FNet: Mixing Tokens with Fourier Transforms

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, Santiago Ontañón
Google Research
{jamesleethorp, jainslie, ilyaeck, santiontanon}@google.com

Abstract

We show that Transformer encoder architectures can be sped up, with limited accuracy costs, by replacing the self-attention sublayers with simple linear transformations that "mix" input tokens. These linear mixers, along with standard nonlinearities in feed-forward layers, prove competent at modeling semantic relationships in several text classification tasks. Most surprisingly, we find that replacing the self-attention sublayer in a Transformer encoder with a standard, unparameterized Fourier Transform achieves 92-97% of the accuracy of BERT counterparts on the GLUE benchmark, but trains 80% faster on GPUs and 70% faster on TPUs at standard 512 input lengths. At longer input lengths, our FNet model is significantly faster: when compared to the "efficient" Transformers on the Long Range Arena benchmark, FNet matches the accuracy of the most accurate models, while outpacing the fastest models across all sequence lengths on GPUs (and across relatively shorter lengths on TPUs). Finally, FNet has a light memory footprint and is particularly efficient at smaller model sizes; for a fixed speed and accuracy budget, small FNet models outperform Transformer counterparts.[1]

1 Introduction

The Transformer architecture (Vaswani et al., 2017) has achieved rapid and widespread dominance in NLP. At its heart is a self-attention mechanism – an inductive bias that connects each token in the input through a relevance weighted basis of every other token. Each hidden unit is represented in the basis of the hidden units of the previous layer. Many papers have prodded and probed the Transformer, and in particular the attention sublayers, in an effort to better understand the architecture; see, for example, Tenney et al. (2019); Vig and Belinkov (2019); Clark et al. (2019); Voita et al. (2019). Although potentially limited in their effectiveness (Hewitt and Liang, 2019), these probes generally back the intuition that, by allowing higher order units to form out of compositions of the input, Transformer models can flexibly capture diverse syntactic and semantic relationships.

In this work, we investigate whether simpler token mixing mechanisms can wholly replace the relatively complicated self-attention layers in Transformer encoder architectures. We first replace the attention sublayer with two parameterized matrix multiplications – one mixing the sequence dimension and one mixing the hidden dimension. Seeing promising results in this simple linear mixing scheme, we further investigate the efficacy of faster, structured linear transformations. Surprisingly, we find that the Fourier Transform, despite having no parameters at all, achieves nearly the same performance as dense linear mixing and scales very efficiently to long inputs, especially on GPUs (owing to the Fast Fourier Transform algorithm). We call the resulting model FNet.

By replacing self-attention with linear transformations, we are able to reduce the complexity and memory footprint of the Transformer encoder. We show that FNet offers an excellent compromise between speed, memory footprint, and accuracy, achieving 92% and 97%, respectively, of the

[1]FNet code is available at `https://github.com/google-research/google-research/tree/master/f_net`.

Preprint. Under review.

accuracy of BERT-Base and BERT-Large on the GLUE benchmark (Wang et al., 2018), while training 80% faster on GPUs and 70% faster on TPUs. Speed gains aside, it is remarkable that an unparameterized (and therefore task independent) token mixing scheme, coupled with simple nonlinearities in feed-forward layers, is sufficient to model complex NLP tasks such as GLUE. We also show that an FNet hybrid model containing only two self-attention sublayers achieves 97% of the BERT-Base accuracy on GLUE (up from 92% of the pure FNet model), while still training 50% faster on GPUs and 40% faster on TPUs.

Furthermore, we demonstrate that FNet is competitive with all the "efficient" Transformers evaluated on the Long-Range Arena (LRA) benchmark (Tay et al., 2020c) – a suite of tasks designed to assess models on longer sequence length inputs. Specifically, we find that FNet achieves comparable accuracies to the most accurate efficient Transformer architectures but is faster. In fact, on GPUs, FNet is significantly faster at both training and inference than all of the evaluated Transformer architectures across all sequence lengths. On TPUs, FNet is faster at training and inference for relatively shorter sequence lengths (≤ 1024 and ≤ 512 tokens, respectively); for longer sequences, the only efficient Transformers that are faster than FNet on TPUs appear to be less accurate on the LRA benchmark. Finally, our experiments also suggest that FNet has a lighter memory footprint than all the efficient Transformer architectures, across all sequence lengths.

The rest of this paper is organized as follows. In Section 2, we situate our work in the current literature. We then review the basics of the discrete Fourier Transform (DFT) and introduce the FNet architecture in Section 3. In Section 4, we summarize the main results of this paper in the form of two sets of experimental results: First, in Section 4.1, we compare FNet, BERT (Devlin et al., 2018), and several simple baselines in a transfer learning setting – masked language modelling (MLM) and next sentence prediction (NSP) pre-training, followed by fine-tuning on the GLUE benchmark. Second, in Section 4.2, we show that FNet offers a better compromise between speed and accuracy than the efficient Transformers evaluated in the LRA benchmark (Tay et al., 2020c). We close in Section 5 with a summary of our results.

2 Related work

Our work continues a large literature of using Fourier Transforms in neural networks and connects with two broad themes of attention based models.

2.1 Fourier Transforms in neural networks

Fourier analysis features heavily in theoretical studies of neural networks. For example, universal approximation properties of neutral networks are typically investigated by showing that a particular class of neutral networks is dense in the space of Fourier series, which in turn are dense in the relevant Hilbert space; see, for example, (Cybenko, 1989; Barron, 1993).

In terms of practical applications, discrete Fourier Transforms (DFT), and in particular the Fast Fourier Transform (FFT), were used to tackle signal processing problems. For example, Minami et al. (1999); Gothwal et al. (2011); Mironovova and Bíla (2015) fit neural networks to FFTs of electrocardiogram signals of the heart to identify abnormal electrical activity. Similarly, Zhang et al. (2013) use FFTs of vibration signals as features for training neural networks to classify fault and predict degradation in manufacturing systems. More recent work by Li et al. (2020) uses DFTs in neural networks to evolve solutions of Partial Differential Equations.

Because ordinary multiplication in the frequency domain corresponds to a convolution in the time domain, FFTs have been deployed in Convolutional Neural Networks (CNNs) to speed up computations (El-Bakry and Zhao, 2004; Mathieu et al., 2014; Highlander and Rodriguez, 2016; Pratt et al., 2017; Lin et al., 2018; Chitsaz et al., 2020). DFTs have also been applied to Recurrent Neural Networks (RNNs), often in an effort to speed up training and reduce exploding and vanishing gradients (Koplon and Sontag, 1997; Zhang and Chan, 2000; Zhang et al., 2018). Additionally, DFTs have been used to approximate dense, linear layers in neural networks to reduce the parameter count and computational complexity (Cheng et al., 2015; Moczulski et al., 2015; Sindhwani et al., 2015).

