IFT6163 Robot Learning Project

Razvan Ciuca and Paloma Fernandez-McAuley

November 29, 2023

Abstract

Provide an report of your course project. You should reuse a lot of content from your project proposals and add more onto them. You can find an overleaf template for this assignment here. Overall, the project report should be **NO MORE THAN 8** (not including references) pages with at least 3 figures/tables. Include lots of information to show your progress with the project. You grade depends more on what you show you learned and less on method performance.

1 [4 pts] Project Introduction (2 paragraphs, with a figure)

Loosely inspired by the research into adversarial examples for image recognition models, we introduce the idea of "gradient compression" for policy gradient models, which consists in finding single state-action sequences which lead to gradients which closely match the full parameter change that occurs over the entire training history of a policy gradient model. Finding such action sequence that produces a gradient with high cosine similarity to the full weight change would let us fully train a model by computing the gradient of a single trajectory, and doing line search in that direction. The feasibility of this approach has interesting implications for the internal structure of policy gradient methods. If gradients turn out to be very compressible, it suggests the existence of a sampling method which might learn to generate these "gradient-compressed trajectories" from scratch, thereby greatly increasing the sample efficiency of policy gradient methods.

On a set of mujoco environments using a quadratic agent and a small neural network agent, we explore just how much the policy gradient and the entire weight change can be compressed into single sequences of actions. The effect of model architecture on this compressibility, how much random weight directions can be compressed in this way, the relationship between the maximum achievable cosine similarity and the length of trajectory we optimise, and finally we "decompress" the compressed gradients we found to get back an approximation of the optimal policy.

2 [2 pts] What is the related work? Provide references: (1-2 paragraph(s))

This work expands first and foremost on the policy gradient class of RL algorithms Williams [1992]. The inspiration for optimizing the inputs of a network in order optimize a weight-dependent function comes from adversarial attacks in image models, first shown by Szegedy et. al Szegedy et al. [2013]. We use the cross-entropy method to optimize action sequences Rubinstein and Kroese [2004], and use the mujoco physics engine Todorov et al. [2012] implemented in the OpenAI gym environment Brockman et al. [2016]. To find maximal Cosine Similarity vectors in the span of single state-action gradients, we use the L-BFGS algorithm Byrd et al. [1995] bundled in the pytorch optim module Paszke et al. [2019].

3 [2 pts] What background/math framework have you used? Provide references: (2-3 paragraph(s) + some math)

The main theoretical background of this work is the policy gradient theorem, which states that the gradient of the sum of future reward $v_\pi(s_0)$ with respect to the free parameters of our policy $\pi_\theta(a|s)$ is given by the following expression, first expressed as an expectation value, and then expressed as a finite sample approximation.

$$\nabla_\theta v_{\pi_\theta}(s_0) = E_{\tau\sim\pi_\theta}\left[G\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{1}$$

$$G \equiv \sum_t r_t \tag{2}$$

$$\nabla_\theta v_{\pi_\theta}(s_0) \approx \frac{1}{N}\sum_i \left(G_i \sum_t \nabla_\theta \log \pi_\theta(a_{t,i}|s_{t,i})\right) \tag{3}$$

This result assumes that we are sampling state trajectories according to the current version of the policy π_θ, if we are sampling according to a different policy, we need to use importance sampling, which allows us to compute expectations of a quantity x with distribution $p(x)$ from samples from $q \neq p$: $E_{x\sim p}[x] = E_{x\sim q}\left[\frac{p(x)}{q(x)}x\right]$. Written for the policy gradient theorem, this becomes:

$$\nabla_\theta v_{\pi_\theta}(s_0) = E_{\tau\sim q}\left[\frac{p(\tau)}{q(\tau)}G\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{4}$$

$$\nabla_\theta v_{\pi_\theta}(s_0) = E_{\tau\sim\pi'}\left[\frac{\prod_t \pi_\theta(a_t|s_t)}{\prod_t \pi'(a_t|s_t)}G\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{5}$$

