Deep Z-Learning

Nathan Bittner

Advisor
Paul Schrater
Readers
Maria Gini & Hyun Soo Park

Submitted under the supervision of Paul Schrater to the University Honors Program at the University of Minnesota-Twin Cities in partial fulfillment of the requirements for the degree of Bachelor of Science, *Summa Cum Laude* in Computer Science

May 12, 2018

Abstract

In this thesis, I present advancements in the theory of Z-learning. In particular, I explicitly define a complete tabular Z-learning algorithm, I provide a number of pragmatic qualifications on how Z-learning should be applied to different problem domains, and I extend Z-learning to non-tabular discrete domains by introducing deep network function-approximation versions of Z-learning that is similar to deep Q-learning [2].

Contents

1 Introduction

In this thesis, I develop and extend the theory of Z-learning. Z-learning was first proposed by Emo Todorov as an application of his LMDP[7] framework. Z-learning is a temporal-difference (TD) off-policy online algorithm that approximates the value function of an MDP and has been shown to out-preform other algorithms in this category (e.g., Q-learning) with respect to learning the value-function as quickly as possible.

Despite this, since its introduction in 2007, there has been little attention given to Z-learning. In this work, I am interested in extending the theory of Z learning to not only flesh out many of the details left open by Todorov with respect to how Z-learning can be applied in tabular domains, but to also extend Z-learning to the deep neural network setting, and thereby expand the applicability of Z-learning to non-tabular domains through the use of function approximation.

In addition to the number of theoretical contributions I make in this work, I also intended to evaluate and benchmark the deep version of Z-learning against the famous deep Q-learning algorithm [2]. For a number of reasons that are discussed later, this empirical evaluation failed as a result of intractable computation times and divergence of the algorithm in the test environments.

Thus, the contributions of this work are, as of now, primarily theoretical, and the algorithms derived are the central contribution. The experimental limitations were, ultimately, a factor of time, and later publications on this topic will deal with the journey of implementating the theory defined here.

The rest of the paper will proceed as follows: In section 2 I provide a review of Z-learning as it has been discussed in the field until now. Following this, in section 3, I make my contributions to the theory of Z-learning. In section 4 I briefly describe the challenges associated with evaluating the algorithm as I have defined it.

2 Relevant Background

Appendices A and B provide background on Markov decision-processes (MDPs) and linearly-solvable MDPs (LMDPs), both of which are requisite material for my present work. The uninitiated reader should review these sections. appendix C Provides a quick review of deep Q-networks [2], which will be important background for the later sections on deep Z-learning.

The remainder of this section provides a review of the concept of Z-learning, to the extent that it has currently been discussed in the literature.

2.1 Z-Learning

As an application of the LMDP framework, Todorov defined **Z-learning**, an online, TD approximation of the desirability function, Z, that makes use of samples in the form of (s, r, s'), recorded from interactions with the environment. Our estimates of Z are calculated as

$$Z(s) \leftarrow (1-\alpha)Z(s) + \alpha e^{r/\lambda} Z(s')^{\gamma}$$

[5] where α is a learning rate that we anneal over time, λ is a regularization parameter that scales the significance of rewards, and γ is a discount factor used in the infinite-horizon case. Notice the similarity of this equation to that of Q-learning (appendix A). Indeed, the two methods share many similarities but differ in a few important respects:

- Z learning multiples an exponentiated version of the reward onto the expectation term. This exponentiation is *crucial* for the theoretical benefits of Z over Q estimation, and will be discussed at more length in section 3.1.
- the dependence on Z-learning is on states only, not state-action pairs, which makes it more efficient than Q-Learning by at least a factor of $|A|$ (the size of the action space).
- in Q-learning, the decision to be made by the agent at each time-step is a matter of choosing the maximum a given s. In Z-learning, however, the decision is not over actions, but rather over desired next states. Remember that in eq. (11), $u(s'|s)$ is a distribution of state-state transitions, not state-action transitions. In Z-learning, actions are selected only *after* a desired state has been sampled this distribution—we choose the action that is most likely to bring about the desired state.

In [5], Todorov carries out a comparison between Q-learning and Z-learning in a small, tabular gridworld environment. In his short experiments, Z-learning was shown to significantly out-preform Q-learning with respect to the time taken to learn the true value-function of the environment (as determined by traditional dynamic programming). For the reader's convenience, a figure summarizing this experiment is reproduced in fig. 1.

