第 1 章

横空出世的 Mamba

Mamba 这一崭新的深度学习架构，宛如一颗璀璨的新星在深度学习的天空中闪耀。它以独特的选择性状态空间模型为基石，旨在突破传统状态空间模型在处理离散且信息密集型数据时的局限。其核心创新在于引入输入依赖的动态机制，让模型犹如灵动的舞者，能够根据当前数据巧妙并选择性地传递或遗忘信息。

基于状态空间模型（State Space Models，SSM），Mamba 创新性地融入选择性机制与硬件感知的并行算法，使其不仅能高效处理序列数据，更在推理速度上远超传统 Transformer，并且实现了序列长度的线性缩放。

在多个领域，Mamba 展现了卓越的性能，它如智慧的精灵，以高效、灵活且硬件友好的特质，在语言模型、音频处理和基因组数据模型等领域翩翩起舞，为序列建模开启了全新的思路与方法。尽管仍需持续探索与优化，但 Mamba 的潜力无限，未来有望在自然语言处理、语音识别和生物信息学等众多领域大放异彩。

1.1 深度学习的前世今生

深度学习宛如一股汹涌澎湃的科技浪潮，在当今时代展现出了无与伦比的力量。它并非一蹴而就，而是历经了漫长的积累与沉淀。

从早期的理论探索到如今的广泛应用，深度学习走过了一条崎岖却充满希望的道路。在这一过程中，无数的科学家和研究人员倾注了智慧与心血，推动着这一领域的不断发展。深度学习就像是一颗被精心培育的种子，在适宜的环境中逐渐生根发芽、茁壮成长，最终绽放出绚烂的花朵。

随着计算能力的提升和数据的爆发式增长，深度学习迎来了它的黄金时代。它在图像识别、语音处理、自然语言理解等诸多领域取得了惊人的突破，为人们的生活带来了翻天覆地的变化。

那些曾经看似遥不可及的梦想，如今正一步一步变为现实，激发了我们对未来无限的憧憬与期待。

1.1.1 深度学习的发展历程

深度学习的发展犹如一部波澜壮阔的史诗，充满了曲折与辉煌。

回溯往昔，深度学习的理念在早期的探索阶段便已开始萌芽。科学家们不断尝试理解大脑的工作机制，试图从中汲取灵感，为构建更强大的学习模型奠定基础。然而，这一时期的技术和认知都存在诸多局限，深度学习的发展之路充满艰辛与挑战。尽管如此，先驱者们依然坚持不懈地前行，在黑暗中摸索着前进的方向。深度学习的发展历程如图 1-1 所示。

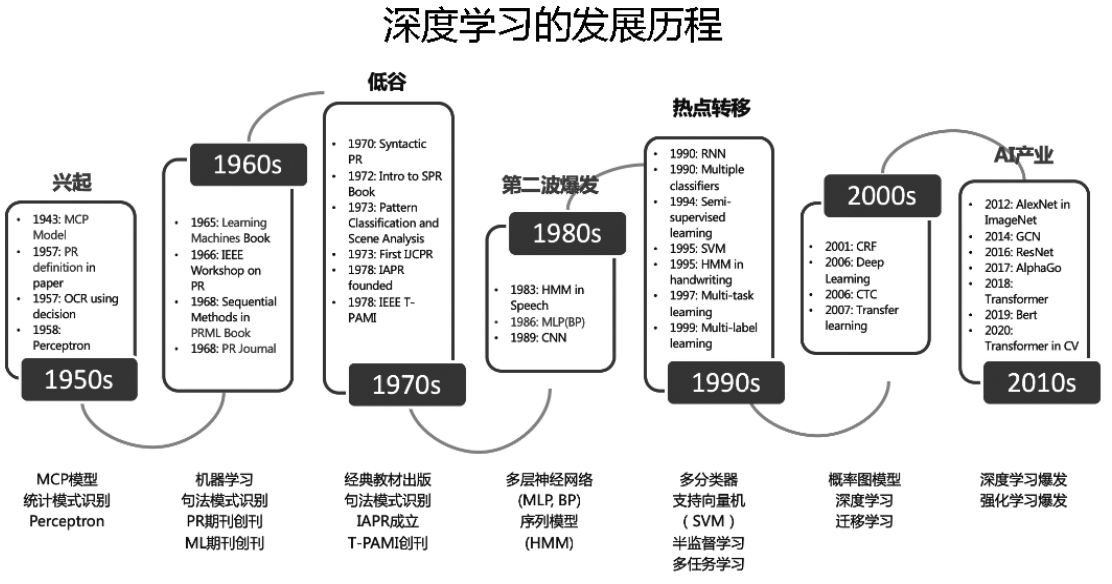

图 1-1 深度学习的发展历程

随着时间的推移，计算机技术的不断进步为深度学习提供了重要支撑。计算能力的提升使得处理大规模数据和复杂模型成为可能，为深度学习的腾飞铺平了道路。在此过程中，一些关键的算法和技术，如反向传播算法等，开始崭露头角，为深度学习的发展引擎点燃了星星之火。

进入新世纪，数据的爆发式增长更是成为深度学习发展的强大助力。海量的数据为模型的训练提供了丰富素材，使得深度学习能够不断优化和提升性能。同时，越来越多的研究人员投身这一领域，他们的智慧和创造力汇聚成强大的推动力，推动着深度学习不断向前突破。

在图像识别领域，深度学习取得了令人瞩目的成就。它能够准确地识别各种物体、场景和人物，准确率甚至超越了人类。这一突破不仅为计算机视觉领域带来了革命性变化，也为智能安防、自动驾驶等众多应用场景打开了大门。同样，在语音处理和自然语言理解方面，深度学习也展现出了强大的实力，使得人机交互变得更加自然和流畅。

尽管深度学习取得了诸多突破，它的发展之路并非一帆风顺，依然面临着诸多质疑和挑战。数据隐私、模型可解释性、伦理道德等问题成为社会关注的焦点。然而，正是这些挑战促使着

研究人员不断反思和改进，推动着深度学习朝着更加健康、可持续的方向发展。

如今，深度学习已经成为科技领域的核心力量之一，影响力渗透到各个行业和生活的方方面面。从医疗健康到金融服务，从教育科研到娱乐传媒，深度学习都在发挥着重要的作用。它不仅改变了我们的生活方式，也为人类社会的进步注入了强大动力。

展望未来，深度学习的发展前景依然广阔。随着技术的不断创新和突破，我们有理由相信它将在更多领域创造奇迹，为人类带来更多的福祉。同时，我们也必须保持清醒的头脑，正确看待深度学习带来的影响，确保其发展始终符合人类的利益和价值观。深度学习的发展历程是一部人类智慧与勇气的赞歌，它将继续在科技的舞台上演绎精彩的篇章。

1.1.2　深度学习与人工智能

深度学习的发展历程对人工智能领域产生了深远的影响。在过去的几年中，深度学习技术作为人工智能的重要组成部分之一，已在各个领域得到广泛应用。神经网络的概念最早出现在 20 世纪 40 年代至 50 年代，主要关注使用神经网络进行模式识别和学习表示。

如今，深度学习在语音识别领域通过提高准确性和速度，发挥着重要作用；在图像识别领域，深度学习技术显著提升了识别效果。此外，深度学习不仅在计算机视觉、自然语言处理等领域取得了显著成果，也推动了人工智能的整体发展，比如推动了人工智能在自然语言处理方面的发展，使计算机能够更好地理解和处理人类语言；促进了计算机视觉的广泛应用，让计算机能够更准确地识别图像等。

同时，深度学习还引领了人工智能在智能机器人、智能制造等方面的融合，为这些领域带来了新的发展机遇。深度学习技术在人工智能领域的应用和影响，使其成为推动人工智能发展的关键力量，未来也将继续发挥重要作用。

具体来看，深度学习的发展对人工智能领域的影响主要体现在以下几个方面：

- 推动技术突破：深度学习使人工智能在图像识别、语音识别、自然语言处理等领域取得了显著进展，提高了模型的准确性和泛化能力。
- 拓展应用领域：深度学习的成功应用推动了人工智能在医疗、金融、交通、教育等领域的发展，为这些领域带来了创新和改进的机会。
- 促进学科融合：深度学习的发展需要跨学科的知识和技术，如数学、统计学、计算机科学等，这促进了不同学科之间的融合和交流。
- 提升计算能力：深度学习模型的训练需要大量计算资源，推动了硬件技术的发展，如 GPU、TPU 等专用芯片的出现，提高了计算效率。
- 引发研究热潮：深度学习的成功吸引了大量研究人员和资源投向人工智能领域，推动了该领域的快速发展。
- 改变行业格局：深度学习技术的应用改变了许多行业的竞争格局，一些传统企业通过应用人工智能技术实现了转型升级，而一些新兴人工智能企业迅速崛起。

