THE RECURRENT NEURAL TANGENT KERNEL

Sina Alemohammad, Zichao Wang, Randall Balestriero, Richard G. Baraniuk
Department of Electrical and Computer Engineering
Rice University
{sa86,zw16,rb42,richb}@rice.edu

ABSTRACT

The study of deep neural networks (DNNs) in the infinite-width limit, via the so-called *neural tangent kernel* (NTK) approach, has provided new insights into the dynamics of learning, generalization, and the impact of initialization. One key DNN architecture remains to be kernelized, namely, the recurrent neural network (RNN). In this paper we introduce and study the *Recurrent Neural Tangent Kernel* (RNTK), which provides new insights into the behavior of overparametrized RNNs. A key property of the RNTK should greatly benefit practitioners is its ability to compare inputs of different length. To this end, we characterize how the RNTK weights different time steps to form its output under different initialization parameters and nonlinearity choices. A synthetic and 56 real-world data experiments demonstrate that the RNTK offers significant performance gains over other kernels, including standard NTKs, across a wide array of data sets.

1 INTRODUCTION

The overparameterization of modern deep neural networks (DNNs) has resulted in not only remarkably good generalization performance on unseen data (Novak et al., 2018; Neyshabur et al., 2019; Belkin et al., 2019) but also guarantees that gradient descent learning can find the global minimum of their highly nonconvex loss functions (Du et al., 2019b; Allen-Zhu et al., 2019b;a; Zou et al., 2018; Arora et al., 2019b). From these successes, a natural question arises: What happens when we take overparameterization to the limit by allowing the width of a DNN's hidden layers to go to infinity? Surprisingly, the analysis of such an (impractical) DNN becomes analytically tractable. Indeed, recent work has shown that the training dynamics of (infinite-width) DNNs under gradient flow is captured by a constant kernel called the *Neural Tangent Kernel* (NTK) that evolves according to a linear ordinary differential equation (ODE) (Jacot et al., 2018; Lee et al., 2019; Arora et al., 2019a).

Every DNN architecture and parameter initialization produces a distinct NTK. The original NTK was derived from the Multilayer Perceptron (MLP)(Jacot et al., 2018) and was soon followed by kernels derived from Convolutional Neural Networks (CNTK) (Arora et al., 2019a; Yang, 2019a), Residual DNNs (Huang et al., 2020), and Graph Convolutional Neural Networks (GNTK) (Du et al., 2019a). In (Yang, 2020a), a general strategy to obtain the NTK of any architecture is provided.

In this paper, we extend the NTK concept to the important class of overparametrized Recurrent Neural Networks (RNNs), a fundamental DNN architecture for processing sequential data. We show that RNN in its infinite-width limit converges to a kernel that we dub the ***Recurrent Neural Tangent Kernel*** (RNTK). The RNTK provides high performance for various machine learning tasks, and an analysis of the properties of the kernel provides useful insights into the behavior of RNNs in the following overparametrized regime. In particular, we derive and study the RNTK to answer the following theoretical questions:

Q: Can the RNTK extract long-term dependencies between two data sequences? RNNs are known to underperform at learning long-term dependencies due to the gradient vanishing or exploding (Bengio et al., 1994). Attempted ameliorations have included orthogonal weights (Arjovsky et al., 2016; Jing et al., 2017; Henaff et al., 2016) and gating such as in Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al., 2014) RNNs. We demonstrate that the RNTK can detect long-term dependencies with proper initialization of the hyperparameters, and moreover, we show how the dependencies are extracted through time via different hyperparameter choices.

Q: Do the recurrent weights of the RNTK reduce its representation power compared to other NTKs? An attractive property of an RNN that is shared by the RNTK is that it can deal with sequences of different lengths via weight sharing through time. This enables the reduction of the number of learnable parameters and thus more stable training at the cost of reduced representation power. We prove the surprising fact that employing tied vs. untied weights in an RNN *does not* impact the analytical form of the RNTK.

Q: Does the RNTK generalize well? A recent study has revealed that the use of an SVM classifier with the NTK, CNTK, and GNTK kernels outperforms other classical kernel-based classifiers and trained finite DNNs on small data sets (typically fewer than 5000 training samples) (Lee et al., 2020; Arora et al., 2019a; 2020; Du et al., 2019a). We extend these results to RNTKs to demonstrate that the RNTK outperforms a variety of classic kernels, NTKs *and* finite RNNs for time series data sets in both classification and regression tasks. Carefully designed experiments with data of varying lengths demonstrate that the RNTK's performance accelerates beyond other techniques as the difference in lengths increases. Those results extend the empirical observations from (Arora et al., 2019a; 2020; Du et al., 2019a; Lee et al., 2020) into finite DNNs, NTK, CNTK, and GNTK comparisons by observing that their performance-wise ranking depends on the employed DNN architecture.

We summarize our contributions as follows:

[C1] We derive the analytical form for the RNTK of an overparametrized RNN at initialization using rectified linear unit (ReLU) and error function (erf) nonlinearities for arbitrary data lengths and number of layers (Section 3.1).

[C2] We prove that the RNTK remains constant during (overparametrized) RNN training and that the dynamics of training are simplified to a set of ordinary differential equations (ODEs) (Section 3.2).

[C3] When the input data sequences are of equal length, we show that the RNTKs of weight-tied and weight-untied RNNs converge to the same RNTK (Section 3.3).

[C4] Leveraging our analytical formulation of the RNTK, we empirically demonstrate how correlations between data at different times are weighted by the function learned by an RNN for different sets of hyperparameters. We also offer practical suggestions for choosing the RNN hyperparameters for deep information propagation through time (Section 3.4).

[C5] We demonstrate that the RNTK is eminently practical by showing its superiority over classical kernels, NTKs, and finite RNNs in exhaustive experiments on time-series classification and regression with both synthetic and 56 real-world data sets (Section 4).

2 Background and Related Work

Notation. We denote $[n] = \{1, \ldots, n\}$, and I_d the identity matrix of size d. $[A]_{i;j}$ represents the (i, j)-th entry of a matrix, and similarly $[a]_i$ represents the i-th entry of a vector. We use $\phi(\cdot) : \mathbb{R} \to \mathbb{R}$ to represent the activation function that acts coordinate wise on a vector and ϕ' to denote its derivative. We will often use the rectified linear unit (ReLU) $\phi(x) = \max(0, x)$ and error function (erf) $\phi(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-z^2} dz$ activation functions. $\mathcal{N}(\mu, \Sigma)$ represents the multidimensional Gaussian distribution with the mean vector μ and the covariance matrix Σ.

Recurrent Neural Networks (RNNs). Given an input sequence data $x = \{x_t\}_{t=1}^T$ of length T with data at time t, $x_t \in \mathbb{R}^m$, a *simple RNN* (Elman, 1990) performs the following recursive computation at each layer ℓ and each time step t

$$g^{(\ell;t)}(x) = W^{(\ell)} h^{(\ell;t-1)}(x) + U^{(\ell)} h^{(\ell-1;t)}(x) + b^{(\ell)}, \qquad h^{(\ell;t)}(x) = \phi\left(g^{(\ell;t)}(x)\right),$$

where $W^{(\ell)} \in \mathbb{R}^{n \times n}$, $b^{(\ell)} \in \mathbb{R}^n$ for $\ell \in [L]$, $U^{(1)} \in \mathbb{R}^{n \times m}$ and $U^{(\ell)} \in \mathbb{R}^{n \times n}$ for $\ell \geq 2$ are the RNN parameters. $g^{(\ell;t)}(x)$ is the pre-activation vector at layer ℓ and time step t, and $h^{(\ell;t)}(x)$ is the after-activation (hidden state). For the input layer $\ell = 0$, we define $h^{(0;t)}(x) := x_t$. $h^{(\ell;0)}(x)$ as the initial hidden state at layer ℓ that must be initialized to start the RNN recursive computation.

The output of an L-hidden layer RNN with linear read out layer is achieved via

$$f(x) = V h^{(L;T)}(x),$$

where $V \in \mathbb{R}^{d \times n}$. Figure 1 visualizes an RNN unrolled through time.

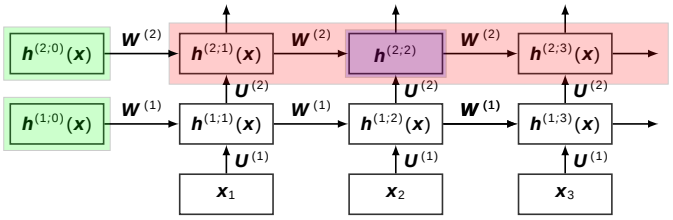

Figure 1: Visualization of a simple RNN that highlights a cell (purple), a layer (red) and the initial hidden state of each layer (green). (Best viewed in color.)

Neural Tangent Kernel (NTK). Let $f(x) \in \mathbb{R}^d$ be the output of a DNN with parameters θ. For two input data sequences x and x', the NTK is defined as (Jacot et al., 2018)

$$\hat{\Theta}_s(x, x') = \langle \nabla_\theta f_s(x), \nabla_\theta f_s(x') \rangle,$$

where f_s and θ_s are the network output and parameters during training at time s.[1] Let X and Y be the set of training inputs and targets, $\ell(\hat{y}, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ be the loss function, and $L = \frac{1}{|X||Y|} \sum_{(x,y) \in X \times Y} \ell(f_s(x), y)$ be the the empirical loss. The evolution of the parameters θ_s and output of the network f_s on a test input using gradient descent with infinitesimal step size (a.k.a gradient flow) with learning rate η is given by

$$\frac{\partial \theta_s}{\partial s} = -\eta \nabla_\theta f_s(X)^T \nabla_{f_s(X)} L \tag{1}$$

$$\frac{\partial f_s(x)}{\partial s} = -\eta \nabla_\theta f_s(x) \nabla_\theta f_s(X)^T \nabla_{f_s(X)} L = -\eta \hat{\Theta}_s(x, X) \nabla_{f_s(X)} L. \tag{2}$$

Generally, $\hat{\Theta}_s(x, x')$, hereafter referred to as the empirical NTK, changes over time during training, making the analysis of the training dynamics difficult. When f_s corresponds to an infinite-width MLP, (Jacot et al., 2018) showed that $\hat{\Theta}_s(x, x')$ converges to a limiting kernel at initialization and stays constant during training, i.e.,

$$\lim_{n \to \infty} \hat{\Theta}_s(x, x') = \lim_{n \to \infty} \hat{\Theta}_0(x, x') := \Theta(x, x') \quad \forall s,$$

which is equivalent to replacing the outputs of the DNN by their first-order Taylor expansion in the parameter space (Lee et al., 2019). With a mean-square error (MSE) loss function, the training dynamics in (1) and (2) simplify to a set of linear ODEs, which coincides with the training dynamics of kernel ridge regression with respect to the NTK when the ridge term goes to zero. A nonzero ridge regularization can be conjured up by adding a regularization term $\frac{\alpha}{2} \|\theta_s - \theta_0\|_2^2$ to the empirical loss (Hu et al., 2020).

3 The Recurrent Neural Tangent Kernel

We are now ready to derive the RNTK. We first prove the convergence of an RNN at initialization to the RNTK in the infinite-width limit and discuss various insights it provides. We then derive the convergence of an RNN after training to the RNTK. Finally, we analyze the effects of various hyperparameter choices on the RNTK. Proofs of all of our results are provided in the Appendices.

3.1 RNTK for an Infinite-Width RNN at Initialization

First we specify the following parameter initialization scheme that follows previous work on NTKs (Jacot et al., 2018), which is crucial to our convergence results:

$$W^{(\ell)} = \frac{\sigma_w}{\sqrt{n}} \mathbf{W}^{(\ell)}, \quad U^{(1)} = \frac{\sigma_u}{\sqrt{m}} \mathbf{U}^{(1)}, \quad U^{(\ell)} = \frac{\sigma_u}{\sqrt{n}} \mathbf{U}^{(\ell)} (\ell \geq 2), \quad V = \frac{\sigma_v}{\sqrt{n}} \mathbf{V}, \quad b^{(\ell)} = \sigma_b \mathbf{b}^{(\ell)}, \tag{3}$$

where

$$[\mathbf{W}^{(\ell)}]_{i,j}, [\mathbf{U}^{(\ell)}]_{i,j}, [\mathbf{V}]_{i,j}, [\mathbf{b}^{(\ell)}]_i \sim \mathcal{N}(0, 1). \tag{4}$$

We will refer to (3) and (4) as the *NTK initialization*. The choices of the hyperparameters σ_w, σ_u, σ_v and σ_b can significantly impact RNN performance, and we discuss them in detail in Section

[1] We use s to denote time here, since t is used to index the time steps of the RNN inputs.

3.4. For the initial (at time $t = 0$) hidden state at each layer ℓ, we set $h^{(\ell,0)}(x)$ to an i.i.d. copy of $N(0, \sigma_h)$ (Wang et al., 2018) . For convenience, we collect all of the learnable parameters of the RNN into $\theta = \text{vect}\left[\{\mathbf{W}^{(\ell)}, \mathbf{U}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{L}, \mathbf{V}g\right]$.

