KerasBERT: Modeling the Keras Language

Connor Shorten
Florida Atlantic University
Boca Raton, Florida
cshorten2015@fau.edu

Taghi M. Khoshgoftaar
Florida Atlantic University
Boca Raton, Florida
khoshgof@fau.edu

***Abstract*—We introduce a new application domain to evaluate the knowledge retention of language models. Our model, KerasBERT, is trained on the Keras code documentation. This is a unique challenge of language modeling small datasets, as well as combining natural language and code data. We evaluate how well KerasBERT learns the Keras Deep Learning framework through cloze test evaluation. We present miscellaneous properties of these cloze tests such as mask positioning and prompt paraphrasing. KerasBERT is an 80 million parameter RoBERTa model, which we compare to the Zero-Shot learning capability of the 6 billion parameter GPT-Neo model. We present a suite of cloze tests crafted from the Keras documentation to evaluate these models. We find some exciting completions that show KerasBERT is a promising direction for question answering and schema-free database querying. We conclude our work by discussing some future directions for KerasBERT and the development of language models for code documentation support.**

***Index Terms*—Natural Language Processing, Language Model, Keras, BERT, Cloze Testing**

I. Introduction

Language modeling has been an impressive success of self-supervised learning. The language modeling learning algorithm begins with large collections of text as input, processing them into token sequences (typically of 512 tokens), randomly masking out tokens in sampled sequences, and tasking the model to predict what the original token had been before it was masked out. For example, beginning with the sentence "I am going outside for a walk", masking "outside" such that the sentence is "I am going [MASK] for a walk", and tasking the model to predict outside as the replacement for the [MASK] token. Doing this with large-scale models and data has resulted in models that can generate fluent text by placing the mask out at the end of a text prompt. More practically, these language models have been used as a pre-trained initialization point for fine-tuning of a supervised learning task. Raffel et al. [16], [34] demonstrated the T5 model. T5 showed that language models could be trained with supervised learning tasks in a similar interface as language modeling by mapping the masked token prediction to a candidate class label. Of the supervised learning tasks unified in the T5 framework, such as natural language inference or text classification, question answering is an especially interesting probe to the knowledge contained in language models.

Roberts et al. [12] explored the question of: how much knowledge can you pack into the parameters of a language model? This study highlights the ability of language models to compress large amounts of text into neural network weights, retrieving information through mask prompts. Mask prompts, as demonstrated earlier in the example of "I am going [MASK] for a walk", can serve as a template to test question answering ability. Thorne et al. [17] seek to not only compress text data through neural networks, but use them to answer questions typically reserved for database languages. Rather than meticulously enter data into pre-defined data schema, neural databases built on language models can perform complex queries solely with mask prompts. These are very exciting directions for language modeling that we explore with our research in KerasBERT. We collect text data describing how to use Keras, such as documentation and coding tutorials. We model this data using the HuggingFace transformers, tokenizers, and trainer APIs. We build our models and tokenizers from scratch with random, rather than pre-trained initializations. We provide a comparison with pre-trained models and tokenizers for reference. We also compare our models trained from scratch with the pre-trained GPT-Neo models [21].

Language models are typically trained on large datasets such as Wikipedia. These large corpora help the model generalize better to the form of language, since it has seen a large diversity of writing style. However, these datasets require a very broad range of knowledge retention for investigating their ability to answer specific questions. We believe more specific datasets such as the Keras documentation serves as a better testbed for factual retention and flexibility. This type of specific dataset has seen a surge of interest using biomedical papers as the dataset [18]. For the sake of designing mask prompts, also referenced as cloze tests [32], this requires researchers to have a deep understanding of both Deep Learning and the Biomedical domain. Designing tests about Deep Learning, such as the details of the Keras framework, is better aligned with our expertise as Deep Learning researchers. This helps us push the limits of knowledge-intensive tests for language models. Our dataset contains 2.5 MB of text made up of 49,876 lines separated by a newline character from 277 unique documents. With our custom tokenizer, which we describe later, this amounts to 678,800 unique tokens. This dataset is much smaller than the datasets used to train BERT [3] or GPT [14]. For comparison, GPT-3 [14] is trained on roughly 500 billion tokens. This data scale also enables us to experiment with more economic models since we do not need billions of parameters to model this amount of data.

Applying techniques from Natural Language Processing (NLP) to code data has seen a surge in interest. Hendrycks

et al. [9] recently demonstrated that transformers [19] could solve interview-style algorithmic challenges with Python code. These models are prompted with a natural language description of the coding challenge and generate the solution in Python code. This is done with the masking interface and a large pre-trained GPT-Neo model [21]. Further benchmarks for Deep Learning with code data include CodeXGLUE [11]. CodeXGLUE is a collection of 10 tasks such as code repair, code translation, and documentation translation, to name a few. We present the KerasDocs dataset as a benchmark for language modeling documentation for code libraries. This is a useful application for developers looking to use small libraries and ask particular questions about it. This presents new challenges for small dataset modeling and avoiding overfitting.

