Kernels, Data and Physics
or
How can (statistical) physics tools help the DL practitioner

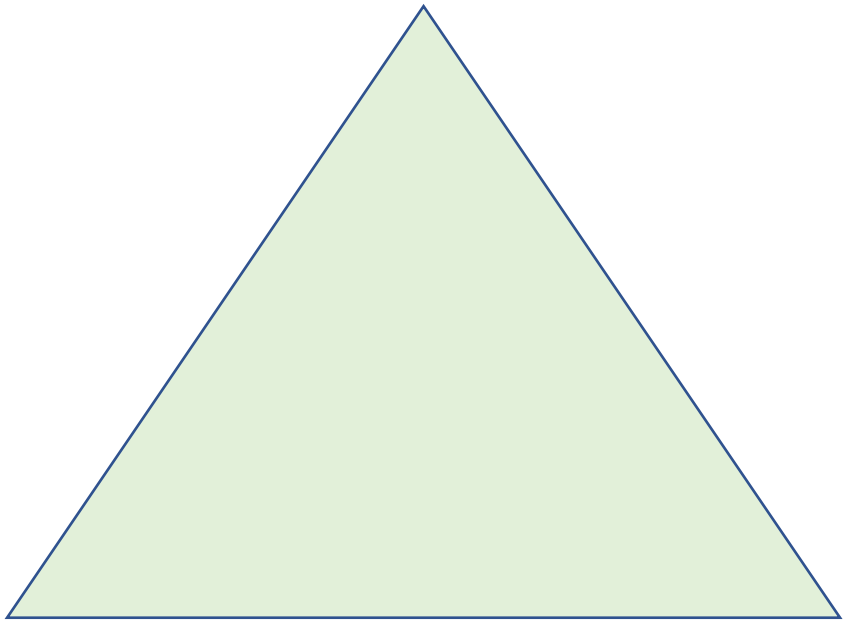
Julia Kempe, CDS & Courant Institute, NYU

Les Houches 2022

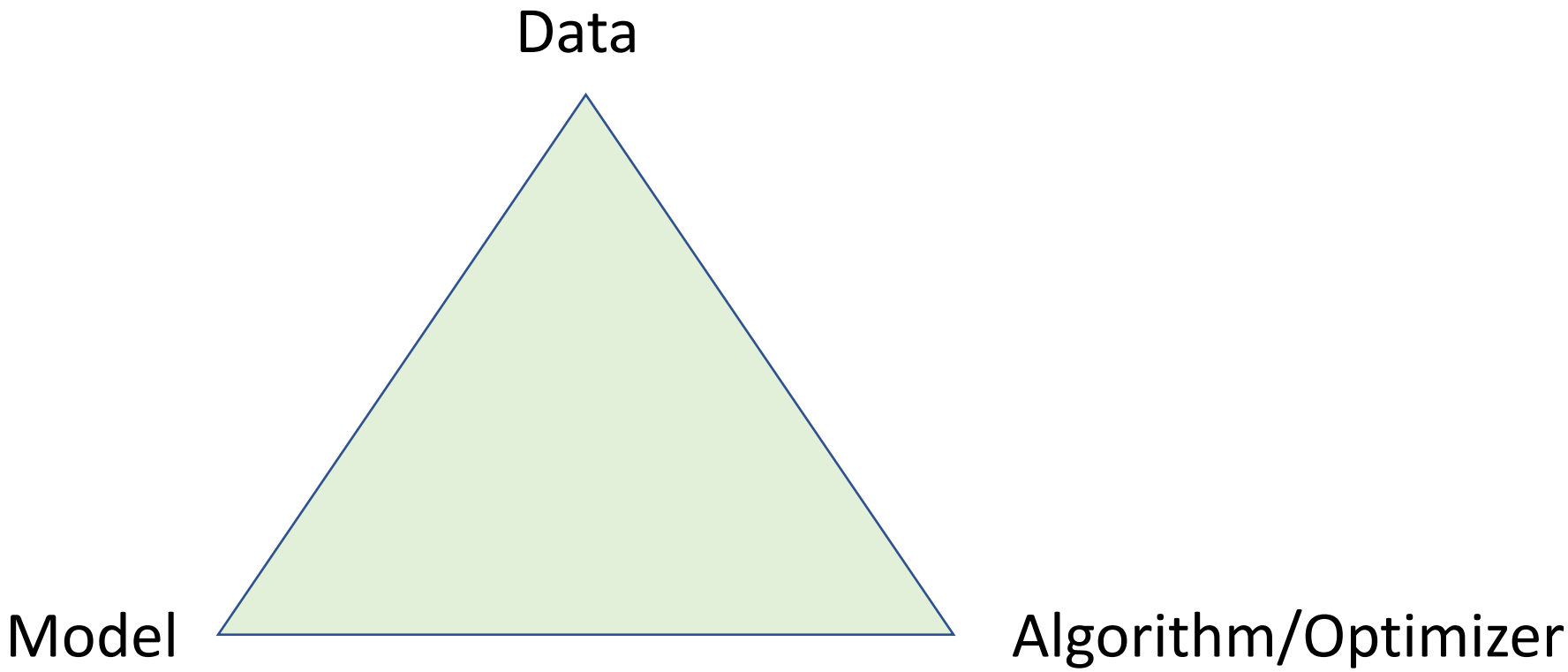
Ingredients of these lectures

- Symmetries – invariances – equivariances
- Sample complexity – computational complexity
- NTK (of course)
- Physical intuitions
- Several “practical” problems: inductive bias, data distillation, adversarial robustness, pruning, architecture search, physics-inspired NN , ….

The triangle…



What makes DL work…?

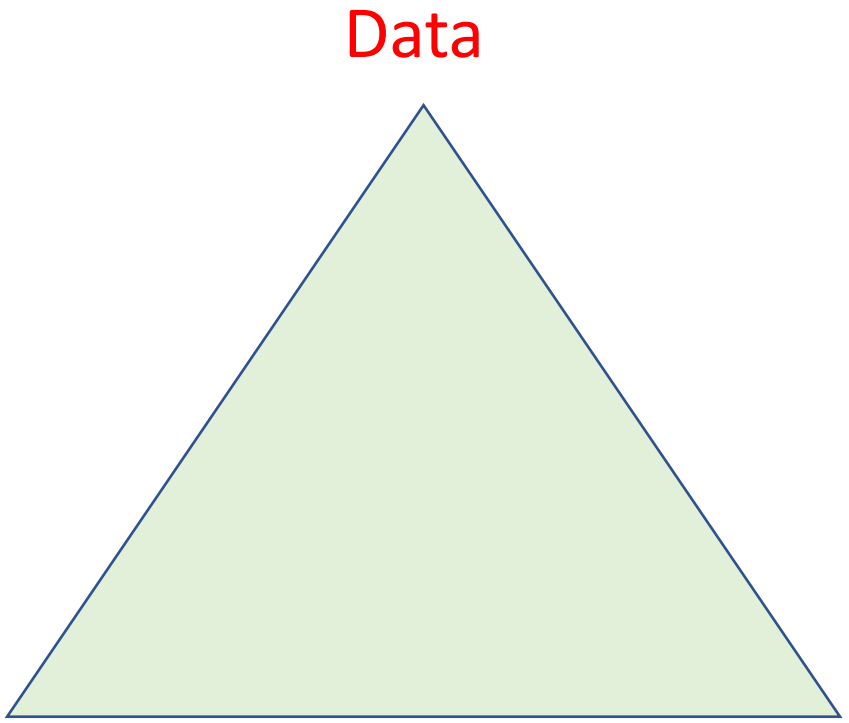


Courtesy of Lenka Z.

What makes DL work…?

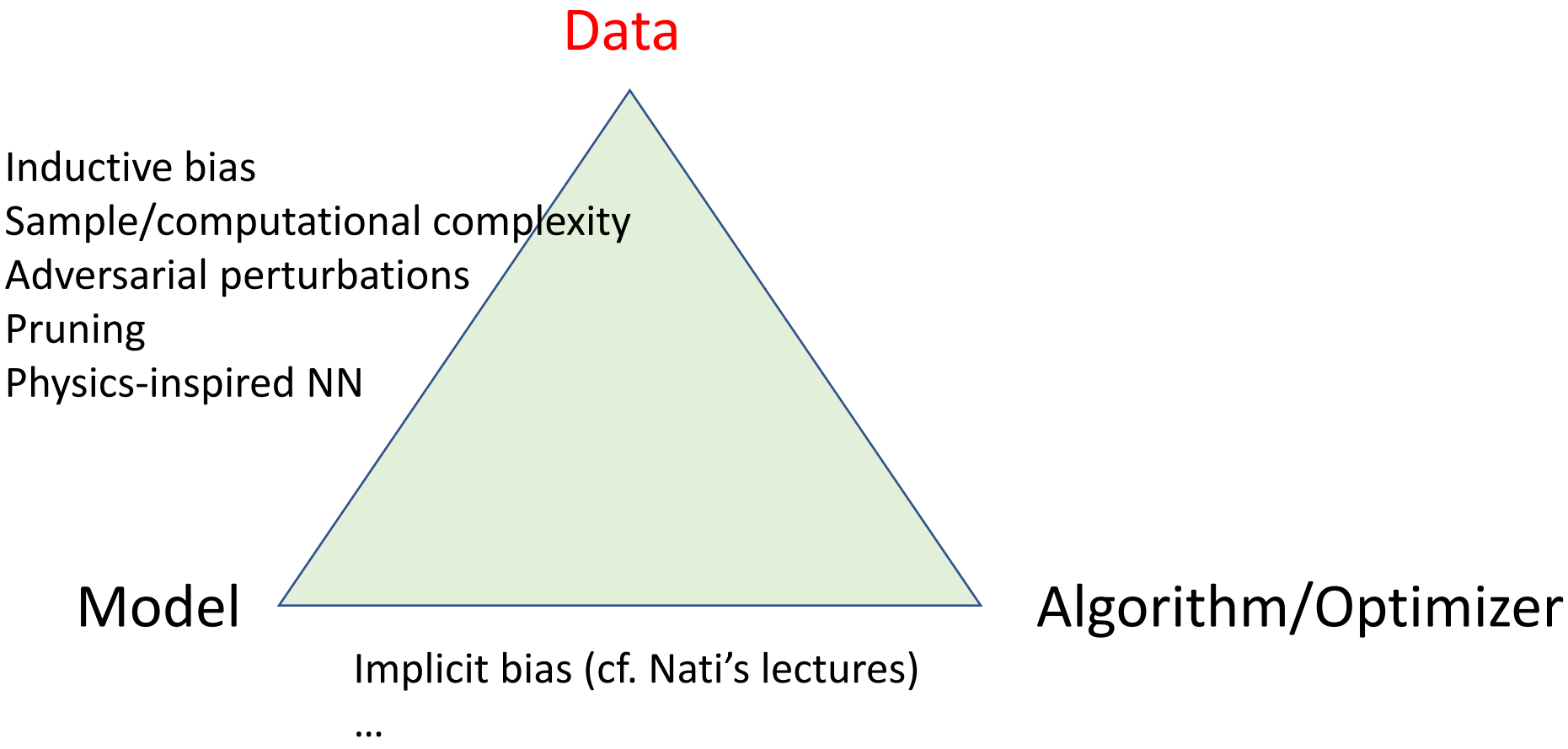
- Model: Architecture, parameters, …
- Algorithm/Optimizer: GD/SGD, Adam, Regularization, …
- Data?
  - Structure
  - Dimensionality
  - Invariances
  - Provenance (e.g. obeys some physical laws)

What makes DL work…?



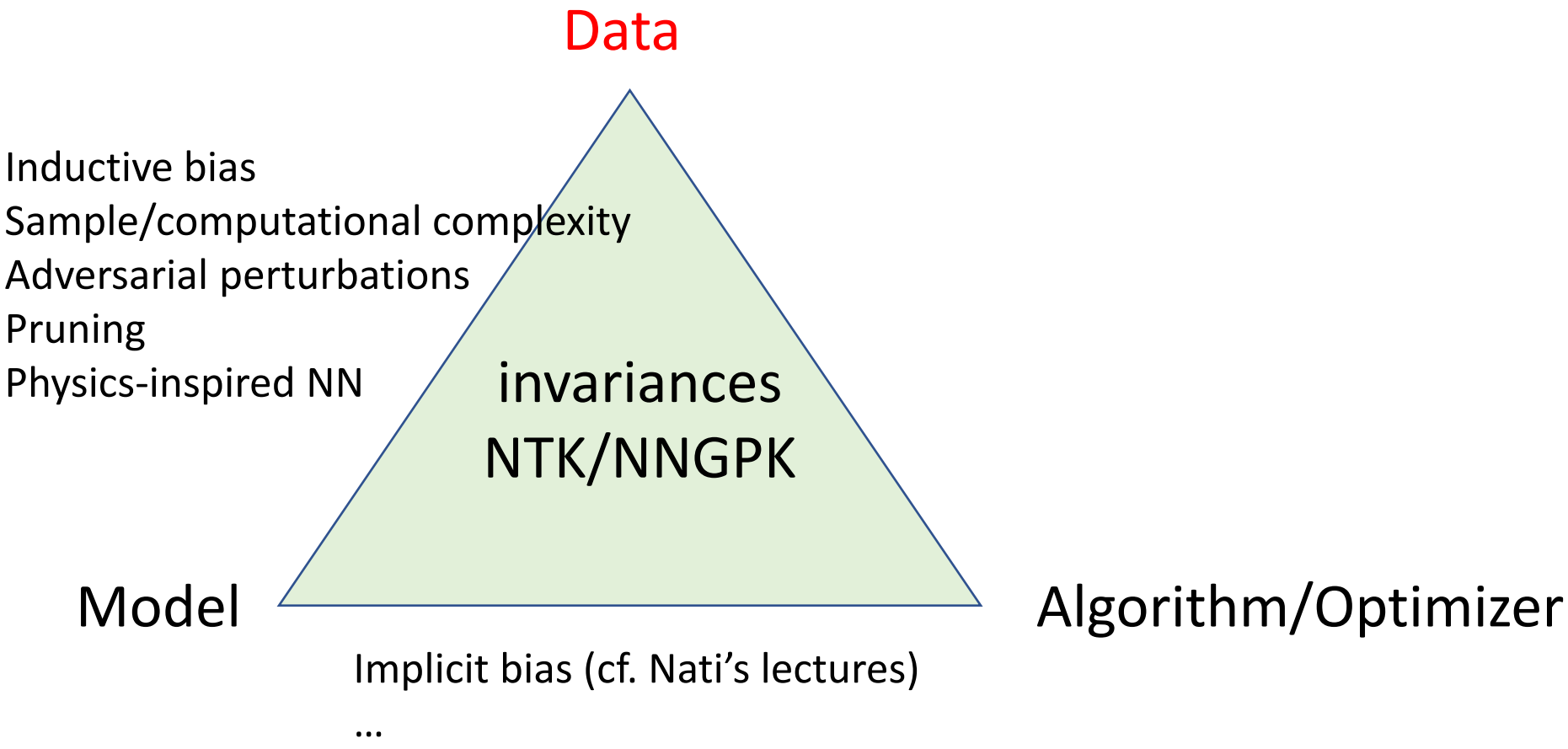
Courtesy of Lenka Z.

