報告書

1 今週の進捗

- 正解データのみを入力としたテスト結果
- 積み上げ棒グラフは未完成
- MLM を用いた実験コードは未完成

2 KG-BERT [1]

2.1 モデルの説明

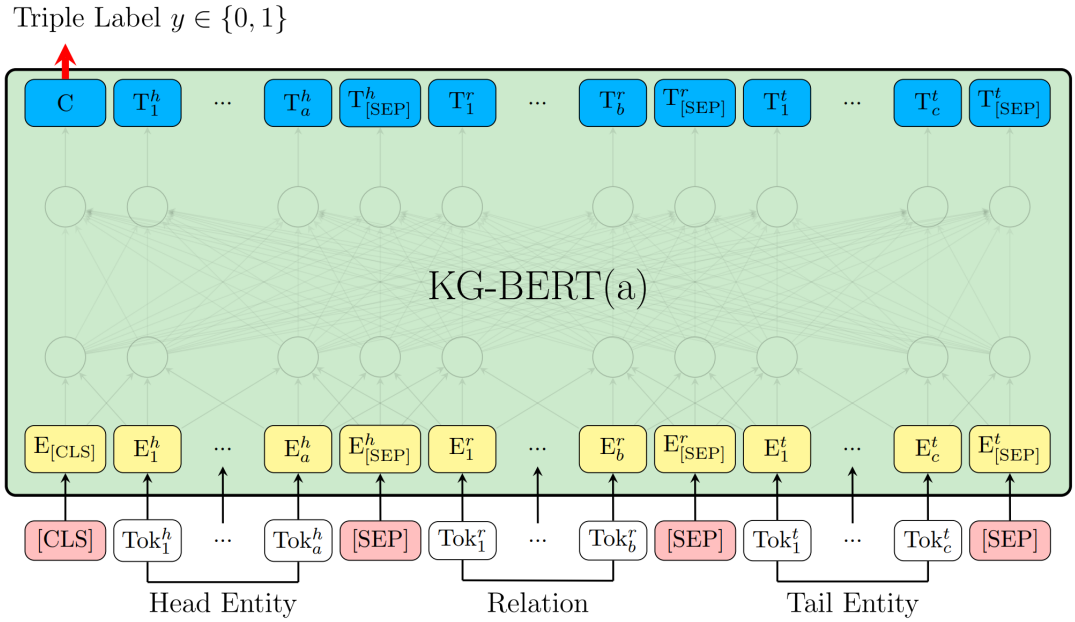

図 1: KG-BERT model [1]

表 1: データセット

Dataset	Entity	Relation	Train	Validation	Test
WN18RR	40,943	11	86,835	3,034	3,134

評価指標として Mean Rank (MR), Mean Reciprocal Rank (MRR), Hits@k を使用する. MR とは, 予測したエンティティのランクの平均を指す. MRR とは, 予測したエンティティのランクの逆数をスコアとしており, こうして得たスコアの平均をとったものを指す. Hits@k とは, 予測したエンティティを順位付けしたときに, 上位 k 個以内に正解が含まれている割合のことを指す.

表 2: パラメータ (WN18RR)

パラメータ	値 (default)
学習率	5e-5 (5e-5)
epoch	5.0 (5.0)
dropout rate	0.1 (0.1)
batch size	32 (32)
eval batch size	128 (5000)
max seq length	50 (50)

$$\mathrm{MR} = \frac{1}{|\mathrm{E}|}\sum_{i=1}^{|\mathrm{E}|}\mathrm{rank}_i \tag{1}$$

$$\mathrm{MRR} = \frac{1}{|\mathrm{E}|}\sum_{i=1}^{|\mathrm{E}|}\frac{1}{\mathrm{rank}_i} \tag{2}$$

|E| はエンティティ数，rank_i は予測したエンティティのランクを表している．
MR は値が小さいとき, MMR, Hits@ k はともに値が大きいとき推定精度が良いと判断される．

2.2 テスト数 10, 100, 250, 500 の実験結果の比較

テストデータからランダムにテスト数 {10, 100, 250, 500} だけトリプルを取ってきてテストする. 表 3 にそれぞれの結果を示す. 以下のテスト数 {10, 100, 250, 500} の実験はすべて同じ学習済みモデルを使用している.

