Predictive Coding Inspires Effective Alternatives to Backpropagation

Yuhang Song

Somerville College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2021

Abstract

The brain has developed a sophisticated hierarchical structure where information is processed across multiple layers (Van Essen et al. 1992), which is critical for forming abstraction from raw information. Inspired by this, modern machine learning is largely built on deep artificial neural networks (LeCun, Bengio, et al. 2015), where neurons are organized in layers, and synaptic weights connect neurons of different layers. One important piece of learning is to correct errors in one's predictions, and the hierarchical structure requires spreading the errors in one's predictions across multiple layers and neurons — creating the core challenge of learning known as "credit assignment" (Lillicrap, Santoro, et al. 2020). How the brain solves credit assignment is a key question to understand learning and for creating artificial learning machines, and many recent studies (Lillicrap, Santoro, et al. 2020; Richards, Lillicrap, et al. 2019; Whittington and Bogacz 2019; Zipser and Andersen 1988; Singer et al. 2018; Whittington, T. H. Muller, et al. 2020; Banino et al. 2018) approached it from the perspective of backpropagation (Werbos 1974; Rumelhart et al. 1985; Parker 1985; LeCun, Bengio, et al. 2015). However, it has been questioned whether it is possible for the brain to implement backpropagation exactly (Crick 1989; Grossberg 1987), and learning in the brain seems to be more efficient than backpropagation in several other critical aspects (Tsividis et al. 2017). On the other hand, neuroscience researchers have developed many biologically plausible models to solve the credit assignment problem (Rao and Ballard 1999; Whittington and Bogacz 2017; Whittington and Bogacz 2019; Lillicrap, Santoro, et al. 2020). However, biologically plausible models are still behind backpropagation in terms of generalization and efficiency. The thesis is inspired by this mismatch: the brain seems to be more advanced than backpropagation, while biologically plausible models learn no better than machine learning models. Thus, the overall goal of the thesis is to demonstrate biologically plausible models can inspire alternative and potentially better solutions for credit assignment than backpropagation, and we take inspiration mainly from predictive coding (PC). Specifically, I first propose a variant of PC that is equivalent to backpropagation. I then propose that the original PC has advantages over backpropagation, implementing a fundamentally different principle from backpropagation, which I call "prospective configuration". Inspired by the above two findings, I finally propose another variant of PC that is potentially more efficient than backpropagation and has a similar generalization quality as backpropagation.

Predictive Coding Inspires Effective Alternatives to Backpropagation

Yuhang Song

Somerville College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2021

Declarations

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text.

Yuhang Song
October 2021

Acknowledgements

I would like to thank my supervisors (in alphabetical order): Professor Rafal Bogacz and Professor Thomas Lukasiewicz. I would also like to thank the many people that helped through comments and discussions during the writing of the various papers composing this thesis (in alphabetical order): Abi Aryan, Andrzej Wojcicki, Beren Millidge, Jianyi Wang, Mai Xu, Shangtong Zhang, Tommaso Salvatori, Zhenghua Xu, Zihan Ding, and Lianlong Wu. I would finally like to thank my fiancee Danni Du for her continuous support throughout my academic adventure in Oxford.

I would like to thank the China Scholarship Council for supporting me with a State Scholarship of China, J.P. Morgan Chase for supporting me with J.P. Morgan AI Research Awards, the China Oxford Scholarship Fund for supporting me with an Honorary China Oxford Award, and Somerville College for supporting me with a Special Project Grant.

JPMorgan Chase & Co. This research was funded in part by JPMorgan Chase & Co. Any views or opinions expressed herein are solely those of the authors listed and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments, or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction.

Contents

List of Figures

List of Tables

List of Algorithms

List of Equations

Abstract

The brain has developed a sophisticated hierarchical structure where information is processed across multiple layers (Van Essen et al. 1992), which is critical for forming abstraction from raw information. Inspired by this, modern machine learning is largely built on deep artificial neural networks (LeCun, Bengio, et al. 2015), where neurons are organized in layers, and synaptic weights connect neurons of different layers. One important piece of learning is to correct errors in one's predictions, and the hierarchical structure requires spreading the errors in one's predictions across multiple layers and neurons — creating the core challenge of learning known as "credit assignment" (Lillicrap, Santoro, et al. 2020). How the brain solves credit assignment is a key question to understand learning and for creating artificial learning machines, and many recent studies (Lillicrap, Santoro, et al. 2020; Richards, Lillicrap, et al. 2019; Whittington and Bogacz 2019; Zipser and Andersen 1988; Singer et al. 2018; Whittington, T. H. Muller, et al. 2020; Banino et al. 2018) approached it from the perspective of backpropagation (Werbos 1974; Rumelhart et al. 1985; Parker 1985; LeCun, Bengio, et al. 2015). However, it has been questioned whether it is possible for the brain to implement backpropagation exactly (Crick 1989; Grossberg 1987), and learning in the brain seems to be more efficient than backpropagation in several other critical aspects (Tsividis et al. 2017). On the other hand, neuroscience researchers have developed many biologically plausible models to solve the credit assignment problem (Rao and Ballard 1999; Whittington and Bogacz 2017; Whittington and Bogacz 2019; Lillicrap, Santoro, et al. 2020). However, biologically plausible models are still behind backpropagation in terms of generalization and efficiency. The thesis is inspired by this mismatch: the brain seems to be more advanced than backpropagation, while biologically plausible models learn no better than machine learning models. Thus, the overall goal of the thesis is to demonstrate biologically plausible models can inspire alternative and potentially better solutions for credit assignment than backpropagation, and we take inspiration mainly from predictive coding (PC). Specifically, I first propose a variant of PC that is equivalent to backpropagation. I then propose that the original PC has advantages over backpropagation, implementing a fundamentally different principle from backpropagation, which I call "prospective configuration". Inspired by the above two findings, I finally propose another variant of PC that is potentially more efficient than backpropagation and has a similar generalization quality as backpropagation.

Abbreviations

BP . . . . . Backpropagation.

PC . . . . . Predictive coding.

Z-PC . . . . . Zero-divergent PC.

A-Z-PC . . . . Autonomous Z-PC.

PPC . . . . . Parallel predictive coding.

MM . . . . . Matrix Multiplications.

SMM . . . . . Simultaneous Matrix Multiplications.

MLP . . . . . Multi-Layer Perceptron.

e.g. . . . . . Exempli gratia (“for the sake of an example”).

i.e. . . . . . Id est (“it is”).

w.l.o.g. . . . . Without loss of generality.

Mathematical Notations

$\circ$	Element-wise product.
l	Layer index of a network.
L	Maximum layer index of weights of a network.
$\boldsymbol{s}^{\mathbf{in}}$	Input pattern.
$\boldsymbol{s}^{\mathbf{target}}$	Target pattern.
$s_i^{\mathbf{in}}$	Value of entry i of input pattern.
$s_i^{\mathbf{target}}$	Value of entry i of target pattern.
x_i^l	Activity of neuron i at layer l.
n^l	Number of neurons at layer l.
$w_{j,i}^l$	Forward weight connecting from neuron i at layer l to neuron j at layer $l+1$.
$m_{j,i}^l$	Backward weight connecting from neuron i at layer $l+1$ to neuron j at layer l.
$\hat{x}_i^l$	Input of neuron i at layer l.
ε_i^l	Error value associated with neuron i at layer l.
$\boldsymbol{x}^l$	Vector of all x_i^l specified to layer l.
$\boldsymbol{w}^l$	Matrix of all $w_{j,i}^l$ specified to layer l.
$\boldsymbol{m}^l$	Matrix of all $m_{j,i}^l$ specified to layer l.
$\hat{\boldsymbol{x}}^l$	Vector of all $\hat{x}_i^l$ specified to layer l.
$\boldsymbol{\varepsilon}^l$	Vector of ε_i^l specified to layer l.
$\boldsymbol{S}$	A datapoint, containing $\boldsymbol{s}^{\text{in}}$ and $\boldsymbol{s}^{\text{target}}$.
$\boldsymbol{X}$	A set of $\boldsymbol{x}^l$ of all layers in a network, unless defined to exclude some layers in a particular context.
$\boldsymbol{W}$	A set of $\boldsymbol{w}^l$ of all layers in a network, unless defined to exclude some layers in a particular context.
$\boldsymbol{M}$	A set of $\boldsymbol{m}^l$ of all layers in a network, unless defined to exclude some layers in a particular context.

$\hat{\boldsymbol{X}}$ . . . . . A set of $\hat{\boldsymbol{x}}^l$ of all layers in a network, unless defined to exclude some layers in a particular context.

$f()$ . . . . . Activation function.

$f'()$ . . . . . Derivative of $f()$.

$E()$ . . . . . Energy function.

$F()$ . . . . . Loss function.

Δ . . . . . The change of.

γ . . . . . Learning rate of relaxation in energy-based models.

$\mathcal{T}$ . . . . . Total number of steps of relaxation in energy-based models.

β . . . . . Learning rate of the iteration in Almeida-Pineda.

$\mathcal{K}$ . . . . . Total number of steps of the iteration in Almeida-Pineda.

α . . . . . learning rate.

$O()$ . . . . . The time complexity of.

ϵ . . . . . A very small positive number.

In summary, I use the following notation throughout the thesis. Standard weight letters (a) denote scalar quantities. Bold lower case letters ($\boldsymbol{a}$) denote vectors or matrices, the elements of which are a of a specific layer l. Bold upper case letters ($\boldsymbol{A}$) denote a set of vectors or matrices. Each elements in $\boldsymbol{A}$ is $\boldsymbol{a}$ in different layers of a network. I use superscripts to denote the variable belongs to a specific layer l, e.g., a_i^l or $\boldsymbol{a}^l$. I use subscripts to denote specific elements of a vector or matrix, e.g., a_i^l or $a_{j,i}^l$. I use a second subscript to denote the time step of relaxation in energy-based models, e.g., $a_{i,t}^l$.

Publications

Publications included in the thesis. During my DPhil, I have published as the first author several of the results I report in this thesis, as three papers:

- One was published at NeurIPS 2020 (Song, Lukasiewicz, et al. 2020), with the title *"Can the Brain Do Backpropagation? — Exact Implementation of Backpropagation in Predictive Coding Networks."* Song, Y., Lukasiewicz, T., Xu, Z., and Bogacz, R. It corresponds to the results reported in Chapter 3.
- One is in preparation for submission to a journal at the time of writing, with the title *"Prospective Configuration: A New Perspective on Learning Beyond Backpropagation."* Song, Y., Millidge, B., Salvatori, T., Lukasiewicz, T., Xu, Z. and Bogacz, R. It corresponds to the results reported in Chapter 4.
- One is under review at AAAI 2022 at the time of writing, with the title *"Parallel Predictive Coding: A Biologically Inspired Learning Algorithm."* Song, Y., Salvatori, T., Millidge, B., Lukasiewicz, T., Xu, Z., and Bogacz, R. It corresponds to the results reported in Chapter 5.

Publications excluded in the thesis. During my DPhil, I have also published as the first author three other papers that are excluded from the thesis:

- One was published at AAAI 2020 (Song, Wojcicki, et al. 2020), with the title: *"Arena: A General Evaluation Platform and Building Toolkit for Multi-Agent Intelligence."* Song, Y., Wojcicki, A., Lukasiewicz, T., Wang, J., Aryan, A., Xu, Z., Xu, M., Ding, Z. and Wu, L.
- One was published at AAAI 2020 (Song, Wang, Lukasiewicz, Z. Xu, Zhang, et al. 2020), with the title: *"Mega-Reward: Achieving Human-Level Play without Extrinsic Rewards."* Song, Y., Wang, J., Lukasiewicz, T., Xu, Z., Zhang, S., Wojcicki, A. and Xu, M.
- One was published at AAAI 2019 (Song, Wang, Lukasiewicz, Z. Xu, and M. Xu 2019), with the title: *"Diversity-driven Extensible Hierarchical Reinforcement Learning."* Song, Y., Wang, J., Lukasiewicz, T., Xu, Z. and Xu, M.

I have also co-authored two papers that are excluded from the thesis:

- One is published at AAAI 2022 (Salvatori, Song, Lukasiewicz, et al. 2022) at the time of writing, with the title: *"Reverse Differentiation via Predictive Coding."* Salvatori, T., Song, Y., Millidge, B., Lukasiewicz, T., Xu, Z. and Bogacz, R.

- The other one was published at NeurIPS 2021 (Salvatori, Song, Y. Hong, et al. 2021), with the title: *"Associative Memories via Predictive Coding."* Salvatori, T., Song, Y., Hong, Y., Sha, L., Simon, F., Xu, Z., Bogacz, R. and Lukasiewicz, T.

1 Introduction

The brain has developed a sophisticated hierarchical structure where information is processed across multiple areas and layers (Van Essen et al. 1992). Such a hierarchical structure is critical for the brain to efficiently extract features from complex input and form abstraction from raw information. Inspired by the hierarchical structure of the brain, modern machine learning is largely built on the foundation of the deep artificial neural networks (LeCun, Bengio, et al. 2015), where neurons are organized in layers and synaptic weights connect neurons of different layers. The brain is constantly trying to make predictions about the world (Friston 2018). For example, the brain may predict one modality (e.g., sound) from another (e.g. vision) (O'reilly and Munakata 2000). Alternatively, the brain may predict the next stimulus from a sequence of previous ones (Singer et al. 2018). Thus, at least one important piece of learning is to correct errors in one's predictions, which is a result of comparing one's prediction with a supervised signal (i.e., supervised learning, or error-driven learning, since the learning is driven by the error in predictions). The aforementioned hierarchical structure of the brain and deep artificial neural networks requires spreading the errors in one's predictions across multiple areas, layers, and neurons — creating the core challenge of learning known as "credit assignment" (Lillicrap, Santoro, et al. 2020).

How the brain solves credit assignment is a key question to understand learning and for creating artificial learning machines, and many recent studies (Lillicrap, Santoro, et al. 2020; Richards, Lillicrap, et al. 2019; Whittington and Bogacz 2019; Zipser and Andersen 1988; Singer et al. 2018; Cadieu et al. 2014; Yamins, H. Hong, et al. 2014; Khaligh-Razavi and Kriegeskorte 2014; Yamins and DiCarlo 2016; Kell et al. 2018; Whittington, T. H. Muller, et al. 2020; Banino et al. 2018) approached it from the perspective of backpropagation (Werbos 1974; Rumelhart et al. 1985; Parker 1985; LeCun, Bengio, et al. 2015). Backpropagation, as a simple yet effective model, is what underpins modern machine learning. Such emphasis on backpropagation is placed by both neuroscientists and computer scientists mainly because it is yet the only model that is capable of solving complex artificial problems in various contexts such as speech, audio, visual, and action selection (Krizhevsky, Sutskever, et al. 2012; He et al. 2016; Hannun et al. 2014; Vaswani et al. 2017; Mnih et al. 2015; Silver et al. 2016). On the one hand, it has been questioned whether it is possible for the brain to implement backpropagation exactly (Crick 1989; Grossberg 1987), and learning in the brain seems to be more efficient than backpropagation in several other critical aspects – for example, the brain can learn with much fewer exposures (Tsividis et al. 2017). On the other hand, neuroscience researchers have developed many other models to solve the credit assignment problem (Rao and Ballard 1999; Whittington and Bogacz 2017; Whittington and Bogacz 2019; Lillicrap, Santoro, et al. 2020). These models are inspired by various observations from neuroscience studies, thus, are biologically plausible, i.e., they satisfy the following constraints (quoted from page 199 in (Bogacz 2017)):

1. "Local computation: a neuron performs computations only on the basis of the activity of its input neurons and synaptic weights associated with these inputs (rather than information encoded in other parts of the circuit)";

2. "Local plasticity: synaptic plasticity is only based on the activity of pre-synaptic and post-synaptic neurons".

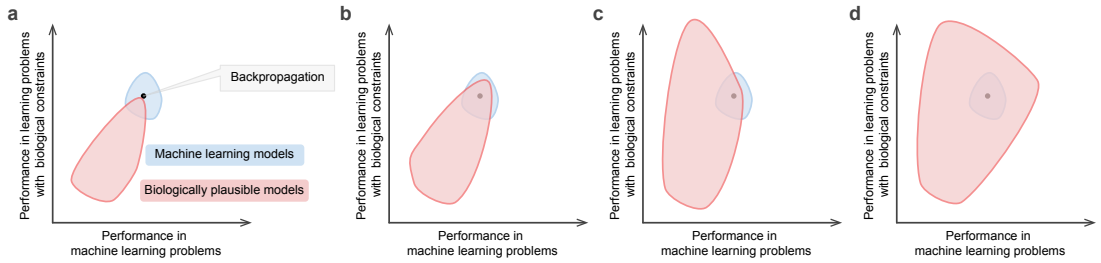


Figure 1.1: Performance in learning problems with biological constraints (y-axis) and machine learning problems (x-axis), with biologically plausible models (red area) and machine learning models (blue area).

I call such models biologically plausible models.

However, biologically plausible models are still behind backpropagation in terms of generalization and efficiency. Note that some of them have demonstrated a similar level of generalization to backpropagation (Whittington and Bogacz 2017; Song, Lukasiewicz, et al. 2020; Millidge et al. 2020a), however, they are still behind backpropagation in terms of efficiency). Thus, there is still a gap between machine learning models (centered around backpropagation) and biologically plausible models for current communities. Particularly, as shown in Figure 1.1:a, machine learning models centered around backpropagation are learning better than biologically plausible models in both machine learning problems (such as learning on a standard image classification dataset FashionMNIST (Xiao et al. 2017)) as well as learning problems with biological constraints (such as learning on the same classification dataset but with fewer examples, with smaller architectures, with the necessity of updating weights after each experience, and learning in changing environment). Here, I separate the investigation of machine learning problems and learning problems with biological constraints, because biological systems actually deal with learning situations different from the setup of machine learning problems, such as learning with fewer data and smaller architectures. (more cases are investigated in Chapter 4). Note that some work has demonstrated that biologically plausible models approximate backpropagation closely in terms of generalization (Section 2.4), i.e., the red area of biologically plausible models tangents with the black dot of backpropagation in Figure 1.1:a.

The thesis is inspired by the above mismatch: the brain seems to be more advanced than backpropagation, while biologically plausible models learn no better than machine learning models. Thus, the overall goal of the thesis is to demonstrate biologically plausible models can inspire alternative and potentially better solutions for the credit assignment than backpropagation.

Among these biologically plausible models, there is a large family called energy-based models. The thesis focuses on this family of energy-based models, since they have been used to successfully model information processing in different brain networks (Rao and Ballard 1999; Friston 2010; Hohwy et al. 2008; Auksztulewicz and Friston 2016; Bastos et al. 2012). The thesis further places much attention on one of the energy-based models: Predictive Coding Network (Rao and Ballard 1999; Whittington and Bogacz 2017; Buckley et al. 2017) (PC), because it is widely used to describe information processing in the cortex (Rao and Ballard 1999; Friston 2010; Hohwy et al. 2008; Auksztulewicz and Friston 2016). Furthermore, it is established that PC is closely related to backpropagation (Whittington and Bogacz 2017; Song, Lukasiewicz, et al. 2020; Millidge et al. 2020a), thus, providing more insights into understanding the difference with backpropagation and making a fair empirical comparison.

Specifically, the thesis first reviews the related models in Chapter 2, and then takes three steps to close the gap between machine learning models and biologically plausible models.

- Firstly, I establish a connection of equivalence between biologically plausible models (particularly, PC) and backpropagation in Chapter 3. Specifically, I propose a variant of PC that produces the same updates of the neural weights as backpropagation (i.e., biologically plausible models can be equivalent to backpropagation). This chapter expands the area of biologically plausible models to Figure 1.1:b, i.e., biologically plausible models can implement exactly backpropagation. In the above chapter, the variant of PC that is

equivalent to backpropagation applies constraints whose biological implementation is not clear yet, i.e., biologically plausible models can be equivalent to backpropagation only with unrealistic constraints.

- Secondly, I proposed that biologically plausible models without these unrealistic constraints (particularly, PC and other energy-based models) have advantages over backpropagation in Chapter 4, implementing a fundamentally different principle that I call "prospective configuration". Prospective configuration is, compared to or in contrast with backpropagation, implemented naturally in a large and well-established family of neural models with solid biological groundings, conceptually more advanced from various perspectives, empirically better in dealing with various learning problems with biological constraints, and reproducing animal learning effects that cannot be explained by backpropagation. This chapter expands the area of biologically plausible models to Figure 1.1:c, i.e., biologically plausible models are superior to backpropagation in learning problems with biological constraints.

- Thirdly, inspired by the above two chapters, I propose another variant of PC that (1) is potentially more efficient than backpropagation, as it is easier to parallelize, and that (2) has a similar generalization quality as backpropagation on several image classification tasks in Chapter 5. This chapter expands the area of biologically plausible models to Figure 1.1:d, i.e., biologically plausible models are superior to backpropagation in machine learning problems.

2 Preliminaries

Contents

This chapter reviews the models that are analyzed and extended in this thesis. In the thesis, I will be mainly comparing four different models: backpropagation, Almeida-Pineda (Almeida 1990; F. Pineda 1987; F. J. Pineda 1988), Predictive Coding Network (Rao and Ballard 1999; Whittington and Bogacz 2017; Buckley et al. 2017) (PC) and GeneRec (O'Reilly 1996). I will review backpropagation

and Almeida-Pineda in Section 2.1, because they are both backpropagation-based models: they directly modify weights to minimize the error on the output neurons, where errors are computed by comparing the activity of output neurons with the target pattern. Then, in Section 2.2, I review PC and GeneRec, which are two energy-based models: they first clamp the output neurons to the target, then change the neural activity towards a configuration that minimizes a defined energy function, and finally update the weights to further decrease the same energy function.

2.1 Backpropagation-based models

Backpropagation (Werbos 1974; Parken 1985; Rumelhart et al. 1986) is the main principle underlying learning in deep artificial neural networks (LeCun, Bengio, et al. 2015). Growing evidence demonstrates that artificial neural networks trained with backpropagation outperform alternative frameworks (Baldi and Sadowski 2016), as well as closely reproduce activity patterns observed in the cortex (Zipser and Andersen 1988; Lillicrap and Scott 2013; Cadieu et al. 2014; Yamins, H. Hong, et al. 2014; Khaligh-Razavi and Kriegeskorte 2014; Yamins and DiCarlo 2016; Kell et al. 2018; Banino et al. 2018; Whittington, T. Muller, et al. 2018), such as the response properties of neurons in the posterior parietal cortex (Zipser and Andersen 1988) and primary motor cortex (Lillicrap and Scott 2013).

I now briefly present different backpropagation-based models. For all models, I describe two stages, namely, (1) the prediction stage, when no supervision target signal is present, and the goal is to use the current parameters and input signals to make a prediction, and (2) the learning stage, when a supervision target signal is present, and the goal is to update the current parameters.

2.1.1 Backpropagation

Artificial neural networks (Rumelhart et al. 1986) are organized in layers, with multiple neuron-like value nodes in each layer as shown in Figure 2.1. Each node gets input from a previous layer weighted by the strengths of the synaptic connection and performs a nonlinear transformation of this input. The synaptic

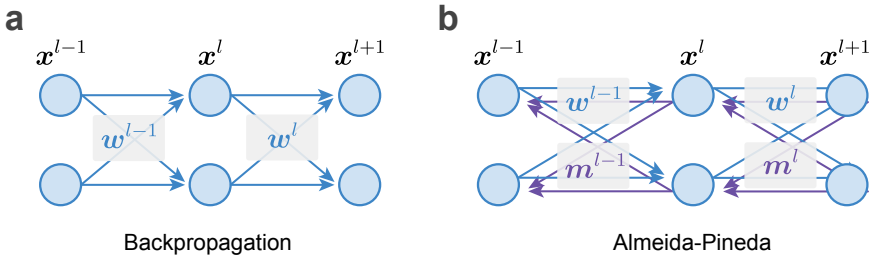


Figure 2.1: The difference between the architecture of backpropagation (i.e., feed-forward network) and that of Almeida-Pineda (i.e., recurrent network).

weights connecting from layer l to layer $l+1$ are indexed with l, thus, the synaptic weights are numbered from 1 till L and the neurons are numbered from 1 (input neurons) till $L+1$ (output neurons). I denote by x_i^l the activity of the i-th node in the l-th layer. Thus, the neural activity of each layer is computed from the weights and activity of the previous layer:

$$x_i^l = \sum_{j=1}^{n^{l-1}} w_{i,j}^{l-1} f\left(x_j^{l-1}\right) \tag{2.1}$$

where $f()$ is the activation function, $w_{i,j}^{l-1}$ is the weight from the j^{th} node in the $(l-1)^{th}$ layer to the i^{th} node in the $(l)^{th}$ layer, and n^{l-1} is the number of nodes in layer $(l-1)$. Note that, in this thesis, I consider only the case where there are only weights as parameters and no bias values. This is due to that the conclusions should generalize to the case with bias terms, thus, excluding bias terms in the thesis simplifies the presentation without loss of generality (as an example of generalizing the conclusions in this thesis to the case with bias terms, see Supplementary Material A.1 in (Song, Lukasiewicz, et al. 2020)).

Prediction Given values of the input

$$\boldsymbol{s}^{\text{in}} = \begin{bmatrix} s_1^{\text{in}} \\ \vdots \\ s_{n^1}^{\text{in}} \end{bmatrix} \tag{2.2}$$

every x_i^1 in the artificial neural network is set to the corresponding s_i^{in}, which are values of the input, and then every x_i^{L+1} is computed as the prediction via

Equation (2.1). Prediction with backpropagation is given in detail in Algorithm 1, where subscripts of indexes are removed and the notation is boldfaced to denote the vector or matrix form of scalar variables. Specifically, I define:

$$\boldsymbol{x}^l = \begin{bmatrix} x_1^l \\ \vdots \\ x_{n^l}^l \end{bmatrix} \quad \boldsymbol{w}^l = \begin{bmatrix} w_{1,1}^l & \cdots & w_{1,n^l}^l \\ \vdots & \ddots & \vdots \\ w_{n^{l+1},1}^l & \cdots & w_{n^{l+1},n^l}^l \end{bmatrix} \tag{2.3}$$

Algorithm 1: Predict with backpropagation or Predictive Coding Networks (PC)

Input: input pattern $\boldsymbol{s}^{\text{in}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$
Result: activity of output neurons $\boldsymbol{x}^{L+1}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
for $l = 1;\ l < L+1;\ l = l+1$ **do** // Forward pass of the network
  $\boldsymbol{x}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;
end

Learning Given a pair $\left(\boldsymbol{s}^{\text{in}}, \boldsymbol{s}^{\text{target}}\right)$ from the training set, where

$$\boldsymbol{s}^{\text{target}} = \begin{bmatrix} s_1^{\text{target}} \\ \vdots \\ s_{n^{L+1}}^{\text{target}} \end{bmatrix} \tag{2.4}$$

x_i^{L+1} is computed via Equation (2.1) from $\boldsymbol{s}^{\text{in}}$ as input and compared with $\boldsymbol{s}^{\text{target}}$ via the following objective function F, the parameters of which are all synaptic weights w:

$$F = \frac{1}{2} \sum_{i=1}^{n^{L+1}} \left(s_i^{\text{target}} - x_i^{L+1}\right)^2 \tag{2.5}$$

Backpropagation updates the weights of the artificial neural network by:

$$\Delta w_{i,j}^{l-1} = -\alpha \cdot \partial F / \partial w_{i,j}^{l-1} = \alpha \cdot \varepsilon_i^l f\left(x_j^{l-1}\right) \tag{2.6}$$

where α is the learning rate, and $\varepsilon_i^l = \partial E / \partial x_i^l$ is the *error term*. We can obtain the error term ε_i^l for the output layer and recursive formula of error term in other layers:

$$\varepsilon_i^l = \begin{cases} s_i^{\text{target}} - x_i^{L+1} & \text{if } l = L+1 \\ f'\left(x_i^l\right) \sum_{k=1}^{n^{l+1}} \varepsilon_k^{l+1} w_{k,i}^l & \text{if } l \in \{1, \ldots, L\} \end{cases} \tag{2.7}$$

Learning with backpropagation is given in detail in Algorithm 2, where α is the learning rate for weights update, $\circ$ denotes element-wise product. Specifically, I additional define:

$$\boldsymbol{\varepsilon}^l = \begin{bmatrix} \varepsilon_1^l \\ \vdots \\ \varepsilon_{n^l}^l \end{bmatrix} \tag{2.8}$$

Algorithm 2: Learn with backpropagation

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

Output: updated synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern

for $l = 1;\ l < L+1;\ l = l+1$ **do** // Forward pass of the network

  $\boldsymbol{x}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;

end

$\boldsymbol{\varepsilon}^{L+1} = \boldsymbol{s}^{\text{target}} - \boldsymbol{x}^{L+1}$; // Compute error of the output neurons

for $l = L+1;\ l > 2;\ l = l-1$ **do** // Backpropagation of error

  $\boldsymbol{\varepsilon}^{l-1} = f'\left(\boldsymbol{x}^{l-1}\right) \circ \left(\left(\boldsymbol{w}^{l-1}\right)^T \boldsymbol{\varepsilon}^l\right)$;

end

for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights

  $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1} \left(f\left(\boldsymbol{x}^l\right)\right)^T$;

  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;

end

2.1.2 Almeida-Pineda

As demonstrated in Figure 2.1, backpropagation works in a network where prediction is made from the input through a series of forwarding weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ (Figure 2.1:a), thus called feed-forward network; Almeida-Pineda (Almeida 1990; F. Pineda 1987; F. J. Pineda 1988) works in a network where prediction is made

from input through a mixture of forwarding weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and backward weights $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$ (Figure 2.1:b), and is thus called a recurrent network. The forward weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and backward weights $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$ are not necessarily related. A recurrent network is an interesting architecture because the connections between areas in the brain are also bi-directional (Van Essen et al. 1992).

When I distinguish feed-forward and recurrent networks, I consider the weights that are used to make a prediction: backpropagation uses only forward weights to make a prediction (Algorithm 1), while Almeida-Pineda uses both forward and backward weights to make a prediction (Algorithm 3).

Backpropagation-based learning in recurrent networks (Figure 2.1:b) is not described by standard backpropagation, but a modified version proposed by Almeida and Pineda (Almeida 1990; F. Pineda 1987; F. J. Pineda 1988) (thus called Almeida-Pineda).

Note that in some papers studying asymmetric forward and backward weights (i.e., the recurrent network), they studied fully-connected networks, where the input and output of a weight matrix $\boldsymbol{w}$ is the same set of neurons, so the asymmetric forward and backward weights are represented by $w_{i,j} \neq w_{j,i}$. Here, I study a layered network, where the input and output of a weight matrix $\boldsymbol{w}^l$ are two sets of neurons at the current layer $\boldsymbol{x}^l$ and a later layer $\boldsymbol{x}^{l+1}$. Thus, I define two sets of weights for Almeida-Pineda as well as GeneRec to be reviewed later that works in the recurrent network: $\boldsymbol{w}^l$ is the forward weights connecting from $\boldsymbol{x}^l$ to $\boldsymbol{x}^{l+1}$; $\boldsymbol{m}^l$ is the backward weights connecting from $\boldsymbol{x}^{l+1}$ to $\boldsymbol{x}^l$.

Almeida-Pineda requires an additional iterative process to propagate error, β and $\mathcal{K}$ are the learning rate and the total number of steps of this iterative process, respectively.

Prediction Prediction with Almeida-Pineda is given in Algorithm 3. The key idea is that since there are both forward $\boldsymbol{w}^l$ and backward $\boldsymbol{m}^l$ connections, neural activity $\boldsymbol{x}^l$ can no longer be calculated recursively as it is done in Equation (2.1). Instead, neural activity $\boldsymbol{x}^l$ is updated iteratively so that it satisfies the input from

Algorithm 3: Predict with Almeida-Pineda or GeneRec

Input: input pattern $\boldsymbol{s}^{\text{in}}$; forward and backward synaptic weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$
Result: activity of output neurons $\boldsymbol{x}^{L+1}$

1 $\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
2 **for** $l = 2;\ l < L+2;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$
3 $\quad \boldsymbol{x}^l = \boldsymbol{0}$;
4 **end**
5 **for** $t = 0;\ t < \mathcal{T};\ t = t+1$ **do** // Relaxation
6 $\quad$ **for** $l = 2;\ l < L+1;\ l = l+1$ **do**
7 $\quad\quad \Delta\boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;
8 $\quad\quad \boldsymbol{x}^l = \boldsymbol{x}^l + \Delta\boldsymbol{x}^l$;
9 $\quad$ **end**
10 $\quad \Delta\boldsymbol{x}^{L+1} = \gamma\left(-\boldsymbol{x}^{L+1} + \boldsymbol{w}^L f'\left(\boldsymbol{x}^L\right)\right)$;
11 $\quad \boldsymbol{x}^{L+1} = \boldsymbol{x}^{L+1} + \Delta\boldsymbol{x}^{L+1}$;
12 **end**

both forward $\boldsymbol{w}^l$ and backward $\boldsymbol{m}^l$ connections. For detailed derivations, one can refer to (Almeida 1990; F. Pineda 1987; F. J. Pineda 1988).

Learning Learning with Almeida-Pineda is given in Algorithm 4. The key idea is that since there are both forward $\boldsymbol{w}^l$ and backward $\boldsymbol{m}^l$ connections, the error terms $\boldsymbol{\varepsilon}^l$ can no longer be calculated recursively as it is done in Equation (2.7). Instead, the error terms $\boldsymbol{\varepsilon}^l$ are updated iteratively so that it satisfies the backpropagated error from both forward $\boldsymbol{w}^l$ and backward $\boldsymbol{m}^l$ connections. For detailed derivations, one can refer to (Almeida 1990; F. Pineda 1987; F. J. Pineda 1988).

Note that learning with Almeida-Pineda involves relaxation of the model, i.e., moving neuron activity, in lines 5-12 of Algorithm 4. However, this iterative process is for the purpose of making a prediction, instead of propagating the error or updating the weights, similar to the function of forwarding the model in backpropagation in lines 2-4 of Algorithm 2. The neuron activity is then fixed during the spreading of error, like in backpropagation. Thus, Almeida-Pineda is a backpropagation-based model instead of an energy-based model (which moves neural activity during the spreading of error).