- 带来社会影响：人工智能的发展对社会产生了深远的影响，如就业结构的变化、隐私和安全问题等，需要我们认真思考和应对。

总的来说，深度学习的发展是人工智能领域发展的重要阶段，为未来发展奠定了坚实基础。未来，我们可以期待深度学习技术继续推动人工智能的发展，为人类社会带来更多福祉。

1.2 深度学习中的主要模型

深度学习模型是深度学习的核心构成部分，宛如智能的“数据处理器”，通过复杂而精巧的结构和算法实现特征的提取和处理。

这些模型具有强大的学习能力，可以从海量数据中自动发现有价值的特征模式。例如，在图像识别任务中，深度学习模型能够从像素级数据中逐步提取出形状、纹理、颜色等高级特征，从而准确识别图像中的物体或场景。

根据不同任务的具体目标，模型会调整和优化其输出结果。无论是在分类任务中确定所属类别，还是在回归任务中预测具体数值，深度学习模型都能灵活地适应并给出准确的回应。

通过不断训练和优化，深度学习模型能够逐渐提升特征提取能力和对任务处理精度，在各种应用场景中展现卓越性能。无论是计算机视觉、自然语言处理、语音识别还是其他领域，深度学习模型都在推动着技术进步和创新，为解决实际问题提供强大支持。它们的发展和应用正不断拓展我们对智能和数据处理的认知边界。

1.2.1 深度学习中的代表性模型和应用

深度学习作为人工智能领域至关重要的一个分支，其发展历程循序渐进，可划分为多个具有鲜明特征的阶段，而每个阶段均拥有极具代表性的模型以及广泛的应用场景。以下将对深度学习处于不同阶段的代表模型设计与应用展开简单的介绍。

1. 深度学习不同阶段的代表模型

1）早期阶段（2006—2010 年）

- 卷积神经网络（Convolutional Neural Network，CNN）：主要应用于图像识别与处理等领域。CNN 的核心精髓在于巧妙地运用卷积核提取输入图像的局部特征，并借助池化层进行特征抽取与降维（即降低维度）。
- 回归神经网络（Quantile Regression Neural Network，QRNN），本质上属于一种非参数、非线性的网络架构。该网络在应用中展现出强大的能力，能够深入揭示响应变量的完整条件分布，还具备模拟金融系统等领域内复杂非线性特征的能力。

2）发展阶段（2011—2015 年）

- 递归神经网络（Recursive Neural Network，RNN）：有效处理非线性和循环的数据结

构。其核心观念是利用递归连接来清晰地呈现序列中的相关信息。

- 长短时记忆网络（Long Short-Term Memory，LSTM）：通过精妙的门控机制来精准地把控信息的输入、输出和更新。LSTM 的核心思路是借助门的作用来有效地控制序列中信息的流动方向与状态。

3）近期阶段（2016 年一至今）

- 自注意力机制（Attention）：可以促使模型更出色地聚焦于序列中的关键信息。Attention 的核心思想是，利用具有特定意义的注意力权重来明确地表示序列中的关键要点。
- Transformer 模型：将自注意力机制与编码器和解码器巧妙地融合在一起。Transformer 的核心要点在于，利用自注意力机制和跨注意力机制构建更为高效的序列模型。

2. 深度学习模型的应用

以上这些代表性模型在多个领域得到了广泛应用，例如：

- 图像识别领域：CNN 在图像识别任务中表现得极为卓越，比如能够精准地识别物体、场景和人物等。
- 语音识别领域：RNN 和 LSTM 被广泛应用于语音识别系统，能够有效处理音频序列并准确识别语音内容。
- 自然语言处理领域：涵盖文本分类、情感分析、机器翻译等任务。尤其是 Transformer 模型在自然语言处理中更是取得了极为显著的成果。
- 医疗诊断领域：深度学习模型在医学图像分析和疾病预测中发挥作用，辅助医生进行诊断和治疗决策。
- 金融预测领域：能够助力预测股票价格、市场趋势等，为投资决策提供有力的支撑与依据。

总的来讲，深度学习的持续发展为解决各式各样复杂的现实问题提供了强大的工具。随着技术的不断演进，深度学习模型必将在更多领域充分发挥重要作用，推动人工智能向更高层次发展。

1.2.2　CNN、RNN 与 Transformer

深度学习的发展离不开在不同阶段涌现的、极具代表性的架构。其中，CNN（卷积神经网络）、RNN（循环神经网络）以及 Transformer 等架构不仅各具优势和适用场景，而且它们常常是交叉使用的。正是这种交叉使用推动了深度学习技术的不断推陈出新，在各个领域展现出越发强大的威力，如图 1-2 所示。

CNN 擅长处理具有空间结构的数据，在图像识别、计算机视觉等方面表现出色；RNN 则对具有序列特征的数据处理有着天然优势，在自然语言处理、时间序列分析等领域发挥着重要作用；而 Transformer 以其强大的并行处理能力和对长序列数据的良好适应性，在大规模语言模型等领域引领着发展方向。

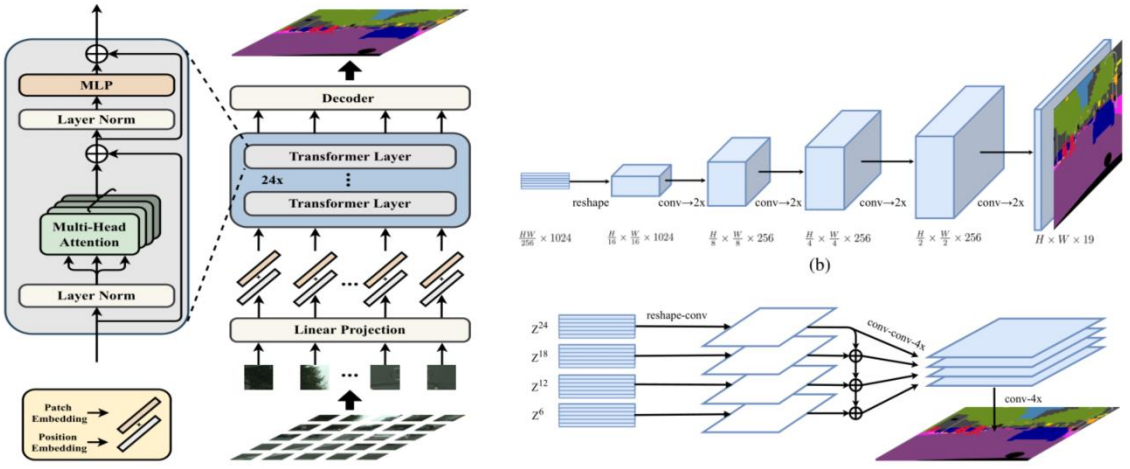

图 1-2　CNN、RNN 与 Transformer 的融合使用

1. 卷积神经网络（CNN）

CNN 宛如一位技艺精湛的画师，以其独特的方式勾勒出数据的精妙轮廓。它的核心算法包括卷积操作和池化操作。卷积操作犹如神奇的画笔，通过不同的卷积核在数据上滑动，捕捉局部的特征模式，细腻地描绘出每个细节。池化操作则像智慧的提炼，对特征进行压缩和简化，提取出关键信息。

在应用方面，CNN 是图像识别领域的明星。它能够精准识别出图像中的物体、场景和各种细节，为我们打开了看清视觉世界的新窗口。无论是在人脸识别中准确辨别身份，还是在自动驾驶中识别路况和障碍物，CNN 都发挥着至关重要的作用。它还在视频分析、医学影像诊断等领域大放异彩，如帮助医生发现微小的病变。

2. 循环神经网络（RNN）

RNN 恰似一位记忆超群的智者，能够记住过去的信息并与当前信息融合。其独特的算法在于循环连接，使得信息能够在时间维度上传递和积累。这种对序列数据的深刻理解就像是在时间的长河中捕捉到连续的音符。

RNN 在自然语言处理领域表现出色，如在机器翻译中，RNN 能够理解源语言句子的结构和语义，生成准确流畅的目标语言。在语音识别中，它能跟随语音的节奏和韵律，准确转化为文字。RNN 还广泛应用于情感分析、文本生成等任务，为人机交流增添了灵动与智慧。

3. Transformer

随着深度学习研究的不断进展，Transformer 横空出世。它摒弃了传统的循环结构，采用了自注意力机制等精妙算法。自注意力机制（Self-Attention）犹如灵动的目光，能够快速而准确地聚焦序列中的关键信息，赋予模型强大的全局信息感知能力。

Transformer 的应用领域极为广泛且成效显著。在自然语言处理中，Transformer 模型已成为主流，如在大规模语言模型中展现出惊人的语言理解和生成能力。它不仅推动了智能聊天机器

人更加智能和自然，也助力文本摘要、知识问答等领域取得巨大进步。同时，Transformer 的影响力逐渐延伸到其他领域，为跨领域的创新提供了强大动力。

Transformer 与其他深度学习模型（如 CNN 和 RNN）相互补充、相互融合，使得深度学习能够更好地应对不同类型、不同复杂度的数据处理需求。例如，在一些复杂的任务中，结合 CNN、RNN 和 Transformer 模型，能够充分发挥它们各自的长处，达到更优的性能和效果。随着研究的深入和技术的持续进步，这种交叉使用的趋势还将继续推动深度学习的发展，开辟出更多的应用领域和探索新的可能性。

