# **脉冲神经网络相关论文学习和实验**

陈子睿

2200012913@stu.pku.edu.cn

# **介绍**

重点对《Optimal ANN-SNN conversion for highaccuracy and ultra-low-latency spiking neural networks》（以下简称论文1）、《Reducing ANN-SNN Conversion Error through Residual Membrane Potential》（以下简称论文2）两篇论文进行了研究，阅读代码，尝试改进算法，复现相关测试数据。主要完成了如下内容：

（1）尝试负脉冲的实现。扩展阅读了相关论文和代码，以QCFS （Bu et al. 2022）的代码为框架基础，参考了SNM（Wang et al. 2022）的思路和代码，实现了对负脉冲Negspike的处理。

（2）学习SRP（Hao et al. 2023）通过预训练标记无效神经元的实现方法，将代码整合到QCFS的代码中。

（3）对QCFS、QCFS+Negspike、QCFS+SRP、QCFS+混合 等实现方式，进行了实验数据对比。

（4）对SRP中的Fig 3数据分析表格进行了复现，在cifar-10数据集上观测了不同Layer中各种误差的分布情况。

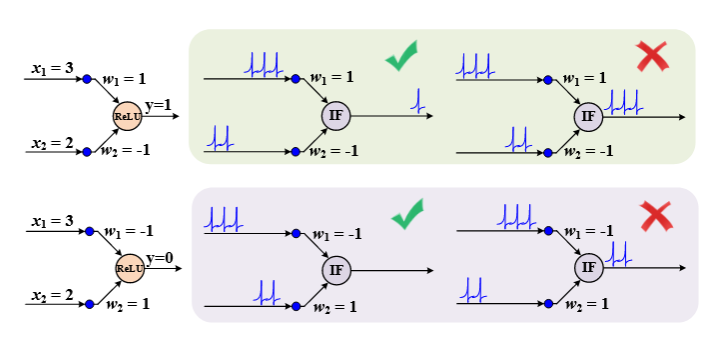
（5）对SNN的编码框架和相关内容进行了初步学习。

# **论文代码的复现和改进**

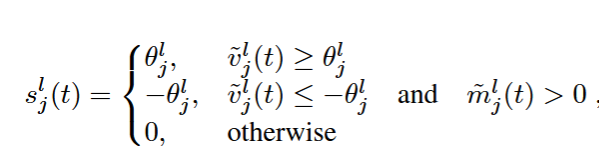
## **增加负脉冲**

论文1中对ANN to SNN的转换误差进行了分类：Clipping error、Quantizaiton erro、Unevenness error。Unevenness error产生的根本原因在于，通常SNN当L层的神经元在接收L-1层的输入时，只有当膜电压 > threshold时才发送脉冲。这样，当L-1层的正负值输入序列发生变化时，L层会产生不同的脉冲数量。而对于ANN来说，不论正负值输入序列如何变化，L层的输出总是保持一致的。

在尝试构造正负脉冲解决该问题时，检索到论文《Signed Neuron with Memory: Towards Simple, Accurate and HighEfficient ANN-SNN Conversion》对此问题进行了详细分析，通过**“Signed Neuron with Memory”**（SNM）方式较好地解决了Unevenness error。



