TRANSFORMER MODELS

Natural Language Processing

What is NLP?

NLP は、人間の言語に関する全てのものを理解することに焦点を当てた言語学と機械学習の分野である。
NLPタスクの目的は、1つの単語を個別に理解するだけでなく、それらの単語の文脈を理解できるようになることである。

NLPタスクの一例

- 文全体の分類
  - レビューの感情の取得
  - メールがスパムかどうかの検出
  - 文が文法的に正しいかどうか
  - 2つの文が論理的に関連しているかどうか
- 文中の各単語の分類
  - 文の文法要素(名詞、動詞、形容詞)
  - 名前付き属性(人、場所、組織)の特定
- テキストコンテンツの生成
  - 自動生成されたテキストでプロンプトを完了し、マスクされた単語でテキストの空白を埋める。
- テキストからの回答の抽出
  - 質問とコンテキストが与えられたとき、コンテキストで提供された情報に基づいて質問に対する回答を抽出する。
- 入力テキストからの新しい文の生成
  - テキストの翻訳
  - 要約

ただし、NLP は書かれたテキストに限定されない。また、音声サンプルの複写や画像の説明の生成など、音声認識と CV の複雑な課題にも取り組む。

Why is it challenging

コンピュータは人間と同じように情報を処理するわけではない。
例えば、「お腹がすいた」という文を読むと、その意味が簡単に理解できる。同様に、「お腹がすいた」と「悲しい」などの2つの文がある場合、それらがどの程度似ているかを簡単に判断できる。機械学習モデルの場合、そのようなタスクは困難である。テキストはモデルがテキストから学習できるように処理する必要がある。また、言語は複雑であるため、この処理をどのように行う必要があるかを慎重に考える必要がある。

Transformers, what can they do?

Working with pipelines

Transformer ライブラリの最も基本的なオブジェクトは `pipeline()` 関数である。モデルと必要な前処理および後処理のステップを結びつけ、任意のテキストを直接入力してわかりやすい答えを得ることを可能にする。

PYTHON

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a HuggingFace course my whole
life.")
```

PYTHON

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

複数の文章を渡すことも可能である。

PYTHON

```
classifier(
    ["I've been waiting for a HuggingFace course my whole life.", "I
hate this so much!"]
)
```

PYTHON

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437},
 {'label': 'NEGATIVE', 'score': 0.9994558095932007}]
```

デフォルトでは、このパイプラインは、英語の感情分析用に fine-tuned された特定の事前学習済みモデルを選択する。モデルは `classifier` オブジェクトの作成時にダウンロードされ、キャッシュされる。コマンドを再実行すると、キャッシュされたモデルが変わりに使用され、モデルを再度ダウンロードする必要はない。

パイプラインにテキストを渡す場合は、主に3つの手順が必要である。

1. テキストは、モデルが理解できる形式に前処理される。
2. 前処理されたにゅうりょくがモデルに渡される。
3. モデルの予測は後処理されるので、それらを理解できる。

使用可能なパイプラインの一部

- `feature-extraction`
- `fill-mask`
- `ner` (固有表現認識)
- `question-answering`
- `sentiment-analysis`
- `summarization`
- `text-generation`
- `translation`

- `zero-shot-classification`

Zero-shot classification

より困難なタスクである、ラベルづけされていないテキストを分類するタスクに取り組むことを始める。テキストの注釈付には通常時間がかかり、ドメインの専門知識が必要なため、実際のプロジェクトでは一般なシナリオである。このユースケースでは、`zero-shot-classification` パイプラインは非常に強力である。分類に使用するラベルを指定できるため、事前学習済みモデルのラベルに頼る必要はない。モデルが文を positve, negative に分類する方法は既に見てきたが、他のラベルセットを用いてテキストを分類することもできる。

PYTHON

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification")
classifier(
    "This is a course about the Transformers library",
    candidate_labels=["education", "politics", "business"],
)
```

PYTHON

```
{'sequence': 'This is a course about the Transformers library',
 'labels': ['education', 'business', 'politics'],
 'scores': [0.8445963859558105, 0.111976258456707,
0.043427448719739914]}
```

このパイプラインは、データを使用するためにモデルの fine-tune を必要としないので、*zero-shot* と呼ばれる。

Text generation

次に、パイプラインを使用してテキストを生成する方法を見る。ここでの主な考え方は、プロンプトを提供すると、モデルが残りのテキストを生成してプロンプトを自動補完することである。これは多くの携帯電話に見えられる予測テキスト機能に似ている。テキスト生成にはランダム性が伴うので、以下に示すように同じ結果が得られないのは正常である。

PYTHON

