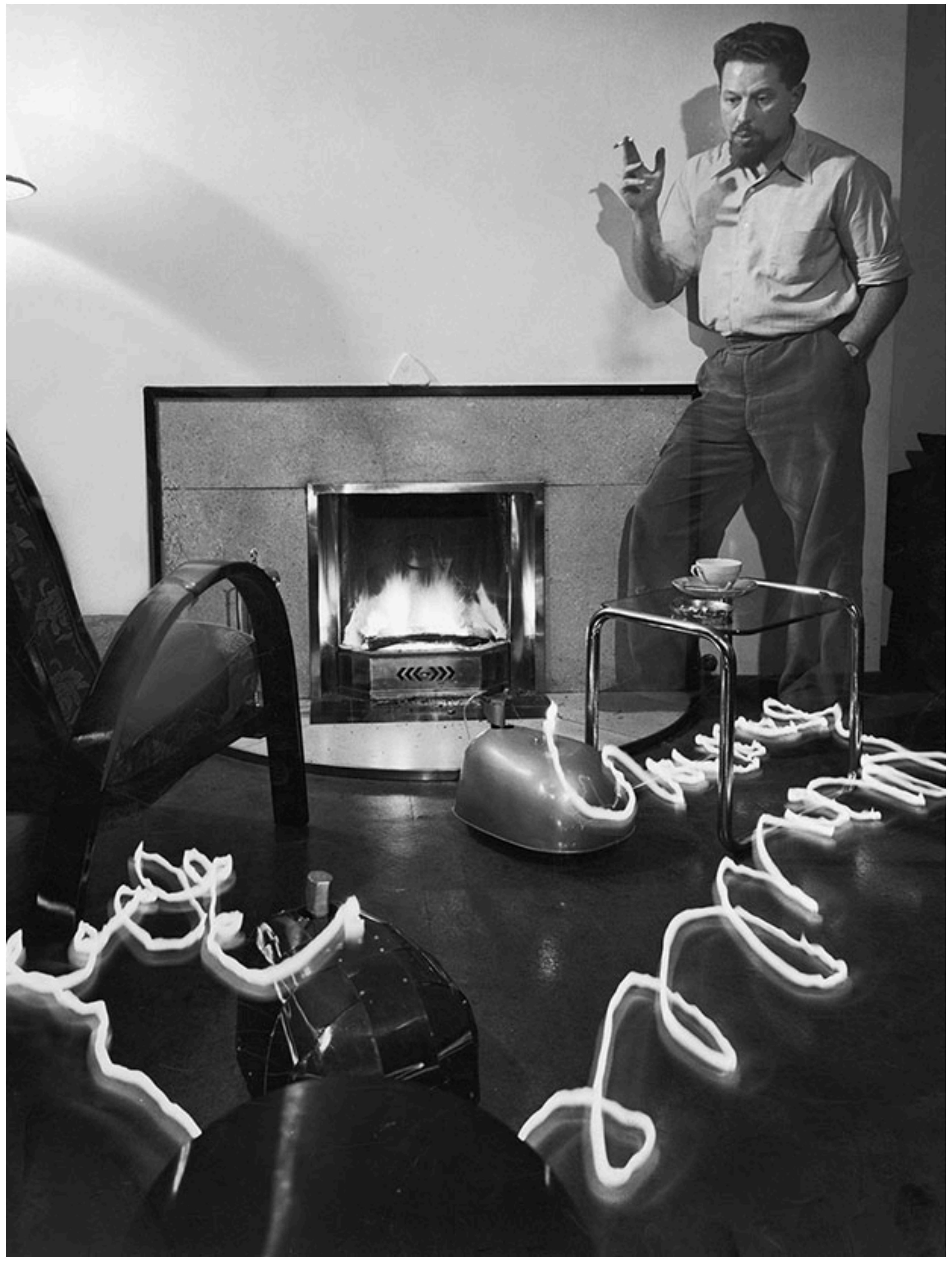


Admin

- Please start picking topics & groups: ask me about projects!
- Presentation signup sheet will be released this week

