*Natural Language Engineering* (2022), **28**, pp. 519–535
doi:10.1017/S1351324922000237

CAMBRIDGE UNIVERSITY PRESS

EMERGING TRENDS

# Emerging trends: General fine-tuning (*gft*)

Kenneth Ward Church[1,*], Xingyu Cai[1], Yibiao Ying[2], Zeyu Chen[2], Guangxu Xun[1] and Yuchen Bian[1]

[1]Baidu, Sunnyvale, CA, USA and [2]Baidu, Beijing, China
*Corresponding author. E-mail: Kenneth.Ward.Church@gmail.com

(Received 02 May 2022; revised 1 May 2022; first published online 23 May 2022)

**Abstract**

This paper describes *gft* (general fine-tuning), a little language for deep nets, introduced at an ACL-2022 tutorial. *gft* makes deep nets accessible to a broad audience including non-programmers. It is standard practice in many fields to use statistics packages such as R. One should not need to know how to program in order to fit a regression or classification model and to use the model to make predictions for novel inputs. With *gft*, fine-tuning and inference are similar to fit and predict in regression and classification. *gft* demystifies deep nets; no one would suggest that regression-like methods are "intelligent."

**Keywords:** Regression; Classification; Fine-tuning; Inference; Deep nets; Fit; Predict

## 1. Introduction

This paper introduces *gft* (general fine-tuning),[1] a little language[2] for deep nets, introduced at an ACL-2022 tutorial.[3] There are two parts to the tutorial:

1. Glass is half-full: make deep nets accessible to a mass audience, including nonprogrammers, and
2. Glass is half-empty: based on the successes of the first part on so many benchmarks, one might come to the mistaken impression that deep nets are more successful than they are. There are always opportunities for improvement. We are advocating an interdisciplinary approach that combines the successes in the first part, with decades of work in AI representation and centuries of work in linguistics and philosophy.

This paper will use *gft* to discuss the first part. It is amazing how much can be done with so little. *gft* demystifies deep nets. No one would suggest that regression-like methods are "intelligent."

There are two main functions in *gft*: *fit* and *predict*. Fit takes a pretrained model, $f_{pre}$, as input, and fine-tunes that on data to produce a post-trained model, $f_{post}$, as output. Predict takes $x$, a novel input, and predicts, $\hat{y} = f(x)$. Hopefully, the prediction, $\hat{y}$, will be close to the gold label, $y$.

We discussed deep nets in two previous articles in this journal: (Church *et al.*, 2021a, b). *gft* makes it possible to do much of that in short (1-line) programs. 1-line programs are easier to read,

[1]https://github.com/kwchurch/gft

[2]The term, *little languages*, is borrowed from Unix (Bentley, 1986). Programs in little languages such as AWK (Aho *et al.*, 1987) are short (often just a single line of code) and powerful.

[3]https://github.com/kwchurch/ACL2022_deepnets_tutorial

write, understand, and port from one environment to another than examples on hubs (typically hundreds of lines of Python, PyTorch,[4] TensorFlow,[5] Jax[6], and/or PaddlePaddle).[7]

*gft* is designed to make much of this functionality accessible to nonprogrammers. Just as one does not need to know Python and Machine Learning to use an off-the-shelf regression package, so too, deep nets should not require much (if any) programming skills.

Following the advice in "Crossing the Chasm" (Moore and McKenna, 1999), the long-term success of deep nets will depend on finding ways to cross the chasm from the current set of loyal users (so-called early adopters) to a much larger set of users. Early adopters may be willing to invest in machine learning and programming, but most users have other priorities.

The *gft* interpreter is based on examples from hubs.[8,9] Hubs encourage users to modify hundreds of lines of Python code as necessary if they want to change models, data sets, and/or tasks. *gft* generalizes the examples so users can do much of that in a single line of *gft* code (with comparable performance).

*gft* supports most of the arguments in the examples on the hubs, so it is possible to tune hyperparameters such as batch size, learning rate, and stopping rules. Tuning matters for (state of the art) SOTA-chasing, though default settings are recommended for most users who prefer results that are easy to replicate and reasonably competitive.

There is already too much SOTA-chasing in the literature (Church and Kordoni, 2022). Users should avoid wasting time on hyper-parameter tuning unless they are about to ship a model to a large number of users for an application where small improvements in performance are worth the effort.

## 2. gft Cheatsheet

*gft* supports the following functions:[10]

1. *fit* (also known as fine-tuning): $f_{pre} + data \rightarrow f_{post}$
2. *predict* (also known as inference): $f(x) = \hat{y}$, where $x$ is an input from *stdin* or from a data set
3. *eval*: $f + data \rightarrow score$ (produce a single score for a data set split, as opposed to a prediction, $\hat{y}$, for each input row in the split, $x$)
4. *summary*: Search hubs for popular data sets, models, and tasks and provide snippets. Popularity is estimated from metrics on downloads.
5. *cat_data*: Output data set on *stdout*

There are four major arguments:

1. –data: a data set on a hub, or a local file
2. –model: a model on a hub, or a local file
3. –task: for example, classify, regress[11]

[4] https://pytorch.org/
[5] https://www.tensorflow.org/
[6] https://github.com/google/jax
[7] https://www.paddlepaddle.org.cn/
[8] https://github.com/huggingface/transformers/blob/master/examples/pytorch/
[9] https://github.com/PaddlePaddle/PaddleNLP/tree/develop/examples
[10] https://github.com/kwchurch/gft/blob/master/doc/sections/cheatsheet.md
[11] Currently supported tasks are: classify (*aka* text-classification), classify_tokens (*aka* token-classification), classify_spans (*aka* QA, question-answering), classify_images (*aka* image-classification), classify_audio (*aka* audio-classification), regress, text-generation, MT (*aka* translation), ASR (*aka* ctc, automatic-speech-recognition), fill-mask. Tasks in parentheses are aliases.

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

**Table 1.** The standard recipe consists of three steps

| Step | Standard Terms | Proposed Terms | gft | Emerging Trends |
|---|---|---|---|---|
| 1 | ~~Pretrained~~ | $f_{pre}$ | Download from Hub | |
| 2 | ~~Fine-Tuning~~ | Fit | $f_{pre} + data \rightarrow f_{post}$ | *Deep Nets for Poets* |
| 3 | ~~Inference~~ | Predict | $f(x) = \hat{y}$ | *A Gentle Introduction to Fine-Tuning* |

4. –eqn (e.g., classify:$y \sim x_1 + x_2$), where a task appears before the colon, and variables refer to columns in the data set.