1.2.3　剑指王者的 Mamba 带来了新的突破

Mamba是一种新型的深度学习模型，通过选择性状态空间模型（Selective State Space Models，SSMs）对传统的状态空间模型进行了改进。Mamba 模型的下载网站为 https://github.com/state-spaces/mamba，下载页面如图 1-3 所示。

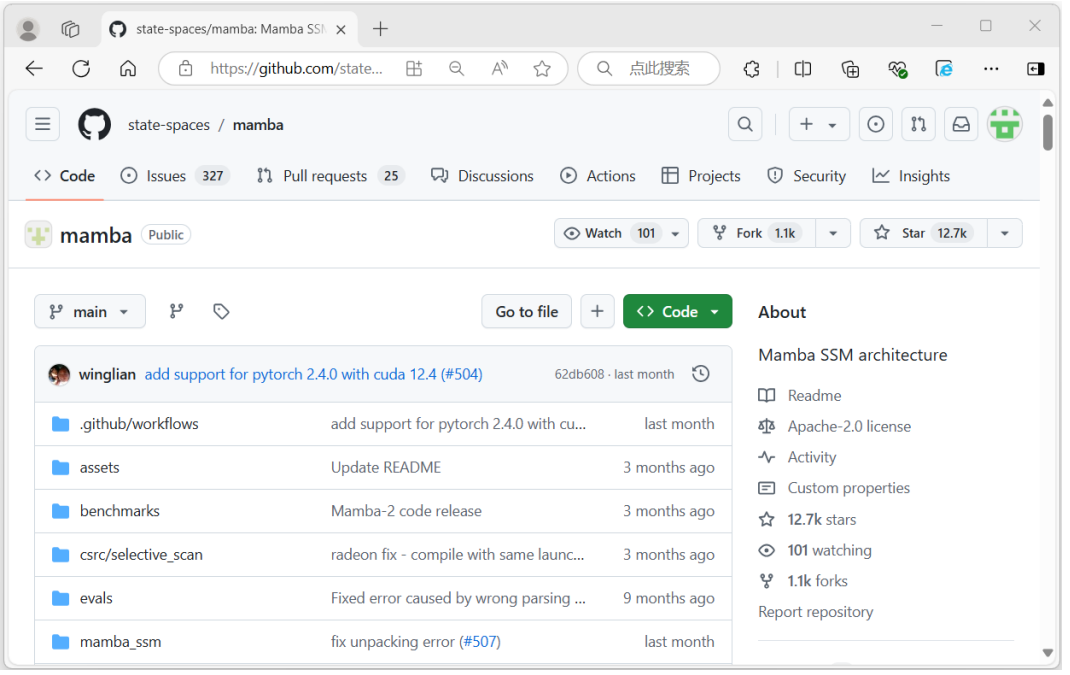

图 1-3　Mamba 下载页面

以下是 Mamba 模型的核心思想和架构的简单介绍。

1. 核心思想

Mamba 的核心思想是利用选择性机制来实现更高效、灵活的序列建模。与传统的 SSMs 不同，Mamba 的 SSMs 参数会根据输入动态调整，从而使模型能够根据当前数据选择性地传递或遗忘信息。这种选择性机制使 Mamba 能够更好地处理离散和信息密集型数据，如文本。

2. 架构

- 固定主干：Mamba 架构的核心是一个固定的主干，用于从一个隐藏状态转换到下一个隐藏状态。该主干由一个矩阵 ***A*** 定义，允许跨序列的预计算，从而提高计算效率。
- 输入相关转换：输入对下一个隐藏状态的影响由矩阵 ***B*** 定义。与传统的 SSMs 不同，Mamba 的矩阵 ***B*** 会根据当前输入进行动态调整，从而使模型能够更好地适应不同的输入数据。
- 选择性机制：作为 Mamba 的关键组成部分，选择性机制通过对 SSM 参数进行输入依赖的调整，使模型能够根据当前数据有选择地传播或遗忘信息。这使得 Mamba 能够更高效地处理长序列数据，并提高模型的性能。
- 硬件感知算法：为了满足选择性机制的计算需求，Mamba 使用了一种硬件感知算法。该算法使用扫描操作而非卷积来循环执行计算，从而实现 GPU 上的高效计算。

综上所述，Mamba 基于选择性状态空间模型的深度学习模型，结合选择性机制和硬件感知算法，实现了更高效、灵活的序列建模。

这些突破使得 Mamba 在处理长序列数据时具有更高的效率和性能，为深度学习模型的进一步发展带来了新的可能性。它在自然语言处理、音频处理、视频内容生成等领域具有广泛的应用前景，并在多个领域取得了很好的性能，是一种非常有前途的深度学习模型。

1.3 本章小结

本章作为全书的开篇，回顾了深度学习的原理，并详细介绍了 Mamba 在深度学习中的应用。在接下来的章节中，我们将引领读者深入探索这一新兴领域，迎接更为严峻的挑战。

随着本书内容的深入，Mamba 模型将以其独特的优势和卓越的性能，展现深度学习领域的无限可能与潜力。读者将会领略到 Mamba 在处理复杂数据、加速训练过程以及优化模型性能方面的卓越能力。让我们携手共进，开启关于深度学习与 Mamba 架构的奇妙旅程吧！

另外，本书是《PyTorch 2.0 深度学习从零开始学》的姊妹篇，部分基础内容不再赘述，有需要的读者可参考《PyTorch 2.0 深度学习从零开始学》进行学习。

第 2 章

挑战注意力机制地位的 Mamba 架构详解

Transformer 及其注意力模型已经在自然语言处理领域树立了基准。然而，它们的效率随着序列的延长而下降。Mamba（曼巴）架构则在这一方面展现了领先优势，它能够更高效地处理长序列，其独特的架构简化了整个过程。

Mamba 的创新点在于它摒弃了传统的注意力模块，并在很大程度上缩减了模型中全连接神经网络（MLP）模块的使用，这一举措有效降低了计算的复杂性和参数数量。

Mamba 的主要特点如下。

- 选择性 SSM: Mamba 利用选择性状态空间模型（Selective State Space Models，SSM），能够过滤无关信息，专注于相关数据，从而增强其在序列处理中的能力。这种选择性对于基于内容的推理至关重要。
- 硬件感知算法：Mamba 采用针对现代硬件，尤其是 GPU 优化的并行算法。与传统模型相比，这种设计显著提高了计算速度，并减少了内存需求。
- 简化架构：通过集成选择性 SSM 并去除注意力机制和 MLP 模块，Mamba 提供了更简洁、更均匀的结构。这不仅带来了更好的可扩展性，还优化了整体性能。

Mamba 在语言处理、音频分析和基因组学等多个领域表现出了卓越的性能，特别是在预训练和特定领域任务中表现出色。例如，在语言建模任务中，Mamba 的性能可与大型 Transformer 模型相媲美甚至超越它们。

可以看到，Mamba 代表了序列建模领域的一次飞跃，为处理信息密集型数据提供了 Transformer 架构的强大替代方案。其设计不仅符合现代硬件的需求，还优化了内存使用和并行处理能力。Mamba 的开源代码库及其预训练模型，使其成为人工智能和深度学习领域研究人员和开发人员易于使用且功能强大的工具。

2.1 Mamba 的优势

Transformer 架构无疑是大型语言模型取得显著成功的基石。从开源模型（如 Mistral）到闭源（如 ChatGPT），Transformer 架构几乎无处不在，其影响力深远。然而，为了突破大型语言模型（LLM）的性能边界，科研人员一直在不懈地探索更为先进的架构。

在这些探索中，一种名为 Mamba 的新型架构引起了业界的广泛关注。Mamba 架构不仅继承了 Transformer 架构的核心优势（如自注意力机制和强大的序列处理能力），而且在多个方面进行了创新和改进。

首先，Mamba 架构通过引入更高效的注意力机制，显著降低了计算复杂度。在 Transformer 中，自注意力机制的计算成本随着序列长度的增加而迅速增长，这限制了模型在处理长文本时的性能。而 Mamba 则采用了更为精细的注意力计算方法，既能更准确地捕捉文本中的关键信息，又能保持较低的计算开销。

其次，Mamba 架构在模型结构设计上进行了优化。它采用了更为灵活的解码器结构，使得模型在处理不同任务时能够更加灵活和高效。

更重要的是，Mamba 架构在保持高性能的同时，还具备出色的可扩展性。这意味着随着模型规模的增大，Mamba 的性能提升将更加显著。这对于构建更大规模、更强大的 LLM 具有重要意义。

总的来说，作为一种新兴的大型语言模型架构，Mamba 架构展现了巨大的潜力和优势。随着研究的不断深入和技术的持续进步，我们有理由相信，Mamba 将成为未来 LLM 发展的重要方向之一。

2.1.1 Transformer 模型存在的不足

在讲解为什么需要使用 Mamba 模型之前，我们首先需要回顾一下前面介绍的注意力模型 Transformer，并深入探讨它的一个显著缺点。

Transformer 在处理自然语言处理（Natural Language Processing，NLP）任务时，将任何文本输入都视为由一系列 Token（词元，俗称标记）组成的序列。这种方法在处理文本数据时确实有其优势，但也存在一个不容忽视的问题。Transformer 处理流程如图 2-1 所示。