What makes DL work…?



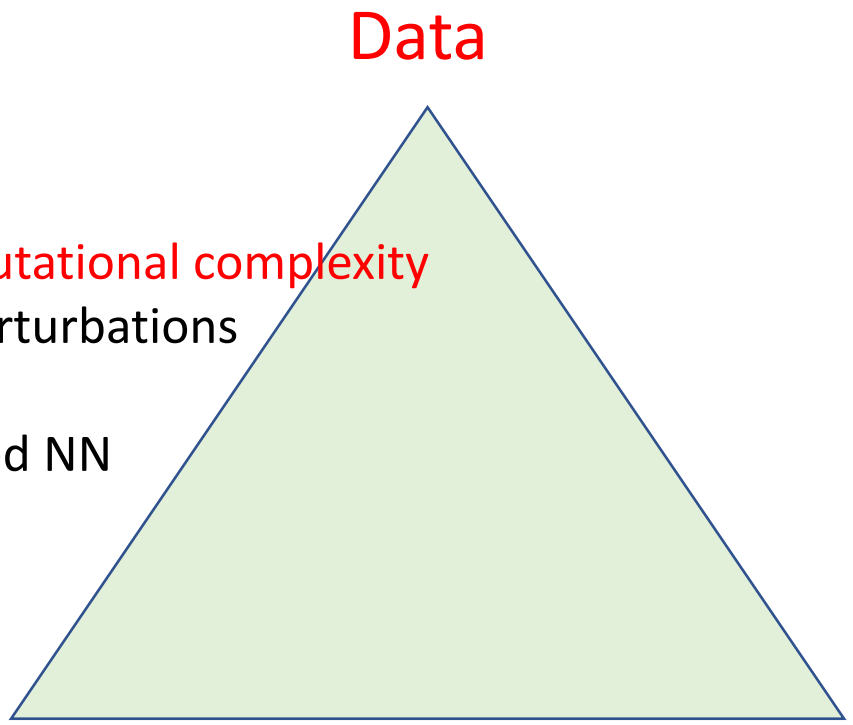
Courtesy of Lenka Z.

What makes DL work…?



Courtesy of Lenka Z.

To start: Inductive bias and sample/computational complexity



Courtesy of Lenka Z.

To start: Inductive bias and sample/computational complexity

- For many tasks (especially vision) convolutional (CONV) architectures perform significantly better than their fully-connected (FC) counterparts (at least given the same amount of training data)

- Practitioners explain this at an intuitive level as better “inductive bias”:
  1. CONV match the underlying structure of the data better
  2. Models with fewer total number of parameters (weight sharing) generalize better

- Make this rigorous?

“Rigorous” Inductive Bias

- Find a task that requires far more training samples on FC than on CONV
  - Hard to do because a large enough FC can simulate a CONV
  - So: not just “expressivity” but combination of training algorithm + architecture
  - [Li, Zhang, Arora ICLR’21] : binary task in d-dimensions such that
    - CONV needs O(1) samples
    - FC needs $\Omega(d^2)$ samples
    - Gives beautiful intuitions based on equivariance

- Find a task that can be efficiently solved by CONV while provably hard for FC with GD
  - [Malach, Shalev-Schwartz ICLR’21] : hidden “consecutive” pattern
    - Poly-size CONV needs poly steps
    - Poly-size FC needs superpoly steps

Unpacking TCS proofs

The main result in this section shows that gradient-descent can learn k-patterns when training convolutional networks for $poly(2^k, n)$ iterations, and when the network has $poly(2^k, n)$ neurons:

Theorem 4. *Assume we uniformly initialize* $W^{(0)} \sim \{\pm 1/k\}^{q \times k}$, $b_i = 1/k - 1$ *and* $\mathbf{u}^{(0,j)} = 0$ *for every* j. *Assume the activation* σ *satisfies* $|\sigma| \leq c, |\sigma'| \leq 1$, *for some constant* c. *Fix some* $\delta > 0$, *some* k*-pattern* f *and some distribution* $\mathcal{D}$ *over* $\mathcal{X}$. *Then, if* $q > 2^{k+3} \log(2^k/\delta)$, *with probability at least* $1 - \delta$ *over the initialization, when training a convolutional network* $h_{\mathbf{u},W,\mathbf{b}}$ *using gradient descent with* $\eta = \frac{\sqrt{n}}{\sqrt{q}T}$ *we have:*

$$\frac{1}{T}\sum_{t=1}^{T} L_{f,\mathcal{D}}(h_{\mathbf{u}^{(t)},W^{(t)},b}) \leq \frac{2cn^2k^2 2^k}{q} + \frac{2(2^k k)^2}{\sqrt{qn}} + \frac{c^2 n^{1.5}\sqrt{q}}{T}$$

Before we prove the theorem, observe that the above immediately implies that when $k = O(\log n)$, gradient-descent can **efficiently** learn to solve the k-pattern problem, when training a CNN:

Unpacking TCS proofs

The main result in this section shows that gradient-descent can learn k-patterns when training convolutional networks for $poly(2^k, n)$ iterations, and when the network has $poly(2^k, n)$ neurons:

Theorem 4. *Assume we uniformly initialize* $W^{(0)} \sim \{\pm 1/k\}^{q \times k}$, $b_i = 1/k - 1$ *and* $\mathbf{u}^{(0,j)} = 0$ *for every* j. *Assume the activation* σ *satisfies* $|\sigma| \leq c, |\sigma'| \leq 1$, *for some constant* c. *Fix some* $\delta > 0$, *some* k*-pattern* f *and some distribution* $\mathcal{D}$ *over* $\mathcal{X}$. *Then, if* $q > 2^{k+3} \log(2^k/\delta)$, *with probability at least* $1 - \delta$ *over the initialization, when training a convolutional network* $h_{\mathbf{u},W,\mathbf{b}}$ *using gradient descent with* $\eta = \frac{\sqrt{n}}{\sqrt{q}T}$ *we have:*

$$\frac{1}{T}\sum_{t=1}^{T} L_{f,\mathcal{D}}(h_{\mathbf{u}^{(t)},W^{(t)},b}) \leq \frac{2cn^2k^2 2^k}{q} + \frac{2(2^k k)^2}{\sqrt{qn}} + \frac{c^2 n^{1.5}\sqrt{q}}{T}$$

Before we prove the theorem, observe that the above immediately implies that when $k = O(\log n)$, gradient-descent can **efficiently** learn to solve the k-pattern problem, when training a CNN:

Unpacking TCS proofs

Truth table!

The main result in this section shows that gradient-descent can learn k-patterns when training convolutional networks for $poly(2^k, n)$ iterations, and when the network has $poly(2^k, n)$ neurons:

Theorem 4. *Assume we uniformly initialize* $W^{(0)} \sim \{\pm 1/k\}^{q \times k}$, $b_i = 1/k - 1$ *and* $\mathbf{u}^{(0,j)} = 0$ *for every* j. *Assume the activation* σ *satisfies* $|\sigma| \leq c, |\sigma'| \leq 1$, *for some constant* c. *Fix some* $\delta > 0$, *some* k*-pattern* f *and some distribution* $\mathcal{D}$ *over* $\mathcal{X}$. *Then, if* $q > 2^{k+3} \log(2^k/\delta)$, *with probability at least* $1 - \delta$ *over the initialization, when training a convolutional network* $h_{\mathbf{u},W,\mathbf{b}}$ *using gradient descent with* $\eta = \frac{\sqrt{n}}{\sqrt{q}T}$ *we have:*

Coupon collector!

$$\frac{1}{T}\sum_{t=1}^{T} L_{f,\mathcal{D}}(h_{\mathbf{u}^{(t)},W^{(t)},b}) \leq \frac{2cn^2k^22^k}{q} + \frac{2(2^k k)^2}{\sqrt{qn}} + \frac{c^2 n^{1.5}\sqrt{q}}{T}$$

Before we prove the theorem, observe that the above immediately implies that when $k = O(\log n)$, gradient-descent can **efficiently** learn to solve the k-pattern problem, when training a CNN:

Statistical Physics of Machine Learning?

- In this school we heard a lot about theoretical developments for machine learning, based on methods from statistical physics (infinite width limits, replica, spin glasses, MP and AMP, quenched disorder, ….)

- Surprising progress has been made to explain some phenomena (e.g. NTK)

- What are the “big” (and “small”) questions that we could hope to solve with our tools?

What we would like to understand better

- What are some of the open problems that could benefit from the statistical physics lens?

- Some more obvious ones:
  - What is the role played by over-parametrization and how do we explain/harness some of its manifestations:
    - With gradient flow, gradient descent, SGD
    - Generalization with overparametrization
    - Double descent
    - Implicit bias
    - Role of Depth? Width?

And for the practitioner

- Can we create new algorithms (for NNs) using NTK insights?
- … or at least inspired by NTK?
- … using closed form expressions?
- … and the corresponding libraries?

But what about some more “practical” questions?

- Learning:
  - How do Neural Networks learn? What do they learn first/last? How fast?
  - What features do they extract?
  - What properties of the data (in practice) give rise to successful learning?
  - How are the first phases of learning different from the last ? (early stage learning is linear ... first stage of learning is chaotic… low frequencies are learned first…)

- Efficient learning? Can we reduce complexity either pre or post inference?
  - Model distillation (“teacher/student models”)
  - Dataset distillation*
  - Few-shot learning
  - Pruning of networks*
    - Lottery ticket hypothesis

But what about some more “practical” questions?

- Adversarial Attacks/Robustness:
  - Why are overparametrized neural nets vulnerable to small “adversarial” attacks even when they are stable to small random noise?
  - How can we create “robust” networks/models?
  - Other adversarial interventions (e.g. “poisoning attacks”)

- Catastrophic forgetting and continual learning
  - Why do neural nets “forget” during continual learning?
  - What methods work to prevent this? How can we create non-forgetting NNs?

But what about some more “practical” questions?

- And more:
  - Matrix completion [Radharishnan et al. PNAS’22]
  - Learning “small data” tasks [Arora et al. ICLR’20]
  - Recommendation networks [Sachdeva et al. ICML-ws‘22]
  - Understanding and quantifying ensembling/different local minima
  - …

In these lectures

- We will “tour” some of these more “practical” problems (overview)
- We will try to identify where our statistical physics methods or thinking can been applied, especially recent NTK methods
- We will stop and ask if we increase our understanding of DL, even though some problems might seem specialized or even irrelevant
- I will diverge to areas that might be interesting for those with a more physics minded background (sampling/computational separations, symmetries, “physical bias” for neural nets, PINNs, symbolic regression), even if at times I lose the connection to *statistical* physics.
- I will ask more questions than provide answers (*Epistemic status: not confident enough to bet against someone who’s likely to understand this stuff.)*

Why do we (the “NTK consumers”) like NTK?

- Kernel Ridge Regression is simple!
- Convenient analytical expression for evolution during training.

- Can take the derivative with respect to x (the data)!

- And as a bonus, NTK describes the infinitely wide limit of NNs. As such, certain insights/algorithms/techniques might transfer.
- Good libraries now exist (e.g. JAX based Neural Tangent Library)

And yes, we know there are limitations of the kernel-regime (for instance in *understanding the power of depth* [Bietti, Bach’21] or *classifying high-dimensional Gaussians* [Refinetti et al. ‘21] or *finding a sparse signal amidst high-variance noise* [Karp et. al ‘21]) – but we take what we get and run as far as we can with it, while waiting for the next set of “beyond kernel” tools.

NTK approach for “practical” problems

- A common approach:
  - Start with a problem for NN (intractable, hard/impossible to solve, …)
  - Find an underlying NTK formulation
  - Solve for the NTK setting
  - Take a deep breath, make a leap of faith
  - Transfer to the NN setting and hope it works
- And sometimes it does! Data Distillation, Poisoning Attacks, Pruning, NAS…
- And sometimes we learn something or reinforce what we already suspected: spectral bias, acquisition of adversarial robustness during training etc. …

Interlude: A practitioner's summary of NTKs

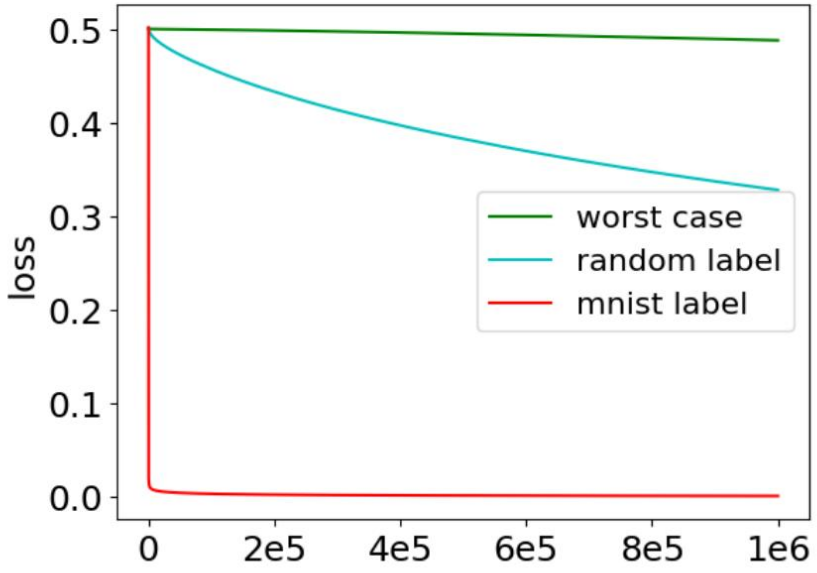
- NTK Theory [Jacot et al. ‘18]: a class of infinitely wide (or ultra-wide) neural networks trained by gradient descent $\Leftrightarrow$ kernel regression with a fixed kernel (NTK)

- Blackboard:

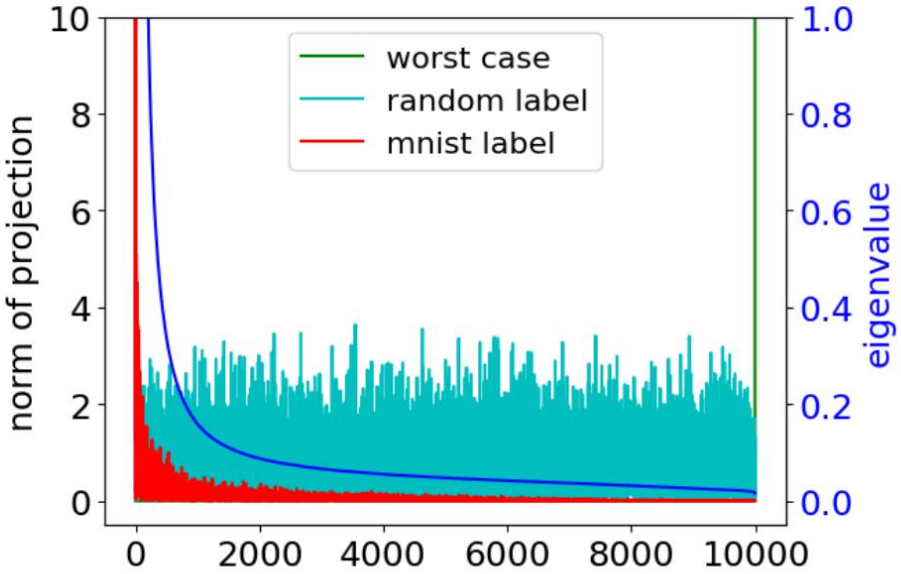
A first “application”: training speed for random labels

