CS234 Project Final Report: Approaches to Hierarchical Reinforcement Learning

Blake Wulfe [*1] **David Hershey** [*1]

Abstract

Vezhnevets et al. recently introduced the Feudal Network, a Hierarchical Reinforcement Learning architecture capable of learning options without the manual specification of subtasks. The approach represents sub-goals as changes to a learned state representation, and in doing so converts the traditionally challenging task of sub-goal discovery into a representation learning problem that can be solved effectively using neural networks. In this paper, we implement the Feudal Network and test its performance against competitive baseline agents. We present the details of our implementation with emphasis on describing certain design decisions in the implementation of the model. We then explore some of the challenging aspects of training the Feudal network, certain subcomponents of the model, and finally discuss conclusions as to the general applicability of the approach.

1. Introduction

Deep reinforcement learning has been particularly successful in domains with large state spaces (Mnih et al., 2013; 2016; Silver et al., 2016). These and other RL methods still struggle with both sparsely-rewarded environments and environments requiring complex, temporally extended action sequences. Many real world tasks to which we would like to apply RL pose these challenges, and methods that allow RL algorithms to deal with them effectively are desirable. One method for partially overcoming these issues is Hierarchical Reinforcement Learning (HRL). HRL methods involve taking temporally extended actions, often in an attempt to achieve sub-goals.

This method conveys a number of advantages, two of which we focus on in this paper. First, temporally extended actions provide more structured exploration of the state space, thereby improving exploration over dithering strategies, and ameliorating the problem of sparse rewards (Lakshminarayanan et al., 2016; Sharma et al., 2017). Second, HRL approaches allow an agent to select actions at a greater level of temporal abstraction, thereby making extended, complex tasks easier simply by requiring fewer appropriate consecutive actions to solve them (Dietterich). This second advantage can also be viewed as allowing information to propagate through the state space as perceived by the agent more rapidly (Sutton et al., 1999).

There are a number of approaches to HRL when these temporally extended actions are provided a priori (Dietterich; Parr & Russell). We focus on the case in which the agent must autonomously learn options. This problem has been researched extensively, both traditionally (Konidaris et al., 2010; Şimşek & Barto, 2004; Bakker & Schmidhuber) and more recently in the deep RL setting (Heess et al., 2016; Bacon et al., 2016; Vezhnevets et al., 2016; Arulkumaran et al., 2016). We implement a recently-proposed method, Feudal Networks (Vezhnevets et al., 2017), which builds on a traditional method (Dayan & Hinton, 1993).

Feudal Networks are notable for how they address the problem of autonomously learning options. Options are typically learned by extracting sub-goals based on characteristics of the environment or from an agents previous trajectories (Konidaris & Barto, 2009), and then using intrinsic rewards to train, using a flat RL algorithm, an option to achieve a sub-goal. Extracting sub-goals in a fully general manner is challenging, particularly in environments with large state spaces, such as those based on visual input. Feudal Networks formulate sub-goals as changes to a learned state-representation, which effectively converts the sub-goal discovery problem into the problem of learning a state representation (Bengio et al., 2013), which can often be accomplished quite well using neural networks. This approach to sub-goal generation is highly general, and Vezhnevets et al. show it to be effective in complex environments.

In this paper, we implement Feudal Networks with three goals in mind. First, we hope to better understand the operation of Feudal Networks, their strengths and weaknesses,

[1]Stanford University, Stanford, CA. Correspondence to: Blake Wulfe <wulfebw@stanford.edu>, David Hershey <dshersh@stanford.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, 2017. JMLR: W&CP.

and possible extensions. Second, we hope to validate the experimental results of Vezhnevets et al. And third, we wish to provide an open source implementation of Feudal Networks against which other researchers may compare their algorithms.

2. Related Work

2.1. Hierarchical Reinforcement Learning

As mentioned briefly in the introduction, a number of RL approaches exist for the case in which either subtasks or options are prespecified (Parr & Russell). For example, MAXQ value function decomposition describes how to represent a value function over a task hierarchy (Dietterich).

Methods that autonomously learn temporally abstract actions can be roughly separated into those that rely on (explicit or implicit) sub-goals, and those that infer options from either agent or expert trajectories (Arulkumaran et al., 2016; Konidaris et al., 2010; Daniel et al., 2016). Heess et al. present an example of the former approach that uses manually defined subtasks, but provides motivation for Feudal Networks (Heess et al., 2016). The researchers pre-train a worker network to achieve absolute state configurations based upon input from a manager network, transfer the worker to different managers and end tasks, and demonstrate that this approach affords the benefits of temporal abstraction. Feudal Networks can be viewed as an extension to this work in which the subtask is not provided. In this case, the manager and worker networks are still present; however, the subtasks are expressed as *relative* changes to *learned* state representations as opposed to absolute changes in the observation space.

Other recent approaches to autonomous option learning include the Option-Critic architecture (Bacon et al., 2016), to which Vezhnevets et al. compare their approach. Furthermore, a recent method uses generative adversarial networks to generate sub-goals (Held et al., 2017).

