Artificial Neural Networks (ANN): The Past, Present, and Future of Deep Learning

Student Name: Wangjun SHEN

Student ID:110248810

School: The University of South Australia

Abstraction

This article reviews the history of artificial neural networks (ANN), from its early concepts to today's deep learning craze. We explore the basic structure, training methods, and various types of ANNs, including feedforward neural networks, convolutional neural networks, and recurrent neural networks. By discussing in detail examples of ANN in practical applications, such as image recognition and speech recognition, we highlight its role in data science and machine learning. Additionally, we discuss the advantages and limitations of ANN and how it compares to other machine learning techniques. Finally, we propose possible directions for future deep learning research and how to better apply ANN in practical scenarios. This article aims to provide researchers and practitioners with a comprehensive understanding of the history, current status, and future potential of ANNs.

Introduction

Artificial Neural Networks (ANN), as a computing model inspired by biological nervous systems, has attracted widespread attention and research in the past decades. The origin of ANN can be traced back to the 1940s, when Warren McCulloch and Walter Pitts proposed a simple mathematical model to simulate the information transfer process between neurons [1]. This early model laid the foundation for ANN research and inspired subsequent development.

An ANN model consists of multiple neurons that are connected to each other and transmit information through weights. Each neuron can be viewed as a nonlinear processing unit that can learn and adapt to various complex tasks. Over the past few decades, ANN has experienced several important technological advances, including the introduction of the backpropagation algorithm [2], which enables ANN to train deeper and more complex networks, further improving its performance.

Today, ANN has become one of the key tools in the field of data science, machine learning and artificial intelligence. It has achieved impressive achievements in tasks such as image recognition, natural language processing, and speech recognition [3]. This widespread application is driven by growing computing power, the availability of large-scale data sets, and research investments in deep learning methods.

In academia, the research and development of ANN has also attracted much attention. More and more researchers are exploring the potential of ANN and conducting various experiments and research to promote its further development [4]. ANN research has transcended traditional computing models, triggered interdisciplinary research collaboration, and promoted innovation in multiple fields.

This article aims to review the past, present and future of ANN and explore its key role in data science and deep learning. We will discuss in detail the basic structure, training methods and various types of ANN, as well as its performance in practical applications. Additionally, we will explore the advantages and limitations of ANN and how it compares to other machine learning techniques. Finally, we will propose possible directions for future deep learning research and how to better apply ANN in practical scenarios.

Analysis Method

From AI to ANN

The history of artificial intelligence (AI) can be traced back to the middle of the last century, when the definition of AI covered multiple aspects, including the simulation of human behavior, logic-based reasoning, rational decision-making, and behavioral performance. The four main definitions are "Thinking Humanly" (thinking like humans), "Thinking Rationally" (reasonable thinking), "Acting Humanly" (acting like humans) and "Acting Rationally" (reasonable action) [5]. In order to achieve different definitions of AI, various AI schools have emerged, including "Symbolic AI", "Probabilistic AI" and "Connectionism".

Symbolist AI focuses on using symbols and symbolic reasoning to simulate human intelligence [6]. It centers on rules and knowledge representation, emphasizing logic and reasoning processes. Probabilistic AI uses probabilistic models to deal with uncertainty and randomness [7]. This school emphasizes probabilistic reasoning and statistical methods, often used to deal with complex real-world problems. Connectionism is the theoretical basis of artificial neural networks (ANN), emphasizing the connections and learning processes between neurons in the simulated nervous system [2]. This school emphasizes data-based learning and simulation of neural networks.

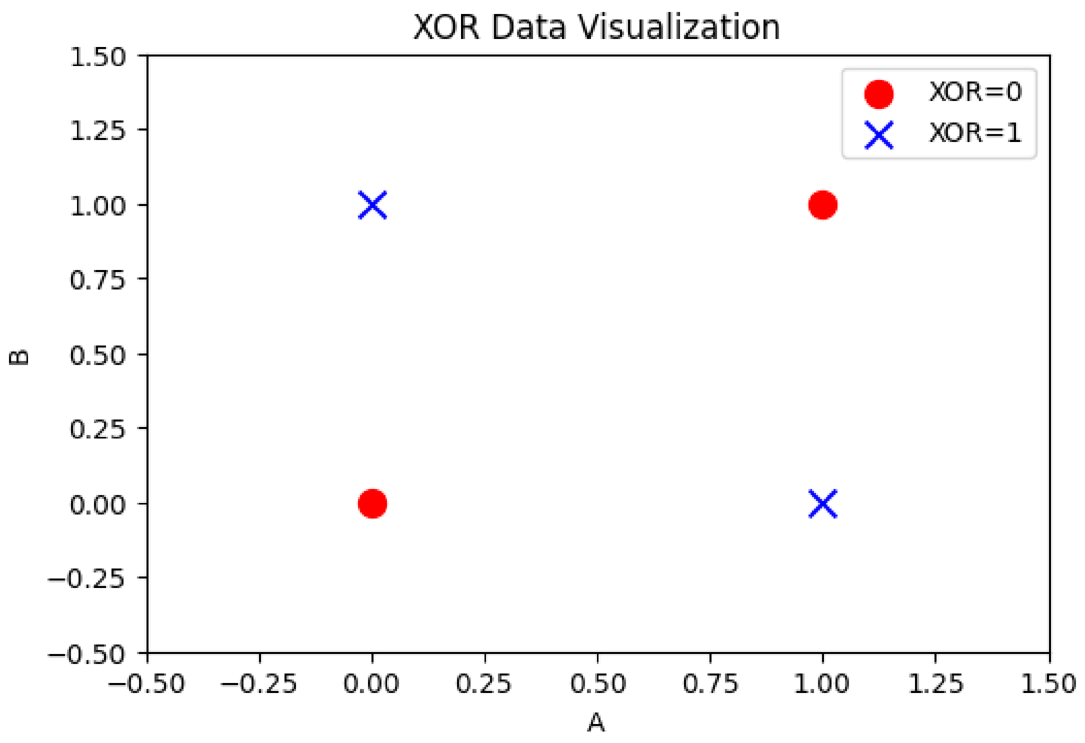
Artificial Neural Network (ANN), as a product of connectionism, belongs to this AI school. ANN simulates the connection and learning methods of biological neural networks. It is composed of a large number of artificial neurons and performs information processing and learning through weighted connections. The development of ANN benefits from in-depth research on connectionist ideas, and it has become a key tool in the fields of data science, machine learning, and deep learning, used to solve various tasks and simulate human intelligence.

The History of ANN

The history of artificial neural networks can be traced back to the 1940s, when Warren McCulloch and Walter Pitts proposed a neuron model to simulate the information processing process in biological nervous systems [1]. This early model was based on a simplified description of neuron behavior, including the concepts of signal inputs, weights, and simple threshold functions. People try to use simple linear models to solve logical operation problems. However, the XOR logical operation problem (XOR problem) shows that the traditional perceptron model (one of the earliest neural network models) cannot solve nonlinear separable problems. The XOR problem involves an XOR operation on two input bits, and its decision boundary is a nonlinear curve, which makes it impossible for a simple perceptron to perform effective classification.

