DialogueGCN: A Graph Convolution Neural Network for Emotional Recognition in Conversation

Deepanway Ghosal, Navonil Majumder, Soujanya Poria, Niyati Chhaya, and Alexander Gelbukh.

Yeon-goon Kim

Seoul National University

SNU CML
Communication and Machine Learning

Contents

- Problem Definition
- Base Model: DialogueRNN (AAAI 2019)
- DialogueGCN (EMNLP 2019)

Figures

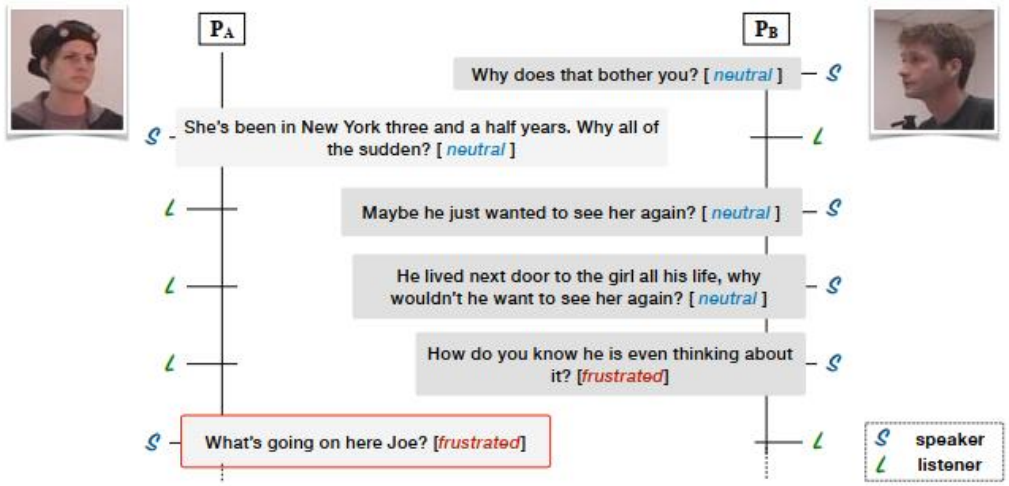
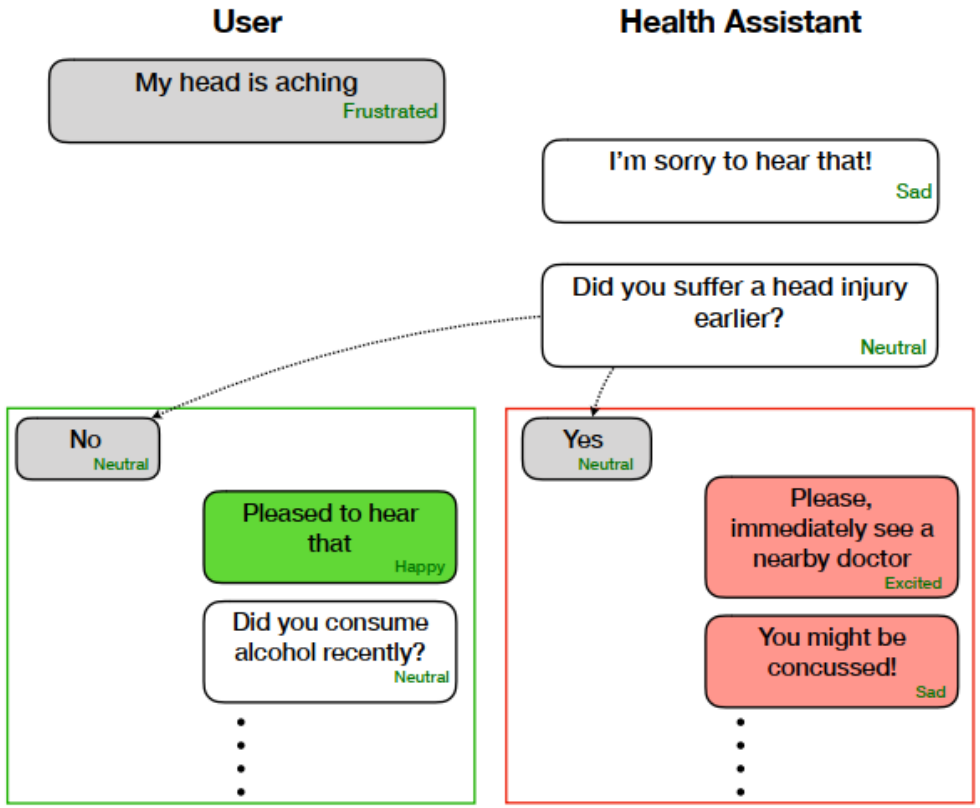


Figure 1: In this dialogue, P_A's emotion changes are influenced by the behavior of P_B.



Problem Definition

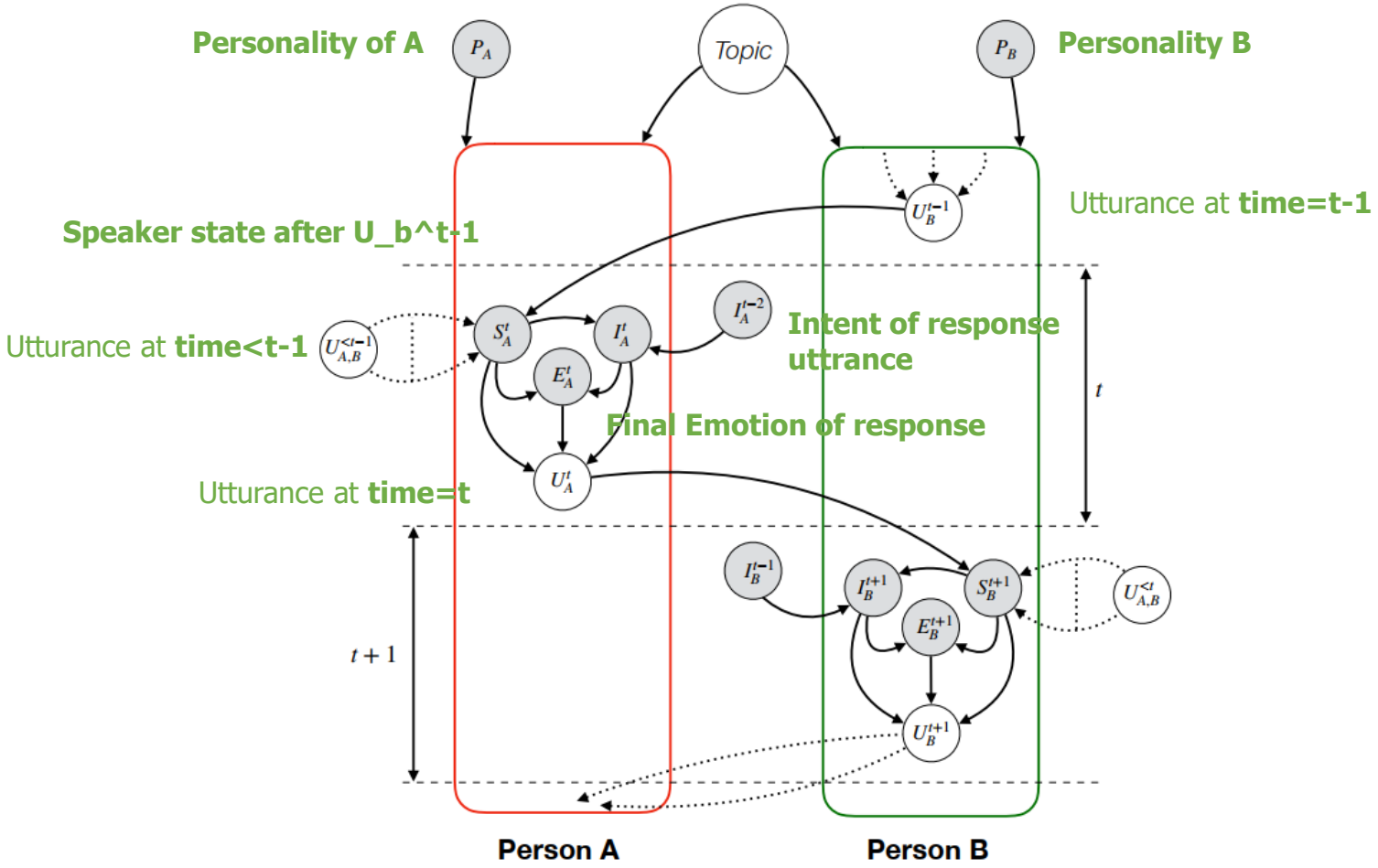
D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.

3.1 Problem Definition

Let there be M parties/participants $p_1, p_2, \ldots, p_M$ ($M = 2$ for the datasets we used) in a conversation. The task is to predict the emotion labels (*happy*, *sad*, *neutral*, *angry*, *excited*, and *frustrated*) of the constituent utterances $u_1, u_2, \ldots, u_N$, where utterance u_t is uttered by party $p_{s(u_t)}$, while s being the mapping between utterance and index of its corresponding party. Also, $u_t \in \mathbb{R}^{D_m}$ is the utterance representation, obtained using feature extractors described below.

ERC Description with Graph

S. Poria *et al.* Emotion Recognition in Conversation: Research Challenges, Datasets, and Recent Advances. IEEE Access, 2019.



