Get started Open in app

Follow 580K Followers

You have **1** free member-only story left this month. Sign up for Medium and get an extra one

BART for Paraphrasing with Simple Transformers

Paraphrasing is the act of expressing something using different words while retaining the original meaning. Let's see how we can do it with BART, a Sequence-to-Sequence Transformer Model.

Thilina Rajapakse Aug 5, 2020 · 8 min read ★

Get started Open in app

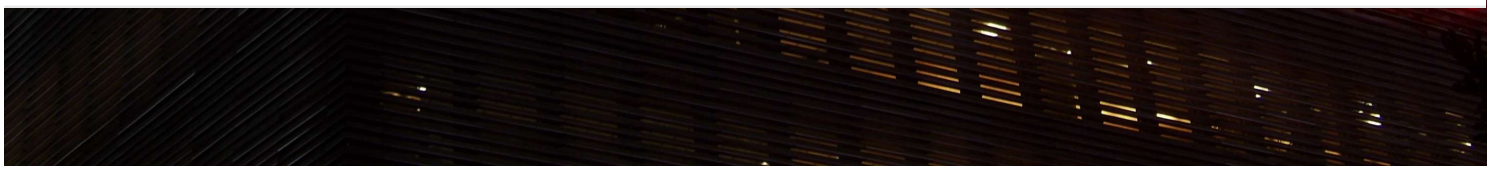


Photo by Alexandra on Unsplash

Introduction

> *BART is a denoising autoencoder for pretraining sequence-to-sequence models. BART is trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text.*
>
> *- BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension -*

Don't worry if that sounds a little complicated; we are going to break it down and see what it all means. To add a little bit of background before we dive into BART, it's time for the now-customary ode to Transfer Learning with self-supervised models. It's been said many times over the past couple of years, but Transformers really have achieved incredible success in a wide variety of Natural Language Processing (NLP) tasks.

BART uses a standard Transformer architecture (Encoder-Decoder) like the original Transformer model used for neural machine translation but also incorporates some changes from BERT (only uses the encoder) and GPT (only uses the decoder). You can refer to the *2.1 Architecture* section of the BART paper for more details.

Pre-Training BART

BART is pre-trained by minimizing the cross-entropy loss between the decoder output and the original sequence.

Masked Language Modeling (MLM)

MLM models such as BERT are pre-trained to predict masked tokens. This process can be broken down as follows:

Get started Open in app

2. The model predicts the original tokens for each of the [MASK] tokens. (Denoising)

Importantly, BERT models can "see" the full input sequence (with some tokens replaced with [MASK]) when attempting to predict the original tokens. This makes BERT a bidirectional model, i.e. it can "see" the tokens before and after the masked tokens.

