

全脳アーキテクチャ・
ハッカソン 2018

成果発表

チーム WIP

太田 晋

2018/10/08

目次

- モジュール全体像
- 実装上のトピック
- デモ
- 考察

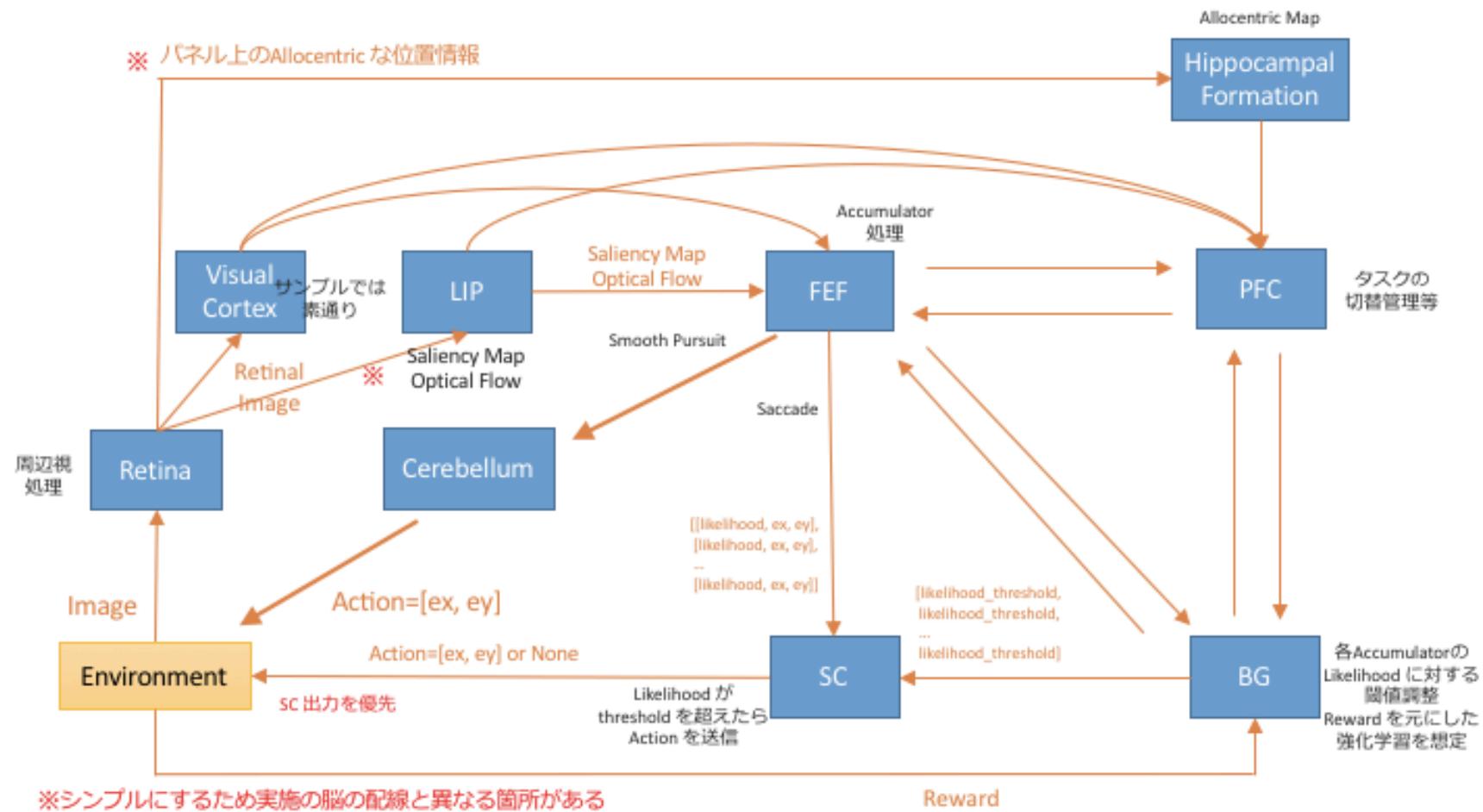
テーマ

- 眼球運動に関する6つの視覚タスクを解く单一の計算モデルを作成
- 並列性・積分・時間軸・記憶等に着目

設計方針

- 並列性
 - モジュール間の結びつきをなるべく疎に
 - 階層的・並列的アーキテクチャ → サブサンプションアーキテクチャ
 - モジュールをまたいだバックプロパゲーションは行わない
 - → 非同期分散計算をやりやすくする
- 積分
 - アキュムレータによる時間積分
- 時間軸
 - ステップ数(時間軸)と状態をもとにフェーズを定義
 - → モジュール毎のサブタスク → サブゴールの学習
- 記憶
 - ワーキングメモリを使って過去の状態を保存
 - 現在の状態と比較して変化を検出