## 3. The standard recipe

Following (Howard and Ruder, 2018; Devlin *et al*., 2019), it has become standard practice to use the 3-step recipe in Table 1. We prefer the terms, *fit* and *predict*, to *fine-tuning* and *inference*. The proposed terminology has a long tradition in statistics and predates relatively recent work on deep nets.[12]

Fit and predict were discussed in two previous Emerging Trends articles in this journal (Church *et al*. 2021a, b). This paper will unify much of that discussion into a single github (see footnote 1) with hundreds of examples of short (1-line) programs.[13]

*gft* makes it easy to use models and data sets on hubs: HuggingFace[14] and PaddleHub/PaddleNLP).[15] The hubs are large (∼40k models and ∼4k data sets) and growing quickly (∼3x/year). The challenge is to make these amazing resources more accessible to as many users as possible. The target audience has diverse interests and skills. It should not be necessary for them to know much (if any) programming to join in on the fun.

The 40k models include both pretrained and post-trained models, $f_{pre}$ and $f_{post}$. *gft* provides tools to make it easy to find popular models, as well as popular data sets. We recommend users make as much use as possible of these resources and resist the temptation to pretrain their own models from scratch, for reasons that will be discussed in Appendix A.1.

### 3.1. An example of fit and predict in R

As mentioned above, *gft* is inspired by glm (general linear models) (Guisan *et al*., 2002) in R.[16] Listing 1 illustrates the use of *fit* and *predict* in R. The R environment provides a number of standard data sets such as *cars*, a data table with two columns, *speed* and *dist*, shown as black points in Figure 1. The model, *g*, fits *dist* as a quadratic function of *speed*. Predictions from this model are shown in red in Figure 1.

[12] In addition to history, there are two more reasons to prefer the terms, fit and predict. First, the proposed terminology, as mentioned above, demystifies deep nets. No one would suggest that regression-like methods are "intelligent." Second, the proposed terminology is intended to discourage work on *foundation models*, $f_{pre}$. As will be discussed in Appendix A.1, the term, *foundation models*, was introduced to encourage work on $f_{pre}$ (Bommasani *et al*., 2021), but we believe it is a mistake for academics to compete with industry on tasks that require large investments, and more logistics and systems work, than creative contributions to computational linguistics research.

[13] https://github.com/kwchurch/gft/tree/master/examples

[14] https://huggingface.co/

[15] https://github.com/PaddlePaddle

[16] https://www.r-project.org/

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

```
# Summarize the cars dataset
summary(cars)
# Create the black points
plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
# Fit a model g to cars dataset,
# assuming dist is a quadratic function of speed
g = glm(dist ~ poly(speed, 2), data=cars)
# Summarize the model g
summary(g)
o = order(cars$speed)
# Show predictions as a red line
lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
```

**Listing 1.** Example of fit and predict in R.

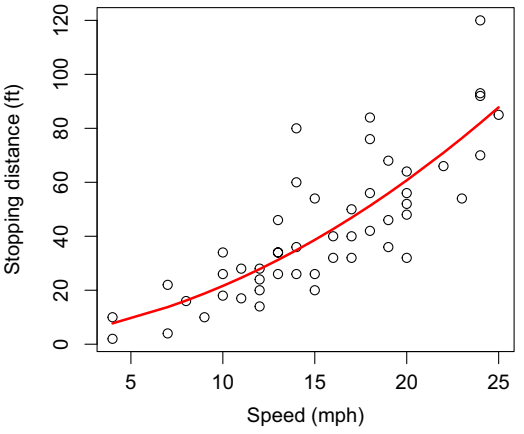


**Figure 1.** Results produced by Listing 1. Predictions from the model *g* are shown in red.

The *summary* function in R is applied to both the data table *cars* as well as the model *g*. The R *summary* function can be applied to almost any object and provides some useful description of its argument.

### *3.2. An example of fit (aka fine-tuning)*

Listing 2 shows an example of *gft_fit*. Listing 2 is similar to Listing 1 in a number of ways. Fit takes a pretrained model, $f_{pre}$, and uses a data set to output a post-trained model, $f_{post}$. In Listing 2, $f_{pre}$ is a BERT model, and the data table is the *emotion* data set on HuggingFace. The model in Listing 1, *g*, is analogous to $f_{post}$ = $*outdir* in Listing 2. The variables in both equations, line 7 of Listing 1 and line 3 of Listing 2, refer to columns in the relevant data table.

```
gft_fit --data       H:emotion \
        --model      H:bert-base-cased \
        --eqn        'classify:label~text' \
        --output_dir $outdir
```

**Listing 2.** Example of *gft_fit*.

Many *gft* programs take four arguments:

1. –data specifies the use of the emotion data set on HuggingFace.[17]

[17] https://huggingface.co/datasets/emotion

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

2. –model specifies the use of a BERT model on HuggingFace[18] as $f_{pre}$.
3. –eqn specifies a task (classification), plus a formula expressed in terms of columns in the data set.
4. –task specifies a task (not necessary when task is specified by –eqn argument).

Fit takes most of these (except for –task); in addition, fit requires –output_dir to specify a location for the output post-trained model, $f_{post}$.

### 3.3. An example of predict (aka inference)

```
x="I love you."
echo $x | gft_predict --task H:classify
```

**Listing 3.** Example of *gft_predict*. The default model performs sentiment analysis.

```
f=H:bhadresh-savani/distilbert-base-uncased-emotion
echo $x | gft_predict --task H:classify --model $f
```

**Listing 4.** Example of *gft_predict* with a model for emotion classification.

Listings 3 and 4 show two examples of *gft_predict*. Predict takes a novel input, $x$, and applies $x$ to a model, $f$, to produce a prediction, $\hat{y} = f(x)$. The default model (for the classification task) performs sentiment analysis; other models output other labels. In particular, the $f$ in Listing 4 outputs emotion classes: *anger, fear, joy, love, sadness, surprise*. To see the set of classes for a model, we recommend the use of *gft_summary*, as illustrated in Listing 5. *gft_summary* outputs the set of classes, among other things.

```
gft_summary --model $f
```

**Listing 5.** Example of *gft_summary*.

Some more classifications of $x = I$ *love you* are shown in Tables 2 and 3 using a number of different models from HuggingFace. Most of these models agree that $x$ is positive, though many of them classify $x$ as fake news and some classify $x$ as spam. One can use other models to classify $x$ in many ways such as offensive or not and hate speech or not.

