Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Simen Salomonsen

Bio-plausible Neural Networks

A Comparative Study of the Computational Demand and Capability to Infer Missing Values of Predictive Coding Networks

Master's thesis in Computer Science
Supervisor: Prof. Downing, K.
June 2023

NTNU
Norwegian University of
Science and Technology

Simen Salomonsen

Bio-plausible Neural Networks

A Comparative Study of the Computational Demand and Capability to Infer Missing Values of Predictive Coding Networks

Master's thesis in Computer Science
Supervisor: Prof. Downing, K.
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Artificial neural networks have had great success since their breakthrough using error backpropagation, sparking innovation in image classification, regression, generation, and many more fields. An issue backpropagation and artificial neural networks face, however, is the biological plausibility of these models. These issues relate to error representation, weight symmetry, and transmission of neuron signals. The first of these three issues is the main issue addressed by the bio-plausible neural network presented in this thesis. It will explore predictive coding networks, a machine-learning model presented by Whittington and Bogacz [2017], which builds on predictive coding theory originally presented by Rao and Ballard [1999].

The thesis presents a study of predictive coding networks' capability to infer missing values in incomplete datasets and an analysis of their computational demand compared against artificial neural networks using backpropagation. One experiment was conducted for each aspect, where the first qualitatively establishes that predictive coding networks can infer missing values in incomplete representational datasets — given the introduction of a decay term to their update rules. The findings also show that the performance of using predictive coding networks as an inference scheme lags behind k-Nearest Neighbors on both representational datasets used in the study. The second experiment quantitatively establishes that the number of floating point operations performed during the training of predictive coding networks is an average of 738% larger than those of artificial neural networks.

The findings of the first experiment show promise for predictive coding networks as a model capable of inference and prediction in both network directions and substantiate the related works by asserting their ability to infer information on visual and representational data. Future work can be aimed at investigating the performance of predictive coding networks as a function of the completeness of the utilized dataset. For the second aspect the thesis addresses, the findings suggest an advantage of artificial neural networks concerning environmental impact. Future research should explore ways to reduce the computational demand of the model.

Sammendrag

Kunstige nevrale nettverk har hatt stor suksess siden de fikk sitt gjennombrudd ved hjelp av backpropagation, og har ført til innovasjon innen bildeklassifisering, regresjon, generering og mange flere områder. Et problem med backpropagation og kunstige nevrale nettverk er imidlertid modellenes biologiske troverdighet. Disse problemene er knyttet til feilrepresentasjon, vektsymmetri og overføring av nevronsignaler. Det første av disse tre problemene er hovedproblemet for det bioplausible nevrale nettverket som presenteres i denne avhandlingen. Den vil utforske prediktiv koding-nettverk, en maskinlæringsmodell presentert av Whittington and Bogacz [2017], som bygger på teori rundt prediktiv koding opprinnelig presentert av Rao and Ballard [1999].

Avhandlingen presenterer en studie av prediktiv koding-nettverks evne til å fylle inn manglende verdier i ufullstendige datasett og en analyse av deres komputasjonelle ressursbehov sammenlignet med kunstige nevrale nettverk som bruker backpropagation. Et eksperiment ble gjennomført for hvert av aspektene, der det første kvalitativt fastslår at prediktiv koding-nettverk kan utlede manglende verdier i ufullstendige representasjonsdatasett — gitt at det innføres et 'decay'-ledd i oppdateringsreglene for nettverket. Funnene viser også at ytelsen av prediktiv koding-nettverk som inferensmetode er noe lavere enn k-Nærmeste Naboer på begge representasjonsdatasettene som ble brukt i studien. Det andre eksperimentet fastslår kvantitativt at antallet flyttallsoperasjoner som utføres under trening av prediktiv koding-nettverk i gjennomsnitt er 738% høyere sammenliknet med kunstige nevrale nettverk.

Funnene fra det første eksperimentet viser potensiale for prediktiv koding-nettverk som en modell som er i stand til både gjennomføre prediksjon i en retning av nettverket og inferens i den andre, og underbygger de relaterte artiklene ved å hevde modellens evne til å utføre inferens på både visuelle og representasjonelle data. Fremtidig arbeid kan ta sikte på å undersøke ytelsen til prediktive koding nettverk som en funksjon av hvor komplett datasettet er. Angående det andre aspektet avhandlingen tar for seg, tyder funnene på at kunstige nevrale nettverk har et fortrinn når det kommer til det miljømessige fotavtrykket. Fremtidig forskning bør undersøke hvordan prediktiv koding-nettverks komputasjonelle behov kan reduseres.

Preface

The research presented in this paper was conducted at the Department of Computer Science under the Faculty of Information Technology and Electrical Engineering at the Norwegian University of Science and Technology (NTNU). The paper presents the culmination of research and experiment executed during the Spring of 2023 as part of the Master Thesis course (TDT4900) for the integrated computer science degree at NTNU. It is a continuation of the work presented in the preceding specialization project (TDT4501) of fall 2022. Consequently, the introduction, background, and related works, chapter 1, 2, and 3 respectively, are heavily based on the chapters under the same name of the unpublished paper from the specialization project Salomonsen [2022].

I want to give a special thanks to prof. Downing, K. for supervising the thesis, providing invaluable insights and guidance through the research and motivating throughout the whole year. A second thanks goes out to prof. Elster, A., and assoc. prof. Hetland, M.L. for helping form the frames of the computational analysis of the models in the thesis. Finally, a big thank you goes out to all the friends and family that have supported me throughout the thesis and years at the university.

Simen Salomonsen
Trondheim, June 10, 2023

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The last decades have shown significant advances in artificial intelligence (AI) research. Especially machine learning (ML) and artificial neural networks (ANN) using error backpropagation (BP) has gained a lot of traction since it was first presented by Rumelhart, Hinton, and Williams (Rumelhart et al. [1986]). Among the architectures that illustrate the success of BP are convolutional neural networks, natural language processing models, and deep neural networks.

An issue regarding neural networks relates to their biological plausibility compared to the human brain. First, the brain is likely incapable of calculating and propagating complex derivates of the error backward in the brain. Therefore, a field of research that has had an upswing in the latter years looks into explicitly representing errors in the network and making local changes based on them. Secondly, another concern posed by researchers in the field is that it is unlikely for the brain to have symmetrical weights for the feedforward and feedback of messages in the brain used in BP. Lastly, it is known that nerve cells in the brain communicate using spikes, communicating their activation only for a limited amount of time. Neurons in ANNs, on the other hand, express their activation continuously to the connected neighbors, creating another discrepancy between ANNs and the brain.

This thesis will focus on the first of these three concerns, how errors are represented in the brain. However, research that addresses the requirement for symmetrical weights for BP to learn has been published (Liao et al. [2016]; Lillicrap et al. [2016]) as well as papers presenting ways to overcome the issue of finding a derivative for the spiking function of biological neurons (Whittington and Bogacz [2019]).

Elaborating on the first issue, two different classes of models have emerged that tackle this problem: explicit error models and temporal models. The thesis

will focus on the former. Explicit error models are reminiscent of neural networks; however, they represent the error of the network explicitly in the network and make updates to the network based on this representation. Of the models following this approach, the predictive coding network presented in (Whittington and Bogacz [2017]) will be the specific model used for the thesis. The theory behind both ANNs using BP and PCNs will be presented in chapter 2.

PCNs have performed comparably to standard ANNs (using BP) on famous machine learning benchmarks such as the MNIST dataset. Additionally, papers have shown that predictive coding (PC) is applicable to more sophisticated models such as CNNs, RNNs, and LSTMs - with comparable results to their BP counterparts. Finally, introducing decay terms to the update rules has also proved the usefulness of PCNs in the generative domains. These papers will be further explored in chapter 3.

With these advancements, PCNs contend with "vanilla" BP, and if further improvements are made with equivalent "supplements" that have been found for BP, such as, e.g., using the ReLU function as an activation function and batching during training, PCNs could take over for ANNs as better and more bio-realistic neural networks. The goal of this thesis project will be to further research the viability of PCNs as a replacement for ANNs. With this, two research questions have been formulated to help address this goal:

RQ1 Are PCNs better equipped to handle missing values compared to ANN (with and without imputation of attributes)

Missing input values are a challenge for machine learning. Poor data gathering leads to sparse datasets, which ANN using BP are not designed to handle. However, PCNs have displayed memorization and reconstruction capabilities for visual data, meaning the models could also be able to reconstruct numerical data in other datasets and automatically infer the missing values while classifying the sample. Finding a model that can handle missing values and possibly infer them could yield great performance and efficiency benefits to the field of AI.

RQ2 How does PCN's environmental impact compare to that of ANN?

Computational demands - As AI methods have become more powerful and widely used, the computation and resource demands have also increased. This has been linked to energy consumption and CO_2 emissions. Neural nets using BP are very computationally intensive, with error derivates that must be propagated along many long paths in the network. Looking into less demanding alternatives for computational resources can help combat some of the adverse effects of AI and Big data on the environment.

These research questions provide the motivation behind the thesis, and this paper will address them to contribute to the overall research goal of investigating if PCN could be an alternative to ANNs that is more biologically plausible.

Outline of the paper: The necessary background theory will be presented in chapter 2, continuing with related works in chapter 3. Moving forward, the methodology for the implementation and experiments of the thesis will be presented in chapter 4, and the results of the experiments will be presented, analyzed, and discussed in chapter 5 along with the implementation details of the planned system. Lastly, chapter 6 will summarize the work and point to future work based on the research.

Chapter 2

Background

The following chapter will provide the background and go further in detail about the preliminary knowledge for the thesis. Then, it will introduce artificial neural networks, continue with the biological issues of ANNs, present a mathematical definition of ANNs, and lastly, present the essential thoughts behind predictive coding networks.

2.1 Artificial Neural Networks

An artificial neural network is a computational graph consisting of nodes connected by edges representing a neural network similar to the ones found in mammals. Typically the nodes in the network are referred to as neurons, and the edges that connect the nodes as the weights between the neurons. The network can have any arbitrary structure between the neurons, but often the neurons are structured in a hierarchy where the nodes are ordered in layers. The connections run from the neurons of one layer to the next layer, and the nodes within the same level have no edges to each other. These networks can be used to make a prediction about some problem given data about it by presenting the information at one level and allowing the network to calculate the changes in the subsequent layers to the output. This prediction-making phase is referred to as the inference phase. The prediction can be improved upon by making adjustments to the weights, and the phase responsible for this tuning is called the learning phase. The tuning is typically done by an external algorithm named backpropagation. Section 2.3 will go further into how this algorithm works, along with the mathematical definitions of inference and learning in artificial neural networks.

2.2 Bioplausibility Issues

The following section will present the biological issues of ANNs in more detail, emphasizing the error representation issue as this is the issue the framework presented later in the thesis tackles. The descriptions of the problems are based on those given in (Whittington and Bogacz [2019]).

2.2.1 Error Representation

When updating an ANN, the error terms computed are based on an external algorithm that propagates the errors backward through computationally intensive calculations from the output to input neurons. Biological neurons update their weight solely based on the activity of adjacent neurons, and seeing how the brain would implement backpropagation which calculates the error terms globally, becomes difficult. With a local error representation in neural networks, the networks can make local changes without using a global error function, bridging the gap between the digital replica of our brain and the biological reality.

2.2.2 Weight Symmetry

During the backpropagation of errors, the error terms are calculated along the same paths as the prediction during the forward pass. Using the same connections in both directions implies that the connections in the brain are bidirectional. While bidirectional connections have been found in the brain, they are not always present. Even if the connections always were symmetrical in the human brain, as in ANNs, there would still be the challenge of having the connections in the brain align themselves correctly.

2.2.3 Spike-timing Dependent Plasticity

The last issue raised by Whittington and Bogacz relates to the communication of neurons. Biological neurons in the brain use discrete spikes to communicate with their neighbors, while artificial neurons transmit their continuous values. The issue here relates to finding the derivative of spikes to propagate the errors in the backpropagation algorithm, presenting another discrepancy between artificial and biological neurons.

2.3 Mathematical Definition of Artificial Neural Networks

Based on the description of neural networks provided in 2.1, a mathematical explanation will be presented to help elaborate on how the BP algorithm functions. Suppose such a network exists where the activation of the neurons in the network is given by $x_i^{(l)}$ where i is the position of the neuron in the layer l that the neuron belongs. Secondly let $\theta_{i,j}^{(l)}$ denote the weight between the i-th neuron at layer l to the j-th neuron at layer $l+1$. The following two sections will explain how ANN executes inference and learning and is based on the description of backpropagation provided in (Downing [2023]).

2.3.1 Inference

The inference is made by presenting an input vector to the bottommost layer and letting the change propagate through the network along the edges. That is

$$\boldsymbol{x}^{(0)} = \boldsymbol{x}' \tag{2.1}$$

where $\boldsymbol{x}^{(0)}$ is the collection of the neurons of the bottommost layer of the network (referred to as the input layer), and $\boldsymbol{x}'$ is a vector of the input. The activity of the next layer is then calculated by aggregating the products of the weight of each incoming edge and the activity corresponding pre-synaptical neuron before passing the aggregate through some non-linear activation function for each neuron in the next layer. This step is repeated recursively until the change has propagated to the final layer of the network, referred to as the output layer. Mathematically, the value of a neuron i in layer $l+1$ is given as

$$x_i^{(l+1)} = \sigma\Big(\sum_{j\in(l)} x_j^{(l)}\theta_{j,i}^{(l)}\Big) \tag{2.2}$$

where σ is some non-linear activation function, such as the sigmoid logistic function or rectified linear unit function (ReLU).

The calculation of changes that takes place to predict a new datapoint is often called forward pass, and using vector notation can simplify this expression, yielding the following equation for the update of the layers:

$$\boldsymbol{x}^{(l+1)} = \boldsymbol{\sigma}(\boldsymbol{x}^{(l)}\boldsymbol{\theta}^{(l)}) \tag{2.3}$$

where $\boldsymbol{\theta}^{(l)}$ denotes the weight matrix from layer (l) to layer $(l+1)$ and $\boldsymbol{\sigma}$ is an element-wise application of the activation function for a vector.

To summarize, the inference phase is executed by presenting a data point to the input layer and allowing the changes to propagate forwards in the network.

The resulting prediction for a given data point will be the resulting values at the output layer.

2.3.2 Learning

Learning in ANNs occurs by updating the network parameters based on the calculated error after a prediction. Let E denote the error term. The two main gradients which are needed then are $\frac{\partial E}{\partial S}$ and $\frac{\partial E}{\partial w}$, which denotes the effect of the sum of weighted inputs to a node on the error and the impact of the output weight of a node on the error respectively. The former is used as a step to calculate the latter, and letting S be defined as:

$$S_i = \sum_j x_j \theta_{j,i} \tag{2.4}$$

yields that the following gradients hold for S:

$$\frac{\partial S_i}{\partial \theta_{j,i}} = x_j \tag{2.5}$$

and

$$\frac{\partial S_i}{\partial x_j} = \theta_{j,i} \tag{2.6}$$

Let δ_i for any neuron i be defined as $\frac{\partial E}{\partial S}$. By using the chain rule, it can be shown that

$$\frac{\partial E}{\partial \theta_{i,j}} = \frac{\partial S_j}{\partial \theta_{i,j}} \frac{\partial E}{\partial S_j} = x_i \delta_j \tag{2.7}$$

The equation tells how any arbitrary weight in the network needs to be updated to increase the error term. Flipping the direction of the gradient will provide the minimization term, meaning

$$\Delta \theta_{i,j} = -\lambda \frac{\partial E}{\partial \theta_{i,j}} = -\lambda x_i \delta_j \tag{2.8}$$

defines the update rule for the network (where λ denotes the learning rate) that will decrease the error in the network's prediction.

Lastly, δ_j for any neuron in the network is found by defining the error term E and using E to derive it. Suppose that

$$E = \frac{1}{2} \sum_i (\hat{x}_i - x_i)^2 \tag{2.9}$$

where $\hat{x}_i$ is the target value for the same neuron x_i. The difference in the $(\hat{x}_i - x_i)$ must decrease to reduce the network error. As such, the gradient

$$\frac{\partial E}{\partial x_i} = -(\hat{x}_i - x_i) \tag{2.10}$$

gives the effect of each neuron on the error. Using the chain rule and eq. 2.10 on the definition of δ_j yields

$$\delta_i = \frac{\partial E}{\partial S_i} = \frac{\partial x_i}{\partial S_i}\frac{\partial E}{\partial x_i} = -f'(S_i)(\hat{x}_i - x_i) \tag{2.11}$$

The target values for the output layer are calculated using the label of the data point $\boldsymbol{x}'$. At the same time, the subsequent error terms are found using the chain rule and propagating the effect each node has on the remainder of the network backward.

$$\delta_i = f'(S_i)\frac{\partial E}{\partial x_i} = f'(S_i)\sum_{j\in(l+1)}\left[\frac{\partial S_j}{\partial x_j}\frac{\partial E}{\partial S_j}\right] = f'(S_i)\sum_{j\in(l+1)}[\theta_{i,j}\delta_j] \tag{2.12}$$

Expanding δ_j equation 2.8 for the case of the output layer and hidden layer then yields

$$\Delta\theta_{i,j}^{(l)} = -\lambda x_i \delta_j = -\lambda x_i \begin{cases} f'(S_i)(\hat{x}_i - x_i) & (l = L_{max}) \\ f'(S_i)(\sum_{k\in(l+1)}\theta_{j,k}\delta_k) & (l \neq L_{max}) \end{cases} \tag{2.13}$$

After a data point has been run, the errors, δ_i, are propagated backward in the network based on the error calculated in the prediction of the output layer. Afterward, equation 2.13 is run to update the weights in the right direction. The layer index has been omitted in many cases to avoid notational clutter.

