Scalable Neural Network Kernels

Arijit Sehanobish*, Krzysztof Choromanski*, Yunfan Zhao*,
Avinava Dubey*, Valerii Likhosherstov*

** Equal Contribution*

Correspondence to kchoro@google.com, arijit.sehanobish1@gmail.com

Random Fourier Feature Maps for Approximating Feed Forward Layers

- Feed Forward Layer (FFL)

$$\mathbf{x} \mapsto f(\mathbf{W}\mathbf{x} + \mathbf{b}),\ x \in \mathbb{R}^d,\ \mathbf{W} \in \mathbb{R}^{l \times d},\ \mathbf{b} \in \mathbb{R}^l(\text{bias}),\ f : \mathbb{R} \to \mathbb{R}\,(\text{ activation function})$$

Random Fourier Feature Maps for Approximating Feed Forward Layers

- Feed Forward Layer (FFL)

$$\mathbf{x} \mapsto f(\mathbf{W}\mathbf{x} + \mathbf{b}),\ x \in \mathbb{R}^d,\ \mathbf{W} \in \mathbb{R}^{l \times d},\ \mathbf{b} \in \mathbb{R}^l(\text{bias}),\ f : \mathbb{R} \to \mathbb{R}\,(\text{ activation function})$$

- Random Features for Approximation of FFL

$$K_f(\mathbf{x}, (\mathbf{W}, \mathbf{b})) := \mathbb{E}[\Phi_f(\mathbf{x})^\top \Psi_f(\mathbf{W}, \mathbf{b})]$$

Random Fourier Feature Maps for Approximating Feed Forward Layers

- Feed Forward Layer (FFL)
  $\mathbf{x} \mapsto f(\mathbf{W}\mathbf{x} + \mathbf{b}),\ x \in \mathbb{R}^d,\ \mathbf{W} \in \mathbb{R}^{l \times d},\ \mathbf{b} \in \mathbb{R}^l(\text{bias}),\ f : \mathbb{R} \to \mathbb{R}(\text{ activation function})$

- Random Features for Approximation of FFL
  $$K_f(\mathbf{x}, (\mathbf{W}, \mathbf{b})) := \mathbb{E}[\Phi_f(\mathbf{x})^\top \Psi_f(\mathbf{W}, \mathbf{b})]$$

- $\Phi_f : \mathbb{R}^d \to \mathbb{R}^m,\ \Psi_f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^m$ satisfy: $f(\mathbf{w}^\top \mathbf{x} + \mathbf{b}) = \mathbb{E}[\Phi_f(\mathbf{x})^\top \Psi_f(\mathbf{w}, b)]$

Random Fourier Feature Maps for Approximating Feed Forward Layers

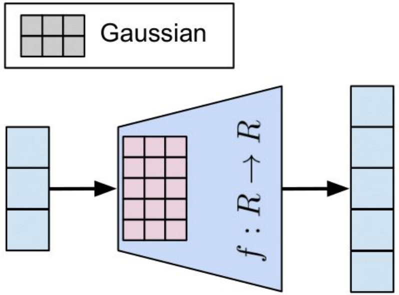
- Feed Forward Layer (FFL)

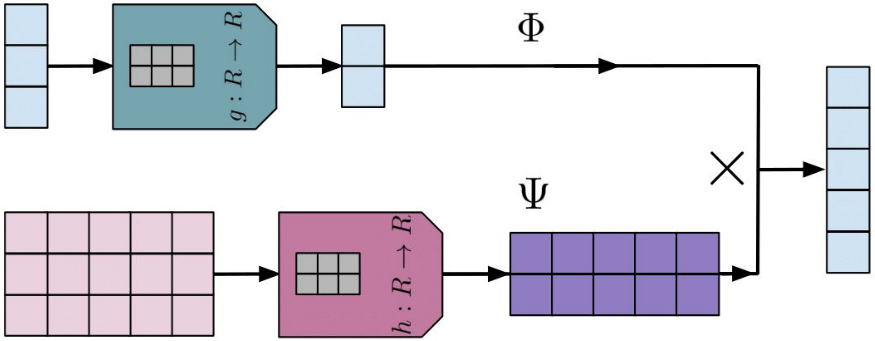
  $\mathbf{x} \mapsto f(\mathbf{W}\mathbf{x} + \mathbf{b}),\ x \in \mathbb{R}^d,\ \mathbf{W} \in \mathbb{R}^{l \times d},\ \mathbf{b} \in \mathbb{R}^l(\text{bias}),\ f : \mathbb{R} \to \mathbb{R}\,(\text{ activation function})$