```
from transformers import pipeline

generator = pipeline("text-generation")
generator("In this course, we will teach you how to")
```

PYTHON

```
[{'generated_text': 'In this course, we will teach you how to
understand and use '
                    'data flow and data interchange when handling
user data. We '
                    'will be working with one or more of the most
commonly used '
                    'data flows — data flows of various types, as
seen by the '
                    'HTTP'}]
```

Using any model from the Hub in a pipeline

前の例では、手元のタスクの既定のモデルを使用したが、Hub から特定のモデルを選択して、特定のタスク(テキスト生成など)のパイプラインで使用することもできる。
distilgpt2 モデルを試してみる。

PYTHON

```
from transformers import pipeline

generator = pipeline("text-generation", model="distilgpt2")
generator(
    "In this course, we will teach you how to",
    max_length=30,
    num_return_sequences=2,
)
```

PYTHON

```
[{'generated_text': 'In this course, we will teach you how to
manipulate the world and '
                    'move your mental and physical capabilities to
your advantage.'},
 {'generated_text': 'In this course, we will teach you how to become
an expert and '
                    'practice realtime, and with a hands on
experience on both real '
                    'time and real'}]
```

The Inference API

全てのモデルは、Hugging face の Web サイトで利用できる Inference API を使用して、ブラウザから直接テストできる。

Mask filling

次に試すパイプラインは `fill-mask` である。このタスクの考え方は、特定のテキストを空白で埋めることである。

PYTHON

```
from transformers import pipeline

unmasker = pipeline("fill-mask")
unmasker("This course will teach you all about <mask> models.",
top_k=2)
```

PYTHON

```
[{'sequence': 'This course will teach you all about mathematical
models.',
  'score': 0.19619831442832947,
  'token': 30412,
  'token_str': ' mathematical'},
 {'sequence': 'This course will teach you all about computational
models.',
  'score': 0.04052725434303284,
  'token': 38163,
  'token_str': ' computational'}]
```

`top_k` の引数は、表示する可能性の数を制御する。ここでは、モデルが特別な単語 `<mask>` (*mask_token* と呼ばれることが多い)を埋めることであることに注意する。他のマスク充填モデルではマスクトークンが異なる場合があるため、他のモデルを探索するときは常に適切なマスクワードを確認することを推奨する、それを確認する1つの方法は、ウィジェットで使用されているマスクワードを確認することである。

Name entity recognition

固有表現認識(NER)は、モデルが入力テキストのどの部分が人、場所、組織などのエンティティに対応するかを見つけるタスクである。

PYTHON

```
from transformers import pipeline

ner = pipeline("ner", grouped_entities=True)
ner("My name is Sylvain and I work at Hugging Face in Brooklyn.")
```

PYTHON

```
[{'entity_group': 'PER', 'score': 0.99816, 'word': 'Sylvain',
'start': 11, 'end': 18},
 {'entity_group': 'ORG', 'score': 0.97960, 'word': 'Hugging Face',
'start': 33, 'end': 45},
 {'entity_group': 'LOC', 'score': 0.99321, 'word': 'Brooklyn',
'start': 49, 'end': 57}
]
```

上記の例では、Sylvain が人(PER)、Hugging face が組織(ORG)、Brooklyn が場所(LOC)であることを正しく識別した。
パイプライン作成関数に `grouped_entities=True` オプションを渡して、同じエンティティに対応する文の部分を最グループ化するようにパイプラインに指示する。これによって、名前が複数の単語で構成されていても正しく認識してくれる。

Question answering

`question-answwering` パイプラインは、特定のコンテキストからの情報を使用して質問に答える。

PYTHON

```
from transformers import pipeline

question_answerer = pipeline("question-answering")
question_answerer(
    question="Where do I work?",
    context="My name is Sylvain and I work at Hugging Face in
Brooklyn",
)
```

PYTHON

```
{'score': 0.6385916471481323, 'start': 33, 'end': 45, 'answer':
'Hugging Face'}
```

このパイプラインは、指定されたコンテキストから情報を抽出することで機能することに注意する。答えは生成されない。

Summarization

要約とは、テキストで参照されている重要な側面の全て(またはほとんど)を保持しながら、テキストを短いテキストに縮小するタスクである。

PYTHON

