FedRL: Federated Reinforcement Learning

Ashwinee Panda, William Song, David Vendrow

Abstract

As artificial intelligence becomes increasingly ubiquitous in technology, it interacts more directly with the end user. This has led to the popularization of deep reinforcement learning in machine learning for sequential decision making. By representing the end user's actions and consequences in a reinforcement learning setting, difficult user behavior problems can be condensed to questions of agents and environments.

In scenarios where large quantities of data are generated on an ongoing basis across distributed devices, it is possible to train these deep reinforcement learning networks with federated learning algorithms. We outline one implementation of a particular federated reinforcement learning algorithm, and outline its advantages over existing alternatives in distributed reinforcement learning. To our knowledge, this is the first reinforcement-learning specific adaptation of federated learning.

We train an ensemble of agents, encouraging them to explore and come to different models, but periodically set the weights of each agent in the ensemble of a reward-weighted average of their weights. Our goal is to conduct efficient training of a large group of actors, reasoning that agents which discover better local optima sooner will have their weights more strongly represented in the final product because they will have higher reward.

We conduct experiments with federated policy gradient and federated deep-Q. In both, results are promising -even naive implementations of the federated reinforcement learning algorithm without much hyperparameter search or optimization provide a nontrivial increase over centralized SGD. In future we would want to consider parameters such as a decaying/increasing communication round parameter, allowing agents to explore more or less between weighted averages so that we can provide faster convergence which is still to a lower error floor.

1 PROBLEM DEFINITION AND MOTIVATION

As artificial intelligence becomes increasingly ubiquitous in technology, it interacts more directly with the end user. From text prediction to virtual assistants, machine learning is increasingly pushed to the periphery of a service's tech stack. Where once it was the norm for any device to communicate with a server that hosted a central model to retrieve an inference, universal increases in computing capability now allow machine learning to be run on mobile phones, laptop computers, and even in the browser. We focus on the consumer application setting, where hundreds of thousands or millions of users generating a relatively small stream of data is normal. However, our analysis is context-agnostic and can be applied to the Internet-of-Things setting or any environment where the desired data is spread across many devices. We are inspired by Facebook's recent Horizon[**?**] system for scalable reinforcement learning as applied directly to the consumer data setting.

1.1 DEEP REINFORCEMENT LEARNING

For the purpose of this paper we will only be talking about deep learning, and more specifically reinforcement learning. First we will quickly revisit why deep learning is dominating nearly all conversations about machine learning. Today machine learning is intrinsically tied to big data; the more data we have, the better predictions we can make. And as we analyze more data, we can afford to use models with more representational capacity.

Specifically, we focus on deep neural networks, which utilize additional layers to achieve better accuracy at the cost of needing large amounts of data to train. In practice, deep reinforcement learning agents need many iterations in their environment to train to any reasonable accuracy. The actual training is usually done on a GPU or GPU cluster which has enough memory to store the millions of requisite datapoints, although outlining the deep network can be done on a laptop computer. Our experiments use Ray[5], a framework for quickly defining and setting up reinforcement learning networks.

For the consumer-facing setting, deep reinforcement learning models can be used for sequential decision-making. To predict when to display an advertisement to a user, what content to show on a newsfeed, etc. Because consumer applications with millions of users will be generating enormous amounts of data, it is clearly attractive to use **deep networks.**

1.2 PROBLEMS WITH DEEP RL

Of course, deep reinforcement learning is nowhere near ubiquitous in industry machine learning. It still faces a few key challenges in security and scalability.

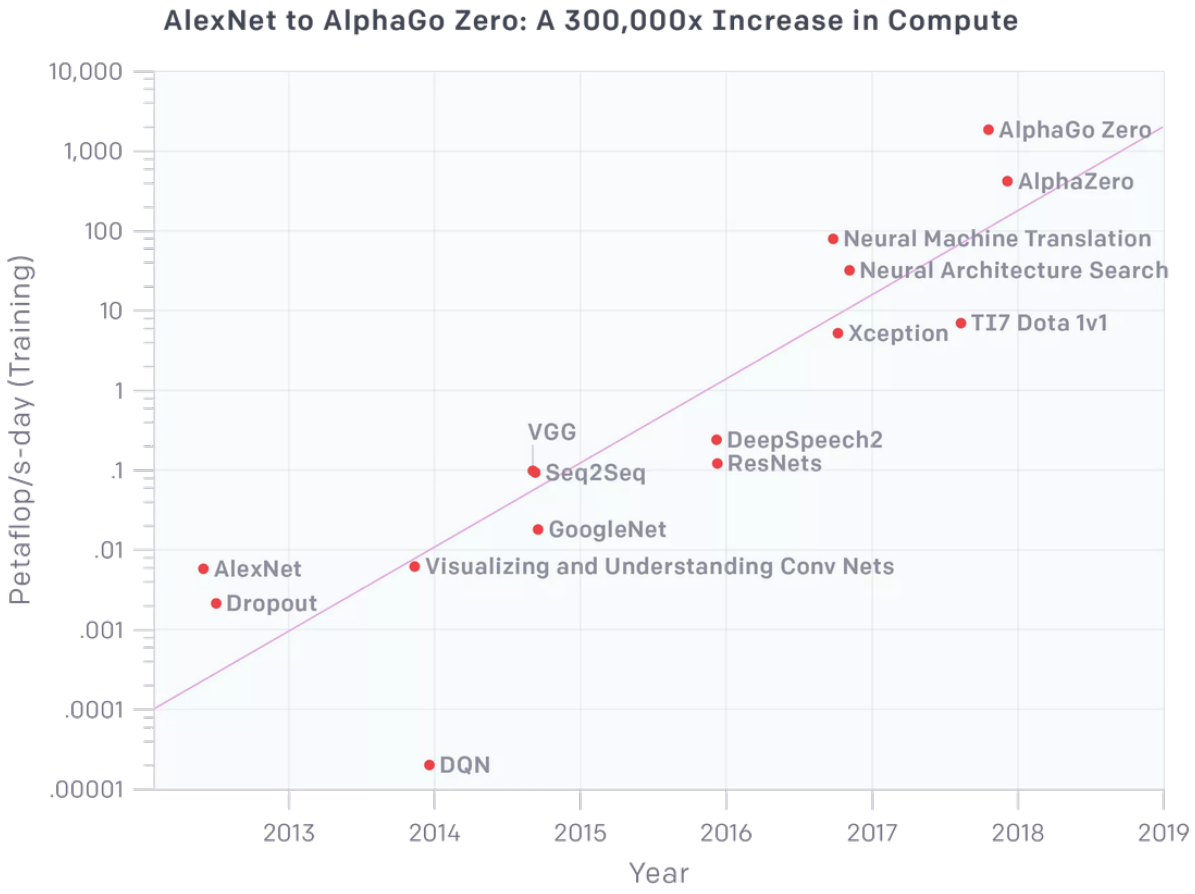
Centralizing observations from users presents several challenges. Bandwidth and storage costs can balloon out of control when trying to store images or large amounts of textual/numerical data. Any data center which contains such sensitive information is a prime target for hackers, but has to sacrifice many security guarantees to be capable of accepting incoming data streams from a dynamic set of consumer-owned devices. Moreover once a hacker is inside the walls of a data center, they have access to all the data stored within.

