TTK4550 – Engineering Cybernetics, Specialization Project

Biologically plausible alternatives to backpropagation for training neural networks

Mats Wiig Martinussen

Supervisor:
Bjarne André Grimstad
Trondheim, December 2025

Department of Engineering Cybernetics, NTNU 2025

Preface

This project thesis constitutes part of my master's degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The work has provided valuable insight into both the theoretical foundations and practical aspects of research within machine learning and biologically inspired learning algorithms, particularly through the formulation of research questions, development of learning methods, and critical evaluation of experimental results. I would like to express my sincere gratitude to my supervisor, Bjarne André Grimstad, for his guidance, constructive feedback, and insightful discussions throughout the project. His support and perspective have been invaluable during the development of this thesis.

Trondheim, December 17, 2025

Abstract

Backpropagation is the dominant algorithm for training artificial neural networks, but it is computationally demanding, its reliance on sequential error propagation limits parallelism, and it is widely regarded as biologically implausible. This motivates the exploration of alternative learning rules that have more effective optimization behavior by taking inspiration from biological principles. The primary objective of this project is to investigate whether biologically inspired learning algorithms can approximate or improve upon the training performance of backpropagation, with particular emphasis on convergence behavior and computational efficiency.

Weight perturbation and node perturbation methods are implemented and extended with three-factor learning variants inspired by neuromodulated plasticity. The resulting algorithms are evaluated against standard backpropagation on a synthetic sinusoid regression task and the California Housing dataset. The results demonstrate that perturbation-based methods can reach low-loss solutions on simple regression problems and achieve performance comparable to backpropagation when normalized for computational cost. At the same time, the findings show that increased biological realism alone does not guarantee improved learning, underscoring the importance of preserving a clear mathematical connection to the underlying optimization problem.

Table of Contents

Preface i

Abstract ii

List of Tables v

List of Figures vi

1 Introduction 1
- 1.1 Background and motivation . . . 1
- 1.2 Research objectives and questions . . . 2

2 Theory 3
- 2.1 How the brain learns . . . 3
  - 2.1.1 Hebbian learning . . . 4
  - 2.1.2 Neuromodulators and three factor learning . . . 5
  - 2.1.3 Stochasticity in biological learning . . . 6
- 2.2 How artificial neural networks learn . . . 6
  - 2.2.1 Backpropagation . . . 7
  - 2.2.2 Perturbation methods . . . 10

3 Method 15
- 3.1 Three-factor learning using weight perturbation . . . 15
- 3.2 Three-factor learning using node perturbation . . . 17
- 3.3 Summary of methods . . . 18

4 Case study 19
- 4.1 Experimental set-up and implementation . . . 19
  - 4.1.1 Learning problems and datasets . . . 19
  - 4.1.2 Model architectures . . . 19
  - 4.1.3 Algorithms . . . 19
  - 4.1.4 Hyperparameters . . . 21
  - 4.1.5 Evaluation metrics . . . 23
- 4.2 Results . . . 23
  - 4.2.1 Loss as a function of training epochs . . . 23
  - 4.2.2 Loss as a function of computational cost . . . 24
  - 4.2.3 Variance across runs . . . 25

5 Concluding remarks 32
- 5.1 Discussion . . . 32
  - 5.1.1 Ability to reach low-loss solutions . . . 32

5.1.2 Convergence efficiency 32
5.1.3 Stability and variance 33
5.1.4 Mathematical grounding and biological inspiration 33
5.1.5 Limitations of the experimental setup 34
5.2 Future work 34
5.3 Conclusion 34

6 Sustainability **36**

7 Use of Artificial Intelligence **37**

List of Tables

1 Average training loss variance across 10 runs for the sinusoid regression task. 25
2 Average training loss variance across 5 runs for the California Housing regression task. 25

List of Figures

1 Hebbian learning illustrated in a feedforward neural network. A single synapse w_{ij} between a presynaptic neuron with activity x_i and a postsynaptic neuron with activity x_j is highlighted. The synaptic weight is updated according to the Hebbian rule $\Delta w_{ij} = \eta\, x_i x_j$, such that correlated pre- and postsynaptic activity leads to synaptic strengthening. 4

2 Illustration of three-factor synaptic plasticity in a feedforward neural network. A single synapse w_{ij} connecting a presynaptic neuron with activity x_i to a postsynaptic neuron with state x_j is highlighted. Local presynaptic and postsynaptic activity generates an eligibility trace e_{ij} at the synapse, while a global modulatory signal M is broadcast across the network. Synaptic plasticity arises from the interaction of these three factors, yielding the update rule $\Delta w_{ij} = \eta\, e_{ij}\, M = \eta\, x_i\, f(x_j)\, M$, where the global signal determines the sign and magnitude of the weight change. 6

3 Training loss as a function of epochs for the sinusoid regression task. BP = Backpropagation, WP = Weight Perturbation, 3-F WP = Three Factor Weight Perturbation, 3-F NP = Three Factor Node Perturbation 26

4 Training loss as a function of epochs for the California Housing regression task. BP = Backpropagation, WP = Weight Perturbation, 3-WP = Three Factor Weight Perturbation, 3-NP = Three Factor Node Perturbation 27

5 Training loss as a function of computational cost for the sinusoid regression task. BP = Backpropagation, WP = Weight Perturbation, 3-WP = Three Factor Weight Perturbation, 3-NP = Three Factor Node Perturbation 28

6 Training loss as a function of computational cost for the California Housing regression task. BP = Backpropagation, WP = Weight Perturbation, 3-WP = Three Factor Weight Perturbation, 3-NP = Three Factor Node Perturbation 29

7 Training loss variance across runs for the sinusoid regression task. Variance is computed per minibatch over 10 random initializations.BP = Backpropagation, WP = Weight Perturbation, 3-F WP = Three Factor Weight Perturbation, 3-F NP = Three Factor Node Perturbation 30

8 Training loss variance across 5 runs for the California Housing regression task. Variance is computed per minibatch over independent random initializations.BP = Backpropagation, WP = Weight Perturbation, 3F-WP = Three Factor Weight Perturbation, 3F-NP = Three Factor Node Perturbation 31

1 Introduction

1.1 Background and motivation

Backpropagation is the dominant learning algorithm underlying modern machine learning and deep learning. Since its introduction by Rumelhart et al. [1986], it has formed the basis for training neural networks across a wide range of domains, including computer vision, natural language processing, and reinforcement learning. Its ability to efficiently compute exact gradients in high-dimensional parameter spaces has enabled the successful optimization of large and expressive models, and it remains the core mechanism behind essentially all state-of-the-art neural network systems in use today.

Despite its empirical success, backpropagation exhibits several limitations that motivate the search for alternative learning rules. From an optimization perspective, gradient descent follows a local direction of steepest descent in a highly non-convex loss landscape, providing no guarantees of reaching globally optimal solutions. Algorithmically, the backward pass introduces a strict layer-wise dependency, making the computation of gradients partially sequential and limiting depth-wise parallelism. In addition, computing and storing gradients incurs significant computational and memory overhead. Most fundamentally, backpropagation is biologically implausible: it requires precise, neuron-specific error signals and symmetric forward and backward weights, assumptions for which no clear biological mechanisms are known [Lillicrap et al., 2020].

Many of the most influential developments in modern machine learning have been inspired, at least at an abstract level, by biological systems and the brain. Artificial neural networks were originally motivated by simplified models of neuronal computation, and their modern formulation can still be viewed as a highly idealized abstraction of biological neural circuits [Goodfellow et al., 2017]. Convolutional neural networks, in particular, are inspired by the hierarchical organization and localized receptive fields observed in biological visual systems, an idea formalized in early computational models of vision and introduced to modern machine learning by Lecun et al. [1998]. More recently, attention mechanisms have emerged as a central architectural component in deep learning, motivated by the idea of selectively focusing computational resources on task-relevant information, as introduced in the Transformer architecture by Vaswani et al. [2017]. Although these inspirations do not aim to replicate biological detail, they illustrate that biologically motivated principles can lead to powerful and practically effective machine learning models. This success motivates the question of whether biological inspiration at the level of learning rules, rather than architecture alone, could similarly yield useful alternatives to backpropagation.

This line of work is motivated by both practical and conceptual considerations. From an engineering perspective, identifying alternatives to backpropagation could enable cheaper or faster training by reducing computational overhead, improving parallelism, or alleviating memory constraints associated with the backward pass. From a scientific perspective, biologically grounded learning rules offer a pathway toward a better understanding of how learning in the brain relates to optimization in artificial systems, potentially narrowing the gap between biological and artificial learning mechanisms. Given that backpropagation underlies virtually all modern machine learning systems, even modest improvements, task-specific advantages, or complementary training regimes would have far-reaching implications. This motivates a careful exploration of biologically inspired learning rules, not with the expectation of replacing backpropagation outright, but to assess whether biology can point toward promising directions for improving or extending the current learning paradigm.

1.2 Research objectives and questions

The primary objective of this project is to investigate whether biologically inspired learning rules can approximate or improve upon backpropagation in terms of convergence speed, computational efficiency, or overall training performance. The objective is exploratory, aiming to identify learning mechanisms that appear promising within a controlled experimental setting on two supervised regression tasks.

A secondary objective is to compare backpropagation with biologically inspired learning rules in order to better understand the relationship between biological learning mechanisms and learning in artificial neural networks. This comparison is intended to provide intuition about whether and how biologically plausible learning rules can help narrow the gap between learning in the brain and gradient-based training in artificial systems.

This project approaches these objectives from a practical and exploratory perspective. Biology is used as a source of inspiration rather than a strict constraint, and several simplifying assumptions are made to ensure compatibility with standard supervised learning setups and commonly used neural network architectures. While outperforming backpropagation would be a notable outcome, the primary aim is to assess whether biologically inspired learning rules can achieve performance that is comparable to backpropagation and therefore represent a promising step in the direction of viable alternatives.

2 Theory

2.1 How the brain learns

Learning, in its most general sense, is described by Stanislas Dehaene in his book How We Learn [Dehaene, 2021] as building an internal model of the external world. Our brains constantly construct internal models such as a mental map of our neighborhood, a sense of how our limbs move, or the rules of a language we use every day. None of us are born with these representations; we acquire them through experience. In this view, learning is simply the process of updating these internal models.

In the brain, internal models are implemented by large networks of neurons connected through synapses. A neuron communicates by producing brief electrical spikes, and whether it spikes depends on the signals it receives from other neurons [Kandel et al., 2021]. Each synapse acts as a weighted connection: strong synapses make it more likely that activity in one neuron triggers activity in another, while weak synapses have little effect. When many neurons interact, their collective activity patterns represent structured information, such as the layout of a familiar street or the sound of a word. These distributed patterns form the internal models that allow the brain to interpret and predict the world.

So how do we update the models in our brain, so that we learn? In Dehaene's second definition, learning is "adjusting the parameters of a mental model" [Dehaene, 2021]. In biological terms, these parameters are the strengths of synapses. Synapses are plastic: they can strengthen or weaken depending on experience [Kandel et al., 2021]. Changing synaptic strengths alters how neurons influence each other, and therefore modifies the internal model encoded by their collective activity. In this way, learning in the brain amounts to tuning the synaptic connections in a vast neural network so that its internal representations better capture the structure of the reality they are meant to model.

The brain contains many mechanisms for adjusting synaptic strengths, and neuroscience offers a wide range of theories that attempt to explain how learning emerges from these processes, some of which is presented in the following. The goal here is not to provide a complete account of all known forms of plasticity, but to introduce the specific ideas that are relevant for the learning algorithms developed later in this work. While no single theory fully captures how the brain learns, the concepts presented in the following sections represent influential principles that have inspired biologically grounded learning algorithms, including this work.

To make these biological learning principles amenable to mathematical analysis and algorithmic implementation, we adopt a set of simplifying modeling assumptions. In the brain, neurons are connected recurrently, meaning that signals propagate back and forth over time rather than flowing once in a single direction. Explicitly modeling such time-dependent dynamics is not compatible with standard deep learning models, which are trained as feedforward networks where information flows only in a single direction. Since the goal of this work is to study biologically inspired alternatives to backpropagation within existing neural network architectures, we therefore adopt a rate-based approximation in which pre- and postsynaptic firing rates are assumed to remain approximately constant over a learning trial. Under this assumption, recurrent biological circuits can be approximated by feedforward networks, and synaptic updates can be expressed without explicit temporal dynamics. As argued by Gerstner and Kistler [2002], this approximation captures the essential dependencies relevant for synaptic plasticity and constitutes a principled abstraction rather than a break with biological realism.

