Исследование алгоритма обучения функциональных систем в стохастической среде

Сорокин А. Ю.
Руководитель: Бурцев М. С.

Кафедра математических и информационных технологий
Академический университет

2014

Нейроморфные системы

Под нейроморфными системами понимаются модели искусственных нейронных сетей, архитектура или свойства, которых основаны на особенностях реальных нейробиологических систем.

Цель и Задачи

Цель:

Исследовать модель функциональных систем на возможность обучения в стохастической среде

Задачи:

1. Реализовать библиотеку на Python
2. Воспроизвести результаты из статьи[1]
3. Исследовать обучение модели в стохастической среде

1. Komarov M.A., Osipov G.V. and Burtsev M.S. Adaptive Functional Systems: Learning with Chaos // Chaos 20(4), 045119, 2010.

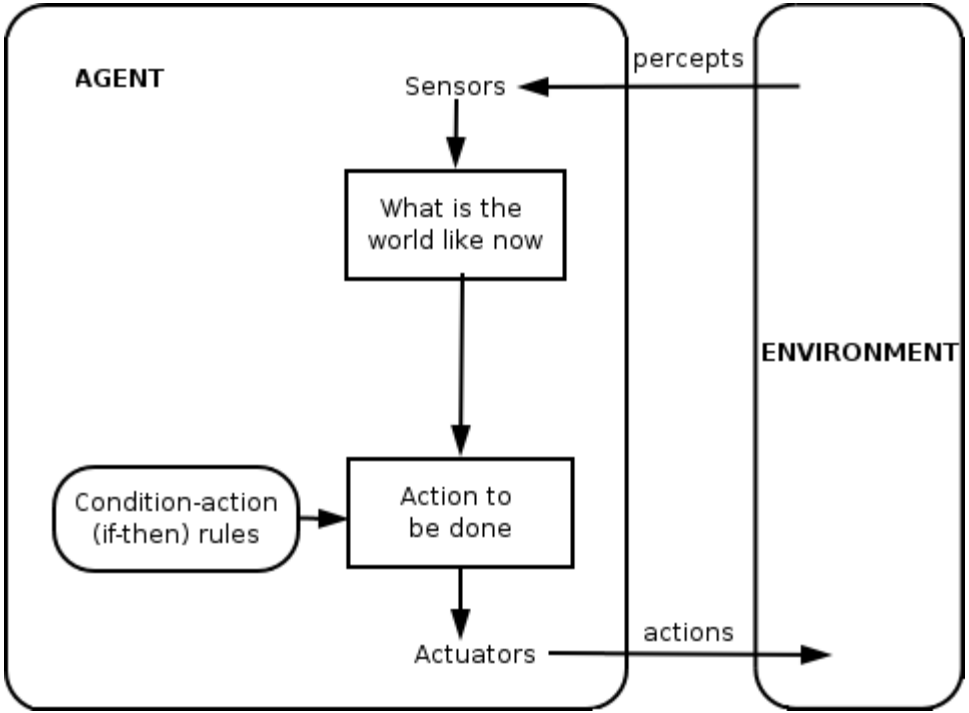
Агентный подход

Цикл взаимодействия:

1. Агент получает информацию о текущем состоянии среды и возможных вознаграждениях

2. Выбирает действие для изменения среды

3. Среда меняет свое в ответ на действие агента.



