中国科学院大学

University of Chinese Academy of Sciences

系统与计算神经科学

建模实验作业

Spiking Neural Network for Sentiment Analysis

第二十一组：

杨万里，202418013270002

严豫，202418013229063

学院：计算机学院

2024 年 11 月 26 日

目录

1 引言

在生物神经科学中，神经元（Neuron）是大脑中最基本的信息处理单元。神经元通过复杂的结构和电化学信号的传递，支持生物体完成感知、学习、记忆和运动控制等功能。如图1.1(a) 所示，神经元由以下几个关键部分组成：1）**胞体（Soma）**，神经元的核心部分，负责整合所有接收到的输入信号，当信号强度累积达到某个阈值时，胞体会触发一个动作电位（Action Potential），即发放一个脉冲信号；2）**树突（Dendrite）**，分支状结构，连接其他神经元，用于接收输入信号。这些信号以电位变化的形式传递到胞体，为胞体的信号整合提供基础；3）**轴突（Axon）**，将动作电位（即脉冲信号）快速传输至下游的突触；4）**突触（Synapse）**，通过释放神经递质将信号传递给相邻神经元，实现信号的跨细胞通信。

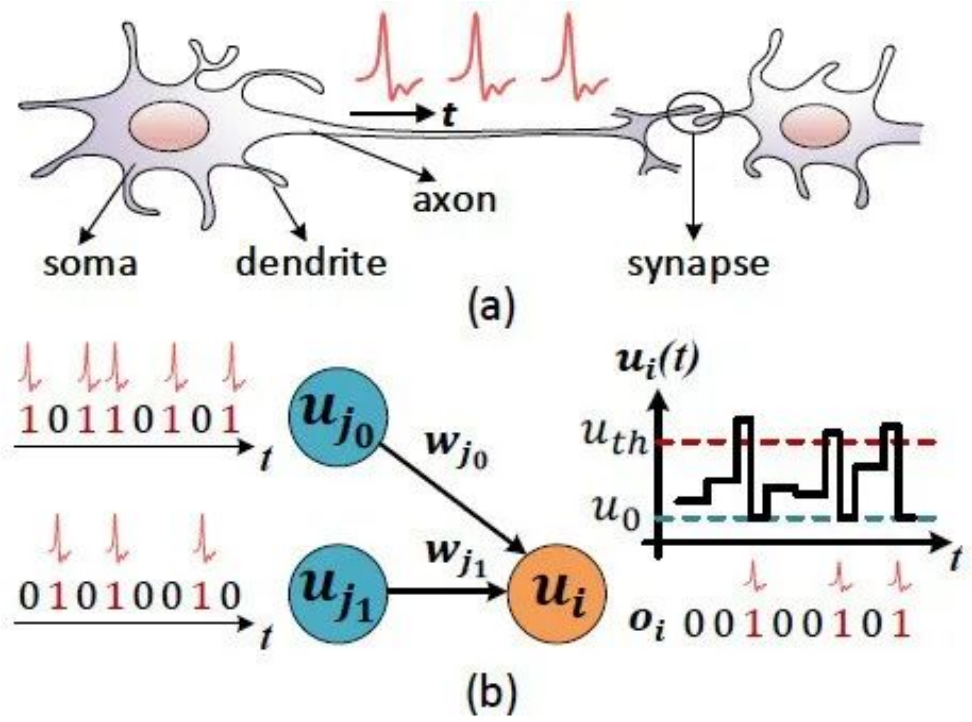

图 1.1: 生物神经元和脉冲神经网络（SNNs）运行机制示意图：(a) 描述了生物神经元信号传递的基本结构，包括树突、胞体（soma）、轴突（axon）及突触（synapse）；(b) 展示了 SNN 中基于脉冲信号的计算过程，其中输入信号通过权重 w 累积到膜电位 $u_i(t)$，当达到阈值 u_{th} 时触发脉冲输出 o_i

神经元之间的信号传递依赖于突触（Synapse）释放的化学物质（神经递质）。这种传递过程是离散的，表现为通过脉冲编码信息。如图1.1(b) 所示，通过将信号的强度被编码为脉冲的频率，而非幅度，可以实现**脉冲编码（Spike Encoding）**的编码方式。这种脉冲编码机制不仅体现了生物神经系统的高效性与鲁棒性，也为构建仿生计算系统提供了灵感。近年来，基于此编码方式的**脉冲神经网络（Spiking Neural Networks, SNNs）**[7] 逐渐成为研究热点，因其基于脉冲驱动的稀疏激活方式和时间动态建模能力，展现出显著的能耗优势和计算潜力。

在自然语言处理（Natural Language Processing, NLP）任务中，深度神经网络（Deep Neural Networks, DNNs）已成为解决文本分类、情感分析等任务的主流方法。然而，DNN 模型的高计算资源需求和高能耗问题阻碍了其在资源受限场景中的广泛应用。例如，大型语言模型 GPT-3 的训练消耗高达 190,000 kWh [2, 1]，而人类大脑仅通过 20W 功率能完成感知、推理等复杂任务。这种能效差异激发了对更节能计算范式的研究，SNN 因其稀疏激活和事件驱动的特性，成为了一种潜在解决方案。

然而，尽管脉冲神经网络（SNNs）提供了一条有前景的实现人工智能模型的途径，但是鲜有工作

表明 SNNs 在语言任务中存在有效性，主要原因在于**语言数据的离散性和变长特性的复杂性**。一方面，语言数据通常以离散的单词或句子形式出现，将其转换为适合 SNN 处理的脉冲信号并非易事；另一方面是 SNN 天然擅长处理固定时序输入，而文本的变长特性要求模型具备更强的适应能力和灵活性。这两点问题通常导致了 SNN 模型的训练效果远不如 DNN 网络。

