Representing high level Knowledge with Predictions - An implementation with Minecraft [DRAFT]

David Quail
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
dquail@ualberta.ca

Abstract

This paper looks to accomplish two goals. First, it lays out a framework for using a layered network of predictions, based on raw sensorimotor data, to represent knowledge. These "predictions" are represented using general value functions - a function common in reinforcement learning, which allows the agent to ask a question about its raw sensorimotor data, and then based on experience, begin to learn estimates for these questions. Secondly, we look to conduct experiments using Minecraft whereby an agent can sense a wall, and move forward or turn, to begin to ask questions about its environment using the simple sensorimotor data. The questions respective answers begin to become more and more abstract and sophisticated as the GVSs are layered on top of each other. In a sense, this will offer insights into how the agent can not only represent, but learn these representations.

This implementation demonstrates how an artificial intelligence agent can build useful abstract knowledge from its raw sensorimotor experience alone using a network of General Value functions. This paper shares the design of such a learning agent and preliminary results.

Introduction

A significant challenge in Artificial Intelligence is connecting a representation with what it represents. "Despite decades of research developing ontologies and datasets for encoding information that most children take for granted, no representation has truly emerged capable of capturing the richness of a child's understanding" (**?**). As human agents interacting with the world, we draw upon our experience with the world to form our knowledge. We know that bananas are yellow, pillows are soft, and that glass will shatter. This knowledge is learned through our interactions, rather than being explicitly told to us by a teacher. These bits of knowledge are represented in an accessible way and can be used for more abstract and deeper reasoning.

AI researchers tend to consider "knowledge" as symbols bound together by a network of relationships, and "reasoning" is an application which leverages these symbols to make decisions. Yet, these symbols tend to not be rooted in the most universal and basic of human knowledge: our sensorimotor interaction with the world.

General Value Function (GVFs) may be the right representation for such real world knowledge. These GVFs are rooted in the most primitive sensorimotor observations from the environment, and can be layered to form increasingly more abstract knowledge. Such a view of learning is often referred to as constructivist. In this constructivist life long learning agent architecture, the GVF is the single mechanism to bridge the gap from low level sensorimotor interaction to abstract knowledge.

Low level sensorimotor data.

All animals seem to be born into the world knowing little, but having an ability to learn facts and abilities about things it experiences in life. These animals only link to the real world is through its interaction with the environment. The knowledge it has derives, ultimately, through this stream of sensorimotor data (vision, touch, hearing). As it interacts, the baby will need to leverage its experience to build and refine its knowledge. From its ears, eyes, nose and skin, which at the beginning may seem confusing. Yet, using only these raw senses, the baby animal begins to represent knowledge and skills little by little. This knowledge, perhaps, is achieved by noticing regularities and relationships between its actions and sensorimotor observations. For example, a child may cry and feel the experience of being picked up or fed. As these experiences are reinforced, they become predictable. The baby becomes to understand the world through these predictions.

Using GVFs to form increasingly more abstract levels of knowledge, results in an architecture which uses one and only one mechanism to form knowledge - namely the GVF. This same unit is used at different levels, the lowest of which is sensorimotor data, to form more abstract knowledge. This approach allows the agent to build knowledge "from the bottom up," starting, as stated before, with using purely raw sensorimotor data. It doesn't require different methods at each level.

Predictions as Knowledge

The output of a General value function is a prediction. It estimates the expected value of an aspect of it's future observations, if it were to follow a certain behavior policy at a given state. For example, a general value function may predict that a robot standing next to a wall, may experience a value of 1 from its bump sensor, should it roll forward, and 0 if it did not. How could such a prediction capture knowledge? To answer this question, consider the state the robot is in just prior to rolling forward. Clearly, there is some object (perhaps a wall) just in front of it, should it experience a bump sensation if it were to roll forward. One way of representing the fact it is against a wall is to have a "teacher" tell it that a wall is in front of it. This is somewhat akin to a supervised learning scenario where the agent is presented with thousands of pictures and told that the picture is of a wall. An alternative approach to learning, is that the agent represents the knowledge that it is in view of a wall because it predicts that if it were to roll forward, it would experience a bump. The statement that "I am in front of a wall" specifies a fact that can be verified and reinforced. This verification can be phrased as a prediction - If I were to roll forward, will I experience the bump sensation? If the prediction is high, I can express knowledge regarding the existence of a "wall," whereby "wall" is the human label we have attached to this experience.

