

Project Report on

Transfer Learning in NLP

By
Harsh Shah
2017B3A30650P

Under the guidance of
Prof. Surekha Bhanot



Department of Electrical & Electronics Engineering
Birla Institute of Technology and Science, Pilani
Pilani, Rajasthan
Second Semester, 2020-2021

Table of Contents

Introduction	3
Problem Statement	4
Literature Review	5
BERT	5
GPT	6
Research Gap	8
Objectives	9
Objectives Covered	10
Learning about current trends in NLP specifically in transfer learning	10
Comparing the performance of State-of-the-Art models	14
FineTuning T5	17
Leveraging a Single T5 model for multiple Downstream tasks	17
Conclusion	21
Major Learnings	22
Limitations	23
Future Scope	24
List of References	25
Appendix	27
1) Fine Tuning Pseudo Code	27
2) Model Arguments	29

Introduction

The recent progress in Machine Learning and Artificial Intelligence can be attributed to the improving computational capabilities of our existing hardware and the growing amount of data. The early 2010s saw a boom in computer vision based research where a lot of data had been labelled and most of the models followed some kind of supervised learning approach. Convolutional neural networks or CNNs were able to perform very well on image related tasks and the idea of using pretrained models for downstream tasks worked very well for a lot of vision based applications. CNNs were also used for Natural Language Processing but what worked best for language related applications was a different kind of Neural Network architecture called Recurrent Neural Networks or RNNs.

Variants of RNNs like Long-Short Term Memory (LSTMs) and Gated Recurrent Unit (GRU) started gaining popularity as NLP progressed. In 2017, the paper titled “Attention is all you need” by Vaswani introduced the concept of Transformers which were not only much more parallelizable than other RNNs but also reached new State-of-the-Art in many language modelling tasks. This attention mechanism coupled with different model architectures gave rise to the current day NLP models which are pre trained on large corpuses of unlabeled text and then fine tuned for other downstream tasks. Some of the most famous models which use this kind of approach are BERT, GPT and T5. In the next section, we will take a closer look at these models and understand what makes them perform so well.

Problem Statement

In this project the main aim is to understand the extent to which transfer learning is possible in Natural Language Processing. Encoder based models like BERT capture the semantics and understanding of the language they are trained on very well and give surprising results on downstream tasks like Sentiment Analysis, Sentence Classification and Information retrieval. If we examine these tasks we find that BERT excels at Sequence-to-vector tasks mainly.

Now if we focus on decoder based models like GPT2, it has amazing results on tasks like Sentence completion, Text Summarization and Neural Machine translation. We can notice that these are traditionally Sequence-to-Sequence tasks where both the input and output are some kind of sequences(text sequence to be more precise in this case).

Combining the best of both worlds, we get the T5 model which is a unified text-to-text model having both multiple Encoders and Decoders stacked in its architecture. It has the ability to understand the language and context while also being able to perform the tasks like Summarization and Translation.

Literature Review

Let us take a look at a few of the architectures which are current state-of-the-art in different Natural Language Processing tasks.

BERT

Bidirectional Encoder Representations from Transformers (BERT) was originally published by Google AI Language in 2018 where they used transformer architecture for the task of language modeling. One of the important things BERT does is stack Encoder-only Transformers blocks on top of each other.

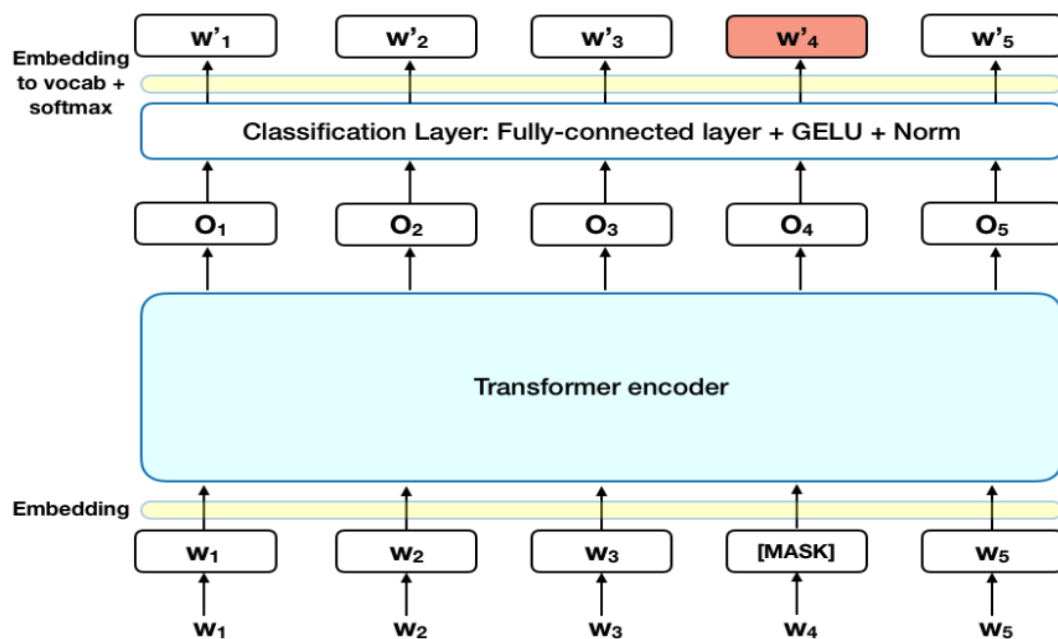


Fig 1. BERT Model for MLM task

BERT is trained on two different tasks :

