Intent Recognition with BERT using Keras and TensorFlow 2

TL;DR Learn how to fine-tune the BERT model for text classification. Train and evaluate it on a small dataset for detecting seven intents. The results might surprise you!

comments

By Venelin Valkov, Machine Learning Engineer

Recognizing intent (IR) from text is very useful these days. Usually, you get a short text (sentence or two) and have to classify it into one (or multiple) categories.

Multiple product support systems (help centers) use IR to reduce the need for a large number of employees that copy-and-paste boring responses to frequently asked questions. Chatbots, automated email responders, answer recommenders (from a knowledge base with questions and answers) strive to not let you take the time of a real person.

This guide will show you how to use a pre-trained NLP model that might solve the (technical) support problem that many business owners have. I mean, BERT is freaky good! It is really easy to use, too!

Run the complete notebook in your browser

The complete project on GitHub

Data

The data contains various user queries categorized into seven intents. It is hosted on GitHub and is first presented in this paper.

Here are the intents:

- SearchCreativeWork (e.g. Find me the I, Robot television show)
- GetWeather (e.g. Is it windy in Boston, MA right now?)
- BookRestaurant (e.g. I want to book a highly rated restaurant for me and my boyfriend tomorrow night)
- PlayMusic (e.g. Play the last track from Beyoncé off Spotify)
- AddToPlaylist (e.g. Add Diamonds to my roadtrip playlist)

```
!gdown --id 1OlcvGWReJMuyYQuOZm149vHWwPtlboR6 --output train.csv
!gdown --id 1Oi5cRlTybuIF2Fl5Bfsr-KkqrXrdt77w --output valid.csv
!gdown --id 1ep9H6-HvhB4utJRLVcLzieWNUSG3P_uF --output test.csv
```

We'll load the data into data frames and expand the training data by merging the training and validation intents:

```
train = pd.read_csv("train.csv")
valid = pd.read_csv("valid.csv")
test = pd.read_csv("test.csv")

train = train.append(valid).reset_index(drop=True)
```

We have `13,784` training examples and two columns - `text` and `intent`. Let's have a look at the number of texts per intent:

The amount of texts per intent is quite balanced, so we'll not be needing any imbalanced modeling techniques.

BERT

The BERT (Bidirectional Encoder Representations from Transformers) model, introduced in the BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper, made possible achieving State-of-the-art results in a variety of NLP tasks, for the regular ML practitioner. And you can do it without having a large dataset! But how is this possible?

BERT is a pre-trained Transformer Encoder stack. It is trained on Wikipedia and the Book Corpus dataset. It has two versions - Base (12 encoders) and Large (24 encoders).

BERT is built on top of multiple clever ideas by the NLP community. Some examples are ELMo, The Transformer, and the OpenAI Transformer.

ELMo introduced contextual word embeddings (one word can have a different meaning based on the words around it). The Transformer uses attention mechanisms to understand the context in which the word is being used. That context is then encoded into a vector representation. In practice, it does a better job with long-term dependencies.

BERT is a bidirectional model (looks both forward and backward). And the best of all, BERT can be easily used as a feature extractor or fine-tuned with small amounts of data. How good is it at recognizing intent from text?

Luckily, the authors of the BERT paper open-sourced their work along with multiple pre-trained models. The original implementation is in TensorFlow, but there are very good PyTorch implementations too!

Let's start by downloading one of the simpler pre-trained models and unzip it:

```
!wget https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zi
!unzip uncased_L-12_H-768_A-12.zip
```

This will unzip a checkpoint, config, and vocabulary, along with other files.

Unfortunately, the original implementation is not compatible with TensorFlow 2. The bert-for-tf2 package solves this issue.

