Scaling out Transformer models for Retrosynthesis on Supercomputers

Joris Mollinga and Valeriu Codreanu

SURF Corporative
Amsterdam, The Netherlands,
joris.mollinga@surf.nl

Abstract. Retrosynthesis is the task of building a molecule from smaller precursor molecules. As shown in previous work, good results can be achieved on this task with the help of deep learning techniques, for example with the help of Transformer networks. Here the retrosynthesis task is treated as a machine translation problem where the Transformer network predicts the precursor molecules given a string representation of the target molecule. Previous research has focused on performing the training procedure on a single machine but in this article we investigate the effect of scaling the training of the Transformer networks for the retrosynthesis task on supercomputers. We investigate the issues that arise when scaling Transformers to multiple machines such as learning rate scheduling and choice of optimizer, and present strategies that improve results compared to previous research. By training on multiple machines we are able to increase the top-1 accuracy by 2.5% to 43.6%. In an attempt to improve results further, we experiment with increasing the number of parameters in the Transformer network but find that models are prone to overfitting, which can be attributed to the small dataset used for training the models. On these runs we manage to achieve a scaling efficiency of nearly 70%.

Keywords: computer aided retrosynthesis, deep learning, transformer, high performance computing, deep learning on supercomputers

1 Introduction

Retrosynthesis is a technique from organic chemistry which involves itself with the planning of organic syntheses. In retrosynthesis a molecule is transformed into smaller and simpler precursor molecules until a set of precursor molecules is found which are commercially available. The precursor molecules can then be used to synthesize the molecule of interest. From the perspective of an organic chemist, retrosynthesis is a very challenging task which is often guided by experience and intuition with often multiple viable paths to the molecule of interest.

The first computer aided retrosynthesis code was developed in 1985 by Corey [1]. This program, called LHASA used a template-based approach. It included

a database of templates which were recursively applied to a target molecule. A specific strategy then selects the set of reagents. This process is repeated until a set of commercially available reagents has been found that can be used to synthesize the target molecule. Usually there are multiple valid sets of reagents that can be used and thus the algorithm has to reason about the most effective route. Most early attempts at computer aided retrosynthesis were not very successful, mainly because of the difficulty of maintaining the database with templates.

Recent developments in the fields of machine learning and deep learning have boosted research on computer aided retrosynthesis. Although the retrosynthesis problem can be tackled with different machine learning techniques, a popular choice is to treat it as a machine translation task. The product of the chemical reaction is the source language, while the smaller precursors molecules are the target language. The source and target molecules share a common alphabet and grammar. Both recurrent neural networks [2] and Transformer architectures [3,4] have been applied to this task, where the transformer models showed superior results.

Almost all prior research is conducted on single GPU machines. In this article we investigate the effect of scaling the training of the retrosynthesis task on GPU-enabled and CPU-enabled supercomputers. Previous research has shown that there are great benefits in performing this training task on more than one machine. This is known to speed up training and can provide superior results. We first replicate the results from [3] and then improve upon those results by scaling out training to multiple machines. Lastly we also experiment with increasing the parameter count in the models and using high memory systems. We perform scaling experiments on up to 128 CPU nodes and discuss the challenges that come with scaling to a distributed system like learning rate scheduling and choice of optimizer.

This paper is structured as follows. In section 2 we discuss previous efforts on computer aided retrosynthesis and distributed training. In section 3 we discuss our approach, describing the dataset, the model, and the distributed training procedure. In section 4 we present the experiments. The results of these experiments are given in section 5. Finally, we discuss our results and present a conclusion in section 6 and 7, respectively.

2 Previous work

2.1 Computer aided retrosynthesis

Computer aided retrosynthesis has been an active field since 1985, when the first software package for retrosynthesis was published [1]. During the past decade the advancement in the field of machine learning has also been applied to the task of retrosynthesis. Modern machine-learning based approached can be divided into two categories: (1) template-based approaches and (2) template-free approaches. In this context templates are rules of product describing product and reactant molecules. Templates can be extracted with the help of algorithms or can be

hand-crafted. Because the number of templates is very large these have to be prioritized. The system proposed in [5] learns a conditional distribution over the template set. In [6] templates are sorted using a ranking mechanism. An alternative approach is used in [7], where the joint distribution of templates and reactions is modeled directly with logic variables.

