A Review: Toward Deeper Understanding of Neural Networks (Daniely et al. 2016)

Yidan (Eden) Xu, Zeyu (Jerry) Wei
Department of Statistics, University of Washington
yx2516@uw.edu, zwei5@uw.edu

Abstract

We review the paper *Toward Deeper Understanding of Neural Networks: The Power of Initialization and a Dual View on Expressivity* [1] by Daniely et al. and answer guided questions in the report. In particular, we make explicit the contributions of the paper in Section 1, and provide a summary of the paper in Section 2. In Section 3, we examine Theorem 6 in the paper with a empirical study on MNIST data.

1 Contribution

Neural network (NN) learning has been widely applied in numerous machine learning task and achieved state of art empirical result. However, it remains an important task to analyze theoretically the representation power of NN, which gives great attribution for its success. In [1], Daniely et al. analyzed the set of functions that can be expressed by varying the weights of the last layer of NN, which has randomly initialize weights. In particular, they showed with high probability, the space of representation functions learnt through NN approximates the RKHS induced from compositional kernels defined by activation functions associated with the network.

Their contributions are in three folds,

1. Introduced Computation skeleton (a DAG) for Neural networks;
2. Derived dual kernels using Hermite polynomial expansions for activation functions $\sigma \in L^{2*}_{\mu}$, e.g. ReLU and sigmoid;
3. Derived a bound for the closeness between the space of representation functions learnt through the NN and the RKHS defined for the corresponding compositional kernel of the skeleton.

These together provide a general framework for studying the function spaces induced by NNs that are from a broad family of architectures, extending previous works on networks with only convolutional layers and fully connected layers.

2 Summary of the paper

2.1 Problem overview

The authors give a supervised learning setting with samples $S = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_m, y_m)\}$ drawn i.i.d from distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. The goal is to find a solution $h : \mathcal{X} \to \mathbb{R}^k$ that minimizes the population risk in Equ (1). In particular, the authors later assumed that $h \in \mathcal{H}$, where $\mathcal{H}$ some RKHS,

* μ the standard Gaussian measure

Conference on Neural Information Processing Systems (NeurIPS), .

and $\|h\|_{\mathcal{H}} < R$ for some $R > 0$. And denote $h \in \mathcal{H}^R$ the functions that lie in the ball of radius R in $\mathcal{H}$.

$$\mathcal{L}_{\mathcal{D}}(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\ell(h(\boldsymbol{x}), y)) \tag{1}$$

where $\ell : \mathbb{R}^k \times \mathcal{Y} \to [0, \infty)$ is a convex and L-Lipschitz loss function. Notably, the assumption over the input space $\mathcal{X}$ is such that for each example $\boldsymbol{x}$,

$$\boldsymbol{x} = (\boldsymbol{x}^1, \cdots, \boldsymbol{x}^n), \text{ where } \boldsymbol{x}^i \in \mathbb{S}^{d-1},\ \mathcal{X} = \mathcal{X}_{n,d} = (\mathbb{S}^{d-1})^n$$

i.e. each coordinates $\boldsymbol{x}^i$ is a vector with length d lying on the unit hypersphere $\mathbb{S}^{d-1}$.

Then the authors considers the NNs, $\mathcal{N} = (V, E)$, which consist of convolutional layers and fully-connected layers with square integrable activation functions $\sigma_v : \mathbb{R} \to \mathbb{R}$ at each internal neurons $v \in V(\mathcal{N})$ (with both incoming and outgoing edges) w.r.t the Gaussian measure μ on $\mathbb{R}$. Therefore the norm of activation functions is of the form

$$\|\sigma_v\| = \sqrt{\mathbb{E}_{X\sim\mathcal{N}(0,1)\sigma^2(X)}} = \sqrt{\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\sigma^2(x)\exp(-x^2/2)\,dx},$$

and σ_v is normalized if $\|\sigma_v\| = 1$.

To match with the setup of supervised learning, the NN should have n input neurons and k output neurons, denoted $o_1, \cdots, o_k$. The NN with a weight vector $\boldsymbol{w} = \{w_{uv} : uv \in E(\mathcal{N})\}$ defines a predictor at each neuron v as the output of the subgraph at v, i.e.

$$h_{v,\boldsymbol{w}}(\boldsymbol{x}) = \sigma_v\left(\sum_{u\in\text{in}(v)} w_{uv}h_{u,\boldsymbol{w}}(\boldsymbol{x})\right),$$

where $\text{in}(v) = \{u \in V(\mathcal{N}) | uv \in E\}$. And finally the predictor given by the NN is $h_{\mathcal{N},\boldsymbol{w}}(\boldsymbol{x}) = (h_{o_1,\boldsymbol{w}}(\boldsymbol{x}), \cdots, h_{o_k,\boldsymbol{w}}(\boldsymbol{x}))$.

Further, the representation $\text{rep}(\mathcal{N})$ is induced from a network $\mathcal{N}$ by removing the output layer. The associated representation function is defined as $\Psi_{\boldsymbol{w}} = h_{\text{rep}(\mathcal{N}),\boldsymbol{w}}$.