The derivation of the RNTK at initialization is based on the correspondence between Gaussian initialized, infinite-width DNNs and Gaussian Processes (GPs), known as the DNN-GP. In this setting every coordinate of the DNN output tends to a GP as the number of units/neurons in the hidden layer (its width) goes to infinity. The corresponding DNN-GP kernel is computed as

$$K(x, x') = \mathbb{E}_N\left[[f(x)]_i \cdot [f(x')]_i\right], \ \forall i \in [d]. \tag{5}$$

First introduced for a single-layer, fully-connected neural network by (Neal, 1995), recent works on NTKs have extended the results for various DNN architectures (Lee et al., 2018; Duvenaud et al., 2014; Novak et al., 2019; Garriga-Alonso et al., 2019; Yang, 2019b), where in addition to the output, all pre-activation layers of the DNN tends to a GPs in the infinite-width limit. In the case of RNNs, each coordinate of the RNN pre-activation $g^{(\ell,t)}(x)$ converges to a centered GP depending on the inputs with kernel

$$\Sigma^{(\ell,t,t')}(x, x') = \mathbb{E}_N\left[[g^{(\ell,t)}(x)]_i \cdot [g^{(\ell,t')}(x')]_i\right] \ \forall i \in [n]. \tag{6}$$

As per (Yang, 2019a), the gradients of random infinite-width DNNs computed during backpropagation are also Gaussian distributed. In the case of RNNs, every coordinate of the vector $\dot{\Sigma}^{(\ell,t)}(x) := \sqrt{n}\left(\nabla_{g^{(\ell,t)}(x)} f(x)\right)$ converges to a GP with kernel

$$\dot{\Sigma}^{(\ell,t,t')}(x, x') = \mathbb{E}_N\left[[\dot{\Sigma}^{(\ell,t)}(x)]_i \cdot [\dot{\Sigma}^{(\ell,t')}(x')]_i\right] \ \forall i \in [n]. \tag{7}$$

Both convergences occur independently of the coordinate index i and for inputs of possibly different lengths, i.e., $T \neq T'$. With (6) and (7), we now prove that an infinite-width RNN at initialization converges to the limiting RNTK.

Theorem 1 *Let x and x' be two data sequences of potentially different lengths T and T', respectively. Without loss of generality, assume that $T \leq T'$, and let $\tau := T' - T$. Let n be the number of units in the hidden layers, the empirical RNTK for an L-layer RNN with NTK initialization converges to the following limiting kernel as $n \to \infty$*

$$\lim_{n \to \infty} \widehat{\Theta}_0(x, x') = \Theta(x, x') = \Theta^{(L,T,T')}(x, x') \otimes I_d, \tag{8}$$

where

$$\Theta^{(L,T,T')}(x, x') = \left(\sum_{\ell=1}^{L}\sum_{t=1}^{T}\left(\Sigma^{(\ell,t,t+\tau)}(x, x') \cdot \dot{\Sigma}^{(\ell,t,t+\tau)}(x, x')\right)\right) + K(x, x'), \tag{9}$$

with $K(x, x')$, $\Sigma^{(\ell,t,t+\tau)}(x, x')$, and $\dot{\Sigma}^{(\ell,t,t+\tau)}(x, x')$ defined in (5)–(7).

Remarks. Theorem 1 holds generally for any two data sequences, including different lengths ones. This highlights the RNTK's ability to produce a similarity measure $\Theta(x, x')$ even if the inputs are of different lengths, without resorting to heuristics such as zero padding the inputs to the to the max length of both sequences. Dealing with data of different length is in sharp contrast to common kernels such as the classical radial basis functions, polynomial kernels, and current NTKs. We showcase this capability below in Section 4.

To visualize Theorem 1, we plot in the left plot in Figure 2 the convergence of a single layer, sufficiently wide RNN to its RNTK with the two simple inputs $x = \{1, -1, 1\}$ of length 3 and $x' = \{\cos(\alpha), \sin(\alpha)\}$ of length 2, where $\alpha = [0, 2\pi]$. For an RNN with a sufficiently large hidden state ($n = 1000$), we see clearly that it converges to the RNTK ($n = \infty$).

RNTK Example for a Single-Layer RNN. We present a concrete example of Theorem 1 by showing how to recursively compute the RNTK for a single-layer RNN; thus we drop the layer index for notational simplicity. ***We compute and display the RNTK for the general case of a multi-layer RNN in Appendix B.3***. To compute the RNTK $\Theta^{(T,T')}(x, x')$, we need to compute the GP

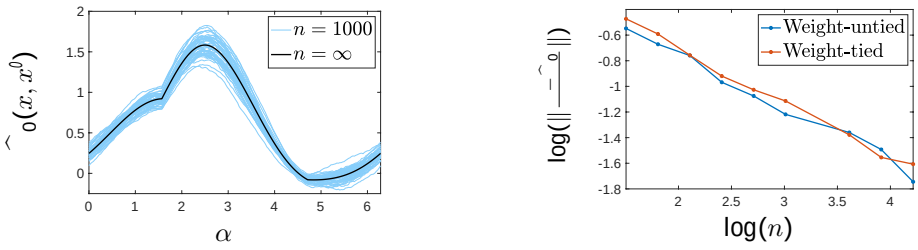

Figure 2: Empirical demonstration of a wide, single-layer RNN converging to its limiting RNTK. **Left**: convergence for a pair of different-length inputs $\boldsymbol{x} = \{1; -1; 1\}$ and $\boldsymbol{x}' = \{\cos(\ ); \sin(\ )\}$, with varying $= [0; 2\ ]$. The vertical axis corresponds to the RNTK values for different values of . **Right**: convergence of weight-tied and weight-untied single layer RNN to the same limiting RNTK with increasing width (horizontal axis). The vertical axis corresponds to the average of the log-normalized error between the empirical RNTK computed using finite RNNs and the RNTK for 50 Gaussian normal signals of length $T = 5$.

kernels $^{(t;t+\ )}(x, x^0)$ and $^{(t;t+\ )}(x, x^0)$. We first define the operator $\mathsf{V}\ [K]$ that depends on the nonlinearity $\phi(\ )$ and a positive semi-definite matrix $K\ 2\ \mathsf{R}^{2\ \ 2}$

$$\mathsf{V}\ [K] = \mathsf{E}[\phi(z_1) \cdot \phi(z_2)], \qquad (z_1, z_2)\quad N(0, K). \tag{10}$$

Following (Yang, 2019a), we obtain the analytical recursive formula for the GP kernel $^{(t;t+\ )}(x, x^0)$ for a single layer RNN as

$$^{(1;1)}(x, x^0) = \sigma_w^2\sigma_h^2 1_{(x=x^0)} + \frac{\sigma_u^2}{m}\langle x_1, x_1^0\rangle + \sigma_b^2 \tag{11}$$

$$^{(t;1)}(x, x^0) = \frac{\sigma_u^2}{m}\langle x_t, x_1^0\rangle + \sigma_b^2 \qquad t > 1 \tag{12}$$

$$^{(1;t^0)}(x, x^0) = \frac{\sigma_u^2}{m}\langle x_1, x_{t^0}^0\rangle + \sigma_b^2 \qquad t^0 > 1 \tag{13}$$

$$^{(t;t^0)}(x, x^0) = \sigma_w^2\mathsf{V}\ [K^{(t;t^0)}(x, x^0)] + \frac{\sigma_u^2}{m}\langle x_t, x_{t^0}^0\rangle + \sigma_b^2 \qquad t, t^0 > 1 \tag{14}$$

$$K(x, x^0) = \sigma_v^2\mathsf{V}\ [K^{(T+1;T^0+1)}(x, x^0)], \tag{15}$$

where

$$K^{(t;t^0)}(x, x^0) = \begin{bmatrix} ^{(t\ \ 1;t\ \ 1)}(x, x) & ^{(t\ \ 1;t^0\ \ 1)}(x, x^0) \\ ^{(t\ \ 1;t^0\ \ 1)}(x, x^0) & ^{(t^0\ \ 1;t^0\ \ 1)}(x^0, x^0) \end{bmatrix}. \tag{16}$$

Similarly, we obtain the analytical recursive formula for the GP kernel $^{(t;t+\ )}(x, x^0)$ as

$$^{(T;T^0)}(x, x^0) = \sigma_v^2\mathsf{V}\ _0[K^{(T+1;T+\ \ +1)}(x, x^0)] \tag{17}$$

$$^{(t;t+\ )}(x, x^0) = \sigma_w^2\mathsf{V}\ _0[K^{(t+1;t+\ \ +1)}(x, x^0)]\ \ ^{(t+1;t+1+\ )}(x, x^0) \qquad t\ 2\ [T\ \ \ 1] \tag{18}$$

$$^{(t;t^0)}(x, x^0) = 0 \qquad t^0\ \ t\ \ \tau. \tag{19}$$

For $\phi = \mathsf{ReLU}$ and $\phi = \mathsf{erf}$, we provide analytical expressions for $\mathsf{V}\ [K]$ and $\mathsf{V}\ _0[K]$ in Appendix B.5. These yield an explicit formula for the RNTK that enables fast and point-wise kernel evaluations. For other activation functions, one can apply the Monte Carlo approximation to obtain $\mathsf{V}\ [K]$ and $\mathsf{V}\ _0[K]$ (Novak et al., 2019).

3.2 RNTK for an Infinite-Width RNN during Training

We prove that an infinitely-wide RNN, not only at initialization but also *during* gradient descent training, converges to the limiting RNTK at initialization.

Theorem 2 *Let n be the number of units of each RNN's layer. Assume that (X, X) is positive definite on X such that $\lambda_{\min}(\ (X, X)) > 0$. Let $\eta\ := 2(\lambda_{\min}(\ (X, X)) + \lambda_{\max}(\ (X, X)))^{\ 1}$. For an L-layer RNN with NTK initialization as in (3), (4) trained under gradient flow (recall (1) and (2)) with $\eta < \eta\ $, we have with high probability*

$$\sup_s \frac{k\theta_s\ \ \theta_0 k_2}{\ \ \sqrt{n}}, \sup_s k\hat{\ }_s(X, X)\quad \hat{\ }_0(X, X)k_2 = O\left(\frac{1}{\ \ \sqrt{n}}\right).$$

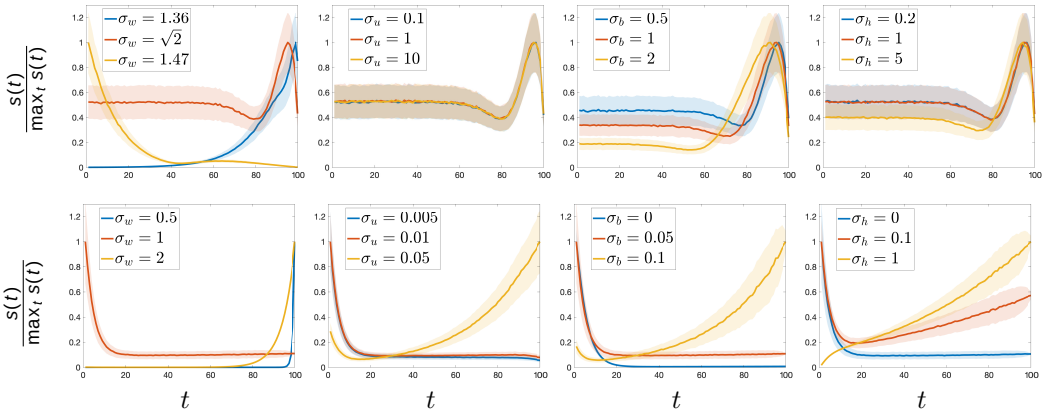

Figure 3: Per time step t (horizontal axis) sensitivity analysis (vertical axis) of the RNTK for the ReLU (top row) and erf (bottom row) activation functions for various weight noise hyperparameters. We also experiment with different RNTK hyperparameters in each of the subplots, given by the subplot internal legend. Clearly, the ReLU (top-row) provides a more stable kernel across time steps (highlighted by the near constant sensitivity through time). On the other hand, erf (bottom row) sees a more erratic behavior either focusing entirely on early time-steps or on the latter ones.

Remarks. Theorem 2 states that the training dynamics of an RNN in the infinite-width limit as in (1), (2) are governed by the RNTK derived from the RNN at its initialization. Intuitively, this is due to the NTK initialization (3), (4) which positions the parameters near a local minima, thus minimizing the amount of update that needs to be applied to the weights to obtain the final parameters.

3.3 RNTK for an Infinite-Width RNN Without Weight Sharing

We prove that, in the infinite-width limit, an RNN without weight sharing (untied weights), i.e., using independent new weights $\mathbf{W}^{(\ell;t)}$, $\mathbf{U}^{(\ell;t)}$ and $\mathbf{b}^{(\ell;t)}$ at each time step t, converges to the same RNTK as an RNN with weight sharing (tied weights). First, recall that it is a common practice to use weight-tied RNNs, i.e., in layer ℓ, the weights $\mathbf{W}^{(\ell)}$, $\mathbf{U}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ are the same across all time steps t. This practice conserves memory and reduces the number of learnable parameters. We demonstrate that, when using untied-weights, the RNTK formula remains unchanged.

Theorem 3 *For inputs of the same length, an RNN with untied weights converges to the same RNTK as an RNN with tied weights in the infinite-width ($n \to \infty$) regime.*

Remarks. Theorem 3 implies that weight-tied and weight-untied RNNs have similar behaviors in the infinite-width limit. It also suggests that existing results on the simpler, weight-untied RNN setting may be applicable for the more general, weight-tied RNN. The plot on the right side of Figure 2 empirically demonstrates the convergence of both the weight-tied and weight-untied RNNs to the RNTK with increasing hidden layer size n; moreover, the convergence rates are similar.

3.4 Insights into the Roles of the RNTK's Hyperparameters

Our analytical form for the RNTK is fully determined by a small number of hyperparameters, which contains the various weight variances collected into $S = \{\sigma_w, \sigma_u, \sigma_b, \sigma_h\}$ and the activation function.[2] In standard supervised-learning settings, one often performs cross-validation to select the hyperparameters. However, since kernel methods become computationally intractable for large datasets, we seek a more computationally friendly alternative to cross-validation. Here we conduct a novel exploratory analysis that provides new insights into the impact of the RNTK hyperparameters on the RNTK output and suggests a simple method to select them a priori in a deliberate manner.

To visualize the role of the RNTK hyperparameters, we introduce the *sensitivity* $s(t)$ of the RNTK of two input sequences x and x^0 with respect to the input x_t at time t

$$s(t) = k_{x_t}(x, x^0)k_2 . \tag{20}$$

[2]From (11) to (18) we emphasize that ν merely scales the RNTK and does not change its overall behavior.

Table 1: Summary of time series classification results on 56 real-world data sets. The RNTK outperforms classical kernels, the NTK, and trained RNNs across all metrics. See Appendix A for detailed description of the metrics.

