Lecture 24: Reinforcement Learning

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer

Resources for this lecture

Stanford CS231n, Lecture 17

Berkeley CS 294: Deep Reinforcement Learning
http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_3_rl_intro.pdf

V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature (2016)

Technical note: Q-Learning
http://www.gatsby.ucl.ac.uk/~Dayan/papers/cjch.pdf

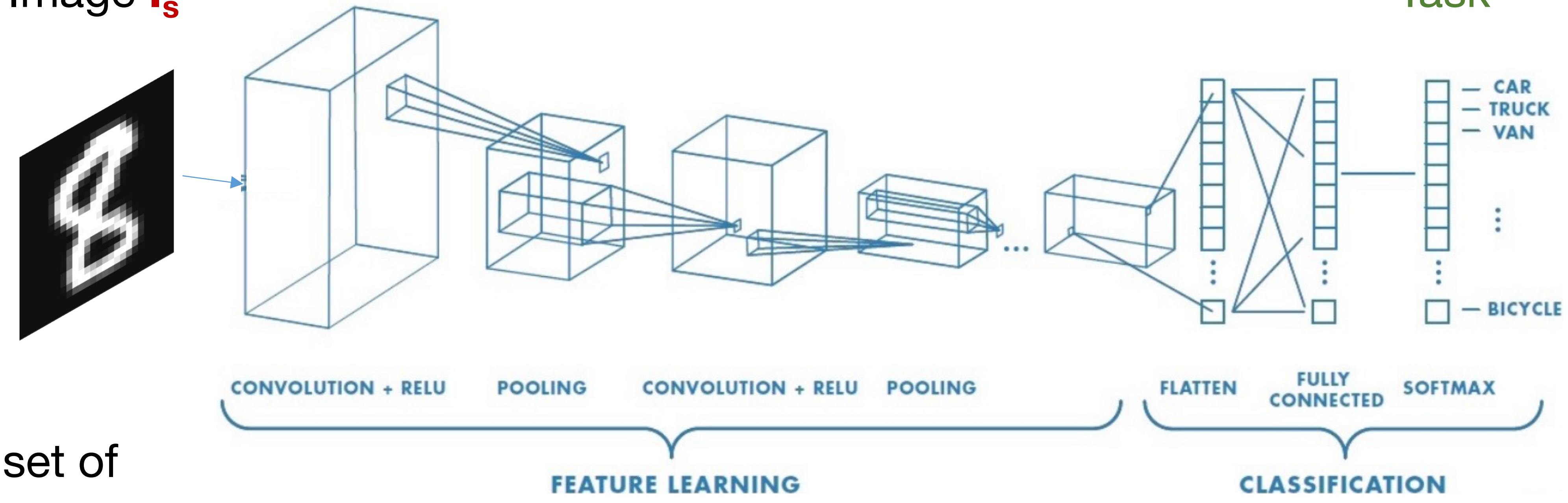
Reinforcement learning - in a nutshell

- So far, we've looked at:
  1) Decisions from fixed images (classification, detection, segmentation)

CNN's

Image $\mathbf{I_s}$

Task



Fixed set of training images

$$\text{Task} = \mathbf{W_n} \ldots \mathrm{ReLU}[\mathbf{W_1}\, \mathrm{ReLU}[\mathbf{W_0}\, \mathbf{I_s}]\ldots]$$

Reinforcement learning - in a nutshell

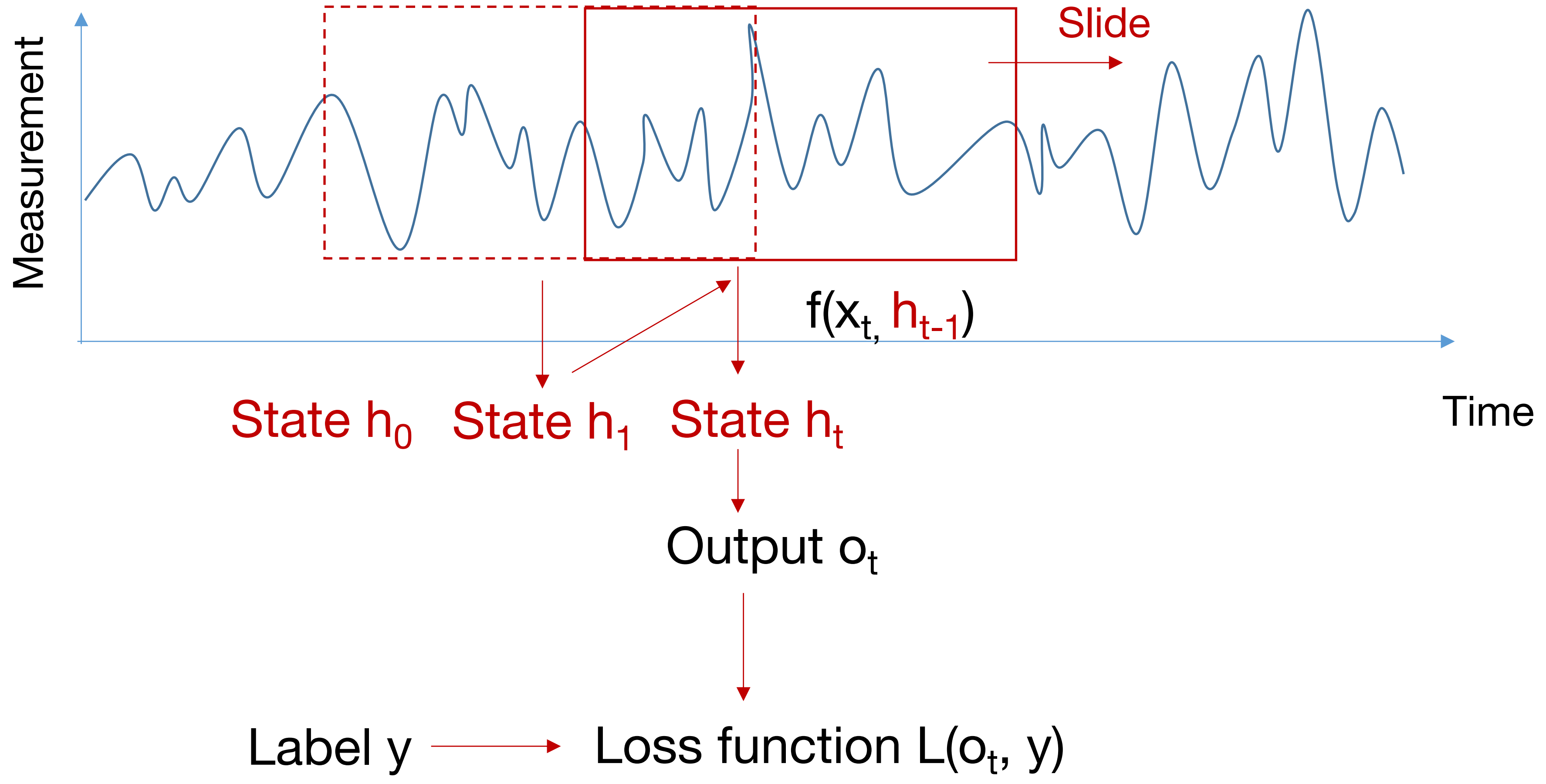
- So far, we've looked at:
  1) Decisions from fixed images (classification, detection, segmentation)

