ソフトウェア開発におけるMathematica利用

藤倉　俊幸

2020年　12月　2日

自己紹介

- 組込ソフトの関連の仕事をほぼ40年やっています.
  - RTOSの開発・研究
  - リアルタイムシステム
  - 形式手法
  - セキュリティ
  - その他
- 著書
  - リアルタイム/マルチタスクシステムの徹底研究
  - 組み込みソフトウェアの設計&検証
  - 組み込みソフトへの数理的アプローチ
  - モデルに基づくシステムズエンジニアリング
  - 仕様書の読み方と書き方

Agenda

1. はじめに: ソフトウェア開発とMathematica
2. ソースコードメトリックスへの応用
3. 開発プロセスの検証
4. AutoEncoderによる自動車へのサイバー攻撃検知
5. まとめと要望

1. はじめに: ソフトウェア開発とMathematica

- ソフトでは，離散系の処理が多い
  - とくにグラフ関連が多い
- Mathematicaは何でもそろっていて使いやすい
  - 数学の専門外でも，数学をそこそこ使いこなせる．
- Mathematica可視化は強力
  - 数学的な話になると身構えてしまう人が多いが，実際に計算してみせると，たいてい「そうなのか・・・」と言う反応になる．
- 関数型プログラミングの元祖?
  - 複雑な処理でもあっさりと書ける．
- フリーのWolfram Playerで動作可能なサンプルを配布できる．

2. ソースコードメトリックスへの応用

- ソースコードメトリックスとは，ソースコードやアーキテクチャの状態を評価するための指標になるモノ
  - ライン数
  - コメント量
  - 複雑度
  - 結合度
  - 凝集度
  - など
- トレードオフの関係にあるので最適値が良く分からないという問題があった.
  - シミュレーションによってそれぞれのメトリックス値の特徴を調べた.

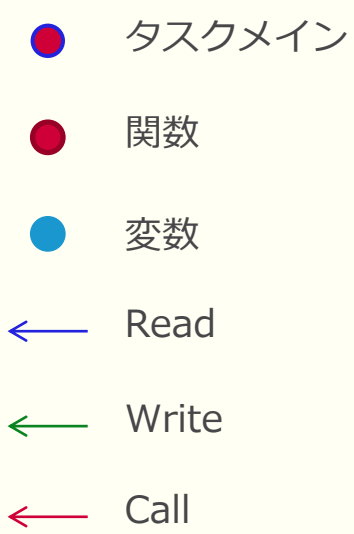
ソフトウェアの構造単位

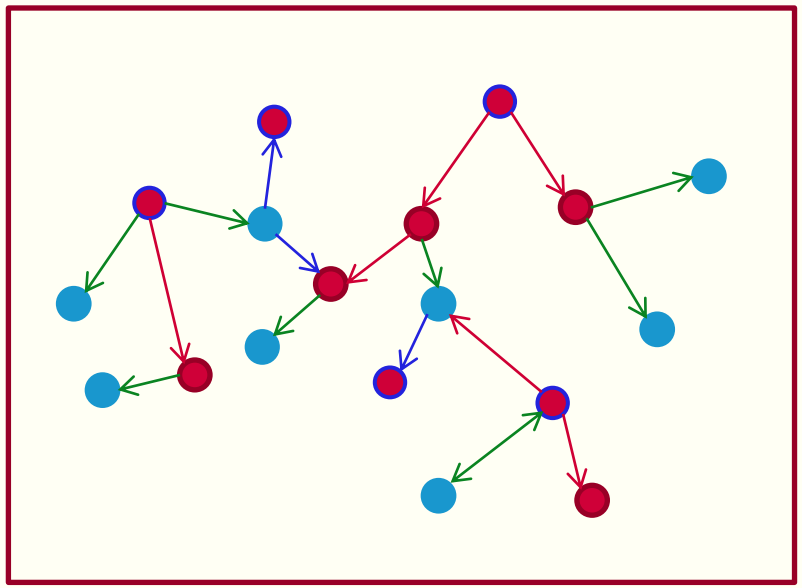
- 関数とグローバル変数
  - 最小構成単位
- ファイル
  - コンパイルの単位
- モジュール/コンポーネント
  - 部品
- システム
  - 製品の単位

} 上位構造，アーキテクチャなどと呼ばれる．

ソフトウェアの基本構造

- ソフトウェアは，基本的には関数と変数だけで動いているし，表現することもできる.
  - Mathematicaのモデルも同じ
- モジュールあるいはコンポーネントはこれらの部分集合で表現できる.
- 部分集合の作り方を設計と言ったり，アーキテクチャと言ったりしている.

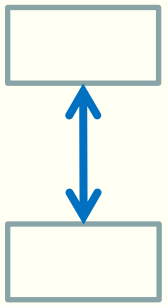


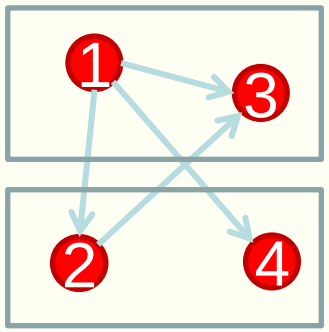


ソフトウェアの上位構造

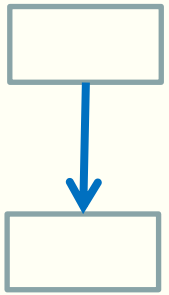
- 同じ基本構造でも，上位構造には良い構造と悪い構造がある．
- 良さ加減を数値化したのがメトリックス

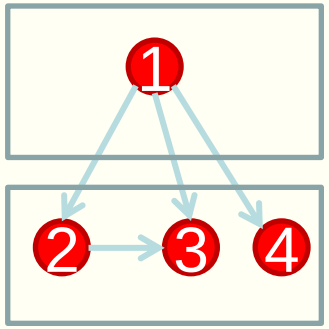
悪い





良い

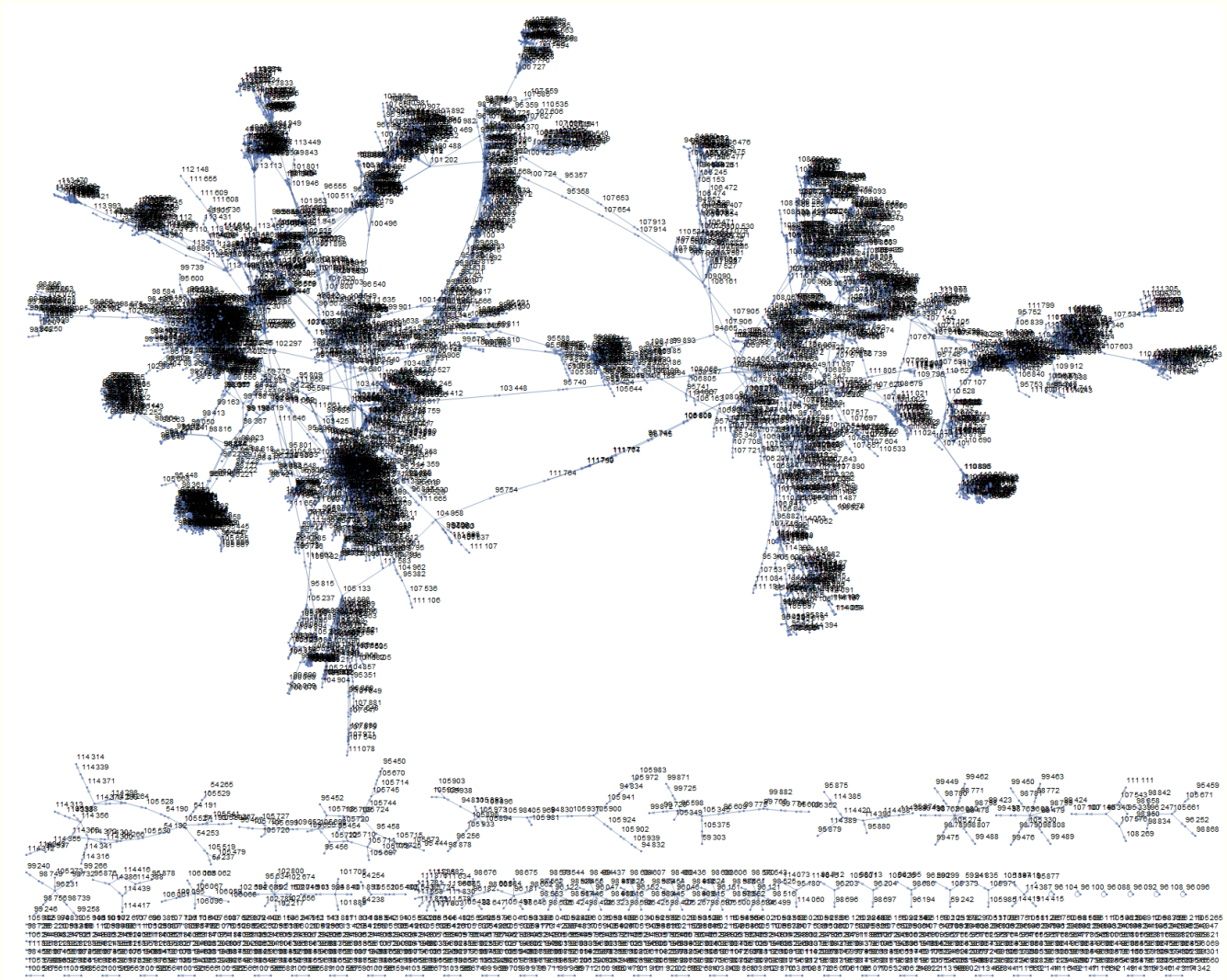




上位構造

実際の基本構造(コールグラフ)

- 実際の製品のソースコードから
  Mathematicaで生成したコールグラフ
- コールグラフ
  - ノード: 関数
  - エッジ: 関数の呼び出し関係
  - 約20万関数，やや小さいシステム



凝集度のシミュレーション

- コールグラフのシミュレーションにより凝集度の特性を調べる.
  - 善し悪しの判断基準を確定するため.
- 手順

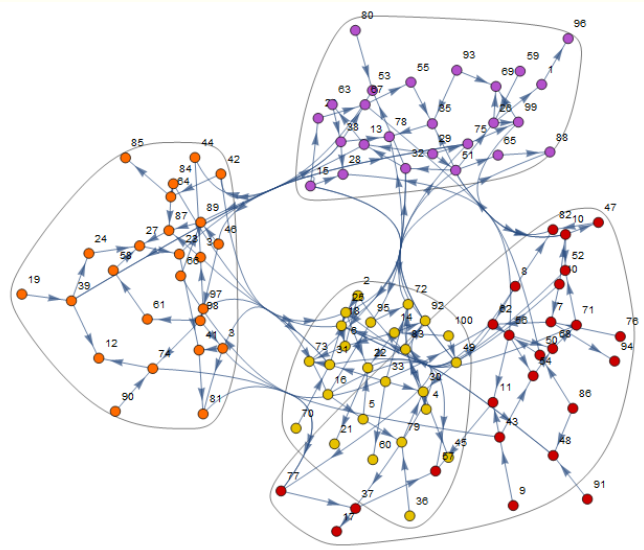
1. コールグラフをランダムに生成する.
2. 関数をグルーピングして上位構造を生成してメトリックスを計算する.
   - グルーピングはグループ数を指定して，ランダム，パーティショニング，コミュニティ生成を利用した.
     - ランダム: Combinatorica`RandomKSetPartition[]
     - パーティショニング: FindGraphPartition, 異なる部分への参照が最小になるような頂点の分割を求める.
     - コミュニティ生成: FindGraphCommunities,多くの辺が同一コミュニティ内の頂点同士を繋いでおり，比較的少数の辺が他のコミュニティの頂点と繋がっているコミュニティを求める.
3. ①②の手順を500から500000回繰り返して分布をとる.

