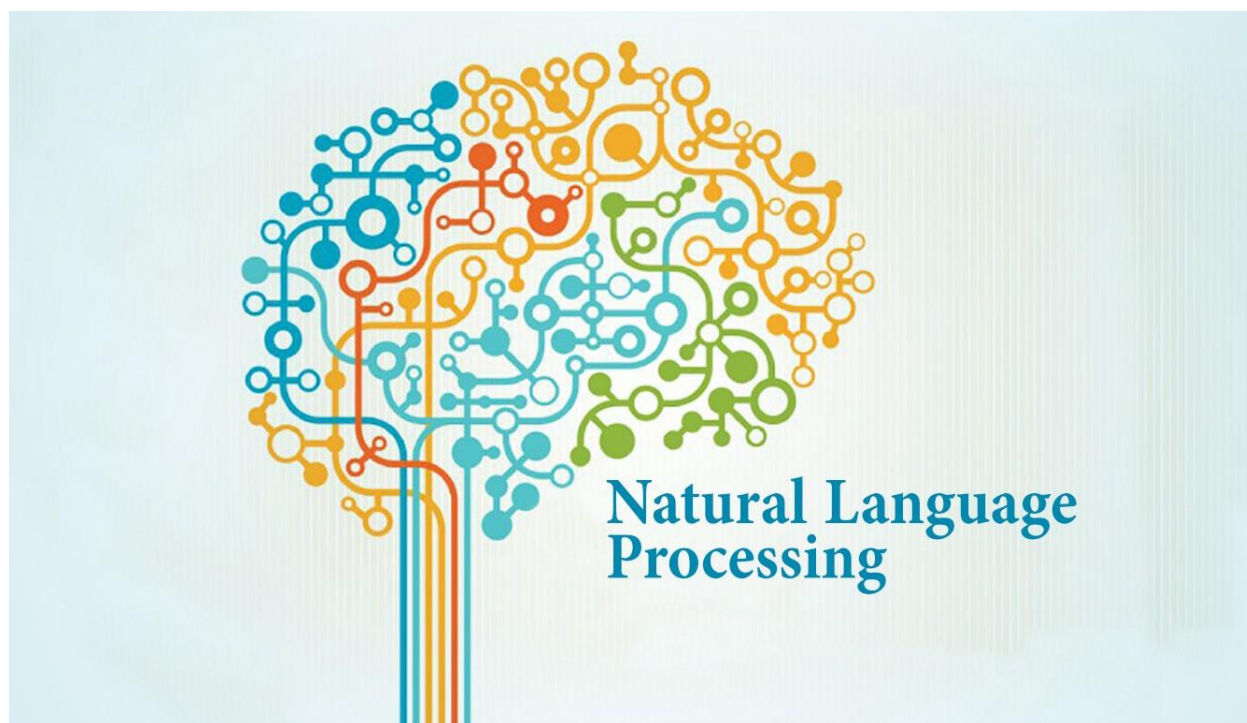


گزارش پروژه پایانی پردازش زبان های طبیعی



نگین درخشان ، کیوان بوشهری

راهنما : رضا قهرمانی

استاد درس : دکتر مینایی

بخش اول : تولید جملات

در این بخش برای تولید جملات از مدل bert استفاده کرده ایم. و برای هر دسته از دیتایمان مدل bert را finetune کردیم.تنظیم دقیق یک تکنیک رایج برای انتقال یادگیری است. مدل هدف همه طرح های مدل را با پارامترهایشان از مدل منبع به جز لایه خروجی کپی می کند و این پارامترها را بر اساس مجموعه داده هدف تنظیم می کند. در مقابل، لایه خروجی مدل هدف باید از ابتدا آموزش داده شود.

```
!python3 pregenerate_training_data.py --train_corpus data_lm_tech.txt --bert_model bert-base-uncased --do_lower_case --o
100% 231508/231508 [00:00<00:00, 26628683.13B/s]
Loading Dataset: 1998 lines [00:00, 4445.77 lines/s]
Epoch: 0% 0/1 [00:00<?, 2it/s]
Document: 100% 999/999 [00:00<00:00, 10020.09it/s]
Epoch: 100% 1/1 [00:00<00:00, 9.96it/s]
```

```
[ ] !python3 finetune_on_pregenerated.py --pregenerated_data training/ --bert_model bert-base-uncased --do_lower_case --trai
2022-07-07 06:58:35,297: device: cuda n_gpu: 1, distributed training: False, 16-bits training: False
2022-07-07 06:58:35,342: loading file https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased-vocab.txt fro
2022-07-07 06:58:35,423: https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased-config.json not found in c
100% 433/433 [00:00<00:00, 635233.87B/s]
2022-07-07 06:58:35,457: copying /tmp/tmpetrpbii1 to cache at /root/.cache/torch/pytorch_transformers/4dad0251492946e18ac
2022-07-07 06:58:35,457: creating metadata file for /root/.cache/torch/pytorch_transformers/4dad0251492946e18ac39290fcfe9
2022-07-07 06:58:35,457: removing temp file /tmp/tmpetrpbii1
2022-07-07 06:58:35,457: loading configuration file https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased
2022-07-07 06:58:35,457: Model config {
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "finetuning_task": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "num_labels": 2,
  "output_attentions": false,
  "output_hidden_states": false,
  "pad_token_id": 0,
  "pruned_heads": {},
  "torchscript": false,
  "type_vocab_size": 2,
  "vocab_size": 30522
}
```

```

2022-07-07 06:58:35,494: https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased-pytorch\_model.bin not found
100% 440473133/440473133 [00:06<00:00, 71357277.13B/s]
2022-07-07 06:58:41,739: copying /tmp/tmp2fvfdu89 to cache at /root/.cache/torch/pytorch_transformers/aaleflaede4482d0dbc
2022-07-07 06:58:43,125: creating metadata file for /root/.cache/torch/pytorch_transformers/aaleflaede4482d0dbcd4d52baad8
2022-07-07 06:58:43,125: removing temp file /tmp/tmp2fvfdu89