Algorithm 4: Learn with Almeida-Pineda

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; forward and backward synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ and $\left\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\right\}$

Output: updated forward and backward synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ and $\left\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
for $l = 2;\ l < L+2;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$
  $\boldsymbol{x}^l = \boldsymbol{0}$;
end
for $t = 0;\ t < \mathcal{T};\ t = t+1$ **do** // Relaxation
  for $l = 2;\ l < L+1;\ l = l+1$ **do**
    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;
    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;
  end
  $\Delta \boldsymbol{x}^{L+1} = \gamma\left(-\boldsymbol{x}^{L+1} + \boldsymbol{w}^L f'\left(\boldsymbol{x}^L\right)\right)$;
  $\boldsymbol{x}^{L+1} = \boldsymbol{x}^{L+1} + \Delta \boldsymbol{x}^{L+1}$;
end
$\boldsymbol{\varepsilon}^{L+1} = \boldsymbol{s}^{\text{target}} - \boldsymbol{x}^{L+1}$; // Compute error of the output neurons
for $l = 1;\ l < L+1;\ l = l+1$ **do** // Initialize $\boldsymbol{\varepsilon}$
  $\boldsymbol{\varepsilon}^l = \boldsymbol{0}$;
end
for $t = 1;\ t < \mathcal{K}+1;\ t = t+1$ **do** // Backpropagation of error
  for $l = 2;\ l < L+1;\ l = l+1$ **do**
    $\Delta \boldsymbol{\varepsilon}^l = \beta\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\boldsymbol{m}^l \boldsymbol{\varepsilon}^{l+1}\right) + f'\left(\boldsymbol{x}^l\right) \circ \left(\boldsymbol{w}^{l-1} \boldsymbol{\varepsilon}^{l-1}\right)\right)$;
    $\boldsymbol{\varepsilon}^l = \boldsymbol{\varepsilon}^l + \Delta \boldsymbol{\varepsilon}^l$;
  end
  $\Delta \boldsymbol{\varepsilon}^1 = \beta\left(-\boldsymbol{\varepsilon}^1 + f'\left(\boldsymbol{x}^1\right) \circ \left(\boldsymbol{m}^1 \boldsymbol{\varepsilon}^2\right)\right)$;
  $\boldsymbol{\varepsilon}^1 = \boldsymbol{\varepsilon}^1 + \Delta \boldsymbol{\varepsilon}^1$;
end
for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights
  $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1}\left(f\left(\boldsymbol{x}^l\right)\right)^T$;
  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;
  $\Delta \boldsymbol{m}^l = \alpha \boldsymbol{\varepsilon}^l\left(f\left(\boldsymbol{x}^{l+1}\right)\right)^T$;
  $\boldsymbol{m}^l = \boldsymbol{m}^l + \Delta \boldsymbol{m}^l$;
end

2.1.3 Biological implausibility

Backpropagation-based models have long been criticized for their biological implausibility (i.e., their computational procedures and principles are unrealistic to be implemented in the brain) (Grossberg 1987; Crick 1989; Abdelghani et al. 2008; Lillicrap, Cownden, et al. 2016; Roelfsema and Holtmaat 2018; Whittington and Bogacz 2019). Such Biological implausibilities are reviewed in (Grossberg 1982; Rao and Ballard 1999; Huang and Rao 2011; Bengio, Lee, et al. 2015), the most crucial and most intensively studied gaps are (1) backpropagation's lack of local plasticity, i.e., weights updates in backpropagation require information that is not locally available, while in biologically plausible models weights are typically modified only on the basis of activities of two neurons (connected via synapses), and (2) backpropagation's lack of autonomy, i.e., backpropagation requires some external control over the network (e.g., to switch between prediction and learning), while biologically plausible models work autonomously.

In contrast, energy-based models are a well-established family of models with solid biological groundings, resolving the above two aspects of biological implausibility.

However, there are other biological implausibilities for backpropagation-based models that this thesis does not address. In other words, predictive coding models investigated in the thesis as well as some other energy-based models, are biologically implausible just as backpropagation-based models in these aspects.

First, backpropagation is criticized for backward connections that are symmetric to forward connections in adjacent layers. A common way to remove the backward connections (Seung 2003; Werfel et al. 2004; Lillicrap, Cownden, et al. 2016; Scellier and Bengio 2017; Lansdell et al. 2019) is based on the idea of zeroth-order optimization (Shamir 2017; Golovin et al. 2019), where weights can randomly introduce perturbation, and use unspecific feedback signals to observe their effect on the objective function and thus approximate their gradient. However, such methods need many trials depending on the number of weights (Spall 1992; Werfel et al. 2004) Such "weight perturbation" methods can be further improved by

perturbing the outputs of the neurons instead of the weights (Mazzoni et al. 1991; Flower and Jabri 1993). Admitting the existence of backward connections, more recent works show that asymmetric connections are able to produce a comparable performance (Nøkland 2016; Liao et al. 2016; Amit 2019). In the predictive coding models (PC, Z-PC, A-Z-PC, and PPC), the errors are backpropagated by correct weights, because the model includes feedback connections that also learn. The weight modification rules for corresponding feedforward and feedback weights are the same, which ensures that they remain equal if initialized to equal values (see (Whittington and Bogacz 2017)). However, predictive coding models in this thesis do not completely resolve this issue, i.e., the models in this thesis do not use random initialized asymmetric forward and backward weights.

Second, backpropagation is also criticized for using unrealistic models of (non-spiking) neurons. Backpropagation has recently been generalized to spiking neurons (Zenke and Ganguli 2018). This thesis is orthogonal to the above, i.e., we still use symmetric backward connections and do not consider spiking neurons.

Also, biological plausibility can be discussed at multiple levels. This thesis discusses it at the level of layers of neurons, in other words, we model neural activities as vectors of continuous values and weight as matrixes of continuous values, and the computation between them as simple vector-matrix multiplication. Another level of discussing biological plausibility is at the level of cells: studying how particles or potentials at different places in a cell represent these computational quantities. This thesis does not give an answer of biological plausibility at this level.

2.2 Energy-based models

Backpropagation-based models directly modify weights to minimize the error on the output neurons from the target pattern. While energy-based models, a well-established family of neuroscience models (Hopfield 1982; Friston 2005), first clamp the output neurons to the target, then change the neural activity towards a configuration that minimizes a defined energy function, and finally update the weights to further decrease the same energy function.

Energy-based models were originally proposed for associative memory tasks, which is to store and retrieve patterns, preferably in a way that allows one to recover stored patterns quickly with a low error rate. The basic idea of the energy-based models is to construct an energy function that defines an energy landscape containing basins of attraction around patterns one wants to store. Starting at any pattern, we want to have an update rule pointing towards the closest stored pattern, guided by a scalar "closeness" score provided by the energy function (Bal 2020).

Predictive Coding Network (Rao and Ballard 1999; Whittington and Bogacz 2017; Buckley et al. 2017) (PC) and GeneRec (O'Reilly 1996) are the two energy-based models I investigate. Specifically, PC is compared against backpropagation in later chapters, because it has been established that PC is closely related to backpropagation (Song, Lukasiewicz, et al. 2020) and they make the same prediction with the same weights and input pattern (Whittington and Bogacz 2017). Therefore the simulation of prediction in these two models is the same (Algorithm 1). However, they learn differently (c.f. Algorithms 2 and 5).

The other energy-based model, GeneRec (O'Reilly 1996), describes learning in recurrent networks, and backpropagation-based learning in this architecture is not described by standard backpropagation, but by Almeida-Pineda as reviewed in Section 2.1.2. In other words, GeneRec should not be compared against backpropagation since they make different predictions with the same weights and input pattern; in contrast, GeneRec should be compared against Almeida-Pineda because they make the same prediction with the same weights and input pattern (O'Reilly 1996). Therefore I simulated prediction in these two models in the same way (Algorithm 3). But they learn differently (c.f. Algorithms 4 and 6). In summary, PC and backpropagation are energy-based and backpropagation-based models working in feed-forward network architecture, respectively; GeneRec and Almeida-Pineda are energy-based and backpropagation-based models working in recurrent network architecture, respectively.

In the thesis, I place more attention on one of the energy-based models, PC, because it is widely used to describe information processing in the cortex (Rao and

Ballard 1999; Friston 2010; Hohwy et al. 2008; Auksztulewicz and Friston 2016) and only relies on local Hebbian plasticity (Rao and Ballard 1999). Furthermore, it is established that PC is closely related to backpropagation (Whittington and Bogacz 2017; Song, Lukasiewicz, et al. 2020; Millidge et al. 2020a), thus, making a fair comparison. Nevertheless, I show the findings generalize to the other energy-based model GeneRec.

2.2.1 Predictive coding networks

Predictive Coding Network (Rao and Ballard 1999; Whittington and Bogacz 2017; Buckley et al. 2017) (PC) is a widely used model of information processing in the brain, originally developed for unsupervised learning (Rao and Ballard 1999). The general principles of PC are: (i) the brain is organized into a hierarchy of areas; (ii) each area generates predictions for the activity at a lower level, (iii) prediction errors drive relaxation and learning. For more details on the general principles of PC see (Friston 2010). In (Rao and Ballard 1999), the input pattern is provided to the lower level of the hierarchy of the network (in the convention of this thesis, $l = L + 1$) and they consider unsupervised learning problems so no target pattern is presented. It has recently been shown that when "reversing the input and output ends" of PC in (Rao and Ballard 1999): providing input pattern to the higher level of hierarchy ($l = L + 1$) and target pattern to the lower level ($l = 1$), it can be used for supervised learning (Whittington and Bogacz 2017). Since I focus on supervised learning in the thesis, I will be discussing the PC for supervised learning, thus, the PC in (Whittington and Bogacz 2017).

Similar as backpropagation, in PC, the input $\hat{x}$ of the neurons at layer l is computed from the activity x of the neuron at layer $l-1$ via weights w:

$$\hat{x}_i^l = \sum_{j=1}^{n^{l-1}} w_{i,j}^{l-1} f\left(x_j^{l-1}\right) \tag{2.9}$$

For backpropagation networks, the activity x of the neuron is set to its input $\hat{x}$ (thus, Equation (2.1)). For PC (as well as other energy-based models), however,

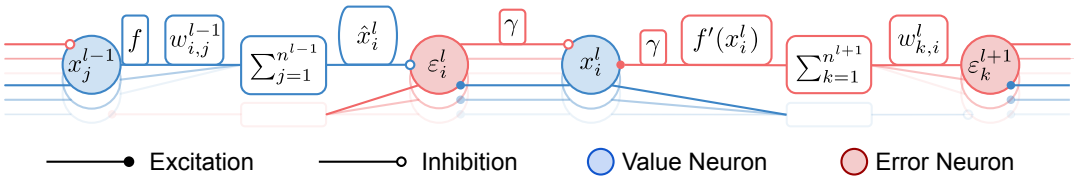


Figure 2.2: Standard neural implementation of PC.

the activity x of the neuron is not set to its input $\hat{x}$. Particularly, for PC, an error ε is defined between the activity x and the input $\hat{x}$ of the neuron:

$$\varepsilon_i^l = x_i^l - \hat{x}_i^l \tag{2.10}$$

The energy E of PC is defined to be the sum of the squares of the error ε (i.e., the energy of error ε) of all neurons at all layers except the input layer $l = 1$:

$$E = \sum_{l=2}^{L+1} \sum_{i=1}^{n^l} \frac{1}{2} \left(\varepsilon_i^l\right)^2 \tag{2.11}$$

For energy-based models, relaxation changes the activity x of a neuron to decrease the energy, i.e., in proportion to the negative of the gradient of the energy function E:

$$\Delta x_i^l = -\gamma \cdot \partial E / \partial x_i^l \tag{2.12}$$

$$= \begin{cases} 0 & \text{if } l = 1 \\ \gamma \cdot \left(-\varepsilon_i^l + f'\left(x_i^l\right) \sum_{k=1}^{n^{l+1}} \varepsilon_k^{l+1} w_{k,i}^l\right) & \text{if } l \in \{2, \ldots, L\} \\ \gamma \cdot \left(-\varepsilon_i^l\right) & \text{if } l = L+1 \text{ during prediction} \\ 0 & \text{if } l = L+1 \text{ during learning} \end{cases} \tag{2.13}$$

where γ is the learning rate of relaxation in energy-based models. Detailed derivation from Equation from (2.12) to (2.13) is given in previous work (Whittington and Bogacz 2017) as well as in the following section.

A standard neural implementation of PC is presented in Figure 2.2. Red nodes compute prediction errors ε according to Equation (2.10); blue nodes encode values x and their dynamics are described by Equation (2.13).

Prediction During prediction, only input neurons are fixed to the input signals, and the value nodes are updated according to Equation (2.13). The value nodes x_i^l will converge to their predictions $\hat{x}_i^l$, thus, prediction with PC is the same as that of backpropagation and is given in Algorithm 1.

Learning During learning both input and output neurons are fixed to the input and target signals, value nodes are updated according to Equation (2.13), after convergence, and the energy can no longer be decreased by changing neural activity, the weights w are modified to decrease energy E:

$$\Delta w_{i,j}^{l-1} = -\alpha \cdot \partial E / \partial w_{i,j}^{l-1} \tag{2.14}$$

$$= \alpha \cdot \varepsilon_i^l f\left(x_j^{l-1}\right) \tag{2.15}$$

where α is the learning rate. Detailed derivation from Equation from (2.14) to (2.15) is given in previous work (Whittington and Bogacz 2017) as well as in the following section.

Learning with PC is given in Algorithm 5, where $\mathcal{T}$ is the total number of steps of relaxation in energy-based models. Specifically, I additional define:

$$\hat{\boldsymbol{x}}^l = \begin{bmatrix} \hat{x}_1^l \\ \vdots \\ \hat{x}_{n^l}^l \end{bmatrix} \tag{2.16}$$

Derivation of networks dynamics and synaptic plasticity This section gives

- derivation of the networks dynamics of PC, i.e., derivation from Equation; (2.12) to (2.13)
- derivation of the synaptic plasticity of PC, i.e., derivation from (2.14) to (2.15).

Algorithm 5: Learn with Predictive Coding Networks (PC)

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

Output: updated synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern

$\boldsymbol{x}^{L+1} = \boldsymbol{s}^{\text{target}}$; // Clamp output neurons to target pattern

for $l = 2;\ l < L+1;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$

  $\boldsymbol{x}^l = \boldsymbol{0}$;

end

for $t = 0;\ t < \mathcal{T};\ t = t+1$ **do** // Relaxation

  for $l = 1;\ l < L+1;\ l = l+1$ **do**

    $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$; // according to Equation (2.9)

    $\boldsymbol{\varepsilon}^{l+1} = \boldsymbol{x}^{l+1} - \hat{\boldsymbol{x}}^{l+1}$; // according to Equation (2.10)

  end

  for $l = 2;\ l < L+1;\ l = l+1$ **do**

    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\left(\boldsymbol{w}^l\right)^T \boldsymbol{\varepsilon}^{l+1}\right)\right)$; // according to Equation (2.13)

    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;

  end

end

for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights

  $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1}\left(f\left(\boldsymbol{x}^l\right)\right)^T$;

  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;

end

Detailed derivations can also be found in previous works (Whittington and Bogacz 2017). Here I write the derivations in matrix and vector form. In later chapters, I will refer to some of the derivations given here. I can then write Equation (2.9), (2.10) and (2.11) in matrix/vector form as follows:

$$\hat{\boldsymbol{x}}^l = \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right) \tag{2.17}$$

$$\boldsymbol{\varepsilon}^l = \boldsymbol{x}^l - \hat{\boldsymbol{x}}^l \tag{2.18}$$

$$E = \sum_{l=2}^{L+1} \frac{1}{2}\left(\left(\boldsymbol{\varepsilon}^l\right)^T \boldsymbol{\varepsilon}^l\right) \tag{2.19}$$

Derivation from Equation (2.12) to (2.13) is as follows:

$$\Delta \boldsymbol{x}^l = -\gamma \frac{\partial E}{\partial \boldsymbol{x}^l} = -\gamma \frac{\partial \frac{1}{2}\left(\left(\boldsymbol{\varepsilon}^l\right)^T \boldsymbol{\varepsilon}^l\right) + \frac{1}{2}\left(\left(\boldsymbol{\varepsilon}^{l+1}\right)^T \boldsymbol{\varepsilon}^{l+1}\right)}{\partial \boldsymbol{x}^l} \tag{2.20}$$

$$= \gamma\left(-\left(\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)\right) + f'\left(\boldsymbol{x}^l\right) \circ \left(\left(\boldsymbol{w}^l\right)^T \left(\boldsymbol{x}^{l+1} - \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)\right)\right)\right)$$

$$= \gamma\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\left(\boldsymbol{w}^l\right)^T \boldsymbol{\varepsilon}^{l+1}\right)\right) \tag{2.21}$$

Derivation from Equation (2.14) to (2.15) is as follows:

$$\Delta \boldsymbol{w}^{l-1} = -\alpha \frac{\partial E}{\partial \boldsymbol{w}^{l-1}} = -\alpha \frac{\partial \frac{1}{2}\left(\left(\boldsymbol{\varepsilon}^l\right)^T \boldsymbol{\varepsilon}^l\right)}{\partial \boldsymbol{x}^l}$$

$$= \alpha\left(\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)\right)\left(f\left(\boldsymbol{x}^{l-1}\right)\right)^T$$

$$= \alpha \boldsymbol{\varepsilon}^l \left(f\left(\boldsymbol{x}^{l-1}\right)\right)^T \tag{2.22}$$

Limitations of PC Some features of PC are argued to be inconsistent with known properties of biological networks. But it has recently been shown that variants of PC can be developed without these implausible elements. Besides, these variants of PC still retain high classification performance. The first unrealistic feature is one-to-one connections between value and error nodes (Fig. 2.2). However, it has been proposed that errors can be represented in apical dendrites of cortical neurons (Sacramento et al. 2018), and the equations of PC can be mapped on such an architecture (Whittington and Bogacz 2019). Alternatively, it has been demonstrated how these one-to-one connections can be replaced by dense connections (Millidge et al. 2020b). The other two unrealistic features of PC are symmetric forward-backward weights and non-linear functions affecting only some outputs of the neurons. Nevertheless, it has been demonstrated that these features may be removed without significantly affecting the classification performance (Millidge et al. 2020b).

2.2.2 GeneRec

GeneRec (O'Reilly 1996) was the first biologically-plausible model that approximates a backpropagation-based model (particularly, Almeida-Pineda) in supervised learning. It is also closely related to Hopfield networks since it uses Hopfield energy. However, Hopfield networks were proposed and used for associative memory. Hopfield networks as associative memories can be traced back to (Hopfield 1982), a model also known as the classical discrete Hopfield networks. Hopfield later developed a continuous version of the classical discrete Hopfield networks (Hopfield 1984). Recently, modern discrete Hopfield networks (also known as dense associative memories) were introduced to improve pattern storage capacity and pattern separation capabilities of associative memory (Krotov and Hopfield 2016; Demircigil et al. 2017). Since machine learning applications are tailored to work with continuous embeddings (vector representations) rather than discrete patterns, most recent work focuses more on the Hopfield networks with continuous embeddings (Soto et al. 2016; Ramsauer et al. 2020).

Since the thesis investigates supervised learning, I review GeneRec, which is, as stated, an energy-based model for supervised learning using Hopfield energy. In contrast to PC, which describes learning in a feed-forward network, GeneRec describes learning in recurrent networks. Thus, it is the energy-based model that corresponds to the backpropagation-based model Almeida-Pineda, as demonstrated in (O'Reilly 1996).

Recent works (Bengio and Fischer 2015; Bengio, Mesnard, et al. 2015) study a model closely related to GeneRec, called "a kind of Hopfield energy model" in (Scellier and Bengio 2017). In particular, they investigated how part of the model resembles backpropagation when the backward weights are the transposes of the forward weights $\boldsymbol{m}^l = \left(\boldsymbol{w}^l\right)^T$ (or for a fully-connected network in their context $w_{i,j} = w_{j,i}$), and how an extreme version of the model approximate backpropagation (Scellier and Bengio 2017). Such connections are reviewed in Section 2.4.

Prediction During the prediction of GeneRec, only input neurons x_i^1 are fixed to the input pattern s_i^{in}, and neural activities are updated to minimize its energy function (i.e., relaxation). Since GeneRec works in recurrent networks, prediction with GeneRec is given in Algorithm 3, which is the same as that of Almeida-Pineda.

Learning Once the activities of the neurons converge after relaxation in prediction, GeneRec updates the weights to increase the energy (known as the "negative phase" of learning, also known as the anti-Hebbian weight modification). Then, input x_i^1 and output x_i^{L+1} neurons are clamped to input s_i^{in} and target s_i^{target} patterns, respectively. Neural activities are then updated to minimize their energy function (i.e., relaxation). Once the activities of the neurons converge again, GeneRec updates the weights to decrease the energy (known as the "positive phase" of learning, also known as the Hebbian weight modification). Learning with GeneRec is given in Algorithm 6 and explained in detail in (O'Reilly 1996).

Algorithms in (Bengio and Fischer 2015; Bengio, Mesnard, et al. 2015) are closely related to GeneRec as mentioned, since the focus of the thesis is PC and backpropagation, they are not reviewed or investigated in the thesis.

2.3 Training with mini-batches

All models are simulated in mini-batch mode for a higher efficiency (Robbins and Monro 1951; Kiefer and Wolfowitz 1952; Bottou et al. 2018). That is to say, one iteration is to update the weights for one step on a mini-batch of data randomly sampled from the training set for classification tasks. The above sampling is without replacement, i.e., the same examples will not be sampled again before the completion of an epoch, which is when the entire training set has been sampled once. For example, considering a dataset of 1000 examples having a batch-size (number of examples in a mini-batch) of 10 and each iteration would update weights for one step on 10 examples, it will take 100 such iterations to complete one epoch. If the dataset contains only 1 example or the number of examples in a dataset is the same as the batch-size, iteration and epoch are the same. To implement the Algorithms 1 to 6

Algorithm 6: Learn with GeneRec

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; forward and backward synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ and $\left\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\right\}$

Output: updated forward and backward synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ and $\left\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
for $l = 2;\ l < L+2;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$
  $\boldsymbol{x}^l = \boldsymbol{0}$;
end
for $t = 0;\ t < \mathcal{T};\ t = t+1$ **do** // Relaxation
  for $l = 2;\ l < L+1;\ l = l+1$ **do**
    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;
    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;
  end
  $\Delta \boldsymbol{x}^{L+1} = \gamma\left(-\boldsymbol{x}^{L+1} + \boldsymbol{w}^L f'\left(\boldsymbol{x}^L\right)\right)$;
  $\boldsymbol{x}^{L+1} = \boldsymbol{x}^{L+1} + \Delta \boldsymbol{x}^{L+1}$;
end
for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights (negative phase)
  $\Delta \boldsymbol{w}^l = -\alpha f\left(\boldsymbol{x}^{l+1}\right)\left(f\left(\boldsymbol{x}^l\right)\right)^T$;
  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;
  $\Delta \boldsymbol{m}^l = -\alpha f\left(\boldsymbol{x}^l\right)\left(f\left(\boldsymbol{x}^{l+1}\right)\right)^T$;
  $\boldsymbol{m}^l = \boldsymbol{m}^l + \Delta \boldsymbol{m}^l$;
end
$\boldsymbol{x}^{L+1} = \boldsymbol{s}^{\text{target}}$; // Clamp output neurons to target pattern
for $t = 0;\ t < \mathcal{T};\ t = t+1$ **do** // Relaxation
  for $l = 2;\ l < L+1;\ l = l+1$ **do**
    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;
    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;
  end
end
for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights (positive phase)
  $\Delta \boldsymbol{w}^l = \alpha f\left(\boldsymbol{x}^{l+1}\right)\left(f\left(\boldsymbol{x}^l\right)\right)^T$;
  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;
  $\Delta \boldsymbol{m}^l = \alpha f\left(\boldsymbol{x}^l\right)\left(f\left(\boldsymbol{x}^{l+1}\right)\right)^T$;
  $\boldsymbol{m}^l = \boldsymbol{m}^l + \Delta \boldsymbol{m}^l$;
end

described below in mini-batch mode, one can simply add an extra dimension, the size of which is batch-size, to all the neuron-specific vectors in the algorithms such as $\boldsymbol{x}^l$, $\hat{\boldsymbol{x}}^l$, and $\boldsymbol{\varepsilon}^l$, then reduce this dimension by summing over it when computing weight update $\Delta \boldsymbol{w}^l$ (and $\Delta \boldsymbol{m}^l$ if the models are GeneRec or Almeida-Pineda).

2.4 Connection between backpropagation-based and energy-based models

Recent works (Bengio and Fischer 2015; Bengio, Mesnard, et al. 2015) show how part of the GeneRec model resembles to backpropagation when the backward weights are the transposes of the forward weights $\boldsymbol{m}^l = \left(\boldsymbol{w}^l\right)^T$ (or for a fully-connected network in their context $w_{i,j} = w_{j,i}$), and how the extreme version of GeneRec approximate backpropagation (Bengio and Fischer 2015; Scellier and Bengio 2017). Similar connections have also been made for PC that PC approximates backpropagation in its extreme versions (Whittington and Bogacz 2017; Song, Lukasiewicz, et al. 2020; Millidge et al. 2020a). This section reviews the literature building connections between energy-based and backpropagation-based models.

They can be summarized as that the energy-based models approximate backpropagation in the following two conditions:

1. Insufficient relaxation: update of the weights is conducted at the very start of relaxation instead of at relaxation convergence. Such condition is demonstrated for both GeneRec (Bengio and Fischer 2015) and PC (Chapter 3);

2. Weakly supervised signal: the supervised signal is infinitely weak, i.e., the provided target pattern is infinitely close to the prediction of the model already (Whittington and Bogacz 2017; Millidge et al. 2020a; Scellier and Bengio 2017). This condition is demonstrated for both GeneRec and PC. Such condition is also generalized to a general energy function (Scellier and Bengio 2017).

I review these works in detail in the following.

2.4.1 Insufficient relaxation

In GeneRec

A recent paper (Bengio and Fischer 2015) shows that the early steps of relaxation (called inference in the paper) in GeneRec, starting from the equilibrium point (called stationary point in the paper), approximate the propagation of gradient, i.e., approximate backpropagation. The two key conditions need to be satisfied for the approximation to hold:

- Starting from the equilibrium point
- The early steps of relaxation

Starting from the equilibrium point means that the hidden neurons are at their equilibrium, i.e., neural activity has been relaxed to minimize the energy function. Then, once the output neurons are clamped to the target pattern, by relaxing the neural activity, the neural activity will diverge from its equilibrium point. Most importantly, a hidden neuron will not diverge from its equilibrium point immediately after the output neuron is clamped to the target pattern, instead, neurons at layers closer to the output neurons diverge from their equilibrium points first. Thus, neurons will diverge from their equilibrium points starting from the time depending on how far away they are from the output neurons: a behavior resembles backpropagation where gradient backpropagates from the output layer into the network layer by layer. The key finding in (Bengio and Fischer 2015) is that the first time a hidden neuron diverges from its equilibrium point, i.e., being perturbed, the direction of such perturbation approximates the backpropagation gradient.

In PC

In this thesis, Chapter 3 presents that such a general idea applies to PC as well (resulting in a variant of PC called Z-PC): the very first time the perturbation caused by clamping output neurons to the target pattern reaches a hidden neuron, the hidden neuron moves towards the direction of backpropagation gradient.

Importantly, Z-PC presented in Chapter 3 is mathematically strictly equivalent to backpropagation instead of being an approximation.

2.4.2 Weakly supervised signal

A more widely studied perspective of links between backpropagation-based and energy-based models is when the supervised signal is infinitely weak.

In PC

This section reviews the literature demonstrating that PC approximates backpropagation when the supervised signal is infinitely weak.

To reveal such an approximation, I first demonstrate the relationship between ε_i^l of different layers l, which will show that ε_i^l at the equilibrium of relaxation during learning can be expressed in a recursive formula like the recursive formula of backpropagation (Equation (2.7)) I can solve the fixed point of the dynamic of the neural activity of PC, i.e., setting the left side of Equation (2.13) and solve for ε_i^l and ε_k^{l+1}. The solution of ε_i^l and ε_k^{l+1} are denoted by $\varepsilon_{i,\mathcal{T}}^l$ and $\varepsilon_{k,\mathcal{T}}^{l+1}$, respectively, because they are the equilibrium at relaxation step $t = \mathcal{T}$. Similarly, subscript $\mathcal{T}$ has been added to other notations as they are the variables at equilibrium at relaxation step $t = \mathcal{T}$:

$$\Delta x_{i,\mathcal{T}}^l = 0 \implies \gamma \cdot \left(-\varepsilon_{i,\mathcal{T}}^l + f'\left(x_{i,\mathcal{T}}^l\right) \sum_{k=1}^{n^{l+1}} \varepsilon_{k,\mathcal{T}}^{l+1} w_{k,i}^l \right) = 0 \tag{2.23}$$

$$\implies -\varepsilon_{i,\mathcal{T}}^l + f'\left(x_{i,\mathcal{T}}^l\right) \sum_{k=1}^{n^{l+1}} \varepsilon_{k,\mathcal{T}}^{l+1} w_{k,i}^l = 0 \tag{2.24}$$

$$\implies \varepsilon_{i,\mathcal{T}}^l = f'\left(x_{i,\mathcal{T}}^l\right) \sum_{k=1}^{n^{l+1}} \varepsilon_{k,\mathcal{T}}^{l+1} w_{k,i}^l \tag{2.25}$$

Obviously, if PC makes a perfect prediction, i.e., error $\varepsilon_{i,\mathcal{T}}^l$ is zero at every layer: $x_{i,\mathcal{T}}^l = \hat{x}_{i,\mathcal{T}}^l$, the above equation is the same as the recursive formula of error in backpropagation: Equation (2.7).

The above findings motivate the results in (Whittington and Bogacz 2017). Specifically, they stated three closely related conditions under which PC approximates backpropagation:

- First, when a PC is trained such that it correctly predicts the output training samples, then the equation of updating weights for PC is the same as that of BP. In particular, the change in weights is equal to 0; thus, the weights resulting in perfect prediction are a shared fixed point for PC and BP.
- Second, if the prediction $x_{i,\mathcal{T}}^{L+1}$ is infinitely close to the target pattern s_i^{target}, error $\varepsilon_{i,\mathcal{T}}^{l}$ is infinitely small: $x_{i,\mathcal{T}}^{l}$ approximates $\hat{x}_{i,\mathcal{T}}^{l}$, and the error term in PC at equilibrium during training, i.e., Equation (2.25) approximates the recursive formula of error in backpropagation, i.e., Equation (2.7).
- Thirdly, in (Whittington and Bogacz 2017) there are additional parameters Σ_i^l (whose inverses determine the influence of corresponding error terms ε_i^l on network dynamics). When the value of parameters Σ_i^{L+1} is increased relative to other Σ_i^l, the impact of fixing x_i^{L+1} on the activity of other nodes is reduced, because ε_i^{L+1} becomes smaller and its influence on the activity of other nodes is reduced.

Recent work (Millidge et al. 2020a) further scales the above result of equivalence to any computational graphs, motivated by the fact that the power of backpropagation lies not in its instantiation in MLP (Multi-Layer Perceptron), but rather in the automatic differentiation which allows for the optimization of any differentiable program expressed as a computation graph (thus, complex and powerful architectures such as CNN, RNN, and LSTMs). They demonstrate that PC converges asymptotically (and in practice rapidly) to backpropagation on arbitrary computation graphs. However, the scope of this thesis is limited to MLP, and provides insights into how PC connects to backpropagation in an alternative way (Chapter 3); how PC is fundamentally different from and more "clever" than backpropagation (Chapter 4); and how PC inspires a potentially more efficient alternative to backpropagation (Chapter 5).

A general form demonstrated with GeneRec

The above works on showing PC approximate backpropagation with a weakly supervised signal are unified in the equilibrium propagation theory (Scellier and Bengio 2017). The equilibrium propagation theory (Scellier and Bengio 2017) uses an example of a model (Bengio and Fischer 2015; Bengio, Mesnard, et al. 2015) minimizing the Hopfield energy (thus, similar to GeneRec), and proposes a theory on how any energy-based model can approximate backpropagation with infinitely weakly supervised signal. Specifically, they first define a coefficient β of the energy term of the output layer, which nicely quantifies how strong the supervised signal is: $\beta = 1$ means the supervised signal is of the standard strength, and $\beta \to 0$ means the supervised signal is of infinitely small strength. Note that in (Scellier and Bengio 2017), they define β as the coefficient of the objective function defined on the output neurons. It is equivalent to what I defined here because in energy-based models the output neurons are clamped to the target pattern during learning thus the energy term of the output layer is actually the objective function defined on the output neurons. The original definition is more general though because it allows the output layer to define a different objective function from the energy function of other layers, for example, I can define a PC with the energy function of the output layer equal to the cross-entropy loss to the target pattern. In this thesis, however, I do not investigate these generalizations, thus, only considering the output neurons clamped to the target pattern, thus, the objective function defined on the output layer is the same as the energy function defined on all other layers.

To formally state the equilibrium propagation theory (Scellier and Bengio 2017), I can define any energy-based model with energy function E, and it is a function of β thus E_β, where β is the coefficient of the energy term of the output layer. Also, I consider the energy function at the equilibrium of relaxation, thus, following the convention I used just now, I add a subscript of $\mathcal{T}$ to denote a variable at the equilibrium of relaxation. Thus, any energy function is a function of $\mathcal{T}$ and β: $E_{\mathcal{T},\beta}$. The equilibrium propagation theory states that for any energy-based

models with energy function $E_{\mathcal{T},\beta}$, it approximates backpropagation as long as the update of weights satisfies:

$$\Delta w_{i,j}^{l-1} = \alpha \cdot \lim_{\beta \to 0} \frac{1}{\beta} \left(\frac{\partial E_{\mathcal{T},0}}{\partial w_{i,j}^{l-1}} - \frac{\partial E_{\mathcal{T},\beta}}{\partial w_{i,j}^{l-1}} \right) \tag{2.26}$$

GeneRec is demonstrated by (Scellier and Bengio 2017) as an example of the above equilibrium propagation theory (Equation (2.26)). Specifically, $\frac{\partial E_{\mathcal{T},0}}{\partial w_{i,j}^{l-1}}$ and $\frac{\partial E_{\mathcal{T},\beta}}{\partial w_{i,j}^{l-1}}$ in Equation (2.26) correspond to the negative and positive phases of GeneRec in Algorithm 6, respectively.

