FeUdal Networks for Hierarchical Reinforcement Learning

Alexander Sasha Vezhnevets [1] **Simon Osindero** [1] **Tom Schaul** [1] **Nicolas Heess** [1] **Max Jaderberg** [1] **David Silver** [1]
Koray Kavukcuoglu [1]

Abstract

We introduce FeUdal Networks (FuNs): a novel architecture for hierarchical reinforcement learning. Our approach is inspired by the feudal reinforcement learning proposal of Dayan and Hinton, and gains power and efficacy by decoupling end-to-end learning across multiple levels – allowing it to utilise different resolutions of time. Our framework employs a Manager module and a Worker module. The Manager operates at a lower temporal resolution and sets abstract goals which are conveyed to and enacted by the Worker. The Worker generates primitive actions at every tick of the environment. The decoupled structure of FuN conveys several benefits – in addition to facilitating very long timescale credit assignment it also encourages the emergence of sub-policies associated with different goals set by the Manager. These properties allow FuN to dramatically outperform a strong baseline agent on tasks that involve long-term credit assignment or memorisation.

1. Introduction

Deep reinforcement learning has recently enjoyed successes in many domains (Mnih et al., 2015; Schulman et al., 2015; Levine et al., 2015; Mnih et al., 2016; Lillicrap et al., 2015). Nevertheless, long-term credit assignment remains a major challenge for these methods, especially in environments with sparse reward signals, such as the infamous Montezuma's Revenge ATARI game. It is symptomatic that the standard approach on the ATARI benchmark suite (Bellemare et al., 2012) is to use an action-repeat heuristic, where each action translates into several (usually 4) consecutive actions in the environment. Yet another dimension of complexity is seen in non-Markovian environments that require memory – these are particularly

[1]DeepMind, London, United Kingdom. Correspondence to: Alexander Sasha Vezhnevets <vezhnick@google.com>.

challenging, since the agent has to learn which parts of experience to store for later, using only a sparse reward signal.

The framework we propose takes inspiration from feudal reinforcement learning (FRL) introduced by Dayan & Hinton (1993), where levels of hierarchy within an agent communicate via explicit goals. Some key insights from FRL are that goals can be generated in a top-down fashion, and that goal setting can be decoupled from goal achievement; a level in the hierarchy communicates to the level below it what must be achieved, but does not specify *how* to do so. Making higher levels reason at a lower temporal resolution naturally structures the agents behaviour into temporally extended sub-policies.

The architecture explored in this work is a fully-differentiable neural network with two levels of hierarchy (though there are obvious generalisations to deeper hierarchies). The top level, the Manager, sets goals at a lower temporal resolution in a latent state-space that is itself *learnt* by the Manager. The lower level, the Worker, operates at a higher temporal resolution and produces primitive actions, conditioned on the goals it receives from the Manager. The Worker is motivated to follow the goals by an intrinsic reward. However, significantly, no gradients are propagated between Worker and Manager; the Manager receives its learning signal from the environment alone. In other words, the Manager learns to select latent goals that maximise extrinsic reward.

The key contributions of our proposal are: (1) A consistent, end-to-end differentiable model that embodies and generalizes the principles of FRL. (2) A novel, approximate *transition policy gradient* update for training the Manager, which exploits the semantic meaning of the goals it produces. (3) The use of goals that are directional rather than absolute in nature. (4) A novel RNN design for the Manager – a dilated LSTM – which extends the longevity of the recurrent state memories and allows gradients to flow through large hops in time, enabling effective back-propagation through hundreds of steps.

Our ablative analysis (Section 5.4) confirms that transitional policy gradient and directional goals are crucial for best performance. Our experiments on a selection of ATARI games (including the infamous Montezuma's re-

venge) and on several memory tasks in the 3D DeepMind Lab environment (Beattie et al., 2016) show that FuN significantly improves long-term credit assignment and memorisation.

2. Related Work

Building hierarchical agents is a long standing topic in reinforcement learning (Sutton et al., 1999; Precup, 2000; Dayan & Hinton, 1993; Dietterich, 2000; Boutilier et al., 1997; Dayan, 1993; Kaelbling, 2014; Parr & Russell, 1998; Precup et al., 1997; 1998; Schmidhuber, 1991; Sutton, 1995; Wiering & Schmidhuber, 1997; Vezhnevets et al., 2016; Bacon et al., 2017). The options framework (Sutton et al., 1999; Precup, 2000) is a popular formulation for considering the problem with a two level hierarchy. The bottom level – an option – is a sub-policy with a termination condition, which takes in environment observations and outputs actions until the termination condition is met. An agent picks an option using its policy-over-options (the top level) and subsequently follows it until termination, at which point the policy-over-options is queried again and the process continues. Options are typically learned using sub-goals and 'pseudo-rewards' that are provided explicitly (Sutton et al., 1999; Dietterich, 2000; Dayan & Hinton, 1993). For a simple, tabular case (Wiering & Schmidhuber, 1997; Schaul et al., 2015), each state can be used as a sub-goal. Given the options, a policy-over-options can be learned using standard techniques by treating options as actions. Recently (Tessler et al., 2016; Kulkarni et al., 2016) have demonstrated that combining deep learning with pre-defined sub-goals delivers promising results in challenging environments like Minecraft and Atari, however sub-goal discovery was not addressed.

