MATLAB與深度學習

深度學習是目前人工智慧的主流，透過MATLAB，我們可以簡單且容易的實現深度學習相關的應用。

前言

“深度學習”(Deep Learning)是機器學習範疇中的分支，其根源為類神經網絡(Artificial Neural Net work)，利用疊加多層的網絡架構來處理複雜的非線性問題。在這樣的架構中，深度學習的網絡可以從資料當中取出特徵(feature)，並得到最後的結果，有時該結果甚至可以超越人類辨識準確度。近期G oogle以AlphaGo人工智慧圍棋程式擊敗排名頂尖的世界棋手、IBM Watson的益智問答比賽等，都是讓人為之驚豔的應用。除了這些特殊應用外，深度學習也被廣為應用在影像的分類、語音的辨識等，可以說是目前人工智慧的主流，MATLABR也納入了深度學習的相關功能。

深度學習模型的訓練 (Training from Scratch)

從以上的影片示範，我們可以知道要能辨識影像需要事先訓練深度學習模型。

深度學習模型的準確性在很大程度上取決於用於訓練模型的數據量。準確的模型需要數千甚至數百萬個樣本，這可能需要很

長時間才能訓練完成。一旦模型被訓練完成，它可以用於即時應用，例如先進駕駛輔助系統（Advanc ed Driver Assistance Systems; ADAS）中的行人偵測。

Create Simple Deep Learning Network for Classification

This example shows how to create and train a simple convolutional neural network for deep learning classification. Convolutional neural networks are essential tools for deep learning, and are especially suited for image recognition.

The example demonstrates how to:

- Load and explore image data.
- Define the network architecture.
- Specify training options.
- Train the network.
- Predict the labels of new data and calculate the classification accuracy.

For an example showing how to interactively create and train a simple image classification network, see Create Simple Image Classification Network Using Deep Network Designer.

Load and Explore Image Data

Load the digit sample data as an image datastore. `imageDatastore` automatically labels the images based on folder names and stores the data as an `ImageDatastore` object. An image datastore enables you to store large image data, including data that does not fit in memory, and efficiently read batches of images during training of a convolutional neural network.

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

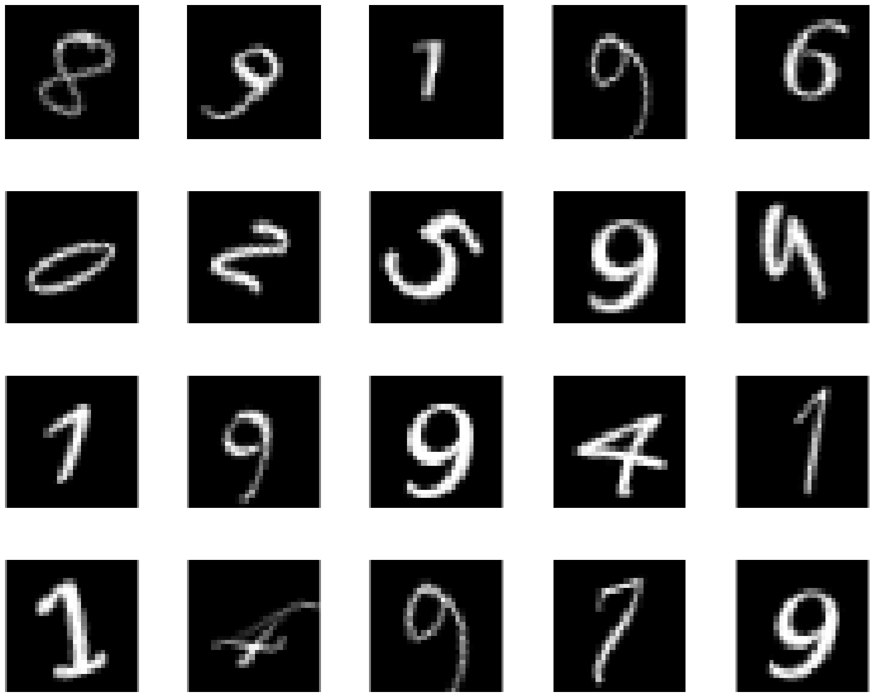
Display some of the images in the datastore.

```
figure;
perm = randperm(10000,20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end
```

Calculate the number of images in each category. `labelCount` is a table that contains the labels and the number of images having each label. The datastore contains 1000 images for each of the digits 0-9, for a total of 10000 images. You can specify the number of classes in the last fully connected layer of your network as the `OutputSize` argument.

```
labelCount = countEachLabel(imds)
```

You must specify the size of the images in the input layer of the network. Check the size of the first image in `digitData`. Each image is 28-by-28-by-1 pixels.



```
img = readimage(imds,1);
size(img)
```

Specify Training and Validation Sets

Divide the data into training and validation data sets, so that each category in the training set contains 750 images, and the validation set contains the remaining images from each label. `splitEachLabel` splits the datastore `digitData` into two new datastores, `trainDigitData` and `valDigitData`.

```
numTrainFiles = 750;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

Define Network Architecture

Define the convolutional neural network architecture.

