Chapter 1 Introduction to Artificial Intelligence

What we want is a machine that can learn from experience.
-Alan Turing

1.1 Artificial Intelligence in Action

Information technology is the third industrial revolution in human history. The popularity of computers, the Internet, and smart home technology has greatly facilitated people's daily lives. Through programming, humans can hand over the interaction logic designed in advance to the machine to execute repeatedly and quickly, thereby freeing humans from simple and tedious repetitive labor. However, for tasks that require a high level of intelligence, such as face recognition, chat robots, and autonomous driving, it is difficult to design clear logic rules. Therefore, traditional programming methods are powerless to those kinds of tasks, whereas Artificial Intelligence (AI), as the key technology to solve this kind of problem, is very promising.

With the rise of deep learning algorithms, AI has achieved or even surpassed human-like intelligence on some tasks. For example, AlphaGo program has defeated Ke Jie, one of the strongest human Go experts, and OpenAI Five has beaten the champion team OG on Dota2 game. In the mean time, practical technologies such as face recognition, intelligent speech, and machine translation have entered people's daily lives. Now our lives are actually surrounded by AI. Although the current level of intelligence that can be reached is still a long way from Artificial General Intelligence (AGI), we still firmly believe that the era of AI has arrived.

Next we will introduce the concepts of AI, machine learning, and deep learning, as well as the connections and differences between them.

1.1.1 Artificial Intelligence

AI is a technology that allows machines to acquire intelligent and inferential mechanisms like humans. This concept first appeared at the Dartmouth Conference in 1956. This is a very challenging task. At present, human beings cannot yet have a comprehensive and scientific understanding of the working mechanism of human brain. It is undoubtedly more difficult to make the intelligent machines that can reach the level of human brain. With that being said, machines that archive similar, or even surpass human intelligence in some way, have been proven to be feasible.

How to realize AI is a very broad question. The development of AI has mainly gone through three stages, and each stage represents the exploration footprint of human trying to realize AI from different angles. In the early stage, people tried to develop intelligent systems by summarizing and generalizing some logical rules and implementing them in the form of computer programs. But such explicit rules are often too simple, and are difficult to be used to express complex and abstract concepts and rules. This stage is called the inference period.

In the 1970s, scientists tried to solve AI through knowledge database and reasoning. They built a large and complex expert system to simulate the intelligence level of human experts. One of the biggest difficulties with these explicitly specified rules is that many complex, abstract concepts cannot be implemented in concrete code. For example, the process of human recognition of pictures and understanding of languages cannot be simulated by established rules at all. To solve such problems, a research discipline that allowed machines to automatically learn rules from data, known as machine learning, was born. Machine learning became a popular subject in AI in the 1980s. This is the second stage.

In machine learning, there is a direction to learn complex, abstract logic through neural networks. Research on the direction of neural networks has experienced two ups and downs. Since 2012, the applications of deep neural network technology have made major breakthroughs in fields like computer vision, natural language processing, and robotics. Some tasks have even surpassed the level of human intelligence. This is the third revival of AI. Deep neural networks eventually have a new name - deep learning. Generally speaking, the essential difference between neural networks and deep learning is not large. Deep learning refers to models or algorithms based on deep neural networks. The relationship between artificial intelligence, machine learning, neural networks, and deep learning is shown in Figure 1.1.

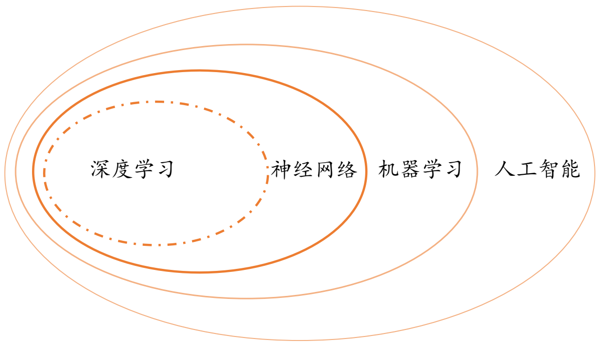


Figure 1.1 Relation of artificial intelligence, machine learning, neural networks and deep learning

1.1.2 Machine Learning

Machine learning can be divided into Supervised Learning, Unsupervised Learning, and Reinforcement Learning, as shown in Figure 1.2.

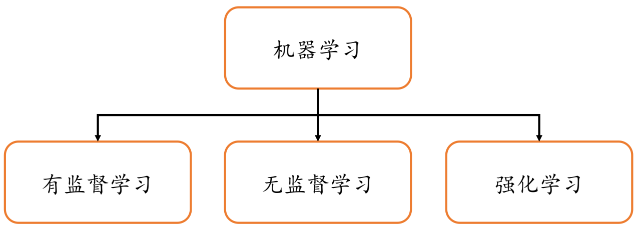


Figure 0.1 Categories of Machine Learning

Supervised Learning The supervised learning data set contains samples $\boldsymbol{x}$ and sample labels $\boldsymbol{y}$. The algorithm needs to learn the mapping relationship $f_\theta: \boldsymbol{x} \rightarrow \boldsymbol{y}$, where f_θ represents the model function and θ is the parameters of the model. During training, the model parameters θ

are optimized by minimizing errors between the model prediction and the real value $\boldsymbol{y}$, so that the model can predict more accurately next. Common supervised learning models include linear regression, logistic regression, support vector machines, and random forests.

Unsupervised Learning Collecting labeled data is often more expensive. For a sample-only data set, the algorithm needs to discover the modalities of the data itself. This kind of algorithm is called unsupervised learning. One type of algorithms in unsupervised learning uses itself as a supervised signal, i.e. $f_\theta: \boldsymbol{x} \to \boldsymbol{x}$, which is known as self-supervised learning. During training, parameters are optimized by minimizing the error between the model's predicted value $f_\theta(\boldsymbol{x})$ and itself $\boldsymbol{x}$. Common unsupervised learning algorithms include Self-Encoders and Generative Adversarial Networks.

Reinforcement learning A type of algorithm that learns strategies for solving problems by interacting with the environment. Unlike supervised and unsupervised learning, reinforcement learning problems do not have a clear "correct" action supervision signal. The algorithm needs to interact with the environment to obtain a lagging reward signal from the environmental feedback. Therefore, it is not possible to calculate the errors between model prediction and “correct values” to optimize the network. Common reinforcement learning algorithms are DQN and PPO.

1.1.3 Neural Networks and Deep Learning

Neural network algorithms are a class of algorithms that learn from data based on neural networks. They still belong to the category of machine learning. Due to the limitation of computing power and data volume, early neural networks were shallow, usually with around 1 to 4 layers. Therefore, the network expression ability was limited. With the improvement of computing power and the arrival of big data era, highly parallelized GPUs and massive data make training of large-scale neural networks possible.

In 2006, Geoffrey Hinton first proposed the concept of deep learning. In 2012, AlexNet, a 8-layer deep neural network, was released and achieved huge performance improvements in the image recognition competition. Since then, neural network models with dozens, hundreds, and even thousands of layers have been developed successively, showing strong learning ability. Algorithms implemented using deep neural networks are generally referred to as deep learning models. In essence, neural networks and deep learning can be considered the same.

Let’s simply compare the difference between deep learning with other algorithms. As shown in Figure 0.2 rule-based systems usually write explicit logic, which is generally designed for specific tasks and is not suitable for other tasks. Traditional machine learning algorithms artificially design feature detection methods with certain generality, such as SIFT and HOG features. These features are suitable for a certain type of tasks and have certain generality. But the performance highly depends on how to design those features. The emergence of neural networks has made it possible for computers to design those features automatically through neural networks without human intervention. Shallow neural networks typically have limited feature extraction capability, while deep neural networks are capable of extracting high-level, abstract features and have better performance.

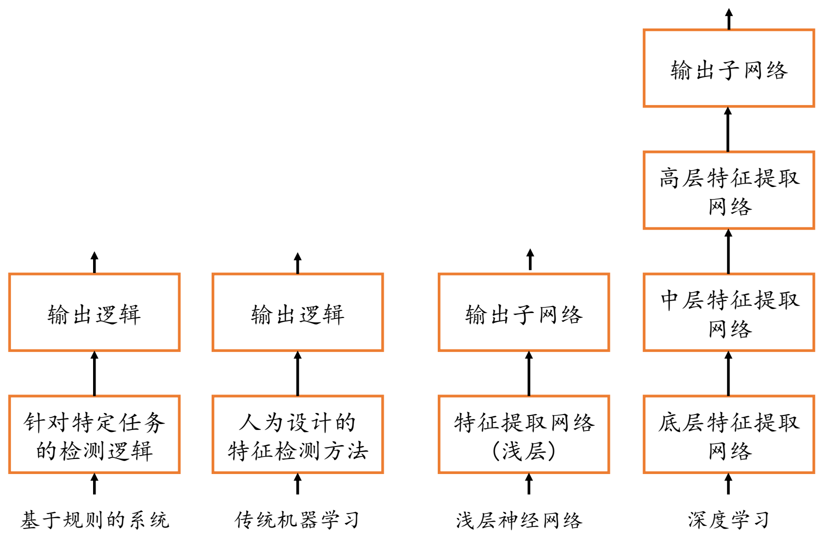


Figure 0.2 Comparison of deep learning and other algorithms

1.2 The History of Neural Networks

We divide the development of neural networks into shallow neural network stages and deep learning stages, with 2006 as the dividing point. Before 2006, deep learning developed under the name of neural networks and connectionism, and experienced two ups and two downs. In 2006, Geoffrey Hinton first named deep neural networks as deep learning, which started the third revival of deep learning.