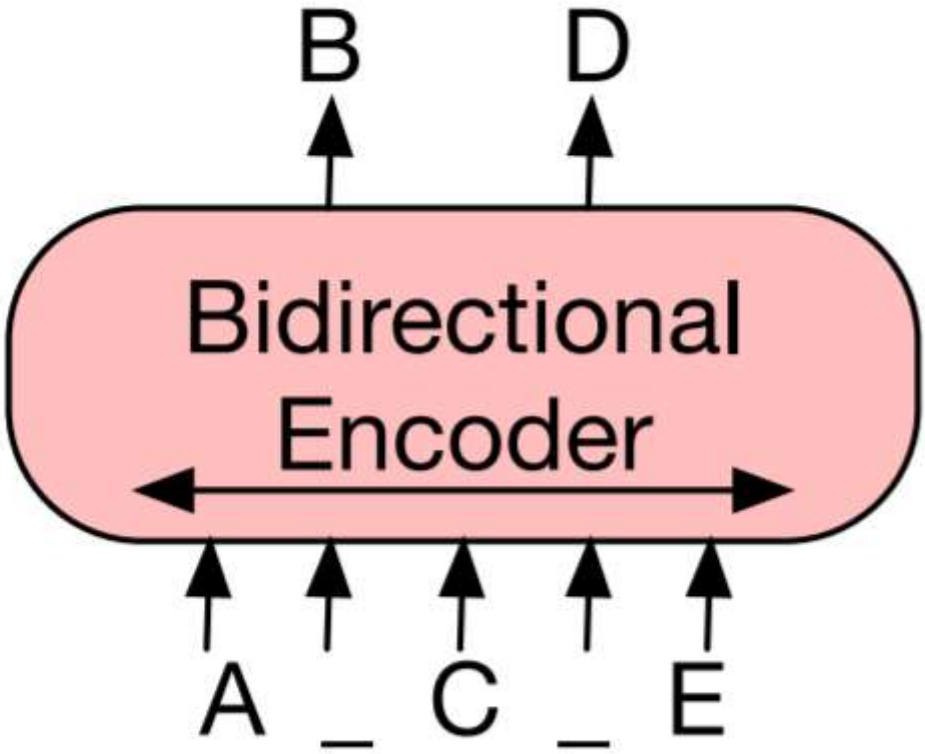


Figure 1 (a) from the BART paper

This is suited for tasks like classification where you can use information from the full sequence to perform the prediction. However, it is less suited for text generation tasks where the prediction depends only on the previous words.

Autoregressive Models

Models used for text generation, such as GPT2, are pre-trained to predict the next token given the previous sequence of tokens. This pre-training objective results in models that are well-suited for text generation, but not for tasks like classification.

Get started Open in app

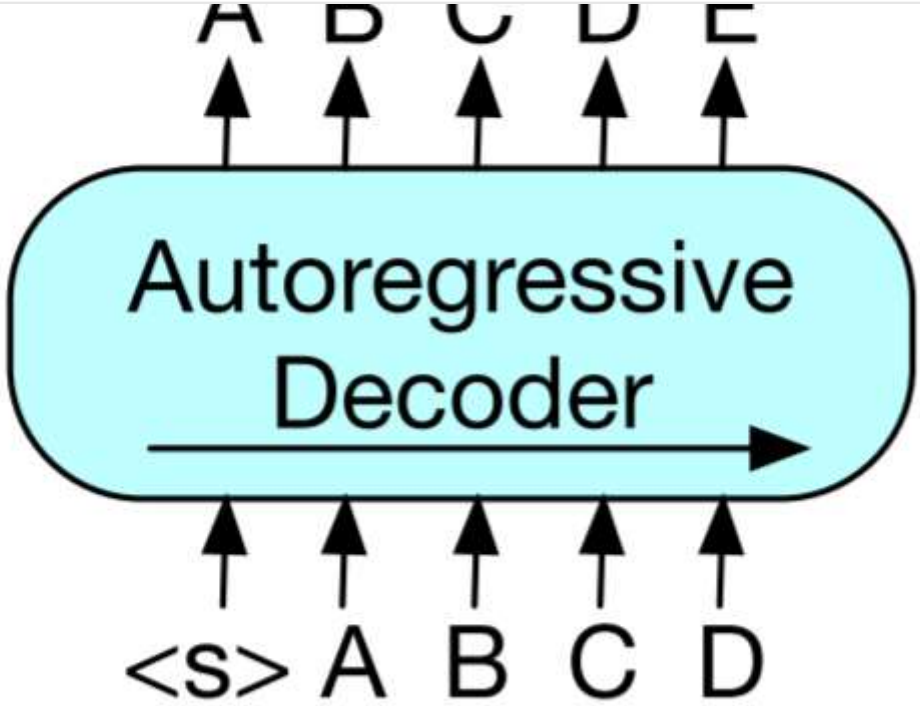
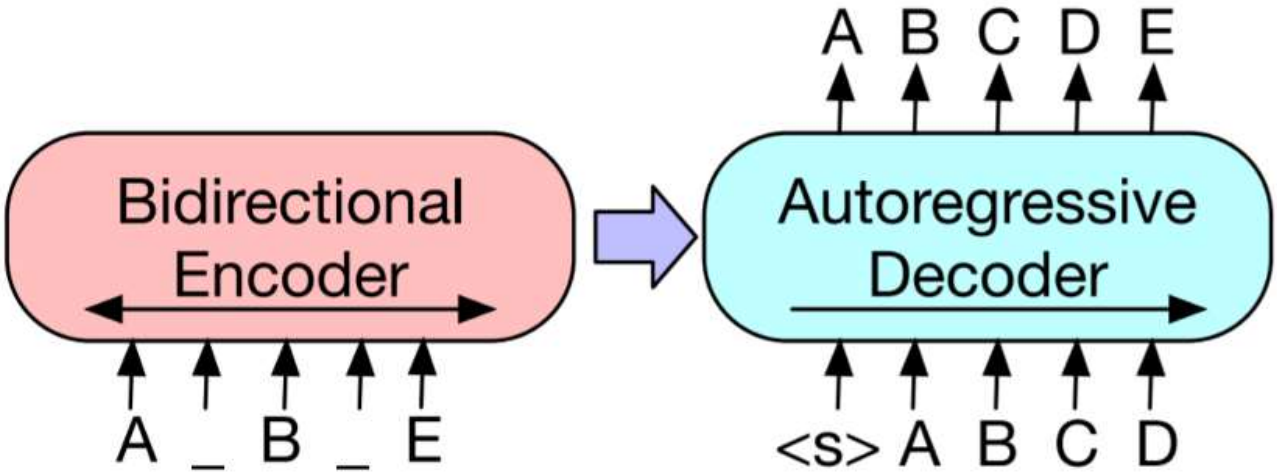


