Continual Backprop: Stochastic Gradient Descent with Persistent Randomness

Shibhansh Dohare,[1] **A. Rupam Mahmood,**[1,2] **Richard S. Sutton**[1,2,3]
[1]Department of Computing Science, University of Alberta, Edmonton, Canada
[2]Canada CIFAR AI Chair, Alberta Machine Intelligence Institute (Amii)
[3]Deepmind,Edmonton, Canada
dohare@ualberta.ca armahmood@ualberta.ca rsutton@ualberta.ca

Abstract

The Backprop algorithm for learning in neural networks utilizes two mechanisms: first, stochastic gradient descent and second, initialization with small random weights, where the latter is essential to the effectiveness of the former. We show that in continual learning setups, Backprop performs well initially, but over time its performance degrades. Stochastic gradient descent alone is insufficient to learn continually; the initial randomness enables only initial learning but not continual learning. To the best of our knowledge, ours is the first result showing this degradation in Backprop's ability to learn. To address this degradation in Backprop's plasticity, we propose an algorithm that continually injects random features alongside gradient descent using a new generate-and-test process. We call this the *Continual Backprop* algorithm. We show that, unlike Backprop, Continual Backprop is able to continually adapt in both supervised and reinforcement learning (RL) problems. Continual Backprop has the same computational complexity as Backprop and can be seen as a natural extension of Backprop for continual learning.

Keywords: Continual Learning; Plasticity; Deep Learning; Reinforcement Learning

Acknowledgements

The authors would like to thank Martha White for her feedback on earlier version of this work. The authors would also like to thank Compute Canada for providing the computational resources to carry out the experiments in this paper. This work was sponsored by Canada CIFAR AI Chairs Program, the Reinforcement Learning and Artificial Intelligence (RLAI) Laboratory, DeepMind, and Alberta Machine Intelligence Institute (Amii).

1 Introduction

In the last decade, deep learning methods have been successful and become the state-of-the-art in many machine learning problems and applications, including supervised classification (Krizhevsky et al., 2012), reinforcement learning (Silver et al., 2016), and natural language processing (Brown et al., 2020). These methods learn the weights of an artificial neural network using Backprop, which is primarily applied to stationary problems. However, a primary challenge to leveraging the strengths of deep learning beyond current applications is that Backprop does not work well in non-stationary problems (McCloskey and Cohen, 1989; Sahoo et al., 2018), for example, a problem that consists a sequence of stationary problems.

Leveraging the strengths of deep learning methods in non-stationary problems is important as many real-world applications of machine learning like robotics involve non-stationrities. Non-stationarities can arise due to high complexity (Sutton et al., 2007), partial observability (Khetarpal et al., 2020), other actors (Foerster et al., 2018), or changes in the environment (Thurn, 1998). Prior works proposed to remember previously learned information (Kirkpatrick et al., 2017; Aljundi et al., 2018; Riemer et al., 2019). Some other works proposed to adapt fast to non-stationarities (Rusu et al., 2016; Al-Shedivat et al., 2018; Finn et al., 2019).

Prior works on non-stationary problems are limited due to their lack of experiments with many non-stationarities. The number of non-stationarities refers to the number of times the data distribution changes. Most works only look at problems with less than ten non-stationarities (Rusu et al., 2016; Kirkpatrick et al., 2017; Al-Shedivat et al., 2018). Finn et al. (2019) studied problems with up to a hundred non-stationarities. Dealing with many non-stationarities is important for systems that continually interact with the real world as non-stationarities frequently occur in the world.

In this work, we study problems with a large number (thousands) of non-stationarities. We start with Permuted MNIST, which belongs to a special class of problems that we call *semi-stationary* problems. These are online supervised learning problems where the input distribution is non-stationary while the target function is stationary. The target function is the function being approximated, for example, the true regression function. These problems are a natural way to study many non-stationarities; slowly moving through a large input space can cause thousands of non-stationarities. Semi-stationarities are especially relevant in the real world where inputs often depend on previous inputs, making the input distribution non-stationary. Finally, we study a non-stationary RL problem. This is a full non-stationarity problem as the input distribution changes when the agent's behaviour changes and the target function—optimal policy—changes as the dynamics of the environment change.

We show that in non-stationary problems Backprop performs well initially, but surprisingly, its performance continues to degrade substantially over time as Backprop loses its ability to adapt. Backprop relies on proper random initialization for its effectiveness (Glorot et al., 2010; Sutskerver et al., 2013; He et al., 2015). However, randomization in Backprop only happens at the beginning. We hypothesize that Backprop's ability to adapt degrades because the benefits of initial random distribution are not present at all times.