近年来，针对语言数据的特性，研究者 [9, 6] 提出基于“转换 + 微调”范式进行 SNN 模型训练，即通过先使用传统深度神经网络进行预训练，然后将其架构和权重转换为脉冲神经网络，并在目标任务上进一步进行微调优化。这种方法以解决 SNN 在语言任务中面临的两个核心问题为目标：离散语言数据的脉冲化编码和变长输入的适应性。具体而言，“转换 + 微调”范式包含以下关键步骤：

- **DNN 模型的预训练**。构建一个定制化的 DNN 模型（如卷积神经网络 TextCNN），通过调整架构以适配后续的 SNN 转换。例如，将最大池化操作替换为平均池化、使用 ReLU 激活函数代替 Sigmoid 以避免负值，并将词嵌入（如 GloVe）转换为正值向量。这一步确保 DNN 能够高效学习文本特征。
- **模型的 SNN 转换**。在完成 DNN 的训练后，模型的权重和架构被映射到 SNN 中。此时，词嵌入中的向量通过特定的脉冲编码方式（如泊松编码或时间编码）转化为脉冲信号，适配 SNN 的输入需求。同时，膜电位的动态建模机制用于支持时间依赖特性的学习。
- **代理梯度的微调优化**。为弥补转换过程中可能导致的性能下降，使用代理梯度法对 SNN 进行微调。代理梯度法通过近似优化离散脉冲信号的梯度，使得 SNN 能够更高效地学习目标任务中的复杂特征，从而实现性能提升。

针对以上“转换 + 微调”的自然语言任务 SNN 训练方案，本实验拟采用一经典的自然语言任务**SNN 在情感分析任务**进行 SNN 实际性能表现的探索和分析。具体而言，本次实验我们拟针对以下问题进行研究：

- 在情感分析任务中，ANN 模型中的典型方法 CNN 的分类性能如何?
- 在情感分析任务中，SNN 的分类性能是否能够接近甚至超越 ANN 模型（CNN）的水平?
- “转换 + 微调”范式是否能够有效提升 SNN 在情感分析任务中的性能?
- SNN 是否能够与其他 DNN 模型协同工作，以进一步提升性能?

2 背景

在正式开始实验探索之前，我们在本节介绍本项目所涉及的相关背景知识，主要包括：目标任务（情感分类）及其研究情况介绍；CNN（卷积神经网络）在情感分类任务中的应用；SNN（脉冲神经网络）基本概念。

2.1 情感分类

情感分类是自然语言处理（NLP）中的一项重要任务，旨在识别和提取文本中的主观信息，如作者的情绪、观点、情感等。这个任务通常涉及到将文本分类为正面、负面或中性等情感类别。例如，产品评论可以被分类为正面、负面或中性，这对于产品改进和客户满意度分析等方面非常有用。情感分类可以应用于许多领域，包括社交媒体监控、品牌管理、客户关系管理等。其中的挑战包括理解语言

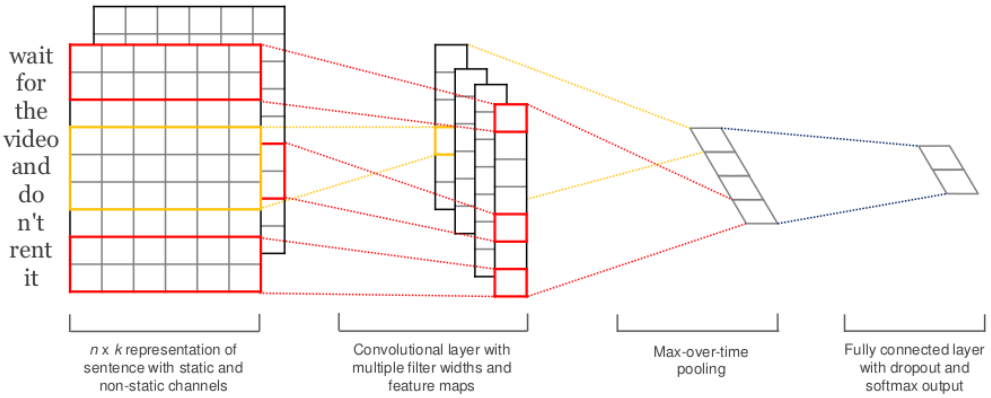

图 2.2: TextCNN 模型架构图

的微妙和复杂性，例如讽刺、双关语和情绪的强度等。此外，处理不同语言、方言和领域特定的语言也是情感分类中的一大挑战。

NLP 研究早期主要基于规则和词典的方法完成情感分类任务，但这些方法在处理复杂和微妙的情绪表达时面临困难。随着机器学习和深度学习技术的发展，情感分类任务被处理得越来越好。特别是深度学习模型，如卷积神经网络（CNN）和循环神经网络（RNN），已经被成功应用于情感分类任务，显著提高了分类的准确性。当 BERT（Bidirectional Encoder Representations from Transformers）等基于 Transformer 架构的模型诞生以后，它们极强的语义特征提取能力使得情感分类问题已经能被较好地解决。到 ChatGPT 等大语言模型出现后，情感分类任务已经被几乎完美地解决了。

但是基于 Transformer 的模型，特别是 ChatGPT 等超大语言模型的参数量极为庞大，其涉及到的资源消耗带来了极高的成本。**因此本次作业中，我们希望探索低能耗、更类人的脉冲神经网络（SNN）在情感分类任务当中的可用性**。

2.2 CNN 处理情感分类

卷积神经网络（CNN）在计算机视觉领域取得了显著的成果。而文本数据通常以一维的序列形式呈现。但这并不代表 CNN 无法进行有效的自然语言处理，相反，CNN 在自然语言处理，特别是本文所研究的情感分类任务上有一定潜力。

其实，文本也可以被视为一维图像，从而可以用一维卷积神经网络来捕捉临近词之间的关联。这其中的代表作就是 TextCNN [5]。为了将 CNN 应用到文本分类任务，TextCNN 利用多个不同 size 的 kernel 来提取句子中的关键信息（类似于多窗口大小的 n-gram），从而能够更好地捕捉局部相关性。与传统图像的 CNN 网络相比，textCNN 在网络结构上没有任何变化（甚至更加简单了）。其基本架构如图2.2所示，可以看出 textCNN 其实只有一层卷积，一层 max-pooling，最后将输出外接 softmax 来 n 分类。

