Learning functions across many orders of magnitudes

Hado van Hasselt HADO@GOOGLE.COM
Arthur Guez AGUEZ@GOOGLE.COM
Matteo Hessel MTTHSS@GOOGLE.COM
David Silver DAVIDSILVER@GOOGLE.COM
Google DeepMind

arXiv:1602.07714v1 [cs.LG] 24 Feb 2016

Abstract

Learning non-linear functions can be hard when the magnitude of the target function is unknown beforehand, as most learning algorithms are not scale invariant. We propose an algorithm to adaptively normalize these targets. This is complementary to recent advances in input normalization. Importantly, the proposed method preserves the unnormalized outputs whenever the normalization is updated to avoid instability caused by non-stationarity. It can be combined with any learning algorithm and any non-linear function approximation, including the important special case of deep learning. We empirically validate the method in supervised learning and reinforcement learning and apply it to learning how to play Atari 2600 games. Previous work on applying deep learning to this domain relied on clipping the rewards to make learning in different games more homogeneous, but this uses the domain-specific knowledge that in these games counting rewards is often almost as informative as summing these. Using our adaptive normalization we can remove this heuristic without diminishing overall performance, and even improve performance on some games, such as Ms. Pac-Man and Centipede, on which previous methods did not perform well.

1. Learning with arbitrary magnitudes

Many current learning algorithms rely on a priori access to actual or sufficiently similar data to properly tune relevant hyper-parameters (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.

Concretely, we are motivated by the work by Mnih et al. (2015), which combines Q-learning (Watkins, 1989) with a deep convolutional neural network (cf. LeCun et al., 2015). The resulting deep Q network (DQN) algorithm was shown to be able to learn to play a varied set of Atari 2600 games from the Arcade Learning Environment (ALE) (Bellemare et al., 2013), by learning action values corresponding to expected sums of future rewards. The ALE was proposed as an evaluation framework to test general learning algorithms on solving many different interesting tasks, and DQN was proposed as a singular solution, using a single set of hyperparameters. However, to achieve this feat the rewards and temporal-difference errors were clipped to $[-1, 1]$ because the magnitudes and frequencies of rewards vary wildly between different games. For instance, in Pong the rewards are sparsely distributed and bounded by -1 and $+1$ while in Ms. Pac-Man eating a single ghost can yield a reward of up to $+1600$, but DQN clips the latter to $+1$ as well. This is not a satisfying solution for two reasons. The first reason is that the clipping introduces a form of domain knowledge. Most games have sparse non-zero rewards outside of $[-1, 1]$. Clipping the rewards then implies that the algorithm optimizes the frequency of rewards, rather than their sum. This is a good heuristic in many Atari games, but it does not generalize to the full setting of reinforcement learning. More importantly, the clipping changes the problem that the learning agent is solving and in some games the learned behaviour is clearly affected by this.

In this paper, we propose a method that adaptively normalizes the targets used in the learning updates. If we know that these targets are guaranteed to fall in some predetermined range, for instance $[-1, 1]$, it is much easier to find suitable hyperparameters. If the magnitudes of incoming targets can vary greatly over time, then a naive adaptive normalization can result in a highly non-stationary and unstable learning problem because it would continually change the outputs of our approximation for all inputs, thereby invalidating earlier learning. To avoid this, the proposed method includes an additional step that ensures that the outputs of the approximation are kept fixed whenever we change the normalization.

The proposed technique is not specific to the application to DQN and is applicable more generally in supervised learning and reinforcement learning. There are several reasons

normalization of the targets can be desirable. First, it is generally useful when we want a single system to be able to solve multiple different problems with varying natural magnitudes. Second, when learning a multi-variate function we can use the adaptive normalization to disentangle the natural magnitude of each target component from its relative importance in the loss function. This is particularly useful when the targets have different units, such as when we simultaneously predict signals from sensors with different modalities. Finally, adaptive scaling can help in problems that are non-stationary. For instance, this is common in reinforcement learning when the policy of behavior, and therefore the distribution and magnitude of the targets, can change repeatedly during learning.

2. Related work

Input normalization has long been recognized as important for efficient learning of non-linear approximations such as neural networks (LeCun et al., 1998). This has led to several publications about how to achieve scale-invariance on the inputs (e.g., Ross et al., 2013; Ioffe and Szegedy, 2015; Desjardins et al., 2015). On the other hand, output or target normalization does not seem to have received the same attention. In classification, normalization of the targets is not necessary. In the common supervised regression setting, where a full data set is available before we commence learning, it is straightforward and commonplace to examine the data to determine an appropriate normalization, or to pre-tune appropriate hyper-parameters. However, this assumes the data is available a priori, which is not true in the online (potentially non-stationary) regression settings that we are interested in. In this paper we focus only on target normalization do not investigate the combination and interaction of target and input normalization, except to note that these are compatible and complementary.

More generally there has been work on normalizing the whole optimization process, for instance by using natural gradients (Amari, 1998). Natural gradients make the learning invariant to reparameterizations of the function approximation, thereby avoiding many scaling issues. Unfortunately, the pure version is computationally expensive for functions with many parameters, such as deep neural networks. This is why approximations are regularly proposed. Sometimes, these approximations focus on a complementary aspect, such as input normalization in the case of Natural Neural Networks (Desjardins et al., 2015) and Batch Normalization (Ioffe and Szegedy, 2015). In other cases, these algorithms directly approximate the full natural gradient (Martens and Grosse, 2015), but then necessarily have to make trade offs in terms of accuracy to obtain computational gains. In addition, these algorithms generally remain more computationally expensive than vanilla stochastic gradients descent and are typically not invariant to rescaling the targets. In contrast, we focus only on target normalization, allowing us to find an effective and computationally efficient algorithm which can potentially be combined with other methods that take care of other aspects of the full learning problem, such as input normalization.

In a different strand of research, several algorithms have been proposed to automatically adapt the step size during learning to handle non-stationarity problems (Sutton, 1992; Mahmood et al., 2012; Schaul et al., 2013). These can also be interpreted as attempting to solve multiple problems at once, including scale invariance, speed of learning, non-stationarity, and stability. While potentially powerful, solving these combined issues with a single adaptive algorithm is harder than the problem of target normalization we set out to solve, which is perhaps why there is not a single clear winner in this category. In addition, these algorithms often come with additional assumptions, such as requiring linear function approximation (Sutton, 1992; Mahmood et al., 2012). In addition, the algorithms typically assume stochastic gradient descent updates, or are themselves specific variants thereof, such as RMSprop (Tieleman and Hinton, 2012) and Adam (Kingma and Ba, 2014), which are not themselves target-scale invariant. The target normalization we propose can be combined with any learning algorithm, and therefore is best interpreted as complementary to such methods, rather than a competitor.

3. Preliminaries

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.[1] The objective is to update the parameters $\boldsymbol{\theta}$ of a function $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^k$ such that the output $f_{\boldsymbol{\theta}}(X_t)$ is in some sense close to the corresponding target Y_t, for instance as measured by the (expected) squared difference.

A canonical and popular example of a learning algorithm is stochastic gradient descent (SGD) on a squared loss

$$l(f_{\boldsymbol{\theta}}) \;\equiv\; \frac{1}{2}\mathbb{E}\left[(f_{\boldsymbol{\theta}}(X_t) - Y_t)^\top (f_{\boldsymbol{\theta}}(X_t) - Y_t)\right] \,.$$

For a given tuple (X_t, Y_t) the update is then

$$\boldsymbol{\theta}_{t+1} \;=\; \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}}^\top l_t(f_{\boldsymbol{\theta}}) \;=\; \boldsymbol{\theta}_t + \alpha \boldsymbol{J}_t \boldsymbol{\delta}_t \,,$$

where $\alpha \in [0, 1]$ is a step size,[2] $\boldsymbol{\delta}_t = Y_t - f_{\boldsymbol{\theta}}(X_t)$ is the

[1] Inputs and outputs may have multi-dimensional structure, as matrices or higher-order tensors. Then n and k are the total number of elements. We use upper case letters to denote random inputs and outputs, regardless of their structure.

[2] For simplicity the step size is a scalar here. Our discussion generalizes to vector (component-wise) step sizes and (e.g., second-order) methods with matrix step sizes.

error, and $\boldsymbol{J}_t$ is the Jacobian defined by

$$\boldsymbol{J}_t \equiv \left(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta},1}(X_t), \ldots, \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta},k}(X_t)\right)^\top .$$

The magnitude of this update depends on the magnitudes of both the step size and the error, and it is hard to pick suitable step sizes when nothing is known about the magnitudes of the errors.

An important special case is when $f_{\boldsymbol{\theta}}$ is a neural network (McCulloch and Pitts, 1943; Widrow and Hoff, 1960; Rosenblatt, 1962; Rumelhart et al., 1986; Bishop, 1995). These networks are often trained with a form of stochastic gradient descent, with hyperparameters that interact with the scale of the targets. Especially for deep neural networks (LeCun et al., 2015; Schmidhuber, 2015) an update that is too large may harm learning, because these networks are highly non-linear and a large update may 'bump' the parameters to regions with high error.[3]

4. 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

$$\tilde{Y}_t = \boldsymbol{\Sigma}_t^{-1}(Y_t - \boldsymbol{\mu}_t) ,$$

where $\boldsymbol{\Sigma}_t$ and $\boldsymbol{\mu}_t$ are *scale* and *shift* parameters that are learned from the data. The scale matrix $\boldsymbol{\Sigma}_t$ can be dense, diagonal, or defined by a scalar σ_t as $\boldsymbol{\Sigma}_t = \sigma_t \mathbf{I}$. Similarly, the shift vector $\boldsymbol{\mu}_t$ can contain separate components, or be defined by a scalar μ_t as $\boldsymbol{\mu}_t = \mu_t \mathbf{1}$. We discuss relative benefits of each approach in Section 4.3.