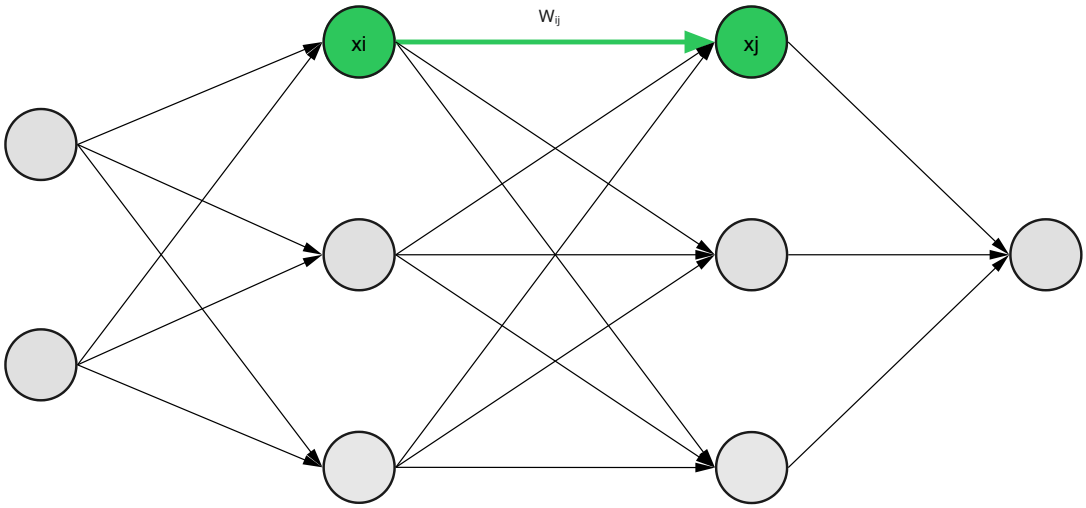


Figure 1: Hebbian learning illustrated in a feedforward neural network. A single synapse w_{ij} between a presynaptic neuron with activity x_i and a postsynaptic neuron with activity x_j is highlighted. The synaptic weight is updated according to the Hebbian rule $\Delta w_{ij} = \eta\, x_i x_j$, such that correlated pre- and postsynaptic activity leads to synaptic strengthening.

2.1.1 Hebbian learning

Hebbian learning is one of the most fundamental and well-established principles of synaptic plasticity. In its original formulation, Hebb proposed that when a presynaptic neuron repeatedly contributes to the firing of a postsynaptic neuron, the synaptic connection between them should strengthen [Hebb, 1949]. This idea is commonly formalized by a local learning rule of the form

$$\Delta w_{ij} = \eta\, x_i\, f(x_j),$$

where w_{ij} denotes the synaptic weight from presynaptic neuron i to postsynaptic neuron j, x_i denotes the activity of the presynaptic neuron, x_j denotes the activity of the postsynaptic neuron, and $f(\cdot)$ is a function of the postsynaptic state. The parameter $\eta > 0$ is a learning-rate parameter controlling the magnitude of synaptic updates. Figure 1 illustrates this principle in a simple feedforward neural network, highlighting a single synapse whose strength is modified according to local pre- and postsynaptic activity. Such a rule is strictly local: the change in synaptic strength depends only on the activity of the two neurons connected by that synapse.

Despite its simplicity, Hebbian learning can extract meaningful statistical structure from the input. Because synaptic changes are driven by correlations between presynaptic and postsynaptic activity, Hebbian plasticity naturally reinforces correlated patterns in the input. A classic example is Oja's rule [Oja, 1982], which can be viewed as a stabilized extension of a Hebbian learning rule, in which the synaptic weights are updated according to

$$\Delta w_{ij} = \eta\, x_i\, x_j \;-\; \eta\, x_j^2\, w_{ij}.$$

The first term $\eta\, x_i\, x_j$ corresponds to the classical Hebbian update and constitutes the driving force of learning. The second term $-\eta\, x_j^2 w_{ij}$ serves to constrain the growth of the synaptic weights by counteracting unbounded amplification. Oja's rule therefore preserves the core Hebbian learning mechanism while ensuring stability, and allows a single neuron to learn the first principal component from PCA (Principal Component Analysis) of its input [Oja, 1982]. This demonstrates that basic Hebbian mechanisms are capable of extracting meaningful structure from data through purely local, unsupervised learning.

However, pure Hebbian learning is incomplete as a general learning mechanism, because it does not incorporate feedback about whether the resulting behavior or prediction was useful. Refining a motor skill, understanding whether an action led to reward, or correcting a mistake in a language task all require outcome-dependent learning. This is information that local co-activity alone cannot provide. Thus, while Hebbian plasticity forms a foundation of synaptic learning, additional mechanisms are needed for the brain to learn from success, failure, and delayed consequences.

2.1.2 Neuromodulators and three factor learning

To learn from success and failure, neuromodulators such as dopamine provide the brain with a global signal about reward, surprise, or outcome quality. Midbrain dopamine neurons increase their firing when an outcome is better than expected, and decrease their firing when it is worse, effectively encoding a reward-prediction error [Schultz et al., 1997]. This signal is broadcast broadly across the brain and acts as a scalar feedback about whether recent behavior or neural activity was beneficial.

By combining this global reward-prediction signal with a local Hebbian-like factor, the brain can identify which synapses contributed to a successful or unsuccessful outcome. Figure 2 schematically illustrates how a global modulatory signal can interact with local synaptic activity to drive plasticity at individual synapses. In biological models, this local term is captured by an eligibility trace, which is a synaptic tag generated whenever presynaptic activity interacts with a postsynaptic response. As highlighted by Gerstner et al. [2018], this tag does not need to take the form of a simple product of pre- and postsynaptic activity; it can be any nonlinear function that reflects the recent state of the two neurons. When dopamine arrives while this eligibility trace is present, the synapse is strengthened or weakened accordingly. This interaction between (i) presynaptic activity, (ii) postsynaptic state, and (iii) a global modulatory signal constitutes a three-factor learning rule [Frémaux and Gerstner, 2016], shown by the equation:

$$\Delta w_{ij} \;=\; \eta\, x_i\, f(x_j)\, M \tag{1}$$

Here, x_i denotes the presynaptic activity, $f(x_j)$ is a (potentially nonlinear) function of the postsynaptic state, M is a global modulatory factor such as a reward-prediction error, and η is a learning-rate parameter.

Three-factor learning rules overcome the core limitation of pure Hebbian plasticity: they allow learning to depend not only on correlations in the input, but also on whether the resulting behavior or prediction was successful. This makes three-factor rules a biologically grounded form of supervised or reinforcement learning, and they are widely considered a plausible mechanism for outcome-driven synaptic change in the brain [Gerstner et al., 2018].

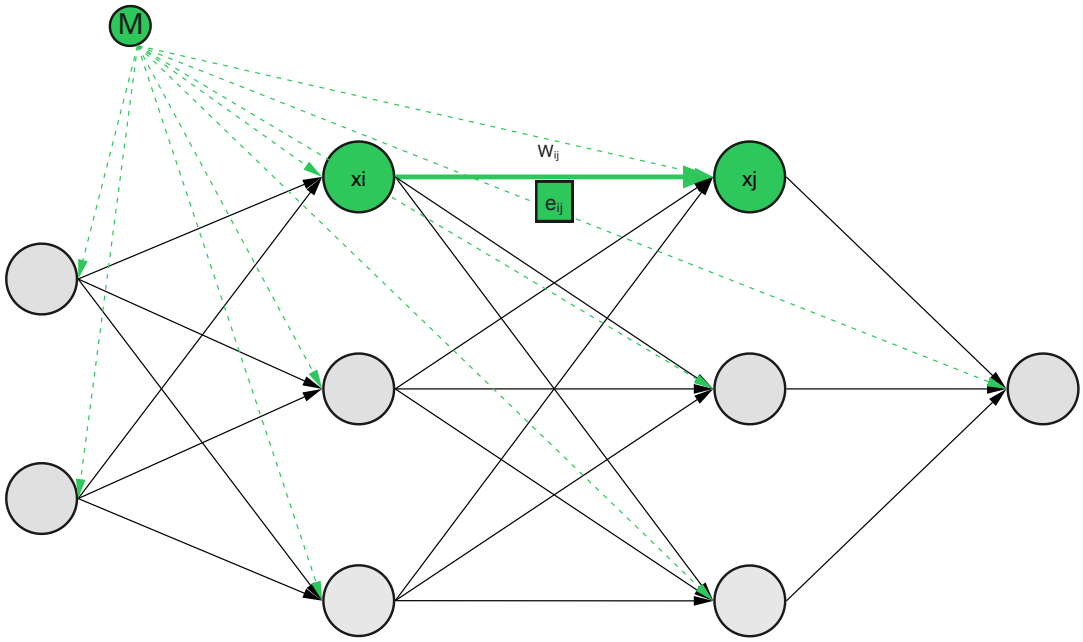


Figure 2: Illustration of three-factor synaptic plasticity in a feedforward neural network. A single synapse w_{ij} connecting a presynaptic neuron with activity x_i to a postsynaptic neuron with state x_j is highlighted. Local presynaptic and postsynaptic activity generates an eligibility trace e_{ij} at the synapse, while a global modulatory signal M is broadcast across the network. Synaptic plasticity arises from the interaction of these three factors, yielding the update rule $\Delta w_{ij} = \eta\, e_{ij}\, M = \eta\, x_i\, f(x_j)\, M$, where the global signal determines the sign and magnitude of the weight change.

2.1.3 Stochasticity in biological learning

Biological learning operates in the presence of intrinsic stochasticity, which plays an active role in how synaptic changes are formed. Neural systems exhibit variability at multiple levels, including stochastic fluctuations in synaptic efficacy (effectively stochastic fluctuations in synaptic strengths/weights) and irregular neuronal firing. As discussed by Seung [2003], correlations between reward signals and these stochastic fluctuations can drive systematic synaptic change. This observation supports the biological plausibility of learning mechanisms that rely on random perturbations, whether arising at the level of synapses or neuronal activity, which will be explored later in this work.

2.2 How artificial neural networks learn

Much like biological neural circuits, artificial neural networks consist of interconnected units that transmit signals through weighted connections, the artificial analogue of synapses. In this thesis we focus on feedforward neural networks, where information flows in one direction layer by layer from

input to output. Each neuron in such a network computes a weighted sum of its incoming signals, adds a bias term, and then applies a nonlinear activation function. Formally, the pre-activation and activation of neuron j in layer ℓ are given by

$$z_j^{(\ell)} = \sum_i w_{ij}^{(\ell)} x_i^{(\ell-1)} + b_j^{(\ell)}, \qquad x_j^{(\ell)} = \phi\Big(z_j^{(\ell)}\Big), \tag{2}$$

Here, $z_j^{(\ell)}$ denotes the pre-activation of neuron j in layer ℓ, $x_i^{(\ell-1)}$ is the activation of neuron i in the previous layer, $w_{ij}^{(\ell)}$ is the corresponding synaptic weight, and $b_j^{(\ell)}$ is a bias term. The function $\phi(\cdot)$ denotes a nonlinear activation function, and $x_j^{(\ell)}$ is the resulting activation of neuron j in layer ℓ. These expressions follow the standard formulation of feedforward networks as presented in Goodfellow et al. [2017]. Stacking many such layers in this way yields a flexible parametric mapping from input to output that serves as the network's internal model of the task at hand. By the universal approximation theorem, such networks can in principle represent any continuous mapping from inputs to outputs on a compact domain Hornik [1991].

In this work, we focus on supervised learning, where the external model we aim to represent is given by a dataset of input–output pairs (x, y). The goal is to learn a function that maps each input x to its corresponding target y. To quantify prediction accuracy, we introduce a loss function $L(y, y_\theta(x))$ measuring the discrepancy between the true output, y, and the network's prediction, $y_\theta(x)$. Learning then consists of adjusting the parameters θ, which here denote all weights and biases appearing in the feedforward equation (2), so as to minimize this loss. Formally, we seek

$$\theta^* = \operatorname*{argmin}_\theta \mathbb{E}_{(x,y)}[L(y, y_\theta(x))], \tag{3}$$

which specifies the target parameters that best fit the dataset.

This supervised learning framework connects directly to Dehaene's description of learning as "adjusting the parameters of an internal model" [Dehaene, 2021], discussed in the previous section. In the biological setting, these parameters correspond to synaptic strengths; in artificial neural networks, they are the weights and biases collected in θ and defined by the feedforward equations (2). The optimization problem in (3) therefore provides a precise mathematical instance of the same principle: by modifying its parameters, the network updates its internal model so that it more accurately reflects the external relationship encoded in the dataset. This correspondence highlights the conceptual alignment between biological and artificial learning.

So far, the brain and artificial neural networks appear conceptually similar: both learn by adjusting the parameters that define an internal model. However, when we now turn to the actual learning algorithm most commonly used in artificial networks, this similarity breaks. We begin by introducing backpropagation [Rumelhart et al., 1986], the golden standard for parameter updates in modern deep learning, and then consider a more biologically plausible alternative based on perturbation-driven updates. Both methods are briefly presented and mathematically derived, before we comment on their effectiveness, computational properties, and biological plausibility.

2.2.1 Backpropagation

Backpropagation [Rumelhart et al., 1986] is the de facto algorithm for training neural networks. The goal is to minimize the objective in (3) by following the gradient of the loss with respect to the parameters. Formally, the update direction is given by

$$\nabla_\theta L(y, y_\theta(x)), \tag{4}$$

which specifies how each component of θ should change to reduce the loss. Backpropagation computes the exact value of (4) by propagating error information backward through the network, layer by layer, using the chain rule. In this way, each parameter receives a precise estimate of how it contributed to the final loss.