2.2. Recurrent Neural Network Models for Temporal Abstraction

A key contribution of the Feudal Network research is a new type of recurrent neural network (RNN) called the Dilated LSTM (dLSTM). This model deals particularly well with long-term dependencies, and is related to a number of similar RNNs, for example Clockwork RNNs (Koutnik et al., 2014), hierarchical RNNs (El Hihi & Bengio, 1996), Multiscale RNNs (Chung et al., 2016), and Gated-feedback RNNs (Chung et al., 2015).

3. Background

3.1. Asynchronous Advantage Actor Critic

Asynchronous Advantage Actor Critic (A3C) is a popular RL method that deals with issues of learning stability by making asynchronous updates to the network using gradients produced by a set of parallel, cpu-based learners (Mnih et al., 2016). This approach carries the additional benefit of improved exploration simply through large amounts of environmental interaction. A3C performs n-step return, policy gradient updates with bootstrapping and the baseline provided by a learned value function. The gradient of the A3C objective takes the form

$$(R_t - V(s_t|\theta))\nabla_\theta \log \pi(a_t|s_t, \theta) + \beta\nabla_\theta H(\pi(a_t|s_t, \theta))$$

where $H(.)$ is the entropy of the policy, which is incorporated into the objective so as to prevent the policy from collapsing to a single action. Note that we use A3C to train both our baseline model, a LSTM-based policy, as well as the Feudal Network.

3.2. Feudal Networks

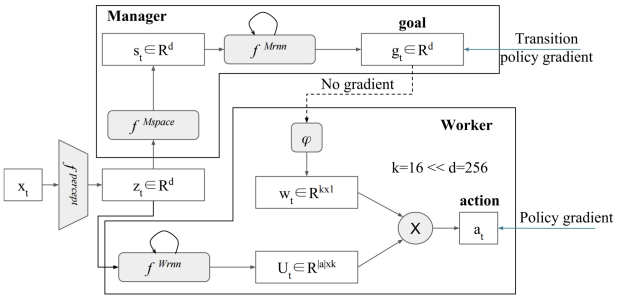


Figure 1. The Feudal Network Architecture (Vezhnevets et al., 2017)

The Feudal network is a HRL architecture that addresses the problem of generating sub-goals by representing sub-goals as changes in a learned state representation. The Feudal network has two major sub-components, a manager and a worker network.

The manager network is tasked with learning the internal representation of the state space, $s \in \mathbb{R}^d$, and processing that state into a goal vector $g \in \mathbb{R}^d$. The goal vector has semantic meaning; it is a desired change in s, c states in the future, that the manager predicts will be valuable. This g is then passed to the worker to guide its policy. The manager is rewarded if the worker both achieves this change in states g, and is rewarded in the environment. Formally, this update rule can be written as:

$$\nabla g_t = A_t^m \nabla_\theta d_{cos}(s_{t+c} - s_t, g_t(\theta)) \quad (1)$$

Where A^m is the manager advantage, and d_{cos} is the direction cosine.

The worker network learns to enact a policy that both achieves high reward in the environment and follows the goals output by the manager. The worker is structurally similar to a LSTM-based policy such as that of our baseline. The major structural difference is that the worker is directly forced to use the goals output by the manager when computing its policy. This is accomplished by having the worker output an action embedding U, each row of which is multiplied by a goal vector w, thereby producing scores for each action. This approach, which is limited to discrete action spaces, strictly forces the worker to not ignore the manager's output. The worker is then updated via two rewards, one intrinsic and one extrinsic. The intrinsic reward is given when the worker successfully enacts the goal given by the manager. Formally:

$$r_t^I = 1/c \sum_{i=1}^{t} d_{cos}(s_t - s_{t-i}, g_{t-i}) \tag{2}$$

The extrinsic reward R is simply the reward given by the environment. This yields the update rule:

$$\nabla \pi_t = A_t^D \nabla_\theta log\pi(a_t|x_t;\theta) \tag{3}$$

Where $A^D = (R + \alpha R^I - V^D(x_t;\theta))$, a_t is the action at time t, and V^D is a value estimate of both rewards.

3.2.1. Dilated LSTM

Vezhnevets et al. propose the Dilated LSTM, which is inspired by dilated convolutions (Daniel & Neumann). This network functions by maintaining a number of internal LSTM cell states, defined by a radius r. At any given timestep, only one of the cell states is updated, and the output of the LSTM is the pooled output of the last c states that have been updated. As such, the network retains memories for at least r time steps, while still updating its output at each timestep. The authors claim that this structure is vital to the functionality of the manager network.

4. Approach

4.1. A3C

We used an open-source implementation of A3C for the LSTM policy [1], and adapted this implementation to work with the Feudal Network as detailed in the next section.

