國 立 中 央 大 學

資訊工程學系

碩士論文

關聯式學習：利用自動編碼器與目標傳遞法分解端到端倒傳遞演算法

Associated Learning: Decomposing End-to-end Backpropagation based on Auto-encoders and Target Propagation

研 究 生：高聿緯

指導教授：陳弘軒　博士

中 華 民 國 一百零九 年 一 月

國立中央大學圖書館
碩博士論文電子檔授權書

(104 年 5 月最新修正版)

本授權書授權本人撰寫之碩/博士學位論文全文電子檔(不包含紙本、詳備註 1 說明)，在「國立中央大學圖書館博碩士論文系統」。(以下請擇一勾選)

(✓)同意 (立即開放)

(　)同意 (請於西元 _____年_____月_____日開放)

(　)不同意，原因是：_____

在國家圖書館「臺灣博碩士論文知識加值系統」

(✓)同意 (立即開放)

(　)同意 (請於西元 _____年_____月_____日開放)

(　)不同意，原因是：_____

以非專屬、無償授權國立中央大學、台灣聯合大學系統圖書館與國家圖書館，基於推動「資源共享、互惠合作」之理念，於回饋社會與學術研究之目的，得不限地域、時間與次數，以紙本、微縮、光碟及其它各種方法將上列論文收錄、重製、與利用，並得將數位化之上列論文與論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

研究生簽名: 高聿緯　　學號：106522127

論文名稱: 關聯式學習：利用自動編碼器與目標傳遞法分解端到端剛傳遞演算法

指導教授姓名：陳弘軒

系所：資訊工程所　☐博士班　☑碩士班

填單日期：_____

備註：

1. 本授權書之授權範圍僅限**電子檔**，紙本論文部分依著作權法第 15 條第 3 款之規定，採推定原則即預設同意圖書館得公開上架閱覽，如您有申請專利或投稿等考量，不同意紙本上架陳列，須另行加填申請書，詳細說明與紙本申請書下載請至本館數位博碩論文網頁。
2. 本授權書請填寫並**親筆**簽名後，裝訂於各紙本論文封面後之次頁（全文電子檔內之授權書簽名，可用電腦打字代替）。
3. 讀者基於個人非營利性質之線上檢索、閱覽、下載或列印上列論文，應遵守著作權法規定。

國立中央大學碩士班研究生

論文指導教授推薦書

資訊工程 學系/研究所 高聿緯 研究生所提之論文 關聯式學習：利用自動編碼器與目標傳遞法分解端到端剛傳遞賃貸法 係由本人指導撰述，同意提付審查。

指導教授 _____ (簽章)

____年____月____日

國立中央大學碩士班研究生

論文口試委員審定書

資訊工程 學系/研究所 高聿鑄 研究生

所提之論文

關聯式學習：利用自動編碼器與目標傳遞法分解端到端傳遞演算法

經本委員會審議，認定符合碩士資格標準。

學位考試委員會召集人 蔡孟峰

委 員 王家慶

洪弘昇

蔡孟峰

中 華 民 國 年 月 日

101.06.15

關聯式學習：利用自動編碼器與目標傳遞法分解端到端倒傳遞演算法

摘要

倒傳遞演算法已被廣泛的運用在深度學習上，但因為有傳遞鎖與梯度消失、爆炸的問題，它不是有效率且穩定的演算法，在較深的網路架構更可以觀察到這些現象。此外，單單只用一個目標來更新神經網路中的參數在生物學來說並非合理的。

在本篇論文中，我們提出了一種新穎且受生物學啟發的學習架構，名為「關聯式學習」，這個訓練方式將原有的神經網路模組化成小單元，每個小單元都有自己的局部目標，又因為這些單元兩兩獨立，關聯式學習能夠獨立且同時訓練彼此獨立的參數。

令人驚訝的是，利用關聯式學習訓練的準確度，也能與直接使用目標訓練的傳統倒傳遞演算法相當，此外，可能是因為模組內的梯度流較短，關聯式學習也能訓練用 sigmoid 當作活化函數的深度學習網路，然而若是用倒傳遞演算法訓練這類網路會容易導致梯度消失。

我們也透過觀察隱藏層中的類間與類內距離，以及 t-SNE 來呈現數量上與品質上的結果，發現聯想式學習能夠生成更好的間特徵 (Metafeatures)。

關鍵字：生物合理性演算法, 深度學習, 平行運算, 模組化

Associated Learning: Decomposing End-to-end Backpropagation based on Auto-encoders and Target Propagation

Abstract

Backpropagation has been widely used in deep learning approaches, but it is inefficient and sometimes unstable because of backward locking and vanishing/exploding gradient problems, especially when the gradient flow is long. Additionally, updating all edge weights based on a single objective seems biologically implausible. In this paper, we introduce a novel biologically motivated learning structure called Associated Learning, which modularizes the network into smaller components, each of which has a local objective. Because the objectives are mutually independent, Associated Learning can learn the parameters independently and simultaneously when these parameters belong to different components. Surprisingly, training deep models by Associated Learning yields comparable accuracies to models trained using typical backpropagation methods, which aims at fitting the target variable directly. Moreover, probably because the gradient flow of each component is short, deep networks can still be trained with Associated Learning even when some of the activation functions are sigmoid—a situation that usually results in the vanishing gradient problem when using typical backpropagation. We also found that the Associated Learning

generates better metafeatures, which we demonstrated both quantitatively (via inter-class and intra-class distance comparisons in the hidden layers) and qualitatively (by visualizing the hidden layers using t-SNE).

Keywords: Biologically plausible algorithm, Deep learning, Parallel computing, Modularization

Contents

List of Figures

page

List of Tables

page

Chapter 1

Introduction

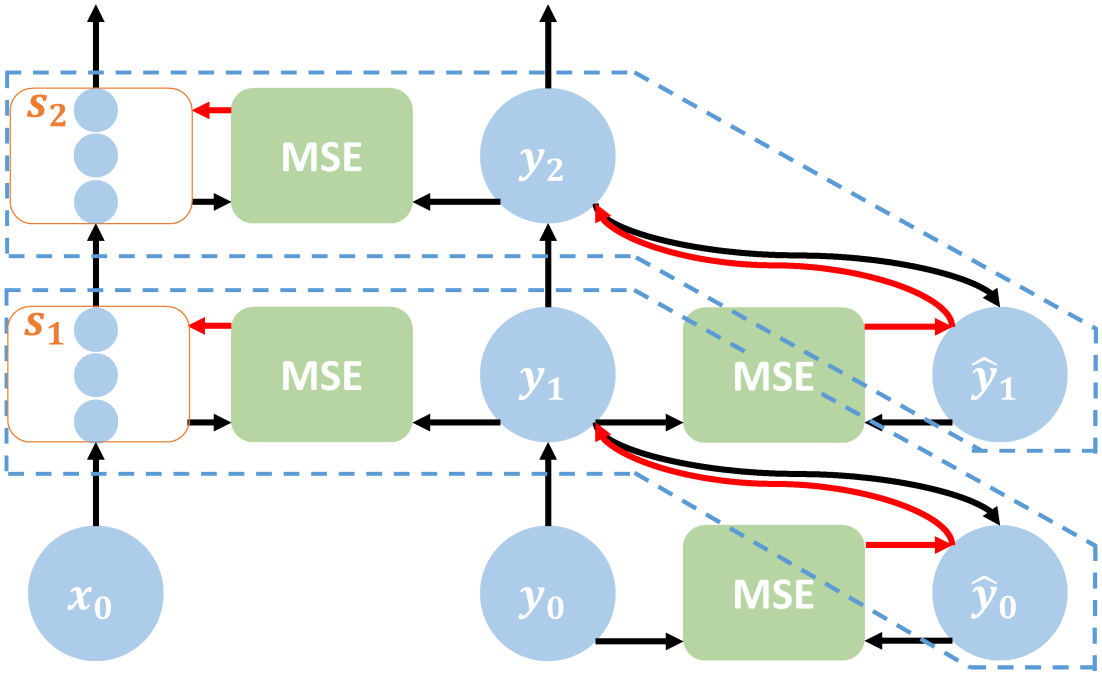

Figure 1.1: The structure of Associated Learning, where x_0 indicates input data, y_0 indicates labels, s_i is a subset of the network, and y_i is the corresponding target. The solid black arrows indicate data paths and the solid red arrows indicate gradient flows. The parameters within the region of each blue dashed line can be updated independently.

