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

Main Idea: the idea for this paper is based on the "Scaling Laws for Neural Language Models", which states that larger models are more sample-efficient, and thus advises that the optimal allocation of a fixed compute budget should prioritize increasing the number of model parameters while decreasing the number of training steps. This created the motivation to scale MoE models, which allow for an increase in parameter count while keeping FLOPs constant. The main issues to be addressed related to scaling MoEs are:

- Complexity
- Communication costs
- Training instability

Top-1 Routing

Sparsely-Gated MoE had claimed that top-k routing had to have $k > 1$ to have non-trivial gradients to the routing function (the routers were thought to not train properly if they didn't have at least two experts to compare results with). Switch challenges this idea and successfully uses top-1 routing. This introduces advantages such as reduced computation, reduced batch size (top-2 routing requires an expert capacity factor of 2, which is not needed in top-1 routing) and reduced communication costs.

Expert Capacity

Each expert has an expert capacity, which is the computation it can perform on each batch. Due to the dynamic nature of routing (load balancing in a batch is not guaranteed), this expert capacity can lead to memory overflow issues (where the overflown tokens in a batch are skipped). This can be managed by setting a capacity factor to the experts (keep some buffer to each expert's machine).

Expert capacity = (tokens per batch/number of experts) * capacity factor

Although this helps with memory overflow and the issue of skipped tokens, it results in increased computation and memory costs.

Load Balancing Loss

For the auxiliary loss, Switch introduces a differentiable load balancing loss that considers both the fraction of tokens assigned to each expert and the probability given to each expert by the router (sum of the probabilities given to each expert when it was selected).

Auxiliary loss = alpha * N * summation over all experts (fi * pi), where N is the number of experts, fi is the fraction of tokens in the batch dispatched to expert I and pi is the fraction of the router probability allocated to expert i.

This loss ensures load balancing by leveraging the fact that the product fi * pi is minimized under a uniform distribution, where both fi and pi are equal or close to 1/N for each expert,

corresponding to a balanced load. The sum of fi and pi is constrained to 1 across all experts, highlighting the zero-sum nature of resource distribution. The non-linear impact of the product fi * pi in the loss function means that the sum of these products across experts is minimized when the load (dispatched tokens) and router probabilities are evenly distributed. This minimization drives the model toward a uniform distribution, promoting load balance by ensuring that no single expert is disproportionally favored in terms of load or router's allocation.

This loss is a complement to the cross-entropy loss -> total loss = cross-entropy + auxiliary loss.

T5 (dense) vs MoE (top-2 routing) vs Switch (top-1 routing)

Models were trained on a masked language modeling objective with 15% token dropout (for MoE and Switch).

The same computation per token is applied (equal FLOPs) for each model. However, MoE has more active parameters since it uses top-2 routing.

- Switch outperforms T5 and MoE in terms of speed-quality (fixed amount of computation and wall-clock time)
- Switch has a lower computational footprint -> increasing its size to match the speed of MoE leads to outperforming MoE and T5 on a per step basis (since MoE is slower than Switch due to higher number of active parameters)
- Switch performs better at lower capacity factors (1, 1.25)

Training and Fine-Tuning Techniques

Instability in MoE comes mainly from the hard-switching routing strategy. This makes it challenging to train in lower precision. To combat this, a few tricks are used:

- Selective precision with large sparse models
    - Selective casting to float32. More specifically, the router input is casted to float32 within the body of the router function (local computations) but back to float16 at the end of the routing function when the results are dispatched for the selection of the router computation (between devices). This optimizes the router stability while keeping the communication costs low.
- Smaller parameter initialization for stability
    - Simple initialization changes (especially reducing the normal initialization scale of a Transformer by 10) drastically helps with stability.
    - A popular initialization strategy is used -> weights randomly initialized from a distribution with mean of 0 and st dev of sq root(s/n), where s is a scale hyper-parameter and n is the number of input units in the weight tensor.
- Regularizing large sparse models
    - Since MoE models have much more parameters than regular dense Transformers, they can be more prone to overfitting when fine-tuned in small downstream tasks.
        - Switch proposes increasing dropout in expert layers while keeping a smaller dropout rate in other layers. This is shown to lead to improvements in fine-tuning.

