Karlsruher Institut für Technologie

Institut für Telematik
Pervasive Computing Systems / TECO
Leiter: Prof. Dr.-Ing. Michael Beigl

Weights Assimilation for Split Manufacturing of printed Neuromorphic Circuits

Bachelor´s Thesis

by

Siyan Li

Department of Informatics

Responsible Supervisor:	Prof. Dr. Michael Beigl
Supervising Staff:	Haibin Zhao
Project Period:	30.08.2022 - 30.12.2022

Karlsruher Institut für Technologie

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und weiterhin die Richtlinien des KIT zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 30. 12. 2022

Siyan Li

Unterschrift

Contents

1 Introduction **5**

2 Background **7**
2.1 Printed Electronics 7
2.1.1 Printing technology 7
2.1.2 Advantages of PE 8
2.2 Artificial Neural Networks 8
2.2.1 Hierarchical structure of neural networks 9
2.2.2 Operations of artificial neural networks 9
2.2.3 Training of artificial neural networks 9
2.3 Printed Neuromorphic Circuits 10
2.3.1 Multiply accumulate (MAC) circuit 11
2.3.2 Negative weights operation 12
2.3.3 Nonlinear transformation circuit 12

3 Preliminary **13**
3.1 Gradient-based Learning 13
3.1.1 Gradient 13
3.1.2 Learning rate 15
3.1.3 Momentum update 16
3.2 Gravitation 16
3.3 Clustering 18

4 Related Work **21**
4.1 Multitask 21
4.2 Printed Neural Network 22

5 Design **23**
5.1 Gravitational Assimilation 23
5.2 Clustering 25
5.3 Trade-off 25

6 Experiment **27**
6.1 Datasets and Models 27
6.2 Hyper-parameter Configurations and Learning 30
6.3 Evaluation 30
6.3.1 Baseline 30
6.3.2 Metric 30
6.4 Results 31

6.5 Discussion 31

7 Conclusion and Future Work **37**
7.1 Conclusion 37
7.2 Future Work 37

Bibliography **39**

List of Figures

2.1 Schematic of the neural perceptron. The circle represents a perceptron. It receives multiple inputs $[x_1, x_2, x_3]^\top$ and produces an output. 9

2.2 Schematic of an artificial neural network (ANN). The input perceptrons receive external input, make a judgment, and then send signals as input to the next perceptrons until the final result is obtained. . . 10

2.3 The schematic of the 2-input crossbar (V_1, V_2) implementing the multiply accumulate operation. 11

2.4 Schematic of the proposed negative weight circuit. Two voltage dividers are deployed to shift the zero-crossing of the output voltage towards $0V$. 12

2.5 Schematic of inverter-based activation function for realizing tanh-like function 12

3.1 An example of gradient descent. The purple area has higher gradient, whereas the red area has lower gradient. 14

3.2 A cartoon depicting the effects of different learning rates. With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape. 15

3.3 TNesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position. 16

3.4 Figure of the effect of different sizes of gamma on the shape of the gravity curve. 17

3.5 Comparison of neuron positions before and after clustering 18

5.1 Figure of the effect of different sizes of lambda and gamma on the shape of the gravity curve. 25

5.2 The ideal case of gravity assimilation of multi neural networks. If the gravity has great impact of neural network. 26

5.3 The normal case of gravity assimilation of multi neural networks . . . 26

6.1 Figures of accuracy compares to different baselines 32

6.2 Figures of cluster number compares to different baselines 32

6.3 Figures of the relation between accuracy and cluster number compares to different baselines 34

6.4 This image shows the relationship between accuracy and lambda in the case of printing neuromorphic. 34

6.5 This image shows the relationship between production costs and lambda in the case of printing neuromorphic. 35

6.6 Scatter plot of normalized accuracy versus cost of printing in the case of printing neuromorphic for all the runs. 35

List of Tables

6.1 Information of 32 Datasets from UCI and Experiment Results 29

6.2 A table of some baselines. In this table, we take full model into consideration. In column "independent training", accuracy of each neural network, which are sepratelly trained, are listed. In column "without gravity" data are trained together, but without the influence of gravity. 33

6.3 Accuracy-Cost Trade-off 34

Abstract

Due to low manufacturing cost, flexibility, non-toxicity, and excellent performance, printed neuromorphic circuits have attracted increasing attention from multiple emerging areas, such as Internet of Things, wearable devices, and soft-robotics.

Regarding printing technologies, high volume manufacturing of printed circuits is usually done by the roll-to-roll printing, where the same circuits can be manufactured in large quantities with high efficiency, leading to an ultra low-cost per circuit. On the other hand, low volume manufacturing methods, such as inkjet printing, are convenient to produce unique parts in low quantity, but with relatively higher cost.

In case a large amount of different circuits need to be printed, neither high nor low volume printing technology can be deployed to guarantee high performance and low cost at the same time.

However, by deliberate design and adjustment, both manufacturing approaches can be leveraged: high volume manufacturing produces the common parts of different circuits and low volume manufacturing adjust the unique part of each circuit. In this paper, we will present a training framework that uses a special penalty term to attract the most of the weights in neural networks for different tasks close to each other or even coincident. The other advantage of this penalty is that, outlier weights, whose value is far apart from others, will not be attracted, so that the performance of circuits will not be destroyed. With this framework, multiple neuromorphic circuits are prepared for lower production costs almost without reduction in performance.

Zusammenfassung

Aufgrund der niedrigen Herstellungskosten, der Flexibilität, der Ungiftigkeit und der ausgezeichneten Leistung haben gedruckte neuromorphe Schaltkreise zunehmende Aufmerksamkeit in verschiedenen neuen Bereichen wie dem Internet der Dinge, tragbaren Geräten und Soft-Robotik auf sich gezogen.

Was die Drucktechnologien betrifft, so werden gedruckte Schaltkreise in großen Mengen in der Regel im Rolle-zu-Rolle-Verfahren hergestellt, bei dem dieselben Schaltkreise in großen Mengen mit hoher Effizienz gefertigt werden können, was zu extrem niedrigen Kosten pro Schaltkreis führt. Andererseits eignen sich Fertigungsverfahren für geringe Stückzahlen, wie der Tintenstrahldruck, für die Herstellung von Einzelteilen in geringer Stückzahl, allerdings zu relativ hohen Kosten.

Wenn eine große Anzahl unterschiedlicher Schaltkreise gedruckt werden muss, kann weder die Hoch- noch die Kleinserien-Drucktechnologie eingesetzt werden, um gleichzeitig hohe Leistung und niedrige Kosten zu gewährleisten.

Durch gezieltes Design und Anpassung können jedoch beide Fertigungsansätze genutzt werden: Bei der Großserienfertigung werden die gemeinsamen Teile verschiedener Schaltungen hergestellt, während bei der Kleinserienfertigung die einzigartigen Teile jeder Schaltung angepasst werden. In diesem Beitrag stellen wir einen Trainingsrahmen vor, bei dem ein spezieller Strafterm verwendet wird, um die meisten Gewichte in neuronalen Netzen für verschiedene Aufgaben nahe beieinander liegen oder sogar übereinstimmen zu lassen. Ein weiterer Vorteil dieser Strafe ist, dass Ausreißergewichte, deren Wert weit von den anderen abweicht, nicht angezogen werden, so dass die Leistung der Schaltkreise nicht zerstört wird. Auf diese Weise lassen sich mehrere neuromorphe Schaltkreise zu niedrigeren Produktionskosten und fast ohne Leistungseinbußen herstellen.

1. Introduction

The rapid development of soft robotics, wearables, smart consumer products, and Internet of Things (IoT) applications is greatly facilitated and benefited by the low-cost, flexible, and computational infrastructures. However, some of these requirements can not be met by silicon-based chips [34].

Specifically, silicon chips have significant strong power in computing [34], but lack of flexibility and requires extremely expensive equipments for production. In this case, printed neuromorphic circuits serves as complement parts of silicon chips, which fulfill the flexibility and low-cost. Printed neuromorphic circuits are an emerging technology. By printing well-trained resistors and transistors on substrates, desired computational processes can be implemented. By choosing appropriate materials, flexibility and even non-toxicity can be achieved. Furthermore, by applying reasonable manufacturing process, extremely low costs can be realized.

There are two main categories of manufacturing printed electronics (PE): high volume and low volume manufacturing. One representative of high-volume manufacturing is gravure printing, whereas inkjet printing is a typical Low volume printing technology. High volume manufacturing has a lower manufacturing time and cost regarding each printing circuit, but can only produce them with the same template. Low-volume manufacturing, in contrast, can meet the high customization requirements for individual circuits, however, the cost and manufacturing time are much higher than high volume production.

In most cases, the manufacturer should produce a large amount of different circuits for different clientele. I.e., despite the large batch-size, low volume production will be applied for keeping acceptable performance of different circuits, leading to high printing cost. To the other end, if the cost is the prominent factor, the manufacturer may train identical circuits to perform all tasks. However, this approach exacerbates the circuit performance.

Is there any possibility to reduce production cost while keeping acceptable circuit performance? The answer is ”yes”. We notice that, the structures of neural networks (NNs) show a great similarity (interconnection of numerous neurons). Therefore, we can speculate that different tasks could also be solved by neural networks with high

similarity, and thus, the manufacturing cost of the common parts can be greatly compressed by high-volume printing technologies, and the distinguishable parts can be done by low-volume point-of-use configuration.

