま見つけた「やや悪い状況」周辺ばかりを探し回る
  - 全然違う状況の方が「もっと悪い状況」があるかも
- 探索知識の活用
  - 「現時点での最良（最悪）」しか覚えていない
  - 「これだけちゃんと探し回った」ということをちゃんと言えるとよい

➡ *「試行結果の知識」をちゃんととっておき，*
「他の可能性を探るための試行」もちゃんとする

やっている研究（一例）：動機

- 囲碁プレーヤーなどで使われる手法を適用
  （Monte Carlo Tree Search，バンディット問題）
  - 探索空間（入力シグナルの時系列で分割）の「ニオイ」を記録
  - 「しっかり一通り確認した」という主張をするためにも活用可能

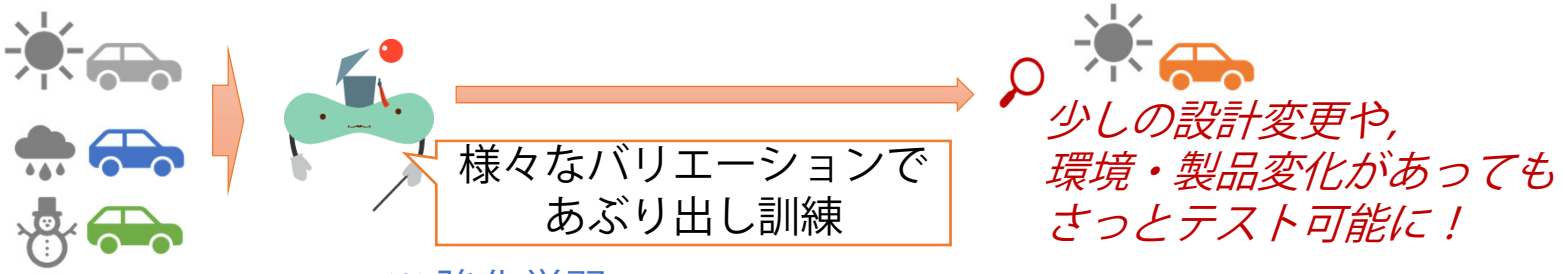


[Zhang et al., Two-Layered Falsification of Hybrid Systems Guided by Monte Carlo Tree Search, EMSOFT'18]

やっている研究（別の例）

- 様々なモデルを検証したい
  - プロダクトライン的発想，ただし，環境の多様性も主軸

例： 複数のプロダクト・環境設定（モデル書換）に対し
効果的な検証器を事前訓練で作りあげる



※ 強化学習

[Kato et al., Falsification of Cyber-Physical Systems with Reinforcement Learning, MT-CPS'18]

例： 危険性が高い・変化に弱い
プロダクトの種別や環境設定などを抽出

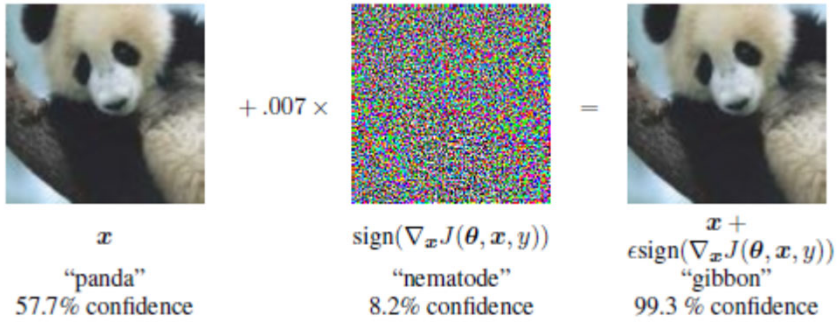
サーチベースド「設計良し悪し分析」「敏感さ分析」

他の人たちの
研究の紹介

AIシステムのテストへの応用

よく知られた課題（の一つ）：敵対的サンプル

- 優れた画像識別器が少しのノイズで誤認識

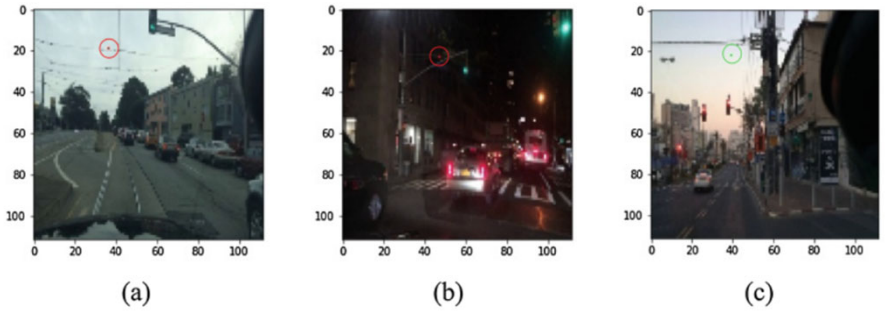


有名な例
「パンダ」が「テナガザル」に

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

物理的なテープ貼付などによる誤認識発生

[画像元： Ackerman, Slight Street Sign Modifications Can Completely Fool Machine Learning Algorithms, 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]