Given how effective deep learning is, its no surprise that more and more companies are centrally aggregating petabytes of data for training these models. But the exponentially increasing number of data breaches[6] makes this untenable. We need an optimization algorithm that is **provably secure and private**

Communicating weights is also expensive, as model weight files can reach several hundreds of megabytes in size very easily. This discourages the use of traditional optimization algorithms such as SGD (stochastic gradient descent) wherein each client would need to communicate their update to a central server, wait for a server to perform the computation of aggregating these updates towards the final update, and receive the new model before continuing to train. If this latency is incurred for every update of SGD on a deep model

that might need a thousand iterations of optimization, clearly it would be impossible to train the model in a reasonable amount of time. For this reason, we need an optimization algorithm that is **communication-efficient.**

A final challenge inherent to reinforcement learning is scalability. This graph shows that the amount of compute required for state-of-the-art deep learning/reinforcement learning doubles every 3.5 months. On the x-axis we have petaflop/s-days to train, and on the y-axis we see that practically all prior approaches pale in comparison to the computational time of AlphaGo, AlphaGoZero, etc. Therefore we are motivated to produce an optimization algorithm which is **scalable.**



In spite of these challenges, the fact remains that deep reinforcement learning is incredibly accurate and versatile. Our goal in this paper is to introduce an improvement and adaptation onto federated learning for the domain of reinforcement learning, an algorithm which aims to tackle many of these issues. It is predicated on a few other advancements in machine learning.

2 RELATED WORK

We have established that security is a concern. The more times we move massive amounts of data over unencrypted connections, the more likely it is that our data is compromised. Instead of moving data to the model, we can move the model to the data. By minimizing the amount of data that we move, we can minimize the security risk.

The privacy of client data is very important, but ultimately its difficult to make any guarantees on privacy after the data has left the clients device. However, recent advances in on-device machine learning and the ubiquity of AI-specific chips in mobile devices present us with the opportunity to do machine learning client-side. The client has some data -thats why we care about them in the first place- and they use this data to train a model locally.

If we never move data off the device, we are left with a slew of different models on different clients devices, and because each client does not have very much data, none of these models can generalize well -they have just overfitted to the small amount of data they were trained on. We have reduced the security and privacy concerns by bringing deep reinforcement learning to the clients device and doing on-device learning instead of ever moving data off the device, but we no longer have an accurate global model.

Here we use ensemble learning, which has been used to great effect in random forests. In random forests, we take a group of overfitted classifiers and average them to yield a vastly more accurate classifier. By treating the locally-trained models as an ensemble and averaging their parameters, we should be able to create a global model which is more accurate.

2.1 FEDERATED LEARNING

The first major component of our algorithm is federated learning[2]. We train an ensemble of models in parallel across the partition of the network whose devices possess our desired data, before weighting them by a metaheuristic of choice and averaging these experts to arrive at a central model, which we redistribute to all relevant nodes[3]. We repeat this iterative procedure (hereafter a **round**) until any stopping condition such as sufficient convergence is met.

The above settings focus on preserving privacy while centralizing data in a datacenter where a network will be trained. This network can still be trained using distributed machine learning for the sake of efficiency, but we will disregard this semantic distinction and refer to these approaches, or the lack of any privacy-preserving techniques, generally as 'centralized machine learning' or 'centralized private machine learning'. By contrast, the main idea behind federated learning is to preserve privacy by distributing a machine learning model over a network to remote clients, bringing the model to the data rather than the data to the model, and then aggregating these trained models, coming to consensus through a specific weighted averaging[4].

Although ensemble learning has been known to create schizophrenic models which overfit to their small subsample of training data, we present an analytical justification for the resistance of federated learning to this model collapse.

Although most prior work dismisses this aggregation (or more aptly, federation) for being vastly inaccurate, federated learning is able to overcome this accuracy by allowing the federation to come to consensus in batches, similar to round-based voting schemes or minibatch stochastic gradient descent.