[1] https://github.com/openai/universe-starter-agent

4.2. Feudal Networks

4.2.1. Implementation

We implemented the Feudal network as described in (Vezhnevets et al., 2017), and source code of the implementation is available online at `https://github.com/dmakian/feudal_networks`. In this section we outline implementation details and design decisions.

Perception Network The perception network is described in the initial paper as a two layer CNN with 8x8 filters in the first layer and 4x4 filters in the second layer. To make the network more directly transferable to tasks with different observation spaces, we have modified the perception network to use a configurable number of layers, each with 3x3 filters. We do not expect that this will significantly impact network performance.

Manager Network Within the manager network, there are a large number of design decisions, and certain details are not included in the original paper. For example, the exact implementation details of the Dilated LSTM, random goal sampling, Feudal batch-padding, and other aspects in the original paper are not entirely clear, and we provide details of our implementation below.

Vezhnevets et al. state that at each timestep t with a small probability ϵ the output goal g_t is sampled randomly from a univariate Gaussian. The value of the parameter ϵ is not specified in the paper, and we experiment with different values below.

When normalizing the value of the goal as output by the manager, we found that backpropagating gradients through the norm operation frequently caused the network to fail to converge on simple debugging environments. As such, the normalization was treated as a constant operation, and no gradients were passed through the calculation of the norm. For both the worker and the manager, the structure of the value function networks were not reported in the original paper. For both, we used a single hidden layer neural network, where the input was the output of the respective worker and manager LSTMs.

Worker Network The worker network implementation is fairly straightforward. The only change we made to this network was the addition of two hidden layers after the LSTM to further separate the policy output from the value function. Making the value function an entirely separate network may provide some advantages, though we do not consider this option.

Loss When calculating the direction cosine component of the Manager loss, there are degenerate cases (such as zero norms) that cause infinite gradients. We added small

Parameter	Description
α	Intrinsic reward weight
β	Entropy loss weight
ϵ	Random goal probability
γ_m	Manager discount factor
γ_w	Worker discount factor
Learning Rate	–

Table 1. Feudal Hyper-Parameters

constant terms to avoid this issue.

Batch Processing Since the loss function for the manager is a function of future performance, updates for the Feudal network require knowledge of future and past rollouts. As such, batches of data need to be pre-processed before they can be fed to the network at training time. The implementation of these batches is not specified in the original paper, and we implemented them by maintaining a moving window around each n-step batch with c elements preceding and following. The first and last c elements of each trajectory must hallucinate values where none exist. We experimented with zero and end-value padding, concluding that end-value padding worked best.

Description of Hyper Parameters The hyperparameters involved in tuning the network are presented in Table 1.

4.2.2. Dilated LSTM

We have also completed an implementation of the proposed dLSTM. The implementation details of the dLSTM are not particularly clear in the original paper, so we will describe our implementation here:

The network maintains r cells $(c^{0:r}, h^{0:r})$, each the full size of the LSTM state (in the Feudal paper, defined as 256). At each time t, the following update to the internal state is performed:

$$c_t^{t\%r}, h_t^{t\%r} = LSTM(s_t, c_{t-1}^{t\%r}, h_{t-1}^{t\%r}; \theta) \tag{4}$$

The output of the network is then the mean of the hidden states:

$$y_t = 1/r \sum_{i=0}^{r} h_t^i \tag{5}$$

In our experiments, we evaluate this network by comparing it against the LSTM baseline agent.

5. Experimental Results

5.1. Maze Experiments

In order to test and evaluate the Feudal Network, we implemented a simple maze environment. This environment varies in two parameters, the side length of each room in the maze, and the number of rooms per side. Varying these parameters helps in evaluating how agent performance changes with the difficulty of the environment. For example, we expect that the Feudal Network will outperform the baseline in a large maze with many rooms because of its ability to account for long-term dependencies and use structured exploration. The maze environments we consider have a stationary goal in one corner of the maze, with the agent starting in random positions [2]. The agent only receives a reward of one on exiting the maze, and takes deterministic actions in one of four directions. Rooms are connected by single-cell doorways.

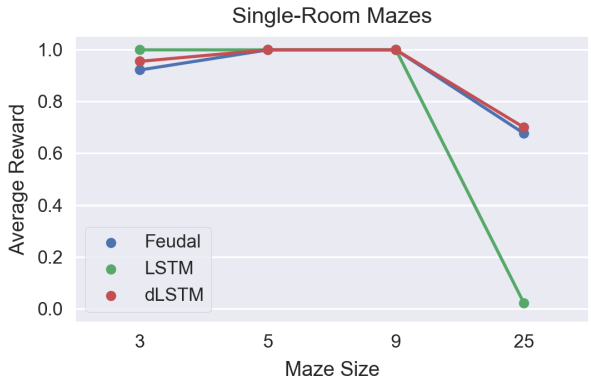


Figure 2. Performance with increasing size of a single-room maze. Size listed is that of one side of the maze.