2022-07-07 06:58:43,176: loading weights file https://s3.amazonaws.com/models.huggingface.co/bert/bert-base-uncased-pytorch\_model.bin
2022-07-07 06:58:57,012: ***** Running training *****
2022-07-07 06:58:57,012:     Num examples = 999
2022-07-07 06:58:57,012:     Batch size = 16
2022-07-07 06:58:57,012:     Num steps = 62
finetune_on_pregenerated.py:88: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence th
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  input_masks = np.zeros(shape=(num_samples, seq_len), dtype=np.bool)
finetune_on_pregenerated.py:89: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence th
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  segment_ids = np.zeros(shape=(num_samples, seq_len), dtype=np.bool)
finetune_on_pregenerated.py:91: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence th
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  is_nexts = np.zeros(shape=(num_samples,), dtype=np.bool)
2022-07-07 06:58:57,014: Loading training examples for epoch 0
Training examples:   0% 0/999 [00:00<?, ?it/s]finetune_on_pregenerated.py:38: DeprecationWarning: `np.int` is a deprecate
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  input_array = np.zeros(max_seq_length, dtype=np.int)
finetune_on_pregenerated.py:41: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence th
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  mask_array = np.zeros(max_seq_length, dtype=np.bool)
finetune_on_pregenerated.py:44: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence th
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  segment_array = np.zeros(max_seq_length, dtype=np.bool)
finetune_on_pregenerated.py:47: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  . . . . .







```

```

Training examples: 100% 999/999 [00:00<00:00, 11731.28it/s]
2022-07-07 06:58:57,099: Loading complete!
Epoch 0:   2% 1/63 [00:00<00:29, 2.07it/s, Loss: 17.25792]/usr/local/lib/python3.7/dist-packages/pytorch_transformers/op
  add_(Number alpha, Tensor other)
Consider using one of the following signatures instead:
  add_(Tensor other, *, Number alpha) (Triggered internally at  ../torch/csrc/utils/python_arg_parser.cpp:1055.)
  exp_avg.mul_(beta1).add_(1.0 - beta1, grad)
Epoch 0: 100% 63/63 [00:25<00:00, 2.43it/s, Loss: 5.19670]
2022-07-07 06:59:23,034: ** * * Saving fine-tuned model * * *

```

نتیجه نهایی برای مدل تکنولوژی در فولدر finetuned_lm:

 added_tokens.json	Today at 11:30 AM	2 bytes	JSON
 config.json	Today at 11:30 AM	588 bytes	JSON
 pytorch_model.bin	Today at 11:37 AM	440.5 MB	MacBin...archive
 special_tokens_map.json	Today at 11:30 AM	112 bytes	JSON
 tokenizer_config.json	Today at 11:31 AM	58 bytes	JSON
 vocab.txt	Today at 11:31 AM	232 KB	Plain Text