Функциональные системы

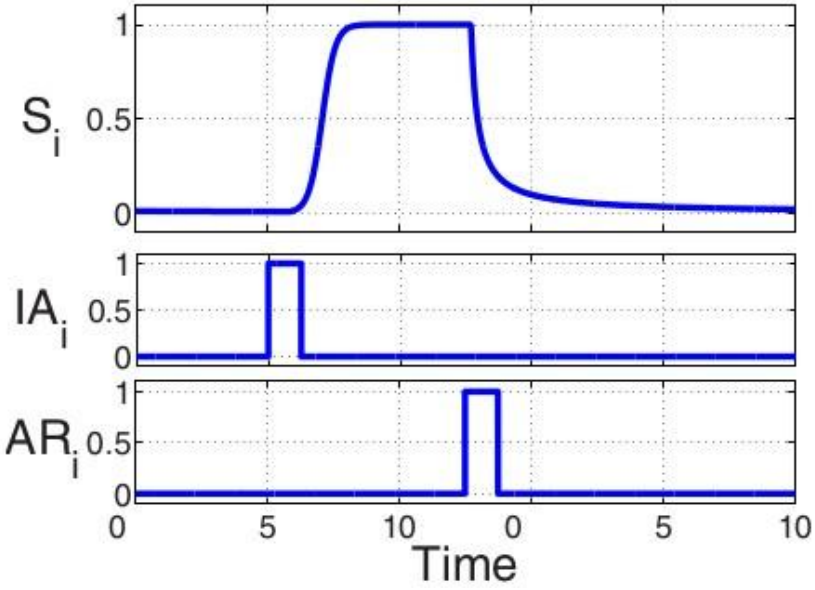
Функциональная система - динамический элемент с двуму стационарными состояниями