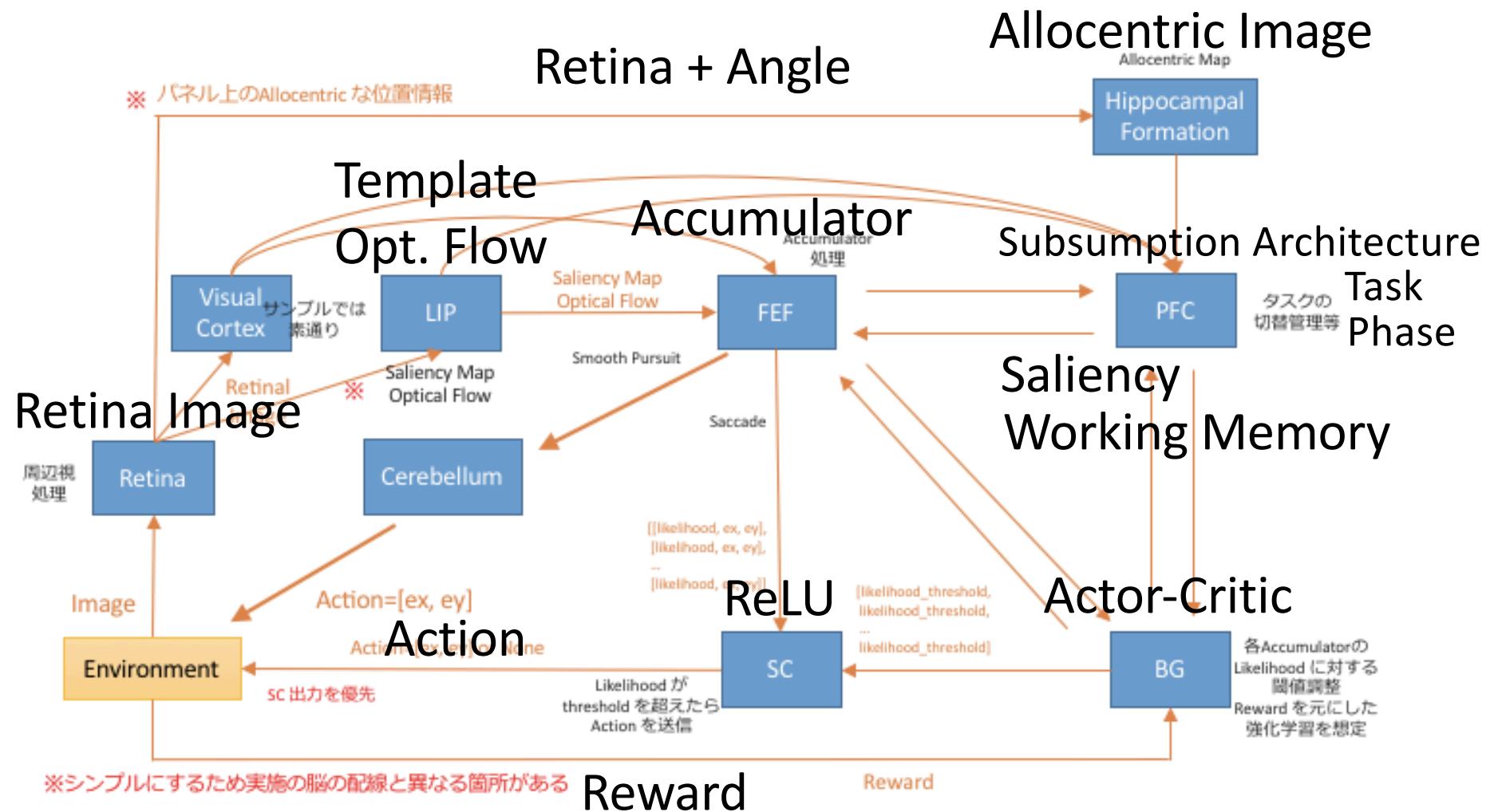
モジュールの全体像



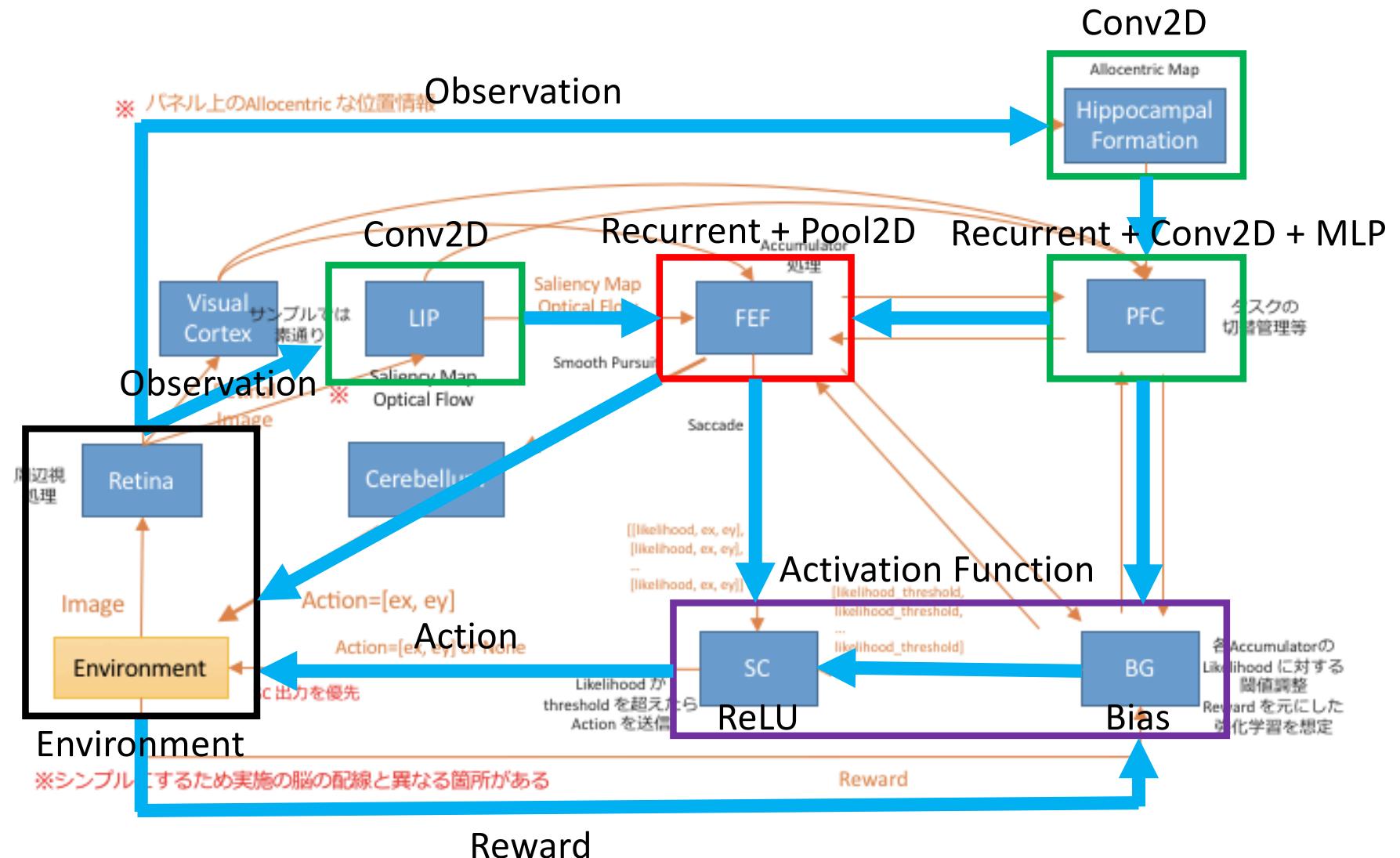
・各タスクが必要とする機能

タスク名	探索	Saliency Map	Template Matching	Working Memory	Optical Flow	備考
Point To Target	○	○	○			探索しながらテンプレートマッチング
Change Detection				◎		ワーキングメモリに変更前の状態を保存
Odd One Out	◎	◎		○		全探索するハメになるとステップ増
Visual Search	○	○	○			一定時間探索後に判断
Multiple Object Tracking	○		○			フェーズを3つに分けて、緑丸/青丸/黒丸のテンプレートマッチ
Random Dot Motion Discrimination					◎	粒子の動きをOpt. Flowで検知

各機能の割り振り



各モジュールを NN で考えると...



実装上のトピック

BG を強化学習ベースにする

- ルールベース(if-then)を強化学習ベースに置き換える
 - ルール: 入力変数7, 出力変数3, 50行, ifネスト最大4段
- 6つのタスクを单一モデルで強化学習
 - Actor-Critic + MLP
- 入力: 7次元 (PFC から4, FEF から3)
