*Natural Language Engineering* (2021), **27**, pp. 631–645
doi:10.1017/S1351324921000231

CAMBRIDGE
UNIVERSITY PRESS

EMERGING TRENDS

# Emerging trends: Deep nets for poets

Kenneth Ward Church[1]* Xiaopeng Yuan[2] Sheng Guo[2] Zewu Wu[2] Yehua Yang[2] and Zeyu Chen[2]

[1]Baidu, Sunnyvale, CA, USA and [2]Baidu, Beijing, China
*Corresponding author. E-mail: KennethChurch@baidu.com

(Received 7 August 2021; revised 9 August 2021)

**Abstract**

Deep nets have done well with early adopters, but the future will soon depend on crossing the chasm. The goal of this paper is to make deep nets more accessible to a broader audience including people with little or no programming skills, and people with little interest in training new models. A github is provided with simple implementations of image classification, optical character recognition, sentiment analysis, named entity recognition, question answering (QA/SQuAD), machine translation, speech to text (SST), and speech recognition (STT). The emphasis is on instant gratification. Non-programmers should be able to install these programs and use them in 15 minutes or less (per program). Programs are short (10–100 lines each) and readable by users with modest programming skills. Much of the complexity is hidden behind abstractions such as pipelines and auto classes, and pretrained models and datasets provided by hubs: PaddleHub, PaddleNLP, HuggingFaceHub, and Fairseq. Hubs have different priorities than research. Research is training models from corpora and fine-tuning them for tasks. Users are already overwhelmed with an embarrassment of riches (13k models and 1k datasets). Do they want more? We believe the broader market is more interested in inference (how to run pretrained models on novel inputs) and less interested in training (how to create even more models).

**Keywords:** Deep nets; training; inference; hubs; pretrained models

## 1. Unix for poets → deep nets for poets

The title of this paper is borrowed from *Unix for Poets*,[a] a slide deck that was prepared for a Linguistics Summer School when empirical methods were just beginning to take off in the early 1990s. There were a few of us that were really excited about corpus-based methods and lexical resources such as dictionaries and ontologies (WordNet). Although there were relatively few papers on empirical methods in those days, a few of us believed there would soon be more, and so there were.

*Unix for Poets* emphasized pragmatism. Data were becoming available like never before. What can we do with it all? It is better to do something simple than nothing at all. Because there was too much low hanging fruit for us to pick by ourselves, we invested in a number of community-building exercises. A number of us created new venues such as Empirical methods in Natural Language Processing (EMNLP), and mechanisms for sharing data such as the Linguistic Data Consortium (LDC).[b]

*Unix for Poets* was yet another community-building exercise. At the time, there were a number of sympathetic potential users in linguistics, lexicography, and translation, but they needed help

[a]https://web.stanford.edu/class/cs124/kwc-unix-for-poets.pdf
[b]https://www.ldc.upenn.edu/

with technology. "Poets" is an imperfect metaphor to refer to some of our colleagues that were more comfortable writing grammars than shell scripts. We argued that they can do simple things themselves, and that DIY (do it yourself) is more satisfying than begging a programmer for "help."

*Unix for Poets* emphasized instant gratification. Make sure that students are rewarded with accomplishments in the first 15 minutes. Students were impressed when they learned how to count bigrams with a one-line shell script in the first 15 minutes.

The situation with Deep Nets today is similar to where we were with corpora in the 1990s. Because there is too much low hanging fruit for us to pick by ourselves, we are investing in community-building exercises such as this paper. We believe there is a large potential audience of non-programmers that may not appreciate just how easy it is to apply pretrained models to applications such as:

1. Image Classification:
   Pictures → Labels
2. OCR (Optical Character Recognition):
   Pictures → text (English and Chinese)
3. Sentiment Analysis:
   Text (English and Chinese) → positive or negative
4. NER (Named Entity Recognition):
   Text → named entities (spans with labels such as person and location)
5. QA (Question Answering and SQuAD (Rajpurkar *et al.* 2016)):
   Questions and documents (text) → Answers (spans)
6. MT (Machine Translation):
   Text in source language → Text in target language
7. TTS (Text to Speech (also known as speech synthesis)):
   Text → Audio
8. STT (Speech to Text (also known as speech recognition and ASR)):
   Audio → Text

Code is posted on github,[c] with an emphasis on instant gratification. In 15 minutes or less, non-programmers should be able to download any one of these examples and run it (as is) on novel inputs, without the need to understand how it works. Most of the programs are short, typically 10–100 lines of code per example. The code should be easy to understand (and modify) by users with modest programming skills.

Each solution is assigned a separate stand-alone subdirectory with no dependencies between directories, so they can be understood one by one, in any order. The code is written in a way to call out certain patterns and similarities between examples.

## 2. Crossing the chasm

Deep nets have done very well with early adapters, but deep nets will soon need to cross the chasm (Moore and McKenna 1991). Many technologies start out well, appealing to early adopters, but soon thereafter, success depends on finding ways to appeal to a broader set of users. Many promising technology companies failed because the early adopter market was too small, and their technology was too challenging to appeal to mainstream users.

When the first author was a student at MIT, we had a twisted set of priorities. We were too heavily invested in our beloved technologies (whatever was hot at the time). We were total elitists and

[c]https://github.com/kwchurch/deepnet_examples

held users in rather low regard because they were insufficiently invested in what was important to us. We referred to our friends as *winners* and users as *lusers*[d] [sic].

