The Evolved Transformer

David R. So [1] **Chen Liang** [1] **Quoc V. Le** [1]

Abstract

Recent works have highlighted the strength of the *Transformer* architecture on sequence tasks while, at the same time, *neural architecture search* (NAS) has begun to outperform human-designed models. Our goal is to apply NAS to search for a better alternative to the Transformer. We first construct a large search space inspired by the recent advances in feed-forward sequence models and then run evolutionary architecture search with *warm starting* by seeding our initial population with the Transformer. To directly search on the computationally expensive WMT 2014 English-German translation task, we develop the *Progressive Dynamic Hurdles* method, which allows us to dynamically allocate more resources to more promising candidate models. The architecture found in our experiments – the *Evolved Transformer* – demonstrates consistent improvement over the Transformer on four well-established language tasks: WMT 2014 English-German, WMT 2014 English-French, WMT 2014 English-Czech and LM1B. At a big model size, the Evolved Transformer establishes a new state-of-the-art BLEU score of 29.8 on WMT'14 English-German; at smaller sizes, it achieves the same quality as the original "big" Transformer with 37.6% less parameters and outperforms the Transformer by 0.7 BLEU at a mobile-friendly model size of ∼7M parameters.

1. Introduction

Over the past few years, impressive advances have been made in the field of neural architecture search. Reinforcement learning and evolution have both proven their capacity to produce models that exceed the performance of those designed by humans (Real et al., 2019; Zoph et al., 2018). These advances have mostly focused on improving vision models, although some effort has also been invested in searching for sequence models (Zoph & Le, 2017; Pham et al., 2018). In these cases, it has always been to find improved recurrent neural networks (RNNs), which were long established as the de facto neural model for sequence problems (Sutskever et al., 2014; Bahdanau et al., 2015).

However, recent works have shown that there are better alternatives to RNNs for solving sequence problems. Due to the success of convolution-based networks, such as Convolution Seq2Seq (Gehring et al., 2017), and full attention networks, such as the Transformer (Vaswani et al., 2017), feed-forward networks are now a viable option for solving sequence-to-sequence (seq2seq) tasks. The main strength of feed-forward networks is that they are faster, and easier to train than RNNs.

The goal of this work is to examine the use of neural architecture search methods to design better feed-forward architectures for seq2seq tasks. Specifically, we apply *tournament selection* architecture search and *warm start* it with the Transformer, considered to be the state-of-art and widely-used, to evolve a better and more efficient architecture. To achieve this, we construct a search space that reflects the recent advances in feed-forward seq2seq models and develop a method called *Progressive Dynamic Hurdles* (PDH) that allows us to perform our search directly on the computationally demanding WMT 2014 English-German (En-De) translation task. Our search produces a new architecture – called the *Evolved Transformer* (ET) – which demonstrates consistent improvement over the original Transformer on four well-established language tasks: WMT 2014 English-German, WMT 2014 English-French (En-Fr), WMT 2014 English-Czech (En-Cs) and the 1 Billion Word Language Model Benchmark (LM1B). At a big model size, the Evolved Transformer establishes a new state-of-the-art BLEU score of 29.8 on WMT'14 En-De. It is also effective at smaller sizes, achieving the same quality as the original "big" Transformer with 37.6% less parameters and outperforming the Transformer by 0.7 BLEU at a mobile-friendly model size of ∼7M parameters.

2. Related Work

RNNs have long been used as the default option for applying neural networks to sequence modeling (Sutskever

[1]Google Research, Brain Team, Mountain View, California, USA. Correspondence to: David R. So <davidso@google.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

et al., 2014; Bahdanau et al., 2015), with LSTM (Hochreiter & Schmidhuber, 1997) and GRU (Cho et al., 2014) architectures being the most popular. However, recent work has shown that RNNs are not necessary to build state-of-the-art sequence models. For example, many high performance convolutional models have been designed, such as WaveNet (Van Den Oord et al., 2016), Gated Convolution Networks (Dauphin et al., 2017), Conv Seq2Seq (Gehring et al., 2017) and the Dynamic Lightweight Convolution model (Wu et al., 2019). Perhaps the most promising architecture in this direction is the Transformer architecture (Vaswani et al., 2017), which relies only on multi-head attention to convey spatial information. In this work, we use both convolutions and attention in our search space to leverage the strengths of both of these layer types.

The recent advances in sequential feed-forward networks are not limited to architecture design. Various methods, such as BERT (Devlin et al., 2018) and Radford et. al's (2018) pre-training technique, have demonstrated how models such as the Transformer can improve over RNN pre-training (Dai & Le, 2015; Peters et al., 2018). For translation specifically, work on scaling up batch size (Ott et al., 2018; Wu et al., 2019), using relative position representations (Shaw et al., 2018), and weighting multi-head attention (Ahmed et al., 2017) have all pushed the state-of-the-art for WMT'14 En-De and En-Fr. However, these methods are orthogonal to this work, as we are only concerned with improving the neural network architecture itself, and not the techniques used for improving overall performance.

The field of neural architecture search has also seen significant recent progress. The best performing architecture search methods are those that are computationally intensive (Zoph & Le, 2017; Baker et al., 2016; Real et al., 2017; Xie & Yuille, 2017; Zoph et al., 2018; Real et al., 2019). Other methods have been developed with speed in mind, such as DARTS (Liu et al., 2018b), ENAS (Pham et al., 2018), SMASH (Brock et al., 2018), and SNAS (Xie et al., 2019). These methods radically reduce the amount of time needed to run each search by approximating the performance of each candidate model, instead of investing resources to fully train and evaluate each candidate separately. However, these methods also have several disadvantages that make them hard to apply in our case: (1) It is hard to warm start these methods with the Transformer, which we found to be necessary to yield strong results. (2) ENAS and DARTS require too much memory at the model sizes we are searching for. (3) The best architecture in the vision domain (e.g., AmoebaNet(Real et al., 2019)) was discovered by evolutionary NAS, not these efficient methods, and we optimize for best architecture over best search efficiency here.

Zela et. al's (2018) utilization of Hyperband (Li et al., 2017) and PNAS's (Liu et al., 2018a) incorporation of a surrogate model are examples of approaches that try to both increase efficiency via candidate performance estimation and maximize search quality by training models to the end when necessary. The Progressive Dynamic Hurdles (PDH) method we introduce here is similar to these approaches in that we train our best models to the end, but optimize efficiency by discarding unpromising models early on. However, it is critically different from comparable algorithms such as Hyperband and Successive Halving (Jamieson & Talwalkar, 2016) in that it allows the evolution algorithm to dynamically select new promising candidates as the search progresses; Hyperband and Successive Halving establish their candidate pool a priori, which we demonstrate is ineffective in our large search space in Section 5.