Figure 1 (b) from the BART paper

BART Sequence-to-Sequence

BART has both an encoder (like BERT) and a decoder (like GPT), essentially getting the best of both worlds.

The encoder uses a *denoising* objective similar to BERT while the decoder attempts to reproduce the original sequence *(autoencoder)*, token by token, using the previous (uncorrupted) tokens and the output from the encoder.



Get started Open in app

A significant advantage of this setup is the unlimited flexibility of choosing the corruption scheme; including changing the length of the original input. Or, in fancier terms, the text can be corrupted with an *arbitrary noising function*.

The corruption schemes used in the paper are summarized below.

1. Token Masking — A random subset of the input is replaced with [MASK] tokens, like in BERT.
2. Token Deletion — Random tokens are deleted from the input. The model must decide which positions are missing (as the tokens are simply deleted and not replaced with anything else).
3. Text Infilling — A number of text spans (length can vary) are each replaced with a single [MASK] token.
4. Sentence Permutation — The input is split based on periods (.), and the sentences are shuffled.
5. Document Rotation — A token is chosen at random, and the sequence is rotated so that it starts with the chosen token.

Search this file…

Corruption Scheme	Original Text	Corrupted Text	Explanation
Token Masking	ABC.DE.	A_C._E.	Both B and D are masked with a single mask token for each.
Token Deletion	ABC.DE.	A.C.E.	Both B and D are deleted (and not replaced).
Text Infilling	ABC.DE.	A_.D_E.	The span BC is replaced with a single mask token. A 0 length spar
Sentence Permutation	ABC.DE.	DE.ABC.	Split into sentences at periods (.) and shuffled.
Document Rotation	ABC.DE.	C.DE.AB	The sequence is rotated around C.

corrupt.csv hosted with ♡ by GitHub view raw

The authors note that training BART with *text infilling* yields the most consistently strong performance across many tasks.

Get started Open in app

sequence (paraphrased sentence) as a Sequence-to-Sequence model.

This also works for tasks like summarization and abstractive question answering.

Setup

We will use the Simple Transformers library, based on the Hugging Face Transformers library, to train the models.