- Random Features for Approximation of FFL

  $K_f(\mathbf{x}, (\mathbf{W}, \mathbf{b})) := \mathbb{E}[\Phi_f(\mathbf{x})^\top \Psi_f(\mathbf{W}, \mathbf{b})]$

- $\Phi_f : \mathbb{R}^d \to \mathbb{R}^m,\ \Psi_f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^m$ satisfy: $f(\mathbf{w}^\top \mathbf{x} + \mathbf{b}) = \mathbb{E}[\Phi_f(\mathbf{x})^\top \Psi_f(\mathbf{w}, b)]$





Benefits of such a Mechanism

- **Network Compression :** Instead of transforming layer parameters with Ψ, one can directly learn vectors $\Psi_f(w, \mathbf{b})$. Then the number of trainable parameters becomes O($m\ell$) rather than O(ℓd) for $m << d$, reducing the parameter count.

- **Computational Savings :** if Random Features (RF) can be constructed efficiently, then the overall time complexity (given pre-computed embeddings $\Psi_f(w,\mathbf{b})$ is **sub-quadratic** in layers' dimensionalities.

- **Deep Neural Network (NN) Bundling Process :** a two-tower representation can be used iteratively to compactify multiple FFLs of NNs, the process we refer to as *neural network bundling* leading to the computational gains.

How can we construct such a mechanism?

- Write f as the inverse Fourier transform of it's Fourier transform F and write F as the sum of positive and negative parts of it's real and imaginary parts.

How can we construct such a mechanism?

- Write f as the inverse Fourier transform of it's Fourier transform F and write F as the sum of positive and negative parts of it's real and imaginary parts.

- $\widehat{f}(\mathbf{x}, \mathbf{w}, b) := c\,\mathbb{E}_{\xi\sim\overline{\mathcal{P}}}[S(\xi, b)\exp(\widehat{\mathbf{x}}^\top(\xi)\widehat{\mathbf{w}}(\xi))]$ where

- $\hat{f}$:= the inverse Fourier transform of positive real part of F,
- c := suitably chosen constant
- $S(\xi, b) = \dfrac{p(\xi)}{\overline{p}(\xi)}\exp(2\pi i\xi b),\ \widehat{\mathbf{x}}(\xi) = \rho(\xi)\mathbf{x},\ \widehat{\mathbf{w}}(\xi) = \eta(\xi)\mathbf{w},$ where $\rho(\xi), \eta(\xi) \in \mathbb{C}$ satisfy: $\rho(\xi)\eta(\xi) = 2\pi i\xi$

How can we construct such a mechanism?

- Write f as the inverse Fourier transform of it's Fourier transform F and write F as the sum of positive and negative parts of it's real and imaginary parts.
- $\widehat{f}(\mathbf{x}, \mathbf{w}, b) := c\,\mathbb{E}_{\xi\sim\overline{\mathcal{P}}}[S(\xi, b)\exp(\widehat{\mathbf{x}}^\top(\xi)\widehat{\mathbf{w}}(\xi))]$ where

- $\hat{f}$:= inverse Fourier transform of positive real part of F,
- c := suitably chosen constant
- $S(\xi, b) = \dfrac{p(\xi)}{\overline{p}(\xi)}\exp(2\pi i\xi b),\ \widehat{\mathbf{x}}(\xi) = \rho(\xi)\mathbf{x},\ \widehat{\mathbf{w}}(\xi) = \eta(\xi)\mathbf{w},\ \text{where } \rho(\xi), \eta(\xi) \in \mathbb{C} \text{ satisfy: } \rho(\xi)\eta(\xi) = 2\pi i\xi$

- p is related to the distribution attached to the inverse Fourier transform of F

How can we construct such a mechanism?

