Neural Tangent Kernel

Neural Tangent Kernel Formula

L-layer NN. For h = 1,…,L:

$$\Sigma^{(0)}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}',$$

$$\boldsymbol{\Lambda}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') = \begin{pmatrix} \Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') \\ \Sigma^{(h-1)}(\boldsymbol{x}', \boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x}', \boldsymbol{x}') \end{pmatrix} \in \mathbb{R}^{2\times 2},$$

$$\Sigma^{(h)}(\boldsymbol{x}, \boldsymbol{x}') = c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}\left(\mathbf{0}, \boldsymbol{\Lambda}^{(h)}\right)} \left[\sigma(u)\,\sigma(v)\right],$$

$$\dot{\Sigma}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') = c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}\left(\mathbf{0}, \boldsymbol{\Lambda}^{(h)}\right)} \left[\dot{\sigma}(u)\dot{\sigma}(v)\right].$$

L-layer recursion.
Encodes NN's architecture.

Dependency on the derivative:
Gradient decent algorithm.

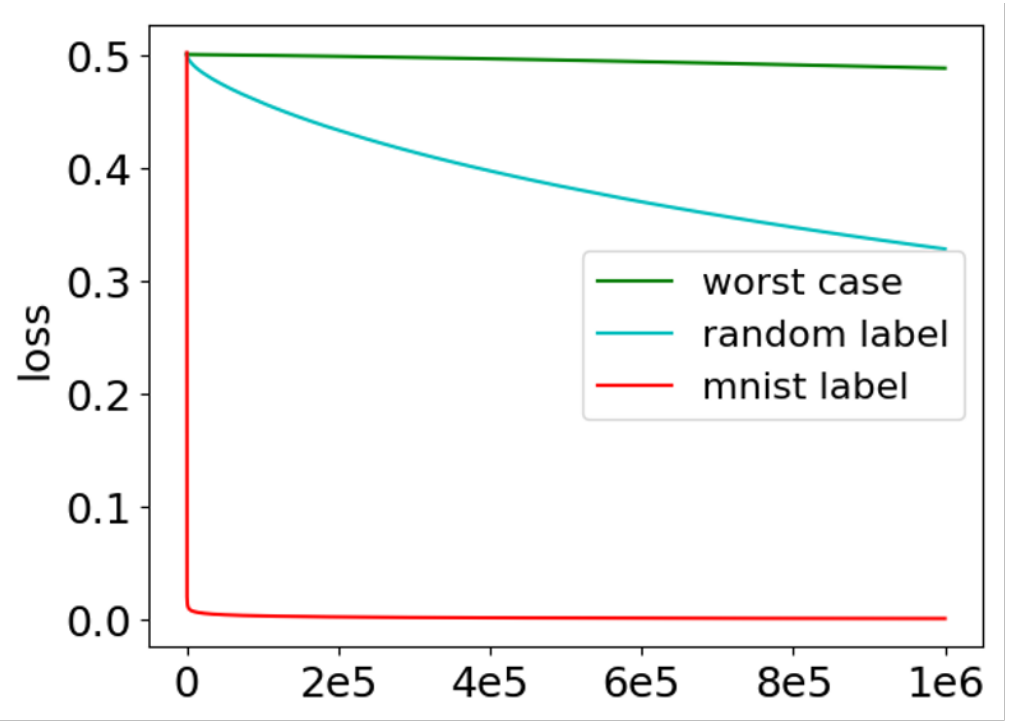
Final output:

