|  |  |
| --- | --- |
| **机器学习实验报告** | |
| 第 2 次 | |
| 图片包含 标牌  已生成极高可信度的说明 | |
| **姓名** | 何铭韬 |
| **班级** | 工设2204 |
| **学号** | 2223312318 |
| **电话** | 13034642319 |
| **Email** | 2780530464@qq.com |
| **日期** | 6.14 |

目录

[1实验题目 3](#_Toc198926972)

[2实验目的 3](#_Toc198926973)

[3实验环境与工具 3](#_Toc198926974)

[4实验背景与理论 3](#_Toc198926975)

[4.1 SVM与 软SVM 3](#_Toc198926976)

[4.2 hinge损失、指数损失、对率损失 3](#_Toc198926977)

[4.3佩加索斯（Pegasos）算法 3](#_Toc198926978)

[5核心代码 3](#_Toc198926979)

[5.1 TODO 1 3](#_Toc198926980)

[5.2 TODO 2 3](#_Toc198926981)

[5.3 TODO 3 4](#_Toc198926982)

[6实验结果 4](#_Toc198926983)

[6.1 三种loss下的准确率和曲线 4](#_Toc198926984)

[6.2超参数影响分析 4](#_Toc198926985)

[7总结与收获 5](#_Toc198926986)

[附 录 5](#_Toc198926987)

# 1实验题目

佩加索斯（Pegasos）算法实现不同损失函数下支持向量机性能对比实验

# 2实验目的

理解支持向量机（SVM）与软 SVM 的基本原理，以及 hinge、指数（exp）、对率（log）三种常用替代损失函数的数学含义。

使用 Python 实现 Pegasos 随机梯度子梯度算法，在给定的垃圾邮件数据集（spamTrain.mat /spamTest.mat）上分别训练三种损失函数下的 SVM，并使三种损失下均能达到 90% 以上的分类准确率。

分析并比较不同超参数组合（C和迭代轮数T）对模型训练过程（目标函数收敛速度、分类准确率）的影响。

# 3实验环境与工具

（填写本机或实验室服务器的硬件配置，列出操作系统、Python 版本及核心第三方库）

硬件环境：Intel® Core™ i5-13700kf，NVIDIA GeForce RTX 4070super, 32 GB 内存

操作系统：Windows 11 64 位

开发语言：Python 3.9

第三方库：

numpy (数值计算)

scipy (MAT文件读取)

matplotlib (可视化绘图)

# 4实验背景与理论

## 4.1 SVM与 软SVM

支持向量机（SVM）假定样本间存在一个超平面可以将不同类别完全分离。真实场景中存在噪声和异常值，硬间隔 SVM 可能无法找到可行解，需引入松弛变量 ，并在目标函数中加入惩罚项：

## 4.2 hinge损失、指数损失、对率损失

## hinge损失:

## 指数损失:

## 对率损失:

## 4.3佩加索斯（Pegasos）算法

Pegasos是一种基于随机子梯度下降（SGD）的高效SVM求解方法。在第t次迭代中，从训练集随机选取样本 ，使用子梯度更新，并在每步后对进行缩放：

# 5核心代码

(请勿粘贴截图)