A recent work of (Bacon et al., 2017) shows the possibility of learning options jointly with a policy-over-options in an end-to-end fashion by extending the policy gradient theorem to options. When options are learnt end-to-end, they tend to degenerate to one of two trivial solutions: (i) only one active option that solves the whole task; (ii) a policy-over-options that changes options at every step, micro-managing the behaviour. Consequently, regularisers (Bacon et al., 2017; Vezhnevets et al., 2016) are usually introduced to steer the solution towards multiple options of extended length. This is believed to provide an inductive bias towards re-usable temporal abstractions and to help generalisation.

A key difference between our approach and the options framework is that in our proposal the top level produces a meaningful and explicit goal for the bottom level to achieve. Sub-goals emerge as directions in the latent state-space and are naturally diverse. We also achieve significantly better scores on ATARI than Option-Critic (section 5).
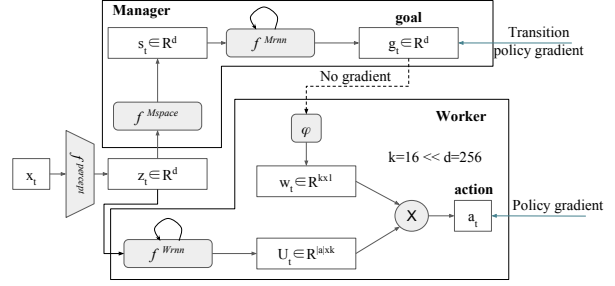

Figure 1. The schematic illustration of FuN (section 3)

There has also been a significant progress in non-hierarchical deep RL methods by using auxiliary losses and rewards. (Bellemare et al., 2016a) have significantly advanced the state-of-the-art on Montezuma's Revenge by using pseudo-count based auxiliary rewards for exploration, which stimulate agents to explore new parts of the state space. The recently proposed UNREAL agent (Jaderberg et al., 2016) also demonstrates a strong improvement by using unsupervised auxiliary tasks to help refine its internal representations. We note that these benefits are orthogonal to those provided by FuN, and that both approaches could be combined with FuN for even greater effect.

3. The model

What is FuN? FuN is a modular neural-network consisting of two modules – the Worker and the Manager. The Manager internally computes a latent state representation s_t and outputs a goal vector g_t. The Worker produces actions conditioned on external observation, its own state, and the Managers goal. The Manager and the Worker share a perceptual module which takes an observation from the environment x_t and computes a shared intermediate representation z_t. The Manager's goals g_t are trained using an approximate *transition policy gradient*. This is a particularly efficient form of policy gradient training that exploits the knowledge that the Worker's behaviour will ultimately align with the goal directions it has been set. The Worker is then trained via intrinsic reward to produce actions that cause these goal directions to be achieved. Figure 1a illustrates the overall design and the following equations describe the forward dynamics of our network:

$$z_t = f^{\text{percept}}(x_t); s_t = f^{Mspace}(z_t) \qquad (1)$$

$$h_t^M, \hat{g}_t = f^{Mrnn}(s_t, h_{t-1}^M); g_t = \hat{g}_t/||\hat{g}_t||; \qquad (2)$$

$$w_t = \phi\left(\sum_{i=t-c}^{t} g_i\right) \qquad (3)$$

$$h^W, U_t = f^{Wrnn}(z_t, h_{t-1}^W); \pi_t = SoftMax(U_t w_t) \qquad (4)$$

where both the Manager and the Worker are recurrent. Here h^M and h^W correspond to the internal states of the Man-

ager and the Worker respectively. A linear transform ϕ maps a goal g_t into an embedding vector $w_t \in \mathbb{R}^k$, which is then combined via product with matrix U_t (Workers output) to produce policy π – vector of probabilities over primitive actions. The next section provides the details on goal embedding and the following sections 3.2, 3.3 describes how FuN is trained.

3.1. Goal embedding

The goal g modulates the policy via a multiplicative interaction in a low dimensional goal-embedding space $\mathbb{R}^k, k \ll d$. The Worker first produces an embedding vector for every action, represented by rows of matrix $U \in \mathbb{R}^{|a| \times k}$ (eq. 4). To incorporate goals from the Manager, the last c goals are first pooled by summation and then embedded into a vector $w \in \mathbb{R}^k$ using a linear projection ϕ (eq. 3). The projection ϕ is linear, with no biases, and is learnt with gradients coming from the Worker's actions. The embedding matrix U is then combined with the goal embedding w via a matrix-vector product (eq. 4). Since ϕ has no biases it can never produce a constant non-zero vector – which is the only way the setup could ignore the Manager's input. This makes sure that the goal output by the Manager always influences the final policy. Notice how, due to pooling of goals over several time-steps, the conditioning from the Manager varies smoothly.

3.2. Learning

We consider a standard reinforcement learning setup. At each step t, the agent receives an observation x_t from the environment and selects an action a_t from a finite set of possible actions. The environment responds with a new observation x_{t+1} and a scalar reward r_t. The process continues until the terminal state is reached, after which it restarts. The goal of the agent is to maximise the discounted return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ with $\gamma \in [0, 1]$. The agent's behaviour is defined by its action-selection policy π. FuN produces a distribution over possible actions (a stochastic policy) as defined in eq. 4.

