Novel Predictive Coding Networks that Achieve Task-Agnostic Learning

University of Oxford

A thesis submitted for the degree of

MSc Computer Science

Trinity 2021

Acknowledgements

I would like to thank my supervisor, Prof. Thomas Lukasiewicz, and his DPhil students, Tommaso Salvatori and Yuhang Song, for providing guidance and feedback throughout this project. At the same time, I also want to thank my parents and my friends for their support and encouragement.

Abstract

This thesis aims to design predictive coding (PC) networks that can perform discriminative and generative tasks simultaneously. Unlike the human brain, standard artificial neural networks with backpropagation are only able to solve one type of problem, such as image classification, generation, or reconstruction. Accordingly, changing the task can be inefficient, since we need to restart training even if the same datasets are used. In this thesis, a bidirectional predictive coding network has been proposed and proven to own the ability of task-agnostic learning. Furthermore, based on the bidirectional model, an assembly model simply composed of a cluster of neurons has been creatively designed, which can not only outperform the former in associative memory tasks while retaining its capability of task-agnostic learning, but also achieve efficiency and flexibility at the level of network architecture.

Contents

List of Figures

1 Introduction

1.1 Background

In 1943, McCulloch and Pitts [3] opened the subject of artificial neural networks (ANNs) and proposed the first computational model inspired by the structure of the animal cerebral cortex. After approximately thirty-year development of deep learning, Werbos's backpropagation algorithm (BP) [15] which was designed for parameter updates, enabled practical training of ANNs and built a solid foundation for future work. Plenty of remarkable success at learning problems was achieved by ANNs with BP, and even until today, BP has been considered the most widely used and influential learning architecture in artificial intelligence and machine learning.

However, ANNs with BP were simultaneously criticized for the lack of biological plausibility. It turns out that computational procedures and principles within BP are dissimilar to the learning process in biological neural networks of the brain (BL) [11]. Specifically, the information BP requires to calculate the loss comes from all previous layers in the network, i.e., the information is locally unavailable, and the model requires external controls to trigger weight updates. Besides the biological implausibility, another limitation of BP is that it forces multilayer networks to learn in one direction along the hierarchy, making the networks task-specific: a network designed for one task usually has trouble solving problems from different types. Over these years, approximations of ANNs with BP that can fill in the two gaps have been the point of discussion.

In terms of biological implausibility, Wittington's predictive coding networks (PCNs) [16] with a novel learning algorithm, *inference learning (IL)* [16] partially overcomes BP's limitation by achieving local plasticity. Later on, Song proposed two powerful PC learning algorithms, *zero-divergence inference learning (Z-IL)* [11], and *Concurrent Predictive Coding (CPC)* [12], and the latter releases the networks from external controls and achieves full autonomy. At the same time, another PC model has been shown to have the capability of performing generative tasks such as associative memory tasks [10]. Though with great success in both aspects, PCNs still show inflexibility in task changing and the incapability of simultaneously functioning different types of tasks.

1.2 Contributions

In summary, the contributions of this thesis are as follows:

- I propose a novel multilayer bidirectional PC model, *B-PCNs*, trained to perform multiple tasks simultaneously. I empirically test this model in different generative and discriminative tasks.
- I then generalize this network to a cluster-like architecture, where every pair of neurons is connected by two different parameters. This new model is called *single assembly networks (SANs)* and can learn datasets regardless of the task it is trained on. Particularly, I have trained it on image classification benchmarks and showed that it could perform image generation, classification, and associative memory tasks. Note that this is not possible to do it using ANNs trained with BP.
- I then conclude by showing that SANs can be seen as a general framework to create different types of PCNs by simply pruning specific parameters. This allows defining models such as standard multilayer networks with residual or recurrent connections or less standard architectures, such as the aforementioned B-PCNs. Particularly, I show that deep networks generated this way and trained with CPC, achieve similar performance compared against standard PCNs trained with IL and MLPs trained with backpropagation.

1.3 Outline

The thesis is organized as follows:

- In **Chapter 2**, I give the preliminaries of this thesis. I start by briefly reviewing the classic ANNs with BP and compare it against PCNs trained with IL, Z-IL, and CPC. Then, I introduce different topics that will be useful in the later chapters of the thesis. Particularly, I will talk about associative memories (AMs) and task-agnostic learning.
- In **Chapter 3**, I propose a novel task-agnostic PC network, called bidirectional PC networks (B-PCNs), displaying the model implementation and related algorithms.

- In **Chapter 4**, I generalize the idea behind B-PCNs and propose a novel, fully-connected architecture, called the single assembly networks (SANs), and develop a method to create different kinds of neural networks by simply pruning specific weight parameters of SANs.
- In **Chapter 5**, I apply B-PCNs and SANs to classification, generation, and associative memory tasks, compare their performance, and experimentally show their capability of task-agnostic learning.
- In **Chapter 6**, I give a discussion about the results of experiments and an outlook for future study.

2 Preliminaries

This section will first briefly review the standard artificial neural network and back-propagation, and analyze their underlying problems. Then, an introduction about *predictive coding networks (PCNs)* will be given. Next, three influential algorithms behind PCNs will be illustrated: *inference learning (IL)*, which is considered a close approximation of BP, *zero-divergence inference learning (Z-IL)*, which is a modification of IL that exactly replicates the weight update of BP, and *Concurrent Predictive Coding (CPC)*, which outperforms the first two in terms of efficiency and biological plausibility while providing equivalent prediction accuracy. Lastly, limitations of current PCNs will be presented, and I will further explain the motivation of the thesis by introducing the concept of associative memories and task-agnostic learning.

2.1 ANNs with BP

ANNs are composed of artificial neurons in the multilayer structure. Each artificial neuron receives inputs from neurons in the adjacent previous layer and produces a single output sent to multiple neurons in the following layer. Therefore, the learning process of the neural network can be formulated as an optimization problem: we are asked to seek the parameter θ to minimize the loss L of the training data set of inputs x and labels y, as below:

$$min\mathbf{E}(L(y, P(x; \theta))), \tag{1}$$

where P denotes the predicted labels generated by the neural network with parameters θ, including weights and biases. The difference between predicted output and the true label is quantified by the loss function, and then the expected value is computed among the training data set.

BP was proposed to solve the optimization problem above [8]. During each learning epoch, stochastic gradient descent will be used in conjunction to minimize the expected loss. Weight update, with respect to the parameters θ, will be computed based on the gradient descent update rule,

$$\theta \leftarrow \theta - \alpha \Delta_\theta L(P(x; \theta), y), \tag{2}$$

where α is a positive constant called the learning rate.

BP is considered one of the most effective learning algorithms in deep artificial neural networks and can be generalized to a wide range of tasks with outstanding performance. Nevertheless, ANNs with BP has been criticized for its lack of biological plausibility, i.e., the way it processes the information is dissimilar to cortical processing, because of two significant reasons:

Lack of local plasticity: As inferred in Eqs. 1 and 2, parameter updates performed by gradient descent follows the calculation of loss function, which only happen in the output layer. Hence, information for weight updates is locally unavailable. This is in contrast with biological networks present in the brain, which only rely on local information to update the strength of the synapses.

Lack of full autonomy: External triggers are required as the weight updates happen at the end of each epoch. On the other hand, the human brain learns with full autonomy, and no external control is needed.

As a result, many efforts have been devoted to the construction of novel models and algorithms which can mimic the process of human brain more precisely, and overcome the limitation of BP while, at the same time, approximating its excellent performance.

2.2 PCNs

In 1992, Mumford proposed a hypothesis that predictive coding (PC) is a processing strategy taking place in cortical networks [4]. The prediction was sent down from the top layer of the hierarchical network. With PC, each layer only uses locally available information and communicates with adjacent layers. Each neuron makes an inference about what state the neurons in the next lower layer should be in and compares its current state with the inference received from a higher layer. In 1999, Rao applied this idea and designed *predictive coding networks (PCNs)*, a hierarchical multilayer model of information processing. Rao used PCNs to conduct unsupervised learning and model the visual processing in the cerebral cortex. Specifically, one side of the model, called the *input layer*, is clamped to the input images, and neurons in all other layers are free to converge. The inference is sent from the output layer to the input layer.

In 2017, Wittington and Bogacz [16] applied Rao's model to solve classification problems after creative modification. Unlike Rao, Wittington clamped both sides of the

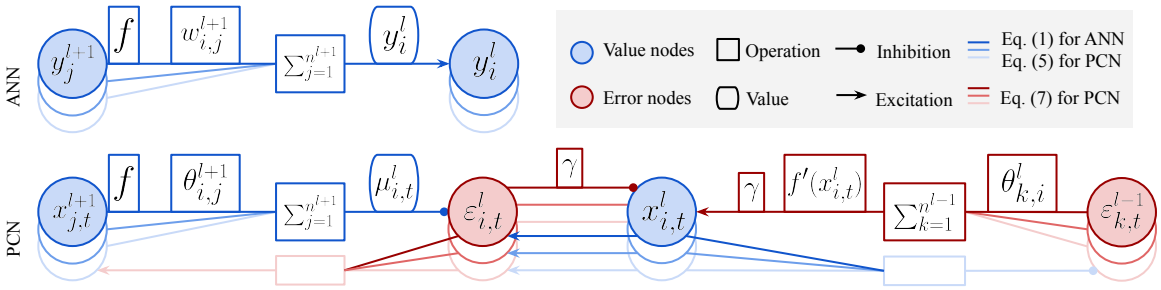


Figure 1: Cited from [11]. Items in the figure: the structure of ANN with BP (top); the structure of PCNs with IL proposed by Wittington. Inference values (blue nodes) are sent from a lower layer $(l+1)$ to a higher layer l, $\varepsilon^l_{i,t}$, the error between the values in the current nodes $x^l_{i,t}$ and the prediction $\mu^l_{i,t}$ is used to update the inferred values in a higher layer $(l-1)$.

hierarchical network, and inference is sent in the opposite direction, i.e., from the input layer to the output one. Simultaneously, they formally introduced the concept of *inference learning (IL)*, the fundamental learning algorithm behind PCNs that enables models to infer values of neurons in hidden layers. We now introduce the architecture of PCNs and the mechanism of IL.

Following [11], this thesis denotes the layer index by l. $l = 0$ denotes the output layer and $l = l_{max}$ denotes the input layer. Also, we use *forward* or *top-down* to describe a pass from layer l_{max} to layer 0, i.e., the direction in which Wittington's model sends the inference, and *backward* or *bottom-up* for the opposite, i.e., the direction of running inference in Rao's model. Furthermore, we use $\bar{x} = (x_1, x_2, \ldots, x_n)$ to denote n-dimensional vectors.

IL The learning process of PCNs trained with IL can be described in two stages: (1) the learning stage: true labels are provided, and the goal is to update parameters to learn the connections within datasets, and (2) the prediction stage: no label provided and the model is expected to generate correct labels based on the current parameters. We will denote by t the time axis during inference, and by $(\bar{s}^{in}, \bar{s}^{out})$ labelled data points. Each neuron has a value node (the blue nodes in Fig. 4), which is a learnable parameter denoted by $x^l_{i,t}$. Here, i represents the i-th neuron of the l-th layer. Each value node is associated with an error node $\varepsilon^l_{i,t}$ representing the error of the prediction. The major difference from BP is that IL enables PCNs to manipulate the value nodes of hidden layers via inference learning. Particularly, the value nodes from the layer $(l+1)$ are used as an input to make predictions in the layer l. Generally, it follows

the rules:

$$\mu_{i,t}^{l} = \sum_{j=1}^{n^{l+1}} \theta_{i,j}^{l+1} f(x_{j,t}^{l+1}) \quad and \quad \varepsilon_{i,t}^{l} = x_{i,t}^{l} - \mu_{i,t}^{l}, \tag{3}$$

where $\mu_{i,t}^{l}$ denotes the predicted value of $x_{i,t}^{l}$ at time t,and f is a non-linear activation function. The overall energy function at a time step t is defined as the sum of the errors:

$$F_t = \frac{1}{2} \sum_{l=0}^{l_{max}} \sum_{i=1}^{n^l} (\varepsilon_{i,t}^{l})^2, \tag{4}$$

where n^l denotes the number of neurons in layer l. The goal of the learning algorithm is to minimize the energy function F_t. While the network is running inference, each value node $x_{j,t}^{l}$ moves towards $\mu_{i,t}^{l}$ following the rule:

$$x_{i,t+1}^{l} = x_{i,t}^{l} + \Delta x_{i,t}^{l}, \tag{5}$$

where $\Delta x_{i,t}^{l}$ is defined as:

$$\Delta x_{i,t}^{l} = \begin{cases} 0 & if\, l = l_{max} \\ \gamma \cdot (-\varepsilon_{i,t}^{l} + f'(x_{i,t}^{l}) \sum_{k=1}^{n^{l-1}} \varepsilon_{k,t}^{l-1}\, \theta_{k,i}^{l} & if\, l \in \{1, ..., l_{max} - 1\} \\ 0 & if\, l = 0 \quad during\, learning \\ \gamma \cdot (-\varepsilon_{i,t}^{l}), & if\, l = 0 \quad during\, prediction. \end{cases} \tag{6}$$

γ denotes the integration step for the value nodes. During the *learning phase*, the value nodes in the input layer $l = l_{max}$ are fixed to $\bar{s}^{in}$, and those in the output layer $l = 0$ are fixed to $\bar{s}^{out}$, i.e.,

$$\varepsilon_{i,t}^{0} = x_{i,t}^{0} - \mu_{i,t}^{0} = \bar{s}_{i}^{out} - \mu_{i,t}^{0}. \tag{7}$$

The rest neurons are free to converge. The training pair will be presented to the model for the duration of T. At $t = t_c$, where t_c is fixed number and $t_c \leq T$, the network will reach an equilibrium and the total energy of the model will converge to a constant. Then, all weight parameters will be updated according to:

$$\theta_{i,j}^{l+1} = \theta_{i,j}^{l+1} + \Delta\theta_{i,j}^{l+1}, \tag{8}$$

where $\Delta\theta_{i,j}^{l+1}$ is defined as:

$$\Delta\theta_{i,j}^{l+1} = -\alpha \cdot \partial F_t / \partial \theta_{i,j}^{l+1} = \alpha \cdot \varepsilon_{i,t}^{l} f(x_{j,t}^{l+1}). \tag{9}$$

During the *prediction phase*, only value nodes in the input layers will be fixed to $\bar{s}^{in}$, and the network will run inference again based on the current weight parameter $\theta^l_{i,t}$. The error will be optimized to zero as $t \to \infty$, then $\mu^0_{i,0}$ will be the predicted labels for each data point.