$$\tag{6}$$

The last bit of theory we need is the Cross-Entropy Method for optimizing a function $f(x)$ for which we do not have access to explicit derivatives $\nabla f(x)$. For us x will be a candidate sequence of actions $a_1, a_2, ..., a_t$. CEM begins by generating an initial population of solutions $x_i \sim N(0, \sigma_0^2)$ with a σ large enough that we are confident the optimal solution will be covered with appreciable probability. We evaluate each point x_t to obtain $y_i = f(x_i)$. We then keep the top r points, fit a new gaussian distribution to these points, and sample again to produce the next iteration of points. We note that since the variance can only decrease with successive iterations, it's important to set the initial variance large enough. A easy improvement to this basic method involves fitting a linear function to $\{x_i, y_i\}$, which gives us an estimate of the gradient, and then updating the mean of our gaussian distribution at each iteration in the direction of this computed gradient.

4 [6 pts] Project Method, How will the method work (1-2 pages + figure(s) + math + algorithm)

4.1 Finding Gradient-Compressed Trajectories

In this section, we will define what we mean by "gradient-compressed trajectories" and justify why such a concept is useful and interesting. Given a parametrised policy π_θ and a set of trajectories $\tau_i \sim \pi_\theta$, we can compute the gradient g_i for each of these trajectories and therefore approximate the true policy gradient by $\frac{1}{n}\sum_i g_i$ only in the limit $n \to \infty$. We notice that the policy gradient theorem provides a procedure for turning numbers representing sequences of actions $A = \{a_0, ..., a_t\}$ into numbers representing gradients with respect to θ. Call this procedure $g(A)$ and let $\nabla_\theta v_\theta(s_0)$ be the full policy gradient.

We can now view the sequence of actions A as our optimization target and attempt to find a single sequence A such that $||g(A) - \nabla_\theta v_\theta(s_0)||^2$ is minimized. The full policy gradient requires a great deal of environment interaction to compute and, by finding a single action sequence which allows us to compute a gradient as close as possible to the true one, we have effectively "compressed" all the useable information of a large sample of trajectories into a single one. This trajectory "punches above its weight", so to speak, in terms of how useful it is for learning the optimal policy. Furthermore, we do not need to limit ourselves to optimizing a single trajectory: we can view a set of trajectories as our optimization variable instead to better approximate the full gradient.

There are three extensions of this basic idea that we can explore. First, we are not limited to compressing the policy gradient: we can in principle try to find action trajectories which lead to gradients close to an arbitrary direction in space, hence we can try to "compress" a completely random direction in parameter space, or directly compress the full parameter change from the beginning of training to the end of training, this latter extension is what we do in this project. Second, finding trajectories which directly minimize the distance $||g_i - \nabla_\theta v_\pi(s_0)||^2$ might be too hard, since the compressed gradient not only has to match the direction of the full target vector, but also its magnitude. Maximizing the cosine similarity instead of the squared L2 norm fixes this problem. We discuss the third extension in

the next section.