We update the output of a parameterized function $\tilde{f}_{\boldsymbol{\theta}}(X_t)$ towards the normalized target $\tilde{Y}_t$. The unnormalized output for any input x can then be recovered using

$$f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu}}(x) = \boldsymbol{\Sigma} \tilde{f}_{\boldsymbol{\theta}}(x) + \boldsymbol{\mu} .$$

We call $\tilde{f}_{\boldsymbol{\theta}}$ the *normalized function* and $f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu}}$ the *unnormalized function*. The goal is, as before, for the outputs of the unnormalized function for an input X_t to be close to the corresponding unnormalized target Y_t.

At first glance it may seem we have made little progress. If we learn $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ using the same algorithm as used for the parameters $\boldsymbol{\theta}$ of the function $\tilde{f}_{\boldsymbol{\theta}}$, 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 the scale and shift to be tunable hyperparameters then setting them appropriately is not fundamentally easier than tuning other hyperparameters, such as the step size, directly.

Fortunately, there is an alternative. We propose to update $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ according to a separate objective with the aim of making the normalized targets fall approximately into some predetermined range $[-1, 1]$. Thereby, we decompose the problem of learning an appropriate scale and shift from learning the specific shape of the function.

Unless care is taken, repeated updates to the scale and shift might make learning harder rather than easier because the normalized targets become non-stationary. More importantly, whenever we adapt the scale and shift 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 we will illustrate in Section 5.

It may not be immediately obvious that we can prevent changing all outputs when updating the normalization, but we will show how this can be ensured in Section 4.1, by including a step that preserves the outputs of the function whenever we change the scale and shift.

Summarizing, the two properties that we want to simultaneously achieve are

(ART) to update scale $\boldsymbol{\Sigma}$ and shift $\boldsymbol{\mu}$ such that $\boldsymbol{\Sigma}^{-1}(Y - \boldsymbol{\mu})$ is appropriately normalized (e.g., $\in [-1, 1]$), and

(POP) to preserve the outputs of the unnormalized function whenever we change the scale and shift.

We will henceforth refer to algorithms that combine output-preserving updates and adaptive rescaling, as **Pop-Art** algorithms, which is an acronym for "Preserving Outputs Precisely, while Adaptively Rescaling Targets". Next, we first discuss how to achieve the desideratum of output preservation, which has a single elegant solution, and then discuss how to appropriately normalize.

4.1. Preserving outputs precisely

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 $\tilde{f}$ itself simultaneously. The goal is to exactly 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 internal optimization algorithm.

Without loss of generality we assume the normalized func-

[3] For linear functions, high magnitude updates are not a problem per se, as long as they are proportional to the errors. In other words, SGD on a linear function is invariant to scaling of the output. This does not hold for non-linear functions.

tion can be written as

$$\tilde{f}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(x) \equiv \mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}\,,$$

where $\tilde{g}_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ is any (non-linear) function, for instance a deep neural network. It is not uncommon for deep neural networks to end in a linear layer, and so $\tilde{g}_{\boldsymbol{\theta}}$ might be thought of as the output of the last (hidden) layer of non-linearities. Alternatively, we can always add a square linear layer to any non-linear function $\tilde{g}_{\boldsymbol{\theta}}$ to ensure this constraint, for instance initialized as $\mathbf{W}_0 = \mathbf{I}$ and $\boldsymbol{b}_0 = \mathbf{0}$.

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

Fact 1. *Consider a function $f : \mathbb{R}^n \to \mathbb{R}^k$ defined by*

$$f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu},\mathbf{W},\boldsymbol{b}}(x) \quad \equiv \quad \boldsymbol{\Sigma}\left(\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}\right) + \boldsymbol{\mu}\,,$$

where $\tilde{g}_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ is any non-linear function of $x \in \mathbb{R}^n$, $\boldsymbol{\Sigma}$ is a $k \times k$ matrix, $\boldsymbol{\mu}$ and $\boldsymbol{b}$ are k-element vectors, and $\mathbf{W}$ is a $k \times m$ matrix. Consider any change of the scale and shift parameters from $\boldsymbol{\Sigma}$ to $\boldsymbol{\Sigma}_2$ and from $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_2$, where $\boldsymbol{\Sigma}_2$ is non-singular. If we then additionally change the parameters $\mathbf{W}$ and $\boldsymbol{b}$ to $\mathbf{W}_2$ and $\boldsymbol{b}_2$, defined by

$$\mathbf{W}_2 = \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}\mathbf{W} \quad \text{and} \quad \boldsymbol{b}_2 = \boldsymbol{\Sigma}_2^{-1}\left(\boldsymbol{\Sigma}\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_2\right)\,,$$

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

$$f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu},\mathbf{W},\boldsymbol{b}}(x) = f_{\boldsymbol{\theta},\boldsymbol{\Sigma}_2,\boldsymbol{\mu}_2,\mathbf{W}_2,\boldsymbol{b}_2}(x)\,, \qquad \forall x\,.$$

This fact and later highlighted facts are proven in the appendix. For the special case of scalar scale and shift, with $\boldsymbol{\Sigma} \equiv \sigma\mathbf{I}$ and $\boldsymbol{\mu} \equiv \mu\mathbf{1}$, the updates to $\mathbf{W}$ and $\boldsymbol{b}$ become

$$\mathbf{W}_2 = (\sigma/\sigma_2)\mathbf{W} \quad \text{and} \quad \boldsymbol{b}_2 = (\sigma\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_2)/\sigma_2\,. \quad (1)$$

After updating the scale and shift we can update the output of the normalized function $\tilde{f}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(X_t)$ 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 will be made more precise in Fact 2 in Section 4.2.

Algorithm 1 is an example implementation of SGD with Pop-Art. Notice that $\mathbf{W}$ and $\boldsymbol{b}$ 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.

4.2. 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 the single-variate case. If we have data $\{(X_i, Y_i)\}_{i=1}^t$ up to some time t, we then desire

$$\sum_{i=1}^{t}(Y_i - \mu_t)/\sigma_t = 0\,, \text{ and } \quad \frac{1}{t}\sum_{i=1}^{t}(Y_i - \mu_t)^2/\sigma_t^2 = 1\,.$$

Solving for μ_t and σ_t gives

$$\mu_t = \frac{1}{t}\sum_{i=1}^{t} Y_i \quad \text{and} \quad \sigma_t = \frac{1}{t}\sum_{i=1}^{t} Y_i^2 - \mu_t^2\,. \qquad (2)$$

This can be generalized to incremental updates

$$\mu_t = (1-\beta_t)\mu_{t-1} + \beta_t Y_t \quad \text{and} \qquad (3)$$
$$\sigma_t^2 = \nu_t - \mu_t^2 \quad \text{where} \quad \nu_t = (1-\beta_t)\nu_{t-1} + \beta_t Y_t^2\,.$$

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 ensure this explicitly by requiring $\nu_t - \mu_t^2 \geq \epsilon$ with $\epsilon > 0$. For full equivalence to (2), we should use $\beta_t = 1/t$. If $\beta_t = \beta$ is constant we instead obtain exponential moving averages, allowing us to place more weight on more 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 fact holds.

Fact 2. *When using updates* (3) *to adapt the normalization parameters σ and μ, the normalized targets are bounded*

Algorithm 1 SGD with Pop-Art

```
For a given differentiable function g̃_θ, initialize θ.
Initialize W = I, b = 0, Σ = I, and μ = 0.
while learning do
  Observe input X and target Y
  Use Y to compute new scale Σ_* and new shift μ_*
  W ← Σ_*^{-1} Σ W                    (rescale W to preserve ouputs)
  b ← Σ_*^{-1}(Σb + μ − μ_*)          (rescale b)
  g ← g̃_θ(X)                          (store output of g̃_θ)
  J ← (∇g̃_{θ,1}(X), ..., ∇g̃_{θ,m}(X))^⊤   (Jacobian)
  δ̃ ← Σ_*^{-1}(Y − μ_*) − Wg − b      (normalized error)
  θ ← θ + αJW^⊤δ̃                      (update θ with SGD)
  W ← W + αδ̃g^⊤                       (update W with SGD)
  b ← b + αδ̃                          (update b with SGD)
  Σ ← Σ_*                             (update scale)
  μ ← μ_*                             (update shift)
end while
```

for all t by

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

For instance, if $\beta_t = \beta = 10^{-4}$ for all t, then Fact 2 implies that the normalized target is guaranteed to be in $(-100, 100)$. Note that Fact 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.

In the appendix we discuss others normalization updates, based on percentiles and mini-batches, and derive some interesting correspondences between all of these.

4.3. Multi-variate outputs

In many applications the targets are vectors or higher-order tensors. Then, there are at least three possibilities for the normalization, depending on how these targets interact. The appropriate choice depends on the domain.

First, we can compute combined statistics to find a global scalar scale and shift. This makes sense when when all targets have the same natural scale and units and we want to preserve relative differences. An example is learning action values in reinforcement learning.

Second, we can normalize each element of the target separately. This is useful when the outputs have differing natural scales, for instance when predicting sensory readings of different modalities. Normalizing per component makes it easier to weight each output appropriately in the desired objective, because we can detangle the relative importance of each output component from its magnitude.

Finally, we can normalize for all elements together, for instance defining as goal that the normalized target vector is distributed according to a standard multi-variate normal distribution. This is a common approach for input normalization (Ioffe and Szegedy, 2015; Desjardins et al., 2015).

In the first scenario the scale can be taken to be a scalar such that $\mathbf{\Sigma} = \sigma\mathbf{I}$. In the second scenario the scale is a vector or, equivalently, $\mathbf{\Sigma}$ is diagonal. In the third scenario, $\mathbf{\Sigma}$ can be dense. We can consider combinations of these approaches, for instance normalizing some subsets of outputs together but separately from other subsets.

