



 清言·AI

# 高级机器学习之大语言模型

by 唐杰、杜晋华等

清华大学 (Tsinghua University)

谨以此书献给所有热爱机器学习的人。

# 序

近年来，随着自然语言处理任务复杂度的不断提高，传统的机器学习算法已难以满足实际需求。在数据、算力和人工智能快速发展的背景下，深度学习已成为推动技术进步的核心力量。深度学习不仅在图像和语音处理等传统领域取得了重大突破，还在大语言模型领域展现出巨大的应用潜力。本书将系统介绍深度学习及其在大语言模型中的应用，帮助读者全面理解和掌握这项前沿技术。

通过学习深度学习和大语言模型的理论与应用，读者将能够有效应对不同复杂环境和任务，发展具有人脑模拟功能的智能模型。这不仅是数据挖掘和信息处理的核心技术，还已成为计算机学科研究生必备的技能。本书通过理论讲解和实践操作，使读者不仅能够熟练掌握深度学习和大语言模型的先进理论知识，还能够将其应用于实际项目和新一代智能信息处理系统中。

本书由清华大学唐杰教授和杜晋华等博士生团队共同编写，旨在为读者提供深入了解深度学习和大语言模型的机会。本书内容全面，涵盖了深度学习和大语言模型的基本理论、算法实现和应用案例，适合作为高等院校计算机相关专业的教材，也可作为从事相关领域的科研人员和工程技术人员的参考书。

本书的编写得到了清华大学计算机系高级机器学习课程组的大力支持，在此表示感谢。同时，感谢所有为本书的编写提供帮助的专家和学者，他们的贡献使得本书得以顺利完成。

具体第一章的作者包括：唐杰、杜晋华、顾晓涛、隋元培、张远达、黄茜瑛。第二章的作者包括：唐杰、杜晋华、杨超群、吴彦辰。

杜晋华  
2025年7月30日

# 目录

# 第一章 整体认知

**【摘要】**本章回顾了人工智能的发展历史，并着重介绍了 GLM 系列模型。这些模型涵盖了语义理解、视频语义理解、图像生成、文本生成视频、图像生成视频、代码生成、智能代理（Agent）和多工具调用等多个领域。此外，讨论了大模型的思考与性能涌现，以及人工通用智能（AGI）的分级和未来的发展方向。

## 1.1 引言

自然语言处理（NLP）的发展历程中，算力从 CPU 到 GPU 的提升起到了关键作用，推动了模型从人工神经网络（ANN）向卷积神经网络（CNN）、循环神经网络（RNN）的演变，具体如下：

- **算力提升历程：**早期 NLP 主要基于 CPU 计算，处理能力有限，难以应对大规模数据和复杂模型的训练需求。2000 年代初，GPU 开始被用于深度学习算法的加速。GPU 具有强大的并行计算能力，能够同时处理大量数据，大大提高了深度学习模型的训练速度，使得训练更深层次、更复杂的神经网络成为可能。随后，NVIDIA、Google、Intel 等公司开发了专门为深度学习设计的芯片，如 NVIDIA 的 Tensor Core、Google 的 Tensor Processing Unit (TPU) 等，进一步为 NLP 领域提供了高效的计算支持，推动了 NLP 技术的快速发展。
- **ANN 阶段：**ANN 是 NLP 早期应用的基础模型，它模拟人脑神经元之间的信息传递方式，由输入层、隐藏层和输出层组成，通过权重和激活函数来处理数据。早期的感知机就是一种简单的 ANN，由 Frank Rosenblatt 在 1958 年提出，它开启了使用神经网络进行数据处理的时代，但只能处理线性可分问题。后来出现的多层感知机（MLP）在感知机基础上增加了隐藏层，并通过反向传播算法来训练，能够处理更复杂的非线性问题。不过，ANN 在处理 NLP 任务时存在一些局限性，例如难以处理高维数据和序列数据，计算量较大且训练效率较低。
- **CNN 阶段：**CNN 是一类包含卷积计算且具有深度结构的前馈神经网络，具有表征学习能力。1989 年，Yann LeCun 提出了最早的卷积神经网络 LeNet，用于图像分类，其包含卷积层和全连接层，规模远超此前的相关网络。1998 年，LeCun 及其合作者构建了更加完备的卷积神经网络 LeNet-5，在手写数字识别中取得成功，定义了现代 CNN 的基本结构。2012 年，AlexNet 在 ImageNet 图像分类比赛中取得革命性成果，通过引入 ReLU 激活函数、Dropout 正则化和 GPU 加速训练，显著提升了图像分类效果，标志着深度 CNN 成为计算机视觉领

域的主流架构。此后，CNN 也逐渐被应用于 NLP 领域，例如 2013 年 Kim 等人将 CNN 应用于文本分类任务，取得了较高的准确率。CNN 通过卷积层和池化层可以自动提取文本的局部特征，减少了模型的参数量，提高了训练效率和泛化能力。

- **RNN 阶段：**在处理文本等序列数据时，CNN 难以捕捉数据的时间依赖性，而 RNN 引入了时间递归的结构，使得模型可以根据前一个时间步的状态输出当前步的预测，能够更好地处理序列信息，常用于机器翻译、文本生成等任务。但 RNN 容易在长序列中出现梯度消失问题，导致难以捕捉长距离依赖关系。为克服这一问题，LSTM（长短期记忆网络）应运而生，它通过引入记忆单元和门控机制，能够选择性地记住或遗忘信息，有效解决了长距离依赖问题，被广泛应用于 NLP 的各个领域。

在 NLP 向大模型演进的过程中，数据规模（data scale）的扩张与利用方式的革新是核心驱动力，从依赖大算力支撑到以大规模数据训练模型，经历了多个关键阶段，具体如下：

### 1.1.1 从 Transformer 到 BERT：数据规模与预训练范式的突破

2017 年，Google 团队提出 **Transformer 模型**，其基于自注意力机制（Self-Attention），摆脱了 RNN 的序列依赖限制，能够并行处理文本序列，为大规模数据训练提供了模型结构基础。Transformer 的出现，让模型能够高效利用海量文本数据，突破了此前 RNN 在长序列处理和并行计算上的瓶颈。

2018 年，Google 基于 Transformer 推出 **BERT (Bidirectional Encoder Representations from Transformers)**，首次将“预训练 + 微调”范式推向成熟。BERT 的预训练数据规模显著扩大，使用了 BooksCorpus（约 8 亿词）和英文维基百科（约 25 亿词），总计超过 30 亿词的文本数据。

- 其核心创新是“双向预训练”，通过“掩码语言模型（MLM）”和“下一句预测（NSP）”任务，让模型从上下文双向学习文本语义，能够更全面地捕捉语言规律。
- BERT 的成功证明：当模型结构支持并行计算（依赖算力）且数据规模足够大时，预训练模型能在多种 NLP 任务（如文本分类、问答、命名实体识别）上超越传统方法，成为 NLP 领域的新基准。

### 1.1.2 从 BERT 到 GPT-1 2：数据利用率的跨越式提升

与 BERT 同期，OpenAI 团队聚焦“自回归语言模型”，推出了 **GPT (Generative Pre-trained Transformer)** 系列，其数据利用方式与 BERT 形成鲜明对比：

- **GPT-1 (2018 年)：**基于 Transformer 的解码器部分，采用“单向自回归预训练”（即通过前序文本预测下一个词），预训练数据包括 BooksCorpus（约 8 亿词），与 BERT 初期数据规模接近。但 GPT-1 的核心目标是“生成式任务”，通过预训练后微调，在文本生成、摘要等任务上展现潜力。

- **GPT-2 (2019 年)**: 数据规模和模型参数大幅提升——预训练数据扩展到“WebText”(从互联网爬取的约 40GB 文本, 包含小说、博客、新闻等, 总计约 100 亿词), 模型参数从 GPT-1 的 1.17 亿增加到 15 亿。更关键的是, GPT-2 通过“零样本迁移”能力颠覆了数据利用效率:
  - BERT 的预训练任务 (MLM) 需要对输入文本进行“掩码”(随机遮盖部分词), 实际参与训练的有效数据仅约 15% (被掩码的词);
  - 而 GPT 的自回归任务 (预测下一个词) 会利用输入的每一个词作为上下文, **数据利用率接近 100%**。这意味着在相同数据量下, GPT 系列能更充分地学习语言模式, 因此 GPT-2 在无微调的情况下, 就能直接完成翻译、问答等多种任务, 展现出“通用语言能力”的雏形。

### 1.1.3 从 GPT-2 到 ChatGPT: 数据对齐与“人类反馈”的融合

2020 年后, 大模型的数据规模进一步爆炸(如 GPT-3 的预训练数据达 45TB, 包含万亿级词), 但 ChatGPT (2022 年) 的突破不仅依赖“更大数据”, 更在于**数据利用方式的升级**——引入“**人类对齐数据**”让模型能力与人类需求匹配:

- **基础: 自回归预训练的延续**: GPT-3 及后续模型(如 GPT-3.5)仍以大规模互联网文本(书籍、网页、论文等)进行自回归预训练, 通过预测下一个词学习通用语言规律, 数据规模的扩张使其具备了更强的“知识储备”和“推理潜力”。
- **关键: 标注数据驱动的对齐训练**: 单纯的预训练数据(无标注的原始文本)只能让模型“懂语言”, 但难以“懂人类意图”(如遵循指令、避免有害输出)。ChatGPT 通过两步对齐机制解决这一问题:
  - **监督微调 (SFT)**: 收集人类标注的“高质量问答对”(如用户指令与理想回答), 用这些数据微调预训练模型, 让模型学会遵循具体指令;
  - **人类反馈强化学习 (RLHF)**: 让人类对模型的多个输出进行排序(标注“优劣”), 再训练一个“奖励模型”, 最后用强化学习(如 PPO 算法)让模型优化输出, 使其更符合人类价值观(如友好、准确、安全)。

这些“标注数据”虽然规模远小于预训练数据(通常仅数万至数十万条), 但针对性极强, 相当于为模型提供了“**人类行为准则**”, 最终让 ChatGPT 从“通用语言模型”升级为“能对话、可交互”的实用工具。

### 1.1.4 总结: 数据 scale 的核心逻辑

从 NLP 到大模型, 数据的演进不仅是“量的扩张”(从亿级到万亿级), 更关键是“质的优化”:

- 早期依赖“无标注的大规模原始文本”, 通过预训练让模型学习语言规律;
- 后期通过“高质量标注数据”(如问答对、人类反馈), 让模型从“学习语言”转向“理解人类需求”。

而算力（如 GPU 集群、TPU）则是支撑这一过程的基础——没有足够的算力，既无法处理万亿级数据的预训练，也难以完成复杂的对齐训练。三者（算力、数据规模、数据利用方式）的协同，最终推动了大模型从实验室走向实际应用。

本书将重点介绍智谱 AI (Zhipu AI) 的相关模型，该公司成立于 2019 年。由清华大学计算机系知识工程实验室 (KEG) 技术成果转化而来，致力于打造认知智能大模型，专注于做大模型的中国创新。

2020 年，智谱 AI 开始专注大模型算法研究。2021 年 9 月，设计 GLM 算法，发布开源百亿大模型 GLM-10B。2022 年，合作研发了中英双语千亿级超大规模预训练模型 GLM-130B，该模型是亚洲唯一入选斯坦福大学大模型中心评测的大模型，准确性、恶意性与 GPT-3 持平，鲁棒性和校准误差表现最佳。同年还发布了代码生成模型 CodeGeeX 和 100+ 语言预训练模型 mGLM-1B。

2023 年，智谱 AI 推出千亿基座的对话模型 ChatGLM 及其单卡开源版本 ChatGLM-6B，全球下载量超过 800 万。后续又发布了 ChatGLM2、ChatGLM3 等升级版本。2024 年 1 月，推出新一代基座大模型 GLM-4。2025 年 7 月发布的 GLM-4.5，在全球模型中排名第三，在国产模型和开源模型中均位居第一。

智谱 AI 在全球多地设有分研究中心，包括美国、英国、中东、新加坡等地。其在技术领域影响力较大，github 排名总榜曾位居世界第五。

公司构建了丰富的产品矩阵，涵盖多种模态和功能，包括 AI 提效助手智谱清言、高效率代码模型 CodeGeeX、多模态理解模型 CogVLM、文生图模型 CogView、文生视频模型 CogVideo 等。这些模型主要分为理解、创作/生成、安全、自动化执行等几个大方向，广泛应用于自然语言处理、计算机视觉等领域，为中国科协、华为、腾讯等 1000 余家企事业单位提供服务。

## 1.2 GLM VS GPT

智谱 AI 以自主研发的 GLM (General Language Model，通用语言模型) 框架为技术基底，逐步构建起覆盖多场景、多模态的系列大模型产品体系，其发展路径与技术定位对标 OpenAI 的 GPT 系列模型，如图??所示，具体体现为：

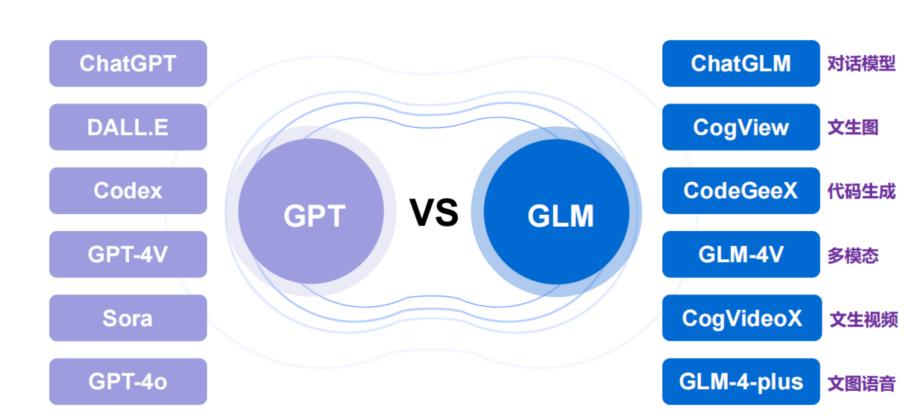


图 1.1: GLM VS GPT

- **核心框架的技术对标:** GLM 框架采用双向注意力与自回归机制融合的创新架构，既具备类似 GPT 系列的长文本生成能力，又强化了对复杂语义的理解精度，在模型设计理念上与 GPT 的 Transformer 架构形成功能性呼应，为系列化模型开发奠定了统一技术基础。
- **产品迭代的路径对标:** 从早期 1300 亿参数的 GLM-130B（对标 GPT-3），到对话优化的 ChatGLM 系列（对标 ChatGPT），再到支持多模态交互的 GLM-4 及 GLM-4.5（对标 GPT-4），智谱 AI 的模型迭代既延续了 GPT 系列“参数规模扩张 + 能力边界拓展”的逻辑，又针对中文语境和垂直场景进行了深度优化，形成了与 GPT 系列在技术代际上的对应关系。
- **功能覆盖的场景对标:** 如同 GPT 系列从文本生成向代码、图像、工具调用等领域延伸，智谱基于 GLM 框架推出的 CodeGeeX（代码生成，对标 GPT-4 Code）、CogVLM（多模态理解，对标 GPT-4V）、Agent 工具链（自动化执行，对标 GPT-4 的插件生态）等产品，构建了覆盖文本、代码、语音、图像、视频及智能执行的全栈能力，实现了与 GPT 系列在功能场景上的全面对标。

这种以 GLM 框架为核心的系列化布局，既体现了对国际前沿技术路线的对标跟进，也通过本土化技术创新形成了差异化竞争力，推动其成为国内大模型领域与 OpenAI GPT 系列具有直接可比性的技术体系。

2024 年 8 月，智谱 AI 在 KDD 国际数据挖掘与知识发现大会上发布了新一代基座大模型 GLM-4-Plus。该模型在多个指标上接近甚至超越了 GPT-4o，如图??所示。

**GLM-4-plus (24年8月发布) 模型性能全面提升, 中/英文接近GPT-4o水平**

	对齐能力 (中文)		基础能力 (英文)				指令跟随能力 (中英)		推理能力 (中英)
	Align-Bench	MMLU	MATH	GPQA	LCB	NCB	IFEval	Logic Game	
Claude 3.5 Sonnet	80.7	88.3	71.1	*56.4	49.8	53.1	80.6	30.4	
Llama 3.1 405B	60.7	88.6	73.8	*50.1	*39.4	50.0	83.9	23.1	
Gemini 1.5 Pro	74.7	85.9	67.7	46.2	33.6	42.3	74.4	22.5	
GPT-4o	83.8	88.7	76.6	*51.0	*45.5	52.3	81.9	24.1	
GLM-4-Plus	83.2	86.8	74.2	50.7	*45.8	50.4	79.5	25.6	
GLM-4-Plus/ GPT-4o	99%	98%	97%	99%	101%	96%	97%	106%	

图 1.2: GLM-4-Plus 与 GPT-4o 的中英文能力对比

GLM-4-Plus 在语言理解、指令遵循、长文本处理等方面性能得到全面提升。在各大语言文本能力数据集上，它获得了与 GPT-4o 及 405B 参数量的 Llama3.1 相当的水平。其通过更精准的长短文本数据混合策略，取得了更强的长文本推理效果，长文本能力比肩 GPT-4o，超过 Gemini 1.5 Pro 和 Claude Sonnet 3.5。

在与 GPT-4o 的全面较量中，GLM-4-Plus 已经可以在大多数任务上做到逼近，甚至在某些逻辑推理等任务上实现了超越。在最新的 SuperBench 大模型评测中，GLM-4-Plus 位列世界前三，打破了此前国外模型垄断前三甲的局面。

ChatGLM 体现出较强的语义推理能力，如图??所示。

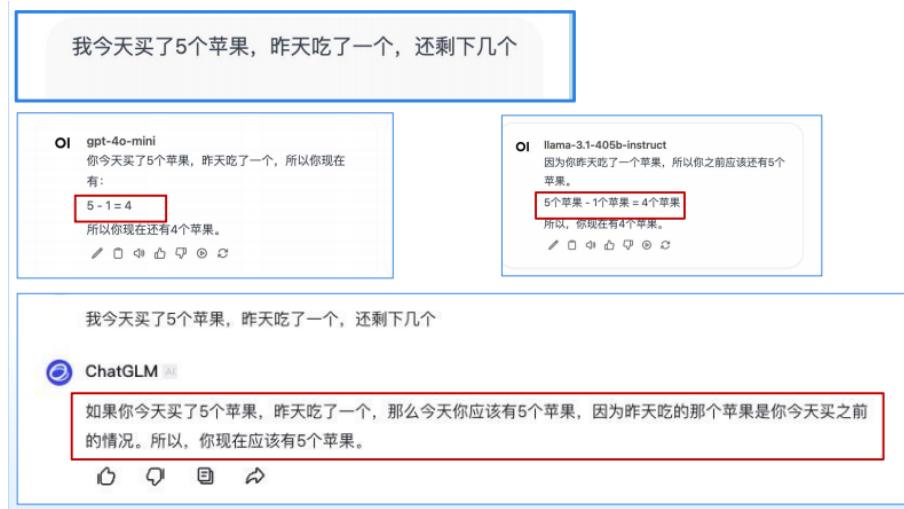


图 1.3: ChatGLM 体现出较强的语义推理能力

ChatGLM 能够精准识别、理解图片中的内容，如图??所示。



图 1.4: ChatGLM 能够精准识别、理解图片中的内容

ChatGLM 能够理解视频内容并回答相应的问题，如图??所示。



Q: 绿色球衣的球员在接到球后朝哪个方向跑了？

A: 穿绿色球衣的球员在接到球后朝球场的右侧跑了。

Q: 这个视频的亮点是什么？它发生在第几秒？

A: 视频的亮点是穿绿色球衣的球员投篮得分。它发生在第5秒。

图 1.5: ChatGLM 能够理解视频内容并回答相应的问题

ChatGLM 能够理解指令要求生成高清图片，如图??所示。

一张新鲜烘焙的可颂特写图片，可颂拥有金黄色的酥脆外皮和柔软分层的内部，放置在乡村风格的木质桌面背景上。可颂周围撒着一些糖粉，旁边是一小杯冒着热气的咖啡，营造出一个温馨的清晨早餐场景。

一只巨大的机械蜘蛛，眼睛发光，齿轮复杂，在月光下高耸的摩天大楼之间编织着闪耀的银色蛛网。



图 1.6: ChatGLM 能够理解指令要求生成高清图片

ChatGLM 能够理解复杂指令生成正确图片，如图??所示。

三个立方体堆叠在一起。最上面是一个红色立方体，它下面也是一个红色立方体。  
中间的红色立方体放在一个绿色立方体上，绿色立方体在最下面。

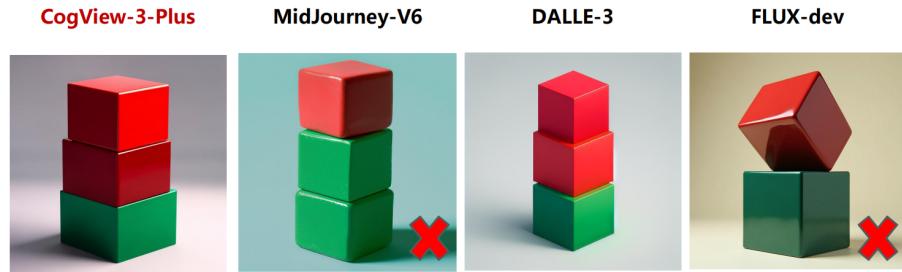


图 1.7: ChatGLM 能够理解复杂指令生成正确图片

ChatGLM 能够理解改图指令生成正确的修改后的图片，如图??所示。



图 1.8: ChatGLM 能够理解改图指令生成正确的修改后的图片

ChatGLM 根据图片与指令生成的视频效果，如图??、??和??所示。

- 让GLM-4-9B和OpenAI、Gemini Pro以及Claude-3打牌。

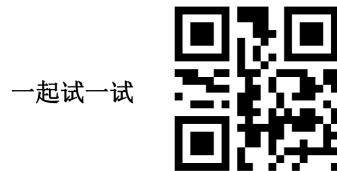


图 1.9: ChatGLM 根据图片与指令生成的视频效果

让画面里的主体  
开始弹吉他



冒出一只小乌龟



试一试



试一试



图 1.10: ChatGLM 根据图片与指令生成的视频效果

图 1.11: ChatGLM 根据图片与指令生成的视频效果

### 1.2.1 模型技术架构与特点

目前主流的自研大模型有 GLM、P-Tuning、CogView 等，而这些模型可以做到从通过世界知识进行一阶推理，到抽象与复杂的 COT 推理，展现了强大的推理能力。