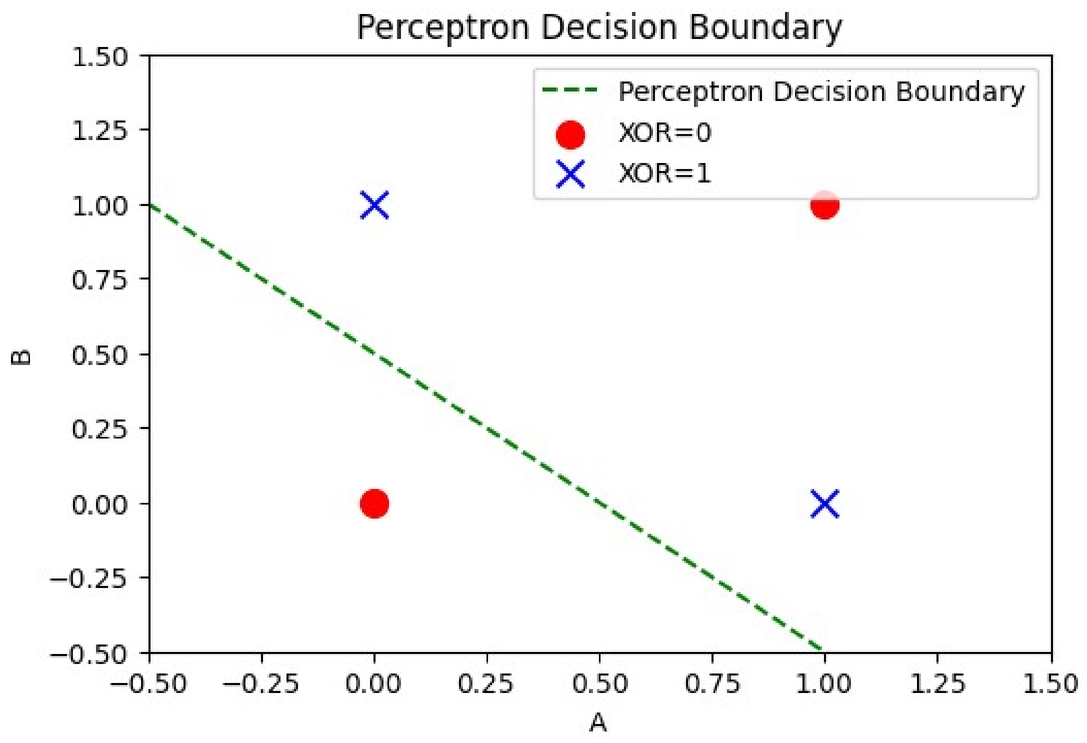
That is the simplest example of XOR Problem:

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.neural_network import MLPClassifier

# data
A = [0, 0, 1, 1]
B = [0, 1, 0, 1]
XOR = [0, 1, 1, 0]

# Extract data points whose XOR is 0 and 1 respectively
x0, y0, x1, y1 = [], [], [], []
for i in range(len(XOR)):
     if XOR[i] == 0:
         x0.append(A[i])
         y0.append(B[i])
     else:
         x1.append(A[i])
         y1.append(B[i])

#Set canvas size and range
plt.figure(figsize=(6, 4))
plt.xlim(-0.5, 1.5)
plt.ylim(-0.5, 1.5)

# Draw visualization of XOR data
plt.scatter(x0, y0, c='red', label='XOR=0', marker='o', s=100)
plt.scatter(x1, y1, c='blue', label='XOR=1', marker='x', s=100)
plt.xlabel('A')
plt.ylabel('B')
plt.title('XOR Data Visualization')
plt.legend(loc='upper right')
plt.show()
```



Then build a traditional perceptron (linear classification) to try to classify:

```
# Draw a visualization of a traditional perceptron
plt.figure(figsize=(6, 4))
plt.xlim(-0.5, 1.5)
plt.ylim(-0.5, 1.5)
x_boundary = np.linspace(-0.5, 1.5, 100)
y_boundary = -x_boundary + 0.5 # Linear decision boundary of the
perceptron
plt.plot(x_boundary, y_boundary, linestyle='--', color='green',
label='Perceptron Decision Boundary')
plt.scatter(x0, y0, c='red', label='XOR=0', marker='o', s=100)
plt.scatter(x1, y1, c='blue', label='XOR=1', marker='x', s=100)
plt.xlabel('A')
plt.ylabel('B')
plt.title('Perceptron Decision Boundary')
plt.legend(loc='upper right')
plt.show()
```

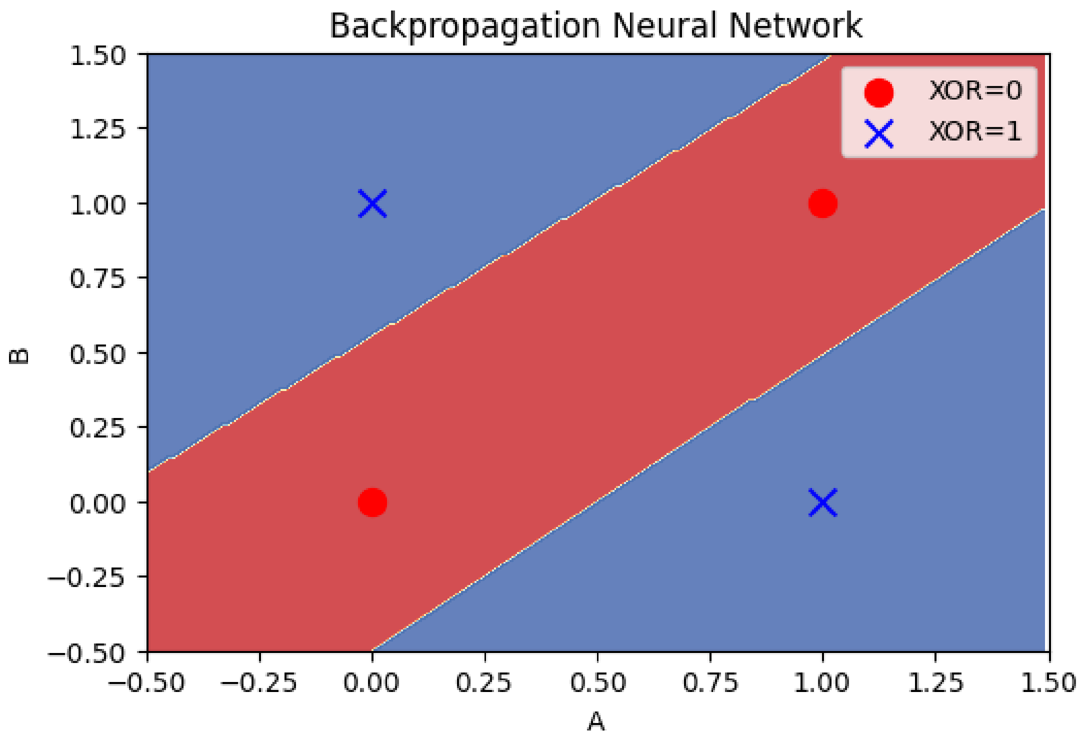


This challenge has driven improvements in neural networks and neuron models. It was not until the 1980s that the development of artificial neural networks ushered in important progress. Scholars such as Rumelhart, Hinton, and Williams proposed the backpropagation algorithm [2]. The introduction of this algorithm solved the problem of training multi-layer neural networks and enabled ANN to learn complex nonlinear functions more effectively.

Try using the backpropagation algorithm to build a nonlinear classifier:

```
# Draw a visualization of the backpropagation neural network
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])
model = MLPClassifier(hidden_layer_sizes=(4,), activation='relu',
solver='lbfgs', random_state=1)
model.fit(X, y)
x_min, x_max = -0.5, 1.5
y_min, y_max = -0.5, 1.5
h = 0.01
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(6, 4))
plt.xlim(-0.5, 1.5)
plt.ylim(-0.5, 1.5)
plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu, alpha=0.8)
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], c='red', label='XOR=0',
marker='o', s=100)
```