The conditions of PC approximating backpropagation stated in (Whittington and Bogacz 2017) reviewed in Section 2.4.2 are examples of the above equilibrium propagation theory (Equation (2.26)), as demonstrated in (Whittington and Bogacz 2019). For PC, if the strength of the supervised signal is zero $\beta = 0$, the error terms ε_i^l converge to zero, thus, $\frac{\partial E_{\mathcal{T},0}}{\partial w_{i,j}^{l-1}} = 0$ according to Equations (2.14) and (2.15). Thus Equation (2.26) for PC becomes:

$$\Delta w_{i,j}^{l-1} = \alpha \cdot \lim_{\beta \to 0} \frac{1}{\beta} \left(-\frac{\partial E_{\mathcal{T},\beta}}{\partial w_{i,j}^{l-1}} \right) \tag{2.27}$$

which actually states the condition in Section 2.4.2 in a formal way: $\beta \to 0$ formally states the approximation of PC to backpropagation is when the supervised signal is infinitely weak; the additional term $\frac{1}{\beta}$ is to normalize the magnitude of the updates, since, as one can imagine, the infinitely weakly supervised signal means infinitely small weight updates, this $\frac{1}{\beta}$ will normalize the updates of weights to the magnitude when setting $\beta = 1$. The third condition in Section 2.4.2 is even more consistent with the equilibrium propagation theory: β in the equilibrium propagation theory is exactly $\frac{1}{2}\left(\frac{1}{\Sigma_i^{L+1}}\right)^2$ in the third condition in Section 2.4.2 (assuming Σ_i^{L+1} for different i are the same).

Studying the above extreme conditions (insufficient relaxation and weakly supervised signal) provides insights into how energy-based models are related to backpropagation, but they are not how the energy-based models work in their natural

form and they impose biologically unrealistic constraints on energy-based models. For example, updating weights at insufficient relaxation requires an additional control mechanism; assuming the presence of an infinitely weakly supervised signal is obviously unrealistic. As I will be demonstrating later in Chapter 4, energy-based models in their natural form correspond to a fundamentally different learning principle than backpropagation, and such insights inspire potentially more efficient alternatives to backpropagation (Chapter 5).

3

Predictive coding can be equivalent to backpropagation

Contents

This chapter discusses the results that biologically plausible models, particularly, Predictive Coding Network (Rao and Ballard 1999; Whittington and Bogacz 2017; Buckley et al. 2017) (PC), can be equivalent to backpropagation under three conditions. The resulting model is a variant of PC that produces exactly the same updates of the neural weights as backpropagation. Thus, I call such model *zero-divergent PC (Z-PC)*. This chapter is published in NeurIPS-2020 (Song,

Lukasiewicz, et al. 2020).

Since this chapter will be investigating the relationship between PC and backpropagation, it is necessary to use different notations for neural activation x, weight w, and error ε between PC and backpropagation, instead of unified notations as Chapter 2 and the other chapters. Specifically, the notations of these three variables are changed to y, θ, and δ for backpropagations in this chapter, while the corresponding notations for PC remain unchanged as x, w and ε. Thus, for backpropagation, Equations (2.1), (2.7), and (2.6) in this chapter becomes:

$$y_i^l = \sum_{j=1}^{n^{l-1}} \theta_{i,j}^{l-1} f\left(y_j^{l-1}\right) \tag{3.1}$$

$$F = \frac{1}{2} \sum_{i=1}^{n^{L+1}} \left(s_i^{\text{target}} - y_i^{L+1}\right)^2 \tag{3.2}$$

$$\Delta\theta_{i,j}^{l-1} = -\alpha \cdot \partial F / \partial \theta_{i,j}^{l-1} = \alpha \cdot \delta_i^l f\left(y_j^{l-1}\right) \tag{3.3}$$

$$\delta_i^l = \begin{cases} s_i^{\text{target}} - y_i^{L+1} & \text{if } l = L+1 \\ f'\left(y_i^l\right) \sum_{k=1}^{n^{l+1}} \delta_k^{l+1} w_{k,i}^l & \text{if } l \in \{1, \ldots, L\} \end{cases} z \tag{3.4}$$

3.1 Temporal training dynamics

We should first notice that training PC requires the neural activation to be updated first (lines 6-15 in Algorithm 5) to minimize an energy function. Weights are then updated to further minimize the same energy function (lines 16-19 in Algorithm 5). Such updating of neural activation is known as relaxation of energy-based models, I here first investigate the temporal training dynamics of different models, which expand each time step of relaxation.

Thus, I denote by t the time axis during relaxation. Thus, notations of PC is augmented with an additional subscript of t if it is changed during relaxation:

$$\hat{x}^l_{i,t} = \sum_{j=1}^{n^{l-1}} w^{l-1}_{i,j} f\left(x^{l-1}_{j,t}\right) \quad \text{and} \quad \varepsilon^l_{i,t} = x^l_{i,t} - \hat{x}^l_{i,t} \tag{3.5}$$

$$E_t = \sum_{l=2}^{L+1} \sum_{i=1}^{n^l} \frac{1}{2} \left(\varepsilon^l_{i,t}\right)^2 \tag{3.6}$$

$$\Delta x^l_{i,t} = \begin{cases} 0 & \text{if } l = 1 \\ \gamma \cdot \left(-\varepsilon^l_{i,t} + f'\left(x^l_{i,t}\right) \sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,t} w^l_{k,i}\right) & \text{if } l \in \{2, \ldots, L\} \\ \gamma \cdot \left(-\varepsilon^l_{i,t}\right) & \text{if } l = L+1 \text{ during prediction} \\ 0 & \text{if } l = L+1 \text{ during learning} \end{cases} \tag{3.7}$$

$$x^l_{i,t+1} = x^l_{i,t} + \Delta x^l_{i,t} \tag{3.8}$$

$$\varepsilon^{L+1}_{i,t} = x^{L+1}_{i,t} - \hat{x}^{L+1}_{i,t} = s^{\text{target}}_i - \hat{x}^{L+1}_{i,t} \tag{3.9}$$

$$\Delta w^{l-1}_{i,j} = -\alpha \cdot \partial E_t / \partial w^{l-1}_{i,j} = \alpha \cdot \varepsilon^l_{i,t} f\left(x^{l-1}_{j,t}\right) \tag{3.10}$$

Notations in the above are with a subscript t to indicate changes during the relaxation, where such t denotes the time steps during relaxation. If a variable does not change during relaxation, e.g., $w^{l+1}_{i,j}$, it is denoted without subscript t.

I now first describe the temporal training dynamics of backpropagation and PC. Based on this, I then propose Z-PC in Section 3.2, which is a variant of PC with zero divergences from backpropagation.

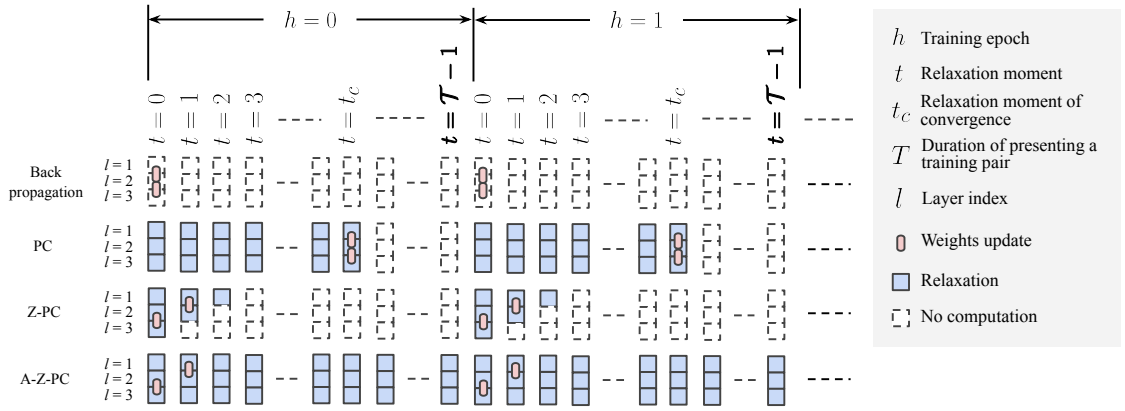


Figure 3.1: Comparison of the temporal training dynamics of backpropagation, PC, Z-PC and A-Z-PC.

I assume that I train the networks on a pair $(\boldsymbol{s}^{\text{in}}, \boldsymbol{s}^{\text{target}})$ from the dataset, which is presented for a period $\mathcal{T}$, before it is changed to another pair, moving to the next training epoch h. Within a single training epoch h, $(\boldsymbol{s}^{\text{in}}, \boldsymbol{s}^{\text{target}})$ stays unchanged, and t runs from 0 to $\mathcal{T}$. As stated before, t is the time axis during relaxation, which means that PC (also Z-PC and A-Z-PC, proposed below) in PCs run relaxation starting from $t = 0$. In Figure 3.1, squares and rounded rectangles represent nodes in one layer and connection weights between nodes in two layers of a neural network, respectively: backpropagation (first row) only conducts weight updates within all layers for one step in one training epoch, and no other computations within this training epoch; while PC (second row) conducts relaxation until it converges ($t = t_c$) and updates weights in all layers for one step within one training epoch (assuming the duration a training pair is presented is much longer than the time needed for neurons to converge $T \geq t_c$).

In the following, I introduce *zero-divergent PC (Z-PC)* and *autonomous zero-divergent PC (A-Z-PC)*, respectively: Z-PC (third row) also conducts relaxation but until specific relaxation moments $t = L - l + 1$ and updates weights between the layers l and $l + 1$, while A-Z-PC (fourth row) conducts relaxation all the time and weights update is trigged autonomously at the same relaxation moments as Z-PC.

3.2 Zero-divergent PC

I now present three conditions C3.2 to C3.2 with which a variant of PC is proposed, denoted *zero-divergent PC* (*Z-PC*), produces exactly the same weights as backpropagation (having the same initial weights) applied to the same datapoints $\boldsymbol{S} = \left(\boldsymbol{s}^{\text{in}}, \boldsymbol{s}^{\text{target}}\right)$. In the following, I first describe the three conditions C3.2 to C3.2, and then formally state and prove that Z-PC produces exactly the same weights as backpropagation.

C1 Every $x_{i,t}^{l}$ and every $\hat{x}_{i,t}^{l}$, $l \in \{1, \ldots, L\}$, at $t = 0$ is equal to y_i^l in the corresponding backpropagation with input $\boldsymbol{s}^{\text{in}}$. In particular, this also implies that $\varepsilon_{i,t}^{l} = 0$ at $t = 0$, for $l \in \{1, \ldots, L+1\}$. This condition is naturally satisfied in PC, if before the start of each training epoch over a training pair $\left(\boldsymbol{s}^{\text{in}}, \boldsymbol{s}^{\text{target}}\right)$, the input $\boldsymbol{s}^{\text{in}}$ has been presented, and the network has converged in the prediction stage (see Section 2.2.1). This condition corresponds to a requirement in backpropagation, that it needs one forward pass from $\boldsymbol{s}^{\text{in}}$ to compute the prediction before conducting weight updates with the supervision signal $\boldsymbol{s}^{\text{target}}$. Note that neither the forward pass for backpropagation nor this initialization for PC are shown in Figure 3.1.

C2 Every weight $w_{i,j}^{l}$, $l \in \{1, \ldots, L\}$, is updated at $t = L - l + 1$, that is, at a very specific relaxation moment, related to the layer that the weight belongs to. This may seem quite strict, but it can actually be implemented with autonomy (see Section 3.3).

C3 The integration step of relaxation γ is set to 1. Note that solely relaxing this condition (keeping C3.2 and C3.2 satisfied) results in backpropagation with a different learning rate for different layers, where γ is the decay factor of this learning rate along layers (see Section 3.2.2).

To prove the above equivalence statement under C3.2 to C3.2, I develop two theorems in order (proved in Section 3.2.1). The following first theorem formally

states that the prediction error in the PC with PC on $\boldsymbol{S}$ under C3.2 to C3.2 is equal to the error term in its ANN with backpropagation on $\boldsymbol{S}$.

Theorem 3.2.1. *Let $\boldsymbol{M}$ be a network to be trained with PC, $\boldsymbol{M'}$ be another network to be trained with backpropagation (with the same initial weights as $\boldsymbol{M}$), and let $\boldsymbol{S}$ be a datapoint. Then, every prediction error $\varepsilon^l_{i,t}$ at $t = L - l + 1$, $l \in \{1, \ldots, L+1\}$, in $\boldsymbol{M}$ trained with PC on $\boldsymbol{S}$ under C3.2 and C3.2 is equal to the error term δ^l_i in $\boldsymbol{M'}$ trained with backpropagation on $\boldsymbol{S}$.*

I next formally state that every weights update in the PC on $\boldsymbol{S}$ under C3.2 to C3.2 is equal to the weights update in the corresponding ANN trained with backpropagation on $\boldsymbol{S}$. This then immediately implies that the final weights of the PC on $\boldsymbol{S}$ under C3.2 to C3.2 are equal to the final weights of backpropagation on $\boldsymbol{S}$.

Theorem 3.2.2. *Let $\boldsymbol{M}$ be a network to be trained with PC, $\boldsymbol{M'}$ be another network to be trained with backpropagation (with the same initial weights as $\boldsymbol{M}$), and let $\boldsymbol{S}$ be a datapoint. Then, every update $\Delta w^l_{i,j}$ at $t = L - l + 1$, $l \in \{1, \ldots, L\}$, in $\boldsymbol{M}$ trained with PC on $\boldsymbol{S}$ under C3.2 and C3.2 is equal to the update $\Delta\theta^l_{i,j}$ in $\boldsymbol{M'}$ trained with backpropagation on $\boldsymbol{S}$.*

3.2.1 Proof of Theorems 3.2.1 and 3.2.2

Theorem 3.2.1. *Let $\boldsymbol{M}$ be a network to be trained with PC, $\boldsymbol{M'}$ be another network to be trained with backpropagation (with the same initial weights as $\boldsymbol{M}$), and let $\boldsymbol{S}$ be a datapoint. Then, every prediction error $\varepsilon^l_{i,t}$ at $t = L - l + 1$, $l \in \{1, \ldots, L+1\}$, in $\boldsymbol{M}$ trained with PC on $\boldsymbol{S}$ under C3.2 and C3.2 is equal to the error term δ^l_i in $\boldsymbol{M'}$ trained with backpropagation on $\boldsymbol{S}$.*

Proof. We should first notice that $\varepsilon^l_{i,t}$ under C3.2 is $\varepsilon^l_{i,L-l+1}$. I give a proof by induction on l. Since $t = L - l + 1$, it is also inducing on the relaxation moments.

◊ *Base Case*: If $l = L + 1$,

- putting C3.2 $\hat{x}^l_{i,0} = y^l_i$ into Equation (3.9), and by comparison with the first case in Equation (3.4): $\varepsilon^l_{i,L-l+1} = \delta^l_i$

◊ *Induction Step(s)*: For $l \in \{L, \dots, 1\}$,

- $\varepsilon^l_{i,L-l+1} = f'\left(\hat{x}^l_{i,0}\right) \sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,L-l} w^l_{k,i}$, by Lemma 3.2.4
- $\delta^l_i = f'\left(y^l_i\right) \sum_{k=1}^{n^{l+1}} \delta^{l+1}_k \theta^l_{k,i}$, by the second case in Equation (3.4)
- $\hat{x}^l_{i,0} = y^l_i$, by C3.2
- $\theta^l_{i,j} = w^l_{i,j}$, as corresponding initial weights in both models are assumed to be identical
- $\varepsilon^l_{i,L-l+1} = \delta^l_i$, if $\varepsilon^{l+1}_{k,L-l} = \delta^{l+1}_k$

□

Theorem 3.2.2. *Let $\boldsymbol{M}$ be a network to be trained with PC, $\boldsymbol{M'}$ be another network to be trained with backpropagation (with the same initial weights as $\boldsymbol{M}$), and let $\boldsymbol{S}$ be a datapoint. Then, every update $\Delta w^l_{i,j}$ at $t = L - l + 1$, $l \in \{1, \dots, L\}$, in $\boldsymbol{M}$ trained with PC on $\boldsymbol{S}$ under C3.2 and C3.2 is equal to the update $\Delta\theta^l_{i,j}$ in $\boldsymbol{M'}$ trained with backpropagation on $\boldsymbol{S}$.*

Proof. Looking at how $\partial E_t / \partial w^{l-1}_{i,j}$ and $\partial F / \partial \theta^{l-1}_{i,j}$ are computed via Equations (3.10) and (3.3), and putting C3.2 $t = L - l + 1$ into the equation,

$$\Delta w^{l-1}_{i,j} = \alpha \cdot \varepsilon^l_{i,L-l+1} f\left(x^{l-1}_{j,L-l+1}\right) \tag{3.11}$$

$$\Delta \theta^{l-1}_{i,j} = \alpha \cdot \delta^l_i f\left(y^{l-1}_j\right) \tag{3.12}$$

We should notice that one of the terms in both equations are equivalent according to Theorem 3.2.1: $\varepsilon^l_{i,L-l+1} = \delta^l_i$. Thus, in the following, I focus on proving that the other terms in both equations are identical: $f\left(x^{l-1}_{j,L-l+1}\right) = f\left(y^{l-1}_j\right)$. First, C3.2 provides the base to start, which is the equivalence of the initial state between PC and backpropagation under C3.2, C3.2, and C3.2: $x^{l-1}_{j,0} = \hat{x}^{l-1}_{j,0} = y^{l-1}_j$. Then, Lemma 3.2.3 links later relaxation moments of PC under C3.2, C3.2, and C3.2 to its initial state: $x^{l-1}_{j,L-l+1} = x^{l-1}_{j,0}$. Thus, we can have $f\left(x^{l-1}_{j,L-l+1}\right) = f\left(y^{l-1}_j\right)$. □

Lemma 3.2.3. *Under C3.2, so that a variable at a specific layer may diverge from its corresponding initial stable states, it needs specific relaxation steps related to the layer that the variable belongs to. Formally,*

$$\boldsymbol{x}^l_{t<L-l+1} = \boldsymbol{x}^l_0, \boldsymbol{\varepsilon}^l_{t<L-l+1} = \boldsymbol{\varepsilon}^l_0 = 0, \hat{\boldsymbol{x}}^{l+1}_{t<L-l+1} = \hat{\boldsymbol{x}}^{l+1}_0 \text{ for } l \in \{L, \dots, 1\}$$
$$\text{i.e., } \Delta\boldsymbol{x}^l_{t<L-l} = \boldsymbol{0}, \Delta\boldsymbol{\varepsilon}^l_{t<L-l} = \boldsymbol{0}, \Delta\hat{\boldsymbol{x}}^{l+1}_{t<L-l} = \boldsymbol{0} \text{ for } l \in \{L, \dots, 1\}$$

Proof. Starting from the relaxation moment $t = 0$, $\boldsymbol{x}^{L+1}_0$ is dragged away from $\hat{\boldsymbol{x}}^{L+1}_0$ and fixed to $\boldsymbol{s}^{\text{target}}$, i.e., $\boldsymbol{\varepsilon}^{L+1}_0$ turns into nonzero from zero. Since $\boldsymbol{x}$ in each layer is updated only on the basis of $\boldsymbol{\varepsilon}$ in the same and previous adjacent layer, as indicated by Equation (3.7), also considering C3.2, $\boldsymbol{\varepsilon}$ is initially zero for all layers but the output layer, it will take l time steps to modify $\boldsymbol{x}^l_t$ at layer l from the initial state. Hence, $\boldsymbol{x}^l_t$ will remain in that initial state $\boldsymbol{x}^l_0$ for all $t < L - l + 1$, i.e., $\boldsymbol{x}^l_{t<L-l+1} = \boldsymbol{x}^l_0$. Furthermore, any change in $\boldsymbol{x}^l_t$ causes a change in $\boldsymbol{\varepsilon}^l_t$ and $\hat{\boldsymbol{x}}^{l+1}_t$ instantly via Equation (3.5) (otherwise $\boldsymbol{\varepsilon}^l_t$ and $\hat{\boldsymbol{x}}^{l+1}_t$ remain in their corresponding initial states). Thus, we should know $\boldsymbol{\varepsilon}^l_{t<L-l+1} = \boldsymbol{\varepsilon}^l_0$ and $\hat{\boldsymbol{x}}^{l+1}_{t<L-l+1} = \hat{\boldsymbol{x}}^{l+1}_0$. Also, according to C3.2, $\boldsymbol{\varepsilon}^l_{t<L-l+1} = \boldsymbol{\varepsilon}^l_0 = 0$. Equivalently, we can have $\Delta\boldsymbol{x}^l_{t<L-l} = \boldsymbol{0}$, $\Delta\boldsymbol{\varepsilon}^l_{t<L-l} = \boldsymbol{0}$, and $\Delta\hat{\boldsymbol{x}}^{l+1}_{t<L-l} = \boldsymbol{0}$. □

Lemma 3.2.4. *The prediction error of PC $\varepsilon^l_{i,t}$ at $t = L - l + 1$ (i.e., $\varepsilon^l_{i,L-l+1}$) under C3.2 and C3.2 can be derived from itself at previous relaxation moments in the later layer: $\boldsymbol{\varepsilon}^{l+1}_{k,t}$ at $t = L - l$ (i.e., $\boldsymbol{\varepsilon}^{l+1}_{i,L-l}$). Formally,*

$$\boldsymbol{\varepsilon}^l_{i,L-l+1} = f'\left(\hat{x}^l_{i,0}\right) \sum_{k=1}^{n^{l+1}} \boldsymbol{\varepsilon}^{l+1}_{k,L-l} w^l_{k,i} \text{ for } l \in \{L, \dots, 1\} \tag{3.13}$$

Proof. I first write a dynamic version of $\varepsilon^l_{i,t} = x^l_{i,t} - \hat{x}^l_{i,t}$:

$$\varepsilon^l_{i,t} = \varepsilon^l_{i,t-1} + \left(\Delta x^l_{i,t-1} - \Delta\hat{x}^l_{i,t-1}\right) \tag{3.14}$$

where $\Delta\hat{x}^l_{i,t-1} = \hat{x}^l_{i,t} - \hat{x}^l_{i,t-1}$. Then, I expand $\varepsilon^l_{i,L-l+1}$ with the above equation and simplify it with Lemma 3.2.3, i.e., $\varepsilon^l_{i,t<L-l+1} = 0$ and $\Delta\hat{x}^{l+1}_{i,t<L-l} = 0$:

$$\begin{aligned}\varepsilon^l_{i,L-l+1} &= \varepsilon^l_{i,L-l} + \left(\Delta x^l_{i,L-l} - \Delta\hat{x}^l_{i,L-l}\right) \\ &= \Delta x^l_{i,L-l} \text{ for } l \in \{L, \ldots, 1\}\end{aligned} \tag{3.15}$$

I further investigate $\Delta x^l_{i,L-l}$ expanded with the relaxation dynamic Equation (3.7) and simplify it with Lemma 3.2.3, i.e., $\varepsilon^l_{i,t<L-l+1} = 0$,

$$\begin{aligned}\Delta x^l_{i,L-l} &= \gamma\left(-\varepsilon^l_{i,L-l} + f'\left(x^l_{i,L-l}\right)\right)\sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,L-l} w^l_{k,i} \\ &= \gamma f'\left(x^l_{i,L-l}\right)\sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,L-l} w^l_{k,i} \text{ for } l \in \{L, \ldots, 1\}\end{aligned} \tag{3.16}$$

Putting Equation (3.16) into Equation (3.15), we can obtain:

$$\varepsilon^l_{i,L-l+1} = \gamma f'\left(x^l_{i,L-l}\right)\sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,L-l} w^l_{k,i} \text{ for } l \in \{L, \ldots, 1\} \tag{3.17}$$

With Lemma 3.2.3, $x^l_{i,L-l}$ can be replaced with $x^l_{i,0}$. With C3.2, we can further replace $x^l_{i,0}$ with $\hat{x}^l_{i,0}$. Thus, the above equation becomes:

$$\varepsilon^l_{i,L-l+1} = \gamma f'\left(\hat{x}^l_{i,0}\right)\sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,L-l} w^l_{k,i} \text{ for } l \in \{L, \ldots, 1\} \tag{3.18}$$

Then, put C3.2, $\gamma = 1$, into the above equation. □

3.2.2 Solely relaxing C3.2

Solely relaxing C3.2 results in the conclusion of Lemma 3.2.4, i.e., Equation (3.13) changing to:

$$\varepsilon^l_{i,L-l+1} = \gamma f'\left(\hat{x}^l_{i,0}\right)\sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,L-l} w^l_{k,i} \text{ for } l \in \{L, \ldots, 1\} \tag{3.19}$$

Algorithm 7: Learning one training pair $(\boldsymbol{s}^{\text{in}}, \boldsymbol{s}^{\text{target}})$ (presented for the duration $\mathcal{T}$) with A-Z-PC

Require: $\boldsymbol{x}^1$ is fixed to $\boldsymbol{s}^{\text{in}}$, $\boldsymbol{x}^{L+1}$ is fixed to $\boldsymbol{s}^{\text{target}}$;
Require: $x^l_{i,0} = \hat{x}^l_{i,0}$ for $l \in \{2, \dots, L+1\}$ (C3.2), and $\gamma = 1$ (C3.2);
for $t = 0;\ t < \mathcal{T};\ t = t+1$ **do** // Presenting a training pair
  for *each neuron* i *in each level* l **do** // In parallel in the brain
    Update $x^l_{i,t}$ to minimize E_t via Equation (3.7) ; // Relaxation
    Update each $w^l_{i,j}$ to minimize E_t via Equation (3.10) with learning rate $\alpha \cdot \phi\left(\varepsilon^l_{i,t}\right)$;
  end
end

since the derivation of Lemma 3.2.4 terminates at Equation (3.18). It further causes the conclusion of Theorem 3.2.1 changing from $\varepsilon^l_{i,t} = \delta^l_i$ to $\varepsilon^l_{i,t} = \gamma^{L-l+1}\delta^l_i$ at $t = L - l + 1$, the proof of which is the same as that of the original Theorem 3.2.1 but using the new Lemma 3.2.4. The change in the conclusion of Theorem 3.2.1 immediately changes the conclusion of Theorem 3.2.2 from $\partial E_t / \partial w^{l-1}_{i,j} = \partial F / \partial \theta^{l-1}_{i,j}$ to $\partial E_t / \partial w^{l-1}_{i,j} = \gamma^{L-l+1} \partial F / \partial \theta^{l-1}_{i,j}$, where $t = L - l + 1$. Thus, solely relaxing the condition $\gamma = 1$, i.e., relaxing C3.2 while keeping C3.2 and C3.2 satisfied, results in backpropagation with a different learning rate for different layers, where γ is the decay factor of this learning rate along layers.

3.3 Autonomous zero-divergent PC

PC can autonomously switch between prediction and learning (some autonomy), via running relaxation. However, a control signal is still needed to trigger the weights update at $t = t_c$; thus, the autonomy of triggering the weight update is not realized yet.

Both PC and Z-PC optimize the unified energy of Equation (3.6) during prediction and learning. Thus, they both enjoy some autonomy already. Specifically, they can autonomously switch between prediction and learning by running relaxation, depending only on whether $\boldsymbol{x}^{L+1}_t$ is fixed to $\boldsymbol{s}^{\text{target}}$ or left unconstrained. However,

Table 3.1: Learning rules and their properties.

	Equivalence to backpropagation	Local plasticity	Autonomously triggers weight update
Backpropagation	✓	✗	✗
PC	✗	✓	✗
Z-PC	✓	✓	✗
A-Z-PC	✓	✓	✓

when to update the weights still requires the control signal at $t = t_c$ and $t = L - l + 1$ for PC and Z-PC, respectively.

Targeting at removing this control signal from Z-PC, I now propose *A-Z-PC*, which realizes Z-PC with the autonomy of triggering weight update. Specifically, I propose a function $\phi(\cdot)$ that takes in $\varepsilon^l_{i,t}$ and modulates the learning rate α, producing a local learning rate for $w^{l-1}_{i,j}$. As can be seen from Figure 2.2, $\varepsilon^l_{i,t}$ is directly connected to $w^{l-1}_{i,j}$, meaning that $\phi(\cdot)$ works locally at a neuron scale. For each $w^{l-1}_{i,j}$, $\phi\left(\varepsilon^l_{i,t}\right)$ produces a spike of 1 at exactly the relaxation moment of $t = L - l + 1$, which equals to trigging $w^{l-1}_{i,j}$ to be updated at $t = L - l + 1$. In this way, I can let the relaxation and weights update run all the time with $\phi(\cdot)$ modulating the local learning rate with local information, thus, the resulting model works autonomously in triggering weight update, performs all computations locally, and updates weights at $t = L - l + 1$, i.e., producing exactly Z-PC/backpropagation as long as C3.2 and C3.2 are satisfied. A-Z-PC is summarized in Algorithm 7.

As the core of A-Z-PC, I found that quite simple functions $\phi(\cdot)$ can detect the relaxation moment of $t = L - l + 1$ from $\varepsilon^l_{i,t}$. Specifically, from Lemma 3.2.3, we should know that $\varepsilon^l_{i,t}$ under C3.2 diverges from its stable states at exactly $t = L - l + 1$, i.e., $\boldsymbol{\varepsilon}^l_{t<L-l+1} = \mathbf{0}$. Thus, $\phi(\cdot)$ can take in $\varepsilon^l_{i,t}$ and return 1 only if $\varepsilon^l_{i,t-t_d} = 0, \varepsilon^l_{i,t-t_d+1} = 0, \ldots, \varepsilon^l_{i,t-1} = 0, \varepsilon^l_{i,t} \neq 0$, where t_d is a hyperparameter. In this way, $\phi\left(\varepsilon^l_{i,t}\right)$ detects the relaxation moment of $t = L - l + 1$ and produces a spike of 1 at exactly $t = L - l + 1$. In special cases where $\varepsilon^l_{i,t=L-l+1} = 0$, the detection fails, however, by Equation (3.10), $\Delta w^{l-1}_{i,j}$ produced at $t = L - l + 1$ is also zero, since $\varepsilon^l_{i,t=L-l+1} = 0$, so the failure does no harm. Since it is possible for $\varepsilon^l_{i,t}$ to go across

Table 3.2: Success rate of detecting relaxation moments $t = L - l + 1$ with $\phi(\cdot)$ of different t_d.

t_d	1	2	3	4	5	6	7	8	16
Success rate	93.4%	92.4%	99.6%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

zero, having a larger t_d means a more accurate detection. In experiments, $\phi(\cdot)$ of $t_d = 4$ is already capable of detecting with 100% success rate; see Table 3.2.

All proposed models are summarized in Table 3.1, where A-Z-PC is the only model that is not only equivalent to backpropagation but also performs all computations *locally* and *autonomously triggers the weight update*.

3.4 Experiments

In this section, I complete the picture of this chapter with experimental results, providing ablation studies on MNIST and measuring the running time of the discussed approaches on ImageNet.

3.4.1 Ablation of zero-divergent conditions

I now show that zero-divergent cannot be achieved when strictly weaker conditions than C3.2, C3.2, and C3.2 in Section 3.2 are satisfied only. Specifically, with experiments on MNIST, I show the divergence of PCs trained with ablated variants of Z-PC from backpropagation.

Setup I use the same setup as in (Whittington and Bogacz 2017). Specifically, I train for 64 epochs with $\alpha = 0.001$, batch size 20, and logistic sigmoid as $f()$. To remove the stochasticity of the batch sampler, models in one group within which I measure divergence are set with the same seed. I evaluate three of such groups, each set with different randomly generated seeds ($\{1482555873, 698841058, 2283198659\}$), and the final numbers are averaged over them. The divergence is measured in terms of the error percentage on the test set summed over all epochs (test error, for short), as in (Whittington and Bogacz 2017; Lee et al. 2015; I. Ororbia et al. 2017; Nøkland and Eidnes 2019; A. G. Ororbia and Mali 2019), and of the final

weights, as in (A. G. Ororbia and Mali 2019). The divergence of the test error and the final weights is measured by the L1 and L2 distances, respectively. I conducted experiments on MNIST (with 784 input and 10 output neurons; the other settings are as in (LeCun and Cortes 2010)). I investigated three different network structures with 1, 2, and 3 hidden layers, respectively (i.e., $L \in \{1, 2, 3\}$), each containing 32 neurons.

Ablation of C3.2 Assuming C3.2 and C3.2 satisfied, to ablate C3.2, I consider situations when the network has not converged in the prediction stage before the training pair is presented, i.e., $x^l_{i,0} \neq \hat{x}^l_{i,0}$. To simulate this, I sampled $x^l_{i,0}$ around the mean of $\hat{x}^l_{i,0}$ with different standard deviations σ, where $\sigma = 0$ corresponds to satisfying C3.2. I swipe $\sigma = \{0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$. Figure 3.2, right column, shows that zero divergence is only achieved when $\sigma = 0$, i.e., C3.2 is satisfied. A larger version of Figure 3.2 is given in Figures 3.3 and 3.4.

Ablation of C3.2 and C3.2 Assuming C3.2 satisfied, to ablate C3.2, I swipe $\gamma = \{0.01, 0.1, 0.5, 1, 5, 10\}$, and to ablate C3.2, I swipe $t = \{l, 0, 1, 2, 3, 4, 16, 64\}$. Here, setting t to a fixed number is exactly the implementation of PC with t_c set to this fixed number. I put $t = L - l + 1$ between $t = L - 1$ and $t = L$. This is due to the fact that Z-PC ($t = L - l + 1$) is doing relaxation at a degree between $t = L - 1$ and $t = L$. Specifically, Z-PC ($t = L - l + 1$) uses E_L to update $w^1_{i,j}$ which is a result from doing relaxation at $t = L - 1$, however, Z-PC did not use E_L for weights update in all layers, instead it uses $E_{l<L}$ at previous relaxation moments to update weights $w^l_{i,j}$ in other layers $l > 1$. Thus, I consider that Z-PC ($t = L - l + 1$) lies between $t = L - 1$ and $t = L$ from the perspective of how much relaxation is conducted. For example, for $L = 2$, I present the result as $t = \{0, 1, l, 2, \ldots\}$. Figure 3.2, three left columns, shows that zero divergence is only achieved when $t = L - l + 1$ and $\gamma = 1$, i.e., C3.2 and C3.2 are satisfied. Note that the settings of $t \geq 16$ and $\gamma \leq 0.5$ are typical for PC in (Whittington, T. Muller, et al. 2018).

