Practical - Analysis of new phenomena in machine/deep learning

Pascal M. Esser

Outline

Machine learning and deep learning research

- Empirical studies, providing benchmark and demonstrating pitfalls.
- Rigorously explain why ML / DL works by analysing theoretical models or algorithms.

Focus:

- Insights for new algorithmic development (example: boosting, methods for regularisation).
- Brings concepts from mathematics to ML (example: Random graphs, Geometry).

Machine learning and deep learning research

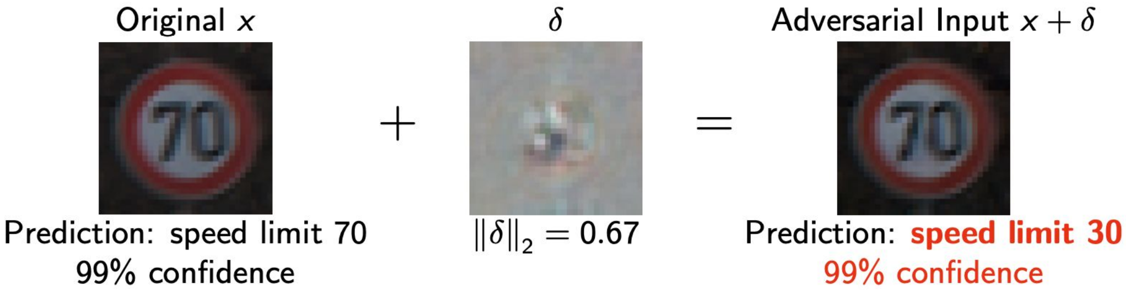
This Practical:

- Understand recent advances
- Reproduce existing results
- Extend research (empirically)

Topics

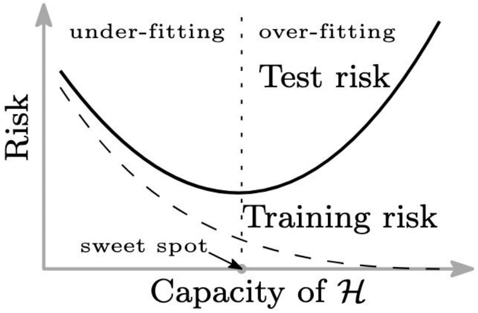
Adversarial ML / Robustness

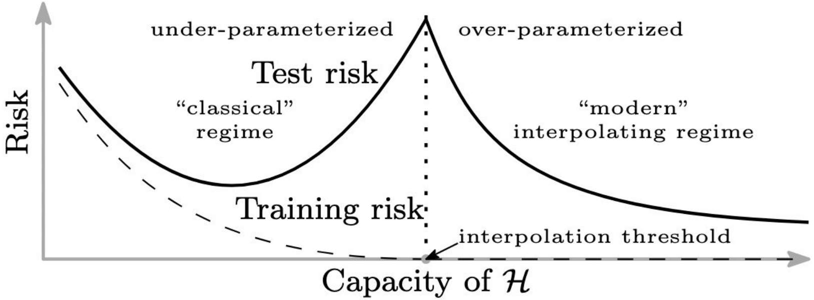
- Performance of NNs significantly affected if data is slightly perturbed.
- Why? How can we robust ML models / guarantee robustness?


Original x
δ
Adversarial Input $x + \delta$
+
=
Prediction: speed limit 70
99% confidence
$\|\delta\|_2 = 0.67$
Prediction: speed limit 30
99% confidence

Generalisation in neural networks

- Over-parameterised NNs deviate from bias-variance trade-off - NNs may perform best in zero training loss / interpolating regime.
- Currently, this behaviour has been analytically derived in simpler settings.



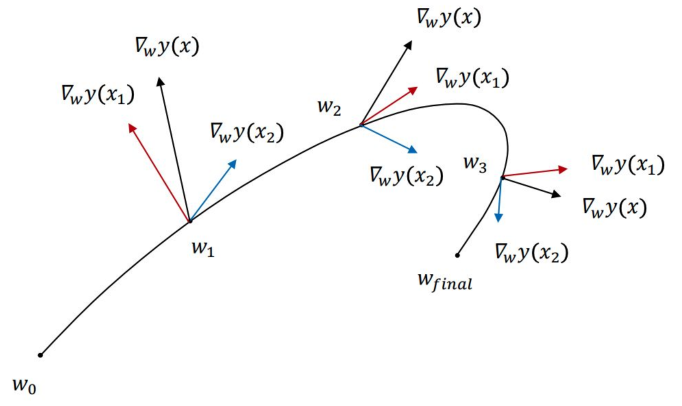


Over-parameterised NN (infinite width)

Analyse Over-parametrised NNs asymptotically as width goes to infinity

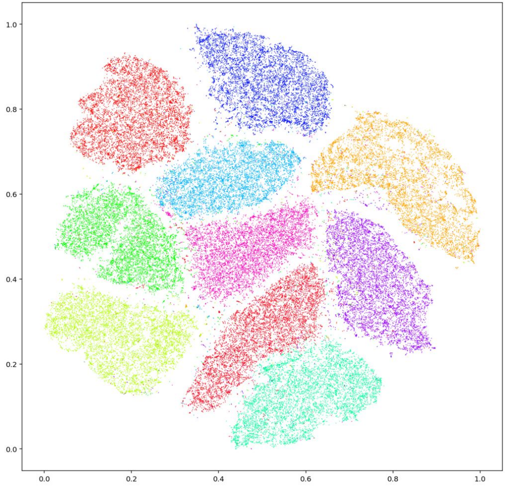
- Under small learning rate, (S)GD training ≡ Neural Tangent Kernel (NTK), a dot product kernel in gradient space of the NN parameters
- Finite width networks can deviate from the kernel regime.