Training a machine learning model on a single user dataset specifically is very likely to overfit, as the dataset is non-IID and does not resemble anything close to an unbiased partition of the overarching dataset. In the motivating setting, this overarching dataset would be something like the collective social media interactions of all users in the entire world, and a single dataset would be the Facebook posts of one user of one demographic group in one location. As a result, the network is very likely to learn 'bad traits' by overfitting, such as autocorrecting obscure phrases that only the user says (Google Keyboard), recommending very niche ads (Facebook) or learning to understand a specific accent (Amazon Alexa). However, in learning these 'bad traits' the network will invariably learn underlying 'good traits'. Therefore, each model can be said to have low bias but high variance.

Here we recall two properties of neural networks [1]: for any neural network, the bias can be decreased in exchange for increased variance, and: for any group of networks, the variance can be reduced in exchange

for no increased bias. This motivates the approach of combining experts in ensemble averaging, which has been accepted as a simple committee machine for some time. In ensemble averaging, a set of models is generated with different random initializations. Each of them is then trained separately, creating a set of models which each have low bias but high variance. This can be done by overfitting to training data, or just using an unbalanced set of data, or both. The set of models is then combined, and their parameters -typically just the synaptic weights of the network- are averaged. A weighted average can be used, creating a linear combination of networks, but finding a good heuristic for weighting networks can be difficult.

The original paper *Communication-Efficient Learning of Deep Networks from Decentralized Data*[2] which builds on several other papers by McMahan et. al. [3][4] introduces one such algorithm for ensemble averaging that utilizes a federation of clients, called **FederatedAveraging**. Machine learning using this algorithm is therefore **FederatedLearning**.

3 ALGORITHM

We propose FedRL, a modification to the federated learning algorithm to adjust it for the reinforcement learning setting. We replace the dataset-magnitude weight with the average reward of an agent over that round. The dataset-magnitude weight is not suitable for the reinforcement learning setting as it is not agnostic to all reinforcement learning approaches. Our reasoning is that agents with higher mean reward over a period should have their parameters weighted more heavily in the weighted average.

In each round, each agent pulls the recently updated global model from a model-parameter server. Each agent goes through some number of rollouts in its local environment using different replay buffers, etc. to update a Q-network. It reports the mean reward incurred for this model in the last 'communication round' as well as the model weights to the model-parameter server, which conducts a weighted average.

Algorithm 1: Federated Reinforcement Learning

$A \leftarrow$ Set of agents;
$M \leftarrow$ Initial model;
$a_{\text{model}} \leftarrow M, \forall a \in A$;
while M*has not converged* **do**
  $a_{\text{model}} \leftarrow$ Locally trained model, $\forall a \in A$;
  $r_a \leftarrow$ reward of agent a, $\forall a \in A$;
  $M \leftarrow \sum_{\forall a \in A} r_a \times a_{\text{model}}$;
  $a_{\text{model}} \leftarrow M$
end while
return M;

4 IMPLEMENTATION

We first implemented a federated policy gradient using Tensorflow operations to get and set weights, and limited multiprocessing.

We then implemented federated deep q learning via a base DQN implementation from the Ray library. We run n agents in serial, for t iterations of training per communication round, to a fixed constant number of total iterations. We found it difficult to parallelize due to the unserializable nature of the DQN agents.

We found that resetting the momentum of the ADAM optimizer used to train DQN was necessary, as we were setting the weights to a different point than what their momentum might suggest. We implemented