2.3 脉冲神经网络

2.3.1 背景

神经网络被认为是当前人工智能发展的主要驱动力，其经历了几个发展阶段：**第一个阶段是感知机**，1957 年美国神经学家 Frank Rosenblatt 提出可以模拟人类感知能力的机器，并称之为“感知机”。**第二个阶段是**基于联结主义的**多层人工神经网络** (Artificial Neural Network, **ANN**)，其兴起于二十世

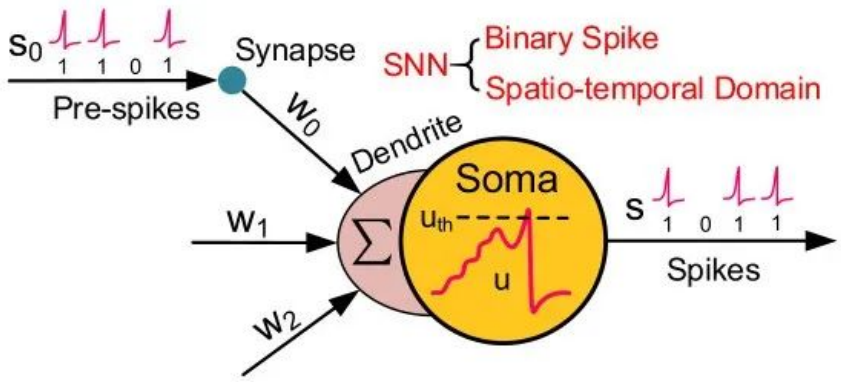

图 2.3: SNN 神经元

纪 80 年代中期，并在 2006 年以后以深度卷积网络引领了近十几年人工智能的发展。然而，ANN 在生物学上是不精确的，**缺少神经内部的动力学机制**，不能较准确地模仿生物大脑神经元的运作机制，并且现在由 scaling law 催生的大模型的范式**需要巨大的资源成本**，这与人脑的工作有显著的差异。近年来起源于脑科学的**脉冲神经网络** (Spiking Neural Network, **SNN**) [3] 被誉**为新一代的神经网络**，以其丰富的时空领域的神经动力学特性、多样的编码机制、事件驱动的优势引起了学者的关注。

2.3.2 神经元工作原理

神经元的典型结构主要包括**树突**、**胞体**以及**轴突**三个部分，其中树突的功能是收集来自其他神经元的输入信号并将其传递给胞体，胞体起到中央处理器的作用，当接受的传入电流积累导致神经元膜电位超过一定阈值时产生神经脉冲（即动作电位），脉冲沿轴突无衰减地传播并通过位于轴突末端的突触结构将信号传递给下一个神经元。

2.3.3 LIF 模型

针对神经元工作时电位的动态特性，神经生理学家建立了许多模型，它们是构成**脉冲神经网络的基本单元**，决定了网络的基础动力学特性。其中影响较大的主要有 H-H 模型，**LIF 模型**，Izhikevich 模型和脉冲响应 SRM 模型等（**本次实验以 LIF 模型为基本单位**）。

早在 1907 年 Lapicque 就提出了 **Integrate-and-fire**（**I&F**）模型，由于当时对动作电位的产生机理知之甚少，动作电位的过程被简化描述为：“当膜电位达到阈值 Vth 时神经元将激发脉冲，而膜电位回落至静息值 Vreset” 模型则针对描述阈下电位的变化规律，其中最为简单且常见的是 LIF 模型：$\tau_m \frac{dV}{dt} = V_{rest} - V + R_m I$。其中 m 表示膜时间常数，Vrest 表示静息电位，Rm 表示细胞膜的阻抗,I 表示输入电流。LIF 模型极大简化了动作电位过程，但**保留了实际神经元膜电位的泄露、积累以及阈值激发这三个关键特征**。典型的脉冲神经网络神经元如图2.3所示。

2.3.4 基本结构

如图2.4所示，SNN 的基本结构包括**输入层**、**隐层**和**输出层**，与其他神经网络结构类似。每一层都包含一个或多个神经元。神经元之间通过突触进行连接，并且每个突触都有一个权重，用于调整通过它的信号强度。

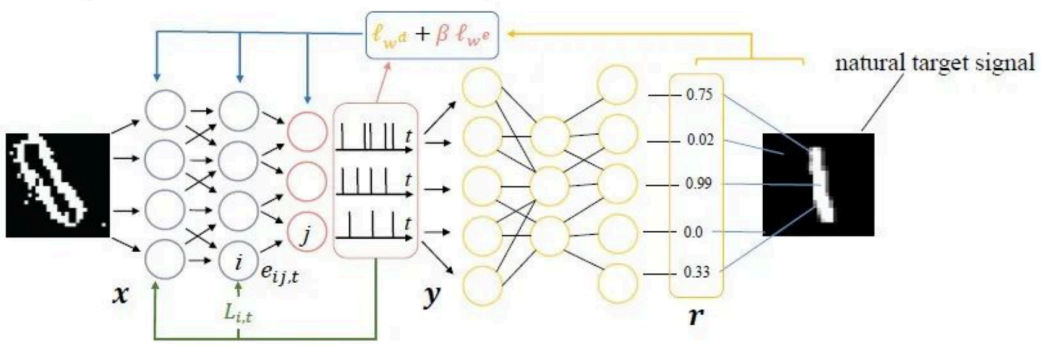

图 2.4: SNN 架构图示例

2.3.5 “转换 + 微调”训练范式

“转换 + 微调”（Conversion + Fine-tuning）是一种有效的两阶段训练策略，旨在解决脉冲神经网络在语言任务中面临的两大核心问题：离散语言数据的脉冲化编码和变长文本的适配性。本实验参考的方法 [6] **先通过训练一个传统的深度神经网络，然后将其架构和权重映射到 SNN，并在目标任务上进一步优化**，以兼顾高性能与低能耗。

