NLP_HW3_111062313 清大資工 陳中和

Q1: Which (pre-trained) model do you use? Why to choose the model?

I tried to train five BERT-base related models, and their Spearman correlation, accuracy, F1 score, and loss are shown in the table below. We can find that the results trained using bert-base-uncased and bert-base-cased models performed the best and had greater stability. In the final validation in epoch3 and test stages, both models achieved relatively high accuracy, approximately 0.87, and also had high Spearman correlation.

Among these two models, I finally chose bert-base-uncased as my BERT model for this task. Since the task of this assignment is mainly to understand the semantic relationship between two sentences, and I believe that whether the word is uppercase or lowercase does not affect the meaning of a sentence. Considering uppercase or lowercase letters might affect the stability of the training process.

The following results were obtained using the approach described in Method 2 from Question 4. In the forward, the pooler_output was passed through a linear layer, followed by a ReLU activation layer, and then through another linear layer to produce the final output.

google-bert/bert-base-uncased				
	Epoch 1 (validate)	Epoch 2 (validate)	Epoch 3 (validate)	Test
Spearman Corr	0.8069	0.8329	0.8339	0.8420
Accuracy	0.8539	0.8720	0.8740	0.8756
F1 Score	0.8529	0.8740	0.8708	0.8694
Relatedness Loss	0.2779	0.2235	0.2341	0.2214
Entailment Loss	0.3839	0.3082	0.3545	0.3562
google-bert/bert-base-cased				
	Epoch 1 (validate)	Epoch 2 (validate)	Epoch 3 (validate)	Test
Spearman Corr	0.8002	0.8063	0.8354	0.8234
Accuracy	0.8520	0.8480	0.8720	0.8831
F1 Score	0.8482	0.8490	0.8694	0.8740
Relatedness Loss	0.2573	0.2775	0.2436	0.2423
Entailment Loss	0.4020	0.4484	0.3467	0.3534
google-bert/bert-base-cased-finetuned-mrpc				
	Epoch 1 (validate)	Epoch 2 (validate)	Epoch 3 (validate)	Test
Spearman Corr	0.7626	0.7812	0.8059	0.7928
Accuracy	0.8080	0.8240	0.8480	0.8677
F1 Score	0.8061	0.8198	0.8424	0.8576
Relatedness Loss	0.3521	0.3122	0.2792	0.2802
Entailment Loss	0.4995	0.4247	0.4010	0.3743
google-bert/bert-base-chinese				
	Epoch 1 (validate)	Epoch 2 (validate)	Epoch 3 (validate)	Test

Spearman Corr	0.6789	0.7511	0.7741	0.7675
Accuracy	0.6780	0.8100	0.8260	0.8348
F1 Score	0.5332	0.8088	0.8201	0.8251
Relatedness Loss	0.5359	0.4178	0.3489	0.3304
Entailment Loss	0.6925	0.5100	0.4618	0.4469
google-bert/bert-base-multilingual-cased				
	Epoch 1 (validate)	Epoch 2 (validate)	Epoch 3 (validate)	Test
Spearman Corr	0.7871	0.7891	0.8030	0.8122
Accuracy	0.8140	0.8320	0.8560	0.8640
F1 Score	0.7996	0.8187	0.8502	0.8545
Relatedness Loss	0.3926	0.2925	0.2882	0.2699
Entailment Loss	0.5137	0.4523	0.4116	0.3885

Q2: Compared with models trained separately on each of the sub-task, does multi-output learning improve the performance?

When training two sub-tasks together means adding the losses of the two tasks together and then doing back propagation together. Training the two subtasks separately means separately doing back propagation on the losses of the two sub-tasks.

The table below compares the results of joint training and separate training. We observe that whether on the validation set, the model trained together performs better, while the model trained separately starts with lower performance, its improvement rate is significant, eventually enhancing the overall performance to a certain extent. However, overall the jointly trained model achiebes better outcomes.

