Chinese News Title Classification

Xinran Xue
z5438685
z5438685@ad.unsw.edu.au

April 27, 2025

Table of Contents

1. Abstract

This project explores the application of **BERT** for sentence classification in the context of Chinese news articles. The task involves fine-tuning a pre-trained BERT model to categorize sentences from the **Toutiao** dataset into one of 15 predefined topics, such as politics, entertainment, and sports.

The experiment was conducted using a standard BERT configuration with 12 Transformer layers and 12 attention heads. The model was trained on a dataset consisting of labeled Chinese news sentences, employing a training setup with a batch size of 64, a learning rate of 5e-5, and 10 epochs.

The evaluation of the model was performed on a test set, with accuracy as the primary evaluation metric. Results showed a promising performance, with the model achieving high accuracy in classifying sentences into their respective topics. The setup also employed **TensorBoard** to monitor training progress and visualize metrics such as training loss and accuracy over epochs.

This experiment demonstrates the effectiveness of fine-tuning pre-trained BERT models for text classification tasks in low-resource languages, particularly for applications involving Chinese language text.

2. Data Processing

2.1. Data collection

In this experiment, the data collection process is facilitated by Apache Kafka, a distributed messaging system designed for high-throughput, fault-tolerant data streaming. Kafka is used to stream news articles from various sources in real-time and send them to a processing pipeline.

News articles are categorized and published to different Kafka topics.I as Kafka consumers subscribe to the relevant topics and fetch the data in real-time. For this project, a Kafka consumer is set up to pull data from the topic containing news articles.

I retrieve messages (news articles) from Kafka, where each message is in **json** format.I can retrieve the title from **Title** field and label from **Section** field. These sentences are then passed through the preprocessing pipeline.

After receiving the message,I put the label and title together with “_!_” symbol,finally saved to a text file.The result is as follows:

2.2. Data Preprocessing

2.2.1. Tokenization

Tokenization is the process of converting raw text into smaller units, such as words or subwords, which are then represented as integer IDs. In this experiment, I use BERT's pre-trained tokenizer for Chinese text. BERT's tokenizer uses a **WordPiece** model, which breaks words into subword units.

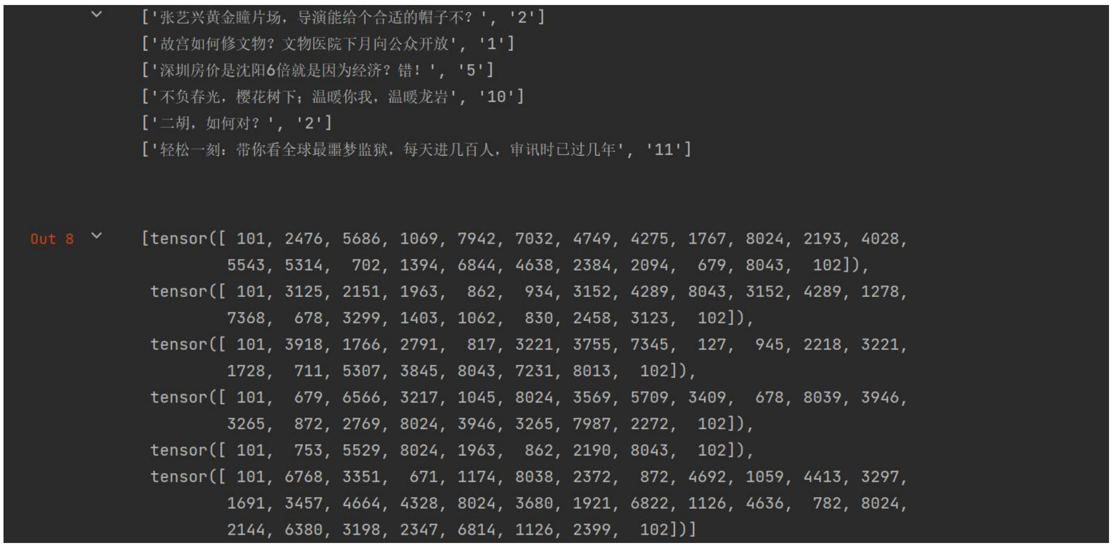
The **vocab.txt** file contains a list of all the tokens (words or subwords) that the BERT model recognizes. Each token is associated with a unique index, and this index is used to convert tokens into integer IDs.

