第 5 章　关系网络与匹配网络

本章将讲解另一个有趣的单样本学习算法——关系网络。它是最简单、最有效的单样本学习算法之一。元学习旨在解决从少量样本中快速学习和泛化的问题，关系网络是为了实现这一目标而设计的。它的核心思想是学习一个关系评分函数，用于比较输入样本之间的相似性，从而进行快速的分类或回归任务。本章将探讨如何在单样本、少样本与零样本学习场景中使用关系网络。

本章内容：

- 关系网络。
- 单样本学习中的关系网络。
- 少样本学习中的关系网络。
- 零样本学习中的关系网络。
- 匹配网络。

5.1　关 系 网 络

在 2018 年的论文 *Learning to compare: relation network for few-shot learning*[1]中，Flood Sung 等提出了一种名为关系网络(relation network，RN)的小样本学习算法。这种算法基于元学习理念，采用有监督学习来估计样本点之间的距离。通过比较新样本点与过去样本点之间的距离，使 RN 算法可以对新样本点进行分类。虽然它主要适用于小样本分类问题，但其实这种思路对于任何分类问题都是有效的。

关系网络算法的结构和思路简洁明了，无论是在小样本分类还是大样本分类问题上，其表现都相当出色。特别是在小样本分类问题上，它超越了之前表现最好的方法，成为了该算法的一大亮点。

RN 的基本结构包括以下两个主要部分。

(1) 特征提取器：对输入样本进行特征提取，通常采用神经网络(如卷积神经网络、循环神经网络或 Transformer 等)。这一阶段的目的是将输入样本映射到一个特征空间，以便更好地衡量相似性。

(2) 关系模块：在特征空间中计算输入样本之间的关系评分。关系模块通常由多层神经网络组成，输入是特征提取器提取的两个样本的特征向量(可以是连接、拼接、相减等操作后的向量)，输出是一个关系评分，表示两个样本之间的相似性。

在 RN 算法中，特征提取器和距离度量都包含可训练的参数，这意味着它们是通过监督学习得到的。通过这种方式可以获得一个更好地反映数据特性的特征提取器和距离度量，从而更准确地对任务特性建模。

通常情况下，距离度量采用的是各种常见的度量方法，如欧氏距离、余弦距离等。然而，在 RN 算法中，距离度量中的参数是可训练的。这意味着在整个元学习框架中，优化目标函数时，会同时估计特征提取器和距离度量中的参数。这样有助于提高元学习模型的性能，并取得更好的模型效果。

在训练阶段，RN 通过最小化预测关系评分与真实类别之间的损失来优化参数。在测试阶段，RN 可以处理新的少样本类别，实现高效的分类或回归任务。

RN 在许多元学习任务中表现出色，例如图像分类、自然语言处理等。通过学习比较输入样本之间的相似性，RN 能够在少量样本的情况下实现快速泛化。

5.1.1 单样本学习中的关系网络

在单样本学习任务中，关系网络的训练和应用方式与元学习中略有不同。因为单样本学习意味着每个类别只有一个样本可用，所以在训练阶段，需要针对每个类别的单个样本进行特征提取和关系度量的学习。在测试阶段，关系网络

可以处理新的单样本类别，实现高效的分类或回归任务。

众所周知，在单样本学习中，每个类只有一个示例。例如，假设支持集包含 3 个类，每个类 1 个示例。如图 5-1 所示，有一个包含 3 个类别的支持集，即{昆虫，鸟，狗}。

假设有一个查询图像 x_j，如图 5-2 所示，希望预测该查询图像的类，图像如下所示。

标签 y_i	图像 x_i
狗	
	
昆虫	
	
鸟	
	

图 5-1　支持集

图 5-2　待查询图像示例

首先，从支持集中获取每个图像 $x[i]$，并将其传递给嵌入函数 $f[\Phi](x[j])$，以提取特征。由于支持集包含图像，因此可以使用卷积网络作为嵌入函数来学习嵌入。嵌入函数将提供支持集中每个数据点的特征向量。类似地，将把查询图像 $x[j]$传递给嵌入函数 $f[\Phi](x[j])$来学习其嵌入。

因此，一旦有了支持集 $f[\Phi](x[i])$和查询集 $f[\Phi](x[j])$的特征向量，就可以使用运算符 Z 组合它们。Z 可以是任何组合运算符；使用连接作为运算符，以合并支持集和查询集的特征向量，即 $Z(f[\Phi](x[i]),f[\Phi](x[j]))$。

如图 5-3 所示，我们将合并支持集 $f[\Phi](x[i])$和查询集 $f[\Phi](x[j])$的特征向量。但是这样的组合有什么用呢？这将帮助我们理解支持集中图像的特征向量与查询图像的特征向量之间的关系。在示例中，它将帮助我们理解昆虫、鸟和狗的图像的特征向量与查询图像的特征向量之间的关系，如图 5-3 所示。