	RNTK		NTK		RBF		Polynomial		Gaussian RNN		Identity RNN		GRU	
Acc. mean "	**80.15%**	**15.99%**	77.74%	16.61%	78.15%	16.59%	77.69%	16.40%	55.98%	26.42%	63.08%	19.02 %	69.50%	22.67
P90 "	**92.86%**		85.71%		87.60%		82.14%		28.57%		42.86%		60.71%	
P95 "	**80.36%**		66.07%		75.00%		67.86%		17.86%		21.43%		46.43%	
PMA "	**97.23%**		94.30%		94.86%		94.23%		67.06%		78.22%		84.31%	
Friedman Rank #	**2.38**		3.00		2.89		3.46		5.86		5.21		4.21	

Here, $s(t)$ indicates how sensitive the RNTK is to the data at time t, i.e., x_t, in presence of another data sequence x^0. Intuitively, large/small $s(t)$ indicates that the RNTK is relatively sensitive/insensitive to the input x_t at time t.

The sensitivity is crucial to understanding to which extent the RNTK prediction is impacted by the input at each time step. In the case where some time indices have a small sensitivity, then any input variation in those corresponding times will not alter the RNTK output and thus will produce a metric that is invariant to those changes. This situation can be beneficial or detrimental based on the task at hand. Ideally, and in the absence of prior knowledge on the data, one should aim to have a roughly constant sensitivity across time in order to treat all time steps equally in the RNTK input comparison.

Figure 3 plots the normalized sensitivity $s(t)/\max_t(s(t))$ for two data sequences of the same length $T = 100$, with $s(t)$ computed numerically for $x_t, x_t^0 \quad N(0,1)$. We repeated the experiments 10000 times; the mean of the sensitivity is shown in Figure 3. Each of the plots shows the changes of parameters $S_{\mathrm{ReLU}} = \{\sqrt{2}, 1, 0, 0\}$ for $\phi = \mathrm{ReLU}$ and $S_{\mathrm{erf}} = \{1, 0.01, 0.05, 0\}$ for $\phi = \mathrm{erf}$.

From Figure 3 we first observe that both ReLU and erf show similar per time step sensitivity measure $s(t)$ behavior around the hyperparameters S_{ReLU} and S_{erf}. If one varies any of the weight variance parameters, the sensitivity exhibits a wide range of behavior, and in particular with erf. We observe that σ_w has a major influence on $s(t)$. For ReLU, a small decrease/increase in σ_w can lead to over-sensitivity of the RNTK to data at the last/first times steps, whereas for erf, any changes in σ_w leads to over-sensitivity to the last time steps.

Another notable observation is the importance of σ_h, which is usually set to zero for RNNs. (Wang et al., 2018) showed that a non-zero σ_h acts as a regularization that improves the performance of RNNs with the ReLU nonlinearity. From the sensitivity perspective, a non-zero σ_h results in reducing the importance of the first time steps of the input. We also see the same behavior in erf, but with stronger changes as σ_h increases. Hence whenever one aims at reinforcing the input pairwise comparisons, such parameters should be favored.

This sensitivity analysis provides a practical tool for RNTK hyperparameter tuning. In the absence of knowledge about the data, hyperparameters should be chosen to produce the least time varying sensitivity. If given a priori knowledge, hyperparameters can be selected that direct the RNTK to the desired time-steps.

4 Experiments

We now empirically validate the performance of the RNTK compared to classic kernels, NTKs, and trained RNNs on both classification and regression tasks using a large number of time series data sets. Of particular interest is the capability of the RNTK to offer high performance even on inputs of different lengths.

Time Series Classification. The first set of experiments considers time series inputs of the same lengths from 56 datasets in the UCR time-series classification data repository (Dau et al., 2019). We restrict ourselves to selected data sets with fewer than 1000 training samples and fewer than 1000 time steps (T) as kernel methods become rapidly intractable for larger datasets. We compare the RNTK with a variety of other kernels, including the Radial Basis Kernel (RBF), polynomial kernel, and NTK (Jacot et al., 2018), as well as finite RNNs with Gaussian, identity (Le et al., 2015) initialization, and GRU (Cho et al., 2014). We use $\phi = \mathrm{ReLU}$ for both the RNTKs and NTKs. For each kernel, we train a C-SVM (Chang & Lin, 2011) classifier, and for each finite RNN we use gradient descent training. For model hyperparameter tuning, we use 10-fold cross-validation. Details on the data sets and experimental setup are available in Appendix A.1.

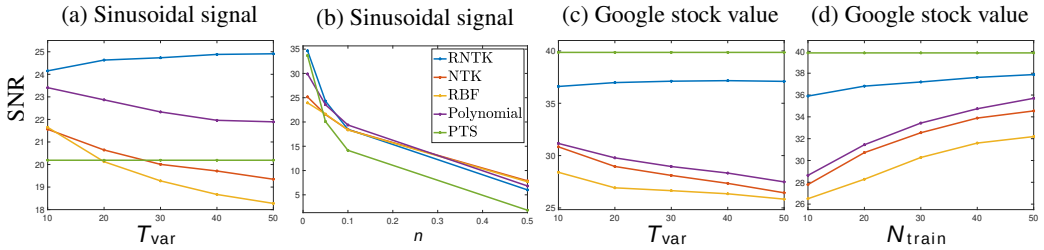

Figure 4: Performance of the RNTK on the synthetic sinusoid and real-world Google stock price data sets compared to three other kernels. We vary the input lengths (a,c), the input noise level (b), and training set size (d). We compute the average SNR by repeating each experiment 1000 times. The RNTK clearly outperforms all of the other kernels under consideration. Figure 4b suggests that the RNTK performs better when input noise level is low demonstrating one case where time recurrence from RNTK might be sub-optimal as it collects and accumulate the high noise from each time step as opposed to other kernels treating each independently.

We summarize the classification results over all 56 datasets in Table 1; detailed results on each data set is available in Appendix A.2. We see that the RNTK outperforms not only the classical kernels but also the NTK and trained RNNs in all metrics. The results demonstrate the ability of RNTK to provide increased performances compare to various other methods (kernels and RNNs). The superior performance of RNTK compared to other kernels, including NTK, can be explained by the internal recurrent mechanism present in RNTK, allowing time-series adapted sample comparison. In addition, RNTK also outperforms RNN and GRU. As the datasets we consider are relative small in size, finite RNNs and GRUs that typically require large amount of data to succeed do not perform well in our setting. An interesting future direction would be to compare RNTK to RNN/GRU on larger datasets.

Time Series Regression. We now validate the performance of the RNTK on time series inputs of *different* lengths on both synthetic data and real data. For both scenarios, the target is to predict the next time-step observation of the randomly extracted windows of different length using kernel ridge regression.

We compare the RNTK to other kernels, the RBF and polynomial kernels and the NTK. We also compare our results with a data independent predictor that requires no training, that is simply to predict the next time step with previous time step (PTS).

For the synthetic data experiment, we simulate 1000 samples of one period of a sinusoid and add white Gaussian noise with default $\sigma_n = 0.05$. From this fixed data, we extract training set size $N_{\text{train}} = 20$ segments of uniform random lengths in the range of $[T_{\text{xed}}, T_{\text{xed}} + T_{\text{var}}]$ with $T_{\text{xed}} = 10$. We use standard kernel ridge regression for this task. The test set is comprised of $N_{\text{test}} = 5000$ obtained from other randomly extracted segments, again of varying lengths. For the real data, we use 975 days of the Google stock value in the years 2014–2018. As in the simulated signal setup above, we extract N_{train} segments of different lengths from the first 700 days and test on the N_{test} segments from days 701 to 975. Details of the experiment are available in Appendix A.2.

We report the predicted signal-to-noise ratio (SNR) for both datasets in Figures 4a and 4c for various values of T_{var}. We vary the noise level and training set size for fixed $T_{\text{var}} = 10$ in Figures 4b and 4d. As we see from Figures 4a and 4c, the RNTK offers substantial performance gains compared to the other kernels, due to its ability to naturally deal with variable length inputs. Moreover, the performance gap increases with the amount of length variation of the inputs T_{var}. Figure 4d demonstrates that, unlike the other methods, the RNTK maintains its performance even when the training set is small. Finally, Figure 4c demonstrates that the impact of noise in the data on the regression performance is roughly the same for all models but becomes more important for RNTK with a large σ_n; this might be attributed to the recurrent structure of the model allowing for a time propagation and amplification of the noise for very low SNR. These experiments demonstrate the distinctive advantages of the RNTK over classical kernels, and NTKs for input data sequences of varying lengths.

In the case of PTS, we expect the predictor to outperform kernel methods when learning from the training samples is hard, due to noise in the data or small training size which can lead to over fitting. In Figure 4a RNTK and Polynomial kernels outperforms PTS for all values of T_{var}, but for larger T_{var}, NTK and RBF under perform PTS due to the increasing detrimental effect of zero padding.

For the Google stock value, we see a superior performance of PTS with respect to all other kernel methods due to the nature of those data heavily relying on close past data. However, RNTK is able to reduce the effect of over-fitting, and provide the closest results to PTS among all kernel methods we employed, with increasing performance as the number of training samples increase.

5 CONCLUSIONS

In this paper, we have derived the RNTK based on the architecture of a simple RNN. We have proved that, at initialization, after training, and without weight sharing, any simple RNN converges to the same RNTK. This convergence provides new insights into the behavior of infinite-width RNNs, including how they process different-length inputs, their training dynamics, and the sensitivity of their output at every time step to different nonlinearities and initializations. We have highlighted the RNTK's practical utility by demonstrating its superior performance on time series classification and regression compared to a range of classical kernels, the NTK, and trained RNNs. There are many avenues for future research, including developing RNTKs for gated RNNs such as the LSTM (Hochreiter & Schmidhuber, 1997) and investigating which of our theoretical insights extend to finite RNNs.

ACKNOWLEDGMENTS

This work was supported by NSF grants CCF-1911094, IIS-1838177, and IIS-1730574; ONR grants N00014-18-12571, N00014-20-1-2787, and N00014-20-1-2534; AFOSR grant FA9550-18-1-0478; and a Vannevar Bush Faculty Fellowship, ONR grant N00014-18-1-2047.

REFERENCES

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *In Advances in Neural Information Processing Systems*, pp. 6155–6166, 2019a.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 6676–6688, 2019b.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pp. 8141–8150. 2019a.

Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019b.

Sanjeev Arora, Simon S. Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkl8sJBYvH.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. URL https://www.pnas.org/content/116/32/15849.

Yoshua. Bengio, Patrice. Simard, and Paolo. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166, 1994.

Erwin Bolthausen. An iterative construction of solutions of the tap equations for the sherrington–kirkpatrick model. *Communications in Mathematical Physics*, 325(1):333–366, 2014.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

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

Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, pp. 342–350, 2009.

Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping Chen, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and ML Hexagon. The UCR time series classification archive, 2019. URL https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

Simon S. Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems*, pp. 5724–5734, 2019a.

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

David Duvenaud, Oren Rippel, Ryan Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. In *Artificial Intelligence and Statistics*, pp. 202–210, 2014.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. URL https://www.sciencedirect.com/science/article/pii/036402139090002E.

M. Fernández-Delgado, M.S. Sirsat, E. Cernadas, S. Alawadi, S. Barro, and M. Febrero-Bande. An extensive experimental survey of regression methods. *Neural Networks*, 111:11 – 34, 2019.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bklfsi0cKm.

Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *International Conference on Machine Learning*, pp. 2034–2042. PMLR, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Wei Hu, Zhiyuan Li, and Dingli Yu. Simple and effective regularization methods for training on noisily labeled data with generalization guarantee. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hke3gyHYwH.

Kaixuan Huang, Yuqing Wang, Molei Tao, and Tuo Zhao. Why do deep residual networks generalize better than deep feedforward networks?–a neural tangent kernel perspective. *arXiv preprint arXiv:2002.06262*, 2020.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning*, pp. 1733–1741, 2017.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Jaehoon Lee, Jascha Sohl-dickstein, Jeffrey Pennington, Roman Novak, Sam Schoenholz, and Yasaman Bahri. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1EA-M-0Z.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, pp. 8572–8583, 2019.

Jaehoon Lee, Samuel S Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *arXiv preprint arXiv:2007.15801*, 2020.

Radford M Neal. *Bayesian Learning for Neural Networks.* PhD thesis, University of Toronto, 1995.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BygfghAcYX.

Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJC2SzZCW.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1g30j0qF7.

Zichao Wang, Randall Balestriero, and Richard Baraniuk. A max-affine spline perspective of recurrent neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJej72AqF7.

Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019a.

Greg Yang. Tensor programs I: Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *arXiv preprint arXiv:1910.12478*, 2019b.

Greg Yang. Tensor programs II: Neural tangent kernel for any architecture. *arXiv preprint arXiv:2006.14548*, 2020a.

Greg Yang. Tensor programs III: Neural matrix laws. *arXiv preprint arXiv:2009.10685*, 2020b.

Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic gradient descent optimizes over-parameterized deep ReLU networks. *arXiv preprint arXiv:1811.08888*, 2018.

A EXPERIMENT DETAILS

A.1 TIME SERIES CLASSIFICATION

Kernel methods settings. We used RNTK, RBF, polynomial and NTK (Jacot et al., 2018). For data pre-processing, we normalized the norm of each x to 1. For training we used C-SVM in `LIBSVM` library (Chang & Lin, 2011) and for hyperparameter selection we performed 10-fold validation for splitting the training data into 90% training set and 10% validation test. We then choose the best performing set of hyperparameters on all the validation sets, retrain the models with the best set of hyperparameters on the entire training data and finally report the performance on the unseen test data. The performance of all kernels on each data set is shown in table 2.