```
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

Image Input Layer An imageInputLayer is where you specify the image size, which, in this case, is 28-by-28-by-1. These numbers correspond to the height, width, and the channel size. The digit data consists of grayscale images, so the channel size (color channel) is 1. For a color image, the channel size is 3, corresponding to the RGB values. You do not need to shuffle the data because `trainNetwork`, by default, shuffles the data at the beginning of training. `trainNetwork` can also automatically shuffle the data at the beginning of every epoch during training.

Convolutional Layer In the convolutional layer, the first argument is `filterSize`, which is the height and width of the filters the training function uses while scanning along the images. In this example, the number 3 indicates that the filter size is 3-by-3. You can specify different sizes for the height and width of the filter. The second argument is the number of filters, `numFilters`, which is the number of neurons that connect to the same region of the input. This parameter determines the number of feature maps. Use the `'Padding'` name-value pair to add padding to the input feature map. For a convolutional layer with a default stride of 1, `'same'` padding ensures that the spatial output size is the same as the input size. You can also define the stride and learning rates for this layer using name-value pair arguments of convolution2dLayer.

Batch Normalization Layer Batch normalization layers normalize the activations and gradients propagating through a network, making network training an easier optimization problem. Use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers, to speed up network training and reduce the sensitivity to network initialization. Use batchNormalizationLayer to create a batch normalization layer.

ReLU Layer The batch normalization layer is followed by a nonlinear activation function. The most common activation function is the rectified linear unit (ReLU). Use reluLayer to create a ReLU layer.

Max Pooling Layer Convolutional layers (with activation functions) are sometimes followed by a down-sampling operation that reduces the spatial size of the feature map and removes redundant spatial information. Down-sampling makes it possible to increase the number of filters in deeper convolutional layers without increasing the required amount of computation per layer. One way of down-sampling is using a max pooling, which you create using maxPooling2dLayer. The max pooling layer returns the maximum values of rectangular regions of inputs, specified by the first argument, `poolSize`. In this example, the size of the rectangular region is [2,2]. The `'Stride'` name-value pair argument specifies the step size that the training function takes as it scans along the input.

Fully Connected Layer The convolutional and down-sampling layers are followed by one or more fully connected layers. As its name suggests, a fully connected layer is a layer in which the neurons connect to all the neurons in the preceding layer. This layer combines all the features learned by the previous layers across the image to identify the larger patterns. The last fully connected layer combines the features to classify the images. Therefore, the `OutputSize` parameter in the last fully connected layer is equal to the number of classes in the target data. In this example, the output size is 10, corresponding to the 10 classes. Use fullyConnectedLayer to create a fully connected layer.

Softmax Layer The softmax activation function normalizes the output of the fully connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer. Create a softmax layer using the softmaxLayer function after the last fully connected layer.

Classification Layer The final layer is the classification layer. This layer uses the probabilities returned by the softmax activation function for each input to assign the input to one of the mutually exclusive classes and compute the loss. To create a classification layer, use classificationLayer.

Specify Training Options

After defining the network structure, specify the training options. Train the network using stochastic gradient descent with momentum (SGDM) with an initial learning rate of 0.01. Set the maximum number of epochs to 4. An epoch is a full training cycle on the entire training data set. Monitor the network accuracy during training by specifying validation data and validation frequency. Shuffle the data every epoch. The software trains the network on the training data and calculates the accuracy on the validation data at regular intervals during training. The validation data is not used to update the network weights. Turn on the training progress plot, and turn off the command window output.

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

Train Network Using Training Data

Train the network using the architecture defined by `layers`, the training data, and the training options. By default, `trainNetwork` uses a GPU if one is available, otherwise, it uses a CPU. Training on a GPU requires Parallel Computing Toolbox™ and a supported GPU device. For information on supported devices, see GPU Support by Release. You can also specify the execution environment by using the `'ExecutionEnvironment'` name-value pair argument of `trainingOptions`.