 - task, phase, internal_phase, target, direction_index, max_template_likelihood, max_change_likelihood
- 出力: Actor 3次元 (離散化して195次元) + Critic 1次元
 - (3 phases) * (13 targets) * (5 likelihood coeffs)

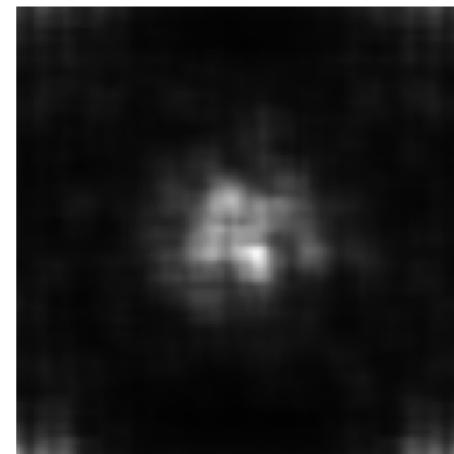
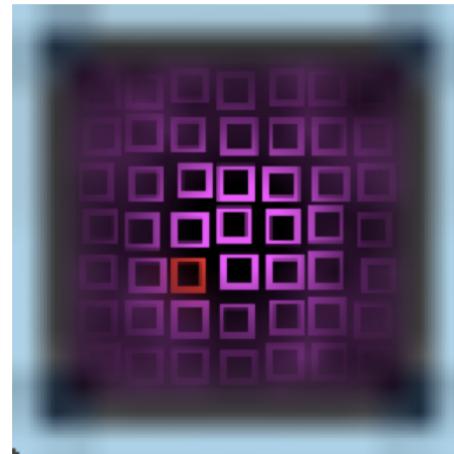
```
class BGNet(nn.Module):  
    def __init__(self):  
        super(BGNet, self).__init__()  
        self.affine1 = nn.Linear(7, 128)  
        self.action_head = nn.Linear(128, 195)  
        self.value_head = nn.Linear(128, 1)  
  
    def forward(self, x):  
        x = F.relu(self.affine1(x))  
        action_scores = self.action_head(x)  
        state_values = self.value_head(x)  
        return F.softmax(action_scores, dim=-1), state_values
```