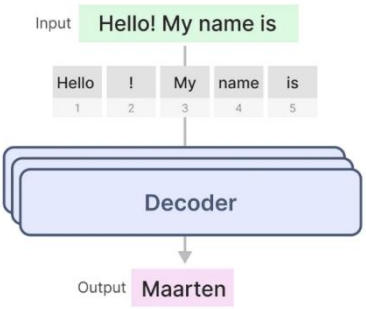

图 2-1 Transformer 的处理流程

Transformer 在处理文本数据时，其核心理念是将任何文本输入视为由一系列 Token（词元）组成的序列。这种处理方式赋予了 Transformer 强大的能力，即无论接收到何种输入，它都能够回顾序列中任一先前的 Token，并据此推导出它们的表示形式。这一特性使得 Transformer 在理解长文本和捕捉文本内部依赖关系方面具有显著优势。Transformers 的编码方法如图 2-2 所示。

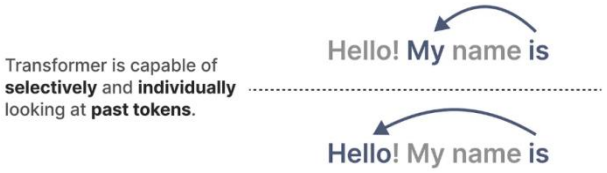

图 2-2　Transformers 的编码方法

Transformer 主要由两部分组成：编码器（Encoder）和解码器（Decoder），如图 2-3 所示。编码器负责将输入的文本序列转换为便于模型处理的内部表示，而解码器则依据这种内部表示生成文本。这两部分结构协同工作，使 Transformer 能够胜任机器翻译、文本摘要等多种 NLP 任务。

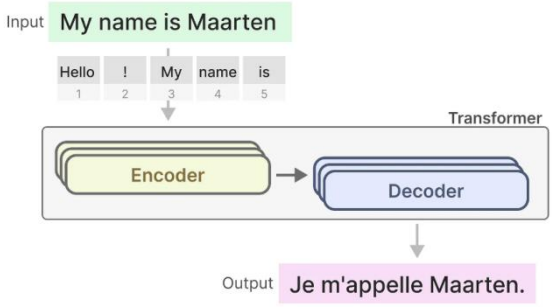

图 2-3　编码器和解码器

另外，在训练过程中，单个解码器块的核心组件包括屏蔽自注意力机制和前馈神经网络。屏蔽自注意力机制使模型在训练时仅关注当前位置之前的 Token，从而避免了未来信息的泄露。该机制通过创建权重矩阵，将每个 Token 与之前的 Token 进行比较（见图 2-4），并根据它们之间的相关程度确定权重，从而帮助模型快速获取整个序列的上下文信息，进而提升文本生成质量。

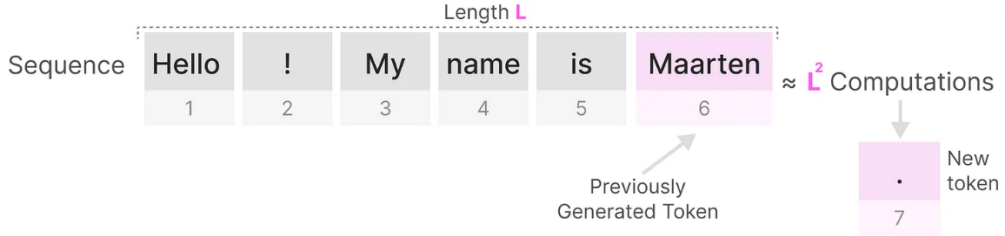

图 2-4　Token 与之前的 Token 进行比较

然而，尽管 Transformer 在训练过程中表现出色，但在推理阶段却面临“推理的诅咒”问题。

具体来说，即使我们已经根据先前的 Token 生成了一些文本，在生成下一个 Token 时仍需重新计算整个序列的注意力。由于这种计算量随着序列长度的增加而呈平方增长，导致在生成长文本时的计算成本极高。全序列的注意力计算如图 2-5 所示。

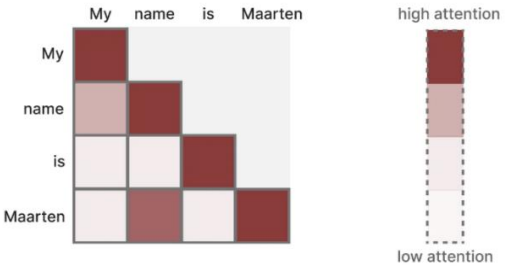

图 2-5　全序列的注意力计算

例如，对于长度为 L 的序列，生成每个 Token 大约需要 L^2 的计算量。随着长度 L 的增大，这一计算成本迅速上升，使 Transformer 在处理长文本时变得力不从心。这不仅成为 Transformer 架构的主要瓶颈，也限制了其在需要生成长文本的应用场景中的使用。图 2-6 展示了使用注意力机制的训练与推理的对比。

图 2-6　使用注意力机制的训练与推理比较

为了解决这一问题，研究人员开始探索新的架构和方法。Mamba 架构正是在这种背景下应运而生的。Mamba 架构通过引入新的技术和机制，旨在降低生成文本时的计算成本，同时保持甚至提升文本生成质量。

2.1.2　循环神经网络

循环神经网络（Recurrent Neural Network，RNN）是一种专门设计用于处理序列数据的神经网络架构。其核心思想是，RNN 在每个时间步（t）内依赖于两个输入：当前时间步 t 的输入和前一个时间步 t–1 的隐藏状态。基于这两个输入，RNN 会生成下一个隐藏状态，并据此预测输出，如图 2-7 所示。

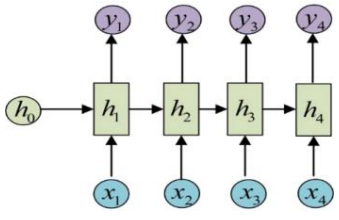

图 2-7　循环神经网络的输出

RNN 的特点在于其内置的循环机制，这种机制使得网络能够将信息从当前时间步传递到下一个时间步。为了更直观地理解这一过程，可以将 RNN 在时间维度上展开，观察其随时间变化的动态行为，如图 2-8 所示。

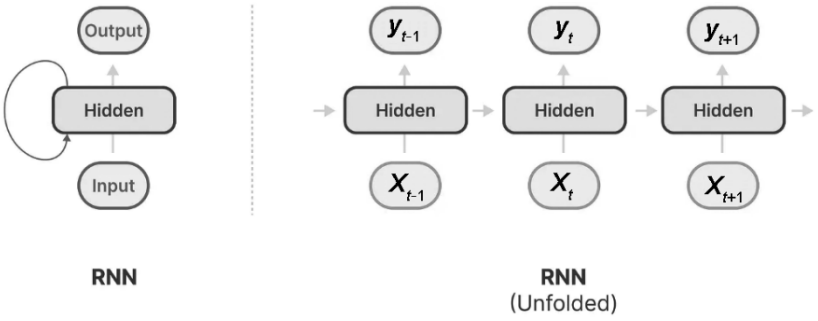

图 2-8　循环神经网络的展开

在生成输出时，RNN 仅依赖当前的输入和上一个时间步的隐藏状态。这种机制使得 RNN 在推理时非常高效，因为它不需要重新计算所有先前的隐藏状态。相比之下，Transformer 这样的架构在推理时需要计算整个序列的隐藏状态，导致计算成本大幅度增加。因此，RNN 的推理速度是随序列长度线性增长的，理论上甚至可以处理无限长的上下文。

为了具体说明这一点，我们可以将 RNN 应用于一个实际的文本输入示例。在这个例子中，随着 RNN 读取文本中的每个单词，它的隐藏状态会不断更新，包含当前及之前所有单词的信息。

然而，由于 RNN 只考虑前一个时间步的隐藏状态，随着时间的推移，一些早期信息可能会被逐渐遗忘。例如，在处理包含 Hello 和 Maarten 这两个单词的文本时，当 RNN 生成 Maarten 这个名称时，它可能不再包含关于 Hello 这个单词的信息，如图 2-9 所示。

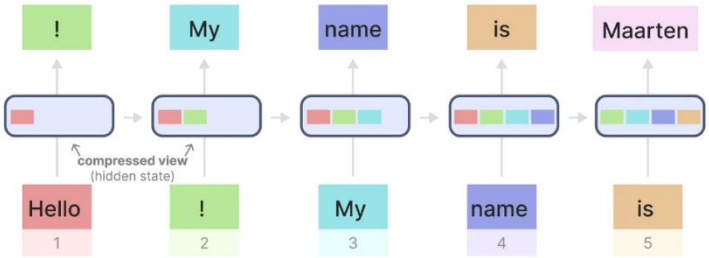

图 2-9　推理过程只受前一个状态影响

此外，RNN 的顺序性质也带来了另一个挑战：训练过程无法并行进行。这是因为 RNN 在每个时间步都需要等待前一个时间步的计算结果，才能继续计算下一个时间步。这种串行计算方式限制了 RNN 的训练速度，尤其在处理长序列时。