$$NTK(x_i, x_j) = \nabla_w y(x_i) \, . \, \nabla_w y(x_j)$$

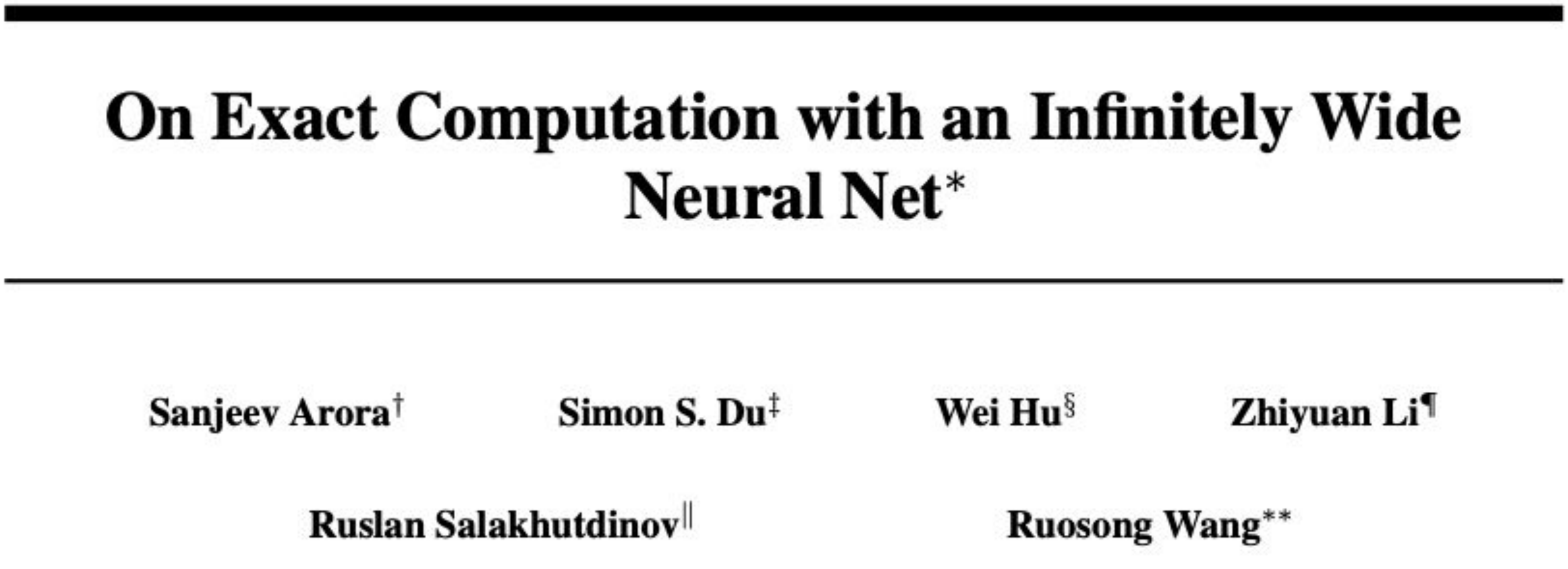


Unsupervised Learning

- What representations do we learn?
- What are (provable) better embeddings?



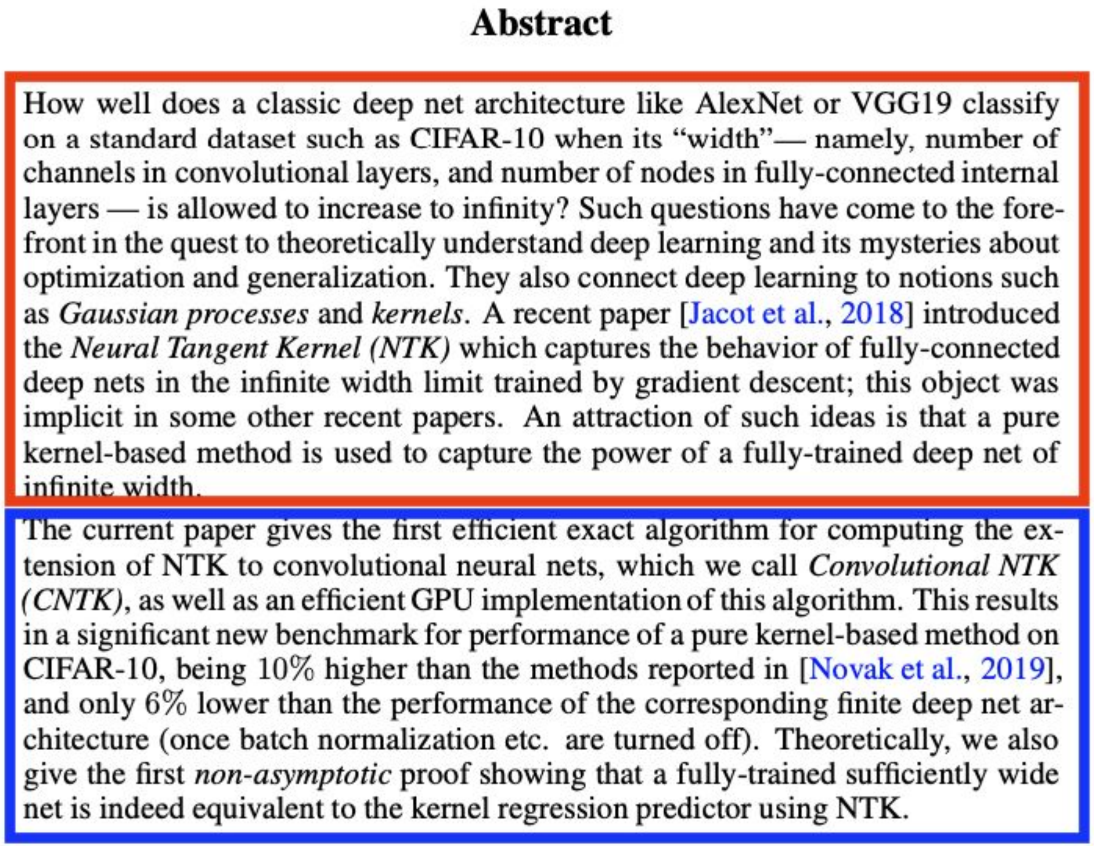
Let’s look at an example



On Exact Computation with an Infinitely Wide Neural Net[*]

Sanjeev Arora[†] **Simon S. Du**[‡] **Wei Hu**[§] **Zhiyuan Li**[¶]

Ruslan Salakhutdinov[||] **Ruosong Wang**[**]