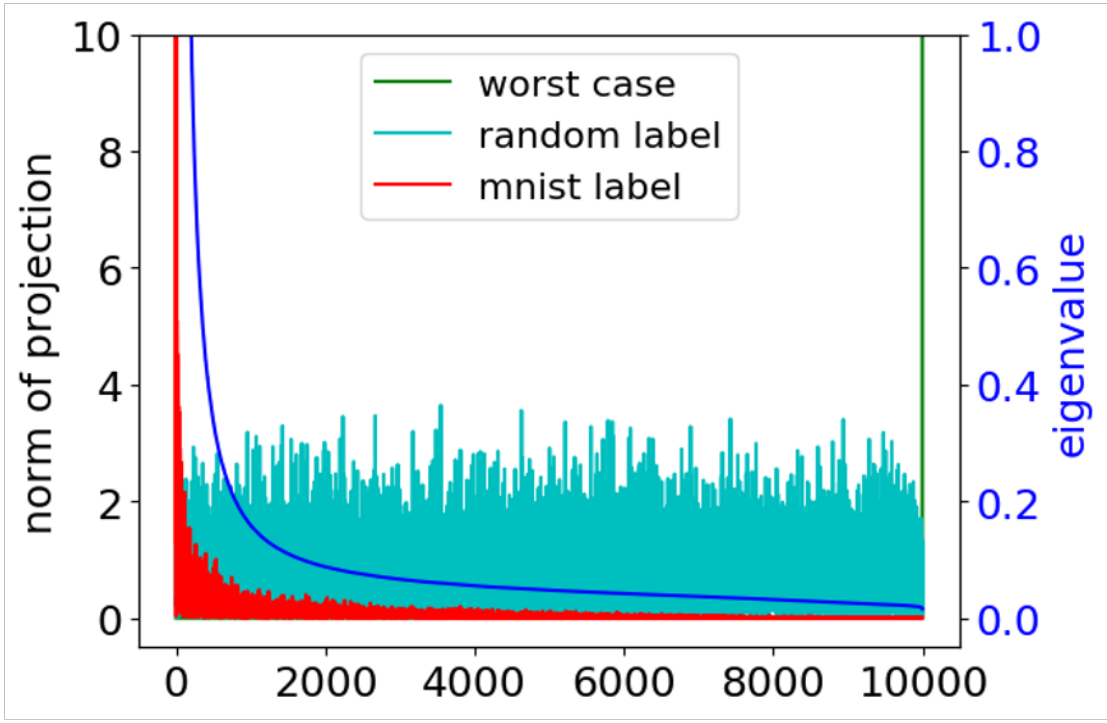
$$\Theta^{(L)}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{h=1}^{L+1} \left(\Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') \cdot \prod_{h'=h}^{L+1} \dot{\Sigma}^{(h')}(\boldsymbol{x}, \boldsymbol{x}') \right)$$

What determines the convergence rate?

$H^* = \sum_{i=1}^{n} \lambda_i v_i v_i^T$, λ_i: eigenvalue, v_i: eigenvector

$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_n$





Convergence Rate $\|y - u(t)\|_2^2 \propto \sum_{i=1}^{n} \exp(-\lambda_i t)(v_i^T y)^2$, $y \in \mathbb{R}^n$

Projections

y concentrates on large eigenvector

Neural Tangent Kernel

Recipe for designing new kernels

$$f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x\right) \Rightarrow k\left(x, x'\right) = \mathbb{E}_{\theta_{\mathrm{NN}} \sim \mathcal{W}}\left[\left\langle \frac{\partial f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x\right)}{\partial \theta_{\mathrm{NN}}}, \frac{\partial f_{\mathrm{NN}}\left(\theta_{\mathrm{NN}}, x'\right)}{\partial \theta_{\mathrm{NN}}} \right\rangle\right]$$

Transform a neural network of any architecture to a kernel!

Fully-connected NN → Fully-connected NTK

Convolutional NN → Convolutional NTK

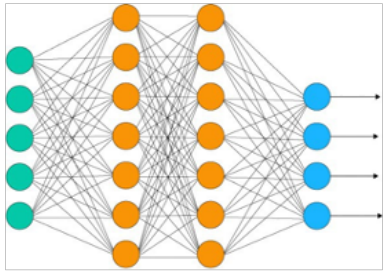
Graph NN → Graph NTK

.....