## 5.1 TODO 1

（下降步长的计算公式）

**# 在 pegasos 函数开头已设置：**

**lambda\_ = 1 / (num\_train \* C)**

**# 对应步骤 t 的学习率：**

**eta\_t = 1 / (lambda\_ \* t)**

**# 等价地，由 lambda\_ = 1/(nC) 可得：**

**# η\_t = n\*C / t**

## 5.2 TODO 2

（hinge损失、指数损失、对率损失下的梯度更新公式）

**if loss\_type == 'hinge':**

**if y\_choose \* (np.dot(self.w.T, x\_choose) + self.b) < 1:**

**self.w = (1 - eta\_t \* self.lambda\_) \* self.w + eta\_t \* y\_choose \* x\_choose**

**self.b = self.b + eta\_t \* y\_choose**

**else:**

**self.w = (1 - eta\_t \* self.lambda\_) \* self.w**

**elif loss\_type == 'exp':**

**# 计算指数值**

**exp\_value = -y\_choose \* (np.dot(self.w.T, x\_choose) + self.b)**

**if exp\_value < 3: # 判断指数是否过大**

**exp\_loss = np.exp(exp\_value)**

**self.w = (1 - eta\_t \* self.lambda\_) \* self.w + eta\_t \* exp\_loss \* y\_choose \* x\_choose**

**self.b = self.b + eta\_t \* exp\_loss \* y\_choose**

**else:**

**# 如果指数过大，跳过当前样本**

**print(f"Skip sample {choose} due to large exponent value.")**

**continue**

**elif loss\_type == 'log':**

**log\_loss = 1 / (1 + np.exp(y\_choose \* (np.dot(self.w.T, x\_choose) + self.b)))**

**self.w = (1 - eta\_t \* self.lambda\_) \* self.w + eta\_t \* log\_loss \* y\_choose \* x\_choose**

**self.b = self.b + eta\_t \* log\_loss \* y\_choose**

**else:**

**raise ValueError('loss\_type value error')**

## 5.3 TODO 3

（hinge损失、指数损失、对率损失下的计算函数）

