
Learning values across many orders of magnitude

Hado van Hasselt Arthur Guez Matteo Hessel Volodymyr Mnih David Silver

Google DeepMind

Abstract

Most learning algorithms are not invariant to the scale of the signal that is being approximated. We propose to adaptively normalize the targets used in the learning updates. This is important in value-based reinforcement learning, where the magnitude of appropriate value approximations can change over time when we update the policy of behavior. Our main motivation is prior work on learning to play Atari games, where the rewards were clipped to a predetermined range. This clipping facilitates learning across many different games with a single learning algorithm, but a clipped reward function can result in qualitatively different behavior. Using adaptive normalization we can remove this domain-specific heuristic without diminishing overall performance.

1 Introduction

Many machine-learning algorithms rely on a-priori access to data to properly tune relevant hyperparameters [Bergstra et al., 2011, Bergstra and Bengio, 2012, Snoek et al., 2012]. It is much harder to learn efficiently from a stream of data when we do not know the magnitude of the function we seek to approximate beforehand, or if these magnitudes can change over time, as is typically the case in reinforcement learning when the policy of behavior improves over time.

Our main motivation is the work by Mnih et al. [2015], in which Q-learning [Watkins, 1989] is combined with a deep convolutional neural network [cf. LeCun et al., 2015]. The resulting deep Q network (DQN) algorithm learned to play a varied set of Atari 2600 games from the Arcade Learning Environment (ALE) [Bellemare et al., 2013], which was proposed as an evaluation framework to test general learning algorithms on solving many different interesting tasks. DQN was proposed as a singular solution, using a single set of hyperparameters.

The magnitudes and frequencies of rewards vary wildly between different games. For instance, in Pong the rewards are bounded by -1 and $+1$ while in Ms. Pac-Man eating a single ghost can yield a reward of up to $+1600$. To overcome this hurdle, rewards and temporal-difference errors were clipped to $[-1, 1]$, so that DQN would perceive any positive reward as $+1$, and any negative reward as -1 . This is not a satisfying solution for two reasons. First, the clipping introduces domain knowledge. Most games have sparse non-zero rewards. Clipping results in optimizing the frequency of rewards, rather than their sum. This is a fairly reasonable heuristic in Atari, but it does not generalize to many other domains. Second, and more importantly, the clipping changes the objective, sometimes resulting in qualitatively different policies of behavior.

We propose a method to adaptively normalize the targets used in the learning updates. If these targets are guaranteed to be normalized it is much easier to find suitable hyperparameters. The proposed technique is not specific to DQN or to reinforcement learning and is more generally applicable in supervised learning and reinforcement learning. There are several reasons such normalization can be desirable. First, sometimes we desire a single system that is able to solve multiple different problems with varying natural magnitudes, as in the Atari domain. Second, for multi-variate functions the

normalization can be used to disentangle the natural magnitude of each component from its relative importance in the loss function. This is particularly useful when the components have different units, such as when we predict signals from sensors with different modalities. Finally, adaptive normalization can help deal with non-stationary. For instance, in reinforcement learning the policy of behavior can change repeatedly during learning, thereby changing the distribution and magnitude of the values.

1.1 Related work

Input normalization has long been recognized as important to efficiently learn non-linear approximations such as neural networks [LeCun et al., 1998], leading to research on how to achieve scale-invariance on the inputs [e.g., Ross et al., 2013, Ioffe and Szegedy, 2015, Desjardins et al., 2015]. Output or target normalization has not received as much attention, probably because in supervised learning data sets are commonly available before learning commences, making it straightforward to determine appropriate normalizations or to tune hyper-parameters. However, this assumes the data is available a priori, which is not true in online (potentially non-stationary) settings.

Natural gradients [Amari, 1998] are invariant to reparameterizations of the function approximation, thereby avoiding many scaling issues, but these are computationally expensive for functions with many parameters such as deep neural networks. This is why approximations are regularly proposed, typically trading off accuracy to computation [Martens and Grosse, 2015], and sometimes focusing on a certain aspect such as input normalization [Desjardins et al., 2015, Ioffe and Szegedy, 2015]. Most such algorithms are not fully invariant to the scale of the target function.

In the Atari domain several algorithmic variants and improvements for DQN have been proposed [van Hasselt et al., 2016, Bellemare et al., 2016, Schaul et al., 2016, Wang et al., 2016], as well as alternative solutions [Liang et al., 2016, Mnih et al., 2016]. However, none of these address the clipping of the rewards or explicitly discuss the impacts of clipping on performance or behavior.

