Real Time Reinforcement Learning

Charles Galambos

22nd June 2015

Aiseedo

What is it all about

- What is reinforcement learning ?
- The Temporal Difference algorithm
- Q-learning and SARSA
- How this can be used with recurrent neural networks
- Dealing with Real-time control

What is reinforcement learning ?

Learning what actions to take to achieve a goal.

Interaction

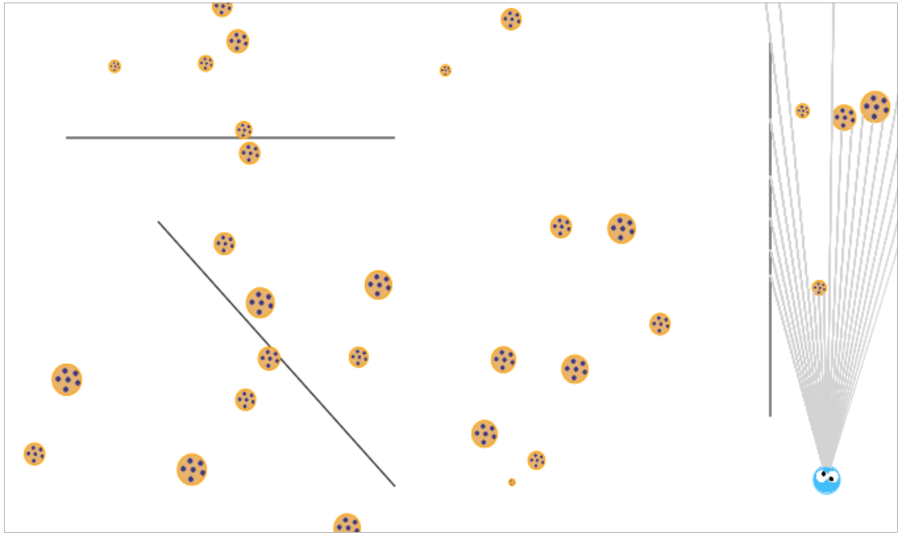
Agent

Rewards r

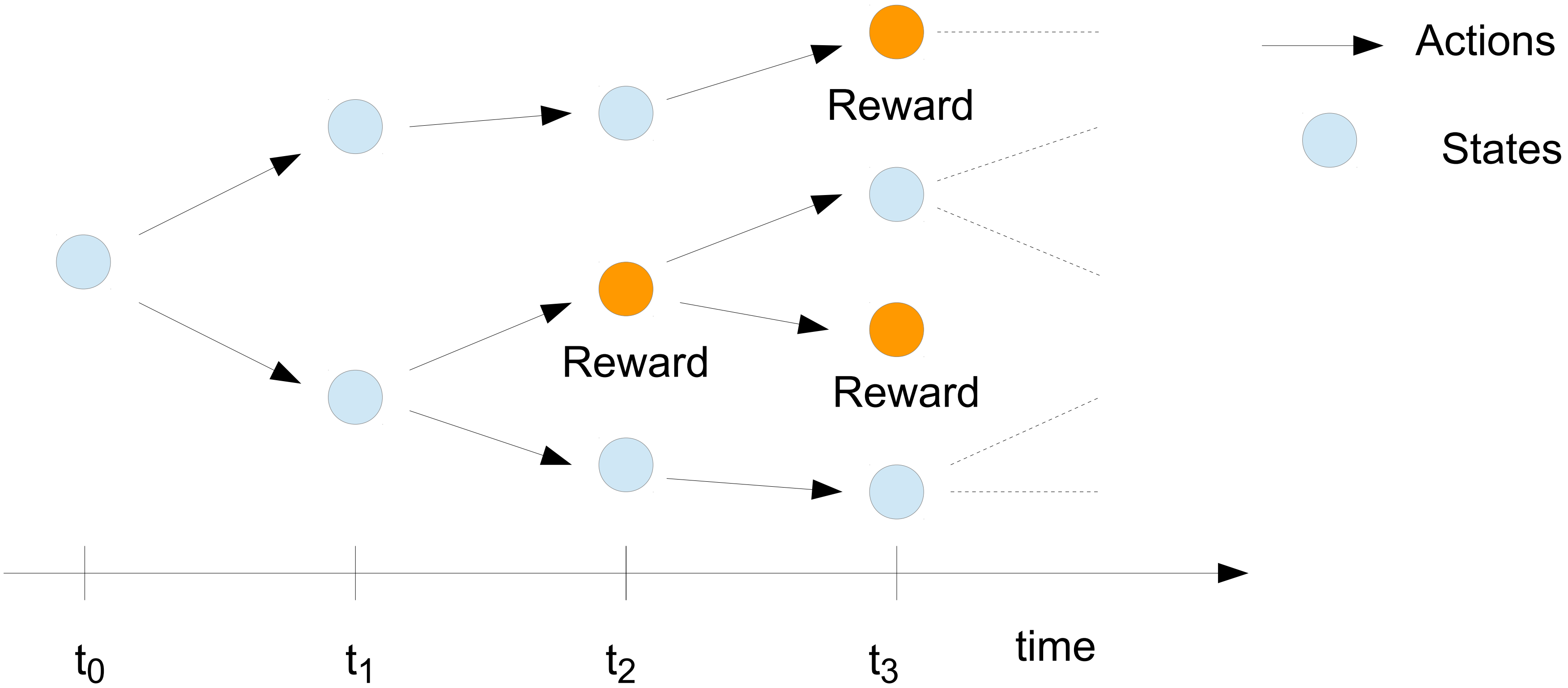
States s

Actions a

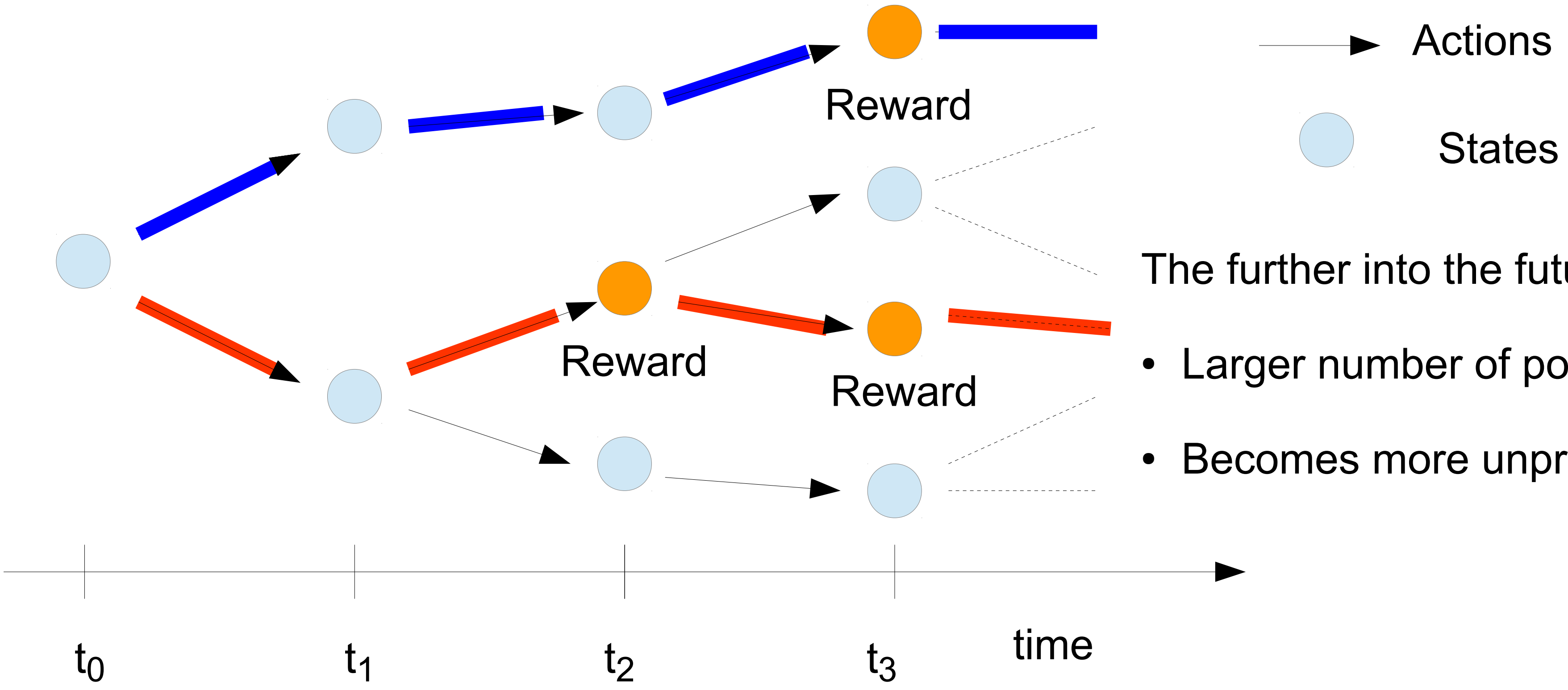
World



Which is the best path ?



Which is the best path ?