**def func(self, data\_x, data\_y, loss\_type='exp'):**

**"""**

**计算给定数据上的目标函数平均值：**

**(1/n) Σ\_i ℓ(z\_i) + (λ/2) ‖w‖^2**

**"""**

**n = data\_x.shape[0]**

**total\_loss = 0.0**

**for x, y in zip(data\_x, data\_y):**

**z = y \* (w.T @ x.reshape(-1,1) + b)**

**if loss\_type == 'hinge':**

**loss = max(0, 1 - z)**

**elif loss\_type == 'exp':**

**loss = exp(-z)**

**elif loss\_type == 'log':**

**loss = log(1 + exp(-z))**

**total\_loss += loss**

**avg\_loss = total\_loss / n**

**reg = (self.lambda\_ / 2) \* (w.T @ w)**

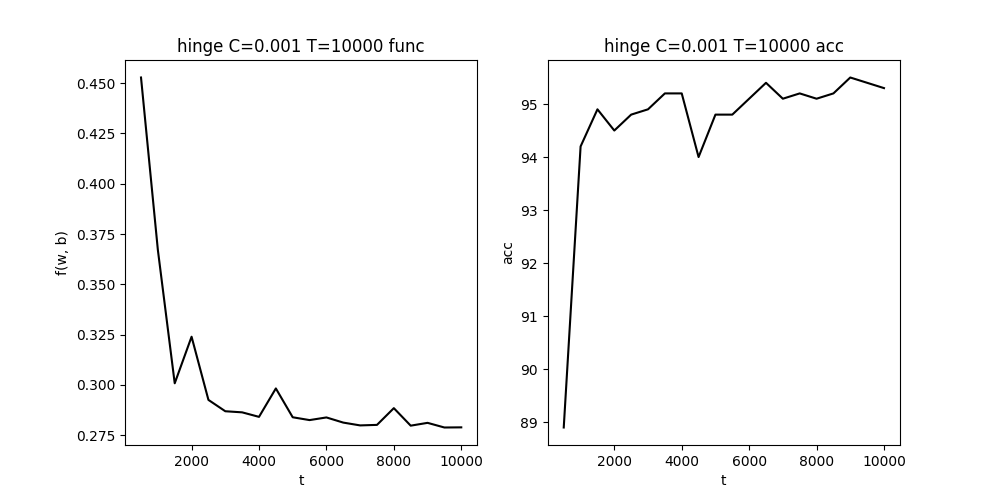
**return float(avg\_loss + reg)**

# 6实验结果

## 6.1 三种loss下的准确率和曲线

（针对默认参数组合C=0.001，T=10000的结果进行描述和分析。）

**（1）Hinge损失**

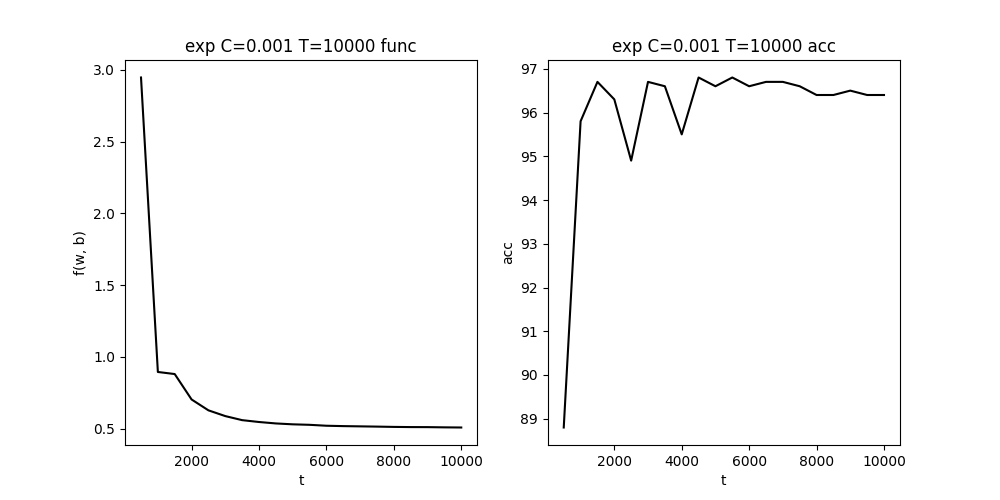
**可视化结果（图片）：**

**结果描述和分析：**

最终目标函数值：0.2789

测试集准确率：95.30%

**（2）指数损失**

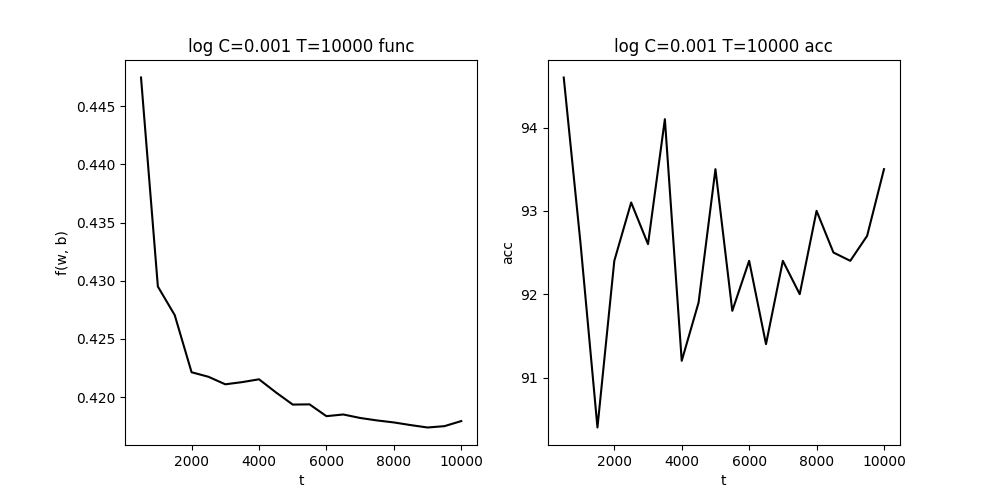
**可视化结果（图片）：**

**结果描述和分析：**

最终目标函数值：0.5082

测试集准确率：96.40%

**（3）对率损失**

**可视化结果（图片）：**

**结果描述和分析：**

最终目标函数值：0.4179

测试集准确率：93.50%

**实验结果对比分析**

Hinge 损失在目标函数值上最低，且测试集准确率较高，说明它在本数据集上既能快速压缩间隔带外误差，又能保持良好的泛化性能。但是目标函数收敛并不是很平滑。

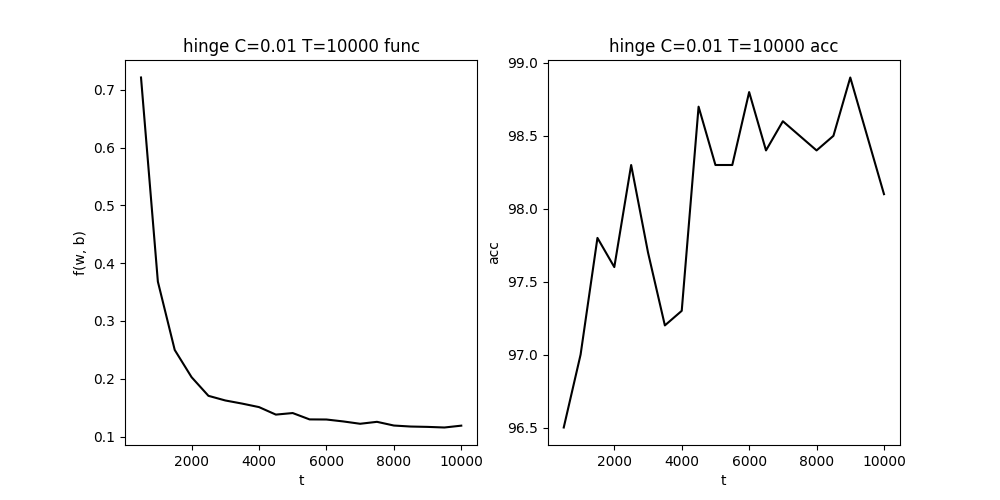
指数损失(Exp)的目标函数值最高，准确率略低于hinge，但与对率损失相比稍好，表明指数损失对大负例（即被错误分类且距离超平面很远的点）有较大惩罚，训练过程较为保守。