Fully-Connect NTK

$$\begin{pmatrix} -0.1 \\ 0.2 \\ \cdots \\ 0.9 \end{pmatrix}$$

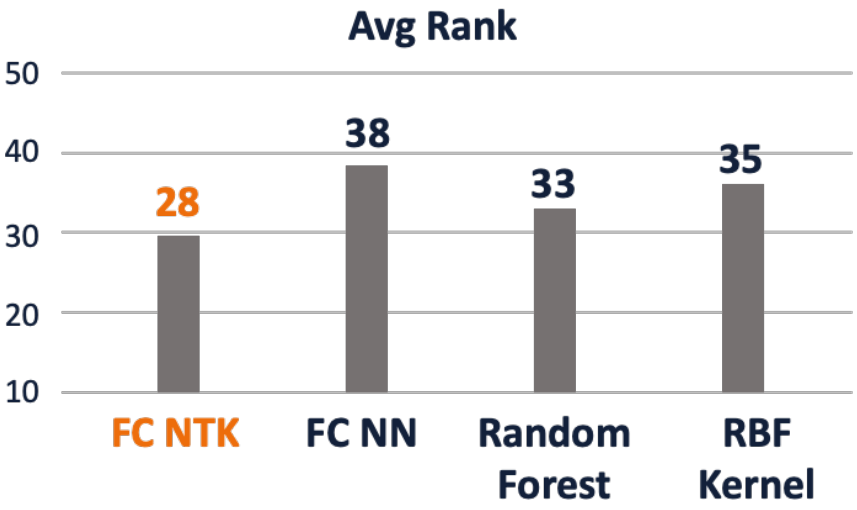
Features



FC NN

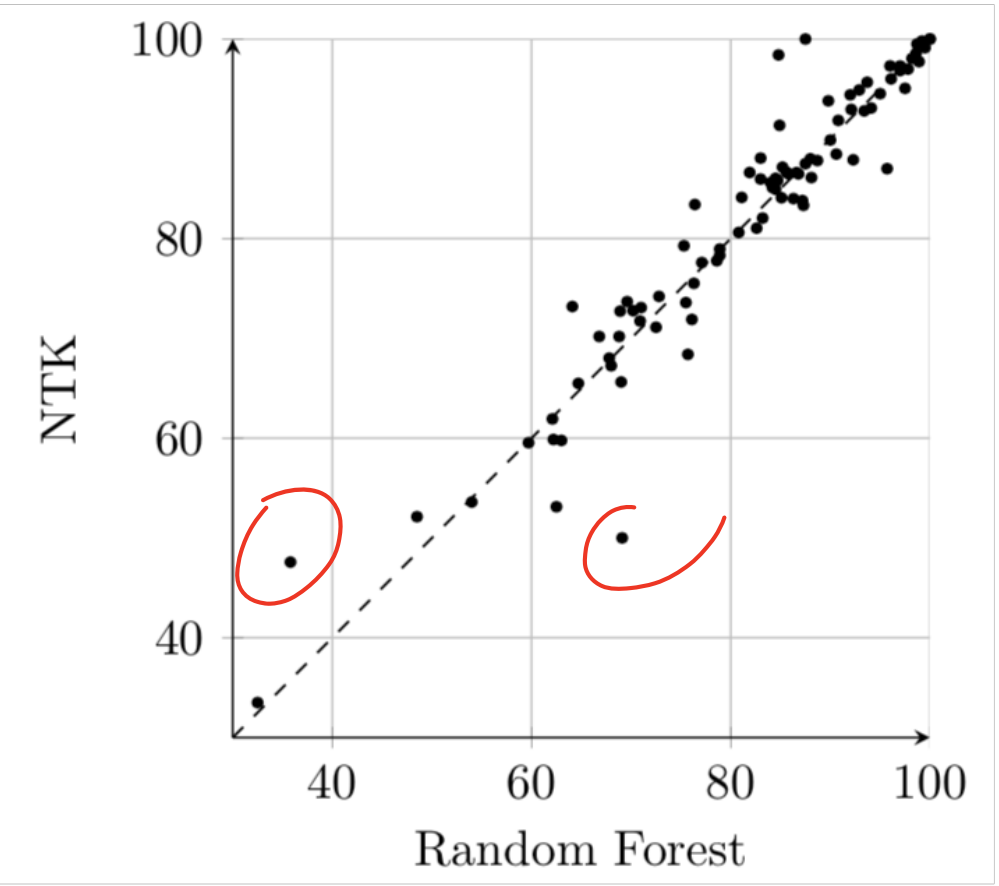
$$k\left(\begin{pmatrix} -0.1 \\ 0.2 \\ \cdots \\ 0.9 \end{pmatrix}, \begin{pmatrix} -0.3 \\ 0.5 \\ \cdots \\ -0.8 \end{pmatrix}\right)$$