To derive backpropagation, we begin at the final layer of the network. This is because the loss function $L(y, y_\theta(x))$ depends directly on the output activations $x_j^{(\ell)}$. Any influence that earlier layers have on the loss must therefore flow through these output units, and the gradients with respect to parameters in earlier layers can consequently be obtained by repeatedly applying the chain rule backward through the network, as we shall see.

The first step is thus to compute the derivative of the loss with respect to each output activation:

$$\frac{\partial L}{\partial x_j^{(\ell)}}.$$

Using the activation relation in (2), this is converted into the corresponding gradient with respect to the pre-activation, which we denote $\delta_j^{(\ell)}$,

$$\delta_j^{(\ell)} := \frac{\partial L}{\partial z_j^{(\ell)}} = \frac{\partial L}{\partial x_j^{(\ell)}}\, \phi'(z_j^{(\ell)}), \tag{5}$$

which represents how a small change in $z_j^{(\ell)}$ affects the final loss. This quantity forms the starting point for propagating gradients backward through the network.

From the pre-activation definition in (2), we get the gradient of the loss with respect to each weight in the final layer by another application of the chain rule:

$$\frac{\partial z_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = x_i^{(\ell-1)}, \frac{\partial L}{\partial w_{ij}^{(\ell)}} = \frac{\partial L}{\partial z_j^{(\ell)}}\, \frac{\partial z_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = \delta_j^{(\ell)}\, x_i^{(\ell-1)}. \tag{6}$$

Similarly, since $z_j^{(\ell)}$ depends linearly on $b_j^{(\ell)}$,

$$\frac{\partial z_j^{(\ell)}}{\partial b_j^{(\ell)}} = 1, \qquad \frac{\partial L}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)}.$$

The expressions in (5) and (6) are not specific to the output layer. They describe how the gradient behaves in *any* layer of the network. In every layer l, the gradient with respect to a weight has the same simple form,

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = x_i^{(l-1)}\, \delta_j^{(l)},$$

and the gradient with respect to a bias is always

$$\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}.$$

This means that once we know the activations $x_i^{(l-1)}$ from the forward pass and the error term $\delta_j^{(l)}$ for each neuron, the gradients in that layer can be computed immediately and in parallel.

Thus, the main remaining task is to compute $\delta_j^{(l)}$ for all hidden layers. For the output layer this follows directly from the loss, as shown above. For a hidden layer, however, the loss does not depend on its activations directly. Instead, any change in $x_j^{(l)}$ influences the loss only through the next layer $(l+1)$. Applying the chain rule yields

$$\frac{\partial L}{\partial x_j^{(l)}} = \sum_k \frac{\partial L}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial x_j^{(l)}} = \sum_k w_{jk}^{(l+1)} \, \delta_k^{(l+1)},$$

where the sum runs over all neurons k in the next layer. Converting this into the pre-activation derivative gives the recursive formula

$$\delta_j^{(l)} = \frac{\partial L}{\partial z_j^{(l)}} = \left(\sum_k w_{jk}^{(l+1)} \, \delta_k^{(l+1)} \right) \phi'(z_j^{(l)}).$$

This relation is what makes the algorithm run "backwards": to compute $\delta_j^{(l)}$ for layer l, we must already know all $\delta_k^{(l+1)}$ in the next layer. In other words, a layer can only compute its error terms once the layer after it has computed theirs. This is why backpropagation proceeds layer by layer, from the output back to the input. After this backward pass, the exact gradient $\partial L/\partial \theta$ is available for all parameters.

Substituting the recursive expression for $\delta_j^{(l)}$ into the weight gradient reveals the full structure of a backpropagation update for a hidden layer weight:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = x_i^{(l-1)} \, \phi'\Big(z_j^{(l)}\Big) \sum_k w_{jk}^{(l+1)} \, \delta_k^{(l+1)}. \tag{7}$$

Backpropagation has been remarkably successful in training artificial neural networks and underlies virtually all modern deep learning systems. Its efficiency, scalability, and ability to compute exact gradients in large architectures have enabled neural networks to solve tasks ranging from image and speech recognition to natural language processing and strategic game play. Despite this success, the algorithm also comes with several caveats that motivate the search for alternative learning rules:

Although backpropagation computes the exact gradient in (4), it does not guarantee that the optimization problem in (3) is solved globally. The loss surfaces of deep neural networks are highly non-convex [Goodfellow et al., 2017], meaning that no analytical solution exists and gradient-based methods can only follow a locally descending direction. The gradient identifies a direction of immediate decrease rather than the direction towards a global minimizer. As a consequence, convergence depends on factors such as initialization, learning rate, and the overall geometry of the loss landscape.

A second limitation arises from the sequential structure of the backpropagation algorithm. While backpropagation exploits parallelism efficiently within layers and across the batch dimension, the gradient for layer ℓ cannot be computed until the error signal from layer $\ell+1$ is available. This strict inter-layer dependency prevents depth-wise parallelism within a single sample and introduces an unavoidable sequential component in the backward pass. In practice, this effect can be mitigated by increasing the batch size, allowing multiple samples to be processed in parallel and thereby improving overall hardware utilization. However, this strategy does not eliminate the fundamental layer-wise dependency, and batch sizes are often in practice limited by memory constraints and

optimization considerations. Consequently, gradient information must still be propagated backward one layer at a time, which prevents full exploitation of the massive parallelism offered by modern GPU architectures.

A third limitation concerns computational cost. Each training step requires a forward evaluation of the network followed by a backward pass to compute gradients. Empirical benchmarks reported by Baydin et al. [2018] show that evaluating gradients using reverse-mode automatic differentiation typically incurs a total runtime of approximately twice that of a single forward evaluation for high-dimensional functions. The exact ratio depends on factors such as network architecture, activation functions, and hardware implementation. Since the present work is exploratory, we adopt an approximation in which one backpropagation update is assumed to cost two forward passes. This estimate is used as a reference when comparing computational efficiency across learning methods in the results section.

When it comes to biological plausibility, there is no direct evidence that the brain implements anything resembling backpropagation's mechanism for computing synapse-specific error derivatives. While biological learning relies primarily on local activity and global neuromodulatory signals, backpropagation requires precise, neuron-specific error terms that must be delivered to each synapse. As highlighted by Lillicrap et al. [2020], implementing standard backpropagation in the brain would demand that each synapse receive a distinct signed error value indicating how a small change in that weight would affect overall performance, an operation for which no known biological substrate exists. This difficulty is compounded by structural constraints such as the weight transport problem, which requires exact symmetry between feedforward and feedback weights, and by the need for specialized pathways capable of transmitting high-precision error signals [Lillicrap et al., 2020]. Together, these issues make a literal implementation of textbook backpropagation biologically implausible and motivate the exploration of learning rules that take inspiration from biology, such as the one presented in the next paragraph.

2.2.2 Perturbation methods

Perturbation-based learning offers a conceptually simple alternative to backpropagation. Instead of computing exact gradients by propagating error signals backward through the network, the idea is to add small random perturbations to the parameters and observe how these perturbations affect the loss. If a particular noise direction decreases the loss, then moving the parameters slightly in that direction is beneficial; if it increases the loss, the direction is harmful. By correlating random perturbations with the resulting change in loss, one can form an unbiased estimate of the gradient using only forward evaluations of the network. To understand why that is, we first need to look at the score function estimator [Williams, 1992]:

Score function estimator. Consider the stochastic objective

$$F(\theta) := \mathbb{E}_{x\sim p_\theta(x)}[f(x)] = \int f(x)\, p_\theta(x)\, dx,$$

Here, $F(\theta)$ denotes the expected objective value, $p_\theta(x)$ is a probability distribution parameterized by θ, x is a stochastic variable drawn from this distribution, and $f(x)$ is an arbitrary scalar function of x, typically corresponding to the loss or objective evaluated at a sampled perturbation. Our goal is to compute $\nabla_\theta F(\theta)$. To obtain an expression in which the θ–dependence is explicit, we differentiate the integrand rather than the already–integrated expression: after integration, the

dependence on θ is hidden inside $F(\theta)$ and cannot be accessed directly. We therefore move the gradient operator inside the integral. This step is valid whenever $p_\theta(x)$ is differentiable in θ and the integral is dominated by an integrable, θ–independent function, ensuring that differentiation under the integral sign is permitted. Under these standard regularity conditions, we obtain

$$\nabla_\theta F(\theta) = \int f(x)\, \nabla_\theta p_\theta(x)\, dx. \tag{8}$$

A useful identity for probability densities is the log-derivative (or likelihood-ratio) identity

$$\nabla_\theta p_\theta(x) = p_\theta(x)\, \nabla_\theta \log p_\theta(x),$$

which follows from applying the chain rule to $p_\theta(x) = \exp(\log p_\theta(x))$. Substituting this into the integral yields

$$\nabla_\theta F(\theta) = \int f(x)\, p_\theta(x)\, \nabla_\theta \log p_\theta(x)\, dx \tag{9}$$

$$= \mathbb{E}_{x\sim p_\theta(x)}[f(x)\, \nabla_\theta \log p_\theta(x)]\,. \tag{10}$$

Equation (9) shows that the gradient of an expectation can be expressed as an expectation involving the score function $\nabla_\theta \log p_\theta(x)$. Approximating this expectation with N i.i.d. samples $x^{(k)} \sim p_\theta(x)$ gives the Monte Carlo *score function estimator*

$$\widehat{g} = \widehat{\nabla_\theta F(\theta)} = \frac{1}{N} \sum_{k=1}^{N} f\big(x^{(k)}\big)\, \nabla_\theta \log p_\theta\big(x^{(k)}\big), \tag{11}$$

which is an unbiased estimator of $\nabla_\theta F(\theta)$.

Baselines. Although unbiased, the estimator will have high variance when the number of parameters is high. To reduce the variance, we can subtract a baseline. The estimator remains unbiased if we subtract any baseline b that does not depend on the sampled value x:

$$\nabla_\theta F(\theta) = \mathbb{E}[(f(x) - b)\, \nabla_\theta \log p_\theta(x)]\,,$$

since $\mathbb{E}[\nabla_\theta \log p_\theta(x)] = 0$. A common variance-reducing choice is the *leave-one-out baseline*: for the k-th sample we set

$$b^{(k)} = \frac{1}{N-1} \sum_{k' \neq k} f\Big(x^{(k')}\Big)\,, \tag{12}$$

The intuition behind subtracting a baseline is the following. Without a baseline, every sampled value $x^{(k)}$ contributes a step in the direction of $\nabla_\theta \log p_\theta(x^{(k)})$, since all values of $f(x^{(k)})$ are treated as positive evidence. Samples with larger values of $f(x^{(k)})$ produce larger updates, but samples with small or unfavorable values still contribute updates in their own direction, adding unnecessary variance. By subtracting a baseline, each sample is instead evaluated relative to a typical outcome under the distribution. Samples with $f(x^{(k)})$ above the baseline reinforce their contribution, while those below the baseline flip sign and push in the opposite direction. This preserves unbiasedness while substantially reducing variance in the gradient estimate.

Gaussian case. A frequently used case is when x is drawn from a normal distribution whose mean is the parameter θ:

$$x \sim \mathcal{N}(\theta, \sigma^2).$$

Then

$$\log p_\theta(x) = -\frac{(x-\theta)^2}{2\sigma^2} + C, \qquad \nabla_\theta \log p_\theta(x) = \frac{x-\theta}{\sigma^2}.$$

Thus the score function estimator becomes

$$\nabla_\theta F(\theta) = \mathbb{E}\left[f(x)\,\frac{x-\theta}{\sigma^2}\right] \approx \frac{1}{N}\sum_{k=1}^{N}\big(f(x^{(k)}) - b^{(k)}\big)\,\frac{x^{(k)}-\theta}{\sigma^2}.$$

If we reparameterize $x = \theta + \sigma\xi$ with $\xi \sim \mathcal{N}(0,1)$, the same expression becomes

$$\nabla_\theta F(\theta) \approx \frac{1}{N}\sum_{k=1}^{N}\big(f(\theta + \sigma\xi^{(k)}) - b^{(k)}\big)\,\frac{\xi^{(k)}}{\sigma}, \tag{13}$$

which is the form used in perturbation-based learning rules.

To apply the score–function estimator to neural network training, we treat the network parameters θ (the weights and biases) as stochastic rather than fixed. Concretely, each parameter is modeled as a Gaussian random variable

$$\theta_i = \mu_i + \sigma\,\xi_i, \qquad \xi_i \sim \mathcal{N}(0,1),$$

where μ_i denotes the deterministic parameter value (weight or bias) that we aim to learn and σ is a small fixed perturbation scale. For a given perturbation ξ_i, we evaluate the loss $F(\theta)$ by running a single forward pass of the network with the perturbed parameters. The score–function estimator then tells us that

$$\nabla_{\mu_i} F(\theta) \approx \big(F(\theta) - b\big)\,\frac{\xi_i}{\sigma}, \tag{14}$$

which follows directly from the Gaussian expression in (13). For neural network training, a natural choice for the baseline is the batch average of the loss values across all perturbations, which serves as a simple and effective approximation of the leave-one-out baseline. Replacing b in (14) with the batch mean ensures that perturbations producing higher-than-average loss push the parameters in their direction, while lower-than-average perturbations flip sign and push in the opposite direction, yielding a lower-variance gradient estimate.

