CS885 Reinforcement Learning
Lecture 12: June 8, 2018

Deep Recurrent Q-Networks

[GBC] Chap. 10

Outline

- Recurrent neural networks
  – Long short term memory (LSTM) networks
- Deep recurrent Q-networks

previously we discussed how you can model a POMDP by using a HMM and then doing planning. In Deep RL, we often replace the HMM with a RNN. If we combine a RNN with a DQN, we get DRQN.

Partial Observability

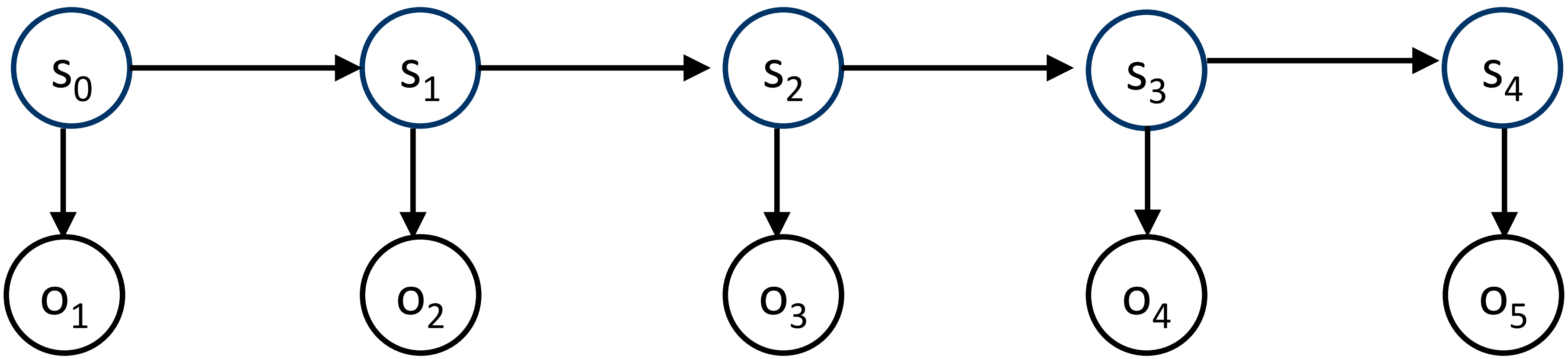
instead of selecting actions based on the state (which we don't have), we select actions based on the belief of the state, which we can get by using all previous observations.

- Hidden Markov model
  - Initial state distribution: $\Pr(s_0)$
  - Transition probabilities: $\Pr(s_{t+1}|s_t)$
  - Observation probabilities: $\Pr(o_t|s_t)$

- Belief monitoring

get a belief of this state based on the observations up until now

$$\Pr(s_t|o_{1..t}) \propto \Pr(o_t|s_t) \sum_{s_{t-1}} \Pr(s_t|s_{t-1}) \Pr(s_{t-1}|o_{1..t-1})$$



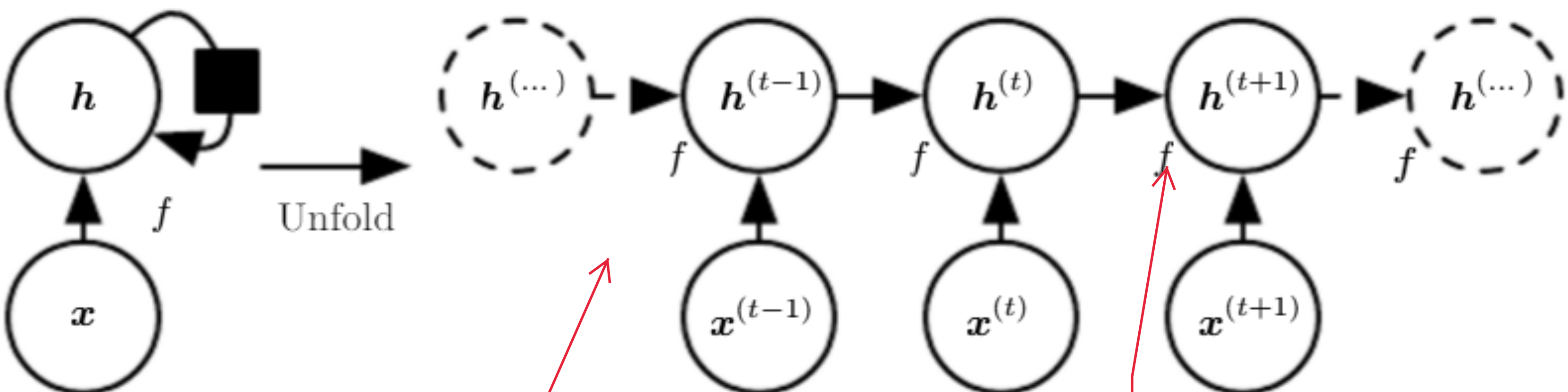
we don't have to use HMM to compute beliefs. Instead we can use a RNN.