Deep neural networks are usually trained using backpropagation [1], which, although common, increases the training difficulty of deep neural networks for several reasons. First, the end-to-end training method

propagates error signals layer by layer; consequently, it prevents us from updating network parameters in parallel. This problem is called backward locking and is discussed in [2]. Backward locking limits the speed of neural network training, especially when the network has many layers. Second, the weights in a deep neural network may be updated slowly because of the vanishing gradient problem [3]. Several methods have been proposed to address this issue. One approach is to replace the traditional activation functions (e.g., sigmoid function or tanh function) with functions that are less likely to become saturated, such as ReLU [4]. Another approach is to modify the network structure so that the gradients are more likely to extend across many layers. The most representative works along these lines include LSTM [5], which adds additional gates in the cells, ResNet [6], which creates explicit shortcuts to help with gradient flows, and Batch Normalization [7], which adjusts and scales activations to mitigate the internal covariate shift. Third, the weights of a deep neural network may become unstable due to the exploding gradient problem. This problem is usually addressed by switching to an LSTM or by gradient clipping [8].

Beyond these computational weaknesses, backpropagation-based learning seems biologically implausible. For example, it is unlikely that all the weights should be adjusted sequentially and in small increments based on a single objective [9]. Additionally, some components that are essential for backpropagation to work correctly have not been observed in the cortex [10]. Therefore, many works have proposed methods that more closely resemble the operations of biological neurons [11, 12, 13, 14, 15, 16].

In this paper, we propose Associated Learning, a method that can either be integrated with or used to replace end-to-end backpropagation when training a deep neural network. The Associated Learning scheme consists of two parts: (1) a sequence of non-linear layers (as shown in the

left side of Figure 1.1); and (2) a sequence of autoencoders (as shown in the right side of Figure 1.1). The number of non-linear layers in the left side equals the number of autoencoders in the right side. We want s_i, the output of every non-linear layer in the left, to be close to y_i, the bottleneck of the corresponding autoencoder. Thus, we decompose the objective function into many small objectives. Consequently, each gradient flow becomes short, and gradients do not flow to the previous layer. As a result, the parameters in each blue dashed line form a group and do not affect the parameters in other groups. Therefore, this learning method decomposes the training process into small subnetworks that are independent to other subnetworks and therefore effectively alleviates the backward lock issue. Now the training process can be parallel to improve the training throughput.

The remainder of this paper is organized as follows. In Chapter 2, we introduce the Associated Learning . In Chapter 3, we show the comparisons between Associated Learning and backpropagation-based learning using different types of neural networks and different datasets. We review the related works in Chapter 4. Finally, we discuss the discoveries and suggest future work in Chapter 5.

Chapter 2

Methodology

2.1 Preliminaries

2.1.1 Artificial Neural Network

Inspired by neurons in our brain, artificial neural network simulate how our brain works. The weights in artificial neural network act like the connection between neurons. In other words, every neuron in layer i is the weighted sum of the neurons in layer $i-1$. We use activation function to simulate the synaptic gating. It controls how much information can pass to the next layer. For a single layer, it received the information from its previous layer and then do an inner product with its weights. Then, feed these outputs to the activation function to calculate how much information can pass through this layer. The equation 2.1 shows how it works:

$$a_i = \sigma(a_{i-1} \cdot W_i) \tag{2.1}$$

where a_i indicates the output, W_i indicates the weights of layer i and σ indicates the activation function. The importance of activation function is: (1)If we stack up multiple layers without activation function, they will

shrink into a single layer by doing inner product. (2)Some data are not linearly separable, so non-linearity plays an important role.

2.1.2 Backpropagation

We use gradient descent to update parameters in some of the machine learning models. The same method is used in deep learning models, but the parameters have to be updated layer by layer. That is, if we want to update the parameters in layer i, we will have to update layer $i+1$ first and then calculate the partial derivative of layer i to update parameters. The error signal is backward propagated, so the algorithm is named backpropagation.

2.1.3 Models

In this section, we introduce the models we used in our experiments.

2.1.3.1 Multilayer Perceptron

Just like we mentioned in 2.1.1, multilayer perceptron stack up multiple non-linear layers to extract higher dimensional features from inputs. The structure is shown in Figure 2.1. With massive parameters and abundant non-linearity, multilayer perceptron performs better than some machine learning technique.

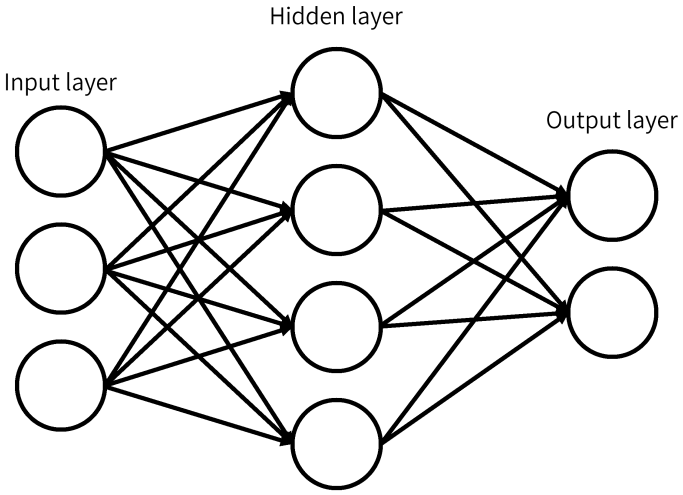

Figure 2.1: The structure of a multilayer perceptron.

2.1.3.2 Convolution Neural Network

Convolution neural network is used commonly in computer vision. For every convolution layer, we are learning the best filters which extract useful information from images. These filters side through every pixel in the image to get plenty of new representatives of the original image. Note that weight sharing is used to decrease the number of parameters. That is, same filters are used for every window sliding.

2.1.3.3 VGGNet

VGGNet stacks up multiple convolution layers to their model and got second place on the ILSVRC competition. In their experiment settings, they even stacked up 19 layers and increased the size of filters to 512 on their VGG-19 model. It starts a new era of "very" deep neural networks. Massive parameters and nonlinearity were equal to better performance at

that time.

2.1.3.4 ResNet

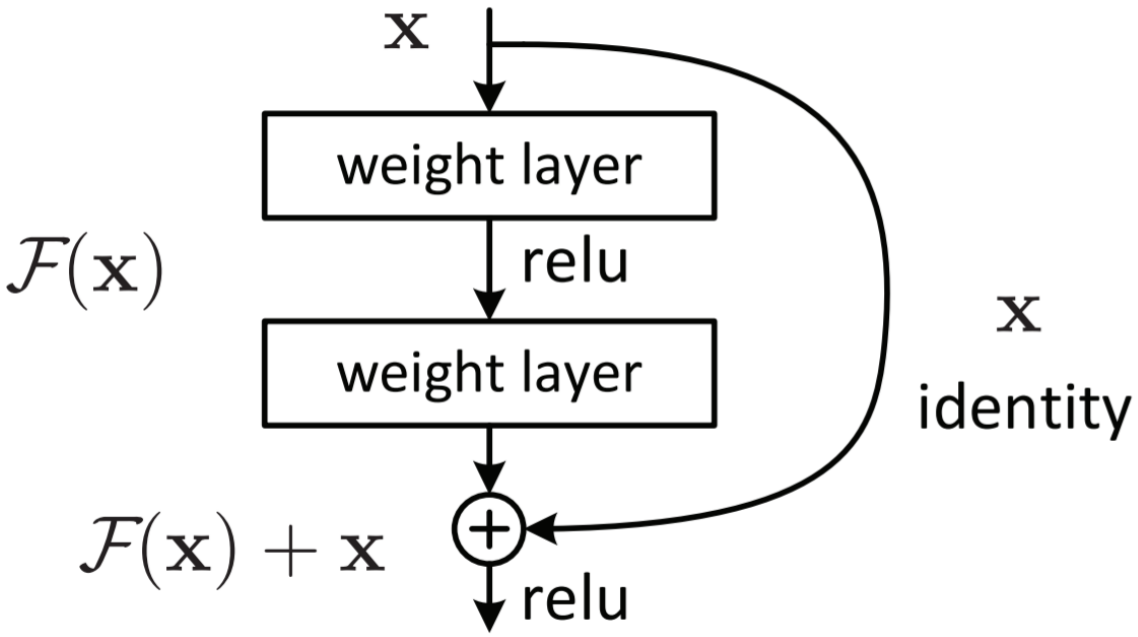

Figure 2.2: A residual block.

The authors of ResNet [6] show that deeper models result in accuracy saturation based on their observation. Therefore, they brought up the idea of shortcuts to help the gradient flows. The structure is shown in Figure 2.2. While input passes through nonlinear layers, it is also delivered directly to the output layer and do an element-wise addition. During training, gradients can be backpropagated via this shortcut directly. Thus, the error signal is guaranteed to flow to the first layer and prevents models from getting degradation.