- Состояние активности
- Состояние покоя

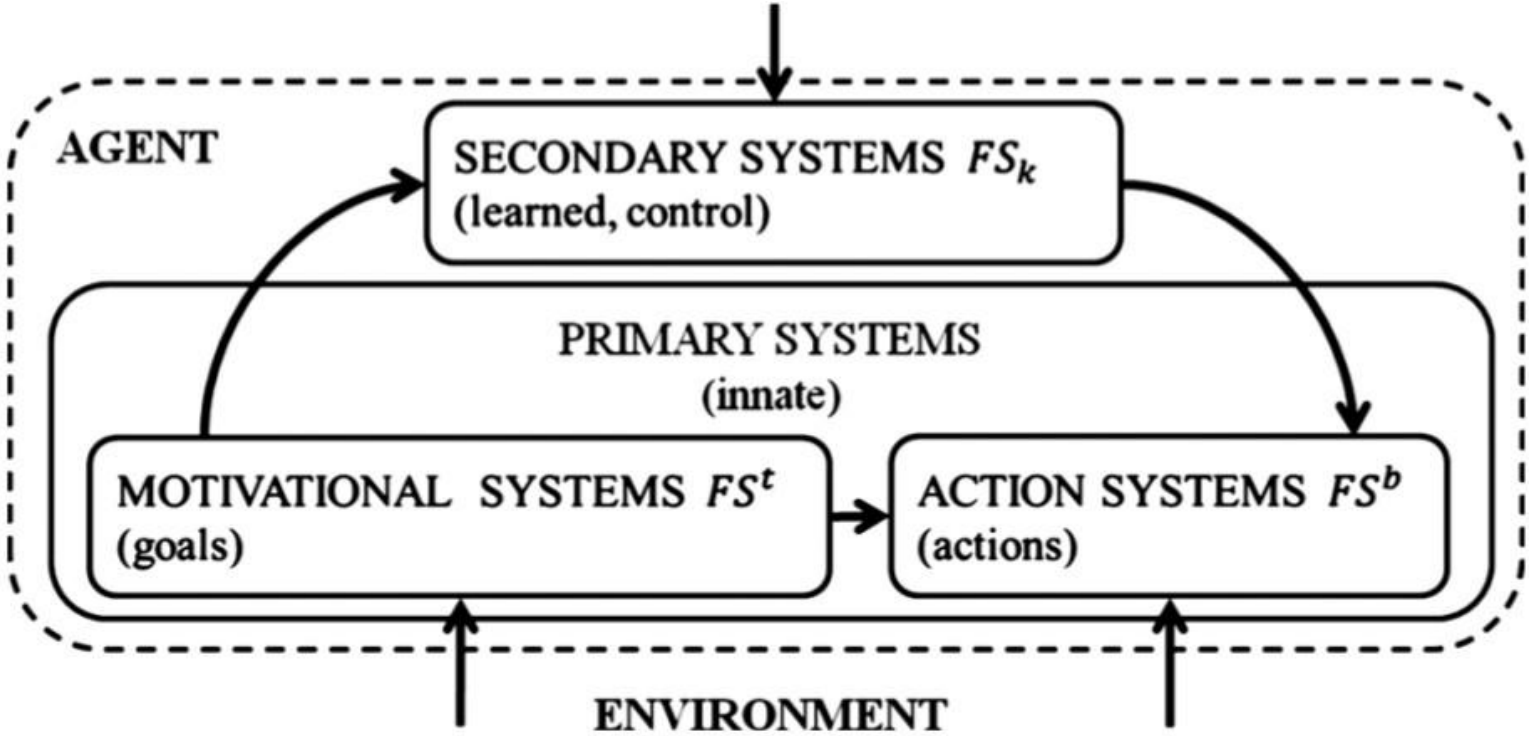
S - активность функциональной системы

IA - входная афферентация

AR - акцептор результата действия



Сеть функциональных систем



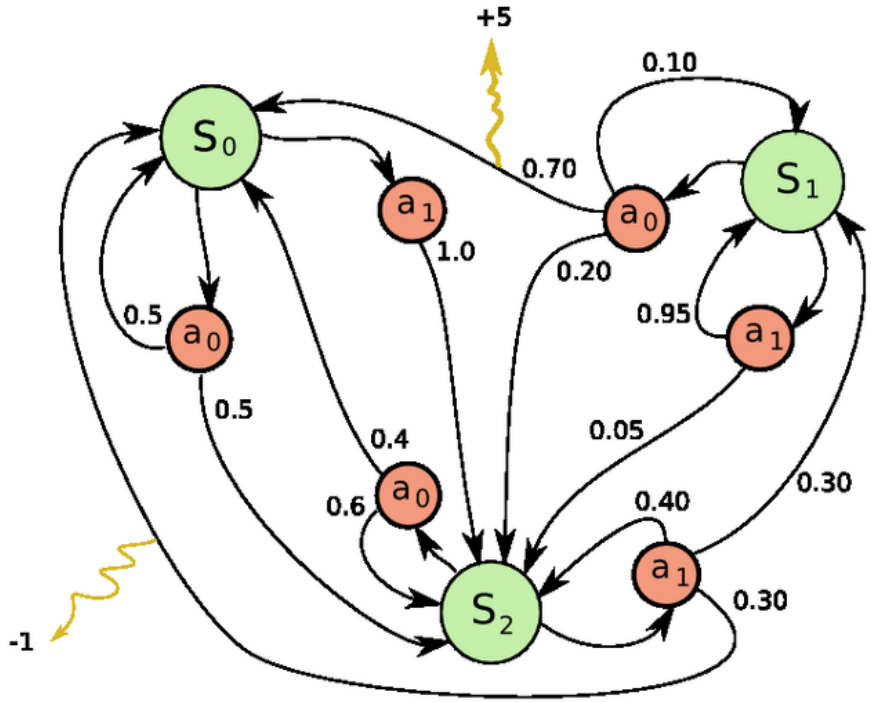
Среда

Среда - граф, где вершины соответствуют различным возможным состояниям, а исходящие ребра являются возможными действиями в этом состоянии.

Стохастическая среда:

- Некоторым переходы в среде присутствуют лишь с определенной вероятностью.
- В каждом конкретном испытаний, такой переход либо возможен, либо нет.
- Существуют переходы среди, которых в каждый конкретный момент присутствует только один.
- Путь

Марковский процесс принятия решений



Существующие подходы

- Алгоритмы обучения нейронных сетей
  - Алгоритм обратного распространения ошибки

    Недостатки:
    - Необходимо значительно изменить структуру сети
    - Сложно разделить случаи вознаграждения и наказания в испытании

