Reservoir Transformers

Sheng Shen[†], Alexei Baevski[‡], Ari S. Morcos[‡], Kurt Keutzer[†], Michael Auli[‡], Douwe Kiela[‡]
[†]UC Berkeley; [‡]Facebook AI Research
sheng.s@berkeley.edu, dkiela@fb.com

Abstract

We demonstrate that transformers obtain impressive performance even when some of the layers are randomly initialized and never updated. Inspired by old and well-established ideas in machine learning, we explore a variety of non-linear "reservoir" layers interspersed with regular transformer layers, and show improvements in wall-clock compute time until convergence, as well as overall performance, on various machine translation and (masked) language modelling tasks.

1 Introduction

Transformers (Vaswani et al., 2017) have dominated natural language processing (NLP) in recent years, from large scale machine translation (Ott et al., 2018) to pre-trained (masked) language modeling (Devlin et al., 2018; Radford et al., 2018), and are becoming more popular in other fields as well, from reinforcement learning (Vinyals et al., 2019) to speech recognition (Baevski et al., 2019) and computer vision (Carion et al., 2020). Their success is enabled in part by ever increasing computational demands, which has naturally led to an increased interest in improving their efficiency. Scalability gains in transformers could facilitate bigger, deeper networks with longer contexts (Kitaev et al., 2020; Wang et al., 2020; Beltagy et al., 2020; Kaplan et al., 2020; Tay et al., 2020b). Conversely, improved efficiency could reduce environmental costs (Strubell et al., 2019) and hopefully help democratize the technology.

In this work, we explore a simple question: if some layers of the transformer are kept frozen—i.e., never updated after random initialization—can we match the performance of fully learned transformers, while being more efficient? Surprisingly, the answer is resoundingly yes; and what is more, we find that freezing layers may actually *improve* performance.

Beyond desirable efficiency gains, random layers are interesting for several additional reasons. Fixed randomly initialized networks (Gallicchio and Scardapane, 2020) converge to Gaussian processes in the limit of infinite width (Daniely et al., 2016), have intriguing interpretations in metric learning (Rosenfeld and Tsotsos, 2019; Giryes et al., 2016), and have been shown to provide excellent "priors" either for subsequent learning (Ulyanov et al., 2018) or pruning (Frankle and Carbin, 2018). Fixed layers allow for efficient low-cost hardware implementations (Schrauwen et al., 2007) and can be characterized using only a random number generator and its seed. This could facilitate distributed training and enables highly efficient deployment to edge devices, since it only requires transmission of a single number. The strong performance of networks with fixed layers also sheds new light on the inner workings of BERT (Devlin et al., 2018), and layer-wise interpretations of such models (Rogers et al., 2020; Tenney et al., 2019). It appears that "not all layers are created equal" (Zhang et al., 2019) is true to such an extent that some layers can simply remain random and fixed.

Random projections have a long history in machine learning. By Cover's theorem (Cover, 1965), any high-dimensional non-linear transformation is more likely to be linearly separable than its lower-or-equal-dimensional input space. By Johnson-Lindenstrauss (Johnson and Lindenstrauss, 1984), random projections distort Euclidean distances very little under mild assumptions, which is useful e.g. for dimensionality reduction and random indexing (Sahlgren, 2005). Fixed random layers in neural networks pre-date deep learning by far (Gamba et al., 1961; Baum, 1988). Indeed, random kernel methods have long been influential in machine learning (Rahimi and Recht, 2008, 2009).

One way to think of such layers is as "reservoirs" (Lukoševičius and Jaeger, 2009), where a highly non-linear high-dimensional black box representation is provided to a lightweight "readout" network, as in echo state networks (Jaeger, 2003) and liquid state machines (Maass et al., 2002). The benefit of such an approach is that the reservoir has fixed parameters and is computationally efficient, as it can be pre-computed and does not (necessarily) require backpropagation.

In NLP, Wieting and Kiela (2019) showed that random sentence encoders present a strong baseline for text classification, with subsequent work showing applications in a variety of tasks from summarization to machine translation (Enguehard et al., 2019; Garg et al., 2020; Pilault et al., 2020). To our knowledge, this work is the first to examine this phenomenon in transformers, and the first to recursively alternate reservoirs with subsequent transformer layers acting as readout functions. We introduce "reservoir transformers", wherein fixed random reservoir layers are interspersed with regular updateable transformer layers. The goal of this work is to put our understanding of transformer models on a more solid footing by providing empirical evidence of their capabilities even when some of their parameters are fixed. Our contributions are as follows:

- We introduce a *area under the convergence curve* metric for measuring performance-efficiency trade-offs, and show that replacing regular transformer layers with reservoir layers leads to improvements.
- We show that the addition of reservoir layers leads to improved test set generalization on a variety of tasks in a variety of settings.
- We show that pre-trained masked language modelling architectures like BERT and RoBERTa (Liu et al., 2019) can benefit from having some of their layers frozen, both during pre-training as well as when fine-tuning on downstream tasks.
- We experiment with different types of reservoir layers, including convolutional and recurrent neural network-based ones.
- We show empirical evidence that the backward pass can be skipped *in its entirety* by approximating upstream gradients using an approach we call *backskipping*, which can reduce the training compute further without sacrificing performance.

2 Approach

This paper is based on a very simple idea. Neural networks are trained via backpropagation, which involves consecutive steps of matrix addition and multiplication, i.e.,

$$\theta_{t+1} \leftarrow \theta_t - \eta \frac{\partial J}{\partial \theta_t}; \frac{\partial J}{\partial \theta_t} = \frac{\partial J}{\partial L_n} \frac{\partial L_n}{\partial L_{n-1}} \cdots \frac{\partial L_0}{\partial x}$$

for some objective J, parameterization θ and learning rate η, with the gradient computed via the chain rule, where L_i is the i-th layer of the neural network and x is the input. Let $L = \text{Transformer}(X)$ be a single layer in a Transformer network (Vaswani et al., 2017), i.e.,

$$\begin{aligned} H &= \text{MultiHeadSelfAttn}(\text{LayerNorm}(X)) + X \\ L &= \text{FFN}(\text{LayerNorm}(H)) + H \end{aligned}$$

Now, during every "backward pass", we compute the Jacobian for parameters θ^L at layer L, which are used to update the parameters of L, θ^L_t, as well as to compute the next layer's Jacobian, thus back-propagating the gradients. In this work however, for some of the layers, we still backpropagate through them to compute gradients for earlier layers, *but we never apply the parameter update*. As a result, these layers stay fixed at their initialization, saving computational resources.

2.1 Background

Naturally, never updating some of the parameters is computationally more efficient, as some matrix addition operations can be skipped in the backward pass, but why is this not detrimental to the performance of the network?

In the early days of neural networks, the bottom layers were often kept fixed as "associators" (Block, 1962), or what (Minsky and Papert, 2017) called the Gamba perceptron (Gamba et al., 1961; Borsellino and Gamba, 1961). Fixed random networks (Baum, 1988; Schmidt et al., 1992; Pao et al., 1994) have been explored from many angles, including as "random kitchen sink" kernel machines (Rahimi and Recht, 2008, 2009), "extreme learning machines" (Huang et al., 2006) and reservoir computing (Jaeger, 2003; Maass et al., 2002; Lukoševičius and Jaeger, 2009). In reservoir computing, input data are represented through fixed random high-dimensional non-linear representations, called "reservoirs", which are followed by a regular (often but not necessarily linear) "readout" network to make the final classification decision.

The theoretical justification for these approaches lies in two well-known results in machine learning: Cover's theorem (Cover, 1965) on the separability of patterns states that high-dimensional non-linear transformations are more likely to be linearly separable; and the Johnson-Lindenstrauss lemma (Johnson and Lindenstrauss, 1984) shows that (most) random projections distort Euclidean distances very little.

Practically, random layers can be seen as a cheap way to increase network depth. There are interesting advantages to this approach. Fixed layers are known to have particularly low-cost hardware requirements and can be easily implemented on high-bandwidth FPGAs with low power consumption (Hadaeghi et al., 2017; Tanaka et al., 2019), or on optical devices (Hicke et al., 2013). This might yield interesting possibilities for training in a distributed fashion across multiple devices, as well as for neuromorphic hardware (Neftci et al., 2017). This approach also facilitates lower-latency deployment of neural networks to edge devices, since weights can be shared simply by sending the seed number, assuming the random number generator is known on both ends.