该论文的主要思路为：在神经元中标记已发送的正脉冲数量，当正脉冲数量>0时，如触发负脉冲阈值，则发送负脉冲，否则不发送。最后对正负脉冲进行叠加处理。如下式：



基于QCFS （Bu et al. 2022）的代码框架，参考该论文及其代码，重现了负脉冲发送机制。主要代码如下：

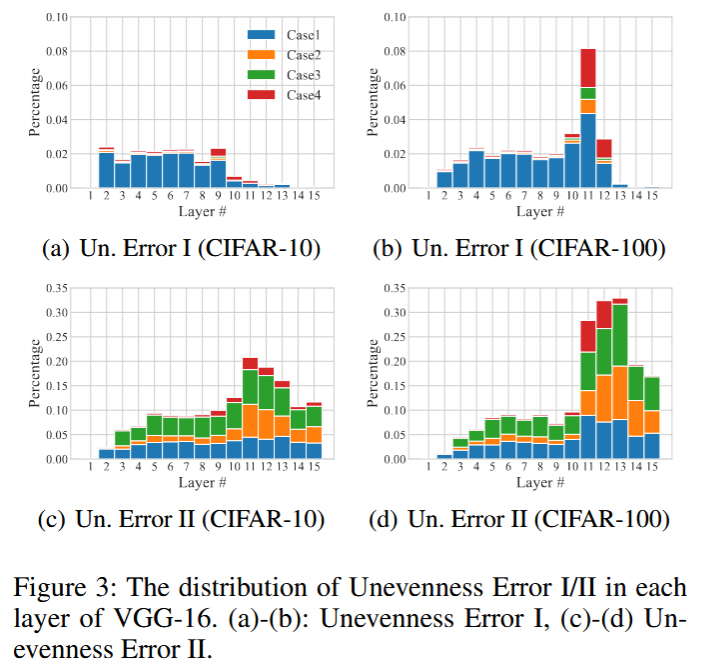
|  |
| --- |
| class IF(nn.Module):      def \_\_init\_\_(self, T=0, L=8, thresh=8.0, tau=1., gama=1.0):          super(IF, self).\_\_init\_\_()          self.act = ZIF.apply          self.thresh = nn.Parameter(torch.tensor([thresh]), requires\_grad=True)          self.tau = tau          self.gama = gama          self.expand = ExpandTemporalDim(T)          self.merge = MergeTemporalDim(T)          self.L = L          self.T = T          self.loss = 0      def forward(self, x):          if self.T > 0:              thre = self.thresh.data              x = self.expand(x)              mem = 0.5 \* thre              # 初始化标志位，用于判断是否产生负脉冲。              # 注意要把T的向量位减掉，否则和下面spike的向量形式不一样              flag = torch.zeros\_like(x[0,...])              spike\_pot = []              for t in range(self.T):                  mem = mem + x[t, ...]                  posspike = self.act(mem - thre, self.gama) \* thre                  # 添加负脉冲计算，直接取了thre的负值                  negspike=mem.le(thre\*(-1)).float() \* thre\*(-1)                      # 根据标志位确定是否产生负脉冲                  compare = torch.where(flag > 0, torch.ones\_like(flag), torch.zeros\_like(flag))                  negspike = negspike \* compare                  # 综合正负脉冲得到最终脉冲向量                  spike = posspike + negspike                  mem = mem - spike                  # 更新标志位                  flag= flag + spike                  spike\_pot.append(spike)              x = torch.stack(spike\_pot, dim=0)              x = self.merge(x)          else:              x = x / self.thresh              x = torch.clamp(x, 0, 1)              x = myfloor(x\*self.L+0.5)/self.L              x = x \* self.thresh          return x |

## **重现SRP**

论文2在论文1的基础上进一步做了优化，对Unevenness error分四种情况进行详细讨论。通过在cifar-10、cifar-100数据集上进行实验得到，Case 1: *αι=0，ϕι (T ) >αι* 是占比最高的误差情形，并进一步证明当*νι (T )<0*时，发生Case 1误差或无误差，即*ϕι (T )= αι=0*。

基于上述证明和分析，增加了预处理环节（*time-step = τ*），如果*νι (τ) < 0，*则将该神经元标记为dead，在正式处理环节不再对神经元进行处理。