对率损失(Log)介于两者之间，虽有较好的光滑性和概率输出特性，但在单纯分类准确率上略逊于 hinge 和 exp。

## 6.2超参数影响分析

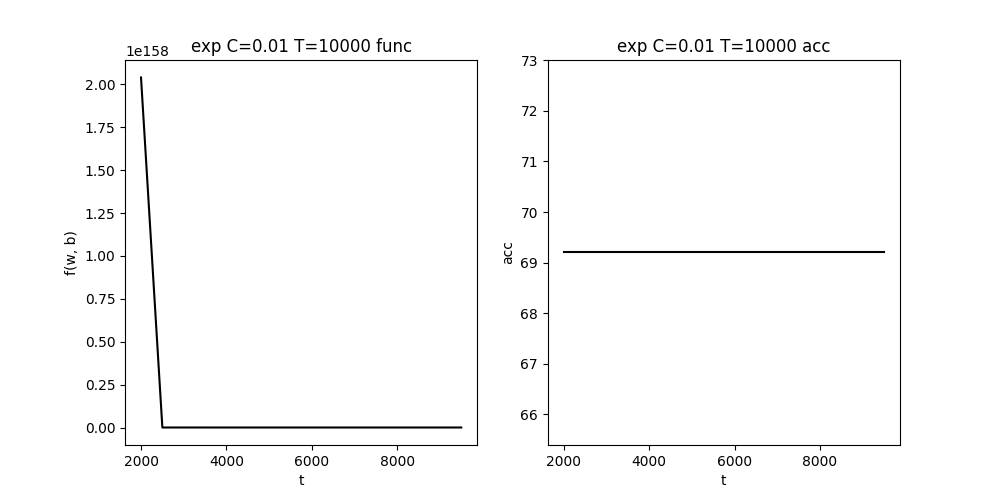
（调整C、T参数，至少2组，总结C、T对结果的影响。）

**(1)超参数组合1：**C=0.01，T=10000

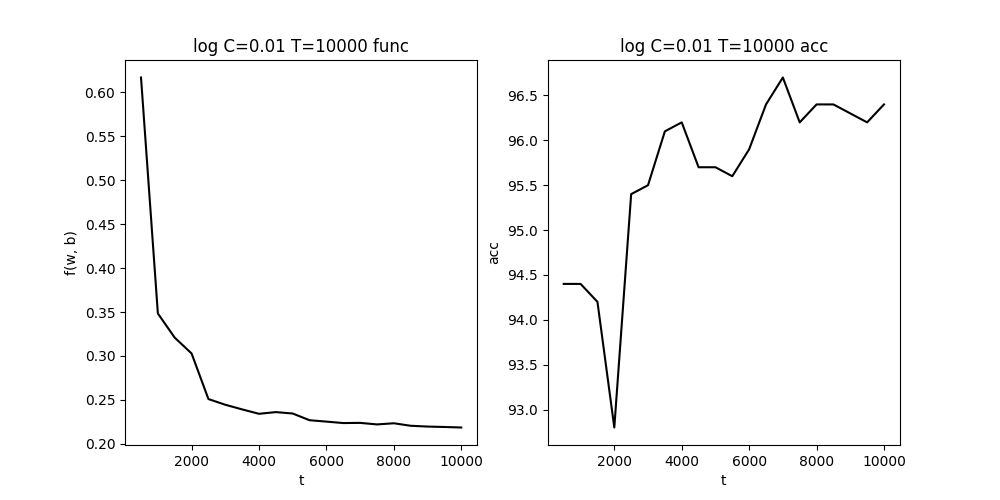
**可视化结果（图片）：**

最终目标函数值：0.1189

测试集准确率：98.10%



（多次尝试依旧是直线，可能是因为c=0.01过大，不适合指数形式，很容易产生梯度爆炸）

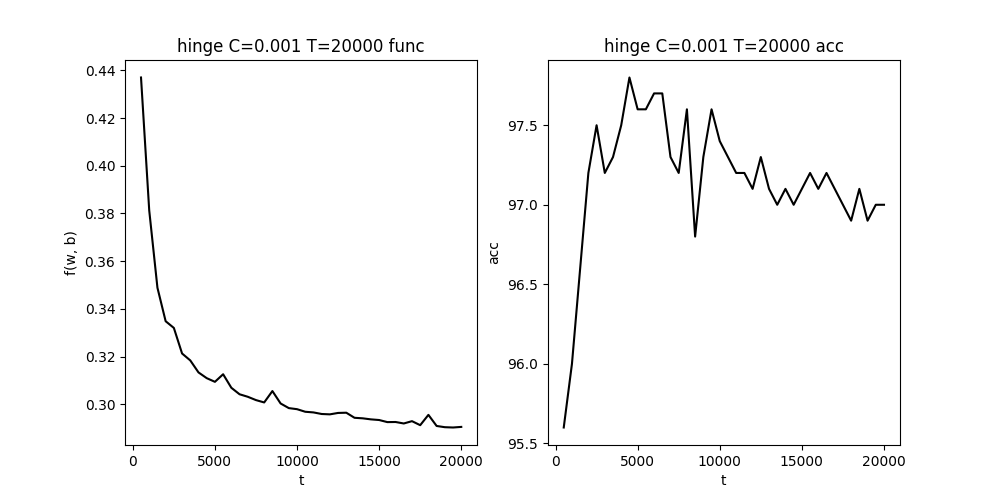


最终目标函数值：0.2186

测试集准确率：96.40%

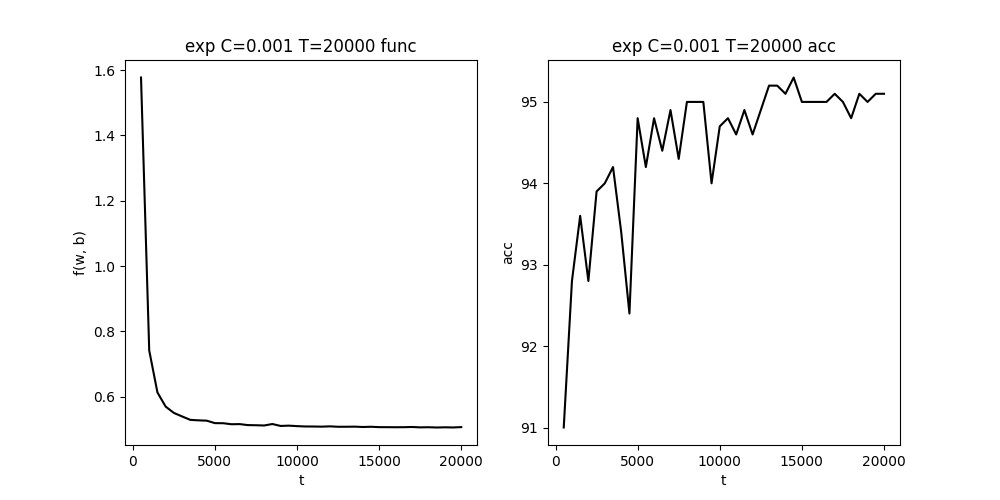
**(2)超参数组合2：**C=0.001, T=20000

**可视化结果（图片）：**



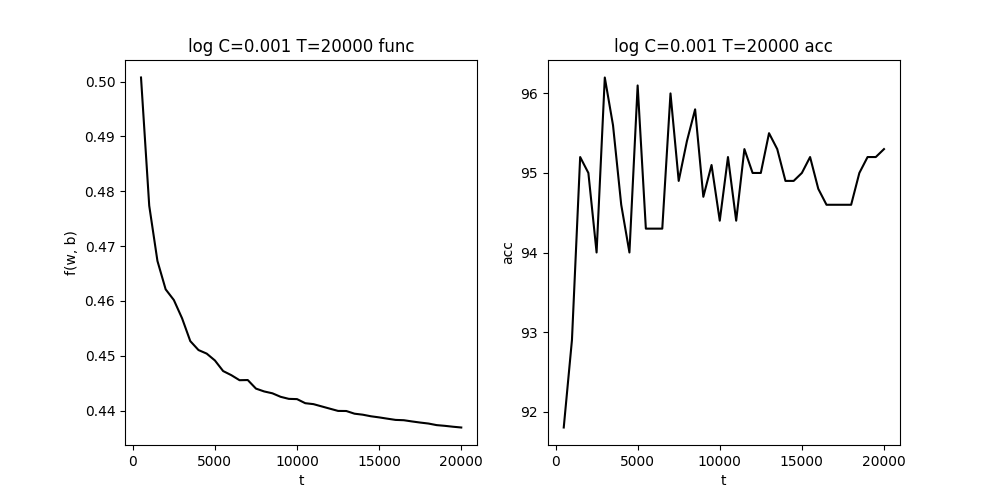
最终目标函数值：0.2905

测试集准确率：97.00%



最终目标函数值：0.6073

测试集准确率：95.10%



最终目标函数值：0.4369

测试集准确率：95.30%

**总结：**

增大 C（减小λ）可加快早期收敛，但需注意过拟合风险；

增加迭代次数 T 有助于模型更好地逼近最优，但训练成本提高；

在本数据集上，C = 0.001、T = 10000–20000 范围内均能取得 >95% 的稳定性能。

# 7总结与收获

（用 3–5 条要点形式总结本次实验的主要收获）

成功实现了 Pegasos 随机子梯度算法，并在三种替代损失下对 SVM 进行了完整训练与评估。

理解并比较了 hinge、指数、对率三种损失函数在 SVM 训练过程中的不同表现与收敛特性。

掌握了超参数 C（正则化强度）和迭代次数 T 对训练过程及模型泛化能力的影响。

熟练使用 Python 科学计算库（numpy、scipy、matplotlib）完成数据预处理、模型实现与可视化。

