UNIVERSAL PRE-TRAINING

Peter Bloem
Learning & Reasoning Group
Vrije Universiteit
Amsterdam, Netherlands
up@peterbloem.nl

ABSTRACT

We investigate the use of randomly generated data for the sake of pre-training a model, before seeing any domain-specific data. We justify this approach theoretically from the perspective of Kolmogorov complexity. We show empirically, in the domain of token sequences, that randomly generated data can be used to pre-train a model a-priori—before the data is seen—and that this approach has several benefits. First, it leads to better-than-chance zero-shot performance. That is, the model shows some in-context test-set performance before seeing *any* training data. Second, when the model is fine-tuned on training data after pretraining, it converges faster and to better performance, even when the pretraining is accounted for in the computational budget. And third, The resulting model shows better generalization both in-domain and out-of-domain. While these experiments are only small-scale, we perform a scaling experiment that shows a promising trend towards larger models.

1 INTRODUCTION

Even in the domain of natural language, machine learning is fast approaching the point where the amount of data on which we would like to train a model is larger than the amount of data available . In many other domains, this point has long been passed. At the same time, as artificial intelligence becomes more and more integrated in production systems, the authors of, for instance, literature and art, are less eager for their work to end up in training datasets. In short the question of whether we can do more with less data is increasingly urgent.

One approach is to generate *synthetic data*. This may seem at first to be a false economy: how can we learn something from a data generator that we built ourselves? In information theoretic terms, the data processing inequality (DPI) tells us that we cannot increase the information content of a signal by applying computation to it. This seems to preclude the possibility of enriching our models with data from any other source than the real world.

However, the key to valuable data is not information content, it's *structure*. The most information-rich signal, a fully random one, is among the least valuable data to learn from. An interesting source of data provides a mix of randomness and structure.

Consider the following analogy (from). A monkey behind a typewriter, bashing keys at random, will famously produce the complete works of Shakespeare in a large, but finite amount of time. A monkey behind a *computer* will also produce the collected works of Shakespeare, and o so more quickly. This is because there is structure in natural language that may be exploited by a computer program. The second monkey only needs to be lucky enough to type out the computer program, whereas the first monkey needs to be lucky enough to get every single character correct.

The key takeaway is that by taking a source of randomness, a monkey or a random number generator, and passing its output through a computer, we can *enrich* data (from the perspective of machine learning). This may well reduce the amount if information in the data

(in the Shannon/Kolmogorov sense of the word), but it adds structure. Structure that a machine learning model can learn, and that may transfer to real-world tasks.

With this principle in mind, we ask the following question. Can we pre-train a model before seeing the data? Can we generate random data, before knowing what our task is, such that pre-training on this random data benefits learning? We show that this is indeed possible, and we call the resulting approach *universal pre-training*. We provide some theoretical foundation for this principle, from the field of Kolmogorov complexity.

Figure **??** shows a simple example of our approach: a string of ASCII characters chosen uniformly at random, and the result of passing that string through a randomly initialized transformer model. The first string is fully random, and nothing useful may be learned from it. The second string contains structure: certain characters and structures re-occur in a predictable way. We can make sensible predictions about one part of the string by studying another.

Some of these basic patterns may transfer to other domains. For instance, the relatiuve frequency of a character at the start of the string, is more likely than not be a reasonable prediction of its probability at the end of the string. Other patterns, such as the specific characters that are frequent, are entirely specific to this sample, and will disappear if we re-initialize the network, and pass another random string through.

Our results show that we can trade off compute for data. By pre-training on entirely synthetic data, sampled in this manner—by passing fully random data through a computational process—we end up with a model that is better initialized once it starts seeing the real-world data. We show that a model prepared with universal pretraining reaches better performance than a model that has been initialized traditionally. We also show better out-of distribution performance, and better generalization.

