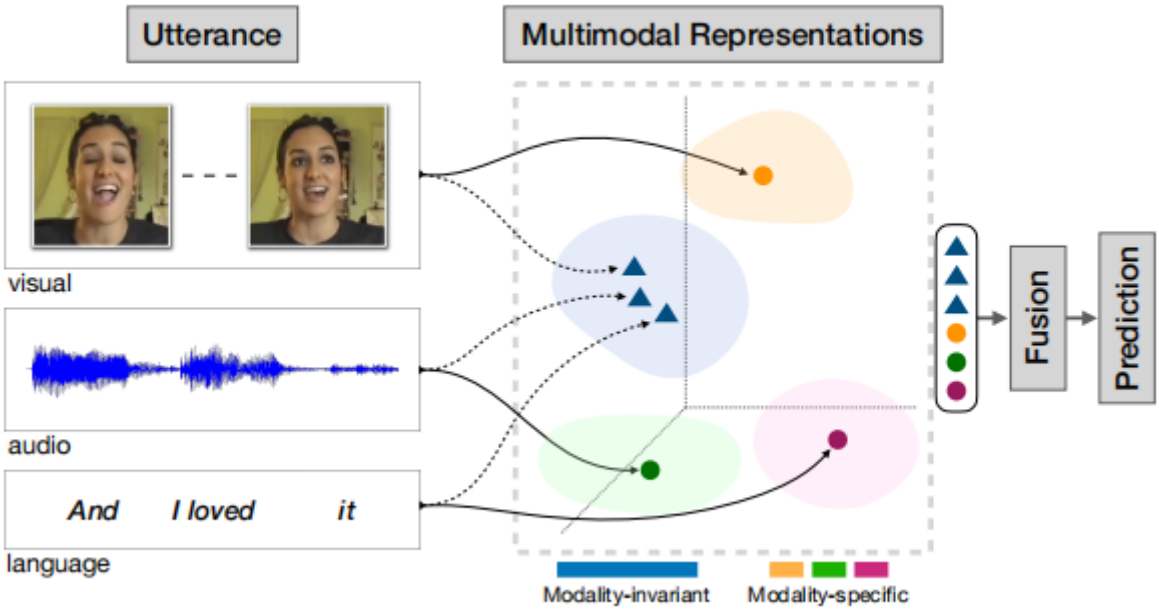
MISA：多模态情绪分析的模态不变和特定表征

我们提出了一个新颖的框架，MISA（Modality-Invariant and -Specific Representations for Multimodal Sentiment Analysis），将每个模态投射到两个不同的子空间。第一个子空间是模态不变的，其中跨模态的表示学习它们的共性并减少模态差距。第二个子空间是特定模态的，这是每个模态独有的，捕捉它们的特征特征。这些表示提供了对多模态数据的全面视图，用于融合，从而进行任务预测。我们在流行的情感分析基准数据集MOSI和MOSEI上进行的实验显示了相较于现有最先进模型的显著优势。

第一种表示是模态不变的，旨在减少模态差距。在这里，所有模态的表达都被映射到一个共享的子空间中，该子空间具有分布对齐的特性。尽管多模态信号来自不同的来源，但它们分享了说话者的共同动机和目标，这决定了表达的整体情感状态。不变的映射有助于捕捉这些潜在的共性和相关特征，作为在共享子空间上的对齐投影。大多数先前的研究在融合之前没有利用这种对齐，这使得他们的融合过程需要额外负担，以桥接模态差距并学习共同特征。

除了不变的子空间外，MISA还学习特定模态的特征，这些特征是每个模态独有的。对于任何表达，每个模态都有独特的特征，包括对说话人敏感的风格化信息。这样的个性化细节通常与其他模态无关，并被归类为噪声。然而，它们在预测情感状态时可能很有用——例如，说话人倾向于讽刺或带有偏向情感极性的特定表达。因此，学习这些特定模态的特征补充了不变空间中捕获的共同潜在特征，并提供一个全面的多模态表示。



上图:通过模态不变和特定子空间学习多模态表示。这些特征后来被用于融合和随后在视频中的影响预测。

为了学习这些子空间，我们合并了一系列损失的组合，包括分布相似性损失（对于不变特征）、正交损失（对于特定特征）、重建损失（对于模态特征的代表性）和任务预测损失。

多模态表示学习：

公共子空间表示

尝试学习跨模态公共子空间的工作可以大致分为：

（i）基于翻译的模型，使用序列到序列[40]、循环翻译[39]和对抗性自动编码器[30]等方法将一种模态转换为另一种模式；

（ii）基于相关的模型[50]，使用典型相关分析[3]学习跨模态相关；

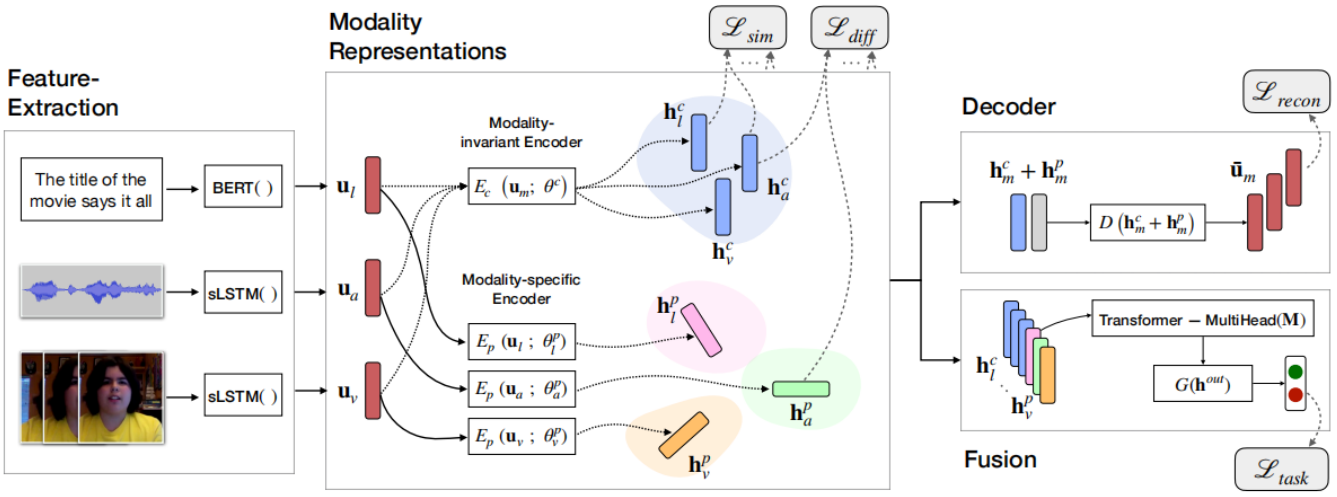
（iii）学习一个新的共享子空间，其中所有模态同时映射，使用对抗学习[35,37]等技术。

与第三类类似，我们也学习了共同的模态不变子空间。然而，我们不使用对抗性鉴别器来学习共享映射。此外，我们还结合了正交模态特定表征——这是在多模态学习任务中较少探索的特征。

因子化表示

在子空间学习的框架内，我们将重点转向因式分解的表示。尽管有一项研究工作试图学习多模态数据的生成-判别因子[51]，我们关注的是学习模态不变和特定的表示。为了实现这一点，我们从有关共享-私有表示的相关文献中汲取灵感。共享-私有[5]学习的起源可以在多视图组件分析[48]中找到。这些早期的工作设计了具有单独共享和私有潜在变量[9]的潜在变量模型（LVMs）。Wang等人[55]通过提出一种概率CCA-深度变分CCA，重新审视了这一框架。与这些模型不同，我们的建议涉及到一个有区别的深度神经结构，它避免了近似推理的需要。

==我们的框架与领域分离网络（Domain Separation Network，DSN）[5]密切相关，后者提出了用于域适应的共享-私有模型。==DSN在多任务文本分类等类似模型的开发中产生了重要影响[25]。虽然我们从DSN中获得灵感，但MISA包含了一些关键区别：（ｉ）DSN学习跨实例的因式分解表示，而MISA则学习模态内的实例（话语）表示；（ii）与DSN不同，我们使用更先进的分布相似性度量——CMD（见第3.5节），而不是对抗训练或MMD；（iii）我们在模态特有（私有）表示之间引入了额外的正交损失（见第3.5.2节）；（iv）最后，虽然DSN仅使用共享表示进行任务预测，但MISA结合了不变和特定表示进行融合，然后进行任务预测。我们认为，提供模态表示有助于通过提供多模态数据的全面视图来辅助融合。



MISA的功能可以分为两个主要阶段：模态表示学习和模态融合。整个框架如上图所示。

模态表示学习：

话语层面的表示

对每个模态的话语Um，使用一个堆叠的双向LSTM，其末端隐藏表示与一个全连接层结合，生成um：

$$\mathbf{u}_m = \text{sLSTM}\left(\mathbf{U}_m;\ \theta_m^{lstm}\right) \qquad (1)$$

模态不变和特定表示

我们现在将每个话语向量um投射到两种不同的表示形式中。

首先是模态不变成分，它在具有分布相似性约束[18]的公共子空间中学习共享表示。这种约束有助于最小化异质性差距——这是多模态融合所需的理想特性。

其次是模态特定成分，它捕捉该模态的独特特征。

在本文中，我们认为模态不变和特定的表示的存在提供了有效的融合所需的整体视图。学习这些表示是我们工作的主要目标。

给定模态m的话语向量um，使用编码函数学习隐藏的模态不变表示和模态特定表示：

$$\mathbf{h}_m^c = E_c\left(\mathbf{u}_m;\theta^c\right),\quad \mathbf{h}_m^p = E_p\left(\mathbf{u}_m;\theta_m^p\right) \qquad (2)$$

为了生成6个隐藏向量，h^(c/p)_(l/v/a)，我们使用简单的前馈神经网络，Ec在三个模态间共享参数θc，Ep为每个模态分配单独的θp。

模态融合

在将这些模式投影到它们各自的表示中之后，我们将它们融合成一个联合向量，用于下游预测。

我们设计了一个简单的融合机制，首先执行一个基于Transformer[54]的自注意，然后将所有6个转换后的模态向量串联起来。

定义Transformer

Transformer利用一个注意模块，定义为一个缩放点积函数：

$$Attention(Q, K, V) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_h}}\right)\mathbf{V} \quad (3)$$

Transformer计算多个这样的平行注意，其中每个注意输出被称为一个头部。

第i个注意力头的计算为：

$$head_i = Attention(\mathbf{Q}W_i^q, \mathbf{K}W_i^k, \mathbf{V}W_i^v) \quad (4)$$

$W_i^{q/k/v} \in \mathbb{R}^{d_h \times d_h}$ *are head-specific parameters to linearly project the matrices into local spaces.*

融合过程

首先将6个隐藏向量堆叠：

$$\mathbf{M} = [\mathbf{h}_l^c, \mathbf{h}_v^c, \mathbf{h}_a^c, \mathbf{h}_l^p, \mathbf{h}_v^p, \mathbf{h}_a^p]$$

$$\mathbb{R}^{6 \times d_h}$$

然后，我们对这些表示进行多头自注意（multi-headed self-attention），使每个向量意识到其他的跨模态（和跨子空间）表示。

这样做可以让每个表征从同伴表征中诱导潜在的信息，这些信息对整体的情感取向是协同的。这种跨模态匹配在最近的跨模态学习方法[22,23,27,49,57]中非常突出。

为了自我注意力，我们设置

$$\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{M} \in \mathbb{R}^{6 \times d_h}$$

Transformer生成一个新的矩阵M_hat:

$$\bar{\mathbf{M}} = \text{MultiHead}(\mathbf{M}; \theta^{att}) = (\text{head}_1 \oplus \cdots \oplus \text{head}_n)W^o \quad (5)$$

每一个headi都是基于公式4计算的，⊕表示连接，θatt={Wq，Wk，Wv，Wo}。

$$\bar{\mathbf{M}} = [\bar{\mathbf{h}}_l^c, \bar{\mathbf{h}}_v^c, \bar{\mathbf{h}}_a^c, \bar{\mathbf{h}}_l^p, \bar{\mathbf{h}}_v^p, \bar{\mathbf{h}}_a^p]$$

预测/推理

连接6个h_hat：

Prediction/Inference. Finally, we take the Transformer output and construct a joint-vector using concatenation, $\mathbf{h}^{out} = [\bar{\mathbf{h}}_l^c \oplus \cdots \oplus \bar{\mathbf{h}}_a^p] \in \mathbb{R}^{6d_h}$. The task predictions are then generated by the function $\hat{\mathbf{y}} = G(\mathbf{h}^{out}; \theta^{out})$.