When users get stuck, they often blame themselves: *We're not worthy.*[e] However, it is not good for business to blame the customer or to allow them to blame themselves. When the technology is too challenging for users, it is the technology that suffers, not the so-called "lusers."

## 3. Lessons from Unix

Unix stood up better to the test of time than many operating systems that you have probably never heard of. The first author wrote his Ph.D. thesis on a computer using an operating system called ITS (incompatible time-sharing system),[f] a reaction to (CTSS).[g,h] UNIX (Uniplexed Information and Computing Services) was a reaction to MULTICS (Multiplexed Information and Computing Services).[i]

What did Unix get right, and what did these other systems get wrong? According to Ken Thompson, one of the two creators of Unix, Multics was

*overdesigned and overbuilt and over everything. It was close to unusable. They [MIT] still claim it's a monstrous success, but it just clearly wasn't* (Seibel 2009).

Unix emphasized *Less is More* and *Small is Beautiful.*[j] Unix borrowed a few good ideas from Multics, but left much of the complexity behind. The emphasis in Unix on simplicity applies to both technology and spelling; the single letter "x" in Unix is shorter than the bigram "cs" in Multics, according to Section 2.4 of (Kernighan 2019). Rumor has it that Unix was derived from "castrated MULTICS," but this rumor has been denied by Dennis Ritchie, the other creator of Unix.[k]

Unix was successful because simplicity[l] sells to the market that matters. Editing is hard work, but it is worth the effort:[m]

*If I Had More Time, I Would Have Written a Shorter Letter.*

We often find that writing helps us understand what is really important. We usually do not start the writing process until we think we understand what is important, but as we write down our thoughts, we begin to appreciate just how little we understood when we started the process, as well as how much more we understand at the end of the process.

One needs to deeply understand what matters to make the user interface so simple that anyone can use it. All too often, the user interface and the documentation is a mess because the organization never made it a priority to understand what matters. We used to worship wizard programmers that could perform magic, but we should have been more concerned with cleaning up and making the mess more accessible to the masses.

[d]https://www.urbandictionary.com/define.php?term=luser
[e]https://www.youtube.com/watch?v=jjaqrPpdQYc
[f]https://github.com/PDP-10/its
[g]https://www.computerhistory.org/timeline/1961/
[h]https://en.wikipedia.org/wiki/Compatible_Time-Sharing_System
[i]https://en.wikipedia.org/wiki/Multics
[j]https://www.usenix.org/legacy/events/bsdcon/mashey_small/
[k]http://article.olduse.net/4743@Aucbvax.UUCP
[l]https://www.youtube.com/watch?v=JoVQTPbD6UY
[m]https://quoteinvestigator.com/2012/04/28/shorter-letter/

**Table 1.** A simple API for several deep net inference applications

| App | Input | Output |
|---|---|---|
| Image classification | Filenames (pictures) | Labels |
| OCR (optical character recognition) | Filenames (pictures) | Bounding boxes and text |
| Sentiment | Text | Labels (positive/negative) |
| NER (named entity recognition) | Text | Entities |
| QA (question answering) | Questions and Documents | Answers |
| MT (machine translation) | Text in source language | Text in target language |
| TTS (speech synthesis) | Text | Filenames (audio) |
| STT (speech recognition) | Filenames (audio) | Text |

**Table 2.** See footnote[ag] for short-cut links to code for each checkmark (√), with an emphasis on instant gratification, simplicity, and readability. More examples will be added soon

| App | PaddleHub | PaddleNLP | HuggingFaceHub | Fairseq |
|---|---|---|---|---|
| Image classification | √ | | √ | |
| OCR | √ | | | |
| Sentiment | √ | | √ | |
| NER | √ | | √ | |
| QA (question answering) | | √ | √ | |
| MT (translation) | √ | | √ | √ |
| TTS (speech synthesis) | √ | | √ | |
| STT (speech recognition) | | | √ | |

## 4. A simple API for deep net inference apps

Table 1 introduces a simple API based on top of deep nets. This API follows the Unix philosophy (McIlroy *et al.* 1978):

*This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*[n]

The solutions in Table 2 emphasize instant gratification, simplicity, consistent terminology, ease of use, etc.; these solutions may not be the best in terms of computational resources (space and time) and performance on shared tasks.

The programs in Table 1

1. do one thing (and only one thing),
2. input and output text streams (universal interface), and
3. work together with Unix pipes.

[n] https://homepage.cs.uri.edu/thenry/resources/unix_art/ch01s06.html

```
# Inference apps read from stdin and write to stdout
# OCR inputs filenames and
#    outputs bounding boxes and text
# translate inputs text in one language and
#    outputs text in another
echo "$OCRdir/Sample_input_for_OCR.png" |
    python $OCRdir/OCR.py |
    # Extract text (and ignore locations of boxes)
    cut -f4- |
    python $MTdir/translate.py -s zh -t en
```

**Figure 1.** Example of combining OCR and translation with Unix pipes.

In most cases, inputs and outputs use standard IO in Unix (stdin and stdout, respectively). Many of the APIs in Table 1 input text and output text, making it possible to combine many of the programs with Unix pipes. Thus, for example, one can pipe the output from OCR into machine translation (MT), as illustrated in Figure 1.

Some of the programs input pictures and audio. In these cases, the input is a sequence of filenames. Standard utilities are used to read and write picture and audio files in standard formats such as jpg, png, and wav.

There are a few additional arguments. MT takes arguments to specify the source and target languages. Some implementations support more language pairs than others. All of these programs accept –help as an argument.