- Components of y on larger eigenvectors converge faster

MNIST (2 Classes)



Convergence Rate



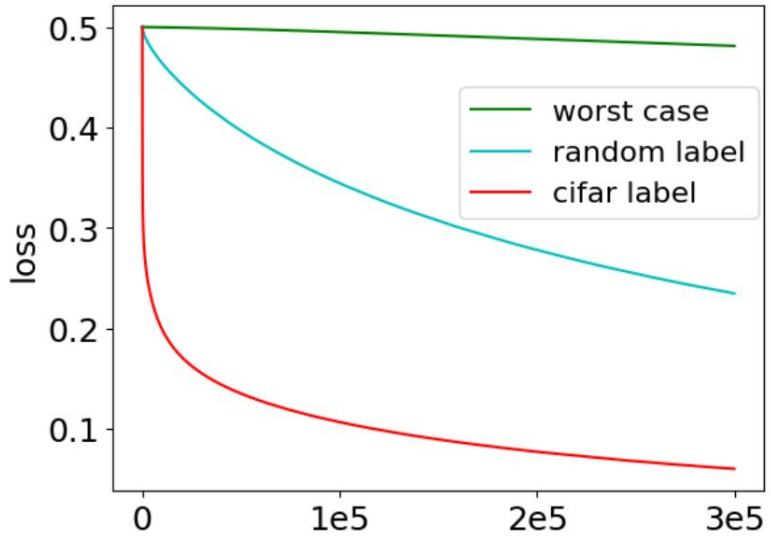
Projections

Arora et al. ‘19

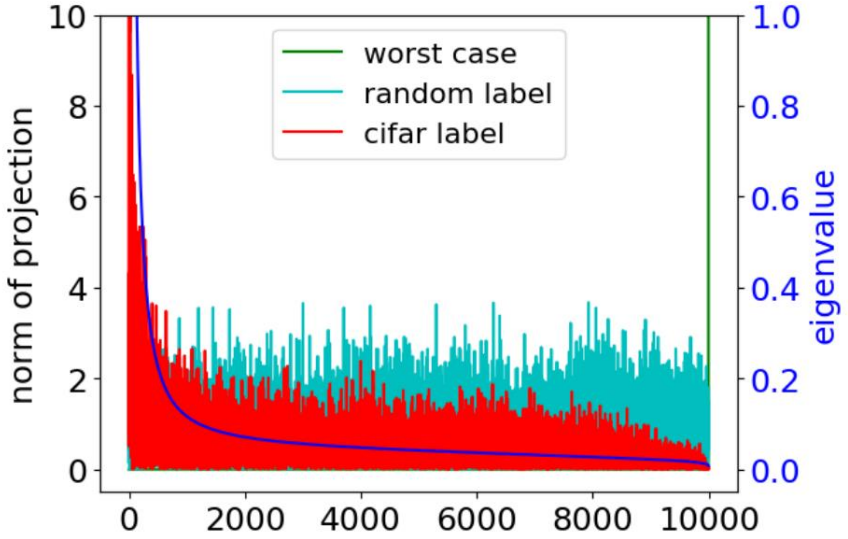
A first “application”: training speed for random labels

- Components of y on larger eigenvectors converge faster

CIFAR-10 (2 Classes)



Convergence Rate



Projections

Arora et al. ‘19

Analytical expressions:

- FC kernels

[Jacot et al. '18]:

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

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

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

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

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

Analytical expressions: CONV kernels [Arora et al. '19

- For $\alpha = 1, \ldots, C^{(0)}$, $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, define

$$\boldsymbol{K}^{(0)}_{(\alpha)}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}_{(\alpha)} \otimes \boldsymbol{x}'_{(\alpha)} \text{ and } \left[\boldsymbol{\Sigma}^{(0)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \sum_{\alpha=1}^{C^{(0)}} \operatorname{tr}\left(\left[\boldsymbol{K}^{(0)}_{(\alpha)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{\mathcal{D}_{ij,i'j'}}\right).$$

- For $h \in [L]$,
  - For $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, define

$$\boldsymbol{\Lambda}^{(h)}_{ij,i'j'}(\boldsymbol{x}, \boldsymbol{x}') = \begin{pmatrix} \left[\boldsymbol{\Sigma}^{(h-1)}(\boldsymbol{x}, \boldsymbol{x})\right]_{ij,ij} & \left[\boldsymbol{\Sigma}^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} \\ \left[\boldsymbol{\Sigma}^{(h-1)}(\boldsymbol{x}', \boldsymbol{x})\right]_{i'j',ij} & \left[\boldsymbol{\Sigma}^{(h-1)}(\boldsymbol{x}', \boldsymbol{x}')\right]_{i'j',i'j'} \end{pmatrix} \in \mathbb{R}^{2\times 2}.$$

  - Define $\boldsymbol{K}^{(h)}(\boldsymbol{x}, \boldsymbol{x}'), \dot{\boldsymbol{K}}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^{P\times Q\times P\times Q}$, for $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$

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

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

  - Define $\boldsymbol{\Sigma}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^{P\times Q\times P\times Q}$, for $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$

$$\left[\boldsymbol{\Sigma}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \frac{c_\sigma}{q^2} \operatorname{tr}\left(\left[\boldsymbol{K}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{D_{ij,i'j'}}\right).$$

On spectral bias

- Cao et al. ‘21:
- Data uniform on d-sphere

NTK vs NN

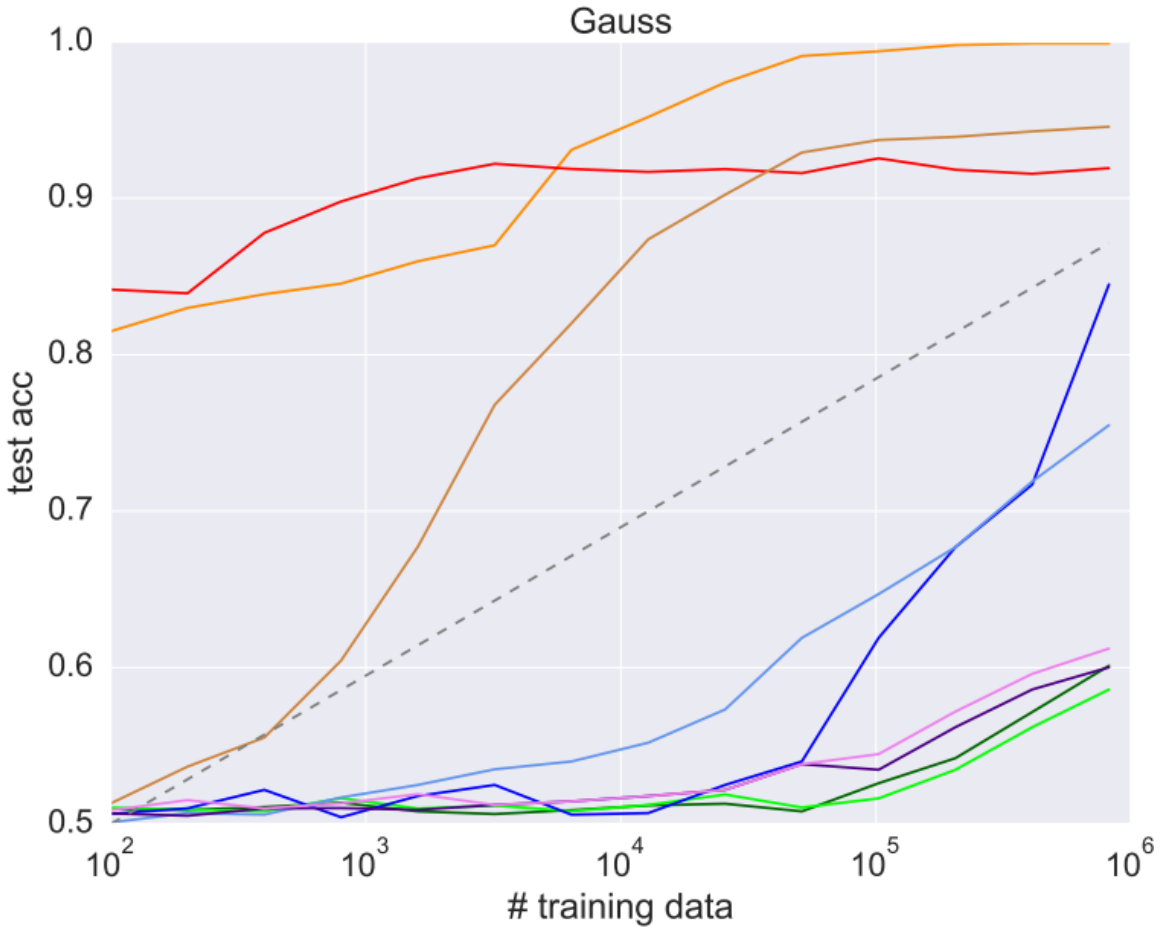
- 2019 – now: many separation results and comparison (cf. seminar Mathieu Wyart and posters here), also [Lee et al. NeurIPS'20: Finite versus Infinite Neural Networks: An Empirical study]
  - Computing kernels scales superquadratically with |data|
  - Tricks: ensembling across data batches
- Lots of things NTK can't capture
- For instance, NTK can't have equivariance
- But NTKs do well on small data sets [Arora et al. '20, Du et al. '19]

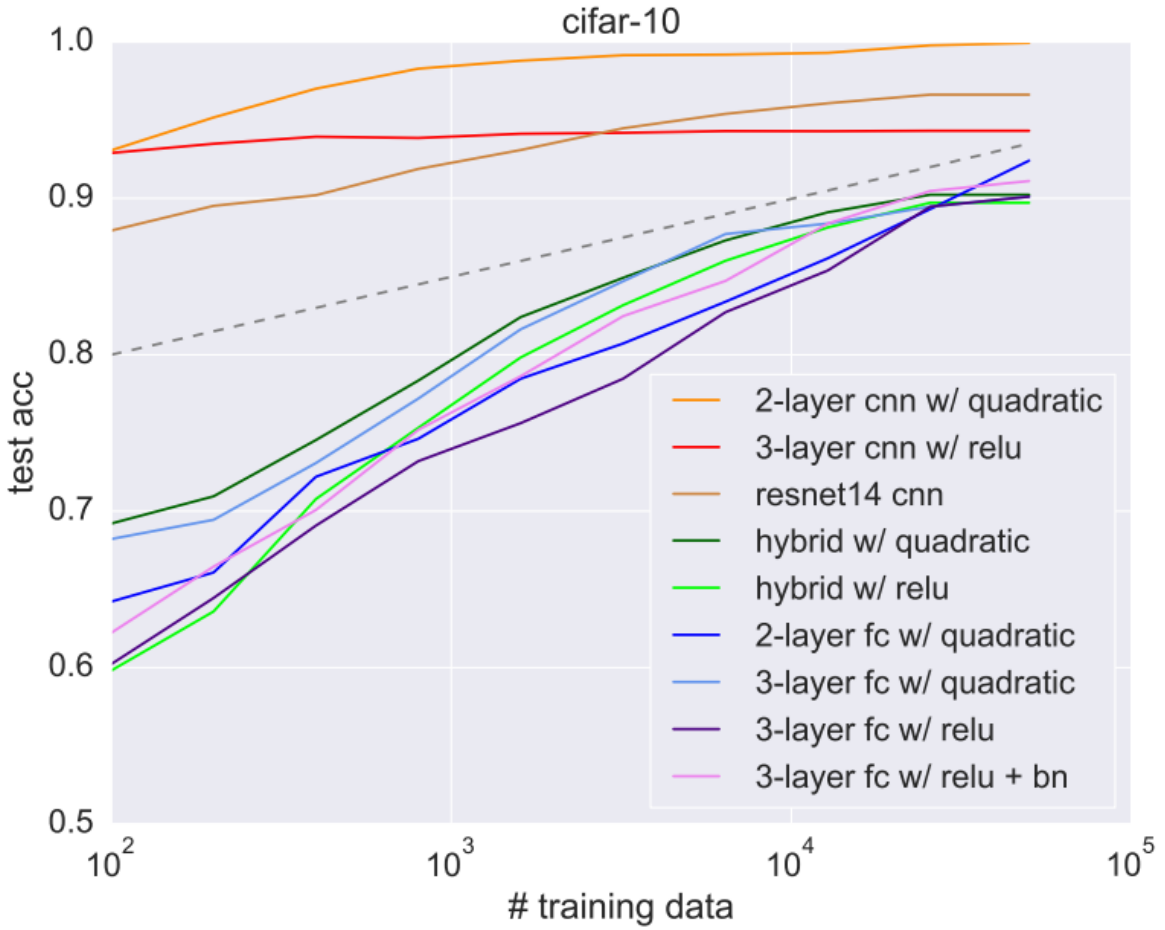
Kernels, Data and Physics
or
How can (statistical) physics tools help the DL practitioner

Julia Kempe, CDS & Courant Institute, NYU

Les Houches 2022

FC vs CONV separation [Li, Zhang, Arora ICLR'21]



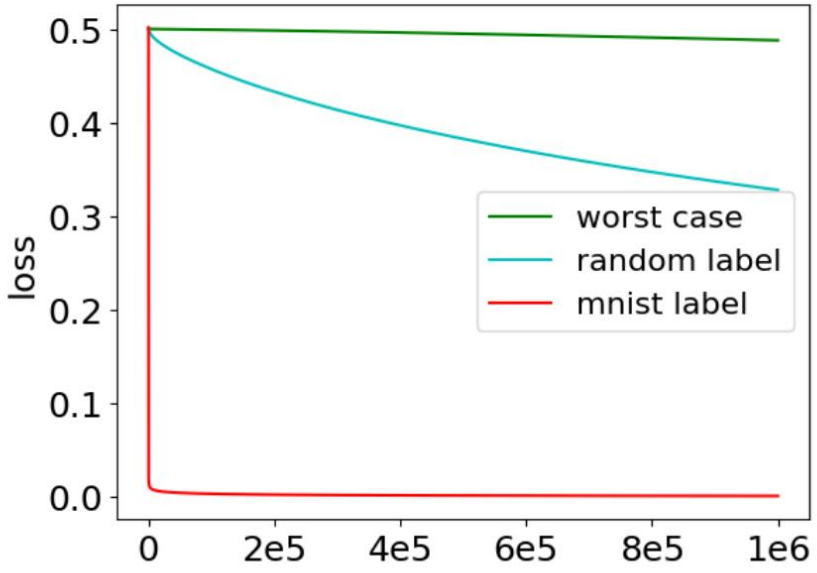


Input: 3 x 32 x 32 RGB images Output: sign (l2-norm of first channel – l2-norm of second channel)

