UpperBound 25 talk notes

Quick background

Id Software

<picture of early games>

Founder of Id Software in the 90's, wrote the Commander Keen, Wolfenstein 3D, Doom, and Quake series. I take some pride in knowing that Quake drove GPU development and adoption, leading indirectly to the modern AI world. DeepMind's DMLab environments are also built on sanitized versions of Quake Arena.

Armadillo Aerospace

<picture of armadillo rockets>

Overlapped with that, I worked on VTVL rockets for a decade with Armadillo Aerospace

Oculus

<picture of Quest headset>

More recently, I laid the technical foundation for the modern era of Virtual Reality at Oculus / Meta

Keen Technologies

While I was still at Meta, the OpenAI founders tried to recruit me. I was flattered – I wasn't an AI guy.

I did the reading, formed some opinions about the state of things, and decided this really was The Most Important Thing I could work on.

Doing research instead of systems engineering is a notable change for me, but I am enjoying it.

Being joined by Richard Sutton, the "Godfather of Reinforcement Learning" was unexpected. Part of "doing the reading" was reading his textbook. Twice. We have a lot of aligned views, with some healthy differences of opinion.

We now have six researchers on the team, split between academic and industry backgrounds.

Where I thought I was going

Not LLMs

I consciously chose to not be involved in LLM work. What LLMs have been accomplishing is awe inspiring, but I think it is broadly unappreciated outside of specialists how an LLM can “know everything without learning anything”.

Fundamentally, I believe in the importance of learning from a stream of interactive experience, as humans and animals do, which is quite different from the throw-everything-in-a-blender approach of pretraining an LLM. The blender approach can still be world-changingly valuable, but there are plenty of people advancing the state of the art there.

Still, I am at least a little wary that transformer based foundation models might be able to subsume traditional RL. Can you ask an LLM to carefully learn how to perform well in an environment never seen in its training set? Even if grotesquely inefficient, that would be an important Bitter Lesson.

Games

My background obviously biases me towards working with games and virtual environments.

PC games had long had “bots” that played at superhuman levels, often used for cheating, but that was looking directly at internal structures, not at the same pixels that players saw. I remember long ago talking about how cloud game streaming could eliminate all the common multiplayer cheating hacks in gaming, because nobody could actually make a program to play just from the pixels. I said that if someone could do that, they deserved to win!

DeepMind’s Atari work did exactly that in 2013.

Given just the output images from a game emulator, the models learned how to play dozens of games well enough to make impressive videos.

However, when you looked closer, there were a lot of caveats.

The standard amount of experience reported in the literature is 200M frames at 60 fps, which would be playing 24 hours a day for over a month. Equalling the performance of a random grad student learning a game in an hour or so doesn’t sound so impressive. Some of the games needed more specialized architectures and BILLIONS of frames to approach the human baseline.

On the other hand, this is like a baby opening its eyes just after birth and learning to play a game in a month, which actually DOES sound kind of impressive.

Work has continued for the last decade, but continuous, efficient, lifelong, single environment, multi-task online learning is still far from a solved problem. *Fundamental things that cats and dogs do are still not replicable by any system today.*

Video

My initial thought was that passively learning from TV content with a moving center of attention would provide a better foundation for game learning – if you understand object permanence, it should be much easier to learn what is happening in video games, versus analyzing full frame patterns.

I was aiming to provide a virtual learning environment composed of an infinitely scrollable “video wall” of mixed passive and interactive content that an agent could migrate between based on its own curiosity.

Missteps

Too low level

I am traditionally a low level guy, so I started in C++ writing CUDA kernels directly, then using cuBLAS, then cuDNN, then finally moving to PyTorch. I still have an eye on how things could be implemented at the lowest level, but the higher level tools are undeniably powerful for rapid experimentation.

Avoided larger experiment too long

Spent too much time with notebooks and local GPUs before just going to Visual Studio Code on remote hosts. We have been happy with Lambda Labs.