```
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], c='blue', label='XOR=1',
marker='x', s=100)
plt.xlabel('A')
plt.ylabel('B')
plt.title('Backpropagation Neural Network')
plt.legend(loc='upper right')
plt.show()
```



In the late 1980s and 1990s, ANN began to be more widely used in the fields of machine learning and data mining. Especially in the fields of image recognition, speech recognition, natural language processing and other fields, ANN has achieved remarkable achievements [3]. LeCun et al. demonstrated the powerful performance of convolutional neural networks in applications in the field of handwritten digit recognition [8]. These successful cases have promoted the continuous development of ANN research and application.

With the enhancement of computing power and the advent of the big data era, ANN has become more powerful and practical. The rise of deep learning technology enables deeper neural networks to be trained, which shows advantages when dealing with complex tasks and large-scale data sets [9].

Overall, the development history of ANN is full of continuous exploration and progress, from the initial simple model to today's deep learning technology. This development process has received contributions from many researchers and has promoted the widespread application of ANN in many fields.

From Neurons to Neural Networks

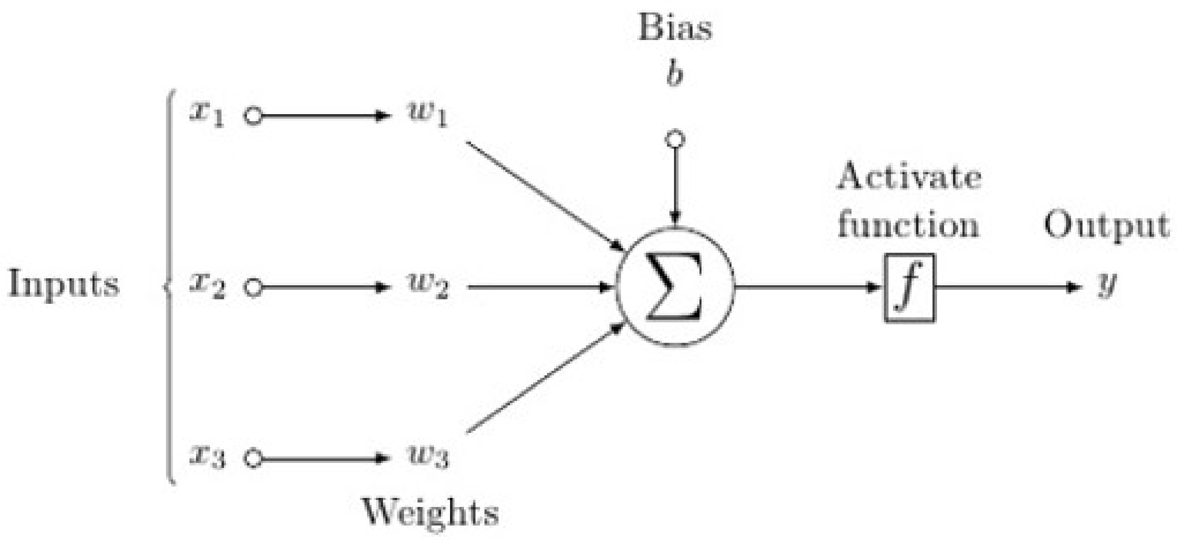


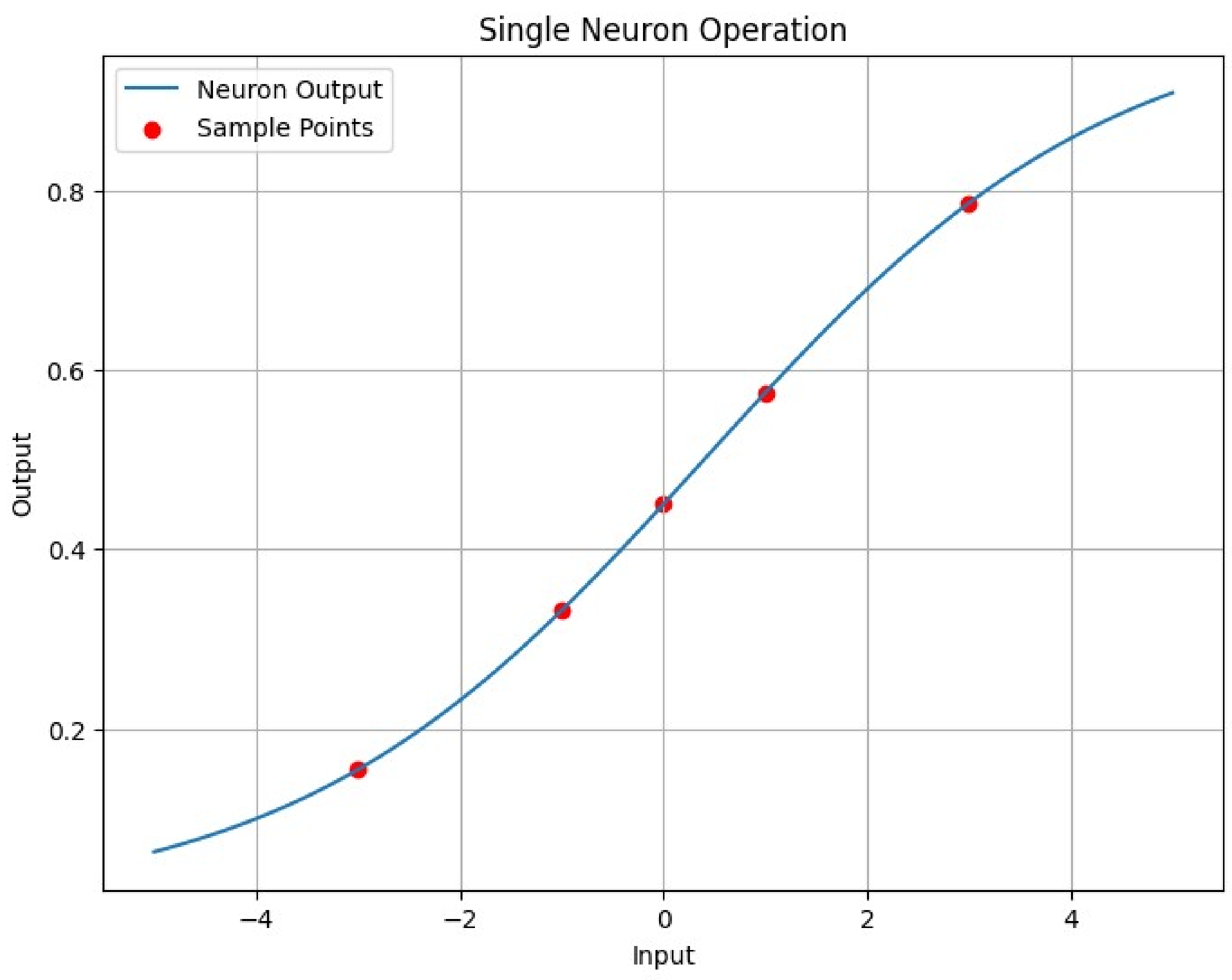
Figure 1: A Neuron

Neuron is the basic component unit of artificial neural network (ANN), which simulates the neurons in the biological nervous system. A single neuron has the following main components:

1. **Inputs**: Neurons receive information from other neurons or external input. These inputs are passed through synapses connected to neurons, and each input has an associated weight.
2. **Weights**: Each input has an associated weight, which is used to adjust the importance of the input signal. The weight determines the degree of influence of the input signal on the neuron, which can make the neuron pay more attention to certain inputs.
3. **Weighted Sum**: The neuron multiplies all input signals by the corresponding weights and then adds them to obtain a weighted sum. This weighted sum represents the degree of activation of the neuron.
4. **Activation Function**: The activation function determines the output of the neuron, usually a nonlinear function. The role of the activation function is to introduce nonlinearity, allowing neurons to learn complex relationships. Common activation functions include Sigmoid, ReLU (Rectified Linear Unit), Tanh, etc.[3][10].
5. **Threshold**: Sometimes, neurons can have a threshold before the activation function. If the weighted sum exceeds the threshold, the neuron will be activated and output a signal; otherwise, it will remain inactive [10].