基于论文1的QCFS代码框架，尝试对论文2中Figure3的统计数据进行复现。



代码如下：

|  |
| --- |
| def val(model, test\_loader, device, T):      correct = 0      total = 0      model.eval()      # 定义钩子函数      def save\_hook(m, x, y):          # 获取当前时间戳          timestamp = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S\_%f")          # 创建保存目录          if T==0:              os.makedirs("hook\_outputs\_ANN", exist\_ok=True)              filename = f"hook\_outputs\_ANN/{m.\_\_class\_\_.\_\_name\_\_}\_{timestamp}.txt"          else:              os.makedirs("hook\_outputs\_SNN", exist\_ok=True)              # 生成文件名              filename = f"hook\_outputs\_SNN/{m.\_\_class\_\_.\_\_name\_\_}\_{timestamp}.txt"          # 保存输出到文件          with open(filename, 'a') as f:              #f.write(f"Input:{x[0].shape[1]}\n")              #np.savetxt(f, x[0].detach().cpu().numpy().reshape(-1, x[0].shape[1]), fmt='%f')              f.write(f"Output:{y.shape[1]}\n")              np.savetxt(f, y.detach().cpu().numpy().reshape(-1, y.shape[1]), fmt='%f')              f.write("\n")          print(f"Saved output of {m.\_\_class\_\_.\_\_name\_\_} to {filename}")          return      #查找所有IF层，返回到一个列表中      def list\_modules(model):          NFmodules=[]          for name, module in model.\_modules.items():              if hasattr(module, "\_modules"):                  for sub\_name, sub\_module in module.\_modules.items():                      if 'IF' in sub\_module.\_\_class\_\_.\_\_name\_\_:                          NFmodules.append(sub\_module)              if 'IF' in module.\_\_class\_\_.\_\_name\_\_:                  NFmodules.append(module)          return Nfmodules      #IF层注册钩子      for test\_layer in list\_modules(model):          hook = test\_layer.register\_forward\_hook(save\_hook)      with torch.no\_grad():          for batch\_idx, (inputs, targets) in enumerate((test\_loader)):              inputs = inputs.to(device)              if T > 0:                  outputs = model(inputs).mean(0)              else:                  outputs = model(inputs)              \_, predicted = outputs.cpu().max(1)              total += float(targets.size(0))              correct += float(predicted.eq(targets).sum().item())          final\_acc = 100 \* correct / total      return final\_acc |

测试数据如下，和论文2中Figure3的数据有一定差异，但各类别数据的分布比例较相似。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **0** | **1** | **2** | **3** |
| ***αι=0*** | 0.6656 | 0.6999 | 0.6837 | 0.7878 |
| ***αι>0*** | 0.3344 | 0.3001 | 0.3163 | 0.2122 |
| **Case 1: *αι=0，ϕι (T ) >αι*** | 0.1593 | 0.2574 | 0.2222 | 0.1947 |
| **Case 2: *αι>0，ϕι (T ) >αι*  (包含Case 4)** | 0.0862 | 0.0691 | 0.0601 | 0.0252 |
| **Case 3: *αι>0，ϕι (T ) <αι*** | 0.2118 | 0.2306 | 0.2561 | 0.1730 |

论文2代码中SNN 神经元是基于**SpikingJelly(惊蜇)**实现。进一步查阅学习了SpikingJelly框架（https://spikingjelly.readthedocs.io/），该框架提供了全栈式的脉冲深度学习解决方案，提供神经形态数据处理、深度SNN的构建、替代梯度训练、ANN转换SNN、权重量化和神经形态芯片部署等功能。

论文1的代码相对较简洁，为了便于实验，将论文2的预处理环节移植到了论文1的代码框架中。

|  |
| --- |
| class IF(nn.Module):      def \_\_init\_\_(self, T=0, L=8, thresh=8.0, tau=1., gama=1.0):          super(IF, self).\_\_init\_\_()          self.act = ZIF.apply          self.thresh = nn.Parameter(torch.tensor([thresh]), requires\_grad=True)          self.tau = tau          self.gama = gama          self.expand = ExpandTemporalDim(T)          self.merge = MergeTemporalDim(T)          self.L = L          self.T = T          self.loss = 0      def forward(self, x):          if self.T > 0:              thre = self.thresh.data              x = self.expand(x)              mem = 0.5 \* thre              spike\_pot = []              # 预处理===========              for tp in range(4):                  mem = mem + x[tp, ...]                  spike = self.act(mem - thre, self.gama) \* thre                  mem = mem - spike              deadneuron\_flag = torch.where(mem > 1e-3,torch.ones\_like(mem), torch.zeros\_like(mem))              # 正式处理              mem = 0.5 \* thre  # 初始化膜电位              for t in range(self.T):                  mem = mem + x[t, ...]                  spike = self.act(mem - thre, self.gama) \* thre                  # 处理dead neuron                  spike = spike \* deadneuron\_flag                  mem = mem - spike                  spike\_pot.append(spike)              x = torch.stack(spike\_pot, dim=0)              x = self.merge(x)          else:              x = x / self.thresh              x = torch.clamp(x, 0, 1)              x = myfloor(x\*self.L+0.5)/self.L              x = x \* self.thresh          return x |