- Динамическое программирование

  Недостатки:
  - Нужно знать точную модель среды

Существующие подходы

- Методы Монте-Карло
  Подходят, но в среднем работают хуже, чем TD-обучение

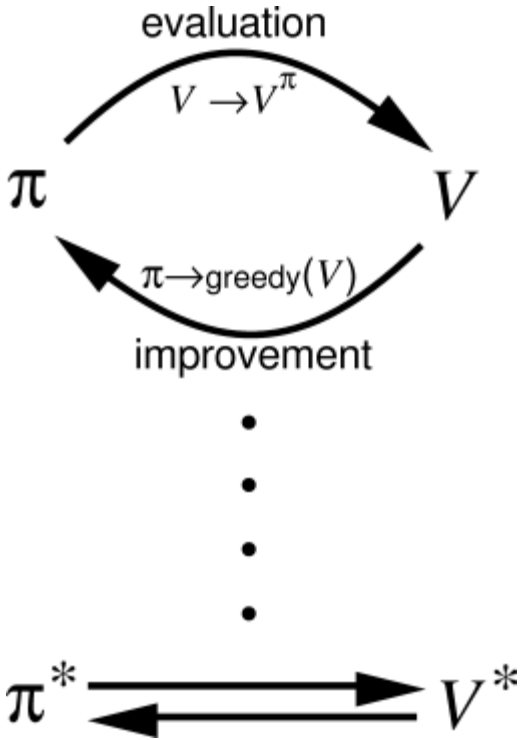
- Обучение на основе временных различий (TD-обучение)
  - Q-обучение и алгоритмы с разделенной оценкой стратегий
    Недостатки: не подходят по идейным соображениям

  - SARSA и алгоритмы с интегрированной оценкой
  - Алгоритмы Исполнитель-Критик

Обобщенная итерация по стратегиям

- Большинство описанных методов является частными случаями Обобщенной Итерации по Стратегиям

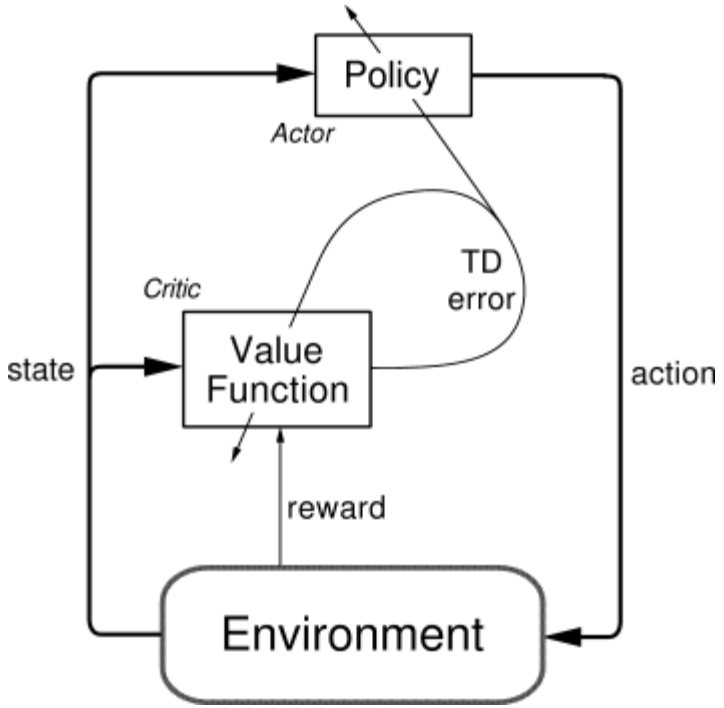
- Основные фазы:
  - Оценивание стратегии
  - Улучшение стратегии



Исполнитель - Критик

- Методы Исполнитель-Критик имеют различную структуру памяти для явного изменения стратегии не зависимо от функции ценности.

- Исполнитель и Критик взаимодействуют только, через передачу значения ошибки от критика исполнителю.