This description of ANNs using BP will serve as a basis for comparison to PCNs covered in the next section. Further, it illustrates the biological issues raised in section 2.2 by showing how the error is not represented locally but globally by the BP algorithm.

2.4 Predictive Coding

The term predictive coding originates from psychology and neuroscience. It denotes a phenomenon where neurons predict the activation of neighboring neurons and inhibits them from traveling further along the nerve path if the prediction is correct. The prediction is also sent back to the neighboring neuron, where the error is calculated and fed back to the original neuron along feed-forward paths. This mechanism makes it so that only when the prediction of neurons is wrong

is the signal propagated through the network. In this way, neural networks can remove redundant signals by filtering out the predictable.

Rao and Ballard published a paper in 1999 presenting a model utilizing this concept to process natural images visually (Rao and Ballard [1999]). The model has sparked several new models inspired by predictive coding and has shown how weight updates and learning can be made local by calculating the error terms based on only adjacent nodes in the network. The dynamics of the inference phase of the network have been summarized in figure 2.1, showing how the predictions of subsequent layers inhibit the forward feeding of signals to successive layers.

In their framework, the optimization function is set to be the internal energy of both the network's activations and error terms, building on theory from energy-based models. Improvement is made by performing gradient descent on the optimization function, like backpropagating ANNs. In their paper, this term contains the sum of the squared error terms in the network, the activation of nodes, and the weights in the network, but for brevity, let E denote this optimization function. Further, their model presents a hierarchical structure where the layers convey their predictions of the activation of the previous layer backward and update their prediction based on the error fed forward again by that previous layer in an iterative manner. This update in the activation of layers can be defined as performing gradient descent on E

$$\Delta \boldsymbol{x} = -\frac{\partial E}{\partial \boldsymbol{x}} \tag{2.14}$$

for each datapoint, where $\boldsymbol{x}$ is the activation matrix of the nodes in the network. In other words, this performs the network inference by reducing the error. Further, after the network has equilibrated during inference, learning takes place by updating the weights between the layers according to the equation

$$\Delta \boldsymbol{\theta} = -\frac{\partial E}{\partial \boldsymbol{\theta}} \tag{2.15}$$

after presenting the correct solution at the top-most layer in the hierarchy, where $\boldsymbol{\theta}$ is the weight matrix of the network. The core concepts are the same between PCN and ANN in the goal of updating the parameters, but they differ in execution. For example, where ANNs' objective function is a function of the error at only the output, PCNs take the error in the whole network into account. Chapter 3 will further explore a paper by Bogacz which illustrates how BP can be seen as an edge case of their PCN framework, and chapter 4 will use this theory in a set of experiments designed to help explore the capabilities of PCNs in light of the *RQs*.

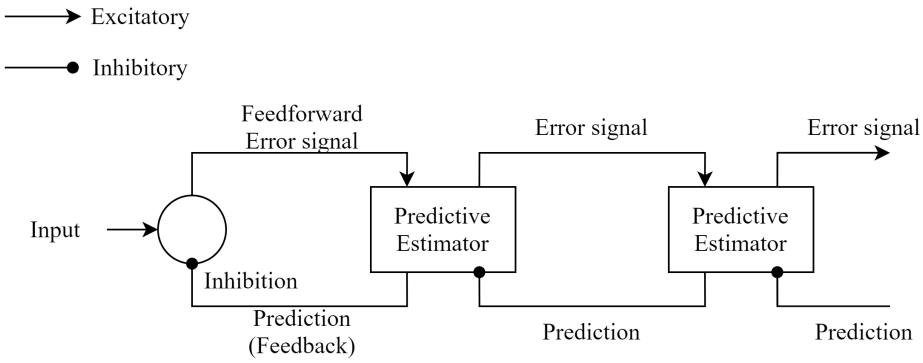


Figure 2.1: Dynamics of a predictive coding network as presented in (Rao and Ballard [1999]). The figure shows how the iterative loop between two levels in the hierarchical model converges towards a fixed point. The lower level will send the residual error to the higher layer, which then sends a new prediction to the lower level again for a new evaluation of the error term, repeating the cycle. (Source: Replication based on figure 1a in (Rao and Ballard [1999]))

2.5 Summary

The theory presented in this chapter lays down the theoretical background for the frameworks that will be developed for the experiments used to research the RQs. In addition, the workings of backpropagation have been illustrated and connected to how it is used in neural networks. Further, the chapter has also presented the preliminary theory for the framework presented in (Whittington and Bogacz [2017]), which will be further explained in section 3.1. Lastly, it has elaborated upon the biological challenges ANNs using BP face. This foundation will be elaborated upon in the next chapter, where related works of the research field are presented and discussed. It will further be used to explain the frameworks presented in chapter 4.

Chapter 3

Related Work

The following chapter will look at a set of related papers further discussing applications of predictive coding and other frameworks which explore different ways to distribute the error term in the network, which have both motivated the research questions, techniques to help research them, and related issues that PCN face that will need to be addressed in the future.

3.1 Whittington & Bogacz - Predictive Coding Networks

(Whittington and Bogacz [2017]) applies the theory of (Rao and Ballard [1999]), and presents a framework for implementing predictive coding networks. By using iterative updating of the activations based on the error nodes of the network, the framework achieves inference according to

$$\dot{x}_b^{(a)} = -\varepsilon_b^{(a)} + \sum_{i=1}^{n(a-1)} \varepsilon_i^{(a-1)} \theta_{i,b}^{(a)} f'(x_b^{(a)}) \tag{3.1}$$

where $\dot{x}_b^{(a)}$ denotes the change in activation of neuron b in layer (a), ε denotes the respective errors of their node, θ denotes the weight between the previous error node and the neuron. Lastly, $f'(\cdot)$ is the inverse of the activation function used in the network (Eq. 2.18 in (Whittington and Bogacz [2017])). ε is defined as the difference between the activation of the respective neuron at the same level and the top-down prediction received from the next layer divided by the variance of the neuron.

$$\varepsilon_i^{(l)} = \frac{x_i^{(l)} - \mu_i^{(l)}}{\Sigma_i^{(l)}} \tag{3.2}$$

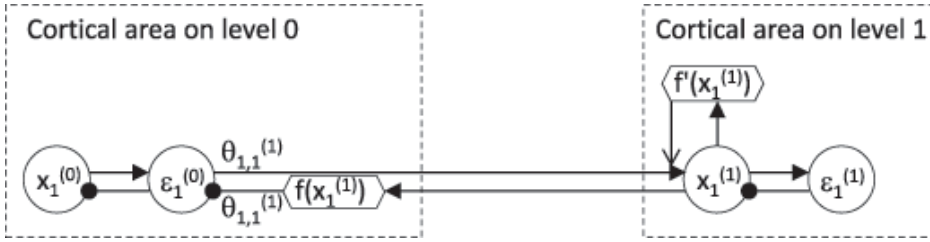


Figure 3.1: Dynamics of neurons in predictive coding network, as presented in Whittington and Bogacz [2017], displaying how prediction errors are propagated forward along excitatory paths, and predictions are fed backward in the network along inhibitory paths. (Source: Figure 3 in (Whittington and Bogacz [2017]))

where μ_i is the top-down prediction and Σ_i denotes the variance of the neuron (Eq. 2.17 in (Whittington and Bogacz [2017])). The update to the activation equates to the gradient descent of the optimization function (ref Eq. (2.14)) from Rao and Ballard's framework and will drive the network to the optimal state given the current weights.

Once the network has stabilized, the weights are updated according to the residual error found in the error nodes to minimize the objective function, here denoted by F

$$\frac{\partial F^*}{\partial \theta_{b,c}^{(a)}} = \varepsilon_b^{*(a-1)} f(x_c^{*(a)}) \tag{3.3}$$

where variables annotated by an asterisk denote their value at equilibrium. This update will further improve on the optimization function F and equates to the second gradient descent done on the network with regards to the weights (ref Eq. (2.15)).

Figure 3.1 summarizes the network dynamics, and using this structure, they implemented a neural network that was of comparable performance to that of an equal BP counterpart. Applying the networks to MNIST, all were able to attain a training error of 0.00% and a validation error of 1.7% to 1.8%.

Lastly, the paper also analyzes how the variance at the different layer affect the network's performance at various tasks. It uses this to argue that PCN can approximate BP when the variance at the output layer is high. The paper bridges the two frameworks and provides the fundamental structure for predictive coding networks while illustrating their applicability to modern tasks.

3.2 Millidge et al. - Application to Modern Structures

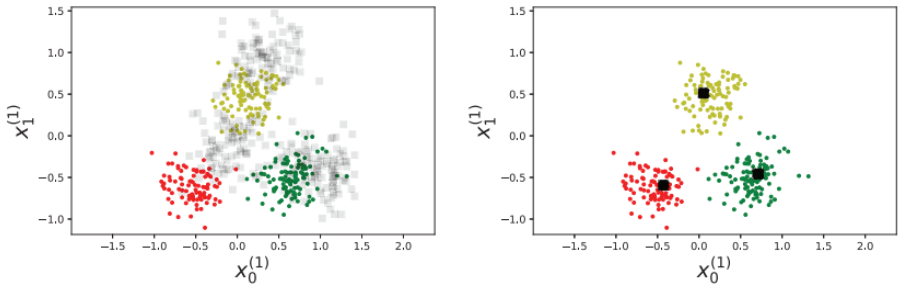
Based on the network structure presented by Whittington & Bogacz (2017), Millidge et al. showed a continuation of the work and implemented equivalent models of convolutional neural networks (CNN), recurrent neural networks (RNN), and long short-term memory networks (LSTM) using PC for the dynamics (Millidge et al. [2022]). The models were also tested against their BP counterparts on several datasets significantly more complex than MNIST; CIFAR, CIFAR100 & SVHN for the CNN models, and character-level classification and prediction of the complete works of Shakespeare for the RNN and LSTM models, respectively. The performance of the ANN and PCN models were practically equivalent; however, the authors stated a 100 times higher computational cost of the PC mode networks due to the need for convergence during the inference cycles, suggesting an advantage of the BP counterparts concerning sustainability.

3.3 Sun & Orchard - Generative PCN

By introducing a new term to the update rules of the activation of neurons and weights, Orchard and Sun made a predictive coding network that was both discriminative and generative (Sun and Orchard [2020]). The added term introduces decay to both activation and weights of the network, which acts as a way to confine the possible causes (inputs) of a given class to the subset of plausible causes. Decay forced the generated samples to converge to plausible explanations for the supplied class instead of all possible, as shown by the narrowing of the grey scatters in figure 3.2 from (a) to (b). The author explains how decay collapses the null space of the inputs, causing the remaining possible explanations for the values observed at the output to correspond to plausible values at the input. The decay networks were applied to the MNIST dataset and could generate more realistic images than a non-decay network. While the model in the paper is applied to visual data, using a similar decay term may also prove helpful in a network tuned to infer missing attributes in numerical datasets where the inputs are incomplete.

3.4 Salvatori et al. - Associative Memories

The paper (Salvatori et al. [2021]) explores the memory capabilities of predictive coding networks. It illustrates that PCNs can retrieve, reconstruct, and clean natural images. After training a network, the abilities were demonstrated by presenting the network with a partial or corrupted image and allowing the predictions of higher layers to flow backward. The experiments were performed on a



(a) PC-non-decay Network

(b) PC-decay Network

Figure 3.2: Comparison of generated samples from non-decay and decay networks. (Source: Figure 6 in (Sun and Orchard [2020]))

2-layer PCN and compared to a 3-layer auto-encoder (AE) using BP. Using images from CIFAR10 and Tiny ImageNet, they found the PCN was superior to the AE model in all instances and was generally able to reconstruct corrupted images and retrieve images based on a fraction of the pixels, given that the network was not overparameterized. In addition, the paper illustrates how PCN creates memories of the images in the dataset and how this can be used to reconstruct the missing or corrupted parts of the picture. These memory abilities may also have applications for inferring missing values in other fields and datasets containing other non-visual data. Further, the paper also illustrates an advantage of PCN over AE using BP of comparable size, suggesting a potential for a more sustainable and resource-effective replacement, improving the performance for the given task.

3.5 Tschantz et al. - Hybrid Predictive Coding

A way to overcome the computationally demanding process of converging to a fixed point during inference of PCN is presented in Tschantz et al. [2022]. They propose a hybrid predictive coding (HPC) model, where standard predictive coding is used in both network directions. They differentiate inference into two parts, *iterative* and *amortized* inference—the former accounts for extracting information from sensory data while the latter accounts for rapid recognition and perception. The main novelty of the framework is an extension where predictions and errors are communicated in both directions. Top-down predictions still learn to generate the data hierarchically, while bottom-up predictions learn the beliefs of the subsequent layers. In this way, the model can be trained to quickly create predic-

tions using amortized inference without requiring the network to converge to an equilibrium. Their model was tested against the MNIST data set, demonstrating capabilities to classify and generate images of the class correctly, and illustrated advantages over amortized inference alone and standard predictive coding both in terms of performance and over standard predictive coding in terms of iterative steps required.

3.6 Scellier & Bengio - Equilibrium Propagation

Looking at other frameworks for bio-plausible learning in neural networks, target-propagation, as presented in (Bengio et al. [2015]), is a model where an energy function drives the network to a local minimum based on an energy function. After presenting the network with input by clamping the bottommost layer, the net will settle to a state u^0. The target value for the output layer is then presented to the network, and the output nodes are nudged in the direction of that target, causing a change to propagate backward in the net akin to the propagation of error derivatives in BP. Once the net has converged to a new equilibrium state, u^β, the weight between the neurons is updated based on the difference in activations of the two states u^β and u^0. The authors dubbed this approach equilibrium propagation, and experimental results show convergence towards 0.00% training error and $2 - 4\%$ validation error on the MNIST dataset. Further, the authors also link the framework to spike-timing dependent plasticity, yielding insight into how the issue discussed in section 2.2.3 might be approached for PCNs as well.

3.7 Lillicarp et al. - Symmetrical Weight Requirement for BP

The last biological issue discussed in section 2.2.2 regarding symmetrical weights has also seen progress. In a paper by Lillicarp et al. (2016), they proved that BP could also learn to classify the MNIST data set with asymmetrical weights correctly. Another article by Liao et al. also researched the same and found it was possible to make BP learn with random feedback weights given some assumptions on their initialization (Liao et al. [2016]). Deploying similar approaches may prove beneficial to strengthen the biological plausibility of PC models further.

3.8 Arild Nøkland - Direct Feedback-Alignment

A novel framework for training deep neural networks is presented in (Nøkland [2016]) where the error at the output layer of the network is fed backward in

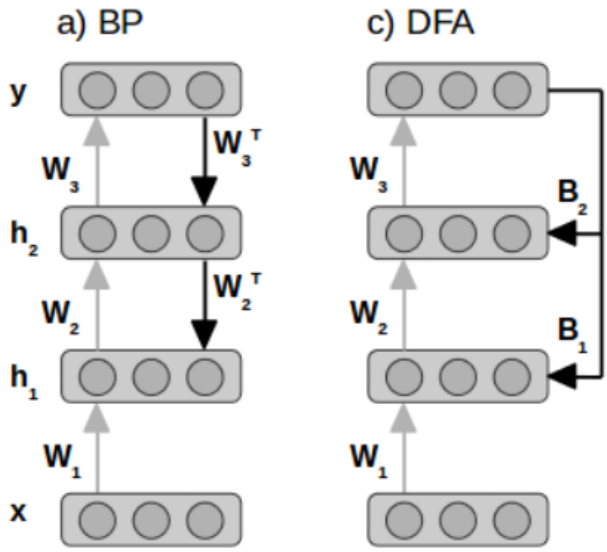


Figure 3.3: Comparison of BP and DFA. Subfigure a) shows how the layers propagate the error terms backward in the network through each layer. Subfigure c) shows DFA, where the error terms are fed backward along direct paths from the output layer to the preceding layers without going through higher layers. (Source: Sections of Figure 1 in (Nøkland [2016]))

the network through direct feedback paths separate from the feed-forward path, called direct feedback alignment (DFA). Figure 3.3 compares the informational flow of BP and DFA. The method is also believed to be biologically plausible and would have clear computational benefits over BP as the computational intensive gradients propagated backward are replaced by a simpler directly computed error term that bypasses the intermediary computations of BP. The paper also presents theoretical results, and by using this method for learning, a DFA network achieved a training error and validation error of zero and 1.45%, respectively. While the performance lags behind BP, the savings in computational power is a clear advantage, and PCNs may also benefit from a similar approach.