- 1) Masked Language Modelling (MLM)
- 2) Next Sentence Prediction (NSP)

BERT is deeply bidirectional ie. the model learns from both left-to-right and right-to-left while going over input sentences. Also the self-attention combined with multi-head attention in each transformer encoder block helps to generate more contextual embeddings by taking into account other words in the sentence when encoding a particular word

Multiple attention heads, deeply bidirectional coupled with the combined training task of MLM and NSP are some of the reasons why the BERT language model has a really good understanding of the structure and semantics of English language. Due to these reasons, a single dense layer on top of the output of the BERT language model performs exceptionally well on many NLP tasks.

The Masked Language Modeling task produces a probability of word from the vocabulary at output and Next Sentence Prediction task is a binary classification model predicting whether Sentence A or B are the next sentence of the input.

BERT is trained in a way to minimize the mean loss from Masked LM + mean loss of Next Sentence Prediction. Both these use the **Cross Entropy Loss function**.

GPT

Generative Pre-trained Transformer or GPT is the model released by OpenAI in 2019. It has exceptional capabilities in terms of generating text and others tasks like Summarization.

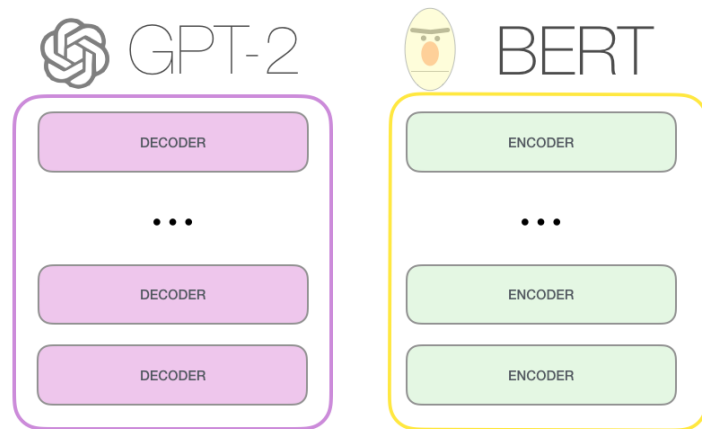


Fig 2. GPT vs BERT - Key Difference

The key difference between BERT and GPT as we can see from the above diagram is that BERT model architecture primarily contains Encoder blocks stacked on top of each other while GPT architecture focuses on Decoder blocks. This explains the language understanding prowess of BERT and the ability of GPT to generate coherent text.

The T5 model architecture aims to combine the powers of both these models in a single model capable of various language tasks from Sentiment Analysis to Summarization in a unified text-to-text model.

Research Gap

The need for a single model which can be used for the wide variety of tasks in NLP motivated researchers to come up with a single model which can handle both Sequence-to-Vector and Sequence-to-Sequence tasks and get good results. This led to the T5 language model which handles all these tasks and also reaches State-of-the-art results on many of them.

Objectives

The broad objectives of this project are:

- 1) Learning about current trends in NLP specifically in transfer learning
- 2) Comparing the performance of State-of-the-art models on certain tasks with finetuned T5/BERT models on same tasks
- 3) Attempt to train/finetune T5 model and reproduce its results
- 4) Leverage a single pretrained model for multiple downstream NLP tasks

Through these objectives we aim to understand the extent to which a single model can be fine tuned to be useful enough for very different tasks.

Objectives Covered

1) Learning about current trends in NLP specifically in transfer learning

Recent trend in NLP is to train large(a few billion parameters) language models on huge amounts of unlabeled text data on tasks like MLM which helps these models understand the language they are trained on well. For example BERT is able to generate meaningful embeddings of it's input text which can later be utilised for many Sequence-to-Vector tasks.

The model architectures of BERT and GPT model have already been covered in the Literature Review section. In this section we will go through the T5 model architecture.

T5 Model

Unified text-to-text transfer transformer model or better known as T5 model was released by Google Research in 2019. The original T5 model contains 11Bn parameters and is one of the largest language models even today.

Figure 3 gives us an idea of the different variety of tasks possible by the T5 model. Tasks are fed to the model in text format with the string of desired task appended at the beginning of the input text. The model also returns the output in text format.