3. Methods

We employ evolution-based architecture search because it is simple and has been shown to be more efficient than reinforcement learning when resources are limited (Real et al., 2019). We use the same *tournament selection* (Goldberg & Deb, 1991) algorithm as Real et al. (2019), with the aging regularization omitted, and so encourage the reader to view their in-depth description of the method. In the interest of saving space, we will only give a brief overview of the algorithm here.

Tournament selection evolutionary architecture search is conducted by first defining a *gene encoding* that describes a neural network architecture; we describe our encoding in the following Search Space subsection. An initial *population* is then created by randomly sampling from the space of gene encodings to create *individuals*. These individuals, each corresponding to a neural architecture, are trained and assigned *fitnesses*, which in our case are the models' negative log perplexities on the WMT'14 En-De *validation set*. The population is then repeatedly *sampled* from to produce *subpopulations*, from which the individual with the highest fitness is selected as a *parent*. Selected parents have their gene encodings *mutated* – encoding fields randomly changed to different values – to produce *child models*. These child models are then assigned a fitness via training and evaluating on the target task, as the initial population was. When this fitness evaluation concludes, the population is sampled from once again, and the individual in the subpopulation with the lowest fitness is *killed*, meaning it is removed from the population. The newly evaluated child model is then added to the population, taking the killed individual's place. This process is repeated and results in a population with high fitness individuals, which in our case represent well-performing architectures.

3.1. Search Space

Our encoding search space is inspired by the NASNet search space (Zoph et al., 2018), but is altered to allow it to express architecture characteristics found in recent state-of-the-art feed-forward seq2seq networks. Crucially, we ensured that the search space can represent the Transformer, so that we could seed the initial population with it.

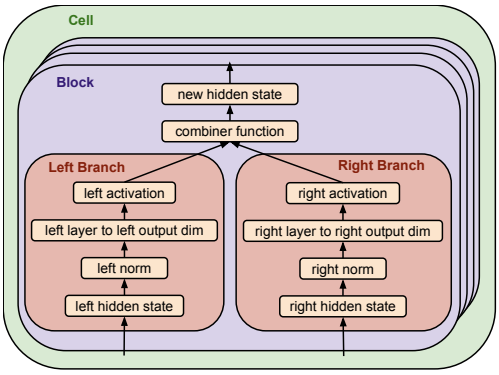

Figure 1. **Architecture composition from encoding.** Each block produces a new hidden state that is added to the pool of hidden states subsequent blocks can select as branch inputs. Each encoder has 6 unique blocks per cell and each decoder has 8 unique blocks per cell. Each cell is repeated *number of cells* times.

Our search space consists of two stackable cells, one for the model encoder and one for the decoder (see Figure 1). Each cell contains NASNet-style blocks, which receive two hidden state inputs and produce new hidden states as outputs; the encoder contains six blocks and the decoder contains eight blocks, so that the Transformer can be represented exactly. The blocks perform separate transformations to each input and then combine the transformation outputs together to produce a single block output; we will refer to the transformations applied to each input as a *branch*. Our search space contains five branch-level search fields (input, normalization, layer, output dimension and activation), one block-level search field (combiner function) and one cell-level search field (number of cells).

In our search space, a child model's genetic encoding is expressed as: [*left input*, *left normalization*, *left layer*, *left relative output dimension*, *left activation*, *right input*, *right normalization*, *right layer*, *right relative output dimension*, *right activation*, *combiner function*] $\times$ 14 + [*number of cells*] $\times$ 2, with the first 6 blocks allocated to the encoder and the latter 8 allocated to the decoder. Given the vocabularies described in the Supplementary Materials, this yields a search space of $7.30 * 10^{115}$ models, although we do shrink this to some degree by introducing constraints (see the Supplementary Materials for more details).

3.2. Seeding the Search Space with Transformer

While previous neural architecture search works rely on well-formed hand crafted search spaces (Zoph et al., 2018), we intentionally leave our space minimally tuned, in a effort to alleviate our manual burden and emphasize the role of the automated search method. To help navigate the large search space, we find it easier to warm start the search process by seeding our initial population with a known strong model, in this case the Transformer. This anchors the search to a known good starting point and guarantees at least a single strong potential parent in the population as the generations progress. We offer empirical support for these claims in our Results section.

3.3. Evolution with Progressive Dynamic Hurdles

The evolution algorithm we employ is adapted from the tournament selection evolutionary architecture search proposed by Real et al. (2019), described above. Unlike Real et al. (2019) who conducted their search on CIFAR-10, our search is conducted on a task that takes much longer to train and evaluate on. Specifically, to train a Transformer to peak performance on WMT'14 En-De requires $\sim$300K training steps, or 10 hours, in the base size when using a single Google TPU V.2 chip, as we do in our search. In contrast, Real et al. (2019) used the less resource-intensive CIFAR-10 task (Krizhevsky & Hinton, 2009), which takes about two hours to train on, to assess their models during their search, as it was a good proxy for ImageNet (Deng et al., 2009) performance (Zoph et al., 2018). However, in our preliminary experimentation we could not find a proxy task that gave adequate signal for how well each child model would perform on the full WMT'14 En-De task; we investigated using only a fraction of the data set and various forms of aggressive early stopping.

To address this problem we formulated a method to dynamically allocate resources to more promising architectures according to their fitness. This method, which we refer to as *Progressive Dynamic Hurdles* (PDH), allows models that are consistently performing well to train for more steps. It begins as ordinary tournament selection evolutionary architecture search with early stopping, with each child model training for a relatively small s_0 number of steps before being evaluated for fitness. However, after a predetermined number of child models, m, have been evaluated, a *hurdle*, h_0, is created by calculating the the mean fitness of the current population. For the next m child models produced, models that achieve a fitness greater than h_0 after s_0 train steps are granted an additional s_1 steps of training and then are evaluated again to determine their final fitness. Once another m models have been considered this way, another hurdle, h_1, is constructed by calculating the mean fitness of all members of the current population that were trained

for the maximum number of steps. For the next m child models, training and evaluation continues in the same fashion, except models with fitness greater than h_1 after $s_0 + s_1$ steps of training are granted an additional s_2 number of train steps, before being evaluated for their final fitness. This process is repeated until a satisfactory number of maximum training steps is reached. Algorithm 1 (Supplementary Materials) formalizes how the fitness of an individual model is calculated with hurdles and Algorithm 2 (Supplementary Materials) describes tournament selection augmented with Progressive Dynamic Hurdles.

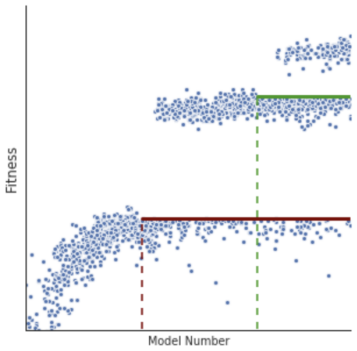

Figure 2. **Evolution architecture search with hurdles.** The y-axis represents architecture fitness and the x-axis represents the order in which candidate models were created. The solid purple and green lines represent the values of the first and second hurdles, respectively. The dashed purple and green lines represent the points at which each of the corresponding hurdles were introduced. Points to the left of the purple dashed line were generated using unaltered tournament selection. Between the purple and green dashed lines, models with a fitness above the solid purple line were granted additional train steps, forming a higher fitness cluster. To the right of the green dashed line, models with a fitness greater than the solid green line were granted a second round of additional train steps.

