CS 229 Project Report: Backpropogation for Continual Learning

Sanjit Neelam
ICME
Stanford University
sneelam@stanford.edu

Bryan Xu
ICME
Stanford University
bryanaxu@stanford.edu

1 Introduction

Continual learning refers to problems where the input distribution and/or target function are non-stationary. This is in contrast to vanilla supervised learning problems such as MNIST image classification (Deng [2012]), where both the input distribution and target function are unchanging. We find continual learning problems interesting as it seems that solutions to continual learning problems can inform the design of more general artificial intelligence.

Traditional methods fail in a continual learning setting after the task distribution changes sufficiently; this phenomenon is known as *loss of plasticity*, or the loss of the ability to learn. A deep neural network equipped with the vanilla backpropogation algorithm falls short in several ways: the percentage of dead units increases (causing its output and thus plasticity to be zero), the magnitude of the weights increases (units become too over-committed and difficult to change), and the effective rank of the activity in the activation layers decreases (loss of diversity and expression). As a result, backprop is outperformed by even a linear classifier after multiple changes in the task.

To remedy this, we build upon the results in Dohare et al. [2022], who propose the continual backprop (CBP) algorithm to learn continually. They hypothesize that backprop is unable to learn new tasks because the benefits of random initialisation are absent after the model has learned a few tasks; indeed, backprop relies on random initialisation with small weights (e.g. Glorot and Bengio [2010], Sutskever et al. [2013], and He et al. [2015]).

We evaluate our proposed algorithms on the *slowly changing regression* problem, where we track the the mean squared error of our regression models during online learning. Our input is randomly generated bit data, and after a certain amount of time-steps the input distribution is randomly shifted. The model we are training is not re-initialised or told when this occurs, and we compare the error of models trained using our approaches, CBP, and other baselines over time.

2 Related Work

One of the first approaches to addressing the loss of plasticity of the backprop algorithm in an online learning setting was Hinton et al. [2012], in which they set a random fraction of units to zero in training, which they called *dropout*. However, this was found to be worse than backprop itself on the permuted MNIST problem. Chiley et al. [2019] studied *online normalization*, in which all signals internal to the network are shifted and scaled online. This approach works well initially, but as the task distribution shifts further and further away from the initial task, its performance falls drastically. Ash and Adams [2020] modify L^2 regularization, in which all weights are shrunk towards zero on each training example, by also adding random noise to all weights.

Inspiration for our modifications to the CBP algorithm stem from *network pruning*, which in short means finding conditions for when to remove, or 'prune', dead neurons in a deep neural network. Molchanov et al. [2019] estimate the contribution of a neuron to the final loss and iteratively removes

those with smaller scores. Hu et al. [2016] introduce network trimming, which iteratively optimizes the network by pruning unimportant neurons based on analysis of their outputs on a large dataset.

3 Methods

Dohare et al. [2022] propose the *continual backprop* (CBP) algorithm. The pipeline for CBP is split into two parts: (1) a generator finds new features by randomly sampling from some distribution, and (2) a tester sorts the features at a time-step t by utility and resets a fraction of them at replacement rate ρ, with recently replaced features protected by some maturity threshold m. For feature i in layer ℓ and time t, we compute its *mean-corrected contribution utility* $z_{\ell,i,t}$. Define

$$f_{\ell,i,t} = (1-\eta)\cdot h_{\ell,i,t} + \eta \cdot f_{\ell,i,t-1}, \tag{1}$$

$$\hat{f}_{\ell,i,t} = \frac{f_{\ell,i,t-1}}{1-\eta^{a_{\ell,i,t}}}, \tag{2}$$

$$z_{\ell,i,t} = (1-\eta)\cdot |h_{\ell,i,t} - \hat{f}_{\ell,i,t}| \cdot \sum_{k=1}^{n_{\ell+1}} |w_{\ell,i,k,t}| + \eta \cdot z_{\ell,i,t-1}. \tag{3}$$