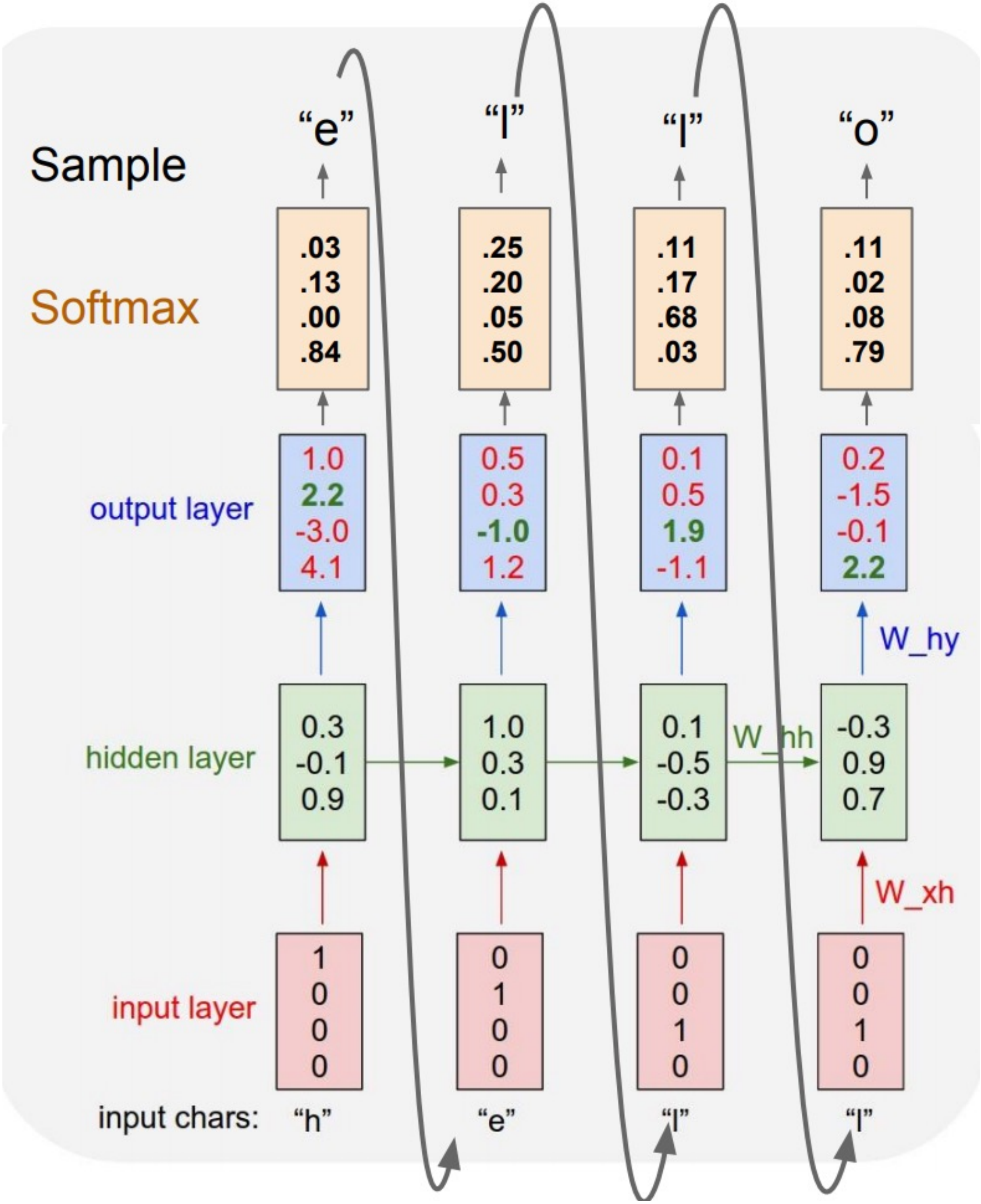
     CNN's

  2) Decisions from time-sequence data (captioning as classification, etc.)

     RNN's

Fixed set of
temporal sequences





From Stanford CS231n Lecture 10 slides

deep imaging

Reinforcement learning - in a nutshell

- So far, we've looked at:
  1) Decisions from fixed images (classification, detection, segmentation)

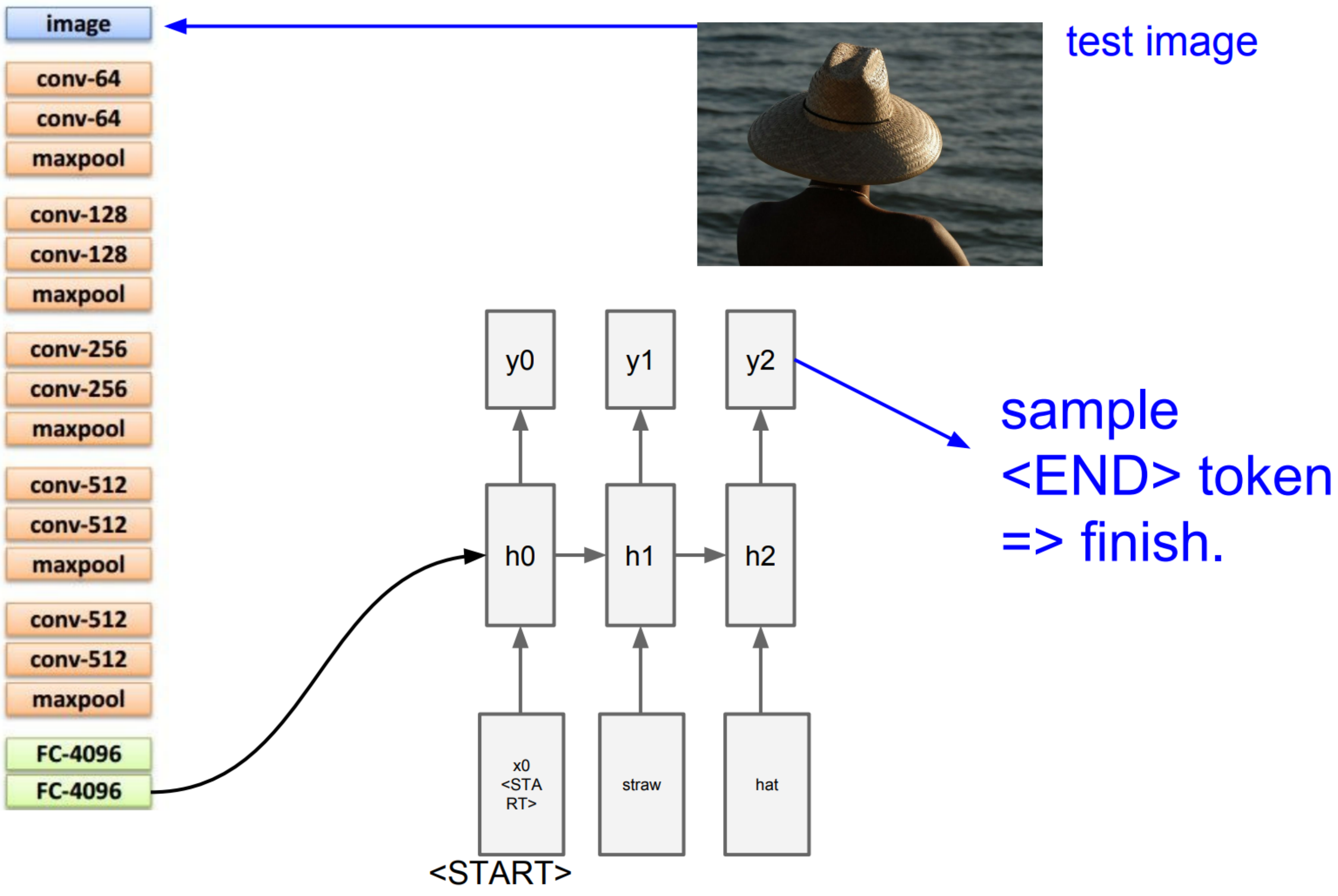
CNN's

2) Decisions from time-sequence data (captioning as classification, etc.)