Base Model: DialogueRNN Architecture

N. Majumder and S. Poria *et al.* DialogueRNN: An Attentive RNN for Emotion Detection in Conversations. In AAAI, 2019.

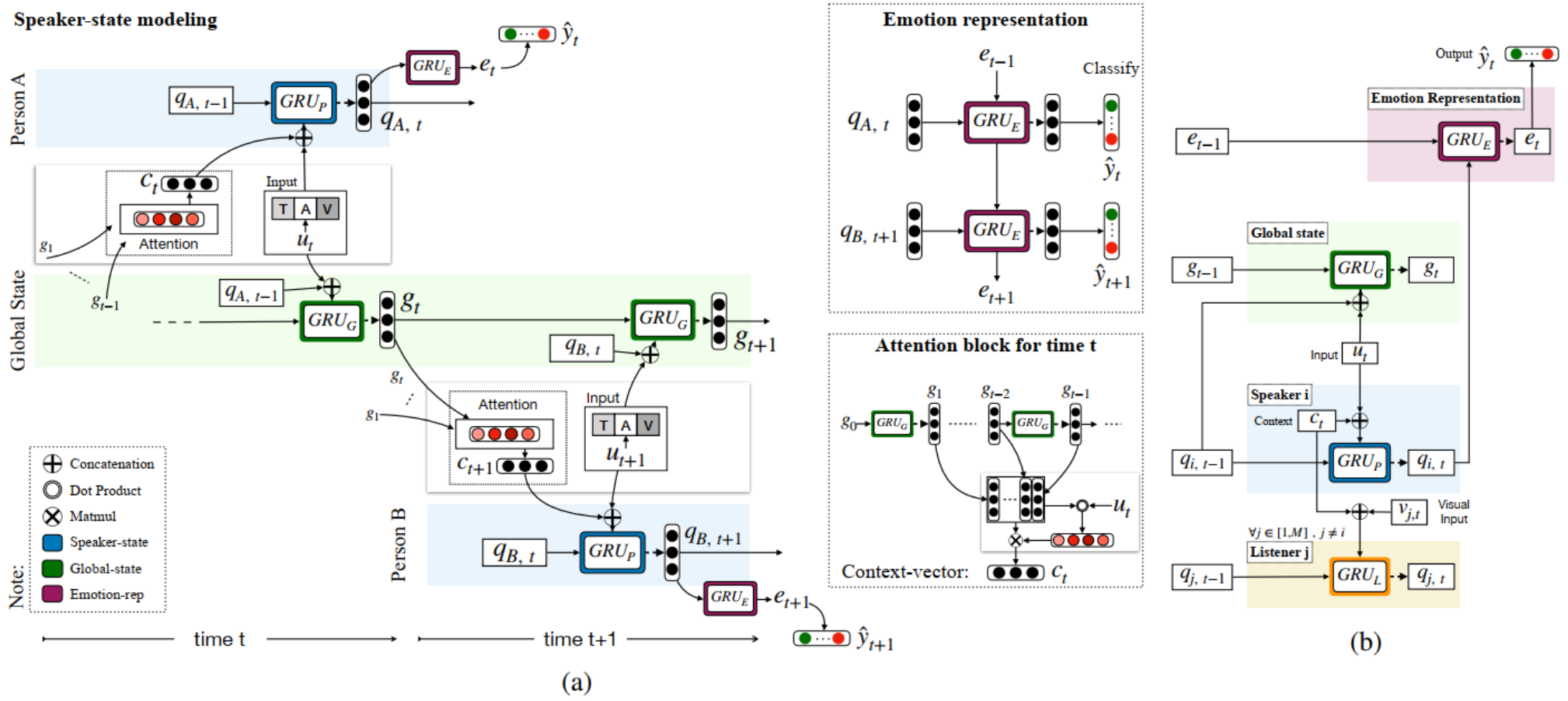


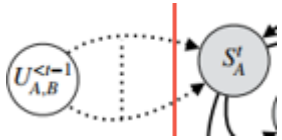
Figure 2: (a) DialogueRNN architecture. (b) Update schemes for global, speaker, listener, and emotion states for t^{th} utterance in a dialogue. Here, Person i is the speaker and Persons $j \in [1, M]$ and $j \neq i$ are the listeners.

3 Updates of DialogueRNN

N. Majumder and S. Poria *et al.* DialogueRNN: An Attentive RNN for Emotion Detection in Conversations. In AAAI, 2019.

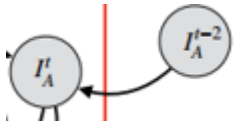
1) Global State: **Overall Emotional flows of conversation.**

g_t



2) Party State: **Emotion behind utterance** that each party have. Emotional changes among conversation flows.

$q_{s(u_t),t}$

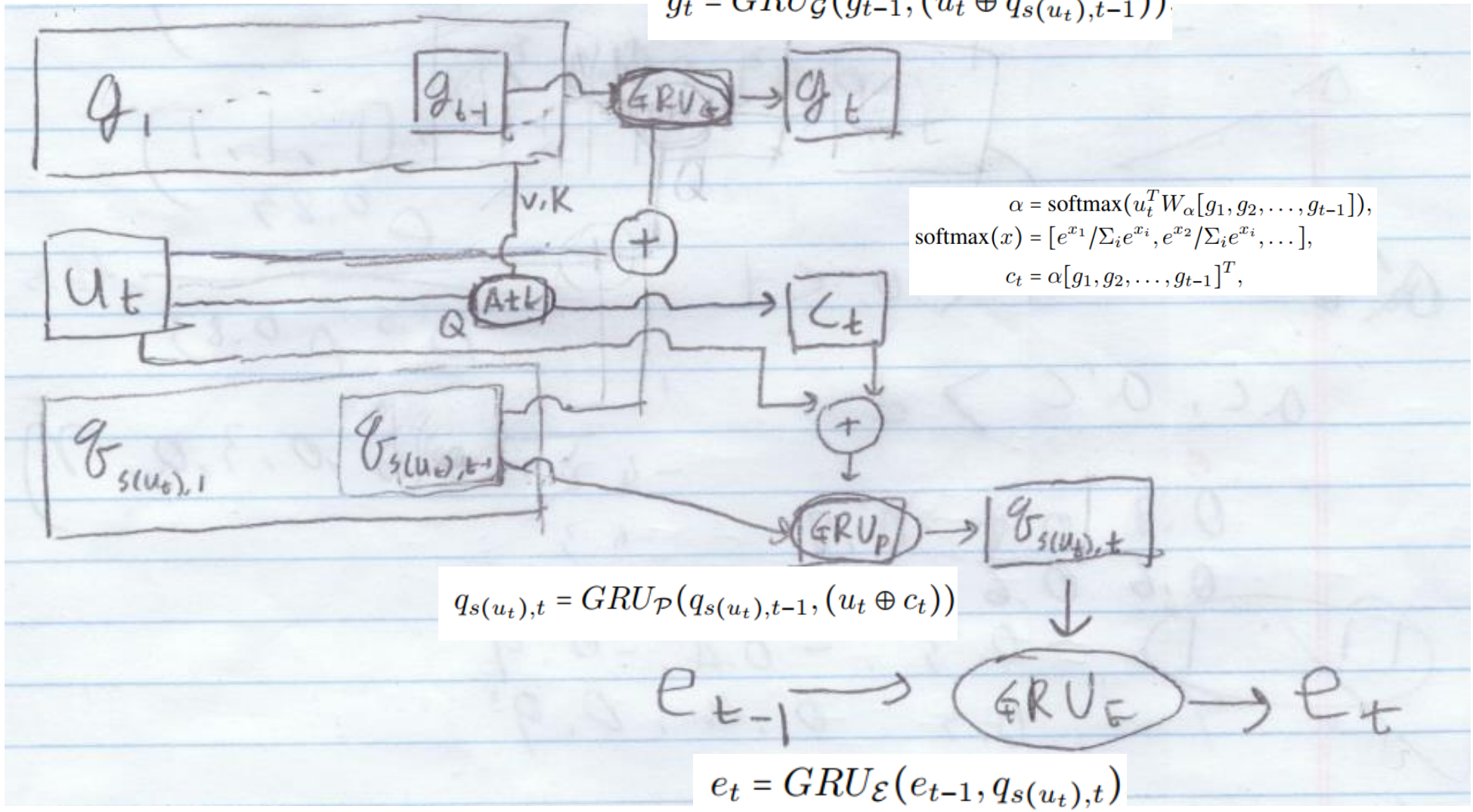


3) Emotion Recognition: Based on **global state history**, **current party sate**, and **just before recognition result**.

e_t

Simplified Architecture Figure

N. Majumder and S. Poria et al. DialogueRNN: An Attentive RNN for Emotion Detection in Conversations. In AAAI, 2019.



DialogueGCN Architecture

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.

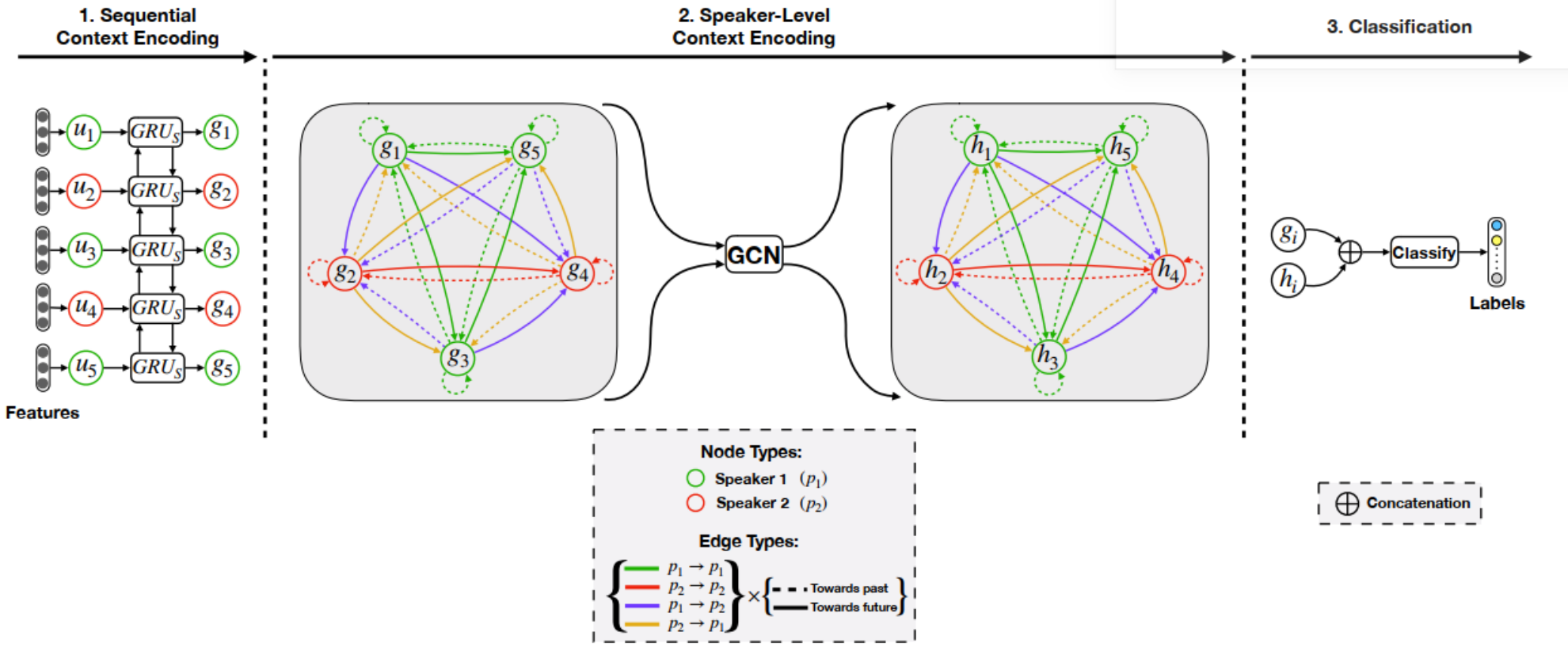
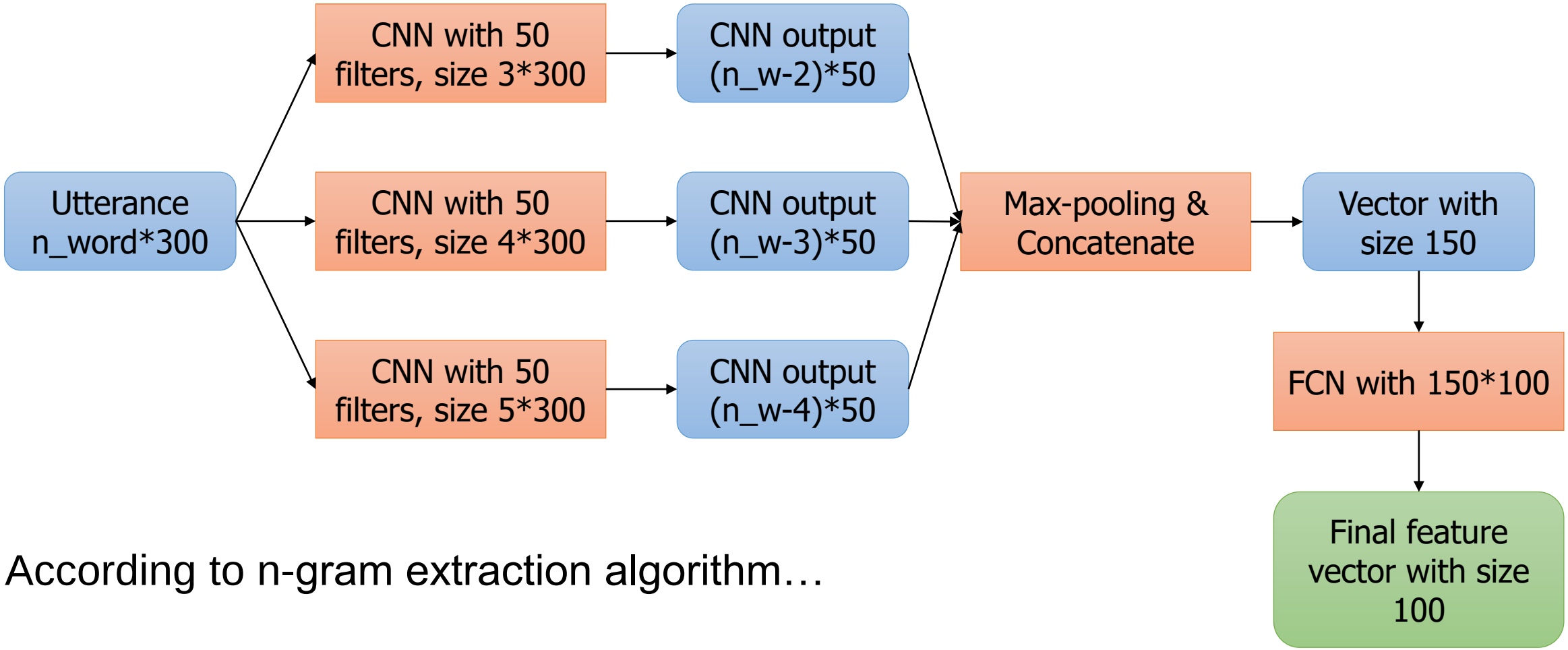