3.9 Hinton & Salakhutdinov - Restricted Boltzmann Machine

A class of models highly influential in machine learning is the restricted Boltzmann machine (RBM). (Hinton and Salakhutdinov [2006]) presents an implementation of the model, and the model is structured as a bidirectional, fully connected, hierarchical network where the layers converge to equilibrium and are trained pairwise in sequence, similar to equilibrium propagation (Scellier and Bengio [2017]) and predictive coding networks (Whittington and Bogacz [2017]). The paper by Hinton & Salakhutdinov also displays reconstructional abilities by implementing the RBM as an autoencoder that could reconstruct images. Similar approaches have been used to make PCNs generative (Sun and Orchard [2020]) and may prove useful in reconstructing missing values.

3.10 Schwartz et al. - Green AI

An analysis of the sustainability of AI research is presented in (Schwartz et al. [2020]). The paper presents a literary study of articles from large AI conferences, including the annual meeting of the Association for Computational Linguistics (ACL), the conference on Neurological Information Processing Systems (NeurIPS), and the conference on Computer Vision and Pattern Recognition (CVPR), looking into the main reported metrics of the models. The authors found that most of the papers focus solely on the accuracy of the presented models, and very few focus on the efficiency and the computational or resource demands of the models. This lack of focus on effectiveness, they argue, is an issue concerning sustainability issues and presents a discussion of metrics that can be used to measure the environmental impact of AI. Ultimately, they advocate for using floating point operations (flops) to report the computational demand of models. As PCN is also a computationally heavy method, the metric also applies to measure models' efficiency. It enables comparison to similar models, such as ANN using BP, that can report the same metric.

3.11 Summary

The works presented in this chapter give an overview of some of the ongoing research in the field of predictive coding networks, as well as related fields. (Whittington and Bogacz [2017]) formulates the basis that will be used to set up the framework for creating PCN models to execute the experiments presented in section 4. Additionally, the viability of this framework has been illustrated by (Millidge et al. [2022]; Tschantz et al. [2022]). Further, (Schwartz et al. [2020])

provides both a motivation and approach for *RQ2* by discussing and proposing a lack of reporting of clear, comparable metrics for computationally heavy AI models as well as proposing using FLOPS as such a metric. A 3rd party library will be used based on this discussion to make a comparison between PCNs and ANNs using BP (Nøkland [2016]; Tschantz et al. [2022]; Salvatori et al. [2021]) also indicated resource-saving aspects of alternative approaches to ANNs using BP, further motivating *RQ2*. Regarding *RQ1*, (Hinton and Salakhutdinov [2006]) illustrates RBMs' capabilities as autoencoders, and (Salvatori et al. [2021]) shows how PCNs can capture the memory of images within the network and can be used to both recover and denoisify images, motivating *RQ1*. (Sun and Orchard [2020]) also addresses the reconstruction of inputs based solely on outputs by including activation- and weight-decay to the update equations. The same technique will be applied to see if PCNs can record the relationship between inputs and reconstruct missing values in datasets. Lastly (Lillicrap et al. [2016]; Liao et al. [2016]; Bengio et al. [2015]) show alternative models and research that help bridge the gap between machine learning and biology, and illustrate ways that can be useful to tackle the other two issues raised in section 2.2.2 and 2.2.3. Chapter 4 will go further in detail on how the framework will be developed and used to perform the experiments to research the *RQ*s.

Chapter 4

Methodology

Based on the *RQs* in the introduction, two experiments were set up to explore these research questions. The following chapter will present the methodology for these experiments, beginning by explaining the general frameworks for creating ANN and PCN models and how they were implemented in section 4.1 before moving on to the experiments. The first experiment, covered in section 4.2, aimed at researching *RQ1* by looking at PCNs' specific capability to infer missing values in a sparse dataset and how models would perform using a reconstructed dataset made by a PCN model. The second experiment, described in section 4.3, performed a computational analysis of both frameworks by collecting metrics on the computational demand of models from both frameworks, looking into the number of flops performed while the models operated. In both cases, the sections are intended to answer three main aspects of the experiment; firstly, which adaptations were made to enable the basic frameworks to cater to the demands of the experiment; secondly, how is the experiment executed; and lastly, which metrics were collected during the experiment.

4.1 Frameworks

Two frameworks were created, which were later adapted to the specific needs of the experiments. The first was a framework for creating ANN models using BP for learning, and the second was for creating PCN models. Both frameworks adhered to a common `learner` interface designed to solve classification tasks. This interface had two important methods: the first is called `train` and is used to train the models, while the second is called `test` for testing the models. The `test` method returns the model's predictions and accuracy and is covered in Alg. 1. A model inheriting the interface would typically first train on the dataset for

a fixed number of epochs by repeatedly calling the `train` method before being evaluated by calling the `test` method. Both methods require a prediction; hence a third vital method was declared, `__predict__`, which differed between the frameworks. The differing details of the `__predict__` method will be given for ANN and PCN in section 4.1.1 and 4.1.2, respectively, along with information about the frameworks. Finally, section 4.1.3 will explain the datasets used for validation during implementation and in the experiments.

Algorithm 1 Pseudo code explaining the test function of the learner interface

function TEST($\boldsymbol{X}$, $\boldsymbol{Y}$)
  $P \leftarrow [\ ]$
  for $\boldsymbol{x'}$ **in** $\boldsymbol{X}$ **do**
    add (self.__predict__($\boldsymbol{x'}$)) **to** P
  end for
  $acc \leftarrow \frac{\sum(\arg\max(P)==Y)}{|Y|}$
  return (P, acc)
end function

4.1.1 Artificial Neural Networks

The framework for the artificial neural networks was set up using the description provided in section 2.3. It implements the `__predict__` method by performing the forward pass that allows the changes at the input layer to propagate through the network, providing a prediction at the output layer. Accordingly, to execute the forward pass, Eq. (2.3) is run repeatedly for each layer of the network, and the prediction is read from the values of the final output layer. Alg. 2 reproduces this functionality.

Algorithm 2 Pseudo code for the forward pass done in the ANN framework, where L_{max} denotes the output layer of the network.

function __PREDICT__($\boldsymbol{x'}$)
  $\boldsymbol{x}^{(0)} = \boldsymbol{x'}$
  for $l = 1$ **to** L_{max} **do**
    $\boldsymbol{x}^{(l)} \leftarrow \boldsymbol{\sigma}(\boldsymbol{x}^{(l)}\boldsymbol{\theta}^{(l)})$
  end for
  return $\boldsymbol{x}^{(L_{max})}$
end function

The `__predict__` method is then reused in the `test` method, as seen in Alg. 1, and in the `train` method. In the `train` method, the ANN produces a

prediction with `__predict__` and uses this prediction in Eq. (4.1), the special case of Eq. 2.11 for the output layer L_{max}, to get the error at the output layer.

$$\delta_i^{(L_{max})} = -f'(S_i)(\hat{x}_i - x_i^{(L_{max})}) \tag{4.1}$$

The error term is then further used to propagate the error margins backward in the net according to Eq. (2.12). Once all deltas have been propagated through the network, the weights are updated according to 2.13. This process is repeated for every sample in the provided dataset, allowing the net to improve its predictions over time. The functionality is summarized in Alg. 3, where $\hat{x}_i$ denoting the label of a sample is replaced by y. For sake of brevity, the for-loops that would iterate through each δ_i and $\theta_{i,j}$ are omitted.

Algorithm 3 Pseudo code for the training of ANN framework. The for loops for the δ- and θ-updates which would increment the i-variable in these equations have been omitted for sake of brevity.

function TRAIN($\boldsymbol{X}, \boldsymbol{Y}$)
  for $(\boldsymbol{x'}, y)$ **in** $\boldsymbol{X}$ **do**
    `self.__predict__`($\boldsymbol{x'}$)
    $\delta_i^{(L_{max})} \leftarrow$ Eq. (4.1)
    for $l = L_{max} - 1$ **to** 0 **do**
      $\delta_i^{(l)} \leftarrow f'(S_i) \sum_{j \in (l+1)} [\theta_{i,j} \delta_j]$
    end for
    for $l = 1$ **to** L_{max} **do**
      $\theta_{i,j}^{(l)} \leftarrow \theta_{i,j}^{(l)} - \lambda x_i \delta_j$
    end for
  end for
end function

For the ANN framework, the activation function used for the forward pass and backpropagation of errors was the sigmoid logistic function. No further extensions were made to the framework, such as batching, where a set number of samples are run before the weights are updated, or soft-maxing the output before the final layer, where the activations are polarized to increase the confidence in the final output to provide enhanced learning. These extensions were omitted to make the comparison between ANN and PCN on the most basic level.

4.1.2 Predictive Coding Networks

The PCN framework bases itself on the description provided in (Whittington and Bogacz [2017]). It performs its inference phase by performing gradient descent on the energy function with respect to the activation of the neurons in the

network. The process includes an iterative updating of the activation and error nodes of the net until it has settled into equilibrium. The inference phase starts with an input presented at the input layer, causing a change in the error node of the second bottommost layer. Consequentially, a cascade of change will ripple throughout the network where activations are sent forward, and error messages are sent backward until the conveyed signals become small enough for the net to be considered equilibrated. An important difference between the implemented framework and the one presented in their paper is that the enumeration of layers has been flipped. In (Whittington and Bogacz [2017]), the input layer was considered as L_{max} and the output layer as layer 0, whereas the implemented framework uses layer 0 as the input layer and L_{max} as the output layer. For the sake of clarity, Eq. 3.1 has been reproduced in Eq. 4.2, showing how the update margins for the activations nodes are calculated.

$$\Delta x_i^{(l)} = -\varepsilon_i^{(l)} + \sum_{j \in (l+1)} \varepsilon_i^{(l+1)} \theta_{j,i}^{(l)} f'(x_i^{(l)}) \tag{4.2}$$

The calculation of errors is given by Eq. (3.2), where the prediction of a node i at layer l, $\mu_i^{(l)}$, is given as the weighted sum of the activations from the previous layer. The equation is slightly adapted from the original framework to ensure the values of the activation nodes stay within the range of the activation function.

$$\mu_i^{(l)} = \sum_{j \in (l-1)} f(\theta_{i,j}^{(l-1)} x_j^{(l-1)}) \tag{4.3}$$

The error nodes are then used to find the update margins for the node activations as defined by Eq. (4.2). Once the net has reached equilibrium, the new prediction can be read from its output layer. Alg. 4 describes how the PCN models created by the framework produce their predictions, and figure 4.1 illustrates a network with two input nodes, one hidden layer with three nodes and two output nodes. The figure also includes examples of the equations used to calculate the value of the error nodes and the update margins for the activation nodes of layer 1.

The last detail of the prediction method for the PCN networks relates to when the network is determined to have converged and reached equilibrium. For this framework, the convergence check was done by using the `allclose`-function from Numpy, which takes as input two vectors, $\boldsymbol{s}$ and $\boldsymbol{s'}$ representing the layers of the network, and returns if the following equation is satisfied or not $|\boldsymbol{s} - \boldsymbol{s'}| \leq (tol_{abs} + tol_{rel} \times |\boldsymbol{s'}|)$. Alg. 5 reproduces this convergence check, and the values used for the tolerances were $tol_{abs} = 1 \times 10^{-2}$ and $tol_{rel} = 1 \times 10^{-}5$. Additionally, a cap, t_{max}, for the timesteps was included to prevent potential endless loops, inhibiting the cycle of updating error and activation nodes from exceeding t_{max}, as seen in Alg. 4.

Algorithm 4 Pseudo code for the forward pass done in the ANN framework, where L_{max} denotes the output layer of the network. The for loops for the ε- and x-updates which would increment the i-variable in these equations have been omitted for sake of brevity.

function __PREDICT__($\boldsymbol{x'}$)
  $\boldsymbol{x}^{(0)} = \boldsymbol{x'}$
  $t \leftarrow 0$
  while True **do**
    $S' \leftarrow$ `self.layers`
    for $l = 1$ **to** L_{max} **do**
      $\varepsilon_i^{(l)} \leftarrow \frac{x_i^{(l)} - \mu_i^{(l)}}{\Sigma_i^{(l)}}$
      $x_i^{(l)} \leftarrow x_i^{(l)} - \varepsilon_i^{(l)} + \sum_{j \in (l+1)} \varepsilon_i^{(l+1)} \theta_{j,i}^{(l)} f'(x_i^{(l)})$
    end for
    if `self.check_convergence(self.layers,` S'`)` **or** $t > t_{max}$ **then**
      break
    end if
    $t \leftarrow t + 1$
  end while
  return $\boldsymbol{x}^{(L_{max})}$
end function

Algorithm 5 Pseudo code for the sub-method used in PCN's `__predict__` method to check if the net has equilibrated. $\boldsymbol{s}$ and $\boldsymbol{s'}$ denotes the vectors representing the layers of the networks stored in the snapshots S and S', respectively.

function CHECK_CONVERGENCE(S, S')
  for $(\boldsymbol{s}, \boldsymbol{s'})$ **in** (S, S') **do**
    if $|\boldsymbol{s} - \boldsymbol{s'}| \leq (tol_{abs} + tol_{rel} \times |\boldsymbol{s'}|)$ **then**
      return True
    end if
  end for
  return False
end function

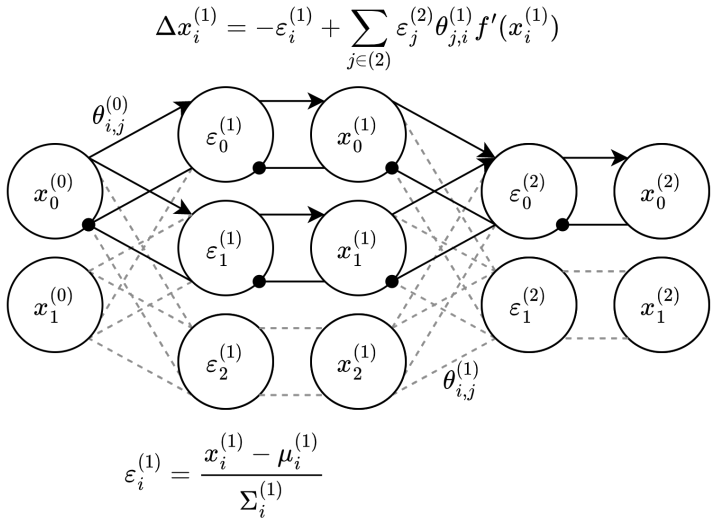


Figure 4.1: Illustration of network structure for a small PCN. Mathematical equations are included for the error nodes and margins of activation nodes at layer $l = 1$. Note that some edges are excluded to avoid cluttering and are instead sketched as grey, dotted lines

Like the ANN framework, the training phase for the PCN framework uses the `__predict__` method before updating the weights. The residual error in the network, meaning the residual values in the error nodes at equilibrium, was used to update the weight matrices of the network with the outer product between the activation of the layers and the error nodes, as captured by Eq. (3.3). A learning rate, λ, was also included here to ensure the steps taken were appropriate regarding gradient descent on the objective function to prevent overshooting. Thus, the update rule for the weights was defined as

$$\Delta\boldsymbol{\theta}^{(l)} = \lambda\boldsymbol{\varepsilon}^{(l+1)} \otimes \boldsymbol{x}^{(l)} \tag{4.4}$$

where $\otimes$ denotes the outer product between the error nodes and the activation nodes. In order to train the network and improve the predictions, the PCN framework clamps both the output and the input. The net is then allowed to run until convergence, and the weights are updated with the margins found by Eq. (4.4). This approach was used to implement the `train` method, as seen in Alg. 6.

Algorithm 6 Pseudo code for the training of PCN framework

```
function TRAIN(X, Y)
    for (x', y) in X do
        x^(L_max) = y
        self.__predict__(x')
        for l = 0 to L_max − 1 do
            θ^(l) ← θ^(l) + Eq. (4.4)
        end for
    end for
end function
```

As with the ANN framework, no further extensions were made to compare the two machine-learning approaches at the most basic level of complexity. For the PCN network, however, tanh was used as the activation function, which casts values between the range $(-1, 1)$ to calculate μ_i and the weight update margins.

4.1.3 Datasets

During the implementation of the framework, the Iris dataset, as mentioned, was used to validate the frameworks to ensure they were learning correctly. An average accuracy over ten agents exceeding 0.8 was deemed acceptable to consider the basic framework complete. Although it has been shown that PCN models and ANN using BP can reach accuracies ≥ 0.95, the chosen threshold was selected to avoid expending unnecessary time training and searching for optimal

hyperparameters. After the frameworks achieved the desired accuracy for the Iris dataset, they were cross-validated against the Wine dataset from UCI. Both datasets were used in the execution of the experiments described in section 4.3 and 4.2, along with MNIST, and all three datasets will be further described in this section.

MNIST

The MNIST database is a labeled dataset of digitally handwritten digits, and is commonly used in machine-learning tasks as an important benchmark. The digits range from 0 to 9 and are represented as a matrix of 28×28 grey scale values in the range of 0 to 255. Together, they compose an image of a digit, as seen in figure 4.2, which shows the plotted image of a 5. The dataset has several applications, including visual image processing or reconstruction training, and is here used to train models of the frameworks to classify the digits. The complete dataset comprises 60 000 training samples and 10 000 test samples, of which 1000 samples from the training set were used for the experiments as preliminary tests showed viable results using a reduced dataset while also cutting the computational time.