1. Install Anaconda or Miniconda Package Manager from here.

2. Create a new virtual environment and install packages.

```
conda create -n st python pandas tqdm
conda activate st
```

3. If using CUDA:

```
conda install pytorch>=1.6 cudatoolkit=10.2 -c pytorch
```

else:

```
conda install pytorch cpuonly -c pytorch
```

4. Install simpletransformers.

```
pip install simpletransformers
```

Data Preparation

Get started Open in app

1. Google PAWS-Wiki Labeled (Final)
2. Quora Question Pairs Dataset
3. Microsoft Research Paraphrase Corpus (MSRP)

The bash script below can be used to easily download and prep the first two datasets, but the MSRP dataset has to be downloaded manually from the link. (Microsoft hasn't provided a direct link 😞)

Make sure you place the files in the same directory (`data`) to avoid annoyances with file paths in the example code.

```
mkdir data
wget https://storage.googleapis.com/paws/english/paws_wiki_labeled_final.tar.gz -P data
tar -xvf data/paws_wiki_labeled_final.tar.gz -C data
mv data/final/* data
rm -r data/final

wget http://qim.fs.quoracdn.net/quora_duplicate_questions.tsv -P data
```

para_data.sh hosted with ♡ by GitHub view raw

We also have a couple of helper functions, one to load data, and one to clean unnecessary spaces in the training data. Both of these functions are defined in `utils.py` .

```
import warnings

import pandas as pd


def load_data(
    file_path, input_text_column, target_text_column, label_column, keep_label=1
):
    df = pd.read_csv(file_path, sep="\t", error_bad_lines=False)
    df = df.loc[df[label_column] == keep_label]
    df = df.rename(
        columns={input_text_column: "input_text", target_text_column: "target_text"}
```

Get started Open in app

```
    return df


def clean_unnecessary_spaces(out_string):
    if not isinstance(out_string, str):
        warnings.warn(f">>> {out_string} <<< is not a string.")
        out_string = str(out_string)
    out_string = (
        out_string.replace(" .", ".")
        .replace(" ?", "?")
        .replace(" !", "!")
        .replace(" ,", ",")
        .replace(" ' ", "'")
        .replace(" n't", "n't")
        .replace(" 'm", "'m")
        .replace(" 's", "'s")
        .replace(" 've", "'ve")
        .replace(" 're", "'re")
    )
    return out_string
```

utils.py hosted with ♡ by GitHub view raw

Some of the data have spaces before punctuation marks that we need to remove. `clean_unnecessary_spaces()` *function is used for this purpose.*

Paraphrasing with BART

Once the data is prepared, training the model is quite simple.

Note that you can find all the code in the Simple Transformers examples here.

First, we import all the necessary stuff and set up logging.

```
import os
from datetime import datetime
import logging

import pandas as pd
from sklearn.model_selection import train_test_split
```

Get started Open in app

```
10
11
12  logging.basicConfig(level=logging.INFO)
13  transformers_logger = logging.getLogger("transformers")
14  transformers_logger.setLevel(logging.ERROR)
```

para_import.py hosted with ♡ by GitHub view raw

Next, we load the datasets.

```
1   # Google Data
2   train_df = pd.read_csv("data/train.tsv", sep="\t").astype(str)
3   eval_df = pd.read_csv("data/dev.tsv", sep="\t").astype(str)
4
5   train_df = train_df.loc[train_df["label"] == "1"]
6   eval_df = eval_df.loc[eval_df["label"] == "1"]
7
8   train_df = train_df.rename(
9       columns={"sentence1": "input_text", "sentence2": "target_text"}
10  )
11  eval_df = eval_df.rename(
12      columns={"sentence1": "input_text", "sentence2": "target_text"}
13  )
14
15  train_df = train_df[["input_text", "target_text"]]
16  eval_df = eval_df[["input_text", "target_text"]]
17
18  train_df["prefix"] = "paraphrase"
19  eval_df["prefix"] = "paraphrase"
20
21  # MSRP Data
22  train_df = pd.concat(
23      [
24          train_df,
25          load_data("data/msr_paraphrase_train.txt", "#1 String", "#2 String", "Quality"),
26      ]
27  )
28  eval_df = pd.concat(
29      [
30          eval_df,
31          load_data("data/msr_paraphrase_test.txt", "#1 String", "#2 String", "Quality"),
```