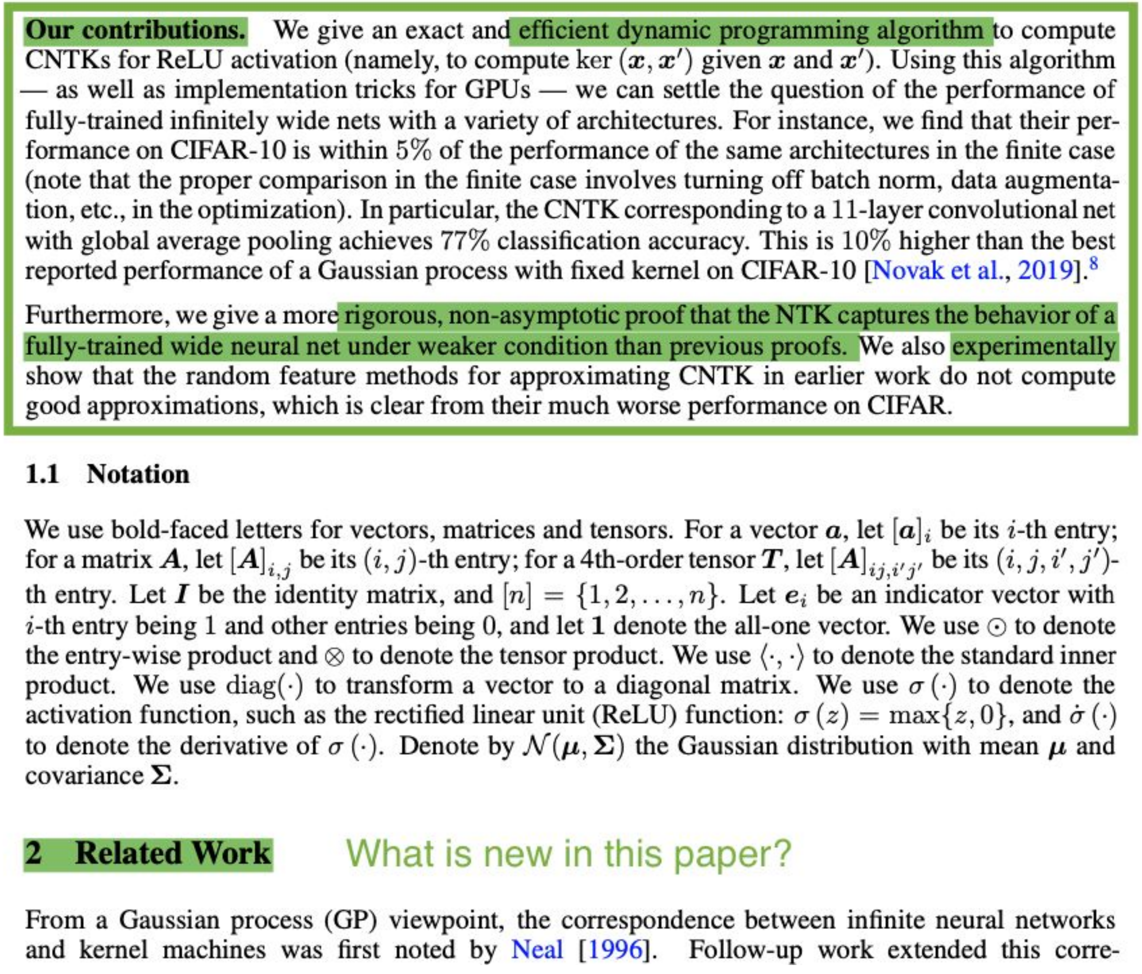
Abstract

How well does a classic deep net architecture like AlexNet or VGG19 classify on a standard dataset such as CIFAR-10 when its "width"— namely, number of channels in convolutional layers, and number of nodes in fully-connected internal layers — is allowed to increase to infinity? Such questions have come to the forefront in the quest to theoretically understand deep learning and its mysteries about optimization and generalization. They also connect deep learning to notions such as *Gaussian processes* and *kernels*. A recent paper [Jacot et al., 2018] introduced the *Neural Tangent Kernel (NTK)* which captures the behavior of fully-connected deep nets in the infinite width limit trained by gradient descent; this object was implicit in some other recent papers. An attraction of such ideas is that a pure kernel-based method is used to capture the power of a fully-trained deep net of infinite width.

The current paper gives the first efficient exact algorithm for computing the extension of NTK to convolutional neural nets, which we call *Convolutional NTK (CNTK)*, as well as an efficient GPU implementation of this algorithm. This results in a significant new benchmark for performance of a pure kernel-based method on CIFAR-10, being 10% higher than the methods reported in [Novak et al., 2019], and only 6% lower than the performance of the corresponding finite deep net architecture (once batch normalization etc. are turned off). Theoretically, we also give the first *non-asymptotic* proof showing that a fully-trained sufficiently wide net is indeed equivalent to the kernel regression predictor using NTK.

General Setup

This Paper shows



Our contributions. We give an exact and efficient dynamic programming algorithm to compute CNTKs for ReLU activation (namely, to compute $\mathrm{ker}(\boldsymbol{x}, \boldsymbol{x}')$ given $\boldsymbol{x}$ and $\boldsymbol{x}'$). Using this algorithm — as well as implementation tricks for GPUs — we can settle the question of the performance of fully-trained infinitely wide nets with a variety of architectures. For instance, we find that their performance on CIFAR-10 is within 5% of the performance of the same architectures in the finite case (note that the proper comparison in the finite case involves turning off batch norm, data augmentation, etc., in the optimization). In particular, the CNTK corresponding to a 11-layer convolutional net with global average pooling achieves 77% classification accuracy. This is 10% higher than the best reported performance of a Gaussian process with fixed kernel on CIFAR-10 [Novak et al., 2019].[8]

Furthermore, we give a more rigorous, non-asymptotic proof that the NTK captures the behavior of a fully-trained wide neural net under weaker condition than previous proofs. We also experimentally show that the random feature methods for approximating CNTK in earlier work do not compute good approximations, which is clear from their much worse performance on CIFAR.