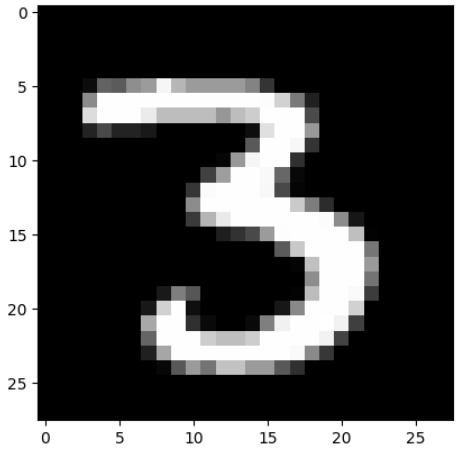


Figure 4.2: Picture of a sample depicting a handwritten '3' from the MNIST dataset produced by plotting the grey scale values into a 28 x 28 grid.

Iris

Used for validation during the implementation of the frameworks, the Iris dataset from UCI (Fisher [1988]) comprises 150 samples. The samples consist of four

attributes: sepal length, sepal width, petal length, and petal width. Based on these, the models predict which species the samples belong to, and the possible species are setosa, versicolor, and virginica. Because of the simplicity of the dataset, it was chosen for the validation, making both computation time low and troubleshooting more straightforward as the required model for classification requires fewer parameters. For the full metadata provided by UCI, please refer to appendix B.

Wine

While the simplicity of the Iris dataset has benefits in terms of rapid testability, it also poses the risk of oversimplifying the model and overlooking vital parts of the implementation. Therefore, the models were cross-validated against the Wine dataset from UCI (Aeberhard [1991]). The dataset comprises 178 samples of 13 attributes that include, but are not limited to, alcohol, color intensity, hue, and total phenols as numerical values. The full list of attributes can be found in appendix C. Based on these attributes, the models should map the inputs to three undefined output classes enumerated from 1 to 3. This dataset posed a more complex learning task relative to the Iris framework. Once both frameworks passed the threshold for validity, the frameworks were considered ready for adaptation to the experiments.

4.2 Inference of Missing Attributes

To research PCNs' ability to handle missing values, as addressed by *RQ1*, an experiment was designed to assert if PCN models can infer missing values in numerical datasets and if the usage of these reconstructed datasets would yield any benefits. The PCN framework needed to be adapted to update and infer information about the input layer from the first hidden layer in the network. The specific details of the adaptation made are covered in section 4.2.1. With the updated framework, a trained model was used to infer the missing values in several datasets of differing degrees of sparsity. Further, the datasets were evaluated by comparing the inferred values to the standard deviation of the respective attribute and class.

Additionally, a k-Nearest Neighbors (KNN) model was set up and used to infer the same missing values. The performance of this model would be used as a benchmark for PCN inferred values. Lastly, the datasets from both approaches were used to train framework models to check their performance against the original test and train split of the used dataset. How the experiment was set up and what metrics were recorded are covered in section 4.2.2 and 4.2.3, respectively.

4.2.1 Adaptations of Frameworks

As the ANN models would only be used to predict the reconstructed datasets from PCN and KNN, no adaptations were made to this framework. On the other hand, the PCN framework needed to convey the errors calculated from the overhead layer and update the values of the specific nodes with missing input values. To update only these nodes, the clamp for the input layer was modified so that non-missing attributes were clamped as before, while missing attributes at the input layer were no longer clamped and left open for updates. With this mechanism in place, the update margins of the input layer are given by Eq. (4.5)

$$\Delta x_i^{(0)} = f'(x_i^{(0)}) \sum_{j \in (1)} \varepsilon_j^{(1)} \theta_{j,i}^{(0)} \tag{4.5}$$

which is based on Eq. (3.1), but with errors nodes of the input layer omitted as no predictions are coming from below to calculate the error term. Accordingly, the models from the framework could also update their beliefs of the missing input attributes while equilibrating. Lastly, the return of the prediction method of the PCN framework was extended with the input layer to provide the predictions of the missing input values.

Building on the theory presented in (Sun and Orchard [2020]), a second extension was made to the PCN framework. As mentioned in section 3.3, introducing a decay term to equations for updating the activations and weights of a network enabled it to generate realistic images of the classes provided at the output layer. The same decay term was added to the equations updating the activations and weights of the network, resulting in Eq. (4.6) and Eq. (4.7), where the activations and weights are multiplied by some small decay-constant, γ where the subscript denotes activation of weight decay, and subtracted from themselves, respectively. The intention of introducing the same decay term was to enable the models to make realistic predictions of missing values in representational data as well.

$$\Delta x_i^{(l)} = -\varepsilon_i^{(l)} + f'(x_i^{(l)}) \sum_{j \in (l+1)} \theta_{j,i} \varepsilon_j^{(l+1)} - \gamma_x x_i^{(l)} \tag{4.6}$$

$$\Delta \theta_{i,j}^{(l)} = \varepsilon_j^{(l+1)} f(x_i^{(l)}) - \gamma_\theta \theta_{i,j}^{(l)} \tag{4.7}$$

4.2.2 Setup

For the execution of the experiment, the input values of a given dataset, D, were masked with progressively larger fractions, starting with 0.01 up to 0.05, creating an array of sparse datasets referred to as $\boldsymbol{D'}$. This fraction of values masked will be referred to as the sparsity of the dataset. Further, a PCN model was trained on D for 30 epochs, using the hyperparameters reported in table 4.1. Afterward,

the model was used to infer the missing attributes in the sparse datasets using the adapted `__predict__` method, and a copy of each sparse dataset was created with the missing attributes replaced by the inferred values from the PCN model. The set of these reconstructed datasets is further referred to as $\boldsymbol{D}_{PCN}$, of which a specific dataset is denoted as $D_{PCN}^{0.01}$, where the superscript denotes the degree of sparsity; the fraction of samples where attributes are missing, which in this case would be 0.01.

Secondly, a KNN Imputer was set up using the class of the same name from the `impute` package of SciKit Learn. The imputer was set up using $k = 5$ and with uniform weights, meaning the imputer would use the mean value for the missing attributes of the five closest samples where the attribute was present, weighting the samples uniformly. Accordingly, the KNN imputer was used to infer the sparse datasets, making a second set of reconstructed datasets, $\boldsymbol{D}_{KNN}$, where specific datasets follow the same naming convention as the PCN-reconstructed datasets regarding sparsity.

Once all missing values of the datasets had been inferred, ten new agents of each framework were set up and trained using the reconstructed training sets. The hyperparameters used for these agents are reproduced in table 4.1, with a change to epochs used for training the ANN agents on the reconstructed datasets based on preliminary performance testing. The epochs used for this training were 500 for the Iris and Wine dataset and 30 for MNIST. After training, the agents were evaluated on the original train and test splits of D. The flow of the experiment is visualized in figure 4.3.

Table 4.1: Hyperparameters used for the agents in the experiments. The listed values were used for the inference and computational analysis experiments if not specified otherwise in the text. The hidden layers are denoted as lists where the elements in the list represent the number of nodes in the respective hidden layer.

Hyperparameter		**Value**
Hidden layers	Iris	[6]
	Wine	[15]
	MNIST	[500 500]
Epochs		
Learning rate		
Σ - *error variance*		4
γ_x - *activation decay*		1×10^{-5}
γ_θ - *weight decay*		1×10^{-6}

4.2.3 Collected Metrics

From the results of the execution of the experiment, two main metrics were collected. The first regards the reconstructed datasets. Given the inferred values in the reconstructed datasets, each value was measured against the respective standard deviation of the class and attribute they belonged to — i.e., the difference between the inferred value and the mean value for the respective attribute and class was divided by the standard deviation of that attribute. Eq. 4.8 reproduces this logic in mathematical notation where α represents an attribute of a class, α' denotes the inferred value by one of the inference schemes, $\bar{\alpha}$ denotes the mean of the attribute for the respective class, and σ_α denotes the standard deviation of α.

$$\sigma(\alpha') = \frac{|\alpha' - \bar{\alpha}|}{\sigma_\alpha} \tag{4.8}$$

Secondly, the experiment aimed to check if the reconstructed datasets improved the accuracy of models using them. Accordingly, the accuracy of the agents was measured using the `test` method described in Alg. 1, which yielded the accuracies listed:

- $\boldsymbol{acc}_{ANN}(\boldsymbol{D}_{PCN})$ - Accuracy of ANN agents on the PCN-reconstructed datasets
- $\boldsymbol{acc}_{ANN}(\boldsymbol{D}_{KNN})$ - Accuracy of ANN agents on the KNN-reconstructed datasets
- $\boldsymbol{acc}_{PCN}(\boldsymbol{D}_{PCN})$ - Accuracy of PCN agents on the PCN-reconstructed datasets
- $\boldsymbol{acc}_{PCN}(\boldsymbol{D}_{KNN})$ - Accuracy of PCN agents on the KNN-reconstructed datasets

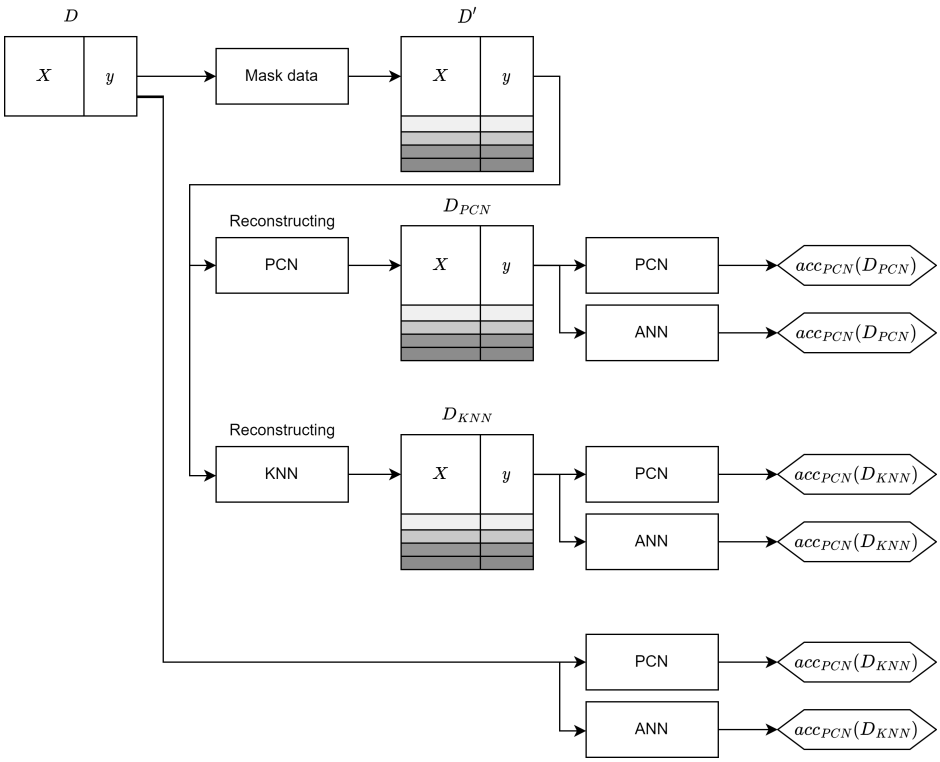


Figure 4.3: Flowchart for the inference experiment. The illustration shows the pipeline of the complete dataset D, from masking data with the sparsity fractions resulting in $\boldsymbol{D'}$, reconstructing them using the PCN model and KNN, yielding $\boldsymbol{D}_{PCN}$ and $\boldsymbol{D}_{KNN}$, and, finally, the evaluation of the agents trained on the reconstructed dataset. The resulting accuracies are denoted by hexagons in the illustration. Note that the final evaluation of the agents was done on the test split from the original dataset with no missing values. The standard deviations of the experiment were calculated based on the reconstructed datasets $\boldsymbol{D}_{PCN}$ and $\boldsymbol{D}_{KNN}$.

4.3 Computational Analysis

The second experiment regarded $RQ2$ and revolved around asserting the computational demands of predictive coding networks and how these demands relate to artificial neural networks using backpropagation. To assert the demands, a third-party library was used to count the number of floating-point operations executed during the training and testing of the two frameworks, a method based on the discussion of Schwartz et al. [2020] covered in section 3.10. The experiment was divided into three parts, each exploring a different approach to training. Section 4.3.2 covers the details of this, preceded by section 4.3.1 discussing the adaptations made to the frameworks. Ultimately, section 4.3.3 will cover the collected metrics from the experiment.

4.3.1 Adaptations of frameworks

For the experiment, no adaptations were made to the frameworks themselves. A wrapper was, on the other hand, implemented for the training of the models from the frameworks, which used a third-party lib, Python PAPI (pypapi), that utilizes hardware counters set up to count the number of flops performed while a snippet of code is running. The wrapper would take in the agent and the number of epochs to train the agent for. Before training the agent, the counter from pypapi was started, which would count the number of flops performed by the agent during training. Upon completion of the epochs, the wrapper would read and reset the counter and return the trained agent along with the flop count.

4.3.2 Setup

The experiment was set up to analyze the flops performed during the training of agents from the two frameworks. Training, however, can take several forms, and three different approaches were settled on for the experiment to see the computational demands from different angles. The first approach allowed the agents to train for the same number of epochs, $epoch_{ANN} = epoch_{PCN}$. The second would allow the model of the two frameworks to, on average, train on the same number of flops, that is, $\overline{\boldsymbol{flops}}_{ANN} = \overline{\boldsymbol{flops}}_{PCN}$. Finally, the third approach would allow the agents to train to the same average accuracy on the test set so that $\overline{\boldsymbol{acc}_{ANN}} = \overline{\boldsymbol{acc}_{PCN}}$.

For all experiments, 10 agents of each framework were set up using the hyperparameters reproduced in table 4.1. Before training the models, the datasets were normalized according to the activation function used by the frameworks. For the ANN, the input values were normalized to the range $(0, 1)$, while for the PCN, they were normalized to $(-1, 1)$. After training, all agents' accuracies were evaluated by testing them against the training and test split. Lastly, the

number of flops was also measured during this final evaluation of the agents. The remainder of this section will cover the details of the abovementioned approaches to training in their respective order.

Equal epochs

During the training of the first approach, the number of epochs the agents trained for remained consistent between the frameworks, and the number of epochs was kept the same between all datasets at 30 epochs of training per agent. The ANN models were trained first, and the PCN models were subsequently trained. Afterward, the models underwent standard evaluation in the same order they were trained.

Equal training flops

The second approach for training would train the agents for the same number of flops. Based on preliminary tests, the PCNs performed the highest number of flops per epoch. Accordingly, the PCN agents were trained first, recording the number of flops each agent performed during training. After that, the average number of flops was calculated based on the recorded flops over the ten PCN agents and used as a stopping condition for the ANN agents. The ANN agents were subsequently trained in intervals of 30 epochs, recording the flops performed. Once the average number of flops over the ANN agents was equal to that of the PCN agents, training of the ANN agents would stop, and the agents would go through the standard end-evaluation.

Table 4.2: Accuracy targets for each dataset used in the accuracy training described in 4.3.2

Dataset	**Accuracy**
MNIST	0.75
Wine	0.8
Iris	0.8

Equal accuracy

Lastly, the approach measuring the flops where the agents would match accuracy involved measuring the agents while they were training. The models of both frameworks were given a shared target accuracy based on initial testing. The specific targets for each dataset can be found in table 4.2 and were used as a stopping criterion. The agents were trained with an epoch stepsize of ten,

$epoch_{step} = 10$, and evaluated against the test set afterward. If the average accuracy of the agents were above the target accuracy, the training of the agents would stop; otherwise, the agents would go on to train for another ten epochs until the average accuracy matched the target accuracy. Lastly, a cap for the maximum number of epochs, $epoch_{max} = 500$ was implemented here in case the agents of a framework plateaued beneath the target accuracy, as seen in Alg. 7, which summarizes how the agents were trained by this approach.

Algorithm 7 Pseudo code illustrating how the accuracy matching was implemented. Assume *agents* is an array of the agents from the framework, acc_{target} is the target accuracy based on the dataset found in table 4.2, $\overline{acc}$ denotes the average over the elements in the array *acc*, and `train_agent` is the wrapper described in section 4.3.1.

```
// ... Setup of agents and preprocessing of the dataset

epoch_curr = 0
while epoch_curr < epoch_max do
    acc = [ ]
    for i in |agents| do
        train_agent(agents[i], X_train, y_train, epoch_step)
        acc[i] ← agents[i].test(X_test, y_test)[1]
    end for
    epoch_curr ← epoch_curr + epoch_step
    if acc̄ > acc_target then
        break
    end if
end while

// Evaluation and saving results ...
```

4.3.3 Collected Metrics

Like the experiment described in section 4.2, this experiment was run once for each dataset presented in section 4.1.3, and for each training approach, the following metrics were recorded:

- Total flops during training
- Train accuracy after training
- Test accuracy after training

- Flops performed during the evaluation

Additionally, the following metrics were recorded for the equal flops and equal accuracy training approaches

Equal flops

- Total number of epochs performed

Equal accuracy

- Total number of epochs performed
- Flops performed during the epoch steps
- Accuracies of each model during training

4.4 Summary

This chapter has presented the experiments designed to explore the research questions posed for the thesis. The first analyses PCNs' inherent ability to handle missing values by applying models to missing attribute problems, and the second looks into the sustainable impact of PCN and compares it to ANN using BP by recording the flops performed. Finally, the metrics collected will be used in the ensuing chapter to analyze and draw a conclusion about the *RQ*s.