Here, $h_{\ell,i,t}$ is the features' output, $w_{\ell,i,k,t}$ is the weight connected feature k in layer ℓ to feature i in layer $\ell+1$, $n_{\ell+1}$ is the number of features in layer $\ell+1$, $a_{\ell,i,t}$ is the age of a feature at time t, and η is a decay hyperparameter. Here, $f_{\ell,i,t}$ represents a running average of $h_{\ell,i,t}$ so as to keep track of its utility over the course of learning, and $\hat{f}_{\ell,i,t}$ is a bias-correction term that scales with the age of the feature. Finally, we multiply this by the *adaptation utility*, the inverse of the average magnitude of the features' input weights, to get the overall utility $\hat{u}_{\ell,i,t}$ of a feature. Thus, we get the system

$$u_{\ell,i,t} = \frac{|h_{\ell,i,t} - \hat{f}_{\ell,i,t}| \cdot \sum_{k=1}^{n_{\ell+1}} |w_{\ell,i,k,t}|}{\sum_{j=1}^{n_{\ell-1}} |w_{\ell-1,j,i,t}|}, \tag{4}$$

$$u_{\ell,i,t} = (1-\eta)\cdot u_{\ell,i,t} + \eta \cdot u_{\ell,i,t-1}, \tag{5}$$

$$\hat{u}_{\ell,i,t} = \frac{u_{\ell,i,t-1}}{1-\eta^{a_{\ell,i,t}}}, \tag{6}$$

and we give the pseudocode for the CBP algorithm below. We note that the *generate-and-test* mechanism is implemented as an additional step between the backward pass and the next forward pass, and does not increase the asymptotic runtime of training with backprop.

Algorithm 1 Continual Backprop (CBP) for Feed-Forward Neural Network with L Hidden Layers

Require: Step size α, replacement rate ρ, decay rate η, and maturity threshold m
Require: Weights $w_0, \ldots, w_L$, where $w_i \sim d_l$, and utilities $u_1, \ldots, u_L$, average feature activation $f_1, \ldots, f_L$, and ages $a_1, \ldots, a_L$ to 0

for each input x_t **do**
  pass input through network, get prediction $\widehat{y}_t$ (forward pass)
  receive loss $\mathcal{L}(x_t, \widehat{y}_t)$
  update weights with stochastic gradient descent (backward pass)
  for layer ℓ in $1 : L$ **do**
    update age a_l++
    update feature utility $u_{\ell,r,t}$
    find features with age more than m
    replace $n_l * \rho$ of eligible features with smallest utility, with indices r
    reset input weights $w_{\ell-r}[r]$ with samples from d_ℓ
    set $w_\ell[r] = 0$
    set $u_{\ell,r,t}$, $f_{\ell,r,t}$, $a_{\ell,r,t}$ to 0
  end for
end for

The general idea of CBP is to identify the 'least important' or 'lowest utility' features over the last few forward passes, and reset these features to have values from some distribution. In CBP, a feature in a layer is considered to have low utility if it has contributed relatively little to the input of the next layer, on average over the last few training examples, and low utility features are reset to have

values from the initial weight distribution. The measure of importance in CBP is a local one since it only considers the contribution of features in a layer to the next layer, and the authors state that one direction for extension of their work is to propose a global measure of importance. The authors also state that their tester is based on heuristics, so a more principled tester is desirable. We propose some modifications to CBP below, which pertaining to these two limitations of the current algorithm.

3.1 Natural Importance

Intuitively, a feature has high utility if predictions are significantly worse when it is absent. To implement this exactly, we can do M forward passes on a training example, each time with one of the M features removed. Then we can reset the features which resulted in the lowest loss when they were excluded, on average over the last few training examples.