Sega Master System

I initially started using Sega Master System games as a step up in sophistication from Atari and to just be a little contrarian, but I eventually decided that there was too much value in comparing experiments with the large body of work on Atari games.

Video can wait

There are plenty of fundamental problems just playing multiple games, it isn’t necessary to bring passive video understanding into it.

Settling in with Atari

Lots of critical research is still open!

Unbiased and diverse

Commercial games are great because they are unbiased by researchers. When you build a bespoke environment to test your pet research idea, you will inevitably make it easier to look good.

The wide diversity of gameplay is a strong plus. Why does Qbert stall here? Why does Atlantis start to lose performance there? Why is this technique beneficial here and not there?

Not perfect

Drawback: sprite flicker from old hardware. One of the things I liked about the Sega Master System emulator was that you could turn off accurate emulation of the sprites and CPU speed so the games always ran smoothly and looked solid.

Drawback: deterministic behavior can be exploited. Sticky actions are important to prevent the deterministic nature of the simulator from being exploited unhelpfully. Unfortunately, Atari 100k results are for deterministic actions.

Deep research history

We have a large body of prior art.

Maybe RL frameworks aren't such a good idea – even things just a few years old can be trouble to run.

Use ALE directly instead of wrapped through Gymnasium.

Isn't Atari solved?

Have the latest models like MEME, Muesli, or BBF "solved" Atari?

Given hundreds of millions of frames, most of the games are now around or above the top reported human scores. There is still room for improvement, but I'm not super interested in it.

The more recent push towards learning much quicker with the Atari 100k benchmark is interesting. BBF and Efficient Zero have demonstrated dramatic improvements in data efficiency, but with some caveats.

Noisy results – random seeds and random play, wide range of episode lengths: Full Pitfall! games take 20 minutes, or 72k frames.

Scatter plots vs averaging and confidence intervals

Final evaluation versus scores during training, smoothing scores during training biases low

At very high scores, details around evaluation matter a lot. Perfect breakout scores across many random seeds in MuZero are questionable, especially with sticky actions.

Why not Pokemon or Minecraft?

Scrolling games and 3D games will certainly bring new challenges, but there are open research topics even with the simple Atari games; making things more complicated isn't going to help solve them.

The extra complications also make the lure of looking at internal data structures instead of just pixels very strong. Cheating!

Reality is not a turn based game

<picture of Go board and a soccer game>

Stop treating Atari like a turn based board game, where the environment waits patiently for the agent to provide an action, then immediately performs the action and shows the results.

The real world keeps going, regardless of if you have finished processing your last observation.

Inverting the RL software environment – the agent gets called, rather than the environment. Works for both a virtual simulation and a physical environment.

Single environment

Some agents are trained with many environments running in parallel, and it is important to understand that this is an easier task than training a single environment, because you have more diverse samples from every update.

Larger batches of running environments should still learn much faster, and that is important for training an AGI, but if a batch-1 run fails to learn effectively, something is wrong with the algorithm relative to biological intelligences, and we should figure out how to fix it.

Speed

The first key issue is that you can only act as often as your policy can be evaluated. Reasonable implementations of classic algorithms can hit 15 fps / action repeat 4 on consumer hardware, but some of the new data efficient systems are orders of magnitude too slow.

I'm fond of CUDA graphs.

Overlapping training with the environment and possibly policy work using cuda streams with different priorities and explicit synchronization

Latency

The second key issue is that the action you select won't have any effect for a significant time after the observation was generated. It is worth trying out one of the many web based reaction time testers – you will find that you average over 160 milliseconds.

Most classic model-free RL algorithms adapt fine to added latency, degrading relatively gracefully – breakout quickly, MsPacman more slowly. However, many of the modern high performance algorithms fall apart when the action doesn't take effect immediately, because their prediction models assume a board-game-like action-effect behavior.