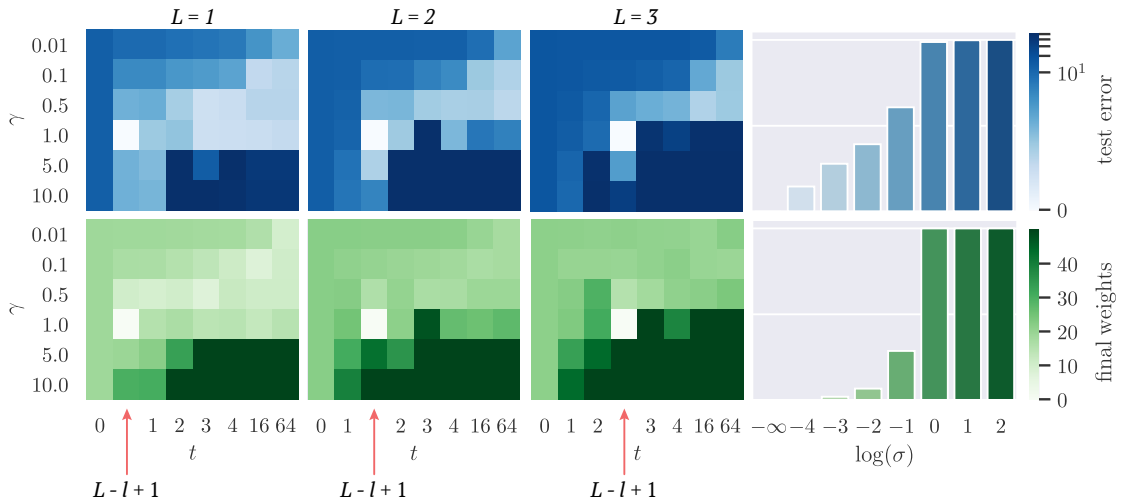


Figure 3.2: Ablation of C3.2, C3.2, and C3.2, where divergence is measured on test error and final weights.

Table 3.3: Unscaled data in Figure 3.4.

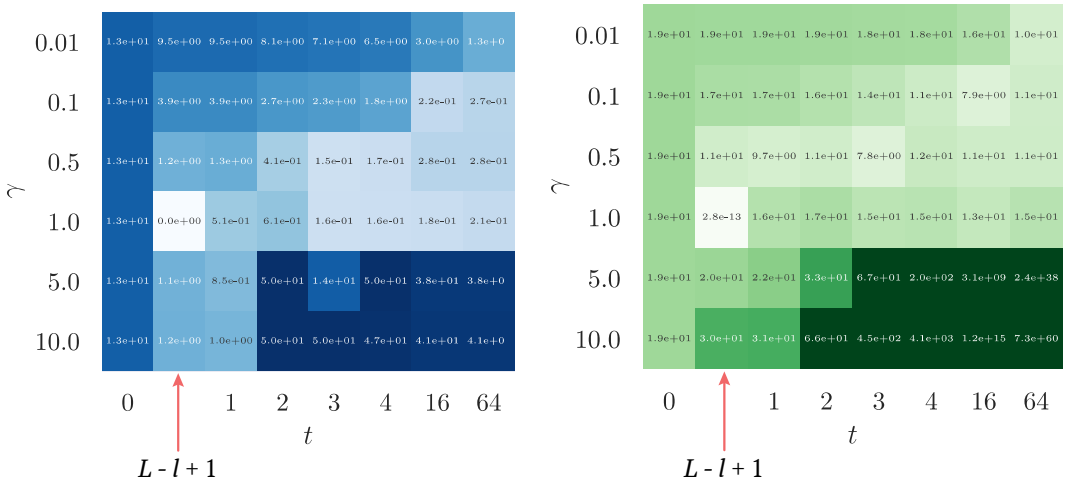
σ	0	0.0001	0.001	0.01	0.1	1	10	100
Divergence of test error	**0**	4.23×10^{-3}	1.77×10^{-2}	6.05×10^{-2}	6.15×10^{-1}	3.73×10^{1}	4.17×10^{1}	4.26×10^{1}
Divergence of final weights	$\mathbf{2.37\times10^{-13}}$	2.40×10^{-2}	9.94×10^{-1}	3.36×10^{0}	1.43×10^{1}	1.08×10^{2}	1.12×10^{3}	1.22×10^{4}

3.4.2 Running time

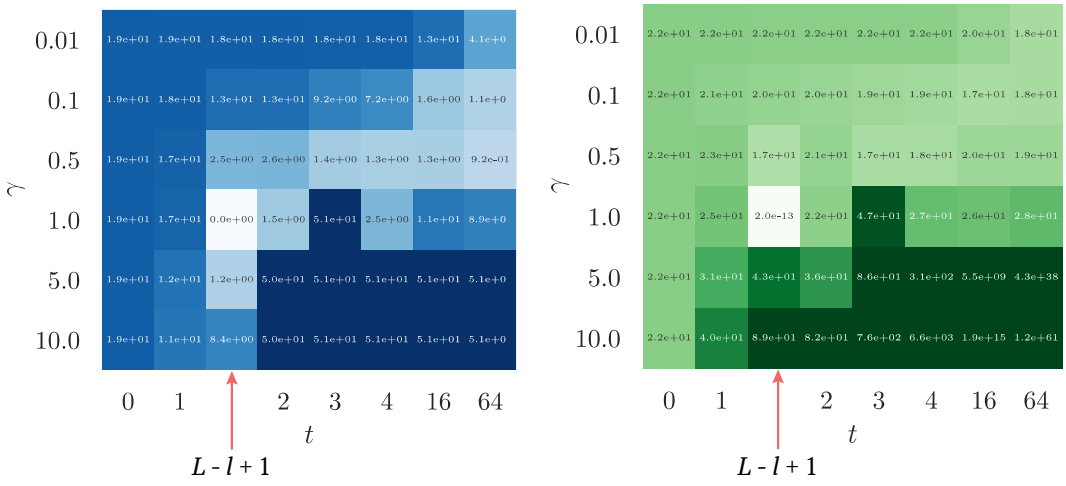
I further conduct experiments on ImageNet to measure the running time of Z-PC and A-Z-PC on large datasets. In detail, I show that Z-PC and A-Z-PC create minor overheads over backpropagation, which supports their scalability. Experiments are conducted on 2 GPUs of Nvidia GeForce GTX 1080Ti, and 8 CPUs of Intel Core i7, with 32 GB RAM. The batch size is set to 1 to test the running time. The input size of the image is 224×224 grayscale, I only implemented fully connected layers: the hidden layers are of size 4096, 2048, and 1024, respectively. The size of the output layer is 27, corresponding to the 27 high-level categories in ImageNet.

Table 3.4 shows the averaged running time of each weights update of backpropagation, PC, Z-PC, and A-Z-PC.

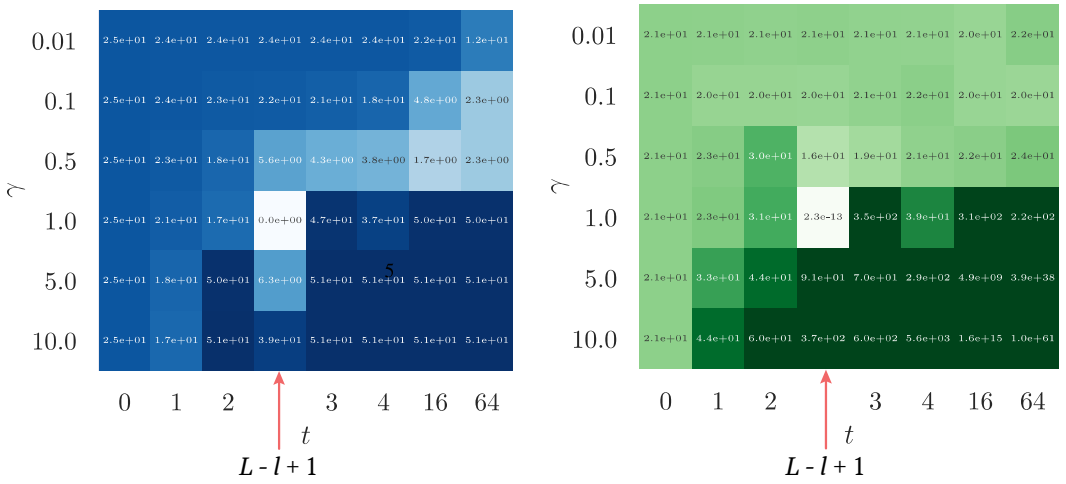
For PC, I set $t_c = 20$, following (Whittington and Bogacz 2017). As can be seen, PC introduces large overheads, since it needs at least t_c relaxation steps before conducting a weights update. In contrast, Z-PC and A-Z-PC run with minor



(a) divergence of test error ($L = 1$) (b) divergence of final weights ($L = 1$)



(c) divergence of test error ($L = 2$) (d) divergence of final weights ($L = 2$)



(e) divergence of test error ($L = 3$) (f) divergence of final weights ($L = 3$)

Figure 3.3: Larger versions (with numbers as annotations) of the subfigures in Figure 3.2.

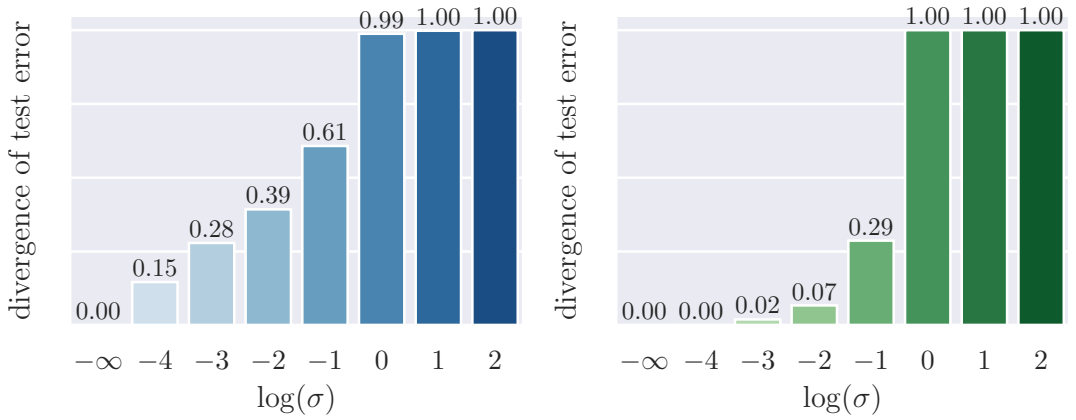


Figure 3.4: Larger versions (with numbers as annotations) of the subfigures in Figure 3.2. For visualization, the divergence of test error has been normalized to $[0, 1]$ by a logarithmic scale and then clipped to $[0, 1]$; the divergence of the final weights has been normalized linearly to $[0, 1]$ from min of 0 and max of 50, values larger than 50 have been clipped to 50. The unscaled data are given in Table 3.3.

Table 3.4: Average runtime of each weights update (in ms) of backpropagation, PC, Z-PC, and A-Z-PC.

Devices	backpropagation	PC	Z-PC	A-Z-PC
CPU	3.1	19.2	3.6	3.6
GPU	3.7	56.3	4.1	4.2

overheads compared to backpropagation, as they require at most $L - 1$ relaxation steps to complete one update of weights in all layers. Comparing A-Z-PC to Z-PC, it is also obvious that the function ϕ creates only minor overheads. These observations support the claim that Z-PC and A-Z-PC are indeed scalable.

3.5 Discussion and related work

One of the main contribution of this chapter is identifying two biologically plausible alternatives to backpropagation: Z-PC and A-Z-PC. Backpropagation has been criticized for being biologically implausible from several aspects. Thus, different efforts have been made in each of these aspects to find biologically plausible alternatives to backpropagation.

The first aspect is that backpropagation encodes error terms in a biologically implausible way, i.e., error terms are not encoded locally. It is often discussed along

with the lack of local plasticity. How error terms can be alternatively encoded and propagated locally has been one of the most intensively studied topics. One promising assumption is that the error term can be represented in dendrites of the corresponding neurons (Körding and König 2001; Körding and König 2000; Richards and Lillicrap 2019). Such efforts are unified in (Lillicrap, Santoro, et al. 2020), with a broad range of works (F. Pineda 1987; F. J. Pineda 1988; O'Reilly 1996; Ackley et al. 1985; G. E. Hinton et al. 1995; Bengio 2014; Lee et al. 2015) encoding the error term in activity differences. PC (thus, also variants of PC introduced in this thesis) was included in such category of encoding the error term in activity differences. However, I cannot fully agree with this, because PC encodes error in separated error neurons, at least in its standard neural implementation as demonstrated in Figure 2.2. Nevertheless, I believe one can come up with a neural implementation of PC that encodes the same error in activity differences.

Backpropagation is also criticized for requiring an external control (e.g., the computation is changed between the prediction and learning phases). An important property of PC, in contrast, is that they autonomously (Whittington and Bogacz 2017) switch between prediction and learning: irrespective of the target pattern being provided, the same rules for neural dynamics and plasticity rules are used. Specifically, if the target pattern is not provided, i.e., the output value neurons are unconstrained, then the error neurons converge to zero, so the weight change is equal to zero. Thus, PC operates without any need for external control except for providing different inputs and target patterns when switching between prediction and learning phases. Z-PC proposed in this chapter also shares this property. However, PC requires an external control over when to update weights ($t = t_c$), and Z-PC requires a similar level of control to only update weights at very specific relaxation moments ($t = L - l + 1$). Among PC and Z-PC, Z-PC can be realized with more autonomy via A-Z-PC proposed in this chapter. In later Chapter 5, I introduce a variant of PC, called PPC, that further removes the requirement of such external control of when to update weights.

Finally, backpropagation is also criticized for backward connections that are symmetric to forward connections in adjacent layers and for using unrealistic models of neurons (i.e., non-spiking neurons). A common way to remove the backward connections (Seung 2003; Werfel et al. 2004; Lillicrap, Cownden, et al. 2016; Scellier and Bengio 2017; Lansdell et al. 2019) is based on the idea of zeroth-order optimization (Shamir 2017; Golovin et al. 2019). In these methods, weights can randomly introduce perturbations and use unspecific feedback signals to observe their effect on the objective function and thus approximate their gradient. However, such methods need many trials depending on the number of weights (Spall 1992; Werfel et al. 2004) Such "weight perturbation" methods can be further improved by perturbing the neural activities instead of the weights (Mazzoni et al. 1991; Flower and Jabri 1993). Admitting the existence of backward connections, more recent works show that asymmetric connections are able to produce a comparable performance (Nøkland 2016; Liao et al. 2016; Amit 2019). As for the models of neurons, backpropagation has recently also been generalized to spiking neurons (Zenke and Ganguli 2018). Our work is orthogonal to the work reviewed in this paragraph, i.e., I still use symmetric backward connections and do not consider spiking neurons.

4

Predictive coding has advantages over backpropagation in learning problems with biological constraints

Contents

This chapter discusses my results that biologically plausible models, particularly Predictive Coding Network (Rao and Ballard 1999; Whittington and Bogacz 2017; Buckley et al. 2017) (PC) and other energy-based models, have advantages over backpropagation in learning problems with biological constraints. In Chapter 3, I demonstrated that a particular modification of PC, Z-PC, is exactly equivalent to BP. In this chapter, I investigate the standard PC. Specifically, I propose that PC and other energy-based models follow a fundamentally different principle compared to backpropagation. I call this principle "prospective configuration" and I demonstrate both theoretically and empirically that this principle enables more effective learning in various contexts faced by biological organisms, and reproduces animal learning effects that cannot be explained by backpropagation. Prospective configuration is not a new specific algorithm, instead, it is a general principle

underlying learning in energy-based models (that includes PC) as reviewed in Section 2.2, where neural activity and synaptic weights simultaneously optimize an energy function. However, I identify and analyze this principle, and demonstrate its superiority over backpropagation for the first time. In experiments, I focus on a specific energy-based model: PC. That is to say, I focus on a specific algorithm that follows the principle of prospective configuration, because PC is closely related to backpropagation (Whittington and Bogacz 2017; Song, Lukasiewicz, et al. 2020; Millidge et al. 2020a) and makes a fair comparison. Nevertheless, I show that prospective configuration's mechanism, definition and empirical advantages also generalize to other energy-based models in Section 4.2, 4.3 and 4.4.2.

4.1 Prospective-configuration in PC

This section demonstrates the principle of prospective configuration in PC, by showing how PC behaves fundamentally differently from backpropagation using a minimal and straightforward example. Later sections describe in detail the mechanisms (Section 4.2) and formal definition (Section 4.3) of prospective configuration, where I demonstrate that the prospective configuration generalizes beyond PC to other energy-based models.

To highlight the difference between backpropagation and PC, let us consider a simple example. Imagine a bear seeing a river where it usually hunts for salmon, the sight generates in the bear's mind predictions of the sound of water and the smell of salmon. On that day the bear indeed hears the sound but does not smell the salmon — the migration season has finished, thus the bear needs to change its expectation related to the smell. The backpropagation would enable such learning, by reducing the weights on the path between visual neurons selective for water and olfactory neurons selective for the smell of salmon (Figure 4.1:b: negative error is backpropagated on the lower branch, so that weights on the shared path and lower branch are reduced). However, this modification would also reduce the expectation of hearing the sound next time the river is visited (Figure 4.1:b:right, neuron activity on the shared path is reduced as a result of weight modification but the weight on the

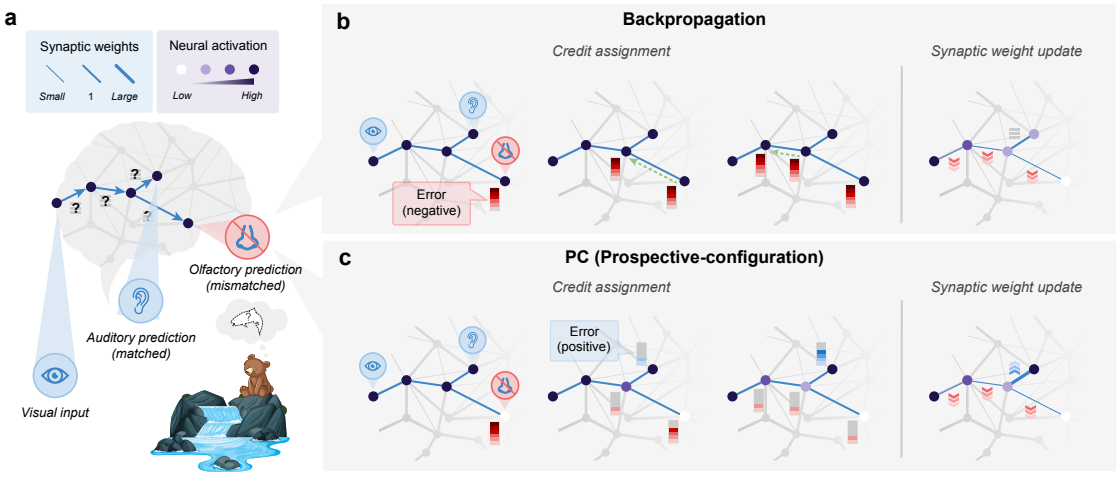


Figure 4.1: PC differs from backpropagation in that the neural activities settle to their prospective state during error spreading, a distinct behavior I called "prospective configuration". An abstract (top) and a concrete (bottom) example of a task inducing interference during learning. One stimulus input (seeing the water) triggers two prediction outputs (hearing the water and smelling the salmon). One output is correct (hearing the water), while the other one is an error (not smelling the salmon). Backpropagation produces interference during learning: not smelling the salmon reduces the expectation of hearing the water (panel b), although the water was indeed heart. Prospective configuration, on the other hand, avoids such interference (panel c).

upper branch stays still), even though the sound was present and correctly predicted in recent experience. Such undesired and unrealistic side-effects of learning with backpropagation are closely related to the phenomenon of catastrophic interference, where learning a new association destroys previously learned memories (McCloskey and N. J. Cohen 1989; McNaughton and O'Reilly 1995). Here, in Figure 4.1:b, I show that with backpropagation even learning one new aspect of an association may interfere with the memory of other aspects of the same association.

In contrast, PC assumes that learning starts from the neurons being re-configured to a new state – which corresponds to the activities that would have occurred after the error is corrected by modifying the weights (Figure 4.1:c:left), i.e., a "prospective" state. The weights are then modified to produce such a state. I call such behavior of PC "prospective configuration", and later demonstrate it is not a unique behavior of PC but generally applies to other energy-based models. Thus, I also call prospective configuration a principle: a behavior commonly followed by a family of models. As a result of this prospective configuration behavior, PC can "foresee" side effects of

potential weight modifications and compensate for them dynamically (Figure 4.1:c: to correct the negative error on the lower branch, the neurons on the shared path settle to their prospective state of lower activities, and as a result, a positive error will be introduced to the upper branch). Consequently, PC increases the weights in the upper branch in the first weight update (Figure 4.1:c:right) while backpropagation does not (cf. right sides of Figure 4.1:b and c). Hence PC is able to correct the side effects of learning an association (thus, reducing interference) within a single learning iteration, while backpropagation would require multiple.

4.2 Mechanisms and theoretical advantages of prospective-configuration

In the last section, the example demonstrates prospective configuration in PC and shows how the principle of prospective configuration present in PC is fundamentally different from backpropagation. This section describes in detail the mechanisms of prospective configuration with PC as an example.

To help understand the behavior of PC and how prospective configuration arises in it, I propose how it can be visualized as an abstract machine with intuitive dynamics.

PC is defined as multivariate dynamical systems, guarded by an energy function: Section 2.2.1 lists equations describing its neural dynamics and plasticity, which are restated in Figure 4.2:a for easier reference. In Figure 4.2:a, Equation (5) and Equation (7) highlight that both the neural dynamics and the plasticity of PC minimize the energy function. Figure 4.2:b restates the list of notations used in describing PC. Figure 4.2:c reminds the standard neural implementation of PC.

To provide an intuition for the dynamics of PC (neural dynamic of Equation (5) and plasticity of Equation (7) in 4.2:a), I show that the same set of equations can also describe a physical machine made of rods and springs that hold potential energy, thus, called "energy machine". As shown in Figure 4.2:d, the energy machine includes nodes sliding on vertical posts, connected with each other via rods and springs. Translating from the PC to the energy machine, neural activity maps to

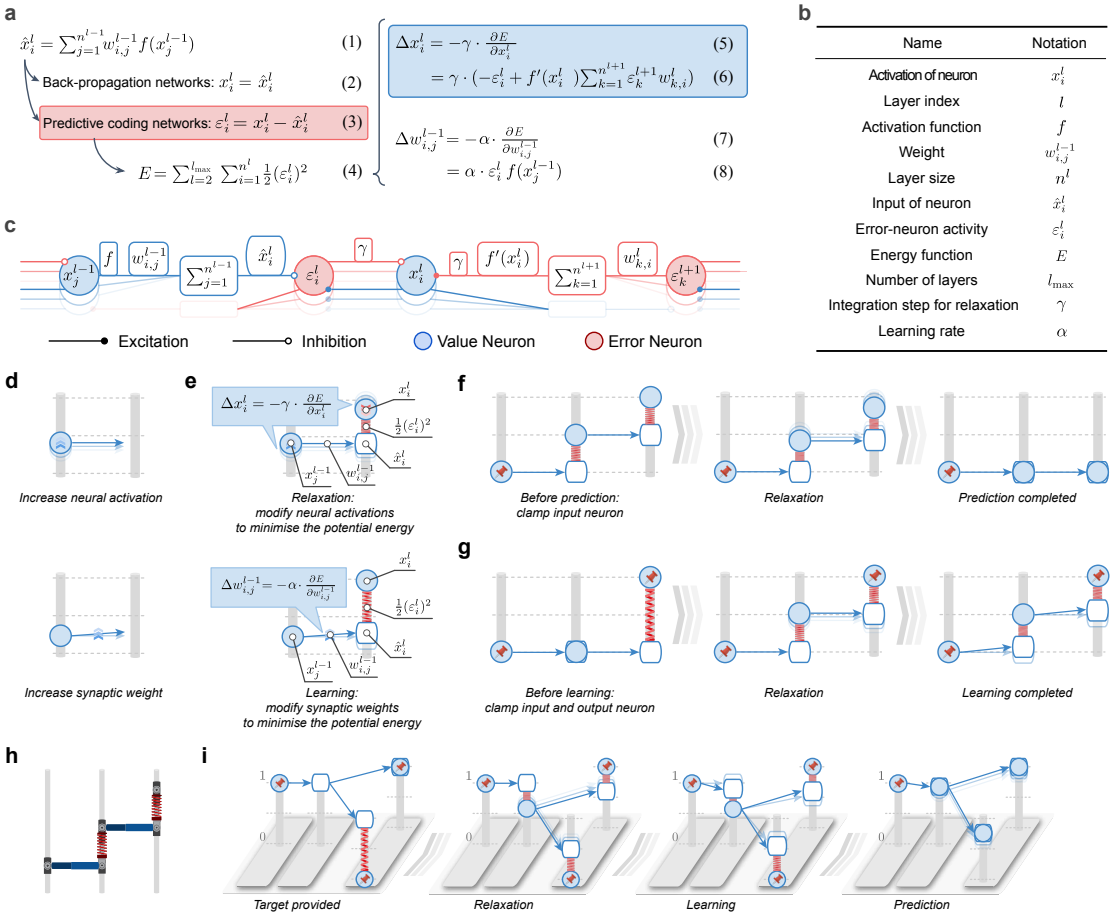


Figure 4.2: A machine analogy (called "energy machine") reveals a new understanding of PC, mechanism of prospective configuration, and its theoretic advantages. In the energy machine, the activity of a neuron corresponds to a height of a node (represented by a solid circle) sliding on a post. The input to the neuron is represented by a hollow node on the same post. A synaptic connection corresponds to a rod pointing from a solid to a hollow node. The synaptic weight determines how the input to a post-synaptic neuron depends on the activity of pre-synaptic neuron, hence it influences the angle of the rod. In energy-based networks, relaxation (i.e., neural dynamics) and weight modification (i.e., weight dynamics) are both driven by minimizing the energy, thus correspond to relaxing the energy machine by moving the nodes and tuning the rods, respectively.

the vertical position of a node (Figure 4.2:d:top), synaptic connection maps to a rod pointing from one node to another and the synaptic weight determines how the end (right) position of the rod relates to the initial (left) position (Figure 4.2:d:bottom).

As shown in Figure 4.2:e, in PC, postsynaptic neural activity (right circle node) is not set as the input of the neuron (rectangle node). Instead, the energy of error is defined between the activity and the input of the neuron, thus, corresponding to the elastic potential energy of a spring, whose relaxed length is 0, attached between

the circle and the rectangle nodes. Thus, in Figure 4.2:e, translating from the PC to the energy machine, the energy function maps to the elastic potential energy of springs with nodes attached on both ends. In this way, the dynamics of PC, which are driven by minimizing the energy function, map to the relaxation of the energy machine, which is driven by reducing the total elastic potential energy on the springs. In Figure 4.2:e:top, I demonstrate that the dynamics of neural activity of PC, i.e., Equation (5) in the figure, map to the energy machine: neural activity x is updated to minimize the energy of error E, corresponding to relaxing the energy machine by moving the circle nodes. In Figure 4.2:e:bottom, I demonstrate that the dynamics of the synaptic weight of PC, i.e., Equation (7) in the figure, map to the energy machine: synaptic weight w is updated to also minimize the energy of error E, corresponding to adjusting the energy machine by tuning the rods.

In Figure 4.2:f, prediction with PC involves clamping the input neuron (e.g. according to the provided input pattern $\boldsymbol{s}^{\text{in}}$) and updating the activity of other neurons, which in the energy machine corresponds to fixing one side of the energy machine and letting the energy machine relax by moving nodes. In Figure 4.2:g, learning with PC involves clamping the input and output neurons to corresponding input $\boldsymbol{s}^{\text{in}}$ and target $\boldsymbol{s}^{\text{target}}$ patterns, letting the activity of remaining neurons converge, and updating weights, which corresponds to fixing both sides of the energy machine and letting the energy machine relax first by moving nodes, then by tuning rods.

Figure 4.2:h shows the detailed version of the energy machine.

This energy machine of PC reveals the essence of PC (and other energy-based models): the relaxation during learning lets the network settle to a new configuration of neural activities, corresponding to those that would have occurred after the error was corrected by the modification of weights, i.e., a prospective state (thus, such behavior is called prospective configuration). For example, the second layer "neuron" in Figure 4.2:g increases its activity, and this increase in activity would also be caused by the subsequent weight modification (of connection between the first and second neurons). In simple terms, the relaxation in PC infers the prospective

state of neural activity after learning, towards which the weights are modified then. Using the energy machine, Figure 4.2:i simulates the learning problem from Figure 1. Here we can see PC indeed foresees the result of learning and its side effects, through relaxation. Particularly, in Figure 4.2:i,

- The first display shows the moment right after the target pattern is provided. Here, there is a negative error on the output neuron of the lower branch, while the output neuron of the top branch makes a correct prediction.
- The second display shows the moment the relaxation converges. At this time, PC foresees the result of learning that the hidden neural activity will be decreased, thus, introducing a positive error to the output neuron of the upper branch.
- The third display shows the weights update according to the error allocated in the previous display. We can see the weights connecting the input to the output neuron of the bottom branch are weakened, while the weight connecting the hidden to the output neuron of the top branch are strengthened.
- The last display shows the result of the above weight update, i.e., making a prediction with the updated weights. We can see from this panel, PC learns to correct the negative error on the output neuron of the bottom branch while trying to avoid introducing error to the output neuron of the upper branch (which made a correct prediction in the recent experience).

Hence, PC learns to avoid interference within one iteration that would otherwise take multiple for backpropagation.

4.3 Definition of prospective configuration with prospective index

Following the straightforward example in Section 4.1 and energy machine in the last Section 4.2, we can see how PC, i.e., prospective configuration is fundamentally different from backpropagation and theoretically more advanced than backpropagation.

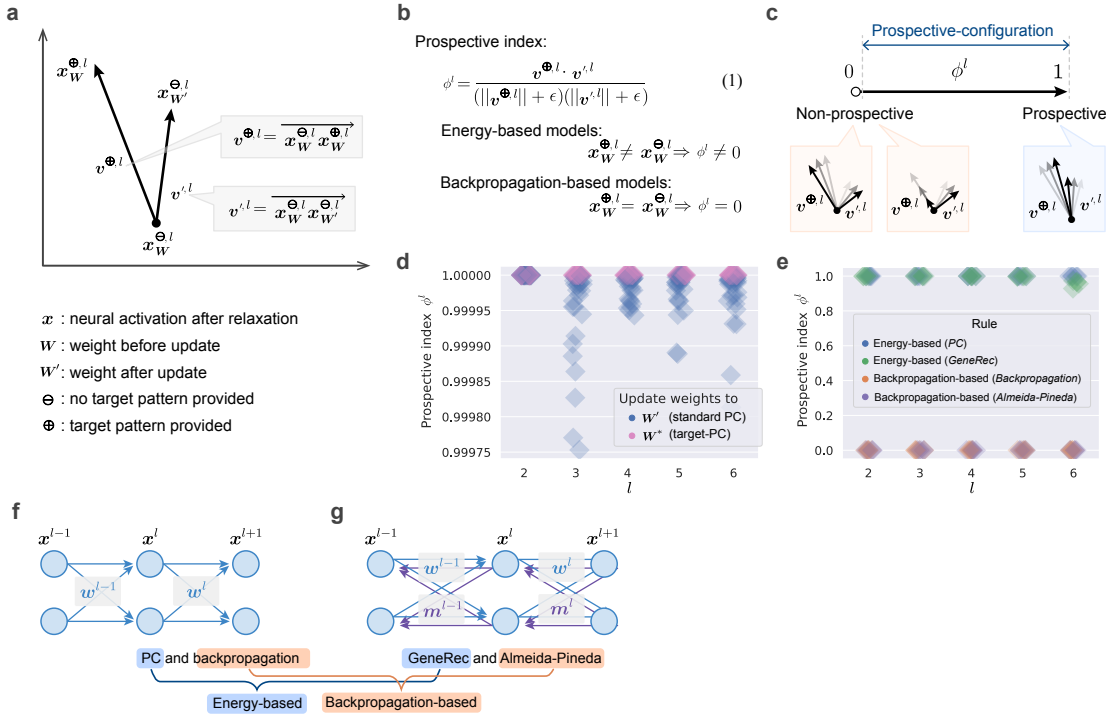


Figure 4.3: Definition of prospective configuration with prospective index and generalization of prospective configuration to other energy-based models. Formal definition of prospective configuration with prospective index (panels a–c), a metric that one can measure for any learning model. With this metric, we show that prospective configuration is present in different energy-based models, but not in backpropagation-based models (panels d–e). And the differences in different energy-based and backpropagation-based models are illustrated in panels f–g.

This section formally defines prospective configuration, where we are able to quantify and qualify prospective configuration in different models. Specifically, I formally define prospective configuration using the prospective index (Figure 4.3:a-c), a metric we can measure for any learning model. With this metric I show that prospective configuration is present in one of energy-based models PC (Figure 4.3:d) as well as other energy-based models (Figure 4.3:e); in contrast, prospective configuration is not present in backpropagation and other backpropagation-based models (Figure 4.3:e).

To introduce prospective index, in Figure 4.3:a, each point on the plane represents the neural activity of the hidden layer l ($1 < l < L+1$ so that it is a hidden layer), denoted by $\boldsymbol{x}^l$. I assume that the model does not make a perfect prediction with the current weights, so that the error in the prediction drives the learning. I consider $\boldsymbol{x}^l$ in the following three phases.

