Congestion Control

December 13, 2017

Congestion Control

- Why Congestion Control? Before congestion control was invented: Everyone sent as much as they pleased $\rightarrow$ *Congestion Collapse*.
- Goal: Estimate available bandwidth. Don't send too much, don't send too little.
- Method: Keep a *Congestion Window*
- E.g. Congestion Window of 5 means that we can have up to 5 packets somewhere in the network.

How does Congestion Control work nowadays?

Simplified:

- Congestion Window 1 in the beginning of a flow.
- Upon receiving an acknowledgement (ACK) for a previously sent packet, increase the Congestion Window (cwnd):

$$\text{cwnd} = \frac{1}{\text{cwnd}}$$

- Recently also a bit more sophisticated $\rightarrow$ *CUBIC* etc.
- When there is packet loss then we sent too much (buffer of a router on the way overflew) $\rightarrow$ congestion $\rightarrow$ decrease Congestion Window
- The most common thing to do is

$$\text{cwnd} = \frac{\text{cwnd}}{2}$$

Problems

- Only decrease window on loss $\rightarrow$ That's too late! Decrease before when buffers of routers fill up and latency increases!
- On wireless connections stochastic packet loss is common $\rightarrow$ TCP thinks it's congestion.

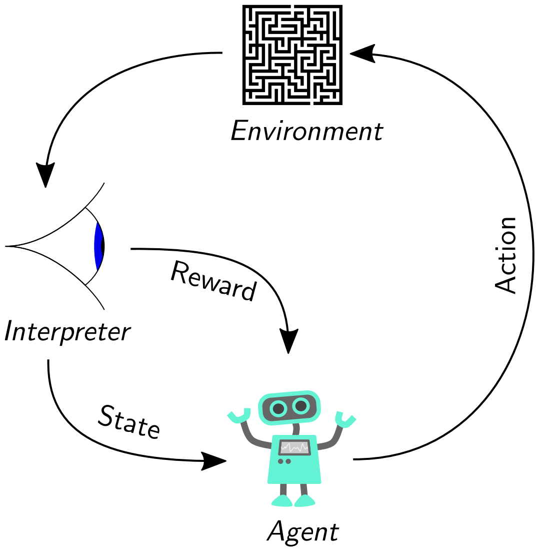
Potential Solution

- Let's build some machine learning thing!
- Solutions already exist → *TCP ex Machina* by Winstein and Balakrishnan (2013).
- They simulate networks and learn an optimum congestion control more or less by using a brute force algorithm.
- Example: Use networks with 1 to 5 senders, RTT from 10 to 100 ms. Use one set of congestion control rules for 1000 simulations. Then change some parameters and check if it improved (actually they do it in a smarter way)
- Problem: Training has to be done offline. But it would be nice to have a Congestion Control that learns in real time, online!

Other Potential Solution

- *PCC: Re-architecting Congestion Control for Consistent High Performance (PCC)* by Dong et al. (2013).
- Sender uses rate r_1 for some time and then b_2 for some time.
- Which rate was better?
- Use better rate and start experimenting again
- Problem: cannot react quickly as it has to wait for a whole measurement period until it changes its action.

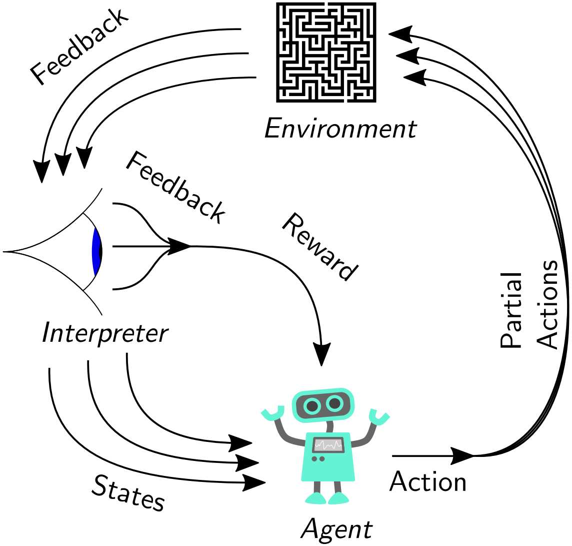
Reinforcement Learning



Problems with RL

- An action is increasing the Congestion Window
- We can calculate the reward after receiving all ACKs
- That takes at least one Round Trip Time
- In the mean time other packets could have arrived
- Example: We increase the cwnd by 2 packets. So we can send two packets. To evaluate if this action was good we have to get the ACKs of these two packets, which happens after one RTT! In the meantime another ACK could have arrived. What do we do? Not defined with RL...