Much of the performance of these algorithms could be recovered by hard coding a matching delay queue for the conditioning actions, but that is avoiding the problem to be addressed.

You can add a "delay line" to the actions in most RL frameworks with only a couple lines of code, and I encourage everyone to try it out, but it is an open question what other challenges "reality" has to offer.

Physical Atari

A reality check for people that think full embodied AGI is right around the corner is to ask your dancing humanoid robot to pick up a joystick and learn how to play an obscure video game.

<picture of physical atari setup>

This is an Atari 2600+ game system, with a camera looking at the screen, servo motors manipulating the joystick, and a laptop running a reinforcement learning agent in real time.

A real classic Atari 2600 on a physical CRT from 40 years ago would have less system latency by avoiding the emulator inefficiencies in the 2600+, but that isn't a dominant part of the challenge.

We will be open sourcing this work.

Game selection

<picture of games>

We are testing on eight different games: MsPacman, Centipede, Qbert, Defender, Krull, Atlantis, UpnDown, and Battle Zone.

Many Atari games require you to pull the reset switch on the console, but some allow you to just press the joystick button to restart after game-over. We could clamp another servo to the console, but there are enough games that restart-on-fire to work with.

Camera observations

The full Atari resolution was only 160x210, and almost all research work downsamples even farther, so it doesn't take a super high resolution camera. Learning works fine with just 640x480 resolution, but identifying April Tags and reading scores can be helped with 1920x1080 resolution.

Many USB cameras jpeg-encode their images, which both hurts the image quality and adds latency. Avoid.

Similarly, it is fun to consider how much of the camera scan could be overlapped with multi-layer CNN processing. Conventional camera interfaces wait until the entire image has been scanned out before delivering it, which means the pixels at the top are 16 ms more stale than the pixels at the bottom. Nvidia GPU Direct for Video offers the ability to

Scan line issues due to unsynchronized camera and video scanout haven't been a problem

Screen rectification, april tags. You can learn without any special cropping of the screen, but if you want it to retain any knowledge across camera position changes, you need to identify what parts of the screen matter to the model.

Lighting can be a significant issue for the tag identification.

Joystick actions

USB IO board to drive joystick

Robotroller: Khurram did a great design that used servo motors to control a real joystick. This adds more latency and variability to the behavior.

Robotics reliability – multiple servo failures and joystick failures.

<pic of Khuram's position-to-position latencies>

Action factoring issue – side and fire for Atlantis

Most Atari RL models don't include the last selected action as part of the new observation, but for cases like this it can be important.

Up, Up, Down, Down, Left, Right, Left, Right, B, A: Konami code as another example, and also an example of the limits of learning.

Score and life detection

Surprisingly, the trickiest thing about this project turned out to be reliably reading the score, which is necessary to provide the rewards that RL algorithms need.

Isn't this solved since MNIST?
You can ask a multimodal LLM what the score is on a screenshot and it will tell you.

We have several different score readers, and we were still concerned that they were going to misbehave in public here.

Some games were culled due to the score not being always visible:
Yar's Revenge has the score hidden most of the time
Berzerk moves the score around and displays different things
Qbert has the score hidden when the player is at the top, but that isn't fatal.

Atari dev box

Emissive April Tags for rectification and April Tags for score.

Sparse rewards / Curiosity

The dense rewards that RL algorithms like are the exception, rather than the rule for most tasks. We want an economically valuable agent to carry out long sequences of actions with just a reward at the end.

People don't actually look at the scores going up as they play very much. In some games like Yar's Revenge, the score is only visible between levels!

Some relative game scores are bad information: breakout and space invaders points for higher targets.

Hard exploration games

<picture: Montezuma's Revenge and Pitfall>

The “hard exploration games” like Montezuma’s Revenge and Pitfall have been a longstanding research topic, and progress has been made, but there are orders of magnitude yet to go for human efficiency on them, and it isn’t clear we are on the right track.