1. Learning starts from $\boldsymbol{x}^l$ under current weights $\boldsymbol{W}$ without target pattern provided ($\ominus$): $\boldsymbol{x}^{\ominus,l}_{\boldsymbol{W}}$. $\boldsymbol{W}$ includes weights in all layers: $\boldsymbol{W} = \left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$, where $\boldsymbol{w}^l$ is weights connecting from layer l to layer $l+1$. For energy models with recurrent connections (e.g., GeneRec), it additionally includes the backward weights: $\boldsymbol{W} = \left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\} \bigcup \left\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\right\}$. For energy-based models, in this phase only input neurons are clamped to the input pattern $\boldsymbol{s}^{\text{in}}$ and the network is relaxed. For backpropagation-based models, in this phase the model makes a prediction with current weight and input pattern;

2. Then, a target pattern $\boldsymbol{s}^{\text{target}}$ is provided ($\oplus$), and the network settles to a new equilibrium $\boldsymbol{x}^{\oplus,l}_{\boldsymbol{W}}$. For energy-based models, in this phase both input and output neurons are fixed to the input $\boldsymbol{s}^{\text{in}}$ and target $\boldsymbol{s}^{\text{target}}$ pattern, respectively, and the hidden neuron activities settle to $\boldsymbol{x}^{\oplus,l}_{\boldsymbol{W}}$ after relaxation, thus, $\boldsymbol{x}^{\oplus,l}_{\boldsymbol{W}} \neq \boldsymbol{x}^{\ominus,l}_{\boldsymbol{W}}$. For backpropagation-based models, the hidden neuron activities do not change when the target pattern $\boldsymbol{s}^{\text{target}}$ is provided, thus, $\boldsymbol{x}^{\oplus,l}_{\boldsymbol{W}} = \boldsymbol{x}^{\ominus,l}_{\boldsymbol{W}}$;

3. Finally, the weights $\boldsymbol{W}$ are updated to $\boldsymbol{W}'$, the target pattern is removed ($\ominus$), and the neural activities settle to $\boldsymbol{x}^{\ominus,l}_{\boldsymbol{W}'}$. For energy-based models, in this phase the output neuron is freed from being clamped to the target pattern but the input neurons are still clamped to the input pattern and the network is relaxed. For backpropagation-based models, in this phase, it makes a prediction with the input pattern and the new weights $\boldsymbol{W}'$.

I define two vectors $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$ as shown in Figure 4.3:a, representing the direction of the hidden neural activity's changes as a result of the target pattern being given ($\ominus \rightarrow \oplus$) and the weights being updated $\boldsymbol{W} \rightarrow \boldsymbol{W}'$, respectively.

I then propose a quantification of prospective configuration by the prospective index in Figure 4.3:b, I define prospective index of hidden layer l, ϕ^l, to be the cos similarity of $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$ (a small constant ϵ is added to the denominator to prevent it from being equal to 0). For energy-based models, the neural activities settle to a new configuration when the target pattern is provided, i.e., $\boldsymbol{x}^{\oplus,l}_{\boldsymbol{W}} \neq \boldsymbol{x}^{\ominus,l}_{\boldsymbol{W}}$, so the

prospective index ϕ^l is non-zero; for backpropagation-based models, the neuron activities are stationary when the target pattern is provided, i.e., $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l} = \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}$, so the prospective index ϕ^l is zero. As demonstrated in Figure 4.3:c, a positive (close to 1) prospective index ϕ^l implies $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$ are pointing in the same direction, indicating that the hidden neural activity after the target pattern provided before the weight update $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$ informs the hidden neural activity after the weight update $\boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,l}$, i.e., $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$ is prospective. Thus, I define models with a positive prospective index averaged over all the updates to be a model following the principle of prospective configuration. In contrast, a zero or negative prospective index ϕ^l implies $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$ are not pointing in the same direction (or $\boldsymbol{v}^{\oplus,l}$ is of zero length), indicating that the hidden neuron activities after the target pattern is provided before the weight update $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$ is not informative of the hidden neural activity after the weight update $\boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,l}$, i.e., $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$ is non-prospective. Thus, models with zero or negative prospective index do not follow the principle of prospective configuration.

In Figure 4.3:d, I demonstrate prospective index ϕ^l of different layers l (x-axis) in standard PC (blue squares). We can see that the prospective index of the first hidden layer ($l = 2$) in standard PC (blue squares) is always one. This observation is formally proved in Section 4.3.1. We can also see that for $l > 2$ prospective index ϕ^l in standard PC (blue squares) is close to 1 but slightly smaller. To shed light on this observation, I define a modified version of standard PC, i.e., target-PC, which has $\phi^l = 1$ for all layers (pink squares). Specifically, in Figure 4.3:d, standard PC (blue squares) updates weights for one step to $\boldsymbol{W}'$ with Equations (2.12) and (2.13). In contrast, target-PC (pink squares) is updating weights for many steps until convergence to $\boldsymbol{W}^*$ with Equations (2.12) and (2.13). Target-PC is summarized in Algorithm 8 and it informs what the prospective index of the standard PC is. I formally prove in Section 4.3.2 that the prospective index ϕ^l for target-PC is 1 for all layers. Thus, target-PC (pink squares) explains why prospective index ϕ^l in standard PC (blue squares) is close to 1: standard PC updates the weights towards the values in target-PC, so the updates in the two models should be in a similar direction.

Algorithm 8: Learn with target-PC

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

Output: updated synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern

$\boldsymbol{x}^{L+1} = \boldsymbol{s}^{\text{target}}$; // Clamp output neurons to target pattern

for $t = 0;\ t < \mathcal{T};\ t = t + 1$ **do** // Relaxation

  for $l = 1;\ l < L + 1;\ l = l + 1$ **do**

    $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$; // according to Equation (2.9)

    $\boldsymbol{\varepsilon}^{l+1} = \boldsymbol{x}^{l+1} - \hat{\boldsymbol{x}}^{l+1}$; // according to Equation (2.10)

  end

  for $l = 2;\ l < L + 1;\ l = l + 1$ **do**

    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\left(\boldsymbol{w}^l\right)^T \boldsymbol{\varepsilon}^{l+1}\right)\right)$; // according to Equation (2.13)

    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;

  end

end

while $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ *not converged* **do** // Update weights till convergence

  for $l = 1;\ l < L + 1;\ l = l + 1$ **do**

    $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$; // according to Equation (2.9)

    $\boldsymbol{\varepsilon}^{l+1} = \boldsymbol{x}^{l+1} - \hat{\boldsymbol{x}}^{l+1}$; // according to Equation (2.10)

  end

  for $l = 1;\ l < L + 1;\ l = l + 1$ **do** // Update weights

    $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1}\left(f\left(\boldsymbol{x}^l\right)\right)^T$;

    $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;

  end

end

In Figure 4.3:e, I demonstrate prospective index ϕ^l of different energy-based models and backpropagation-based models. Specifically, there are two energy-based models: PC (blue) and GeneRec (green); two backpropagation-based models: backpropagation (orange) and Almeida-Pineda (purple) as reviewed in Chapter 2. Here, we can see that all the energy-based models produce prospective indices ϕ^l close to 1, i.e., the prospective configuration is commonly observed in both energy-based models. In other words, these energy-based models follow the principle of prospective configuration. In contrast, both backpropagation-based models produce a prospective index 0.

The experiments in Figure 4.3:d-e were performed in the following way. I trained the energy-based models and backpropagation-based to predict (with output neurons) a randomly generated pattern (32 entries) from a randomly generated input (32 entries). The structure of the network was $32 \to 32 \to 32 \to 32 \to 32 \to 32$. I used random patterns instead of specific choices of tasks so that the findings are more general. The models were trained for one iteration (i.e., one update of the weights), and the prospective index was then measured for this update. Experiments were done with 10 random seeds.

4.3.1 Prospective index of standard PC is one for the first hidden layer

As stated earlier, I assume that the model does not make a perfect prediction with the current weights, so that the error in the prediction drives the learning. As defined, vectors $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$ describe the changes in hidden neuron activity, due to the target pattern being provided and learning respectively. Specifically for layer $l = 2$, these vectors are:

$$\boldsymbol{v}^{\oplus,2} = \boldsymbol{x}_{\boldsymbol{W}}^{\oplus,2} - \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2} \tag{4.1}$$

$$\boldsymbol{v}'^{,2} = \boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,2} - \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2} \tag{4.2}$$

I will now show that for PC the above vectors $\boldsymbol{v}^{\oplus,2}$ and $\boldsymbol{v}'^{,2}$ point in the same direction. The change in activity due to learning $\boldsymbol{v}'^{,2}$ is equal to

$$\boldsymbol{v}'^{,2} = \boldsymbol{w}'^{,1} f\left(\boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,1}\right) - \boldsymbol{w}^{1} f\left(\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,1}\right) \tag{4.3}$$

Since the value nodes of the first (input) layer $\boldsymbol{x}^1$ are always fixed to the input signal $\boldsymbol{s}^{\text{in}}$, the above Equation (4.3) can further be written as,

$$\begin{aligned}\boldsymbol{v}'^{,2} &= \boldsymbol{w}'^{,1} f\left(\boldsymbol{s}^{\text{in}}\right) - \boldsymbol{w}^1 f\left(\boldsymbol{s}^{\text{in}}\right) \\ &= \left(\boldsymbol{w}'^{,1} - \boldsymbol{w}^1\right) f\left(\boldsymbol{s}^{\text{in}}\right) \\ &= \Delta \boldsymbol{w}^1 f\left(\boldsymbol{s}^{\text{in}}\right)\end{aligned} \tag{4.4}$$

Using Equations (2.15) and (2.10), I write

$$\boldsymbol{v}'^{,2} = \alpha \left(\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,2} - \hat{\boldsymbol{x}}_{\boldsymbol{W}}^{\oplus,2}\right) \left(f\left(\boldsymbol{s}^{\text{in}}\right)\right)^T f\left(\boldsymbol{s}^{\text{in}}\right) \tag{4.5}$$

Note that $\hat{\boldsymbol{x}}_{\boldsymbol{W}}^{\oplus,2} = \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2}$, because both of these quantities are equal to $\boldsymbol{w}^1 f\left(\boldsymbol{s}^{\text{in}}\right)$ (the input of the first hidden layer ($l = 2$) does not change in response to output neuron being clamped). Using $\hat{\boldsymbol{x}}_{\boldsymbol{W}}^{\oplus,2} = \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2}$, the above Equation (4.5) can further be written as,

$$\boldsymbol{v}'^{,2} = \left(\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,2} - \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2}\right) \alpha \left(f\left(\boldsymbol{s}^{\text{in}}\right)\right)^T f\left(\boldsymbol{s}^{\text{in}}\right) \tag{4.6}$$

Note that $\alpha \left(f\left(\boldsymbol{s}^{\text{in}}\right)\right)^T f\left(\boldsymbol{s}^{\text{in}}\right)$ is a positive scalar (if at least one entry in the input pattern is non-zero). Comparing Equations (4.1) and (4.6), we can see that vectors $\boldsymbol{v}'^{,2}$ and $\boldsymbol{v}^{\oplus,2}$ are just scaled versions of each other, hence the cosine of the angle between them and thus prospective index are equal to 1.

4.3.2 Prospective index of target-PC is one for all layers

Similarly, I assume that the model does not make a perfect prediction with the current weights, so that the error in the prediction drives the learning. I start with recapping what happens in sequence in one iteration of a standard PC.

1. Start from relaxation with only input neurons clamped to input pattern ($\oplus$) and with current weight, $\boldsymbol{W}$, the hidden neuron activity settles to $\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}$;

2. Both input and output neurons are clamped to the input and target pattern respectively ($\oplus$) and then the hidden neuron activity is relaxed to $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$;

3. Weights $\boldsymbol{W}$ are updated for one step to $\boldsymbol{W'}$ to decrease the energy, while hidden neuron activity stays still from the last step: $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$;

4. Output neurons are freed but the input neuron is still clamped to the input pattern and then the hidden neuron activity is relaxed to $\boldsymbol{x}_{\boldsymbol{W'}}^{\ominus,l}$;

In the above step 3, weights are updated for one step from $\boldsymbol{W}$ to $\boldsymbol{W'}$. However, one can investigate the case of updating weights $\boldsymbol{W}$ for many steps until convergence $\boldsymbol{W}^*$ in the above step 3. This will result in weights $\boldsymbol{W}^*$ that represent: "the target towards which the weights $\boldsymbol{W}$ are updated". Thus, I call this variant "target-PC". Specifically, the procedure of target-PC is to replace the above steps 3 and 4 of standard PC with:

3. Weights are updated for many steps from $\boldsymbol{W}$ to $\boldsymbol{W}^*$ to decrease the energy till convergence, while hidden neuron activity stays still from the last step: $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$;

4. Output neurons are freed but the input neuron is still clamped to the input pattern and then the hidden neuron activity is relaxed to $\boldsymbol{x}_{\boldsymbol{W}^*}^{\ominus,l}$;

In the following, I demonstrate the prospective index of target-PC is one for all layers. First, we should notice that the minimum of energy $\boldsymbol{E}$ of PC is zero, since the energy function is a sum of quadratic terms (Equation (2.11)).

Then, we should notice that such energy $\boldsymbol{E}$ of PC can be optimized to its minimum of zero by optimizing only $\boldsymbol{W}$. Particularly, looking at the energy term of layer l:

$$\begin{aligned}\frac{1}{2}\left(\boldsymbol{\varepsilon}^l\right)^T\boldsymbol{\varepsilon}^l &= \frac{1}{2}\left(\boldsymbol{x}^l-\hat{\boldsymbol{x}}^l\right)^T\left(\boldsymbol{x}^l-\hat{\boldsymbol{x}}^l\right)\\ &= \frac{1}{2}\left(\boldsymbol{x}^l-\boldsymbol{w}^{l-1}f\left(\boldsymbol{x}^{l-1}\right)\right)^T\left(\boldsymbol{x}^l-\boldsymbol{w}^{l-1}f\left(\boldsymbol{x}^{l-1}\right)\right)\end{aligned} \tag{4.7}$$

In the above equation, $\boldsymbol{x}^l-\boldsymbol{w}^{l-1}f\left(\boldsymbol{x}^{l-1}\right)$ can be optimized to produce a zero vector by optimizing only $\boldsymbol{w}^{l-1}$, as long as $f\left(\boldsymbol{x}^{l-1}\right)$ is not a zero vector. Specifically,

let's denote all the non-zero entries in $f\left(\boldsymbol{x}^{l-1}\right)$ by $\left\{f\left(x_i^{l-1}\right)\right\}_{i\in I}$, where I is the set of indexes i so that $f\left(x_i^{l-1}\right)$ is non-zero. Since $f\left(\boldsymbol{x}^{l-1}\right)$ is not a zero vector, $I \neq \emptyset$. For any entry x_j^l of any value in vector $\boldsymbol{x}^l$, $\left\{w_{j,k}^{l-1}\right\}_{k\notin I}$ can be of any value, and there is a solution for $\left\{w_{j,i}^{l-1}\right\}_{i\in I}$ so that $x_j^l = \sum_{i\in I} w_{j,i}^{l-1} f\left(x_i^{l-1}\right)$. At least one of the solution is to pick one index g from I, then have $w_{j,g}^{l-1} = \frac{x_j^l}{f\left(x_i^{l-1}\right)}$ and $\left\{w_{j,i}^{l-1} = 0 : i \in I, i \notin \{g\}\right\}$. Thus, as long as $f\left(\boldsymbol{x}^{l-1}\right)$ is not a zero vector ($I \neq \emptyset$), there is a solution of $\boldsymbol{w}^{l-1}$ that makes $\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)$ a zero vector.

Thus, in step 3 of the target-PC, the energy of the network is at its minimum of zero. This further gives that in the step 4 of the target-PC, the neural activity does not move, i.e.,

$$\boldsymbol{x}_{\boldsymbol{W}^*}^{\ominus,l} = \boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l} \tag{4.8}$$

According to the definition of prospective index in Section 4.3, the prospective index of this target-PC ($\phi^{*,l}$) is:

$$\begin{aligned}
\phi^{*,l} &= \frac{\boldsymbol{v}^{\oplus,l} \cdot \boldsymbol{v}^{*,l}}{\left(||\boldsymbol{v}^{\oplus,l}|| + \epsilon\right)\left(||\boldsymbol{v}^{*,l}|| + \epsilon\right)} \\
&\approx cos\left(\boldsymbol{v}^{\oplus,l}, \boldsymbol{v}^{*,l}\right) \\
&= cos\left(\overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}}, \overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}^*}^{\ominus,l}}\right) \\
&= cos\left(\overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}}, \overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}}\right) \text{ according to Equation (4.8)} \\
&= 1
\end{aligned} \tag{4.9}$$

This theoretical result is further confirmed by empirical observation in Figure 4.3:d. One can also imagine that such conclusion should generalize to a family of energy-based models with a fixed lower bound and the same lower bound can be reached by optimizing solely weights or neural activity.

4.4 Comparison of accuracy and efficiency

4.4.1 Comparison of accuracy and efficiency in learning problems with biological constraints

Inspired by the theoretical advantage of prospective configuration in PC, I show that PC indeed deals with various learning problems that biological systems would face better than backpropagation. Since the field of machine learning has developed effective setups for testing learning performance, I use versions of classic machine learning problems that share key features with the learning tasks in natural environments. Such problems include online learning with the necessity to update synaptic weights after each experience (rather than a batch of training examples) (Igelnik 2013), learning with a limited amount of training examples, with a limited size of architectures, and in changing environments. In all the aforementioned learning problems, PC demonstrates notable superiority over backpropagation, confirming the theoretical advantage of prospective configuration with empirical evidence. I will first present the results of the simulations (Figure 4.4), and the details of the simulations are described later in this section.

Based on my analysis of prospective configuration in PC (Figures 4.2:i), we should expect PC to require fewer episodes for learning than backpropagation. Before presenting the comparison, it is useful to clarify how backpropagation is used to train artificial neural networks. As reviewed in Section 2.3, the weights are typically only modified after a batch of training examples, based on the average of updates derived from individual examples (Figure 4.4:a). In fact, backpropagation relies heavily on averaging over multiple experiences to stabilize training (Ioffe and Szegedy 2015), and to train models that reach human-level performance (Jia et al. 2018; Puri et al. 2018; Berner et al. 2019). By contrast, biological systems need to update the synaptic weights after each experience, and I compare learning performance in such a setting. Figure 4.4:b demonstrates the classification error on the test set (i.e., test error) as training progresses on FashionMNIST (Xiao et al. 2017), where it is demonstrated that PC is more sample-efficient than backpropagation. Figure 4.4:c and d measure the sum and minimum of the test error as the training

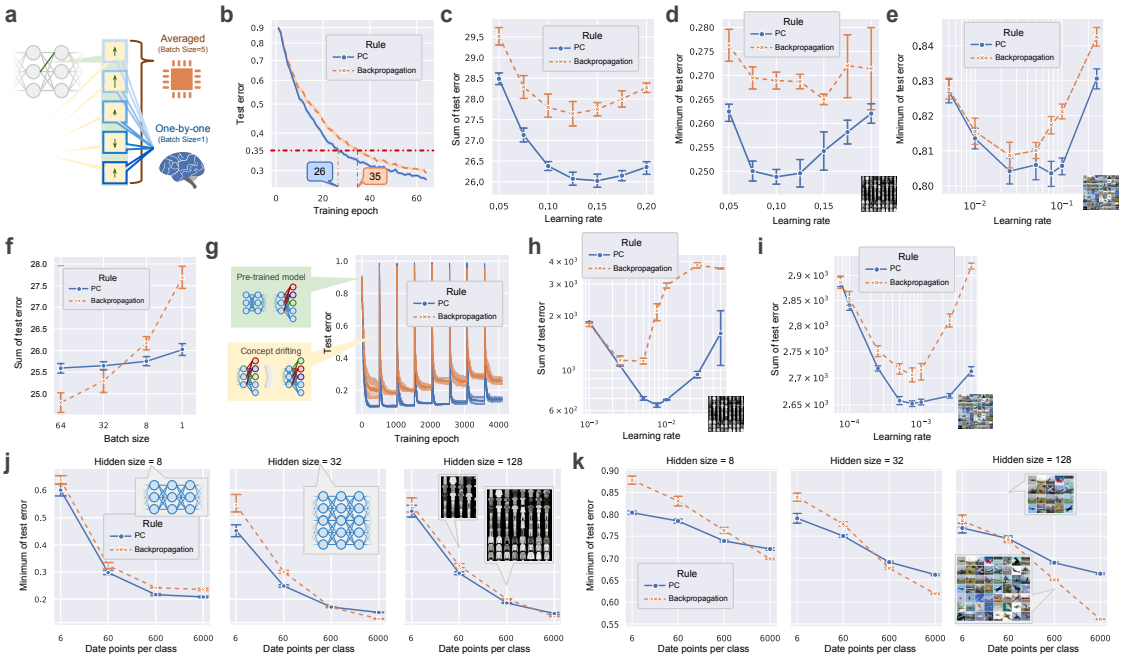


Figure 4.4: PC achieves superior performance over backpropagation on various learning situations faced by biological systems, confirming the theoretical advantage of prospective configuration with empirical evidence. These situations are: online learning (Fontenla-Romero et al. 2013) (a–e), learning in changing environments (Gama et al. 2014) (f–i), learning with a limited amount of training examples and limited size of architecture (j–k). Every learning setup is evaluated on both FashionMNIST (Xiao et al. 2017) (default) and CIFAR-10 (Krizhevsky and G. Hinton 2009) datasets. If the learning rate is not presented in a plot, the plot corresponds to the best learning rate for each rule.

progresses. Figure 4.4:e shows that the gap between PC and backpropagation opens as the batch size (the number of experiences updates are averaged over) decreases, indicating that PC learns better when the batch size gets smaller, as in biological settings. Backpropagation requires batch-averaging to get a "statistically good" modification, while PC produces weight changes without "side-effects" because of the mechanism of prospective configuration, and hence directly yields less erratic weight modifications (see Section 4.4.2).

Biological learning is often characterized by limited architecture size and data availability. Figure 4.4:f-g shows the test error when trained with different sizes of the networks and amounts of training examples (such subsets are randomly selected according to different seeds). In Figure 4.4:f-g, it is shown that PC outperforms backpropagation as the model is limited to smaller architectures or trained with fewer examples.

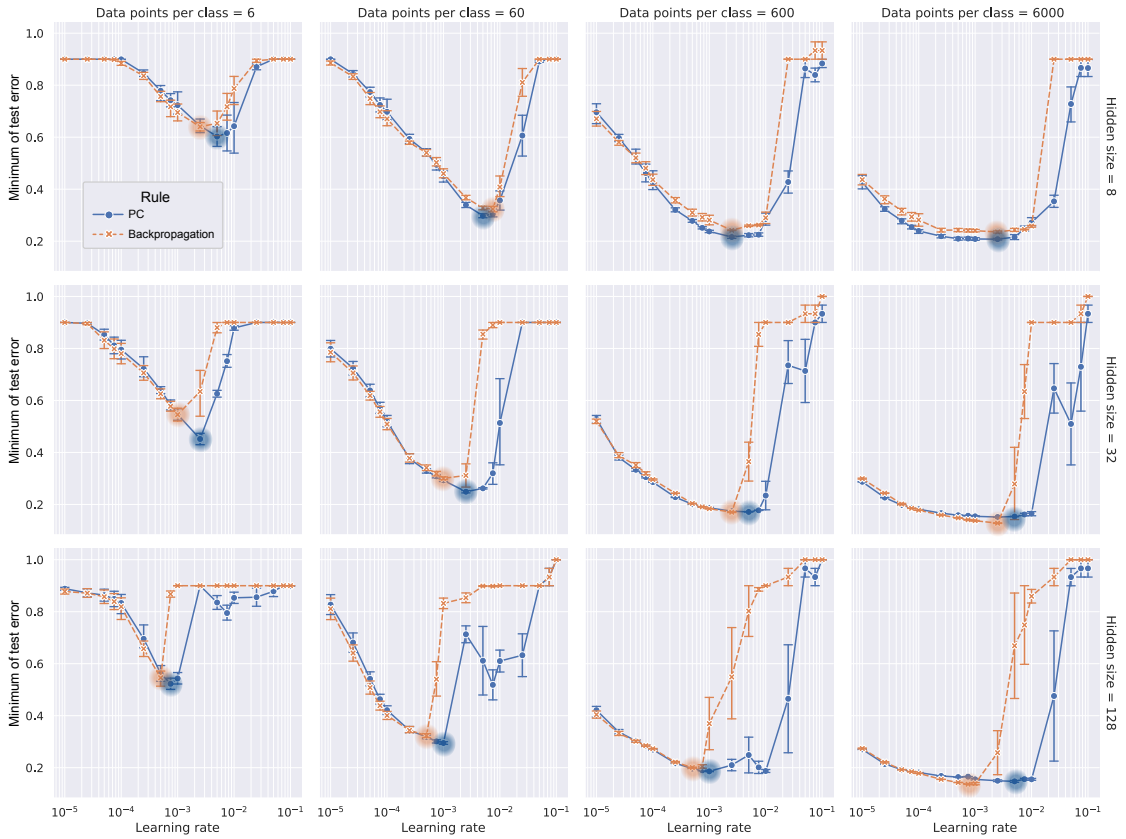


Figure 4.5: The search of learning rate. The highlighted points are picked and presented as Figure 4.4:f. In Fig. 4.4f, we demonstrate the learning performance of prospective configuration and backpropagation on FashionMNIST (Xiao et al. 2017), trained with different amounts of training examples and sizes of architecture. The result of each configuration there is obtained with the optimal learning rate, which was searched for independently for each configuration. Here, we expand the plots to show the search over learning rates. The highlighted points are selected to be reported in Fig. 4.4f.

The learning rate of each point in Figure 4.4:f-g is independently optimized. Specifically, in Figure 4.5, I present the search of learning rate. The highlighted points in Figure 4.5 are picked and presented as Figure 4.4:f.

Biological systems also need to rapidly adapt to changing environments. A common way to simulate this is "concept drifting (Gama et al. 2014)", where the mapping between the output neurons to the semantic meaning is shuffled every a period of time (Figure 4.4:h). Figure 4.4:h shows the test error during training with concept drifting. Figure 4.4:i summarizes the results of Figure 4.4:h. Figure 4.4:j repeats the same experiments of Figure 4.4:i on a natural-image dataset. Here, we can see PC learns better with concept drifting, indicating better adaptation

to changing environments.

Details of simulations

Default configuration All simulations are configured to be as close to common practice as possible, except for the investigated parameters (such as batch size, size of the model, and size of the dataset.), which are adjusted as specified in the simulations. By default, the neural network is composed of 4 layers and 32 hidden neurons in each hidden layer. The weights in the network are initialized from a normal distribution with mean 0 and standard deviation $\sqrt{\frac{2}{n^l+n^{l+1}}}$, where n^l and n^{l+1} are number of neurons of the layer before and after the synaptic weight, respectively. Such initialization is known as Xavier normal initialization (Glorot and Bengio 2010) and is adopted widely in the field. This initialization is also applied in the first paper describing PC for supervised classification (Whittington and Bogacz 2017). In all simulations, the networks are trained and tested on dataset FashionMNIST (Xiao et al. 2017), specifically, trained to do the classification of gray-scaled fashion item images into 10 categories such as Trouser, Pullover, and Dress. FashionMNIST is chosen because it is of moderate and appropriate difficulty for multi-layer non-linear deep neural networks investigated in the chapter, so that various comparisons are informative. A more difficult dataset CIFAR-10 (Krizhevsky, Nair, et al. 2010) of classifying colored natural images such as Cars, Birds, and Cats is also investigated in the experiments. A simpler dataset MNIST (LeCun 1998) of classifying handwritten digits is too simple because all models produce test errors less than 0.05, thus, excluded in this chapter. The dataset consists of 60000 training examples (i.e., training set) and 10000 test examples (i.e., test set). The activation function *sigmoid* was used. Mini-batches are sampled without replacement, i.e., in each epoch, if some examples are sampled as a mini-batch to train the network, they will not be sampled again in the same epoch. Thus, one epoch comprises presenting the entire training set, split over multiple mini-batches. The number of examples in a mini-batch, called the batch-size, is by default 32. I define one iteration of learning as updating weights based on a single mini-batch of training examples. If the

dataset contains only 1 example or the number of examples in a dataset is the same as the batch-size, iteration and epoch are the same (which is the case for some other simulations). At the end of each epoch, the model is tested on the test set and the classification error is recorded as the "test error" of this epoch. The neural network is trained for 64 epochs. Thus, it ends up with a series of 64 test errors. The mean or sum of the test errors over epochs is an indicator of how fast the model learns. The minimum of the test errors over epochs is an indicator of how well the model can learn, ignoring the possibility that the model may over-fit due to being trained for too long. Learning rates are searched independently for each configuration and each learning rule, and it is always guaranteed that the optimal learning rate for each learning rule in each setup lies within the range of the search conducted.

Figure 4.4:b-e Based on the default configuration, panels b-d use a batch size of 1 and panel e uses different batch sizes as specified by the x-axis. Each configuration is repeated with 10 different random seeds.

Figure 4.4:f-g Based on the default configuration, panels f-g use a different size of training set and size of the hidden layer as specified by the x-axis. Thus, 6000 datapoints per class are the default configuration of 60000 training examples (full training set), and the plot in the middle is the default of 32 hidden neurons in each hidden layer. Each configuration is repeated with 10 different random seeds.

Figure 4.4:h-j Based on the default configuration panels h-j show performance with "concept drifting" every 512 epochs and train for 4096 epochs in total, i.e., 8 times of concept drifting. Concept drifting (Gama et al. 2014; Žliobaitė 2010; Tsymbal 2004) refers to occasional sudden changes to class labels. During each such change, the first 5 output neurons (out of 10 output neurons as there are 10 classes) are selected, and the mapping from these 5 output neurons to the semantic meaning is shuffled. Each configuration is repeated with 10 different random seeds.

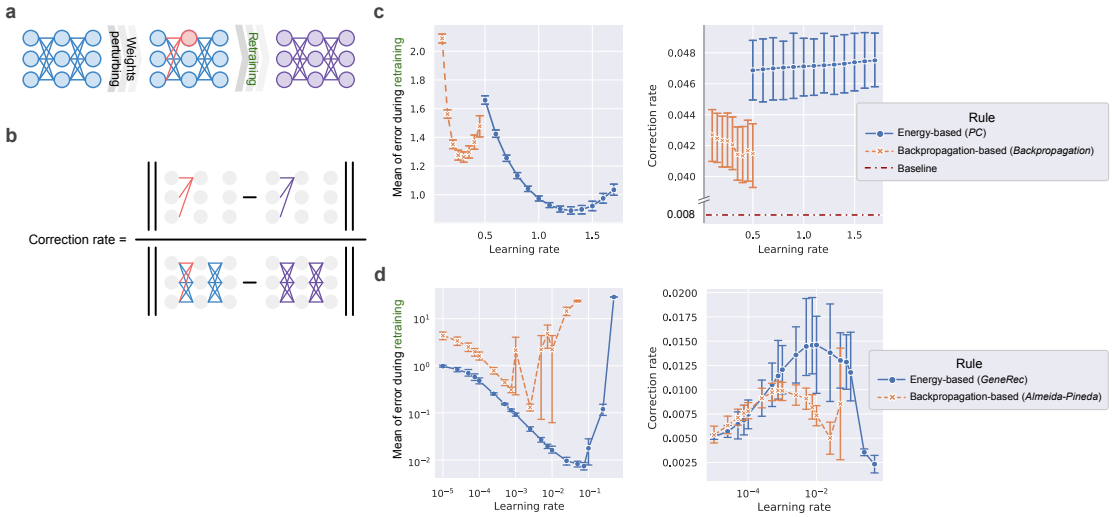


Figure 4.6: PC assigning error more optimally than backpropagation, confirming the theoretical advantage of prospective configuration with empirical evidence. A numerical experiment (panels a–b) verifies that energy-based models yield a accurate weight modification than backpropagation-based models (panels c–d). The following intuition can be provided for why the PC enables an accurate weight modification. In energy-based models, if more error is assigned to a neuron, this neuron will settle to a prospective activity that reduces the error. The prospective activity of this neuron is then propagated through the network, resulting in less error being assigned to other neurons, thus the error being assigned more accurately.

4.4.2 Comparison of accuracy and efficiency in assigning errors

In Section 4.2, one may anticipate that PC and other energy models assign error more optimally than backpropagation, because of the mechanism of prospective configuration: the neural activity settles to their prospective states. For example, if more error is assigned to a neuron, the neuron will settle to a prospective state which minimizes this error. The prospective state of the neuron is then propagated through the network, resulting in less error being assigned to other neurons, thus the error being assigned more optimally. Here, I demonstrate an experiment to empirically verify this prediction, and I demonstrate this for two popular energy-based models: PC and GeneRec (O'Reilly 1996). In other words, I demonstrate this prediction generalizes to different energy-based models because prospective configuration is a common principle they follow.

In Figure 4.6:a, I demonstrate a summary of the experimental procedure (details

will be given in a later part of the section): I take a pre-trained model (blue), randomly select a hidden neuron and perturb the synaptic weights connecting to this neuron (red), and retrain this model (purple). During retraining, an optimal learning rule should identify that the error in the output neurons is due to the perturbed weights, thus, (1) correct the error faster and (2) correct the perturbed weights more. I refer to the above two properties as speed and specificity.

In Figure 4.6:b, I measure the specificity by a quantity I call the correction rate. The correction rate is computed as the ratio of how much the perturbed weights are corrected compared to how much all the weights are corrected during retraining.

In Figure 4.6:c, I demonstrate a comparison between the energy-based model (particularly, PC) and the backpropagation-based model (particularly, backpropagation). The comparison is conducted to investigate both the speed and the specificity. The right plot is the mean of error during retraining, corresponding to measuring the speed: if a learning rule corrects the introduced error fast. The left plot is the correction rate defined in panel b, corresponding to measuring the specificity: if a learning rule corrects the perturbed weights much. The two plots verify that the energy-based model (particularly, PC), compared to the backpropagation-based model (particularly, backpropagation), (1) corrects such errors faster and (2) corrects the perturbed weights more. In this plot, there is an additional baseline, which is the number of perturbed weights divided by the number of all the weights. This baseline indicates the expected correction rate if a learning rule randomly assigns errors. This baseline highlights that the correction rates measured in both models are well beyond that of randomly assigning error.