2.2 Computation Skeleton and Neural Network

To connect neural networks to kernels, the authors use the concept of computation skeleton, which is defined as a Directed Acyclic Graph (DAG) whose non-input nodes are labeled by activations. Such

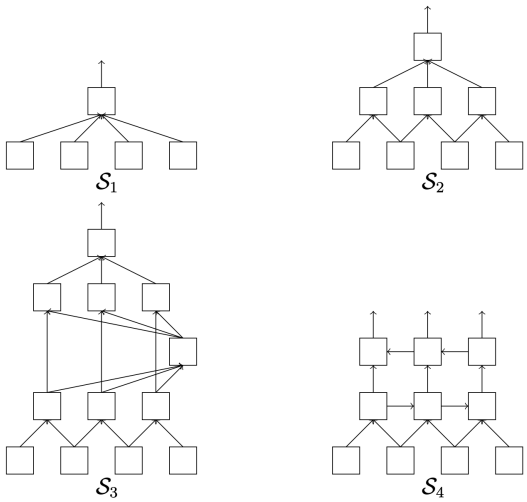

Figure 1: Examples of computation skeleton.

skeleton, on one hand, captures the architecture of a class of neural networks, and one the other hand,

stands for a kernel that can be used to describe such network structure. Examples of computation skeletons are shown in Figure 1.

Given a computation skeleton, neural network can be constructed by specifying the replication parameter r and the number of output nodes k. Let $\mathcal{S}$ be a computation skeleton with input coordinates in $\mathbb{S}^{d-1} \equiv \{x \in \mathbb{R}^d : ||x|| = 1\}$, then the neural network induced by the computation skeleton $\mathcal{N} = \mathcal{N}(\mathcal{S}, r, k)$ is defined as:

- For each input node in $\mathcal{S}, \mathbb{N}$ has d corresponding input neurons with weight $1/d$.
- For each internal node $v \in \mathcal{S}$ with activation function σ, such node is replicated into r neurons in $\mathcal{N}$, $v^1, \ldots, v^r$, each with an activation σ and weight $1/r$.
- In addition, $\mathcal{N}$ has k output neurons $o_1, \ldots, o_k$ with identity activation and weight 1.
- There is an edge $v^i u^j \in E(\mathcal{N}), v^i, u^j \in V(\mathcal{N})$ whenever $uv \in E(\mathcal{S})$ for the corresponding nodes u, v in $V(\mathcal{S})$
- For every output node $v \in V(\mathcal{S})$, each realized neuron $v^j \in V(\mathcal{N})$ is connected to all output neurons $o_1, \ldots, o_k$.

An example of realizing a computation skeleton into a neural network is illustrated in Figure 2.

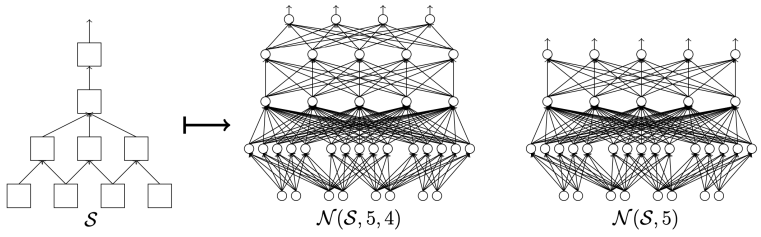

Figure 2: A (5,4)-fold and 5-fold realisations of the computation skeleton $\mathcal{S}$ with d=2.

Let $\delta(v)$ be the weight for neurons in $V(\mathcal{N})$. That is, for input neurons $\delta(v) = 1/d$, for internal neurons $\delta(v) = 1/r$, and for output neurons $\delta(v) = 1$. Such weights are assigned to neurons according to the architecture and the specific realization. However, for neural network, weights also need to be assigned to edges, and this paper [1] studies edge weights under random initialization in the way that for edge weights $w_{uv}, uv \in E(\mathcal{N})$

- If u is an input neuron, w_{uv} is sampled from a normal distribution with mean 0 and variance $\frac{d}{|in(v)|}$, where d is the dimension of the observation.
- Otherwise w_{uv} is sampled independently from a normal distribution with mean 0 and variance $\frac{1}{|\text{in}(v)|\|\sigma_u\|^2_{L^2_\mu}}$, where μ is the standard Gaussian measure.

2.3 Computation Skeleton and Reproducing Kernels

We have seen how computation skeleton can lead to neural networks, and then the authors illustrate how computation skeletons can induce reproducing kernels through dual activation and composition.

For $\rho \in [-1, 1]$ be the correlation coefficient, and denote N_ρ the multivariate Gaussian distribution on $\mathbb{R}^2$ with mean 0 and covariance matrix $\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$. Then the dual activation $\hat{\sigma}$ of an activation σ is defined as

$$\hat{\sigma}(\rho) = \mathbb{E}_{(X,Y)\sim N_\rho} \sigma(X)\sigma(Y) \tag{2}$$

and the dual kernel defined on a Hilbert space $\mathcal{H}$ is the kernel $\kappa_\sigma : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ such that for $f, g \in \mathcal{H}$

$$\kappa_\sigma(f, g) = \hat{\sigma}\langle f, g\rangle_{\mathcal{H}} \tag{3}$$

With dual activation and dual kernel, compositional kernels corresponding to a computation skeleton $\mathcal{S}$ with normalized activations and output node o can be defined inductively that

- For an input node v^i corresponding to the ith observation, the kernel at node v^i is $\kappa_{v^i}(x, y) = \langle x^i, y^i \rangle$ with the vector inner product in $\mathbb{R}^d$
- For internal node v, the kernel at v is

$$\kappa_v(x, y) = \hat{\sigma}_v\left(\frac{\sum_{u \in in(v)} \kappa_u(x, y)}{|in(v)|}\right) \tag{4}$$

- The final kernel $\kappa_{\mathcal{S}}$ for the skeleton $\mathcal{S}$ is κ_o, the compositional kernel inductively defined for the output node.