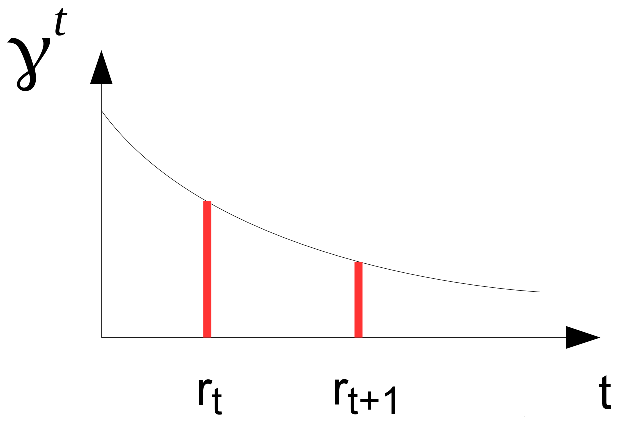


The further into the future we go:

- Larger number of possible states
- Becomes more unpredictable

How to measure success ?

Discount more distant rewards, as it is less certaint they will be received.



$$V(s_t)=\sum_{i=0}^{\infty}\gamma^{t+i}r_{t+i}$$

$V(s_t)$ the weighted sum of future rewards from time t

r_t the reward at time t

γ the discount rate of the rewards

Computing rewards

How to choose the best next action ?

- Traces – Attribute rewards to previous actions
- Sampling – Average of possible outcomes
- Temporal Differencing – We'll see next.

Temporal Differencing

$$Q(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Q is an estimate of our future rewards.

$$Q(s_t) = r_t + \gamma Q(s_{t+1})$$

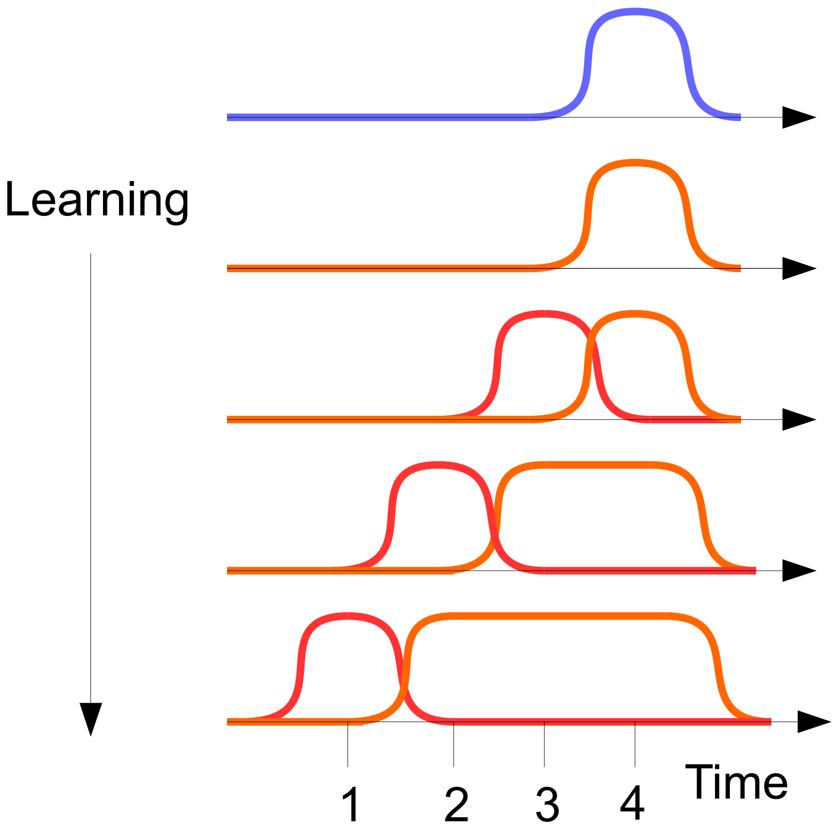
Estimate terms of current reward and future sum.

$$\delta_t = r_t + \gamma Q(s_{t+1}) - Q(s_t)$$

Rearrange to compute the error on Q.

Iterate: $Q'(s_t) = Q(s_t) + \alpha \delta_t$ Where α is the learning rate.