In both the scalar and vector (diagonal) cases, we can treat the input to the normalization as a single stream of data, with a single scalar scale and shift. This stream of data is either combined over several components for the global scalar normalization, or it belongs to a single output in component-wise vector normalization. For clarity and conciseness, in the remainder of this paper we will focus on these cases, and leave the more general setting where $\mathbf{\Sigma}$ can be any non-singular matrix as a natural extension.

4.4. 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 $f_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(X) = \mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(X) + \boldsymbol{b}$. The update for the weight matrix $\mathbf{W}$ is

$$\mathbf{W}_t = \mathbf{W}_{t-1} + \alpha_t \boldsymbol{\delta}_t \tilde{g}_{\boldsymbol{\theta}_t}(X_t)^\top\,,$$

where $\boldsymbol{\delta}_t = Y_t - f_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(X)$ is 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.[4]

Now consider the SGD update to the parameters of $\tilde{g}_{\boldsymbol{\theta}}$,

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \alpha \boldsymbol{J}_t \mathbf{W}_{t-1}^\top \boldsymbol{\delta}_t\,,$$

where $\boldsymbol{J}_t = (\nabla g_{\boldsymbol{\theta},1}(X), \ldots, \nabla g_{\boldsymbol{\theta},m}(X))^\top$ is the Jacobian for $\tilde{g}_{\boldsymbol{\theta}}$. The magnitudes of both the weights $\mathbf{W}$ and the errors $\boldsymbol{\delta}$ depend linearly on the magnitude of the targets. This means that the magnitude of the update for $\boldsymbol{\theta}$ 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.

Fact 3. *Consider two functions defined by*

$$f_{\boldsymbol{\theta},\mathbf{\Sigma},\boldsymbol{\mu},\mathbf{W},\boldsymbol{b}}(x) = \mathbf{\Sigma}(\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}) + \boldsymbol{\mu}\,, \text{ and}$$
$$f_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(x) = \mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}\,,$$

where $\tilde{g}_{\boldsymbol{\theta}}$ is the same differentiable function in both cases, and the functions are initialized identically, using $\mathbf{\Sigma}_0 = \mathbf{I}$ and $\boldsymbol{\mu} = \mathbf{0}$, and the same initial $\boldsymbol{\theta}_0$, $\mathbf{W}_0$ and $\boldsymbol{b}_0$. Consider updating the first function using Algorithm 1 and the second using Algorithm 2. Then, for any sequence of

[4]In 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).

Algorithm 2 Normalized SGD

Given a function $f_{\boldsymbol{\theta}}(X) \equiv \mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(X) + \boldsymbol{b}$, where $\tilde{g}_{\boldsymbol{\theta}}$ is any differentiable function of $\boldsymbol{\theta}$ and X.
while learning **do**
  Observe input X and target Y
  Use Y to compute new scale $\boldsymbol{\Sigma}$
  $\boldsymbol{g} \leftarrow \tilde{g}_{\boldsymbol{\theta}}(X)$ (store output of $\tilde{g}$)
  $\boldsymbol{\delta} \leftarrow Y - \mathbf{W}\boldsymbol{g} - \boldsymbol{b}$ (compute unnormalized error)
  $\boldsymbol{J} \leftarrow (\nabla\tilde{g}_{\boldsymbol{\theta},1}(X), \ldots, \nabla\tilde{g}_{\boldsymbol{\theta},m}(X))^\top$ (compute Jacobian)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{J}(\boldsymbol{\Sigma}^{-1}\mathbf{W})^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\delta}$ (update $\boldsymbol{\theta}$ with scaled SGD)
  $\mathbf{W} \leftarrow \mathbf{W} + \alpha\boldsymbol{\delta}\boldsymbol{g}^\top$ (update $\mathbf{W}$ with SGD)
  $\boldsymbol{b} \leftarrow \boldsymbol{b} + \alpha\boldsymbol{\delta}$ (update $\boldsymbol{b}$ with SGD)
end while

non-singular scales $\{\boldsymbol{\Sigma}_t\}_{t=1}^{\infty}$ and shifts $\{\boldsymbol{\mu}_t\}_{t=1}^{\infty}$, the algorithms are equivalent in the sense that 1) the sequences $\{\boldsymbol{\theta}_t\}_{t=0}^{\infty}$ are identical, 2) the outputs of the functions are identical, for any input.

This fact shows that Algorithms 1 and 2 are fully equivalent, allowing us to make other interesting comparisons. Consider RMSprop (Tieleman and Hinton, 2012), AdaGrag (Duchi et al., 2011), and Adam (Kingma and Ba, 2014), all of which divide the updates by the square root of a recency-weighted empirical second moment such that

$$\boldsymbol{\theta}_t \approx \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}_t}^\top l_t(\boldsymbol{\theta}_t) / \sqrt{\mathbb{E}[(\nabla_{\boldsymbol{\theta}_t}^\top l_t(\boldsymbol{\theta}_t))^2]} \,,$$

where the division is per component. This is somewhat similar to the normalization discussed above, which also uses a trace of the recent empirical second moment. However, then Pop-Art can be interpreted as not applying the scaling to the updates at the top layer of the network but only to the gradient passing through the top layer into the rest of the network, as can be seen by its equivalence to Algorithm 2. This means the updates in the top layer are allowed to be big if the targets are big, whereas the aforementioned algorithms all scale down these updates. The effect is that RMSprop, AdaGrad, and Adam will slow down the updates to weights in the top layer when the magnitudes of the errors are bigger, which means these algorithms are not scale-invariant, while Pop-Art is not similarly affected. This argument is even more important when the non-linearities of the last hidden layer are bounded, for instance when using the $\tanh$ function, because then the top layer is the only layer that can scale the output of the network up to the appropriate range.

The above discussion compares RMSprop, AdaGrad, and Adam to the combination of SGD with Pop-Art. However, Pop-Art is agnostic to the choice of optimization algorithm and it is possible, and straightforward, to combine it with these other methods.

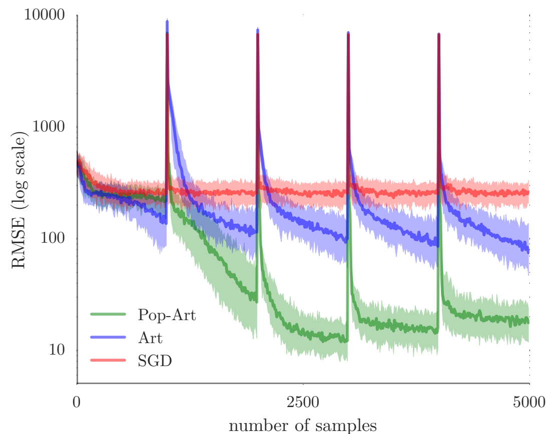

Figure 1. Median RMSE on binary regression for SGD without normalization (**red**), with normalization but without preserving outputs (**blue**, labeled 'Art'), and with Pop-Art (**green**). Area shaded 10th and 90th percentiles.

5. Binary regression experiments

In our first experiment, we analyze the effect of rare events in online learning, when infrequently a target is observed with a much higher magnitude. Such events are expected to occur, for instance, when we try to learn from a sensor that is mostly white noise but sometimes captures an actual signal. Additionally, such settings occur in reinforcement learning with sparse rewards.

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 setup is as follows. 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 one much larger target by giving as input the binary representation of $2^{16} - 1$ (i.e., all 16 inputs are 1) and as target $2^{16} - 1 = 65,535$.

The approximating function approximation 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 for extensive sweeps over hyper-parameters, to avoid inadvertent bias towards any of the algorithms by the way we tune these. The step sizes α for SGD and β for the normalization are both tuned by a grid search over $\{10^{-5}, 10^{-4.5}, \ldots, 10^{-1}, 10^{-0.5}, 1\}$.

Figure 1 shows results for online learning on 5000 samples. The x-axis shows the number of samples observed so far, and the y-axis shows the root mean squared error (RMSE) on a log scale for the current sample, before up-

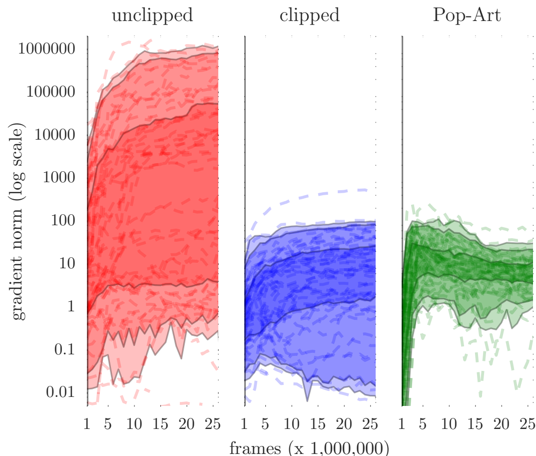

Figure 2. ℓ^2 gradient norms for DQN during learning on 57 Atari games with actual unclipped rewards (left, **red**), clipped rewards (middle, **blue**), and using Pop-Art (right, **green**) instead of clipping. Shaded areas correspond to 95%, 90% and 50% of games.

dating the function (so this is a test error, not a train error). The solid line is the median of 50 repetitions of the experiment. The shaded region covers the 10th to 90th percentiles over all repetitions. 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 from the larger targets, but this slows learning on the other samples; the error curve is almost flat in between spikes. SGD with adaptive normalization (labeled 'Art') can use a higher step size and learns faster, but has high error after the spikes because the changing scale also changes the outputs of the smaller inputs, increasing the errors on these. In comparison, Pop-Art performs much better. Both Art and Pop-Art used a step size of $\alpha = 10^{-2.5}$, but Pop-Art could exploit a much faster rate for the statistics ($\beta = 10^{-0.5}$ for Pop-Art, 10^{-4} for Art). The faster tracking of statistics protects Pop-Art from the much higher spikes, while the output preservation avoids invalidating the outputs at smaller magnitudes. We also ran experiments with RMSprop but chose to leave these out of the figure because the results were very similar to SGD.

