腾讯项目组汇报（四十七）--2020/02/26

1. 论文模型相关公式

1. Language Model Part

For each sentence $s_i = \{w_1, w_2, \ldots, w_m\}$ in the bag with `m` words, following other pretrained models(add references later) we first covert each sentence s_i into a sequence of byte pair tokens $b_i = \{bp_1, bp_2, \ldots, bp_k\}$ via byte pair encoding, then utilize these tokens b_i as transformer decoder's input.

Sentence with m words:

$$s_i = \{w_1, w_2, \ldots, w_{m_i}\}$$

Byte pair encoded sequence for each sentence:

$$b_i = \{bp_1, bp_2, \ldots, bp_{n_i}\}, where \;\; n_i \geq m_i$$

Byte pair sentences matrix/Transformer decoder input $X \in R^{B*N}$:

$$X = \{b_1, b_2, \ldots, b_B\}$$

where B is batch size, N is max byte pair sequence length in a batch, that is, $N = max(n_1, n_2, \ldots, n_B)$.

Transformer decoder `l` layer output for each sentence $H_i^l \in R^{N*d}$:

$$H_i^l = \{h_1^l, h_2^l, \ldots, h_N^l\}$$

, where d is hidden dim, and corresponding `l` layer sentence representations matrix is $H^l = \{H_1^l, H_2^l, \ldots, H_B^l\}, H^l \in R^{B*N*d}$.

2. Dependency link recall & Dependency type embed part

2.1. Byte Pair Dependency Tree

Since NLP tools implement dependency parse for each sentence at word level, ~~the resulting~~ regular dependency tree can not naturally capture all dependency relations between byte pairs of a sentence. Instead of directly modeling ~~dependency tree~~

regular one as a directed~~, labeled~~ graph of words (add reference later), we proposed a novel variant ~~of dependency tree~~ -- Byte Pair Dependency Tree for compatibility with ~~all~~ pretrained language models based on byte pair encoding ~~(which is consistent with Robinson's principles (add reference later) and compatible with all pretrained language models based on byte pair encoding)~~. In a byte pair encoded sentence, each word may be encoded into multiple byte pair tokens, this one-to-many encoding causes regular ~~word-level~~ dependency tree ~~missing out~~ unlinking several ~~byte pair~~ tokens, resulting in an unconnected graph (need graph illustration example). To avoid this issue, ~~our~~ byte pair dependency tree novelly assign the last byte pair token ~~of a word~~ with dependency typed edges of its corresponding word, and complete rest tokens with ~~dependency~~ `compound` typed edges ~~to~~ heading the last token (need graph illustration example). Following (add reference later), Byte Pair Dependency tree also use an edge set including self loops with `self` type and inverse edges. With adaptions above, we ~~can~~ consider Byte Pair Dependency Tree as a byte pair level connected graph $G = (V, \epsilon)$ with directed and typed edges, where nodes V consist of all byte pairs (i.e., holi, day) and edge set contains all typed edges between byte pairs. Further, our model utilize this graph feature as input to recall dependency edges and incorporate dependency type embedding.

2.2. Incorporating dependency edges recall & dependency types embedding

Several works have proven pretrained language models are capable of automatically capturing semantic and syntactic features from unstructured plain text, and also a notable amount of "common-sense" knowledge. Naturally, we hypothesize attention weights of transformer could provide an implicit clue about important dependency edges. In our model, we utilize a learned variable θ and attention weights averaged over transformer layers to recall dependency edges missing by NLP tools. In detail, each edge with higher attention than θ would be explicitly added into Byte Pair Dependency Tree Matrix with `unk` type. Despite of not strictly complying with principles of dependency parse, those edges could benefit subsequent GAT to aggregate more potential neighbors' information for relation extraction.

given Transformer decoder's `l` layer attention coef matrix $\alpha^{l}_{gpt} \in R^{B*heads*N*N}$, the averaged attention weights $\alpha_{gpt-avg} \in R^{B*heads*N*N}$ as follow:

$$\alpha_{gpt-avg} = \frac{1}{L}\sum_{l=0}^{L}\alpha^{l}_{gpt}$$

Edeges Recalled by attetion weights $E_{recall} \in R^{B*N*N}$:

$$E_{recall} = \alpha_{gpt-avg} > \theta$$

updated byte pair dependency edges matrix $E^{'} \in R^{B*N*N}$:

$$E^{'} = E \oplus E_{recall}$$

where $E \in R^{B*N*N}$ is byte pair dependency edges matrix derived from byte pair dependency tree, $\oplus$ represents logic OR operator.

updated byte pair dependency edges matrix with dependency type embedding $E^{embed} \in R^{B*N*N*p}$:

$$E^{embed} = embed(E^{'})$$

incorporate edge type embedding and head node features to compute attention coef for aggregating edges features from neighborhood:

$$\beta_{ij} = \frac{exp\{w^T[h_i||WE_{ij\cdot}^{embed}]\}}{\sum_k exp\{w^T[h_i||WE_{ik\cdot}^{embed}]\}}$$

edges type feature aggregation function / aggregated edge type feature of node i :

$$h_i^{edge} = ||_{k=1}^{K} \delta(\sum_{j \in N(i)} \beta_{ij}^k WE_{ij\cdot}^{embed})$$

concatenate dependency type features aggregated from node i's dependency edges and node feature aggregated from its dependency edge neighborhood, for subsequent relation classification:

$$h_i^{final} = h_i||h_i^{edge}$$