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 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, 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

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

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 $\boldsymbol{\theta}$ of a function $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^k$ such that the output $f_{\boldsymbol{\theta}}(X_t)$ is (in expectation) close to the target Y_t according to some loss $l_t(f_{\boldsymbol{\theta}})$, for instance defined as a squared difference: $l_t(f_{\boldsymbol{\theta}}) = \frac{1}{2}(f_{\boldsymbol{\theta}}(X_t) - Y_t)^{\top}(f_{\boldsymbol{\theta}}(X_t) - Y_t)$. A canonical update is stochastic gradient descent (SGD). For a sample (X_t, Y_t), the update is then $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} l_t(f_{\boldsymbol{\theta}})$, 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 $f_{\boldsymbol{\theta}}$ 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

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

where $\boldsymbol{\Sigma}_t$ and $\boldsymbol{\mu}_t$ are *scale* and *shift* parameters that are learned from 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 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 $f(x) = \boldsymbol{\Sigma} g(x) + \boldsymbol{\mu}$, 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 $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ using the same algorithm as used for the parameters of the function g, then the problem has not become fundamentally different

2

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 $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ 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 $\boldsymbol{\Sigma}$ and shift $\boldsymbol{\mu}$ such that $\boldsymbol{\Sigma}^{-1}(Y - \boldsymbol{\mu})$ 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

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

where $h_{\boldsymbol{\theta}}$ is a parametrized (non-linear) function, and $g_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}} = \mathbf{W}h_{\boldsymbol{\theta}}(x) + \boldsymbol{b}$ is the normalized function. It is not uncommon for deep neural networks to end in a linear layer, and then $h_{\boldsymbol{\theta}}$ 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 $h_{\boldsymbol{\theta}}$ to ensure this constraint, for instance initialized as $\mathbf{W}_0 = \mathbf{I}$ and $\boldsymbol{b}_0 = \mathbf{0}$.

The following proposition 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.

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

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

where $h_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ 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 $\boldsymbol{\Sigma}_{\mathrm{new}}$ and from $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\mathrm{new}}$, where $\boldsymbol{\Sigma}_{\mathrm{new}}$ is non-singular. If we then additionally change the parameters $\mathbf{W}$ and $\boldsymbol{b}$ to $\mathbf{W}_{\mathrm{new}}$ and $\boldsymbol{b}_{\mathrm{new}}$, defined by

$$\mathbf{W}_{\mathrm{new}} = \boldsymbol{\Sigma}_{\mathrm{new}}^{-1}\boldsymbol{\Sigma}\mathbf{W} \qquad \text{and} \qquad \boldsymbol{b}_{\mathrm{new}} = \boldsymbol{\Sigma}_{\mathrm{new}}^{-1}\left(\boldsymbol{\Sigma}\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_{\mathrm{new}}\right)\,,$$

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

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

This and later propositions 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 $\mathbf{W}_{\mathrm{new}} = (\sigma/\sigma_{\mathrm{new}})\mathbf{W}$ and $\boldsymbol{b}_{\mathrm{new}} = (\sigma\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_{\mathrm{new}})/\sigma_{\mathrm{new}}$. After updating the scale and shift we can update the output of the normalized function $g_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(X_t)$ 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 is made more precise in Proposition 2 in Section 2.2.

3

Algorithm 1 SGD on squared loss with Pop-Art

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

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

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

$$\text{such that} \qquad \mu_t = \frac{1}{t}\sum_{i=1}^{t}Y_i \qquad \text{and} \qquad \sigma_t = \frac{1}{t}\sum_{i=1}^{t}Y_i^2 - \mu_t^2\,. \qquad (3)$$

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.

4

Algorithm 2 Normalized SGD

For a given differentiable function $h_{\boldsymbol{\theta}}$, initialize $\boldsymbol{\theta}$.
while learning **do**
    Observe input X and target Y
    Use Y to compute new scale $\boldsymbol{\Sigma}$
    $\boldsymbol{h} \leftarrow h_{\boldsymbol{\theta}}(X)$ (store output of $h_{\boldsymbol{\theta}}$)
    $\boldsymbol{J} \leftarrow (\nabla h_{\boldsymbol{\theta},1}(X), \dots, \nabla h_{\boldsymbol{\theta},m}(X))^{\top}$ (compute Jacobian of $h_{\boldsymbol{\theta}}$)
    $\boldsymbol{\delta} \leftarrow \mathbf{W}\boldsymbol{h} + \boldsymbol{b} - Y$ (compute *un*normalized error)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{J}(\boldsymbol{\Sigma}^{-1}\mathbf{W})^{\top}\boldsymbol{\Sigma}^{-1}\boldsymbol{\delta}$ (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

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

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

where $\boldsymbol{\delta}_t = f_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{b}}(X) - Y_t$ 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 $h_{\boldsymbol{\theta}}$, $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \boldsymbol{J}_t \mathbf{W}_{t-1}^{\top}\boldsymbol{\delta}_t$ where $\boldsymbol{J}_t = (\nabla g_{\boldsymbol{\theta},1}(X), \dots, \nabla g_{\boldsymbol{\theta},m}(X))^{\top}$ is the Jacobian for $h_{\boldsymbol{\theta}}$. 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.