حال برای تولید چند جمله در زمینه تکنولوژی از مدلمان استفاده می کنیم:

Writing with **BERT**

Seed text:

Technology is

6

☐ Open World ☒ Technology

Random Hop

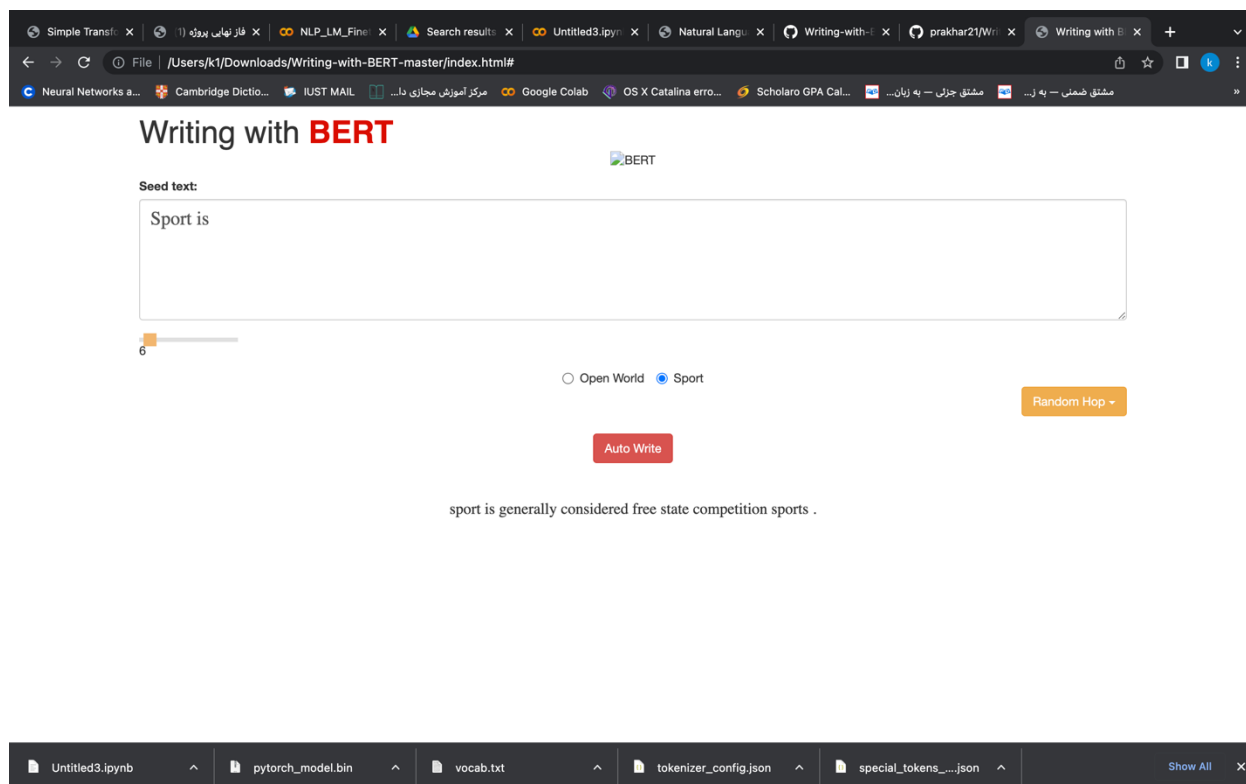
Auto Write

technology is sometimes considered often extremely expensive technology .

pytorch_model.bin vocab.txt tokenizer_config.json special_tokens_...json config.json Show All X

مشاهده می شود که جمله تولید شده مربوط به زمینه علم و فن آوری است که نشان دهنده یادگیری موفقیت آمیز مدل ما است.

همین کار را برای دسته ورزشی تکرار کردیم:



نکته مهم: زمانی که مدل bert را finetuned می کردیم زمان زیادی برای آموزش لازم بود به این جهت از gpu کلب استفاده کردیم.

بخش دوم: شبکه با معماری ساده

در این قسمت ابتدا جملات تولید شده در قسمت های قبلی را که در دو دسته train , test هستند را خوانده و آرایه های xtrain, ytrain, xtest, ytest را تشکیل دادیم. در قدم بعدی با استفاده از tokenizer جملات آرایه x را token کرده و تبدیل به دنباله (sequences) کردیم و دنباله را به آرایه ورودی شبکه عصبی تبدیل کردیم.