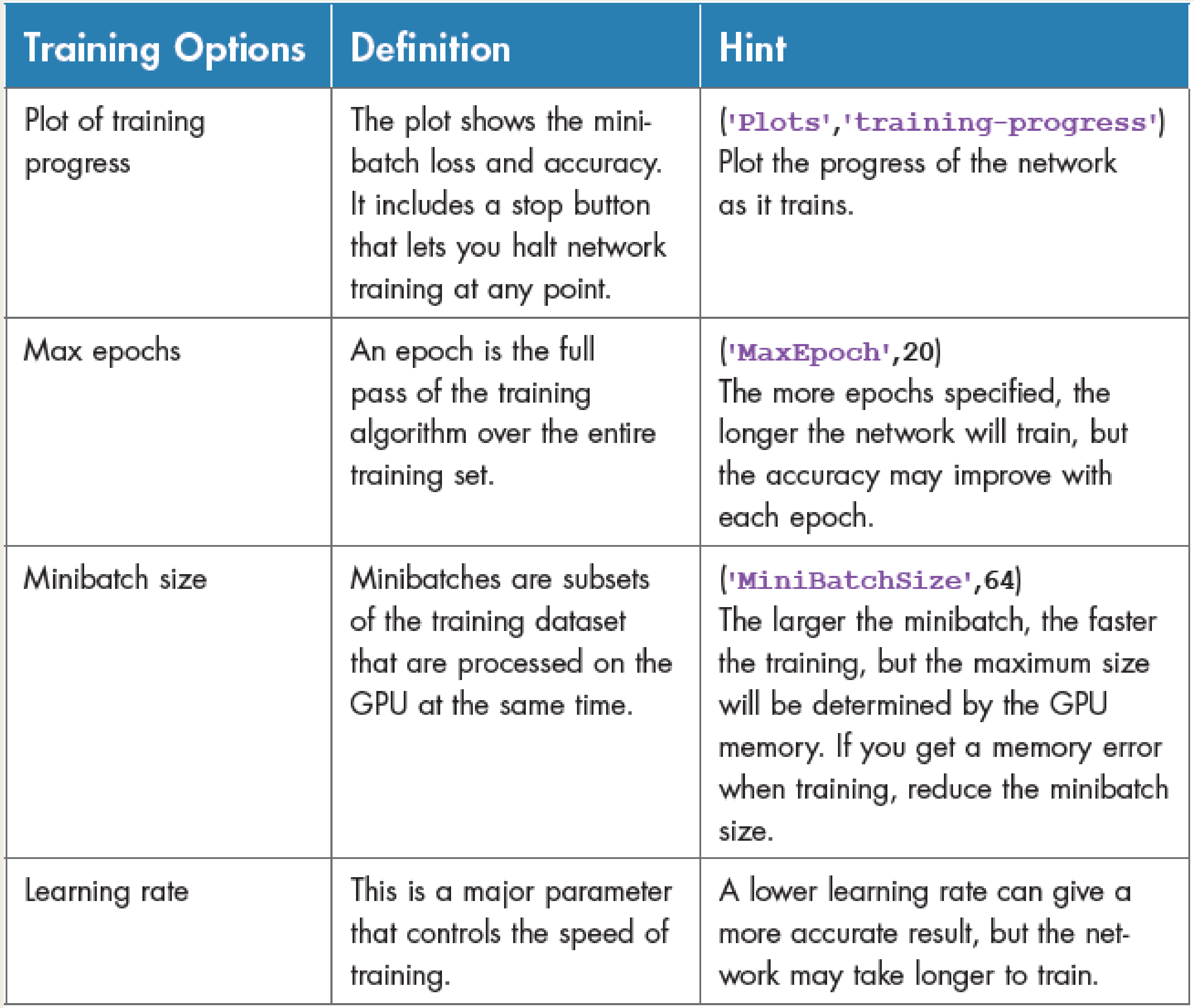
The training progress plot shows the mini-batch loss and accuracy and the validation loss and accuracy. For more information on the training progress plot, see Monitor Deep Learning Training Progress. The loss is the cross-entropy loss. The accuracy is the percentage of images that the network classifies correctly.

```
net = trainNetwork(imdsTrain,layers,options);
```

Classify Validation Images and Compute Accuracy

Predict the labels of the validation data using the trained network, and calculate the final validation accuracy. Accuracy is the fraction of labels that the network predicts correctly. In this case, more than 99% of the predicted labels match the true labels of the validation set.

```
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation)
```

