FOUNDERS AND CODERS

Week 6

Machine Learning Institute

MoE & recap

Fine-tuning, RL, LoRA, and all that jazz

FOUNDERS AND CODERS

Machine Learning Institute

Overview

Train a base model

Instructions fine-tuning

Reinforcement Learning

arXiv:2203.15556v1 [cs.CL] 29 Mar 2022

DeepMind

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly undertrained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

1. Introduction

Recently a series of *Large Language Models* (LLMs) have been introduced (Brown et al., 2020; Lieber et al., 2021; Rae et al., 2021; Smith et al., 2022; Thoppilan et al., 2022), with the largest dense language models now having over 500 billion parameters. These large autoregressive transformers (Vaswani et al., 2017) have demonstrated impressive performance on many tasks using a variety of evaluation protocols such as zero-shot, few-shot, and fine-tuning.

The compute and energy cost for training large language models is substantial (Rae et al., 2021; Thoppilan et al., 2022) and rises with increasing model size. In practice, the allocated training compute budget is often known in advance: how many accelerators are available and for how long we want to use them. Since it is typically only feasible to train these large models once, accurately estimating the best model hyperparameters for a given compute budget is critical (Tay et al., 2021).

Kaplan et al. (2020) showed that there is a power law relationship between the number of parameters in an autoregressive language model (LM) and its performance. As a result, the field has been training larger and larger models, expecting performance improvements. One notable conclusion in Kaplan et al. (2020) is that large models should not be trained to their lowest possible loss to be compute optimal. Whilst we reach the same conclusion, we estimate that large models should be trained for many more training tokens than recommended by the authors. Specifically, given a 10× increase computational budget, they suggests that the size of the model should increase 5.5× while the number of training tokens should only increase 1.8×. Instead, we find that model size and the number of training tokens should be scaled in equal proportions.

Following Kaplan et al. (2020) and the training setup of GPT-3 (Brown et al., 2020), many of the recently trained large models have been trained for approximately 300 billion tokens (Table 1), in line with the approach of predominantly increasing model size when increasing compute.

Corresponding authors: {jordanhoffmann|sborgeaud|amensch|sifre}@deepmind.com

© 2022 DeepMind. All rights reserved

Hoffmann et al. (NeurIPS 2022)

Communicated by Jeffrey Elman

Adaptive Mixtures of Local Experts

Robert A. Jacobs
Michael I. Jordan
Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139 USA

Steven J. Nowlan
Geoffrey E. Hinton
Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4

We present a new supervised learning procedure for systems composed of many separate networks, each of which learns to handle a subset of the complete set of training cases. The new procedure can be viewed either as a modular version of a multilayer supervised network, or as an associative version of competitive learning. It therefore provides a new link between these two apparently different approaches. We demonstrate that the learning procedure divides up a vowel discrimination task into appropriate subtasks, each of which can be solved by a very simple expert network.

1 Making Associative Learning Competitive

If backpropagation is used to train a single, multilayer network to perform different subtasks on different occasions, there will generally be strong interference effects that lead to slow learning and poor generalization. If we know in advance that a set of training cases may be naturally divided into subsets that correspond to distinct subtasks, interference can be reduced by using a system composed of several different "expert" networks plus a gating network that decides which of the experts should be used for each training case.[1] Hampshire and Waibel (1989) have described a system of this kind that can be used when the division into subtasks is known prior to training, and Jacobs *et al.* (1990) have described a related system that *learns* how to allocate cases to experts. The idea behind such a system is that the gating network allocates a new case to one or a few experts, and, if the output is incorrect, the weight changes are localized to these experts (and the gating network).

[1] This idea was first presented by Jacobs and Hinton at the Connectionist Summer School in Pittsburgh in 1988.