This is a very simple example of a general predictive mechanism for knowledge which we suggest may underly all knowledge. The implication is that knowledge is subjective and is fundamentally a set of predictions about the sensorimotor stream. A statement such as "The Edmonton Oilers are playing hockey tonight," fundamentally means something predictive and subjective. To the human, this bit of knowledge about the Oilers implies that if it follows a certain set of actions, a predictable observation will occur. It suggests that the knowledge of the Oilers game isn't gained through someone telling you that the Oilers play tonight, but rather that the agent predicts, perhaps, that if it were to open the NHL app on their smart phone, scroll to the schedule, they would see that the Oilers have a game. The latter example is an incredibly abstract prediction. The demonstration discussed in this paper clearly is not nearly this sophisticated. However, it demonstrates an increasingly more abstract level of predictive based knowledge for which predictions such as those regarding Oilers games, could be built upon.

Background

Reinforcement Learning basics

Within the field of Artificial Intelligence, Reinforcement Learning involves a learner, also responsible to make decisions, called the agent. The entity which the agent interacts with is called the environment. The agent interacts with the environment at discrete time steps. At the beginning of each time-step t = 1, 2, 3, ... , the agent receives an observation - a representation of the environment's state S_t , where S is a finite set of all possible states. Based on S_t, the agent then selects an action denoted A_t, according to a target policy. On the next time-step, the agent receives a scalar reward R_{t+1} and the environment moves to a new state S_{t+1}. This interaction, as depicted below, continues to produce these temporal interactions of states, actions and rewards.

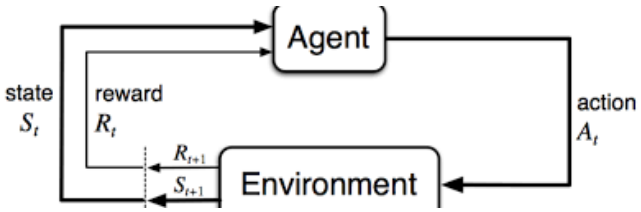

Figure 1: The reinforcement learning architecture.

Value Functions

The goal of many reinforcement learning algorithms, is to estimate the discounted sum of future rewards, called the return. The return G_t at time t is defined by: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...\gamma^n R_{t+n}$

where the discount factor γ discounts future rewards. In other words, future rewards are less valuable than immediate ones. The degree to which this is true depends on the value of γ.

The agent may compute a value function so as to estimate how good a particular state is, with respect to its future reward. A state value function is a function, that estimates expected returns, given a state. This value is equal to the discounted sum of future rewards from a given state, if the agent were to choose actions based on a specific behavior policy.

Function Approximation

Much of the introductory literature within Reinforcement Learning does not require function approximation. The number of states and actions is small enough that a value can be estimated for each state, or state action value. This would be the case of any experiment involving a grid world.

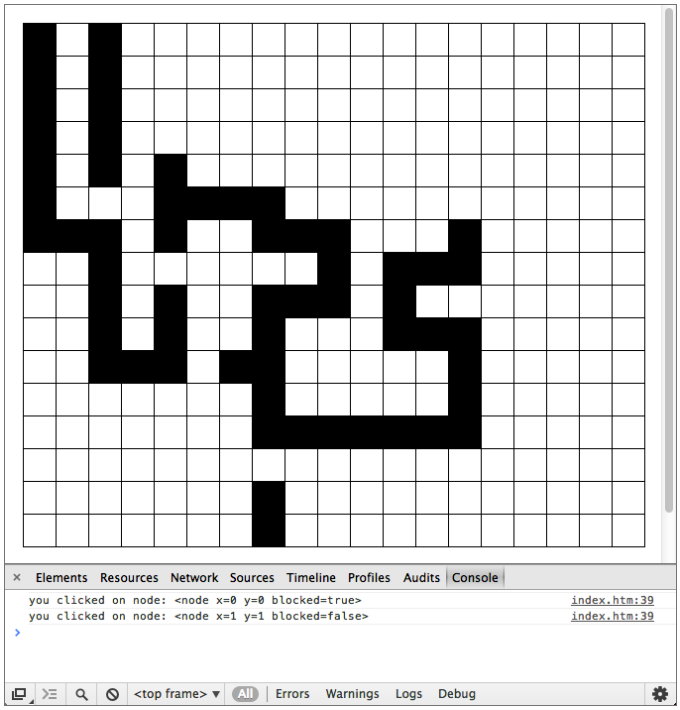

Figure 2: A typical gridworld for which the agent must navigate.