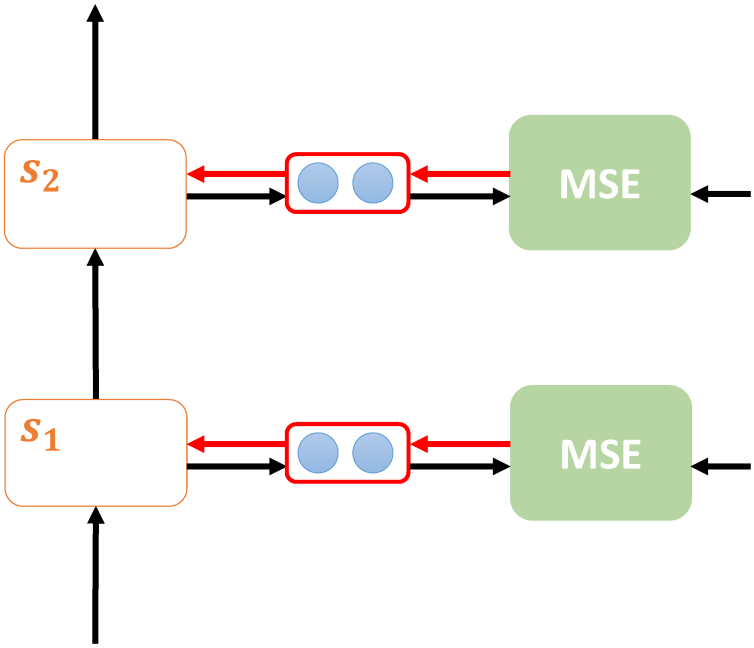

Figure 2.3: The red rounded rectangle is "Bridge", which added multiple non-linear layers to help S_is fit well. The black arrows indicate the data flow and the red arrows indicate the gradient flow.

2.2 Motivation

A typical deep network training process requires features to pass through multiple nonlinear layers, allowing the output to approach to the ground truth labels. Therefore, there is only one objective. In Associated Learning, however, we modularize the training path by splitting it into smaller components and assign a local objective to each small component. Consequently, the Associated Learning divides the original long gradient flow into many independent small gradient flows and effectively eliminates the backward lock problem.

2.3 Forward Loss of Associated Learning

Referring to Figure 1.1, let x_0 and y_0 be the input and output vectors of a training sample. We split the network into N subnetworks such that each subnetwork i $(i = 1, \ldots, N)$ consists of a local forward function, f_i, a local activation value, s_i, a local target vector, y_i, and a local objective function that is independent of the objective functions of the other subnetworks. A local forward function f_i can be a simple single-layer perceptron, a multi-layer perceptron, a convolution layer, or other functions. We compute s_i and y_i using Equations 2.2 and 2.3, respectively.

$$s_i = f_i(s_{i-1}),\ i = 1, \ldots, N. \tag{2.2}$$

Note that here, s_0 equals x_0:

$$y_i = g_i(y_{i-1}) = \sigma(Q_i y_{i-1}),\ i = 1, \ldots, N, \tag{2.3}$$

where y_i and g_i are the generated local target variable and the nonlinear mapping, respectively, of the i-th layer. Biases are included in Q_i.

Also note that we plotted Figure 1.1 based on the TensorBoard convention, where the black arrows indicate the data path, but the weights to be learned (e.g., Q_is and weights in f_is) are not shown explicitly. For example, the arrow connecting x_0 and s_1 denotes that the value of x_0 is passed to the neighboring subnetwork, which needs to compute $f_1(x_0)$ to obtain s_1. This approach differs from some neural network figures where the arrows are associated with the weights to be learned.

We define the *forward loss function* for each pair of (s_i, y_i) $(i = 1, \ldots, N)$ by Equation 2.4. This concept is the same as in target propagation [17, 18, 19, 15]: our goal is to minimize the distance between s_i and y_i for every

local subnetwork i.

$$L_i(s_i, y_i) = ||s_i - y_i||^2,\ i = 1, \ldots, N. \tag{2.4}$$

Because the optimizer in the i-th subnetwork updates the parameters in f_i only to reduce the local loss function (Equation 2.4), we create N independent objectives that can be trained independently.

2.4 Inverse Loss of Associated Learning

In addition to updating all the parameters in the f_is to minimize the forward loss, $L_i(s_i, y_i)$, an inverse mapping must be trained at every subnetwork-i so that at the prediction phase, we can perform a transformation from y_i to y_{i-1} $(i = 1, \ldots, N)$. This transformation process is shown in Equation 2.5:

$$\hat{y}_{i-1} = h_i(y_i) = \sigma(V_i y_i) \tag{2.5}$$

where the biases are included in V_i.

We call the function h_i the "inverse mapping function" because it transforms y_i to y_{i-1}. To obtain the inverse mapping function h_i, we use the mapping function g_i in Equation 2.3, which is called the forward mapping function, such that $h_i(g_i(y_{i-1})) \approx y_{i-1}$. Thus, we can view $g_i(.)$ as the encoder and $h_i(.)$ as the decoder for the subnetwork i.

The inverse loss L_i' for the layer i is defined by Equation 2.6.

$$L_i'(h_i(g_i(y_{i-1})), y_{i-1}) = \text{MSE}(\hat{y}_{i-1}, y_{i-1}),\ i = 1, \ldots, N \tag{2.6}$$

where MSE(.) returns the mean square error of the two arguments.

For a well-trained network, s_N and y_N should eventually be very similar. We then input s_N into $h_N(.)$ to infer our prediction.

2.5 Bridge of Associated Learning

In early experiments, we observed that s_is are difficult to fit to their corresponding targets y_is, especially for convolutional neural networks and their extensions. Thus, we insert nonlinear layers to improve the fit between the s_is and y_is. As shown in Figure 2.3, each orange rounded rectangle represents the s_i in Figure 1.1, and the solid circles represent the nonlinear layers. We call the set of all these nonlinear layers in one subnetwork a *bridge*, which is denoted by the red rounded rectangle. As a result, the forward loss is reformulated to the following equation:

$$L_i(s_i, y_i) = ||b_i(s_i) - y_i||^2, \ i = 1, \ldots, N, \tag{2.7}$$

where the function $b_i(.)$ serves as the bridge.

Although this approach increases many parameters and the nonlinear layers to decrease the forward loss; in the inferencing phase, except for the last bridge, these parameters do not count because they do not affect the predictions, as we will explain in the next section.

2.6 Effective Parameters and Hypothesis Space

We can categorize the above-mentioned parameters into two parts: the effective parameters and the affiliated parameters. The affiliated parameters help the model determine the values of the effective parameters, which in turn determine the hypothesis space of the final prediction function.

Therefore, while increasing the number of affiliated parameters may help in obtaining better values for the effective parameters, this will not increase the hypothesis space of the prediction model.

Specifically, in the training phase, we search for the parameters in f_is and b_is that minimize the forward loss, g_is and h_is to minimize the inverse loss. However, in the prediction phase, we make predictions based only on Equation 2.2, Equation 2.5, and $b_N(s_N)$. Therefore, the effective parameters include only the parameters in f_is, b_N (i.e., the last bridge), and h_is. The parameters in the other functions (i.e., all the g_is and b_js ($j = 1, \ldots, N-1$)) are affiliated parameters; they do not increase the expressiveness of the model but only help determine the values of the effective parameters.

Equation 2.8 shows the prediction function:

$$\hat{y}_0 = (h_1 \circ h_2 \circ \ldots \circ h_{N-1} \circ h_N \circ b_N \circ f_N \circ f_{N-1} \circ \ldots \circ f_2 \circ f_1)(x_0), \quad (2.8)$$

where $\circ$ denotes the function composition operation.

Chapter 3

Experiments

In this chapter, we show the results of performance comparisons between backpropagation (BP) and Associated Learning (AL) from different aspects. First, we quantify the performance of AL and BP based on their test accuracies. Surprisingly, although the AL aims at minimizing the local losses, its prediction accuracy is comparable to or better than that of backpropagation-based learning, whose goal is to directly minimize the prediction error. Second, we visualize the activations of hidden layers using t-SNE [20] to measure the qualities of the metafeatures learned by AL and BP. Finally, we list the intra-class and inter-class distances of BP and AL to assess the metadata quality. We conducted these experiments by applying AL and BP to different deep neural network structures (including MLP, CNN, VGG [21], ResNet-20, and ResNet-32) and testing them on different datasets (including MNIST [22], CIFAR-10, and CIFAR-100 [23]).