The purpose of this thesis is to train multiple NNs simultaneously with different penalties to attract their weights (which correspond to the resistances) similar. With this approach, two extreme situations should be emphasized:

- With large penalty, the weights in all NNs at the same position are fused. Thus, all the circuits can be printed by high volume technology such as roll-to-roll printing.
- With small (zero) penalty, the weights in all NNs are trained independently. Thus, the weights (resistances) are hardly the same, consequently, the circuits are printed by low volume technology such as inkjet printing.

Actually, these two situations are exactly the previous solutions. With adjustment of the strength of the penalty, the aforementioned two solutions can be bridged, and the trade-off between circuit performance vs. printing costs is enabled.

The contribution of this paper is to propose a method that uses gravitational attraction as penalty term in the gradient descent to assimilate the weights during the training process. Moreover, we introduce a clustering algorithm to coincide similar weights. Weights that are farther away from the clusters will not be affected by the gravitational penalty, thus allowed to maintain their original movement trend during the training process.

2. Background

2.1 Printed Electronics

Similar to color printing, components in PE are manufactured by gradually adding materials to the substrates. Gravure printing and inkjet printing are two typical process in PE, representing high and low volume production respectively. Due to the simplicity of the production method and the cheap manufacturing equipment, printed circuits have a lower cost compared to traditional silicon-based chips. In addition, PE can be printed on a wider variety of materials. PE can not outperform traditional silicon-based chips in terms of computing efficiency and area, therefore, it serves more as a compliment of traditional silicon chips where silicon chips are not capable of doing so. In this regard, PE has a great role in emerging scenarios such as wearable devices.

2.1.1 Printing technology

Printed electronics is receiving more and more attention as an emerging electronic technology. Information electronic and optoelectronic devices and systems manufactured by printing methods have a range of advantages such as large area, flexibility, personalization, low cost, and environmental friendliness. Although the printed process itself does not have the high resolution and integration of traditional micro and nano processing, these advantages are sufficient to make printed electronics useful in many new applications. Printed electronics, also known as "print + electronics," is the use of traditional printing technology to create electronic devices or circuits. Traditional printing technology is known as the technology used to print newspapers and magazines. Newspapers or magazines are printed with black and white or color inks. If these inks, which express only color, are replaced with inks that have electronic functions, the printed graphics will have electronic functions. Similar to the multicolor overprint technology used for printing color pictures, multiple electronic function inks can be overprinted to create complex electronic function structures. Currently, in common use in a variety of electronic devices integrated circuit chip IC and printed circuit board PCB (printed circuit board) is achieved through a complex series of processing steps such as photolithography, development, etching, etc., while the print manufacturing is much simpler.

Printed manufacturing is essentially an additive manufacturing technology, as is the now familiar 3D printing. The traditional IC chip processing or circuit board manufacturing techniques commonly used in various electronic devices are all subtractive manufacturing techniques, where unwanted materials are removed by plasma etching or acid etching to form a graphic structure of functional materials. Printed additive manufacturing has these advantages.

Printing can be done on large areas and in large quantities. Conventional printing technology can already print newspaper or printed fabric on a surface several meters wide in a continuous roll-to-roll fashion at high speed, and the same method can be applied to printed electronic functional materials. While integrated circuit chip processing is currently possible for wafers up to 300 mm in diameter, printed electronics can be realized on areas over 1 m in diameter.

Inkjet printing methods in print manufacturing are characterized by digital and personalized manufacturing. Like the personalized manufacturing features of 3D printing, inkjet printed electronics do not require templates and can be rapidly manufactured in small quantities for personalized electronics.

Since printed electronics can be both mass-produced and customized, they have the advantage of being low-cost for high-volume production. In addition, based on the products obtained from high-volume production, we can also customize the manufacturing to meet specific needs for different tasks. This paper proposes a method to optimize the design of multiple printing neural networks based on the characteristics of both types of production. It enables a significant reduction in production costs with minimal impact on the working capacity of the neural network.

2.1.2 Advantages of PE

Among all advantages of PE, the most significant features are:

- Low cost: Thanks to additive manufacturing, few materials are wasted, and relatively cheap equipment is required. By selecting appropriate printing technology, production of various batch sizes can be achieved at a very small cost.
- Flexibility: By printing materials on certain substrates such as Kapton [20] and PET [4], the whole printed device can be highly flexible.
- non-toxicity: There are multiple conductive materials, that are also non-toxic or even bio-compatible [18, 1], therefore, they can perfectly employed in medical or wearable applications [39].

Due to these remarkable properties, PE draws increasing attention from both academia and industry.

2.2 Artificial Neural Networks

Artificial neural network (ANN), often called neural network (NN), are computational systems inspired by the biological neural networks that make up the human brain. In 1958, American psychologist Frank Rosenblatt conceived and attempted to build a machine that would respond like a human mind. He named his machine "perceptron" [24] (Fig. 2.1). For all practical purposes, the ANN learns by example in a human-like manner, in which external inputs are received, processed, and manipulated in the same way as the human brain.

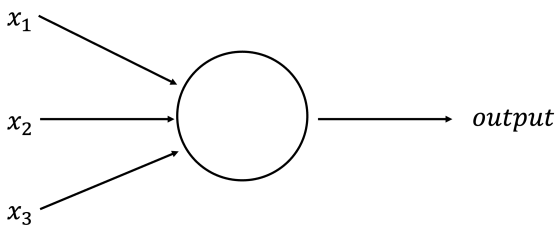


Figure 2.1: Schematic of the neural perceptron. The circle represents a perceptron. It receives multiple inputs $[x_1, x_2, x_3]^\top$ and produces an output.

2.2.1 Hierarchical structure of neural networks

Different parts of the human brain are used to process various kinds of information [26]. These parts of the brain are arranged in a hierarchy. When information enters the brain, each layer or level of neurons does its particular job, which is to process the incoming information, gain insight, and then pass them on to the next higher-level layer. Generally, ANNs have three types of neuron layers: input layer, hidden layers, and output layer. Information flows from one layer of neurons to another, just as it does in the human brain:

- Input layer: the entry point of data into the system.
- Hidden layer: the location where the information is processed.
- Output layer: the location where the system decides how to proceed based on the data

The more complex the ANNs are, the more hidden layers they will have. ANN operates through collections of nodes or connecting units. These nodes loosely mimic the network of neurons in human brains. Just like in humans, artificial neurons receive signals, process them, and send signals to other neurons connected to them.

2.2.2 Operations of artificial neural networks

In ANNs, neurons receive a stimulus in the form of a real number of signals. Then the output of each neuron is calculated by the sum of its inputs followed by a nonlinear transformation (activation function). The connections between neurons are called edges. Each edge has its own weight. This parameter is adjusted and changed by gradient descent in training process. The weights indicate the strength of the signal at the connection. Once a signal is put into a ANN, it will be transferred from the input layer via hidden layers to the output layer in the manner discussed above.

2.2.3 Training of artificial neural networks

As mentioned in the previous subsection, the output of an ANN is actually the input signal, operated by a series of weighted-sums and nonlinear transformations. To find appropriate weights that can successfully transform the input to the desired output is referred to as training of the ANN.

It is known that, machine learning, especially ANN, is a data-driven approach, whose training requires a large amount of labeled data. These data will be entered to the

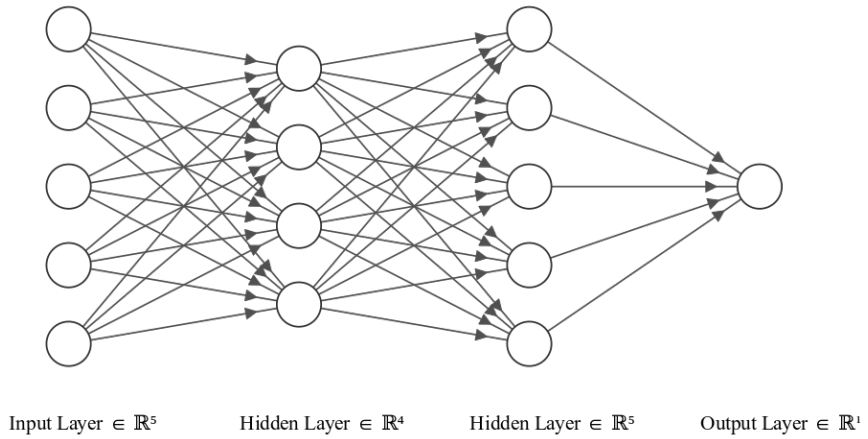


Figure 2.2: Schematic of an ANN. The input perceptrons receive external input, make a judgment, and then send signals as input to the next perceptrons until the final result is obtained.

neural network from the input layer, and the results will be compared with their corresponding labels. The objective of training is to minimize the difference between network output and the target output. To this end, a differentiable loss function is usually introduced as the objective function to guide the training towards desired direction. Since both operations in the network and the loss functions are differentiable, the partial derivative of the objective function over learnable parameters (gradients) can be easily calculated by "back propagation".

In each epoch, all data are used to calculate the loss through forward pass (input layer - hidden layers - output layer), and then, the gradients of are calculated by back propagation. With these gradients, the learnable parameters (usually the weights) can be optimized easily through gradient-based optimization algorithm, e.g., gradient descent, Adam, etc.

2.3 Printed Neuromorphic Circuits

Neuromorphic circuits is an emerging domain. It refers to implementing the same functionality of ANNs by electronic components. The neuromorphic circuits fabricated by printed electronics are called printed neuromorphic circuits. Weller et al. proposed several basic circuit structures by which a printable ANN can be obtained by a combination of these circuits [34].