2.2 Reservoir Transformers

This work explores inserting random non-linear transformations, or what we call reservoir layers, into transformer networks. Specifically, we experiment with a variety of reservoir layers:

- Transformer Reservoir: The standard transformer layer as described above, but with all parameters fixed after initialization, including the self-attention module.
- FFN Reservoir: A transformer-style fixed feed-forward layer without any self-attention, i.e., FFN(LayerNorm(Previous_layer)) + Previous_layer.
- BiGRU Reservoir: A fixed bidirectional Gated Recurrent Unit (Cho et al., 2014) layer, which is closer in spirit to previous work on reservoir computing, most of which builds on recurrent neural network architectures.
- CNN Reservoir: A fixed Convolutional Neural Network (LeCun et al., 1998) layer, specifically light dynamical convolution layers (Wu et al., 2019), which are known to be competitive with transformers in sequence-to-sequence tasks.

We find that all these approaches work well, to a certain extent. For clarity, we focus primarily on the first two reservoir layers, but include a broader comparison in Appendix A.

In each case, contrary to traditional reservoir computing, our reservoir layers are interspersed throughout a regular transformer network, or what we call a reservoir transformer. Since random projections are not learned and might introduce noise, subsequent normal transformer "readout" layers might be able to benefit from additional depth while allowing us to recover from any adverse effects of randomness. For example, previous work has shown that ResNets, with all of their parameters fixed except for the scale and shift parameters of batch normalization, can still achieve high performance, simply by scaling and shifting random features (Frankle et al., 2020). Adding some form of noise to the parameters is also known to help convergence and generalization (Jim et al., 1995, 1996; Gulcehre et al., 2016; Noh et al., 2017).

3 Evaluation

We evaluate the proposed approach on a variety of well-known tasks in natural language processing, namely: machine translation, language modelling and masked language model pre-training.

We set out to do this work with the main objective of examining any potential efficiency gains, i.e. the relationship between compute time and task performance. This is closely related to efforts in Green AI, which are concerned with the trade-offs between compute, data, and performance (Schwartz et al., 2019). We propose to measure this trade-off via the *area under the convergence curve* (AUCC): similarly to how the area under the receiver operating characteristic (Bradley, 1997, AUC-ROC) measures a classifier's performance independent of the classification threshold, AUCC measures a model's performance independent of the specific compute budget. Specifically, AUCC is computed as follows:

$$\int_{t=0}^{\hat{T}} \sum_{x,y \in \mathcal{D}} g_t(f(x), y) \tag{1}$$

where f is the network and g is the evaluation metric, measured until convergence time $\hat{T}$, which is the maximum convergence time of all models included in the comparison. Note that time here is wall-clock time, not iterations. By convergence, we mean that validation performance has stopped improving, and hence the convergence curve whose area we measure plots the desired metric over time. Runs are averaged over multiple seeds and reported with standard deviation. We normalize raw AUCC scores by their maximum to ensure a more interpretable $[0-1]$ range.

One potential downside of this approach is that the AUCC metric could lead to higher scores for a model that converges quickly but to ultimately worse performance, if measured in a small window. This can be solved by making sure that $\hat{T}$ is set sufficiently high. We include the raw validation curves in the appendix to demonstrate that the chosen window sizes are sufficient and the results are not a influenced by this limitation. In addition, we report the number of trainable parameters and the wall-clock training time until maximum performance (plus 95% and 99% convergence results in the appendix). Finally, we show test set generalization in each experiment. Overall, this gives us a wide set of axes along which to examine models.

3.1 Experimental Settings

We evaluate on IWSLT de-en (Cettolo et al., 2015) and WMT en-de (Bojar et al., 2014) for machine translation; enwiki8 (LLC, 2009) for language modelling; and experiment with RoBERTa (Liu et al., 2019) in our pretraining experiments. For IWSLT, we follow the pre-processing steps in Edunov et al. (2018). The train/val/test split is 129k/10k/6.8k sentences. For WMT, we follow pre-process as in Ott et al. (2018), with 4.5M/16.5k/3k sentences in train/val/test. For enwiki8, we follow the pre-processing steps in Dai et al. (2019). The train/val/test split is 1M/54k/56k sentences. For RoBERTa pretraining, we follow the pre-processing steps in Liu et al. (2019).

We use 8 Volta V100 GPUs for WMT and enwik8, 32 V100 GPUs for RoBERTa and a single V100 for IWSLT. The hyperparameters for IWSLT14 and WMT16 were set to the best-performing values from Ott et al. (2018) and Kasai et al. (2020) respectively. The enwik8 experiment settings followed Bachlechner et al. (2020) and the RoBERTa experiments followed Liu et al. (2019).

All the experiments in this paper were run with 3 random seeds and the mean and standard deviation are reported. For the relatively small IWSLT, the $\hat{T}$ value in the AUCC metric was set to 4 hours. For the larger WMT, we set it to 20 hours. For enwiki8, it was 30 hours; and for the RoBERTa pre-training experiments, it was set to 60 hours.

The projection weights in random layers were initialized using orthogonal initialization (Saxe et al., 2013), since random orthogonal projections should ideally be maximally information-preserving, and which was found to work well empirically for initializing fixed random representations in previous work (Wieting and Kiela, 2019). Biases and layer norm parameters were initialized using their respective PyTorch defaults (based on Xavier init; Glorot and Bengio, 2010).

We intersperse reservoir layers in alternating fashion starting from the middle. Specifically, we alternate one reservoir layer with one transformer layer, and place the alternating block in the middle. For example: a 7-layer encoder LLLLLLL in which we replace three layers with reservoirs becomes LRLRLRL, and with two becomes LLRLRLL. See Appendix C for a study comparing this strategy to alternative approaches (e.g., freezing in the bottom, middle or top).

4 Experiments

In what follows, we first show our main result, on a variety of tasks: reservoir transformers mostly have better AUCC metrics; less training time per epoch; less convergence time until the best validation performance is achieved; and even improved test set generalization metrics. As a strong baseline method, we compare to LayerDrop (Fan et al., 2019). LayerDrop can also be seen as a method that dynamically bypasses parts of the computation during Transformer training in an attempt to improve efficiency, and making it a strong comparison to examine our methods. Then, we examine whether we can minimize the expectation over the gradients of upstream layers in the network such that we do not *at all* have to pass gradients through the reservoir layers, skipping their backward pass.

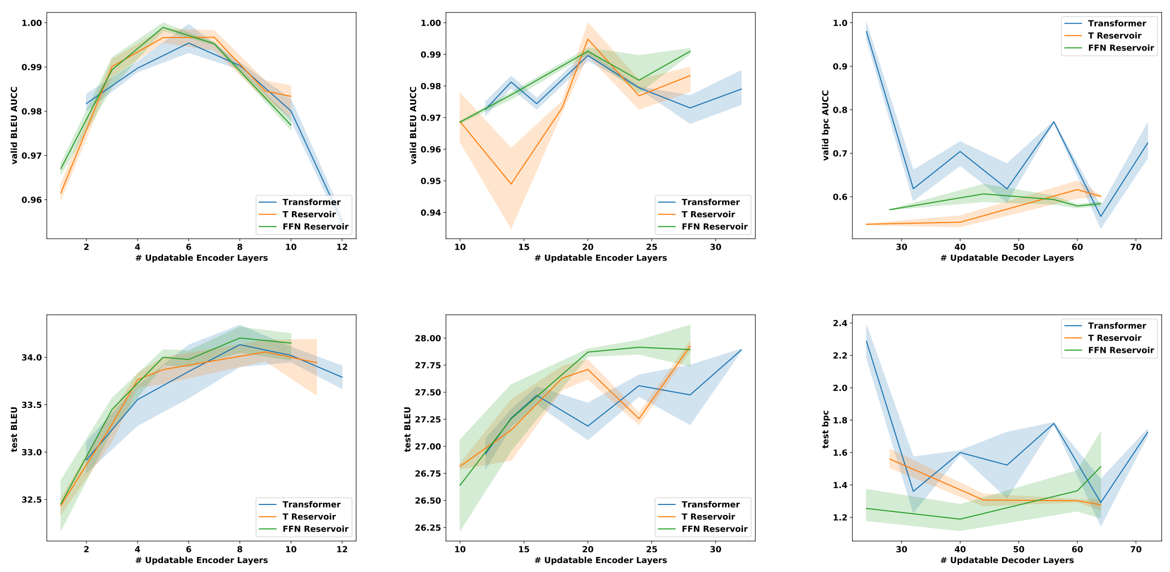

Figure 1: Validation (top) and test (bottom) results for IWSLT (left), WMT (middle) and enwiki8 language modelling (right). IWSLT and WMT are BLEU (high is good); enwiki8 is BPC (low is good). Comparison of regular transformer (blue) and reservoir transformer with FFN (green) or Transformer reservoir (orange) layers added.