The resulting Hilbert space and norm corresponding to $\kappa_{\mathcal{S}}$ are denoted $\mathcal{H}_{\mathcal{S}}$ and $||.||_{\mathcal{S}}$. Similarly, $\mathcal{H}_v, ||.||_v$ denote the space and norm formed at node v. By the inductive nature, of the kernel, the kernel spaces can be written as

$$\mathcal{H}_v = \hat{\sigma}_v\left(\frac{\oplus_{u \in in(v)} \mathcal{H}_u}{|in(v)|}\right)$$

So far the theoretical kernel for the skeleton is defined. However, in practice we are working with neural networks with weights, and hence we need to define empirical quantities. Let $\mathcal{N}$ be a network and w the given edge weights, and we defined the normalized representation Ψ_w be the representation of the network normalized by $||\sigma_v||\sqrt{q}$, where q is the number of output neurons. In other words, Ψ_w is the normalized features/outputs generated by the given realized network. Then the empirical kernel corresponding to $\mathcal{N}, w$ is

$$\kappa_w(x, y) = \langle \Psi_w(x), \Psi_w(y) \rangle \tag{5}$$

For $x, y \in (\mathbb{S}^{d-1})^n$ datasets and vector inner product in $\mathbb{R}^q$. The empirical kernel space hence can be defined as

$$\mathcal{H}_{\kappa_w} = \mathcal{H}_w \equiv \{h_v(x) = \langle v, \Psi_w(x) \rangle | v \in \mathbb{R}^q\} \tag{6}$$

with norm in $\mathcal{H}_w$ defined as $||h||_w = inf\{||v|| : h = h_v\}$.

Q2: For convolutional skeletons, as shown in the Example2 in [1], we can find functions in $\mathcal{H}_{\mathcal{S}}$ that are more complex, provided the width q of the filter is small, comparing to those found in the compositional RKHS of skeletons with only fully-connected layers. They showed for skeleton $\mathcal{S}_2$ in Fig 1 with all activations taken as $\sigma(x) = e^x$ and q constant. For any $f \in \mathcal{H}_{\mathcal{S}_2}$,

$$\|f\|_{\mathcal{S}_2} \leq C\sqrt{n}\|f\|_2 = O(\sqrt{n}),$$

where C is a constant depends on q and the number of filters. The consequence of the findings is that, we generally want to have small normed functions in the RKHS because they are smoother, less complex, and in the setting of supervised learning, prevent overfitting.

2.4 Main Results: Empirical Kernel Approximates the Compositional Kernel

In the main results section, the paper provides some theoretical guarantees on how the empirical quantities can approximate theoretical quantities. Firstly, the convergence in probability for the empirical kernel to the compositional kernel on computation skeleton, under random initialization, is proved for C-bounded activations and for ReLU actitivation. By C-bounded activation, the activation function σ is twice continuously differentiable and $||\sigma||_\infty, ||\sigma'||_\infty, ||\sigma''||_\infty < C$ for constant C.

Theorem 2.1 *Let $\mathcal{S}$ be a skeleton with C-bounded activations. Let w be a random initialization of $\mathcal{N} = \mathcal{N}(\mathcal{S}, r, 1)$. For $\varepsilon > 0, \delta > 0$, assume*

$$r \geq \frac{(4C^4)^{depth(\mathcal{S})+1} \log(8|\mathcal{S}|/\delta)}{\varepsilon^2}$$

Then for all $x, x' \in \mathcal{X}$,

$$\mathbb{P}\{|\kappa_w(x, x') - \kappa_{\mathcal{S}}(x, x')| > \varepsilon\} \leq \delta \tag{7}$$

Theorem 2.2 *Let $\mathcal{S}$ be a skeleton with ReLU activations. Let w be a random initialization of $\mathcal{N} = \mathcal{N}(\mathcal{S}, r, 1)$. For $\varepsilon > 0, \delta > 0$, assume*

$$r \geq \frac{depth^2(\mathcal{S}) \log(|\mathcal{S}|/\delta)}{\varepsilon^2}$$

Then for all $x, x' \in \mathcal{X}$ and $\varepsilon \lesssim 1/depth(\mathcal{S})$,

$$\mathbb{P}\{|\kappa_w(x, x') - \kappa_{\mathcal{S}}(x, x')| > \varepsilon\} \leq \delta \tag{8}$$

2.5 Proof for Theorem 3 in [1]

We sketch the proof of the theorem that for ReLU activations the empirical kernel converge to the compositional kernel on skeleton. The proof heavily depends on the behaviors of the activation function. To fully study the activation function, the dual activation is generalized to be a map from the collection of 2×2 PSD matrices $\mathcal{M}_+$ to $\mathbb{R}$ such that

$$\bar{\sigma}(\Sigma) = \mathbb{E}_{(X,Y)\sim N(0,\Sigma)}\sigma(X)\sigma(Y)$$

and hence the dual kernel can be generalized as

$$\kappa_\sigma(x,y) = \bar{\sigma}\bigg(\begin{bmatrix} ||x||^2 & \langle x,y\rangle \\ \langle x,y\rangle & ||y||^2 \end{bmatrix}\bigg)$$