表 3: テスト数ごとの実験結果

	WN18RR					
テスト数	MR	MRR	Hits@1	Hits@3	Hits@10	実験時間
3,134 (文献値)	97	-	-	-	52.4	
3,134 (再現実験)	117.77	0.25	12.41	29.44	51.85	約 10 日
10	54.7	0.0958	0.0	0.0	50.0	約 0.5 日
100	20.84	0.1238	0.0	9.68	51.61	約 2 日
250	85.528	0.146	5.2	12.4	38.0	約 4 日
500	81.332	0.131	3.80	12.60	33.00	約 8 日

テスト数が少なくなると文献値や再現実験の結果とは異なる結果か得られるため, テスト数を極端に減らすことは不正確な結果につながると考えられる. しかし, テスト数を増やすと実験時間がかかりすぎてしまう.
1 つのテストトリプルに対して約 40,000 個の存在しないトリプルを作成してテストしているが, これを 10,000 個まで減らして実験することで実験時間の短縮が望まれる. 10,000 個の存在しないトリプルの選択方法としてはランダム, もしくはテストトリプルにある tail と似た tail を用いて作成したトリプルを選択する方法が考えられる.

2.2.1 正解データだけでテストした結果

表 4 に正解トリプルだけをテスト時の入力として実験した結果を示す.

表 4: 正解データだけの実験結果

テスト数	正解数	Accuracy
3,134	3,000	0.9572

2.3 エンティティ数 (40,943) 分類

2.3.1 存在するトリプルの数の分だけ上位にあるトリプルの割合を指標とした実験

あるトリプルの tail を予測するとき, その head, relation をもつ他のトリプルの数を N, 予測した結果の上位 N のうち存在するトリプルの数を n とすると, 指標は以下のように表すことができる.

$$\frac{n}{N} \tag{3}$$

前回言っていた棒グラフを積み上げた形の図はまだ作成できていない.

2.4 MLM を用いた実験

現在, プログラムを書いており, まだ完成していない. 以下にコードと現在出ているエラー内容を示す.

Listing 1: MLM

```
def convert_examples_to_features(examples, label_list, max_seq_length, tokenizer, print_info = True):
    features = []
    for (ex_index, example) in enumerate(examples):
        if ex_index % 10000 == 0 and print_info:
            logger.info("Writing␣example␣%d␣of␣%d" % (ex_index, len(examples)))

        tokens_a = tokenizer.tokenize(example.text_a)

        tokens_b = None
        tokens_c = None

        if example.text_b and example.text_c:
            tokens_b = tokenizer.tokenize(example.text_b)
            tokens_c = tokenizer.tokenize(example.text_c)
            _truncate_seq_triple(tokens_a, tokens_b, tokens_c, max_seq_length - 4)
        else:
            if len(tokens_a) > max_seq_length - 2:
                tokens_a = tokens_a[:(max_seq_length - 2)]

        tokens = ["[CLS]"] + tokens_a + ["[SEP]"]

        if tokens_b:
            tokens += tokens_b + ["[SEP]"]
        if tokens_c:
            tokens += tokens_c + ["[SEP]"]

        tokens = tokenizer.encode(tokens, add_special_tokens=True, max_length=max_seq_length, padding="max_length", truncation=True)
        input_ids = tokenizer.convert_tokens_to_ids(tokens)

