Prototyping biologically-inspired algorithms for continuous real-time onboard learning

Final Report

Authors: Matthew Yedutenko[1], Dominik Dold[2], Alexander Hadjiivanov[2], Dario Izzo[2], Guido de Croon[1]
Affiliation: [1]MAVLab, Faculty of Aerospace Engineering, TU Delft, [2]ESA ACT

Date: 19/03/2025

ACT research category: Biomimetics / Artificial Intelligence

Contacts:

Prof. Guido C.H.E. de Croon
Tel: +31-152781402
Fax: N/A
e-mail: g.c.h.e.decroon@tudelft.nl

Leopold Summerer (Technical Officer)
Tel: +31(0)715654192
Fax: +31(0)715658018
e-mail: act@esa.int

Available on the ACT website
http://www.esa.int/act

Ariadna ID: 24-8501

Ariadna study type: Standard
Contract Number:
4000143984/24/NL/GLC/my

Introduction

Space vehicles operate in environments that are not known in advance, under severe energy constraints and with long delays in signal communication with Earth. To perform efficiently in such conditions, spacecrafts must possess some sort of artificial intelligence (AI) that would allow to adapt to the specifics of environment and task at hand.

Deep Neural Networks (DNN) are the most powerful and scalable modern AI systems[1] and therefore have enormous potential for space missions. However, offline-trained DNNs poorly generalize to unseen scenes. An instructive illustration of this failure is shown in Figure 1. A deep neural network[2] trained to estimate depth on outdoor images (left) struggles to correctly estimate depth (right) in an indoor scene (center) despite strong similarities in the scene composition. Hence, adaptation to the environment and task condition requires a mechanism for real-time learning in DNNs.

The main issue of real-time learning in DNNs is the limitations of backpropagation of error (backprop) — the most effective training algorithm for offline learning. Backprop cannot be executed in real time under strict latency, energy, and memory constraints[3]. As a result, backprop in the standard implementation is unsuitable for edge applications such as robots and spacecrafts.

There are two major issues with the backprop when it comes to the real-time learning: the weight transport problem and the update locking problem[3]. The weight transport problem results from the backprop assumption of symmetrical feedforward and feedback connections, with weight updates in one layer explicitly depending on the weights in all downstream layers. This leads to significant memory overhead. The update locking problem stems from the requirement that the forward (activation) and backward (calculation of error of gradient) passes must be fully completed before weights can be updated. This also leads to significant memory, energy and latency overhead.

There exist algorithms for real-time learning[3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16] that yield massive (up to 3 orders of magnitude) gains in energy efficiency[17], [18], [19], when implemented in brain-inspired neuromorphic hardware[3], [6], [7], [9], [12]. Yet, they are not up to learning complex tasks onboard, under tight performance constraints. Most of these algorithms perform online learning only at the output layer[5], [6], [7], [9], [10], [12], [14], or, at most, with a single hidden layer[4], [11], [13]. They often rely on correlation-based methods that do not guarantee convergence[7], [9], [13], suffer from the weight transport problem[4], [15], [16], [20], or include alternating inference and learning phases (albeit short) and thus are not truly online[5], [10], [14], [15]. Moreover, many of these algorithms[3], [7], [11], [12], [15], [16], [20] have been applied to

classification tasks, while real-time learning is especially crucial in regression tasks such as ones where a control strategy needs to accommodate to changes in the environment to avoid a crash. Along the same lines, only a few methods have been used in conditions where energy-efficient real-time learning is paramount such as onboard a drone or other edge applications[3], [5], [6], [10], [13], [14].

Figure 1. Even with large, varying training data sets, learned capabilities may generalize poorly to a robot's actual real-world environment. Left: Image used in ref[2]. Middle: image from our lab at TU Delft. Right: application of the trained deep network from ref[2] on the lab image. Colormap indicates that purple / blue represents large distances, while red/yellow is close by. The distance to the large pillar (the closest obstacle) is estimated very badly and would likely lead to a collision in a navigation task. Figure 1. Even with large, varying training data sets, learned capabilities may generalize poorly to a robot's actual real-world environment.

Since animals are the only known autonomous agents around that are certainly capable of real-time learning, it is natural to look to neuroscience for inspiration for a real-time learning algorithm. There are a few theories of how supervised learning with deep neural networks can be organized in the brain[21]. One particularly promising candidate is called burst-dependent synaptic plasticity or burstprop[22]. Burstprop elegantly avoids weight transport and update locking problem by running inference and learning signals in two parallel streams and keeping weight information local (see Methods for the details).

In this study, we evaluate the performance of burstprop on a task essential for efficient and safe (crash free) visual-based navigation in moving systems ranging from bee to spacecrafts: learning the distance to a surface based on its appearance. This information is essential for navigation as it enables the disentangling of velocity from distance in optical flow, thereby preventing self-induced oscillations and collisions with obstacles, including the landing surface (see Methods for details). The universality of this task allows us not only to assess burstprop's performance in real-world conditions, but also ensures that the results can be generalized beyond the specific conditions of the present study.

We compare burstptop's performance with conventional algorithms, namely: linear ridge regression, backprop, feedback alignment[23] (see Methods for the details), and burstprop augmented with the Kollen-Pollack algorithm that ensures weight symmetry through local learning[22], [24] (see Methods for the details). Our key metrics for this comparison are: i) quality of learning an association between visual features and surface distance; ii) dependence of the performance on the number of surfaces

needed to be simultaneously stored in the network memory; iii) generalization to the unseen surfaces.

Our results demonstrate that burstprop with the Kollen-Pollack algorithm shows the best performance for distance estimation for a single surface, while burstprop with random feedback weights provides the best performance for multiple surfaces. However, all of the methods fail to generalize to unseen surfaces. Despite the promising potential of burstprop in our simulation, the ultimate test of performance will be the air vehicle's ability to achieve a smooth landing. This aspect will be explored in future research.

Methods

Errors, gradients and weight updates

Before discussing various training algorithms, it would be useful to understand computation performed by neural networks and how their performance can be tuned by means of error backpropagation. This will provide some intuition about information is transmitted during learning and what are the problems that burstprop aims to solve.

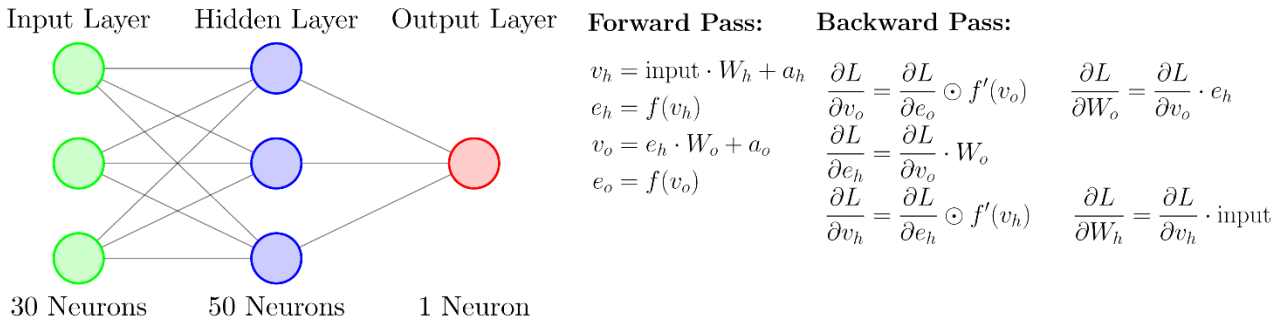

Figure 2 Schematic depiction of a neural network (left) together with equations to compute networks forward (center) and backward (right) passes.

Figure 2 schematically depicts a neural network with a single hidden layer. During the forward pass an input (row vector) is fed into the hidden layer that multiplies the input by weight matrix W_h and adds a bias a_h. The result is a weighted sum of the inputs v_h that is subsequently transformed into the hidden layer activation e_h by applying activation function f (Figure 2, Forward Pass). The same operation is repeated on the next layer, where e_h serves as the input and e_o is the final output of the network. To improve the performance of this network it is necessary to calculate the error using the loss function L and to adjust the network's weights and biases during the backward pass in order to minimize this error. A straightforward way to modify the network parameters is to compute the gradient of the error with

respect to the parameters and update them accordingly (Figure 2, Backward pass)[1].