Let our loss for a training example be $\mathcal{L}(\Theta_t)$, where Θ_t represents the parameters of the network. To compute the loss when feature i is absent, define a function g which "masks" the contribution of feature i. Denoting this function as $g(\Theta, i)$, the loss with feature i removed can be computed as $\mathcal{L}(g(\Theta, i))$. We can then define the relative importance of feature i in terms of the increase in the loss when feature i is removed:

$$J_i(\Theta_t) = \mathcal{L}(g(\Theta_t, i)) - \mathcal{L}(\Theta_t). \tag{7}$$

To compute $\mathcal{L}(g(\Theta_t, i))$ in practice, we set the appropriate columns in a weight matrix θ in Θ to zero and perform a forward pass. Then we let $u_{\ell,i,t} = J_i(\Theta)/(\sum_{j=1}^{n_{\ell-1}} |w_{\ell-1,j,i,t}|)$ in Equation 4.

Although this combinatorial search is prohibitively inefficient, we believe it is interesting to test the natural importance hypothesis that "a feature has high utility if predictions are significantly worse when it is absent". Moreover, the relative training loss when a feature is removed is a global measure of importance of that feature, rather than a local one. We also investigate a naïve relaxation of the combinatorial search, in which we only compute $J_i(\Theta)$ for a feature i with some probability $p \in [0, 1]$ for each training example. For the features for which we don't compute $J_i(\Theta)$, we let $u_{\ell,i,t} = 0$.

3.2 Taylor Approximation

To estimate the value of an objective function J when some of the weights in a parameter are set to zero, we can use a Taylor approximation. If e.g. the parameters of our network are $\Theta = (W^{(1)} \in \mathbb{R}^{5\times 20}, b^{(1)}, W^{(2)} \in \mathbb{R}^{1\times 5}, b^{(2)})$, then the first-order Taylor approximation of the value of J when the first feature in the hidden layer is excluded is given by

$$J_i(\Theta) = J(\Theta') \approx J(\Theta) + \frac{\partial J(\Theta)}{\partial W^{(2)}}(\tilde{W}^{(2)} - W^{(2)})^T = -\frac{\partial J(\Theta)}{\partial W_{11}^{(2)}} W_{11}^{(2)},$$

where $\Theta' = (W^{(1)}, b^{(1)}, \tilde{W}^{(2)}, b^{(2)})$ and $\tilde{W}^{(2)} = W^{(2)} \odot (0, 1, 1, 1, 1)$. Then as above, we let $u_{\ell,i,t} = J_i(\Theta)/(\sum_{j=1}^{n_{\ell-1}} |w_{\ell-1,j,i,t}|)$ in Equation 4. We propose this method as a more computationally efficient realisation of natural importance, so we don't include second and higher order terms in our Taylor approximation. We note that Molchanov et al. [2019] also use a Taylor expansion to estimate the importance of a feature, but they approximate e.g. the squared difference $(J(\Theta) - J(\Theta'))^2$, whereas we approximate $J(\Theta')$ directly.

We believe our approach is more appropriate for the continual learning problem. We observed that the loss for a training example can decrease when a particular feature is removed, especially just after a change in the input distribution. Such a feature has low utility under the natural importance assumption, yet the squared difference $(J(\Theta) - J(\Theta'))^2$ could be the same for both this feature and a feature for which the loss for that training example increases when that feature is removed. I.e., their formulation implicitly assumes that $J(\Theta)$ is the ground-truth, but we believe it should be zero.

3.3 Step Sizes

An alternative global measure of utility can be obtained by considering the size of the update $\theta_{t+1} - \theta_t$ for each parameter θ in the network. The motivation is as follows: a parameter θ_t for which the step $\theta_{t+1} - \theta_t$ over the last few training examples has been small may be a 'non-plastic' parameter, or one that no longer learns. This would mean that it would be a likely candidate for resetting. This

measure of utility is not directly related to the idea of natural importance described in the previous sections, as this approach focuses more on getting rid of non-plastic features rather than low-utility features. To formulate this mathematically for a parameter θ_t, we let

$$u_{\ell,i,t} = \frac{||(\theta_{t+1} - \theta_t)_i||_2^2}{\sum_{j=1}^{n_{\ell-1}} |w_{\ell-1,j,i,t}|}$$

in Equation 4, where $(\theta_{t+1} - \theta_t)_i$ is the i-th column of $\theta_{t+1} - \theta_t$.