Denote

$$\mathcal{M}_+^\gamma \equiv \bigg\{ \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \in \mathcal{M}_+ | 1-\gamma \le \Sigma_{11}, \Sigma_{22} \le 1+\gamma \bigg\}$$

A normalize activation σ is (α,β,γ)-decent for $\alpha,\beta,\gamma > 0$ if the following conditions hold:

- The dual activation $\bar{\sigma}$ is β-Lipschitz in $\mathcal{M}_+^\gamma$ w.r.t the ∞-norm.
- If $(X_1,Y_1),\dots,(X_n,Y_n)$ are iid samples from $N(0,\Sigma)$ for $\Sigma \in \mathcal{M}_+^\gamma$, then

$$\mathbb{P}(|\frac{1}{n}\sum_{i=1}^n \sigma(X_i)\sigma(Y_i - \bar{\sigma}(\Sigma)|\varepsilon) \le 2exp\bigg(-\frac{r\varepsilon^2}{2\alpha^2}\bigg)$$

It is easy to check that ReLU activation is decent. To be specific, we have

Lemma 2.3 *There exists a constant $\alpha_0 \ge 1$ such that for $0 \le \gamma \le 1$, the normalized ReLU activation $\sigma(x) = \sqrt{2}max(0,x)$ is $(\alpha_0, 1+o(\gamma), \gamma)$-decent.*

Given the activation is decent, then the approximation of the empirical kernel can be shown by the theorem below, and the theorem on ReLU activation is an easy corollary.

Theorem 2.4 *Let $\mathcal{S}$ be a skeleton with (α,β,γ)-decent activations, $0 < \varepsilon \le \gamma$, and $B_d = \sum_{i=1}^{d-1}\beta^i$. Let w be a random initialization of the network $\mathcal{N} = \mathcal{N}(\mathcal{S}, r, 1)$. For $\varepsilon > 0, \delta > 0$, assume*

$$r \ge \frac{2\alpha^2(B_{depth(\mathcal{S})})^2 log(8|\mathcal{S}|/\delta)}{\varepsilon^2}$$

Then for every $x \in \mathcal{X}, y \in \mathcal{Y}$,

$$\mathbb{P}\{|\kappa_w(x,y) - \kappa_{\mathcal{S}}(x,y)| > \varepsilon\} \le \delta$$

Previous lemma gives that ReLU activation is $(\alpha_0, 1+o(\gamma), \gamma)$-decent, and combining with the theorem above we have the convergence of the empirical kernel with the fact that $(1+o(\varepsilon))^i \le e^{o(\varepsilon)depth(\mathcal{S})}$.

It remains to prove the theorem above. Related to the inductive natural of the compositional kernel, such consistency result is also proved in an inductive way. First some empirical quantities related to the empirical kernel are needed. For a node u, let $\Psi_{u.w} : \mathcal{X} \to \mathbb{R}^r$ be the normalized representation of the sub-skeleton at u, and $\kappa_{u,w}$ be the empirical kernel of the sub-network resulted in u. Given $x, y \in \mathcal{X}$ and a node $u \in \mathcal{S}$, define

$$\mathcal{K}_w^u = \begin{pmatrix} \kappa_{u,w}(x,x) & \kappa_{u,w}(x,y) \\ \kappa_{u,w}(x,y) & \kappa_{u,w}(y,y) \end{pmatrix}, \mathcal{K}^u = \begin{pmatrix} \kappa_u(x,x) & \kappa_u(x,y) \\ \kappa_u(x,y) & \kappa_u(y,y) \end{pmatrix},$$

which can be viewed as the covariance between the resulting features for x and y. Also define

$$\mathcal{K}_w^{\leftarrow u} = \frac{\sum_{v\in in(u)}\mathcal{K}_w^u}{|in(u)|}, \mathcal{K}^{\leftarrow u} = \frac{\sum_{v\in in(u)}\mathcal{K}^v}{|in(u)|}$$

Those are direct average of the matrices corresponding to nodes in the previous layer. It can be verified that $\mathcal{K}_w^u$ is the empirical covariance matrix of independent variables distributed according to $(\sigma(X), \sigma(Y))$ where $(X, Y) \sim N(0, \mathcal{K}_w^{\leftarrow u})$. Intuitively applying the dual activation $\hat{\sigma}$ to $^{\leftarrow\sqcap}$ would result in $\mathcal{K}^u$ and similarly for $\mathcal{K}_w^{\leftarrow u}$ and $\mathcal{K}_w^u$. To make this formally true we need to modify the generalized dual activation a little bit that

$$\hat{\sigma}^p(\mathcal{K}) = \begin{pmatrix} \hat{\sigma}\begin{pmatrix} \mathcal{K}_1 1 & \mathcal{K}_1 1 \\ \mathcal{K}_1 1 & \mathcal{K}_1 1 \end{pmatrix} & \hat{\sigma}(\mathcal{K}) \\ \hat{\sigma}(\mathcal{K}) & \hat{\sigma}\begin{pmatrix} \mathcal{K}_2 2 & \mathcal{K}_2 2 \\ \mathcal{K}_2 2 & \mathcal{K}_2 2 \end{pmatrix} \end{pmatrix},$$