To extend the benefits of the initial randomization throughout learning, we propose the Continual Backprop (CBP) algorithm. CBP continually injects random features alongside SGD and uses a generate-and-test mechanism that consists of two parts: a *generator* proposes new features, and a *tester* finds and replaces low utility features with the features proposed by the generator. We show that unlike BP, CBP can continually adapt in non-stationary problems.

The first contribution of our work is to show that in non-stationary problems with many non-stationarities BP loses its ability to adapt over time. In other words, we contribute to the understanding of why BP fails in non-stationary problems. Our second contribution is the CBP algorithm that extends the benefits of the initialization in BP to all times.

2 Backprop loses the ability to adapt under extended tracking

Permuted MNIST (Zenke et al., 2017) is an online image classification problem with ten classes. The images in permuted MNIST are generated from the images in the MNIST dataset by randomly permuting the pixels in the images. We present the images sequentially and measure online classification accuracy. The MNIST dataset has 60k images. We present these 60k images in random order, and after all the images have been presented, we use a single permutation to change all the images. This cycle of presenting all the 60k images and then changing the permutation of pixels can be continued indefinitely, which allows us to create a long-term continual learning problem. However, the original work by Zenke et al. (2017) only used this task with ten non-stationarities.

All solution methods used a feed-forward network with three hidden layers with 2000 hidden units, ReLU activations, and used SGD for updates.

Figure 1 shows that in the permuted MNIST problem, the performance of Backprop gets worse for all steps-sizes. The degradation in performance is slower for smaller step-sizes. From these results, we conclude that Backprop is not suitable for extended tracking as it continues to lose its ability to adapt over time.

The weights of the learning network are different at the beginning and after some gradient-based updates. In the beginning, they were small random weights; however, after learning for some time, the weights became optimized to reduce the error for the most recent data distribution. Thus, the starting weights to get to the next solution have changed. As this difference in the weights is the only difference in the learning algorithm across time, the initial weight distribution must have some special properties that allow the learning network to adapt fast.

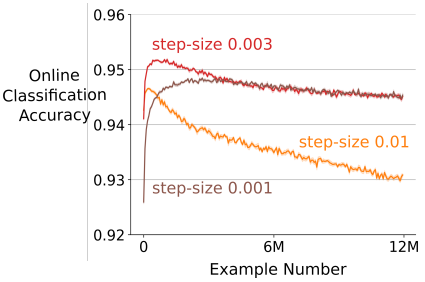

Figure 1: The online classification accuracy of a deep ReLU-network trained using Backprop on Permuted MNIST. The online accuracy is binned among bins of size 60k. The performance of BP gets worse over time for all step-sizes, meaning that BP loses its ability to adapt under extended tracking.

3 Continual Backprop

The benefits of initializing with small random numbers in Backprop are only present initially, as initialization with small random weights only happens at the start. This special initialization makes Backprop temporally asymmetric as it does a special computation at the beginning which is not continued later. But continually learning algorithms should be temporally symmetric as they should do similar computations at all times.

One way to extend the benefits of initialization throughout learning is by continually injecting random features. A feature refers to a hidden unit in a network. However, we are interested in continually learning methods that have similar memory requirements at all times because methods whose memory requirements grow with the amount of data are not scalable. So we also need to remove some features without affecting the already learned function too much. The features to be replaced can be chosen either randomly or using some measure of utility.

Continually replacing low utility features with new random features is a natural way to extend the benefits of initialization to all times while using constant memory. We use a generate-and-test process to continually replace low utility features with new random features. This process has two parts. First is a generator that provides new random features from the initial distribution. The second is a tester that finds low utility features and replaces them with features proposed by the generator. Our generate-and-test process is applicable to arbitrary feed-forward networks, modern activations, and optimizers. While the prior algorithm (Mahmood & Sutton, 2013) was limited to learning networks with a single hidden layer, only one output, LTU activation and could only use SGD to learn.

The generator creates new features by randomly sampling from the distribution that was used to initialize the weights. When a new feature is added, its outgoing weights are initialized to zero. Initializing as zero ensures that the newly added feature does not affect the already learned function. However, initializing the outgoing weight to zero makes it vulnerable to immediate replacement. The new features are protected for *maturity-threshold*, m, number of updates.

The tester finds low utility features and replaces them. At every time-step, *replacement-rate*, ρ, fraction of features are replaced in every layer.