The conventional wisdom would be to train the whole architecture monolithically through gradient descent on either the policy directly or via TD-learning. Notice, that since FuN is fully differentiable we could train it end-to-end using a policy gradient algorithm operating on the actions taken by the Worker. The outputs g of the Manager would be trained by gradients coming from the Worker. This, however would deprive Manager's goals g of any semantic meaning, making them just internal latent variables of the model. We propose instead to independently train Manager to predict advantageous directions (transitions) in state space and to intrinsically reward the Worker to follow these directions. If the Worker can fulfil the goal of moving in these directions (as it is rewarded for doing), then we ought to end up taking advantageous trajectories through

state-space. We formalise this in the following update rule for the Manager:

$$\nabla g_t = A_t^M \nabla_\theta d_{\cos}(s_{t+c} - s_t, g_t(\theta)), \qquad (5)$$

where $A_t^M = R_t - V_t^M(x_t, \theta)$ is the Manager's advantage function, computed using a value function estimate $V_t^M(x_t, \theta)$ from the internal critic; $d_{\cos}(\alpha, \beta) = \alpha^T \beta / (|\alpha||\beta|)$ is the cosine similarity between two vectors. Note: the dependence of s on θ is ignored when computing $\nabla_\theta d_{\cos}$ – this avoids trivial solutions. Notice that now g_t acquires a semantic meaning as an advantageous direction in the latent state space at a horizon c, which defines the temporal resolution of the Manager.

The intrinsic reward that encourages the Worker to follow the goals is defined as:

$$r_t^I = 1/c \sum_{i=1}^{c} d_{\cos}(s_t - s_{t-i}, g_{t-i}) \qquad (6)$$

We use directions because it is more feasible for the Worker to be able to reliably cause directional shifts in the latent state than it is to assume that the Worker can take us to (potentially) arbitrary new absolute locations. It also gives a degree of invariance to the goals and allows for structural generalisation – the same directional sub-goal g can invoke a sub-policy that is valid and useful in a large part of the latent state space; e.g. evade an enemy, swim up for air, etc. We compare absolute against directional goals empirically in section 5.4.

The original feudal reinforcement learning formulation of Dayan & Hinton (1993) advocated completely concealing the reward from the environment from lower levels of hierarchy. In practice we take a softer approach by adding an intrinsic reward for following the goals, but retaining the environment reward as well. The Worker is then trained to maximise a weighted sum $R_t + \alpha R_t^I$, where α is a hyper-parameter that regulates the influence of the intrinsic reward. The Workers policy π can be trained to maximise intrinsic reward by using any off-the shelf deep reinforcement learning algorithm. Here we use an advantage actor critic (Mnih et al., 2016):

$$\nabla \pi_t = A_t^D \nabla_\theta \log \pi(a_t | x_t; \theta) \qquad (7)$$

The advantage function estimator $A_t^D = (R_t + \alpha R_t^I - V_t^D(x_t; \theta))$ is calculated using an internal critic, which estimates the value functions for both rewards.

Note that the Worker and Manager can potentially have different discount factors γ for computing the return. This allows, for instance, the Worker to be more greedy and focus on immediate rewards while the Manager can consider a long-term perspective.

3.3. Transition Policy Gradients

We now motivate our proposed update rule for the Manager as a novel form of policy gradient with respect to a *model* of the Worker's behaviour. Consider a high-level policy $o_t = \mu(s_t, \theta)$ that selects among sub-policies (possibly from a continuous set), where we assume for now that these sub-policies are fixed duration behaviours (lasting for c steps). Corresponding to each sub-policy is a transition distribution, $p(s_{t+c}|s_t, o_t)$, that describes the distribution of states that we end up at the end of the sub-policy, given the start state and the sub-policy enacted. The high-level policy can be composed with the transition distribution to give a 'transition policy' $\pi^{TP}(s_{t+c}|s_t) = p(s_{t+c}|s_t, \mu(s_t, \theta))$ describing the distribution over end states given start states. It is valid to refer to this as a policy because the original MDP is isomorphic to a new MDP with policy π^{TP} and transition function $s_{t+c} = \pi^{TP}(s_t)$ (i.e. the state always transitions to the end state picked by the transition policy). As a result, we can apply the policy gradient theorem to the transition policy π^{TP}, so as to find the performance gradient with respect to the policy parameters,

$$\nabla_\theta \pi_t^{TP} = \mathbb{E}\left[(R_t - V(s_t))\nabla_\theta \log p(s_{t+c}|s_t, \mu(s_t, \theta))\right] \tag{8}$$

In general, the Worker may follow a complex trajectory. A naive application of policy gradients requires the agent to learn from samples of these trajectories. But if we know where these trajectories are likely to end up, by modelling the transitions, then we can skip directly over the Worker's behaviour and instead follow the policy gradient of the predicted transition. FuN assumes a particular form for the transition model: that the direction in state-space, $s_{t+c} - s_t$, follows a von Mises-Fisher distribution. Specifically, if the mean direction of the von Mises-Fisher distribution is given by $g(o_t)$ (which for compactness we write as g_t) we would have $p(s_{t+c}|s_t, o_t) \propto e^{d_{cos}(s_{t+c}-s_t, g_t)}$. If this functional form *were* indeed correct, then we see that our proposed update heuristic for the Manager, eqn.5, is in fact the proper form for the transition policy gradient arrived at in eqn.8.

Note that the Worker's intrinsic reward (eqn. 6) is based on the log-likelihood of state trajectory. Through that the FuN architecture actively encourages the functional form of the transition model to hold true. Because the Worker is learning to achieve the Manager's direction, its transitions should, over time, closely follow a distribution around this direction, and hence our approximation for transition policy gradients should hold reasonably well.