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

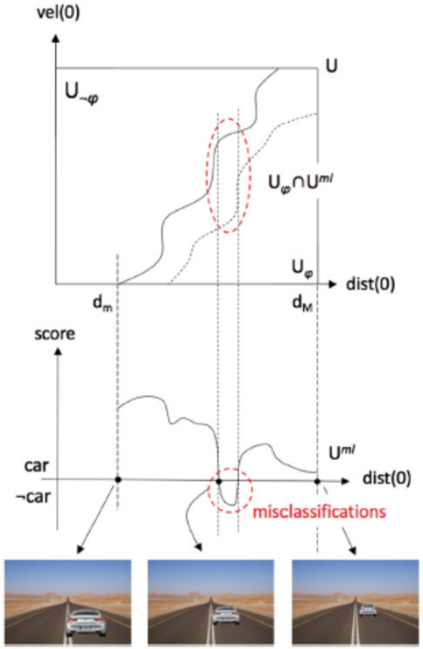
1.1 original　　1.2 with added rain

[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]

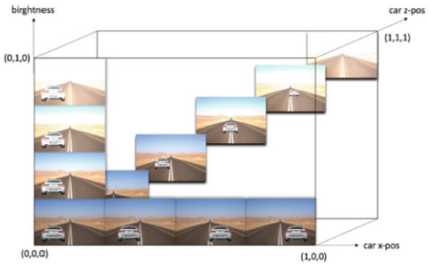
機械学習システム全体のテスティング

- 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キリがない
  - 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



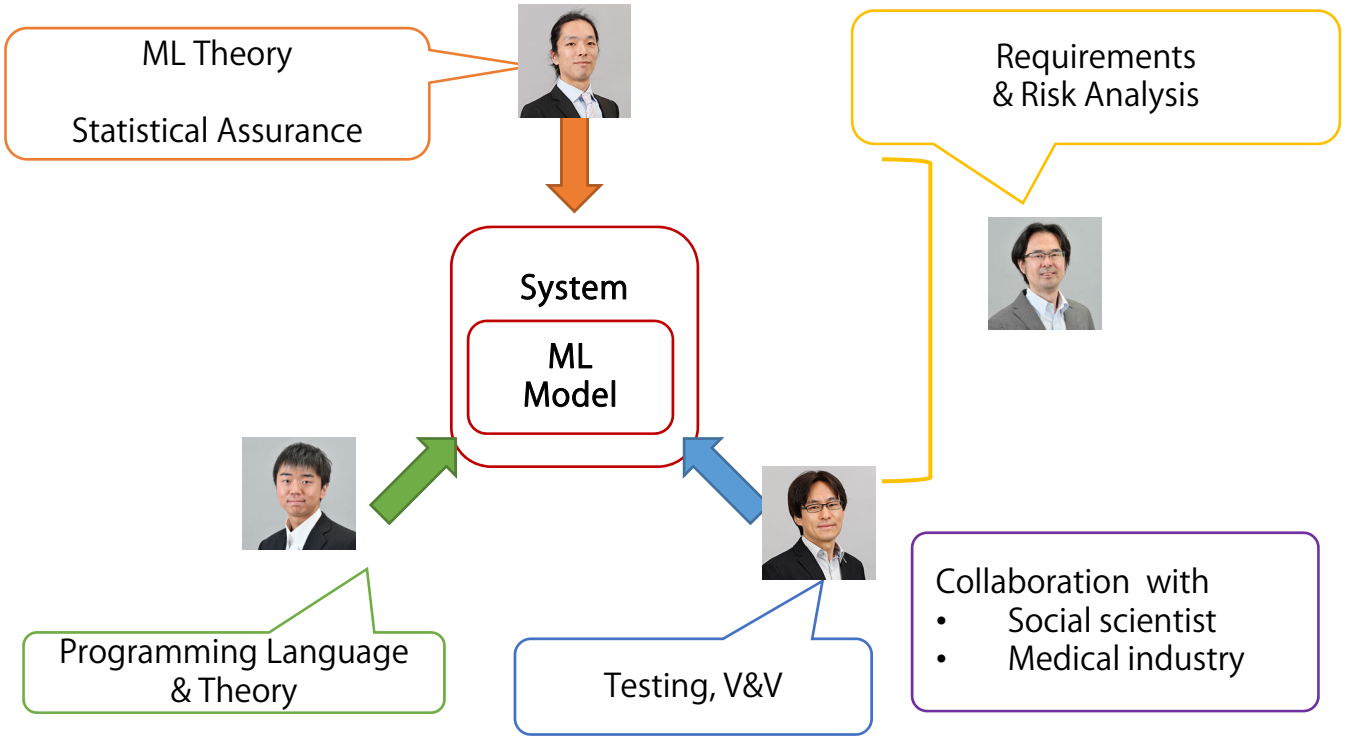
2．その領域に限定して機械学習部品が誤認識するような画像を探す

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]

自分周りでのこれから

- 1か月前からJSTプロジェクト開始
  - 「高信頼な機械学習応用システムによる価値創造」
  - 今紹介したような技術の深化＆応用先拡大も取り組み内容の一つ（今は画像・DNNばかり）



おわりに

- ますます複雑になるシステム
  - 人手で何かを探り当てるには複雑過ぎる
  - 特に自動運転が「高い信頼性要求」「機械学習利用」「オープンな要求・環境」の全部入り

➡「賢い力業」を追求しています！

問題持ち込み・相談・議論大歓迎です！
（適切か，とかはあまり考えなくてよいです）