In each experiment, the networks are trained using 200 epochs with a batch size of 32. We initialize all the weights based on the He normal initializer, and use Adam as the optimizer. We experimented with different activation functions and adopted ELU for all the local forward functions (i.e., f_i) and sigmoid for the functions related to the autoencoders (i.e., g_i,

h_i and b_i). Likewise, for backpropagation, we experimented with different activation functions and decided to adopt ReLU for all the layers. In addition, because Associated Learning includes extra parameters in the function b_N (the last bridge), as explained in Section 2.6, we increased the number of layers in the corresponding baseline models when training by backpropagation so that the models trained by Associated Learning and trained by backpropagation have identical parameters.

3.1 Datasets

We conduct experiments on three datasets, MNIST [22], CIFAR-10, and CIFAR-100 [23].

3.1.1 MNIST

MNIST is an abbreviation of **Modified National Institute of Standards and Technology**. That is, MNIST is a subset of NIST dataset. The dataset is composed of handwritten digits images from 0 to 9. 50% of these images were collected from Census Bureau employees and the other 50% were collected from high school students. These images are in gray-scale and the resolution is 28×28 as shown in Figure 3.1. There are 60,000 training data and 10,000 testing data in MNIST dataset.

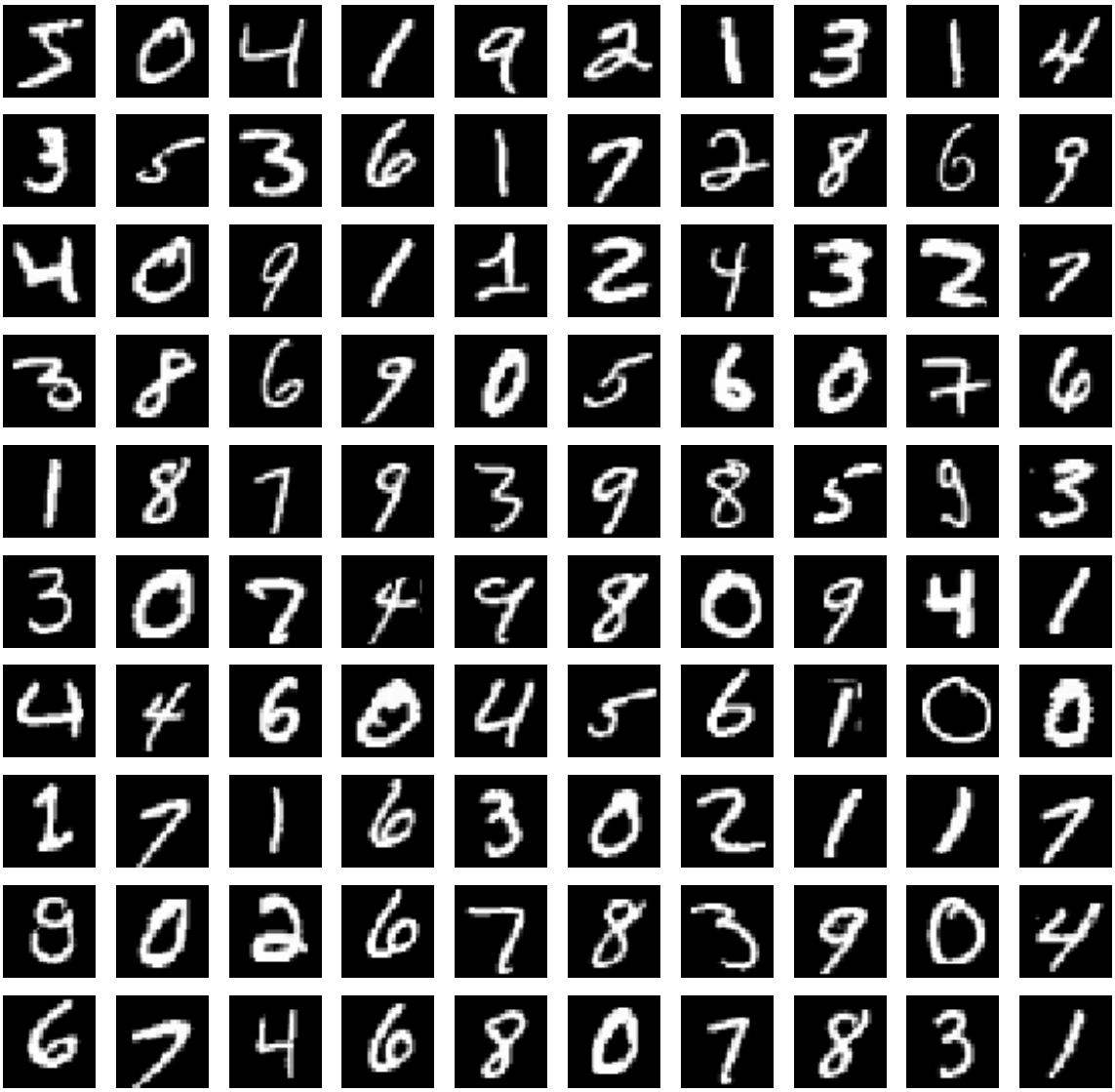

Figure 3.1: MNIST is a dataset of handwritten digits. The digits are in grayscale and are either 0 to 9.

3.1.2 CIFAR

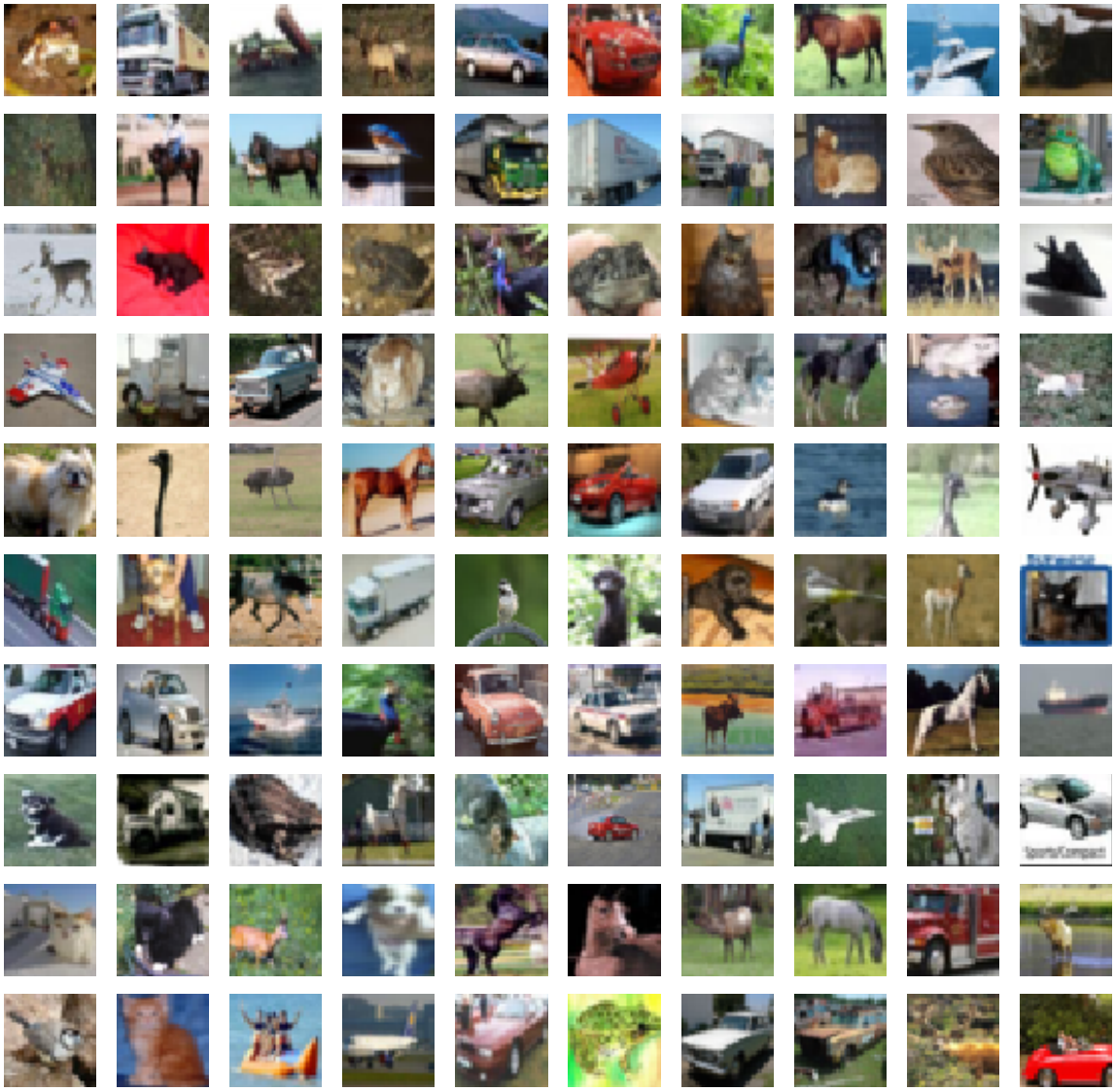

Figure 3.2: CIFAR-10 is a dataset of real-life images. These images are color images and are labeled either airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

Both CIFAR-10 and CIFAR-100 datasets contain $50,000$ for training data and $10,000$ for testing data. As we showed in Figure 3.2, they are color images and the resolution is 32×32 which means that they contain more features for training. The notation 10 and 100 denote the number of

classes and the number of data is equally distributed. That is, each class in CIFAR-10, there are $5,000$ training data and $1,000$ testing data. And each class in CIFAR-100, there are 500 training data and 100 testing data.