## **叠加上述处理步骤**

在论文1的QCFS代码框架中叠加论文2的SRP预处理环节、负脉冲处理。

# **实验结果**

**Table 1： cifar10数据集测试结果**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **0** | **1** | **2** | **4** | **8** | **16** | **32** | **64** |
| **QCFS** | 95.78 | 88.93 | 90.67 | 94.21 | 95.4 | 95.73 | 95.74 | 95.83 |
| **NegSpike** |  | 88.93 | 90.55 | 94.54 | 95.56 | 95.78 | 95.77 | 95.82 |
| **SRP(*τ*=4)** |  |  |  | 95.43 | 95.55 | 95.58 | 95.57 | 95.6 |
| **SRP+Neg** |  |  |  | 95.48 | 95.56 | 95.6 | 95.62 | 95.61 |
|  |  |  |  |  |  |  |  |  |
| **补偿神经元的脉冲** |  |  |  | 95.67 | 95.85 | 95.84 | 95.81 | 95.81 |

**Table 2： cifar100数据集测试结果**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **0** | **1** | **2** | **4** | **8** | **16** | **32** | **64** |
| **QCFS** | 77.19 | 58.65 | 64.95 | 71.24 | 75.14 | 77.01 | 77.18 | 77.42 |
| **NegSpike** |  | 58.65 | 65.31 | 72.0 | 76.04 | 77.23 | 77.37 | 77.17 |
| **SRP(*τ*=4)** |  |  |  | 76.27 | 76.45 | 76.59 | 76.61 | 76.55 |
| **SRP+Neg** |  |  |  | 76.06 | 76.56 | 76.68 | 76.66 | 76.63 |
|  |  |  |  |  |  |  |  |  |
| **补偿神经元的脉冲** |  |  |  | 76.75 | 77.08 | 77.12 | 77.11 | 77.12 |

上面实验数据，T=0为ANN的测试结果。当T=1时，因只有一个时间序列，QCFS和NegSpike的结果相同。对于SRP(τ=4)，当T=4时，其准确率大于QCFS T=8的准确率，具有一定优势。相比而言，NegSpike无论在cifar-10还是cifar-100数据集上的测试结果均较稳定。

另，在4090显卡上运行，当T>=128就开始报错，超出内存。

# **相关体会和思考**

**（1）ANN to SNN的转换路径**

ANN中的ReLU神经元非线性激活和SNN中IF神经元(采用减去阈值threshold 方式重置)的发放率有着极强的相关性，目前各类转换主要借助这个特性来进行优化。