In such a world, the number of states can be modeled exactly by the square identity within the world. In other words, if the environment is modeled by a x*y sized grid, there are exactly x*y possible states. The agent can often estimate the exact values for all of the states. That said, in real applications, the number of possible states is continuous or is simply way to big for this approach. In these environments, the agent may never experience the same state more than once, or not at all. This is the case in the game of Go - where famously, the number of possible positions in a Go board outnumber the number of atoms in the universe.

In such domains we look to generalize the values of states previously observed to other states. In order to make this generalization, we can use function approximation to estimate the state value, rather than store it in a look up table. In this paper, we use linear function approximation, in which each state is represented by a vector, called the feature vector θ. In linear function approximation, the value function estimate is a linear function of a weight vector w, and the θ. The weight vector is adjusted as the agent learns through interaction with the environment.

Tile coding is a form of creating a feature vector, from multi-dimensional continuous state spaces. It may be the most practical feature representation for modern sequential digital computers. -(**?**). When using tile coding, receptive fields of the features are grouped into discrete units to allow for generalization.

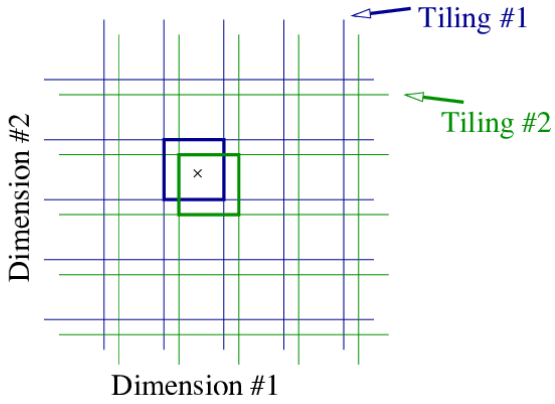

Figure 3: Course coding a value in two dimensional space using tile coding.

General Value Functions

Within this paper, we look to represent predictive knowledge by general value functions. So what exactly are general value functions? Much like value functions, "general" value functions attempt to estimate the future cumulative value of a certain value from the environment. "General" value functions, are an adaptation of conventional value functions in order to a) represent potentially useful predictive knowledge, and b) leverage the strength of value function learning methods. (**?**). General value functions are an adaptation of conventional value functions in an attempt to represent predictive knowledge. These GVFs have the ability to learn from continuous valued inputs, such as those from robots, or in our case, video input. The goal of a GVF is not to maximize a specific reward signal, but rather to predict its value. In addition to the specific goal of representing predictive knowledge, GVFs differ from conventional value functions in two implementation details. First, rather than a reward that is being measured, each GVF is predicting a certain cumulant value z. The agent is not attempting to maximize this cumulant, unlike the reward. The main concern is to predict the future cumulant, rather than to maximize it. Secondly, "termination" is handled much differently with GVFs than conventional value functions. In the latter case, γ is a discount factor that weights the value of future rewards. By changing γ the designer is deciding how much future rewards are worth, in comparison to immediate rewards. This is also the case with a GVF, however, in the case of GVFs this γ is state dependent, thus allowing the timescale of the prediction to be sculpted. For example, a robot attempting to predict the number of steps to a wall, could have a γ of 1.0 for every state other than the one where it's at the wall. When at the wall, $\gamma = 0.0$. In this way, if the cumulant is 1.0, the GVF will predict the exact number of steps to the wall, and then terminates the prediction once there. GVFs defined this way have demonstrated a compact way to express knowledge. Finally, a GVF

contains a behavior policy π. This policy indicates the behavior which the agent is attempting to learn about. Thus, using these three items (cumulant z, gamma γ, and behavior policy π), the agent is able to construct questions about the environment to learn about.