It can be noticed from Eqs. 6 and 9 that updates of weight and value nodes rely solely on local information in PCNs, unlike in ANNs with BP, which calculates the error in the output layer and updates the weight parameters in all the previous layers accordingly. From this, we can conclude that PCNs with IL have the property of local plasticity. The learning algorithm is displayed in Alg. 1.

Algorithm 1 Learning $(\bar{s}^{in}, \bar{s}^{out})$ in duration T, PCNs with IL

Require: $\bar{x}^{l_{max}}_0$ is fixed to $\bar{s}^{in}$, $\bar{x}^0_0$ is fixed to $\bar{s}^{out}$
  for $t = 0$ to T **do**
    for each neuron i in each level l **do**
      Update $x^l_{i,t}$ to minimize F_t via Eq. (6)
      if $t = t_c$ **then**
        Update each $\theta^{l+1}_{i,j}$ to minimize F_t via Eq. (9)
      end if
    end for
  end for

Nevertheless, it can be realized that IL only updates weight parameters once at $t = t_c$, and such restriction indicates that (1) this model is far less efficient than BP, and (2) that external controls are still required, and therefore PCNs with IL fail to achieve full autonomy as desired. Hence, further improvements are needed.

Z-IL Proposed in Song's article [11], *IL with zero divergence from BP (Z-IL)* solves the first problem to some degree. It differs from IL in two main aspects. First, Z-IL sets $x^l_{j,t} = \mu^l_{i,t}$ for every neuron before starts learning, i.e., set $\varepsilon^l_{j,t} = 0$. By doing this, F_t can decay to zero in the learning phase, and the first forward pass can be considered identical to the learning pass in feed-forward ANNs with BP. Since BP runs much faster in general than inference, Z-IL significantly improves the efficiency of the original algorithm. Second, rather than performing all weight updates at $t = t_c$, Z-IL requires the network to update weights $\theta^{l_k}_{j,t}$ in the layer l_k at $t = l_k$. The equations used to update value nodes and weights are identical to Eqs. 6 and 9. The learning algorithm is shown in Alg. 2.

Algorithm 2 Learning $(\bar{s}^{in}, \bar{s}^{out})$ in duration T, PCNs with Z-IL

Require: $\bar{x}_0^{l_{max}}$ is fixed to $\bar{s}^{in}$, $\bar{x}_0^0$ is fixed to $\bar{s}^{out}$
Require: $x_{i,0}^l = \mu_{i,0}^l$ for $l \in 1, ..., l_{max} - 1$, and $\gamma = 1$.
  for $t = 0$ to T **do**
    for each neuron i in each level l **do**
      Update $x_{i,t}^l$ to minimize F_t via Eq. (6)
      if $t = l$ **then**
        Update each $\theta_{i,j}^{l+1}$ to minimize F_t via Eq. (9)
      end if
    end for
  end for

As formally proven in the original paper [11], if we set the integration step γ to 1, Z-IL is able to achieve the exact same parameter update as BP, and updating $\theta_{i,j}^{l_k}$ at time l_k improves the autonomy to some degree. Nevertheless, this network again fails to achieve full autonomy since we need to check $t = l$ for each l. We now introduce another algorithm proposed based on Z-IL that successfully frees the conditions.

2.3 CPC

Concurrent Predictive Coding (CPC), proposed by Song [12], owns two competitive properties: full autonomy and high efficiency.

CPC has the identical energy function with IL, namely defined by Eq. 4. Simultaneously, they use identical formulas to update $x_{i,t}^l$ and $\theta_{i,j}^l$, shown in Eqs. 6 and 9. The only difference between the two algorithms is that, rather than update all $\theta_{i,j}^l$ once when the inference converges $(t = t_c)$ or update $\theta_{i,j}^{l_k}$ for each layer at $t = l_k$, CPC constantly updates all $\theta_{i,j}^l$ while updating $x_{i,t}^l$. Clearly, no extra control signal is needed.

In CPC, We will set $\mu_{i,0}^l$ to $x_{i,0}^l$ as in Z-IL. After F_t decays to zero or $t = T$, the learning phase terminates and the model applies the current weight parameters and runs inference to make a prediction. The learning algorithm is shown in Alg. 3, and Fig. 2 directly visualizes the difference between IL, Z-IL and CPC.

As the first learning algorithm for deep neural networks which allows neurons to work fully autonomously, CPC overcomes the biological implausibility of BP. Further, it is proven to be faster than BP and IL while achieving comparable performance.

Algorithm 3 Learning $(\bar{s}^{in}, \bar{s}^{out})$ in duration T, CPC

Require: $\bar{x}_0^{l_{max}}$ is fixed to $\bar{s}^{in}$, $\bar{x}_0^0$ is fixed to $\bar{s}^{out}$
  for $t = 0$ to T **do**
    for each neuron i in each level l **do**
      Update $x_{i,t}^l$ to minimize F_t via Eq. (6)
      Update each $\theta_{i,j}^{l+1}$ to minimize F_t via Eq. (9)
    end for
  end for

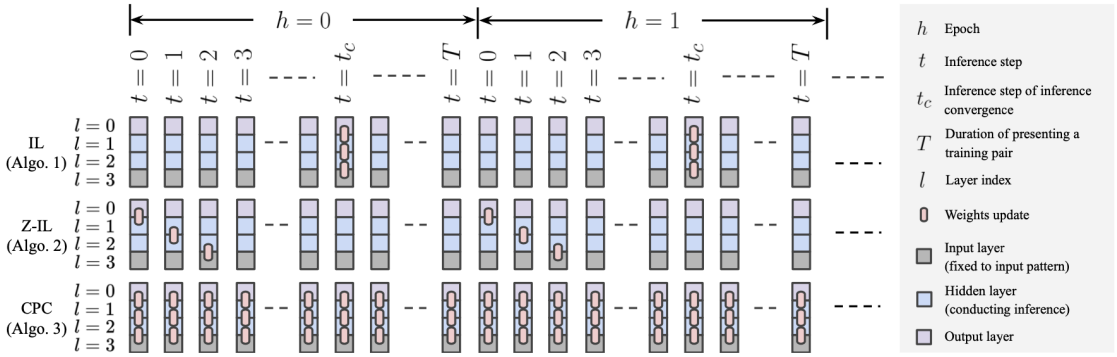


Figure 2: Left: visualization of the process of weight updates in IL, Z-IL and CPC, cited from [12]. Right: important notations which help understand this figure.
We can see that with IL the network updates all weight parameters at $t = t_c$, and with Z-IL it updates the weights for layer l_k at $t = l_k$. A network with CPC constantly updates weights in all layers along with the training.

2.4 Associative memory

Associative memory (AM) is a psychological concept defined as the ability to memorize and learn the relationship between unrelated objects. In machine learning, associative memories are pattern storage and retrieval systems composed of two different forms: auto-associative memories and hetero-associative memories. While both indicate the ability to recall patterns of inputs, hetero-associative memories represent the ability to make associations between concepts from different categories. For example, in Fig. 3(a), the network will recall the concept of "fish" when provided with the concept of "cat". On the contrary, auto-associative memories represent the ability to recall patterns of data points while given the corrupted version. For example, the network will recall the original cat image when provided with the top half of it or a blurry version of it (Fig. 3(b)).

Many efforts have been devoted to the development of AMs in recent years, especially

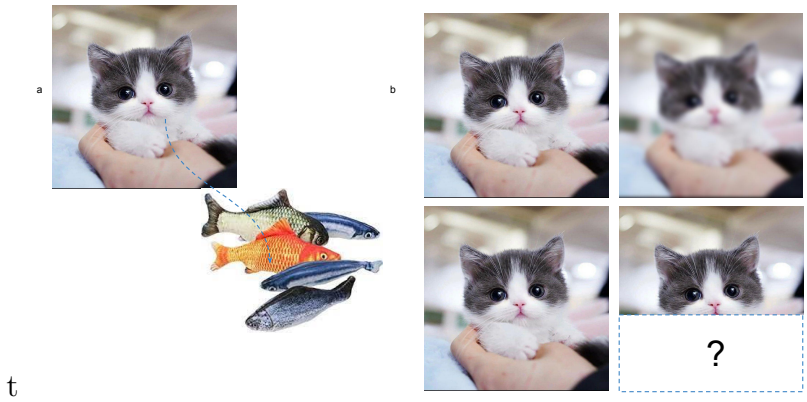


Figure 3: Visualization of hetero-associative memories (a) and auto-associative memories (b).

in the area of auto-associative memories. As one of the most classic and challenging tasks in auto-associative memories, image reconstruction expects the model to recover the data based on the corrupted images. Fig. 3(b) displays two significant types of reconstruction are (1) retrieval from corrupted data points with noise (top) and (2) retrieval from partial data points (bottom).

Associative memories were first proposed by Steinbuch [13], and in 1982, Hopfield designed a discrete energy-based model that memorizes and retrieves data points by minimizing a suitable energy function [2]. It generally consists of one single layer of neurons related to the input and output size, which must be identical. The model stores the binary patterns of images and takes partial data as attractors to do image reconstruction. Other types of models are also found showing excellent performance these years. For example, standard over-parameterized neural networks trained with standard optimization methods can implement AMs [6]. In contrast to the Hopfield model, the memorization and retrieval mechanisms in over-parameterized networks are automatic after the training phase and do not require the construction and minimization of an energy function.

It turns out that PC is naturally related to AMs. As discussed previously, Rao proposed the first PC model for generative tasks [7]. While Wittington was working on its application in classification, some researchers followed Rao's work and trained PC networks as autoencoders rather than classifiers, even though most of the time, these networks can only reconstruct samples from a specific class [14]. In 2021, Salvatori implemented AMs with Rao's bottom-up PCNs, which achieved outstanding performance in reconstruction [10].

2.5 Task-agnostic learning

Before presenting the models proposed in this thesis, we shall first introduce the concept of *task-agnostic learning*. By saying it, we indicate that a model can simultaneously perform generative and discriminative tasks without re-training. Task-agnostic learning shows substantial similarity with cortical processing, as in daily life, we usually do not need to be specifically trained for a particular task before performing it. For example, we can use the generic knowledge to distinguish a cat from other animals, draw a cat, or draw a cat's body when shown with its head. On the contrary, most ANNs are trained for a specific task, and performing different types of tasks is usually implausible at the level of architecture. In other words, when we want to switch between tasks even over the same datasets, we need to restart training with a different model, which can be inefficient and inflexible.

2.6 Short summary

Based on the previous discussion, the primary goal of this thesis is to propose a model which can overcome ANNs and PCNs' limitations of being task-specific and can switch flexibly among different test modes after the learning phase.