1.2 Preliminaries

Concretely, we consider learning from a stream of data $\{(X_t, Y_t)\}_{t=1}^{\infty}$ where the inputs $X_t \in \mathbb{R}^n$ and targets $Y_t \in \mathbb{R}^k$ are real-valued tensors. The aim is to update parameters θ of a function $f_{\theta} : \mathbb{R}^n \rightarrow \mathbb{R}^k$ such that the output $f_{\theta}(X_t)$ is (in expectation) close to the target Y_t according to some loss $l_t(f_{\theta})$, for instance defined as a squared difference: $l_t(f_{\theta}) = \frac{1}{2}(f_{\theta}(X_t) - Y_t)^{\top}(f_{\theta}(X_t) - Y_t)$. A canonical update is stochastic gradient descent (SGD). For a sample (X_t, Y_t) , the update is then $\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} l_t(f_{\theta})$, where $\alpha \in [0, 1]$ is a step size. The magnitude of this update depends on both the step size and the loss, and it is hard to pick suitable step sizes when nothing is known about the magnitude of the loss.

An important special case is when f_{θ} is a neural network [McCulloch and Pitts, 1943, Rosenblatt, 1962], which are often trained with a form of SGD [Rumelhart et al., 1986], with hyperparameters that interact with the scale of the loss. Especially for deep neural networks [LeCun et al., 2015, Schmidhuber, 2015] large updates may harm learning, because these networks are highly non-linear and such updates may ‘bump’ the parameters to regions with high error.

2 Adaptive normalization with Pop-Art

We propose to normalize the targets Y_t , where the normalization is learned separately from the approximating function. We consider an affine transformation of the targets

$$\tilde{Y}_t = \Sigma_t^{-1}(Y_t - \mu_t), \tag{1}$$

where Σ_t and μ_t are *scale* and *shift* parameters that are learned from data. The scale matrix Σ_t can be dense, diagonal, or defined by a scalar σ_t as $\Sigma_t = \sigma_t \mathbf{I}$. Similarly, the shift vector μ_t can contain separate components, or be defined by a scalar μ_t as $\mu_t = \mu_t \mathbf{1}$. We can then define a loss on a normalized function $g(X_t)$ and the normalized target \tilde{Y}_t . The unnormalized approximation for any input x is then given by $f(x) = \Sigma g(x) + \mu$, where g is the *normalized function* and f is the *unnormalized function*.

At first glance it may seem we have made little progress. If we learn Σ and μ using the same algorithm as used for the parameters of the function g , then the problem has not become fundamentally different

or easier; we would have merely changed the structure of the parameterized function slightly. Conversely, if we consider tuning the scale and shift as hyperparameters then tuning them is not fundamentally easier than tuning other hyperparameters, such as the step size, directly.

Fortunately, there is an alternative. We propose to update Σ and μ according to a separate objective with the aim of normalizing the updates for g . Thereby, we decompose the problem of learning an appropriate normalization from learning the specific shape of the function. The two properties that we want to simultaneously achieve are

- (ART) to update scale Σ and shift μ such that $\Sigma^{-1}(Y - \mu)$ is appropriately normalized, and
- (POP) to preserve the outputs of the unnormalized function when we change the scale and shift.

We discuss these properties separately below. We refer to algorithms that combine output-preserving updates and adaptive rescaling, as **Pop-Art** algorithms, an acronym for ‘‘Preserving Outputs Precisely, while Adaptively Rescaling Targets’’.

2.1 Preserving outputs precisely

Unless care is taken, repeated updates to the normalization might make learning harder rather than easier because the normalized targets become non-stationary. More importantly, whenever we adapt the normalization based on a certain target, this would simultaneously change the output of the unnormalized function of all inputs. If there is little reason to believe that other unnormalized outputs were incorrect, this is undesirable and may hurt performance in practice, as illustrated in Section 3. We now first discuss how to prevent these issues, before we discuss how to update the scale and shift.

The only way to avoid changing all outputs of the unnormalized function whenever we update the scale and shift is by changing the normalized function g itself simultaneously. The goal is to preserve the outputs from before the change of normalization, for all inputs. This prevents the normalization from affecting the approximation, which is appropriate because its objective is solely to make learning easier, and to leave solving the approximation itself to the optimization algorithm.