コールグラフ生成と凝集度計算

- コールグラフ生成

nf: 関数の数　　nr: 関連の数 (callの数)

```
CreateCallGraph[nf_, nr_] := Block[{allEdges},
  allEdges = Map[DirectedEdge[#[[1]], #[[2]]] &, Permutations[Range[nf], {2}]];
  System`Graph[Range[nf], RandomSample[allEdges, nr], VertexLabels -> "Name"]
  ]
```



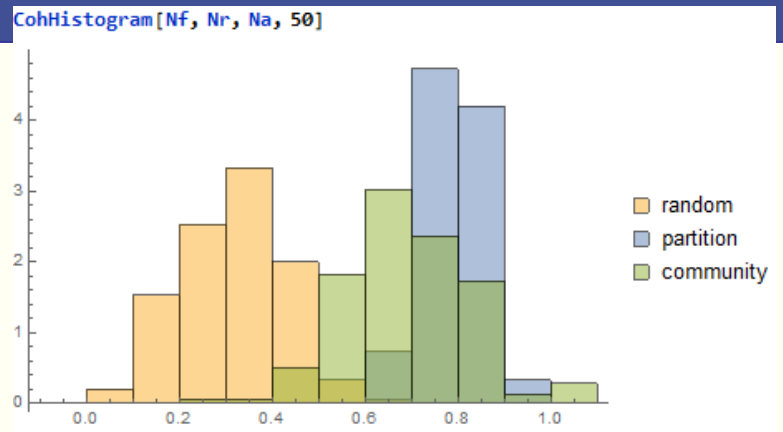
- 凝集度計算

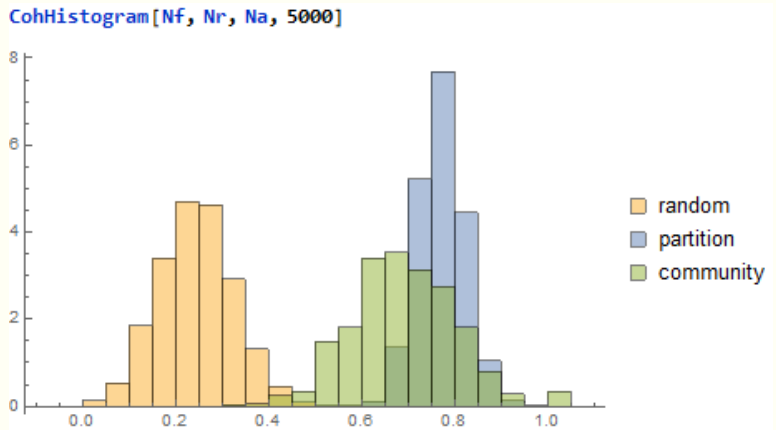
gr: 関数のコールグラフ　archList: アーキテクチャ:関数のグルーピング

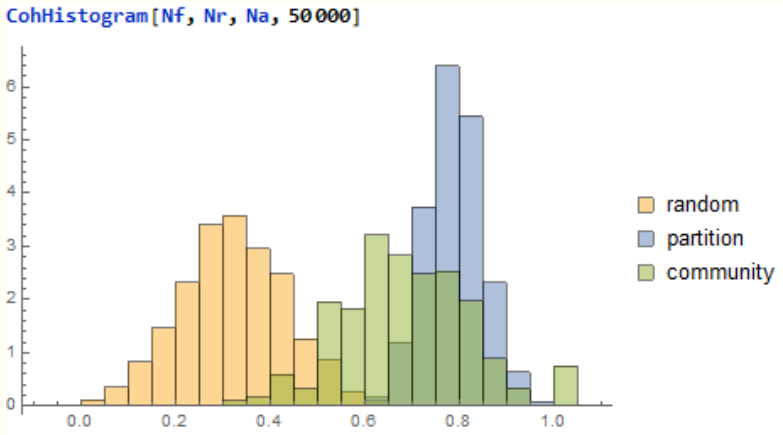
```
CohesionList[gr_, archList_List] := Block[{IList, EIList, coList},
  IList = (*内部リンク*)Map[ System`EdgeCount[System`Subgraph[gr,#]]&, archList];
  EIList = (*外部リンク+内部リンク*) Map[Total[ Map[ System`EdgeCount[gr,# ¥[DirectedEdge]_]&,#]]&,archList];
  EIList[[Flatten[Position[EIList, 0]]]] = 1; (*0/0を1とするため*)
  coList = IList/EIList  (*{各モジュールの凝集度リスト,システム凝集度}*)
  ]
```

シミュレーション結果

- 条件
  - Nf : 関数の数: 100
  - Nr : 関連の数 (callの数): Nf ×2
  - Na: モジュール数: 4
- 分かったこと
  - でたらめにやっても凝集度は0.3ぐらいになる.
  - 0.7から0.8ぐらいが最適
- 実際のシステムも人間が判断して良いと思われるモノは0.7ぐらい
- 0.2ぐらいのシステムもあった.
  - ランダムよりも悪い構造

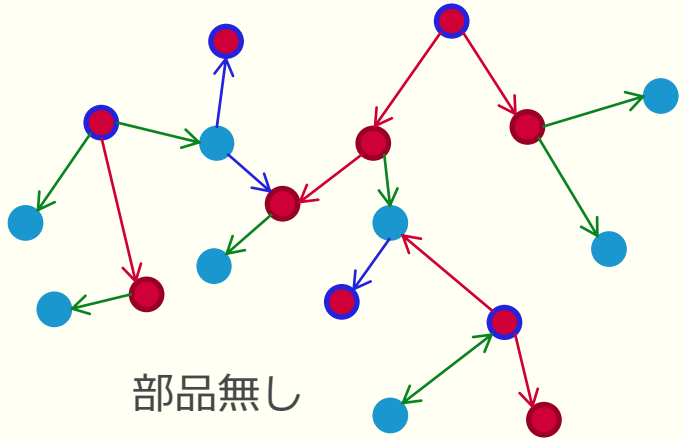




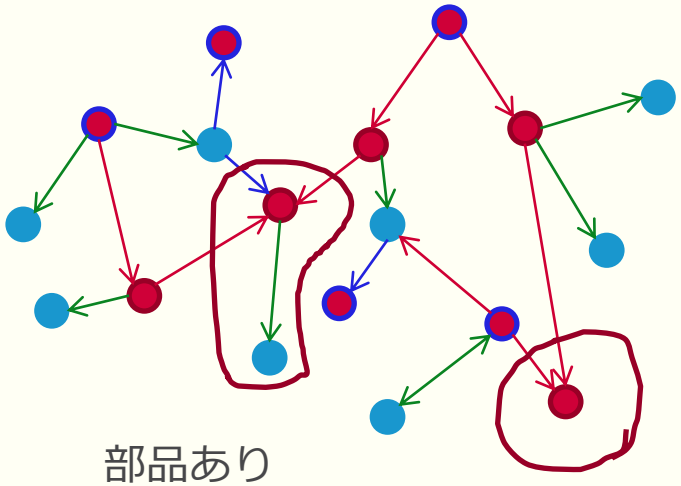


その他

- Mathematicaを利用すること(グラフ処理)で可能になったこと
  - 部品の切り出し
- ソフトウェア部品とは
  - 他に依存しない部分
  - 複数から呼ばれる
- 評価
  - 部品の多いソフトは良いソフト



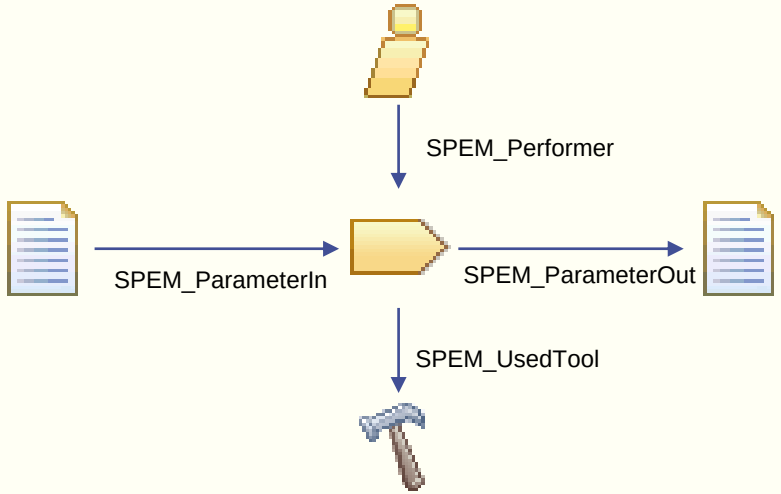
部品無し



部品あり

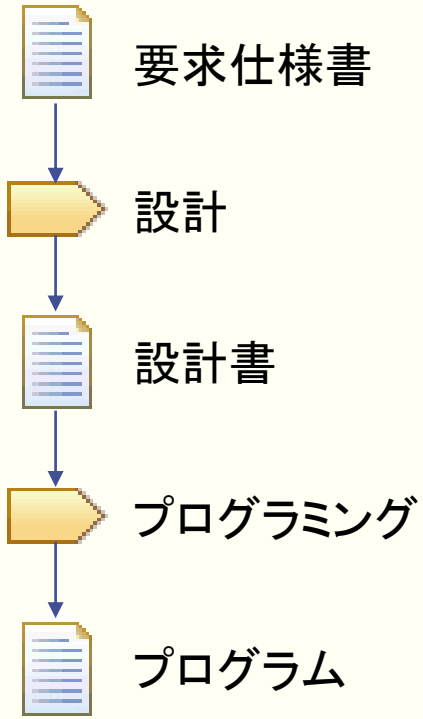
3. 開発プロセスの検証

- 開発プロセスについて
  - 何時，誰が，何をするか定義したもの
  - Role, Task, Workproduct, Toolで構成される
  - 開発プロセスの定義方法としてOMG(Object Management Group)で定義したSoftware Process Engineering Metamodel (以下、SPEM)がある