Figures 2 and 3 present the results of our experiments across eight different maze environments. Results are averaged across three training runs for each algorithm, where each run is comprised of 100,000 training steps. Feudal and LSTM have similar performance on the single-room mazes, where sub-policies and long-horizon dependencies are expected to be less of a factor.

On the multi-room mazes, Feudal out-performs the LSTM. The Feudal network is able to more consistently converge to a good solution for these mazes, especially as the mazes grow larger. This result is in line with the expectation - the Feudal network can potentially leverage sub-policies (such as navigating a room) in order to more efficiently explore the state space.

[2] We note upon review that placing the goal in one corner of the maze does not provide the best evaluation environment because agents can potentially achieve high performance simply by appropriately setting the bias units on the final output layer.

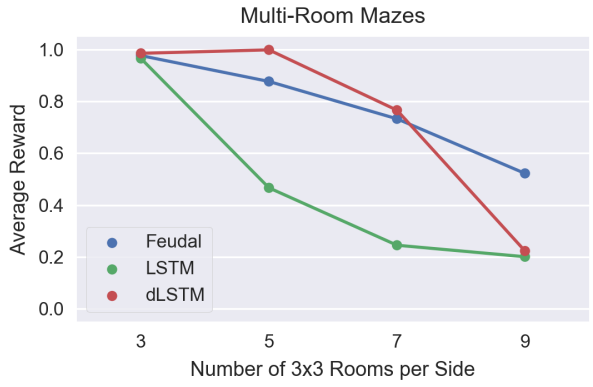


Figure 3. Performance with increasing number of same-sized rooms. Size listed is number of rooms per side of the maze.

In order to understand how the Feudal network is learning, we can visualize the direction cosine between the goals set by the manager, and the actual state transitions that occurred under the worker's policy. If the worker is exactly following the manager's output, then this value would be 1. Figure 4 shows the value of this function during learning on a 5x5 room maze.

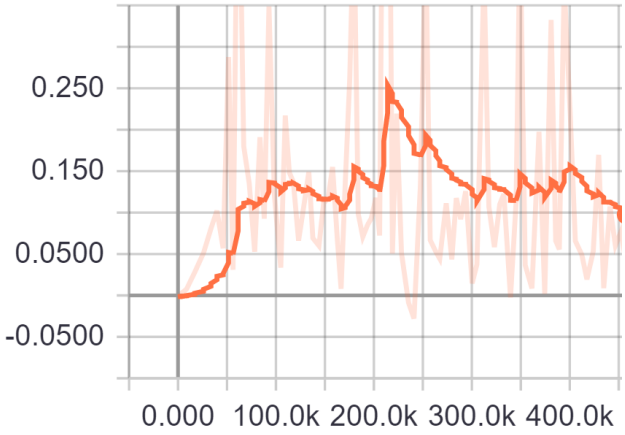


Figure 4. Plot of Direction Cosine while Learning 5x5 Room Maze

The worker learns to enact the changes in the state space requested by the manager. This means that the network is indeed learning to optimize its intrinsic reward, which suggests that the Feudal network is taking advantage of the structure of the multi-room mazes.

5.2. Atari 2600 Games

Atari is a common baseline for Deep RL (Mnih et al., 2013). Atari games feature a large state space (potentially all permutations of the pixels of the screen), and a discrete action space. Further, the accessibility of many games allows one to test the ability of an agent to generalize to highly varied tasks using the same hyperparameters. We test our agents on two Atari games, Pong, which is a relatively simple game, and Amidar, a more complex game where we would expect the Feudal Network to outperform the baseline.

5.2.1. A3C Baseline on Atari

As a baseline, we train an A3C agent based upon an LSTM model on the Atari game Pong. The training curve can be seen in Figure 5 The A3C agent is able to learn Pong (score

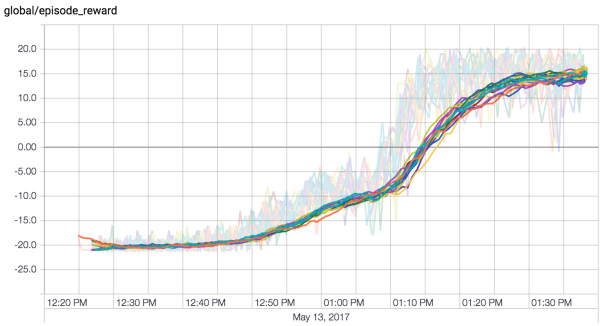


Figure 5. Learning Curve of Pong on A3C

of +21) within 10 million frames. In general, many deep RL agents can learn Pong to this level of success.