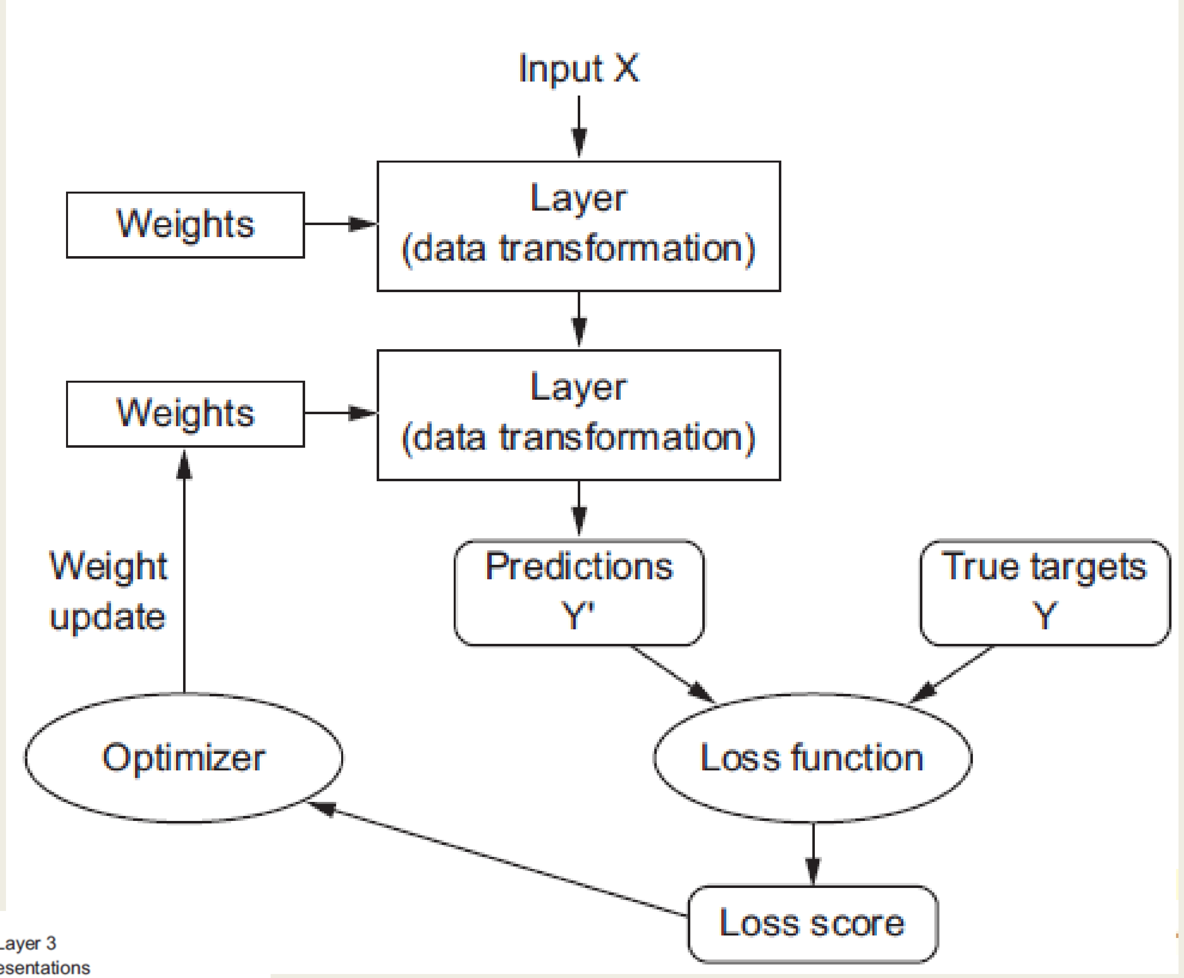


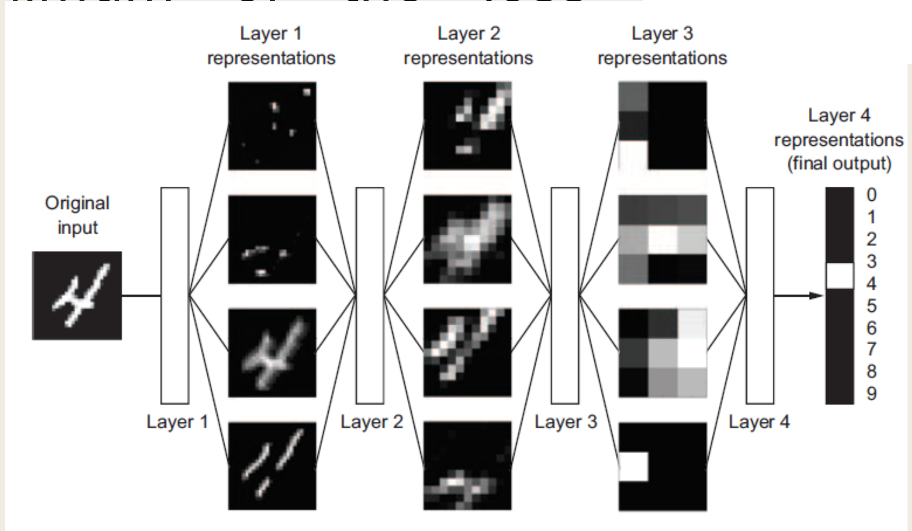
Training Options	Definition	Hint
Plot of training progress	The plot shows the mini-batch loss and accuracy. It includes a stop button that lets you halt network training at any point.	(`'Plots','training-progress'`) Plot the progress of the network as it trains.
Max epochs	An epoch is the full pass of the training algorithm over the entire training set.	(`'MaxEpoch',20`) The more epochs specified, the longer the network will train, but the accuracy may improve with each epoch.
Minibatch size	Minibatches are subsets of the training dataset that are processed on the GPU at the same time.	(`'MiniBatchSize',64`) The larger the minibatch, the faster the training, but the maximum size will be determined by the GPU memory. If you get a memory error when training, reduce the minibatch size.
Learning rate	This is a major parameter that controls the speed of training.	A lower learning rate can give a more accurate result, but the network may take longer to train.