Many of these classifiers were trained on corpora that may not be appropriate for this task. In particular, we really should not apply a Spanish classifier on English inputs, but mistakes like that are likely to happen given how easy it is to make such mistakes.

Most of the models on the hubs were created by the community. The hubs do not vet models for quality. The best models on the hubs are very good, though maybe not state of the art (SOTA). We rarely see results that are as good as PWC[19] and leaderboards.[20] Some models produce poor results, or no results (using standard mechanisms in *gft*). The most popular models (in terms of downloads) often produce competitive results, though the most popular models rarely produce the best results.

[18] https://huggingface.co/bert-base-cased
[19] https://paperswithcode.com/
[20] https://gluebenchmark.com/leaderboard

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

**Table 2.** Sentiment classification of *x* = *I love you*

| $\hat{y}$ | Score | Model | Classes |
|---|---|---|---|
| positive | 0.999 | moshew/tiny-bert-aug-sst2-distilled | negative, positive |
| positive | 0.999 | AdapterHub/bert-base-uncased-pf-sst2 | negative, positive |
| positive | 0.917 | SetFit/deberta-v3-large__sst2__train-32-1 | negative, positive |
| POSITIVE | 0.871 | ayameRushia/roberta-base-indonesian-sentiment-analysis-smsa | POSITIVE, NEUTRAL, NEGATIVE |
| positive | 0.807 | SetFit/distilbert-base-uncased__sst2__train-32-2 | negative, positive |
| positive | 0.651 | rohansingh/autonlp-Fake-news-detection-system-29906863 | negative, positive |
| positive | 0.512 | SetFit/deberta-v3-large__sst2__train-16-7 | negative, positive |
| stars | 0.872 | tomato/sentiment_analysis | 1 star, 2 stars, 3 stars, 4 stars, 5 stars |
| stars | 0.872 | nlptown/bert-base-multilingual-uncased-sentiment | 1 star, 2 stars, 3 stars, 4 stars, 5 stars |
| stars | 0.424 | cmarkea/distilcamembert-base-sentiment | 1 star, 2 stars, 3 stars, 4 stars, 5 stars |

**Table 3.** More classifications of *x* = *I love you*

| $\hat{y}$ | Score | Model | Classes |
|---|---|---|---|
| Fake | 0.998 | yaoyinnan/bert-base-chinese-covid19 | Neutral, Fake, Real |
| Fake | 0.986 | yaoyinnan/roberta-fakeddit | Fake, Real |
| FAKE | 0.959 | Narrativaai/fake-news-detection-spanish | REAL, FAKE |
| fake | 0.958 | Qiaozhen/fake-news-detector | real, fake |
| not spam | 1.000 | sureshs/distilbert-large-sms-spam | not spam, spam |
| spam | 0.826 | SetFit/distilbert-base-uncased__enron_spam__all-train | ham, spam |

### *3.4. Embarrassment of riches*

As mentioned at the beginning of this section, there are a huge number of models and data sets on the hubs. There are currently 40k models and 4k data sets, and these numbers are increasing rapidly (∼3x/year). How do we find the good stuff? And how do we use it?

The hubs provide a number of useful tools to answer these questions. There are GUI interfaces (as illustrated by footnotes[17] and [18]), as well as APIs. *gft_summary* uses the APIs to provide much of this functionality, as illustrated in Listing 6, which finds the five most popular data sets (or models) that contain the substring: "emotion." Popularity is estimated from downloads.

```
gft_summary --data  H:__contains__emotion --topn 5
gft_summary --model H:__contains__emotion --topn 5
```

**Listing 6.** Example of *gft_summary* as a search engine.

Listing 7 finds the most popular data sets and models by searching for data sets and models that contain the null string:

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

```
gft_summary --data  H:__contains__ --topn 5
gft_summary --model H:__contains__ --topn 5
```

**Listing 7.** Example of *gft_summary* with the null string as a query.

There are a few common naming conventions. Models containing the string "base" are likely to be base models, $f_{pre}$ (also known as pretrained models or foundation models). Models containing the string "distil" are like to be distilled (compressed models). Models containing the names of popular tasks such as "squad" and GLUE subtasks are likely to be post-trained models, $f_{post}$.

*gft_summary* can also be used to summarize data sets, models, tasks, etc. As mentioned in Section 3.1, these summaries are modeled after the summary function in R, which takes many different types of objects and produces useful descriptions.

```
f=H:bhadresh-savani/distilbert-base-uncased-emotion
gft_summary --data  H:emotion
gft_summary --model $f
gft_summary --task  H:classify
```

**Listing 8.** Examples of *gft_summary*.

### 3.5. Portability across hubs and frameworks

#### 3.5.1. Portability → stability over time

The code in the listings above take a dependency on HuggingFace, a small start-up company that has done very well recently. There are also dependencies on a number of Python packages that are constantly changing. We have seen many hardware and software platforms come and go. Many companies do well for a while, but success rarely lasts for long (decades). Deep nets will be more likely to survive the test of time if they are written in high-level languages such as *gft* that can be ported from one environment to another, as necessary.

Consider the example of operating systems. Unix survived the test of time better than alternatives such as VMS[21] because Unix was designed to port easily across suppliers. There was a time when Unix was mostly running on DEC machines,[22] and then there was a time when Unix was mostly running on Sun computers.[23] These days, Unix has moved on to other platforms. If programs are written in a relatively stable higher level environment like Unix (and *gft*), then old programs are more likely to continue to work for decades, despite instabilities at lower levels in the hardware and software stacks.

Too many deep nets are taking dependencies on Python packages that are updated very frequently (almost daily), often in incompatible ways. Many of these resources are supported by companies that could go out of business, or could decide to sunset support at any time. Given recent events, there is a risk that support could also be cutoff by sanctions and other instabilities in international relations. Because of these realities, *gft* is designed to make it easy to port from one hub to another.

#### 3.5.2. H is for HuggingFace and P is for PaddleNLP/PaddleHub

Listing 9 is similar to Listing 2, though dependencies on one company (H → HuggingFace) are replaced by dependencies on another company (P → Baidu's PaddleNLP/PaddleHub). *gft* supports mixing and matching models and data sets from different suppliers. "H:" uses resources

[21] https://en.wikipedia.org/wiki/OpenVMS
[22] https://digital.com/digital-equipment-corporation/
[23] https://thenewstack.io/sun-microsystems-a-look-back-at-a-tech-company-ahead-of-its-time/

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

```
gft_fit --data       P:chnsenticorp \
        --model      P:ernie-tiny \
        --eqn        'classify:label~text' \
        --output_dir $outdir
```

**Listing 9.** An example of *gft_fit* using P for PaddleNLP/PaddleHub.

from Huggingface, and "P:" uses resources from PaddleNLP/PaddleHub. gft also supports "C:" for custom resources on the local file system.