**（2）如何更有效地利用数据集**

目前的处理方式均为将一个input向量通过add dimension方式扩展，然后循环处理T遍，无疑会增加推理时间成本和资源消耗。

**（3）SNN的研究热点和方向**

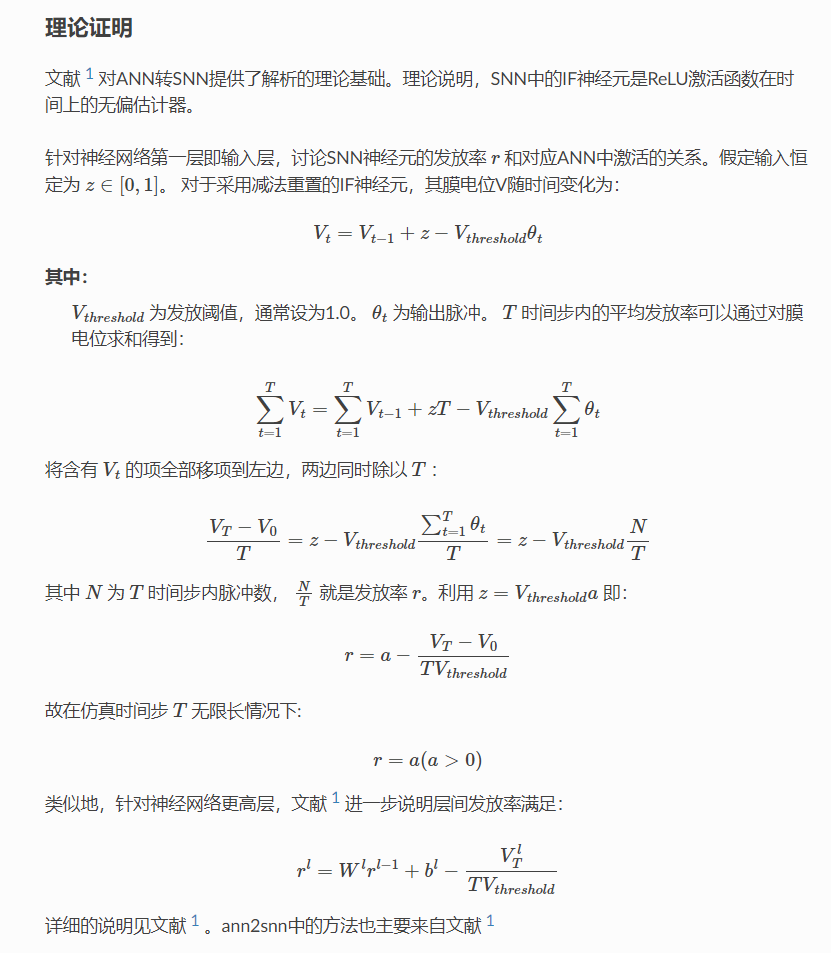
**训练成本、推理成本和准确率的平衡。**从相关论文来看，SNN的准确率在cifar等数据集上已经很容易接近ANN。最近的这篇论文"Inference-Scale Complexity in ANN-SNN Conversion for High-Performance and  Low-Power Applications"（Bu et al. 2024）提到，"Nevertheless, the pursuit of larger models raises concerns about high energy consumption for model inference and training. The deployment of large models on resource-constrained devices has also become a challenge." 最近大模型的火爆、deepseek在计算资源方面的大幅度降低，让人们将注意力在关注进一步优化效果的同时，也将更多的注意力转移到更节省资源的模型算法上。

**适用于更多的数据集和场景。**除了传统的图像分类，扩展到语义分割、对象检测和视频分类等领域。

在这个过程中，我们必须注意到，复制得到的4份一样的数据，并没有包含有意义的随着时间变化的信息，SNN是否更像是一种概率处理机制？

ANN转换SNN — spikingjelly alpha 文档

https://spikingjelly.readthedocs.io/zh-cn/0.0.0.0.14/activation\_based/ann2snn.html



Parameter containing:

tensor([0.2143], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.4723], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.4069], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.5692], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.4018], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.3899], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.3906], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.2314], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.1108], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.0931], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.0879], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.1005], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.5079], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.3124], device='cuda:0', requires\_grad=True)

Parameter containing:

tensor([0.3885], device='cuda:0', requires\_grad=True)