Neural Computation 3, 79–87 (1991) © 1991 Massachusetts Institute of Technology

Jacobs et al. (Neural Computation 1991)

Under review as a conference paper at ICLR 2017

arXiv:1701.06538v1 [cs.LG] 23 Jan 2017

OUTRAGEOUSLY LARGE NEURAL NETWORKS: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer[1], Azalia Mirhoseini*†[1], Krzysztof Maziarz*[2], Andy Davis[1], Quoc Le[1], Geoffrey Hinton[1] and Jeff Dean[1]

[1]Google Brain, {noam,azalia,andydavis,qvl,geoffhinton,jeff}@google.com
[2]Jagiellonian University, Cracow, krzysztof.maziarz@student.uj.edu.pl

ABSTRACT

The capacity of a neural network to absorb information is limited by its number of parameters. Conditional computation, where parts of the network are active on a per-example basis, has been proposed in theory as a way of dramatically increasing model capacity without a proportional increase in computation. In practice, however, there are significant algorithmic and performance challenges. In this work, we address these challenges and finally realize the promise of conditional computation, achieving greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters. We introduce a Sparsely-Gated Mixture-of-Experts layer (MoE), consisting of up to thousands of feed-forward sub-networks. A trainable gating network determines a sparse combination of these experts to use for each example. We apply the MoE to the tasks of language modeling and machine translation, where model capacity is critical for absorbing the vast quantities of knowledge available in the training corpora. We present model architectures in which a MoE with up to 137 billion parameters is applied convolutionally between stacked LSTM layers. On large language modeling and machine translation benchmarks, these models achieve significantly better results than state-of-the-art at lower computational cost.

1 INTRODUCTION AND RELATED WORK

1.1 CONDITIONAL COMPUTATION

Exploiting scale in both training data and model size has been central to the success of deep learning. When datasets are sufficiently large, increasing the capacity (number of parameters) of neural networks can give much better prediction accuracy. This has been shown in domains such as text (Sutskever et al., 2014; Bahdanau et al., 2014; Jozefowicz et al., 2016; Wu et al., 2016), images (Krizhevsky et al., 2012; Le et al., 2012), and audio (Hinton et al., 2012; Amodei et al., 2015). For typical deep learning models, where the entire model is activated for every example, this leads to a roughly quadratic blow-up in training costs, as both the model size and the number of training examples increase. Unfortunately, the advances in computing power and distributed computation fall short of meeting such demand.

Various forms of conditional computation have been proposed as a way to increase model capacity without a proportional increase in computational costs (Davis & Arel, 2013; Bengio et al., 2013; Eigen et al., 2013; Ludovic Denoyer, 2014; Cho & Bengio, 2014; Bengio et al., 2015; Almahairi et al., 2015). In these schemes, large parts of a network are active or inactive on a per-example basis. The gating decisions may be binary or sparse and continuous, stochastic or deterministic. Various forms of reinforcement learning and back-propagation are proposed for trarining the gating decisions.

*Equally major contributors
†Work done as a member of the Google Brain Residency program (g.co/brainresidency)

1

Shazeer et al. (ICLR 2017)

Journal of Machine Learning Research 23 (2022) 1-40 Submitted 8/21; Revised 3/22; Published 4/22

arXiv:2101.03961v3 [cs.LG] 16 Jun 2022

Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

William Fedus*
LIAMFEDUS@GOOGLE.COM

Barret Zoph*
BARRETZOPH@GOOGLE.COM

Noam Shazeer
NOAM@GOOGLE.COM
Google, Mountain View, CA 94043, USA

Editor: Alexander Clark

Abstract

In deep learning, models typically reuse the same parameters for all inputs. Mixture of Experts (MoE) models defy this and instead select *different* parameters for each incoming example. The result is a sparsely-activated model—with an outrageous number of parameters—but a constant computational cost. However, despite several notable successes of MoE, widespread adoption has been hindered by complexity, communication costs, and training instability. We address these with the introduction of the Switch Transformer. We simplify the MoE routing algorithm and design intuitive improved models with reduced communication and computational costs. Our proposed training techniques mitigate the instabilities, and we show large sparse models may be trained, for the first time, with lower precision (bfloat16) formats. We design models based off T5-Base and T5-Large (Raffel et al., 2019) to obtain up to 7x increases in pre-training speed with the same computational resources. These improvements extend into multilingual settings where we measure gains over the mT5-Base version across all 101 languages. Finally, we advance the current scale of language models by pre-training up to trillion parameter models on the "Colossal Clean Crawled Corpus", and achieve a 4x speedup over the T5-XXL model.[1,2]

Keywords: mixture-of-experts, natural language processing, sparsity, large-scale machine learning, distributed computing

*. Equal contribution.
1. JAX code for Switch Transformer and all model checkpoints are available at https://github.com/google-research/t5x
2. Tensorflow code for Switch Transformer is available at https://github.com/tensorflow/mesh/blob/master/mesh_tensorflow/transformer/moe.py

©2022 William Fedus, Barret Zoph and Noam Shazeer.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v23/21-0998.html.