DFTs have been used indirectly in several Transformer works. The Performer (Choromanski et al., 2020a) linearizes the complexity of the Transformer self-attention mechanism by leveraging random Fourier features. Specifically, Choromanski et al. (2020a) first represent the (attention) softmax kernel in terms of the Gaussian kernel, and then approximate the latter by sampling from the Fourier Transform (Rahimi and Recht, 2008). In our work, rather than approximating the self-attention kernel with sampled Fourier features, we replace the entire self-attention sublayer with a DFT. Tamkin et al. (2020) experiment with spectral filters in the activations of neurons in BERT models, showing that the filtered embeddings perform well in different tasks (word-level, sentence-level or document-level), depending on which frequency scales are filtered. Finally, through personal communication, we were alerted to concurrent, unpublished work (Backurs et al., 2021) that describes an FFT based neural model that is very similar to FNet.

2.2 The power of attention to model semantic relationships

Since attention (Bahdanau et al., 2014) and then the Transformer (Vaswani et al., 2017) architecture were first introduced, attention models have achieved state of the art results across virtually all NLP tasks and even some image tasks (Dosovitskiy et al., 2020). This success is generally attributed to the flexibility and capacity of attention models. Although some works (Ramsauer et al., 2020) have endeavoured to gain a deeper theoretical understanding of self-attention, the pervasive (and arguably, very appealing) intuition is that the success of attention models derives from the token-dependent attention patterns in different layers; see, for example, (Tenney et al., 2019). However, it is natural to ask: Do we really need the flexibility, and associated cost, of attention?

Tay et al. (2020a) empirically investigated the importance of the dot product operation in the self-attention mechanism in their Synthesizer (related to our "Linear" baseline below). They find that learning token-dependent attention weights from (dot-product) token-token interactions is powerful, but not necessarily crucial for realizing accurate NLP models. You et al. (2020) replace attention weights in the Transformer encoder and decoder with unparameterized Gaussian distributions, showing minimal degradation in NMT BLEU scores if they retain learnable cross-attention weights. Similarly, Raganato et al. (2020) find little to no accuracy degradation on NMT tasks when they replace all but one of the attention heads of each attention sublayer in the *encoder* with fixed, non-learnable positional patterns. Finally, Tolstikhin et al. (2021) present MLP-Mixer, where the attention is replaced by MLPs, with limited to no performance losses in image classification tasks. In contrast to MLP-Mixer, FNet has no learnable parameters that mix along the spatial dimension.

2.3 Efficient Transformers and long sequence models

The standard self-attention mechanism (Vaswani et al., 2017) has a quadratic time and memory bottleneck with respect to sequence length. This limits its applicability in text tasks involving long range dependencies, character-level modelling, speech processing, image and video processing.

Most efforts to improve attention efficiency are based on sparsifying the attention matrix. Tay et al. (2020d) survey many of the recent efficient attention works, which they broadly categorize as: (1) Data independent sparsity approaches that use fixed attention patterns (Child et al., 2019; Qiu et al., 2019; Parmar et al., 2018; Beltagy et al., 2020; Ainslie et al., 2020; Zaheer et al., 2020); (2) Data dependent sparsity approaches that dynamically compress the attention matrix (Wang et al., 2020; Tay et al., 2020b,a; Kitaev et al., 2020; Roy et al., 2021; Vyas et al., 2020; Liu et al., 2018); and (3) Linearization of the attention matrix using kernel decompositions (Katharopoulos et al., 2020; Choromanski et al., 2020b; Peng et al., 2021). Several of these works achieve $\mathcal{O}(N\sqrt{N})$ or even $\mathcal{O}(N)$ theoretical complexity. However, the scaling constants hidden by this notation can be large. For example, in models such as Longformer (Beltagy et al., 2020), ETC (Ainslie et al., 2020), and BigBird (Zaheer et al., 2020), attention is $\mathcal{O}(N)$ as a function of the input length, but quadratic in the number of "global tokens"; the latter must be sufficiently large to ensure good performance.

The Long-Range Arena benchmark (Tay et al., 2020c) attempts to compare many of the above "efficient" Transformers in a series of tasks requiring long range dependencies, finding that the Performer (Choromanski et al., 2020b), Linear Transformer (Katharopoulos et al., 2020), Linformer (Wang et al., 2020), and Image Transformer (Local Attention) (Parmar et al., 2018) were the fastest on 4×4 TPU v3 chips and had the lowest peak memory usages per device.[2] The latter memory footprint results are important as empirical studies have shown that Transformer architectures are often memory-bound (Ivanov et al., 2020; Shazeer, 2019). Instead, in this paper we completely replace self-attention with a different mixing, namely the Fourier Transform, which offers: (1)

[2]Although, as the authors state, the results may be somewhat sensitive to implementation.

performance (the DFT is extremely fast on TPUs and GPUs), (2) reduced model size (no learnable parameters), and (3) simplicity.

Finally, we note that there are many optimizations, beyond seeking more efficient attention mechanisms, that are used to improve Transformer performance from fusing operations and sharing key-value-query matrices to general pruning, quantization, and distillation; see, for example, (Kim and Awadalla, 2020; Shleifer and Rush, 2020). In a similar vein, we ignore several improvements to the original BERT model (Devlin et al., 2018), such as Adafactor (instead of Adam) for optimization, inverse square root learning schedule, and pre-layer (as opposed to post-layer) normalization (Narang et al., 2021). We consider these directions orthogonal to this work, wherein we compare a vanilla Transformer with a vanilla FNet in an effort to investigate different token mixing mechanisms.

3 Model

3.1 Background on the Discrete Fourier Transform

The Fourier Transform decomposes a function into its constituent frequencies. Given a sequence $\{x_n\}$ with $n \in [0, N-1]$, the discrete Fourier Transform (DFT) is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}, \quad 0 \leq k \leq N-1. \tag{1}$$

For each k, the DFT generates a new representation X_k as a sum of all of the original input tokens x_n, with so-called "twiddle factors". There are two primary approaches to computing the DFT: the Fast Fourier Transform (FFT) and matrix multiplication. The standard FFT algorithm is the Cooley–Tukey algorithm (Cooley and Tukey, 1965; Frigo and Johnson, 2005), which recursively re-expresses the DFT of a sequence of length $N = N_1 N_2$ in terms of N_1 smaller DFTs of sizes N_2 to reduce the computation time to $\mathcal{O}(N \log N)$.

An alternative approach is to simply apply the DFT matrix to the input sequence. The DFT matrix, W, is a Vandermonde matrix for the roots of unity up to a normalization factor:

$$W_{nk} = \left(e^{-\frac{2\pi i}{N} nk} / \sqrt{N}\right), \tag{2}$$

where $n, k = 0, \ldots, N-1$. This matrix multiplication is an $\mathcal{O}(N^2)$ operation, which has higher asymptotic complexity than the FFT, but is faster for relatively shorter sequences on TPUs.