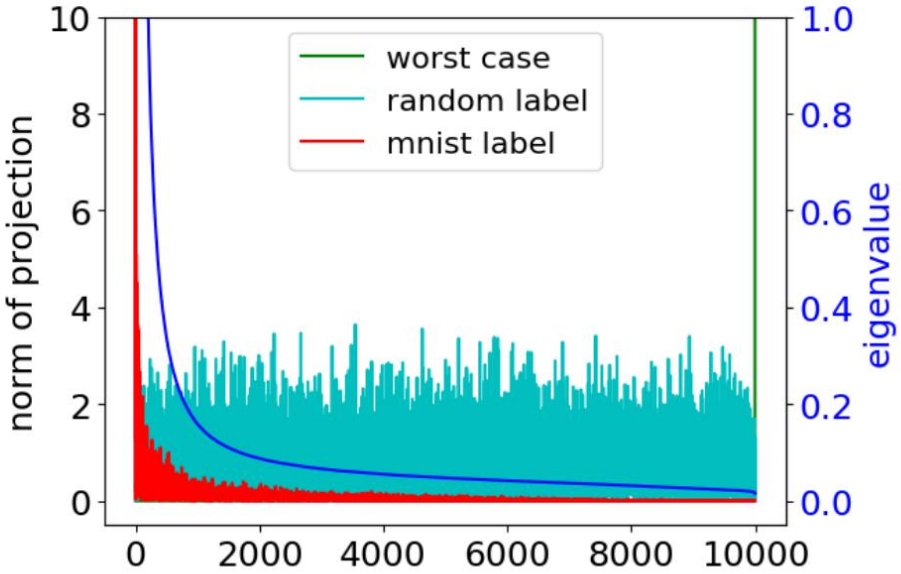
A first “application”: training speed for random labels

- Components of y on larger eigenvectors converge faster

MNIST (2 Classes)



Convergence Rate



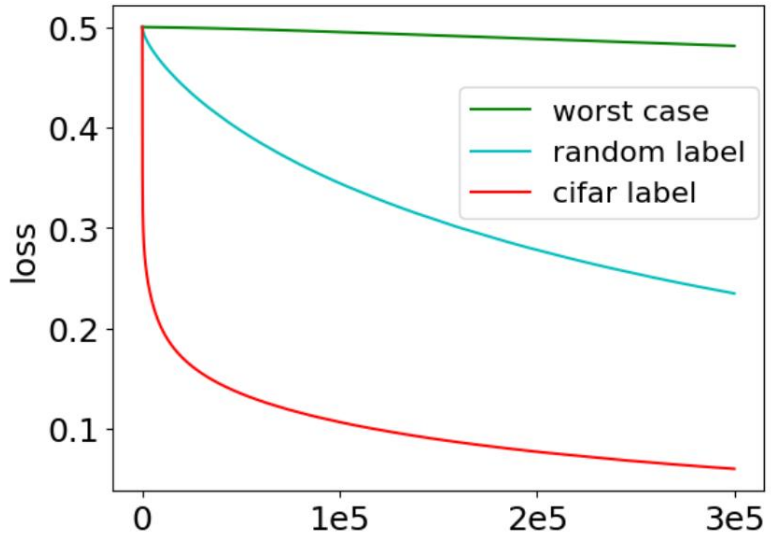
Projections

Arora et al. ‘19

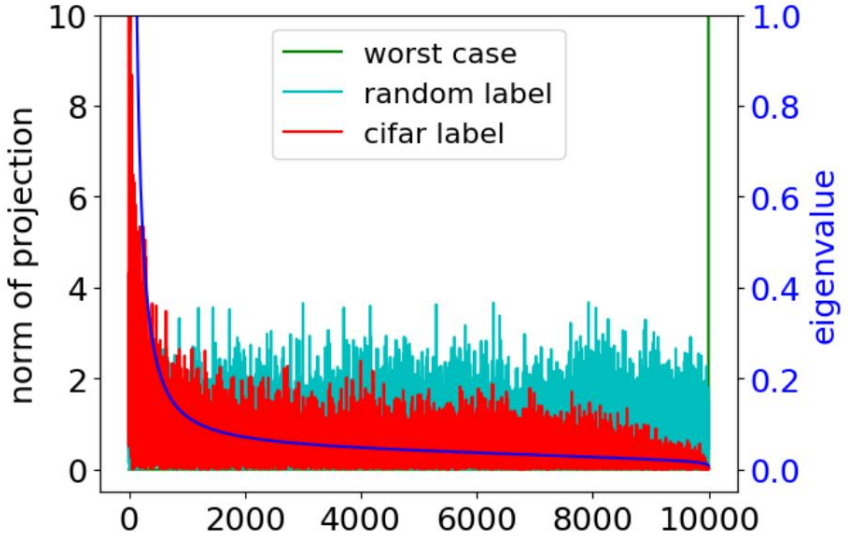
A first “application”: training speed for random labels

- Components of y on larger eigenvectors converge faster

CIFAR-10 (2 Classes)



Convergence Rate



Projections

Arora et al. ‘19

Spectral Bias ("NN learn functions of increasing complexity")

- Components of y on larger eigenvectors converge faster

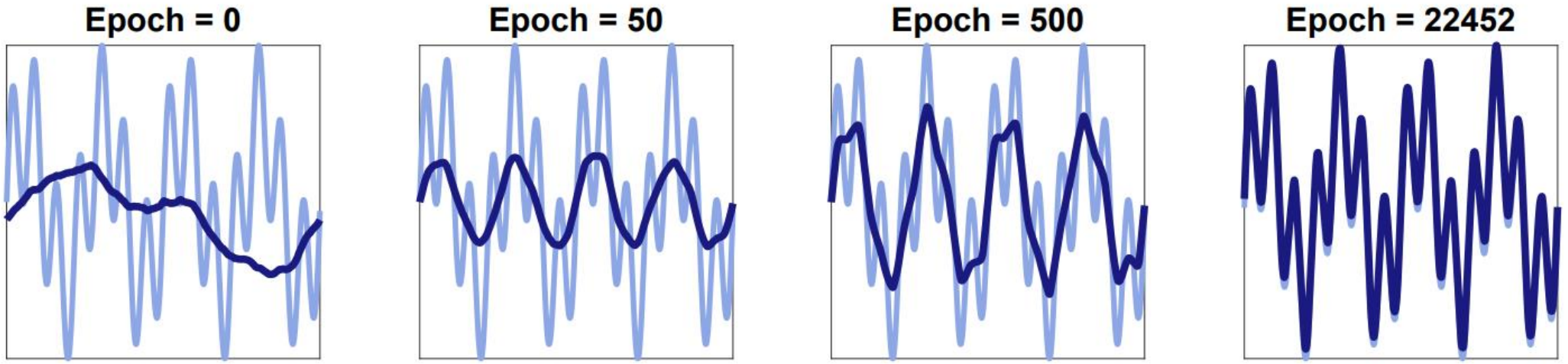


Figure 2: Network prediction (dark blue) for a superposition of two sine waves with frequencies $k = 4, 14$ (light blue). The network fits the lower frequency component of the function after 50 epochs, while fitting the full function only after ∼22K epochs.

Basri et al. NeurIPS'19

On spectral bias

- Bach (2017): studied two-layer ReLU networks by relating it to kernel methods, and proposed a harmonic decomposition for the functions in the reproducing kernel Hilbert
- Bietti and Mairal (2019) studied the eigenvalue decay of integrating operator defined by the neural tangent kernel on unit sphere by using spherical harmonics.
- Vempala and Wilmes (2019) calculated the eigenvalues of neural tangent kernel corresponding to two-layer neural networks with sigmoid activation function.
- Basri et al. (2019): considered the case of training the first layer parameters of a two-layer networks with bias terms.
- Yang and Salman (2019) studied the the eigenvalues of integral operator with respect to the NTK on Boolean cube by Fourier analysis.
- Bordelon et al. (2020) gave a spectral analysis on the generalization error of NTK-based kernel ridge regression.
- Basri et al. (2020) studied the convergence of full training residual with a focus on one-dimensional, non-uniformly distributed data.
- Cao et al. (2021) : characterization of NTK spectra for inputs over uniform sphere

NTK approach for “practical” problems

- A common approach:
  - Start with a problem for NN (intractable, hard/impossible to solve, …)
  - Find an underlying NTK formulation
  - Solve for the NTK setting
  - Take a deep breath, make a leap of faith
  - Transfer to the NN setting and hope it works

- And sometimes it does! Data Distillation, Poisoning Attacks, Pruning, NAS…

Parenthesis: Why the NTK and not any other kernel?

- (empirical) NTK corresponds to first order (in w) evolution of a NN
- NTK is *architecture-specific* (FC, Conv, …)
- Extend to deep architectures (compositional kernels)
- Some evidence that NN is linear in early stages of learning [Hu et al.: *The surprising simplicity of the Early-Time Learning Dynamics of Neural Networks* NeurIPS'20]
- Some evidence that NTK is a good approximation in later phases of learning [Fort et al.'20]
- And, of course, it describes the infinite-width limit of NNs (with "NTK" initialization)

- And lastly: Because, often, it works !

A first case in point: Dataset Distillation with NTK

- [Nguyen et al. ICLR'21, NeurIPS'21] developed the Kernel Inducing Point (KIP) algorithm for Dataset Distillation based on NTK
- Currently state of the art in dataset distillation

Dataset Distillation (DD) Background

- DD is a significant reduction in the sample size by creating synthetic data s.th. ML algorithms learn as efficiently as on full data
- Term “distillation” appears first in [Hinton, Vinyals, Dean: Distilling the Knowledge in a Neural Network NIPS’15] as “knowledge distillation”: construct a simple model based on a complex one
- Dataset Distillation first proposed in [Wang, Zhu, Torralba and Efros ‘18]
  - “distill” 60,000 MNIST images into 10 (one per class) such that simple NN trained on it still generalizes well

Dataset Distillation (DD) Background

- Why do we care about small synthetic datasets?
  - Scientific question: how much data is encoded in a training set
  - Since the synthetic dataset differs from “natural” data we also understand something about generalization
  - Akin to dimension reduction (“MNIST has low intrinsic dimension” [Pope et al. ICLR’21])
  - Can speed up search for Lottery Tickets (making them actually practical), help with pruning, continual learning, privacy preservation etc.

- Excursion: Low dimensionality of common data

THE INTRINSIC DIMENSION OF IMAGES AND ITS IMPACT ON LEARNING

[Pope et al. ICLR'21]

Phillip Pope[1], **Chen Zhu**[1], **Ahmed Abdelkader**[2], **Micah Goldblum**[1], **Tom Goldstein**[1]
[1]Department of Computer Science, University of Maryland, College Park
[2]Oden Institute for Computational Engineering and Sciences, University of Texas at Austin
{pepope,chenzhu}@umd.edu, akader@utexas.edu, {goldblum,tomg}@umd.edu

ABSTRACT

It is widely believed that natural image data exhibits low-dimensional structure despite the high dimensionality of conventional pixel representations. This idea underlies a common intuition for the remarkable success of deep learning in computer vision. In this work, we apply dimension estimation tools to popular datasets and investigate the role of low-dimensional structure in deep learning. We find that common natural image datasets indeed have very low intrinsic dimension relative to the high number of pixels in the images. Additionally, we find that low dimensional datasets are easier for neural networks to learn, and models solving these tasks generalize better from training to test data. Along the way, we develop a technique for validating our dimension estimation tools on synthetic data generated by GANs allowing us to actively manipulate the intrinsic dimension by controlling the image generation process. Code for our experiments may be found here.

[Pope et al. ICLR'21]

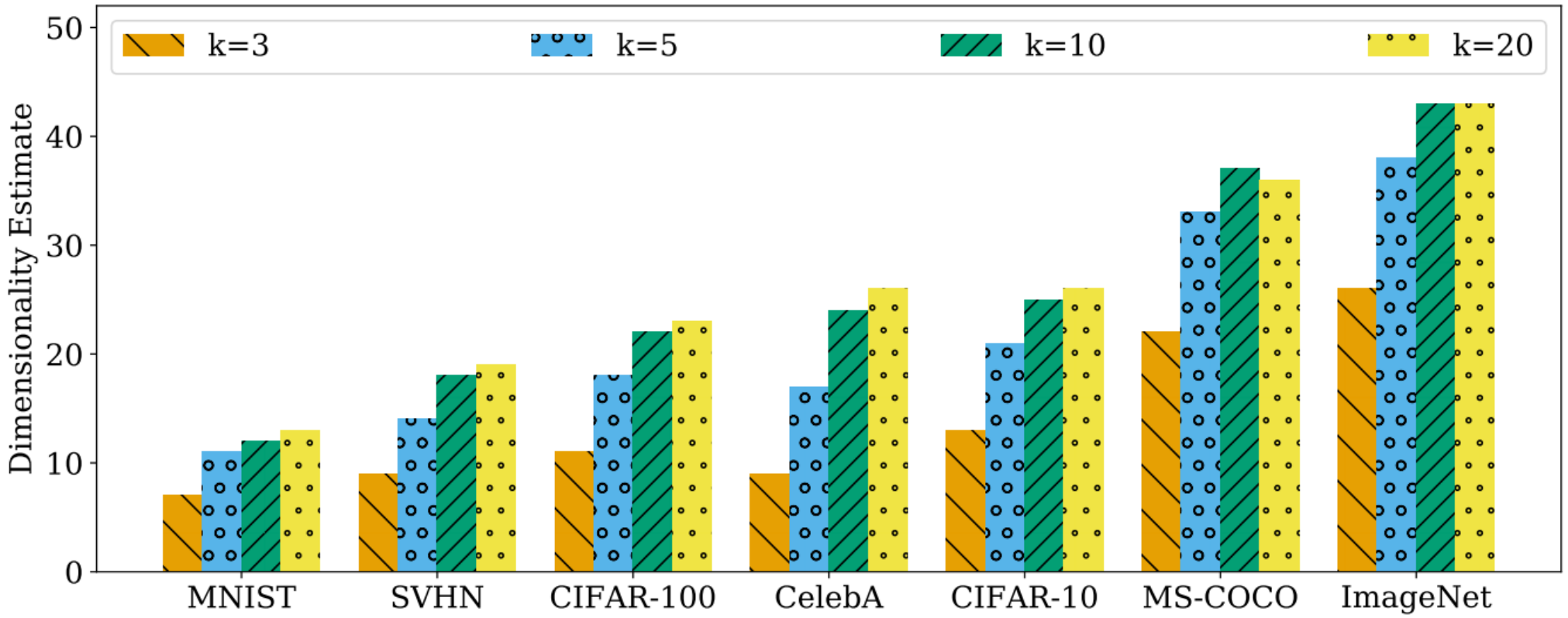


Figure 1: Estimates of the intrinsic dimension of commonly used datasets obtained using the MLE method with $k = 3, 5, 10, 20$ nearest neighbors (left to right). The trends are consistent using different k's.