1.1 Notation

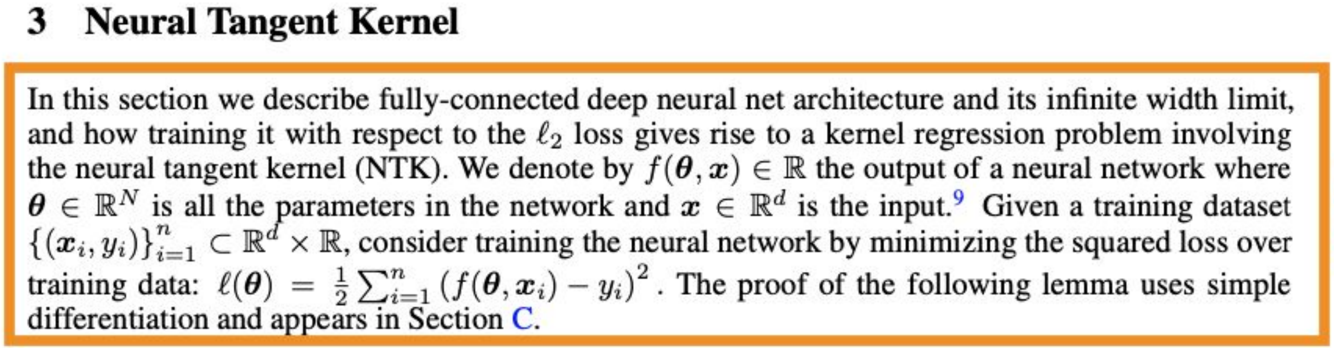
We use bold-faced letters for vectors, matrices and tensors. For a vector $\boldsymbol{a}$, let $[\boldsymbol{a}]_i$ be its i-th entry; for a matrix $\boldsymbol{A}$, let $[\boldsymbol{A}]_{i,j}$ be its (i,j)-th entry; for a 4th-order tensor $\boldsymbol{T}$, let $[\boldsymbol{A}]_{ij,i'j'}$ be its (i,j,i',j')-th entry. Let $\boldsymbol{I}$ be the identity matrix, and $[n] = \{1, 2, \ldots, n\}$. Let $\boldsymbol{e}_i$ be an indicator vector with i-th entry being 1 and other entries being 0, and let $\mathbf{1}$ denote the all-one vector. We use $\odot$ to denote the entry-wise product and $\otimes$ to denote the tensor product. We use $\langle \cdot, \cdot \rangle$ to denote the standard inner product. We use $\mathrm{diag}(\cdot)$ to transform a vector to a diagonal matrix. We use $\sigma(\cdot)$ to denote the activation function, such as the rectified linear unit (ReLU) function: $\sigma(z) = \max\{z, 0\}$, and $\dot{\sigma}(\cdot)$ to denote the derivative of $\sigma(\cdot)$. Denote by $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ the Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.

2 Related Work

From a Gaussian process (GP) viewpoint, the correspondence between infinite neural networks and kernel machines was first noted by Neal [1996]. Follow-up work extended this corre-

What exactly does this paper proof /. what do we reproduce

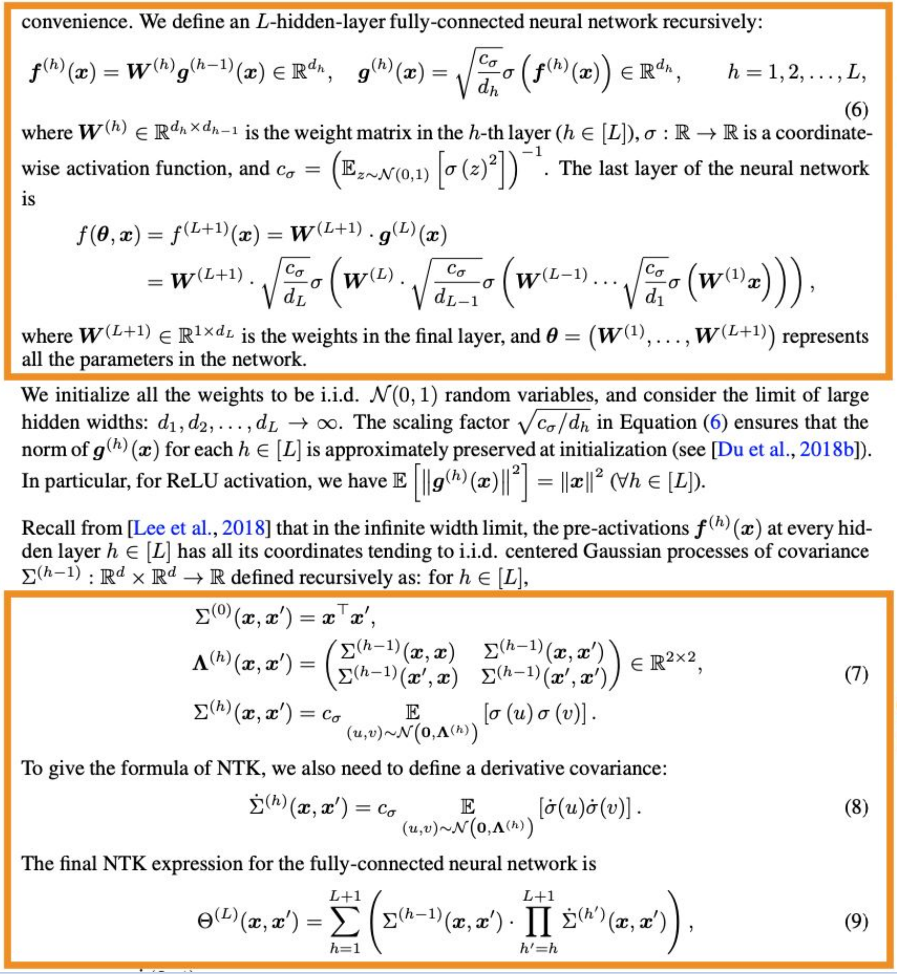
What is new in this paper?



3 Neural Tangent Kernel

In this section we describe fully-connected deep neural net architecture and its infinite width limit, and how training it with respect to the ℓ_2 loss gives rise to a kernel regression problem involving the neural tangent kernel (NTK). We denote by $f(\boldsymbol{\theta}, \boldsymbol{x}) \in \mathbb{R}$ the output of a neural network where $\boldsymbol{\theta} \in \mathbb{R}^N$ is all the parameters in the network and $\boldsymbol{x} \in \mathbb{R}^d$ is the input.[9] Given a training dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$, consider training the neural network by minimizing the squared loss over training data: $\ell(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^n \left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - y_i\right)^2$. The proof of the following lemma uses simple differentiation and appears in Section C.