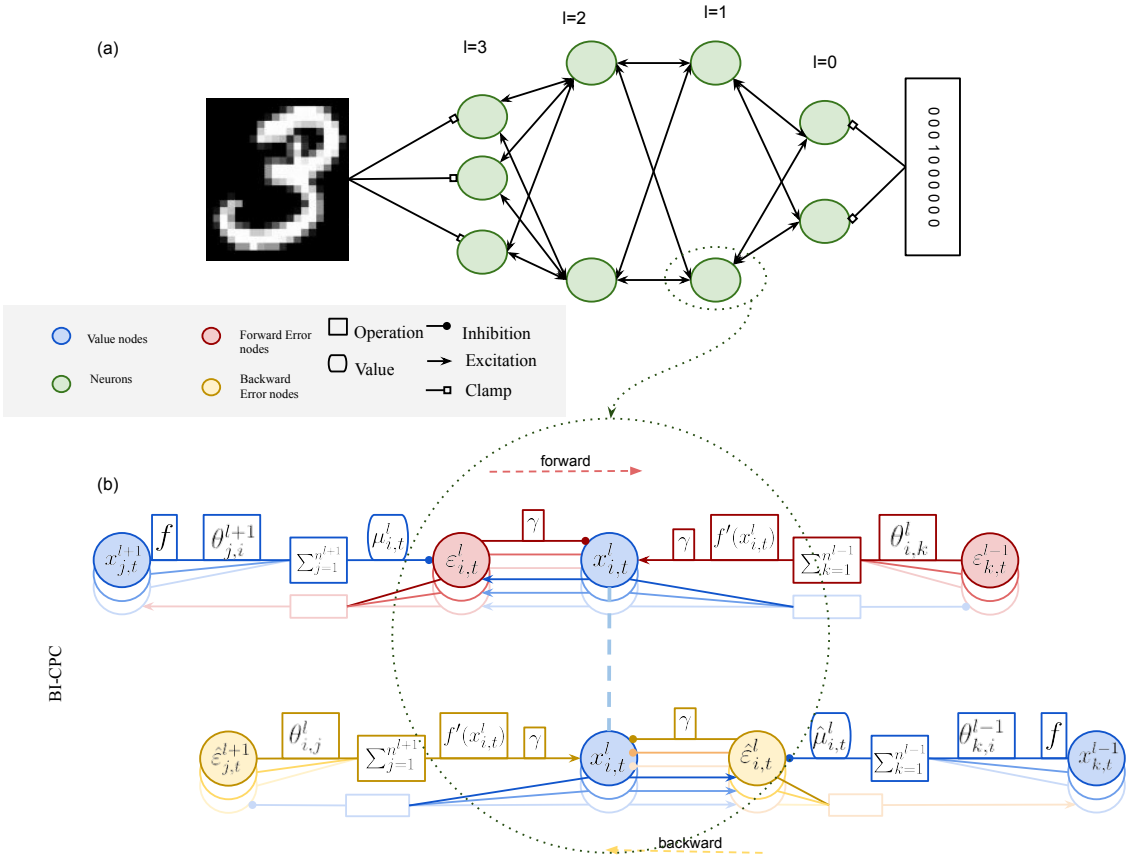


Figure 4: This figure visualizes B-PCNs. (a) shows an overview of the network structure. Every two neurons is connected in the forward and backward directions. The input, digit 3, is fixed to neurons in layer l_{max}(3 in this case), and neurons in the output layer (l = 0) relate to the 1-hot label. (b) shows a concrete structure of a neuron, including a value nodes and two error nodes and presents how it relates to neurons in adjacent layers.

3 Bidirectional PCNs

As stated in the previous chapter, different PCNs are proven to be able to conduct classification or AM tasks, and the significant difference between the structure of Wittington's model (the discriminative model) and Rao's model (the generative model) is the direction of inference sending. Therefore, intuitively, a network running inference in both directions might be equipped with the ability of both models simultaneously and achieve task-agnostic learning. I hence propose a novel model, **bidirectional predictive coding networks (B-PCNs)** based on this hypothesis.

In general, a backward pass is now added to Wittington's PCNs: for the i-th neurons

in layer l, except sending its value to neurons in layer $(l-1)$, it also sends the value to neurons in layer $(l+1)$. Therefore, B-PCNs run inference processes in opposite directions: the first one is sent down the hierarchy, and the other is sent up. Fig. 4 displayed the structure with a sample network consisting of 2 hidden layers, each including 2 neurons.

It might still be unclear why we do not modify ANNs and run BP bidirectionally to achieve task-agnostic learning. The major reason is that cycles are introduced after a backward pass is added. If a cycle is presented inside the neural structure of ANNs, an infinite loop would be created during the first forward pass, and hence would make learning not feasible. On the contrary, since PCNs solely rely on locally available information for parameter updates, inverting the direction of learning and incorporating cycles can be achieved.

3.1 Network architecture

Similar to feed-forward PCNs, each neuron in the bidirectional architecture contains a value node (the blue ones, with the activities of $x_{i,t}^l$), yet simultaneously associated with two corresponding prediction-error nodes (red ones for the forward error, and yellow ones for the backward error, with the activity $\varepsilon_{i,t}^l$ and $\hat{\varepsilon}_{i,t}^l$). Assume we have neuron i in layer l and neuron j in layer $(l-1)$. The edge between i and j now has two weight parameters: $\theta_{i,j}^l$, which denotes the weight from i to j, and $\theta_{j,i}^{l-1}$, which denotes the weight from j to i. Specifically, the i-th neuron in layer j sends $\theta_{i,j}^l f(x_{i,t}^l)$ to j and neuron j sends $\theta_{j,i}^{l-1} f(x_{j,t}^{l-1})$ back to i, where f denotes activation function. Prediction and error are calculated based on the formula:

$$\mu_{i,t}^l = \sum_{j=1}^{n^{l+1}} \theta_{j,i}^{l+1} f(x_{j,t}^{l+1}) \quad and \quad \varepsilon_{i,t}^l = x_{i,t}^l - \mu_{i,t}^l \quad (forward) \tag{10}$$

$$\hat{\mu}_{i,t}^l = \sum_{q=1}^{n^{l-1}} \theta_{q,i}^{l-1} f(x_{q,t}^{l-1}) \quad and \quad \hat{\varepsilon}_{i,t}^l = x_{i,t}^l - \hat{\mu}_{i,t}^l \quad (backward). \tag{11}$$

Eq. 10 is identical to Eq. 3. In Eq. 11, $\hat{\mu}_{i,t}^l$ denotes the backward predicted value of neuron i in layer l at time t, calculating by applying the activation function f to each value node $x_{q,t}^{l-1}$ in a lower layer, multiplying it by the backward weight $\theta_{q,i}^{l-1}$ and doing summation over all the neurons in layer $(l-1)$. $\hat{\varepsilon}_{i,t}^l$ quantifies the difference

between the actual value node and backward prediction, and the goal is to minimize the overall energy F_t, defined as follows:

$$F_t = F_t^{forward} + F_t^{backward} = \sum_{l=0}^{l_{max}} \sum_{i=1}^{n^l} \frac{1}{2}((\varepsilon_{i,t}^l)^2 + (\hat{\varepsilon}_{i,t}^l)^2). \tag{12}$$

Clearly, $x_{i,t}^l$ has to move close to both $\mu_{i,t}^l$ and $\hat{\mu}_{i,t}^l$ in the training phase, and eventually find a balance between the red nodes and yellow nodes. $x_{i,t}^l$ will be updated according to Eq. 5, where $\Delta x_{i,t}^l$ is defined as:

$$\Delta x_{i,t}^l = \begin{cases} 0 & if\, l = l_{max} \quad during\, prediction \\ \gamma \cdot (-\varepsilon_{i,t}^l + f'(x_{i,t}^l) \sum_{q=1}^{n^{l-1}} \varepsilon_{q,t}^{l-1}\, \theta_{q,i}^l), & if\, l = l_{max} \quad during\, learning \\ \gamma \cdot (-\varepsilon_{i,t}^l + f'(x_{i,t}^l) \sum_{q=1}^{n^{l-1}} \varepsilon_{q,t}^{l-1}\, \theta_{q,i}^l & \\ \quad + f'(x_{i,t}^l) \sum_{k=1}^{n^{l+1}} \hat{\varepsilon}_{k,t}^{l+1}\, \theta_{k,i}^l), & if\, l \in \{1, ..., l_{max} - 1\}. \\ 0 & if\, l = 0 \quad during\, prediction \\ \gamma \cdot (-\varepsilon_{i,t}^l + f'(x_{i,t}^l) \sum_{k=1}^{n^{l+1}} \hat{\varepsilon}_{k,t}^{l+1}\, \theta_{k,i}^l), & if\, l = 0 \quad during\, learning \end{cases} \tag{13}$$

γ is the integration step for $x_{i,t}^l$. Inference happens in both the learning and prediction phase as in the feed-forward PCNs.

Learning: Given the training pair of data $(\overline{s}^{in}, \overline{s}^{out})$, the value nodes of the input layer are fixed to $\overline{s}^{in}$, and the value nodes of the output layers are fixed to $\overline{s}^{out}$. Hence,

$$\varepsilon_{i,t}^0 = x_{i,t}^0 - \mu_{i,t}^0 = \overline{s}^{out} - \mu_{i,t}^0 \tag{14}$$

$$\hat{\varepsilon}_{i,t}^{l_{max}} = x_{i,t}^{l_{max}} - \hat{\mu}_{i,t}^{l_{max}} = \overline{s}^{in} - \hat{\mu}_{i,t}^{l_{max}}. \tag{15}$$

$x_{i,t}^l$ and $\theta_{i,t}^l$ are updated constantly and simultaneously in both direction to minimize F_t. Specifically, the rule to update $\theta_{i,t}^l$ is as follows:

$$\Delta\theta_{i,j}^l = -\alpha \cdot \partial F_t / \partial \theta_{i,j}^l = \alpha \cdot \varepsilon_{i,t}^l f(x_{j,t}^{l+1}) + \alpha \cdot \hat{\varepsilon}_{i,t}^l f(x_{j,t}^{l-1}), \tag{16}$$

where α is the learning rate. $\mu_{i,t}^l$ is set to $x_{i,t}^l$ at the beginning, so $\varepsilon_{k,t}^l$ and $\hat{\varepsilon}_{k,t}^l$ will decay to 0 at $t = t_c$ where $t_c \leq T$. The learning algorithm is shown in Alg. 4.

Test: B-PCNs are capable of performing three type of tasks: **classification**, **generation** and **AM tasks**. In general, the only difference among three modes is the way to fix the value nodes in layer l_{max} and layer 0.

Algorithm 4 Learning $(\bar{s}^{in}, \bar{s}^{out})$ in duration T, Bidirectional PCNs

Require: $\bar{x}_0^{l_{max}}$ is fixed to $\bar{s}^{in}$, $\bar{x}_0^0$ is fixed to $\bar{s}^{out}$
  for $t = 0$ to T **do**
    for each neuron i in each level l **do**
      Update $x_{i,t}^l$ to minimize Eq. (12) via Eq. (13)
      Update each $\theta_{i,j}^{l+1}$ to minimize Eq. (12) via Eq. (16)
    end for
  end for

Mode	$x_{i,t}^{l_{max}}$	$x_{i,t}^{l_0}$	Results
Classification	all to $\bar{s}_{test}^{in}$	free to converge	$\mu_{i,t}^0$
Generation	free to converge	all to $\bar{s}^{out}$	$\mu_{i,t}^{l_{max}}$
AM tasks	fix $\bar{x'}$ to $\bar{s}^{in'}$, the rest free to converge	all to $\bar{s}^{out}$	$\mu_{k,t}^{l_{max}}$, where $x_{k,t}^{l_{max}} \in (\bigcup(x_{i,t}^{l_{max}})$ - $\bar{x'})$

Table 1: The table displays the difference among three test modes by showing how to fix the value nodes (column 1 and 2) and what we should consider as results (column 3) during each test mode.

The answers to the first two can be intuitive. As shown in Tab. 1, in **classification**, the value nodes in the input layer, $x_{i,t}^{l_{max}}$, are fixed to images from the test set, $\bar{s}_{test}^{in}$; in **generation**, value nodes in the output layer, $x_{i,t}^0$, are fixed to $\bar{s}^{out}$, the labels of the images which we want to generate. All other neurons are free to converge.

It can be a little complicated for **AM tasks**, and we now introduce the parameters in details. The mechanism is presented vividly in Fig. 5. Specifically, this thesis only focuses on the recovery of cropped images, so for any input $\bar{s}^{in}$, we partially crop it to $\bar{s}^{in'}$ such that

$$\bar{s}^{in'} \subsetneqq \bar{s}^{in}. \tag{17}$$

For example, in Fig. 5, $\bar{s}^{in}$ denotes the the full cat image provided in learning, and $\bar{s}^{in'}$ denotes the the half cropped cat image provided before the network performs AM tasks.

Assume neurons fixed to $\bar{s}^{in'}$ during the training phase is $\bar{x'}$. In the AM tasks, we will fix $\bar{x'}$ to $\bar{s}^{in'}$ again, whereas $(\bigcup(x_{i,t}^{l_{max}})$ - $\bar{x'})$ will be free to converge. In the cat example, $\bar{x'}$ represents values of the top two neurons clamped to the partial cat

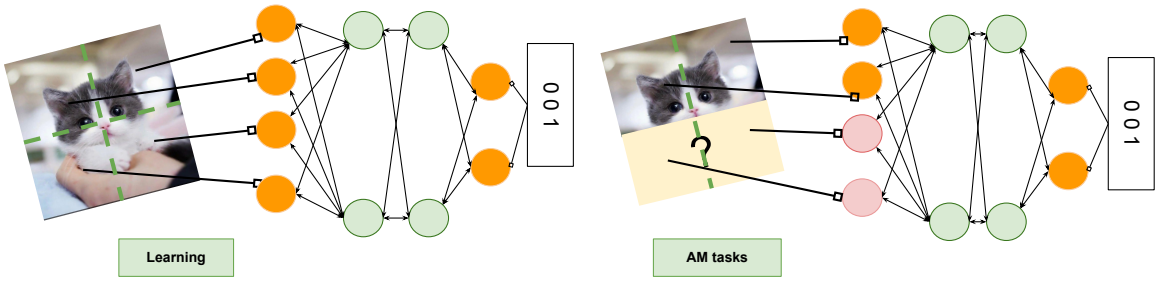


Figure 5: This figure visualizes how B-PCNs learn (left) and reconstruct (right) image of a cat. Orange value nodes are fixed, and green and red nodes are free to converge. After F_t decays to zero, the predicted values of the red nodes are the reconstructed portion.

image, and ($\bigcup(x_{i,t}^{l_{max}})$ - $\bar{x'}$) apparently represents the values of other two neurons in the leftmost layer. Simultaneously, value nodes in layer 0, $x_{i,t}^0$, will be fixed to labels of the images we want to reconstruct, $\overline{s}^{out}$. All value nodes in the hidden layers are free to converge.