4. Architecture details

This section provides the particular details of the model as described in section 3. The perceptual module $f^{percept}$ is a convolutional network (CNN) followed by a fully connected layer. The CNN has a first layer with 16 8x8 filters of stride 4, followed by a layer with with 32 4x4 filters of stride 2. The fully connected layer has 256 hidden units. Each convolutional and fully-connected layer is followed by a rectifier non-linearity[1]. The state space which the Manager implicitly models in formulating its goals is computed via f^{Mspace}, which is another fully connected layer followed by a rectifier non-linearity. The dimensionality of the embedding vectors, w, is set as $k = 16$. To encourage exploration in transition policy, at every step with a small probability ϵ we emit a random goal sampled from a uni-variate Gaussian.

The Worker's recurrent network f^{Wrnn} is a standard LSTM (Hochreiter & Schmidhuber, 1997). For the Manager's recurrent network, f^{Mrnn}, we propose a novel design – the dilated LSTM, which is introduced in the next section. Both f^{Mrnn} and f^{Wrnn} have 256 hidden units.

4.1. Dilated LSTM

We propose a novel RNN architecture for the Manager, which operates at lower temporal resolution than the data stream. The main contribution here is the inductive bias towards slowly varying outputs, which have very long-term temporal dependencies. We define a dilated LSTM analogously to dilated convolutional networks (Yu & Koltun, 2016). For a dilation radius r let the full state of the network be $h = \{\hat{h}^i\}_{i=1}^r$, i.e. it is composed of r separate groups of sub-states or 'cores'. At time t the network is governed by the following equations: $\hat{h}_t^{t\%r}, g_t = LSTM(s_t, \hat{h}_{t-1}^{t\%r}; \theta^{LSTM})$, where % denotes the modulo operation and allows us to indicate which group of cores is currently being updated. We make the parameters of the LSTM network θ^{LSTM} explicit to stress that the same set of parameters governs the update for each of the r groups within the dLSTM.

At each time step only the corresponding part of the state is updated and the output is pooled across the previous c outputs. This allows the r groups of cores inside the dLSTM to preserve the memories for long periods, yet the dLSTM as a whole is still able to process and learn from every input experience, and is also able to update its output at every step. This idea is similar to clockwork RNNs (Koutník et al., 2014), however there the top level "ticks" at a fixed, slow pace, whereas the dLSTM observes all the available training data instead. In the experiments we set $r = 10$, and this was also used as the predictions horizon, c.

5. Experiments

The goal of our experiments is to demonstrate that FuN learns non-trivial, helpful, and interpretable sub-policies

---

[1]This is substantially the same CNN as in (Mnih et al., 2016; 2015), the only difference is that in the pre-processing stage we retain all colour channels.
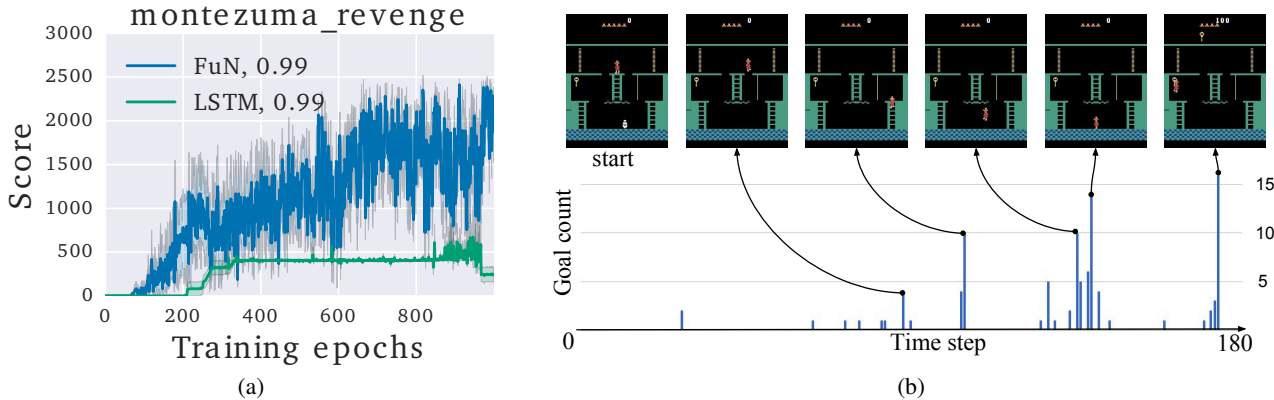
Figure 2. a) Learning curve on Montezuma's Revenge b) This is a visualisation of sub-goals learnt by FuN in the first room. For each time step we compute the latent state s_t and the corresponding goal g_t. We then find a future state for which $\cos(s_t - s_t, g_t)$ is maximized. The plot corresponds to the number of past states for which a frame maximizes the goal - i.e. the taller the bar, the more frequently that state was a maximizer of the goal for some previous state. Notice that FuN has learnt a semantically meaningful sub-goals – the tall bars in the plot (i.e. consistent goals) correspond to interpretably useful waypoints in Montezuma.

and sub-goals, and also to validate components of the architecture. We start by describing technical details of the experimental setup and then present results on Montezuma's revenge – an infamously hard ATARI game – in section 5.1. Section 5.2 presents results on more ATARI games and extensively compares FuN to LSTM baseline with different discount factors and BPTT lengths. In section 5.3 we present results on a set of visual memorisation tasks in 3D environment. Section 5.4 presents an ablation study of FuN, validating our design choices.