General Setup



convenience. We define an L-hidden-layer fully-connected neural network recursively:

$$\boldsymbol{f}^{(h)}(\boldsymbol{x}) = \boldsymbol{W}^{(h)}\boldsymbol{g}^{(h-1)}(\boldsymbol{x}) \in \mathbb{R}^{d_h}, \quad \boldsymbol{g}^{(h)}(\boldsymbol{x}) = \sqrt{\frac{c_\sigma}{d_h}}\sigma\left(\boldsymbol{f}^{(h)}(\boldsymbol{x})\right) \in \mathbb{R}^{d_h}, \qquad h = 1, 2, \ldots, L, \tag{6}$$

where $\boldsymbol{W}^{(h)} \in \mathbb{R}^{d_h \times d_{h-1}}$ is the weight matrix in the h-th layer ($h \in [L]$), $\sigma : \mathbb{R} \to \mathbb{R}$ is a coordinate-wise activation function, and $c_\sigma = \left(\mathbb{E}_{z\sim\mathcal{N}(0,1)}\left[\sigma\left(z\right)^2\right]\right)^{-1}$. The last layer of the neural network is

$$\begin{aligned} f(\boldsymbol{\theta}, \boldsymbol{x}) &= f^{(L+1)}(\boldsymbol{x}) = \boldsymbol{W}^{(L+1)} \cdot \boldsymbol{g}^{(L)}(\boldsymbol{x}) \\ &= \boldsymbol{W}^{(L+1)} \cdot \sqrt{\frac{c_\sigma}{d_L}}\sigma\left(\boldsymbol{W}^{(L)} \cdot \sqrt{\frac{c_\sigma}{d_{L-1}}}\sigma\left(\boldsymbol{W}^{(L-1)} \cdots \sqrt{\frac{c_\sigma}{d_1}}\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)\right)\right), \end{aligned}$$

where $\boldsymbol{W}^{(L+1)} \in \mathbb{R}^{1\times d_L}$ is the weights in the final layer, and $\boldsymbol{\theta} = \left(\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L+1)}\right)$ represents all the parameters in the network.

We initialize all the weights to be i.i.d. $\mathcal{N}(0, 1)$ random variables, and consider the limit of large hidden widths: $d_1, d_2, \ldots, d_L \to \infty$. The scaling factor $\sqrt{c_\sigma/d_h}$ in Equation (6) ensures that the norm of $\boldsymbol{g}^{(h)}(\boldsymbol{x})$ for each $h \in [L]$ is approximately preserved at initialization (see [Du et al., 2018b]). In particular, for ReLU activation, we have $\mathbb{E}\left[\left\|\boldsymbol{g}^{(h)}(\boldsymbol{x})\right\|^2\right] = \|\boldsymbol{x}\|^2$ ($\forall h \in [L]$).

Recall from [Lee et al., 2018] that in the infinite width limit, the pre-activations $\boldsymbol{f}^{(h)}(\boldsymbol{x})$ at every hidden layer $h \in [L]$ has all its coordinates tending to i.i.d. centered Gaussian processes of covariance $\Sigma^{(h-1)} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ defined recursively as: for $h \in [L]$,

$$\begin{aligned} \Sigma^{(0)}(\boldsymbol{x}, \boldsymbol{x}') &= \boldsymbol{x}^\top \boldsymbol{x}', \\ \boldsymbol{\Lambda}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') &= \begin{pmatrix} \Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') \\ \Sigma^{(h-1)}(\boldsymbol{x}', \boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x}', \boldsymbol{x}') \end{pmatrix} \in \mathbb{R}^{2\times 2}, \\ \Sigma^{(h)}(\boldsymbol{x}, \boldsymbol{x}') &= c_\sigma \underset{(u,v)\sim\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}^{(h)}\right)}{\mathbb{E}}\left[\sigma\left(u\right)\sigma\left(v\right)\right]. \end{aligned} \tag{7}$$

To give the formula of NTK, we also need to define a derivative covariance:

$$\dot{\Sigma}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') = c_\sigma \underset{(u,v)\sim\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}^{(h)}\right)}{\mathbb{E}}\left[\dot{\sigma}(u)\dot{\sigma}(v)\right]. \tag{8}$$

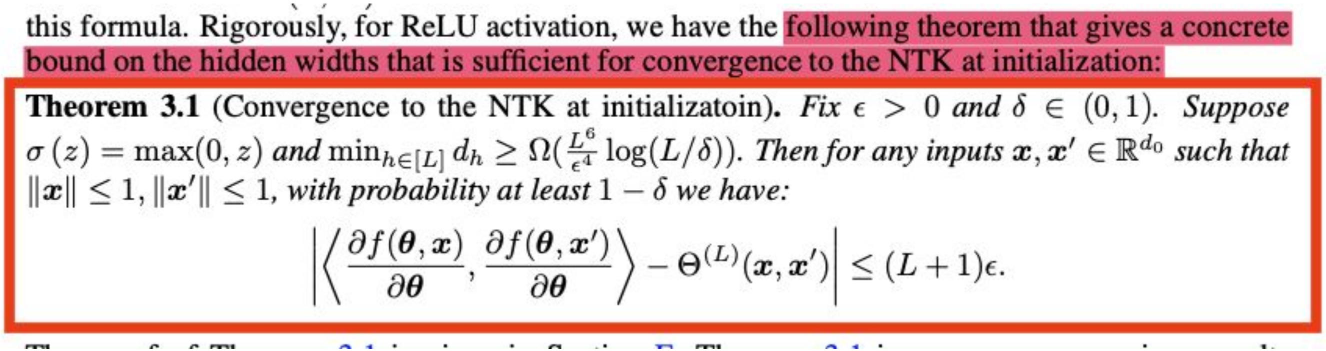
The final NTK expression for the fully-connected neural network is

$$\Theta^{(L)}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{h=1}^{L+1}\left(\Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') \cdot \prod_{h'=h}^{L+1} \dot{\Sigma}^{(h')}(\boldsymbol{x}, \boldsymbol{x}')\right), \tag{9}$$

Exact definition of the model.