1.2.1 Shallow Neural Networks

In 1943, psychologist Warren McCulloch and logician Walter Pitts proposed the earliest mathematical model of neurons based on the structure of biological neurons, called MP neuron models. The model $f(\boldsymbol{x}) = h(g(\boldsymbol{x}))$, where $g(\boldsymbol{x}) = \sum_i x_i, x_i \in \{0,1\}$, takes values from $g(\boldsymbol{x})$ to predict output values as shown in Figure 0.3. If $g(\boldsymbol{x}) \geq 0$, output is 1; if $g(\boldsymbol{x}) < 0$, output is 0. It can be seen that the MP neuron models have no learning ability, and can only complete fixed logic judgments.

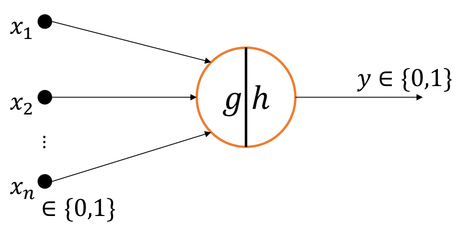


Figure 0.3 MP neuron model

In 1958, American psychologist Frank Rosenblatt proposed the first neuron model that can automatically learn weights, called Perceptron. As shown in Figure 0.4, the error between the output value o and the true value y is used to adjust the weights of the neurons $\{w_1, w_2, \dots, w_n\}$. Frank Rosenblatt then implemented the perceptron model based on the "Mark 1 perceptron" hardware. As shown inFigrue 0.5 and Figure 0.6, the input is an image sensor with 400 units and the output is 8 node terminals. It can successfully identify some English letters. It is generally believed that 1943 to 1969 is the first prosperous period of artificial intelligence development.

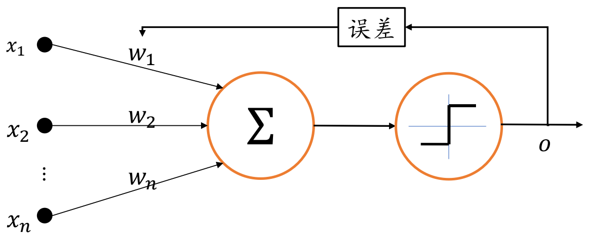
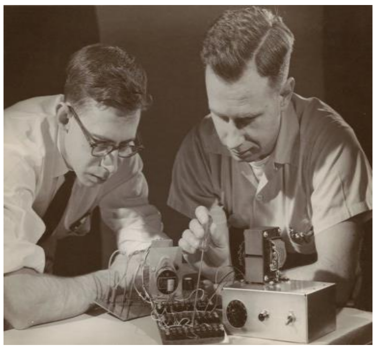
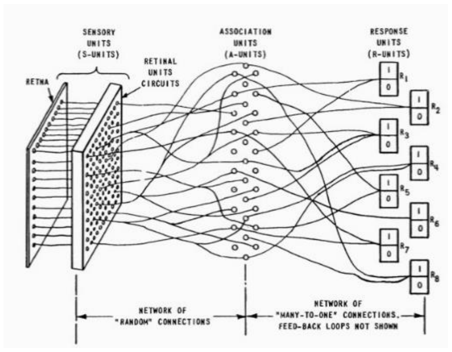


Figure 0.4 Perceptron model




SENSORY UNITS (S-UNITS)
RETNA
RETINAL UNITS CIRCUITS
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS. FEED-BACK LOOPS NOT SHOWN

Figrue 0.5 Frank Rosenblatt and Mark 1 Perception[①]   Figure 0.6 Mark 1 Perception network architectue[②]

In 1969, the American scientist Marvin Minsky and others pointed out the main flaw of linear models such as Perceptrons in the book "Perceptrons". They found that Perceptrons cannot handle simple linear inseparable problems such as XOR or XOR. This directly led to the trough period of Perceptron-related research on neural networks. It is generally considered that 1969 ~ 1982 was the first winter of the artificial intelligence.

Although it is in the trough period of AI, there are still many significant studies published one after another. The most important one is the Back Propagation (BP) algorithm, which is still the core foundation of modern deep learning algorithms. In fact, the mathematical idea of back propagation has been derived as early as the 1960s, but it had not been applied to neural networks. In 1974, American scientist Paul Werbos first proposed that the BP algorithm can be applied to neural networks in his doctoral dissertation. Unfortunately, this result has not received enough attention. Until 1986, David Rumelhart et al. published a paper using BP algorithm for feature learning in Nature, and the BP algorithm started gaining widespread attention.

In 1982, with the introduction of John Hopfild's cyclically connected Hopfield network, the second wave of artificial intelligence renaissance from 1982 to 1995 was started. During this period, convolutional neural networks, recurrent neural networks, and back propagation algorithms were developed once after another. In 1986, David Rumelhart, Geoffrey Hinton and others applied the BP algorithm to multi-layer perceptrons. In 1989, Yann LeCun and others applied the BP algorithm to handwritten digital image recognition and achieved great success.

[①] Picture source: https://slideplayer.com/slide/12771753/

[②] Picture source: https://www.glass-bead.org/article/machines-that-morph-logic/?lang=enview

This system was successfully commercialized in zip code recongition, bank check recognition and other systems. In 1997, one of the most widely used recurrent neural network variants, LSTM, was proposed by Jürgen Schmidhuber. In the same year, a bi-directional recurrent neural network was also proposed.

Unfortunately, the study of neural networks has gradually entered a trough with the rise of traditional machine learning algorithms represented by Support Vector Machines (SVM), which is known the second winter of artificial intelligence. Support Vector Machines have a rigorous theoretical foundation, requires a small number of training samples, and also have good generalization capabilities. In contrast, neural networks lack theoretical foundation and are hard to interpret. Deep networks are difficult to train, and the performance is normal. Figure Figure 0.7 shows significant time nodes of AI development between 1943 and 2006.

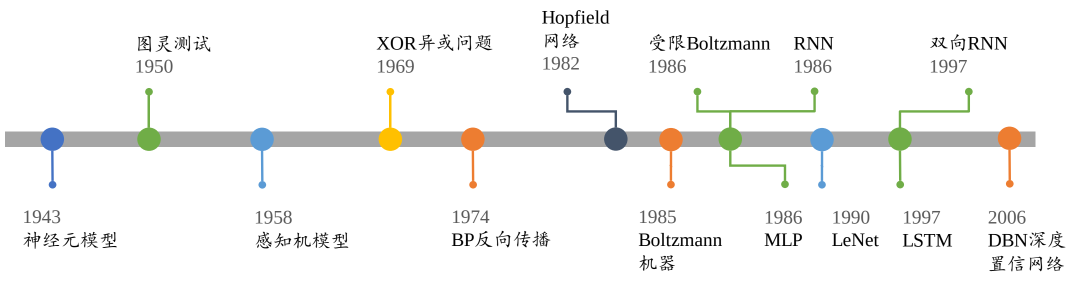


Figure 0.7 Shallow neural network development timeline

1.2.2 Deep Learning

In 2006, Geoffrey Hinton et al. found that multi-layer neural networks can be better trained through layer-by-layer pre-training, and achieved an error rate better than SVM on the MNIST handwritten digital picture data set, turning on the third artificial intelligence revival. In that paper, Geoffrey Hinton first proposed the concept of deep learning, which is also the origin of deep neural networks called deep learning. In 2011, Xavier Glorot proposed a Rectified Linear Unit (ReLU) activation function, which is one of the most widely used activation functions now. In 2012, Alex Krizhevsky proposed 8-layer deep neural network AlexNet, which used the ReLU activation function and Dropout technology to prevent overfitting. At the same time, it abandoned the layer-by-layer pre-training method and directly trained the network on two NVIDIA GTX580 GPUs. AlexNet won the first place in the ILSVRC-2012 picture recognition competition, showing a stunning 10.9% reduction in the top-5 error rate compared to the second place.

Since the AlexNet model was developed, various models have been published successively, including VGG series, GoogLeNet series, ResNet series, and DenseNet series. The ResNet series models increase the number of layers in the network to hundreds or even thousands, while maintaining the same or even better performance. Its algorithm is simple, universal, and has significant performance, which is the most representative model of deep learning.

In addition to the amazing results in supervised learning, huge achievements have also been made in unsupervised learning and reinforcement learning. In 2014, Ian Goodfellow proposed Generating Adversarial Networks, which learned the true distribution of samples through adversarial training to generate samples with higher approximation. Since then, a large number of Generative Adversarial Network models have been proposed. The latest image generation models can generate images that reach a degree of fidelity hard to discern from the naked eye. In 2016, DeepMind applied deep neural networks to the field of reinforcement learning and proposed the DQN algorithm, which achieved a level comparable to or even higher than that of humans in 49 games in the Atari game platform. In the field of Go, AlphaGo and AlphaGo Zero intelligent programs from DeepMind have successively defeated human top Go experts Li Shishi and Ke Jie etc. In the multi-agent collaboration Dota2 game platform, OpenAI Five intelligent programs developed by OpenAI defeated the TI8 champion team OG in a restricted game environment, showing a large number of professional high-level intelligent operations. Figure 0.8 lists the major time nodes between 2006 and 2019 for AI development.

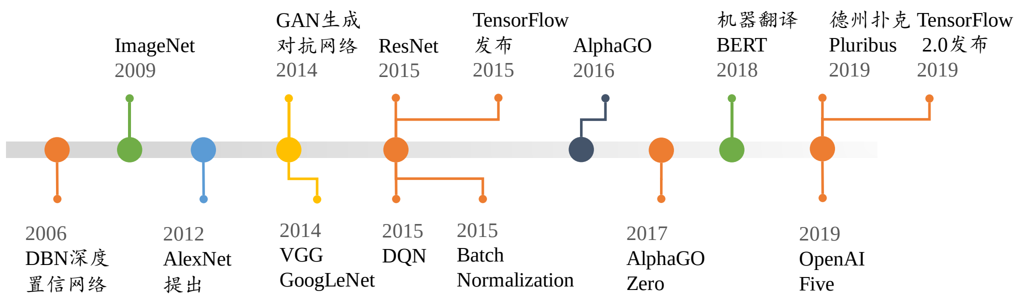


Figure 0.8 Timeline for deep learning development

1.3 Deep Learning Features

Compared with traditional machine learning algorithms and shallow neural networks, modern deep learning algorithms usually have the following characteristics.