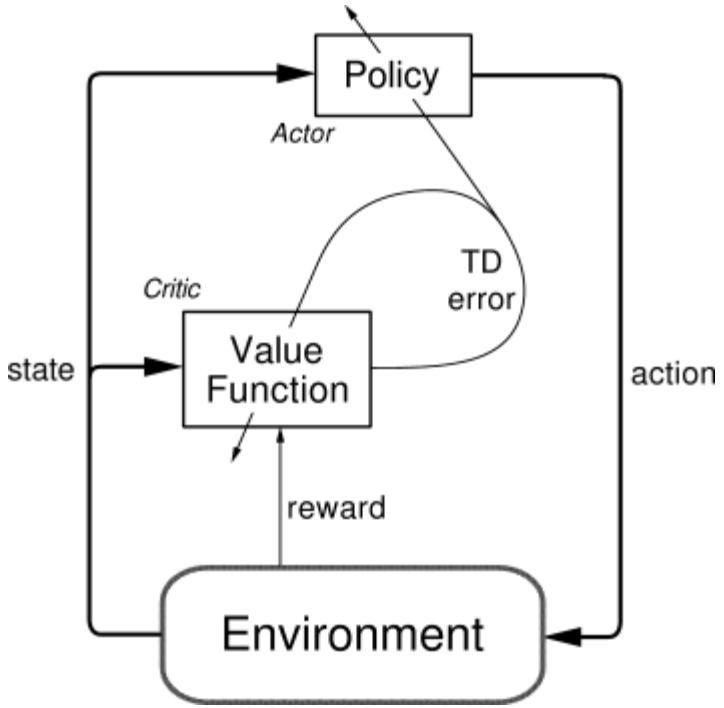


Исполнитель - Критик

В качестве исполнителя выбран алгоритм Direct Actor

- В отличии от других алгоритмов обновляет не только значение совершенного действия, но и всех альтернативных вариантов действия

В качестве критика используется модифицированная версия SARSA



Недостатки Методов TD-обучения (1)

Медленное распространение ошибки:

- Пошаговое обновление
- Угасание ошибки при распространении:

$$\Delta Q(s_t, a_t) = \alpha[R_t - Q(s_t, a_t)] \qquad R_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$$

Решение:

- Обновление всех посещенных состояний по окончании эпизода, как в методах Монте-Карло
- Модификация техники n-шагового дублирования вознаграждений:

$$R_t = r_{t+1} + \ldots + r_{t+n} + \gamma Q(s_{t+n}, a_{t+n})$$

Недостатки Методов TD-обучения (2)

Эпизод всегда заканчивается удовлетворением потребности:

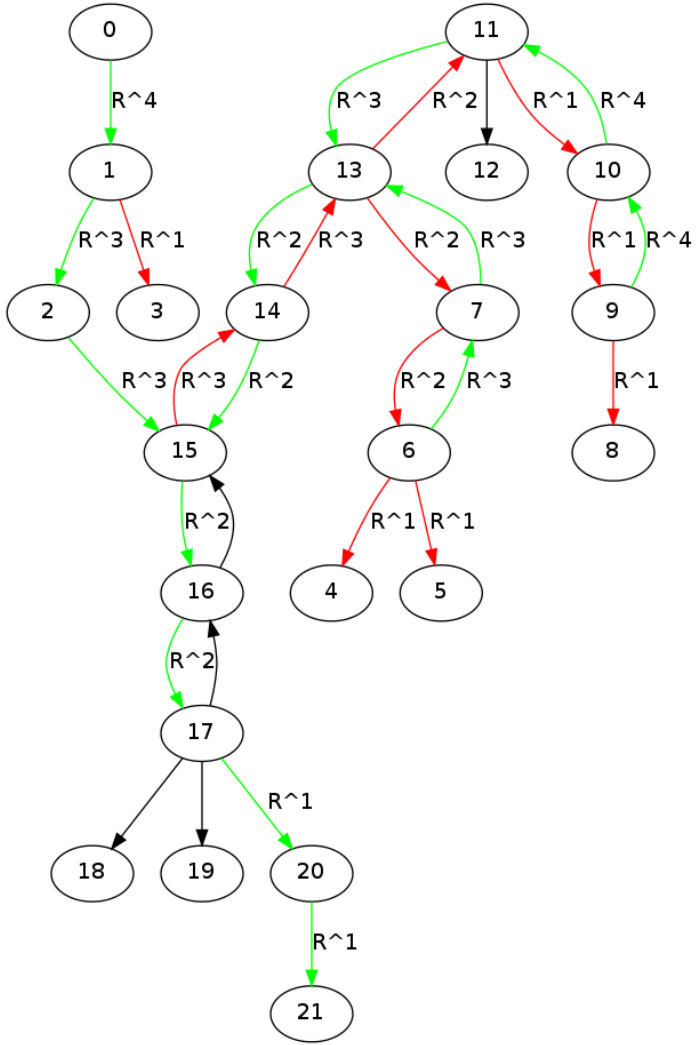
- ➢ Сложно определить какие именно действия в эпизоде были полезны, а какие вредны.