Preprocessing

We need to convert the raw texts into vectors that we can feed into our model. We'll go through 3 steps:

- Tokenize the text
- Convert the sequence of tokens into numbers
- Pad the sequences so each one has the same length

Let's start by creating the BERT tokenizer:

```
tokenizer = FullTokenizer(
  vocab_file=os.path.join(bert_ckpt_dir, "vocab.txt")
)
```

Let's take it for a spin:

```
tokenizer.tokenize("I can't wait to visit Bulgaria again!")
```

```
['i', 'can', "'", 't', 'wait', 'to', 'visit', 'bulgaria', 'again', '!']
```

The tokens are in lowercase and the punctuation is available. Next, we'll convert the tokens to numbers. The tokenizer can do this too:

```
tokens = tokenizer.tokenize("I can't wait to visit Bulgaria again!")
tokenizer.convert_tokens_to_ids(tokens)
```

We'll package the preprocessing into a class that is heavily based on the one from this notebook:

```
class IntentDetectionData:
  DATA_COLUMN = "text"
  LABEL_COLUMN = "intent"

  def __init__(
    self,
    train,
    test,
    tokenizer: FullTokenizer,
    classes,
    max_seq_len=192
  ):
    self.tokenizer = tokenizer
    self.max_seq_len = 0
    self.classes = classes

    ((self.train_x, self.train_y), (self.test_x, self.test_y)) =\
     map(self._prepare, [train, test])

    print("max seq_len", self.max_seq_len)
    self.max_seq_len = min(self.max_seq_len, max_seq_len)
    self.train_x, self.test_x = map(
      self._pad,
      [self.train_x, self.test_x]
    )

  def _prepare(self, df):
    x, y = [], []

    for _, row in tqdm(df.iterrows()):
      text, label =\
       row[IntentDetectionData.DATA_COLUMN], \
       row[IntentDetectionData.LABEL_COLUMN]
      tokens = self.tokenizer.tokenize(text)
      tokens = ["[CLS]"] + tokens + ["[SEP]"]
      token_ids = self.tokenizer.convert_tokens_to_ids(tokens)
      self.max_seq_len = max(self.max_seq_len, len(token_ids))
      x.append(token_ids)
      y.append(self.classes.index(label))

    return np.array(x), np.array(y)

  def _pad(self, ids):
    x = []
    for input_ids in ids:
      input_ids = input_ids[:min(len(input_ids), self.max_seq_len - 2)]
      input_ids = input_ids + [0] * (self.max_seq_len - len(input_ids))
      x.append(np.array(input_ids))
    return np.array(x)
```

We figure out the padding length by taking the minimum between the longest text and the max sequence length parameter. We also surround the tokens for each text with two special tokens: start with `[CLS]` and end with `[SEP]`.

Fine-tuning

```
  with tf.io.gfile.GFile(bert_config_file, "r") as reader:
      bc = StockBertConfig.from_json_string(reader.read())
      bert_params = map_stock_config_to_params(bc)
      bert_params.adapter_size = None
      bert = BertModelLayer.from_params(bert_params, name="bert")

  input_ids = keras.layers.Input(
    shape=(max_seq_len, ),
    dtype='int32',
    name="input_ids"
  )
  bert_output = bert(input_ids)

  print("bert shape", bert_output.shape)

  cls_out = keras.layers.Lambda(lambda seq: seq[:, 0, :])(bert_output)
  cls_out = keras.layers.Dropout(0.5)(cls_out)
  logits = keras.layers.Dense(units=768, activation="tanh")(cls_out)
  logits = keras.layers.Dropout(0.5)(logits)
  logits = keras.layers.Dense(
    units=len(classes),
    activation="softmax"
  )(logits)

  model = keras.Model(inputs=input_ids, outputs=logits)
  model.build(input_shape=(None, max_seq_len))

  load_stock_weights(bert, bert_ckpt_file)

  return model
```

We're fine-tuning the pre-trained BERT model using our inputs (text and intent). We also flatten the output and add Dropout with two Fully-Connected layers. The last layer has a softmax activation function. The number of outputs is equal to the number of intents we have - seven.

You can now use BERT to recognize intents!

Training

It is time to put everything together. We'll start by creating the data object:

```
classes = train.intent.unique().tolist()

data = IntentDetectionData(
  train,
  test,
  tokenizer,
  classes,
  max_seq_len=128
)
```

We can now create the model using the maximum sequence length:

You'll notice that even this "slim" BERT has almost 110 million parameters. Indeed, your model is HUGE (that's what she said).