For C-SVM we chose the cost function value

$$C \in \{0.01, 0.1, 1, 10, 100\}$$

and for each kernel we used the following hyperparameter sets

- RNTK: We only used single layer RNTK, we $\phi = \mathrm{ReLU}$ and the following hyperparameter sets for the variances:

$$\sigma_w \in \{1.34, 1.35, 1.36, 1.37, 1.38, 1.39, 1.40, 1.41, 1.42, \sqrt{2}, 1.43, 1.44, 1.45, 1.46, 1.47\}$$
$$\sigma_u = 1$$
$$\sigma_b \in \{0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1, 2\}$$
$$\sigma_h \in \{0, 0.01, 0.1, 0.5, 1\}$$

- NTK: The formula for NTK of L-layer MLP (Jacot et al., 2018) for $x, x' \in \mathbb{R}^m$ is:

$$\Sigma^{(1)} = \frac{\sigma_w^2}{m}\langle x, x'\rangle + \sigma_b^2$$
$$\Sigma^{(\ell)}(x, x') = \sigma_w^2 \mathcal{V}_{[K^{(\ell)}(x,x')]} + \sigma_b^2 \qquad \ell \in [L]$$
$$\dot{\Sigma}^{(\ell)}(x, x') = \sigma_w^2 \mathcal{V}_{\phi[K^{(\ell+1)}(x,x')]} \qquad \ell \in [L]$$
$$K^{(\ell)}(x, x') = \begin{bmatrix} \Sigma^{(\ell-1)}(x, x) & \Sigma^{(\ell-1)}(x, x') \\ \Sigma^{(\ell-1)}(x, x') & \Sigma^{(\ell-1)}(x', x') \end{bmatrix}$$
$$K(x, x') = \sigma_v^2 \mathcal{V}_{[K^{(L+1)}(x,x')]}$$
$$k_{\mathrm{NTK}} = \sum_{\ell=1}^{L} \left(\Sigma^{(\ell)}(x, x') \prod_{\ell'=\ell}^{L} \dot{\Sigma}^{(\ell')}(x, x') \right) + K(x, x')$$

  and we used the following hyperparamters

$$L \in [10]$$
$$\sigma_w \in \{0.5, 1, \sqrt{2}, 2, 2.5, 3\}$$
$$\sigma_b \in \{0, 0.01, 0.1, 0.2, 0.5, 0.8, 1, 2, 5\}$$

- RBF:

$$k_{\mathrm{RBF}}(x, x') = e^{\left(-\frac{\|x - x'\|_2^2}{\alpha}\right)}$$
$$\alpha \in \{0.01, 0.05, 0.1, 0.2, 0.5, 0.6, 0.7, 0.8, 1, 2, 3, 4, 5, 10, 20, 30, 40, 100\}$$

- Polynomial:

$$k_{\mathrm{Polynomial}}(x, x') = (r + \langle x, x'\rangle)^d$$
$$d \in [5]$$
$$r \in \{0, 0.1, 0.2, 0.5, 1, 2\}$$

Table 2: Test accuracy of each model on 56 time series data set from UCR time-series classification data repository (Dau et al., 2019).

Dataset	RNTK	NTK	RBF	POLY	Gaussian RNN	Identity RNN	GRU
Strawberry	**98.38**	97.57	97.03	96.76	94.32	75.4	91.62
ProximalPhalanxOutlineCorrect	**89**	87.97	87.29	86.94	82.81	74.57	86.94
PowerCons	97.22	97.22	96.67	91.67	96.11	95	**99.44**
Ham	70.48	**71.63**	66.67	71.43	53.33	60	60.95
SmallKitchenAppliances	67.47	38.4	40.27	37.87	60.22	**76**	71.46
ScreenType	41.6	43.2	**43.47**	38.4	40	41.06	36.26
MiddlePhalanxOutlineCorrect	57.14	57.14	48.7	64.29	**76.28**	57.04	74.57
RefrigerationDevices	46.93	37.07	36.53	41.07	36	**50.93**	46.66
Yoga	**84.93**	84.63	84.63	84.87	46.43	76.66	61.83
Computers	**59.2**	55.2	58.8	56.4	53.2	55.2	58.8
ECG5000	93.76	**94.04**	93.69	93.96	88.4	93.15	93.26
Fish	**90.29**	84	85.71	88	28	38.28	24
UWaveGestureLibraryX	**79.59**	78.7	78.48	65.83	55.97	75.34	73.64
UWaveGestureLibraryY	**71.56**	70.63	70.35	70.32	44.5	65.18	65.38
UWaveGestureLibraryZ	**73.95**	73.87	72.89	71.94	43.29	67.81	70.32
StarLightCurves	95.94	**96.19**	94.62	94.44	82.13	86.81	96.15
CricketX	60.51	59.49	62.05	62.56	8.46	**63.58**	26.41
CricketY	**63.85**	58.97	60.51	59.74	15.89	59.23	36.15
CricketZ	**60.26**	59.23	62.05	59.23	8.46	57.94	41.28
DistalPhalanxOutlineCorrect	**77.54**	**77.54**	75.36	73.91	69.92	69.56	75
Worms	**57.14**	50.65	55.84	50.65	35.06	49.35	41.55
SyntheticControl	98.67	96.67	98	97.67	92.66	97.66	**99**
Herring	56.65	**59.38**	**59.38**	**59.38**	23.28	**59.37**	**59.38**
MedicalImages	74.47	73.29	**75.26**	74.61	48.15	64.86	69.07
SwedishLeaf	90.56	91.04	**91.36**	90.72	59.2	45.92	91.04
ChlorineConcentration	90.76	77.27	86.35	**91.54**	65.99	55.75	61.14
SmoothSubspace	**96**	87.33	92	86.67	94	95.33	92.66
TwoPatterns	94.25	90.45	91.25	93.88	99.7	99.9	**100**
Faceall	74.14	**83.33**	83.25	82.43	53.66	70.53	70.65
DistalPhalanxTW	66.19	**69.78**	66.91	67.37	67.62	64.74	69.06
MiddlePhalanxTW	57.79	**61.04**	59.74	60.39	58.44	58.44	59.09
FacesUCR	81.66	80.2	80.34	**82.98**	53.21	75.26	79.46
OliveOil	**90**	86.67	86.67	83.33	66.66	40	40
UMD	91.67	92.36	97.22	90.97	44.44	71.52	**100**
nsectEPGRegular	99.6	99.2	99.6	96.79	**100**	**100**	98.39
Meat	**93.33**	**93.33**	**93.33**	**93.33**	0.55	55	33.33
Lightning2	**78.69**	73.77	70.49	68.85	45.9	70.49	67.21
Lightning7	61.64	60.27	63.01	60.27	23.28	69.86	**76.71**
Car	**83.33**	78.83	80	80	23.33	58.33	26.66
GunPoint	**98**	95.33	95.33	94	82	74.66	80.66
Arrowhead	80.57	**83.43**	80.57	74.86	48	56	37.71
Coffee	**100**	**100**	92.86	92.86	**100**	42.85	57.14
Trace	96	81	76	76	70	71	**100**
ECG200	**93**	89	89	86	86	72	76
plane	**98.1**	96.19	97.14	97.14	96.19	84.76	96.19
GunPointOldVersusYoung	**98.73**	97.46	**98.73**	94.6	53.96	52.38	98.41
GunPointMaleVersusFemale	99.05	**99.68**	99.37	**99.68**	68.67	52.53	99.68
GunPointAgeSpan	**96.52**	94.62	95.89	93.99	47.78	47.78	95.56
FreezerRegularTrain	**97.44**	94.35	96.46	96.84	76.07	7.5	86.59
SemgHandSubjectCh2	84.22	85.33	86.14	86.67	20	36.66	**89.11**
WormsTwoClass	**62.34**	**62.34**	61.04	59.74	51.94	46.75	57.14
Earthquakes	74.82	74.82	74.82	74.82	65.46	**76.97**	**76.97**
FiftyWords	68.57	68.57	**69.67**	68.79	34.28	60.21	65.27
Beef	90	73.33	83.33	**93.33**	26.67	46.67	36.67
Adiac	76.63	71.87	73.40	**77.75**	51.4	16.88	60.61
WordSynonyms	57.99	58.46	61.13	**62.07**	17.71	45.77	53.76

Finite-width RNN settings. We used 3 different RNNs. The first is a ReLU RNN with Gaussian initialization with the same NTK initialization scheme, where parameter variances are $\sigma_w = \sigma_v = \sqrt{2}$,

$\sigma_u = 1$ and $\sigma_b = 0$. The second is a ReLU RNN with identity initialization following (Le et al., 2015). The third is a GRU (Cho et al., 2014) with uniform initialization. All models are trained with RMSProp algorithm for 200 epochs. Early stopping is implemented when the validation set accuracy does not improve for 5 consecutive epochs.

We perform standard 5-fold cross validation. For each RNN architecture we used hyperparamters of number of layer, number of hidden units and learning rate as

$$L \in \{1, 2\}$$
$$n \in \{50, 100, 200, 500\}$$
$$\eta \in \{0.01, 0.001, 0.0001, 0.00001\}$$

Metrics descriptions First, only in this paragraph, let $i \in \{1, 2, ..., N\}$ index a total of N datasets and $j \in \{1, 2, ..., M\}$ index a total of M classifiers. Let y_{ij} be the accuracy of the j-th classifer on the i-th dataset. We reported results on 4 metrics: average accuracy (Acc. mean), P90, P95, PMA and Friedman Rank. P90 and P95 is the fraction of datasets that the classifier achieves at least 90% and 95% of the maximum achievable accuracy for each dataset, i.e.,

$$P90_j = \frac{1}{N} \sum_i \mathbf{1}(y_{ij} \quad 0.9(\max_j y_{ij})) \,. \tag{21}$$

PMA is the accuracy of the classifier on a dataset divided by the maximum achievable accuracy on that dataset, averaged over all datasets:

$$\text{PMA}_j = \frac{1}{N} \sum_i \frac{y_{ij}}{\max_j y_{ij}} \,. \tag{22}$$

Friedman Rank (Fernández-Delgado et al., 2019) first ranks the accuracy of each classifier on each dataset and then takes the average of the ranks for each classifier over all datasets, i.e.,

$$\text{FR}_j = \frac{1}{N} \sum_i r_{ij} \,, \tag{23}$$

where r_{ij} is the ranking of the j-th classifier on the i-th dataset.

Note that a better classifier achieves a lower Friedman Rank, Higher P/90 and PMA.

Remark. In order to provide insight into the performance of RNTK in long time steps setting, we picked two datasets with more that 1000 times steps: SemgHandSubjectCh2 ($T = 1024$) and StarLightCurves ($T = 1024$).

A.2 Time Series Regression

For time series regression, we used the 5-fold validation of training set and same hyperparamter sets for all kernels. For training we kernel ridge regression with ridge term chosen form

$$\lambda \in \{0, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.8, 1, 2, 3, 4, 5, 6, 7, 8, 10, 100\}$$

B Proofs for Theorems 1 and 3: RNTK Convergence at Initialization

B.1 Preliminary: Netsor programs

Calculation of NTK in any architecture relies on finding the GP kernels that correspond to each pre-activation and gradient layers at initialization. For feedforward neural networks with $n_1, \ldots, n_L$ number of neurons (channels in CNNs) at each layer the form of this GP kernels can be calculated via taking the limit of $n_1, \ldots, n_L$ sequentially one by one. The proof is given by induction, where by conditioning on the previous layers, each entry of the current layer is sum of infinite i.i.d Gaussian random variables, and based on Central Limit Theorem (CLT), it becomes a Gaussian process with

kernel calculated based on the previous layers. Since the first layer is an affine transformation of input with Gaussian weights, it is a Gaussian process and the proof is completed. See (Lee et al., 2018; Duvenaud et al., 2014; Novak et al., 2019; Garriga-Alonso et al., 2019) for a formal treatment. However, due to weight-sharing, sequential limit is not possible and condoning on previous layers does not result in i.i.d. weights. Hence the aforementioned arguments break. To deal with it, in (Yang, 2019a) a proof using Gaussian conditioning trick (Bolthausen, 2014) is presented which allows use of recurrent weights in a network. More precisely, it has been demonstrated than neural networks (without batch normalization) can be expressed and a series of matrix multiplication and (piece wise) nonlinearity application, generally referred as *Netsor programs*. It has been shown that any architecture that can be expressed as *Netsor programs* that converge to GPs as width goes to infinity in the same rate, which a general rule to obtain the GP kernels. For completeness of this paper, we briefly restate the results from (Yang, 2019a) which we will use later for calculation derivation of RNTK.

There are 3 types of variables in *Netsor programs*; A-vars, G-vars and H-vars. A-vars are matrices and vectors with i.i.d Gaussian entries, G-vars are vectors introduced by multiplication of a vector by an A-var and H-vars are vectors after coordinate wise nonlinearities is applied to G-vars. Generally, G-vars can be thought of as pre-activation layers which are asymptotically treated as a Gaussian distributed vectors, H-vars as after-activation layers and A-vars are the weights. Since in neural networks inputs are immediately multiplied by a weight matrix, it can be thought of as an G-var, namely g_{in}. Generally *Netsor programs* supports G-vars with different dimension, however the asymptotic behavior of a neural networks described by *Netsor programs* does not change under this degree of freedom, as long as they go to infinity at the same rate. For simplicity, let the G-vars and H-vars have the same dimension n since the network of interest is RNN and all pre-activation layers have the same dimension. We introduce the *Netsor programs* under this simplification. To produce the output of a neural network, *Netsor programs* receive a set of G-vars and A-vars as input, and new variables are produced sequentially using the three following operators:

- `Matmul` : multiplication of an A-var: A with an H-var: h, which produce a new G-var, g.
$$g = Ah \tag{24}$$
- `Lincomp`: Linear combination of G-vars, $g^i,\ 1 \quad i \quad k$, with coefficients $a^i \ 2 \ \mathbb{R}$ $1 \quad i \quad k$ which produce of new G-var:
$$g = \sum_{i=1}^{k} a^i g^i \tag{25}$$
- `Nonlin`: creating a new H-var, $\mathbf{h}$, by using a nonlinear function $\phi : \mathbb{R}^k \ ! \ \mathbb{R}$ that act coordinate wise on a set of G-vars, $g^i,\ 1 \quad i \quad k$:
$$h = \varphi(g^1, \ldots, g^k) \tag{26}$$

Any output of the neural network $y \ 2 \ \mathbb{R}$ should be expressed as inner product of a new A-var which has not been used anywhere else in previous computations and an H-var:

$$y = v^\top h$$

Any other output can be produced by another v^0 and h^0 (possibility the same h or v).