Temporal Differencing



Reward

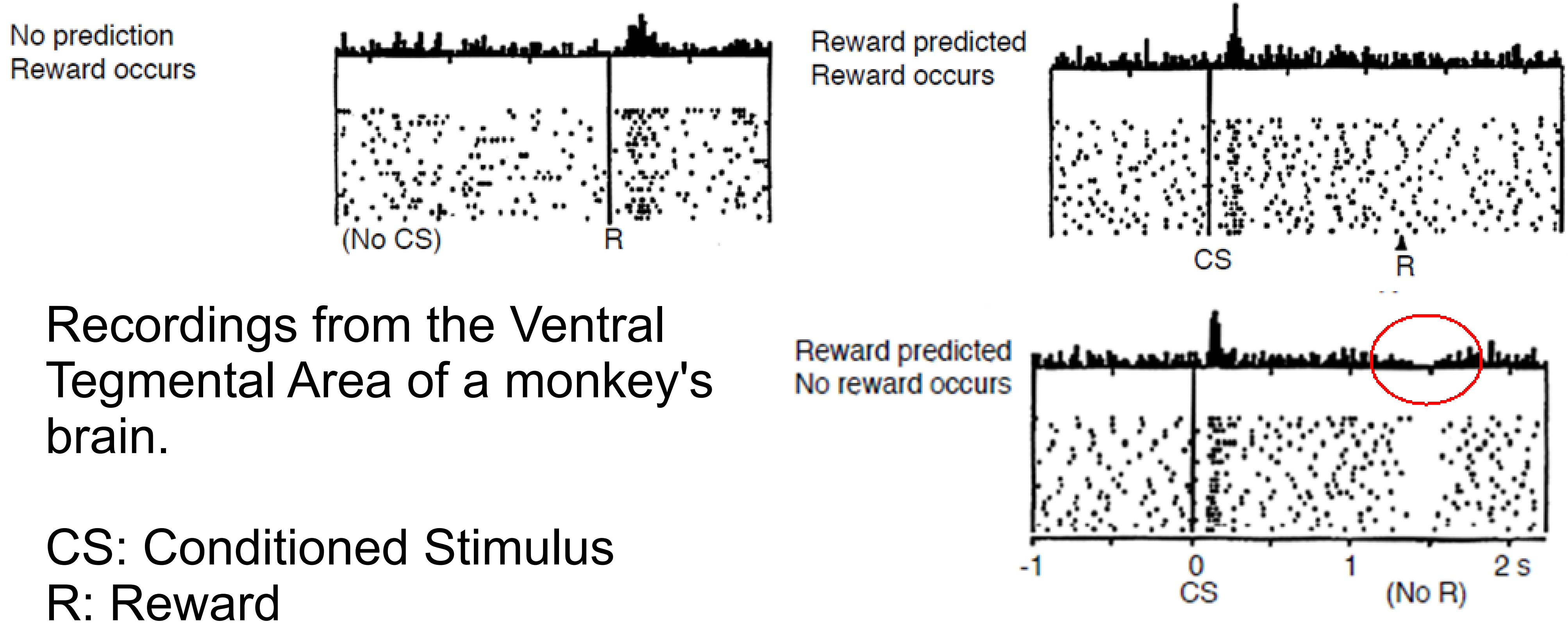
$Q(s_t)$

δ_t or $r_t + \gamma Q(s_{t+1}) - Q(s_t)$

As learning proceeds it looks for the 'cause' further back in time.

The algorithm looks to discriminate between states it expects a reward from those that it doesn't.

Rewards in the brain



Recordings from the Ventral Tegmental Area of a monkey's brain.

CS: Conditioned Stimulus
R: Reward

Algorithm: SARSA

State, Action -> Reward -> State, Action

$Q(s_t, a_t)$

- Is the expected rewards for taking action 'a' in state 's'.
- It assumes we always take the best action.

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

This is an 'on policy' algorithm, it works as long as we always choose the best action.

Algorithm: Q Learning

If we want to learn while doing something else we need an 'off policy' method.

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \cancel{Q(s_{t+1}, a_{t+1})} - Q(s_t, a_t)]$$

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \boxed{Max_a(Q(s_{t+1}, a))} - Q(s_t, a_t)]$$

Now we can learn while exploring.

Caution

- Temporal Differencing – is not a true gradient
- It can be unstable if used 'off policy'
- Shown to have problems with non-linear models

- Richard Suttons's TDC improves stability
- In practice it works well

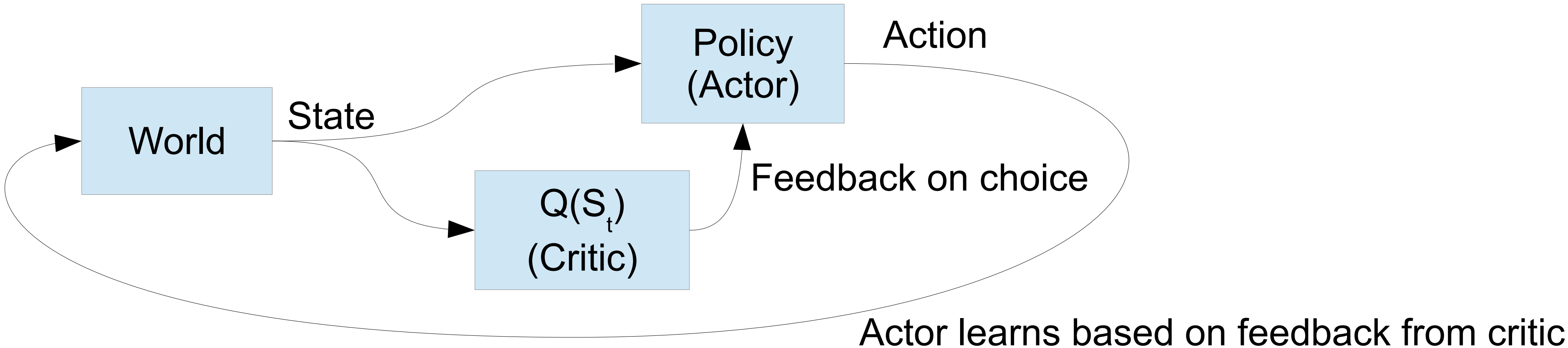
Applying Temporal Differencing

Challenges when applied to practical problems:

- No access to the true state of the world
- Huge number of potential actions
- Events are probabilistic and asynchronous
- It takes time for actions to have an effect

Actor / Critic

- Pick 100 random actions and use the best ?
- Good for simple actions, poor for complex ones
- So learn a function to suggest an action

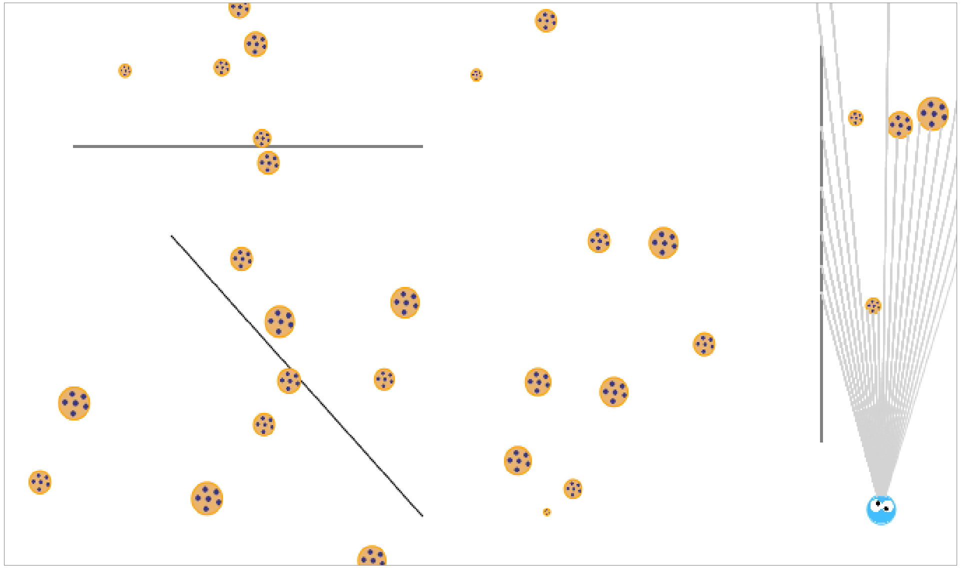


Message stream

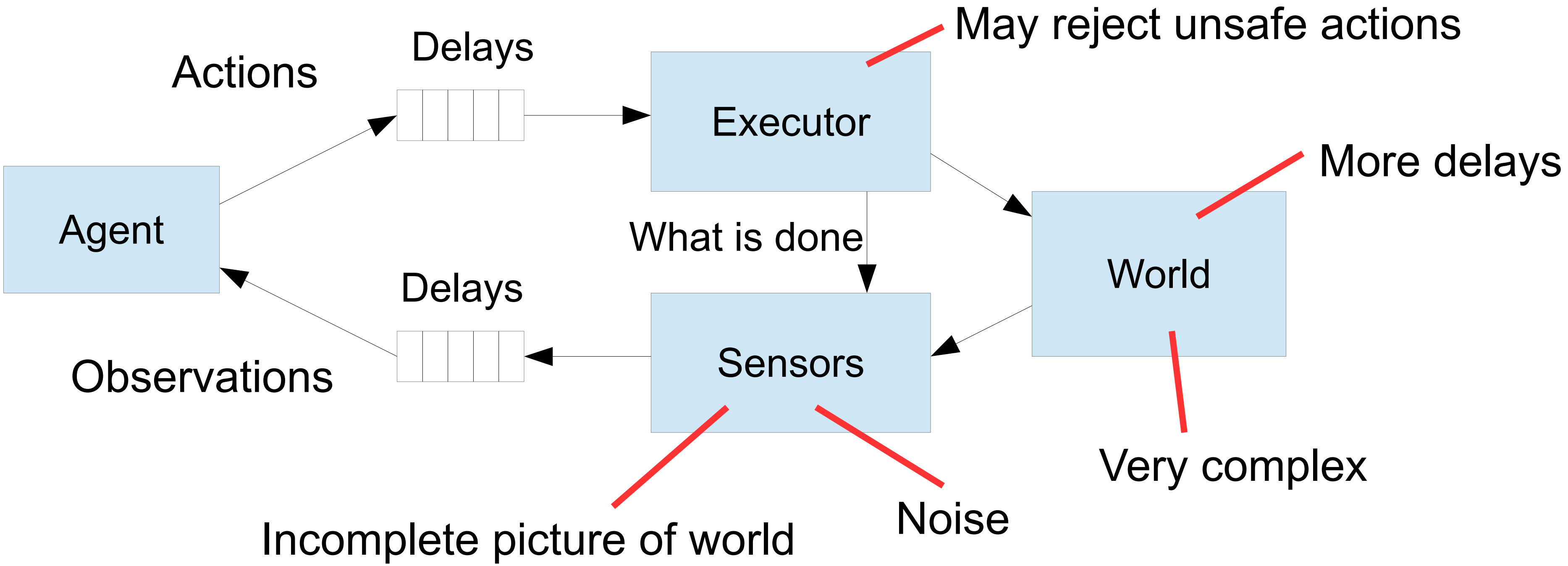
What does an message stream look like ?