但是如何衡量这种关联性呢？这就是为什么使用关系函数 $g(\Phi)$的原因。

图 5-3　关系图

将这些组合的特征向量传递给关系函数，该函数将生成从 0 到 1 的关系得分，代表支持集 $x[i]$中的样本与查询集 $x[j]$中的样本之间的相似性。

以下等式说明了如何计算关系网络中的关系得分。

$$r[i,j]=g_{\Phi}(Z(f_{\Phi}(x_i),f_{\Phi}(x_j))) \tag{5-1}$$

在该等式中，$r[i,j]$表示在支持集中的每个类别和查询图像之间的相似性的关系分数。由于支持集中有 3 个类别，在查询集中有 1 个图像，因此将获得 3 个分数，表明支持集中的所有 3 个类别与查询图像的相似程度。

图 5-4 显示了在单样本学习设置中关系网络的整体表示。

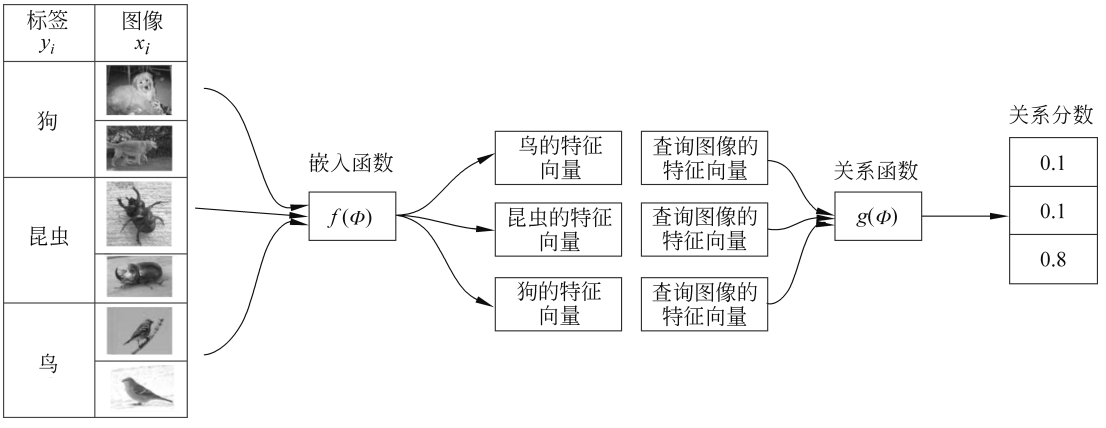

图 5-4　关系网络的整体表示

5.1.2　少样本学习中的关系网络

在少样本学习任务中，关系网络的训练方式通常采用 N-way K-shot 的设置。这意味着在每个训练任务中，从 N 个类别中每个类别选择 K 个样本作为支持集。关系网络根据支持集中的样本学习类别之间的相似性度量。在测试阶段，关系网络需要对新的样本进行分类，这些样本可能来自训练中未见过的类别。关系网络将新样本与支持集中的样本进行关系度量比较，根据相似性对新样本进行分类[7-10]。

关系网络在少样本学习任务中的具体流程主要分为以下几个步骤。

(1) 数据集划分。将数据集划分为训练集、验证集和测试集。训练集用于训练关系网络模型，验证集用于调整模型超参数，测试集用于评估模型在未知数据上的性能。

(2) 任务构建。在少样本学习中，通常使用 N-way K-shot 的设置。对于每个训练任务，从训练集中随机选择 N 个类别，然后从每个类别中选择 K 个样本作为支持集。此外，从每个类别中选择一个或多个样本作为查询集，用于计算损失和更新模型参数。

(3) 特征提取。利用特征提取器(如卷积神经网络、循环神经网络或 Transformer 等)对支持集和查询集中的样本进行特征提取。这一阶段的目的是将输入样本映射到一个特征空间，以便更好地衡量相似性。

(4) 关系度量。在特征空间中计算查询集中每个样本与支持集中所有样本之间的关系评分。关系模块通常由多层神经网络组成，输入是特征提取器提取的两个样本的特征向量(可以是连接、拼接、相减等操作后的向量)，输出是一个关系评分，表示两个样本之间的相似性。

(5) 损失计算与参数更新。根据查询集中样本的关系评分计算损失函数(如交叉熵损失)，然后使用梯度下降法(如 SGD、Adam 等优化器)更新模型参数。

(6) 验证与模型选择。在验证集上进行类似的任务构建和关系网络应用过程，根据验证集上的性能调整模型超参数和选择最优模型。

(7) 测试与性能评估。在测试集上应用训练好的关系网络,评估模型在未知数据上的泛化性能。

5.1.3 零样本学习中的关系网络

零样本学习(zero-shot learning,ZSL)是一种特殊的少样本学习任务,要求模型在训练过程中没有看到目标类别的任何样本。在零样本学习中,关系网络可以通过学习类别属性(如语义信息、视觉特征等)之间的关系来推断未见过的类别。