Template-free methods learn a direct mapping from products to reactants. Template-free methods often use machine translation techniques like sequence to sequence (seq2seq) models as in [2] or Transformers models as in [3]. The methods use the SMILES [8] representation of molecules which is a string based representation. By treating the problems as a machine translation problem the reactants are generated token by token, or in this context, atom by atom. Different improvements have been suggested to improve template-free methods, like data augmentation [4, 9] and a syntax corrector [10]. Another template-free approach is the work presented in [11] which uses Monte Carlo Tree Search (MCTS) combined with neural networks for retrosynthesis planning.

2.2 Distributed training

In the quest for developing increasingly complex and expressive neural network architectures demanding more and more computational resources, distributed training has become commonplace and the only way forward to sustain innovation in the field. This is most clearly outlined by the GPT-3 [12] language model featuring 175 billion parameters and requiring $3.14E + 23$ floating point operations to train. These state of the art deep learning models are currently impossible to train on a single GPU or CPU. However, scaling out from a single machine to multiple machines is not a straightforward task and may yield inferior results if not performed correctly. When training a model in a distributed fashion, several training strategies can be adopted, with the most prominent being: *data parallel*, *model parallel*, and *pipeline parallel* training.

In data parallel training, each worker holds a complete copy of the model, and each training batch gets split across the workers. After the forward pass is completed independently on each worker, the gradients are averaged across all workers during the backward pass, allowing for the neural network parameters to be updated. The advantage of data parallel training is that the workload is uniform, leading to low degrees of load imbalance, particularly if gradient communication is overlapped with computation. However, the disadvantage of this training scheme is that the global batch size (i.e. batch size aggregated across all workers) grows linearly with the number of workers, and this is known to lead to the generalization gap [13] phenomenon, with the proposed solution to increase the number of updates performed in the optimization process. Still, this would cancel most of the benefits of distributed data-parallel training. Nonetheless, there are alternative ways to address the generalization gap, mostly focused on more evolved learning rate schedules and improved optimization techniques [14], [15], [16].

In model parallel training one divides the model over different workers. This is especially useful when dealing with models that have very high memory re-

quirements and do not fit in the memory of a single device. This comes at the expense of high training times, as one worker has to wait for the output of the previous worker, leading to an inherently sequential process. More modern pipeline-parallel approaches such as GPipe [17] and PipeDream [18] aim to split a batch into a collection of "micro-batches" to address the computational inefficiencies of model parallel training. This allows the time per training iteration to be reduced, although some overhead related to the "bubbles" in the pipeline remains.

In this work we use a data parallel training strategy for the proposed Transformer model, since we consider relatively small models which fit on a single GPU and therefore do not need model parallel training strategies. However, due to the fact that the dataset used is relatively small, the number of updates per epoch decreases heavily as we increase the number of parallel workers. This leads to the generalization gap phenomenon, and ways to overcome it will be discussed in remainder sections of the current work. We use a number of guidelines from [19] and [20] for training models on multiple machines and continue to meet or exceed the accuracy targets.

3 Approach

3.1 Dataset

For this research the same dataset as in [3] is used. The dataset contains approximately 50,000 chemical reactions from the USPTO database [21], split in ten reaction classes using the SMILES notation. The SMILES notation is a string notation of molecules. Fig. 1 shows an example of a molecule and its corresponding SMILES notation. Each character in the SMILES notation is tokenized according to the same procedure as in [3]. This gives a vocabulary size of 66 tokens, including a start-of-sequence, end-of-sequence and padding token. The dataset contains 40,029, 5,004, and 5,004 examples for training, validation and testing, respectively. Just like in [3] we combine the training and validation set to generate a larger training set of 45,033 samples. In this work we discard the reaction class.

3.2 Transformer model

A Transformer model [22] is a type of encoder-decoder neural network. It has shown promising performance on machine translation tasks and other sequence to sequence problems. The transformer architecture does not use any convolutional or recurrent architectures but uses the principle of multi-head self-attention: a technique to learn what part of the input is important. The input to the multi-head attention layer is a set of token embeddings $\mathbf{h} = \{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_N\},\ \mathbf{h}_i \in \mathbb{R}^{d_h}$ with the output being a separate set of token embeddings $\mathbf{h}'$ of the same size: $\mathbf{h}' = \{\mathbf{h}'_1, \mathbf{h}'_2, ..., \mathbf{h}'_N\},\ \mathbf{h}'_i \in \mathbb{R}^{d_h}$, where N is the sequence length. The first

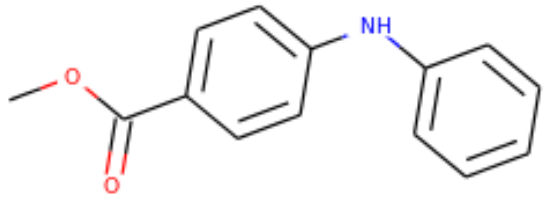

Fig. 1. A molecule and its SMILES representation.