Note that most of the models on HuggingFace are based on PyTorch, whereas models on PaddleNLP and PaddleHub use a different framework called PaddlePaddle. *gft* hides much of this complexity.

Listing 9 uses the chnsenticorp data set,[24] which is different from the emotion dataset in Listing 2. The chnsenticorp data set specifies a sentiment analysis task in Chinese, whereas the emotion data set specifies an emotion classification task in English.

Listing 9 uses the ernie-tiny model (Su *et al.*, 2021), a compressed version of an ERNIE model. ERNIE models are similar to BERT models, though ERNIE models may be more appropriate for Chinese applications. Distillation (Hinton *et al.*, 2015) is a popular method to compress models. Compressed models tend to trade-off a little bit of performance (accuracy) in order to save a substantial amount of space and time when making predictions at inference time (Ganesh *et al.*, 2021). Distillation can be important for commercial applications.

## 4. Data sets and equations

### *4.1. Data sets*

As mentioned in Section 3.4, there are currently more than 4000 data sets on the hubs. We have already mentioned the emotion data set. Many data sets provide splits for training, validation, and test, though different data sets may name these splits differently. Each split provides a data table with columns and rows. The emotion data set, for example, contains two columns, named *text* and *label*. As can be seen in HuggingFace's data set viewer,[25] each row specifies a text field (e.g., "i didnt feel humiliated") and a label field (e.g., "sadness"). We will refer to the label field as a gold label. The task is to predict the gold labels.

SQuAD[26,27] (Rajpurkar *et al.*, 2016, 2018) is a popular data set for question answering. This data set has 5 columns: id, title, context, question, answers. The answers are substrings of the context, which makes this task considerably easier than the general case of Q&A (question answering), where the answer could be almost anything, and need not be mentioned in any of the other columns.

In Section 2.1 of (Church and Kordoni, 2022), there is a discussion of constructed queries like SQuAD. The TREC QA track[28] started with "constructed" questions in 1999, but quickly moved to "real" questions from query logs for subsequent TREC QA tracks (2000–2007) because constructed questions are too easy for systems and unrealistic (Voorhees, 2001).

Another popular data set is GLUE[29,30] (Wang *et al.*, 2018). GLUE contains a number of subsets: cola, sst2, wnli, mrpc, rte, qnli, qqp, sstb, mnli. Each subset contains 3 splits (train, validation, test). Different subsets have different columns.

[24] https://paperswithcode.com/sota/chinese-sentiment-analysis-on-chnsenticorp
[25] https://huggingface.co/datasets/emotion/viewer/default/train
[26] https://huggingface.co/datasets/squad/viewer/plain_text/train
[27] https://huggingface.co/datasets/squad_v2/viewer/squad_v2/train
[28] https://trec.nist.gov/data/qamain.html
[29] https://gluebenchmark.com/leaderboard
[30] https://paperswithcode.com/dataset/glue

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

**Table 4.** Fine-tuning for downstream tasks: GLUE, SQuAD, etc.

| –Data | –eqn |
|---|---|
| H:glue,cola | classify: label ~ sentence |
| H:glue,sst2 | classify: label ~ sentence |
| H:glue,wnli | classify: label ~ sentence |
| H:glue,mrpc | classify: label ~ sentence1 + sentence2 |
| H:glue,rte | classify: label ~ sentence1 + sentence2 |
| H:glue,qnli | classify: label ~ question + sentence |
| H:glue,qqp | classify: label ~ question1 + question2 |
| H:glue,sstb | regress: label ~ sentence1 + sentence2 |
| H:glue,mnli | classify: label ~ premise + hypothesis |
| H:squad | classify_spans: answers ~ question + context |
| H:squad_v2 | classify_spans: answers ~ question + context |
| H:tweet_eval,hate | classify: label ~ text |
| H:conll2003 | classify_tokens: pos_tags ~ tokens |
| H:conll2003 | classify_tokens: ner_tags ~ tokens |
| H:conll2003 | classify_tokens: chunk_tags ~ tokens |
| H:timit_asr | ctc: text ~ audio |
| H:librispeech_asr | ctc: text ~ audio |
| C:$gft/datasets/VAD/VAD | regress: Valence + Arousal + Dominance ~ Word |

GLUE has been updated with another task, SUPERGLUE (Wang *et al.*, 2019). Both GLUE and SUPERGLUE are popular on HuggingFace (in terms of downloads), though there are currently more downloads for GLUE.[31]

### *4.2. Examples of –data and –eqn*

Short (1-line) *gft* programs can fit (fine-tune) many benchmarks, as illustrated in Table 4. Table 4 shows –data and –eqn arguments for a number of popular benchmarks.

- data arguments start with a supplier, for example, H, P, C. After the colon, there can be one or two substrings, delimited by comma. For example, for the cola subtask of GLUE, the –data argument is H:glue,cola.
- eqn arguments consist of a task, plus a formula expressed in terms of columns in the dataset. See Table 5 for some examples of some tasks. For a more comprehensive list of tasks, see footnote [11].

[31] https://huggingface.co/datasets?sort=downloads&search=glue

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

**Table 5.** Some examples of tasks

| Task | Example | Description |
|---|---|---|
| classify, text-classification | Listing 3 | The left-hand side (lhs) of the equation is a categorical variable (integer or string) |
| QA, question-answering, classify_spans | Table 4 | Classify the beginning and end of spans (substrings); assumes the answer is a substring of the right-hand side (rhs) following conventions in the SQuAD task, as discussed in Section 4.1 |
| token-classification | Listing 10 | Classify each token (as opposed to one classification per row); this case includes NER (named entity recognition) and POS (part of speech tagging) |
| fill-mask | Listing 12 | Replace mask token, <mask>, with fillers |
| translation, MT | Listing 14 | Translate the input to another language; use different models for different language pairs |
| ASR, automatic-speech-recognition, ctc | Listing 15 | Input audio and output text |
| image-classification | Listing 16 | Input images and output classes |
| regress | Listing 20 | The lhs is a float (or a vector of floats) |

## 5. More examples and more tasks

As mentioned in footnote [13], there are hundreds of examples of *gft* in the github: fit,[32] predict,[33] summary,[34] and eval.[35] A few examples have already been discussed in Sections 3.2 and 3.3. Many more will be discussed in the next few subsections:

1. Predict (Section 5.1): token-classification, fill-mask, MT, ASR, etc.
2. Input from datasets (as opposed to *stdin*) (Section 5.2).
3. gft_predict → gft_eval (Section 5.3).