2.2.2. Padding and Token ID Conversion

Once tokenized, each token is converted into a unique token ID using the BERT vocabulary. This is necessary because BERT processes numerical data, and each token in the sentence needs to be represented by an integer corresponding to its position in the vocabulary.

All sequences in a batch must have the same length. Since sentences can vary in length, padding is applied to make them consistent. This is done by adding the [PAD](0) token to sentences that are shorter than the longest sentence in the batch. If a sentence exceeds the maximum sequence length (512 tokens for BERT), it is truncated.

Example: [CLS] as 101,[SEP] as 102



```
['张艺兴黄金瞳片场，导演能给个合适的帽子不？', '2']
['故宫如何修文物？文物医院下月向公众开放', '1']
['深圳房价是沈阳6倍就是因为经济？错！', '5']
['不负春光，樱花树下；温暖你我，温暖龙岩', '10']
['二胡，如何对？', '2']
['轻松一刻：带你看全球最噩梦监狱，每天进几百人，审讯时已过几年', '11']


Out 8   [tensor([ 101, 2476, 5686, 1069, 7942, 7032, 4749, 4275, 1767, 8024, 2193, 4028,
                 5543, 5314,  702, 1394, 6844, 4638, 2384, 2094,  679, 8043,  102]),
         tensor([ 101, 3125, 2151, 1963,  862,  934, 3152, 4289, 8043, 3152, 4289, 1278,
                 7368,  678, 3299, 1403, 1062,  830, 2458, 3123,  102]),
         tensor([ 101, 3918, 1766, 2791,  817, 3221, 3755, 7345,  127,  945, 2218, 3221,
                 1728,  711, 5307, 3845, 8043, 7231, 8013,  102]),
         tensor([ 101,  679, 6566, 3217, 1045, 8024, 3569, 5709, 3409,  678, 8039, 3946,
                 3265,  872, 2769, 8024, 3946, 3265, 7987, 2272,  102]),
         tensor([ 101,  753, 5529, 8024, 1963,  862, 2190, 8043,  102]),
         tensor([ 101, 6768, 3351,  671, 1174, 8038, 2372,  872, 4692, 1059, 4413, 3297,
                 1691, 3457, 4664, 4328, 8024, 3680, 1921, 6822, 1126, 4636,  782, 8024,
                 2144, 6380, 3198, 2347, 6814, 1126, 2399,  102])]
```

2.2.3. Masking and Label Encoding

An attention mask is used to indicate which tokens are real tokens and which are padding tokens. The model is instructed to ignore padding tokens during the attention computation. In the attention mask, 1 represents a real token, and 0 represents a padding token.

The labels corresponding to each sentence are converted into integer values. Since this is a multi-class classification problem, each category (e.g., politics, sports, technology) is mapped to an integer label ranging from 0 to 14.

To optimize the preprocessing pipeline, the processed data is cached for faster future access. When data is preprocessed, it is stored in cache files, preventing the need to reprocess the raw data repeatedly. This is especially useful when working with large datasets.

3. Model

In this experiment, I use a BERT-based model for the task of sentence classification. The model is built on top of the BERT architecture, The model used here is specifically fine-tuned for the task of classifying Chinese news sentences into 15 predefined categories.

3.1. Embedding Layer

The first component of the BERT model is the embedding layer, which converts input tokens into dense vectors of fixed size (768-dimensional vectors in this case).

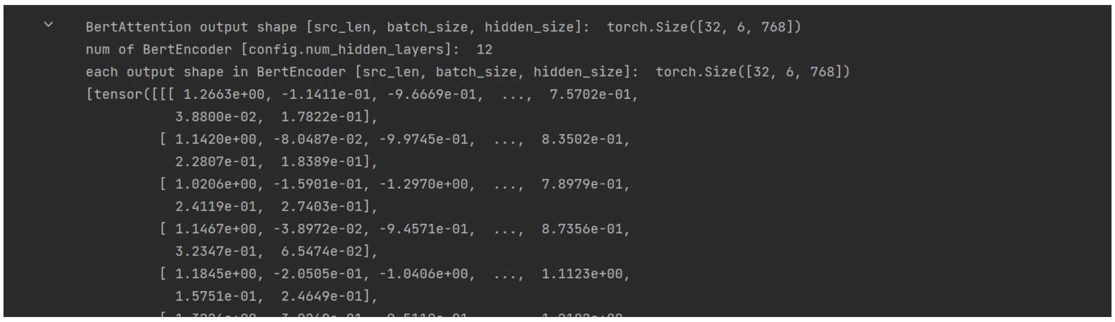
Token Embeddings: The token embeddings are derived from a vocabulary file (e.g., vocab.txt), which maps each unique token to an integer index. This vocabulary is used to convert raw text into token IDs, which are then fed into the model,which is discussed above. The embedding matrix is of shape **[vocab_size, hidden_size]** where **vocab_size** is the number of tokens in the vocabulary,in bert-base-chinese is 21128 and **hidden_size** is the dimensionality of the token embeddings (768 for BERT base).