Software Process Engineering Metamodel

開発プロセスの例

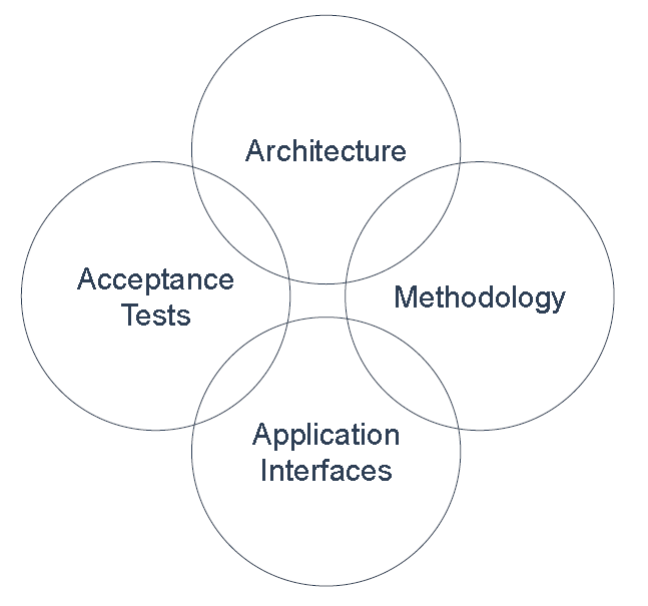


Software Process Engineering Model

- 簡単に書くとこんな感じ
  - 実際はもっと詳細に定義する.
- 詳細化したとき問題になるのは
  - 全部繋がっているか
  - ループは無いか

AUTOSARと言うものがある

- AUTOSAR = **AUT**omotive **O**pen **S**ystem **Ar**chitecture
- 目的
  - 車載システムアーキテクチャの複雑さ管理の改善
  - ソフトウェアモジュールの再利用と交換性の向上
- 標準化されている領域
  - アーキテクチャ
  - **開発プロセス(Methodology)**
  - アプリケーションインターフェース
  - テストケース

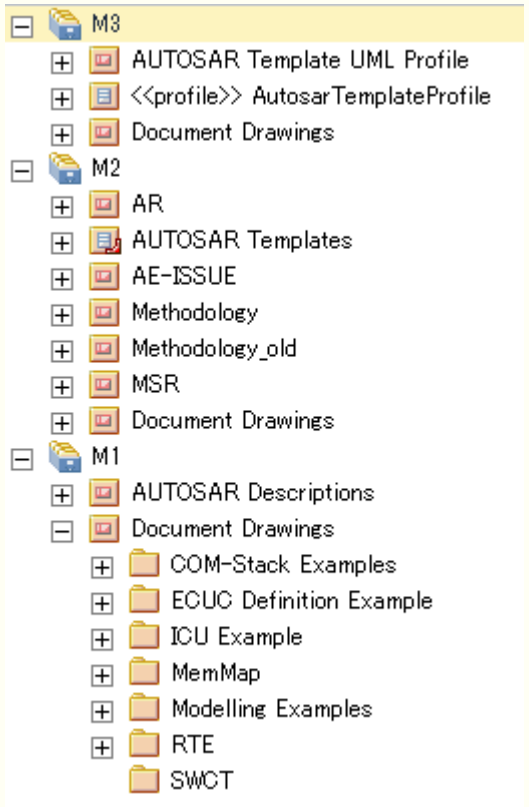


AUTOSARの実態はpdfの塊

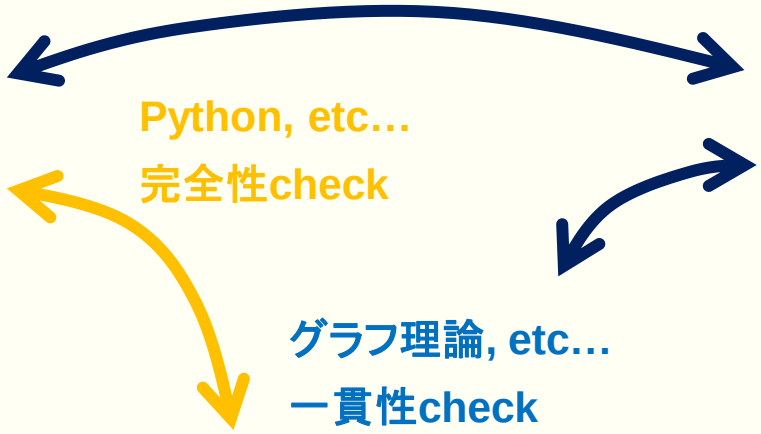
- 仕様は時間と共に拡大
- 取扱が困難
- しかし、有用な情報も多い

- 開発プロセスについて言うと，各社独自の開発プロセスとAUTOSAR開発プロセスとを統合する必要がある.

アシストするツールが必要



UMLモデル

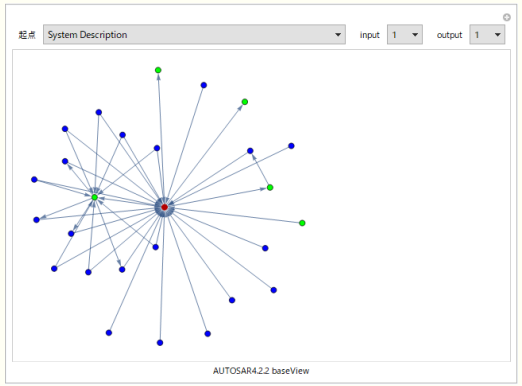


興味のある部分を取り出す

Pdfドキュメント

どうやって作ったのか

- AUTOSARが公開しているMethodology情報を取り出して有向グラフをMathematica上に構築する.
  - UMLモデルからexportしてフォーマット変換する.
- Manipulate関数を使って動くnotebookを作る.
- 動作確認が済んだらコード部分を削除して保存する.
- 保存したnotebookをWolfram Playerで利用する.




```
Manipulate[
  HighlightGraph[Subgraph[m2G,Union[VertexInComponent[m2G, {point},Round[in]],VertexOutComponent[m2G,
{point},Round[out]]], VertexStyle->vtxstyle, VertexLabels->Placed["Name",Tooltip]],point],
  Row[{Control[{{point,"System Extract","起点"}, Sort[VertexList[m2G]],ControlType-
>PopupMenu}],Control[{{in,1,input}, Range[0,10]}],Control[{{out,1,output},Range[0,10]}]},Spacer[10]],

Initialization:>(
vtx ={"A2L File","Abstract System Description", ...};
edgs  ={"A2L File"¥[DirectedEdge]"ECU Software Delivered", ... };
vtxstyle={"Basic Software Designer"->RGBColor[1, 0, 0], ...};
style = Association[vtxstyle];
m2G=Graph[vtx,edgs,VertexStyle->vtxstyle,VertexLabels->Placed["Name",Tooltip]];
),
FrameLabel->"AUTOSAR4.2.2 baseView"]
```

ここまでのまとめ

- アーキテクチャと開発プロセスにおけるMathematica利用ついて説明した.
  - 有向グラフとして表現することでMathematicaを使って分析や検証ができる.
- Manipulate関数とPlayerで配布可能なツールとして，分析した結果を利用できる.

4. AutoEncoderによる自動車へのサイバー攻撃検知

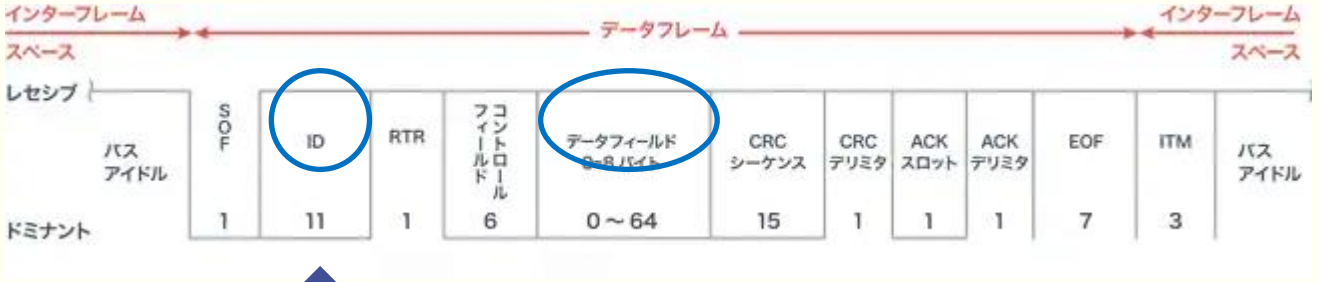
- AutoEncoder
  - ニューラルネットの応用形態で対象の特徴抽出が可能
  - 車載ネットワークの通信状態の特徴を抽出し，普段と違う(サイバー攻撃あり)を検出する.
  - また逆に，セキュリティ検証用の攻撃データを生成する.
- 車載ネットワーク
  - 自動車の中には複数のコンピュータがあり相互に通信している.
- サイバー攻撃
  - そこに外部から侵入して，異常なパケットを流すことで制御の乗っ取りが発生する.

https://wired.jp/2015/07/23/connected-car-bug/

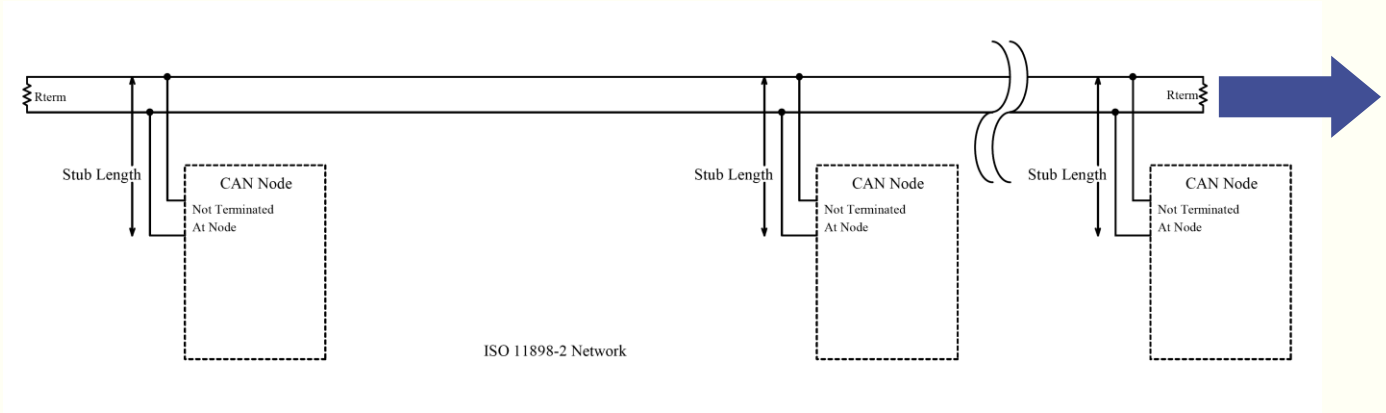
C. Miller, C. Valasek,“Remote Exploitation of an Unaltered Passenger Vehicle ”, Defcon, 2015
Jeep をリモートから乗っ取り可能という脅威の実証

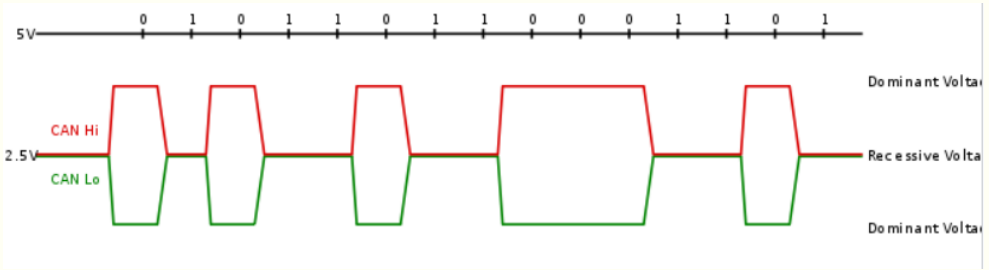
車載ネットワーク: Controller Area Network (CAN)

- Jeepの事例ではCANに不正なパケットを流されたことで乗っ取りが発生した.
- パケット(データフレーム)
  - ID : 宛先
  - データフィールド : 制御パラメータ



意味のあるまとまりをフレームと呼ぶ.