4.1 Machine Translation

Machine translation (MT) is one of the core tasks of NLP. We demonstrate on two well-known MT datasets, IWSLT'14 German-English and WMT'16 English-German, that reservoir transformers obtain a better AUCC. For the raw validation plots over time that were used to calculate the AUCC, please refer to Appendix F.

Following Kasai et al. (2020), the architecture of the network is an N-layer reservoir transformer encoder, followed by a regular shallow one- or two-layer decoder. This design choice has been shown to lead to very good speed and efficiency trade-offs, and serves as a good baseline for our experiments. Moreover, shallow decoders make it easier to decide where to place reservoir layers (in the encoder) and makes it more straightforward to identify where performance gains come from.

Figure 1 shows the results for IWSLT (left) and WMT (middle). On the y-axis we show validation AUCC for the BLEU metric; on the x-axis we show the number of updatable layers in the encoder. The performance of a regular transformer encoder with 6 layers and a reservoir transformer encoder with 6 layers plus N additional reservoir layers are plotted for the same x-axis value to show the total number of *updated* layers. Plots for the *total* number of layers (updatable plus not-updatable, so essentially shifted versions of the plots) are shown in Appendix E.

WMT is much larger and requires a much deeper encoder, as illustrated by the fact that a certain minimum depth is required for reservoir transformers to achieve a comparable validation AUCC. At test time, reservoir transformers outperform regular transformers for almost all encoder depths. The FFN Reservoir seems to work best in both cases, which is surprising because it does not have any self-attention component at all. This finding shows that self-attention, or the mechanism to summarize context information, should be learned if present. Once the context features have been gathered, a random projection via a fixed FFN module appears to be beneficial.

Table 1 and 2 show the time it took to achieve the maximum validation BLEU score and how that relates to the regular transformer, demonstrating that reservoir transformers consistently converge faster in terms of wall-clock time. We save up to 22% convergence wall-clock time using reservoir transformers as much with the same number of updateable layers. We save as much as 27% time until convergence a 24 layer model on WMT, as shown in Table 2. One other noticeable point is that we can see that the T Reservoir achieves similar performance to LayerDrop on IWSLT and WMT in terms of wall-clock per epoch and wall-clock time to the best performance. However, on both tasks, FFN Reservoir performs much better than LayerDrop in terms of efficiency per epoch

Model	**# Layers**	**Frozen**	**Max BLEU**	**Train time until max** (in hours)	**Ratio**	**# Params Trainable** (Total)	**Train Time each epoch** (in seconds)
Transformer	6	0	34.52 ± 0.07	2.548 ± 0.06	1	26.8M	122.73 ± 1.16
	8	0	34.59 ± 0.11	2.557 ± 0.05	1	31.1M	142.28 ± 1.87
	10	0	34.56 ± 0.05	3.173 ± 0.04	1	35.3M	161.66 ± 1.54
	12	0	34.29 ± 0.12	3.521 ± 0.09	1	39.5M	172.45 ± 1.98
T Reservoir	6	2	34.37 ± 0.12	2.422 ± 0.03	0.95	22.6M (26.8M)	120.59 ± 1.32
	8	2	34.80 ± 0.07	2.450 ± 0.06	0.96	26.8M (31.1M)	134.49 ± 1.76
	10	2	34.70 ± 0.03	2.831 ± 0.05	0.89	31.1M (35.3M)	144.42 ± 1.98
	12	2	34.78 ± 0.04	3.476 ± 0.04	0.98	35.3M (39.5M)	159.43 ± 1.67
FFN Reservoir	6	2	34.43 ± 0.15	2.120 ± 0.04	0.83	22.6M (25.8M)	107.71 ± 1.73
	8	2	34.56 ± 0.16	2.203 ± 0.06	0.86	26.8M (29.1M)	120.07 ± 1.65
	10	2	34.66 ± 0.02	2.493 ± 0.05	0.79	31.1M (33.3M)	130.11 ± 1.43
	12	2	34.76 ± 0.03	3.241 ± 0.04	0.92	35.3M (37.5M)	156.32 ± 1.87
LayerDrop	6	2	34.59 ± 0.15	2.364 ± 0.08	0.92	22.6M (26.8M)	119.30 ± 1.36
	8	2	34.58 ± 0.16	2.554 ± 0.05	0.99	26.8M (31.1M)	138.62 ± 1.44
	10	2	34.57 ± 0.07	3.404 ± 0.06	1.07	31.1M (35.3M)	140.88 ± 1.62
	12	2	33.65 ± 0.24	3.251 ± 0.04	0.92	35.3M (39.5M)	160.85 ± 1.49

Table 1: Wall-clock time (averaged over multiple runs) saved for IWSLT for different model types and encoder depths. Max BLEU is for validation. Number of layers is for encoder, decoder depth is kept fixed at 2. The ratio is computed compared to the corresponding number of layers in the regular transformer case.

Model	**# Layers**	**Frozen**	**Max BLEU**	**Train time until max** (in hours)	**Ratio**	**# Params Trainable** (Total)	**Train Time each epoch** (in hours)
Transformer	12	0	24.46 ± 0.04	15.15 ± 0.15	1	75.6M	0.505 ± 0.005
	16	0	24.52 ± 0.03	16.05 ± 0.18	1	88.2M	0.643 ± 0.006
	24	0	24.69 ± 0.05	17.61 ± 0.85	1	113.4M	0.877 ± 0.029
	32	0	24.83 ± 0.04	18.42 ± 0.28	1	138.6M	1.036 ± 0.010
T Reservoir	12	4	24.26 ± 0.08	14.11 ± 0.21	0.93	72.4M (75.6M)	0.472 ± 0.007
	16	4	24.50 ± 0.05	15.25 ± 0.28	0.95	75.6M (88.2M)	0.596 ± 0.009
	24	4	25.11 ± 0.07	15.89 ± 0.74	0.90	100.8M (113.4M)	0.776 ± 0.024
	32	4	24.66 ± 0.04	16.38 ± 0.24	0.88	126.0M (138.6M)	0.998 ± 0.009
FFN Reservoir	12	4	24.42 ± 0.05	14.01 ± 0.09	0.92	72.4M (71.4M)	0.441 ± 0.003
	16	4	24.65 ± 0.07	14.53 ± 0.17	0.91	75.6M (83.9M)	0.524 ± 0.006
	24	4	24.93 ± 0.04	12.62 ± 1.53	0.71	100.8M (109.2M)	0.743 ± 0.018
	32	4	24.98 ± 0.03	13.96 ± 0.19	0.73	126.0M (134.4M)	0.964 ± 0.007
LayerDrop	12	4	24.27 ± 0.03	14.61 ± 0.14	0.96	72.4M (75.6M)	0.489 ± 0.006
	16	4	24.15 ± 0.06	15.55 ± 0.54	0.97	75.6M (88.2M)	0.597 ± 0.017
	24	4	24.37 ± 0.05	16.25 ± 0.36	0.92	100.8M (113.4M)	0.823 ± 0.013
	32	4	23.84 ± 0.03	15.27 ± 0.38	0.83	126.0M (138.6M)	1.028 ± 0.012

Table 2: Wall-clock time (averaged over multiple runs) saved for WMT for different model types and encoder depths. Decoder depth is kept fixed at 1.

and achieves better/similar performance in less time in each case. As a point of reference, a half hour gain on IWSLT would translate to a gain of several days in the training of bigger transformer models like GPT-3 (Brown et al., 2020).

We observe that reservoir transformers consistently perform better than, or are competitive to, regular transformers, both in terms of validation BLEU AUCC as well as test time BLEU, for all examined encoder depths.

4.2 Language Modelling

To examine whether the same findings hold for other tasks, we evaluate on the enwiki8 (LLC, 2009) language modelling task. We examine the BPC (bits per character) rate for a variety of network depths (since the task is language modelling, these layers are in the decoder). The results show that except for the 64-layer regular transformer, which appears to be particularly optimal for this task, we obtain consistently better BPC for all depths. We observe similar trends during test time.