转换阶段 一个定制化的 DNN 模型被设计为适配 SNN 的特性，主要包括以下调整：

- 词嵌入正值化：传统词嵌入（如 GloVe）中的值通常存在负数，需通过标准化和位移将其映射到正值区间（0 和 1），以确保 SNN 的输入信号全为非负值。
- 激活函数替换：将 DNN 中的激活函数（Sigmoid 函数）替换为 ReLU，以避免负值输出的产生。
- 池化方式调整：将最大池化操作替换为平均池化，这种调整能够更好地适应 SNN 中基于脉冲的计算。
- 权重映射：将训练好的 DNN 权重直接转换为 SNN 的突触权重，同时在 SNN 中引入“泊松编码”或“时间编码”方法，将词嵌入值转化为脉冲信号输入。

通过上述调整，DNN 被成功转换为具有时间动态特性的 SNN。

微调阶段 在完成模型转换后，使用代理梯度法（Surrogate Gradient）对 SNN 进行微调。代理梯度法通过近似脉冲信号的离散梯度，解决了 SNN 中因非连续脉冲激发而导致的不可微性问题。具体优化过程包括：

- 生成脉冲信号：针对训练数据中的每个样本，生成基于词嵌入的泊松脉冲信号序列。
- 时间步优化：通过反向传播算法在多个时间步内调整权重，优化模型的时间动态特性。
- 损失函数：使用交叉熵损失（Cross-Entropy Loss）结合时序误差（Spike Rate Error）对模型的预测能力进行优化。

3 实验

3.1 基本设定

本文针对情感分类任务选择其中最具代表性的数据集 **SST2** [8] 开展实验探索。本次实验探索了四种模型架构的有效性，分别为：单纯的卷积神经网络（**CNN**）、脉冲神经网络（**SNN**，基于 LSTM 的特征提取）、基于 **CNN 转化得到的 SNN**、以及 **CNN 与 SNN 的结合**。

本次实验探索的**基本超参数在不同模型中均保持统一**，如 learning_rate=1e-4，epoch=50 等，以保证模型之间的公平对比。

3.2 数据预处理

在自然语言处理中，通常会将文本数据转化为**预训练得到的 word embedding**，再输入模型中。这些 word embedding 在预训练阶段充分学习到了对应词语的语义特征，能极大程度提高模型对文本的理解。对于正常的 CNN、RNN 模型来说，预训练的 word embedding 可以直接输入模型。但是，**SNN 对于要求输入数据必须是非负值**。为了保留 embedding 的语义特征同时符合 SNN 的输入要求，我们参考 [6] 的做法，首先计算预训练 word embedding 的平均值 μ 和标准差 σ，将所有值映射到 $[\mu-3\sigma, \mu+3\sigma]$ 的范围内，然后通过减去 μ 并除以 $6\times\sigma$ 来进行标准化，最后将向量的所有分量移动到 $[0,1]$ 的范围内。

3.3 CNN

为探究 CNN 在情感分析任务中的表现，本实验在 SST2 数据集上构建 CNN 模型进行实验。

3.3.1 模型实现

在情感分析任务中，卷积神经网络 CNN 作为人工神经网络的一种典型方法，凭借其高效的特征提取能力和快速收敛的训练过程，成为广泛应用的主流模型之一。本实验采用了经典的 TextCNN 模型，通过多个卷积核提取文本的局部上下文信息，并通过池化层对特征进行降维和选择，以增强模型的泛化能力。模型的关键模块主要包含卷积层、池化层和全连接层。

CNN 模型代码

```
class Normal_TextCNN(nn.Module):
    def __init__(self, args) -> None:
        super().__init__()
        self.convs_1 = nn.ModuleList([
            nn.Conv2d(in_channels=1, out_channels=args.filter_num,
                kernel_size=(filter_size, args.hidden_dim))
            for filter_size in args.filters
        ])
        self.middle_relu = nn.ModuleList([
            nn.ReLU()
            for _ in args.filters
        ])
        self.maxpool_1 = nn.ModuleList([
            nn.MaxPool2d((args.sentence_length - filter_size + 1, 1)) for filter_size
                in args.filters
```

```
        ])
        self.relu_2 = nn.ReLU()
        self.drop = nn.Dropout(p=args.dropout_p)
        self.fc_1 = nn.Linear(len(args.filters)*args.filter_num, args.label_num)

    def forward(self, x):
        x = x.float()
        batch_size = x.shape[0]
        x = x.unsqueeze(dim=1)
        conv_out = [conv(x) for conv in self.convs_1]
        conv_out = [self.middle_relu[i](conv_out[i]) for i in
            range(len(self.middle_relu))]
        pooled_out = [self.maxpool_1[i](conv_out[i]) for i in
            range(len(self.maxpool_1))]
        pooled_out = [self.relu_2(pool) for pool in pooled_out]
        flatten = torch.cat(pooled_out, dim=1).view(batch_size, -1)
        flatten = self.drop(flatten)
        fc_output = self.fc_1(flatten)
        return fc_output
```

3.3.2 模型训练

在 SST2 数据集上进行训练时，TextCNN 模型表现出显著的学习效率和性能优势。如图3.6所示，训练过程中模型的分类正确率在前 10 个 epoch 内迅速上升，最终收敛到高水平，并最终在 18 个 epoch 达到最佳性能（82.5%），表明 CNN 能够快速有效地学习文本情感特征。