We provide the network topology of the functions $sLSTM()$, $E_c()$, $E_p()$, $G()$ and $D()$ (explained later) in the appendix.

学习

该模型的整体学习是通过最小化L来完成的：

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \alpha\,\mathcal{L}_{\text{sim}} + \beta\,\mathcal{L}_{\text{diff}} + \gamma\,\mathcal{L}_{\text{recon}} \tag{6}$$

α、β，γ是决定每个正则化分量对总体损失l的贡献的交互权重。这些分量损失中的每一个都负责实现所期望的子空间属性。我们接下来讨论它们。

Lsim-相似性损失：

==最小化相似性损失减少了每个模态的共享表示之间的差异。这有助于在共享子空间中对齐在一起。==在许多选择中，我们为此使用中心矩差（CMD）[63]度量。CMD是一种最先进的距离度量方法，它通过匹配两个表示形式的顺序矩差来衡量它们的分布之间的差异。直观地看，CMD距离随着两个分布变得更加相似而减小。

CMD定义：

设X和Y为有界随机样本，在区间[a，b]^N上的概率分布分别为p和q。中心矩差正则化器CMDK被定义为CMD度量的经验估计，由

$$CMD_K(X,Y) = \frac{1}{|b-a|}\|\mathbf{E}(X) - \mathbf{E}(Y)\|_2 + \sum_{k=2}^{K} \frac{1}{|b-a|^k}\|C_k(X) - C_k(Y)\|_2 \tag{7}$$

$$\mathbf{E}(X) = \frac{1}{|X|}\sum_{x\in X} x$$

是样本的经验期望向量。

$$C_k(X) = \mathbf{E}\left((x - \mathbf{E}(X))^k\right)$$

是X坐标的所有kth阶样本中心矩的向量。

在我们的例子中，我们计算了每一对模态的不变表示之间的CMD损失：

$$\mathcal{L}_{\text{sim}} = \frac{1}{3} \sum_{\substack{(m_1,m_2) \in \\ \{(l,a),(l,v),\\(a,v)\}}} CMD_K(\mathbf{h}^c_{m_1}, \mathbf{h}^c_{m_2}) \tag{8}$$

在这里，我们做了两个重要的观察：

（i）我们选择CMD而不是KL散度或MMD，因为CMD是一种流行的度量[36]，它可以在没有昂贵的距离和核矩阵计算的情况下执行高阶矩的显式匹配。

（ii）对抗性损失是相似性训练的另一种选择，其中一个鉴别器和共享编码器参与了一个极大极小博弈。但是，我们选择CMD是因为它的公式很简单。相比之下，对抗性训练需要鉴别器的额外参数，以及增加的复杂性，如训练[53]中的振荡。

Ldiff-差异性损失：

这种损失是为了确保模态不变表示（modality-invariant-representations）和模态特定表示（modality-specific-representations）捕获了输入的不同方面。

非冗余是通过在两个表示[5,25,47]之间强制执行软正交约束来实现的。在一training batch的话语中，设Hmc和Hmp为矩阵（我们将矩阵变换为零均值和单位l2范数。），其行表示每个话语的模态m的隐藏向量hmc和hmp。然后计算出该模态向量对的正交性约束为：

$$\left\| \mathbf{H}^{c^\top}_m \mathbf{H}^p_m \right\|^2_F$$

$\|\cdot\|F$2是弗罗比尼乌斯范数。除了不变向量hmc和特定向量hmp之间的约束外，我们还增加了特定模态向量之间的正交约束。

$$\mathcal{L}_{\text{diff}} = \sum_{m \in \{l,v,a\}} \left\| \mathbf{H}^{c^\top}_m \mathbf{H}^p_m \right\|^2_F + \sum_{\substack{(m_1,m_2) \in \\ \{(l,a),(l,v),\\(a,v)\}}} \left\| \mathbf{H}^{p^\top}_{m_1} \mathbf{H}^p_{m_2} \right\|^2_F \tag{10}$$

Lrecon-重建损失：

由于差异损失被强制执行，仍然存在学习模态特定编码器生成平凡表示的风险。如果编码函数近似于一个正交但不具代表性的模态向量，就可能出现平凡情况。为了避免这种情况，我们添加了一个重建损失，以确保隐藏的表示来捕获它们各自模态的细节。首先，我们利用解码器函数

$$\hat{\mathbf{u}}_m = D(\mathbf{h}^c_m + \mathbf{h}^p_m; \theta^d)$$

重构模态向量um，重构损失为um和u^m之间的均方误差损失：

$$\mathcal{L}_{\text{recon}} = \frac{1}{3}\left(\sum_{m\in\{l,v,a\}} \frac{\|\mathbf{u}_m - \hat{\mathbf{u}}_m\|_2^2}{d_h}\right) \tag{11}$$

$\|\cdot\|_2^2$ 为$L2$-范数的平方。

Ltask-任务丢失:

特定任务的损失估计了训练过程中的预测质量。对于分类任务，我们使用标准的交叉熵损失，而对于回归任务，我们使用均方误差损失。对于一批中的Nb话语，这些被计算为:

$$\mathcal{L}_{\text{task}} = -\frac{1}{N_b}\sum_{i=0}^{N_b} \mathbf{y}_i \cdot \log \hat{\mathbf{y}}_i \qquad \text{for classification} \tag{12}$$

$$= \frac{1}{N_b}\sum_{i=0}^{N_b} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_2^2 \qquad \text{for regression} \tag{13}$$

实验

我们同时考虑了MSA（多模态情绪分析）和MHD（多模态幽默检测）任务的基准数据集。

数据集:

CMU-MOSI.CMU-MOSEI.是MSA任务常用数据集

ur_funny 是MHD任务常用数据集

评估指标:

在MOSI和MOSEI数据集中，情感强度预测是回归任务，使用平均绝对误差（MAE）和皮尔逊相关系数（Corr）作为评估指标。此外，基准测试还涉及分类评分，包括七类准确率（Acc-7，范围从-3到3）、二分类准确率（Acc-2）和F分数。
对于二分类准确率分数，过去考虑了两种不同的方法。第一种是负/非负分类，其中非负标签基于分数≥0 [61]。

在最近的研究中，二分类准确率是基于更准确的负/正类别划分计算的，其中负类别和正类别分别对应于<0和>0的情感分数 [52]。

我们报告了这些指标的结果，使用分段标记−/−，左侧得分用于负/非负分类，而右侧得分用于负/正分类。对于UR_FUNNY数据集，任务是一个标准的二分类，用二分类准确率（Acc-2）作为评估指标 [19]。

特征提取：

为了公平比较，我们使用了各个基准测试提供的标准的低级特征，这些特征也被最先进的方法所采用。

语言特征

传统上，语言模态特征是对话语中每个标记的全局[38]嵌入。然而，在最近的作品[7]之后，包括最先进的ICCN [50]，我们使用预先训练过的BERT [11]作为文本话语的特征提取器。使用BERT（Ul；$\theta bert$）替换等式中的$sLSTM$（Ul；$\theta llstm$）(1)。然而，对于UR_FUNNY来说，技术是基于GloVe特性。因此，为了进行公平的比较，我们提供了同时使用GloVe和BERT的结果。

虽然GloVe特征是300维的标记嵌入，但**对于BERT，我们使用了BERT-base-uncased预训练模型**。该模型由12个堆叠的Transformer层组成。与最近的工作 [1] 一致，我们选择话语向量ul为最终768维隐藏状态中标记的平均表示。

不幸的是，对于我们考虑的UR_FUNNY版本，原始转录本无法获得。相反，只提供了GloVe嵌入。为了检索原始文本，我们选择与GloVe词汇表中每个词嵌入余弦距离最小的标记。通过随机抽取100个话语进行手动检查，验证了这个流程获取可读原始转录本的质量。

视觉特征

MOSI和MOSEI都使用Facet 3来提取面部表情特征，其中包括基于面部动作编码系统（FACS）[14]的面部动作单元和面部姿态。这个过程对话语视频序列中的每个采样帧重复。对于UR_FUNNY，我们使用面部行为分析工具OpenFace [4]来提取与说话者的面部表情相关的特征。最终的视觉特征维度dv为MOSI为47，MOSEI为35，UR_FUNNY为75。

声学特征

声学特征包含了从covarep[10]-一个声学分析框架中提取的各种低级统计音频函数。其中包括12毫米频率中性系数、音高、语音/无声分割特征（VUV）[12]、声门源参数[13]，以及其他与情绪和音调相关的特征4。特征尺寸，da，MOSI/MOSEI为74，UR_FUNNY为81。

基线Baseline

以前的模型

许多关于多模态学习的方法已经被提出用于多模态学习，特别是在情绪分析和一般的人类语言任务中。如第2节所述，这些工作可以大致分为话语级和话语间上下文模型。

话语级基准包括：

执行时间建模和话语融合的网络： MFN [59]，MARN [61]，MV-LSTM [46]，RMFN [24]。

利用注意力和转换模块使用非语言信号改进标记表示的模型： RAVEN [56]，MulT [52]。

基于图的融合模型：图-mfn[60]。

话语向量融合方法，使用基于张量的融合和低秩变体：TFN [58]、LMF [26]、LMFN [31]、HFFN [29]。

使用循环平移（MCTN [39]）、对抗性自动编码器（ARGF [30]）和广义判别因子分解表示（MFM [51]）的通用子空间学习模型。

话语间上下文基准包括：

基于RNN的模型： BC-LSTM [44]，具有分层融合-CH-Fusion[32]。

话语间注意和多任务模式： CIA [6]，CIM-MTL [2]，DFF-ATMF [7]

最新进展

对于MSA任务，交互典型相关网络（ICCN）[50] 是MOSI和MOSEI上的最先进（SOTA）模型。ICCN首先从音频和视频模态中提取特征，然后与文本嵌入融合，得到两个外积，分别是文本-音频和文本-视频。接着，这些外积被输入到典型相关分析（CCA）网络，其输出用于预测。