### *5.1. Predict*

A few examples of predict were shown in Listing 3. The gft documentation has many more examples of predict.[36]

#### *5.1.1. Token classification*

Some examples of token classification with PaddleNLP are shown in Listing 11.

Many of these tasks have been in the literature for a long time. Fill-mask is similar to the cloze task (Taylor, 1953), as illustrated in Listing 12.

Text generation is one of the more popular use cases for GPT-3, though Listing 13 uses a different model.

[32] https://github.com/kwchurch/gft/tree/master/examples/fit_examples
[33] https://github.com/kwchurch/gft/tree/master/examples/predict_examples
[34] https://github.com/kwchurch/gft/tree/master/examples/summary_examples
[35] https://github.com/kwchurch/gft/tree/master/examples/eval_examples
[36] https://github.com/kwchurch/gft/blob/master/doc/sections/functions/gft_predict.md

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

```
# token-classification: NER (Named Entity Recognition)
t=H:token-classification
echo 'I love New York.' | gft_predict --task $t

# part of speech (POS) tagging
f=H:vblagoje/bert-english-uncased-finetuned-pos
echo 'I love you' | gft_predict --model $f --task $t

# insert punctuation
f=H:Qishuai/distilbert_punctuator_en
echo 'I love you' | gft_predict --model $f --task $t
```

**Listing 10.** Example of token classification.

```
# NER with PaddleNLP
echo 'I love you.' | gft_predict --task P:ner

# Part of speech tagging with PaddleNLP
echo 'I love you.' | gft_predict --task P:pos_tagging
```

**Listing 11.** Example of token classification with PaddleNLP.

```
# Mask filling (also known as cloze task)
echo 'I <mask> you.' | gft_predict --task H:fill-mask
```

**Listing 12.** Example of fill-mask (also known as cloze task).

```
# Text Generation
echo 'I love ' | gft_predict --task H:text-generation
```

**Listing 13.** Example of text generation.

### *5.1.2. MT, ASR and more*

There are translation models for many language pairs, as illustrated in Listing 14.[37]

```
f=H:Helsinki-NLP/opus-mt-en-fr
echo 'I love you.' | gft_predict --task H:MT --model $f

f=H:Helsinki-NLP/opus-mt-en-zh
echo 'I love you.' | gft_predict --task H:MT --model $f
```

**Listing 14.** Example of machine translation (MT).

```
find $gft/doc -name '*.WAV' | gft_predict --task ASR
```

**Listing 15.** Example of automatic speech recognition (ASR).

```
f=H:nateraw/vit-base-cats-vs-dogs
find $gft/doc -name '*.jpg' | egrep PetImages |
gft_predict --task H:image-classification --model $f
```

**Listing 16.** Example of image classification.

[37] https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

### 5.2. Input from data sets (as opposed to stdin)

Listing 17 shows an example of input from a data set.

```
f=H:bhadresh-savani/distilbert-base-uncased-emotion
gft_predict --eqn 'classify:label~text' --model $f \
    --data H:emotion --split test
```

**Listing 17.** Example of input from data set (as opposed to *stdin*).

### 5.3. gft_predict → gft_eval

Listing 18 illustrates *gft_eval*.

```
gft_eval --eqn 'classify:label~text' --model $f \
    --data H:emotion --split test
```

**Listing 18.** *gft_eval* outputs a single score for a data set, as opposed to *gft_predict*, which outputs a prediction for each row.

### 5.4. Debugging, confusion matrices, and error analysis

In addition to producing a score with *gft_eval*, suppose we want to do some deep dives to look at particular errors. The code in Listing 19 will create a confusion matrix based on the validation split.

```
f=H:bhadresh-savani/distilbert-base-uncased-emotion
gft_predict --eqn 'classify:label~text' --model $f \
        --data H:emotion --split val > /tmp/pred
cut -f2,3 < /tmp/pred | sort | uniq -c | sort -nr > /tmp/conf
```

**Listing 19.** Code to create confusion matrix.

*gft_predict* outputs TSV (tab separated values) with 4 columns:

1. Input, $x$
2. Gold label, $y$
3. Predicted label, $\hat{y}$
4. Score

The cut statement on line 4 in Listing 19 selects $y$ and $\hat{y}$. The sort and uniq statements count the number of confusions, producing the confusion matrix shown in Table 6. Standard Unix tools such as grep (or AWK) can be used to find more details for particular confusions.

### 5.5. Vectors on the left hand side (LHS)

With regression and classification, the left-hand side (lhs) of the equation is typically a scalar, but *gft* has been generalized so the lhs can also be a point in a vector space, as shown in Listing 20. This example fine-tunes BERT with the NRC-VAD lexicon[38] (Mohammad, 2018). Words are assigned to points in $\mathbb{R}^3$, Valance, Arousal, and Dominance, based on VAD norms in psychology (Osgood *et al.*, 1957).

[38] https://saifmohammad.com/WebPages/nrc-vad.html

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

**Table 6.** Confusion matrix from Listing 19

| | Predicted Labels, $\hat{y}$ | | | | | |
|---|---|---|---|---|---|---|
| Gold Labels $y$ | sadness | joy | love | anger | fear | surprise |
| sadness | 530 | 1 | 1 | 7 | 11 | 0 |
| joy | 1 | 670 | 26 | 0 | 3 | 4 |
| love | 4 | 21 | 153 | 0 | 0 | 0 |
| anger | 10 | 4 | 1 | 254 | 6 | 0 |
| fear | 5 | 2 | 0 | 3 | 194 | 8 |
| surprise | 1 | 3 | 0 | 0 | 10 | 67 |