### 通用语言模型 (GLM)

架构创新：采用自回归填空架构，统一自然语言理解与生成任务。与自回归 (GPT)、自编码 (BERT)、编码器-解码器 (T5) 等架构相比，在不同任务上有独特优势。之前没有通用预训练框架能同时在理解、有条件生成和无条件生成任务中达到最优，GLM 实现了突破。如图??所示。

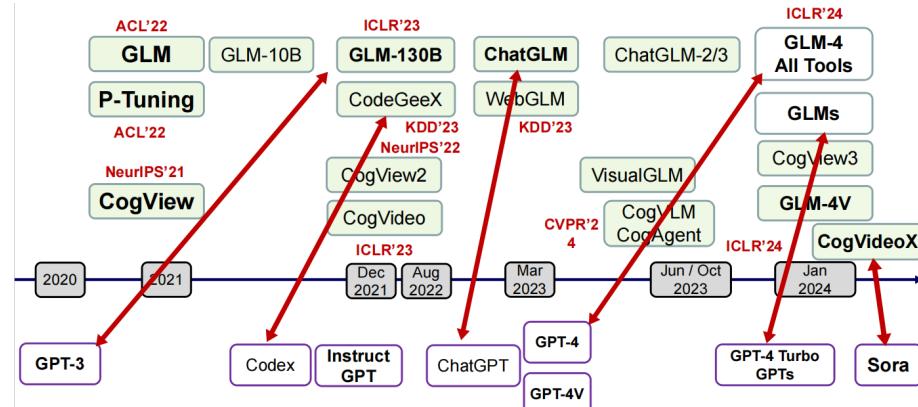


图 1.12: GLM 系列模型框架图

模型优势：GLM-130B 是亚洲唯一入选斯坦福大学大模型中心报告评测的模型，准确性、恶意性与 GPT-3 持平，鲁棒性和校准误差表现最佳。具备双语高精度，可在 4\*RTX3090 运行，支持模型量化，推理加速 2-3 倍，适配多种芯片和平台。GLM-4-plus 在通用能力、指令追随、中文对齐、代码、数学、安全等方面与 GPT-4o 性能相当。GLM3-130B 性能对比如图??所示。



图 1.13: GLM3-130B 性能对比

### 国际影响力

学术引用：Cogview 被 Yann LeCun 等学者在多篇文章中引用，作为文本到图像生成的代表算法；Philip 认为项目提出的，ChatGLM 模型是当下最受欢迎的语言模型之一；P-tuning 算法被认为是软性提示嵌入与硬性提示词结合的代表工作，相关研究在学术领域得到认可。

大会与报道：模型在国际顶级 AI 大会（如 ICLR’24、WWW’24）展示，ChatGLM 模型被 Nature 报道，相关研究人员成为受邀参与领导人座谈的 AI 专家，提升了模型在国际上的知名度和影响力。

### 多模态模型（Cog 系列）

核心模块优势：以 CogVideoX 为例，如图??所示，核心模块 3D Transformer 可以实现文本与视频的 full attention，提升语义理解。同时通过 Text Expert AdaLN 和 Vision Expert AdaLN 实现文本和视频对齐。与 2D VAE 相比，3D VAE 在连续性和压缩倍数上表现更好。在进行 VAE 训练时，需要在时间维度上做 context parallel。



图 1.14: Cog 系列模型

训练流程优化：data pipeline 包括视频 caption，训练过程采用 Frame Pack、Image-Video joint training、Mixed duration training、Progressive Training 等方法，解决视频长度变化、任务差距等问题，提高训练效率和模型性能。其中视频 caption 需要详细描述视频中的内容，开源的视频理解模型视频描述能力较差。Version1：图像重标注对每帧标注，结合视频 caption 用大语言模型得到视频描述。Version2：version 1 得到的数据微调 CogVLM2-Video，得到一个端到端的视频描述。与训练对应，推理时输入详细的 prompt，才能最大限度激发模型能力。

### 1.2.2 模型能力表现

从数据??可以看出，CogVideoX 在所有评估指标上均表现出色，得分均高于其他模型。这表明 CogVideoX 在生成视频时，不仅能够准确捕捉人类动作和场景细节，还能在动态程度、多个对象处理、外观风格和动态质量方面表现出色。此外，其 GPT4o-MT 得分也显著高于其他模型，进一步证明了其在视频生成任务中的优越性能。

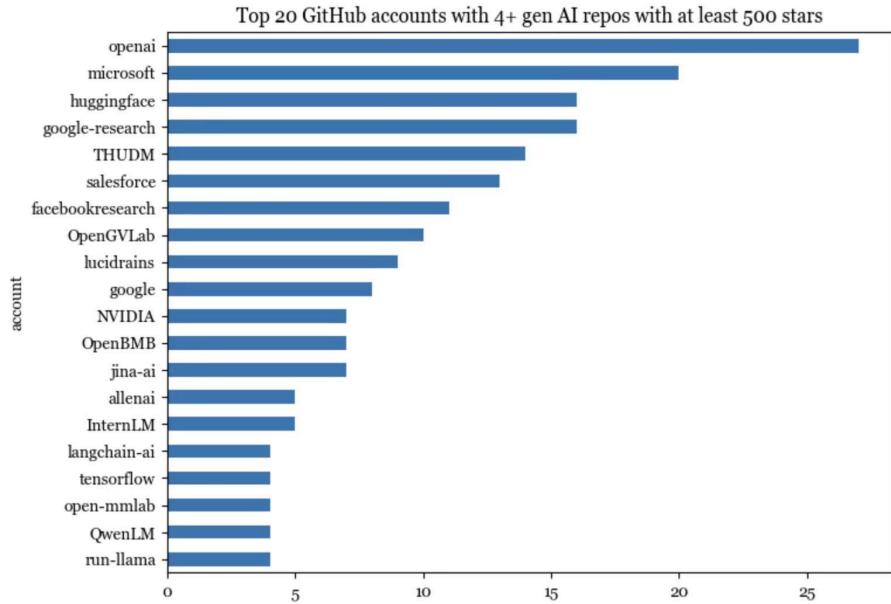


图 1.15: CogVideoX 与主流模型对比

### 1.2.3 开源项目

#### 模型开源情况

多个模型开源，包括 ChatGLM-6B、ChatGLM2-6B、ChatGLM3、GLM-130B、CodeGeeX、CodeGeeX2、CogVideo 等，涵盖多种类型，在 GitHub 上获得较高关注度，如 ChatGLM-6B 项目获 40,324 颗星。

#### 项目排名

在 GitHub 公布的超过 500 星的 AI 项目数排名??中，THUDM 在拥有 4 个以上生成式 AI 仓库且至少 500 星的前 20 个 GitHub 账户中表现出色，展示了其在开源 AI 项目领域的影响力。

	OpenAI	我们的思考
Level 1	有语言能力的AI	AI学会使用语言，在大多数自然语言任务上突破图灵测试
Level 2	人类水准的问题求解能力	AI学会求解问题，涌现世界知识和类人的复杂逻辑推理能力，在问题求解方面突破图灵测试
Level 3	使用工具，系统可以执行动作	AI学会使用工具，利用工具完成多数人类物理世界问题，在工具使用方面突破图灵测试
Level 4	AI将能自己发明创新	AI通过自我学习，实现GPT到GPT-zero的升级，具备自我批判、自我改进以及自我反思能力
Level 5	AI可以融入组织或者自成组织	AI能力全面超越人类，具备探究科学规律、世界起源等终极问题的能力

图 1.16: GitHub 公布的超过 500 星的 AI 项目数排名

### 1.3 大模型的 AGI 之路

#### 1.3.1 人工智能分级

人工智能分级，如图??所示。



图 1.17: 人工智能分级

#### 1.3.2 教会大模型使用工具

GLM-4 系列演进路径（语言 → 多模态 → 工具使用）清晰展现了大模型从单一能力向复杂智能的发展逻辑，这也是当前 AI 领域的重要趋势。以下从三个阶段的核心能力、技术特点和应用场景展开说明：

### 1. GLM-4：纯语言大模型

- 核心能力：专注于自然语言处理（NLP），包括文本生成、问答、翻译、逻辑推理等。
- 技术特点：
  - 基于 Transformer 架构的深度预训练模型，通过海量文本数据学习语言规律。
  - 优化长文本理解和复杂语义建模，支持多轮对话的连贯性。
- 应用场景：智能客服、文本创作、知识问答、代码生成等纯文本交互场景。

### 2. GLM-4V：多模态大模型

- 核心能力：在 GLM-4 语言能力基础上，新增 \*\* 视觉理解能力 \*\*，可处理图像、视频等视觉输入，并实现“文本-视觉”跨模态交互。
- 技术特点：
  - 引入视觉编码器（如 ViT 变体），将图像/视频转化为与语言模型兼容的特征向量。
  - 通过跨模态注意力机制，实现“看图说话”“图像内容解析”“文本生成图像描述”等功能。
- 应用场景：图像内容识别（如物体检测、场景分类）、图文结合的问答（如“这张图里有几只猫？”）、视频内容摘要等。

### 3. GLM-4V-All Tools：具备工具使用能力的多模态模型

- 核心能力：在 GLM-4V 的多模态基础上，新增 \*\* 工具调用能力 \*\*，可自主选择并使用外部工具（如计算器、搜索引擎、API 接口等）解决复杂问题。
- 技术特点：
  - 引入“工具规划模块”，通过逻辑推理判断何时需要调用工具、调用哪种工具，以及如何解析工具返回结果。
  - 支持多工具协同，例如先通过图像识别工具分析图片内容，再调用搜索引擎补充相关知识，最后生成综合回答。
- 应用场景：复杂任务处理（如“根据这张股票走势图，结合最新财经新闻，预测未来一周的趋势”）、跨领域问题解决（如“用计算器算一下这个公式，再把结果翻译成英文”）等。

### 总结：演进逻辑与价值

从 GLM-4 到 GLM-4V-All Tools，本质是模型从“被动理解”到“主动交互”的升级：

- 语言 → 多模态：扩展了信息输入的维度，更贴近人类“视听结合”的认知方式；

- 多模态 → 工具使用：突破模型自身知识和能力的局限，通过外部工具延伸智能边界，更接近“通用人工智能（AGI）”的雏形。

这一路径与 GPT-4（语言）→GPT-4V（多模态）→GPT-4 with Tools 的演进高度相似，反映了大模型向“更全面、更实用”方向发展的行业共识。

### 1.3.3 未来通用人工智能 AGI 之路在哪里？

超级智能与超级对齐，如图??所示。



图 1.18: 超级智能与超级对齐

### 1.3.4 AGI 应用栈

AGI 应用栈是指实现通用人工智能（AGI）相关应用所需的技术架构和组件集合。根据国际 AGI 标准，其涉及重构计算、重构 OS、AGI 助手、重构应用和超级应用等方面，以下是具体介绍：

- 重构计算：**核心是“模型即产品（MaaS）”模式。通过模型服务平台，提供如 GLM 等大模型的推理、训练等服务，结合数万亿 token 的中英文文本、数亿图/视频数据，利用大模型的自学习能力，为各种应用赋能。同时，依赖智谱智算平台等硬件生态，实现跨平台多处理器支持，以及芯算一体专用芯片的高效计算，满足 AGI 对大规模数据处理和复杂模型运算的需求。
- 重构 OS：**以大语言模型为中心的通用计算系统，如 GLM OS。它不仅要管理主板、外围设备等硬件资源，还要提供内核服务，支持多模态处理，实现对物联网设备的控制和网页自动化等功能。此外，还需具备长上下文处理能力，结合知识库存储，为应用提供强大的知识支撑和运算环境，使各种 AGI 应用能够流畅运行。
- AGI 助手：**基于 AGI 技术的智能助手，能够理解用户自然语言指令，完成复杂任务。例如，可根据用户需求筛选特定价格区间且包邮的商品，查询地点距离，或在订单中找到指定外卖

并重新下单等。它具备智能规划、分析汇总等能力，可调用外部工具和知识，通过迭代交互为用户提供准确服务。

- 重构应用：对传统应用进行重构，摒弃传统菜单表单式交互，转向“业务窗口式服务 + 智能大搜”模式，支持自然语言交互。例如在金融领域，可实现智能投资分析与风险预警；在医疗领域，能辅助诊断和制定治疗方案等。通过整合大模型能力与向量数据库，构建私有知识库，实现精准的信息匹配与生成，为各行业应用提供更智能、高效的解决方案。
- 超级应用：是 AGI 应用的高级形态，它融合了多种技术和功能，能够跨领域、跨平台为用户提供综合性的服务，具备强大的智能决策和自主学习能力，可根据用户行为和环境变化不断优化服务，实现真正意义上的智能交互和复杂任务处理，如同时整合电商、出行、金融、生活服务等多种功能，根据用户需求自动规划最优生活方案等。

### 1.3.5 搜索的革命

“搜索的革命”正围绕着“精准定位信息、可靠、快速、避免幻觉、可追溯”这些核心目标展开，本质上是从传统“关键词匹配”向“智能理解与验证”的范式升级。以下从技术突破、关键特性和应用场景三个维度解析这场革命：

#### 一、技术突破：支撑搜索革命的底层逻辑

##### 1. 大模型驱动的语义理解

- 传统搜索依赖关键词匹配，容易因“词不达意”导致结果偏差（例如搜索“苹果”可能返回水果或品牌）。
- 新一代搜索基于大语言模型（如 GPT-4、GLM-4）的深度语义解析，能理解用户 query 的上下文、隐含意图甚至情感（例如“推荐适合新手的苹果产品”会精准定位电子设备）。

##### 2. 多模态融合搜索

- 突破纯文本限制，支持“文本 + 图像 + 语音”混合输入（例如拍一张植物照片，搜索“这是什么花”），通过跨模态模型将视觉特征与文本知识关联，实现更直观的信息定位。

##### 3. 实时数据与知识图谱结合

- 传统搜索引擎的索引更新存在延迟，而新一代搜索通过对接实时数据库、API 接口（如新闻、天气、股票数据），确保信息时效性。
- 结合知识图谱（实体关系网络），可追溯信息的来源和关联（例如搜索“北京冬奥会冠军”，不仅返回名单，还能关联其国籍、项目成绩等结构化数据）。

##### 4. 工具链协同验证

- 引入“搜索-验证”闭环：当模型对答案存疑时，自动调用计算器、权威数据库等工具验证（例如计算“2024 年世界杯冠军是谁”，会实时检索官方结果而非依赖过时训练数据），从源头减少“幻觉”。

## 二、核心特性：从“找到信息”到“信得过、用得上”

### 1. 精准定位：从“海量结果”到“答案直达”

- 传统搜索返回的是网页链接列表，用户需自行筛选；新搜索通过“意图拆解 + 结果聚合”，直接输出精准答案（例如搜索“今天上海到北京的高铁票余票”，会直接显示车次、时间和余票数量，而非跳转多个购票网站）。
- 支持复杂逻辑查询（例如“2023 年全球 GDP 排名前三的国家中，人均收入最高的是哪个”，通过多步推理定位信息）。

### 2. 可靠性与避免幻觉：从“可能正确”到“可验证”

- 来源标注：**所有答案附带明确来源（如“数据来自世界银行 2023 年报告”“信息引用自《自然》期刊论文”），用户可追溯原始出处。
- 冲突校验：**当不同来源信息矛盾时，模型会主动提示差异并分析可信度（例如“关于某事件的时间，A 新闻报道为 3 月，B 官网显示为 4 月，建议以官网为准”）。
- 拒绝不确定答案：**对于超出知识范围或缺乏权威依据的问题（如“2050 年房价预测”），会明确说明“无法提供可靠答案”，而非生成虚构内容。

### 3. 快速与高效：从“等待加载”到“实时响应”

- 依托边缘计算和分布式索引优化，将搜索响应时间压缩至毫秒级，尤其支持移动端、物联网设备的低延迟交互（例如智能手表语音搜索“附近的 24 小时药店”，1 秒内返回结果）。
- 结合用户历史行为，实现“个性化预加载”（例如经常搜索科技新闻的用户，打开 APP 时会优先缓存最新科技资讯）。

### 4. 可追溯性：从“黑箱结果”到“过程透明”

- 提供“搜索链路可视化”功能，用户可查看模型的推理过程：例如“如何解这道数学题”，不仅返回答案，还能展示“调用了计算器 → 验证公式正确性 → 代入数值计算”的步骤，甚至回溯每一步的参考数据来源。

## 三、应用场景：搜索革命正在改变的领域

- 学术研究：**科研人员搜索“某基因的最新研究”时，系统会自动筛选近 3 年核心期刊论文，标注实验数据来源，并提示“该结论在 2 篇论文中被验证，1 篇存在争议”。

- 医疗健康**: 用户上传体检报告照片, 搜索“指标异常原因”, 系统结合医学知识库和实时临床指南, 给出可能病因, 并标注“信息来自国家卫健委官网”, 同时建议“以医生诊断为准”。
- 商业决策**: 企业搜索“某行业 2024 年增长率”, 会聚合统计局、行业协会、第三方智库的多方数据, 生成对比图表, 并注明各数据源的统计口径差异。
- 教育学习**: 学生搜索“相对论的通俗解释”, 系统不仅用简单语言说明, 还会链接到权威科普视频、物理学家的原始论文摘要, 并提示“这一步推理参考了《时间简史》第 3 章”。

### 总结: 搜索革命的本质

这场革命的核心是让搜索从“信息搬运工”升级为“智能验证者”——不仅要快速找到信息, 更要确保信息的准确性、可追溯性, 同时理解用户的真实需求。未来, 随着 AGI 技术的融入, 搜索可能进一步进化为“主动服务”: 例如根据用户的健康数据, 提前推送相关医疗资讯并附权威来源, 真正实现“信息可信、决策有据”。

### 1.3.6 计算的革命

计算的革命, 如图??所示。



图 1.19: 计算的革命

### 1.3.7 2024-AGI 元年?

- 趋势一: 用计算模型可描述的认知问题, AI 很快超过人类水平很快 =5-20 年 (2019 年)
- 趋势二: 通用问题上, AI 通过自我学习, 实现 GPT 到 GPT-zero 的升级, 能力全面超越人类, 实现超级智能; 探究科学规律、世界起源等终极问题? 很快 =5-20 年 (2023 年)
- 趋势三: OS 的革命, 如图??和??所示。GLM OS: LLM-centric General Computing System

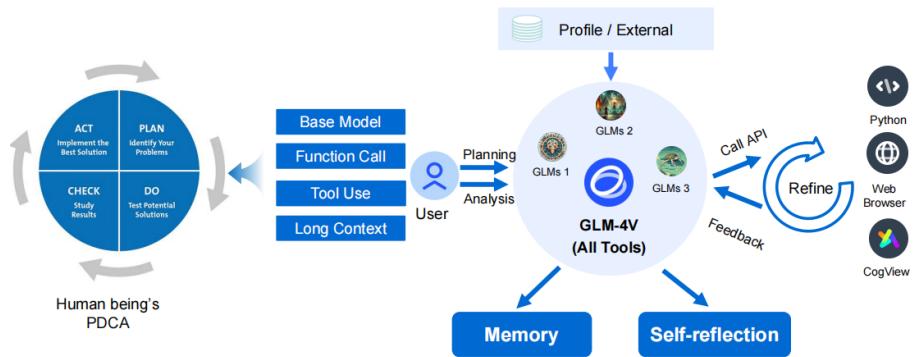


图 1.20: OS 的革命

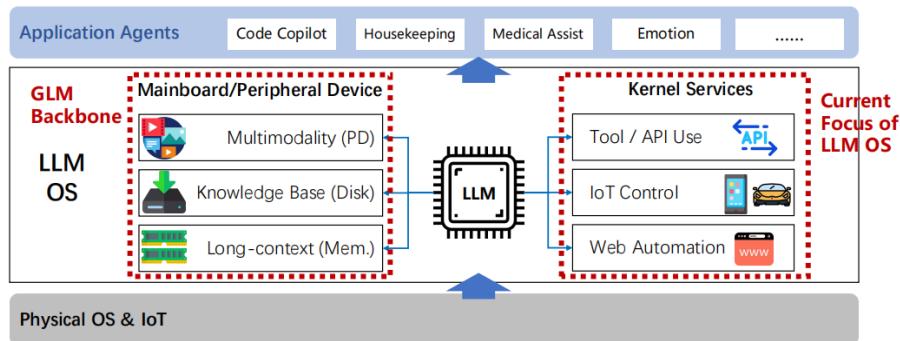


图 1.21: OS 的革命

## 1.4 致谢

技术贡献: 清华大学知识工程实验室 (KEG) 智谱 AI 清华大学 PACMAN 实验室 清华大学自然语言处理实验室 清华大学交互式人工智能实验室

算力赞助: 前期算力: 中科曙光、鹏城实验室、神威·海洋之光千亿训练: 济南超算中心 (GLM-130B) GLM 训练: 智谱 AI

## 参考文献

- Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, and Hongning Wang. Black-box prompt optimization: Aligning large language models without model training. *arXiv preprint arXiv:2311.04155*, 2023.
- Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, and Chang Zhou. Cogview: Mastering text-to-image generation via transformers. *Advances in neural information processing systems*, 34: 19822–19835, 2021.
- Ming Ding, Wendi Zheng, Wenyi Hong, and Jie Tang. Cogview2: Faster and better text-to-image generation via hierarchical transformers. *Advances in Neural Information Processing Systems*, 35:16890–16902, 2022.
- Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360*, 2021.
- Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, and Chenhui Zhang. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.
- Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. Cogvideo: Large-scale pre-training for text-to-video generation via transformers. *arXiv preprint arXiv:2205.15868*, 2022.
- Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, and Aohan Zeng. Webglm: Towards an efficient web-enhanced question answering system with human preferences. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4549–4560, 2023.
- Jiayan Teng, Wendi Zheng, Ming Ding, Wenyi Hong, Jianqiao Wangni, Zhuoyi Yang, and Jie Tang. Relay diffusion: Unifying diffusion process across resolutions for image synthesis. *arXiv preprint arXiv:2309.03350*, 2023.
- Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, and Ji Qi. Cogvlm: Visual expert for pretrained language models. *Advances in Neural Information Processing Systems*, 37:121475–121499, 2024.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, and Hanyu Lai. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agent-tuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, and Shan Wang. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5673–5684, 2023.