Models and datasets will be downloaded as necessary and cached on the local filesystem. Downloads sometimes encounter intermittent timeouts. If you encounter such a timeout, please try again.

## 5. Publish or perish

Academics understand the importance of publishing, but many technologies failed because documentation was treated as an after-thought. Documentation is often delegated to technical writers that are never given much respect. Technical writers are treated as second-class minions, only slightly above customer service agents in the pecking order.

In contrast, Unix took documentation very seriously. Before Unix, it was hard to imagine that a manual on an operating system could become a best seller, but Brian Kernighan wrote a number of popular books, some of which have been translated into a number of languages (Kernighan and Plauger 1974; Kernighan *et al.* 1978; Aho *et al.* 1979; Kernighan and Ritchie 1988; Kernighan and Pike 1999; Kernighan 2019). Brian's contributions made Unix accessible to the audience that ultimately made Unix the success that it has become.[o]

Vic Vyssotsky, an executive in charge of the Unix group, told us in an off-site meeting that we should not write man pages for our friends in the office down the hall. Rather, we should be thinking of a broader audience of future users. One might hope that product teams will find ways to make our mess more accessible to the masses, but realistically, that is unlikely to happen. Simplicity and accessibility need to be designed into the final product from the very beginning, but all too often users are treated as an after-thought.

In retrospect, it turned out to be super-important to the success of Unix to treat technical writing and Brian Kernighan with respect. Brian not only documented what they did, but he helped them understand what they had accomplished, as well as what they needed to fix.

As for documentation on deep nets, there are textbooks on machine learning such as (Bishop 2006) and (Goodfellow *et al.* 2016), as well as more practical books such as (Chollet 2017)

[o]https://www.youtube.com/watch?v=xnCgoEyz31M

and (Géron 2019), but we often find ourselves reading online documentation directly from frameworks (PaddlePaddle,[p] TensorFlow[q] and PyTorch),[r] as well as online documentation from HuggingFace (Wolf *et al.* 2019), Sklearn (Pedregosa *et al.* 2011), and Scipy.

Blogs are often useful for learning about deep nets. Bertviz, for example, is a good way to learn about deep nets and transformer models such as BERT (Develin *et al.* 2019) and ERNIE (Sun *et al.* 2019). We would start with the blog.[s] More motivated users should then read the paper (Vig 2019) and download code from github.[t]

While all of these resources are valuable, none of them provide the kind of crisp broad overview tutorial like what Kernighan has done for Unix. And none of these resources are appropriate for poets. Note that Kernighan teaches an undergraduate class for poets.[u]

## 6. Deep nets for the Unix group

The API in Table 1 was designed to appeal to non-programmers (poets), but it may also appeal to highly skilled programmers such as the people that created Unix. The Unix group never warmed to AI. The first author joined Bell Labs in 1983 as a token gesture to AI during a boom and began to embrace the Unix philosophy only after his beloved technology (Lisp Machines) fell into the chasm during the subsequent AI winter.

At the fiftieth reunion of Unix,[v] Mike Lesk, a member of the Unix group (Lesk 1976; Kernighan *et al.* 1978), and early pioneer in information retrieval (Salton and Lesk 1968; Lesk 1969), lexical resources (Lesk 1996) and what is now known as map search (Elliott and Lesk 1982), asked a question about machine learning.[w] Al Aho answered the question by making it clear that he would like to see an abstraction of machine learning that is more like automata theory. While this topic is beyond the scope of this paper, a promising possible step in this direction was suggested in Bronstein's keynote at ICLR-2021[x] (Bronstein *et al.* 2021).

Al Aho likes theory. Others in the Unix group would like to see a description of deep nets with more emphasis on brevity and clarity. Doug McIlroy, the inventor of Unix pipes, was also the driving force behind accessible documentation and short man pages. When he was the boss of the Unix group, he insisted that man pages be no longer than a single page. He was also a strong advocate for *the Unix philosophy* (McIlroy *et al.* 1978). His influence was very much behind the suggestion of piping apps together as discussed in Section 4.

## 7. Priorities: Different strokes for different folks

Much of the literature on deep nets emphasizes different priorities. This literature appeals to early adopters, largely engineers that are deeply invested in machine learning, and have spent many hours learning how to program and how to use various frameworks such as PaddlePaddle, TensorFlow, and PyTorch. Many of these papers train new models from raw corpora or fine-tune an existing model for a particular task. Papers are expected to provide an evaluation showing that the proposed model performs better than a strong baseline on a standard task.

The literature on deep nets is considered extremely successful, as evidenced by the number of publications. There are so many papers that there are now tools such as papers with code[y] to help

[p]https://github.com/PaddlePaddle/Paddle
[q]https://www.tensorflow.org/
[r]https://pytorch.org/
[s]https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1
[t]https://github.com/jessevig/bertviz
[u]https://www.cs.princeton.edu/courses/archive/fall21/cos109/
[v]https://www.bell-labs.com/events-news/events/unix-50-event/#gref
[w]https://youtu.be/kZd8e2pzZ7Q?t=1578
[x]https://towardsdatascience.com/geometric-foundations-of-deep-learning-94cdd45b451d
[y]https://paperswithcode.com/sota

researchers find tasks and strong baselines to include in the evaluation section of their next paper. Strong baselines are referred to as state of the art (SOTA).

The priorities in research may not be optimal.[z] Users (poets) are already overwhelmed with an embarrassment of riches. We already have 13k models and 1k datasets in hubs to be discussed in Section 9. Do we want more? Or should we do more with the ones we already have?