Without loss of generality the unnormalized function can be written as

$$f_{\theta, \Sigma, \mu, \mathbf{W}, \mathbf{b}}(x) \equiv \Sigma g_{\theta, \mathbf{W}, \mathbf{b}}(x) + \mu \equiv \Sigma(\mathbf{W}h_{\theta}(x) + \mathbf{b}) + \mu, \quad (2)$$

where h_{θ} is a parametrized (non-linear) function, and $g_{\theta, \mathbf{W}, \mathbf{b}} = \mathbf{W}h_{\theta}(x) + \mathbf{b}$ is the normalized function. It is not uncommon for deep neural networks to end in a linear layer, and then h_{θ} can be the output of the last (hidden) layer of non-linearities. Alternatively, we can always add a square linear layer to any non-linear function h_{θ} to ensure this constraint, for instance initialized as $\mathbf{W}_0 = \mathbf{I}$ and $\mathbf{b}_0 = \mathbf{0}$.

The following proposition shows that we can update the parameters \mathbf{W} and \mathbf{b} to fulfill the second desideratum of preserving outputs precisely for any change in normalization.

Proposition 1. Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^k$ defined as in (2) as

$$f_{\theta, \Sigma, \mu, \mathbf{W}, \mathbf{b}}(x) \equiv \Sigma(\mathbf{W}h_{\theta}(x) + \mathbf{b}) + \mu,$$

where $h_{\theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is any non-linear function of $x \in \mathbb{R}^n$, Σ is a $k \times k$ matrix, μ and \mathbf{b} are k -element vectors, and \mathbf{W} is a $k \times m$ matrix. Consider any change of the scale and shift parameters from Σ to Σ_{new} and from μ to μ_{new} , where Σ_{new} is non-singular. If we then additionally change the parameters \mathbf{W} and \mathbf{b} to \mathbf{W}_{new} and \mathbf{b}_{new} , defined by

$$\mathbf{W}_{\text{new}} = \Sigma_{\text{new}}^{-1} \Sigma \mathbf{W} \quad \text{and} \quad \mathbf{b}_{\text{new}} = \Sigma_{\text{new}}^{-1} (\Sigma \mathbf{b} + \mu - \mu_{\text{new}}),$$

then the outputs of the unnormalized function f are preserved precisely in the sense that

$$f_{\theta, \Sigma, \mu, \mathbf{W}, \mathbf{b}}(x) = f_{\theta, \Sigma_{\text{new}}, \mu_{\text{new}}, \mathbf{W}_{\text{new}}, \mathbf{b}_{\text{new}}}(x), \quad \forall x.$$

This and later propositions are proven in the appendix. For the special case of scalar scale and shift, with $\Sigma \equiv \sigma \mathbf{I}$ and $\mu \equiv \mu \mathbf{1}$, the updates to \mathbf{W} and \mathbf{b} become $\mathbf{W}_{\text{new}} = (\sigma/\sigma_{\text{new}})\mathbf{W}$ and $\mathbf{b}_{\text{new}} = (\sigma \mathbf{b} + \mu - \mu_{\text{new}})/\sigma_{\text{new}}$. After updating the scale and shift we can update the output of the normalized function $g_{\theta, \mathbf{W}, \mathbf{b}}(X_t)$ toward the normalized output \tilde{Y}_t , using any learning algorithm. Importantly, the normalization can be updated first, thereby avoiding harmful large updates just before they would otherwise occur. This observation is made more precise in Proposition 2 in Section 2.2.

Algorithm 1 SGD on squared loss with Pop-Art

For a given differentiable function h_θ , initialize θ .
Initialize $\mathbf{W} = I$, $\mathbf{b} = \mathbf{0}$, $\Sigma = I$, and $\mu = \mathbf{0}$.
while learning **do**
 Observe input X and target Y
 Use Y to compute new scale Σ_{new} and new shift μ_{new}
 $\mathbf{W} \leftarrow \Sigma_{\text{new}}^{-1} \Sigma \mathbf{W}$, $\mathbf{b} \leftarrow \Sigma_{\text{new}}^{-1} (\Sigma \mathbf{b} + \mu - \mu_{\text{new}})$ (rescale \mathbf{W} and \mathbf{b})
 $\Sigma \leftarrow \Sigma_{\text{new}}$, $\mu \leftarrow \mu_{\text{new}}$ (update scale and shift)
 $\mathbf{h} \leftarrow h_\theta(X)$ (store output of h_θ)
 $\mathbf{J} \leftarrow (\nabla_\theta h_{\theta,1}(X), \dots, \nabla_\theta h_{\theta,m}(X))$ (compute Jacobian of h_θ)
 $\delta \leftarrow \mathbf{W} \mathbf{h} + \mathbf{b} - \Sigma^{-1} (Y - \mu)$ (compute normalized error)
 $\theta \leftarrow \theta - \alpha \mathbf{J} \mathbf{W}^\top \delta$ (compute SGD update for θ)
 $\mathbf{W} \leftarrow \mathbf{W} - \alpha \delta \mathbf{h}^\top$ (compute SGD update for \mathbf{W})
 $\mathbf{b} \leftarrow \mathbf{b} - \alpha \delta$ (compute SGD update for \mathbf{b})
end while