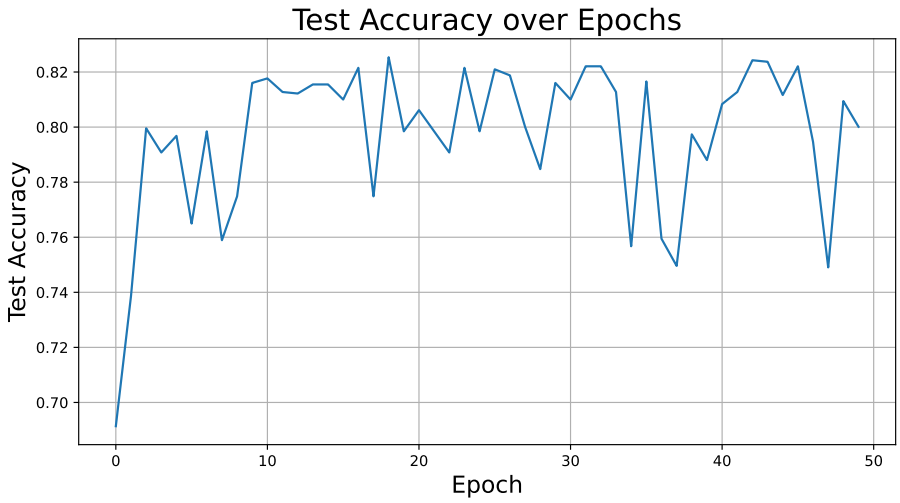

图 3.5: 训练过程中分类正确率的变化图

模型的训练过程和结果还体现了以下特性：

- 快速收敛：TextCNN 在少量训练轮次内即可达到较高的分类准确率，显示出其对情感特征的高效捕获能力。
- 鲁棒性较强：在多次实验中，TextCNN 对输入数据的变化表现出稳定的性能，证明其在情感分

类任务中的适用性。

以上训练实验结果表明，传统 DNN 模型在处理情感分析任务时具有较强的学习效率。CNN 的训练时间成本低，在数据集 SST2 上训练 50 个 epoch 仅需约 10 分钟。

3.3.3 小结

综合来看，CNN 在情感分析任务中的分类性能表现出色，具有学习效率高、泛化能力强、适配性好的特性。然而，CNN 在高能耗场景中可能面临一定的局限性，尤其是在追求节能和实时计算的应用中。相比之下，SNN 作为一种低能耗模型，提供了潜在的替代方案，其性能与能耗之间的平衡将是下一节讨论的重点。

3.4 SNN

为探究 SNN 在情感分析任务中的表现，本实验在 SST2 数据集上构建 SNN 模型进行实验。

3.4.1 模型实现

CNN 和 RNN 等人工神经网络已经在情感分类任务当中展现其有效性，脉冲神经网络虽以更接近人脑的计算模式以及低能耗备受关注，但是我们却不知道其是否能有效处理文本数据。文本数据本质上也是一种序列数据，从这个角度来说，它和脉冲序列是有一定相似性的。但是，直接用 SNN 处理文本数据很难有效处理文本语义信息。作为中和，我们这部分尝试用一个小的双向 LSTM 网络 [4] 初步处理文本数据，再通过交替使用 SNN 中的 LIF 单元和全连接层，最终输出预测结果。模型的代码如下所示。**我们希望观察是否能通过 BiLSTM+SNN 的架构，即保留对文本语义信息的处理，又能发挥 SNN 模拟人脑中脉冲传递信息的模式优势。**

LSTM+SNN 模型代码

```
class SNN_BiLSTM(nn.Module):
    def __init__(self, args, spike_grad=surrogate.fast_sigmoid(slope=25)) -> None:
        super().__init__()

        self.lstm = nn.LSTM()
        if args.bidirectional == "True":
            self.fc_1 = nn.Linear(args.lstm_hidden_size * 2, args.lstm_fc1_num)
        else:
            self.fc_1 = nn.Linear(args.lstm_hidden_size, args.lstm_fc1_nume)

        self.lif1 = snn.Leaky()
        self.output_fc = nn.Linear()
        self.lif2 = snn.Leaky()

    def forward(self, x):
        x = x.float()
        batch_size = x.shape[0]
        output, (hidden, cell) = self.lstm(x)
        x = self.fc_1(output)
        spks = self.lif1(x)
        fc_output = self.output_fc(spks)
```

```
        fc_output = fc_output[:, -1, :].squeeze(1)
        spk2, mem2 = self.lif2(fc_output)
        return spks, spk2, mem2
```