実際にBGを強化学習させてみると…

- 学習
 - oculomotor の枠組み内で学習
 - CPU マシン, 1時間, 10800 ステップ, 報酬3
 - → ハッカソン期間中に学習は無理
 - 2層の MLP (入力7, 出力196)なので学習は比較的軽いはず
 - 後ほど見せる評価結果はルールベース
- ボトルネック
 - Retina 画像生成
 - GPU で Retina なし (oculovenv 単体) 250 FPS
 - GPU で Retina あり (oculovenv 単体) 150 FPS
 - サリエンシーマップを生成する際の FFT と逆FFT
 - → CPU に律速されている可能性

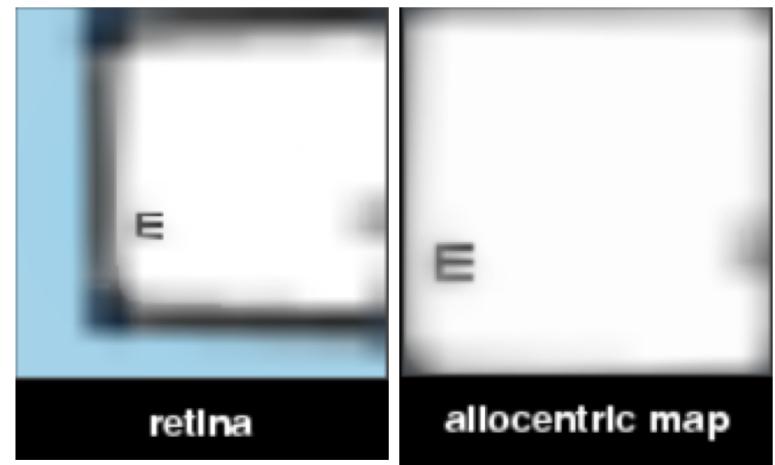
サリエンシーマップの問題点

- 現在のサリエンシーマップ実装
 - フーリエ変換 → データをスムースにして差分をとる
→ 逆フーリエ変換
- 問題点
 - パネルの枠に反応しそぎ
 - 大域的な周期性が検出できない
 - 中心はクッキリ, 周辺はぼけている