Training process 1. Define the loss function.

2. 求 梯 度 : Evaluate the gradient function through backpropagation algorithm for each weight & bias. (i.e. to change the weight & bias, it will lead what change for the loss function)

3. 梯度法 : In each batch, change the weight & bias to minimize the loss function in the gradient direction to achieve the minimum of the loss function.





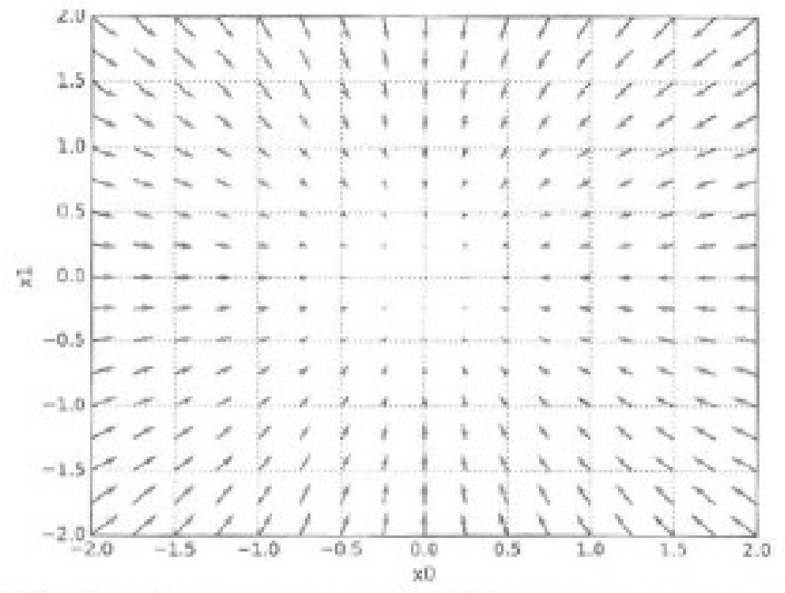
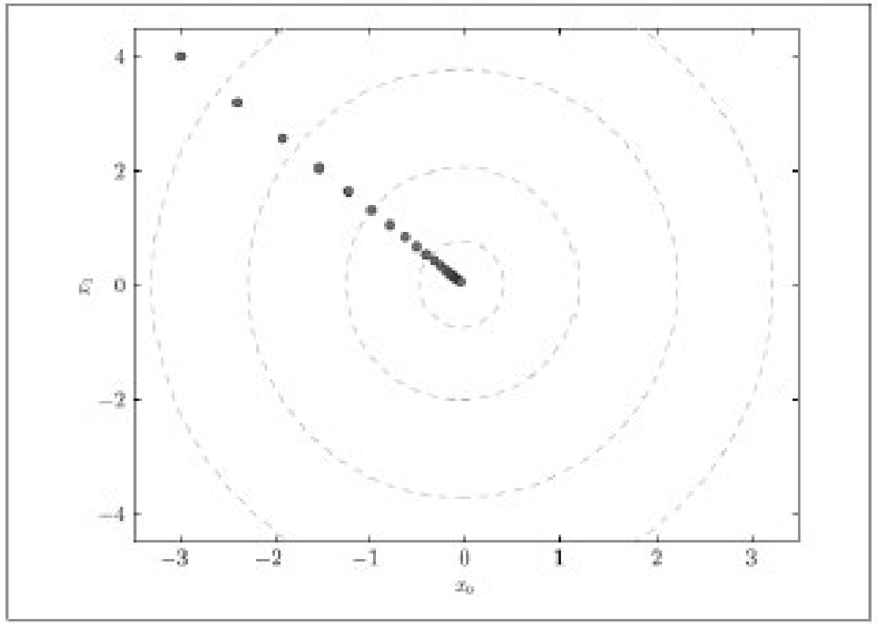