In each mode, the network will use the current weight parameters to run inference forward and backward, updating the unfixed value nodes. When $t \to \infty$, $\varepsilon_{i,t}^l$ and $\hat{\varepsilon}_{i,t}^l$ both decay to 0.

Tab. 1 summarizes the results for all three modes. In classification, the predicted values of neurons in layer 0, $\mu_{i,t}^0$, will be the labels constructed by the network; in generation, the predicted values of the neurons in layer l_{max}, $\mu_{i,t}^{l_{max}}$, will be the images constructed by the network; for reconstruction, the predicted values of unfixed nodes in layer l_{max} will be the reconstructed portion.

3.2 Further exploration

We showed that adding a backward pass to Wittington's PCNs and making it bidirectional achieve task-agnostic learning theoretically. We shall then intuitively ask another question: will the model performs even better if we go further, adding passes across the hierarchy or entirely breaking the multilayer structure? To be specific, now neurons in layer l can only connect to the neurons in the adjacent layers, $(l+1)$ and $(l-1)$. What if we allow it to send the inferred values to and receive error signal from neurons in layer $(l-2)$ or $(l+2)$? Further, what if we allow each neuron to freely and bidirectionally connect to any neuron in the model except itself? To find answers

to these questions, we break the hierarchical structure of B-PCNs and generalize it to another network including only a cluster of neurons without layers - the single assembly network.

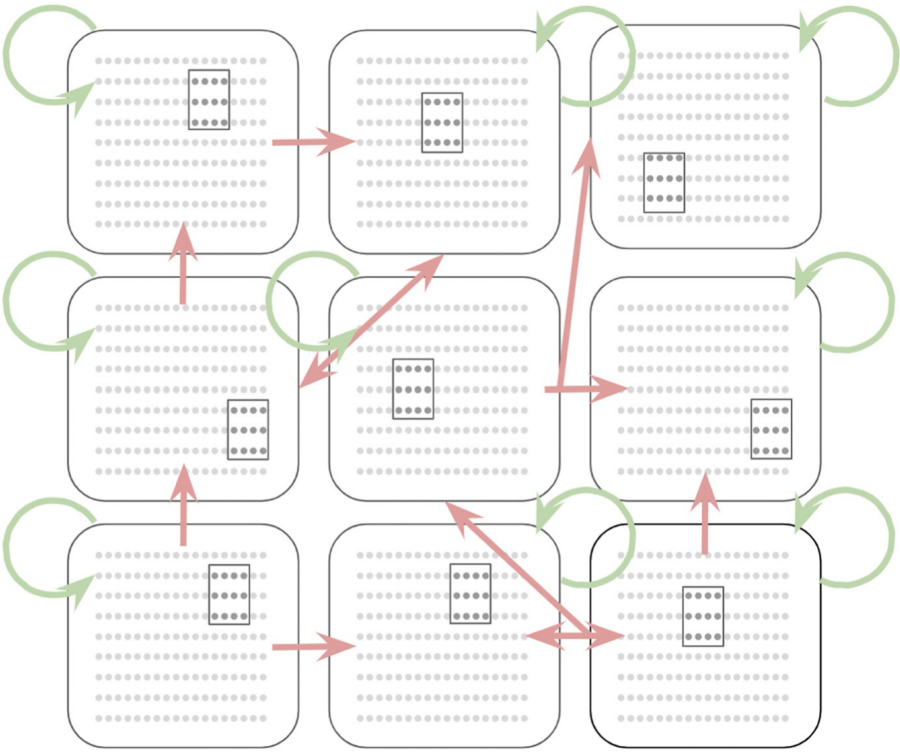


Figure 6: This figure shows the computational model proposed by Papadimitriou [5] called the assembly calculus. In general, a block represents an assembly including multiple fully connected neurons associated with a specific task. The whole block gets excitatory while receiving the associated external stimulus. For example, if the top-left assembly is related to sound, all the neurons inside it will fire when someone hears talking. Green arrows represent the recurrent connections, and red ones represent connections between two assemblies in the directions implied by the arrow symbols.

4 SANs

A formal computational model of the brain based on assemblies of neurons called the assembly calculus proposed in [5] drew our attention at this time. As shown in Fig. 6, the elementary entity of the Assembly Calculus is a set of fully connected excitatory neurons which all belong to the same brain area and can near-simultaneously fire when given a specific task [5]. Inspired by the work, we realized that one single PC layer could be suitable for implementing the single assembly structure in animal cortex, as the model can freely construct bidirectional connections between neurons. However, the assembly calculus is a theoretical model and the functionality of it has not been proven in practice. Hence, I now propose a network with the structure of a cluster of fully connected neurons called *the single assembly networks (SANs)*, and prove its functionality in classification, generation, and AM tasks.

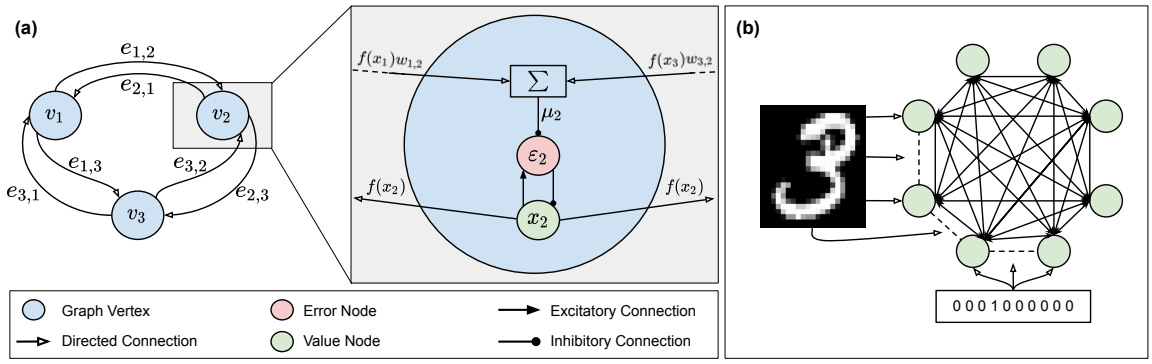


Figure 7: (a) Proposed assembly of neurons, an example with three neurons (left). Neural implementation of the proposed the assembly of neurons (right, in the gray box), where all computation is made locally via demonstrated inhibitory and excitatory connections. (b) Learning of an assembly of neurons, where the value node of every sensory neuron is fixed to the entries of $\bar{s}_{in}$. In this case, $\bar{s}_{in}$ corresponds to the pixels of an handwritten digit image and its 1-hot label.

4.1 Network architecture

The idea of assemblies is defined as *predictive coding graphs (PCGs)*, a directed complete graph of neurons that resembles a single assembly shown in Fig. 6. Let G(V, E) be a directed complete graph, where V is a set of N neurons $\{v_0, \ldots, v_N\}$ and E is a set of directed edges between them, where every edge $e_{i,j}$ has two weight parameter: $\theta_{i,j}$, which denotes the synaptic strength applied while we send values from neuron i to neuron j, and $\theta_{j,i}$ which denotes the strength from j to i.

The network randomly chooses k neurons and defines them as *sensory neurons*, which are used to receive the external stimuli. k equals the sum of the dimension of data and labels. For example, in Fig. 7(b), the assembly should have $(784 + 10)$ sensory neurons to learn MNIST.

Fig. 7(a) shows the structure of SANs in details. As in PCNs, each neuron has a value node (the green node), $x_{i,t}$, which is a trainable parameter, and a error node $\varepsilon_{i,t}$ (the red one), defined by

$$\varepsilon_{i,t} = x_{i,t} - \mu_{i,t}, \tag{18}$$

where $\mu_{i,t}$ are the predicted values generated according to:

$$\mu_{i,t} = \sum_{j \neq i}^{N} \theta_{j,i}^{t} f(x_{j,t}). \tag{19}$$

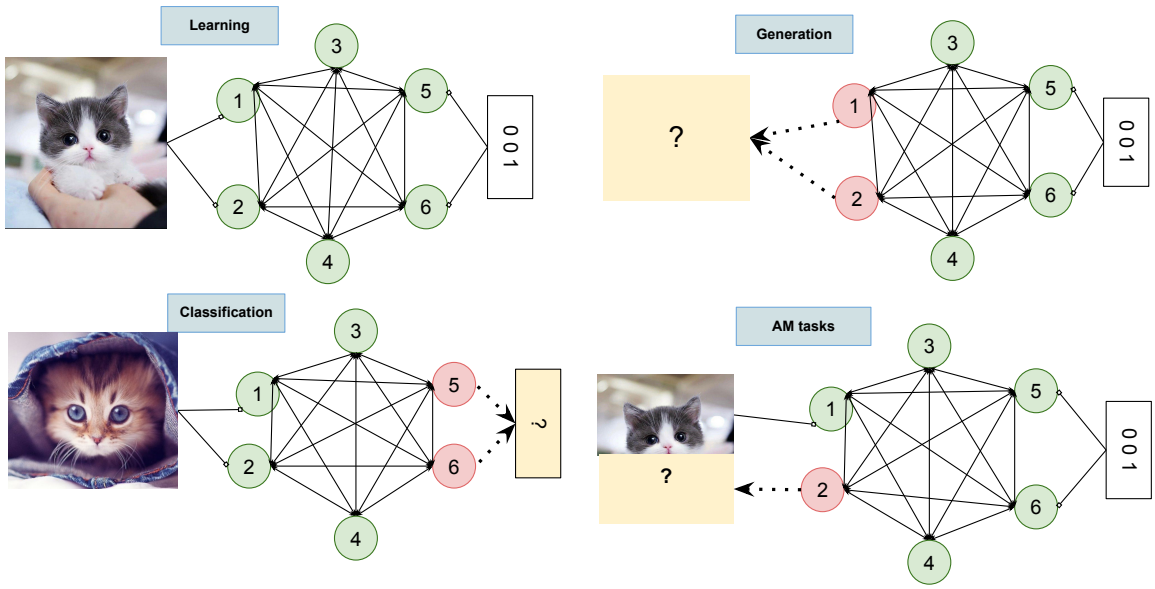


Figure 8: This figure visualizes how SANs process the information in four modes: learning (top-left), classification (bottom-left), generation (top-right), and AMs (bottom-right). Indices of neurons are used for easy description and do not imply orders. In this case, sensory neurons are neuron 1, 2, 5, and 6. Predicted values of the red nodes are the results gained in that test mode.

f is an activation function and the summation is over all the vertices v_j connected to v_i. Note that we add an index t to θ here, which represents the time axis. The goal is to minimize the energy function:

$$E_t = \sum_{i=0}^{N} \frac{1}{2}(\varepsilon_{i,t})^2. \tag{20}$$

Note that we do not need the layer index l nor two error nodes as in B-PCNs, as the assembly model does not have any hierarchical structure as standard deep learning networks. Rather than sending inference *forward* or *backward*, each neuron v_i simply sends its value $x_{i,t}$ to all other neurons $v_j \in V$, while using all $x_{j,t}$ to predict its current state and calculate the error. I now show how SANs work in the learning mode and different test modes.

Learning: The value nodes of sensory neurons are fixed to a training point $\bar{s}$. For supervised tasks, where points are labeled and are denoted as $(\bar{s}^{in}, \bar{s}^{out})$, we assume that the input vector $\bar{s}$ is simply the concatenation of the data point and its label. Since all neurons are identical, we can set the first k neurons to be sensory neurons, where k equals the sum of input dimension and output dimension ($(784 + 10)$ for

MNIST). More formally, we assume that the sensory neurons are $\{v_0,..., v_k\}$, such that

$$x_{c_1,t} = \bar{s}^{in}, c_1 \in \{0, ..., k_0\} \quad and \quad x_{c_2,t} = \bar{s}^{out}, c_2 \in \{k_0 + 1, \ldots, k\}, \tag{21}$$

where k_0 equals the input dimension.

The rest neurons are free to converge. Particularly, weight parameters and value nodes are updated simultaneously to minimize E_t according to the following equations:

$$\Delta x_{i,t} = -\gamma \cdot \partial E_t / \partial x_{i,t} = \gamma \cdot (-\varepsilon_{i,t} + f'(x_{i,t}) \sum_{k=1}^{N} \varepsilon_{k,t} \theta^t_{k,i}) \tag{22}$$

$$\Delta \theta^t_{i,j} = -\alpha \cdot \partial E_t / \partial \theta^t_{i,j} = \alpha \cdot \varepsilon_{i,t} f(x_{j,t}), \tag{23}$$

where γ is the integration step of inference, and α is the learning rates for weight updates. We use CPC to update the parameters. Inference will be running bidirectionally between every two distinct neurons. Training runs for $t = T$ iterations, or until the total energy of the model decays to 0. The learning algorithm is displayed in Alg. 5.