Then we have $\mathcal{K}^u = \hat{\sigma}_u^p(\mathcal{K}^{\leftarrow u})$. Then to set up induction, the estimation performance of the empirical kernel at node $u \in \mathcal{S}$ is characterized by the concept of well-initialized, which is defined as

$$||\mathcal{K}_w^u - \mathcal{K}^u||_\infty \leq \varepsilon \frac{B_{depth(u)}}{B_{depth(\mathcal{S})}}$$

Lemma 2.5 *Assume that all the nodes in $in(u)$ are well-initiated. Then the node u is well-initiated with probability of at least $1 - \delta/|\mathcal{S}|$.*

This lemma is easy to verify and note that the input nodes are all well-initiated with probability 1, and hence inductively we conclude the proof.

2.6 NN Approximates Compositional RKHS with Random Weight Initialisation

Followed by the approximation result of the empirical kernel $\kappa_{\boldsymbol{w}}$ to the compositional kernel $\kappa_{\mathcal{S}}$, the authors also provides the approximation of the $\mathcal{H}_{\boldsymbol{w}}$ to the $\mathcal{H}_{\mathcal{S}}$, in terms of the ϵ-approximation of population risk, which is defined as follows

Definition 2.6 *Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \mathcal{Y}$. A space $\mathcal{H}_1 \subset \mathbb{R}^{\mathcal{X}}$ ε-approximates $\mathcal{H}_2 \subset \mathbb{R}^{\mathcal{X}}$ w.r.t $\mathcal{D}$ if for every $h_2 \in \mathcal{H}_2$ there is $h_1 \in \mathcal{H}_1$ such that $\mathcal{L}_{\mathcal{D}}(h_1) \leq \mathcal{L}_{\mathcal{D}}(h_2) + \varepsilon$*

Then for L-Lipschitz loss ℓ, the ε-approximation between $\mathcal{H}_{\mathcal{S}}$ and $\mathcal{H}_w$ is provided for skeletons with C-bounded activations and ReLU activations

Theorem 2.7 *Let $\mathcal{S}$ be a skeleton with C-bounded activations and let $R > 0$. Let $\boldsymbol{w}$ be a random initialisation of $\mathcal{N}(\mathcal{S}, r)$ with*

$$r \gtrsim \frac{L^4 R^4 (4C^4)^{depth(\mathcal{S})+1} \log\left(\frac{LRC|\mathcal{S}|}{\varepsilon\delta}\right)}{\varepsilon^4}$$

Then, with probability of at least $1 - \delta$ over the choices of $\boldsymbol{w}$ we have that, for any data distribution $\mathcal{D}$, $\mathcal{H}_{\boldsymbol{w}}^{\sqrt{2}R}$ ε-approximates $\mathcal{H}_{\mathcal{S}}^R$ and vice versa.

We examine the special case with ReLU activation function, which adds an extra condition to the Theorem 2.7. i.e. with $\epsilon \lesssim 1/\text{depth}(S)$, it has the constraint that

$$r \gtrsim \frac{L^4 R^4 \text{depth}^2(\mathcal{S}) \log\left(\frac{\|\mathcal{D}\|_0|\mathcal{S}|}{\delta}\right)}{\varepsilon^4}$$

Then, the same conclusion follows.

3 Empirical Study

We then study Theorem 2.7 with simulations on a simple skeleton in Fig 3.1 with ReLU activation functions $\sigma(x) = \max(0, x)$ taken for all nodes, and weight initialize according to the construction in Section 2.2 illustrated as in the [1].

3.1 Setup

We implement a simple binary classification on the first two classes of MNIST dataset, where each 28×28 image is preprocessed to be in $\mathcal{X} = (\mathbb{S}^{28-1})^{28}$; and we let $\mathcal{Y} = \{0, 1\}$. Then, since $\|\sigma\|_{\mathcal{L}^2_\mu} = \sqrt{2}$, the weights $w_{uv}, uv \in E(\mathcal{N})$ are randomly initialize as follows

- If $u \in V(\mathcal{N})$ in the input layer of NN, $w_{uv} \sim N(0, 1)$
- Otherwise, $w_{uv} \sim N(0, \frac{1}{300\times 2})$

Note that, the condition on the number of neurons at each node in the hidden layers corresponds to the assumption that r is large enough comparing to the depth of the skeleton $\mathcal{S}$, and the smoothness of the functions in the two RKHSs, where $\|f\|_{\kappa_{\boldsymbol{w}}} \leq \sqrt{2}R$ and $\|g\|_{\kappa_{\mathcal{S}}} \leq R$, for $f \in \mathcal{H}_{\boldsymbol{w}}$ and $g \in \mathcal{H}_{\mathcal{S}}$ respectively.

Therefore, with 1 hidden layers, we set $r = 300$ as the number of neurons, and the normalized representation vector $\Psi_{\boldsymbol{w}}(\boldsymbol{x}) \in \mathbb{R}^{300}$. Define an empirical kernel $\kappa_{\boldsymbol{w}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \Psi_{\boldsymbol{w}}(\boldsymbol{x}_i), \Psi_{\boldsymbol{w}}(\boldsymbol{x}_i)\rangle$, and the composition kernel $\kappa_{\mathcal{S}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma(\frac{\sum_{k=1}^{28}\langle \boldsymbol{x}_i^k, \boldsymbol{x}_j^k\rangle}{28})$, for $\boldsymbol{x}_i^k, \boldsymbol{x}_j^k \in \mathbb{S}^{28-1}$, where $\sigma(\rho) = \frac{\sqrt{1-\rho^2}+(\pi-\cos^{-1}(\rho))\rho}{\pi}$, the dual kernel of ReLU.