```python
from transformers import pipeline

summarizer = pipeline("summarization")
summarizer(
    """
    America has changed dramatically during recent years. Not only has the number of
    graduates in traditional engineering disciplines such as mechanical, civil,
    electrical, chemical, and aeronautical engineering declined, but in most of
    the premier American universities engineering curricula now concentrate on
    and encourage largely the study of engineering science. As a result, there
    are declining offerings in engineering subjects dealing with infrastructure,
    the environment, and related issues, and greater concentration on high
    technology subjects, largely supporting increasingly complex scientific
    developments. While the latter is important, it should not be at the expense
    of more traditional engineering.

    Rapidly developing economies such as China and India, as well as other
    industrial countries in Europe and Asia, continue to encourage and advance
    the teaching of engineering. Both China and India, respectively, graduate
    six and eight times as many traditional engineers as does the United States.
    Other industrial countries at minimum maintain their output, while America
    suffers an increasingly serious decline in the number of engineering graduates
    and a lack of well-educated engineers.
"""
)
```

```
PYTHON
[{'summary_text': ' America has changed dramatically during recent
years . The '
                  'number of engineering graduates in the U.S. has
declined in '
                  'traditional engineering disciplines such as
mechanical, civil '
                  ', electrical, chemical, and aeronautical
engineering . Rapidly '
                  'developing economies such as China and India, as
well as other '
                  'industrial countries in Europe and Asia, continue
to encourage '
                  'and advance engineering .'}]
```

テキスト生成と同様に、結果に `max_length` と `min_length` を指定できる。

Translation

翻訳では、タスク名に言語ペア(`translation_en_to_fr`)を指定するとデフォルトのモデルを使用できるが、最も簡単な方法は Model Hub で使用するモデルを選択することである。

```
PYTHON
from transformers import pipeline

translator = pipeline("translation", model="Helsinki-NLP/opus-mt-fr-
en")
translator("Ce cours est produit par Hugging Face.")
```