Algorithm 5 Learning ($\bar{s}$) in duration T, Single Assembly

Require: Fix the value nodes of sensory neurons to input $\bar{s}$
  for $t = 0$ to T **do**
    for each neuron i **do**
      Update $x_{i,t}$ to minimize E_t via Eq. (22)
      Update each $\theta^t_{i,j}$ to minimize E_t via Eq. (23)
    end for
  end for

Test: I now show SANs are task-agnostic and capable of performing the **classification**, **generation** and **AM tasks**. Similar to B-PCNs, the only difference among the three modes is the way of fixing the value nodes. The mechanisms are visualized in a simple way in Fig. 8. This thesis will focus on the supervised learning only. Hence, for the sensory neurons $\{v_0, \ldots, v_k\}$, in the classification mode, we will fix $x_{c_1,t}, c_1 \in \{0, ..., k_0\}$ to $\bar{s}^{in}_{test}$, where $\bar{s}^{in}_{test}$ denotes the test inputs we used. In generation, we fix $x_{c_2,t}, c_2 \in \{k_0 + 1, \ldots, k\}$ to $\bar{s}^{out}_{test}$, where $\bar{s}^{out}_{test}$ denotes the 1-hot labels of inputs we hope to generate.

After fixing the related sensory neurons, inference runs bidirectionally, and all other neurons in the assembly are free to update their value based on the weight parameters

Mode	$x_{c,t},\ c \in \{0,\ldots,k\}$	Results
Classification	$x_{c_1,t}, c_1 \in \{0,...,k_0\}$ to $\bar{s}^{in}_{test}$, the rest free to converge	$\mu_{c_2,t}$, $c_2 \in \{k_0+1,\ldots,k\}$
Generation	$x_{c_2,t},\ c_2 \in \{k_0+1,\ldots,k\}$ to $\bar{s}^{out}$, the rest free to converge	$\mu_{c_1,t}$, $c_1 \in \{0,...,k_0\}$
AM	fix $\bar{x'}$ to $\bar{s}^{in'}$, fix $x_{c_2,t}$ to $\bar{s}^{out}$, where $c_2 \in \{k_0+1,\ldots,k\}$, the rest free to converge	$\mu_{c_3,t}$, where $x_{c_3,t} \in (\bigcup(x_{c_1,t}) - \bar{x'},\ c_1 \in \{0,...,k_0\}$

Table 2: The table displays the difference among three test modes of the assembly model, showing which neurons should be fixed (column 1 and 2) and what we should consider as a result (column 3) of each mode.

gained from the training phase until E_t converges to zero. Tab. 2 succinctly shows the results in each mode. Generally, the generated labels of test inputs in the classification mode equals $\mu_{c_2,t}, c_2 \in \{k_0+1,\ldots,k\}$, i.e. the predicted values of sensory neurons fixed to the labels in the training phase. On the other hand, the images constructed in the generation mode equal to $\mu_{c_1,t}, c_1 \in \{0,...,k_0\}$, i.e., the predicted values of sensory neurons fixed to the inputs in the training phase.

For the AM mode, we first define the cropped images as $s^{\bar{in}'}$, following Eq. 17, and denote $\bar{x'}$ the neurons which were fixed to $s^{\bar{in}'}$ during the training phase. At the beginning of the retrieval process, we fix $\bar{x'}$ to $s^{\bar{in}'}$ and $x_{c_2,t}, c_2 \in \{k_0+1,\ldots,k\}$ to $\bar{s}^{out}$ as in the training phase. The rest neurons are free to converge. After the energy function decays to zero, the reconstructed portion should be the predicted values of the sensory neurons, which were fixed to the inputs image in the training phase yet free to converge in the AM mode, defined formally in the Tab. 2.

4.2 Density of SANs

SANs are now defined to be fully connected, and bidirectional edges exist between every two neurons in the graph. At this point, another intuitive question appears: is it possible to prune some connections and make SANs non-fully connected? Theoretically, as parameter updates rely solely on locally available information, adding or pruning edges will not influence the function of SANs. We may realize that the

feasibility of precisely carving SANs indicates the possibility of gaining B-PCNs from SANs directly. In other words, we can interpret B-PCNs as a subtype of non-fully connected SANs with specific edges pruned.

4.2.1 Randomly pruning

Before turning to the carving problem, we start with a simpler task: randomly pruning some edges from *fully connected SANs (FC-SANs)*. Specifically, we now define a matrix $\bar{\theta} \in \mathbb{R}^{N \times N}$, where $\theta_{i,j}$ represents the synaptic strength connecting neuron i to neuron j (direction implied). We then define a 1-hot mask matrix $M \in \mathbb{R}^{N \times N}$, where $M_{i,j} = 1$ if the edge $e_{i,j}$ exists, and $M_{i,j} = 0$ otherwise. For FC-SANs, $M_{i,j} = 1$ for all $i \neq j$.

Given $\bar{\theta}$ and M, we now define a parameter related to the density of SANs, s, a constant between $[0, 1]$ which means that we randomly prune $(1 - s)$ edges from SANs. Specifically, we apply s to M following:

$$M = (M' < s), \tag{24}$$

where M' is a N by N matrix consisting of random numbers from a uniform distribution on the interval $[0, 1)$. Note that for any k such that $x_{k,t}$ is a sensory neuron, $m_{k,i}$ or $m_{j,k}$ should be set to 1, or the network may fail to read inputs or print outputs. After multiplying $\bar{\theta}$ by M, we will have SANs with density s. Clearly, SANs stay fully connected when $s = 1$.

4.2.2 Carving SANs to B-PCNs

It can be realized that getting B-PCNs from SANs is easily achievable by manipulating entities of M. Take the network shown in Fig. 9 as an example. We can interpret it as applying M to FC-SANs of 6 neurons, where M equals:

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{25}$$

Fig. 9 visualizes the process of pruning.

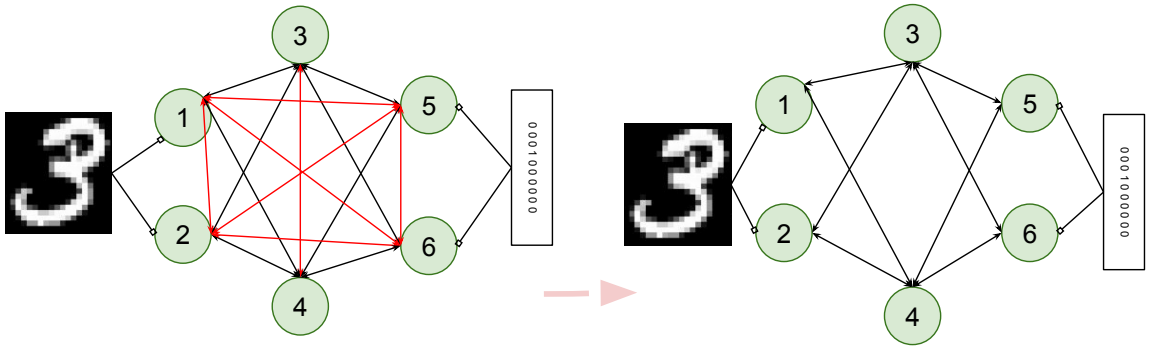


Figure 9: The figure displays how we can curve fully-connected SANs (left) to B-PCNs (right). Edges in red are those we want to prune by applying the mask matrix. Both models run inference bidirectionally along each edge.

B-PCNs can be easily constructed based on given SANs through this method.

4.2.3 Generalization

We can further generalize this method to constructing other standard neural architectures by pruning from or even adding connections to FC-SANs. Following the notations in chapter 2 and 3, in a fully connected graph $G(V, E)$, we denote the pass from neuron i to j as *forward* and j to i as *backward* if $i > j$. Therefore, it can be noticed that the entities of M **above** diagonal determine the existence of forward passes between every two neurons, and the entities **below** determine the backward ones. Those **on** the diagonal are indicators of recurrent connections. In general, the weight and mask matrices allow the creation of unidirectional architectures such as multilayer perceptrons (MLPs) and the variations (MLPs with recurrent and/or residual connections) (Fig. 10). We take SANs shown in Fig. 9 as an specific example again. To make Wittington's PCNs, we can set M to:

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{26}$$

PCNs constructed through this weight-mask method is proven to be equivalent to standard PCNs, and running it with CPC achieves 98.44% accuracy over the MNIST database as models in [12]. Therefore, this thesis confidently declares that this method

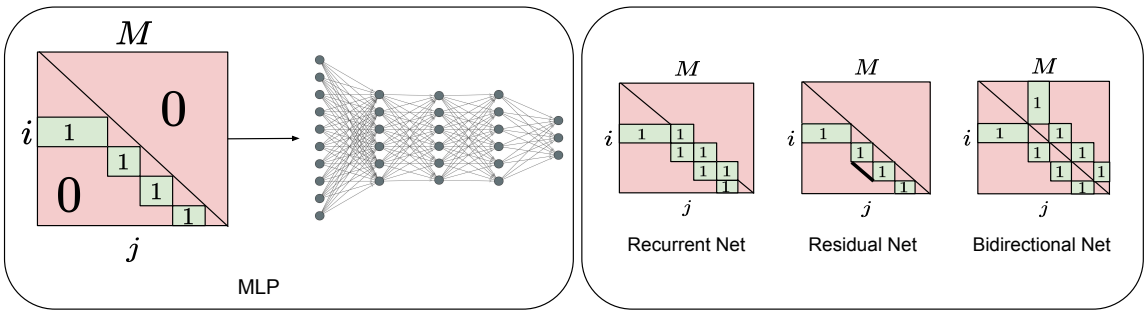


Figure 10: This figure shows the examples of networks that can be built by masking part of the weights of an assembly. Left: an example of how to build a standard MLP architecture. Particularly, the weight matrix $\bar{\theta}$ is pruned via entry-wise multiplication with the represented binary matrix M. Right, different variations of an MLP are presented. From left to right: a network with recurrent in-layer connections, a network with residual connections (the black diagonal line below the green squares represents entries equal to one), and a network with backward connections between layers. Note that the three methods can be merged in any combination: it is in fact possible to generate a recurrent network with bidirectional and residual connections by simply updating the mask M.

successfully bridges the gaps between different neural architecture without degrading the performance of models.

Detailed experiments and results related to classification, generation, and AM tasks will be included in the following chapter to prove the capability of task-agnostic learning in practice. Note that this thesis will mainly focus on B-PCNs and SANs at the experimental level, comparing their performance in three types of experiments with each other as well as with some classic neural networks.

5 Experiments and Results

In the previous chapter, I proved that task-agnostic learning is achievable by B-PCNs and SANs theoretically. I now show the performance of the two models in the classification, generation, and AM tasks and prove their capabilities experimentally.

5.1 Classification

Classification tasks are conducted over full **MNIST** dataset, a image collection including 60000 greyscale handwritten digits from 0 to 9. In general, I first conducted multiple experiments for each network by changing the number of hyperparameters to see how the structure of a network influences test accuracy. After that, I explored how long it will take for a model to get the optimal prediction accuracy by studying the number of epochs (H), the duration of presenting a training pair in the training phase (T), and a test pair in the test phase (T_{test}). Finally, I assessed the stability of each network and compared their performance with some classic neural networks.

5.1.1 B-PCNs

Experiments on model structure: As we usually did for multilayer networks, I manipulated the number of hidden layers, L, and the dimension of hidden layers, *hidden_dim*, to find their influence over the prediction accuracy. Specifically, I set $L = \{1, 2, 3, 4\}$ and $hidden\_dim = \{2048, 4096, 6144, 8000, 10000\}$. All hidden layers share an identical dimension.

Results: Increasing the number of layers does not improve the performance accordingly. From Tab. 3, we can see that B-PCNs with multiple hidden layers do not outperform B-PCNs with a single hidden layer regardless of the layer dimension. For *hidden_dim*, inadequate neurons may lower accuracy slightly. However, once hidden layers have sufficient neurons to process the information (4096 in this case), the model performance remains stable as the layer dimension increases. In Tab. 3, regardless of L, B-PCNs with $hidden\_dim = 4096$ always outperform B-PCNs with $hidden\_dim = 2048$ by approximately 2%, but perform equivalently well as B-PCNs with $hidden\_dim = 10000$.

$L \setminus hidden\_dim$	2048	4096	6144	8000	10000
1	91.19	92.69	92.24	92.11	92.34
2	90.9	92.4	92.5	**92.71**	92.66
3	89.19	92.52	91.59	91.87	91.71
4	90.48	92.61	91.92	91.45	89.71

Table 3: This table shows how the number of hidden layers (columns) and the dimension of the hidden layers (rows) influence B-PCNs' prediction accuracy over MNIST dataset. The best accuracy is bold.

Experiments on duration: While conducting the experiments, I found the number of epochs, H, shows weak relevance to the accuracy if training pairs are presented to the network in a long duration, i.e., T is large enough. So I further explored how H and duration of inference process in both the training phase and prediction phase, T and T_{test} influence the performance. Specifically, I set T to $\{10, 100, 500, 1000\}$ and T_{test} to $\{100, 250, 1000, 2500\}$, running 10 epochs for each combination of parameters.

Results: The results summarized in Tabs. 4 and 5 are consistent with intuition. H only relates to the performance when T is small, and increasing H cannot make up for the low accuracy caused by running inference for insufficient time. With the long enough duration of inference learning, B-PCNs only take 1 or 2 epochs to get the best results. For example, the accuracy increases epoch by epoch when $T = 10$, but approximately stays unchanged when $T \geq 100$. Nevertheless, running the inference lengthily (such as $T = 1000$ in this case) may cause overfitting and decrease test accuracy. T_{test} can be least important and barely influence the final results.

Overall, the best test accuracy is 92.71%, achieved by B-PCNs with 2 hidden layers, each of 8000 neurons, running inference for 100 time steps at the training phase and 250 at the prediction phase. Detailed parameters are shown in Tab. 6.