We can turn every game into a hard exploration game by only giving a total reward at end of game or loss of life.

Intrinsic rewards driven by artificial curiosity. It might wind up learning how to read the score digits visible on screen as an intrinsic reward.

Meta-curiosity

choosing to play different games

If a human was in a room with nothing but atari games, how would we expect them to behave?
Playing one game for a while, get bored, then switch to another game.
Some games are frustrating to progress on, and are abandoned.
Rewarded for getting a new top-10 score on a game, proportional to how much it improves the average human-normalized score
    Need some tail-off for games that get massively superhuman performance
    Need to keep the best average, so there is no incentive to score badly for a while, then score well
Need to force trying other games?
Minimum play time? Make change decisions at end of games?
Can agents rage quit?

Sequential multi-task learning

Sequential is much harder than parallel

There have been demonstrations of a single agent that learns many games in parallel, but an agent should be able to get competent at a game, then spend millions of learning steps on other tasks, then return to the game and very quickly return to the previous performance. Humans retain good fractions of skills mastered years ago, even with no intermittent practice.

This does not happen today with online learning – the old game starts out at almost random play performance when returned to. It isn’t completely lost, and is re-learned faster than initially, but it is unacceptable.

Neural network training has a very strong recency bias, and tends to suffer from “catastrophic forgetting” of older learning.

Updating anything in a conventional model essentially changes everything by at least a tiny amount. You can hope that when learning a different task that the weight updates will be mostly balanced random noise, but even random walks get destructive after billions of steps.

Richard and Joseph are fond of a “streaming RL” ideal that eschews replay buffers, but I am skeptical that they will be able to learn diverse tasks over days of training. Biological brains certainly don’t store raw observations that can be called back up, but long term episodic memory does feel distinct from simple policy evaluation.

Reduce the damage: learning rate adjustment, sparsity, error correction

perfect memory of all history is viable, we can store terabytes of observations

Model based RL vs Model free

No task ID!

Feeding in an explicit task ID is cheating. There is no moral difference between getting to a new level in a game and switching games.

No ramp alignment!

Any kind of hyperparameter ramp across learning for exploration or training needs to be implicit instead of explicit.

Offline RL

Old games in your memory are an offline RL problem.

Even if you have a perfect memory of everything, RL algorithms can go off the rails if they aren’t being constantly “tested against reality”. Offline learning is hard.

Why doesn’t offline optimization work better?

The difference between online (traditional) RL and offline RL is that online RL is constantly "testing" its model by taking potentially new actions as a result of changes to the model, while the offline training can bootstrap itself off into a coherent fantasy untested by reality.

Big LLMs are trained today with an effective batch of millions and random samples. I propose that such huge batches are not just an unfortunate artifact of cluster scaling, but actually necessary to generate gradients sufficient to steadily learn all the diverse information. You definitely can’t train an LLM by walking online through a corpus, even given infinite time – it will forget each previous book as it “reads” the next. Even if you did IID sampling of contexts with

batch 1, you would see massively sublinear performance, because there might be a million optimizer steps between references to two different obscure topics.

Transfer Learning

Don’t play like an idiot

Watching even a state-of-the-art agent learning a new game, it plays like an idiot. This is understandable for the “infant opening its eyes for the first time” view, but at some point, after weeks spent mastering dozens of games, it should be able to pick up a new game much more effectively.

Gotta Learn Fast

OpenAI’s Gotta Learn Fast challenge with Sonic tried to encourage this, but the winners just learned from scratch in the available 1M steps.

GATO

GATO showing negative transfer learning – learning a new game is harder if done after learning other games in parallel.

It is a challenge to just not get worse!

This “loss of plasticity” in a network with continuous use is a well known phenomenon in deep learning, and it may even have biological parallels with aging brains at some levels – old dogs and new tricks. But humans

It may be necessary to give up some initial learning speed “learn slow so you can learn fast”

New benchmark

A new benchmark should be created that cycles through a set of games and is judged based on all the scores in the last cycle, so raw learning speed is coupled with avoiding forgetting and taking advantage of transfer learning. No minimal action set. Sticky actions.