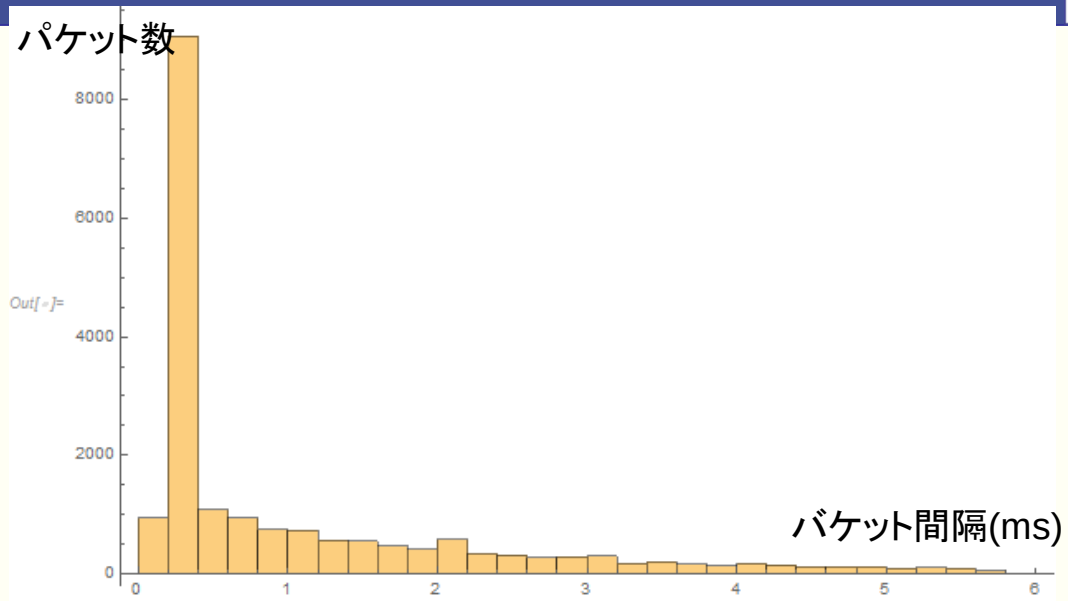


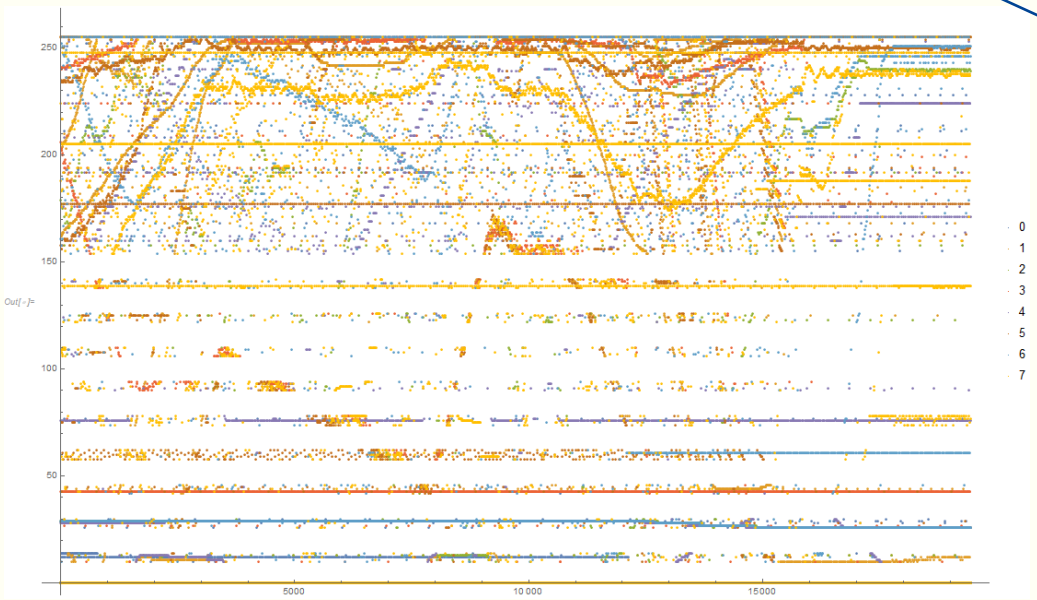
2線間の電位差で1と0を表現する

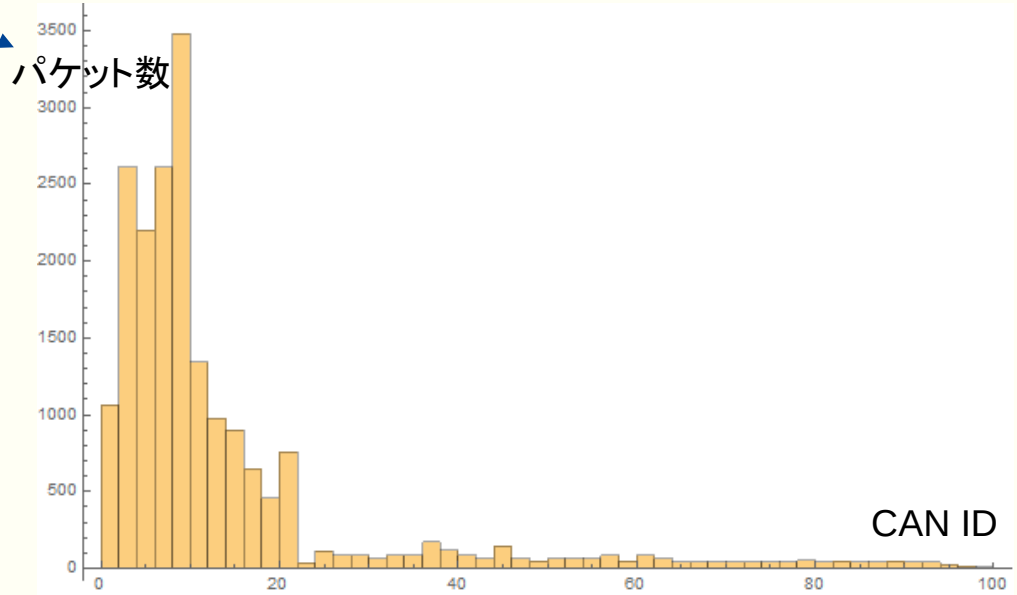
実際の車で流れるパケット

- 走行中から18秒で停止の場合の20,000パケット

1 : 時間データ，パケット間の間隔 (ms)
2 : CAN IDのコード化したもの
3 - 10: データフィールド部分の符号無し10進数

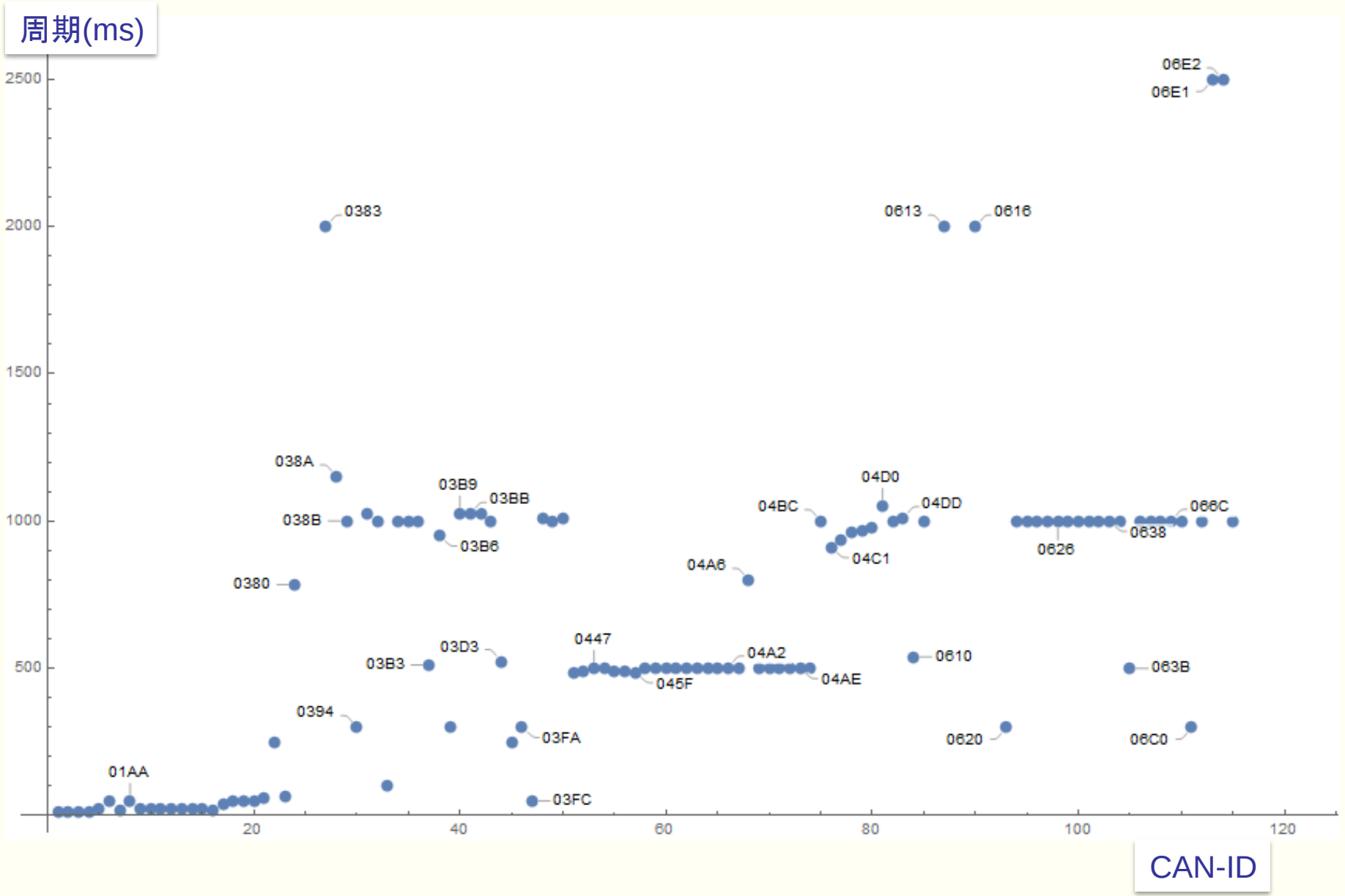






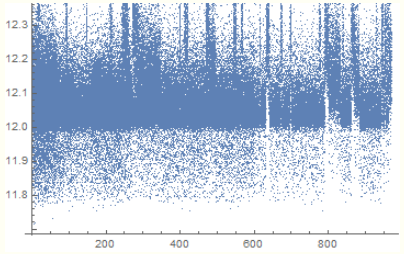
CAN-IDと送信周期

- 平均送信周期

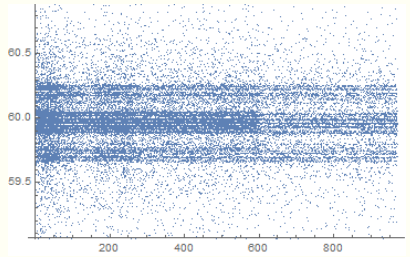


周期パターン

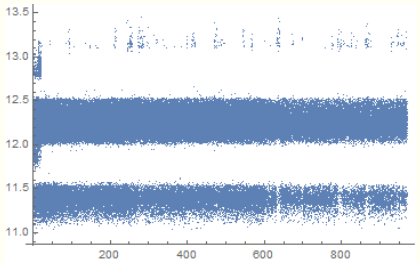
- 全てのシナリオを重ね合わせた周期パターン
  - 縦軸:周期
  - 横軸:シナリオ時間
- 全部で190万バケット
- シナリオ
  - エンジン停止-始動操作-ウインカー操作
  - アイドリング中
  - 走行中から停止
  - 市街地走行
  - エンジン始動-停止操作
  - アイドリング中-パッシング操作5回
  - アイドリング中
  - エンジン停止-始動操作