Chapter 5

Results and Analysis

This chapter will present the results from the experiments and an analysis of them with the primary objective of presenting key findings that can explain and help conclude the research questions posed in the introduction of the thesis. Firstly, the chapter will address the experiment analyzing PCNs' ability to infer missing values related to their inherent ability to handle them in section 5.1 as addressed by *RQ1*. Secondly, the computational analysis of the two frameworks will be discussed in section 5.2, providing the results necessary to conclude *RQ2*, which looks at how the environmental impact of PCNs compares to ANNs. In both sections, the aggregated metrics will be presented along with relevant analysis before these findings are discussed in the following subsections.

5.1 Inference

Based on the inferred values, the standard deviation was calculated for each class attribute for the given dataset on the representational datasets, Iris and Wine. The inferred samples in the MNIST dataset were plotted into images which will be presented in this section.

Further, the viability of the reconstructed datasets was evaluated by training ten agents of each framework on each dataset and subsequently testing them on the original train and test split. A selection of these results will be presented in section 5.1.1 and discussed in section 5.2.2.

5.1.1 Results

The inference experiment results will bin the following two segments. In the first segment, the standard deviations for each representational dataset will be

presented, along with the reconstructed images of the MNIST dataset. Subsequently, the accuracies of the agents for each dataset will follow in the second segment.

Standard Deviation of Inferred Values

The mean standard deviation of the predicted values in D_{KNN} and D_{PCN} aggregated across all sparsities, attributes, and classes are represented in table 5.1. Standard deviations for each attribute for each class in the Iris dataset are included in table D.3a and D.3b for the PCN and the KNN datasets, respectively, in appendix D. Similarly, for the Wine dataset, the means and standard deviations are included in table D.4a and D.4b of the same appendix for the PCN and the KNN datasets, respectively. For the mean values of the attributes for each class in the original Iris and Wine datasets, please refer to table D.1 and D.2, respectively.

Table 5.1: Average standard deviation in predicted values of the KNN and PCN reconstructed datasets aggregated across all attributes and classes of the dataset. MNIST is omitted as it contains visual and not representational data.

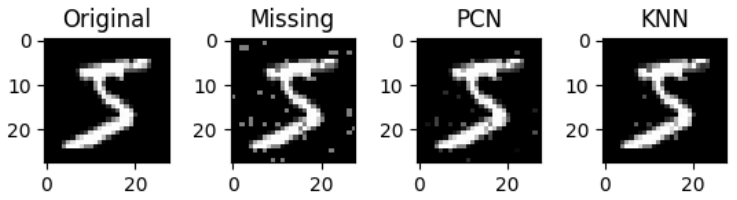
	Iris	**Wine**
KNN	0.717	0.423
PCN	4.236	1.268

For the MNIST dataset, the standard deviation was omitted as it would be more relevant to evaluate the result of the inference visually. Figure 5.1 provides three examples, showing the samples from the original dataset, how they were masked, and how each of the two inference schemes reconstructed the images. For the sake of brevity, three samples are included in this section, with a sample of the remaining output classes included in appendix D.2, figure D.1 - D.7.

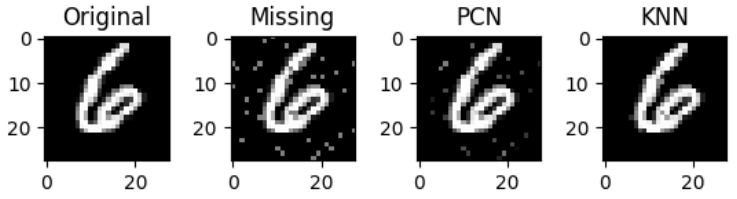
Usage of Reconstructed Datasets

A comparison of the accuracies of the agents of each framework over the Iris dataset is shown in figure 5.2. Appendix D reproduces the accuracies for $D_{KNN}^{0.05}$ and $D_{PCN}^{0.05}$ in table D.5, and points to resources for the accuracies of the other sparsity degrees. Due to low variability in the same metrics across sparsity degrees, the remaining results presented in this section are based on the data collected from the evaluation of $D^{0.05}$.

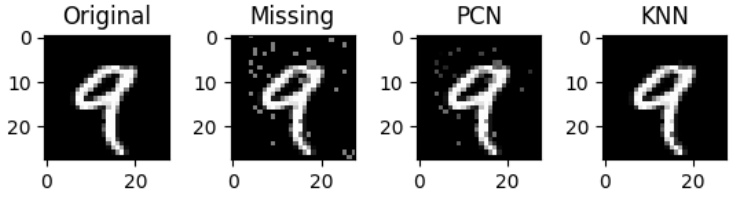
The test accuracies of the ANN and PCN agents on the reconstructed datasets are illustrated in figure 5.3, with the accuracies of the ANN agents trained on $D_{KNN}^{0.05}$ and $D_{PCN}^{0.05}$ shown in subfigure 5.3a and the PCN agents in subfigure 5.3b.



(a) Sample depicting a '5'



(b) Sample depicting a '6'



(c) Sample depicting a '9'

Figure 5.1: Plots of samples from the sparse dataset where 5% of the input values were removed. From the left, the figure shows the original samples before having the inputs masked, the masked sample with the missing values set to the median value of the dataset (128), the reconstructed sample from the PCN approach, and the reconstructed sample from the KNN approach. Subfigure 5.1a, 5.1b and 5.1c show the process for samples depicting a '5', '6' and '9', respectively.

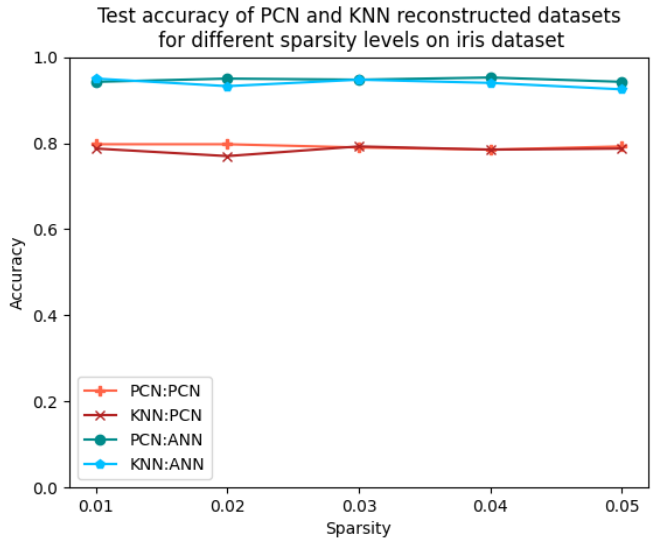
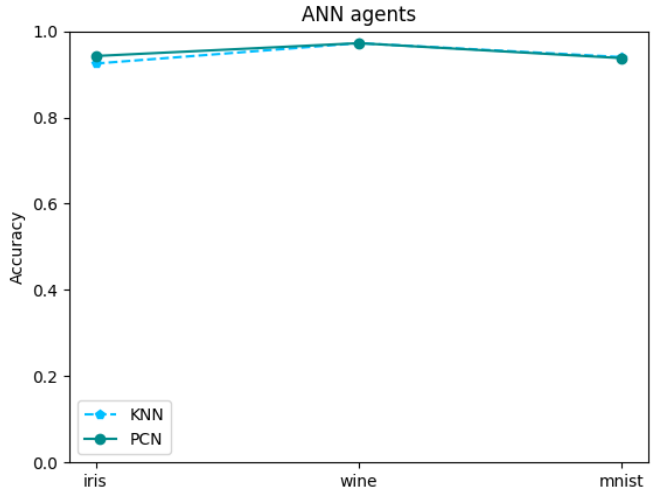
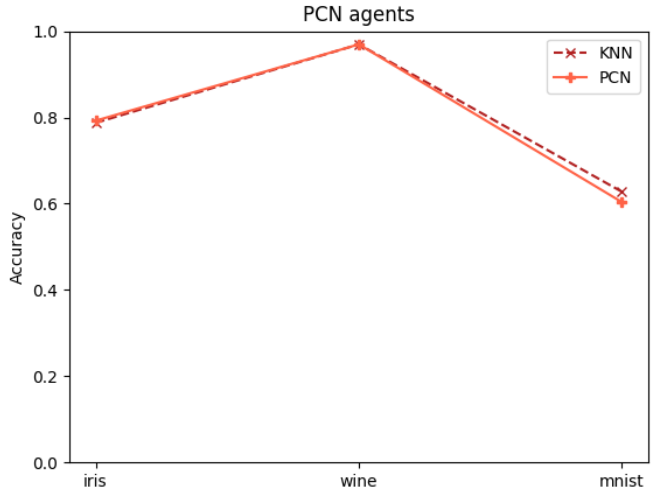


Figure 5.2: Graphs showing mean accuracy of ANN and PCN agents on KNN and PCN reconstructed datasets for the Iris dataset as a function of sparsity. $acc_{ANN}(D_{KNN}^{sparsity})$ is marked by pentagons, $acc_{ANN}(D_{PCN}^{sparsity})$ is marked by circles, $acc_{PCN}(D_{KNN}^{sparsity})$ is marked by $\times$, and $acc_{PCN}(D_{PCN}^{sparsity})$ is marked by $+$. The legend in the image follows an X:Y pattern where X denotes the inference scheme used for the datasets the agents of framework Y were trained on.



(a) Mean accuracy of ANN agents on the reconstructed dataset with sparsity set to 0.05. The accuracy on $D^{0.05}_{KNN}$ is marked by pentagons, and on $D^{0.05}_{PCN}$ is marked by circles



(b) Accuracies of PCN agents on the reconstructed dataset with sparsity set to 0.05. The accuracy on $D^{0.05}_{KNN}$ is marked by $\times$, and on $D^{0.05}_{PCN}$ is marked by $+$

Figure 5.3: Graph showing the accuracy of the ANN and PCN agents on the KNN and PCN reconstructed datasets, with the accuracies for the ANN agents shown in subfigure 5.3a and the PCN agents in subfigure 5.3b.

As a comparison of the agents trained on the PCN-reconstructed dataset and the control agents, metrics are listed in table 5.2, with the accuracies in subtable 5.2a and the respective changes shown in subtable 5.2b. A positive number in subtable 5.2b indicates an improvement, as the margins were calculated as $\overline{acc}_{sparse} - \overline{acc}_{control}$.

Table 5.2: Accuracies and their changes to the sparsely trained agents in the inference experiment. Subtable 5.2a shows the mean accuracies of the control agents for each dataset, and subtable 5.2b shows their respective improvements.

(a) Mean accuracy of the ANN and PCN control agents for each dataset on the train and test split.

	Iris		**Wine**		**MNIST**	
	acc_{train}	acc_{test}	acc_{train}	acc_{test}	acc_{train}	acc_{test}
ANN	0.945	0.945	0.985	0.972	0.940	0.946
PCN	0.812	0.812	0.918	0.972	0.577	0.575

(b) Changes to the accuracies compared to the sparsely trained agents on the PCN reconstructed datasets. A positive number in the table denotes an improvement in the accuracy of the sparsely trained agents.

	Iris		**Wine**		**MNIST**	
	acc_{train}	acc_{test}	acc_{train}	acc_{test}	acc_{train}	acc_{test}
ANN	-0.007	-0.002	-0.020	0.000	-0.003	-0.008
PCN	-0.016	-0.020	-0.006	-0.003	0.019	0.029

5.1.2 Analysis

The following section will address the results of the inference experiment, discussing the results, the limitations of the experiment, and their implications for *RQ1*.

Discussion

Table 5.1 shows that the values inferred by the KNN method were better on both the Iris and Wine dataset. The KNN method managed to predict, on average, the missing values within less than one standard deviation for the respective value on both datasets. The PCN approach performed worse, with the predicted values being an average of 4.236 standard deviations away from the respective mean value for the Iris dataset and an average of 1.268 away for the Wine dataset. The difference here may be explained by the difference in cardinality of the samples in the two datasets. If a value is missing in a sample of the Iris dataset, 25%

of the information on the sample is missing, whereas, in the Wine dataset, only 7.70% is missing. Therefore, Wine samples have proportionally more information the PCN model can use when inferring the missing value compared to the Iris dataset. This explanation for the difference in standard deviation between the datasets is supported by the same trend being observable for the KNN method in table 5.1.

On the MNIST images of digitally written numbers, no numerical metric is given, but the two approaches can be seen to make improvements to the corrupted samples in figure 5.1, and in figure D.1 through D.7 of appendix D. The KNN-reconstructed images display closer similarity to the original sample here as well compared to the PCN-reconstructed image, further supporting the trend of KNN outperforming PCN observed for the Iris and Wine dataset in table 5.1. For this difference, two possible explanations are theorized. Firstly, it could be that the PCN model cannot utilize the information of adjacent pixels effectively and therefore does not completely reconstruct the image and only makes a general improvement in the sample based on the average of the samples. This can be seen in figure 5.1 where the PCN should have been able to use the adjacent pixels in the black areas outside of the nine to infer that the corrupter pixels should be zero as well. Secondly, the PCN model settles into equilibrium, and the used convergence detection algorithm may not be sensitive enough to capture the minor discrepancies that remain in the reconstructed samples. Still, the samples are recognizable as their respective output class for both humans and, as the evaluation of the reconstructed datasets shows, for the models as well.

Lastly, before moving on to the evaluation of the agents trained on $D_{KNN}^{0.05}$ and $D_{PCN}^{0.05}$, an advantage of the PCN approach for inferring missing values is how they are used afterward. While the KNN might perform better, as seen for all three datasets, it relies on an external algorithm to make sample predictions. The PCN approach can reuse the model used to infer the missing values and predict new samples. This finding suggests that the PCN model is able to use the bilateral connections to not only drive predictions forward from input to output but also to convey information backward by the layers learning to predict the activation of upstream layers by observing the activation of downstream layers, presenting an ML model capable of predicting and inferring data in both network directions.

The accuracy graphs of figure 5.2 reveal that the sparsity of the masked dataset did not affect the accuracy of the agents, as evidenced by their low variability. Several reasons may cause this; first, the sparsity of the datasets used for training may still be low enough that the remaining undisturbed samples were sufficient for the models to learn the patterns in the dataset and consequentially not suffer from the masked samples. Secondly, it could indicate that the agents were able to use the inferred values efficiently to solve the classification task for both $D_{KNN}^{0.05}$ and $D_{PCN}^{0.05}$ despite the analysis of the standard deviation for $D_{PCN}^{0.05}$

shows they did not closely resemble their respective mean.

As a reminder, all results presented after figure 5.2 in section 5.1.1 are based on the evaluation of the agents on $D_{KNN}^{0.05}$ and $D_{PCN}^{0.05}$. Hence the remaining discussion presented in this segment is also based on the data collected from evaluating the reconstructed datasets at 0.05 sparsity.

Looking at the performance of the ANN and PCN agents trained on $D_{KNN}^{0.05}$ and $D_{PCN}^{0.05}$ in subfigure 5.3a and subfigure 5.3b, no advantage of the agents trained on one reconstructed dataset can be established over agents trained on the other. This observation can have two possible explanations, one of which relates to the previously discussed limitation of low sparsity and the performance degradation being negligible as a consequence. The other could be that using the PCN-reconstructed dataset results in a comparable performance despite the difference in the standard deviation of the predicted output values to the KNN approach shown in table 5.1. If the latter is the case, the finding also indicates that using PCN models to infer missing values is viable for subsequent use of PCN agents and potentially for other ML models, as the ANN performed equally well on the PCN and the KNN reconstructed datasets.

The control accuracies were used as a benchmark to compare with the accuracies of the agents trained on the reconstructed datasets, and the margins presented in subtable 5.2b and table D.6 of appendix D shows minor changes to the accuracies between the agents. The explanation also lends itself to the raised argument of low sparsity causing negligible degradation in accuracy, or the agents were able to utilize the inferred values of both reconstructed datasets well enough for there to be no observable difference to the control agents beyond expected statistical variability.

Limitations

This segment will address the limitations of the experiment. First, it will explain low sparsity's impact on the observed results. Afterward, how the cardinality of the dataset and, consequentially, the collected metrics affect the quantitative differences will be discussed. Finally, the last limitation concerns the applicability of the results to real-life situations of ML problems.

The sparsity level has been a reoccurring explanation for the presented observation of the evaluation of the agents trained on the reconstructed datasets. With a low sparsity fraction, the masked values may be too few to impact the agents' performance. As such, the effect of masking the values becomes indetectable in the results, causing the research to be inconclusive. The percentage of masked data was not increased due to preliminary testing that indicated unusable outcomes for the Iris dataset at higher percentages. This could be partly attributed to the dataset's small size and low attribute count, resulting in the loss of too much information for the agent to comprehend the data patterns that

link input to output. Consequentially, the agent's ability to reconstruct samples in the opposite direction is also impacted.

The experiment was limited due to a few samples in the dataset linked to both the dataset and the sparsity fractions. This low cardinality of samples significantly affected the quantitative differences observed in the results. With a small number of samples, it becomes challenging to extrapolate trends from the data due to increased variance. An example illustrating this issue is the Iris standard deviation in table D.3a. In this table, two outliers are present with values of 10.220 and 17.811, while the remaining values range between 0.132 and 3.433. Comparing these values to the mean standard deviation presented for the PCN approach in table 5.1, 4.236, it becomes clear that these outliers significantly inflate the mean, causing it to surpass the range of the majority of the values in the table. This observation demonstrates how poor performance on one class attribute reduces the competitiveness of the PCN approach. The low sparsity fraction contributes to the issue by further decreasing the samples which are eligible for evaluation, as only missing samples are included in the calculation of the standard deviations.