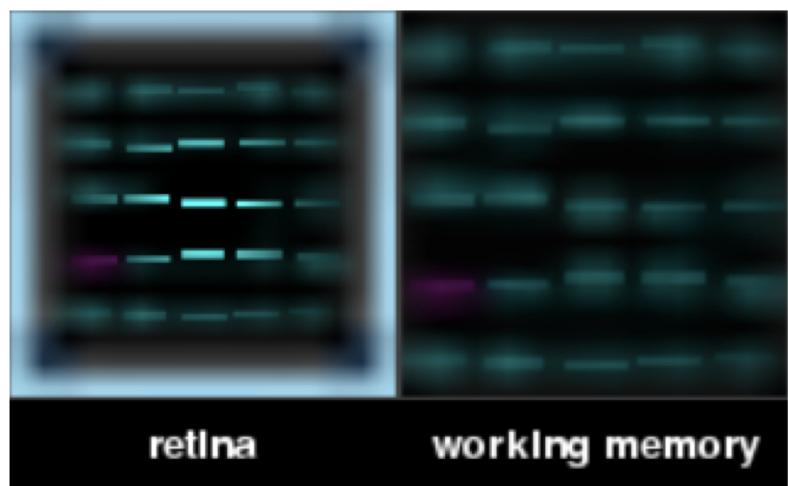


サリエンシーマップのハック

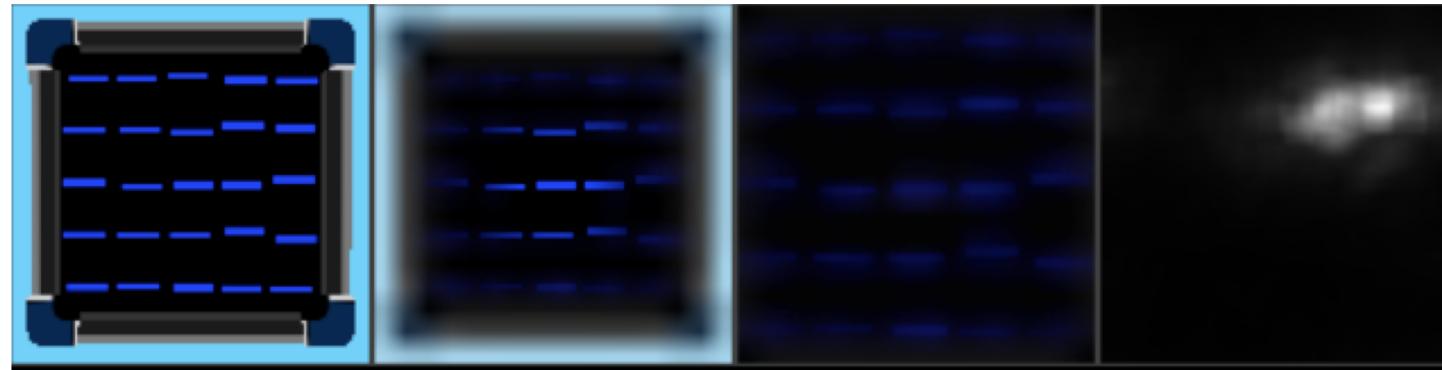