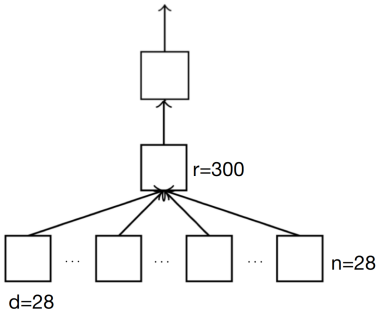

Figure 3: Simple skeleton $\mathcal{S}$ in the simulation study, with the number of input neurons $n = 28$ each with a vector of size $d = 28$, and the number of neurons in the hidden layers $r = 300$. Moreover, $q = 300$. The two hidden layers have activation function ReLU, and the output layer has the identity map.

Under the supervised learning setting, we may take any L-Lipschitz and convex loss function. If a square loss function is assumed, the dual of the optimisation problem reduces to a Kernel ridge regression (KRR) objective, i.e.

$$\min_{f\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} \|f(\boldsymbol{x}_i) - y_i\|_2^2 + \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2,$$

for the two RKHS $\mathcal{H} = \mathcal{H}_{\boldsymbol{w}}$ and $\mathcal{H}_{\mathcal{S}}$, where regularisation parameter $\lambda > 0$ corresponds to some R s.t. $\|f\|_{\boldsymbol{w}} \leq R$ and similarly for $\|f\|_{\mathcal{S}}$. The KRR admits a closed form solution by invoking Representre Theorem, where $f = \sum_{i=1}^{m} \alpha_i \kappa(\boldsymbol{x}, \cdot)$, then $\boldsymbol{\alpha} = (K + m\lambda I_m)^{-1}\boldsymbol{y}$, $\boldsymbol{y} \in \mathbb{R}^m$ and K the gram matrix given by $\langle \kappa(\boldsymbol{x}, \cdot), \kappa(\boldsymbol{x}, \cdot)\rangle$ for the two kernels $\kappa = \kappa_{\boldsymbol{w}}$ and $\kappa = \kappa_{\mathcal{S}}$.

Therefore, provided with any test sample $\boldsymbol{x}_j \in \mathcal{X}$, the prediction $f(\boldsymbol{x}_j) = \langle f, \kappa(\boldsymbol{x}_j, \cdot)\rangle_{\mathcal{H}}$ can be evaluated with MSE, for the solutions found $f = f_{\boldsymbol{w}}$ and $f_{\mathcal{S}}$. Thus, the Theorem 2.7 states that the solution $f_{\boldsymbol{w}}$ to the KRR in $\mathcal{H}_{\boldsymbol{w}}$ is close to the solution $f_{\mathcal{S}} \in \mathcal{H}_{\mathcal{S}}$ in terms of the population risk, given a regularisation parameter $\lambda > 0$.

Notably, if we instead use hinge loss for the binary classification problem, we set $\mathcal{Y} = \{-1, 1\}$, the dual problem is reduces to the Kernel SVM. And admits the following form,

$$\min_{f\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} \max(0, 1 - y_i f(x_i))^p + \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2,$$

which is convex and smooth for $p = 2$, thus gradient method can be applied to solve the problem with Representre Theorem. And when $p = 1$, sub-gradient methods can be applied.

Furthermore, we can also study the performance of the two kernels for the skeletons with Sigmoid-like activation functions. e.g. for sigmoid activation $\sigma(x) = \frac{1}{1-e^{-x}}$, the dual kernel can be found as $\hat{\sigma}(\rho) = \sum_{\alpha,\beta=0}^{\infty} \exp\{\frac{1}{2}(\alpha^2 + \beta^2 + 2\rho\alpha\beta)\}$.

3.2 Results

We run the simulation for increasing size of training set with balanced two classes. And perform the KRR for the two kernels with increasing λ. We included three examples as in Fig **??**, **??**, **??** and additionally **??** for a closer look at the MSE for small $\lambda \in (0, 5)$. And as in Fig 4, the smallest MSE obtained by the empirical kernel improves as train set size increases, and the improvement of compositional kernel is not as evident. However, if we look at individual plot for varying λ, we see that the MSE of both kernels converges more slowly to 0.56 for growing train set size.

We note that, for the calculation of the MSE, we restrict < 0 predictions to 0, and > 1 predictions to 1. And for the skeleton we chose, the empirical kernel is less sensitive to the increase of regularisation parameter, i.e. the MSE has a slower convergence to 0.56 (the test set class "1" ratio). Since for our chosen skeleton, the $\Psi_{\boldsymbol{w}}$ has relatively large entry values, and consequently for the gram matrix. And according to the normalisation scheme introduced by the authors, the empirical kernel obtained does not guarantee $\kappa_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{x}') \in [0, 1]$. However, the compositional kernel does.