Examining the equations for the backward pass in Figure 2 reveals several problems with backprop. First, the gradient of the error in the hidden layer explicitly depends on the error gradient in the output layer. While this is not a big problem for a network with one hidden layer, such global dependencies would be prohibitively expensive in terms of energy and memory if the network has more layers, as the gradient of the error in the very first layer would explicitly depend on the gradient of the error in all downstream layers. All of these gradients have to be computed, stored in memory and multiplied with each other during the backward pass. This is one side of the so-called weight transport problem.

There is another problem with the weight transport problem: the weights through which error gradients propagate, the derivatives of the pre-activation value v with respect to the input to a layer, are the transpose of the feedforward weights. This creates issues for real-time learning in edge applications as maintaining perfect symmetry between feedforward and feedback weights requires either centralized control or additional hardware for precise synchronization. This is not achievable without a significant overhead in highly promising neuromorphic hardware, platforms that operate locally and asynchronously.

Finally, the update locking problem arises from the shared use of the communication channel for both forward and backward passes. This temporal constraint locks training in time, requiring strictly sequential forward and backward phases. Consequently, all activations from the forward pass must be stored to compute gradients during the backward pass, significantly increasing the memory overhead. This locking mechanism not only delays updates but also makes simultaneous learning and inference impossible, further reducing efficiency in real-time or hardware-constrained systems.

Burstprop

From the discussion above, it is clear that to solve backpropagation inefficiencies, it is necessary to make computations local. The locality of computation removes global dependencies, allowing greatly reduced energy, memory and latency overhead and streamlined learning in real-time systems.

Although conventional implementations of the burstprop algorithm compute gradient globally, the chain rule for the gradient computation can be re-written in recursive form, such that the error gradient at hidden layer l is an explicit function of the error gradient at layer $l+1$ only:

$$\frac{\partial L}{\partial v_l} = \left(W_{l+1}^T \cdot \frac{\partial L}{\partial v_{l+1}}\right) \odot f_l'(v_l) \quad (1)$$

$$\frac{\partial L}{\partial W_l} = \frac{\partial L}{\partial v_l} \cdot e_{l-1} \quad (2)$$

Thus, in principle one can keep gradient computations local. Therefore, a lot of research on alternatives to the backprop algorithm is focused on developing learning rules that allow robust learning, while keeping gradient computations local.

The brain also uses neural networks, faces the weight transport and update locking problems and, as evident from day-to-day life, is able to solve both of these problems. Hence, it is useful to study bioinspired solutions for hierarchical credit assignment. Burstprop is one of the algorithms that have been proposed to explain learning in the brain. Burstprop cleverly avoids the weight transport and update locking problems[3] by employing key properties of biological neurons: bursts, local synaptic plasticity, and separate signal integration in basal (feedforward inputs from shallower layers) and apical (feedback inputs from deeper layers) dendrites[22]. Below we give a short account of how burstprop achieves real-time learning across multiple layers (Figure 3).

There are two types of electrical output signals in biological neurons: spikes and bursts[25], [26]. Spikes are all-or-none rapid changes in neuronal membrane potential that mediate signal transmission in the brain[27]. In contrast, bursts are simply occurrences of multiple (at least 2) spikes within short intervals of time[22], [25]. These bursts participate in local learning[22], [28](see Figure 3). When an input spike causes a burst of output spikes, the connectivity between input and output neurons strengthens, while when there is only a single output spike, the connectivity weakens. To be more precise, the change in connectivity averaged over time would depend on whether bursting occurs more or less frequently than a certain baseline burst probability[22]:

$$\delta W_l = \left(b_l - \overline{b}\right) \cdot e_{l-1} = (p - p_b) \cdot e_l \cdot e_{l-1} \quad (3)$$

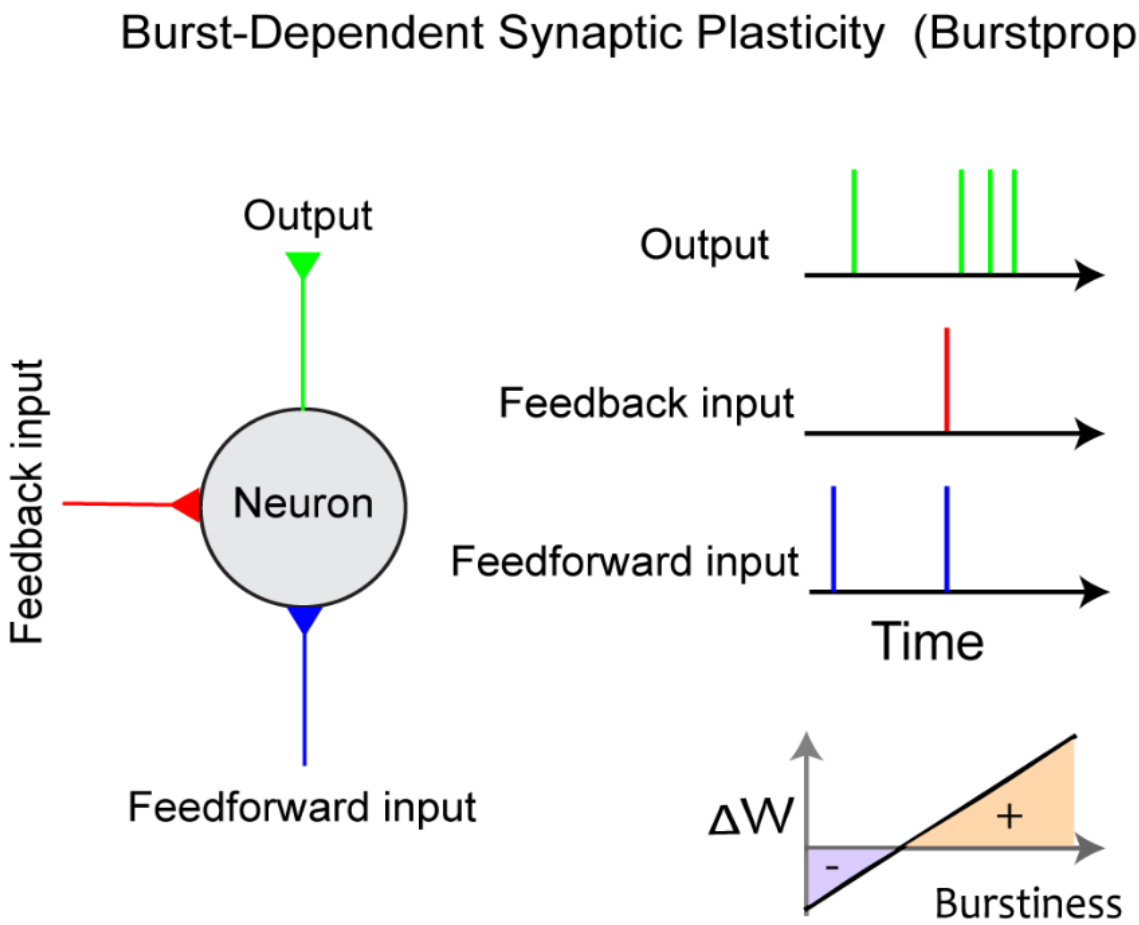

Figure 3. Burst-dependent synaptic plasticity. (Left) A two compartment neuron that separately integrates feedforward and feedback inputs. (Right) A schematic representation of burstprop. When an input spike causes only a single output spike, the connectivity weakens.

Whether there will be a spike or a burst depends on the feedback inputs to the apical dendrite: one hypothesis is that coordinated feedforward and feedback inputs lead to bursts and that plasticity in the feedforward pathway is mediated by the spread of electrical excitation from apical to baseline dendrites[22]. Learning in deep networks can be achieved via a feedback pathway. Moreover, a supervisory (or self-supervisory) signal applied to the output layer of the network would trickle down downstream along the feedback pathway[22].