Baseline. Our main baseline is a recurrent LSTM network on top of a representation learned by a CNN. The LSTM (Hochreiter & Schmidhuber, 1997) architecture is a widely used recurrent network and it was demonstrated to perform very well on a suite of reinforcement learning problems (Mnih et al., 2016). LSTM uses 316 hidden units[2] and its inputs are the feature representation of an observation and the previous action of the agent. Action probabilities and the value function estimate are regressed from its hidden state. All the methods the same CNN architecture, input pre-processing, and an action repeat of 4.

Optimisation. We use the A3C method (Mnih et al., 2016) for all reinforcement learning experiments. It was shown to achieve state-of-the-art results on several challenging benchmarks (Mnih et al., 2016). We cut the trajectory and run backpropagation through time (BPTT) (Mozer, 1989) after K forward passes of a network or if a terminal signal is received. For FuN $K = 400$, for LSTM, unless otherwise stated, $K = 40$. We discuss different choice of K for LSTM in section 5.2. The optimization process runs 32 asynchronous threads using shared RMSProp. There are 3 hyper-parameters in

FuN and 2 in the LSTM baselines. For each method, we ran 100 experiments, each using randomly sampled hyper-parameters. Learning rate and entropy penalty were sampled from a $LogUniform(10^{-4}, 10^{-3})$ interval for LSTM. For FuN the learning rate was sampled from $LogUniform(10^{-4.5}, 10^{-3.5})$, to account for higher gradients due to longer BPTT unrolls. The learning rate was linearly annealed from a sampled value to half the initial rate for all agents. To explore intrinsic motivation in FuN, we sample its weight $\alpha \sim Uniform(0, 1)$. We define a training epoch as one million observations. When reporting learning curves, we plot the average episode score of the top 5 agents (according to the final score) against the training epochs. For all ATARI experiments we clip the reward to $[-1, +1]$ interval

5.1. Montezuma's revenge

Montezuma's revenge is one of the hardest games available through the ALE (Bellemare et al., 2012). The game is infamous for challenging agents with lethal traps and sparse rewards. We had to broaden and intensify our hyper-parameter search for the LSTM baseline to see any progress at all for that model. We have experimented with many different hyper-parameter configurations for LSTM baseline, for instance expanding learning rate search to $LogUniform(10^{-3}, 10^{-2})$, and we report on the configuration that worked best. We use a small discount 0.99 for LSTM; for FuN we use 0.99 in Worker and 0.999 in Manager. Figure 2b analyses the sub-goals learnt by FuN in the first room. They turn out to be meaningful milestones, which bridge the agents progress to its first extrinsic reward – picking up the key. Interestingly, two of the learnt sub-goals correspond to roughly the same locations as the ones hand-crafted in (Kulkarni et al., 2016) (ladder and key), but here they are learnt by the agent itself. Figure 2a

---

[2]This choice means that FuN and the LSTM baseline to have roughly the same number of total parameters.
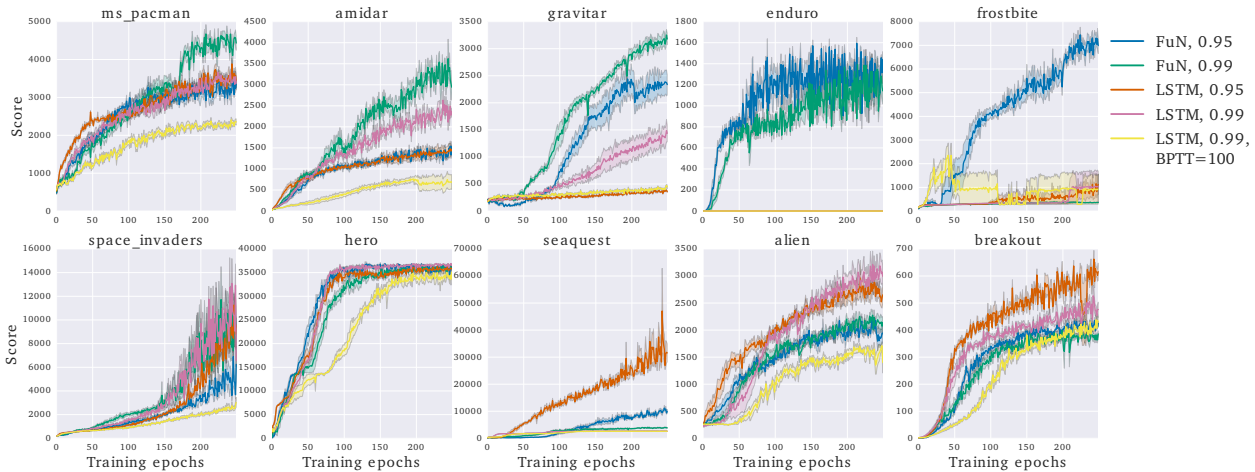
Figure 3. ATARI training curves. Epochs corresponds to a million training steps of an agent. The value is the average per episode score of top 5 agents, according to the final score. We used two different discount factors 0.95 and 0.99.
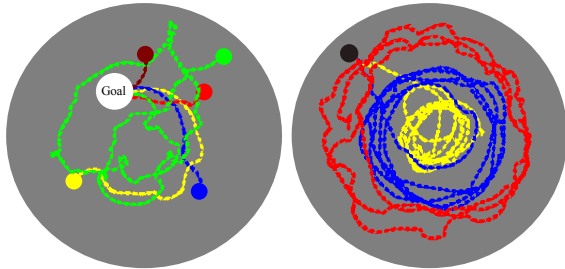

Figure 4. FuN in water maze, top down view. The left plot visualises FuN trajectories during one episode. The first trajectory (green) performs a search for the target in different locations, while subsequent ones (other colours) perform searches along a circle of a fixed radius matched to that of the target, always finding the target. The right plot visualises different learnt sub-policies, produced by sampling a random g and fixing it for 200 steps. Each colour corresponds to a different g, the black circle represents the starting location.

plots the learning curves. Notice how FuN starts learning much earlier and achieves much higher scores. It takes > 300 epochs for LSTM to reach the score 400, which corresponds to solving the first room (take the key, open a door); it stagnates at that score until about 900 epochs, when it starts exploring further. FuN solves the first room in less than 200 epochs and immediately moves on to explore further, eventually visiting several other rooms and scoring up to 2600 points.