3.2 FNet architecture

FNet is an attention-free Transformer architecture, wherein each layer consists of a Fourier mixing sublayer followed by a feed-forward sublayer. The architecture is shown in Figure 1. Essentially, we replace the self-attention sublayer of each Transformer encoder layer with a Fourier sublayer, which applies a 2D DFT to its (sequence length, hidden dimension) embedding input – one 1D DFT along the sequence dimension, $\mathcal{F}_{\text{seq}}$, and one 1D DFT along the hidden dimension, $\mathcal{F}_{\text{h}}$:[3]

$$y = \Re\left(\mathcal{F}_{\text{seq}}\left(\mathcal{F}_{\text{h}}(x)\right)\right). \tag{3}$$

As indicated by Equation (3), we only keep the real part of the result; hence, we do not need to modify the (nonlinear) feed-forward sublayers or output layers to handle complex numbers. We found that FNet obtained the best results when the real part of the total transformation was only extracted at the end of the Fourier sublayer; that is, after applying both $\mathcal{F}_{\text{seq}}$ and $\mathcal{F}_{\text{h}}$.[4]

The simplest interpretation for the Fourier Transform is as a particularly effective mechanism for mixing tokens, which evidently provides the feed-forward sublayers sufficient access to all tokens. Because of the duality of the Fourier Transform, we can also view each alternating encoder block as applying alternating Fourier and inverse Fourier Transforms, transforming the input back and

[3]The relative ordering of $\mathcal{F}_{\text{seq}}$ and $\mathcal{F}_{\text{h}}$ in Equation (3) is immaterial because the two 1D DFTs commute.

[4]FNet was less accurate and less stable during training if only the real part of the Fourier Transform was used; i.e. replacing Equation (3) with $\Re(\mathcal{F}_{\text{seq}}(\Re(\mathcal{F}_{\text{h}}(x))))$. See Appendix A.3 for other transformations that we experimented with.

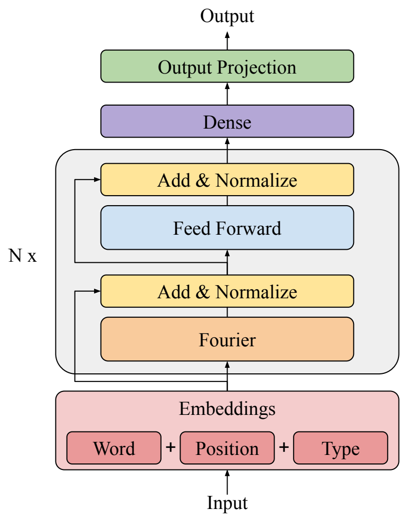

Figure 1: FNet encoder architecture with N encoder blocks.

forth between the "time" and frequency domain. Because multiplying by the feed-forward sublayer coefficients in the frequency domain is equivalent to convolving (with a related set of coefficients) in the time domain, FNet can be thought of as alternating between multiplications and (large kernel) convolutions.[5]

We use the same embedding layers as in Devlin et al. (2018); namely, we combine the word embeddings, absolute position embeddings of the tokens and type embeddings of the sentences.

3.3 Implementation

Empirically, we found that: On GPUs: the FFT is faster than matrix multiplications for all sequence lengths we consider ($512 - 8192$ tokens), and on TPUs: for relatively shorter sequences (≤ 4096 tokens), it is faster to pre-compute the DFT matrix and then compute the DFT through matrix multiplications than using the FFT; for longer sequences, the FFT is faster. As a result, our GPU FNet implementation always uses the FFT, while our TPU implementation computes the 2D DFT using matrix multiplications for sequences up to lengths of 4096 and the FFT for longer lengths. Presumably the GPU vs TPU difference is primarily a result of two factors: (1) TPUs are even more highly optimized for matrix multiplications than GPUs, and (2) GPUs offer a more efficient FFT implementation than TPUs. We suspect that FNet will only become more performant on TPUs as the TPU implementation of the FFT improves. Our model uses JAX via the Flax framework[6]. Core model code is given in Appendix A.6 and the full source code is available online[7].

4 Results

4.1 Transfer learning

We compare the FNet and Transformer architectures in a common transfer learning setting. For a fuller picture, we compare multiple models (see Table 2 for their parameter counts in their "Base" configuration):

- BERT-Base: a Transformer encoder model.
- FNet encoder: we replace every self-attention sublayer with a Fourier sublayer.

[5]This is merely an intuition; the reality is more complicated due to the presence of residual connections and since the transformation in Equation (3) is no longer invertible if we only use the real component.

[6]`https://github.com/google/flax`

[7]`https://github.com/google-research/google-research/tree/master/f_net`

- Linear encoder: we replace each self-attention sublayer with a two learnable, dense, linear sublayers, one applied to the hidden dimension and one applied to the sequence dimension.
- Random encoder: we replace each self-attention sublayer with a two constant random matrices, one applied to the hidden dimension and one applied to the sequence dimension.
- Feed Forward-only (FF-only) encoder: we remove the self-attention sublayer from the Transformer layers; this model has no token mixing.

Despite its simplicity, the Linear baseline turns out to be surprisingly accurate and fast, especially on TPUs. Our Linear model is similar to the MLP-Mixer (Tolstikhin et al., 2021) (for vision) and also the Random Synthesizer (Tay et al., 2020a), but simplifies the latter model further by removing the multiple heads and softmax projections, resulting in just two matrix multiplications in the mixing sublayer. In our limited cursory experiments, we found no major accuracy benefits from the softmax or multiple head projections, although they did result in slightly decreased training speeds.

The 2D DFT of FNet can be represented by two complex matrix multiplications. So, although the Linear encoder cannot exactly recreate the DFT (due to real-valued dense layers), it is reasonable to expect the Linear encoder to be a more flexible model. As we shall demonstrate, the Linear model can outperform FNet on certain tasks but has several drawbacks relative to FNet: it is slower (particularly on GPUs), has a much larger memory footprint, and is often more difficult to train due to gradient blow up (especially in its larger configuration).

We adopt the same fixed "Base" and "Large" model and learning and training configurations as for the original BERT (Devlin et al., 2018), except that we pre-train on the much larger C4 dataset (Raffel et al., 2019) and use a 32000 SentencePiece vocabulary model (Kudo and Richardson, 2018) (see Appendix A.1 for full pre-training details). For fine-tuning on the GLUE benchmark (Wang et al., 2018), we found that different BERT runs with the same base learning rate could yield slightly different results. Consequently, for the Base (Large) models, we performed 3 (6) trials, respectively, for each base learning rate and reported the best result across all experiments. This reflects our observation that BERT-Large was less stable than BERT-Base, as noted in Devlin et al. (2018).

We report the results for the best base learning rate (no early stopping) on the GLUE Validation split in Table 1.[8] For Base models, results mirror the pre-training metrics (see Appendix A.1): BERT performs best. FNet and the Linear model both underperform BERT by $7.5 - 8\%$. Referring to the right-hand side of Table 2, we see that although less accurate, FNet trains significantly faster than BERT – 80% faster on GPUs and 70% faster on TPUs. Measured in isolation, the Fourier sublayers perform forward and backward passes an order of magnitude faster than the self-attention sublayers (see Appendix A.4 for details), but FNet's overall training speed is impeded by the feed-forward sublayers that all models share.

Returning to Table 1: the FF-only model severely underperforms all other models: as expected, token mixing is critical to the expressivity of the model. For example, 50% accuracy scores on the binary classification tasks (QNLI, SST-2, RTE), indicate that the model fails to learn the tasks. The weak accuracy of the Random model suggests that not just any mixing will do; rather, a structured mixing is required. We also include metrics from a hybrid FNet attention model. In the hybrid model, we replace the final two Fourier sublayers of FNet with self-attention sublayers.[9] With the addition of just two self-attention sublayers, the hybrid FNet model achieves 97% of BERT's accuracy with only limited speed degradation (see Table 2).