Positional Embeddings: The positional embeddings are represented as a matrix of shape [max_position_embeddings, hidden_size], where max_position_embeddings is the maximum sequence length that BERT can handle (typically 512 for BERT base). The [CLS] token has position 0, and the positions of the subsequent tokens increase from left to right.

Segment Embeddings: In tasks involving sentence pairs, segment embeddings distinguish between different sentences. For this classification task, all tokens are assigned the same segment ID of 0.

3.2. Encoder and Pooler

The encoder consists of a stack of 12 transformer layers (in the base BERT model) with multi-head self-attention.



```
BertAttention output shape [src_len, batch_size, hidden_size]:  torch.Size([32, 6, 768])
num of BertEncoder [config.num_hidden_layers]:  12
each output shape in BertEncoder [src_len, batch_size, hidden_size]:  torch.Size([32, 6, 768])
[tensor([[[ 1.2663e+00, -1.1411e-01, -9.6669e-01,  ...,  7.5702e-01,
            3.8800e-02,  1.7822e-01],
          [ 1.1420e+00, -8.0487e-02, -9.9745e-01,  ...,  8.3502e-01,
            2.2807e-01,  1.8389e-01],
          [ 1.0206e+00, -1.5901e-01, -1.2970e+00,  ...,  7.8979e-01,
            2.4119e-01,  2.7403e-01],
          [ 1.1467e+00, -3.8972e-02, -9.4571e-01,  ...,  8.7356e-01,
            3.2347e-01,  6.5474e-02],
          [ 1.1845e+00, -2.0505e-01, -1.0406e+00,  ...,  1.1123e+00,
            1.5751e-01,  2.4649e-01],
```

Multi-Head Attention: In each layer, the model computes attention scores that measure how much focus each token should have on other tokens in the sequence. Multi-head attention allows the model to capture different types of relationships in the data.