The important quality is that Z-learning, like Q-learning is *off-policy*—it will converge to the correct value function, *regardless* of the mechanism used to collect the sample experience, so long as everywhere in the sample space is seen at least once (see fig. 1). To this end, a random-walk policy is often introduced for exploration purposes.

As discussed in appendix B, a random walk is often used as the passive dynamics of an LMDP. For Z-learning, a more efficient alternative to sampling from this passive dynamics is to sample from the estimated Z-optimal control dynamics, computed according to eq. (11). Sampling from the optimal control distribution means that we do a one-step look-ahead and evaluate eq. (11) over the set of next states. After constructing this distribution, we may sample a desired next state, and then select the action that is most likely to get us to that state. This proceedure may seem counter-intuitive. What we are doing is taking a discrete problem (an MDP with discrete actions) and embedding it in a continuous space (a MDP with continuous "actions") where we can specify our ideal transition probabilities between states rather than having these transition probabilities be mediated by actions. Once we sample according to this ideal setting, we then simply undo our embedding and find the action that most likely brings us to the desired next state.

This process of embedding into a continuous space, solving, and then un-embedding is similar, Todorov points out, to the relaxation from integer programming to linear programming.

If we do decide to preform this sampling proceedure, we must introduce an importance sampling weight into our current Z-update function. This weight is the term $w_u(s, s') = \frac{P(s'|s)}{u(s'|s)}$[1] (see also [3]), suchthat our TD update equation becomes:

$$Z(s) \leftarrow (1-\alpha)Z(s) + \alpha e^{r/\lambda} Z(s') w_u(s, s') \tag{1}$$

Note, importantly, that if we are to expand out the value of the importance sampling weight w, we achieve the following for the rightmost term of the above equation:

$$\begin{aligned} e^{r/\lambda} Z(s') w_u(s, s') &= e^{r/\lambda} Z(s') \frac{P(s'|s)}{u(s'|s)} \\ &= e^{r/\lambda} Z(s') \frac{P(s'|s)}{\frac{P(s'|s)Z(s')}{\mathbb{E}[Z(s')]}} \\ &= e^{r/\lambda} \sum_{s'} P(s'|s) Z(s') \end{aligned}$$

the last line of which is exactly equivalent to the tabular LMDP update in eq. (12)! I will show why this result is significant in the function approximation case in section 3.2.

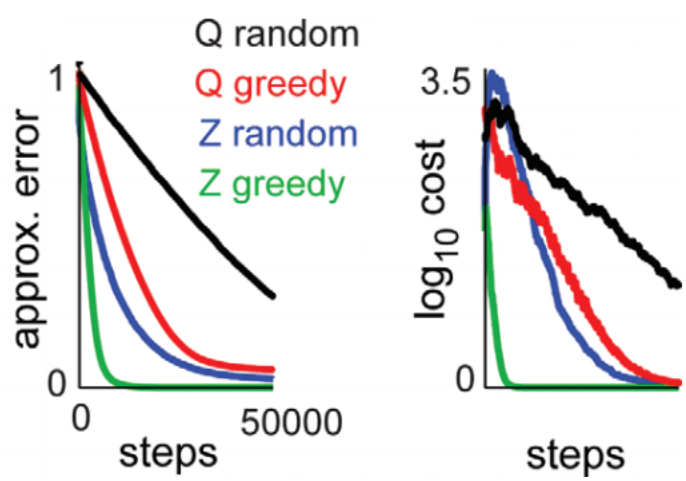

Figure 1: Reproduced from [5]: a comparison between Z-Learning and Q-Learning. Note that Z-learning learns the true value states faster than greedy Q-learning even when using a random exploration policy.

With this, we are now equipped to extend this Z-learning framework for DZL, which we do in the next section.

3 Theory

Before providing an algorithm for both classical Z-learning (for tabular domains) and the deep, function-approximation version (for non-tabular discrete domains), it is first important to discuss the statistical mechanisms that make Z-learning an optimal learning algorithm.