Although different child models may train for different numbers of steps before being assigned their final fitness, this does not make their fitnesses incomparable. Tournament selection evolution is only concerned with relative fitness rank when selecting which subpopulation members will be killed and which will become parents; the margin by which one candidate is better or worse than the other members of the subpopulation does not matter. Assuming no model overfits during its training, which is what we observed in our experiments, and that its fitness monotonically increases with respect to the number of train steps it is allocated, a comparison between two child models can be viewed as a comparison between their fitnesses at the lower of the two's cumulative train steps. Since the model that was allocated more train steps performed, by definition, above the fitness hurdle for the lower number of steps and the model that was allocated less steps performed, by definition, at or below that hurdle at the lower number of steps, it is guaranteed that the model with more train steps was better when it was evaluated at the lower number of train steps.

The benefit of altering the fitness algorithm this way is that poor performing child models will not consume as many resources when their fitness is being computed. As soon as a candidate's fitness falls below a tolerable amount, its evaluation immediately ends. This may also result in good candidates being labeled as bad models if they are only strong towards the latter part of training. However, the resources saved as a result of discarding many bad models improves the overall quality of the search enough to justify potentially also discarding some good ones; this is supported empirically in our Results section.

4. Experiment Setup

4.1. Datasets

Machine Translation We use three different machine translation datasets to perform our experiments, all of which were taken from their Tensor2Tensor implementations[1]. The first is WMT English-German, for which we mimic Vaswani et al.'s (2017) setup, using WMT'18 En-De training data without ParaCrawl (ParaCrawl, 2018), yielding 4.5 million sentence pairs. In the same fashion, we use newstest2013 for development and test on newstest2014. The second translation dataset is WMT En-Fr, for which we also replicate Vaswani et.al's (2017) setup. We train on the 36 million sentence pairs of WMT'14 En-Fr, validate on newstest2013 and test on newstest2014. The final translation dataset is WMT English-Czech (En-Cs). We used the WMT'18 training dataset, again without ParaCrawl, and used newstest2013 and newstest2014 as validation and test sets. For all tasks, tokens were split using a shared source-target vocabulary of about 32K word-pieces (Wu et al., 2016).

All datasets were generated using Tensor2Tensor's "packed" scheme; sentences were shuffled and concatenated together with padding to form uniform 256 length inputs and targets, with examples longer than 256 being discarded. This yielded batch sizes of 4096 tokens per GPU or TPU chip; accordingly, 16 TPU chip configurations had $\sim$66K tokens per batch and 8 GPU chip configurations had $\sim$33K tokens per batch.

Language Modeling For language modeling we used the 1 Billion Word Language Model Benchmark (LM1B) (Chelba et al., 2013), also using its "packed" Tensor2Tensor implementation. Again the tokens are split into a vocabulary of approximately 32K word-pieces and the sentences are shuffled.

[1]https://github.com/tensorflow/tensor2tensor/tree/master/tensor2tensor/data_generators

4.2. Training Details and Hyperparameters

Machine Translation All of our experiments used Tensor2Tensor's Transformer TPU hyperparameter settings[2]. These are nearly identical to those used by Vaswani et al. (2017), but modified to use the memory-efficient Adafactor (Shazeer & Stern, 2018) optimizer. Aside from using the optimizer itself, these hyperparameters also set the warmup to a constant learning rate of 10^{-2} over 10K steps and then uses inverse-square-root learning-rate decay. For our experiments, we make only one change, which is to alter this decay so that it reaches 0 at the final step of training, which for our non-search experiments is uniformly 300K. We found that the our search candidate models, the Transformer, and the Evolved Transformer all benefited from this and so experimented with using linear decay, single-cycle cosine decay (Loshchilov & Hutter, 2017) and a modified inverse-square-root decay to 0 at 300K steps: $lr = step^{-0.00303926} - .962392$; every decay was paired with the same constant 10^{-2} warmup. We used WMT En-De validation perplexity to gauge model performance and found that the Transformer preferred the modified inverse-square-root decay. Therefore, this is what we used for both all our Transformer trainings and the architecture searches themselves. The Evolved Transformer performed best with cosine decay and so that is what we used for all of its trainings. Besides this one difference, the hyperparameter settings across models being compared are exactly the same. Because decaying to 0 resulted in only marginal weight changes towards the end of training, we did not use checkpoint averaging, except where noted.

Per-task there is one additional hyperparameter difference, which is dropout rate. For ET and all search child models, dropout was applied uniformly after each layer, approximating the Transformer's more nuanced dropout scheme. For En-De and En-Cs, all "big" and "deep" sized models were given a higher dropout rate of 0.3, keeping in line with Vaswani et al. (2017), and all other models with an input embedding size of 768 are given a dropout rate of 0.2. Aside from this, hyperparameters are identical across all translation tasks.

For decoding we used the same beam decoding configuration used by Vaswani et al. (2017). That is a *beam size* of 4, *length penalty* (α) of 0.6, and *maximum output length* of input length + 50. All BLEU is calculated using case-sensitive tokenization[3] and for WMT'14 En-De we also use the compound splitting that was used in Vaswani et al. (2017).

[2]https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/models/transformer.py

[3]https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

Language Modeling Our language model training setup is identical to our machine translation setup except we remove label smoothing and lower the intra-attention dropout rate to 0. This was taken from the Tensor2Tensor hyperparameters for LM1B[2].

4.3. Search Configurations

All of the architecture searches we describe were run on WMT'14 En-De. They utilized the search space and tournament selection evolution algorithm described in our Methods section. Unless otherwise noted, each search used 200 workers, which were equipped with a single Google TPU V.2 chip for training and evaluation. We maintained a population of size 100 with subpopulation sizes for both killing and reproducing set to 30. Mutations were applied independently per encoding field at a rate of 2.5%. For fitness we used the negative log perplexity of the validation set instead of BLEU because, as demonstrated in our Results section, perplexity is more consistent and that reduced the noise of our fitness signal.

5. Results

In this section, we will first benchmark the performance of our search method, Progressive Dynamic Hurdles, against other evolutionary search methods (Real et al., 2017; 2019). We will then benchmark the Evolved Transformer, the result of our search method, against the Transformer (Vaswani et al., 2017).

5.1. Ablation Study of Search Techniques

We tested our evolution algorithm enhancements – using PDH and warm starting by seeding the initial population with the Transformer – against control searches that did not use these techniques; without our enhancements, these controls function the same way as Real et. al's (2019) searches, without aging regularization. Each search we describe was run 3 times and the top model from each run was retrained on a single TPU V.2 chip for 300K steps. The performance of the models after retraining is given in Table 1.