Algorithm 1 is an example implementation of SGD with Pop-Art for a squared loss. It can be generalized easily to any other loss by changing the definition of δ . Notice that \mathbf{W} and \mathbf{b} are updated twice: first to adapt to the new scale and shift to preserve the outputs of the function, and then by SGD. The order of these updates is important because it allows us to use the new normalization immediately in the subsequent SGD update.

2.2 Adaptively rescaling targets

A natural choice is to normalize the targets to approximately have zero mean and unit variance. For clarity and conciseness, we consider scalar normalizations. It is straightforward to extend to diagonal or dense matrices. If we have data $\{(X_i, Y_i)\}_{i=1}^t$ up to some time t , we then may desire

$$\begin{aligned} \sum_{i=1}^t (Y_i - \mu_t) / \sigma_t &= 0 & \text{and} & & \frac{1}{t} \sum_{i=1}^t (Y_i - \mu_t)^2 / \sigma_t^2 &= 1, \\ \text{such that } \mu_t &= \frac{1}{t} \sum_{i=1}^t Y_i & \text{and} & & \sigma_t &= \frac{1}{t} \sum_{i=1}^t Y_i^2 - \mu_t^2. \end{aligned} \quad (3)$$

This can be generalized to incremental updates

$$\mu_t = (1 - \beta_t) \mu_{t-1} + \beta_t Y_t \quad \text{and} \quad \sigma_t^2 = \nu_t - \mu_t^2, \quad \text{where} \quad \nu_t = (1 - \beta_t) \nu_{t-1} + \beta_t Y_t^2. \quad (4)$$

Here ν_t estimates the second moment of the targets and $\beta_t \in [0, 1]$ is a step size. If $\nu_t - \mu_t^2$ is positive initially then it will always remain so, although to avoid issues with numerical precision it can be useful to enforce a lower bound explicitly by requiring $\nu_t - \mu_t^2 \geq \epsilon$ with $\epsilon > 0$. For full equivalence to (3) we can use $\beta_t = 1/t$. If $\beta_t = \beta$ is constant we get exponential moving averages, placing more weight on recent data points which is appropriate in non-stationary settings.

A constant β has the additional benefit of never becoming negligibly small. Consider the first time a target is observed that is much larger than all previously observed targets. If β_t is small, our statistics would adapt only slightly, and the resulting update may be large enough to harm the learning. If β_t is not too small, the normalization can adapt to the large target before updating, potentially making learning more robust. In particular, the following proposition holds.

Proposition 2. *When using updates (4) to adapt the normalization parameters σ and μ , the normalized targets are bounded for all t by*

$$-\sqrt{(1 - \beta_t) / \beta_t} \leq (Y_t - \mu_t) / \sigma_t \leq \sqrt{(1 - \beta_t) / \beta_t}.$$

For instance, if $\beta_t = \beta = 10^{-4}$ for all t , then the normalized target is guaranteed to be in $(-100, 100)$. Note that Proposition 2 does not rely on any assumptions about the distribution of the targets. This is an important result, because it implies we can bound the potential normalized errors before learning, without any prior knowledge about the actual targets we may observe.

Algorithm 2 Normalized SGD

For a given differentiable function h_θ , initialize θ .

while learning **do**

 Observe input X and target Y

 Use Y to compute new scale Σ

$\mathbf{h} \leftarrow h_\theta(X)$ (store output of h_θ)

$\mathbf{J} \leftarrow (\nabla h_{\theta,1}(X), \dots, \nabla h_{\theta,m}(X))^\top$ (compute Jacobian of h_θ)

$\delta \leftarrow \mathbf{W}\mathbf{h} + \mathbf{b} - Y$ (compute *unnormalized* error)

$\theta \leftarrow \theta - \alpha \mathbf{J}(\Sigma^{-1}\mathbf{W})^\top \Sigma^{-1} \delta$ (update θ with scaled SGD)

$\mathbf{W} \leftarrow \mathbf{W} - \alpha \delta \mathbf{g}^\top$ (update \mathbf{W} with SGD)

$\mathbf{b} \leftarrow \mathbf{b} - \alpha \delta$ (update \mathbf{b} with SGD)

end while

It is an open question whether it is uniformly best to normalize by mean and variance. In the appendix we discuss other normalization updates, based on percentiles and mini-batches, and derive correspondences between all of these.

