- [x] ~~看懂Self-MM原始代码~~
- [ ] 改用我们的方法，修改loss等
- [ ] 这个方法是多任务学习的，我试试多任务（多任务就涉及怎么评估单模态标签的好坏了）和不多任务
- [x] ~~sims数据集，是有单模态标签的，看看他怎么评估生成的单模态标签的好坏~~
- [x] ~~mosi和mosei数据集，没有单模态标签，看看他怎么评估生成的单模态标签的好坏~~

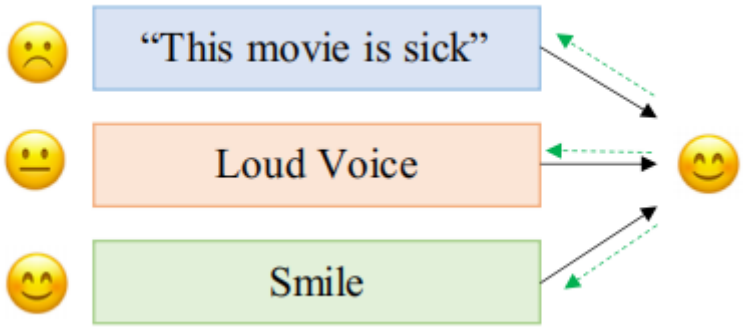
> ~~答：他没有评估，只是说随着epoch增加，趋于稳定。~~

由于统一的多模态注释，现有方法在捕捉多样化信息方面受到限制。然而，额外的单模态注释需要高昂的时间和劳动成本。

在本文中，我们设计了一个基于自监督学习策略的标签生成模块，以获得独立的单模态监督信号。然后，通过联合训练多模态和单模态任务，分别学习一致性和差异性。

此外，在训练阶段，我们设计了一个权重调整策略，以平衡不同子任务之间的学习进度。即指导子任务专注于模态监督信号之间差异较大的样本。

在前瞻指导方法中，研究致力于设计捕获跨模态信息的交互模块（Zadeh等2018a；Sun等2020；Tsai等2019年）。然而，由于统一的多模态注释，它们很难捕获特定于模态的信息。如图1所示，统一的多模态标签并不总是适合于单峰学习。



基于独立的单峰注释和先进的特定模态的表示学习，我们提出了一种新的自监督多任务学习策略。

与Yu等人（2020a）不同，我们的方法不需要人工标注的单峰标签，而是使用自动生成的单峰标签。

它是基于两种直觉的。首先，标签差与模态表征与类中心之间的距离差呈正相关。第二，单模态标签与多模态标签高度相关。因此，我们设计了一个基于多模态标签和模态表示的单峰标签生成模块ULGM。

考虑到自动生成的单峰标签在初始阶段不够稳定，我们设计了一种基于动量的更新方法，对以后生成的单峰标签应用较大的权重。

此外，在集成最终的多任务损失函数时，我们引入了一种自调整策略来调整每个子任务的权重。我们认为，在自动生成的单峰标签和人工注释的多模态标签之间，具有小标签差异的子任务很难学习模式特异性表征。因此，子任务的权重与标签的差异呈正相关。

工作贡献：

1.我们提出了基于模态表示与类中心之间的距离的相对距离值，并与模型输出呈正相关。

2.我们设计了一个基于自监督策略的单峰标签生成模块。此外，还引入了一种新的权重自调整策略来平衡不同的任务损失约束。

3.在三个基准数据集上的大量实验验证了自动生成的单峰标签的稳定性和可靠性。

我们的工作旨在实现基于晚期融合结构的表示学习。与以往的研究不同，我们采用自监督策略联合学习单峰态和多模态任务。

我们的方法从多模态任务中学习相似性信息，从单模态任务中学习差异化信息。

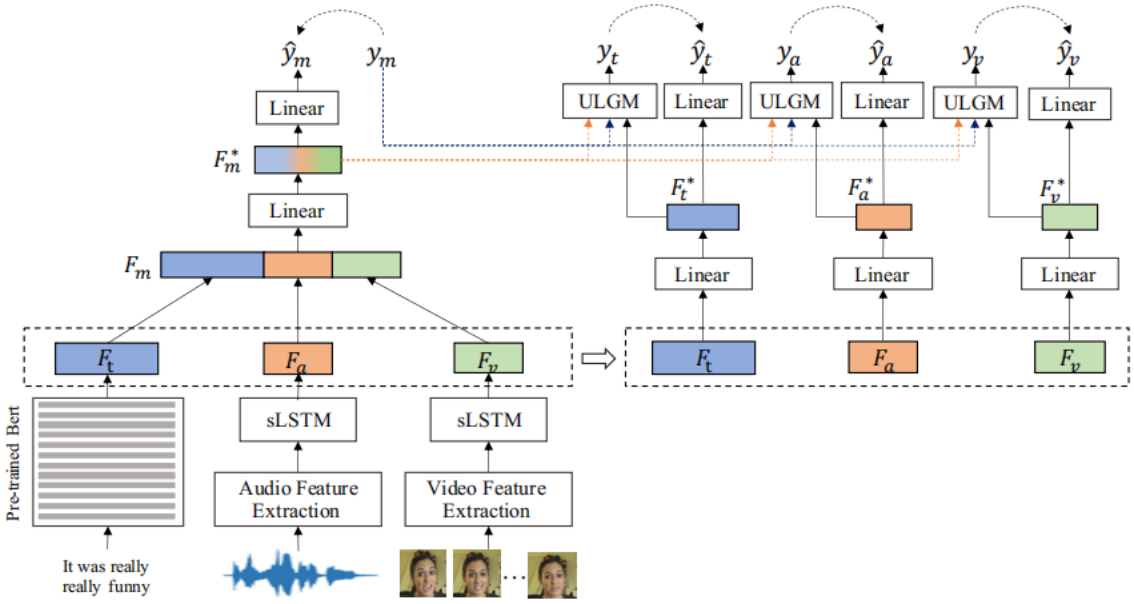
与单任务学习相比，多任务学习在训练阶段存在两个主要挑战。首先是如何共享网络参数，包括硬共享和软共享方法。二是如何平衡不同任务的学习过程。最近，多任务学习在MSA中得到了广泛的应用。在这项工作中，我们引入了单峰子任务来帮助模态特定的表示学习。我们采用了一种硬共享策略，并设计了一种权重调整方法来解决如何平衡的问题。