Get started Open in app

```
# Quora Data

# The Quora Dataset is not separated into train/test, so we do it manually the first time.
df = load_data(
    "data/quora_duplicate_questions.tsv", "question1", "question2", "is_duplicate"
)
q_train, q_test = train_test_split(df)

q_train.to_csv("data/quora_train.tsv", sep="\t")
q_test.to_csv("data/quora_test.tsv", sep="\t")

# The code block above only needs to be run once.
# After that, the two lines below are sufficient to load the Quora dataset.

# q_train = pd.read_csv("data/quora_train.tsv", sep="\t")
# q_test = pd.read_csv("data/quora_test.tsv", sep="\t")

train_df = pd.concat([train_df, q_train])
eval_df = pd.concat([eval_df, q_test])

train_df = train_df[["prefix", "input_text", "target_text"]]
eval_df = eval_df[["prefix", "input_text", "target_text"]]

train_df = train_df.dropna()
eval_df = eval_df.dropna()

train_df["input_text"] = train_df["input_text"].apply(clean_unnecessary_spaces)
train_df["target_text"] = train_df["target_text"].apply(clean_unnecessary_spaces)

eval_df["input_text"] = eval_df["input_text"].apply(clean_unnecessary_spaces)
eval_df["target_text"] = eval_df["target_text"].apply(clean_unnecessary_spaces)

print(train_df)
```

Then, we set up the model and hyperparameter values. Note that we are using the pre-trained `facebook/bart-large` model, and fine-tuning it on our own dataset.

Finally, we'll generate paraphrases for each of the sentences in the test data.

```
model_args = Seq2SeqArgs()
```

Get started Open in app

```
model_args.evaluate_during_training = True
model_args.evaluate_during_training_steps = 2500
model_args.evaluate_during_training_verbose = True
model_args.fp16 = False
model_args.learning_rate = 5e-5
model_args.max_length = 128
model_args.max_seq_length = 128
model_args.num_beams = None
model_args.num_return_sequences = 3
model_args.num_train_epochs = 2
model_args.overwrite_output_dir = True
model_args.reprocess_input_data = True
model_args.save_eval_checkpoints = False
model_args.save_steps = -1
model_args.top_k = 50
model_args.top_p = 0.95
model_args.train_batch_size = 8
model_args.use_multiprocessing = False
model_args.wandb_project = "Paraphrasing with BART"


model = Seq2SeqModel(
    encoder_decoder_type="bart",
    encoder_decoder_name="facebook/bart-large",
    args=model_args,
)

model.train_model(train_df, eval_data=eval_df)

to_predict = [
    prefix + ": " + str(input_text)
    for prefix, input_text in zip(eval_df["prefix"].tolist(), eval_df["input_text"].tolist())
]
truth = eval_df["target_text"].tolist()

preds = model.predict(to_predict)

# Saving the predictions if needed
os.makedirs("predictions", exist_ok=True)

with open(f"predictions/predictions_{datetime.now()}.txt", "w") as f:
    for i, text in enumerate(eval_df["input_text"].tolist()):
```

Get started Open in app

```
        f.write(truth[i] + "\n\n")

        f.write("Prediction:\n")
        for pred in preds[i]:
            f.write(str(pred) + "\n")
        f.write(
            "________________________________________________________________________________\n"
        )
```

This will write the predictions to the `predictions` *directory.*

Hyperparameters

The hyperparameter values are set to general, sensible values without doing hyperparameter optimization. For this task, the *ground truth* does not represent the only possible correct answer (nor is it necessarily the *best* answer). Because of this, tuning the hyperparameters to nudge the generated text as close to the *ground truth* as possible doesn't make much sense.

Our aim is to generate good paraphrased sequences rather than to produce the exact paraphrased sequence from the dataset.

If you are interested in Hyperparameter Optimization with Simple Transformers (particularly useful with other models/tasks like classification), do check out my guide here.