Interestingly, the gap between BERT and FNet shrinks to just 3% for Large models; this is likely due to FNet-Large being more stable during training than BERT-Large.[10] The Linear-Large model severely underperforms its Base counterpart on GLUE benchmark due to training instabilities. We generally found that the Linear model and BERT were less stable than the models with no parameters in their mixing sublayers, namely the FNet, Random and FF-only models.

The speed vs MLM accuracy curve for GPU (8 V100 chips) pre-training is shown in Figure 2 (see Appendix A.2 for TPU results). Both TPU and GPU models are trained for 1 million steps as in

[8]WNLI is excluded in Devlin et al. (2018). BERT's accuracy on WNLI is below baseline, unless a special training recipe is used. See also (12) in `https://gluebenchmark.com/faq`.

[9]Other configurations are possible, but we generally found that replacing the final layers worked best.

[10]Devlin et al. (2018) obtain a roughly 2.5 average point boost on the *Test* split going from BERT-Base to BERT-Large. We only see a roughly 1.5 boost on the *Validation* split, which may be due to reduced headroom.

Table 1: GLUE Validation results on TPUs, after finetuning on respective tasks. We report the mean of accuracy and F1 scores for QQP and MRPC, Spearman correlations for STS-B and accuracy scores for all other tasks. The MNLI metrics are reported by the match/mismatch splits. Average scores exclude any failure cases. After controlling for batch size and training steps, the GPU metrics (not shown) are similar.

Model	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
BERT-Base	**84/81**	**87**	**91**	93	73	**89**	**83**	**69**	**83.3**
Linear-Base	74/75	84	80	94	67	67	83	69	77.0
FNet-Base	72/73	83	80	**95**	69	79	76	63	76.7
Random-Base	51/50	70	61	76	67	4	73	57	56.6
FF-only-Base	34/35	31	52	48	67	FAIL	73	54	49.3
FNet-Hybrid-Base	78/79	85	88	94	**76**	86	79	60	80.6
BERT-Large	**88/88**	**88**	**92**	**95**	71	**88**	86	66	**84.7**
Linear-Large	35/36	84	80	79	67	24	73	60	59.8
FNet-Large	78/76	85	85	94	**78**	84	**88**	**69**	81.9

Table 2: Computational characteristics of models. Left: number of mixing layer operations (forward pass) and learnable parameters (ignoring the output projection layers) for Base models. n is the sequence length and d_h is the model hidden dimension. The mixing layer operations are given on a per layer basis. Right: Pre-training speeds in milliseconds per pre-training step with batch sizes of 64 on GPU (8 V100 chips) and 256 on TPU (4×4 v3 chips).

Model	Mixing layer ops	Params
BERT	$2n^2d_h + 4nd_h^2$	111M
Linear	$n^2d_h + nd_h^2$	93M
FNet (mat)	$n^2d_h + nd_h^2$	83M
FNet (FFT)	$nd_h \log(n) + nd_h \log(d_h)$	83M
Random	$n^2d_h + nd_h^2$	83M
FF-only	0	83M

Model	GPU	TPU
BERT-Base	305	213
Linear-Base	199 (1.5x)	149 (1.4x)
FNet-Base	169 (1.8x)	128 (1.7x)
Random-Base	182 (1.7x)	130 (1.6x)
FF-only-Base	**162 (1.9x)**	**118 (1.8x)**
FNet-Hybrid-Base	198 (1.5x)	149 (1.4x)
BERT-Large	OOM	503
Linear-Large	592	397 (1.3x)
FNet-Large	**511**	**275 (1.8x)**

Table 3: Pre-training model sizes (ignoring output projection layers). As in Turc et al. (2019), for all models, we fix the feed-forward size to $4d_h$ and the number of self-attention heads to $d_h/64$. Smaller architectures have a similar number of parameters across all models because the majority of parameters are in the embedding layers. Each FNet-Hybrid model contains 2 self-attention sublayers. We therefore exclude FNet-Hybrid models with only 2 total layers.

Dimensions		Parameters (millions)			
d_h	Layers	BERT	Linear	FNet	FNet-Hybrid
768	12	111	93	83	88
512	12	55	49	42	44
512	8	42	38	34	36
256	8	15	15	13	13
512	4	30	28	26	28
256	4	12	12	11	11
256	2	10	10	10	-
128	2	5	5	4	-

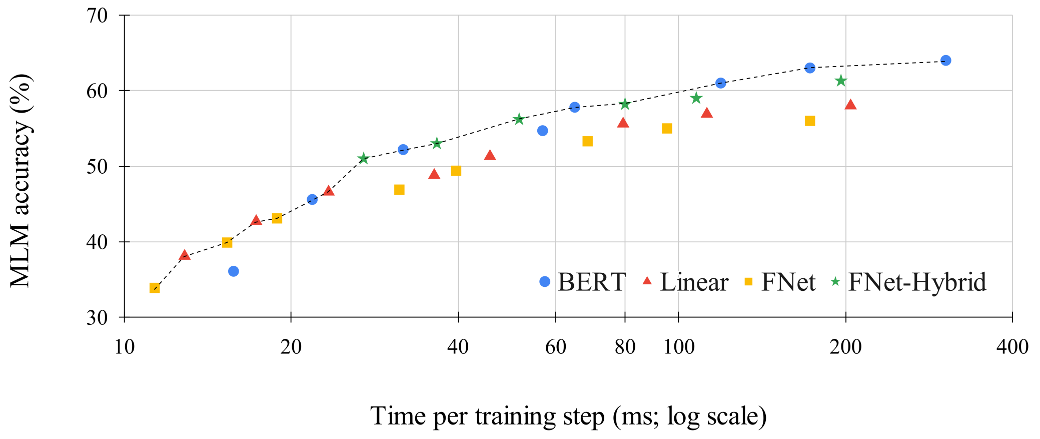

Figure 2: Speed-accuracy trade-offs for GPU pre-training. The dashed line shows the Pareto efficiency frontier, indicating the best speed-accuracy trade-offs. For smaller models (which have faster training speeds; left-hand side of figure), the FNet (yellow squares) and Linear (red triangles) models define the frontier, while for larger models (which have slower training speeds; right-hand side of figure), BERT (blue circles) and FNet-Hybrid (green stars) define the frontier.

Table 4: Accuracy results on the Long-Range Arena (LRA) benchmark, obtained on TPUs as in Tay et al. (2020c). Asterisked results are quoted from Tay et al. (2020c). Average scores do not include the Path-X task, which all models fail, but for different reasons: Transformer fails due to memory limits, whereas the other models perform no better than chance on the task.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg.
Transformer (ours)	36.06	61.54	**59.67**	41.51	80.38	OOM	**55.83**
Linear (ours)	33.75	53.35	58.95	41.04	**83.69**	FAIL	54.16
FNet (ours)	35.33	65.11	59.61	38.67	77.80	FAIL	55.30
Transformer (*)	**36.37**	64.27	57.46	42.44	71.40	OOM	54.39
Performer (*)	18.01	**65.40**	53.82	**42.77**	77.05	FAIL	51.41

Devlin et al. (2018). Motivated by the models considered in Turc et al. (2019), we evaluated several model sizes; see Table 3. We found that the smaller model architectures benefited from larger learning rates, so, for all models, we choose the best results from two base learning rates: 10^{-3} and 10^{-4}.