因此，尽管 RNN 在推理速度上具有优势，但其无法并行训练的问题却限制了其在实际应用中的性能。那么，是否存在一种既能像 Transformer 那样并行训练，又能保持 RNN 随序列长度线性扩展推理速度的架构呢？答案是肯定的，但这需要我们深入了解一种被称为状态空间模型（State Space Model，SSM）的新概念。训练与推理速度的对比如图 2-10 所示。

图 2-10　训练与推理速度的比较

2.1.3　结合 Transformer 与 RNN 优点的 SSM

为了融合 Transformer 模型的并行训练优势与 RNN 在处理序列时聚焦于前一个 Token 的特点，同时实现对前文信息的选择性整合，Mamba 团队创新性地采用了 SSM（状态空间模型）这一新架构来处理信息的续写。这一架构不仅能够有效结合 Transformer 的高效并行计算能力，还能模拟 RNN 的逐步处理机制，从而在保持全局信息处理能力的同时，也能聚焦于序列中的局部依赖关系。使用状态空间模型的 Mamba 如图 2-11 所示。

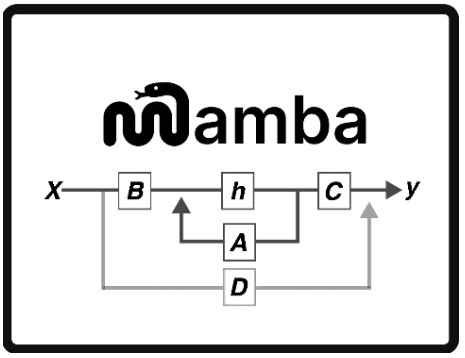

图 2-11　使用状态空间模型的 Mamba

SSM 的核心思想在于其独特的状态空间设计，使得模型能够在处理每个新的 Token 时，动态地更新和维护一个内部状态。这一状态不仅编码了之前所有 Token 的信息，还能根据当前的处理需求进行选择性的信息整合。通过这种方式，SSM 能够在处理长序列时保持高效的计算性能，同时又能捕捉到序列中的重要信息。

在实际应用中，SSM 首先通过 Transformer 的并行训练机制快速处理输入序列，提取全局特征。随后，在推理阶段，它借鉴了 RNN 的工作方式，每次只关注前一个 Token，并根据空间状态进行信息的更新和传递。这种机制使得 SSM 在处理长文本时能够更有效地利用历史信息，从而提高了模型的续写能力和文本生成的连贯性。

可以看到，Mamba 通过引入 SSM，成功地将 Transformer 的高效并行性与 RNN 的序列依赖性结合起来，为自然语言处理和文本生成任务提供了一种全新的解决方案。

2.2　环境搭建 1：安装 Python

2.2.1　Miniconda 的下载与安装

1. 下载和安装

（1）通过百度访问 Miniconda 官方网站，如图 2-12 所示。按页面左侧菜单的提示进入下载页面。

图 2-12　Miniconda 官方网站

（2）下载页面如图 2-13 所示，可以看到官方网站支持不同 Python 版本的 Miniconda。根据自己的操作系统选择相应的 Miniconda 下载即可。这里推荐使用 Windows Python 3.9 版本，相比更高版本，3.9 版本经过一段时间的使用，稳定性较好。当然，读者也可根据自己的喜好选择其他版本。

图 2-13　Miniconda 下载页面

建议读者选择 Python 3.9 版本，以便后续与 PyTorch 2.0.1 GPU 和 CUDA 11.8 版本配合使用。如果想使用其他更高版本的配合方式，可参考 PyTorch 官方网站的文档。

（3）下载完成后，得到的文件是 EXE 版本，直接运行即可进入安装过程，安装时可以选择默认的安装目录。安装完成后，应该能看到如图 2-14 所示的目录结构，说明安装正确。

图 2-14　Miniconda 安装目录

2. 打开控制台

依次单击“开始”→“所有程序”→Miniconda3→Miniconda Prompt，打开 Miniconda Prompt 窗口，它与 CMD 控制台类似，可以在其中输入命令来控制和配置 Python。在 Miniconda 中，最常用的命令是 conda，该命令可以执行一些基本操作，读者可以自行测试它的用法。

3. 验证 Python

接下来验证一下是否成功安装了 Python。在控制台中输入 python，如果安装正确，应该会打印出版本号及控制符号。然后，在控制符号下输入以下代码：

```
print("hello Python")
```

输出结果如图 2-15 所示。

```
(base) C:\Users\xiaohua>python
Python 3.9.10 | packaged by conda-forge | (main, Feb  1 2022, 21:22:07) [MSC v.1929 64 bit
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello")
hello
>>>
```

图 2-15　验证 Miniconda Python 安装成功

4. 使用 pip 命令

使用 Miniconda 工具包的好处在于，它能帮助我们安装和管理大量第三方类库，并维护它们之间的依赖关系。查看已安装的第三方类库的命令如下：

```
pip list
```

读者可以在 Miniconda Prompt 控制台输入 pip list，查看命令执行结果。

Miniconda 中使用 pip 操作的方法有很多，其中最重要的是安装第三方类库，命令如下：

```
pip install name
```

这里的 name 是需要安装的第三方类库的名称。假设需要安装 NumPy 包（这个包已安装过），

输入的命令为：

```
pip install numpy
```

结果如图 2-16 所示。

```
PS C:\Users\xiayu> pip install numpy
Collecting numpy
  Obtaining dependency information for numpy from https://files.pythonhoste
d.org/packages/df/18/181fb40f03090c6fbd061bb8b1f4c32453f7c602b0dc7c08b307ba
ca7cd7/numpy-1.25.2-cp39-cp39-win_amd64.whl.metadata
  Using cached numpy-1.25.2-cp39-cp39-win_amd64.whl.metadata (5.7 kB)
Using cached numpy-1.25.2-cp39-cp39-win_amd64.whl (15.6 MB)
Installing collected packages: numpy
Successfully installed numpy-1.25.2
```

图 2-16　安装第三方类库

使用 Miniconda 工具包的一个好处是，它默认安装了大部分学习所需的第三方类库，这样可以避免在安装和使用某些特定类库时，出现依赖类库缺失的情况。

2.2.2　PyCharm 的下载与安装

和其他编程语言类似，Python 程序的编写可以使用 Windows 自带的控制台。但这种方式对于较为复杂的程序工程来说，容易混淆文件层级和相互之间的关联性。因此，建议在编写程序工程时使用专用的 Python 编译器 PyCharm。

1. PyCharm 的下载和安装

（1）进入 PyCharm 官方网站主页，单击 Download 图标，进入下载页面。在该页面可以选择下载收费的专业版 PyCharm 或免费的社区版 PyCharm。这里我们选择免费的社区版 PyCharm，如图 2-17 所示。

图 2-17　选择免费的社区版 PyCharm

（2）下载下来的安装文件其名为 pycharm-community-2023.3.exe。双击该安装文件以启动它，随后进入安装界面，如图 2-18 所示。单击 Next 按钮，进入下一个安装界面。

（3）在安装 PyCharm 的过程中需要确定相关的配置选项（见图 2-19）。建议勾选窗口中的所有复选框，然后单击 Next 按钮，继续安装。

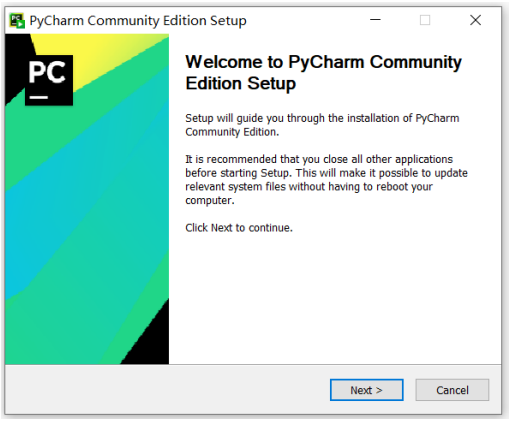

图 2-18　安装界面

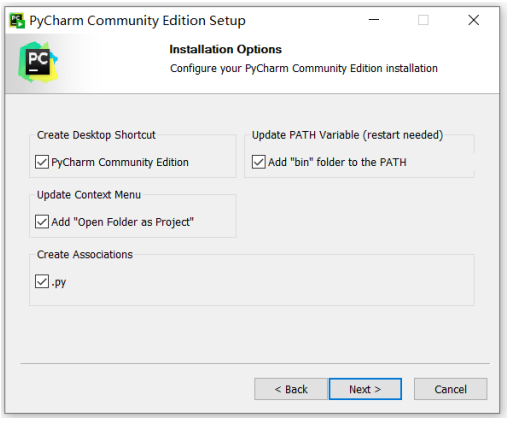

图 2-19　勾选所有的复选框

（4）安装过程比较简单，按照提示逐步单击 Next 按钮即可完成安装。安装完成后，单击 Finish 按钮退出安装向导，如图 2-20 所示。

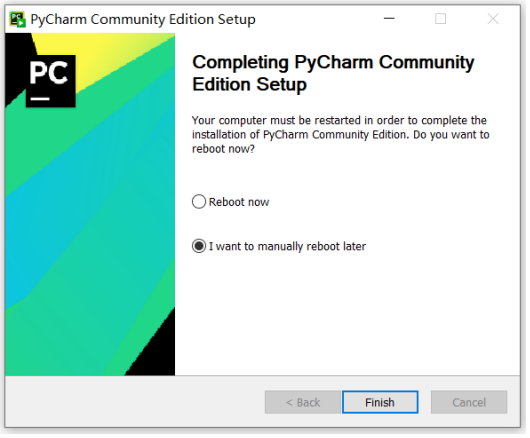

图 2-20　安装完成

2. 使用 PyCharm 创建程序

（1）双击桌面上新生成的图标进入 PyCharm 程序界面，进行第一次启动时的配置，如图 2-21 所示。在此步骤中，需要选择程序存储的定位，一般建议选择第 2 个选项：Do not import settings。