We collect our dataset from the Keras documentation page [22]. We found three separate styles of documentation for our dataset. These include the API documentation (which we term KerasAPI), developer guides (DevGuides), and code examples (CodeExamples). Each flavor of documentation has a unique style that we view as a useful proxy for constructing validation sets. The KerasAPI has the most formal structure with a sequential presentation of an object and its attached methods and attributes. The DevGuides and CodeExamples have a similar style to each other, but the CodeExamples are much more likely to reference something not entirely contained in the Keras library, such as a description of the Reptile algorithm from OpenAI [23]. We model these datasets with an 80 million parameter RoBERTa model [2].

We additionally test the Zero-Shot learning ability of GPT-Neo on Keras cloze tasks. Zero-Shot learning describes reporting the performance of a pre-trained model without any additional training. GPT-Neo is pre-trained by language modeling an 800 GB text dataset known as The Pile [6]. We evaluate GPT-Neo on Keras knowledge without any fine-tuning on the Keras datasets. Zero-Shot learning is distinct from Few-Shot learning in which a very small amount of training data is provided. In GPT models, the demonstrations for Few-Shot training are typically prepended to the context. In Figure 9, we show some examples of prompting, where some context is added to the input, but we do not test Few-Shot learning where demonstrations of Keras knowledge retention are prepended to the input. We only compare GPT-Neo's Zero-Shot learning with our BERT-style models on rightmost token completion. This is due to the difference in task formulation of bidirectional context (BERT) versus left-to-right generation (GPT). Bi-directional context refers to optionally sampling the middle of the sequence for mask replacement, whereas left-to-right training always masks out the rightmost word. We describe the context distinction between these models further in our Discussion and analysis of the study.

We initially report the sparse categorical cross entropy loss on the validation data set through training. Our experiments of leaving one category of documentation data out of the training set consistently have a trend of initial improvement, but experience quick saturation. We continue to qualitatively inspect mask completions, demonstrating that KerasBERT has clearly "memorized" some details of the Keras programming language. Furthermore, we manually construct cloze tests from the API guide to the Model API. Our manual prompts represent a distribution shift in writing style compared to random masking in the validation setting. We divide these cloze tests based on salient masking, where we use our domain knowledge to select the most knowledge-intensive token to replace, as well as the leftmost, rightmost, and random masking baselines.

An important issue in Deep Learning is reliance on spurious correlations to make predictions. In Natural Language Processing, this topic has manifested as the debate between meaning and form [1]. We highlight this problem with our KerasBERT model, firstly showing that paraphrases of the cloze task dramatically hinder performance, and secondly that KerasBERT cannot solve the same cloze task when reformulated as a question with the rightmost mask position. However, we show that GPT-Neo is much more robust to this, demonstrating that model scale and training data size could be the solution to this particular issue. Although GPT-Neo performs well at replacing the rightmost token, we inspect longer outputs and observe heavy hallucination. Hallucination [28] is used to describe text generation that is not grounded in factual knowledge, such as recommending an article with a broken URL. Similar to KerasBERT, GPT-Neo also shows similar biases to particular forms of language. In our discussion section, we present future directions to achieve better robustness to the question form.

KerasBERT is an exploration into the potential of language models as question answering systems, knowledge graph aids, and schema-free databases. KerasBERT focuses on the application domain of code documentation, a salient case study for question answering. We provide a brief background on related works in language models for question answering, language modeling small datasets, and deep learning with code data. We describe how we collected these datasets and our modeling results using different subsets for validation. We then evaluate these models on manually crafted cloze tests. These first models are trained from scratch with a custom tokenizer, we report the difference with a pre-trained model and tokenizer, finding similar performance. We continue to compare our models trained from scratch with the GPT-Neo model. We further illustrate how GPT-Neo responds to adding additional context, also known as prompting, to Keras knowledge tests. We end with a discussion on tokenization, meaning versus form, and BERT versus GPT models. Our contributions in KerasBERT are as follows:

- We demonstrate the efficacy of Byte-Pair Encoding (BPE) and an 80M parameter RoBERTa Transformer for masked language modeling of the Keras Documentation through learning curves, qualitative evaluation, and manually designed cloze tests.
- We divide the Keras Documentation into semantic splits and show the state of generalization from one split to another.
- We compare the performance of using a pre-trained RoBERTa and tokenizer with our custom tokenizer and

randomly initialized models.
- We construct carefully crafted cloze, or fact, tests to probe the knowledge of KerasBERT. This additionally includes rightmost masking to compare with GPT-Neo's Zero-Shot learning ability of Keras documentation.
- We showcase longer generations with GPT-Neo where we insert additional context to the input, also known as prompting.
- We discuss ideas for improving robustness to variation in contexts which are used in knowledge testing, as well as how to improve the state of language modeling with small datasets more generally, such as code library documentation.