The crucial point about separating signals into spikes and bursts is not the sign of local plasticity but the solution of the update locking problem via the multiplexing of the feedforward and feedback signals[22]. To multiplex means to send multiple signals within a single transmission channel. In the case of neural networks, this channel is neuronal output. The conventional algorithm of backpropagation runs into the update locking problem due to multiplexing a signal in time: alternating inference and learning phases. Burstprop avoids the update locking problem because multiplexing is done by two types of signals - bursts and spikes, with bursts being proportional to feedback inputs and

spikes to the feedforward inputs[22].

Such arrangement effectively allows the neural network to compute the error gradient and weight update in a recursive manner[22], [29], [30]. A mathematical formulation of burstprop is presented below.

The signal flow through basal dendrites (feedforward pathway) is identical to the forward pass shown in Figure 2, therefore we focus on the backward pass (feedback pathway). Neurons in a layer *l* have a dendritic potential *u*, which is a function of bursting input from neurons in layer *l+1* weighted by weight matrix *Y* and some function of the neuron's feedforward (or event) output $g(e_l)$. We define the output $g(e_l)$ to be equal to the ratio of the derivative of a neuron's activation function to its activity:

$$u_l = Y_l \cdot b_{l+1} \odot g(e_l) \quad (4)$$

$$g(e_l) = \frac{f'(v_l)}{e_l} \quad (5)$$

The sigmoid of the dendritic potential u_l yields burst probability p_l, which is multiplied with event output e_l to generate burst output b_l:

$$p_l = \sigma(u_l), b_l = p_l e_l \quad (6)$$

We assume that the change in burst-rate is equal to error gradient with respect to neural pre-activation:

$$\delta b_l = \frac{\partial L}{\partial v_l} \quad (7)$$

Then, the burstprop weight update rule (3) would be very similar to the backprop weight update rule (2):

$$\delta W_l = \delta b_l \cdot e_{l-1} \quad (8)$$

To ensure that burstprop is also able to propagate the error gradient, we combine $\delta b_l = \delta p_l \cdot e_l$ and equation (6) to get equation (9):

$$\delta b_l = \delta p_l \cdot e_l = \delta\big(\sigma(Y_l \cdot b_{l+1} \odot g(e_l)\ )\big) \cdot e_l \quad (9)$$

Now, assuming a linear range of σ outputs and keeping in mind equation (5) we arrive at equation (10):

$$\delta b_l = Y_l \cdot \delta b_{l+1} \odot f'(v_l) \quad (10)$$

Since in equation (7) we define the change in burst rate as the gradient of the error with respect to neural pre-activation, equation (10) shows that burstprop indeed supports the propagation of error gradients up to a weight factor Y_l. When Y_l is equal to W^T_{l+1} equation (10) becomes identical to equation (1).

Note that although the assumption about the approximately linear range of $\sigma(u_l)$ can be enforced by additional mechanisms[22], here we opted not to do so, because it has been shown that even in the non-linear regions of the sigmoid function equation (9) approximates (1)[29].

To fully mimic backrprop, we also need feedback weights Y_l to be equal to W^T_{l+1}, which is not trivial to ensure in the brain and neuromorphic hardware. There are two potential solutions that rely on local mechanisms: feedback alignment[23] and the Kollen-Pollack algorithm[24]. Here, we implemented both of these solutions and compared their characteristics.

The feedback alignment algorithm was proposed in a study by Lillicrap et al.[23], who showed that it is not necessary to propagate exact gradients, and that even random set of the feedback weights would lead to efficient learning. In fact, studies have shown that proper alignment of error sign[31] is the most crucial information for learning and feedback alignment does just that. Indeed, although in the feedback alignment errors are sent via random weights, these random weights contribute to the update of feedforward weights. Hence, the network learns to change its feedforward weights such as to minimize the error transported through the random feedback matrix.

The Kollen-Pollack algorithm was proposed[32] and further modified[24] to ensure symmetry between feedforward and feedback weights. It has been shown[24] that one can ensure symmetry between feedforward and feedback weights if i) weights have equal update A and ii) weights have equal decay λ to offset any initial difference in feedforward and feedback weights:

$$\delta W_l = A - \lambda W_l$$

$$\delta Y_l = A - \lambda Y_l$$

Therefore, we update weights Y_l following ref.[22]:

$$\delta Y_l = e_l^{\top} \cdot \left(b_{l+1} - \overline{b}_{l+1}\right) - \lambda Y_l \quad (11)$$

Case study: learning an association between visual appearance and distance

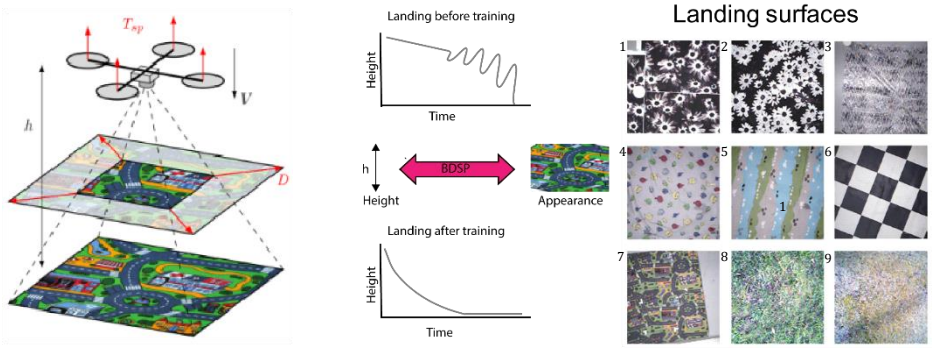

Figure 4. Case study: Learning an association between visual appearance and distance

As a case study to test burstprop's performance in the robotics context we adopted an approach developed by de Croon et al. [5] and trained a network to learn the distance to a surface from its visual appearance in a visual-based landing task. The eventual goal of the training was to augment classical optical-flow based navigation strategy[33] with a distance estimation mechanism to avoid self-induced oscillations in the final landing stage. The benefits of our approach were two-fold. On one hand, we tested a bioinspired learning rule on a biologically relevant task. On the other hand, the task has immediate practical application for any air vehicle, including spacecraft.

The classical vision-based landing strategy[33] relies on optical flow - changes in scene contrast distribution due to relative motion between scene and observer. Specifically, the rate of image expansion, also known as divergence of optical flow, is very useful for landing. Indeed, divergence of optical flow is equal to the ratio between the speed with which a surface is approached and the distance to this surface. Hence, by keeping the divergence constant it should be possible to safely land as the velocity decreases upon approach.

However, in practice, delays between sensing and actuation lead to self-induced oscillations close to the landing surface (Figure 4)[34]. If not addressed, these oscillations would not allow the drone to actually land, leading to a crash. To avoid oscillations, it is possible to decrease the gain that is used in the control policy upon getting closer to the landing surface. The dilemma here is that a low gain does not ensure that the agent follows the policy, while a high gain leads to the oscillations. Thus, the gain should be properly adjusted based on the distance to the landing surface[34].

To achieve this, de Croon et al.[35] proposed to learn the distance to a surface from its visual appearance

(Figure 4). Since the distance at the onset of self-induced oscillation depends linearly on the control gain[34], one can infer the relationship between the visual appearance of a surface and the distance to it by learning an association between visual features of the surface and the value of the control gain at the onset of oscillation. As a result, air vehicles are able to regulate the control gain depending on the distance to the surface to achieve smooth landing (Figure 4). de Croon et al.[35] showed how one can use linear regression to achieve onboard learning of association between visual features and the distance to a surface.