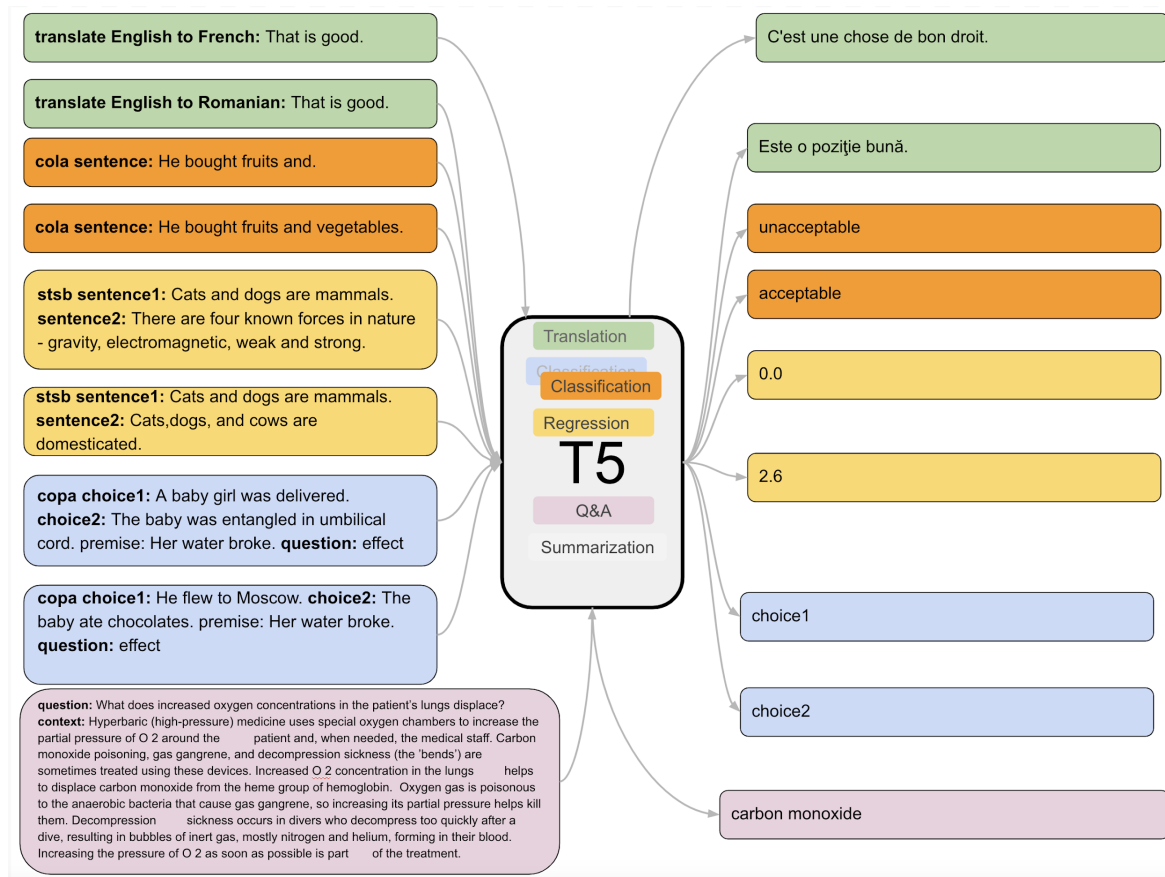


Fig 3. Tasks possible by T5 model

The similarities of T5 model with BERT is that they both use Masked Language modeling for pretraining while also utilising transformers as their building blocks. Figure 4 shows us the T5 model architecture.

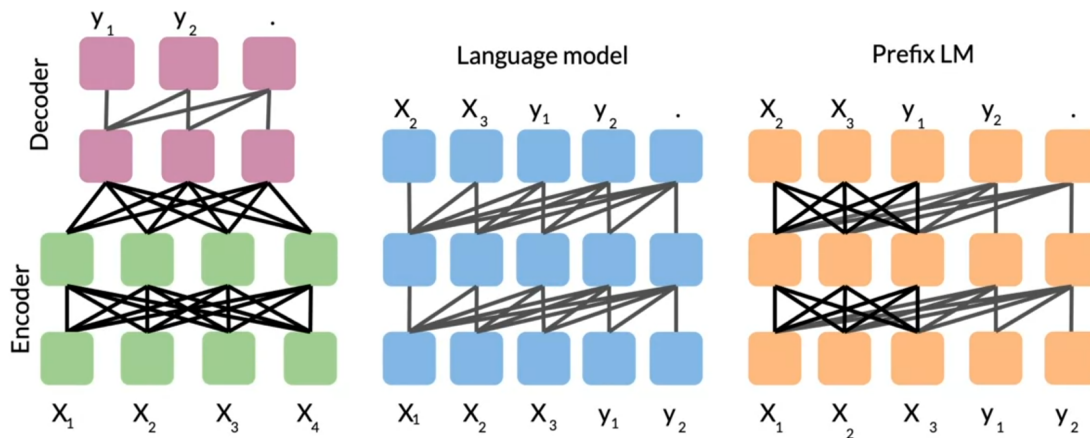


Fig 4. T5 model Architecture and Training

- 1) **Encoder-Decoder :** This is the standard encoder-decoder, seq2seq architecture wherein the encoder is trained in a BERT-style, fully visible manner
- 2) **Language Model :** This is essentially the causal attention mechanism, an autoregressive modeling approach
- 3) **Prefix LM :** This is a combination of the BERT-style and language model approaches. For example, the task of translating from English to German can have a BERT-style attention on: “translate English to German: That is good. target:”. And then the translation “Das ist gut.” will be attended autoregressively

T5 model follows the Prefix LM as shown above and the explanation for the same was given in the paper -

In the English to German translation example mentioned above, fully-visible masking would be applied to the prefix “translate English to German: That is good. target:” and causal masking would be used during training for predicting the target “Das ist gut.”