对于MHD任务，最先进的方法是上下文记忆融合网络（CMFN）[19]，它通过提出单模态和多模态上下文网络扩展了MFN模型，考虑了之前的陈述，并使用MFN模型作为其骨干进行融合。最初，MFN [59] 是一个多视角门控记忆网络，在其记忆中存储了跨模态和模态内的陈述互动。

结果与分析

定量结果

多模态情绪分析

MSA的比较结果见表1（MOSI）和表2（MOSEI）。在这两个数据集中，MISA都取得了最好的性能，并在所有指标（回归和分类组合）上超过了基线——包括最先进的ICCN。在结果中，可以看出，我们的模型，这是一个话语级的模型，比上下文模型表现得更好。这是一个令人鼓舞的结果，因为即使使用较少的信息，我们也能表现得更好。我们的模型也超越了一些复杂的融合机制，如TFN和LFN，这证明了在融合阶段之前学习多模态表示的重要性。

Models	MOSI				
	MAE (↓)	Corr (↑)	Acc-2 (↑)	F-Score (↑)	Acc-7 (↑)
BC-LSTM	1.079	0.581	73.9 / -	73.9 / -	28.7
MV-LSTM	1.019	0.601	73.9 / -	74.0 / -	33.2
TFN	0.970	0.633	73.9 / -	73.4 / -	32.1
MARN	0.968	0.625	77.1 / -	77.0 / -	34.7
MFN	0.965	0.632	77.4 / -	77.3 / -	34.1
LMF	0.912	0.668	76.4 / -	75.7 / -	32.8
CH-Fusion	-	-	80.0 / -	-	-
MFM$^{\otimes}$	0.951	0.662	78.1 / -	78.1 / -	36.2
RAVEN$^{\otimes}$	0.915	0.691	78.0 / -	76.6 / -	33.2
RMFN$^{\otimes}$	0.922	0.681	78.4 / -	78.0 / -	38.3
MCTN$^{\otimes}$	0.909	0.676	79.3 / -	79.1 / -	35.6
CIA	0.914	0.689	79.8 / -	- / 79.5	38.9
HFFN$^{\oslash}$	-	-	- / 80.2	- / 80.3	-
LMFN$^{\oslash}$	-	-	- / 80.9	- / 80.9	-
DFF-ATMF (B)	-	-	- / 80.9	- / 81.2	-
ARGF	-	-	- / 81.4	- / 81.5	-
MulT	0.871	0.698	- / 83.0	- / 82.8	40.0
TFN (B)$^{\diamond}$	0.901	0.698	- / 80.8	- / 80.7	34.9
LMF (B)$^{\diamond}$	0.917	0.695	- / 82.5	- / 82.4	33.2
MFM (B)$^{\diamond}$	0.877	0.706	- / 81.7	- / 81.6	35.4
ICCN (B)	0.860	0.710	- / 83.0	- / 83.0	39.0
MISA (B)	**0.783**	**0.761**	**81.8† / 83.4†**	**81.7 / 83.6**	**42.3**
Δ_{SOTA}	↓ 0.077	↑ 0.051	↑ 2.0 / ↑ 0.4	↑ 2.6 / ↑ 0.6	↑ 3.3

Table 1: Performances of multimodal models in MOSI. NOTE: (B) means the language features are based on BERT; $^{\otimes}$ from [52]; $^{\oslash}$ from [30]; $^{\diamond}$ from [50]. Final row presents our best model per metric. $^{\dagger}p < 0.05$ under McNemar's Test for binary classification. Here, the statistical significance tests are compared with publicly available models of [26, 51, 58].

多模态幽默检测

在MHD也观察到类似的趋势（见表3），比上下文SOTA，C-MFN有非常显著的改善。即使在使用GloVe特性作为语言形态时，情况也是如此。事实上，我们的GloVe变体与基于bert的基线相当，比如TFN。这表明，对多模态表示的有效建模有很大的帮助。幽默检测对不同模式[19]的特殊特征高度敏感。这种依赖关系被我们的表示很好地建模，这反映在结果中。

Algorithms	context	target	UR_FUNNY Accuracy-2 (↑)
C-MFN	✓		58.45
C-MFN		✓	64.47
TFN		✓	64.71
LMF		✓	65.16
C-MFN	✓	✓	65.23
LMF (Bert)		✓	67.53
TFN (Bert)		✓	68.57
MISA (GloVe)		✓	68.60
MISA (Bert)		✓	**70.61**†
Δ_{SOTA}			↑ 2.07

Table 3: Performances of multimodal models in UR_FUNNY. $^{\dagger}p < 0.05$ **under McNemar's Test for binary classification when compared against [26, 58].** ***Context*****-based models use additional data that include the utterances preceding the target punchline.**

BERT vs. GloVe

在我们的实验中，我们观察到在使用BERT替代传统的基于GloVe的语言特征时，性能有所提升。这就引出了一个问题，即我们的性能改进是否仅仅是由于BERT特征所致。为了找到答案，我们研究了同样基于BERT的最先进方法ICCN。我们的模型在所有指标上均优于ICCN，通过这一点我们可以推断出，多模态建模的改进是一个关键因素。

消融研究Ablation Study

模态的作用

在表4（模型2、3、4）中，我们每次删除一个模态，以观察其对性能的影响。首先，我们看到一个多模态组合提供了最好的性能，这表明该模型可以学习互补的特征。如果没有这种情况，三模态组合将不会比语言视觉MISA更好。接下来，我们观察到，当语言模式被删除时，性能急剧下降。在去除其他两种模式时，没有观察到类似的下降，这表明文本模式在音频和视觉模式上有显著的优势。这可能有两个原因： 1)文本模态的数据质量在本质上可能会更好，因为它们是手动转录的。相比之下，音频和视觉信号都是未经过滤的原始信号。2) BERT是一种预先训练好的模型，比随机初始化的视听特征提取器具有更好的表达能力，提供了更好的话语级特征。然而，这些观察结果是特定于数据集的，不能推广到任何多模态场景。

	Model	MOSI		MOSEI		UR_FUNNY
		MAE (↓)	Corr (↑)	MAE (↓)	Corr (↑)	Acc-2 (↑)
1)	MISA	**0.783**	**0.761**	**0.555**	**0.756**	**70.6**
2)	(-) language l	1.450	0.041	0.801	0.090	55.5
3)	(-) visual v	0.798	0.756	0.558	0.753	69.7
4)	(-) audio a	0.849	0.732	0.562	0.753	70.2
5)	(-) $\mathcal{L}_{\text{sim}}$	0.807	0.740	0.566	0.751	69.3
6)	(-) $\mathcal{L}_{\text{diff}}$	0.824	0.749	0.565	0.742	69.3
7)	(-) $\mathcal{L}_{\text{recon}}$	0.794	0.757	0.559	0.754	69.7
8)	*base*	0.810	0.750	0.568	0.752	69.2
9)	*inv*	0.811	0.737	0.561	0.743	68.8
10)	*sFusion*	0.858	0.716	0.563	0.752	70.1
11)	*iFusion*	0.850	0.735	**0.555**	0.750	69.8

Table 4: Ablation Study. Here, (−) represents removal for the mentioned factors. Model 1 represents the best performing model in each dataset; Model 2,3,4 depicts the effect of individual modalities; Model 5,6,7 presents the effect of regularization; Model 8,9,10,11 presents the variants of MISA as defined in Section 5.2.3.

正则化的作用

正则化在实现第3.5节中讨论的期望表示中起着关键作用。在本节中，我们首先观察在训练时模型中损失的学习情况，以及验证集是否遵循类似的趋势。接下来，我们通过观察学习模型的特征分布来进行定性验证。最后，我们通过消融研究来观察每个损失的重要性。

正则化趋势：

损失{Lsim，Ldiff，Lrecon}作为量化模型学习模式不变和特定表示的措施。因此，我们在训练和验证集中的训练过程中跟踪损失。如图3所示，三种损失均随时代数量的增加呈下降趋势。这表明，该模型确实是在按照设计学习表示的。与训练集一样，验证集也表现出类似的行为。

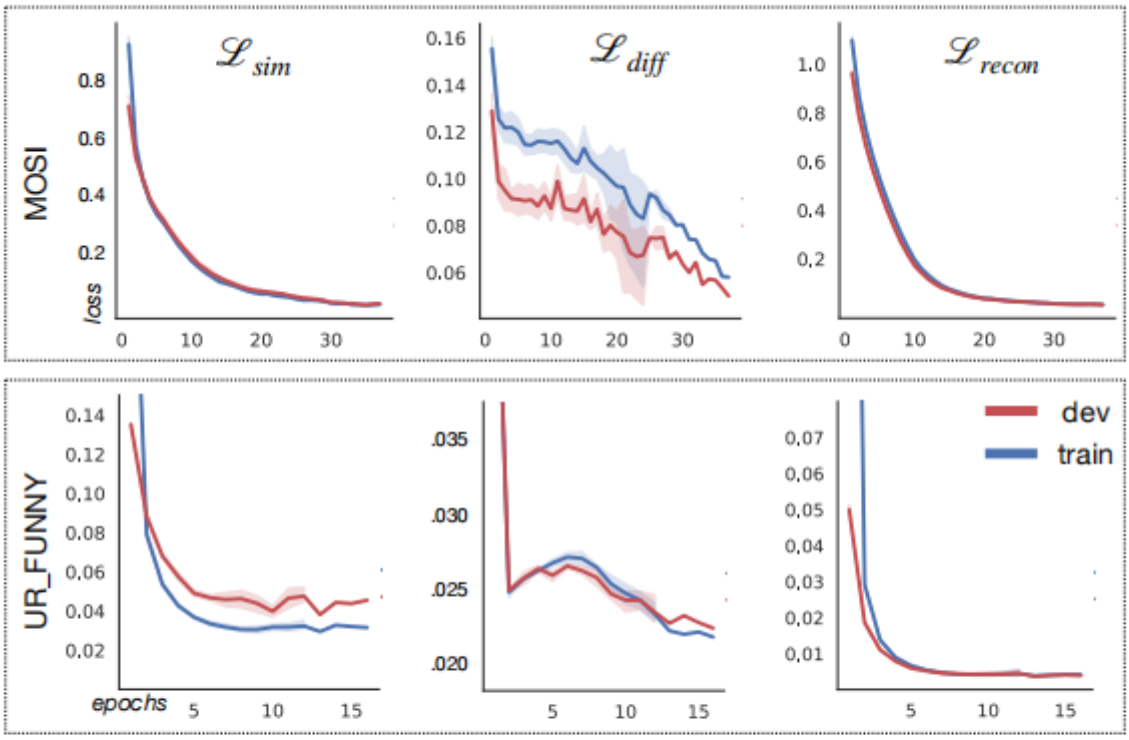


Figure 3: Trends in the regularization losses as training proceeds (values are for five runs across random seeds). Graphs depict losses in both training and validation sets for MOSI and UR_FUNNY. Similar trends are also observed in MOSEI.