SEED MODEL	TRAIN STEPS	NUM MODELS	TOP MODEL PERPLEXITY
TRANSFORMER	PDH	6000	**4.50** $\pm$ 0.01
RANDOM	PDH	6000	5.23 $\pm$ 0.19
TRANSFORMER	15K	29714	4.57 $\pm$ 0.01
TRANSFORMER	30K	14857	4.53 $\pm$ 0.07
TRANSFORMER	180K	2477	4.58 $\pm$ 0.05
TRANSFORMER	300K	1486	4.61 $\pm$ 0.02

Table 1. **Top model validation perplexity of various search setups.** Number of models were chosen to equalize resource consumption.

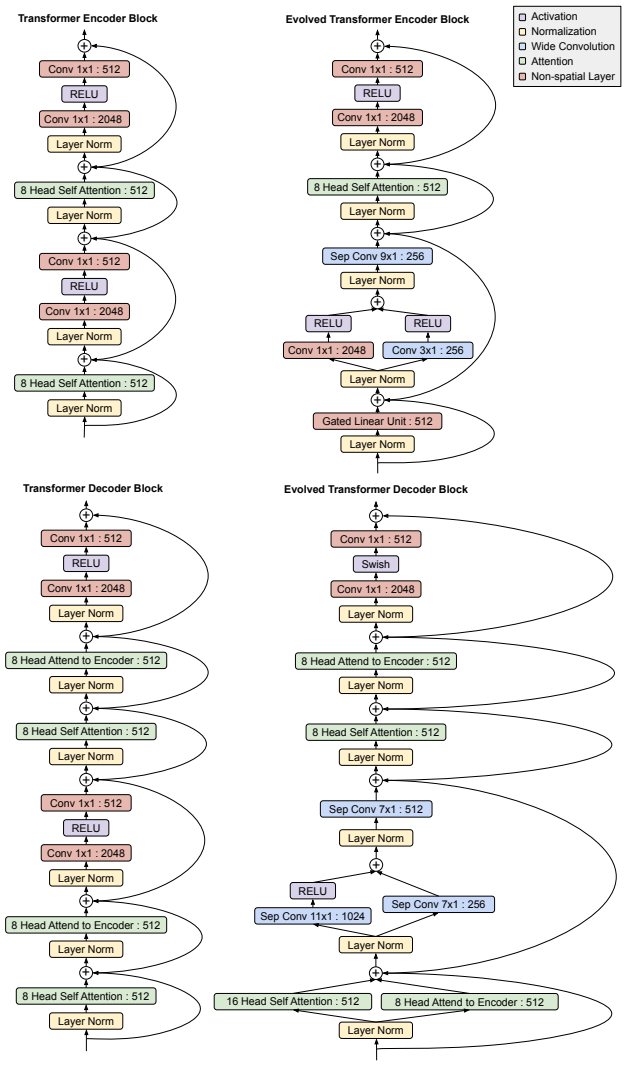

Figure 3. **Transformer and Evolved Transformer architecture cells.** The four most notable aspects of the found architecture are the use of 1) wide depth-wise separable convolutions, 2) Gated Linear Units (Dauphin et al., 2017), 3) branching structures and 4) swish activations (Ramachandran et al., 2017). Both the ET encoder and decoder independently developed a branched lower portion of wide convolutions. Also in both cases, the latter portion is almost identical to the Transformer.

Our proposed search (Table 1 row 1), which used both PDH and Transformer seeding, was run first, with hurdles created every 1K models ($m = 1000$) and six 30K train step (1 hour) increments ($s =< 30, 30, 30, 30, 30, 30 >$). To test the effectiveness of seeding with the Transformer, we ran an identical search that was instead seeded with random valid encodings (Table 1 row 2). To test the effectiveness of PDH, we ran four controls (Table 1 rows 3-6) that each used a fixed number of train steps for each child model instead of hurdles (Table 1 column 2). For these we used the step increments (30K), the maximum number of steps our proposed search ultimately reaches (180K), the total number of steps each top model receives when fully trained to gauge its final performance (300K), and half the step increments (15K), recognizing the gains from evaluating a larger number of models in the 30K steps control case. To determine the number of child models each of these searches would be able to train, we selected the value that would make the total amount of resources used by each control search equal to the maximum amount of resources used for our proposed searches, which require various amounts of resources depending on how many models fail to overcome hurdles. In the three trials we ran, our proposed search's total number of train steps used was 422M $\pm$ 21M, with a maximum of 446M. Thus the number of child models allotted for each non-PDH control search was set so that the total number of child model train steps used would be 446M.

As demonstrated in Table 1, the search we propose, with PDH and Transformer seeding, has the best performance on average. It also is the most consistent, having the lowest standard deviation. Of all the searches conducted, only a single control run – "30K no hurdles" (Table 1 row 3) – produced a model that was better than any of our proposed search's best models. At the same time, the "30K no hurdles" setup also produced models that were significantly worse, which explains its high standard deviation. This phenomenon was a chief motivator for our developing this method. Although aggressive early stopping has the potential to produce strong models for cheap, searches that utilize it can also venture into modalities in which top fitness child models are only strong early on; without running models for longer, whether or not this is happening cannot be detected. For example, the 15K search performed worse than the 30K setting, despite evaluating twice as many models. Although the 180K and 300K searches did have insight into long term performance, it was in a resource-inefficient manner that hurt these searches by limiting the number of generations they produced; for the "180K no hurdles" run to train as many models as PDH would require 1.08B train steps, over double what PDH used in our worst case.

Searching with random seeding also proved to be ineffective, performing considerably worse than every other configuration. Of the five searches run, random seeding was the only one that had a top model perplexity higher than the Transformer, which is 4.75 $\pm$ 0.01 in the same setup.

5.2. Main Search.

After confirming the effectiveness of our search procedure, we launched a larger scale version of our search using 270 workers. We trained 5K models per hurdle ($m = 5000$) and used larger step increments to get a closer approximation to 300K step performance: $s =< 60, 60, 120 >$. The setup was the same as the Search Techniques experiments, except after 11K models we lowered the mutation rate to 0.01 and

introduced the NONE value to the normalization mutation vocabulary.

The search ran for 15K child models, requiring a total of 979M train steps. Over 13K models did not make it past the first hurdle, drastically reducing the resources required to view the 240 thousandth train step for top models, which would have cost 3.6B train steps for the same number of models without hurdles. After the search concluded, we then selected the top 20 models and trained them for the full 300K steps, each on a single TPU V.2 chip. The model that ended with the best perplexity is what we refer to as the Evolved Transformer (ET). Figure 3 shows the ET architecture. The most notable aspect of the Evolved Transformer is the use of wide depth-wise separable convolutions in the lower layers of the encoder and decoder blocks. The use of depth-wise convolution and self-attention was previously explored in QANet (Yu et al., 2018), however the overall architectures of the Evolved Transformer and QANet are different in many ways: e.g., QANet has smaller kernel sizes and no branching structures. The performance and analysis of the Evolved Transformer will be shown in the next section.