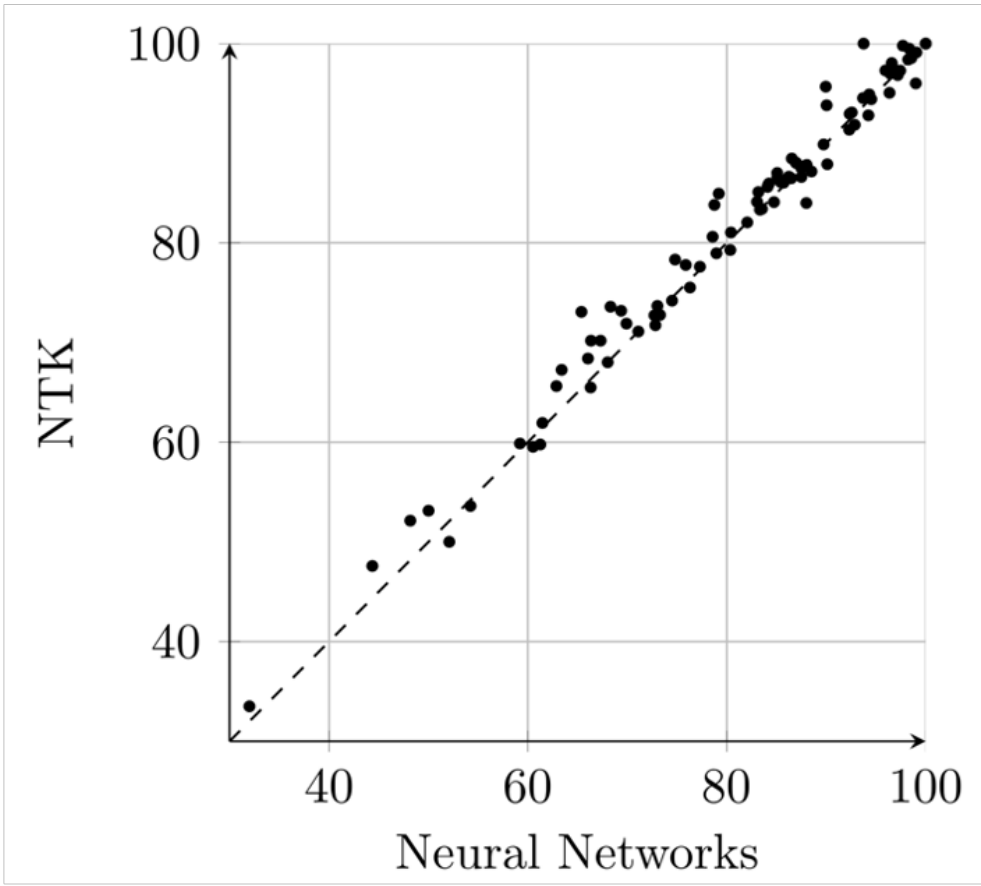
FC NTK



Classifier	Avg Acc	P95	PMA
FC NTK	82%	72%	96%
FC NN	81%	60%	95%
Random Forest	82%	68%	95%
RBF Kernel	81%	72%	94%