It is assumed that each entry of any A-var : $A \ 2 \ \mathbb{R}^n \ \ ^n$ in the *netsor programs* computations is drawn from $\mathcal{N}(0, \frac{2}{n})$ and the input G-vars are Gaussian distributed. The collection of a specific entry of all G-vars of in the *netsor program* converges in probability to a Gaussian vector $f[g^1]_i, \ldots, [g^k]_i g$ $\mathcal{N}(\ , \Sigma)$ for all $i \ 2 \ [n]$ as n goes to infinity.
Let $\mu(g) := \mathbb{E}\big[[g]_i\big]$ be the mean of a G-var and $\quad (g, g^0) := \mathbb{E}\big[[g]_i \cdot [g^0]_i\big]$ be the covariance between any two G-vars. The general rule for $\mu(g)$ is given by the following equations:

$$\mu(g) = \begin{cases} \mu^{in}(g) & \text{if } g \text{ is input} \\ \sum_{i=1}^{k} a^i \mu(g^i) & \text{if } g = \sum_{i=1}^{k} a^i g^i \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

For g and g', let $G = \{g^1, \ldots, g^r\}$ be the set of G-vars that has been introduced *before* g and g' with distribution $N(\ _G, \Sigma_G)$, where $\Sigma_G \in \mathbb{R}^{|G|\times|G|}$ containing the pairwise covariances between the G-vars. (g, g') is calculated via the following rules:

$$(g, g') = \begin{cases} {}^{\text{in}}(g, g') & \text{if } g \text{ and } g' \text{ are inputs} \\ \sum_{i=1}^{k} a^i \ (g^i, g') & \text{if } g = \sum_{i=1}^{k} a^i g^i \\ \sum_{i=1}^{k} a^i \ (g, g^i) & \text{if } g' = \sum_{i=1}^{k} a^i g^i \\ \sigma_A^2 \ \mathop{\mathbb{E}}_{\mathbf{z} \sim N(\ ,\Sigma_G)}[\varphi(\mathbf{z})\varphi(\mathbf{z})] & \text{if } g = Ah \text{ and } g' = Ah' \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

Where $h = \varphi(g^1, \ldots, g^r)$ and $h' = \varphi(g^1, \ldots, g^r)$ are functions of G-vars in G from possibly different nonlinearities. This set of rules presents a recursive method for calculating the GP kernels in a network where the recursive formula starts from data dependent quantities ${}^{\text{in}}$ and μ^{in} which are given.

All the above results holds when the nonlinearities are bounded uniformly by $e^{(cx^2 \quad)}$ for some $\alpha > 0$ and when their derivatives exist.

Standard vs. NTK initialization. The common practice (which *netsor programs* uses) is to initialize DNNs weights $[A]_{i;j}$ with $N(0, \frac{a}{n})$ (known as *standard* initialization) where generally n is the number of units in the previous layer. In this paper we have used a different parameterization scheme as used in (Jacot et al., 2018) and we factor the standard deviation as shown in 3 and initialize weights with standard standard Gaussian. This approach does not change the the forward computation of DNN, but normalizes the backward computation (when computing the gradients) by factor $\frac{1}{n}$, otherwise RNTK will be scales by n. However this problem can be solved by scaling the step size by $\frac{1}{n}$ and there is no difference between *NTK* and *standard* initialization (Lee et al., 2019).

B.2 Proof for Theorem 1: Single layer Case

We first derive the RNTK in a simpler setting, i.e., a single layer and single output RNN. We then generalize the results to multi-layer and multi-output RNNs. We drop the layer index ℓ to simplify notation. From 3 and 4, the forward pass for computing the output under NTK initialization for each input $x = \{x_t\}_{t=1}^{T}$ is given by:

$$g^{(t)}(x) = \frac{\sigma_w}{m}\mathbf{W}h^{(t\ \ 1)}(x) + \frac{\sigma_u}{n}\mathbf{U}x_t + \sigma_b\mathbf{b} \tag{29}$$

$$h^{(t)}(x) = \phi\left(g^{(t)}(x)\right) \tag{30}$$

$$f\ (x) = \frac{\sigma_v}{n}v^{\top}h^{(T)}(x) \tag{31}$$

Note that (29), (30) and (31) use all the introduced operators introduced in 24, 25 and 26 given input variables $\mathbf{W}$, $\{\mathbf{U}x_t\}_{t=1}^{T}$, $\mathbf{b}$, v and $h^{(0)}(x)$.

First, we compute the kernels of forward pass ${}^{(t;t')}(x, x')$ and backward pass ${}^{(t;t')}(x, x')$ introduced in (6) and (7) for two input x and x'. Note that based on (27) the mean of all variables is zero since the inputs are all zero mean. In the forward pass for the intermediate layers we have:

$$\begin{aligned} {}^{(t;t')}(x, x') &= \ (g^{(t)}(x), g^{(t')}(x')) \\ &= \left(\frac{\sigma_w}{n}\mathbf{W}h^{(t\ \ 1)}(x) + \frac{\sigma_u}{m}\mathbf{U}x_t + \sigma_b\mathbf{b}, \frac{\sigma_w}{n}\mathbf{W}h^{(t'\ \ 1)}(x') + \frac{\sigma_u}{m}\mathbf{U}x'_{t'} + \sigma_b\mathbf{b}\right) \\ &= \left(\frac{\sigma_w}{n}\mathbf{W}h^{(t\ \ 1)}(x), \frac{\sigma_w}{n}\mathbf{W}h^{(t'\ \ 1)}(x')\right) + {}^{\text{in}}\left(\frac{\sigma_u}{m}\mathbf{U}x_t, \frac{\sigma_u}{m}\mathbf{U}x'_{t'}\right) + {}^{\text{in}}(\sigma_b b, \sigma_b b)\,. \end{aligned}$$

We have used the second and third rule in (28) to expand the formula, We have also used the first and fifth rule to set the cross term to zero, i.e.,

$$\left(\frac{\sigma_w}{\cancel{p}\overline{n}}\mathbf{W}h^{(t\ \ 1)}(x), \frac{\sigma_u}{\cancel{p}\overline{m}}\mathbf{U}x^0_{t^0}\right) = 0$$
$$\left(\frac{\sigma_w}{\cancel{p}\overline{n}}\mathbf{W}h^{(t\ \ 1)}(x), \sigma_b\mathbf{b}\right) = 0$$
$$\left(\frac{\sigma_u}{\cancel{p}\overline{m}}\mathbf{U}x_t, \frac{\sigma_w}{\cancel{p}\overline{n}}\mathbf{W}h^{(t^0\ \ 1)}(x^0)\right) = 0$$
$$\left(\sigma_b\mathbf{b}, \frac{\sigma_w}{\cancel{p}\overline{n}}\mathbf{W}h^{(t^0\ \ 1)}(x^0)\right) = 0$$
$$\text{in}\left(\frac{\sigma_u}{\cancel{p}\overline{m}}\mathbf{U}x_t, \sigma_b\mathbf{b}\right) = 0$$
$$\text{in}\left(\sigma_b\mathbf{b}, \frac{\sigma_u}{\cancel{p}\overline{m}}\mathbf{U}x^0_{t^0}\right) = 0.$$

For the non-zero terms we have

$$\text{in}\,(\sigma_b b, \sigma_b b) = \sigma_b^2$$
$$\text{in}\left(\frac{\sigma_u}{\cancel{p}\overline{m}}\mathbf{U}x_t, \frac{\sigma_u}{\cancel{p}\overline{m}}\mathbf{U}x^0_{t^0}\right) = \frac{\sigma_u^2}{m}hx_t, x^0_{t^0}i,$$

which can be achieved by straight forward computation. If $t \cancel{6} 1$ and $t^0 \cancel{6} 1$, by using the forth rule in (28) we have

$$\left(\frac{\sigma_w}{\cancel{p}\overline{n}}\mathbf{W}h^{(t\ \ 1)}(x), \frac{\sigma_w}{\cancel{p}\overline{n}}\mathbf{W}h^{(t^0\ \ 1)}(x^0)\right) = \sigma_w^2 \underset{\mathbf{z}\ \ N(0;K^{(t;t^0)}(x;x^0))}{\mathsf{E}}[\phi(z_1)\phi(z_2)] = \mathsf{V}\ \left[K^{(t;t^0)}(x,x^0)\right].$$

With $K^{(t;t^0)}(x,x^0)$ defined in (16). Otherwise, it will be zero by the fifth rule (if t or $t = 1$) .

Here the set of previously introduced G-vars is $G = \left\{fg^{(\ \ )}(x)g, \mathbf{U}x\ \ g^{t\ \ 1}_{=1}, fg^{(\ \ ^0)}(x^0), \mathbf{U}x^0{}_0g^{t^0}{}_{0=1}, h^{(0)}(x), h^{(0)}(x^0)\right\}$, but the dependency is only on the last layer G-vars, $\varphi(fg : g\ 2\ Gg) = \phi(g^{(t\ \ 1)}(x)), \varphi((fg : g\ 2\ Gg)) = \phi(g^{(t^0\ \ 1)}(x^0))$, leading the calculation to the operator defined in (10). As a result

$$^{(t;t^0)}(x,x^0) = \sigma_w^2\mathsf{V}\ \left[K^{(t;t^0)}(x,x^0)\right] + \frac{\sigma_u^2}{m}hx_t, x^0_{t^0}i + \sigma_b^2.$$

To complete the recursive formula, using the same procedure for the first layers we have

$$^{(1;1)}(x,x^0) = \sigma_w^2\sigma_h^2 1_{(x=x^0)} + \frac{\sigma_u^2}{m}hx_1, x^0_1i + \sigma_b^2,$$
$$^{(1;t^0)}(x,x^0) = \frac{\sigma_u^2}{m}hx_1, x^0_{t^0}i + \sigma_b^2,$$
$$^{(t;1)}(x,x^0) = \frac{\sigma_u^2}{m}hx_t, x^0_1i + \sigma_b^2.$$

The output GP kernel is calculated via

$$K(x,x^0) = \sigma_v^2\mathsf{V}\ \left[K^{(T+1;T^0+1)}(x,x^0)\right]$$

The calculation of the gradient vectors $^{(t)}(x) = \overline{p}{}_{\overline{n}}\left(r_{g^{(t)}(x)}f\ (x)\right)$ in the backward pass is given by

$$^{(T)}(x) = \sigma_v v\quad \phi^0(g^{(T)}(x))$$
$$^{(t)}(x) = \frac{\sigma_w}{\cancel{p}\overline{n}}\mathbf{W}^{>}\left(\phi^0(g^{(t)}(x))\quad ^{(t+1)}(x)\right)\qquad t\ 2\ [T\quad 1]$$

To calculate the backward pass kernels, we rely on the following Corollary from (Yang, 2020b)

Corollary 1 *In infinitely wide neural networks weights used in calculation of back propagation gradients ($\mathbf{W}^{>}$) is an i.i.d copy of weights used in forward propagation ($\mathbf{W}$) as long as the last layer weight (v) is sampled independently from other parameters and has mean 0.*

The immediate result of Corollary 1 is that $g^{(t)}(x)$ and $^{(t)}(x)$ are two independent Gaussian vector as their covariance is zero based on the fifth rule in (28). Using this result, we have:

$$
\begin{aligned}
^{(t;t^0)}(x,x^0) &= \left({}^{(t)}(x),\ {}^{(t^0)}(x) \right) \\
&= \mathsf{E}\left[[\ {}^{(t)}(x)]_i \cdot [\ {}^{(t^0)}(x^0)]_i \right] \\
&= \sigma_w^2 \mathsf{E}\left[[\phi^0(g^{(t)}(x))]_i \cdot [\ {}^{(t+1)}(x)]_i \cdot [\phi^0(g^{(t^0)}(x^0))]_i \cdot [\ {}^{(t^0+1)}(x^0)]_i \right] \\
&= \sigma_w^2 \underset{\mathbf{z}\ \ N(0;K^{(t+1;t+1^0)}(x;x^0))}{\mathsf{E}} [\phi^0(\mathrm{z}_1)\cdot\phi^0(\mathrm{z}_2)] \cdot \mathsf{E}\left[[\ {}^{(t+1)}(x)]_i \cdot [\ {}^{(t^0+1)}(x^0)]_i \right] \\
&= \sigma_w^2 \mathsf{V}\ {}_0\left[K^{(t+1;t^0+1)}(x,x^0) \right]\ \ {}^{(t+1;t^0+1)}(x,x^0).
\end{aligned}
$$

If $T^0\ \ t^0 = T\ \ t$, then the the formula will lead to

$$
\begin{aligned}
^{(T;T^0)}(x,x^0) &= \mathsf{E}\left[[\ {}^{(T)}(x)]_i, [\ {}^{(T^0)}(x^0)]_i \right] \\
&= \sigma_v^2 \mathsf{E}\left[[v]_i \cdot [\phi^0(g^{(T)}(x))]_i \cdot [v]_i \cdot [\phi^0(g^{(T^0)}(x^0))]_i \right] \\
&= \mathsf{E}\left[[\phi^0(g^{(T)}(x))]_i \cdot [\phi^0(g^{(T^0)}(x^0))]_i \right] \cdot \mathsf{E}\left[[v]_i\, [v]_i \right] \\
&= \sigma_v^2 \mathsf{V}\ {}_0\left[K^{(T+1;T+\ \ +1)}(x,x^0) \right].
\end{aligned}
$$

Otherwise it will end to either of two cases for some $t^{00} < T$ or T^0 and by the fifth rule in (28) we have:

$$
\begin{aligned}
\left({}^{(t^{00})}(x),\ {}^{(T^0)}(x) \right) &= \left(\frac{\sigma_w}{\sqrt{n}} \mathbf{W}^{>} \left(\phi^0(g^{(t^{00})}(x))\ \ {}^{(t^{00}+1)}(x^0) \right), v\ \ \phi^0(g^{(T^0)}(x)) \right) = 0 \\
\left({}^{(T)}(x),\ {}^{(t^{00})}(x) \right) &= \left(v\ \ \phi^0(g^{(T)}(x)), \frac{\sigma_w}{\sqrt{n}} \mathbf{W}^{>} \left(\phi^0(g^{(t^{00})}(x^0))\ \ {}^{(t^{00}+1)}(x^0) \right) \right) = 0.
\end{aligned}
$$