Fine-tuning models like BERT is both art and doing tons of failed experiments. Fortunately, the authors made some recommendations:

- Batch size: 16, 32
- Learning rate (Adam): 5e-5, 3e-5, 2e-5
- Number of epochs: 2, 3, 4

```
model.compile(
  optimizer=keras.optimizers.Adam(1e-5),
  loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")]
)
```

We'll use Adam with a slightly different learning rate (cause we're badasses) and use sparse categorical crossentropy, so we don't have to one-hot encode our labels.

Let's fit the model:

```
log_dir = "log/intent_detection/" +\
 datetime.datetime.now().strftime("%Y%m%d-%H%M%s")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=log_dir)

model.fit(
  x=data.train_x,
  y=data.train_y,
  validation_split=0.1,
  batch_size=16,
  shuffle=True,
  epochs=5,
  callbacks=[tensorboard_callback]
)
```

We store the training logs, so you can explore the training process in Tensorboard. Let's have a look:

I got to be honest with you. I was impressed with the results. Training using only 12.5k samples we got:

```
_, train_acc = model.evaluate(data.train_x, data.train_y)
_, test_acc = model.evaluate(data.test_x, data.test_y)

print("train acc", train_acc)
print("test acc", test_acc)
```

```
train acc 0.9915119
test acc 0.9771429
```

Impressive, right? Let's have a look at the confusion matrix:

Finally, let's use the model to detect intent from some custom sentences:

```
sentences = [
  "Play our song now",
  "Rate this book as awful"
]

pred_tokens = map(tokenizer.tokenize, sentences)
pred_tokens = map(lambda tok: ["[CLS]"] + tok + ["[SEP]"], pred_tokens)
pred_token_ids = list(map(tokenizer.convert_tokens_to_ids, pred_tokens))

pred_token_ids = map(
  lambda tids: tids +[0]*(data.max_seq_len-len(tids)),
  pred_token_ids
)
pred_token_ids = np.array(list(pred_token_ids))

predictions = model.predict(pred_token_ids).argmax(axis=-1)

for text, label in zip(sentences, predictions):
  print("text:", text, "\nintent:", classes[label])
  print()
```

```
text: Play our song now
intent: PlayMusic

text: Rate this book as awful
intent: RateBook
```

Man, that's (clearly) gangsta! Ok, the examples might not be as diverse as real queries might

You now know how to fine-tune a BERT model for text classification. You probably already know that you can use it for a variety of other tasks, too! You just have to fiddle with the layers. EASY!

Run the complete notebook in your browser

The complete project on GitHub

Doing AI/ML feels a lot like having superpowers, right? Thanks to the wonderful NLP community, you can have superpowers, too! What will you use them for?

References

- BERT Fine-Tuning Tutorial with PyTorch
- SNIPS dataset
- The Illustrated BERT, ELMo, and co.
- BERT for dummies — Step by Step Tutorial
- Multi-label Text Classification using BERT – The Mighty Transformer

- Deep Learning
- Keras
- NLP
- Text Classification
- Python

Continue Your Machine Learning Journey:

Hacker's Guide to Machine Learning with Python

A hands-on guide to solving real-world Machine Learning problems with Scikit-Learn, TensorFlow 2, and Keras

Bio: Venelin Valkov (GitHub) is a Machine Learning Engineer working on document data extraction using Deep Learning. In his free time, he likes to write, work on side projects, ride his bike, and do deadlifts!

Original. Reposted with permission.

Related:

- BERT is changing the NLP landscape
- Lit BERT: NLP Transfer Learning In 3 Steps
- BERT, RoBERTa, DistilBERT, XLNet: Which one to use?

More On This Topic

- Building and Training Your First Neural Network with TensorFlow and Keras
- How to Train a Joint Entities and Relation Extraction Classifier...
- Building a Knowledge Graph for Job Search Using BERT
- Classifying Long Text Documents Using BERT

<= Previous post Next post =>