我来计算当L=8, thresh=2时，这些输入值的输出：

| **输入x** | **步骤1: x/2** | **步骤2: clamp(x,0,1)** | **步骤3: floor(x*8+0.5)/8 | 步骤4: x*2** | **输出** |
| --- | --- | --- | --- | --- |
| -1 | -0.5 | 0 | 0 | **0** |  |
| -0.3 | -0.15 | 0 | 0 | **0** |  |
| -0.5 | -0.25 | 0 | 0 | **0** |  |
| -0.7 | -0.35 | 0 | 0 | **0** |  |
| 0 | 0 | 0 | 0 | **0** |  |
| 0.1 | 0.05 | 0.05 | 0 | **0** |  |
| 0.124 | 0.062 | 0.062 | 0 | **0** |  |
| 0.125 | 0.0625 | 0.0625 | 0.125 | **0.25** |  |
| 0.5 | 0.25 | 0.25 | 0.25 | **0.5** |  |
| 0.8 | 0.4 | 0.4 | 0.375 | **0.75** |  |
| 1 | 0.5 | 0.5 | 0.5 | **1** |  |
| 1.125 | 0.5625 | 0.5625 | 0.625 | **1.25** |  |
| 1.3 | 0.65 | 0.65 | 0.625 | **1.25** |  |
| 1.375 | 0.6875 | 0.6875 | 0.75 | **1.5** |  |
| 1.5 | 0.75 | 0.75 | 0.75 | **1.5** |  |
| 1.7 | 0.85 | 0.85 | 0.875 | **1.75** |  |
| 2 | 1 | 1 | 1 | **2** |  |
| 2.2 | 1.1 | 1 | 1 | **2** |  |
| 2.5 | 1.25 | 1 | 1 | **2** |  |
| 2.6 | 1.3 | 1 | 1 | **2** |  |

# **References**

Tong Bu, Wei Fang, Jianhao Ding, PengLin Dai, Zhaofei Yu, and Tiejun Huang. Optimal ANN-SNN conversion for highaccuracy and ultra-low-latency spiking neural networks. In International Conference on Learning Representations, 2022. 2

Wang, Y.; Zhang, M.; Chen, Y.; and Qu, H. 2022. Signed Neuron with Memory: Towards Simple, Accurate and HighEfficient ANN-SNN Conversion. In International Joint Conference on Artificial Intelligence.

Zecheng Hao, Tong Bu, Jianhao Ding, Tiejun Huang, and Zhaofei Yu. Reducing ann-snn conversion error through residual membrane potential. arXiv preprint arXiv:2302.02091, 2023.

# **实验结果（过程稿，包含修改初始模电压、根据统计信息修改thre）**

**Table 1： cifar10数据集测试结果**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **0** | **1** | **2** | **4** | **8** | **16** | **32** | **64** |
| **QCFS** | 95.78 | 88.93 | 90.67 | 94.21 | 95.4 | 95.73 | 95.74 | 95.83 |
| **NegSpike** |  | 88.93 | 90.55 | 94.54 | 95.56 | 95.78 | 95.77 | 95.82 |
| **SRP(*τ*=4)** |  |  |  | 95.43 | 95.55 | 95.58 | 95.57 | 95.6 |
| **初始模电压修改为51%thre** | | |  | 95.56 | 95.67 | 95.68 | 95.67 | 95.7 |
| **SRP+Neg** |  |  |  | 95.48 | 95.56 | 95.6 | 95.62 | 95.61 |
| **根据统计信息修改模型的thre** | | |  | 94.43 | 95.12 | 95.43 | 95.52 | 95.55 |
|  |  |  |  |  |  |  |  |  |
| **补偿神经元的脉冲** |  |  |  | 95.67 | 95.85 | 95.84 | 95.81 | 95.81 |

**Table 2： cifar100数据集测试结果**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **0** | **1** | **2** | **4** | **8** | **16** | **32** | **64** |
| **QCFS** | 77.19 | 58.65 | 64.95 | 71.24 | 75.14 | 77.01 | 77.18 | 77.42 |
| **NegSpike** |  | 58.65 | 65.31 | 72.0 | 76.04 | 77.23 | 77.37 | 77.17 |
| **SRP(*τ*=4)** |  |  |  | 76.27 | 76.45 | 76.59 | 76.61 | 76.55 |
|  |  |  |  | 76.2 | 76.18 | 76.3 | 76.31 | 76.37 |
| **SRP+Neg** |  |  |  | 76.06 | 76.56 | 76.68 | 76.66 | 76.63 |
|  |  |  |  | 72.08 | 75.23 | 76.55 | 76.72 | 76.67 |
|  |  |  |  |  |  |  |  |  |
| **补偿神经元的脉冲** |  |  |  | 76.75 | 77.08 | 77.12 | 77.11 | 77.12 |