2 PRELIMINARIES

We will briefly review the required preliminaries. See for a more detailed treatment. These are mostly required to understand the theoretical framework we use to show that universal pre-training is feasible. To understand the experimental part of the paper, the intuitive explanation in the introduction should suffice.

Let $\mathbb{B}$ be the set of finite binary strings. We will use Turing machines to model computable probability distributions on $\mathbb{B}$. To this end we, fix Turing machines with a single input tape and a separate output tape. The read-head on the input tape is only allowed to move one way: the effect is that after reading a bit from the input tape, the machine either halts, reads the next bit, or computes infitenly without doing either. This means that the set of inputs on which a machine halts forms a prefix-free set: no halting input is the prefix of another halting input.

If a machine T halts, we take the input bits it has read as its input x, and the value on its output tape when it halts as its output y, writing $T(x) = y$. We write $T(x) = \infty$ for an input on which the machine does not halt.

We can now use such machines to simulate probability distributions, but feeding it random bits. We start the machine, and whenever it reads from the input tape, we sample a random bit and provide it. If the machine halts, we take the output as our sample.[1]

We assume that we are given some enumeration T_i of all Turing machines. We refer to the probability of sampling y from T_i as $p_i(y)$. Since each input x is samples with exactly probability $2^{|x|}$, where $|\cdot|$ denotes string lengths, we can write p_i as

[1]Since the machine may not halt, this is a deficient distribution on $\mathbb{B}$. However, we will mostly deal with subsets of the Turing machines that always halt. If we place no restrictions on the Turing machines used, we have defined the class of *lower semicomputable semimeasures*. We will refer to these as the computable distributions for the sake of simplicity.

$$p_i(y) = \sum_{x:T_i(x)=y} 2^{|x|}.$$

Next, assume a pairing function (i, x) that encodes i and x into a single string in $\mathbb{B}$, then we can define a universal Turing machine $U((i, x)) = T_i(x)$, which unpacks the pairing function and then simulates the running of the i-th Turing machine on input x. If the pairing function uses prefix-free codewords, which is easy to achieve, then U itself occurs somewhere in the enumeration T_i. [2]

With this, we can sample from U. This effectively samples a Turing machine i according to some prior probability $p(i)$, and then samples an output y according to $p_i(y)$. We call this the *universal distribution* $m(y)$. The universal distribution is a Bayesian mixture over the class of computable distributions.

3 RELATED WORK

3.1 SYNTHETIC DATA

Training on synthetic data goes back to the simple principles of data augmentation. In recent years, much research has been devoted to the use of simulators generating synthetic data to encode very specific domain assumptions. For instance, in training self-driving cars, robotics control, tabular data, ...

Our work with this trend, but with the change that we aim to make the assumptions on the domain *minimal*. In its most general form, universal pre-training only assumes that the source of the downstream dataset is *computational* in nature. This allows one form of pre-training to be used for a wide variety of downstream tasks.

3.2 THE NO-FREE LUNCH THEOREM

The idea of pre-training a model irrespective of its task seems to fly in the face of the no-free-lunch theorem , which states that, averaged over all tasks, all models perform the same. This suggests that if our pre-training makes any change at all to the model's performance on a subset of tasks, it must reduce its performance on other tasks.

This is the case, but it's possible to get out of the NFL theorem with a very broad, almost universal assumption about our data: that the process —p that generated it is computational. That is, it can be simulated by a probabilistic Turing machine. Under this assumption, the source of our data is *dominated* by the universal distribution m, and by any class-universal distribution m_C for which $p \in C$.

This means that there are computable probability distributions $m_C C$ that fit our data as well as its source p does (up to a multiplicative constant). Moreover, this holds for all $p \in C$. Therefore, m_C provides a universal predictor for C. This does not violate the NFL theorem, because we do make an assumption about the data, namely that it comes from a source in C. However, C can be made *very* broad, for instance the class of all polynomial-time Turing machines, while still allowing practical sampling.