پس از تولید ورودی مناسب، شبکه عصبی را ساختیم. ابتدا مدل را به صورت Sequential تعریف کرده و یک شبکه lstm به آن اضافه کردیم. در ادامه مدل را کامپایل کرده و روی داده های train آموزش دادیم.

```

model=Sequential()
model.add(layers.Embedding(input_dim=V_size,output_dim=1, input_length=length))
model.add(layers.LSTM(units=1))
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=['accuracy'])
model.summary()

```

در ادامه با استفاده از predict و داده تست، داده پیشبینی شده را بدست آوردیم. این داده مقدار بین ۰ و ۱ دارد که با توجه به ۰ یا ۱ بودن کلاس ها با تابع sigmoid کلاس پیش بینی شده را بدست آوردیم. در نهایت با استفاده از مقدار کلاس واقعی مقدار دقت را بدست آوردیم.

```

ypred[ypred>0.5]=1
ypred[ypred<=0.5]=0
print("Accuracy:",accuracy_score(y_test, ypred)*100)

```

شبکه ساخته شده و مقدار دقت این شبکه روی داده تست به صورت زیر است:

Model: "sequential_21"

Layer (type)	Output Shape	Param #
=====		
embedding_17 (Embedding)	(None, 5, 1)	6936
lstm_20 (LSTM)	(None, 1)	12
=====		

Total params: 6,948

Trainable params: 6,948

Non-trainable params: 0

Accuracy: 92.44791666666666

بخش سوم: شبکه با معماری پیچیده تر

در این قسمت با استفاده از مدل roberta که pretrained می باشد و بر پایه transformers است مدل را با دیتاست خودمان آموزش دادیم:

RoBERTa - Hugging Face

برای این کار از library به اسم simpletransformers استفاده کردیم:

Simple Transformers

```
[ ] !pip install simpletransformers
from simpletransformers.classification import ClassificationModel
model = ClassificationModel('roberta', 'roberta-base', num_labels=2, args={'learning_rate':1e-5, 'num_train_epochs': 1, 'reprocess_input_data': True,
```

در مرحله بعدی دیتا فریم های آموزش و آزمون که به نسبت ۲۰/۸۰ split کرده بودیم را به کد معرفی می کنیم:

```
import pandas as pd

train_df = pd.read_csv('train.csv', header=None)

eval_df = pd.read_csv('test.csv', header=None)
```

در مرحله بعدی مدل را آموزش می دهیم:

```
model.train_model(train_df)
```

سپس از مدل خود ارزیابی به عمل می آوریم:

```
model.train_model(train_df)

from sklearn.metrics import f1_score, accuracy_score

def f1_multiclass(labels, preds):
    return f1_score(labels, preds, average='micro')

result, model_outputs, wrong_predictions = model.eval_model(eval_df, f1=f1_multiclass, acc=accuracy_score)

print(result)
```

```
{'mcc': 0.989655681759032, 'tp': 186, 'tn': 197, 'fp': 2, 'fn': 0, 'auroc': 1.0, 'auprc': 1.0, 'f1': 0.9948051948051949, 'acc': 0.9948051948051948, 'eval_loss': 0.01752868355369476}
```

همانطور که انتظار می رفت عملکرد مدل بر پایه transformers به طور قابل توجهی از مدل بر پایه lstm بهتر است و نتیجه بهتری دارد که در ادامه این دو ساختار را مقایسه و نتیجه گیری می کنیم.

نتیجه گیری:

همانطور که بحث شد، ترانسفورماتورها سریعتر از مدل های مبتنی بر RNN هستند، زیرا تمام ورودی یک بار وارد می شود. آموزش LSTM در مقایسه با شبکه های ترانسفورماتور سخت تر است، زیرا تعداد پارامترها در شبکه های LSTM بسیار بیشتر است. علاوه بر این، انجام یادگیری انتقالی در شبکه های LSTM غیرممکن است. ترانسفورماتورها در حال حاضر شبکه ای پیشرفته برای مدل های seq2seq هستند. از این رو، ما با این واقعیت نتیجه می گیریم که شبکه های ترانسفورماتور بهترین دقت را ارائه می دهند و همچنین با پیچیدگی و هزینه محاسباتی کمتری همراه هستند.

منابع :

- [Why are LSTMs struggling to matchup with Transformers?](#)
- [Natural Language Generation using BERT](#)
- [Simple Transformers — Multi-Class Text Classification with BERT, RoBERTa, XLNet, XLM, and DistilBERT](#)