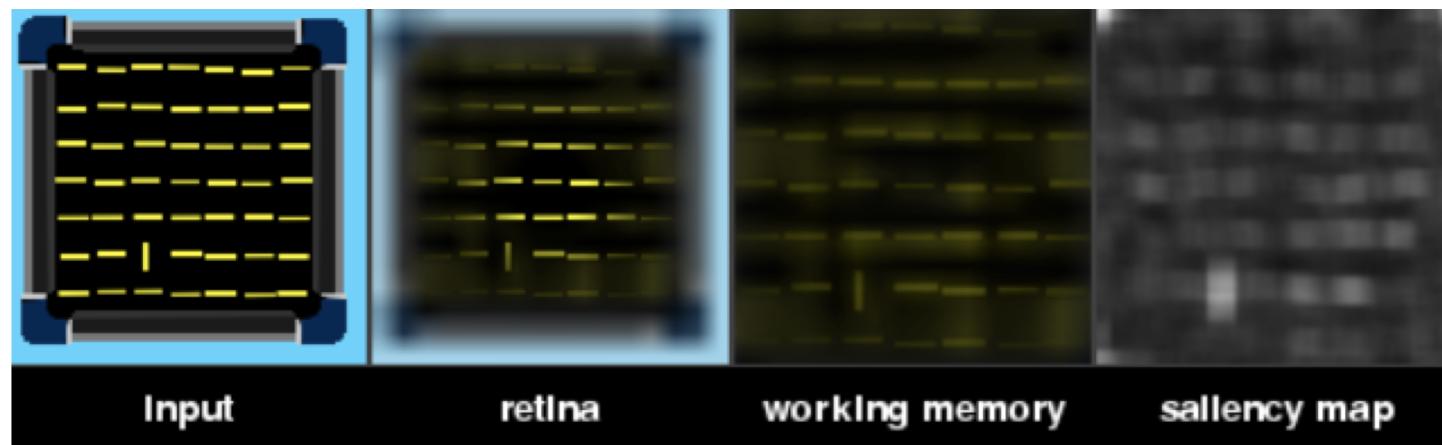
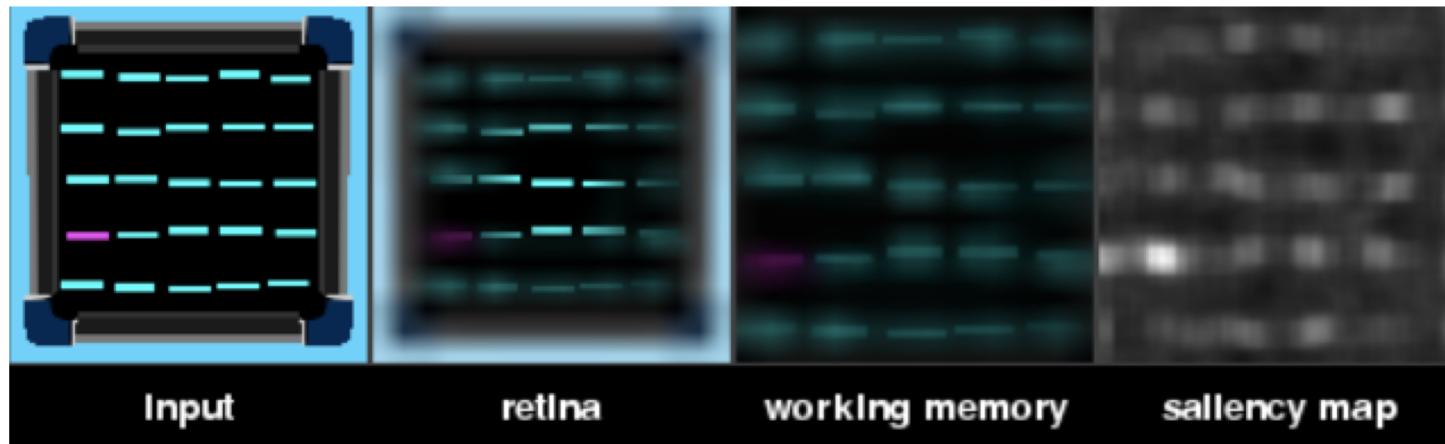
- 枠を消す
 - Allocentric Image を利用