1.3.1 Data Volume

Early machine learning algorithms are relatively simple, fast to train, and the size of the required dataset is relatively small, such as the Iris flower data set collected by the British statistician Ronald Fisher in 1936, which contains only 3 categories of flowers with each category having 50 samples. With the development of computer technology, the designed algorithms are more and more complex, and the demand for data volume is also increasing. The MNIST handwritten digital picture data set collected by Yann LeCun in 1998 contains a total of 10 categories of numbers from 0 to 9, with up to 7,000 pictures in each category. With the rise of neural networks, especially deep learning networks, the number of network layers is generally deep, and the number of model parameters can reach one million, ten million, or even one billion. To prevent overfitting, the size of the training dataset is usually huge. The popularity of modern social media also makes it possible to collect huge amounts of data. For example, the ImageNet dataset released in 2010 included a total of 14,197,122 pictures, and the compressed file size of the entire dataset was 154GB. Figure 0.9 and Figure 0.10 list the number of samples and the size of the data set over time.

Although deep learning has a high demand for large datasets, collecting data, especially collecting labeled data, is often very expensive. The formation of a dataset usually requires manual collection, crawling of raw data, and cleaning out invalid samples, and then annotating the data samples with human intelligence, so subjective bias and random errors are inevitably introduced. Therefore, algorithms with small data volume requirement are very hot topics.

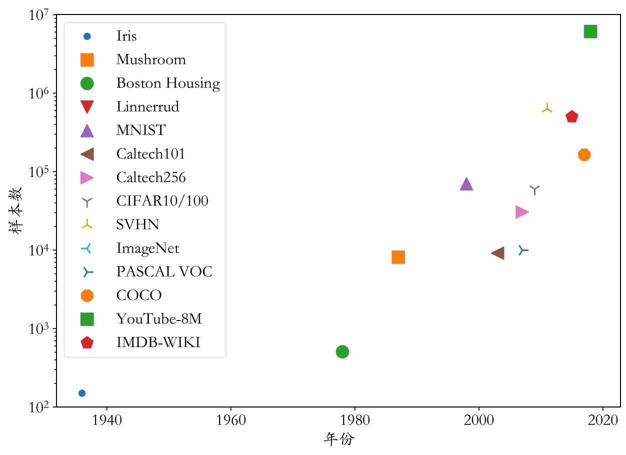


Figure 0.9 Dataset sample size change over time

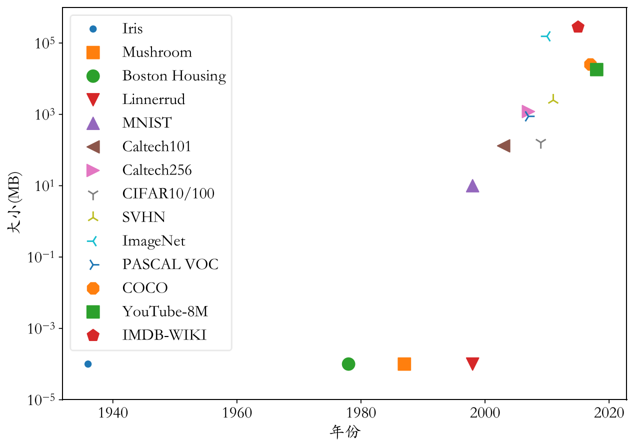


Figure 0.10 Dataset size change over time

1.3.2 Computing Power

The increase in computing power is an important factor in the third artificial intelligence renaissance. In fact, the basic theory of modern deep learning was proposed in the 1980s, but the real potential of deep learning was not realized until the release of AlexNet based on training on two GTX580 GPU in 2012. Traditional machine learning algorithms do not have stringent requirements on data volume and computing power like neural networks. Usually, serial training on CPU can get satisfactory results. But deep learning relies heavily on parallel acceleration computing devices. Most of current neural networks use parallel acceleration chips such as NVIDIA GPU and Google TPU to train model parameters. For example, the AlphaGo Zero program trained on 64 GPUs from scratch for 40 days before being able to surpass all AlphaGo historical versions; the automatic network structure search algorithm used 800 GPUs to optimize a better network structure.

At present, the deep learning acceleration hardware devices that ordinary consumers can use are mainly from NVIDIA GPU graphics cards. Figure 0.11 illustrates the variation of 1 billion floating point operations per second (GFLOPS) of NVIDIA GPU and x86 CPU from 2008 to 2017. It can be seen that the curve of x86 CPU changes relatively slowly, and the floating-point computing capacity of NVIDIA GPU grows exponentially, which is mainly driven by the increasing business of game and deep learning computing.

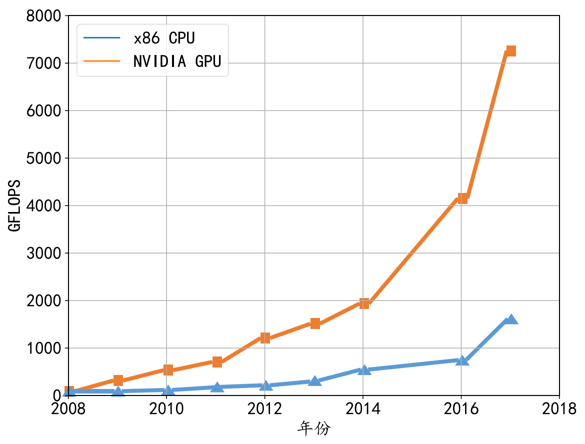


Figure 0.11 NVIDIA GPU FLOPS change (data source: NVIDIA)

1.3.3 Network Scale

Early perceptron models and multi-layer neural networks only have 1 or 2~4 layers, and the network parameters are also around tens of thousands. With the development of deep learning and the improvement of computing capabilities, models such as AlexNet (8 layers), VGG16 (16 layers), GoogLeNet (22 layers), ResNet50 (50 layers), DenseNet121 (121 layers) have been proposed successively, while the size of inputting pictures has also gradually increased from 28x28, to 224x224, 299x299 and even larger. These changes make the total number of parameters of the network reach 10 million levels, as shown in Figure 0.12.

The increase of network scale enhances the capacity of the neural network correspondingly, so that the network can learn more complex data modalities, and the performance of the model will be improved accordingly. On the other hand, the increase of the network scale also means that we need more training data and computational power to avoid overfitting.

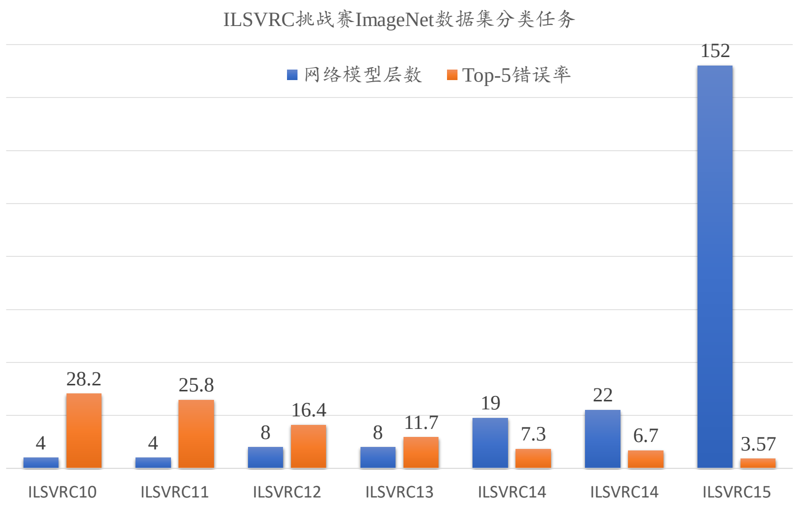


Figure 0.12 Change of network layers

1.3.4 General Intelligence

In the past, in order to improve the performance of an algorithm on a certain task, it is often necessary to use prior knowledge to manually design corresponding features to help the algorithm better converge to the optimal solution. This type of feature extraction method is often strongly related to the specific task. Once the scenario changes, these artificially designed features and prior settings cannot adapt to the new scenario, and people often need to redesign the algorithms.

Designing a universal intelligent mechanism that can automatically learn and self-adjust like the human brain has always been the common vision of human beings. From the current perspective, deep learning is one of the algorithms closest to general intelligence. In the field of computer vision, previous methods that need to design features for specific tasks and add a priori assumptions has been abandoned by deep learning algorithms. At present, almost all algorithms in image recognition, object detection, and semantic segmentation are based on end-to-end deep learning models, which present good performance and strong adaptability. On the Atria game platform, the DQN algorithm designed by DeepMind can reach human equivalent level in 49 games under the same algorithm, model structure and hyperparameter settings, showing a certain degree of general intelligence. Figure 0.13 is the network structure of the DQN algorithm. It is not designed for a certain game, but can control 49 games on the Atria game platform.

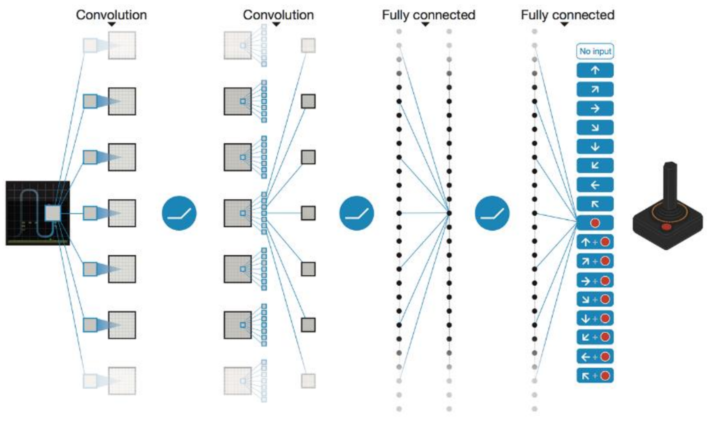


Figure 0.13 DQN network structure [1]

1.4 Deep Learning Applications

Deep learning algorithms have been widely used in our daily life, such as voice assistants in mobile phones, intelligent assisted driving in cars, face payments, etc. We will introduce some mainstream applications of deep learning starting with computer vision, natural language processing, and reinforcement learning.