Different from ANNs in silico, printed neuromorphic circuits are limited by some physical constraints, for example, the resistance must be positive and it suffers from fabrication errors. To this end, Zhao et al. [38] and Weller et al. [34] has proposed multiple solutions. Since this part is beyond the scope of this paper, we do not dicuss this further.

In next sections, we will introduce the important subcircuits in the printed neuromorphic circuits.

2.3.1 Multiply accumulate (MAC) circuit

Multiply accumulate (MAC) operations are part of the computation of each neuron in an ANN. It adds up the inputs, which are scaled and weighted by the weights w_i. The output of the multiply accumulate can be computed by:

$$a = x^\top w = \sum_i w_i x_i + b,$$

where x_i is the i-th input, w_i denote the corresponding weight, and b is the bias.

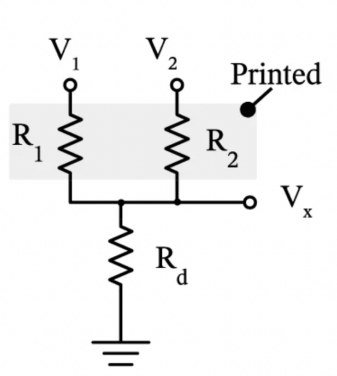


Figure 2.3: The schematic of the 2-input crossbar (V_1, V_2) implementing the multiply accumulate operation.

In the printed neuromorphic circuit, MAC is implemented by a crossbar circuit, which consists of multiple printed resistors R_i. According to Ohm's law, the current on the resistor R_i is $I_i = (V_i - V_x)/R_i$. Then, the currents are summed up based on Kirchhoff's rule:

$$\sum_{i=1} I_i + I_d = 0, \tag{2.1}$$

i.e.,

$$V_x = \frac{\sum_i \frac{V_i}{R_i}}{\sum_i \frac{1}{R_i} + \frac{1}{R_b} + \frac{1}{R_d}}. \tag{2.2}$$

By reformulating R_i as $R_i = 1/g_i$ (g_i refers to the conductance), the formula can be abbreviated to:

$$V_x = \frac{\sum_i g_i V_i}{\sum_j g_j + g_d} =: \frac{\sum_i g_i V_i}{G} = \sum_i \frac{g_i}{G} V_i$$

If we express the weights by

$$w_i = \sum_i \frac{g_i}{G}.$$

The behavior of the crossbar becomes evident:

$$V_x = \sum_i w_i V_i.$$

This is exactly the weighted-sum operation in ANNs. Note that, the bias term can be extended by an additional resistor with a constant input voltage.

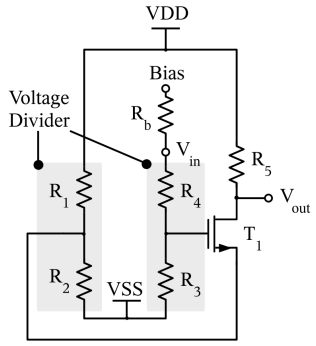


Figure 2.4: Schematic of the proposed negative weight circuit. Two voltage dividers are deployed to shift the zero-crossing of the output voltage towards $0V$.

2.3.2 Negative weights operation

In each printed neuron, the MAC operation can implement the computation of the input signal vector x and the weight w. However, since the weights in MAC circuit is composed by resistances, they must be positive, which would limit its applicability to potential classification or regression problems.

To implement negative weights, Weller et al. [34] proposed an inverter-based transfer circuit (Fig. 2.4), which is capable of converting positive neuron input voltages to negative voltages, in this way, the negative relationship can be realized. I.e., if we need a negative weight

$$-|w_i| \cdot V_i,$$

instead of printing a negative w_i, we transform the input voltage V_i into a negative one, i.e.,

$$|w_i'| \cdot \mathrm{neg}(V_i).$$

Note that, the function $\mathrm{neg}(\cdot)$ is nonlinear, therefore, the w_i will be adjusted to w_i', which keep the absolute value of $|w_i| \cdot V_i$ and $|w_i'| \cdot \mathrm{neg}(V_i)$ are the same.

2.3.3 Nonlinear transformation circuit

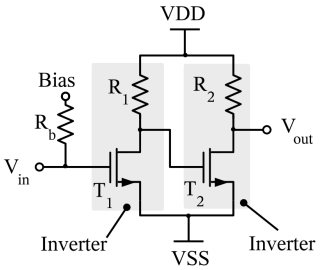


Figure 2.5: Schematic of inverter-based activation function for realizing tanh-like function

Each neuron in printed neuromorphic circuits requires a nonlinear activation function in addition to the two components mentioned above. The activation function is located directly after the MAC operation. Weller et al. proposed ptanh based on printed electronic circuits (Fig. 2.5), which uses two transistors to simulate the tanh-like activation function.

3. Preliminary

In this chapter, we introduce some knowledge employed in this thesis, and several related works. Firstly, gradient-based learning is introduced, as it is the most common method for training neural networks. Besides, gravitation is used to attract the corresponding weights in different networks close to each other. Subsequently, we introduce DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [10], a clustering algorithm to coincide similar weights to exactly same values.

3.1 Gradient-based Learning

3.1.1 Gradient

Gradient descent is a famous parameter optimization algorithm that is widely used to minimize a certain differentiable objective function. Although it does not guarantee a global optimum in nonconvex situation, it is still the easiest and the most popular way in machine learning community. For a certain model, e.g.,

$$h(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b = w^TX + b.$$

We exemplarily define a loss function (objective function):

$$J(w) = \frac{\sum_i^n (h(x^{(i)}) - y^{(i)})^2}{2},$$

where $x^{(i)}$ and $y^{(i)}$ is the input and target output of the i-th instance, while w is the optimization variable.

The goal of optimization is to make $J(w)$ reach the minimum value. The gradient descent process functions as follows:

- First assign a initial value to the optimization variable w. This value is usually randomly selected.
- Change the value of w so that $J(w)$ decreases in the direction of negative gradient.

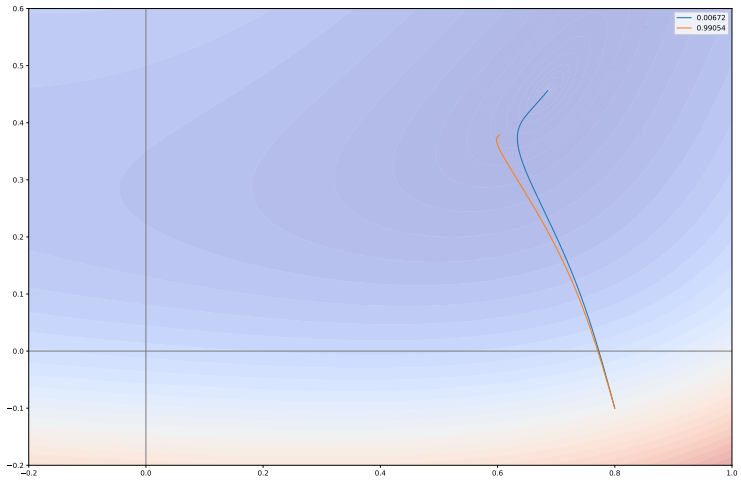


Figure 3.1: An example of gradient descent. The purple area has higher gradient, whereas the red area has lower gradient.

As shown in Fig. 3.1, each gradient changes as follows:

$$w = w - \alpha \frac{\partial J(w)}{\partial w},$$

where α is the learning rate, which is a positive to determine the step size.

The commonly used gradient descent method also specifically contains three different forms, each of which also has different advantages and disadvantages.

- Batch Gradient Descent (BGD)
  BGD is the most primitive form of gradient descent method. The specific idea is to use all training samples to update each parameter.
  Advantage: optimal solution with relatively small number of iterations.
  Disadvantage: slow training process with large datasets.

- Stochastic Gradient Descent (SGD)
  SGD iterates through each sample to update once, if the sample size is large (e.g., more than 100,000), then only a few tens of thousands or thousands of samples may be used to iterate to the optimal solution, compared with the above BGD iteration requires more than 100,000 training samples, one iteration is not optimal. If we iterate 10 times, we need to iterate through the training samples 10 times. However, one of the problems associated with SGD is that there is more noise than BGD, so that SGD does not always iterate in the direction of overall optimality. This optimization algorithm, which uses only a single sample at a time, is sometimes called an on-line algorithm.
  Advantages: fast training.
  Disadvantages: worse solution.

- Mini-batch Gradient Descent (MBGD)
  The MBGD method takes a compromise between the above two approaches

by using batch size samples for each parameter update, which ensures that the training process of the algorithm is relatively fast and also guarantees the accuracy of the final parameter training.

Most deep learning algorithms are based on small-batch gradient descent algorithms for solving, and they are now often referred to simply as stochastic gradient descent methods.

3.1.2 Learning rate

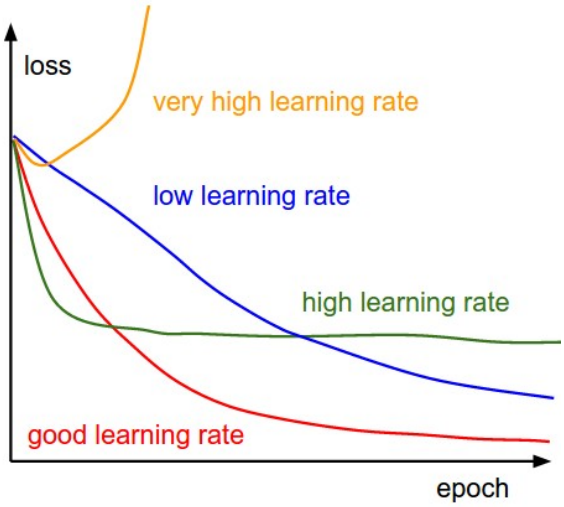


Figure 3.2: A cartoon depicting the effects of different learning rates. With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape.