3.2 Testing Accuracy

3.2.1 MNIST

On the MNIST dataset, we conducted experiments with only two network structures, MLP and CNN, beacause using even these simple structures yielded decent test accuracy (from 98.6% to 99.5%). The MLP contains 6 layers (5 hidden layers and 1 output layer), and the number of neurons in these layers are $1024, 1024, 5120, 1024, 1024$, and 10, respectively. The CNN contains 14 layers (13 hidden layers and 1 output layer). The first 4 are convolutional layers with a size of $3\times3\times32$ (i.e., width: 3; height: 3; and 32 kernels) in each layer, followed by 4 convolutional layers with a size of $3 \times 3 \times 64$ in each layer, followed by a fully connected layer with 1280 neurons, followed by 4 fully connected layers with 256 neurons in each layer, and ending with a fully connected layer with 10 neurons. The initial learning rate is 10^{-4}, which is reduced after 80, 120, 160, and 180 epochs. We neither augment the input images nor perform any regularization in this experiment.

The results are shown in Table 3.1. For both the MLP and the CNN

Table 3.1: Test accuracy comparison on the MNIST dataset. We highlight the winner in bold font.

	MLP	CNN
BP	98.6%	99.4%
AL	**98.7%**	**99.5%**

structure, Associated Learning performs slightly better than does backpropagation. Note that half the layers $(h_1, \ldots, h_N, b_N)$ in Associated Learning use sigmoid activation functions, which suffer from the vanishing gradient problem and usually performs unsatisfactorily as the number of layers increases. However—likely because each gradient flow is short in Associated Learning—the model still manages to obtain reasonable parameters, as demonstrated by the high test accuracy.

The training and testing accuracy curves are shown in Figure 3.3. We found that both MLP and CNN can be well trained with backpropagation and Associated Learning. However, training and testing curves with Associated Learning converge slightly faster than with backpropagation.

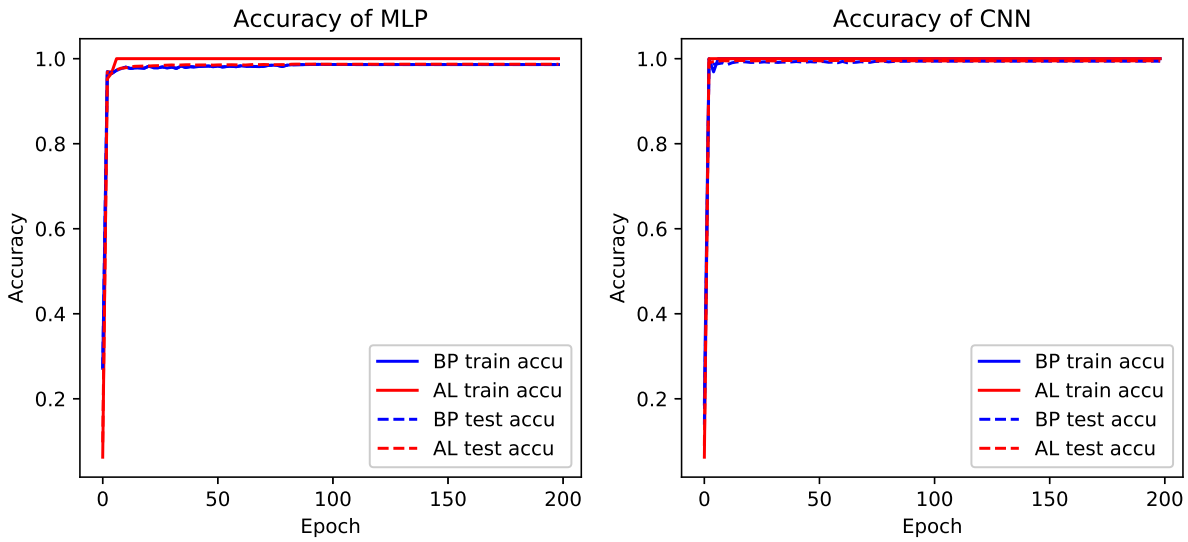

Figure 3.3: Training and testing accuracies on MNIST dataset. From left to right are results of MLP and CNN. The blue line and blue dashed line indicate the training and testing accuracies when training by BP. The red line and red dashed line indicate the training and testing accuracies with AL.

Table 3.2: Test accuracy comparison on the CIFAR-10 dataset. We highlight the winner in bold font.

	MLP	CNN	ResNet-20	ResNet-32	VGG
BP	61.6%	85.4%	**90.6%**	**91.9%**	**93.1%**
AL	**64.2%**	**86.0%**	89.3%	88.8%	92.6%

3.2.2 CIFAR-10

The CIFAR-10 dataset is more challenging than the MNIST dataset. The input image size is $32 \times 32 \times 3$ as we introduced in 3.1.2, i.e., the images have a higher resolution and each pixel includes RGB information. To make good use of these abundant features, we not only applied MLP and CNN in this experiment but also VGG and ResNet. The input images are augmented by 2-pixel jittering [24]. We applied the L2 norm using 5×10^{-4} on VGG and 1×10^{-4} on ResNet as the regularization weight.

Because ResNet uses batch normalization and the shortcut trick, we set its learning rate to 10^{-3}, slightly larger than the other models. Besides, to ensure a fair comparison, we added extra layers to ResNet-20, ResNet-32 and VGG when using backpropagation for learning.

Table 3.2 shows the results. Associated Learning still outperformed backpropagation on the MLP and CNN structures. With the state-of-the-art network structures VGG and ResNet, Associated Learning performs slightly worse than backpropagation. This result probably occurs because backpropagation aims to fit the target directly, but most of the layers in Associated Learning can leverage only indirect clues to update the parameters.

Figure 3.4 shows the training and testing accuracy curves in CNN, ResNet-20 and ResNet-32 In the early phase of training, with Associated Learning as the training method, the testing accuracies are higher than

the experiments training with backpropagation. Nevertheless, at the 80-*th* epoch, the first decay of learning rate, the accuracy improvement of backpropagation are more than Associated Learning.

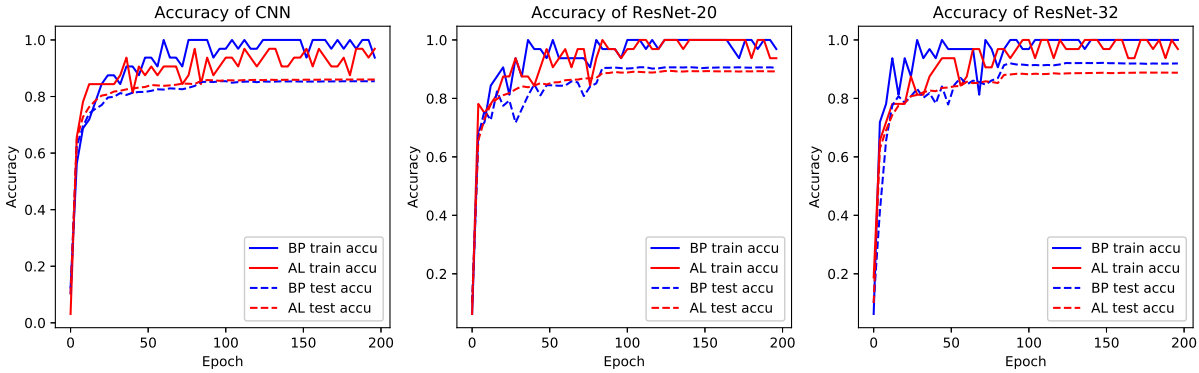

Figure 3.4: Training and testing accuracies on CIFAR-10 dataset. From left to right are results of CNN, ResNet-20 and ResNet-32. The blue line and blue dashed line indicate the training and testing accuracies when training by BP. The red line and red dashed line indicate the training and testing accuracies with AL.