5.2.2. Feudal Network on Atari

We then trained the Feudal network on Pong. The training curve can be seen in figure 6. We have not been able to find

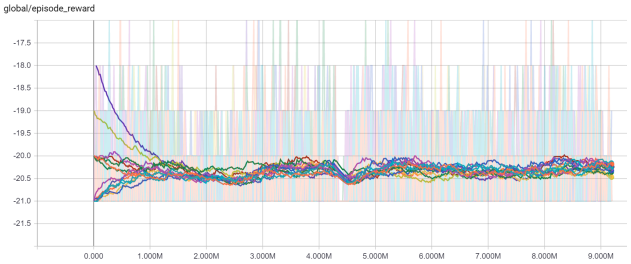


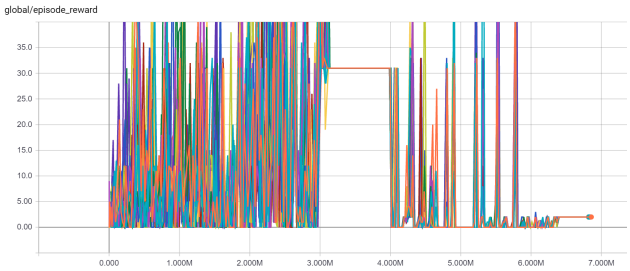
Figure 6. Learning Curve of Pong on Feudal

tuning parameters such that Feudal network converges to a good solution on Pong.

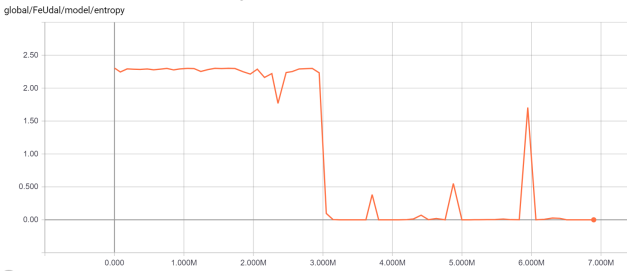
5.3. Failure Analysis

5.3.1. Entropy Analysis

One of the symptoms that occurs when the Feudal network fails to converge to a good solution on Atari is entropy collapse, where the algorithm always takes the same action.



(a) Learning Curve of Amidar on A3C



(b) Plot of Entropy during Amidar Training

Figure 7. Entropy Collapse in Amidar Training

An example of this occurring on the game Amidar is shown in figure 7. This entropy collapse can occur for a range of reasons. If the hyperparameter β is too low, there is no reward for keeping a high-entropy distribution, which can cause this type of collapse.

5.3.2. Dilated LSTM

A novel contribution in the original paper is the dLSTM used by the manager network. As shown in the original paper, the dLSTM can be used on top of a CNN as its own RL agent. Some of the properties of the dLSTM could prove useful on tasks that need long term memory. To further understand the Feudal network, we have implemented this dLSTM based agent. We include the dLSTMs performance on the simple maze environments in figures 2 and 3. The dLSTM outperforms the LSTM on the multi-room maze, an environment that can benefit from accounting for longer horizons.

We also compare LSTM and dLSTM in a maze environment in which the goal changes location in each episode and is part of the environment. Figure 8 shows the average reward during training of the two models, with $r = 4$ for the dLSTM. In this experiment, the LSTM policy outperformed the dLSTM (we stopped LSTM execution once it converged). This may be because the maze considered was relatively small (3 rooms per side of side length 3), or may be due to the fact that the dLSTM simply learns more slowly due to making fewer weight updates per step. While the dLSTM does eventually reach the same performance, it is also susceptible to entropy collapse, indicated by the dramatic decrease in average reward late in training.

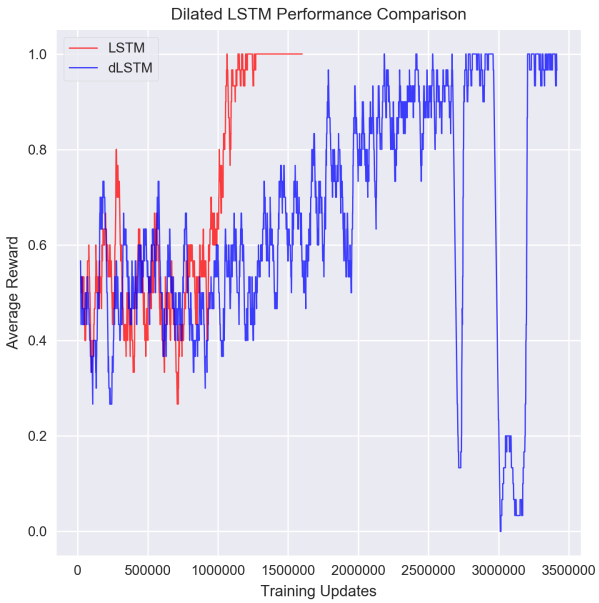


Figure 8. Comparison of LSTM and Dilated LSTM in a maze with moving goal location.

We also tested the dLSTM on Atari Pong, across multiple values of r. This should allow us to see how the dLSTM compares to the LSTM on a more complex environment. For $r = 4$, the dLSTM begins to learn, but much more slowly than the LSTM. We cut off learning after ten million frames, at which point it had learned at roughly half the rate of the LSTM.

With $r = 10$, the dLSTM did not learn within five million frames. It is possible that given much more time it would have converged, but it clearly was much slower than the LSTM.