在零样本学习中使用关系网络的具体流程如下。

(1) 数据集划分。将数据集划分为训练集、验证集和测试集。训练集用于训练关系网络模型,验证集用于调整模型超参数,测试集用于评估模型在未知数据上的性能。

(2) 任务构建。与传统的少样本学习任务不同,零样本学习中训练集包含已知类别的样本,而测试集包含未知类别的样本。这些未知类别没有出现在训练过程中。

(3) 属性信息。为每个类别分配一个属性向量,通常采用语义嵌入(如Word2Vec、Glove 等)表示类别的语义信息。属性向量可以帮助关系网络理解类别之间的关系,从而实现对未见过类别的泛化。

(4) 特征提取。利用特征提取器(如卷积神经网络、循环神经网络或Transformer 等)对训练集中的样本进行特征提取。

(5) 关系模块训练。在特征空间中计算训练集中样本与对应属性向量之间的关系评分。通过最小化训练集上的损失函数(如交叉熵损失)来训练关系模块。

(6) 验证与模型选择。在验证集上进行类似的任务构建和关系网络应用过程,根据验证集上的性能调整模型超参数和选择最优模型。

(7) 测试与性能评估。在测试集上应用训练好的关系网络,利用属性向量为未知类别的样本计算关系评分。根据关系评分对未知类别的样本进行分类,并评估模型在未知数据上的泛化性能。

以下是一个基于 Relation Network 的单样本学习示例代码，数据集是常用的 mini-ImageNet。

(1) 导入所需依赖。

```
import tensorflow as tf
from tensorflow.keras import layers
import task_generator as tg
import math
from PIL import Image
import matplotlib.pyplot as plt
import os
import argparse
import numpy as np
import scipy as sp
import scipy.stats
```

(2) 定义超参数，可以根据实际需要更改。

```
parser = argparse.ArgumentParser(description="One Shot Visual Recognition")
parser.add_argument("-f","--feature_dim",type = int, default = 64)
parser.add_argument("-r","--relation_dim",type = int, default = 8)
parser.add_argument("-w","--class_num",type = int, default = 5)
parser.add_argument("-s","--sample_num_per_class",type = int, default = 1)
parser.add_argument("-b","--batch_num_per_class",type = int, default = 15)
parser.add_argument("-e","--episode",type = int, default= 500000)
parser.add_argument("-t","--test_episode", type = int, default = 600)
parser.add_argument("-l","--learning_rate", type = float, default = 0.001)
parser.add_argument("-u","--hidden_unit",type=int,default=10)
args = parser.parse_args()
#Hyper Parameters
FEATURE_DIM = args.feature_dim
RELATION_DIM = args.relation_dim
CLASS_NUM = args.class_num
SAMPLE_NUM_PER_CLASS = args.sample_num_per_class
BATCH_NUM_PER_CLASS = args.batch_num_per_class
EPISODE = args.episode
TEST_EPISODE = args.test_episode
LEARNING_RATE = args.learning_rate
HIDDEN_UNIT = args.hidden_unit
```

（3）定义函数，计算测试准确率的平均值和置信区间。

```
def mean_confidence_interval(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a)
    h = se * sp.stats.t._ppf((1+confidence)/2., n-1)
    return m,h
```