2.3 An equivalence for stochastic gradient descent

We now step back and analyze the effect of the magnitude of the errors on the gradients when using regular SGD. This analysis suggests a different normalization algorithm, which has an interesting correspondence to Pop-Art SGD.

We consider SGD updates for an unnormalized multi-layer function of form $f_{\theta, \mathbf{W}, \mathbf{b}}(X) = \mathbf{W}h_\theta(X) + \mathbf{b}$. The update for the weight matrix \mathbf{W} is

$$\mathbf{W}_t = \mathbf{W}_{t-1} + \alpha_t \delta_t h_{\theta_t}(X_t)^\top,$$

where $\delta_t = f_{\theta, \mathbf{W}, \mathbf{b}}(X) - Y_t$ is gradient of the squared loss, which we here call the unnormalized error. The magnitude of this update depends linearly on the magnitude of the error, which is appropriate when the inputs are normalized, because then the ideal scale of the weights depends linearly on the magnitude of the targets.¹

Now consider the SGD update to the parameters of h_θ , $\theta_t = \theta_{t-1} - \alpha \mathbf{J}_t \mathbf{W}_{t-1}^\top \delta_t$ where $\mathbf{J}_t = (\nabla g_{\theta,1}(X), \dots, \nabla g_{\theta,m}(X))^\top$ is the Jacobian for h_θ . The magnitudes of both the weights \mathbf{W} and the errors δ depend linearly on the magnitude of the targets. This means that the magnitude of the update for θ depends quadratically on the magnitude of the targets. There is no compelling reason for these updates to depend at all on these magnitudes because the weights in the top layer already ensure appropriate scaling. In other words, for each doubling of the magnitudes of the targets, the updates to the lower layers quadruple for no clear reason.

This analysis suggests an algorithmic solution, which seems to be novel in and of itself, in which we track the magnitudes of the targets in a separate parameter σ_t , and then multiply the updates for all lower layers with a factor σ_t^{-2} . A more general version of this for matrix scalings is given in Algorithm 2. We prove an interesting, and perhaps surprising, connection to the Pop-Art algorithm.

Proposition 3. Consider two functions defined by

$$f_{\theta, \Sigma, \mu, \mathbf{W}, \mathbf{b}}(x) = \Sigma(\mathbf{W}h_\theta(x) + \mathbf{b}) + \mu \quad \text{and} \quad f_{\theta, \mathbf{W}, \mathbf{b}}(x) = \mathbf{W}h_\theta(x) + \mathbf{b},$$

where h_θ is the same differentiable function in both cases, and the functions are initialized identically, using $\Sigma_0 = \mathbf{I}$ and $\mu = \mathbf{0}$, and the same initial θ_0 , \mathbf{W}_0 and \mathbf{b}_0 . Consider updating the first function using Algorithm 1 (Pop-Art SGD) and the second using Algorithm 2 (Normalized SGD). Then, for any sequence of non-singular scales $\{\Sigma_t\}_{t=1}^\infty$ and shifts $\{\mu_t\}_{t=1}^\infty$, the algorithms are equivalent in the sense that 1) the sequences $\{\theta_t\}_{t=0}^\infty$ are identical, 2) the outputs of the functions are identical, for any input.

The proposition shows a duality between normalizing the targets, as in Algorithm 1, and changing the updates, as in Algorithm 2. This allows us to gain more intuition about the algorithm. In particular,

¹In general care should be taken that the inputs are well-behaved; this is exactly the point of recent work on input normalization [Ioffe and Szegedy, 2015, Desjardins et al., 2015].