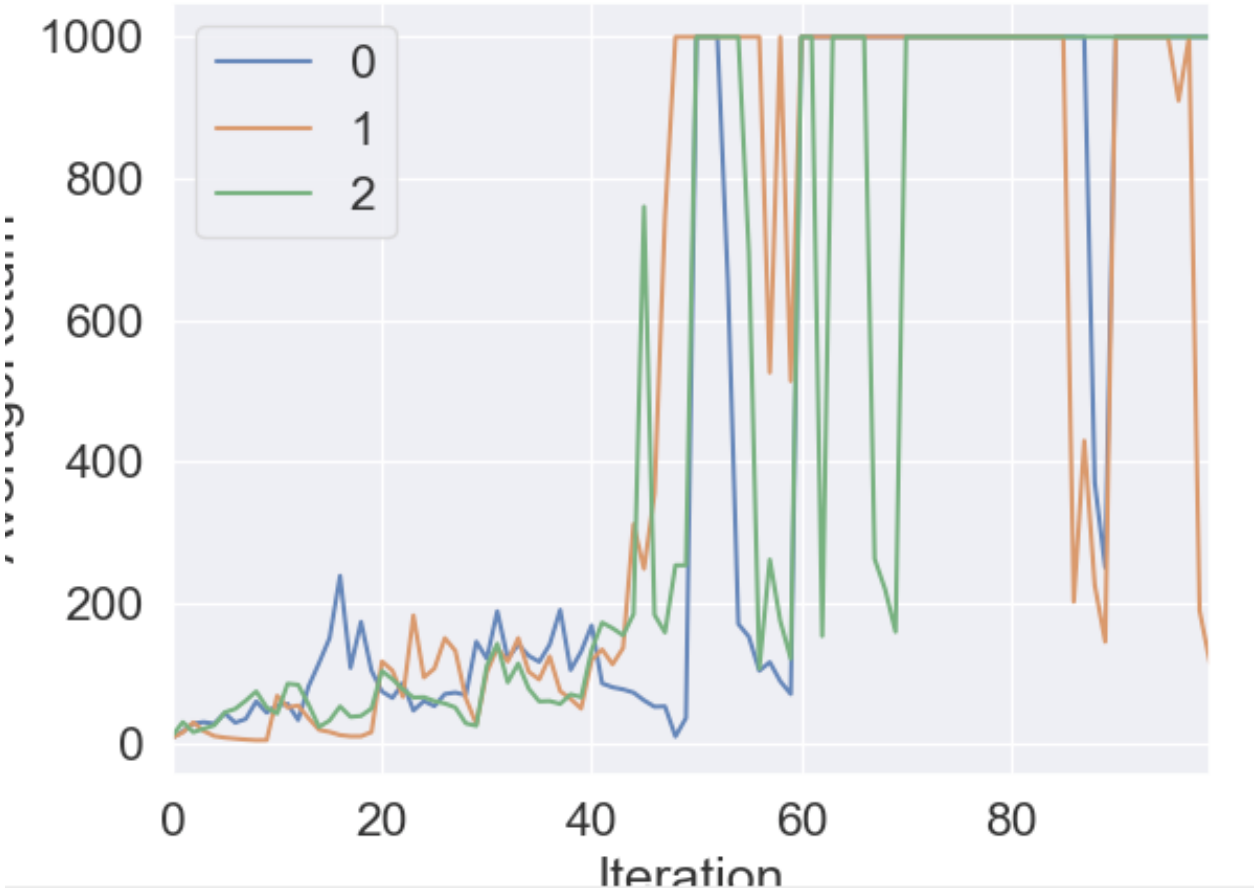


Figure 1: Reward-weighted 5 agents

this with custom a Tensorflow operation to reset momentum.

5 RESULTS

All experiments are done on CartPole-v1 from the standard OpenAI gym library; the specific model configurations can be found in the codebase.

Our first implementation of federated learning was to policy gradient. We conducted experiments with CartPole to determine which weighting scheme would serve better for the rest of our experiments: "Reward" (Figure 1, we do a weighted average by reward), "Independent" (Figure 2, no weighting, no averaging), "Max" (Figure 3, all agents copy the weights of the agent that had the max reward for the previous round). As shown below, reward-weighting performed significantly better than the independent baseline and better than the max-weighting scheme.

We next turn to deep-Q learning for the rest of our experiments. First we do a similar baseline comparison between reward-weighting (Figure 4) and an ensemble of agents who do not communicate at all (Figure 5). We observe that with such a low communication round, the agents converge extremely quickly to the 160-200 reward range but fluctuate wildly because they continue to communicate often even after they have converged to near-optimal reward.

We compare our reward-weighting modification to federated learning to a 'naive' implementation where we