可视化表示:

虽然图3显示了正则化损失在训练过程中的表现，但研究这些特征的广义性也很重要。因此，我们可视化了测试集中样本的隐藏表示。图4所示，可以清楚地看到，在没有正则化（$\alpha = 0$，$\beta = 0$）的情况下，不学习模态不变性。然而，当引入损失时，可以观察到模态不变表示之间的重叠。这表明MISA能够执行所需的子空间学习，即使是在广义的场景中，即在测试集中。我们将在第5.2.3节中进一步深入研究这些子空间的效用。

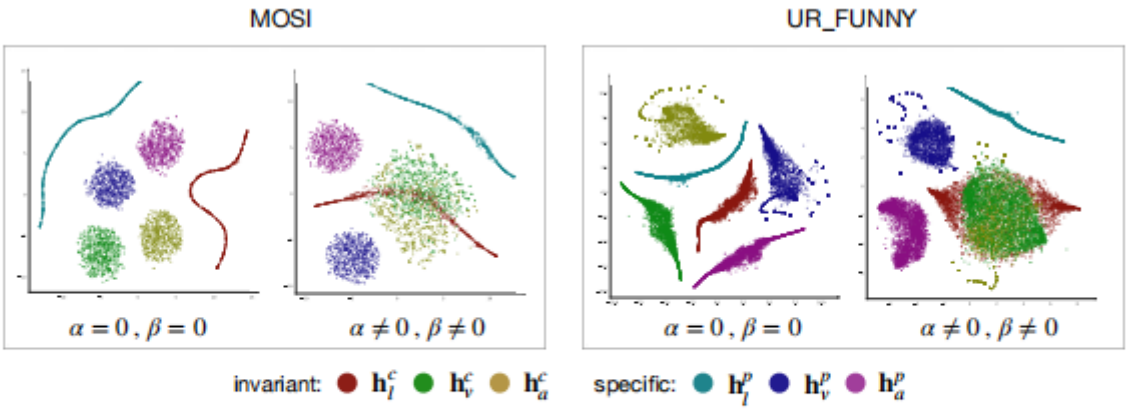


Figure 4: Visualization of the modality-invariant and -specific subspaces in the testing set of MOSI and UR_FUNNY datasets using t-SNE projections [28]. Observations on MOSEI are also similar.

正则化的重要性：

为了定量地验证这些损失的重要性，我们在每个数据集中采用最佳模型，并通过一次消融一个损失来重新训练它们。为了消除每个损失，我们将{α，β，γ}设置为0。结果见表4（模型5、6、7）。如上所示，当涉及到所有的损失时，就会达到最佳的性能。仔细观察，我们可以看到，模型对相似性和差异性损失特别敏感，这确保了模态不变性和特异性。这种依赖性表明，有单独的子空间确实是有用的。对于重建损失，我们看到模型对其依赖性较小。一种可能性是，尽管缺少重建损失，模态特定的编码器并没有采取简单的解决方案，而是利用任务损失学习到了信息丰富的表示。如果仅使用模态不变特征进行预测，情况就不会如此。

子空间的作用

在本节中，我们将研究我们提出的模型的几个变体，以研究其他假设：

1.MISA-base是一个基线版本，其中我们不学习不相交的子空间。相反，我们为每个模态使用三个独立的编码器——类似于以前的工作——并在它们上使用融合。

2.MISA-inv是一种没有模态特异性表示的变体。在这种情况下，只学习模态不变表示，并随后用于融合。

3.接下来的两个变体，MISA-sFusion和MISA-iFusion在表征学习阶段与MISA相同。在MISA-sFusion中，我们只使用模态特异性特征（hp{$l/v/a$}）来进行融合和预测。类似地，MISA-iFusion只使用模态不变特征（hc{$l/v/a$}）来进行融合。

我们总结了表4（模型8-11)中的结果。总的来说，我们发现我们的最终设计比变体更好。在这些变异中，我们观察到只学习一个不变的空间可能太严格，因为不是一个话语中的所有模式都共享相同的极性刺激。这反映在MISAinv并不比一般的MISA-base模型更好的结果中。MISA-sFusion和-iFusion都提高了性能，但最好的组合是当表示学习和融合同时利用模态子空间时，即所提出的模型MISA。

	Model	MOSI		MOSEI		UR_FUNNY
		MAE (↓)	Corr (↑)	MAE (↓)	Corr (↑)	Acc-2 (↑)
1)	MISA	**0.783**	**0.761**	**0.555**	**0.756**	**70.6**
2)	(-) language l	1.450	0.041	0.801	0.090	55.5
3)	(-) visual v	0.798	0.756	0.558	0.753	69.7
4)	(-) audio a	0.849	0.732	0.562	0.753	70.2
5)	(-) $\mathcal{L}_{\text{sim}}$	0.807	0.740	0.566	0.751	69.3
6)	(-) $\mathcal{L}_{\text{diff}}$	0.824	0.749	0.565	0.742	69.3
7)	(-) $\mathcal{L}_{\text{recon}}$	0.794	0.757	0.559	0.754	69.7
8)	*base*	0.810	0.750	0.568	0.752	69.2
9)	*inv*	0.811	0.737	0.561	0.743	68.8
10)	*sFusion*	0.858	0.716	0.563	0.752	70.1
11)	*iFusion*	0.850	0.735	**0.555**	0.750	69.8

Table 4: Ablation Study. Here, (−) represents removal for the mentioned factors. Model 1 represents the best performing model in each dataset; Model 2,3,4 depicts the effect of individual modalities; Model 5,6,7 presents the effect of regularization; Model 8,9,10,11 presents the variants of MISA as defined in Section 5.2.3.

可视化的注意力

为了分析学习到的表示的效用，我们来看看它们在融合步骤中的作用。如第3.4节所述，融合包括对模态表征的自我注意过程，该过程将h$c/p\ l/v/a$增强到h$c/p\ l/v/a$_hat，使用所有其他表征（包括其本身）的软注意组合。图5显示了测试集的平均注意力分布。图中的每一行都是各自表示的概率分布（在所有测试样本上的平均值）。查看这些列，每一列都可以看作是任何向量h∈{h$c/p\ l/v/a$_hat}对所有结果向量h$c/p\ l/v/a$_hat的增强表示的贡献。我们在图中观察到两个重要的模式。首先，我们注意到不变表示在所有三种模态中的影响是平等的。这在所有数据集中都是如此，并且是预期的，因为它们在共享空间中对齐。这也确立了不变特征之间的模态差距减少了。其次，我们注意到模态特定表示有显著的贡献。尽管每种模态的平均重要性取决于数据集，但语言如定量结果所示贡献最多，而声学和视觉模态提供不同程度的影响。然而，不变和特定的表示都在融合中提供了信息，正如在这些影响图中观察到的那样。

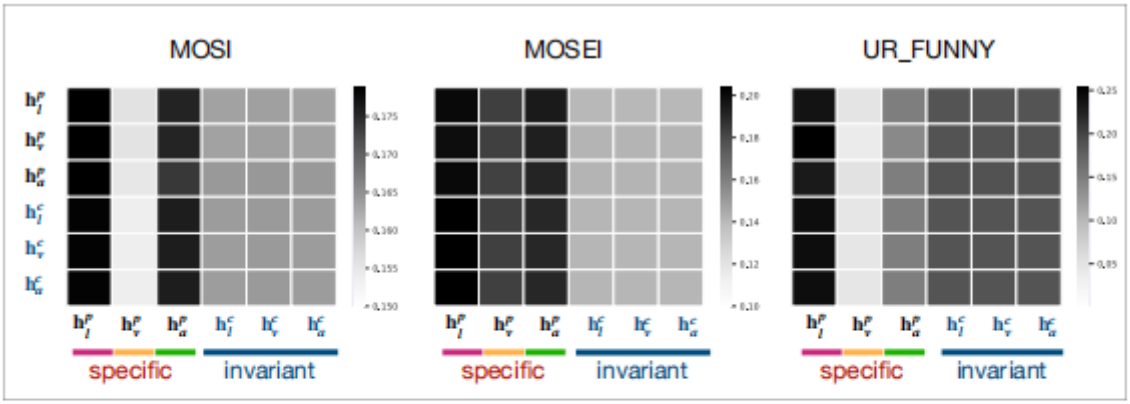


Figure 5: Average self-attention scores from the Transformer-based fusion module. The rows depict the *queries*, columns depict the *keys* (see Section 3.4). Essentially, each column represents the contribution of an input feature vector $\in \{h_l^c, h_v^c, h_a^c, h_l^p, h_v^p, h_a^p\}$ to generate the output feature vectors $[\bar{h}_l^c, \bar{h}_v^c, \bar{h}_a^c, \bar{h}_l^p, \bar{h}_v^p, \bar{h}_a^p]$.

结论

在本文中，我们提出了MISA-一个多模态情感框架，它将模态分解为模态不变特征和模态特异性特征，然后融合它们来预测情感状态。尽管包含简单的前馈层，我们发现MISA是非常有效的，并且在多模态情绪分析和幽默检测任务中比最先进的方法有显著的收益。探索性分析揭示了理想的特征，如减少模态差距，被表示学习功能学习，这避免了复杂的融合机制的需要。总的来说，我们强调了表征学习作为融合的一个预先步骤的重要性，并通过严格的实验证明了其有效性。我们的实验代码可在： https://github.com/declare-lab/MISA上找到。

在未来，我们计划在情感的其他维度上分析MISA，如情绪。此外，我们还打算将MISA框架与其他融合方案相结合，以尝试进一步的改进。最后，相似性和差异损失建模允许各种度量和正则化选择。因此，我们打算分析在这方面的其他选择。

复现实验

模块构建：

特征提取器：

```
rnn = nn.LSTM if self.config.rnncell == "lstm" else nn.GRU
# defining modules - two layer bidirectional LSTM with layer norm in between
```

```
if config.use_bert:
    # text subnets
    self.bertmodel = BertTextEncoder(language=config.language,
use_finetune=config.use_finetune)

self.vrnn1 = rnn(input_sizes[1], hidden_sizes[1], bidirectional=True)
self.vrnn2 = rnn(2*hidden_sizes[1], hidden_sizes[1], bidirectional=True)

self.arnn1 = rnn(input_sizes[2], hidden_sizes[2], bidirectional=True)
self.arnn2 = rnn(2*hidden_sizes[2], hidden_sizes[2], bidirectional=True)
```

将其话语序列Um∈RTm×dm映射到一个固定大小的向量um∈Rdh上：

```
# mapping modalities to same sized space
if self.config.use_bert:
    self.project_t = nn.Sequential()
    self.project_t.add_module('project_t', nn.Linear(in_features=768,
out_features=config.hidden_size))
    self.project_t.add_module('project_t_activation', self.activation)
    self.project_t.add_module('project_t_layer_norm', nn.LayerNorm(config.hidden_size))
else:
    self.project_t = nn.Sequential()
    self.project_t.add_module('project_t', nn.Linear(in_features=hidden_sizes[0]*4,
out_features=config.hidden_size))
    self.project_t.add_module('project_t_activation', self.activation)
    self.project_t.add_module('project_t_layer_norm', nn.LayerNorm(config.hidden_size))

self.project_v = nn.Sequential()
self.project_v.add_module('project_v', nn.Linear(in_features=hidden_sizes[1]*4,
out_features=config.hidden_size))
self.project_v.add_module('project_v_activation', self.activation)
self.project_v.add_module('project_v_layer_norm', nn.LayerNorm(config.hidden_size))

self.project_a = nn.Sequential()
self.project_a.add_module('project_a', nn.Linear(in_features=hidden_sizes[2]*4,
out_features=config.hidden_size))
self.project_a.add_module('project_a_activation', self.activation)
  self.project_a.add_module('project_a_layer_norm', nn.LayerNorm(config.hidden_size))
```