图 4-9 $f(x_0$



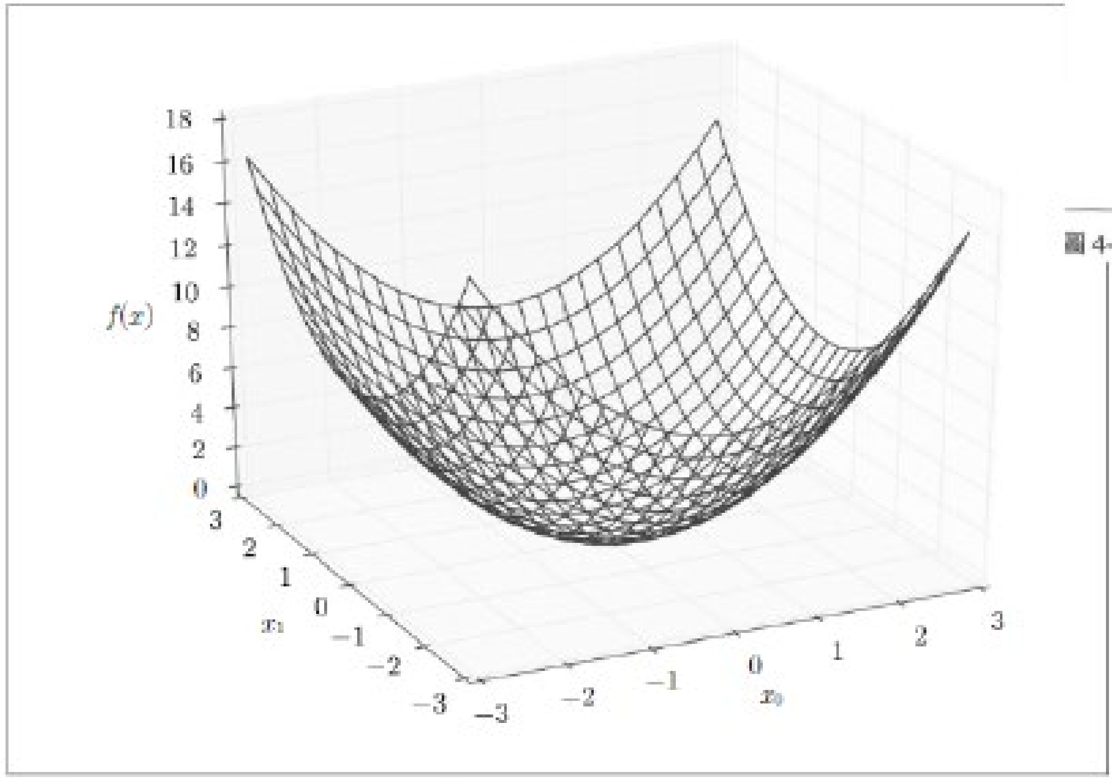


图4-8 $f(x_0, x_1) = x_0^2 + x_1^2$ 的图像

訓練神經網路，預測數據並計算準確度

- 使用訓練用資料訓練神經網路

```
net = trainNetwork(imdsTrain,layers,options);
```

- 預測數據並計算及顯示每批次的準確度

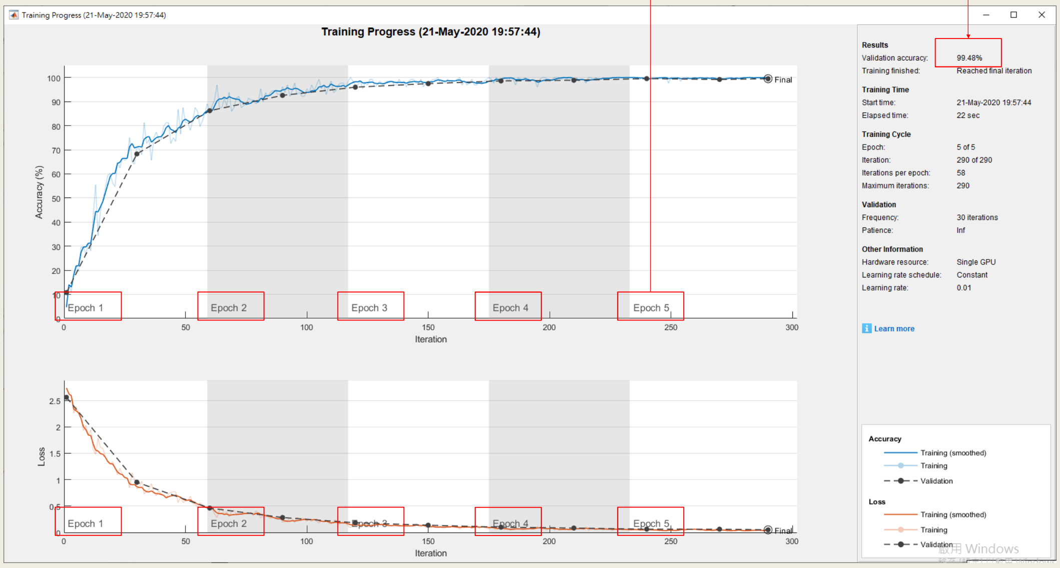
```
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation)
```

```
accuracy = 0.9988
```