Partial Actions



Partial Actions: Example

- Example: the window is 1.9
- Increase the window by 0.3 (action)
- Send two packets (partial actions).
- Receive the ACK for the first packet (feedback). Update the state and perform a new action.
- Receive the second ACK. Again, we update the state and perform an action. However, because we got all partial rewards of the previous action, we can calculate the reward and update our agent.

Key point: We can update the state without receiving the full reward yet.

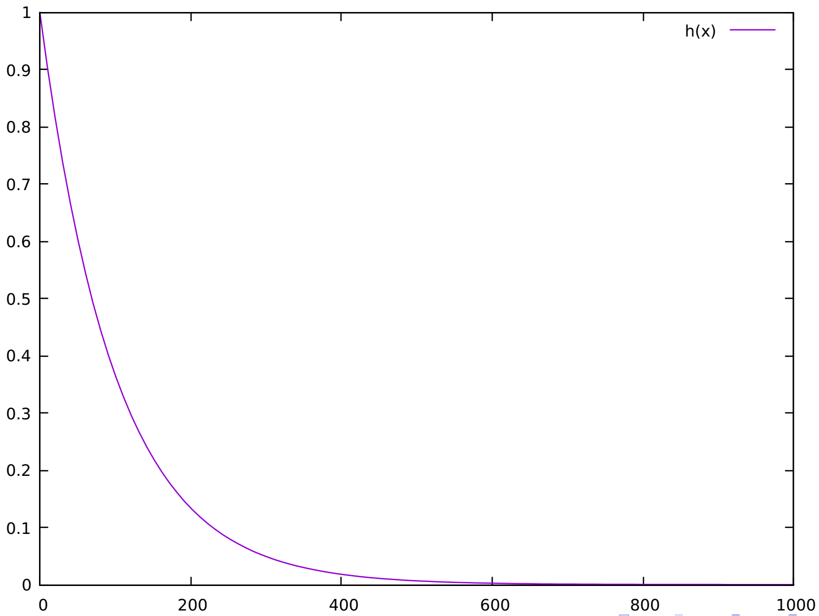
Asynchronous Actor Critic

- A Deep Learning framework for reinforcement learning. Used for learning how to play video games.
- Maximize expected long term reward:

$$R_t = \gamma r_t + (1 - \gamma) R_{t+1}$$

- The **Critic** tries to estimate how much (long-term) reward one can expect considering the current state.

Exponential decay with $\gamma = 0.01$



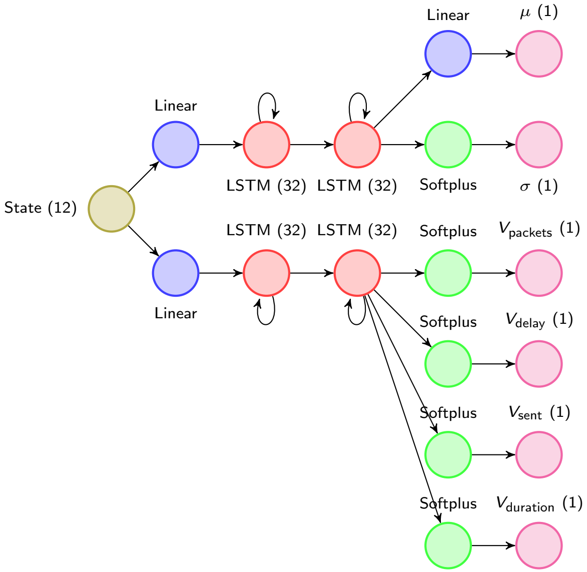
Asynchronous Actor Critic – How to choose γ

- In literature people just set $\gamma = 0.01$.
- Problem: The larger the congestion window the more often we'll receive new packets!
- $\frac{2}{k+1}$ is equivalent to a moving average considering the next k elements.
- Choose $\frac{2}{w_t+2}$ where w_t is the congestion window at time step t.