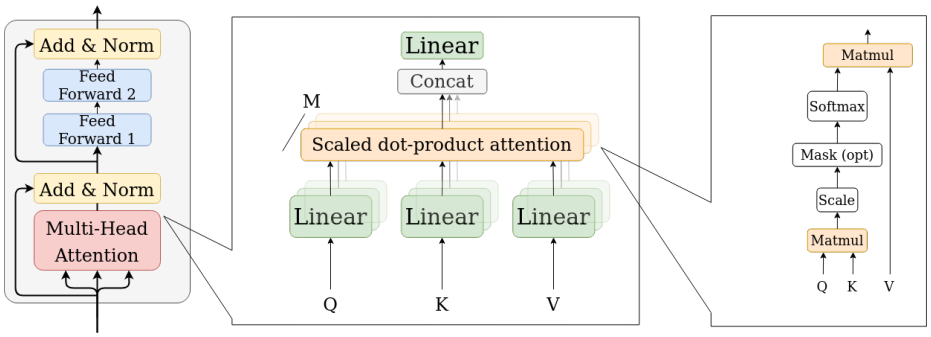

Fig. 2. Multi-head attention architecture.

step in the attention mechanism is to get the query, key and value vector by projecting the token embedding:

$$\mathbf{q}_i = W_Q \mathbf{h}_i,\ \mathbf{k}_i = W_K \mathbf{h}_i,\ \mathbf{v}_i = W_V \mathbf{h}_i \tag{1}$$

Here W_Q, W_K and W_V are different weight matrices. W_Q and W_K are of shape $d_k \times d_h$ and W_V is of shape $d_v \times d_h$. Here d_k and d_v are the key and value dimensionality. We then compute the compatibility of the key and query pair.

$$u_{ij} = \frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d_k}} \tag{2}$$

The next step is to calculate the attention scores using a soft-max function:

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{ij'}}} \tag{3}$$

Lastly, the output vector $\mathbf{h}'_i$ is a weighted average of the attention score and value vector:

$$\mathbf{h}'_i = \sum_j a_{ij} \mathbf{v}_j \tag{4}$$

In [23], Veličković et al. found that using multiple attention heads gives better performance. This allows each node to send different kinds of messages to each neighbor. To do this, Equation 1 to 4 are calculated M times with $d_k = d_v = \frac{d_h}{M}$. This produces M different outputs called $\mathbf{h}'_{im}$, $m \in 1, ..., M$ which are combined using Equation 5. Here W_m^O is a matrix of shape $d_h \times d_v$.

$$\mathbf{h}'_i = \sum_{m=0}^{M} W_m^O \mathbf{h}'_{im} \tag{5}$$

Lastly the output of the multi-head self-attention layer is added to the original data, layer-wise normalized and fed through a couple of dense layers with again a skip-connection. We refer to the output dimension of the first feedforward layer and the input dimension of the second feedforward layer as *hidden size*. A schematic overview of the multi-head attention layer is shown in Fig. 2.

Transformer networks look at the entire input sequence at once, instead of analyzing it per token like recurrent models. This alleviates problems that can be caused by long range dependencies between tokens. To give the model more understanding of the importance of the position of tokens in the sequence we also use positional embeddings [22] that are added to the token embeddings. The positional embeddings are implemented using the same procedure as in [22]. Unless stated otherwise we use 3 layers in the encoder and decoder, 8 attention heads and an embedding, key size and value size of 64.

For inference, we can feed the source sequence to the encoder. This embedded source sequence is then fed to the decoder which generates a target sequence step by step by taking the previous characters it has predicted into account. Upon reporting our results on the test set, we provide results from both a greedy decoding strategy as well as from a beam search decoding strategy [24]. A greedy decoding strategy selects the next token by choosing the token with the highest probability while beam search explores multiple options and chooses the sequence of tokens that eventually leads to the highest probability. This allows for better inference and exploration of the probability landscape.