I speculate that the reason for this difference is that during joint training, the model considers the output of both sub-tasks simultaneously and adjusts its parameters based on both results. In contrast, when training the tasks separately, the model only receives information related to a single sub-task, leading to isolated adjustments for each sub-task, and this separation slows the improvement rate since the model lacks the shared context.

Training two models together					
	Epoch 1 (Val)	Epoch 2 (Val)	Epoch 3 (Val)	Epoch 4 (Val)	Epoch 5 (Val)
Spearman Corr	0.8195	0.8485	0.8394	0.8446	0.8305
Accuracy	0.8500	0.8860	0.8680	0.8660	0.8880
F1 Score	0.8460	0.8798	0.8647	0.8637	0.8808
Relatedness Loss	0.2801	0.2643	0.2578	0.2310	0.2413
Entailment Loss	0.4115	0.3139	0.3968	0.3962	0.3710
Training two models separately					
	Epoch 1 (Val)	Epoch 2 (Val)	Epoch 3 (Val)	Epoch 4 (Val)	Epoch 5 (Val)
Spearman Corr	0.5432	0.7007	0.7329	0.7530	0.7402

Accuracy	0.7720	0.8000	0.7960	0.8020	0.7940
F1 Score	0.7573	0.7897	0.7956	0.7793	0.7965
Relatedness Loss	0.6246	0.5170	0.4288	0.4115	0.5030
Entailment Loss	0.5683	0.5121	0.4944	0.5800	0.5651
Training two models together					
	Epoch 1 (Test)	Epoch 2 (Test)	Epoch 3 (Test)	Epoch 4 (Test)	Epoch 5 (Test)
Spearman Corr	0.7988	0.8222	0.8288	0.8280	0.8343
Accuracy	0.8411	0.8754	0.8809	0.8811	0.8803
F1 Score	0.8345	0.8652	0.8724	0.8727	0.8717
Relatedness Loss	0.2925	0.2653	0.2579	0.2290	0.2552
Entailment Loss	0.4479	0.3495	0.3663	0.3847	0.3750
Training two models separately					
	Epoch 1 (Test)	Epoch 2 (Test)	Epoch 3 (Test)	Epoch 4 (Test)	Epoch 5 (Test)
Spearman Corr	0.5557	0.7009	0.7318	0.7468	0.7434
Accuracy	0.7753	0.8141	0.8161	0.8112	0.8217
F1 Score	0.7588	0.8011	0.8092	0.7884	0.8162
Relatedness Loss	0.6212	0.4825	0.4050	0.3894	0.4692
Entailment Loss	0.5629	0.4964	0.4743	0.5630	0.5411

Q3: Why does your model fail to correctly predict some data points? Please provide an error analysis.

For the misclassified examples, I will discuss the two sub-tasks separately.

Sub-task 1: Relatedness Score

Here, I list some examples where the difference between the correct answer and the predicted score by my model exceeds 1.00:

	premise	hypothesis	relatedness_score	error
e.g. 1	A person in a black jacket is doing tricks on a motorbike	A person on a black motorbike is doing tricks with a jacket	Predicted: 2.30 True: 1.30	1.00
e.g. 2	A lone biker is jumping in the air	A man is jumping into an empty swimming pool	Predicted: 2.70 True: 1.60	1.10
e.g. 3	A dog is chasing another and is holding a stick in its mouth	A dog is chasing a stick and holding another dog in its mouth	Predicted: 4.34 True: 3.20	1.14
e.g. 4	Two men are holding fishing poles	Two men are holding some water standing near fishing poles	Predicted: 3.68 True: 2.60	1.08
e.g. 5	Two people wearing snowsuits are on the ground making snow angels	Two people in snowsuits are lying in the snow and making snow angels	Predicted: 3.58 True: 4.80	-1.22
e.g. 6	Two children are lying in the snow and are making snow	Two people wearing snowsuits are on the ground making snow	Predicted: 3.34 True: 4.60	-1.26

	angels	angels		
e.g. 7	Two children are hanging on a large branch	Two children are climbing a tree	Predicted: 2.57 True: 4.20	-1.63
e.g. 8	A man is performing a trick on a green bicycle	There is no man performing a trick on a green bicycle	Predicted: 3.34 True: 4.80	-1.46
e.g. 9	Two cats are playing with a red ball	Two brown dogs are fighting over a red ball and are playing in the grass	Predicted: 1.58 True: 2.95	-1.37