Self-MM的目标是通过联合学习一个多模态任务和三个单峰子任务来获得信息丰富的单峰表示。

为了方便下面的章节，我们将人工标注的多模态标签称为m型标签，将自动生成的单峰标签称为u型标签。

一般来说，MSA可以看作是回归任务或分类任务。在本工作中，我们将其视为回归任务。

总体结构：



对于多模态任务，我们采用了一种经典的多模态情感分析架构。它包含三个主要部分：特征表示模块、特征融合模块和输出模块。

在文本模态中，由于预训练语言模型的巨大成功，我们使用预训练的12层BERT来提取情感表示。通常，最后一层的第一个词向量被选作整句的表示Ft。

$$F_t = BERT(I_t; \theta_t^{bert}) \in R^{d_t}$$

在音频和视觉模态中，我们遵循Zadeh等人（2017年）；Yu等人（2020b），使用预训练的工具包从原始数据中提取初始向量特征Ia ∈ Rla×da 和 Iv ∈ Rlv×dv。这里，la和lv分别是音频和视觉的序列长度。然后，我们使用单向长短期记忆网络（sLSTM）来捕捉时间特征。最后，采用端状态隐向量作为全序列表示。

$$F_a = sLSTM(I_a; \theta_a^{lstm}) \in R^{d_a}$$
$$F_v = sLSTM(I_v; \theta_v^{lstm}) \in R^{d_v}$$

然后，我们连接所有的单模态表示，并将它们投射到一个低维空间：

$$F_m^* = ReLU(W_{l1}^{m^T}[F_t; F_a; F_v] + b_{l1}^m)$$

思考：把三个特征融合，相当于早期融合这里，他这样就相当于同时有早期晚期融合的思想吧。

最后，利用融合表示法Fm∗来预测多模态情绪：

$$\hat{y}_m = W_{l2}^{m^T} F_m^* + b_{l2}^m$$

单模态任务：

对于这三个单峰任务，它们与多模态任务共享模态表示。为了减少不同模式之间的维度差异，我们将它们投射到一个新的特征空间中。然后，用线性回归得到单模态的结果。

$$F_s^* = ReLU(W_{l1}^{s\,T} F_s + b_{l1}^s)$$

$$\hat{y}_s = W_{l2}^{s\,T} F_s^* + b_{l2}^s$$

where $s \in \{t, a, v\}$.

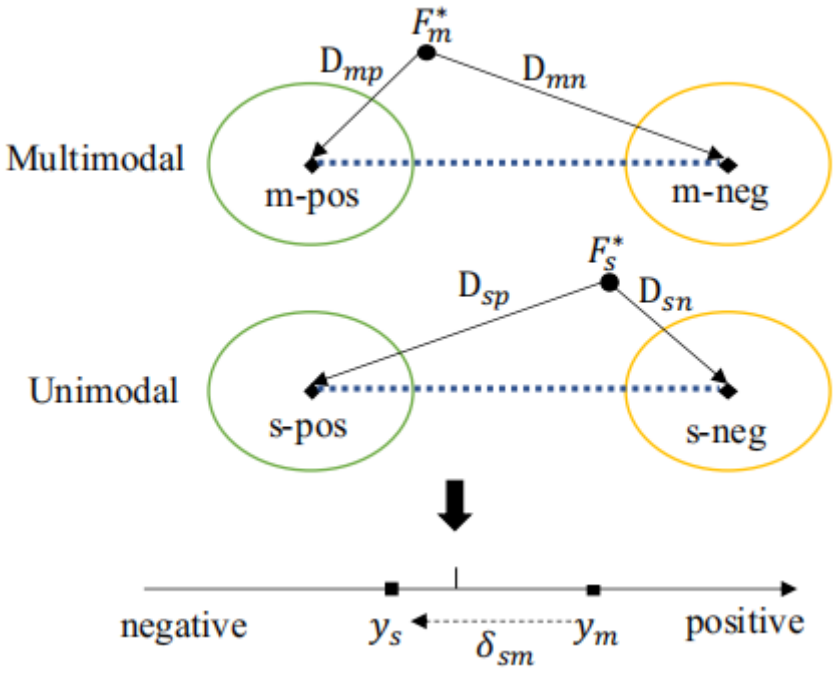
为了指导单峰任务的训练过程，我们设计了一个单峰标签生成模块（ULGM）来获得超标签。

$$y_s = ULGM(y_m, F_m^*, F_s^*)$$

值得注意的是，这些单峰任务只存在于训练阶段。

单模态标签生成模块ULGM：

ULGM旨在生成基于多模态注释和模态表示的单峰监督值。为了避免对网络参数更新的不必要的干扰，ULGM被设计为一个非参数模块。**一般来说，单峰监督值与多模态标签高度相关。因此，ULGM根据从模态表示到类中心的相对距离来计算偏移量，如图所示。**