1.4.1 Computer Vision

Image classification is a common classification problem. The input of the neural network is pictures, and the output value is the probability that the current sample belongs to each category. Generally, the category with the highest probability is selected as the predicted category of the sample. Image recognition is one of the earliest successful applications of deep learning. Classic neural network models include VGG series, Inception series, and ResNet series.

Object detection refers to the automatic detection of the approximate location of common objects in a picture by an algorithm. It is usually represented by a bounding box and classifies the category information of objects in the bounding box, as shown in Figure 0.14. Common object detection algorithms are RCNN, Fast RCNN, Faster RCNN, Mask RCNN, SSD, and YOLO series.

Semantic Segmentation is an algorithm to automatically segment and identify the content in a picture. We can understand semantic segmentation as the classification of each pixel and analyze the category information of each pixel, as shown in Figure 0.15. Common semantic segmentation models include FCN, U-net, SegNet, and DeepLab series.

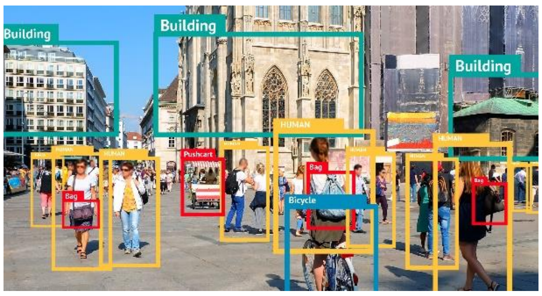


Figure 0.14 Object detection example

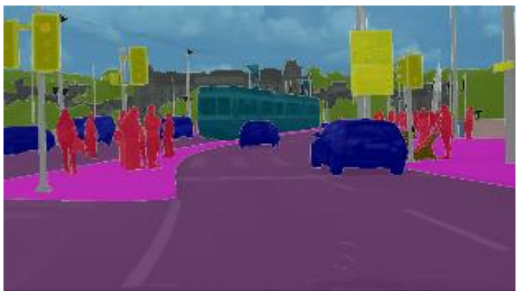


Figure 0.15 Semantic segmentation example

Video Understanding As deep learning achieves better results on 2D picture-related tasks, 3D video understanding tasks with temporal dimension information are receiving more and more attention. Common video understanding tasks include video classification, behavior detection, and video subject extraction. Common models are C3D, TSN, DOVF, and TS_LSTM.

Image Generation learns the distribution of real pictures and samples from the learned distribution to obtain highly realistic generated pictures. At present, common image generation models include VAE series and GAN series. Among them, the GAN series of algorithms have made great progress in recent years. The picture effect produced by the latest GAN model has reached a level where it is difficult to distinguish the authenticity with the naked eye, as shown in Figure 0.16.

In addition to the above applications, deep learning has also achieved significant results in other areas, such as artistic style transfer (Figure 0.17), super-resolution, picture de-nosing / hazing, gray-scale picture coloring, and many others.

Figure 0.16 Model generated image

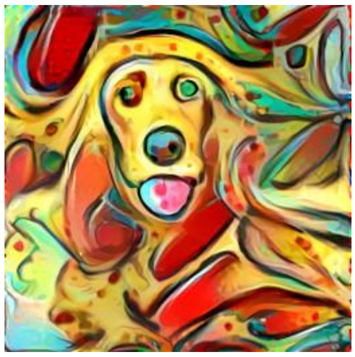


Figure 0.17 Artistic style transfer image

1.4.2 Natural Language Processing

Machine Translation In the past, machine translation algorithms were usually based on statistical machine translation models, which is also the technology used by Google's translation system before 2016. In November 2016, Google launched the Google Neural Machine Translation System (GNMT) based on the Seq2Seq model, and for the first time realized the direct translation technology from source language to target language, and achieved 50 ~ 90% effect improvement on multiple tasks. Commonly used machine translation models are Seq2Seq, BERT, GPT and GPT-2. Among them, the GPT-2 model proposed by OpenAI has about 1.5 billion parameters. At the beginning, OpenAI refused to open source the GPT-2 model due to technical security reasons.

Chatbot is also a mainstream task of natural language processing. Machines automatically learn to talk to humans, provide satisfactory automatic responses to simple human demands, and improve customer service efficiency and service quality. Chatbot is often used in consulting systems, entertainment systems and smart homes.

1.4.3 Reinforcement Learning

Virtual Games Compared to the real environment, virtual game platforms can both train and test reinforcement learning algorithms, and can avoid interference from irrelevant factors, while also minimizing the cost of experiments. Currently, commonly used virtual game platforms include OpenAI Gym, OpenAI Universe, OpenAI Roboschool, DeepMind OpenSpiel and MuJoCo, and commonly used reinforcement learning algorithms include DQN, A3C, A2C and PPO. In the field of Go, the DeepMind AlaphGo program has surpassed human Go experts. In Dota2 and StarCraft games, the intelligent programs developed by OpenAI and DeepMind have also defeated professional teams under restriction rules.

Robotics In the real environment, the control of robots has also made some progress. For example, UC Berkeley Lab has made a lot of progress in the areas of Imitation Learning, Meta Learning, Few-shot Learning in the field of robotics. Boston Dynamics has made gratifying achievements in robot applications. The robots it manufactures perform well on tasks such as complex terrain walking and multi-agent collaboration (Fgiure 0.18).

Autonomous Driving is considered as an application direction of reinforcement learning in the short term. Many companies have invested a lot of resources in autonomous driving, such as

Baidu, Uber, and Google. Appollo from Baidu has begun trial operations in Beijing, Xiong'an, Wuhan and other places. Figure 0.19 shows Baidu's self-driving car Appollo.

Fgiure 0.18 Robots from Boston Dynamics[3]

Figure 0.19 Baidu Apollo Self-Driving Car[4]

1.5 Deep Learning Framework

If a workman wants to be good, he must first sharpen his weapon. After learning about the basic knowledge of deep learning, let's pick the tools used to implement deep learning algorithms.

1.5.1 Major Frameworks

- ❑ Theano is one of the earliest deep learning frameworks. It was developed by Yoshua Bengio and Ian Goodfellow. It is a Python-based computing library for positioning low-level operations. Theano supports both GPU and CPU operations. Due to Theano's low development efficiency, long model compilation time, and developers switching to TensorFlow, etc., Theano has now stopped maintenance.
- ❑ Scikit-learn is a complete computing library for machine learning algorithms. It has built-in support for common traditional machine learning algorithms, and it has rich documentation and examples. However, Scikit-learn is not specifically designed for neural networks. It does not support GPU acceleration, and the implementation of neural network related layers is also lacking.
- ❑ Caffe was developed by Jia Yangqing in 2013. It is mainly used for applications using convolutional neural networks, and is not suitable for other types of neural networks. Caffe's main development language is C ++, and also provides interfaces for other languages such as Python. It also supports GPU and CPU. Due to the earlier development time and higher visibility in the industry, in 2017 Facebook launched an upgraded version of Caffe, Cafffe2. Caffe2 has now been integrated into the PyTorch library.
- ❑ Torch is a very good scientific computing library, developed based on the less popular

[3] Picture source: https://www.bostondynamics.com/

[4] Picture source: https://venturebeat.com/2019/01/08/baidu-announces-apollo-3-5-and-apollo-enterprise-says-it-has-over-130-partners/

programming language Lua. Torch is highly flexible and easy to implement a custom network layer, which is also an excellent gene inherited by PyTorch. However, due to the small number of Lua language users, Torch has been unable to obtain mainstream applications.

- ❑ MXNet was developed by Chen Tianqi and Li Mu, and is the official deep learning framework of Amazon. It adopts a mixed method of imperative programming and symbolic programming, which has high flexibility, fast running speed, and rich documentation and examples.
- ❑ PyTorch is a deep learning framework launched by Facebook based on the original Torch framework using Python as the main development language. PyTorch borrowed the design style of Chainer and adopted imperative programming, which made it very convenient to build and debug the network. Although PyTorch was only released in 2017, due to its sophisticated and compact interface design, PyTorch has received wide acclaim in the academic world. After the 1.0 version, the original PyTorch and Caffe2 were merged to make up for PyTorch's deficiencies in industrial deployment. Overall, PyTorch is an excellent deep learning framework.
- ❑ Keras is a high-level framework implemented based on the underlying operations provided by frameworks such as Theano and TensorFlow. It provides a large number of high-level interfaces for rapid training and testing. For common applications, developing with Keras is very efficient. But because there is no low-level implementation, the underlying framework needs to be abstracted, so the operation efficiency is not high, and the flexibility is average.
- ❑ TensorFlow is a deep learning framework released by Google in 2015. The initial version only supported symbolic programming. Thanks to its earlier release and Google's influence in the field of deep learning, TensorFlow quickly became the most popular deep learning framework. However, due to frequent changes in the interface design, redundant functional design, and difficulty in symbolic programming development and debugging, TensorFlow 1.x was once criticized by the industry. In 2019, Google launched the official version of TensorFlow 2, which runs in dynamic graph priority mode and can avoid many defects of TensorFlow 1.x version. Tensorflow 2 has been widely recognized by the industry.