（2）单击 OK 按钮后，进入 PyCharm 配置窗口，如图 2-22 所示。

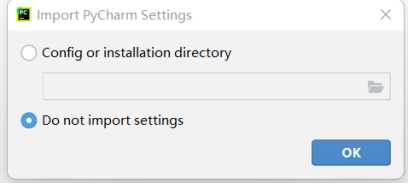

图 2-21　由 PyCharm 自动指定

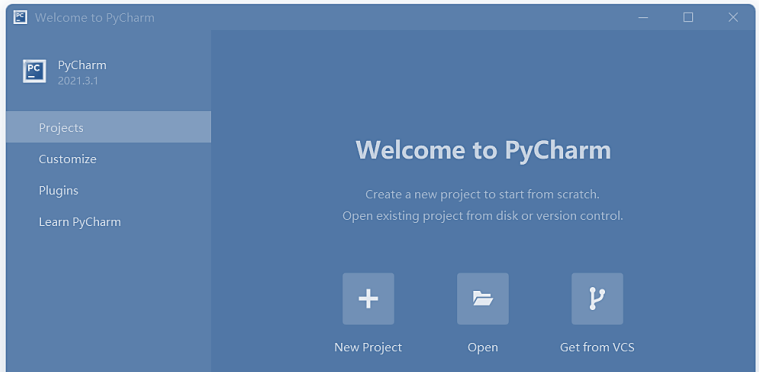

图 2-22　界面配置

（3）在配置窗口中，可以选择自己喜欢的使用风格。如果不熟悉配置，建议使用默认配置，如图 2-23 所示。

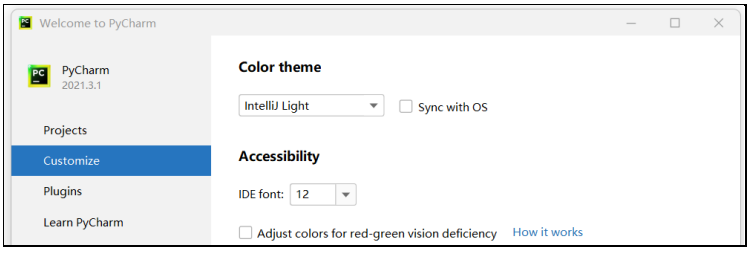

图 2-23　对 PyCharm 的界面进行配置

（4）在如图 2-22 所示的界面上，单击 New Project 按钮创建一个新项目 jupyter_book，如图 2-24 所示。

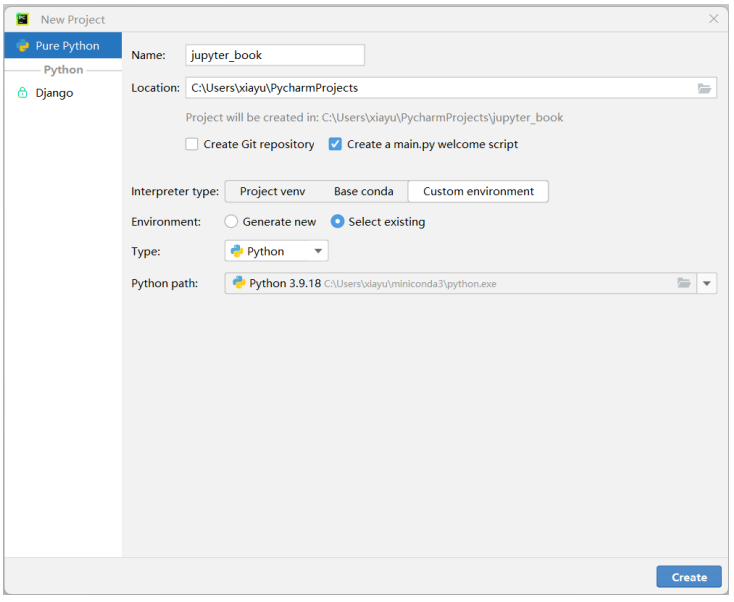

图 2-24　创建一个新项目

单击 Create 按钮，新建一个 PyCharm 项目，如图 2-25 所示。之后，右击新建的项目名 jupyter_book，选择 New 菜单项，可以在本项目目录（jupyter_book）下创建目录、Python 文件等。例如，依次选择 New→Python File 菜单项，新建一个 helloworld.py 文件，并输入一条简单的打印代码，内容如图 2-26 所示。

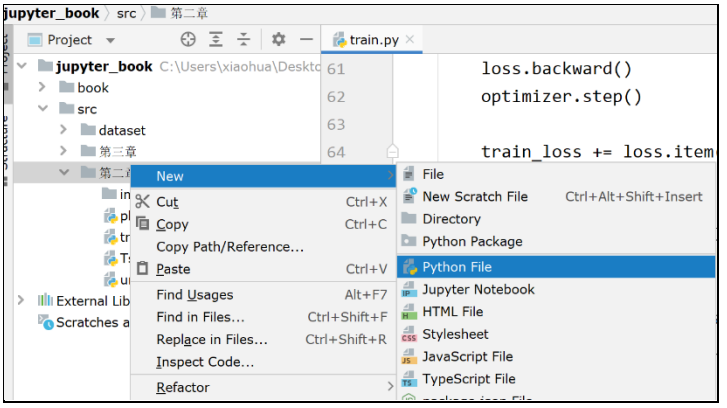

图 2-25　新建一个 PyCharm 项目

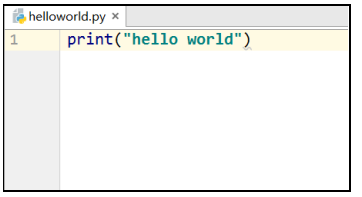

图 2-26　helloworld.py 文件内容

输入代码后，依次单击菜单栏的 Run→run...菜单选项以运行代码，或者直接右击 helloworld.py 文件名，在弹出的快捷菜单中选择 run。如果成功输出 hello world，那么恭喜你，Python 与 PyCharm 的配置就顺利完成了。读者可以尝试把本书配套的示例代码作为项目加入 PyCharm 中进行调试和运行。

2.3　环境搭建 2：安装 PyTorch 2.0

在完成 Python 运行环境调试后，接下来的重点是安装本书的主角——PyTorch 2.0。如果没有 GPU 显卡，完全可以从 CPU 版本的 PyTorch 开始深度学习之旅，但这并不是推荐的方式。与 GPU 版本的 PyTorch 相比，CPU 版本的运行速度存在较大差距，很可能会影响深度学习进程。

PyTorch 2.0 CPU 版本的安装命令如下：

```
    pip install numpy --pre torch==2.0.1 torchvision torchaudio --force-reinstall
--extra-index-url https://download.pytorch.org/whl/nightly/cpu
```

2.3.1　Nvidia 10/20/30/40 系列显卡选择的 GPU 版本

由于 40 系列显卡的推出，市场上存在 Nvidia 10/20/30/40 系列显卡并存的情况。对于需要调用专用编译器的 PyTorch，不同显卡需要安装不同的依赖计算包。表 2-1 总结了不同显卡的 PyTorch 版本以及 CUDA 和 cuDNN 的对应关系。

表 2-1　Nvidia 10/20/30/40 系列显卡的版本对比

显卡型号	PyTorch GPU 版本	CUDA 版本	cuDNN 版本
10 系列及以前	PyTorch 2.0 以前的版本	11.1	7.65
20/30/40 系列	PyTorch 2.0 向下兼容	11.6+	8.1+

这里主要是显卡运算库 CUDA 与 cuDNN 版本的搭配。在 10 系列版本的显卡上，建议优先使用 PyTorch 2.0 之前的版本。在 20/30/40 系列显卡上使用 PyTorch 时，可以参考官方网站（https://developer.nvidia.com/rdp/cudnn-archive），根据本机安装的 CUDA 版本选择相应的 cuDNN 版本。以下是一些具体的例子：