6. 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.

As a successful example, Mnih et al. (2015) used a form of Q-learning (Watkins, 1989) together with a deep neural network (LeCun et al., 2015) in an algorithm called DQN. Impressively, DQN learned to play many games using a single set of hyper-parameters. However, to handle the different reward structures with a single system, DQN clips all rewards 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 it introduces a form of domain knowledge that optimizing the number of rewards is a good proxy for optimizing the sum of rewards. In addition, it 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. The hypotheses are 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 the lost performance by then applying Pop-Art, thereby opening up the real problem as defined by the real rewards and reducing the dependence on a domain-specific heuristic without suffering a large performance loss.

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

$$Y_t = R_{t+1} + \gamma Q(S_t, \operatorname*{argmax}_a Q(S_t, a; \boldsymbol{\theta}); \boldsymbol{\theta}^-) , \qquad (4)$$

where $Q(s, a; \boldsymbol{\theta})$ is the estimated action value of action a in state s according to current parameters $\boldsymbol{\theta}$, and where $\boldsymbol{\theta}^-$ 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, 2011). 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 2 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

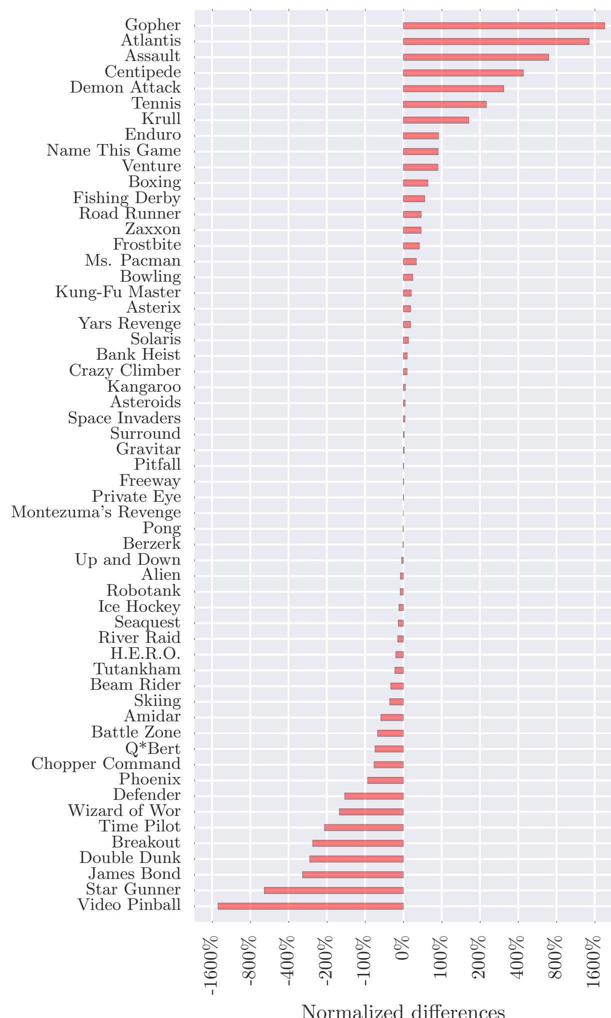

Figure 3. Differences between normalized scores for Double DQN with and without Pop-Art on 57 Atari games.

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 3 shows the differences in normalized scores between (clipped) Double DQN and Double DQN with Pop-Art. There is a slight gain for using Pop-Art: on 32 out of 57 games Double DQN with Pop-Art is at least as good as Double DQN and the median (+0.4%) and mean (+34%) differences are both positive. This means we have successfully removed the domain-dependent arbitrary clipping without suffering an overall loss in performance.

However, 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.

Performance on some games improves greatly because that uncovering the true problem immediately results in better behavior. For instance, in Ms. Pac-Man the original DQN and Double DQN agents do not care more about ghosts than pellets, but Double DQN with Pop-Art learned to actively hunt ghosts. This results in higher scores. Especially remarkable is the improved performance on games like Centipede and Gopher, but also notable are games like Frostbite which went from below 50% to a near-human performance level. Raw scores can be found in the appendix.

On the other end of the spectrum, we see that some games suffer from the unclipping of the rewards. The reason is similar to the reason performance increases on others. Consider Breakout as an example. In this game the aim is to hit blocks, where higher-placed blocks are worth more than lower-placed blocks. The original DQN was blind to this fact, and would happily and diligently work at removing many blocks regardless of position. The unclipped agent is greedier, in a sense, and tries to get to the higher-valued blocks quickly. This strategy is riskier, however, and therefore the unclipped agent loses more lives in the process and ultimately performs worse. One potential reason the agent adopts this riskier strategy in pursuit of higher rewards is that the learning process is fairly myopic: the discount used in target (4) was $\gamma = 0.99$, as per DQN and Double DQN, which along with the knowledge that the agent takes 15 actions per second translates into a horizon of perhaps a dozen seconds.

In other words, we see that in several games the heuristic of optimizing the number of rewards, rather than their sum, actually makes the learning problem easier. However, in order to make real progress, for instance when we consider how to efficiently explore to get to rare high rewards, ultimately we are going to have to consider the real rewards of the domains. These results show that Pop-Art can be a useful tool for this.

7. Discussion

We have demonstrated that Pop-Art methods can be used to adapt to different and non-stationary target magnitudes. As discussed above, this seems especially useful for the combination with deep reinforcement learning, although Pop-Art is not specific to this setting.

We saw that Pop-Art can successfully replace the clipping of rewards as done by 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 combining Pop-Art with other recent advances, such as dueling architectures (Wang et al., 2015) and prioritized replay (Schaul et al., 2015). In addition, it is important to investigate methods to escape from the sub-optimal policies these algorithms currently seems to get stuck in, for which we need better exploration. Fortunately, such exploration can now be informed by the true problem we aim to solve.

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.

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.

C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

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.

B. Efron. Regression percentiles using asymmetric squared error loss. *Statistica Sinica*, 1(1):93–125, 1991.

S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

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. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

H. J. Kushner and G. Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

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.

A. R. Mahmood, R. S. Sutton, T. Degris, and P. M. Pilarski. Tuning-free step-size adaptation. In *Proceedings of the IEEE International Conference on Acoustics*, 2012.

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.

W. K. Newey and J. L. Powell. Asymmetric least squares estimation and testing. *Econometrica: Journal of the Econometric Society*, pages 819–847, 1987.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

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, S. Zhang, and Y. LeCun. No More Pesky Learning Rates. In *International Conference on Machine Learning (ICML)*, 2013.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

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.

R. S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 171–176. MIT Press, 1992.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with Double Q-learning. *AAAI*, 2016.

H. P. van Hasselt. *Insights in Reinforcement Learning*. PhD thesis, Utrecht University, 2011.

Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

B. Widrow and M. E. Hoff. Adaptive switching circuits. *1960 IRE WESCON Convention Record*, 4:96–104, 1960. New York: IRE. Reprinted in Anderson and Rosenfeld [1988].

Appendix

In this appendix, we introduce and analyze several extensions and variations, including normalizing based on percentiles or minibatches. Additionally, we prove all facts in the main text and the appendix.

Experiment setup

For the experiments described in Section 6, we closely followed the setup described in Mnih et al. (2015) and van Hasselt et al. (2016). In particular, the Double DQN algorithm is identical to that described by van Hasselt et al. The shown results were obtained by running the trained agent for 30 minutes of simulated play (or 108,000 frames). This was repeated 100 times, where diversity over different runs was ensured by a small probability of exploration on each step (ϵ-greedy exploration with $\epsilon = 0.01$), as well as by performing up to 30 'no-op' actions, as also used and described by Mnih et al. In summary, the evaluation setup was the same as used by Mnih et al., except that we allowed more evaluation time per game (30 minutes instead of 5 minutes), as also used by Wang et al. (2015).

The results in Figure 3 were obtained by normalizing the raw scores by first subtracting the score by a random agent, and then dividing by the absolute difference between human and random agents, such that

$$\text{score}^{\text{normalized}} \equiv \frac{\text{score}^{\text{agent}} - \text{score}^{\text{random}}}{|\text{score}^{\text{human}} - \text{score}^{\text{random}}|} \,.$$

The raw scores are given below, in Table 1.

Generalizing normalization by variance

We can change the variance of the normalized targets to influence the magnitudes of the updates. For a desired standard deviation of $s > 0$, we can use

$$\sigma_t = \frac{\sqrt{\nu_t - \mu_t^2}}{s} \,,$$

with the updates for ν_t and μ_t as normal. It is straightforward to show that then a generalization of Fact 2 holds with a bound of

$$-s\sqrt{\frac{1-\beta_t}{\beta_t}} \leq \frac{Y_t - \mu_t}{\sigma_t} \leq s\sqrt{\frac{1-\beta_t}{\beta_t}} \,.$$

This additional parameter is for instance useful when we desire fast tracking in non-stationary problems. We then want a large step size α, but without risking overly large updates.