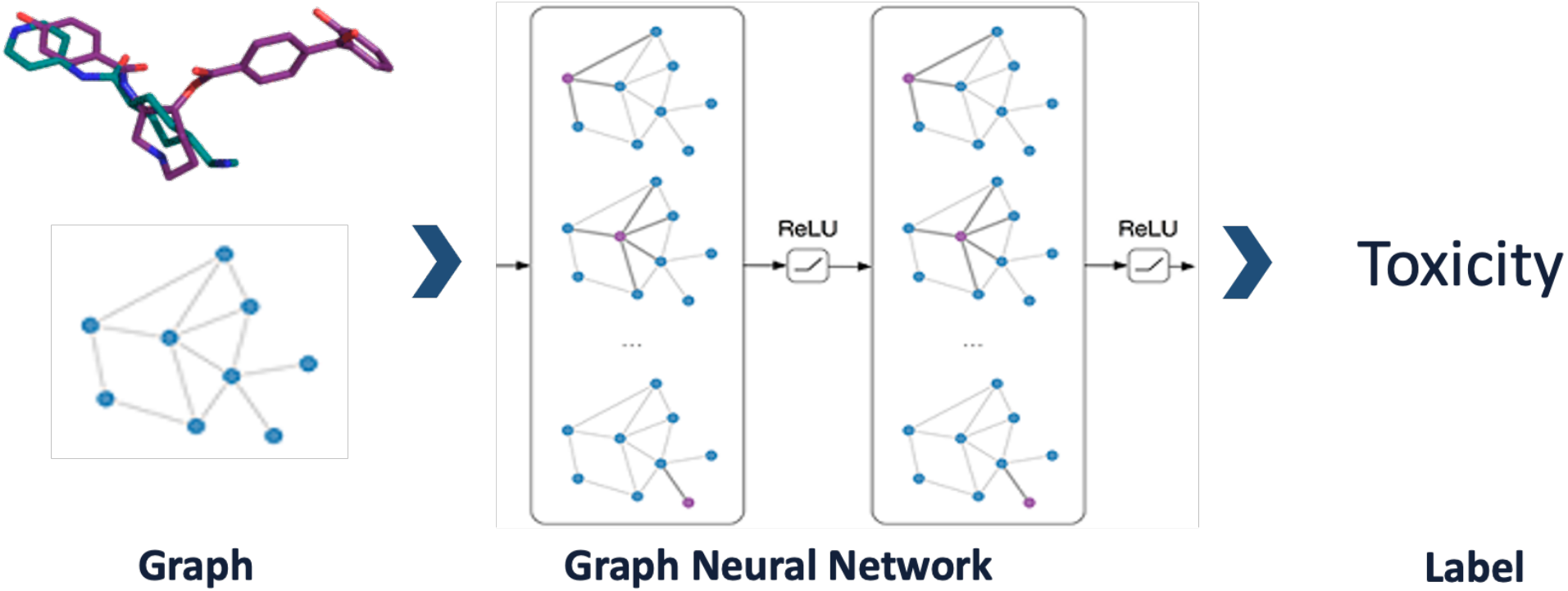
Pairwise Comparisons





Classification Accuracy

Graph Neural Network

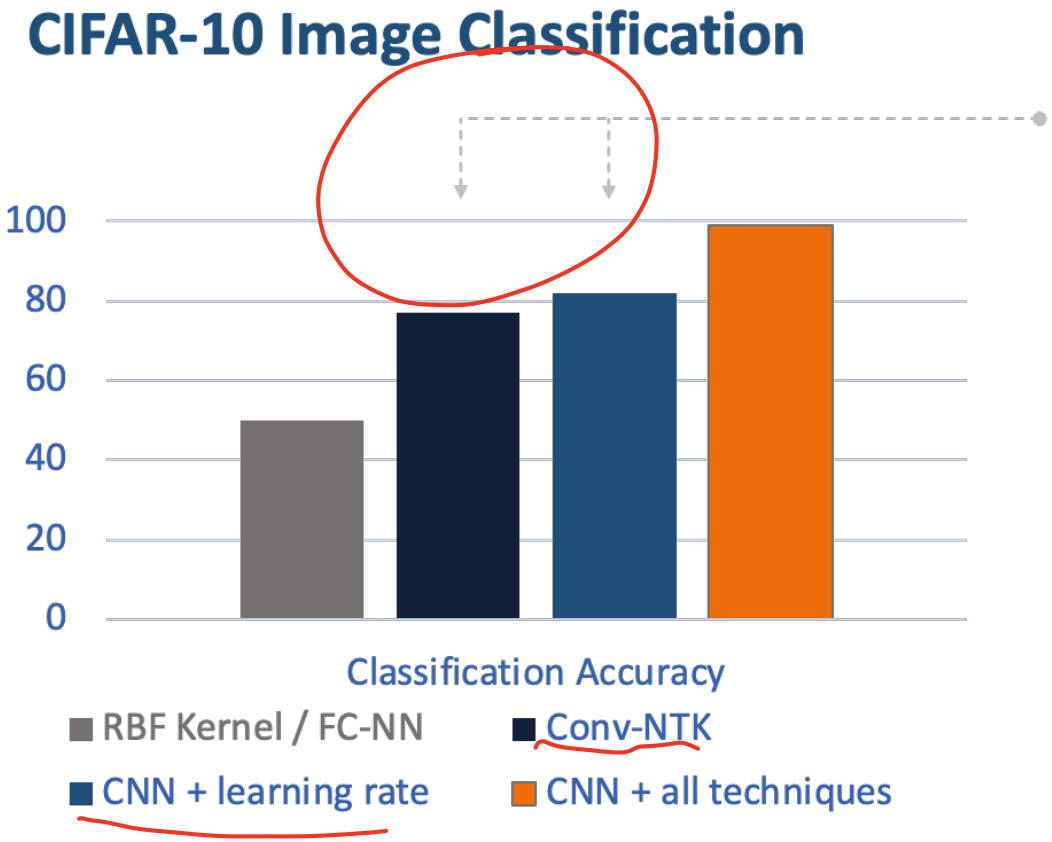


Graph Neural Tangent Kernel

Graph **Graph NN** **Graph NTK**

	Method	COLLAB	IMDB-B	IMDB-M	PTC
GNN	GCN	79%	74%	51%	64%
	GIN	80%	75%	52%	65%
GK	WL	79%	74%	51%	60%
	GNTK	84%	77%	53%	68%

Gap between NN and NTK



Open Problems:

Why there is a gap:

finite-width?

learning rate?

Understanding techniques:

batch-norm

dropout

data-augmentation

...