3.3 Distributed training

We scale the training procedure to multiple machines using Horovod [25]. Horovod is a library for distributed data parallel training, working together with various machine learning frameworks. With Horovod, each machine (or worker) holds a local replica of the model. After the forward pass on each worker the gradients of all workers are communicated to all other workers and averaged after which each worker performs the backward pass. This synchronization point ensures that all model parameters on all workers are identical after each update step. Of course, one has to broadcast the weights of the model at the beginning of training. One critical step we take in order to allow for efficient scaling of Transformer models is converting assumed-sparse tensors to dense tensors, as described in [26]. This is easily achieved by enabling the *sparse_as_dense* flag from Horovod's *DistributedOptimizer* class. Apart from syncing the gradient updates, training using

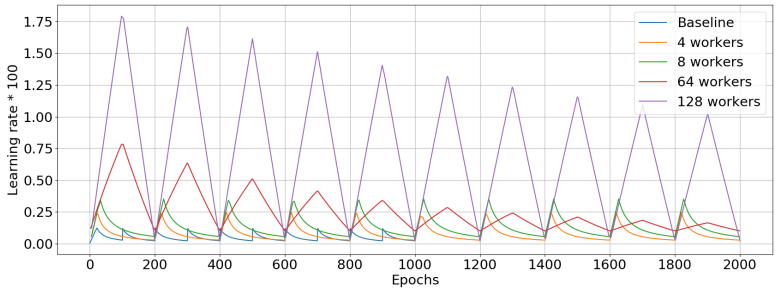

Fig. 3. Learning rate schedules for various workers

multiple workers is not straightforward and one has to take a number of considerations into account for efficient distributed training. These are discussed in the next sections.

Dataset replication Each worker loads a (local) copy of the dataset. To generate batches each worker samples B samples from the dataset, where B is the batch size on a specific worker. By approximation this ensures that each data point is used for training equally often. In order to allow for each worker to draw a random sequence of B samples, we use different per-worker random seeds.

Learning rate scheduling Transformer models are known to benefit from gradually warming up the learning rate. The original Transformer paper [22] uses $4,000$ warmup steps and a decay inversely proportional to the number of optimizer steps taken after the warmup phase. Our baseline method from [3] reports $16,000$ warmup steps as an empirically found number of warmup steps for this specific retrosynthesis task. When training Transformers in a distributed fashion, [19] advises to scale the number of warmup steps inversely proportionally to the number of workers, but less than linearly.

After the warmup phase, [22] lets the learning rate decrease proportional to the inverse square root of the number of steps taken. The work from [3] uses a slightly more aggressive decay proportional to the inverse of the number of steps taken. According to [27], applying a cyclic learning rate schedule where the learning rate resets to the maximum value (that is the value reached at the end of the warmup phase) improves results. We also adopt this strategy.

An alternative learning rate scheduling strategy is the *exponential rate* decay strategy from [28]. In this strategy the learning rate is also cyclical but the warmup and decay are linear. Furthermore the maximum learning rate decreases according to an exponential decay. We refer the reader to Fig 3 for a graphical comparison of the different learning rate schedules used in this work.

Choice of optimizer During training we found that the performance of the model is very sensitive to the choice of ϵ for the Adam optimizer especially with more than 8 workers. A higher value of ϵ yields less numerical instabilities but reduces the performance. Numerical under or overflow results in *NaN* values (Not a Number) during training upon which training terminates. In order to mitigate this we train with the Adamax optimizer from [29] which does not suffer from this problem.

3.4 System specifications

All computations are done on worker nodes of the LISA Compute Cluster at the Dutch National Supercomputing Centre SURFSara[1], consisting of dual-socket Intel Xeon Gold 5118 CPUs and four NVIDIA Titan RTX GPUs with 24 GB of GDDR6 SDRAM. We also performed experiments for larger number of workers on the Endeavour Supercomputer[2], as this also enabled us to experiment with larger Transformer models. The Endeavour system consists of dual-socket Intel Xeon 8260L nodes with 192GB of memory per node. For runs on Endeavour we use two processes per node which allows us to double the number of workers on the same number of nodes. Each process is pinned to a separate socket, maximizing the throughput of the system. For all our experiments we use Python 3.6.10 with Tensorflow(-gpu) 1.15.3. We use Horovod 0.19.5 together with NCCL 2.4.7 (in the GPU case) and Intel MPI (in the CPU case) for the distributed communication backend.

4 Experiments

In this work, our first goal was to replicate the model and results from [3]. We then gradually increase the number of workers for training and optimize the hyper parameters for that specific number of workers. For an increasing number of workers we found that changes to the hyper parameters were required for numerical stability and to improve the validation accuracy of the models. These changes are discussed below. Lastly we experiment with wider models by increasing the number of heads and the hidden size on when using large memory machines. Runs with 1, 4 and 8 workers are performed on GPUs on the LISA Compute Cluster. Runs with 64 and 128 workers are performed on the Endeavour Supercomputer. We summarize the hyper-parameters and experimental settings used in Table 1.