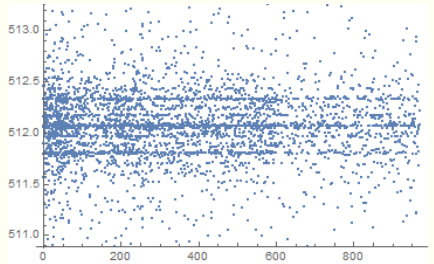
0020, 0025



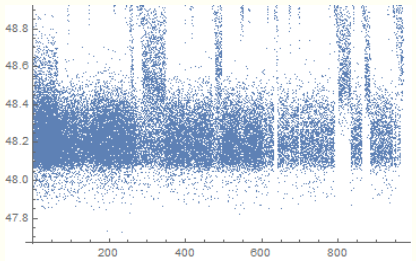
0351



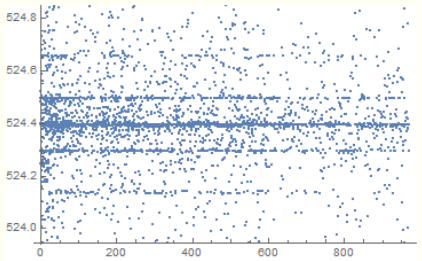
0024



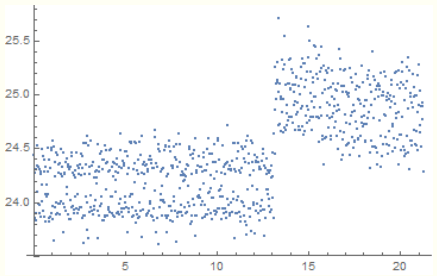
03B3



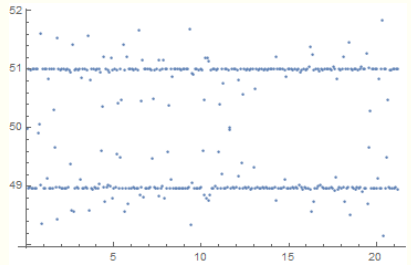
00B4,00B6,
04A0,04A1
04A2



03D3
045C



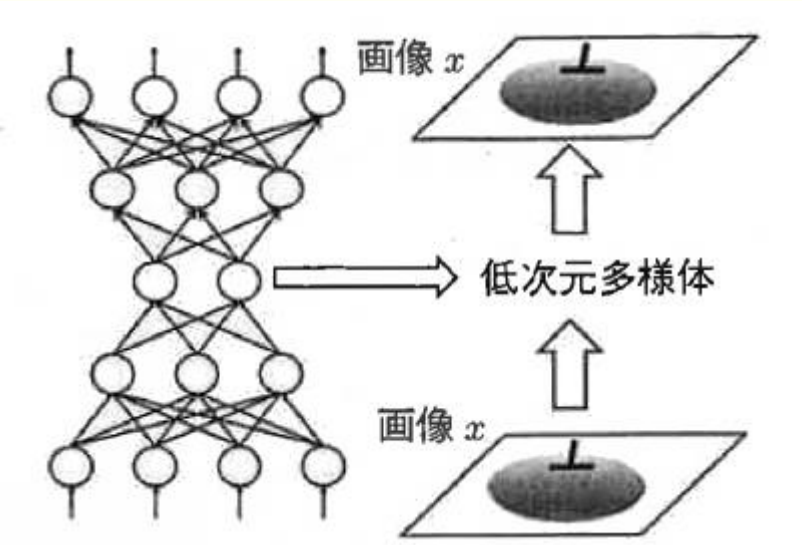
0226, 02AA



03FC

AutoEncoderについて

- ニューラルネットの応用形態で主成分解析と同様の働きをする.
- 回路網に対して入力と出力に同じデータを学習させることでデータの本質的な情報を集約して取り出すことができる.
- 教師無し学習に属する.
- 中間層のユニット数を少なくすると主成分分析の非線形化を実現できる．これは砂時計型神経回路網[1]として発明されたものであるが，この考え方はオート符号化器(オートエンコーダー)に発展して広く使われている.



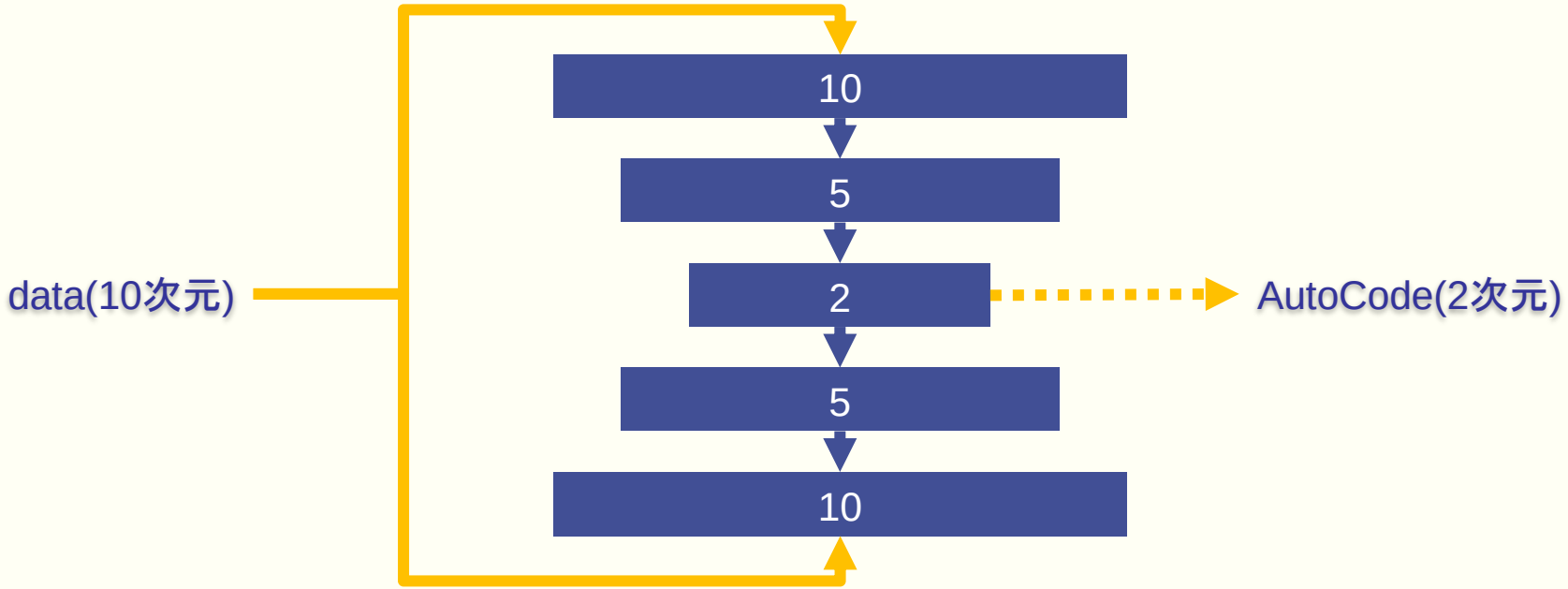
機械学習の数学入門, 数理科学, NO.662, AUGUST 2018.

[1] 入江文平他，‘多層パーセプトロンによる内部表現の獲得,’電子情報通信学会論文誌 D-II，Vol.J74-D-II，pp.1173-1178, 1990.

https://en.wikipedia.org/wiki/Autoencoder
2006年にジェフリー・ヒントンらが提案した。
これは間違い?

AutoEncoderの構築

- 10次元データを2次元に圧縮する.
  - パケット間隔，CAN-ID, データフィールド
- 定義によれば5層以上なので，線形変換層を5層用意する.



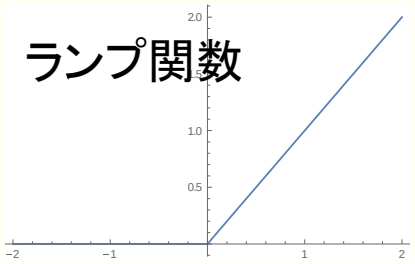
使用したAutoEncoderの構造と学習

CANデータ id
送信間隔
payload値
最大9次元

Input vector (size: 9)
1 LinearLayer vector (size: 25)
2 Ramp vector (size: 25)
3 LinearLayer vector (size: 2)
4 LinearLayer vector (size: 25)
5 Ramp vector (size: 25)
6 LinearLayer vector (size: 9)
Output vector (size: 9)