相对距离值：

由于在不同的特征空间中存在不同的模态表示，因此使用绝对距离值还不够准确。因此，我们提出了相对距离值，它与空间差无关。首先，在训练过程中，我们保持不同模态表征的正中心（Ci p）和负中心（Cin）：

$$C_i^p = \frac{\sum_{j=1}^{N} I(y_i(j)>0)\cdot F_{ij}^g}{\sum_{j=1}^{N} I(y_i(j)>0)} \quad (1)$$

$$C_i^n = \frac{\sum_{j=1}^{N} I(y_i(j)<0)\cdot F_{ij}^g}{\sum_{j=1}^{N} I(y_i(j)<0)} \quad (2)$$

其中i ∈ {m, t, a, v}，N是训练样本的数量，I(·)是一个指示函数。F_i_j_g是模态i中第j个样本的全局表示。

思考：我有一个问题，你这个是只有两个类中心德意思吗，一个label为正的样本的类中心，一个label为负的样本的类中心。如果我不是二分类呢?

对于模态表示，我们使用L2归一化作为Fi*和类中心之间的距离。

$$D_i^p = \frac{||F_i^* - C_i^p||_2^2}{\sqrt{d_i}} \quad (3)$$

$$D_i^n = \frac{||F_i^* - C_i^n||_2^2}{\sqrt{d_i}} \quad (4)$$

把三个特征融合，相当于早期融合这里，他这样就相当于同时有早期晚期融合的思想吧。

$$\alpha_i = \frac{D_i^n - D_i^p}{D_i^p + \epsilon}$$

直觉上，αi与最终结果正相关。为了得到监督值和预测值之间的联系，我们考虑了以下两种关系。

$$\frac{y_s}{y_m} \propto \frac{\hat{y}_s}{\hat{y}_m} \propto \frac{\alpha_s}{\alpha_m} \Rightarrow y_s = \frac{\alpha_s * y_m}{\alpha_m}$$

$$y_s - y_m \propto \hat{y}_s - \hat{y}_m \propto \alpha_s - \alpha_m \Rightarrow y_s = y_m + \alpha_s - \alpha_m$$

引入方程7是为了避免“零值问题”。在方程6中，当ym等于零时，生成的单峰监督值ys始终为零。

然后，结合考虑上述关系，我们可以通过等权值求和得到单峰监督。

$$\begin{aligned} y_s &= \frac{y_m * \alpha_s}{2\alpha_m} + \frac{y_m + \alpha_s - \alpha_m}{2} \\ &= y_m + \frac{\alpha_s - \alpha_m}{2} * \frac{y_m + \alpha_m}{\alpha_m} \\ &= y_m + \delta_{sm} \end{aligned}$$

这个ys的推导就是上边公式6和公式7各自乘1/2再相加，然后通分化简。

基于动量的更新策略：

由于模态表示的动态变化，由公式(8)计算生成的u标签足够不稳定。为了减轻不利影响，我们设计了一个基于动量的更新策略，它将新生成的值与历史值结合起来。

$$y_s^{(i)} = \begin{cases} y_m & i = 1 \\ \frac{i-1}{i+1} y_s^{(i-1)} + \frac{2}{i+1} y_s^i & i > 1 \end{cases}$$

这意味着稍后生成的u型标签的权重大于前一个标签。这是根据我们的经验得出的结论。因为生成的单峰标签是之前所有的时期的累积总和，它们将在足够的迭代后稳定下来（在我们的实验中大约是20个）。然后，单峰式任务的训练过程将逐渐趋于稳定。单峰标签更新策略见算法1。

Algorithm 1 Unimodal Supervisions Update Policy

Input: unimodal inputs I_t, I_a, I_v, m-labels y_m
Output: u-labels $y_t^{(i)}, y_a^{(i)}, y_v^{(i)}$ where i means the number of training epochs

Initialize model parameters $M(\theta; x)$
Initialize u-labels $y_t^{(1)} = y_m, y_a^{(1)} = y_m, y_v^{(1)} = y_m$
Initialize global representations $F_t^g = 0, F_a^g = 0, F_v^g = 0, F_m^g = 0$
for $n \in [1, end]$ **do**
  for mini-batch in dataLoader **do**
    Compute mini-batch modality representations $F_t^*, F_a^*, F_v^*, F_m^*$
    Compute loss L using Equation (10)
    Compute parameters gradient $\frac{\partial L}{\partial \theta}$
    Update model parameters: $\theta = \theta - \eta \frac{\partial L}{\partial \theta}$
    if $n \neq 1$ **then**
      Compute relative distance values $\alpha_m, \alpha_t, \alpha_a$, and α_v using Equation (1∼5)
      Compute y_t, y_a, y_v using Equation (8)
      Update $y_t^{(n)}, y_a^{(n)}, y_t^{(n)}$ using Equation (9)
    end if
    Update global representations F_s^g using F_s^*, where $s \in \{m, t, a, v\}$
  end for
end for

优化目标：

最后，我们使用l1损失作为基本的优化目标。对于单模态任务，我们使用u标签和m标签之间的差值作为损失函数的权重。这说明网络应更加关注差异较大的样本。

$$L = \frac{1}{N}\sum_{i}^{N}(|\hat{y}_m^i - y_m^i| + \sum_{s}^{\{t,a,v\}} W_s^i * |\hat{y}_s^i - y_s^{(i)}|)$$

where N is the number of training samples. $W_s^i = tanh(|y_s^{(i)} - y_m|)$ is the weight of i_{th} sample for auxiliary task s.