By training on samples from m_C, we are essentially approximating it in our pre-trained model.[3]

–Yuki Asano's work.

[2] A simple way to achieve this is to define a prefix-free encoding of $\mathbb{B}$, written as $\bar{x}$. We can then use the pairing function $(i, x) = \bar{i}x$, since concatenating two prefix-free codewords results in another prefix-free code.

4 METHOD

4.1 THEORETICAL FRAMEWORK

Let C be a set of computable functions from $\mathbb{B}$ to $\mathbb{B}$. If de define some prior on C, we can easily sample from m_C. We can now sample a random model from C, feed it random bits, and observe the output.

By the logic spelled out in the introduction, this output is more likely to contain interesting structure than the original random bits. We could proceed by pre-training a model on samples from m_C directly, but we can enhance the depth of our data still more.

The idea is that if taking a uniform random string and passing it through a computable function produce 'interesting' data, the it's possible that taking interesting data and passing it through a computable function, creates even more interesting data.

That is, if we take the output of the monkey behind the computer, and feed it into another computer, are we even more likely to produce Shakespeare? If the computers are universal, we cannot boost its power this way, but if their resources are limited, we can. For instance, if the computer is only allowed to run for 1 minute, this provides a computer that is allowed to run for 2 minutes.

We can formalize and prove this intuition as follows. Let p_u be a base distribution on $\mathbb{B}$ which is uniform in the sense that all string of the same length have equal probability.

Let $m_C^1 = m_C$ be the distribution defined by the process of sampling u from p_u and f from $p(C)$ and computing $f(u)$. Then, let m_C^{n+1} be the distribution defined by the process of sampling u from m_C^n and f from $p(C)$ and computing $f(u)$

4.2 PRACTICAL APPROACH

To test these ideas in practice, we make the following changes and implementation choices.

First, instead of training on sequences of bits, we take our token alphabet from the expected downstream task. These are ASCII characters in the first experiment and WordPiece tokens in the second. While this does reduce the universality of our approach somewhat, it suffices for a proof-of-concept (see also the discussion).

For ease of implementation we limit ourselves to sequences of a fixed length. This makes memory use predictable. Our class of source models C is a model class of single-stack transformers with causal self-attention. The architecture is detailed in the appendix. We sample from C simply by initializing its parts using the standard algorithm for the part (also detailed in the appendix), and then scaling all parameters by a fixed constant afterwards.

For efficiency and ease of implementation, we initialize a single model f_i, and sample a batch of n instances from it. This means that the batch is *not* independently samples from m_C. Training directly on such batches would create a bias at each gradient update for the patterns that are specific to i.

To approximate independent sampling, we instead keep a buffer of k samples. Each iteration, we take n instances x from the buffer and replace them by $f_i(x)$. We then sample n random instances again for our target model to train on. We also replace a small proportion α of instances in the buffer with (pseudo-)random noise, to stop the amount of entropy becoming to low.

While this does not fully ensure that the samples are independent, it ensures that that patterns from different models i appear in the data.

Moreover, because we are sampling structured noise, and feeding it back into the data, Theorem **??** tells us that we are approximating a sample from a mixture over the infinite

sequence of models $m_C^1, m_C^2 \ldots$, while only passing one batch of instances through one model from C (the cost of a direct sample from m_C^1).[4]

The complete algorithm is detailed in Algorithm **??**.

On these samples, we train a standard autoregressive transformer (architecture and training details in the appendix) by shifting the sequences one one character to the left to form the target value.

5 EXPERIMENTS

First, we sample sequences not consisting of bits but of a character set or 128 or 256 characters.[5]

6 DISCUSSION

There are three main ingredients in the modern machine learning process that limit how well we do: the fundamental abilities of the algorithm, the available amount of data and the available amount of computational resources. This paper shows that, at least to some extent, it is possible to trade off computational resources for data. We can generate structured random data to pre-train our models, making them require less real-world training data by comparison.