We define the overall utility of a feature as the product of its mean-corrected contribution-utility and adaptation-utility. Where we define, the mean-corrected contribution utility as the magnitude of the product of connecting weight and input minus the average value of the input. The basic intuition behind the contribution part is that magnitude of the product of feature activation, and outgoing weight gives information about how useful this connection is to the next feature. And we define the *adaptation-utility* as the inverse of the average magnitude of the features' input weights. The adaptation-utility captures how fast a feature can adapt. The overall utility, $\hat{u}_{l,i,t}$, becomes

$$u_{l,i,t} = (1-\eta) * \frac{|h_{l,i,t} - \hat{f}_{l,i,t}| * \sum_{k=1}^{n_{l+1}} |w_{l,i,k,t}|}{\sum_{j=1}^{n_{l-1}} |w_{l-1,j,i,t}|} + \eta * u_{l,i,t-1}, \tag{1}$$

$$\hat{u}_{l,i,t} = \frac{u_{l,i,t-1}}{1-\eta^{a_{l,i,t}}}. \tag{2}$$

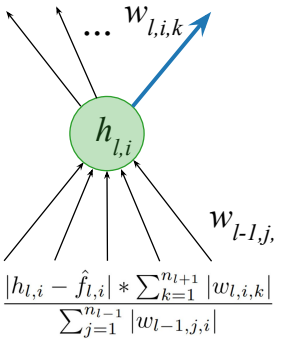

Figure 2: A feature/hidden-unit in a network and its utility at time t.

where $h_{l,i,t}$ is features' output, $w_{l,i,k,t}$ is the weight connecting the feature to the kth unit in layer $l+1$, n_{l+1} is the number of units is layer $l+1$, $a_{l,i,t}$ is the age of the feature at time t. Here, $\hat{f}_{l,i,t}$ is the bias-corrected running average of $h_{l,i,t}$.

The final algorithm combines BP with our generate-and-test algorithm to continually inject random features from the initial distribution. We refer to this algorithm as *Continual Backprop* (CBP). CBP performs a gradient-descent and a generate-and-test step at every time step. Algorithm 1 specifies the CBP algorithm for a feed-forward neural network.

Continual Backprop on Permuted MNIST

Algorithm 1: Continual Backprop (CBP) for a feed forward neural network with L hidden layers

Set: step-size α, replacement rate ρ, decay rate η, and maturity threshold m (e.g. 10^{-4}, 10^{-4}, 0.99, and 100)
Initialize: Initialize the weights $\mathbf{w}_0, ..., \mathbf{w}_L$. Let, $\mathbf{w}_l$ be sampled from a distribution d_l
Initialize: Utilities $\mathbf{u}_1, ..., \mathbf{u}_L$, average feature activation $\mathbf{f}_1, ..., \mathbf{f}_l$, and ages $\mathbf{a}_1, ..., \mathbf{a}_L$ to 0
for *each input* x_t **do**
  Forward pass: pass input through the network, get the prediction, $\hat{y}_t$
  Evaluate: Receive loss $l(x_t, \hat{y}_t)$
  Backward pass: update the weights using stochastic gradient descent
  for *layer* l *in* $1 : L$ **do**
    Update age: $\mathbf{a}_l \mathrel{+}= 1$
    Update feature utility: Using Equation 5
    Find eligible features: Features with age more than m
    Features to replace: $n_l * \rho$ of eligible features with smallest utility, let their indices be $\mathbf{r}$
    Initialize input weights: Reset the input weights $\mathbf{w}_{l-1}[\mathbf{r}]$ using samples from d_l
    Initialize output weights: Set $\mathbf{w}_l[\mathbf{r}]$ to zero
    Initialize utility, feature activation, and age: Set $\mathbf{u}_{l,\mathbf{r},t}$, $\mathbf{f}_{l,\mathbf{r},t}$, and $\mathbf{a}_{l,\mathbf{r},t}$ to 0

The most common ways to keep the weight distribution close to the initial distribution are L2 regularization and BatchNorm. So, we use these two methods along with BP and compare them to CBP. As we study online problems and we consider incremental updates, we used OnlineNorm (Chiley et al., 2019), an online variant of BatchNorm. Figure 3 show that among all the algorithms, only CBP is stable as its performance does not degrade over time. The performance of all the other algorithms degrades over time.

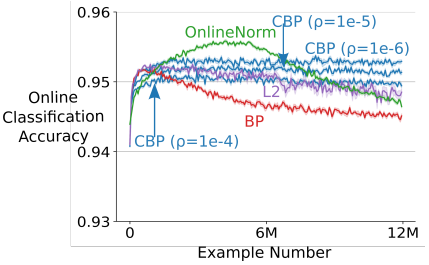

Figure 3: The online classification accuracy of various algorithms on Permuted MNIST. The performance of all algorithms except CBP degrade over time. CBP maintains a good level of performance for a wide range of replacement rates, ρ.