5.3. The Evolved Transformer: Performance and Analysis

To test the effectiveness of the found architecture – the Evolved Transformer – we compared it to the Transformer in its Tensor2Tensor training regime on WMT'14 En-De. Table 3 shows the results of these experiments run on the same 8 NVIDIA P100 hardware setup that was used by Vaswani et al. (2017). Observing ET's improved performance at parameter-comparable "base" and "big" sizes, we were also interested in understanding how small ET could be shrunk while still achieving the same performance as the Transformer. To create a spectrum of model sizes for each architecture, we selected different input embedding sizes and shrank or grew the rest of the model embedding sizes with the same proportions. Aside from embedding depths, these models are identical at all sizes, except the "big" 1024 input embedding size, for which all 8 head attention layers are upgraded to 16 head attention layers, as was done in Vaswani et al. (2017).

ET demonstrates stronger performance than the Transformer at all sizes, with the largest difference of 0.7 BLEU at the smallest, mobile-friendly, size of ∼7M parameters. Performance on par with the "base" Transformer was reached when ET used just 78.4% of its parameters and performance of the "big" Transformer was exceeded by the ET model that used 37.6% less parameters. Figure 4 shows the FLOPS vs. BLEU performance of both architectures.

To test if ET's strong performance generalizes, we also compared it to the Transformer on an additional three well-established language tasks: WMT'14 En-Fr, WMT'14 En-Cs, and LM1B.[4] Upgrading to 16 TPU V.2 chips, we doubled the number of synchronous workers for these experiments, pushing both models to their higher potential (Ott et al., 2018). We ran each configuration 3 times, except WMT En-De, which we ran 6 times; this was a matter of resource availability and we gave priority to the task we searched on. As shown in Table 2, ET performs at least one standard deviation above the Transformer in each of these tasks. Note that the Transformer mean BLEU scores in all of our experiments for WMT'14 En-Fr and WMT'14 En-De are higher than those originally reported by Vaswani et al. (2017).

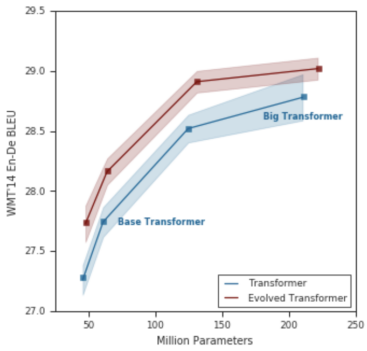

Figure 4. **Performance comparison of the Evolved Transformer against the Transformer across number of parameters.**

As can be seen in Tables 3 and 2, the Evolved Transformer is much more effective than the Transformer at smaller model sizes. At the "big" model size, its BLEU performance saturates and the gap between the Evolved Transformer and the Transformer becomes smaller. One explanation for this behavior is that overfitting starts to occur at big model sizes, but we expect that data augmentation (Ott et al., 2018) or hyperparameter tuning could improve performance. For example, we found that simply increasing the embedding size was not the best way to grow ET from the "base" size we searched over to a larger size. Depth should also be tuned in conjunction with embedding size, when controlling for number of parameters. For both the Transformer and ET, we tried four additional embedding sizes, [512, 640, 768, 896], adjusting the depth accordingly so that all resulting models had a similar number of parameters. Using validation BLEU to determine the best configuration, we found that ET performed best with an embedding depth of 640, increasing its number of encoder cells from 3 to 9 and its number of decoder cells from 4 to 10. The Transformer also benefited from additional depth, although not to the same degree, achieving maximum performance at the 768 embedding size, with 6 encoder cells and 6 decoder cells. These results are included in Table 2, labeled as "Deep" size.

[4]For LM1B, we only use the decoder architecture, with *attend to encoder* layers removed.

TASK	SIZE	TRAN PARAMS	ET PARAMS	TRAN PERP	ET PERP	TRAN BLEU	ET BLEU
WMT'14 EN-DE	BASE	61.1M	64.1M	4.24 ± 0.03	**4.03** ± 0.02	28.2 ± 0.2	**28.4** ± 0.2
WMT'14 EN-DE	BIG	210.4M	221.7M	3.87 ± 0.02	**3.77** ± 0.02	29.1 ± 0.1	**29.3** ± 0.1
WMT'14 EN-DE	DEEP	224.0M	218.1M	3.86 ± 0.02	**3.69** ± 0.01	29.2 ± 0.1	**29.5** ± 0.1
WMT'14 EN-FR	BASE	60.8	63.8M	3.61 ± 0.01	**3.42** ± 0.01	40.0 ± 0.1	**40.6** ± 0.1
WMT'14 EN-FR	BIG	209.8M	221.2M	3.26 ± 0.01	**3.13** ± 0.01	41.2 ± 0.1	**41.3** ± 0.1
WMT'14 EN-CS	BASE	59.8M	62.7M	4.98 ± 0.04	**4.42** ± 0.01	27.0 ± 0.1	**27.6** ± 0.2
WMT'14 EN-CS	BIG	207.6M	218.9M	4.43 ± 0.01	**4.38** ± 0.03	28.1 ± 0.1	**28.2** ± 0.1
LM1B	BIG	141.1M	151.8M	30.44 ± 0.04	**28.60** ± 0.03	-	-

Table 2. **Comparison between the Transformer and ET trained on 16 TPU V.2 chips.** For Translation, perplexity was calculated on the validation set and BLEU was calculated on the test set. For LM1B, perplexity was calculated on the test set. ET shows consistent improvement by at least one standard deviation on all tasks. It excels at the base size the search was conducted in, with an improvement of 0.6 BLEU in both En-Fr and En-Cs.

Model	Embedding Size	Parameters	Perplexity	BLEU	Δ BLEU
Transformer	128	7.0M	8.62 ± 0.03	21.3 ± 0.1	-
ET	128	7.2M	**7.62** ± 0.02	**22.0** ± 0.1	**+ 0.7**
Transformer	432	45.8M	4.65 ± 0.01	27.3 ± 0.1	-
ET	432	47.9M	**4.36** ± 0.01	**27.7** ± 0.1	+ 0.4
Transformer	512	61.1M	4.46 ± 0.01	27.7 ± 0.1	-
ET	512	64.1M	**4.22** ± 0.01	**28.2** ± 0.1	+ 0.5
Transformer	768	124.8M	4.18 ± 0.01	28.5 ± 0.1	-
ET	768	131.2M	**4.00** ± 0.01	**28.9** ± 0.1	+ 0.4
Transformer	1024	210.4M	4.05 ± 0.01	28.8 ± 0.2	-
ET	1024	221.7M	**3.94** ± 0.01	**29.0** ± 0.1	+ 0.2

Table 3. **WMT'14 En-De comparison on 8 NVIDIA P100 GPUs.** Each model was trained 10 to 15 times, depending on resource availability. Perplexity is calculated on the validation set and BLEU is calculated on the test set.