There are a number of hubs that encourage users to do more with what we have by making it easy to download models and datasets:

1. PaddleHub[aa],
2. PaddleNLP[ab],
3. HuggingFaceHub[ac], and
4. Fairseq[ad].

Venture Capital is investing in hubs.[ae] Hubs have different priorities than research. Hubs are growing their customer base, whereas research is creating new models with better and better performance.

While performance is less of a priority for hubs, they can easily catch up, using a quick-follow approach borrowed from bicycle racing. In bicycle racing, it is hard work to be at the front of the pack, breaking wind for everyone else. A winning strategy is to spend much of the race in the peloton drafting behind others, avoiding the temptation to sprint to the front, except when it is worth the effort to do so.[af] Hubs are taking a similar approach. They are resisting the temptation to sprint up leaderboards. The best of those models will show up on hubs soon enough, given the incentives in research to share results as widely as possible. In this way, hubs are likely to win the race by drafting behind others that are doing much of the hard work for them.

## 8. Simple code for Table 1

Table 2 uses a number of hubs to provide one or more implementations of the apps in Table 1. There is a README file with short-cut links[ag] that point directly to separate stand-alone subdirectories in the github for each check mark ($\surd$) in Table 2. To encourage users to study these examples in whichever order they prefer, we have avoided dependencies across subdirectories. Each subdirectory has its own README with instructions for installation and usage. Most subdirectories can be installed with the same one-line pip command that references requirements.txt.

As mentioned above, the solutions are short: 10–100 lines of code per example. Figure 2 shows an implementation of sentiment analysis based on PaddleHub. Many of the programs in the github are similar to Figure 2. One needs to know which model to load, and how to call it, and how to retrieve the results. Since there are so many ways to do this, some hubs are introducing abstractions such as pipelines[ah] and auto classes[ai] to hide much of the complexity. Many of the examples in the github make use of these abstractions.

[z] https://www.youtube.com/watch?v=YUUhDoCx8zc
[aa] https://www.paddlepaddle.org.cn/hub
[ab] https://github.com/PaddlePaddle/PaddleNLP
[ac] https://huggingface.co/
[ad] https://github.com/pytorch/fairseq/blob/master/examples/translation
[ae] https://techcrunch.com/2021/03/11/hugging-face-raises-40-million-for-its-natural-language-processing-library/
[af] https://www.pelotonadvisory.com/advisor-topics/lessons-from-the-peloton/
[ag] https://github.com/kwchurch/deepnet_examples#short-cuts
[ah] https://huggingface.co/transformers/main_classes/pipelines.html
[ai] https://huggingface.co/transformers/model_doc/auto.html

```
import sys
import paddlehub as hub
senta = hub.Module(name="senta_bilstm")
texts = [line.rstrip() for line in sys.stdin.readlines()]
for res in senta.sentiment_classify(texts=texts,
                                     use_gpu=False,
                                     batch_size=1):
    print('\t'.join([res['text'],
                     res['sentiment_key'],
                     str(res['positive_probs'])]))
```

**Figure 2.** Sentiment analysis using PaddleHub.

```
model_string = 'Helsinki-NLP/opus-mt-' \
               + args.source_language \
               + '-' + args.target_language
model = AutoModelWithLMHead.from_pretrained(model_string)
tokenizer = AutoTokenizer.from_pretrained(model_string)
for line in sys.stdin:
    sent = line.rstrip()
    if len(sent) <= 0: continue
    toks = tokenizer.encode(sent, return_tensors="pt")
    res = model.generate(toks,
                         max_length=400,
                         num_beams=40,
                         early_stopping=True)
    print(tokenizer.decode(res[0]))
```

**Figure 3.** Translate between many language pairs.

## 9. Priorities and hubs for deep nets

Different hubs for deep nets have different priorities:

1. PaddleHub and PaddleNLP emphasize the Chinese language,
2. Fairseq does well in competitions such as Workshop on Machine Translation,[aj] and
3. HuggingFaceHub has more models and datasets.

HuggingFace has more models than the others because their hub encourages contributions from third parties. They offer a simple mechanism to push your model to their hub.[ak] The large number of models makes it easy to translate between many pairs of languages, as illustrated in Figure 3. There are currently 1329 models that start with the prefix: helsinki-NLP/opus-mt (Tiedemann and Thottingal 2020). The names of these models typically end with the source and target language. In this way, our code based on HuggingFaceHub[al] covers more language pairs than alternatives based on PaddleHub[am] and Fairseq.[an]

[aj] http://www.statmt.org/wmt19/

[ak] https://huggingface.co/transformers/model_sharing.html

[al] https://github.com/kwchurch/deepnet_examples/blob/main/pretrained/examples/HuggingFaceHub/inference/translate/translate.py

[am] https://github.com/kwchurch/deepnet_examples/blob/main/pretrained/examples/PaddleHub/inference/translate/translate.py

[an] https://github.com/kwchurch/deepnet_examples/blob/main/pretrained/examples/Fairseq/inference/translate/translate.py

On the other hand, the alternatives may perform better for certain pairs since different hubs have different priorities for different languages. And therefore, one should try to avoid writing your code in a way that locks you into one hub and prevents you from using other hubs. Much has been written on the trade-offs between buying from a single source and buying from multiple vendors.[ao] So too, there is a similar trade-off between taking a dependency on single hub and making the effort to take advantage of multiple hubs.