Continual Learning in a non-stationary RL problem

Slippery Ant is a non-stationary RL problems and is a continual variant of Pybullet's (Coumans and Bai, 2016) Ant problem. In our problems the environment changes after every 10M time-steps, making it necessary for the learner to adapt. In the standard problem, the value of the friction is fixed at 1.5. We change the friction after every 10M time-steps by log-uniformly sampling it from $[1e-4, 1e4]$.

We used the PPO algorithm as described by Schaulman et al. (2017). We used separate networks for policy and value function, and both had two hidden layers with 256 hidden units with tanh activation. To use CBP with PPO, we use CBP instead of BP to update the networks' weights. Whenever the weights are updated using Adam, we also update them using generate-and-test. We call the resulting algorithm *Continual PPO*.

4 Conclusion

In this work, we studied learning problems that have a large number of non-stationarities. We showed that in these problems, the performance of Backprop degrades over time, and it loses its ability to adapt. In other words, we showed that neural networks have a "decaying plasticity" problem. Ours is the first result showing this degradation in Backprop's ability to adapt. This result means that Backprop can be unsuitable for continual adaptation.

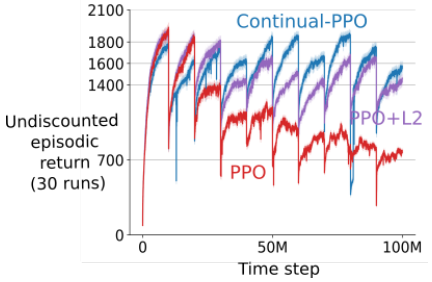

Figure 4: Performance of PPO, PPO+L2, and Continual-PPO on Slippery Ant. The performance of PPO degraded rapidly. In contrast, both CPPO and PPO+L2 perform substantially better than PPO, and CPPO performed best.

We proposed the Continual Backprop (CBP) algorithm to get a robust continual learning algorithm where both components of Backprop are present continually. CBP extends the benefits of special random initialization throughout learning using a generate-and-test process. We showed that CBP could continually adapt even after thousands of non-stationarities.

Continual learning is becoming more and more relevant in machine learning. These systems can not use BP or its variants in their current form, as they are not stable in non-stationary problems. We expect that by establishing Backprop's decaying plasticity, our work will invite many solutions similar to how many works have addressed "catastrophic forgetting". We foresee that our work will open a path for better Continual Backprop algorithms.

We expect Continual Backprop to be particularly relevant for Reinforcement Learning. RL problems are inherently non-stationary as the state-action distribution changes as the policy changes, and the target function of the value network changes as the policy changes. Studying the effect of decaying plasticity in RL is an exciting direction for future work.

References

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., & Abbeel, P. (2018). Continuous adaptation via meta-learning in nonstationary and competitive environments. International Conference on Learning Representations.

Aljundi, R., Lin, M., Goujaud, B., & Bengio, Y. (2019). Gradient based sample selection for online continual learning. Advances in Neural Information Processing Systems.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems (pp. 1877-1901).

Chiley, V., Sharapov, I., Kosson, A., Koster, U., Reece, R., Samaniego de la Fuente, S., ... & James, M. (2019). Online normalization for training neural networks. Advances in Neural Information Processing Systems, 32, 8433-8443.

Coumans, E., and Bai, Y. (2016–2021). Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`

Finn, C., Rajeswaran, A., Kakade, S., and Levine, S. (2019, May). Online meta-learning. In International Conference on Machine Learning (pp. 1920-1930). PMLR.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018, April). Counterfactual multi-agent policy gradients. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

Khetarpal, K., Riemer, M., Rish, I., & Precup, D. (2020). Towards Continual Reinforcement Learning: A Review and Perspectives. arXiv preprint arXiv:2012.13490.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., ... & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences, 114(13), 3521-3526.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.

McCloskey, M., and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In Psychology of learning and motivation (Vol. 24, pp. 109-165). Academic Press.

Mahmood, A. R., and Sutton, R. S. (2013, June). Representation Search through Generate and Test. In AAAI Workshop: Learning Rich Representations from Low-Level Sensors.

Sahoo, D., Pham, H. Q., Lu, J., & Hoi, S. C. Online deep learning: Learning deep neural networks on the fly.(2018). In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence IJCAI 2018, July 13-19, Stockholm (pp. 2660-2666).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Sebastian Thrun. Lifelong learning algorithms.Learning to learn, 8:181–209, 1998.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013, February). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147).

Sutton, R. S., Koop, A., and Silver, D.. On the role of tracking in stationary environments.In Proceedings of the 24th international conference on Machine learning, pp. 871–878. ACM, 2007.

Zenke, Friedemann, Ben Poole, and Surya Ganguli. "Continual learning through synaptic intelligence." In International Conference on Machine Learning, pp. 3987-3995. PMLR, 2017.