Model	Params	BLEU	SacreBLEU (Post, 2018)
Gehring et al. (2017)	216M	25.2	-
Vaswani et al. (2017)	213M	28.4	-
Ahmed et al. (2017)	213M	28.9	-
Chen et al. (2018)	379M	28.5	-
Shaw et al. (2018)	213M	29.2	-
Ott et al. (2018)	210M	29.3	28.6
Wu et al. (2019)	213M	29.7	-
Evolved Transformer	218M	**29.8**	**29.2**

Table 4. **Model comparison on WMT'14 En-De.**

To compare with other previous results, we trained the ET Deep model three times in our TPU setup on WMT'14 En-De, selected the best run according to validation BLEU and did a single decoding on the test set. We also copied previous state-of-the-art result setups by averaging the last 20 model checkpoints from training and decoding with a beam width of 5 (Vaswani et al., 2017; Ott et al., 2018; Wu et al., 2019). As a result, the Evolved Transformer achieved a new state-of-the-art BLEU score of 29.8 (Table 4).

6. Conclusion

We presented the first neural architecture search conducted to find improved feed-forward sequence models. We first constructed a large search space inspired by recent advances in seq2seq models and used it to search directly on the computationally intensive WMT En-De translation task. To mitigate the size of our space and the cost of training child models, we proposed using both our Progressive Dynamic Hurdles method and warm starting, seeding our initial population with a known strong model, the Transformer.

When run at scale, our search found the Evolved Transformer. In a side by side comparison against the Transformer in an identical training regime, the Evolved Transformer showed consistent stronger performance on both translation and language modeling. On the task we searched over, WMT'14 En-De, the Evolved Transformer established a new state-of-the-art of 29.8 BLEU. It also proved to be efficient at smaller sizes, achieving the same quality as the original "big" Transformer with 37.6% less parameters and outperforming the Transformer by 0.7 BLEU at a mobile-friendly model size of $\sim$7M parameters.

Acknowledgements

We would like to thank Ashish Vaswani, Jakob Uszkoreit, Niki Parmar, Noam Shazeer, Lukasz Kaiser and Ryan Sepassi for their help with Tensor2Tensor and for sharing their understanding of the Transformer. We are also grateful to David Dohan, Esteban Real, Yanping Huang, Alok Aggarwal, Vijay Vasudevan, and Chris Ying for lending their expertise in architecture search and evolution.

References

Ahmed, K., Keskar, N. S., and Socher, R. Weighted transformer network for machine translation. *CoRR*, abs/1711.02132, 2017. URL http://arxiv.org/abs/1711.02132.

Ba, L. J., Kiros, R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2016.

Brock, A., Lim, T., Ritchie, J., and Weston, N. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rydeCEhs-.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., and Koehn, P. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013. URL http://arxiv.org/abs/1312.3005.

Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Parmar, N., Schuster, M., Chen, Z., Wu, Y., and Hughes, M. The best of both worlds: Combining recent advances in neural machine translation. *CoRR*, abs/1804.09849, 2018. URL http://arxiv.org/abs/1804.09849.

Cho, K., Merrienboer, B. V., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoderdecoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

Dai, A. M. and Le, Q. V. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pp. 3079–3087, 2015.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. In *International Conference on Machine Learning*, 2017.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. volume abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 2018.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, 2017.

Goldberg, D. E. and Deb, K. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, 1991.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. In *Neural Computation*, pp. 1735–1780. Massachusetts Institute of Technology, 1997.

Jamieson, K. G. and Talwalkar, A. S. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*, 2016.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. 2009.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. S. Hyperband: a novel bandit-based approach to hyperparameter optimization. In *Journal of Machine Learning Research*, 2017.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A. L., Huang, J., and Murphy, K. Progressive neural architecture search. In *European Conference on Computer Vision*, 2018a.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *DARTS: differentiable architecture search*, 2018b.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.

Ott, M., Edunov, S., Grangier, D., and Auli, M. Scaling neural machine translation. In *Workshop on Machine Translation, Empirical Methods in Natural Language Processing*, 2018.

ParaCrawl, 2018. URL http://paracrawl.eu/download.html.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, 2018.

Post, M. A call for clarity in reporting BLEU scores. *CoRR*, abs/1804.08771, 2018. URL http://arxiv.org/abs/1804.08771.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training, 2018. URL https://blog.openai.com/language-unsupervised/.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q., and Kurakin, A. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 2017.

Real, E., Aggarawal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *International Conference on Learning Representations*, 2019.

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. volume abs/1803.02155, 2018. URL http://arxiv.org/abs/1803.02155.

Shazeer, N. and Stern, M. Adafactor: adaptive learning rates with sublinear memory cost. *CoRR*, abs/1804.04235, 2018. URL http://arxiv.org/abs/1804.04235.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 31043112, 2014.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Neural Information Processing Systems*, 2017.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A. N., Gouws, S., Jones, L., ukasz Kaiser, Kalchbrenner, N., Parmar, N., Sepassi, R., Shazeer, N., and Uszkoreit, J. Tensor2tensor for neural machine translation. In *Computing Research Repository*, 2018. abs/1803.07416.

Wu, F., Fan, A., Baevski, A., Dauphin, Y., and Auli, M. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*, 2019.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G. S., Hughes, M., and Dean, J. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

Xie, L. and Yuille, A. L. Genetic cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1388–1397, 2017.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

Yu, A. W., Dohan, D., Luong, T., Zhao, R., Chen, K., Norouzi, M., and Le, Q. V. Fast and accurate reading comprehension by combining self-attention and convolution. In *International Conference on Learning Representations*, 2018.

Zela, A., Klein, A., Falkner, S., and Hutter, F. Towards automated deep learning: efficient joint neural architecture and hyperparameter search. In *Workshop on AutoML, International Conference on Machine Learning*, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2018.

A. Search Algorithms

In the following, we describe the algorithm that we use to calculate child model fitness with hurdles (Algorithm 1) and evolution architecture search with Progressive Dynamic Hurdles (Algorithm 2).

Algorithm 1 Calculate Model Fitness with Hurdles

inputs:
$model$: the child model
s: vector of train step increments
h: queue of hurdles

append ∞ to h
TRAIN_N_STEPS($model$, s_0)
$fitness \leftarrow$ EVALUATE($model$)
$i \leftarrow 0$
$hurdle \leftarrow h_i$

while $fitness > hurdle$ **do**
  $i \leftarrow i + 1$
  TRAIN_N_STEPS($model$, s_i)
  $fitness \leftarrow$ EVALUATE($model$)
  $hurdle \leftarrow h_i$
end while
return $fitness$

Algorithm 1 Calculating fitness with hurdles takes as arguments a child model, a vector of train step increments (s) and a queue of hurdles(h). The child model is the candidate model in our neural architecture search. The vector of step increments describes the number of steps between each hurdle; its length must be greater than 0. The queue of hurdles describes what hurdles have already been established; its length must be in [0, $length(s)$).