Major topics of the quiz

- Q1: differences between Bandits and MDPs
- Q2: understanding generalization and discrimination
- Q3: Comfort with features, computing values, and updating weights
- Q4: The role of state weightings and errors with function approximation
- Q5: policy improvement theorem
- Q6: Tile coding, NNs and initialization

Why each topic is relevant for the course

- Policy improvement theorem
- Differences between Bandits and MDPs
    - Test your basic foundational knowledge
- Understanding generalization and discrimination
- Comfort with features, computing values, and updating weights
- The role of state weightings and errors with function approximation
- Tile coding, NNs and initialization
    - Most of your projects will involve function approximation

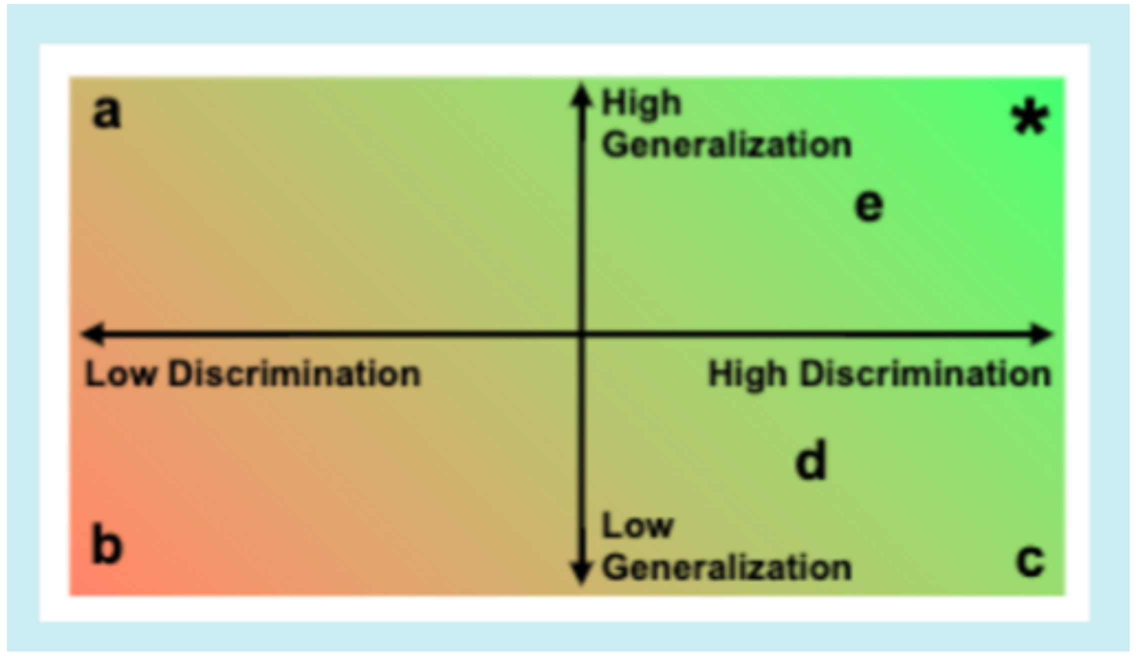
Warning signs

- Did you need to consult the book and online resources on several questions?
- Did the quiz take a long time?
- Did you understand the fundamental knowledge each question was trying to evaluate -or- did you think: what are they asking about here?
- Did you get one question completely wrong? Did you get several questions wrong?
- You don’t want to be learning RL while taking this course (and likely another course)
  - I will not mark your project more lightly because you are learning RL

