# Problem Set 4

Name:

Due Midnight on February 2, 2023

The goal of this problem set is to understand how to compute and use the NTK for a wide class of network architectures. We use (T) to denote theory exercises and (C) to denote coding exercises. We lastly indicate more involved problems with a *.

**Students should solve at least 1 problem.**

## NTK Derivation and Origin

**Problem 1 (T).** Let $\phi(z) = \sqrt{2}\max(0, z)$ (i.e. the normalized ReLU function), and let $\Lambda = \begin{bmatrix} \alpha & \xi \\ \xi & \beta \end{bmatrix}$. Find a closed form expression for:

$$\mathbb{E}_{u,v\sim\mathcal{N}(\mathbf{0},\Lambda)}[\phi(u)\phi(v)] \quad \text{and} \quad \mathbb{E}_{u,v\sim\mathcal{N}(\mathbf{0},\Lambda)}[\phi'(u)\phi'(v)]$$

**Hint:** Recall that these expectations are similar to those used in the definition of the dual activation (for which $\alpha = \beta = 1$). When computing the above integrals, use the change of variables $u = \sqrt{\alpha}u'$ and $v = \sqrt{\beta}v'$, the homogeneity of ReLU (i.e. the fact that $\phi(cx) = c\phi(x)$), and the dual of a ReLU network given by:

$$\check{\phi}(\xi) = \frac{1}{\pi}(\xi(\pi - \arccos(\xi)) + \sqrt{1-\xi^2}) \ .$$

**Remarks:** Remember that for examples $x, \tilde{x} \in \mathbb{R}^d$, the NNGP and NTK involve computing the expectations above where $\alpha = \|x\|_2^2$, $\beta = \|\tilde{x}\|_2^2$, $\xi = x^T\tilde{x}$. Thus, the above technique gives a method for extending the NNGP and NTK computations from those for $x, \tilde{x} \in \mathcal{S}^{d-1}$ to general $x, \tilde{x} \in \mathbb{R}^d$ for homogeneous functions.

**Problem 2 (T).** For $x \in \mathcal{S}^{d-1}$, let $f_x(w) = A\frac{1}{\sqrt{k}}\phi(Bx)$ where $A \in \mathbb{R}^{c\times k}$ with $c > 1$, $B \in \mathbb{R}^{k\times d}$ and $\phi$ is an element-wise activation function. For $x, \tilde{x} \in \mathcal{S}^{d-1}$, prove that:

$$\langle \nabla f_x(w)_i, \nabla f_x(w)_j \rangle = \begin{cases} K(x,\tilde{x}) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad ;$$

where $K : \mathcal{S}^{d-1} \times \mathcal{S}^{d-1} \to \mathbb{R}$ is the NTK.
**Remarks:** Recall that we established the NTK when $c = 1$ in the lectures. This problem introduces a key aspect of the NTK of fully connected networks that we utilized in deriving the NTK for a deep fully connected network (see Lecture 6).

**Problem 3 (T).** For $x \in \mathbb{R}$, let $f_x(w) = A\frac{1}{\sqrt{k}}\phi(Bx)$ where $A \in \mathbb{R}^{1\times k}$, $B \in \mathbb{R}^k$ and $\phi$ is an element-wise activation function. Let $A_{1i}, B_i$ be bounded for $i \in [k]$ and consider the setting in which the parameters are a frozen (e.g. fixed constants) and $B$ are the only trainable parameters. Prove that for $x, \tilde{x} \in [-m, m]$:

$$K(x,\tilde{x}) = O(1) \quad ; \quad \|H(f_x(w))\|_2 = O\left(\frac{1}{\sqrt{k}}\right)$$

**Remarks.** In Lecture 5, we performed a similar analysis for the case when $A$ and $B$ were both trainable parameters. In this case, the Hessian will be a matrix of size $\mathbb{R}^{k\times k}$ instead of $\mathbb{R}^{2k\times 2k}$, as the only trainable parameters are given in $B$.

**Problem 4 (C*).** In this exercise, we will verify that solving kernel regression with the NTK leads to the same predictions on test data as wide neural networks trained using gradient descent. Consider a 1 hidden layer neural network $f : \mathbb{R}^d \to \mathbb{R}$ such that:

$$f(x) = A\frac{\sqrt{2}}{\sqrt{k}}\phi(Bx) \ \ ;$$

where $\phi(z) = \max(0, z)$ is the ReLU activation, $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^{k\times d}$. For $d = 100, n = 10$, let $X \in \mathbb{R}^{d\times n}$ with entries sampled i.i.d. from $\mathcal{N}(0, 1)$. Let $g(x) = \frac{1}{d}\left(\sum_{i=1}^{d} x_i + \sin(10x_i)\right)$, and let $y = g(X) \in \mathbb{R}^{1\times n}$, where $y_i = g(X_{:,i})$. For $n_t = 1000$, let $X_t \in \mathbb{R}^{d\times n_t}$ and let $y_t = g(X_t) \in \mathbb{R}^{1\times n_t}$.