3.1 Duality between estimation and control

The LMDP framework is able to perform its optimal MDP solution approximation in time linear because of a fundamental duality that exists between estimation and control, as discovered in [5]. In that work, Todorov generalizes the well-known duality between estimation and control that arises with the Kalman filter in LQG settings to more general, non-LQG settings. The crux of this generalization is a proof of the following relationship:

$$r(x,t) \propto \exp(-v(s,t)) \tag{2}$$

where v is the familliar cost-to-go function that is the solution to the bellman equation (eq. (6)), $c(x)$ is a scalar, and $r(x,t)$ is the *backwards filtering density*—the probability a future observation trajectory given the current state:

$$r(x,t) = \bar{p}(y_t|x) \sum_{x'} p(x'|x) r(x', t+1) \tag{3}$$

where $\bar{p}(y|x)$ is the *emission* probability of seeing an observation y in state x, and $p(x'|x)$ is the passive dynamics as discussed in the LMDP context appendix B. This result [6] is general and applies to any observations emitted from hidden markov models and partially-observable markov processes more generally, but for the purposes of the MDP problem formulation, our observations y are simply the rewards, r (or costs, c). Thus, the probability of a future cost/reward trajectory starting in a given state is proportional to the the optimal value function's value at that state.

The significance of this duality is that it now allows us a means of converting costs into probabilities and vice-versa, which in turns allows us to use optimal estimation algorithms to solve for the value function and optimal controls. Now the power of the LMDP is elucidated.

3.1.1 Scaling costs/rewards to respect Z as a probability

From the view of the duality described by Todorov in eq. (2), Z values are seen to be probabilities of expected optimal future reward. And the control distribution $u(s'|s)$ is a normalized and passive-dynamics-weighted distribution over next-state probabilities of the same kind.

Thus it is important that in Z-learning, Z-values be restricted to the range [0, 1]. This can be done by keeping the costs, q, positive, such that we have $\exp(-q)Z$, or in the case of rewards, that we re-normalize the rewards suchthat the maximum possible reward is zero, and all other rewards are negative suchthat we have $0 \leq \exp(r)Z \leq 1$.

3.2 The importance of w

As touched on in section 2.1, the importance-sampling weight w allows us to sample behavior according to the current estimate of the optimal distribution. If we ever want to store a memory buffer of transitions, like in the case of deep Q-learning (appendix C), the ability to *store* w is significant because it allows us to to behave according to the optimal control law at decision time, but use what is effectively the full Z update equation (eq. (12)).

The proceedure goes as follows: while interacting with the environment we save tuples of the form (s, r, s', ρ), where $\rho = u(s'|s)$ at the time of the sample. Assuming P remains constant, we can then use ρ to calculate w when it comes time to learn from the transition sample; in other words, we can maintain the full power of eq. (12) in the simple TD setting without having to have to compute $\sum_{s'} P(s'|s)Z(s')$

With this, we have the necessary mechanisms for understanding my derivation of tabular Z-learning, as described in algorithm 1.

Algorithm 1 Tabular Z-learning

```
Initialize Z(s) = 1    ∀s ∈ S
recieve initial s
for t = 1, T do
    Initialize L, an empty list of possible next states
    # one-step look-ahead
    for a = 1, A do
        make a copy of env state
        execute action a in the copy, recieve (simulated) state ŝ'
        append tupple (a, ŝ') to list of next states
    end for
    Create distribution U over states in L according to eq. (11)
    Sample desired x̂' from U
    Set w ← P(x̂|x)/U(x̂'|x̂)
    # retrieve the action that most closely achieves what we want
    Select the action in L that produced ŝ'
    Execute this action and observe reward r_t and true state s'
    Set y_z = { e^{r_j}                  for terminal s'
              { e^{r_j/λ} Z(s')^γ w      for non-terminal s'
    update Z(s) ← (1 − α)Z(s) + α y_z
end for
```

3.3 From Z-Learning to Deep Z-Learning

I now turn to deriving the key contribution of this work – the deep Z-learning algorithm.

Following in the steps for the development of DQL (discussed in appendix C) we now introduce an approximate version of Z-Learning by expressing Z as a parameterized function, $Z_\theta(s)$, giving us the following updated equations:

$$u^*(\cdot|s;\theta) = \frac{P(\cdot|s)Z(\cdot;\theta)}{G[Z](s)} = \frac{P(\cdot|s)Z(\cdot;\theta)}{\sum_{s'} P(s'|s)Z(s';\theta)}$$

$$w_u(s'|s;\theta) = \frac{P(s'|s)}{u(s'|s;\theta)}$$

As was the case with approximate Q-learning, this switch from a look-up table to a function allows us to scale to non-tabular discrete and even continuous state-spaces. This gained flexibility both as a result of not needing to keep values for the entire state space around in memory, but also because with a sufficiently powerful function approximator, we can abstract meaningful features and compress the amount of information needed to decide the desirability of a state.