The quantity in (14) provides an estimate of $\nabla_{\mu_i} F(\theta)$, the direction in which the loss increases most rapidly with respect to parameter μ_i. To minimize the loss, we therefore update the parameters in the opposite direction, giving the perturbation-based learning rule

$$\Delta\mu_i \;=\; -\eta\,\big(F(\theta) - b\big)\,\frac{\xi_i}{\sigma}, \tag{15}$$

where η is a learning-rate parameter. This method requires only forward passes and allows all parameters to be updated in parallel, and yields a simple perturbation–based learning rule with unbiased gradient estimates and minimal computational structure.

A key practical advantage of perturbation-based learning is its parallel structure. Unlike backpropagation, which requires gradients to be computed layer by layer due to the backward dependency chain, the perturbation updates for all parameters are independent once the forward pass has produced the loss. Each parameter update depends only on its own perturbation and the global loss signal, allowing the updates to be computed simultaneously as a single elementwise operation. This observation is consistent with the analysis of Salimans et al. [2017], who note that perturbation-based optimizers are naturally suited to modern parallel hardware. Although the forward pass remains sequential across layers, the update step can be distributed across many cores, enabling efficient parallel execution on contemporary GPU and distributed compute systems.

A second practical advantage of perturbation-based learning lies in its computational cost. To obtain a simple and conservative estimate of the computational cost, we compare the operations required for a perturbation update with those of a forward pass. From (2), we have that one forward pass involves one multiplication and one accumulation per weight, amounting to approximately $2N$ primitive operations, where N denotes the number of parameters. Nonlinear activation functions are evaluated once per neuron rather than per weight, and in sufficiently wide networks the number of weights dominates the number of neurons. Activation costs are therefore a lower-order contribution and are ignored, yielding a best-case estimate for the forward pass and a worst-case estimate for the relative cost of the perturbation update. Constructing the perturbation update requires only a single multiplication per parameter to form $(F(\theta) - b)\epsilon$, i.e. N operations, half that of a forward pass. Under this crude accounting, one perturbation sample (forward pass plus multiplication) therefore costs on the order of 1.5 forward-pass equivalents. In contrast, backpropagation requires the weight update shown in (7), which involves additional multiplications with presynaptic activations, evaluation of activation derivatives, and accumulation of error signals from downstream neurons. Relative to this structure, the perturbation update is substantially simpler, making the assumption of roughly 1.5 forward-pass equivalents conservative and reasonable when backpropagation is treated as approximately 2 forward passes.

In comparison to backpropagation, perturbation-based learning is more consistent with the established biological principles above. It requires only a global scalar feedback signal, analogous to the neuromodulatory reward signals, rather than synapse-specific error derivatives. This avoids the core biological implausibilities associated with backpropagation. Moreover, as discussed above, the use of random perturbations of weights is biologically plausible, as stochastic fluctuations in synaptic efficacy in the brain can be correlated with global reward signals to drive learning [Seung, 2003]. At the same time, perturbation methods do not incorporate any local synaptic factor reflecting presynaptic and postsynaptic activity, such as the eligibility traces that characterize experimentally supported three-factor learning rules. In this sense, perturbation-based learning is biologically permissible but incomplete.

In terms of effectiveness, perturbation-based learning has been shown to be capable of driving parameter updates and improving performance in a range of settings, including early work on stochastic optimization [Williams, 1992] and large-scale evolution strategies [Salimans et al., 2017]. However, the method is limited by the high variance of the score–function estimator, a difficulty that grows with the dimensionality of the parameter space. This variance reduces sample efficiency and makes learning unstable on more complex tasks. As noted by Lillicrap et al. [2020], perturbation-based methods have not yet been successfully applied to training large, deep networks on challenging problems, largely due to these variance-related issues. Their practical effectiveness therefore remains strongly problem dependent.

Taken together, perturbation-based learning combines several appealing properties: it avoids the

biologically implausible requirements of backpropagation, is highly parallelizable, computationally efficient, and has demonstrated the ability to learn in a variety of settings. At the same time, its high-variance updates limit its performance on more complex tasks. This raises a natural question for the remainder of this work: can additional biological principles be incorporated to develop learning rules that retain these advantages while achieving more reliable performance?

3 Method

Our main research objective is to develop a learning rule that is both biologically motivated and competitive with backpropagation in terms of convergence speed, computational efficiency, and overall performance. Perturbation-based methods already move in this direction: they rely on stochastic fluctuations and a global reward-like signal, making them inherently closer to biological learning than traditional backpropagation. At the same time, they require only forward computations with a small perturbation overhead, which makes them more parallelizable and computationally cheaper than a full backward pass. Because these methods remain mathematically coherent estimators of the gradient, the very quantity that backpropagation compute, they also serve as a conceptual bridge between biological learning principles and gradient-based optimization, aligning with our secondary research objective.

At the same time, standard perturbation methods do not fully achieve the biological plausibility or level of performance we ultimately seek. Although they rely on global reward signals, they lack the presynaptic–postsynaptic eligibility structure that characterizes genuine three-factor plasticity in the brain. Their updates also suffer from high variance, which limits learning efficiency and makes it unlikely that they can compete with backpropagation without further refinement. For this reason, it is natural to investigate whether adding additional biologically inspired components can both reduce variance and move perturbation-based learning closer to the mechanisms observed in biological systems.

The remainder of this chapter derives two proposed learning rules and outlines their computational and biological properties, in addition to their estimated performance. Each method is introduced and developed from the principles discussed above, and pseudocode is included to specify the corresponding training procedures. These formulations form the basis for the empirical comparisons in the next chapter.

Before we go through the two methods, we need to clarify some technical details. In both methods, we restrict the network to sigmoid activation functions,

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

which map all neuron activations to the interval $[0, 1]$. This constraint is useful when interpreting Hebbian-like factors, since negative activations would require additional sign handling in the update rules.

We also perturb all weights simultaneously during training. Because many parameters are perturbed at once, a single forward pass does not reliably indicate which specific perturbations influenced the loss; in principle, many forward evaluations are needed before the aggregated signal forms a stable gradient estimate. The Hebbian components introduced in the following sections are designed to guide this process by highlighting which perturbations most strongly affect the loss, thereby aiming at reducing variance and improving the efficiency of the perturbation-based updates.

3.1 Three-factor learning using weight perturbation

The first method modifies the standard weight–perturbation update in (15) by scaling each weight change with the presynaptic and postsynaptic activations, $x_i x_j$. This inserts the two local factors of a three–factor rule into an otherwise unchanged perturbation algorithm, where the global reward

signal provides the third factor. The result is a minimal Hebbian-like extension of perturbation-based learning. With the additional Hebbian factor $x_i x_j$, the update becomes

$$\Delta w_{ij} \;=\; -\eta\, x_i x_j \left(F(\theta) - b\right) \frac{\xi_{ij}}{\sigma}. \tag{16}$$

Recall that in (15), the parameter μ_i represented both weights and biases. We now separate these cases explicitly. The discussion will primarily focus on weight updates, but biases are updated analogously. For a bias parameter b_j, the update rule omits the presynaptic factor as it is not connected to it and becomes

$$\Delta b_j \;=\; -\eta\, x_j \left(F(\theta) - b\right) \frac{\xi_j}{\sigma}. \tag{17}$$

The update rule in (16) is fully consistent with biologically plausible three–factor plasticity. The term $x_i x_j$ provides a simple eligibility trace, marking a synapse when both the presynaptic and postsynaptic neurons are active in line with the classical "cells that fire together, wire together" principle [Hebb, 1949]. The global modulation $F(\theta) - b$ then determines the sign and magnitude of the resulting weight change, completing the three–factor structure.

From a computational standpoint, the Hebbian–perturbation rule in (16) lies between standard weight perturbation and full backpropagation. Like plain weight perturbation, the update remains fully parallelizable: all weights are perturbed and updated independently using a single global scalar signal, and the addition of a local Hebbian factor does not introduce any sequential dependencies. The Hebbian factor does increase the computational cost relative to standard weight perturbation, but the update remains substantially simpler than backpropagation. With sigmoid activations, the backpropagation weight gradient in (7) reduces to

$$\Delta w_{ij} \;=\; x_i^{(l-1)}\, x_j^{(l)} (1 - x_j^{(l)}) \sum_k w_{jk}^{(l+1)} \delta_k^{(l+1)}, \tag{18}$$

which requires evaluating activation derivatives and accumulating error signals from all downstream neurons. In contrast, the Hebbian–perturbation update in (16) involves only local activity variables x_i and x_j together with a single global scalar signal, replacing the expensive downstream summation with a simple multiplication. The additional cost of the Hebbian factor therefore places the method between standard weight perturbation and backpropagation. For the purpose of normalizing computational cost in the results section, we accordingly treat this method as requiring approximately 1.75 forward-pass equivalents per update, lying between standard weight perturbation (≈ 1.5) and backpropagation (≈ 2).

Mathematically, the effect of the additional factors is mixed. Multiplying the update by the presynaptic activity x_i is reasonable, since a weight connected to an inactive input cannot influence the output and should therefore not contribute to the update; this should help reduce variance when all weights are perturbed simultaneously. However, the postsynaptic activity x_j is not directly related to the neuron's sensitivity to its inputs, which is highest near the midpoint of the sigmoid activation and lowest when the neuron is saturated (0 or 1). As a result, the variance effect is unclear, and the update no longer corresponds to a perturbation-based gradient estimate, since both factors introduce bias. Although an unbiased estimator is not strictly required for learning, as the goal is ultimately to find a descent direction rather than the exact gradient, these limitations motivate exploring a learning rule that is both biologically plausible and more closely aligned with the underlying optimization problem.

3.2 Three-factor learning using node perturbation

Method 2 follows the same perturbation-based principle as before, but the noise is applied to the neuron activations rather than to the weights. Each activation is perturbed, the network output is evaluated, and the resulting change in loss provides an estimate of $\partial L/\partial x_j$, which is then translated into a weight update through the chain rule. Conceptually, the approach is similar to weight perturbation, but as we shall see, this leads directly to a three–factor learning rule.

To derive the update rule, we treat each postsynaptic activation as a stochastic variable of the form $x_j = \sigma(z_j) + \sigma\xi_j$, where $\xi_j \sim \mathcal{N}(0,1)$. Applying the score–function estimator to the perturbed activation yields an estimate of the derivative,

$$\widehat{\frac{\partial L}{\partial x_j}} \;=\; \big(F(\theta) - b\big)\,\frac{\xi_j}{\sigma}.$$

Using the chain rule,

$$\frac{\partial L}{\partial w_{ij}} \;=\; \frac{\partial L}{\partial x_j}\,\frac{\partial x_j}{\partial z_j}\,\frac{\partial z_j}{\partial w_{ij}},$$

and with the sigmoid activation $\sigma(z_j)$, we have

$$\frac{\partial x_j}{\partial z_j} \;=\; \sigma(z_j)\big(1-\sigma(z_j)\big) \;=\; x_j(1-x_j), \qquad \frac{\partial z_j}{\partial w_{ij}} \;=\; x_i.$$

Substituting these expressions together with the estimator above gives the final update,

$$\Delta w_{ij} \;=\; -\eta\, x_i\, x_j(1-x_j)\,\big(F(\theta) - b\big)\,\frac{\xi_j}{\sigma}. \tag{19}$$

The main focus in the following is on the weight updates, but the corresponding bias update follows by the same chain-rule argument. Since $\frac{\partial z_j}{\partial b_j} = 1$, the bias rule is obtained from (19) by omitting the presynaptic factor x_i, giving

$$\Delta b_j \;=\; -\eta\, x_j(1-x_j)\,\big(F(\theta) - b\big)\,\frac{\xi_j}{\sigma}. \tag{20}$$

The update rule in (19) has a direct interpretation as a biologically plausible three–factor learning rule. The presynaptic term x_i, the postsynaptic term $x_j(1-x_j)$, and the global modulatory signal $F(\theta) - b$ correspond to the components that determine synaptic change in classical models of neuromodulated plasticity. Stochastic perturbations at the level of neuronal activity are, just like weight perturbations, biologically plausible. Neurons in the brain exhibit irregular firing, and this intrinsic variability can serve as a source of learning-relevant noise [Seung, 2003]. Although the postsynaptic factor is a nonlinear function of the activation rather than just the activation itself, this remains consistent with the formal definition of three–factor rules, where the local term may be any function of the postsynaptic state [Gerstner et al., 2018]. The resulting update therefore satisfies the structural requirements for biological plausibility.