At present, TensorFlow and PyTorch are the two most widely used deep learning frameworks in industry. TensorFlow has a complete solution and user base in the industry. Thanks to its streamlined and flexible interface design, PyTorch can quickly build and debug networks, which has received rave reviews in academia. After TensorFlow 2 was released, it makes TensorFlow easier for users to learn and seamlessly deploy models to production. This book uses TensorFlow 2.0 as the main framework to implement deep learning algorithms.

Here are the connections and differences between TensorFlow and Keras. Keras can be understood as a set of high-level API design specifications. Keras itself has an official implementation of the specifications. The same specifications is also implemented in TensorFlow, which is called the tf.keras module, and tf.keras will be used as the unique high-level interface to avoid duplicate interface redundancy. Unless otherwise specified, Keras in this book refers to tf.keras.

1.5.2 TensorFlow 2 and 1.x

TensorFlow 2 is a completely different framework from TensorFlow 1.x in terms of user experience. TensorFlow 2 is not compatible with TensorFlow 1.x code. At the same time, it is very different in programming style and function interface design. TensorFlow 1.x code needs to rely on artificial migration, and automated migration methods are not reliable. Google is about to stop updating TensorFlow 1.x. It is not recommended to learn TensorFlow 1.x.

TensorFlow 2 supports the dynamic graph priority mode. You can obtain both the computation graph and the numerical results during the calculation. You can debug the code and print the data in real time. The network is built like a building block, stacked layer by layer, which is in line with software development thinking.

Taking simple addition 2.0 + 4.0 as an example, in TensorFlow 1.x, first we need to create a calculation graph, the code is as follows:

```
import tensorflow as tf
# 1. Create computation graph with tf 1.x
# Create 2 input variables with fixed name and type
a_ph = tf.placeholder(tf.float32, name='variable_a')
b_ph = tf.placeholder(tf.float32, name='variable_b')
# Create output operation and name

c_op = tf.add(a_ph, b_ph, name='variable_c')
```

The process of creating a computation graph is analogous to the process of establishing a formula $c = a + b$ through symbols. It only records the computation steps of the formula and does not actually calculate the numerical results. The numerical results can only be obtained by running the output terminal c and assigning values $a = 2.0, b = 4.0$. The code is as follows:

```
# 2.Run computation graph with tf 1.x
# Create running environment
sess = tf.InteractiveSession()
# Initialization
init = tf.global_variables_initializer()
sess.run(init) # Run the initialization
# Run the computation graph and return value to c_numpy
c_numpy = sess.run(c_op, feed_dict={a_ph: 2., b_ph: 4.})
# print out the output

print('a+b=',c_numpy)
```

It can be seen that it is so tedious to perform simple addition operations in TensorFlow, let alone how difficult it is to create complex neural network algorithms. This programming method of creating a computation graph and then running it is called symbolic programming.

Next we use TensorFlow 2 to complete the same operation, the code is as follows:

```
import tensorflow as tf
# Use tf 2 to run
# 1.Create and initialize variable
a = tf.constant(2.)
```

```
b = tf.constant(4.)
# 2.Run and get result directly
print('a+b=',a+b)
```

As you can see, the calculation process is very simple and there are no extra calculation steps.

The method of getting both computation graphs and numerical results at the same time is called imperative programming, also known as dynamic graph mode. TensorFlow 2 and PyTorch are both developed using dynamic graph (priority) mode, which is easy to debug and WYSIWYG. In general, the dynamic graph mode is highly efficient for development, but it may not be as efficient as the static graph mode for running. TensorFlow 2 also supports converting the dynamic graph mode to the static graph mode through tf.function, achieving a win-win situation of development and operation efficiency.

1.5.3 Demo

The core of deep learning is the design idea of algorithms, and deep learning frameworks are just our tools for implementing algorithms. Below we will demonstrate the three core functions of the TensorFlow deep learning framework to help us understand the role of frameworks in algorithm design.

a) Accelerated Calculation

The neural network is essentially composed of a large number of basic mathematical operations such as matrix multiplication and addition. One important function of TensorFlow is to use the GPU to conveniently implement parallel computing acceleration functions. In order to demonstrate the acceleration effect of GPU, we can compare mean running time for multiple matrix multiplications on CPU and GPU as follow.

We create two matrices A and B with shape [1, n] and [n, 1] separately. The size of the matrices can be adjusted using parameter n. The code is as follows:

```
# Create two matrices running on CPU
with tf.device('/cpu:0'):
    cpu_a = tf.random.normal([1, n])
    cpu_b = tf.random.normal([n, 1])
    print(cpu_a.device, cpu_b.device)
# Create two matrices running on GPU
with tf.device('/gpu:0'):
    gpu_a = tf.random.normal([1, n])
    gpu_b = tf.random.normal([n, 1])
    print(gpu_a.device, gpu_b.device)
```

Next, implement the functions of the CPU and GPU operations, and measure the operation time of the two functions through the timeit.timeit () function. It should be noted that additional environment initialization work is generally required for the first calculation, so this time cannot be counted. We remove this time through the warm-up session, and then measure the calculation time, the code is as follows:

```
def cpu_run(): # CPU function
    with tf.device('/cpu:0'):
        c = tf.matmul(cpu_a, cpu_b)
    return c

def gpu_run():# GPU function
    with tf.device('/gpu:0'):
        c = tf.matmul(gpu_a, gpu_b)
    return c
# First calculation needs warm-up
cpu_time = timeit.timeit(cpu_run, number=10)
gpu_time = timeit.timeit(gpu_run, number=10)
print('warmup:', cpu_time, gpu_time)
# Calculate and print mean running time
cpu_time = timeit.timeit(cpu_run, number=10)
gpu_time = timeit.timeit(gpu_run, number=10)
print('run time:', cpu_time, gpu_time)
```

We plot the computation time under CPU and GPU environments at different matrix size as shown in Figure 0.20. It can be seen that when the matrix size is small, the CPU and GPU time are almost the same, which does not reflect the advantages of GPU parallel computing. When the matrix size is larger, the CPU computing time significantly increases, and the GPU takes full advantage of parallel computing without almost any change of computation time.

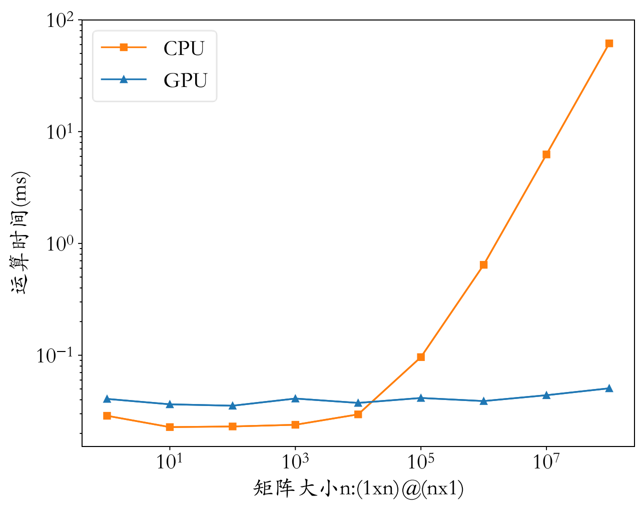


Figure 0.20 CPU/GPU matrix multiplication time

b) Automatic Gradient Calculation

When using TensorFlow to construct the forward calculation process, in addition to being able to obtain numerical results, TensorFlow also automatically builds a computation graph. TensorFlow provides automatic differentiation that can calculate the derivative of the output on network parameters without manual derivation. Consider the expression of the following function:

$$y = aw^2 + bw + c$$

The derivative relationship of the output y to the variable w is:

$$\frac{\mathrm{d}y}{\mathrm{d}w} = 2aw + b$$

Consider the derivative at $(a, b, c, w) = (1,2,3,4)$, we can get $\frac{\mathrm{d}y}{\mathrm{d}w} = 2 \cdot 1 \cdot 4 + 2 = 10$.

With TensorFlow, we can directly calculate the derivative given the expression of a function without manually deriving the expression of the derivative. TensorFlow can automatically derive it. The code is implemented as follows:

```
import tensorflow as tf
# Create 4 tensors
a = tf.constant(1.)
b = tf.constant(2.)
c = tf.constant(3.)
w = tf.constant(4.)

with tf.GradientTape() as tape:# Track derivative
    tape.watch([w]) # Add w to derivative watch list
    # Design the function
    y = a * w**2 + b * w + c
# Auto derivative calculation
[dy_dw] = tape.gradient(y, [w])
print(dy_dw) # print the derivative
```

The result of the program is:

```
tf.Tensor(10.0, shape=(), dtype=float32)
```

It can be seen that the result of TensorFlow's automatic differentiation is consistent with the result of manual calculation.

c) Common Neural Network Interface

In addition to the underlying mathematical functions such as matrix multiplication and addition, TensorFlow also has a series of convenient functions for deep learning systems such as commonly used neural network operation functions, commonly used network layers, network training, model saving, loading, and deployment. Using TensorFlow development, you can easily use these functions to complete common production processes, which is efficient and stable.

1.6 Development Environment Installation

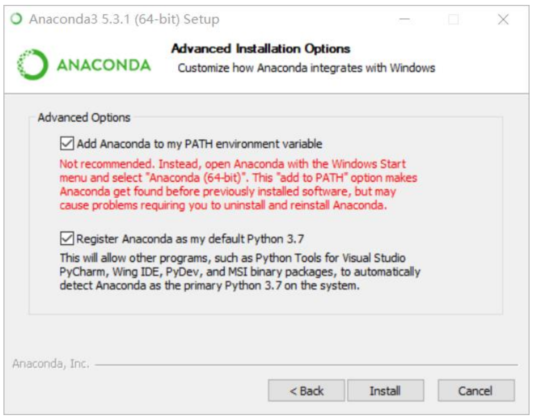
After knowing the convenience brought by the deep learning framework, we are now ready to install the latest version of TensorFlow in the local desktop. TensorFlow supports a variety of common operating systems, such as Windows 10, Ubuntu 18.04 and Mac OS. It supports both GPU version running on NVIDIA graphics cards and CPU version that use only the CPU to do calculations. We take the most common operational system - Windows 10, NVIDIA GPU and Python as examples to introduce how to install the TensorFlow framework and other development software.