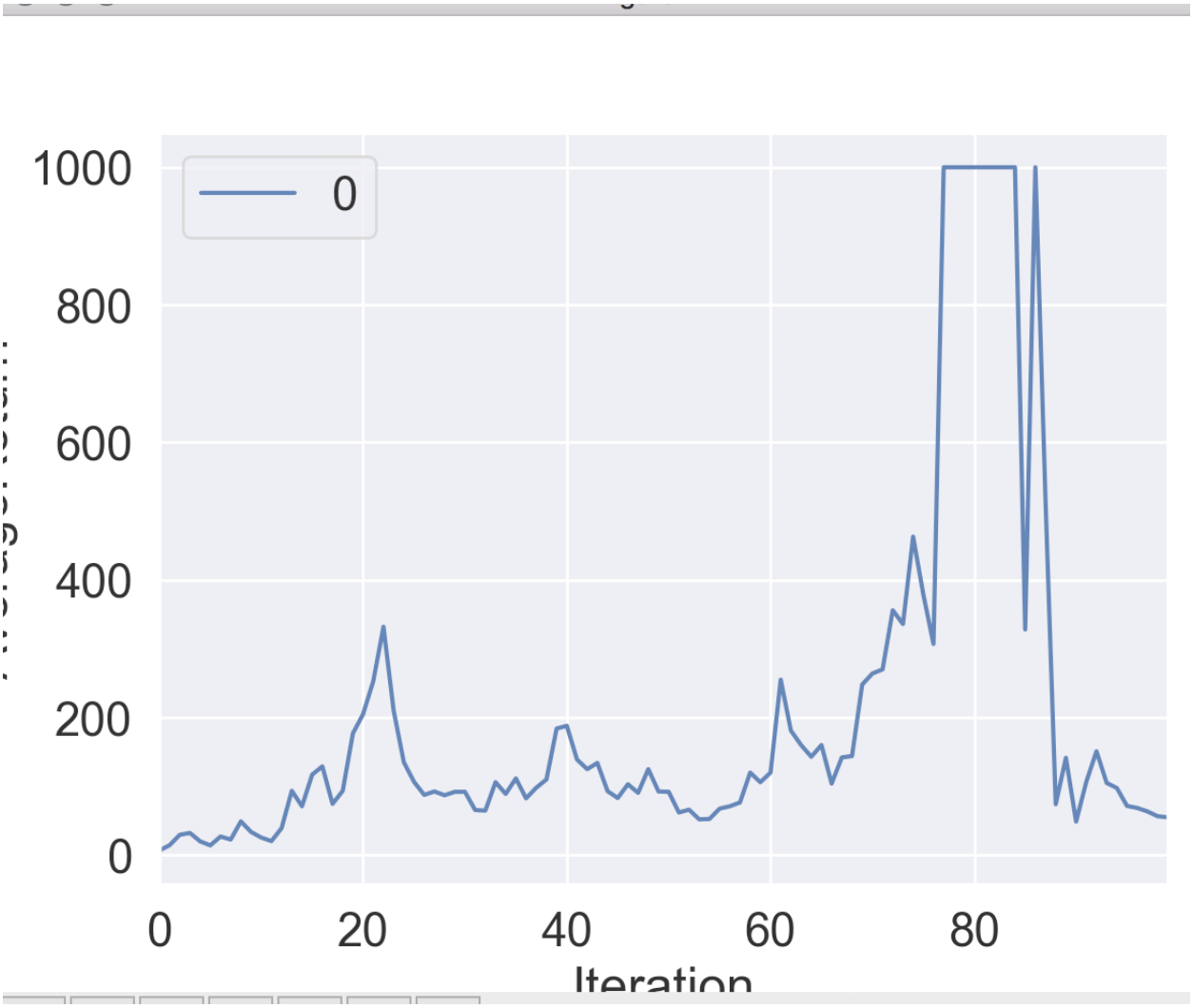


Figure 2: Independent comparison

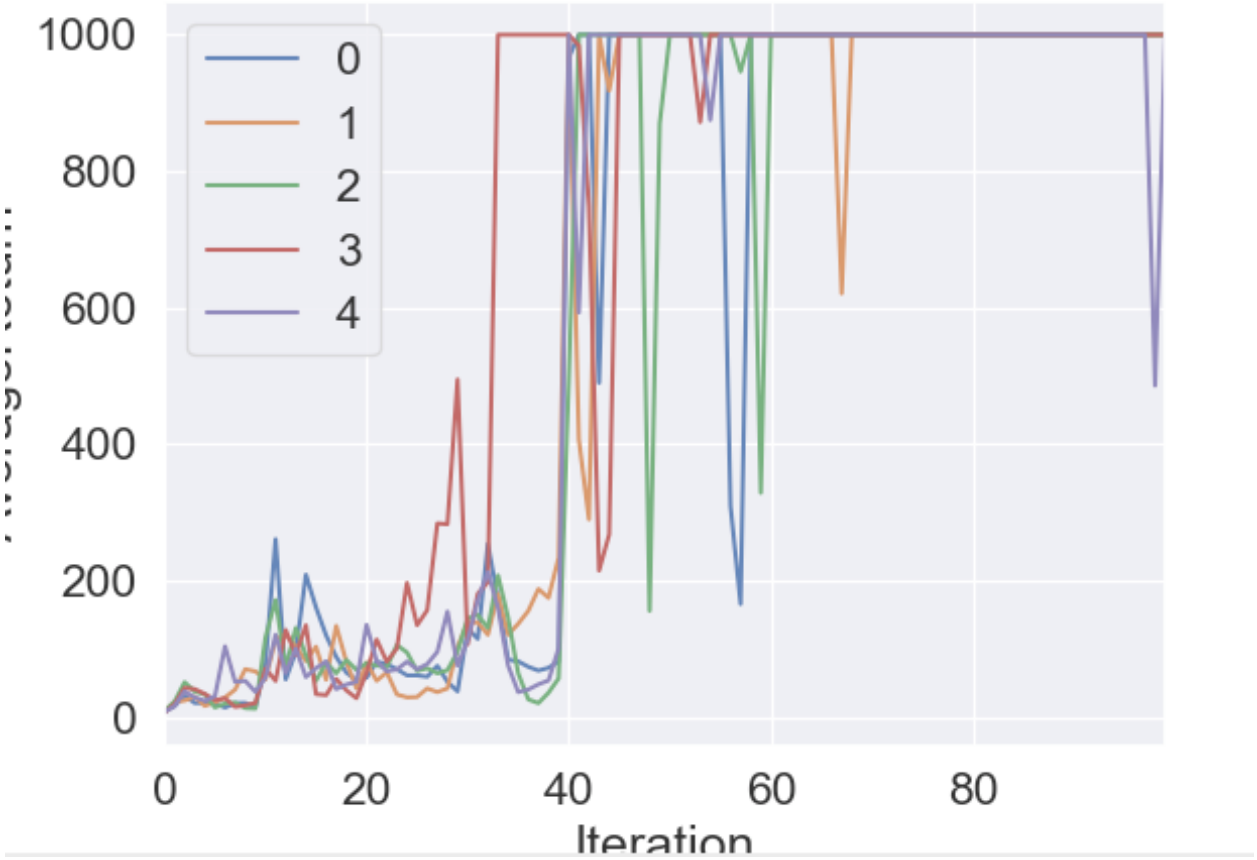


Figure 3: Max-weighted 5 agents

Figure 4: Reward-weighted with c=0.01, 10 agents

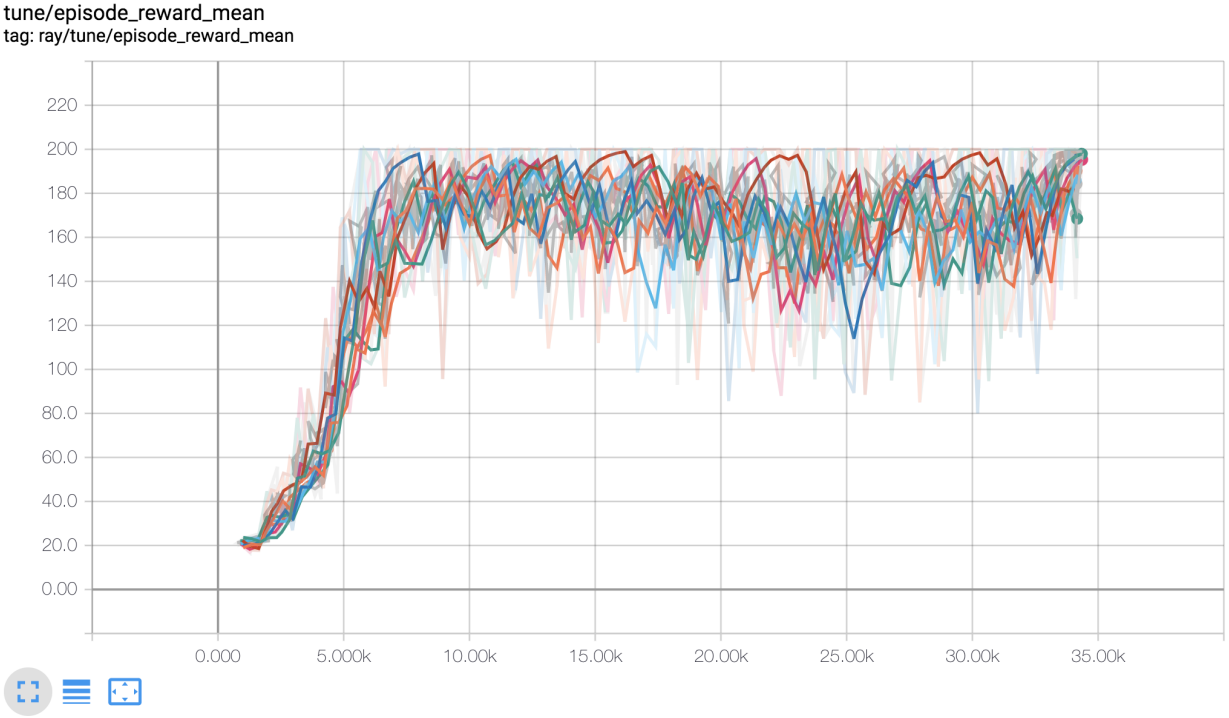
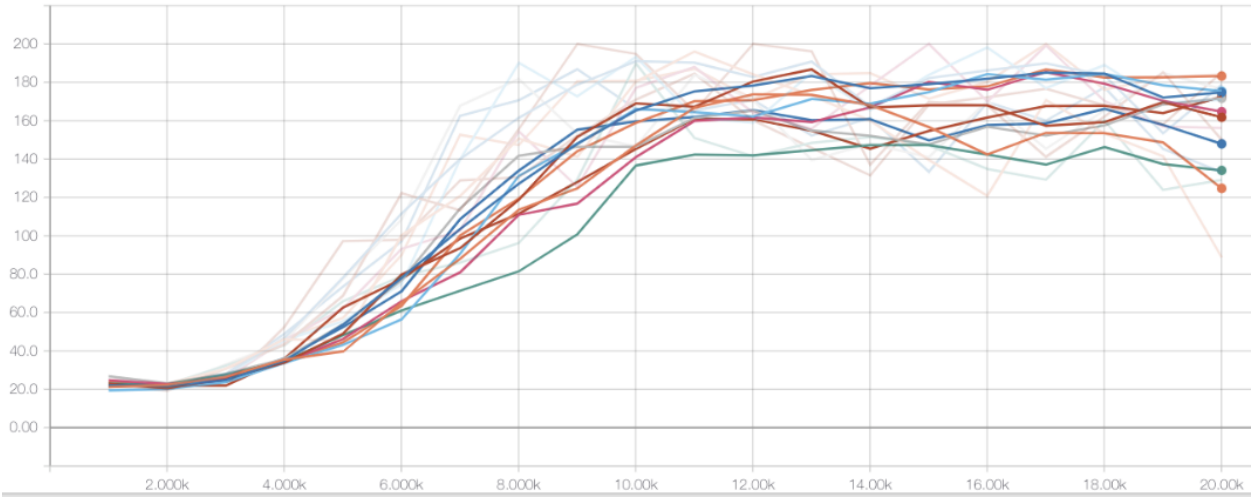