3.2.3 CIFAR-100

The CIFAR-100 dataset includes 100 classes. We used model settings that were nearly identical to the settings used on CIFAR-10 but increased the number of neurons in bridge. To ensure a fair comparison, we increased the number of layers in the corresponding baseline models when training by backpropagation so that the models trained by Associated Learning and trained by backpropagation have identical parameters. Table 3.3 shows the

Table 3.3: Test accuracy comparison on the CIFAR-100 dataset. We highlight the winner in bold font.

	MLP	CNN	ResNet-20	ResNet-32	VGG
BP	26.9%	51%	**63.8%**	**63.8%**	**70.8%**
AL	**31.2%**	**52.4%**	60.2%	60.5%	67.5%

results. As in CIFAR-10, Associated Learning performs better on the MLP and the CNN structures, but slightly worse than backpropagation on the VGG and ResNet structures—likely because AL uses only local errors to update the parameters.

Figure 3.5 shows the training and testing accuracy curves. The results are roughly the same as the results in CIFAR10. But the accuracy gaps between backpropagation and Associated Learning on ResNet are larger than in CIFAR10. This shows that shortcut and Batch Normalization technique indeed benefit backpropagation algorithm a lot.

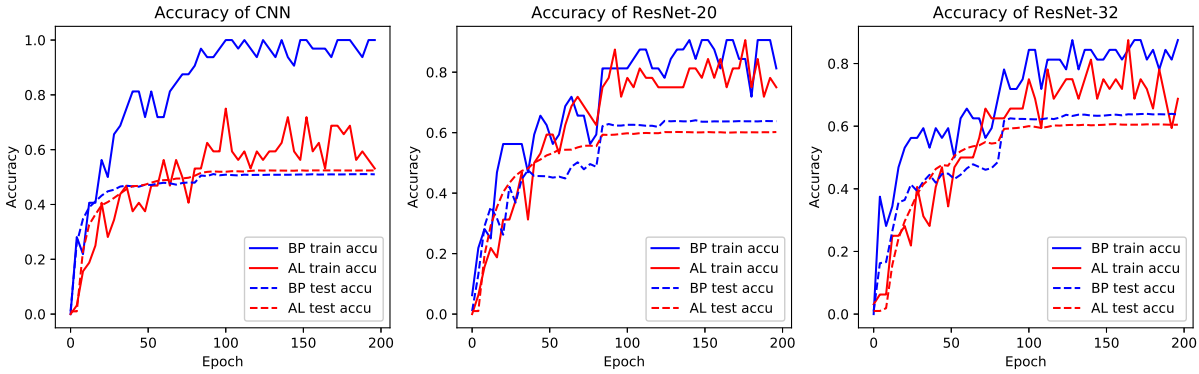

Figure 3.5: Training and testing accuracies on CIFAR-100 dataset. From left to right are results of CNN, ResNet-20 and ResNet-32. The blue line and blue dashed line indicate the training and testing accuracies when training by BP. And the red line and red dashed line indicate the training and testing accuracies with AL.

3.3 Metafeature Visualization and Quantification

To determine whether the hidden layers truly learn useful metafeatures when using Associated Learning, we used t-SNE [20] to visualize the 2^{nd}, 4^{th} hidden layers and output layer in the 6-layer MLP model and the 4^{th}, 8^{th},

and 12^{th} hidden layers in the 14-layer CNN model on MNIST and CIFAR-10 dataset. For comparison purposes, we also visualize the corresponding hidden layers trained using backpropagation. As shown in Figure 3.6, Figure 3.7, Figure 3.8 and Figure 3.9, the initial layers seem to extract less useful metafeatures, because the labels are difficult to distinguish in the corresponding figures. However, a comparison of the last few layers shows that Associated Learning groups the data points of the same label more accurately, which suggests that Associated Learning likely learns better metafeatures.

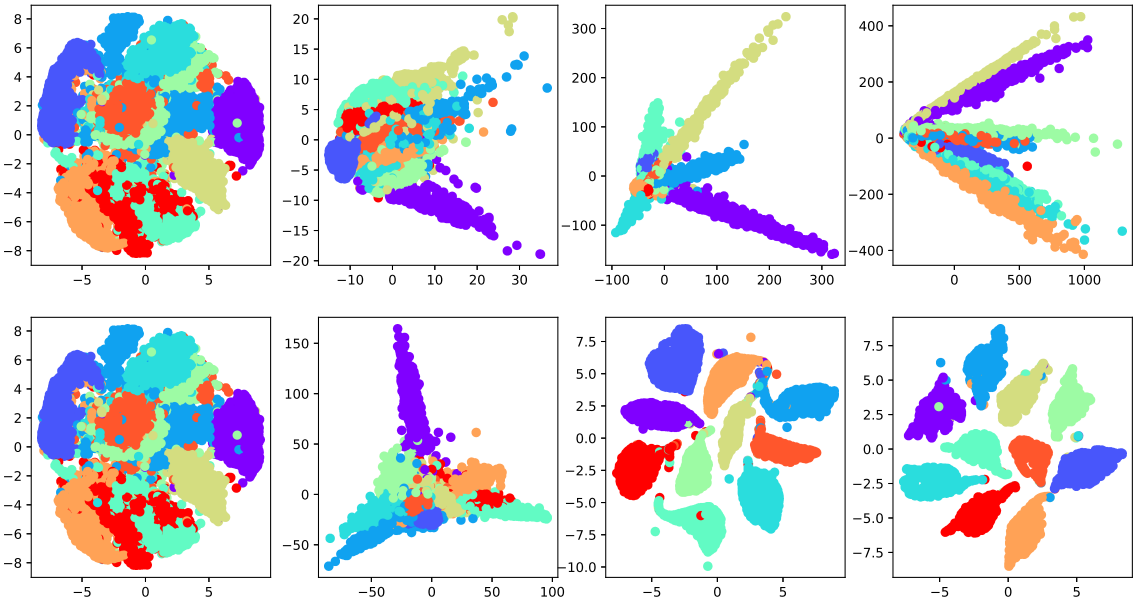

Figure 3.6: t-SNE visualization of MLP with MNIST dataset. The different colors represent different labels. The figures in the first row are the results of the raw data, 2^{nd} layer, 4^{th} layer, and output layer when using BP. The second row shows the corresponding results for AL.

To assess the quality of the learned metafeatures, we calculated the intra- and inter-class distances of the data points based on the metafeatures. To compute the intra-class distance, we first calculated d_k^{intra}, the average distance between any two data points in class k for each class.The

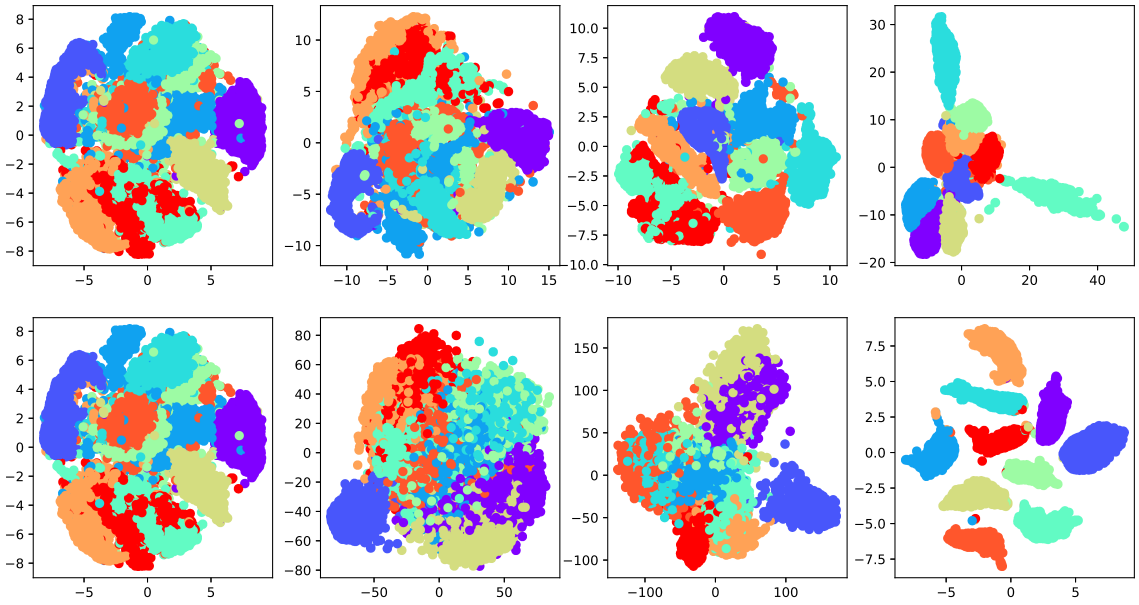

Figure 3.7: t-SNE visualization of CNN with MNIST dataset. The different colors represent different labels. The figures in the first row are the results of the raw data, 4^{th} layer, 8^{th} layer and 12^{th} layers when using BP. The second row shows the corresponding results for AL

inter-class distance is the average distance between the centroids of the classes. We also computed the ratio between intra- and inter-class distance to determine the quality of the metafeatures generated by Associated Learning and backpropagation [25, 26]. As shown in Table 3.4, Associated Learning outperforms backpropagation on both the CIFAR-10 and CIFAR-100 datasets.