3.4.2 模型训练

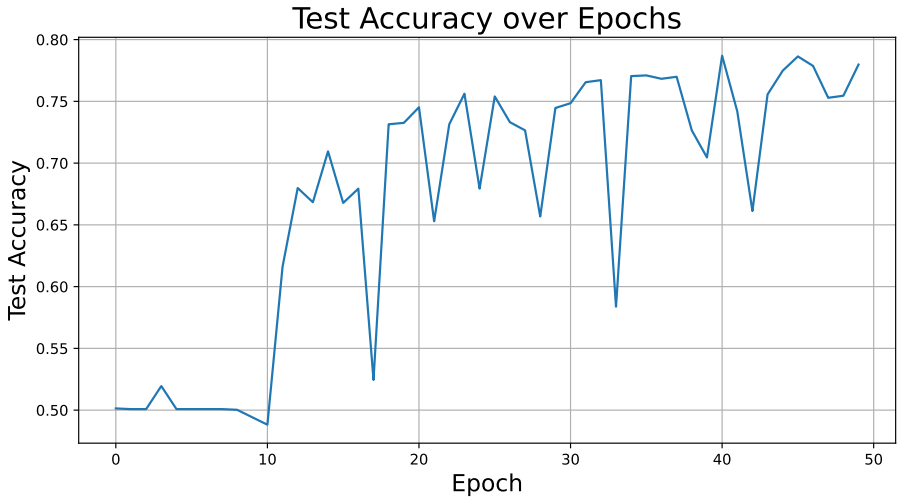

图 3.6: 训练过程中分类正确率的变化图

BiLSTM+SNN 模型训练过程中分类正确率的变化情况如图3.6所示。在训练过程中，我发现 SNN 模型的训练和传统 ANN 模型的训练有两个显著的不同：

- **训练时间明显更长**：针对 SST2 的全部训练数据，在 A100 上对 SNN 模型训练 50 个 epoch（单卡上只跑一个 SNN 模型训练进程），需要 9.5 小时，作为对比，相同条件下 CNN 模型的训练只需要 10 分钟，**也就是说 SNN 模型训练的时间成本是 CNN 模型的 50 余倍**。不过从理论上，SNN 模型的训练确实会比 CNN 模型更慢，影响因素有：
  - **信息处理方式**：CNN 处理的是**连续的实数值数据**，而 SNN 处理的是二**进制脉冲**，这导致 SNN 的计算复杂性更高。SNN 需要模拟神经元的动态行为，这通常涉及复杂的数学模型，如积分-发放（Integrate-and-Fire）机制。
  - **事件驱动计算**：SNN 的事件驱动特性意味着计算是**异步和离散**的，只有在脉冲到达时才进行更新。这种异步计算虽然在推理时能**节省能量**，但在训练过程中通常需要更复杂的机制来处理时间上的动态性。
  - **复杂的训练过程**：SNN 的训练通常使用基于事件的学习规则，如 Spike-Timing-Dependent Plasticity (STDP)，或者通过间接方法如代理梯度（surrogate gradient），**本次实验采取的是代理梯度**。这些方法通常比标准的反向传播算法复杂，且可能需要更长的时间来收敛。
  - **硬件限制**：目前，大多数深度学习硬件和框架，如 GPU 和 TensorFlow、PyTorch 等，是为连续值的神经网络优化的。**这意味着在这些平台上模拟 SNN 可能不是最有效的**。

  以上因素共同作用造成了极大的训练时间成本，这也是节省能耗的一个代价。

- **“冷启动”问题**：传统的 ANN 模型，通常在训练开始的几个 epoch 会有明显的 loss 下降和准确率升高，随着训练过程推进开始逐渐收敛，增益越来越低。但是**在 SNN 模型中，前 10 个 epoch，模型的性能几乎没有任何改进，始终接近随机猜测状态**，这在 ANN 模型中是比较少见的。直到第 10 个 epoch 之后，模型才开始了真正的学习过程。可能的原因是
  - **脉冲活动的稀疏性**：在训练初期，网络中的脉冲活动可能较为稀疏，导致**网络难以有效学习输入特征**。随着训练的进行，脉冲活动逐渐增加或调整，学习效果才会显现。
  - **时间动态性**：SNN 需要在多个时间步中聚合信息，初期**可能需要多个 epoch 来适应输入数据的时间动态特征**，才能有效调整权重。

3.4.3 小结

从本小节的实验探索中可以看出，虽然 SNN 具有低能耗、更接近人脑的计算模式的优势，但是也需要明显更多的训练时间，并且最终最佳分类准确率为 **78.7%**，无法超越前文 CNN 的 **82.5%**。这说明 SNN 的发展和研究还有比较长的路要走。但是从中我们也可以获得很多有益的观察和发现。

3.5 “转换 + 微调” SNN

为探究“转换 + 微调”范式是否能够有效提升 SNN 在情感分析任务中的性能，我们将 CNN 转换为 SNN，并进行性能对比实验。

3.5.1 模型实现

本实验基于“转换 + 微调”范式构建了一个 SNN 模型，该模型在预训练的 TextCNN 基础上进行转换，并结合脉冲神经网络的特性进行了优化处理。本实验的模型包含以下几个关键部分：

- **卷积层的脉冲化：**使用多层卷积核提取文本特征，并将提取后的特征通过 Leaky Integrate-and-Fire（LIF）单元转化为脉冲信号。相比传统的激活函数，LIF 单元能够模拟生物神经元的发放机制，并支持时间动态建模。
- **池化与特征整合：**替换传统 TextCNN 中的最大池化（Max Pooling）为平均池化（Avg Pooling），确保特征选择适配脉冲信号的特性。同时，池化后的特征通过额外的 LIF 单元进一步处理，以生成稀疏的脉冲信号。
- **分类器与输出层：**使用全连接层和 LIF 单元构成分类器，模型的最终输出通过脉冲信号表征，结合代理梯度优化实现分类任务的准确预测。

“转换 + 微调” SNN 模型代码

```
class SNN_TextCNN(nn.Module):
    def __init__(self, args, spike_grad=surrogate.fast_sigmoid(slope=25)) -> None:
        super().__init__()
        self.dead_neuron_checker = args.dead_neuron_checker
        self.initial_method = args.initial_method
        self.positive_init_rate = args.positive_init_rate
        self.convs_1 = nn.ModuleList([
            nn.Conv2d(in_channels=1, out_channels=args.filter_num,
                kernel_size=(filter_size, args.hidden_dim))
```

```
            for filter_size in args.filters])
        self.middle_lifs = nn.ModuleList([
            snn.Leaky(beta=args.beta, spike_grad = spike_grad, init_hidden=True,
                threshold=args.threshold)
            for _ in args.filters])
        self.avgpool_1 = nn.ModuleList([
            nn.AvgPool2d((args.sentence_length - filter_size + 1, 1)) for filter_size
                in args.filters])
        self.lif1 = snn.Leaky(beta=args.beta, spike_grad=spike_grad,
            init_hidden=True, threshold=args.threshold)
        self.fc_1 = nn.Linear(len(args.filters)*args.filter_num, args.label_num)
        self.lif2 = snn.Leaky(beta=args.beta, spike_grad=spike_grad,
            init_hidden=True, threshold=args.threshold, output=True)

    def initial(self):
        for c in self.convs_1:
            c.weight.data.add_(INITIAL_MEAN_DICT['conv-kaiming']
            [self.positive_init_rate])
        m = self.fc_1
        m.weight.data.add_(INITIAL_MEAN_DICT["linear-kaiming"]
        [self.positive_init_rate])

    def forward(self, x):
        batch_size = x.shape[0]
        x = x.unsqueeze(dim=1)
        conv_out = [conv(x) for conv in self.convs_1]
        conv_out = [self.middle_lifs[i](conv_out[i]) for i in
            range(len(self.middle_lifs))]
        pooled_out = [self.avgpool_1[i](conv_out[i]) for i in
            range(len(self.avgpool_1))]
        spks = [self.lif1(pooled) for pooled in pooled_out]
        spks_1 = torch.cat(spks, dim=1).view(batch_size, -1)
        hidden_1 = self.fc_1(spks_1)
        # cur2 = self.fc_2(hidden_1)
        spk2, mem2 = self.lif2(hidden_1)
        if self.dead_neuron_checker == "True":
            temp_spks = spks_1.sum(dim=0)
            Monitor.add_monitor(temp_spks, 0)
        return spks_1, spk2, mem2
```