Important for reproducing results

Corresponding NTK

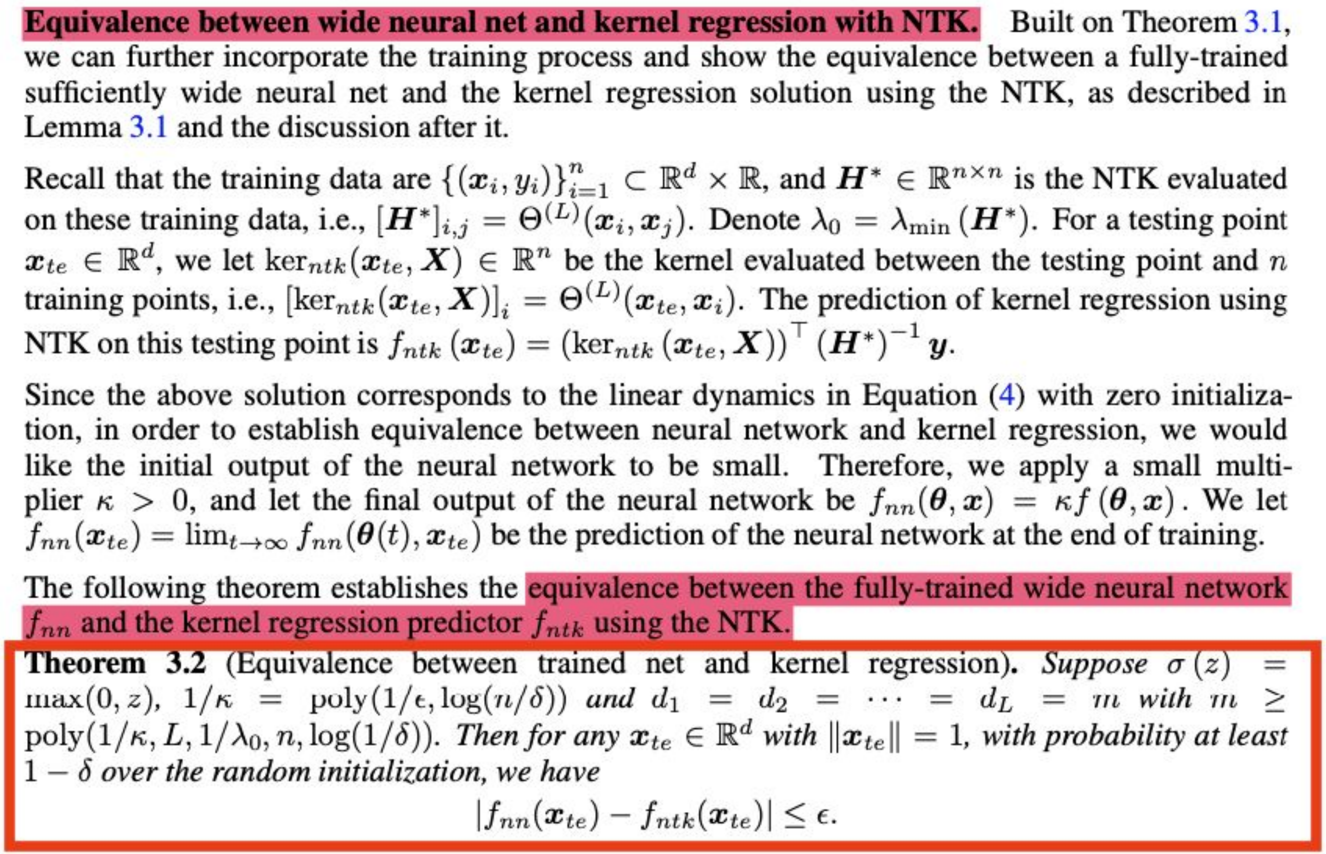


this formula. Rigorously, for ReLU activation, we have the following theorem that gives a concrete bound on the hidden widths that is sufficient for convergence to the NTK at initialization:

Theorem 3.1 (Convergence to the NTK at initializatoin)**.** *Fix* $\epsilon > 0$ *and* $\delta \in (0,1)$*. Suppose* $\sigma(z) = \max(0, z)$ *and* $\min_{h\in[L]} d_h \geq \Omega(\frac{L^6}{\epsilon^4}\log(L/\delta))$*. Then for any inputs* $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^{d_0}$ *such that* $\|\boldsymbol{x}\| \leq 1, \|\boldsymbol{x}'\| \leq 1$*, with probability at least* $1 - \delta$ *we have:*

$$\left|\left\langle \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{x})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}, \boldsymbol{x}')}{\partial \boldsymbol{\theta}} \right\rangle - \Theta^{(L)}(\boldsymbol{x}, \boldsymbol{x}')\right| \leq (L+1)\epsilon.$$

Theoretical Result 1



Equivalence between wide neural net and kernel regression with NTK. Built on Theorem 3.1, we can further incorporate the training process and show the equivalence between a fully-trained sufficiently wide neural net and the kernel regression solution using the NTK, as described in Lemma 3.1 and the discussion after it.

Recall that the training data are $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n} \subset \mathbb{R}^d \times \mathbb{R}$, and $\boldsymbol{H}^* \in \mathbb{R}^{n \times n}$ is the NTK evaluated on these training data, i.e., $[\boldsymbol{H}^*]_{i,j} = \Theta^{(L)}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Denote $\lambda_0 = \lambda_{\min}(\boldsymbol{H}^*)$. For a testing point $\boldsymbol{x}_{te} \in \mathbb{R}^d$, we let $\ker_{ntk}(\boldsymbol{x}_{te}, \boldsymbol{X}) \in \mathbb{R}^n$ be the kernel evaluated between the testing point and n training points, i.e., $[\ker_{ntk}(\boldsymbol{x}_{te}, \boldsymbol{X})]_i = \Theta^{(L)}(\boldsymbol{x}_{te}, \boldsymbol{x}_i)$. The prediction of kernel regression using NTK on this testing point is $f_{ntk}(\boldsymbol{x}_{te}) = (\ker_{ntk}(\boldsymbol{x}_{te}, \boldsymbol{X}))^\top (\boldsymbol{H}^*)^{-1} \boldsymbol{y}$.

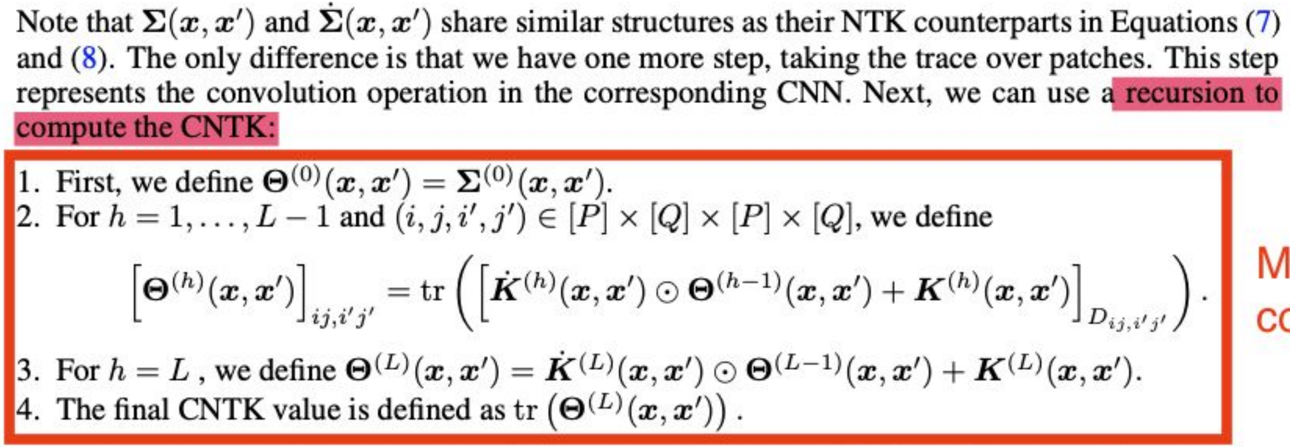
Since the above solution corresponds to the linear dynamics in Equation (4) with zero initialization, in order to establish equivalence between neural network and kernel regression, we would like the initial output of the neural network to be small. Therefore, we apply a small multiplier $\kappa > 0$, and let the final output of the neural network be $f_{nn}(\boldsymbol{\theta}, \boldsymbol{x}) = \kappa f(\boldsymbol{\theta}, \boldsymbol{x})$. We let $f_{nn}(\boldsymbol{x}_{te}) = \lim_{t \to \infty} f_{nn}(\boldsymbol{\theta}(t), \boldsymbol{x}_{te})$ be the prediction of the neural network at the end of training.