Here, we compared the performance of five different training algorithms (linear regression, backprop, feedback alignment, burstprop with feedback alignment, burstprop with Kollen-Pollack algorithm). We used the dataset from de Croon et al.[35], which contained processed images and height values recalculated from the control gain values upon onset of self-induced oscillations, for 9 different surfaces. The processed images represented visual scenes as texton histograms. Each texton corresponded to a spatial pattern in the visual scene extracted using a model-based algorithm. Thus, each image was represented as a 30-dimensional vector, where each value corresponded to the prevalence of the corresponding texton in the visual scene (the so-called visual bag of words method[35]). Here, we focused on the comparison between different algorithms in simulation, while onboard implementations will be addressed in forthcoming studies.

Network

To align with memory and energy constraints of the real-time onboard implementation, we aimed to train a lightweight model with a single hidden layer of 50 neurons and one output neuron that reports height (Figure 2). For the hidden layer, we chose the sigmoid activation function. In the output layer, since we were interested in predicting absolute height, we selected a function capable of producing unbounded positive values. Specifically, we designed a custom-made function called SqRRL that provides leaky signal rectification to generate unbounded positive values and has a smooth derivative (Figure 5).

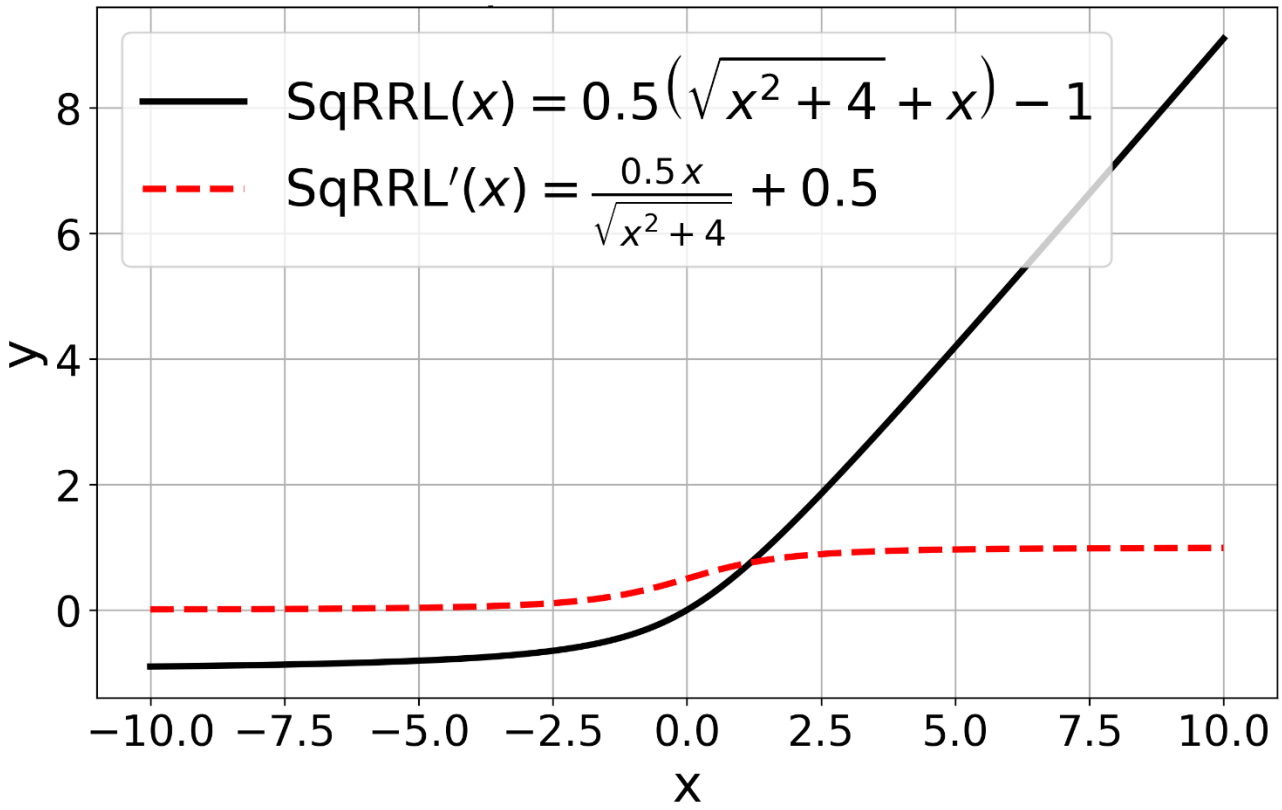

Figure 5. SqRRL function (black) and its derivative (red)

The loss function for the burstprop learning was computed as the difference between the output activation e_o and the ground truth height. The output layer had a fixed baseline input probability p_b during the inference phase. During learning, the error, which is calculated as the difference between the network output and the ground truth, is used to compute the output burst probability p_o using the squashing function inspired by previous implementations[22], [29]:

$$p_o = p_b \left(\frac{(\text{target} - e_o) \cdot \text{SqRRL}'(e_o)}{e_o} + 1 \right) \quad (12)$$

p_o was further clamped to ensure that it is always between 0 and 1. Note that (*target*-e_o), up to a scaling factor, equals the gradient of the error with respect to e_o when the mean squared error (MSE) loss function is employed. Therefore, in the non-burst implementation (backprop and feedback alignment), we employed MSE as the loss function.

Controls

To gain insight into the mechanics of learning with burstprop, we performed experiments using five

training algorithms: ridge linear regression, backprop, feedback alignment (FA), burstprop with feedback alignment (burst+FA), burstprop with Kollen-Pollack algorithm (burst+KP). A comparison of the results achieved with different algorithms served to highlight the importance of i) deep learning (linear regression vs. other algorithms), ii) precise gradients (FA/burst+FA vs. backprop/burst+KP), iii) burst-like signal processing (backprop/FA vs burst+KP/burst+FA).

Results

To evaluate the performance of various training algorithms, we conducted three types of experiments. First, we assessed how effectively a training algorithm could learn an association between height and texton distribution for a single landing surface (Figure 6). This represents the minimum required performance for an onboard AI system.

Next, we gradually increased the number of surfaces in the dataset in order to test the ability of the model to learn an association between height and visual appearance for various surfaces. Compared to learning for a single surface, this task demands a higher level of generalization from the learning algorithm. For a single surface, a decrease in distance typically corresponds to an increased prevalence of one or two textons[35]. However, for another surface, the dominant textons could differ entirely. Thus, the network must either i) learn a general principle - where decreasing distance leads to peaks in the texton histogram; or ii) memorize each specific texton distribution-height pair. Both tasks pose significant challenges.

Finally, we examined whether the learning algorithm enables the network to generalize to the principle that “shorter distances result in sharper peaks in the texton distribution”[35] by testing its performance on unseen surfaces.

Learning of an association between distance and appearance: single surface

In the simplest possible case, the network should learn an association between distance and texton distribution for a single landing surface. To compare training with different algorithms, we performed tests with various randomly selected surfaces. In each experimental run, we randomly selected 90% of the data for training, while the remaining 10% served as a testing dataset. In each experimental run, the neural network was initialized and trained *de novo*. In total, we performed 100 experimental tests for ridge linear regression, 50 for backprop, and 20 tests for FA, burst+FA and burst+KP algorithms.

Figure 6A shows comparison of the mean absolute error (MAE) in distance estimation across for all

five training algorithms. Linear regression (purple) performed the worst among all of the algorithms, with median (across different experiments) MAE of 0.29 meters, highlighting the benefits of the deep learning for this task. Burstprop aided with the Kollen-Pollack algorithm (green) had the best performance with median MAE of 0.2m, while backprop (blue), FA (orange), and burst+FA (red) had similar median errors of ~0.25m. Burst+KP performed significantly better than backprop (p=2e-3, two sample t-test).

Figure 6B qualitatively compares the ground truth height with the height inferred using the burst+KP-trained network for a test dataset in one of the experimental runs. Apart from the highest distance of 6m, where the network estimate error is very high (~2.8m), the network output closely follows the ground truth.