The GPU (Figure 2), and TPU (Figure 3 in the Appendix) results display the same trends. For larger, slower models, BERT and FNet-Hybrid define the Pareto speed-accuracy efficiency frontier. For smaller, faster models, FNet and the Linear model define the efficiency frontier.

4.2 Long-Range Arena (LRA) benchmark

Of the efficient Transformers evaluated on LRA benchmark by Tay et al. (2020c), their results suggest that (1) the vanilla Transformer is (by a small margin) the second most accurate model, and (2) the Performer (Choromanski et al., 2020b) is the fastest model. We benchmark FNet's accuracy against both of these models using Tay et al. (2020c)'s codebase and running on the same hardware (4×4 TPU v3 chips); the results are shown in Table 4.[11] To ensure a fair comparison, we also report the results of our own experiments for the vanilla Transformer (see Appendix A.5 for details).

Table 4 suggests that, in aggregate, the (vanilla) Transformer and FNet obtain very comparable results. Given that the Transformer is the second most accurate model evaluated by Tay et al. (2020c) and that the relative differences in the average accuracy scores within Table 4 is small, our results suggest that FNet is competitive with the most accurate of the efficient Transformers.

[11]The "Linear" model included in Tables 4-6 is the baseline model introduced in Section 4.1 and not the "Linear Transformer" from Katharopoulos et al. (2020).

Table 5: GPU training on the Long-Range Text classification task, for sequence lengths up to 8192. Left: training speeds (in steps per second; larger is better), with speed-up multipliers relative to the Transformer given in parentheses. Right: peak memory usage (in GB; smaller is better).

	Training Speed (steps/s)					Peak Memory Usage (GB)				
Seq. length	512	1024	2048	4096	8192	512	1024	2048	4096	8192
Transformer	21	10	4	OOM	OOM	1.6	4.0	12.2	OOM	OOM
Linear	34 (1.6x)	19 (1.8x)	9 (2.0x)	4	1	0.9	1.6	2.8	6.9	OOM
FNet (FFT)	**43 (2.0x)**	**24 (2.3x)**	**14 (3.2x)**	**7**	**4**	**0.8**	**1.3**	**2.2**	**3.9**	**7.4**
Performer	28 (1.3x)	15 (1.5x)	9 (1.9x)	4	2	1.1	1.9	3.1	5.5	10.4

Table 6: GPU inference speeds on the Long-Range Arena Text classification task (in milliseconds per batch; smaller is better). Speed up multipliers relative to the Transformer are given in parentheses.

Seq. length	512	1024	2048	4096	8192	16384
Transformer	12	28	76	244	OOM	OOM
Linear	9 (1.4x)	14 (2.0x)	30 (2.6x)	72 (3.4x)	208	OOM
FNet (FFT)	**8 (1.5x)**	**12 (2.3x)**	**23 (3.4x)**	**43 (5.7x)**	**83**	**164**
Performer	11 (1.2x)	17 (1.6x)	32 (2.4x)	60 (4.0x)	116	238

Turning to efficiency, in Table 5, we provide training speed and memory usage statistics from our experiments on GPUs (8 V100 chips); see Appendix A.2 for results on TPUs. We perform a sweep over sequence lengths $\{512, 1024, 2048, 4096, 8192\}$ instead of just $\{1K, 2K, 3K, 4K\}$. On GPUs, FNet is much faster than all other models across all sequence lengths, due to the highly efficient FFT implementation on GPUs. Table 5 also indicates that FNet has a lighter memory footprint than all of the efficient Transformers (this holds for both GPUs and TPUs; see extended results in Appendix A.2). This is partly because FNet has no learnable parameters in its mixing sublayer, but also due to the FFT's efficiency, especially at longer sequence lengths. Lastly, Table 6 shows that training speed gains generally carry over to inference gains, with similar behaviors on GPUs vs TPUs (see Appendix A.2 for detailed TPU results).

5 Conclusions

In this work, we studied simplified token mixing modules for Transformer-like encoder architectures, making several contributions. First, we showed that simple, linear mixing transformations, along with the nonlinearities in feed-forward layers, can competently model diverse semantic relationships in text. Second, we introduced FNet, a Transformer-like model wherein the self-attention sublayer is replaced by an unparameterized Fourier Transform. FNets achieve 92 and 97% of their respective BERT-Base and BERT-Large counterparts' accuracy on the GLUE benchmark, but train 80% faster on GPUs and 70% faster on TPUs. Third, because of its favorable scaling properties, FNet is very competitive with the "efficient" Transformers evaluated on the Long-Range Arena benchmark, matching the accuracy of the most accurate models while being much faster. Specifically, on GPUs, FNet was significantly faster at both training and inference than all efficient Transformers across all sequence lengths; on TPUs, FNet is faster at training and inference for relatively shorter sequence lengths (≤ 1024 and ≤ 512 tokens, respectively). Our experiments also show that FNet has a lighter memory footprint than any of the efficient Transformer architectures evaluated by Tay et al. (2020c), across all sequence lengths. Finally, we demonstrated that, for a fixed speed and accuracy budget, small FNet encoders outperform small Transformer models.

Our work highlights the potential of linear units as a drop-in replacement for the attention mechanism in text classification tasks. We found the Fourier Transform to be a particularly effective mixing mechanism, in part due to the highly efficient FFT. Remarkably, this unparameterized mixing mechanism can yield relatively competitive models. On a practical note, we only performed a cursory survey of other linear transformations (see Appendix A.3), and additional fast alternatives may well be worth exploring – as is the relation to convolutions.

Given the speed and accuracy advantages of smaller FNet models relative to Transformers, we suspect that FNet will be effective as a lightweight, distilled student model deployed in resource-constrained settings such as production services or on edge devices. The need for such lightweight serving models is only forecast to grow given the surging interest in giant models (Raffel et al., 2019; Brown et al., 2020; Lepikhin et al., 2020). A natural avenue to explore in this regard will be using knowledge distillation on small FNet models from larger Transformer teacher models, following approaches established by Sanh et al. (2019); Jiao et al. (2019); Turc et al. (2019) and others.

Another aspect of interest and worthy of further study is hybrid FNet-attention models. We found that adding only a few self-attention sublayers to FNet offers a simple way to trade off speed for accuracy. Specifically, replacing the final two Fourier sublayers with self-attention provided 97% of BERT's accuracy on the GLUE benchmark with a limited penalty in speed.

Throughout this work we have restricted our attention to encoders. FNet decoders can be designed using causally masked discrete Fourier Transform matrices. However, in settings where the FFT is preferable, a lower level implementation would be required to introduce causal masking. Lastly, designing the equivalent of encoder-decoder cross-attention remains an avenue for future work, as evidence suggests that it is crucial to decoder performance (You et al., 2020).

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)*, pages 268–284.

