Analytics Vidhya

Syed Abdul Gaffar Shakhadri
12 Jun, 2021 • 6 min read

This article was published as a part of the [Data Science Blogathon](#)

Introduction

Natural Language Processing (NLP) is a field at the convergence of artificial intelligence, and linguistics. The aim is to make the computers understand real-world language or natural language so that they can perform tasks like Question Answering, Language Translation, and many more.

NLP has lots of applications in different fields.

1. NLP enables the recognition and prediction of diseases based on electronic health records.

2. It is used to obtain customer reviews.

3. To help to identify fake news.

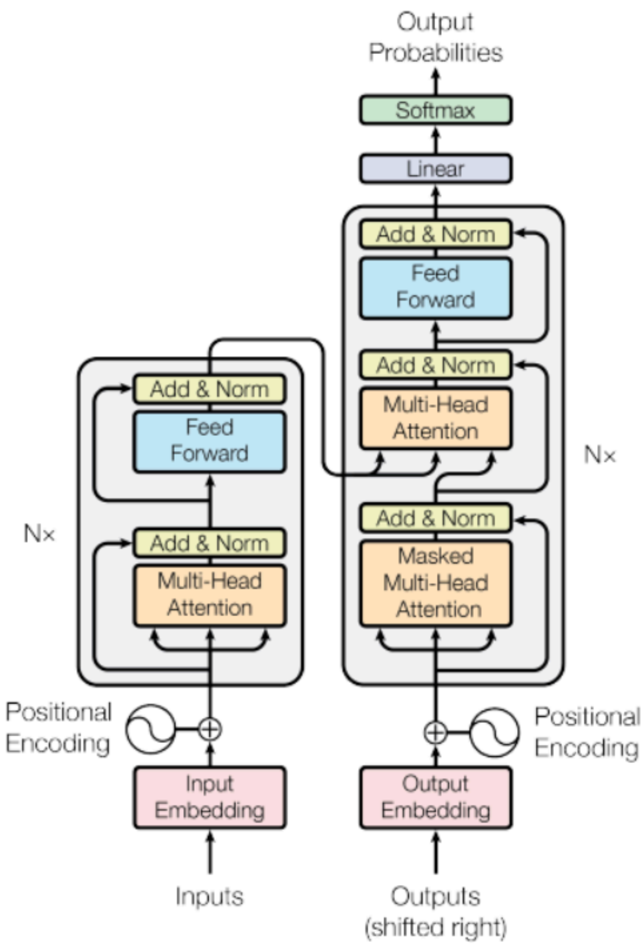
4. Chatbots.

5. Social Media monitoring etc.

What is the Transformer?

and Illia Polosukhin in their paper "Attention Is All You Need". [1]

The Transformer model extracts the features for each word using a self-attention mechanism to know the importance of each word in the sentence. No other recurrent units are used to extract this feature, they are just activations and weighted sums, so they can be very efficient and parallelizable.



Source: " Attention Is All You Need" paper

In the above figure, there is an encoder model on the left side and the decoder on the right. Both encoder and decoder contain a core block of **attention and a feed-forward network** repeated N number of times.

In the above figure, there is an encoder model on the left side and the decoder on the right. Both encoder and decoder contain a core block of attention and a feedcontains two layers(sub-layers), that is a multi-head self-attention layer, and a fully connected feed-forward network. The Decoder contains three layers(sub-layers), a multi-head self-attention layer, another multi-head self-attention layer to perform self-attention over encoder outputs, and a fully connected feed-forward network. Each sub-layer in Decoder and Encoder has a Residual connection with layer normalization.

Let's Start Building Language Translation Model

Here we will be using the Multi30k dataset. Don't worry the dataset will be downloaded with a piece of code.