用编码器Ec和Ep学习模态的不变性表示hmc和特异性表示hmp：

```
# private encoders Ep 3个模态参数不共享
self.private_t = nn.Sequential()
self.private_t.add_module('private_t_1', nn.Linear(in_features=config.hidden_size,
out_features=config.hidden_size))
self.private_t.add_module('private_t_activation_1', nn.Sigmoid())

self.private_v = nn.Sequential()
```

```
self.private_v.add_module('private_v_1', nn.Linear(in_features=config.hidden_size,
out_features=config.hidden_size))
self.private_v.add_module('private_v_activation_1', nn.Sigmoid())

self.private_a = nn.Sequential()
self.private_a.add_module('private_a_3', nn.Linear(in_features=config.hidden_size,
out_features=config.hidden_size))
self.private_a.add_module('private_a_activation_3', nn.Sigmoid())


# shared encoder Ec 3个模态参数共享
self.shared = nn.Sequential()
self.shared.add_module('shared_1', nn.Linear(in_features=config.hidden_size,
out_features=config.hidden_size))
self.shared.add_module('shared_activation_1', nn.Sigmoid())
```

模态话语的重建模块D：

```
# reconstruct #3个模态参数不共享
self.recon_t = nn.Sequential()
self.recon_t.add_module('recon_t_1', nn.Linear(in_features=config.hidden_size,
out_features=config.hidden_size))
self.recon_v = nn.Sequential()
self.recon_v.add_module('recon_v_1', nn.Linear(in_features=config.hidden_size,
out_features=config.hidden_size))
self.recon_a = nn.Sequential()
self.recon_a.add_module('recon_a_1', nn.Linear(in_features=config.hidden_size,
out_features=config.hidden_size))
```

然后还有一个暂时没看懂的：

```
# shared space adversarial discriminator
if not self.config.use_cmd_sim:
  self.discriminator = nn.Sequential()
  self.discriminator.add_module('discriminator_layer_1',
nn.Linear(in_features=config.hidden_size, out_features=config.hidden_size))
  self.discriminator.add_module('discriminator_layer_1_activation', self.activation)
  self.discriminator.add_module('discriminator_layer_1_dropout',
nn.Dropout(dropout_rate))
  self.discriminator.add_module('discriminator_layer_2',
nn.Linear(in_features=config.hidden_size, out_features=len(hidden_sizes)))
```

```
# shared-private collaborative discriminator

self.sp_discriminator = nn.Sequential()
self.sp_discriminator.add_module('sp_discriminator_layer_1',
nn.Linear(in_features=config.hidden_size, out_features=4))
```

融合模块：

```
self.fusion = nn.Sequential()
self.fusion.add_module('fusion_layer_1',
nn.Linear(in_features=self.config.hidden_size*6,
out_features=self.config.hidden_size*3))
self.fusion.add_module('fusion_layer_1_dropout', nn.Dropout(dropout_rate))
self.fusion.add_module('fusion_layer_1_activation', self.activation)
#这个融合网络的最后输出，如果是回归问题会返回output_size=1
#如果是分类问题则返回output_size=num_classes
self.fusion.add_module('fusion_layer_3',
nn.Linear(in_features=self.config.hidden_size*3, out_features= output_size))

encoder_layer = nn.TransformerEncoderLayer(d_model=self.config.hidden_size, nhead=2)
self.transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=1)
```

前向传播：

获得每个模态的话语表示，并且映射到同一大小的向量um Rdh上，通过双向LSTM最后接一个全连接层实现：

$$\mathbf{u}_m = \text{sLSTM}\left(\mathbf{U}_m;\ \theta_m^{lstm}\right) \qquad (1)$$

```
# bert_sent_mask : consists of seq_len of 1, followed by padding of 0.
        bert_sent, bert_sent_mask, bert_sent_type = text[:,0,:], text[:,1,:],
text[:,2,:]

        batch_size = text.size(0)

        if self.config.use_bert:
            bert_output = self.bertmodel(text) # [batch_size, seq_len, 768]

            # Use the mean value of bert of the front real sentence length as the final
representation of text.
            masked_output = torch.mul(bert_sent_mask.unsqueeze(2), bert_output)
            mask_len = torch.sum(bert_sent_mask, dim=1, keepdim=True)
            bert_output = torch.sum(masked_output, dim=1, keepdim=False) / mask_len

            utterance_text = bert_output


        lengths = mask_len.squeeze().int().detach().cpu().view(-1)
        # extract features from visual modality
        final_h1v, final_h2v = self.extract_features(visual, lengths, self.vrnn1,
self.vrnn2, self.vlayer_norm)
        utterance_video = torch.cat((final_h1v, final_h2v), dim=2).permute(1, 0,
2).contiguous().view(batch_size, -1)
```

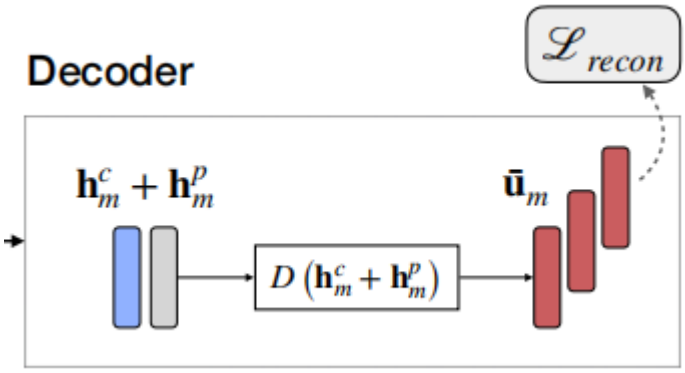
```
        # extract features from acoustic modality
        final_h1a, final_h2a = self.extract_features(acoustic, lengths, self.arnn1,
self.arnn2, self.alayer_norm)
        utterance_audio = torch.cat((final_h1a, final_h2a), dim=2).permute(1, 0,
2).contiguous().view(batch_size, -1)
```

获得模态不变性表示和模态特异性表示，通过编码器实现：

$$\mathbf{h}_m^c = E_c\left(\mathbf{u}_m; \theta^c\right), \quad \mathbf{h}_m^p = E_p\left(\mathbf{u}_m; \theta_m^p\right) \tag{2}$$

```
# Shared-private encoders
self.shared_private(utterance_text, utterance_video, utterance_audio)

def shared_private(self, utterance_t, utterance_v, utterance_a):

    # Projecting to same sized space
    #self.project_t:mapping modalities to same sized space
    self.utt_t_orig = utterance_t = self.project_t(utterance_t)
    self.utt_v_orig = utterance_v = self.project_v(utterance_v)
    self.utt_a_orig = utterance_a = self.project_a(utterance_a)


    # Private-shared components
    self.utt_private_t = self.private_t(utterance_t)
    self.utt_private_v = self.private_v(utterance_v)
    self.utt_private_a = self.private_a(utterance_a)

    self.utt_shared_t = self.shared(utterance_t)
    self.utt_shared_v = self.shared(utterance_v)
    self.utt_shared_a = self.shared(utterance_a)
```

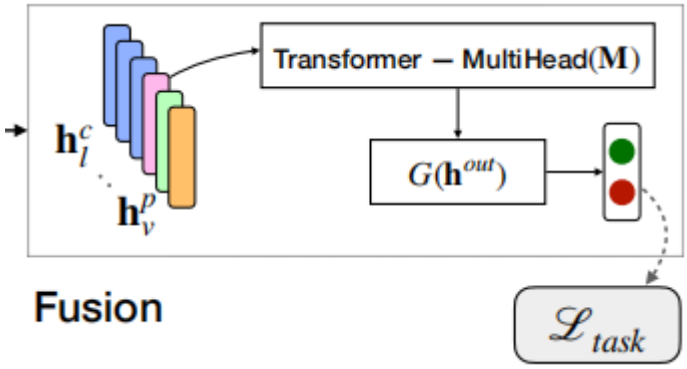
使用hmc和hmp重建模态话语表示um_hat：




```
# For reconstruction
    self.reconstruct()
```

```
def reconstruct(self,):

    self.utt_t = (self.utt_private_t + self.utt_shared_t)
    self.utt_v = (self.utt_private_v + self.utt_shared_v)
    self.utt_a = (self.utt_private_a + self.utt_shared_a)

    self.utt_t_recon = self.recon_t(self.utt_t)
    self.utt_v_recon = self.recon_v(self.utt_v)
    self.utt_a_recon = self.recon_a(self.utt_a)
    #这3个就是um_hat，重建的模态话语表示
```

使用三个模态的不变性表示和特异性表示融合：



$$\mathbf{M} = [\mathbf{h}_l^c, \mathbf{h}_v^c, \mathbf{h}_a^c, \mathbf{h}_l^p, \mathbf{h}_v^p, \mathbf{h}_a^p]$$

$$\bar{\mathbf{M}} = \text{MultiHead}(\mathbf{M}; \theta^{att}) = (\text{head}_1 \oplus \cdots \oplus \text{head}_n)W^o \quad (5)$$

$$\bar{\mathbf{M}} = [\bar{\mathbf{h}}_l^c, \bar{\mathbf{h}}_v^c, \bar{\mathbf{h}}_a^c, \bar{\mathbf{h}}_l^p, \bar{\mathbf{h}}_v^p, \bar{\mathbf{h}}_a^p]$$

$$\mathbf{h}^{out} = [\bar{\mathbf{h}}_l^c \oplus$$

$$\cdots \oplus \bar{\mathbf{h}}_a^p] \in \mathbb{R}^{6d_h}$$

$$\hat{\mathbf{y}} = G(\mathbf{h}^{out}; \theta^{out})$$

```
# 1-LAYER TRANSFORMER FUSION
h = torch.stack((self.utt_private_t, self.utt_private_v, self.utt_private_a,
self.utt_shared_t, self.utt_shared_v,  self.utt_shared_a), dim=0)
h = self.transformer_encoder(h)
h = torch.cat((h[0], h[1], h[2], h[3], h[4], h[5]), dim=1)
o = self.fusion(h)
return o
```