To make sure when autoregressively predicting Das - ist - gut in output, the model doesn't pay attention to "translate English to German" and only take it as the objective/task - fully visible masking is used on the input and causal masking is used on the output prediction part of training

Similar to the BERT model, T5 also uses **Cross Entropy Loss** in its loss function. The key difference is how we can use T5 and BERT for downstream tasks.

2) Comparing the performance of State-of-the-Art models

Comparing BERT and T5 on the [GLUE](#) benchmark we see that the T5 model ranks 6th on the [leaderboard](#) while the original BERT model ranks 32nd.

Although many BERT based models which take inspiration from the original architecture and improve on it like Microsoft's DeBERTa is ranked 2nd on the leaderboard and many other similar models are in top 5 also.











Rank	Name	Model	URL	Score	CoLA	SST-2
1	ERNIE Team - Baidu	ERNIE		90.9	74.4	97.8
2	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8	71.5	97.5
3	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0
4	Alibaba DAMO NLP	StructBERT + TAPT		90.6	75.3	97.3
5	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2
6	T5 Team - Google	T5		90.3	71.6	97.5
7	Microsoft D365 AI & MSR AI & GATECHMT-DNN-SMART			89.9	69.5	97.5
8	Huawei Noah's Ark Lab	NEZHA-Large		89.8	71.7	97.3
9	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)		89.7	70.5	97.5
10	ELECTRA Team	ELECTRA-Large + Standard Tricks		89.4	71.7	97.1
11	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)		88.4	68.0	96.8
12	Junjie Yang	HIRE-RoBERTa		88.3	68.6	97.1
13	Facebook AI	RoBERTa		88.1	67.8	96.7

Fig 5. GLUE Leaderboard

A few code examples for T5:

1) Importing relevant libraries and tokenizers:

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

import numpy as np
import pandas as pd
import torch
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, RandomSampler, SequentialSampler
from transformers import T5Tokenizer, T5ForConditionalGeneration, T5Config, T5Model

Invidia-smi

...

from torch import cuda
device = 'cuda' if cuda.is_available() else 'cpu'
print(device)

cuda

t5_tokenizer = T5Tokenizer.from_pretrained('t5-small')
t5_model = T5ForConditionalGeneration.from_pretrained('t5-small')

#Shift the model to GPU from CPU
t5_model.to('cuda')
```

2) Code and Output for Translation task:

```
text = """
that is good
"""

preprocess_text = text.strip().replace("\n","")
t5_prepared_Text = "translate English to German: "+preprocess_text
print ("original text preprocessed: \n", preprocess_text)

tokenized_text = t5_tokenizer.encode(t5_prepared_Text, return_tensors="pt").to(device)
summary_ids = t5_model.generate(tokenized_text,
                                num_beams=4,
                                no_repeat_ngram_size=2,
                                min_length=2,
                                max_length=5,
                                early_stopping=True)

output = t5_tokenizer.decode(summary_ids[0], skip_special_tokens=True)
print ("\n\nTranslated text: \n",output)

original text preprocessed:
that is good

Translated text:
Das ist gut,
```

3) Code and Output for Summarization task:

```
preprocess_text = text.strip().replace("\n","")
t5_prepared_Text = "summarize: "+preprocess_text
print ("original text preprocessed: \n", preprocess_text)

tokenized_text = t5_tokenizer.encode(t5_prepared_Text, return_tensors="pt").to(device)
summary_ids = t5_model.generate(tokenized_text,
                                num_beams=4,
                                no_repeat_ngram_size=2,
                                min_length=30,
                                max_length=100,
                                early_stopping=True)