Generally speaking, the development environment installation is divided into 4 major steps: the Python interpreter - Anaconda, the CUDA acceleration library, the TensorFlow framework, and commonly used editors.

1.6.1 Anaconda Installation

The Python interpreter is the bridge that allows code written in Python to be executed by CPU, and is the core software of the Python language. Users can download the latest version (Python 3.7) of the interpreter from https://www.python.org/. After the installation is completed, you can call the python.exe program to execute the source code file written in Python. (.py files).

Here we choose to install Anaconda software that integrates a series of auxiliary functions such as the Python interpreter, package management, and virtual environment. We can download Anaconda from https://www.anaconda.com/distribution/#download-section, select the latest version of Python to download and install. As shown in Figure 0.21, check the "Add Anaconda to my PATH environment variable" option, so that you can call the Anaconda program through the command line. As shown in Figure 0.22, the installer asks whether to install the VS Code software together. Select Skip. The entire installation process lasts about 5 minutes, and the specific time depends on the computer performance.



Anaconda3 5.3.1 (64-bit) Setup

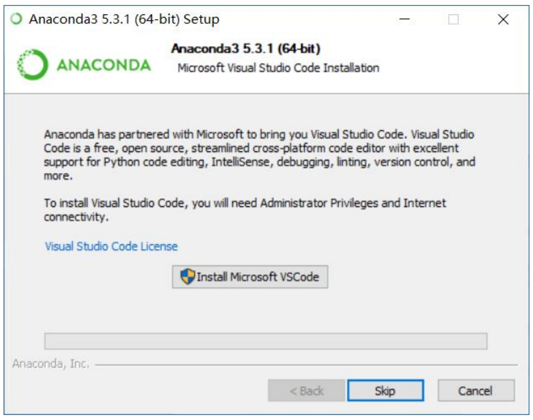
Advanced Installation Options
Customize how Anaconda integrates with Windows

Advanced Options

[✓] Add Anaconda to my PATH environment variable
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

[✓] Register Anaconda as my default Python 3.7
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.

Figure 0.21 Anaconda installation-1



Anaconda3 5.3.1 (64-bit) Setup

Anaconda3 5.3.1 (64-bit)
Microsoft Visual Studio Code Installation

Anaconda has partnered with Microsoft to bring you Visual Studio Code. Visual Studio Code is a free, open source, streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more.

To install Visual Studio Code, you will need Administrator Privileges and Internet connectivity.

Figure 0.22Anaconda installation-2

After the installation is complete, how can we verify that Anaconda was successfully installed? Pressing Windows key + R key on the keyboard, you can bring up the running program dialog box, enter "cmd" and press “Enter” to open the command line program “cmd.exe” that comes with Windows. Or click the start menu and enter "cmd" to find the “cmd.exe” program and open it. Enter the “conda list” command to view the installed libraries in the Python environment. If it is a newly installed Python environment, the listed libraries are all libraries that come with Anaconda, as shown in Figure 0.23. If the “conda list” can pop up a series of library list information normally, the Anaconda software installation is successful. Otherwise, the installation failed, and you need to reinstall.

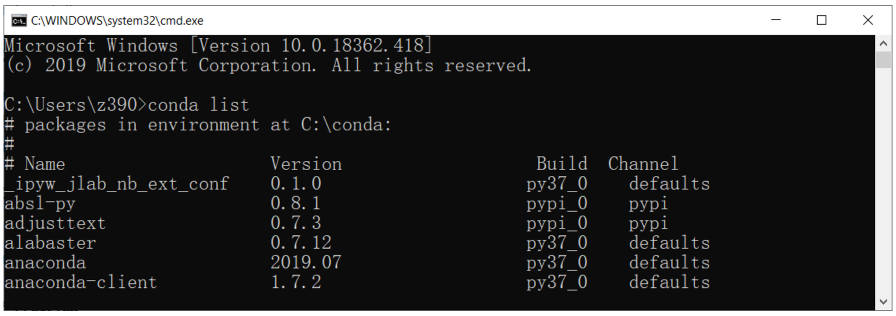

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\z390>conda list
# packages in environment at C:\conda:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0                    py37_0    defaults
absl-py                   0.8.1                    pypi_0    pypi
adjusttext                0.7.3                    pypi_0    pypi
alabaster                 0.7.12                   py37_0    defaults
anaconda                  2019.07                  py37_0    defaults
anaconda-client           1.7.2                    py37_0    defaults
```

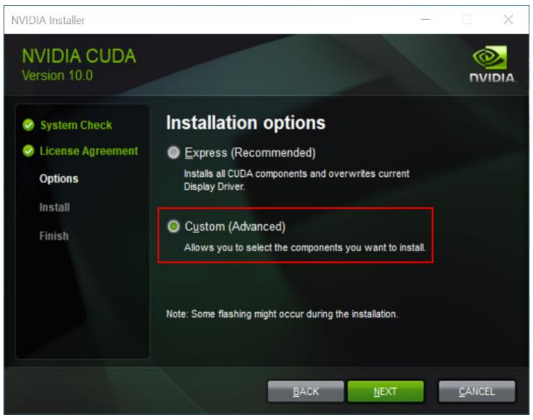
Figure 0.23 Anaconda installation test

1.6.2 CUDA Installation

Most of the current deep learning frameworks are based on NVIDIA's GPU graphics card for accelerated calculations, so you need to install the GPU acceleration library provided – CUDA provided by NVIDIA. Before installing CUDA, make sure your computer has an NVIDIA graphics device that supports the CUDA program. If your computer does not have an NVIDIA graphics card, for example, some computer graphics card manufacturers are AMD or Intel, you cannot install the CUDA program, so you can skip this step and directly enter the installation of the TensorFlow CPU version.

The installation of CUDA is divided into three steps: CUDA software installation, cuDNN deep neural network acceleration library installation, and environment variable configuration. The installation process is a bit tedious. We will go through them step by step using the Windows 10 system as an example.

CUDA software installation Open the official downloading website of the CUDA program: https://developer.nvidia.com/cuda-10.0-download-archive. Here we use CUDA 10.0 version: select Windows platform, x86_64 architecture, 10 system, exe (local) installation package, and then select “Download” to download the CUDA installation software. After the download is complete, open the software. As shown in Figure 0.22, select the "Custom" option and click the “NEXT” button to enter the installation program selection list as shown in Figure 0.25. Here you can select the components that need to be installed and unselect that do not need to be installed. Under the “CUDA” category, unselect the "Visual Studio Integration" item. Under the "Driver components" category, compare the version number "Current Version" and “New Version” at the “Display Driver” row. If "Current Version" is greater than "New Version", you need to uncheck the "Display Driver". If "Current Version" is less than or equal to "New Version", leave “Display Drive” checked, as shown in Figure 0.26. After the setup is complete, you can click “Next” and follow the instructions to install.



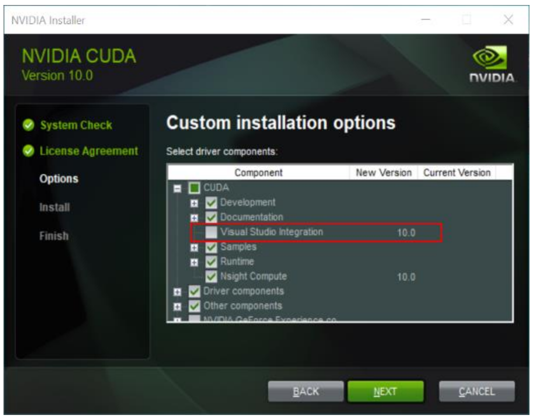
Installation options

Express (Recommended)
Installs all CUDA components and overwrites current Display Driver.

[✓ Custom (Advanced)]
Allows you to select the components you want to install.

Note: Some flashing might occur during the installation.

Figure 0.24 CUDA installation-1



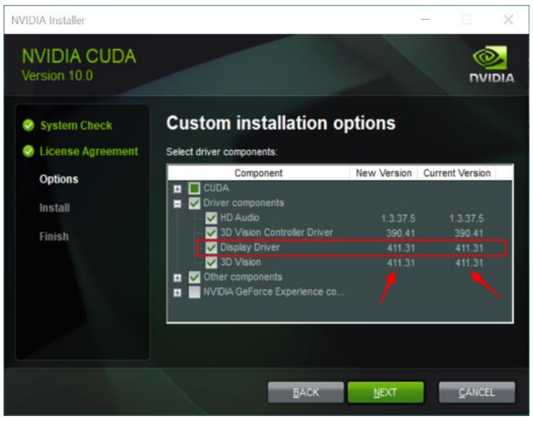
Custom installation options

Select driver components:

Component	New Version	Current Version
☐ CUDA		
✓ Development		
✓ Documentation		
☐ Visual Studio Integration	10.0	
✓ Samples		
✓ Runtime		
✓ Nsight Compute	10.0	
✓ Driver components		
✓ Other components		

Figure 0.25 CUDA installation-2

After the installation is complete, let's test whether the CUDA software is successfully installed. Open the “cmd” terminal and enter "nvcc -V" to print the current CUDA version information, as shown in Figure 0.28. If the command is not recognized, the installation has failed. We can find the "nvcc.exe" program from the CUDA installation path "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin", as shown in Figure 0.27.

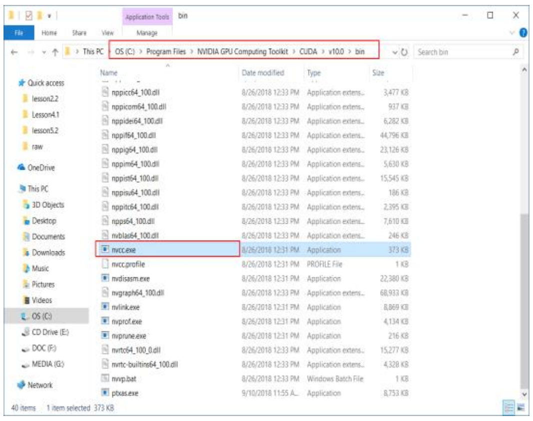


Custom installation options

Select driver components:

Component	New Version	Current Version
☐ CUDA		
✓ Driver components		
✓ HD Audio	1.3.37.5	1.3.37.5
✓ 3D Vision Controller Driver	390.41	390.41
✓ Display Driver	411.31	411.31
✓ 3D Vision	411.31	411.31
✓ Other components		
☐ NVIDIA GeForce Experience co...		

Figure 0.26 CUDA installation-3

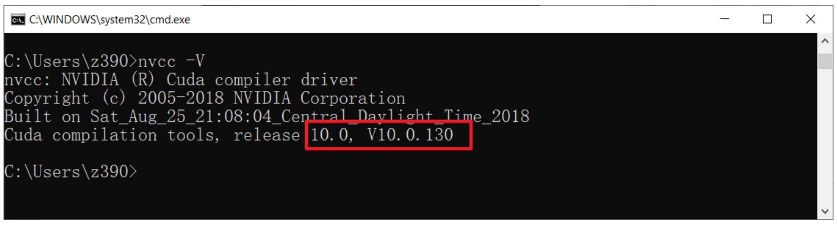


OS (C:) › Program Files › NVIDIA GPU Computing Toolkit › CUDA › v10.0 › bin

Name	Date modified	Type	Size
nppicc64_100.dll	8/26/2018 12:33 PM	Application extens...	3,477 KB
nppicom64_100.dll	8/26/2018 12:33 PM	Application extens...	937 KB
nppidei64_100.dll	8/26/2018 12:33 PM	Application extens...	6,282 KB
nppif64_100.dll	8/26/2018 12:33 PM	Application extens...	44,796 KB
nppig64_100.dll	8/26/2018 12:33 PM	Application extens...	23,126 KB
nppim64_100.dll	8/26/2018 12:33 PM	Application extens...	5,630 KB
nppist64_100.dll	8/26/2018 12:33 PM	Application extens...	15,545 KB
nppisu64_100.dll	8/26/2018 12:33 PM	Application extens...	186 KB
nppitc64_100.dll	8/26/2018 12:33 PM	Application extens...	2,395 KB
npps64_100.dll	8/26/2018 12:33 PM	Application extens...	7,610 KB
nvblas64_100.dll	8/26/2018 12:33 PM	Application extens...	246 KB
nvcc.exe	8/26/2018 12:31 PM	Application	373 KB
nvcc.profile	8/26/2018 12:31 PM	PROFILE File	1 KB
nvdisasm.exe	8/26/2018 12:31 PM	Application	22,380 KB
nvgraph64_100.dll	8/26/2018 12:33 PM	Application extens...	68,933 KB
nvlink.exe	8/26/2018 12:31 PM	Application	8,869 KB
nvprof.exe	8/26/2018 12:31 PM	Application	4,134 KB
nvprune.exe	8/26/2018 12:31 PM	Application	216 KB
nvrtc64_100_0.dll	8/26/2018 12:33 PM	Application extens...	15,277 KB
nvrtc-builtins64_100.dll	8/26/2018 12:33 PM	Application extens...	4,328 KB
nvvp.bat	8/26/2018 12:33 PM	Windows Batch File	1 KB
ptxas.exe	9/10/2018 11:55 A...	Application	8,753 KB

40 items 1 item selected 373 KB

Figure 0.27 CUDA installation test-1



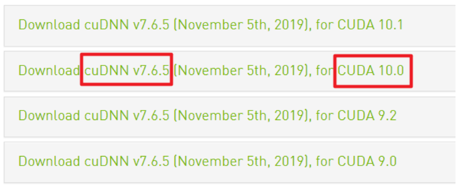
```
C:\Users\z390>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Sat_Aug_25_21:08:04_Central_Daylight_Time_2018
Cuda compilation tools, release 10.0, V10.0.130