The algorithm starts by first training the child model a fixed number of s_0 steps and evaluating on the validation set to produce a fitness, as is done in Real et al. (2019). After this baseline fitness is established, the hurdles (h) are compared against to determine if training should continue. Each h_i denotes the fitness a child model must have after $\sum_{j=0}^{i} s_j$ train steps to continue training. Each time a hurdle h_i is passed, the model is trained an additional s_{i+1} steps. If the model's fitness ever falls below the hurdle corresponding to the number of steps it was trained for, training ends immediately and the current fitness is returned. If the model never falls below a hurdle and all hurdles have been passed, the child model receives one final training of $s_{length(h)}$ steps before fitness is returned; this is expressed in Algorithm 1 with ∞ being appended to the end of h.

Algorithm 2 Evolution Architecture Search with PDH

inputs:
s: vector of train step increments
m: number of child models per hurdle

$h \leftarrow$ *empty queue*
$i \leftarrow 0$
$population \leftarrow$ INITIAL_POPULATION()

while $i <$ LENGTH(s) - 1 **do**
  $population \leftarrow$ EVOL_N_MODELS($population$, m, s, h)
  $hurdle \leftarrow$ MEAN_FITNESS_OF_MAX($population$)
  append $hurdle$ to h
end while

$population \leftarrow$ EVOL_N_MODELS($population$, m, s, h)
return $population$

Algorithm 2 Evolution architecture search with PDH takes as arguments a vector of train step increments (s) and a number of child models per hurdle (m). It begins as Real et al.'s (2019) evolution architecture search with a fixed number of child model train steps, s_0. However, after m child models have been produced, a hurdle is created by taking the mean fitness of the current population and it is added to the hurdle queue, h. Algorithm 1 is used to compute each child model's fitness and so if they overcome the new hurdle they will receive more train steps. This process is continued, with new hurdles being created using the mean fitness of all models that have trained the maximum number of steps and h growing accordingly. The process terminates when $length(s) - 1$ hurdles have been created and evolution is run for one last round of m models, using all created hurdles.

B. Search Space Information

In our search space, a child model's genetic encoding is expressed as: [*left input*, *left normalization*, *left layer*, *left relative output dimension*, *left activation*, *right input*, *right normalization*, *right layer*, *right relative output dimension*, *right activation*, *combiner function*] $\times$ 14 + [*number of cells*] $\times$ 2, with the first 6 blocks allocated to the encoder and the latter 8 allocated to the decoder. In the following, we will describe each of the components.

Input. The first branch-level search field is *input*. This specifies what hidden state in the cell will be fed as input to the branch. For each i^{th} block, the input vocabulary of its branches is $[0, i)$, where the j^{th} hidden state corresponds to the j^{th} block output and the 0^{th} hidden state is the cell

input.

Normalization. The second branch-level search field is *normalization*, which is applied to each input before the layer transformation is applied. The normalization vocabulary is [LAYER NORMALIZATION (Ba et al., 2016), NONE].

Layers. The third branch-level search field is *layer*, which is the neural network layer applied after the normalization. It's vocabulary is:

- STANDARD CONV wx1: for $w \in \{1, 3\}$
- DEPTHWISE SEPARABLE CONV wx1: for $w \in \{3, 5, 7, 9, 11\}$
- LIGHTWEIGHT CONV wx1 r: for $w \in \{3, 5, 7, 15\}$, $r \in \{1, 4, 16\}$ (Wu et al., 2019). r is the reduction factor, equivalent to d/H described in Wu et al. (2019).
- h HEAD ATTENTION: for $h \in \{4, 8, 16\}$
- GATED LINEAR UNIT(Dauphin et al., 2017)
- ATTEND TO ENCODER: (Only available to decoder)
- IDENTITY: No transformation applied to input
- DEAD BRANCH: No output

For decoder convolution layers the inputs are shifted by $(w - 1)/2$ so that positions cannot "see" later predictions.

Relative Output Dimension. The fourth branch-level search field is *relative output dimension*, which describes the output dimension of the corresponding layer. The Transformer is composed mostly of layers that project to the original input embedding depth (512 in the "base" configuration), but also contains 1x1 convolutions that project up to a dimension of 4 times that depth (2048 in the "base" configuration). The relative output dimension search field accounts for this variable output depth. It's vocabulary consists of 10 relative output size options: $[1, 10]$.

Here "relative" refers to the fact that for every layer i and j, each of their absolute output dimensions, a, and relative output dimensions, d, will obey the ratio: $a_i/a_j = d_i/d_j$. We determine the absolute output dimensions for each model by finding a scaling factor, s, such that for every layer i, $a_i = d_i * s$ and the resulting model has an appropriate number of parameters; at the end of this section, we describe our constraints on number of model parameters. There may be multiple values of s for any one model that satisfy this constraint, and so for our experiments we simply perform a binary search and use the first valid value found. If no valid value is found, we reject the child model encoding as invalid and produce a new one in its stead.

We chose a vocabulary of relative sizes instead of absolute sizes because we only allow models within a fixed parameter range, as described later in this section (Constraints). Using relative sizes allows us to increase the number of configurations that represent valid models in our search space, because we can dynamically shrink or grow a model to make it fit within the parameter bounds. We found that using absolute values, such as $[256, 512, 1024, 2048]$, increases the number of rejected models and thereby decreases the possible models that can be expressed.

This relative output dimensions field is ignored for both the IDENTITY and DEAD BRANCH layers.

Activations. The final branch-level search field is *activation*, which is the non-linearity applied on each branch after the neural network layer. The activation vocabulary is {SWISH (Ramachandran et al., 2017; Elfwing et al., 2018), RELU, LEAKY_RELU (MAAS ET AL., 2013), NONE}.

Combiner Functions. The block-level search field, *combiner function*, describes how the left and right layer branches are combined together. Its vocabulary is {ADDITION, CONCATENATION, MULTIPLICATION}. For MULTIPLICATION and ADDITION, if the right and left branch outputs have differing embedding depths, then the smaller of the two is padded so that the dimensionality matches. For ADDITION the padding is 0's; for MULTIPLICATION the padding is 1's.

Number of Cells. The cell-level search field is *number of cells* and it describes the number of times the cell is repeated. Its vocabulary is [1,6].

Composition. Each child model is defined by two cells, one for the encoder and one for the decoder. The encoder cell contains 6 blocks and the decoder cell contains 8 blocks. Each block contains two branches, each of which takes a previous hidden layer as input, and then applies its normalization, layer (with specified relative output dimensions) and activation to it. The two branches are then joined with the combiner function. Any unused hidden states are automatically added to the final block output via addition. Both the encoder and decoder cells defined this way are repeated their corresponding *number of cells* times and connected to the input and output embedding portions of the network to produce the final model; we use the same embedding scheme described by Vaswani et al. (2017) for all models. See Figure 1 for a depiction of this composition.