Recurrent Neural Network (RNN)

- In RNNs, outputs can be fed back to the network as inputs, creating a recurrent structure
- HMMs can be simulated and generalized by RNNs
- RNNs can be used for belief monitoring

$\boldsymbol{x}_t$: vector of observations $\boldsymbol{h}_t$: belief state

even though it's not a probability like it is in HMM, its a vector of numbers that sufficiently describes our belief of the state



similar structure to HMM on the previous slide

function f takes the belief and the state to produce a belief for the future state. If we want we can make f the same equation as that in the previous slide for HMM to make the RNN equal to a HMM. That's why RNN is a generalized version of HMM.

you can think of the unrolled RNN as equivalent to a typical feedforward NN

Training

- Recurrent neural networks are trained by backpropagation on the unrolled network
  - E.g. backpropagation through time

but really it's just backprop

- Weight sharing:
  - Combine gradients of shared weights into a single gradient

it's more challenging to optimize RNN than feedforward

- Challenges:
  - Gradient vanishing (and explosion)
  - Long range memory
  - Prediction drift

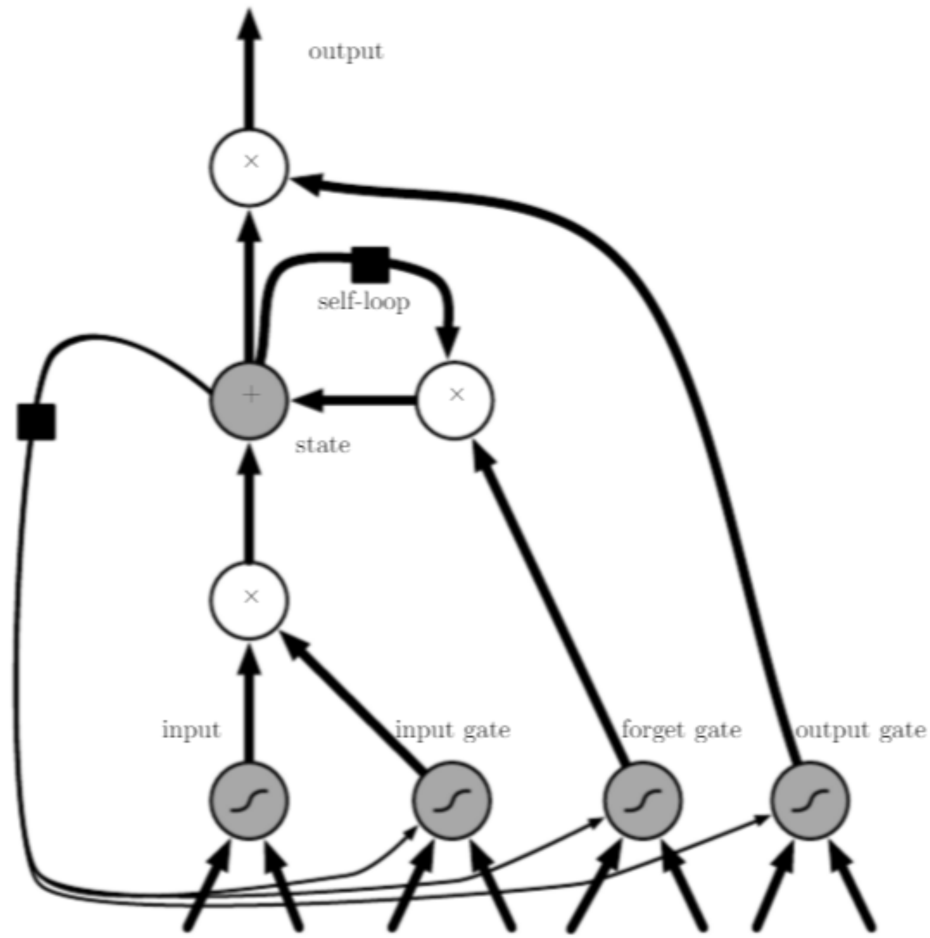
Long Short Term Memory (LSTM)