Dependence of the error in distance estimation on the number of different surfaces in the dataset

Next, we tested how the performance of the network depends on the number of surfaces for which an association between distance and texton distribution needs to be learned. To do so, we randomly sampled *n* surfaces from the full dataset and allocated 10% of examples for testing and 90% for training. Just like in the experiments with a single surface, we performed multiple experimental runs in each of which we *de novo* sampled input data and initialized neural networks. In total we performed 100 experiments for linear regression and bakcprop, and 20 experiments for FA, burst+FA, and burst+KP for each *n (2 to 9)* selected surfaces.

The results achieved with each of the algorithms are compared in Figure 7. It shows that the performance of linear regression (purple) becomes consistently worse with the increase in number of surfaces, from median MAE of 0.3m (1 surface) to median MAE of 0.38m (9 surfaces). For backprop (blue) and FA (orange) the decrease in performance is less pronounced but similar: from MAE of 0.25/0.26m to MAE of 0.3/0.29m. Although burst+KP showed the best performance for a single surface, as soon as the number of surfaces increased to 6, its performance was indistinguishable from backprop and FA. For the burst+FA model (red) there were no significant differences between median MAE for 1 and 9 surfaces (p=0.32, two sample t-test). Overall, burst+FA had the best performance and performed significantly better than the backprop (MAE of 0.24m vs MAE of 0.3m for 9 surfaces, p=2e-30, two sample t-test).

To gain qualitative intuition about distance estimation across multiple surfaces, we compared the ground truth and estimated height in Figure 8 for linear regression (Figure 8A), backprop (Figure 8B) and burst+FA (Figure 8C). The left side of Figure 8 shows the performance in terms of absolute values, therefore there we sorted values from highest to lowest ground truth height value. The right side of Figure 8 focuses on correlations between estimated and ground truth height. Therefore, there we plotted height values in a so-called “chronological order”, i.e., following the order in which they appear in the testing dataset.

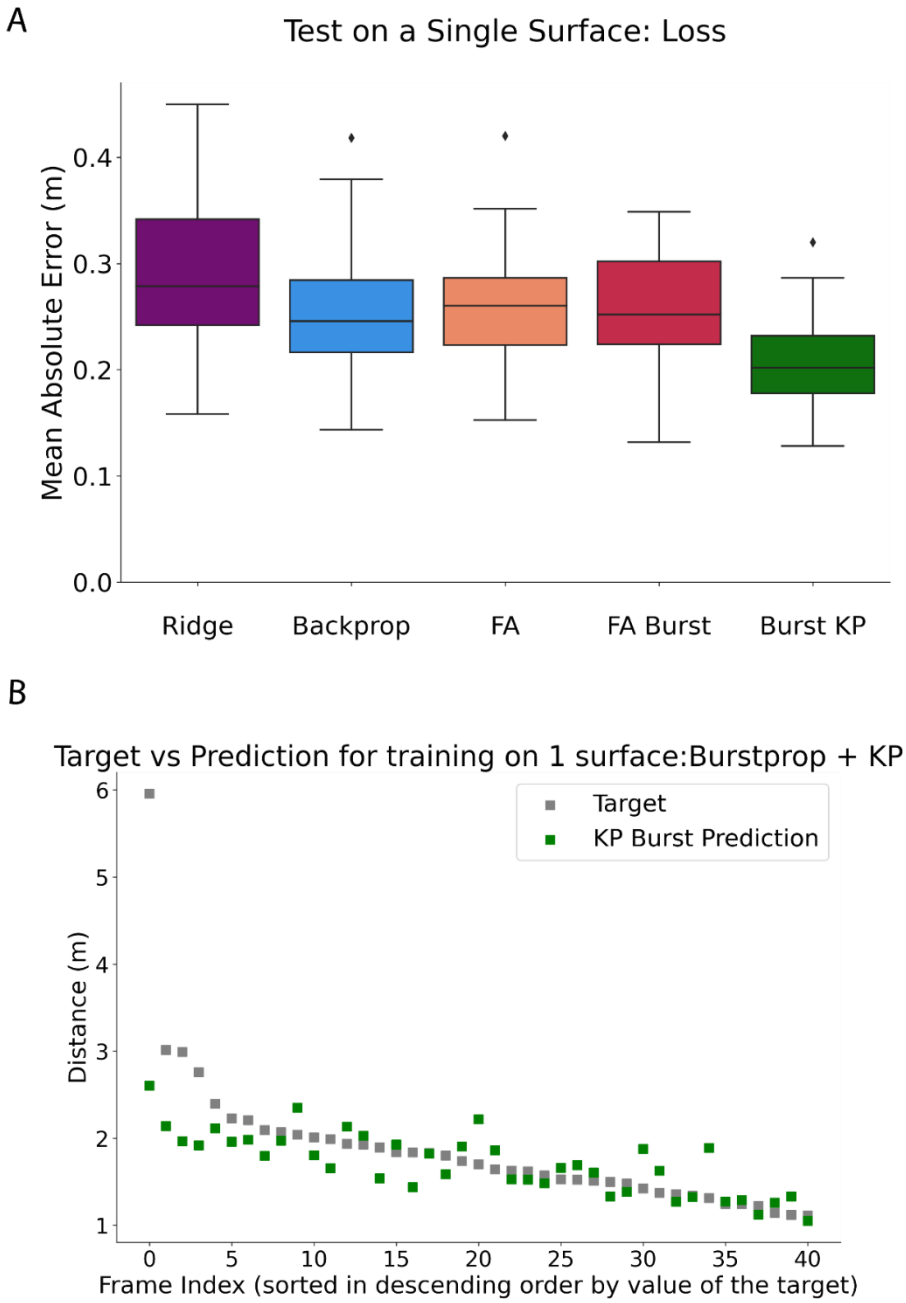

Figure 6. Learning an association between distance and texton distribution. (A) MAE for different algorithms. (B). Qualitative comparison between ground truth and the output of the network trained with Burst+KP and ground truth