Bandits and MDPs

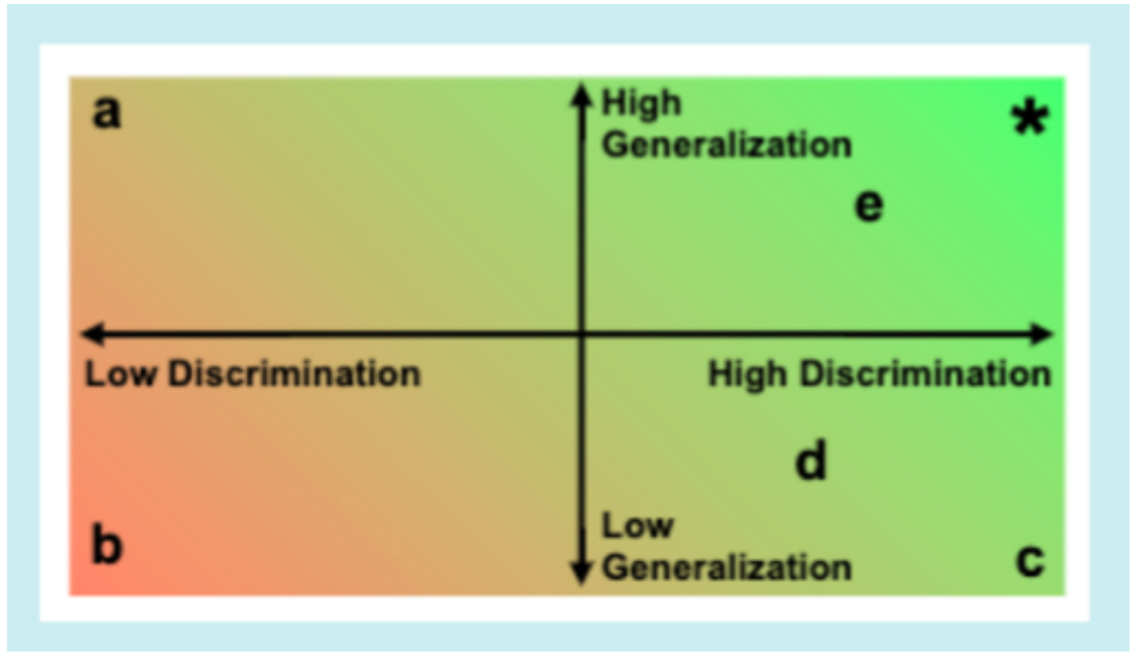
- One important difference between MDP and a bandit:
  - MDP has multiple states, bandit just one
  - Objective: max expected return vs expected reward
  - In MDPs we care about long term reward, bandits are effectively gamma=0
  - In MDPs the next state depends on the previous state and action
  - Other ideas?
- Incorrect:
  - Exploration/exploitation; epsilon-greedy; non-stationarity; number of actions per state
  - Didn’t actually discuss a difference (giving two definitions is not enough)
  - No example of a problem that cannot be formalized as a bandit; unclear/poorly explained example

Generalization and discrimination



- Where is state-aggregation (one bin). Explain
  - one bin for the entire state space will lead to high generalization (across the entire state space) but no discrimination as all states are assigned the same value
- Incorrect:
  - Wrong letter; Invalid explanation
  - Generic explanation of state aggregation not state agg. with one bin
  - Why high generalization why low discrimination!