The learning rate α determines how fast the parameters move to the optimal value. If the learning rate is too large, it is likely to go beyond the optimal value and overshoot, i.e., the extreme point at both ends of the divergence, or violent oscillation, in short, as the number of iterations increases the loss does not reduce the trend; on the contrary, if the learning rate is too small, the efficiency of optimization may be too low, the algorithm can not converge for a long time, can not quickly find a good direction of descent, as the number of iterations increases the loss is basically unchanged, so The learning rate is crucial to the performance of the algorithm.

Therefore, to make the gradient descent method have a good performance, we need to set the value of the learning rate in a suitable range. The following two types of learning rate adjustments are commonly used:

- Manual tuning based on experience. By trying different fixed learning rates, such as 0.1, 0.01, 0.001, etc., we observe the relationship between the number of iterations and the change of loss, and find the learning rate corresponding to the fastest decreasing loss relationship.
- Adaptive learning rate algorithms. E.g., AdaGrad, Root Mean Squared Propagation (RMSprop), Adam, etc.

3.1.3 Momentum update

While stochastic gradient descent remains a very popular optimization method, its learning process can sometimes be slow. Momentum update methods are designed to speed up learning, especially when dealing with high curvature, small but consistent gradients, or gradients with noise. The momentum algorithm accumulates a moving average of the exponential decay of previous gradients and continues to move in that direction. The effect of momentum is shown in the Fig. 3.3[1].

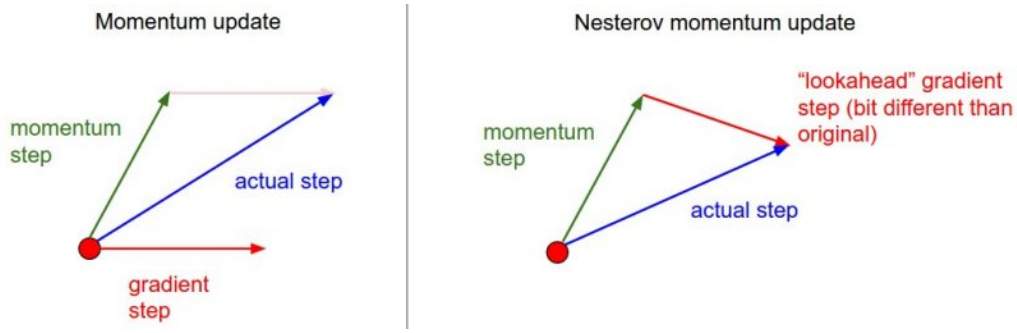


Figure 3.3: TNesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

3.2 Gravitation

The law of gravitation is a general physical law derived from empirical observations that Isaac Newton called inductive reasoning [31]. It is a part of classical mechanics and was first published in 1687 in Philosophie Naturalis Principia Mathematica. The law states that each point of mass attracts the mass of every other point by pointing to a force along the line of intersection of the two points. The force is proportional to the product of the two masses and inversely proportional to the square of the distance between them.

The equation for universal gravitation thus takes the form:

$$F = \frac{Gm_1m_2}{r^2}, \tag{3.1}$$

where F is the gravitational force received by the two masses, G is the gravitational constant, m_1, m_2 are the masses of the objects, and r is the distance between the centers of their masses.

Suppose the gravitational acceleration of the mass is a , and according to Newton's second law:

$$F = m_1 \cdot a_1,$$

that is,

[1] CS231n: Convolutional Neural Networks for Visual Recognition https://cs231n.github.io

$$a_1 = \frac{F}{m_1}. \tag{3.2}$$

Bringing this expression (3.2) into Newton's equation of gravity 3.1, we get:

$$a = G\frac{m_2}{r^2}.$$

In this paper, we will train several neural networks applied to different datasets at the same time. In order to make the trained neural networks have the most similar structure, i.e., to make the parameters of neurons at corresponding positions of different neural networks as identical as possible. For n neurons at the corresponding positions of n neural networks, we can use gravity to make them actively close during the training process.

The weight w of each neuron is used as its position and the weight difference between every two neurons is used as their distance. To simplify the algorithm and to reduce the computational effort, we consider the mass of all neurons as $m_i = 1, \forall i$.

This yields, for two neurons, their mutual gravitational acceleration is:

$$a = \frac{1}{(w_1 - w_2)^2}.$$

To avoid pathological problems caused by the overlap of two neurons, we normalize the gravitational acceleration by:

$$a = \frac{\gamma}{(w_1 - w_2)^2 + \gamma}.$$

Here, γ modifies the shape of gravity function, see Fig. 3.4.

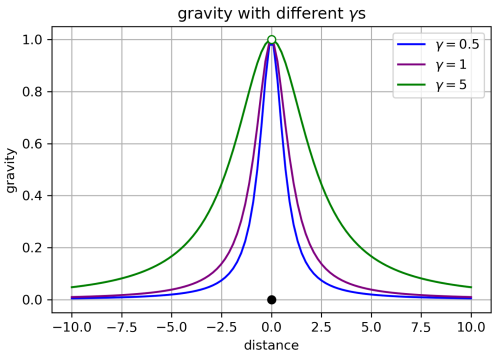
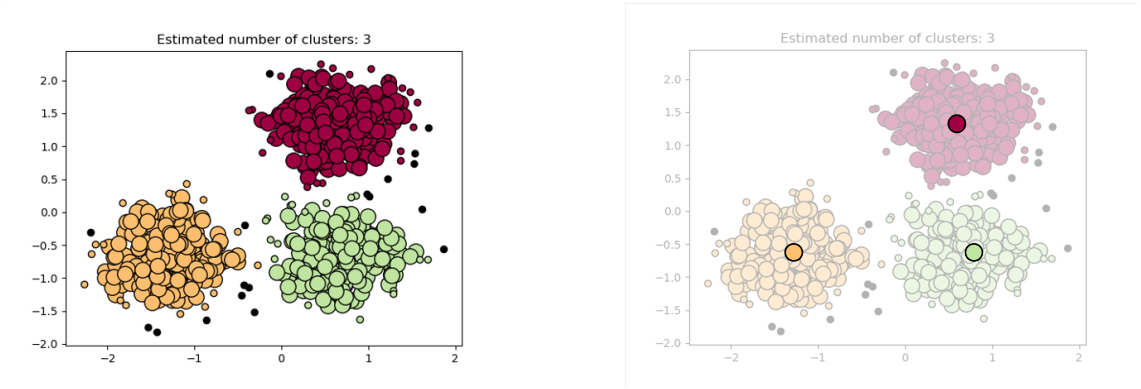


Figure 3.4: Figure of the effect of different sizes of gamma on the shape of the gravity curve.



(a) An example of neuron position before clustering.

(b) An example of neuron position after clustering.

Figure 3.5: Comparison of neuron positions before and after clustering

3.3 Clustering

Clustering is a technique for finding the intrinsic structure between data. Clustering organizes all data instances into groups of similarities, which are called clusters. Data instances in the same cluster are the same as each other, and instances in different clusters are different from each other. Clustering techniques are often referred to as unsupervised learning, unlike supervised learning, where there is no categorization or grouping information that indicates the class of data in a cluster. The similarity between data is discriminated by defining a distance or similarity coefficient(Fig. 3.5).

In this thesis, we use the DBSCAN clustering, one of the most popular clustering algorithm, to cluster the neurons in different neural networks. For several different neural networks, the neurons in them should have different weights. To make these neural networks as similar as possible, the DBSCAN clustering algorithm analyzes these weights. Several neurons with similar weights are grouped together. In a later operation, the neurons in the same group are fused so that they can be manufactured using a high-volume manufacturing method. The purpose of the clustering algorithm is to find some neurons that can be grouped together.

DBSCAN is a density-based clustering algorithm, which defines a cluster as the largest set of densely connected points and is able to classify regions with sufficient density into clusters, and can find arbitrary clusters of arbitrary shape in a spatial database of noise.

DBSCAN is based on a set of neighborhoods to describe the closeness of the sample set, and the parameters (ϵ, MinPts) are used to describe the closeness of the sample distribution in the neighborhood. Where ϵ describes the neighborhood distance threshold and MinPts describes the minimum number of data points in a neighborhood with distance ϵ.

- A point p is a core point if at least minPts points are within distance ϵ of it (including p).
- A point q is directly reachable from p if point q is within distance ϵ from core point p. Points are only said to be directly reachable from core points.

- A point q is reachable from p if there is a path $p_1, ..., p_n$ with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i. Note that this implies that the initial point and all points on the path must be core points, with the possible exception of q.
- All points not reachable from any other point are outliers or noise points.