```
PYTHON
[{'translation_text': 'This course is produced by Hugging Face.'}]
```

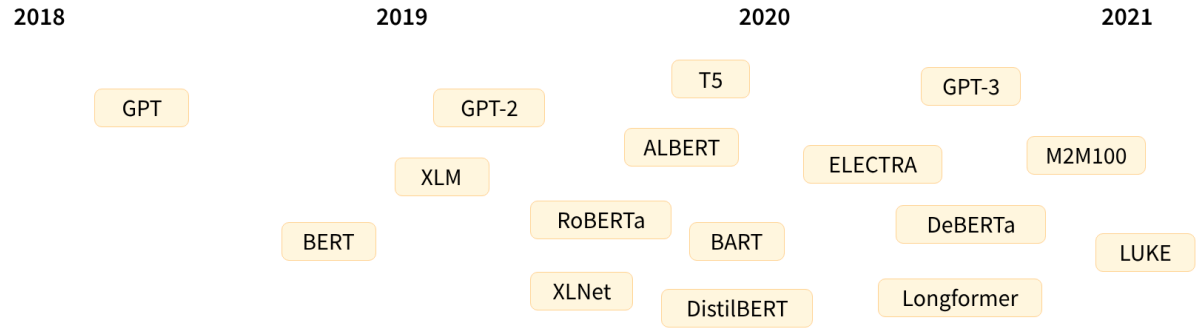
テキスト生成と同様に、結果に `max_length` と `min_length` を指定できる。

これまでに示したパイプラインは、ほとんどがデモ用である。それらは特定のタスク用にプログラムされており、それらのバリエーションを実行することはできない。次の章では `pipeline()` 関数の内部と、その動作をカスタマイズする方法について説明する。

How do Transformers work?

A bit of Transformer history

Transformer のアーキテクチャは 2017年6月に導入された。当初の研究の焦点は翻訳タスクにある。これに続いて、次のようないくつかの影響力のあるモデルが導入された。



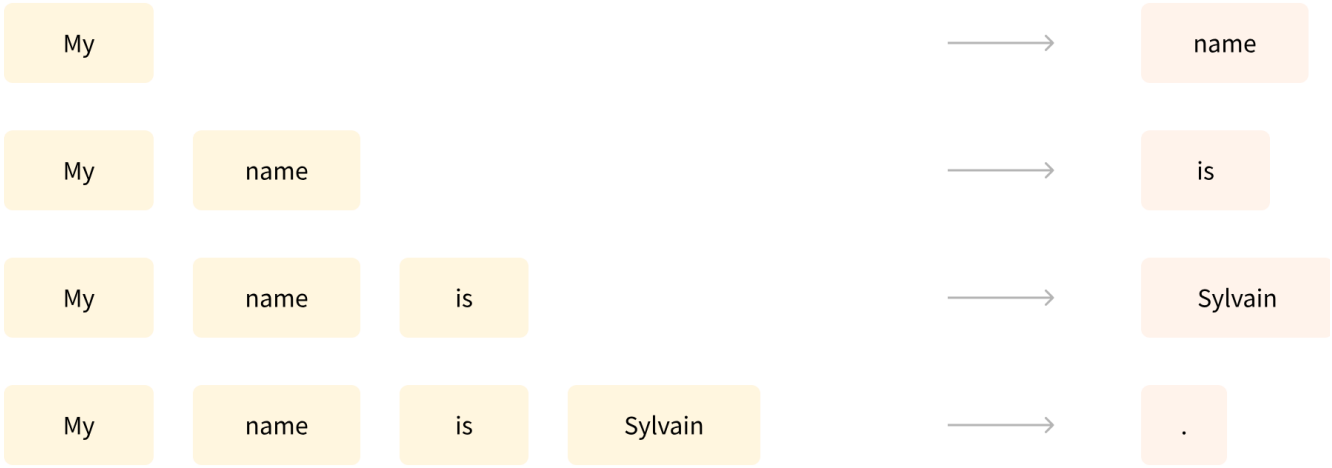
- June 2018
  - GPT は様々な NLP の fine-tune に使用され、最先端の結果を得た最初の事前学習済み Transfomrer モデルである。
- October 2018
  - BERT は、別の大規模な事前学習済みモデルで、これは文のより良い要約を生成するように設計されている。
- February 2019
  - GPT-2 は GPT の改良版だが、倫理的な懸念からすぐには公開されなかった。
- October 2019
  - DistilBERT は BERT の蒸留バージョンで、60%高速、メモリ40%軽量でありながら、BERT のパフォーマンスの 97%を保持している。
  - BART と T5 は、元の Transformer モデルと同じアーキテクチャを使用した2つの大規模な事前学習済みモデルである。
- May 2020
  - GPT-3 は、GPT-2 のさらに大きなバージョンで、fine-tune(ゼロショット学習とよばれる)を必要とせずに様々なタスクで優れたパフォーマンスを発揮することができる。