4.3 Masked Language Model Pretraining

We train RoBERTa (Liu et al., 2019) models from scratch at a variety of depths, both in the normal and reservoir setting. We find that these networks show minor differences in their best perplexity

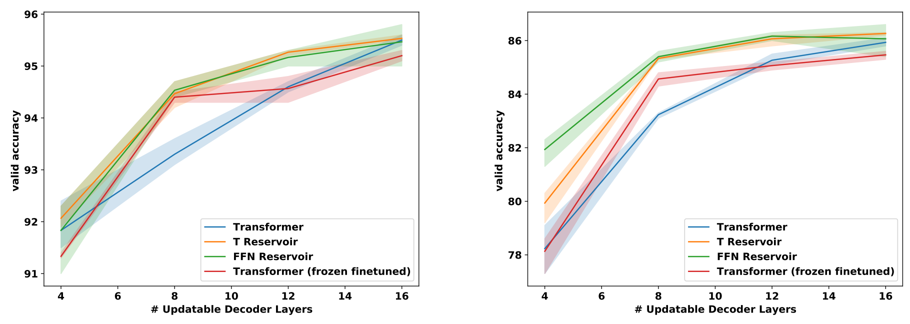

Figure 2: Downstream RoBERTa performance on SST-2 (left) and MultiNLI-matched (right).

Model	Max BLEU	AUCC	Train time
Transformer	34.59 ± 0.11	114.57 ± 0.08	142.28 ± 1.87
T Reservoir	34.80 ± 0.07	115.26 ± 0.26	134.49 ± 1.70
Backskip Reservoir	34.75 ± 0.05	115.99 ± 0.23	119.54 ± 1.78

Table 3: Validation max BLEU, AUCC at 4h and wall-clock time per epoch (averaged over multiple runs, in seconds) on IWSLT comparing backskipping with regular and reservoir transformers.

and similar AUCC perplexity (see Appendix D). We then examine the performance of these models when fine-tuned on downstream tasks, specifically the well known SST-2 (Socher et al., 2013) and MultiNLI-matched (Williams et al., 2017) tasks. When fine-tuning the reservoir models, we keep the reservoir layers fixed (also fine-tuning them did not work very well, see Appendix D).

Figure 2 shows the results of fine-tuning. We observe that the reservoir transformer outperforms normal RoBERTa at all depths in both tasks. At lower depth, the improvements are substantial. As a sanity check, we also experiment with freezing some of the layers in a regular pre-trained RoBERTa model during fine-tuning only (Transformer "frozen finetuned" in the Figure) and show that this helps a little but is still outperformed by the reservoir transformer.

These findings suggest that we can train a RoBERTa model without updating all of the layers, achieving similar perplexity at a similar computational cost, but with *better* downstream performance. This strategy could prove to be beneficial in a wide variety of pre-training scenarios.

We follow Jawahar et al. (2019) and investigate what the frozen layers in the Reservoir Transformer have actually "learned" (while being frozen) as measured by probing tasks, reported in Table 4. The set of tasks comprises one surface task, three syntactic tasks, and five semantic tasks. From the table, we can see that generally probing performance is quite similar between Transformer and the T Reservoir model. We also noticed that the representations collected after the reservoir layer (3, 5, 7, 9) in the T Reservoir actually have significantly better performance over the regular Transformer representations across all the probing tasks. Related to our findings, Voita and Titov (2020) show that the wholly-randomly-initialized model representations can still have reasonable probing accuracy if they are contextualized, though the accuracy is strictly worse than a trained one. These findings raise interesting repercussions for the study of "BERTology", as it clearly shows that even completely random and frozen layers can represent linguistic phenomena.

4.4 Backskipping

With the reservoir transformers as described above, we obtain better efficiency by skipping the "gradient application" matrix addition step in some of the layers (i.e., updating the weights). One step further would be to investigate skipping the entire backward pass for reservoirs altogether, which would save us from having to do the much more expensive matrix multiplication for these layers that is required for the propagation of gradients through a regular layer.

We report on preliminary experiments where in the backward pass we replace the gradients for the layer L_i *going into* the reservoir L_{i+1} with a noisy estimate (Jaderberg et al., 2017; Czarnecki et al., 2017). Promisingly, Oktay et al. (2020) recently asked "why spend resources on exact gradients when we're going to use stochastic optimization?" and show that we can do randomized auto-differentiation quite successfully.

Model	Layer	**SentLen** (Surface)	**TreeDepth** (Syntactic)	**TopConst** (Syntactic)	**BShift** (Syntactic)	**Tense** (Semantic)	**SubjNum** (Semantic)	**ObjNum** (Semantic)	**SOMO** (Semantic)	**CoordInv** (Semantic)
Transformer	1	84.56 ± 0.54	32.30 ± 0.41	54.40 ± 0.33	49.99 ± 0.01	80.98 ± 0.32	76.26 ± 0.09	50.01 ± 0.19	76.38 ± 0.61	54.33 ± 0.47
	2	87.22 ± 0.07	33.63 ± 0.57	58.38 ± 0.20	50.12 ± 0.17	82.84 ± 0.68	78.65 ± 0.19	51.47 ± 0.53	78.00 ± 1.12	54.66 ± 0.55
	3	84.25 ± 0.16	32.60 ± 0.17	54.41 ± 0.10	50.02 ± 0.01	81.72 ± 0.59	77.00 ± 0.13	51.32 ± 0.64	76.57 ± 1.13	54.13 ± 0.51
	4	87.37 ± 0.20	32.59 ± 0.29	50.06 ± 0.21	69.76 ± 0.26	81.63 ± 1.17	76.47 ± 0.09	52.41 ± 1.49	76.15 ± 0.84	52.62 ± 1.34
	5	84.61 ± 0.24	31.14 ± 0.48	44.76 ± 0.38	74.82 ± 0.11	80.16 ± 0.19	73.66 ± 0.16	52.95 ± 1.77	72.90 ± 0.21	51.26 ± 1.14
	6	82.56 ± 0.25	30.31 ± 0.40	39.30 ± 0.40	78.80 ± 0.38	81.88 ± 0.47	75.30 ± 0.07	56.21 ± 1.26	74.37 ± 0.16	51.44 ± 1.04
	7	70.85 ± 0.13	26.65 ± 0.72	40.70 ± 0.13	78.98 ± 0.32	85.11 ± 0.31	72.03 ± 0.46	58.15 ± 0.46	68.71 ± 0.91	55.39 ± 0.27
	8	66.23 ± 1.33	23.46 ± 0.44	25.19 ± 1.02	77.42 ± 0.27	80.35 ± 0.45	67.55 ± 0.99	54.94 ± 2.04	63.69 ± 2.32	50.58 ± 0.83
	9	71.17 ± 0.29	31.21 ± 0.31	58.42 ± 0.29	85.55 ± 0.44	86.77 ± 0.19	80.30 ± 0.08	64.36 ± 1.20	81.68 ± 0.45	66.90 ± 0.49
	10	73.19 ± 0.50	27.74 ± 0.53	41.01 ± 0.22	83.56 ± 0.96	86.13 ± 0.35	83.04 ± 0.04	62.01 ± 0.59	79.73 ± 0.21	62.60 ± 1.04
	11	71.37 ± 0.42	30.22 ± 0.28	48.58 ± 0.35	84.40 ± 0.44	87.28 ± 0.59	82.34 ± 0.15	61.10 ± 0.14	80.00 ± 0.40	64.44 ± 0.38
	12	71.66 ± 0.12	33.43 ± 0.18	64.38 ± 0.20	87.38 ± 0.02	88.41 ± 0.09	84.46 ± 0.25	63.01 ± 0.05	81.80 ± 0.27	65.72 ± 0.16
T Reservoir	1	87.75 ± 0.10	31.60 ± 0.21	50.38 ± 0.23	50.00 ± 0.00	80.40 ± 0.18	76.47 ± 0.20	50.53 ± 0.14	73.48 ± 0.15	53.55 ± 0.70
	2	81.28 ± 0.23	34.20 ± 0.41	61.41 ± 0.42	60.64 ± 0.65	81.50 ± 0.77	76.33 ± 0.08	50.73 ± 0.34	74.28 ± 0.67	56.82 ± 0.10
	3	$\mathbf{89.28 \pm 0.09}$	$\mathbf{36.42 \pm 0.11}$	$\mathbf{67.36 \pm 0.45}$	$\mathbf{75.64 \pm 0.52}$	$\mathbf{85.42 \pm 0.18}$	$\mathbf{80.53 \pm 0.02}$	$\mathbf{52.50 \pm 1.80}$	$\mathbf{78.47 \pm 1.81}$	$\mathbf{57.16 \pm 0.27}$
	4	74.31 ± 0.32	32.42 ± 0.83	55.19 ± 0.33	73.41 ± 0.00	79.56 ± 0.00	75.15 ± 0.08	53.68 ± 0.66	75.02 ± 0.19	56.89 ± 0.08
	5	$\mathbf{88.03 \pm 0.22}$	$\mathbf{38.34 \pm 0.64}$	$\mathbf{68.65 \pm 0.29}$	$\mathbf{82.25 \pm 0.12}$	$\mathbf{86.80 \pm 0.02}$	$\mathbf{82.27 \pm 0.33}$	$\mathbf{57.95 \pm 0.24}$	$\mathbf{80.82 \pm 0.91}$	$\mathbf{58.05 \pm 0.10}$
	6	74.55 ± 0.37	33.13 ± 0.29	52.70 ± 0.81	79.21 ± 0.13	85.70 ± 0.36	77.43 ± 0.03	57.26 ± 0.19	75.38 ± 0.66	51.95 ± 1.30
	7	$\mathbf{85.82 \pm 0.37}$	$\mathbf{37.63 \pm 0.13}$	$\mathbf{70.43 \pm 0.05}$	$\mathbf{84.12 \pm 0.35}$	$\mathbf{86.88 \pm 0.07}$	$\mathbf{82.86 \pm 0.30}$	$\mathbf{61.17 \pm 0.21}$	$\mathbf{80.79 \pm 0.17}$	$\mathbf{61.83 \pm 0.95}$
	8	71.69 ± 0.71	30.32 ± 0.01	48.44 ± 0.30	79.12 ± 0.12	84.75 ± 0.09	79.23 ± 0.11	59.53 ± 0.16	76.80 ± 0.41	57.34 ± 0.14
	9	$\mathbf{85.86 \pm 0.12}$	$\mathbf{37.89 \pm 0.03}$	$\mathbf{69.53 \pm 0.37}$	$\mathbf{85.55 \pm 0.12}$	$\mathbf{87.98 \pm 0.22}$	$\mathbf{84.13 \pm 0.01}$	$\mathbf{63.06 \pm 0.01}$	$\mathbf{82.55 \pm 0.31}$	$\mathbf{66.07 \pm 0.05}$
	10	69.22 ± 0.23	25.58 ± 0.35	29.20 ± 0.58	78.57 ± 0.09	85.02 ± 0.03	75.68 ± 0.16	57.55 ± 1.57	74.70 ± 0.02	55.02 ± 0.64
	11	65.70 ± 0.05	30.57 ± 0.03	47.56 ± 0.02	81.20 ± 0.00	86.78 ± 0.02	83.73 ± 0.05	60.38 ± 0.17	80.59 ± 0.15	62.50 ± 0.11
	12	70.61 ± 0.18	34.45 ± 0.20	64.19 ± 0.10	84.53 ± 0.03	87.48 ± 0.16	84.86 ± 0.14	62.75 ± 0.14	82.08 ± 0.03	64.73 ± 0.06