II. Related Work

A. Language Models for Question Answering

KerasBERT explores the efficacy of language models for question answering. Roberts et al. [12] explore how much knowledge can be packed into the parameters of a language model. The study evaluates T5 models up to 11 billion parameters on question answering datasets such as Natural Questions [29]. Khashabi et al. [15] unified all question answering tasks into a similar interface for adapting language models. Thorne et al. [13] further explore questions formatted as database queries such as aggregate operations. Brown et al. [14] demonstrated that a transformer scaled up to 175 billion parameters could answer questions and perform tasks without supervised learning, also known as Zero-Shot learning.

B. Language Modeling Small Datasets

Language modeling is typically used for transfer learning [7], in which the representations learned from language modeling are then fine-tuned for a downstream supervised learning task. In application to transfer learning, most works focus on pre-training with large datasets. For example, BERT [3] was trained with 3.3 billion words from BooksCorpus [8] and Wikipedia [24]. Later works showed the efficacy of pre-training with more domain relevant datasets [25]. This led to models such as BioBERT [26] and SciBERT [27] that model datasets containing biomedical scientific papers, rather than arbitrary chunks of text from the internet. However, BioBERT does not face the problem of small data, collecting 13.5 billion words from PubMedCentral (PMC) [30] full-text articles. SciBERT additionally has access to a large dataset of 1.14 million scientific papers. This is also relevant for fine-tuning models on specific portions of knowledge such as literature mining applications specific to COVID-19 [18], [43] or clinical reports [42].

C. Deep Learning with Code Data

Our work is inspired by successes in Deep Learning and NLP techniques with code data. Hendrycks et al. [9] prompt GPT-Neo with interview-style coding challenges in natural language. GPT-Neo in turn generates Python code and passes 20% of the introductory level coding challenges. In a similar vein as the GLUE benchmark for NLP [10], Lu et al. [11] unify Deep Learning with code tasks such as clone detection, defect detection, cloze testing, code completion, code repair, code translation, natural language code search, text to code generation, code summarization, and documentation translation. Feng et al. developed CodeBERT [31] trained on many different programming languages, including 460K tokens of Python code.

III. Experiments

The first step in the NLP pipeline is tokenization. This describes constructing atomic units of language, referred to as tokens, and mapping these tokens to their indexed position in an embedding table. Our experiments use the Byte-Pair Encoding (BPE) scheme, implemented in the HuggingFace tokenizers library [33]. At a high-level, BPE [5] builds the token dictionary bottom-up, looking at co-occurrence frequencies to merge tokens. We build up a brand new token dictionary for our problem to account for the token distribution mismatch between models pre-trained on datasets such as Wikipedia and our mix of Keras and Python code.

We experiment with an 80 million parameter RoBERTa Transformer model [2]. RoBERTa has some architectural changes to the BERT architecture, but shares the namespace category of "BERT" models since they are both trained with bi-directional masked language modeling, compared to left-to-right language modeling. Experimenting at the 80M parameter scale enables rapid experimentation between different data splits. We are mostly interested in the impact of data splits and resulting styles of cloze test such as modeling KerasCodeExamples and testing knowledge about the API documentation. We also test the GPT-Neo 6 billion parameter model's Zero-Shot learning ability. GPT-Neo is an open-sourced 6 billion parameter model trained on 400 GB of text from the Pile dataset. The Pile contains 95 GB of text from GitHub, which we suspect contains a significant amount of TensorFlow and Keras code.

Our dataset consists of three main subsets, KerasAPI, KerasCodeExamples, and KerasDevGuides. The KerasAPI contains 10 main folders, Models API, Layers API, Callbacks API, Data Preprocessing, Optimizers, Metrics, Losses, Built In Small Datasets, Keras Applications, and Utilities. This amounts to a total of 183 pages of documentation, with a large majority (128) describing different layers in the Layers API. The KerasCodeExamples contains annotated use cases across applications categorized as: Computer Vision, Natural Language Processing, Audio, Timeseries, Graph, Structured Data, Reinforcement Learning, Generative Deep Learning, and Quick Keras Recipes. The KerasCodeExamples contains 80 examples, with a majority (34) of Computer Vision examples. The KerasDevGuides contains 14 tutorials. These tutorials are longer than the average page in the KerasAPI or example in CodeExamples. Despite only consisting of 14 pages, KerasDevGuides contains 341 KB of text data, which is 13.5% of the overall dataset. Figure 1 illustrates the size of each subset when concatenated together to form the "AllData" superset.