Time

- Sensor - Speed 1.0 m/s
- Sensor - Range 1 m
- Intention – Accelerate -1 m/s
- Sensor - Speed 1.1 m/s
- Sensor - Speed 1.0 m/s
- Sensor – Range 0.7 m
- Action – Accelerate -1 m/s
- Sensor – Speed 0.7 m/s
- Sensor – Speed 0.4 m/s
- Sensor – Range 0.2 m

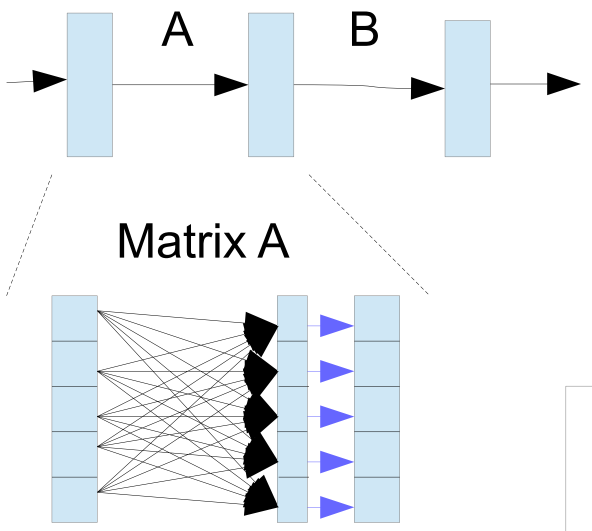


Deployment



May reject unsafe actions
Actions
Delays
Executor
More delays
Agent
What is done
World
Delays
Sensors
Observations
Very complex
Noise
Incomplete picture of world