First the Data processing part we will use the *torchtext* module from PyTorch. The *torchtext* has utilities for creating datasets that can be easily iterated for the purposes of creating a language translation model. The below code will download the dataset and also tokenizes a raw text, build the vocabulary, and convert tokens into a tensor.

```
import math
import torchtext
import torch
import torch.nn as nn
from torchtext.data.utils import get_tokenizer
from collections import Counter
from torchtext.vocab import Vocab
from torchtext.utils import download_from_url, extract_arch
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import DataLoader
from torch import Tensor
from torch.nn import (TransformerEncoder, TransformerDecode
import io
import time
```

```
url_base = 'https://raw.githubusercontent.com/multi30k/data
train_urls = ('train.de.gz', 'train.en.gz')
val_urls = ('val.de.gz', 'val.en.gz')
test_urls = ('test_2016_flickr.de.gz', 'test_2016_flickr.en

train_filepaths = [extract_archive(download_from_url(url_ba
val_filepaths = [extract_archive(download_from_url(url_base
test_filepaths = [extract_archive(download_from_url(url_bas

de_tokenizer = get_tokenizer('spacy', language='de_core_new
en_tokenizer = get_tokenizer('spacy', language='en_core_web

def build_vocab(filepath, tokenizer):
counter = Counter()
with io.open(filepath, encoding="utf8") as f:
for string_ in f:
counter.update(tokenizer(string_))
```

```
raw_de_iter = iter(io.open(filepaths[0], encoding="utf8"))
raw_en_iter = iter(io.open(filepaths[1], encoding="utf8"))
data = []
for (raw_de, raw_en) in zip(raw_de_iter, raw_en_iter):
de_tensor_ = torch.tensor([de_vocab[token] for token in de_
dtype=torch.long)
en_tensor_ = torch.tensor([en_vocab[token] for token in en_
dtype=torch.long)
data.append((de_tensor_, en_tensor_))
return data


train_data = data_process(train_filepaths)
val_data = data_process(val_filepaths)
test_data = data_process(test_filepaths)
device = torch.device('cuda' if torch.cuda.is_available() e


BATCH_SIZE = 128
PAD_IDX = de_vocab['<pad>']
BOS_IDX = de_vocab['<bos>']
EOS_IDX = de_vocab['<eos>']
```

Then we will use the PyTorch DataLoader module which combines a dataset and a sampler, and it enables us to iterate over the given dataset. The DataLoader supports both iterable-style and map-style datasets with single or multi-process loading, also we can customize loading order and memory pinning.

```
# DataLoader
def generate_batch(data_batch):
  de_batch, en_batch = [], []
  for (de_item, en_item) in data_batch:
    de_batch.append(torch.cat([torch.tensor([BOS_IDX]), de_
    en_batch.append(torch.cat([torch.tensor([BOS_IDX]), en_
  de_batch = pad_sequence(de_batch, padding_value=PAD_IDX)
  en_batch = pad_sequence(en_batch, padding_value=PAD_IDX)
  return de_batch, en_batch
```

```
train_iter = DataLoader(train_data, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=generate_batch)
valid_iter = DataLoader(val_data, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=generate_batch)
test_iter = DataLoader(test_data, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=generate_batch)
```

Then we are designing the transformer. Here the Encoder processes the input sequence by propagating it through a series of Multi-head Attention and Feedforward network layers. The output from this Encoder is referred to as memory below and is fed to the decoder along with target tensors. Encoder and decoder are trained in an end-to-end fashion.

```
# transformer
class Seq2SeqTransformer(nn.Module):
```

```
        decoder_layer = TransformerDecoderLayer(d_model=emb
                                                dim_feedfor
        self.transformer_decoder = TransformerDecoder(decod

        self.generator = nn.Linear(emb_size, tgt_vocab_size
        self.src_tok_emb = TokenEmbedding(src_vocab_size, e
        self.tgt_tok_emb = TokenEmbedding(tgt_vocab_size, e
        self.positional_encoding = PositionalEncoding(emb_s

    def forward(self, src: Tensor, trg: Tensor, src_mask: T
                tgt_mask: Tensor, src_padding_mask: Tensor,
                tgt_padding_mask: Tensor, memory_key_paddin
        src_emb = self.positional_encoding(self.src_tok_emb
        tgt_emb = self.positional_encoding(self.tgt_tok_emb
        memory = self.transformer_encoder(src_emb, src_mask
        outs = self.transformer_decoder(tgt_emb, memory, tg
                                        tgt_padding_mask, m
        return self.generator(outs)

    def encode(self, src: Tensor, src_mask: Tensor):
        return self.transformer_encoder(self.positional_enc
                            self.src_tok_emb(src)), src_mas

    def decode(self, tgt: Tensor, memory: Tensor, tgt_mask:
        return self.transformer_decoder(self.positional_enc
                          self.tgt_tok_emb(tgt)), memory,
                          tgt_mask)
```