Formally, the GVFs output (which is the value it is attempting to estimate), is defined as a function of a state with three auxiliary function inputs: a target policy specifying behavior, a cumulant function which determines the cumulant value z given, and a gamma function.

$$v(s;\pi,\gamma,z) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\left(\Pi_{j=1}^{k}\gamma(S_{t+j})\right)z(S_{t+k+1})|S_t = s\right]$$
$$= \sum_{a\in\mathcal{A}}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\left[z(s') + \gamma(s')v(s';\pi,\gamma,z)\right].$$

Figure 4: The value of a given state, behavior policy gamma and cumulant is estimated by this equation.

In our paper, we use the GTD lambda algorithm for learning to approximate the value. This algorithm uses a sequence of feature vectors and actions and is given by:

$$\delta_t = Z_{t+1} + \gamma_{t+1}\mathbf{x}_{t+1}^\top\mathbf{w}_t - \mathbf{x}_t^\top\mathbf{w}_t$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left(\delta_t\mathbf{e}_t - \gamma_{t+1}(1-\lambda)(\mathbf{e}_t^\top\mathbf{h}_t)\mathbf{x}_{t+1}\right)$$
$$\mathbf{h}_{t+1} = \mathbf{h}_t + \alpha_h\left(\delta_t\mathbf{e}_t - (\mathbf{h}_t^\top\mathbf{x}_t)\mathbf{x}_t\right)$$
$$\mathbf{e}_t = \rho_t(\gamma_t\lambda\mathbf{e}_{t-1} + \mathbf{x}_t).$$

Figure 5: GTD lambda algorithm.

Related work

In this, my CS 608 term paper, the goal is to demonstrates building abstract knowledge directly from an agents raw, low level sensorimotor stream of data - namely video, and actions. This abstract knowledge is constructed in a layer by layer approach using a single mechanism, the General Value Function (GVF). As such, these layers of GVFs, and GVFs themselves, form representations for higher level abstractions of the raw input. Given this, there are several areas of related work.

There are several papers which discuss learning multiple predictions in parallel as the agent experiences the environment. How they arrange and leverage these predictions varies, but the fact they learn multiple predictions in the Reinforcement Learning environment is common. A Deep Hierarchical Approach to Lifelong Learning in Minecraft (**?**) discusses an agent that is able to learn and transfer knowledge referred to as skills. These skills are high level predictions mapping to actions and can be layered with others. Hord: A Scalable Real Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction (**?**) also outlines an agent architecture that allows the agent to make tens of thousands of these GVF based predictions in parallel, based on a robots sensorimotor data. The paper looks at how each prediction is responsible for answering a single predictive question about the world, and thereby contributing to the systems overall knowledge.

Much of the groundwork for understanding General Value Functions and their algorithms is drawn from Adam White's pHD thesis (**?**). Not only does the thesis propose that knowledge can be modeled in a predictive nature, but it also outlines a number of algorithms, not the least of which is GTD lambda - the one which our agents use to learn. The paper also discusses off policy learning - an idea whereby the agent can learn to make predictions about behavior policies that the agent isn't necessarily following. This off-policy learning is paramount to our experiments where each GVF needs to be able to learn in this fashion.

Experimental Setup

To simplify our experiments, we use a 3D virtual environment whereby our agent has a first person view of the environment. It's actions and observations are similar to that with a robot. The video produced by the game engine is comparable to the visual input produced by an agent operating in the real world. The temporal nature of reinforcement learning ties video input at the core of sensorimotor learning.

Minecraft Environment

Project Malmo is a Minecraft based platform designed for artificial intelligence experiments and research. The environment wraps the Minecraft game engine with a python API to simplify interaction within a reinforcement learning framework.

Figure 6: Video input from minecraft game engine.

Action space

At any time step, our agent can take one of 5 actions. It can turn left 90 degrees, it can turn right 90 degrees, it can move forward, it can move backwards, or it can extend it's hand. The actions are deterministic. Turning left or right always results in a 90 degree change, moving forward or back, unless against a wall, always results in a change of 1 in the x or y coordinate system. Of course, the x and y values are not observable to the agent, but as experimenters, we can observe this.