- 中心をあえてぼかす
 - 全体が均一にぼけていれば周期性を検出可能
 - Retina 画像の生成に使われているボケ分布の逆関数で中心をぼかす



Odd One Out 仲間はずれを探すタスク



右上のバー
が動いている

モデ^ア

評価結果

評価スクリプトを5回実行して平均と標準偏差を計算

BG はルールベース

タスク	難易度	平均精度	標準偏差	平均ステップ	標準偏差
Point To Target	0	0.969	0.019	23.228	0.395
	1	0.776	0.090	27.382	1.339
	2	0.669	0.109	29.686	1.343
Change Detection	0	1.000	0.000	28.530	0.076
	2	0.991	0.019	28.592	0.071
	4	1.000	0.000	28.508	0.087
Odd One Out	-1	1.000	0.000	99.252	12.281
Visual Search	0	0.931	0.044	55.388	6.001
	2	0.870	0.105	56.170	3.216
	5	0.916	0.054	55.612	3.362
Multiple Object Tracking	0	0.474	0.103	50.346	5.708
	2	0.439	0.114	48.610	5.370
	5	0.541	0.050	46.904	4.990
Random Dot Motion Discrimination	0	1.000	0.000	55.982	1.275
	2	0.983	0.037	56.238	1.309
	4	0.329	0.131	69.638	12.227

考察

- モジュールに機能を分ける
- タスクを時間軸と状態に沿ってフェーズに分ける
- 各モジュールをできるだけシンプルに(次元削減)
- モジュール間の結合を疎に (個別に学習)