Decisions from images and time-sequence data (video classification, etc.)

RNN's

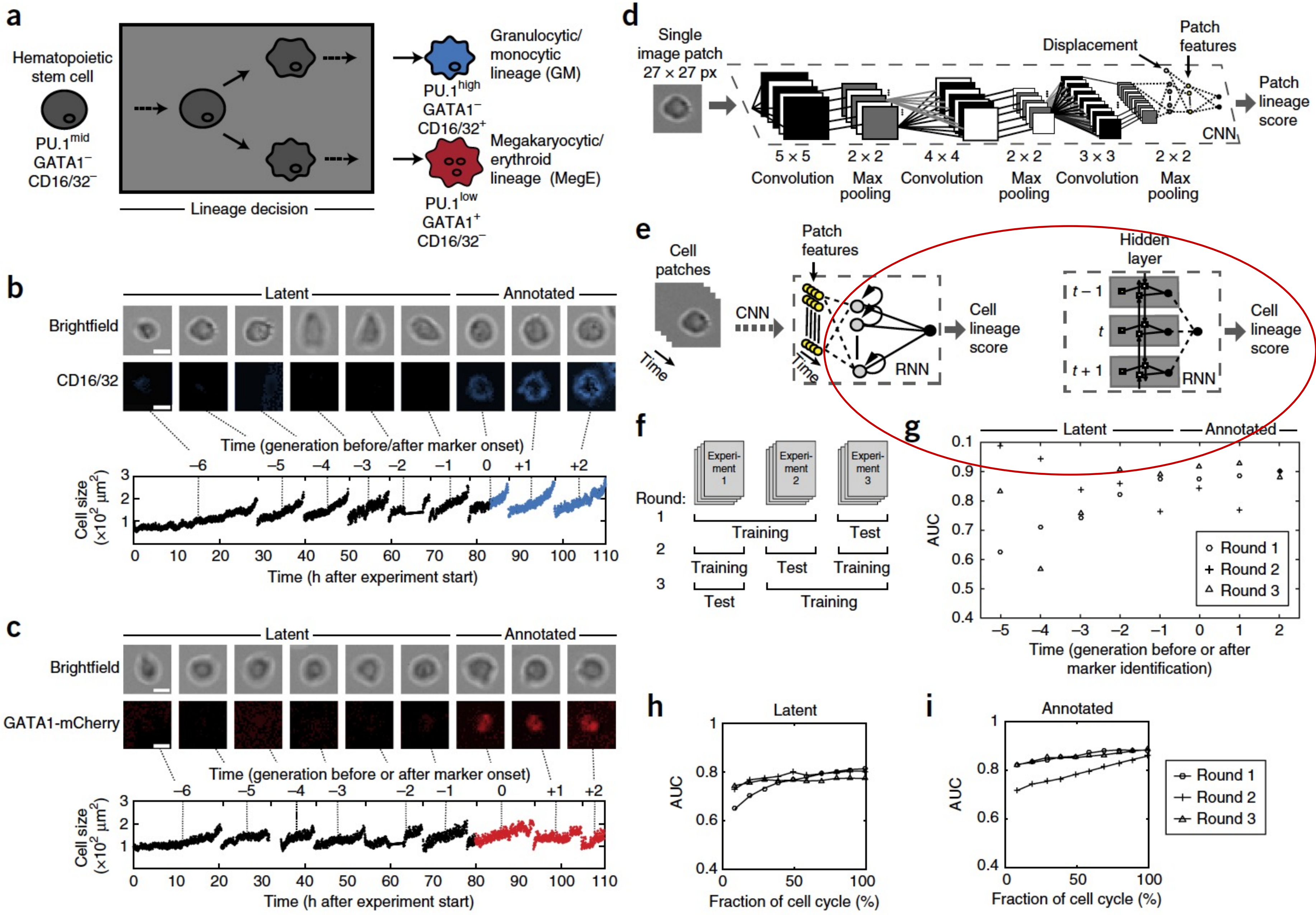
Example: Image captioning



From Stanford CS231n Lecture 10 slides

Prospective identification of hematopoietic lineage choice by deep learning

Felix Buggenthin[1,6], Florian Buettner[1,2,6], Philipp S Hoppe[3,4], Max Endele[3], Manuel Kroiss[1,5], Michael Strasser[1], Michael Schwarzfischer[1], Dirk Loeffler[3,4], Konstantinos D Kokkaliaris[3,4], Oliver Hilsenbeck[3,4], Timm Schroeder[3,4], Fabian J Theis[1,5] & Carsten Marr[1]



Reinforcement learning - in a nutshell

- So far, we've looked at:
  1) Decisions from fixed images (classification, detection, segmentation)

CNN's

  2) Decisions from time-sequence data (captioning as classification, etc.)

  Decisions from images and time-sequence data (video classification, etc.)

RNN's

- Now, we're going to consider decisions for *dynamic data*
  - Most successful application: dynamic image data
    e.g.: video games, images of a Go game, car turning through obstacles

Reinforcement Learning

The step from fixed video to dynamic video

Outcome:
Cell type B

Goal: examine *all* data to make final decision

The step from fixed video to dynamic video

deep imaging

Outcome:
Cell type B

Goal: examine *all* data to make final decision

“jump”

block

forward

The step from fixed video to dynamic video

deep imaging

Outcome:
Cell type B

Goal: examine *all* data to make final decision

Goal: decide on path through data to get to final result

"jump"

block

forward

"jump"

block

forward

"jump"