The new parameter s may seem superfluous because increasing the normalization step size β also reduces the hard bounds on the normalized targets. However, β additionally influences the distribution of the normalized targets. The histograms in the left-most plot in Figure 4 show what happens when we try to limit the magnitudes using only β. The red histogram shows normalized targets where the unnormalized targets come from a normal distribution, shown in blue. The normalized targets are contained in $[-1, 1]$, but the distribution is very non-normal even though the actual targets are normal. Conversely, the red histogram in the middle plot shows that the distribution remains approximately normal if we instead use s to reduce the magnitudes. The right plot shows the effect on the variance of normalized targets for either approach. When we change β while keeping $s = 1$ fixed, the variance of the normalized targets can drop far below the desired variance of one (magenta curve). When we use change s while keeping $\beta = 0.01$ fixed, the variance remains predictably at approximately s

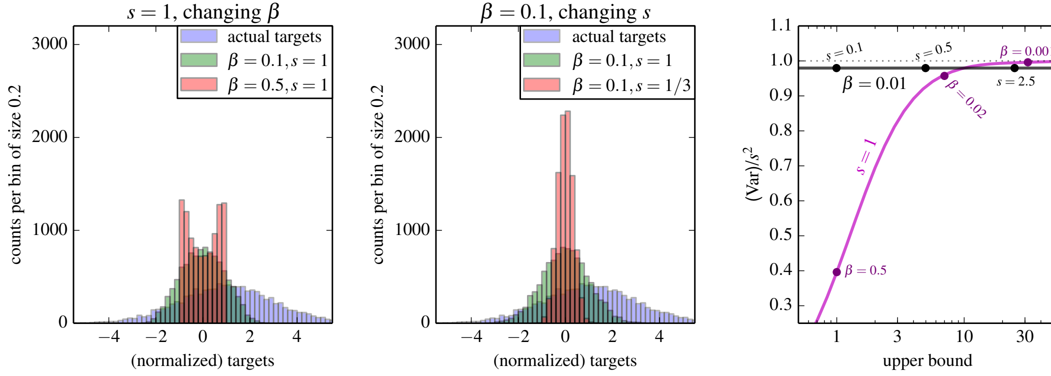

Figure 4. The **left** plot shows histograms for 10,000 normally distributed targets with mean 1 and standard deviation 2 (**blue**) and for normalized targets for $\beta = 0.1$ (**green**) and $\beta = 0.5$ (**red**). The **middle** plot shows the same histograms, except that the histogram for $\beta = 0.5$ and $s = 1$ is replaced by a histogram for $\beta = 0.1$ and $s = 1/3$ (**red**). The **right** plot shows the variance of the normalized targets as a function of the upper bound $s\sqrt{(1-\beta)/\beta}$ when we either change β while keeping $s = 1$ fixed (**magenta** curve) or we change s while keeping $\beta = 0.01$ fixed (**black** straight line).

(black line). The difference in behavior of the resulting normalization demonstrates that s gives us a potentially useful additional degree of freedom.

Sometimes, we can simply roll the additional scaling s into the step size, such that without loss of generality we can use $s = 1$ and decrease the step size to avoid overly large updates. However, sometimes it is easier to separate the magnitude of the targets, as influenced by s, from the magnitude of the updates, for instance when using an adaptive step-size algorithm. In addition, the introduction of an explicit scaling s allows us to make some interesting connections to normalization by percentiles, in the next section.

Adaptive normalization by percentiles

Instead of normalizing by mean and variance, we can normalize such that a given ratio p of normalized targets is inside the predetermined interval. The per-output objective is then

$$\mathrm{P}\left(\frac{Y-\mu}{\sigma} \in [-1, 1]\right) = p\,.$$

For normally distributed targets, there is a direct correspondence to normalizing by means and variance.

Fact 4. *If scalar targets $\{Y_t\}_{t=1}^{\infty}$ are distributed according to a normal distribution with arbitrary finite mean and variance, then the objective $P((Y-\mu)/\sigma \in [-1,1]) = p$ is equivalent to the joint objective $\mathbb{E}[Y-\mu] = 0$ and $\mathbb{E}\left[\sigma^{-2}(Y-\mu)^2\right] = s^2$ with*

$$p \;=\; \mathit{erf}\left(\frac{1}{\sqrt{2}s}\right)\,.$$

For example, percentiles of $p = 0.99$ and $p = 0.95$ correspond to $s \approx 0.4$ and $s \approx 0.5$, respecticely. Conversely, $s = 1$ corresponds to $p \approx 0.68$. The fact only applies when the targets are normal. For other distributions the two forms of normalization differ even in terms of their objectives.

We now discuss a concrete algorithm to obtain normalization by percentiles. Let $Y_t^{(n)}$ denote order statistics of the targets up to time t,[5] such that $Y_t^{(1)} = \min_i\{Y_i\}_{i=1}^t$, $Y_t^{(t)} = \max_i\{Y_i\}_{i=1}^t$, and $Y_t^{((t+1)/2)} = \mathrm{median}_i\{Y_i\}_{i=1}^t$. For notational simplicity, define $n^+ \equiv \frac{t+1}{2} + p\frac{t-1}{2}$ and $n^- \equiv \frac{t+1}{2} - p\frac{t-1}{2}$. Then, for data up to time t, the goal is

$$\frac{Y_t^{(n^+)} - \mu_t}{\sigma_t} = -1\,, \qquad \text{and} \qquad \frac{Y_t^{(n^-)} - \mu_t}{\sigma_t} = 1\,.$$

Solving for σ_t and μ_t gives

$$\mu_t = \frac{1}{2}\left(Y_t^{(n^+)} + Y_t^{(n^-)}\right)\,, \qquad \text{and}$$
$$\sigma_t = \frac{1}{2}\left(Y_t^{(n^+)} - Y_t^{(n^-)}\right)\,.$$

In the special case where $p = 1$ we get $\mu_t = \frac{1}{2}(\max_i Y_i + \min_i Y_i)$ and $\sigma_t = \frac{1}{2}(\max_i Y_i - \min_i Y_i)$. We are then guaranteed that all normalized targets fall in $[-1, 1]$, but this could result in an overly conservative normalization that is sensitive to outliers and may reduce the overall magnitude of the updates too far. In other words, learning will then be safe in the sense that no updates will be too big, but it may be slow because many updates may be very small. In general it is probably typically better to use a ratio $p < 1$.

Exact order statistics are hard to compute online, because we would need to store all previous targets. To obtain

[5] For non-integer x we can define $Y^{(x)}$ by either rounding x to an integer or, perhaps more appropriately, by linear interpolation between the values for the nearest integers.

more memory-efficient online updates for percentiles we can store two values $y_t^{\min}$ and $y_t^{\max}$, which should eventually have the property that a proportion of $(1-p)/2$ values is larger than $y_t^{\max}$ and a proportion of $(1-p)/2$ values is smaller than $y_t^{\min}$, such that

$$P\left(Y > y_t^{\max}\right) = P\left(Y < y_t^{\min}\right) = (1-p)/2\,. \quad (5)$$

This can be achieved asymptotically by updating $y_t^{\min}$ and $y_t^{\max}$ according to

$$y_t^{\max} = y_{t-1}^{\max} + \beta_t\left(\mathcal{I}(Y_t > y_{t-1}^{\max}) - \frac{1-p}{2}\right) \quad \text{and} \quad (6)$$

$$y_t^{\min} = y_{t-1}^{\min} - \beta_t\left(\mathcal{I}(Y_t < y_{t-1}^{\min}) - \frac{1-p}{2}\right), \quad (7)$$

where the indicator function $\mathcal{I}(\cdot)$ is equal to one when its argument is true and equal to zero otherwise.

Fact 5. *If $\sum_{t=1}^{\infty}\beta_t$ and $\sum_{t=1}^{\infty}\beta_t^2$, and the distribution of targets is stationary, then the updates in* (6) *converge to values such that* (5) *holds.*

If the step size β_t is too small it will take long for the updates to converge to appropriate values. In practice, it might be better to let the magnitude of the steps depend on the actual errors, such that the update takes the form of an asymmetrical least-squares update (Newey and Powell, 1987; Efron, 1991).

Online learning with minibatches

Online normalization by mean and variance with minibatches $\{Y_{t,1}, \ldots, Y_{t,B}\}$ of size B can be achieved by using the updates

$$\mu_t = (1-\beta_t)\mu_{t-1} + \beta_t \frac{1}{B}\sum_{b=1}^{B} Y_{t,b}\ , \text{ and}$$

$$\sigma_t = \frac{\sqrt{\nu_t - \mu_t^2}}{s}\ , \text{ where}$$

$$\nu_t = (1-\beta_t)\nu_{t-1} + \beta_t \frac{1}{B}\sum_{b=1}^{B} Y_{t,b}^2\,.$$

Another interesting possibility is to update $y_t^{\min}$ and $y_t^{\max}$ towards the extremes of the minibatch such that

$$y_t^{\min} = (1-\beta_t)y_{t-1}^{\min} + \beta_t \text{min}_b Y_{t,b}\,, \quad \text{and} \quad (8)$$
$$y_t^{\max} = (1-\beta_t)y_{t-1}^{\max} + \beta_t \text{max}_b Y_{t,b}\,,$$

and then use

$$\mu_t = \frac{1}{2}(y_t^{\max} + y_t^{\min})\,, \qquad \text{and} \qquad \sigma_t = \frac{1}{2}(y_t^{\max} - y_t^{\min})\,.$$

The statistics of this normalization depend on the size of the minibatches, and there is an interesting correspondence to normalization by percentiles.

Fact 6. *Consider minibatches $\{\{Y_{t,1}, \ldots, Y_{t,B}\}\}_{t=1}^{\infty}$ of size $B \geq 2$ whose elements are drawn i.i.d. from a uniform distribution with support on $[a, b]$. If $\sum_t \beta_t = \infty$ and $\sum_t \beta_t^2 < \infty$, then in the limit the updates* (8) *converge to values such that* (5) *holds, with $p = (B-1)/(B+1)$.*

This fact connects the online minibatch updates (8) to normalization by percentiles. For instance, a minibatch size of $B = 20$ would correspond roughly to online percentile updates with $p = 19/21 \approx 0.9$ and, by Fact 4, to a normalization by mean and variance with a $s \approx 0.6$. These different normalizations are not strictly equivalent, but may behave similarly in practice.