The following theorem establishes the equivalence between the fully-trained wide neural network f_{nn} and the kernel regression predictor f_{ntk} using the NTK.

Theorem 3.2 (Equivalence between trained net and kernel regression). *Suppose* $\sigma(z) = \max(0, z)$, $1/\kappa = \text{poly}(1/\epsilon, \log(n/\delta))$ *and* $d_1 = d_2 = \cdots = d_L = m$ *with* $m \geq \text{poly}(1/\kappa, L, 1/\lambda_0, n, \log(1/\delta))$. *Then for any* $\boldsymbol{x}_{te} \in \mathbb{R}^d$ *with* $\|\boldsymbol{x}_{te}\| = 1$, *with probability at least* $1 - \delta$ *over the random initialization, we have*

$$|f_{nn}(\boldsymbol{x}_{te}) - f_{ntk}(\boldsymbol{x}_{te})| \leq \epsilon.$$

Theoretical Result 2



Note that $\boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{x}')$ and $\dot{\boldsymbol{\Sigma}}(\boldsymbol{x}, \boldsymbol{x}')$ share similar structures as their NTK counterparts in Equations (7) and (8). The only difference is that we have one more step, taking the trace over patches. This step represents the convolution operation in the corresponding CNN. Next, we can use a recursion to compute the CNTK:

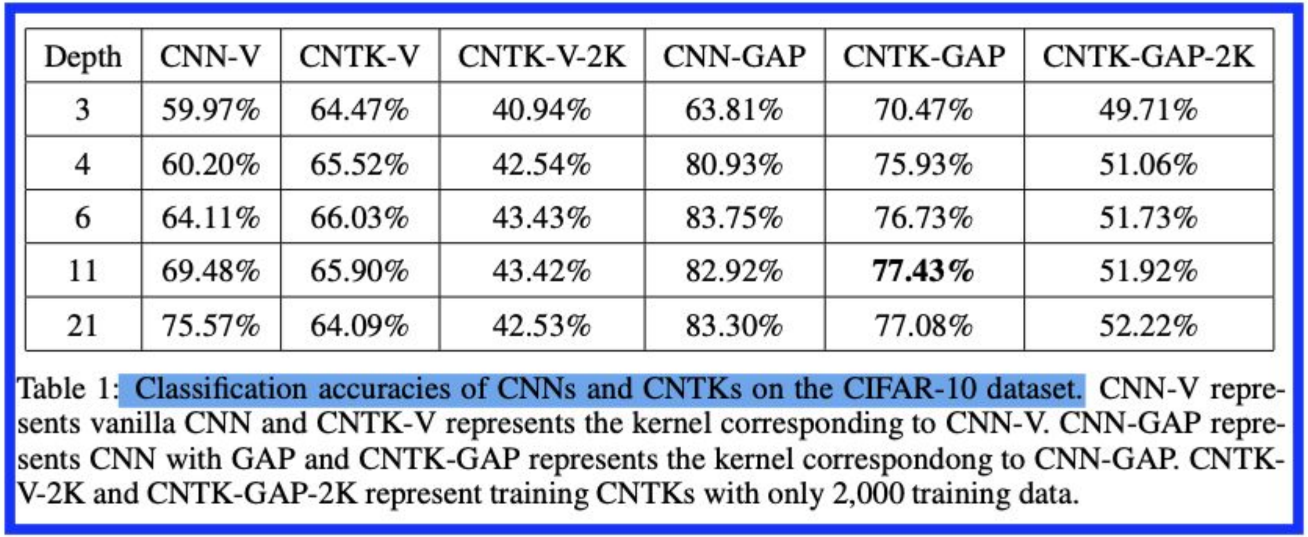
1. First, we define $\boldsymbol{\Theta}^{(0)}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\Sigma}^{(0)}(\boldsymbol{x}, \boldsymbol{x}')$.
2. For $h = 1, \ldots, L-1$ and $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, we define

$$\left[\boldsymbol{\Theta}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \operatorname{tr}\left(\left[\dot{\boldsymbol{K}}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') \odot \boldsymbol{\Theta}^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') + \boldsymbol{K}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{D_{ij,i'j'}}\right).$$

3. For $h = L$, we define $\boldsymbol{\Theta}^{(L)}(\boldsymbol{x}, \boldsymbol{x}') = \dot{\boldsymbol{K}}^{(L)}(\boldsymbol{x}, \boldsymbol{x}') \odot \boldsymbol{\Theta}^{(L-1)}(\boldsymbol{x}, \boldsymbol{x}') + \boldsymbol{K}^{(L)}(\boldsymbol{x}, \boldsymbol{x}')$.
4. The final CNTK value is defined as $\operatorname{tr}\left(\boldsymbol{\Theta}^{(L)}(\boldsymbol{x}, \boldsymbol{x}')\right)$.