output = t5_tokenizer.decode(summary_ids[0], skip_special_tokens=True)
print ("\n\nSummarized text: \n",output)
```

original text preprocessed:

The US has "passed the peak" on new coronavirus cases, President Donald Trump said and predicted that some states would reopen this month. The US has over 637,000 confirmed Covid-19 cases and over 30,826 deaths, the highest for any country in the world. At the daily White House coronavirus briefing on Wednesday, Trump said new guidelines to reopen the country would be announced on Thursday after he speaks to governors. "We'll be the comeback kids, all of us," he said. "We want to get our country back." The Trump administration has previously fixed May 1 as a possible date to reopen the world's largest economy, but the president said some states may be able to return to normalcy earlier than that.

Summarized text:

the us has over 637,000 confirmed Covid-19 cases and over 30,826 deaths. president Donald Trump predicts some states will reopen the country in april, he said. "we'll be the comeback kids, all of us," the president says.

3) Fine Tuning T5

We perform fine tuning on the **T5-Small** (60 million parameters) model for tasks of Sentiment Analysis and Emotion Classification which are binary and multi-class classification tasks respectively.

	Name	Type	Params
0	model	T5ForConditionalGeneration	60 M
1	model.shared	Embedding	16 M
2	model.encoder	T5Stack	35 M
3	model.encoder.block	ModuleList	18 M
4	model.encoder.block.0	T5Block	3 M
5	model.encoder.block.0.layer	ModuleList	3 M
6	model.encoder.block.0.layer.0	T5LayerSelfAttention	1 M
7	model.encoder.block.0.layer.0.SelfAttention	T5Attention	1 M
8	model.encoder.block.0.layer.0.SelfAttention.q	Linear	262 K
9	model.encoder.block.0.layer.0.SelfAttention.k	Linear	262 K
10	model.encoder.block.0.layer.0.SelfAttention.v	Linear	262 K
11	model.encoder.block.0.layer.0.SelfAttention.o	Linear	262 K
12	model.encoder.block.0.layer.0.SelfAttention.relative_attention_bias	Embedding	256
13	model.encoder.block.0.layer.0.layer_norm	T5LayerNorm	512
14	model.encoder.block.0.layer.0.dropout	Dropout	0
15	model.encoder.block.0.layer.1	T5LayerFF	2 M
16	model.encoder.block.0.layer.1.DenseReluDense	T5DenseReluDense	2 M
17	model.encoder.block.0.layer.1.DenseReluDense.wi	Linear	1 M
18	model.encoder.block.0.layer.1.DenseReluDense.wo	Linear	1 M
19	model.encoder.block.0.layer.1.DenseReluDense.dropout	Dropout	0
20	model.encoder.block.0.layer.1.layer_norm	T5LayerNorm	512
21	model.encoder.block.0.layer.1.dropout	Dropout	0
22	model.encoder.block.1	T5Block	3 M
23	model.encoder.block.1.layer	ModuleList	3 M
24	model.encoder.block.1.layer.0	T5LayerSelfAttention	1 M
25	model.encoder.block.1.layer.0.SelfAttention	T5Attention	1 M
26	model.encoder.block.1.layer.0.SelfAttention.q	Linear	262 K
27	model.encoder.block.1.layer.0.SelfAttention.k	Linear	262 K
28	model.encoder.block.1.layer.0.SelfAttention.v	Linear	262 K
29	model.encoder.block.1.layer.0.SelfAttention.o	Linear	262 K
30	model.encoder.block.1.layer.0.layer_norm	T5LayerNorm	512
31	model.encoder.block.1.layer.0.dropout	Dropout	0

```

args_dict = dict(
    data_dir="", # path for data files
    output_dir="", # path to save the checkpoints
    model_name_or_path='t5-small',
    tokenizer_name_or_path='t5-small',
    max_seq_length=512,
    learning_rate=3e-4,
    weight_decay=0.0,
    adam_epsilon=1e-8,
    warmup_steps=0,
    train_batch_size=8,
    eval_batch_size=8,
    num_train_epochs=2,
    gradient_accumulation_steps=16,
    n_gpu=1,
    early_stop_callback=False,
    fp_16=False, # if you want to enable 16-bit training then install a
    opt_level='O1', # you can find out more on optimisation levels here
    max_grad_norm=1.0, # if you enable 16-bit training then set this to
    seed=21,
)

```

Model Layers

Fine Tuning Arguments

4) Leveraging a Single T5 model for multiple Downstream tasks

Task 1 : Sentiment Analysis

In sentiment analysis we finetune the model of the Imdb review classification dataset where the task for the model is to output “positive” or “negative” based on the given review as input. For models like BERT we would have to add a new layer at the end of existing BERT model to have it perform binary classification task while in T5, since it’s a text-to-text model, we can just pose the task a string prediction problem with positive and negative being the two string options. No

need to change any hyperparameters, loss function for different NLP tasks - the only thing you need to change is the dataset. This makes finetuning T5 much easier.