Figure 5: Independent baseline, 10 agents



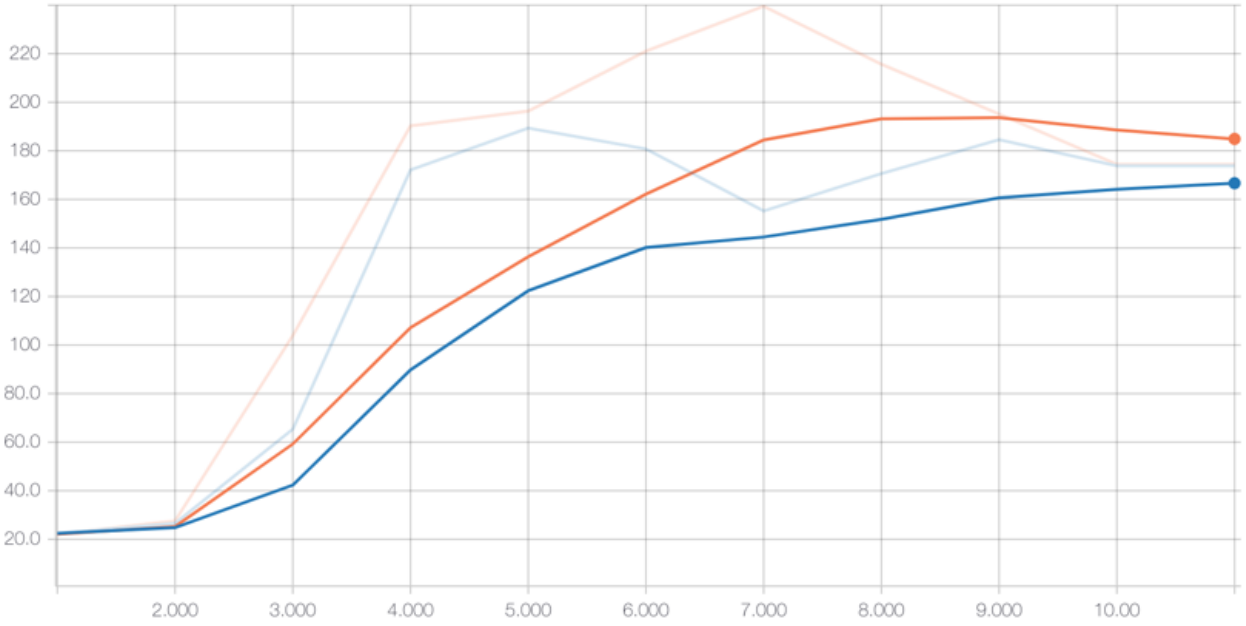
only implement dataset magnitude weighting (which in the case of deep-q is uniform averaging) and find that reward weighting performs rather well (Figure 6).

