# 神经网络模型总结

MLE 算法指北

2025 年 2 月 1 日

## 1 神经网络模型结构

神经网络（Neural Networks, NN）是一类模拟生物神经元行为的数学模型，主要模型结构包括：

- **MLP（多层感知机）**
- **CNN（卷积神经网络）**
- **RNN（循环神经网络）**
- **LSTM（长短时记忆网络）**

### 1.1 MLP（多层感知机）

MLP 由多个全连接层（Fully Connected Layers）构成，其计算过程如下：

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

其中：

- $\mathbf{x}$ 为输入向量
- $\mathbf{W}$ 为权重矩阵
- $\mathbf{b}$ 为偏置项
- $\sigma$ 为激活函数（如 ReLU, Sigmoid）

**应用场景：**表格数据、特征提取后的数据分析。

### 1.2 CNN（卷积神经网络）

CNN 通过局部感受野和共享权重提取特征，核心操作如下：

$$\mathbf{h}_{i,j} = \sum_m \sum_n \mathbf{W}_{m,n}\mathbf{x}_{i-m,j-n} + b$$

**关键层：**

- 卷积层（Convolutional Layer）
- 池化层（Pooling Layer）
- 全连接层（Fully Connected Layer）

**应用场景：**图像处理、语音识别、视频分析。

### 1.3 RNN（循环神经网络）

RNN 适用于序列数据，核心计算过程如下：

$$\mathbf{h}_t = \sigma(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + \mathbf{b})$$

**应用场景：**语音识别、文本生成、时间序列预测。

### 1.4 LSTM（长短时记忆网络）

LSTM 通过引入门控机制解决 RNN 长期依赖问题，其核心结构如下：

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$$

**应用场景：**自然语言处理（NLP）、金融预测。

—

## 2 激活函数的选择与比较

神经网络中的激活函数决定了神经元的输出，它将线性变换结果映射到非线性空间，以学习复杂的数据模式。常见激活函数如下：

### 2.1 常见激活函数

**1. Sigmoid**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **优点：**值域在 $(0, 1)$，适合二分类问题。
- **缺点：**存在梯度消失问题，输出分布不均衡。
- **适用场景：**二分类问题的输出层。

**2. Tanh**

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **优点：**值域 $(-1, 1)$，相比 Sigmoid 居中零点，更适合深层网络。
- **缺点：**依然存在梯度消失问题。
- **适用场景：**隐藏层中的非线性激活。

**3. ReLU**

$$f(x) = \max(0, x)$$

- **优点：**计算高效，缓解梯度消失问题。
- **缺点：**存在“神经元死亡”问题，即负输入始终输出零。
- **适用场景：**深层网络的隐藏层，尤其是 CNN。

**4. Leaky ReLU**

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}$$

- **优点：**解决 ReLU 的神经元死亡问题，允许小负值梯度。
- **缺点：**需要调节参数 $\alpha$。
- **适用场景：**解决 ReLU 局限时使用。

**5. ELU（指数线性单元）**

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases}$$

- **优点：**负值具有非零梯度，避免神经元死亡。
- **缺点：**计算成本高于 ReLU。
- **适用场景：**需要负值贡献的深度学习任务。

**6. Softmax 函数**

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **优点：**输出概率分布，适用于多分类问题。
- **缺点：**对极端数值敏感，可能导致梯度消失。
- **适用场景：**多分类任务的输出层。

—

### 2.2 激活函数的比较

| 激活函数 | 数学公式 | 优点 | 缺点 |
|---|---|---|---|
| Sigmoid | $\frac{1}{1+e^{-x}}$ | 适用于二分类问题 | 梯度消失，训练缓慢 |
| Tanh | $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ | 归一化到 (-1,1) | 梯度消失 |
| ReLU | $\max(0, x)$ | 简单高效，减少梯度消失 | 神经元死亡问题 |
| Leaky ReLU | $\max(\alpha x, x)$ | 解决 ReLU 死亡问题 | 需调整超参数 |
| ELU | $\alpha(e^{x} - 1)$ | 负输入更平滑 | 计算复杂 |
| Softmax | $\frac{e^{x}}{\sum e^{x}}$ | 概率归一化 | 易受极端值影响 |

表 1: 激活函数的比较

—

### 2.3 激活函数的选择建议

- **二分类任务：**使用 Sigmoid 作为输出层，ReLU 作为隐藏层。
- **多分类任务：**使用 Softmax 作为输出层，ReLU 作为隐藏层。
- **深层网络：** ReLU 或 Leaky ReLU 适用于 CNN、深度 MLP。
- **负值敏感任务：**考虑使用 ELU 以保持负输入的贡献。