Without loss of generality, from now on assume $T^0 < T$ and $T^0\ \ T = \tau$, the final formula for computing the backward gradients becomes:

$$
\begin{aligned}
^{(T;T+\ \ )}(x,x^0) &= \sigma_v^2 \mathsf{V}\ {}_0\left[K^{(T+1;T+\ \ +1)}(x,x^0) \right] \\
^{(t;t+\ \ )}(x,x^0) &= \sigma_w^2 \mathsf{V}\ {}_0\left[K^{(t+1;t+\ \ +1)}(x,x^0) \right]\ \ {}^{(t+1;t+1+\ \ )}(x,x^0) \qquad t\ 2\ [T\ \ 1] \\
^{(t;t^0)}(x,x^0) &= 0 \qquad\qquad t^0\ \ t\ \cancel{6}\ \tau
\end{aligned}
$$

Now we have derived the single layer RNTK. Recall that $\theta = \mathsf{Vect}\left[f\mathbf{W}, \mathbf{U}, \mathbf{b}, vg \right]$ contains all of the network's learnable parameters. As a result, we have:

$$
r\ \ f\ (x) = \mathsf{Vect}\left[f \frac{\partial f\ (x)}{\partial \mathbf{W}}, \frac{\partial f\ (x)}{\partial \mathbf{U}}, \frac{\partial f\ (x)}{\partial \mathbf{b}}, \frac{\partial f\ (x)}{\partial v} g \right].
$$

As a result

$$
\begin{aligned}
h\, r\ \ f\ (x), r\ \ f\ (x^0) i = &\left\langle \frac{\partial f\ (x)}{\partial \mathbf{W}}, \frac{\partial f\ (x^0)}{\partial \mathbf{W}} \right\rangle + \left\langle \frac{\partial f\ (x)}{\partial \mathbf{U}}, \frac{\partial f\ (x^0)}{\partial \mathbf{U}} \right\rangle + \left\langle \frac{\partial f\ (x)}{\partial \mathbf{b}}, \frac{\partial f\ (x^0)}{\partial \mathbf{b}} \right\rangle \\
&+ \left\langle \frac{\partial f\ (x)}{\partial v}, \frac{\partial f\ (x^0)}{\partial v} \right\rangle
\end{aligned}
$$

Where the gradients of output with respect to weights can be formulated as the following compact form:

$$\frac{\partial f(x)}{\partial \mathbf{W}} = \sum_{t=1}^{T} \left(\frac{1}{\sqrt{n}} \dot{h}^{(t)}(x) \right) \cdot \left(\frac{\sigma_w}{\sqrt{n}} h^{(t-1)}(x) \right)^\top$$

$$\frac{\partial f(x)}{\partial \mathbf{U}} = \sum_{t=1}^{T} \left(\frac{1}{\sqrt{n}} \dot{h}^{(t)}(x) \right) \cdot \left(\frac{\sigma_u}{\sqrt{m}} x_t \right)^\top$$

$$\frac{\partial f(x)}{\partial \mathbf{b}} = \sum_{t=1}^{T} \left(\frac{\sigma_b}{\sqrt{n}} \dot{h}^{(t)}(x) \right)$$

$$\frac{\partial f(x)}{\partial v} = \frac{\sigma_v}{\sqrt{n}} h^{(T)}(x).$$

As a result we have:

$$\left\langle \frac{\partial f(x)}{\partial \mathbf{W}}, \frac{\partial f(x')}{\partial \mathbf{W}} \right\rangle = \sum_{t'=1}^{T'} \sum_{t=1}^{T} \left(\frac{1}{n} \left\langle \dot{h}^{(t)}(x), \dot{h}^{(t')}(x') \right\rangle \right) \cdot \left(\frac{\sigma_w^2}{n} \left\langle h^{(t-1)}(x), h^{(t'-1)}(x') \right\rangle \right)$$

$$\left\langle \frac{\partial f(x)}{\partial \mathbf{U}}, \frac{\partial f(x')}{\partial \mathbf{U}} \right\rangle = \sum_{t'=1}^{T'} \sum_{t=1}^{T} \left(\frac{1}{n} \left\langle \dot{h}^{(t)}(x), \dot{h}^{(t')}(x') \right\rangle \right) \cdot \left(\frac{\sigma_u^2}{m} \langle x_t, x'_{t'} \rangle \right)$$

$$\left\langle \frac{\partial f(x)}{\partial \mathbf{b}}, \frac{\partial f(x')}{\partial \mathbf{b}} \right\rangle = \sum_{t'=1}^{T'} \sum_{t=1}^{T} \left(\frac{1}{n} \left\langle \dot{h}^{(t)}(x), \dot{h}^{(t')}(x') \right\rangle \right) \cdot \sigma_b^2$$

$$\left\langle \frac{\partial f(x)}{\partial v}, \frac{\partial f(x')}{\partial v} \right\rangle = \left(\frac{\sigma_v^2}{n} \left\langle h^{(T)}(x), h^{(T')}(x') \right\rangle \right).$$

Remember that for any two G-var $\mathsf{E}[[g]_i[g']_i]$ is independent of index i. Therefore,

$$\frac{1}{n} \left\langle h^{(t-1)}(x), h^{(t'-1)}(x') \right\rangle \to \mathsf{V}\left[K^{(t;t')}(x, x') \right] \qquad t > 1$$

$$\frac{1}{n} \left\langle h^{(0)}(x), h^{(0)}(x') \right\rangle \to \sigma_h^2.$$

Hence, by summing the above terms in the infinite-width limit we get

$$\Theta^{(f)}(x, x') \to \left(\sum_{t'=1}^{T'} \sum_{t=1}^{T} \Theta^{(t;t')}(x, x') \cdot \dot{\Theta}^{(t;t')}(x', x') \right) + K(x, x'). \tag{32}$$

Since $\Theta^{(t;t')}(x, x') = 0$ for $t' \neq t$ it is simplified to

$$\Theta^{(f)}(x, x') = \left(\sum_{t=1}^{T} \Theta^{(t;t)}(x, x') \cdot \dot{\Theta}^{(t;t)}(x', x') \right) + K(x, x').$$

Multi-dimensional output. For $f(x) \in \mathbb{R}^d$, the i-th output for $i \in [d]$ is obtained via

$$[f(x)]_i = \frac{\sigma_v}{\sqrt{n}} v_i^\top h^{(T)}(x),$$

where v_i is independent of v_j for $i \neq j$. As a result, for The RNTK $\Theta(x, x') \in \mathbb{R}^{d \times d}$ for multi-dimensional output we have

$$[\Theta(x, x')]_{i;j} = \left\langle \nabla [f(x)]_i, \nabla [f(x')]_j \right\rangle$$

For $i = j$, the kernel is the same as computed in (32) and we denote it as

$$\Theta\left\langle \nabla [f(x)]_i, \nabla [f(x')]_i \right\rangle = \Theta^{(T;T')}(x, x').$$

For $i \neq j$, since v_i is independent of v_j, $\quad^{(T;T^0)}(x, x^0)$ and all the backward pass gradients become zero, so

$$\left\langle r \quad [f\ (x)]_i\ ,\ r \quad [f\ (x^0)]_j \right\rangle = 0 \qquad i \neq j$$

which gives us the following formula

$$(x, x^0) = \quad^{(T;T^0)}(x, x^0) \quad I_d.$$

This concludes the proof for Theorem 1 for single-layer case.

B.3 Proof for Theorem 1: Multi-Layer Case

Now we drive the RNTK for multi-layer RNTK. We will only study single output case and the generalization to multi-dimensional case is identical as the single layer case. The set of equations for calculation of the output of a L-layer RNN for $x = \{x_t\}_{t=1}^T$ are

$$\begin{aligned}
g^{(\ell;t)}(x) &= \frac{\sigma_w}{\sqrt{n}}\mathbf{W}^{(\ell)}h^{(\ell;t-1)}(x) + \frac{\sigma_u}{\sqrt{m}}\mathbf{U}^{(\ell)}x_t + \sigma_b\mathbf{b}^{(\ell)} && \ell = 1\\
g^{(\ell;t)}(x) &= \frac{\sigma_w}{\sqrt{n}}\mathbf{W}^{(\ell)}h^{(\ell;t-1)}(x) + \frac{\sigma_u}{\sqrt{n}}\mathbf{U}^{(\ell)}h^{(\ell-1;t)}(x) + \sigma_b\mathbf{b}^{(\ell)} && \ell > 1\\
h^{(\ell;t)}(x) &= \phi\left(g^{(\ell;t)}(x)\right)\\
f\ (x) &= \frac{\sigma_v}{\sqrt{n}}v^\top h^{(L;T)}(x)
\end{aligned}$$

The forward pass kernels for the first layer is the same as calculated in B.2. For $\ell \quad 2$ we have:

$$\begin{aligned}
^{(\ell;t;t^0)}(x, x^0) &= \quad (g^{(\ell;t)}(x), g^{(\ell;t^0)}(x^0))\\
&= \quad \left(\frac{\sigma_w}{\sqrt{n}}\mathbf{W}^{(\ell)}h^{(\ell;t-1)}(x), \frac{\sigma_w}{\sqrt{n}}\mathbf{W}^{(\ell)}h^{(\ell;t^0-1)}(x^0)\right)\\
&+ \quad \left(\frac{\sigma_u}{\sqrt{n}}\mathbf{U}^{(\ell)}h^{(\ell-1;t)}(x), \frac{\sigma_u}{\sqrt{n}}\mathbf{U}^{(\ell)}h^{(\ell-1;t^0)}(x^0)\right) + \quad^{\text{in}}\left(\sigma_b\mathbf{b}^{(\ell)}, \sigma_b\mathbf{b}^{(\ell)}\right)\\
&= (\sigma_w)^2\mathbb{V}\ \left[K^{(\ell;t;t^0)}(x, x^0)\right] + (\sigma_u)^2\mathbb{V}\ \left[K^{(\ell-1;t+1;t^0+1)}(x, x^0)\right] + (\sigma_b)^2,
\end{aligned}$$

where

$$K^{(\ell;t;t^0)}(x, x^0) = \begin{bmatrix} \Sigma^{(\ell;t-1;t-1)}(x, x) & \Sigma^{(\ell;t-1;t^0-1)}(x, x^0)\\ \Sigma^{(\ell;t-1;t^0-1)}(x, x^0) & \Sigma^{(\ell;t^0-1;t^0-1)}(x^0, x^0)\end{bmatrix},$$

and $\quad^{\text{in}}$ is defined in (B.2). For the first first time step we have:

$$\begin{aligned}
^{(\ell;1;1)}(x, x^0) &= (\sigma_w)^2\sigma_h^2\mathbb{1}_{(x=x^0)} + (\sigma_u)^2\mathbb{V}\ \left[K^{(\ell-1;2;2)}(x, x^0)\right] + (\sigma_b)^2,\\
^{(\ell;t;1)}(x, x^0) &= (\sigma_u)^2\mathbb{V}\ \left[K^{(\ell-1;t+1;2)}(x, x^0)\right] + (\sigma_b)^2,\\
^{(\ell;1;t^0)}(x, x^0) &= (\sigma_u)^2\mathbb{V}\ \left[K^{(\ell-1;2;t^0+1)}(x, x^0)\right] + (\sigma_b)^2.
\end{aligned}$$

And the output layer

$$K(x, x^0) = \sigma_v^2\mathbb{V}\ \left[K^{(L;T+1;T^0+1)}(x, x^0)\right].$$

Note that because of using new weights at each layer we get

$$(g^{(\ell;t)}(x), g^{(\ell^0;t^0)})(x)) = 0 \qquad \ell \neq \ell^0$$

Now we calculate the backward pass kernels in multi-layer RNTK. The gradients at the last layer is calculated via

$$^{(L;T)}(x) = \sigma_v v \quad \phi^0(g^{(L;T)}(x)).$$

In the last hidden layer for different time steps we have

$$^{(L;t)}(x) = \frac{\sigma_w^L}{\sqrt{n}}\left(\mathbf{W}^{(L)}\right)^\top\left(\phi^0(g^{(L;t)}(x)) \qquad ^{(L;t+1)}(x)\right) \qquad t\ 2\ [T \quad 1]$$

In the last time step for different hidden layers we have

$$^{(`;T)}(x) = \frac{\sigma_u^{`+1}}{n}\left(\mathbf{U}^{(`+1)}\right)^{>}\left(\phi'(g^{(`;T)}(x)) \quad ^{(`+1;T)}(x)\right) \qquad `\ 2\ [L \quad 1]$$

At the end for the other layers we have

$$\begin{aligned}^{(`;t)}(x) = {}& \frac{\sigma_w^{`}}{n}\left(\mathbf{W}^{(`)}\right)^{>}\left(\phi'(g^{(`;t)}(x)) \quad ^{(`;t+1)}(x)\right) \\ & + \frac{\sigma_u^{`+1}}{n}\left(\mathbf{U}^{(`+1)}\right)^{>}\left(\phi'(g^{(`;t)}(x)) \quad ^{(`+1;t)}(x)\right) \qquad `\ 2\ [L \quad 1], t\ 2\ [T \quad 1]\end{aligned}$$

The recursive formula for the $^{(L;t;t^0)}(x, x^0)$ is the same as the single layer, and it is non-zero for $t^0 \quad t = T^0 \quad T = \tau$. As a result we have

$$\begin{aligned}^{(L;T;T+\ )}(x, x^0) &= \sigma_v^2 \mathsf{V}_{\ 0}\left[K^{(L;T+1;T+\ +1)}\right](x, x^0) \\ ^{(L;t;t+\ )}(x, x^0) &= (\sigma_w^L)^2 \mathsf{V}_{\ 0}\left[K^{(L;t+1;t+\ +1)}\right](x, x^0) \cdot \ ^{(L;t+1;t+1+\ )}(x, x^0) \quad t\ 2\ [T \quad 1] \\ ^{(L;t;t^0)}(x, x^0) &= 0 \qquad\qquad t^0 \quad t \neq \tau\end{aligned} \tag{33}$$