```
for i in range(32):
    lines = textwrap.wrap("Review:\n%s\n" % texts[i], width=100)
    print("\n".join(lines))
    print("\nActual sentiment: %s" % targets[i])
    print("Predicted sentiment: %s" % dec[i])
    print("=====")

Actual sentiment: positive
Predicted sentiment: positive
=====

Review: After waiting years for a definitive collection of Led Zeppelin performances on video fans
have finally been rewarded with what is undoubtedly the greatest concert video ever Much better than
the dismal Song Remains the Same this video includes performances from no less than 5 different
venues spanning a decade It also includes rare interviews and TV appearances The sound quality is
amazing considering the source material used And the video quality is even more impressive This is
an ABSOLUTE MUST for any Led Zeppelin fan

Actual sentiment: positive
Predicted sentiment: positive
=====

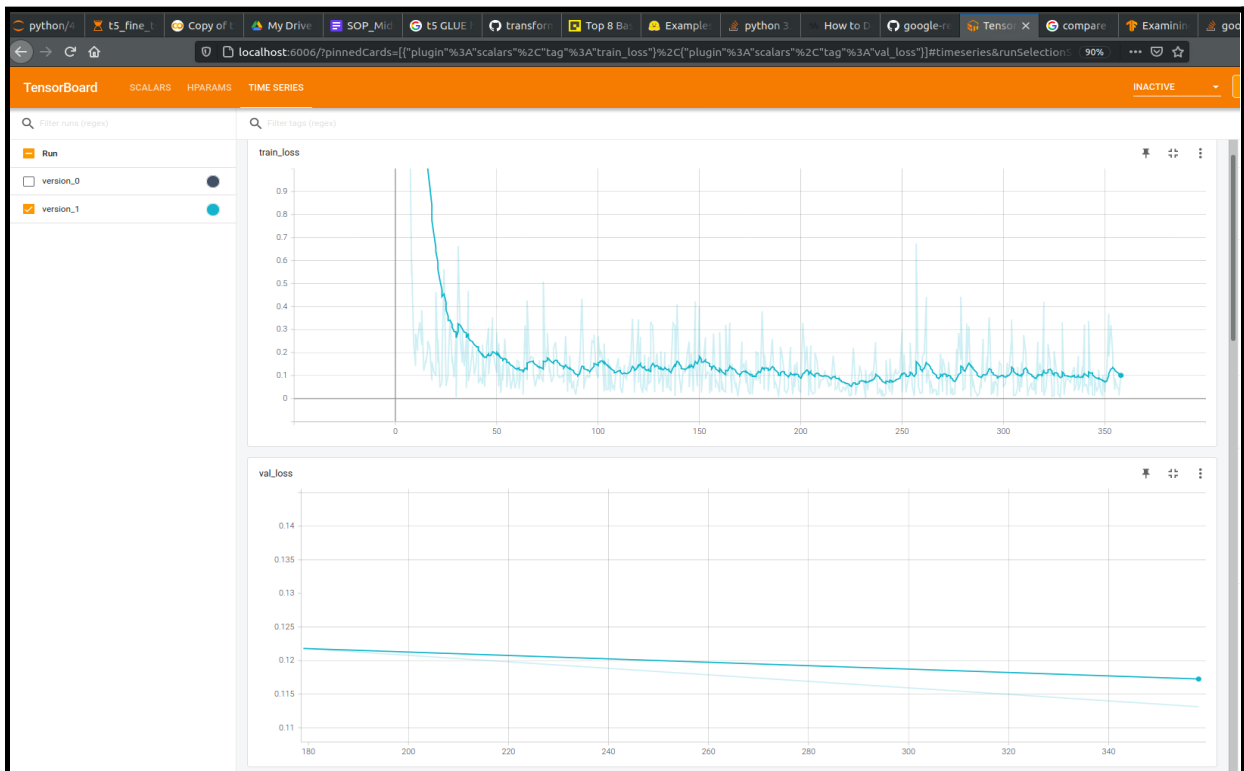
Review: This is an enjoyable project that is not a film as the title suggests Good performance by
```

```
print(metrics.classification_report(targets, outputs))
```

	precision	recall	f1-score	support
negative	0.93	0.91	0.92	12500
positive	0.91	0.93	0.92	12500
accuracy			0.92	25000
macro avg	0.92	0.92	0.92	25000
weighted avg	0.92	0.92	0.92	25000

Model Predictions vs Actual

Metrics



Training and Validation Loss

Task 2 : Emotion Classification

Moving from a binary classification task to multi-class classification task, we finetune the T5 model on this [dataset](#).

```
for i in range(32):
    c = texts[i]
    lines = textwrap.wrap("text:\n%s\n" % c, width=100)
    print("\n".join(lines))
    print("\nActual sentiment: %s" % targets[i])
    print("predicted sentiment: %s" % dec[i])
    print("=====")

Actual sentiment: joy
predicted sentiment: joy
=====

text: i am still feeling a tad strange in those pearly whites

Actual sentiment: surprise
predicted sentiment: surprise
=====

text: i have been learning and re learning the lesson that no matter how i feel about myself or even
how others may feel about me i am treasured by god

Actual sentiment: love
predicted sentiment: joy
```

Dataset format and Actual/Predicted Comparison



Confusion Matrix for different emotions

As we can see from the results obtained from the previous 2 tasks, the same trainer function with a modification in the dataset can help us finetune the t5 small model on any text-to-text NLP task.

These both tasks were Sequence-to-Vector tasks and while I did want to test the model fine tuning on Sequence-to-Sequence tasks like Summarization or Translation, due to the humongous size of the model and limited hardware constraints, I was only able to finetune on these mentioned tasks.

Conclusion

The T5 model shows a lot of potential in terms of a universal pre trained model for all downstream NLP tasks. Just by changing the dataset and marking the input and target appropriately, we can finetune the T5 model for most NLP tasks if modelled in text-to-text format. It performs very well on classification tasks showing near human accuracy using the smallest variant of the available models (t5-small with 60 Mn parameters) and only finetuning for 2 epochs. Using the base model which has 11Bn parameters would yield results proportional to its size but to train or even finetune such a model would require a staggering amount of GPUs or TPUs.

Metrics