Feed Forward Neural Networks



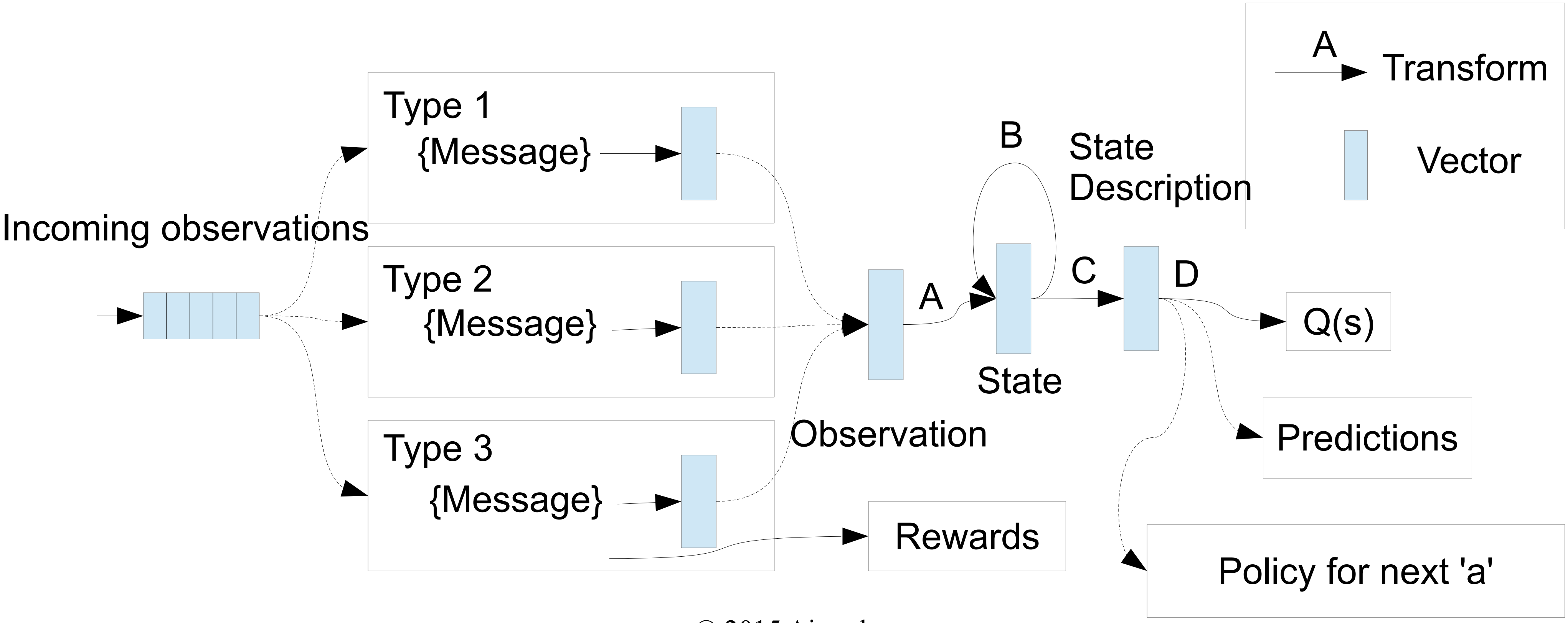
Vector

A Transform A

Element wise non linear operation.