周六看懂看懂代码结构

https://blog.csdn.net/qq_51392112/article/details/128806998?fromshare=blogdetail&sharetype=blogdetail&sharerId=128806998&sharerefer=PC&shareresource=wenchlove&sharefrom=from_link

然后周日尝试在他的基础上试试我们的方法

看代码：

评估指标

作者认为MSA任务既可以是分类也可以是回归任务。MSA当作回归任务时，y_pred是个数值，在sims和mosi/mosei数据集上的评估如下：

由于sims的y_true是固定数值的标签值，所以计算acc2时，先把y_true和y_pred数值映射成所在区间序号当作类别号，再计算分类准确率。

acc3，acc5的计算同理。

```
def __eval_sims_regression(self, y_pred, y_true):
    test_preds = y_pred.view(-1).cpu().detach().numpy()
    test_truth = y_true.view(-1).cpu().detach().numpy()
    test_preds = np.clip(test_preds, a_min=-1., a_max=1.)
    test_truth = np.clip(test_truth, a_min=-1., a_max=1.)

    # two classes{[-1.0, 0.0], (0.0, 1.0]}
    ms_2 = [-1.01, 0.0, 1.01]
    test_preds_a2 = test_preds.copy()
    test_truth_a2 = test_truth.copy()
    for i in range(2):
        test_preds_a2[np.logical_and(test_preds > ms_2[i], test_preds <= ms_2[i+1])] =
i
    for i in range(2):
```

```
        test_truth_a2[np.logical_and(test_truth > ms_2[i], test_truth <= ms_2[i+1])] =
i

    # three classes{[-1.0, -0.1], (-0.1, 0.1], (0.1, 1.0]}
    ms_3 = [-1.01, -0.1, 0.1, 1.01]
    test_preds_a3 = test_preds.copy()
    test_truth_a3 = test_truth.copy()
    for i in range(3):
        test_preds_a3[np.logical_and(test_preds > ms_3[i], test_preds <= ms_3[i+1])] =
i
    for i in range(3):
        test_truth_a3[np.logical_and(test_truth > ms_3[i], test_truth <= ms_3[i+1])] =
i

    # five classes{[-1.0, -0.7], (-0.7, -0.1], (-0.1, 0.1], (0.1, 0.7], (0.7, 1.0]}
    ms_5 = [-1.01, -0.7, -0.1, 0.1, 0.7, 1.01]
    test_preds_a5 = test_preds.copy()
    test_truth_a5 = test_truth.copy()
    for i in range(5):
        test_preds_a5[np.logical_and(test_preds > ms_5[i], test_preds <= ms_5[i+1])] =
i
    for i in range(5):
        test_truth_a5[np.logical_and(test_truth > ms_5[i], test_truth <= ms_5[i+1])] =
i

    mae = np.mean(np.absolute(test_preds - test_truth))   # Average L1 distance between
preds and truths
    corr = np.corrcoef(test_preds, test_truth)[0][1]
    mult_a2 = self.__multiclass_acc(test_preds_a2, test_truth_a2)
    mult_a3 = self.__multiclass_acc(test_preds_a3, test_truth_a3)
    mult_a5 = self.__multiclass_acc(test_preds_a5, test_truth_a5)
    f_score = f1_score(test_truth_a2, test_preds_a2, average='weighted')

    eval_results = {
        "Mult_acc_2": mult_a2,
        "Mult_acc_3": mult_a3,
        "Mult_acc_5": mult_a5,
        "F1_score": f_score,
        "MAE": mae,
        "Corr": corr, # Correlation Coefficient
    }
    return eval_results
```