（4）定义两个神经网络模型：CNNEncoder 和 RelationNetwork。CNNEncoder 用于提取特征，RelationNetwork 用于比较两个样本之间的相似性。

```
class CNNEncoder(tf.keras.Model):
    def __init__(self):
        super(CNNEncoder, self).__init__(name='CNNEncoder')
        self.conv2a = layers.Conv2D(filters=64,input_shape=(84,84,3),kernel_
        size=(3,3),padding='valid',bias_initializer='glorot_uniform')
        self.bn2a = layers.BatchNormalization(axis=3,momentum=0.0,
        epsilon=1e-05,fused=False)
        self.pool2a = layers.MaxPool2D(pool_size=(2,2))
        self.conv2b = layers.Conv2D(filters=64,kernel_size=(3,3),
        padding='valid',bias_initializer='glorot_uniform')
        self.bn2b = layers.BatchNormalization(axis=3,momentum=0.0,
        epsilon=1e-05,fused=False)
        self.pool2b = layers.MaxPool2D(pool_size=(2,2))
        self.conv2c = layers.Conv2D(filters=64,kernel_size=(3,3),
        padding='same',bias_initializer='glorot_uniform')
        self.bn2c = layers.BatchNormalization(axis=3,momentum=0.0,
        epsilon=1e-05,fused=False)
        self.conv2d = layers.Conv2D(filters=64,kernel_size=(3,3),
        padding='same',bias_initializer='glorot_uniform')
        self.bn2d = layers.BatchNormalization(axis=3,momentum=0.0,
        epsilon=1e-05,fused=False)
    def call(self, input_tensor, training=False):
        x = self.conv2a(input_tensor)
        x = self.bn2a(x,training)
        x = tf.nn.relu(x)
        x = self.pool2a(x)
        x = self.conv2b(x)
        x = self.bn2b(x,training)
```

```
            x = tf.nn.relu(x)
            x = self.pool2b(x)
            x = self.conv2c(x)
            x = self.bn2c(x,training)
            x = tf.nn.relu(x)
            x = self.conv2d(x)
            x = self.bn2d(x,training)
            return tf.nn.relu(x)
    class RelationNetwork(tf.keras.Model):
        def __init__(self):
            super(RelationNetwork, self).__init__(name='RelationNetwork')
            self.conv2a = layers.Conv2D(filters=64,input_shape=(19,19,128),
            kernel_size=(3,3),padding='valid',bias_initializer='glorot_uniform')
            self.bn2a = layers.BatchNormalization(axis=3,momentum=0.0,
            epsilon=1e-05,fused=False)
            self.pool2a = layers.MaxPool2D(pool_size=(2,2))
            self.conv2b = layers.Conv2D(filters=64,kernel_size=(3,3),padding=
            'valid',bias_initializer='glorot_uniform')
            self.bn2b = layers.BatchNormalization(axis=3,momentum=0.0,
            epsilon=1e-05,fused=False)
            self.pool2b = layers.MaxPool2D(pool_size=(2,2))
            self.fc1 = layers.Dense(8,activation='relu')
            self.fc2 = layers.Dense(1,activation='sigmoid')
        def call(self, input_tensor, training=False):
            x = self.conv2a(input_tensor)
            x = self.bn2a(x,training)
            x = tf.nn.relu(x)
            x = self.pool2a(x)
            x = self.conv2b(x)
            x = self.bn2b(x,training)
            x = tf.nn.relu(x)
            x = self.pool2b(x)
            x = layers.Flatten()(x)
            x = self.fc1(x)
            x = self.fc2(x)
            return x
```

(5) 使用 TensorFlow 2.x 中的@tf.function 装饰器来构建计算图。通过 train_one_step()函数实现单步训练的计算图，使用 MSE 损失函数来计算训练

误差，并通过反向传播算法更新网络参数。通过 test() 函数实现测试的计算图，用于评估网络在测试集上的分类精度。

```
@tf.function
def train_one_step(feature_encoder, relation_network, feature_encoder_optim,
relation_network_optim, samples, sample_labels, batches, batch_labels):
    with tf.GradientTape() as feature_encoder_tape, tf.GradientTape()
        as relation_network_tape:
            sample_features = feature_encoder(samples,True)
            batch_features = feature_encoder(batches,True)
            sample_features_ext = tf.repeat(tf.expand_dims(sample_features,
            0),BATCH_NUM_PER_CLASS*CLASS_NUM,axis=0)
        batch_features_ext = tf.repeat(tf.expand_dims(batch_features,0),
        CLASS_NUM,axis=0)
        batch_features_ext = tf.transpose(batch_features_ext,(1,0,2,3,4))
        relation_pairs = tf.reshape(tf.concat([sample_features_ext,batch_
        features_ext],axis=4),(-1,19,19,FEATURE_DIM*2))
        relations = tf.reshape(relation_network(relation_pairs,True),(-1,
        CLASS_NUM))
        mse = tf.keras.losses.MeanSquaredError()
        one_hot_labels = tf.squeeze(batch_labels)
        loss = mse(relations,one_hot_labels)
    grads_feature_encoder = feature_encoder_tape.gradient(loss,
    feature_encoder.trainable_variables)
    grads_relation_network = relation_network_tape.gradient(loss,
    relation_network.trainable_variables)
    feature_encoder_optim.apply_gradients(zip(grads_feature_encoder,
    feature_encoder.trainable_variables))
    relation_network_optim.apply_gradients(zip(grads_relation_network,
    relation_network.trainable_variables))
    return loss
@tf.function
def test(feature_encoder, relation_network, samples, sample_labels, batches,
batch_labels):
    sample_features = feature_encoder(samples,True)
    batch_features = feature_encoder(batches,True)

    sample_features_ext = tf.repeat(tf.expand_dims(sample_features,0),
3*CLASS_NUM,axis=0)
```