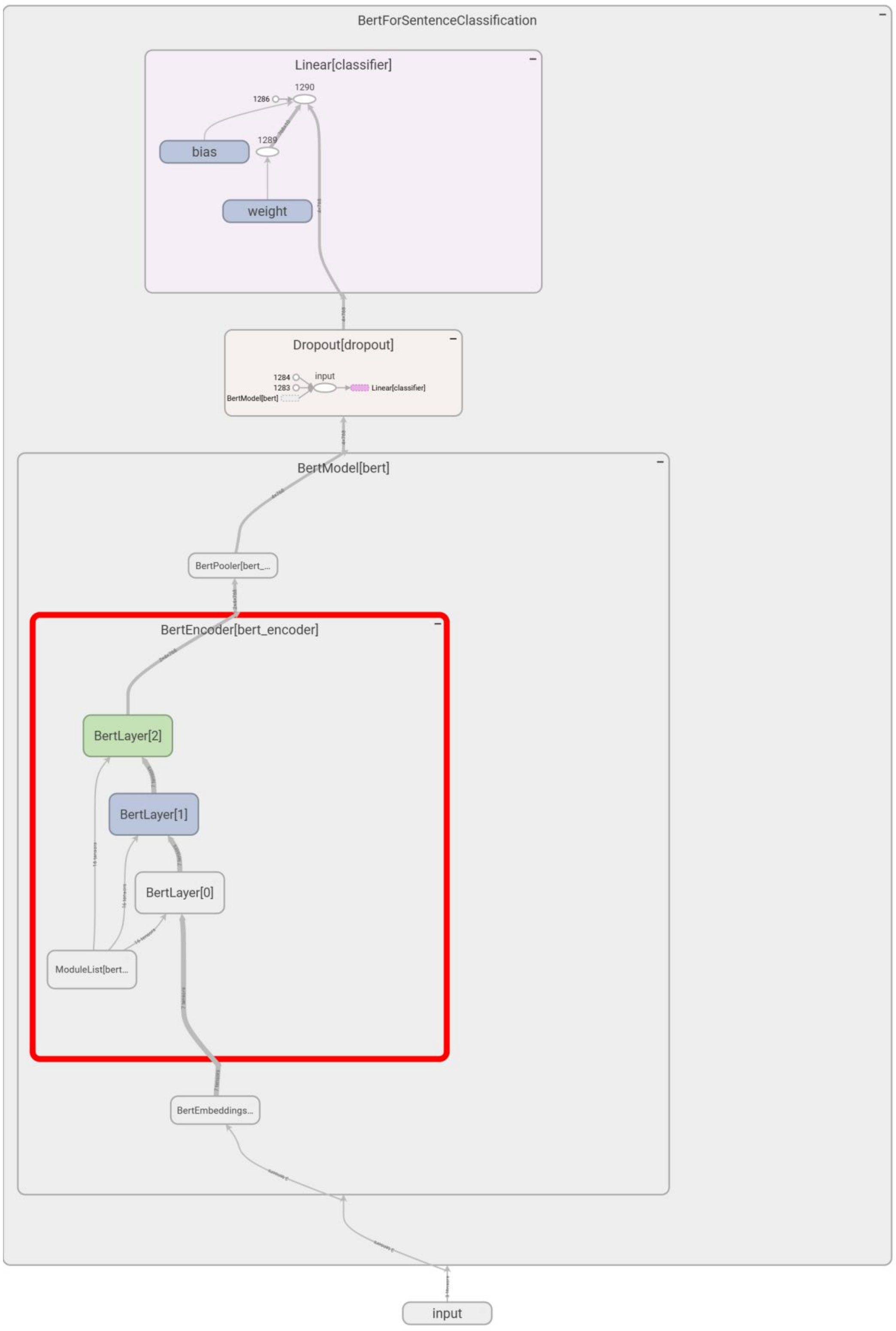
After processing through the transformer layers, BERT uses a pooling layer to extract a single vector that represents the entire input sequence.

For this task, the representation of the [CLS] token (the first token in the input) is used to summarize the entire sentence.

The pooled output from the [CLS] token is passed through a fully connected layer and a Gelu activation to produce the final sentence-level feature vector.

The pooled representation of the input sentence is then passed to a linear classifier. The classifier is a simple fully connected layer that maps the 768-dimensional feature vector (from the [CLS] token) to a vector of size 15, corresponding to the 15 possible categories (labels) for classification.

You can view the architecture via **TensorBoard:**


BertForSentenceClassification
Linear[classifier]
1290
1286
1289
bias
weight
Dropout[dropout]
1284
1283
input
BertModel[bert]
Linear[classifier]
BertModel[bert]
BertPooler[bert_...
BertEncoder[bert_encoder]
BertLayer[2]
BertLayer[1]
BertLayer[0]
ModuleList[bert...
BertEmbeddings...
input

3.3. HowNet Lexicon

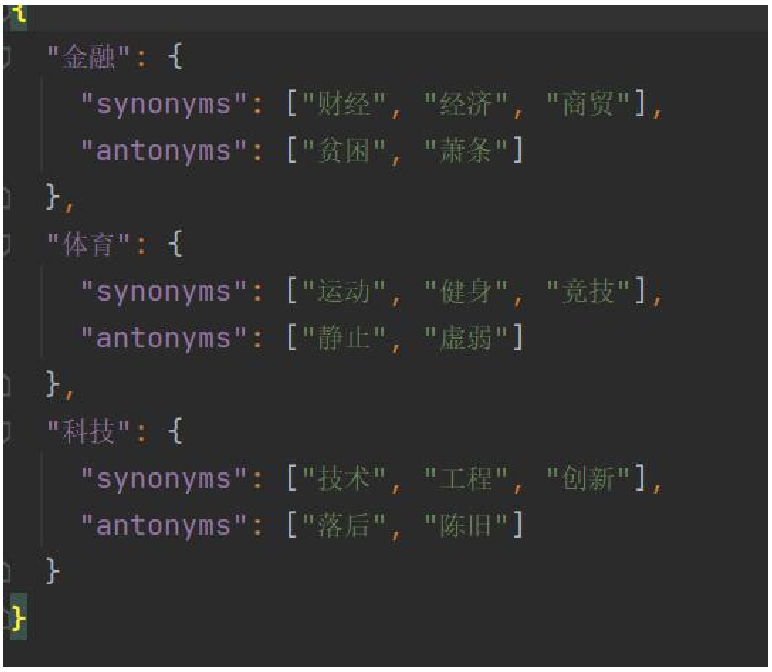
HowNet is a large-scale Chinese lexical database that contains rich semantic information about words, including:Synonyms: Words with similar meanings.Antonyms: Words with opposite meanings.