While hubs make it easy to make use of many models and datasets, there are opportunities to make it even easier. Users hate connector hell.[ap] Before Universal Serial Bus (USB), most devices came with their own incompatible connectors and power cords. USB is a promising step in the right direction, but if USBs were truly universal, why do we need so many adaptors between USB-A, USB-B, USB-B mini, USB-B macro, USB-C, and lightning?[aq]

Hubs are a bit-like USB; there are too many ways to connect models and datasets, and too many inconsistencies in terminology (let alone semantics). Hubs are beginning to provide some abstractions such as pipelines and auto classes to hide some of this complexity, though it should be possible to hide even more complexity behind APIs such as Table 1. The next section discusses a proposal for hiding some of the complexities between datasets.

## 10. cat_dataset

To use a dataset, users currently need to know quite a few details about json structures, names of splits, and names of configurations. The hubs provide tools such as the dataset explorer and model explorer to make it easier for users to figure this out,[ar] but given how many datasets there are, it is too much work to figure this out for each dataset. All too often, users special case their code for a small number of datasets, making it difficult to run someone else's code on some other dataset (on novel inputs). Non-programmers should be able to run code written by others on novel inputs without the need to modify the code.

*cat_dataset* makes it easier to use more datasets. This program outputs many datasets on stdout in a standard format. Thus, one can combine *cat_dataset* with many of the inference apps in Table 1 using Unix pipes, avoiding the need to special case one's code for a small number of datasets. In this way, one can mix and match most datasets with most inference apps.

One can list all datasets with:

```
cat_dataset –list 1
```

To output the bookcorpus, say:

```
cat_dataset –dataset bookcorpus
```

To output the test split of the wikitext, say:

```
python cat_dataset.py –dataset wikitext\
–dataset_config wikitext-2-raw-v1\
–split test
```

*cat_dataset* outputs the contents of a dataset as a text file, using tab-separated fields. Each line is record. The first three columns are used for the name of the dataset, the configuration (often "None"), and the split. The remaining fields are a flattened representation of a json dictionary, where there is a tab-separated field for each key in dictionary. Each field is a pair of a key and a value delimited by a vertical bar.

[ao]https://spendmatters.com/2013/02/28/evaluating-supply-chain-risks-with-single-vs-multiple-vendor-sourcing-strategies/
[ap]https://www.electronicdesign.com/blogs/article/21799752/cable-and-connector-hell
[aq]https://www.tripplite.com/products/usb-connectivity-types-standards
[ar]https://huggingface.co/datasets/viewer/?dataset=wikitext

If the –split argument is not specified, *cat_dataset* will output all splits in the dataset. The github in footnote[c] provides two implementations of *cat_dataset*, one for datasets in HuggingFace and another for datasets in PaddleHub. The github contains a file[as] explaining which datasets require which –dataset_config arguments. The documentation[at] lists which datasets have been tested, with links into the dataset explorer.

## 11. Warning: Some datasets have been processed in horrible ways

Warning, when much of the content on hubs is provided by the community, it is hard to be as careful as lexicographers are about what came from where. For example, the documentation for the dataset, ptb_text_only, suggests that this dataset came from (Marcus *et al.* 1993), though it is clear from the name of the dataset, that the main contribution of (Marcus *et al.* 1993), the annotations, is no longer there. Moreover, if one reads the second sentence in the documentation (below) carefully, it becomes clear that many of the content words have been replaced with: $< unk >$.

*This is the Penn Treebank Project: Release 2 CDROM, featuring a million words of 1989 Wall Street Journal material. The rare words in this version are already replaced with token. The numbers are replaced with token.*[au]

Since hubs are as convenient as they are, authors may not be aware of important differences between different versions of these datasets. The version of the Penn TreeBank distributed by the Linguistic Data Corsortium[av] has no $< unk >$ processing, unlike the version from HuggingFaceHub. After processing, some versions have a very small vocabulary of just 10k types. Some versions have also been normalized in other ways to simplify numbers and punctuation. Note that the Penn TreeBank is based on the Brown Corpus, and yet the Brown Corpus (Francis and Kučera 1982) has a much larger vocabulary of 50k types.

Such processing has consequences for estimates of perplexity (and more).[aw] There are quite a few papers that report perplexity on various versions of the Penn TreeBank[ax] (and other corpora), though it is not always clear which papers are referring to which version.

Moreover, it can be risky to compare such numbers since $< unk >$ processing has dramatic consequences for estimates of perplexity. Perplexity is an estimate of how hard it is to guess the next word. If we replace many of the rare words with $< unk >$, then it is easier to guess $< unk >$ than the rare word that was originally there. In the limit, if we replace every word with $< unk >$, then perplexity becomes trivial (and useless). It is not clear why the literature is so interested in estimates of perplexity on corpora with artificially small vocabularies due to $< unk >$ processing (and normalization of numbers and punctuation).

## 12. Bertology

### *12.1 Unmasking*

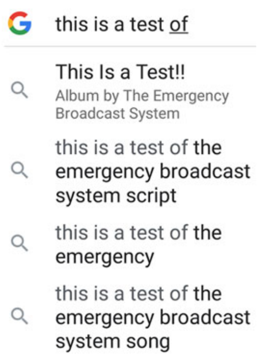
The previous sections have discussed how to use much of the code on the github to address tasks in Table 1. The rest of this paper attempts to provide an intuitive sense of what these nets are doing.