Figure 3: Overview of DialogueGCN, congruent to the illustration in Table 1.

Feature Extraction with CNN

Yoon Kim, Convolutional Neural Networks for Sentence Classification. In EMNLP, 2014



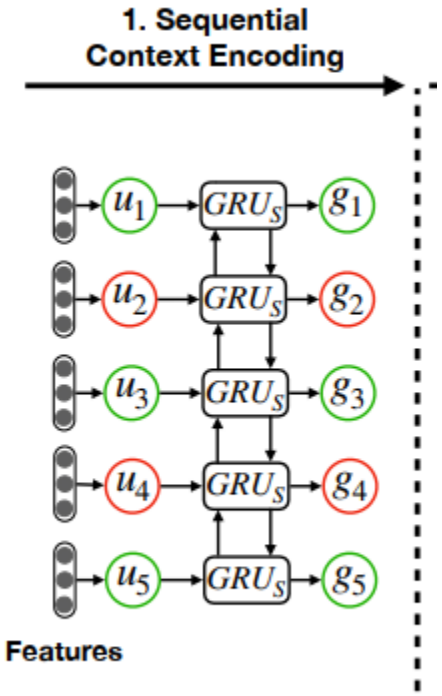
According to n-gram extraction algorithm…

Part ①. Sequential Context Encoding

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.

$$g_i = \overleftrightarrow{GRU_S}(g_{i(+,-)1}, u_i)$$

So, **DO NOT consider speaker information** at global state calculation. We'll consider it at next step.



Graph Initialization

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{W})$$

Vertices: Each utterance in the conversation is represented as a vertex $v_i \in \mathcal{V}$ in $\mathcal{G}$. Each vertex v_i is initialized with the corresponding sequentially encoded feature vector g_i, for all $i \in [1, 2, ..., N]$.

Graph Initialization

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.

2) Edge : 기본적으로는 fully connected graph => $O(N^2)$이라 계산량 너무 많으므로 앞뒤 window 설정 (± 10)
각 edge에 a) Relation b) weight 할당되어 있음

a) Relation (fixed)
두 Nodes의 speaker와 전후 관계 고려 (U_i가 U_j보다 시간적으로 먼저인가 나중인가에 따라 $2N^2$개의 relation 할당.

b) Weight → 일종의 self-Attention?

$\alpha_{ij} = \text{softmax}(g_i^T W_e [g_{i-10}, \cdots, g_{i+10}])$ for $j = i-10 \sim +10$

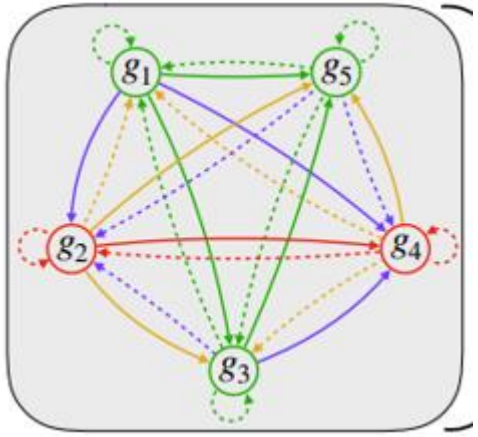
$$\alpha_{ij} = \text{softmax}(g_i^T W_e [g_{i-p}, \ldots, g_{i+f}]), \quad \text{for } j = i-p, \ldots, i+f.$$

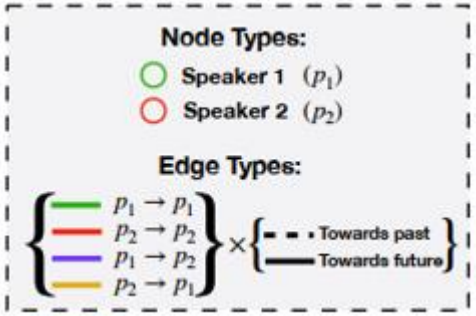
=> Attention module that $Q = g_i$;
$V, K = g_{i-10} \sim g_{i+10}$

=> 각 vertex V_i에 대하여, incoming edge의 종합은 DRNN의 Party State와 비슷 (Schlichtkrull 2018)

Graph Initialization (Relations)

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.





Relation	$p_s(u_i), p_s(u_j)$	$i < j$	(i, j)
1	p_1, p_1	Yes	(1,3), (1,5), (3,5)
2	p_1, p_1	No	(1,1), (3,1), (3,3) (5,1), (5,3), (5,5)
3	p_2, p_2	Yes	(2,4)
4	p_2, p_2	No	(2,2), (4,2), (4, 4)
5	p_1, p_2	Yes	(1,2), (1,4), (3,4)
6	p_1, p_2	No	(3,2), (5,2), (5,4)
7	p_2, p_1	Yes	(2,3), (2,5), (4,5)
8	p_2, p_1	No	(2,1), (4,1), (4,3)

Relational Graph Convolutional Network

M. Schlichtkrul and T. Kipf *et al*. Modeling Relational Data with Graph Convolutional Networks. In ESWC, 2018.

In class… (Spatial View of Shebnet)

Hence,

$$\boldsymbol{F}_O[i,:] = \sum_{v_j \in N(v_i) \cup \{v_i\}} \boldsymbol{C}[i,j] \boldsymbol{F}_I[j,:] \boldsymbol{\Theta}$$

With different notation…

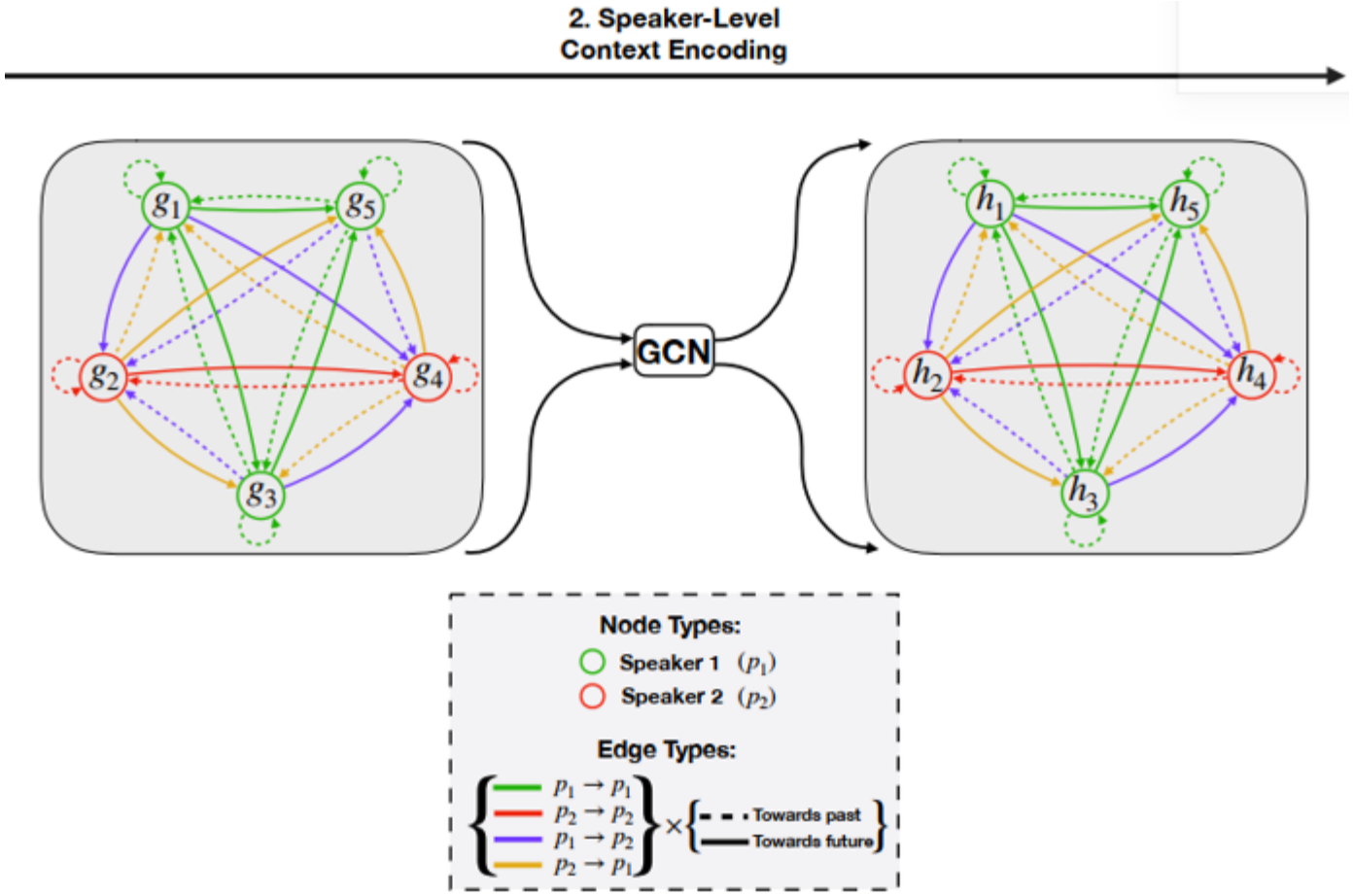
$$h_{v_i}^{(l+1)} = \sigma\left(\sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)}\right)$$

To apply on relational graph…

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right)$$

Part ②. Speaker-Level Context Encoding

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.