5.2. ATARI

Experiments in this section validate that the capabilities of FuN go beyond what standard tools for long-term credit assignment – discount factors and BPTT unroll length – can provide for a baseline LSTM agent. We use two discounts 0.99 and 0.95 for both FuN and LSTM agents. (For the experiments on FuN only the discount for the Manager changes, while the Worker's discount is fixed at
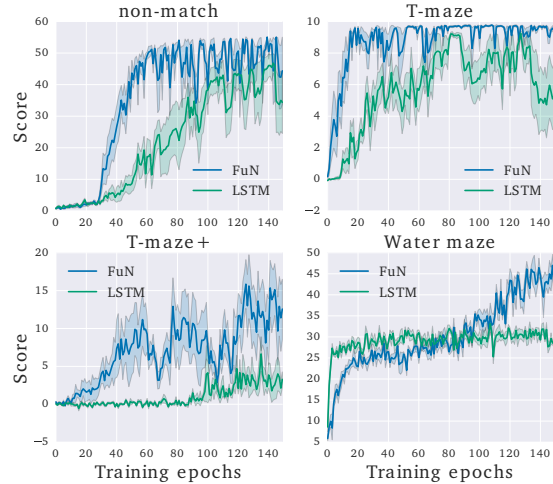

Figure 5. Training curves for memory tasks on Labyrinth.

0.95.) For the LSTM we explore BPTT of 40 and 100, while for FuN we use a BPTT unroll of 400. For LSTM with BPTT 100 we search for learning rate in the interval $LogUniform(10^{-4.5}, 10^{-3.5})$, as for FuN. We use a diverse set of ATARI games, some of which involve long-term credit assignment and some which are more reactive.

Figure 3 plots the learning curves. A few categories emerge. On Ms. Pacman, Amidar, and Gravitar FuN with a low Manager discount of 0.99 strongly outperforms all other methods. All of these games are known to require long-term reasoning to play well. Enduro stands out as all the LSTM agents completely fail at it. In this game the agent controls a racing car and scores points for overtaking other racers; this requires accelerating and steering for significant amount of time before the first reward is experienced. Frostbite is a hard game (Vezhnevets et al., 2016; Lake et al., 2016) that requires both long-term credit assignment and good exploration. The best-performing frost-

bite agent is FuN with 0.95 Manager discount, which outperforms the rest by a factor of 7. On Hero and Space Invaders all agents perform equally well. On Seaquest and Breakout, the baseline LSTM with a more aggressive discount of 0.95 is the best. This suggests that in these games long-term credit assignment is not important and the agent is better off optimising more immediate rewards in a greedy fashion. Alien is the only game where using different discounts doesn't meaningfully influence the agents performance; here we see the baseline LSTM outperforms our FuN model, although both still achieve a satisfactory scores. We provide qualitative analysis of subpolicies learnt on Seaquest in supplementary material.

Note how using an unroll for BPTT=100 in the baseline LSTM significantly hurts its performance (hence we do not explore longer unrolls), while FuN performs very well with BPTT of 400 thanks to its ability to leverage the dLSTM. Being able to train a recurrent network over very long sequences could be an enabling tool for many memory related task, as we demonstrate in section 5.3.

Option-critic architecture (Bacon et al., 2017) is, to the best of our knowledge, the only other end-to-end trainable system with sub-policies. The experimental results for Option-Critic on 4 ATARI (Bacon et al., 2017) games show scores similar those from a flat DQN (Mnih et al., 2015) baseline agent. Notice that our baseline (Mnih et al., 2016) is much stronger than DQN. We also ran FuN on the same games as Option-Critic (Asterix, Ms. Pacman, Seaquest and Zaxxon) and after 200 epochs it achieves a similar score on Seaquest, doubles it on Ms. Pacman, more than triples it on Zaxxon and gets more than 20x improvement on Asterix (see supplementary material for plots).
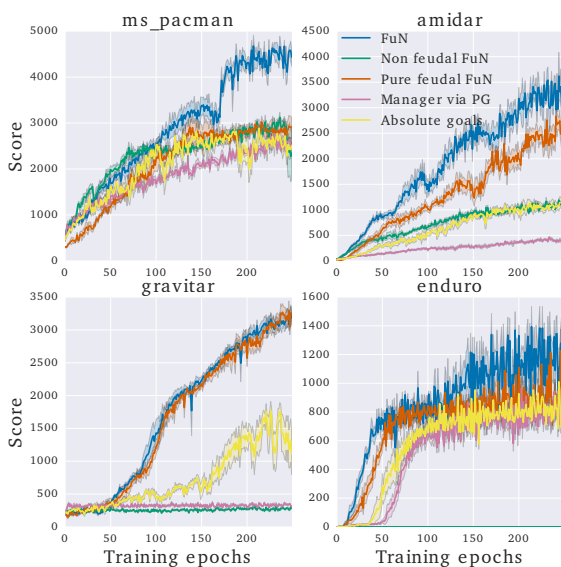

Figure 6. Ablative analysis

5.3. Memory in Labyrinth

DeepMind Lab (Beattie et al., 2016) is a first-person 3D game platform extended from OpenArena. It's a visually complex 3D environment with agent actions corresponding to movement and orientation. We use 4 different levels that test long-term credit assignment and visual memory:

Water maze is a reproduction of the Morris water maze experiment (Morris, 1981) from the behavioural science literature. An agent is dropped into a circular pool of water with a concealed platform at unknown random location. The agent can move around and upon stepping on the platform it receives a reward and the trial restarts. The platform remains in the same location for the rest of the episode, while agent starts each trial at a random location. The walls of the pool are decorated with visual cues to assist localisation.