In this project, HowNet is leveraged to improve the model's understanding of word meanings and relationships during the preprocessing and fine-tuning stages. Specifically, the HowNet lexicon is used for the following tasks:

Semantic Enrichment:
During tokenization, words are broken down into subwords. The subword tokens can be mapped to their corresponding HowNet entries to gather additional semantic information such as synonyms and antonyms. This can help improve the representation of words during training, especially for rare or unseen words.
Example:
For a word like "故宫" (The Forbidden City), the HowNet entry could provide semantic features, helping the model understand that it refers to a specific historical site, enriching its representation in the model's embeddings.

Synonym Expansion:
HowNet can be used to expand the training data by replacing words with their synonyms during data augmentation. By replacing a word with its synonyms, the model is exposed to a broader range of similar words, making it more robust. This can improve generalization by introducing more lexical diversity into the training set.
Example:
The word "文化" (culture) could be replaced with "文明" (civilization) in the training process, while keeping the meaning of the sentence the same.

Handling Ambiguity with Antonyms:
In tasks such as sentiment analysis or topic classification, words might have different meanings based on context. HowNet provides antonyms for words, which can be used to identify contextual polarity or disambiguate between different senses of a word. This is particularly useful for tasks involving sentiment classification, where a word's meaning might flip depending on its context.
Example:
The word "繁荣" (prosperity) has antonyms like "衰退" (decline). By integrating these antonyms during training, the model can better understand context and improve its ability to classify sentiment or topic-related sentences.
Here is example:



```
{
  "金融": {
    "synonyms": ["财经", "经济", "商贸"],
    "antonyms": ["贫困", "萧条"]
  },
  "体育": {
    "synonyms": ["运动", "健身", "竞技"],
    "antonyms": ["静止", "虚弱"]
  },
  "科技": {
    "synonyms": ["技术", "工程", "创新"],
    "antonyms": ["落后", "陈旧"]
  }
}
```

HowNet Loss

The compute_semantic_loss function integrates HowNet’s semantic knowledge (synonyms and antonyms) into the model training process by penalizing the embeddings of synonyms to be close together and antonyms to be far apart. This additional loss term enhances the semantic understanding of the model and improves its ability to generate embeddings that are semantically meaningful.

```
for b in range(seq_len):
        tokens = [vocab.itos[i] for i in input_ids[:, b].tolist()]
        for i, tok in enumerate(tokens):
            syns = lexicon.get_synonyms(tok)
            ants = lexicon.get_antonyms(tok)
            if syns:
                # find first synonym in this sentence
                for j, tok2 in enumerate(tokens):
                    if tok2 in syns:
                        v1, v2 = hs[b,i], hs[b,j]  # [H]
                        loss_syn += F.mse_loss(v1, v2)
                        count_syn += 1
                        break
            if ants:
                for j, tok2 in enumerate(tokens):
                    if tok2 in ants:
                        v1, v2 = hs[b,i], hs[b,j]
                        # margin loss: max(0, margin - ||v1-v2||)
                        dist = F.pairwise_distance(v1.unsqueeze(0), v2.unsqueeze(0))
                        loss_ant += torch.clamp(margin - dist, min=0.0).mean()
                        count_ant += 1
                        break
```

Synonym Loss (MSE):

The MSE loss between embeddings of synonyms ensures that similar words (synonyms) have embeddings that are close in the vector space. This is beneficial because words with similar meanings should be mapped to similar locations in the embedding space.