C:\Users\z390>
```

Figure 0.28 CUDA installation test-2

cuDNN neural network acceleration library installation CUDA is not a special GPU acceleration library for neural networks, it is designed for a variety of applications that require parallel computing. If you want to accelerate for neural network applications, you need to install an additional cuDNN library. It should be noted that the cuDNN library is not an executable program. You only need to download and decompress the cuDNN file and configure the Path environment variable.

Open the website https://developer.nvidia.com/cudnn and select "Download cuDNN". Due to

NVIDIA regulations, users need to log in or create a new user to continue downloading. After logging in, enter the cuDNN download interface and check "I Agree To the Terms of the cuDNN Software License Agreement", and the cuDNN version download option will pop up. Select the cuDNN version that matches CUDA 10.0, and click the "cuDNN Library for Windows 10" link to download the cuDNN file, as shown in Figure 0.29. It should be noted that cuDNN itself has a version number, and it also needs to match the CUDA version number.



Download cuDNN v7.6.5 (November 5th, 2019), for CUDA 10.1

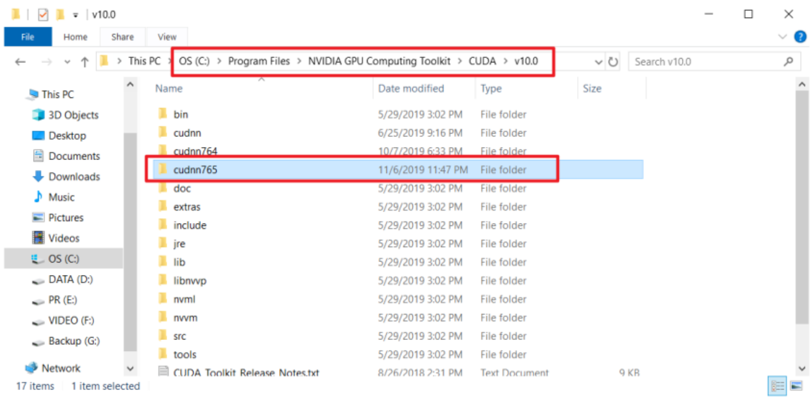
Download cuDNN v7.6.5 (November 5th, 2019), for CUDA 10.0

Download cuDNN v7.6.5 (November 5th, 2019), for CUDA 9.2

Download cuDNN v7.6.5 (November 5th, 2019), for CUDA 9.0

Figure 0.29 cuDNN version selection interface

After downloading the cuDNN file, unzip it and rename the folder "cuda" to "cudnn765". Then copy the “cudnn765” folder to the CUDA installation path “C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0” (Figure 0.30. A dialog box that requires administrator rights may pop up here. Select Continue to paste.



Name	Date modified	Type	Size
bin	5/29/2019 3:02 PM	File folder	
cudnn	6/25/2019 9:16 PM	File folder	
cudnn764	10/7/2019 6:33 PM	File folder	
cudnn765	11/6/2019 11:47 PM	File folder	
doc	5/29/2019 3:02 PM	File folder	
extras	5/29/2019 3:02 PM	File folder	
include	5/29/2019 3:02 PM	File folder	
jre	5/29/2019 3:02 PM	File folder	
lib	5/29/2019 3:02 PM	File folder	
libnvvp	5/29/2019 3:02 PM	File folder	
nvml	5/29/2019 3:02 PM	File folder	
nvvm	5/29/2019 3:02 PM	File folder	
src	5/29/2019 3:02 PM	File folder	
tools	5/29/2019 3:02 PM	File folder	
CUDA Toolkit Release Notes.txt	8/26/2018 2:31 PM	Text Document	9 KB

Figure 0.30 cuDNN installation path

Environment variable configuration We have completed the installation of cuDNN, but in order for the system to be aware of the location of the cuDNN file, we need to configure the path environment variable as follow. Open the file browser, right-click on "My Computer", select "Properties", select "Advanced System Properties", and select "Environment Variables", as shown in Figure 0.31. Select the "Path" environment variable in the "System variables" column and select "Edit", as shown in Figure 0.32. Select "New", enter the cuDNN installation path "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\cudnn765\bin", and use the "Move Up" button to move this item to the top.

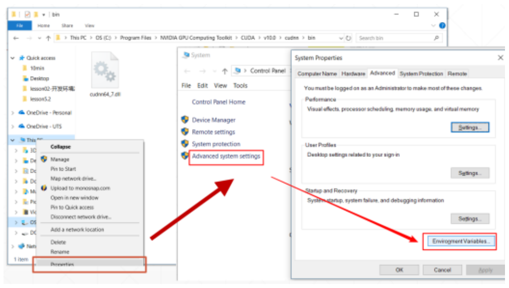


Figure 0.31 Environment variable configuration-1

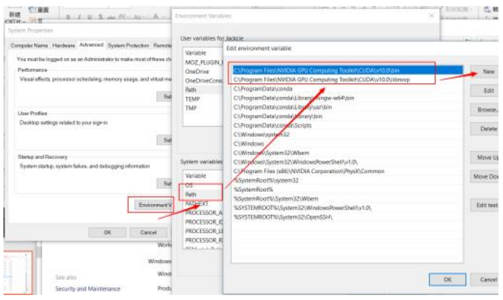


Figure 0.32 Environment variable configuration -2

After the CUDA installation is complete, the environment variables should include "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin", "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\libnvvp " and " C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\cudnn765\bin". The above path may differ slightly according to the actual path, as shown in Figure 0.33. After confirmation, click “OK” to close all dialog boxes.

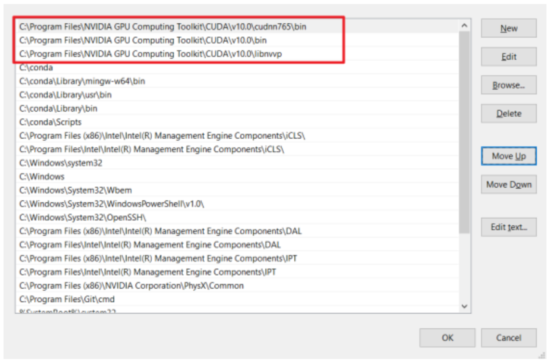

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\cudnn765\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\libnvvp
C:\conda
C:\conda\Library\mingw-w64\bin
C:\conda\Library\usr\bin
C:\conda\Library\bin
C:\conda\Scripts
C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\iCLS\
C:\Program Files\Intel\Intel(R) Management Engine Components\iCLS\
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL
C:\Program Files\Intel\Intel(R) Management Engine Components\DAL
C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT
C:\Program Files\Intel\Intel(R) Management Engine Components\IPT
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files\Git\cmd
```