Dataset Distillation (DD) Background

- Alternatives:
  - Core-set selection selects a subset of the data. For instance solutions to linear algorithms or SVM are linear combinations of training data. Sparsification induces a core set. However, this is usually still quite large, since the data (e.g. images) is required to be “real”.
  - Low dimensional projections

However, these give coarse approximations of the original dataset and hence are worse for model training, when normalized for dataset size.

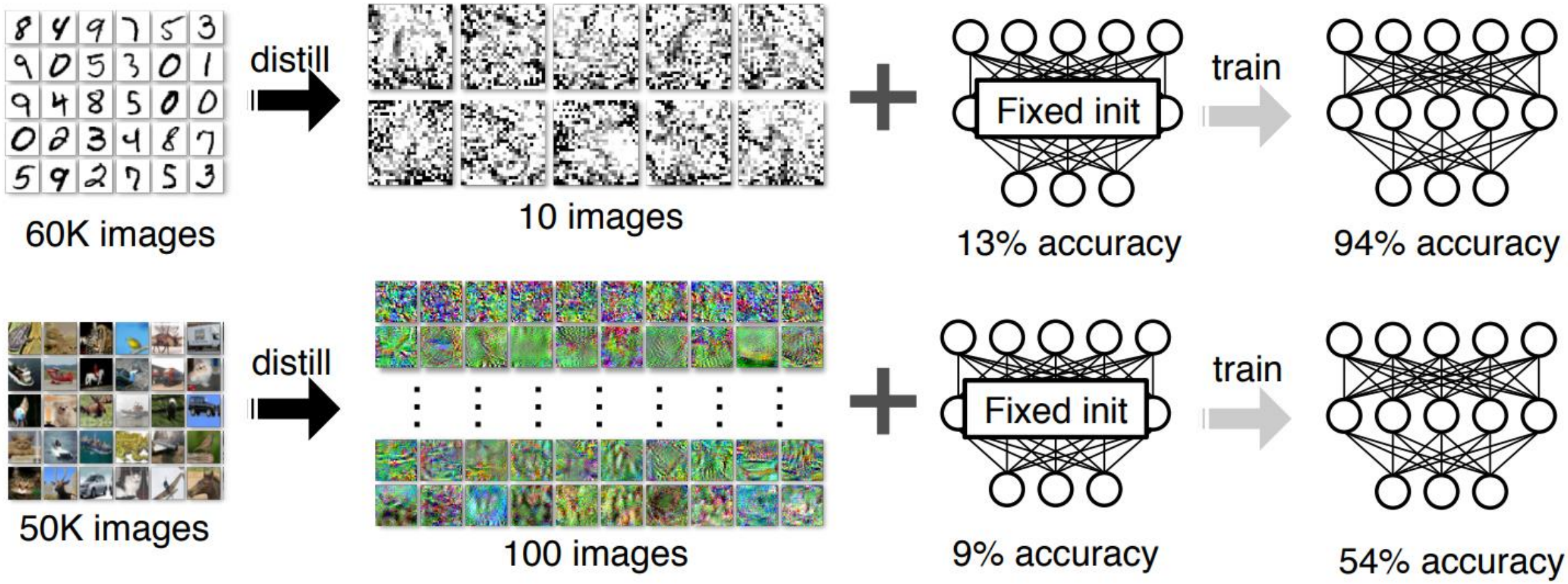
Will capture features “useful” to a model (neural net).

Dataset Distillation (DD) Background

- There is more! We distill data with respect to the model!

- For instance, using a ConvNet on images (some inductive bias) we might “adapt” the data to be even more amenable to be learned by a ConvNet (“align inductive bias”?)

Dataset Distillation (DD) Fixed initialization yields noisy images:



(a) Dataset distillation on MNIST and CIFAR10

Fits to the initialization

Dataset Distillation (DD) Wang et al. ‘18

Algorithm 1 Dataset Distillation

Input: $p(\theta_0)$: distribution of initial weights; M: the number of distilled data
Input: α: step size; n: batch size; T: the number of optimization iterations; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

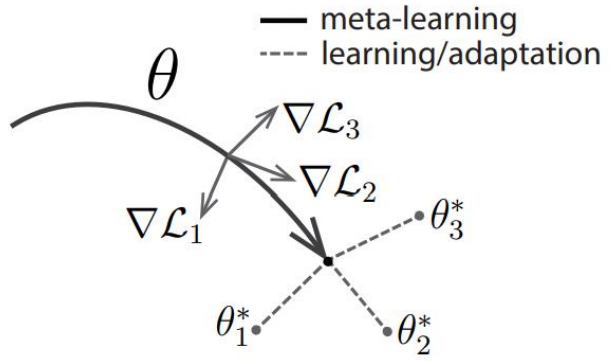
Initialize $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly, $\tilde{\eta} \leftarrow \tilde{\eta}_0$
for each training step $t = 1$ to T **do**
    Get a minibatch of real training data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^{n}$
    Sample a batch of initial weights $\theta_0^{(j)} \sim p(\theta_0)$
    for each sampled $\theta_0^{(j)}$ **do**
        Compute updated parameter with GD: $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta}\nabla_{\theta_0^{(j)}}\ell(\tilde{\mathbf{x}}, \theta_0^{(j)})$
        Evaluate the objective function on real training data: $\mathcal{L}^{(j)} = \ell(\mathbf{x}_t, \theta_1^{(j)})$
    end for
    Update $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha\nabla_{\tilde{\mathbf{x}}}\sum_j \mathcal{L}^{(j)}$, and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha\nabla_{\tilde{\eta}}\sum_j \mathcal{L}^{(j)}$
end for
Output: distilled data $\tilde{\mathbf{x}}$ and optimized learning rate $\tilde{\eta}$

Meta-learning Framework: e.g. fewshot learning

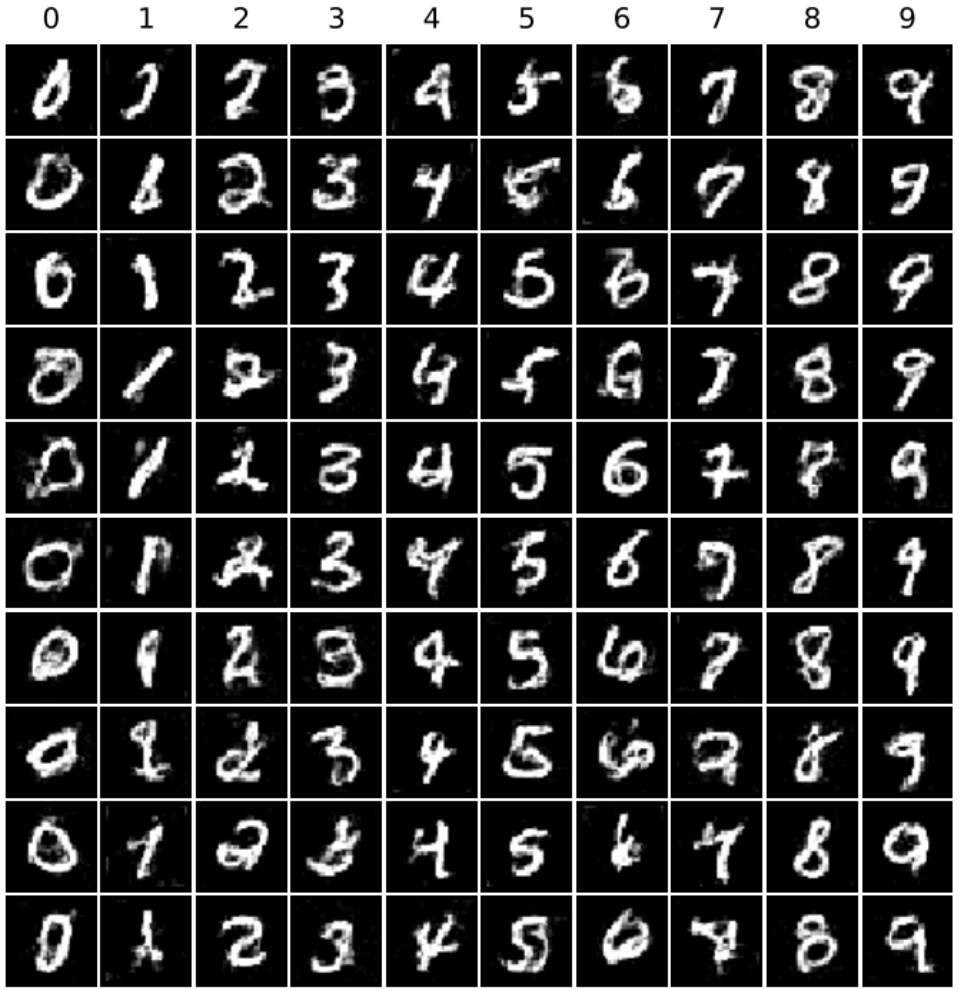
Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

MAML: Finn, Abbeel, Levine ICML'17



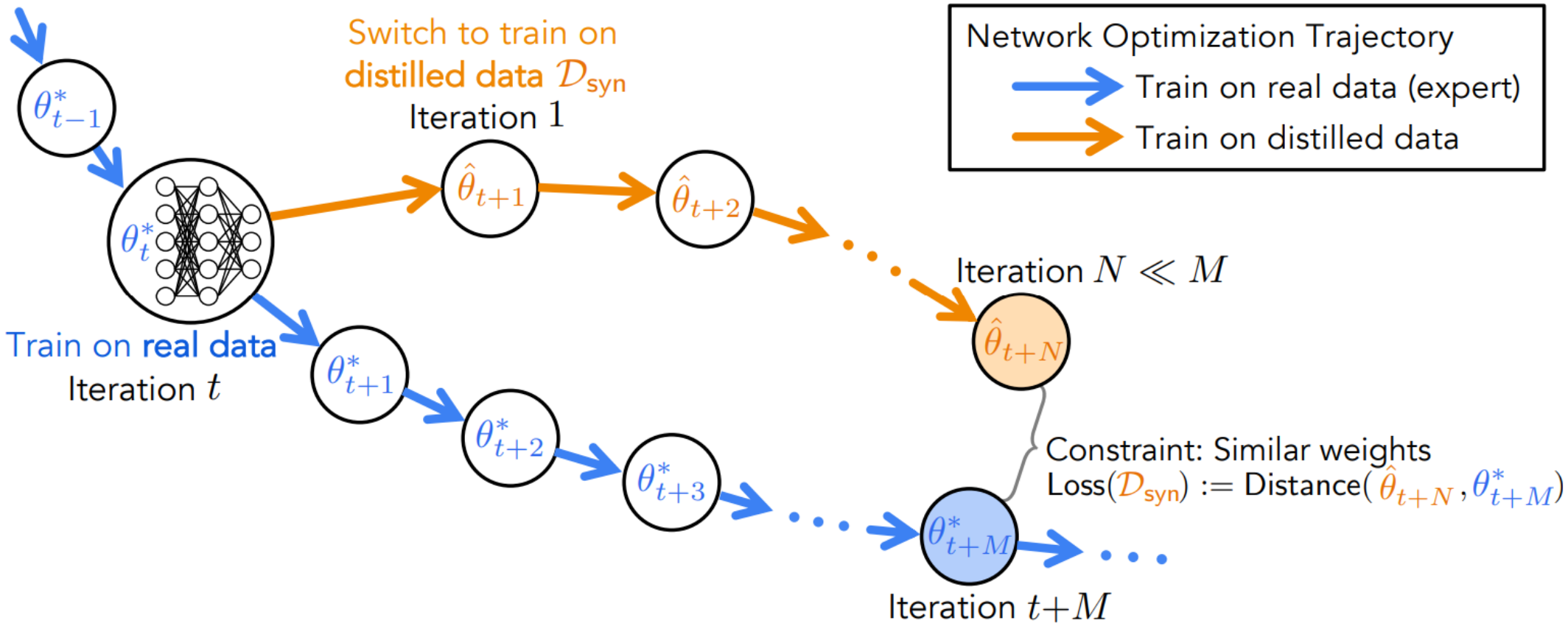
Dataset Distillation (DD) with Gradient Matching



(a) MNIST

(d) CIFAR10

Dataset Distillation (DD) Matching Training Trajectories



Cazenavette et al. CVPR'22

Dataset Distillation (DD) Matching Training Trajectories

Figure 1. Example distilled images from 32x32 CIFAR-100 (top), 64x64 Tiny ImageNet (middle), and 128x128 ImageNet subsets (bottom).

Cazenavette et al. CVPR'22

Dataset Distillation (DD) with KIP

Algorithm 1: Kernel Inducing Point (`KIP`)

Require: A target labeled dataset (X_t, y_t) along with a kernel or family of kernels.

1: Initialize a labeled support set (X_s, y_s).
2: **while** not converged **do**
3: Sample a random kernel. Sample a random batch $(\bar{X}_s, \bar{y}_s)$ from the support set. Sample a random batch $(\bar{X}_t, \bar{y}_t)$ from the target dataset.
4: Compute the kernel ridge-regression loss given by (7) using the sampled kernel and the sampled support and target data.
5: Backpropagate through $\bar{X}_s$ (and optionally $\bar{y}_s$ and any hyper-parameters of the kernel) and update the support set (X_s, y_s) by updating the subset $(\bar{X}_s, \bar{y}_s)$.
6: **end while**
7: **return** Learned support set (X_s, y_s)

Nguyen et al. ICLR'21, NeurIPS'21

Table 1: **Comparison with other methods.** The left group consists of neural network based methods. The right group consists of kernel ridge-regression. All settings for KIP involve the use of label-learning. Grayscale datasets use standard channel-wise preprocessing while RGB datasets use regularized ZCA preprocessing.