R-GCN

$$h_i^{(1)} = \sigma\Big(\sum_{r\in\mathcal{R}}\sum_{j\in N_i^r}\frac{\alpha_{ij}}{c_{i,r}}W_r^{(1)}g_j + \alpha_{ii}W_0^{(1)}g_i\Big),$$

$$\text{for } i = 1, 2, \ldots, N,$$

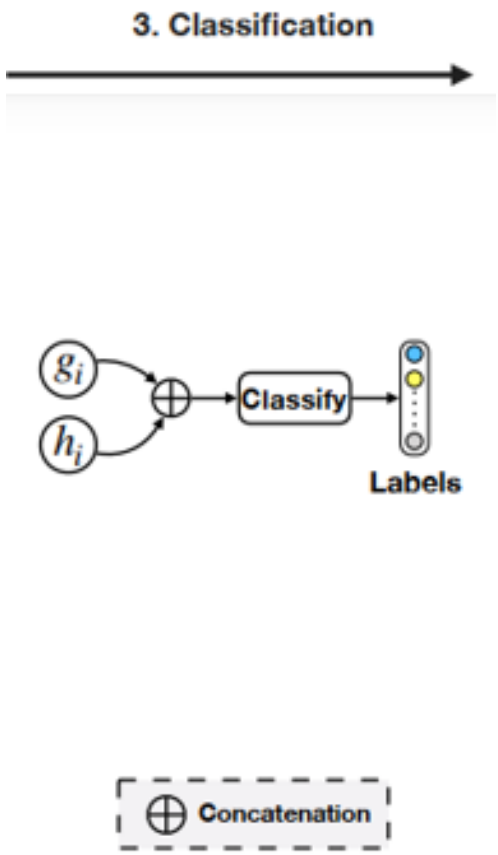
Simple GNN

$$h_i^{(2)} = \sigma\Big(\sum_{j\in N_i^r} W^{(2)}h_j^{(1)} + W_0^{(2)}h_i^{(1)}\Big),$$

$$\text{for } i = 1, 2, \ldots, N,$$

Part ③. Emotional Recognition

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.



$$h_i = [g_i, h_i^{(2)}],$$
$$\beta_i = \text{softmax}(h_i^T W_\beta [h_1, h_2 \ldots, h_N]),$$
$$\tilde{h}_i = \beta_i [h_1, h_2, \ldots, h_N]^T.$$

Attention Score

$$l_i = \text{ReLU}(W_l \tilde{h}_i + b_l),$$
$$\mathcal{P}_i = \text{softmax}(W_{smax} l_i + b_{smax}),$$
$$\hat{y}_i = \underset{k}{\text{argmax}}(\mathcal{P}_i[k]).$$

FCN

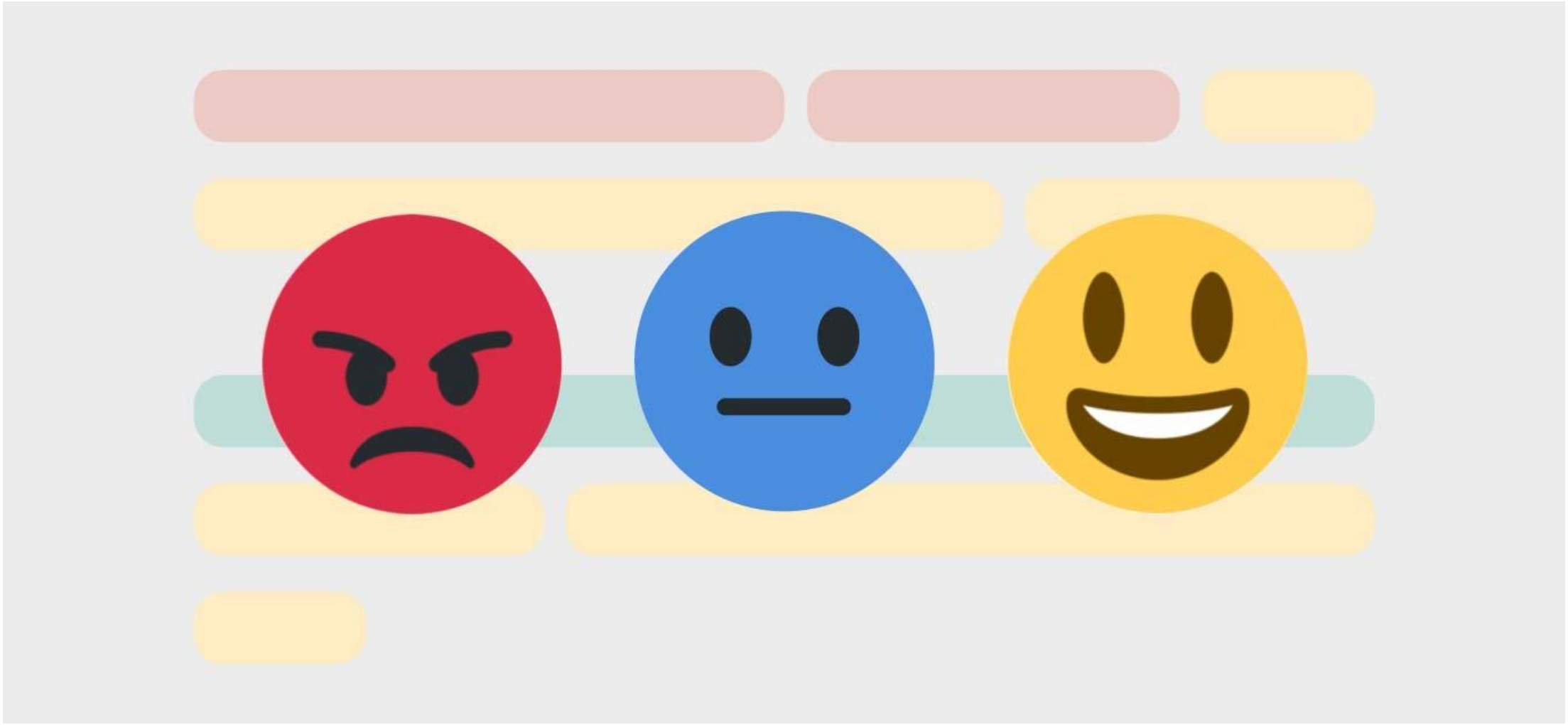
Results

D. Ghosal *et al.* DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation. In EMNLP, 2019.

Methods	IEMOCAP													
	Happy		Sad		Neutral		Angry		Excited		Frustrated		**Average(w)**	
	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
CNN	27.77	29.86	57.14	53.83	34.33	40.14	61.17	52.44	46.15	50.09	62.99	55.75	48.92	48.18
Memnet	25.72	33.53	55.53	61.77	58.12	52.84	59.32	55.39	51.50	58.30	67.20	59.00	55.72	55.10
bc-LSTM	29.17	34.43	57.14	60.87	54.17	51.81	57.06	56.73	51.17	57.95	67.19	58.92	55.21	54.95
bc-LSTM+Att	30.56	35.63	56.73	62.90	57.55	53.00	59.41	59.24	52.84	58.85	65.88	59.41	56.32	56.19
CMN	25.00	30.38	55.92	62.41	52.86	52.39	61.76	59.83	55.52	60.25	71.13	60.69	56.56	56.13
ICON	22.22	29.91	58.78	64.57	62.76	57.38	64.71	63.04	58.86	63.42	67.19	60.81	59.09	58.54
DialogueRNN	25.69	33.18	75.10	78.80	58.59	59.21	64.71	**65.28**	80.27	**71.86**	61.15	58.91	63.40	62.75
DialogueGCN	40.62	**42.75**	89.14	**84.54**	61.92	**63.54**	67.53	64.19	65.46	63.08	64.18	**66.99**	65.25	**64.18**

Comments

- Graph may contain much more fruitful model.
- Overfitting when code reproducing.



Thank You!