mosi/mosei数据集的y_true是随机数值，在计算acc7时，先把y_true和y_pred映射到（-3，3）内，然后将数值四舍五入到最近的整数,然后把这个整数当作它的类别号，再计算准确率。

```
def __eval_mosei_regression(self, y_pred, y_true, exclude_zero=False):
    test_preds = y_pred.view(-1).cpu().detach().numpy()
    test_truth = y_true.view(-1).cpu().detach().numpy()

    test_preds_a7 = np.clip(test_preds, a_min=-3., a_max=3.)
```

```
    test_truth_a7 = np.clip(test_truth, a_min=-3., a_max=3.)
    test_preds_a5 = np.clip(test_preds, a_min=-2., a_max=2.)
    test_truth_a5 = np.clip(test_truth, a_min=-2., a_max=2.)
    test_preds_a3 = np.clip(test_preds, a_min=-1., a_max=1.)
    test_truth_a3 = np.clip(test_truth, a_min=-1., a_max=1.)


    mae = np.mean(np.absolute(test_preds - test_truth))   # Average L1 distance between
preds and truths
    corr = np.corrcoef(test_preds, test_truth)[0][1]
    mult_a7 = self.__multiclass_acc(test_preds_a7, test_truth_a7)
    mult_a5 = self.__multiclass_acc(test_preds_a5, test_truth_a5)
    mult_a3 = self.__multiclass_acc(test_preds_a3, test_truth_a3)

    non_zeros = np.array([i for i, e in enumerate(test_truth) if e != 0])
    non_zeros_binary_truth = (test_truth[non_zeros] > 0)
    non_zeros_binary_preds = (test_preds[non_zeros] > 0)

    non_zeros_acc2 = accuracy_score(non_zeros_binary_preds, non_zeros_binary_truth)
    non_zeros_f1_score = f1_score(non_zeros_binary_truth, non_zeros_binary_preds,
average='weighted')

    binary_truth = (test_truth >= 0)
    binary_preds = (test_preds >= 0)
    acc2 = accuracy_score(binary_preds, binary_truth)
    f_score = f1_score(binary_truth, binary_preds, average='weighted')

    eval_results = {
        "Has0_acc_2":  round(acc2, 4),
        "Has0_F1_score": round(f_score, 4),
        "Non0_acc_2":  round(non_zeros_acc2, 4),
        "Non0_F1_score": round(non_zeros_f1_score, 4),
        "Mult_acc_5": round(mult_a5, 4),
        "Mult_acc_7": round(mult_a7, 4),
        "MAE": round(mae, 4),
        "Corr": round(corr, 4)
    }
    return eval_results
```

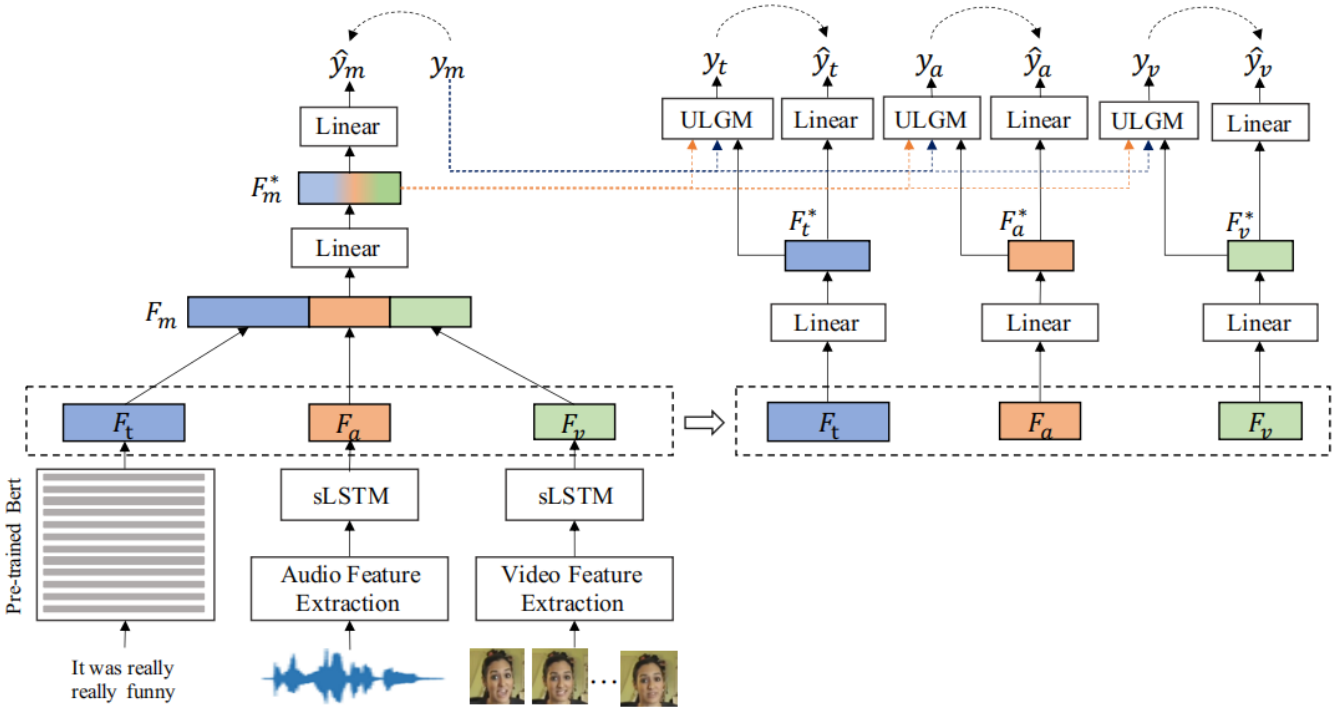
相较于损失函数时使用loss = torch.mean(weighted * torch.abs(y_pred - y_true))，在评估指标时，acc的判定更为宽松了，不过mae的判定倒是还是一样的。

bug太多了，下周再复现这个，我头麻了看的我。

终于能运行了，改着改着突然能运行了，头昏了忘记改的啥东西了。。。。

网络结构代码：

结合下图看更容易理解：




```
#定义特征三个模态的特征提取网络
# text subnets
self.aligned = args.need_data_aligned
self.text_model = BertTextEncoder(language=args.language,
use_finetune=args.use_finetune)
# audio-vision subnets
#args.feature_dims (768, 33, 709)
audio_in, video_in = args.feature_dims[1:]
self.audio_model = AuViSubNet(audio_in, args.a_lstm_hidden_size, args.audio_out, \
    num_layers=args.a_lstm_layers, dropout=args.a_lstm_dropout)
self.video_model = AuViSubNet(video_in, args.v_lstm_hidden_size, args.video_out, \
    num_layers=args.v_lstm_layers, dropout=args.v_lstm_dropout)
```