	Imgs/ Class	DC[1]	DSA[1]	KIP FC[1] aug	LS ConvNet[2,3]	KIP ConvNet[2] no aug	KIP ConvNet[2] aug
MNIST	1	91.7±0.5	88.7±0.6	85.5±0.1	73.4	**97.3±0.1**	**96.5±0.1**
	10	97.4±0.2	97.8±0.1	97.2±0.2	96.4	**99.1±0.1**	**99.1±0.1**
	50	98.8±0.1	99.2±0.1	98.4±0.1	98.3	**99.4±0.1**	**99.5±0.1**
Fashion-MNIST	1	70.5±0.6	70.6±0.6	-	65.3	**82.9±0.2**	76.7±0.2
	10	82.3±0.4	84.6±0.3	-	80.8	**91.0±0.1**	88.8±0.1
	50	83.6±0.4	88.7±0.2	-	86.9	**92.4±0.1**	91.0±0.1
SVHN	1	31.2±1.4	27.5±1.4	-	23.9	62.4±0.2	**64.3±0.4**
	10	76.1±0.6	79.2±0.5	-	52.8	79.3±0.1	**81.1±0.5**
	50	82.3±0.3	**84.4±0.4**	-	76.8	82.0±0.1	**84.3±0.1**
CIFAR-10	1	28.3±0.5	28.8±0.7	40.5±0.4	26.1	**64.7±0.2**	63.4±0.1
	10	44.9±0.5	52.1±0.5	53.1±0.5	53.6	**75.6±0.2**	**75.5±0.1**
	50	53.9±0.5	60.6±0.5	58.6±0.4	65.9	78.2±0.2	**80.6±0.1**
CIFAR-100	1	12.8±0.3	13.9±0.3	-	23.8	**34.9±0.1**	33.3±0.3
	10	25.2±0.3	32.3±0.3	-	39.2	47.9±0.2	**49.5±0.3**

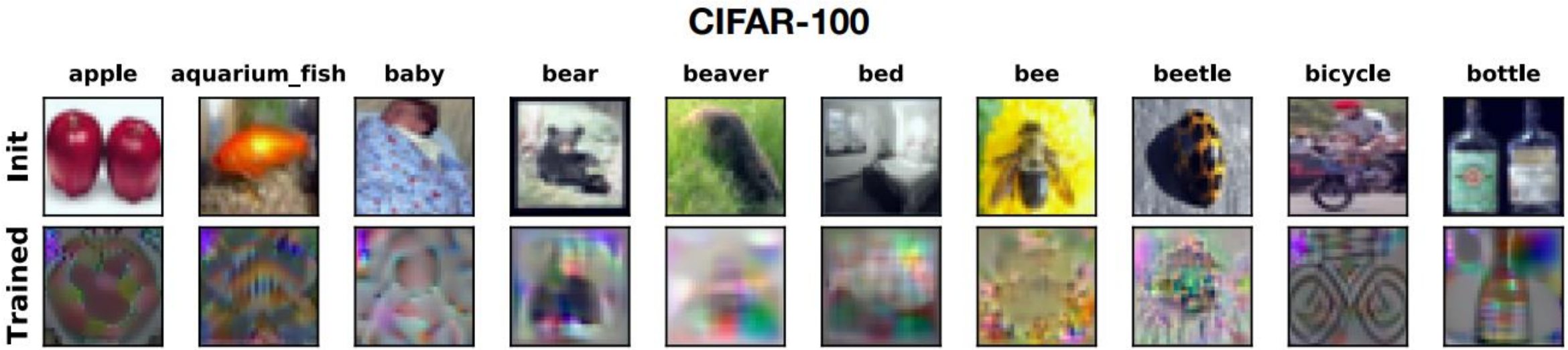
[1] DC [Zhao et al., 2021], DSA [Zhao and Bilen, 2021], KIP FC [Nguyen et al., 2021].

[2] Ours.

[3] LD [Bohdal et al., 2020] is another baseline which distills only labels using the AlexNet architecture. Our LS achieves higher test accuracy than theirs in every dataset category.

Nguyen et al. NeurIPS'21

Dataset Distillation (DD) with KIP



Nguyen et al. NeurIPS'21

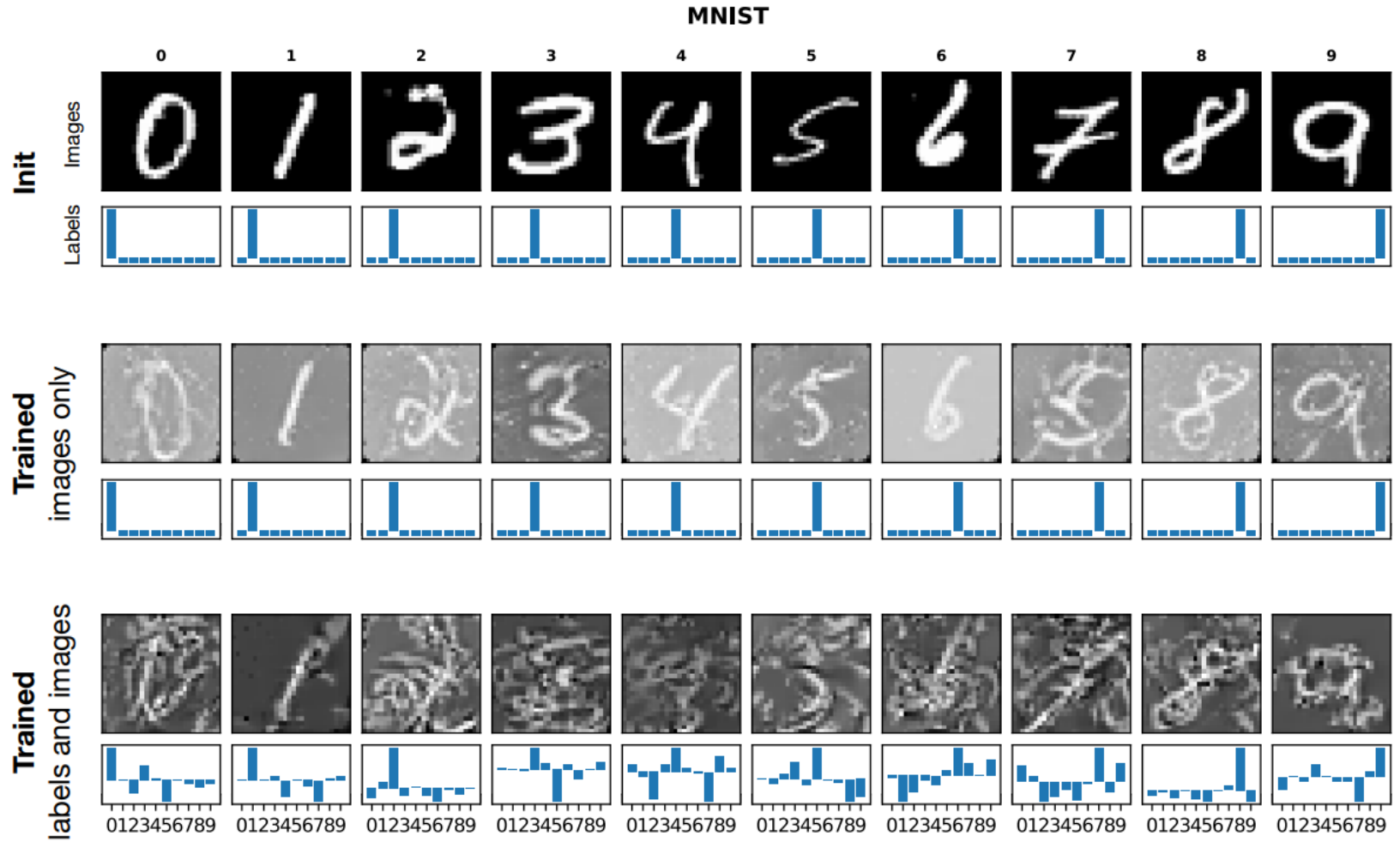
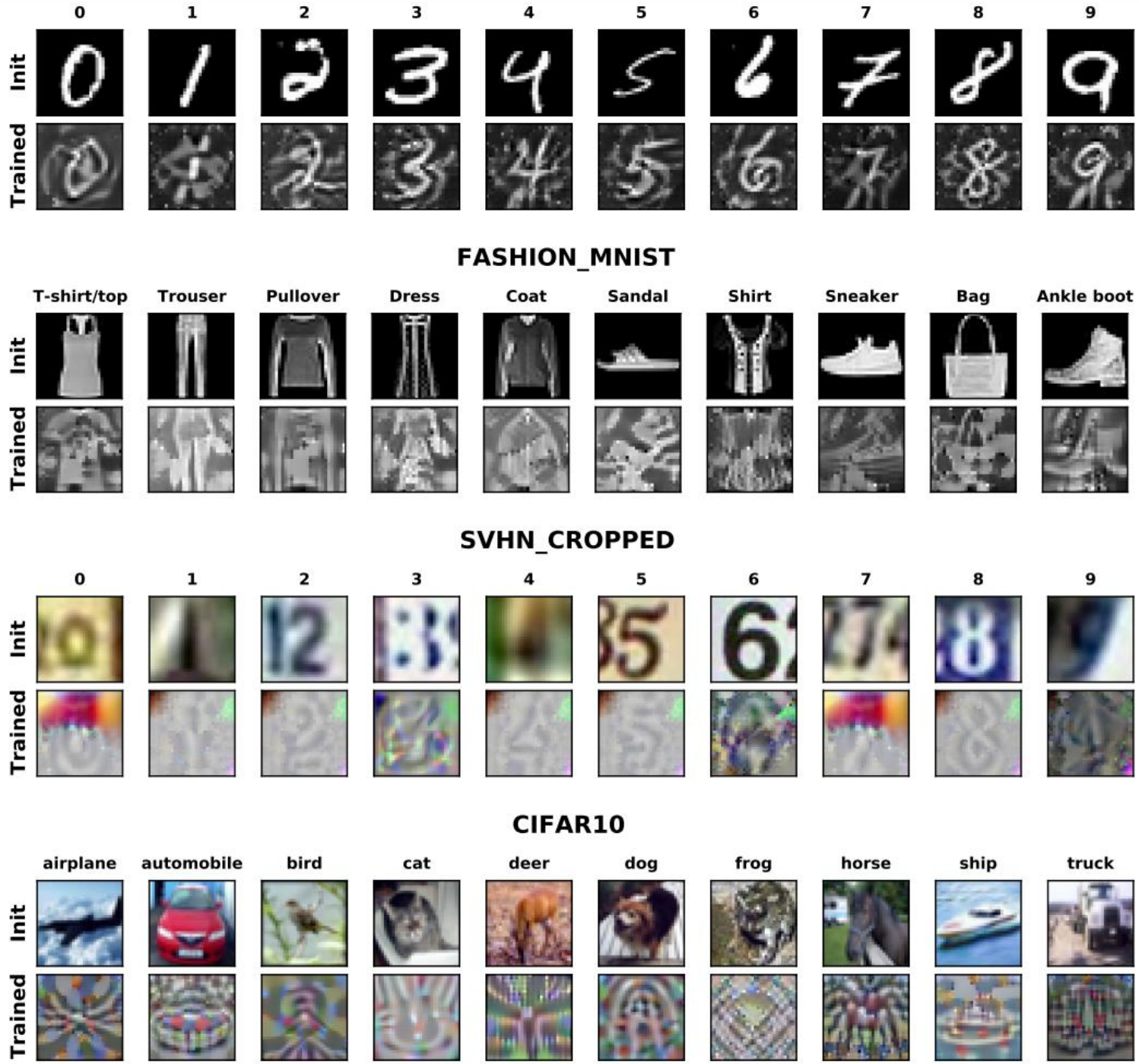


Figure 6: **Dataset distillation with trainable and non-trainable labels.** *Top row:* initialization of support images and labels. *Middle row:* trained images if labels remain fixed. *Bottom row:* trained images and labels, jointly optimized. Settings: 100 images distilled, no augmentations.

Nguyen et al. NeurIPS'21



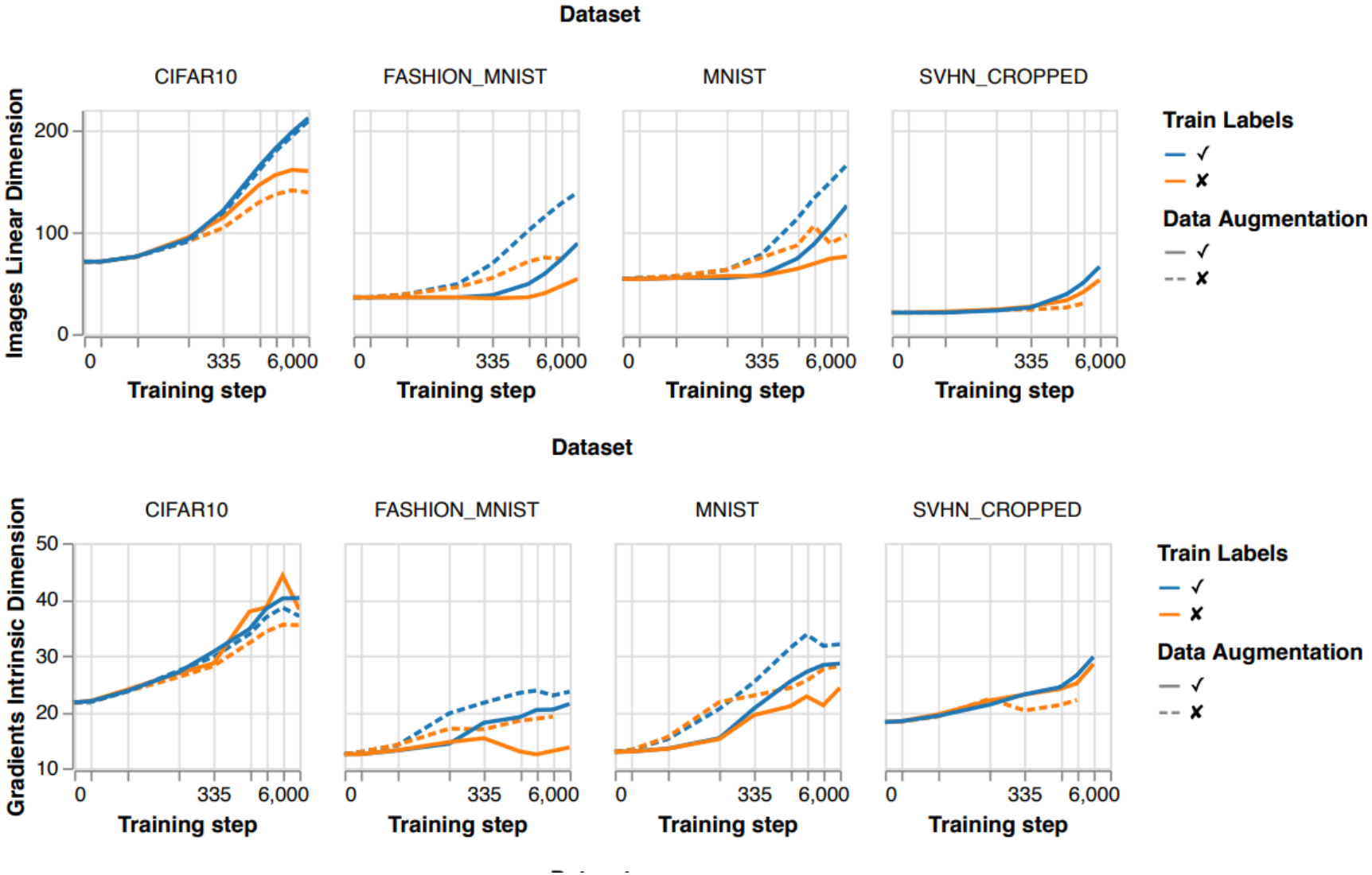
Nguyen et al. NeurIPS'21

Dataset Distillation (DD) with KIP

- Distilled dataset becomes more corelated across images
- Intrinsic dimension of images increases

Nguyen et al. NeurIPS'21

Dimension of Data grows



Nguyen et al. NeurIPS'21

Kernels, Data and Physics
or
How can (statistical) physics tools help the DL practitioner

Julia Kempe, CDS & Courant Institute, NYU

Les Houches 2022

Dataset Distillation (DD) with KIP

Algorithm 1: Kernel Inducing Point (`KIP`)

Require: A target labeled dataset (X_t, y_t) along with a kernel or family of kernels.