```python
import numpy as np
import matplotlib.pyplot as plt

#Define the activation function of the neuron (the Sigmoid function is used here)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
# Define neuron weights and biases
w = 0.5 # weight
b = -0.2 # Offset

# Generate input data
x = np.linspace(-5, 5, 100) # Generate 100 data points between -5 and
5

# Calculate the output of the neuron
output = sigmoid(w * x + b)

# Generate some sample data points
sample_points_x = np.array([-3, -1, 0, 1, 3])
sample_points_y = sigmoid(w * sample_points_x + b)

# Visualize the operation of neurons
plt.figure(figsize=(8, 6))
plt.plot(x, output, label='Neuron Output')
plt.scatter(sample_points_x, sample_points_y, color='red', marker='o',
label='Sample Points')
plt.xlabel('Input')
plt.ylabel('Output')
plt.title('Single Neuron Operation')
plt.legend()
plt.grid(True)
plt.show()
```



The aboving code defines a simple neuron model, including activation function (here using the Sigmoid function), weights and biases. It then generates a sequence of input data (100 data points between -5 and 5 in this example), calculates the output of the neuron, and then visualizes it via Matplotlib. It can be seen that the typical S-shaped curve of the Sigmoid function represents the activation process of neurons.

Neurons are connected to form a neural network, and each neuron in the network can receive input from other neurons and produce an output. These connections form the topology of the ANN, which determines how information is transferred and processed in the network.

Neural networks are usually divided into multiple layers, including input layers, hidden layers, and output layers. Each layer is composed of multiple neurons. Information is passed from the input layer to the hidden layer and then further to the output layer. Connections exist not only between layers, but also between neurons in the same layer, forming a densely connected structure [3]. The training of neural networks involves adjusting the weights of the connections so that the network can respond appropriately to the input . This is typically accomplished through a backpropagation algorithm, which compares the differences between the network's output and the desired output and updates the weights of the connections based on these differences. The training process aims to minimize the loss function so that the network can correctly classify data or predict targets [3].

The number of layers and neurons in a neural network, the structure of the connections, and the choice of activation functions can be adapted to the specific task, enabling different types of problem solving. The complexity and performance of ANN are usually related to the size, depth and architecture of the network, which is also one of the core areas of technologies such as deep learning [9].

Propagation Algorithm and Training Process

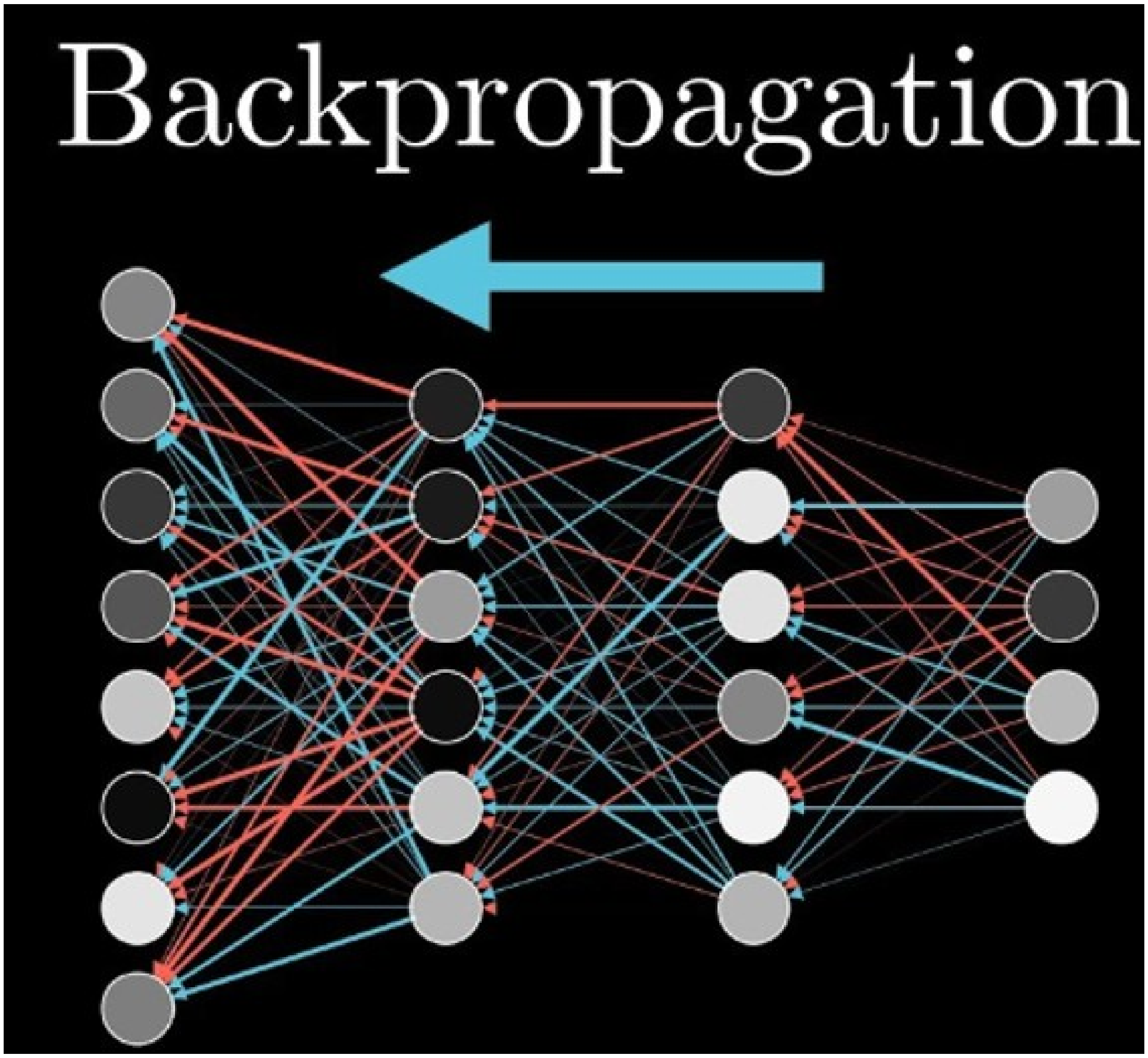


Figure 2: Backpropagation

Forward Propagation is the information transfer process in neural networks, in which input data is transferred between layers of the network, calculated and transferred layer by layer, and finally produces output. Each neuron receives the output from the previous layer's neurons, performs a weighted summation of them, and then applies an activation function to generate the input to the next layer. This process continues until the output layer [4] is reached.

Backward Propagation is a key step in training a neural network. Its core goal is to calculate the error between the network prediction and the actual label, and backpropagate the error from the output layer to each layer of the network. This process uses the chain rule to calculate the gradient, the derivative of the error with respect to the weights and biases, in order to adjust these parameters to minimize the error. Backpropagation is the basis for implementing gradient descent optimization [2].

The training process is a process in which the neural network iteratively learns through multiple forward propagation and back propagation. It aims to adjust the weights and biases of the network to minimize the loss function. The loss function measures the difference between the predicted results and the actual labels, and the optimizer is used to update the parameters, usually using algorithms such as gradient descent to find the minimum value of the loss function [11].

Loss functions and optimizers play a key role in the training process. Loss Function measures the performance of the model. It is a function that measures the difference between the predicted value and the true label. Common loss functions include mean square error, cross entropy, etc. Different loss functions are chosen for different tasks and models [3].

Optimizer is responsible for adjusting model parameters to minimize the loss function. It determines the direction and step size of parameter updates so that the model converges to the optimal solution. Common optimizers include stochastic gradient descent (SGD), Adam, RMSprop, etc. [11].

These concepts and steps constitute the basic operations in deep learning and are key components of neural network training and learning.

Different Types of ANN and Their Applications

In the field of deep learning, there are many different types of neural networks, each with a specific structure and purpose. These network types include Feedforward Neural Networks (FNN), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). Their application areas and characteristics are discussed below.

Feedforward neural network (FNN) is the most basic neural network structure in which information can only flow in one direction, from the input layer to the hidden layer and finally to the output layer. FNN is commonly used to solve classification and regression problems, such as image classification, text classification and regression analysis. Its main feature is that it has multiple hidden layers and can learn non-linear relationships, but it is not suitable for processing sequence data.

Convolutional neural network (CNN) is a type of neural network specifically designed to process images and two-dimensional data. The main feature of CNN is the convolutional layer, which can effectively capture the local features in the image, and the shared weights and pooling operations make it perform well to translation and scale invariance. CNN is widely used in image recognition, target detection, face recognition and other fields.

Recurrent neural network (RNN) is a neural network that processes sequence data. Its main feature is a recurrent structure that allows it to take into account contextual information in the sequence. RNN is widely used in natural language processing, speech recognition, time series prediction and other fields, and is favored for its sequence modeling capabilities. However, traditional RNN has problems such as gradient disappearance and gradient explosion, so improved RNN structures such as long short-term memory network (LSTM) and gated recurrent unit (GRU) have been introduced.

Each neural network type has its unique application areas and characteristics. Feedforward neural networks excel in classification and regression tasks, convolutional neural networks excel in image processing, and recurrent neural networks excel in modeling sequence data. In addition, hybrid network structures in deep learning, such as the combination of convolutional neural networks and recurrent neural networks (CNN-RNN), are also widely used in various complex tasks, such as video analysis and natural language generation. These different types of neural networks provide powerful tools for various application fields and promote the development and application of deep learning technology.

Application Examples

Image Identification

Artificial neural networks (ANN) are widely and profoundly used in the field of image recognition. Among them, Convolutional Neural Networks (CNN) is one of the most famous and successful types of neural networks, used to solve image recognition problems. The application of CNN has achieved remarkable achievements in image recognition.

Some well-known CNN architectures, such as LeNet, AlexNet, VGG, ResNet, and Inception, have achieved remarkable results in image recognition competitions. Their success is due to the special design of CNN, including convolutional layers, pooling layers, and fully connected layers, as well as the concepts of weight sharing and local receptive fields. These architectures can automatically learn features from data without manually engineering features.

In CNN applications, image data is first passed through convolutional layers for feature extraction, which use convolutional kernels to detect specific features in the image, such as edges, texture, and shape. Then, the pooling layer is used to reduce the size of the feature map, reduce the amount of computation, and enhance the translation invariance of the model. Finally, a fully connected layer is used to map the extracted features to image categories.

The application of CNN in image recognition is not limited to static images, but also includes tasks such as video analysis, object detection and segmentation. In these applications, CNNs achieve advanced image understanding by processing multiple pixels in sequential image frames or segmentation tasks.

In the literature, AlexNet [12] by Krizhevsky et al. and VGGNet [13] by Simonyan and Zisserman are iconic CNN architectures that achieved significant breakthroughs in the ImageNet image classification competition. These works and subsequent research have made CNN a mainstream method in the field of image recognition.

Implementation of a Neural Network for Handwritten Digits

Training the model requires the use of a data set. The data set used here is MNIST (Modified National Institute of Standards and Technology database). MNIST is a classic handwritten digit image dataset used for research and experiments in the fields of machine learning and computer vision. This dataset contains images of handwritten digits from different authors, covering ten categories of numbers 0 to 9. The MNIST data set includes a set of 28x28 pixel grayscale images with a total of 60,000 training samples and 10,000 test samples. Each of these images contains a handwritten number ranging from 0 to 9. Each image has a label associated with it that represents the handwritten digit contained in the image. These labels allow the machine learning algorithm to do supervised learning, i.e. learn how to correctly classify these numbers.

Since training with the complete data set requires large computing resources, only a sub-data set is used for training.

Load necessary packages:

```
import numpy
# scipy.special for the sigmoid function expit()
import scipy.special
# library for plotting arrays
import matplotlib.pyplot
# ensure the plots are inside this notebook, not an external window
%matplotlib inline
# helper to load data from PNG image files
import imageio
```

Define the neural network class:

```
# neural network class definition
class neuralNetwork:


    # initialise the neural network
    def __init__(self, inputnodes, hiddennodes, outputnodes, 
learningrate):
        # set number of nodes in each input, hidden, output layer
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # link weight matrices, wih and who
        # weights inside the arrays are w_i_j, where link is from node 
i to node j in the next layer
        # w11 w21
        # w12 w22 etc
        self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), 
(self.hnodes, self.inodes))
        self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), 
(self.onodes, self.hnodes))

        # learning rate
        self.lr = learningrate

        # activation function is the sigmoid function
        self.activation_function = lambda x: scipy.special.expit(x)
        pass


    # train the neural network
    def train(self, inputs_list, targets_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        # output layer error is the (target - actual)
        output_errors = targets - final_outputs
        # hidden layer error is the output_errors, split by weights,
recombined at hidden nodes
        hidden_errors = numpy.dot(self.who.T, output_errors)

        # update the weights for the links between the hidden and
output layers
        self.who += self.lr * numpy.dot((output_errors * final_outputs
* (1.0 - final_outputs)), numpy.transpose(hidden_outputs))

        # update the weights for the links between the input and
hidden layers
        self.wih += self.lr * numpy.dot((hidden_errors *
hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))

        pass


    # query the neural network
    def query(self, inputs_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list, ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)
        # calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        return final_outputs
```

Setting parameters of the ANN Model:

```
# number of input, hidden and output nodes
input_nodes = 784
hidden_nodes = 200
output_nodes = 10

# learning rate
learning_rate = 0.1

# create instance of neural network
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes,
learning_rate)
```

Import data and train:

```
import os

file_path = "mnist_dataset/mnist_train_100.csv"

if os.path.exists(file_path):
     # The file exists and can be opened and loaded with data
     with open(file_path, 'r') as training_data_file:
         training_data_list = training_data_file.readlines()
else:
     # The file does not exist and an error message is output.
     print(f"File not found: {file_path}")

# load the mnist training data CSV file into a list
training_data_file = open("mnist_dataset/mnist_train_100.csv", 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()

# train the neural network

# epochs is the number of times the training data set is used for
training
epochs = 10

for e in range(epochs):
    # go through all records in the training data set
    for record in training_data_list:
        # split the record by the ',' commas
        all_values = record.split(',')
        # scale and shift the inputs
        inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) +
0.01
        # create the target output values (all 0.01, except the
desired label which is 0.99)
        targets = numpy.zeros(output_nodes) + 0.01
        # all_values[0] is the target label for this record
        targets[int(all_values[0])] = 0.99
        n.train(inputs, targets)
        pass
    pass
```

The training process covers multiple epochs, and in each epoch, each sample in the training data set is looped through.

For each sample, the image data was preprocessed, scaling the pixel values to the range of 0.01 to 0.99, and creating the target output vector. Only one element is 0.99, which corresponds to the true category of the sample, and the other elements are all 0.01.

These inputs and target outputs are then used to train the network, updating the weights by backpropagating errors.

The network uses a gradient descent algorithm with a learning rate of 0.1 for weight update.

Then we can try using the model to test our own handwritten digits:

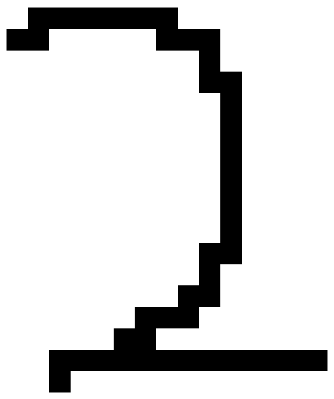


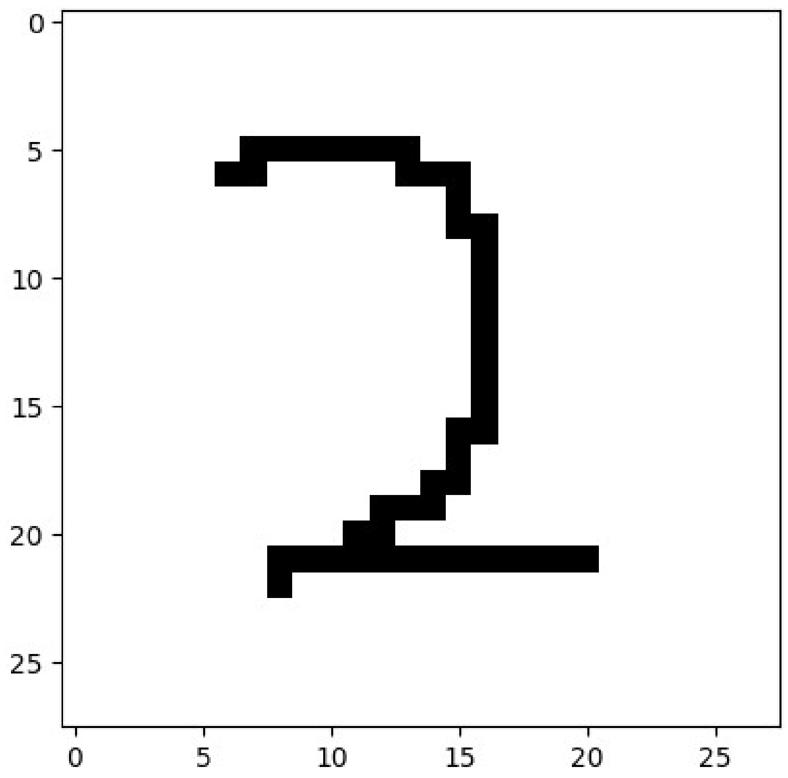
Figure 3: 28 * 28 Pix Number Picture