The dLSTM converging more slowly makes sense, as updates are shared across r steps. For some environments the dLSTM may be more successful, but on environments where long term memory is not necessary the LSTM is clearly a better option.

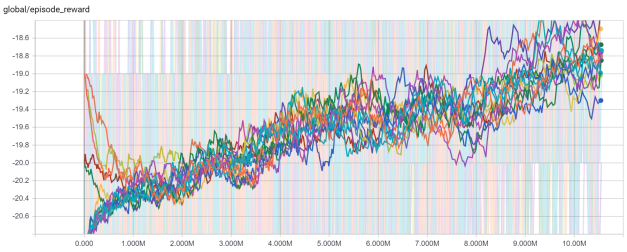


Figure 9. Learning Curve for Pong - dLSTM r=4

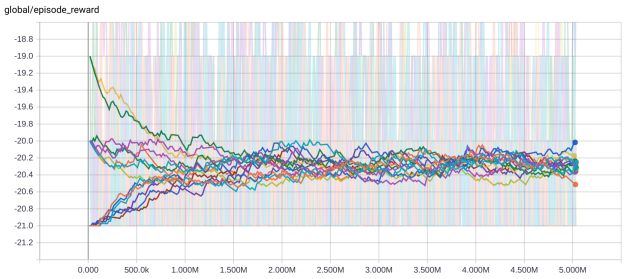


Figure 10. Learning Curve for Pong - dLSTM r=10

5.3.3. Random Goal Probability

We conducted a series of tests on Feudal networks in which we varied the probability ϵ with which a random goal is sampled in the manager. In these tests, we trained the Feudal network on the seven room maze across a range of initial values of ϵ, and found that the network was not sensitive to this parameter.

6. Conclusion

The Feudal Network uses great intuition to potentially scale to more challenging environments. The observation that sub-goals can be posed as changes in a learned internal state representation is a highly compelling methodology. In this paper, we showed that in some complex environments, this structure can allow for more consistent solutions than a competitive baseline agent.

Specifically, by analyzing the internals of the Feudal network, we determined that the worker can learn to successfully enact the goals produced by the manager. This shows the potential of the model, as it has clearly successfully produced meaningful sub-goals.

While investigating the network, we found it sensitive to small changes in hyperparameters. Problems like entropy collapse plague performance on complicated tasks like Atari, and the large number of parameters driving the network make it difficult to build and tune. While its possible that our issues stem from a misinterpretation of the network as proposed in the original paper, it still seems that the network is less robust than presented.

Nevertheless, the Feudal network is an insightful step forward in Deep HRL, and moving forward, research that leverages the motivating insight of the model seems to be a promising avenue of future work.

References

Arulkumaran, Kai, Dilokthanakul, Nat, Shanahan, Murray, and Bharath, Anil Anthony. Classifying options for deep reinforcement learning. *arXiv preprint arXiv:1604.08153*, 2016.

Bacon, Pierre-Luc, Harb, Jean, and Precup, Doina. The option-critic architecture. *arXiv preprint arXiv:1609.05140*, 2016.

Bakker, Bram and Schmidhuber, Jürgen. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization.

Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Chung, Junyoung, Gülçehre, Caglar, Cho, Kyunghyun, and Bengio, Yoshua. Gated feedback recurrent neural networks. In *ICML*, pp. 2067–2075, 2015.

Chung, Junyoung, Ahn, Sungjin, and Bengio, Yoshua. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Daniel, Christian and Neumann, Gerhard. Hierarchical relative entropy policy search.

Daniel, Christian, Neumann, Gerhard, Kroemer, Oliver, and Peters, Jan. Hierarchical relative entropy policy search. *The Journal of Machine Learning Research*, 17 (1):3190–3239, 2016.

Dayan, Peter and Hinton, Geoffrey E. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–271. Morgan Kaufmann Publishers, 1993.

Dietterich, Thomas G. Hierarchical reinforcement learning with the maxq value function decomposition.

El Hihi, Salah and Bengio, Yoshua. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pp. 493–499, 1996.

Heess, Nicolas, Wayne, Greg, Tassa, Yuval, Lillicrap, Timothy, Riedmiller, Martin, and Silver, David. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Held, David, Geng, Xinyang, Florensa, Carlos, and Abbeel, Pieter. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*, 2017.

Konidaris, George and Barto, Andrew G. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, pp. 1015–1023, 2009.

Konidaris, George, Kuindersma, Scott, Grupen, Roderic, and Barto, Andrew G. Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *Advances in neural information processing systems*, pp. 1162–1170, 2010.

Koutnik, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Juergen. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Lakshminarayanan, Aravind S, Krishnamurthy, Ramnandan, Kumar, Peeyush, and Ravindran, Balaraman. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*, 2016.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Parr, Ronald and Russell, Stuart. Reinforcement learning with hierarchies of machines.