Similarly to DQL, we calculate the loss as the squared difference between our targeted value with previous theta and current Z prediction.

$$L_i(\theta_i) = \mathbb{E}_{s,r}[(\text{target} - Z(s;\theta_i))^2] \tag{4}$$

Fortunately, this function also has a simple gradient w.r.t the non-fixed weights θ_i

$$\nabla L(\theta_i) = \mathbb{E}_{s,r}[(\text{target} - Z(s;\theta_i)) + \nabla Z(s;\theta_i)] \tag{5}$$

We use a convolutional network identical to DQL's for our Z approximator, and consequently the $\nabla Z(s;\theta_i)$ is calculated via backpropogation.

4 Empirical Evaluation

In previous work, I preformed comparisons between DQL and a simpler variant of DZL. I have moved both a discussion on this alternative DZL formulation, as well as the results, to appendix D.

Shortly after completing that prior work, which showed great potential for DZL over DQL (see appendix D), I came upon related work in [4]. After some investigation, I determined that my prior method and theirs were mathematically equivalent (see again D). In that work, Schulman et al. do much more extensive and thorough testing than I had done on the method, and so I determined it was not necessary to pursue this simpler version of the Z-learning algorithm further.

Instead, I elected to pursue this updated variant, which is is closer in both spirit and mechanism to the LMDP. The prior model avoided doing one-step lookaheads with a model by instead having the network approximate $Z(u(s'|s))$ for an assumed fixed number of possible s'. This was essentially equivalent to the DQL tactic of predicting $Q(s', a')$, for each possible a'. The new Z-learning algorithms I present here do actual 1-step lookaheads, which are both philosophically cleaner and mathematically different.

However, experimentation with this new model has yet to yield conclusory results due to a number of complications with the testing environment and the numerical instabilities related to the exponential transform. In particular, the ALE game environment

Algorithm 2 Deep Z-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize desirability function Z with random weights θ
Initialize desirability target function Z_y with random weights θ'
for episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    Initialize L, an empty list of possible next states
    for $a = 1, A$ **do**
      make a copy of env state
      execute action a in the copy, recieve state $\hat{x}$'
      append tupple (a, $\hat{x}$') to list of next states
    end for
    Create distribution U over states in L according to eq. (12)
    Sample desired $\hat{x}$' from U
    Store probability, ρ, of this state selection
    Select the action in L that produced $\hat{x}'$
    Execute the action and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, \rho_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    # Experience Replay
    Sample random minibatch of transitions $(\phi_j, \rho_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $w \leftarrow P(\hat{x}|x)/\rho_j$
    Set $y_j = \begin{cases} e^{r_j} & \text{for terminal } \phi_{j+1} \\ e^{r_j/\lambda} Z(\phi_{j+1};\theta)^{\gamma} {w_u}^*(\phi_{j+1}|\phi_j) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(\text{target} - Z(\phi_j;\theta))^2$ according to eq. eq. (5)
  end for
end for

used to run the ATARI simulations is not thread-safe, meaning that for the DZL one-step look-aheads could not be easily parallelized and therefore training DZL on these environments took intractably long for the time-frame allotted for experiments.

The biggest challenged faced for experimentation, however, was numerical instability and hyper-parameter tuning. The experiments taking long was not in-and-of-itself a problem; the issue was that the update equations paired with neural network function approximation pose a challenging problem, and all the experiments I ran with DZL diverged.

I argue that this is not a problem with the theory, but rather with my attempts at implementation; the choice of activation functions, hyperparameters, and network initialization were all facets of the function approximation aspect of this thesis that were difficult to gauge.

I plan to continue experimentation with DZL, though I will scale back the scope and demonstrate it working in simpler problem domains before returning to the ATARI

environments.

5 Conclusion

In this thesis, I developed the theory of Z-learning. In particular, I applied the duality between estimation and control to develop a detailed and principled approach to Z-learning, and then extended the theory of Z-learning to the online, non-tabular discrete domain via neural network function approximation.