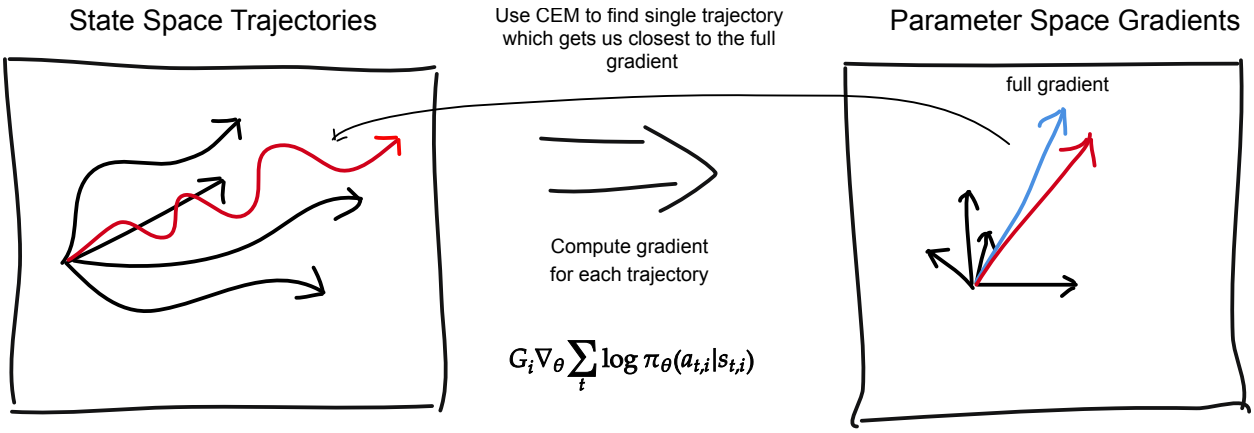

Figure 1: Visualization of gradient compression, going from state-space trajectories to weight space gradients. The blue arrow is the full policy gradient computed from a large sample of trajectories. The red trajectory is our optimized action sequence resulting in a gradient as close as possible to the blue arrow.

4.2 Finding Maximal Cosine Similarity Vectors in the Span of Single State-Action Gradients

Instead of simply summing over the log likelihoods $\sum_t \nabla_\theta \log \pi_\theta(a_{t,i}|s_{t,i})$ of each action in the trajectory that we are considering in order to compute the gradient that we wish to be close to our target compression vector, we can also learn a weighing $\sum_t \alpha_t \nabla_\theta \log \pi_\theta(a_{t,i}|s_{t,i})$ for each action. Roughly, instead of finding a trajectory that should be uniformly encouraged in order to lead to the vector we want to reconstruct, we allow for differences between the actions considered. More precisely, if $g_t \equiv \nabla_\theta \log \pi_\theta(a_t|s_t)$ is the gradient for the state-action pair (s_t, a_t), we wish to find coefficients α_t such that $D_{\text{cosine}}(w_{\text{target}}, \sum_t \alpha_t \nabla_\theta \log \pi_\theta(a_t|s_t))$ is maximized. Finding these parameters α_t is a nontrivial optimisation problem and we use the BFGS optimization algorithm to solve it. See figure 2 and algorithm 2 for a visualisation and precise description.