you could make the function f in head hidden cell a LSTM

- Special gated structure to control memorization and forgetting in RNNs

- Mitigate gradient vanishing

- Facilitate long term memory

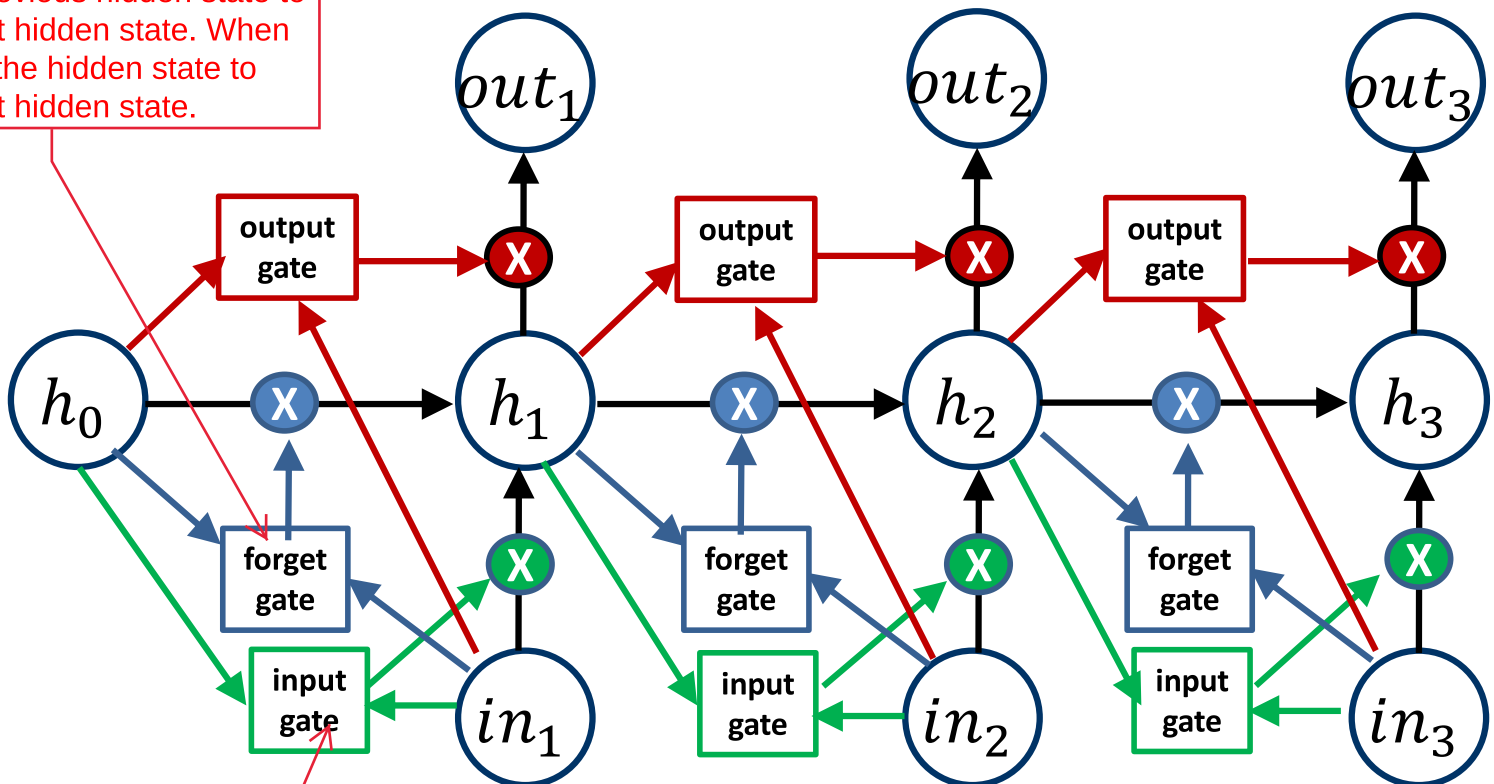


the gates determine which information can persist in the network. It also helps us keep long term memory by ensuring that inputs at each state don't override past memories