T-maze is another classic animal cognition test. The agent spawns in a small T-shaped maze. Two objects with randomly chosen shape and colour are spawned at the left and right "baiting" locations. One of them is assigned a reward of +1 and the other a reward of -1. When the agent collects one of the objects, it receives the reward and is re-spawned at the beginning of the T-maze. The objects are also re-instantiated in the same locations and with the same rewards on the re-spawn event. The agent should remember which object gives the positive reward across re-spawns and collect it as many times as possible within the fixed time given for the episode. **T-maze+** is a modification of T-maze, where at each trial the length of corridors can vary, adding additional dimension of complexity.

Non-match is a visual memorisation task. Each trial begins in small room with an out of reach object being displayed in one of two display pods. There is a pad in the middle, which upon touching, the agent is rewarded with 1 point, and is teleported to a second room which has two objects in it, one of which matches the object in the previous room. Collecting the matching object gives a reward of -10 points, collecting the non matching object gives a reward of 10 points. Once either is collected, the agent is teleported back to the first room, with the same object being shown.

For all agents we include reward as a part of the observation. Figure 5 plots the learning curves. FuN consistently outperforms the LSTM baseline – it learns faster and also reaches a higher final reward. We analyse the FuN agent's behaviour in more detail in Figure 4b. It demonstrates that FuN learns meaningful sub-policies, which are then efficiently integrated with memory to produce rewarding behaviour. Interestingly, the LSTM agent doesn't appear to use its memory for water maze task at all, always circling the maze at the roughly the same radius.
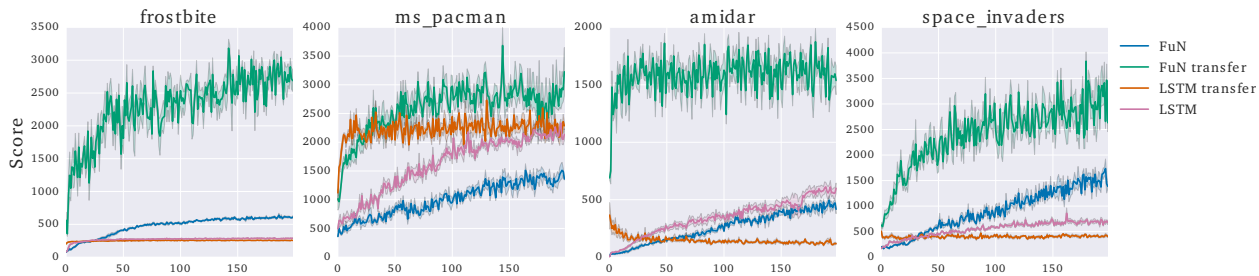
Figure 7. Action repeat transfer

5.4. Ablative analysis

This section empirically validates the main innovations of this paper: transition policy gradient for training the Manager; relative rather than absolute goals; intrinsic motivation for the Worker. First we consider a 'non-Feudal' FuN – it has exactly the same network architecture as FuN, but the Managers output g is trained with gradients coming directly from the Worker and no intrinsic reward is used, much like in Option-Critic architecture (Bacon et al., 2017). Second, g is learnt using a standard policy gradient approach with the Manager emitting the mean of a Gaussian distribution from which goals are sampled (as if the Manager were solving a continuous control problem (Schulman et al., 2016; Mnih et al., 2016; Lillicrap et al., 2015)). Third, we explore a variant of FuN in which g specifies absolute, rather than relative/directional, goals (and the Worker's intrinsic reward is adjusted accordingly) but otherwise everything is the same. The experiments (Figure 6) reveal that, although alternatives do work to some degree their performance is significantly inferior. We also evaluate a purely feudal version of FuN – in which the Worker is trained from the intrinsic reward alone. This ablation performs better than other, but still inferior to the full FuN approach. It shows that allowing the Worker to experience the external reward is beneficial.

5.5. ATARI action repeat transfer

One of the advantages of FuN is the clear separation of duties between Manager and Worker. The Manager learns a transition policy, while the Worker learns to operate primitive actions to enact these transitions. This transition policy is invariant to the underlying embodiment of the agent – the way its primitive actions translate into state space transitions. Potentially, the transition policy can be transferred between agents with different embodiment – e.g. robot models with different bodies or different operational frequency. We provide evidence towards that possibility by transferring policies across agents with different action repeat on ATARI. Action repeat is a heuristic used in all successful agents (Mnih et al., 2015; 2016; Bellemare et al., 2016b; Vezhnevets et al., 2016). It enables better exploration, eases credit assignment, and saves computation by repeating an action chosen by the agent several ($= 4$) times.