Fact 6 quantifies an interesting correspondence between minibatch updates and normalizing by percentiles. Although the fact as stated holds only for uniform targets, the proportion of normalized targets in the interval $[-1, 1]$ more generally becomes larger when we increase the minibatch size, just as when we increase p or decrease s, potentially resulting in better robustness to outliers at the possible expense of slower learning.

A note on initialization

When using constant step sizes it is useful to be aware of the start of learning, to trust the data rather than arbitrary initial values. This can be done by using a step size as defined in the following fact.

Fact 7. *Consider a recency-weighted running average $\bar{z}_t$ updated from a stream of data $\{Z_t\}_{t=1}^{\infty}$ using $\bar{z}_t = (1-\beta_t)\bar{z}_{t-1} + \beta_t Z_t$, with β_t defined by*

$$\beta_t = \beta(1-(1-\beta)^t)^{-1}\,. \quad (9)$$

Then 1) the relative weights of the data in Z_t are the same as when using a constant step size β, and 2) the estimate $\bar{z}_t$ does not depend on the initial value $\bar{z}_0$.

A similar result was derived to remove the effect of the initialization of certain parameters by Kingma and Ba (2014) for a stochastic optimization algorithm called Adam. In that work, the initial values are assumed to be zero and a standard exponentially weighted average is explicitly computed and stored, and then divided by a term analogous to $1-(1-\beta)^t$. The step size (9) corrects for any initialization in place, without storing auxiliary variables, but for the rest the method and its motivation are very similar.

Alternatively, it is possible to initialize the normalization safely, by choosing a scale that is relatively high initially. This can be beneficial when at first the targets are relatively small and noisy. If we would then use the step size in (9), the updates would treat these initial observations as important, and would try to fit our approximating function to the noise. A high initialization (e.g., $\nu_0 = 10^4$ or $\nu_0 = 10^6$)

would instead reduce the effect of the first targets on the learning updates, and would instead use these only to find an appropriate normalization. Only after finding this normalization the actual learning would then commence.

Deep Pop-Art

Sometimes it makes sense to apply the normalization not to the output of the network, but at a lower level. For instance, the ith output of a neural network with a soft-max on top can be written

$$\tilde{f}_{\boldsymbol{\theta},i}(X) = \frac{\mathrm{e}^{[\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(X)+\boldsymbol{b}]_i}}{\sum_j \mathrm{e}^{[\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(X)+\boldsymbol{b}]_j}}\,,$$

where $\mathbf{W}$ is the weight matrix of the last linear layer before the soft-max. The actual outputs are already normalized by using the soft-max, but the outputs $\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(X) + \boldsymbol{b}$ of the layer below the soft-max may still benefit from normalization. To determine the targets to be normalized, we can either back-propagate the gradient of our loss through the soft-max or invert the function.

More generally, we can consider applying normalization at any level of a hierarchical non-linear function. This seems a promising way to counteract undesirable characteristics of back-propagating gradients, such as vanishing or exploding gradients (Hochreiter, 1998).

In addition, normalizing gradients further down in a network can provide a straightforward way to combine gradients from different sources in more complex network graphs than a standard feedforward multi-layer network. First, the normalization allows us to normalize the gradient from each source separately before merging gradients, thereby avoiding one source to fully drown out any others and allowing us to weight the gradients by actual relative importance, rather than implicitly relying on the current magnitude of each as a proxy for this. Second, the normalization can prevent undesirably large gradients when many gradients come together at one point of the graph, by normalizing again after merging gradients.

Proofs

Fact 1. *Consider a function $f : \mathbb{R}^n \to \mathbb{R}^k$ defined by*

$$f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu},\mathbf{W},\boldsymbol{b}}(x) \quad \equiv \quad \boldsymbol{\Sigma}\left(\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}\right) + \boldsymbol{\mu}\,,$$

where $\tilde{g}_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ is any non-linear function of $x \in \mathbb{R}^n$, $\boldsymbol{\Sigma}$ is a $k \times k$ matrix, $\boldsymbol{\mu}$ and $\boldsymbol{b}$ are k-element vectors, and $\mathbf{W}$ is a $k \times m$ matrix. Consider any change of the scale and shift parameters from $\boldsymbol{\Sigma}$ to $\boldsymbol{\Sigma}_2$ and from $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_2$, where $\boldsymbol{\Sigma}_2$ is non-singular. If we then additionally change the parameters $\mathbf{W}$ and $\boldsymbol{b}$ to $\mathbf{W}_2$ and $\boldsymbol{b}_2$, defined by

$$\mathbf{W}_2 = \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}\mathbf{W} \quad \textit{and} \quad \boldsymbol{b}_2 = \boldsymbol{\Sigma}_2^{-1}\left(\boldsymbol{\Sigma}\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_2\right)\,,$$

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

$$f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu},\mathbf{W},\boldsymbol{b}}(x) = f_{\boldsymbol{\theta},\boldsymbol{\Sigma}_2,\boldsymbol{\mu}_2,\mathbf{W}_2,\boldsymbol{b}_2}(x)\,, \qquad \forall x\,.$$

Proof. The stated result follows from

$$\begin{aligned}
&f_{\boldsymbol{\theta},\boldsymbol{\Sigma}_2,\boldsymbol{\mu}_2,\mathbf{W}_2,\boldsymbol{b}_2}(x)\\
&= \boldsymbol{\Sigma}_2\tilde{f}_{\boldsymbol{\theta},\mathbf{W}_2,\boldsymbol{b}_2}(x) + \boldsymbol{\mu}_2\\
&= \boldsymbol{\Sigma}_2\left(\mathbf{W}_2\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}_2\right) + \boldsymbol{\mu}_2\\
&= \boldsymbol{\Sigma}_2\left(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{\Sigma}_2^{-1}\left(\boldsymbol{\Sigma}\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_2\right)\right) + \boldsymbol{\mu}_2\\
&= \left(\boldsymbol{\Sigma}\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{\Sigma}\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_2\right) + \boldsymbol{\mu}_2\\
&= \boldsymbol{\Sigma}\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{\Sigma}\boldsymbol{b} + \boldsymbol{\mu}\\
&= \boldsymbol{\Sigma}\tilde{f}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(x) + \boldsymbol{\mu}\\
&= f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu},\mathbf{W},\boldsymbol{b}}(x)\,. \qquad \square
\end{aligned}$$

Fact 2. *When using updates* (3) *to adapt the normalization parameters σ and μ, the normalized target $\sigma_t^{-1}(Y_t - \mu_t)$ is bounded for all t by*

$$-\sqrt{\frac{1-\beta_t}{\beta_t}} \leq \frac{Y_t - \mu_t}{\sigma_t} \leq \sqrt{\frac{1-\beta_t}{\beta_t}}\,.$$

Proof.

$$\begin{aligned}
&\left(\frac{Y_t - \mu_t}{\sigma_t}\right)^2\\
&= \left(\frac{Y_t - (1-\beta_t)\mu_{t-1} - \beta_t Y_t}{\sigma_t}\right)^2\\
&= \frac{(1-\beta_t)^2(Y_t - \mu_{t-1})^2}{\nu_t - \mu_t^2}\\
&= \frac{(1-\beta_t)^2(Y_t - \mu_{t-1})^2}{(1-\beta_t)\nu_{t-1} + \beta_t Y_t^2 - \left((1-\beta_t)\mu_{t-1} + \beta_t Y_t\right)^2}\\
&= \frac{(1-\beta_t)^2(Y_t - \mu_{t-1})^2}{(1-\beta_t)\left(\nu_{t-1} + \beta_t Y_t^2 - (1-\beta_t)\mu_{t-1}^2 - 2\beta_t\mu_{t-1}Y_t\right)}\\
&= \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\nu_{t-1} + \beta_t Y_t^2 - (1-\beta_t)\mu_{t-1}^2 - 2\beta_t\mu_{t-1}Y_t}\\
&\leq \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\mu_{t-1}^2 + \beta_t Y_t^2 - (1-\beta_t)\mu_{t-1}^2 - 2\beta_t\mu_{t-1}Y_t}\\
&= \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\beta_t Y_t^2 + \beta_t\mu_{t-1}^2 - 2\beta_t\mu_{t-1}Y_t}\\
&= \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\beta_t(Y_t - \mu_{t-1})^2}\\
&= \frac{(1-\beta_t)}{\beta_t}\,,
\end{aligned}$$

The inequality follows from the fact that $\nu_{t-1} \geq \mu_{t-1}^2$. $\square$

Fact 3. *Consider two functions defined by*

$$f_{\boldsymbol{\theta},\boldsymbol{\Sigma},\boldsymbol{\mu},\mathbf{W},\boldsymbol{b}}(x) = \boldsymbol{\Sigma}(\mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}) + \boldsymbol{\mu}\,, \text{ and}$$
$$f_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(x) = \mathbf{W}\tilde{g}_{\boldsymbol{\theta}}(x) + \boldsymbol{b}\,,$$

where $\tilde{g}_{\boldsymbol{\theta}}$ is the same differentiable function in both cases, and the functions are initialized identically, using $\boldsymbol{\Sigma}_0 = \mathbf{I}$ and $\boldsymbol{\mu} = \mathbf{0}$, and the same initial $\boldsymbol{\theta}_0$, $\mathbf{W}_0$ and $\boldsymbol{b}_0$. Consider updating the first function using Algorithm 1 and the second using Algorithm 2. Then, for any sequence of non-singular scales $\{\boldsymbol{\Sigma}_t\}_{t=1}^{\infty}$ and shifts $\{\boldsymbol{\mu}_t\}_{t=1}^{\infty}$, the algorithms are equivalent in the sense that 1) the sequences $\{\boldsymbol{\theta}_t\}_{t=0}^{\infty}$ are identical, 2) the outputs of the functions are identical, for any input.