We then perform hyperparameter search (Figure 7) over c which is the proportion of total iterations allotted to each communication round. Larger values of c mean that each agent performs more iterations between averaging, so there is more opportunity for divergence but similarly more opportunity for each agent to discover better local minima.

The best value of c we found was $c = 0.02$ which performed significantly better than a baseline in a larger-scale test with 100 agents (Figure 8).

Many more results can be found in the Github repository.

Figure 6: Reward-weighted (red) vs dataset-magnitude (blue) with c=0.1, 10 agents



6 CONCLUSION AND FUTURE OBJECTIVES

6.1 Security and Privacy

Under naive views of security and privacy, FedRL seems to accomplish the goals we set out for ourselves. If deep reinforcement learning networks were trained on consumer devices, the potentially private/valuable observations would never need to be transferred off the device in order to contribute to the collaborative training of a global model.

In future it would be valuable to see how we can account for differentially private reinforcement learning and how this influences the communication period frequency. We might implement this with a moments accountant[9].

6.2 Scalability

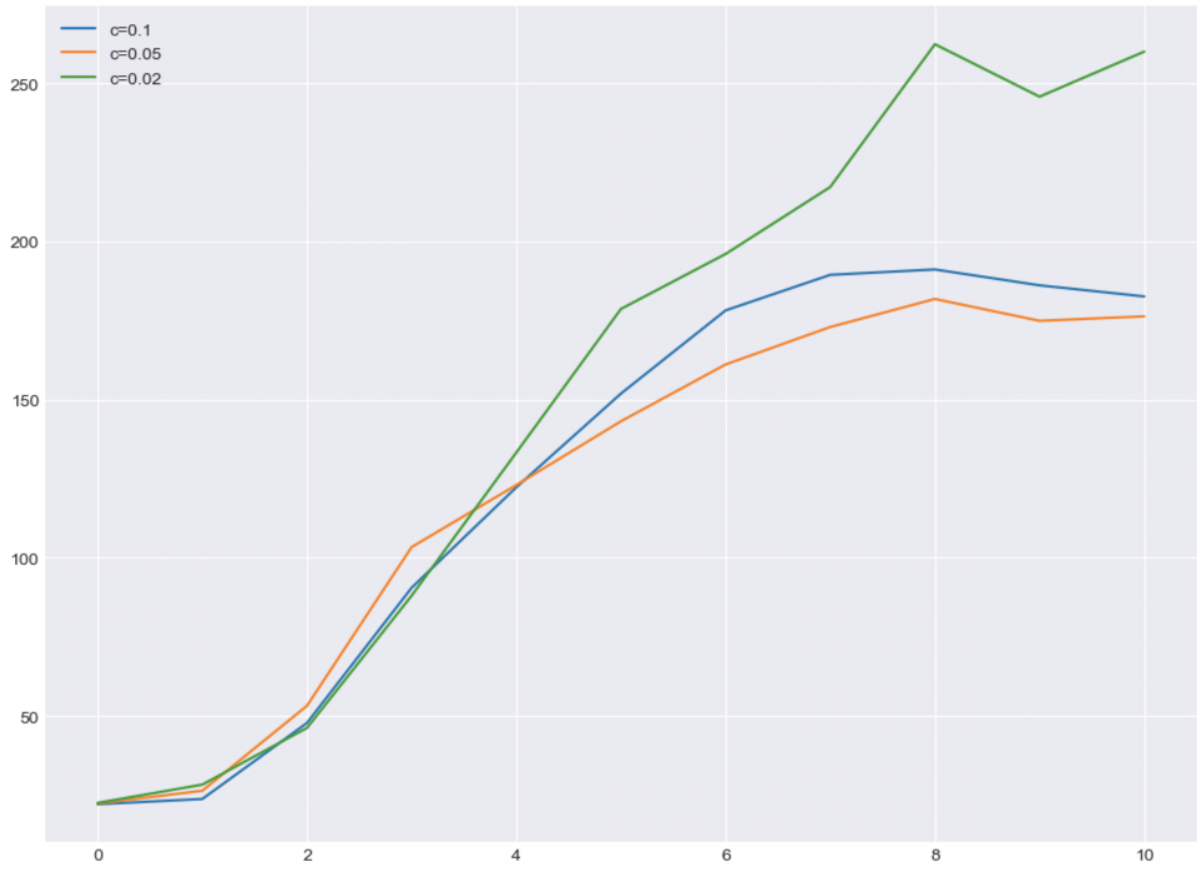
We wanted to benchmark FedRL not only against centralized training or simple distributed training, but more sophisticated approaches such as A3C or IMPALA. However we did not find other distributed reinforcement learning algorithms which set the strict requirement that we imposed on FedRL. That is, of not exchanging observations of their environments.