Proposition 3. *Consider two functions defined by*

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

where $h_{\boldsymbol{\theta}}$ 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 (Pop-Art SGD) and the second using Algorithm 2 (Normalized SGD). 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.

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,

---

[1] 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].

5
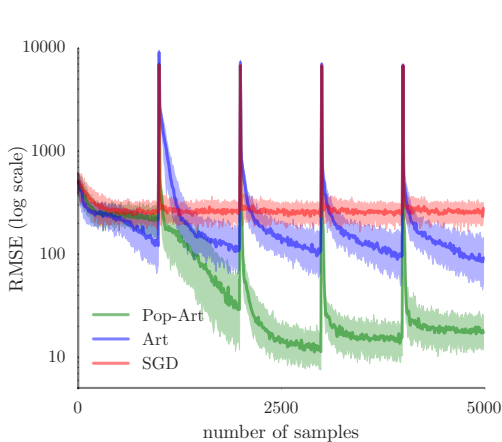
Fig. 1a. Median RMSE on binary regression for SGD without normalization (**red**), with normalization but without preserving outputs (**blue**, labeled 'Art'), and with Pop-Art (**green**). Shaded 10–90 percentiles.
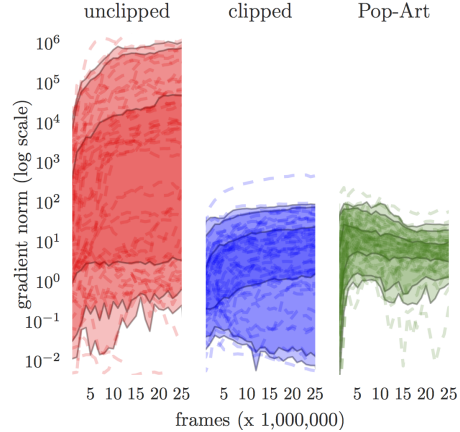
Fig. 1b. ℓ^2 gradient norms for DQN during learning on 57 Atari games with actual unclipped rewards (left, **red**), clipped rewards (middle, **blue**), and using Pop-Art (right, **green**) instead of clipping. Shaded areas correspond to 95%, 90% and 50% of games.

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 $\{10^{-5}, 10^{-4.5}, \ldots, 10^{-1}, 10^{-0.5}, 1\}$.

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

6

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 is 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:

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

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, 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.

7
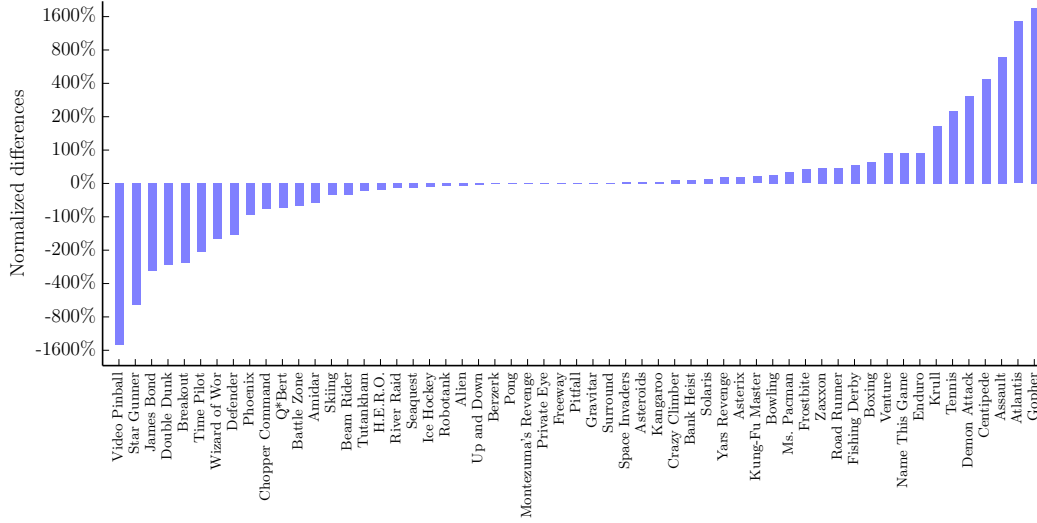
Figure 2: Differences between normalized scores for Double DQN with and without Pop-Art on 57 Atari games.

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.

8

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.

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. B. Adam. A method for stochastic optimization. In *International Conference on Learning Representation*, 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.

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.