In Figure 4.6:d, I demonstrate the same comparison as panel c, but for another energy-based model: GeneRec. GeneRec describes learning in recurrent networks, and backpropagation-based learning in this architecture is not described by standard backpropagation, but a modified version proposed by Almeida and Pineda (Almeida 1990; F. Pineda 1987; F. Pineda 1987; F. J. Pineda 1988) (thus called Almeida-Pineda algorithm), as reviewed in Section 2.2.2. In a word, PC and backpropagation are energy-based and backpropagation-based models working

in feed-forward network architecture, respectively; GeneRec and Almeida-Pineda are energy-based and backpropagation-based models working in recurrent network architecture, respectively. In this panel d, the same conclusion as that in panel c is drawn from the observation: energy-based model (particularly, GeneRec), compared to backpropagation-based model (particularly, Almeida-Pineda), (1) corrects the error faster and (2) corrects the perturbed weights more.

Details of simulations

I first pre-train the models to predict a randomly generated target pattern (32 entries) from a randomly generated input pattern (32 entries). The structure of the network is $32 \rightarrow 32 \rightarrow 32 \rightarrow 32$. I use random patterns instead of specific choices of tasks so that the conclusion is more general. The pre-training session is long enough to allow for close to perfect prediction of the random target pattern by the output neurons. Then, as stated in Figure 4.6:a, one neuron is randomly selected from the $(32 + 32)$ hidden neurons, and all weights connecting to this neuron are flipped (i.e., multiplied by negative one). The current weights of the network are recorded as $\boldsymbol{W}_{before\ re\text{-}train}$. The part of the current weights which are just flipped are recorded as $\boldsymbol{W}^{flipped}_{before\ re\text{-}train}$. The network is then re-trained on the same pattern to correct this error introduced by the flipped weights, for 64 iterations. At the end of each re-training iteration, the model makes a prediction on the target pattern, the error of which is recorded as the "error during re-train" of this re-training iteration. Here, the error is measured as half of the square of the difference between prediction and target pattern. At the end of the entire re-training session, the "errors during re-train" over a series of re-training iterations, are then averaged, producing the left plots of Figure 4.6:c-d. Current weights of the network are recorded as $\boldsymbol{W}_{after\ re\text{-}train}$. The part of the current weights which are flipped before the re-training session are recorded as $\boldsymbol{W}^{flipped}_{after\ re\text{-}train}$. Finally, "correction rate" is computed as $\frac{\|\boldsymbol{W}^{flipped}_{after\ re\text{-}train} - \boldsymbol{W}^{flipped}_{before\ re\text{-}train}\|}{\|\boldsymbol{W}_{after\ re\text{-}train} - \boldsymbol{W}_{before\ re\text{-}train}\|}$, producing the right plots of Figure 4.6:c-d. Each configuration is repeated with 10 different random seeds.

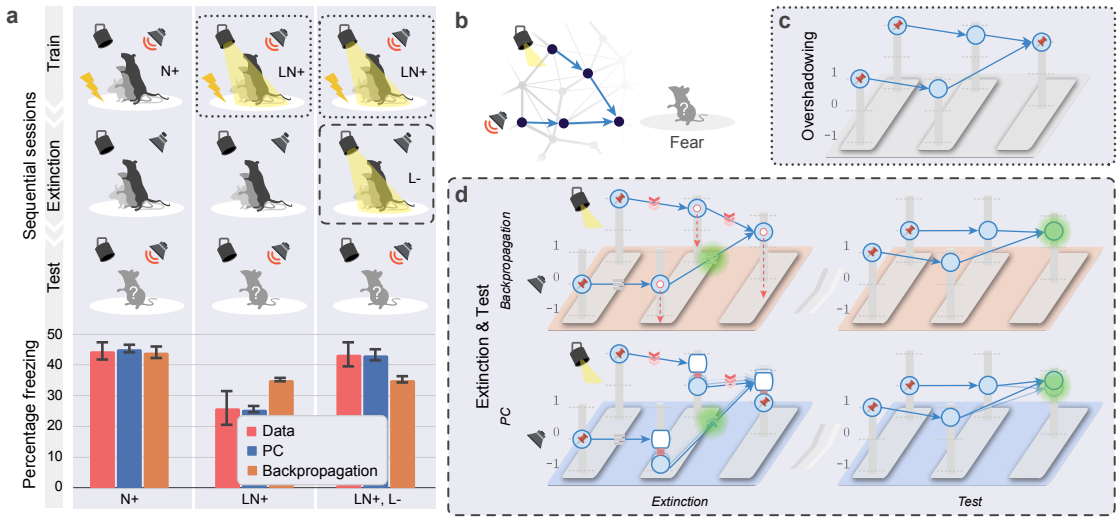


Figure 4.7: Established effects in animal learning, which cannot be explained by backpropagation, can be reproduced accurately by PC because of the mechanism of prospective configuration. Panel a top, the fear conditioning task, where rats are first trained to associate fear (electric shock) with noise and light; then in one of the groups, fear related to light is eliminated in extinction session; finally, the predicted fear (percentage of rats freezing) of noise is measures in test session. Panel a bottom, the predicted fear from networks trained with PC and backpropagation, compared against the fear (percentage freezing) measured in rats. Models are trained with the same procedure as rats. PC reproduces the key finding that eliminating the fear to light changes the fear to noise. Panel b, the architecture of simulated networks. Panel c–d, explaining the overshadowing, extinction and test sessions with the machien analog in Figure 4.2.

4.5 Explanation of established animal learning effects

Based on the superior performance and biological plausibility of PC due to the mechanism of prospective configuration, we may expect that this learning mechanism has been favored by evolution, and its signatures are visible in the learning behavior of animals. The distinction of prospective configuration from backpropagation is that error on one branch may spread to the others. Several previous studies have observed analogous effects in which learning about one stimulus affected memory for another stimulus (Kaufman and Bolles 1981; Matzel, Schachtman, et al. 1985; Matzel, Shuster, et al. 1987; Hallam et al. 1990; Miller, Esposito, et al. 1992). These effects were explained by assuming additional mechanisms (Miller and Schachtman 1985; Miller and Matzel 1988), but prospective configuration can explain them naturally with just the learning rule itself, e.g., with PC.

Figure 4.7 demonstrates one such learning effect (Kaufman and Bolles 1981), accurately reproduced by PC but not backpropagation. As shown in Figure 4.7:a, rats are divided into three groups, corresponding to three columns. Each group underwent three sessions sequentially, corresponding to the top three rows in Figure 4.7:a, namely, "Train", "Extinction", and "Test". In "Train", rats experienced an electric shock paired with different stimuli (noise alone "N+" or both light and noise "LN+") depending on the group, which is to associate fear with the presented stimuli. In "Extinction", for the last group, the light is presented but without the shock ("L-"), which is to eliminate the fear of light. In "Test", the noise is presented and the percentage of freezing of rats is measured for all groups, which indicated how much fear is associated with the noise. The bar plot in Figure 4.7:a reports the metrics for each group, both measured in the animal experiments (i.e., Data) and simulated by the two learning rules.

Figure 4.7:b demonstrates the neural architecture considered: both stimuli are processed by hidden neurons (i.e., intermediate neurons corresponding to visual and auditory cortices) and are then combined to produce the prediction of electric shock (i.e., fear).

Two effects are present in experimental data. First, comparing groups "N+" and "LN+" demonstrates the overshadowing effect: there is less fear of noise if it had been compounded with the light when paired with shock than if the noise alone had been paired with shock (in even simpler words, light overshadows noise in a conditioned fear experiment). This effect can be accounted for by the Rescorla-Wagner model (Rescorla 1972) which assumes that fear $\mathcal{F}$ is a weighted combination of light $\mathcal{L}$ and noise $\mathcal{N}$: $\mathcal{F} = w^{\mathcal{L}}\mathcal{L} + w^{\mathcal{N}}\mathcal{N}$, and these weights, $w^{\mathcal{L}}$ and $w^{\mathcal{N}}$, are adjusted during conditioning. The Rescorla-Wagner model would learn $w^{\mathcal{L}} = w^{\mathcal{N}} = 0.5$ for the "LN+" group resulting in reduced fear to noise alone ($w^{\mathcal{N}}\mathcal{N} = 0.5$). Figure 4.7:c shows the network that predicts fear from sensory receptors. Both PC and backpropagation would learn similar weights and thus reproduce overshadowing in a similar way as the Rescorla-Wagner model (Figure 4.7:c).

Second, comparing groups "LN+" and "LN+, L-" shows a striking effect: presenting the light during extinction increases fear response to the non-presented stimuli noise. As shown in Figure 4.7:d:top, the Rescorla-Wagner model and backpropagation cannot explain this, since the error cannot be backpropagated to a non-activated branch, i.e., a branch where no stimuli are presented. As shown in Figure 4.7:d:bottom, PC, however, can account for this with the mechanism of prospective configuration. Specifically, on the non-activated bottom branch (i.e., the branch corresponds to the auditory cortex), hidden neural activity decrease from zero to a small negative value (it may correspond to neural activity decreasing below baseline), in response to the output neural being clamped to the target pattern. Consequently, the weight connecting from the hidden neuron of the bottom branch to the output neuron is strengthened, even when no input stimuli are presented in this bottom branch.

4.5.1 Details of simulations

The structure of the network is as demonstrated in Figure 4.7:b. As mentioned before, the default configuration is to initialize the weights with the Xavier normal initialization (Glorot and Bengio 2010). In this simulation, unlike the default configuration, the network is initialized from randomly generated small weights before the training session. Specifically, the initial weights are randomly generated from a normal distribution of mean 0.1 and standard deviation 0.01. This is to simulate that the animal subjects have not built any association between stimulus and electric shock before the training session.

Presenting or not presenting the stimulus (noise, light, or shock) is encoded as 1 and 0, respectively. The network includes 2 input, 2 hidden, and 1 output neurons, as illustrated in Figure 4.7:b. The first and second input neurons are considered to be auditory and visual neurons, where their activity corresponds to perceiving light and noise, respectively. The output neuron is considered to encode the prediction of the electric shock. The training and extinction sessions are both simulated for 32 epochs with a learning rate of 0.01. Specifically, for simulating

training, the input neurons are set to the stimulus shown to a given group (i.e. $\boldsymbol{x}^1 = [1, 0]$ for "L+" group, while $\boldsymbol{x}^1 = [1, 1]$ for "LN+" groups), and the target is set to 1 . Subsequently, for simulating extinction the input is set to stimulus shown to a given group (i.e. $\boldsymbol{x}^1 = [0, 0]$ for "L+" and "NL+" groups, while $\boldsymbol{x}^1 = [1, 0]$ for "LN+,L-" group), and the target is set to 0.

In the test session, for both PC and backpropagation, the input pattern is set to $\boldsymbol{x}^1 = [0, 1]$, the model makes a prediction with the output neuron and the prediction is recorded. Since it is a prediction of the electric shock, it is considered to be an indicator of the percentage of freezing. I denote different groups of experiments with g, and the set of all groups $\mathcal{G}$. The percentage of freezing measured in (Kaufman and Bolles 1981) is s_g^{Data}. The activity of output neuron is x_g^{BP} for backpropagation and x_g^{PC} for PC. The activity of output neuron is scaled by a coefficient (a^{BP} for backpropagation and a^{PC} for PC) and a bias (b^{BP} for backpropagation and b^{PC} for PC). I called the scaled output neural activity the scaled prediction. The coefficient and bias are optimized for PC and backpropagation independently to minimize the difference between the scaled prediction and the percentage of freezing measured in animal experiments in (Kaufman and Bolles 1981) (i.e., Data in Figure 4.7:a):

$$a^{\text{BP}}, b^{\text{BP}} = \arg\min_{w,b} \sum_{g \in \mathcal{G}} \left(w x_g^{\text{BP}} + b - s_g^{\text{Data}} \right)^2 \tag{4.10}$$

$$a^{\text{PC}}, b^{\text{PC}} = \arg\min_{w,b} \sum_{g \in \mathcal{G}} \left(w x_g^{\text{PC}} + b - s_g^{\text{Data}} \right)^2 \tag{4.11}$$

Note that the coefficient and bias map the prediction of a learning rule to the actual measured data in animal experiments, thus, they should be optimized independently for each learning rule only, so that the scaled prediction reflects only the difference in learning rules. It is also a common practice to add these coefficients and bias when modeling behavioral data, and these coefficients and bias should be optimized independently for each learning rule only (Bogacz and J. D. Cohen 2004).

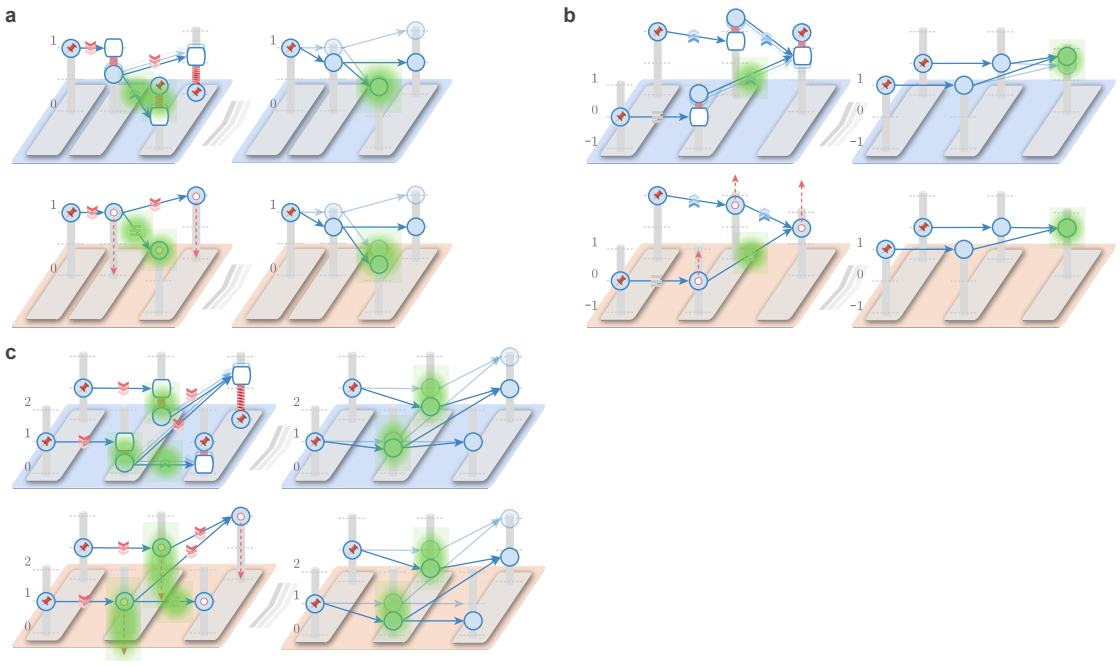


Figure 4.8: PC predicts distinct observation in physiological experiments from backpropagation because of the mechanism of prospective configuration: a comprehensive study. To provide examples of experimental predictions of PC, this figure highlight the different behaviour of the learning rules in simple network motifs, which are minimal networks displaying given behaviour. Two motifs in this figure have been already analysed earlier in the thesis, but there we focused on differences corresponding to experimentally observed effects, while in this figure we also highlight other qualitative differences that reveal a range of untested predictions of prospective configuration. Here, we consider PC with the energy machine in Figure 4.2. In each one of panels a-c, the top and bottom rows demonstrate the prediction of PC and backpropagation, respectively. The left column highlights the differences in the prediction errors during learning and the resulting weight update. The right column demonstrates the neuron activity before (transparent) and after (opaque) weight update. The differences between the rules are highlighted in green.

4.6 Physiological experiment predictions

Prospective configuration makes predictions on changes in neural activity during learning that are distinct from those of backpropagation. This can be used to further test if prospective configuration takes place in neural systems. Here I demonstrate the predictions of PC, which is one of the models that follow prospective configuration, however, a similar analysis can be applied to other energy-based models which also follow the principle of prospective configuration. Although some of these predictions were evident from earlier examples, in this section I list the predictions and differences systematically. To list the predictions and

Table 4.1: Differences of predictions between backpropagation and PC, summarized from Figure 4.8.

	Motif in Figure 4.8	Panel a	Panel b	Panel c
Error neurons	**PC**	**Positive** activity of the output error neuron of the bottom branch.	None.	**Different** negative activities of the hidden error neurons on two branches.
	Backpropagation	**Zero** activity of the output error neuron of the bottom branch.	None.	**Same** negative activities of the hidden error neurons on two branches.
Synaptic weights	**PC**	**Strengthened** connection from the hidden to the output neuron of the bottom branch.	**Strengthened** connection from the hidden neuron of the bottom branch to the output neuron.	**Strengthened** connection from the hidden to the output neuron (both of the bottom branch).
	Backpropagation	**Unchanged** connection from the hidden to the output neuron of the bottom branch.	**Unchanged** connection from the hidden neuron of the bottom branch to the output neuron.	**Unchanged** connection from the hidden to the output neuron (both of the bottom branch).
Value neurons	**PC**	**Higher** activity of the output value neuron of the bottom branch.	**Higher** activity of the output value neuron.	**Different** change of the activities of the hidden value neurons on two branches.
	Backpropagation	**Lower** activity of the output value neuron of the bottom branch. +	**Lower** activity of the output value neuron.	**Same** change of the activities of the hidden value neurons on two branches.

differences systematically, one would have to enumerate all possible configurations of physiological experiments, which is unrealistic. However, we can identify the minimal motifs, with which complex configurations can be constructed. I demonstrate the predictions and differences in these minimal motifs in this section.

As shown in Figure 4.8, different minimal motifs are investigated in the section, corresponding to each panel among panels a-c. In each panel, the top row demonstrates the prediction of PC, the bottom row demonstrates the prediction of backpropagation. The left column demonstrates the error spreading (i.e., credit assignment) and the consequential weight update. The right column demonstrates the neuron activities before (transparent) and after (opaque) weight update.

Note that there are two kinds of nodes in a network: error nodes and value nodes. Value nodes are commonly believed to map on the activities of neurons (i.e., value neurons). However, currently, it is not clear how the error nodes map on different parts of the neural circuits. Three key hypotheses have been proposed in the literature that error nodes map to:

- activities of separate error neurons (Rao and Ballard 1999; Bastos et al. 2012; Attinger et al. 2017);
- membrane potential of value neurons (Boerlin et al. 2013; Brendel et al. 2020).
- membrane potential in apical dendrites of value neurons (Sacramento et al. 2018; Whittington and Bogacz 2019);

Whether value and error nodes are easy to measure in practice also depends on which one of the above assumptions is correct. For example, if the above first assumption is correct: error nodes map to activities of error neurons, and value nodes map on the activities of value neurons, we can assume the error neurons are at rest if no supervised signal is provided. Then, value nodes mapped on the activities of value neurons can be measured if the subject makes a prediction without observing any supervised signal. With time the encoding of error in neural circuits is likely to be identified, so I make predictions for signals encoding both error and values, and to make these predictions concrete I assume that separate error and value neurons exist, in the following.

In Figure 4.8, the left columns present the predictions of error neurons, and also the predictions of the change of the weights; the right columns present the predictions of value neurons. Differences in the above predictions are highlighted in Figure 4.8. As can be seen from the highlighted parts in the figure, the key features of PC (distinct from those of backpropagation) for each motif are that:

- The error may spread to the branch where the prediction is correctly made (the motif in Figure 4.8:a);
- The error may cause weight change in the sensory regions associated with absent stimuli (the motif in Figure 4.8:b);
- The error of a hidden node depends on the proportion of the correctly predicted associated outputs (the motif in Figure 4.8:c).

Such differences are described in detail in Table 4.1.

Figure 4.1 and 4.2:i investigate the motif in Figure 4.8:a. Figure 4.7 investigates a similar motif as the one in Figure 4.8:b. The difference is that Figure 4.7 introduces a negative error while Figure 4.8:b introduces positive error on the same architecture. Interestingly, we can see introducing negative (Figure 4.7) or positive (Figure 4.8:b) error to the same architecture produce a similar prediction by PC.

Actually, for all the motifs in Figure 4.8, predictions may be generated either by creating a positive or negative prediction error in the output.

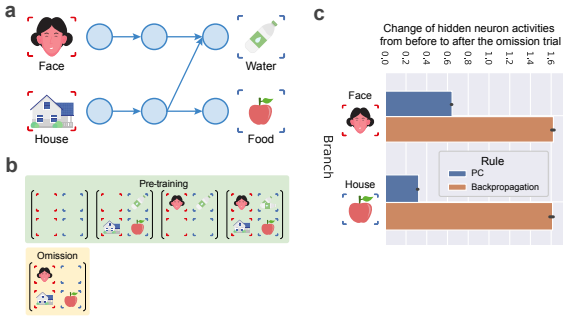


Figure 4.9: PC predicts distinct observation in physiological experiments from backpropagation because of the mechanism of prospective configuration: a case study. Here I demonstrate a striking difference in how PC and backpropagation assign error to hidden nodes. Namely, in PC, the error assigned to a hidden node is reduced if the node is also connected to correctly predicted outputs. This difference is illustrated in a motif (Figure 4.8:a), for which we illustrate behaviour of learning rules with the energy machine, and describe a sample experiment testing model predictions (panels a-b). Finally, we report the simulation results of the two learning rules (panel c), confirming that they indeed make distinct predictions for this motif.

Additionally, as long as the prediction differences are visible on the output value neurons in Table 4.1, the predictions are testable from behavioral experiments. For example, Figure 4.8:b makes a behavioral prediction (presenting light with stronger shock should also increase freezing for tone) that can be tested in a similar way as described in Figure 4.7. Testing this prediction would also validate our explanation of the experimental result in Figure 4.7.

In this section, I focus on investigating a new motif in Figure 4.8:c. Specifically, I first demonstrate how the two learning rules give distinct predictions in this motif with the energy machine (Figure 4.8:c). Then, I demonstrate the experiment procedure of testing this motif (Figure 4.9:a-b). Finally, I report the simulation results of the two learning rules on this motif with the proposed experiment procedure, confirming that they indeed make distinct predictions here.

As shown in Figure 4.8:c, in the motif I will be investigating here, two stimuli are presented and two predictions are made. One of the two stimuli (the top stimuli) contributes only to one of the two predictions (the top prediction); the other stimuli (the bottom stimuli) contribute to both predictions. With this motif, a negative error is introduced to the top prediction. Since it is the prediction both stimuli contribute to, we would expect the error to be assigned to hidden neurons on both

branches. Both learning rules do as expected, however, they assign errors differently. With the energy machine, we can see that PC (top row) allocates less error on the bottom hidden neuron than the top hidden neuron, because it sees that the bottom hidden neuron also contributes to another prediction that is correctly predicted. While backpropagation assigns the same error to the two hidden neurons.

Next, I describe the experiment that simulates the motif in Figure 4.8:c. Figure 4.9:a shows the network corresponding to the task. There are two stimuli (e.g., face and house) and two predictions (e.g., water and food). Each one of the two stimuli should excite an independent group of hidden neurons, since faces and houses each excite a particular area of the brain (Heekeren et al. 2004). Thus, it is feasible to measure the activity of these hidden neurons by measuring the activity of these particular areas. The pre-trained model should have the weight connection pattern as in the figure: the face only predicts water, and the house predicts both water and food. How to get this pre-trained model is described in the experimental procedure below.

Figure 4.9:b presents the experimental procedure. The experimental procedure consists of two stages of pre-training and omission. The participants should be shown with different outcomes of water and food paired with different stimuli of face and house, and brain activities are measured in face and house areas at specific moments. Specifically, the participants should first experience the pre-training stage: they should be shown with the four examples in Figure 4.9:b:green box for multiple times. After the pre-training stage, the participants should have been trained to develop the pre-trained model described in Figure 4.9:a. We wish to measure the response in the face and house areas after the pre-training stage. Thus, the participants are shown with face and house, and the brain activities are measured in the face and house areas as x_{face} and x_{house}, respectively. Next, the participants should experience the omission stage: they should be shown with the example in Figure 4.9:b:yellow box for only one trial. This will introduce a negative error in the prediction of water for only one trial. We now wish to measure the response in face and house areas after the omission stage. Thus, the participants

are shown with face and house, and the brain activities are measured in the face and house areas as x'_{face} and x'_{house}, respectively.

Figure 4.9:c presents the simulation results. Specifically, the horizontal axis shows the change of hidden neuron activities from before to after the single omission trial. The top row shows that of the face branch (i.e., $x_{face} - x'_{face}$); the bottom row shows that of the house branch (i.e., $x_{house} - x'_{house}$). Different colors represent results from different learning rules. PC predicts that the change of hidden neural activity from before to after the one omission trial on the two branches to be different (i.e., $x_{face} - x'_{face} \neq x_{house} - x'_{house}$), while backpropagation predicts that they should be the same (i.e., $x_{face} - x'_{face} = x_{house} - x'_{house}$). Below I describe the details of this simulation.

4.6.1 Details of simulations

The structure of the network is $2 \rightarrow 2 \rightarrow 2$ as demonstrated in Figure 4.9:a. No activation function is applied in this experiment (the network is purely linear). The network is initialized to the pre-trained connection pattern demonstrated in Figure 4.9:a as the focus of this simulation is not to pre-train the model. In the connection pattern of Figure 4.9:a, the visible weights are set to one, and other weights are set to zero. One can of course pre-train the network to develop such a pre-trained connection pattern with the four examples in Figure 4.9:b (in the green "Pre-training" box).

Presenting and not presenting a stimulus (face, house, water, or food) are encoded as 1 and 0, respectively. Presenting a reward of two bottles of water is encoded as 2. I first set both inputs to 1 and record the hidden neural activity of the two branches as x_{face} and x_{house} for the upper and lower branches, respectively. With the pre-trained model, the omission session trains on a pattern that involves an error in the prediction of water, for only one trail. Such pattern is demonstrated in Figure 4.9:b (in the yellow "Omission" box). After this one trial of omission, I set both inputs to 1 and record the hidden neural activity of the two branches for the second time as x'_{face} and x'_{house} for the upper and lower branches, respectively. The change of the hidden neuron activity from before to after the one trial is computed

and demonstrated in Figure 4.9:c. Specifically, the face and house branches on y-axis in Figure 4.9:c compute $x_{face} - x'_{face}$ and $x_{house} - x'_{house}$ as the value of the x-axis of the plot, respectively. The learning rate is fixed to 0.8. This is because the network is trained for just one trial and the purpose of the simulation is to show that PC produces different changes of hidden neural activities on different branches, while backpropagation produces the same, which is irrelevant to the choice of learning rate.

4.7 Discussion and related work

Much recent work has focused on understanding how biological neural networks could learn in a way similar to backpropagation (Sacramento et al. 2018; Grossberg 1987; Scellier and Bengio 2017; Whittington and Bogacz 2017; Lillicrap, Cownden, et al. 2016; Roelfsema and Ooyen 2005; Pozzi et al. 2018; Körding and König 2001; Richards, Lillicrap, et al. 2019; Payeur et al. 2021; Millidge et al. 2020a). Although many proposed models do not implement backpropagation exactly, they nevertheless try to approximate backpropagation and much emphasis is placed on how close this approximation is (Sacramento et al. 2018; Grossberg 1987; Scellier and Bengio 2017; Whittington and Bogacz 2017; Lillicrap, Cownden, et al. 2016; Roelfsema and Ooyen 2005; Pozzi et al. 2018; Whittington and Bogacz 2019; Millidge et al. 2020a). However, it has been questioned if it is possible for the brain to implement backpropagation exactly (Crick 1989; Grossberg 1987) and backpropagation may result in catastrophic interference of newly learned facts with stored information (McCloskey and N. J. Cohen 1989; McNaughton and O'Reilly 1995). Also, learning in the brain seems to be more efficient than backpropagation in several critical aspects: for example, the brain can learn with much fewer exposures (Tsividis et al. 2017). This raises the question of whether employing backpropagation to understand learning in the brain should be the main focus of the field. Besides, energy-based models (that includes PC) are often solidly biological-grounded, however, overlooked in favor of the backpropagation theory. This is mainly due to a lack of theoretical understanding or empirical evidence of whether or why energy-based models are superior to backpropagation.

In this chapter, I set out a new perspective that the brain instead solves credit assignment with a fundamentally different principle, which I refer as "prospective configuration", which, in contrast to backpropagation, is implemented naturally in energy-based models. I demonstrate with PC and other energy-based models both theoretically (Section 4.2) and empirically (Section 4.4) that the mechanism of prospective configuration reduces the interference, thus enabling more effective learning in various contexts faced by biological organisms, and reproduces animal learning effects that cannot be explained by backpropagation (Section 4.5). The advantage of prospective configuration, particularly PC, in learning problems with biological constraints demonstrated in this chapter suggests a great potential to apply PC in machine learning problems. An obstacle to employing PC is that they are computationally slow. However, I will demonstrate in the following Chapter 5 that the model can be adjusted by modifying weights after each step of relaxation and it then becomes comparably fast to backpropagation or theoretically even faster with appropriate implementation.

5

Predictive coding has advantages over backpropagation in machine learning problems

This chapter discusses my results that biologically plausible models may inspire alternatives to backpropagation. Particularly, I propose Parallel Predictive Coding (PPC), which is a variant of Predictive Coding Network (Rao and Ballard 1999; Whittington and Bogacz 2017; Buckley et al. 2017) (PC) that is (1) more computationally efficient theoretically and (2) likely to be more computationally efficient empirically when fully parallelized on suitable hardware, compared to backpropagation.

PC can perform supervised learning as reviewed in Section 2.2.1, has advantages over backpropagation as demonstrated in Chapter 4, and Z-PC is further equivalent to backpropagation as shown in Chapter 3. However, both PC and Z-PC suffer from critical drawbacks.

PC is orders of magnitude slower than backpropagation, as the necessary process, relaxation (moving neural activities according to a particular rule), needs to run for a large number of iterations until convergence before performing a single weight update. Also, the PC requires a control signal to detect such convergence and trigger the weight update. Z-PC demonstrates that the relaxation process does not have to converge before the weight update, as it produces a weight update

equivalent to that of backpropagation at the first few steps of relaxation. However, this algorithm requires an even more delicate control signal to trigger the weight update of specific layers at specific moments of relaxation. Nevertheless, PC and Z-PC possess an interesting property that computations of both relaxation and weight updates are local, thus can be parallelized across not only neurons but also layers (computations in backpropagation are believed to be parallelizable across neurons but not layers). However, the gain of this parallelization does not outmatch the burden of running relaxation until convergence (PC) or running relaxation for the first few steps (Z-PC).

Inspired by PC and Z-PC, I propose Parallel Predictive Coding (PPC) as a simple combination of both (or a continuous shift from Z-PC to PC). Specifically, in PPC, the relaxation process runs in parallel with the weight update, i.e., the weight update is not conducted after the convergence of relaxation or at very specific moments of relaxation but all the time in parallel with relaxation. Thus, PPC updates weights at the end of relaxation (as in PC), at the beginning of relaxation (as in Z-PC), and at all relaxation steps in between.

Though motivated by the previous algorithms, PPC decently solves their drawbacks and shows promising properties that theoretically make it more computationally efficient than backpropagation. Specifically, PPC requires no control signal to trigger the weight update, which runs from the start of relaxation and in parallel with relaxation since then. Hence, all the computations in PPC can be executed *simultaneously*, *locally*, and *autonomously*, not only across neurons but also across layers.

The main contributions of this chapter are briefly as follows:

- I propose PPC as a new biologically plausible learning algorithm and formally show that when training a network with L layers on a perfect parallel machine, the time complexity of PPC is $O(1)$, while that of backpropagation is $O(L)$.
- I give experimental evidence that when training on the perfect parallel machine, the training loss and test error decrease faster using PPC than backpropagation (Fig. 5.4, left and center).

- I show empirically that when training on CPUs, the above higher efficiency of PPC over backpropagation is preserved (Fig. 5.5). But this is not the case on GPUs for now, as the implementation requires further optimization.

- I show empirically that the final test accuracy (i.e., generalization quality) of PPC is similar to that of backpropagation on multiple image classification benchmarks (Table 5.2).

Overall, the proposed PPC is more computationally efficient than backpropagation by being easier to parallelize (both theoretically and empirically on CPUs), while achieving a similar generalization quality as backpropagation on several image classification tasks.

5.1 Temporal training dynamics

In this chapter, I will again be investigating what happens during the relaxation, so it is important to denote by t the time axis during relaxation. Thus, notations of PC is augmented with an additional subscript of t if a variable is changed during relaxation (as in Chapter 3):

$$\hat{x}^l_{i,t} = \sum_{j=1}^{n^{l-1}} w^{l-1}_{i,j} f\left(x^{l-1}_{j,t}\right) \quad \text{and} \quad \varepsilon^l_{i,t} = x^l_{i,t} - \hat{x}^l_{i,t} \tag{5.1}$$

$$E_t = \sum_{l=2}^{L+1} \sum_{i=1}^{n^l} \frac{1}{2} \left(\varepsilon^l_{i,t}\right)^2 = \sum_{l=2}^{L+1} \sum_{i=1}^{n^l} \frac{1}{2} \left(x^l_{i,t} - \hat{x}^l_{i,t}\right)^2 \tag{5.2}$$

$$x^l_{i,t+1} - x^l_{i,t} = \Delta x^l_{i,t} = \begin{cases} 0 & l = 1 \\ \gamma \cdot \left(-\varepsilon^l_{i,t} + f'\left(x^l_{i,t}\right) \sum_{k=1}^{n^{l+1}} \varepsilon^{l+1}_{k,t} w^l_{k,i}\right) & l \in \{2, \ldots, L\} \\ \gamma \cdot \left(-\varepsilon^l_{i,t}\right) & l = L+1 \text{ in prediction} \\ 0 & l = L+1 \text{ in learning} \end{cases} \tag{5.3}$$

$$\Delta w^l_{i,j} = -\alpha \cdot \partial E_t / \partial w^l_{i,j} = \alpha \cdot \varepsilon^{l+1}_{i,t} f\left(x^l_{j,t}\right) \tag{5.4}$$

As mentioned earlier, subscript t is added if a variable is changing during the relaxation, where such t denotes the time step during relaxation. In PC, $w^l_{i,j}$ is not changing during the relaxation, thus, it is listed without the subscript t. However, $w^l_{i,j}$ is changing during the relaxation for the proposed PPC in this Chapter, thus, Equation (5.4) will need to be modified for PPC, as will be demonstrated later.