损失函数：

Lsim:

$$\mathcal{L}_{\text{sim}} = \frac{1}{3} \sum_{\substack{(m_1,m_2)\in \\ \{(l,a),(l,v),\\(a,v)\}}} CMD_K(\mathbf{h}^c_{m_1}, \mathbf{h}^c_{m_2}) \tag{8}$$

$$CMD_K(X,Y) = \frac{1}{|b-a|}\|\mathbf{E}(X) - \mathbf{E}(Y)\|_2 + \sum_{k=2}^{K} \frac{1}{|b-a|^k}\|C_k(X) - C_k(Y)\|_2 \tag{7}$$

```
self.loss_cmd = CMD()

def get_cmd_loss(self,):

    if not self.args.use_cmd_sim:
        return 0.0

    # losses between shared states
    loss = self.loss_cmd(self.model.Model.utt_shared_t, self.model.Model.utt_shared_v,
5)
    loss += self.loss_cmd(self.model.Model.utt_shared_t, self.model.Model.utt_shared_a,
5)
    loss += self.loss_cmd(self.model.Model.utt_shared_a, self.model.Model.utt_shared_v,
5)
    loss = loss/3.0

    return loss

class CMD(nn.Module):
    """
    Adapted from https://github.com/wzell/cmd/blob/master/models/domain_regularizer.py
    """

    def __init__(self):
        super(CMD, self).__init__()

    def forward(self, x1, x2, n_moments):
        mx1 = torch.mean(x1, 0)
        mx2 = torch.mean(x2, 0)
        sx1 = x1-mx1
        sx2 = x2-mx2
        dm = self.matchnorm(mx1, mx2)
        scms = dm
        for i in range(n_moments - 1):
            scms += self.scm(sx1, sx2, i + 2)
        return scms
```

```
    def matchnorm(self, x1, x2):
        power = torch.pow(x1-x2,2)
        summed = torch.sum(power)
        sqrt = summed**(0.5)
        return sqrt
        # return ((x1-x2)**2).sum().sqrt()

    def scm(self, sx1, sx2, k):
        ss1 = torch.mean(torch.pow(sx1, k), 0)
        ss2 = torch.mean(torch.pow(sx2, k), 0)
        return self.matchnorm(ss1, ss2)
```

Ldiff:

设Hmc和Hmp为矩阵（将矩阵变换为零均值和单位$l2$范数），其行表示每个话语的模态m的隐藏向量hmc和hmp。然后计算了该模态向量对的正交性约束为

$$\mathcal{L}_{\text{diff}} = \sum_{m \in \{l,v,a\}} \left\| \mathbf{H}_m^{c\top} \mathbf{H}_m^{p} \right\|_F^2 + \sum_{\substack{(m_1,m_2) \in \\ \{(l,a),(l,v),\\(a,v)\}}} \left\| \mathbf{H}_{m_1}^{p\top} \mathbf{H}_{m_2}^{p} \right\|_F^2 \qquad (10)$$

```
self.loss_diff = DiffLoss()

def get_diff_loss(self, ):

    shared_t = self.model.Model.utt_shared_t
    shared_v = self.model.Model.utt_shared_v
    shared_a = self.model.Model.utt_shared_a
    private_t = self.model.Model.utt_private_t
    private_v = self.model.Model.utt_private_v
    private_a = self.model.Model.utt_private_a

    # Between private and shared
    loss = self.loss_diff(private_t, shared_t)
    loss += self.loss_diff(private_v, shared_v)
    loss += self.loss_diff(private_a, shared_a)

    # Across privates
    loss += self.loss_diff(private_a, private_t)
    loss += self.loss_diff(private_a, private_v)
    loss += self.loss_diff(private_t, private_v)

    return loss

class DiffLoss(nn.Module):

    def __init__(self):
```

```
        super(DiffLoss, self).__init__()

    def forward(self, input1, input2):

        batch_size = input1.size(0)
        input1 = input1.view(batch_size, -1)
        input2 = input2.view(batch_size, -1)

        # Zero mean
        input1_mean = torch.mean(input1, dim=0, keepdims=True)
        input2_mean = torch.mean(input2, dim=0, keepdims=True)
        input1 = input1 - input1_mean
        input2 = input2 - input2_mean

        #L2 归一化
        input1_l2_norm = torch.norm(input1, p=2, dim=1, keepdim=True).detach()
        input1_l2 = input1.div(input1_l2_norm.expand_as(input1) + 1e-6)


        input2_l2_norm = torch.norm(input2, p=2, dim=1, keepdim=True).detach()
        input2_l2 = input2.div(input2_l2_norm.expand_as(input2) + 1e-6)

        diff_loss = torch.mean((input1_l2.t().mm(input2_l2)).pow(2))

        return diff_loss
```

Lrecon:

$$\mathcal{L}_{\text{recon}} = \frac{1}{3}\left(\sum_{m\in\{l,v,a\}} \frac{\|\mathbf{u}_m - \hat{\mathbf{u}}_m\|_2^2}{d_h}\right) \tag{11}$$

Where, $\|\cdot\|_2^2$ is the squared L_2-norm.

```
self.loss_recon = MSE()

def get_recon_loss(self, ):

    loss = self.loss_recon(self.model.Model.utt_t_recon, self.model.Model.utt_t_orig)
    loss += self.loss_recon(self.model.Model.utt_v_recon, self.model.Model.utt_v_orig)
    loss += self.loss_recon(self.model.Model.utt_a_recon, self.model.Model.utt_a_orig)
    loss = loss/3.0
    return loss

class MSE(nn.Module):
    def __init__(self):
        super(MSE, self).__init__()

    def forward(self, pred, real):
```

```
        diffs = torch.add(real, -pred)
        n = torch.numel(diffs.data)
        mse = torch.sum(diffs.pow(2)) / n

        return mse
```

Ltask：

$$\mathcal{L}_{\text{task}} = -\frac{1}{N_b}\sum_{i=0}^{N_b} \mathbf{y}_i \cdot \log \hat{\mathbf{y}}_i \qquad \text{for classification} \qquad (12)$$

$$= \frac{1}{N_b}\sum_{i=0}^{N_b} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_2^2 \qquad \text{for regression} \qquad (13)$$

```
self.criterion = nn.MSELoss() if args.train_mode == 'regression' else
nn.CrossEntropyLoss()
cls_loss = self.criterion(outputs, labels)
```

do_train 和do_valid

def do_train(self, model, dataloader, return_epoch_results=False):

```
#初始化模型参数和优化器
self.model = model
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),
lr=self.args.learning_rate)
# initilize results
epochs, best_epoch = 0, 0
if return_epoch_results:
    epoch_results = {
        'train': [],
        'valid': [],
        'test': []
    }
min_or_max = 'min' if self.args.KeyEval in ['Loss'] else 'max'
best_valid = 1e8 if min_or_max == 'min' else 0


while True:
    epochs += 1
    # train
    y_pred, y_true = [], []
    losses = []
    model.train()
    train_loss = 0.0
    left_epochs = self.args.update_epochs
    with tqdm(dataloader['train']) as td:
```

```
        for batch_data in td:
            # using accumulated gradients
            if left_epochs == self.args.update_epochs:
                optimizer.zero_grad()
            left_epochs -= 1
            text = batch_data['text'].to(self.args.device)
            audio = batch_data['audio'].to(self.args.device)
            vision = batch_data['vision'].to(self.args.device)
            labels = batch_data['labels']['M'].to(self.args.device)
            if self.args.train_mode == 'classification':
                labels = labels.view(-1).long()
            else:
                labels = labels.view(-1, 1)
            # forward
            outputs = model(text, audio, vision)['M']
            # compute loss

            cls_loss = self.criterion(outputs, labels)
            diff_loss = self.get_diff_loss()
            recon_loss = self.get_recon_loss()
            cmd_loss = self.get_cmd_loss()
            domain_loss = self.get_domain_loss()

            cmd_sim1=self.args.use_cmd_sim

            if self.args.use_cmd_sim:
                similarity_loss = cmd_loss
            else:
                similarity_loss = domain_loss

            loss = cls_loss + \
                   self.args.diff_weight * diff_loss + \
                   self.args.sim_weight * similarity_loss + \
                   self.args.recon_weight * recon_loss
            # backward
            loss.backward() #反向传播，计算梯度
            #如果设置了梯度裁剪值，则对梯度进行裁剪,以防止梯度爆炸
            #将每个梯度的值限制在 [-self.args.grad_clip, self.args.grad_clip] 范围内
            clip1=self.args.grad_clip
            if self.args.grad_clip != -1.0:
                torch.nn.utils.clip_grad_value_([param for param in model.parameters()
if param.requires_grad], self.args.grad_clip)
            # store results
            train_loss += loss.item()
            y_pred.append(outputs.cpu())
            y_true.append(labels.cpu())

            #如果剩余更新次数为0，则执行参数更新步骤，并重置剩余更新次数。
            if not left_epochs:
                optimizer.step()
                left_epochs = self.args.update_epochs
        if not left_epochs:
            # update
```

```
            optimizer.step()
        train_loss = train_loss / len(dataloader['train'])

        pred, true = torch.cat(y_pred), torch.cat(y_true)
        train_results = self.metrics(pred, true)
        logger.info(
            f"TRAIN-({self.args.model_name}) [{epochs -
best_epoch}/{epochs}/{self.args.cur_seed}] >> loss: {round(train_loss, 4)}
{dict_to_str(train_results)}"
        )


        # validation
        val_results = self.do_test(model, dataloader['valid'], mode="VAL")
        cur_valid = val_results[self.args.KeyEval]
        # save best model
        isBetter = cur_valid <= (best_valid - 1e-6) if min_or_max == 'min' else cur_valid
>= (best_valid + 1e-6)
        # save best model
        if isBetter:
            best_valid, best_epoch = cur_valid, epochs
            # save model
            torch.save(model.cpu().state_dict(), self.args.model_save_path)
            model.to(self.args.device)
        # epoch results
        if return_epoch_results:
            train_results["Loss"] = train_loss
            epoch_results['train'].append(train_results)
            epoch_results['valid'].append(val_results)
            test_results = self.do_test(model, dataloader['test'], mode="TEST")
            epoch_results['test'].append(test_results)
        # early stop
        if epochs - best_epoch >= self.args.early_stop:
            return epoch_results if return_epoch_results else None
```

do_valid/do_test

```
def do_test(self, model, dataloader, mode="VAL", return_sample_results=False):
    model.eval()
    y_pred, y_true = [], []
    eval_loss = 0.0
    if return_sample_results:
        ids, sample_results = [], []
        all_labels = []
        features = {
            "Feature_t": [],
            "Feature_a": [],
            "Feature_v": [],
            "Feature_f": [],
        }
    with torch.no_grad():
```

```
        with tqdm(dataloader) as td:
            for batch_data in td:
                vision = batch_data['vision'].to(self.args.device)
                audio = batch_data['audio'].to(self.args.device)
                text = batch_data['text'].to(self.args.device)
                labels = batch_data['labels']['M'].to(self.args.device)
                if self.args.train_mode == 'classification':
                    labels = labels.view(-1).long()
                else:
                    labels = labels.view(-1, 1)
                outputs = model(text, audio, vision)

                if return_sample_results:
                    ids.extend(batch_data['id'])
                    # TODO: add features
                    # for item in features.keys():
                    #     features[item].append(outputs[item].cpu().detach().numpy())
                    all_labels.extend(labels.cpu().detach().tolist())
                    preds = outputs["M"].cpu().detach().numpy()
                    # test_preds_i = np.argmax(preds, axis=1)
                    sample_results.extend(preds.squeeze())

                loss = self.criterion(outputs['M'], labels)
                eval_loss += loss.item()
                y_pred.append(outputs['M'].cpu())
                y_true.append(labels.cpu())
    eval_loss = eval_loss / len(dataloader)
    pred, true = torch.cat(y_pred), torch.cat(y_true)
    eval_results = self.metrics(pred, true)
    eval_results["Loss"] = round(eval_loss, 4)
    logger.info(f"{mode}-({self.args.model_name}) >> {dict_to_str(eval_results)}")

    if return_sample_results:
        eval_results["Ids"] = ids
        eval_results["SResults"] = sample_results
        # for k in features.keys():
        #     features[k] = np.concatenate(features[k], axis=0)
        eval_results['Features'] = features
        eval_results['Labels'] = all_labels

    return eval_results
```