Constraints. In the interest of having a fair comparison across child models, we limit our search to only architectures configurations that can contain between 59.1 million and 64.1 million parameters when their relative output dimensions are scaled; in the Tensor2Tensor (Vaswani et al., 2018) implementation we use, the base Transformer has roughly 61.1 million parameters on WMT En-De, so our models are allowed 3 million less or more parameters than that. Models that cannot be represented within this parame-

ter range are not included in our search space.

Additionally, in preliminary experiment runs testing the effectiveness of our search space, we discovered three trends that hurt performance in almost every case. Firstly and most obviously is when a proposed decoder contains no ATTEND TO ENCODER layers. This results in the decoder receiving no signal from the encoder and thus the model output will not be conditioned on the input. Therefore, any model that does not contain ATTEND TO ENCODER is not in our search space. The second trend that we noticed was that models that had layer normalization removed were largely worse than their parent models. For this reason, we remove NONE from the *normalization* mutation vocabulary for each experiment, unless otherwise specified. Lastly, we observed that an important feature of good models was containing an unbroken residual path from inputs to outputs; in our search space, this means a path of IDENTITY layers from cell input to output that are combined with ADDITION at every *combination function* along the way. Our final constraint is therefore that models that do not have unbroken residual paths from cell inputs to outputs are not in our search space.

C. Ablation Study of the Evolved Transformer

To understand what mutations contributed to ET's improved performance we conducted two rounds of ablation testing. In the first round, we began with the Transformer and applied each mutation to it individually to measure the performance change each mutation introduces in isolation. In the second round, we began with ET and removed each mutation individually to again measure the impact of each single mutation. In both cases, each model was trained 3 times on WMT En-De for 300k steps with identical hyperparameters, using the inverse-square-root decay to 0 that the Transformer prefers. Each training was conducted on a single TPU V.2 chip. The results of these experiments are presented in Table 5; we use validation perplexity for comparison because it was our fitness metric.

In all cases, the augmented ET models outperformed the the augmented Transformer models, indicating that the gap in performance between ET and the Transformer cannot be attributed to any single mutation. The mutation with the seemingly strongest individual impact is the increase from 3 to 4 decoder cells. However, even when this mutation is introduced to the Transformer and removed from ET, the resulting augmented ET model still has a higher fitness than the augmented Transformer model.

To highlight the impact of each augmented model's mutation, we present not only their perplexities but also the difference between their mean perplexity and their unaugmented base model mean perplexity in the "Mean Diff" columns:

base model mean perplexity - augmented mean perplexity

This delta estimates the change in performance each mutation creates in isolation. Red highlighted cells contain evidence that their corresponding mutation hurt overall performance. Green highlighted cells contain evidence that their corresponding mutation helped overall performance.

In half of the cases, both the augmented Transformer's and the augmented Evolved Transformer's performances indicate that the mutation was helpful. Changing the number of attention heads from 8 to 16 was doubly indicated to be neutral and changing from 8 head self attention to a GLU layer in the decoder was doubly indicated to have hurt performance. However, this and other mutations that seemingly hurt performance may have been necessary given how we formulate the problem: finding an improved model with a comparable number of parameters to the Transformer. For example, when the Transformer decoder cell is repeated 4 times, the resulting model has 69.6M parameters, which is outside of our allowed parameter range. Thus, mutations that shrank ET's total number of parameters, even at a slight degradation of performance, were necessary so that other more impactful parameter-expensive mutations, such as adding an additional decoder cell, could be used.

Other mutations have inconsistent evidence about how useful they are. This ablation study serves only to approximate what is useful, but how effective a mutation is also depends on the model it is being introduced to and how it interacts with other encoding field values.

Mutation Field	Mutation Block Index	Mutation Branch	Transformer Value	ET Value	Transformer Perplexity	ET Perplexity	Transformer Mean Diff	ET Mean Diff
DECODER ACTIVATION	6	LEFT	RELU	SWISH	4.73 ± 0.01	4.51 ± 0.02	-0.02	0.04
DECODER ACTIVATION	2	RIGHT	RELU	NONE	4.73 ± 0.01	4.48 ± 0.00	-0.02	0.02
DECODER INPUT	1	LEFT	1	0	4.74 ± 0.04	4.46 ± 0.00	-0.01	-0.01
DECODER LAYER	0	LEFT	8 HEAD ATTENTION	16 HEAD ATTENTION	4.75 ± 0.01	4.47 ± 0.01	0.0	0.0
DECODER LAYER	2	LEFT	STANDARD CONV 1X1	SEPARABLE CONV 11X1	4.67 ± 0.01	4.55 ± 0.00	-0.08	0.09
DECODER LAYER	3	LEFT	STANDARD CONV 1X1	SEPARABLE CONV 7X1	4.72 ± 0.01	4.46 ± 0.01	-0.03	0.0
DECODER LAYER	2	RIGHT	DEAD BRANCH	SEPARABLE CONV 7X1	4.71 ± 0.02	4.47 ± 0.00	-0.04	0.01
DECODER NORM	3	LEFT	NONE	LAYER NORM	4.73 ± 0.00	4.45 ± 0.01	-0.02	-0.01
DECODER NORM	7	LEFT	NONE	LAYER NORM	4.73 ± 0.02	4.47 ± 0.02	-0.02	0.01
DECODER OUTPUT DIM	2	LEFT	8	4	4.74 ± 0.01	4.45 ± 0.01	-0.01	-0.02
DECODER NUM CELLS	-	-	3	4	4.62 ± 0.00	4.59 ± 0.01	-0.13	0.12
ENCODER LAYERS	0	LEFT	8 HEAD ATTENTION	GATED LINEAR UNIT	4.80 ± 0.03	4.45 ± 0.02	0.05	-0.01
ENCODER LAYERS	2	LEFT	STANDARD CONV 1X1	SEPARABLE CONV 9X1	4.69 ± 0.01	4.50 ± 0.00	-0.06	0.04
ENCODER LAYERS	1	RIGHT	DEAD BRANCH	STANDARD CONV 3X1	4.73 ± 0.01	4.47 ± 0.03	-0.02	0.01
ENCODER NORMS	2	LEFT	NONE	LAYER NORM	4.79 ± 0.03	4.46 ± 0.02	0.04	0.0
ENCODER OUTPUT DIM	2	LEFT	2	1	4.74 ± 0.01	4.45 ± 0.0	-0.01	-0.01

Table 5. **Mutation Ablations**: Each mutation is described by the first 5 columns. The augmented Transformer and augmented ET perplexities on the WMT'14 En-De validation set are given in columns 6 and 7. Columns 7 and 8 show the difference between the unaugmented base model perplexity mean and the augmented model perplexity mean. Red highlighted cells indicate evidence that the corresponding mutation hurts overall performance. Green highlighted cells indicate evidence that the corresponding mutation helps overall performance.