END OF PRESENTATION

DialogueRNN Code Result

```
lakeking@8e39d57de40c:/conv-emotion/DialogueRNN$ python train_IEMOCAP.py
Namespace(active_listener=False, attention='general', batch_size=30, class_weight=True, dropout=0.1, epochs=60, l2=1e-05, lr=0.0001, no_cuda=False, rec_dropout=0.1, tensorboard=False)
Running on GPU
epoch 1 train_loss 1.7815 train_acc 18.81 train_fscore15.37 valid_loss nan valid_acc nan val_fscorenan test_loss 1.7774 test_acc 22.43 test_fscore 16.26 time 6.18
epoch 2 train_loss 1.7423 train_acc 33.17 train_fscore26.44 valid_loss nan valid_acc nan val_fscorenan test_loss 1.7558 test_acc 30.62 test_fscore 22.06 time 5.87
epoch 3 train_loss 1.7065 train_acc 44.2 train_fscore40.03 valid_loss nan valid_acc nan val_fscorenan test_loss 1.7336 test_acc 38.63 test_fscore 31.19 time 5.91
epoch 4 train_loss 1.6708 train_acc 49.93 train_fscore46.97 valid_loss nan valid_acc nan val_fscorenan test_loss 1.709 test_acc 42.82 test_fscore 37.86 time 5.89
epoch 5 train_loss 1.6281 train_acc 54.53 train_fscore51.94 valid_loss nan valid_acc nan val_fscorenan test_loss 1.6816 test_acc 48.92 test_fscore 46.15 time 5.64
epoch 6 train_loss 1.5833 train_acc 57.69 train_fscore55.25 valid_loss nan valid_acc nan val_fscorenan test_loss 1.6497 test_acc 51.94 test_fscore 49.26 time 5.92
epoch 7 train_loss 1.5298 train_acc 58.98 train_fscore56.0 valid_loss nan valid_acc nan val_fscorenan test_loss 1.6135 test_acc 52.87 test_fscore 50.5 time 6.23
epoch 8 train_loss 1.4744 train_acc 59.76 train_fscore56.58 valid_loss nan valid_acc nan val_fscorenan test_loss 1.5724 test_acc 55.33 test_fscore 53.43 time 5.91
epoch 9 train_loss 1.4178 train_acc 61.14 train_fscore58.18 valid_loss nan valid_acc nan val_fscorenan test_loss 1.5271 test_acc 57.86 test_fscore 55.68 time 5.69
epoch 10 train_loss 1.3598 train_acc 62.01 train_fscore58.58 valid_loss nan valid_acc nan val_fscorenan test_loss 1.4791 test_acc 58.47 test_fscore 56.07 time 5.93
epoch 11 train_loss 1.3004 train_acc 62.82 train_fscore59.81 valid_loss nan valid_acc nan val_fscorenan test_loss 1.4294 test_acc 60.2 test_fscore 57.41 time 5.8
epoch 12 train_loss 1.2361 train_acc 63.22 train_fscore59.95 valid_loss nan valid_acc nan val_fscorenan test_loss 1.3801 test_acc 60.94 test_fscore 58.15 time 5.98
epoch 13 train_loss 1.1745 train_acc 64.22 train_fscore61.11 valid_loss nan valid_acc nan val_fscorenan test_loss 1.3322 test_acc 60.87 test_fscore 58.05 time 6.04
epoch 14 train_loss 1.121 train_acc 65.96 train_fscore63.27 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2874 test_acc 60.81 test_fscore 58.2 time 6.04
epoch 15 train_loss 1.0741 train_acc 65.96 train_fscore63.53 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2466 test_acc 61.31 test_fscore 59.0 time 6.05
epoch 16 train_loss 1.0219 train_acc 66.63 train_fscore64.7 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2085 test_acc 61.06 test_fscore 58.94 time 6.01
epoch 17 train_loss 0.9763 train_acc 68.9 train_fscore67.34 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1763 test_acc 61.12 test_fscore 59.22 time 5.88
epoch 18 train_loss 0.9296 train_acc 70.36 train_fscore69.43 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1492 test_acc 61.12 test_fscore 59.41 time 5.9
epoch 19 train_loss 0.8856 train_acc 71.33 train_fscore70.43 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1271 test_acc 61.68 test_fscore 60.21 time 5.79
epoch 20 train_loss 0.8475 train_acc 72.05 train_fscore71.36 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1091 test_acc 61.8 test_fscore 60.54 time 6.07
epoch 21 train_loss 0.8038 train_acc 73.43 train_fscore72.92 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0945 test_acc 62.29 test_fscore 61.19 time 6.11
epoch 22 train_loss 0.7645 train_acc 75.23 train_fscore74.79 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0826 test_acc 62.17 test_fscore 61.29 time 5.74
epoch 23 train_loss 0.7343 train_acc 76.97 train_fscore76.68 valid_loss nan valid_acc nan val_fscorenan test_loss 1.072 test_acc 62.05 test_fscore 61.26 time 5.91
epoch 24 train_loss 0.7014 train_acc 77.4 train_fscore77.16 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0617 test_acc 61.8 test_fscore 61.14 time 5.92
epoch 25 train_loss 0.6609 train_acc 79.17 train_fscore79.0 valid_loss nan valid_acc nan val_fscorenan test_loss 1.056 test_acc 61.92 test_fscore 61.32 time 5.87
epoch 26 train_loss 0.6319 train_acc 80.84 train_fscore80.75 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0487 test_acc 61.61 test_fscore 61.13 time 5.69
epoch 27 train_loss 0.5969 train_acc 82.15 train_fscore82.06 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0428 test_acc 61.49 test_fscore 61.05 time 6.05
epoch 28 train_loss 0.5766 train_acc 82.77 train_fscore82.71 valid_loss nan valid_acc nan val_fscorenan test_loss 1.036 test_acc 61.55 test_fscore 61.13 time 5.91
epoch 29 train_loss 0.5457 train_acc 84.32 train_fscore84.24 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0336 test_acc 61.55 test_fscore 61.22 time 6.14
epoch 30 train_loss 0.5151 train_acc 84.97 train_fscore84.9 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0333 test_acc 61.68 test_fscore 61.39 time 5.95
epoch 31 train_loss 0.4913 train_acc 85.71 train_fscore85.65 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0367 test_acc 61.55 test_fscore 61.33 time 5.99
epoch 32 train_loss 0.4673 train_acc 86.64 train_fscore86.61 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0382 test_acc 61.18 test_fscore 60.99 time 5.99
epoch 33 train_loss 0.447 train_acc 87.9 train_fscore87.86 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0401 test_acc 61.12 test_fscore 60.99 time 6.33
epoch 34 train_loss 0.425 train_acc 88.36 train_fscore88.34 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0415 test_acc 61.18 test_fscore 61.12 time 6.03
epoch 35 train_loss 0.4126 train_acc 88.5 train_fscore88.46 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0432 test_acc 61.31 test_fscore 61.28 time 6.19
epoch 36 train_loss 0.3841 train_acc 89.69 train_fscore89.67 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0514 test_acc 61.18 test_fscore 61.23 time 5.76
epoch 37 train_loss 0.3777 train_acc 89.69 train_fscore89.68 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0631 test_acc 60.81 test_fscore 60.94 time 6.03
epoch 38 train_loss 0.3644 train_acc 89.79 train_fscore89.77 valid_loss nan valid_acc nan val_fscorenan test_loss 1.0744 test_acc 60.94 test_fscore 61.07 time 5.84
epoch 39 train_loss 0.3475 train_acc 90.52 train_fscore90.5 valid_loss nan valid_acc nan val_fscorenan test_loss 1.088 test_acc 60.69 test_fscore 60.83 time 5.89
epoch 40 train_loss 0.337 train_acc 90.86 train_fscore90.85 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1022 test_acc 60.57 test_fscore 60.76 time 5.69
epoch 41 train_loss 0.3244 train_acc 91.33 train_fscore91.31 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1111 test_acc 60.75 test_fscore 60.94 time 6.2
epoch 42 train_loss 0.3198 train_acc 91.29 train_fscore91.28 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1176 test_acc 61.0 test_fscore 61.16 time 6.19
epoch 43 train_loss 0.3086 train_acc 91.51 train_fscore91.5 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1259 test_acc 61.06 test_fscore 61.25 time 6.05
epoch 44 train_loss 0.3039 train_acc 91.58 train_fscore91.57 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1302 test_acc 61.06 test_fscore 61.26 time 6.1
epoch 45 train_loss 0.2957 train_acc 92.08 train_fscore92.08 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1324 test_acc 60.75 test_fscore 60.93 time 5.89
epoch 46 train_loss 0.2867 train_acc 92.27 train_fscore92.27 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1412 test_acc 60.87 test_fscore 61.05 time 6.06
epoch 47 train_loss 0.2702 train_acc 92.82 train_fscore92.81 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1515 test_acc 61.0 test_fscore 61.17 time 6.38
epoch 48 train_loss 0.2683 train_acc 92.7 train_fscore92.69 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1614 test_acc 61.06 test_fscore 61.19 time 5.55
epoch 49 train_loss 0.2737 train_acc 92.69 train_fscore92.67 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1698 test_acc 60.81 test_fscore 60.98 time 6.07
epoch 50 train_loss 0.2591 train_acc 93.03 train_fscore93.02 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1738 test_acc 61.24 test_fscore 61.36 time 6.36
```

DialogueRNN Code Result

```
epoch 51 train_loss 0.2614 train_acc 92.79 train_fscore92.78 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1773 test_acc 60.87 test_fscore 60.95 time 6.04
epoch 52 train_loss 0.2557 train_acc 92.93 train_fscore92.92 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1788 test_acc 61.06 test_fscore 61.14 time 6.28
epoch 53 train_loss 0.2531 train_acc 93.17 train_fscore93.16 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1833 test_acc 60.94 test_fscore 60.98 time 6.0
epoch 54 train_loss 0.2516 train_acc 93.2 train_fscore93.19 valid_loss nan valid_acc nan val_fscorenan test_loss 1.1909 test_acc 61.06 test_fscore 61.11 time 6.32
epoch 55 train_loss 0.2444 train_acc 93.15 train_fscore93.14 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2025 test_acc 60.75 test_fscore 60.8 time 6.25
epoch 56 train_loss 0.2415 train_acc 93.32 train_fscore93.31 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2179 test_acc 60.81 test_fscore 60.86 time 6.25
epoch 57 train_loss 0.2295 train_acc 93.73 train_fscore93.72 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2335 test_acc 60.44 test_fscore 60.54 time 6.01
epoch 58 train_loss 0.2369 train_acc 93.49 train_fscore93.49 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2445 test_acc 60.57 test_fscore 60.66 time 6.11
epoch 59 train_loss 0.232 train_acc 93.86 train_fscore93.85 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2509 test_acc 60.38 test_fscore 60.48 time 6.15
epoch 60 train_loss 0.2317 train_acc 93.84 train_fscore93.84 valid_loss nan valid_acc nan val_fscorenan test_loss 1.2472 test_acc 60.94 test_fscore 60.96 time 6.09
Test performance..
Loss 1.0333 accuracy 61.68
```