Proof. Let $\boldsymbol{\theta}_t^1$ and $\boldsymbol{\theta}_t^2$ denote the parameters of $\tilde{g}_{\boldsymbol{\theta}}$ for Algorithms 1 and 2, respectively. Similarly, let $\mathbf{W}^1$ and $\boldsymbol{b}^1$ be parameters of the first function, while $\mathbf{W}^2$ and $\boldsymbol{b}^2$ are parameters of the second function. It is enough to show that single updates of both Algorithms 1 and 2 from the same starting points have equivalent results. That is, if

$$\boldsymbol{\theta}_{t-1}^2 = \boldsymbol{\theta}_{t-1}^1\,, \text{ and}$$
$$f_{\boldsymbol{\theta}_{t-1}^2,\mathbf{W}_{t-1}^2,\boldsymbol{b}_{t-1}^2}(x) = f_{\boldsymbol{\theta}_{t-1}^1,\boldsymbol{\Sigma}_{t-1},\boldsymbol{\mu}_{t-1},\mathbf{W}_{t-1}^1,\boldsymbol{b}_{t-1}^1}(x)\,,$$

then it must follow that

$$\boldsymbol{\theta}_t^2 = \boldsymbol{\theta}_t^1\,, \text{ and}$$
$$f_{\boldsymbol{\theta}_t^2,\mathbf{W}_t^2,\boldsymbol{b}_t^2}(x) = f_{\boldsymbol{\theta}_t^1,\boldsymbol{\Sigma}_t,\boldsymbol{\mu}_t,\mathbf{W}_t^1,\boldsymbol{b}_t^1}(x)\,,$$

where the quantities $\boldsymbol{\theta}^2$, $\mathbf{W}^2$, and $\boldsymbol{b}^2$ are updated with Algorithm 2 and quantities $\boldsymbol{\theta}^1$, $\mathbf{W}^1$, and $\boldsymbol{b}^1$ are updated with Algorithm 1. We do not require $\mathbf{W}_t^2 = \mathbf{W}_t^1$ or $\boldsymbol{b}_t^2 = \boldsymbol{b}_t^1$, and indeed these quantities will generally differ.

We use the shorthands f_t^1 and f_t^2 for the first and second function, respectively. First, we show that $\mathbf{W}_t^1 = \boldsymbol{\Sigma}_t^{-1}\mathbf{W}_t^2$, for all t. For $t = 0$, this holds trivially because $\mathbf{W}_0^1 = \mathbf{W}_0^2 = \mathbf{W}_0$, and $\boldsymbol{\Sigma}_0 = \boldsymbol{I}$. Now assume that $\mathbf{W}_{t-1}^1 = \boldsymbol{\Sigma}_{t-1}^{-1}\mathbf{W}_{t-1}^2$. Let $\delta_t = Y_t - f_t^1(X_t)$ be the unnormalized error at time t. Then, Algorithm 1 results in

$$\begin{aligned}
\mathbf{W}_t^1 &= \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}_{t-1}\mathbf{W}_{t-1}^1 + \alpha\boldsymbol{\Sigma}_t^{-1}\delta_t g_{\boldsymbol{\theta}_{t-1}}(X_t)^\top \\
&= \boldsymbol{\Sigma}_t^{-1}\left(\boldsymbol{\Sigma}_{t-1}\mathbf{W}_{t-1}^1 + \alpha\delta_t g_{\boldsymbol{\theta}_{t-1}}(X_t)^\top\right) \\
&= \boldsymbol{\Sigma}_t^{-1}\left(\mathbf{W}_{t-1}^2 + \alpha\delta_t g_{\boldsymbol{\theta}_{t-1}}(X_t)^\top\right) \\
&= \boldsymbol{\Sigma}_t^{-1}\mathbf{W}_t^2\,.
\end{aligned}$$

Similarly, $\boldsymbol{b}_0^1 = \boldsymbol{\Sigma}_0^{-1}(\boldsymbol{b}_0^2 - \mu_0)$ and if $\boldsymbol{b}_{t-1}^1 = \boldsymbol{\Sigma}_{t-1}^{-1}(\boldsymbol{b}_{t-1}^2 - \mu_{t-1})$ then

$$\begin{aligned}
\boldsymbol{b}_t^1 &= \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{\Sigma}_{t-1}\boldsymbol{b}_{t-1}^1 + \mu_{t-1} - \mu_t) + \alpha\boldsymbol{\Sigma}_t^{-1}\delta_t \\
&= \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{b}_{t-1}^2 - \mu_t) + \alpha\boldsymbol{\Sigma}_t^{-1}\delta_t \\
&= \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{b}_{t-1}^2 - \mu_t + \alpha\delta_t) \\
&= \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{b}_t^2 - \mu_t)\,.
\end{aligned}$$

Now, assume that $\boldsymbol{\theta}_{t-1}^1 = \boldsymbol{\theta}_{t-1}^2$. Then,

$$\begin{aligned}
\boldsymbol{\theta}_t^1 &= \boldsymbol{\theta}_{t-1}^1 + \alpha\boldsymbol{J}_t(\mathbf{W}_{t-1}^1)^\top\boldsymbol{\Sigma}_t^{-1}\delta \\
&= \boldsymbol{\theta}_{t-1}^2 + \alpha\boldsymbol{J}_t(\boldsymbol{\Sigma}_{t-1}^{-1}\mathbf{W}_{t-1}^2)^\top\boldsymbol{\Sigma}_t^{-1}\delta \\
&= \boldsymbol{\theta}_t^2\,.
\end{aligned}$$

As $\boldsymbol{\theta}_0^1 = \boldsymbol{\theta}_0^2$ by assumption, $\boldsymbol{\theta}_t^1 = \boldsymbol{\theta}_t^2$ for all t.

Finally, we put everything together and note that $f_0^1 = f_0^2$ and that

$$\begin{aligned}
f_t^1(x) &= \boldsymbol{\Sigma}_t(\mathbf{W}_t^1\tilde{g}_{\boldsymbol{\theta}_t^1}(x) + \boldsymbol{b}_t^1) + \boldsymbol{\mu}_t \\
&= \boldsymbol{\Sigma}_t(\boldsymbol{\Sigma}_t^{-1}\mathbf{W}_t^2\tilde{g}_{\boldsymbol{\theta}_t^2}(x) + \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{b}_t^2 - \boldsymbol{\mu}_t)) + \boldsymbol{\mu}_t \\
&= \mathbf{W}_t^2\tilde{g}_{\boldsymbol{\theta}_t^2}(x) + \boldsymbol{b}_t^2 \\
&= f_t^2(x) \quad \forall x, t\,.
\end{aligned}$$

□

Fact 4. *If the targets $\{Y_t\}_{t=1}^{\infty}$ are distributed according to a normal distribution with arbitrary finite mean and variance, then the objective $P(\sigma^{-1}(Y - \mu) \in [-1, 1]) = p$ is equivalent to the joint objective $\mathbb{E}\left[Y - \mu\right] = 0$ and $\mathbb{E}\left[\sigma^{-2}(Y - \mu)^2\right] = s^2$ for*

$$p \;=\; \mathit{erf}\left(\frac{1}{\sqrt{2}s}\right)$$

Proof. For any μ and σ, the normalized targets are distributed according to a normal distribution because the targets themselves are normally distributed and the normalization is an affine transformation. For a normal distribution with mean zero and variance v, the values 1 and -1 are both exactly $1/\sqrt{v}$ standard deviations from the mean, implying that the ratio of data between these points is $\Phi(1/\sqrt{v}) - \Phi(-1/\sqrt{v})$, where

$$\Phi(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

is the standard normal cumulative distribution. The normalization by mean and variance is then equivalent to a normalization by percentiles with a ratio p defined by

$$\begin{aligned}
p \;&=\; \Phi(1/\sqrt{v}) - \Phi(-1/\sqrt{n}) \\
\;&=\; \frac{1}{2}\left(1 + \text{erf}\left(\frac{1}{\sqrt{2v}}\right)\right) - \frac{1}{2}\left(1 + \text{erf}\left(-\frac{1}{\sqrt{2v}}\right)\right) \\
\;&=\; \text{erf}\left(\frac{1}{\sqrt{2v}}\right)\,,
\end{aligned}$$

where we used the fact that erf is odd, such that $\text{erf}(x) = -\text{erf}(-x)$. □

Fact 5. *If $\sum_{t=1}^{\infty}\beta_t$ and $\sum_{t=1}^{\infty}\beta_t^2$, and the distribution of targets is stationary, then the updates*

$$y_t^{max} = y_{t-1}^{max} + \beta_t\left(\mathcal{I}(Y_t > y_{t-1}^{max}) - \frac{1-p}{2}\right) \quad \text{and}$$
$$y_t^{min} = y_{t-1}^{min} - \beta_t\left(\mathcal{I}(Y_t < y_{t-1}^{min}) - \frac{1-p}{2}\right)\,,$$

converge to values such that

$$P\left(Y > y_t^{max}\right) = P\left(Y < y_t^{min}\right) = \frac{1-p}{2}\,.$$

Proof. Note that

$$\begin{aligned}\mathbb{E}\left[y_t^{\max} = y_{t-1}^{\max}\right] &\iff \mathbb{E}\left[\mathcal{I}(Y > y_t^{\max})\right] = (1-p)/2 \\ &\iff P\left(Y > y_t^{\max}\right) = (1-p)/2\,,\end{aligned}$$

so this is a fixed point of the update. Note further that the variance of the stochastic update is finite, and that the expected direction of the updates is towards the fixed point, so that this fixed point is an attractor. The conditions on the step sizes ensure that the fixed point is reachable ($\sum_{t=1}^{\infty} \beta_t = \infty$) and that we converge upon it in the limit ($\sum_{t=1}^{\infty} \beta_t^2 < \infty$). For more detail and weaker conditions, we refer to reader to the extensive literature on stochastic approximation (Robbins and Monro, 1951; Kushner and Yin, 2003). The proof for the update for $y_t^{\min}$ is exactly analogous. □

Fact 6. *Consider minibatches $\{\{Y_{t,1}, \ldots, Y_{t,B}\}\}_{t=1}^{\infty}$ of size $B \geq 2$ whose elements are drawn i.i.d. from a uniform distribution with support on $[a, b]$. If $\sum_t \beta_t = \infty$ and $\sum_t \beta_t^2 < \infty$, then in the limit the updates*

$$\begin{aligned}y_t^{min} &= (1-\beta_t) y_{t-1}^{min} + \beta_t min_b Y_{t,b}\,, \quad \textit{and} \\ y_t^{max} &= (1-\beta_t) y_{t-1}^{max} + \beta_t max_b Y_{t,b}\end{aligned}$$

converge to values such that

$$P\left(Y > y_t^{max}\right) = P\left(Y < y_t^{min}\right) = (1-p)/2\,,$$

with $p = (B-1)/(B+1)$.