Figure 0.33 CUDA related environment variables

1.6.3 TensorFlow Installation

TensorFlow 和其他的 Python 库一样，使用 Python 包管理工具 pip install 命令即可安装。安装 TensorFlow 时，需要根据电脑是否具有 NVIDIA GPU 显卡来确定是安装性能更强的 GPU 版本还是性能一般的 CPU 版本。

国内使用 pip 命令安装时，可能会出现下载速度缓慢甚至连接断开的情况，需要配置国内的 pip 源，只需要在 pip install 命令后面带上“-i 源地址”参数即可。例如使用清华源安装 numpy 包，首先打开 cmd 命令行程序，输入：

TensorFlow, like other Python libraries, can be installed using the Python package management tool “pip install” command. When installing TensorFlow, you need to determine whether to install a more powerful GPU version or a general-performance CPU version based on whether your computer has an NVIDIA GPU graphics card.

When using the “pip command” in China, the download speed may be slow or the connection may be disconnected. In that case, you need to configure the domestic pip source using the "pip install -i source address" command. For example, to install numpy package using Tsinghua source, first open the cmd command line program and enter:

```
# Install numpy using Tsinghua source
```

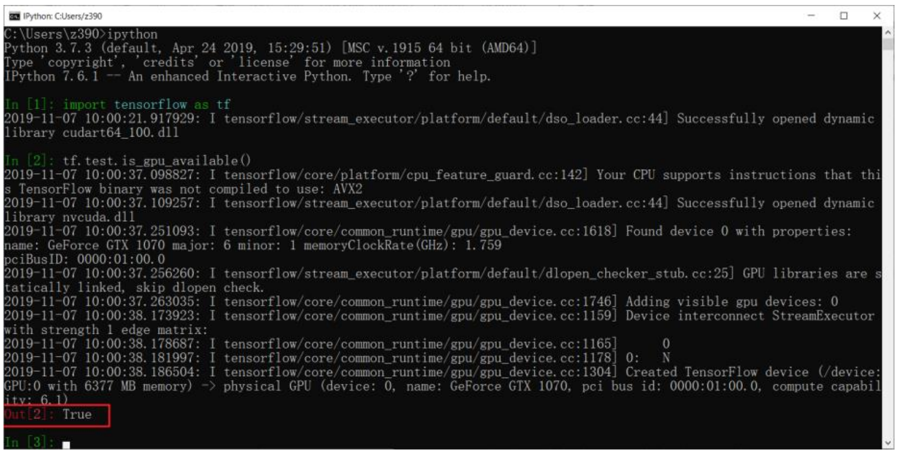
```
pip install numpy -i https://pypi.tuna.tsinghua.edu.cn/simple
```

With above command, you should be able to automatically download and install the numpy library, and the download speed will be significantly improved. Now let’s install the latest GPU version of TensorFlow, the command is as follow:

```
# Install TensorFlow GPU version using Tsinghua source
pip install -U tensorflow-gpu -i https://pypi.tuna.tsinghua.edu.cn/simple
```

The above command should automatically download and installs the TensorFlow GPU version, which is currently the official version of TensorFlow 2.x. The "-U" parameter specifies that if this package is installed, the upgrade command is executed.

Now let's test whether the GPU version of TensorFlow is successfully installed. Enter “ipython” on the “cmd” command line to enter the ipython interactive terminal, and then enter the "import tensorflow as tf" command. If no errors occur, continue to enter "tf.test.is_gpu_available ()" to test whether the GPU is available. This command will print a series of The information beginning with "I" (Information) contains information about the available GPU graphics devices, and will return "True" or "False" at the end, indicating whether the GPU device is available, as shown in Figure 0.34. If True, the TensorFlow GPU version is successfully installed; if False, the installation fails. You may need to check the steps of CUDA, cuDNN, and environment variable configuration again, or copy the error and seek help from the search engine.



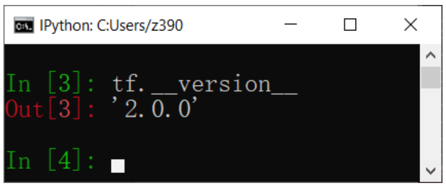
```
C:\Users\z390>ipython
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.6.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import tensorflow as tf
2019-11-07 10:00:21.917929: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudart64_100.dll

In [2]: tf.test.is_gpu_available()
2019-11-07 10:00:37.098827: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
2019-11-07 10:00:37.109257: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library nvcuda.dll
2019-11-07 10:00:37.251093: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties:
name: GeForce GTX 1070 major: 6 minor: 1 memoryClockRate(GHz): 1.759
pciBusID: 0000:01:00.0
2019-11-07 10:00:37.256260: I tensorflow/stream_executor/platform/default/dlopen_checker_stub.cc:25] GPU libraries are s
tatically linked, skip dlopen check.
2019-11-07 10:00:37.263035: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1746] Adding visible gpu devices: 0
2019-11-07 10:00:38.173923: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1159] Device interconnect StreamExecutor
with strength 1 edge matrix:
2019-11-07 10:00:38.178687: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1165]      0
2019-11-07 10:00:38.181997: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1178] 0:   N
2019-11-07 10:00:38.186504: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1304] Created TensorFlow device (/device:
GPU:0 with 6377 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1070, pci bus id: 0000:01:00.0, compute capabil
ity: 6.1)
Out[2]: True

In [3]:
```

Figure 0.34 TensorFlow-GPU installation test

If you don’t have GPU, you can install the CPU version. The CPU version cannot use the GPU to accelerate calculations, and the calculation speed is relatively slow. However, because the models introduced as learning purposes in this book are generally not computational expensive, the CPU version can also be used. It is also possible to add the NVIDIA GPU device after having a better understanding of the deep learning in the future. If the installation of the TensorFlow GPU version fails, we can also use the CPU version directly. The command to install the CPU version is:

```
# Using Tsinghua source to insall TensorFlow CPU version
pip install -U tensorflow -i https://pypi.tuna.tsinghua.edu.cn/simple
```

After installation, enter the "import tensorflow as tf" command in ipython terminal to verify that the CPU version is successfully installed. After the TensorFlow is installed, you can view the version number through "tf .__ version__", as shown in Figure 0.35.


IPython: C:Users/z390

```
In [3]: tf.__version__
Out[3]: '2.0.0'

In [4]:
```

Figure 0.35 TensorFlow version test

The above manual process of installing CUDA and cuDNN, configuring the Path environment variable and installing TensorFlow is the standard installation method. Although the steps are tedious, it is of great help to understand the functional role of each library. In fact, for the novice, you can complete the above steps by two commands as follow:

```
# Create virtual environment tf2 with tensorflow-gpu setup required
# to automatically install CUDA,cuDNN,and TensorFlow GPU
conda create -n tf2 tensorflow-gpu
# Activate tf2 environment

conda activate tf2
```

This quick installation method is called the minimal installation method. This is also the convenience of using Anaconda distribution. TensorFlow installed through the minimal version requires activation of the corresponding virtual environment before use, which needs to be distinguished from the standard version. The standard version is installed in Anaconda's default environment base, and generally does not require manual activation of the base environment.

Common Python libraries can also be installed by default, the command is as follow:

```
# Install common python libraries
pip install -U ipython numpy matplotlib pillow pandas -
i https://pypi.tuna.tsinghua.edu.cn/simple
```

TensorFlow 在运行时，默认会占用所有 GPU 显存资源，这是非常不友好的行为，尤其是当计算机同时有多个用户或者程序在使用 GPU 资源时，占用所有 GPU 显存资源会使得其他程序无法运行。因此，一般推荐设置 TensorFlow 的显存占用方式为增长式占用模式，即根据实际模型大小申请显存资源，代码实现如下：

When TensorFlow is running, it will consume all GPU resources by default, which is very unfriendly, especially when the computer has multiple users or programs using GPU resources at the same time. Occupying all GPU resources will make other programs unable to run. Therefore, it is generally recommended to set the GPU memory usage of TensorFlow to the growth mode, that is, to apply for GPU memory resources based on the actual model size. The code implementation is as follow:

```
# Set GPU resource usage method
# Get GPU device list
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  try:
    # Set GPU usage to growth mode
    for gpu in gpus:
      tf.config.experimental.set_memory_growth(gpu, True)
```

```
except RuntimeError as e:
  # print error
  print(e)
```

1.6.4 Common Editor Installation

There are many ways to write programs in Python. You can use Ipython or Jupyter Notebook to interactively write code. You can also use Sublime Text, PyCharm, and VS Code to develop medium and large projects. This book recommends using PyCharm to write and debug code, and using VS Code for interactive project development. Both of them are free. Users can download and install themselves.

Next, let's start the deep learning journey!

1.7 Reference

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature,* 518, pp. 529-533, 2 2015.