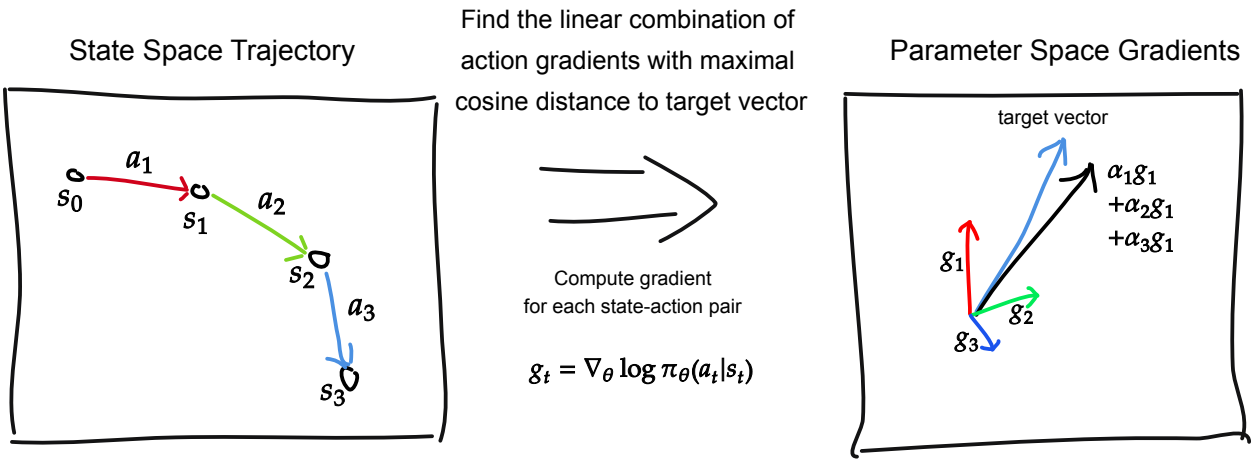

Figure 2: Visualization of algorithm 2. For a fixed state-action trajectory, we can find the optimal weighing α_i for each action in the sequence in order to maximize the similarity with a target vector, shown here in blue. The red, green and blue arrows g_i correspond to individual transition gradients which are then weighed over to produce the black arrow. Note that in two dimensions any two independent vectors would be enough to fully reproduce any target vector, but this situation is very different in high dimensional spaces.

5 [2 pts] What new skills have you(s) learned from this project? (2 paragraphs)

We learned to implement policy gradients on our own from scratch, as well as how to how to make them work on Mujoco. We learned the problems associated with importance sampling in policy gradient methods, and managing importance sampling ratio blow-up and squashing. We learned to try simple things before complicated things, linear functions before quadratic functions, and those before neural networks, and shallow nets before deep nets. We were very surprised to find that neural nets are not needed at all on mujoco tasks (when not using vision inputs), a quadratic regression model actually performs better.

We also learned project management skills, and to temper expectations about how many features and experiments it's possible to implement in a given time. We learned good experimental design, explicitly writing our expectations about the result of an experiment before the experiment is run, so surprising results are more legible and easier to update upon.

6 [8 pts] Experiments and Analysis

6.1 Experiment 1: Using Algorithm 2 on Ant-V4, compressing the entire weight change across learning

In the first experiment, we train a quadratic model on the Ant-V4 gym environment, obtaining a good final policy (see videos), then we run Algorithm 2 above on the entire weight change $w_{\text{final}} - w_{\text{initial}}$, finding a sequence of actions which produces a gradient with high cosine similarity with this weight change vector.

Algorithm 1 Finding compressed-gradient action sequences

init network parameters θ parametrising policy π_θ, and known gradient $\nabla_\theta v_\pi(s_0)$
init $\mu = 0$ and σ to large value
for $l \in 0, \ldots, L$ **do**
  Sample population of action trajectories $A_i = \{a_{i,0}, a_{i,1}, ..., a_{i,t}\}$ with $a_{i,k} \sim N(\mu, \sigma^2)$
  run A_i in environment to collect trajectories τ_i
  compute the gradients $g_i = G_i \sum_t \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})$
  compute $y_i = ||g_i - \nabla_\theta v_\pi(s_0)||^2$
  find the indices of the top k y_i values
  update μ and σ with the mean and variance of the top k sequences
end for
return g_i with minimal $||g_i - \nabla_\theta v_\pi(s_0)||^2$

Our predictions Before Running The Experiment Will trajectory compression actually find a trajectory with large cosine similarity? YES, we think that trajectory compression will successfully find a sequence of actions with high cosine similarity. We have a lot of free parameters to optimise here, a full trajectory is 1000 actions, which is a lot of room to exert optimisation pressure. Regarding our expectations of what the optimal trajectory will look like, we expect that the way to make the gradient be in the direction of the final weights will be to have the trajectory basically exhibit all the most common mistakes and the way to correct them, thereby producing a trajectory that looks quite weird, and not necessarily be of high return

Experiment Results We achieve a cosine similarity of 0.862 with the full training weight vector for optimising a trajectory of length 500, which is quite a bit higher than we expected. The video of the final policy looks very weird to us, with no discernable walking behavior, but more like a baby stretching its arms and exploring the state space. As figure 3 shows, there's a also a smooth scaling of the cosine similarity and the optimized trajectory, which makes obvious sense: having more optimisation freedom should give a large final similarity.