DialogueGCN Code Result

```
lakeking@8e39d57de40c:/conv-emotion/DialogueGCN$ python train_IEMOCAP.py --base-model 'LSTM' --graph-model --nodal-attention --dropout 0.4 --lr 0.0003 --batch-size 32 --class-weight --l2 0.0
Namespace(active_listener=False, attention='general', base_model='LSTM', batch_size=32, class_weight=True, dropout=0.4, epochs=60, graph_model=True, l2=0.0, lr=0.0003, no_cuda=False, nodal_attention=True, rec_dropout=0.1, tensorboard=False,
windowf=10, windowp=10)
Running on GPU
Graph NN with LSTM as base model.
epoch: 1, train_loss: 1.773, train_acc: 24.96, train_fscore: 13.84, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.7592, test_acc: 20.39, test_fscore: 8.59, time: 34.02 sec
epoch: 2, train_loss: 1.6881, train_acc: 38.61, train_fscore: 29.49, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.6944, test_acc: 33.83, test_fscore: 24.21, time: 33.72 sec
epoch: 3, train_loss: 1.5605, train_acc: 42.41, train_fscore: 34.48, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.5806, test_acc: 36.35, test_fscore: 26.1, time: 34.07 sec
epoch: 4, train_loss: 1.3841, train_acc: 43.84, train_fscore: 36.8, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.4159, test_acc: 37.58, test_fscore: 29.44, time: 34.36 sec
epoch: 5, train_loss: 1.2274, train_acc: 47.95, train_fscore: 42.44, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.243, test_acc: 53.42, test_fscore: 45.84, time: 34.19 sec
epoch: 6, train_loss: 1.1441, train_acc: 53.01, train_fscore: 48.71, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1419, test_acc: 57.05, test_fscore: 55.24, time: 33.64 sec
epoch: 7, train_loss: 1.0916, train_acc: 55.01, train_fscore: 52.92, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1294, test_acc: 55.21, test_fscore: 55.37, time: 33.97 sec
epoch: 8, train_loss: 1.0449, train_acc: 55.03, train_fscore: 54.01, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1297, test_acc: 57.49, test_fscore: 57.64, time: 34.21 sec
epoch: 9, train_loss: 1.0148, train_acc: 56.83, train_fscore: 55.67, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1273, test_acc: 61.8, test_fscore: 61.75, time: 34.27 sec
epoch: 10, train_loss: 0.9814, train_acc: 59.43, train_fscore: 58.22, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1102, test_acc: 61.49, test_fscore: 61.14, time: 34.52 sec
epoch: 11, train_loss: 0.9534, train_acc: 60.03, train_fscore: 59.18, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0772, test_acc: 62.11, test_fscore: 61.75, time: 33.76 sec
epoch: 12, train_loss: 0.9396, train_acc: 60.46, train_fscore: 59.42, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0559, test_acc: 62.11, test_fscore: 61.56, time: 33.51 sec
epoch: 13, train_loss: 0.9123, train_acc: 61.38, train_fscore: 60.3, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0461, test_acc: 62.11, test_fscore: 61.75, time: 33.65 sec
epoch: 14, train_loss: 0.8989, train_acc: 62.89, train_fscore: 62.01, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0343, test_acc: 62.78, test_fscore: 62.48, time: 33.97 sec
epoch: 15, train_loss: 0.8773, train_acc: 63.29, train_fscore: 62.55, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0187, test_acc: 62.42, test_fscore: 61.92, time: 33.86 sec
epoch: 16, train_loss: 0.8679, train_acc: 63.67, train_fscore: 62.92, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0109, test_acc: 62.78, test_fscore: 62.36, time: 33.71 sec
epoch: 17, train_loss: 0.8399, train_acc: 64.29, train_fscore: 63.49, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.004, test_acc: 62.35, test_fscore: 62.16, time: 33.57 sec
epoch: 18, train_loss: 0.8231, train_acc: 65.01, train_fscore: 64.31, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9979, test_acc: 62.29, test_fscore: 61.92, time: 33.38 sec
epoch: 19, train_loss: 0.8108, train_acc: 65.49, train_fscore: 64.79, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9858, test_acc: 63.22, test_fscore: 62.78, time: 34.12 sec
epoch: 20, train_loss: 0.7963, train_acc: 66.13, train_fscore: 65.61, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9796, test_acc: 62.91, test_fscore: 62.59, time: 33.78 sec
epoch: 21, train_loss: 0.7823, train_acc: 66.73, train_fscore: 66.27, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9785, test_acc: 63.03, test_fscore: 62.67, time: 33.98 sec
epoch: 22, train_loss: 0.7787, train_acc: 66.95, train_fscore: 66.32, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9809, test_acc: 63.09, test_fscore: 62.52, time: 34.46 sec
^[[A^[[B^[[A^H^H^Hepoch: 23, train_loss: 0.7651, train_acc: 67.42, train_fscore: 66.84, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.973, test_acc: 63.09, test_fscore: 62.63, time: 34.29 sec
epoch: 24, train_loss: 0.7607, train_acc: 67.64, train_fscore: 67.28, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9774, test_acc: 63.52, test_fscore: 63.17, time: 34.0 sec
epoch: 25, train_loss: 0.7375, train_acc: 68.5, train_fscore: 68.03, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9726, test_acc: 62.72, test_fscore: 62.21, time: 33.33 sec
epoch: 26, train_loss: 0.7275, train_acc: 69.26, train_fscore: 68.76, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9607, test_acc: 63.03, test_fscore: 62.54, time: 33.61 sec
epoch: 27, train_loss: 0.7145, train_acc: 69.45, train_fscore: 68.94, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9611, test_acc: 63.34, test_fscore: 62.96, time: 33.89 sec
epoch: 28, train_loss: 0.7119, train_acc: 69.85, train_fscore: 69.57, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9736, test_acc: 63.59, test_fscore: 63.1, time: 34.2 sec
epoch: 29, train_loss: 0.696, train_acc: 70.34, train_fscore: 70.0, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9659, test_acc: 62.35, test_fscore: 61.8, time: 34.16 sec
epoch: 30, train_loss: 0.6852, train_acc: 70.77, train_fscore: 70.38, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9594, test_acc: 62.78, test_fscore: 62.07, time: 33.91 sec
epoch: 31, train_loss: 0.6816, train_acc: 71.62, train_fscore: 71.23, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9725, test_acc: 62.6, test_fscore: 62.12, time: 33.81 sec
epoch: 32, train_loss: 0.6707, train_acc: 72.27, train_fscore: 72.15, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9671, test_acc: 63.22, test_fscore: 62.78, time: 34.0 sec
epoch: 33, train_loss: 0.6517, train_acc: 72.82, train_fscore: 72.66, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.955, test_acc: 63.15, test_fscore: 62.56, time: 33.33 sec
epoch: 34, train_loss: 0.6405, train_acc: 73.24, train_fscore: 72.94, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9602, test_acc: 63.15, test_fscore: 62.59, time: 33.68 sec
epoch: 35, train_loss: 0.622, train_acc: 74.17, train_fscore: 73.93, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9675, test_acc: 64.33, test_fscore: 63.72, time: 34.5 sec
epoch: 36, train_loss: 0.6065, train_acc: 75.52, train_fscore: 75.4, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9791, test_acc: 63.89, test_fscore: 63.29, time: 33.68 sec
epoch: 37, train_loss: 0.594, train_acc: 76.33, train_fscore: 76.21, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9676, test_acc: 63.71, test_fscore: 63.24, time: 34.23 sec
epoch: 38, train_loss: 0.583, train_acc: 76.11, train_fscore: 75.95, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9743, test_acc: 63.34, test_fscore: 62.73, time: 34.24 sec
epoch: 39, train_loss: 0.5735, train_acc: 76.83, train_fscore: 76.68, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9841, test_acc: 63.4, test_fscore: 62.74, time: 33.77 sec
epoch: 40, train_loss: 0.5569, train_acc: 77.8, train_fscore: 77.74, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9908, test_acc: 63.46, test_fscore: 62.86, time: 34.12 sec
epoch: 41, train_loss: 0.5516, train_acc: 78.04, train_fscore: 77.94, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9901, test_acc: 63.4, test_fscore: 62.89, time: 34.36 sec
epoch: 42, train_loss: 0.5348, train_acc: 78.61, train_fscore: 78.52, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.996, test_acc: 63.34, test_fscore: 62.83, time: 33.59 sec
epoch: 43, train_loss: 0.5218, train_acc: 79.83, train_fscore: 79.75, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0253, test_acc: 63.03, test_fscore: 62.59, time: 33.42 sec
epoch: 44, train_loss: 0.5122, train_acc: 80.14, train_fscore: 80.11, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 0.9992, test_acc: 62.97, test_fscore: 62.5, time: 34.18 sec
epoch: 45, train_loss: 0.5009, train_acc: 80.28, train_fscore: 80.24, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0139, test_acc: 63.28, test_fscore: 62.7, time: 33.59 sec
epoch: 46, train_loss: 0.4928, train_acc: 81.19, train_fscore: 81.21, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0331, test_acc: 62.35, test_fscore: 61.99, time: 33.94 sec
epoch: 47, train_loss: 0.489, train_acc: 81.33, train_fscore: 81.35, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0631, test_acc: 62.66, test_fscore: 61.03, time: 34.81 sec
epoch: 48, train_loss: 0.4983, train_acc: 80.65, train_fscore: 80.57, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0347, test_acc: 63.34, test_fscore: 63.0, time: 34.18 sec
epoch: 49, train_loss: 0.4788, train_acc: 81.2, train_fscore: 81.21, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0344, test_acc: 63.03, test_fscore: 62.68, time: 34.4 sec
epoch: 50, train_loss: 0.4699, train_acc: 81.55, train_fscore: 81.53, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0759, test_acc: 61.12, test_fscore: 59.89, time: 33.73 sec
```

DialogueGCN Code Result

```
epoch: 51, train_loss: 0.4497, train_acc: 83.65, train_fscore: 83.62, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0606, test_acc: 63.15, test_fscore: 62.86, time: 34.17 sec
epoch: 52, train_loss: 0.4456, train_acc: 83.98, train_fscore: 84.02, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0405, test_acc: 63.59, test_fscore: 63.18, time: 33.82 sec
epoch: 53, train_loss: 0.4248, train_acc: 84.8, train_fscore: 84.85, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0795, test_acc: 61.98, test_fscore: 61.53, time: 33.98 sec
epoch: 54, train_loss: 0.4311, train_acc: 83.65, train_fscore: 83.65, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0639, test_acc: 62.78, test_fscore: 62.39, time: 33.87 sec
epoch: 55, train_loss: 0.4158, train_acc: 85.35, train_fscore: 85.38, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0943, test_acc: 62.23, test_fscore: 61.44, time: 33.6 sec
epoch: 56, train_loss: 0.4129, train_acc: 84.48, train_fscore: 84.49, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.0793, test_acc: 62.6, test_fscore: 62.21, time: 33.51 sec
epoch: 57, train_loss: 0.399, train_acc: 85.47, train_fscore: 85.5, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1083, test_acc: 62.6, test_fscore: 62.06, time: 33.79 sec
epoch: 58, train_loss: 0.3896, train_acc: 86.97, train_fscore: 86.99, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1059, test_acc: 62.11, test_fscore: 61.77, time: 33.6 sec
epoch: 59, train_loss: 0.3824, train_acc: 86.3, train_fscore: 86.3, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1066, test_acc: 61.8, test_fscore: 61.29, time: 33.47 sec
epoch: 60, train_loss: 0.37, train_acc: 87.62, train_fscore: 87.65, valid_loss: nan, valid_acc: nan, valid_fscore: nan, test_loss: 1.1577, test_acc: 60.69, test_fscore: 59.68, time: 34.18 sec
Test performance..
F-Score: 63.72
```

How to use

```
# Prerequisite : Docker on Linux
# 0. Change lakeking -> (YOUR USER NAME)
# 1. Build image by below command
# docker build --tag dialoguegcn:1.0 --build-arg USER_ID=$(id -u) --build-arg GROUP_ID=$(id -g) .
# 2. Run container by below command. Change path of '~/Desktop/2020/2020-1/conv-emotion' to your own code path
# docker run -it --name dialoguegcn --gpus all -v ~/Desktop/2020/2020-1/conv-emotion:/conv-emotion dialoguegcn:1.0 /bin/bash
# 3. Execute command below at /conv-emotion/DialogueGCN
# python train_IEMOCAP.py --base-model 'DialogRNN' --graph-model --nodal-attention --dropout 0.4 --lr 0.0003 --batch-size 32 --class-weight --l2 0.0


FROM pytorch/pytorch:1.5-cuda10.1-cudnn7-devel

RUN apt-get update
RUN pip install pandas
RUN pip install torch-scatter==latest+cu101 -f https://pytorch-geometric.com/whl/torch-1.5.0.html
RUN pip install torch-sparse==latest+cu101 -f https://pytorch-geometric.com/whl/torch-1.5.0.html
RUN pip install torch-cluster==latest+cu101 -f https://pytorch-geometric.com/whl/torch-1.5.0.html
RUN pip install torch-spline-conv==latest+cu101 -f https://pytorch-geometric.com/whl/torch-1.5.0.html
RUN pip install torch-geometric
RUN pip install -U scikit-learn
ARG USER_ID
ARG GROUP_ID
RUN addgroup --gid $GROUP_ID lakeking
RUN adduser --disabled-password --gecos '' --uid $USER_ID --gid $GROUP_ID lakeking
USER lakeking
```

Code Link:
https://drive.google.com/drive/folders/1MAYTK5hPHXW8tTweSKzoDbiTuUupHyeO?usp=sharing
Source from authors: https://github.com/declare-lab/conv-emotion

Code Review: Main Function (1) (train_IEMOCAP.py)