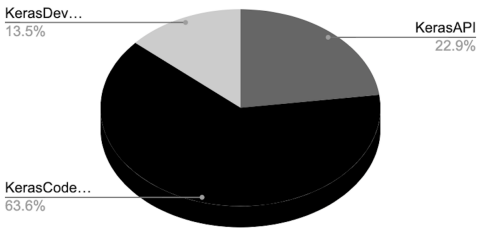

Fig. 1. Relative sizes of each data split.

Our experiments use the HuggingFace [33] Transformers, Tokenizers, and Trainer APIs. The KerasBERT models have a vocabulary size of 30,000 tokens and contain 8 hidden layers, each with 12 attention heads, totaling 80 million parameters. The experiments with the pre-trained model have a vocabulary size of 50,265, which increases the parameter count to 96 million. Each sequence used for training contains 128 tokens. These sequences are processed with masked language modeling where a token is selected for mask replacement with 15% probability. All models are trained for 100 epochs with a batch size of 64. Each training run takes approximately 10 hours using a NVIDIA V100 GPU hosted on Google Colab Pro. We plan to release the datasets used to train these models upon publication for the sake of reproducibility. We do not train the GPT-Neo variant, GPT-J-6B, at all. We sample from the provided demo API [20] at a value of 0.9 for Top-P and 1 for Temperature.

A. Learning Curves

Figure 2 shows the learning curves for the different validation splits of the Keras Documentation. We sanity check our experiments on the leftmost plot by showing the training curve using all available training data and evaluated on a subset of the training data. Somewhat interestingly, this shows that the model is performing better on the API subset than the other subsets. A possible explanation for this is the more consistent writing style of this data split. The other three curves illustrate leaving one dataset out of the training set for validation. We see a consistent trend of initial improvement and saturation. Even though the validation curve saturates in improvement, it does not decrease in performance which would indicate problematic overfitting. To summarize, our model trained on all available data achieves a sparse categorical cross entropy loss of 0.352 on the KerasAPI, whereas when we leave the KerasAPI out of the training data, this validation error is much higher at 3.89. The combination of the KerasAPI and the CodeExamples, performs the best on the leave-one-out test, with a categorical cross entropy loss of 2.77 on the KerasDevGuides. As shown in Figure 1, this is likely because this is the split with the most available training data. For this reason, we expect performance to improve as we collect more data in future work.

The representations KerasBERT learns from the training data generally improve validation performance. However, we want to drill into this and isolate mask completions that require knowledge retention compared to structural words such as "the" or "and". We further inspect performance by displaying examples of mask prompts directly extracted from the documentation. Qualitatively, we find that the model trained on all data performs far better, with the second best model having left out the Keras API documentation. We see that the models have not quite memorized the training data, and the subsets clearly impact these evaluations, as also shown in Figure 2. From these inspections, we can see that these KerasBERT models have retained some knowledge about the Keras programming language.

B. Manual Cloze Tests

We scale up these tests by constructing 222 cloze tests from the Model API documentation. We report the top-1 and top-5 accuracy of KerasBERT across different strategies of masking. We see that salient masking, in which we manually select the most knowledge intensive token, is much less successful than consistently masking out the rightmost token. This test highlights a drawback of automated test evaluation where the mask may not be useful for question answering particular to the Keras programming language.

C. Pretrained Model and Tokenizer

We also report the performance by taking the pretrained RoBERTa model "off-the-shelf" from the HuggingFace model hub. This model is slightly larger than the other models at 96M parameters, and has a significantly larger vocabulary size of 56,205. This vocabulary was also constructed from BPE, but on a significantly larger dataset. This larger dataset is limited in modeling co-occurrences particular to the Keras programming language, but is likely to be better for more common words that would appear in datasets such as Wikipedia. For example, the KerasCodeExamples, which represents a significant majority of our data, mostly contains natural language describing the code and application. As shown in Figure 8, we can see completions such as "code" that do not appear in our KerasBERT models trained from scratch. We find that fine-tuning the pre-trained checkpoint of RoBERTa, alongside the matching pre-trained tokenizer, achieves roughly the same performance. Figure 7 illustrates the similar learning curves of the pretrained 96M parameter RoBERTa-base, compared with our 80M KerasBERT. Both models are trained on all available data and evaluated on the KerasDocs validation as a sanity check of model capacity. We expect the custom tokenizer to surpass this model as we collect more data in future work.