Arturs Backurs, Mingda Chen, and Kevin Gimpel. 2021. A note on more efficient architectures for nlp. *http://www.mit.edu/~backurs/NLP.pdf*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Andrew R Barron. 1993. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945.

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

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*.

Yu Cheng, X Yu Felix, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2857–2865. IEEE.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Kamran Chitsaz, Mohsen Hajabdollahi, Nader Karimi, Shadrokh Samavi, and Shahram Shirani. 2020. Acceleration of convolutional neural network using fft-based split convolutions. *arXiv preprint arXiv:2003.12621*.

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

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020b. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301.

George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

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*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Hazem M El-Bakry and Qiangfu Zhao. 2004. Fast object/face detection using neural networks and fast fourier transform. *International Journal of Signal Processing*, 1(3):182–187.

Matteo Frigo and Steven G Johnson. 2005. The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231.

Himanshu Gothwal, Silky Kedawat, Rajesh Kumar, et al. 2011. Cardiac arrhythmias detection in an ecg beat signal using fast fourier transform and artificial neural network. *Journal of Biomedical Science and Engineering*, 4(04):289.

John Hewitt and Percy Liang. 2019. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, Hong Kong, China. Association for Computational Linguistics.

Tyler Highlander and Andres Rodriguez. 2016. Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add. *arXiv preprint arXiv:1601.06815*.

Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. 2020. Data movement is all you need: A case study of transformer networks. *arXiv preprint arXiv:2007.00072*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR.

Young Jin Kim and Hany Hassan Awadalla. 2020. Fastformers: Highly efficient transformer models for natural language understanding. *arXiv preprint arXiv:2010.13382*.

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

Renée Koplon and Eduardo D Sontag. 1997. Using fourier-neural recurrent networks to fit sequential input/output data. *Neurocomputing*, 15(3-4):225–248.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Henry O. Kunz. 1979. On the equivalence between one-dimensional discrete walsh-hadamard and multidimensional discrete fourier transforms. *IEEE Computer Architecture Letters*, 28(03):267–268.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2020. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.

Sheng Lin, Ning Liu, Mahdi Nazemi, Hongjia Li, Caiwen Ding, Yanzhi Wang, and Massoud Pedram. 2018. Fft-based deep learning deployment in embedded systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1045–1050. IEEE.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*.

Michael Mathieu, Mikael Henaff, and Yann LeCun. 2014. Fast training of convolutional networks through ffts: International conference on learning representations (iclr2014), cbls, april 2014. In *2nd International Conference on Learning Representations, ICLR 2014*.

Kei-ichiro Minami, Hiroshi Nakajima, and Takeshi Toyoshima. 1999. Real-time discrimination of ventricular tachyarrhythmia with fourier-transform neural network. *IEEE transactions on Biomedical Engineering*, 46(2):179–185.

Martina Mironovova and Jirí Bíla. 2015. Fast fourier transform for feature extraction and neural network for classification of electrocardiogram signals. In *2015 Fourth International Conference on Future Generation Communication Technology (FGCT)*, pages 1–6. IEEE.

Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. 2015. Acdc: A structured efficient linear layer. *arXiv preprint arXiv:1511.05946*.

Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, et al. 2021. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. 2021. Random feature attention. *arXiv preprint arXiv:2103.02143*.

Harry Pratt, Bryan Williams, Frans Coenen, and Yalin Zheng. 2017. Fcnn: Fourier convolutional neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 786–798. Springer.

Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. 2019. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann. 2020. Fixed encoder self-attention patterns in transformer-based machine translation. *arXiv preprint arXiv:2002.10260*.

Ali Rahimi and Benjamin Recht. 2008. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, et al. 2020. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Sam Shleifer and Alexander M Rush. 2020. Pre-trained summarization distillation. *arXiv preprint arXiv:2010.13002*.

Vikas Sindhwani, Tara N Sainath, and Sanjiv Kumar. 2015. Structured transforms for small-footprint deep learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 3088–3096.

Alex Tamkin, Dan Jurafsky, and Noah Goodman. 2020. Language through a prism: A spectral approach for multiscale language representations. In *Advances in Neural Information Processing Systems*, volume 33, pages 5492–5504. Curran Associates, Inc.

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, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020b. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pages 9438–9447. PMLR.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020c. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*.

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

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Jesse Vig and Yonatan Belinkov. 2019. Analyzing the structure of attention in a transformer language model. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 63–76, Florence, Italy. Association for Computational Linguistics.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. 2020. Fast transformers with clustered attention. *arXiv preprint arXiv:2007.04825*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

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

Weiqiu You, Simeng Sun, and Mohit Iyyer. 2020. Hard-coded gaussian attention for neural machine translation. *arXiv preprint arXiv:2005.00742*.

Table 7: Loss and accuracy pre-training metrics on TPUs. The GPU metrics are very similar.

	Loss			Accuracy	
Model	Total	MLM	NSP	MLM	NSP
BERT-Base	**1.76**	**1.48**	**0.28**	**0.68**	**0.86**
Linear-Base	2.12	1.78	0.35	0.62	0.83
FNet-Base	2.45	2.06	0.40	0.58	0.80
Random-Base	5.02	4.48	0.55	0.26	0.70
FF-only-Base	7.54	6.85	0.69	0.13	0.50
FNet-Hybrid-Base	2.13	1.79	0.34	0.63	0.84
BERT-Large	**1.49**	**1.23**	**0.25**	**0.72**	**0.88**
Linear-Large	1.91	1.60	0.31	0.65	0.85
FNet-Large	2.11	1.75	0.36	0.63	0.82

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*.

Jiong Zhang, Yibo Lin, Zhao Song, and Inderjit Dhillon. 2018. Learning long term dependencies via fourier recurrent units. In *International Conference on Machine Learning*, pages 5815–5823. PMLR.

Y Zhang and Lai-Wan Chan. 2000. Forenet: fourier recurrent networks for time series prediction. In *7th International Conference on Neural Information Processing (ICONIP 2000)*, pages 576–582.

Zhenyou Zhang, Yi Wang, and Kesheng Wang. 2013. Fault diagnosis and prognosis using wavelet packet decomposition, fourier transform and artificial neural network. *Journal of Intelligent Manufacturing*, 24(6):1213–1227.

A Appendices

A.1 Pre-training details

We adopt the same fixed "Base" and "Large" model and learning configurations as for the original BERT (Devlin et al., 2018). We train on the much larger C4 dataset (Raffel et al., 2019) and use a 32000 SentencePiece vocabulary model (Kudo and Richardson, 2018) trained on a 100 million sentence subset of C4. Our TPU experiments use a batch size of 256 as in Devlin et al. (2018) and are each run on 4×4 TPU v3 chips. Our GPU experiments use a smaller batch size of 64 and are run on 8 V100 chips. Because the training configuration is lifted from Devlin et al. (2018), it may be slightly biased towards the BERT attention model.

Tables 7 summarize the pre-training metrics for the different models; the pre-training speeds are shown in Table 2 in the main text. Although they have weaker accuracy metrics, the Linear model and FNet train nearly 80% faster than BERT on GPUs, and 70% faster on TPUs (see Table 2). We also find that the three models with no learnable parameters in their mixing layer, namely FNet, the Random model and the FF-only model, are the most stable during training.