Lastly, the final limitation of the experiment regards the methodology and its applicability in real-life ML problems. As described in section 4.2.2, the PCN model responsible for reconstructing the masked value in the dataset was trained on the complete dataset with no missing values before making the inferences. This was done because the main focus of the experiment was to assert if a PCN could infer missing values in representational data. Typically, datasets used in ML problems are incomplete, and the PCN model cannot automatically be trained on the complete dataset as in the experiment. Either the dataset needs to be reduced to only complete samples, sacrificing the size of the training set, or the model would have to train with the missing values, improving the prediction of the missing values as the agent trained on similar samples.

Implications for *RQ1*

The results have shown that PCN models can infer missing values in incomplete datasets with representational data. The values in table D.3 and D.4 of appendix D show standard deviations for the PCN approach that are comparable to the KNN approach. Consequentially, a qualitative answer to *RQ1* can be given, and the experiment has asserted that PCNs are not better equipped at handling missing values than ANN if compared to a typical inference scheme used by ANNs such as KNN.

A quantitative answer was asserted by evaluating the reconstructed datasets on agents of both frameworks. However, due to the discussed limitations, the results remain inconclusive and further research needs to be conducted to conclude a quantitative answer to *RQ1*.

5.2 Computational Analysis

By running the computational analysis on agents of the two frameworks, metrics on the number of flops performed during training and accuracies were collected. The following section will present these results in section 5.2.1 before discussing the differences between the two frameworks in light of the presented results in section 5.2.2.

5.2.1 Results

The results of the computational analysis experiment will be presented in the following three segments. The training flops and post-training accuracies will be given for equal epochs, equal flops, and equal accuracy training approaches described in section 4.3.2, along with figures of the flops plotted against each other and over time.

Equal Epochs

Given the same number of epochs to train on, the PCN models performed more flops over the same training period than the ANN models. The trend is summarized in the bar chart of figure 5.4, showing the PCN training flops as orange bars to the right and ANN training flops as blue bars to the left for each dataset. Further, the exact values of the flops performed during training are reproduced in table E.1 of appendix E.

Secondly, table 5.3 reproduces the accuracies from the post-training evaluation of the agents, displaying the means over the ten agents of each framework for the train and test split.

Table 5.3: Table reproducing the mean accuracy of the ANN and PCN agents per dataset from the equal epoch training approach ($epochs = 30$) of the computational analysis experiment.

	Iris		**Wine**		**MNIST**	
	acc_{train}	acc_{test}	acc_{train}	acc_{test}	acc_{train}	acc_{test}
ANN	0.364	0.250	0.401	0.389	0.768	0.762
PCN	0.793	0.788	0.923	0.931	0.713	0.706

Equal Flops

For the approach with equal number of flops performed during the training of the agents, the aggregated metrics are displayed in table 5.4, with the mean

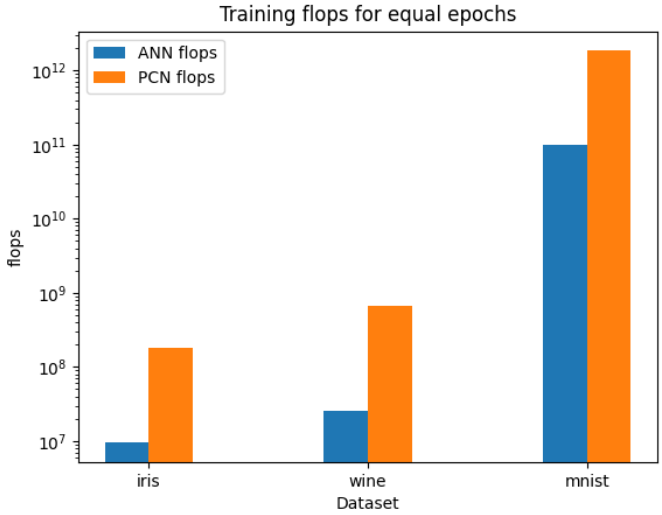


Figure 5.4: Performed flops during training of models on each dataset for the equal epochs training approach ($epochs = 30$). The figure shows the average flops performed over the ten agents used in the experiment, with the PCN models shown as orange bars and ANN as blue bars.

accuracies from the post-training evaluation contained in subtable 5.4a and the number of epochs needed to achieve these results shown in subtable 5.4b. The average number of flops performed by the PCN agents, which became the stopping condition for the ANN models, were 1.649×10^8, 6.652×10^8, and 1.835×10^{12} for the Iris, Wine, and MNIST dataset, respectively.

Equal Accuracy

Given a shared accuracy goal, the models produced the bar chart shown in figure 5.5, with information about the number of epochs per dataset shown in table 5.5.

Further, the intermediate accuracies recorded during training for the Iris dataset are visualized in figure 5.6 for the ANN models and in figure 5.7 for the PCN agents. Additionally, figure 5.8 displays the number of flops performed per epoch-step during training on the Iris dataset.

Table 5.4: Mean accuracies over the ANN and PCN agents for the train and test split for each dataset from the experiment with equal flops performed during training, shown in subtable 5.4a, along with the number of epochs to achieve the accuracy, shown in subtable 5.4b.

(a) Mean accuracies over the train and test split of the datasets

	Iris		**Wine**		**MNIST**	
	acc_{train}	acc_{test}	acc_{train}	acc_{test}	acc_{train}	acc_{test}
ANN	0.873	0.785	0.998	0.972	0.974	0.970
PCN	0.775	0.745	0.922	0.925	0.707	0.693

(b) Number of epochs training was performed over

	Iris	**Wine**	**MNIST**
	epochs	*epochs*	*epochs*
ANN	540	780	570
PCN	30	30	30

Table 5.5: Epochs performed during training for the equal accuracy training approach. Shows the epochs performed by models of the ANN and PCN framework, respectively, for each dataset. Note that while the ANN agents can be seen to perform more epochs than the PCN agents, the number of flops performed per epoch by the ANN agents is lower than by the PCN agents, meaning a higher epoch count does not necessarily equate to larger computational demand.

	Iris	**Wine**	**MNIST**
	epochs	*epochs*	*epochs*
ANN	500	170	30
PCN	210	20	10

5.2.2 Analysis

The following section will address the results of the computational analysis, discussing the results, the limitations of the experiment, and their implications for *RQ2*.

Discussion

From the analysis of the recorded metrics, the training flops performed by the PCN models were an order of magnitude larger than those performed by the ANN models. This trend was observed for the equal epochs and equal accuracy training approaches, where the agents were allowed to train for differing amounts of flops,

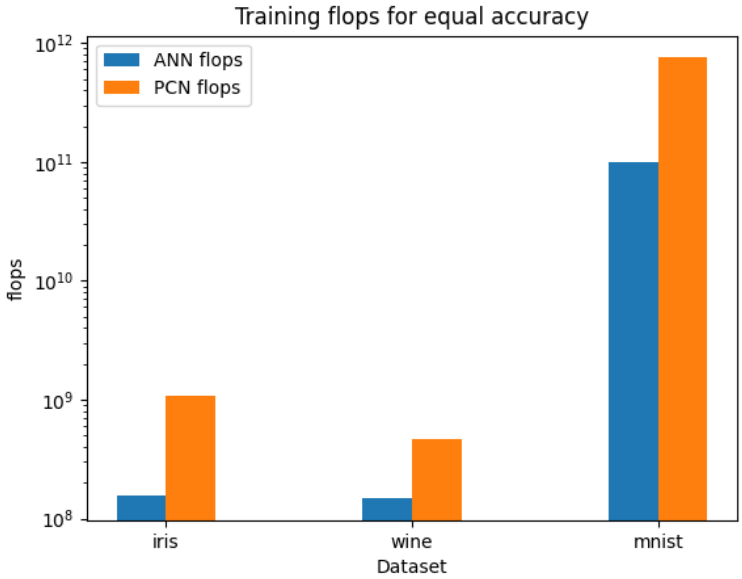


Figure 5.5: Performed flops during training of models for the equal accuracy training approach. The figure shows the average flops performed over the ten agents used in the experiment, with the PCN models shown as orange bars and ANN as blue bars.

as seen in figure 5.4 and figure 5.5. The difference in the order of magnitude between the recorded flops performed,

$$\log_{10} flops_{PCN} - \log_{10} flops_{ANN}$$

supports this trend and reveals a 1.318 difference for the equal epoch approach and a 0.738 difference for the equal accuracy approach on average over the datasets — the difference between the training approaches being caused by the ANN agents being required to achieve the target accuracy and thereby performing more epochs and not stopping at $epochs = 30$ in the equal accuracy approach. Although the PCN agents outperformed the ANN agents concerning accuracy in the equal epoch approach, the results from the equal accuracy approach prove that the ANN models remain more computationally efficient at the same accuracy performance as the PCN models.

Looking from a different angle, where the agents were granted the same computational budget, the ANN models also outperformed the PCN models. The

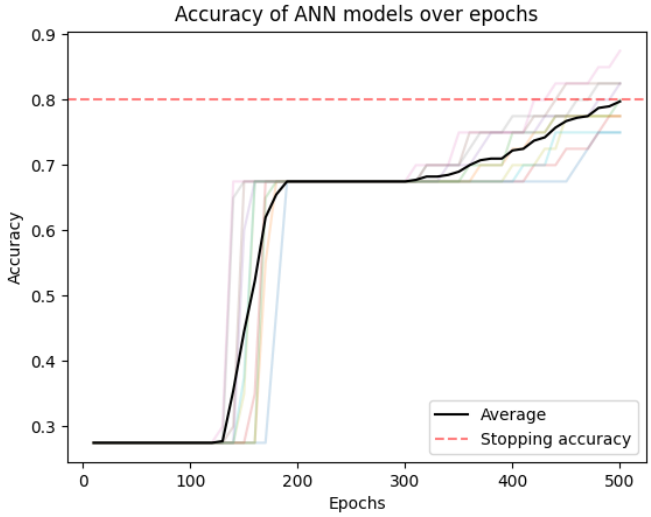


Figure 5.6: Accuracy of ANN models on test split of Iris dataset as a function of the training time over epochs. The black line displays the average accuracy over the ten agents used in the experiment. Accuracies of the individual models are also included, albeit with a lower opacity, to avoid cluttering the graph.

accuracies shown in subtable 5.4a confirm this advantage and show that the accuracies achieved by the PCN lag slightly behind those of ANN for the representational dataset with approximately 5% on the test split. For the visual data of the MNIST, the PCN models were outperformed by the ANN with 26.7% and 27.7% for the train and test split, respectively. Further, the number of epochs also reproduces the trend with an order of magnitude difference between the two frameworks, as seen in subtable 5.4b, which is to be expected as the ANN models would perform one-tenth of the flops from the PCN models, hence needing roughly ten times as many epochs to achieve the same number of flops.

The results from the equal accuracy training approach further establish the advantage of the ANN framework over the PCN framework by showing the ANN models arrive at the target accuracy quicker than the PCN models regarding the flops performed during training. Figure 5.5 displays the flops performed per dataset to achieve the accuracies used as stopping conditions for each dataset. Like figure 5.4, it again shows lower flops performed by the ANN models than the PCN models, albeit with slightly lower flops as the models were allowed to stop before 30 epochs had passed if the average accuracy over the agents had surpassed

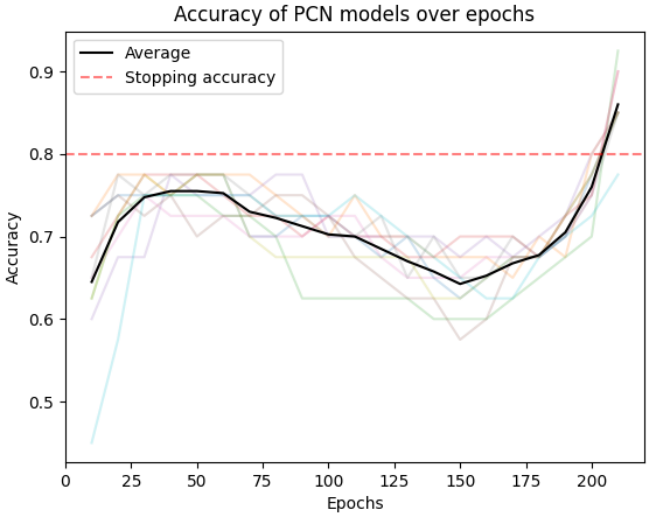


Figure 5.7: Accuracy of PCN models on test split of Iris dataset as a function of the training time over epochs. The black line displays the average accuracy over the ten agents used in the experiment. Accuracies of the individual models are also included, albeit with a lower opacity, to avoid cluttering the graph.

the threshold. This finding shows that to achieve the same level of accuracy, the flops performed by the PCN models will be, on average, 738% higher than those performed by the ANN models.

Secondly, the equal accuracy approach also showed a correlation between the flops performed during prediction and the accuracy of the PCN models. As seen in 5.8, the graph increases at $epoch = 180$, being relatively constant before that point. This increase signifies the number of flops performed during the epoch step-size increases. Seen in relation to the graph of accuracy over the same training, a similar upswing in the graph can be observed at the same point in figure 5.7. The reason behind this correlation in the data is not known. Still, one explanation might be that the increase in accuracy requires more finetuning of the network before settling into an equilibrium. Looking at Eq. (3.2) could hint to the variance being an issue for this significant increase in flops. If the variance is chosen too low, it could result in the updates made in the activation nodes of the network being too big concerning the gradient descent of the energy function of the network with regard to the activation nodes. Should this be the case, the network may end up oscillating back and forth between the local minimum

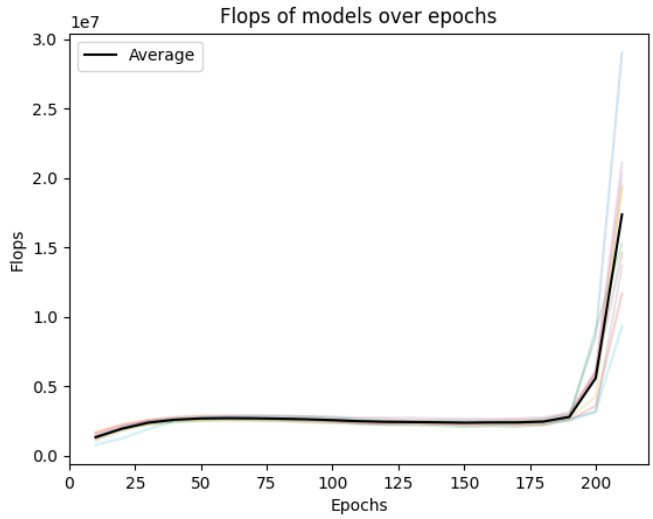


Figure 5.8: Flops performed of PCN models as a function of the training time over epochs on the Iris dataset. The black line displays the mean flop over the ten PCN agents used in the experiment. Flops performed by the individual agents are also included, albeit with a lower opacity, to avoid cluttering the graph.

of the sample for the energy function, resulting in an increase in the number of computations performed and increasing the flop count.

Why this would happen towards the end of training, however, remains unknown. One theory might be that two of the classes in the Iris dataset have very similar input values, and the agent learns to first distinguish the last class from the other two, making only rough calculations. Later in the training period, the agent could perform more precise calculations, but because the variance in the error nodes is set too low, the agent overshoots and has to perform many cycles to equilibrate.

Limitations

Regarding the limitations of the experiment, several concerns have been identified. The first regard the metrics collected and the relevance to the environmental impact of the framework. The second revolves around integrating the 3rd party library used to collect the metrics. Further, low sampling frequency leads to few data points in the collected metrics, and, lastly, the normalization scheme used during the experiments. The following segment will address these limitations in

their listed order.

Firstly, one limitation of the experiment regards if the collected metric accurately represents the environmental impact of the model. Flops can give an idea of an ML model's computational demands and power usage. Still, they do not constitute the entire picture of the power usage and environmental impact of it. Other metrics relate to the power usage of an ML model, as discussed in (Schwartz et al. [2020]). One such metric is memory migrations which address how data is moved between different levels of memory. The number of memory migrations can vary significantly based on how the frameworks and models are implemented and on which systems they are implemented. Flops, on the other hand, remain consistent as they only target the operations done to floats by the model. Hence, they serve as a metric that makes comparing models across different operating systems and implementations possible. They can indicate which model is more efficient but not a definitive answer to which model has a lower environmental impact.

Secondly, a limitation of the experiment's reproducibility regards the integration of the 3rd party library used. The pypapi library utilizes hardware performance counters from the Performance Application Programming Interface (PAPI), which is operable on Linux/Unix-type systems. As such, if the system the models are run on does not support PAPI or the hardware does not contain the hardware counters, the experiment will not be reproducible.

Thirdly, the sampling frequency used for the Wine and the MNIST datasets during the equal accuracy training approach proved too low. With $epoch_{step} = 10$, the agent trained in 10 epoch intervals. As such, the ANN and PCN framework had three data points and one data point for the MNIST dataset for ANN and PCN, respectively, as is implied by table 5.5, because the agents achieved the target accuracy by the completion of these first few epoch steps. Similarly, PCN has two data points on the Wine dataset. Consequentially, extrapolating any trend from the collected data becomes problematic when the data points are few. Therefore, the findings regarding the upswing in flops and the correlation between performed flops and accuracy for the PCN framework, as seen in figure 5.7 and figure 5.8, cannot be confirmed across the different datasets.