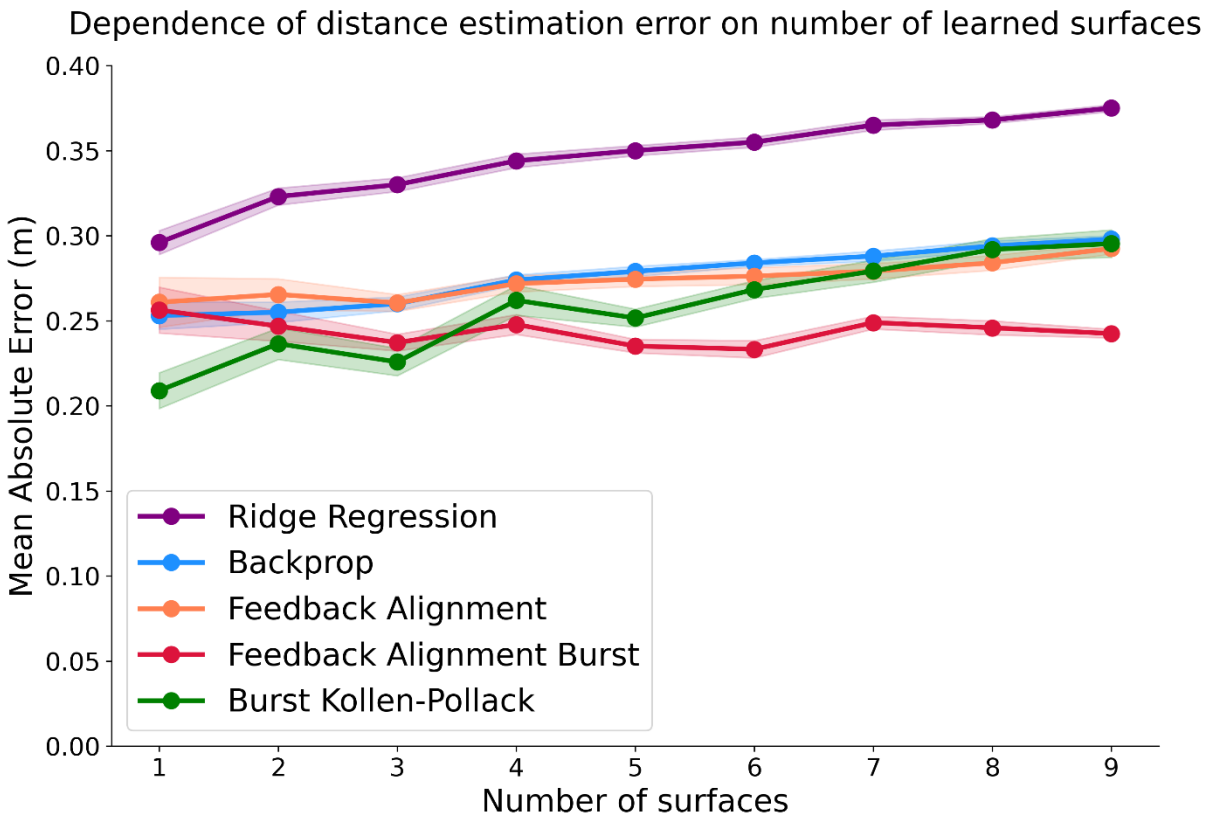

Figure 7. Dependence of MAE in height estimation on the number of different surfaces in the dataset

The left side of Figure 8 shows that the neural network trained with burst+FA performs best at estimating absolute distances. Apart from heights in the range of 3.5–4 m, the network's outputs closely follow the ground truth height (Figure 8A). Some variance in the height estimates is present, but this is expected given the inherent complexity of the task. In contrast, the backprop-trained network performs noticeably worse, particularly for ground truth heights >2.5 m, but still manages to approximate the general trend in ground truth (Figure 8B). Linear regression performs the worst, with its corresponding plot appearing nearly flat, indicating that it provides a poor estimate of the ground truth height (Figure 8C).

Although estimation of absolute distance is important, sometimes it might be enough to obtain a rough understanding of whether the distance is increasing or decreasing. Therefore, to understand whether different training methods help the network learn not only the absolute distance, but also the general trend in the ground truth data, we plotted the ground truth and estimated the height in chronological order, i.e. as a function of a frame index in the testing dataset (Figure 8 right). The figure shows that for all three visualized algorithms the estimated output is more or less correlated with the ground truth output. For the quantitive evaluation, we calculated the correlation between the estimated and ground

truth height in the testing dataset for each experimental run in the experiments with 9 surfaces (Figure 9). Figure 9 shows that linear regression has the worst median correlation with the ground truth height (0.48), while burst+FA again has the best performance, with median correlation of 0.75, for the remaining three algorithms this correlation was 0.65.

Thus, deep learning methods helped the network to learn not only absolute scale of the distance, but also improved estimates of the general trend of changes in height upon change in visual appearance. Burst+FA method led to the best overall performance, regardless of the metric.

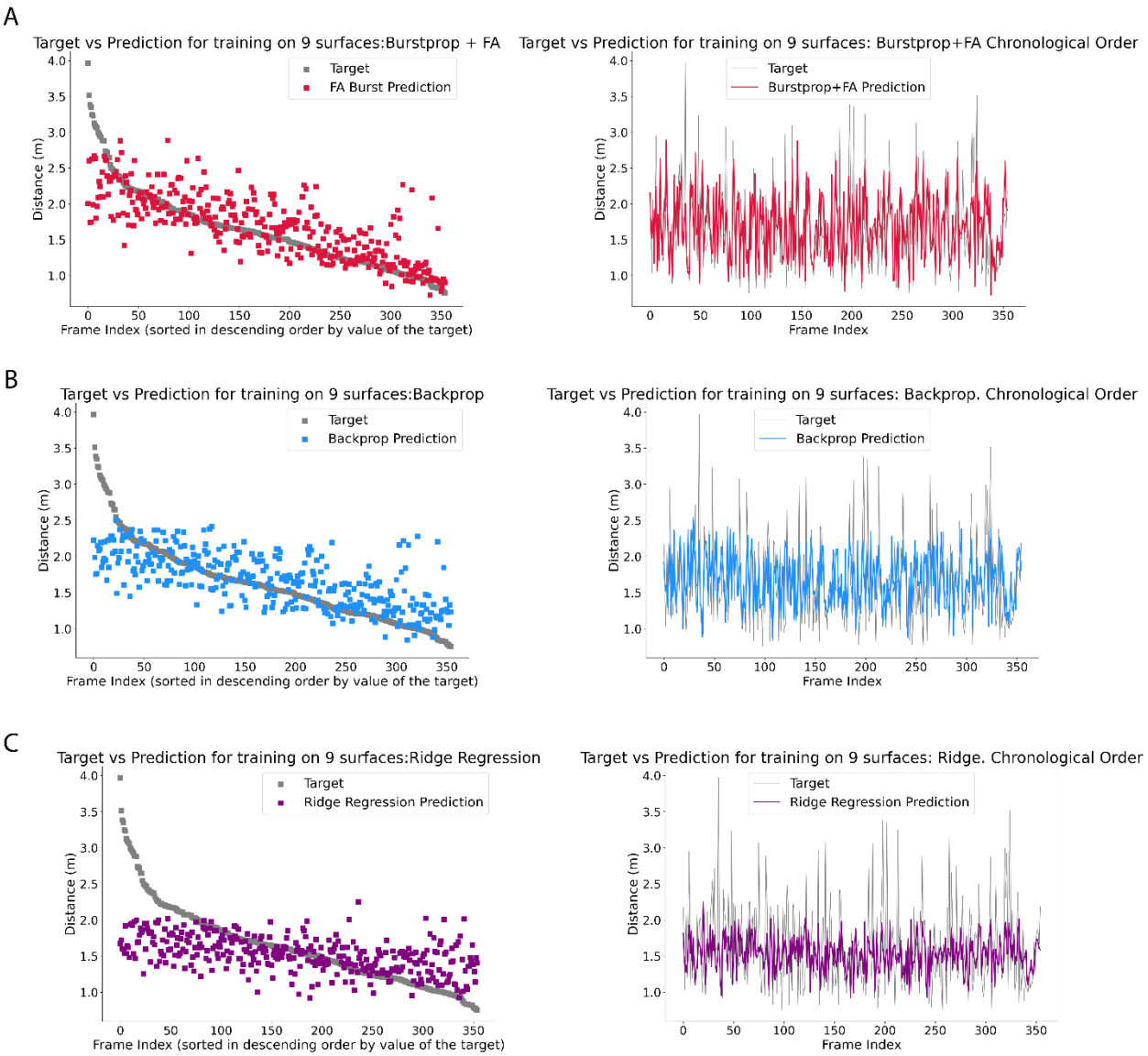

Figure 8. Distance estimation with burst+FA (A), backprop (B) and Ridge Linear Regression (C) for learning associations between height and texton distribution for 9 surfaces. Values sorted by ground truth height (left) and chronological order (right).

.

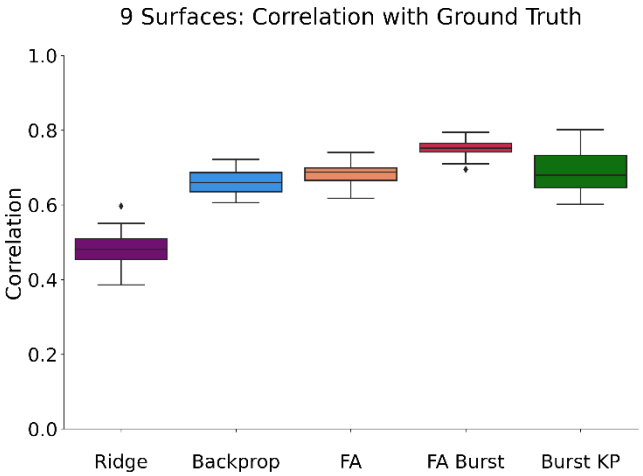

Figure 9. Correlation of the network output with ground truth.

.