The exact algorithm is expressed as follows:

Algorithm 1 DBSCAN(D, ϵ, MinPts)

```
C=0
for all unvisited point P in dataset D do
  mark P as visited
  N = D.regionQuery(P, ε)
  if sizeof(N) < MinPts then
    mark P as NOISE
  else
    C = next cluster
    expandCluster(P, N, C, ε, MinPts)
  end if
end for
```

Algorithm 2 expandCluster(P, N, C, ϵ, MinPts)

```
add P to cluster C
for all point P' in N do
  if P' is not visited then
    mark P' as visited
    N' = D.regionQuery(P', ε)
    if sizeof(N') >= MinPts then
      N = N joined with N'
    end if
  end if
  if P' is not yet member of any cluster then
    add P' to cluster C
    unmark P' as NOISE if necessary
  end if
end for
```

Algorithm 3 regionQuery(P, ϵ)

```
return all points within P's ε-neighborhood (including P)
```

4. Related Work

Motivation revisit

Our goal is to increase the neural network similarity while processing multiple neural networks at the same time. This allows to reduce the average production cost by using high volume manufacturing as much as possible. Therefore, we will introduce the related work about multitasks.

Moreover, since the work is based on printed neuromorphic circuits, we will also state some related work about printed neural network (pNN).

4.1 Multitask

Multitask learning (MTL) aims at solving the related tasks simultaneously by exploiting shared knowledge to improve performance on individual tasks. MTL has led to successes in many applications of machine learning, from natural language processing and speech recognition to computer vision and drug discovery. In [30], Ruder gives a general overview of MTL, particularly in deep neural networks. It introduces the two most common methods for MTL in Deep Learning, which is hard parameter sharing and soft parameter sharing. Hard parameter sharing is generally applied by sharing the hidden layers between all tasks, while keeping several task-specific output layers. In soft parameter sharing, on the other hand, each task has its own model with its own parameters. Our paper is very similar to soft parameter sharing, where each layer of each neural network can, but does not have to, share their weights, thereby increasing the number of shared neurons and reducing unnecessary accuracy degradation due to shared weights. In [41] Zhou, Fan, et al. revisited the adversarial multitask neural network and proposed a new training algorithm to learn the task relation coefficients and neural network parameters automatically. In [6], Chou, Yi-Min, et al. propose a novel method to merge convolutional neural-nets for the inference stage. Given two well-trained networks that may have different architectures that handle different tasks, this method aligns the layers of the original networks and merges them into a unified model by sharing the representative codes of weights.

4.2 Printed Neural Network

Weller, Dennis D., et al. demonstrate in [34] printed hardware building blocks such as inverter-based comprehensive weight representation and resistive crossbars as well as printed transistor-based activation functions. In addition, to find the optimal component values in printed neuromorphic circuits, they also proposed the printed neural network, which is a NN-based simulation model of the printed neuromorphic circuits. In printed neural networks, the learnable parameters are exactly the component values of the printed neuromorphic circuits.

pNNs are also capable to respect hardware constraints, such as printing technology. Given a printable conductance range $\{0\} \cup [g_{min}, g_{max}]$, a straight through gradient estimator can be employed to solve this problem [38]. It can also simulate the variation of component values caused by aging problem [38] and printing errors [34].

Regarding split manufacturing, we have only found a single work [40], which split the conductances as the sum of a common part (which is shared among all networks) and an individual part (which is the point-of-use of each individual task). After training, the common part of all circuits will be printed by high volume process, while the individual configuration will be done by low volume printing. By encoring a larger common part and smaller individual parts, the cost for low volume production can be saved.

5. Design

In this chapter, we will specifically describe how the attraction of gravity allows different neural networks to achieve maximum similarity. First, the role of gravity and the implementation method are introduced. Next, we will describe in detail the cluster algorithm in this thesis. Finally, we use trade-off to balance the degree of similarity versus the performance of each neural network.

5.1 Gravitational Assimilation

Our purpose is to reduce the manufacturing cost and time using a high-volume manufacturing approach when given multiple neural networks. Therefore, we can make neurons close together during the training process. However, we only want to fuse neurons that are close together. Forcing fusion on distant neurons would make the neural network much less accurate. This feature of greater influence for near neurons and less influence for distant neurons is similar to gravity. Thus, gravity is a particularly good choice in this paper.

Since weights at the same position among different neural networks can be correlated with each other by gravity, we want to achieve the goal of neural network assimilation by gravity. Therefore, if we wish to change the motion trend of neuron weights through gravity, we need to associate the gravity with the gradient of neuron weights. This is done as follows:

We achieve the goal of neural network training while assimilation by changing the specific details of the gradient descent mechanism. The ordinary neural network uses a loss function to calculate the deviation of the current neural network prediction from the actual value. Then the gradient of each weight is calculated to determine the direction of gradient descent.

We assume the cost function

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (f(x^{(i)} - y^{(i)})^2,$$

where

$$f(x) = w_0x_0 + w_1x_1 + \cdots + w_nx_n = \sum_{j=0}^{n} w_jx_j.$$

Our goal is:

$$\min_w J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)} - y^{(i)})^2.$$

The gradient descent method is expressed by the formula:

$$w_j = w_j - \alpha\frac{\partial J(w)}{\partial w_j}.$$

In the case of all neurons with mass = 1, ignore the gravitational constant, a neuron alone receives the force of another neuron as

$$F = \text{sign}(w_1 - w_2) \cdot \frac{1}{(w_1 - w_2)^2}.$$

In the case of two neurons with very close weights, the force obtained will tend to infinity, to avoid this situation, we add an identical γ to the numerator denominator:

$$F = \text{sign}(w_1 - w_2) \cdot \frac{\gamma}{(w_1 - w_2)^2 + \gamma}.$$

In all neural networks, the total force it receives, shall be the gravitational force of other neurons on it only and

$$F_{sum} = \sum F_i = \sum \text{sign}(w_1 - w_i) \cdot \frac{\gamma}{(w_1 - w_i)^2 + \gamma}.$$

Since its mass equals 1, its acceleration is numerically the same as the combined force:

$$a_{\text{sum}} = \sum_i \text{sign}(w_1 - w_i) \cdot \frac{\gamma}{(w_1 - w_i)^2 + \gamma}.$$

We take into the gravity of neurons account on this basis as well. From this, the gradient change caused by gravity is added to the gradient after back-propagation to obtain the direction of the final gradient descent.

The gradient descent after the influence of gravity is:

$$\begin{aligned} w_j =& w_j - \alpha\left(\frac{\partial J(w)}{\partial w_j} + \lambda a_{\text{sum}}\right) \\ =& w_j - \alpha\left(\frac{\partial J(w)}{\partial w_j} + \lambda\sum_i \text{sign}(w_1 - w_i) \cdot \frac{\gamma}{(w_1 - w_i)^2 + \gamma}\right), \end{aligned}$$

where λ is a new hyper-parameter to regulate the degree of gravitational influence on the gradient descent.

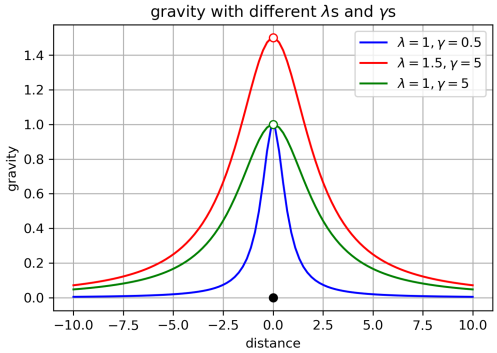


Figure 5.1: Figure of the effect of different sizes of lambda and gamma on the shape of the gravity curve.

5.2 Clustering

Under the influence of gravity, the weights of neurons at the corresponding position of different neural networks can become very close, but it is difficult to make the weights of neurons overlap exactly by gravity alone. Therefore, we use a clustering algorithm after each weight update, and we dynamically group the neurons in the corresponding positions of different neural networks. The weights within the same group are redistributed as the average of the weights within the group.

Therefore the selection of a suitable clustering algorithm is crucial to the effectiveness of the experiment. There are many clustering algorithms, such as the classical k-mean algorithm, but such clustering requires an artificially given number of groups, which is obviously not possible in this thesis. Since the weights of neurons in different neural networks need to be grouped dynamically, we choose the clustering algorithm DBSCAN, which does not require a predetermined number of groups. DBSCAN takes two parameters: ϵ and the minimum number of points (minPts) are needed to form a high-density region, which starts with an arbitrary unvisited point and then explores the ϵ-neighborhood of that point. If there are enough points in the neighborhood, a new cluster is created, otherwise the point is labeled as a clutter. If a point is in the dense region of a cluster, the points in its ϵ-neighborhood also belong to that cluster, and when these new points are added to the cluster, if it is also in the dense region, the points in its ϵ-neighborhood will also be added to the cluster. This process will be repeated until no more points can be added, so that a density-linked cluster is found in its entirety. Then, an unvisited point will be explored to find a new cluster or clutter.

In this paper, we use the DBSCAN algorithm from the scikit-learn [25] package. where ϵ equals to the hyper-parameter cluster threshold and minPts uses the default value of 5.

5.3 Trade-off

We need to recognize that neuron attraction varies for different sizes of λ. When the λ is very large, the gravity term will dominate the update direction, in this way, all neurons will converge to the same value, i.e., the weights among all neural networks at the same position will have the same weights. In this case, the cluster

number becomes small, but as a side effect, the accuracy decreases significantly. For smaller λ, gravity has limited effect on the update direction of weights, therefore, fewer neurons are assimilated, but for each dataset, their respective accuracies will be close to those of independently trained neural networks.

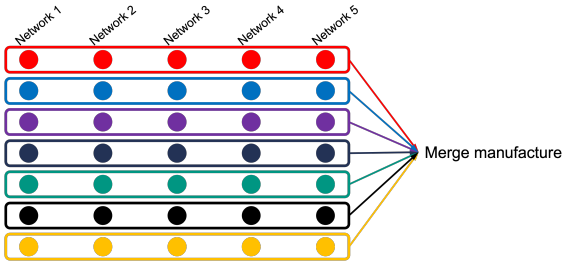


Figure 5.2: The ideal case of gravity assimilation of multi neural networks. If the gravity has great impact of neural network.