In addition, the choice of n and r is important since it dictates the variances of the initialize weights, and for skeletons that are not “reasonable", e.g. input layer taking $n = 28 \times 28$, $d = 1$ for vectorised image supported on the sphere, the weights can be really small and empirical kernel KRR yields close to 0 predictions for all test cases. Similarly, if we take several fully connected layers with large r, which may not be wise for any machine learning task using NN, the resulting predictions are all close to 0.

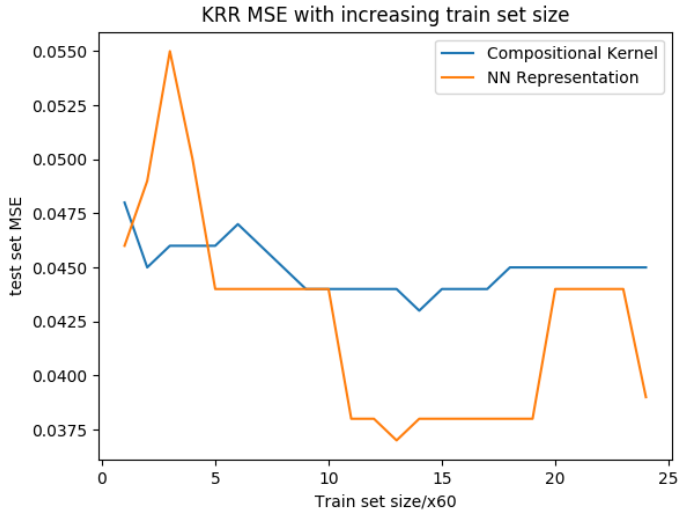

Figure 4: MSE for KRR with increasing train set size and test set (80, 103) for the empirical kernel defined for NN representation and compositional kernel defined on the skeleton.

References

[1] Amit Daniely, Roy Frostig, and Yoram Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2253–2261. Curran Associates, Inc., 2016.