From a computational standpoint, the node–perturbation update in (19) has the same structure as standard weight perturbation and remains fully parallelizable. For a fixed postsynaptic neuron j, all factors except the presynaptic activity x_i are shared across incoming synapses: the postsynaptic sensitivity $x_j(1-x_j)$, the global feedback signal $F(\theta) - b$, and the perturbation ξ_j can be

computed once per neuron and reused for all associated weights. After these neuron-level quantities are formed, each synaptic update reduces to a single multiplication of the form $x_i \cdot C_j$, where $C_j = x_j(1 - x_j)(F(\theta) - b)\xi_j/\sigma$. The additional computations relative to plain weight perturbation therefore occur at the level of neurons rather than weights and become negligible in sufficiently wide layers, where the number of synapses per neuron is large. To remain conservative, we nonetheless account for this extra neuron-level overhead when normalizing computational cost. In the results section, we therefore treat node perturbation as requiring approximately 1.6 forward-pass equivalents per update, slightly higher than standard weight perturbation but still substantially cheaper than backpropagation.

Mathematically, this method is much cleaner than Method 1. The estimator remains unbiased, since no additional factors are introduced beyond the perturbation of the activations, making it analogous to the original weight–perturbation approach in this respect. At the same time, the local terms x_i and $x_j(1 - x_j)$ provide information about how strongly the weight w_{ij} contributes to the loss: the presynaptic factor x_i removes updates from inactive inputs, and the postsynaptic factor $x_j(1 - x_j)$ reflects the sensitivity of the neuron, peaking near the midpoint of the sigmoid activation and vanishing when it is saturated. These terms therefore introduce an eligibility trace that identifies which perturbations actually influenced the loss, which could reduce variance. Since neuron perturbations introduce far fewer stochastic degrees of freedom than weight perturbations, the method could further exhibit lower variance and faster learning. This is consistent with the analysis by Werfel et al. [2005], which shows that perturbation methods with lower dimensional noise sources can learn substantially faster. Taken together, these properties suggest that this method *may* provide the kind of biologically inspired alternative to backpropagation that we seek.

3.3 Summary of methods

We have now introduced four learning rules that will be compared empirically in the next chapter. The first is standard backpropagation, which provides an exact gradient and serves as the benchmark for both performance and convergence. The second is the classical weight–perturbation method, which does not require backward error propagation and relies solely on forward evaluations and a global reward signal. The third method augments weight perturbation with a Hebbian factor, yielding a minimal three–factor learning rule. The final method, node perturbation, applies stochasticity at the level of neuron activations and produces a three–factor rule that is both unbiased and more tightly aligned with the structure of gradient descent.

Each method carries distinct advantages and limitations, and theory alone does not determine which one will perform best in practice. To best possibly answer our research objective, namely: examine whether a biologically inspired learning rule can approximate or improve upon backpropagation in terms of convergence speed, computational efficiency, or overall training performance, we therefore need to test the methods in pracice. We do this by evaluating all four methods on the tasks introduced in the next chapter.

4 Case study

This chapter examines how four learning methods, standard backpropagation, weight perturbation, weight perturbation with a Hebbian eligibility factor, and node perturbation, perform on two supervised regression problems. The first problem is a simple one-dimensional sinusoid, while the second is a medium-scale real-world task based on the California Housing dataset. These experiments are not intended to provide a complete benchmark, but rather to indicate how promising the biologically inspired methods are when compared to backpropagation. By evaluating their behaviour across tasks of increasing complexity, the results contribute directly to the research objective of assessing whether perturbation-based learning rules can serve as viable alternatives to backpropagation.

4.1 Experimental set-up and implementation

4.1.1 Learning problems and datasets

The first task evaluates the learning rules on a simple nonlinear regression problem based on noisy samples from a sinusoidal function. The input consists of one-dimensional values uniformly spaced over a fixed interval, and the target is the corresponding value of a sine wave with added Gaussian noise. This task isolates core learning behaviour because the underlying mapping is smooth, low-dimensional, and free of structural complexity. Its purpose is therefore to test whether each method can learn a basic continuous function and establish stable convergence dynamics.

The second task assesses the methods on a medium-scale real-world regression problem using the California Housing dataset provided by the `scikit-learn` library. The dataset contains eight numerical features describing demographic and geographical properties of Californian housing districts, with the median house value as the target variable. It originates from the 1990 U.S. Census and is distributed through the `sklearn.datasets` module [Pedregosa et al., 2011]. This task introduces higher dimensionality, heterogeneity, and noise, making it suitable for evaluating whether the learning rules scale beyond synthetic settings. Its purpose is to test the behaviour of the methods under more realistic conditions and to identify how well they handle complex feature interactions.

4.1.2 Model architectures

For the sinusoid task, the models are shallow multilayer perceptrons with one input unit, two hidden layers of sizes 8 and 4, and one output unit. All hidden layers use the sigmoid activation function, and the output layer is linear. This architecture is intentionally small so that the task tests the learning dynamics of the methods rather than their representational capacity.

For the California Housing task, the models are deeper multilayer perceptrons with eight input units, hidden layers of sizes 128 and 64, and one output unit. As in the sinusoid setting, the hidden layers use the sigmoid activation function and the output layer is linear. This architecture provides sufficient capacity for a structured real-world regression problem while remaining simple enough to isolate differences between the learning algorithms.

4.1.3 Algorithms

Backpropagation is implemented using stochastic gradient descent on the exact loss gradient, and no further details are included since the algorithm is standard and fully specified in the literature. The perturbation-based methods are implemented as described in the following algorithms.

Algorithm 1 Weight perturbation training step

n indexes samples in the minibatch of size B

$\epsilon^{W,(n)},\ \epsilon^{b,(n)} \sim \mathcal{N}(0, \sigma^2)$ — weight and bias noise

$\tilde{W}^{(n)} \leftarrow W + \epsilon^{W,(n)}$ — perturbed weights

$\tilde{b}^{(n)} \leftarrow b + \epsilon^{b,(n)}$ — perturbed biases

$\tilde{y}^{(n)} \leftarrow f(x^{(n)}; \tilde{W}^{(n)}, \tilde{b}^{(n)})$ — forward pass with noise

$L^{(n)} \leftarrow \ell(\tilde{y}^{(n)}, y^{(n)})$ — loss

$R^{(n)} \leftarrow -L^{(n)}$ — reward

$\bar{R} \leftarrow \frac{1}{B} \sum_n R^{(n)}$ — baseline

$A^{(n)} \leftarrow R^{(n)} - \bar{R}$ — centered reward

$\Delta W \leftarrow \frac{\eta}{B} \sum_n A^{(n)}\, \epsilon^{W,(n)} / \sigma$ — weight update

$\Delta b \leftarrow \frac{\eta}{B} \sum_n A^{(n)}\, \epsilon^{b,(n)} / \sigma$ — bias update

$W \leftarrow W + \Delta W$

$b \leftarrow b + \Delta b$

Algorithm 2 Three-factor weight perturbation training step

n indexes samples in the minibatch of size B

$\epsilon^{W,(n)},\ \epsilon^{b,(n)} \sim \mathcal{N}(0, \sigma^2)$ — weight and bias noise

$\tilde{W}^{(n)} \leftarrow W + \epsilon^{W,(n)}$ — perturbed weights

$\tilde{b}^{(n)} \leftarrow b + \epsilon^{b,(n)}$ — perturbed biases

$\tilde{y}^{(n)},\ x_{\text{in}}^{(n)},\ x_{\text{out}}^{(n)} \leftarrow f(x^{(n)}; \tilde{W}^{(n)}, \tilde{b}^{(n)})$ — forward pass with noise

$L^{(n)} \leftarrow \ell(\tilde{y}^{(n)}, y^{(n)})$ — loss

$R^{(n)} \leftarrow -L^{(n)}$ — reward

$\bar{R} \leftarrow \frac{1}{B} \sum_n R^{(n)}$ — baseline

$A^{(n)} \leftarrow R^{(n)} - \bar{R}$ — centered reward

$E^{(n)} \leftarrow x_{\text{in}}^{(n)} x_{\text{out}}^{(n)}$ — Hebbian eligibility trace

$\Delta W \leftarrow \frac{\eta}{B} \sum_n A^{(n)}\, \epsilon^{W,(n)}\, E^{(n)} / \sigma$ — three-factor weight update

$\Delta b \leftarrow \frac{\eta}{B} \sum_n A^{(n)}\, \epsilon^{b,(n)}\, x_{\text{out}}^{(n)} / \sigma$ — bias update

$W \leftarrow W + \Delta W$

$b \leftarrow b + \Delta b$

Algorithm 3 Three-factor node perturbation training step

n indexes samples in the minibatch of size B

$\epsilon^{(n)} \sim \mathcal{N}(0, I)$ — noise for each neuron in sample n

$a_0^{(n)} \leftarrow x^{(n)}$ — clean input activations

$\tilde{a}_0^{(n)} \leftarrow x^{(n)}$ — noisy pass starts identical to clean

for $\ell = 1, \ldots, L$ **do**

$a_\ell^{(n)} \leftarrow \phi(W_\ell a_{\ell-1}^{(n)} + b_\ell)$ — forward pass (clean)

$\tilde{a}_\ell^{(n)} \leftarrow \phi(W_\ell \tilde{a}_{\ell-1}^{(n)} + b_\ell) + \sigma\, \epsilon_\ell^{(n)}$ — forward pass (noisy)

end for

$\tilde{y}^{(n)} \leftarrow \tilde{a}_L^{(n)}$ — noisy network output

$L^{(n)} \leftarrow \ell(\tilde{y}^{(n)}, y^{(n)})$

$R^{(n)} \leftarrow -L^{(n)}$ — reward signal

$\bar{R} \leftarrow \frac{1}{B} \sum_n R^{(n)}$ — baseline

$A^{(n)} \leftarrow R^{(n)} - \bar{R}$ — centered reward

$g(a_\ell^{(n)}) \leftarrow a_\ell^{(n)}(1 - a_\ell^{(n)})$ — sigmoid derivative

$E_\ell^{(n)} \leftarrow a_{\ell-1}^{(n)}\, g(a_\ell^{(n)})$ — eligibility trace (pre × post)

$\delta_\ell^{(n)} \leftarrow A^{(n)}\, E_\ell^{(n)}\, \epsilon_\ell^{(n)} / \sigma$ — global × eligibility × noise/var

$\Delta W_\ell \leftarrow \frac{\eta}{B} \sum_n \delta_\ell^{(n)}$

$\Delta b_\ell \leftarrow \frac{\eta}{B} \sum_n A^{(n)}\, \epsilon_\ell^{(n)}\, g(a_\ell^{(n)}) / \sigma$

$W_\ell \leftarrow W_\ell + \Delta W_\ell$

$b_\ell \leftarrow b_\ell + \Delta b_\ell$

4.1.4 Hyperparameters

Hyperparameters were selected using a grid search conducted independently for each learning method and task. For each method, we first identified a reasonable operating regime through exploratory runs and then evaluated a compact grid centered around this regime. Hyperparameter selection was based on training loss behaviour, taking into account convergence speed, final loss level, and stability across runs. Batch size and number of training epochs were not treated as tunable hyperparameters, but were instead chosen to ensure stable convergence for all methods on each task. All reported results use the best-performing hyperparameter configuration identified by this procedure.

Hyperparameter search space

Sinusoid task:

- **Backpropagation:** learning rate $\{0.05, 0.1, 0.2, 0.5, 0.7\}$.
- **Weight perturbation:** learning rate $\eta \in \{0.3, 0.5, 0.7, 0.9\}$, noise scale $\sigma \in \{0.05, 0.1, 0.2, 0.3\}$.
- **Three-factor weight perturbation:** learning rate $\eta \in \{0.3, 0.5, 0.7, 0.9\}$, noise scale $\sigma \in \{0.05, 0.1, 0.2, 0.3\}$.
- **Three-factor node perturbation:** learning rate $\eta \in \{0.07, 0.1, 0.2\}$, noise scale $\sigma \in \{0.05, 0.1, 0.2, 0.3\}$.

California Housing task:

- **Backpropagation:** learning rate $\{0.007, 0.01, 0.02, 0.03\}$.
- **Weight perturbation:** learning rate $\eta \in \{0.03, 0.05, 0.07, 0.1\}$, noise scale $\sigma \in \{0.05, 0.1, 0.2\}$.
- **Three-factor weight perturbation:** learning rate $\eta \in \{0.03, 0.05, 0.07, 0.1\}$, noise scale $\sigma \in \{0.05, 0.1, 0.2\}$.
- **Three-factor node perturbation:** learning rate $\eta \in \{0.007, 0.01, 0.02, 0.03\}$, noise scale $\sigma \in \{0.05, 0.1, 0.2, 0.3\}$.

Selected hyperparameters

The following configurations achieved the best performance and are used for all results reported in the next section.

Sinusoid task: Batch size 128, epochs 40.

- **Backpropagation:** learning rate 0.5.
- **Weight perturbation:** $\eta = 0.7$, $\sigma = 0.2$.
- **Three-factor weight perturbation:** $\eta = 0.9$, $\sigma = 0.1$.
- **Three-factor node perturbation:** $\eta = 0.2$, $\sigma = 0.1$.