1 worker as a baseline the exact same training setup and model architecture reported in [3] are used. The learning rate schedule is given by 6 and 7.

$$\alpha = \lambda \cdot \frac{\min(1, u(step), warmup)}{\max(u(step), warmup)} \tag{6}$$

[1] https://userinfo.surfsara.nl/
[2] https://www.top500.org/system/176908

Table 1. Hyper parameters for various experiments

Hyper	**# workers**						
parameters	1 [3]	4	8	64	128	128-L1	128-L2
Batch size	64	64	64	32	32	32	32
# heads	8	8	8	8	8	8	24
Key size	64	64	64	64	64	128	64
Embedding size	64	64	64	64	64	128	64
Hidden size	512	512	512	512	512	1024	2048
# parameters	1,587,264	1,587,264	1,587,264	1,587,264	1,587,264	6,320,256	5,135,424
# epochs	1,000	2,000	2,000	2,000	2,000	1,600	2,000
Optmizer		Adam	Adam	Adamax	Adamax	Adamax	Adamax
LR scheduler	Cyclic w/o warm restarts	Cyclic w warm restarts	Cyclic w warm restarts	Exp range	Exp range	Exp range	Exp range

$$u(step) = \begin{cases} warmup + (step \ \% \ cycle) & \text{if } step \geq cycle \\ step & \text{otherwise} \end{cases} \tag{7}$$

Here $warmup$ is the number of warmup steps, λ is a constant, $cycle$ is the number of steps per cycle and $step$ is the number of optimizer steps taken. We use $16,000$ warmup steps and set λ to 20. Training is conducted for $1,000$ epochs, similarly to [3], with a batch size of 64. We show the learning rate in Fig. 3. Lastly we also average the checkpoints of the last 5 cycles after training.

4 and 8 workers for training on 4 and 8 workers we found that dividing the number of warmup steps by $2\sqrt{n_{workers}}$ gives stable training. We train for $2,000$ epochs with a batch size of 64 and reset the learning rate every 200 epochs. We found that a sudden increase in the learning rate at the start of a new cycle often results in numerical instabilities and therefore we added a warmup phase to the learning rate in every cycle. During this warmup the learning rate is increased to the maximum learning rate with the same slope as during the initial warmup. The value for λ is left at 20. The model architecture is the same as in [3]. Similarly to [3] we average the checkpoints of the last 5 cycles.

64 workers and 128 workers for training on 64 and 128 workers we found that training with the exponential-range cyclic learning rate from [28] results in stabler training. This learning rate schedule is parametrized by Equation 8 and 9.

$$\alpha = \alpha_{min} + \gamma^{cycle} \cdot (\alpha_{max} - \alpha_{min}) \cdot \max(0, 1 - x) \tag{8}$$

$$x = |\frac{step}{warmup} - 2 \cdot l_{cycle} + 1| \tag{9}$$

Here γ is a parameter for controlling the exponential decay, $cycle$ is the index of the current cycle, α_{max} and α_{min} are the maximum and minimum learning rate

respectively, *step* is the number of steps taken, *warmup* is the number of warmup steps and l_{cycle} is the number of steps per cycle. γ is set to 0.99994, α_{min} is set to 0.0002 and α_{max} is set to 0.0075 and 0.0175 for 64 and 128 workers, respectively. We set $warmup$ to $0.5 \cdot l_{cycle}$ which in turn can be calculated with the bacth size, epochs per cycle (200) and number of workers. We refer the reader to Fig 3 to graphically compare learning rate schedules. We lower the batch size to 32 per worker, and apply the Adamax optimizer. Again we train for $2,000$ epochs and average the last 5 cycles.

128 workers - Large models we experiment with larger models by increasing the number of heads, the key size, the embedding size and the hidden size. We report two different experimental settings which we call "L1" and "L2". For L1 we increase the key size and embedding size to 128 and increase the hidden size to $1,024$. For L1 train for $1,600$ epochs and report results on the last checkpoint while for L2 we increase the number of heads to 24 and increase the hidden size to $2,048$. Changing the key size and the hidden size increases the number of parameters in the model, allowing for more learning capacity. By increasing the number of attention heads we hope the model is able to focus on more parts of the input sequence leading to better results. The learning rate scheduling options are the same as the 128 worker scenario from above. We train for $2,000$ epochs and average weights over the last 5 cycles.

5 Results

5.1 Accuracy on test set

For the training runs with varying number of workers we report the accuracy of correctly predicting the source molecules on the test with a greedy decoding strategy and with a beam search decoding strategy. We use a beam size of 5 and report the $top-1$ accuracy. We compare our results to the results from [3]. These results are shown in Table 2. The results presented in this table is percentage of source molecules predicted correctly from the test set. In order to predict an entire molecule correctly, the model will need to predict all the atoms correctly. Because a molecule can have up to 160 atoms, the model can easily make a mistake. This explains the difference in performance between the character (or atom) based accuracy (reported later on) and the greedy and $top-1$ accuracy scores.