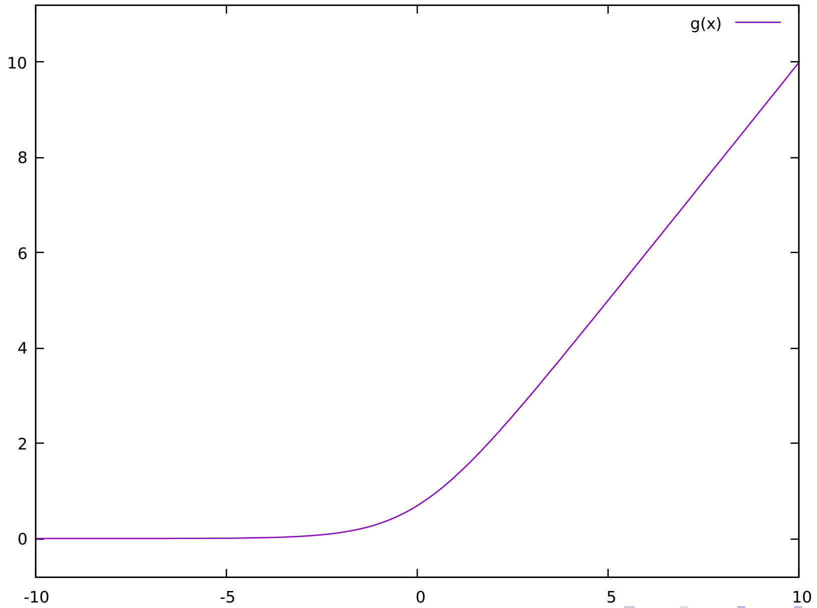
Asynchronous Actor Critic – Actor

- Wants to performs an action that are better than what the Critic would expect.
- It outputs two things:
  - What it thinks is the best action (e.g. increase the window by 0.3)
  - A standard deviation to experiment a little bit (e.g. 0.45)
- So we get a normal distribution from which we sample.
- Thanks to the standard deviation we experiment and don't get stuck with suboptimal actions.

Neural Network



Softplus function



Utility function

$$U_t = \text{throughput} \cdot \text{Sigmoid}_\alpha\left(\text{loss rate} - 0.05\right) - \text{sending rate} \cdot \text{loss rate}$$

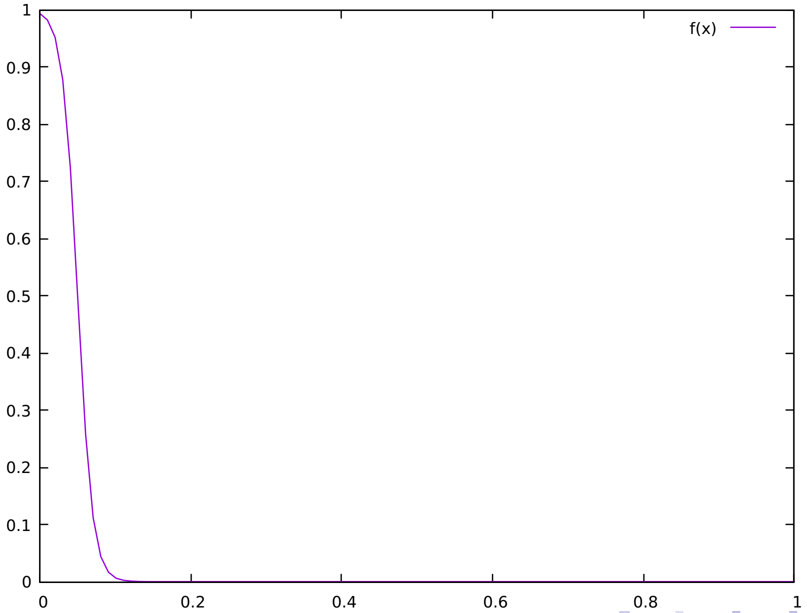
Utility function

$$U_t = \frac{r_{\text{received},t}}{r_{\text{duration},t}} \text{Sigmoid}_\alpha \left(\frac{r_{\text{sent},t} - r_{\text{received},t}}{r_{\text{sent},t}} - 0.05 \right) - \frac{r_{\text{sent},t} - r_{\text{received},t}}{r_{\text{duration},t}}$$