# 第二章 深度学习基础

**【摘要】**本章介绍了深度前馈网络（Deep Feedforward Network）、损失函数、反向传播算法（Backpropagation Algorithm）、激活函数以及常见的模型架构。讨论了卷积神经网络（CNN）的卷积操作、池化操作，不同的 CNN 模型架构及其在二维（2D）和三维（3D）上的应用。此外，还介绍了循环神经网络（RNN）的基础结构、双向 RNN、编码器-解码器架构、长期依赖（long-term dependencies）以及长短期记忆网络（LSTM）和门控循环单元（GRU）等高级模型。

## \*2. 深度学习基础 \*

本章介绍了深度前馈网络（Deep Feedforward Network）、损失函数、反向传播算法（Backpropagation Algorithm）、激活函数以及常见的模型架构。讨论了卷积神经网络（CNN）的卷积操作、池化操作，不同的 CNN 模型架构及其在二维（2D）和三维（3D）上的应用。此外，还介绍了循环神经网络（RNN）的基础结构、双向 RNN、编码器-解码器架构、长期依赖（long-term dependencies）以及长短期记忆网络（LSTM）和门控循环单元（GRU）等高级模型。

2024 版本：唐杰、杜晋华、杨超群、吴彦辰

### 2.0.1 2.1 引言 \*\* (删?) \*\*

#### 2.1.1 人工智能

人工智能（AI）的架构和运作方式：首先，从最底层的“环境”（Environment）开始，这个环境包括了 AI 所处的所有外部条件和变化。它相当于现实世界中的输入信息，这些信息通过各种传感器（Sensing）采集，进入 AI 系统。

接下来，感知是 AI 系统接收环境信息的第一步，类似于人类的五感。通过感知，AI 能够接收到大量来自环境的原始数据，例如视觉、声音、温度等。

获取到环境的感知数据后，信息传递到“思维”（Thinking）模块。这部分相当于 AI 的“大脑”，负责数据处理和决策。思维过程包括了模式识别、预测分析以及复杂的逻辑判断等等。它是 AI 能否做出智能决策的核心所在。

有了处理后的决策信息，AI 会将这些信息传递给“行动”（Acting）模块。行动模块是系统执行各种任务的部分，类似于我们的人体动作。根据从思维模块中的决策，AI 在实际环境中执行具体的操作，例如移动一只机械手臂或者回复一个文本消息。

高级一些的 AI 系统可能还会引入“情感和动机”（Emotion, motivation）模块，这就像人类的心智。虽然这些因素现在还在研究开发中，但赋予 AI 系统以情感和动机能够使之更好地理解和

响应人类需求，从而提供更加人性化的服务。

值得注意的是，这些模块之间是相互联系和循环的。行动后的结果会再次影响到环境，产生新的感知信息，从而再次进入这个循环。这种方式确保了 AI 能够不断调整和优化自身的行为和决策。

总结来说，人工智能通过“感知”、“思维”及“行动”三个基本环节的循环往复，不断地与环境互动，逐步实现了对复杂问题的解决能力，甚至可能达成人类智能级别的认知和反应水平。

### 2.1.2 人工智能的发展历史

第一代，时间范围大约在 1950 年代末到 1980 年代初。标志性事件之一是 1956 年，这一年通常被认为是“人工智能这一术语首次被提出的时间。这段时间的主要技术是符号 AI (Symbolic AI)，这种技术利用符号和逻辑规则来模拟人的推理过程。

第二代的时间范围大约在 1980 年代到 2010 年代初。这个时期标志性事件之一是 1980 年代的专家系统 (Expert System) 出现，专家系统利用知识库和推理机制来解决特定领域的问题。此外，1980 年清华开始了 AI 方向的研究。在 1990 年左右，专家系统处于活跃的发展期。这段时间的主要技术是专家系统，通过使用规则和知识库来解析特定领域的问题。

第三代从 2010 年代初至今。在这个时期的重要标志性事件之一是 2011 年的深度学习 (Deep Learning) 的迅速发展，深度学习利用多层神经网络极大提高了图像和语言等复杂任务的处理能力。到了 2020 年代中期到 2030 年，大规模语言模型 (Large Language Models, LLM) 继续发展，标志着 AI 在自然语言处理和更加复杂任务中的应用进一步提升。这个时期的主要技术包括深度学习和大规模语言模型，在大数据和强大计算能力的支撑下，AI 能够处理更加复杂和精确的任务。

其实，AI 历史上存在几次“冬天”和复苏期。每次低谷期之后，随着新理论和技术的突破，AI 领域都会迎来新的高峰。例如，符号 AI 在 20 世纪 80 年代初期出现了低谷，但随着专家系统的出现，AI 在 90 年代迎来了新一轮发展。进入 21 世纪以后，深度学习和大规模语言模型的出现再度推动了 AI 的快速发展。

### 2.1.3 GPT 与 GLM

在探讨现代人工智能语言模型的发展时，我们可以通过比较两种具有代表性的语言模型家族——GPT (Generative Pre-trained Transformer) 和 GLM (General Language Model) 来深入了解它们各自的功能和典型应用。

基于 GPT 技术的应用：GPT 家族基于生成预训练 Transformer 技术，涵盖了一系列应用，这些模型在自然语言处理和生成方面显示出了强大的能力：

1. ChatGPT：一个用于对话生成的模型，能够处理和生成自然语言对话。
2. DALL·E：一个图像生成模型，能够根据文本描述生成逼真的图像。
3. Codex：一个用于代码生成的模型，能够从自然语言描述生成代码，支持多种编程语言。
4. GPT-4V：一个多模态模型，可以处理包括文本、图像等多种形式的数据。
5. Sora：文生视频模型。

6. GPT-4o: 具备更强的推理能力。

基于 GLM 技术的应用：GLM 技术也拥有其独特的模型家族，适用于多种生成任务，这些模型强调通用性和多模态处理能力：

1. ChatGLM: 一个对话生成模型，设计用于自然语言对话的处理和生成。
2. CogView: 一个用于生成图像的模型，通过文本描述生成高质量的图像。
3. CodeGeeX: 一个代码生成模型，能够根据自然语言编写代码，支持广泛的编程任务。
4. GLM-4V: 一个多模态模型，能够处理和生成包括文本、图像在内的多种数据形式。
5. CogVideoX: 一个视频生成模型，通过文本描述生成视频内容，开拓了新的生成领域。
6. GLM-4-plus: 一个更为先进的多模态模型，不仅处理文本和图像，还扩展到音频数据的生成和处理。

这两种模型家族在功能上存在许多相似之处，包括自然语言对话生成、图像和视频生成、代码生成以及多模态数据的处理。然而，它们在技术实现、性能表现、应用领域和独特功能上存在一定差异。

#### 2.1.4 大模型的架构

大规模语言模型的架构及其演变方，涵盖了三种主要的模型类型：RNN 风格的 Transformer、传统 Transformer 和状态空间模型。

传统 Transformer 是当前许多大规模语言模型的核心架构，以其强大的并行处理能力和高效的注意力机制广泛应用。其中的例子有 GPT、Llama、GLM、BERT 和 T5，这些是现已有的代表性模型。2023 年的 Griffin 和 2024 年的 Megalodon 在局部注意力机制上有所改进，以提升性能和效率。另有在 2024 年推出的 Lory 和 DeepSeek-v2，这些模型增强了子模块优化和专家混合的使用。

其中一个改进分支是：RNN 风格 Transformer，这些模型结合了循环神经网络和 Transformer 的元素，以提高计算效率和模型性能。具体例子有 2023 年的 RWKV 和 RetNet，它们集中在注意力机制的改进上。2024 年推出的 xLSTM 是一种不依赖于传统注意力机制的模型。另一款模型是 YoCo，它进一步优化了线性和自由形式的注意力机制。

状态空间模型侧重于利用信号处理技术和状态空间表示，以增强序列建模能力。现有的例子包括在 2023 年之前存在的 S4 和 H3，这些展示了状态空间模型技术的早期应用。2023 年的 Mamba 是当前最新的状态空间模型。在 2024 年推出的 Jamba 是这种技术最新的发展。

#### 2.1.5 机器学习概览

机器学习任务分为四类：1. 分类（Classification）2. 回归（Regression）3. 聚类（Clustering）4. 降维（Dimensionality Reduction）每种任务类型都有不同的算法。

**分类 (Classification)**：分类任务旨在预测数据属于哪一类。如果你有文本数据，使用 Naive Bayes。如果样本数量少于 100K，尝试 Linear SVC，如果不工作，使用其他算法 (SVC, Ensemble Classifiers, SGD Classifier 等)。如果有内核近似 (Kernel Approximation) 则优先选择相应的分类器。

**回归 (Regression)**：回归任务旨在预测数值。如果特征重要，尝试 Ridge Regression 或 SVR (支撑向量回归)。如果样本数量少于 100K，尝试 SGD Regressor。如果你样本超过 50，若不工作，则获取更多数据。

**聚类(Clustering)**: 聚类任务旨在对未标记数据进行分组。若样本数量少于 10K，尝试 MiniBatch KMeans 或 KMeans。若类别数已知，使用 MiniBatch KMeans 或 MeanShift。若大于 10K 样本，并且不工作，使用其他算法。

**降维 (Dimensionality Reduction)**：降维任务用于简化数据以减少计算负载。若样本少于 10K，尝试 Randomized PCA。若需要内核近似，选择相应算法 (如 Isomap 或 Spectral Embedding)。否则尝试更多其他特定降维算法。

**2.1.5.1 序列标注** 在自然语言处理领域中，序列标注是一项重要的任务，常见的算法包括支持向量机 (SVM)、隐马尔可夫模型 (HMM)、最大熵 (ME)、最大熵马尔科夫模型 (MEMM) 以及条件随机场 (CRF)。以下是这些算法的优缺点解析。

支持向量机 (SVM) 是一种强大的分类算法，充分利用多种特征进行线性可分的二分类。其主要优点是分类能力强，但计算复杂度高，这是它的一个明显缺点。

隐马尔可夫模型 (HMM) 常用于处理具有隐含状态的序列数据。HMM 的优点在于它的方法有效且常用，但存在特征选择受限，不能应用较长上下文文本特征的问题。

最大熵 (ME) 是一种灵活的无偏算法，解决了隐马尔可夫模型在处理上下文特征时的局限性。然而，其计算复杂度仍然较高，这使得最大的缺点在于运算量大，适用广泛性有所限制。

最大熵马尔科夫模型 (MEMM) 进一步解决了最大熵的计算复杂度问题。尽管 MEMM 能够有效地处理复杂特征，但其标注偏置问题依然存在，这成为了该算法的主要缺点。

条件随机场 (CRF) 算法在序列标注中显示出优越的表现，解决了 MEMM 标注偏置的问题。CRF 能够高效地进行训练并适应复杂的特征工程，但其训练过程难度较高，这是它的主要不足之处。

总结来说，不同的算法在序列标注任务中有其独特的应用场景和优势。支持向量机通过线性分类方法解决简单的标注任务；隐马尔可夫模型在处理隐状态序列时表现出色。而最大熵和最大熵马尔科夫模型在处理复杂上下文特征时更具灵活性，虽然计算复杂度较高；条件随机场则是解决标注偏置问题最为有效的方法，但也面临着较高的训练复杂度。在具体应用中，应根据任务需求选择合适的算法，充分发挥其特长，克服短板，实现最优的序列标注效果。

## 2.0.2 2.1 深度学习概览

维基百科中，深度学习的定义为：深度学习是机器学习的分支，是一种以人工神经网络为架构，对资料进行表征学习的算法。其中“深”是指有多层神经网络，“学习”既指机器学习，也指表征学习，是指从数据中学习表示。

### 2.1.1 “深度学习”为什么叫“深度学习”

某种意义上，深度学习就是多层的神经网络。但现在，“深度学习”这个词似乎远比“神经网络”要火，甚至有和“机器学习”平起平坐之势。为什么要把“深度学习”和“神经网络”“机器学习”区分开呢？

在早期，神经网络一直不温不火，学术界普遍觉得这个新事物不靠谱，神经网络一词被许多学术期刊编辑所排斥，有些稿件的标题甚至因为包含“神经网络”就会被退回。为了不刺激这些人的敏感神经，2006年，Hinton 取了个新名字，将模型命名为“深度信念网络”(Deep Belief Networks)，正式提出了“深度学习”的概念。

深度学习与传统机器学习的不同之处在于，当网络变深后，模型能从数据自动提取特征，如单词的语义表示、句子的语法结构等，不再需要人工提取特征。这也是深度学习和传统机器学习的最主要区别之一。

其次是深度学习对大数据的处理能力。传统机器学习在小规模数据上表现较好（通常是结构化的表格数据），但遇到数量庞大、结构复杂的大数据时，如文本、图像、语音等，传统机器学习就显得捉襟见肘了。以前计算机的计算能力有限，难以处理大规模的计算，训练一个小型的神经网络模型都需要几天。在高性能计算硬件发展起来后（尤其是 GPU），深度学习才展现出优势，在图像识别、自然语言处理、信息检索等大数据领域取得显著突破。

### 2.1.2 发展历史

深度学习的发展历史要从人工神经网络说起，其起源可以追溯到 20 世纪 40 年代，这一进程经历了多次创新与变革。

在 1943 年，McCulloch 和 Pitts 提出了电子脑的概念，这一概念奠定了神经网络的基础。他们提出了使用可调权重的简单神经元模型。

1957 年，Rosenblatt 引入了感知器 (Perceptron) 模型，这是一种能够进行二分类任务的简单神经网络，并且权重是可学习的。感知器的提出标志着神经网络黄金时代的开始。

在 1960 年，Widrow 和 Hoff 提出了自适应线性神经元 (ADALINE) 模型，这个模型也是一种神经网络结构，并引入了可学习的权重和阈值。

1969 年，Minsky 和 Papert 指出了感知器的一个重要缺陷——无法解决 XOR 问题，这一发现导致了人工智能研究领域的第一次低谷，也被称为“人工智能的冬天”。

1986 年，通用多层感知器 (Multi-layered Perceptron, MLP) 模型引入了反向传播算法，这一突破由 Rumelhart、Hinton 和 Williams 提出，解决了传统感知器无法处理非线性可分问题的困境。反向传播算法的出现使得训练深度神经网络成为可能。

1995 年，Vapnik 和 Cortes 提出了支持向量机 (SVM)，这是一种非常有效的监督学习模型，特别适用于小样本数据集。SVM 通过引入核函数，扩展了线性模型的能力，能够处理非线性问题。

2006 年，Hinton 和 Ruslan 提出了深度神经网络的预训练方法，这一方法通过逐层预训练和微调，成功训练了更深层次的神经网络，从此开启了深度学习 (Deep Learning) 的新时代。深度神经网络能够进行层次化特征学习，大幅提升了计算机视觉、语音识别和自然语言处理等领域的性能。

综上所述，人工神经网络的发展经历了从简单神经元模型、感知器、ADALINE、反向传播算法、支持向量机到深度学习预训练等多个关键阶段，每一阶段的创新都为现代深度学习技术的兴起奠定了重要基础。

### 2.1.3 基本网络架构

回望过去，从 2006 年 Hinton 提出“深度学习”的概念开始，深度学习其实才发展了不到 20 年，但已经对人类社会产生了深远的影响。网络结构和模型设计也发展出了各种各样的架构，但大部分模型都由几种基础模型演变而来，下面将一一介绍他们。

全连接神经网络（Fully Connected Neural Network, NN）是一种很基础又常用的神经网络结构。在这种网络里，不同节点之间都是相互连接的，这样的结构让它能够充分利用数据，提取出全局特征，灵活性和表达能力都很强。然而，全连接的特点也增长了计算复杂度，容易出现过拟合问题。

卷积神经网络（Convolutional Neural Network, CNN）则擅长提取局部特征。CNN 通过卷积操作实现特征提取，相较于 NN，它的权重数量少，还能通过参数共享减轻计算负担。不同卷积层大小的 CNN 能够捕获不同尺度的特征，适用于处理图像数据或局部相关性强的文本数据。此外，改良卷积神经网络如 Dilated CNN (D-CNN) 能够扩大感受野，进一步提升特征提取能力。

循环神经网络（Recurrent Neural Network, RNN）特别适合处理序列数据，能够捕获上下文序列特征。长短时记忆网络（Long Short-Term Memory, LSTM）和门控循环单元（Gated Recurrent Unit, GRU）是 RNN 的两种常见变体，它们可以缓解传统 RNN 的梯度消失问题，从而更好地捕捉长距离依赖关系。其他改进版本如全互连 LSTM、更轻量化的双向 RNN (Bi-RNN) 以及改进的注意力机制，也进一步提升了 RNN 的性能。

基于注意力机制的神经网络（Transformer）则通过引入自注意力（Self-Attention）机制，有效处理长距离依赖，并且能够并行化处理，大幅减少了训练时间。BERT (Bidirectional Encoder Representations from Transformers) 是 Transformer 的经典应用，具备强大的特征提取能力，开启了预训练和微调的范式。在此基础上改进的 RoBERTa 具有更优异的性能表现，能够在多种下游任务中取得领先成绩。

图神经网络（Graph Neural Network, GNN）是用于处理非欧几里得空间数据的模型，适用于图结构数据分析。其变体如图卷积网络（GCN）和关系图卷积网络（R-GCN）能够分别处理同构图和异构图的数据，具有很强的表达能力，广泛应用于社交网络分析、推荐系统等领域。

总的来说，这几种神经网络模型各有千秋，全连接神经网络适用于通用场景，卷积神经网络突出局部特征提取，循环神经网络强调序列信息捕捉，基于注意力机制的神经网络在特征提取上表现优异，而图神经网络专注于处理复杂图结构数据。在实际应用中，应根据具体任务和数据的特点选择最合适的模型结构，以实现最佳的效果。

### 2.0.3 2.2 深度前馈网络

#### 2.2.1 整体概览

深度前馈网络（Deep Feedforward Network），也称为多层感知器（Multilayer Perceptron, MLP），是人工神经网络中最基础最常见的一种类型。它由一系列全连接层组成，每层的神经元与下一层的神经元全部连接。数据在网络中单向传递，从输入层经过隐藏层直到输出层，称为前馈网络。深度前馈网络有 3 个重要组成部分：损失函数、反向传播算法和激活函数。

1. **损失函数。**损失函数用于衡量预测结果与实际结果之间的差异，常见的损失函数包括均方误差和交叉熵。这一差异通过反向传播算法用于更新网络的权重，以最小化损失函数。
2. **反向传播算法。**反向传播是训练前馈网络的核心技术。它首先计算输出层的损失，然后逐层反向传递，通过链式法则计算每个权重的梯度。利用这些梯度，使用优化算法如随机梯度下降更新权重，逐步减小损失。
3. **激活函数。**激活函数是前馈网络的关键组件，它引入非线性能力，使网络能够学习和表示复杂的非线性关系。常用的激活函数有 ReLU、tanh 和 sigmoid 等。它们在每一层神经元的计算过程中进行非线性变换。

在基础架构之外，深度前馈网络可以有许多高级结构，如多层次网络、残差网络和生成对抗网络等。这些高级架构可以通过更复杂的网络设计和训练策略增强模型的性能和表达能力。

最后，总结来说，深度前馈网络是深度学习的基础模型，理解其结构和工作原理对学习其他复杂模型至关重要。

#### 2.2.2 从感知器到神经网络

感知器（Perceptron）是有着最简单形式的前馈神经网络，有着悠久的发展历史。在 1957 年，感知器首次由 Frank Rosenblatt 在康奈尔航空实验室发明。早在 20 世纪 40 年代，McCulloch 和 Pitts 就描述了一种类似的神经元模型。

1958 年，Rosenblatt 发表声明称，感知器将发展成为一种能“行走、说话、看见、写作、自我复制并意识到自己存在的”计算机。

20 世纪 60 年代，人们认识到多层感知器相对于单层感知器具有更强的处理能力。这个观点得到了 Minsky 和 Papert 以及 Grossberg 的支持。

1964 年，Aizerman 引入了核感知器，将感知器扩展到核方法，以处理更多复杂的数据集。

1998 年，Freund 和 Schapire 首次提出了在一般不可分情况下的边界保证。

1999 年，投票感知器被提出，这种方法提供了与核支持向量机（SVM）相当的泛化界限。

感知器模型如图所示，通过加权输入和偏置值，经过激活函数生成输出。感知器的这些创新和发展为现代神经网络和机器学习算法奠定了重要基础。

### 2.2.3 单层感知器

单层感知器 (Single-layer Perceptron) 是最基本的神经网络模型之一，其输出通过线性组合输入信号和偏置值，再通过一个激活函数计算得出。具体公式如下：

$$[ y = f(W^T x) = f(\left( \sum_{i=1}^3 W_i x_i + b \right)) ]$$

其中， $[f]$  是激活函数。

常见的激活函数包括：

#### 1. Sigmoid 函数：

$$[ f(z) = \frac{1}{1 + e^{-z}} ]$$

#### 1. 双曲正切函数 (Tanh)：

$$[ \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} ]$$

#### 1. ReLU 函数 (Rectified Linear Unit)：

$$[ f(z) = \max(0, z) ]$$

这些激活函数有助于引入非线性变换，使神经网络能够拟合更复杂的关系。

另外，这些激活函数的导数也非常有用，在反向传播算法中计算梯度时需要用到：

#### 1. Sigmoid 函数的导数：

$$[ f'(z) = f(z) (1 - f(z)) ]$$

#### 1. 双曲正切函数的导数：

$$[ f'(z) = 1 - (\tanh(z))^2 ]$$

#### 1. ReLU 函数的导数：

$$[ f'(z) = \begin{cases} 0, & \text{if } z \leq 0 \\ 1, & \text{if } z > 0 \end{cases} ]$$