6.2 Experiment 2: Algorithm 2 on Ant-V4, compressing random weight directions instead

Our predictions Before Running The Experiment Will trajectory compression be able to compress arbitrary weight directions as well as compressing gradients? We don't think so. The weight directions we're trying to reproduce were produced by gradient descent on trajectories from the same environment, the gradients from the environment shouldn't be able to reproduce an arbitrary weight direction.

Experiment Results The maximum cosine similarity with a random weight vector was 0.384 when training a quadratic model on Ant-V4, which is higher than we expected, but much lower than the one achieved for the entire weight difference. So random vectors are in fact harder to compress than vectors generated from the learning process. The green curves

Algorithm 2 Compressed-gradient trajectory for arbitrary weight direction

init model parameters θ parametrising policy π_θ, and target weights to reconstruct w_{target}
init $\mu = 0$ and σ to large value
for $l \in 0, \ldots, L$ **do**
  Sample population of action trajectories $A_i = \{a_{i,0}, a_{i,1}, ..., a_{i,t}\}$ with $a_{i,k} \sim N(\mu, \sigma^2)$
  run A_i in environment to collect trajectories τ_i
  compute the gradients $g_{i,t} = \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})$
  for $i \in 0, \ldots, n$ **do**
    init parameters $\alpha_t = 1$
    Maximise $D_{\text{cosine}}(w_{\text{target}}, \sum_t \alpha_t \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}))$ with respect to α_t using L-BFGS.
  end for
  compute $d_{i,\text{best}}$ the best cosine similarity found in the previous step
  find the indices of the top k $D_{i,\text{best}}$ values
  update μ and σ with the mean and variance of the top k sequences
end for
return g_i and A_i with maximal cosine similarity with w_{target}

in figure 3 show the growth the similarity as a function of trajectory length is much slower than the growth for the weight vectors coming from learning agents.

6.3 Experiment 3: Algorithm 1 on Ant-V4, compressing the entire weight change across learning

In this experiment we ran Algorithm 1 on Ant-V4, both with a quadratic agent and a neural network agent, aiming to find trajectories which lead to gradients in the same direction as the entire weight change across learning.

Our predictions Before Running The Experiment We thought that Algorithm 1 would succeed in finding trajectories of high similarity, albeit not as well as algorithm 2.

Experiment Results Algorithm 1 just doesn't work very well. Figure 3 shows the cosine similarity barely improve with the length of the optimized trajectory. The requirement that all state-action pairs contribute an equal amount to the gradient vector seems to make it impossible to control the environment in a way which will lead to that vector being close to the total training weight difference. This experiment confirms that the improvements of Algorithm 2 are crucial.

6.4 Experiment 4: Line search in the direction of the compressed gradient on Ant-V4

In this experiment we take our compressed gradient trajectory that we found in experiment 1, compute its gradient on the untrained quadratic model, weighing each state-action gradient

by the parameters α_t we found, then we do a line search in that direction, finding the model with maximal average return in that direction. The purpose is to see if doing a line search in a direction with high, but not perfect, cosine similarity with the total weight change will yield good final behavior.

Our predictions Before Running The Experiment We expect the policy learned from the reconstructed gradient to not be as good as the original policy, but still somewhat ok.

Experiment Results While the average return of the best policy in the line search was nowhere close to the return of the optimal policy (return 230 vs 1300 for trajectories of length 1000), the behavior itself was impressively good. The ant convincingly learned to walk (though slowly, and in a circle) from a single trajectory which did not contain any walking behavior.