Observation space

At each time-step, when the agent takes an action, the environment responds by providing an observation. This observation includes the visual data (what the agent sees) as a 320X480 array of pixels. Each pixel contains the RGB valued data. In addition to the visual data, the observation contains the value of its bump sensor. This value is one if the agent "touched" something and zero if it did not.

Feature Representation

Our agent is exposed to the entire visual pixel data at each time-step. However the state space for such an environment is too massive to store the value of each state. To be precise, there are 320X480 pixels, each with RGB values between 0 and 255, therefore 320X480X3X255 possible states. Clearly we require function approximation. Without function approximation, learning value functions for each state would not only be computationally difficult because of the memory requirements for tracking each possible state, but more importantly would not generalize. Using function approximation, our value functions are able to learn about states similar to others.

The feature representation we choose is to represent each state linearly by a feature vector θ. At each time step, our agent selects the same 100 random pixels to represent the state. The RGB values of each pixel are then tile coded (a form of course coding which enables generalization) to create a feature vector for that pixel. These coded pixels are then linearly assembled together to form a final representation θ.

More can be read about tile coding in the Reinforcement Learning book by Sutton and Barto. (**?**)

Environment configuration

We as designers, are able to design the virtual environment our agent interacts with. Our environment is simple. It contains a small room, with 4 walls of different colors. In addition to seeing the walls, the agent is able to see the floor and the sky above the walls.

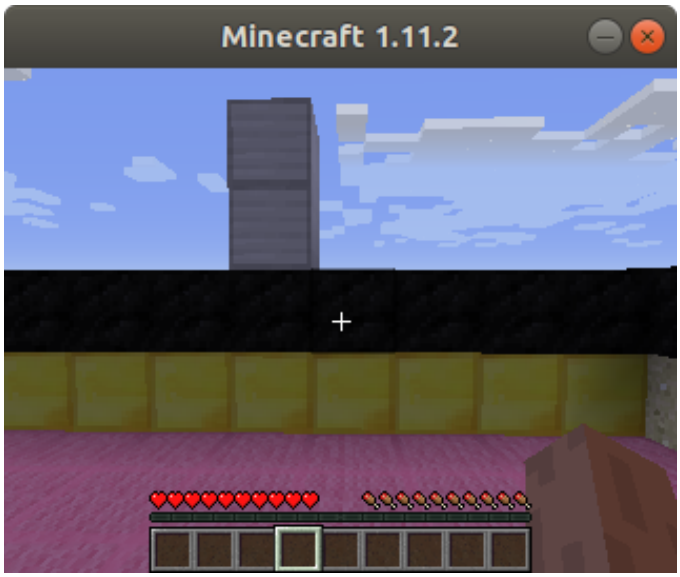

Figure 7: What our agent sees looking at one of the walls in the Minecraft world we created/.

Figure 8: What our agent sees looking at another wall in the Minecraft world.

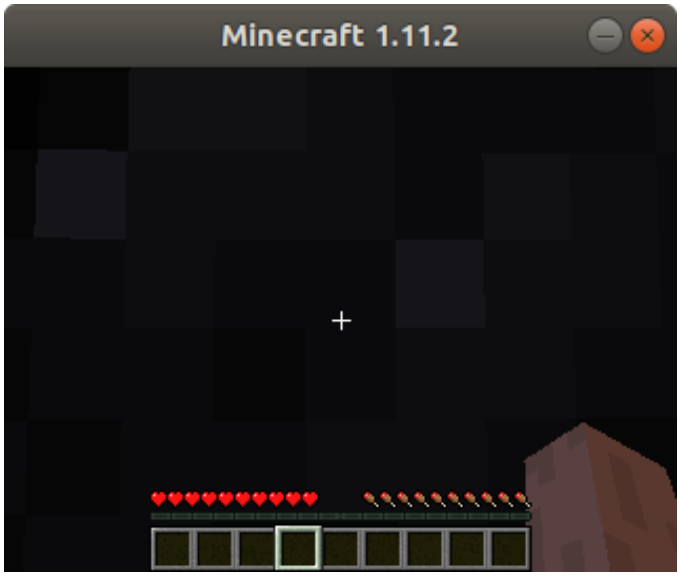

Figure 9: What our agent sees when it is right up against a wall in its environment. In this state, the agent would experience the touch sensation should it take the action of extending its hand.