The results indicate that there is a benefit to training on distributed systems. Following the greedy decoding strategy, the performance on the test set is improved by 1.3%, 2.5% and 0.5% for 4, 8 and 64 workers, respectively. Following the beam search decoding strategy this yields an improvement of 2.1% , 1.6% and -0.2%, respectively. Up to 64 workers either the accuracy using a greedy decoding strategy or the accuracy using a beam search decoding strategy improves with respect to the baseline. After that it no longer improves. We assume this is because when training using so many workers, the optimizer does not perform

Table 2. Results for various experiments. Results in **bold** indicate the best results in that column.

# workers	Results	
	Greedy	***Top-1***
1 - [3]	40.6%	42.7%
4	41.1%	**43.6%**
8	**41.6%**	43.4%
64	40.8%	42.6%
128	37.6%	40.1%
128 - L1	38.9%	40.6%
128 - L2	40.4%	41.6%

enough gradient updates to improve with respect to the baseline (11 gradient updates per training epoch), a conclusion similar to the one from [13]. Training larger models (the L1 and L2 run) does not improve results with respect to the baseline, however these models that were also trained on 128 workers improve upon the base model trained on similar number of workers. Because of the high number of parameters these models tend to overfit on the relatively small training set used, resulting in poorer performance on the test set. Evaluation on the training set shows a training accuracy of 99.6% proving the points on overfitting.

5.2 Scaling performance

For evaluating the scaling performance we report the ***time per epoch*** which is the amount of time it takes to evaluate the entire training dataset once. From this metric we calculate the ***computational speedup factor***. We also report the ***time till score*** which is the time it takes to reach a validation accuracy of 97.0%. From these metrics we calculate the ***convergence speedup factor***. This 97.0% is based on character (or atom) prediction performance, not the capability of predicting the entire molecule correctly. This was also discussed in Sect. 5.1. We split the analysis in training performed on GPUs and training performed on CPUs in order to have a fair comparison. We present the scaling results using GPUs in Table 3 and the scaling results using CPUs in Fig. 4.

Table 3. Scaling performance on GPU.

# of workers	Results			
	Time per epoch	***Time till score***	***Computational speedup factor***	***Convergence speedup factor***
1 - [3]	125.6s	13h 57m	1×	1×
4	36.9s	3h 35m	3.4×	3.9×
8	21.0s	2h 11m	6.0×	6.4×

GPU the time needed to train is drastically reduced by training on an increasing number of machines. If scaling were to be ideal we would expect a speedup of 4× when training on 4 workers for both the computational speedup factor and convergence speedup factor. The scaling performance in terms of convergence speedup is near optimal for 4 workers but this performance reduced when running with 8 workers. The scaling performance in term of the computational speedup factor are sub-optimal. We attribute this sub-optimal scaling to the relatively low computational complexity of the models. The computational footprint of the models is not very high, therefore the overhead introduced by communicating gradients across workers is relatively large compared to the time spent on computation. Furthermore it can be seen that scaling performance gets worse with increasing number of workers which is to be expected because the overhead introduced by communication to other workers is larger with more workers.

CPU in order to make a fair comparison we also performed a number of training runs with up to 256 CPU workers in order to measure the *time per epoch* metric. All CPU training runs use 2 workers per node, as this makes optimal use of a dual-socket system. The computational speedup factors for these runs tend to exceed the ones on GPUs, even though the per-worker batch size in the CPU case is only 32. We do not report the results of evaluating these training runs on the test set since these are identical to the runs on GPUs. We show the scaling performance in Fig. 4. We obtain a scaling efficiency close to 70%, leading to dramatic reductions in training times. Furthermore, we perform scaling runs for the L2 large model described previously. These results are also presented in Fig. 4. The speed-up factors achieved on the L2 model are similar to the ones for the original model, and the limitations are mostly due to the relatively small batch size that we use per node (32) in order to achieve reasonable validation accuracy for the trained models. Besides measuring the computational speed-up factors in terms of training throughput, we also present 4 runs going from 8 workers to 64 workers, in order to assess the time it takes to reach certain model accuracy. Fig. 5 outlines the benefits of the proposed distributed training system, showing the amount of time saved for reaching a given accuracy target when increasing the node count. All models reach at least 97.0% accuracy on the test set, as also discussed in the previous section. The sudden jumps in the accuracy are caused by the learning rate entering a new cycle and not all runs reported in Fig. 5 have been completed, as a limited training budget of 12 hours per run was used for this comparison.