BERT's higher accuracy on the MLM pre-training task is not simply a result of having more parameters than the other models. Indeed, Table 7 shows that BERT-Base is actually more accurate than FNet-Large, which contains more than twice as many parameters. BERT is presumably more expressive because the mixing (attention) weights are both task specific and token dependent, determined by token-token (query-key) dot products; see also Tay et al. (2020a). FNet's mixing weights, on the other hand, are neither task specific nor token dependent.

A.2 TPU results

In this section, we report FNet efficiency results for TPUs; the main text focuses on GPUs. Figure 3 shows the speed vs MLM pre-training accuracy curve when training on TPU (4×4 v3 chips). As on GPUs, FNet and the Linear model define the Pareto efficiency frontier for smaller, faster models, while BERT defines the frontier for larger, slower models.

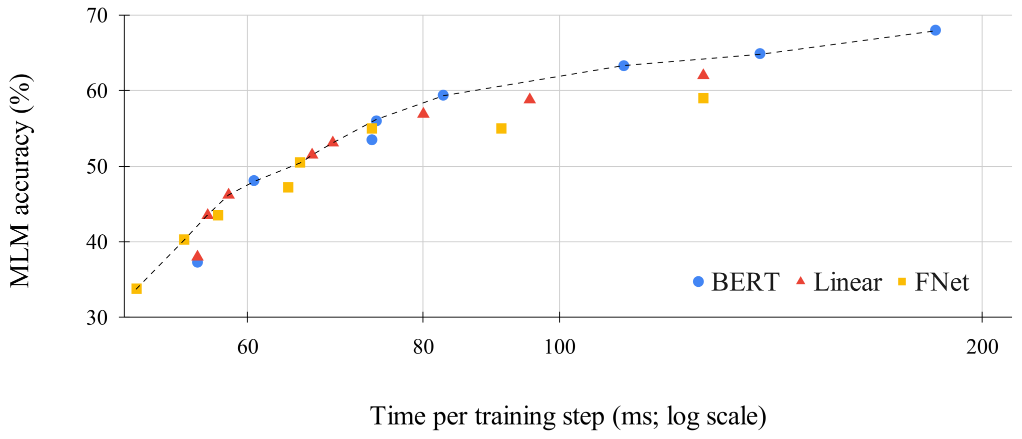

Figure 3: Speed-accuracy trade-offs for TPU pre-training. The dashed line shows the Pareto efficiency frontier.

Table 8: TPU training speeds (in steps per second; larger is better), inference speeds (in milliseconds per batch; smaller is better) and peak memory usage during training (in GB; smaller is better) on the Long-Range Text classification task. Speed up multipliers relative to the Transformer are given in parentheses.

Seq. length	512	1024	2048	4096	8192	16386
			Training Speed (steps/s)			
Transformer	8.0	5.6	1.7	OOM	OOM	OOM
Linear	9.4 (1.2x)	**9.1 (1.6x)**	**7.6 (4.5x)**	3.9	1.4	OOM
FNet (mat)	**9.5 (1.2x)**	**9.1 (1.6x)**	6.1 (3.6x)	3.0	0.8	0.2
FNet (FFT)	8.6 (1.1x)	6.0 (1.1x)	3.2 (1.9x)	1.6	0.8	0.3
Performer	9.2 (1.2x)	8.4 (1.5x)	6.9 (4.1x)	**4.2**	**2.2**	**1.1**
			Inference Speed (ms/batch)			
Transformer	7.0	13.2	39.4	129.9	490.2	OOM
Linear	**5.6 (1.2x)**	**6.5 (2.0x)**	**9.6 (4.1x)**	20.4 (6.4x)	54.6 (9.0x)	OOM
FNet (mat)	6.0 (1.2x)	7.7 (1.7x)	15.4 (2.6x)	40.7 (3.2x)	137.0 (3.6x)	454.5
FNet (FFT)	10.8 (0.7x)	16.8 (0.8x)	29.9 (1.3x)	58.8 (2.2x)	113.6 (4.3x)	263.2
Performer	6.1 (1.2x)	7.2 (1.8x)	10.1 (3.9x)	**17.5 (7.4x)**	**31.8 (15.4x)**	**61.0**
			Peak Memory Usage (GB)			
Transformer	1.1	2.1	5.8	9.1	OOM	OOM
Linear	0.9	1.1	1.9	4.9	14.8	OOM
FNet (mat)	**0.8**	**0.9**	**1.3**	2.2	4.8	11.9
FNet (FFT)	**0.8**	**0.9**	**1.3**	**2.0**	**3.5**	**6.3**
Performer	1.0	1.3	1.8	3.0	5.1	9.6

Table 8 shows Long Range Arena Text classification efficiency results on TPUs (4×4 v3 chips). The Linear model and FNet training faster than all the efficient Transformers for sequence lengths ≤ 2048 and 512, respectively. For longer sequences, FNet is slower than the Performer and, based on results in Tay et al. (2020c), likely also slower than the other efficient Transformers that linearize attention, namely Local Attention (Parmar et al., 2018), Linformer (Wang et al., 2020) and Linear Transformer (Katharopoulos et al., 2020). However, it is worth noting that Table 1 in Tay et al. (2020c) and Table 4 in this paper together suggest that FNet is more accurate than all of the aforementioned models. Moreover, we expect that the GPU speed gains will transfer to TPUs as the TPU FFT implementation improves.

A.3 Additional configurations that we experimented with

We experimented with a number of additional ideas to improve FNet.

Fourier Transform algorithm On GPUs, the FFT was the fastest algorithm for computing the DFT across all sequence lengths that we experimented with ($512 - 8192$). On TPUs, it is faster to compute the DFT directly using matrix multiplications for relatively shorter sequence lengths (up to lengths of 4096; see Table 8) and using the FFT for longer lengths. This efficiency boundary between matrix multiplication and FFT on TPUs will change depending on the XLA precision for the matrix multiplications. We found that, although (slower) HIGHEST XLA precision was required to very accurately reproduce FFT in computing the DFT, (faster) DEFAULT XLA precision was sufficient to facilitate accurate model convergence.

Modifying the Fourier Transform computation To keep the entire FNet architecture simple, the Fourier sublayer accepts real input and returns real output. The standard Fourier sublayer in FNet simply extracts the real part after computing the 2D DFT. We found that FNet was less accurate and less stable during training if only the real part of the DFT was used throughout the computation. Simply extracting the absolute value (instead of the real part) also led to a significantly less accurate model. Because the feed-forward sublayer mixes the hidden dimension, we experimented with applying a 1D DFT along the token dimension only in the Fourier sublayer (i.e. no hidden dimension mixing in the Fourier sublayer). This yielded some training speed gains but hurt accuracy. The 1D (token mixing only) DFT model still significantly outperformed the (no token mixing) FF-only model, indicating that token mixing is most important mechanism in the Fourier sublayer.

Other transformations The Discrete Cosine Transform (DCT), which is related to the DFT but transforms real input to real output, offers a natural alternate mixing mechanism to the Fourier Transform. However, we found that the DCT model was less accurate than FNet. We also experimented with using the Hadamard Transform[12], which, although slightly faster than the DFT, also yielded less accurate results.