单层感知器通过这些激活函数将输入信号转换为输出值，实现简单的分类和回归任务。下侧的图示展示了单层感知器的基本架构，输入信号经过加权和偏置处理，通过激活函数生成输出结果。

### 2.2.4 Softmax 输出单元

Softmax 输出单元定义如下：

$$[\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}]$$

Softmax 单元自然地表示一个离散变量的概率分布，该变量有  $k$  个可能值。

一个线性层预测未归一化的对数概率，其公式为

$$[z = W^T h + b]$$

$$\text{其中, } [z_i = \log \tilde{P}(y=i|x)]$$

这种方法可以视为 sigmoid 函数的推广，后者用于表示二元变量的概率分布。相较于 sigmoid，softmax 能够处理多分类问题。

这可以视为 sigmoid 函数的推广，sigmoid 函数用于表示二元变量的概率分布。

Softmax 函数通常用作分类器的输出。对数似然中的对数项可以抵消 softmax 的指数项，其公式为

$$[\log \text{softmax}(z)_i = z_i - \log \sum_j \exp(z_j)]$$

### 2.2.5 深度前馈网络

深度前馈网络 (Deep Feedforward Networks)，也称为多层感知器 (Multi-layer Perceptron)，是一种基础的神经网络结构。它由多个层组成，包括输入层、一个或多个隐藏层和输出层。此外，网络中还包含偏置单元，以增强模型的表现力。

深度前馈网络的目标是逼近某个映射函数 ( $f^*$ )，例如，将输入 ( $x$ ) 映射到一个类别 ( $y$ )。具体来说，前馈网络通过定义一个映射关系，形式上可以表示为

$$[y = f^*(x; \theta)]$$

其中，参数 ( $\theta$ ) 包括权重 ( $W$ ) 和偏置 ( $b$ )。这些参数通过学习过程进行优化，以实现最佳逼近。

图中的示例展示了一个包含四个隐藏层的深度前馈网络。网络从输入层开始，将输入信号 ( $x_i$ ) 传递到第一隐藏层的神经元 ( $h_1^1, h_2^1, h_3^1$ )，经过多层处理，最终传递到输出层的 ( $y_1, y_2$ )。

前馈网络的训练过程包括前向传播和反向传播。在前向传播过程中，输入信号经过层层传递和处理，最终输出预测结果。在反向传播过程中，根据预测结果和真实标签之间的差异（损失），计算每个参数的梯度，然后通过梯度下降法（或其他优化算法）调整参数，以最小化损失函数。

通过多层结构、非线性激活函数和偏置单元的组合，深度前馈网络具有强大的表达能力，可以拟合复杂的非线性关系，广泛应用于分类、回归和其他任务。

**2.2.5.1 形式化计算** 深度前馈网络 (Deep Feedforward Networks) 是一种基础且常见的神经网络架构。它由多个层组成，包括输入层 (Layer ( $L_1$ ))、一个或多个隐藏层 (Layer ( $L_2, L_3, \dots$ )) 和输出层 (Layer ( $L_5$ )))。每一层的神经元与下一层的神经元全连接，从输入层向输出层单向传递信息，没有反馈连接。

网络中的输入层接受原始输入信号，例如 ( $x_1, x_2, x_3$ )。这些信号经过加权求和和加上偏置后，传递到第一个隐藏层。公式如下：

$$[ z^2 = W^1 \times x + b^1 ]$$

其中 ( $z^2$ ) 是传递到隐藏层 ( $h^2$ ) 的线性组合结果，( $W^1$ ) 是输入到第一个隐藏层的权重矩阵，( $b^1$ ) 是偏置向量。

经过激活函数的非线性变换后，得到第一个隐藏层的输出：

$$[ h^2 = f(z^2) ]$$

激活函数 ( $f$ ) 的常见选择包括 ReLU、sigmoid 或 tanh。

这一过程在随后的隐藏层中重复进行，例如：

$$[ z^{l+1} = W^l \times h^l + b^l ]$$

$$[ h^{l+1} = f(z^{l+1}) ]$$

最终，最后一个隐藏层的输出传递到输出层，经过最后一次线性变换和激活函数处理，得到输出：

$$[ y = f(z^L) ]$$

其中 ( $z^L$ ) 是最后一个隐藏层的输出经过线性变换后的结果。

深度前馈网络通过多个隐藏层和非线性激活函数进行复杂的特征提取和模式识别，广泛应用于分类、回归和其他任务。每一层的权重和偏置参数通过训练过程进行学习和优化，以最小化预测结果与真实结果之间的误差，通常使用反向传播算法来调整权重和偏置。

## 2.2.6 示例：学习异或（XOR）

让我们通过一个经典例子——学习 XOR 问题，来说明深度前馈网络的工作原理。XOR 问题是一个著名的非线性分类任务，用简单的线性模型无法解决，而深度前馈网络可以通过隐藏层和非线性激活函数完成这一任务。

### 2.2.6.1 扩展线性模型 线性函数 ( $\phi(x) = W^T x$ )

在某些情况下无法有效工作，因为它们的表达能力有限。当处理复杂的数据模式和非线性关系时，我们需要在模型中引入非线性元素。

为了扩展线性模型，我们可以使用非线性变换 ( $\phi(x)$ )，这种方法可以捕捉数据中的非线性特征。通过这种方式，输入信号 ( $x$ ) 经过非线性变换 ( $\phi(x)$ ) 后，再进行线性组合。

公式如下：

$$[ y = w^T \phi(x) ]$$

图中展示了这一过程：首先，将输入 ( $x$ ) 传递到一个包含非线性激活函数的隐藏层，经过这些隐藏层的处理，得到非线性变换后的特征 ( $\phi(x)$ )。最后，将这些特征输入到线性模型中，得到最终的输出 ( $y$ )。

这种方法利用了深度神经网络中的隐藏层和非线性激活函数，使得模型可以更好地拟合复杂的数据模式和非线性关系，从而提高模型的表现力和预测能力。

为了表示 ( $x$ ) 的非线性函数，可以使用非线性函数 ( $\phi(x)$ ) 来转换 ( $x$ )。类似的“核”技巧也能获得非线性效果。

核技巧是一种通过在高维空间中处理数据来获得非线性结果的方法。这就意味着在不显式计算高维映射的情况下，引入非线性。

支持向量机 (SVM) 中：

$$[ f(x) = W^T x + b \rightarrow f(x) = b + \sum_i \alpha_i \phi(x) \phi(x^i) ]$$

通过核技巧，我们将 SVM 问题映射到高维空间，通过内积计算得到决策函数  $f(x)$ 。其中， $i$  是拉格朗日乘数， $(x)$  是输入向量  $x$  的高维映射。

其中，为了计算  $(x)$  和  $(x^i)$  的内积，我们引入了核函数  $K(x, x^i)$

$$(\phi(x) \phi(x^i)) = K(x, x^i).$$

那么如何选择映射 ( $\phi$ ) 呢？

使用非常通用的 ( $\phi$ )，例如无限维的 ( $\phi$ )，但这种方法的泛化能力较差。

手动设计 ( $\phi$ )，但在不同领域之间的迁移性较差。

深度学习的策略是学习 ( $\phi$ )。我们将表示参数化为 ( $\phi(x; \theta)$ )  
，并使用优化算法找到 ( $\theta$ )。

这样一来，无需显式计算高维空间的映射，只需使用核函数直接计算内积。常见的核函数包括：

线性核  $K(x, x_i) = x^T x_i$

多项式核  $K(x, x_i) = (x^T x_i + 1)^d$

高斯核 (RBF 核)  $K(x, x_i) = \exp(-\|x - x_i\|^2)$

这种策略结合了前两种方法的优势，使其具有高度的通用性——通过使用非常广泛的函数族 ( $\phi(x; \theta)$ )。人类设计者只需找到合适的一般函数族，而不需要精确找到特定函数。

### 2.2.6.2 线性模型不适用 在学习异或 (XOR) 例子中，我们关注的不是统计泛化，而是对四个特定点的正确分类。

这些点是 ( $\mathbf{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$ )。

我们的唯一挑战是拟合这个训练集。

异或 (XOR) 函数提供了我们要学习的目标函数 ( $y = f(x)$ )。我们的模型提供了一个函数 ( $y = f(x; \theta)$ )，然后我们的学习算法将调整参数 ( $\theta$ )，使得 ( $f$ ) 尽可能接近 ( $f^*$ )。

我们将这个问题视为回归问题，并使用均方误差 (MSE) 进行处理。均方误差损失函数的定义如下：

$$[ J(\theta) = \frac{1}{4} \sum_{x \in \mathbf{X}} (f^*(x) - f(x; \theta))^2 ]$$

通常，均方误差不用于二元数据，但在这里数学处理较为简单。我们选择一个线性模型的形式：

$$[ f(x; \mathbf{w}, b) = \mathbf{x}^T \mathbf{w} + b ]$$

通过最小化 ( $J(\theta)$ )，我们得到 ( $\mathbf{w} = \frac{1}{2}$ ) 和 ( $b = \frac{1}{2}$ )。因此，线性模型在每个输入点上输出的值都是 ( $\frac{1}{2}$ )。

解决 XOR 问题需要通过学习表示来完成。当 ( $x_1 = 0$ ) 时，输出必须随 ( $x_2$ ) 增加；当 ( $x_1 = 1$ ) 时，输出必须随 ( $x_2$ ) 减少。线性模型 ( $f(x; \mathbf{w}, b) = x_1 w_1 + x_2 w_2 + b$ ) 必须给 ( $x_2$ ) 分配一个唯一的权重 ( $w_2$ )，因此无法解决这个问题。

一个更好的解决方案是使用一个简单的前馈网络。这个网络包含一层隐藏层，其中有两个隐藏单元。非线性特征将输入 ( $x = [1, 0]^T$ ) 和 ( $x = [0, 1]^T$ ) 都映射到特征空间中的一个单点 ( $h$

$= [1, 0]^T$ )。这样，一个线性模型现在能够解决这个问题。

### 2.2.6.3 前馈网络解决 XOR 问题 第一层（隐藏层）：

隐藏层的输出通过以下公式计算：

$$[ h_i = f(x^T W_{\{\cdot, i\}} + c_i) ]$$

其中，( $c$ )是偏置变量。通常选择( $g$ )作为逐元素激活函数。默认的激活函数是修正线性单元(ReLU)，定义为：

$$[ g(z) = \max\{0, z\} ]$$

完整的网络：

整个网络的输出通过以下公式计算：

$$[ f(x; W, c, w, b) = w^T f(W^T x + c) + b ]$$

这个公式表示输入( $x$ )经由权重矩阵( $W$ )和偏置( $c$ )的线性组合，再通过非线性激活函数( $f$ )进行变换，最终得到输出( $y$ )。完整的网络模型包括输入层到隐藏层，再到输出层的所有信息传递和计算过程。

前馈网络解决 XOR 问题

设权重矩阵( $W$ )为：

$$[ W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} ]$$

偏置向量( $c$ )为：

$$[ c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} ]$$

输出层权重向量( $w$ )为：

$$[ w = \begin{bmatrix} 1 \\ -2 \end{bmatrix} ]$$

偏置值( $b = 0$ )。我们将得到所有点的正确答案。

现在我们一步步解析模型的处理过程：

1. 设计矩阵( $X$ )包含四个点：

$$[ X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} ]$$

1. 计算( $XW$ )：

$$[ XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} ]$$

\end{bmatrix} ]

1. 加上偏置 ( c ):

```
[ XW + c = \begin{bmatrix}
0 & -1 \\
1 & 0 \\
1 & 0 \\
2 & 1
\end{bmatrix} ]
```

1. 应用 ReLU 激活函数:

```
[ g(XW + c) = \begin{bmatrix}
0 & 0 \\
1 & 0 \\
1 & 0 \\
2 & 1
\end{bmatrix} ]
```

1. 乘以权重向量 ( w ):

```
[ w g(XW + c) + b = \begin{bmatrix}
0 \times 1 + 0 \times (-2) \\
1 \times 1 + 0 \times (-2) \\
1 \times 1 + 0 \times (-2) \\
2 \times 1 + 1 \times (-2)
\end{bmatrix}
= \begin{bmatrix}
0 \\
1 \\
1 \\
0
\end{bmatrix} ]
```

通过这个过程，我们成功地用前馈网络正确地解决了 XOR 问题。

前馈网络解决 XOR 问题

我们通过定义如下参数简单地指定了解决方案，从而实现了零误差:

```
[ W = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} ]
[ c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} ]
[ w = \begin{bmatrix} 1 \\ -2 \end{bmatrix} ]
```

[  $b = 0$  ]

在实际情况中，可能会有数十亿个参数和数十亿个训练样本，因此无法简单地猜测出解决方案。相反，使用梯度下降优化可以找到具有很小误差的参数集。

梯度下降法是一种迭代优化算法，用于找到函数的极小值或极大值。其目标是最小化目标函数（通常是损失函数），并通过计算该函数的梯度来指导参数更新。算法从初始参数值开始，每一步都沿着梯度的反方向移动参数，调整的步长由学习率确定。根据使用的数据量，梯度下降法分为批量梯度下降、随机梯度下降和小批量梯度下降。选择合适的学习率和优化技巧，如动态学习率和动量，可以加快收敛速度并提高算法的稳定性。梯度下降法广泛应用于机器学习和深度学习中，用于优化模型参数，使其在训练数据上的表现更佳。

梯度下降法可以类比为在山坡上行走，目的是找到山谷的最低点。梯度表示当前所在位置坡度的方向和陡峭程度，每一步按照负梯度的方向前进一定步长（学习率），逐步逼近最低点。

梯度下降法的核心思想是：

1. 目标函数（损失函数）：我们希望最小化的函数，通常表示为  $J(\theta)$ ，其中  $\theta$  是参数向量。
2. 梯度：目标函数  $J(\theta)$  对参数  $\theta$  的偏导数，表示为  $\nabla J(\theta)$ 。梯度指向函数上升最快的方向，而负梯度则指向下降最快的方向。
3. 学习率（步长）：一个标量，表示每次迭代中参数更新的步长，通常表示为  $\alpha$ 。

梯度下降法的具体步骤如下：

1. 初始化：初始化参数  $\theta$ （常用随机值或特定初始化方法）。
2. 计算梯度：计算损失函数  $J(\theta)$  对参数  $\theta$  的梯度  $\nabla J(\theta)$ 。
3. 更新参数：按梯度下降的方向更新参数：

$$\theta := \theta - \alpha \nabla J(\theta)$$

其中， $\alpha$  是学习率。

迭代：重复步骤 2 和 3，直到损失函数收敛到最小值（或最大值），即梯度接近零，或者达到预先设定的迭代次数。

## 2.2.7 损失函数

假设我们有一组训练数据  $\{(x^1, y^1), \dots, (x^n, y^n)\}$ ，

共有  $n$  个训练样本。为了学习我们的参数  $(\theta = (W, b))$ ，我们可以定义以下损失函数：

$$[ \text{mathcal}{J}(W, b; x, y) = \frac{1}{2} \| \hat{y} - y \|^2 ]$$

其中， $(\hat{y})$  是学到的神经网络预测的值。

损失函数衡量了预测输出与真实值之间的差异。

### 2.2.7.1 训练损失 我们定义用于训练的数据集的损失函数为：

$$[\mathcal{J}(W, b; \mathbf{x}_{\text{train}}, y_{\text{train}}) = \frac{1}{2} \|\hat{y} - y_{\text{train}}\|^2]$$

**训练损失：**模型的预测应与训练数据相匹配。训练损失用于指导模型优化的方向。

图中的蓝色点表示训练数据。在训练损失为零的情况下，模型完美地拟合了所有训练数据点。通过最小化损失函数，模型的参数不断调整，以提高预测的准确性。

### 2.2.7.2 验证损失 我们定义用于验证数据集的损失函数为：

$$[\mathcal{J}(W, b; \mathbf{x}_{\text{valid}}, y_{\text{valid}}) = \frac{1}{2} \|\hat{y} - y_{\text{valid}}\|^2]$$

**验证损失：**用于测试模型在未见过的数据上的泛化能力。

图中绿色方块表示验证数据。验证损失较大意味着模型在验证数据上的表现不佳，说明存在过拟合 (Over-fitting)。过拟合问题表现在模型可以很好地拟合训练数据，但在未见过的数据上表现较差。

### 2.2.7.3 正则项 为了避免过拟合，我们可以在损失函数中添加一个正则项。正则化后的损失函数定义如下：

$$[\mathcal{J}(W, b) = \frac{1}{n} \left[ \sum_{i=1}^n \frac{1}{2} \|\hat{y}_i - y_i\|^2 + \frac{\lambda}{L} \sum_{l=1}^L \sum_{j=1}^{s_l} (W_{j|i})^2 \right]]$$

模型应当保持“简单”，以便在测试数据上表现良好。需要注意的是，我们通常不会对偏置项 ( $b$ ) 进行正则化，因为它影响不大。

图中绿色的线表示经过正则化后的模型。正则化有助于抑制模型的过拟合现象，使模型在训练数据和验证数据上都能表现得比较平稳，不会过度拟合训练数据的噪声，从而提升模型的泛化能力。

### 2.2.7.4 损失函数类型 在机器学习中，损失函数用于衡量模型预测输出与真实值之间的差异。不同类型的任务使用不同的损失函数。

**回归任务：**常用的损失函数包括均方误差 (Mean Squared Error, MSE) 和平均绝对误差 (Mean Absolute Error, MAE)。

**分类任务：**常用的损失函数是交叉熵 (Cross-Entropy)。

**其他：**例如支持向量机 (SVM) 中使用的 Hinge Loss。

常见的损失函数及其公式如下：

1. 均方误差 (Mean Squared Error, MSE) :

$$[\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2]$$

1. 平均绝对误差 (Mean Absolute Error, MAE) :

$$[\frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i|]$$

1. 类别交叉熵 (Categorical Cross-Entropy) :

$$[ -\sum_{i=1}^n y_i \log(\hat{y}_i) ]$$

1. Hinge Loss (用于 SVM) :

$$[ \max(0, 1 - y \cdot \hat{y}) ]$$

不同的损失函数适用于不同的应用场景，根据具体任务和数据特点选择合适的损失函数，有助于提升模型的性能和效果。

## 2.2.8 反向传播算法

### 2.2.8.1 模型学习

我们的目标是最小化损失函数 ( $\mathcal{J}(W, b)$ )

，这是关于参数 ( $W$ ) 和 ( $b$ ) 的函数。

首先，我们将每个参数 ( $W_{ij}^{(l)}$ ) 和 ( $b_i^{(l)}$ ) 初始化为接近零的小随机值，例如按照均值为零、方差为 ( $\epsilon^2$ )

的正态分布来随机取值。全零初始化效果很差，因为它会导致所有单元的输出相同，无法学到有效的特征。

接下来，我们应用优化算法，例如批量梯度下降，来调整参数以最小化损失函数。

由于 ( $\mathcal{J}(W, b)$ )

是一个非凸函数，梯度下降容易陷入局部最优解。然而，实际应用中，梯度下降通常能够很好地工作。通过不断更新参数，模型在训练数据上的表现将不断提升，最终达到较好的效果。

一次梯度下降迭代更新参数 ( $W$ ) 和 ( $b$ )，公式如下：

$$[ W_{ij}^{(l)} = W_{ij}^{(l)} + \alpha \frac{\partial \mathcal{J}(W, b)}{\partial W_{ij}^{(l)}} ]$$

$$[ b_i^{(l)} = b_i^{(l)} + \alpha \frac{\partial \mathcal{J}(W, b)}{\partial b_i^{(l)}} ]$$

其中 ( $\alpha$ ) 是学习率。

关键步骤是计算上述偏导数。接下来我们描述反向传播算法，它提供了一种高效的计算这些偏导数的方法。通过反向传播算法，神经网络能够快速地调整权重和偏置，从而最小化损失函数，提高模型性能。

### 2.2.8.2 反向传播

前向传播：输入 ( $x$ ) 提供初始信息，这些信息传递到每一层的隐藏单元，最后产生预测值 ( $\hat{y}$ )。

反向传播：简称 backprop，它允许从损失函数中传递的信息反向流经网络，以计算梯度。通过梯度计算，网络的权重和偏置可以进行调整，从而最小化损失函数，提高模型的准确性。

#### 2.2.8.2.1 训练

梯度下降的一次迭代通过如下方式更新参数 ( $W, b$ )：

$$[ W_{ij}^{(l)} = W_{ij}^{(l)} + \alpha \frac{\partial \mathcal{J}(W, b)}{\partial W_{ij}^{(l)}} ]$$

$$[ b_i^{(l)} = b_i^{(l)} + \alpha \frac{\partial \mathcal{J}(W, b)}{\partial b_i^{(l)}} ]$$

其中 ( $\alpha$ ) 是学习率。

偏导数的计算如下：

$$\begin{aligned} \frac{\partial}{\partial W_{ij}} \hat{W}_{ij}(l) &= \left[ \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial W_{ij}} \hat{W}_{ij}(l) \right] \mathcal{J}(W, b) + \lambda W_{ij}(l) \\ \frac{\partial}{\partial b_i} \hat{b}_i(l) &= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial b_i} \hat{b}_i(l) \mathcal{J}(W, b; x, y) \end{aligned}$$

现在，我们的问题变成了如何通过反向传播算法计算：

$$\begin{aligned} &\left[ \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial W_{ij}} \hat{W}_{ij}(l) \mathcal{J}(W, b; x, y) \right] \end{aligned}$$