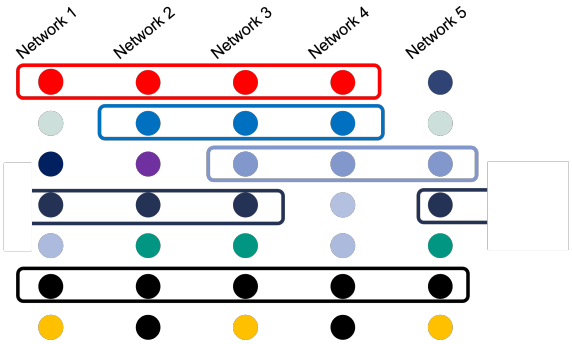


Figure 5.3: The normal case of gravity assimilation of multi neural networks

6. Experiment

6.1 Datasets and Models

To evaluate the effectiveness of the theory proposed in this paper, we conducted experiments using 32 classification datasets, that are suitable for PE and PNC context. These datasets were obtained from the UCI machine learning database [8]. The names and sources of datasets are:

- acuteinflammation [7]
- acutenephritis [7]
- balancescale [19]
- blood [37]
- breastcancer [35]
- breastcancerwisc [32]
- breasttissue [17]
- ecoli [15]
- energyy1 [33]
- energyy2 [33]
- fertility [12]
- glass [22]
- habermansurvival [13]
- hayesroth [14]
- ilpdindianliver [28]

- iris [11]
- mammographic [9]
- monks1 [16]
- monks2 [16]
- monks2 [16]
- monks3 [16]
- pima [3]
- pittsburgbridgesMATERIAL [29]
- pittsburgbridgesRELL [29]
- pittsburgbridgesSPAN [29]
- pittsburgbridgesTORD [29]
- pittsburgbridgesTYPE [29]
- postoperative [36]
- seeds [5]
- teaching [21]
- tictactoe [23]
- vertebralcolumn2clases [2]
- vertebralcolumn3clases [2]

The criterion of dataset filtering are: 1) The number of features, and the number of classes in these datasets are less than 10, as the scenario of printed neuromorphic circuits are generally simple. 2) The data number is less than 1000, as they are not too much of a burden on the hardware for training.

For all datasets, 60% of the data is used for training, 20% for validation, and the remaining 20% for testing.

We implement pNN in PyTorch. In pNN, each neural network consists of multiple printed layers. Each printed layer consists of MACs and activation functions. Unlike normal neural networks: the learnable parameters in printed layers are resistance values instead of weights w.

Table 6.1: Information of 32 Datasets from UCI and Experiment Results

Dataset	**#features**	**#classes**	**#data**	**random guess**
acute-inflammation	6	2	120	0.3200
acute-nephritis	6	2	120	0.6400
balance-scale	4	3	625	0.4206
blood	4	2	748	0.7467
breast-cancer	9	2	286	0.7241
breast-cancer-wisc	9	2	699	0.6929
breast-tissue	9	6	106	0.1818
ecoli	7	8	336	0.4265
energy-y1	8	3	768	0.5000
energy-y2	8	3	768	0.5260
fertility	9	2	100	0.8571
glass	9	6	214	0.3256
haberman-survival	3	2	306	0.7903
hayes-roth	3	3	132	0.3438
ilpd-indian-liver	9	2	583	0.6838
iris	4	3	150	0.3226
mammographic	5	2	961	0.5389
monks-1	6	2	124	0.5045
monks-2	6	2	169	0.6167
monks-3	6	2	122	0.4909
pima	8	2	768	0.6429
pittsburg-bridges-MATERIAL	7	3	106	0.8636
pittsburg-bridges-REL-L	7	3	103	0.1905
pittsburg-bridges-SPAN	7	3	92	0.5000
pittsburg-bridges-T-OR-D	7	2	102	0.8000
pittsburg-bridges-TYPE	7	6	105	0.6364
post-operative	8	3	90	0.6316
seeds	7	3	210	0.2093
teaching	5	3	151	0.2581
tic-tac-toe	9	2	958	0.6302
vertebral-column-2clases	6	2	310	0.6349
vertebral-column-3clases	6	3	310	0.4762

6.2 Hyper-parameter Configurations and Learning

In this paper, there are three hyper-parameters: λ, γ, and cluster threshold ϵ. Among them, λ controls the effect of gravitational force on neurons, γ controls the relationship between the distance of neurons and the gravity, ϵ controls the clustering process of neurons in different neural networks at corresponding positions.

To find good networks for each task, a grid-search over the λ, γ and ϵ is performed. All networks are trained for maximal 15,000 epochs. In addition, we set up three different model architectures: semi, hidden, and full. The semi model has the architecture #inputs-3-max #outputs, where #inputs is the number of features in each dataset corresponding to each neural network, and max #outputs represents the number of maximum classes in all neural networks. This means that in the semi model, the input layer is not shared by all neural networks. The hidden layer is shared with the output layer. The full model has the architecture max #inputs-3-max #outputs, which means that the weights can be shared by the corresponding neurons of all neural networks. The last model is the hidden model, which has an architecture of #inputs-3-3-# outputs, where each neural network has a different input and output layer, but the hidden layer can be shared.

Training runs are canceled if the training did not improve over 500 consecutive updates(see early stopping [27]). In the experiments, the λ, γ, ϵ are chosen logarithmically with base equals 2 as $\lambda \in [2^{-16}, 2^{8}]$ with 50 points, $\gamma \in [2^{-9}, 2^{-3}]$ with 8 points, $\epsilon \in [2^{-10}, 2^{-8}]$ with 5 points. Additionally, all experiments are run with 5 different random seeds leading to a total of $3 \times 50 \times 8 \times 5 \times 5 = 30000$ configurations for each dataset.

6.3 Evaluation

6.3.1 Baseline

In this thesis, we have several baselines to evaluate the performance of the proposed training framework.

The first baseline is "independent training", which means that all the networks are trained independently, this is equivalent to "no gravity", i.e., $\lambda = 0$. Since no coupling exists in training, the accuracies of each task can be seen as the upper bound of each task.

To the other end, we select "full gravity" as the second baseline, meaning that, we use the identical network to perform all tasks. This is equivalent to train networks with very large gravity strength. This can be seen as the worst performance of the networks, but with the lowest production cost.

6.3.2 Metric

As evaluation metric, we do not only consider the accuracy, but also the number of clusters. Accuracy is used to evaluate the suitability of the neural network after assimilation for each dataset, while the number of clusters indicates how well the neural networks are assimilated. A higher number of clusters indicates a worse degree of assimilation after the experiment. A lower number of clusters indicates a better

degree of assimilation. Similarly, for a specific dataset, if the accuracy changes too much before and after assimilation, it indicates that the assimilation process has too much influence on this dataset and the assimilation is not effective. Also, the same assimilation process may have different effects on different datasets.

Regarding the accuracy, since we trained multiple networks at the same time, the resulted accuracies are hardly comparable to each other. Therefore, we calculated the average value of the accuracies among all the tasks. In this way, the scalar result can be easily compared to the results from other experiment setups. Moreover, we normalize the accuracies by their upper bound, i.e., the "independent training". This can eliminate the difficulties of each task.

The number of clusters is a good estimation to the printing cost, as the cost of printing templates is usually the number of different patterns, i.e., the number of different resistance in our case.

In the table 6.2, we list the accuracy that can be achieved for each dataset in several baseline cases:

6.4 Results

After training, we test the performance of each dataset with test data. In the figures 6.1, 6.2, 6.3, we show the trend of accuracy and cluster number with gravity, and the relationship between different cluster numbers and accuracy. In addition, we also show the results of the full model with simulated pNN.

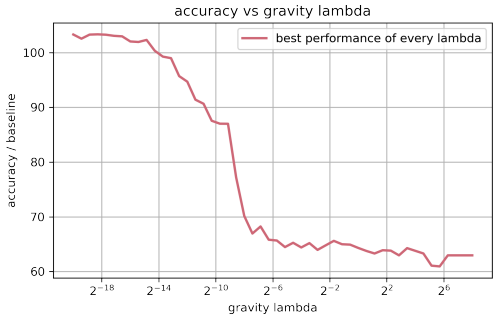
6.5 Discussion

From the Fig. 6.1, we can see, that with the influence of weak gravity, the best performance is always better than the baselines. With the increase of hyperparameter lambda, neural network accuracy decreases rapidly, until lambda reaches 2^{-5}. After this point, the change in accuracy flattens and converges to approximately 63% on average compares to baseline split training and 0 gravity.

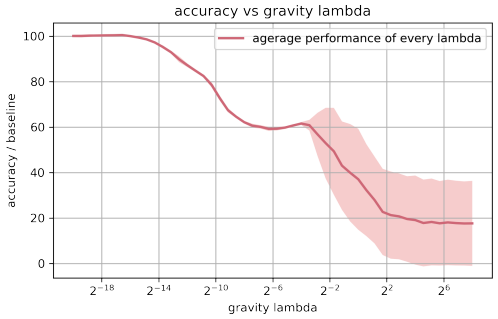
Regarding average performance, with the influence of feeble gravity, the neural networks have approximately the same accuracy as baselines. With the increase in lambda, the accuracy slides gently down and pauses at 60%. When lambda is larger than 2^{-4}, the accuracy decreases and the variance also becomes larger.