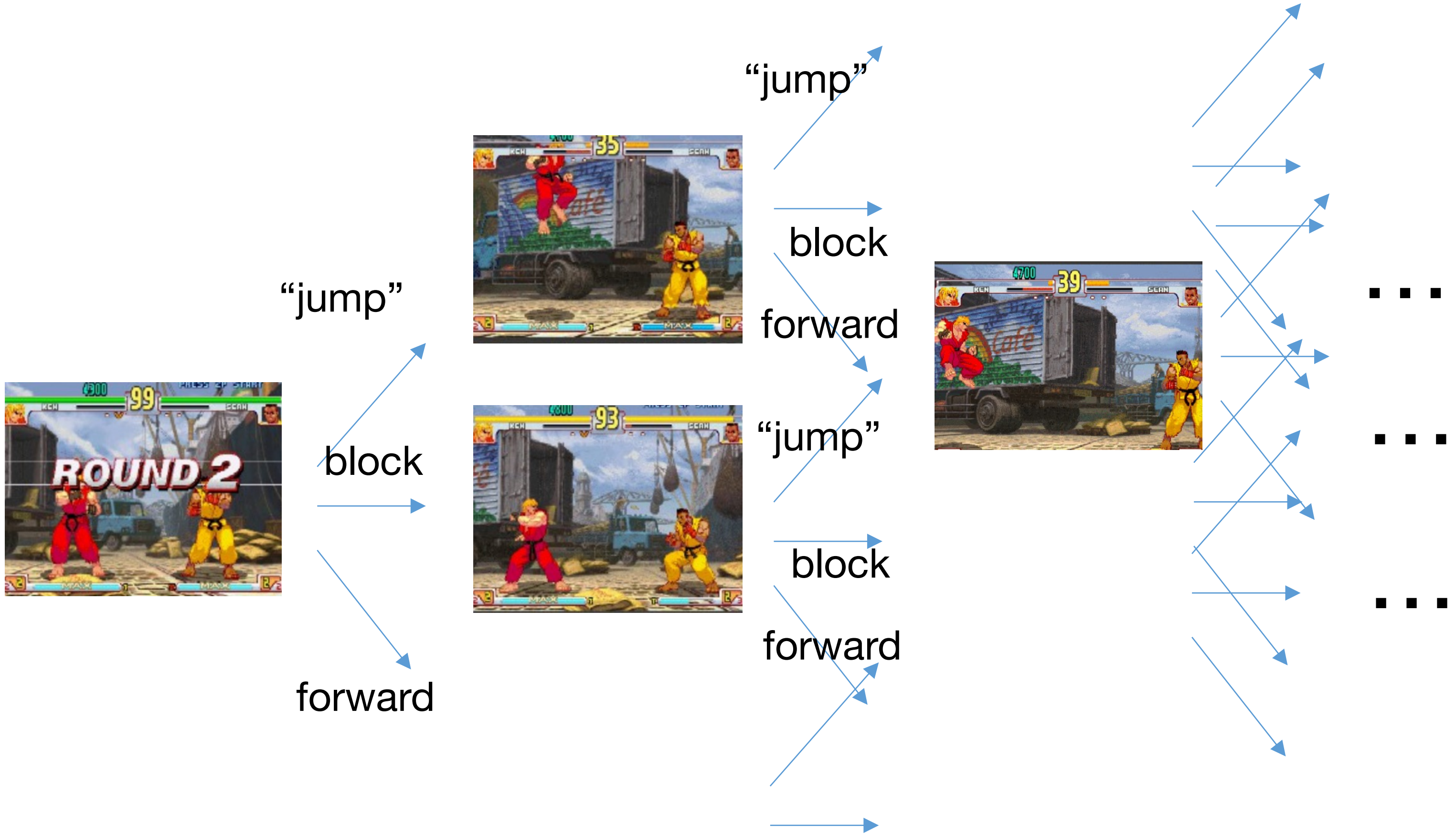
block

forward

The step from fixed video to dynamic video

Outcome:
Cell type B

Goal: examine *all* data to make final decision



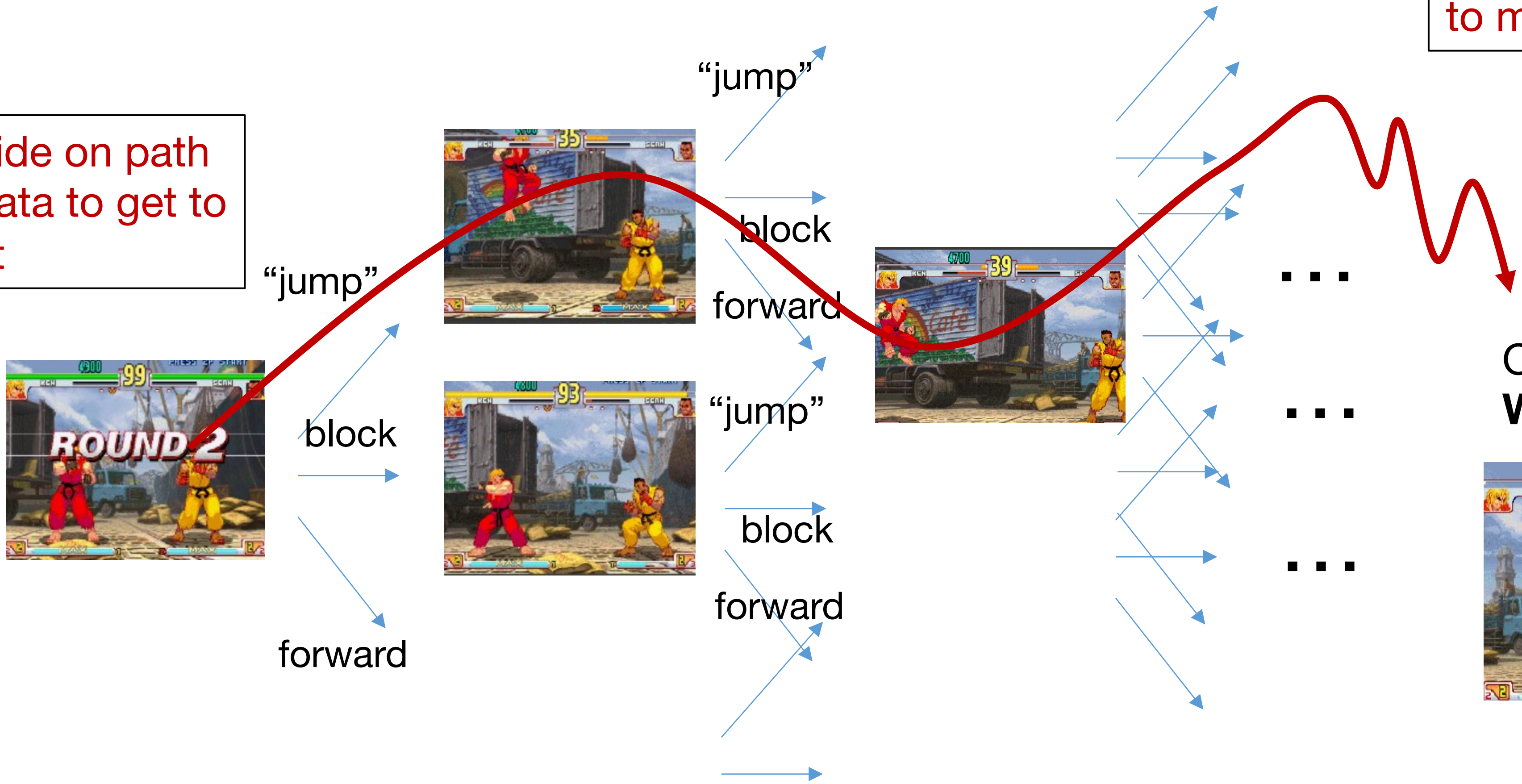
Outcome:
Win the game!

The step from fixed video to dynamic video

Outcome:
Cell type B

Goal: examine *all* data to make final decision

Goal: decide on path through data to get to final result



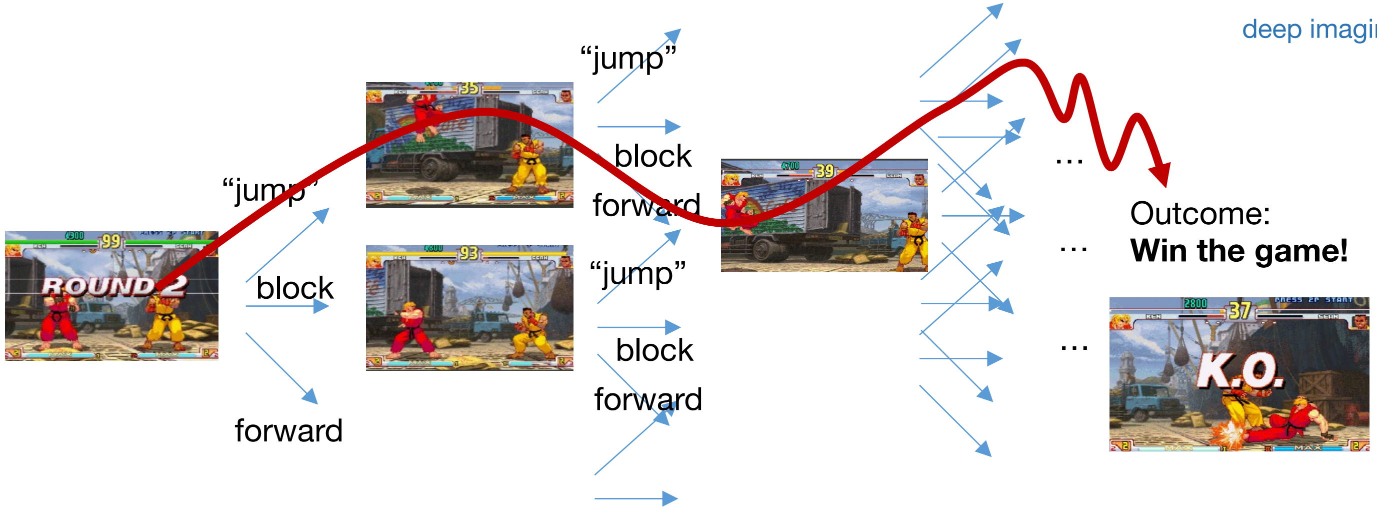
Outcome:
Win the game!