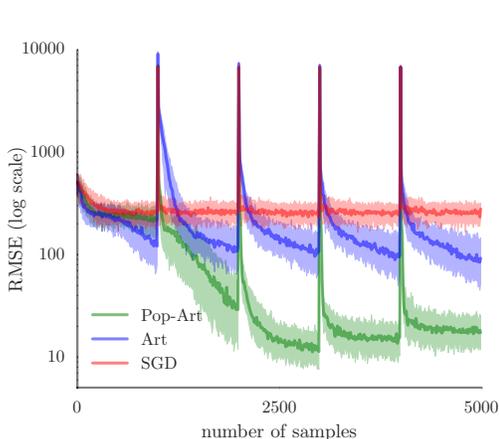


Fig. 1a. Median RMSE on binary regression for SGD without normalization (**red**), with normalization but without preserving outputs (**blue**, labeled ‘Art’), and with Pop-Art (**green**). Shaded 10–90 percentiles.

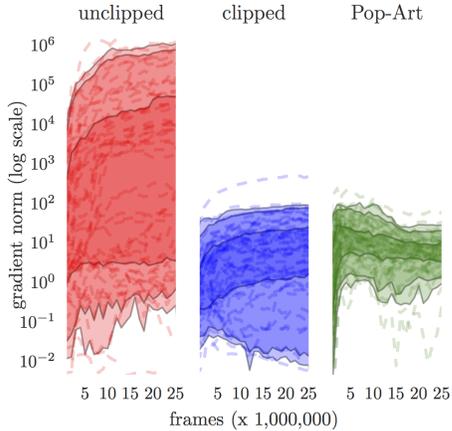


Fig. 1b. ℓ^2 gradient norms for DQN during learning on 57 Atari games with actual unclipped rewards (left, **red**), clipped rewards (middle, **blue**), and using Pop-Art (right, **green**) instead of clipping. Shaded areas correspond to 95%, 90% and 50% of games.

in Algorithm 2 the updates in top layer are not normalized, thereby allowing the last linear layer to adapt to the scale of the targets. This is in contrast to other algorithms that have some flavor of adaptive normalization, such as RMSprop [Tieleman and Hinton, 2012], AdaGrad [Duchi et al., 2011], and Adam [Kingma and Adam, 2015] that each component in the gradient by a square root of an empirical second moment of that component. That said, these methods are complementary, and it is straightforward to combine Pop-Art with other optimization algorithms than SGD.

3 Binary regression experiments

We first analyze the effect of rare events in online learning, when infrequently a much larger target is observed. Such events can for instance occur when learning from noisy sensors that sometimes captures an actual signal, or when learning from sparse non-zero reinforcements. We empirically compare three variants of SGD: without normalization, with normalization but without preserving outputs precisely (i.e., with ‘Art’, but without ‘Pop’), and with Pop-Art.

The inputs are binary representations of integers drawn uniformly randomly between 0 and $n = 2^{10} - 1$. The desired outputs are the corresponding integer values. Every 1000 samples, we present the binary representation of $2^{16} - 1$ as input (i.e., all 16 inputs are 1) and as target $2^{16} - 1 = 65,535$. The approximating function is a fully connected neural network with 16 inputs, 3 hidden layers with 10 nodes per layer, and tanh internal activation functions. This simple setup allows extensive sweeps over hyper-parameters, to avoid bias towards any algorithm by the way we tune these. The step sizes α for SGD and β for the normalization are tuned by a grid search over $\{10^{-5}, 10^{-4.5}, \dots, 10^{-1}, 10^{-0.5}, 1\}$.

Figure 1a shows the root mean squared error (RMSE, log scale) for each of 5000 samples, before updating the function (so this is a test error, not a train error). The solid line is the median of 50 repetitions, and shaded region covers the 10th to 90th percentiles. The plotted results correspond to the best hyper-parameters according to the overall RMSE (i.e., area under the curve). The lines are slightly smoothed by averaging over each 10 consecutive samples.

SGD favors a relatively small step size ($\alpha = 10^{-3.5}$) to avoid harmful large updates, but this slows learning on the smaller updates; the error curve is almost flat in between spikes. SGD with adaptive normalization (labeled ‘Art’) can use a larger step size ($\alpha = 10^{-2.5}$) and therefore learns faster, but has high error after the spikes because the changing normalization also changes the outputs of the smaller inputs, increasing the errors on these. In comparison, Pop-Art performs much better. It prefers the same step size as Art ($\alpha = 10^{-2.5}$), but Pop-Art can exploit a much faster rate for the statistics (best performance with $\beta = 10^{-0.5}$ for Pop-Art and $\beta = 10^{-4}$ for Art). The faster tracking of statistics protects Pop-Art from the large spikes, while the output preservation avoids invalidating

the outputs for smaller targets. We ran experiments with RMSprop but left these out of the figure as the results were very similar to SGD.