Sharma, Sahil, Lakshminarayanan, Aravind S, and Ravindran, Balaraman. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Şimşek, Özgür and Barto, Andrew G. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 95. ACM, 2004.

Sutton, Richard S, Precup, Doina, and Singh, Satinder. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Vezhnevets, Alexander, Mnih, Volodymyr, Osindero, Simon, Graves, Alex, Vinyals, Oriol, Agapiou, John, et al. Strategic attentive writer for learning macro-actions. In *Advances in Neural Information Processing Systems*, pp. 3486–3494, 2016.

Vezhnevets, Alexander Sasha, Osindero, Simon, Schaul, Tom, Heess, Nicolas, Jaderberg, Max, Silver, David, and Kavukcuoglu, Koray. Feudal networks for hierarchical reinforcement learning. *CoRR*, abs/1703.01161, 2017. URL http://arxiv.org/abs/1703.01161.

7. Appendix I: OpenAI Universe

Much of our original focus was on developing an agent capable of performing the OpenAI Universe Mini World of Bit tasks. We ultimately shifted to evaluating the Feudal Network, and so have included this portion of the research as an appendix.

7.1. OpenAI Universe Tasks

OpenAI Universe facilitates the training of RL agents on domains where actions are input with the mouse and keyboard. Specifically, we test on the OpenAI Mini World Of Bits benchmark, which provides a series of test suites to attempt to learn to manipulate the mouse and keyboard. These tests range from clicking buttons on forms to playing tic tac toe, and they are likely to benefit substantially from hierarchical approaches.

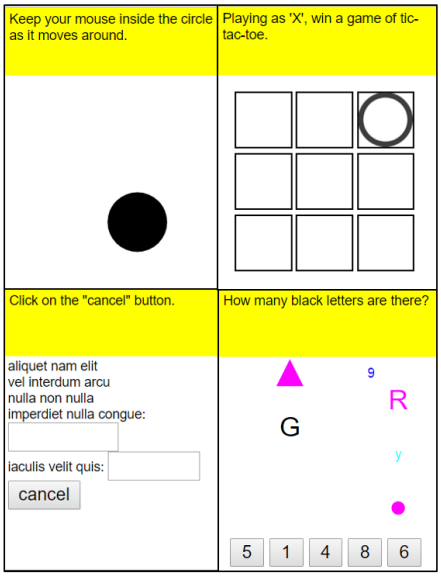


Figure 11. Examples of OpenAI Mini World of Bits tasks

7.2. Mouse Control

We have explored a subset of the tasks included in the OpenAI Universe Mini World of Bits [3]. The upper left image in Figure 11 depicts the ChaseCircle task on which we have focused. We adapted the Asynchronous Advantage Actor-Critc (Mnih et al., 2016) starter agent code to use a mouse, and explored the different control options. Just controlling the mouse in simple cases is challenging because there are naively 'screen height' $*$ 'screen width' many actions (2560 in the webtask case).

We explored four control options. First, an agent with a discrete action set of size 256 output as a fully connected layer from the LSTM policy. Second, a similar agent, but instead of using a fully connected layer, we reshape the output of the LSTM to an 8 x 8 grid, and apply layers of convolutions and up-sampling. This approach resembles the manner in which images are generated by generative models such as Variational Autoencoders and Generative Adversarial Networks in the vision setting, and is intended to reflect the spatial correlation in the action space. Third, we considered an approach where the mouse makes discrete movements of a fixed distance in the x and y directions from its current location [4] Fourth, we considered a continuous action space in which the agent outputs real-valued x and y coordinates, which are then binned by an environment wrapper to the nearest pixel.

We found that all of these approaches failed in the ChaseCircle task, and to determine why this was the case we implemented our own version of the environment. This version allowed for removing three of the challenges: (1) ChaseCircle provides a single return at the end of the episode, whereas the test environment can provide rewards at each timestep, (2) ChaseCircle has a moving circle, whereas the test environment can have a stationary, centered circle, and (3) ChaseCircle only gives reward when the mouse is placed on top of the circle, whereas the test environment rewards proportionally to the distance to the circle (reward shaping).

In this simplified setting, the second option for mouse control achieves good performance in all cases except with delayed rewards as shown by the reward curve of figure 12 and logical policy distribution of figure 13. The other approaches still performed poorly. We believe the first approach fails because of the amount of training necessary to individually learn each of the 256 actions, the third due to slow exploration, and the fourth because in the continuous case the agent learns to optimize the entropy term of the loss function, leading to an extremely large value for σ of the Gaussian distribution .

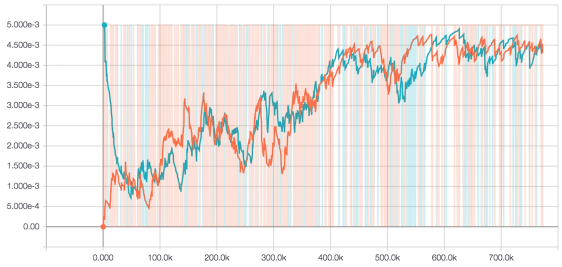


Figure 12. Per-timestep rewards of the convolutional action-space agent during training. The optimal per-timestep reward is 0.005.