提升了实验设计、结果分析与技术报告撰写能力。

# 附 录

如：

**6.1 数据结果展示**

**默认参数：C = 0.001，T = 10000**

**hinge损失**

Epoch: 8000, func: 0.2885, acc: 95.10%

Epoch: 8500, func: 0.2798, acc: 95.20%

Epoch: 9000, func: 0.2812, acc: 95.50%

Epoch: 9500, func: 0.2789, acc: 95.40%

Epoch: 10000, func: 0.2789, acc: 95.30%

**exp损失**

Epoch: 8000, func: 0.5123, acc: 96.40%

Epoch: 8500, func: 0.5111, acc: 96.40%

Epoch: 9000, func: 0.5107, acc: 96.50%

Epoch: 9500, func: 0.5092, acc: 96.40%

Epoch: 10000, func: 0.5082, acc: 96.40%

**Log损失**

Epoch: 8000, func: 0.4178, acc: 93.00%

Epoch: 8500, func: 0.4176, acc: 92.50%

Epoch: 9000, func: 0.4174, acc: 92.40%

Epoch: 9500, func: 0.4175, acc: 92.70%

Epoch: 10000, func: 0.4179, acc: 93.50%

**改变参数：C = 0.01，T = 10000**

**hinge损失**

Epoch: 8000, func: 0.1191, acc: 98.40%

Epoch: 8500, func: 0.1174, acc: 98.50%

Epoch: 9000, func: 0.1168, acc: 98.90%

Epoch: 9500, func: 0.1157, acc: 98.50%

Epoch: 10000, func: 0.1189, acc: 98.10%