Решение:

- ➢ Пенальти за каждое попадание в тупик
- ➢ Специальное правило для получения оценки ожидаемого вознаграждения:

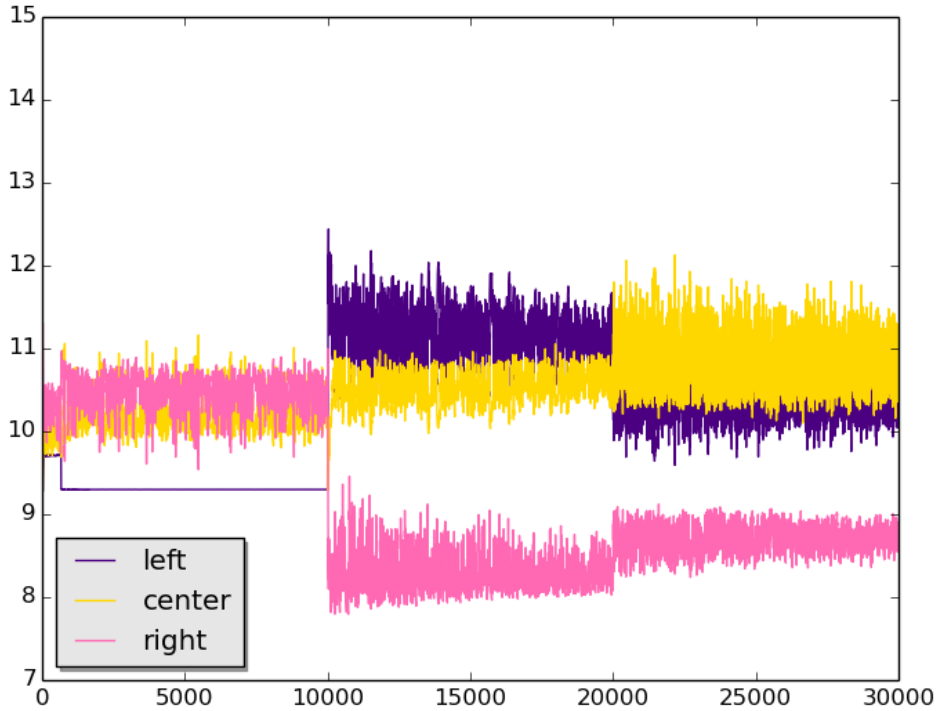
$$a' = argmax_{a_{t+k}}(R_{t+k}), \forall a_{t+k} \in A(s')$$

$$R_t = r_{t+1} + Q(s', a')$$


0
R^4
1
R^3
R^1
2
3
R^3
R^3
R^2
15
R^2
16
R^2
17
R^1
18
19
20
R^1
21
11
R^3
R^2
R^1
R^4
13
12
10
R^2
R^3
R^2
R^3
14
7
9
R^1
R^4
R^1
R^2
R^3
6
8
R^1
R^1
4
5

Поведение алгоритма в нестационарной среде

- Первый этап:
  - Prob(left) = 0.3
  - Prob(center) = 0.7
  - Prob (right) = 0.9
- Второй этап:
  - Prob(left) = 0.9
  - Prob(center) = 0.1
  - Prob (right) = 0.5
- Третий этап:
  - Prob(left) = 0.9
  - Prob(center) = 0.1
  - Prob (right) = 0.5



Результаты:

- Реализована библиотека на Python для работы с моделью функциональных систем

- Разработан алгоритм способный к адаптивному обучению в условиях стохастической среды и сохраняющий основные свойства модели и способный сравнивать эффективность различных вариантов действий и выбирать более эффективные из них.

Спасибо за внимание!