- cuDNN v8.9.6 (November 1st, 2023), for CUDA 12.x
- cuDNN v8.9.6 (November 1st, 2023), for CUDA 11.x
- cuDNN v8.9.5 (October 27th, 2023), for CUDA 12.x
- cuDNN v8.9.5 (October 27th, 2023), for CUDA 11.x

下面以 PyTorch 2.0 为例，演示 CUDA 和 cuDNN 的安装步骤。不同版本的安装过程类似，但需要特别注意软件版本之间的搭配。

2.3.2　PyTorch 2.0 GPU Nvidia 运行库的安装

本节讲解 PyTorch 2.0 GPU 版本前置软件的安装。由于 GPU 版本的 PyTorch 需要使用 NVIDA 显卡，因此额外需要安装 NVIDA 提供的运行库。

我们以 PyTorch 2.0 为例。安装 PyTorch 2.0 最好的方法是根据官方网站提供的安装命令进行安装，具体参考官方网站文档（https://pytorch.org/get-started/previous-versions/）。根据页面内容，官网为 Windows 版本的 PyTorch 2.0 提供了几种安装模式，分别对应 CUDA 11.7、CUDA 11.8 和 CPU only。以下是通过 conda 安装的命令：

```
    # CUDA 11.7
    conda install pytorch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2
pytorch-cuda=11.7 -c pytorch -c nvidia
    # CUDA 11.8
    conda install pytorch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2
pytorch-cuda=11.8 -c pytorch -c nvidia
    # CPU Only
    conda install pytorch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 cpuonly -c
pytorch
```

下面以 CUDA 11.8+cuDNN 8.9 为例讲解安装步骤：

（1）首先是 CUDA 的安装。在百度搜索“CUDA 11.8 download”，进入官方网站下载页面，选择合适的操作系统安装方式（推荐使用 exe(local)本地化安装方式），如图 2-27 所示。

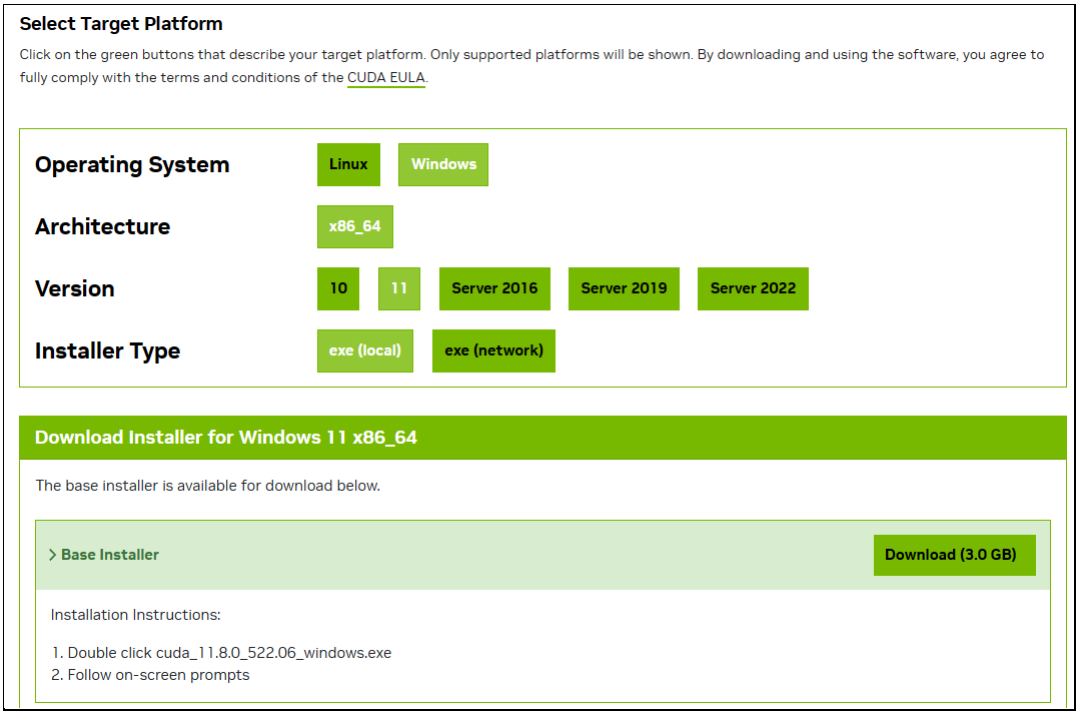

图 2-27　CUDA 11.8 下载页面

（2）下载得到的是一个 EXE 文件，按照默认路径安装即可，无须修改安装路径。

（3）下载并安装对应的 cuDNN 文件。要下载 cuDNN，需要先注册一个用户，然后进入下载页面，如图 2-28 所示。请注意选择与所安装 CUDA 版本匹配的 cuDNN 版本。如果使用的是 Windows 64 位的操作系统，需要下载 x86_64 版本的 cuDNN。

图 2-28　cuDNN 8.9.4 下载页面

（4）下载的 cuDNN 是一个压缩文件，将其解压并把其所有目录复制到 CUDA 安装主目录下（直接覆盖原来的目录）。CUDA 安装主目录如图 2-29 所示。

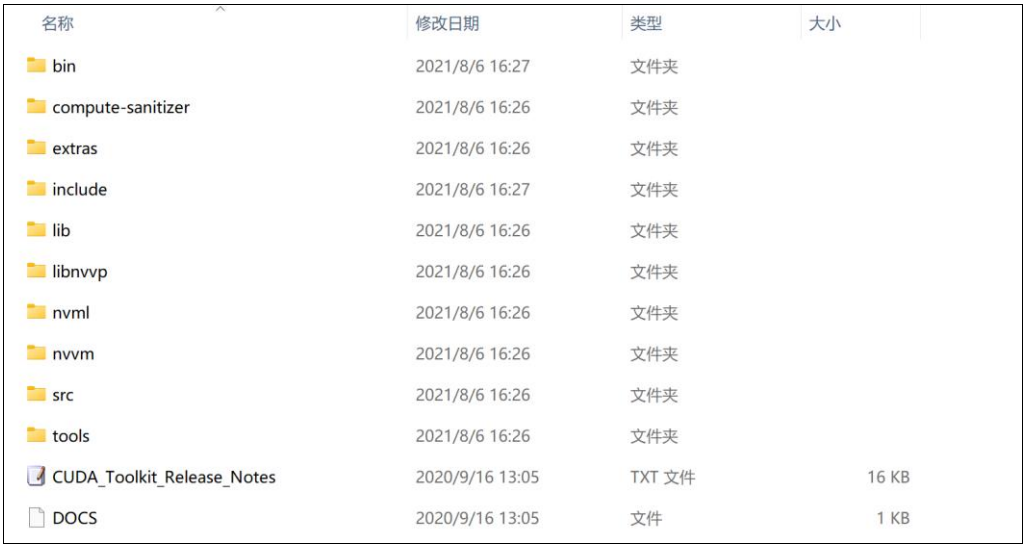

图 2-29　CUDA 安装主目录

（5）确认 PATH 环境变量。安装 CUDA 时，安装向导会自动将 CUDA 运行路径加入 PATH 环境变量中，确保 CUDA 路径加载到这个环境变量即可，如图 2-30 所示。

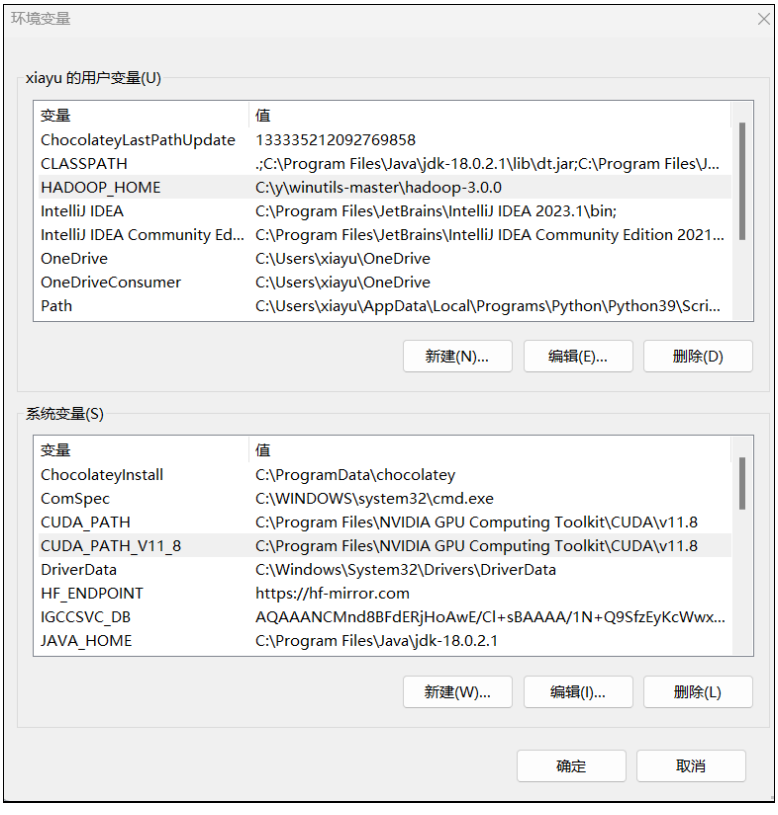

图 2-30　将 CUDA 路径加载到环境变量的 PATH 路径中

（6）最后，在终端窗口执行本节开始给出的 PyTorch 安装命令，完成 PyTorch 2.0 GPU 版本的安装。

2.4 第一次使用 Mamba

前面我们深入探讨了Mamba的使用优点与好处，充分展示了它在自然语言处理领域的独特魅力。现在，我们将进一步开展Mamba的基本学习，带领读者逐步掌握这款强大的工具。