D. GPT-Neo

As shown in Figure 6, GPT-Neo performs much better than the 80 million parameter KerasBERT model on the curated cloze tests. Qualitative inspection of GPT-Neo's outputs shows an impressive knowledge of language form more generally, such as completing prompts that list input arguments with bullet-point style summaries of each argument. However, GPT-Neo is far from perfect and we observe heavy hallucination

No Data Left Out (Sanity Check)	KerasDevGuides Left Out	KerasAPI Left Out	KerasCodeExamples Left Out
			

Fig. 2. Learning curves for different categories used in data splitting.

Input: The <mask> class groups layers into an object with training and inference features.

Ground Truth	Model
All Data	Model: 0.717 Keras: 0.025 Layer: 0.012 Preprocessing: 0.011 Wide: 0.01
KerasAPI Left Out	Model: 0.621 Other: 0.009 Split: 0.008 Correct: 0.007 Dataset: 0.007
DevGuides Left Out	Preprocessing: 0.32 Resulting: 0.29 FNet: 0.132 Default: 0.013 Encode: 0.011
CodeExamples Left Out	Layer: 0.121 Model: 0.1 Config: 0.066 Following: 0.038 Embedding: 0.025

Fig. 3. Cloze test inspection. The number to the right of the token are how much weight the model assigned to the candidate token completion on the left. For example, the All Data model assigned 71.7% probability to "Model" as the token replacement.

Input: A <mask> is an object that can perform actions at various stages of pretraining (e.g. at the start or end of an epoch, before or after a single batch, etc).

Ground Truth	Callback
All Data	Callback: 0.134 Mask: 0.085 Metric: 0.067 Model: 0.031 Compile: 0.025
KerasAPI Left Out	GAN: 0.391 Model: 0.216 Autoencoder: 0.018 Setup: 0.016 Callback: 0.013
DevGuides Left Out	Callback: 0.185 Layer: 0.029 Useful: 0.022 Way: 0.021 Simple: 0.021
CodeExamples Left Out	Layer: 0.3 Callback: 0.066 Example: 0.045 Dict: 0.014 Method: 0.013

Fig. 4. Cloze test inspection. The number to the right of the token are how much weight the model assigned to the candidate token completion on the left. For example, the All Data model assigned 13.4% probability to "Callback" as the token replacement.

(generated text that is not factually correct). Even though GPT-Neo infers to describe each argument in further detail, these descriptions are far from accurate. Another common hallucination is broken links to stackoverflow posts. GPT-Neo's performance on the cloze testing is impressive. However, inspecting the longer generations shows serious flaws. These longer generations are shown in Figure 9.

E. Prompting GPT-Neo

A common error in the Keras framework is when users try to call the summary method before compiling the model. This is common because users often want to get a quick look at the parameter count or see that the architecture is constructed correctly. Calling this method will throw an error if the user hasn't first "built" or compiled the model. This is counter intuitive to new Keras users and represents a good use case for KerasBERT or GPT-Neo to help. Unfortunately, this is an example that GPT-Neo gets very incorrect out of the box. In Figure 9, we illustrate how "prompting", or adding additional input context to the question, helps GPT-Neo in this example. We test the prompts "Information about the Keras programming language:", "Keras is a Deep Learning Framework built on top of TensorFlow", and "Question about the Keras programming language:" [Cloze Test] "what?". These examples illustrate the hallucination of GPT-Neo mentioned previously.

We note that this is not a fair comparison of training from scratch compared to the Zero-Shot learning ability of GPT-Neo, because our KerasBERT models are only tasked with completing one token rather than long generations. However, we have found that although GPT-Neo performs fairly well at the rightmost single token completion, long generations are far from ready for real-world application. Since GPT-Neo is a

Input: The Reptile algorithm was developed by OpenAI to perform model-agnostic meta-learning. Specifically, this algorithm is designed to quickly learn to perform new <mask> with minimal training (few-shot learning).

Ground Truth	tasks
All Data	Learning: 0.201 Models: 0.195 Networks: 0.048 Workflow: 0.047 tasks: 0.039
KerasAPI Left Out	tasks: 0.636 Data: 0.088 Performance: 0.048 Models: 0.018 Model: 0.012
DevGuides Left Out	Models: 0.17 tasks: 0.124 Dataset: 0.066 Learning: 0.055 Performance: 0.054
CodeExamples Left Out	Model: 0.4 Data: 0.14 Models: 0.092 Layers: 0.043 Inputs: 0.025

Fig. 5. Cloze test inspection. The number to the right of the token are how much weight the model assigned to the candidate token completion on the left. For example, the All Data model assigned 20.1% probability to "Learning" as the token replacement.

	KerasBERT - 80M	GPT-Neo
Salient Masking	1.4% top-1; 2.7% top-5	N/A
Rightmost Masking	3.6% top-1; 13.1% top-5	57.6% top-1

Fig. 6. Performance on manually crafted tests.