- Write f as the inverse Fourier transform of it's Fourier transform F and write F as the sum of positive and negative parts of it's real and imaginary parts.
- $\widehat{f}(\mathbf{x}, \mathbf{w}, b) := c\,\mathbb{E}_{\xi\sim\overline{\mathcal{P}}}[S(\xi, b)\exp(\widehat{\mathbf{x}}^\top(\xi)\widehat{\mathbf{w}}(\xi))]$ where
  - $\hat{f}$:= inverse Fourier transform of positive real part of F,
  - c := suitably chosen constant
  - $S(\xi, b) = \dfrac{p(\xi)}{\overline{p}(\xi)}\exp(2\pi i\xi b),\ \widehat{\mathbf{x}}(\xi) = \rho(\xi)\mathbf{x},\ \widehat{\mathbf{w}}(\xi) = \eta(\xi)\mathbf{w},\ \text{where } \rho(\xi), \eta(\xi) \in \mathbb{C} \text{ satisfy: } \rho(\xi)\eta(\xi) = 2\pi i\xi$
- p is related to the distribution attached to the inverse Fourier transform of F
- Note : the exp term can be linearized by techniques developed by many authors (see Performer for ex.)

SNNKs are strictly more general than FFL

- Φ and Ψ are defined as:

$$\Phi(\mathbf{x}) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{x}), \; \Psi(\mathbf{w}, b) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{w})$$

for the Gaussian matrix: $\mathbf{G} \in \mathbb{R}^{l \times d}$ with entries sampled independently at random from $\mathcal{N}(0, 1)$

SNNKs are strictly more general than FFL

- Φ and Ψ are defined as:

$$\Phi(\mathbf{x}) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{x}),\ \Psi(\mathbf{w}, b) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{w})$$

  for the Gaussian matrix: $\mathbf{G} \in \mathbb{R}^{l \times d}$ with entries sampled independently at random from $\mathcal{N}(0, 1)$

- **Arc-cosine Kernels (Cho and Saul, 2011) :** The nth-order arc-cosine kernel $\mathrm{K}_n : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is defined as: $\mathrm{K}_n(\mathbf{x}, \mathbf{y}) = \frac{1}{\pi}\|\mathbf{x}\|_2^n\|\mathbf{y}\|_2^n J_n(\alpha_{\mathbf{x},\mathbf{y}})$ where $\alpha_{\mathbf{x},\mathbf{y}} \in [0, \pi]$ is the angle between $\mathbf{x}$ and $\mathbf{y}$ and $J(\theta) := (-1)^n(\sin\theta)^{n+1}\frac{\partial^n}{\partial\theta^n}\left(\frac{\pi - \theta}{\sin\theta}\right)$

SNNKs are strictly more general than FFL

- Φ and Ψ are defined as:

$$\Phi(\mathbf{x}) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{x}),\ \Psi(\mathbf{w}, b) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{w})$$

for the Gaussian matrix: $\mathbf{G} \in \mathbb{R}^{l \times d}$ with entries sampled independently at random from $\mathcal{N}(0, 1)$

- **Arc-cosine Kernels (Cho and Saul, 2011) :** The nth-order arc-cosine kernel $\mathrm{K}_n : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is defined as : $\mathrm{K}_n(\mathbf{x}, \mathbf{y}) = \frac{1}{\pi}\|\mathbf{x}\|_2^n \|\mathbf{y}\|_2^n J_n(\alpha_{\mathbf{x},\mathbf{y}})$ where $\alpha_{\mathbf{x},\mathbf{y}} \in [0, \pi]$ is the angle between $\mathbf{x}$ and $\mathbf{y}$ and $J(\theta) := (-1)^n (\sin\theta)^{n+1} \frac{\partial^n}{\partial\theta^n}\left(\frac{\pi - \theta}{\sin\theta}\right)$

- Then K_n can be linearized as:

$$\mathrm{K}_n(\mathbf{x}, \mathbf{y}) = 2\mathbb{E}[\Gamma_n(\mathbf{x})^\top \Gamma_n(\mathbf{y})] \text{ for } \Gamma_n(\mathbf{v}) \stackrel{\text{def}}{=} \mathrm{ReLU}((\mathbf{v}^\top \boldsymbol{\omega})^n) \text{ and } \boldsymbol{\omega} \sim \mathcal{N}(0, \mathbf{I}_d)$$

SNNKs are strictly more general than FFL

- Φ and Ψ are defined as:

$$\Phi(\mathbf{x}) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{x}),\ \Psi(\mathbf{w}, b) = \mathrm{ReLU}(\frac{1}{\sqrt{l}}\mathbf{G}\mathbf{w})$$

for the Gaussian matrix: $\mathbf{G} \in \mathbb{R}^{l \times d}$ with entries sampled independently at random from $\mathcal{N}(0, 1)$