The Text which is converted to tokens is represented by using token embeddings. The Positional encoding function is added to the token embedding so that we can get the notions of word order.

```
class PositionalEncoding(nn.Module):
    def __init__(self, emb_size: int, dropout, maxlen: int
        super(PositionalEncoding, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2) * ma
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)
```

```
        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding

    def forward(self, token_embedding: Tensor):
        return self.dropout(token_embedding +
                            self.pos_embedding[:token_embed

class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size: int, emb_size):
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.emb_size = emb_size
    def forward(self, tokens: Tensor):
        return self.embedding(tokens.long()) * math.sqrt(se
```

Here in the below code, a subsequent word mask is created to stop a target word from attending to its subsequent words. Here the masks are also created, for

```
    return mask
```

```
def create_mask(src, tgt):
    src_seq_len = src.shape[0]
    tgt_seq_len = tgt.shape[0]

    tgt_mask = generate_square_subsequent_mask(tgt_seq_len)
    src_mask = torch.zeros((src_seq_len, src_seq_len), devi

    src_padding_mask = (src == PAD_IDX).transpose(0, 1)
    tgt_padding_mask = (tgt == PAD_IDX).transpose(0, 1)
    return src_mask, tgt_mask, src_padding_mask, tgt_paddin
```

Then define the model parameters and instantiate the model.

```
SRC_VOCAB_SIZE = len(de_vocab)
TGT_VOCAB_SIZE = len(en_vocab)
EMB_SIZE = 512
NHEAD = 8
FFN_HID_DIM = 512
BATCH_SIZE = 128
NUM_ENCODER_LAYERS = 3
NUM_DECODER_LAYERS = 3
NUM_EPOCHS = 50
DEVICE = torch.device('cuda:0' if torch.cuda.is_available()
```

```
transformer = Seq2SeqTransformer(NUM_ENCODER_LAYERS, NUM_DE
                                 EMB_SIZE, SRC_VOCAB_SIZE, 
                                 FFN_HID_DIM)

for p in transformer.parameters():
    if p.dim() > 1:
        nn.init.xavier_uniform_(p)

transformer = transformer.to(device)

loss_fn = torch.nn.CrossEntropyLoss(ignore_index=PAD_IDX)

optimizer = torch.optim.Adam(
    transformer.parameters(), lr=0.0001, betas=(0.9, 0.98),
)
```

Define two different functions, that is for train and evaluation.

```
def train_epoch(model, train_iter, optimizer):
    model.train()
    losses = 0
    for idx, (src, tgt) in enumerate(train_iter):
        src = src.to(device)
        tgt = tgt.to(device)

        tgt_input = tgt[:-1, :]

        src_mask, tgt_mask, src_padding_mask, tgt_padding_m

        logits = model(src, tgt_input, src_mask, tgt_mask,
                       src_padding_mask, tgt_padding_mask, 

        optimizer.zero_grad()

        tgt_out = tgt[1:, :]
```

```
    return losses / len(train_iter)


def evaluate(model, val_iter):
    model.eval()
    losses = 0
    for idx, (src, tgt) in (enumerate(valid_iter)):
        src = src.to(device)
        tgt = tgt.to(device)

        tgt_input = tgt[:-1, :]

        src_mask, tgt_mask, src_padding_mask, tgt_padding_m

        logits = model(src, tgt_input, src_mask, tgt_mask,
                       src_padding_mask, tgt_padding_mask,
        tgt_out = tgt[1:, :]
        loss = loss_fn(logits.reshape(-1, logits.shape[-1])
        losses += loss.item()
    return losses / len(val_iter)
```

Now training the model.

```
for epoch in range(1, NUM_EPOCHS+1):
    start_time = time.time()
    train_loss = train_epoch(transformer, train_iter, optim
    end_time = time.time()
    val_loss = evaluate(transformer, valid_iter)
    print((f"Epoch: {epoch}, Train loss: {train_loss:.3f}, '
```


This model is trained using transformer architecture in such a way that it trains faster and also it converges to a lower validation loss compared to other RNN models.

```
def greedy_decode(model, src, src_mask, max_len, start_symb
    src = src.to(device)
    src_mask = src_mask.to(device)
```

```
    memory = model.encode(src, src_mask)
    ys = torch.ones(1, 1).fill_(start_symbol).type(torch.lo
    for i in range(max_len-1):
        memory = memory.to(device)
        memory_mask = torch.zeros(ys.shape[0], memory.shape
        tgt_mask = (generate_square_subsequent_mask(ys.size
                                    .type(torch.bool)).to(d
        out = model.decode(ys, memory, tgt_mask)
        out = out.transpose(0, 1)
        prob = model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim = 1)
        next_word = next_word.item()

        ys = torch.cat([ys,
                        torch.ones(1, 1).type_as(src.data).
        if next_word == EOS_IDX:
          break
    return ys


def translate(model, src, src_vocab, tgt_vocab, src_tokeniz
    model.eval()
    tokens = [BOS_IDX] + [src_vocab.stoi[tok] for tok in sr
    num_tokens = len(tokens)
```