4 Experiments and Results

We test our algorithms on the *slowly changing regression* problem. This setting differs from a traditional supervised learning problem in two ways: we do not assume that the training examples are i.i.d., and we also assume that the target function is strictly more complex than the learner network. As the input distribution is non-stationary, the best possible approximation continually changes.

The input at a time step t is represented by $\mathbf{x}_t \in \{0,1\}^m$, where $x_{i,t} \in \{0,1\}$ for each $i \in 1, \ldots, m$. After every T time-steps one of the first f bits is flipped at random; these f bits are constant at other times. This event corresponds to a shift in the input distribution. Each of the next $m - f$ bits is randomly sampled from $U[0,1]$ at every time-step. We choose $m = 20$, $f = 15$, and $T = 1e4$, and test the ReLU activation function on all algorithms. In our feed-forward neural network, all weights are initialised using He initialisation. Updates to weights are done using SGD. The experiment was run 1M times, where for each algorithm we performed 10 independent runs and averaged the results. All of our hyperparameters were chosen to be consistent with the experiments performed in Dohare et al. [2022].

Below, we provide plots for the performance of each of our algorithms against three baselines: (1) backpropagation with ReLU, (2) a linear classifier, and (3) continual backprop.

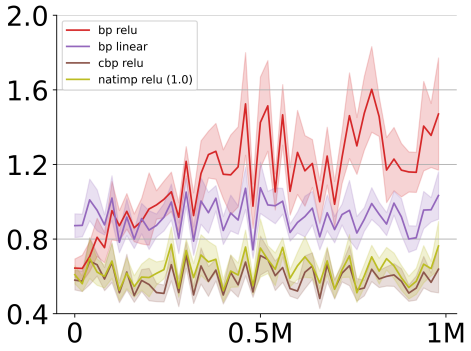

Figure 1: Natural Importance vs. Baseline

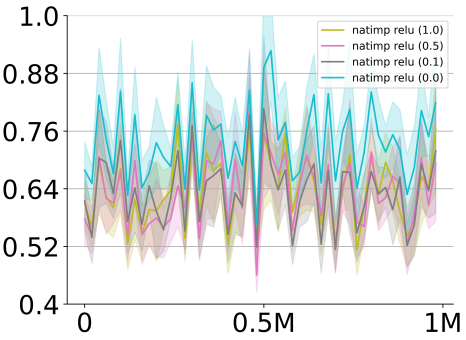

Figure 2: Sparse Nat. Importance Variation

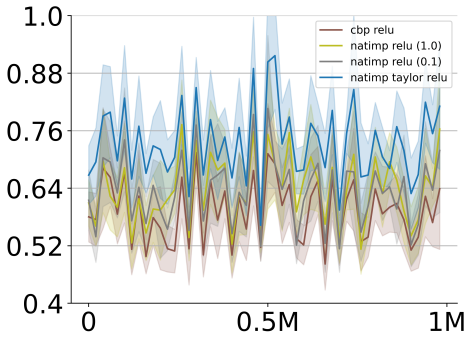

Figure 3: Taylor Nat. Importance vs. CBP

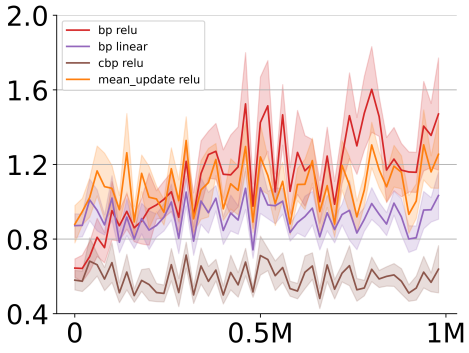

Figure 4: Step Size vs. Baseline

5 Discussion

As we can see in Figure 1, natural importance outperformed both linear regression and backpropagation, while it performed slightly worse than CBP. We believe that this could be due to a variety of reasons: CBP may be better at handling noise, and since the natural importance approach focuses on the impact of individual features and tends to retain features that individually reduce the error compared to CBP, this could have resulted in a higher variance (but lower bias) model.