代码复现结果为：

sims-regression:

Model	Mult_acc_2	Mult_acc_3	Mult_acc_5	F1_score	MAE	Corr	Loss
misa	(78.12, 0.0)	(63.02, 0.0)	(36.76, 0.0)	(78.12, 0.0)	(44.64, 0.0)	(56.78, 0.0)	(32.99, 0.0)

方差0是因为就只跑了一次seed=1111的

mosi-regression：

Model	Has0_acc_2	Has0_F1_score	Non0_acc_2	Non0_F1_score	Mult_acc_5	Mult_acc_7	MAE	Corr	Loss
misa	(80.76, 0.0)	(80.77, 0.0)	(82.32, 0.0)	(82.39, 0.0)	(48.1, 0.0)	(42.86, 0.0)	(74.76, 0.0)	(79.5, 0.0)	(100.88, 0.0)

解释一下：

Has0_acc_2与Non0_acc_2的区别在于，评估的二分类分别为：负/非负，负/正

Models	MOSI				
	MAE (↓)	Corr (↑)	Acc-2 (↑)	F-Score (↑)	Acc-7 (↑)
BC-LSTM	1.079	0.581	73.9 / -	73.9 / -	28.7
MV-LSTM	1.019	0.601	73.9 / -	74.0 / -	33.2
TFN	0.970	0.633	73.9 / -	73.4 / -	32.1
MARN	0.968	0.625	77.1 / -	77.0 / -	34.7
MFN	0.965	0.632	77.4 / -	77.3 / -	34.1
LMF	0.912	0.668	76.4 / -	75.7 / -	32.8
CH-Fusion	-	-	80.0 / -	-	-
MFM$^{\otimes}$	0.951	0.662	78.1 / -	78.1 / -	36.2
RAVEN$^{\otimes}$	0.915	0.691	78.0 / -	76.6 / -	33.2
RMFN$^{\otimes}$	0.922	0.681	78.4 / -	78.0 / -	38.3
MCTN$^{\otimes}$	0.909	0.676	79.3 / -	79.1 / -	35.6
CIA	0.914	0.689	79.8 / -	- / 79.5	38.9
HFFN$^{\oslash}$	-	-	- / 80.2	- / 80.3	-
LMFN$^{\oslash}$	-	-	- / 80.9	- / 80.9	-
DFF-ATMF (B)	-	-	- / 80.9	- / 81.2	-
ARGF	-	-	- / 81.4	- / 81.5	-
MulT	0.871	0.698	- / 83.0	- / 82.8	40.0
TFN (B)$^{\diamond}$	0.901	0.698	- / 80.8	- / 80.7	34.9
LMF (B)$^{\diamond}$	0.917	0.695	- / 82.5	- / 82.4	33.2
MFM (B)$^{\diamond}$	0.877	0.706	- / 81.7	- / 81.6	35.4
ICCN (B)	0.860	0.710	- / 83.0	- / 83.0	39.0
MISA (B)	**0.783**	**0.761**	**81.8**† **/ 83.4**†	**81.7 / 83.6**	**42.3**
Δ_{SOTA}	↓ 0.077	↑ 0.051	↑ 2.0 / ↑ 0.4	↑ 2.6 / ↑ 0.6	↑ 3.3

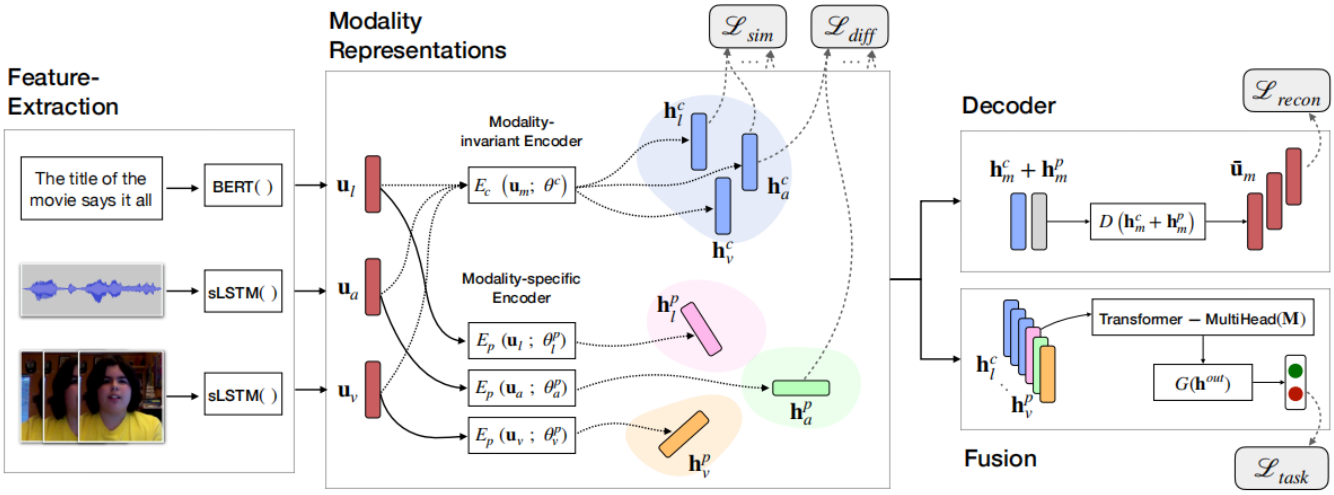
mosei-regression：

Model	Has0_acc_2	Has0_F1_score	Non0_acc_2	Non0_F1_score	Mult_acc_5	Mult_acc_7	MAE	Corr	Loss
misa	(79.61, 0.0)	(80.34, 0.0)	(84.4, 0.0)	(84.49, 0.0)	(53.83, 0.0)	(52.11, 0.0)	(54.89, 0.0)	(76.12, 0.0)	(52.82, 0.0)

Models	MOSEI				
	MAE (↓)	Corr (↑)	Acc-2 (↑)	F-Score (↑)	Acc-7 (↑)
MFN⊗	-	-	76.0 / -	76.0 / -	-
MV-LSTM⊗	-	-	76.4 / -	76.4 / -	-
Graph-MFN⊗	0.710	0.540	76.9 / -	77.0 / -	45.0
RAVEN	0.614	0.662	79.1 / -	79.5 / -	50.0
MCTN	0.609	0.670	79.8 / -	80.6 / -	49.6
CIA	0.680	0.590	80.4 / -	78.2 / -	50.1
CIM-MTL	-	-	80.5 / -	78.8 / -	-
DFF-ATMF (B)	-	-	- / 77.1	- / 78.3	-
MulT	0.580	0.703	- / 82.5	- / 82.3	51.8
TFN (B)◇	0.593	0.700	- / 82.5	- / 82.1	50.2
LMF (B)◇	0.623	0.677	- / 82.0	- / 82.1	48.0
MFM (B)◇	0.568	0.717	- / 84.4	- / 84.3	51.3
ICCN (B)	0.565	0.713	- / 84.2	- / 84.2	51.6
MISA (B)	**0.555**	**0.756**	**83.6**† **/ 85.5**†	**83.8 / 85.3**	**52.2**
Δ_{SOTA}	↓ 0.010	↑ 0.043	↑ 3.1 / ↑ 1.3	↑ 5.0 / ↑ 1.1	↑ 0.6

差不多吧，可能seed多了取平均是的，时间原因只跑了seed=1111的。

今天复现了MISA代码，明天试试在此基础上改成我们的方法试试。



- [x] ~~先把他的回归代码调成分类的，其实改个train_mode参数，再改几个地方就行了。~~
- [ ] 我基于他的ideal——把um使用编码器弄成 模态不变性hc 和 模态特异性hp，然后用这个去跑一下特征提取网络出e_c,e_p然后融合试试。
- [ ] 或者不用他的ideal，直接从um开始跑证据网络得e。
- [x] ~~试试他这个方法改进rcml里，然后跑pie那几个数据集~~

1.原始代码由regression模式改为classification模式：

train_mode=classification，然后相应的其他几个地方改了改

融合网络的最后一层output_size=1 改为 output_size=num_classes

损失函数Ltask从MSE换成交叉熵损失

sims运行结果：

Model	Has0_acc_2	Has0_F1_score	Non0_acc_2	Non0_F1_score	Acc_3	F1_score_3	Loss
misa	(78.12, 0.0)	(78.03, 0.0)	(56.44, 0.0)	(53.94, 0.0)	(67.18, 0.0)	(61.31, 0.0)	(84.86, 0.0)

mosi运行结果：

Model	Has0_acc_2	Has0_F1_score	Non0_acc_2	Non0_F1_score	Acc_3	F1_score_3	Loss
misa	(82.22, 0.0)	(82.11, 0.0)	(50.91, 0.0)	(49.29, 0.0)	(78.86, 0.0)	(76.98, 0.0)	(55.56, 0.0)

mosei运行结果（now seed is:1111 whole train_valid_test time: 1227.2670822143555 seconds）：

Model	Has0_acc_2	Has0_F1_score	Non0_acc_2	Non0_F1_score	Acc_3	F1_score_3	Loss
misa	(72.96, 0.0)	(72.42, 0.0)	(29.25, 0.0)	(29.33, 0.0)	(68.34, 0.0)	(65.75, 0.0)	(73.68, 0.0)

这么看我那个不是过拟合啊，你们都是这么垃啊。。。

今天小摆一天，周天先把self-mm那个源码跑一下train_mode=classification,看看是不是也这么垃。然后这个misa得到h后用我们的理论，在此基础上，我试试用本文的模态共性/特异性或者不用，损失函数相应的使用Lsim，Ldiff，Lrecon试试，然后Ltask在分类的时候把交叉熵换成我们那个edl_digamma_loss，edl_mse_loss试试。