Evaluation on an unseen scene

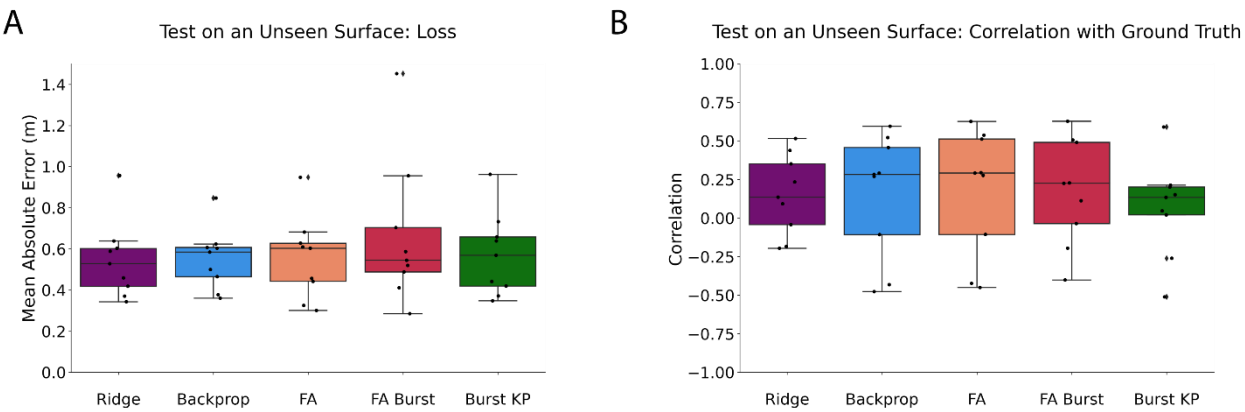

Figure 10. Summary of the tests on unseen surfaces. A - absolute error, B - correlation with ground truth.

Next, we tested how well the learned associations between distance and texton distribution generalizes to unseen surfaces. To do this, we performed 9 training iterations, each time excluding one surface from the training dataset and using that surface for evaluation. Figure 10 shows that overall all of the algorithms performed poorly in terms of absolute error (Figure 10A) and correlation with the target (Figure 10B).

However, in certain cases some generalization was present. Specifically, each of the algorithms showed the highest correlation with the ground truth for surface 4. The likely reason is that this surface is similar to many other surfaces in the dataset. In terms of texture, fish were similar to flower centers on surfaces

1 and 2, clouds on surface 5, and houses on surface 7.

Discussion

Here we explored burstprop as a potential algorithm for continuous real-time onboard learning. Specifically, we tested i) how well burstprop can learn an association between the distance to a surface and its visual appearance (Figure 6); ii) how this performance depends on the number different surfaces in the training dataset (Figures 7-9); and iii) whether training results generalize to unseen surfaces. We implemented burstprop in two versions: with random feedback weights (burst+FA) and with trainable feedback weights (burst+KP). We compared the performance of these two algorithms with linear ridge regression, backprop, and feedback alignment. We found that burst+KP had the best performance with learning associations between distance and visual appearance for a single surface, while burst+FA showed the best performance for learning distance-appearance associations for multiple surfaces. Yet, neither of the tested algorithms showed an ability to generalize to unseen surfaces.

Below we discuss why burstprop-based algorithms performed relatively well, the feasibility of its onboard implementation and their potential for space-based applications.

Reasons for burstprop's success

Our experiments demonstrate that burstprop-based algorithms outperform backprop, feedback alignment, and linear regression in experiments with both single (Figure 6) and multiple (Figures 7-9) landing surfaces. Intriguingly, while for a single landing surface the best performance was shown by the burst+KP algorithm, for the multiple learning surfaces the best performance was shown by burst+FA.

Why did burst+KP outperform backprop in learning an association between distance and visual appearance for a single learning surface (Figure 6)? Examining the equations of these algorithms (see Methods for the details) reveals that burst+KP is nearly identical to backprop due to the plasticity of its feedback weights. However, two key differences likely account for the observed performance gap: i) the learning rate and ii) additional non-linearities in the burst+KP backward pass. Specifically, the squashing function in the output layer (equation (11)) and the dendritic sigmoid function (equations (6) and (9)) introduce slight distortions to the error gradient, which may help avoid local minima.

Nevertheless, we believe that the main factor behind difference in performance is the learning rate. In backpropagation, the learning rate is the same for the output and hidden layers. In contrast, burstprop (both burst+KP and burst+FA) achieves optimal performance when the learning rate in the output layer is 1000 times higher than in the hidden layer. For example, our preliminary results showed that in tests with 9 surfaces, equalizing the learning rates in the output and hidden layers increased the MAE of the burst+FA trained network from 0.3m to 0.36m. A plausible explanation for the importance of a higher learning rate in the hidden layer is that this layer contains 50 neurons interacting with 30 input neurons, thus having much higher representational capacity than the output layer, which has only a single neuron.

However, different learning rates in the output and hidden layers cannot alone explain why with multiple surfaces burst+FA performs the best, as it has the same learning rates as the burst+KP, which performs significantly worse (Figure 7, p=2e-7, two sample t-test). A plausible explanation is that in the random feedback weights (the only difference between burst+KP and burst+FA) act as a form of regularization, helping the algorithm avoid overfitting to the training dataset.

Additionally, there appears to be a synergistic effect between learning rate asymmetry and random feedback weights. Neither feature alone - random feedback weights (FA vs. backprop) nor asymmetric learning rates (burst+KP vs. backprop) - achieved the performance observed with burst+FA. Together, however, these features create a complementary dynamic that drives the algorithm's success.
In future studies, we plan to investigate the importance of the learning rate in a more rigorous way.

Potential for onboard implementation and future directions

In the present work, we conducted experiments in simulation and evaluated algorithm performance using MAE. However, since the ultimate goal of learning associations between distance and visual appearance is safe landing, smooth landing is the ultimate criterion for an algorithm’s performance.

A study by de Croon et al. [35] demonstrated that for a single landing surface, a MAE of 0.3m achieved with linear regression is sufficient for a safe landing - at least for the specific drone used in that study. The performance threshold may vary for different vehicles. However, while linear regression performs adequately for single-surface tasks, it struggles when associations between distance and visual appearance across multiple surfaces need to be learned. This limitation underscores the potential

advantages of burstprop-based algorithms.

On the other hand, no algorithm tested so far performs well on unseen surfaces (Figure 10), highlighting the critical need for real-time onboard learning. From this perspective, the best algorithm is one that is easiest to implement directly on a drone. On conventional hardware, simulated on a laptop, linear regression holds an edge due to its short training time, which is on the order of seconds, whereas other algorithms take a few minutes[35].

However, we believe the future of onboard applications lies in neuromorphic hardware due to its unparalleled energy efficiency[17]. Compared to traditional algorithms, burstprop is far more compatible with such hardware. In future work, we aim to explore burstprop implementation on neuromorphic hardware, such as Loihi 2[18], and its deployment onboard a drone. This direction is especially promising given that the learning task explored here – mapping texton distributions to distance – is only a simple starting example. Real-world space applications involve far more complex challenges, such as learning to construct texton distributions from raw data or executing sophisticated control tasks.

References

[1] Y. Lecun, Y. Bengio, and G. Hinton, 'Deep learning', *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[2] M. Mancini, G. Costante, P. Valigi, and T. A. Ciarfuglia, 'Fast robust monocular depth estimation for Obstacle Detection with fully convolutional networks', *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, pp. 4296–4303, 2016, doi: 10.1109/IROS.2016.7759632.

[3] C. Frenkel, M. Lefebvre, and D. Bol, 'Learning Without Feedback: Fixed Random Learning Signals Allow for Feedforward Training of Deep Neural Networks', *Front Neurosci*, vol. 15, no. February, pp. 1–13, 2021, doi: 10.3389/fnins.2021.629892.