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

I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped DQN. *CoRR*, abs/1602.04621, 2016.

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.

9

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. COURSERA: 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.

Appendix

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

Experiment setup

For the experiments described in Section 4 in the main paper, 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. [2016].

The results in Figure 2 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 Proposition 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.

10
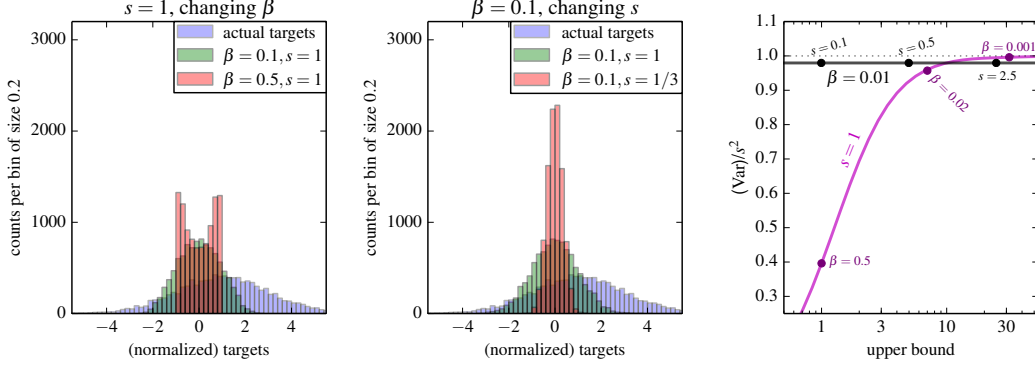
Figure 3: The **left** plot shows histograms for 10,000 normally distributed targets with mean 1 and standard deviation 2 (**blue**) and for normalized targets for $\beta = 0.1$ (**green**) and $\beta = 0.5$ (**red**). The **middle** plot shows the same histograms, except that the histogram for $\beta = 0.5$ 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).

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 3 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 (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

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

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

Proposition 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 $\mathbb{E}\left[Y - \mu\right] = 0$ and $\mathbb{E}\left[\sigma^{-2}(Y - \mu)^2\right] = s^2$ with*

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

11

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,[2] such that $Y_t^{(1)} = \min_i \{Y_i\}_{i=1}^t$, $Y_t^{(t)} = \max_i \{Y_i\}_{i=1}^t$, and $Y_t^{((t+1)/2)} = \text{median}_i \{Y_i\}_{i=1}^t$. 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

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

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 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

$$P\left(Y > y_t^{\max}\right) = P\left(Y < y_t^{\min}\right) = (1 - p)/2 \,. \tag{6}$$

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

$$y_t^{\max} = y_{t-1}^{\max} + \beta_t\left(\mathcal{I}(Y_t > y_{t-1}^{\max}) - \frac{1-p}{2}\right) \qquad \text{and} \tag{7}$$

$$y_t^{\min} = y_{t-1}^{\min} - \beta_t\left(\mathcal{I}(Y_t < y_{t-1}^{\min}) - \frac{1-p}{2}\right) \,, \tag{8}$$

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

Proposition 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* (7) *converge to values such that* (6) *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

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

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

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

---

[2] 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.

12

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

$$y_t^{\min} = (1 - \beta_t)y_{t-1}^{\min} + \beta_t \min_b Y_{t,b}, \quad \text{and} \qquad (9)$$
$$y_t^{\max} = (1 - \beta_t)y_{t-1}^{\max} + \beta_t \max_b Y_{t,b},$$

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.

Proposition 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 (9) converge to values such that (6) holds, with $p = (B - 1)/(B + 1)$.*

This fact connects the online minibatch updates (9) 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 Proposition 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.

Proposition 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.

Proposition 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*

$$\beta_t = \beta(1 - (1 - \beta)^t)^{-1}. \qquad (10)$$

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 (10) 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 (10), 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

$$g_{\boldsymbol{\theta},i}(X) = \frac{e^{[\mathbf{W}h_{\boldsymbol{\theta}}(X) + \boldsymbol{b}]_i}}{\sum_j e^{[\mathbf{W}h_{\boldsymbol{\theta}}(X) + \boldsymbol{b}]_j}},$$

13

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 $\mathbf{W}h_{\boldsymbol{\theta}}(X) + \boldsymbol{b}$ 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

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

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

where $h_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$ 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

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

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

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

Proposition 2. *When using updates (4) to adapt the normalization parameters σ and μ, the normalized target $\sigma_t^{-1}(Y_t - \mu_t)$ is bounded for all t by*

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

14

Proof.