6 Discussion

After extensive experimentation with scaling out Transformer models for this retrosynthesis problem, we realize that the bottleneck is the size of the training dataset. This dataset only contains $45,033$ samples and with 128 workers in just 11 iterations with a per-worker batch size of 32 examples, the entire data sat has been fed to the model once. This limits the scale at which we can train,

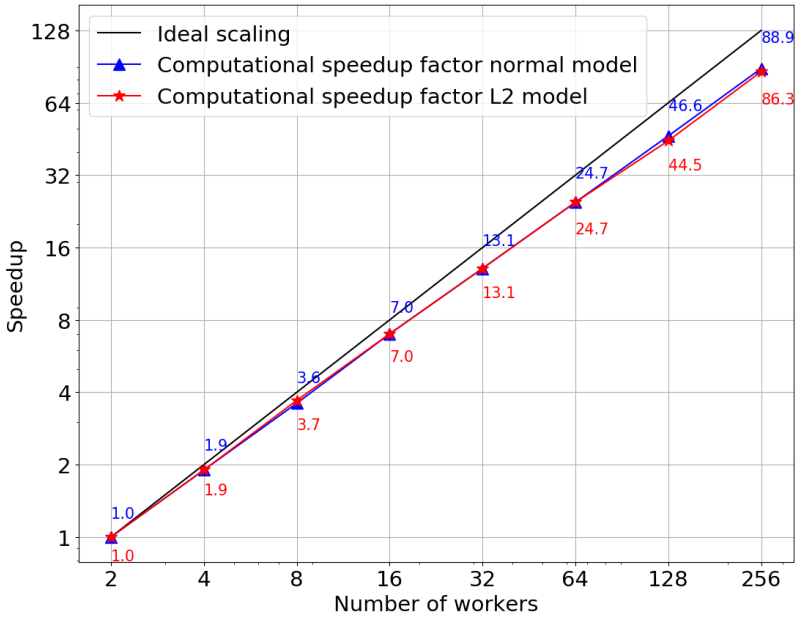

Fig. 4. Scaling performance on CPU.

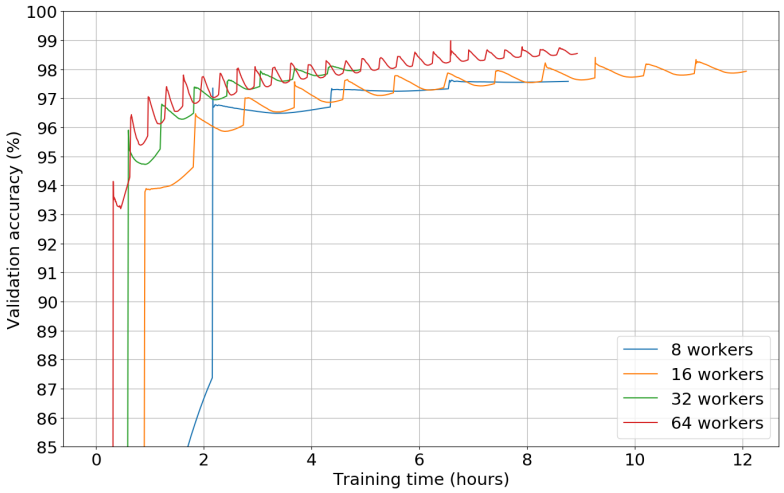

Fig. 5. Validation accuracy for increasing number of workers.

and also the scaling efficiency due to the fact that small per-worker batch sizes can be employed. We think this problem would benefit from a larger dataset, both in terms of solution quality and scaling performance. Even so, we show it is possible to speed up training by a factor of 24.7× using 32 compute nodes (64 workers) and reach reasonable accuracy. We have tried to train our models with augmented data from [4] but found this yield inferior results. The small USPTO-50k dataset used in this article warrants overfitting. This is confirmed by the high training accuracy for the wider models which tend to overfit even sooner. This is a known problem when training on small datasets, and in the context of retrosynthesis, [30] mitigates this by pretraining their models on the USPTO-380K dataset after which the model is fine tuned on the USPTO-50k dataset. It is left for further research to investigate how this approach can be applied when training at scale.