Lastly, regarding the normalization scheme used for the experiment, the train and test split were normalized separately. In datasets, the test and train split should ideally be representative of each other so that the min- and max-values used to normalize the dataset should be similar. Should this not be the case, however, the difference in the extremal values could cause two different values to map to the same normalized values. This error was corrected, and the experiment was rerun for the Iris and Wine dataset, reproducing accuracies approximating the ones represented in table 5.3 and subtable 5.4a, and was therefore not considered a major limitation of the experiment. Due to time constraints, however, the

experiment was not rerun for the MNIST dataset, and it could explain the difference in the accuracies observed for the MNIST dataset in subtable 5.4a between ANN and PCN.

As a side note, this phenomenon of train and test split that do not represent each other well may also be the cause of the abnormal shape of the test-accuracy graph in figure 5.7 where the accuracies are actually seen to decrease from around epoch 60 to 150. A more typical learning graph can be seen for the ANN agents in figure 5.6 where the accuracy monotonically increases. However, because these graphs are based on the accuracy of the test split, the direction taken by the PCN agents between 60 and 150 could be in the right direction with respect to the observed samples of the train split but wrong for the test split.

Another possible explanation for the difference in accuracy between the agents of each framework on the MNIST dataset and an error in the normalization scheme was that the MNIST samples were normalized to be between 0 and 1 and subsequently used by both frameworks regardless of their activation function. Hence only half the range was used for the PCN models with respect to the utilized activation function, tanh. This would affect how the models differentiate between the black and white pixels in the original sample and, thereby, the model's accuracy. Consequentially, the implications of the results for the research questions could be skewed.

Implications for *RQ2*

As discussed by the limitations, looking at only the flops to address the environmental impact of an ML model does not constitute the entire picture. However, flops can address the computational demands of a framework and thereby also indicate the power consumption and environmental impact of the models. They also allow for easy comparisons between different frameworks and, based on the collected metrics and their presented analysis, do not support a computational advantage of PCN models compared to ANN using BP.

The difference in computational demand between the two models is caused by the way PCNs perform their prediction. While ANN using BP only needs to perform the calculations for a node once to land on a prediction, PCN performs calculations several times during the iterative equilibration process. While potential improvements could be made to the convergence detection algorithms that reduce the number of cycles the PCN models need to equilibrate when performing predictions, the computational demand will still likely be higher than ANN because it has to calculate the node values iteratively.

A definitive answer to the research question cannot be given because, as mentioned, flops do not capture the complete picture of the environmental impact of an ML model. However, the results establish an advantage of ANNs using BP over PCNs concerning computational demands, which also suggests a benefit

regarding environmental impact, as the two aspects are linked through power consumption and CO2 emissions.

5.3 Summary

The following section has covered the results of the experiments, presented an analysis of them, discussed the limitations of the experiments, and lastly, addressed the implications of the findings for the research questions.

For the inference experiment, the results have shown the ability of PCN to infer missing values in representational data and partially reconstruct visual data. Based on the standard deviation of the inferred values compared to those of the commonly used benchmark, KNN, it can be quantitatively concluded that the inferred values by the PCN model are not as strong, indicating a negative response to *RQ1*. Secondly, the experiment also tried to quantitatively assert the difference in the performance of agents trained on the reconstructed datasets. Due to the discussed limitations, however, the research remains inconclusive.

The second experiment performed a computational analysis between the frameworks by recording the number of flops performed by agents of each framework. Three different approaches, looking at the performed flops from different angles. All three approaches suggested an advantage of the ANN agents, two of which showed fewer flops performed by the ANN agents. The last showed higher accuracy of the ANN agents over the same number of flops performed during training. The findings established a higher computational demand for PCN agents and performance differences favoring ANN. Consequentially, the findings suggest that the environmental impact of PCN is higher than ANN, as addressed by *RQ2*, although further research needs to be done to address the complete environmental impact of PCN as an ML model.

Chapter 6

Conclusion

6.1 Thesis Review

The first chapter of the thesis introduced the research questions, which have served as a connecting thread throughout the thesis and the motivation behind them. The first research question addresses PCNs' inherent ability to handle missing values and asks if PCNs are better equipped at handling missing values than ANNs using BP. The second research question revolved around the sustainability of ML models and queries how PCN's environmental impact compares to ANN's.

Chapter 2 lays the theoretical background to understand the frameworks implemented for the thesis. The chapter visits the biological issues of ANNs, listing three main issues, namely the issue of error representation, the issue of symmetrical weights not being ubiquitous in the human brain, and the issue of transmission of signals between neurons and finding a derivate of the spikes found in biological neurons. Further, the chapter visits the theory behind ANNs using BP and derives the equations for generating predictions from the forward pass and update rules to improve the network's accuracy. Lastly, the chapter ends with an introduction to the main ideas behind PCN and briefly explains how ANNs and PCNs relate to each other.

Moving on to the related works of the thesis, chapter 3 picks up where chapter 2 leaves off by elaborating on and providing a specific framework for PCN presented by Whittington and Bogacz (Whittington and Bogacz [2017]) that implements the main ideas presented in the background. Additionally, the chapter explores other articles that have researched similar topics relevant to the RQs. These articles include ones that show how PCNs can be made to provide realistic predictions of input data on visual datasets, how to evaluate the computational

demand of ML methods, and how other frameworks have addressed the last two biological issues of ANNs.

The Methodology, chapter 4, presents the frameworks implemented for the thesis, one for creating ANN models and one for creating PCN models. The chapter further explains two experiments that use the frameworks to address the *RQs*. The first experiment explored PCNs' capability to infer missing values and how subsequent use of these inferred values affects the performance of ML models. The second experiment analyzed the computational demand of the two ML frameworks by recording the number of flops they performed during training.

Lastly, chapter 5, Results and Analysis, reproduces the results from the collected metrics of the experiment, and presents an analysis of them and their implications for the research questions. For the inference experiments, the results presented include the standard deviation of the inferred values in the representational datasets, visualization of the reconstructed images of visual data, and a comparison of the accuracy of agents utilizing the reconstructed datasets for training. The results of the computational analysis experiment focused on the flops performed and the resulting accuracies of the agents.

This final chapter will conclude the thesis, addressing the contributions of the thesis research to the research questions and the field in section 6.2, and discussing future directions the area may take in section 6.3.

6.2 Research Questions

Two frameworks have been created for the thesis to contribute new information to the research field and question. The first is a framework to set up ANN models, and the second is to set up PCN models for classification tasks. Further, two experiments have been designed to gather quantitative data related to the research questions. For both experiments, information from related articles in the field has been used, and the following two subsections will briefly discuss the adaptations, the key findings from the respective experiment, and the implications these findings have on the research questions.

6.2.1 *RQ1*: PCNs' Ability to Handle Missing Values

In the first experiment, the PCN framework was extended to use activation and weight decay. This extension was made based on the theory presented in (Sun and Orchard [2020]), which shows how PCN can generate realistic input images provided a label at the output layer on datasets containing visual data. The transferability from visual data to representational data was tested in the first part of the inference experiment. Qualitatively, the results found that the PCN model could create predictions of the missing values. Quantitatively, the standard

deviations reveal that using PCN to infer missing values lags behind KNN, which was used as a benchmark in the experiment.

The second part of the experiment could not establish the advantage of using the PCN to the KNN reconstructed dataset as the resulting accuracies were too close to the control agents trained on the complete dataset. The sparsity fraction used in masking the data has been identified and discussed as the main limitation, causing the number of samples affected to be ignorable for the agents, causing inconclusive results from the second part of the experiment. Low sparsity fractions were, however, seen as necessary to uphold the patterns in the Iris dataset to test the inference capabilities of the PCN agents based on preliminary testing.

Regardless the results have established that PCNs can be used as an inference scheme for missing values in incomplete datasets. Concerning the research question, this indicates that PCNs are better equipped to handle missing values than ANN alone. Still, the quantitative results show that it does not beat typical methods used with ANN to manage missing values.

6.2.2 *RQ2*: Computational Demand of PCN Compared to ANN

The second experiment builds on the positional paper by Schwartz et al. [2020], advocating for using flops as a metric to report the computational demand of ML models. While the computational demand of a model does not constitute the complete environmental impact, as discussed in the experiment's limitations, it can give an indication of the environmental impact. The results show for all three training approaches tested an advantage in favor of ANNs over PCNs, qualitatively indicating that PCNs have a higher environmental impact than ANNs, supported by the quantitative result showing that the flops performed by the PCN models, on average, were 738% higher than those by the ANN models given the same target accuracy.

This aspect was researched because it was deemed possible for ANNs to have a higher computational demand than PCNs because of the complex error gradients that must be propagated backward in the network for each node during training. PCN, on the other hand, performs the error gradient calculations locally and using a simpler calculation scheme. However, due to the equilibration process, they still perform more flops than ANNs, and finding an effective way of reaching equilibrium proved difficult both in terms of not overshooting the adjustments and choosing the right convergence detection algorithm. Similarly, for the inference scheme, choosing the correct values for both variances of error nodes, convergence detection scheme, weight decay, and activation decay proved difficult to ensure the PCN agents performed optimally.

6.3 Future work

The following section will revisit some of the raised limitations of the experiments and new directions for future work within the research field. Subsection 6.3.1 will address the limitations of the inference experiment, visiting the issue with sparsity fractions and how it can be established if PCN models are a viable approach to fill in missing values in datasets. Subsequently, subsection 6.3.2 will address the concerns raised regarding the computational comparison between the two frameworks and other future directions that came to light through the experiment.

6.3.1 Inference of Missing Values

For the inference experiment, one of the main theorized limitations of the second part of the experiment relates to the sparsity fraction. Because of the percentage of masked values, the agents could likely ignore the missing values and still perform their predictions correctly, as evidenced by the marginal improvements presented for the agents trained on the PCN- and KNN-reconstructed datasets. It would be interesting to repeat the experiment with higher sparsity fractions to see if it would impact the agents' accuracies and the standard deviations of the inferred values. Additionally, a second set of agents trained on the missing dataset could be integrated, giving an estimator of the expected degradation of the agents against which the reconstructed-trained agents could be compared.

Secondly, a side-by-side comparison of the performance between the inferred values from PCN and KNN would also be interesting to investigate. Increasing sparsity fractions could be used, and using the standard deviation of the inferred values as a metric, the experiment could research the performance of the inference schemes as a function of the completeness of the employed datasets.

6.3.2 Environmental impact

While the analysis of the training flops for the second experiment suggests a higher environmental impact of PCN over ANN, other metrics also affect the footprint of machine learning models. Further research into metrics such as memory migrations can help conclude the second research question. Secondly, it would also be interesting to compare the power usage of PCN in embedded systems and if PCNs are implementable on organic computing structures further down the line.

Lastly, the discussion of the results of the second experiment addressed a link between the accuracy of the PCN network and the flops performed during each step size. It would be helpful to investigate if the same link also holds up for other datasets. The connection could not be established in the current

experiment due to a low number of data points for these datasets. To combat this, a future experiment could reduce the epoch step size used for the equal accuracy approach, thereby increasing the number of data points over the number of epochs trained for. Secondly, the target accuracy could be increased to encourage more epochs to be performed during training, thereby producing more intermediary accuracies. Additionally, the research could also check if the hypothesized cause of this increase, the variance in the error nodes being set too low, is true. If this increase could be avoided, it could help make PCNs more environmentally friendly.

6.3.3 Final Remarks/Parting Thoughts

Revisiting the overall research goal of investigating if PCNs can be a more biologically plausible alternative to ANNs mentioned in the introduction, PCN does show promise as a unified model of the brain. Qualitatively the study illustrates that PCNs are able to infer values in incomplete datasets, something ANNs are not capable of alone. It is, however, more uncertain of PCNs will replace ANNs. The findings from the computational analysis indicate that PCN lags behind ANNs both with respect to accuracy and computational demand.

Regardless, the research field is still in development, as the related works have shown with other models addressing the other three biological issues of ANNs that PCN might utilize in the future and with findings that show the scalability of the framework to more complex machine learning problems.

Bibliography

Aeberhard, Stefan & Forina, M. (1991). Wine. UCI Machine Learning Repository. DOI: 10.24432/C5PC7J.

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. (2015). Towards biologically plausible deep learning. arXiv preprint arXiv:1502.04156.

Downing, K. (2023). Gradient expectations. Publisher: MIT Press.

Fisher, R. (1988). Iris. UCI Machine Learning Repository.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. science, 313(5786):504–507.

Liao, Q., Leibo, J., and Poggio, T. (2016). How important is weight symmetry in backpropagation? In Proceedings of the AAAI Conference on Artificial Intelligence, volume 30.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. Nature communications, 7(1):1–10.

Millidge, B., Tschantz, A., and Buckley, C. L. (2022). Predictive coding approximates backprop along arbitrary computation graphs. Neural Computation, 34(6):1329–1368. Publisher: MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ....

Nøkland, A. (2016). Direct feedback alignment provides learning in deep neural networks. Advances in neural information processing systems, 29.

Rao, R. P. and Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nature neuroscience, 2(1):79–87.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088):533–536.

Salomonsen, S. (2022). Bio-plausible learning in neural networks.

Salvatori, T., Song, Y., Hong, Y., Sha, L., Frieder, S., Xu, Z., Bogacz, R., and Lukasiewicz, T. (2021). Associative memories via predictive coding. Advances in Neural Information Processing Systems, 34:3874–3886.

Scellier, B. and Bengio, Y. (2017). Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. Frontiers in computational neuroscience, 11:24. Publisher: Frontiers Media SA.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green ai. Communications of the ACM, 63(12):54–63.

Sun, W. and Orchard, J. (2020). A predictive-coding network that is both discriminative and generative. Neural computation, 32(10):1836–1862. Publisher: MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ....

Tschantz, A., Millidge, B., Seth, A. K., and Buckley, C. L. (2022). Hybrid predictive coding: Inferring, fast and slow. arXiv preprint arXiv:2204.02169.

Whittington, J. C. and Bogacz, R. (2017). An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. Neural computation, 29(5):1229–1262. Publisher: MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ....

Whittington, J. C. and Bogacz, R. (2019). Theories of error back-propagation in the brain. Trends in cognitive sciences, 23(3):235–250. Publisher: Elsevier.

Appendices

Appendix A

Access to source code

The source code containing the framework, experiments, and results is available at github.com/simensal/BioplausibleNN_MasterThesis_Code. The source code is not meant as a part of the submission for the thesis; all necessary logic to explain the system is included in the pseudocode snippets and equations of the paper. A link to the repository is only included for readers interested in using the system for further research or exploring the raw data collected from the experiments. A README file is included in the repository explaining the structure of it.