California Housing task: Batch size 256, epochs 100.

- **Backpropagation:** learning rate 0.02.
- **Weight perturbation:** $\eta = 0.05$, $\sigma = 0.1$.
- **Three-factor weight perturbation:** $\eta = 0.05$, $\sigma = 0.1$.
- **Three-factor node perturbation:** $\eta = 0.01$, $\sigma = 0.1$.

4.1.5 Evaluation metrics

The models are evaluated using the mean squared error (MSE) between the predicted outputs and the target values. During training, the MSE is recorded at each minibatch to monitor convergence behaviour and stability. This metric directly reflects the quality of the learned input–output mapping and is therefore used as the primary measure of training performance.

To address the primary research objective, we evaluate learning dynamics along two axes. Convergence is measured both as a function of training epochs and as a function of computational cost, expressed in forward-pass equivalents using the cost normalizations derived earlier. These cost estimates are not exact, but are intentionally conservative for all perturbation-based methods and based on an average-case estimate for backpropagation. This normalization enables a consistent and transparent comparison between methods with different per-update computational requirements.

For the sinusoid task, each method is evaluated over ten independent runs with different random initializations. We report both the average performance across runs and the best-performing run, where "best" is defined as the run that reaches an MSE below 0.1 in the fewest updates. Results are plotted both per epoch and per unit of compute. In addition, we plot the variance of the MSE across runs as a function of training progress, computed at each minibatch, and report the average variance over the full training run as a summary measure of stability. The same evaluation protocol is applied to the California Housing task using five runs and an MSE threshold of 0.6.

4.2 Results

We now present the empirical results for the sinusoid regression task and the California Housing regression task. For each problem, we compare backpropagation with the proposed perturbation-based methods in terms of convergence behaviour, computational efficiency, and stability across runs. Results are reported first as a function of training progress per epoch, then as a function of computational cost expressed in forward-pass equivalents, and finally in terms of variance across independent runs. Throughout, we focus on relative performance trends rather than absolute error levels, as the primary objective is to assess how closely biologically inspired learning rules can match backpropagation under comparable conditions.

4.2.1 Loss as a function of training epochs

Sinusoid regression: In the average-loss plot (Figure 3a), backpropagation exhibits the fastest overall reduction in training error. Weight perturbation converges more slowly but ultimately reaches a slightly lower final MSE. Node perturbation displays pronounced instability, with large fluctuations throughout training; however, its lowest loss values are comparable to those achieved by backpropagation and weight perturbation, and it reaches these levels at a similar epoch. Three-factor weight perturbation converges more slowly than the other methods and stabilizes at a higher training loss.

In the best-performing run (Figure 3b), backpropagation achieves both the fastest convergence and the lowest final training error, although the descent is characterized by substantial variability before stabilizing. Weight perturbation converges slightly more slowly and plateaus at a marginally higher MSE. Three-factor weight perturbation shows a delayed onset of consistent descent but eventually converges to a level similar to weight perturbation. Node perturbation appears as a highly oscillatory variant of weight perturbation, reaching comparable minima at similar speeds but with persistent fluctuations throughout training.

California Housing regression: Figure 4a shows the average training loss as a function of epochs for the California Housing regression task. Backpropagation converges faster than all perturbation-based methods and reaches the lowest final training MSE. Its learning curve is smooth and stable throughout training. Standard weight perturbation converges more slowly and plateaus at a slightly higher MSE, but exhibits relatively stable behaviour across epochs. The three-factor weight perturbation method converges more slowly and reaches the highest MSE among the methods. Node perturbation displays substantially higher variability across epochs. Its average loss decreases at a rate comparable to weight perturbation early in training, but the curve remains noisy and settles at a higher average MSE.

The best-performing runs, Figure 4b, show a similar ordering. Backpropagation reaches the lowest MSE in the fewest epochs, confirming its strongest best-case performance. Weight perturbation converges more slowly and stabilizes at a slightly higher loss. Three-factor weight perturbation again shows delayed convergence and a higher final MSE. Node perturbation is capable of reaching loss values comparable to standard weight perturbation in its best run, but the training trajectory remains visibly spiky throughout, indicating persistent instability.

4.2.2 Loss as a function of computational cost

This subsection is based on the same training runs as above, and the loss values are therefore identical. The only difference is that the horizontal axis is rescaled by the estimated computational cost of each method, which shifts the trajectories relative to one another. Accordingly, the discussion focuses on differences in apparent convergence speed rather than on properties already described in the previous subsection.

Sinusoid regression: When averaged across runs (Figure 5a), backpropagation decreases the loss more rapidly at the very beginning of training when measured per unit of compute. Weight perturbation continues to make steady progress and ultimately converges faster in terms of total computational cost, reaching low-loss regions earlier than backpropagation. Node perturbation follows a similar overall convergence speed but exhibits substantial fluctuations throughout training. The three-factor weight perturbation method converges more slowly than the other methods in compute-normalized terms.

In the best-performing runs (Figure 5b), backpropagation, weight perturbation, and node perturbation show very similar convergence speed when measured per unit of compute, reaching low-loss regions at roughly the same computational cost. Differences between these methods are therefore primarily reflected in trajectory smoothness rather than convergence speed. The three-factor weight perturbation method again converges more slowly and does not match the convergence speed of the other methods.

California Housing regression: Backpropagation, weight perturbation, and node perturbation exhibit similar convergence speed when measured per unit of compute (Figure 6a). All three methods reduce the loss at comparable rates under compute-normalized scaling. Node perturbation follows a similar speed profile but with substantially higher variability throughout training. Three-factor weight perturbation converges more slowly than the other methods in terms of computational cost.

In the best-performing runs (Figure 6b), backpropagation, weight perturbation, and node perturbation reach low error levels at similar rates per unit of compute. Differences between these methods are therefore not driven by convergence speed but by the characteristics of their trajectories. Three-factor weight perturbation again shows slower convergence and does not match the

Table 1: Average training loss variance across 10 runs for the sinusoid regression task.

Method	**Mean Variance**	**Mean Std. Dev.**
Backpropagation	0.01110	0.09460
Weight perturbation	0.00196	0.03305
Three-factor weight perturbation	0.00445	0.06209
Three-factor node perturbation	0.01812	0.09688

Table 2: Average training loss variance across 5 runs for the California Housing regression task.

Method	**Mean Variance**	**Mean Std. Dev.**
Backpropagation	7.9×10^{-5}	0.00693
Weight perturbation	2.46×10^{-4}	0.01283
Three-factor weight perturbation	3.11×10^{-4}	0.01433
Three-factor node perturbation	1.71×10^{-2}	0.09404

compute-efficiency of the other methods.

4.2.3 Variance across runs

Sinusoid regression:

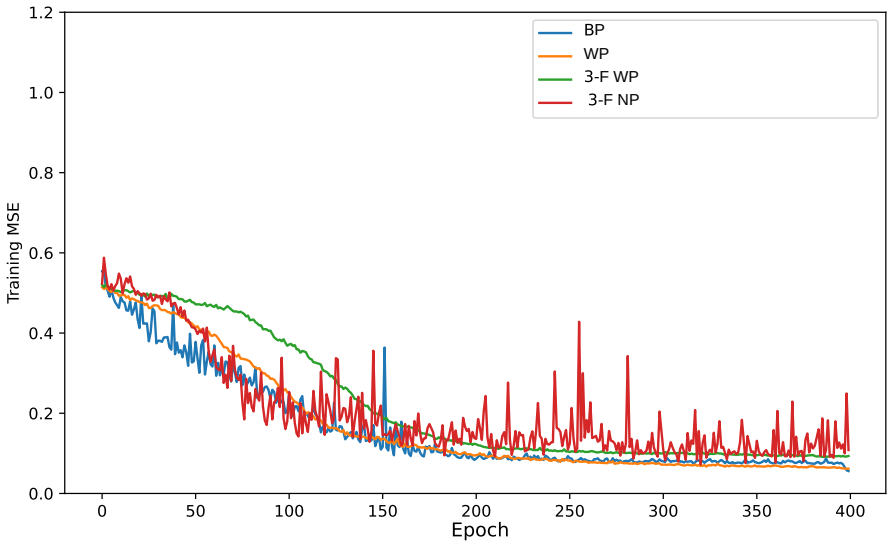
Figure 7 shows the variance of the training loss across independent runs as a function of training progress for the sinusoid task. Weight perturbation exhibits the lowest variance throughout training, with some variability early on but highly consistent behavior toward the end across random initializations. Three-factor weight perturbation shows higher variance than plain weight perturbation, but remains more stable than both backpropagation and node perturbation. Backpropagation displays moderate variance across the training process. Three-factor node perturbation exhibits the highest variance among all methods, with pronounced spikes persisting throughout training, indicating substantial variability both across runs and within individual runs.

The average variance over the full training run is summarized in Table 1. Weight perturbation achieves the lowest mean variance, followed by three-factor weight perturbation, backpropagation, and three-factor node perturbation. These results highlight clear differences in stability across methods, even when final performance levels are comparable.

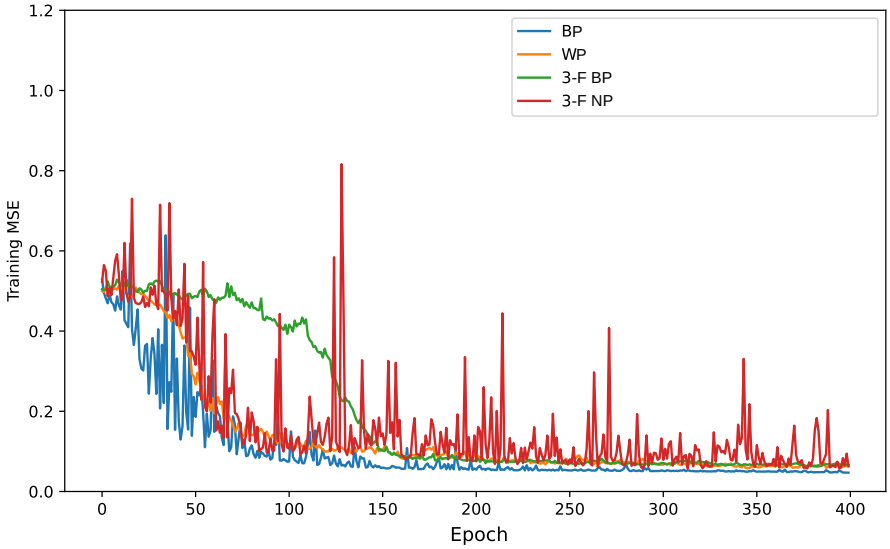
California Housing regression:

Figure 8 shows the variance of the training loss across independent runs for the California Housing task. Backpropagation exhibits the lowest variance throughout training, indicating highly consistent convergence across random initializations. Weight perturbation and three-factor weight perturbation show slightly higher, but still relatively low, variance. In contrast, three-factor node perturbation displays substantially higher variance across all stages of training, with large fluctuations persisting even after apparent convergence.

The average variance over the full training run is summarized in Table 2. Backpropagation achieves the lowest mean variance, followed by weight perturbation and three-factor weight perturbation, while node perturbation exhibits variance more than an order of magnitude larger than that of the other methods.

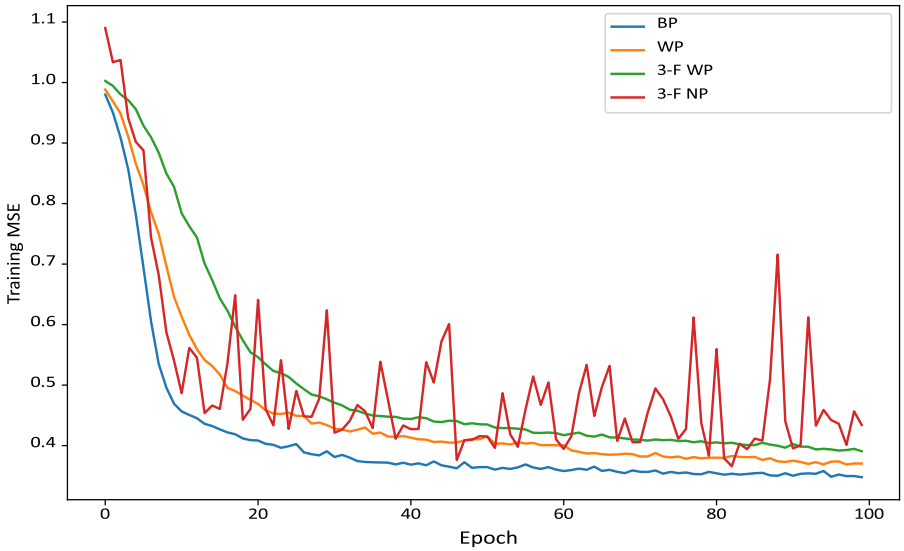


(a) Average training MSE across runs.