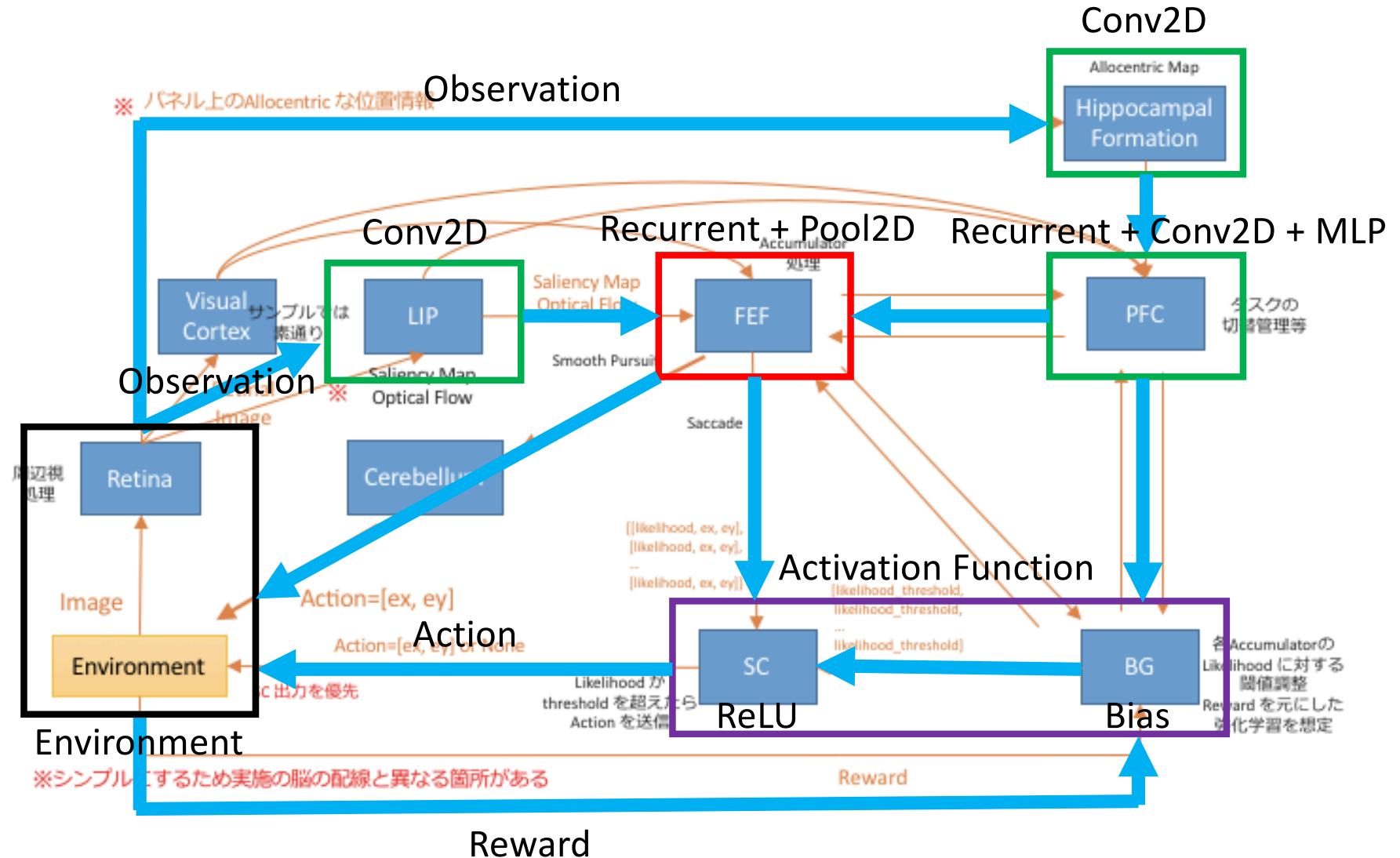


- 強化学習でなくともルール(if-then)ベースである程度解ける



- BriCA モデル(oculomotor)自体の妥当性がある程度証明された?
 - モジュールへの機能の割り振り方とデータの流れ
 - スタブ開発(モジュール毎に学習)

各モジュールを NN で考えると...



今後の課題

- BG を単一モデルで強化学習
 - Retina 周りを numpy 化すると速くなるかも
- PFC をルールベースから強化学習(or 教師あり学習)ベースに
 - 現在, タスクの検知だけは教師あり学習
- モジュールをまたいだバックプロパーション
- 未知の問題に対して, タスクをサブタスクに分解できるか?
 - 新しいフェーズを学習で獲得できるか?
 - サブゴールは何かを学習できるか?

ソースコード

- github
 - <https://github.com/susumuota/oculomotor>
 - <https://github.com/susumuota/oculoenv>