1: Initialize a labeled support set (X_s, y_s).
2: **while** not converged **do**
3: Sample a random kernel. Sample a random batch $(\bar{X}_s, \bar{y}_s)$ from the support set. Sample a random batch $(\bar{X}_t, \bar{y}_t)$ from the target dataset.
4: Compute the kernel ridge-regression loss given by (7) using the sampled kernel and the sampled support and target data.
5: Backpropagate through $\bar{X}_s$ (and optionally $\bar{y}_s$ and any hyper-parameters of the kernel) and update the support set (X_s, y_s) by updating the subset $(\bar{X}_s, \bar{y}_s)$.
6: **end while**
7: **return** Learned support set (X_s, y_s)

Nguyen et al. ICLR'21, NeurIPS'21

Extend this framework: Adversarial Robustness

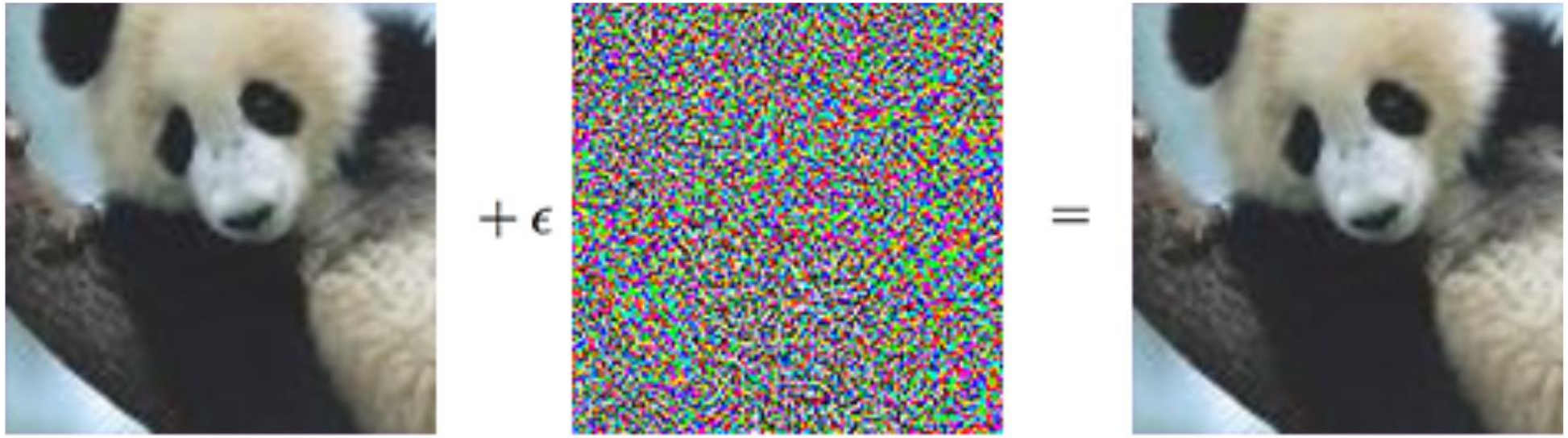
Overview

Sample complexity: Schmidt et al. Adversarially robust generalization requires more data

Current best defense: Adversarial Training

Accuracy-Robustness trade-off: intuition

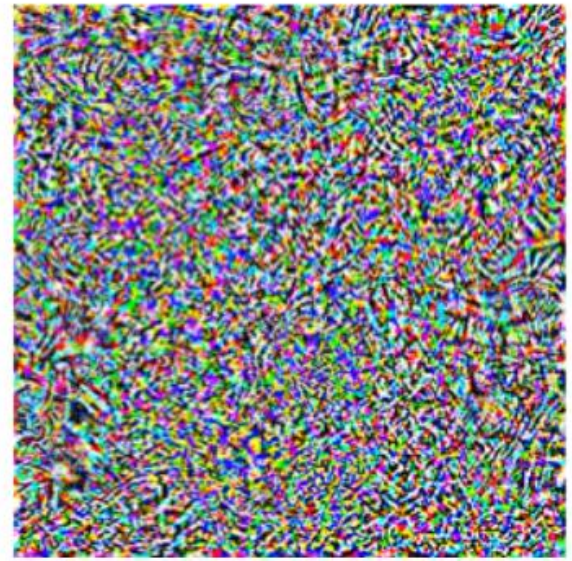
Adversarial Attacks



An adversarial input, overlaid on a typical image, can cause a classifier to miscategorize a panda as a gibbon.

“pig”

+ 0.005 x



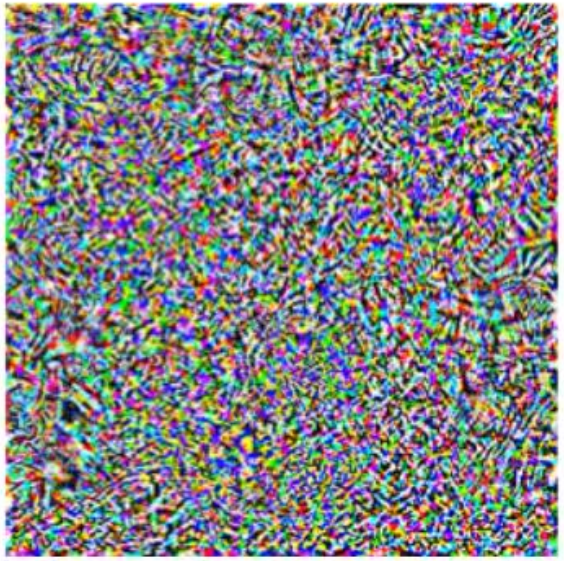
=

“airliner”

“Deep Learning makes Pigs Fly” [Szegedy et al. ‘13]

“pig”

+ 0.005 x

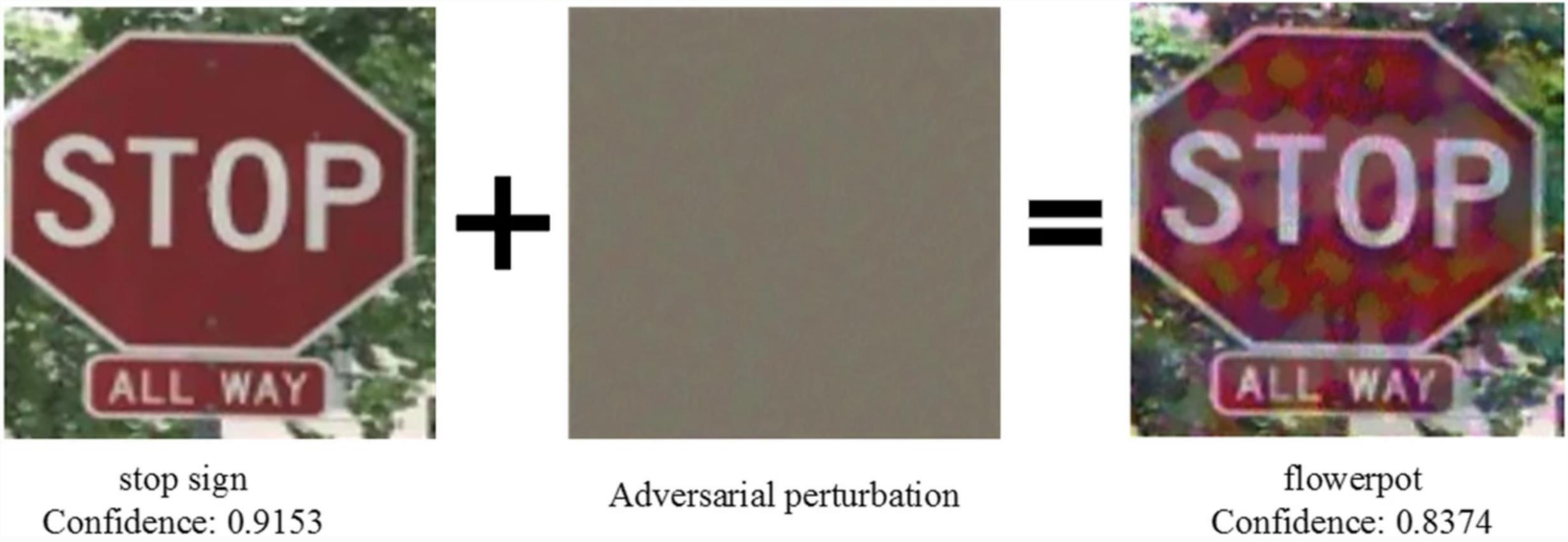


=

“airliner”

Adversarial Attacks are pervasive!

From: Defense against adversarial attacks in traffic sign images identification based on 5G

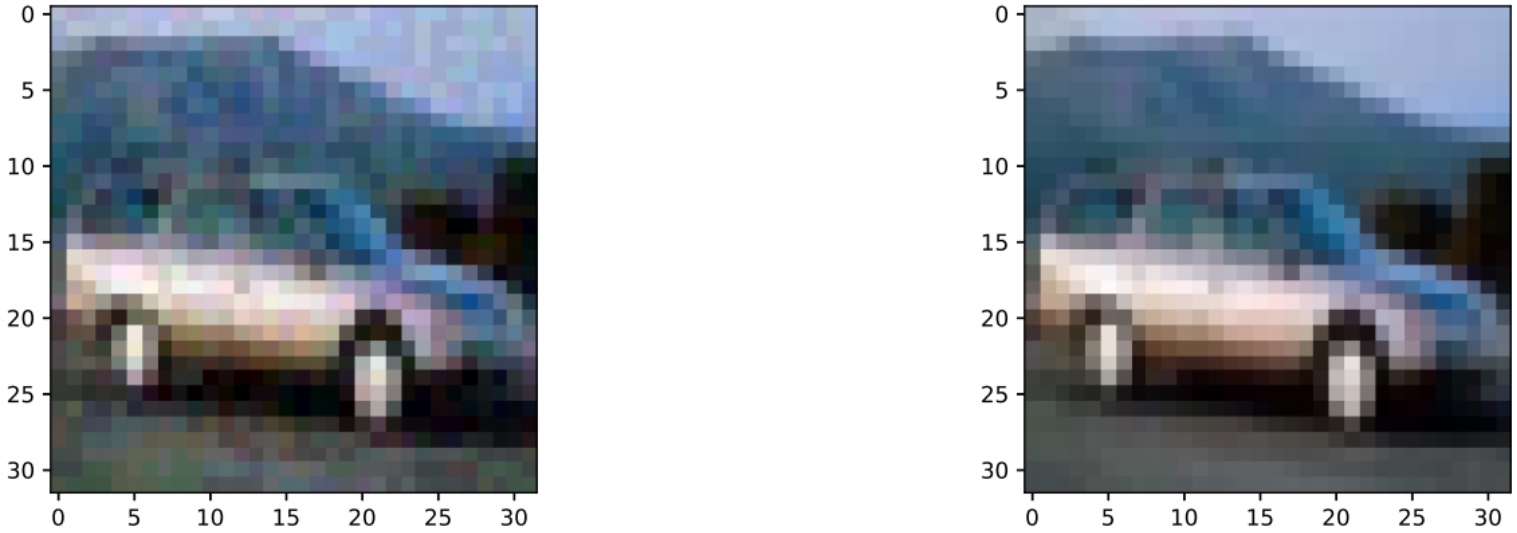


This is an adversarial example crafted for deep learning model (this figure shows an adversarial example generated by FGSM method)

Adversarial Attacks are pervasive!

“Shape-shifter” [Chao et al ‘18]

Adversarial Attacks are pervasive and transfer well…



(a) Adversarially perturbed input

(b) Clean input

Figure 2: Comparison between adversarial image computed by an infinite neural network and its clean counterpart.

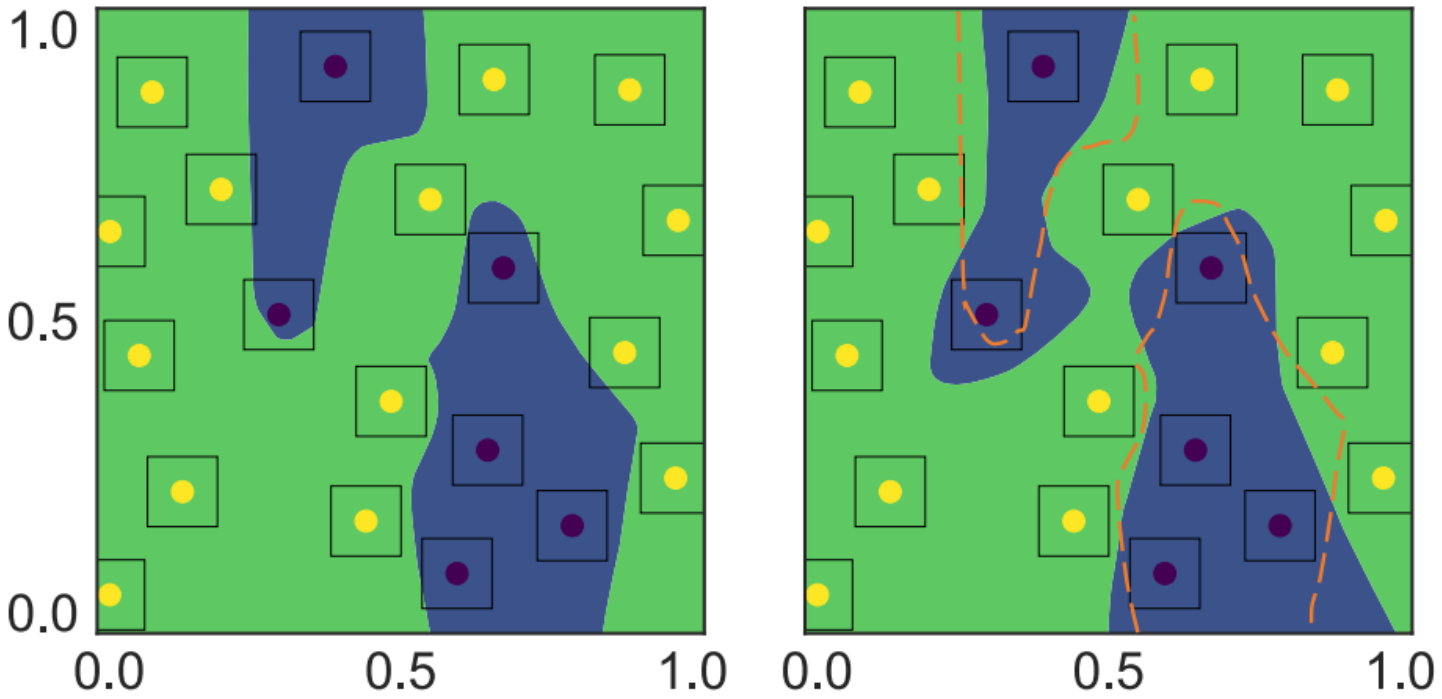
“NTK-adversary” [Tsilivis, JK ‘21]