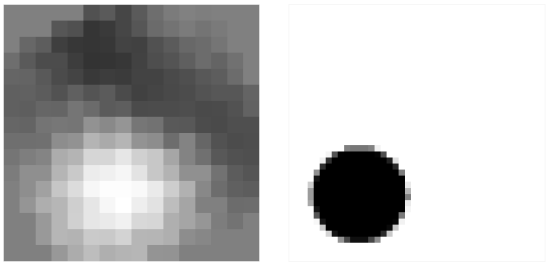


Figure 13. Convolutional action space for mouse control. The left image depicts the action probabilities from the network, and the right depicts the location of the moving circle (the mouse is hidden by the circle).

[3] http://alpha.openai.com/miniwob/index.html

[4] Our understanding is an approach similar to this third one was used to produce initial results on the webtasks, with the primary difference being that the approach used multiple bins for the action distance.

Stanford University

Implementing FeUdal Networks

State-of-the-Art Deep Hierarchical Reinforcement Learning

David Hershey and Blake Wulfe for CS234

Introduction

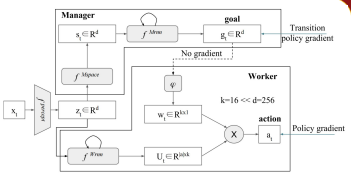
- RL algorithms can struggle in environments that are sparsely-rewarded or that require complex, temporally extended action sequences
- Hierarchical RL methods address these challenges through problem decomposition and temporal abstraction using subgoals
- Autonomously defining subgoals is challenging, particularly in vision-based environments with large state spaces
- Feudal Networks convert the problem of defining subgoals into one of learning good state representations, which can be accomplished using neural networks
- In this project, we implement Feudal Networks, evaluate the model in simple and full scale tasks, and propose changes to alleviate weaknesses of the algorithm

Goals

- Understand the strengths and weaknesses of Feudal Networks relative to baseline, non-hierarchical approaches
- Consider alternative architectures exploiting the use of state-representation changes as subgoals
- Implement an open-source Feudal Network
- Reproduce the results of Vezhnevets et al.

FeUdal Networks

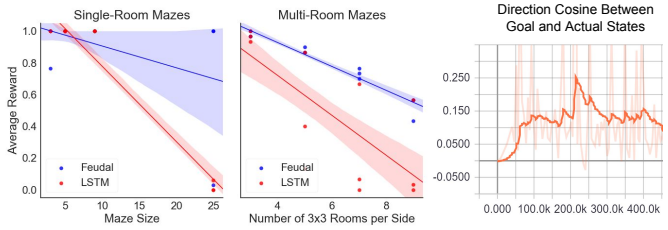
Feudal networks are an HRL architecture designed to learn and exploit the high-level problem structure. The Feudal network is composed of two components, a manager and worker network, that collaborate to solve the RL problem.



Manager Network: The manager network learns a representation of the environment state, which it uses to formulate subgoals for the worker. The manager is rewarded when the worker adheres to those goals and succeeds in the environment.

Worker Network: The worker network enacts a policy within the environment, and receives two rewards - one for matching the goal produced by the manager, and another for succeeding in the environment.

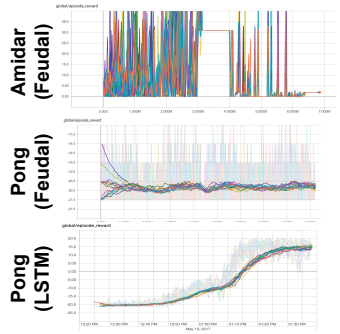
Motivating Results



- We evaluate Feudal Networks against a worker LSTM in variable-size maze environments, which are a classic HRL environment containing intuitive subgoals (doorways) and repeated substructure (rooms)
- In these environments, the Feudal Network is better able to maintain performance as the size of the environment grows due to three advantages:

1. **Structured Exploration:** The smooth temporal variation of the manager goal enforces structured rather than dithering exploration
2. **Subgoal Transfer:** Room-level policies can be reused, speeding learning
3. **Temporal Dependencies:** The manager operates at a higher discount factor with longer backpropagation through time rollouts

Atari Results



The Feudal Network did not converge to a good solution on any tested ATARI games, likely due to the following problems:

1. Prone to entropy collapse, as seen in Amidar
2. Sensitivity to hyperparameters
3. Unstable Worker-Manager interaction

Conclusions

- Our implementation of the Feudal Network demonstrates advantages over baselines in simple problems with sparse rewards and repeated substructure
- The sensitivity of the architecture to hyperparameters and environment factors makes it difficult to tune for more complex problems such as Atari
- The complexity of the interaction between the manager and worker networks make FuNs challenging to tune and debug

Ref: A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. CoRR, abs/1703.01161, 2017.