As shown in the Fig. 6.2, in terms of neural network fusion, this framework shows very good results. When compared with split training, even with a small gravitational force, the average cluster reached about 1200, and the smallest network even required only 900 clusters. Compared with the 0 gravity baseline, only 60% of the cluster is required. When the lambda is 2^{-8}, the cluster reaches the lowest point. At this time, the average cluster is less than 100, which only needs to correspond to 5% of the 0 gravity baseline.

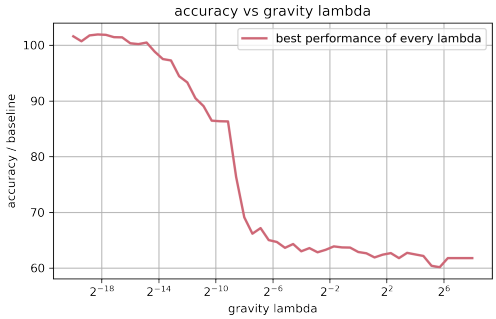
In the Fig. 6.3, we plot the relationship between accuracy and the number of clusters. It can be seen that when the number of clusters is small, a small increase in the number of clusters can achieve a great improvement in accuracy. Immediately afterward, the improvement of accuracy becomes slow. The effect of the cluster number



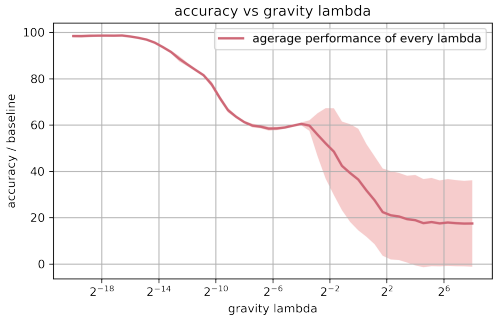
(a) best accuracy compared with split training



(b) mean accuracy compared with split training

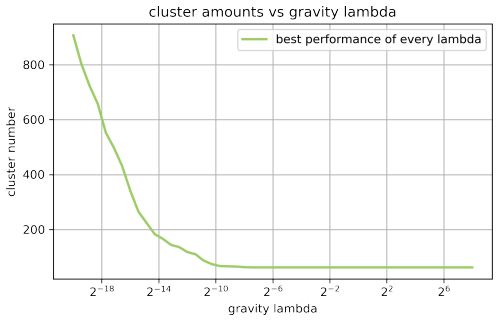


(c) best accuracy compared with 0 gravity

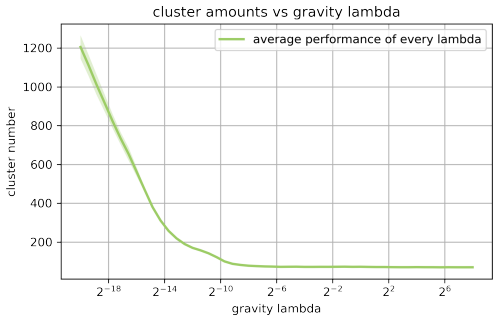


(d) mean accuracy compared with 0 gravity

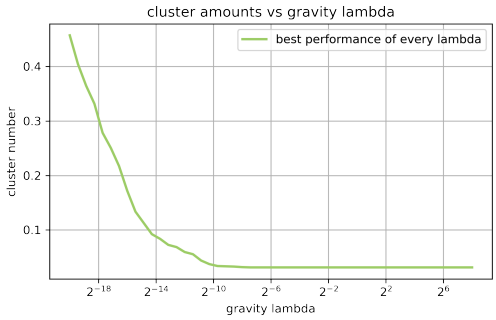
Figure 6.1: Figures of accuracy compares to different baselines



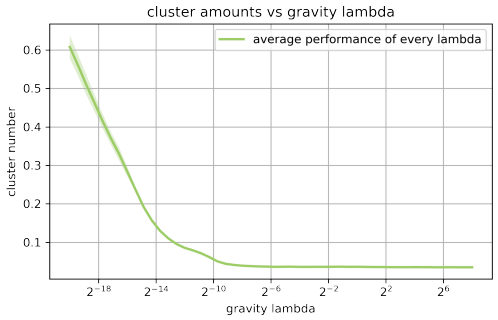
(a) best accuracy compared with split training



(b) mean accuracy compared with split training



(c) best accuracy compared with 0 gravity



(d) mean accuracy compared with 0 gravity

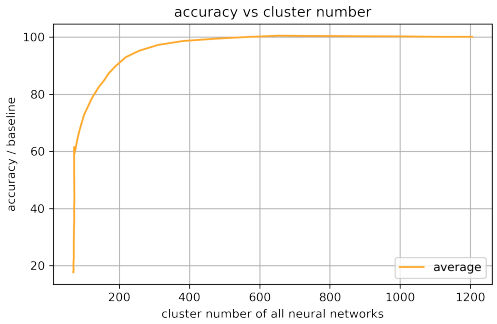
Figure 6.2: Figures of cluster number compares to different baselines

Some baseline results of 32 datasets (take full model as example)		
Dataset	independent training	without gravity
acuteinflammation	1.000	0.920
acutenephritis	1.000	1.000
balancescale	0.897	0.873
blood	0.773	0.767
breastcancer	0.681	0.724
breastcancerwisc	0.957	0.971
breasttissue	0.545	0.636
ecoli	0.861	0.809
energyy1	0.919	0.916
energyy2	0.890	0.916
fertility	0.710	0.810
glass	0.665	0.698
habermansurvival	0.706	0.758
hayesroth	0.628	0.531
ilpdindianliver	0.650	0.658
iris	0.968	1.000
mammographic	0.835	0.819
monks1	0.839	0.685
monks2	0.696	0.592
monks3	0.775	0.791
pima	0.738	0.760
pittsburgbridgesMATERIAL	0.900	0.818
pittsburgbridgesRELL	0.471	0.571
pittsburgbridgesSPAN	0.633	0.611
pittsburgbridgesTORD	0.800	0.900
pittsburgbridgesTYPE	0.755	0.818
postoperative	0.632	0.579
seeds	0.951	0.930
teaching	0.410	0.581
tictactoe	1.000	1.000
vertebralcolumn2clases	0.848	0.857
vertebralcolumn3clases	0.825	0.841

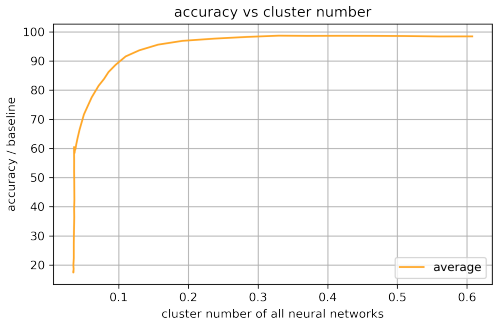
Table 6.2: A table of some baselines. In this table, we take full model into consideration. In column "independent training", accuracy of each neural network, which are sepratelly trained, are listed. In column "without gravity" data are trained together, but without the influence of gravity.

on accuracy becomes invisible when it reaches 400, or the number of clusters reaches 20% of the baseline 0 gravity.

In the scatter plot, both production cost and accuracy are considered. At the black point, each neural network has different structures, therefore, their accuracy is also the highest. Then, at the blue point, under the action of gravity and clustering algorithm, the neural network saves nearly 80% of the production cost, and the accuracy is only slightly reduced. Finally, at the yellow point. When the production



(a) relation between accuracy and cluster number compares to split training



(b) relation between accuracy and cluster number compares to 0 gravity

Figure 6.3: Figures of the relation between accuracy and cluster number compares to different baselines

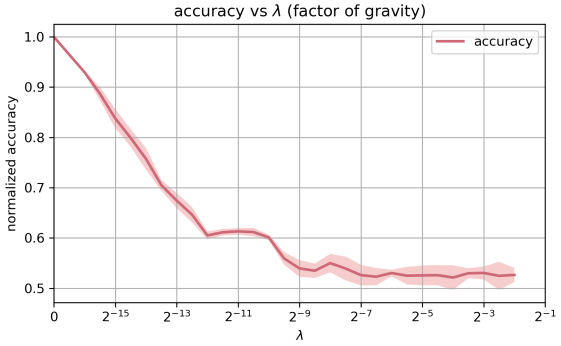


Figure 6.4: This image shows the relationship between accuracy and lambda in the case of printing neuromorphic.

cost is less than 10% of the maximum value, the neural network still maintains an accuracy greater than 70%. In the right part of the Fig. 6.6, production cost is reduced without even influencing the accuracy. In the left part, we provide the opportunity to choose from a better performance or a lower production cost.

Table 6.3: Accuracy-Cost Trade-off

	summerized accuracy	**normalized production cost**
independent training	100%	100%
our approach	101.8%	97.2%
	100.0%	20.2%
	95.2%	11.3%
	90.2%	7.8%
	85.3%	9.0%

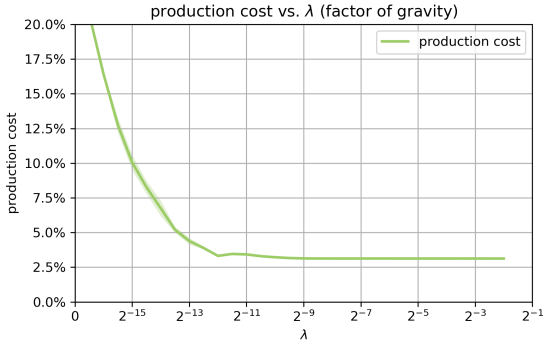


Figure 6.5: This image shows the relationship between production costs and lambda in the case of printing neuromorphic.