3.5.2 模型训练

训练过程中，“转换 + 微调” SNN 的分类正确率变化如图3.7所示。从图中可以看出，SNN 模型在训练初期同样表现出 **“冷启动” 现象**：前几个 epoch 内，模型的分类正确率提升较慢，甚至接近随机水平，最终在 41epoch 的时候，达到最佳分类性能（78.6%）。这一现象与以下因素相关：

- **脉冲信号的稀疏性：**在训练初期，神经元的发放频率较低，导致特征学习较为缓慢。
- **时间动态适应：**SNN 需要一定的训练时间来适应输入数据的时序特性，逐步调整权重以捕获语

义信息。

在经过一定的训练轮次后，模型的分类正确率开始大幅上升，并在后期逐渐趋于稳定。此外，与传统 CNN 相比，SNN 模型的训练时间明显更长，但在引入“转换 + 微调”SNN 的训练方法后，模型的训练时间变短，从 9.5 小时降低为大概为 6 小时。并且能取得比 SNN 更好的性能（79.8%）。

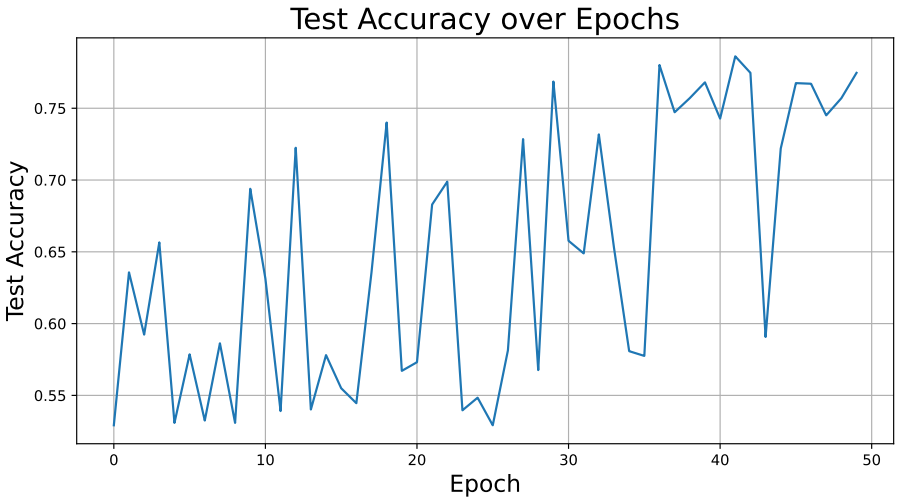

图 3.7: 训练过程中分类正确率的变化图

3.5.3 小结

通过“转换 + 微调”方法，本实验进一步验证了 SNN 在情感分析任务中的可行性。一方面，该方法显著缩短了 SNN 的训练时间，从传统 SNN 模型的 9.5 小时降低至约 6 小时；另一方面，分类性能达到了 79.8% 的最佳准确率。实验结果还表明，“转换 + 微调”方法在以下几个方面具有显著优势：

- **解决冷启动问题：**通过引入预训练的 DNN 模型作为权重初始化，“转换 + 微调”方法有效地缓解了 SNN 训练初期的“冷启动”问题，加快了模型性能的提升。
- **优化训练效率：**在传统 SNN 训练时间较长的情况下，“转换 + 微调”方法通过预训练与权重映射显著减少了训练时间，为实际应用提供了更高的效率。
- **性能与能耗的权衡：**尽管分类性能略低于 CNN，SNN 的能耗优势使其在特定场景中具有更高的实用价值，尤其是在需要节能计算的应用场景中。

总体而言，本次实验证明了“转换 + 微调”方法的有效性。

3.6 CNN + SNN

在前文，我们分别对 CNN 和 SNN 用于情感分类任务进行了独立的探索，也体现了它们各自的优势。对于两个具有不同优势的模型，一个常见的想法是：能不能将二者结合，取长补短，同时发挥二者的优势？对此，本小节展开对于 SNN 和 ANN 协同工作的初步探索。

3.6.1 模型实现

对于两个不同模型的合并，一个常见的方式是，二者各自进行特征的提取和处理，得到处理好的特征之后进行拼接，再通过一个全连接神经网络转换为输出结果。这样做的好处是二者不会互相影响，可以根据模型自身的架构特性提取到不同的特征信息，并且可能互为补充，充分发挥二者各自的优势。

需要注意的是，我们在前文中已经训练得到了最佳的 CNN 和 SNN 模型，因此在合并模型中，我们直接加载这两个最优模型的参数，并且在训练过程中冻结这部分参数。这样做的好处是：

- **避免训练模式冲突**：CNN 和 SNN 模型的训练模式有显著差异，很难在同一个过程中兼容。加载预训练模型的参数并在训练过程中冻结，可以将训练过程集中在优化全连接网络上。
- **节省时间**：模型的训练过程，尤其是 SNN 的训练过程需要极大的时间成本。直接采用预训练模型可以避免这部分成本。