```
print(metrics.classification_report(targets, outputs))
```

	precision	recall	f1-score	support
negative	0.93	0.91	0.92	12500
positive	0.91	0.93	0.92	12500
accuracy			0.92	25000
macro avg	0.92	0.92	0.92	25000
weighted avg	0.92	0.92	0.92	25000

IMDB Review Classification

```
print(metrics.classification_report(targets, outputs))
```

	precision	recall	f1-score	support
anger	0.88	0.85	0.86	275
fear	0.88	0.87	0.88	224
joy	0.91	0.92	0.92	695
love	0.79	0.70	0.74	159
sadness	0.93	0.94	0.93	581
surprise	0.68	0.80	0.74	66
accuracy			0.89	2000
macro avg	0.84	0.85	0.85	2000
weighted avg	0.89	0.89	0.89	2000

Emotion Classification

This huge size is a problem since it limits the commercial usage of the model due to high memory requirements and also inference time. Methods like knowledge distillation might be helpful in making such models smaller but they often come with a hit on accuracy in exchange for smaller size.

Major Learnings

Through the course of this project, I was able to learn many things. Due to the very fast pace of research in the Machine learning and Artificial Intelligence domain, new and better models are coming faster than ever.

Currently, the trend in NLP is to train a very large model and then finetune it according to the requirement of the downstream task. I was able to learn how State-of-the-Art models like BERT, GPT and T5 have been developed, trained and are being used today. I also got to learn to finetune a T5 model on any specific task using the pytorch-lightning and transformers library. I was able to finetune and evaluate the t5-small model on tasks like Sentiment Analysis and Emotion Classification.

Apart from all this I also got to know how authors build upon existing research, introducing novelties of their own which end up being ingenious sometimes leading to astonishing results like with T5 model.

Limitations

Due to the humongous size of the T5 model, I faced a number of challenges in the implementation of this model for fine tuning and pretraining.

1) Pretraining

- Pretraining this model is very difficult unless you have TPU clusters in hundreds. Even on Google colab with a 24 GB Nvidia Titan GPU, I could barely train a T5-small model which is around 0.5% of the T5 base model. This helps to put into perspective how huge the model and its computation cost is

2) Fine Tuning

- Even for fine tuning, any model other than T5 small gave CUDA Out of Memory for even the smallest batch size on a 24 GB GPU. This limited the finetuning to Sequence-to-Vector tasks

3) Google Colab

- Since I only had a 4GB GPU on my local system, it wouldn't have been possible to train or even finetune the smallest T5 model on it. Training on Colab had its own issues where the runtime would disconnect abruptly making me lose hours of training. The only solution was to train for a less number of epochs with small batch sizes

Future Scope

The ease of training this model for a variety of NLP tasks in text-to-text format comes at the cost of a large number of parameters thus making it difficult to train and finetune.

In this project I was able to explore the model's performance on sequence-to-vector tasks and even with minimal training it was performing very well. With larger variants of the model along with finetuning for more number of epochs would result in even better results. The authors and creators of this model boast good results on sequence-to-sequence tasks also which we have explored and confirmed by carrying out Summarization and Translation using the pretrained t5-small model.

But the need of the hour is to come up with techniques to increase the performance/size ratio as much we can. The results of this such models have started to approach human level performance, but to use them requires a lot of hardware accelerators which is not a feasible solution for many research and businesses which could be benefiting from this innovation.

List of References

1) BERT

- a) [\[1810.04805\] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)
- b) [BERT Explained: State of the art language model for NLP](#)
- c) [\[1506.06724\] Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books](#)

2) GPT

- a) [\[2005.14165\] Language Models are Few-Shot Learners](#)
- b) [The Illustrated GPT-2 \(Visualizing Transformer Language Models\) – Jay Alammar – Visualizing machine learning one concept at a time.](#)
- c) [OpenAI “Better Language Models and Their Implications”](#)

3) T5

- a) [Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer](#)
- b) [Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer](#)
- c) [T5 — a model that explores the limits of transfer learning](#)
- d) [t5-small · Hugging Face](#)
- e) [c4](#)

4) Other references

- a) [GLUE Leaderboard](#)
- b) [Pretrained models — transformers 4.5.0.dev0 documentation](#)
- c) [Fine Tuning T5 Transformer Model with PyTorch](#)
- d) [mrm8488/t5-base-finetuned-imdb-sentiment · Hugging Face](#)
- e) [PyTorchLightning/pytorch-lightning: The lightweight PyTorch wrapper for high-performance AI research. Scale your models, not the boilerplate.](#)

Appendix

Link for complete code : [T5 Finetuning](#)

1) Fine Tuning Pseudo Code