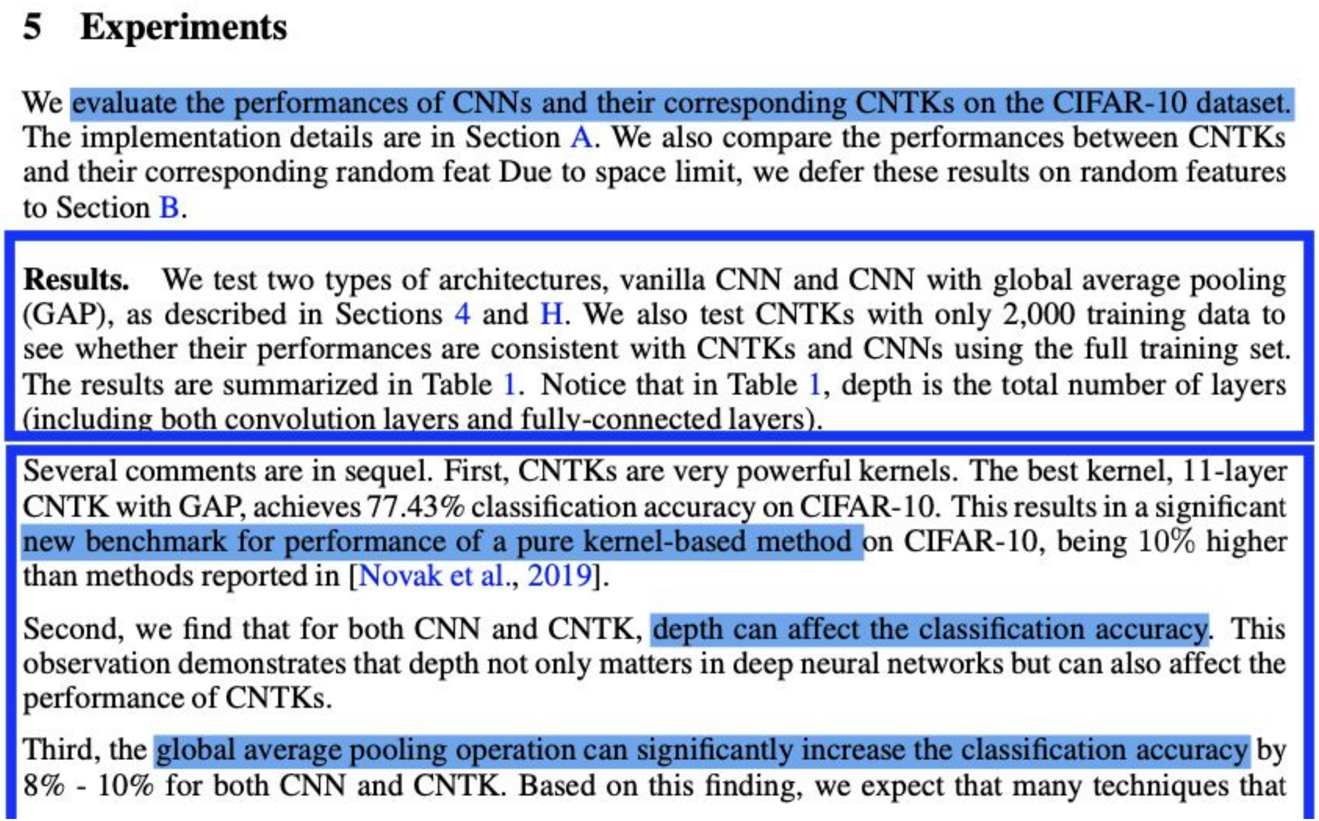
Main Algorithm to compute CNTK

Summery of Empirical Results



Depth	CNN-V	CNTK-V	CNTK-V-2K	CNN-GAP	CNTK-GAP	CNTK-GAP-2K
3	59.97%	64.47%	40.94%	63.81%	70.47%	49.71%
4	60.20%	65.52%	42.54%	80.93%	75.93%	51.06%
6	64.11%	66.03%	43.43%	83.75%	76.73%	51.73%
11	69.48%	65.90%	43.42%	82.92%	**77.43%**	51.92%
21	75.57%	64.09%	42.53%	83.30%	77.08%	52.22%

Table 1: Classification accuracies of CNNs and CNTKs on the CIFAR-10 dataset. CNN-V represents vanilla CNN and CNTK-V represents the kernel corresponding to CNN-V. CNN-GAP represents CNN with GAP and CNTK-GAP represents the kernel correspondong to CNN-GAP. CNTK-V-2K and CNTK-GAP-2K represent training CNTKs with only 2,000 training data.



5 Experiments

We evaluate the performances of CNNs and their corresponding CNTKs on the CIFAR-10 dataset. The implementation details are in Section A. We also compare the performances between CNTKs and their corresponding random feat Due to space limit, we defer these results on random features to Section B.

Results. We test two types of architectures, vanilla CNN and CNN with global average pooling (GAP), as described in Sections 4 and H. We also test CNTKs with only 2,000 training data to see whether their performances are consistent with CNTKs and CNNs using the full training set. The results are summarized in Table 1. Notice that in Table 1, depth is the total number of layers (including both convolution layers and fully-connected layers).

Experiment Setup

Several comments are in sequel. First, CNTKs are very powerful kernels. The best kernel, 11-layer CNTK with GAP, achieves 77.43% classification accuracy on CIFAR-10. This results in a significant new benchmark for performance of a pure kernel-based method on CIFAR-10, being 10% higher than methods reported in [Novak et al., 2019].

Second, we find that for both CNN and CNTK, depth can affect the classification accuracy. This observation demonstrates that depth not only matters in deep neural networks but can also affect the performance of CNTKs.

Third, the global average pooling operation can significantly increase the classification accuracy by 8% - 10% for both CNN and CNTK. Based on this finding, we expect that many techniques that

Analysis of Results

Possible Extensions / Further Experiments

- How does data pre-processing influence the performance of CNN and CNTK?
- Analyze the influence of depth for CNN and CNTK
- CNTK with vector output and global average pooling: What helps performance improve with depth?

Practical Structure

Structure

Groups of 2 students - 2 research papers per group

1. understand the main theoretical ideas of the paper and reproduce the empirical findings.
2. extend on the empirical observations with further experiments.

Timeline

- First or second week of semester: Introduction Meeting
- Mid June: Reproducibility report submission
- End of semester: Empirical extensions + functional code submission
- End of semester: Final presentations

Weekly:

- ~10 min update presentation from every student
- Office Hours for further questions

Grading

- Report on reproducibility (40%)
- Report on extensions (20%)
- Final group presentation (40%).

Prerequisite

- Machine learning (IN2064)
- Introduction to deep learning (IN2346)
- Statistical foundations of learning (IN2378) - option

Survey: https://forms.gle/so6b5GL9PCBofiXb7

(Link also on website of

Theoretical Foundations of Artificial Intelligence)

THIS DOES NOT REPLACE THE MATCHING SYSTEM