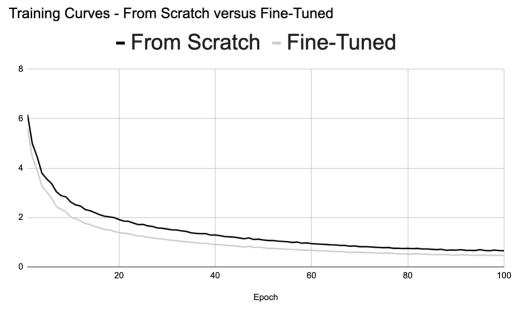

Fig. 7. The KerasBERT model trained from scratch with a random initializer and custom tokenizer has about the same learning curve as a slightly larger RoBERTa model fine-tuned from a pre-trained checkpoint and tokenizer.

Input: The [MASK] class groups layers into an object with training and inference features.

Ground Truth	From Scratch - All Data	Fine Tuned - All Data
Model	Model: 0.717 Keras: 0.025 Layer: 0.012 Preprocessing: 0.011 Wide: 0.01	Model: 0.19 Layer: 0.055 second: 0.35 Lookup: 0.03 Raw: 0.021

Input: The [MASK] is an object that can perform actions at various stages of pretraining (e.g. at the start or end of an epoch, before or after a single batch, etc).

Ground Truth	From Scratch - All Data	Fine Tuned - All Data
Callback	Callback: 0.134 Mask: 0.085 Metric: 0.067 Model: 0.031 Compile: 0.025	callback: 0.266 Model: 0.217 model: 0.081 metric: 0.065 batch: 0.034

Input: The Reptile algorithm was developed by OpenAI to perform model-agnostic meta-learning. Specifically, this algorithm is designed to quickly learn to perform new [MASK] with minimal training (few-shot learning).

Ground Truth	From Scratch - All Data	Fine Tuned - All Data
tasks	Learning: 0.201 Models: 0.195 Networks: 0.048 Workflow: 0.047 tasks: 0.039	models: 0.421 layers: 0.15 tasks: 0.052 model: 0.045 code: 0.143

Fig. 8. Cloze test completions comparing the KerasBERT model trained from scratch with the fine-tuned model.

6 billion parameter model, it is difficult to fine-tune and tweak for this task. Therefore, we believe it is more promising for the application of using language models as knowledge bases and question answering systems to train smaller models on the smaller datasets. With these smaller models, we can better integrate training interventions such as data augmentations [36], [37] or curriculum learning [35].

IV. Discussion

A. Tokenization

Tokenization is a critical component of this modeling pipeline. As a reminder, BPE builds tokens from the ground up, merging commonly co-occurring characters. As an example, our tokenizer breaks up "Keras is a programming language" into: "Keras", "is", "a", "program", "ing", "language". For the sake of comparison, the pre-trained RoBERTa tokenizer chunks this up into: "K", "er", "as", "is", "a", "programming", "language". Another interesting detail of our

Input: “The summary method of the Model class raises a ValueError if summary() is called before the model is ”

Ground Truth	built
No Prompt	Fitted. If you want to check the summary methods of the different models before calling fit() or predict(), then you will need to filter out the appropriate ones yourself. This is generally not recommended though.
Information about the Keras programming language:	Trained. Keras adds time to the x, y, and sample_weight Tensors before summary() is called. The actual time the summaries are calculated can vary from 0ms to thousands of ms based on the data to summary. If you call summary() before the model is ready, this method throws a ValueError.
Keras is a Deep Learning framework built on top of TensorFlow.	Fully trained. It is not possible to tell whether the model is finished training unless you look at the tr_loss, tr_loss/loss and tr_loss_no_batch. In my case, after the training for first few epochs it stop but I was able to train for more than 100 epochs. I am not sure why is this happening?
Question about the Keras programming language: [cloze test inserted here] what?	Thank you very much for the clarification! A: I think there is a typo: model is the instance of Keras’s model class, therefore summary should be: Import tensorflow as tf Model = tf.kears.models.load_model(‘models/model.h5’) tf.summary.plot_variables(tf.summary.plot_variables(model.summary)) Unfortunately I do not have access to the tf-library at

Fig. 9. GPT-Neo generations guided by prompting.

tokenization is the importance of casing. The mesh between programming and natural languages differs in case sensitivity. Whereas “Model” refers to a specific object with methods and attributes, lowercase “model” generally describes parametric modeling or neural networks. In our cloze tests described in Figures 3, 4, and 5, each ground truth token has an interesting characteristic with regard to casing or overall frequency in the corpus. For example, in our All Data corpora “Model” appears 80 times, whereas “model” appears 1,257 times. Similarly, “Callback” appears 10 times and “callback” appears 36 times. The other tested completion, “tasks”, only appears 16 times. We leave it to future work to explore the long-tail and class imbalance of language modeling and downstream knowledge retention evaluation [38]–[40].