- **Arc-cosine Kernels (Cho and Saul, 2011) :** The nth-order arc-cosine kernel $\mathrm{K}_n : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is defined as : $\mathrm{K}_n(\mathbf{x}, \mathbf{y}) = \frac{1}{\pi}\|\mathbf{x}\|_2^n \|\mathbf{y}\|_2^n J_n(\alpha_{\mathbf{x},\mathbf{y}})$ where $\alpha_{\mathbf{x},\mathbf{y}} \in [0, \pi]$ is the angle between $\mathbf{x}$ and $\mathbf{y}$ and $J(\theta) := (-1)^n (\sin\theta)^{n+1} \frac{\partial^n}{\partial\theta^n}\left(\frac{\pi - \theta}{\sin\theta}\right)$

- Then K_n can be linearized as:

$$\mathrm{K}_n(\mathbf{x}, \mathbf{y}) = 2\mathbb{E}[\Gamma_n(\mathbf{x})^\top \Gamma_n(\mathbf{y})] \text{ for } \Gamma_n(\mathbf{v}) \stackrel{\text{def}}{=} \mathrm{ReLU}((\mathbf{v}^\top \boldsymbol{\omega})^n) \text{ and } \boldsymbol{\omega} \sim \mathcal{N}(0, \mathbf{I}_d)$$

- ReLU-SNNK layer is *not* a regular FFL since it can not be written as $f(\mathbf{x}^\top \mathbf{w} + b)$ for some $f : \mathbb{R} \to \mathbb{R}$

Experiments-1

- Based on SNNK, design novel adapter layers. In this case, only adapter layers are trained while the base model is frozen.

Dataset	# Training Parameters	RTE	MRPC	QNLI	QQP	SST-2	MNLI	STSB	COLA
Full fine-tuning	110M	66.2	90.5	**91.3**	**91.4**	**92.6**	**84.1**	88.8	**59.5**
Adapter (Moosavi et al 2022)	.9M	63.83	84.8	90.63	88.12	91.74	83.53	88.48	56.51
ReLU-SNNK-Adapter (ours)	$.3M$	**69.68**	**91.26**	90.44	85.82	92.31	82.06	**88.81**	58.21

Table showing results on SNNK-adapters on the GLUE benchmark using the BERT model as the pretrained model. Our methods perform favorably with various baselines even with at least 3x fewer parameters.