Scaling properties

- When keeping the FLOPs per token fixed, having more total parameters (increase in number of experts) speeds up training (although at a cost in memory) in a per-step basis (training is more sample-efficient).
- MoE models have higher communication costs than dense models. So even though Switch is more efficient in a per-step basis, this can fail to hold in a time basis.
    - With a fixed training duration and computational budget, Switch achieves a 7x speedup in training compared to T5 (Switch achieves the same loss 7x faster).
    - Switch shows improvements in both per-step and time basis during pre-training over T5 even when compared to T5-Large (3.5x increase in FLOPs).

Fine-tuning

With an increase in dropout rate (0.4 vs 0.1), Switch was shown to have improved fine-tuning results over T5-Base and T5-Large in a FLOP-matched basis in NLP tasks, including reasoning and knowledge-heavy tasks.

Distillation

When distilling a large sparse model into a small dense model, it is found that reducing the model to $1/20^{th}$ of its original parameter count still retains 30% of the Switch gains over T5. This is a sign

that not all gains are due to increased parameter count, indicating that some part of the gains can be due to other reasons related to the MoE capturing parameters more efficiently.

Parallelism (Data, Model, Experts)

Data parallelism – data is shared over all cores available, while keeping a copy of the model in each core (model is replicated over each core). Each core (model) only needs to communicate at the end of each batch to perform an update on the model's parameters.

Model parallelism – model is distributed over all cores, while passing all tokens through each core. This method leads to high communication costs between cores since each token needs to be passed from core to core to produce a label.

Model and data parallelism – model is split through m cores and data is split through n cores (mix of pure model parallelism and pure data parallelism.

Expert and data parallelism – the model is distributed by having one expert in each core while sharding the data over all cores. This sharding is done by the routing function, assuming the auxiliary loss will help with load balancing to prevent the token overflow issue.

Expert, model and data parallelism – more complex method where each expert is distributed through multiple cores (in case a single expert does not fit in a single core, which can happen if we want to increase the number of FLOPs – this leads to a decreased batch size because more memory is needed for the experts and the communication costs between cores, leading to less

memory available for the data). This needs to consider the communication costs between the routing function distributing the data and the model/expert sharding.

Increasing the number of experts does not seem to lead to instability in training (as seen in training the 1.6T model). What caused instability is increasing the number of FLOPs (increasing the size of each expert).

My takeaways:

- The claim made on Sparsely-Gated MoE that $k > 1$ is needed for top-k routing initially seems to make sense. This would help the gradient to differentiate between good and bad experts for that input. For example, with $k = 2$, the router can compare the gradients that come from each expert, and therefore learn which expert was more useful to the final output. With $k = 1$, this property is not present.
    - Top-1 routing, even if it works, would not benefit from overlaps in the clusters that each expert specializes in. Coupled with reduced computation (less parameters used during inference), it seems that this would lead to efficiency gains but with a loss in performance.
- It is true that increasing the model parameters makes the model more prone to overfitting, especially when there is not enough data available (the more data, the less the risk of overfitting), which is more likely during fine-tuning. Increasing regularization

(dropout in this case) is logical when it comes to helping with that. Remember that dropout will randomly drop training samples, allowing the model to go through the data more times.

- Switch is shown to perform significantly better than dense models in pre-training (in both a per-step and per-time basis).
- Increasing regularization during fine-tuning shows promise for MoE models. However, MoE architectures are not as well suited as dense models for fine-tuning due a higher amount of data being needed to prevent overfitting.
    - Results from Switch show that it outperformed T5 in fine-tuning tasks such as GLUE and SQuAD. However, these seem like tasks that have enough data to prevent the issue of overfitting in Switch. It would be interesting to see how this holds when fine-tuning on tasks with less data available.
- Distillation results from Switch show great promise, as it hints that models can be pre-trained in a MoE architecture and then distilled while still performing better than just pre-training on a dense architecture.
- When training an MoE model, it would make sense to use expert parallelism in the scenario where a single expert fits into a core, and to use expert, model and data parallelism in the case of a single expert not fitting into a core.
- The observation that increasing the size of each expert (and not the number of experts) is what causes instability is interesting as it shows that this perhaps leads to experts that are too complex for the clusters they are assigned to (although this should be true for an

increase in the number of experts – more experts = smaller clusters for each expert). From intuition, it seems that a balance between number of experts and expert size is needed.