To perform transfer, we initialise the FuN system with parameters extracted from an agent trained with action repeat of 4 and then make the following adjustments: (i) we accordingly adjust the discounts for all rewards; (ii) we increase the dilation of the dLSTM by a factor of 4; (iii) we increase the Manager's goal horizon c by a factor of 4. (These modifications adapt all the "hard-wired" but explicitly temporally sensitive aspects of the agent.) We then train this agent without action repeat. As a baseline we use an LSTM agent transferred in a similar way (with adjusted discounts) as well as FuN and LSTM agents trained without action repeat from scratch. Figure 7 shows the corresponding learning curves. The transferred FuN agent (green curve) significantly outperforms every other method. Furthermore it shows positive transfer on each environment, whereas LSTM only shows positive transfer on Ms. Pacman.

6. Discussion and future work

How to create agents that can learn to decompose their behaviour into meaningful primitives and then reuse them to more efficiently acquire new behaviours is a long standing research question. The solution to this question may be an important stepping stone towards agents with general intelligence and competence. This paper introduced FeUdal Networks, a novel architecture that formulates sub-goals as directions in latent state space, which, if followed, translate into a meaningful behavioural primitives. FuN clearly separates the module that discovers and sets sub-goals from the module that generates the behaviour through primitive actions. This creates a natural hierarchy that is stable and allows both modules to learn in complementary ways. Our experiments clearly demonstrate that this makes long-term credit assignment and memorisation more tractable. This also opens many avenues for further research, for instance: deeper hierarchies can be constructed by setting goals at multiple time scales, scaling agents to truly large environments with sparse rewards and partial observability. The modular structure of FuN is also lends itself to transfer and multitask learning – learnt behavioural primitives can be re-used to acquire new complex skills, or alternatively the transitional policies of the Manager can be transferred to agents with different embodiment.

Acknowledgements

References

Bacon, Pierre-Luc, Precup, Doina, and Harb, Jean. The option-critic architecture. In *AAAI*, 2017.

Beattie, Charles, Leibo, Joel Z., Teplyashin, Denis, Ward, Tom, Wainwright, Marcus, Küttler, Heinrich, Lefrancq, Andrew, Green, Simon, Valdés, Víctor, Sadik, Amir, Schrittwieser, Julian, Anderson, Keith, York, Sarah, Cant, Max, Cain, Adam, Bolton, Adrian, Gaffney, Stephen, King, Helen, Hassabis, Demis, Legg, Shane, and Petersen, Stig. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016a.

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

Bellemare, Marc G., Ostrovski, Georg, Guez, Arthur, Thomas, Philip S., and Munos, Rémi. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016b.

Boutilier, Craig, Brafman, Ronen I, and Geib, Christopher. Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *IJCAI*, 1997.

Dayan, Peter. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 1993.

Dayan, Peter and Hinton, Geoffrey E. Feudal reinforcement learning. In *NIPS*. Morgan Kaufmann Publishers, 1993.

Dietterich, Thomas G. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 2000.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 1997.

Jaderberg, Max, Mnih, Volodymyr, Czarnecki, Wojciech Marian, Schaul, Tom, Leibo, Joel Z, Silver, David, and Kavukcuoglu, Koray. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Kaelbling, Leslie Pack. Hierarchical learning in stochastic domains: Preliminary results. In *ICML*, 2014.

Koutník, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Jürgen. A clockwork rnn. In *ICML*, 2014.

Kulkarni, Tejas D., Narasimhan, Karthik R., Saeedi, Ardavan, and Tenenbaum, Joshua B. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.

Lake, Brenden M, Ullman, Tomer D, Tenenbaum, Joshua B, and Gershman, Samuel J. Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*, 2016.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. *ICML*, 2016.

Morris, Richard GM. Spatial localization does not require the presence of local cues. *Learning and motivation*, 12 (2):239–260, 1981.

Mozer, Michael C. A focused back-propagation algorithm for temporal pattern recognition. *Complex systems*, 1989.

Parr, Ronald and Russell, Stuart. Reinforcement learning with hierarchies of machines. *NIPS*, 1998.

Precup, Doina. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts, 2000.

Precup, Doina, Sutton, Richard S, and Singh, Satinder P. Planning with closed-loop macro actions. Technical report, 1997.

Precup, Doina, Sutton, Richard S, and Singh, Satinder. Theoretical results on reinforcement learning with temporally abstract options. In *European Conference on Machine Learning (ECML)*. Springer, 1998.

Schaul, Tom, Horgan, Dan, Gregor, Karol, and Silver, David. Universal value function approximators. *ICML*, 2015.

Schmidhuber, Jürgen. Neural sequence chunkers. Technical report, 1991.

Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I, and Abbeel, Pieter. Trust region policy optimization. In *ICML*, 2015.

Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. *ICLR*, 2016.

Sutton, Richard S. Td models: Modeling the world at a mixture of time scales. In *ICML*, 1995.

Sutton, Richard S, Precup, Doina, and Singh, Satinder. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999.

Tessler, Chen, Givony, Shahar, Zahavy, Tom, Mankowitz, Daniel J, and Mannor, Shie. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.

Vezhnevets, Alexander, Mnih, Volodymyr, Osindero, Simon, Graves, Alex, Vinyals, Oriol, Agapiou, John, and kavukcuoglu, koray. Strategic attentive writer for learning macro-actions. In *NIPS*, 2016.

Wiering, Marco and Schmidhuber, Jürgen. Hq-learning. *Adaptive Behavior*, 1997.

Yu, Fisher and Koltun, Vladlen. Multi-scale context aggregation by dilated convolutions. *ICLR*, 2016.