5.1.2 SANs

Experiments on model structure: For SANs, I studied how the number of neurons, N, and density of networks, s, influence the accuracy, setting $N = \{1000, 2500, 5000\}$ and $s = \{0.01, 0.05, 0.1, 0.5, 1\}$.

Results: Fully connected SANs with 5000 neurons show the best performance in classification, providing the test accuracy of 91.77%. It turns out that density can be

$H\backslash T$	10	100	500	1000
0	82.15	91.37	90.39	88.63
1	86.27	91.85	92.51	88.83
2	87.26	92.4	90.62	88.21
3	87.58	92.27	92.06	88.79
4	87.95	92.12	91.21	89.38
5	87.62	92.02	92.59	89.02

Table 4: This stable shows the relationship between the test accuracy over MNIST dataset and the number of training epochs (columns) and the inference duration (rows). $L = 2$ and $hidden\_dim = 4096$.

$H\backslash T_{test}$	250	2500
0	82.15	82.08
1	86.27	85.61
2	87.26	87.07
3	87.58	87.23
4	87.95	87.99
5	87.62	87.48

Table 5: This table shows the relationship between the test accuracy over MNIST dataset and the number of training epochs (columns) and the inference duration in the test phase (rows). $L = 2$, $hidden\_dim = 4096$, $T = 10$.

the most influential parameter. Regardless of the number of neurons, the performance of SANs gets better as fewer connections are pruned, and fully connected SANs perform the best. For example, in Tab. 7, the prediction accuracy is constantly below 80% when density is only 0.01, whereas constantly above 90% when the network is fully connected. In terms of the number of neurons, N shows a stronger influence over the performance for sparser SANs. For example, for the networks with density 0.01, the test accuracy when $N = 1000, 2500, 5000$ are 42.12%, 47.55% and 73.55%, whereas for fully connected SANs with 1000, 2500, 5000 neurons, the accuracy are 91.45%, 91.76%, 91.77% (Tab. 7).

It turns out that to function CPC in both networks, we need to have the learning rate, α, at the level of 1×10^{-5}, and the integration step for inference, γ, at the level of 1×10^{-1}.

Parameters	B-PCNs
Input Dim	28×28
Output Dim	10
Hidden Dim	8000
L	2
γ	0.5
α	1×10^{-5}
T	100
T_test	250
H	2
Batch Size (training)	500
Batch Size (test)	2000

Table 6: Parameters of the best model for classification over MNIST (B-PCNs).

$N \backslash s$	0.01	0.05	0.1	0.5	1
1000	42.12	77.23	82.43	87.88	91.45
2500	47.55	76.03	83.79	84.47	91.76
5000	73.55	84.35	85.45	86.71	**91.77**
8000	77.09	86.38	88.0	88.88	89.76

Table 7: This figure shows the relationship between the test accuracy and the number of neurons (columns) and the density (rows) of an assembly model. The best accuracy is bold

5.1.3 Performance and stability

Overall accuracy: As displayed in Tab. 9, B-PCNs and SANs both show relatively good performance in classification tasks. When we compare the results with the test accuracy achieved by MLPs with BP [12], PCNs with CPC [12], and Hopfield networks [1], we can see that both models make prediction slightly less accurate than standard MLPs or PCNs, but significantly outperforming the generative AMs, Hopfield networks. Hopfield networks have been modified by [1] to perform discriminative tasks to achieve task-agnostic learning, as the primary goal of this thesis. Clearly, B-PCNs and SANs provide more promising results.

Stability: Additionally, the stability of both networks was examined by the test accuracy of each digit. Results are displayed in Tab. 10, compared with results of Hopfield networks. It turns out that B-PCNs can be considered the most stable, showing at least 83.23% accuracy for all digits. SANs can be slightly less stable, but

Parameters	Single Assembly
N	5000
wd (weight decay rate)	0.5
s	1
γ	0.5
α	5×10^{-5}
N. of iterations (training)	5
N. of iterations (test)	2000
N. of epochs	10
Batch Size (training)	250
Batch Size (test)	2000

Table 8: Parameters of the best model for MNIST image classification (SANs).

Classic MLPs with BP	Belyaev's Hopfield network	Feed-forward PCNs with CPC	B-PCNs	SANs
98.41	61.5	98.54	92.27	91.77

Table 9: This table shows the overall test accuracy achieved by five models over MNIST dataset. From left to right: standard deep learning network with BP [9], Belyaev's Hopfield network [1], Wittington's PCNs with Song's CPC algorithm [12], B-PCNs, SANs.

the results are still better than those achieved by Hopfield networks.

In summary, both models have an outstanding ability to perform classification tasks with stability. To state they are task-agnostic, we now show that they can accomplish AM tasks.

5.2 AM tasks

Introduced in the preliminary section, associative memory tasks can be considered a subtype of memorization problems. Hence, models are supposed to reconstruct previously learned images rather than new untrained images. In other words, the word "reconstruction" in this section explicitly means retrieval of trained images based on the corrupted version. Datasets used are subsets of (1) **MNIST**, (2) **FashionMNIST**, which includes grayscale images of items ten types of clothing, such as shoes, trousers, dresses, and more in the dimension of 28×28, and (3) **CIFAR10**, a collec-

Digit	Belyaev's Hopfield network	B-PCNs	SANs
0	69	95.06	96.15
1	90.2	95.06	97.12
2	73.7	86.27	79.51
3	67.7	83.23	83.27
4	0	85.53	89.44
5	52.4	87.1	75.31
6	80.4	88.72	91.79
7	69.6	88.65	89.48
8	57.3	89.32	63.39
9	29.9	87.18	75.58

Table 10: This table shows the test accuracy of each digit in the MNIST database. For left to right, the accuracy is achieved by Belyaev's Hopfield network [1], B-PCNs and SANs.

tion of 32×32 RGB images in ten different classes including airplanes, cars, birds, cats, etc.

To quantify the quality of reconstructed images, this thesis uses **mse** to describe the standard mean-squared error between two images (Eq. 27).

$$mse = MSE(img_1 - img_2). \tag{27}$$

In this thesis, if $mse \leq 0.01$, the reconstruction is considered "correct"; if $mse \leq 0.005$, the reconstruction is considered "excellent". The rate of correct retrieval, H_c, and the rate of excellent reconstruction, H_e, will be used to analyze a network's performance. Further, we denote the cardinality of datasets used for reconstruction by Num and the fraction of the size of cropped images to the size of original images by $frac$. When the quality of results is equivalent, larger Num and smaller $frac$ imply stronger capability and capacity of a model in AM tasks. We start with $Num = 30$ and $frac = 1/2$ for each model, and if the performance is satisfactory, we then ask the model to retrieve larger datasets with small fraction pixels.

5.2.1 B-PCNs

Experiments: Experiments were conducted to analyze B-PCNs' performance and find the optimal structure by changing L and $hidden\_dim$. Specifically, I set the number of hidden layers to $\{1, 2, 3, 4\}$ and the dimension of hidden layers to {4096,

Figure 11: This figure shows examples of B-PCNs' reconstruction of images from MNIST (left), FashionMNIST (centre) and CIFAR10 (right) database.

6144, 8000}. The inference was running for more than 10000 time steps in each case until the network converged. After the learning phase, the top half of images were cropped, and networks were expected to construct the whole based on the partial.

Results: In general, though theoretically capable of AM tasks, in practice, B-PCNs showed a relatively disappointing performance. No structure outperforming the rest in AM tasks has been detected. Samples of the reconstruction are displayed in Fig. 11. For MNIST and FashionMNIST images, though objects are identifiable in the right place with clear shape, some dead pixels are also incorrectly restored. The results of CIFAR10 seem to be less reasonable.

5.2.2 SANs

Though disappointed by B-PCNs to some extent, we then delightfully find out that SANs perform reasonably well in AM tasks in practice.

Experiments on model structure: I first performed multiple experiments by changing the number of neurons, N, and the density of an assembly, s, to see how the combination influences the model performance and find the optimal structure for each type of data. Values assigned to N and s in the experiments are shown in Fig. 12.

Results: Fully-connected SANs with 8000 neurons perform the best while reconstructing MNIST and FashionMNIST images. For MNIST, the model retrieves 97% images correctly and 80% excellently; for FashionMNIST, 67% reconstructed images are correct, and 43% are excellent. For CIFAR10, SANs which have 8000 neurons and density 0.8 can reconstruct 73% images correctly and 43.33% images excellently (Fig. 12), outperforming other pairs of (N, s). Samples of best reconstructed images are displayed in Fig. 14.

It can be seen that N and s influence the performance as a combination. In general, denser SANs with more neurons guarantee a more robust ability in data retrieval before overfitting is detected. We may also see from Fig. 12 that the performances of sparser SANs with more neurons and denser SANs with fewer neurons can be equivalent. For example, both a 4000-neuron SAN with 0.3 density and a 1000-neuron assembly with density 0.5 can retrieve 20% FashionMNIST images correctly and 10% excellently, yet a 4000-neuron assembly with density 1 is capable of reconstructing 56.66% correctly and 33.33% excellently.

However, density plays a more important role between the two parameters, as an assembly with plenty of neurons in extremely sparse structure never shows satisfactory performance for three types of images. For example, 10000-neuron SANs with $s = 0.1$ cannot correctly reconstruct any MNIST images.

Enlarging the size of assemblies by adding more neurons only significantly improves the model performance while the assembly is currently small and has a density no less than 0.3. This can be intuitive since more neurons usually imply a stronger ability of memorization. However, adding neurons to a large assembly shows little enhancement in performance. For example, when SANs with density 0.8 are reconstructing MNIST images, H_c increases 70% and H_e increases 40% as N goes from 1000 to 7000, but both remain approximately unchanged as N goes from 7000 to 10000 (Fig. 12).

Samples displayed in Fig. 13 support the statistical analysis. Noises are obvious in MNIST images reconstructed by SANs with density 0.1 but can be barely seen in those reconstructed by SANs with density 1, and a dense 8000-neuron assembly can reconstruct CIFAR10 images with little noise and bright color as a 12000-neuron assembly.

Since SANs perform well for small datasets, experiments were then conducted to further explore the model capability and capacity in AM tasks.

Experiments on model capability: The networks were then presented with a fewer fraction of pixels so that we can observe how the reconstruction quality changes. Specifically, I set $frac = \{1/2,\ 1/3,\ 1/4,\ 1/5,\ 1/6\}$. The cardinality of datasets is equal to 30 as in the previous section.

Results: It turns out both the rate of correct and excellent retrieval are much higher when every image is half presented to the model. Even reducing $frac$ from 1/2

Parameters	MNIST	FashionMNIST	CIFAR10
N	8000	8000	8000
weight decay	1	1	1
s	1	1	0.8
γ	0.5	0.5	0.5
α	5×10^{-5}	5×10^{-5}	5×10^{-5}
T	10000	10000	10000
T_test	250	250	250
N. of epochs	1	1	1

Table 11: This table includes parameters of the best SANs in AM tasks. Full-batch training is used in all cases.

to 1/3 significantly degrades the quality of reconstruction. For example, SANs can reconstruct 97% FashionMNIST images correctly when presented with 1/2 of total pixels but only 17% when presented with 1/3 (Fig. 15), and the rate of excellent reconstruction is 43% when $frac = 1/2$ but only 7% when $frac = 1/3$. Samples displayed in Fig. 15 directly reveal such degrading, as at $frac = 1/2$, the van in the reconstruction looks identical to the original but at $frac = 1/6$ the dog in the reconstruction seems not identifiable.

Experiments on model capacity: Simultaneously, I studied the network's capacity in image pattern storage by increasing the cardinality of datasets and observing the H_c and H_e. Specifically, Num is set to $\{30, 50, 100, 250, 500\}$.

Results: It turns out that the performance of the model degrades sharply as the cardinality of datasets increases. For example, if the network is asked to retrieve 50 MNIST images rather than 30, the rate of correct retrieval reduces to 75%, and further reduces to below 25% when the cardinality increases to 100. Results of FashionMNIST and CIFAR10 are consistent with the statement, and samples shown in Fig. 16 visualize the change in the performance. When $Num = 500$, the shape of the reconstructed digit 2 is blurry, and its tail was barely reconstructed.

In summary, B-PCNs can perform AM tasks though further improvement in reconstruction quality might be needed. SANs reconstruct small sets of grayscale and color images remarkably well, though we have to admit that the performance degrades while a smaller quantity of pixels is presented or the size of datasets increases. Nevertheless, they both show the capability of AM tasks, and we now apply them to perform generation tasks to demonstrate their capability.

Parameters	MNIST	FashionMNIST
Input Dim	28×28	28×28
Output Dim	10	10
Hidden Dim	4096	4096
N. of hidden layers	2	2
Activation function	RELU	RELU
γ	0.5	0.5
α	1×10^{-5}	5×10^{-5}
wd	1	1
N. of iterations	10	10
N. of iterations (test)	250	250
N. of epochs	1	1
Batch Size (training)	500	500
Batch Size (test)	2000	2000

Table 12: This table includes parameters of B-PCNs which can generate the MNIST and FashionMNIST images of highest quality (samples displayed in Fig. 17)

5.3 Generation

5.3.1 Experiments

Experiments were conducted over **MNIST** and **FashionMNIST** database. After the training phase, labels of ten classes were presented to both networks, and they were expected to generate images based on the labels.