**Table 7.** Some gold labels and predictions from model, $f_{post}$, from Listing 20

| | Predictions, $\hat{y}$ | | | Gold, $y$ | | |
|---|---|---|---|---|---|---|
| Input, $x$ | $\hat{V}$ | $\hat{A}$ | $\hat{D}$ | $V$ | $A$ | $D$ |
| love | 0.976 | 0.530 | 0.675 | 1.000 | 0.519 | 0.673 |
| she loves me | 0.931 | 0.452 | 0.648 | NA | NA | NA |
| lovable | 0.920 | 0.318 | 0.608 | 0.948 | 0.335 | 0.565 |
| she loves me not | 0.382 | 0.375 | 0.349 | NA | NA | NA |
| ugly duckling | 0.300 | 0.517 | 0.242 | NA | NA | NA |
| unlovable | 0.203 | 0.502 | 0.399 | NA | NA | NA |
| she does not love me | 0.189 | 0.433 | 0.262 | NA | NA | NA |
| she hates me | 0.135 | 0.685 | 0.363 | NA | NA | NA |
| loath | 0.094 | 0.790 | 0.413 | 0.135 | 0.714 | 0.445 |
| hate | 0.017 | 0.780 | 0.451 | 0.031 | 0.802 | 0.430 |

```
gft_fit --model H:bert-base-cased \
        --data  C:$gft/datasets/VAD/VAD \
        --eqn   'regress: Valence + Arousal + Dominance ~ Word'
        --output_dir $outdir
```

**Listing 20.** An equation with a vector on the left-hand side (lhs).

Listing 20 is our first example of a custom data set. There are three CSV files on the local filesystem:

1. train split: $gft/datasets/VAD/VAD.train
2. validation split: $gft/datasets/VAD/VAD.val
3. test split: $gft/datasets/VAD/VAD.test

The three CSV files start with a header row that specifies the names of the columns. The variables in the equation refer to these columns in the CSV files.

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

In addition to illustrating the use of custom data sets, Listing 20 introduces two new features. First, we normally train models on corpora, but Listing 20 trains a model on a lexicon, the NRC-VAD lexicon. Second, regression usually takes scalar values on the left-hand side (lhs), but in this case, the lhs is a point in $\mathbb{R}^3$.

Listing 20 produces a post-trained model $f_{post}$. A few results with $f_{post}$ are shown in Table 7. This table shows some predictions, $\hat{y}$, for some inputs, $x$, using $f_{post}$. These predictions, $\hat{h}$, can be compared with gold labels, $y$, VAD scores from NRC-VAD (last three columns).

Although the model was trained on words (lemmas in the NRC Lexicon), the inputs, $x$, in Table 7 include a number of words, phrases, and texts, many of which are not in the NRC-VAD Lexicon (by construction). That is, $f_{post}$ can be applied to any input text (up to 512 subword units). Table 7 shows predictions, $\hat{V}$, $\hat{A}$, and $\hat{D}$, as well as gold values, $V$, $A$, and $D$. When the input, $x$, is not in the NRC-Lexicon, the gold value, $y$, is NA (not available). Since NRC-VAD is based on lemmas, NAs are to be expected for inflected forms, OOVs (out-of-vocabulary) words such as *unlovable*, MWEs (multiword expressions) such as *ugly duckling*, sentences, documents.

## 6. Conclusions

This paper proposed *gft*, a little language for fine-tuning pretrained base (foundation) models. Little languages make it easier for a broader audience (including non-programmers) to join in on the fun. Just as most users of regression do not need to know how to solve the regression optimization, so too users of deep nets should not need to understand hundreds of lines of Python and PyTorch. Higher level environments offer a number of advantages: ease of use, transparency, portability. *gft* removes much of the complexity, and much of the magic (and the alchemy) in deep nets, reducing fine-tuning to an optimization similar to regression. No one would suggest that regression-like methods are "intelligent."

## References

**Aho A.V.**, **Kernighan B.W.** and **Weinberger P.J.** (1987). *The AWK Programming Language*. Addison-Wesley Longman Publishing Co., Inc.

**Baevski A.**, **Zhou H.**, **Mohamed A.** and **Auli M.** (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv preprint arXiv:2006.11477*.

**Bentley J.L.** (1986). Little languages. *Communications of the ACM* **29**(8), 711–721.

**Bommasani R.**, **Hudson D.A.**, **Adeli E.**, **Altman R.**, **Arora S.**, **von Arx S.**, **Bernstein M.S.**, **Bohg J.**, **Bosselut A.**, **Brunskill E.**, **Brynjolfsson E.**, **Buch S.**, **Card D.**, **Castellon R.**, **Chatterji N.**, **Chen A.**, **Creel K.**, **Davis J.Q.**, **Demszky D.**, **Donahue C.**, **Doumbouya M.**, **Durmus E.**, **Ermon S.**, **Etchemendy J.**, **Ethayarajh K.**, **Fei-Fei L.**, **Finn C.**, **Gale T.**, **Gillespie L.**, **Goel K.**, **Goodman N.**, **Grossman S.**, **Guha N.**, **Hashimoto T.**, **Henderson P.**, **Hewitt J.**, **Ho D.E.**, **Hong J.**, **Hsu K.**, **Huang J.**, **Icard T.**, **Jain S.**, **Jurafsky D.**, **Kalluri P.**, **Karamcheti S.**, **Keeling G.**, **Khani F.**, **Khattab O.**, **Kohd P.W.**, **Krass M.**, **Krishna R.**, **Kuditipudi R.**, **Kumar A.**, **Ladhak F.**, **Lee M.**, **Lee T.**, **Leskovec J.**, **Levent I.**, **Li X.L.**, **Li X.**, **Ma T.**, **Malik A.**, **Manning C.D.**, **Mirchandani S.**, **Mitchell E.**, **Munyikwa Z.**, **Nair S.**, **Narayan A.**, **Narayanan D.**, **Newman B.**, **Nie A.**, **Niebles J.C.**, **Nilforoshan H.**, **Nyarko J.**, **Ogut G.**, **Orr L.**, **Papadimitriou I.**, **Park J.S.**, **Piech C.**, **Portelance E.**, **Potts C.**, **Raghunathan A.**, **Reich R.**, **Ren H.**, **Rong F.**, **Roohani Y.**, **Ruiz C.**, **Ryan J.**, **Ré C.**, **Sadigh D.**, **Sagawa S.**, **Santhanam K.**, **Shih A.**, **Srinivasan K.**, **Tamkin A.**, **Taori R.**, **Thomas A.W.**, **Tramèr F.**, **Wang R.E.**, **Wang W.**, **Wu B.**, **Wu J.**, **Wu Y.**, **Xie S.M.**, **Yasunaga M.**, **You J.**, **Zaharia M.**, **Zhang M.**, **Zhang T.**, **Zhang X.**, **Zhang Y.**, **Zheng L.**, **Zhou K.** and **Liang P.** (2021). On the opportunities and risks of foundation models.