2.我把我们RCML那个网络和损失函数都改了一下：

```
class RCML_misa(nn.Module):
    def __init__(self, num_views, dims, num_classes):
        super(RCML_misa, self).__init__()
        self.activation = nn.ReLU()
        self.dropout_rate  = 0.2
        self.hidden_size=128
        self.num_views = num_views
        self.num_classes = num_classes
        #模态私有表示的证据网络

self.EvidenceCollector_privates=nn.ModuleList([EvidenceCollector([self.hidden_size],
self.num_classes) for i in range(self.num_views)])
        #模态共享表示的证据网络

self.EvidenceCollector_shared=nn.ModuleList([EvidenceCollector([self.hidden_size],
self.num_classes) for i in range(self.num_views)])


#self.EvidenceCollector_shared=nn.ModuleList([EvidenceCollector([self.hidden_size],
self.num_classes)])
        #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



        #project 设置了num_views个
        self.project_layers = nn.ModuleList()
        for i in range(self.num_views):
            project_seq = nn.Sequential()
            project_seq.add_module(f'project_{i}', nn.Linear(in_features=dims[i][0],
out_features=self.hidden_size))
            project_seq.add_module(f'project_{i}_activation', self.activation)
```

```
            project_seq.add_module(f'project_{i}_layer_norm',
nn.LayerNorm(self.hidden_size))
            self.project_layers.append(project_seq)


        #shared encoder
        self.shared_encoder = nn.Sequential()
        self.shared_encoder.add_module('shared_1',
nn.Linear(in_features=self.hidden_size, out_features=self.hidden_size))
        self.shared_encoder.add_module('shared_activation_1', nn.Sigmoid())


        #private encoder 设置了self.num_views个
        self.private_encoders = nn.ModuleList()
        for i in range(self.num_views):
            private_seq = nn.Sequential()

private_seq.add_module(f'private_{i}_1',nn.Linear(in_features=self.hidden_size,
out_features=self.hidden_size))
            private_seq.add_module(f'private_{i}_activation_1', nn.Sigmoid())
            self.private_encoders.append(private_seq)


        #reconstruct模块,设置了num_views个
        self.reconstructs=nn.ModuleList()
        for i in range(self.num_views):
            reconstruct=nn.Sequential()

reconstruct.add_module(f'recon_{i}_1',nn.Linear(in_features=self.hidden_size,
out_features=self.hidden_size))
            self.reconstructs.append(reconstruct)


        encoder_layer=nn.TransformerEncoderLayer(d_model=self.hidden_size,nhead=2)
        self.transformer_encoder=nn.TransformerEncoder(encoder_layer,num_layers=1)


    def forward(self, X):
        #X是特征向量

        X_new=dict()
        X_private=dict()
        X_shared=dict()
        X_recon=dict()
        for v in range(self.num_views):
            # 1.先将X[0],X[1],,X[v]投影成相同维度的特征向量
            xv=X[v]
            X_new[v]=self.project_layers[v](X[v])
            #2.获得模态私有表示
            X_private[v]=self.private_encoders[v](X_new[v])
            #3.获得模态共享表示
            X_shared[v]=self.shared_encoder(X_new[v])
```

```
        #使用X_private[v]和X_shared[v]重建模态表示
        for v in range(self.num_views):
            X_recon[v]=(X_private[v]+X_shared[v])
            X_recon[v]=self.reconstructs[v](X_recon[v])


        #=torch.stack((X_private[0],,,X_private[v],X_shared[0],,X_shared[v]),dim=0)
        h = torch.stack([X_private[v] for v in range(self.num_views)] + [X_shared[v]
for v in range(self.num_views)],dim=0)
        h1=h
        h=self.transformer_encoder(h) #(2*num_views,batch_size,hidden_size=128)
        h2=h
        #h[0],h[1],h[2]
        #h[3],h[4],h[5]
        #上面时对应的模态私有，下面是模态共享


        #1.先提取证据向量
        evidences=dict()
        for v in range(self.num_views):
            id1=v #0
            id2=v+self.num_views #0+3=3
            hid1=h[id1]
            hid2=h[id2]
            evidences[id1]=self.EvidenceCollector_privates[v](h[id1])
            #evidences[id2]=self.EvidenceCollector_shared[0](h[id2])
            evidences[id2]=self.EvidenceCollector_shared[v](h[id2])

        #现在得到了2*num_views个证据向量
        #e[0],e[1],e[2]
        #e[3],e[4],e[5]
        # 上面时对应的模态私有，下面是模态共享

        #2.ABF融合
        evs=evidences
        evidence_private_a=evidences[0]
        evidence_shared_a=evidences[0+self.num_views]

        for v in range(1,self.num_views):
            id1=v
            id2=v+self.num_views
            evidence_private_a+=evidences[id1]
            evidence_shared_a+=evidences[id2]

        evidence_a=((evidence_private_a/self.num_views)+
(evidence_private_a/self.num_views))/2

        #要改损失函数了。。。
        return X_private,X_shared,X_recon,X_new,evidences,evidence_a

#司师兄的证据获取网络
class EvidenceCollector(nn.Module):
    def __init__(self, dims, num_classes):
```

```
        super(EvidenceCollector, self).__init__()
        #print('self.dims',dims) #dims=484
        self.num_layers = len(dims)
        #print('num_layers',self.num_layers) #num_layers=1
        self.net = nn.ModuleList()
        for i in range(self.num_layers - 1):
            #print('进入了')
            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
```

```
def get_loss1(num_views,X_private, X_shared,X_recon,X_new,evidences, evidence_a,
target, epoch_num, num_classes, annealing_step, gamma, device):

    evidences_p=dict()
    for v in range(num_views):
        evidences_p[v]=evidences[v]

    loss_sim=get_cmd_loss(num_views,X_shared)
    loss_diff=get_diff_loss(num_views,X_private,X_shared)
    loss_recon=get_recon_loss(num_views,X_recon,X_new)

    #求ltask
    target = F.one_hot(target, num_classes)
    alpha_a = evidence_a + 1
    loss_task=edl_digamma_loss(alpha_a,target, epoch_num, num_classes, annealing_step,
device)
    for v in range(num_views):
        alpha=evidences_p[v]+1
        loss_task+=edl_digamma_loss(alpha,target, epoch_num, num_classes,
annealing_step, device)
    loss_task=loss_task/(1+num_views)
    loss_task+=gamma*get_dc_loss(evidences_p, device)
    loss=loss_sim+loss_diff+loss_recon+loss_task

    return loss


#求loss_sim用的:
def matchnorm(x1, x2):
```

```
    power = torch.pow(x1-x2,2)
    summed = torch.sum(power)
    sqrt = summed**(0.5)
    return sqrt
    # return ((x1-x2)**2).sum().sqrt()
def scm( sx1, sx2, k):
    ss1 = torch.mean(torch.pow(sx1, k), 0)
    ss2 = torch.mean(torch.pow(sx2, k), 0)
    return matchnorm(ss1, ss2)
def CMD(X_shared1,X_shared2,n_moments):
    mx1 = torch.mean(X_shared1, 0)
    mx2 = torch.mean(X_shared2, 0)
    sx1 = X_shared1 - mx1
    sx2 = X_shared2 - mx2
    dm = matchnorm(mx1, mx2)
    scms = dm
    for i in range(n_moments - 1):
        scms += scm(sx1, sx2, i + 2)
    return scms

def get_cmd_loss(num_views,X_shared):
    loss=0
    for v1 in range(num_views):
        for v2 in range(num_views):
            if v1==v2:
                continue
            loss+=CMD(X_shared[v1],X_shared[v2],5)
    loss/=2
    loss/=num_views
    return loss


#求loss_diff用的
def get_diff_loss(num_views,X_private,X_shared):
    loss=0
    loss1=0
    loss2=0
    # Across privates
    for v1 in range(num_views):
        for v2 in range(num_views):
            if v1==v2:
                continue
            loss1+=loss_diff(X_private[v1],X_private[v2])
    loss1/=2

    for v in range(num_views):
        loss2+=loss_diff(X_private[v],X_shared[v])

    loss=loss1+loss2
    return loss

def loss_diff(input1, input2):
    batch_size = input1.size(0)
```

```
    input1 = input1.view(batch_size, -1)
    input2 = input2.view(batch_size, -1)
    # Zero mean
    input1_mean = torch.mean(input1, dim=0, keepdims=True)
    input2_mean = torch.mean(input2, dim=0, keepdims=True)
    input1 = input1 - input1_mean
    input2 = input2 - input2_mean

    # L2 归一化
    input1_l2_norm = torch.norm(input1, p=2, dim=1, keepdim=True).detach()
    input1_l2 = input1.div(input1_l2_norm.expand_as(input1) + 1e-6)

    input2_l2_norm = torch.norm(input2, p=2, dim=1, keepdim=True).detach()
    input2_l2 = input2.div(input2_l2_norm.expand_as(input2) + 1e-6)

    diff_loss = torch.mean((input1_l2.t().mm(input2_l2)).pow(2))

    return diff_loss

#求loss_recon用的
def get_recon_loss(num_views,X_recon,X_new):
    loss=0
    for v in range(num_views):
        loss+=loss_recon(num_views,X_recon[v],X_new[v])
    loss/=num_views
    return loss


def loss_recon(num_views,pred,real):
    diffs = torch.add(real, -pred)
    n = torch.numel(diffs.data)
    mse = torch.sum(diffs.pow(2)) / n
    return mse
```

PIE上运行结果：

十次acc，均值，方差，标准差

[[0.6764705882352942], [0.7647058823529411], [0.625], [0.75], [0.6764705882352942], [0.6544117647058824], [0.7941176470588235], [0.8088235294117647], [0.6911764705882353], [0.7352941176470589], [0.7176470588235294], [0.003440743944636676], [0.05865785492699742 6]]

然后改了个小地方：

```
#模态共享表示的证据网络
#self.EvidenceCollector_shared=nn.ModuleList([EvidenceCollector([self.hidden_size],
self.num_classes) for i in range(self.num_views)])

self.EvidenceCollector_shared=nn.ModuleList([EvidenceCollector([self.hidden_size],
self.num_classes)])
```

PIE上运行结果：

[[0.7132352941176471], [0.7794117647058824], [0.6691176470588235], [0.7279411764705882], [0.7720588235294118], [0.7426470588235294], [0.6691176470588235], [0.8014705882352942], [0.8014705882352942], [0.7647058823529411], [0.7441176470588236], [0.00214965397923875 66], [0.046364361089513105]]

这样设置比那个性能好点

突然发现三个正则化损失忘记加系数了，umo了。。。

```
gamma = 1
sim_weight=0.8
diff_weight=0.3
recon_weight=0.8

def get_loss1(num_views,X_private, X_shared,X_recon,X_new,evidences, evidence_a,
target, epoch_num, num_classes, annealing_step, gamma,
sim_weight,diff_weight,recon_weight,device):

    evidences_p=dict()
    for v in range(num_views):
        evidences_p[v]=evidences[v]

    loss_sim=get_cmd_loss(num_views,X_shared)
    loss_diff=get_diff_loss(num_views,X_private,X_shared)
    loss_recon=get_recon_loss(num_views,X_recon,X_new)

    #求ltask
    target = F.one_hot(target, num_classes)
    alpha_a = evidence_a + 1
    loss_task=edl_digamma_loss(alpha_a,target, epoch_num, num_classes, annealing_step,
device)
    for v in range(num_views):
        alpha=evidences_p[v]+1
        loss_task+=edl_digamma_loss(alpha,target, epoch_num, num_classes,
annealing_step, device)
    loss_task=loss_task/(1+num_views)
    loss_task+=gamma*get_dc_loss(evidences_p, device)
    loss=sim_weight*loss_sim+diff_weight*loss_diff+recon_weight*loss_recon+loss_task

    return loss
```

PIE上运行结果：

[[0.7573529411764706], [0.7647058823529411], [0.7058823529411765], [0.7279411764705882], [0.7352941176470589], [0.7720588235294118],

[0.7426470588235294], [0.7941176470588235], [0.8014705882352942], [0.8088235294117647], [0.7610294117647058], [0.0010299524221453282], [0.032092871827639986]]

改个超参数还能提！！！

```
gamma = 1
sim_weight=1
diff_weight=0.5
recon_weight=1
```

[[0.7132352941176471], [0.7205882352941176], [0.75], [0.7647058823529411], [0.7867647058823529], [0.8235294117647058], [0.7867647058823529], [0.6911764705882353], [0.7352941176470589], [0.75], [0.7522058823529412], [0.0014170631487889261], [0.03764389922403]]