6.3 Communication-Efficiency

We find that federated reinforcement learning with a fixed communication round hyperparameter converges much faster than single-agent training, but to worse reward. Even for the best choice of the communication round hyperparameter c, past 15000-20000 iterations single-agent training catches up to and overtakes the federated model. In future, we would like to experiment with a decaying communication round.

We could begin with a large period for communication, so that agents have more time to learn independently of other agents, and then decay the period to converge to a lower error floor by ensuring that the agents learn from each other later in the lifetime of their training. This adaptive communication strategy is

Figure 7: Different values of c



dubbed ADAComm[8] and was originally designed for federated supervised learning.

Although we had planned to take this approach, the results that we saw using our reward-weighting quickly contradicted the thesis of ADAComm: that more frequent communication would lead to slower convergence to a lower error floor. Indeed, we find that more frequent communication leads to quicker convergence to lower reward -the opposite result observed in hyperparameter tuning of the communication period in federated supervised learning.

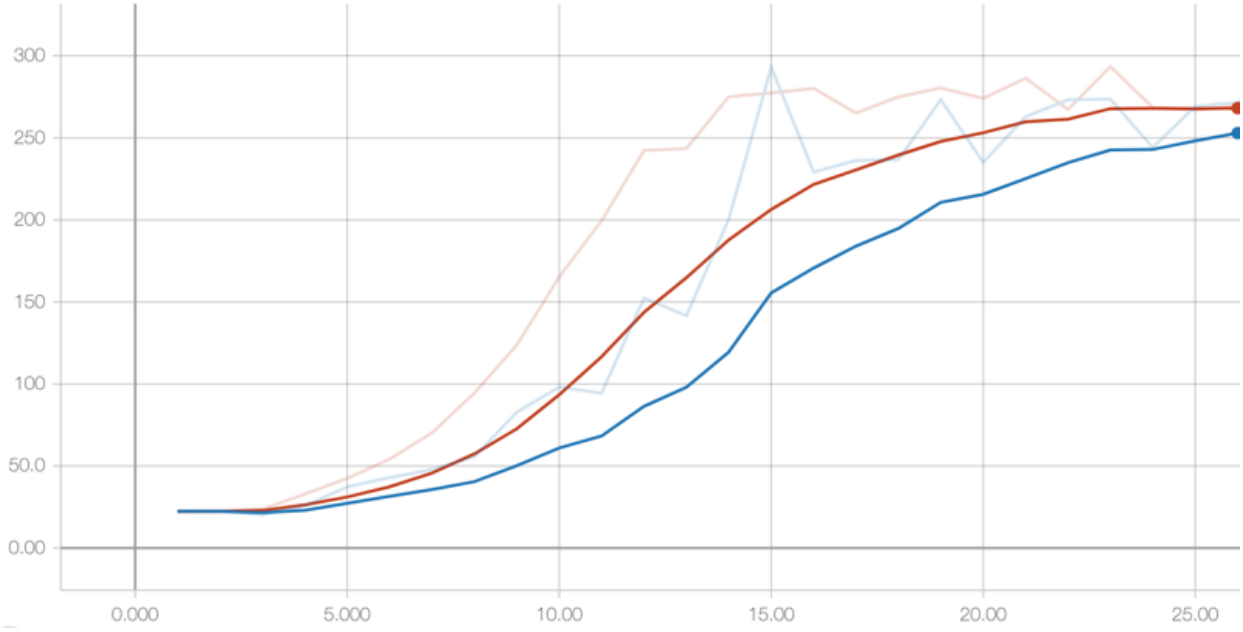
Rather, we think we will instead adopt an 'increasing' communication period strategy, where we begin with frequent communication to guide agents towards locally good strategies and then increase the length between averaging to allow agents to explore and find higher-reward policies on their own.

Currently, the choice of fixed communication period length does not impact the performance nearly as much as the number of agents, the weighting strategy, etc.

References

[1] Naftaly, U., N. Intrator, and D. Horn. "Optimal ensemble averaging of neural networks." Network: Computation in Neural Systems 8, no. 3 (1997): 283296.

Figure 8: 100 agents federated training c=0.02 (red), average of 10 individual agents (blue)



[2] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson and Blaise Agera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data, 2016, Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017. JMLR: W&CP volume 54; arXiv:1602.05629.

[3] b Konen, H. Brendan McMahan, Felix X. Yu, Peter Richtrik, Ananda Theertha Suresh and Dave Bacon. Federated Learning: Strategies for Improving Communication Efficiency, 2016; arXiv:1610.05492.

[4] "Distributed Mean Estimation with Limited ... - Research at Google." `https://research.google.com/pubs/pub45672.html`. Accessed 19 Mar. 2018.

[5] "Ray - Distributed Execution Environment" `https://github.com/ray-project/ray`. Accessed 19 Mar. 2018

[6] "2018 Verizon Data Breach Investigations Report" `https://verizonenterprise.com/verizon-insights-lab/dbir/`. Accessed 12 Mar. 2018

[7] Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan and Xiaohui Ye. Horizon: Facebook's Open Source Applied Reinforcement Learning Platform, 2018; arXiv:1811.00260.

[8] Jianyu Wang and Gauri Joshi. Adaptive Communication Strategies to Achieve the Best Error-Runtime Trade-off in Local-Update SGD, 2018; arXiv:1810.08313.

[9] Martn Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar and Li Zhang. Deep Learning with Differential Privacy, 2016; arXiv:1607.00133. DOI: 10.1145/2976749.2978318.