Generalization and discrimination



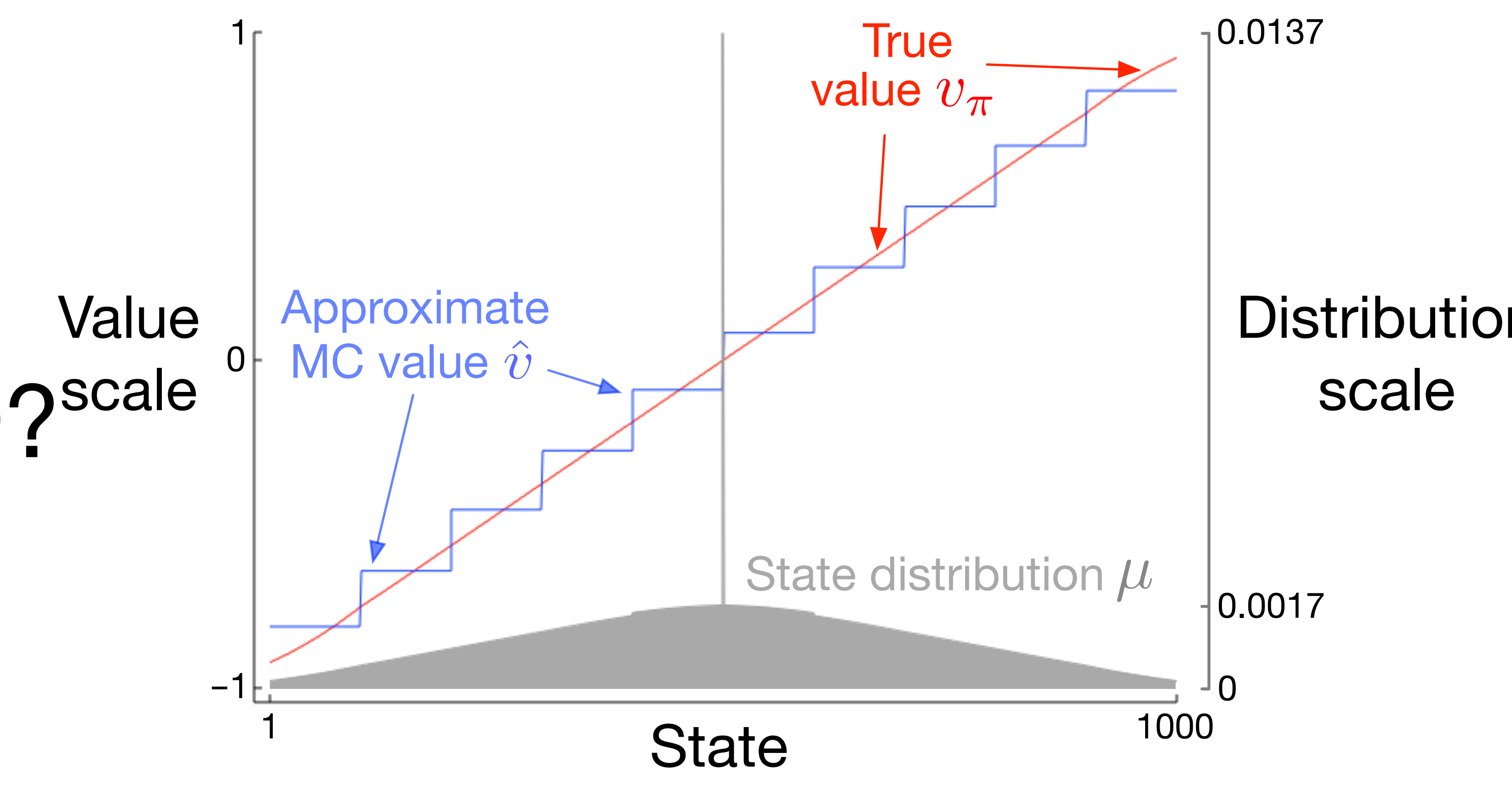
- Where is course coding (many large overlapping circles)?
  - Coarse coding with a large number of large overlapping circles would lead to plenty of generalization due to the circles being large, and plenty of discrimination due to the large overlaps leaving only a small number of states to share estimates.
- Incorrect:
  - Wrong letter; Invalid explanation
  - Generic explanation of course coding or binary reps
  - Why high generalization why high discrimination!

Features, value functions & weight updating

- Give the feature vectors for **hot** and **cold** state
  - x(hot) = [1,1]^T; x(cold)=[1,-1]^T
- Create state-action features (actions are research and advertise):
  - x(hot, research) = [1,1,0,0]^T; x(hot, advertise) = [0,0,1,1]^T
  - x(cold, research)=[1,-1,0,0]^T; x(cold, advertise)=[0,0,1,-1]^T
- What is the value of hot and cold with initialization of 1.0:
  - v_hat(hot, w) = w^Tx = [1,1] [1,1]^T = 2
- Do a Gradient Monte Carlo update from s=hot (you can use that, alpha, and g):
  - Option 1: w = w + alpha (G - v_hat(hot, w)) x(hot)
  - Option 2: w = w - alpha g   //why is it minus alpha?