先特征级融合，然后用融合后的特征向量，预测一个回归值：

```
# the post_fusion layers
self.post_fusion_dropout = nn.Dropout(p=args.post_fusion_dropout)
self.post_fusion_layer_1 = nn.Linear(args.text_out + args.video_out + args.audio_out,
args.post_fusion_dim)
self.post_fusion_layer_2 = nn.Linear(args.post_fusion_dim, args.post_fusion_dim)
self.post_fusion_layer_3 = nn.Linear(args.post_fusion_dim, 1)
```

只使用text模态的特征向量，预测一个回归值：

```
# the classify layer for text
self.post_text_dropout = nn.Dropout(p=args.post_text_dropout)
self.post_text_layer_1 = nn.Linear(args.text_out, args.post_text_dim)
self.post_text_layer_2 = nn.Linear(args.post_text_dim, args.post_text_dim)
```

```
self.post_text_layer_3 = nn.Linear(args.post_text_dim, 1)
```

只使用audio模态的特征向量，预测一个回归值：

```
# the classify layer for audio
self.post_audio_dropout = nn.Dropout(p=args.post_audio_dropout)
self.post_audio_layer_1 = nn.Linear(args.audio_out, args.post_audio_dim)
self.post_audio_layer_2 = nn.Linear(args.post_audio_dim, args.post_audio_dim)
self.post_audio_layer_3 = nn.Linear(args.post_audio_dim, 1)
```

只使用video模态的特征向量，预测一个回归值：

```
# the classify layer for video
self.post_video_dropout = nn.Dropout(p=args.post_video_dropout)
self.post_video_layer_1 = nn.Linear(args.video_out, args.post_video_dim)
self.post_video_layer_2 = nn.Linear(args.post_video_dim, args.post_video_dim)
self.post_video_layer_3 = nn.Linear(args.post_video_dim, 1)
```

都是特征向量，dropout,线性层，relu，线性层，relu，线性层然后得出一个回归值。

前向传播代码：

```
def forward(self, text, audio, video):
    audio, audio_lengths = audio
    #shape=(32,16)=(batch_size,audio_out)
    video, video_lengths = video
    #shape=(32,32)=(batch_size,video_out)
    mask_len = torch.sum(text[:,1,:], dim=1, keepdim=True)
    text_lengths = mask_len.squeeze().int().detach().cpu()


    text = self.text_model(text)[:,0,:]
    #shape=(32,768)=(batch_size,text_out)
    if self.aligned:
        audio = self.audio_model(audio, text_lengths)
        video = self.video_model(video, text_lengths)
    else:
        audio = self.audio_model(audio, audio_lengths)
        video = self.video_model(video, video_lengths)
    # 从现在开始，我们获得了三个初始Feature,存放在text，audio，video里

    t=text
    a=audio
    v=video
    tcat=torch.cat([text, audio, video], dim=-1)

    # 定义将三个模态的特征向量投影到一个新的特征空间中（为了减少不用模态间维度差异，以便于ULGM模块的使用）
```

```
    #然后为了让三个模态的特征向量的维度差异尽量小（以便于ULGM模块），将它们投射到一个新的特征空间中。
    # fusion
    fusion_h = torch.cat([text, audio, video], dim=-1)
    fusion_h = self.post_fusion_dropout(fusion_h) #(32,128)
    fusion_h = F.relu(self.post_fusion_layer_1(fusion_h), inplace=False)
    # # text
    tt=self.post_text_dropout(text)
    text_h = self.post_text_dropout(text) #(32,64)
    text_h = F.relu(self.post_text_layer_1(text_h), inplace=False)
    # audio
    audio_h = self.post_audio_dropout(audio) #(32,16)
    audio_h = F.relu(self.post_audio_layer_1(audio_h), inplace=False)
    # vision
    video_h = self.post_video_dropout(video) #(32,32)
    video_h = F.relu(self.post_video_layer_1(video_h), inplace=False)

    # classifier-fusion
    x_f = F.relu(self.post_fusion_layer_2(fusion_h), inplace=False)
    output_fusion = self.post_fusion_layer_3(x_f)

    # classifier-text
    x_t = F.relu(self.post_text_layer_2(text_h), inplace=False)
    output_text = self.post_text_layer_3(x_t)

    # classifier-audio
    x_a = F.relu(self.post_audio_layer_2(audio_h), inplace=False)
    output_audio = self.post_audio_layer_3(x_a)

    # classifier-vision
    x_v = F.relu(self.post_video_layer_2(video_h), inplace=False)
    output_video = self.post_video_layer_3(x_v)

    res = {
        'M': output_fusion,
        'T': output_text,
        'A': output_audio,
        'V': output_video,
        'Feature_t': text_h,
        'Feature_a': audio_h,
        'Feature_v': video_h,
        'Feature_f': fusion_h,
    }
    return res
```