Unrolled long short term memory

at some probability the forget gate will be between 0 or 1. When it's 0, it "forgets" everything it learned by not allowing the previous hidden state to impact the next hidden state. When it's 1, it allows the hidden state to impact the next hidden state.

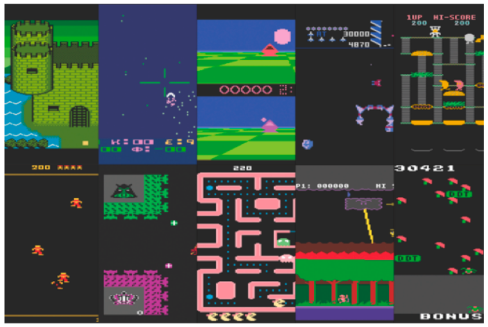
we typically represent the gates as a sigmoid unit that outputs a number between 0 and 1



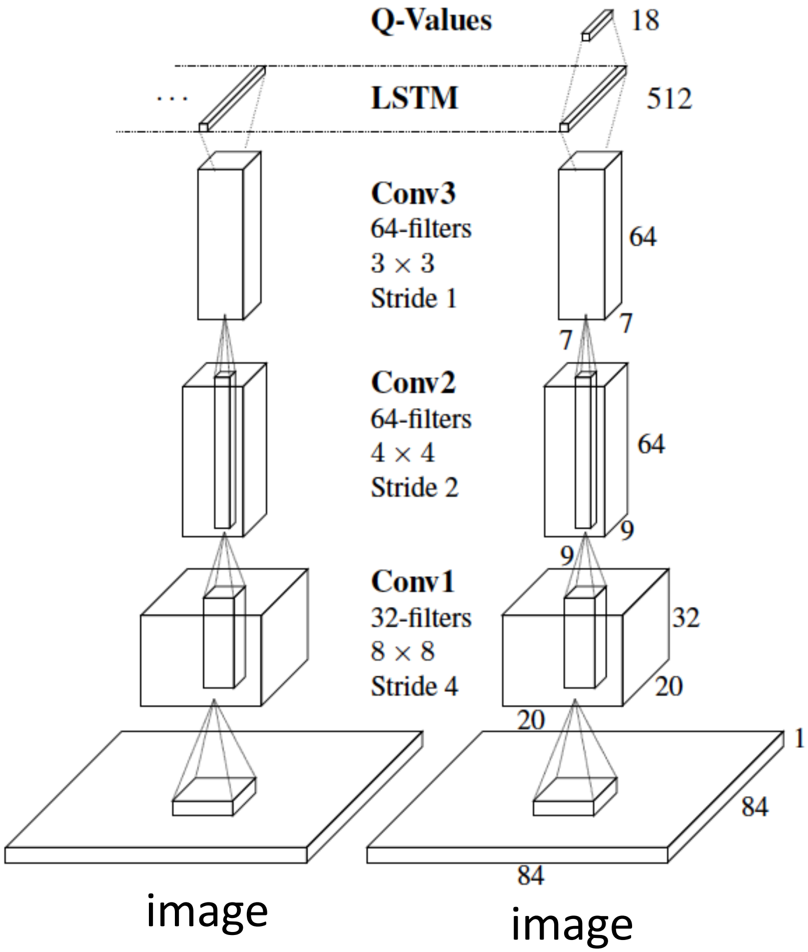
at some probability the input gate will be between 0 or 1. When it's 0, it blocks the input from being passed to the hidden state. When 1, it allows the input to go to the hidden state. So effectively it determines whether or not the input can be used to calculate the output.