You have to at least beat BBF learning from scratch!

You need to be careful about switching games in the middle of an episode – it can look to the algorithm as if they just died for no visible reason, which will cause them to hallucinate a reason. Need a “truncation” signal as well as a “termination” signal.

Plasticity vs generalization

Are they in tension?

This is hard, because we really don't have strong theories of either, but it feels like there might be an intrinsic conflict that should be actively addressed instead of passively accepted.

Generalization is an ignoring of details, while plasticity involves recognizing new patterns that are currently meaningless to you.

Primacy bias, recency bias

Facts versus understanding -- the products of reinforcement learning in policies and value functions are sort of fundamentally different from simple traces of environment interactions.

You mine facts for understanding

Generalization is not well founded. CNNs have some helpful inductive bias.

Exploration

Selecting a random action from the entire action space can’t be right.

Action space factorization

<pic of Xbox controller>

Modern game controller has billions of combinations even ignoring analog

Surprisingly small impact of 18 action space vs 4 — quickly settles on a subset. Still has implications for exploration.

Even a small chance of picking a random action severely limits scores in many games. With a frame_repeat of 4 and a fast incoming ball, going left instead of right in Breakout is often a loss of life.

High scores are dependent on having very low epsilon-greedy values – 0.9 scores much worse than 0.99. Playing Qbert, half the time you are at the edge of the board where randomly pushing the joystick in the wrong direction will lead to loss of life. Games that need a fire button to start can get stuck forever if greedy 1.0 and a bad value function.

Explicit policies or a softmax greedy with temperature.

Options? Learn a set of actions, as we learn a set of features

Confidence

Confidence in values versus a distribution of possible outcomes

Distributional values, exploring a chance of a high reward

Dropout values

Timescale

Selecting actions for a single time step can’t be right.

The characteristic vibrating behavior of an untrained deep RL model.

Frame skip

Small action gap problem with 60 fps

Games like breakout benefit from acting at 60 fps, but even with the discount adjusted, learning at 60 is worse due to the small action gap.

Variable timescale options are likely important.

Recurrence vs frame stacks

Frame stacks are unfortunately effective as a state representation for Atari, making recurrent state learning less appealing.

Brains are recurrent neural networks, it seems unwise to ignore that.

Transformers with large contexts work remarkably well for minibatch training, but we still don't have the algorithm for general recurrent online learning.

Forward differentiation

Reversible networks

Prefix sums

Actions as notes to self if you feed the actions in with observations

Behavior of function approximation is almost everything

“Function approximation” is just a black box in classic RL, but the implementation dominates performance.

Neural networks are doing several things at once for us:

Learning new results for novel inputs.
Generalizing results across “similar” inputs.
Averaging together samples from stochastic processes to give an expected value.
Updating previous values for non-stationary processes, as with improving policies. “Concept drift”

It is amazing that it works as well as it does!

The Q values for every action not taken are a result of the model generalizing from what did happen to what might have happened. This is outside the realm of RL algorithms, and purely a matter of model architecture and training.

A hash of observation pixels, or even no observation and just a history of actions, can learn in a deterministic environment, but becomes useless with the slightest amount of noise or when starting in an unseen state.

I think it is underappreciated that EVERY weight update changes EVERY value output for EVERY observation. We just hope that the changes are positive generalizations or at worst random noise that averages out.

The interplay between initialization, activation, loss function, and optimizer may have better combinatorial sweet spots than current practice.

Adam is surprisingly hard to beat. Lucas has tried a dozen hot new things without any clear advantage across a suite of games.

The fact that resetting networks can lead to dramatic performance improvement in BBF is a clear sign that something fundamental is not right.

Overfitting / generalization

Are floating point calculations and backprop gradients even the right base to build on? Scary to consider stepping away from standard practice.