Similarly by using the same course of arguments used in the single layer setting, for the last time step we have

$$^{(`;T;T+\ )}(x, x^0) = (\sigma_u^{`+1})^2 \mathsf{V}_{\ 0}\left[K^{(`;T+1;T+\ +1)}\right](x, x^0) \cdot \ ^{(`+1;T+1;T+\ +1)}(x, x^0) \qquad `\ 2\ [L \quad 1]$$

For the other layers we have

$$\begin{aligned}^{(`;t;t^0)}(x, x^0) = {}& (\sigma_w^{`})^2 \mathsf{V}_{\ 0}\left[K^{(`;t+1;t+\ +1)}\right](x, x^0) \cdot \ ^{(`;t+1;t+1+\ )}(x, x^0) \\ & + (\sigma_u^{`+1})^2 \mathsf{V}_{\ 0}\left[K^{(`;t+1;t+\ +1)}\right](x, x^0) \cdot \ ^{(`+1;t+1;t^0+1)}(x, x^0).\end{aligned}$$

For $t^0 \quad t \neq \tau$ the recursion continues until it reaches $^{(L;T;t'')}(x, x^0), t'' < T^0$ or $^{(L;t'';T^0)}(x, x^0), t'' < T$ and as a result based on (33) we get

$$^{(`;t;t^0)}(x, x^0) = 0 \qquad t^0 \quad t \neq \tau \tag{34}$$

For $t^0 \quad t = \tau$ it leads to $^{(L;T;T^0)}(x, x^0)$ and has a non-zero value.
Now we derive RNTK for multi-layer:

$$\begin{aligned}\langle f\ (x), f\ (x^0)\rangle = {}& \sum_{`=1}^{L}\left\langle \frac{\partial f\ (x)}{\partial \mathbf{W}^{(`)}}, \frac{\partial f\ (x^0)}{\partial \mathbf{W}^{(`)}}\right\rangle + \sum_{`=1}^{L}\left\langle \frac{\partial f\ (x)}{\partial \mathbf{U}^{(`)}}, \frac{\partial f\ (x^0)}{\partial \mathbf{U}^{(`)}}\right\rangle \\ & + \sum_{`=1}^{L}\left\langle \frac{\partial f\ (x)}{\partial \mathbf{b}^{(`)}}, \frac{\partial f\ (x^0)}{\partial \mathbf{b}^{(`)}}\right\rangle + \left\langle \frac{\partial f\ (x)}{\partial v}, \frac{\partial f\ (x^0)}{\partial v}\right\rangle,\end{aligned}$$

where

$$\left\langle \frac{\partial f\ (x)}{\partial \mathbf{W}^{(`)}}, \frac{\partial f\ (x^0)}{\partial \mathbf{W}^{(`)}}\right\rangle = \sum_{t^0=1}^{T^0}\sum_{t=1}^{T}\left(\frac{1}{n}\left\langle \ ^{(`;t)}(x), \ ^{(`;t^0)}(x^0)\right\rangle\right) \cdot \left(\frac{(\sigma_w^{`})^2}{n}\left\langle h^{(`;t\ \ 1)}(x), h^{(`;t^0\ \ 1)}(x^0)\right\rangle\right)$$

$$\left\langle \frac{\partial f\ (x)}{\partial \mathbf{U}^{(`)}}, \frac{\partial f\ (x^0)}{\partial \mathbf{U}^{(`)}}\right\rangle = \sum_{t^0=1}^{T^0}\sum_{t=1}^{T}\left(\frac{1}{n}\left\langle \ ^{(`;t)}(x), \ ^{(`;t^0)}(x^0)\right\rangle\right) \cdot \left(\frac{(\sigma_u^{`})^2}{m} h x_t, x_{t^0}^0 i\right) \qquad ` = 1$$

$$\begin{aligned}\left\langle \frac{\partial f\ (x)}{\partial \mathbf{U}^{(`)}}, \frac{\partial f\ (x^0)}{\partial \mathbf{U}^{(`)}}\right\rangle = {}& \sum_{t^0=1}^{T^0}\sum_{t=1}^{T}\Bigg[\left(\frac{1}{n}\left\langle \ ^{(`;t)}(x), \ ^{(`;t^0)}(x^0)\right\rangle\right) \\ & \cdot \left(\frac{(\sigma_u^{`})^2}{n}\left\langle h^{(`\ \ 1;t)}(x), h^{(`\ \ 1;t^0)}(x^0)\right\rangle\right)\Bigg] \qquad ` > 1\end{aligned}$$

$$\left\langle \frac{\partial f\ (x)}{\partial \mathbf{b}^{(`)}}, \frac{\partial f\ (x^0)}{\partial \mathbf{b}^{(`)}}\right\rangle = \sum_{t^0=1}^{T^0}\sum_{t=1}^{T}\left(\frac{1}{n}\left\langle \ ^{(`;t)}(x), \ ^{(`;t^0)}(x^0)\right\rangle\right) \cdot (\sigma_b^{`})^2$$

$$\left\langle \frac{\partial f\ (x)}{\partial v}, \frac{\partial f\ (x^0)}{\partial v}\right\rangle = \left(\frac{\sigma_v^2}{n}\left\langle h^{(T)}(x), h^{(T^0)}(x^0)\right\rangle\right)$$

Summing up all the terms and replacing the inner product of vectors with their expectations we get

$$\langle \nabla f(x), \nabla f(x^0)\rangle = \Theta^{(L;T;T^0)} = \left(\sum_{\ell=1}^{L}\sum_{t=1}^{T}\sum_{t^0=1}^{T^0} \Theta^{(\ell;t;t^0)}(x,x^0)\cdot \dot{\Sigma}^{(\ell;t;t^0)}(x,x^0)\right) + K(x,x^0).$$

By (34), we can simplify to

$$\Theta^{(L;T;T^0)} = \left(\sum_{\ell=1}^{L}\sum_{t=1}^{T} \Theta^{(\ell;t;t^0)}(x,x^0)\cdot \dot{\Sigma}^{(\ell;t;t+\tau)}(x,x^0)\right) + K(x,x^0).$$

For multi-dimensional output it becomes

$$\Theta(x,x^0) = \Theta^{(L;T;T^0)}(x,x^0) \otimes I_d.$$

This concludes the proof for Theorem 1 for the multi-layer case.

B.4 Proof for Theorem 3: Weight-Untied RNTK

The architecture of a weight-untied single layer RNN is

$$\begin{aligned} g^{(t)}(x) &= \frac{\sigma_w}{m}\mathbf{W}^{(t)}h^{(t-1)}(x) + \frac{\sigma_u}{n}\mathbf{U}^{(t)}x_t + \sigma_b\mathbf{b}^{(t)} \\ h^{(t)}(x) &= \phi\left(g^{(t)}(x)\right) \\ f(x) &= \frac{\sigma_v}{n}v^\top h^{(T)}(x) \end{aligned}$$

Where we use new weights at each time step and we index it by time. Like previous sections, we first derive the forward pass kernels for two same length data $x = \{x_t\}_{t=1}^T, x = \{x_{t^0}^0\}_{t^0=1}^T$

$$\begin{aligned} \Sigma^{(t;t)}(x,x^0) &= \sigma_w^2\mathbb{V}_{\sim}\left[K^{(t;t)}(x,x^0)\right] + \frac{\sigma_u^2}{m}\langle x_t, x_t^0\rangle + \sigma_b^2. \\ \Sigma^{(t;t^0)}(x,x^0) &= 0 \qquad t \neq t^0 \end{aligned}$$

Since we are using same weight at the same time step, $\Sigma^{(t;t)}(x,x^0)$ can be written as a function of the previous kernel, which is exactly as the weight-tied RNN. However for different length, it becomes zero as a consequence of using different weights, unlike weight-tied which has non-zero value. The kernel of the first time step and output is also the same as weight-tied RNN. For the gradients we have:

$$\begin{aligned} \dot{\Sigma}^{(T)}(x) &= \sigma_v^2 \qquad \phi'(g^{(T)}(x)) \\ \dot{\Sigma}^{(t)}(x) &= \frac{\sigma_w^2}{n}(\mathbf{W}^{(t+1)})^\top\left(\phi'(g^{(t)}(x)) \qquad \dot{\Sigma}^{(t+1)}(x)\right) \qquad t \in [T-1] \end{aligned}$$

For $t^0 = t$ we have:

$$\begin{aligned} \Sigma^{(t;t)}(x,x^0) &= \sigma_w^2\mathbb{V}_{\sim}\left[K^{(t+1;t+1)}(x,x^0)\right] \dot{\Sigma}^{(t+1;t+1)}(x,x^0) \\ \dot{\Sigma}^{(t;t)}(x,x^0) &= \sigma_v^2\mathbb{V}_{\sim}\left[K^{(T+1;T+1)}(x,x^0)\right]. \end{aligned}$$

Due to using different weights for $t \neq t^0$, we can immediately conclude that $\Sigma^{(t;t^0)}(x,x^0) = 0$. This set of calculation is exactly the same as the weight-tied case when $\tau = T - T = 0$.
Finally, with $\theta = \mathrm{Vect}\left[\{\mathbf{W}^{(t)}, \mathbf{U}^{(t)}, \mathbf{b}^{(t)}\}_{t=1}^T, v\right]$ we have

$$\begin{aligned} \langle \nabla f(x), \nabla f(x^0)\rangle = &\sum_{t=1}^{T}\left\langle \frac{\partial f(x)}{\partial \mathbf{W}^{(t)}}, \frac{\partial f(x^0)}{\partial \mathbf{W}^{(t)}}\right\rangle + \sum_{t=1}^{T}\left\langle \frac{\partial f(x)}{\partial \mathbf{U}^{(t)}}, \frac{\partial f(x^0)}{\partial \mathbf{U}^{(t)}}\right\rangle \\ &+ \sum_{t=1}^{T}\left\langle \frac{\partial f(x)}{\partial \mathbf{b}^{(t)}}, \frac{\partial f(x^0)}{\partial \mathbf{b}^{(t)}}\right\rangle + \left\langle \frac{\partial f(x)}{\partial v}, \frac{\partial f(x^0)}{\partial v},\right\rangle \end{aligned}$$

with

$$\left\langle \frac{\partial f(x)}{\partial \mathbf{W}^{(t)}}, \frac{\partial f(x^0)}{\partial \mathbf{W}^{(t)}} \right\rangle = \left(\frac{1}{n} \left\langle \quad^{(t)}(x), \quad^{(t)}(x^0) \right\rangle \right) \cdot \left(\frac{\sigma_w^2}{n} \left\langle h^{(t\ \ 1)}(x), h^{(t\ \ 1)}(x^0) \right\rangle \right)$$
$$\left\langle \frac{\partial f(x)}{\partial \mathbf{U}^{(t)}}, \frac{\partial f(x^0)}{\partial \mathbf{U}^{(t)}} \right\rangle = \left(\frac{1}{n} \left\langle \quad^{(t)}(x), \quad^{(t)}(x^0) \right\rangle \right) \cdot \left(\frac{\sigma_u^2}{m} h x_t, x_t^0 i \right)$$
$$\left\langle \frac{\partial f(x)}{\partial \mathbf{b}^{(t)}}, \frac{\partial f(x^0)}{\partial \mathbf{b}^{(t)}} \right\rangle = \left(\frac{1}{n} \left\langle \quad^{(t)}(x), \quad^{(t)}(x^0) \right\rangle \right) \cdot \sigma_b^2$$
$$\left\langle \frac{\partial f(x)}{\partial v}, \frac{\partial f(x^0)}{\partial v} \right\rangle = \left(\frac{\sigma_v^2}{n} \left\langle h^{(T)}(x), h^{(T^0)}(x^0) \right\rangle \right).$$

As a result we obtain

$$h r \quad f(x), r \quad f(x^0) i = \left(\sum_{t=1}^{T} \quad^{(t;t)}(x, x^0) \cdot \quad^{(t;t)}(x^0, x^0) \right) + K(x, x^0),$$

same as the weight-tied RNN when $\tau = 0$. This concludes the proof for Theorem 3.

B.5 Analytical Formula for $\mathsf{V}\ [K]$

For any positive definite matrix $K = \begin{bmatrix} K_1 & K_3 \\ K_3 & K_2 \end{bmatrix}$ we have:

- $\phi = \mathsf{ReLU}$ (Cho & Saul, 2009)

$$\mathsf{V}\ [K] = \frac{1}{2\pi} \left(c(\pi \quad \arccos(c)) + \sqrt{1 \quad c^2} \right) \sqrt{K_1 K_2},$$
$$\mathsf{V}\ _0[K] = \frac{1}{2\pi}(\pi \quad \arccos(c)).$$

  where $c = K_3 / \sqrt{K_1 K_2}$

- $\phi = \mathsf{erf}$ (Neal, 1995)

$$\mathsf{V}\ [K] = \frac{2}{\pi} \arcsin \left(\frac{2K_3}{\sqrt{(1 + 2K_1)(1 + 2K_3)}} \right),$$
$$\mathsf{V}\ _0[K] = \frac{4}{\pi \sqrt{(1 + 2K_1)(1 + 2K_2) \quad 4K_3^2}}.$$

C Proof for Theorem 2: RNTK Convergence after Training

To prove theorem 2, we use the strategy used in (Lee et al., 2019) which relies on the the local lipschitzness of the network Jacobian $J(\theta, X) = r \quad f(x) \ 2 \ \mathsf{R}^{jX jd \ \ j \ j}$ at initialization.