B. Meaning versus Form

Our KerasBERT models have demonstrated some ability to retain knowledge about the Keras programming language. Our qualitative inspections in Figures 3, 4, and 5 show that often the models have a reasonable distribution over words, even if does not get the top-1 answer exactly correct. However, once we graduate to manually designed fact prompts, deviating from the exact style of the training data, performance degrades heavily. Also interestingly, is the performance difference between salient masking and rightmost position masking. These results resonate with the debate between meaning and form in natural language understanding [1]. Although KerasBERT can fit the training data, it cannot generalize to knowledge tests in our writing style.

The debate between meaning and form describes the tendency to model the style of text rather than the semantic meaning it describes, such as entity and relation descriptions. As we have demonstrated with our KerasBERT model, overfitting to form will be particularly challenging to avoid when language modeling small datasets. This is a key problem setup for applications such as schema-free database querying over text corpora. We believe Data Augmentation [36], [37] may be a promising solution going forward. We have found it is non-trivial to design paraphrases of the contexts used in masked language modeling training and leave this design to future work.

C. BERT versus GPT

Another solution to language modeling small datasets that is gaining popularity, is to turn to the Zero-Shot learning ability of large pre-trained models, such as GPT-Neo. Although we were impressed with the generation from GPT-Neo, the hallucination problem is very prominent. We are suspicious of the efficacy of these models for knowledge base or database-style functionality. Very importantly for the proposed application, GPT-Neo contains 6 billion parameters. These models cannot be fine-tuned with economic computational resources and Zero-Shot learning is not well suited to the task of retaining knowledge the models have not been trained on.

KerasBERT is a BERT-style architecture, meaning that it is pretrained with masked tokens in the middle of the context. GPT architectures are pretrained with the masked token at the rightmost part of the context, or the end of the sequence. GPT-style models may be better aligned for the question answering tests explored in this study. However, we think that code support models may benefit from bi-directional context. We suspect Keras users are likely to have errors in the middle of a block of code, thus BERT-style models that can integrate left and right context are useful. We leave it to future work to extend KerasBERT to debugging tasks.

V. Conclusion

In conclusion, we present the KerasBERT models and the tasks of language modeling Keras Documentation. We have web scraped three portions of the Keras documentation for training data. These include the API documentation, developer guides, and code examples. We have also manually extracted a set of fact tests about the Model object in Keras to evaluate robustness to the question form. We compared our 80 million parameter RoBERTa models with the 6 billion parameter GPT-Neo and found GPT-Neo to be superior in a fair comparison

of rightmost token completion. However, we found many hallucinations in GPT-Neo's output, and due to the challenge of fine-tuning or crafting Zero-Shot learning prompts, we argue for training from scratch for future experiments. We are currently looking into distributed training techniques to increase the size of our KerasBERT models [41]. Training language models from scratch on small corpora for the task of knowledge retention comes with many challenges that we suspect can be remedied with novel self-supervised task designs and data augmentation.

REFERENCES

[1] E. Bender, and A. Koller, "Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data," ACL, 2020, pp. 5185-5198.

[2] Y. Liu, M. Ott, N. Goyal, and J. Du, "RoBERTa: A Robustly Optimized BERT Pretraining Approach". CoRR abs/1907.11692, 2019. URL https://arxiv.org/1907.11692.

[3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". CoRR abs/1810.04805, 2018. URL https://arxiv.org/1810.04805.

[4] R. Nadkarni, D. Wadden, I. Beltagy, N. A. Smith, H. Hajishirzi, and T. Hope, "Scientific Language Models for Biomedical Knowledge Base Completion: An Empirical Study". CoRR abs/2106.09700, 2021. URL https://arxiv.org/2106.09700.

[5] Y. Shibata, T. Kida, S. Fukamachi, and M. Takeda, "Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching".

[6] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The Pile: An 800GB Dataset of Diverse Text for Language Modeling". CoRR abs/2101.00027, 2020. URL https://arxiv.org/2101.00027.

[7] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning". In Journal of Big Data 2016.

[8] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "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, 2015.

[9] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt, "Measuring Coding Challenge Competence with APPS". CoRR abs/2105.09938, 2021. URL https://arxiv.org/2105.09938.

[10] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". CoRR abs/1804.07461, 2018. URL https://arxiv.org/1804.07461.

[11] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation". CoRR abs/2102.04664, 2018. URL https://arxiv.org/2102.04664.