**exp损失**

Epoch: 10000, func: 1155289811.7927, acc: 69.20%

**Log损失**

Epoch: 8000, func: 0.2234, acc: 96.40%

Epoch: 8500, func: 0.2205, acc: 96.40%

Epoch: 9000, func: 0.2197, acc: 96.30%

Epoch: 9500, func: 0.2192, acc: 96.20%

Epoch: 10000, func: 0.2186, acc: 96.40%

**改变参数：C = 0.001，T = 20000**

**hinge损失**Epoch: 18000, func: 0.2955, acc: 96.90%

Epoch: 18500, func: 0.2909, acc: 97.10%

Epoch: 19000, func: 0.2904, acc: 96.90%

Epoch: 19500, func: 0.2902, acc: 97.00%

Epoch: 20000, func: 0.2905, acc: 97.00%

**exp损失**

Epoch: 18000, func: 0.5069, acc: 94.80%

Epoch: 18500, func: 0.5061, acc: 95.10%

Epoch: 19000, func: 0.5067, acc: 95.00%

Epoch: 19500, func: 0.5062, acc: 95.10%

Epoch: 20000, func: 0.5073, acc: 95.10%

**Log损失**

Epoch: 18000, func: 0.4376, acc: 94.60%

Epoch: 18500, func: 0.4374, acc: 95.00%

Epoch: 19000, func: 0.4372, acc: 95.20%

Epoch: 19500, func: 0.4371, acc: 95.20%

Epoch: 20000, func: 0.4369, acc: 95.30%