State weightings and function approximation

- Why do we need state weightings with FA but not the tabular case:
  - In tabular case we have a value for every state
  - In FA we don't we have far less weights than states; we can't get the value of every state perfect so we need to decide which states we care most about
  - Not about off-vs-on policy
  - Nice example of the Random Walk in Sutton & Barto
- What state weighting is used in on-policy semi-gradient TD?
  - on-policy distribution of states under the policy pi, also known as the stationary distribution under pi



Policy improvement

- How do we get the new greedified policy?
    - pi’(s) = argmax_a q_pi(s,a) for all s
- Bonus questions!
    - Q1: What do we know about the relationship between pi and pi’?
    - Q2: If pi’ is the same as pi, then what do we know about the two policies?
    - Q3: If pi’ is different than pi, do we know anything more than what we said in Q1?

Difference function approximation architectures

- How to we optimistically initialize Sarsa with tile coding?
    - w[:] = G_max/m
    - Using 1.0 may not be optimistic
    - Need m in there otherwise values might be way to big…we want equal to G_max
- Why is this difficult to use optimistic init with a NN?
    - all the weights are used to compute the value of any state. This means that when an update is made for one state, it can potentially affect **ALL the weights**. Thus we might start with some optimistic initialization of the weights, but they **might not remain** so as soon as a single update is made.
    - It is not always obvious how to initialize weights in the NN to **ensure we have optimistic values**. For example, if we use tanh activations, then large weight values might result in negative features rather than positive features. Consequently, even if we make weights on the last layer very big, it might actually make values pessimistic for a state rather than optimistic.
    - Not about Xavier …

Roadmap

- Lectures from me:
    - Scientific method
    - How to organize your research into issues, problems and a roadmap
    - Bad practice for evaluation
    - Looking at the data generated by RL experiments
    - Good practice for evaluating agents
    - Good practice for comparing agents
- After that presentations from you, mixed with lecturing from me

Project marking and expectations

- Remember there will be no new algorithms, no new domains in the project:
    - Empirical projects only
- This should remove much of the anxiety about project grades: you are not responsible for coming up with a new algorithmic idea!!
- I will provide plenty of guidance on running good experiments, writing, and presenting ideas and results clearly: **in the draft feedback and in lecture**
    - And that is what you will be marked on
- This is to teach you that a big part of research is working hard, focusing on the details, and producing high quality & clear work
    - Innovation & creativity are only one aspect of RL research

The purpose of (scientific) computing is insight, not numbers

-Richard Hamming

No matter what motivates you, understanding is a key aspect

- **Avoid the trap of building and advocating for things you don't understand**
- Subject your ideas and implementations to **critical evaluation:**
  - design tests to highlight the weakness of your new thing
- Few things dominate all other approaches
- Stress testing your ideas leads to:
  - refinement, new ideas, and future work

Work on what others are *not* working on

- Roughly speaking, my research has focused on investigating things that are not well understood
  - e.g., do we really know which off-policy methods work well in practice?
- This has lead to good insights, new algorithms, and long productive lines of research
- I consider myself a careful empiricist. I think we need many more people like that!

There are many different types of experiments

- Demonstration that something is possible (aspirational problems)
- Benchmarking
- Limited verification of a new idea
- Empirical studies

Demonstrations are a specialist activity

- Often requires expertise in the domain of application
- Often involves large teams, large compute, large time commitment
- Certain amount of marketing and PR involved in this

- Are we (RLAI) well setup to do demonstrations? Is it a good idea for an MSc or PhD project?