このリストは包括的なものではなく、様々な種類の Transformer モデルのいくつかを強調することを意図している。大まかに言うと、それらは3つのカテゴリに分類できる。

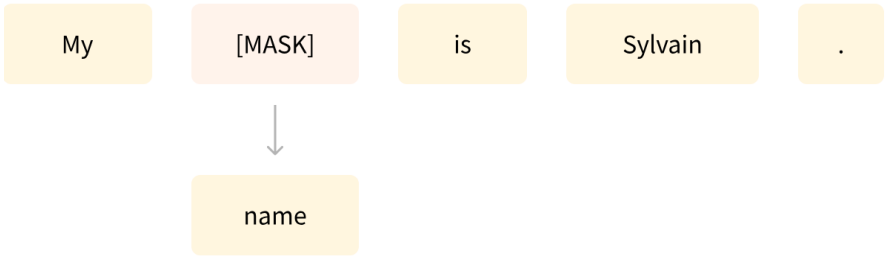
- GPT-like (also called *auto-regressive* Transformer models)
- BERT-like (also called *auto-encoding* Transformer models)
- BART/T5-like (also called *sequence-to-sequence* Transformer models)

Transformer are language models

上記の全ての Transformer モデル(GPT, BERT, BART, T5 など) は *言語モデル* として学習されている。これは、これらのモデルが自己教師ありの方法で大量の生のテキストで訓練されていることを意味する。自己教師あり学習は、モデルの入力から目標が自動的に計算される手法である。つまり、データにラベルをつけるのに人間は必要ない。
このタイプのモデルは、トレーニングされた言語の統計的理解を発展させるが、特定の実用的なタスクにはあまり役立たない。そのため、一般的な事前学習済みモデルは、*転移学習* とよばれるプロセスを経る。このプロセスでは、モデルは特定のタスクに対して教師ありの方法で(つまり、人間が注釈付けしたラベルを使用して)fine-tune される。
タスクの例としては、n 個前の単語を読んだ文の次の単語を予測する事がある。これが *causal language modeling* と呼ばれるのは、出力が過去と現在の入力に依存し、将来の入力には依存しないためである。

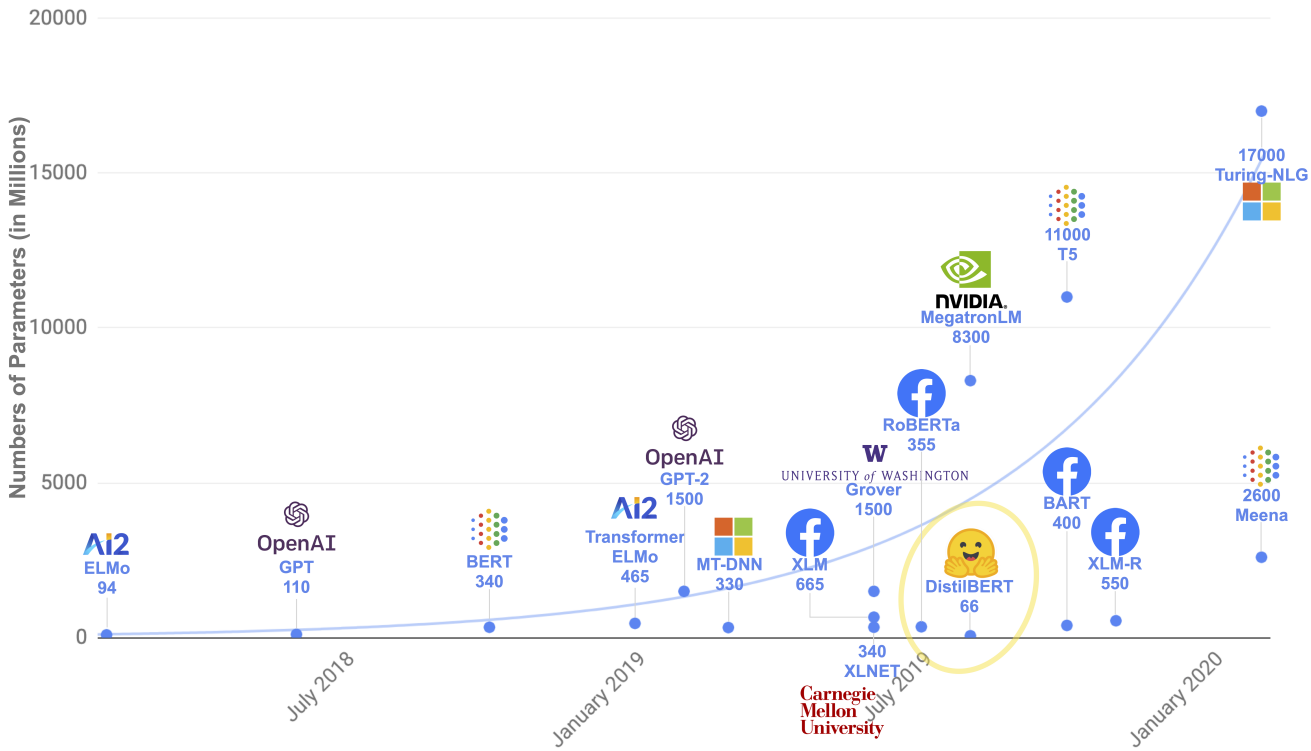


もう一つの例は、モデルが文中のマスクされた単語を予測する *masked language modeling* である。

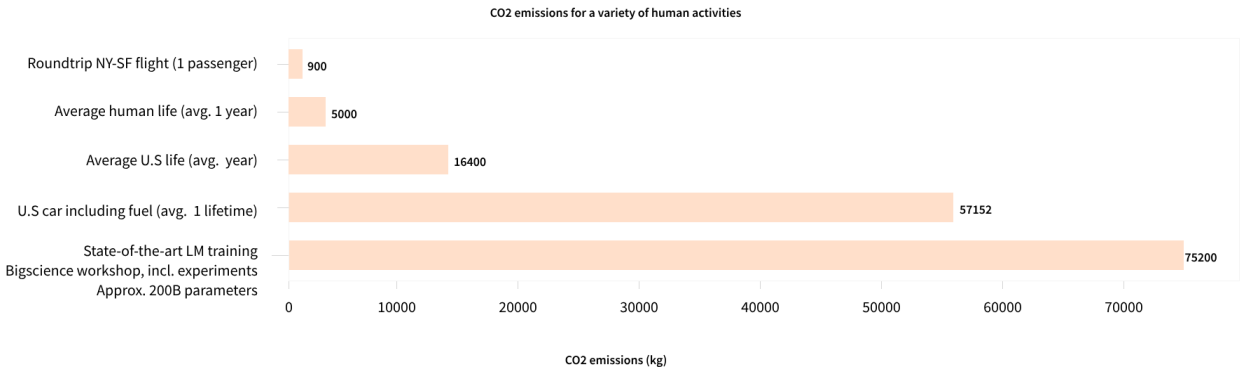