Antonym Loss (Margin Loss):

The margin loss for antonyms uses a distance-based approach to ensure that antonyms are positioned far apart in the embedding space. The loss function encourages the model to maintain a certain distance (specified by the margin) between antonyms, which helps preserve their opposite meanings.

3.4. Fine tune

In this project, I employ fine-tuning to adapt a pre-trained Chinese BERT base model to classify sentences into one of 15 topic categories.

Instead of training a model from scratch, we reuse:
Word representations: already learned during pre-training(e.g. vocab.txt).
Transformer architecture: with initialized weights capturing syntax and semantics.
[CLS] token embedding: used as the sentence-level representation for classification.

MyBert has a total of 200 parameters, while bert-base-chinese has 207 parameters. It is important to note that the **position_ids** parameter in the MyBert model is not a parameter that needs to be trained; it is just a default initial value. Finally, after analysis (comparing the two one by one), it was found that except for the last 8 parameters in bert-base-chinese, the remaining 199 parameters are the same as those in MyBert model and in the same order.

So, finally, we can initialize the parameters in MyBert with the parameters in bert-base-Chinese by adding another function like the one shown below in the Bert class (**Bert.py** file)

```python
1.  @classmethod
2.   def from_pretrained(cls, config, pretrained_model_dir=None):
3.       model = cls(config)
4.       pretrained_model_path = os.path.join(pretrained_model_dir, "pytorch_model.bin")
5.       loaded_paras = torch.load(pretrained_model_path)
6.       state_dict = deepcopy(model.state_dict())
7.       loaded_paras_names = list(loaded_paras.keys())[:-8]
8.       model_paras_names = list(state_dict.keys())[1:]
9.       for i in range(len(loaded_paras_names)):
10.          state_dict[model_paras_names[i]] = loaded_paras[loaded_paras_names[i]]
11.          logging.info(f"Successfully load {loaded_paras_names[i]} to {model_paras_names[i]}")
12.      model.load_state_dict(state_dict)
13.     return model
```

In the above code, lines 4-5 are used to load the local bert-base-Chinese parameters; Line 6 is used to copy the network parameters in a copy of MyBert, because we can't directly change the values in it; Lines 7-10 assign the parameters from bert-base-Chinese to the state_dict based on our analysis above; Line 12 initializes the parameters in the MyBert with the parameters in the state_dict.

Finally, we can return a BERT model initialized with bert-base-Chinese just call the pretrained.

4. Training and Evaluation

4.1. Training

The training process aims to fine-tune a pre-trained BERT model to classify Chinese news headlines into one of 15 topic categories. In addition to the conventional **cross-entropy loss** used for classification, we integrate a **semantic loss** computed using the HowNet lexicon.

The total loss function used during training combines two components:

$$\mathrm{L_{total}} = \mathrm{L_{CE}} + \lambda \mathrm{L_{SE}}$$

L_{CE} is the cross-entropy loss, which measures the discrepancy, L_{SE} is a semantic loss computed using HowNet. λ is a weighting factor (default: 0.1) that controls the influence of semantic loss on the overall objective.

```python
    for epoch in range(config.epochs):
        losses = 0
        start_time = time.time()
        for idx, (sample, label) in enumerate(train_iter):
            sample = sample.to(config.device)  # [src_len, batch_size]
            label = label.to(config.device)
            padding_mask = (sample == data_loader.PAD_IDX).transpose(0, 1)
            loss, logits = model(
                input_ids=sample,
                attention_mask=padding_mask,
                token_type_ids=None,
                position_ids=None,
                labels=label)
            sem_loss = compute_semantic_loss(
                input_ids=sample,
                hidden_states=hidden,  # Last hidden layer [seq_len, batch_size, hidden_size]
                vocab=data_loader.vocab,
                lexicon=lexicon,
                lamda=λ_sem
            )
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            losses += loss.item()+sem_loss*λ_sem
            acc = (logits.argmax(1) == label).float().mean()
```