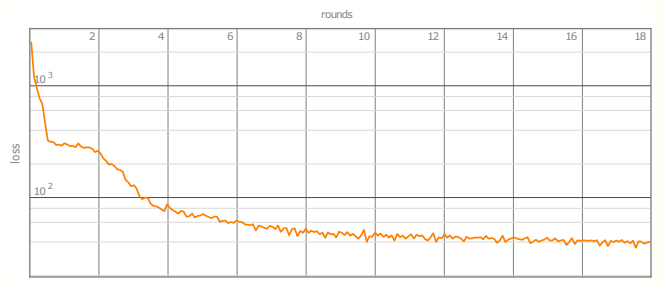
w.x+b を計算する

ランプ関数



MeanSquaredLossLayer

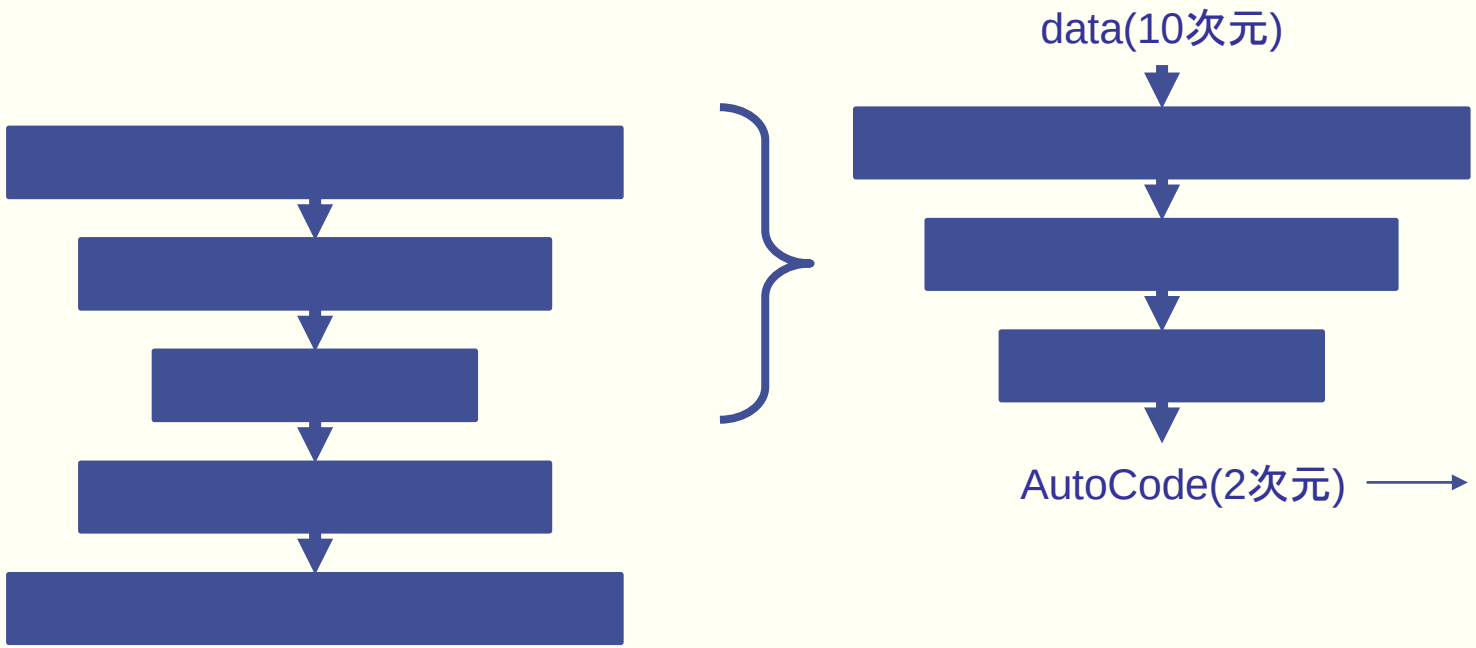
loss

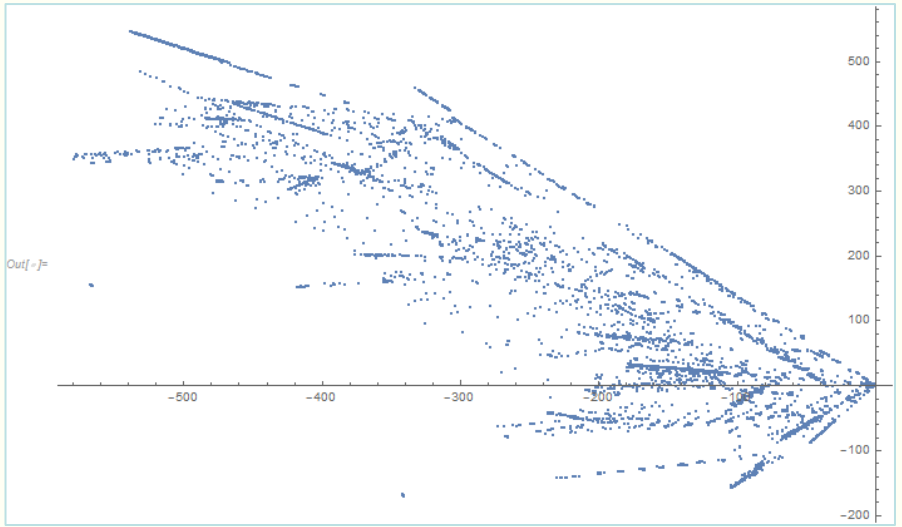


平均二乗損失を計算する損失層

AutoEncoder

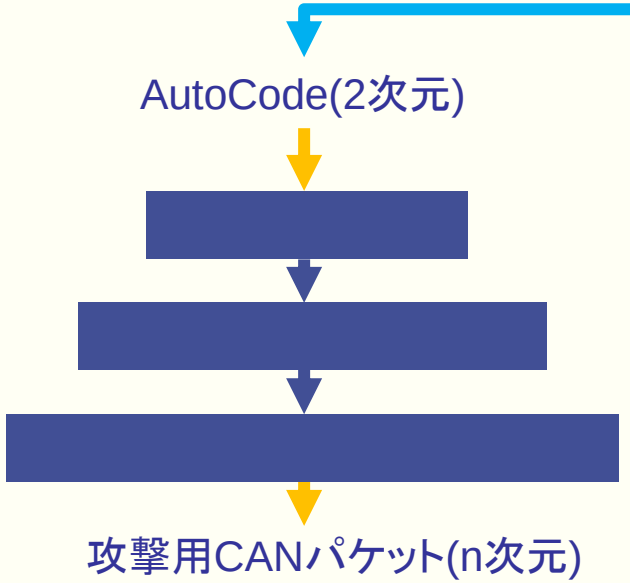
- 学習後，AutoEncoder部分を取り出して使用する．

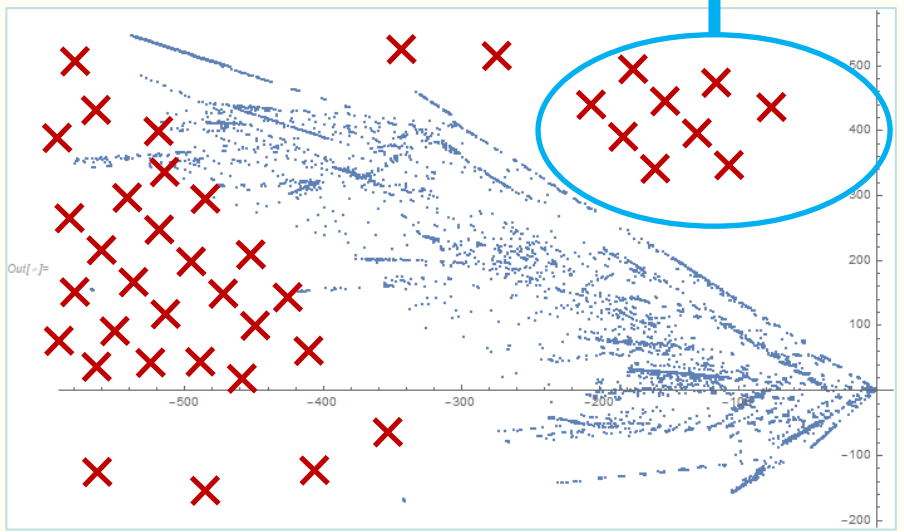




攻撃データの生成

- 訓練したネットの下半部を用いて，攻撃実験が不十分な部分の攻撃データを生成する.

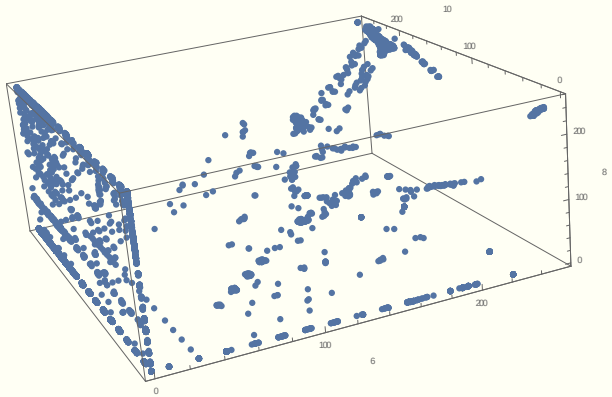




AutoEncoderの使い方

実データ空間
送信間隔
payload値
} 最大9次元

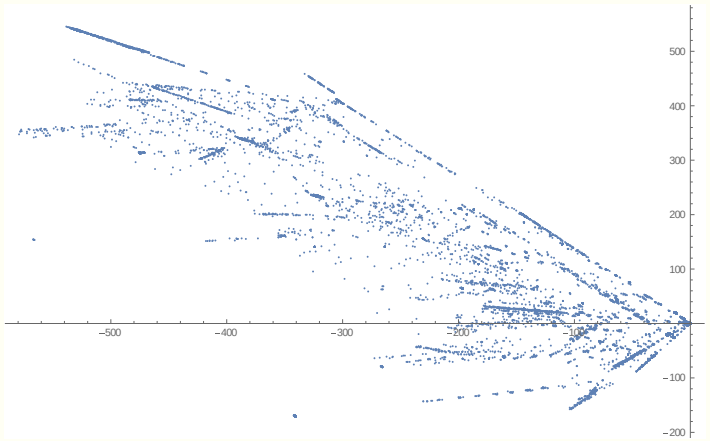
```
Input vector ( size: 9 )
1  LinearLayer   vector ( size: 25 )
2  Ramp   vector ( size: 25 )
3  LinearLayer   vector ( size: 2 )
4  LinearLayer   vector ( size: 25 )
5  Ramp   vector ( size: 25 )
6  LinearLayer   vector ( size: 9 )
   Output   vector ( size: 9 )
```