Auxiliary losses

like Self Predictive Representations are just ways to coerce the function approximator into working better for values and policies.

The correct action in breakout is mostly determined by the dozen or so pixels representing the ball and paddle, not the many thousands of other pixels.

IID assumption. High correlation. Non-stationary as “distribution shift” (getting to new levels, trying different tasks) and “concept shift” (getting better in same situations).

High noise in value — is a difference a result of a behavior change, a failure of generalization, or just noise? Rewards may be hundreds of steps in the future, and any of the policy choices between now and then might be crucial, yet we bootstrap values far back into the past.

Looking at it from a tabular RL perspective, it is surprising that Atari works at all. Best action for a random policy is often the best action for the optimal policy?

Extrapolation is going to be wrong more often than interpolation. Unclamped relu can be dangerous.

Dropout should be useful, but hasn’t been yet. Same with weight decay and decay to initialization.

Value representation

Even for a policy gradient algorithm, you still have a state value. It seems unlikely that brains are representing scalar values over many orders of magnitude. One of my pet ideas is to use polar values with rotary wrap-around, since all you care about is which action is best, and all actions will usually be close in range.

Classic DQN clamping

Clamping rewards at -1/1 for stability and to allow the same learning rate to work well for all games.

Ms Pac-Man single dot is 10 points, while bonus items can be thousands of points.

Categorical representations

Categorical representation, values vs full distributional

Stop regressing stated that a simple categorical value with a gaussian spread worked better than a full categorical operator, which seems surprising.

Probably still not the best because it ignores the ordinal nature, something like earthmover loss might be better

MuZero quadratic value compression

Doesn't interpolate linearly, and still has a range hyperparameter.

MSE was all we needed?

Somewhat surprisingly, not clamping rewards works fine with MSE loss, even when the losses are in the millions on some games – nothing explodes, even without gradient clamping or Huber loss, but learning is slower.

Scaling experiments

Adam/RMSProp normalization, SGD for the final layer, weight norm on earlier layers.

There may be challenges with multi-task across games with huge point differences – Pong with Video Pinball.

Conv Nets

Still seems to be the architecture of choice for image digestion.

Big image nets don't work well for RL

Twice I have replaced my simple strided CNN with some of the large scale categorization models from the pytorch library like ConvNeXT, and despite being a bigger model that is state of the art for many classification and backbone tasks, seen it perform worse for my QV RL tasks. Classification performance for a model does not directly correlate to online RL performance. Figuring out exactly how the needs diverge may be worthwhile at some point.

Kernel subsets

Can we learn better function architectures? CNN kernel subset failure example: A 5x5 kernel is a strict superset of a 3x3 kernel, yet it often learns worse, even with more parameters. Weight decay should reduce the impact of noise features, but it can take a very long time.

Parameter sharing increases performance

CNN shared weights are biologically implausible, but sort of like moving a focus point around.

Testing with unshared CNN parameters was worse, at least in the few-million-steps regime. At the limit, unshared should be better, and you could transition.

Factored 1D CNNs

Factoring CNN to 2x1D should help more, and did in CIFAR-10, but not for RL.

Isotropic CNNs

CNN's aren't fully position independent as soon as you have a pooling layer or strided conv. Subpixel offsets.

Edge behavior, standard MaxPool doesn't support wrapping options
Confidence – "I don't have a good idea what the value should be, because I am focused on the wrong corner to see the player"
Partial isotropy

Dilated isotropic CNNs

can exactly calculate all possible image offsets efficiently.

Isotropic DenseNet CNNs

I'm fond of DenseNets, where each channel has access to everything before it, not just the channels of the previous layer. Not obvious how to extend across changes in image size, but clean with isotropic CNNs.
"Activation optimal"

Recurrent Isotropic Semi-Dense CNNs

No frame stacks, evaluate one convolution every frame. Latency increases with distance from each pixel.