"YG : ~~~" : My own annotation on code.

```python
#   YG: Main function
if __name__ == '__main__':

#   YG: Select models and set train arguments

    path = './saved/IEMOCAP/'

    parser = argparse.ArgumentParser()

    parser.add_argument('--no-cuda', action='store_true', default=False, help='does not use GPU')

    parser.add_argument('--base-model', default='LSTM', help='base recurrent model, must be one of DialogRNN/LSTM/GRU')

    parser.add_argument('--graph-model', action='store_true', default=False, help='whether to use graph model after recurrent encoding')

    parser.add_argument('--nodal-attention', action='store_true', default=False, help='whether to use nodal attention in graph model: Equation 4,5,6 in Paper')

    parser.add_argument('--windowp', type=int, default=10, help='context window size for constructing edges in graph model for past utterances')

    parser.add_argument('--windowf', type=int, default=10, help='context window size for constructing edges in graph model for future utterances')

    parser.add_argument('--lr', type=float, default=0.0001, metavar='LR', help='learning rate')

    parser.add_argument('--l2', type=float, default=0.00001, metavar='L2', help='L2 regularization weight')

    parser.add_argument('--rec-dropout', type=float, default=0.1, metavar='rec_dropout', help='rec_dropout rate')

    parser.add_argument('--dropout', type=float, default=0.5, metavar='dropout', help='dropout rate')

    parser.add_argument('--batch-size', type=int, default=32, metavar='BS', help='batch size')

    parser.add_argument('--epochs', type=int, default=60, metavar='E', help='number of epochs')

    parser.add_argument('--class-weight', action='store_true', default=False, help='use class weights')

    parser.add_argument('--active-listener', action='store_true', default=False, help='active listener')

    parser.add_argument('--attention', default='general', help='Attention type in DialogRNN model')

    parser.add_argument('--tensorboard', action='store_true', default=False, help='Enables tensorboard log')

    args = parser.parse_args()
    print(args)
```

Code Review: Main Function (2) (train_IEMOCAP.py)

```
259     if args.graph_model:
260         seed_everything() ## YG: train_IEMOCAP.py line 14. Set random seed of code.
261         # YG: From model.py line 814
262         model = DialogueGCNModel(args.base_model,
263                                  D_m, D_g, D_p, D_e, D_h, D_a, graph_h,
264                                  n_speakers=2,
265                                  max_seq_len=110,
266                                  window_past=args.windowp,
267                                  window_future=args.windowf,
268                                  n_classes=n_classes,
269                                  listener_state=args.active_listener,
270                                  context_attention=args.attention,
271                                  dropout=args.dropout,
272                                  nodal_attention=args.nodal_attention,
273                                  no_cuda=args.no_cuda)
274
275         print ('Graph NN with', args.base_model, 'as base model.')
276         name = 'Graph'
```

```
13  #   YG: Set random seed of code
14  def seed_everything(seed=seed):
15      random.seed(seed)
16      np.random.seed(seed)
17      torch.manual_seed(seed)
18      torch.cuda.manual_seed(seed)
19      torch.cuda.manual_seed_all(seed)
20      torch.backends.cudnn.benchmark = False
21      torch.backends.cudnn.deterministic = True
```

Code Review: Main Function (3) (train_IEMOCAP.py)

```
321     if args.class_weight:
322         if args.graph_model:
323             loss_function  = nn.NLLLoss(loss_weights.cuda() if cuda else loss_weights)
324         else:
325             loss_function  = MaskedNLLLoss(loss_weights.cuda() if cuda else loss_weights)
326     else:
327         if args.graph_model:
328             loss_function = nn.NLLLoss()
329         else:
330             loss_function = MaskedNLLLoss()
331 
332     optimizer = optim.Adam(model.parameters(), lr=args.lr, weight_decay=args.l2)
333 
334     train_loader, valid_loader, test_loader = get_IEMOCAP_loaders(valid=0.0,
335                                                                   batch_size=batch_size,
336                                                                   num_workers=0)
337 
338     best_fscore, best_loss, best_label, best_pred, best_mask = None, None, None, None, None
339     all_fscore, all_acc, all_loss = [], [], []
```

```
23  #   YG: Separate train set and valid set
24  def get_train_valid_sampler(trainset, valid=0.1):
25      size = len(trainset)
26      idx = list(range(size))
27      split = int(valid*size)
28      return SubsetRandomSampler(idx[split:]), SubsetRandomSampler(idx[:split])
29 
30  #   YG: Dataloader from IEMOCAPDataset(), dataloader.py line 6.
31  def get_IEMOCAP_loaders(batch_size=32, valid=0.1, num_workers=0, pin_memory=False):
32      trainset = IEMOCAPDataset()
33      train_sampler, valid_sampler = get_train_valid_sampler(trainset, valid)
34 
35      train_loader = DataLoader(trainset,
36                                batch_size=batch_size,
37                                sampler=train_sampler,
38                                collate_fn=trainset.collate_fn,
39                                num_workers=num_workers,
40                                pin_memory=pin_memory)
41 
42      valid_loader = DataLoader(trainset,
43                                batch_size=batch_size,
44                                sampler=valid_sampler,
45                                collate_fn=trainset.collate_fn,
46                                num_workers=num_workers,
47                                pin_memory=pin_memory)
48 
49      testset = IEMOCAPDataset(train=False)
50      test_loader = DataLoader(testset,
51                               batch_size=batch_size,
52                               collate_fn=testset.collate_fn,
53                               num_workers=num_workers,
54                               pin_memory=pin_memory)
55 
56      return train_loader, valid_loader, test_loader
```

Code Review: Main Function (4) (train_IEMOCAP.py)

```
341     for e in range(n_epochs):
342         start_time = time.time()
343 
344         if args.graph_model:
345             # YG: These lines are executed while train, valid and test
346             train_loss, train_acc, _, _, train_fscore, _, _, _, _, _ = train_or_eval_graph_model(model, loss_function, train_loader, e, cuda, optimizer, True)
347             valid_loss, valid_acc, _, _, valid_fscore, _, _, _, _, _ = train_or_eval_graph_model(model, loss_function, valid_loader, e, cuda)
348             test_loss, test_acc, test_label, test_pred, test_fscore, _, _, _, _, _ = train_or_eval_graph_model(model, loss_function, test_loader, e, cuda)
349             all_fscore.append(test_fscore)
350             # torch.save({'model_state_dict': model.state_dict()}, path + name + args.base_model + '_' + str(e) + '.pkl')
```

```
359         # YG: Display statistics
360         if args.tensorboard:
361             writer.add_scalar('test: accuracy/loss', test_acc/test_loss, e)
362             writer.add_scalar('train: accuracy/loss', train_acc/train_loss, e)
363 
364         print('epoch: {}, train_loss: {}, train_acc: {}, train_fscore: {}, valid_loss: {}, valid_acc: {}, valid_fscore: {}, test_loss: {}, test_acc: {}, test_fscore: {},
365                 format(e+1, train_loss, train_acc, train_fscore, valid_loss, valid_acc, valid_fscore, test_loss, test_acc, test_fscore, round(time.time()-start_time, 2)))
366 
367 
368     if args.tensorboard:
369         writer.close()
370 
371     print('Test performance..')
372     print ('F-Score:', max(all_fscore))
```

Code Review: Train/Eval Function (1) (train_IEMOCAP.py)

```python
#   YG : Train & Evaluate model. Model description in model.py
def train_or_eval_graph_model(model, loss_function, dataloader, epoch, cuda, optimizer=None, train=False):
    losses, preds, labels = [], [], []
    scores, vids = [], []

    ei, et, en, el = torch.empty(0).type(torch.LongTensor), torch.empty(0).type(torch.LongTensor), torch.empty(0), []

    #if torch.cuda.is_available():
    if cuda:
        ei, et, en = ei.cuda(), et.cuda(), en.cuda()

    assert not train or optimizer!=None
    if train:
        model.train() ## YG: Set train mode so do back-propagate or dropout on iteration
    else:
        model.eval()  ## YG: Set evaluatikon mode so do not back-propagate or dropout on iteration

    # YG: train_IEMOCAP.py line 14. Set random seed of code
    seed_everything()
```

Code Review: Train/Eval Function (2) (train_IEMOCAP.py)

```
137     for data in dataloader:
138         if train:
139             optimizer.zero_grad()
140 
141         textf, visuf, acouf, qmask, umask, label = [d.cuda() for d in data[:-1]] if cuda else data[:-1]
142 
143         lengths = [(umask[j] == 1).nonzero().tolist()[-1][0] + 1 for j in range(len(umask))]
144         # YG: We use
145         # YG: textf: Utturance embedded as 100 dimension by CNN
146         # YG: qmask: Speaker ID
147         # YG: umask: Whether emotion labels are assigned or not
148         # YG: label: Emotion labels of each tasks
149         log_prob, e_i, e_n, e_t, e_l = model(textf, qmask, umask, lengths)
150         # YG: Flatten label matrix to vector to calculate loss function
151         label = torch.cat([label[j][:lengths[j]] for j in range(len(label))])
152         # YG: line 320. NLLloss
153         loss = loss_function(log_prob, label)
154     # YG: Store results to return & display statistics
155         ei = torch.cat([ei, e_i], dim=1)
156         et = torch.cat([et, e_t])
157         en = torch.cat([en, e_n])
158         el += e_l
159 
160         preds.append(torch.argmax(log_prob, 1).cpu().numpy())
161         labels.append(label.cpu().numpy())
162         losses.append(loss.item())
163 
164         if train:
165             loss.backward()
166             if args.tensorboard:
167                 for param in model.named_parameters():
168                     writer.add_histogram(param[0], param[1].grad, epoch)
169             optimizer.step()
170 
```

Code Review: Train/Eval Function (3) (train_IEMOCAP.py)

```
171     if preds!=[]:
172         preds  = np.concatenate(preds)
173         labels = np.concatenate(labels)
174     else:
175         return float('nan'), float('nan'), [], [], float('nan'), [], [], [], [], []
176 
177     vids += data[-1]
178     ei = ei.data.cpu().numpy()
179     et = et.data.cpu().numpy()
180     en = en.data.cpu().numpy()
181     el = np.array(el)
182     labels = np.array(labels)
183     preds = np.array(preds)
184     vids = np.array(vids)
185 
186     avg_loss = round(np.sum(losses)/len(losses), 4)
187     avg_accuracy = round(accuracy_score(labels, preds)*100, 2)
188     avg_fscore = round(f1_score(labels,preds, average='weighted')*100, 2)
189 
190     return avg_loss, avg_accuracy, labels, preds, avg_fscore, vids, ei, et, en, el
```