对照着他的代码，我改了一下：

```
#ECML那篇的证据提取网络，输入特征向量，输出证据向量
class EvidenceCollector(nn.Module):
    def __init__(self, dims, num_classes):
        super(EvidenceCollector, self).__init__()
```

```
        self.num_layers = len(dims)
        self.net = nn.ModuleList()
        for i in range(self.num_layers - 1):
            self.net.append(nn.Linear(dims[i], dims[i + 1]))
            self.net.append(nn.ReLU())
            self.net.append(nn.Dropout(0.1))
        self.net.append(nn.Linear(dims[self.num_layers - 1], num_classes))
        self.net.append(nn.BatchNorm1d(num_classes)) #加了个正则化
        self.net.append(nn.Softplus())

    def forward(self, x):
        h = self.net[0](x)
        for i in range(1, len(self.net)):
            h = self.net[i](h)
        return h
```

#根据Self-MM这篇，我改了一下 1.首先三个模态的特征提取网络，我一下没改

2.ULGM需要让三个模态的特征向量的维度差异尽量小，因此我也沿用了他将原始特征向量投影到一个新的特征空间中

3.现在处理完的特征向量，输入ECML的证据网络获得证据向量

4.我虽然保留了他把三个模态的原始特征向量融合起来当一个‘总特征向量’（仅仅是因为ULGM需要这个东西），但我删除了使用‘总特征向量’获得多模态任务的预测结果的部分，改用了我们的方法获得多模态任务的预测结果evidence_a。

5.输出由原来的回归值改成了证据向量，所以还要改别的

```
class SELF_MM_copy(nn.Module):
    def __init__(self, args):

        print("self_mm_copy")
        super(SELF_MM_copy, self).__init__()

        # 定义特征三个模态的特征提取网络，这块我不能改
        # text subnets
        self.aligned = args.need_data_aligned
        self.text_model = BertTextEncoder(language=args.language,
use_finetune=args.use_finetune)
        # audio-vision subnets
        #args.feature_dims (768, 33, 709)
        audio_in, video_in = args.feature_dims[1:]
        self.audio_model = AuViSubNet(audio_in, args.a_lstm_hidden_size,
args.audio_out, \
                                      num_layers=args.a_lstm_layers, dropout=args.a_lstm_dropout)
        self.video_model = AuViSubNet(video_in, args.v_lstm_hidden_size,
args.video_out, \
                                      num_layers=args.v_lstm_layers, dropout=args.v_lstm_dropout)
```

```
        # 定义3个证据网络，特征向量跑完这个网络可以得到每个模态的证据向量
        self.num_views=3
        self.num_classes=3
        dims = [[args.text_out], [args.audio_out], [args.video_out]]
        self.EvidenceCollectors = nn.ModuleList([EvidenceCollector(dims[i],
self.num_classes) for i in range(self.num_views)])
        # 这个dims[i]是第i个视图的数据的维度，这里是定义了num_views个单模态网络


    def forward(self, text, audio, video):

        #print("self_mm_copy forward")
        audio, audio_lengths = audio
        #shape=(32,16)=(batch_size,audio_out)
        video, video_lengths = video
        #shape=(32,32)=(batch_size,video_out)
        mask_len = torch.sum(text[:,1,:], dim=1, keepdim=True)
        text_lengths = mask_len.squeeze().int().detach().cpu()
        text = self.text_model(text)[:,0,:]
        #shape=(32,768)=(batch_size,text_out)
        if self.aligned:
            audio = self.audio_model(audio, text_lengths)
            video = self.video_model(video, text_lengths)
        else:
            audio = self.audio_model(audio, audio_lengths)
            video = self.video_model(video, video_lengths)

        X=[text,audio,video]
        evidences=dict()
        for v in range(self.num_classes):
            evidences[v]=self.EvidenceCollectors[v](X[v])
        #ABF
        evidence_a=evidences[0]
        for i in range(1, self.num_views):
            evidence_a = (evidences[i] + evidence_a)
        evidence_a=evidence_a/self.num_views
        #print('evidences',evidences)
        #print('evidence_a',evidence_a)
        return evidences,evidence_a
```