Table 4: RoBERTa Probing Results. The line in bold text are the the frozen layers in the T Reservoir. Mean accuracy with standard deviation, gathered over 3 random seeds.

Here, rather than minimizing the actual gradients $\frac{\partial L_i}{\partial \theta^{L_i}}$, we minimize their expectation and train via continuous-action REINFORCE (Williams, 1992). That is, L_i becomes a policy π_a: $s \rightarrow \mu$ where we sample actions $a \sim \mathcal{N}(\mu, 1)$. We train to minimize the gradient prediction loss via MSE, i.e., $\frac{1}{n}\sum_{i=0}^{n}(R^i - V^i(a))^2$, and the REINFORCE loss $\mathbb{E}_a\left[\log(a)\left(R - V(a)\right)\right]$, where the value network V acts as the baseline. R is defined as the mean of the gradients of the top layer L_{i+2}, with the sign flipped. Thus, simply put, we train to minimize the expectation of the true gradients at the layer directly following the reservoir. We employ an annealing scheme where we first train the value network and propagate the true gradients during warmup. Afterwards, we anneal the probability of backskipping instead of doing a true backward pass (multiplying the probability by 0.99 every iteration until we only backskip). We experimented with setting R to the negation of the total loss but found the mean upstream gradient reward to work better. We call this approach *backskipping*.

As shown in Table 3, the backskip reservoir approach leads to a higher maximum BLEU score than the regular transformer, with a much higher AUCC and much lower training time. The encoder depth is 8 with 2 frozen. Appendix G shows the raw validation BLEU curves over time. We observe that this approach helps especially during the earlier stages of training. This finding opens up intriguing possibilities for having parts of neural networks be completely frozen both in the forward as well as in the backward pass, while still contributing to the overall model computation.

The computational cost is heavily reduced given that we completely bypass the expensive backpropagation computation in the reservoir layers. Backskipping is shown to be a promising approach to further reduce computational costs, and would be even more efficient from a hardware perspective since the circuitry for such layers (which do not need to propagate gradients) can be hardwired.

5 Related Work

Recent work has shown that modern NLP models are able to function with different numbers of layers for different examples (Elbayad et al., 2019; Fan et al., 2019; He et al., 2021); that different layers specialize for different purposes (Zhang et al., 2019); that layers can be compressed (Li et al., 2020; Zhu et al., 2019; Shen et al., 2020; Sun et al., 2020); and, that layers can be reordered (Press et al., 2019). There is a growing body of work in efficient self-attention networks (Tay et al., 2020b), such as linear attention (Wang et al., 2020), on how to process long context information (Beltagy et al., 2020; Ainslie et al., 2020) and on approximations to make transformers more scalable (Kitaev et al., 2020; Katharopoulos et al., 2020). BigBIRD (Zaheer et al., 2020) provides random keys as additional inputs to its attention mechanism. Locality sensitive hashing (LSH) as employed e.g. in Reformer (Kitaev et al., 2020) utilizes a fixed random projection. Random Feature Attention (Peng et al., 2021) uses random feature methods to approximate the softmax function. Performer (Choromanski et al., 2020) computes the transformer's multi-head attention weights as a fixed orthogonal random projection. Closely related to this work, Tay et al. (2020a) showed that randomized alignment matrices in their "Synthesizer" architecture are sufficient for many NLP tasks. While these works focus on random attention, we show that *entire* layers can be random and fixed. We also show that entire layers can be replaced by fixed random projections that do not have any attention whatsoever.

Beyond transformers, random features have been extensively explored. Examples of this include FreezeOut (Brock et al., 2017), deep reservoir computing networks (Scardapane and Wang, 2017; Gallicchio and Micheli, 2017), as well as applications in domains as varied as text classification (Conneau et al., 2017; Zhang and Bowman, 2018; Wieting and Kiela, 2019) or music classification (Pons and Serra, 2019). It is well known that randomly initialized networks can display impressive performance on their own (Ulyanov et al., 2018; Rosenfeld and Tsotsos, 2019; Ramanujan et al., 2020; Voita and Titov, 2020), which underlies, for example, the recently popularized lottery ticket hypothesis (Frankle and Carbin, 2018; Zhou et al., 2019). We know that learning deep overparameterized networks appears to help in general (Li and Liang, 2018; Du et al., 2019). Our method constitutes a way to add both depth and parameters to transformer networks without much computational cost.

6 Conclusion

This work demonstrated that state-of-the-art transformer architectures can be trained without updating all of the layers. This complements a long history in machine learning of harnessing the power of random features. We use the "area under the convergence curve" (AUCC) metric to demonstrate that on a variety of tasks, and in a variety of settings, "reservoir transformers" achieve better performance-efficiency trade-offs. We show that such reservoir transformers show better convergence rates and test-set generalization. We demonstrated that the backward pass can be skipped altogether, opening up exciting vanues for future research. Future work includes further investigating hybrid networks and backskipping strategies, as well as utilizing pruning.

Acknowledgements

We thank Eric Wallace, Zhewei Yao, Kevin Lin, Zhiqing Sun, Zhuohan Li, Angela Fan, Shaojie Bai, and anonymous reviewers for their comments and suggestions. SS and KK were supported by grants from Samsung, Facebook, and the Berkeley Deep Drive Consortium.

References

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W Cottrell, and Julian McAuley. 2020. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*.

Alexei Baevski, Steffen Schneider, and Michael Auli. 2019. vq-wav2vec: Self-supervised learning of discrete speech representations. *arXiv preprint arXiv:1910.05453*.

Eric B Baum. 1988. On the capabilities of multilayer perceptrons. *Journal of complexity*, 4(3):193–215.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Hans-Dieter Block. 1962. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, Baltimore, Maryland, USA. Association for Computational Linguistics.

A Borsellino and A Gamba. 1961. An outline of a mathematical theory of papa. *Il Nuovo Cimento (1955-1965)*, 20(2):221–231.

Andrew P Bradley. 1997. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. 2017. Freezeout: Accelerate training by progressively freezing layers. *arXiv preprint arXiv:1706.04983*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*.

M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and Marcello Federico. 2015. Report on the 11 th iwslt evaluation campaign , iwslt 2014. In *Proceedings of IWSLT*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Jared Davis, Tamas Sarlos, David Belanger, Lucy Colwell, and Adrian Weller. 2020. Masked language modeling for proteins via linearly scalable long-context transformers. *arXiv preprint arXiv:2006.03555*.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Thomas M Cover. 1965. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334.

Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Understanding synthetic gradients and decoupled neural interfaces. *arXiv preprint arXiv:1703.00522*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy. Association for Computational Linguistics.

Amit Daniely, Roy Frostig, and Yoram Singer. 2016. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances In Neural Information Processing Systems*, pages 2253–2261.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. 2019. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685.

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. 2018. Classical structured prediction losses for sequence to sequence learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana. Association for Computational Linguistics.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2019. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*.

Joseph Enguehard, Dan Busbridge, Vitalii Zhelezniak, and Nils Hammerla. 2019. Neural language priors. *arXiv preprint arXiv:1910.03492*.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Jonathan Frankle, David J Schwab, and Ari S Morcos. 2020. Training batchnorm and only batchnorm: On the expressive power of random features in cnns. *arXiv preprint arXiv:2003.00152*.

Claudio Gallicchio and Alessio Micheli. 2017. Echo state property of deep reservoir computing networks. *Cognitive Computation*, 9(3):337–350.

Claudio Gallicchio and Simone Scardapane. 2020. Deep randomized neural networks. In *Recent Trends in Learning From Data*, pages 43–68. Springer.

A. Gamba, L. Gamberini, G. Palmieri, and R. Sanna. 1961. Further experiments with papa. *Il Nuovo Cimento (1955-1965)*, 20(2):112–115.

Ankush Garg, Yuan Cao, and Qi Ge. 2020. Echo state neural machine translation. *arXiv preprint arXiv:2002.11847*.

Raja Giryes, Guillermo Sapiro, and Alex M Bronstein. 2016. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. 2016. Noisy activation functions. In *International conference on machine learning*, pages 3059–3068.

Fatemeh Hadaeghi, Xu He, and Herbert Jaeger. 2017. *Unconventional Information Processing Systems, Novel Hardware: A Tour D'Horizon*.

Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. 2021. Pipetransformer: Automated elastic pipelining for distributed training of transformers. In *ICML*.

Konstantin Hicke, Miguel Escalona-Moran, Daniel Brunner, Miguel Soriano, Ingo Fischer, and Claudio Mirasso. 2013. Information processing using transient dynamics of semiconductor lasers subject to delayed feedback. *Selected Topics in Quantum Electronics, IEEE Journal of*, 19:1501610–1501610.

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. 2006. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. 2017. Decoupled neural interfaces using synthetic gradients. In *International Conference on Machine Learning*, pages 1627–1635. PMLR.

Herbert Jaeger. 2003. Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems*.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Kam Jim, Bill G Horne, and C Lee Giles. 1995. Effects of noise on convergence and generalization in recurrent networks. In *Advances in neural information processing systems*, pages 649–656.

Kam-Chuen Jim, C Lee Giles, and Bill G Horne. 1996. An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on neural networks*, 7(6):1424–1438.

William B Johnson and Joram Lindenstrauss. 1984. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *arXiv preprint arXiv:2006.10369*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. *arXiv preprint arXiv:2006.16236*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Yuanzhi Li and Yingyu Liang. 2018. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, pages 8157–8166.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joseph E Gonzalez. 2020. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *arXiv preprint arXiv:2002.11794*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

MultiMedia LLC. 2009. Large text compression benchmark.

Mantas Lukoševičius and Herbert Jaeger. 2009. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3).

Wolfgang Maass, Thomas Natschläger, and Henry Markram. 2002. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

Marvin Minsky and Seymour A Papert. 2017. *Perceptrons: An introduction to computational geometry*. MIT press.

Emre O Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. 2017. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in neuroscience*, 11:324.

Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. 2017. Regularizing deep neural networks by noise: Its interpretation and optimization. In *Advances in Neural Information Processing Systems*, pages 5109–5118.

Deniz Oktay, Nick McGreivy, Joshua Aduol, Alex Beatson, and Ryan P Adams. 2020. Randomized automatic differentiation. *arXiv preprint arXiv:2007.10412*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*.

Yoh-Han Pao, Gwang-Hoon Park, and Dejan J Sobajic. 1994. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. 2021. Random feature attention. In *International Conference on Learning Representations*.

Jonathan Pilault, Jaehong Park, and Christopher Pal. 2020. On the impressive performance of randomly weighted encoders in summarization tasks. *arXiv preprint arXiv:2002.09084*.

Jordi Pons and Xavier Serra. 2019. Randomly weighted cnns for (music) audio classification. In *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 336–340. IEEE.

Ofir Press, Noah A Smith, and Omer Levy. 2019. Improving transformer models by reordering their sublayers. *arXiv preprint arXiv:1911.03864*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language models are unsupervised multitask learners.

Ali Rahimi and Benjamin Recht. 2008. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184.

Ali Rahimi and Benjamin Recht. 2009. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, pages 1313–1320.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. 2020. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. *arXiv preprint arXiv:2002.12327*.

Amir Rosenfeld and John K Tsotsos. 2019. Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing. In *2019 16th Conference on Computer and Robot Vision (CRV)*, pages 9–16. IEEE.

Magnus Sahlgren. 2005. An introduction to random indexing. In *Methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering*.

Andrew M Saxe, James L McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Simone Scardapane and Dianhui Wang. 2017. Randomness in neural networks: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(2):e1200.

Wouter F Schmidt, Martin A Kraaijveld, and Robert PW Duin. 1992. Feedforward neural networks with random weights. In *Proceedings of the 11th International Conference on Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4.

Benjamin Schrauwen, Michiel D'Haene, David Verstraeten, and Jan Campenhout. 2007. Compact hardware for real-time speech recognition using a liquid state machine. pages 1097 – 1102.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2019. Green ai. *arXiv preprint arXiv:1907.10597*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170.

Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. 2019. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100 – 123.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking self-attention in transformer models. *arXiv preprint arXiv:2005.00743*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020b. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2018. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Elena Voita and Ivan Titov. 2020. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

John Wieting and Douwe Kiela. 2019. No training required: Exploring random encoders for sentence classification. *arXiv preprint arXiv:1901.10444*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. 2019. Are all layers created equal? *arXiv preprint arXiv:1902.01996*.

Kelly Zhang and Samuel Bowman. 2018. Language modeling teaches you more than translation does: Lessons learned through auxiliary syntactic task analysis. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. 2019. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3597–3607.

Wei Zhu, Xiaofeng Zhou, Keqiang Wang, Xun Luo, Xiepeng Li, Yuan Ni, and Guotong Xie. 2019. PANLP at MEDIQA 2019: Pre-trained language models, transfer learning and knowledge distillation. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 380–388, Florence, Italy. Association for Computational Linguistics.

A Hybrid Networks and Non-Transformer Reservoirs

We investigate whether reservoir layers need to be transformer-based (or transformers-without-attention, i.e., FFN). We examine two different alternatives: bidirectional Gated Recurrent Units (Cho et al., 2014) and Convolutional Neural Networks (LeCun et al., 1998; Kim, 2014), specifically light dynamical convolutions (Wu et al., 2019). Figure 3 shows the results for these hybrids: depending on the setting, they may obtain a better AUCC than the regular transformer, but this is less consistent than with the other reservoir layers, most likely because these layers have different computational properties. It's possible that these hybrids simply require further tuning, as we found e.g. up-projecting to help for BiGRUs, but studying this is outside of the scope of the current work.