```
# load image data from png files into an array
print ("loading ... my_own_images")
img_array = imageio.imread('my_own_images/my_onw_number_2.png',
mode='L')

# reshape from 28x28 to list of 784 values, invert values
```

```python
img_data  = 255.0 - img_array.reshape(784)

# then scale data to range from 0.01 to 1.0
img_data = (img_data / 255.0 * 0.99) + 0.01
print("min = ", numpy.min(img_data))
print("max = ", numpy.max(img_data))

# plot image
matplotlib.pyplot.imshow(img_data.reshape(28,28), cmap='Greys',
interpolation='None')

# query the network
outputs = n.query(img_data)
print (outputs)

# the index of the highest value corresponds to the label
label = numpy.argmax(outputs)
print("network says ", label)
```

```
loading ... my_own_images
min =  0.01
max =  1.0
[[0.01048235]
 [0.18580256]
 [0.2695291 ]
 [0.06494361]
 [0.03405985]
 [0.13015911]
 [0.05484294]
 [0.21231046]
 [0.03465366]
 [0.02791571]]
network says  2
```

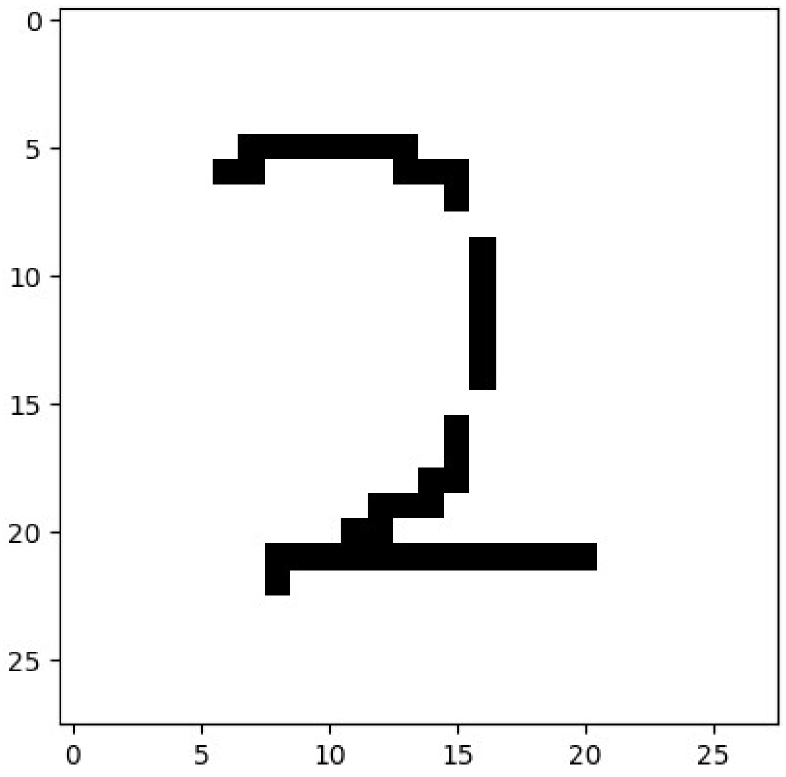
```
C:\Users\yejiu\AppData\Local\Temp\ipykernel_18492\1567663577.py:3:
DeprecationWarning: Starting with ImageIO v3 the behavior of this
function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  img_array = imageio.imread('my_own_images/my_onw_number_2.png',
mode='L')
```



The model's immunity to noise can then be tested, using "corrupted" digital images.

```
# load image data from png files into an array
print ("loading ... my_own_images")
img_array =
imageio.imread('my_own_images/my_onw_number_2_destoyed.png', mode='L')

# reshape from 28x28 to list of 784 values, invert values
img_data  = 255.0 - img_array.reshape(784)

# then scale data to range from 0.01 to 1.0
img_data = (img_data / 255.0 * 0.99) + 0.01
print("min = ", numpy.min(img_data))
print("max = ", numpy.max(img_data))

# plot image
matplotlib.pyplot.imshow(img_data.reshape(28,28), cmap='Greys',
interpolation='None')

# query the network
outputs = n.query(img_data)
print (outputs)

# the index of the highest value corresponds to the label
label = numpy.argmax(outputs)
print("network says ", label)
```

```
loading ... my_own_images
min =  0.01
max =  1.0
[[0.01292659]
 [0.2048991 ]
 [0.31940498]
 [0.05486945]
 [0.03603111]
 [0.14394597]
 [0.076837  ]
 [0.16229104]
 [0.04933541]
 [0.02707867]]
network says  2

C:\Users\yejiu\AppData\Local\Temp\ipykernel_18492\2580933180.py:3:
DeprecationWarning: Starting with ImageIO v3 the behavior of this
function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  img_array =
imageio.imread('my_own_images/my_onw_number_2_destoyed.png', mode='L')
```

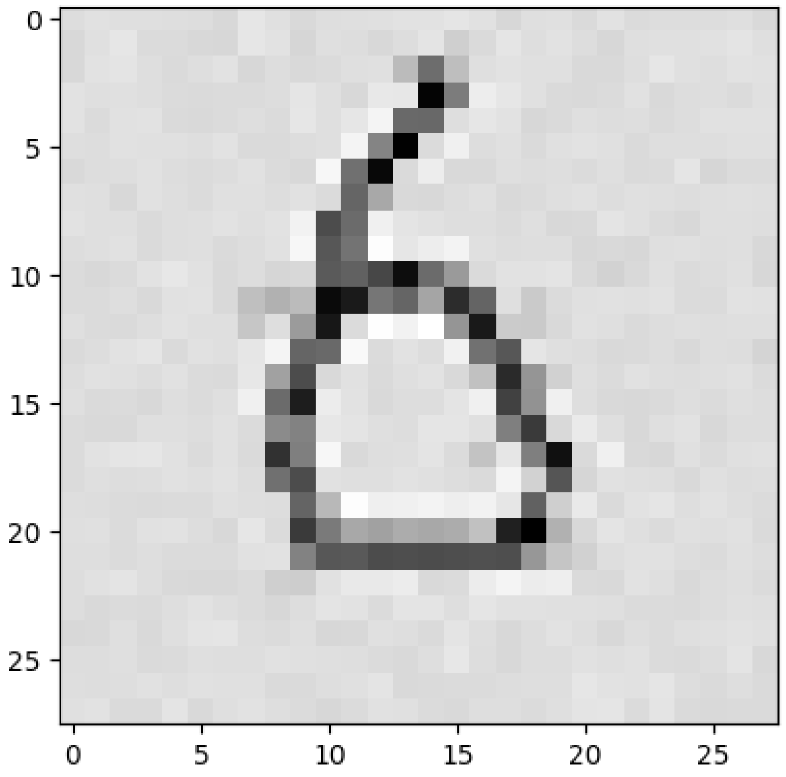


It can be found that the model can still recognize numbers even when the coherence of the digital image is destroyed (that is, noise is added).

But this does not mean that this model is perfect. We can try to add noise by changing the background color to gray to test the performance of this model:

```
# load image data from png files into an array
print ("loading ... my_own_images")
img_array = imageio.imread('my_own_images/my_own_noisy_6.png',
mode='L')

# reshape from 28x28 to list of 784 values, invert values
img_data  = 255.0 - img_array.reshape(784)

# then scale data to range from 0.01 to 1.0
img_data = (img_data / 255.0 * 0.99) + 0.01
print("min = ", numpy.min(img_data))
print("max = ", numpy.max(img_data))

# plot image
matplotlib.pyplot.imshow(img_data.reshape(28,28), cmap='Greys',
interpolation='None')

# query the network
outputs = n.query(img_data)
print (outputs)

# the index of the highest value corresponds to the label
label = numpy.argmax(outputs)
print("network says ", label)
```

```
loading ... my_own_images
min =  0.1458823529411765
max =  0.7748235294117647
[[0.11484987]
 [0.01261609]
 [0.03435632]
 [0.04607629]
 [0.11360233]
 [0.04399229]
 [0.03841073]
 [0.06041205]
 [0.02302952]
 [0.00588872]]
network says  0

C:\Users\yejiu\AppData\Local\Temp\ipykernel_18492\4220750661.py:3:
DeprecationWarning: Starting with ImageIO v3 the behavior of this
function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  img_array = imageio.imread('my_own_images/my_own_noisy_6.png',
mode='L')
```



The reason why the model cannot accurately recognize handwritten digits is because changing the background color causes the content and characteristics of the image to change. Deep learning models are very sensitive to the characteristics and structure of images, including pixel values and relationships between pixels. When the background color is all modified to gray, the pixel value distribution and contrast of the image are changed, resulting in information loss and feature extraction errors.

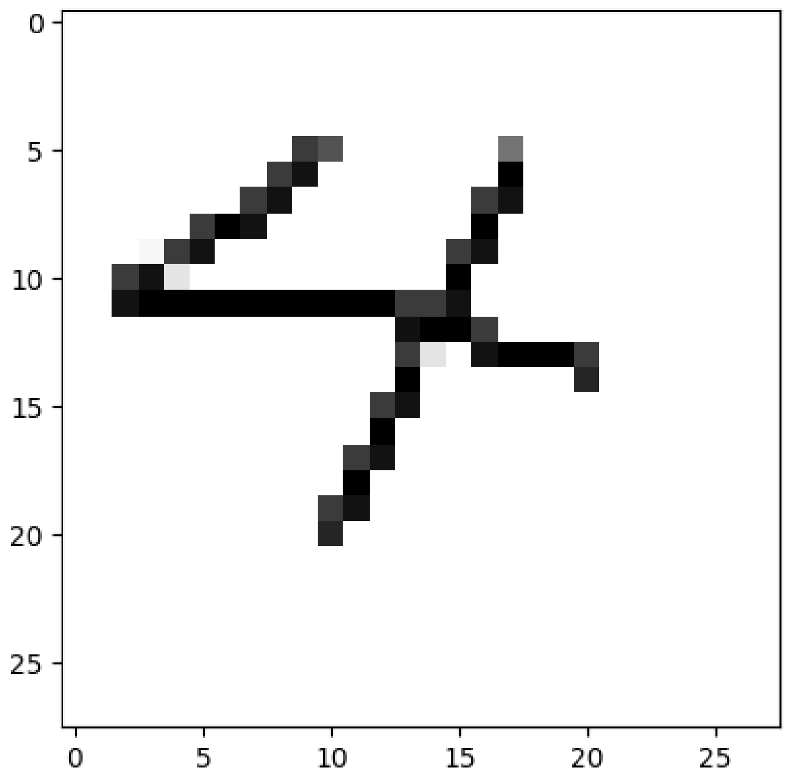
The model also doesn't always correctly identify digits without noise:

```
# load image data from png files into an array
print ("loading ... my_own_images")
img_array = imageio.imread('my_own_images/John_Number.png', mode='L')

# reshape from 28x28 to list of 784 values, invert values
img_data  = 255.0 - img_array.reshape(784)

# then scale data to range from 0.01 to 1.0
img_data = (img_data / 255.0 * 0.99) + 0.01
print("min = ", numpy.min(img_data))
print("max = ", numpy.max(img_data))

# plot image
matplotlib.pyplot.imshow(img_data.reshape(28,28), cmap='Greys',
interpolation='None')

# query the network
```

```
outputs = n.query(img_data)
print (outputs)

# the index of the highest value corresponds to the label
label = numpy.argmax(outputs)
print("network says ", label)
```

```
loading ... my_own_images
min =  0.01
max =  1.0
[[0.02141367]
 [0.19428378]
 [0.07060907]
 [0.13730757]
 [0.09900421]
 [0.07306868]
 [0.01541563]
 [0.23574287]
 [0.09720228]
 [0.01574207]]
network says  7
```