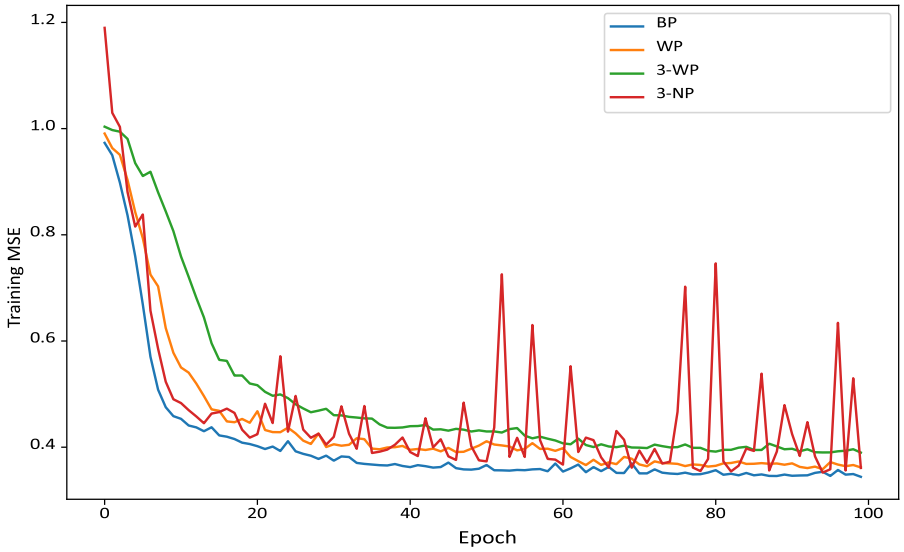


(b) Best-performing run (fastest to reach target MSE of 0.1).

Figure 3: Training loss as a function of epochs for the sinusoid regression task. BP = Backpropagation, WP = Weight Perturbation, 3-F WP = Three Factor Weight Perturbation, 3-F NP = Three Factor Node Perturbation

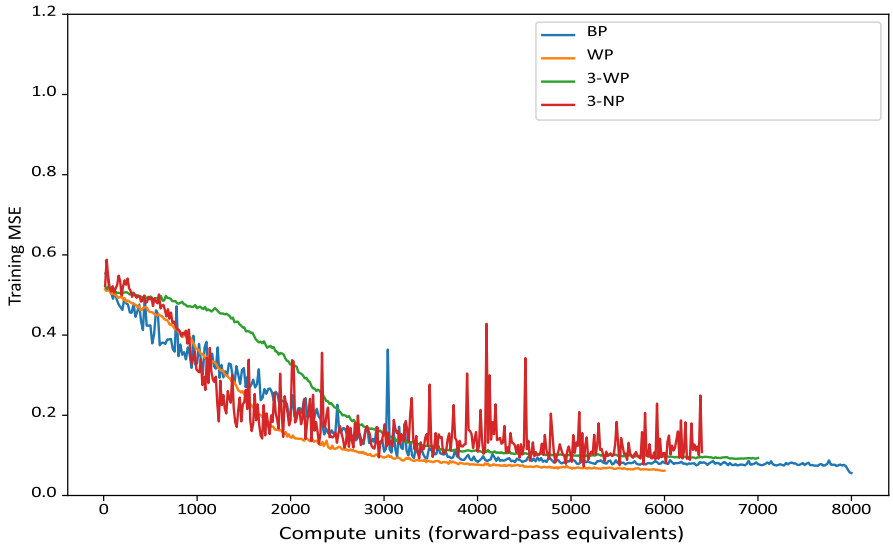


(a) Average training MSE across runs.

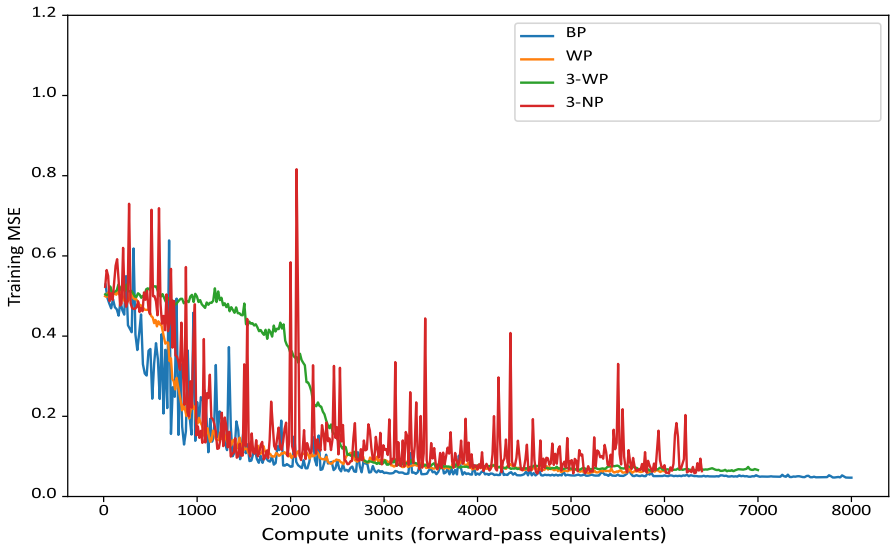


(b) Best-performing run (fastest to reach target MSE 0.6).

Figure 4: Training loss as a function of epochs for the California Housing regression task. BP = Backpropagation, WP = Weight Perturbation, 3-WP = Three Factor Weight Perturbation, 3-NP = Three Factor Node Perturbation

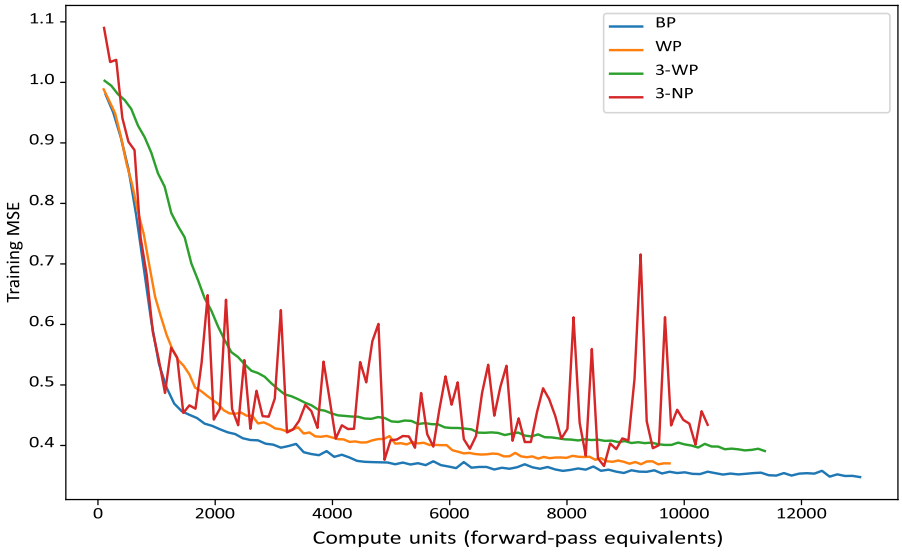


(a) Average training MSE across runs.

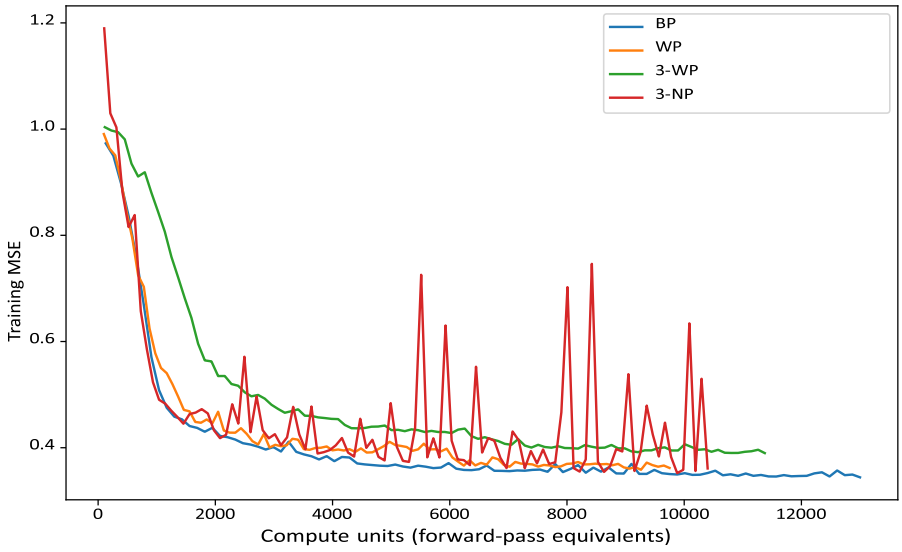


(b) Best-performing run (fastest to reach target MSE 0.1).

Figure 5: Training loss as a function of computational cost for the sinusoid regression task. BP = Backpropagation, WP = Weight Perturbation, 3-WP = Three Factor Weight Perturbation, 3-NP = Three Factor Node Perturbation

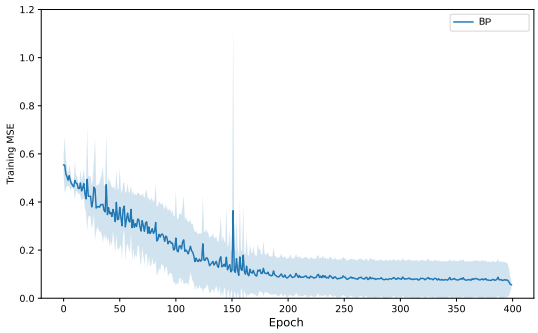


(a) Average training MSE across runs.

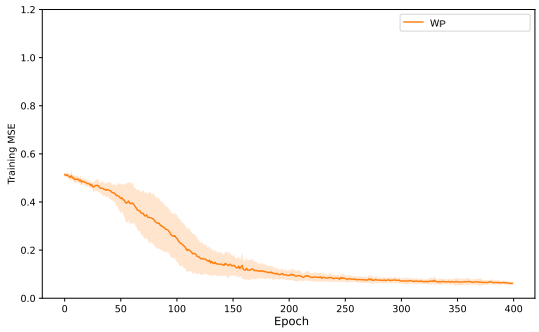


(b) Best-performing run (fastest to reach target MSE 0.6).

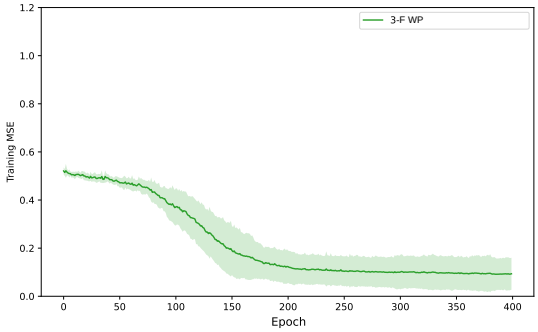
Figure 6: Training loss as a function of computational cost for the California Housing regression task. BP = Backpropagation, WP = Weight Perturbation, 3-WP = Three Factor Weight Perturbation, 3-NP = Three Factor Node Perturbation



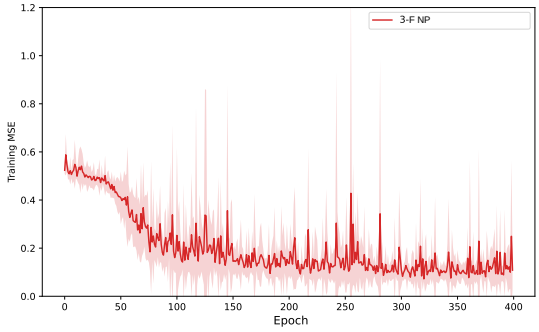
(a) Backpropagation: variance across runs during training.



(b) Weight perturbation: variance across runs during training.

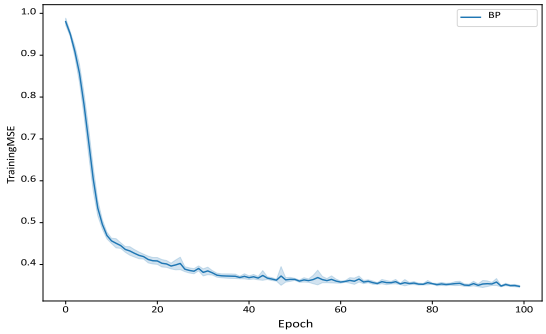


(c) Three-factor weight perturbation: variance across runs during training.

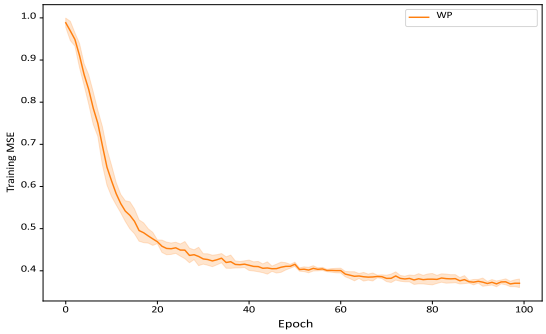


(d) Three-factor node perturbation: variance across runs during training.