[4] G. Bellec *et al.*, 'A solution to the learning dilemma for recurrent networks of spiking neurons', *Nat Commun*, vol. 11, no. 1, pp. 1–15, 2020, doi: 10.1038/s41467-020-17236-y.

[5] G. C. H. E. de Croon, C. De Wagter, and T. Seidl, 'Enhancing optical-flow-based control by learning visual appearance cues for flying robots', *Nat Mach Intell*, vol. 3, no. 1, pp. 33–41, 2021, doi: 10.1038/s42256-020-00279-7.

[6] T. DeWolf, P. Jaworski, and C. Eliasmith, 'Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics', *Front Neurorobot*, vol. 14, pp. 1–15, 2020, doi: 10.3389/fnbot.2020.568359.

[7] N. Imam and T. A. Cleland, 'Rapid online learning and robust recall in a neuromorphic olfactory circuit', *Nat Mach Intell*, vol. 2, no. 3, pp. 181–191, 2020, doi: 10.1038/s42256-020-0159-4.

[8] V. Kornijcuk and D. S. Jeong, ‘Recent Progress in Real-Time Adaptable Digital Neuromorphic Hardware’, *Advanced Intelligent Systems*, vol. 1, no. 6, 2019, doi: 10.1002/aisy.201900030.
[9] J. Lindsey and J. B. Aimone, ‘Sequence Learning and Consolidation on Loihi using On-chip Plasticity’, *ACM International Conference Proceeding Series*, vol. 1, no. 1, pp. 70–72, 2022, doi: 10.1145/3517343.3517367.
[10] M. O’Connell *et al.*, ‘Neural-Fly enables rapid learning for agile flight in strong winds’, *Sci Robot*, vol. 7, no. 66, pp. 1–15, 2022, doi: 10.1126/scirobotics.abm6597.
[11] F. M. Quintana, F. Perez-Peña, P. L. Galindo, E. O. Neftci, E. Chicca, and L. Khacef, ‘ETLP: Event-based Three-factor Local Plasticity for online learning with neuromorphic hardware’, pp. 1–11, 2023, [Online]. Available: http://arxiv.org/abs/2301.08281
[12] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci, ‘Online Few-Shot Gesture Learning on a Neuromorphic Processor’, *IEEE J Emerg Sel Top Circuits Syst*, vol. 10, no. 4, pp. 512–521, 2020, doi: 10.1109/JETCAS.2020.3032058.
[13] G. Tang and K. P. Michmizos, ‘Gridbot: An autonomous robot controlled by a spiking neural network mimicking the brain’s navigational system’, *ACM International Conference Proceeding Series*, 2018, doi: 10.1145/3229884.3229888.
[14] K. van Hecke, G. de Croon, L. van der Maaten, D. Hennes, and D. Izzo, ‘Persistent self-supervised learning principle: from stereo to monocular vision for obstacle avoidance’, pp. 1–17, 2016, [Online]. Available: http://arxiv.org/abs/1603.08047
[15] M. Xiao, Q. Meng, Z. Zhang, D. He, and Z. Lin, ‘Online Training Through Time for Spiking Neural Networks’, *Adv Neural Inf Process Syst*, vol. 35, no. NeurIPS, pp. 1–14, 2022.
[16] Y. Yin, X. Chen, C. Ma, J. Wu, and K. C. Tan, ‘Efficient Online Learning for Networks of Two-Compartment Spiking Neurons’, 2024, [Online]. Available: http://arxiv.org/abs/2402.15969
[17] M. Davies *et al.*, ‘Advancing Neuromorphic Computing with Loihi: A Survey of Results and Outlook’, *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021, doi: 10.1109/JPROC.2021.3067593.
[18] G. Orchard *et al.*, ‘Efficient Neuromorphic Signal Processing with Loihi 2’, *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, vol. 2021-Octob, no. 1, pp. 254–259, 2021, doi: 10.1109/SiPS52927.2021.00053.
[19] F. Paredes-Vallés, J. J. Hagenaars, J. Dupeyroux, S. Stroobants, Y. Xu, and G. C. H. E. de Croon, ‘Fully neuromorphic vision and control for autonomous drone flight’, *Sci Robot*, vol. 9, no. 90, 2024, doi: 10.1126/scirobotics.adi0591.
[20] J. Kaiser, H. Mostafa, and E. Neftci, ‘Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)’, *Front Neurosci*, vol. 14, no. May, pp. 1–11, 2020, doi: 10.3389/fnins.2020.00424.
[21] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, ‘Backpropagation and the brain’, *Nat Rev Neurosci*, vol. 21, no. 6, pp. 335–346, 2020, doi: 10.1038/s41583-020-0277-3.
[22] A. Payeur, J. Guerguiev, F. Zenke, B. A. Richards, and R. Naud, ‘Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits’, *Nat Neurosci*, vol. 24, no. 7, pp. 1010–1019, 2021, doi: 10.1038/s41593-021-00857-x.
[23] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, ‘Random synaptic feedback weights support error backpropagation for deep learning’, *Nat Commun*, vol. 7, pp. 1–10, 2016, doi: 10.1038/ncomms13276.
[24] M. Akrout, C. Wilson, P. C. Humphreys, T. Lillicrap, and D. Tweed, ‘Deep learning without weight transport’, *Adv Neural Inf Process Syst*, vol. 32, no. NeurIPS, 2019.

[25] R. Naud and H. Sprekeler, ‘Sparse bursts optimize information transmission in a multiplexed neural code’, *Proc Natl Acad Sci U S A*, vol. 115, no. 27, pp. E6329–E6338, 2018, doi: 10.1073/pnas.1720995115.
[26] F. Zeldenrust, W. J. Wadman, and B. Englitz, ‘Neural coding with bursts—Current state and future perspectives’, *Front Comput Neurosci*, vol. 12, no. July, pp. 1–14, 2018, doi: 10.3389/fncom.2018.00048.
[27] C. Koch, *Biophysics of computation: information processing in single neurons*, vol. 36, no. 10. 1999. doi: 10.5860/choice.36-5662.
[28] Y. Inglebert, J. Aljadeff, N. Brunel, and D. Debanne, ‘Synaptic plasticity rules with physiological calcium levels’, *Proc Natl Acad Sci U S A*, vol. 117, no. 52, pp. 33639–33648, 2020, doi: 10.1073/PNAS.2013663117.
[29] W. Greedy, H. W. Zhu, J. Pemberton, J. Mellor, and R. P. Costa, ‘Single-phase deep learning in cortico-cortical networks’, *Adv Neural Inf Process Syst*, vol. 35, 2022.
[30] M. Stuck and R. Naud, ‘Burstprop for Learning in Spiking Neuromorphic Hardware’, pp. 1–5, 2023, doi: 10.1145/3589737.3605968.
[31] Q. Liao, J. Z. Leibo, and T. Poggio, ‘How important is weight symmetry in backpropagation?’, *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 1837–1844, 2016, doi: 10.1609/aaai.v30i1.10279.
[32] J. F. Kolen and J. B. Pollack, ‘Back-propagation without weight transport’, *IEEE International Conference on Neural Networks - Conference Proceedings*, vol. 3, pp. 1375–1380, 1994, doi: 10.1109/icnn.1994.374486.
[33] E. Baird, N. Boeddeker, M. R. Ibbotson, and M. V. Srinivasan, ‘A universal strategy for visually guided landing’, *Proc Natl Acad Sci U S A*, vol. 110, no. 46, pp. 18686–18691, 2013, doi: 10.1073/pnas.1314311110.
[34] G. C. H. E. De Croon, ‘Monocular distance estimation with optical flow maneuvers and efference copies: A stability-based strategy’, *Bioinspir Biomim*, vol. 11, no. 1, 2016, doi: 10.1088/1748-3190/11/1/016004.
[35] G. C. H. E. de Croon, C. De Wagter, and T. Seidl, ‘Enhancing optical-flow-based control by learning visual appearance cues for flying robots’, *Nat Mach Intell*, vol. 3, no. 1, pp. 33–41, 2021, doi: 10.1038/s42256-020-00279-7.