Supervised ML

Outcome:
Cell type B

- Fixed image sequence
- Goal: match to known label (large labeled dataset needed)
- Output: label
- Examines all data

Reinforcement learning



- *Dynamic/active* image sequence
- Goal: get to known desired outcome (no labels needed, really…)
- Output: sequence of actions
- Not possible to examine *all* data

deep imaging

Terms and notation

deep imaging

Example situation: Preparing for surgery using a robotically controlled instrument with an endoscope camera. You want endoscope to guide itself to tumor as quickly as possible

Movement choices:

First image (of skin) → Up / Down / Left / Right →

Initial state s_0

Action a_0

Terms and notation

Example situation: Preparing for surgery using a robotically controlled instrument with an endoscope camera. You want endoscope to guide itself to tumor as quickly as possible

Movement choices: How'd we do?

First image (of skin) → Up / Down / Left / Right → +1 for darker image / -1 for lighter image → Second image → ...

Initial state s_0

Action a_0

Reward r_0

state s_1

Continue until final state (see tumor)

Terms and notation

Example situation: Preparing for surgery using a robotically controlled instrument with an endoscope camera. You want endoscope to guide itself to tumor as quickly as possible

Movement choices: How'd we do?

First image (of skin) → Up / Down / Left / Right → +1 for darker image / -1 for lighter image → Second image → ...

Initial state s_0

Policy π

Action a_0

Reward r_0

state s_1

Continue until final state (see tumor)

Optimization Goal: Find policy π* that maximizes total "discounted" reward $\sum_{t \geq 0} \gamma^t r_t$

TL;DR

-> Use a CNN to map images to actions, optimize CNN with respect to loss function that depends on reward *in a recursive manner*

Movement choices: How'd we do?

First image (of skin)

Initial state s_0

Policy π

Up

Down

Left

Right

Action a_0

+1 for darker image

-1 for lighter image

Reward r_0

Second image

state s_1

...

Continue until final state (see tumor)

Optimization Goal: Find policy π* that maximizes total "discounted" reward $\sum_{t \geq 0} \gamma^t r_t$

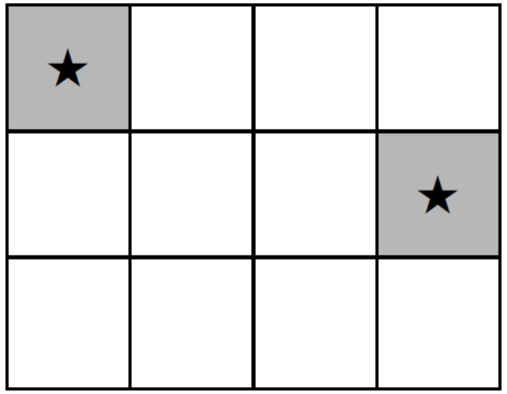
A simple MDP: Grid World

actions = {

1. right
2. left
3. up
4. down

}

states

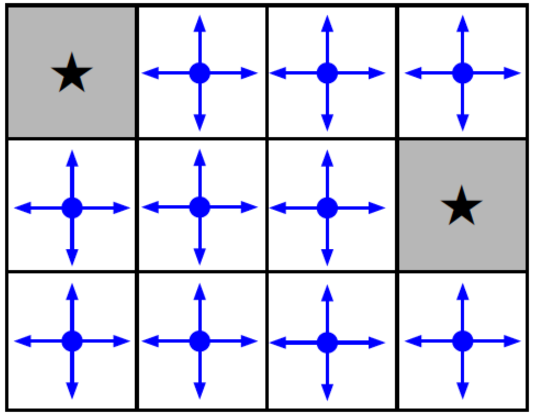


Set a negative “reward”
for each transition
(e.g. $r = -1$)

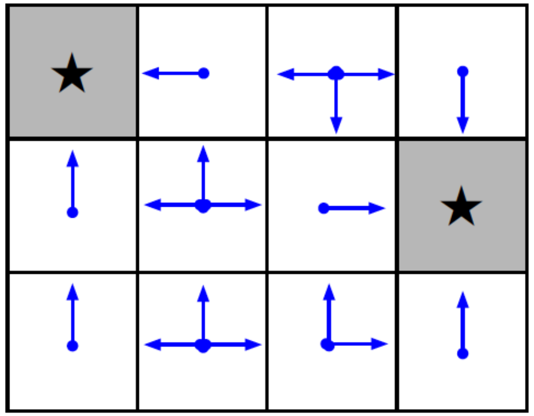
Objective: reach one of terminal states (greyed out) in least number of actions

From Stanford CS231n Lecture 17

A simple MDP: Grid World



Random Policy



Optimal Policy

From Stanford CS231n Lecture 17

deep imaging

Let's jump into the math….

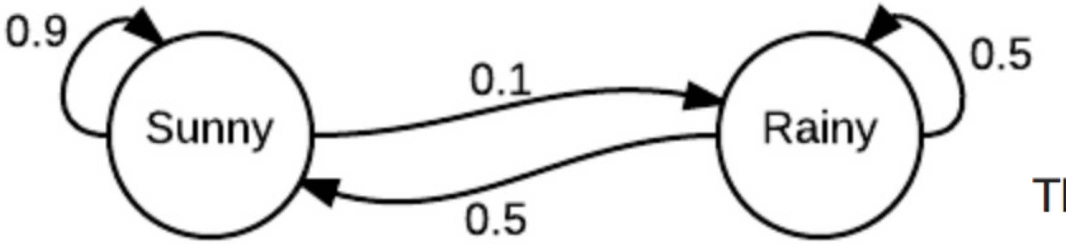
Definition of a Markov process:

$$\Pr(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \Pr(X_{n+1} = x \mid X_n = x_n)$$

Let's jump into the math….

Definition of a Markov process:

$$\Pr(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \Pr(X_{n+1} = x \mid X_n = x_n)$$



$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The weather on day 2 can be predicted by:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} P = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \end{bmatrix}$$

Thus, there is a 90% chance that day 2 will also be sunny.

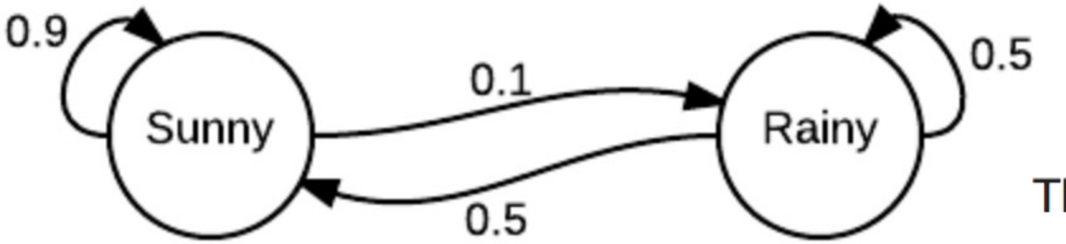
2 states: Sunny and Rainy

https://en.wikipedia.org/wiki/Markov_chain

Let's jump into the math….