4 Atari 2600 experiments

An important motivation for this work is reinforcement learning with non-linear function approximators such as neural networks (sometimes called *deep reinforcement learning*). The goal is to predict and optimize action values defined as the expected sum of future rewards. These rewards can differ arbitrarily from one domain to the next, and non-zero rewards can be sparse. As a result, the action values can span a varied and wide range which is often unknown before learning commences.

Mnih et al. [2015] combined Q-learning with a deep neural network in an algorithm called DQN, which impressively learned to play many games using a single set of hyper-parameters. However, as discussed above, to handle the different reward magnitudes with a single system all rewards were clipped to the interval $[-1, 1]$. This is harmless in some games, such as Pong where no reward is ever higher than 1 or lower than -1 , but it is not satisfactory as this heuristic introduces specific domain knowledge that optimizing reward frequencies is approximately as useful as optimizing the total score. However, the clipping makes the DQN algorithm blind to differences between certain actions, such as the difference in reward between eating a ghost (reward ≥ 100) and eating a pellet (reward = 25) in Ms. Pac-Man. We hypothesize that 1) overall performance decreases when we turn off clipping, because it is not possible to tune a step size that works on many games, 2) that we can regain much of the lost performance by with Pop-Art. The goal is not to improve state-of-the-art performance, but to remove the domain-dependent heuristic that is induced by the clipping of the rewards, thereby uncovering the true rewards.

We ran the Double DQN algorithm [van Hasselt et al., 2016] in three versions: without changes, without clipping both rewards and temporal difference errors, and without clipping but additionally using Pop-Art. The targets are the cumulation of a reward and the discounted value at the next state:

$$Y_t = R_{t+1} + \gamma Q(S_t, \underset{a}{\operatorname{argmax}} Q(S_t, a; \theta); \theta^-), \quad (5)$$

where $Q(s, a; \theta)$ is the estimated action value of action a in state s according to current parameters θ , and where θ^- is a more stable periodic copy of these parameters [cf. Mnih et al., 2015, van Hasselt et al., 2016, for more details]. This is a form of Double Q-learning [van Hasselt, 2010]. We roughly tuned the main step size and the step size for the normalization to 10^{-4} . It is not straightforward to tune the unclipped version, for reasons that will become clear soon.

Figure 1b shows ℓ^2 norm of the gradient of Double DQN during learning as a function of number of training steps. The left plot corresponds to no reward clipping, middle to clipping (as per original DQN and Double DQN), and right to using Pop-Art instead of clipping. Each faint dashed lines corresponds to the median norms (where the median is taken over time) on one game. The shaded areas correspond to 50%, 90%, and 95% of games.

Without clipping the rewards, Pop-Art produces a much narrower band within which the gradients fall. Across games, 95% of median norms range over less than two orders of magnitude (roughly between 1 and 20), compared to almost four orders of magnitude for clipped Double DQN, and more than six orders of magnitude for unclipped Double DQN without Pop-Art. The wide range for the latter shows why it is impossible to find a suitable step size with neither clipping nor Pop-Art: the updates are either far too small on some games or far too large on others.

After 200M frames, we evaluated the actual scores of the best performing agent in each game on 100 episodes of up to 30 minutes of play, and then normalized by human and random scores as described by Mnih et al. [2015]. Figure 2 shows the differences in normalized scores between (clipped) Double DQN and Double DQN with Pop-Art.

The main eye-catching result is that the distribution in performance drastically changed. On some games (e.g., Gopher, Centipede) we observe dramatic improvements, while on other games (e.g., Video Pinball, Star Gunner) we see a substantial decrease. For instance, in Ms. Pac-Man the clipped Double DQN agent does not care more about ghosts than pellets, but Double DQN with Pop-Art learns to actively hunt ghosts, resulting in higher scores. Especially remarkable is the improved performance on games like Centipede and Gopher, but also notable is a game like Frostbite which went from below 50% to a near-human performance level. Raw scores can be found in the appendix.