結果

訓練批次

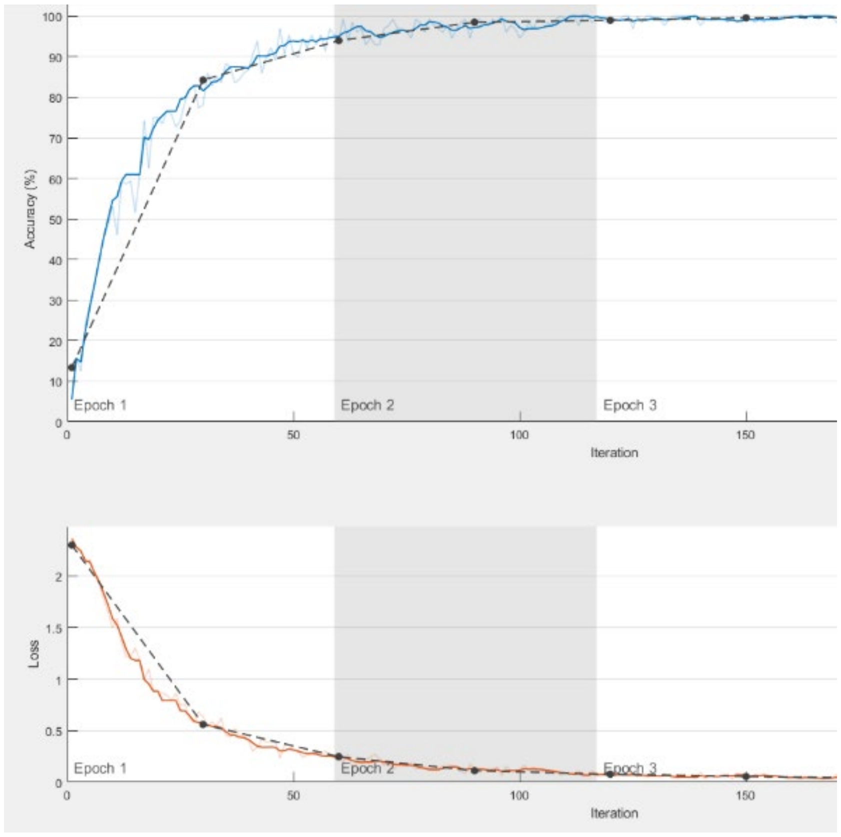
準確度：99.48%



Results	
Validation accuracy:	99.48%
Training finished:	Reached final iteration
Training Time	
Start time:	21-May-2020 19:57:44
Elapsed time:	22 sec
Training Cycle	
Epoch:	5 of 5
Iteration:	290 of 290
Iterations per epoch:	58
Maximum iterations:	290
Validation	
Frequency:	30 iterations
Patience:	Inf
Other Information	
Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.01

請參閱各版本的 GPU 支持。 您還可以使用 trainingOptions 的 “ExecutionEnvironment” 名稱-值對參數指定執行環境。

```
net = trainNetwork(imdsTrain,layers,options);
```



Classify Validation Images and Compute Accuracy

使用訓練好的網絡預測驗證數據的標籤，併計算最終的驗證準確率。 準確性是網絡正確預測的標籤的分數。 在這種情況下，超過 99% 的預測標籤與驗證集的真實標籤匹配。

```
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;
accuracy = sum(YPred == YValidation)/numel(YValidation)
```

```
accuracy = 0.9988
```

2. Create Simple Deep Learning Network for Classification

載入 MNIST 訓練資料

```
[XTrain,YTrain] = digitTrain4DArrayData;
```

建立卷積神經網路

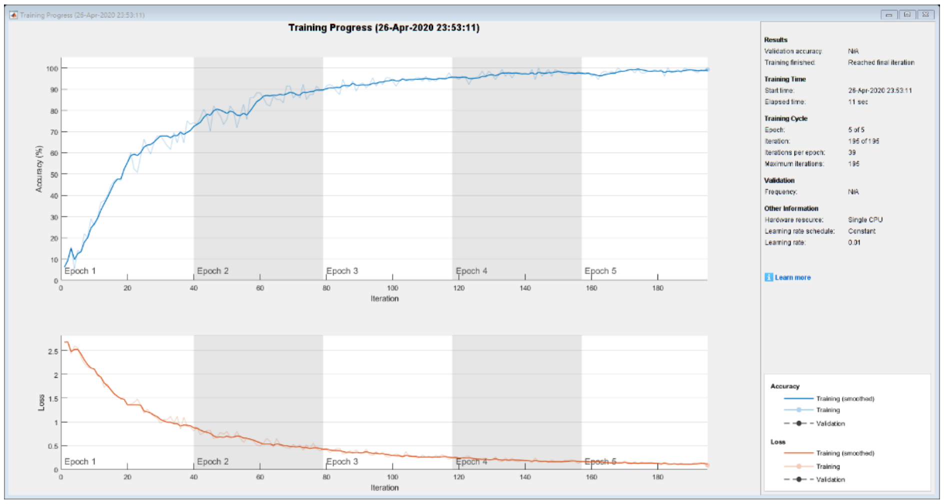
```
layers = [

    imageInputLayer([28 28 1]) %輸入大小為 28*28*1

    convolution2dLayer(3,8,'Padding','same') %卷積核大小為 3*3，其數量為 8，卷積後
大小不變
    batchNormalizationLayer %批次正規化，避免過度擬合
    reluLayer
    convolution2dLayer(3,16,'Padding','same','Stride',2)
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,32,'Padding','same','Stride',2)
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

設置訓練選項