$$\left(\frac{Y_t - \mu_t}{\sigma_t}\right)^2 = \left(\frac{Y_t - (1-\beta_t)\mu_{t-1} - \beta_t Y_t}{\sigma_t}\right)^2$$

$$= \frac{(1-\beta_t)^2(Y_t - \mu_{t-1})^2}{\nu_t - \mu_t^2}$$

$$= \frac{(1-\beta_t)^2(Y_t - \mu_{t-1})^2}{(1-\beta_t)\nu_{t-1} + \beta_t Y_t^2 - ((1-\beta_t)\mu_{t-1} + \beta_t Y_t)^2}$$

$$= \frac{(1-\beta_t)^2(Y_t - \mu_{t-1})^2}{(1-\beta_t)\left(\nu_{t-1} + \beta_t Y_t^2 - (1-\beta_t)\mu_{t-1}^2 - 2\beta_t \mu_{t-1} Y_t\right)}$$

$$= \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\nu_{t-1} + \beta_t Y_t^2 - (1-\beta_t)\mu_{t-1}^2 - 2\beta_t \mu_{t-1} Y_t}$$

$$\leq \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\mu_{t-1}^2 + \beta_t Y_t^2 - (1-\beta_t)\mu_{t-1}^2 - 2\beta_t \mu_{t-1} Y_t}$$

$$= \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\beta_t Y_t^2 + \beta_t \mu_{t-1}^2 - 2\beta_t \mu_{t-1} Y_t}$$

$$= \frac{(1-\beta_t)(Y_t - \mu_{t-1})^2}{\beta_t(Y_t - \mu_{t-1})^2}$$

$$= \frac{(1-\beta_t)}{\beta_t},$$

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

Proposition 3. *Consider two functions defined by*

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

where $h_{\boldsymbol{\theta}}$ 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 $h_{\boldsymbol{\theta}}$ 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 $\mathbf{W}_t^1 = \mathbf{W}_0^2 = \mathbf{W}_0$, and $\boldsymbol{\Sigma}_0 = \mathbf{I}$. 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.

15

Then, Algorithm 1 results in

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

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

$$\boldsymbol{b}_t^1 = \mathbf{\Sigma}_t^{-1}(\mathbf{\Sigma}_{t-1}\boldsymbol{b}_{t-1}^1 + \mu_{t-1} - \mu_t) + \alpha\mathbf{\Sigma}_t^{-1}\delta_t$$
$$= \mathbf{\Sigma}_t^{-1}(\boldsymbol{b}_{t-1}^2 - \mu_t) + \alpha\mathbf{\Sigma}_t^{-1}\delta_t$$
$$= \mathbf{\Sigma}_t^{-1}(\boldsymbol{b}_{t-1}^2 - \mu_t + \alpha\delta_t)$$
$$= \mathbf{\Sigma}_t^{-1}(\boldsymbol{b}_t^2 - \mu_t).$$

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

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

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

$$f_t^1(x) = \mathbf{\Sigma}_t(\mathbf{W}_t^1 h_{\boldsymbol{\theta}_t^1}(x) + \boldsymbol{b}_t^1) + \mu_t$$
$$= \mathbf{\Sigma}_t(\mathbf{\Sigma}_t^{-1}\mathbf{W}_t^2 h_{\boldsymbol{\theta}_t^2}(x) + \mathbf{\Sigma}_t^{-1}(\boldsymbol{b}_t^2 - \mu_t)) + \mu_t$$
$$= \mathbf{W}_t^2 h_{\boldsymbol{\theta}_t^2}(x) + \boldsymbol{b}_t^2$$
$$= f_t^2(x) \quad \forall x, t. \qquad \square$$

Proposition 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 $\mathbb{E}[Y - \mu] = 0$ and $\mathbb{E}\left[\sigma^{-2}(Y - \mu)^2\right] = s^2$ for*

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

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

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

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

$$p = \Phi(1/\sqrt{v}) - \Phi(-1/\sqrt{n})$$
$$= \frac{1}{2}\left(1 + erf\left(\frac{1}{\sqrt{2v}}\right)\right) - \frac{1}{2}\left(1 + erf\left(-\frac{1}{\sqrt{2v}}\right)\right)$$
$$= erf\left(\frac{1}{\sqrt{2v}}\right),$$

where we used the fact that erf is odd, such that $erf(x) = -erf(-x)$. $\qquad \square$

16

Proposition 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*

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

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

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

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

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. □

Proposition 6. *Consider minibatches $\{\{Y_{t,1},\dots,Y_{t,B}\}\}_{t=1}^{\infty}$ 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*

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

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. □

17

Proposition 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

$$\mu_t = \frac{\mu_t^{\beta} - (1 - \beta)^t \mu_0}{1 - (1 - \beta)^t}, \quad \forall t, \tag{11}$$

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 (11) 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 (11) holds for μ_{t-1}, then

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

so that then (11) holds for μ_t. Finally, verify that $\mu_1 = Y_1$. Therefore, (11) holds for all t by induction. $\square$

18

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. [2016].

19