**Brown T.B.**, **Mann B.**, **Ryder N.**, **Subbiah M.**, **Kaplan J.**, **Dhariwal P.**, **Neelakantan A.**, **Shyam P.**, **Sastry G.**, **Askell A.**, **Agarwal S.**, **Herbert-Voss A.**, **Krueger G.**, **Henighan T.**, **Child R.**, **Ramesh A.**, **Ziegler D.M.**, **Wu J.**, **Winter C.**, **Hesse C.**, **Chen M.**, **Sigler E.**, **Litwin M.**, **Gray S.**, **Chess B.**, **Clark J.**, **Berner C.**, **McCandlish S.**, **Radford A.**, **Sutskever I.** and **Amodei D.** (2020). Language models are few-shot learners. NeurIPS.

**Buck C.**, **Heafield K.** and **van Ooyen B.** (2014). N-gram counts and language models from the common crawl. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, Reykjavik, Iceland: European Languages Resources Association (ELRA), pp. 3579–3584.

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

**Chelba C.**, **Mikolov T.**, **Schuster M.**, **Ge Q.**, **Brants T.**, **Koehn P.** and **Robinson T.** (2013). One billion word benchmark for measuring progress in statistical language modeling. arXiv preprint arXiv:1312.3005.

**Chowdhery A.**, **Narang S.**, **Devlin J.**, **Bosma M.**, **Mishra G.**, **Roberts A.**, **Barham P.**, **Chung H.W.**, **Sutton C.**, **Gehrmann S.**, et al. (2022). PaLM: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311.

**Church K.**, **Chen Z.** and **Ma Y.** (2021a). Emerging trends: A gentle introduction to fine-tuning. *Natural Language Engineering* **27**(6), 763–778.

**Church K.W.** and **Kordoni V.** (2022). Emerging trends: Sota-chasing. *Natural Language Engineering* **28**(2), 249–269.

**Church K.W.**, **Yuan X.**, **Guo S.**, **Wu Z.**, **Yang Y.** and **Chen Z.** (2021b). Emerging trends: Deep nets for poets. *Natural Language Engineering* **27**(5), 631–645.

**Conneau A.**, **Khandelwal K.**, **Goyal N.**, **Chaudhary V.**, **Wenzek G.**, **Guzmán F.**, **Grave E.**, **Ott M.**, **Zettlemoyer L.** and **Stoyanov V.** (2019). Unsupervised cross-lingual representation learning at scale. CoRR, abs/1911.02116.

**Dale R.** (2021). GPT-3: What's it good for? *Natural Language Engineering* **27**(1), 113–118.

**Deng J.**, **Dong W.**, **Socher R.**, **Li L.-J.**, **Li K.** and **Fei-Fei L.** (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 248–255.

**Devlin J.**, **Chang M.-W.**, **Lee K.** and **Toutanova K.** (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186.

**Du J.**, **Na X.**, **Liu X.** and **Bu H.** (2018). Aishell-2: Transforming mandarin asr research into industrial scale. arXiv preprint arXiv:1808.10583.

**Ganesh P.**, **Chen Y.**, **Lou X.**, **Khan M.A.**, **Yang Y.**, **Sajjad H.**, **Nakov P.**, **Chen D.** and **Winslett M.** (2021). Compressing large-scale transformer-based models: A case study on BERT. *Transactions of the Association for Computational Linguistics* **9**, 1061–1080.

**Guisan A.**, **Edwards** Jr, **T.C.** and **Hastie T.** (2002). Generalized linear and generalized additive models in studies of species distributions: Setting the scene. *Ecological Modelling* **157**(2–3), 89–100.

**He K.**, **Zhang X.**, **Ren S.** and **Sun J.** (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

**Hinton G.**, **Vinyals O.** and **Dean J.** (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

**Howard J.** and **Ruder S.** (2018). Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume* 1: *Long Papers)*, Melbourne, Australia: Association for Computational Linguistics, pp. 328–339.

**Ito K.** and **Johnson L.** (2017). The LJ speech dataset. https://keithito.com/LJ-Speech-Dataset/

**Liu Y.**, **Ott M.**, **Goyal N.**, **Du J.**, **Joshi M.**, **Chen D.**, **Levy O.**, **Lewis M.**, **Zettlemoyer L.** and **Stoyanov V.** (2019). Roberta: A robustly optimized bert pretraining approach.

**Mohammad S.** (2018). Obtaining reliable human ratings of valence, arousal, and dominance for 20,000 English words. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume* 1: *Long Papers)*, Melbourne, Australia: Association for Computational Linguistics, pp. 174–184.

**Moore G.A.** and **McKenna R.** (1999). *Crossing the Chasm*. Capstone Oxford.

**Osgood C.E.**, **Suci G.J.** and **Tannenbaum P.H.** (1957). *The Measurement of Meaning*. vol. **47**. University of Illinois press.

**Panayotov V.**, **Chen G.**, **Povey D.** and **Khudanpur S.** (2015). Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5206–5210.

**Radford A.**, **Wu J.**, **Child R.**, **Luan D.**, **Amodei D.** and **Sutskever I.** (2019). Language models are unsupervised multitask learners. OpenAI Blog.

**Rajpurkar P.**, **Jia R.** and **Liang P.** (2018). Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume* 2: *Short Papers)*, Melbourne, Australia: Association for Computational Linguistics, pp. 784–789.

**Rajpurkar P.**, **Zhang J.**, **Lopyrev K.** and **Liang P.** (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas: Association for Computational Linguistics, pp. 2383–2392.

**Su W.**, **Chen X.**, **Feng S.**, **Liu J.**, **Liu W.**, **Sun Y.**, **Tian H.**, **Wu H.** and **Wang H.** (2021). Ernie-tiny: A progressive distillation framework for pretrained transformer compression. arXiv preprint arXiv:2106.02241.

**Sun Y.**, **Wang S.**, **Feng S.**, **Ding S.**, **Pang C.**, **Shang J.**, **Liu J.**, **Chen X.**, **Zhao Y.**, **Lu Y.**, et al. (2021). Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*.

**Sun Y.**, **Wang S.**, **Li Y.**, **Feng S.**, **Tian H.**, **Wu H.** and **Wang H.** (2020). Ernie 2.0: A continual pre-training framework for language understanding. *AAAI*.

**Taylor W.L.** (1953). "Cloze procedure": A new tool for measuring readability. *Journalism Quarterly* **30**(4), 415–433.

**Voorhees E.M.** (2001). The TREC question answering track. *Natural Language Engineering* **7**(4), 361–378.

**Wang A.**, **Pruksachatkun Y.**, **Nangia N.**, **Singh A.**, **Michael J.**, **Hill F.**, **Levy O.** and **Bowman S.** (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in Neural Information Processing Systems* 32.