(a) Let $k = 2048$ and let $A_{1i}^{(0)}, B_{ij}^{(0)} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$. Construct the adjusted training set $(X, y - f(w^{(0)}\ ;\ X))$ where $f(w^{(0)}\ ; X) \in \mathbb{R}^{1\times n}$ is the vector of neural network predictions on the training data.

(b) Solve kernel regression on adjusted training set $(X, y - f(w^{(0)}\ ;\ X))$ with the empirical NTK given by:

$$K_E(x, \tilde{x}) = \langle \nabla f_x(w^{(0)}), \nabla f_{\tilde{x}}(w^{(0)})\rangle \ \ ;$$

for $x, \tilde{x} \in \mathbb{R}^d$.

(c) For $x \in X_t$, compute the predictions:

$$\hat{y}_t = (y - f(w^{(0)}\ ;\ X))\hat{K}_E^{-1}K_E(X, x) + f(w^{(0)}\ ;\ x) \ \ ;$$

where $\hat{K}_E \in \mathbb{R}^{n\times n}$ is the training kernel matrix and $K_E(X, x) \in \mathbb{R}^n$ such that $K_E(X, x)_i = K_E(X_{:,i}, x)$.

(d) Use gradient descent[1] to train the network $f(w^{(0)}\ ;\ x)$ to fit the training data $(X, y)$ until the MSE is less than $10^{-3}$, and compute the predictions $\tilde{y}_t = f(w\ ;\ X_t)$.
**Remarks:** You may need to experiment with the learning rate selection for gradient descent. We recommend starting with a learning rate of 0.01 and then decreasing the learning rate if needed.

(e) Report the MSE $r_k = \frac{1}{n_t}\|\hat{y}_t - \tilde{y}_t\|_2^2$. Repeat steps (a) through (d) for $k \in [128, 256, 512, 1024, 4096]$ and plot the value of $r_k$ vs. $k$.

**Problem 5 (T, C*).** For $d = 1, n = 10$, let $X \in \mathbb{R}^{d\times n}$ with entries sampled i.i.d. from $\mathcal{N}(0, 1)$. Let $g(x) = x + \sin(10x)$, and let $y = g(X) \in \mathbb{R}^{1\times n}$, where $y_i = g(X_{:,i})$. For $n_t = 1000$, let $X_t \in \mathbb{R}^{d\times n_t}$ and let $y_t = g(X_t) \in \mathbb{R}^{1\times n_t}$.

(a) Solve kernel regression to fit the training data $(X, y)$ using the NTK for the ReLU network $f(x) = A\frac{\sqrt{2}}{\sqrt{k}}\phi(Bx)$ where $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^k$, and $\phi(z) = \max(0, z)$ is the ReLU activation. Recall that the data in this case is not on the unit sphere and so the NTK is given by:

$$K(x, \tilde{x}) = \frac{1}{\pi}\left(x^T\tilde{x}\left(\pi - \arccos\left(\frac{x^T\tilde{x}}{\|x\|_2\|\tilde{x}\|_2}\right)\right) + \sqrt{\|x\|_2^2\|\tilde{x}\|_2^2 - (x^T\tilde{x})^2}\right) + x^T\tilde{x}\frac{1}{\pi}\left(\pi - \arccos\left(\frac{x^T\tilde{x}}{\|x\|_2\|\tilde{x}\|_2}\right)\right)$$

(b) Compute the MSE of your solution to (a) on the training data. Is the training error 0?

[1] You can either code up gradient descent yourself or use a deep learning library such as PyTorch [3].

(c) We will now compute the NTK for a 1 hidden layer neural network with biases. In particular, let $f(x) = A\frac{\sqrt{2}}{\sqrt{k}}\phi(Bx + C)$ where $\phi(z) = \max(0, z)$ is the ReLU activation, $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^k$, and the biases $C \in \mathbb{R}^k$. Assuming $A_{1i}, B_i, C_i \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$ for $i \in [k]$, compute the NTK given by:

$$K(x, \tilde{x}) = \lim_{k\to\infty} \langle \nabla f_x(w^{(0)}), \nabla f_{\tilde{x}}(w^{(0)}) \rangle$$

(d) Solve kernel regression to fit the training data $(X, y)$ using the NTK for the ReLU network with biases computed in part (c). Is the training MSE 0?

## NTK for Deep Networks and the CNTK

**Problems 6 and 7 below will make use of the Neural Tangents library described below.**

The Neural Tangents library [2] provides an API for extracting the NNGP and NTK given a neural network architecture. Several layer structures and activation functions are supported including fully connected layers, 1D convolutional layers, 2D convolutional layers, etc. For the purposes of the exercises below, we need only use the `kernel_fn` extracted from a given architecture to solve kernel regression directly or using EigenPro. Note that the Neural Tangents library offers a variety of functions (including functions for analyzing finite width networks) that may be useful for independent research.