```
class T5FineTuner(pl.LightningModule):
    def __init__(self, hparams):
        super(T5FineTuner, self).__init__()
        self.hparams = hparams

        self.model = T5ForConditionalGeneration.from_pretrained(hparams.model_name_or_path)
        self.tokenizer = T5Tokenizer.from_pretrained(hparams.tokenizer_name_or_path)

    def is_logger(self):
        return self.trainer.proc_rank <= 0

    def forward(
        self, input_ids, attention_mask=None, decoder_input_ids=None, decoder_attention_mask=None, lm_labels=None
    ):
        return self.model(
            input_ids,
            attention_mask=attention_mask,
            decoder_input_ids=decoder_input_ids,
            decoder_attention_mask=decoder_attention_mask,
            lm_labels=lm_labels,
        )

    def _step(self, batch):
        lm_labels = batch["target_ids"]
        lm_labels[lm_labels[:, :] == self.tokenizer.pad_token_id] = -100

        outputs = self(
            input_ids=batch["source_ids"],
            attention_mask=batch["source_mask"],
            lm_labels=lm_labels,
            decoder_attention_mask=batch['target_mask']
        )

        loss = outputs[0]
```

```

    return loss

def training_step(self, batch, batch_idx):
    loss = self._step(batch)

    tensorboard_logs = {"train_loss": loss}
    return {"loss": loss, "log": tensorboard_logs}

def training_epoch_end(self, outputs):
    avg_train_loss = torch.stack([x["loss"] for x in outputs]).mean()
    tensorboard_logs = {"avg_train_loss": avg_train_loss}
    return {"avg_train_loss": avg_train_loss, "log": tensorboard_logs, 'progress_bar': tensorboard_logs}

def validation_step(self, batch, batch_idx):
    loss = self._step(batch)
    return {"val_loss": loss}

def validation_epoch_end(self, outputs):
    avg_loss = torch.stack([x["val_loss"] for x in outputs]).mean()
    tensorboard_logs = {"val_loss": avg_loss}
    return {"avg_val_loss": avg_loss, "log": tensorboard_logs, 'progress_bar': tensorboard_logs}

def configure_optimizers(self):
    "Prepare optimizer and schedule (linear warmup and decay)"

    model = self.model
    no_decay = ["bias", "LayerNorm.weight"]
    optimizer_grouped_parameters = [
        {
            "params": [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)],
            "weight_decay": self.hparams.weight_decay,
        },
        {
            "params": [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)],
            "weight_decay": 0.0,
        },
    ]
    optimizer = AdamW(optimizer_grouped_parameters, lr=self.hparams.learning_rate, eps=self.hparams.adam_epsilon)
    self.opt = optimizer
    return [optimizer]

def optimizer_step(self, epoch, batch_idx, optimizer, optimizer_idx, second_order_closure=None):
    if self.trainer.use_tpu:
        xm.optimizer_step(optimizer)

```

```

else:
    optimizer.step()
    optimizer.zero_grad()
    self.lr_scheduler.step()

def get_tqdm_dict(self):
    tqdm_dict = {"loss": "{:.3f}".format(self.trainer.avg_loss), "lr": self.lr_scheduler.get_last_lr()[-1]}

    return tqdm_dict

def train_dataloader(self):
    train_dataset = get_dataset(tokenizer=self.tokenizer, type_path="train", args=self.hparams)
    dataloader = DataLoader(train_dataset, batch_size=self.hparams.train_batch_size, drop_last=True, shuffle=True,
num_workers=4)
    t_total = (
        (len(dataloader.dataset) // (self.hparams.train_batch_size * max(1, self.hparams.n_gpu)))
        // self.hparams.gradient_accumulation_steps
        * float(self.hparams.num_train_epochs)
    )
    scheduler = get_linear_schedule_with_warmup(
        self.opt, num_warmup_steps=self.hparams.warmup_steps, num_training_steps=t_total
    )
    self.lr_scheduler = scheduler
    return dataloader

def val_dataloader(self):
    val_dataset = get_dataset(tokenizer=self.tokenizer, type_path="val", args=self.hparams)
    return DataLoader(val_dataset, batch_size=self.hparams.eval_batch_size, num_workers=4)

```

2) Model Arguments

```

checkpoint_callback = pl.callbacks.ModelCheckpoint(
    filepath=args.output_dir, prefix="checkpoint", monitor="val_loss", mode="min", save_top_k=5)

train_params = dict(
    accumulate_grad_batches=args.gradient_accumulation_steps,
    gpus=args.n_gpu,
    max_epochs=args.num_train_epochs,
    early_stop_callback=False,
    precision= 16 if args.fp_16 else 32,
    amp_level=args.opt_level,
    gradient_clip_val=args.max_grad_norm,
    checkpoint_callback=checkpoint_callback,
    callbacks=[LoggingCallback()])

```