Empirically, I was unsuccessful at evaluating DZL, and this leaves much to be desired with respect to the results of this work. Future work will, for now, focus solely on developing a robust implementation of Z-learning that does not suffer from the divergence issues I faced in my experiments. In particular, working in log space will be a much more prudent choice going forward.

References

[1] Anders Jonsson and Vicenç Gómez. "Hierarchical Linearly-Solvable Markov Decision Problems (Extended Version with Supplementary Material)". In: *ArXiv preprint* (2016). arXiv: 1603.03267. URL: http://arxiv.org/abs/1603.03267.

[2] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (Dec. 2013). ISSN: 0028-0836. DOI: 10.1038/nature14236. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[3] Art B. Owen. *Monte Carlo theory, methods and examples*. 2013.

[4] John Schulman, Xi Chen, and Pieter Abbeel. "Equivalence Between Policy Gradients and Soft Q-Learning". In: (2017). arXiv: 1704.06440. URL: http://arxiv.org/abs/1704.06440.

[5] Emanuel Todorov. "Efficient computation of optimal actions". In: *Proceedings of the National Academy of Sciences* 106.28 (2009), pp. 11478–11483. ISSN: 0027-8424. DOI: 10.1073/pnas.0710743106. URL: http://www.pnas.org.ezp1.lib.umn.edu/content/106/28/11478.full.pdf%20http://www.pnas.org/lookup/doi/10.1073/pnas.0710743106.

[6] Emanuel Todorov. "General duality between optimal control and estimation". In: (2008), pp. 4286–4292. ISSN: 01912216. DOI: 10.1109/CDC.2008.4739438. URL: https://homes.cs.washington.edu/%7B~%7Dtodorov/papers/TodorovCDC08.pdf.

[7] Emanuel Todorov. "Linearly-solvable Markov decision problems". In: *Cognitive Science* vol. 19.i (2007), pp. 1369–1376. URL: https://papers.nips.cc/paper/3002-linearly-solvable-markov-decision-problems.pdf.

Appendices

A Markov Decision Processes

Markov-Decision Processes (MDPs) are Markov chains in which an agent has influence. The agent has actions, which affect the Markov chain's state-transition probabilities, and rewards, which motivate the agent to choose a sequence of actions to maximize its expected total reward. We represent this formally with discrete state $s \in S$, discrete action $a \in A$, discrete reward $R(s, a) \in R^{|A|x|S|}$. In some problem formulations, there is another subset of states $J \subset S$ that are **terminal states** or **absorbing states**, at which point the processes ends.

We can formalize the decision process of an agent attempting to maximize its reward with the *Bellman equations*, which define a dynamic-programming approach to the problem. The Bellman equations are recursively defined, where the action at each time is decided based on the sum of the current reward plus and expected value of all future rewards if you were to act optimally from the next state onward. Formally, we write:

$$V(s) = \max_a \left(R(s, a) + \gamma \sum_{s'} p(s'|s, a) V(s') \right) \tag{6}$$

Where V is the "value function", $R(s, a)$ is the probability of observing a scalar reward value, r for being in state s and taking action a, $0 \leq \gamma \leq 1$ is a discount factor which biases a trade-off between short and long-term payoff. $P(\cdot|s, a)$ is called the transition function, and is sometimes represented as T.

Value functions are typically calculated through **Value-Iteration**, an algorithm which repeatedly executes the bellman equation to propagate reward from future time events.

In addition to this Value function, we also have a policy $\pi(s)$, which is a mapping from states to actions. Given a value function, we can define a policy as:

$$\pi(s) = \operatorname*{argmax}_a \big(R(s, a) + \gamma * \mathbb{E}_a V(s') \big) \tag{7}$$

Notice that the functions for π and V differ with respect to the use of a max or an argmax. While value iteration solves for the value function V and then constructs a policy according to eq. (7) for this solved value function, another algorithm, **Policy Iteration** is a method that alternates between equations eq. (6) and eq. (7), by starting with a policy π_o to use in eq. (6) instead of the max operator to solve for V_π and then using V_{pi} to calculate a new policy π'. So eq. (6) and eq. (7) can be re-written as:

$$V_\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{\pi(s)} V(s')$$
$$\pi'(s) = \operatorname*{argmax}_a R(s, a) + \gamma \mathbb{E}_a V_\pi(s')$$

One final method for solving MDPs is **Q-Learning**. Q functions take as argument a state-action pair, (s, a) and returns a "quality" score for being in action s and taking

action a. This interesting combination allows us to dispense with policies, because now actions are considered as part of our state. The Q-learning algorithm is based on *temporal difference learning*, which combines online information according to a type of running-average proceedure, where for each new sample an agent experiences,

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} \mathbb{E}_{a'} Q(s',a')\right) \tag{8}$$

Where $0 \leq \alpha \leq 1$ is a learning rate that trades-off between our old and new estimates of Q. Since this temporal difference (TD) update is loosely a type of running average, it is good practice to decay the *alpha* parameter with time to properly weight the value of new observations relative to the amount of observations we have seen so far.