## 3 损失函数概述

损失函数是深度学习模型训练过程中用于衡量预测值与真实值之间差异的函数，合理选择损失函数有助于提高模型性能，主要分为以下几类：

- **分类任务损失函数**
- **回归任务损失函数**
- **拟合分布损失函数**
- **分位点学习损失函数**

—

### 3.1 分类任务损失函数

分类任务目标是将输入样本划分至正确类别，常见损失函数如下：

**交叉熵损失（Cross Entropy Loss）**

**定义：**

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

**特点：**

- 适用于多分类任务，输出概率分布。

- 对概率分布敏感，但易受异常值影响。

**适用场景：**图像分类、文本分类。

—

**Hinge 损失（SVM 损失）**
**定义：**

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^{n} \max(0, 1 - y_i \hat{y}_i)$$

**特点：**

- 适用于支持向量机（SVM），最大化分类边界。
- 不适合概率预测。

**适用场景：** SVM 分类任务，如文本、图像识别。

—

### 3.2 回归任务损失函数

回归任务目标是预测连续值，常见损失函数如下：

均方误差（MSE）
**定义：**

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

**特点：**

- 放大大误差的影响，适合精确预测任务。
- 易受异常值影响。

**适用场景：**房价预测、温度预测。

—

**平均绝对误差（MAE）**
**定义：**

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

**特点：**

- 对异常值更鲁棒，适合稳健性预测。
- 收敛速度较慢。

**适用场景：**经济数据、能源消耗预测。

—

**Huber 损失**
**定义：**

$$\mathcal{L}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta), & |y - \hat{y}| > \delta \end{cases}$$

**特点：**

- 结合 MSE 和 MAE 优势，对异常值具有一定容忍性。
- 需调整超参数 $\delta$。

**适用场景：**时间序列预测、金融分析。

—

### 3.3 拟合分布损失函数

用于估计数据的概率分布，常见损失函数如下：

**负对数似然损失（Negative Log-Likelihood, NLL）**
**定义：**

$$\mathcal{L}(y, \hat{y}) = -\log P(y|\theta)$$

**特点：**

- 适用于概率建模，提供不确定性估计。
- 需假设正确的分布类型。

**适用场景：**贝叶斯深度学习，统计模型训练。
**KL 散度（Kullback-Leibler Divergence）**
**定义：**

$$D_{KL}(P||Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)}$$

**特点：**

- 衡量两个概率分布的差异，适用于分布拟合。
- 对分布估计准确性要求高。

**适用场景：**生成模型、VAE。

—

**分位数损失（Quantile Loss）**
**定义：**

$$\mathcal{L}_\tau(y, \hat{y}) = \sum_{i=1}^{n} \begin{cases} \tau(y_i - \hat{y}_i), & y_i \geq \hat{y}_i \\ (1-\tau)(\hat{y}_i - y_i), & y_i < \hat{y}_i \end{cases}$$

**特点：**

- 适合预测分布特定分位点，减少异常影响。

**适用场景：**风险评估、库存预测。

# 4 优化器的特点与对比

**1. 梯度下降法（GD）**

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

**特点：**

- 计算所有样本的梯度，精度高但计算成本大。
- 适合小规模数据集，不适合大数据。

—

**2. 随机梯度下降（SGD）**

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t; x_i)$$

**特点：**

- 每次更新仅使用一个样本，计算快但震荡大。
- 适用于在线学习和大规模数据集。

—

**3. 动量 SGD（Momentum）**

$$v_t = \beta v_{t-1} + (1-\beta)\nabla J(\theta_t)$$

$$\theta_{t+1} = \theta_t - \alpha v_t$$

**特点：**

- 在梯度下降方向上增加惯性，减少振荡，提高收敛速度。
- 适用于深度神经网络。

—

**4. Adagrad**

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \nabla J(\theta_t)$$

$$G_t = \sum_{i=1}^{t} \nabla J(\theta_i)^2$$

**特点：**

- 适应不同参数的更新率，适合处理稀疏数据。
- 学习率随时间减小，可能导致训练停止。

—

**5. RMSprop**

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) \nabla J(\theta_t)^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \nabla J(\theta_t)$$

**特点：**

- 平滑梯度波动，适合非平稳目标，如时序数据。
- 常用于 RNN 训练。

—

**6. Adam**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \nabla J(\theta_t)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) \nabla J(\theta_t)^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

**特点：**

- 结合 Momentum 和 RMSprop 优势，适应性强，收敛速度快。
- 适用于大多数深度学习任务，如 CNN、NLP。

—

## 4.1 优化器比较

| 优化器 | 优点 | 缺点 | 适用场景 |
|---|---|---|---|
| GD | 精确收敛 | 计算成本高 | 小数据集 |
| SGD | 计算快，易于实现 | 震荡大 | 大规模数据、在线学习 |
| Momentum | 减少振荡，加速收敛 | 需调整动量参数 | 深度神经网络 |
| Adagrad | 适合稀疏数据 | 学习率过早衰减 | 文本、推荐系统 |
| RMSprop | 平滑梯度，适合时序数据 | 需调节参数 | RNN、时序预测 |
| Adam | 自适应收敛快 | 需较多计算资源 | CNN、NLP |