Definition 1 *The Jacobian of a neural network is local lipschitz at NTK initialization* ($\theta_0 \quad N(0,1)$) *if there is constant $K > 0$ for every C such that*

$$\begin{cases} kJ(\theta, X)k_F < K \\ kJ(\theta, X) \quad J(\tilde{\theta}, X)k_F < K k\theta \quad \tilde{\theta}k \end{cases}, \qquad 8\ \theta, \tilde{\theta}\ 2\ B(\theta_0, R)$$

where

$$B(\theta, R) := f\theta : k\theta_0 \quad \theta k < R g.$$

Theorem 4 *Assume that the network Jacobian is local lipschitz with high probability and the empirical NTK of the network converges in probability at initialization and it is positive definite over the input set. For $\epsilon > 0$, there exists N such that for $n > N$ when applying gradient flow with $\eta < 2(\lambda_{\min}(\quad(X, X)) + \lambda_{\max}(\quad(X, X)))^{-1}$ with probability at least $(1 \quad \epsilon)$ we have:*

$$\sup_s \frac{k\theta_s \quad \theta_0 k_2}{p \overline{n}}, \sup_s k\hat{\ }_s(X, X) \quad \hat{\ }_0(X, X)k = O\left(\frac{1}{p \overline{n}} \right).$$

Proof: See (Lee et al., 2019)

Theorem 4 holds for any network architecture and any cost function and it was used in (Lee et al., 2019) to show the stability of NTK for MLP during training.

Here we extend the results for RNTK by proving that the Jacobian of a multi-layer RNN under NTK initialization is local lipschitz with high probability.

To prove it, first, we prove that for any two points $\theta, \tilde{\theta} \in B(\theta_0, R)$ there exists constant K_1 such that

$$\|g^{(\ell,t)}(x)\|_2, \|\delta^{(\ell,t)}(x)\|_2 \leq K_1 \sqrt{n} \tag{35}$$

$$\|g^{(\ell,t)}(x) - \tilde{g}^{(\ell,t)}(x)\|_2, \|h^{(\ell,t)}(x) - \tilde{h}^{(\ell,t)}(x)\|_2 \leq \|\theta - \tilde{\theta}\| \leq K_1 \sqrt{n} \|\theta - \tilde{\theta}\|. \tag{36}$$

To prove (35) and (36) we use the following lemmas.[3]

Lemma 1 *Let $A \in \mathbb{R}^{n \times m}$ be a random matrix whose entries are independent standard normal random variables. Then for every $t \geq 0$, with probability at least $1 - e^{(-ct^2)}$ for some constant c we have:*

$$\|A\|_2 \leq \sqrt{m} + \sqrt{n} + t.$$

Lemma 2 *Let $a \in \mathbb{R}^n$ be a random vector whose entries are independent standard normal random variables. Then for every $t \geq 0$, with probability at least $1 - e^{(-ct^2)}$ for some constant c we have:*

$$\|a\|_2 \leq \sqrt{n} + t.$$

Setting $t = \sqrt{n}$ for any $\theta \in R(\theta_0, R)$. With high probability, we get:

$$\|\mathbf{W}^{(\ell)}\|_2, \|\mathbf{U}^{(\ell)}\|_2 \leq 3\sqrt{n}, \quad \|\mathbf{b}^{(\ell)}\|_2 \leq 2\sqrt{n}, \quad \|\mathbf{h}^{(\ell,0)}(x)\|_2 \leq 2\sigma_h \sqrt{n}.$$

We also assume that there exists some finite constant C such that

$$|\phi(x)| < C|x|, \quad |\phi(x) - \phi(x^0)| < C|x - x^0|, \quad |\phi'(x)| < C, \quad |\phi'(x) - \phi'(x^0)| < C|x - x^0|.$$

The proof is obtained by induction. From now on assume that all inequalities in (35) and (36) holds with some k for the previous layers. We have

$$\begin{aligned}
\|g^{(\ell,t)}(x)\|_2 &= \left\| \frac{\sigma_w}{\sqrt{n}} \mathbf{W}^{(\ell)} h^{(\ell,t-1)}(x) + \frac{\sigma_u}{\sqrt{n}} \mathbf{U}^{(\ell)} h^{(\ell-1,t)}(x) + \sigma_b \mathbf{b}^{(\ell)} \right\|_2 \\
&\leq \frac{\sigma_w}{\sqrt{n}} \|\mathbf{W}^{(\ell)}\|_2 \|\phi\left(g^{(\ell,t-1)}(x)\right)\|_2 + \frac{\sigma_u}{\sqrt{n}} \|\mathbf{U}^{(\ell)}\|_2 \|\phi\left(g^{(\ell-1,t)}(x)\right)\|_2 + \sigma_b \|\mathbf{b}^{(\ell)}\|_2 \\
&\leq \left(3\sigma_w C k + 3\sigma_u C k + 2\sigma_b\right) \sqrt{n}.
\end{aligned}$$

And the proof for (35) and (36) is completed by showing that the first layer is bounded

$$\begin{aligned}
\|g^{(1,1)}(x)\|_2 &= \left\| \frac{\sigma_w^1}{\sqrt{n}} \mathbf{W}^{(\ell)} h^{(1,0)}(x) + \frac{\sigma_u^1}{\sqrt{m}} \mathbf{U}^{(\ell)} x_1 + \sigma_b^1 \mathbf{b}^{(1)} \right\|_2 \\
&\leq \left(3\sigma_w^1 \sigma_h + \frac{3\sigma_u}{\sqrt{m}} \|x_1\|_2 + 2\sigma_b\right) \sqrt{n}.
\end{aligned}$$

For the gradient of first layer we have

$$\begin{aligned}
\|\delta^{(L,T)}(x)\|_2 &= \|\sigma_v v \phi'(g^{(L,T)}(x))\|_2 \\
&\leq \sigma_v \|v\|_2 \|\phi'(g^{(L,T)}(x))\|_1 \\
&= 2\sigma_v C \sqrt{n}.
\end{aligned}$$

And similarly we have

$$\|\delta^{(\ell,t)}(x)\| \leq \left(3\sigma_w C k^0 + 3\sigma_u C k^0\right) \sqrt{n}.$$

[3] See `math.uci.edu/~rvershyn/papers/HDP-book/HDP-book.pdf` for proofs

For $\theta, \tilde{\theta} \in B(\theta_0, R)$ we have

$$\|g^{(1;1)}(x) - \tilde{g}^{(1;1)}(x)\|_2 = \|\frac{\sigma_w^1}{\sqrt{n}}(\mathbf{W}^{(1)} - \tilde{\mathbf{W}}^{(1)})h^{(1;0)}(x) + \frac{\sigma_u^1}{\sqrt{m}}(\mathbf{U}^{(1)} - \tilde{\mathbf{U}}^{(1)})h^{(1;0)}(x)\|_2$$
$$\left(3\sigma_w^1\sigma_h + \frac{3\sigma_u^1}{m}\|x_1\|_2\right)\|\theta - \tilde{\theta}\|_2 \sqrt{n}.$$

$$\begin{aligned}\|g^{(\ell;t)}(x) - \tilde{g}^{(\ell;t)}(x)\|_2 &\le \|\phi(g^{(\ell;t-1)}(x))\|_2\|\frac{\sigma_w^\ell}{\sqrt{n}}(\mathbf{W}^{(\ell)} - \tilde{\mathbf{W}}^{(\ell)})\|_2 \\ &+ \|\frac{\sigma_w^\ell}{\sqrt{n}}\tilde{\mathbf{W}}^{(\ell)}\|_2\|\phi(g^{(\ell;t-1)}(x)) - \phi(\tilde{g}^{(\ell;t-1)}(x))\|_2 \\ &+ \|\phi(g^{(\ell-1;t)}(x))\|_2\|\frac{\sigma_u^\ell}{\sqrt{m}}(\mathbf{U}^{(\ell)} - \tilde{\mathbf{U}}^{(\ell)})\|_2 \\ &+ \|\frac{\sigma_u^\ell}{\sqrt{m}}\tilde{\mathbf{U}}^{(\ell)}\|_2\|\phi(g^{(\ell-1;t)}(x)) - \phi(\tilde{g}^{(\ell-1;t)}(x))\|_2 + \sigma_b\|\mathbf{b}^{(\ell)} - \tilde{\mathbf{b}}^{(\ell)}\| \\ &(\|\sigma_w^\ell + 3\sigma_w^\ell Ck + \|\sigma_u^\ell + 3\sigma_u^\ell Ck + \sigma_b)\|\theta - \tilde{\theta}\|_2 \sqrt{n}.\end{aligned}$$

For gradients we have

$$\begin{aligned}\|\alpha^{(L;T)}(x) - \tilde{\alpha}^{(L;T)}(x)\|_2 &\le \sigma_v\|\phi'(g^{(L;T)})\|_1\|(v - \tilde{v})\|_2 + \sigma_v\|v\|_2\|\phi'(g^{(L;T)}(x)) - \phi'(\tilde{g}^{(L;T)}(x))\|_2 \\ &(\sigma_v C + 2\sigma_v Ck)\|\theta - \tilde{\theta}\|_2 \sqrt{n}.\end{aligned}$$

And similarly using same techniques we have

$$\|\alpha^{(\ell;t)}(x) - \tilde{\alpha}^{(\ell;t)}(x)\|_2 \le (\sigma_w C + 3\sigma_w Ck + \sigma_u C + 3\sigma_u Ck)\|\theta - \tilde{\theta}\|_2 \sqrt{n}.$$

As a result, there exists K_1 that is a function of $\sigma_w, \sigma_u, \sigma_b, L, T$ and the norm of the inputs.
Now we prove the local Lipchitzness of the Jacobian

$$\begin{aligned}\|J(\theta, x)\|_F &\le \sum_{\ell=2}^{L}\sum_{t=1}^{T}\left(\frac{1}{n}\left\|\alpha^{(\ell;t)}(x)\left(\sigma_w^\ell h^{(\ell;t-1)}(x)\right)^\top\right\|_F\right. \\ &+ \left.\frac{1}{n}\left\|\alpha^{(\ell;t)}(x)\left(\sigma_u^\ell h^{(\ell;t-1)}(x)\right)^\top\right\|_F + \frac{1}{\sqrt{n}}\left\|\alpha^{(\ell;t)}(x)\cdot\sigma_b^\ell\right\|_F\right) \\ &+ \sum_{t=1}^{T}\left(\frac{1}{n}\left\|\alpha^{(1;t)}(x)\left(\sigma_w^1 h^{(1;t-1)}(x)\right)^\top\right\|_F\right. \\ &+ \left.\frac{1}{\sqrt{nm}}\left\|\alpha^{(1;t-1)}(x)\left(\sigma_u^1 x_t\right)^\top\right\|_F + \frac{1}{\sqrt{n}}\left\|\alpha^{(1;t)}(x)\cdot\sigma_b^1\right\|_F\right) + \frac{\sigma_v}{\sqrt{n}}\|h^{(L;T)}(x)\|_F \\ &\left(\sum_{\ell=2}^{L}\sum_{t=1}^{T}(K_1^2 C\sigma_w^\ell + K_1^2 C\sigma_u^\ell + \sigma_b^\ell K_1)\right. \\ &+ \left.\sum_{t=1}^{T}(K_1^2 C\sigma_w^1 + \frac{K_1\sigma_u^1}{\sqrt{m}}\|x_t\|_2 + \sigma_b^1 K_1) + \sigma_v C K_1\right).\end{aligned}$$

And for $\theta, \tilde{\theta} \in B(\theta_0, R)$ we have

$$
\begin{aligned}
\|J(\theta, x) - \tilde{J}(\theta, x)\|_F \quad & \sum_{\ell=2}^{L}\sum_{t=1}^{T}\left(\frac{1}{n}\left\| \dot{\Sigma}^{(\ell,t)}(x)\left(\sigma_w^{\ell} h^{(\ell,t-1)}(x)\right)^{\top} - \tilde{\dot{\Sigma}}^{(\ell,t)}(x)\left(\sigma_w^{\ell} \tilde{h}^{(\ell,t-1)}(x)\right)^{\top}\right\|_F\right. \\
& + \frac{1}{n}\left\| \dot{\Sigma}^{(\ell,t)}(x)\left(\sigma_u^{\ell} h^{(\ell,t-1)}(x)\right)^{\top} - \tilde{\dot{\Sigma}}^{(\ell,t)}(x)\left(\sigma_u^{\ell} \tilde{h}^{(\ell,t-1)}(x)\right)^{\top}\right\|_F \\
& + \frac{1}{\sqrt{n}}\left\| \dot{\Sigma}^{(\ell,t)}(x)\cdot\sigma_b^{\ell} - \tilde{\dot{\Sigma}}^{(\ell,t)}(x)\cdot\sigma_b^{\ell}\right\|_F \\
& + \sum_{t=1}^{T}\left(\frac{1}{n}\left\| \dot{\Sigma}^{(1,t)}(x)\left(\sigma_w^{1} h^{(1,t-1)}(x)\right)^{\top} - \tilde{\dot{\Sigma}}^{(1,t)}(x)\left(\sigma_w^{1} \tilde{h}^{(1,t-1)}(x)\right)^{\top}\right\|_F\right. \\
& + \frac{1}{\sqrt{nm}}\left\| \dot{\Sigma}^{(1,t-1)}(x)\left(\sigma_u^{1} x_t\right)^{\top} - \tilde{\dot{\Sigma}}^{(1,t-1)}(x)\left(\sigma_u^{1} x_t\right)^{\top}\right\|_F \\
& \left. + \frac{1}{\sqrt{n}}\left\| \dot{\Sigma}^{(\ell,t)}(x)\cdot\sigma_b^{\ell} - \tilde{\dot{\Sigma}}^{(\ell,t)}(x)\cdot\sigma_b^{\ell}\right\|\right) + \frac{\sigma_v}{\sqrt{n}}\|h^{(L,T)}(x) - \tilde{h}^{(L,T)}(x)\|_F \\
& \left(\sum_{\ell=2}^{L}\sum_{t=1}^{T}(4K_1^2 C\sigma_w^{\ell} + 4K_1^2 C\sigma_u^{\ell} + \sigma_b^{\ell} K_1)\right. \\
& \left. + \sum_{t=1}^{T}(4K_1^2 C\sigma_w^{1} + \frac{K_1\sigma_u^{1}}{\sqrt{m}}\|x_t\|_2 + \sigma_b^{1} K_1) + \sigma_v C K_1\right)\|\theta - \tilde{\theta}\|_2.
\end{aligned}
$$

The above proof can be generalized to the entire dataset by a straightforward application of the union bound. This concludes the proof for Theorem 2.