B Linearly Solvable MDPs

Emo Todorov introduced a class of MDPs that are guaranteed solvable in linear time[7][5]. We will reproduce an abbreviated derivation of these linearly solvable MDPs (LMDPs) as they relate and lead up to the theory of Z-learning. For a more in-depth treatment, we refer the reader to the original derivation found in [7] as well as a more RL-focused derivation, which we follow closely, from [1].

The derivation proceeds as follows: We reformulate the classic MDP problem into a continuous control problem by removing explicit representation of actions; rather than having discrete actions that index into transition probabilities, we can have our agent inject **control variables** that directly reshape the transition probabilities themselves. Therefore we can make a distinction between the **passive dynamics** of a system (what the markov chain does without the influence of the agent) and the **control dynamics**, which are the transition probabilities under influence of the agent. In discrete MDPs, we will define the passive dynamics as a random walk.

This formulation allows us to express a cost over different controls u, which we can express as the Kullback-Leibler (KL) divergence between the control and passive dynamics. We then define the reward of being in state s and injecting control u as:

$$R(s,u) = R(s) + \mathbb{KL}(u(\cdot|s)||p(\cdot|s)) \tag{9}$$

The next component of the derivation is to preform a clever mathematical transform of the value function with which which we can express the optimal action s as a function of V. we define the **desirability** of a state as

$$Z(s) = \exp(V(s)) \tag{10}$$

Substituting eq. (9) and eq. (10) back into the Bellman equation, we may complete

the derivation:

$$
\begin{aligned}
V(S) &= \max_u \left(R(s,u) + \mathbb{E}_{s'\sim u(\cdot|s)}[V(s')]\right) \\
\log(Z(s)) &= R(s) - \min_u \left(\mathbb{E}_{s'\sim u(\cdot|s)}\left[\log \frac{u(s'|s)}{p(s'|s)} - \log Z(s')\right]\right) \\
\log(Z(s)) &= R(s) - \min_u \left(\mathbb{E}_{s'\sim u(\cdot|s)}\left[\log \frac{u(s'|s)}{p(s'|s)Z(s')}\right]\right) \\
\log(Z(s)) &= R(s) - \min_u \left(\mathbb{E}_{s'\sim u(\cdot|s)}\left[\frac{u(s'|s)G[Z](s)}{p(s'|s)Z(s')} - \log G[Z](s)\right]\right) \\
\log(Z(s)) &= R(s) + \log G[Z](s) - \min_u \left(\mathrm{KL}\left(u(\cdot|s)\middle|\middle|\frac{P(\cdot|s)Z(\cdot)}{G[Z](s)}\right)\right)
\end{aligned}
$$

[1] Where $G[Z] = \sum_{s'} P(s'|s)Z(s')$ is a normalization term to ensure the KL is well-defined. Our optimal control is the choice of u that minimizes the KL divergence, namely

$$
u^*(\cdot|s) = \frac{P(\cdot|s)Z(\cdot)}{G[Z](s)} \tag{11}
$$

Substituting this term back in and exponentiating both sides gives us a concise expression for Z:

$$
Z(s) = \exp R(s)G[Z](s) = \exp R(s) \sum_{s'} P(s'|s)Z(s') \tag{12}
$$

This equation may now be vectorized and solved with an eigen-solver in linear time.

One thing left out of our derivation for clarity is an optional regularization term, λ, which [1] use in their derivation. With the lambda parameter, our final equation becomes $Z(s) = \exp\left(\frac{R(s)}{\lambda}\right) \sum_{s'} P(s'|s)Z(s')$.