Transformers are big models

DistilBERT のようないくつかの例外を除けば、性能を向上させるための一般的な戦略は、モデルのサイズと事前学習の対象となるデータ量を増やすことである。



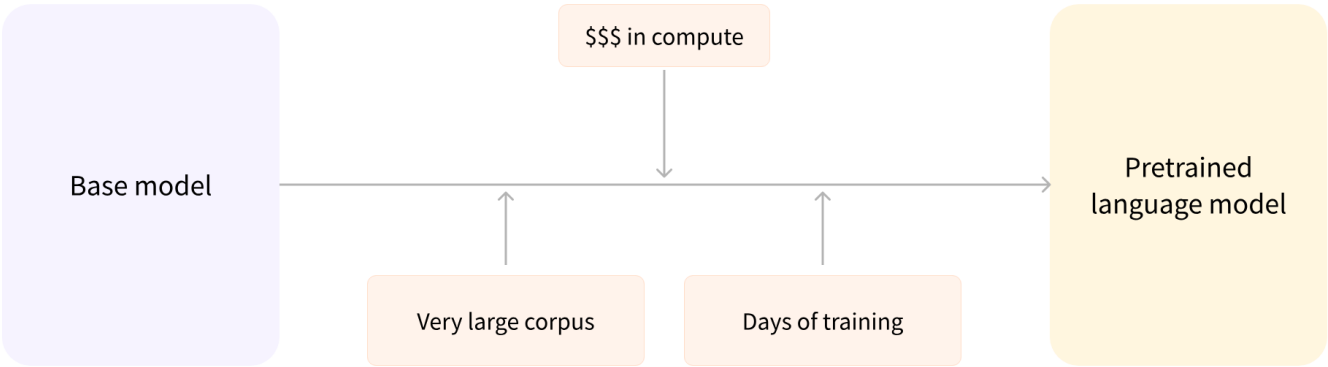
残念ながら、モデル(特に大規模なモデル)の訓練には大量のデータが必要である。これは、時間と計算資源の点で非常にコストがかかる。それは環境への影響とも言いかえられる。



研究チーム、学生団体、企業がモデルを訓練するたびに、ゼロから訓練を行うと、莫大で不必要なグローバルコストになる。
これが言語モデルの共有が最も重要である理由だ。学習済みの重みを共有し、既に訓練済みの重みに基づいて構築することで、コミュニティの全体的な計算コストと二酸化炭素排出力が削減される。

Transfer Learning

事前学習 とは、モデルをゼロから学習する行為である。重みはランダムに初期化され、事前の知識がなくても学習が開始される。



この事前学習は、通常、日以上に大量のデータに対して行われる。そのため、非常に大規模なデータコーパスが必要であり、訓練には数週間かかる場合がある。
一方、*fine-tuning* は、モデルが事前学習された**後に**行われる訓練である。fine-tuning を実行するには、まず事前学習済みの言語モデルを取得し、次にタスクに固有のデータセットを使用して追加の学習を実行する。

なぜゼロから訓練しない?

- 事前学習済みモデルは、fine-tuning データセットといくつかの類似点があるデータセットで既に訓練されている。したがって、fine-tuning の過程では、事前学習中に初期モデルによって取得された知識を活用できる。
  - たとえば、NLP の問題の場合、事前学習済みモデルは、タスクに使用している言語について何らかの統計的理解を持っている。
- 事前学習済みモデルは既に大量のデータで学習されているため、適切な結果を得るための fine-tuning に必要なデータが少なくて済む。
- 同じ理由で、良い結果を得るために必要な時間とリソースの量は遥かに少なくなる。