Model	# Layers	Frozen	Max BLEU	**Train time until max** (in hours)	Ratio	**# Params Trainable** (Total)	**Train Time each epoch** (in seconds)
Transformer	6	0	34.97 ± 0.05	1.984 ± 0.02	1	39.5M	177.84 ± 2.98
	8	0	34.99 ± 0.08	2.161 ± 0.03	1	43.7M	206.59 ± 3.47
	10	0	34.98 ± 0.04	2.345 ± 0.02	1	47.9M	236.72 ± 3.52
	12	0	34.78 ± 0.11	2.535 ± 0.05	1	52.0M	265.90 ± 4.97
T Reservoir	6	2	34.73 ± 0.11	1.838 ± 0.01	0.92	35.3M (39.5M)	166.11 ± 2.21
	8	2	35.07 ± 0.05	1.912 ± 0.03	0.88	39.5M (43.7M)	190.08 ± 3.73
	10	2	35.02 ± 0.01	1.970 ± 0.04	0.84	43.7M (47.9M)	204.42 ± 2.89
	12	2	35.06 ± 0.02	2.429 ± 0.02	0.95	47.8M (52.0M)	236.41 ± 4.35
FFN Reservoir	6	2	34.85 ± 0.10	1.729 ± 0.03	0.87	35.3M (37.4M)	161.72 ± 2.32
	8	2	34.99 ± 0.11	1.751 ± 0.02	0.81	39.5M (41.6M)	180.21 ± 2.68
	10	2	34.92 ± 0.03	1.907 ± 0.02	0.81	43.7M (45.8M)	191.40 ± 2.49
	12	2	35.16 ± 0.04	2.395 ± 0.01	0.94	47.8M (49.9M)	216.08 ± 2.57
LayerDrop	6	2	34.51 ± 0.12	1.908 ± 0.04	0.96	35.3M (39.5M)	169.62 ± 3.16
	8	2	34.77 ± 0.11	2.023 ± 0.02	0.94	39.5M (43.7M)	186.71 ± 2.17
	10	2	34.06 ± 0.05	1.912 ± 0.02	0.97	43.7M (47.9M)	205.52 ± 3.31
	12	2	34.08 ± 0.13	2.524 ± 0.01	0.99	47.8M (52.0M)	222.45 ± 2.21

Table 5: Wall-clock time (averaged over multiple runs) for IWSLT for different model types and encoder depths. Max BLEU is for validation. Number of layers is for encoder, decoder depth is kept fixed at 6. Ratio is computed compared to comparable number of layers in the normal case.

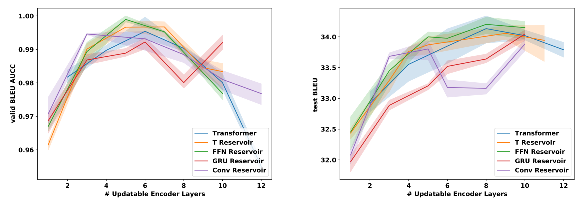

Figure 3: IWSLT comparison of different hybrid architectures with different reservoir layers.

B Deep Decoders

We show that the same results hold for a 6-layer decoder on IWSLT (although less pronounced for AUCC, probably because the decoder is computationally heavier). See Figure 4 and Table 5.

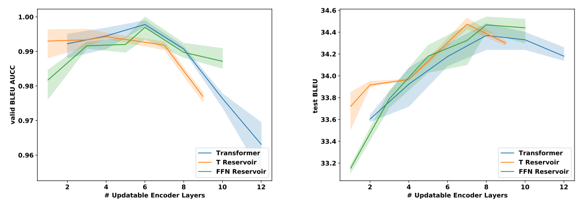

Figure 4: IWSLT validation AUCC and test BLEU with 6-layer decoder.

C Freezing Strategy

We explored different strategies for the placement of reservoir layers and found the "alternating" strategy reported in the main body of the paper to work best. Generally, we found repetitive application of reservoirs to yield diminishing returns, as might be expected. See Figure 5.

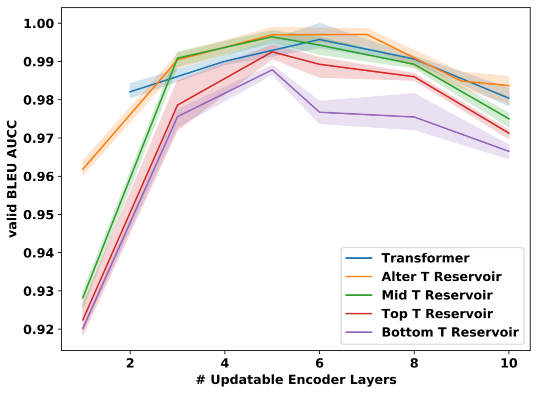

Figure 5: IWSLT with 2-layer decoder using different freezing strategies.

D RoBERTa Results

Here we present the additional results for RoBERTa , i.e., convergence plots and AUCCs for various depth settings, in Figure 7. As stated in the main paper, the differences in terms of AUCC and convergence between RoBERTa models with and without reservoir layers are limited. Moreover, we plot downstream task performance for SST-2 and MNLI compared to the pretraining wall-clock time in Figure 6. It can be seen that the FFN Reservoir can achieve up to 25% and 10% pretraining time savings while matching the best performance

Model	# Layers	**IWSLT-Dec2** **Train time** **until 95% max** (in hours)	**Max BLEU** (95%)	**# Layers**	**IWSLT-Dec6** **Train time** **until 95% max** (in hours)	**Max BLEU** (95%)	**# Layers**	**WMT-Dec1** **Train time** **until 95% max** (in hours)	**Max BLEU** (95%)
Transformer	6	0.647 ± 0.03	32.89 ± 0.04	6	0.642 ± 0.02	33.36 ± 0.03	12	3.788 ± 0.053	23.36 ± 0.06
	8	0.711 ± 0.05	33.04 ± 0.03	8	0.765 ± 0.03	33.41 ± 0.08	16	3.820 ± 0.072	23.41 ± 0.05
	10	0.808 ± 0.02	33.96 ± 0.08	10	0.898 ± 0.04	33.32 ± 0.07	24	5.262 ± 0.607	23.50 ± 0.03
	12	1.037 ± 0.03	33.07 ± 0.09	12	1.037 ± 0.03	33.07 ± 0.11	32	6.212 ± 0.232	23.81 ± 0.04
T Reservoir	6	0.569 ± 0.02	32.78 ± 0.03	6	0.599 ± 0.01	33.09 ± 0.05	12	3.563 ± 0.061	23.21 ± 0.04
	8	0.619 ± 0.04	33.12 ± 0.05	8	0.726 ± 0.02	33.38 ± 0.09	16	3.603 ± 0.056	23.80 ± 0.06
	10	0.729 ± 0.04	33.13 ± 0.07	10	0.738 ± 0.03	33.37 ± 0.04	24	4.923 ± 0.771	23.75 ± 0.02
	12	0.982 ± 0.02	33.03 ± 0.11	12	0.958 ± 0.01	33.46 ± 0.09	32	5.780 ± 0.214	23.71 ± 0.03
FFN Reservoir	6	0.521 ± 0.05	32.85 ± 0.02	6	0.594 ± 0.03	33.13 ± 0.04	12	3.417 ± 0.046	23.22 ± 0.07
	8	0.533 ± 0.03	33.84 ± 0.04	8	0.651 ± 0.04	33.36 ± 0.06	16	3.527 ± 0.063	23.54 ± 0.05
	10	0.614 ± 0.01	33.05 ± 0.08	10	0.627 ± 0.05	33.26 ± 0.03	24	4.197 ± 0.697	23.74 ± 0.06
	12	0.811 ± 0.02	33.26 ± 0.10	12	0.780 ± 0.02	33.46 ± 0.08	32	4.984 ± 0.321	23.82 ± 0.02
LayerDrop	6	0.837 ± 0.08	32.87 ± 0.05	6	0.706 ± 0.01	33.08 ± 0.03	12	3.912 ± 0.068	23.33 ± 0.08
	8	0.934 ± 0.07	33.12 ± 0.03	8	0.753 ± 0.04	33.14 ± 0.05	16	3.581 ± 0.076	23.17 ± 0.04
	10	0.901 ± 0.06	33.18 ± 0.02	10	0.691 ± 0.03	32.39 ± 0.05	24	4.875 ± 0.728	23.43 ± 0.07
	12	0.914 ± 0.01	32.33 ± 0.06	12	0.803 ± 0.02	32.94 ± 0.10	32	5.980 ± 0.219	22.97 ± 0.08

Table 6: Wall-clock time (averaged over multiple runs) for IWSLT/WMT for different model types and encoder depths. 95% Max BLEU is for validation.