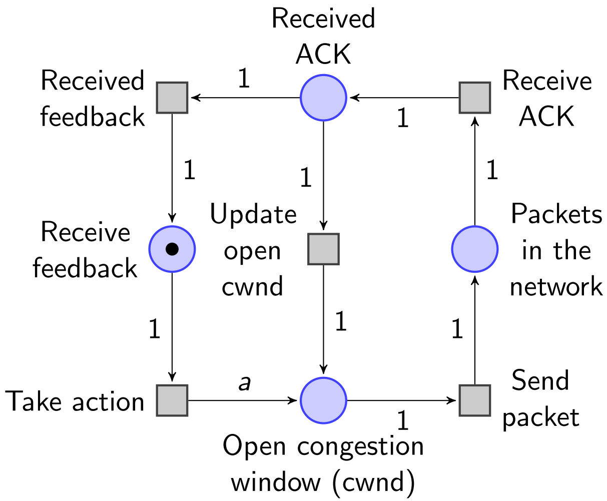
Sigmoid function

$$\text{Sigmoid}_\alpha(x) = \frac{1}{1 + e^{\alpha(x-0.05)}}$$

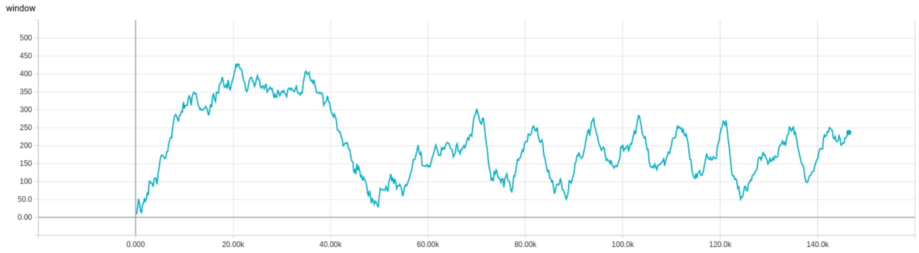
Sigmoid function for $\alpha = 100$ with 0.05 cutoff



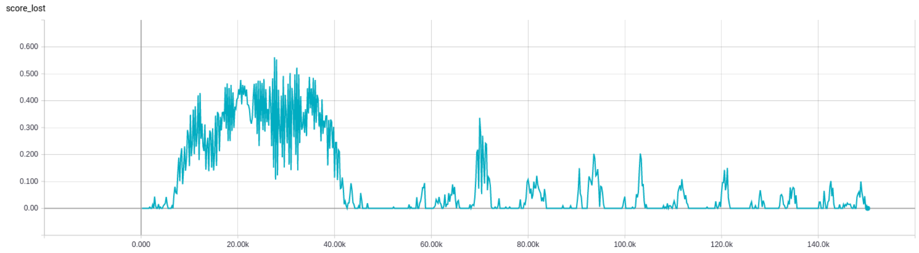
Petri net


Received ACK
Received feedback
Receive ACK
1
1
1
1
1
Receive feedback
Update open cwnd
Packets in the network
1
1
1
Take action
a
Send packet
1
Open congestion window (cwnd)

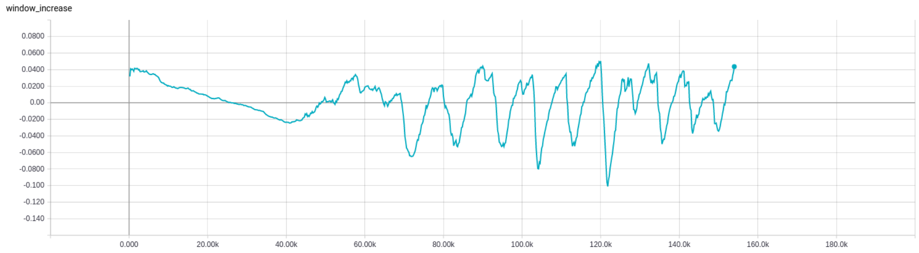
Some data


window
500
450
400
350
300
250
200
150
100
50.0
0.00
0.000
20.00k
40.00k
60.00k
80.00k
100.0k
120.0k
140.0k

Some data


score_lost
0.600
0.500
0.400
0.300
0.200
0.100
0.00
0.000
20.00k
40.00k
60.00k
80.00k
100.0k
120.0k
140.0k

Some data



window_increase
0.0800
0.0600
0.0400
0.0200
0.00
-0.0200
-0.0400
-0.0600
-0.0800
-0.100
-0.120
-0.140
0.000
20.00k
40.00k
60.00k
80.00k
100.0k
120.0k
140.0k
160.0k
180.0k