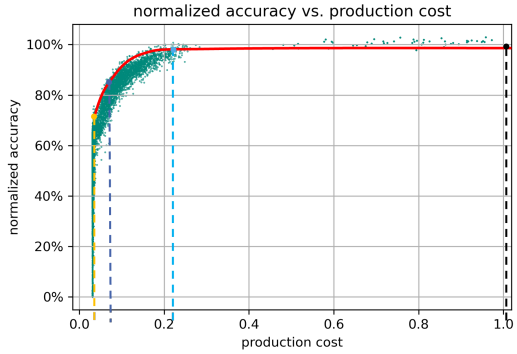


Figure 6.6: Scatter plot of normalized accuracy versus cost of printing in the case of printing neuromorphic for all the runs.

7. Conclusion and Future Work

7.1 Conclusion

This paper presents a novel approach to the training of neural networks, considering not only the accuracy but also the cost of printing multiple different circuits. Based on gravity attraction and the DBSCAN clustering method, we can fuse a large amount of weights in multiple neural networks, and thus, bridge the gap between high-volume production and low-volume production when it comes to neural network printing. This approach also inherits the advantages of both production methods. Testing on the basis of different datasets and models has shown that this training method has excellent performance. Under the action of gravity and clustering algorithm, the production of neural networks saves nearly 80% of the cost, and the accuracy is only slightly reduced. When the production cost is less than 10% of the maximum value, the neural network still maintains an accuracy greater than 70%. This training method allows pNN to achieve significant production cost savings at the cost of not losing too much accuracy.

7.2 Future Work

In this paper, we proved the effectiveness of the proposed training framework. However, a critical drawback of this work is the scalability, as the computation for gravity is $\mathcal{O}(n^2)$. To overcome this problem, advanced method for reducing computational cost should be considered, e.g., Monte-Carlo approach.

Secondly, the cost of printing is estimated by the number of clusters. A more precise cost model should be established.

Thirdly, the number of clusters is implicitly controlled by the scaling factor of the gravity term. Other techniques might be introduced to explicitly control the number of clusters, but without accuracy loss.

Lastly, too much additional tuning hyperparameters are introduced to this work. Some tricks such as progressive tuning can be considered.

Bibliography

[1] Masha Asulin et al. “One-step 3d Printing of Heart Patches With Built-in Electronics for Performance Regulation”. In: *Advanced Science* 8.9 (2021), p. 2004205.

[2] Eric Berthonnaud et al. “Analysis of the sagittal balance of the spine and pelvis using shape and orientation parameters”. In: *Clinical Spine Surgery* 18.1 (2005), pp. 40–47.

[3] Rafał Biedrzycki and Jarosław Arabas. “Evolutionary and greedy exploration of the space of decision trees”. In: *Evolutionary Computation and Global Optimization. Prace Naukowe, Elektronika Warsaw, Poland: Warsaw University of Technology Publishing House* (2006), pp. 479–489.

[4] Joseph Chang, Tong Ge, and Edgar Sanchez-Sinencio. “Challenges of Printed Electronics on Flexible Substrates”. In: *2012 IEEE 55th international midwest symposium on circuits and systems (MWSCAS)*. IEEE. 2012, pp. 582–585.

[5] Małgorzata Charytanowicz et al. “Complete gradient clustering algorithm for features analysis of x-ray images”. In: *Information technologies in biomedicine*. Springer, 2010, pp. 15–24.

[6] Yi-Min Chou et al. “Unifying and merging well-trained deep neural networks for inference stage”. In: *arXiv preprint arXiv:1805.04980* (2018).

[7] Jacek Czerniak and Hubert Zarzycki. “Application of rough sets in the presumptive diagnosis of urinary system diseases”. In: *Artificial intelligence and security in computing systems*. Springer, 2003, pp. 41–51.

[8] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017.

[9] Matthias Elter, Rüdiger Schulz-Wendtland, and Thomas Wittenberg. “The prediction of breast cancer biopsy outcomes using two CAD approaches that both emphasize an intelligible decision process”. In: *Medical physics* 34.11 (2007), pp. 4164–4172.

[10] Martin Ester et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[11] Ronald A Fisher. “The use of multiple measurements in taxonomic problems”. In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[12] David Gil et al. “Predicting seminal quality with artificial intelligence methods”. In: *Expert Systems with Applications* 39.16 (2012), pp. 12564–12573.

[13] Shelby J Haberman. *Generalized residuals for log-linear models*. 1976.

[14] Barbara Hayes-Roth and Frederick Hayes-Roth. “Concept learning and the recognition and classification of exemplars”. In: *Journal of Verbal Learning and Verbal Behavior* 16.3 (1977), pp. 321–338.

[15] Paul Horton and Kenta Nakai. “A probabilistic classification system for predicting the cellular localization sites of proteins.” In: *Ismb*. Vol. 4. 1996, pp. 109–115.

[16] Cezary Z Janikow. “A knowledge-intensive genetic algorithm for supervised learning”. In: *Genetic Algorithms for Machine Learning*. Springer, 1993, pp. 33–72.

[17] J Jossinet. “Variability of impedivity in normal and pathological breast tissue”. In: *Medical and biological engineering and computing* 34.5 (1996), pp. 346–350.

[18] Altynay Kaidarova et al. “Wearable Multifunctional Printed Graphene Sensors”. In: *NPJ Flexible Electronics* 3.1 (2019), pp. 1–10.

[19] David Klahr and Robert S Siegler. “The representation of children’s knowledge”. In: *Advances in child development and behavior*. Vol. 12. Elsevier, 1978, pp. 61–116.

[20] Isidoro Ibanez Labiano and Akram Alomainy. “Flexible Inkjet-printed Graphene Antenna on Kapton”. In: *Flexible and Printed Electronics* 6.2 (2021), p. 025010.

[21] Wei-Yin Loh and Yu-Shan Shih. “Split selection methods for classification trees”. In: *Statistica sinica* (1997), pp. 815–840.

[22] Yanping Lu et al. “Particle swarm optimizer for variable weighting in clustering high-dimensional data”. In: *Machine learning* 82.1 (2011), pp. 43–70.

[23] Christopher J Matheus and Larry A Rendell. “Constructive Induction On Decision Trees.” In: *IJCAI*. Vol. 89. 1989, pp. 645–650.

[24] WS McCullock and W Pitts. “A logical calculus of ideas immanent in nervous activity. archive copy of 27 november 2007 on wayback machine”. In: *Avtomaty [Automated Devices] Moscow, Inostr. Lit. publ* (1956), pp. 363–384.

[25] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[26] Tomaso Poggio. “A theory of how the brain might work”. In: *Cold Spring Harbor symposia on quantitative biology*. Vol. 55. Cold Spring Harbor Laboratory Press. 1990, pp. 899–910.

[27] Lutz Prechelt. “Early stopping-but when?” In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[28] Bendi Venkata Ramana, M Surendra Prasad Babu, and NB Venkateswarlu. “A critical comparative study of liver patients from USA and INDIA: an exploratory analysis”. In: *International Journal of Computer Science Issues (IJCSI)* 9.3 (2012), p. 506.

[29] Yoram Reich and Steven Joseph Fenves. *Incremental learning for capturing design expertise*. [Carnegie Mellon University], Engineering Design Research Center, 1989.

[30] Sebastian Ruder. “An overview of multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1706.05098* (2017).

[31] George Smith. "Newton's philosophiae naturalis principia mathematica". In: (2007).

[32] W Nick Street, William H Wolberg, and Olvi L Mangasarian. "Nuclear feature extraction for breast tumor diagnosis". In: *Biomedical image processing and biomedical visualization*. Vol. 1905. SPIE. 1993, pp. 861–870.

[33] Sathishkumar VE, Changsun Shin, and Yongyun Cho. "Efficient energy consumption prediction model for a data analytic-enabled industry building in a smart city". In: *Building Research & Information* 49.1 (2021), pp. 127–143.

[34] Dennis D Weller et al. "Realization and training of an inverter-based printed neuromorphic computing system". In: *Scientific reports* 11.1 (2021), pp. 1–13.

[35] William H Wolberg and Olvi L Mangasarian. "Multisurface method of pattern separation for medical diagnosis applied to breast cytology." In: *Proceedings of the national academy of sciences* 87.23 (1990), pp. 9193–9196.

[36] Linda Woolery et al. "The use of machine learning program LERS-LB 2.5 in knowledge acquisition for expert system development in nursing." In: *Computers in nursing* 9.6 (1991), pp. 227–234.

[37] I-Cheng Yeh, King-Jang Yang, and Tao-Ming Ting. "Knowledge discovery on RFM model using Bernoulli sequence". In: *Expert Systems with Applications* 36.3 (2009), pp. 5866–5871.

[38] Haibin Zhao et al. "Aging-Aware Training for Printed Neuromorphic Circuits". In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 2022, pp. 1–9.

[39] Haibin Zhao et al. "Printed Electrodermal Activity Sensor with Optimized Filter for Stress Detection". In: *Proceedings of the 2022 ACM International Symposium on Wearable Computers*. 2022, pp. 112–114.

[40] Haibin Zhao et al. "Split Additive Manufacturing for Printed Neuromorphic Circuits". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2023.

[41] Fan Zhou et al. "Task similarity estimation through adversarial multitask neural network". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.2 (2020), pp. 466–480.