7 [2 pts] Video Results

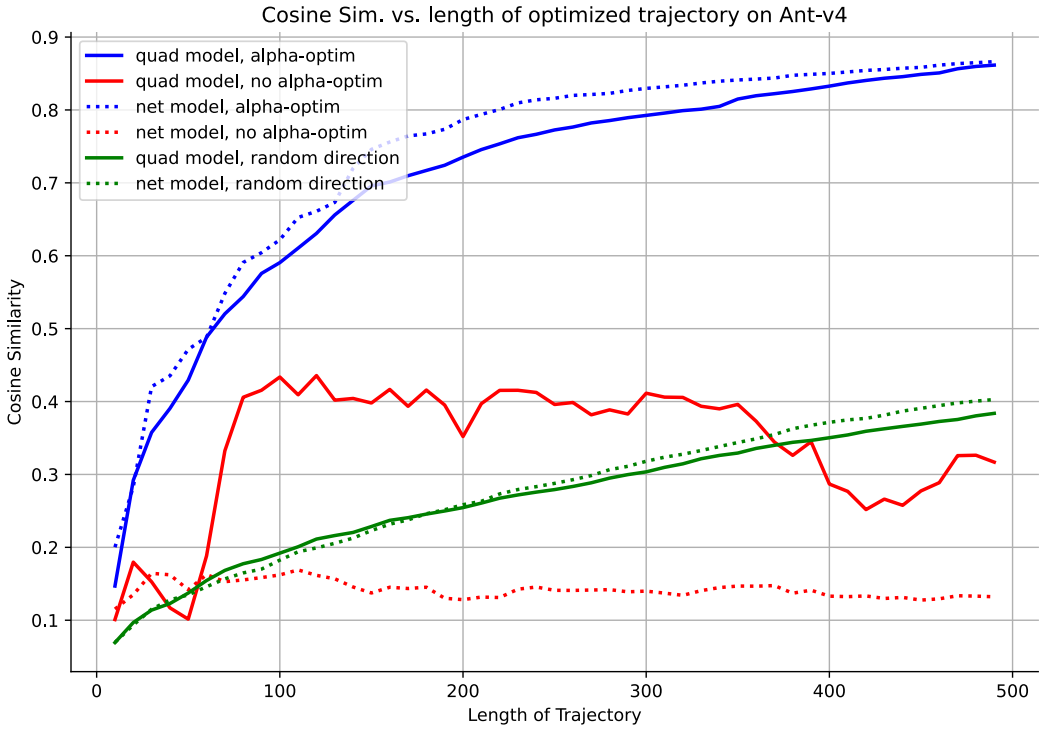

Figure 3: test

(videos includes in submission)

8 [4 pts] Conclusions

Our results indicated that trajectory compression using algorithm 2 finds a trajectory with a large cosine similarity. We confirmed that random vectors are harder to compress than ones generated from the learning process. Algorithm 2 works works much better than algorithm 1. We thought there would be a slight improvement, but only algorithm 2 succeeds in finding trajectories of high similarity. Finally, we showed that the ant convincingly learned to walk (though slowly, and in a circle) from a single trajectory which did not contain any walking behavior

Next time, it would be better to have a more realistic idea of what can accomplished in a limited amount of time. This would have allowed us to put more work into the main idea, instead of having to put aside certain parts of the project that would not be finished in time. The original scope of the project ended up being too large. We ended up focusing on the gradient compression which was a good sized project and leaves other experiments for future work.

9 [2 pts] How was the work divided?

Provide a description on what each group member is working on as part of the project. I recommend each student work on most of the parts of the project so everyone learns about the content.

Student Name: Razvan Ciuca Wrote the initial version of policy gradients used for training. Developed the algorithm ideas. Wrote the quadratic learner pytorch model and implemented the algorithm that finds the optimal linear combination of vectors to maximize cosine sim. with another vector.

Student Name: Paloma Fernandez Modified the policy gradients code to fix bugs. Wrote part of the experiments code and plotting code. Implemented CEM and another progressive random-trial hill-climbing algorithm.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. In *arXiv preprint*, 2016. URL https://arxiv.org/abs/1606.01540.

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. doi: 10.1137/0916069. URL https://doi.org/10.1137/0916069.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Reuven Y. Rubinstein and Dirk P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, New York, 2004.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *arXiv preprint*, 2013. URL https://arxiv.org/abs/1312.6199.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn*, 8:229–256, 1992. doi: 10.1007/BF00992696.