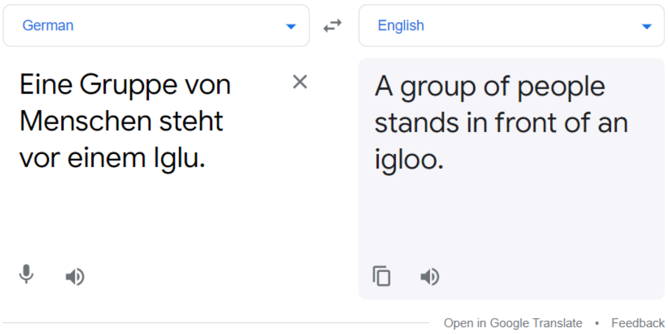
Now, let's test our model on translation.

```
output = translate(transformer, "Eine Gruppe von Menschen s
print(output)
```

```
C:\Users\syeda\AppData\Local\Programs\Python\Python37\python.exe C:/U
Epoch: 1, Train loss: 5.309, Val loss: 4.049, Epoch time = 36.965s
Epoch: 2, Train loss: 3.732, Val loss: 3.281, Epoch time = 36.074s
Epoch: 3, Train loss: 3.132, Val loss: 2.853, Epoch time = 35.444s
Epoch: 4, Train loss: 2.743, Val loss: 2.596, Epoch time = 35.158s
Epoch: 5, Train loss: 2.453, Val loss: 2.408, Epoch time = 35.740s
Epoch: 6, Train loss: 2.222, Val loss: 2.265, Epoch time = 36.425s
Epoch: 7, Train loss: 2.029, Val loss: 2.172, Epoch time = 35.370s
Epoch: 8, Train loss: 1.866, Val loss: 2.096, Epoch time = 36.051s
Epoch: 9, Train loss: 1.722, Val loss: 2.017, Epoch time = 37.421s
Epoch: 10, Train loss: 1.600, Val loss: 1.969, Epoch time = 36.303s
Epoch: 11, Train loss: 1.486, Val loss: 1.938, Epoch time = 37.810s
Epoch: 12, Train loss: 1.388, Val loss: 1.903, Epoch time = 38.530s
Epoch: 13, Train loss: 1.297, Val loss: 1.875, Epoch time = 37.677s
Epoch: 14, Train loss: 1.216, Val loss: 1.874, Epoch time = 40.317s
Epoch: 15, Train loss: 1.140, Val loss: 1.851, Epoch time = 37.000s
 A group of people stand in front of an igloo

Process finished with exit code 0
```

Above the red line is the output from the translation model. You can also compare it with google translator.



Source: Google Translator

The above translation and the output from our model matched. The model is not the best but still does the job up to some extent.

Reference

[1]. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin: Attention Is All You Need, Dec 2017, DOI: https://arxiv.org/pdf/1706.03762.pdf

Also for more information refer to

https://pytorch.org/tutorials/

Thank you

Analytics Vidhya and are used at the Author's discretion.

blogathon deep learning NLP transformer

Syed Abdul Gaffar Shakhadri
12 Jun 2021

I am an enthusiastic AI developer, I love playing with different problems and building solutions.

Advanced Data Science NLP Project Python

Responses From Readers

What are your thoughts?...

Submit reply

Salik Ahmed
04 Feb, 2022

Can you please provide me or help me yo find tenserflow code for the same..

• 70 Lessons ★ 5

Introduction to Python

FREE

Python

Enroll now

• 21 Lessons ★ 4.85

Introduction to Natural Language Processing

FREE

NLP

Enroll now

Write, Shine, Succeed

Write, captivate, and earn accolades and rewards for your work

- Reach a Global Audience
- Get Expert Feedback
- Build Your Brand & Audience
- Cash In on Your Knowledge
- Join a Thriving Community
- Level Up Your Data Science Game

Rahul Shah
27

Sion Chakrat
16

Company

About Us

Discover

Blogs