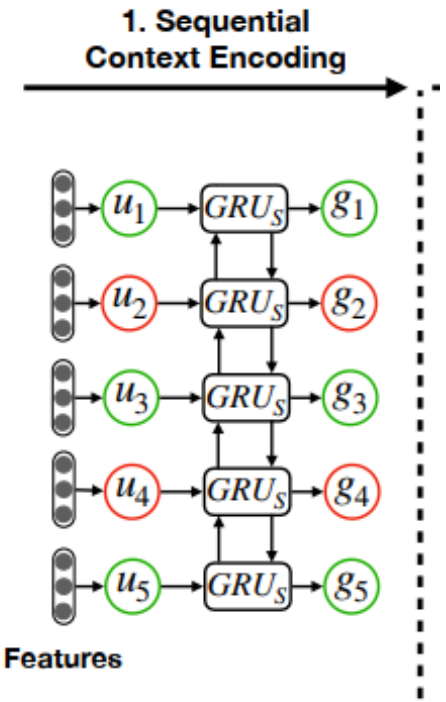
Code Review: Dataset (dataloader.py)

```python
class IEMOCAPDataset(Dataset):

    def __init__(self, train=True):
        self.videoIDs, self.videoSpeakers, self.videoLabels, self.videoText,\
        self.videoAudio, self.videoVisual, self.videoSentence, self.trainVid,\
        self.testVid = pickle.load(open('./IEMOCAP_features/IEMOCAP_features.pkl', 'rb'), encoding='latin1')
        '''
        label index mapping = {'hap':0, 'sad':1, 'neu':2, 'ang':3, 'exc':4, 'fru':5}
        '''
        self.keys = [x for x in (self.trainVid if train else self.testVid)]

        self.len = len(self.keys)

    def __getitem__(self, index):
        vid = self.keys[index]
        # See train_IEMOCAP.py line 144.
        return torch.FloatTensor(self.videoText[vid]),\
               torch.FloatTensor(self.videoVisual[vid]),\
               torch.FloatTensor(self.videoAudio[vid]),\
               torch.FloatTensor([[1,0] if x=='M' else [0,1] for x in\
                                  self.videoSpeakers[vid]]),\
               torch.FloatTensor([1]*len(self.videoLabels[vid])),\
               torch.LongTensor(self.videoLabels[vid]),\
               vid

    def __len__(self):
        return self.len

    def collate_fn(self, data):
        dat = pd.DataFrame(data)
        return [pad_sequence(dat[i]) if i<4 else pad_sequence(dat[i], True) if i<6 else dat[i].tolist() for i in dat]
```

Code Review: Model (1) (model.py)

```python
class DialogueGCNModel(nn.Module):

    def __init__(self, base_model, D_m, D_g, D_p, D_e, D_h, D_a, graph_hidden_size, n_speakers, max_seq_len, window_past, window_future,
                 n_classes=7, listener_state=False, context_attention='simple', dropout_rec=0.5, dropout=0.5, nodal_attention=True, avec=False, no_cuda=False):

        super(DialogueGCNModel, self).__init__()

        self.base_model = base_model
        self.avec = avec
        self.no_cuda = no_cuda

        # The base model is the sequential context encoder.
        if self.base_model == 'DialogRNN':
            self.dialog_rnn_f = DialogueRNN(D_m, D_g, D_p, D_e, listener_state, context_attention, D_a, dropout_rec)
            self.dialog_rnn_r = DialogueRNN(D_m, D_g, D_p, D_e, listener_state, context_attention, D_a, dropout_rec)

        elif self.base_model == 'LSTM':
            self.lstm = nn.LSTM(input_size=D_m, hidden_size=D_e, num_layers=2, bidirectional=True, dropout=dropout)

        elif self.base_model == 'GRU':
            self.gru = nn.GRU(input_size=D_m, hidden_size=D_e, num_layers=2, bidirectional=True, dropout=dropout)


        elif self.base_model == 'None':
            self.base_linear = nn.Linear(D_m, 2*D_e)

        else:
            print ('Base model must be one of DialogRNN/LSTM/GRU')
            raise NotImplementedError
```

```python
    def forward(self, U, qmask, umask, seq_lengths):
        """
        U -> seq_len, batch, D_m
        qmask -> seq_len, batch, party
        """
        if self.base_model == "DialogRNN":

            if self.avec:
                emotions, _ = self.dialog_rnn_f(U, qmask)

            else:
                emotions_f, alpha_f = self.dialog_rnn_f(U, qmask) # seq_len, batch, D_e
                rev_U = self._reverse_seq(U, umask)
                rev_qmask = self._reverse_seq(qmask, umask)
                emotions_b, alpha_b = self.dialog_rnn_r(rev_U, rev_qmask)
                emotions_b = self._reverse_seq(emotions_b, umask)
                emotions = torch.cat([emotions_f,emotions_b],dim=-1)

        elif self.base_model == 'LSTM':
            emotions, hidden = self.lstm(U)

        elif self.base_model == 'GRU':
            emotions, hidden = self.gru(U)

        elif self.base_model == 'None':
            emotions = self.base_linear(U)
```

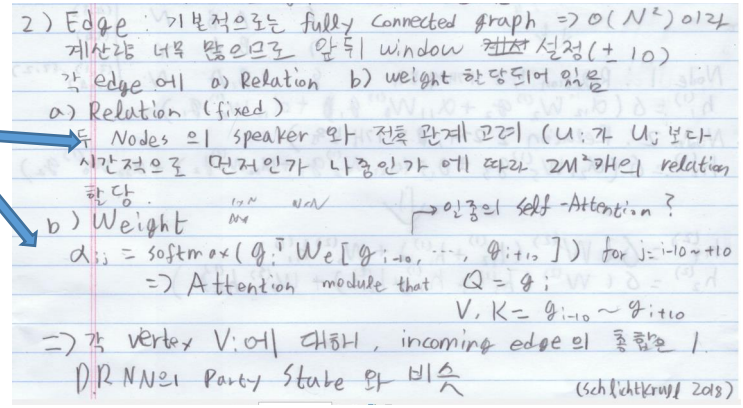


Code Review: Model (2) (model.py)

```
844         n_relations = 2 * n_speakers ** 2
845         self.window_past = window_past
846         self.window_future = window_future
847
848         self.att_model = MaskedEdgeAttention(2*D_e, max_seq_len, self.no_cuda)
849         self.nodal_attention = nodal_attention
853         edge_type_mapping = {}
854         for j in range(n_speakers):
855             for k in range(n_speakers):
856                 edge_type_mapping[str(j) + str(k) + '0'] = len(edge_type_mapping)
857                 edge_type_mapping[str(j) + str(k) + '1'] = len(edge_type_mapping)
858
859         self.edge_type_mapping = edge_type_mapping
```

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{W})$$

Vertices: Each utterance in the conversation is represented as a vertex $v_i \in \mathcal{V}$ in $\mathcal{G}$. Each vertex v_i is initialized with the corresponding sequentially encoded feature vector g_i, for all $i \in [1, 2, ..., N]$.




```
905         features, edge_index, edge_norm, edge_type, edge_index_lengths = batch_graphify(emotions, qmask, seq_lengths, self.window_past, self.window_future, self.edge_type_mapping, self.att_model, self.no_cuda)
```

```
661 def batch_graphify(features, qmask, lengths, window_past, window_future, edge_type_mapping, att_model, no_cuda):
662     """
663     Method to prepare the data format required for the GCN network. Pytorch geometric puts all nodes for classification
664     in one single graph. Following this, we create a single graph for a mini-batch of dialogue instances. This method
665     ensures that the various graph indexing is properly carried out so as to make sure that, utterances (nodes) from
666     each dialogue instance will have edges with utterances in that same dialogue instance, but not with utternaces
667     from any other dialogue instances in that mini-batch.
668     """
```

$$\alpha_{ij} = \text{softmax}(g_i^T W_e [g_{i-p}, \ldots, g_{i+f}]),$$
$$\text{for } j = i-p, \ldots, i+f.$$

Code Review: Model (3) (model.py)

```
851         self.graph_net = GraphNetwork(2*D_e, n_classes, n_relations, max_seq_len, graph_hidden_size, dropout, self.no_cuda)
906         log_prob = self.graph_net(features, edge_index, edge_norm, edge_type, seq_lengths, umask, self.nodal_attention, self.avec)
```

```
789 class GraphNetwork(torch.nn.Module):
790     def __init__(self, num_features, num_classes, num_relations, max_seq_len, hidden_size=64, dropout=0.5, no_cuda=False):
791         """
792         The Speaker-level context encoder in the form of a 2 layer GCN.
793         """
794         super(GraphNetwork, self).__init__()
795 
796         self.conv1 = RGCNConv(num_features, hidden_size, num_relations, num_bases=30)  ## YG: Implemented in torch_geometric package
797         self.conv2 = GraphConv(hidden_size, hidden_size) ## YG: Implemented in torch_geometric package
798         self.matchatt = MatchingAttention(num_features+hidden_size, num_features+hidden_size, att_type='general2')
799         self.linear   = nn.Linear(num_features+hidden_size, hidden_size)
800         self.dropout = nn.Dropout(dropout)
801         self.smax_fc  = nn.Linear(hidden_size, num_classes)
802         self.no_cuda = no_cuda
803 
804     def forward(self, x, edge_index, edge_norm, edge_type, seq_lengths, umask, nodal_attn, avec):
805 
806         out = self.conv1(x, edge_index, edge_type, edge_norm)
807         out = self.conv2(out, edge_index)
808         emotions = torch.cat([x, out], dim=-1)
809         log_prob = classify_node_features(emotions, seq_lengths, umask, self.matchatt, self.linear, self.dropout, self.smax_fc, nodal_attn, avec, self.no_cuda)
810         return log_prob
```

R-GCN

$$h_i^{(1)} = \sigma\Big(\sum_{r\in\mathcal{R}}\sum_{j\in N_i^r}\frac{\alpha_{ij}}{c_{i,r}}W_r^{(1)}g_j + \alpha_{ii}W_0^{(1)}g_i\Big),$$
$$\text{for } i = 1, 2, \ldots, N,$$

Simple GNN

$$h_i^{(2)} = \sigma\Big(\sum_{j\in N_i^r} W^{(2)}h_j^{(1)} + W_0^{(2)}h_i^{(1)}\Big),$$
$$\text{for } i = 1, 2, \ldots, N,$$

Code Review: Model (4) (model.py)

```python
def classify_node_features(emotions, seq_lengths, umask, matchatt_layer, linear_layer, dropout_layer, smax_fc_layer, nodal_attn, avec, no_cuda):
    """
    Function for the final classification, as in Equation 7, 8, 9. in the paper.
    """

    if nodal_attn:

        emotions = attentive_node_features(emotions, seq_lengths, umask, matchatt_layer, no_cuda)
        hidden = F.relu(linear_layer(emotions))
        hidden = dropout_layer(hidden)
        hidden = smax_fc_layer(hidden)

        if avec:
            return torch.cat([hidden[:, j, :][:seq_lengths[j]] for j in range(len(seq_lengths))])

        log_prob = F.log_softmax(hidden, 2)
        log_prob = torch.cat([log_prob[:, j, :][:seq_lengths[j]] for j in range(len(seq_lengths))])
        return log_prob

    else:

        hidden = F.relu(linear_layer(emotions))
        hidden = dropout_layer(hidden)
        hidden = smax_fc_layer(hidden)

        if avec:
            return hidden

        log_prob = F.log_softmax(hidden, 1)
        return log_prob
```

$$
\begin{aligned}
h_i &= [g_i, h_i^{(2)}], \\
\beta_i &= \text{softmax}(h_i^T W_\beta [h_1, h_2 \ldots, h_N]), \\
\tilde{h}_i &= \beta_i [h_1, h_2, \ldots, h_N]^T.
\end{aligned}
$$

Attention Score

$$
\begin{aligned}
l_i &= \text{ReLU}(W_l \tilde{h}_i + b_l), \\
\mathcal{P}_i &= \text{softmax}(W_{smax} l_i + b_{smax}), \\
\hat{y}_i &= \underset{k}{\text{argmax}}(\mathcal{P}_i[k]).
\end{aligned}
$$

FCN