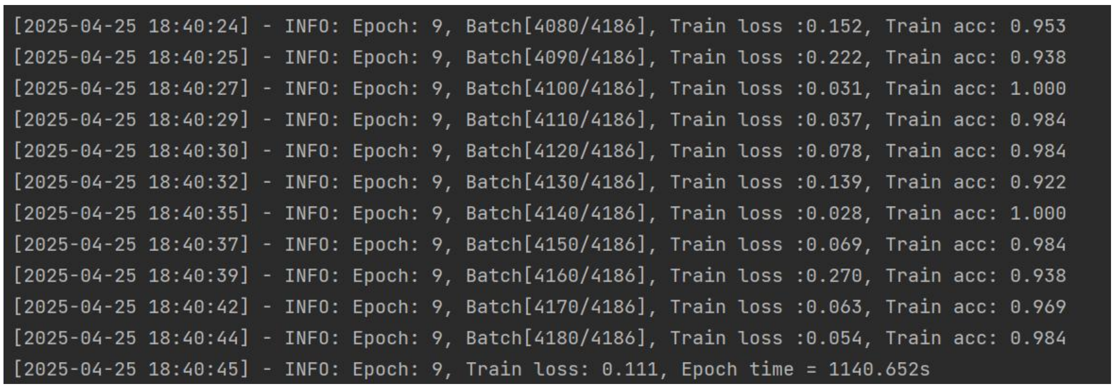
This training loop fine-tunes a pre-trained BERT model for this task, integrating both standard cross-entropy loss and an additional semantic loss derived from HowNet. For each epoch, the model iterates over batches of input samples and labels, moving them to the specified device and computing the attention mask to ignore padding tokens. During the forward pass, the model returns the classification loss, predicted logits, and the final hidden layer output. The hidden states are then used in the compute_semantic_loss function, which leverages HowNet to enforce semantic consistency by bringing synonyms closer and pushing antonyms apart in the embedding space. The total loss, a sum of classification and semantic loss, is backpropagated to update the model parameters.

4.2. Evaluation

```python
def evaluate(data_iter, model, device, PAD_IDX):
    model.eval()
    with torch.no_grad():
        acc_sum, n = 0.0, 0
        for x, y in data_iter:
            x, y = x.to(device), y.to(device)
            padding_mask = (x == PAD_IDX).transpose(0, 1)
            logits = model(x, attention_mask=padding_mask)
            acc_sum += (logits.argmax(1) == y).float().sum().item()
            n += len(y)
        model.train()
        return acc_sum / n
```

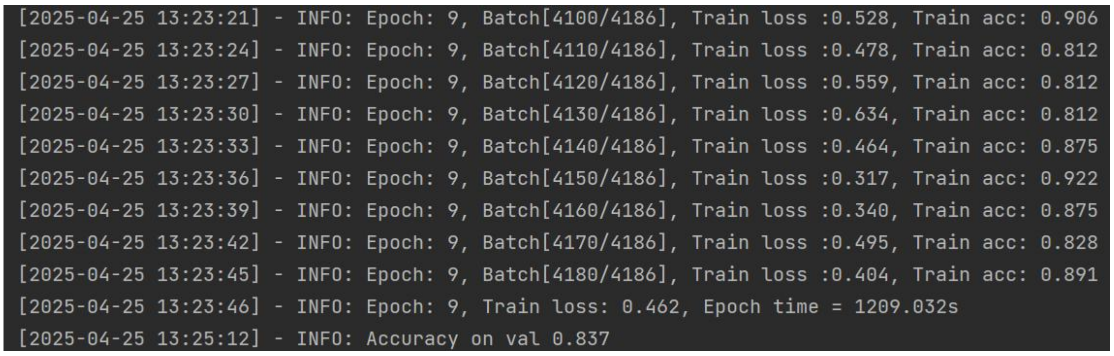
This evaluate function assesses the performance of a trained classification model on a given dataset iterator. It sets the model to evaluation mode using **model.eval()** to deactivate layers like dropout and batch normalization, ensuring consistent behavior during inference. An attention mask is generated by identifying padding tokens using the **PAD_IDX**. The model then outputs logits, and the number of correct predictions is computed by comparing the predicted labels (obtained using **argmax**) with the true labels. .