[12] A. Roberts, C. Raffel, and N. Shazeer, "How Much Knowledge Can You Pack Into the Parameters of a Language Model?". CoRR abs/2002.08910, 2020. URL https://arxiv.org/2002.08910.

[13] J. Thorne, M. Yazdani, M. Saeidi, F. Silvestri, S. Riedel, and A. Halevy, "Database Reasoning Over Text". CoRR abs/2106.01074, 2021. URL https://arxiv.org/2106.01074.

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners". CoRR abs/2005.14165, 2020. URL https://arxiv.org/2005.14165.

[15] D. Khashabi, S. Min, T. Khot, A. Sabharwal, O. Tafjord, P. Clark, H. Hajishirzi, "UnifiedQA: Crossing Format Boundaries With a Single QA System". CoRR abs/2005.00700, 2020. URL https://arxiv.org/2005.00700.

[16] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In Journal of Machine Learning Research 2020.

[17] J. Thorne, M. Yazdani, M. Saeidi, F. Silvestri, S. Riedel, and A. Halevy, "Neural Databases." CoRR abs/2010.06973, 2020. URL https://arxiv.org/2010.06973.

[18] L. L. Wang and K. Lo, "Text mining approaches for dealing with the rapidly expanding literature on COVID-19." In Briefings in Bioinformatics 2020.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need". In Neural Information Processing Systems 2017.

[20] https://6b.eleuther.ai/, Accessed July 2021.

[21] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, "Large Scale Autoregressive Language Modeling with Mesh-Tensorflow".

[22] https://keras.io/, Accessed June 2021.

[23] A. Nichol, J. Achiam, and J. Schulman, "On First-Order Meta-Learning Algorithms". CoRR abs/1803.02999, 2018. URL https://arxiv.org/1803.02999.

[24] en.wikipedia.org, Accessed June 2021.

[25] S. Gururangan, A. Marasovic, S. Swayamidpta, K. Lo I. Beltagy, D. Downey, N. A. Smith, "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks". In ACL 2020.

[26] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "BioBERT: a pre-trained biomedical language representation model for biomedical text mining". In Briefings in Bioinformatics 2020.

[27] I. Beltagy, K. Lo, and A. Cohan, "SciBERT: A Pretrained Language Model for Scientific Text". In ACL 2019.

[28] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval Augmentation Reduces Hallucination in Conversation,"

[29] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, M. Kelcey, J. Devlin, K. Lee, K. N. Toutanova, L. Jones, M. Chang, A. Dai, J. Uszkoreit, Q. Le, and S. Petrov, "Natural Questions: a Benchmark for Question Answering Research". In ACL 2019.

[30] https://www.ncbi.nlm.nih.gov/pmc/. Accessed July 2021.

[31] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages". In ACL 2020.

[32] P. Lewis, L. Denoyer, and S. Riedel, "Unsupervised Question Answering by Cloze Translation". In ACL 2019.

[33] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. v. Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-Art Natural Language Processing". In EMNLP 2020.

[34] A. Mastropaolo, S. Scalabrino, N. Cooper, D. N. Palacio, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Studying the Usage of Text-To-Text Transfer Transformer to Support Code-Related Tasks". CoRR abs/2102.02017, 2021. URL https://arxiv.org/2102.02017.

[35] I. Kanitscheider, J. Huizinga, D. Farhi, W. H. Guss, B. Houghton, R. Sampedro, P. Zhokhov, B. Baker, A. Ecoffet, J. Tang, O. Klimov, J. Clune, "Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft". CoRR abs/2106.14876, 2021. URL https://arxiv.org/2106.14876.

[36] C. Shorten, T. M. Khoshgoftaar, B. Furht, "Text Data Augmentation for Deep Learning". In Journal of Big Data 2021.

[37] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning". In Journal of Big Data 2019.

[38] J. D. Prusa and T. M. Khoshgoftaar, "Designing a better data representation for deep neural networks and text classification". In 2016 IEEE 17th International Conference on Information Reuse and Integration.

[39] J. D. Prusa and T. M. Khoshgoftaar, "Improving deep neural network design with new text data representations." In Journal of Big Data 2017.

[40] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance." In Journal of Big Data 2019.

[41] R. K. L. Kennedy and T. M. Khoshgoftaar, "Accelerated Deep Learning on HPCC Systems". In 2020 19th IEEE International Conference on Machine Learning and Applications.

[42] J. L. Leevy and T. M. Khoshgoftaar. "A Short Survey of LSTM Models for De-identification of Medical Free Text." In 2020 IEEE 6th International Conference on Collaboration and Internet Computing.

[43] C. Shorten, T. M. Khoshgoftaar, B. Furht. "Deep Learning applications for COVID-19." In Journal of Big Data 2021.