Definition of a Markov process:

$$\Pr(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \Pr(X_{n+1} = x \mid X_n = x_n)$$



$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The weather on day 2 can be predicted by:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} P = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \end{bmatrix}$$

Thus, there is a 90% chance that day 2 will also be sunny.

2 states: Sunny and Rainy

Transition matrix – try to learn this from state to state

https://en.wikipedia.org/wiki/Markov_chain

Assume transition between states follows Markov process

$P(s_{t+1}|s_t, s_{t-1}\ldots s_0) = P(s_{t+1} \mid s_t)$

Andrey Markov

Markov chain

$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{T}$ – transition operator $p(s_{t+1}|s_t)$

why "operator"? let $\mu_{t,i} = p(s_t = i)$ $\vec{\mu}_t$ is a vector of probabilities

let $\mathcal{T}_{i,j} = p(s_{t+1} = i|s_t = j)$ then $\vec{\mu}_{t+1} = \mathcal{T}\vec{\mu}_t$

$$\mathbf{s}_1 \xrightarrow[p(\mathbf{s}_{t+1}|\mathbf{s}_t)]{} \mathbf{s}_2 \xrightarrow[p(\mathbf{s}_{t+1}|\mathbf{s}_t)]{} \mathbf{s}_3$$

Markov property
independent of $\mathbf{s}_{t-1}$

Add in dependence on action: Markov *decision* process

$P(s_{t+1}|s_t) \Rightarrow P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_t, a_t, \ldots s_0, a_0)$

Markov decision process $\quad \mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space $\quad$ states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space $\quad$ actions $a \in \mathcal{A}$ (discrete or continuous)

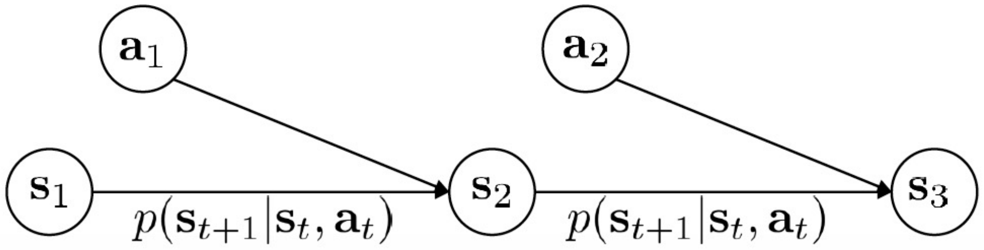
$\mathcal{T}$ – transition operator (now a tensor!)

let $\mu_{t,j} = p(s_t = j)$

let $\xi_{t,k} = p(a_t = k)$

let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$

$$\mu_{t,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$



Andrey Markov

Richard Bellman

Add in dependence on action: Markov *decision* process

$P(s_{t+1}|s_t, a_t)$ can include reward $r(s_t, a_t)$

Markov decision process $\quad \mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

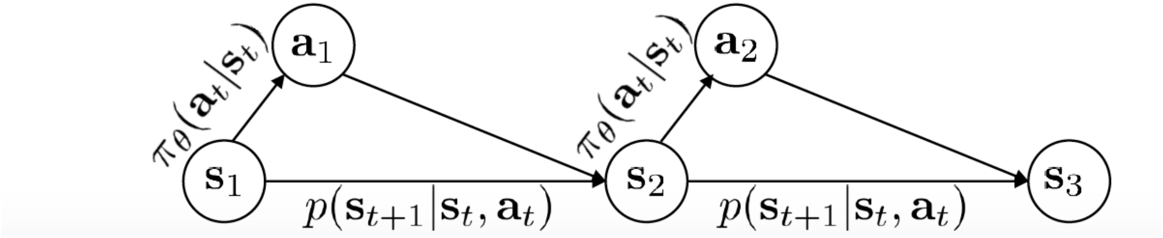
$\mathcal{S}$ – state space $\quad$ states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space $\quad$ actions $a \in \mathcal{A}$ (discrete or continuous)

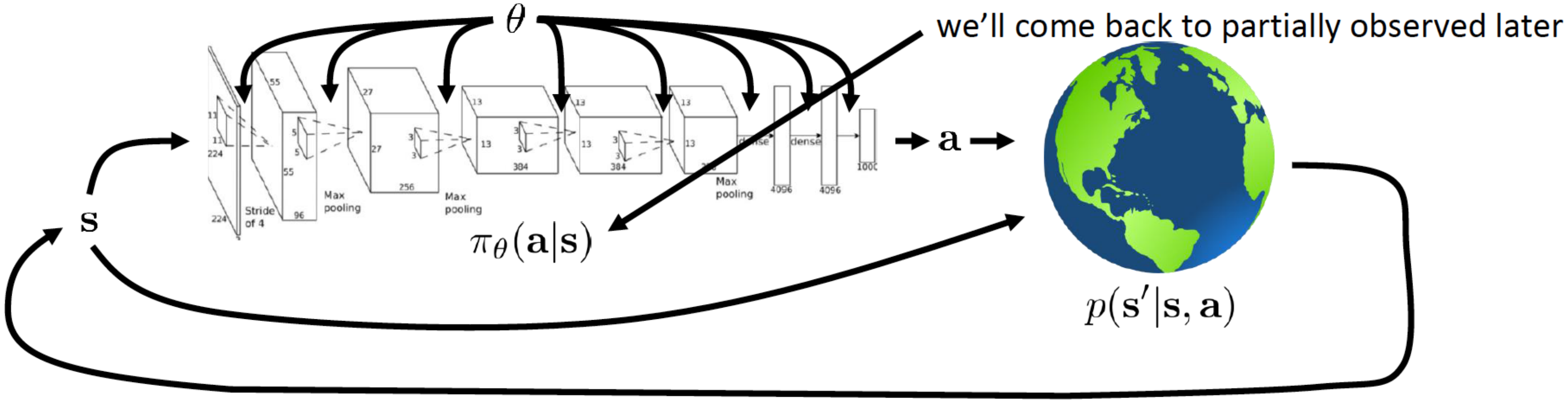
$\mathcal{T}$ – transition operator (now a tensor!)

r – reward function $\quad r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