Furthermore, to simplify the discussion, I assume that the duration of presenting the training pairs $\mathcal{T}$ is sufficiently long so that within this time the relaxation has already converged. Thus, in the rest of this chapter, I discuss only the situation in which the weights are updated at time $\mathcal{T}$ rather than at the time of relaxation convergence t_c. This is how normally PC is implemented in practice: instead of detecting the convergence relaxation moment t_c (which could be non-trivial), relaxation is run for a fixed but sufficiently long duration until the end of presenting the datapoint ($t = \mathcal{T}$), e.g., $\mathcal{T}$ = 100 to 200 in (Millidge et al. 2020a).

5.2 Efficiency and autonomy of PC, Z-PC, and backpropagation

To motivate PPC, in this section, I study the properties of the aforementioned baselines, i.e., PC, Z-PC, and backpropagation, from the perspective of efficiency and autonomy.

5.2.1 Efficiency

Efficiency over matrix multiplications I first study the complexity of one relaxation step. I consider the number of *matrix multiplications* (*MMs*) required because it is by far the most complex operation performed. One relaxation step requires $(2L-1)$ MMs: L for updating all the error nodes (i.e., Eq. (5.1)) and $(L-1)$ for updating all the value nodes (i.e., Eq. (5.3)). To complete a single update of all

Table 5.1: Number of operations (MMs or SMMs) per weight update of PC, Z-PC, backpropagation, and PPC. Note that one relaxation step requires $(2L-1)$ MMs or 2 SMMs.

	PC	Z-PC	Backpropagation	PPC
MMs	$(2L-1)\mathcal{T}$	$(2L-1)(L-1)$	$(2L-1)$	$(2L-1)$
SMMs	$2\mathcal{T}$	$2(L-1)$	$(2L-1)$	2

weights, PC and Z-PC run for $\mathcal{T}$ and $(L-1)$ relaxation steps, respectively. Thus, to complete one weight update, PC and Z-PC require $(2L-1)\mathcal{T}$ and $(2L-1)(L-1)$ MMs, respectively. We should also notice that backpropagation requires $(2L-1)$ MMs to complete a single weight update: L for the forward and $(L-1)$ for the backward pass. Such numbers are summarized in the first row of Table 5.1. According to this measure, backpropagation is the most computationally efficient algorithm, Z-PC ranks second, and PC third ($\mathcal{T}$ is normally much larger than L). However, this measure only considers the total number of MMs needed, without taking into account whether some of them can be performed in parallel, which could significantly reduce the time complexity. I now address this point.

Efficiency over simultaneous MMs The MMs performed during relaxation can be parallelized across layers. Particularly, computations in Eqs. (5.1) and (5.3) only depend on information in the current and adjacent layers. Thus, L MMs that update all the error nodes in all layers take the time of only one MM if properly parallelized. Similarly, in Eq. (5.3), $(L-1)$ MMs that update all the value nodes in all layers take the time of only one MM if properly parallelized. As a result, one relaxation step only takes the time of 2 MMs if properly parallelized. Hence, I now define a measure that takes into account whether some MMs can run in parallel, called *simultaneous MMs* (*SMMs*), and used to better define the time complexity of the studied algorithms. Thus, one relaxation step takes 2 SMMs; one weight update with PC and Z-PC takes $2\mathcal{T}$ and $2(L-1)$ SMMs, respectively. Backpropagation propagates information through the entire network, i.e., $\boldsymbol{\varepsilon}^l$ is re-solved recursively from $\boldsymbol{\varepsilon}^0$. Thus, no MM can be paralleled in backpropagation, i.e., one weight update with backpropagation takes $(2L-1)$ SMMs. These numbers are summarized in

the second row of Table 5.1. Overall, measured over SMMs, backpropagation, and Z-PC are equally computationally efficient and much more efficient than PC.

5.2.2 Autonomy

Both PC and Z-PC lack full autonomy, as an external control signal is always needed to trigger the update of the weights: PC waits for reaching $\mathcal{T}$ relaxation steps, while Z-PC updates the weights of specific layers at specific relaxation moments $t = L - l + 1$. Backpropagation is considered to be less autonomous than PC and Z-PC: a control signal is required to switch between transmitting forward signals and backward errors, and additional places to store the backward errors are required.

5.3 Parallel predictive coding

Algorithm 9: Learn with Parallel Predictive Coding (PPC)

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

Output: updated synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern

$\boldsymbol{x}^{L+1} = \boldsymbol{s}^{\text{target}}$; // Clamp output neurons to target pattern

for $l = 1;\ l < L;\ l = l + 1$ **do** // Initialize $\boldsymbol{x}$

  $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;

  $\boldsymbol{x}^{l+1} = \hat{\boldsymbol{x}}^{l+1}$;

end

for $t = 0;\ t < \mathcal{T};\ t = t + 1$ **do** // Relaxation and weight update

  for $l = 1;\ l < L + 1;\ l = l + 1$ **do** // Parallelized, thus takes one SMM

    $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;

    $\boldsymbol{\varepsilon}^{l+1} = \boldsymbol{x}^{l+1} - \hat{\boldsymbol{x}}^{l+1}$;

  end

  for $l = 2;\ l < L + 1;\ l = l + 1$ **do** // Parallelized, thus takes one SMM

    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\left(\boldsymbol{w}^l\right)^T \boldsymbol{\varepsilon}^{l+1}\right)\right)$;

    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;

  end

  for $l = 1;\ l < L + 1;\ l = l + 1$ **do** // Update weights

    $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1}\left(f\left(\boldsymbol{x}^l\right)\right)^T$;

    $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;

  end

end

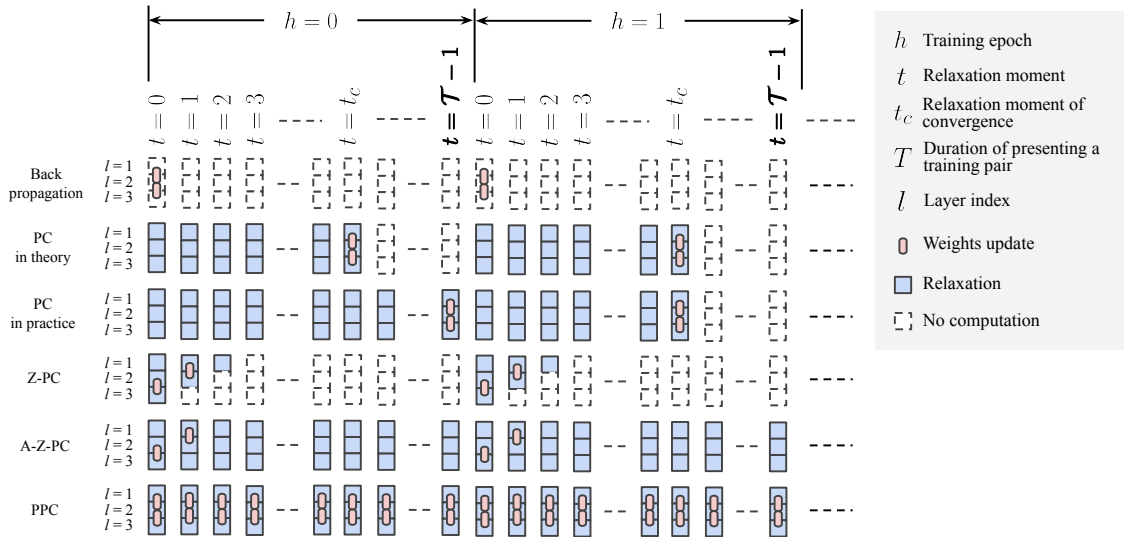


Figure 5.1: Comparison of the temporal training dynamics of backpropagation, PC, PC in practice, Z-PC, A-Z-PC, and PPC. I assume that we train the networks on a training pair from a dataset, which is presented for a period $\mathcal{T}$, before it is changed to another pair. Here, t is the time axis during relaxation, which always starts at $t = 0$. The squares represent nodes in one layer: blue squares indicate relaxation is conducted while empty squares indicate no relaxation is conducted. Pink rounded rectangles indicate when the connection weights are modified. Theoretically, PC (second row) conducts relaxation until it converges ($t = t_c$) and updates the weights, PC in practice (third row) conducts relaxation until the end of presenting the datapoint ($t = \mathcal{T}$) and updates the weights. This chapter inspects only PC in practice (third row). Z-PC and A-Z-PC (fourth and fifth rows) only update the weights at specific relaxation moments depending on which layer the weights belong to. PPC proposed in this chapter updates the weights at every relaxation moment t.

This section is dedicated to my proposed algorithm, called Parallel Predictive Coding (PPC), which continuously runs relaxation, as PC, and it performs weight updates before the relaxation has converged, as Z-PC. The key insights and motivations here are

- PC requires the settling of the relaxation of the network to achieve the correct $\varepsilon^l_{i,t}$ values for the weight updates.
- Z-PC in Chapter 3 proves that $\varepsilon^l_{i,t}$ in PC does not need to be obtained through full settling of the relaxation of the network to be informative for updating the weights. In fact, the first time $\varepsilon^l_{i,t}$ is updated from 0 to non-zero, it resembles the error term in backpropagation.

PPC is illustrated in the last row of Figure 5.1 and summarized in Algorithm 9, where the weights of all layers are updated simultaneously along with the running relaxation. Thus, all equations describing PC applies to PPC, except for Equation (5.4) which is modified to:

$$w^l_{i,j,t+1} - w^l_{i,j,t} = \Delta w^l_{i,j,t} = -\alpha \cdot \partial E_t / \partial w^l_{i,j,t} = \alpha \cdot \varepsilon^{l+1}_{i,t} f\left(x^l_{j,t}\right) \tag{5.5}$$

In simple words, PPC can be seen as a continuous shift from Z-PC to PC; the error used to update weights shifts from that of Z-PC (i.e., backpropagation) to that of PC, but is informative for updating weights at all time steps.

I now analyze two interesting properties of my proposed model: efficiency and autonomy. Particularly, I show that (1) PPC is, to my knowledge, the first learning algorithm for deep neural networks in which neurons work fully autonomously, and (2) PPC theoretically (on suitable parallel hardware) takes L times less time to complete a weight update compared to backpropagation, as well as being the most computationally efficient one among PC and Z-PC.

5.3.1 Autonomy

In terms of autonomy, PPC removes the control signal of triggering a weight update, which is required by the other models, Z-PC and PC. PPC does not require a control signal to switch between prediction and learning either. Because, as mentioned, the error nodes decay to zero if the output neuron is not clamped to the target pattern, thus, weights are not updated. To my knowledge, PPC is the first learning algorithm for deep neural networks that is fully automatic (i.e., no control signal is required).

5.3.2 Efficiency

I first consider the training on a single datapoint for multiple iterations, as this is the most basic setup of learning: a datapoint is presented for a duration of time, during which the machine learns continuously from it and tries to reduce the error to zero. I discuss the generalization to multiple datapoints afterwards.

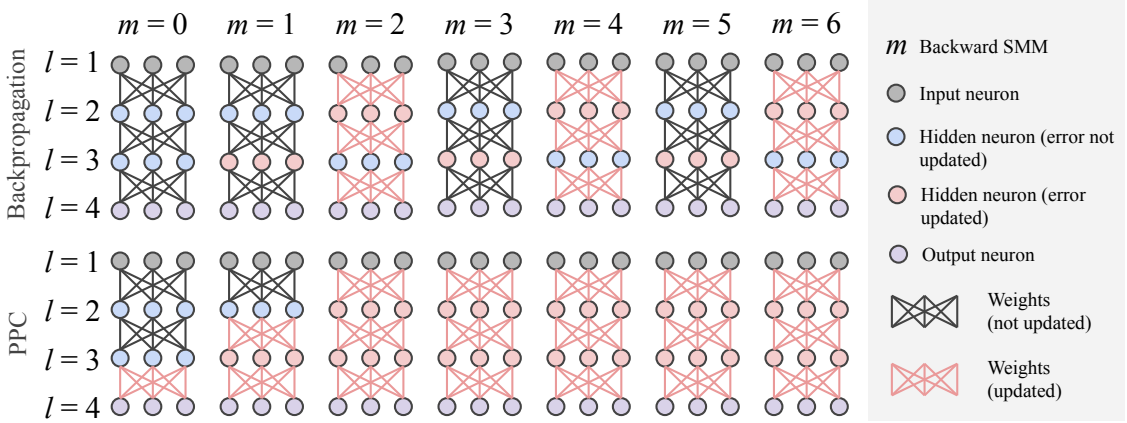


Figure 5.2: Graphical illustration of the efficiency over backward SMMs of backpropagation and PPC. PPC never clears the error (red neurons), while backpropagation clears it after every update. This allows PPC to perform 5 full and 2 partial updates of the weights in the first 6 SMMs. In the same time frame, backpropagation only performs 3 full updates. Note that the SMMs of forward passes during training are excluded for simplicity, w.l.o.g., as the insight from this example generalizes to the SMMs of the forward pass.

Efficiency with single datapoint

To complete one weight update, PPC requires one relaxation step, so $(2L-1)$ MMs or 2 SMMs. Thus, as shown in Table 5.1, compared to backpropagation, PPC takes L times fewer SMMs per weight update. Intuitively, this follows, as MMs in backpropagation have to be done sequentially along with layers (errors need to be solved recursively from the error on the output layer), while the ones in PPC can all be done in parallel across layers. Formally, to complete one update of all weights on a network with L layers, the time complexity over SMMs is $\mathcal{O}(2)$ and $\mathcal{O}(2L-1)$ (i.e., $\mathcal{O}(1)$ and $\mathcal{O}(L)$) for PPC and backpropagation, respectively.

To make the difference more visible and provide more insights, I explain this in detail with a sketch of this process on a small network in Figure 5.2, where the horizontal axis of m is the time step measured by SMMs. Note that here I only consider the SMMs for spreading errors, i.e., Eq. (5.3) for PPC. For backpropagation, it means that I only consider the SMMs for backpropagating the error. I ignore Eq. (5.1) for PPC and correspondingly the forward pass of backpropagation, since the insights after including them would remain the same. Thus, m in Figure 5.2 describes the number of backward SMMs performed.

As stated above, for backpropagation, backpropagating the error from one layer to an adjacent layer requires one MM; for PPC, one step of relaxation on one layer via Eq. (5.3) requires one MM. Backpropagation and PPC are presented in the first and second rows, respectively. Before both methods are able to update weights in all layers, they need two MMs for spreading the error through the network, i.e., a weights update of all layers occurs for the first time at $m = 2$ for both methods. After $m = 2$, backpropagation cleared all errors on all neurons, so at $m = 3$, backpropagation backpropagates the error from $l = 4$ to $l = 3$, and at $m = 4$, backpropagation backpropagates the error from $l = 3$ to $l = 2$ after which it can make an update of weights at all layers again for the second time. Note that the MM that backpropagates errors from $l = 3$ to $l = 2$ at $m = 4$ cannot be put at $m = 3$, as it requires the results of the MM at $m = 3$, i.e., it requires the error to be backpropagated to $l = 3$ from $l = 4$ at $m = 3$. However, this is different for PPC. After $m = 2$, PPC does not reset $x^l_{i,t}$ to $\hat{x}^l_{i,t}$, i.e., the error signals are still held in $\varepsilon^l_{i,t}$. At $m = 3$, PPC performs two MMs in parallel, corresponding to two relaxations steps at two layers, $l = 3$ and $l = 2$, updating $x^l_{i,t}$, and hence the error signals are held in $\varepsilon^l_{i,t}$ of these two layers. Note that the above two MMs of two relaxation steps can run in parallel and be put into a single m, as the relaxation in Eq. (5.3) requires only locally and immediately available information in current and adjacent layers. In this way, a weight update in PPC is able to be performed at every m ever since the very first few steps of m.

I further demonstrate the computational graphs of the same network as Figure 5.2 in Figure 5.3, for both backpropagation and PPC. For backpropagation, in Figure 5.3, we can see that $\boldsymbol{\varepsilon}^4$ is computed first at $m = 0$, i.e. as $\boldsymbol{\varepsilon}^4_0$, then $\boldsymbol{\varepsilon}^3$ needs to be computed at $m = 1$, i.e., as $\boldsymbol{\varepsilon}^3_1$, because it is computed from $\boldsymbol{\varepsilon}^4_0$. Analogous updates of error terms are performed at other steps m. Thus, $\boldsymbol{\varepsilon}^2$ can only be computed at $m = 2, 4$ and 6, i.e., as $\boldsymbol{\varepsilon}^2_2$, $\boldsymbol{\varepsilon}^2_4$ and $\boldsymbol{\varepsilon}^2_6$. For PPC, in contrast, updating $\boldsymbol{x}^2$ and $\boldsymbol{\varepsilon}^2$ rely only on variables in the current and adjacent layers: $\boldsymbol{\varepsilon}^2$ and $\boldsymbol{\varepsilon}^3$ in the previous moment, according to Eqs. (5.1) and (5.3). Thus, $\boldsymbol{x}^2$ and $\boldsymbol{\varepsilon}^2$ are updated at all m steps.

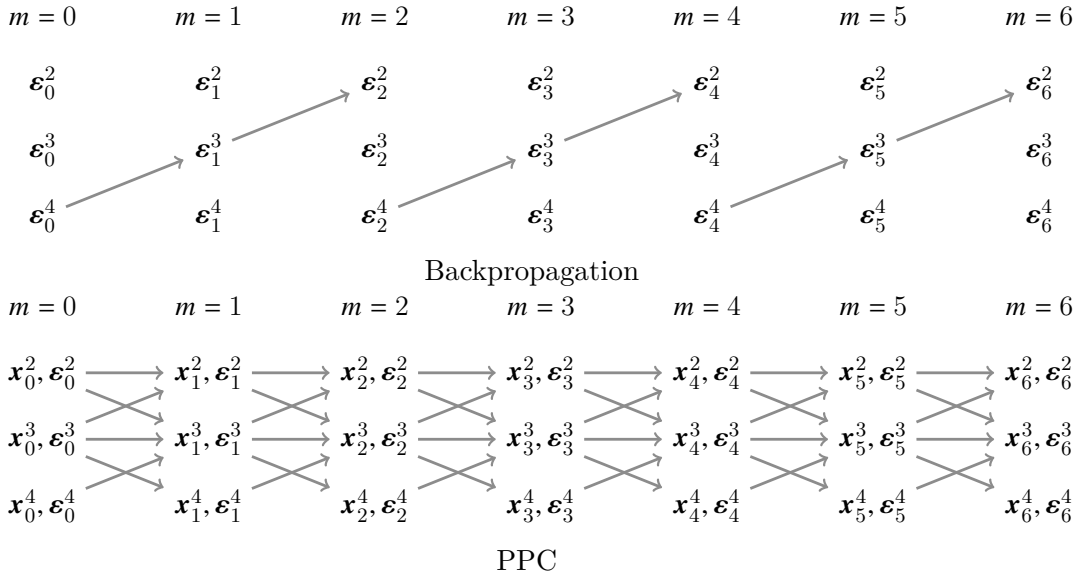


Figure 5.3: Computational graph of backpropagation and PPC.

Efficiency with multiple datapoints

Every time the algorithm switches to a new datapoint, PPC requires the first few SMMs, as shown in Figure 5.2, to spread the error over hidden neurons, before it can take 2 SMMs per weight update (one forward SMM for Eq. (5.1) and one backward SMM for Eq. (5.3)). Such overhead is introduced when learning on datasets with multiple datapoints, as PPC updates weights multiple times on a single datapoint. Thus, so that the advantage of PPC can be preserved in training with multiple datapoints, I train the whole dataset in a single batch, where there is no such switching to a new datapoint. Instead, I store in the memory $\boldsymbol{x}$, $\hat{\boldsymbol{x}}$, and $\boldsymbol{\varepsilon}$ corresponding to all datapoints and perform continuous computations on them. Specifically, $\boldsymbol{s}^{\text{in}}$, $\boldsymbol{s}^{\text{target}}$, $\boldsymbol{x}^l$, $\hat{\boldsymbol{x}}^l$, and $\boldsymbol{\varepsilon}^l$ are all tensors with the last dimension being the size of the dataset, so each entry along the last dimension of these variables corresponds to these variables specified to a datapoint. Taking $\boldsymbol{x}^l$ for example, assuming that layer l contains 32 neurons, and the size of the dataset is 6000, the tensor $\boldsymbol{x}^l$ is of size $[32, 1, 6000]$. Thus, $\boldsymbol{x}^l[:, :, 4]$ is $\boldsymbol{x}^l$ specified to the 4-th datapoint in the dataset. No lines in Algorithm 9 (except line 17, which will be mentioned later) require any changes to describe training in this setup. For example, following the above example, $\boldsymbol{x}^l$ is of shape $[32, 1, 6000]$, layer $l + 1$ contains 16 neurons,

thus, $\boldsymbol{w}^l$ is of shape $[16, 32]$, $\boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$ gives $\hat{\boldsymbol{x}}^{l+1}$ of shape $[16, 1, 6000]$. However, the update of weights $\boldsymbol{w}^l$ (line 17 in Algorithm 9) is changed to:

$$\Delta \boldsymbol{w}^l = \alpha \sum_d \boldsymbol{\varepsilon}^{l+1}[:, :, d] \left(f\left(\boldsymbol{x}^l[:, :, d]\right)\right)^T \tag{5.6}$$

This setup of training with a single batch is suitable for problems with small datasets. However, when dealing with large datasets, full batch training is not realistic. Thus, my claim of efficiency is restricted to the situation when the dataset can fit into a single batch. Dividing the dataset into multiple batches while still preserving the high efficiency of PPC is left as future work.

5.4 Experiments

My experiments investigate two aspects of PPC that are important in deep learning: time efficiency and generalization quality. To test the time efficiency of my method, I show that my model reduces the training loss and test error faster than backpropagation over SMMs as well as over the actual running time, implemented on CPUs and GPUs. To test the generalization quality, I compare the test accuracies of backpropagation, PC, and PPC on multiple architectures and datasets, showing that PPC achieves performances that are comparable, and sometimes better, than the ones obtained by backpropagation.

5.4.1 Efficiency

I used an MLP with 4 hidden layers and 128 hidden neurons trained on FashionMNIST (Xiao et al. 2017). The dataset is trained with a single batch of size 60000. Every network was trained on FashionMNIST with independently optimized learning rate from $\alpha \in \{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005\}$. The objective function of the search over learning rates is the mean of the test error during training. Thus, the reported results are from the learning rate with the minimal value of the mean of the test error during training. I additionally verified

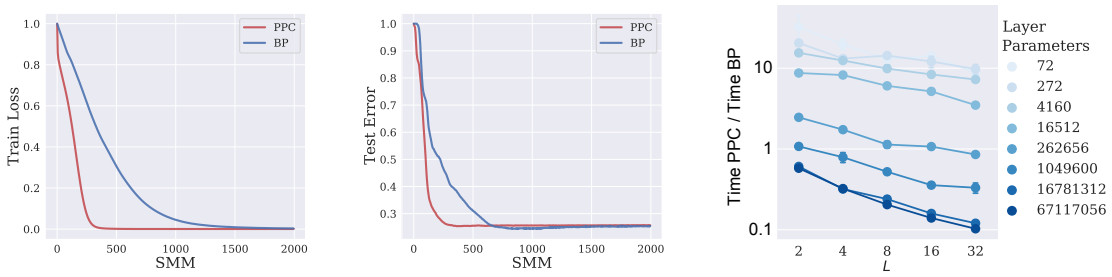


Figure 5.4: Training loss (left) and test error (center) of backpropagation (BP in the figure) and PPC on a 4-layer MLP trained on FashionMNIST over SMMs. Right: Ratio of the actual running time needed to perform a single weight update between backpropagation and PPC.

that the optimized learning rate for each algorithm at each setup lies within such a range of searched learning rates.

Efficiency over SMMs I trained an MLP and compared the training losses and the test errors as a function of SMMs. My results in Figure 5.4, left and center, show that the training loss and the test error of PPC decrease significantly faster compared to backpropagation over SMM, empirically confirming the theoretical efficiency of PPC. I additionally show the same plots as the above plots but for networks with different widths (number of hidden neurons) and depths (number of layers) in the Figures 5.6 and 5.7. A similar observation is made that training loss and the test error of PPC decrease significantly faster compared to backpropagation over SMMs.

Note that what I demonstrate above, over SMMs, is the theoretical upper bound of PPC's high efficiency. To demonstrate how much we can approximate this theoretical upper bound, I plot the ratio of the actual running time needed to perform a single weight update between backpropagation and PPC in Figure 5.4, right side. In this experiment, both backpropagation and PPC are implemented on CPUs, to parallelize MMs, any MMs that can be executed at the same time are sent to different CPUs, thus, measuring the running time over SMMs implemented on CPUs. There are in total 32 CPUs on the machine, thus, parallelizing 32 MMs (a network with 32 layers of weights) is the best I can do and it takes all the 32 CPUs. Parallelizing fewer MMs, of course, takes fewer CPUs. In Figure 5.4, every

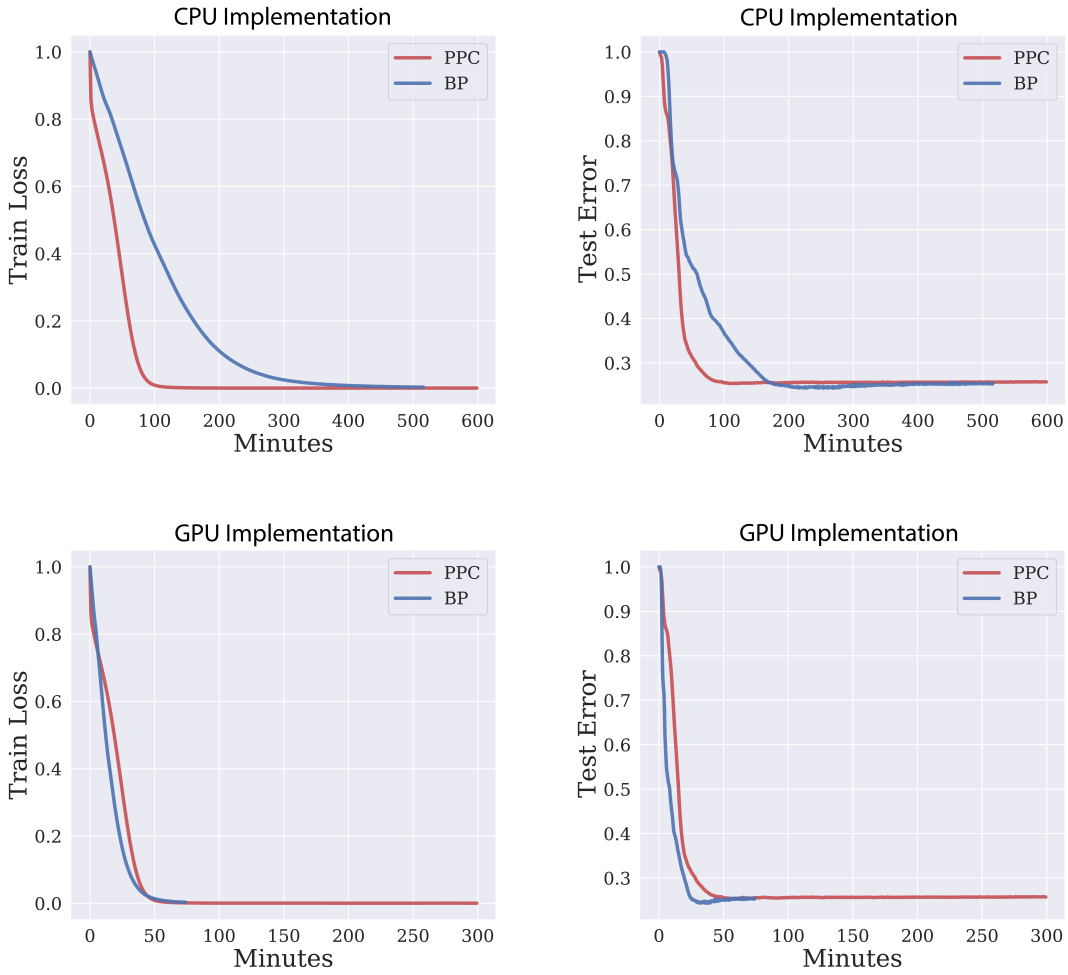


Figure 5.5: Top: Training loss and test error of PPC and backpropagation (BP in the figure) on a 4-layer MLP trained on FashionMNIST over actual running time implemented on CPUs. Bottom: The same as the top but over actual running time implemented on GPUs. I used Intel Xeon Silver 4110 CPUs running at 2.10GHz and a single Nvidia GeForce RTX 2080 GPU on an internal cluster.

dot is a model. If the dot lies below the horizontal line with label 1, one update of PPC is faster than one of backpropagation. In this plot, two observations can be made: PPC is faster than backpropagtion (1) as L increases, and (2) as the number of parameters per layer increases. The first observation empirically confirms the theoretical result that the time complexity over SMMs is $O(1)$ and $O(L)$ for PPC and backpropagation, respectively. The second observation indicates that when parallelising MMs communication overhead is created. For example, if completing each MM takes 1 second (each layer is small), we have 5 MMs to parallelize,

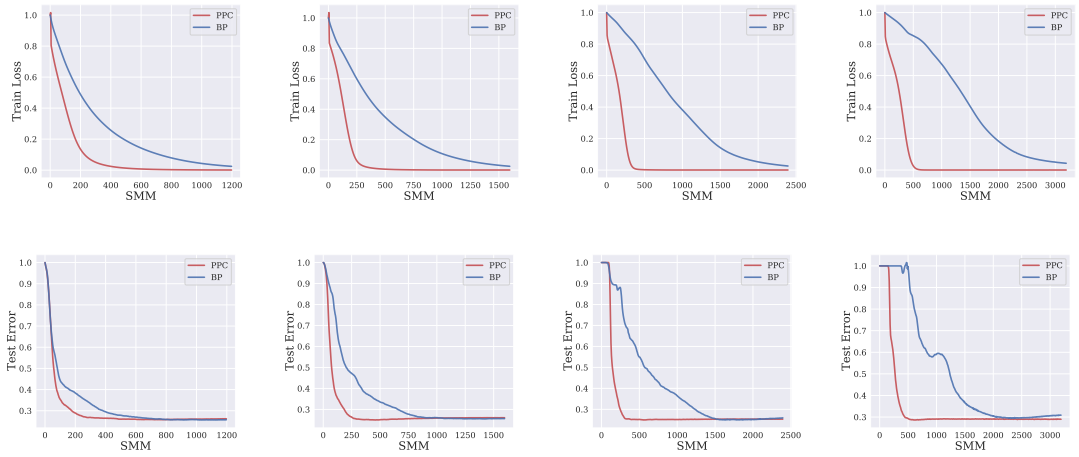


Figure 5.6: Top row: training loss of PPC and backpropagation (BP in the figure) on networks with 128 hidden neurons and different depths (from left to right: $L = 3, 4, 6, 8$) over SMM. Bottom row: corresponding test error of the top row.

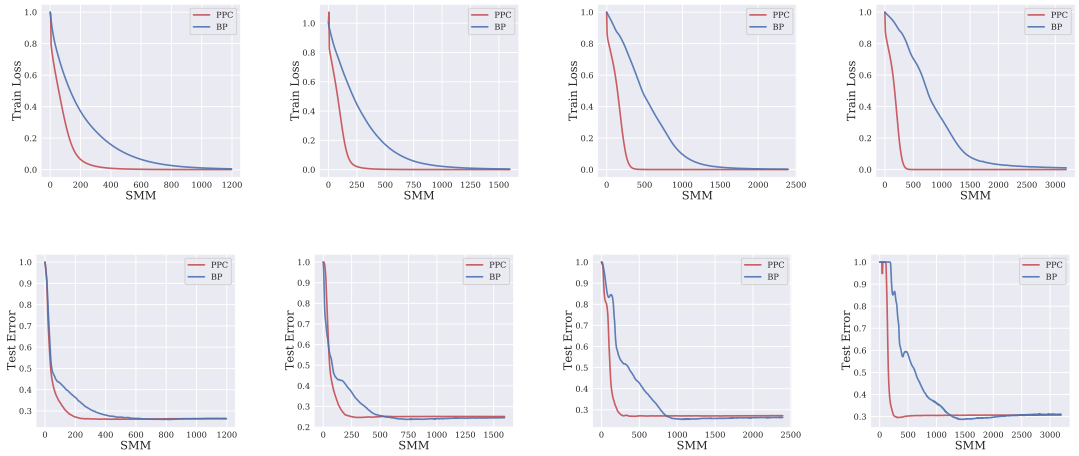


Figure 5.7: The same experiment as Figure 5.6, but trained on networks with 256 hidden neurons.

parallelising these MMs creates a fixed communication overhead of 10 seconds. If we don't parallelize these MMs (backpropagation) it takes $1 \times 5 = 5$ seconds in total; if we parallelize these MMs (PPC) it takes $1 + 10 = 11$ seconds in total — the ratio is $11/5 = 2.2$. On the other hand, completing each MM takes 10 seconds (each layer is large). If we don't parallelize these MMs (backpropagation) it takes $10 \times 5 = 50$ seconds in total; if we parallelize these MMs (PPC) it takes $10 + 10 = 20$ seconds in total — the ratio is $20/50 = 0.4$. In simple words, as the number of parameters per layer increases, overheads created by parallelising MMs become less dominant and

over-weighted by the benefits. Thus, the theoretical upper bound of PPC being L time more computationally efficient than backpropagation can be approximated as the number of parameters per layer ("Layer Parameters") and L increase.

Efficiency over actual running time In the last paragraph, I have presented the training loss and test error along SMMs, as well as shown that the actual running time can approximate the theoretical SMMs. Such approximation is not close to perfect and only works on CPU, thus is merely an initial step inspiring future engineering work. Still, one may be interested in the training loss and test error as a function of the actual running time, so they are provided in Figure 5.5 for implementations of SMMs on both CPUs and GPUs. SMMs on CPUs are implemented by sending parallelizable MMs to different CPU cores. SMMs on GPUs are implemented by sending parallelizable MMs to different threads (also called streams) on a single GPU. In Figure 5.5, we can see that PPC still shows higher efficiency than backpropagation on CPU implementations, but not on GPU implementations.

5.4.2 Generalization quality

I now demonstrate that PPC shows a similar level of generalization quality compared to backpropagation. I test the performance of PPC on different benchmarks.