For examples 1 to 4, the model overestimated the true relatedness score. Based on the highlighted words, we observe that when the subject and object positions are swapped, the model mistakenly interprets the sentences as having the same meaning, even though their meanings are entirely opposite. Additionally, the model appears to ignore the objects following the verbs. For instance, in examples 2 and 4, the model assumes that the sentences "two men holding something" are similar, but in reality, the items they are holding are different.

For examples 5 to 9, the model underestimated the true relatedness score. Based on the highlighted words, we can observe that the model has not fully learned to recognize synonyms, such as the relationship between *"hanging on a large branch"* and *"climbing a tree"*. Additionally, sentences with the same meaning but different subjects also lead to lower predicted scores. For instance, in examples 8 and 9. Furthermore, the addition of irrelevant adjective modifiers can also confuse the model. For example, *"fighting over a red ball"* implies they are playing with the red ball, but the model fails to grasp this connection. Similarly, in example 6, the hypothesis includes the modifier *"wearing snowsuits"*, which causes the model to misjudge the relatedness.

Sub-task 2: entailment_judgement

	Premise	Hypothesis	Predicted	True
e.g. 1	A person in a black jacket is doing tricks on a motorbike	A person on a black motorbike is doing tricks with a jacket	Neutral	Entailment
e.g. 2	The player is missing the basket and a crowd is in background	The player is dunking the basketball into the net and a crowd is in background	Entailment	Contradiction
e.g. 3	There is no man dunking the ball at a basketball game	The player is dunking the basketball into the net and a crowd is in background	Entailment	Contradiction
e.g. 4	Two spectators are kickboxing and some people are watching	Two people are kickboxing and spectators are watching	Neutral	Entailment
e.g. 5	Children in red shirts are playing in the leaves	Children covered by leaves are playing with red shirts	Neutral	Entailment

In the second sub-task, one key issue is the model's inability to distinguish between sentences with completely opposite meanings—one affirmative and the other negative. For instance, in Example 2, *"missing*

the basket" and *"dunking the basketball into the net"*, or in Example 3, *"no man"* and *"The player"*. This error also appears in sub-task 1: Relatedness Score.

Additionally, the model lacks the ability to recognize sentences that express the same meaning in different ways. The model tends to classify such cases as Neutral. For example, in Example 1, while the details differ, the core meaning is the same: a person juggling on a bicycle. Similarly, in Example 4, the main meaning is that two people are practicing kickboxing while others are watching. The inability to recognize paraphrased sentences limits the model's performance on this sub-task.

Q4: How do you improve your model performance?

In the MultiLabelModel, I implement three different approaches to handle BERT's pooler_output.

- Method 1: Directly passed through a single linear layer to produce the output.
- Method 2: Passed through a linear layer, followed by a ReLU activation function layer, and then another linear layer before producing the final output.
- Method 3: For the entailment subtask, the implementation is the same as Method 2. For the relatedness subtask, the implementation is as follows:

Method 2	Method 3

`torch.nn.Linear(768, 256),`
`torch.nn.ReLU(),`
`torch.nn.Linear(256, 1)`
	`torch.nn.Linear(768, 256),`
	`torch.nn.ReLU(),`
	`torch.nn.Dropout(0.5),`
	`torch.nn.Linear(256, 1),`
	`torch.nn.Sigmoid()`

The final output is scaled from [0,1] to [1,5] by multiplying the sigmoid function output by 4 and then adding 1. The training results are shown in the table below. We can observe that although Method 3 uses more layers for processing, its performance is not as stable as Method 2. Therefore, I conclude that the Method 2 is the optimal approach for processing the pooler_output.