C Deep Q-Learning

Deep Q-Learning (DQL) was introduced by [2], who were able to apply Q-learning techniques (see appendix A) to play Atari2600 video games at a level that exceeded human performance in several domains. In addition to applying convolutional networks to learn feature representations of the input images of the Atari games, this work made some meaningful alterations to the vanilla Q-learning formulation to make it amenable to the deep learning setting.

Firstly, they treated Q as a parameterized function Q_θ, which can be trained with gradient descent as long as the function is differentiable. Secondly, they utilized *two* networks for the purposes of training this function approximator. One network was continuously, actively being trained to approximate Q-values. More on this will be said in a moment.

Practically, we keep two sets of parameters θ_i and θ_{i-1} – one for the training network, and one for the target network, respectively. Doing so, we can re-write eq. eq. (8) as:

$$
Q(s,a;\theta_i) = (1-\alpha)Q(s,a;\theta_i) + \alpha\mathbb{E}_{p(s'|s,a')}\left(R(s,a,s') + \gamma \max_{a'} Q(s',a';\theta_{i-1})\right) \tag{13}
$$

Note that, having both the training and the target network use the same architecture, we can update the target network by simply copying the parameters from the training network at update time: $\theta_{i-1} = \theta_i$

Additionally, [2] used *experience replay* as a technique to increase data-efficiency and stationarity of the Q-values. Experience replay attempts to produce *i.i.d* samples from otherwise correlated experience. It does so by maintaining a history of (s, r, a, s') tuples, recorded from interaction with the environment, that can be randomly sampled from the memory buffer and used as training data.

Before reproducing the definition of the algorithm, we lastly highlight one key insight of the DQL paper. This key idea was to make RL look like supervised learning. Keeping one set of parameters for the network fixed to predict the quality of the next state provides a target that the other set of parameters can be trained to hit. In this way, the target is a type of label for supervised learning. As the network increases in accuracy, the targets will be come more accurate, bootstrapping to the true values of the state-action pairs. With this nomenclature in mind, we re-write eq. eq. (13) as $Q(s, a; \theta_i) = (1-\alpha)Q(s, a; \theta_i) + \alpha * \text{target}[s']$ and can calculate the gradient of the network parameters θ using MSE loss between the target and $Q(s, a)$ output:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}[(\text{target} - Q(s, a; \theta_i))^2] \tag{14}$$

[2]

This function has a simple gradient w.r.t the weights θ_i

$$\nabla L(\theta_i) = \mathbb{E}_{s,a,r}[(\text{target} - Q(s, a; \theta_i)) + \nabla Q(s, a; \theta_i)] \tag{15}$$

and since $Q(s, a; \theta_i)$ is represented by a neural network, its gradient is calculated via backpropogation.

Algorithm 1 provides a reproduction of the original DQL algorithm as defined in [2]

D Model-Based vs Model-Free Z-Learning

As we've been discussing, Z-learning requires the ability to do a one-step look-ahead in order to behave according to its optimal control law, eq. (11). In this way, Z-learning is fundamentally a model-based method.

However, it is possible to regain a type of model-freeness when using Z-learning. If we take the input to the Z-function (which we'll now abstract and refer to by the variable, x) to be a tuple containing variables s and a, where s is an environment state and a is an action, we can learn the value of (s, a) through Z-learning. As with Q-learning, learning (s, a) values allows us to not have to worry about simulating $p(s'|a)$. Thus Z-learning with this composite state is similar to Q-learning, insofar as both are model-free *with respect to s, a*. In this case, learning Z-values is almost equivalent to learning Q-values, except for the difference in how Q and Z calculate their respective targets. whereas in Q we have

$$y_Q = r + \gamma \max_a Q(s', a')$$

Algorithm 3 Deep Q-learning with Experience Replay **(Reproduced from [2])**

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize desirability function Z with random weights
for episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to eq. 6
  end for
end for

we have in Z,

$$y_z = exp(r)Z((s', a'))w = exp(r) \sum_{(s',a')} Z((s', a'))$$

The main difference is with respect to the max over actions in the case of Q-learning vs the summand over actions in the case of Z-learning. This difference will actually prove significant. Indeed, though I will not reproduce the proof here for the sake of brevity[1], it can be proven that this latter update equation is equivalent to the recently developments in "soft Q-learning" (e.g., [4]), which has the following target equation in place of the ones listed above:

$$y_Q = r + \gamma \log \sum_{a'} \exp(Q(s', a'))$$

in which surface similarity can be seen through calculating the logarithmic transform of the y_z target equation.