5.3.2 Results

Both networks showed equivalently satisfactory performance in this task. Samples of generation of MNIST and FashionMNIST images are displayed in Fig. 17, and parameters of the networks which achieve the results are displayed in Tabs. 12 and 13. For B-PCNs, the network with 2 hidden layers in the dimension of 4096 can accomplish the generation. For SANs, unlike in the AM tasks, a sparser assembly generates images of higher quality. Overall, SANs with 6000 neurons and density 0.1 generate MNIST images from all classes satisfactorily, and SANs with 6500 neurons and density 0.1 achieve the equivalent success for FashionMNIST. No apparent difference between the quality of images generated by the two models has been detected.

Parameters	MNIST	FashionMNIST
N	6000	6500
wd (weight decay rate)	0.1	0.1
s	0.1	0.1
γ	0.52	0.5
α	3×10^{-5}	3×10^{-5}
Activation function	Sigmoid	Sigmoid
N. of iterations (training)	50	50
N. of iterations (test)	5×10^3	5×10^3
N. of epochs	1	1
Batch Size (training)	200	200
Batch Size (test)	1×10^4	1×10^4

Table 13: This table includes parameters of SANs which can generate the MNIST and FashionMNIST images of highest quality (samples displayed in Fig. 17)

5.4 Reconstruction beyond memorization

So far, experimental results demonstrate that both networks can perform the classification, generation, and AM tasks, and SANs even show satisfactorily good performance in all three tasks. Before turning to the conclusion, I now further display some exciting findings from another type of reconstruction experiment beyond the scope of AMs.

Inspiration came from a reconstructed image made by SANs in AM tasks. In Fig. 18, we see that the assembly reconstructed a dress into a pair of trousers based on the top half of the dress. Such a mistake can be reasonable when we look at the trousers, as the top half of both images are similar. This might indicate that rather than constantly memorizing stiffly, the assembly network can learn the patterns and predict the cropped portion when presented with the top half. Therefore, a hypothesis is proposed that besides reconstructing trained images, SANs can also reconstruct partially cropped images from a test set after learning the training set. The process is similar to the classification task: the network will be trained over large-size image sets to learn the patterns and then predict the missing part of the partial test image presented to it.

5.4.1 Experiments

To apply SANs to reconstruct untrained images, we can easily realize that the mechanism behind it is identical to what happens in the AM test mode, discussed in chapter 4, except that the sensory neurons will be set to the partial untrained images and their labels. I then studied SANs capability by manipulating the model structure, i.e., N and s and aimed to find the best model.

Other factors that influence the model's performance can be the duration of inference learning, T, and the size of training sets, Num. Intuitively, as in classification, learning more extensive training sets and running inference for a longer time may enable the networks to understand the image patterns better, and accordingly provide a higher quality reconstruction. However, in practice, we must balance the two factors, as setting both to large values causes great decrements in the program's running speed. Hence, two ideas guide the way of experiments: (1) having an extensive training set, i.e., 30000 images, but running inference for a relatively short period (i.e., $T = 3$), or (2) running inference long enough until the network converges, i.e., 10000 time steps, but using a much smaller training set, such as a set of 3000 images.

5.4.2 Results

Samples of the best-generated images from an untrained dataset are displayed in Fig. 18, achieved by fully connected SANs with 13000 neurons. The cardinality of the training set is 5000, and the training session lasts 10000 time steps and 1 epoch. Besides, unlike the AM tasks which full-batch training provide the best results, a small batch size has been found to be able to perform better. Specifically, I set batch size to 50 to get the results in Fig. 18.

Though the quality of the images is not remarkably high, digits are accurately constructed, and the shape of each digit is relatively precise. This indicates that SANs are showing a competitive ability in learning the image patterns and can make good prediction besides memorizing it.

Num = 30, fraction = 1/2											
MNIST, mse<0.01	0.1	0.3	0.5	0.8	1	MNIST, mse<0.005	0.1	0.3	0.5	0.8	1
1000	0%	0%	0%	23.33%	53.33%	1000	0.00%	0.00%	0.00%	0%	3.33%
2000	0%	6.66%	26.66%	60.00%	80.00%	2000	0.00%	0.00%	3.33%	6.66%	37%
3000	0%	6.66%	36.66%	70.00%	90.00%	3000	0.00%	0.00%	6.66%	23.33%	43.33%
4000	0.00%	23.33%	56.66%	86.66%	90.00%	4000	0.00%	0.00%	6.66%	33.33%	43.33%
5000	0.00%	26.66%	56.66%	90.00%	90.00%	5000	0.00%	3.33%	6.66%	46.66%	46.66%
6000	0.00%	26.66%	76.66%	93.33%	93.33%	6000	0.00%	6.66%	13.33%	50.00%	60.00%
7000	0.00%	36.66%	80.00%	96.66%	96.66%	7000	0.00%	6.66%	30.00%	56.66%	66.66%
8000	0.00%	46.66%	73.33%	96.66%	96.66%	8000	0.00%	3.33%	23.33%	56.66%	80.00%
9000	0.00%	43.33%	76.66%	96.66%	96.66%	9000	0.00%	6.66%	26.66%	56.66%	73.33%
10000	0.00%	33.33%	83.33%	96.66%	96.66%	10000	0.00%	3.33%	30.00%	56.66%	80%

Num = 30, fraction = 1/2											
Fmnist, mse<0.01	0.1	0.3	0.5	0.8	1	Fmnist, mse<0.005	0.1	0.3	0.5	0.8	1
1000	3.33%	13.33%	20.00%	36.66%	36.66%	1000	3.33%	3.33%	10%	17%	20.00%
2000	10.00%	16.66%	33.33%	46.66%	53.33%	2000	3.33%	10.00%	13.33%	20.00%	33.33%
3000	10.00%	23.33%	50.00%	53.33%	56.66%	3000	3.33%	10.00%	16.66%	26.66%	33.33%
4000	10.00%	20.00%	50.00%	53.33%	56.66%	4000	3.33%	10.00%	20.00%	33.33%	33.33%
5000	13.33%	40.00%	50.00%	56.66%	60.00%	5000	3.33%	16.66%	13.33%	33.33%	33.33%
6000	13.33%	40.00%	53.33%	56.66%	63.33%	6000	6.66%	13.33%	30.00%	36.66%	46.66%
7000	13.33%	46.66%	50.00%	56.66%	63.33%	7000	6.66%	16.66%	26.66%	33.33%	46.66%
8000	13.33%	46.66%	53.33%	63.33%	66.66%	8000	6.66%	16.66%	26.66%	43.33%	46.66%
9000	13.33%	50.00%	53.33%	60.00%	66.66%	9000	6.66%	16.66%	30.00%	36.66%	46.66%
10000	13.33%	50.00%	53.33%	60.00%	63.33%	10000	10.00%	16.66%	33.33%	40.00%	40%

Num = 30, fraction = 1/2											
Cifar10, mse<0.01	0.1	0.3	0.5	0.8	1	Cifar10, mse<0.005	0.1	0.3	0.5	0.8	1
4000	6.66%	36.66%	43.33%	60.00%	63.33%	4000	6.66%	10%	13.33%	30%	33.33%
5000	20.00%	43.33%	56.66%	63.33%	66.66%	5000	6.66%	16.66%	30%	33.33%	33.33%
6000	23.33%	53.33%	60.00%	70.00%	73.33%	6000	6.66%	26.66%	36.66%	36.66%	40.00%
7000	30.00%	56.66%	60.00%	70.00%	73.33%	7000	6.66%	33.33%	36.66%	43.33%	33.33%
8000	43.33%	60.00%	60.00%	73.33%	70.00%	8000	10.00%	40.00%	36.66%	43.33%	33.33%
9000	43.33%	60.00%	60.00%	73.33%	66.66%	9000	13.33%	36.66%	43.33%	40.00%	30.00%
10000	46.66%	60.00%	73.33%	66.66%	73.33%	10000	20.00%	36.66%	36.66%	36.66%	33.33%
11000	43.33%	60.00%	70%	73.33%	73.33%	11000	20.00%	40%	40%	36.66%	33.33%
12000	50.00%	60.00%	66.66%	73.33%	63.33%	12000	20.00%	40%	40%	36.66%	30%
13000	46.66%	60.00%	66.66%	73.33%	73.33%	13000	20.00%	40%	43.33%	33.33%	33.33%

Figure 12: This figure statistically shows how the combination of the number of neurons and the network density of SANs, (N, s), influence the rate of correctly (left) and excellently (right) retrieved images. Experiments were conducted over three datasets, MNIST (top), FashionMNIST (centre) and CIFAR10 (bottom). In each sequential palette, a single cell is located by (N, s) and darker color represents larger H_c or H_e values, indicating that a model with N neurons and density s performs better in the AM tasks.

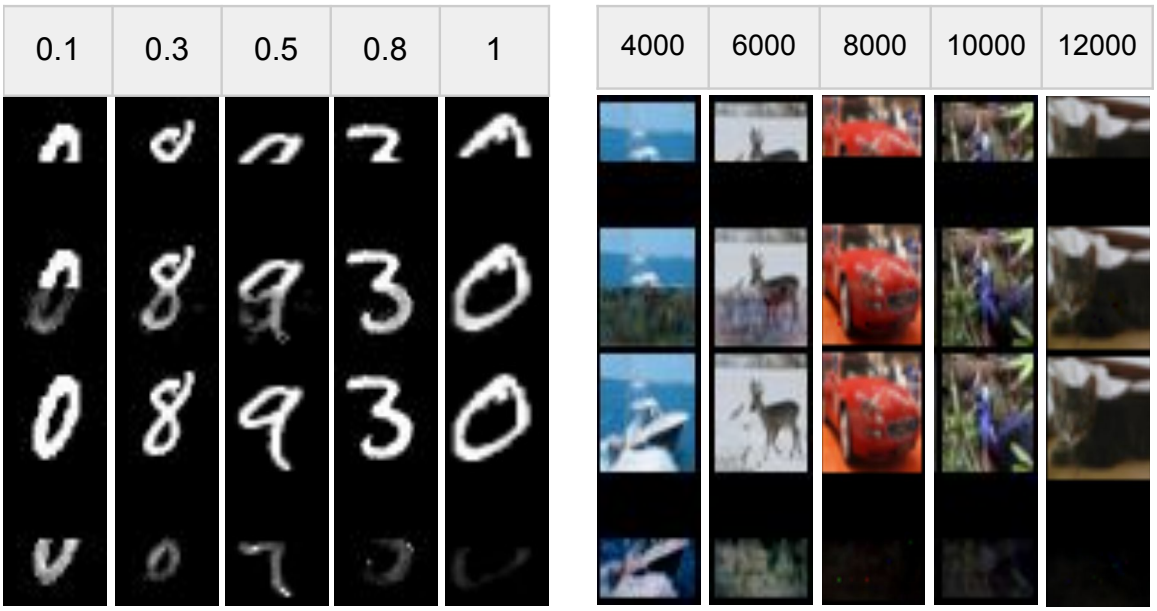


Figure 13: This figure provides samples of reconstruction made by models which have the same number of neurons but different density (left), or which have the same level of density but different number of neurons (right). Each image is displayed in the format from top to bottom: the **cropped** image presented to the model, the **reconstruction** made by the model, the **original** image, the **noise** gained by subtracting the reconstruction from the original.

Figure 14: Samples of the best reconstruction made by SANs for MNIST (left), FashionMNIST (centre) and CIFAR10 (right) images. Parameters of the models are shown in Tab 11). Overall, each reconstructed image looks identical to the original via human eyes. For MNIST and FashionMNIST, objects are accurately reconstructed with little noise, and for CIFAR10, the model also precisely retrieve details from both the objects and the background, as well as the color information.