Behavior policy

Our agent has a challenge to pick its actions in a way as to optimize learning. Creating such a behavior policy π is a massive challenge in and of itself. One option our agent has when choosing actions, would be to choose them randomly (a random policy.) Doing so, would eventually result in the agent being able to learn behavior based predictions about the environment if the agent were able to run indefinitely. However, in reality, a random agent will not learn much within a reasonable amount of time. In contrast, the agent will appear to twitch around a given state, rarely moving far from it.

Our agent adopts a pseudo random policy. It takes random actions which are weighted in such a way that the prior action is biased to repeat again. This allows the agent to continue to roll forward to explore the environment, as well as make several turns in a row, which again encourages exploration.

There are more sophisticated exploration strategies but our pseudo random agent works well enough for our experiments.

Experimental Results

Our goal is to demonstrate the ability to form abstract knowledge about the environment, using only sensorimotor visual data. Mainly the stream of visual data (video). But what exactly do we mean by that? To begin to answer that question, consider predictive "knowledge" which could be formulated by a prediction. Such a prediction might say something like "If I were to extend my finger, I predict I will touch something." Using only the video input, our agent, using a general value function, could learn to answer this question. As such, our agent would be able to learn "knowledge" about its environment - namely knowledge that "I am standing looking at a wall," where "standing" and "wall" just so happen to be the human labels we attach to these observations. Nonetheless, the agent has learned a very primitive amount of knowledge about itself and the environment.

In addition to this primitive knowledge, we look to leverage this basic predictive knowledge to build more abstract knowledge. Specifically, we look for our agent to demonstrate an ability to learn, using only visual data, to answer these predictive questions:

- If I extend my hand, would I then be able to touch a wall? (ie. is there a wall in front of me?) T
- If I turn left, would I then be able to touch a wall? (ie. is there a wall to my left?) TL
- If I turn right, would I then be able to touch a wall? (ie. is there a wall to my right?) TR
- If I turned 180 degrees, would I then be able to touch a wall? (ie. is there a wall behind me?) TB
- Is there a combination of turns I could take that would put me in a position where I could touch a wall? (ie. Am I adjacent to a wall?) TA

Each of these questions leverages the predictions made before it. For example, to predict whether an agent could turn left and then be able to touch a wall (TL), requires the agent to have accurately learned to predict whether extending its hand would result in touching a wall (T). Learning whether turning 180 degrees would result in a position where a wall is touchable (TB), requires the agent to have learned whether turning right would result in being in a touch position

(TR). As such, we claim that more abstract knowledge of the environment is learned.

Through experimentation, our agent was able to make reasonable predictions for all of the above 4 questions. The rest of this write up looks to explain and demonstrate how these predictions were made.

Visualizing the learning

As the agent learns, we would like to see its progress. In other words, how accurate its predictions are. Therefore, at each time step, I wanted to be able to see:

- The ground truth visualization of the game.
- A representation of what our agent observes of the world (given that our agent is only privy to a 100 pixel subset of the 320X480 pixels at each frame.
- The current time step.
- The prediction value for each prediction Touch (T), Touch Right (TR), Touch Left (TL), Touch Behind (TB), Touch Adjacent (TA)
- The truth value for each prediction - so that we could see whether our prediction was accurate or not.

Below is a screenshot which shows how we chose to visualize this data.