**Problem 6 (C).** Load the CIFAR10 [1] image classification dataset. CIFAR10 contains 10 classes of color images of size $32 \times 32$. There are 50k training examples and 10k test examples, i.e. the training sample matrix, $X$, should have shape $50000 \times 32 \times 32 \times 3$ and the training labels, $y$, should have shape $50000 \times 10$ (where each row of $y$ is a one-hot vector indicating the class label). Consider the subset of CIFAR10, which consists of only the dog and cat images. There should now be 10000 training images and 2000 test images.

(a) For $k \in [1, 3, 5, 7]$, use the Neural Tangents library to solve kernel regression with the NTK of fully connected ReLU networks with $k$ hidden layers for the dog/cat subset of CIFAR10. Plot the accuracy of each model on the 2000 test examples as a function of $k$.
**Remarks:** Since the size of the training set is small, kernel regression can be solved exactly. Alternatively, one can use EigenPro.

(b) Repeat part (a) using another activation function support by the Neural Tangents library. Examples include $\sin(x)$, leaky ReLU, GeLU, etc.

**Problem 7 (C*).** Load the CIFAR10 [1] image classification dataset. CIFAR10 contains 10 classes of color images of size $32 \times 32$. There are 50k training examples and 10k test examples, i.e. the training sample matrix, $X$, should have shape $50000 \times 32 \times 32 \times 3$ and the training labels, $y$, should have shape $50000 \times 10$ (where each row of $y$ is a one-hot vector indicating the class label). Consider the subset of CIFAR10, which consists of only the dog and cat images. There should now be 10000 training images and 2000 test images.

(a) For $k \in [1, 3, 5, 7]$, use the Neural Tangents library to solve kernel regression exactly (i.e. using the `solve` numpy command) with the CNTK of fully connected convolutional networks with $k$ hidden layers and ReLU activation for the dog/cat subset of CIFAR10. Recall, that these networks are a series of convolutional layers followed by a flatten operation and then a fully connected layer. Plot the accuracy of each model on the 2000 test examples as a function of $k$.
**Remarks:** To speed up computation, you may consider networks where the first convolutional layer has a stride size of 2 in each direction.

(b) Repeat part (a) but instead, solving kernel regression with EigenPro. Plot the best test accuracy of each model on the 2000 test examples as a function of $k$. Is there any benefit to early stopping with EigenPro on this dataset?

**Problem 8 (T*).** In this problem, we will compare the kernels induced by the NTK of a fully connected convolutional network and the global average pooling convolutional network.

(a) Let $f(X) = A\left(\frac{1}{\sqrt{k}}\phi(B * X)\right)_v$ be a 1 hidden layer fully connected convolutional network where $A \in \mathbb{R}^{1\times kpq}$ is a fully connected layer, $B \in \mathbb{R}^{k\times 1\times 3\times 3}$ is a convolutional layer with $k$ filter of size $3\times 3$, $x \in \mathbb{R}^{p\times q}$, and $\phi$ is an elementwise activation function. For any tensor, $D$, we use $D_v$ to denote the vectorized version of $D$. In particular, we have that:

$$f(X) = \sum_{\ell=1}^{k}\sum_{i=1}^{p}\sum_{j=1}^{q} A_{(i,j,\ell)_v} Z_{i,j}^{(\ell)}$$

$$Z_{i,j}^{(\ell)} = \frac{1}{\sqrt{k}}\phi\left(\sum_{a,b\in\{-1,0,1\}} B_{a,b}^{(\ell)} X_{i+a,j+b}\right) \quad ;$$

where $(i,j,\ell)_v$ is the entry corresponding to indices $i,j,\ell$ in the flattened representation of $Z$. For any two images $X, \tilde{X}$, compute the NTK of $f$ given by the usual formula:

$$K(X,\tilde{x}) = \langle \nabla f_X(w), \nabla f_{\tilde{X}}(w)\rangle$$

(b) Let $g(X)$ be a 1 hidden layer global average pooling convolutional network defined as follows:

$$g(X) = \sum_{\ell=1}^{k} A_\ell \left(\frac{1}{pq}\sum_{i=1}^{p}\sum_{j=1}^{q} Z_{i,j}^{(\ell)}\right)$$

$$Z_{i,j}^{(\ell)} = \frac{1}{\sqrt{k}}\phi\left(\sum_{a,b\in\{-1,0,1\}} B_{a,b}^{(\ell)} X_{i+a,j+b}\right) \quad .$$

For any two images $X, \tilde{X}$, compute the NTK of $g$ given by the usual formula:

$$K(X,\tilde{x}) = \langle \nabla g_X(w), \nabla g_{\tilde{X}}(w)\rangle$$

(c) How does the NTK for (a) compare with that of (b)?

### RFMs

**Problem 9 (C).** Benchmark RFMs on the course challenge (e.g., try out the notebook posted today). Try experimenting with the number of iterations and centering in RFMs to see how these impact performance and the resulting learned features.

## References


[1] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[2] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz. Neural Tangents: Fast and easy infinite neural networks in Python. In *International Conference on Learning Representations*, 2020.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.