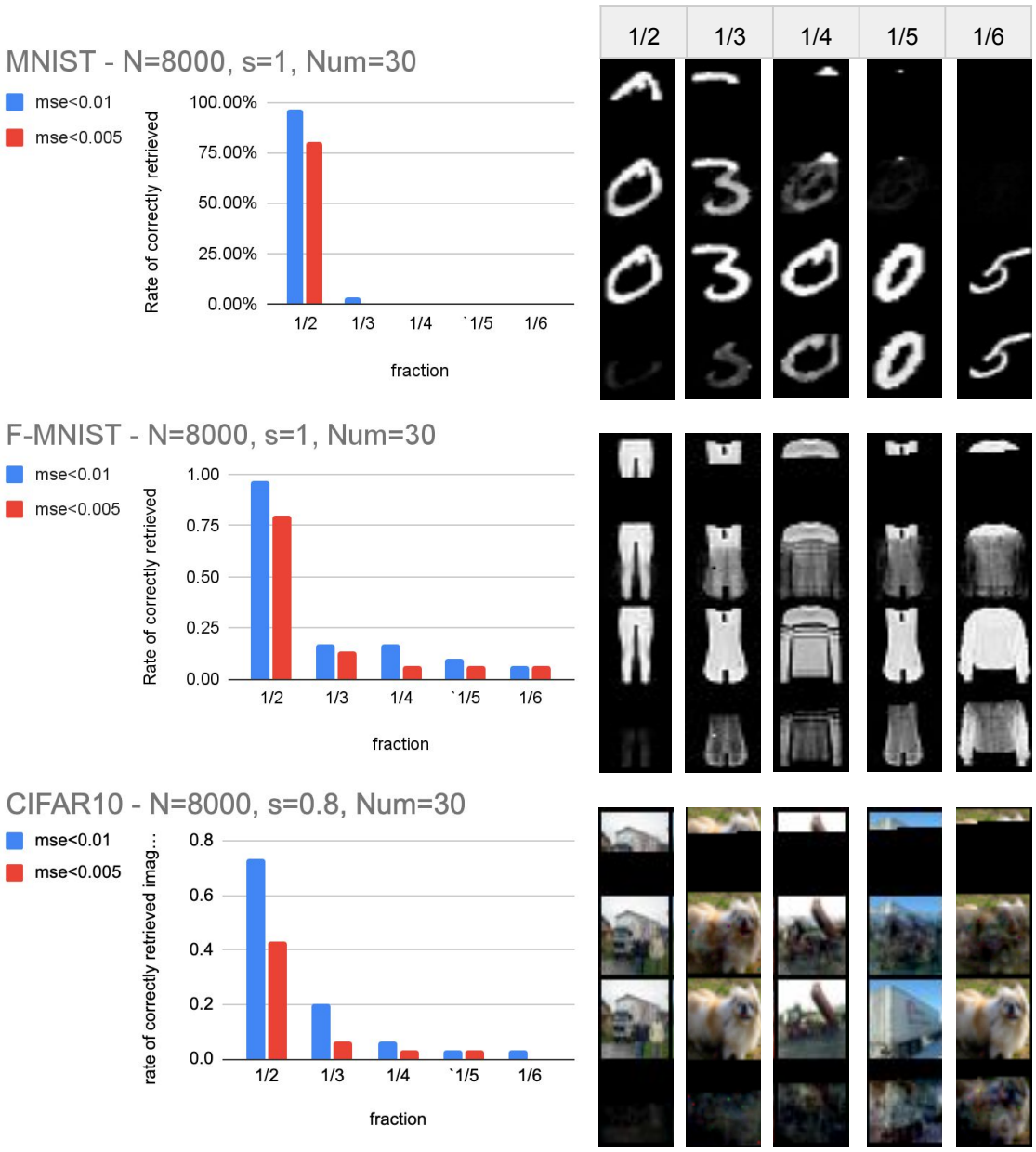


Figure 15: This figure shows how the fraction of pixels presented to SANs influences the rate of correct (blue) and excellent (red) reconstruction in a dataset of 30 MNIST (top), FashionMNIST (centre) and CIFAR10 (bottom) images. Images on the right are samples of reconstruction while $frac$ pixels are provided, where $frac = \{1/2, 1/3, 1/4, 1/5, 1/6\}$. See Fig. 13 for detailed explanations for the format of sample displays.

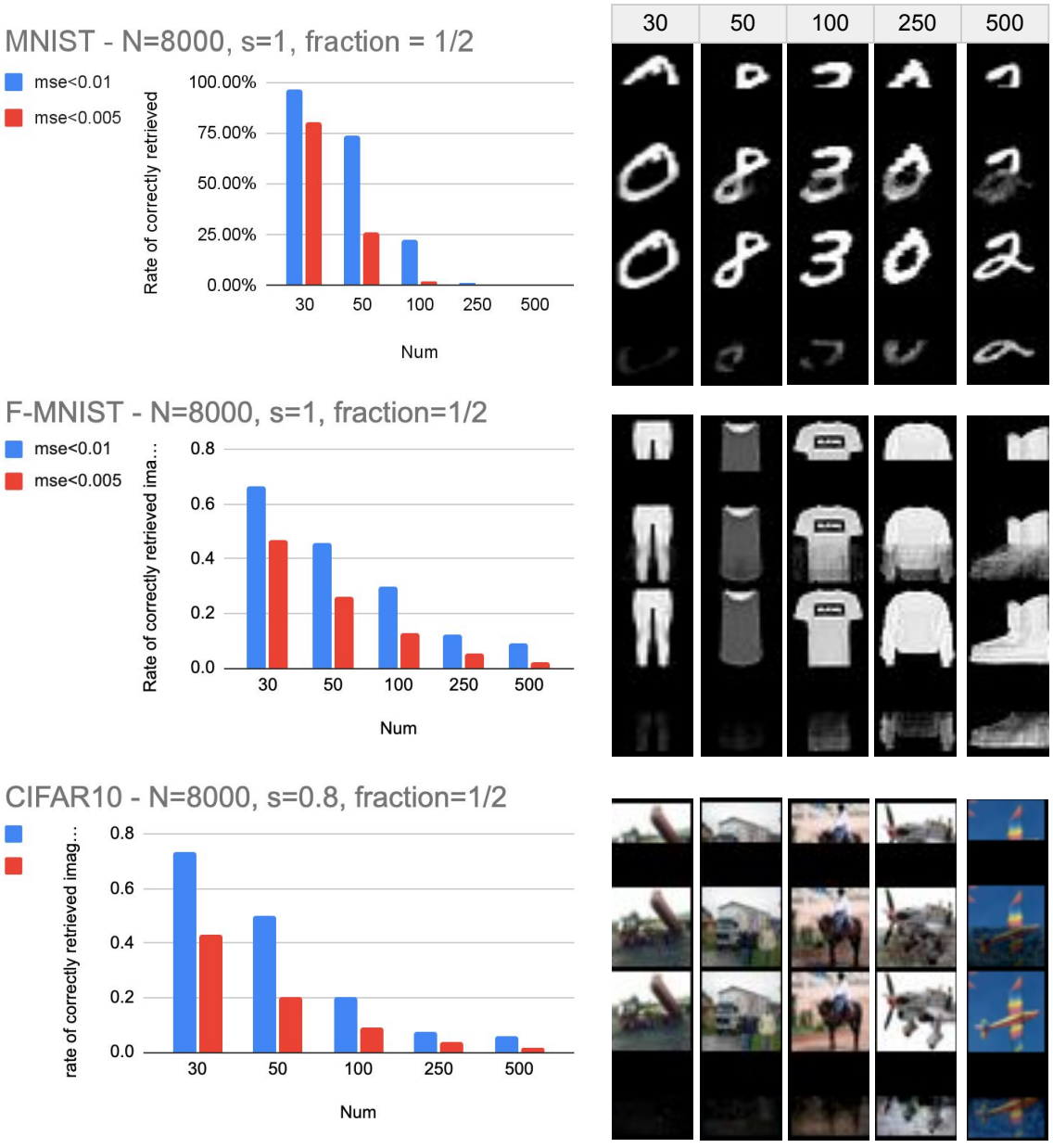
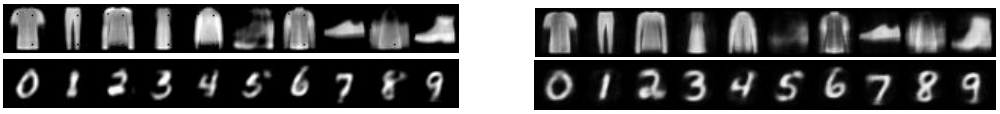


Figure 16: This figure shows how the cardinality of a dataset influences the rate of correct (blue) and excellent (red) reconstruction. Images on the right are samples of reconstructed images while models learn and retrieve a training set in the size of Num, where $Num = \{30, 50, 100, 250, 500\}$. See Fig. 13 for detailed explanations of the sample-displaying format.



(a) Generated by B-PCNs (b) Generated by SANs

Figure 17: This figure shows the generation results of MNIST (top) and FashionMNIST (bottom) images, constructed by B-PCNs (a) and SANs (b). Parameters of the related networks are shown in Tabs. 13 and 12.

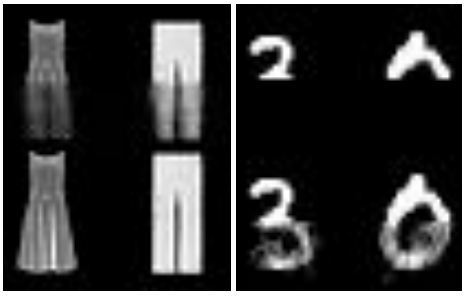


Figure 18: Left: what triggers us to conduct this experiment. While we compare the retrieval of a dress gained in AM tasks (top) and the original (bottom), we may realize that with the top half images provided, an assembly reconstructs the dress to a pair of trousers, which can be an indicator of prediction. We can see from the pictures that the top half of a dress and a trouser look similar to each other.
Right: samples of reconstructed MNIST images from a **test set** consisting of 50 **new** images. The assembly was presented with 1/2 fraction pixels of the original (top) to perform reconstruction (bottom).

6 Conclusions and future work

6.1 Conclusions

In this thesis, I propose a novel hierarchical multilayer PC model, bidirectional predictive coding networks (B-PCNs), based on Wittington's PCNs with Song's CPC algorithm. Then I break the multilayer structure and generalize B-PCNs to a novel model consisting of a cluster of neurons without hierarchical structure, the single assembly networks (SANs). Application of B-PCNs and SANs to classification, generation, and AM tasks demonstrate their capability of task-agnostic learning. In general, both models perform equivalently well even compared to some classic task-specific models. For classification, B-PCNs achieves the best prediction accuracy of 92.71%, and SANs accomplish the best of 91.77%. For AM tasks, B-PCNs shows reasonable capability of reconstructing grayscale images, and fully connected SANs retrieve 97% MNIST images and 67% FashionMNIST images correctly when the cardinality of datasets presented is not large. Simultaneously, the generation results of both models are of relatively high quality. Therefore, both models are deemed be task-agnostic. They can flexibly switch test modes in each mode after the training phase and have the capability of performing each type of task relatively well.

Further, I propose a method that can precisely carve the structure of SANs by multiplying a 1-hot mask matrix M to the weight matrix $\bar{\theta}$. This allows the creation of various types of neural networks such as MLPs and B-PCNs, achieving an extraordinary efficiency in architecture construction. Moreover, MLPs constructed by the manipulation of SANs' weights have been proven to achieve equivalently high prediction accuracy in the classification of MNIST images as [12].

Excitingly, it has been found that SANs can reconstruct corrupted images from a new test set after learning an extensive training set, which indicates that the model can not only memorize trained images but use its knowledge to predict image patterns.

Findings of this thesis are believed to be of both theoretical and practical importance — the achievement of task-agnostic learning bridges the disciplines of theoretical neuroscience and machine learning and significantly increases the biological plausibility of deep neural networks.

6.2 Future work

6.2.1 Improving B-PCNs' performance in AM tasks

One can aim to improve B-PCNs' performance in AM tasks, figuring out ways to reduce the noise of current retrieval. Further, one can find better methods to apply B-PCNs to process RGB images, reconstructing CIFAR10 images with more identifiable objects and less noise.

6.2.2 Improving SANs' capacity in AM tasks

Now SANs only show satisfactory performance in AM tasks while the cardinality of the dataset is small and half pixels are presented. Therefore, finding a way to generalize its capability to retrieve larger datasets with fewer pixels presented can be a direction for future research.

6.2.3 Improving SANs' performance in the reconstruction of untrained images

In chapter 5.4, reconstruction of untrained images has been accomplished, yet the quality can be further improved. One may figure out a way to reconstruct the test images of higher quality or apply SANs to reconstruct new FashionMNIST and CIFAR10 images.

6.2.4 Generalizing the carving method

This thesis has theoretically proved that standard neural networks can be constructed by pruning specific edges of fully connected SANs. However, in practice, it only includes experimental results of MLPs and B-PCNs constructed via this method. In the future, One may try different variations of MLPs and experimentally demonstrate their equivalence to the models constructed via classic methods.

Bibliography

[1] M A Belyaev and A A Velichko. Classification of handwritten digits using the hopfield network. *IOP Conference Series: Materials Science and Engineering*, 862:052048, may 2020.

[2] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[3] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[4] D. Mumford. On the computational architecture of the neocortex. *Biological Cybernetics*, 66(3):241–251, Jan 1992.

[5] Christos H. Papadimitriou, Santosh S. Vempala, Daniel Mitropolsky, Michael Collins, and Wolfgang Maass. Brain computation by assemblies of neurons. *Proceedings of the National Academy of Sciences*, 117(25):14464–14472, 2020.

[6] Adityanarayanan Radhakrishnan, Mikhail Belkin, and Caroline Uhler. Overparameterized neural networks implement associative memory. September 2020.

[7] Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, Jan 1999.

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[9] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323:533–536, 1986.

[10] Tommaso Salvatori, Thomas Lukasiewicz, Yuhang Song, Zhenghua Xu, Yujian Hong, and Rafal Bogacz. Associative memories via predictive coding. *Manuscript*, 2021.

[11] Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation? — exact implementation of backpropagation in predictive coding networks. *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, December 2020.

[12] Yuhang Song, Tommaso Salvatori, Thomas Lukasiewicz, Zhenghua Xu, Lei Sha, and Rafal Bogacz. Parallel predictive coding: A biologically inspired learning algorithm. *Manuscript*, 2021.

[13] Karl Steinbuch. Adaptive networks using learning matrices. *Kybernetik*, 2(4):148–152, Feb 1965.

[14] Wei Sun and Jeff Orchard. A predictive-coding network that is both discriminative and generative. *Neural Computation 32, 1836–1862 (2020)*, May 2020.

[15] Paul Werbos and Paul John. Beyond regression : new tools for prediction and analysis in the behavioral sciences /. 01 1974.

[16] James Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, 29:1–34, 03 2017.