Much has been written on "bertology" (Rogers *et al.* 2020). What are transformers doing? Transformers, deep nets such as BERT and ERNIE, can be used to predict the next word, somewhat like auto-completion in a search engine, as shown in Figure 4. It may be helpful to

[as]https://github.com/kwchurch/deepnet_examples/blob/main/datasets/HuggingFace/dataset_config.txt
[at]https://github.com/kwchurch/deepnet_examples/tree/main/datasets/HuggingFace
[au]https://huggingface.co/datasets/ptb_text_only
[av]https://catalog.ldc.upenn.edu/LDC99T42
[aw]https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/
[ax]https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

**Table 3.** Unmasking: replace each input word with [MASK] and predict fillers. Red is added to highlight differences between the top prediction and the original input

| Word | Rank 0 | Rank 1 | Rank 2 |
|---|---|---|---|
| This | this:0.595 | it:0.375 | there:0.004 |
| is | was:0.628 | is:0.334 | included:0.007 |
| a | a:0.935 | another:0.029 | the:0.023 |
| test | part:0.253 | subset:0.125 | variation:0.082 |
| of | of:0.886 | for:0.070 | on:0.017 |
| the | the:0.883 | an:0.088 | our:0.005 |
| emergency | ratio:0.257 | television:0.048 | satellite:0.031 |
| broadcast | braking:0.620 | response:0.070 | management:0.024 |
| system | system:0.244 | technique:0.077 | capability:0.059 |
| . | .:0.969 | ;:0.029 | !:0.001 |


this is a test of

This Is a Test!!
Album by The Emergency Broadcast System

this is a test of the emergency broadcast system script

this is a test of the emergency

this is a test of the emergency broadcast system song

**Figure 4.** Auto-completion from a search engine.

compare auto-completion with Table 3, produced by *unmasking*. A short program for unmasking can be found on the github.[ay] Unmasking replaces each input word with **[MASK]** and predicts the $n$ best fillers. Note that the predictions in Table 3 are remarkably good, though not as good as auto-completion (Figure 4), at least in this case.

Table 4 applies unmasking to a few cliches.[az] Each of these cliches has at least one incorrect prediction, highlighted in red, except for *better safe that sorry*. Several out of vocabulary words (OOVs) are replaced with function words.

It is surprising that BERT is not better than this at filling in the blank in highly predictable contexts such as cliches. It seems like this should be an easy task. Alternatives based on nearest neighbors and/or web search may be more effective than BERT, at filling in missing pieces of cliches, at least in some cases.

### 12.2 Calibration

Unmasking outputs scores as well as fillers. Can we interpret these scores as probabilities? Calibration results, shown in Figure 5, suggest the scores are too high.

[ay] https://github.com/kwchurch/deepnet_examples/tree/main/pretrained/examples/HuggingFaceHub/Bertology/unmask
[az] https://prowritingaid.com/art/21/List-of-Clich\%C3\%A9s.asp

**Table 4.** Unmasking applied to cliches. Incorrect predictions are highlighted in red

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| MWEs | are | a | pain | in | the | neck | for | NLP | . |
| there:0.5 | is:0.4 | a:0.4 | pain:0.2 | in:1.0 | the:1.0 | ass:0.9 | in:0.6 | me:0.1 | .:1.0 |
| | | better | safe | than | sorry | . | | | |
| | | better:1.0 | safe:1.0 | than:1.0 | sorry:1.0 | .:1.0 | | | |
| | | read | between | the | lines | . | | | |
| | | in:0.1 | between:0.8 | the:0.9 | lines:0.9 | .:1.0 | | | |
| | | play | your | cards | right | . | | | |
| | | play:0.8 | the:0.5 | cards:0.8 | right:0.4 | .:1.0 | | | |
| | | it's | an | uphill | battle | . | | | |
| | | in:0.3 | an:1.0 | epic:0.5 | battle:0.4 | .:1.0 | | | |
| you | can't | judge | a | book | by | its | cover | . | |
| you:0.9 | can:0.6 | judge:0.8 | a:0.8 | book:0.8 | by:1.0 | its:0.9 | quality:0.1 | .:0.9 | |
| | | ignorance | is | bliss | . | | | | |
| | | this:0.4 | is:0.4 | bliss:0.3 | .:1.0 | | | | |
| the | grass | is | always | greener | on | the | other | side | . |
| the:1.0 | land:0.2 | was:0.8 | much:0.6 | planted:0.1 | on:0.9 | the:1.0 | other:0.2 | side:0.8 | .:0.9 |

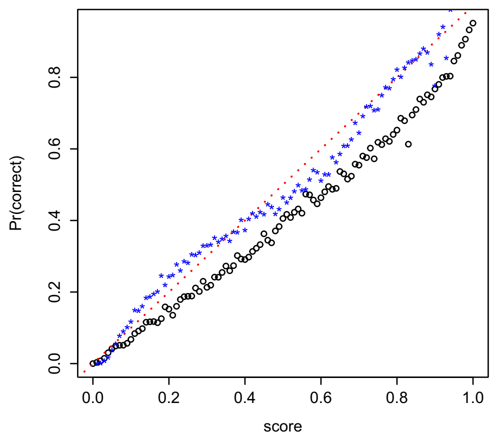


**Figure 5.** Calibration shows scores are too high (**black** points are below red line).

Calibration extracts the best filler (rank 0) and its score for each word in the WikiText test set. Outputs from unmasking are posted on the github.[ba] This output consists of 238k rows, one row for each word in the test set. In addition to the original word in column 1, there are predicted fillers with scores in the other columns.