反向传播算法提供了一种高效的方法来计算这些偏导数，以更新网络参数并最小化损失函数。

偏导数的具体计算

对于第 (1) 层的权重 ( $W_{ij}(l)$ ) 和偏置 ( $b_i(l)$ )，其偏导数如下：

$$\begin{aligned} \frac{\partial}{\partial W_{ij}} \hat{W}_{ij}(l) &= \left[ \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial W_{ij}} \hat{W}_{ij}(l) \right] \mathcal{J}(W, b; x, y) + \lambda W_{ij}(l) \\ \text{其中,} \end{aligned}$$

1. ( $\frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial W_{ij}} \hat{W}_{ij}(l) \mathcal{J}(W, b; x, y)$ ) 是对每个训练样本计算得来的平均导数。

1. ( $\lambda W_{ij}(l)$ ) 是正则化项的导数，这部分用于防止过拟合，确保模型的复杂度不超过。

类似地，对于偏置 ( $b_i(l)$ )，其偏导数为：

$$\left[ \frac{\partial}{\partial b_i} \hat{b}_i(l) \mathcal{J}(W, b; x, y) \right]$$

这个公式表示对每个训练样本计算偏置的梯度，然后取平均值。

**2.2.8.2.2 直观理解** 给定一个训练样本 (( $x, y$ ))，我们首先进行一次“前向传播”来计算网络中所有节点的激活值，包括假设的输出值 ( $\hat{y}$ )。

然后，对于第 (1) 层中的每个节点 ( $i$ )，我们需要计算一个“误差项” ( $\delta_i(l)$ )，用于衡量该节点对输出错误的“责任”有多大。

对于输出层的节点，我们可以直接测量网络的激活值与真实目标值之间的差异，并使用这个差异来定义 ( $\delta_i(L)$ )。

对于隐藏层的单元，我们根据使用 ( $z_i(l)$ ) 作为输入的节点的误差项的加权平均值来计算 ( $\delta_i(l)$ )。通过这种方式，可以逐层向后传播误差，以此来调整每个层的权重，从而最小化整个网络的误差。

**2.2.8.3 前向传播** 给定一个训练样本 (( $x, y$ ))，我们首先进行一次“前向传播”来计算网络中所有节点的激活值，包括假设的输出值 ( $\hat{y}$ )。

前向传播的具体步骤如下：

对于第一个隐藏层的节点：

$$\begin{aligned} z_1^{(2)} &= f(\left( W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)} \right)) \\ z_2^{(2)} &= f(\left( W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)} \right)) \\ z_3^{(2)} &= f(\left( W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)} \right)) \end{aligned}$$

对于输出层的节点：

$$\hat{y} = z_1^{(3)} = f(\left( W_{11}^{(2)} z_1^{(2)} + W_{12}^{(2)} z_2^{(2)} + W_{13}^{(2)} z_3^{(2)} + b^{(2)} \right))$$

通过前向传播，我们计算每一层的节点激活值，再通过这些激活值计算网络的输出值 ( $\hat{y}$ )。

**2.2.8.4 输出层的误差** 当一个训练样本  $((x, y))$  被输入到网络中，我们首先计算前向传播的结果，即计算输出值 ( $\hat{y}$ )。

对于每个第  $(l)$  层的节点  $(i)$ ，我们需要计算一个“误差项” ( $\delta_i^{(l)}$ ) 来衡量该节点对于输出误差的“责任”有多大。

对于输出层的节点，我们可以直接测量网络的激活值与真实目标值之间的差异，并使用这个差异来定义 ( $\delta_i^{(L)}$ )。

具体来说，输出层的误差项 ( $\delta_i^{(L)}$ ) 的计算公式如下：

$$\delta_i^{(L)} = \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} |y - \hat{y}|^2 = (y_i - f(z_i^{(L)})) \cdot f'(z_i^{(L)})$$

这个公式体现了输出节点的误差项是网络激活值与实际值之差，再乘以激活函数的导数。通过计算这些误差项，网络能够逐步调整参数，以减少整体误差。

**2.2.8.5 误差的反向传播** 对于输出层的节点，我们可以直接测量网络的激活值与真实目标值之间的差异，并使用这个差异来定义 ( $\delta_i^{(L)}$ )。

对于隐藏单元，我们将基于使用 ( $z_i^{(l)}$ ) 作为输入的节点的误差项的加权平均值来计算 ( $\delta_i^{(l)}$ )。

具体来说，对于  $(l = L-1, L-2, \dots, 2)$  和第  $(l)$  层中的每个节点  $(i)$ ，其误差项的计算公式是：

$$\delta_i^{(l)} = \left( \sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

这个公式表示，第  $(l)$  层每个节点的误差项是来自下一个层（即第  $(l+1)$  层）节点误差项的加权和，再乘以下一层节点的激活函数导数。通过这种方式，我们逐层向后传播误差，逐渐更新每一层的权重，从而优化整个网络。

对于隐藏单元，我们基于使用 ( $z_i^{(l)}$ ) 作为输入的节点的误差项的加权平均值来计算 ( $\delta_i^{(l)}$ )。对于 ( $l = L-1, L-2, \dots, 2$ ) 和第 ( $l$ ) 层中的每个节点 ( $i$ )，其误差项的计算公式是：

$$[\delta_i^{(l)} = \left( \sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} f'(z_i^{(l)}) \right)]$$

回顾链式法则：如果 ( $y = g(x)$ ) 并且 ( $z = f(y)$ )，则：  
 $y=g(x)$  并且  $z=f(y)$

$$[\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}]$$

利用链式法则，我们可以将误差从输出层逐层传播到输入层。这一步骤允许我们计算每个隐藏层神经元的误差，实现对每层权重和偏置的优化调整。

#### 2.2.8.6 求导 计算所需的偏导数，如下所示：

$$[\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = z_j^{(l)} \delta_i^{(l+1)}]$$

$$[\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}]$$

公式展示了反向传播算法中损失函数 ( $J$ ) 对权重 ( $W$ ) 和偏置 ( $b$ ) 的偏导数。通过这些偏导数，可以更新神经网络的参数，使得损失函数逐步减小，从而提高模型的预测准确性。具体来看：

##### 1. 损失函数对权重的偏导数

公式：

$$[\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = z_j^{(l)} \delta_i^{(l+1)}]$$

解释：

$(\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y))$  表示损失函数 ( $J$ ) 对第 ( $l$ ) 层神经网络中从第 ( $j$ ) 个神经元到第 ( $i$ ) 个神经元的权重 ( $W_{ij}^{(l)}$ ) 的偏导数。

$(z_j^{(l)})$  是第 ( $l$ ) 层中第 ( $j$ ) 个神经元的输出。

$(\delta_i^{(l+1)})$  是下一层（第 ( $l+1$ ) 层）中第 ( $i$ ) 个神经元的误差项。

这个公式表示，损失函数对权重的偏导数等于当前层神经元的输出与下一层神经元的误差项的乘积。从直觉上看，这个结果表示当前层的输出对下一个层的误差的重要性，它说明了如果当前层的输出影响了下一层，那该如何调整当前层的权重。

##### 1. 损失函数对偏置的偏导数

公式：

$$[\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}]$$

解释：

$(\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y))$  表示损失函数 ( $J$ ) 对第 ( $l$ ) 层神经网络中第 ( $i$ ) 个神经元的偏置 ( $b_i^{(l)}$ ) 的偏导数。

$(\delta_i^{(l+1)})$  是下一层（第 ( $l+1$ ) 层）中第 ( $i$ ) 个神经元的误差项。

这个公式表示，损失函数对某一层偏置的偏导数等于下一层中相应神经元的误差项。由于偏置 (b) 的偏导数不取决于特定的激活值，它只和误差项有关，这表示该偏置对于误差的贡献。

#### 反向传播深度解释

在训练神经网络时，我们通过最小化某个损失函数 ( $J$ ) 来优化模型的参数 ( $W$ ) 和 ( $b$ )。反向传播算法计算损失函数对每一个参数的偏导数，这些偏导数表示误差如何随每个参数的变化而变化。通过这些偏导数，使用梯度下降或者其他优化算法，逐步更新参数以减少损失。

具体步骤如下：

1. 前向传播：计算每一层的激活值，最终计算损失。
2. 反向传播：从输出层开始，逐层向后计算每一层的误差项。
3. 计算梯度：根据误差项计算损失对权重和偏置的偏导数。
4. 参数更新：使用计算得到的梯度，按梯度下降法等优化算法更新权重和偏置。

所以，这两个公式是反向传播过程中关键的步骤。在反向传播时，利用这些公式更新每一层的权重和偏置，使得下一次前向传播时模型的预测更精确，逐步优化模型。

将其代入：

$$\begin{aligned} \frac{\partial}{\partial W_{ij}} \hat{J}(l) &= \left[ \frac{1}{n} \sum_{i=1}^n \hat{J}(l) \right] \frac{\partial}{\partial W_{ij}} J(W, b; x, y) + \lambda W_{ij} \hat{J}(l) \\ \frac{\partial}{\partial b_i} \hat{J}(l) &= \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial b_i} J(W, b; x, y) \end{aligned}$$

这些公式表示我们可以通过计算每个训练样本的梯度，然后取平均值，并结合正则化项来更新神经网络的参数。这个过程在反向传播中高效地进行，使得神经网络能够在训练过程中持续优化。

#### 2.2.8.7 总结（伪代码）

反复执行以下步骤：

1. 初始化 ( $\Delta W$ ) 和 ( $\Delta b$ ) 为 0 对于所有层 ( $l$ )：

$\Delta W$  和  $\Delta b$

1. 对于 ( $i = 1$ ) 到 ( $n$ )：

执行前向传播；

使用反向传播计算每层的误差；

计算损失函数 ( $J(W, b; x, y)$ ) 相对于权重 ( $\nabla_w J(W, b; x, y)$ ) 和偏置 ( $\nabla_b J(W, b; x, y)$ ) 的偏导数；

(计算损失函数  $J(W, b; x, y)$  相对于权重 ( $w J(W, b; x, y)$ ) 和偏置 ( $b J(W, b; x, y)$ ) 的偏导数)

更新累积梯度：

[

$\Delta W := \Delta W + \nabla_w J(W, b; x, y)$

```

]
[
\Delta b' := \Delta b' + \nabla_b \mathcal{J}(W, b; x, y)
]

```

1. 更新参数：

```

[
W' = W' \alpha \left[ \left( \frac{1}{n} \Delta W' \right) + \lambda W' \right]
]
[
b' = b' \alpha \left( \frac{1}{n} \Delta b' \right)
]
```

重复以上步骤直到收敛为止。这一过程不断调整网络参数，以最小化损失函数，从而提高模型性能。

### 2.2.8.8 问题：反向传播的常见局限性是什么？ A. 局部极小值问题

- B. 收敛速度慢
- C. 对噪声数据敏感
- D. 以上全部

反向传播算法在神经网络训练中的确面临一些局限性。局部极小值问题使得算法可能会陷入非全局最优解，导致模型性能不佳。收敛速度慢会使训练时间过长，特别是在大型数据集和复杂模型中。此外，反向传播对噪声数据非常敏感，噪声数据会影响模型的准确性和稳定性。综合来看，这些限制在一定程度上会影响反向传播算法的效率和效果。正确答案是 D. 以上全部。

### 2.2.9 激活函数

**2.2.9.1 为什么我们需要激活函数？** 我们使用激活函数来模拟神经元的激活和非激活两种状态。这些函数将输入信号转换为输出信号，使神经网络能够捕捉复杂的非线性关系。

举例来说，在生物神经元中，信号通过树突传递到细胞体，然后通过轴突传输出去。类似地，在人工神经网络中，输入信号经过加权求和后传递到激活函数，再将输出信号传递到下一层神经元。

激活函数可以是非线性的，例如 ReLU、sigmoid 或 tanh。非线性激活函数使得网络能够更好地表示复杂的数据模式和特征，增强网络的表达能力。

总之，激活函数在神经网络中起着关键作用，模拟生物神经元的行为，并通过非线性变换提升网络的表现力。

### 2.2.9.2 激活函数 隐藏单元的激活函数设计是一个极其活跃的研究领域。常见的激活函数包括：

- 逻辑 Sigmoid 函数 (Logistic Sigmoid)
- 双曲正切函数 (Hyperbolic Tangent)

修正线性单元 (Rectified Linear Units, ReLU)

广义修正线性单元 (Generalized Rectified Linear Units, GELU)

门控线性单元 (Gated Linear Units, GLU)

Maxout 单元 (Maxout Units)

这些激活函数通过不同的方式处理输入信号，赋予神经网络以非线性变换的能力，从而提高网络的表达能力和学习复杂模式的能力。

#### 2.2.9.2.1 逻辑 Sigmoid 函数 ( $g(z) = \sigma(z)$ )。 $g(z) = \sigma(z)$ 它也用于输出单元。

逻辑 Sigmoid 激活函数，Sigmoid 单元在其大部分区域内都会饱和，这可能使基于梯度的学习变得非常困难。

循环网络、许多概率模型和一些自动编码器有其他要求，这些要求排除了分段线性激活函数的使用，尽管饱和存在缺点，但使 Sigmoid 单元更具吸引力。

#### 2.2.9.2.2 双曲正切函数 ( $g(z) = \tanh(z) = 2\sigma(2z) - 1$ )。

$g(z) = \tanh(z) = 2(2z) - 1$ 。

双曲正切激活函数通常比逻辑 Sigmoid 函数表现更好。

因为 ( $\tanh$ ) 在接近 0 时类似于恒等函数，模型 ( $\hat{y} = w^T \tanh(U^T \tanh(V^T x))$ )

类似于训练一个线性模型 ( $\hat{y} = w^T U^T V^T x$ )

，只要网络激活保持较小。这使得训练 ( $\tanh$ ) 网络更加容易。

#### 2.2.9.2.3 修正线性单元 ReLU

修正线性单元是隐层单元的一个优秀的默认选择(Nair&Hinton, 2010)。

激活函数 ReLU (修正线性单元):  $ReLU(z) = \max\{z, 0\}$ 。

修正线性函数的二阶导数几乎处处为 0，而一阶导数为 1。

因此，梯度方向比引入二阶效应的激活函数更有利于学习。修正线性单元及其推广基于一个原则：如果模型的行为更接近线性，优化将更容易。

#### 2.2.9.2.4 广义修正线性单元

ReLU 的一个缺点是，当它们的激活值为零时，无法通过基于梯度的方法进行学习。

ReLU 的推广形式  $gReLU(z) = \max\{z, 0\} + \min\{z, 0\}$

Leaky-ReLU( $z$ ) =  $\max\{z, 0\} + 0.01 \min\{z, 0\}$

Parametric-ReLU( $z$ ): 参数 是可学习的

它用于图像的目标识别，当输入光照极性反转时，寻求不变特征是有意义的。

#### 2.2.9.2.5 门控线性单元 (GLU)

输入被分成两部分：一部分进行线性变换，另一部分通过门控机制。

$GLU(x, W, V, b, c) = (xW + b) \cdot (xV + c)$

这种机制允许模型动态调节信息的流动。

Sigmoid 可以被 GELU 或 Swish 函数替代。

$$\text{GeGLU}(x, W, V, b, c) = \text{GELU}(xW + b) \cdot (xV + c)$$

$$\text{SwiGLU}(x, W, V, b, c) = \text{Swish}(xW + b) \cdot (xV + c)$$

**2.2.9.2.6 Maxout 单元** Maxout 单元进一步推广了修正线性单元。与其应用逐元素的函数 ( $g(z)$ )，Maxout 单元将 ( $z$ ) 划分为 ( $k$ ) 个值的组。这些组之一为：

$$(g(z))_i = \max\{j \in G^{\hat{(i)}} \mid z_j\}$$

其中 ( $G^{\hat{(i)}}$ ) 是组 ( $i$ ) 的输入索引集合，即 ( $\{(i-1)k+1, \dots, ik\}$ )。

$$\{(i-1)k+1, \dots, ik\}$$

Maxout 单元可以学习一个最多由 ( $k$ ) 段组成的分段线性凸函数。因此，Maxout 单元可视为学习激活函数本身而不仅仅是单元之间的关系。

Maxout 单元

在卷积网络中，通过对 ( $k$ ) 个仿射特征图进行求最大值（即跨通道池化）构建一个 Maxout 特征图。

一个单一的 Maxout 单元可以逼近任意凸函数。图中展示了 Maxout 在一维输入下的表现。

### 2.2.9.3 问题：DNN 和逻辑回归之间的关系是什么？ 提示：考虑 Sigmoid 激活函数。

逻辑回归是神经网络的一种特殊情况。当以下条件满足时，一个 DNN 等价于逻辑回归：

仅前馈

无隐藏层

Sigmoid/Softmax 激活

交叉熵损失

**2.2.9.4 架构考虑** 图 6.6：实验证据表明，当用于从地址照片中转录多位数字时，较深的网络具有更好的泛化能力。数据来自 Goodfellow 等人（2014 年）。随着深度的增加，测试集的准确率稳定提升。参见图 6.7 以了解一个对照实验，该实验表明模型规模的其他增加并不产生相同的效果。

架构考虑

浅层模型在大约 2000 万个参数时出现过拟合，而深层模型在拥有超过 6000 万个参数时仍能受益。

这表达了一种信念，即函数应该由许多简单函数组合而成。

这可能会导致学习到由简单表示组合而成的表示（例如，用边缘定义的角点）。

或者学习到一个具有顺序依赖步骤的程序（例如，首先定位一组对象，然后将它们相互分割，然后识别它们）。

**2.2.9.5 训练不稳定性** 随着模型层数加深，在反向传播过程中梯度可能呈指数级增长，这会导致训练不稳定并导致模型崩溃。

### 2.2.9.6 稳定的隐藏状态：归一化

归一化是将神经网络中输入数据或中间激活值调整到标准范围的过程。

它通过减少内部协变量偏移，帮助稳定和加速训练过程。

公式：

$$(\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}})$$

归一化方法

### 2.2.10 高级架构

Perceptron (1958 年)：由 Frank Rosenblatt 发明。

Hopfield Network (1982 年)：提出了递归和反馈机制。

Neocognitron (1980 年)：引入卷积和池化操作。

BPNN/MLP (1986 年)：建立在多层感知器 (MLP) 的基础上。

RNN/LSTM (1997 年)：用于处理序列数据，特别是语音识别 (2013 年)。

Neural Probabilistic Language Model (2003 年)：在语言模型中取得突破。

LeNet/CNN (1998 年)：Yann Lecun 开发的卷积神经网络 (CNN)。

Deep Q-learning (2013 年)：在强化学习中表现出色。

AutoEncoder (1989/2006 年)：利用编码器和解码器进行无监督学习。Denoising Autoencoder (2008 年) 是其扩展。

word2Vec (2013 年)：在词嵌入方面取得重要进展。

Seq2Seq (2014 年)：用于序列到序列的学习任务。

GAN (2014 年)：生成对抗网络。它由生成器网络（试图通过生成来欺骗鉴别器真正的图片）和鉴别器网络（尝试区分真假图像）组成，训练好这个网络需要联合训练一个 minima game 的优化问题：

判别器的目标是最大化目标函数，使得对于真实数据  $x$ ，判别器的输出  $D(x)$  接近 1 (表示真实)，而对于生成的假数据  $G(z)$ ，判别器的输出  $D(G(z))$  接近 0 (表示假)。相反，生成器的目标是最小化目标函数，使得  $D(G(z))$  接近 1，从而欺骗判别器认为生成的数据是真实的。

VAE (2013 年)：变分自编码器。DCGAN (2014 年)，WGAN (2017 年)，PGGAN (2017 年) 进一步扩展了 GAN 的应用。

ResNet (2016 年)：深度残差网络，解决了深度网络的梯度消失问题。DenseNet (2017 年) 进一步改进。

AlphaGo (2016 年) 和 Alpha Zero (2017 年)：在人工智能游戏中表现优异。

Capsule Nets (2017 年)：提高了图像分类的精度和效率。

这些架构和模型代表了神经网络和深度学习领域的重大进展。

### 2.2.10.1\*\* 当今的神经网络结构 \*\*

通常，一个深度神经网络是由多个模块层堆叠而成的。

基本模块包括：MLP 模块、CNN 模块、Attention 模块、RNN 模块等。

MLP：多层感知器 (MLP, Multilayer Perceptron)

简单且灵活，可以逼近任意函数

对于高维输入效率低下，易于过拟合

CNN:

优秀的捕捉局部空间模式，并使用较少的参数。主要用于图像任务。

对长程依赖处理较差，不适用于非结构化数据。

Attention block:

高效捕捉长程依赖性，并能适应多种数据类型。

计算开销较大。

示例：UNet（稳定扩散版本）

在每个分辨率下： $N * \text{CNN}$  模块 +  $M * (\text{注意力模块} + \text{MLP} \text{ 模块})$

生成任务的一些技巧：长残差连接

### 2.2.11 总结

深度前馈网络是人工神经网络中最基础最常见的一种类型。它由一系列全连接层组成，每层的神经元与下一层的神经元全部连接。数据在网络中单向传递，从输入层经过隐藏层直到输出层，称为前馈网络。

深度前馈网络的结构和工作原理如下：

首先是输入层，负责接收原始数据。输入层的神经元数目等于输入数据的维度。然后是隐藏层，在输入层和输出层之间可能存在一个或多个隐藏层，隐藏层越多，网络越“深”，提取特征的能力就越强。每个隐藏层由多个神经元组成，这些神经元通过激活函数（如 ReLU、tanh、sigmoid）对输入信号进行非线性变换。最后是输出层，负责将隐藏层的输出转换为最终的预测结果。输出层的神经元数目取决于具体任务，例如，对于二分类任务，输出层通常有一个神经元（通过 sigmoid 激活函数），而对于多分类任务，输出层的神经元数目等于类别数（通过 softmax 激活函数）。