7 Conclusion

In this work we have used the Transformer model from [3] and trained it in a distributed fashion using both GPU-enabled nodes and a large-scale CPU-based supercomputer. We have shown that the top-1 accuracy increases by 2.5% when training on a distributed system with 8 workers. Furthermore we shown an improvement in accuracy on a distributed system with up to 64 workers. This trend does not continue when training on a system with 128 workers. We also experimented with heavier models composed of more parameters in an attempt to increase performance. However, these experiments were not successful and led to overfitting. We attribute this to the small size of the training dataset used in this specific task ($45,033$ examples) and speculate that pretraining on a larger dataset would boost performance. However, we leave this investigation for future work. Lastly we discussed issues inherent to scaling transformer models to multiple machines, such as learning rate scheduling and optimizer choices. We show that using the *exponential range* learning rate scheduling scheme does not result in under or overflow issues during training, and mitigates the effects of large-batch training. Furthermore we found the Adamax optimizer to be more stable compared to Adam. On this specific problem we manage to achieve a scaling efficiency of nearly 70% on 128 nodes (256 workers).

Acknowledgment

We thank the anonymous referees for their constructive comments, which helped to improve the paper. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 814416. We would also like to acknowledge Intel for providing us the resources to run on the Endeavour Supercomputer.

References

1. Elias James Corey, Alan K Long, and Steward D Rubenstein. Computer-assisted analysis in organic synthesis. *Science*, 228(4698):408–418, 1985.
2. Bowen Liu, Bharath Ramsundar, Prasad Kawthekar, Jade Shi, Joseph Gomes, Quang Luu Nguyen, Stephen Ho, Jack Sloane, Paul Wender, and Vijay Pande. Retrosynthetic reaction prediction using neural sequence-to-sequence models. *ACS central science*, 3(10):1103–1113, 2017.
3. Pavel Karpov, Guillaume Godin, and Igor V Tetko. A transformer model for retrosynthesis. In *International Conference on Artificial Neural Networks*, pages 817–830. Springer, 2019.
4. Igor V Tetko, Pavel Karpov, Ruud Van Deursen, and Guillaume Godin. Augmented transformer achieves 97% and 85% for top5 prediction of direct and classical retrosynthesis. *arXiv preprint arXiv:2003.02804*, 2020.
5. Marwin HS Segler and Mark P Waller. Neural-symbolic machine learning for retrosynthesis and reaction prediction. *Chemistry–A European Journal*, 23(25):5966–5971, 2017.
6. Connor W Coley, Luke Rogers, William H Green, and Klavs F Jensen. Computer-assisted retrosynthesis based on molecular similarity. *ACS central science*, 3(12):1237–1245, 2017.
7. Hanjun Dai, Chengtao Li, Connor Coley, Bo Dai, and Le Song. Retrosynthesis prediction with conditional graph logic network. In *Advances in Neural Information Processing Systems*, pages 8872–8882, 2019.
8. David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.
9. Esben Jannik Bjerrum. Smiles enumeration as data augmentation for neural network modeling of molecules. *arXiv preprint arXiv:1703.07076*, 2017.
10. Shuangjia Zheng, Jiahua Rao, Zhongyue Zhang, Jun Xu, and Yuedong Yang. Predicting retrosynthetic reactions using self-corrected transformer neural networks. *Journal of Chemical Information and Modeling*, 60(1):47–55, 2019.
11. Marwin HS Segler, Mike Preuss, and Mark P Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610, 2018.
12. Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
13. Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.
14. Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
15. Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.
16. Valeriu Codreanu, Damian Podareanu, and Vikram Saletore. Scale out for large minibatch sgd: Residual network training on imagenet-1k with improved accuracy and reduced time to train. *arXiv preprint arXiv:1711.04291*, 2017.

17. Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in neural information processing systems*, pages 103–112, 2019.
18. Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.
19. Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, 2018.
20. Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*, 2018.
21. Daniel Mark Lowe. Extraction of chemical structures and reactions from the literature. 2012.
22. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
23. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2017.
24. Lowerre Goodman and Raj Reddy. Effects of branching factor and vocabulary size on performance. *Speech understanding systems: summary of results of the five-year research effort at Carnegie-Mellon University.*, page 39.
25. Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.
26. Derya Cavdar, Valeriu Codreanu, Can Karakus, John A Lockman, Damian Podareanu, Vikram Saletore, Alexander Sergeev, Don D Smith, Victor Suthichai, Quy Ta, et al. Densifying assumed-sparse tensors. In *International Conference on High Performance Computing*, pages 23–39. Springer, 2019.
27. Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.
28. Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.
29. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
30. Renren Bai, Chengyun Zhang, Ling Wang, Chuansheng Yao, Jiamin Ge, and Hongliang Duan. Transfer learning: Making retrosynthetic predictions based on a small chemical reaction dataset scale to a new level. *Molecules*, 25(10):2357, 2020.