We now have the necessary mechanisms for understanding my derivations for both the model-based and model-free versions of the Z-learning algorithm, provided below.

D.1 The Model-Free Z-Learning Algorithm

This algorithm here in a tabular domain applies only to cases where actions are deterministic. If we have discrete actions, sampling from this optimal control distribution amounts to sampling from a distribution over the actions. This gives us the following model-free Z-learning algorithm.

[1]This proof will feature in my future works that will focus more on the relationships between estimation and control

Algorithm 4 The Z-Learning Algorithm

Initialize $Z(s) = 0 \forall s \in S$ randomly
receive initial x, r, x'
for t $= 1, \ldots$ **do**
  Retrieve x' from priority sampling $u^*(\cdot|s;\theta)$ (equivalent to sampling and executing an action in deterministic domains)
  Observe reward r during transition to s'
  set target $= \begin{cases} \exp(r/\lambda) & \text{for terminal } s' \\ \exp(r/\lambda) Z(s';\theta)^\gamma w_u(s, s';\theta) & \text{for non-terminal } s' \end{cases}$
  update $Z(s) \leftarrow (1-\alpha)Z(s) + \alpha \text{target}$
end for

D.1.1 The Model-Free Deep Z-Learning Algorithm

The following algorithm is a direct modification of the original DQL algorithm in [2], where we have simply updated the parts of the algorithm relating to Z-learning.

Algorithm 5 Deep Z-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize desirability function Z with random weights θ
for episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    Select a_t according to eq. eq. (1) given θ
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} e^{r_j} & \text{for terminal } \phi_{j+1} \\ e^{r_j/\lambda} Z(\phi_{j+1};\theta)^\gamma {w_u}^*(\phi_{j+1}|\phi_j) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(\text{target} - Z(\phi_j;\theta))^2$ according to eq. eq. (5)
  end for
end for

D.2 Results

We present the average performance of the two algorithms in figure 2, and more detailed statistics in table 1.

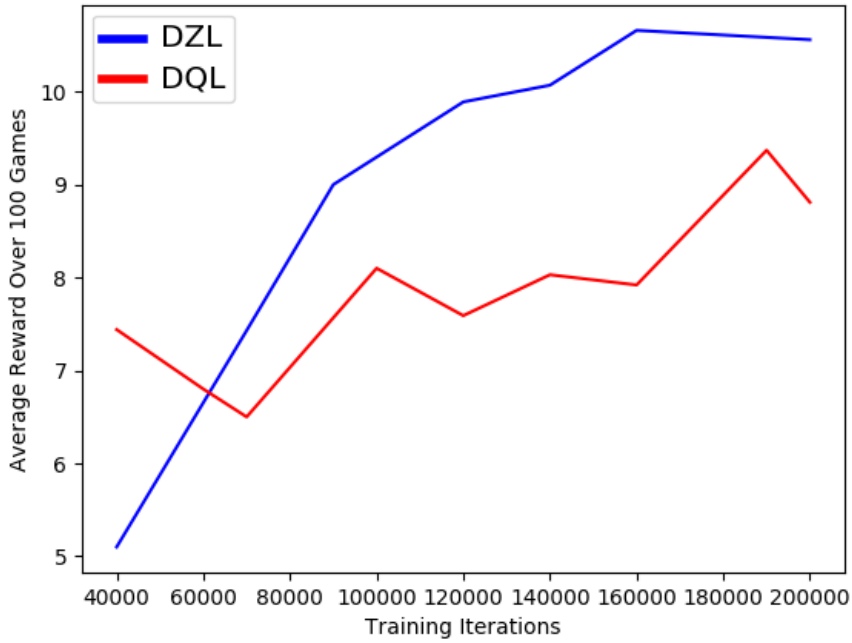

Figure 2: A comparison of average reward as a function of iteration count. DZL appears to have both a smoother trajectory and higher average reward in all but the earliest period of training

Iterations	**Avg.**	**Std**	**Max**	**Min**
40,000	5.1, 2.06	2.06, 3.24	12, 19	1, 1
120,000	9.89, 7.59	4.08, 2.97	20, 18	2, 2
200,000	10.56, 8.81	4.33, 3.84	21, 21	3, 2

Table 1: Statistics of selected runs of the DZL and DQL algorithms. Blue is DZL and red is DQL