[ba] https://github.com/kwchurch/deepnet_examples/blob/main/pretrained/examples/HuggingFaceHub/Bertology/unmask/wikitext-103-raw-v1.test.unmasked.gz

We compute a boolean value, *correct*, which is 1 iff the top ranking filler is the same as the original word (ignoring differences between upper and lower case). There are 238k values for correct, one for each row. Each row also has a score for the top ranking filler.

Calibration uses binning to compare scores with *correct*, that is, we create 100 bins for values of score between 0 and 1. Use the score to assign each of the 238k rows to the 100 bins. For each bin, estimate $Pr(correct)$ by averaging *correct* over the rows in the bin.

Calibration results are shown in Figure 5. The **black** points are the result of the binning process described above. If the scores were well calibrated, the points would be close to the dashed red line. The fact that most of the **black** points are below the red line suggests the scores are often too high: $score > Pr(correct)$.

Logistic regression can be viewed as an alternative to binning, that is, use logistic regression to fit, $correct \sim score$. This will find coefficients $\beta_0$ and $\beta_1$ for $z = \beta_0 + \beta_1 score$. $z$ can be used to estimate $Pr(correct)$, that is, $Pr(correct) \approx \sigma(z)$, where $\sigma(z) = 1/(1+e^{-z})$.

Results of calibration with logistic regression are shown in Figure 5, though we added an additional feature to take advantage of unigram word frequencies, *freq*. That is, instead of $correct \sim score$, we used equation (1):

$$correct \sim score + log(1 + freq) \tag{1}$$

The two features, *score* and *freq*, both depend on candidate fillers. Word frequencies are computed from the training set. Thus,

$$z = \beta_0 + \beta_1 score + \beta_2 log(1 + freq) \tag{2}$$

where $Pr(correct) \approx \sigma(z)$.

Logistic regression is used to fit the three coefficients: $\beta_0$, $\beta_1$, $\beta_2$. All three coefficients turn out to be significant, suggesting that both features, *score* and *freq*, are useful.

The blue stars in Figure 5 show mean *correct*, averaged over bins based on $\sigma(z)$. The fact that the blue stars are closer to the dashed red line suggests that calibrated scores, $\sigma(z)$, are better than estimates of *Pr(correct)* raw scores directly from BERT without calibration.

Why did we include unigram frequencies in the calibration? We guessed that frequency might be helpful because of a well-known word frequency effect in psycholinguistics.[bb] Eye balling some of the output described in footnote[ba] suggested that BERT scores increase with word frequency. Figure 6 confirms this relation. We hypothesized the relation might be helpful for calibrating scores. Testing the coefficients, $\beta_0$, $\beta_1$, $\beta_2$, for significance confirmed this hypothesis, that is, BERT scores are too high, and the errors depend on the frequency of the candidate fillers.

Why are raw BERT scores too high? We suspect these scores are based on a softmax. The denominator in the softmax should not only include values for the top candidates, but there should also be some mass for none of the above (NOTA), since there are many less likely candidates. While each of these unlikely candidates may be unlikely, in aggregate, their mass can be sizeable and may be too large to ignore.

Moreover, we suspect that *freq* is significant in the regression because the NOTA mass depends on *freq*. The NOTA mass tends to be small for function words, where the first few candidates cover most of the possibilities, whereas, for content words, there can be a much longer tail of plausible fillers, including many OOVs. Thus, the NOTA mass tends to be larger for infrequent content words than more frequent function words.

To summarize this section, we discussed the unmasking tool in the github. This tool predicts fillers for each input word, by replacing each word with **[MASK]** and running BERT. We showed some anecdotal examples suggesting that although these predictions are remarkably good, they may not be as good as alternatives from web search, at least for auto-completion tasks. We also looked at some examples of filling in missing words in cliches. The errors can help us understand

[bb]https://en.wikipedia.org/wiki/Word_frequency_effect

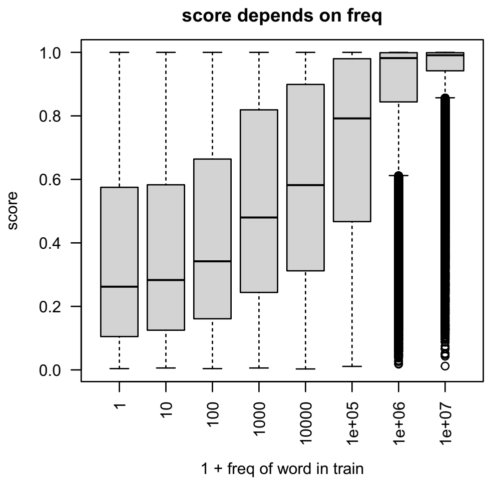


**Figure 6.** BERT scores increase with frequency of candidate fillers. Frequencies are estimated from training set.

what BERT is doing, and what it is not doing. Nearest neighbor methods would probably be more effective at filling in the blanks in highly predictable contexts such as cliches, and therefore, BERT is probably doing something other than nearest neighbor search. Finally, we used calibration to show that raw scores from BERT are probably too high, and that the errors depend on frequency, suggesting that BERT is probably not taking advantage of frequency as much as it could. Despite these concerns, transformer models such as BERT and ERNIE are the method of choice for many tasks, and they are doing very well on many leaderboards.[bc]

## 13. Conclusions

Deep nets have done well with early adopters, but the future will soon depend on crossing the chasm and engaging a broader market. The goal of this paper is to make deep nets more accessible to users with little or no programming skills, and people with little interest in training new models.