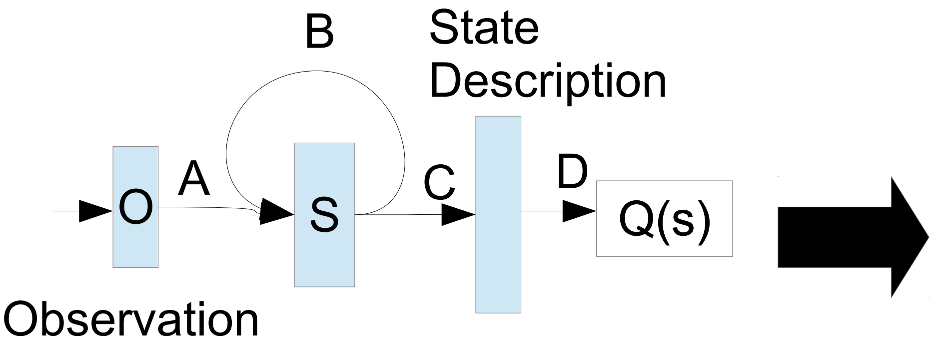
Sigmoid / Tanh / Rectified linear

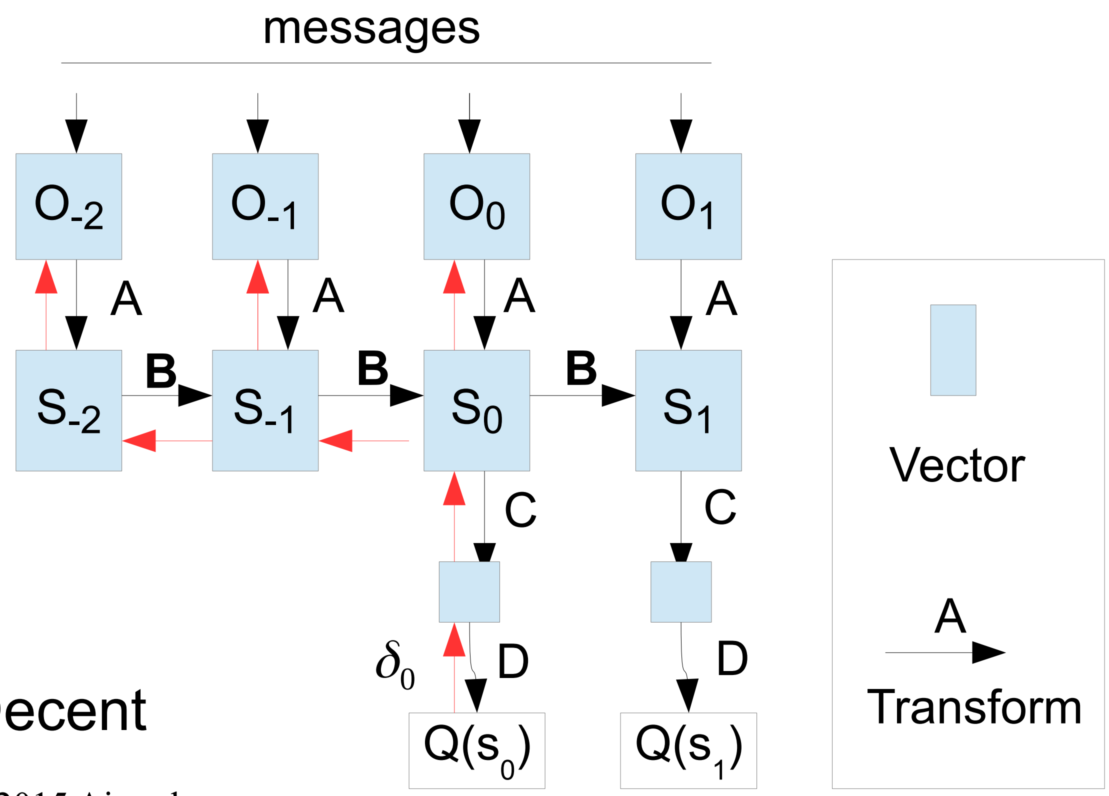
Recurrent Neural Networks



Training Recurrent Networks

To train you unroll the loop





$$\delta_t = r_t + \gamma Q(s_{t+1}) - Q(s_t)$$

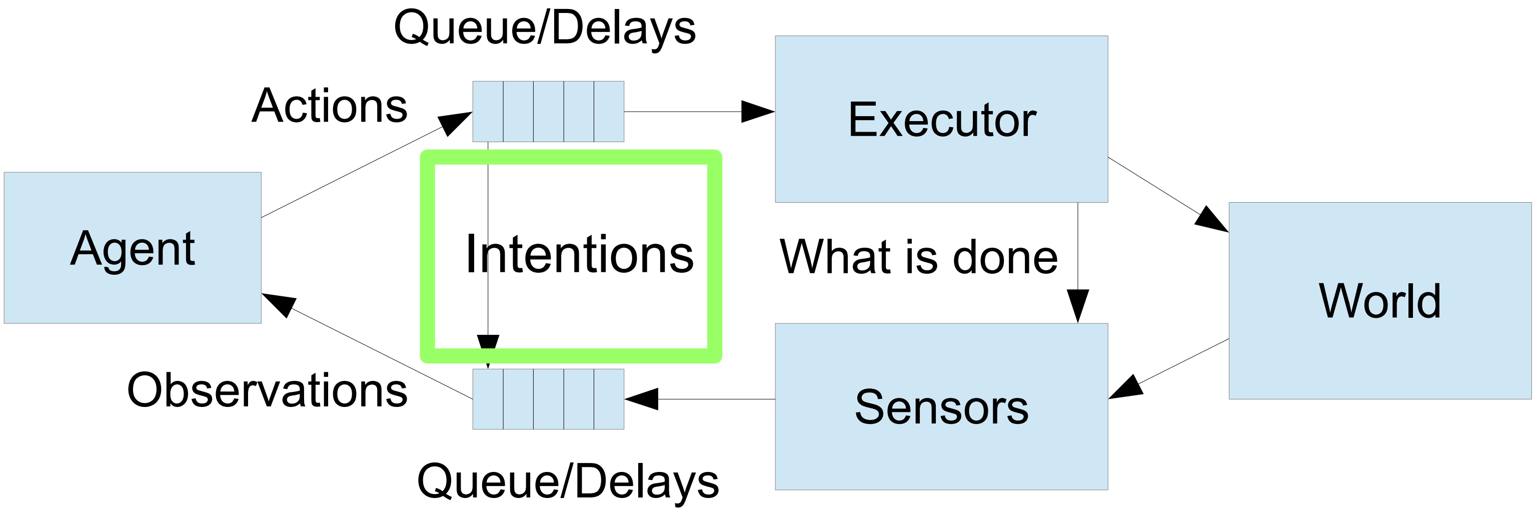
Trained with Stochastic Gradient Decent

Notes on model

- We are no longer using state, but observations
- Temporal Differencing improves retention of important state
- Actions and observations are treated equally

Intentions

- Our understanding of the world is delayed
- Need to anticipate the situation an action is performed in



Intentions

What does an event stream with intentions look like

Time

- Sensor - Speed 1.0 m/s
- Sensor - Range 1 m
- Intention – Accelerate -1 m/s
- Sensor - Speed 1.1 m/s
- Sensor - Speed 1.0 m/s
- Sensor – Range 0.7 m
- Action – Accelerate -1 m/s
- Sensor – Speed 0.7 m/s
- Sensor – Speed 0.4 m/s
- Sensor – Range 0.2 m

Delay between starting action and execution

Action and outcome closely associated

Intentions

- Intentions appear in training when the decisions are made
- Actions appear in the stream when the action is executed

- It makes it easier to associate actions and their outcomes
- Allows an external 'trainer' to teach the agent
- The delay in action execution is explicit

Cookie Monster Demo

Sensors creating messages at different rates

- Speed – When speed changes more than 5%
- Range scanner – 10 Hz
- Touch – When the monster bumps into something
- Taste – When a cookie is found

Actions:
- Turn left or right, speed up or slow down

Rewards:
- Positive rewards for eating cookies
- Negative rewards for crashing into walls hard

Cookie time!

Types of reward.

Decomposing future rewards, r_t directly.

$max(r_t, 0)$	Sum of likely wins
$min(r_t, 0)$	Sum of likely losses
$min(r_t + x, 0)$	Sum of losses larger than x, avoid large loses
$\lvert\delta_t\rvert$	If taken actions are set to 0, we can predict uncertainty

These can then be modelled separately and re-weighted.

Thanks for listening

Contact: charles@aiseedo.com

Follow Aiseedo on twitter: @aiseedo