前馈网络的数据流动过程称为前向传播。具体步骤如下：首先，输入数据通过输入层传递到第一个隐藏层，然后每个隐藏层的输出通过激活函数计算，结果作为下一个隐藏层的输入，最后一个隐藏层的输出通过输出层计算，得到最终的预测结果。

为了训练前馈网络，需要通过监督学习调整网络的权重，这个过程称为反向传播。反向传播的步骤包括：首先，计算损失函数，使用预测结果和实际标签计算损失（如均方误差、交叉熵等）。然后计算损失的梯度，即计算损失函数关于每个权重的导数（梯度）。接着，使用梯度下降方法（如随机梯度下降，SGD）更新每个权重，以最小化损失函数。

深度前馈网络在许多应用中表现出色，包括图像分类、语音识别、自然语言处理和推荐系统。其主要优势包括表达能力强，通过多个隐藏层，前馈网络能够表示复杂的非线性关系；通用性，适用于各种类型的数据和任务，只需相应调整网络结构和损失函数；可扩展性，通过增加层数和神经元数目，可以提升模型的表达能力和性能。

总之，深度前馈网络是深度学习中最基础也是最重要的模型之一。理解其架构和工作原理是学习其他复杂模型（如卷积神经网络、循环神经网络和生成对抗网络）的重要基础。

## 2.0.4 2.3 卷积神经网络

### 2.3.1 机器如何识别我们的面孔？

清华校园的天猫超市

### 2.3.2 计算机视觉任务

图像：标注图像中的对象如人、羊、狗。

目标检测：识别并标记图像中的对象，并框出对象的位置。

语义分割：将图像中的每个像素分类，如人、羊、狗等。

图像生成：根据输入生成新图像。

#### 2.3.2.1 用于视觉的 MLP? 输入图像尺寸为 372x372。

输入数据采用多层感知器 (MLP) 处理。

隐藏层数量 (k)：

第一层：× 138,384

第二层：× 138,384

第三层：× 138,384

对于识别别墅的结果：“是别墅”或“不是别墅”。

问题：

空间信息丢失。

参数爆炸。

难以适应视点变化、尺度变化、光照条件等场景。

#### 2.3.2.2 目标检测 问题：图片中有多少只猫？

10 只猫

**2.3.2.3 2D 卷积** 通过卷积可以进行边缘检测。卷积操作包括将一个大的核（或过滤器）与输入图像的不同区域进行点积。右侧的例子展示了如何使用一个简单的核进行边缘检测。输入图像通过卷积操作生成输出图像，突显了图像中的边缘。

### 2.3.3 大脑神经科学

大脑皮层中的拓扑映射：皮层中相邻的细胞代表视觉场中相邻的区域。

简单细胞：对光的方向有反应

复杂细胞：对光的方向和运动有反应

超复杂细胞：对运动及其终点有反应

视觉刺激通过视网膜神经节细胞感受野传导至 LGN 和 V1 的简单细胞，形成对应的反应模式。

### 2.3.4 概述

扫描图像：输入图像通过扫描得到。

生成特征层次结构：图像经过每一层的处理，提取并生成多层次的特征。

从高层特征中识别：最终通过高层次特征进行识别和分类。

#### 2.3.4.1 想法 1：局部连接 局部性假设：局部信息足以进行识别。

只将每个神经元连接到输入体积中的局部区域。

#### 2.3.4.2 想法 2：参数共享 平移不变性假设：如果某个特征在空间位置 $((x, y))$ 处是有用的，那么它在其他位置 $((x', y'))$ 处也应该是有用的。

在不同的空间位置上共享滑动窗口的权重。

#### 2.3.4.3 什么是卷积？ 在数学中，卷积是对两个函数 $((f) \text{ 和 } (g))$ 的操作。

$$\begin{aligned} & [ \\ & (f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \\ & ] \\ & [ \\ & = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \\ & ] \end{aligned}$$

对于定义在整数集合 ( $\mathbb{Z}$ ) 上的函数  $(f)$  和  $(g)$ ，我们可以定义它们的离散卷积：

$$\begin{aligned} & [ \\ & (f * g)(n) = \sum_{m=-\infty}^{\infty} f[m] g[n - m] \\ & ] \\ & [ \\ & = \sum_{m=-\infty}^{\infty} f[m] g[n - m] \\ & ] \end{aligned}$$

假设我们正在使用 GPS 估计位置。 $(f[n])$  是来自 GPS 的值，但可能带有噪声。

为了获得更少噪声的估计，我们可以使用最近的 GPS 值，并使用加权函数  $(g[m])$  进行平滑估计，当然最近的值应该有更高的权重。

$$\begin{aligned} & [ \\ & (f * g)(n) = \sum_{m=-\infty}^{\infty} f[m] g[n - m] \\ & ] \end{aligned}$$

假设  $f[n]$  是一个包含噪声的 GPS 位置数据序列。在时间  $n$  处，它的值可能是：

$f[-1], f[0], f[1], f[2], \dots$

假设我们选择  $g[m]$  为平滑加权函数，比如一个简单的三点平均：

$g[m]=[0.2,0.5,0.3]$

这个加权函数让最近值（中心）有更大的权重。则计算  $(f*g)(n)$  在时间  $n$  处的具体步骤是：  
 $(f*g)(n)=0.3 f[n-1]+0.5 f[n]+0.2 f[n+1]$  通过这样的方式，我们对时间点  $n$  处的 GPS 数据进行平滑，利用了它周围的点并进行了加权，使位置估计更平滑、更准确。

在机器学习中，输入通常形成多维结构。例如，卷积用于将二维输入 ( $I$ ) 与二维卷积核 ( $K$ ) 进行变换：

$$[  
Z(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i+m, j+n)  
]$$

在机器学习中，基于卷积的学习算法使用翻转卷积核会学习到相对于不翻转的卷积核的翻转版本。

因此，在很多机器学习中的卷积变为：

$$[  
Z(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)  
]$$

输入体积为  $32 \times 32 \times 3$ 。

过滤器（核）大小为  $5 \times 5 \times 3$ 。

每个神经元的输出：

卷积是过滤器和一个  $5 \times 5 \times 3$  体积块之间的点积。

卷积运算表示为：

$$[  
S(w, h) = (I * K)(w, h) = \sum_{i=1}^k \sum_{j=1}^l I(w+i, h+j) K(i, j)  
]$$

**2.3.4.3.1 步幅 (Stride)** 在某些情况下，我们希望减少空间分辨率（对输出进行下采样）。

步幅是在每个通道的空间维度上执行的。

**2.3.4.3.2 填充 (Padding)** 可能无法适配？例如，无法在步幅为 3 的  $7 \times 7$  输入上应用  $3 \times 3$  滤波器。

多层卷积会减少空间尺寸 ( $W \times H$ ) 并减少边界上的信息。

通过添加 (P) 圈的零到原始地图外部，可以使用以下公式计算新的高度和宽度：

$$[\begin{aligned} \text{\textbackslash text{height}}_{\text{\textbackslash text{new}}} &= \left\lfloor \frac{\text{\textbackslash text{height}}}{\text{\textbackslash text{filter}}} + 2 \right\rfloor \times \text{\textbackslash text{pad}} \times \text{\textbackslash text{stride}} \\ &\quad + 1 \\ \text{\textbackslash text{width}}_{\text{\textbackslash text{new}}} &= \left\lfloor \frac{\text{\textbackslash text{width}}}{\text{\textbackslash text{filter}}} + 2 \right\rfloor \times \text{\textbackslash text{pad}} \times \text{\textbackslash text{stride}} \\ &\quad + 1 \end{aligned}]$$

**2.3.4.3.3 膨胀卷积** 膨胀卷积支持感受野的指数扩展，而不会损失分辨率或覆盖范围。展示了  $3 \times 3$  的 1 膨胀卷积、2 膨胀卷积和 4 膨胀卷积如何在更大区域内捕获信息。

膨胀卷积是一种卷积操作，它通过在常规卷积核之间插入空洞（即不连续的像素）来扩大感受野。膨胀卷积的优势在于，它能够在不增加计算复杂度的情况下捕捉更大范围的上下文信息，而不会丢失细节信息。

具体来说：

1 膨胀卷积：这是标准的  $3 \times 3$  卷积，其中每个滤波器元素都是相邻的像素。感受野大小为  $3 \times 3$ 。

2 膨胀卷积：在每个滤波器元素之间插入一个空洞，即跳过一个像素。这样，感受野大小为  $5 \times 5$ ，但实际上只计算了  $3 \times 3$  的元素。

4 膨胀卷积：在每个滤波器元素之间插入三个空洞，即跳过三个像素。感受野大小为  $9 \times 9$ ，但实际上只计算了  $3 \times 3$  的元素。

通过这种方式，膨胀卷积允许感受野在指数级扩展，从而在更多的空间维度上捕获特征，而无需进行很多次平移操作。这在处理需要捕获长距离依赖关系的任务（如图像分割或语义分割）中特别有用。

**2.3.4.3.4 卷积：公式** 在处理图像时，我们通常将输入 ( $I$ ) 和输出 ( $Z$ ) 视为 3D 张量：

[

$$Z_{\{j,k\}}^{\{(i)\}} = \sum_{l,m,n} I_{\{j+m-1, k+n-1\}}^{\{(l)\}} K_{\{m,n\}}^{\{(i,l)\}} + b^{\{(i)\}}$$

]

( $Z_{\{j,k\}}^{\{(i)\}}$ ) 是第 ( $i$ ) 通道中位于第 ( $j$ ) 行和第 ( $k$ ) 列的输出单元的值。

卷积核 ( $K$ ) 是一个 4D 张量，其元素 ( $K_{\{m,n\}}^{\{(i,l)\}}$ ) 表示第 ( $l$ ) 通道的输入单元与第 ( $i$ ) 通道的输出单元之间的连接强度，行之间偏移量为 ( $m$ )，列之间偏移量为 ( $n$ )。

为了减少计算成本，我们可以跳过卷积核的一些位置：

[

$$Z_{\{j,k\}}^{\{(i)\}} = \sum_{l,m,n} I_{\{(j-1) \times s + m, (k-1) \times s + n\}}^{\{(l)\}} K_{\{m,n\}}^{\{(i,l)\}} + b^{\{(i)\}}$$

+  $b^{\{(i)\}}$

]

( $s$ ) 是此下采样卷积的步幅。

假设我们要最小化某个损失函数 ( $J(K, b)$ )。

**2.3.4.4 池化** 一个卷积层由多个阶段组成：

卷积层（阶段）

[

$$Z_{\{j,k\}}^{\{(i)\}} = \sum_{l,m,n} I_{\{j+m-1, k+n-1\}}^{\{(l)\}} K_{\{m,n\}}^{\{(i,l)\}}$$

]

非线性层（阶段）

[

$$O = f(\sum_i X_i K_i + b)$$

]

### 池化层（阶段）

池化（Pooling）是一种用来简化输出的方法，通过将某一位置附近的输出用统计值来代替该位置的输出。池化函数种类包括：

1. 最大池化（Max pooling）
2. 平均池化（Average pooling）
3. L2-正则池化（L2-norm pooling）
4. 概率加权池化（Probability weighted pooling）

图示解释了最大池化的过程：采用  $2 \times 2$  的滤波器和步幅为 2 的设置，将原矩阵中的局部区域最大值抽取出来，形成一个新的矩阵。具体示例如下：

原矩阵：

```
““
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
““
```

经过  $2 \times 2$  滤波器最大池化后的新矩阵：

```
““
6 8
3 4
““
```

池化（Pooling）的特性包括：

1. 对输入的小幅平移具有不变性（invariance to small translations）。这意味着即使输入发生了微小的变动，池化层的输出也不会有显著变化，从而提升模型的鲁棒性。
1. 提高统计效率（statistical efficiency）并减少内存需求。池化通过减少数据的空间维度，使得模型能够更高效地处理数据，降低计算资源的占用及内存需求。
1. 减少下一层的输入处理量以及处理不同大小的输入（handle inputs of variable size）。池化层将输入压缩为较小的尺寸，使得之后的层能够处理更少的数据，同时便于处理不同大小的输入，提高模型的灵活性。

#### 2.3.4.5 全连接层 卷积层和池化层可以被看作是具有无限强先验的一种全连接层。

这种先验指的是，对于其中一个隐藏神经元的权重，必须与其相邻的隐藏神经元的权重相同，但在空间中有所平移。

换句话说，除了分配给该隐藏神经元的小区域外，其他位置的权重必须为零。

同样，池化的使用也是一种无限强的先验，这个先验认为每个单元应该对小幅平移保持不变。

我们可以以比较简单易懂的方式来解释卷积和池化的原理：

1. 卷积层：可以把卷积层想象成一种特殊的全连接层。全连接层中的每个神经元都连接到前一层的所有神经元，而卷积层中的每个神经元只连接到前一层中某些邻近的神经元。具体来说，卷积层中的权重是共享的，意思是说，在一个区域内，同样的权重会应用在不同的位置上，使得它们对局部特征具有相同的响应。
1. 强先验：这里的强先验指的是一种非常强的约束条件。在卷积层中，这个先验条件是每一个局部区域的权重必须和它的邻居相同，只是位置有所不同。这样一来，当输入稍微平移时，输出仍然保持相似。
1. 池化层：池化层则是一种用来简化数据、减少计算量的方法。池化层可以看作是对输入的小幅改变不敏感的全连接层。这也是一种强先验，即认为即使输入发生了小幅度的变化，输出也应该保持稳定。

通过这种方式，卷积层能够有效地检测出图像中的局部特征，而池化层则能帮助模型对这些特征进行简化和概括，使得模型更加稳健和高效。

#### 2.3.4.6 CNN 的特征提取 卷积神经网络（CNN）在处理图像时，会逐层提取不同层次的特征。具体过程如下：

1. 低级特征 (Low-Level Feature)：在初始层，CNN 会检测到一些简单的视觉特征，如边缘、纹理等。这些特征往往是由图像中的像素值变化直接引起的。
1. 中级特征 (Mid-Level Feature)：随着层数的增加，CNN 会开始识别更复杂的模式，比如基本的形状、轮廓和角等。这些特征是低级特征的组合，形成一些较为明显的几何形状。
1. 高级特征 (High-Level Feature)：在更深的层，CNN 会提取出更加抽象和复杂的特征，比如特定物体的部件或形态。这些高级特征能够更好地描述图像中的实际内容。
1. 可训练分类器 (Trainable Classifier)：当所有特征都被提取出来后，这些特征会被送入一个可训练的分类器中，通过对这些特征的综合分析，最终实现对图像的分类或识别。

通过这种逐层提取特征的方式，CNN 能够从低级到高级逐渐理解图像的内容，从而完成复杂的视觉任务。

#### 2.3.4.7 AlexNet

AlexNet，诞生于 2012 年，是最早的“深度学习”模型代表之一。

AlexNet 是首个在图像分类上取得显著改善的深度卷积神经网络 (CNN) 方法。它采用了包括 7 层的深度卷积神经网络，并使用了 ReLU 激活函数和 dropout 技术。

这一巨大成功极大地推动了卷积神经网络的发展和创新。

具体来说，AlexNet 的结构包括多层卷积层和池化层，用于提取图像的特征，并通过全连接层进行分类。通过使用局部响应归一化 (Local Response Normalization) 和最大池化 (Max Pooling)，AlexNet 在图像识别任务中取得了显著的性能提升。

AlexNet 的成功证明了深度学习方法在处理复杂视觉任务中的强大能力，并引领了后续许多深度学习模型的发展。

#### 2.3.4.8 ResNet

ResNet，即残差网络，是 2015 年推出的深度卷积神经网络链条中的一个重要版本。它通过引入残差连接 (residual connection) 有效地解决了深层网络训练困难的问题。残差连接允许信息在网络层之间轻松穿梭，使得网络变得更深的同时仍能保持或提升性能。

微软研究院 (MSRA) 在 2015 年的 ILSVRC 比赛中使用了 ResNet 的集合模型，并赢得了大多数比赛。ResNet-34 就是其中一个典型的架构，它与 VGG-19 和传统的 34 层卷积神经网络相比，具有显著的优势。

ResNet 之所以能够实现超深网络 (如达到 1202 层) 的训练得益于其独特的残差连接设计，这种设计帮助网络在极高深度下仍然能够有效地进行学习和优化，从而获得优异的性能表现。

##### 2.3.4.8.1 残差神经网络

残差神经网络 (Residual Neural Networks/ResNet) 由 He 等人于 2015 年提出，通过在神经网络中总结低层次隐含表示和高层次隐含表示来解决深层网络的训练问题。

其核心思想是通过高层次表示学习低层次表示与目标标签之间的“残差”。具体实现中，残差块会在原始输入和经过权重层处理后的输出之间增加一个快捷连接 (identity mapping)，将输入直接传递给输出。这种结构可以有效避免深层网络中梯度消失的问题，使得网络可以更好地训练，并显著提升性能。

残差神经网络 (ResNet) 的表现略优于深度相同的普通卷积神经网络 (plain CNNs)。残差神经网络通过学习残差，同时保留低层次表示，因而能够更深地扩展。

左图显示了普通卷积神经网络在不同层数上的误差变化。可以看到，随着层数增加，误差反而变大，说明普通网络在训练深层网络时遇到了困难。

右图展示了不同层数的 ResNet 在训练过程中的误差变化。即便层数增加到 110 层，误差仍然能够维持在较低水平，表明 ResNet 极大地改善了深层网络的训练效果，使得网络可以在深度增加的情况下依然保持良好的性能。

总体来说，ResNet 通过引入残差连接，使得信息能够在层之间更快捷地传递，从而有效解决了深层网络中常见的训练问题，显著提升了深度学习模型的效果。

#### 2.3.4.9 U-Net

U-Net 在 2015 年由 Olaf Ronneberger 等人首次提出，主要用于生物医学图像分割。其也可以用于生成二维信息的任何网络，比如图像生成、风格迁移、深度估计等。

U-Net 的基本结构是 U 形的，包括对称的编码器（收缩路径）和解码器（扩展路径）两部分。

**跳跃连接 (Skip Connections):** 跳跃连接将编码器和解码器中对应的对称层相连接，以保留详细信息并改善特征传输。这种结构使得在处理高分辨率图像时能够更好地恢复细节，提高了分割精度。

U-Net 的设计理念使得它在分割任务中表现出色，尤其适用于需要精细分割的场景，如医学图像分析等领域。

**2.3.4.10 DenseNet** 密集连接神经网络 (Densely Connected Neural Networks)，简称 DenseNet，于 2017 年提出。DenseNet 的核心概念是每一层都与之前的所有层相连接，而不仅仅是与前一层相连接。

这种结构使得特征可以在网络中高效地复用，确保了低层特征能被高层使用，从而增强了模型的表达能力和梯度传递效果，减少了梯度消失的问题。

通过保留低层次表示，DenseNet 能够进行非常深的网络扩展，同时保持良好的性能。这种密集连接的策略使得 DenseNet 在多个图像识别任务中取得了优异的效果。

DenseNet 的设计突显了特征复用的重要性，显著提高了网络的计算效率和性能，为深度学习的发展提供了新的思路。

密集连接神经网络 (DenseNet) 在使用相同数量的参数或计算量 (flops) 时，表现优于残差神经网络 (ResNet)。

然而，DenseNet 的密集连接特性导致其并行计算效率较低。这构成了一种权衡 (tradeoff)：DenseNet 在精度上有所优势，但在计算效率上有所牺牲。

图表左侧显示了 DenseNet 和 ResNet 在使用不同参数数量时的验证错误率；右侧显示了在测试时不同计算量下的验证错误率。可以看到，DenseNet 在同类条件下的表现更佳。

此外，需要注意的是，ResNet 和 DenseNet 的作者均毕业于清华大学，这表明在深度学习领域，清华大学培养了大量杰出的研究人员并做出了重要贡献。

**2.3.4.11 T\*\*emporal convolutional network (TCN)\*\*** Temporal convolutional network (TCN) 是一种考虑时间因素的卷积神经网络，它于 2018 年被 Bai, Shaojie 等人提出。研究背景是解决序列建模问题，利用卷积架构解决该问题并证明结果优于传统的循环网络，如 lstm，同时显示出更长的有效记忆。

对于一个 1 维输入  $x_0, x_1, \dots, x_T$ ，TCN 考虑了因果约束，也就是如果要预测一段时间  $t$  的输出，那么被限制只使用那些已经观察到的输入： $x_0, x_1, \dots, x_T$ ，而不是任何“未来”的输入  $x_{t+1}, \dots, x_T$ 。

该网络的学习目标是：找到一个网络  $f$ ，使实际输出和预测之间的预期损失最小化：

$$\hat{y}_0, \dots, \hat{y}_T = f(x_0, \dots, x_T)$$

这种思想也被称为：因果卷积 (causal convolutions)，即  $t$  时刻的输出只由  $t$  时刻和更早前一层的元素卷积计算得到。

## 2.0.5 2.4 序列建模：循环神经网络和递归神经网络

### 2.4.1 引言

循环神经网络（RNN）是一类人工神经网络，在这种网络中，单元之间的连接形成一个有向循环。

与前馈神经网络不同，循环神经网络可以利用其内部记忆来处理任意的输入序列。

第一个循环神经网络在 1980 年代被发明。

递归神经网络（Recurrent Neural Networks, RNNs）是一类用于处理序列数据的神经网络。RNN 的发展历史可以追溯到 20 世纪 80 年代以来，经历了多个重要的阶段和突破。

**20 世纪 80 年代 RNN 概念的提出**

1982 年: John Hopfield 提出了 Hopfield 网络，类似于反馈神经网络，它是早期递归神经网络的基础。

1986 年: David Rumelhart 等人引入了反向传播算法，并随后扩展到了时间序列数据，成为后来的 RNN 的基础。

**20 世纪 90 年代 RNN 的早期研究**

1990 年: Elman 网络的提出（由 Jeffrey Elman），这种简单的 RNN 结构使用了隐层状态来记录过去的信息。

1992 年: Jürgen Schmidhuber 和 Sepp Hochreiter 提出了长期依赖问题，并且初步讨论了如何使用 RNN 处理长时间依赖关系。Hochreiter 更是提出了 BPTT (Backpropagation Through Time) 算法来训练 RNN。