```
C:\Users\yejiu\AppData\Local\Temp\ipykernel_18492\3314835913.py:3:
DeprecationWarning: Starting with ImageIO v3 the behavior of this
function will switch to that of iio.v3.imread. To keep the current
behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  img_array = imageio.imread('my_own_images/John_Number.png',
mode='L')
```



This is because the display of numbers in MNIST is written as standard as possible, and when training this model, only a smaller sub-data set is used for training due to limitations in computing resources. Therefore, the generalization ability of the model is insufficient, and the recognition accuracy is low when facing relatively non-standard handwritten digits.

This is the challenge of ANN models. The ANN model requires a large amount of data to train a model with strong generalization capabilities to cope with various possible types of inputs, which in turn requires a large amount of computing resources.

Other Applications

Artificial neural networks (ANN) have achieved remarkable application results in many fields. In addition to image recognition, ANN is widely used in speech recognition, natural language processing, financial prediction, drug discovery, bioinformatics and other fields. In the field of speech recognition, deep learning models based on neural network architecture have achieved great success in improving speech recognition accuracy and fluency. This includes models based on Long Short-Term Memory Networks (LSTM) and Recurrent Neural Networks (RNN), which can more accurately transcribe speech to text [14].

Furthermore, natural language processing (NLP) is another area where ANN has made significant progress. Deep learning methods, such as the Transformer model, have achieved breakthrough performance in tasks such as machine translation, text generation, and language modeling [15]. In the financial field, ANN is used for stock prediction, market analysis, and risk management. The nonlinear modeling capabilities of ANN provide more accurate prediction and decision support for the financial field [16].

In the field of life sciences, ANN also has important applications. For example, neural networks play a key role in drug discovery and bioinformatics. They can predict molecular interactions, drug activity and other information, helping to accelerate the development of new drugs [17]. These applications have contributed to the widespread use of ANN in interdisciplinary fields. The application provides strong support.

Discussion

Advantages and Disadvantages of ANN

Artificial Neural Networks (ANNs) are formidable tools in the realm of machine learning, replete with several notable advantages. Firstly, ANNs excel in modeling intricate nonlinear relationships, endowing them with exceptional performance across a wide spectrum of applications. Tasks such as image recognition, natural language processing, and sound recognition greatly benefit from the capacity of ANNs to discern and encapsulate highly intricate data features. Secondly, ANNs possess the remarkable ability to autonomously glean and adapt to patterns and features within datasets. Unlike traditional models, ANNs obviate the need for laborious manual feature engineering, instead autonomously extracting pertinent information during the training process, thus simplifying model development. Lastly, ANNs exhibit a remarkable propensity to handle large-scale datasets efficiently. Their prodigious computational capabilities render them well-suited for processing extensive datasets comprising millions of data points, making them indispensable for substantial undertakings like large-scale image processing, text analysis, and data mining.

Nonetheless, ANNs do harbor certain limitations. Notably, they impose substantial computational demands, particularly when delving into deep neural networks. These demands encompass the requisites of copious training data, high-performance hardware, and the deployment of extensive computing clusters, rendering ANNs financially burdensome in certain applications. Deep neural networks are notably susceptible to overfitting, particularly in instances where training data is scarce. Overfitting engenders models that exhibit exemplary performance on the training data but fail to generalize effectively to new data. Furthermore, ANNs are often regarded as opaque, characterized by a 'black-box' nature, rendering it arduous to elucidate their internal decision-making processes. This characteristic can prove prohibitive in applications such as medical diagnostics and legal fields where interpretability and transparency are paramount.

Comparison with Other Machine Learning Algorithms

Artificial Neural Networks (ANNs) have emerged as a potent computational paradigm inspired by the biological nervous system. They closely replicate the interconnectedness and information exchange among neurons, enabling them to adeptly handle complex nonlinear relationships. Consequently, they have found particular utility in various applications, including natural language processing, image recognition, and sound processing.

One of the most salient attributes of ANNs is their capacity to process vast volumes of data, excelling particularly in the domain of deep learning. Deep neural networks, characterized by multiple hidden layers, autonomously learn abstract features from data, underscoring their supremacy when confronted with large datasets. Adaptability is a hallmark of ANNs, enabling them to adjust their weights and structure to accommodate variations across diverse domains and tasks. This versatility allows ANNs to perform effectively in a wide array of applications. Furthermore, ANNs support not only supervised learning but also unsupervised learning and reinforcement learning. This means that ANNs can be applied in tasks such as clustering, dimensionality reduction, and autoencoders, expanding their utility beyond traditional classification and regression. A significant advantage of ANNs is their ability to automatically learn and extract features from data without the need for manual feature engineering, which is particularly advantageous when dealing with intricate datasets. This capability alleviates the burden of human intervention. Additionally, ANNs exhibit proficiency in continuous learning, enabling them to adapt to evolving environments by continually learning from new data. This makes them highly effective in scenarios such as online learning, time series analysis, and incremental learning.

However, it is essential to acknowledge the limitations of ANNs, including the requirement for substantial labeled data for training, lengthy training times, and sensitivity to hyperparameters. In comparison to other machine learning techniques, ANNs may not always be the optimal choice for specific tasks.

Therefore, the decision to use ANNs or other machine learning methods should be contingent upon the nature of the task and the available data. In many instances, ANNs can deliver exceptional performance, especially when dealing with large-scale, high-dimensional, nonlinear data and tasks demanding automated feature extraction and pattern recognition.

Conclusion

Combined with the previous discussion of artificial neural networks (ANN), ANN, as a bionic learning model, plays a crucial role in the field of machine learning. Its development history has a long history, from the simple structure of early models to today's deep learning technology, it has continued to evolve and innovate. In this process, ANN has shown amazing advantages and potential.

First, ANN has excellent nonlinear modeling capabilities, allowing it to exhibit excellent performance in multiple fields. Especially in tasks such as image recognition, natural language processing, and voice recognition, ANN can efficiently capture and utilize the complex features of data, becoming one of the main tools in these fields. Secondly, ANN has the ability of autonomous learning and self-adaptation, without the need for tedious manual feature engineering, and can automatically extract important information from the data, simplifying the complexity of model development. In addition, ANN performs well in processing large-scale data sets, especially in projects such as large-scale image processing, text analysis, and data mining, showing indispensable value.

However, ANN also has some limitations. Its computing requirements are large, especially in deep neural networks, which require a large amount of training data, high-performance hardware and huge computing resources, making its application in some fields economically burdensome. In addition, deep neural networks are prone to overfitting, especially when there is less training data, resulting in insufficient generalization capabilities of the model on new data. In addition, ANN is often regarded as a "black box" whose internal decision-making process is difficult to explain, which may become a limitation in applications that require transparency, such as medical diagnosis and legal fields.

Taken together, ANN, as a multifunctional and versatile machine learning model, has both advantages and limitations. ANNs have demonstrated impressive capabilities across different domains and tasks, but the choice of whether to use ANNs or other machine learning methods should be weighed based on the nature of the task and the available data. ANN performs well in processing large-scale, high-dimensional, non-linear data and automatic feature extraction and pattern recognition, especially in these fields.

In the future, the development and application prospects of ANN are still broad. With the continuous advancement of technology, ANN may be able to better solve the challenges it currently faces, such as computing resource requirements, model interpretability, etc., and is also expected to further expand its applications in more fields. The continuous evolution of ANN will continue to promote progress in the fields of machine learning and artificial intelligence, contributing to solving more complex problems and promoting the development of science and technology.

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," The Bulletin of Mathematical Biophysics, vol. 5, no. 4, pp. 115–133, Dec. 1943, doi: https://doi.org/10.1007/bf02478259.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533–536, Oct. 1986, doi: https://doi.org/10.1038/323533a0.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436–444, May 2015, doi: https://doi.org/10.1038/nature14539.

[4] Y. Bengio, Y. Lecun, and G. Hinton, "Deep learning for AI," Communications of the ACM, vol. 64, no. 7, pp. 58–65, Jul. 2021, doi: https://doi.org/10.1145/3448250.

[5] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. New Jersey: Pearson, 2010.

[6] A. Newell et al., "GPS, a program that simulates human thought: Computers & amp; thought," Guide books, https://dl.acm.org/doi/10.5555/216408.216432 (accessed Nov. 10, 2023).

[7] S. K. Andersen, "Probabilistic reasoning in intelligent systems: Networks of plausible inference," Artificial Intelligence, vol. 48, no. 1, pp. 117–124, Feb. 1991, doi: https://doi.org/10.1016/0004-3702(91)90084-w.

[8] "Gradient-based learning applied to document recognition - IEEE Journals & Magazine," Ieee.org, 2019. https://ieeexplore.ieee.org/document/726791

[9] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 85–117, Jan. 2015, doi: https://doi.org/10.1016/j.neunet.2014.09.003.

[10] S. Haykin and S. Hakin, The Neural networks a comprehensive foundation. Macmillan College Publishing Company.

[11] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv.org, Dec. 22, 2014. https://arxiv.org/abs/1412.6980

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Communications of the ACM, vol. 60, no. 6, pp. 84–90, May 2012, doi: https://doi.org/10.1145/3065386.

[13] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv.org, Apr. 10, 2015. https://arxiv.org/abs/1409.1556

[14] A. Graves, A. Mohamed, and G. Hinton, “Speech Recognition with Deep Recurrent Neural Networks,” arXiv.org, 2013. https://arxiv.org/abs/1303.5778

[15] A. Vaswani et al., “Attention Is All You Need,” arXiv.org, 2017. https://arxiv.org/abs/1706.03762

[16] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, “Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks,” 2017 IEEE 19th Conference on Business Informatics (CBI), Jul. 2017, doi: https://doi.org/10.1109/cbi.2017.23.

[17] A. Mayr, G. Klambauer, T. Unterthiner, and S. Hochreiter, “DeepTox: Toxicity Prediction using Deep Learning,” Frontiers in Environmental Science, vol. 3, Feb. 2016, doi: https://doi.org/10.3389/fenvs.2015.00080.