```

```
        features.append(InputFeatures(input_ids))

    return features

train_features = convert_examples_to_features(
            train_examples, label_list, args.max_seq_length, tokenizer)
        logger.info("*****␣Running␣training␣*****")
        logger.info("␣␣Num␣examples␣=␣%d", len(train_examples))
        logger.info("␣␣Batch␣size␣=␣%d", args.train_batch_size)
        logger.info("␣␣Num␣steps␣=␣%d", num_train_optimization_steps)
        all_input_ids = torch.tensor([f.input_ids for f in train_features], dtype=torch.long)

        train_data = TensorDataset(all_input_ids)

        if args.local_rank == -1:
            train_sampler = RandomSampler(train_data)
        else:
            train_sampler = DistributedSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=args.train_batch_size
    )

model.train()
tr_loss_list = [] # トレーニングロスを記録するためのリスト

# 指定されたエポック数にわたるトレーニングループ
for _ in trange(int(args.num_train_epochs), desc="Epoch"):
    tr_loss = 0 # 各エポックの合計ロス
    nb_tr_examples, nb_tr_steps = 0, 0 # トレーニングサンプル数とステップ数を追跡

    # トレーニングデータローダーをイテレート
    for step, batch in enumerate(tqdm(train_dataloader, desc="Iteration")):
        inputs = data_collator(batch) # バッチデータを前処理

        # モデルに入力データを提供し、ロスを計算
        logits = model(**inputs, return_dict=True)
        loss_fct = CrossEntropyLoss()
        loss = loss_fct(logits.view(-1, num_labels), label_ids.view(-1))

        # ロスの蓄積（勾配の蓄積）が行われる場合の処理
        if args.gradient_accumulation_steps > 1:
            loss = loss / args.gradient_accumulation_steps

        # 勾配の計算とモデルパラメータの更新
        loss.backward()

        tr_loss += loss.item()
        nb_tr_examples += input_ids.size(0) # トレーニングサンプル数の更新
        nb_tr_steps += 1

        # 勾配の蓄積が行われる場合、指定されたステップごとに最適化プロセスを実行
        if (step + 1) % args.gradient_accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()
```

```
82              global_step += 1 # グローバルステップを更新
83
84      tr_loss_list.append(tr_loss) # このエポックのトレーニングロスを記録
85      print("Training␣loss:␣", tr_loss, nb_tr_examples) # トレーニングロスを表示
```

Listing 2: Error Message

```
1  Traceback (most recent call last):
2    File "/home/horimotoデスクトップ//B4/kg-bert/run_bert_mlm.py", line 1226, in <module>
3      main()
4    File "/home/horimotoデスクトップ//B4/kg-bert/run_bert_mlm.py", line 871, in main
5      inputs = data_collator(batch) # バッチデータを前処理
6    File "/home/horimoto/.pyenv/versions/anaconda3-2022.05/lib/python3.9/site-packages/transformers/
         data/data_collator.py", line 45, in __call__
7      return self.torch_call(features)
8    File "/home/horimoto/.pyenv/versions/anaconda3-2022.05/lib/python3.9/site-packages/transformers/
         data/data_collator.py", line 741, in torch_call
9      batch["input_ids"], batch["labels"] = self.torch_mask_tokens(
10   File "/home/horimoto/.pyenv/versions/anaconda3-2022.05/lib/python3.9/site-packages/transformers/
         data/data_collator.py", line 768, in torch_mask_tokens
11     probability_matrix.masked_fill_(special_tokens_mask, value=0.0)
12 RuntimeError: The size of tensor a (128) must match the size of tensor b (16) at non-singleton
       dimension 2
```

エラーメッセージでは特殊トークンの処理中にサイズが異なっていることを示している.

3 今後したいこと

- KG-BERT に MLM を適用して実装
- エンティティ数 (40,943) 分類の改良
- KG-BERT の改良

参考文献

[1] Liang Yao, Chengsheng Mao, and Yuan Luo. KG-BERT: BERT for knowledge graph completion. *CoRR*, Vol. abs/1909.03193, , 2019.