模型训练和验证实际运行的代码块：

```
def do_train(self,model,dataloader):

    # 参数分组与优化器设置
    bert_no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
    bert_params = list(model.Model.text_model.named_parameters())
    audio_params = list(model.Model.audio_model.named_parameters())
    video_params = list(model.Model.video_model.named_parameters())
```

```
    bert_params_decay = [p for n, p in bert_params if not any(nd in n for nd in
bert_no_decay)]
    bert_params_no_decay = [p for n, p in bert_params if any(nd in n for nd in
bert_no_decay)]
    audio_params = [p for n, p in audio_params]
    video_params = [p for n, p in video_params]
    model_params_other = [p for n, p in list(model.Model.named_parameters()) if
'text_model' not in n and \
                          'audio_model' not in n and 'video_model' not in n]

    optimizer_grouped_parameters = [
        {'params': bert_params_decay, 'weight_decay': self.args.weight_decay_bert,
         'lr': self.args.learning_rate_bert},
        {'params': bert_params_no_decay, 'weight_decay': 0.0, 'lr':
self.args.learning_rate_bert},
        {'params': audio_params, 'weight_decay': self.args.weight_decay_audio, 'lr':
self.args.learning_rate_audio},
        {'params': video_params, 'weight_decay': self.args.weight_decay_video, 'lr':
self.args.learning_rate_video},
        {'params': model_params_other, 'weight_decay': self.args.weight_decay_other,
         'lr': self.args.learning_rate_other}
    ]
    optimizer = optim.Adam(optimizer_grouped_parameters)

    annealing_step=self.args.annealing_step
    gamma = self.args.gamma
    device=self.args.device
    model.to(device)
    acc_max = 0

    print('acc_max',acc_max)

    for epoch in range(1, self.args.epochs + 1):
        print(f'====> {epoch}')
        for phase in ["train", "valid"]:
            if phase == "train":
                print('now is batch_size train')
                model.train()
                num_correct, num_sample = 0, 0
                with tqdm(dataloader['train']) as td:
                    for batch_data in td:
                        train_batch_data=batch_data
                        Y = batch_data['labels']
['M'].to(self.args.device).long().squeeze()
                        #print('Y',Y)
                        vision = batch_data['vision'].to(self.args.device)
                        audio = batch_data['audio'].to(self.args.device)
                        text = batch_data['text'].to(self.args.device)
                        #indexes = batch_data['index'].view(-1)
                        #cur_id = batch_data['id']
                        #ids.extend(cur_id)

                        if not self.args.need_data_aligned:
```

```
                        audio_lengths =
batch_data['audio_lengths'].to(self.args.device)
                        vision_lengths =
batch_data['vision_lengths'].to(self.args.device)
                    else:
                        audio_lengths, vision_lengths = 0, 0

                    evidences, evidence_a = model(text, (audio, audio_lengths),
(vision, vision_lengths))
                    _, Y_pre = torch.max(evidence_a, dim=1)
                    num_correct += (Y_pre == Y).sum().item()
                    num_sample += Y.shape[0]
                    #print('Y_pre',Y_pre)
                    # print('evidences', evidences)
                    # print('evidence_a', evidence_a)

                    loss = get_loss(evidences, evidence_a, Y, epoch, num_classes=3,
annealing_step=annealing_step, gamma=gamma,device=device)
                    optimizer.zero_grad()
                    loss.backward()
                    optimizer.step()
                print('train:','num_correct:', num_correct, 'num_sample',
num_sample)  # 出问题了这里，晚上再debug
                acc = num_correct / num_sample
                print('train acc:',acc)
            elif phase == "valid":
                print('now is batch_size valid')
                model.eval()
                num_correct, num_sample = 0, 0
                with torch.no_grad():
                    with tqdm(dataloader['valid']) as td:
                        for batch_data in td:
                            valid_batch_data=batch_data
                            Y = batch_data['labels']
['M'].to(self.args.device).long().squeeze()
                            vision = batch_data['vision'].to(self.args.device)
                            audio = batch_data['audio'].to(self.args.device)
                            text = batch_data['text'].to(self.args.device)
                            #indexes = batch_data['index'].view(-1)
                            #cur_id = batch_data['id']
                            #ids.extend(cur_id)

                            if not self.args.need_data_aligned:
                                audio_lengths =
batch_data['audio_lengths'].to(self.args.device)
                                vision_lengths =
batch_data['vision_lengths'].to(self.args.device)
                            else:
                                audio_lengths, vision_lengths = 0, 0

                            evidences, evidence_a = model(text, (audio, audio_lengths),
(vision, vision_lengths))
                            _, Y_pre = torch.max(evidence_a, dim=1)
```

```
                    num_correct += (Y_pre == Y).sum().item()
                    num_sample += Y.shape[0]
                print('num_correct:',num_correct,'num_sample',num_sample) #出问
题了这里，晚上再debug
                acc = num_correct / num_sample
                if acc > acc_max:
                    acc_max = acc
                    acc_max_epoch = epoch
                    # save model
                    torch.save(model.cpu().state_dict(),
self.args.model_save_path)
                    model.to(self.args.device)

        print('====> acc_max: {:.4f}'.format(acc_max))

    return acc_max
```

模型测试运行的代码块：

```
def do_test(self, model, dataloader):
    print('now is test')
    model.eval()
    num_correct, num_sample = 0, 0
    with torch.no_grad(): #这个放外面更好
        with tqdm(dataloader['test']) as td:
            for batch_data in td:

                Y = batch_data['labels']['M'].to(self.args.device).long().squeeze()
#这里出了点问题！！！！！！！！！
                vision = batch_data['vision'].to(self.args.device)
                audio = batch_data['audio'].to(self.args.device)
                text = batch_data['text'].to(self.args.device)
                if not self.args.need_data_aligned:
                    audio_lengths = batch_data['audio_lengths'].to(self.args.device)
                    vision_lengths = batch_data['vision_lengths'].to(self.args.device)
                else:
                    audio_lengths, vision_lengths = 0, 0

                evidences, evidence_a = model(text, (audio, audio_lengths), (vision,
vision_lengths))
                _, Y_pre = torch.max(evidence_a, dim=1)
                num_correct += (Y_pre == Y).sum().item()
                num_sample += Y.shape[0]
            acc_test = num_correct / num_sample
    return acc_test
```

出问题了，现在在train_set上acc能到99%，但是在valid_set上acc才59%，然后在test_set上也是只有59%，应该是过拟合了。