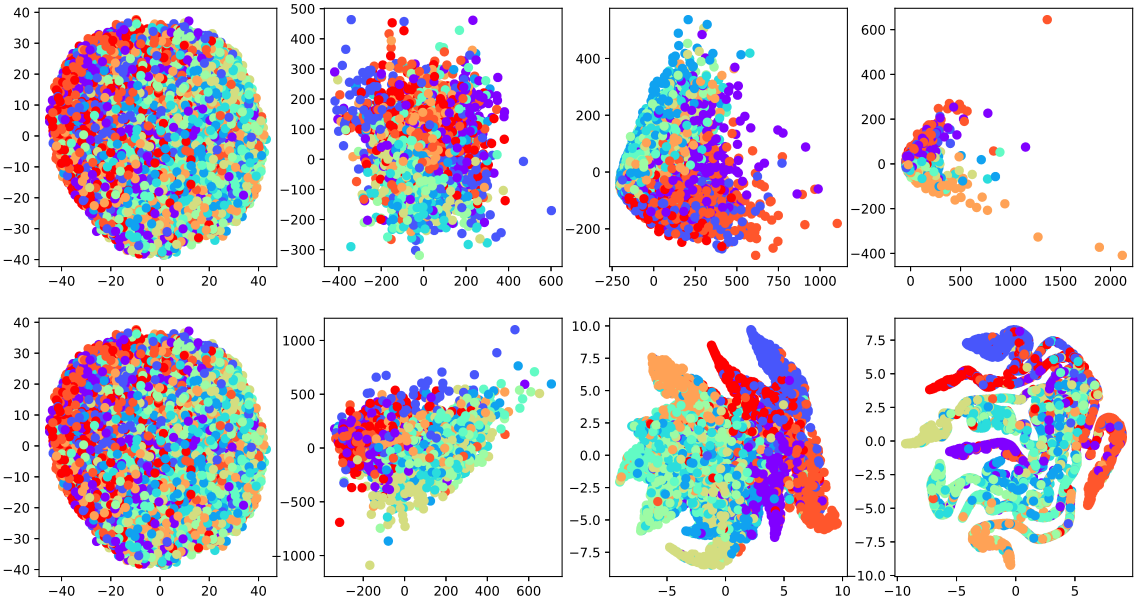

Figure 3.8: t-SNE visualization of MLP with CIFAR-10 dataset. The different colors represent different labels. The figures in the first row are the results of the raw data, 2^{nd} layer, 4^{th} layer, and output layer when using BP. The second row shows the corresponding results for AL.

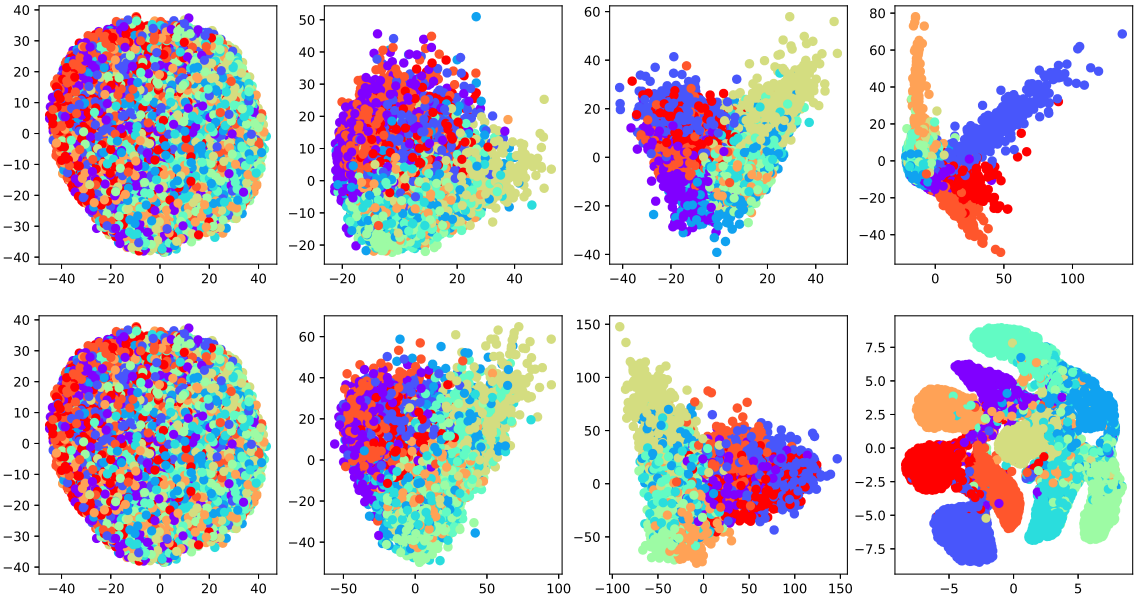

Figure 3.9: t-SNE visualization of CNN with CIFAR-10 dataset. The different colors represent different labels. The figures in the first row are the results of the raw data, 4^{th} layer, 8^{th} layer and 12^{th} layers when using BP. The second row shows the corresponding results for AL.

Table 3.4: A comparison of the inter- and intra-class distances, and the ratio of the two. The best performances are highlighted in bold font.

Datasets	Models	Methods	Interclass distance	Intraclass distance	Inter:Intra ratio
CIFAR-10	MLP	BP	39.36	67.97	0.58
		AL	0.73	0.66	**1.11**
	CNN	BP	41.82	26, 87	1.56
		AL	1.17	0.36	**3.25**
CIFAR-100	MLP	BP	114.42	342.65	0.33
		AL	0.23	0.28	**0.82**
	CNN	BP	114.71	163.43	0.70
		AL	0.55	0.51	**1.08**

Chapter 4

Related Work

Backpropagation [1] is an essential algorithm for training deep neural networks and is the foundation of the success of many models in recent decades [5, 22, 6]. However, because of "backward lock" (i.e., the weights must be updated layer by layer), training a deep neural network can be extremely inefficient [2]. In addition, empirical evidence shows that backpropagation is biologically implausible [9, 10, 27]. Thus, many studies have suggested replacing backpropagation by more biologically plausible methods or by gradient-free methods [28] in the hope of decreasing the computational time and memory consumption [27, 29, 30].

To address the backward lock problem, the authors of [2] proposed using a synthetic gradient, which is an estimation of the real gradient generated by a separate neural network for each layer. The structure is shown in Figure 4.1. By adopting the synthetic gradient as the true gradient, the parameters of every layer can be updated simultaneously and independently. This approach eliminates the backward lock problem. However, the experimental results have shown that this approach tends to underfit —probably because the gradients are difficult to predict.

It is also possible to eliminate backward lock by computing the local

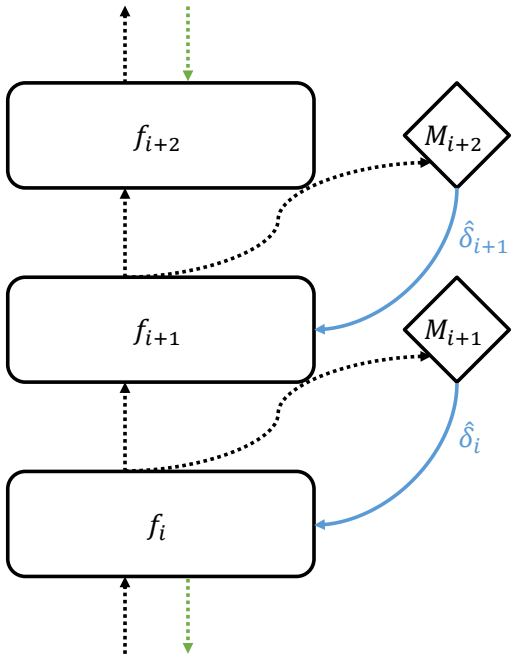

Figure 4.1: The structure of synthetic gradient, where f_i indicates sub-network and M_i indicates the gradient generator of layer-i. The black dotted arrows denote the data flow and the blue arrows denote the synthetic gradient flow.

errors for the different components of a network. In [31], every layer in a deep neural network is trained by a local classifier. However, experimental results show that this type of model is not comparable with backpropagation. The authors of [16] mixed the idea of local errors and similarity measurement [32, 33, 34]. Their experimental results show that this technique improves testing accuracy; however, this design requires each local component to receive signals directly from the target variable to allow the calculation of both the similarity matching loss and the prediction loss. Biologically, it is unlikely that neurons distant from the target would be able to access the target signal directly. Therefore, even though these methods do not require global backpropagation, they may still be biologically implausible.