模型的实现代码如下所示。

CNN+SNN 模型代码

```python
class CombinedModel(nn.Module):
    def __init__(self, snn_lstm_model, cnn_model, args):
        super(CombinedModel, self).__init__()
        self.snn_lstm = snn_lstm_model
        self.cnn = cnn_model
        # 全连接层
        fc_input_dim = 2 * args.label_num
        self.fc_combined = nn.Linear(fc_input_dim, args.label_num)

    def forward(self, x):
        with torch.no_grad():
            # 并行，分别通过snn_lstm和cnn处理
            spks, spk2, mem2 = self.snn_lstm(x)
            cnn_output = self.cnn(x)
        combined_out = torch.cat((mem2, cnn_output), dim=1)
        # 通过额外的全连接层
        output = self.fc_combined(combined_out)
        return output
```

3.6.2 模型训练

CNN+SNN 模型训练过程中分类正确率的变化情况如图3.8所示。在前 2 个 epoch，模型的表现还是接近随机猜测的状态，但是在第 3 个 epoch，模型的性能显著提升（正确率 74.1%），并且在第 4 个 epoch 就达到了**最高值（正确率 81.3%）**，这也体现出基于预训练模型范式的优越性。同时因为只需要优化一个全连接层，整个过程的训练时间只需要五分钟。

同时，最终的结果并不像我们所期待的，CNN 和 SNN 模型各自提取到不同的特征，同时发挥二者的优越性取得更好的效果。实际上，合并模型并没有带来收益，模型最佳的正确率 **81.3%** 甚至不如单独使用 CNN 模型达到的 **82.5%**，不过也并没有明显的下降。使用 SNN 模型虽然没有带来性能的增益，但是可能会有其他的好处，比如 [6] 中提到，**SNN 模型对于文本扰动和攻击的鲁棒性更强**。不过本次实验的重点不在此，所以没有对这方面做进一步测试。

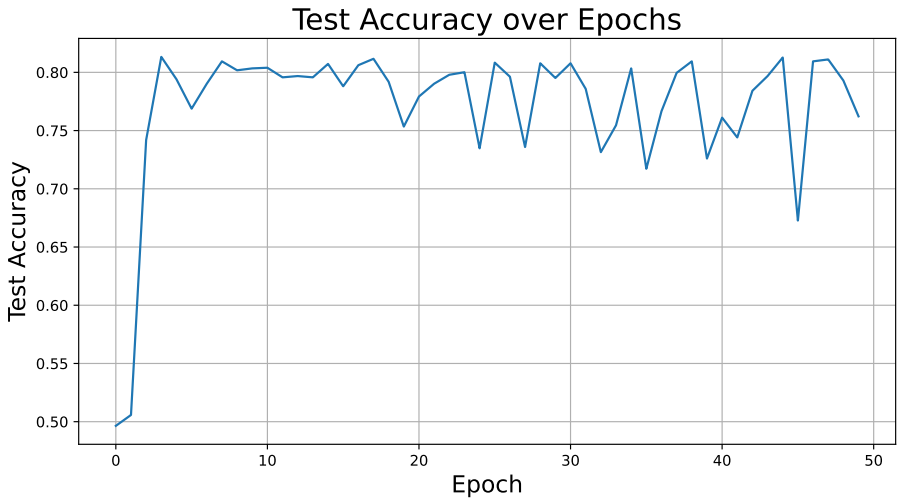

图 3.8: 训练过程中分类正确率的变化图

3.6.3 小结

本小节主要对 CNN 和 SNN 的混合模型进行了探索，寄希望于混合模型能取 CNN 和 SNN 各自所长，获得更好的情感分类表现。但是最终却没有取得理想的效果，不过作为一个尝试还是有一定的收获。

4 结论

脉冲神经网络 SNN 是一种基于脉冲驱动的稀疏激活方式和时间动态建模的神经网络模型，相比于人工神经网络，展现出显著的能耗优势和计算潜力。本实验验证了 SNN 和 ANN 相结合的方法在情感分析任务中应用的有效性，并从分类性能、训练效率等方面展示了 SNN 的潜力。尽管 SNN 在分类性能上略低于 CNN，但其独特的低能耗特性和对抗性鲁棒性为其在实际应用中的推广提供了新的可能性。同时，实验也揭示了 SNN 模型训练中的瓶颈，如训练时间成本高和对文本特征的学习能力需进一步优化。在未来的研究中，SNN 与 DNN 的协同工作、更加高效的训练方法，以及对复杂文本数据的适配性将是值得深入探索的方向。

5 分工

本次对脉冲神经网络（SNN）在文本情感分类任务中的有效性的实验探索中：

- **严豫**：负责直接使用 CNN 和“转换 + 微调”SNN 部分；
- **杨万里**：负责直接使用 SNN 和 CNN + SNN 部分。

在实验报告撰写中：

- **严豫**：负责引言，实验中的 CNN 和“转换 + 微调”SNN，以及结论部分的撰写；
- **杨万里**：负责背景，实验中的基本设定、数据预处理、SNN 和 CNN+SNN 部分的撰写。

参考文献

[1] Anthony, L.F.W., Kanding, B., Selvan, R.: Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. arXiv preprint arXiv:2007.03051 (2020)

[2] Dhar, P.: The carbon impact of artificial intelligence. Nat. Mach. Intell. **2**(8), 423–425 (2020)

[3] Ghosh-Dastidar, S., Adeli, H.: Spiking neural networks. International journal of neural systems **19**(04), 295–308 (2009)

[4] Graves, A., Graves, A.: Long short-term memory. Supervised sequence labelling with recurrent neural networks pp. 37–45 (2012)

[5] Kim, Y.: Convolutional neural networks for sentence classification. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1746–1751. Association for Computational Linguistics, Doha, Qatar (Oct 2014). https://doi.org/10.3115/v1/D14-1181, https://aclanthology.org/D14-1181

[6] Lv, C., Xu, J., Zheng, X.: Spiking convolutional neural networks for text classification. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=pgU3k7QXuz0

[7] Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural networks **10**(9), 1659–1671 (1997)

[8] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. pp. 1631–1642. Association for Computational Linguistics, Seattle, Washington, USA (Oct 2013), https://www.aclweb.org/anthology/D13-1170

[9] Zenke, F., Vogels, T.P.: The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. Neural computation **33**(4), 899–925 (2021)