	Epoch 3 (validate)			Test		
	Method 1	Method 2	Method 3	Method 1	Method 2	Method 3
Spearman Corr	0.8269	0.8339	0.8426	0.8043	0.8420	0.8043
Accuracy	0.8620	0.8740	0.8620	0.8671	0.8756	0.8624
F1 Score	0.8583	0.8708	0.8508	0.8583	0.8694	0.8503
Relatedness Loss	0.2507	0.2341	0.2348	0.2618	0.2214	0.2692
Entailment Loss	0.3598	0.3545	0.4722	0.3738	0.3562	0.4189

Next, I adjusted the number of epochs to 10 to observe how the number of training iterations affects the model's performance. The results are shown in the table below.

	Epoch 1 (Val)	Epoch 2 (Val)	Epoch 3 (Val)	Epoch 4 (Val)	Epoch 5 (Val)
Spearman Corr	0.8341	0.8431	0.8286	0.8266	0.8432
Accuracy	0.8600	0.8780	0.8800	0.8820	0.8860
F1 Score	0.8531	0.8723	0.8724	0.8705	0.8737
	Epoch 6 (Val)	Epoch 7 (Val)	Epoch 8 (Val)	Epoch 9 (Val)	Epoch 10 (Val)
Spearman Corr	0.8437	0.8477	0.8402	0.8331	0.8436
Accuracy	0.8840	0.8780	0.8780	0.8759	0.8580
F1 Score	0.8784	0.8680	0.8714	0.8710	0.8441

	Epoch 1 (Test)	Epoch 2 (Test)	Epoch 3 (Test)	Epoch 4 (Test)	Epoch 5 (Test)
Spearman Corr	0.8077	0.8311	0.8297	0.8300	0.8372
Accuracy	0.8577	0.8717	0.8776	0.8841	0.8805
F1 Score	0.8464	0.8631	0.8688	0.8763	0.8709
	Epoch 6 (Test)	Epoch 7 (Test)	Epoch 8 (Test)	Epoch 9 (Test)	Epoch 10 (Test)
Spearman Corr	0.8358	0.8386	0.8301	0.8381	0.8312
Accuracy	0.8859	0.8784	0.8788	0.8782	0.8776
F1 Score	0.8757	0.8689	0.8659	0.8679	0.8650

From the table above, we can observe that the model achieves its best performance on the validation and test datasets around the fifth or sixth epoch. Increasing the number of training epochs beyond five or six does not improve the model's performance further.

Q5: Why can a model fine-tuned in bert-base-chinese understand English sentences?

When using bert-base-chinese as my pre-trained model, I observed an interesting phenomenon. Although the performance in the first epoch was relatively poor, after training for three epochs, the validation set accuracy reached 0.82, and the test set accuracy reached 0.83. This is a remarkably high number. In Question 1, detailed training data is provided. Upon examining the vocabulary of bert-base-chinese, I found that most of the tokens were indeed Chinese, but there were also a few English words. This means that after training, the model can use these very small numbers of English token along with the semantic context of Chinese sentences to effectively map the meaning of Chinese sentences to their English equivalients. In the paper *"Zero-shot Reading Comprehension by Cross-lingual Transfer Learning with Multi-lingual Language Representation Model"*, it is also mentioned that multi-BERT exhibits the ability to understand and learn from unseen languages.

Running environment: Colab

Python version: Colab

GPU(s) you used: T4 GPU (Colab)

References:

1. https://leemeng.tw/attack_on_bert_transfer_learning_in_nlp.html
2. https://arxiv.org/pdf/1909.09587 (*Zero-shot Reading Comprehension by Cross-lingual Transfer Learning with Multi-lingual Language Representation Model*)
3. https://youtu.be/gh0hewYkjgo?si=egZ_D20eID2AKRnc
4. https://blog.csdn.net/ningyanggege/article/details/132206331