**1997 年 LSTM 的提出**

1997 年: Sepp Hochreiter 和 Jürgen Schmidhuber 提出了 Long Short-Term Memory (LSTM) 网络，目的是解决传统 RNN 存在的长时间依赖信息容易遗忘的问题。LSTM 通过引入门机制，显著改善了传统 RNN 对长期依赖的学习效果。

**2000 年代 RNN 相关改进与应用**

2001 年: LeCun 等人提出了 GRU (Gated Recurrent Unit)，它是对 LSTM 的一种简化，不同的是 GRU 只有两个门（更新门和重置门），但在许多任务上表现不错。

这一时期，RNN 和 LSTM 逐渐在实践中应用于各种实际问题，包括自然语言处理、语音识别和时间序列预测等。

**2010 年代深度学习的兴起和 RNN 的大规模应用**

随着深度学习的兴起，计算能力和大数据的发展，RNN 和 LSTM 得到了更广泛的应用。

2014 年: Google 的研究中，LSTM 被用于机器翻译任务，取得了显著的成功。

同年，RNN 结构的变体，如双向 RNN、深度 RNN 也被提出和应用。

2014 年: Facebook 推出了针对序列数据建模的著名工具包 Torch，其中包含了 RNN 的各种实现。

**近年来的发展**

Transformer 模型的引入：2017 年，Vaswani 等人在论文《Attention is All You Need》中提出 Transformer 模型，该模型通过自注意力机制在许多任务上超过了传统的 RNN 和 LSTM，尤其是

在自然语言处理任务中表现出色。

Transformers 引发的新架构：基于 Transformer 的预训练模型，如 BERT、GPT、T5 等，在 NLP 领域成为新的标准，逐渐取代了 RNN 在许多任务中的地位。

尽管 Transformer 和其变种如 BERT、GPT 等在很多任务上显示出更优越的性能，但 RNN 及其变种（如 LSTM 和 GRU）仍在许多特定领域和应用中具有重要地位。

总结而言，RNN 自 20 世纪 80 年代的提出至今，经历了概念设立、复杂模型（如 LSTM、GRU）的提出到大规模应用，以及现代深度学习模型（如 Transformer）的冲击。RNN 及其变体在序列数据处理上发挥了重要作用，并且结合新的架构和方法，不断推动着前沿技术的发展。

**2.4.1.1 循环神经网络提供了很大的灵活性** 在第一种情况中，我们看到一个“one to one”（一对一）的结构，类似于简单的神经网络。一个输入对应一个输出。

在第二种情况中，是“one to many”（一对多）的结构，这种结构从一个输入生成多个输出，适用于序列生成任务，比如图像描述生成。

例如，图像字幕：图像 → 的单词序列

第三种情况，“many to one”（多对一）表示多个输入对应一个输出，常用在情感分析中，通过多个词预测情感标签。

例如，情绪分析：单词序列的 → 情绪

最后一种情况是“many to many”（多对多），多个输入对应多个输出，可以处理诸如翻译和视频处理的任务。

例如，机器翻译：序列 → 序列

最后两种情况都是“多对多”（many to many）。第一个“多对多”结构适用于输入与输出的序列长度不相等的情况，例如机器翻译。第二个“多对多”结构适用于输入与输出的序列长度相等的情况，例如词性标注（POS Tagging）。

图例中的箭头表示数据流的方向，红色箭头表示输入，绿色方框表示中间的隐藏层，蓝色箭头表示输出。

**2.4.1.2 历史** 这张图展示了深度学习和自然语言处理（NLP）领域的一些关键发展历程。

1. 最早可以追溯到 1958 年，Rosenblatt 提出了感知器（Perceptron），为神经网络奠定了基础。
2. 1982 年，Hopfield 提出了 Hopfield 网络，这是一种具有反馈的循环神经网络。
3. 1986 年，分布式表示理论被引入，显著推动了深度学习的发展。
4. 1997 年，LSTM（长短期记忆网络）由 Schmidhuber 等人提出，大大改善了序列数据处理的能力。
5. 2003 年，Bengio 提出了神经概率语言模型（Neural Probabilistic Language Model），进一步发展了语言模型的概念。
6. 2010 年，基于 RNN（循环神经网络）的语言模型被提出，进一步提升了语言模型的表现。

7. 2013 年, word2vec 被 Mikolov 提出, 带来了词嵌入技术的突破。
8. 2014 年, Deepwalk 提出, 用于网络嵌入。
9. 2014 年, Glove 也被引入作为另一种基于计数的词嵌入方法。
10. 2014 年, GRU (门控循环单元) 和 Seq2Seq 模型被提出, 极大地提升了序列到序列任务的表现。
11. 2015 年, 注意力机制被引入, 如”show, attend and tell” 模型, 增强了模型的灵活性和性能。
12. 2016 年, ByteNet 和 context2vec 被提出, 在嵌入和翻译任务上有了新的进展。
13. 2017 年, Conv Seq2Seq 模型被 Facebook 引入, 用于机器翻译。
14. 2017 年, Google 推出了新的 AI 架构, 进一步改善了深度学习的应用。
15. 2018 年, BERT 和 OpenAI GPT 被提出, 标志着预训练语言模型的重大突破。
16. 2018 年, ELMo 模型也被引入, 用于动态词嵌入。

这张图展示了从感知器到 LSTM, 再到现代的预训练语言模型 (如 BERT 和 GPT), 深度学习和自然语言处理技术的演进过程。每一个节点和箭头都展示了关键人物和他们的研究成果如何相互影响与推动。

**2.4.1.3 循环神经网络实例** 当这种深度前馈网络的连接扩展到包括反馈连接时, 被称为循环神经网络 (Recurrent Neural Networks, RNN)。

图 (a) 显示了一个深度前馈网络 (DFN) 的架构, 数据从输入层经过隐藏层, 最后到达输出层, 没有任何反馈连接。图 (b) 则展示了一个循环神经网络 (RNN) 的架构, 其中包括反馈连接, 允许信息在网络内进行循环。

深度前馈网络和循环神经网络的主要区别在于连接方式, 前者没有循环连接, 后者具备反馈机制。

1. 输入  $x$  和输出  $y$  来自训练数据;
2.  $o$  是估计的输出值;
3.  $L$  用于评估估计输出  $\hat{y} = \text{softmax}(o)$  与真实输出  $y$  之间的差异 (损失);
4.  $U, W, V$  是三个权重矩阵。

图解说明了一个循环神经网络 (RNN) 的工作流程。输入  $x$  通过权重矩阵  $U$  进行处理, 生成隐藏状态  $h$ 。隐藏状态  $h$  通过权重矩阵  $W$  来考虑前一个时间步的信息, 更新后的隐藏状态通过权重矩阵  $V$  计算出估计的输出  $o$ 。损失函数  $L$  根据估计输出  $o$  和真实输出  $y$  来评估两者之间的差距, 通过训练过程来最小化这个损失。

循环神经网络的一个例子

让我们展开图形表示。

左边的图表示一个循环神经网络 (RNN) 的单个时间步。输入  $x$  通过权重矩阵  $U$  生成隐藏状态  $h$ 。隐藏状态  $h$  通过权重矩阵  $W$  与前一时间步的信息结合，然后通过权重矩阵  $V$  计算出估计的输出  $o$ 。损失函数  $L$  根据估计输出  $o$  和真实输出  $y$  评估两者之间的差异。

右边的图是展开后的 RNN，展示了多个时间步 ( $t-1, t, t+1$ ) 的处理过程。每个时间步的输入  $x$  由相应的权重矩阵  $U$  处理后生成当前的隐藏状态  $h$ 。隐藏状态  $h$  通过权重矩阵  $W$  传递给下一个时间步。每个时间步的估计输出  $o$  通过权重矩阵  $V$  计算并用损失函数  $L$  评估，最后与相对应时间步的真实输出  $y$  进行比较。

这里展示了 RNN 的更新方程：

隐藏状态 ( $h^{\{(t)\}}$ )

$h(t)$  的更新公式为：

$$[ h^{\{(t)\}} = \tanh(b + W h^{\{(t-1)\}} + U x^{\{(t)\}}) ]$$

其中，( $b$ ) 是偏置，( $W$ ) 是隐藏状态的权重矩阵，( $U$ ) 是输入的权重矩阵。

输出 ( $o^{\{(t)\}}$ )

$o(t)$  的计算公式为：

$$[ o^{\{(t)\}} = c + V h^{\{(t)\}} ]$$

其中，( $c$ ) 是偏置，( $V$ ) 是输出的权重矩阵。

估计输出 ( $\hat{y}^{\{(t)\}}$ )

$\hat{y}^{\{(t)\}}$  使用 softmax 函数计算：

$$[ \hat{y}^{\{(t)\}} = \text{softmax}(o^{\{(t)\}}) ]$$

目标函数 (损失函数) 为：

$$[ L = \sum_t L^{\{(t)\}} = \sum_t \log P(\hat{y}^{\{(t)\}} | x^{\{(1\}, \dots, x^{\{(t)\}}}) ]$$

此损失函数对多个时间步上的估计输出与真实输出之间的差异进行求和。

图示展示了多个时间步的展开过程，每个时间步的信息通过权重矩阵传递并影响后续时间步的计算。这样能够有效捕捉序列数据中的时间依赖性。

#### 2.4.1.4 循环神经网络类型

循环神经网络有许多不同的类型：

隐藏单元之间的循环连接

从一个时间步的输出到下一个时间步的隐藏单元的循环连接

隐藏单元之间的循环连接，但只产生一个输出

.....

##### 2.4.1.4.1 输出循环

图 (a): 输出循环的网络

这个图展示了输出循环网络，唯一的循环连接是从输出层到隐藏层。网络展开后展示了多个时间步的处理过程。

图 (b): 教师强制

教师强制是一种训练循环神经网络的技术。训练时使用真实的输出值进行计算，而在测试时使用模型的预估值。如果在训练和测试中使用相同的输入方式，就能更好地逼近理想输出状态。

条件似然函数：

教师强制的条件似然函数表示为：

$$[\log p(\hat{y}(1), \hat{y}(2) | \hat{x}(1), \hat{x}(2)) = \log p(\hat{y}(2) | \hat{y}(1), \hat{x}(1), \hat{x}(2)) + \log p(\hat{y}(1) | \hat{x}(1), \hat{x}(2))]$$

这种条件似然函数反映了输出值与输入数据之间的概率关系，通过最大化这个似然函数，可以更好地训练模型。

#### 2.4.1.4.2 关于输出循环的讨论 优点：

所有时间步都是解耦的，可以并行训练

缺点：

缺乏隐藏层到隐藏层的循环，严格来说功能较弱

解决方案：

随机选择使用生成的值或实际值作为输入

**2.4.1.5 循环神经网络中的梯度计算** 梯度可以通过一种广义的反向传播算法来计算，这种算法具体叫做时间上的反向传播（BPTT）。

以下是梯度计算的步骤和公式：

1. 对于输出层的梯度：

$$[(\nabla_{o(t)} L) = \frac{\partial L}{\partial o_i(t)} = \frac{\partial L}{\partial \hat{o}_i(t)} = \hat{y}_i(t) - y_i(t)]$$

1. 对于隐藏层的梯度：

$$[(\nabla_{h(t)} L) = W^T (\nabla_{h(t+1)} L) \text{diag}(1(h(t+1))^2) + V^T (\nabla_{o(t)} L)]$$

1. 对于偏置向量 (c) 的梯度：

$$[\nabla_c L = \sum_t \nabla_{o(t)} L]$$

1. 对于偏置向量 (b) 的梯度：

$$[\nabla_b L = \sum_t \text{diag}(1(h(t))^2) \nabla_{h(t)} L]$$

1. 对于权重矩阵 (V) 的梯度：

$$[\nabla_V L = \sum_t (\nabla_{o(t)} L) (h(t))^T]$$

1. 对于权重矩阵 (W) 的梯度：

$$[\nabla_W L = \sum_t \text{diag}(1/(h^{(t)})^2) (\nabla_{h^{(t)}} L) (h^{(t-1)})^{\top}]$$

1. 对于权重矩阵 ( U ) 的梯度:

$$[\nabla_U L = \sum_t \text{diag}(1/(h^{(t)})^2) (\nabla_{h^{(t)}} L) (x^{(t)})^{\top}]$$

这些公式通过时间上的反向传播 (BPTT) 来计算循环神经网络的梯度，从而在训练过程中调整网络的参数，使损失函数 ( L ) 最小化。

**2.4.1.6 循环神经网络作为有向图模型** 下图总结了我们讨论的两种 RNN 类型。左侧图展示了隐藏层-隐藏层循环的网络，右侧图展示了输出-隐藏层循环的网络。两种类型分别通过不同的对数似然函数来最大化训练目标。

图 (a): 隐藏层-隐藏层循环

这个图展示了循环神经网络中的隐藏层到隐藏层的循环连接。展开后的图展示了多个时间步的处理过程，每个时间步间的隐藏状态通过权重矩阵 ( W ) 传递给下一个时间步。训练目标是最大化对数似然函数 ( $\log P(y^{(t)} | x^{(1)}, \dots, x^{(t)})$ )。

图 (b): 输出-隐藏层循环

这个图展示了循环神经网络中的输出层到隐藏层的循环连接。展开后的图展示了多个时间步的处理过程，每个时间步的输出通过权重矩阵 ( W ) 反馈回隐藏状态。训练目标是最大化对数似然函数 ( $\log P(y^{(t)} | x^{(1)}, \dots, x^{(t)}, y^{(1)}, \dots, y^{(t-1)})$ )。

**2.4.1.7 用循环神经网络对序列进行建模** 用循环神经网络对序列进行建模，并根据上下文条件生成

图展示了循环神经网络 (RNN) 在图像描述等任务中的应用。

图 (a): 这个图展示了一个 RNN 模型如何从单个图像输入生成描述图像的词语序列。通过将图像信息与时间步的序列数据结合，模型能够逐步生成描述图像内容的词语。

图 (b): 这是一个 Feifei Li 在 CVPR'15 会议论文中的实际例子。这里展示了 RNN 如何生成描述图像内容的词语。但是，在这个例子中，RNN 只在第一个时间步使用了图像信息作为条件。

在这些应用中，RNN 通过从图像中提取特征来生成一系列描述词语。例如，对一张含有草帽的图片进行描述，模型会生成类似于“straw hat”的词语序列。模型首先通过卷积神经网络 (CNN) 提取图像特征，然后这些特征被输入到 RNN 的第一个时间步，接下来每个时间步通过结合前面的词语生成新的词语，直到生成完整描述。

## 2.4.2 双向 RNN

**2.4.2.1 双向循环神经网络** 为了进行分类，你需要结合来自前后文的信息。

图例展示了双向循环神经网络 (Bidirectional RNNs) 的结构，它通过结合序列中前向传播和后向传播的数据来进行处理。

双向 RNN 有两个隐藏层，一个用于从左到右的前向传播，另一个用于从右到左的后向传播。

更新公式如下：

前向隐藏状态 ( $\overrightarrow{h_t}$ ):

$$\begin{aligned} & [ \\ & \overrightarrow{h_t} = f(\overrightarrow{W} x_t + \overrightarrow{V} \overrightarrow{h_{t-1}} + \overrightarrow{b}) \end{aligned}$$

后向隐藏状态 ( $\overleftarrow{h_t}$ ):

$$\begin{aligned} & [ \\ & \overleftarrow{h_t} = f(\overleftarrow{W} x_t + \overleftarrow{V} \overleftarrow{h_{t-1}} + \overleftarrow{b}) \end{aligned}$$

最终输出 ( $\hat{y}$ ):

$$\begin{aligned} & [ \\ & \hat{y} = g(U \overrightarrow{h_t} + c) = g(U[\overrightarrow{h_t}, \overleftarrow{h_t}] + c) \end{aligned}$$

在双向 RNN 中，每个时间步的隐藏状态都包括前向和后向两个方向的信息，这使得模型可以更全面地理解输入序列中的信息。这种架构特别适用于需要结合前后文信息的任务，如自然语言处理中的命名实体识别和语音识别。

**2.4.2.2 深层双向循环神经网络 (Deep Bidirectional RNN)** 一个深层双向循环神经网络包含三层 RNN，用于更高效地捕捉序列数据中的复杂依赖关系。

更新公式：

前向隐藏状态 ( $\overrightarrow{h_t}^i$ ) 在第 (i) 层：

$$\begin{aligned} & [ \\ & \overrightarrow{h_t}^i = f(\overrightarrow{W}^i \overrightarrow{h_{t-1}}^i \overrightarrow{V}^i \overrightarrow{h_t}^{i-1} + \overrightarrow{b}^i) \end{aligned}$$

后向隐藏状态 ( $\overleftarrow{h_t}^i$ ) 在第 (i) 层：

$$\begin{aligned} & [ \\ & \overleftarrow{h_t}^i = f(\overleftarrow{W}^i \overleftarrow{h_{t-1}}^i \overleftarrow{V}^i \overleftarrow{h_t}^{i-1} + \overleftarrow{b}^i) \end{aligned}$$

最终输出 ( $\hat{y}$ ):

$$\begin{aligned} & [ \\ & \hat{y} = g(U \overrightarrow{h_t}^L + c) = g(U [\overrightarrow{h_t}^L, \overleftarrow{h_t}^L] + c) \end{aligned}$$

其中，( L ) 表示网络的最后一层。

在此结构中，每层的前向和后向信息都通过多个时间步传播，从而在每一层中捕捉序列的双向依赖关系。这种架构能够更好地理解和处理复杂的序列数据，是许多自然语言处理和时间序列预测

任务中的常见选择。

### 2.4.3 编码器-解码器架构

#### 2.4.3.1 编码器-解码器序列到序列架构

上图展示了我们目前讨论的循环神经网络（RNN）架构。

图 (a): 序列到固定大小向量

这种架构将输入序列映射到一个固定大小的向量，应用于将可变长度输入转换为固定长度表示，如句子编码。

图 (b): 固定大小向量到序列

这种架构从固定大小的向量生成输出序列，常用于生成任务，如将一个语义向量解码为句子。

图 (c): 序列到同长度序列

这种架构将输入序列映射到相同长度的输出序列，通过 RNN 捕捉输入序列的时间依赖，逐步生成与输入相同长度的输出。

这些架构都用于不同的任务，如语音识别、机器翻译和问答系统。其中一个关键问题是，能否训练 RNN 将输入序列映射到不一定相同长度的输出序列。这在许多应用中非常重要，如语音识别、机器翻译和问答系统等。RNN 通过编码器-解码器架构实现这一点，能够处理多种序列到序列的任务。

上图展示了一个编码器-解码器或序列到序列的循环神经网络（RNN）架构。

在这个架构中，输入序列 ( $x$ ) 的每个元素 ( $x^{\{i\}}$ ) 被编码器逐步处理，最终生成一个固定大小的向量表示 ( $C$ )。这个向量 ( $C$ ) 捕捉了整个输入序列的信息。然后，解码器使用这个固定大小的向量 ( $C$ ) 来生成输出序列 ( $y$ )。

当 ( $C$ ) 是一个固定大小的向量时，这种架构可以视为“序列到固定大小向量”的 RNN 和“固定大小向量到序列”的 RNN 的结合。这种组合使得模型能够将可变长度的输入序列映射到可变长度的输出序列，在许多应用中非常重要，如机器翻译和文本生成。

当 ( $C$ ) 为固定大小向量时，这种架构可以视为“序列到固定大小向量”的 RNN 和“固定大小向量到序列”的 RNN 的组合。

上图展示了编码器-解码器或序列到序列的循环神经网络（RNN）架构。具体来说，输入序列 ( $x$ ) 的每个元素 ( $x^{\{i\}}$ ) 被编码器逐步处理，生成一个固定大小的向量表示 ( $C$ )。这个向量 ( $C$ ) 捕捉了整个输入序列的信息。然后，解码器使用固定大小的向量 ( $C$ ) 生成输出序列 ( $y$ )。

此架构的局限在于，固定大小的向量 ( $C$ ) 可能过于小，无法总结长序列的信息 (Bahdanau 等, 2015 年)。这意味着对于较长的输入序列，信息压缩至固定大小的向量可能导致信息损失，从而影响输出的准确性。

#### 2.4.3.2 编码器-解码器架构与注意力机制

固定大小的向量 ( $C$ ) 可能过小，无法有效总结长序列 (Bahdanau 等, 2015 年)。为了解决这个问题，他们建议将 ( $C$ ) 改为可变长度序列。此外，他们引入了注意力机制。

注意力机制通过对每个输入位置的相关性进行评分来动态调整，它使模型能够关注到更加相关的输入信息，而不必压缩所有信息到固定大小的向量中。

下图展示了这种机制的工作原理。公式中的 ( $a(\cdot)$ )

$a(\cdot)$

是一个对齐模型，它计算输入位置 ( $j$ ) 与输出位置 ( $i$ ) 之间的匹配程度。这个对齐模型被参数化为一个前馈神经网络。通过这种方式，注意力机制允许解码器在生成每个时间步的输出时动态选择最相关的输入部分，提高了模型在处理长序列时的表现。

在图中，( $\alpha$ ) 代表不同时间步上注意力权重，它们决定了每个隐藏状态 ( $h$ ) 对生成当前输出的重要性。这种机制有效地解决了固定大小向量 ( $C$ ) 导致的信息瓶颈问题。

考虑一个输入（或中间）序列，以及一个更高级别的表示，这个表示可以在执行某些任务（例如机器翻译）时选择“在哪里看”每个位置。

在注意力机制中，高级别表示（如图中的蓝色节点）会在处理每个时间步的任务时，根据输入序列（下级别的蓝色节点）的不同位置进行加权。Softmax 函数用于计算每个下级别位置的权重，这些权重根据上下文在较低和较高位置之间进行调整。

该方法允许模型在执行任务时动态调整聚焦点，提高了对长序列的处理能力，也使得模型在处理复杂任务时具有更大的灵活性和表现力。例如，在机器翻译中，注意力机制可以帮助模型准确地对齐源语言和目标语言的词语，提高翻译质量。