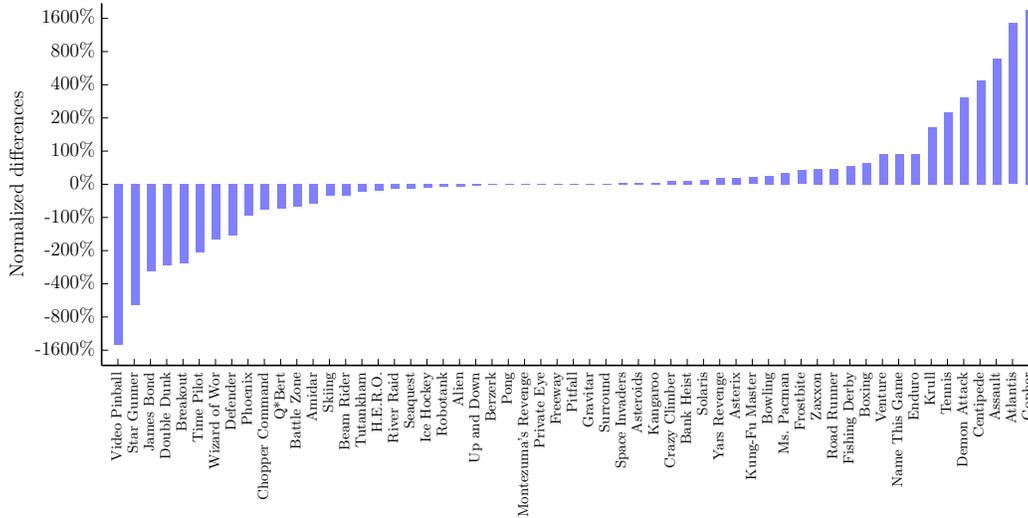


Figure 2: Differences between normalized scores for Double DQN with and without Pop-Art on 57 Atari games.

Some games fare worse with unclipped rewards because it changes the nature of the problem. For instance, in Time Pilot the Pop-Art agent learns to quickly shoot a mothership to advance to a next level of the game, obtaining many points in the process. The clipped agent instead shoots at anything that moves, ignoring the mothership. However, in the long run in this game more points are scored with the safer and more homogeneous strategy of the clipped agent. One reason for the disconnect between the seemingly qualitatively good behavior combined with lower scores is that the agents are fairly myopic: both use a discount factor of $\gamma = 0.99$, and therefore only optimize rewards that happen within a dozen or so seconds into the future.

On the whole, the results show that with Pop-Art we can successfully remove the clipping heuristic that has been present in all prior DQN variants, while retaining overall performance levels. Double DQN with Pop-Art performs slightly better than Double DQN with clipped rewards: on 32 out of 57 games performance is at least as good as clipped Double DQN and the median (+0.4%) and mean (+34%) differences are positive.

5 Discussion

We have demonstrated that Pop-Art can be used to adapt to different and non-stationary target magnitudes. This problem was perhaps not previously commonly appreciated, potentially because in deep learning it is common to tune or normalize a priori, using an existing data set. This is not as straightforward in reinforcement learning when the policy and the corresponding values may repeatedly change over time. This makes Pop-Art a promising tool for deep reinforcement learning, although it is not specific to this setting.

We saw that Pop-Art can successfully replace the clipping of rewards as done in DQN to handle the various magnitudes of the targets used in the Q-learning update. Now that the true problem is exposed to the learning algorithm we can hope to make further progress, for instance by improving the exploration [Osband et al., 2016], which can now be informed about the true unclipped rewards.

References

- S. I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998. ISSN 0899-7667.
- M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res. (JAIR)*, 47:253–279, 2013.

- M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas, and R. Munos. Increasing the action gap: New operators for reinforcement learning. In *AAAI*, 2016.
- J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.
- J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.
- G. Desjardins, K. Simonyan, R. Pascanu, and K. Kavukcuoglu. Natural neural networks. In *Advances in Neural Information Processing Systems*, pages 2062–2070, 2015.
- J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.
- S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- D. P. Kingma and J. B. Adam. A method for stochastic optimization. In *International Conference on Learning Representation*, 2015.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 05 2015.
- Y. Liang, M. C. Machado, E. Talvitie, and M. H. Bowling. State of the art control of atari games using shallow reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2016.
- J. Martens and R. B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2408–2417, 2015.
- W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.
- I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped DQN. *CoRR*, abs/1602.04621, 2016.
- F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.
- S. Ross, P. Mineiro, and J. Langford. Normalized online learning. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence*, 2013.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.
- T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.
- J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
- J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.
- T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSE: Neural Networks for Machine Learning, 2012.
- H. van Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems*, 23:2613–2621, 2010.
- H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with Double Q-learning. *AAAI*, 2016.
- Z. Wang, N. de Freitas, T. Schaul, M. Hessel, H. van Hasselt, and M. Lanctot. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, New York, NY, USA, 2016.
- C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.