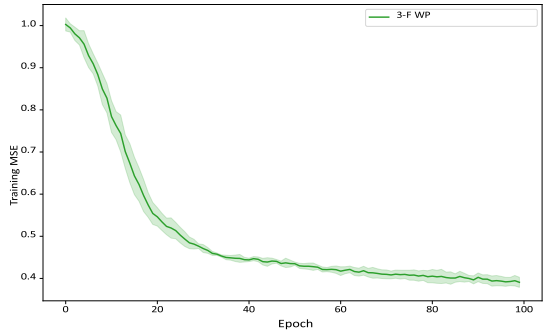
Figure 7: Training loss variance across runs for the sinusoid regression task. Variance is computed per minibatch over 10 random initializations.BP = Backpropagation, WP = Weight Perturbation, 3-F WP = Three Factor Weight Perturbation, 3-F NP = Three Factor Node Perturbation



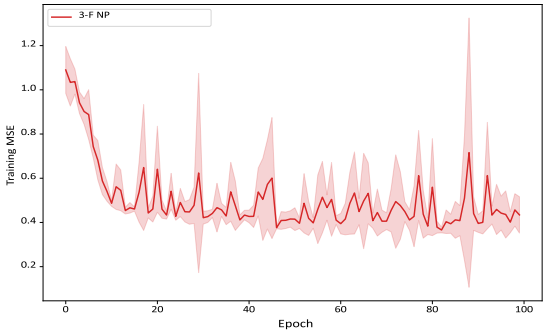
(a) Backpropagation: variance across runs during training.



(b) Weight perturbation: variance across runs during training.



(c) Three-factor weight perturbation: variance across runs during training.



(d) Three-factor node perturbation: variance across runs during training.

Figure 8: Training loss variance across 5 runs for the California Housing regression task. Variance is computed per minibatch over independent random initializations.BP = Backpropagation, WP = Weight Perturbation, 3F-WP = Three Factor Weight Perturbation, 3F-NP = Three Factor Node Perturbation

5 Concluding remarks

5.1 Discussion

In this chapter, we interpret the results of the previous chapter in the context of the problem formulation introduced earlier. The purpose of the discussion is to assess what the empirical findings imply about the extent to which our biologically inspired learning rules can approach or improve the behavior of backpropagation. In particular, we analyze the results along five dimensions: the ability to reach low-loss solutions, convergence efficiency, stability, the interplay between biological plausibility and mathematical grounding, and the limitations imposed by the experimental setup.

5.1.1 Ability to reach low-loss solutions

The results suggest that on the simple sinusoid regression task, perturbation-based methods are capable of achieving minima comparable to those reached by backpropagation, whereas on the more complex California Housing task they tend to stall at slightly higher loss values. While this gap is consistent across runs, it remains relatively small, indicating that the perturbation methods are still learning meaningful solutions rather than failing outright. One possible explanation for the gap is that perturbation-based updates lack an intrinsic mechanism for reducing update magnitudes near stationary points: backpropagation naturally produces smaller updates as the gradient norm decreases close to a minimum, while perturbation methods continue to apply stochastic updates even when near an optimum, potentially maintaining a residual noise floor. Notably, augmenting weight perturbation with an additional Hebbian factor consistently leads to worse convergence and higher final loss values, an effect that is analyzed further in the mathematics-versus-biological-plausibility discussion. In the context of the problem formulation, these findings indicate that biologically inspired perturbation-based algorithms are capable of learning and can approximate the solutions found by backpropagation on simple tasks, but that additional structure may be required to consistently match or exceed backpropagation performance as task complexity increases.

5.1.2 Convergence efficiency

The results indicate that perturbation-based methods are capable of reaching their respective minima using a similar amount of computation as backpropagation. At first glance this may seem counterintuitive, since the updates are stochastic and one might expect additional iterations to be required before useful descent directions emerge. However, each perturbation update requires fewer computations than a full backward pass, which offsets the noisier learning signal. We further expected node perturbation to converge faster than standard weight perturbation due to its lower number of stochastic degrees of freedom, but this was not observed in practice, as both methods exhibit comparable convergence behavior. Using the conservative cost estimates derived earlier, we observe that the perturbation-based methods are all able to reach their minima with comparable computational efficiency to backpropagation on the considered tasks.

An additional factor not captured directly by the experiments is parallelizability. Perturbation-based updates are fully parallel across layers and do not involve the sequential backward dependency structure of backpropagation. While the present experiments do not measure wall-clock runtime and do not explicitly optimize GPU utilization, the comparable convergence observed per unit of compute provides a meaningful starting point for assessing performance per unit time. In settings where parallel execution can be exploited effectively, beyond what is already achieved through

minibatch parallelism, this structural advantage suggests that perturbation-based learning rules may improve training efficiency relative to backpropagation.

5.1.3 Stability and variance

The results reveal clear differences in stability across learning methods, as measured by the variance of the training loss across runs. Contrary to initial expectations, node perturbation exhibits substantially higher variability than the other methods, with pronounced loss spikes persisting throughout training. This is notable given that node perturbation introduces fewer stochastic degrees of freedom than weight perturbation, and was therefore expected to reduce, rather than increase, variance. A plausible interpretation is that perturbing a neuron directly influences many connected weights at once, so an unfavorable perturbation can temporarily disrupt a larger part of the network. In contrast, weight perturbation and three-factor weight perturbation display moderate variance that remains comparable to backpropagation, indicating that stochasticity at the level of individual parameters leads to more controlled updates. Overall, the results suggest that while all perturbation-based methods are capable of learning, node perturbation would likely require variance-mitigation mechanisms to be practically attractive, whereas the weight-based methods already exhibit reasonable stability.

5.1.4 Mathematical grounding and biological inspiration

Biologically inspired learning rules do appear capable of learning meaningful structure, but the results suggest that this is only the case when the learning rule remains grounded in the underlying mathematics of the optimization problem. Both weight perturbation and node perturbation are able to achieve low loss values, indicating that biologically plausible mechanisms can approximate the solutions found by backpropagation, at least on the considered tasks. In contrast, augmenting weight perturbation with an additional Hebbian factor consistently degrades performance, despite increasing biological realism. This suggests that incorporating biologically motivated components in an ad hoc manner, without regard for how they affect the mathematical properties of the update, such as bias in the gradient estimator, can be detrimental to learning. Taken together, these findings indicate that biological inspiration can be valuable, but only insofar as it preserves essential mathematical structure; biological plausibility alone is not sufficient.

A complementary implication of these results is that mathematical grounding need not come at the expense of biological plausibility. In particular, the node perturbation rule demonstrates that an unbiased estimator of the gradient with respect to the weights can be constructed using only local activity variables and a global modulatory signal, ingredients that are widely regarded as biologically realistic. Although the brain cannot explicitly compute or propagate exact gradients as in backpropagation, it may not need to: stochastic perturbations provide a principled mechanism for estimating the same underlying quantity in expectation. From this viewpoint, the apparent biological implausibility of backpropagation lies not in the objective it optimizes, but in the specific way it is computed. Perturbation-based learning therefore suggests that biological systems may implement a noisy but mathematically faithful approximation to gradient descent, placing their learning dynamics closer to backpropagation than is often assumed.

5.1.5 Limitations of the experimental setup

The conclusions that will be drawn are necessarily limited by the scope of the experimental setup. The evaluation is based on two regression tasks, which is sufficient to assess feasibility and relative behavior, but not to support general performance claims across tasks, domains, or model classes. In particular, conclusions regarding convergence, stability, and variance are sensitive to initialization and stochastic effects, and would be more robust under evaluation across a broader range of problems, architectures, and random seeds. In addition, computational efficiency is assessed using approximate and intentionally conservative cost models rather than explicit operation counts or wall-clock measurements. As such, the results should be interpreted as an initial empirical exploration of biologically inspired learning rules, aligned with the goals of this project, rather than as definitive evidence of their performance in general settings

5.2 Future work

Several directions for future work follow naturally from the results of this project. First, the proposed learning rules should be evaluated on a broader and more diverse set of tasks and architectures, including deeper networks and problems beyond regression, in order to assess the generality of the observed behavior. Second, a more precise analysis of computational efficiency would require explicit operation counts and wall-clock measurements, ideally combined with implementations optimized for GPU execution, to move beyond approximate compute normalizations. Third, further algorithmic refinements should be explored and evaluated in a comparative manner. Perturbation-based methods may benefit from variance mitigation techniques, alternative noise schedules, normalization schemes, momentum, or different activation functions, many of which are known to improve optimization in standard backpropagation. Any such modifications should be assessed alongside equally strong backpropagation baselines, rather than vanilla stochastic gradient descent, to ensure fair comparison. Finally, hybrid training strategies, such as using perturbation-based learning during early optimization and switching to backpropagation for fine-tuning, could be explored as a way to combine fast initial progress with precise convergence.

5.3 Conclusion

This project set out with two closely related objectives. The primary objective was to explore whether biologically plausible learning rules can serve as viable alternatives to backpropagation by learning meaningful solutions on supervised learning tasks. The secondary objective was to investigate whether such learning rules can help bridge the conceptual and mathematical gap between biological learning mechanisms and gradient-based optimization. Both objectives were exploratory in nature, with the aim of identifying promising directions rather than establishing general performance guarantees or claims of superiority over backpropagation.

The empirical results indicate that both objectives were partially met. Perturbation-based learning rules were able to learn meaningful solutions on both considered tasks. On the simpler problem, they achieved performance comparable to backpropagation, while on the more challenging task they converged to slightly higher loss values despite clearly learning the underlying structure. The accompanying theoretical analysis shows that perturbation-based learning admits an interpretation as an unbiased gradient estimator under expectation using only local activity variables and a global modulatory signal, providing a principled link between biological learning constraints and gradient-based optimization, such as backpropagation. Backpropagation nevertheless remained the

most stable and consistently best-performing method across experiments, and the results do not support claims of general superiority for the proposed alternatives.

These findings have several implications for future work. While the results do not suggest that biologically plausible learning rules currently outperform backpropagation, they indicate that alternatives grounded in biological constraints can be both mathematically principled and practically effective on simple tasks. Given the central role of backpropagation in modern machine learning, even modest improvements, task-specific advantages, or gains in interpretability or efficiency would be of substantial significance. Consequently, this work motivates further investigation into biologically grounded learning rules, particularly through broader algorithmic refinements of our methods and more extensive empirical evaluation, positioning this line of research as a promising direction for future study.

6 Sustainability

This project relates indirectly to the United Nations' sustainability goals through its focus on computational efficiency. One part of the primary objective is to explore learning algorithms that may be more computationally efficient than standard backpropagation. Improved efficiency in training neural networks can reduce energy consumption and associated environmental impact, particularly as machine learning models continue to grow in scale and are deployed on energy-intensive computing infrastructure. In this sense, the work aligns modestly with broader sustainability considerations by contributing to research on more energy-efficient learning methods.

7 Use of Artificial Intelligence

Artificial intelligence tools have been used as support during the completion of this project. ChatGPT was used for discussion, background research, and for reviewing and commenting on written text to improve clarity and structure. In addition, GitHub Copilot was used as a programming aid for debugging and for discussing code fragments. These tools were used in a supportive capacity and not to produce full paragraphs or complete the assignment as a whole. All final decisions regarding content, analysis, implementation, and conclusions were made by the author, who takes full responsibility for the work.

References

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018. URL `http://jmlr.org/papers/v18/17-468.html`.

Stanislas Dehaene. *How We Learn: The New Science of Education and the Brain*. Penguin Books Ltd, 2021.

Wulfram Gerstner and Werner M. Kistler. Mathematical formulations of hebbian learning. *Biological cybernetics*, 87(5-6):404–415, 2002. ISSN 0340-1200.

Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: Experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53–, 2018. ISSN 1662-5110.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 2017. ISBN 978-0-262-03561-3. URL `https://contents.bibs.aws.unit.no/files/images/large/8/1/0262035618.jpg`.

Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. John Wiley, New York, 1949. ISBN 0471367273.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991. ISSN 0893-6080.

Eric R. Kandel, John D. Koester, Sarah Mack, and Steven A. Siegelbaum. *Principles of neural science*. McGraw-Hill, New York, sixth edition edition, 2021. ISBN 9781259642234. URL `https://contents.bibs.aws.unit.no/files/images/large/4/3/9781259642234.jpg`.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. ISSN 0018-9219.

Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature reviews. Neuroscience*, 21(6):335–346, 2020. ISSN 1471-003X.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982. ISSN 0303-6812.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 2011. ISSN 1532-4435.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature (London)*, 323(6088):533–536, 1986. ISSN 0028-0836.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2952–2961. PMLR, 2017.

Wolfram Schultz, Peter Dayan, and P. Read Montague. A neural substrate of prediction and reward. *Science (American Association for the Advancement of Science)*, 275(5306):1593–1599, 1997. ISSN 0036-8075.

H.Sebastian Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron (Cambridge, Mass.)*, 40(6):1063–1073, 2003. ISSN 0896-6273.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Justin Werfel, Xiaohui Xie, and H. Sebastian Seung. Learning curves for stochastic gradient descent in linear feedforward networks. *Neural computation*, 17(12):2699–2718, 2005. ISSN 0899-7667.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. ISSN 0885-6125.