With pretrained model:



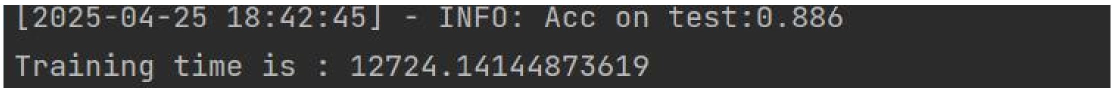
```
[2025-04-25 18:40:24] - INFO: Epoch: 9, Batch[4080/4186], Train loss :0.152, Train acc: 0.953
[2025-04-25 18:40:25] - INFO: Epoch: 9, Batch[4090/4186], Train loss :0.222, Train acc: 0.938
[2025-04-25 18:40:27] - INFO: Epoch: 9, Batch[4100/4186], Train loss :0.031, Train acc: 1.000
[2025-04-25 18:40:29] - INFO: Epoch: 9, Batch[4110/4186], Train loss :0.037, Train acc: 0.984
[2025-04-25 18:40:30] - INFO: Epoch: 9, Batch[4120/4186], Train loss :0.078, Train acc: 0.984
[2025-04-25 18:40:32] - INFO: Epoch: 9, Batch[4130/4186], Train loss :0.139, Train acc: 0.922
[2025-04-25 18:40:35] - INFO: Epoch: 9, Batch[4140/4186], Train loss :0.028, Train acc: 1.000
[2025-04-25 18:40:37] - INFO: Epoch: 9, Batch[4150/4186], Train loss :0.069, Train acc: 0.984
[2025-04-25 18:40:39] - INFO: Epoch: 9, Batch[4160/4186], Train loss :0.270, Train acc: 0.938
[2025-04-25 18:40:42] - INFO: Epoch: 9, Batch[4170/4186], Train loss :0.063, Train acc: 0.969
[2025-04-25 18:40:44] - INFO: Epoch: 9, Batch[4180/4186], Train loss :0.054, Train acc: 0.984
[2025-04-25 18:40:45] - INFO: Epoch: 9, Train loss: 0.111, Epoch time = 1140.652s
```

Without pretrained model



```
[2025-04-25 13:23:21] - INFO: Epoch: 9, Batch[4100/4186], Train loss :0.528, Train acc: 0.906
[2025-04-25 13:23:24] - INFO: Epoch: 9, Batch[4110/4186], Train loss :0.478, Train acc: 0.812
[2025-04-25 13:23:27] - INFO: Epoch: 9, Batch[4120/4186], Train loss :0.559, Train acc: 0.812
[2025-04-25 13:23:30] - INFO: Epoch: 9, Batch[4130/4186], Train loss :0.634, Train acc: 0.812
[2025-04-25 13:23:33] - INFO: Epoch: 9, Batch[4140/4186], Train loss :0.464, Train acc: 0.875
[2025-04-25 13:23:36] - INFO: Epoch: 9, Batch[4150/4186], Train loss :0.317, Train acc: 0.922
[2025-04-25 13:23:39] - INFO: Epoch: 9, Batch[4160/4186], Train loss :0.340, Train acc: 0.875
[2025-04-25 13:23:42] - INFO: Epoch: 9, Batch[4170/4186], Train loss :0.495, Train acc: 0.828
[2025-04-25 13:23:45] - INFO: Epoch: 9, Batch[4180/4186], Train loss :0.404, Train acc: 0.891
[2025-04-25 13:23:46] - INFO: Epoch: 9, Train loss: 0.462, Epoch time = 1209.032s
[2025-04-25 13:25:12] - INFO: Accuracy on val 0.837
```

Training with pretrained model reaches lower loss and higher accuracy after 10 epochs. We can see the benefits of using a pretrained model.BERT-base-chinese have already learned rich lexical, syntactic, and semantic representations from massive corpora.
Because the weights are already trained, fine-tuning typically requires only tens of epochs to converge, whereas training from scratch may take hundreds of epochs to reach similar performance.
Finally,the train loss is



```
[2025-04-25 18:42:45] - INFO: Acc on test:0.886
Training time is : 12724.14144873619
```

5. Conclusion

This project implemented a BERT-based sentence classification model trained entirely from scratch without relying on pre-trained weights. To enhance semantic understanding, I integrated a HowNet-based semantic loss alongside the standard cross-entropy loss. The model achieved strong performance on a Chinese news classification task, demonstrating that even without pre-training, meaningful results can be obtained through careful design and the use of external lexical knowledge. Future work may extend this approach to other NLP tasks and languages.

You can preview detailed experiment steps in **report.ipynb**,and run the demo by using **python demo.py**.