Appendix B

Iris - Metadata

```
1. Title: Iris Plants Database
Updated Sept 21 by C.Blake - Added discrepency information

2. Sources:
     (a) Creator: R.A. Fisher
     (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
     (c) Date: July, 1988

3. Past Usage:
   - Publications: too many to mention!!!  Here are a few.
   1. Fisher,R.A. "The use of multiple measurements in taxonomic problems"
      Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions
      to Mathematical Statistics" (John Wiley, NY, 1950).
   2. Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis.
      (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   3. Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
      Structure and Classification Rule for Recognition in Partially Exposed
      Environments".  IEEE Transactions on Pattern Analysis and Machine
      Intelligence, Vol. PAMI-2, No. 1, 67-71.
      -- Results:
         -- very low misclassification rates (0% for the setosa class)
   4. Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE
      Transactions on Information Theory, May 1972, 431-433.
      -- Results:
         -- very low misclassification rates again
   5. See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al's AUTOCLASS II
```

```
      conceptual clustering system finds 3 classes in the data.

4. Relevant Information:
   --- This is perhaps the best known database to be found in the pattern
       recognition literature.  Fisher's paper is a classic in the field
       and is referenced frequently to this day.  (See Duda & Hart, for
       example.)  The data set contains 3 classes of 50 instances each,
       where each class refers to a type of iris plant.  One class is
       linearly separable from the other 2; the latter are NOT linearly
       separable from each other.
   --- Predicted attribute: class of iris plant.
   --- This is an exceedingly simple domain.
   --- This data differs from the data presented in Fishers article
(identified by Steve Chadwick,  spchadwick@espeedaz.net )
The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa"
where the error is in the fourth feature.
The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa"
where the errors are in the second and third features.

5. Number of Instances: 150 (50 in each of three classes)

6. Number of Attributes: 4 numeric, predictive attributes and the class

7. Attribute Information:
   1. sepal length in cm
   2. sepal width in cm
   3. petal length in cm
   4. petal width in cm
   5. class:
      -- Iris Setosa
      -- Iris Versicolour
      -- Iris Virginica

8. Missing Attribute Values: None

Summary Statistics:
         Min  Max   Mean    SD   Class Correlation
   sepal length: 4.3  7.9   5.84  0.83    0.7826
    sepal width: 2.0  4.4   3.05  0.43   -0.4194
   petal length: 1.0  6.9   3.76  1.76    0.9490  (high!)
    petal width: 0.1  2.5   1.20  0.76    0.9565  (high!)
```

9. Class Distribution: 33.3% for each of 3 classes.

Appendix C

Wine - Metadata

```
1. Title of Database: Wine recognition data
Updated Sept 21, 1998 by C.Blake : Added attribute information

2. Sources:
   (a) Forina, M. et al, PARVUS - An Extendible Package for Data
       Exploration, Classification and Correlation. Institute of Pharmaceutical
       and Food Analysis and Technologies, Via Brigata Salerno,
       16147 Genoa, Italy.

   (b) Stefan Aeberhard, email: stefan@coral.cs.jcu.edu.au
   (c) July 1991
3. Past Usage:

   (1)
   S. Aeberhard, D. Coomans and O. de Vel,
   Comparison of Classifiers in High Dimensional Settings,
   Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
   Mathematics and Statistics, James Cook University of North Queensland.
   (Also submitted to Technometrics).

   The data was used with many others for comparing various
   classifiers. The classes are separable, though only RDA
   has achieved 100% correct classification.
   (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
   (All results using the leave-one-out technique)
```

```
   In a classification context, this is a well posed problem
   with "well behaved" class structures. A good data set
   for first testing of a new classifier, but not very
   challenging.

   (2)
   S. Aeberhard, D. Coomans and O. de Vel,
   "THE CLASSIFICATION PERFORMANCE OF RDA"
   Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
   Mathematics and Statistics, James Cook University of North Queensland.
   (Also submitted to Journal of Chemometrics).

   Here, the data was used to illustrate the superior performance of
   the use of a new appreciation function with RDA.

4. Relevant Information:

   -- These data are the results of a chemical analysis of
      wines grown in the same region in Italy but derived from three
      different cultivars.
      The analysis determined the quantities of 13 constituents
      found in each of the three types of wines.

   -- I think that the initial data set had around 30 variables, but
      for some reason I only have the 13 dimensional version.
      I had a list of what the 30 or so variables were, but a.)
      I lost it, and b.), I would not know which 13 variables
      are included in the set.

   -- The attributes are (dontated by Riccardo Leardi,
riclea@anchem.unige.it )
  1) Alcohol
  2) Malic acid
  3) Ash
4) Alcalinity of ash
  5) Magnesium
6) Total phenols
  7) Flavanoids
  8) Nonflavanoid phenols
  9) Proanthocyanins
10)Color intensity
```

```
  11)Hue
  12)OD280/OD315 of diluted wines
  13)Proline

5. Number of Instances

       class 1 59
class 2 71
class 3 48

6. Number of Attributes

13

7. For Each Attribute:

All attributes are continuous

No statistics available, but suggest to standardise
variables for certain uses (e.g. for us with classifiers
which are NOT scale invariant)

NOTE: 1st attribute is class identifier (1-3)

8. Missing Attribute Values:

None

9. Class Distribution: number of instances per class

       class 1 59
class 2 71
class 3 48
```

Appendix D

Inference of Missing Attributes

The following appendix contains the data from the experiments run to research PCNs' abilities to reconstruct missing data in datasets. Section D.1 contains results regarding the standard deviations on the representational datasets, section D.2 contains the remaining samples of reconstructed images of the MNIST dataset, and lastly section D.3 contains the metrics from the post-training evaluation of the agents trained on the reconstructed datasets.

D.1 Standard Deviation

Table D.1 contains the mean value for each attribute of each class in the iris dataset, and table D.2 contains the mean for each attribute of each class in the wine dataset. Subsequently, the mean standard deviation for each attribute of each class for the iris and wine dataset can be found in table D.3 and D.4. The inferred values at each sparsity upon which the values presented in table D.3 and D.4 are available in the thesis repository referenced in appendix A.

Table D.1: Mean values for the attributes for each class of the iris dataset

	Setosa	**Versicolor**	**Virginica**
Sepal Length	4.969	5.977	6.688
Sepal Width	3.372	2.741	3.000
Petal Length	1.438	4.241	5.588
Petal Width	0.236	1.300	2.044

Table D.2: Mean values for the attributes for each class of the wine dataset

	Class 1	**Class 2**	**Class 3**
ABV	13.721	12.284	13.124
Malic acid	2.045	1.933	3.243
Ash	2.462	2.248	2.436
Ash alkalinity	17.173	20.421	21.449
Magnesium	105.4	95.776	99.795
Total Phen.	2.849	2.228	1.676
Flavanoids	2.979	2.082	0.799
Non-flav. Phen.	0.28	0.366	0.439
Proanth.	1.93	1.654	1.159
Color intesity	5.548	3.102	7.329
Hue	1.06	1.062	0.677
OD280/OD315	3.156	2.794	1.67
Proline	1099.156	526.966	627.564

Table D.3: Mean standard deviation for each attribute for each class in the reconstructed iris dataset. The mean was calculated over $\boldsymbol{D}_{PCN}$ and $\boldsymbol{D}_{KNN}$, respectively.

(a) PCN

	Setosa	**Versicolor**	**Virginica**
Sepal Length	2.828	0.227	1.255
Sepal Width	0.893	0.710	0.132
Petal Length	17.811	0.864	3.433
Petal Width	10.220	0.852	3.067

(b) KNN

	Setosa	**Versicolor**	**Virginica**
Sepal Length	0.790	0.897	0.730
Sepal Width	0.282	0.729	0.438
Petal Length	0.389	0.659	0.715
Petal Width	0.442	1.301	0.905

Table D.4: Mean standard deviation for each attribute for each class in the reconstructed wine dataset. The mean was calculated over $\boldsymbol{D}_{PCN}$ and $\boldsymbol{D}_{KNN}$, respectively.

(a) PCN

	Class 1	**Class 2**	**Class 3**
ABV	1.088	1.350	0.488
Malic acid	1.546	1.316	0.105
Ash	0.641	0.286	0.638
Ash alkalinity	1.281	0.171	0.700
Magnesium	0.728	1.081	1.155
Total Phen.	1.154	0.471	2.018
Flavanoids	1.086	0.766	6.249
Non-flav. Phen.	2.003	0.237	0.527
Proanth.	0.193	0.604	1.935
Color intesity	0.987	3.802	0.372
Hue	0.526	0.149	3.548
OD280/OD315	1.468	0.334	3.712
Proline	0.857	2.908	2.892

(b) KNN

	Class 1	**Class 2**	**Class 3**
ABV	0.374	0.574	0.240
Malic acid	0.311	0.284	0.336
Ash	0.498	0.442	0.230
Ash alkalinity	0.420	0.299	0.155
Magnesium	0.368	0.347	0.685
Total Phen.	0.593	0.680	0.313
Flavanoids	0.635	0.476	0.155
Non-flav. Phen.	0.176	0.313	0.472
Proanth.	0.233	0.199	0.356
Color intesity	0.646	1.003	0.473
Hue	0.317	0.379	0.392
OD280/OD315	0.327	0.498	0.371
Proline	0.651	0.440	0.352

D.2 Reconstructed Images from MNIST dataset

Figure D.1-D.7 contain reconstructed images for each output class in the MNIST dataset. The figures show the original, the corrupted, the PCN reconstructed, and the KNN reconstructed samples.

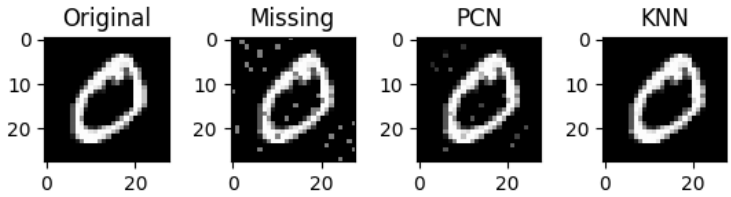


Figure D.1: Reconstructed images of sample from the MNIST dataset depicting a '0'

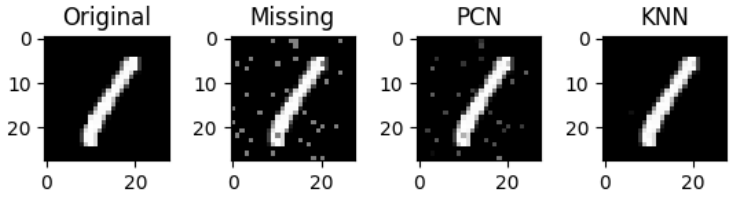


Figure D.2: Reconstructed images of sample from the MNIST dataset depicting a '1'

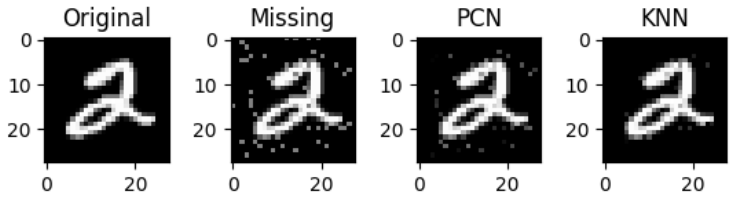


Figure D.3: Reconstructed images of sample from the MNIST dataset depicting a '2'

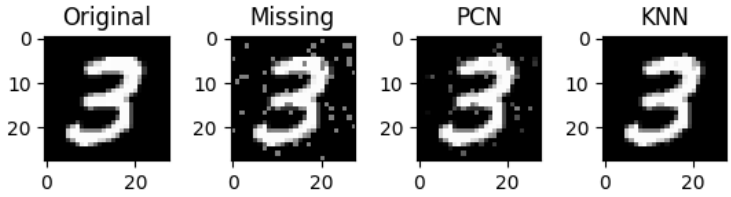


Figure D.4: Reconstructed images of sample from the MNIST dataset depicting a '3'

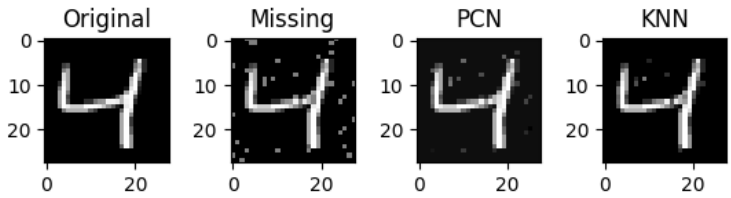


Figure D.5: Reconstructed images of sample from the MNIST dataset depicting a '4'

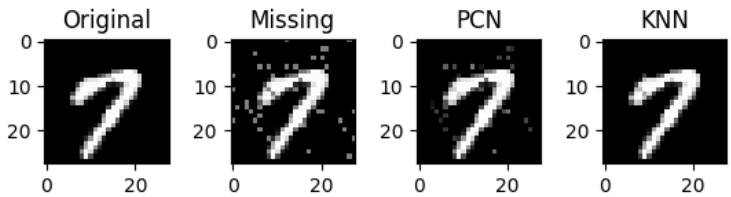


Figure D.6: Reconstructed images of sample from the MNIST dataset depicting a '7'

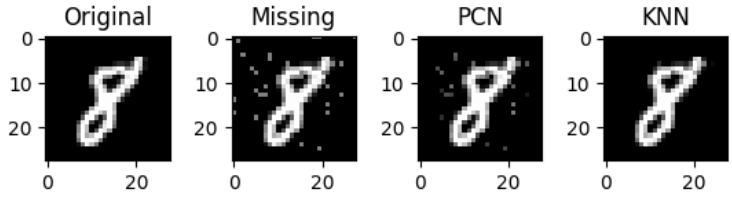


Figure D.7: Reconstructed images of sample from the MNIST dataset depicting an '8'

D.3 Data from Reconstructed-Trained Agents

D.3.1 Mean Accuracy of Agents Trained on $D^{0.05}_{KNN}$ and $D^{0.05}_{PCN}$.

Table D.5 contains the accuracies of the ANN and PCN agents trained on the reconstructed datasets $D^{0.05}_{KNN}$ and $D^{0.05}_{PCN}$. The raw data for the other sparsities can be found in the repository referenced in appendix A

Table D.5: Mean accuracy of agents on test split that had been trained on $D^{0.05}_{KNN}$ and $D^{0.05}_{PCN}$, for each dataset.

	Iris		**Wine**		**MNIST**	
	D_{KNN}	D_{PCN}	D_{KNN}	D_{PCN}	D_{KNN}	D_{PCN}
ANN	0.956	0.949	0.972	0.972	0.940	0.938
PCN	0.757	0.754	0.969	0.969	0.627	0.604

D.3.2 Margins of Agents Trained on $D^{0.05}_{KNN}$

Table D.6 contains the margins for the mean accuracy of the agents trained on $D^{0.05}_{KNN}$ and the control accuracies of subtable 5.2a.

Table D.6: Changes to the accuracies compared to the sparsely trained agents on $D^{0.05}_{KNN}$. A positive number in the table denotes an improvement in the accuracy of the sparsely trained agents.

	Iris		**Wine**		**MNIST**	
	acc_{train}	acc_{test}	acc_{train}	acc_{test}	acc_{train}	acc_{test}
ANN	0.000	-0.020	-0.015	0.000	-0.003	-0.006
PCN	-0.013	-0.025	-0.001	-0.003	0.042	0.053

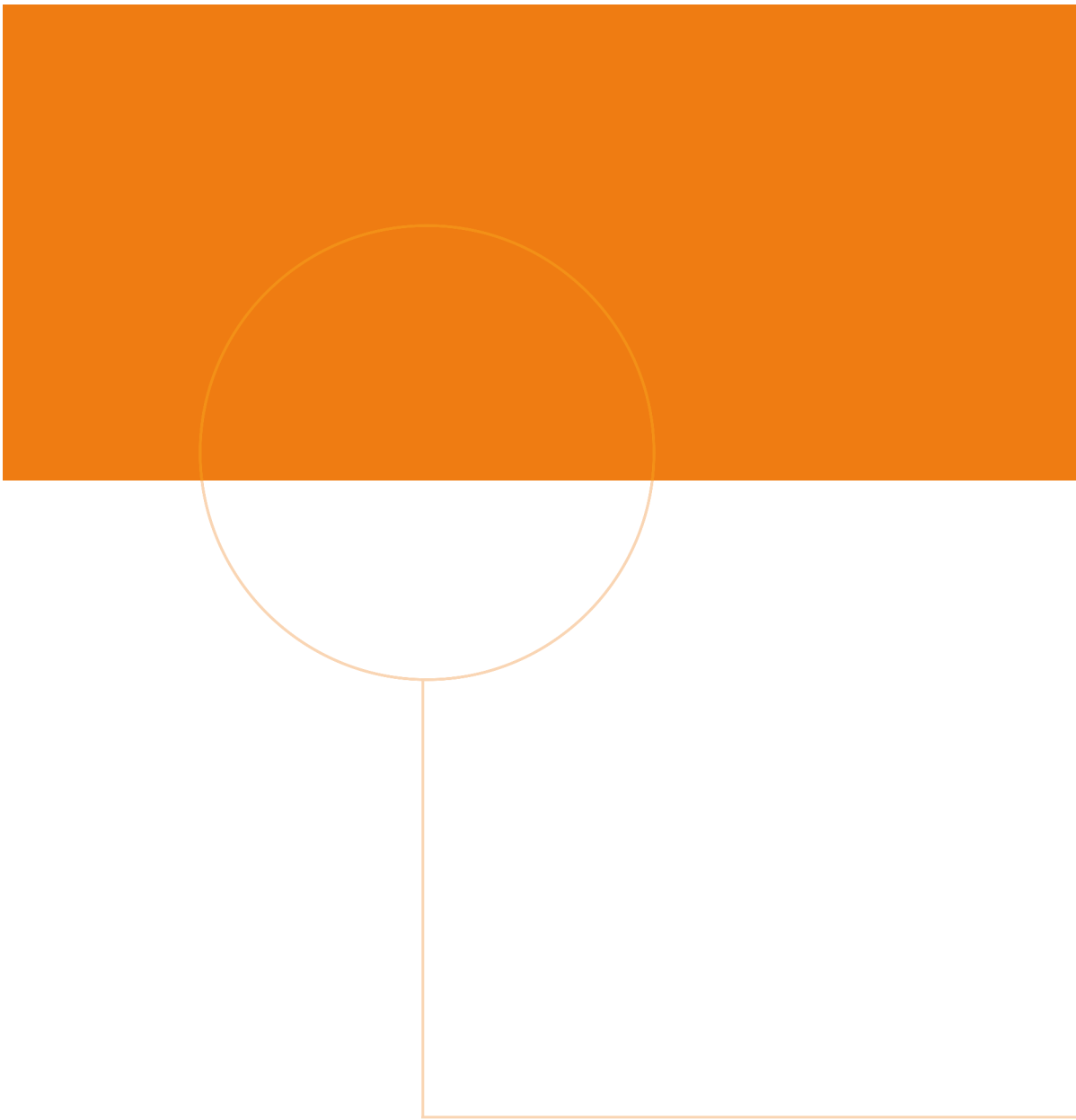
Appendix E

Computational Analysis

The following appendix contains data aggregated from the computational analysis. Table E.1 contains the mean flops performed during training of the equal epoch approach and contains the data figure 5.4 visualizes.

Table E.1: Mean training flops of ANN and PCN agents for equal epoch training experiment for each dataset.

	Iris	**Wine**	**MNIST**
ANN	9.474×10^6	2.591×10^7	9.966×10^{10}
PCN	1.818×10^8	6.567×10^8	1.840×10^{12}



NTNU

Norwegian University of
Science and Technology