Feedback alignment [11] suggests propagating error signals similar to backpropagation, but the error signals are propagated with fixed random

weights in every layer. Later, the authors in [12] suggested delivering error signals directly from the output layer using fixed weights. The result is that the gradients are propagated by weights, while the signals remain local to each layer. Figure 4.2 demonstrates the difference between feedback alignment and direct feedback alignment. The problem with this approach is similar to the problem discussed in the preceding paragraph: biologically, distant neurons are unlikely to be able to obtain signals directly from the target variable.

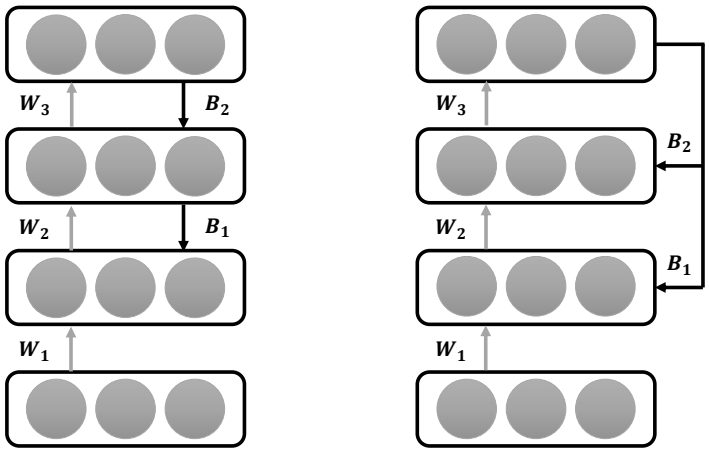

Figure 4.2: The structure of FA(left) and DFA(right). The gray arrows indicate data flow and the black arrows indicate gradient flow. W_is are trainable weight whereas B_i are fixed random weights.

Yet another biologically motivated algorithm is target propagation [17, 18, 19, 13, 14, 15]. Rather than computing the gradient for every layer, the target propagation computes the target that each layer should learn. This approach relies on an autoencoder [35] to compute the inverse mapping of the forward pass and then pass the ground truth information to every layer. Each training step includes two losses that must be minimized for

each layer: the loss of inverse mapping and the loss between activations and targets. However, this learning method alleviates the need for symmetric weights and is both biologically plausible and more robust than backpropagation when applied to stochastic networks. Nonetheless, the targets are still generated layer by layer.

Overviews of the biologically plausible (or at least partially plausible) methods are presented in [27, 15]. Although most of these methods perform worse than does conventional backpropagation, optimization beyond backpropagation is still an active research area, largely for computational efficiency and biological compatibility reasons.

Our work is highly motivated by target propagation. However, we create intermediate mappings instead of directly transforming features into targets. As a result, the local signals in each layer are independent of the signals in other layers, and most of these signals are not obtained directly from the output label.

Chapter 5

Discussion and future works

This paper discusses Associated Learning, a novel process for training deep neural networks. Rather than calculating gradients in a layerwise fashion based on backpropagation, Associated Learning removes the dependencies between the parameters of different subnetworks; thus allowing each subnetwork to be trained simultaneously and independently. Moreover, our method is also biologically plausible because the targets are local, and the gradients are not obtained from the output layer. Additionally, we observed that the metafeatures generated by Associated Learning seem to be better than those generated by backpropagation. Although Associated Learning does not directly minimize the prediction error, its test accuracy is comparable to that backpropagation, which does directly attempt to minimize the prediction error. Although recent studies have started using local losses instead of backpropagating the global loss [16], these local losses are largely computed based on (or at least partially based on) the difference between the target variable and the prediction. Our method is unique because in Associated Learning, most of the layers do not interact with the target variable.

One aspect of our current work involves validating Associated Learning

on other datasets (e.g., ImageNet, MS COCO, and Google' s Open Images). We are also interested in validating Associated Learning on datasets beyond computer vision, such as those used in signal processing, natural language processing, recommender systems, and so on. Meanwhile, we are investigating strategies to better fit s_i to y_i with fewer parameters. In the longer term, we are highly interested in investigating optimization algorithms beyond backpropagation and gradients. Additionally, even though we expect AL to increase the throughput of the training phase through the pipeline strategy, we still need empirical validation, so integrating AL with pipeline is also one of the future work.

Bibliography

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, p. 533, 1986.

[2] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," *arXiv preprint arXiv:1608.05343*, 2016.

[3] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[4] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[8] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, pp. 1310–1318, 2013.

[9] F. Crick, "The recent excitement about neural networks.," *Nature*, vol. 337, no. 6203, pp. 129–132, 1989.

[10] D. Balduzzi, H. Vanchinathan, and J. M. Buhmann, "Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks.," in *AAAI*, pp. 485–491, 2015.

[11] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Communications*, vol. 7, p. 13276, 2016.

[12] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 1037–1045, 2016.

[13] A. G. Ororbia, A. Mali, D. Kifer, and C. L. Giles, "Conducting credit assignment by aligning local representations," *arXiv preprint arXiv:1803.01834*, 2018.

[14] A. G. Ororbia and A. Mali, "Biologically motivated algorithms for propagating local target representations," *arXiv preprint arXiv:1805.11703*, 2018.

[15] S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," in *Advances in Neural Information Processing Systems*, pp. 9390–9400, 2018.

[16] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," *arXiv preprint arXiv:1901.06656*, 2019.

[17] D.-H. Lee, S. Zhang, A. Biard, and Y. Bengio, "Target propagation," 12 2014.

[18] Y. Bengio, "How auto-encoders could provide credit assignment in deep networks via target propagation," *arXiv preprint arXiv:1407.7906*, 2014.

[19] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 498–515, Springer, 2015.

[20] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[24] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, pp. 3856–3866, 2017.

[25] M. Michael and W.-C. Lin, "Experimental study of information measure and inter-intra class distance ratios on feature selection and orderings," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 2, pp. 172–181, 1973.

[26] Y. Luo, Y. Wong, M. Kankanhalli, and Q. Zhao, "G-softmax: Improving intraclass compactness and interclass separability of features," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[27] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, "Towards biologically plausible deep learning," *arXiv preprint arXiv:1502.04156*, 2015.

[28] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable admm approach," in *International Conference on Machine Learning*, pp. 2722–2731, 2016.

[29] Z. Huo, B. Gu, Q. Yang, and H. Huang, "Decoupled parallel backpropagation with convergence guarantee," *arXiv preprint arXiv:1804.10574*, 2018.

[30] Z. Huo, B. Gu, and H. Huang, "Training neural networks using features replay," in *Advances in Neural Information Processing Systems*, pp. 6659–6668, 2018.

[31] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers in Neuroscience*, vol. 12, p. 608, 2018.

[32] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, p. 788, 1999.

[33] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," in *Proceedings of the 24th International Conference on Machine Learning*, pp. 759–766, ACM, 2007.

[34] A. Coates and A. Y. Ng, "Selecting receptive fields in deep networks," in *Advances in Neural Information Processing Systems*, pp. 2528–2536, 2011.

[35] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 37–49, 2012.

Appendix A

Source Code

A.1 Code link

Github link: `https://github.com/SamYWK/Associated_Learning`

A.2 Usage

Specifying the model you want to test with –model flag and run the following:

Code A.1: .

```
python main.py --model <MODEL>
```

For MNIST dataset, there are 4 models you can choose:

- MLP
- MLP_AL
- CNN
- CNN_AL

For CIFAR-10 and CIFAR-100, there are 10 models you can choose:

- MLP
- MLP_AL
- CNN
- CNN_AL
- ResNet_20
- ResNet_20_AL
- ResNet_32
- ResNet_32_AL
- VGG
- VGG_AL