Hyperparameter Optimization for Optimum Transformer Models

How to tune your hyperparameters with Simple Transformers for better Natural Langauge Processing.

towardsdatascience.com

The decoding algorithm (and the relevant hyperparameters) used has a considerable impact on the quality and nature of the generated text. The values I've chosen (shown below) are generally suited to produce "natural" text.

Get started Open in app

Try out the model on your own sentences

You can use the script below to test the model on any sentence.

Results

Let's look at some of the paraphrased sequences generated by the model for the test data. For each input sequence, the model will generate three (`num_return_sequences`) paraphrased sequences.

1.

```
Orignal:

A recording of folk songs done for the Columbia society in 1942 was
largely arranged by Pjetër Dungu.

Truth:

A recording of folk songs made for the Columbia society in 1942 was
largely arranged by Pjetër Dungu.

Prediction:

A recording of folk songs made in 1942 for the Columbia Society was
largely arranged by Pjetr Dungu.

A recording of folk songs for the Columbia society in 1942 was
largely arranged by Pjetr Dungu.

A recording of folk songs done for the Columbia Society in 1942 was
largely arranged by Pjetr Dungu.
```

2.

```
Original:

In mathematical astronomy, his fame is due to the introduction of the
astronomical globe, and his early contributions to understanding the
movement of the planets.
```

Get started Open in app

```
astronomical globe and to his early contributions to the
understanding of the movement of the planets.

Prediction:

His fame in mathematical astronomy is due to the introduction of the
astronomical globe and his early contributions to understanding the
movement of the planets.

In mathematical astronomy, his fame is due to the introduction of the
astronomical globe and his early contributions to understanding the
motion of the planets.

In mathematical astronomy his fame is due to the introduction of the
astronomical globe and his early contributions to understanding the
movement of the planets.
```

3.

```
Original:

Why are people obsessed with Cara Delevingne?

Truth:

Why are people so obsessed with Cara Delevingne?

Prediction:

Why do people fall in love with Cara Delevingne?

Why is everyone obsessed with Cara Delevingne?

Why do people like Cara Delevingne?
```

4.

```
Original:

Earl St Vincent was a British ship that was captured in 1803 and
became a French trade man.
```

Get started Open in app

```
French merchantman in 1803.

Prediction:

Earl St Vincent was a British ship captured in 1803 and became a
French trader.

Earl St Vincent was a British ship captured in 1803 and became a
French trade man.

Earl St Vincent was a British ship that was captured in 1803 and
became a French trade man.
```

5.

```
Original:

Worcester is a town and county city of Worcestershire in England.

Truth:

Worcester is a city and county town of Worcestershire in England.

Prediction:

Worcester is a town and county of Worcestershire in England.

Worcester is a town and county town in Worcestershire in England.

Worcester is a town and county town of Worcestershire in England.
```

6. Out of domain sentence

```
Original:

The goal of any Deep Learning model is to take in an input and
generate the correct output.

Predictions >>>
The goal of any deep learning model is to take an input and generate
the correct output.
```

Get started Open in app

```
Any Deep Learning model the goal of which is to take in an input and
generate the correct output.
```

As can be seen from these examples, our BART model has learned to generate paraphrases quite well!

Discussion

Potential Problems

The generated paraphrases can sometimes have minor issues, some of which are listed below.

1. The generated sequence is almost identical to the original with only minor differences in a word or two.
2. Incorrect or awkward grammar.
3. Might not be as good on out of domain (from training data) inputs.

Encouragingly, these issues seem to be quite rare and can most likely be averted by using better training data (the same problems can sometimes be seen in the training data ground truth as well).

Wrap Up

Sequence-to-Sequence models like BART are another arrow in the quiver of NLP practitioners. They are particularly useful for tasks involving text generation such as paraphrasing, summarization, and abstractive question answering.

Paraphrasing can be used for data augmentation where you can create a larger dataset by paraphrasing the available data.

Sign up for The Variable
By Towards Data Science

Get started Open in app

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Artificial Intelligence Machine Learning Data Science NLP

About Help Legal

Get the Medium app

Download on the App Store GET IT ON Google Play