Encoder

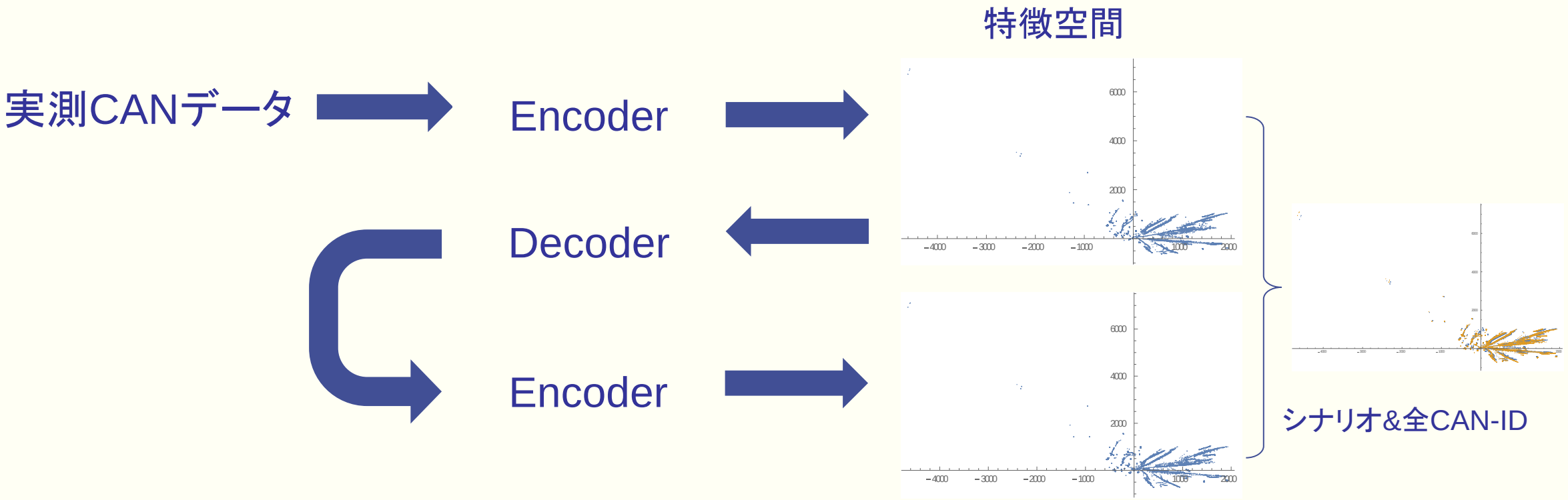
Decoder

2次元の特徴空間



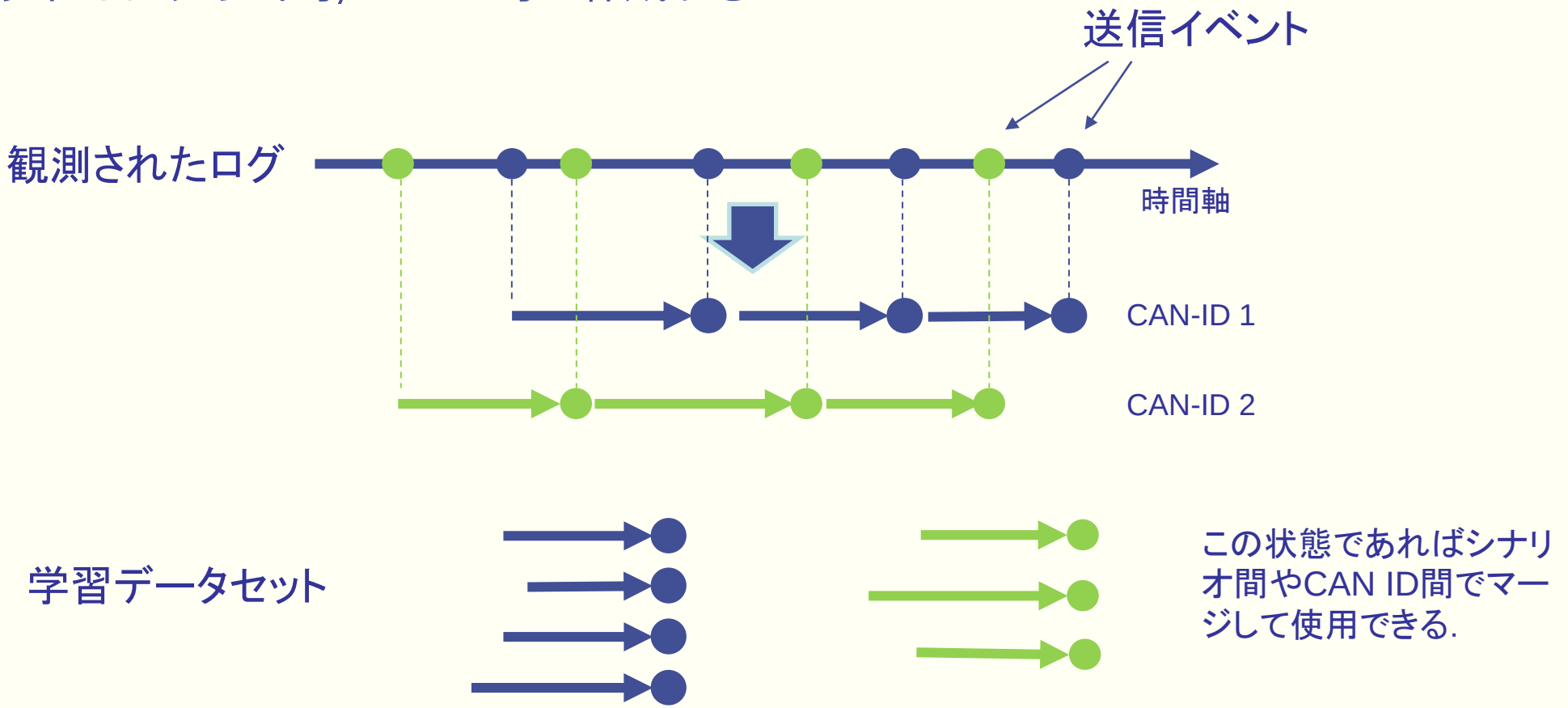
AutoEncoderの検証

- 特徴空間の再現性で評価した.



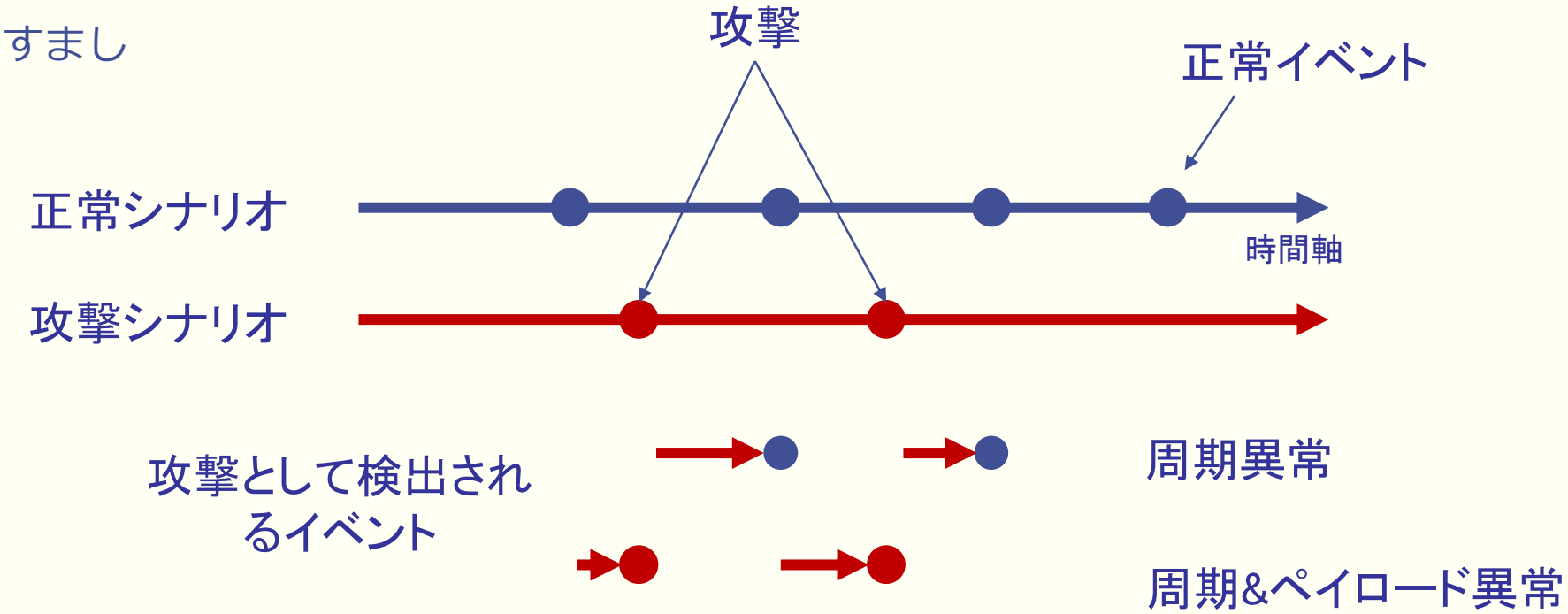
AutoEncoder用データの作成

- CAN ID毎の送信間隔とpayload値を使用する.
- データセットはシナリオ毎/CAN ID毎に作成する.



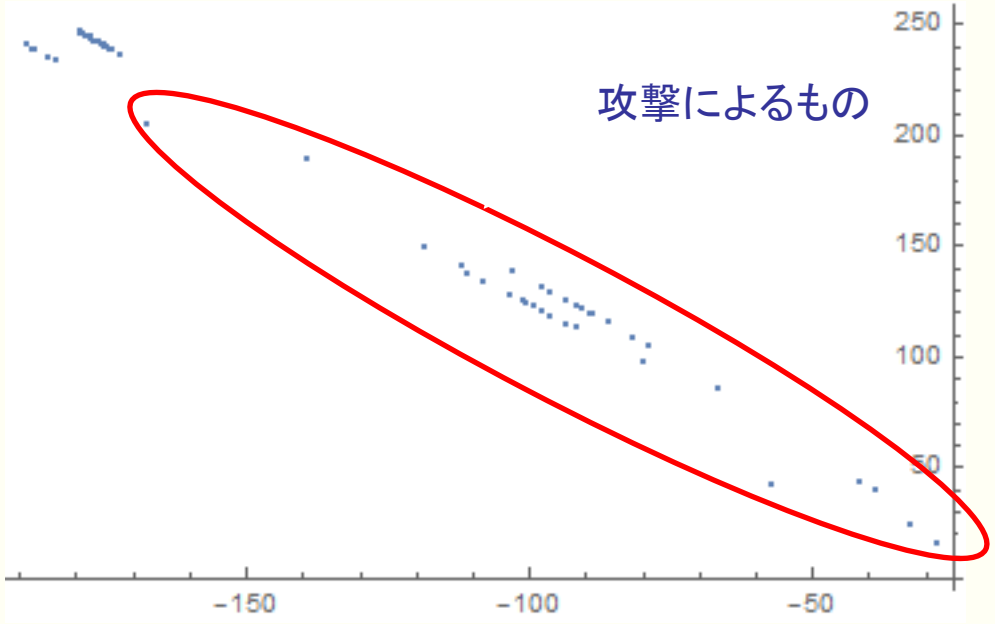
攻撃シミュレーション

- 攻撃シナリオと攻撃用データの作成
  - なりすまし



045C: 攻撃シミュレーション 1

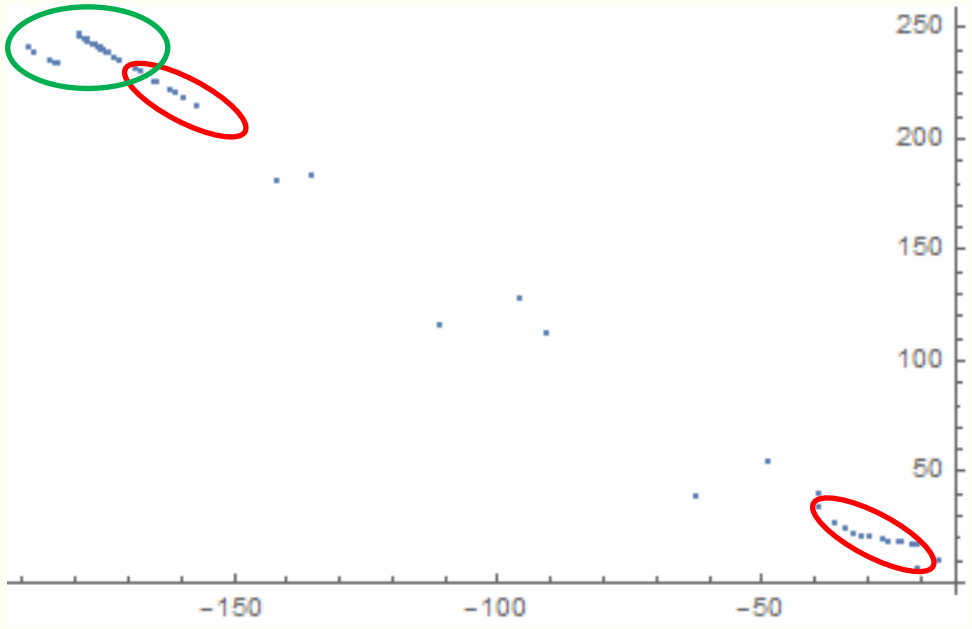
- シナリオ#9にシナリオ#5のログの一部をマージして攻撃と見なしてエンコードした.
  - かなり派手に検出できる.