Benchmarks: proceed with caution

- Ask yourself: are the problems we use good benchmarks?
    - don't benchmark on tasks not compatible with scientific study
- Benchmarks are hard to design, sometimes we care about relevance
- Focusing on benchmarks can limit an individual's creativity and progress of the field
    - Must have Atari, DMLab, and Continuous Control experiments; else reject
- Benchmarks emphasise numbers and ranking, and de-emphasise understanding and creative experiment design

Tips & Tricks for success in empirical research

Work on ideas not code

- Research is hard work
- Research guarantees lots of failure
- Research guarantees lots of criticism
- Research guarantees lots of rejection
- Working on tools is fun, progress is steady, nobody tells you its all wrong
- Writing code, big software frameworks, and continually tuning your workflow has rapidly diminishing returns
- Most code and frameworks becomes obsolete quickly!
- Your main job as a researcher is to learn about things and generate new knowledge:
  - Code is a tool to aid in this goal

But do optimize your code

- Profile
- Use parallelism
- Use high-performance languages if possible
- Experiments in RL typically involve many repetitions of many different configurations
- If one run (seed) takes 3 hrs to run, you might be in trouble:
  - If you run each seed X setting in parallel you could have too many jobs for the cluster
  - If you don’t you, your experiment could take weeks
- **Ideally you want initial experiments to run in minutes or hours:** this allows multiple experiments a day
- Start with smaller probing experiments, later run bigger exhaustive experiments!

Code frameworks & hardware can stifle research

- Tensorflow forces us to think about fixed computational graphs
- JAX forces us to think in matrices and batches
- These packages come from people trying to optimize a specific set of algorithms for specific hardware:
    - Massive sample inefficient agents with multiple copies of the environment running on GPUs
    - “We have to write the agent this way so it doesn’t waste the GPU”
- Use the tools that accelerate your research and support your ideas
- In the end: general tools (C, GOlang, Lisp, etc) will win!

Focus on getting the first result

- *A classic mistake is to spend too much time:*
  - Setting up the code
  - Manual exploratory probing
  - Doing other things
- Every empirical project has a sequence of key results:
  - A set of graphs that describe the story
- **Try to get the first result as soon as possible**
  - Likely there will be bugs, design changes, etc that force you to rerun it anyway
- The first result shapes the project, and shapes how you plan

There are many components of a good experiment

- How baselines are selected
- How we fairly implement and test the baselines
  - overcoming designer bias is a challenge
- What to measure
  - how many runs
  - what to report
- Setting meta-parameters challenges, pitfalls **(Mike and Martha)**

Repeating what you see in papers is not always OK

Good empirical work requires creativity and thinking

- **You have to think** about the right experiment to run
- Your goal is to **convince yourself** that the results is meaningful, first and foremost
- We **cannot rely on a list of instructions** or rules (including this one) of how to do a good experiment

Closing thought: avoid wasting compute and people's time

- Imagine your solution (agent) or problem (benchmark) requires massive computation to run
- So you decide to report insignificant results (perhaps 3 runs), and not report parameter sensitivity
- You are saying, due to computational reasons, I decided to run a bad experiment
  - have you succeeded and effectively creating and transferring useful knowledge??
- **Avoid the trap of building and advocating for things you don't understand**

Rich's Slogans

1. Approximate the solution, not the problem (no special cases)
2. Drive from the problem
3. Take the agent’s point of view
4. Don’t ask the agent to achieve what it can’t measure
5. Don't ask the agent to know what it can't verify
6. Set measurable goals for subparts of the agent
7. Discriminative models are usually better than generative models
8. Work by orthogonal dimensions. Work issue by issue
9. Work on ideas, not software
10. Experience is the data of AI
11. Don’t be impressed by what you don’t understand

Projects

- Let’s discuss potential projects

- From the doc:
  - Evaluate representation methods (e.g. sparse, predictive, meta-learned and etc) in domain transfer tasks.
    - Testing the effect of nexting on generalization performance (task transfer).
  - Compare representations learned by RNNs vs Generate-and-Test on a semi-stationary (fixed targets but input distribution changes over time) example (Daniel)