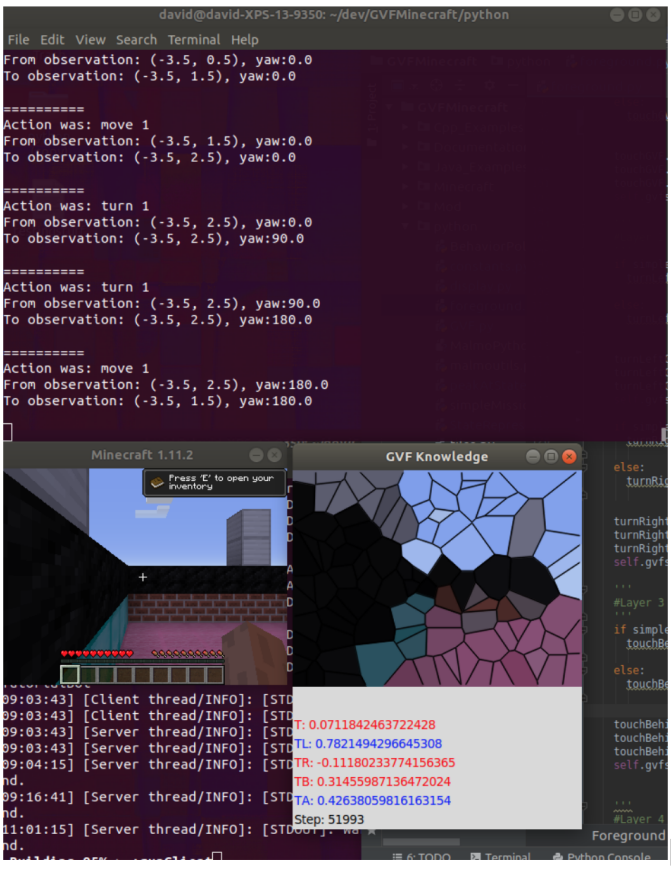

Figure 10: Screenshot of the agent learning knowledge about its environment

Explaining the results

There is a lot of information displayed in the above screenshot of the agent interacting with the world. First off, there is a log of which actions (move 1, move -1, turn 1, etc) were chosen at each time-step and what x, y coordinates and orientation this resulted in. Much like a human in the real world unaware of its geo coordinates and orientation, our agent is not privy to the x, y, and coordinate information. It must rely purely on its visual input. However, we, as experimenters, can use this state information to help debugging the system, and thus I have made it available to visualize.

The actual game video is displayed as is, such that we can see the ground truth about where the agent is within its environment. To visualize what the agent is seeing, you can see a Voronoi representation of the 100 pixels that agent used . This is updated at each time step the agent takes an observation. At each time step and respective state, you will notice the prediction values displayed. These prediction values represent what value the agent estimates with respect to the question the respective GVF is asking for the current state.

- T (Touch). If I extend my hand, would I touch a wall? (ie. Is there a wall in front of me?)
- TL (Touch Left) If I turn left, would I then be able to touch a wall? (ie. Is there a wall to my left?)
- TR (Touch Right) If I turn right, would I then be able to touch a wall? (ie. Is there a wall to my right?)
- TB (Touch Back) If I turned 180 degrees, would I then be able to touch a wall? (ie. Is there a wall behind me?)
- TA (Touch Adjacent)Is there a combination of turns I could take that would put me in a position where I could touch a wall? (ie. Am I adjacent to a wall?)

It is also important to display the ground truth for all of these predictions. Although our agent only has the visual input to make these predictions, we, as designers "know better." We're exposed to the state of the entire world, thus can determine whether the agent is indeed in front of, beside, or behind a wall. We want to display this information to determine whether our predictions are accurate or not. Our solution to do this was to display the predictions in red when the ground truth prediction should be zero (ie. it is false). If the prediction should be true, it is displayed in blue. For example, in the above screenshot, you can see that the agent has made 5 predictions.

T It is predicting that there is a 7 percent chance that there is a wall in front of it. We visualize that there is indeed no wall in front by displaying that the prediction is in red. This can also be confirmed just by looking at the image data the agent sees.

TL is predicting that there is a 78 percent chance that there is a wall on its left. Again, it has learned this about the environment through interaction. It has taken a number of actions turning left and then used the Touch prediction at that state to learn which states

have walls on the left. In the example, there is indeed a wall on the left so the prediction is displayed in blue.

TR It is predicting that there is a -10 percent chance that there is a wall on its right. In theory values should be bound between 0 and 100 percent, however, learning can overshoot these values slightly, as the prediction can never perfectly converge t0 0 or 100 percent, unless the learning rate goes to zero. Instead, the prediction can go above or below this value slightly. You can see that the ground truth is that there is no wall to the right so the prediction is relatively accurate.