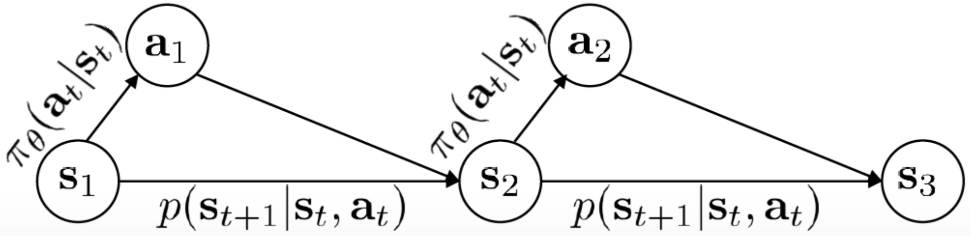
$r(s_t, a_t)$ – reward



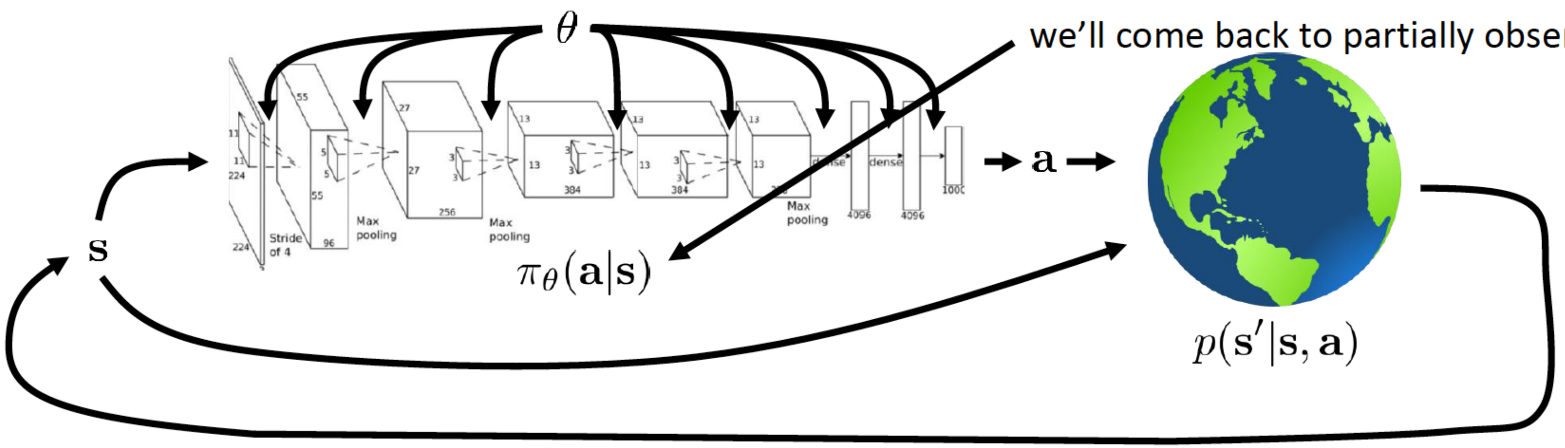
The goal of reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s}, \mathbf{a})}$$



The goal of reinforcement learning



$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\underbrace{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{Markov chain on } (\mathbf{s},\mathbf{a})}$$

$$\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]$$

Machine Learning and Imaging – Roarke Horstmeyer (2023)

The optimal policy **π***

We want to find optimal policy **π*** that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?
Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

Discount factor: accumulate the rewards "acquired" up to current state, but they become less important the longer they were in the past

From Stanford CS231n Lecture 17

The optimal policy **π***

We want to find optimal policy **π*** that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?
Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Don't have access to all policies, so use Q in practice

Bellman equation

The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Q* satisfies the following **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a')|s,a\right]$$

Intuition: if the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$

From Stanford CS231n Lecture 17

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning**!

From Stanford CS231n Lecture 17

Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s, a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s, a \right]$

From Stanford CS231n Lecture 17

Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a \right]$

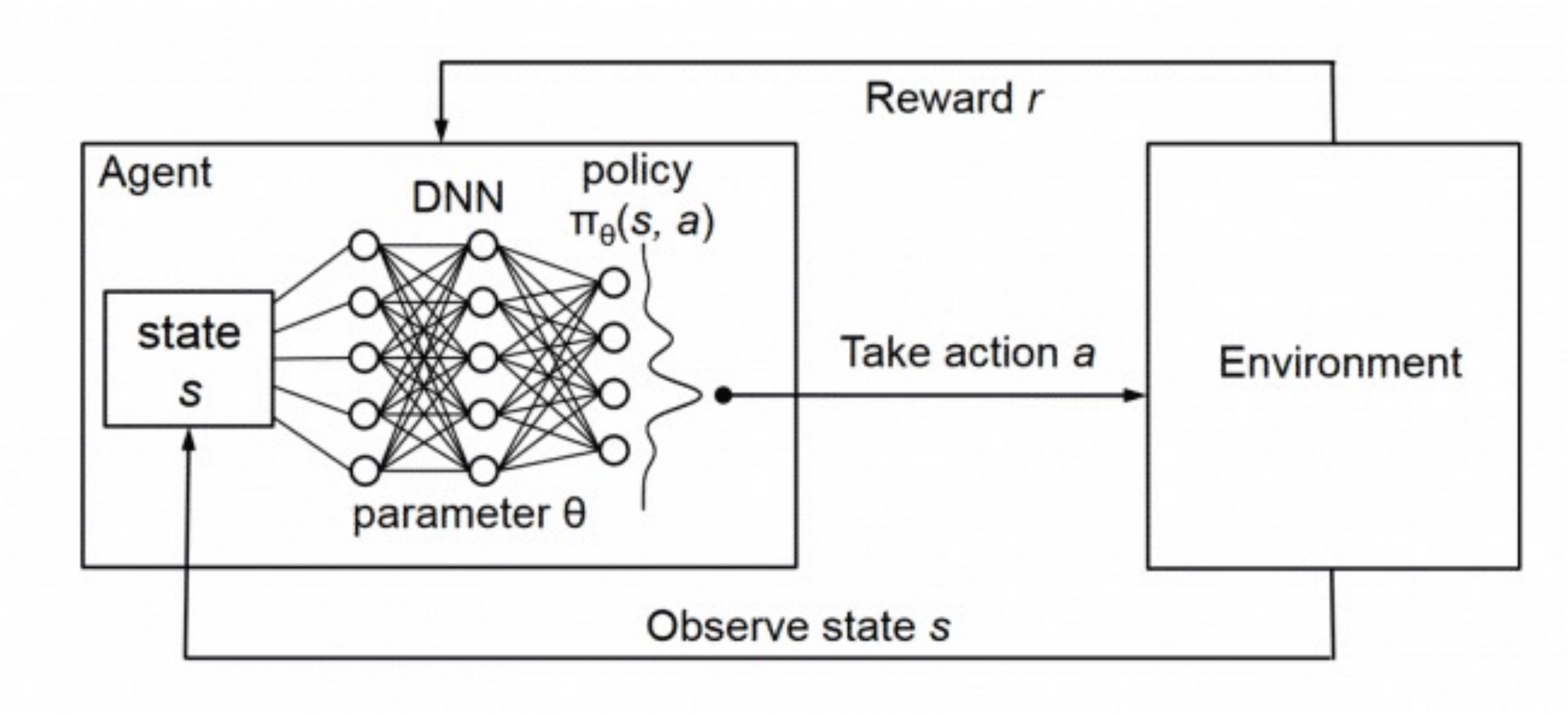
Iteratively try to make the Q-value close to the target value (y_i) it should have, if Q-function corresponds to optimal Q* (and optimal policy $\boldsymbol{\pi}$*)

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i) \right]$$

From Stanford CS231n Lecture 17


Reward r
Agent
DNN
policy
$\pi_\theta(s, a)$
state
s
parameter θ
Take action a
Environment
Observe state s

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Case Study: Playing Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