Proof. Because of the conditions on the step size, the quantities $y_t^{\min}$ and $y_t^{\max}$ will converge to the expected value for the minimum and maximum of a set of B i.i.d. random variables. The cumulative distribution function (CDF) for the maximum of B i.i.d. random variables with CDF $F(x)$ is $F(x)^B$, since

$$P(x < \max_{1 \leq b \leq B} Y_b) = \prod_{b=1}^{B} P(x < Y_b) = F(x)^B$$

The CDF for a uniform random variables with support on $[a, b]$ is

$$F(x) = \begin{cases} 0 & \text{if } x < a, \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b, \\ 1 & \text{if } x > b. \end{cases}$$

Therefore,

$$P(x \leq \max_{1 \leq b \leq B} Y_b) = \begin{cases} 0 & \text{if } x < a, \\ \left(\frac{x-a}{b-a}\right)^B & \text{if } a \leq x \leq b, \\ 1 & \text{if } x > b. \end{cases}$$

The associated expected value can then be calculated to be

$$\mathbb{E}\left[\max_{1 \leq b \leq B} Y_b\right] = a + \frac{B}{B+1}(b-a)\,,$$

so that a fraction of $\frac{1}{B+1}$ of samples will be larger than this value. Through a similar reasoning, an additional fraction of $\frac{1}{B+1}$ will be smaller than the minimum, and a ratio of $p = \frac{B-1}{B+1}$ will on average fall between these values. □

Fact 7. *Consider a weighted running average x_t updated from a stream of data $\{Z_t\}_{t=1}^{\infty}$ using*

$$(1-\beta_t) x_{t-1} + \beta_t Z_t\,,$$

with

$$\beta_t \equiv \frac{\beta}{1-(1-\beta)^t}\,,$$

where β is a constant. Then 1) the relative weights of the data in x_t are the same as when only the constant step size β is used, and 2) the average does not depend on the initial value x_0.

Proof. The point of the fact is to show that

$$\mu_t = \frac{\mu_t^{\beta} - (1-\beta)^t \mu_0}{1-(1-\beta)^t}\,, \quad \forall t\,, \tag{10}$$

where

$$\mu_t^{\beta} = (1-\beta)\mu_{t-1}^{\beta} + \beta Z_t\,, \quad \forall t\,,$$

and where $\mu_0 = \mu_0^{\beta}$. Note that μ_t as defined by (10) exactly removes the contribution of the initial value μ_0, which at time t have weight $(1-\beta)^t$ in the exponential moving average μ_t^{β}, and then renormalizes the remaining value by dividing by $1-(1-\beta)^t$, such that the relative weights of the observed samples $\{Z_t\}_{t=1}^{\infty}$ is conserved.

If (10) holds for μ_{t-1}, then

$$\begin{aligned}
\mu_t &= \left(1 - \frac{\beta}{1-(1-\beta)^t}\right)\mu_{t-1} + \frac{\beta}{1-(1-\beta)^t}Y_t \\
&= \left(1 - \frac{\beta}{1-(1-\beta)^t}\right)\frac{\mu^\beta_{t-1} - (1-\beta)^{t-1}\mu_0}{1-(1-\beta)^{t-1}} \\
&\qquad + \frac{\beta}{1-(1-\beta)^t}Y_t \\
&= \frac{1-(1-\beta)^t - \beta}{1-(1-\beta)^t}\frac{\mu^\beta_{t-1} - (1-\beta)^{t-1}\mu_0}{1-(1-\beta)^{t-1}} \\
&\qquad + \frac{\beta}{1-(1-\beta)^t}Y_t \\
&= \frac{(1-\beta)(1-(1-\beta)^{t-1})}{1-(1-\beta)^t}\frac{\mu^\beta_{t-1} - (1-\beta)^{t-1}\mu_0}{1-(1-\beta)^{t-1}} \\
&\qquad + \frac{\beta}{1-(1-\beta)^t}Y_t \\
&= \frac{(1-\beta)(\mu^\beta_{t-1} - (1-\beta)^{t-1}\mu_0)}{1-(1-\beta)^t} + \frac{\beta}{1-(1-\beta)^t}Y_t \\
&= \frac{(1-\beta)\mu^\beta_{t-1} + \beta Y_t - (1-\beta)^t\mu_0}{1-(1-\beta)^t} \\
&= \frac{\mu^\beta_t - (1-\beta)^t\mu_0}{1-(1-\beta)^t}\,,
\end{aligned}$$

so that then (10) holds for μ_t. Finally, verify that $\mu_1 = Y_1$. Therefore, (10) holds for all t by induction. □

Game	Random	Human	Double DQN	Double DQN with Pop-Art
Alien	227.80	7127.70	**3747.70**	3213.50
Amidar	5.80	1719.50	**1793.30**	782.50
Assault	222.40	742.00	5393.20	**9011.60**
Asterix	210.00	8503.30	17356.50	**18919.50**
Asteroids	719.10	47388.70	734.70	**2869.30**
Atlantis	12850.00	29028.10	106056.00	**340076.00**
Bank Heist	14.20	753.10	1030.60	**1103.30**
Battle Zone	2360.00	37187.50	**31700.00**	8220.00
Beam Rider	363.90	16926.50	**13772.80**	8299.40
Berzerk	123.70	2630.40	**1225.40**	1199.60
Bowling	23.10	160.70	68.10	**102.10**
Boxing	0.10	12.10	91.60	**99.30**
Breakout	1.70	30.50	**418.50**	344.10
Centipede	2090.90	12017.00	5409.40	**49065.80**
Chopper Command	811.00	7387.80	**5809.00**	775.00
Crazy Climber	10780.50	35829.40	117282.00	**119679.00**
Defender	2874.50	18688.90	**35338.50**	11099.00
Demon Attack	152.10	1971.00	58044.20	**63644.90**
Double Dunk	−18.60	−16.40	**−5.50**	−11.50
Enduro	0.00	860.50	1211.80	**2002.10**
Fishing Derby	−91.70	−38.70	15.50	**45.10**
Freeway	0.00	29.60	33.30	**33.40**
Frostbite	65.20	4334.70	1683.30	**3469.60**
Gopher	257.60	2412.50	14840.80	**56218.20**
Gravitar	173.00	3351.40	412.00	**483.50**
H.E.R.O.	1027.00	30826.40	**20130.20**	14225.20
Ice Hockey	−11.20	0.90	**−2.70**	−4.10
James Bond	29.00	302.80	**1358.00**	507.50
Kangaroo	52.00	3035.00	12992.00	**13150.00**
Krull	1598.00	2665.50	7920.50	**9745.10**
Kung-Fu Master	258.50	22736.30	29710.00	**34393.00**
Montezuma's Revenge	0.00	4753.30	**0.00**	**0.00**
Ms. Pacman	307.30	6951.60	2711.40	**4963.80**
Name This Game	2292.30	8049.00	10616.00	**15851.20**
Phoenix	761.40	7242.60	**12252.50**	6202.50
Pitfall	−229.40	6463.70	−29.90	**−2.60**
Pong	−20.70	14.60	**20.90**	20.60
Private Eye	24.90	69571.30	129.70	**286.70**
Q*Bert	163.90	13455.00	**15088.50**	5236.80
River Raid	1338.50	17118.00	**14884.50**	12530.80
Road Runner	11.50	7845.00	44127.00	**47770.00**
Robotank	2.20	11.90	**65.10**	64.30
Seaquest	68.40	42054.70	**16452.70**	10932.30
Skiing	−17098.10	−4336.90	**−9021.80**	−13585.10
Solaris	1236.30	12326.70	3067.80	**4544.80**
Space Invaders	148.00	1668.70	2525.50	**2589.70**
Star Gunner	664.00	10250.00	**60142.00**	589.00
Surround	−10.00	6.50	−2.90	**−2.50**
Tennis	−23.80	−8.30	−22.80	**12.10**
Time Pilot	3568.00	5229.20	**8339.00**	4870.00
Tutankham	11.40	167.60	**218.40**	183.90
Up and Down	533.40	11693.20	**22972.20**	22474.40
Venture	0.00	1187.50	98.00	**1172.00**
Video Pinball	16256.90	17667.90	**309941.90**	56287.00
Wizard of Wor	563.50	4756.50	**7492.00**	483.00
Yars Revenge	3092.90	54576.90	11712.60	**21409.50**
Zaxxon	32.50	9173.30	10163.00	**14402.00**

Table 1. Raw scores for a random agent, a human tested, Double DQN as described by van Hasselt et al. (2016), and Double DQN with Pop-Art and no reward clipping on 30 minutes of simulated play (108,000 frames). The random, human, and Double DQN scores are all taken from Wang et al. (2015).