In Figure 2, we show the effect of varying p, the probability with which we compute $J_i(\Theta)$ for a feature i for each training example. Setting p as low as 0.1 is sufficient to achieve performance similar to when $p = 1$, and the latter is much more computationally expensive. Figure 3 compares all variants of natural importance against each other and CBP. Figure 4 shows that the step size utility modification to CBP does better than backprop, it is out-performed by a learner network with linear activations. However, the mean squared error of the step size utility algorithm does not increase over time, and in this sense it has accomplished its goal of not losing plasticity.

Since random replacement (which corresponds to natural importance with $p = 0$) performed so well, we suspect that the slowly-changing regression problem is not complex enough to discriminate between naïve and intelligent approaches. Moreover, the network used in our experiments has only one-hidden layer so the importance measure in CBP is global here. This means that our proposed global importance measures are unable to demonstrate this strength of theirs, if it is indeed present.

5.1 Runtime Analysis

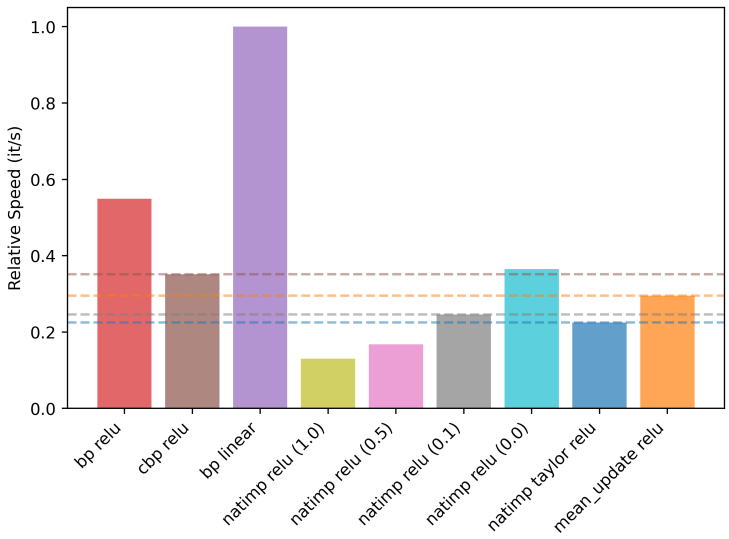

Figure 5: Runtimes of Algorithms

In Figure 5.1, we see the runtimes of each of the algorithms and their variants. We note that the Taylor approximation does indeed speed up the natural importance algorithm, albeit not without sacrificing accuracy. We were only able to achieve a runtime as low as CBP with random replacement, but this method cannot compete with CBP on accuracy.

6 Conclusion and Future Work

While our approaches do not strictly outperform CBP, they do significantly better than traditional backprop, which satisfies our goal of maintaining plasticity in a continual learning setting.

The next step would be to test our algorithms in more complex settings such as the permuted MNIST problem, which would require a deeper network, or predicting the parameters for a Gaussian distribution whose true parameters shift over time. It is also interesting to see how they perform in RL settings, such as the Slippery Ant problem in PyBullet. Other work that can be done is to extend the method to work with Adam-style optimisers instead of just SGD, and explore the effects of using different activation functions. Due to time constraints related to processing power, we only tested with the hyperparameters used in Dohare et al. [2022] in order to be able to compare results.

7 Contributions

We both contributed equally on the coding/theory aspects; we brainstormed the ideas for our algorithm together and also implemented it together. We also contributed equally to the writeup.

References

Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online normalization for training neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Shibhansh Dohare, Richard S. Sutton, and A. Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness, 2022. URL `https://openreview.net/forum?id=86sEVRfeGYS`.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272, 2019.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.