解码器常常被训练来预测下一个单词 ( $y_t$ )，给定上下文向量 ( $c$ ) 和所有之前的单词 ( $\{y_1, y_2, \dots, y_{t-1}\}$ )。

$\{y_1, y_2, \dots, y_{t-1}\}$

如公式所示：

$$[ p(y_t | y_1, \dots, y_{t-1}, x) = g(y_{t-1}, s_t, c_t) ]$$

其中，( $s_t$ ) 是位置 ( $t$ ) 的 RNN 隐藏状态，通过以下公式计算 (5)：

$$[ s_i = f(s_{i-1}, y_{i-1}, c_i) ]$$

图解展示了注意力机制在解码过程中的工作方式，通过计算每个时间步的上下文向量 ( $c_t$ ) 来确定解码器在生成每个输出单词时应关注输入序列的哪些部分。注意力权重 ( $\alpha_{t,i}$ ) 用于表示输入序列中每个位置的重要性。这种机制显著提高了模型在处理长序列和复杂翻译任务时的表现。

**2.4.3.3 注意力机制** 上下文向量 ( $c_t$ ) 依赖于注释序列 ( $(h_1, \dots, h_T)$ )，这些注释由编码器将输入句子映射而来。上下文向量的计算公式如下：

$$[ c_t = \sum_{j=1}^T \alpha_{t,j} h_j ]$$

图解展示了注意力机制的工作方式。注意力权重 ( $\alpha_{t,j}$ ) 表示在时间步 ( $t$ ) 时，输入序列中位置 ( $j$ ) 的重要性。这些权重由模型动态计算，并用于加权输入序列的隐藏状态 ( $h_j$ )。最终，上下文向量 ( $c_t$ ) 是所有隐藏状态的加权和，它在解码器生成下一个输出 ( $y_t$ ) 时提供了相关的上下文信息。这种机制使得模型能够更加有效地捕捉长序列中的依赖关系。

权重 ( $\alpha$ ) 称为“注意力矩阵”，用于选择“关注哪里”，其计算公式如下：

$$[ \alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^T \exp(e_{t,k})} ]$$

其中，

$$[ e_{tj} = h(s_{t-1}, h_j) ]$$

两个公式展示了注意力权重的计算方式。注意力矩阵通过计算输入序列中每个位置 ( $j$ ) 与位置 ( $t$ ) 的输出的匹配度来决定关注哪些部分。匹配度 ( $e_{tj}$ ) 是由注意力模型 ( $h$ ) 根据前一时间步的隐藏状态 ( $s_{t-1}$ ) 和输入位置 ( $h_j$ ) 计算的分数。然后，注意力权重 ( $\alpha_{tj}$ ) 通过对这些分数进行 softmax 归一化获得，确保其总和为 1。这种机制使得模型能够动态地选择与当前任务最相关的信息，从而提高模型的表现。

注意力模型可以被参数化为前馈神经网络（或多层次感知器等），并与系统的所有其他组件共同训练。这意味着在训练过程中，注意力模型会结合整个网络的反馈进行优化，以提升整体系统的性能。

#### 2.4.3.4 带有序列到序列架构的注意力机制

总结来说，带有序列到序列架构的注意力机制具有以下形式化表示：

输出单词的概率由以下公式计算：

$$[ p(y_i | y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i) ]$$

隐藏状态 ( $s_i$ ) 通过以下公式更新：

$$[ s_i = f(s_{i-1}, y_{i-1}, c_i) ]$$

上下文向量 ( $c_i$ ) 由输入序列的加权求和得到：

$$[ c_i = \sum_{j=1}^T \alpha_{ij} h_j ]$$

注意力权重 ( $\alpha_{ij}$ ) 使用 softmax 函数计算：

$$[ \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j=1}^T \exp(e_{ij})} ]$$

匹配得分 ( $e_{ij}$ ) 定义为：

$$[ e_{ij} = a(s_{i-1}, h_j) ]$$

图展示了一个对齐模型 ( $a(\cdot)$ )，它计算输入序列中位置 ( $j$ ) 处的输入和生成目标序列的位置 ( $i$ ) 处的输出之间的匹配程度。

这个注意力机制允许解码器在生成每个输出单词时动态关注输入序列的不同部分，从而能够更准确地处理长序列和复杂任务。通过这种机制，模型能够根据输入序列中的相关信息生成更好的输出结果。

#### 2.4.4 长期依赖

##### 2.4.4.1 让我们回顾一下梯度消失和梯度爆炸问题

让我们首先回顾在“深度模型训练优化”中提到的长时依赖性问题。

###### 2.4.4.1.1 纯线性情况

- 考虑一个没有非线性的递归神经网络 (RNN)

递归神经网络的状态更新公式为：

[

$$h^{\wedge}\{(t)\} = Wh^{\wedge}\{(t-1)\} + Vx^{\wedge}\{(t)\} + b$$

]

梯度的递推公式为:

[

$$\frac{\partial h^{\wedge}\{(t)\}}{\partial h^{\wedge}\{(k)\}} = \prod_{j=k+1}^t W^{\wedge}\top = (W^{\wedge}\top)^{t-k}$$

]

### 1. 假设矩阵 $W$ 已经进行特征值分解

令 ( $W^{\wedge}\top = V \text{diag}(\lambda) V^{-1}$ ),

则有:

[

$$(W^{\wedge}\top)^{t-k} = V \text{diag}(\lambda^{t-k}) V^{-1}$$

]

### 1. 梯度消失与梯度爆炸问题

如果任何特征值 ( $\lambda_i$ ) 的绝对值不接近 1, 它将会出现两种情况之一:

若 ( $|\lambda_i| > 1$ ), 梯度会爆炸。

若 ( $|\lambda_i| < 1$ ), 梯度会消失。

这种现象就是我们所称的梯度消失与梯度爆炸问题。当进行反向传播时, 超过或小于某个阈值的特征值导致梯度增大或减小到不可忽视的程度, 这会影响模型的训练效果。

#### 2.4.4.1.2 非线性情况 让我们讨论雅可比矩阵及其范数:

对于递归神经网络, 状态更新公式为:

$$[ h^{\wedge}\{(t)\} = f(Wh^{\wedge}\{(t-1)\} + Vx^{\wedge}\{(t)\} + b) ]$$

其梯度表示为:

$$[\frac{\partial h^{\wedge}\{(t)\}}{\partial h^{\wedge}\{(k)\}} = \prod_{j=k+1}^t \frac{\partial h^{\wedge}\{(j)\}}{\partial h^{\wedge}\{(j-1)\}} = \prod_{j=k+1}^t W^{\wedge}\top \text{diag} \left( f'(h^{\wedge}\{(j-1)\}) \right)]$$

并且有如下范数不等式:

$$[\left| \frac{\partial h^{\wedge}\{(t)\}}{\partial h^{\wedge}\{(j-1)\}} \right| \leq |W^{\wedge}\top| \times \left| \text{diag} \left( f'(h^{\wedge}\{(j-1)\}) \right) \right|]$$

使用非线性激活函数可能会带来以下影响:

它能够约束递归神经网络的动态行为, 从而减轻梯度爆炸问题。

但是, 可能会使梯度消失问题变得更加严重。

在剩余的内容中, 我们将讨论用于减少学习长时依赖困难的各种方法。

**2.4.4.2 处理梯度爆炸的技巧：梯度剪裁** 解决梯度爆炸问题的一种方法是梯度剪裁，这种方法由 Mikolov 首次提出。

在没有进行梯度剪裁时，梯度可能会变得非常大，从而导致模型参数更新不稳定。而进行梯度剪裁后，梯度被限制在一定的范围内，使参数更新更加稳定。

下面是梯度剪裁的伪代码：

Algorithm 1 Pseudo-code for norm clipping

```
[  
    \hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}  
]  
if ( |\hat{g}| \geq \text{threshold} ) 阈值 then  
[  
    \hat{g} \leftarrow \text{threshold} \times \frac{\hat{g}}{|\hat{g}|}  
]  
end if
```

在梯度剪裁中，threshold（阈值）是一个预先设定的标量值，用来限制梯度的最大范数。具体来说，当梯度的范数（即梯度的长度）超过这个预先设定的阈值时，就进行剪裁操作，将梯度的范数调整到这个阈值，从而防止梯度爆炸。

伪代码解释

1. 计算梯度 ( $\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ ), 这里的 ( $\mathcal{E}$ ) 是损失函数, ( $\theta$ ) 是模型参数。
2. 检查梯度的范数 ( $|\hat{g}|$ ) 是否超过阈值 (threshold)。
3. 如果 ( $|\hat{g}| \geq \text{threshold}$ ), 则对梯度进行剪裁:

将梯度的范数重设为阈值, 但保持方向不变。具体操作是 ( $\hat{g} \leftarrow \text{threshold} \times \frac{\hat{g}}{|\hat{g}|}$ )。

1. 这样限制了梯度的最大值，防止了梯度爆炸问题。

这个阈值 (threshold) 通常是根据数据集和模型结构实验调整得到的一个参数。设定一个合适的阈值可以有效地防止梯度爆炸，而不会对梯度的正常更新产生太大的负面影响。

这种方法在递归神经网络 (RNN) 中效果显著，可以大大提高模型的稳定性和训练效果。

**2.4.4.3 处理梯度消失问题的方法：权重初始化与 ReLU 激活函数** 初始化权重 ( $W$ ) 为单位矩阵 ( $I$ ) 并使用 ReLU 激活函数 ( $f(z) = \max(0, z)$ ).

这种方法可以显著改善梯度消失问题。

关于深度学习中的初始化，有多种想法。最早的初始化方法之一是在 Socher 等人 2013 年发表的《解析组合向量语法》中提出的。

图中展示了不同初始化和激活函数对递归神经网络 (RNN) 训练的影响。使用单位矩阵初始化和 ReLU 激活函数显著提高了模型的准确性和收敛速度。

**2.4.4.4 门控递归神经网络 (RNNs) 与长短期记忆 (LSTM)** 长短期记忆 (Long-Short-Term-Memory, LSTM) 算法是在 1997 年由 Hochreiter 和 Schmidhuber 提出的。

近些年对门控递归神经网络 (gated RNNs) 的研究, 提出了门控循环单元 (Gated Recurrent Units, GRU), 时间是 2014 年。

LSTM 在很多应用中被证明非常成功, 比如无限制的手写识别、语音识别、手写生成、机器翻译、图像标注和解析等。

图中展示了 LSTM 和 GRU 的结构, 每种结构都有其独特的机制来处理并记忆长时间跨度的数据序列。

**2.4.4.4.1 长短期记忆网络 (LSTM)** 长短期记忆网络 (LSTM) 通过几个门控机制来有效地管理和保存信息。以下是 LSTM 的主要组成部分及其作用:

遗忘门 (Forget Gate): 控制哪些信息将会被遗忘。

$$\begin{aligned} & [ \\ & f^{\wedge}\{t\} = \sigma(W^f h^{\wedge}\{t-1\} + U^f x^{\wedge}\{t\} + b^f) \\ & ] \end{aligned}$$

输入门 (Input Gate): 控制哪些新信息将会被写入细胞状态。

$$\begin{aligned} & [ \\ & i^{\wedge}\{t\} = \sigma(W^i h^{\wedge}\{t-1\} + U^i x^{\wedge}\{t\} + b^i) \\ & ] \end{aligned}$$

输出门 (Output Gate): 控制当前细胞状态的输出。

$$\begin{aligned} & [ \\ & o^{\wedge}\{t\} = \sigma(W^o h^{\wedge}\{t-1\} + U^o x^{\wedge}\{t\} + b^o) \\ & ] \end{aligned}$$

细胞状态向量 (Cell State Vector): 记忆单元, 用于保存重要的信息。

$$\begin{aligned} & [ \\ & c^{\wedge}\{t\} = f^{\wedge}\{t\} \odot c^{\wedge}\{t-1\} + i^{\wedge}\{t\} \odot \tanh(W^c h^{\wedge}\{t-1\} + U^c x^{\wedge}\{t\} + b^c) \\ & ] \end{aligned}$$

最终的隐藏状态 (Final Hidden State): 基于当前细胞状态的激活值。

$$\begin{aligned} & [ \\ & h^{\wedge}\{t\} = o^{\wedge}\{t\} \odot \tanh(c^{\wedge}\{t\}) \\ & ] \end{aligned}$$

图中展示了 LSTM 的结构, 包括输入门、遗忘门、细胞状态、输出门等。通过这些门控机制, LSTM 可以在长时间序列数据中有效地保存和提取重要信息, 避免梯度消失和梯度爆炸问题。

#### 2.4.4.4.1.1 LSTM: 遗忘门

遗忘门在长短期记忆网络 (LSTM) 中起到选择性遗忘信息的作用。

遗忘门会查看前一时刻的隐藏状态 ( $h^{\wedge}\{t-1\}$ ) 和当前输入 ( $x_t$ ), 然后为细胞状态 ( $C^{\wedge}\{t-1\}$ ) 中的每个数输出一个在 0 和 1 之间的数值。

```
[  
f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)  
]
```

输出的数值为 1 表示“完全保留这个信息”，为 0 表示“完全丢弃这个信息”。

通过这样的机制，LSTM 可以动态调整细胞状态中的信息，保留对当前任务有用的信息，忘记无用的信息，从而有效地处理长时依赖问题。

#### 2.4.4.4.1.2 LSTM: 输入门

在长短期记忆网络 (LSTM) 中，输入门决定了将哪些新的信息存储到细胞状态中。

输入门包含一个 sigmoid 层，称为“输入门层”，它决定了哪些值需要更新。

```
[  
i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)  
]
```

另一个 tanh 层用于创建新的候选细胞状态值向量。

```
[  
\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)  
]
```

首先，sigmoid 层会生成一个介于 0 和 1 之间的值，这些值告诉我们哪些信息是重要的需要更新。然后，tanh 层生成新的候选细胞状态值，这些值将被添加到细胞状态中去。

通过这个过程，输入门控制了新信息的引入，有助于模型在处理长时序列时保持更新信息的有效性。

#### 2.4.4.4.1.3 LSTM: 更新细胞状态

在长短期记忆网络 (LSTM) 中，更新细胞状态是一个关键步骤，它结合了遗忘旧信息和添加新信息。

更新过程包括以下步骤：

1. 使用遗忘门的输出 ( $f_t$ ) 来乘以旧的细胞状态 ( $C_{t-1}$ )，这会遗忘掉之前选择要丢弃的部分信息。

```
[  
C_t = f_t * C_{t-1}  
]
```

1. 将新的候选细胞状态 ( $\tilde{C}_t$ ) 按照输入门的输出 ( $i_t$ ) 进行缩放，然后将其添加到已经遗忘部分信息的旧细胞状态中，形成新的细胞状态。

```
[  
C_t = C_t + i_t * \tilde{C}_t  
]
```

通过这些步骤，LSTM 单元能够选择性地保留重要信息，并在每个时间步更新细胞状态。这种机制使 LSTM 在处理长时间序列数据时能够保持有效记忆，不容易出现梯度消失或梯度爆炸问题。

#### 2.4.4.4.1.4 LSTM：输出门

在长短期记忆网络（LSTM）中，输出门控制了细胞状态中的哪些信息将作为输出。

具体步骤如下：

1. 计算输出门的激活值：通过 sigmoid 层，它决定了细胞状态中哪些部分将输出。

$$[  
o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)  
]$$

1. 生成最终的隐藏状态：将细胞状态 ( $C_t$ ) 通过 tanh 函数进行非线性变换，然后将结果与输出门的激活值 ( $o_t$ ) 相乘，得到最终的隐藏状态 ( $h_t$ )。

$$[  
h_t = o_t \cdot \tanh(C_t)  
]$$

首先，sigmoid 层输出一个介于 0 和 1 之间的数值，这些数值决定了细胞状态的哪些部分需要输出。然后，通过 tanh 函数对细胞状态进行变换，并与 sigmoid 层的输出结果相乘，得到最终的隐藏状态。

通过这个机制，LSTM 能够在每个时间步输出与当前输入和过去状态相关的信息，有效处理长时间序列数据中的长时依赖问题。

#### 2.4.4.4.2 门控循环单元 (GRU, 2014)

门控循环单元 (GRU) 是一种简化版的 LSTM，包含了两个主要的门：更新门和重置门。

更新门 (Update Gate): 决定了过去的状态信息有多少需要传递到当前状态。

$$[  
z_t = \sigma(W^z h_{t-1} + U^z x_t + b^z)  
]$$

重置门 (Reset Gate): 决定了要忘记多少过去的状态信息。

$$[  
r_t = \sigma(W^r h_{t-1} + U^r x_t + b^r)  
]$$

新内存内容计算：

$$[  
\tilde{h}_t = \tanh(W(r_t \odot h_{t-1}) + U x_t + b)  
]$$

最终内存/隐藏状态计算：

$$[  
h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t  
]$$

备注：

当重置门 ( $r_t$ ) 接近 0 时，GRU 会忽略之前的隐藏状态，这允许模型丢弃不相关的信息。更新门 ( $z_t$ ) 控制了过去的状态在当前时刻的重要性。

GRU 通过这些门控机制，能够有效地处理长时间依赖问题，同时相对于 LSTM 相对简单，计算效率更高。

**2.4.4.4.3 LSTM vs GRU** LSTM 和 GRU 是两种常用的递归神经网络 (RNN) 变体，各自有其优缺点和应用场景。

# 比较

1. 性能相似性：

GRU 在多种任务中表现与 LSTM 相近。

1. 训练速度：

GRU 在较少的训练数据上训练速度比 LSTM 快。

1. 长距离关系建模：

在需要建模长距离关系的任务中，LSTM 的表现优于 GRU。

# 结构对比

LSTM：

包含三个门：输入门、遗忘门和输出门。

具有细胞状态，用于在多个时间步间记忆信息。

GRU：

包含两个门：重置门和更新门。

状态更新更加简单，没有显式的细胞状态。

如图所示，LSTM 具有更复杂的门控机制，使其在处理长距离依赖时具有更强的能力，而 GRU 则通过简化的结构实现了更快的训练速度和更少的数据需求。

总结

选择 GRU：当数据量较少且需要较快的训练速度时。

选择 LSTM：当任务要求对长距离关系有较好建模能力时。

这些特点使得 GRU 和 LSTM 在实际应用中各有所长，选择时应根据具体任务需求进行权衡。

## 2.4.5 总结

RNN 能够灵活地对序列数据进行建模。

RNN 还可以扩展，以建模双向依赖，并与注意力机制结合使用。

LSTM 是一种特殊的 RNN，可以用于建模长期依赖。

GRU 是 LSTM 的简化版，运行速度更快且需要较少的训练数据。

这些特性使得递归神经网络 (RNN) 在处理序列数据任务中表现出色，并能够适应不同的要求和场景。LSTM 和 GRU 各自有独特的优势，选择时应依据具体的任务需求进行合理取舍。

## 2.0.6 2.5 总结

下面的表格简要总结了不同模型在若干方面的表现：

[
\begin{array}{ l l l l l l l }
\hline
\text{模型} & \text{长期依赖} & \text{并行化} & \text{计算} & \text{长度适应性} & \text{梯度流动性} & \text{复杂性} \\ \hline
\text{RNN} & \text{较差, 由于递归结构} & \text{有限, 因为是序列处理} & \text{慢, 特别是长序列} & \text{输入灵活, 输出固定} & \text{存在梯度消失/爆炸问题} & \text{简单, 超参数少} \\ \hline
\text{LSTM} & \text{更好, 但仍有改进空间} & \text{有限, 因为是序列处理} & \text{慢, 特别是长序列} & \text{输入灵活, 输出固定} & \text{更好的梯度流动, 较少消失} & \text{复杂, 超参数多} \\ \hline
\text{CNN} & \text{没有内在长依赖} & \text{高度可并行化} & \text{快, 由于可并行计算} & \text{灵活性较小, 通常需要固定大小输入} & \text{梯度流动性稳定} & \text{复杂, 超参数多} \\ \hline
\text{Bahdanau Attention} & \text{与注意力机制配合良好} & \text{有限, 因为是序列处理} & \text{慢, 特别是长序列} & \text{输入和输出都灵活} & \text{更好的梯度流动, 较少消失} & \text{简单, 超参数少} \\ \hline
\text{Transformer} & \text{与注意力机制配合良好} & \text{高度可并行化} & \text{快, 由于可并行计算} & \text{输入和输出都灵活} & \text{梯度流动性稳定} & \text{简单, 超参数少} \\ \hline
\end{array}
]

### 2.5.1 数据总结

1 **RNN**: 擅长处理序列数据, 但在处理长距离依赖时表现较差, 无法高效并行化, 计算速度慢, 梯度消失或爆炸问题突出。不过其结构较简单, 超参数较少。

1 **LSTM**: 在处理长期依赖上表现更好, 但仍需要改进, 并行化能力有限, 计算速度较慢。梯度流动性较好, 但模型结构复杂, 超参数较多。

1 **CNN**: 不擅长处理长距离依赖，但具有高效的并行化能力和较快的计算速度，通常需要固定大小的输入。梯度流动性稳定，但复杂性较高，超参数较多。

1 **Bahdanau Attention**: 与注意力机制配合良好，适应性强，但并行化能力有限，计算速度较慢。梯度流动性较好，模型较为简单，超参数较少。

1 **Transformer**: 与注意力机制配合良好，具有高度的并行化能力和快速计算速度，适应性强，梯度流动性稳定，模型结构较简单，超参数较少。

### 2.5.2 选择指南

- 1 当需要处理长距离依赖且并行化要求高时，**Transformer** 是首选。
- 1 **LSTM** 适用于处理长距离依赖，但计算要求并不高的任务。
- 1 **CNN** 适用于图像处理等不需要长距离依赖但需要高并行化的任务。
- 1 **RNN** 和 **Bahdanau Attention** 适用于序列处理任务，前者简单，后者适应性强。

## 2.1 参考文献