We discussed a github (see footnote[c]) with simple implementations of the API in Table 1. The emphasis is on simplicity, brevity, and instant gratification, as opposed to SOTA performance and computational resources. We used to worship wizard programmers that could perform magic, but we should have been more concerned with cleaning up and making the mess more accessible to the masses. Simplicity and accessibility need to be designed into the final product from the very beginning, but all too often users are treated as an after-thought.

Non-programmers should be able to install these programs and use them in 15 minutes or less (per program). Users with modest programming skills should be able to read the code since it is short: 10–100 lines per program. Much of the code is based on abstractions such as pipelines and auto classes that hide much of the complexity. Models and datasets are provided by hubs: PaddleHub, PaddleNLP, HuggingFaceHub, and Fairseq.

Hubs have different priorities than research. Research is training new models, with an emphasis on SOTA performance and computational resources. But users are already overwhelmed with an embarrassment of riches (13k models and 1k datasets). Do users want more? Or do they want to do more with what we already have?

The last section, Section 12, provides some intuition about what deep nets are doing, and what they are not doing. An unmasking tool was introduced that replaces each input word with **[MASK]** and predicts the *n* best fillers. Many of the predictions are remarkably good, though it

[bc]https://gluebenchmark.com/leaderboard

is easy to find exceptions. We provided some anecdotal evidence, as well calibration results, suggesting some opportunities for improvement. Nearest neighbor methods and auto-completion in web search may be better than transformers, at least for filling in the blank in predictable contexts such as cliches.

## References

**Aho A.V. and Kernighan B.W. and Weinberger, P.J.** (1979). Wiley Online Library, USA.

**Bishop C.M.** (2006) *Pattern Recognition and Machine Learning.* Springer Science.

**Bronstein M.M., Bruna J., Cohen T. and Veličković P.** (2021). Geometric *Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*, 2104.13478, arXiv.

**Chollet F.** (2017). *Deep Learning with Python.* Simon and Schuster.

**Devlin J., Chang M.-W., Lee K. and Toutanova K.** (2019) BERT: *Pre-training of Deep Bidirectional Transformers for Language Understanding*. ACL, pp. 4171–4186.

**Géron A.** (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media.

**Goodfellow I., Bengio Y. and Courville A.** (2016). MIT Press.

**Elliott R.J. and Lesk M.** (1982). *Route Finding in Street Maps by Computers and People.* AAAI, pp. 258–261

**Francis N. and Kučera H.** (1982). *Frequency Analysis of English Usage: Lexicon and Grammar.* Houghton Mifflin.

**Kernighan B.W.** (2019), *UNIX: A History and a Memoir*, Kindle Direct Publishing.

**Kernighan B.W., Lesk M.E. and Ossanna J.F.** (1978) *The Bell System Technical Journal*, **57**, 2115–2135.

**Kernighan B.W. and Pike R.** (1999). Addison-Wesley, USA.

**Kernighan B.W. and Plauger P.J.** (1974). *Elements of programming style.* McGraw-Hill.

**Kernighan B.W. and Ritchie D.M.** (1988). *The C Programming Language.* Englewood Cliffs, USA: Prentice Hall.

**Lesk M.** (1969). Word-word associations in document retrieval systems. *American documentation* 20, 27–38.

**Lesk M.** (1976). *Lex: A Lexical Analyzer Generator.* Murray Hill, NJ: Bell Laboratories.

**Lesk M.** (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation*, pp. 24–26.

**Marcus M., Santorini B. and Marcinkiewicz M.A.** (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* **19**, 263–311.

**McIlroy M., Pinson E.N. and Tague B.A.** (1978). UNIX time-sharing system. *The Bell System Technical Journal* **57**, 1899–1904

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

**Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M. and Duchesnay E.** (2011). Coders at work: Reflections on the craft of programming, Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830.

**Rajpurkar P., Zhang J., Lopyrev K. and Liang P.** (2016). *SQuAD: 100,000+ Questions for Machine Comprehension of Text.* EMNLP, pp. 2383–2392

**Rogers A., Kovaleva O. and Rumshisky A.** (2020). A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics* **8**, 842–866.

**Salton G. and Lesk M.E.** (1968). Computer evaluation of indexing and text processing. *Journal of the ACM (JACM)* **15**, 8–36.

**Seibel P.** (2009). *Coders at Work: Reflections on the Craft of Programming.* New York: Springer-Verlag.

**Sun Y., Wang S., Li Y., Feng S., Tian H., Wu H. and Wang H.** (2019). *Ernie 2.0: A Continual Pre-Training Framework for Language Understanding.* AAAI.

**Tiedemann J. and Thottingal S.** (2020). OPUS-MT–building open translation services for the world. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pp. 479–480.

**Vig J.** (2019). *A Multiscale Visualization of Attention in the Transformer Model.* ACL, pp. 37–42. https://www.aclweb.org/anthology/P19-3007

**Wolf T., Debut L., Sanh V., Chaumond J., Delangue C., Moi A., Cistac P., Rault T., Louf R., Funtowicz M., Davison J., Shleifer S., von Platen P., Ma C., Jernite Y., Plu J., Xu C., Le Scao T., Gugger S., Drame M., Lhoest Q. and Rush A.M.** (2019). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*, arXiv:1910.03771

**Cite this article:** Church KW, Yuan X, Guo S, Wu Z, Yang Y and Chen Z (2021). Emerging trends: Deep nets for poets. *Natural Language Engineering* **27**, 631–645. https://doi.org/10.1017/S1351324921000231