Model	# Layers	**IWSLT-Dec2** **Train time** **until 99% max** (in hours)	**Max BLEU** (99%)	**# Layers**	**IWSLT-Dec6** **Train time** **until 99% max** (in hours)	**Max BLEU** (99%)	**# Layers**	**WMT-Dec1** **Train time** **until 99% max** (in hours)	**Max BLEU** (99%)
Transformer	6	1.454 ± 0.06	34.24 ± 0.05	6	1.297 ± 0.03	34.69 ± 0.05	12	9.961 ± 0.053	24.27 ± 0.04
	8	1.475 ± 0.09	34.32 ± 0.09	8	1.390 ± 0.02	34.75 ± 0.09	16	12.623 ± 0.072	24.35 ± 0.06
	10	1.526 ± 0.04	34.25 ± 0.04	10	1.622 ± 0.05	34.64 ± 0.03	24	13.412 ± 0.837	24.49 ± 0.07
	12	2.259 ± 0.07	34.24 ± 0.11	12	1.748 ± 0.01	34.66 ± 0.08	32	15.117 ± 0.232	24.56 ± 0.02
T Reservoir	6	1.257 ± 0.04	34.05 ± 0.09	6	1.291 ± 0.03	34.51 ± 0.10	12	8.314 ± 0.062	24.15 ± 0.06
	8	1.472 ± 0.06	34.47 ± 0.05	8	1.339 ± 0.03	34.80 ± 0.04	16	9.221 ± 0.073	24.41 ± 0.05
	10	1.530 ± 0.03	34.36 ± 0.02	10	1.419 ± 0.04	34.72 ± 0.03	24	10.413 ± 0.580	24.56 ± 0.03
	12	2.043 ± 0.05	34.53 ± 0.07	12	1.642 ± 0.02	34.87 ± 0.02	32	11.465 ± 0.227	24.49 ± 0.01
FFN Reservoir	6	1.138 ± 0.03	34.10 ± 0.13	6	1.169 ± 0.02	34.71 ± 0.09	12	7.407 ± 0.087	24.33 ± 0.08
	8	1.101 ± 0.07	34.32 ± 0.11	8	1.201 ± 0.03	34.79 ± 0.08	16	9.336 ± 0.036	24.42 ± 0.05
	10	1.281 ± 0.01	34.36 ± 0.03	10	1.276 ± 0.03	34.63 ± 0.03	24	9.978 ± 0.546	24.91 ± 0.07
	12	1.785 ± 0.03	34.42 ± 0.06	12	1.440 ± 0.01	34.87 ± 0.02	32	10.524 ± 0.341	24.96 ± 0.01
LayerDrop	6	1.363 ± 0.05	34.58 ± 0.14	6	1.253 ± 0.01	34.42 ± 0.10	12	8.372 ± 0.059	24.17 ± 0.04
	8	1.468 ± 0.03	34.50 ± 0.12	8	1.244 ± 0.04	34.44 ± 0.09	16	9.741 ± 0.043	23.93 ± 0.08
	10	1.678 ± 0.04	34.52 ± 0.07	10	1.343 ± 0.04	33.83 ± 0.06	24	10.145 ± 0.628	24.07 ± 0.09
	12	2.071 ± 0.02	33.45 ± 0.23	12	1.423 ± 0.02	33.97 ± 0.12	32	10.168 ± 0.329	23.81 ± 0.03

Table 7: Wall-clock time (averaged over multiple runs) saved for IWSLT/WMT for different model types and encoder depths. 99% Max BLEU is for validation.

of vanilla transformers for MNLI-m and SST2, respectively.

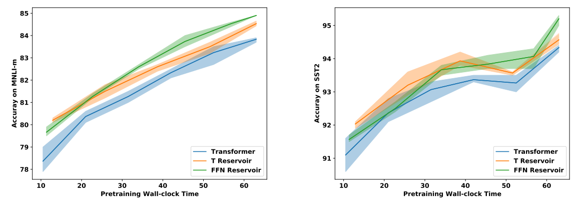

Figure 6: RoBERTa Reservoir Results, Pre-training versus downstream task plot for 12 layer RoBERTa. MNLI-m (left). SST-2 (right).

E Reservoir Results for Total Layers

Here we present the *shifted* Reservoir Results for IWSLT14, WMT16, Enwik8 and RoBERTa fine-tuning in Figure 8, 9, 10, 11, respectively. We show the same results also hold when it comes to replace normal transformer blocks with Reservoir blocks at least for MT.

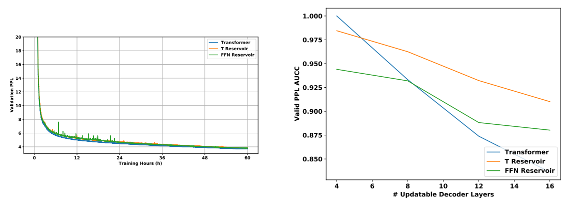

Figure 7: RoBERTa Reservoir Results, Training plot for 12 layer RoBERTa (left). AUCC result (right).

F Validation Plots

Here we present the validation plots for training a 8-layer encoder, 2-layer decoder model for IWSLT14, a 24-layer encoder, 1-layer decoder model for WMT16, a 48-layer decoder model for enwik8 and a 12-layer decoder model for RoBERTa for detailed steps to calculate the AUCC. It can be clearly observed that given the configurations from Section 3.1, all the models have converged. So when we compute the area under the convergence curve, this depicts the training efficiency of the model (basically time x performance) until convergence. Specifically, we set T

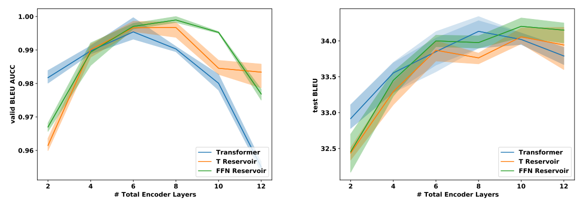

Figure 8: Validation BLEU AUCC and test BLEU for IWSLT (high is good). Comparison of regular transformer and reservoir transformer with FFN or Transformer reservoir layers added.

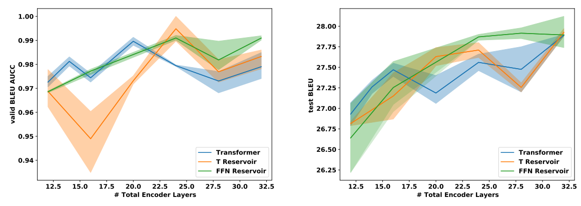

Figure 9: Validation BLEU AUCC and test BLEU for WMT (high is good). Comparison of regular transformer and reservoir transformer with FFN or Transformer reservoir layers added.

sufficiently high for computing the AUCC, which is 4h for IWSLT, 20h for WMT, 30h for enwik8 and 60h for RoBERTa pretraning. From the training plot in the appendix, we can see that each model has converged at that point. The Reservoir model in Figure 12 has 2 layers frozen for IWSLT14, 8 layers frozen for enwik8, and 4 layers frozen for WMT16 and RoBERTa.

G Backskipping

Figure 13 shows the BLUE curves for IWSLT comparing regular vs reservoir vs backskipped transformers, with the latter performing surprisingly well.

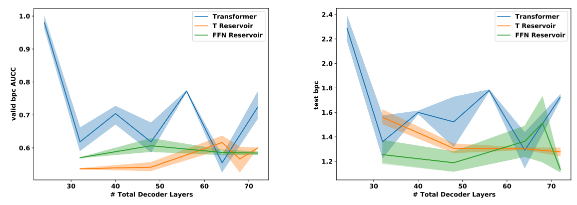

Figure 10: Validation BPC AUCC and test BPC on the enwik8 language modelling task (low is good). Comparison of regular and reservoir transformers for varying depths.

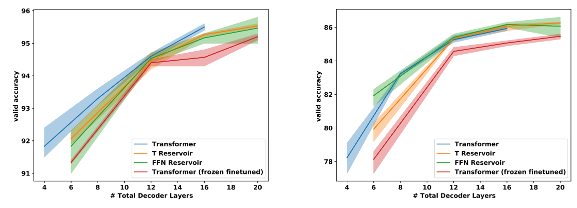

Figure 11: Downstream RoBERTa performance on SST-2 (left) and MultiNLI-matched (right).

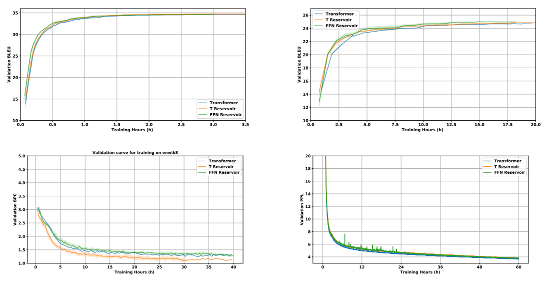

Figure 12: IWSLT with 2-layer decoder validation plot (upper left). WMT with 24-layer decoder validation plot (upper right). Enwik8 with 48-layer decoder validation plot (lower left). RoBERTa with 12-layer decoder validation plot (lower right).

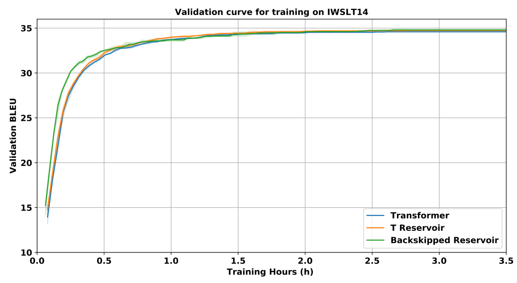

Figure 13: IWSLT comparison of the regular, reservoir and backskipped transformer architectures (encoder has 8 layers with 2 frozen, if any).