首先，我们将手把手教你如何完成第一次使用Mamba预训练模型的基本操作，以文本生成任务为起点，开启你的Mamba之旅。在这个过程中，你将亲身体验到Mamba如何快速、准确地生成流畅自然的文本内容，感受其强大的语言生成能力。

然后，我们将详细介绍Mamba的三大核心模块。这三大模块是Mamba功能的基石，了解它们将为你更深入地使用Mamba打下坚实的基础。我们会逐一剖析每个模块的工作原理、功能特点以及应用场景，让你对Mamba有一个全面而系统的认识。

通过学习这三大模块，你将能够更好地理解Mamba的运行机制，从而更加熟练地运用它来完成各种复杂的自然语言处理任务。无论是文本生成、文本分类，还是情感分析、问答系统等，Mamba都能为你提供强大的支持。

让我们共同踏上Mamba的学习之旅，探索自然语言处理的无限可能。

2.4.1 Hello Mamba：使用预训练 Mamba 模型生成实战

Hello Mamba！为了让读者能够顺利与Mamba接触，作者准备了一段实战代码供读者学习，读者首先需要打开Miniconda Prompt控制台界面，安装所需的类库modelscope和Mamba库。具体的安装代码如下：

```
    conda install modelscope
    conda install Mamba
```

之后，在 PyCharm 中新建一个可执行的 Python 程序，直接输入作者提供的代码。注意，在此代码中，这里 snapshot_download 函数可以直接下载对应的存档，而 AutoTokenizer.from_pretrained 则需要使用本书提供的文件。

```
    import model
    from model import Mamba

    from modelscope import snapshot_download,AutoTokenizer
    model_dir = snapshot_download('AI-ModelScope/mamba-130m',
cache_dir="./mamba/")
    mamba_model = Mamba.from_pretrained("./mamba/AI-ModelScope/mamba-130m")
    tokenizer = AutoTokenizer.from_pretrained('./mamba/tokenizer')
    print(model.generate(mamba_model, tokenizer, 'Mamba is the'))
```

这里需要提示一下，在上述代码段中，Mamba is the 是起始内容，然后会根据所输入的起始内容生成后续文本，当然读者也可以自行定义起始句子，如下所示：

```
    Mamba is the player on the side as he provides a lot of offense for the Dukes
and also does some awesome blocking that the Mamba must work hard to avoid. This
unit is a must for any offensive player to have when playing on the d1 side
```

另外，需要做一个预告，目前使用训练好的 Mamba 生成中文内容尚不可行。在后续章节中，我们将详细讲解 Mamba 在文本生成任务中的应用，并提供解决方案。读者可以按照章节顺序循序渐进地学习相关内容。

2.4.2　了解 Mamba：构建 Mamba 的三大模块说明

接下来，我们将深入了解 Mamba 模型。通过文本生成任务，我们可以看到 Mamba 展现出了令人瞩目的性能，这主要得益于其内部三种核心模块的精妙融合。每个模块都设计独到，通过协同作用，赋予了 Mamba 卓越的学习和泛化能力。这种精心的组合不仅提高了模型处理复杂任务的准确性，还增强了其稳定性和效率，使 Mamba 能在多个领域中发挥出色的表现，如图 2-31 所示。

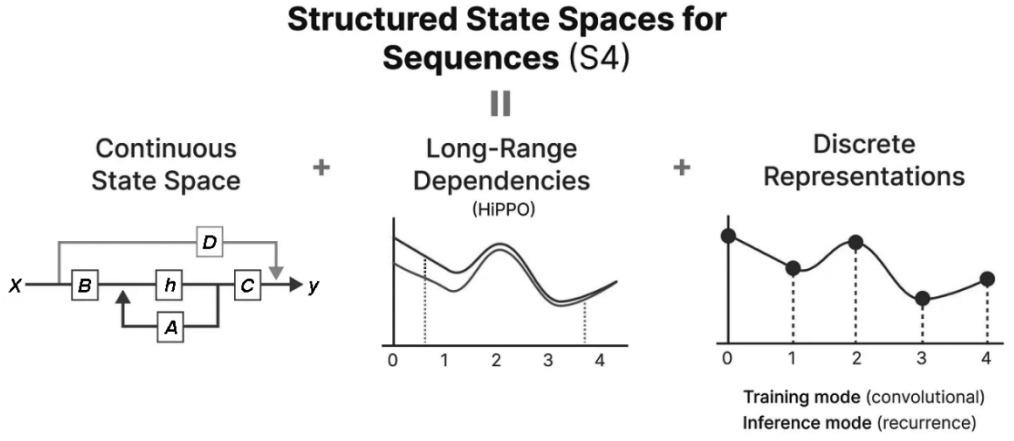

图 2-31　Mamba 的组成构建

- 状态空间模型（SSM）：它为 Mamba 提供了强大的动态系统建模能力，使得网络能够更好地理解和模拟复杂数据的内在状态变化。状态空间模型通过捕捉系统的动态行为，为 Mamba 带来了对序列数据的深入洞察能力。
- 离散化技术：为了创建循环的离散化方法，Mamba 采用了先进的离散化技术。这项技术不仅提高了模型的计算效率，还使得 Mamba 能够灵活地使用循环进行计算，从而适应不同类型的数据和任务需求。通过这种离散化方法，Mamba 在处理序列数据时展现出了更高的灵活性和准确性。
- HiPPO 算法：Mamba 引入了 HiPPO（High-order Polynomial Projection Operator，高阶

多项式投影算子）算法对状态转移矩阵进行初始化，这一创新技术显著增强了模型处理远程依赖关系（long-range dependencies）的能力。在处理长序列数据时，远程依赖关系的捕捉至关重要，而 HiPPO 通过高效压缩历史信息为系数向量，使得 Mamba 能够轻松应对这一挑战。

因此，在了解 Mamba 架构的基本模块之后，Mamba 的基本内容可以总结如下。

1. 基本思想

Mamba 代表“结构化状态空间序列建模”，是一种新型的神经网络架构，专为处理非常长的序列而设计，如视觉、语言和音频数据。

Mamba 的核心思想是通过将长序列数据压缩成更紧凑的表示形式，从而更有效地捕获长距离依赖关系。

2. 工作原理

在传统的序列模型中，如 RNN 或 LSTM，信息的传递是逐步的，这可能导致在处理非常长的序列时出现信息丢失或梯度消失的问题。

Mamba 通过使用高阶多项式来近似输入信号，从而能够在有限的内存预算内压缩长序列的信息。这种方法允许模型在接收到更多信号时，仍能在有限的内存中对整个信号进行压缩和处理。

3. 性能表现

Mamba 架构在处理包含数万个元素的长序列时表现出了令人印象深刻的能力。例如，在某些基准测试中，S4 能够准确地推理出包含 16 000 多个元素的长序列，显示出其强大的长距离依赖捕获能力。

4. 应用潜力

Mamba 架构由于其出色的长序列处理能力，在需要处理长距离依赖关系的任务中具有巨大的应用潜力。这包括语音识别、自然语言处理、时间序列预测等领域，其中对长序列数据的理解和建模至关重要。

可以看到，Mamba 通过状态空间模型、HiPPO 以及先进的离散化技术的有机结合，构建了一个强大而灵活的神经网络模型，为处理复杂序列数据提供了新的解决方案。后文将分别对其进行讲解。

2.5 本章小结

本章对 Mamba 的显著优势进行了详尽的阐述，并与 Transformer 和 RNN 的短板进行了比较。通过这一对比分析，我们清晰地揭示了 Mamba 底层 SSM（状态空间模型）算法的独特性

和创新性。SSM 算法将 Transformer 与 RNN 的优势融为一体，使其在序列数据处理方面展现出了令人瞩目的性能。

具体来说，Mamba 借助 SSM 算法，成功解决了 Transformer 在处理长序列时可能遭遇的注意力分散难题，同时也缓解了 RNN 在处理长距离依赖关系时容易出现的梯度消失或梯度爆炸问题。这一创新融合策略，使得 Mamba 在处理冗长且复杂的序列数据时，能够持续展现出高效稳定的性能。

为了更直观地展现 Mamba 的实用性，我们利用预先训练的 Mamba 模型，顺利完成了文本生成任务。通过此次实践，Mamba 在生成流畅、连贯文本方面的卓越能力得以充分展现。这不仅有力证明了 Mamba 算法的高效性，同时也彰显了它在自然语言处理领域的无限潜力。