TB It is predicted that there is a 31 percent chance that there is a wall behind the agent. There is no wall behind it and the prediction would become more accurate as time goes on. However, this prediction will always struggle to become highly accurate. Given our environment and how we represent it via linear function approximation, it will be tough to differentiate the state seen in the image below, and one step "behind it" where the agent would actually be against a wall. The pixels in both states are very similar, so the predictions will be challenged to differentiate between the two. Given our predictions generalize across states, because there is much overlap between the states, our predictions will never be 100 percent accurate. Nonetheless, we are able to make reasonably accurate predictions even for this prediction.

TA It is predicting that there is a 42 percent chance that there is a wall to touch if it were to turn the correct number of times. The ground truth for this prediction is actually 100 percent (as you can visually see because the prediction is in blue and you can clearly see a wall on its left). So our agent is predicting it incorrectly. This is simply because our agent, at 51993 time steps, has not had enough time to accurately make this prediction. The reason it takes a lot longer to learn this prediction accurately, is that it relies on accurate predictions preceding it. In other words, it requires the TB prediction to be made accurately. The TB prediction requires the TL and TR predictions to be made accurately. And the TR and TL predictions require the T prediction to be made accurately. Therefore, more abstract predictions such as the TA prediction will inherently take longer to form. At the time the screenshot was taken, our agent simply hadn't had enough time to learn.

Video of agent learning

A video of the agent learning to make these values is available at https://vimeo.com/303392126 Within the video you can observe the agent as it navigates the environment, along with its predictions at each time step. As you'll see, the predictions are not perfect, but for the most part making high predictions (values above 80 percent when it is actually near a wall) and values around 10 percent when it is not.

General Value function network

We have to set up our network to form increasing levels of abstraction. Following describes the hierarchy of GVFs.

Level 1: Touch (T)

Cumulant: Touch sensor value (1 or 0) Gamma: 0 Behavior Policy: Action always is to touch

Level 2a: Touch right (TR)

Cumulant: T prediction for new state Gamma: 0 Behavior Policy: Action always is to turn right

Level 2b: Touch right (TL)

Cumulant: T prediction for new state Gamma: 0 Behavior Policy: Action always is to turn left

Level 3: Touch behind(TB)

Cumulant: TR prediction for new state Gamma: 0 Behavior Policy: Action always is to turn right

Level 4: Touch adjacent (TA)

Cumulant: Max(predictions(T,TR,TL,TR,TB)) Gamma: 0 Behavior Policy: Action always is to turn right

Conclusions

A claim of this paper is that a great problem in AI is that of connecting representations to what it represents. So naturally, it is reasonable to ask, how does a predictive representation deal with this problem. Our suggestion is that knowledge exists only within the context of interaction within an environment. This knowledge can be verified in the agents sensorimotor stream. The agents knowledge is constrained to what it experiences, building up an awareness of how things respond to certain actions.

Our experiments, demonstrate how an agent may not only represent knowledge by a layered predictive approach, but it demonstrates the ability for such an agent to learn to make accurate predictions. The level of abstraction which is demonstrated is limited, but it is shown that the levels of abstraction can be increased in a similar fashion as the first 4 layers in our network were, until the agent is representing knowledge such as the existence of walls, corridors, and doors.

Code

All code is available at https://github.com/dquail/GVFKnowledge

Future work

There are seceral immediate extensions to this paper I would like to make.

- Learning rates should be displayed for each prediction. It would be useful to plot the average accuracy of each of the T, TL, TR, TB, TB predictions as a function of time. This would highlight the dependency on previous predictions to make more abstract ones. To create such a chart, multiple learning runs

would need to be made to get the average error rates at each time step. Given the current state of the game engine producing only 5 time steps per second, we are computationally bound to create such averages. However, this would be possible and should be done with more computational power.

- Naturally, I would like to create a deeper network of General Value functions. There are several other layers possible. We were only able to learn the first 4 layers. Learning further layers becomes increasingly more difficult because of the challenges to explore the state space. A behavior policy beyond pseudo random will likely be needed for such experiments. Furthermore, the way we're representing the state space (by linearly encoding pixel values) may prove to be limiting as we attempt to build increasing levels of abstraction.
- It would be an additional challenge to implement the experiments on a physical robot.