```
options = trainingOptions('sgdm',... %優化器為具有動量的隨機梯度下降
    'MaxEpochs',5,... %最大訓練回合數
    'Verbose',false,... %是否顯示訓練資訊
    'Plots','training-progress',... %是否畫出訓練過程
    "ExecutionEnvironment","cpu");
```

訓練網路

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



Results
Validation accuracy: N/A
Training finished: Reached final iteration

Training Time
Start time: 26-Apr-2020 23:53:11
Elapsed time: 11 sec

Training Cycle
Epoch: 5 of 5
Iteration: 195 of 195
Iterations per epoch: 39
Maximum iterations: 195

Validation
Frequency: N/A

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.01

載入 MNIST 測試資料

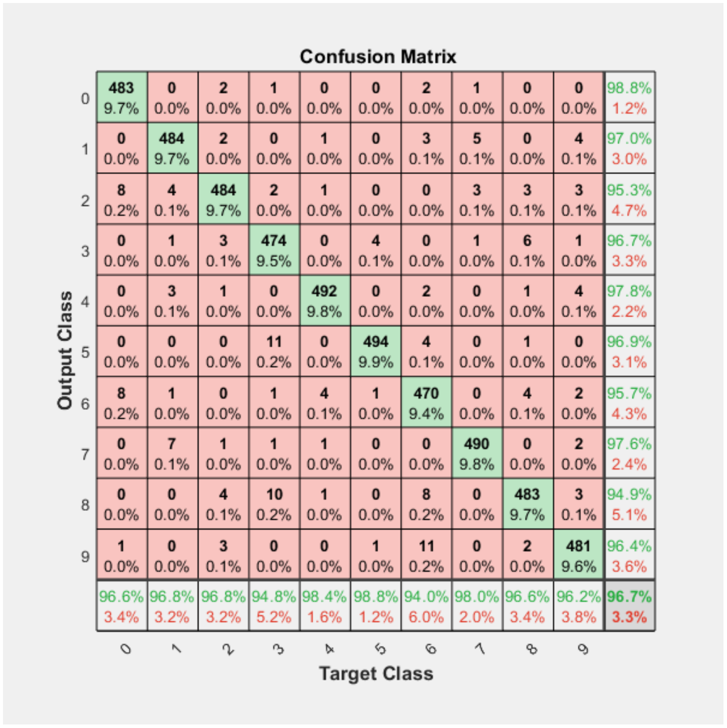
```
[XTest,YTest] = digitTest4DArrayData;
```

分類 MNIST 測試資料

```
YPredicted = classify(net,XTest);
```

顯示混淆矩陣

```
plotconfusion(YTest,YPredicted)
```



Confusion Matrix

Output Class \ Target Class	0	1	2	3	4	5	6	7	8	9	
0	483 9.7%	0 0.0%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	98.8% 1.2%
1	0 0.0%	484 9.7%	2 0.0%	0 0.0%	1 0.0%	0 0.0%	3 0.1%	5 0.1%	0 0.0%	4 0.1%	97.0% 3.0%
2	8 0.2%	4 0.1%	484 9.7%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	3 0.1%	3 0.1%	3 0.1%	95.3% 4.7%
3	0 0.0%	1 0.0%	3 0.1%	474 9.5%	0 0.0%	4 0.1%	0 0.0%	1 0.0%	6 0.1%	1 0.0%	96.7% 3.3%
4	0 0.0%	3 0.1%	1 0.0%	0 0.0%	492 9.8%	0 0.0%	2 0.0%	0 0.0%	1 0.0%	4 0.1%	97.8% 2.2%
5	0 0.0%	0 0.0%	0 0.0%	11 0.2%	0 0.0%	494 9.9%	4 0.1%	0 0.0%	1 0.0%	0 0.0%	96.9% 3.1%
6	8 0.2%	1 0.0%	0 0.0%	1 0.0%	4 0.1%	1 0.0%	470 9.4%	0 0.0%	4 0.1%	2 0.0%	95.7% 4.3%
7	0 0.0%	7 0.1%	1 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	490 9.8%	0 0.0%	2 0.0%	97.6% 2.4%
8	0 0.0%	0 0.0%	4 0.1%	10 0.2%	1 0.0%	0 0.0%	8 0.2%	0 0.0%	483 9.7%	3 0.1%	94.9% 5.1%
9	1 0.0%	0 0.0%	3 0.1%	0 0.0%	0 0.0%	1 0.0%	11 0.2%	0 0.0%	2 0.0%	481 9.6%	96.4% 3.6%
	96.6% 3.4%	96.8% 3.2%	96.8% 3.2%	94.8% 5.2%	98.4% 1.6%	98.8% 1.2%	94.0% 6.0%	98.0% 2.0%	96.6% 3.4%	96.2% 3.8%	96.7% 3.3%