表 2: 优化器的比较

—

### 4.2 优化器选择建议

- **小规模数据：**使用 GD 或 SGD，适合精准收敛。
- **深度神经网络：** Adam 适用于 CNN 和 NLP 任务。
- **时序数据：** RMSprop 适合非平稳数据处理。
- **稀疏特征数据：**选择 Adagrad 以自动适应学习率。

—

### 4.3 总结

优化器的选择对深度学习模型的收敛速度和性能至关重要。**推荐选择：**

- 如果数据集较大，推荐使用 **SGD with Momentum** 或 **Adam**。
- 如果特征稀疏，推荐使用 **Adagrad**。
- 处理时序数据时，推荐使用 **RMSprop**。

## 5 归一化与正则化

在机器学习和深度学习中，**归一化（Normalization）**和 **正则化（Regularization）**是两种重要的技术，分别用于提升模型的训练稳定性和防止过拟合。

—

### 5.1 归一化（Normalization）

归一化的目的是将输入特征的数值范围调整到相同的尺度，以加速模型的收敛，提高训练稳定性，并防止某些特征对损失函数的主导作用。

**1. 最小-最大归一化（Min-Max Normalization）**

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

**特点：**

- 将数据缩放到 $[0, 1]$ 或 $[-1, 1]$。
- 受异常值影响较大。

**适用场景：**适合深度学习输入（如图像数据）。

—

**2. Z-score 归一化（标准化，Standardization）**

$$x' = \frac{x - \mu}{\sigma}$$

**特点：**

- 将数据转换为均值为 0，标准差为 1。
- 更适合服从正态分布的数据。

**适用场景：**线性回归、SVM 等算法。

—

**3. 批归一化（Batch Normalization, BN）**

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}}$$

$$y^{(i)} = \gamma \hat{x}^{(i)} + \beta$$

**特点：**

- 减少内部协变量偏移，加速收敛。
- 适用于深度神经网络（CNN、RNN）。

---

**4. 层归一化（Layer Normalization, LN）**

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

**特点：**

- 独立于 mini-batch 计算归一化。
- 适用于 NLP 任务，如 Transformer。

---

## 5.2 正则化（Regularization）

正则化的目的是在损失函数中增加惩罚项，以防止模型过拟合，增强泛化能力。

**1. L1 正则化（Lasso）**

$$\mathcal{L}_{\text{L1}} = \lambda \sum_{i=1}^{n} |w_i|$$

**特点：**

- 使部分权重变为零，实现特征选择。
- 适用于稀疏模型构建。

**为何 L1 会导致权重稀疏？**
L1 正则化的梯度恒定，为 $\pm 1$ 或 0，促使参数被压缩到零：

$$\frac{\partial}{\partial w_i}|w_i| = \begin{cases} +1, & w_i > 0 \\ -1, & w_i < 0 \end{cases}$$

---

**2. L2 正则化（Ridge）**

$$\mathcal{L}_{\text{L2}} = \lambda \sum_{i=1}^{n} w_i^2$$

**特点：**

- 使权重趋近于零，但不等于零。
- 提高模型的稳定性，减少过拟合。

**为何 L2 不导致权重稀疏？**
L2 正则化的梯度随权重线性变化：

$$\frac{\partial}{\partial w_i} w_i^2 = 2w_i$$

因此，权重在接近零时，梯度较小，难以完全收缩至零。

---

**3. Elastic Net 正则化**

$$\mathcal{L} = \lambda_1 \sum |w_i| + \lambda_2 \sum w_i^2$$

**特点：**

- 结合 L1 的稀疏性和 L2 的稳定性。
- 适用于高维稀疏数据。

---

## 5.3 归一化与正则化的区别

---

| 维度 | 归一化 | 正则化 |
|---|---|---|
| 目的 | 统一特征尺度 | 防止过拟合 |
| 影响对象 | 训练数据特征 | 模型权重 |
| 方法 | Min-Max, Z-score | L1, L2, Dropout |
| 适用阶段 | 训练前 | 训练过程中 |

表 3: 归一化与正则化的区别

### 5.4 归一化与正则化的选择建议

- 对于特征取值范围差异较大，应使用 **Z-score 归一化**或 **Min-Max 归一化**。
- 处理深度神经网络时，推荐使用 **批归一化（BN）**。
- 防止过拟合时，推荐使用 **L2 正则化**，如果需要特征选择，则使用 **L1 正则化**。
- 高维数据（如文本分类）推荐使用 **Elastic Net** 正则化。

—

**总结：**

- 归一化用于调整特征尺度，提高训练稳定性。
- 正则化用于控制模型复杂度，防止过拟合。
- L1 正则化促使权重稀疏，L2 正则化使权重趋近零但不为零。