045C: 攻撃シミュレーション 2

- シナリオ#9にシナリオ#5のログの一部をマージして攻撃と見なしてエンコードした.
  - マージする際に時間的にシフトして正常データに接近させたもの
    - 接近する点と離れる点がペアで出てくる.

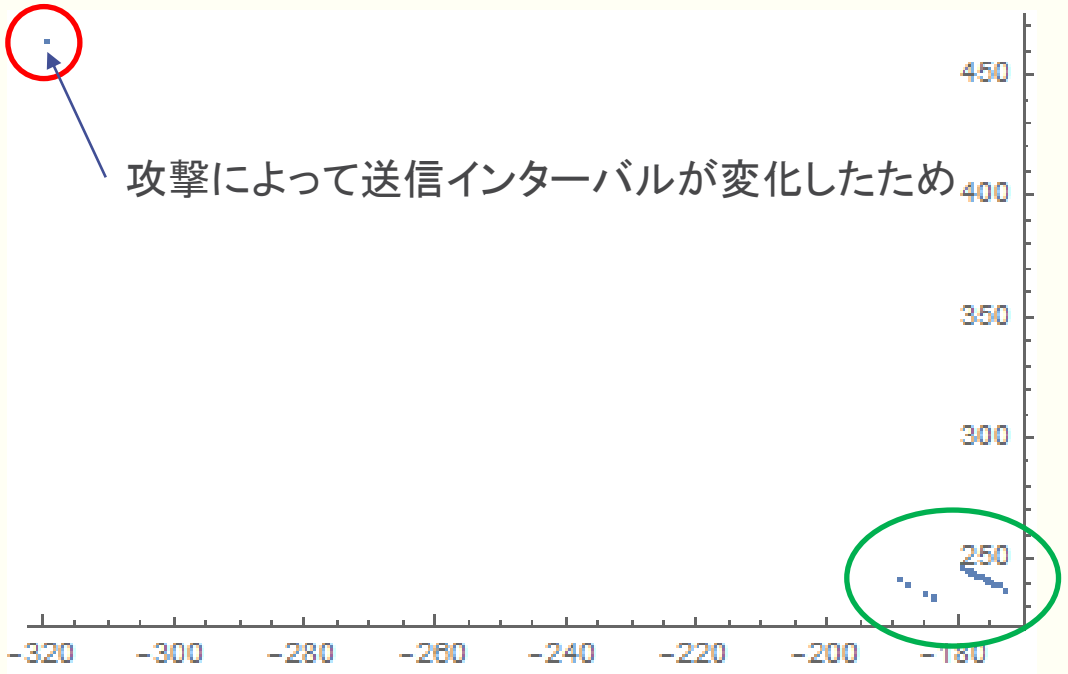
オフセット操作した場合



オフセット(+50ms)

045C: 攻撃シミュレーション 3

- シナリオ#9からデータを1つ削除した.
  - 打ち落とし攻撃

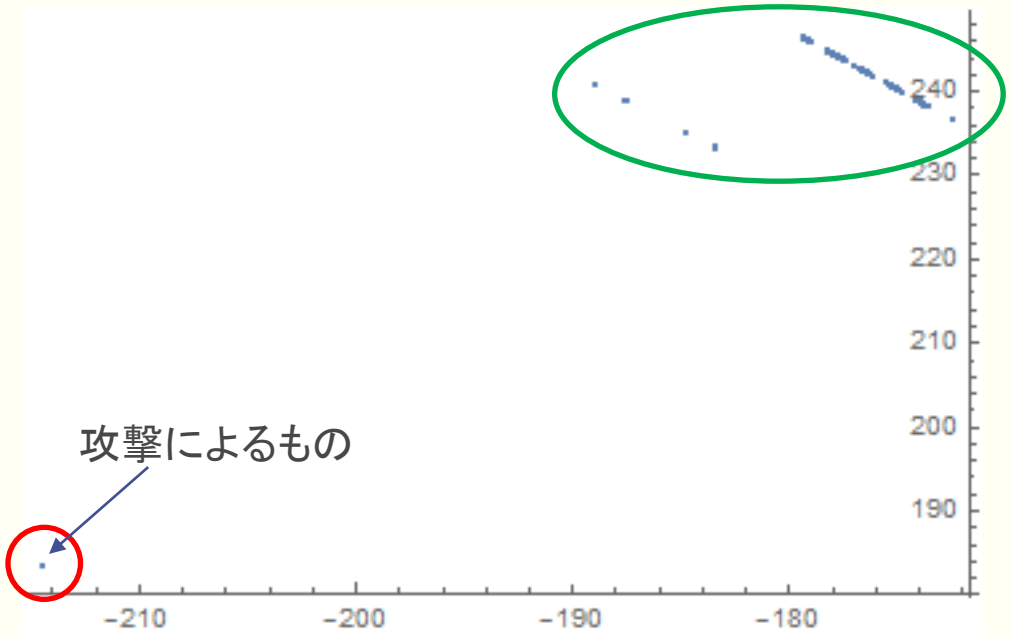


045C: 攻撃シミュレーション 4

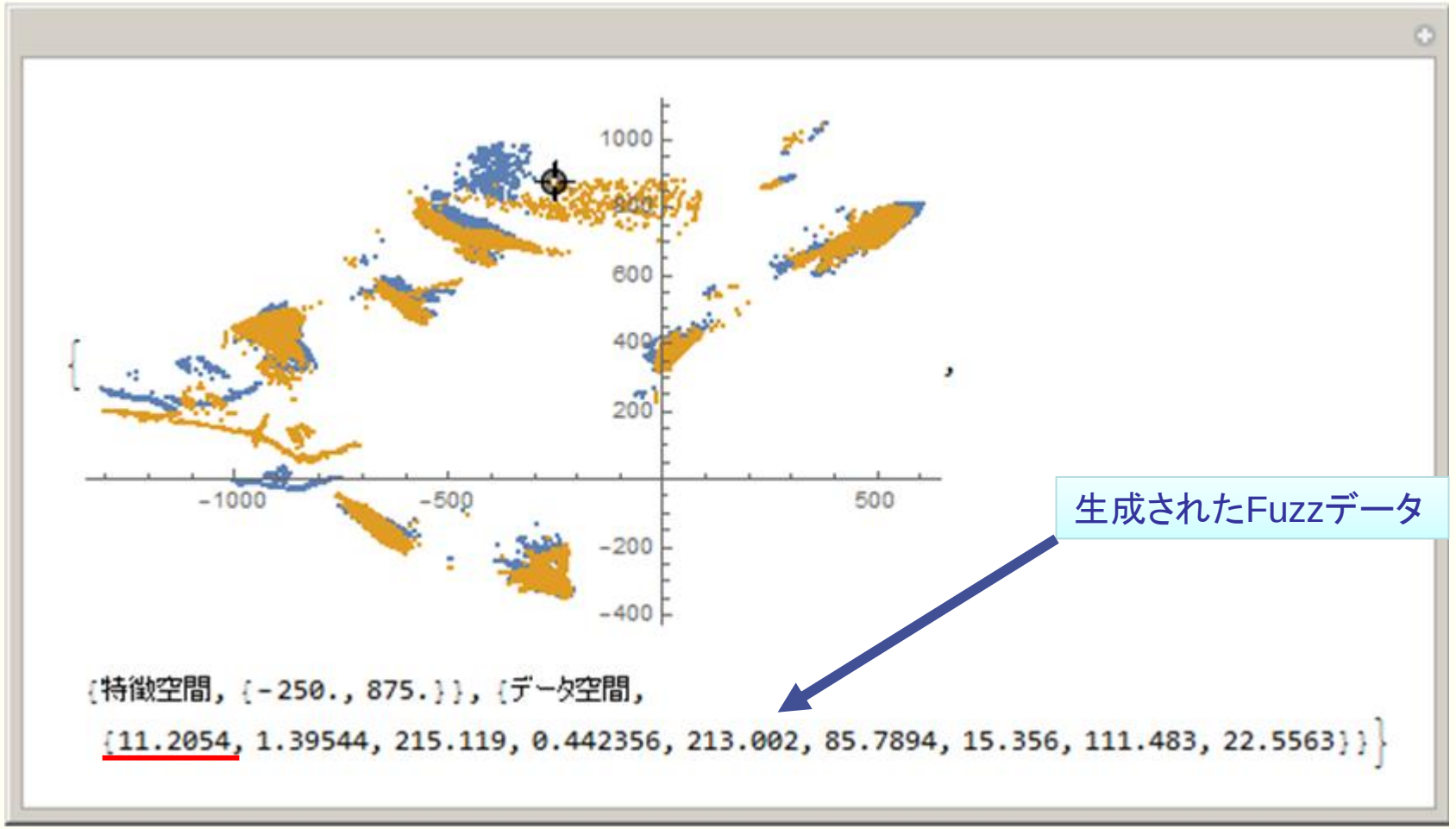
- シナリオ#9から1つのデータのペイロードを乱数で書き換えた.
  - 改竄攻撃

{95, 2, 0, 32, 7, 0, 0, 0}

{216, 59, 156, 13, 212, 165, 51, 131}



Fuzzデータ生成簡易ツール



AutoEncoderまとめと今後の予定

- 学習時間はほとんど掛からなかった．数分レベル．
- ログで学習した検出器の検出力はかなり高い．
- 取りあえず新しいタイプのファジングツール(偽データを作るツール)を作った．

- 今後の予定
  - シーケンスデータを扱えるようにする．
    - AutoEncoderにLSTMを取り込む?

全体のまとめと要望

- Mathematica
  - 有向グラフ関連の関数をソフトウェア開発に使う方法を説明した.
  - 表現力が高い，使用した図は全てMathematicaで生成したものである.
  - 配布用の環境がそろっている.
- 要望
  - プレゼン動画を作る機能を充実してはどうか?
  - 群論関係の充実
    - 置換群を作ってそれがどの有限群に属するか判定する関数が欲しい.

Thank you for listening!

https://www.embresearch.com