Introducing learnable parameters to the Fourier sublayer Our attempts to introduce learnable parameters into the Fourier sublayer were either detrimental or inconsequential, and generally slightly slowed the model. For the (sequence length, hidden dimension) input in each Fourier sublayer, we tried two approaches to introduce learnable parameters: (1) element wise multiplication with a (sequence length, hidden dimension) matrix, and (2) regular matrix multiplication with (sequence length, sequence length) and (hidden dimension, hidden dimension) matrices. We experimented with these approaches in various configurations: preceding and/or following the DFT, and also in combination with inverse DFT (e.g. transform to frequency domain, apply element wise multiplication, transform back to time domain), but most setups degraded accuracy and reduced training stability, while a few offered did not change accuracy but lead to limited speed decreases. In a slightly different set of experiments and in an effort to provide more flexibility to the model, we added (complex) learnable weights to the 2D DFT matrix. This model was stable but did not yield any accuracy gains, suggesting that the DFT is locally optimal in some sense.

FNet block modifications The standard FNet encoder block structure follows that of the Transformer: a Fourier sublayer followed by a feed-forward sublayer, with residual connections and layer norms after each sublayer; see Figure 1. We tried several modifications to this structure, based on the intuition of moving in and out of the frequency domain between multiplications. For example, the sandwiching of Fourier, feed-forward, Fourier (or inverse Fourier) sublayers and only applying the residual connections and layer norms to the final result, yields a structure that more closely mimics convolutions. However, these setups degraded accuracy and lead to a more unstable model during training. Adding extra feed-forward sublayers to this layering, or swapping out the feed-forward sublayers for simpler dense sublayers, did not help either.

A.4 Mixing layer speeds

Table 9 summarizes the inference and training speeds for the different mixing layers. For each of the Base and Large configurations, we have removed all other sublayers and transformations and then calculated the speed per batch of input examples. The FNet training speeds are particularly fast because no parameters are updated. The Linear model has faster inference than FNet on TPUs because it is performing real matrix multiplications, whereas FNet performs complex matrix multiplications; see Equation (3).

[12]Whereas the DFT matrix in Equation (2) contains N roots of unity, the Hadamard Transform simply contains two roots of unity: $\{\pm 1\}$; see also Kunz (1979).

Table 9: Inference (forward pass; left) and training (forward and backward passes; right) speeds for *only* the mixing sublayers – all other model sublayers are removed. Both speeds are measured in milliseconds per batch (smaller is better), with batch sizes of 64 (GPU) and 256 (TPU). All batch examples have the sequence length fixed at 512. FNet uses the FFT for GPUs and matrix multiplications for TPUs. Speed up multipliers relative to self-attention are given in parentheses.

	Training speed (ms/batch)		Inference speed (ms/batch)	
	GPU	TPU	GPU	TPU
Self-attention (Base)	136	76	43	16
Linear (Base)	36 (3.7x)	12 (6.1x)	15 (2.8x)	**4 (3.9x)**
FNet (Base)	**11 (12.2x)**	**8 (9.9x)**	**11 (4.0x)**	8 (2.1x)
Self-attention (Large)	404	212	128	43
Linear (Large)	103 (3.9x)	35 (6.1x)	36 (3.6x)	**10 (4.5x)**
FNet (Large)	**18 (22.2x)**	**22 (9.7x)**	**18 (7.3x)**	22 (2.0x)

Although the Fourier mixing sublayer itself performs forward and backward passes significantly faster than the self-attention sublayer, FNet is overall 70-80% faster than BERT because the overall training and inference speeds are bottle-necked by the feed-forward sublayers that all models share.

A.5 A note on Long-Range Arena hyperparameter settings

Concerning the Long-Range Arena setup, several hyperparameters are not described in Tay et al. (2020c) and there a few mismatches between the configurations described in the paper and the code repository. Where possible, we prioritize configurations described in the paper with only two exceptions. Firstly, for the CIFAR10 (Image) task, we perform a sweep of the number of layers in the range $[1, 2, 3, 4]$. We found that 1 layer worked best for all models; Tay et al. (2020c) suggest 3 layers yielded the best results. Secondly, for the Pathfinder task, we found that a base learning rate of 0.001 (as given in the code repository) yielded better results for all models than the 0.01 value indicated in Tay et al. (2020c). We also perform a very small sweep over the embedding dimension and batch size, which are not listed in Tay et al. (2020c).

We also remark that the accuracy comparisons of Table 4 with Tay et al. (2020c), should be performed with two general caveats. First, we found that results for certain tasks - Text and Retrieval in particular - can vary quite a bit between runs, especially for the Transformer; we report the best results. Second, the learning configuration may not be fully optimized even for the Transformer; for example, we found that the Retrieval and ListOps tasks may both benefit from training for longer than suggested in Tay et al. (2020c).

A.6 FNet code

```
import flax.linen as nn
import jax
import jax.numpy as jnp


class FourierTransformLayer(nn.Module):
  @nn.compact
  def __call__(self, x):
    return jax.vmap(jnp.fft.fftn)(x).real


class FeedForwardLayer(nn.Module):
  d_ff: int
  dropout_rate: float

  @nn.compact
  def __call__(self, x, deterministic):
    x = nn.Dense(self.d_ff,
        kernel_init=nn.initializers.normal(2e-2),
        bias_init=nn.initializers.normal(2e-2),
        name="intermediate")(x)
```

```python
    x = nn.gelu(x)
    x = nn.Dense(x.shape[-1],
      kernel_init=nn.initializers.normal(2e-2),
      name="output")(x)
    return nn.Dropout(self.dropout_rate)(x, deterministic)


class FNetEncoderBlock(nn.Module):
  fourier_layer: FourierTransformLayer
  ff_layer: FeedForwardLayer

  @nn.compact
  def __call__(self, x, deterministic):
    mixing_output = self.fourier_layer(x)
    x = nn.LayerNorm(1e-12, name="mixing_layer_norm")(x + mixing_output)
    feed_forward_output = self.ff_layer(x, deterministic)
    return nn.LayerNorm(
        1e-12, name="output_layer_norm")(x + feed_forward_output)


class FNetEncoder(nn.Module):
  num_layers: int
  d_model: int
  d_ff: int
  dropout_rate: float

  def setup(self):
    encoder_blocks = []
    for layer in range(self.num_layers):
      encoder_blocks.append(FNetEncoderBlock(
        FourierTransformerLayer(),
        FeedForwardLayer(self.d_ff, self.dropout_rate),
        name=f"encoder_{layer}"))
    self.encoder_blocks = encoder_blocks
    self.pooler = nn.Dense(
        self.d_model,
        kernel_init=nn.initializers.normal(2e-2),
        name="pooler")

  def __call__(self, x, deterministic):
    for encoder_block in self.encoder_blocks:
      x = encoder_block(x, deterministic)
    pooled_output = self.pooler(x[:, 0])
    pooled_output = jnp.tanh(pooled_output)
    return x, pooled_output
```
Listing 1: FNet code written in JAX/Flax. Embedding and output projection layers are omitted for simplicity.