Fedus et al. (JMLR 2022)

MoE

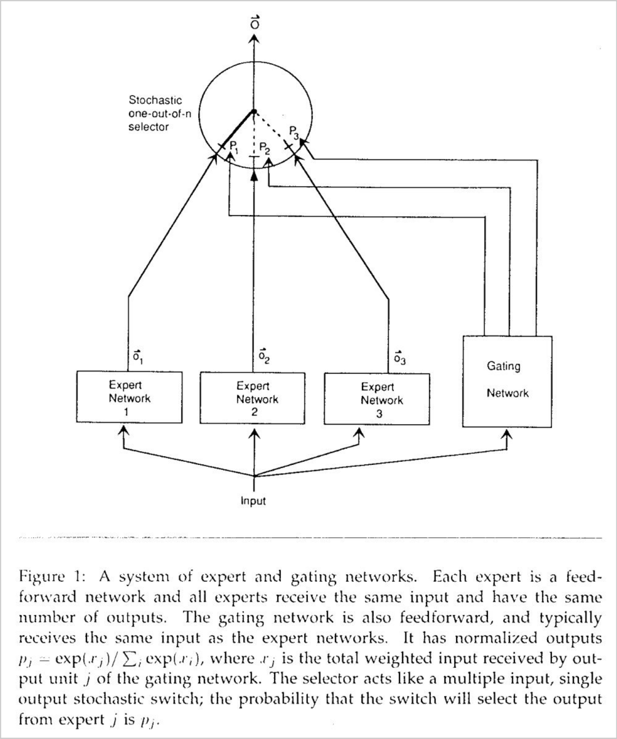


Figure 1: A system of expert and gating networks. Each expert is a feedforward network and all experts receive the same input and have the same number of outputs. The gating network is also feedforward, and typically receives the same input as the expert networks. It has normalized outputs $p_j = \exp(x_j)/\sum_i \exp(x_i)$, where x_j is the total weighted input received by output unit j of the gating network. The selector acts like a multiple input, single output stochastic switch; the probability that the switch will select the output from expert j is p_j.