Experiments-2

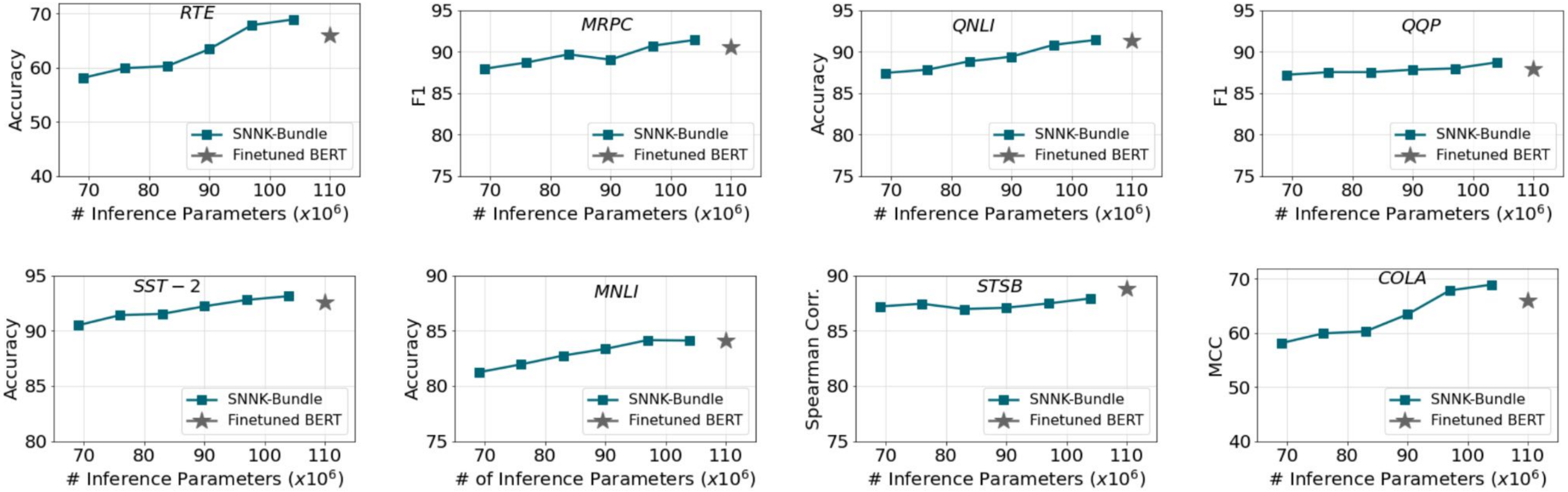
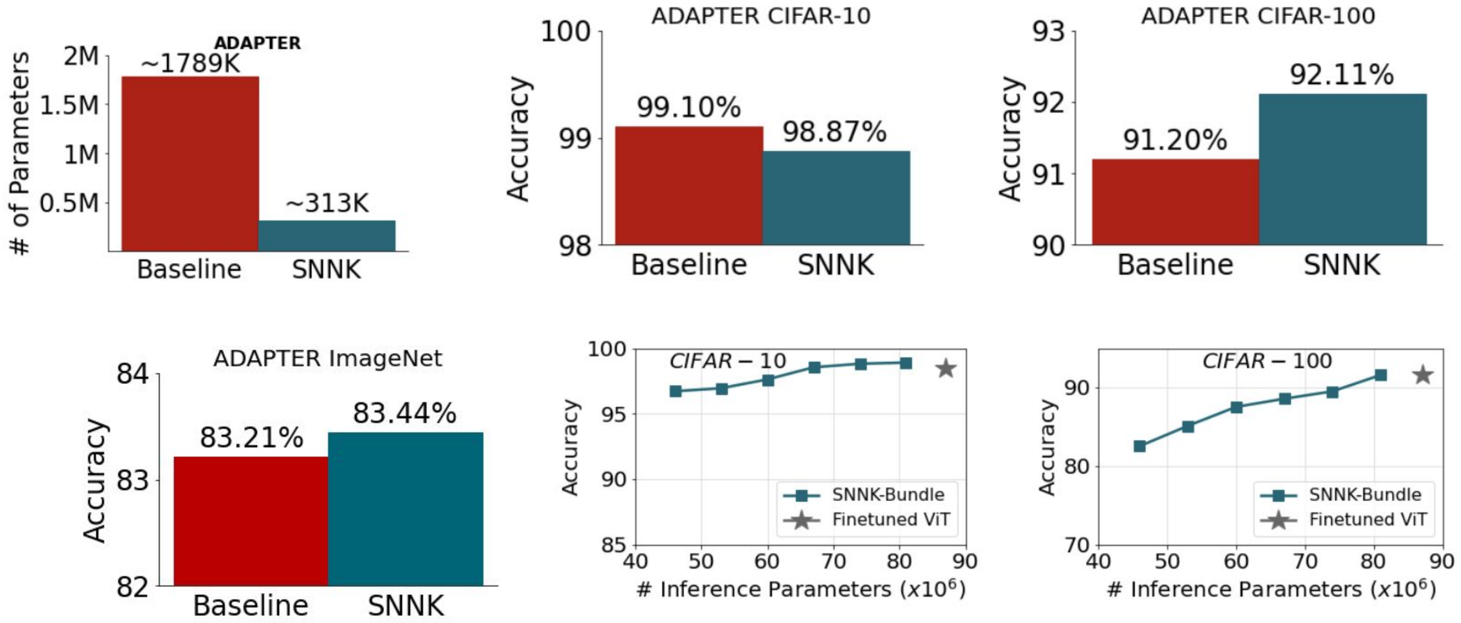


Table showing results on SNNK uptraining on the GLUE benchmark using the BERT model as the pretrained model. We compress the BERT model by almost half without much degradation in performance.

Experiments-3



Results on CiFAR-10, CiFAR-100 and ImageNet. **Top row:** SNNK-adapter results. **Bottom row: (left)** Adapter-SNNK on ImageNet and **(right)** Bundled ViT results

For more results see our paper and come to our poster session

Thank you!