Deep Recurrent Q-Network

- Hausknecht and Stone (2016)
  - Atari games



- Transition model
  - LSTM network
- Observation model
  - Convolutional network



Deep Recurrent Q-Network

Initialize weights $\boldsymbol{w}$ and $\overline{\boldsymbol{w}}$ at random in $[-1,1]$

Observe current state s

instead of taking a single experience tuple (s,a,r,s',d), we now take an entire trajectory b/c the LSTM network needs this

Loop

Execute policy for entire episode

Add episode $(o_1, a_1, o_2, a_2, o_3, a_3, \dots, o_T, a_T)$ to experience buffer

Sample episode from buffer

Initialize h_0

the Q-network takes the output of the RNN as an input

For $t = 1$ till the end of the episode do

$$\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(RNN_{\boldsymbol{w}}(\hat{o}_{1..t}), \hat{a}_t) - \hat{r} - \gamma \max_{\hat{a}_{t+1}} Q_{\overline{\boldsymbol{w}}}(RNN_{\overline{\boldsymbol{w}}}(\hat{o}_{1..t+1}), \hat{a}_{t+1})\right] \frac{\partial Q_{\boldsymbol{w}}(RNN_{\boldsymbol{w}}(\hat{o}_{1..t}), \hat{a}_t)}{\partial \boldsymbol{w}}$$

Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$

Every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Recurrent Q-Network

same as the previous slide with a slightly different/more efficient update calculation

Initialize weights $\boldsymbol{w}$ and $\overline{\boldsymbol{w}}$ at random in $[-1,1]$

Observe current state s

Loop

Execute policy for entire episode

Add episode $(o_1, a_1, o_2, a_2, o_3, a_3, \dots, o_T, a_T)$ to experience buffer

Sample episode from buffer

Initialize h_0

For $t = 1$ till the end of the episode do

$$\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(RNN_{\boldsymbol{w}}(h_{t-1}\hat{o}_t), \hat{a}_t) - \hat{r} - \gamma \max_{\hat{a}'} \mathrm{Q}_{\overline{\boldsymbol{w}}}(RNN_{\overline{\boldsymbol{w}}}(h_{t-1}\hat{o}_t\hat{o}_{t+1}), \hat{a}_{t+1})\right] \frac{\partial Q_{\boldsymbol{w}}(RNN_{\boldsymbol{w}}(h_{t-1}\hat{o}_t), \hat{a})}{\partial \boldsymbol{w}}$$

$h_t \leftarrow RNN_{\overline{\boldsymbol{w}}}(h_{t-1}, \hat{o}_t)$

Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$

Every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

in some games, the images are good approximations of the states, so the RNN doesn't make a big difference. But in others, it makes a big difference

Results

every once in a while some frames are skipped

	DRQN $\pm std$	DQN $\pm std$	
Game		Ours	Mnih et al.
Asteroids	1020 ($\pm$312)	1070 ($\pm$345)	1629 ($\pm$542)
Beam Rider	3269 ($\pm$1167)	**6923** ($\pm$1027)	6846 ($\pm$1619)
Bowling	62 ($\pm$5.9)	72 ($\pm$11)	42 ($\pm$88)
Centipede	3534 ($\pm$1601)	3653 ($\pm$1903)	8309 ($\pm$5237)
Chopper Cmd	2070 ($\pm$875)	1460 ($\pm$976)	6687 ($\pm$2916)
Double Dunk	**-2** ($\pm$7.8)	-10 ($\pm$3.5)	-18.1 ($\pm$2.6)
Frostbite	**2875** ($\pm$535)	519 ($\pm$363)	328.3 ($\pm$250.5)
Ice Hockey	-4.4 ($\pm$1.6)	-3.5 ($\pm$3.5)	-1.6 ($\pm$2.5)
Ms. Pacman	2048 ($\pm$653)	2363 ($\pm$735)	2311 ($\pm$525)

Table 1: On standard Atari games, DRQN performance parallels DQN, excelling in the games of Frostbite and Double Dunk, but struggling on Beam Rider. Bolded font indicates statistical significance between DRQN and our DQN.[5]

Flickering games
(missing observations)