**Wang A.**, **Singh A.**, **Michael J.**, **Hill F.**, **Levy O.** and **Bowman S.** (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Brussels, Belgium: Association for Computational Linguistics, pp. 353–355.

**Wu B.**, **Xu C.**, **Dai X.**, **Wan A.**, **Zhang P.**, **Yan Z.**, **Tomizuka M.**, **Gonzalez J.**, **Keutzer K.** and **Vajda P.** (2020). Visual transformers: Token-based image representation and processing for computer vision.

**Zhu Y.**, **Kiros R.**, **Zemel R.**, **Salakhutdinov R.**, **Urtasun R.**, **Torralba A.** and **Fidler S.** (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 19–27.

## A Appendix

### *A.1 Pretraining ($f_{pre}$): Don't do it (yourself)*

Recent work on foundation models[39] (Bommasani *et al.*, 2021) attempts to compete with industry on what industry does best. We think this is a mistake. Industry has "unfair" advantages[40] on tasks like pretraining $f_{pre}$, which require large investments in people and machines, as shown in Table 8.

We recommend that academics focus on fit and predict, which are much more affordable than pretraining $f_{pre}$. The last two columns in Table 8, time and hardware, obviously depend on many factors such as the size of the model. One of the motivations behind distillation (Hinton *et al.*, 2015; Ganesh *et al.*, 2021) is to reduce the size of the model. Smaller models tend to run faster at inference time. While inference times are relatively faster than training times, inference time is often a bottleneck for commercial applications since training is a one-time investment, whereas inference is a recurring cost. For successful applications with millions or billions of users, recurring costs can easily dominate one-time training costs.

As for training costs, pretraining is much more expensive than fine-tuning, especially for large models. Pretraining is already very expensive and will become even more expensive in the future as models become larger and larger. Pretraining large models will be beyond the means of academics (and governments).

Consider the pretrained models in Table 9, and especially the largest model, PaLM (Chowdhery *et al.*, 2022). PaLM produces impressive results, using a huge model (540B parameters). That said, the size of the investment is even more impressive: the paper has dozens of authors using thousands of TPUs (distributed over multiple data centers).

When the investments are this large, projects become risk adverse. Projects of this size cannot afford to fail. Academics should focus on projects that reward creativity and avoid projects that are too big to fail.

We like to think of $f_{pre}$ like Intel CPU chips. Universities can afford to program CPUs, but universities cannot afford to compete with Intel and fabricate their own CPUs. So too, we argue that universities can afford to fit and predict deep nets, but they cannot afford to compete with industry on $f_{pre}$. When the first author was a student at MIT, his thesis advisor, Jon Allen, urged the university to make large investments in VLSI fabrication. In retrospect, it was probably a mistake for a university to invest in VLSI fabrication, though others may disagree with that assessment.[41]

In short, we recommend users start by downloading $f_{pre}$ from hubs and focus on steps 2 (fit) and 3 (predict) of the standard recipe. Some examples of $f_{pre}$ are shown in Table 9. Many of these models can be downloaded from hubs, with a few exceptions, especially for larger models such as ERNIE 3.0, GPT-3, PaLM. Most models are trained on corpora, as shown in Table 10.

[39] https://crfm.stanford.edu/workshop.html

[40] "Unfair advantages" is management jargon, common in industry, especially when discussing strategy. Obviously, there is nothing "unfair" about taking advantage of one's strengths.

[41] http://www.eecs.mit.edu/docs/newsletter/VLSI.pdf

https://doi.org/10.1017/S1351324922000237 Published online by Cambridge University Press

**Table 8.** Most $f_{pre}$ models are trained in industry because pretraining requires large capital investments in large teams and GPU clusters

| Step | Description | Time | Hardware |
|---|---|---|---|
| 1 | $f_{pre}$ (Table 9) | Days/Weeks | Large GPU Cluster |
| 2 | Fit (Section 3.2) | Hours/Days | 1+ GPUs |
| 3 | Predict (Section 3.3) | Seconds/Minutes | 0+ GPUs |

**Table 9.** gft starts with large pre-trained base models, $f_{pre}$, typically trained on large corpora in Table 10, using expensive GPU clusters

| Base Model ($f_{pre}$) | Params | Training Data |
|---|---|---|
| ResNet-50 (He *et al*., 2016) | 23M | 14M images from ImageNet |
| VT (Wu *et al*., 2020) | 11.7M-21.9M | 14M images from ImageNet |
| Wav2vec (Baevski *et al*., 2020) | 95M-317M | 960 hours from LibriSpeech |
| BERT (Devlin *et al*., 2019) | 110M-340M | 3.3B words from Books and Wikipedia |
| ERNIE 2.0 (Sun *et al*., 2020) | 110M-340M | 7.9B en + 15B zh tokens |
| RoBERTa (Liu *et al*., 2019) | 110M | 160GBs of text |
| GPT-2 (Radford *et al*., 2019) | 1.5B | 40GBs of text |
| ERNIE 3.0 (Sun *et al*., 2021) | 10B | 375B tokens of text and knowledge graph |
| GPT-3 (Brown *et al*., 2020; Dale, 2021) | 125M-175B | 1TB from Common Crawl, Books and Wikipedia |
| XLM-RoBERTa (Conneau *et al*., 2019) | 12B-16B | 2.5TBs multlingual |
| PaLM (Chowdhery *et al*., 2022) | 540B | 740B tokens (multilingual) |

**Table 10.** Some popular corpora for training pre-trained models, $f_{pre}$

| Dataset | Description |
|---|---|
| Billion Word (Chelba *et al*., 2013) | a billion words of English |
| Common Crawl (Buck *et al*., 2014) | https://github.com/commoncrawl |
| Book Corpus (Zhu *et al*., 2015) | speech with text |
| ImageNet (Deng *et al*., 2009) | 14M images, annotated with 21k classes |
| LibriSpeech (Panayotov *et al*., 2015) | 960 hours of speech with text |
| LJ Speech (Ito and Johnson, 2017) | https://keithito.com/LJ-Speech-Dataset/ |
| AISHELL (Du *et al*., 2018) | 1000 Hours of Mandarin speech with text |

**Cite this article:** Ward Church K, Cai X, Ying Y, Chen Z, Xun G and Bian Y (2022). Emerging trends: General fine-tuning (*gft*). *Natural Language Engineering* **28**, 519–535. https://doi.org/10.1017/S1351324922000237