From Stanford CS231n Lecture 17

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

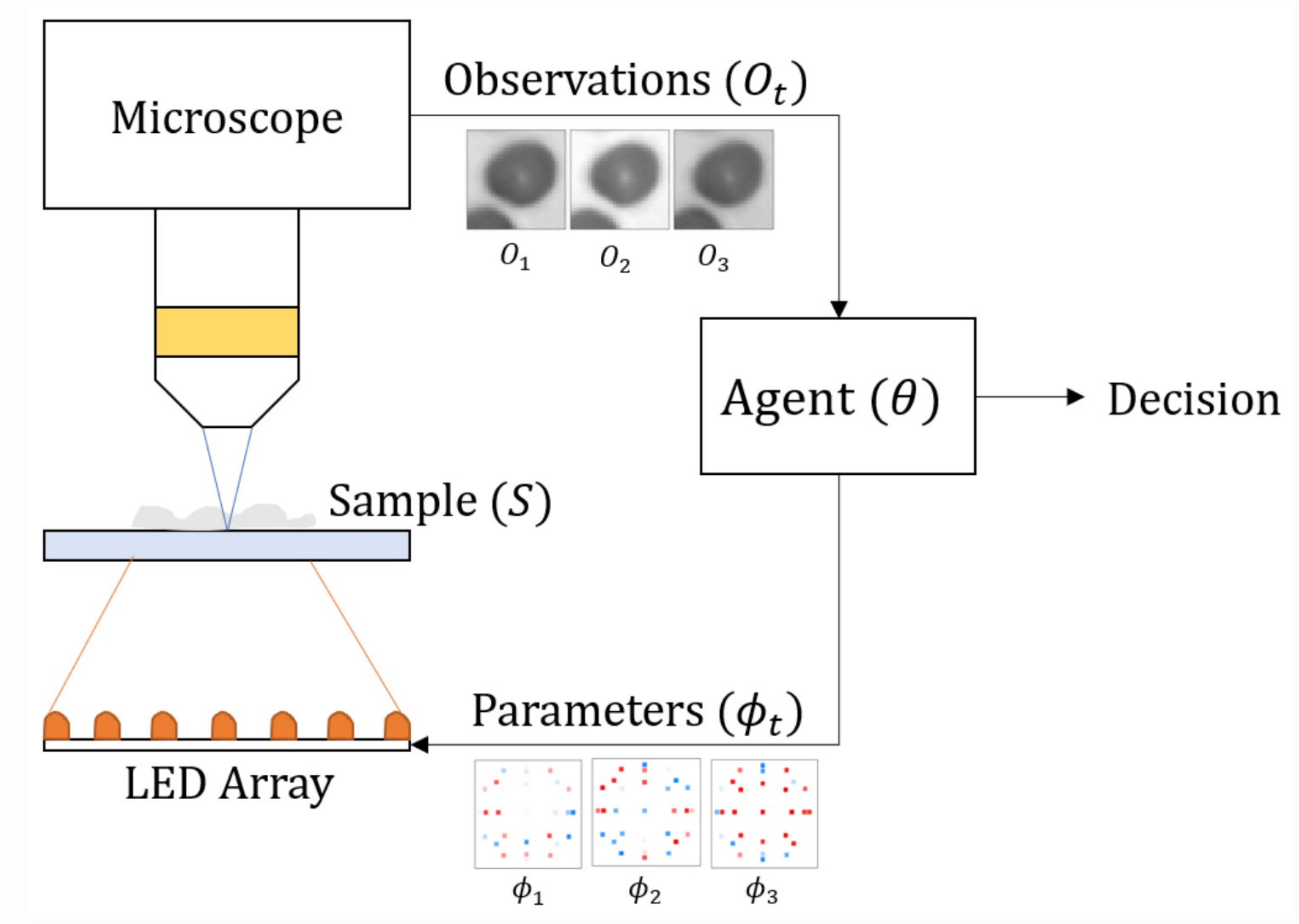
Input: state s_t

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

From Stanford CS231n Lecture 17

https://www.youtube.com/watch?v=V1eYniJ0Rnk

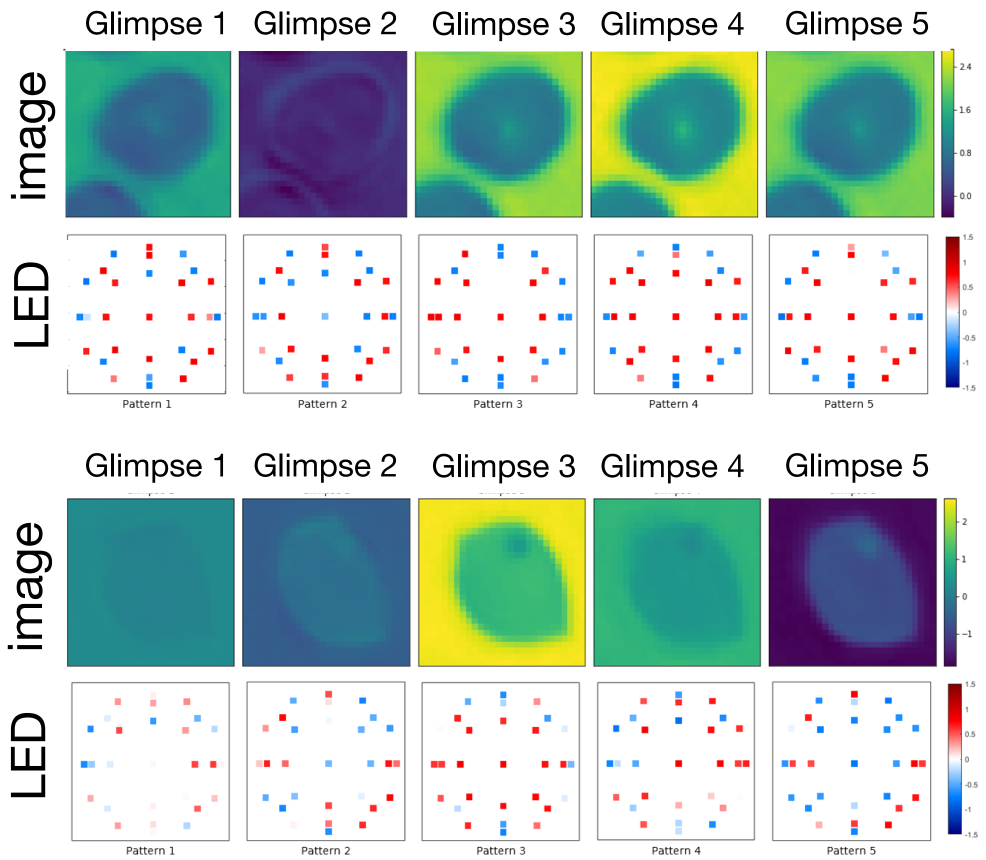
How can this be applied to optimized imaging?



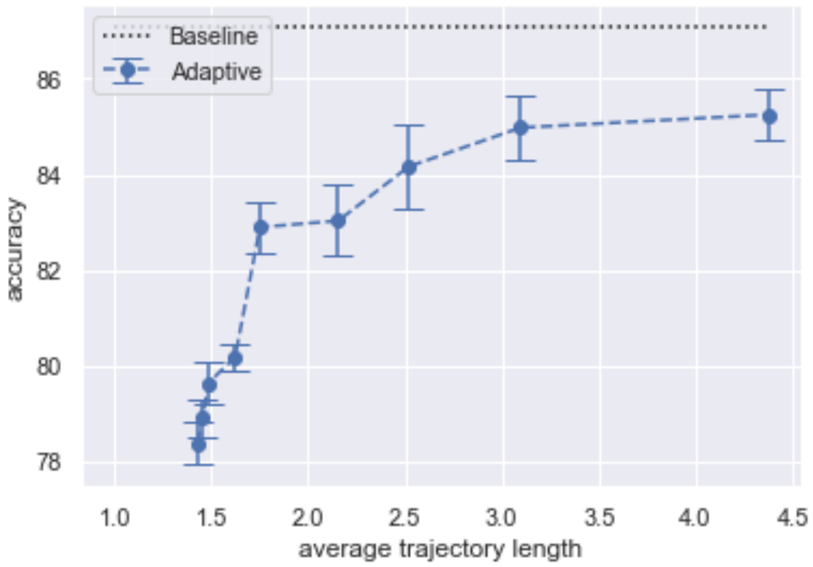
Reinforcement learning for intelligent multi-image classification

deep imaging

Experiment: Malaria-infected blood cells



Accuracy vs. trajectory length, malaria



Classification accuracy increases with more intelligent “glimpses”

A. Chaware et al., “Towards an intelligent microscope: adaptively learned illumination for optimal sample classification,” ICASSP (2020)

Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**

- Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Each transition can also contribute to multiple weight updates => greater data efficiency

From Stanford CS231n Lecture 17