Setup Since I focus on generalization quality in this section, all methods are run for enough iterations (until convergence of test accuracy). Particularly, I guarantee the convergence by terminating the training only if the test accuracy does not increase for continuous 20 iterations. I investigate image classification benchmarks using three different learning algorithms: PC, PPC, and backpropagation. Particularly, in my first experiment, I trained a fully connected network with 2 hidden layers and 64 hidden neurons per layer on the MNIST dataset (LeCun and Cortes 2010). Then, I trained a mid-size CNN with three convolutional layers with $64 - 128 - 64$ kernels followed by two fully connected layers on FashionMNIST, the Street View House Number (SVHN) dataset (Netzer et al. 2011), and CIFAR10 (Krizhevsky and G.

Table 5.2: Final accuracy (%) of backpropagation, PC, and PPC on different architectures trained with different datasets.

	Backpropagation	PC	PPC
MLP on MNIST	98.26% ± 0.12%	**98.55**% ± 0.14%	98.54% ± 0.86%
MLP on FashionMNIST	88.54% ± 0.64%	87.53% ± 0.75%	**89.13**% ± 0.86%
CNN on SVHN	95.35% ± 1.53%	94.53% ± 1.54%	**96.45**% ± 1.04%
CNN on CIFAR10	69.34% ± 0.54%	52.75% ± 0.64%	**72.54**% ± 0.93%
AlexNet on CIFAR10	**75.64**% ± 0.64%	64.63% ± 1.55%	72.42% ± 0.53%

Hinton 2009). Finally, I trained AlexNet (Krizhevsky, Sutskever, et al. 2012), a large-scale CNN, on CIFAR10. All experiments with MLPs are conducted with a single large batch, the size of which equals the size of the training set of the dataset. All experiments with CNNs are conducted with a batch size of 100. An experiment with AlexNet is conducted with a batch size of 60. The number of batches per dataset is thus determined by the size of the training set of the dataset divided by the specified batch size. To make sure that my results are not the consequence of a specific choice of hyperparameters, I performed a comprehensive grid search on hyperparameters, and reported the highest accuracy obtained, and the search is further made robust by averaging over 5 seeds. Particularly, I tested over 8 learning rates (from 0.000001 to 0.01), 4 values of weight decay $(0.0001, 0.001, 0.01, 0.1)$, and 3 values of the integration step γ $(0.1, 0.5, 1.0)$, and each combination of hyperparameters are evaluated with 5 seeds with mean and standard error reported. I additionally verified that the optimized value of each hyperparameter lies within the searched range of that hyperparameter. I used standard Pytorch initialization for the parameters.

Main Results According to the results in Table 5.2, PPC performs slightly better than backpropagation in all the small- and medium-size architectures. For the simplest setup (MNIST on a small MLP), PC outperforms all the other training methods, with PPC followed by a tiny margin (0.01%). However, PC fails to scale to more complex problems, where it gets outperformed by all the other training methods. The performance of PPC, on the other hand, is stable under changes in size, architecture, and dataset. Overall, my proposed algorithm reaches a similar accuracy as backpropagation on the considered tasks.

Table 5.3: Change of final accuracy (%) when increasing the width.

C	1	2	3	4	5	6	7	8	10	15	20
Backpropagation	67.92	71.23	71.65	72.64	73.35	73.71	74.19	74.51	74.62	75.08	75.51
PPC	**70.61**	**74.12**	**74.91**	**75.88**	**76.61**	**77.04**	**77.48**	**77.41**	**76.51**	**76.55**	**76.12**

Change of width Table 5.2 shows that PPC surprisingly performs better on a standard CNN than on AlexNet, an architecture with many more parameters and max-pooling layers. To investigate how PPC behaves when adding max-pooling layers and increasing the width (i.e., number of hidden neurons for MLPs and number of kernels for CNNs), I trained a CNN with three convolutional layers (8, 16, and 8 kernels) and max-poolings, followed by a fully connected layer (128 hidden neurons) on CIFAR10. Then, I increased the width of the network by multiplying a constant C by the number of hidden neurons for MLPs and the number of kernels for CNNs, and repeated the experiment. For example, $C = 3$ means the number of kernels for the three convolutional layers is changed to 24, 48, and 24, and the number of hidden neurons is changed to 384. The results in Table 5.3 show that PPC (i) outperforms backpropagation under every C, (ii) needs fewer parameters to obtain good results, but (iii) sees its performance decrease, once it has reached a specific C. This is in contrast to backpropagation, which is able to generalize well even when extremely over-parametrized (i.e., large C). This suggests that PPC is likely to be more efficient than backpropagation in terms of the number of parameters.

5.5 Discussion

In this chapter, I proposed PPC. PPC removes one of the requirements for which PC has been criticized. Such requirement is the external control of when to update weights. However, one should notice that when training on a new datapoint $(\boldsymbol{s}^{\text{in}}, \boldsymbol{s}^{\text{target}})$, $\boldsymbol{x}^l$ is initialized with a forward pass: line 3-6 in Algorithm 9. Therefore there is a control signal still remaining in the algorithm if not trained with a single batch, i.e., in response to a new datapoint provided a forward pass is required to initialize $\boldsymbol{x}^l$. This is done so that the error used to update weights is zero before the error computed from this datapoint reaches a layer. Such initialization with a

forward pass can be done through relaxation of the network, but it would result in prediction errors before the relaxation converges, which may cause unnecessary weight modifications in PPC. How much impact this issue has on the learning performance and how to solve it are left as future work.

Most importantly, PPC differs from backpropagation, as all the computations can be executed simultaneously, locally, and autonomously. Hence, PPC is more computationally efficient theoretically and likely to be more efficient empirically when fully parallelized on suitable hardware. I also demonstrate that PPC has a similar level of generalization quality to backpropagation on several image classification tasks. The empirical study of efficiency in this paper is limited by the current deep learning libraries and computing infrastructures, where multiple matrix multiplications cannot be conducted in parallel with little overhead and with acceleration on GPUs. Thus, engineering work on parallelizing multiple matrix multiplications is a potentially fruitful future work, releasing the full power of PPC as a more biologically plausible and more computationally efficient alternative to backpropagation.

6 Conclusion

6.1 Summary

This thesis presents several effective alternatives to backpropagation inspired by PC.

In Chapter 3, I propose two variants of PC, namely Z-PC and A-Z-PC, that are equivalent to backpropagation. Z-PC is theoretically interesting, as it shows that the relaxation in PC does not need to have converged to produce informative weight updates. However, the additional constraint of triggering weight update at relaxation moments specified to each layer seems to be too strict and unrealistic for biological systems. Nevertheless, A-Z-PC is then proposed to provide a possible biological implementation of the above constrain. Thus, A-Z-PC suggests an interesting possibility for backpropagation to be implemented in a biologically plausible way. However, I think the most intriguing insight of Z-PC is, as mentioned, that relaxation in PC does not need to be converged to produce informative weight updates. Such insight results in the proposal of PPC in Chapter 5.

In Chapter 4, I proposed that the standard PC implements a principle fundamentally different from backpropagation, called "prospective configuration". Prospective configuration is not a new specific algorithm I invent, but a common behavior I discovered in a family of established models, thus, called a principle. However, I discover this principle for the first time, demonstrate its advantages

over backpropagation both theoretically and empirically, and provide a thorough understanding of it. Specifically, I demonstrate that this new principle (1) is present in established models of information processing in the cortex (that includes not only PC but also other energy-based models), (2) enables more effective learning in situations faced by biological organisms than backpropagation, and (3) naturally accounts for surprising effects in animal learning that cannot be explained by backpropagation.

In Chapter 5, I proposed a variant of PC, called PPC. PPC is inspired by the observation of Z-PC in Chapter 3: the relaxation in PC does not need to be converged to produce informative weight updates. Thus, PPC is a combination of Z-PC and PC: weights and relaxation are conducted at the same time. Although PPC is a simple modification made based on PC and Z-PC, it immediately removes the undesirable properties of PC and Z-PC, and presents new appealing properties. Such appealing properties are that in PPC all the computations can be executed simultaneously, locally, and autonomously. Hence, PPC is more efficient theoretically and likely to be more efficient empirically when fully parallelized on suitable hardware, compared to backpropagation. I also demonstrate that PPC has a similar level of generalization quality as backpropagation on several image classification tasks in Chapter 5.

Interestingly, the prospective index (ϕ) proposed in Chapter 4 draws a spectrum that covers all the variants of PC investigated in this thesis:

- $\phi = 0$ This part of the spectrum covers backpropagation, Z-PC, and A-Z-PC.
- $0 < \phi < 1$ This part of the spectrum covers PPC, since it is a continuous shift from Z-PC to PC. It would be an interesting direction of future work to investigate how ϕ of PPC depends on model parameters (e.g., number of relaxation and weights update steps $\mathcal{T}$).
- ϕ close to 1. This part of the spectrum covers PC and PC should have a larger ϕ compared to PPC.
- $\phi = 1$. This part of the spectrum covers target-PC introduced in Chapter 4.

6.2 Future work

The thesis places much emphasis on the PC. Although it has been proposed how the original architecture of PC could be mapped onto cortical microcircuits (Bastos et al. 2012), some features of PC may seem biologically unrealistic (Millidge et al. 2020b). Nevertheless, it has been demonstrated that these features can be removed from PC and it still retains good performance (Millidge et al. 2020b; Whittington and Bogacz 2019; Sacramento et al. 2018). Thus, it needs to be investigated further how PC can be exactly implemented in cortical circuits.

The function ϕ proposed in A-Z-PC in Chapter 3 is computationally simple, as it only requires detecting a change in an input. It is possible that such a computation could be performed by biological neurons because some types of neurons are well-known to respond predominantly to changes in their input (Frazor et al. 2004). Verifying such possibilities would be interesting for future research.

The new prospective configuration principle proposed in Chapter 4 makes distinct predictions about the patterns of neural activity that should be observed during learning (Section 4.6), and testing them would be an interesting future research to confirm whether prospective configuration takes place in the brain.

Furthermore, the investigation of prospective configuration in Chapter 4 largely focuses on PC. Extending the investigation comprehensively to other energy-based models would make interesting future work. Also, when investigating the prospective index (the qualification of prospective configuration) of PC in Chapter 4, one should have an intuition that prospective configuration should generalize to a family of energy-based models with a lower bound of their energy function. Thus, it is interesting to develop a more general theory of energy-based models, targeting at answering the following questions:

- How to define an energy function so that the model learns?
- How to define an energy function so that the model has biologically plausible neural implementations?

- How to define an energy function so that the model follows the prospective configuration principle so as to process its nice properties?

- Does the answer to the last question relate to or overlap with the answer to the first question?

In a similar spirit as the above paragraph, the energy machine in Chapter 4 is of huge help in identifying all the distinct properties of PC, however, the energy machine is restricted to PC. It would be interesting to investigate what are the energy machines correspond to other energy-based models, which may help understand other energy-based models better or spot their unique advantages. For example, PPC corresponds to an energy machine similar to that of PC we introduced in Chapter 4, the only difference is that: moving the nodes and tuning the rods are conducted at the same time. This immediately reminds us that PPC is an asynchronous system, which means the possibility of being implemented easily with asynchronous hardware, such as analog hardware.

In Chapter 4, I also pointed out that the new principle implemented in PC reduces the interference between different pieces of learned information, while previous computational models proposed that the catastrophic interference (McCloskey and N. J. Cohen 1989; McNaughton and O'Reilly 1995) is avoided in the brain thanks to multiple learning systems. It is possible that both mechanisms operate in the brain to reduce the interference most effectively. Verifying this possibility would be promising for future research.

The empirical study of the efficiency of PPC in Chapter 5 is limited by the current deep learning libraries and computing infrastructures, where multiple matrix multiplications cannot be conducted in parallel with little overhead and with acceleration on GPUs. Thus, engineering work on parallelizing multiple matrix multiplications is a potentially fruitful future work, releasing the full power of PPC as a more biologically plausible and more efficient alternative to backpropagation. Specifically, as mentioned in Chapter 5, three problems need to be resolved from the perspective of engineering:

- Parallelizing matrix multiplications with little overhead;
- Storing and restoring neural activities with little overhead;
- Identifying better solutions than storing and restoring neural activities to deal with the problem of switching between datapoints.

The advantage of PC, PPC, and other energy-based models in performance demonstrated in this thesis suggest a great potential to apply them in machine learning tasks. It has been demonstrated that the speed of energy-based models can be greatly increased by analog hardware (Foroushani et al. 2020; Hertz et al. 1997), potentially even one order of magnitude faster than backpropagation. Therefore, I anticipate the thesis may inspire new machine learning hardware based on analog circuits. The parallelization problem in PPC should also be resolved naturally with such analog hardware. Thus, PPC implemented in such hardware is expected to

- Be order of magnitude faster than backpropagation, as a result of the parallelization and fast convergence of analog circuits;
- Preserve all the advanced properties of PC presented in Chapter 4, as a result of following the prospective configuration principle.

Overall, the thesis tries to offer alternative perspectives for understanding biological learning and designing artificial learning, inspired by PC. Although backpropagation is so successful and lies as the foundation of the huge modern machine learning research, it does not mean we should take for granted that this is the best credit assignment method and stop looking for possibilities of a new foundation.w

References

Abdelghani, MN, Timothy P Lillicrap, and Douglas B Tweed (2008). "Sensitivity derivatives for flexible sensorimotor learning". In: *Neural Computation* 20.8, pp. 2085–2111.

Ackley, David H, Geoffrey E Hinton, and Terrence J Sejnowski (1985). "A learning algorithm for Boltzmann machines". In: *Cognitive Science* 9.1, pp. 147–169.

Almeida, Luis B (1990). "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment". In: *Artificial Neural Networks: Concept Learning*. IEEE Computer Society Press, pp. 102–111.

Amit, Yali (2019). "Deep learning with asymmetric connections and Hebbian updates". In: *Frontiers in Computational Neuroscience* 13, p. 18.

Attinger, Alexander, Bo Wang, and Georg B Keller (2017). "Visuomotor coupling shapes the functional development of mouse visual cortex". In: *Cell* 169.7, pp. 1291–1302.

Auksztulewicz, Ryszard and Karl Friston (2016). "Repetition suppression and its contextual determinants in predictive coding". In: *Cortex* 80, pp. 125–140.

Bal, Matthias (2020). *An Energy-Based Perspective on Attention Mechanisms in Transformers*. https://mcbal.github.io/post/an-energy-based-perspective-on-attention-mechanisms-in-transformers.

Baldi, Pierre and Peter Sadowski (2016). "A theory of local learning, the learning channel, and the optimality of backpropagation". In: *Neural Networks* 83, pp. 51–74.

Banino, Andrea, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, and Joseph Modayil (2018). "Vector-based navigation using grid-like representations in artificial agents". In: *Nature* 557.7705, pp. 429–433.

Bastos, Andre M, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston (2012). "Canonical microcircuits for predictive coding". In: *Neuron* 76.4, pp. 695–711.

Bengio, Yoshua (2014). "How auto-encoders could provide credit assignment in deep networks via target propagation". In: *arXiv preprint arXiv:1407.7906*.

Bengio, Yoshua and Asja Fischer (2015). "Early inference in energy-based models approximates back-propagation". In: *arXiv preprint arXiv:1510.02777*.

Bengio, Yoshua, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin (2015). "Towards biologically plausible deep learning". In: *arXiv preprint arXiv:1502.04156*.

Bengio, Yoshua, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu (2015). "STDP as presynaptic activity times rate of change of postsynaptic activity". In: *arXiv preprint arXiv:1509.05936*.

Berner, Christopher, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, and Chris Hesse (2019). "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680*.

Boerlin, Martin, Christian K Machens, and Sophie Denève (2013). "Predictive coding of dynamical variables in balanced spiking networks". In: *PLoS Computational Biology* 9.11, e1003258.

Bogacz, Rafal (2017). "A tutorial on the free-energy framework for modelling perception and learning". In: *Journal of Mathematical Psychology* 76, pp. 198–211.

Bogacz, Rafal and Jonathan D Cohen (2004). "Parameterization of connectionist models". In: *Behavior Research Methods, Instruments, & Computers* 36.4, pp. 732–741.

Bottou, Léon, Frank E Curtis, and Jorge Nocedal (2018). "Optimization methods for large-scale machine learning". In: *Siam Review* 60.2, pp. 223–311.

Brendel, Wieland, Ralph Bourdoukan, Pietro Vertechi, Christian K Machens, and Sophie Denéve (2020). "Learning to represent signals spike by spike". In: *PLoS Computational Biology* 16.3, e1007692.

Buckley, Christopher L, Chang Sub Kim, Simon McGregor, and Anil K Seth (2017). "The free energy principle for action and perception: A mathematical review". In: *Journal of Mathematical Psychology* 81, pp. 55–79.

Cadieu, Charles F, Ha Hong, Daniel LK Yamins, Nicolas Pinto, Diego Ardila, Ethan A Solomon, Najib J Majaj, and James J DiCarlo (2014). "Deep neural networks rival the representation of primate IT cortex for core visual object recognition". In: *PLoS Computational Biology* 10.12, e1003963.

Crick, Francis (1989). "The recent excitement about neural networks". In: *Nature* 337.6203, pp. 129–132.

Demircigil, Mete, Judith Heusel, Matthias Löwe, Sven Upgang, and Franck Vermet (2017). "On a model of associative memory with huge storage capacity". In: *Journal of Statistical Physics* 168.2, pp. 288–299.

Flower, Barry and Marwan Jabri (1993). "Summed weight neuron perturbation: An $O(N)$ improvement over weight perturbation". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 212–219.

Fontenla-Romero, Óscar, Bertha Guijarro-Berdiñas, David Martinez-Rego, Beatriz Pérez-Sánchez, and Diego Peteiro-Barral (2013). "Online machine learning". In: *Efficiency and Scalability Methods for Computational Intellect*. IGI Global, pp. 27–54.

Foroushani, Armin Najarpour, Hussein Assaf, Fereidoon Hashemi Noshahr, Yvon Savaria, and Mohamad Sawan (2020). "Analog circuits to accelerate the relaxation process in the equilibrium propagation algorithm". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 1–5.

Frazor, Robert A, Duane G Albrecht, Wilson S Geisler, and Alison M Crane (2004). "Visual cortex neurons of monkeys and cats: Temporal dynamics of the spatial frequency response function". In: *Journal of Neurophysiology* 91.6, pp. 2607–2627.

Friston, Karl (2005). "A theory of cortical responses". In: *Philosophical Transactions of the Royal Society B: Biological sciences* 360.1456, pp. 815–836.

— (2010). "The free-energy principle: A unified brain theory?" In: *Nature Reviews Neuroscience* 11.2, pp. 127–138.

— (2018). "Does predictive coding have a future?" In: *Nature Neuroscience* 21.8, pp. 1019–1021.

Gama, João, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia (2014). "A survey on concept drift adaptation". In: *ACM Computing Surveys (CSUR)* 46.4, pp. 1–37.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, pp. 249–256.

Golovin, Daniel, John Karro, Greg Kochanski, Chansoo Lee, and Xingyou Song (2019). "Gradientless Descent: High-Dimensional Zeroth-Order Optimization". In: *arXiv preprint arXiv:1911.06317*.

Grossberg, Stephen (1982). "How does a brain build a cognitive code?" In: *Studies of Mind and Brain*. Springer, pp. 1–52.

— (1987). "Competitive learning: From interactive activation to adaptive resonance". In: *Cognitive Science* 11.1, pp. 23–63.

Hallam, Steve C, Louis D Matzel, Josh S Sloat, and Ralph R Miller (1990). "Excitation and inhibition as a function of posttraining extinction of the excitatory cue used in Pavlovian inhibition training". In: *Learning and Motivation* 21.1, pp. 59–84.

Hannun, Awni, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, and Adam Coates (2014). "Deep speech: Scaling up end-to-end speech recognition". In: *arXiv preprint arXiv:1412.5567*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Heekeren, Hauke R, Sean Marrett, Peter A Bandettini, and Leslie G Ungerleider (2004). "A general mechanism for perceptual decision-making in the human brain". In: *Nature* 431.7010, pp. 859–862.

Hertz, John, Anders Krogh, Benny Lautrup, and Torsten Lehmann (1997). "Nonlinear backpropagation: Doing backpropagation without derivatives of the activation function". In: *IEEE Transactions on Neural Networks* 8.6, pp. 1321–1327.

Hinton, Geoffrey E, Peter Dayan, Brendan J Frey, and Radford M Neal (1995). "The" wake-sleep" algorithm for unsupervised neural networks". In: *Science* 268.5214, pp. 1158–1161.

Hohwy, Jakob, Andreas Roepstorff, and Karl Friston (2008). "Predictive coding explains binocular rivalry: An epistemological review". In: *Cognition* 108.3, pp. 687–701.

Hopfield, John J (1982). "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the National Academy of Sciences* 79.8, pp. 2554–2558.

— (1984). "Neurons with graded response have collective computational properties like those of two-state neurons". In: *Proceedings of the National Academy of Sciences* 81.10, pp. 3088–3092.

Huang, Yanping and Rajesh PN Rao (2011). "Predictive coding". In: *Wiley Interdisciplinary Reviews: Cognitive Science* 2.5, pp. 580–593.

Igelnik, Boris (2013). "Efficiency and scalability methods for computational intellect". In:

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *Proceedings of the International Conference on Machine Learning (ICML)*.

Jia, Xianyan, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, and Liwei Yu (2018). "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes". In: *arXiv preprint arXiv:1807.11205*.

Kaufman, Mark A and Robert C Bolles (1981). "A nonassociative aspect of overshadowing". In: *Bulletin of the Psychonomic Society* 18.6, pp. 318–320.

Kell, Alexander JE, Daniel LK Yamins, Erica N Shook, Sam V Norman-Haignere, and Josh H McDermott (2018). "A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy". In: *Neuron* 98.3, pp. 630–644.

Khaligh-Razavi, Seyed-Mahdi and Nikolaus Kriegeskorte (2014). "Deep supervised, but not unsupervised, models may explain IT cortical representation". In: *PLoS Computational Biology* 10.11.

Kiefer, Jack and Jacob Wolfowitz (1952). "Stochastic estimation of the maximum of a regression function". In: *The Annals of Mathematical Statistics*, pp. 462–466.

Körding, Konrad P and Peter König (2000). "Learning with two sites of synaptic integration". In: *Network: Computation in Neural Systems* 11.1, pp. 25–39.

— (2001). "Supervised and unsupervised learning with two sites of synaptic integration". In: *Journal of Computational Neuroscience* 11.3, pp. 207–215.

Krizhevsky, Alex and Geoffrey Hinton (2009). "Learning multiple layers of features from tiny images". In: *Report*.

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton (2010). "CIFAR-10". In: *Canadian Institute for Advanced Research*. Vol. 5, p. 4.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 25, pp. 1097–1105.

Krotov, Dmitry and John J Hopfield (2016). "Dense associative memory for pattern recognition". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 29, pp. 1172–1180.

Lansdell, Benjamin James, Prashanth Prakash, and Konrad Paul Kording (2019). "Learning to solve the credit assignment problem". In: *arXiv preprint arXiv:1906.00889*.

LeCun, Yann (1998). "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/*.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: *The MNIST Database*. URL: http://yann.lecun.com/exdb/mnist/.

Lee, Dong-Hyun, Saizheng Zhang, Asja Fischer, and Yoshua Bengio (2015). "Difference target propagation". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 498–515.

Liao, Qianli, Joel Z Leibo, and Tomaso Poggio (2016). "How important is weight symmetry in backpropagation?" In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Lillicrap, Timothy P, Daniel Cownden, Douglas B Tweed, and Colin J Akerman (2016). "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nature Communications* 7.1, pp. 1–10.

Lillicrap, Timothy P, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton (2020). "Backpropagation and the brain". In: *Nature Reviews Neuroscience* 21.6, pp. 335–346.

Lillicrap, Timothy P and Stephen H Scott (2013). "Preference distributions of primary motor cortex neurons reflect control solutions optimized for limb biomechanics". In: *Neuron* 77.1, pp. 168–179.

Matzel, Louis D, Todd R Schachtman, and Ralph R Miller (1985). "Recovery of an overshadowed association achieved by extinction of the overshadowing stimulus". In: *Learning and Motivation* 16.4, pp. 398–412.

Matzel, Louis D, Karl Shuster, and Ralph R Miller (1987). "Covariation in conditioned response strength between stimuli trained in compound". In: *Animal Learning & Behavior* 15.4, pp. 439–447.

Mazzoni, Pietro, Richard A Andersen, and Michael I Jordan (1991). "A more biologically plausible learning rule for neural networks." In: *Proceedings of the National Academy of Sciences* 88.10, pp. 4433–4437.

McCloskey, Michael and Neal J Cohen (1989). "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of Learning and Motivation*. Vol. 24. Elsevier, pp. 109–165.

McNaughton, Bruce L and Randall C O'Reilly (1995). "Why There Are Complementary Learning Systems in the Hippocampus and Neocortex: Insights From the Successes and Failures of". In: *Psychological Review* 102.3, pp. 419–457.

Miller, Ralph R, James J Esposito, and Nicholas J Grahame (1992). "Overshadowing-like effects between potential comparator stimuli: Covariation in comparator roles of context and punctate excitor used in inhibitory training as a function of excitor salience". In: *Learning and Motivation* 23.1, pp. 1–26.

Miller, Ralph R and Louis D Matzel (1988). "The comparator hypothesis: A response rule for the expression of associations". In: *Psychology of Learning and Motivation*. Vol. 22. Elsevier, pp. 51–92.

Miller, Ralph R and Todd R Schachtman (1985). "The several roles of context at the time of retrieval". In: *Context and Learning*, pp. 167–194.

Millidge, Beren, Alexander Tschantz, and Christopher L Buckley (2020a). "Predictive Coding Approximates Backprop along Arbitrary Computation Graphs". In: *arXiv preprint arXiv:2006.04182*.

— (2020b). "Relaxing the constraints on predictive coding models". In: *arXiv preprint arXiv:2010.01047*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski (2015). "Human-level control through deep reinforcement learning". In: *Nature*.

Netzer, Yuval, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng (2011). "Reading Digits in Natural Images with Unsupervised Feature Learning". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Nøkland, Arild (2016). "Direct feedback alignment provides learning in deep neural networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1037–1045.

Nøkland, Arild and Lars Hiller Eidnes (2019). "Training neural networks with local error signals". In: *arXiv preprint arXiv:1901.06656*.

O'Reilly, Randall C (1996). "Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm". In: *Neural Computation* 8.5, pp. 895–938.

O'reilly, Randall C and Yuko Munakata (2000). *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. MIT Press Cambridge.

Ororbia, Alexander G and Ankur Mali (2019). "Biologically motivated algorithms for propagating local target representations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 4651–4658.

Ororbia, II, G Alexander, Patrick Haffner, David Reitter, and C Lee Giles (2017). "Learning to adapt by minimizing discrepancy". In: *arXiv preprint arXiv:1711.11542*.

Parken, D. B. (1985). *Learning-logic: Casting the Cortex of the Human Brain in Silicon*. Tech. rep. Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science.

Parker, David B (1985). "Learning-logic: Casting the cortex of the human brain in silicon". In: *Technical report TR-47*.

Payeur, Alexandre, Jordan Guerguiev, Friedemann Zenke, Blake A Richards, and Richard Naud (2021). "Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits". In: *Nature Neuroscience*, pp. 1–10.

Pineda, Fernando (1987). "Generalization of back propagation to recurrent and higher order neural networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 602–611.

Pineda, Fernando J (1988). "Dynamics and architecture for Neural Computation". In: *Journal of Complexity* 4.3, pp. 216–245.

Pozzi, Isabella, Sander Bohté, and Pieter Roelfsema (2018). "A biologically plausible learning rule for deep learning in the brain". In: *arXiv preprint arXiv:1811.01768*.

Puri, Raul, Robert Kirby, Nikolai Yakovenko, and Bryan Catanzaro (2018). "Large scale language modeling: Converging on 40gb of text in four hours". In: *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, pp. 290–297.

Ramsauer, Hubert, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, and Geir Kjetil Sandve (2020). "Hopfield networks is all you need". In: *arXiv preprint arXiv:2008.02217*.

Rao, Rajesh PN and Dana H Ballard (1999). "Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects". In: *Nature Neuroscience* 2.1, pp. 79–87.

Rescorla, Robert A (1972). "A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement". In: *Current research and theory*, pp. 64–99.

Richards, Blake A and Timothy P Lillicrap (2019). "Dendritic solutions to the credit assignment problem". In: *Current Opinion in Neurobiology* 54, pp. 28–36.

Richards, Blake A, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, and Surya Ganguli (2019). "A deep learning framework for neuroscience". In: *Nature Neuroscience* 22.11, pp. 1761–1770.

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method". In: *The Annals of Mathematical Statistics*, pp. 400–407.

Roelfsema, Pieter R and Anthony Holtmaat (2018). "Control of synaptic plasticity in deep cortical networks". In: *Nature Reviews Neuroscience* 19.3, p. 166.

Roelfsema, Pieter R and Arjen van Ooyen (2005). "Attention-gated reinforcement learning of internal representations for classification". In: *Neural Computation* 17.10, pp. 2176–2214.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science.

— (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536.

Sacramento, João, Rui Ponte Costa, Yoshua Bengio, and Walter Senn (2018). "Dendritic cortical microcircuits approximate the backpropagation algorithm". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8721–8732.

Salvatori, Tommaso, Yuhang Song, Yujian Hong, Simon Frieder, Lei Sha, Zhenghua Xu, Rafal Bogacz, and Thomas Lukasiewicz (2021). "Associative Memories via Predictive Coding". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Salvatori, Tommaso, Yuhang Song, Thomas Lukasiewicz, Rafal Bogacz, and Zhenghua Xu (2022). "Reverse Differentiation via Predictive Coding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Scellier, Benjamin and Yoshua Bengio (2017). "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation". In: *Frontiers in Computational Neuroscience* 11, p. 24.

Seung, H Sebastian (2003). "Learning in spiking neural networks by reinforcement of stochastic synaptic transmission". In: *Neuron* 40.6, pp. 1063–1073.

Shamir, Ohad (2017). "An optimal algorithm for bandit and zero-order convex optimization with two-point feedback". In: *The Journal of Machine Learning Research* 18.1, pp. 1703–1713.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Singer, Yosef, Yayoi Teramoto, Ben DB Willmore, Jan WH Schnupp, Andrew J King, and Nicol S Harper (2018). "Sensory cortex is optimized for prediction of future input". In: *Elife* 7, e31557.

Song, Yuhang, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz (2020). "Can the brain do backpropagation?—Exact implementation of backpropagation in predictive coding networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Europe PMC Funders, p. 22566.

Song, Yuhang, Jianyi Wang, Thomas Lukasiewicz, Zhenghua Xu, and Mai Xu (2019). "Diversity-driven extensible hierarchical reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 4992–4999.

Song, Yuhang, Jianyi Wang, Thomas Lukasiewicz, Zhenghua Xu, Shangtong Zhang, Andrzej Wojcicki, and Mai Xu (2020). "Mega-reward: Achieving human-level play without extrinsic rewards". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04, pp. 5826–5833.

Song, Yuhang, Andrzej Wojcicki, Thomas Lukasiewicz, Jianyi Wang, Abi Aryan, Zhenghua Xu, Mai Xu, Zihan Ding, and Lianlong Wu (2020). "Arena: A general evaluation platform and building toolkit for multi-agent intelligence". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05, pp. 7253–7260.

Soto, Victor, Alberto Suárez, and Gonzalo Martıénez-Muñoz (2016). "An urn model for majority voting in classification ensembles". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Neural Information Processing Systems Foundation.

Spall, James C (1992). "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation". In: *IEEE Transactions on Automatic Control* 37.3, pp. 332–341.

Tsividis, Pedro A, Thomas Pouncy, Jaqueline L Xu, Joshua B Tenenbaum, and Samuel J Gershman (2017). "Human learning in Atari". In: *2017 AAAI Spring Symposium Series*.

Tsymbal, Alexey (2004). "The problem of concept drift: Definitions and related work". In: *Computer Science Department, Trinity College Dublin* 106.2, p. 58.

Van Essen, David C, Charles H Anderson, and Daniel J Felleman (1992). "Information processing in the primate visual system: An integrated systems perspective". In: *Science* 255.5043, pp. 419–423.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5998–6008.

Werbos, Paul (1974). "Beyond regression: new tools for prediction and analysis in the behavioral sciences". In: *Ph. D. dissertation, Harvard University*.

Werfel, Justin, Xiaohui Xie, and H Sebastian Seung (2004). "Learning curves for stochastic gradient descent in linear feedforward networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1197–1204.

Whittington, James CR and Rafal Bogacz (2017). "An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity". In: *Neural Computation* 29.5, pp. 1229–1262.

— (2019). "Theories of error back-propagation in the brain". In: *Trends in Cognitive Sciences*.

Whittington, James CR, Timothy Muller, Shirely Mark, Caswell Barry, and Tim Behrens (2018). "Generalisation of structural knowledge in the hippocampal-entorhinal system". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8484–8495.

Whittington, James CR, Timothy H Muller, Shirley Mark, Guifen Chen, Caswell Barry, Neil Burgess, and Timothy EJ Behrens (2020). "The Tolman-Eichenbaum machine: Unifying space and relational memory through generalization in the hippocampal formation". In: *Cell* 183.5, pp. 1249–1263.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). "Fashion MNIST: A novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747*.

Yamins, Daniel LK and James J DiCarlo (2016). "Using goal-driven deep learning models to understand sensory cortex". In: *Nature Neuroscience* 19.3, p. 356.

Yamins, Daniel LK, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo (2014). "Performance-optimized hierarchical models predict neural responses in higher visual cortex". In: *Proceedings of the National Academy of Sciences* 111.23, pp. 8619–8624.

Zenke, Friedemann and Surya Ganguli (2018). "Superspike: Supervised learning in multilayer spiking neural networks". In: *Neural Computation* 30.6, pp. 1514–1541.

Zipser, David and Richard A Andersen (1988). "A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons". In: *Nature* 331.6158, pp. 679–684.

Žliobaitė, Indrė (2010). "Learning under concept drift: An overview". In: *arXiv preprint arXiv:1010.4784*.