Moreover, the pre-training is in some sense universal. That is, the same pre-training can benefit a large amount of downstream tasks. This allows us to amortize the pre-training over many different tasks, reducing the cost of the tradeoff.

6.1 LIMITATIONS AND FUTURE WORK

approximation of m_C We can show that m_C is a universal model for any $p \in C$. We then proceed by approximating m_C by training a model on samples from it. However, the model we use is not much more complex than p.

There is no guarantee that a simple model that is not much more complex than p can simulate m_C. Generally, it's likely the case that m_C needs to be of higher complexity than any element in C.[6]

In our work we simply give our model which approximates m_C the same complexity as the models in C. It's unlikely that this leads to a convergent approximation, even in optimal conditions. Most likely the approximation simply lacks the required computational complexity.

For the time being, without any theoretical guarantees we simple assume that an imperfect approximation of m_C is still likely to produce effective pretraining for any task in C, and proceed empirically. This leaves several theoretical questions unanswered: (a) what is the computational complexity of m_C? (b) Under what conditions is m_C effectively learnable? (c) If there are fundamental limitations in our learned approximation of m_C, what are the implications for the effectiveness pretraining?

In short, we use our theoretical framework to show what is possible *in principle*, despite seeming objections from results like the NFL theorem. We then show experimentally, that the expected results can, in at least one domain, be achieved. However, the theoretical framework is insufficient, as yet to say anything about when universal pre-training is guaranteed to work. We leave this to future work.

[4]The probability of a sample coming from m_C^n decays exponentially with n, so this claim should be taken with a pinch of salt.

[5]We use 7 bits ASCII characters for experiments where we print the sampled strings, and 8 bits extended ASCII character

[6]There aren't (to the best of our knowledge) any non-trivial separation proofs.

Universality of assumptions Calling the specific approach used in our experiments *universal* is perhaps a slight overstatement. We do not sample from a universal distribution (which is not tractable). A class-universal distribution with a sufficiently broad class (like P) would still justify the name universal pre-training (since it's very likely that all data sources are in that class), but we fall short of that as well.

First, in the interest of efficient generation, our class C is limited to finite strings, and to the transformer architecture. While the stacking trick allows, arbitrary sampling depths (in theory), This does not necessarily mean that we are sampling from the distribution on arbitrarily deep transformers.

Moreover, we have manipulated the specifics of the source model somewhat to generate data that looks, on inspection like it contains reasonable structure. This may provide some bias towards the kind of patterns that are likely to appear in our downstream tasks.

In short, while the framework as we have described it, in its most general terms deserves the moniker *universal*, our practical implementation of it is more of an approximation to this ideal.

Finally, in our experiments, we pre-train our model using the token alphabet of the downstream task. This approach limits the universal applicability to any model with that token set. Note that the content of the token set doesn't matter, so we only need to know the number of tokens that will be expected in the downstream task. We accept this limitation for the sake of simple proof-of-concept. In more serious applications, one solution would be to agree on a universal number of tokens (or at least an upper bound).

Another solution might be to dynamically set the number of tokens and use a method analogous to position encodings to create non-learnable vector representations for the tokens.

Of course, token-based transformers are in some sense an artifact of natural language modeling. As models like the IO performer show, we can move to a more universal data representation (such as bits strings) to increase the universality of our per-trained model.

Other domains Our experiments are limited to token sequences. That is, string of tokens from a finite vocabulary. In principle, the same idea could be applied to computer vision, continuous time series and any other domains. We leave this to future work.

Other computational sources For ease of implementation, we use the same type of model for our source of randomness as for our pre-trained model. this is not strictly necessary. For instance, we could easy sample probabilistic string models from different levels of the Chomsky hierarchy. These models can be sampled very efficiently, and may allow more interesting structure to emerge more quickly than it would from our source model.

ACKNOWLEDGMENTS

A APPENDIX

You may include other additional sections here.