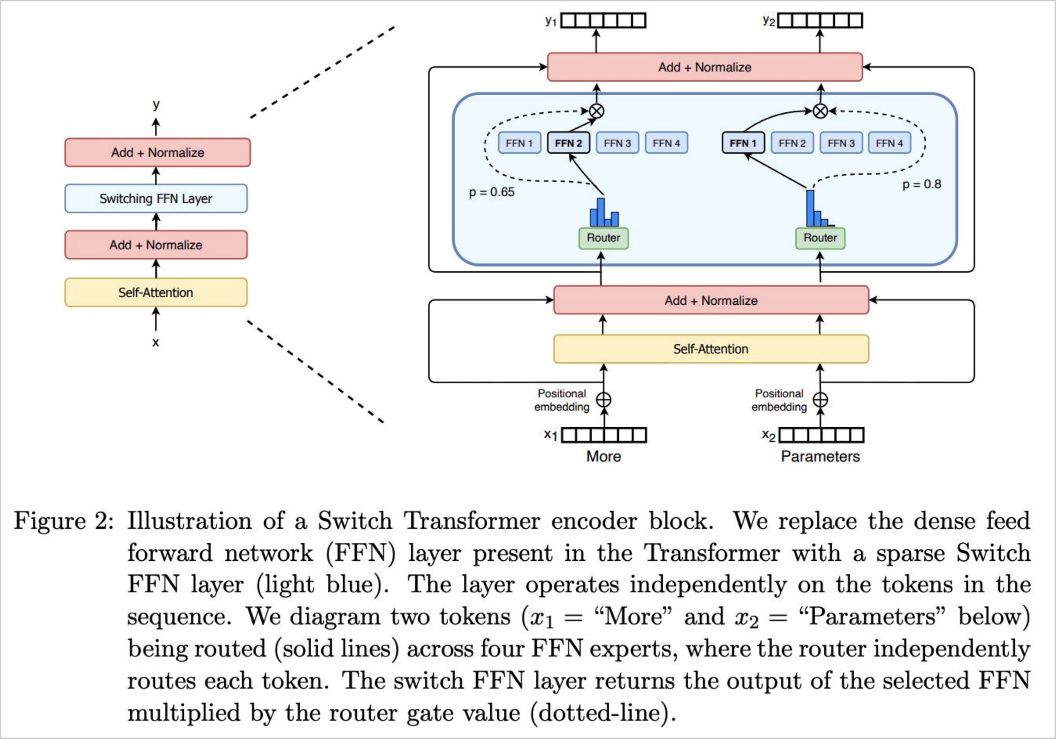


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

FOUNDERS AND CODERS

RLHF

Machine Learning Institute

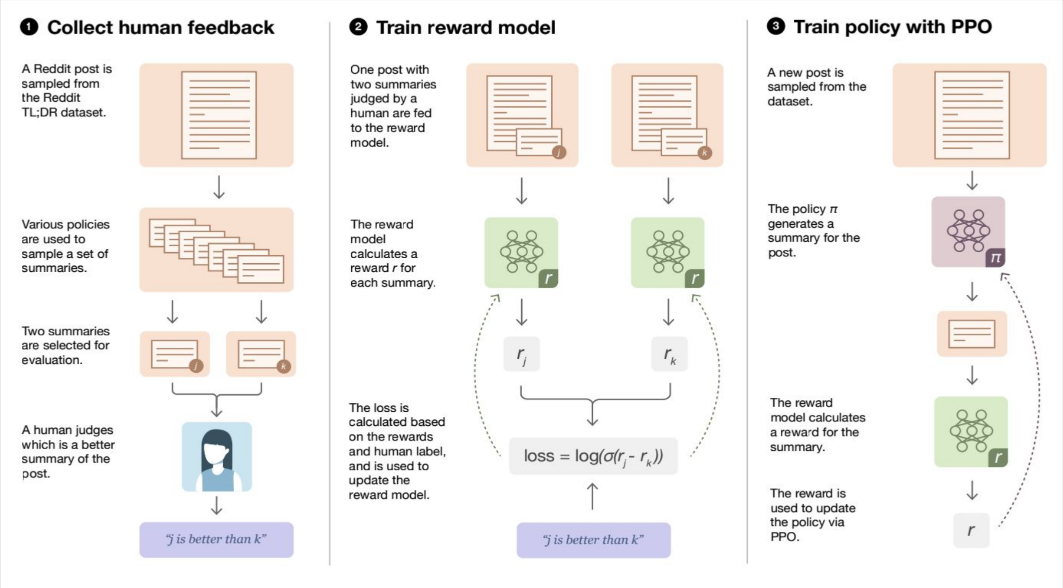
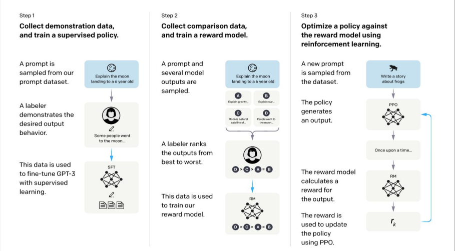

1 Collect human feedback
A Reddit post is sampled from the Reddit TL;DR dataset.
Various policies are used to sample a set of summaries.
Two summaries are selected for evaluation.
j
k
A human judges which is a better summary of the post.
"j is better than k"
2 Train reward model
One post with two summaries judged by a human are fed to the reward model.
j
k
The reward model calculates a reward r for each summary.
r
r
r_j
r_k
The loss is calculated based on the rewards and human label, and is used to update the reward model.
loss = $\log(\sigma(r_j - r_k))$
"j is better than k"
3 Train policy with PPO
A new post is sampled from the dataset.
The policy π generates a summary for the post.
π
The reward model calculates a reward for the summary.
r
The reward is used to update the policy via PPO.
r

Figure 2: Diagram of our human feedback, reward model training, and policy training procedure.

2020


Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

2022

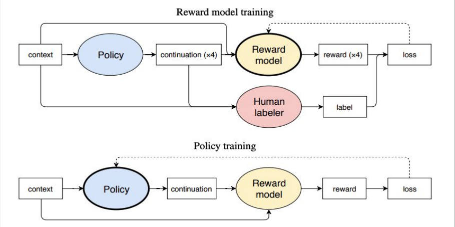


Figure 1: Our training processes for reward model and policy. In the online case, the processes are interleaved.

2019