Appendices

```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
from itertools import product

# Import MNIST data all classes
    ----------------------------------------------------------------------
n_epochs = 1
batch_size_train = 64 + 200
batch_size_test = 1000 # fixed for 300 test case

# Importing MNIST data: first 2 classes
    ---------------------------------------------
from torch.utils.data._utils.collate import default_collate
def my_collate(batch):
    modified_batch = []
    for item in batch:
        image, label = item
        if label==1 or label==2:
            modified_batch.append(item)
    return default_collate(modified_batch)

train_loader12 = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('data12_train.pt', train=True, download=True,
    transform=torchvision.transforms.Compose([
                            torchvision.transforms.ToTensor()
                           ])),
    batch_size=batch_size_train, shuffle=True, collate_fn=my_collate)

test_loader12 = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('data12_test.pt', train=False, download=True,
    transform=torchvision.transforms.Compose([
                            torchvision.transforms.ToTensor()
                           ])),
    batch_size=batch_size_train, shuffle=True, collate_fn=my_collate)

# Building the network
    ------------------------------------------------------------------
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28, 1)
        self.fc2 = nn.Linear(28, 300)

    def forward(self, x):
        # flatten from 28x28x1 to 784
        x = self.fc1(x)
        x = x.view(-1, 28)
        x = F.relu(self.fc2(x))

        return x

# intialise weight
def init_weights(m):
     classname = m.__class__.__name__
     # for every Linear layer in a model..
```

```
    if classname.find('Linear') != -1:
        # get the number of the inputs
        n = m.in_features
        print("number of feature is ", n)
        if(n==28):
            m.weight.data.normal_(0, 1)
        # get the numnber of channels
        else:
            m.weight.data.normal_(0, 1/(n*2))
        m.bias.data.fill_(0)


# get NN representation layer ----------------------------------------------------
def getFeature(data_loader, nbatch=1):
    network.train()

    random_seed = 13223
    torch.backends.cudnn.enabled = False
    torch.manual_seed(random_seed)

    n = 0
    imag_vec = torch.Tensor()
    feature = torch.Tensor()
    label = torch.LongTensor()

    for batch_idx, (data, target) in enumerate(data_loader):
        n += 1
        # normalise the data
        norm_dat = data/torch.norm(data, p=2, dim=(1,2), keepdim=True)
        norm_dat[torch.isnan(norm_dat)] = 0

        imag_vec = torch.cat((imag_vec, norm_dat.view(-1, 784)))

        feature = torch.cat((feature, network(norm_dat)))
        label = torch.cat((label, target))

        if n == nbatch:
            # change the labels of second class to 0
            label[label==2] = 0
            # check the proportion of classes
            check_labs = label.data.numpy()
            print(np.unique(check_labs, return_counts=True))

            return imag_vec, feature, label


# ReLU kernel arccos0
    ----------------------------------------------------------------------
def Kernel_relu(p): return (torch.sqrt(1-p**2) + (np.pi - torch.acos(p)) * p)/np.pi

def centering(ker):
    n, m = ker.shape
    return ((torch.eye(n) - torch.ones((n,n))/n) @ ker @ (torch.eye(m) -
        torch.ones((m,m))/m))

# dat_train, dat_test, lab_train, lab_test are tensors
# l the regularisation parameter, corresonding to ||f||<R
def kernelRidge_12(dat_train, dat_test, lab_train, lab_test, if_NN=False):
    n = dat_train.shape[0]
    vlen = dat_train.shape[1]
    print("length of representation is", vlen)

    if if_NN:

        # normalise
```

```
            norm_dat_train = dat_train/(np.sqrt(vlen*2))
            norm_dat_test = dat_test/(np.sqrt(vlen*2))

            # find gram matrix
            Knn = torch.mm(norm_dat_train, norm_dat_train.t())
            Knm = torch.mm(norm_dat_train, norm_dat_test.t())
            print("NN gram", Knn[1:10, 1:10])
        else:
            # find compositional kernel for the skeleton, 3 hidden layers
            Knn = Kernel_relu(torch.matmul(dat_train, dat_train.t())/28)
            Knm = Kernel_relu(torch.matmul(dat_train, dat_test.t())/28)
            # print(torch.norm(dat_train, p=2, dim=1))
            print("Skeleton kernel", Knn[0:10, 0:10])

        return Knn, Knm

    def pred_KRR(l, lab_train, lab_test, Knn, Knm):
        n = lab_train.shape[0]
        # prediction
        pred = lab_train.float() @ torch.inverse(Knn/n + l*torch.eye(n)) @ Knm

        pred[pred<0] = 0
        pred[pred>1] = 1

        # get MSE for test set
        error = torch.mean((lab_test - pred)**2)

        return error.data.numpy()



    # plot the test set MSR
    def plot_MSR(l_lis, nbatch=1, plot=True):
        imagvec_train, repvec_train, lab_train = getFeature(train_loader12,
            nbatch=nbatch)

        imagvec_test, repvec_test, lab_test = getFeature(test_loader12, nbatch=3)

        Knn_s, Knm_s = kernelRidge_12(imagvec_train, imagvec_test,
                                      lab_train, lab_test, if_NN=False)
        Knn_w, Knm_w = kernelRidge_12(repvec_train, repvec_test,
                                      lab_train, lab_test, if_NN=True)
        cKnn_s = centering(Knn_s); cKnm_s = centering(Knm_s)
        cKnn_w = centering(Knn_w); cKnm_w = centering(Knm_w)


        error_lis_s = []
        error_lis_w = []
        for l in l_lis:
            error_lis_s += [pred_KRR(l, lab_train, lab_test, cKnn_s, cKnm_s)]
            error_lis_w += [pred_KRR(l, lab_train, lab_test, cKnn_w, cKnm_w)]

        temp_s = np.min(np.array(error_lis_s))
        min_error_s = np.round(temp_s.astype(float), decimals=3)
        argmin_error_s = np.round(l_lis[np.argmin(np.array(error_lis_s))], decimals=3)

        temp_w = np.min(np.array(error_lis_w))
        min_error_w = np.round(temp_w.astype(float), decimals=3)
        argmin_error_w = np.round(l_lis[np.argmin(np.array(error_lis_w))], decimals=3)

        mins_w = np.array(error_lis_s)[np.argmin(np.array(error_lis_w))]
        diff_argmins_w = np.round(np.abs(mins_w.astype(float) - min_error_w),
            decimals=3)

        if plot:
```

```
        plt.figure()
        plt.plot(l_lis, error_lis_s, label='Compositional Kernel')
        plt.plot(l_lis, error_lis_w, label='NN Representation')
        plt.hlines(min_error_s, xmin=l_lis.min(), xmax=l_lis.max(),
                   label='Comp:Min Error {a}; Argmin {b}'.format(a=min_error_s,
                       b=argmin_error_s),
                   linestyle=':')
        plt.hlines(min_error_w, xmin=l_lis.min(), xmax=l_lis.max(),
                   label='NN:Min Error {a}; Argmin {b}'.format(a=min_error_w,
                       b=argmin_error_w),
                   linestyle='-.')
        plt.title('Kernel Ridge Regression with Square Loss')
        plt.xlabel('Regularisation parameter $\lambda$/Difference at NN argmin is
            {}'.format(diff_argmins_w))
        plt.ylabel('test set MSE')
        plt.legend()
        plt.savefig('KRR_test{}.png'.format(nbatch))
        plt.show()
        plt.close()

    return min_error_s, min_error_w, diff_argmins_w

if __name__=='__main__':
    import os
    os.environ['KMP_DUPLICATE_LIB_OK']='True'

    network = Net().apply(init_weights)

    l_lis = np.linspace(-0.01, 10, 300)

    plot_MSR(l_lis, nbatch=12)

    # min_lis_s = []; min_lis_w = []; diff_s_w = []
    # for nbatch in np.arange(1, 25):
    #     s, w, s_w = plot_MSR(l_lis, nbatch=nbatch)
    #     min_lis_s += [s]; min_lis_w += [w]; diff_s_w += [s_w]

    # # min_min = np.min(np.array(min_lis))
    # plt.figure()
    # plt.plot(np.arange(1, 25), min_lis_s, label='Compositional Kernel')
    # plt.plot(np.arange(1, 25), min_lis_w, label='NN Representation')
    # plt.title('KRR MSE with increasing train set size')
    # plt.xlabel('Train set size/x60')
    # plt.ylabel('test set MSE')
    # plt.legend()
    # plt.savefig('KRR_nbatch.png')
    # plt.show()
    # plt.close()

    # plt.figure()
    # plt.plot(np.arange(1, 25), diff_s_w)
    # plt.title('Difference of MSE at argmin Empirical Kernel KRR')
    # plt.xlabel('Train set size/x60')
    # plt.ylabel('test set MSE')
    # plt.legend()
    # plt.savefig('KRR_diff.png')
    # plt.show()
    # plt.close()
```