Kernels, Data and Physics
or
How can (statistical) physics tools help the DL practitioner

Julia Kempe, CDS & Courant Institute, NYU

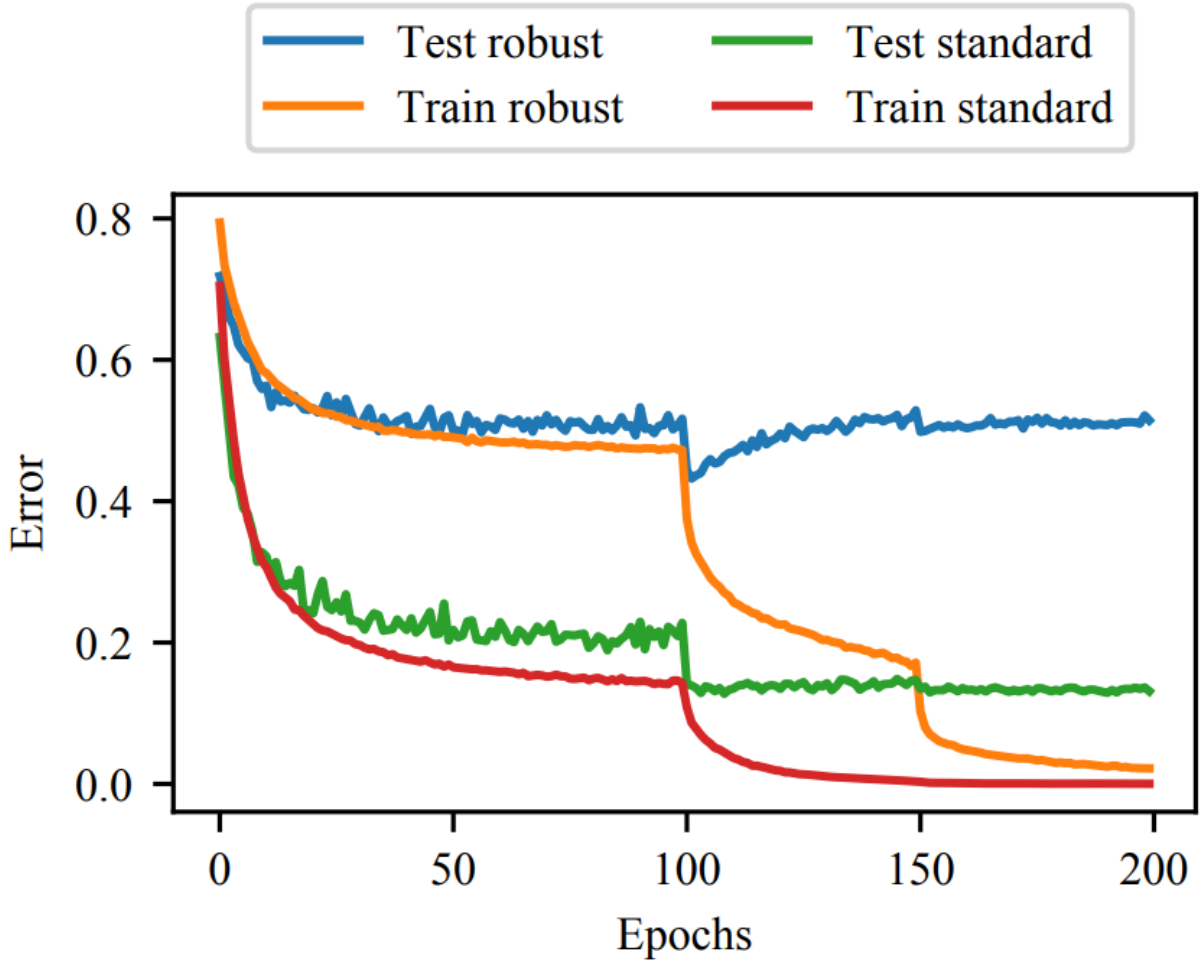
Les Houches 2022

Intuition



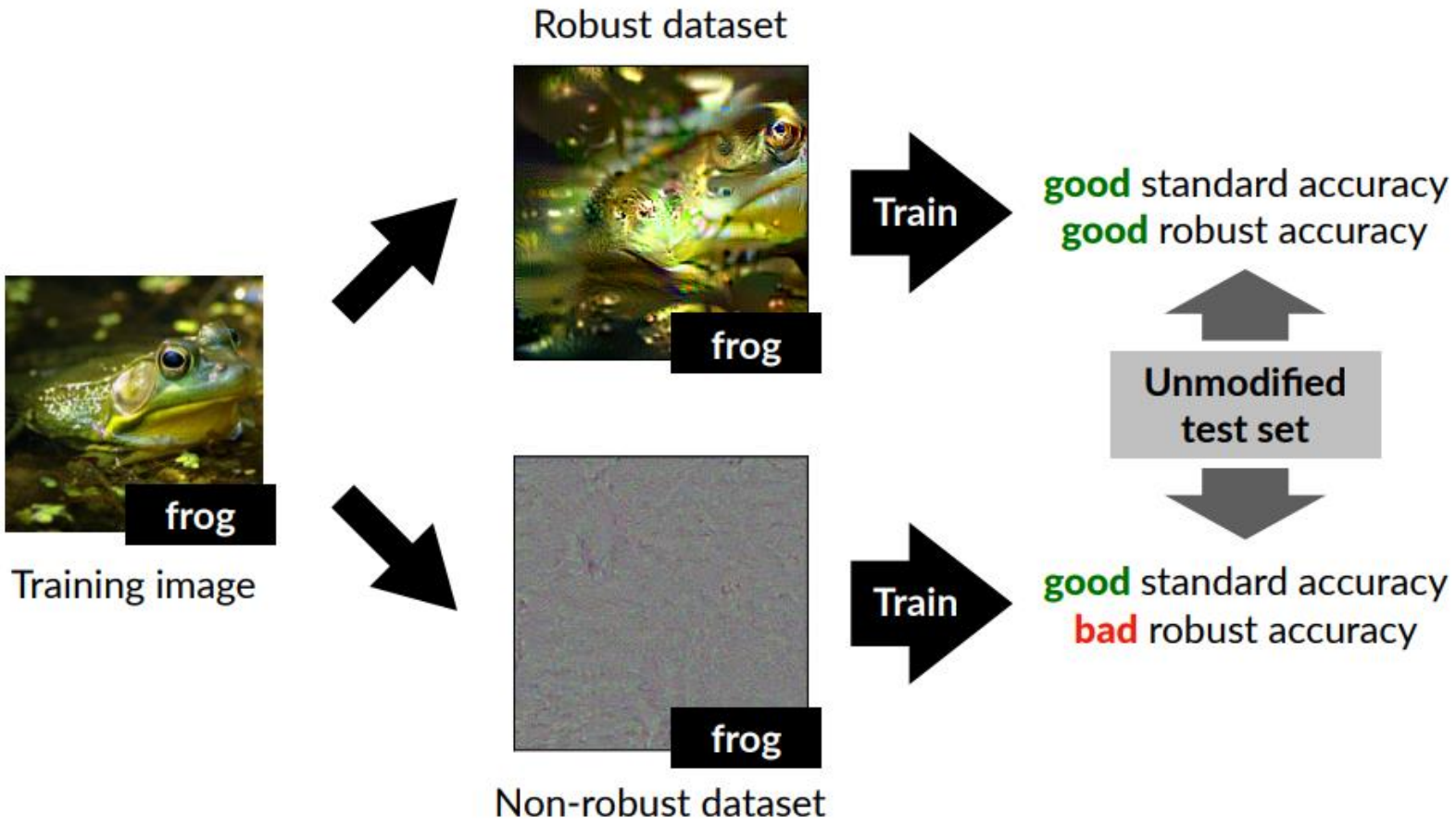
Zhang et al ICML'19

Robust overfitting differs from “standard” overfitting



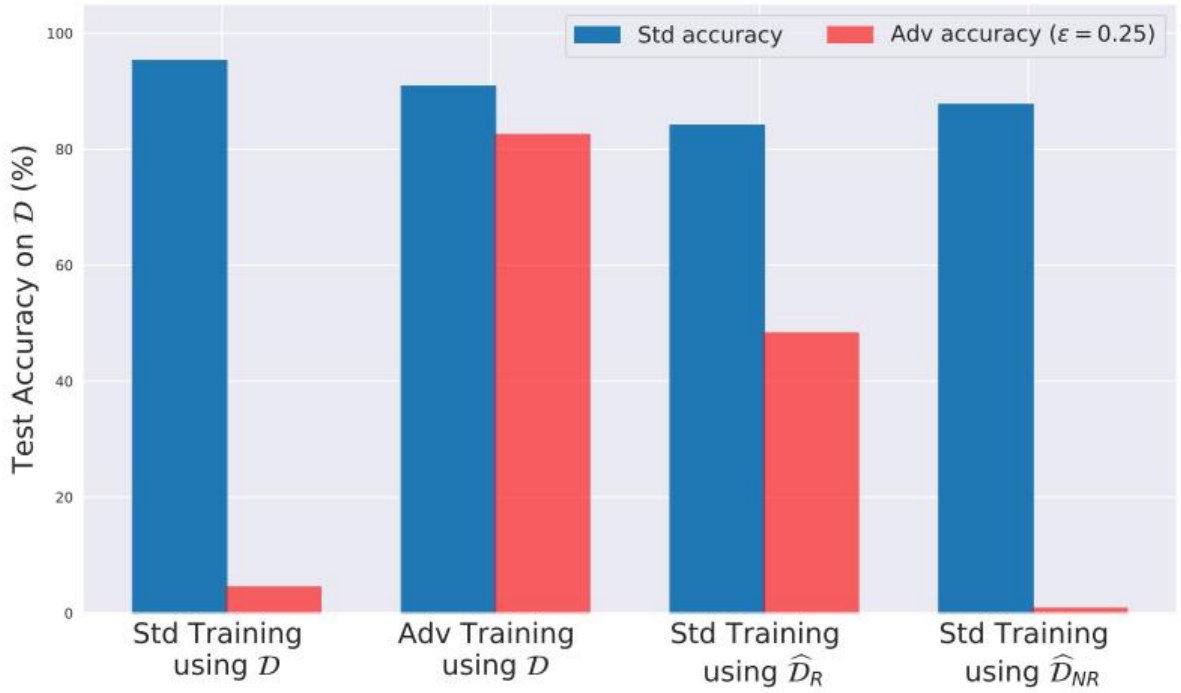
[Rice, Wong, Kolter ICML’20]

“Adversarial Examples are not bugs, they are features” [Ilyas et al.’19]



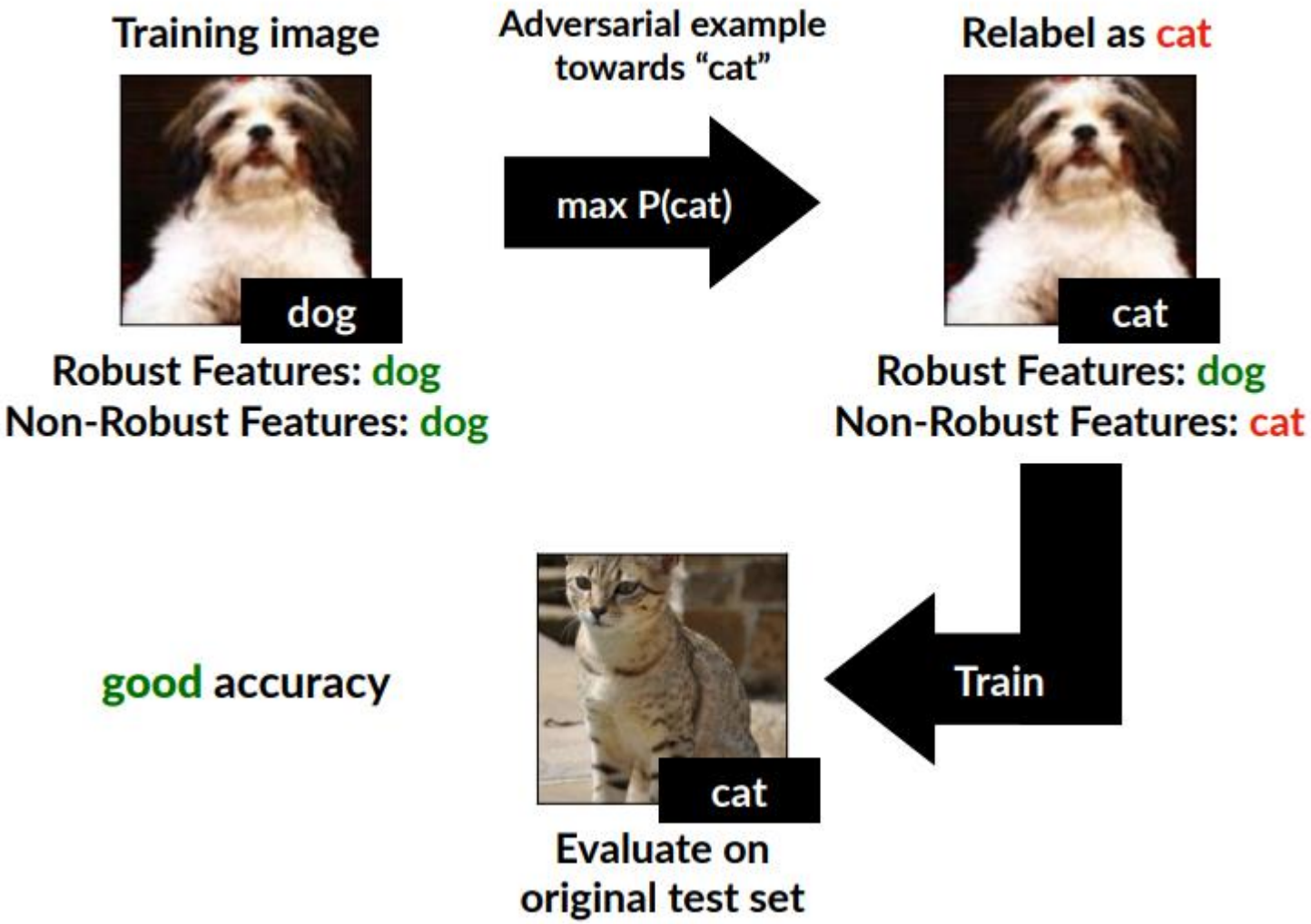
“…not bugs, features” [Ilyas et al.’19]

(a)



(b)

“…not bugs, features” [Ilyas et al.’19]



“…not bugs, features” [Ilyas et al.’19]

- Uses only “non-robust” features
- Achieves std accuracy of 63% (compared to 95% when trained on original dataset)

Adversarial Robustness Remains an Open Problem

Advers oblem

Taxonomy	Publication	Model Architecture	Attack	ϵ	Dataset	Accuracy
Adversarial Regularization	[Qin *et al.*, 2019]	ResNet-152	PGD_{50}	4/255	ImageNet	47.00%
	[Zhang *et al.*, 2019b]	Wide ResNet	CW_{10}	0.031/1	CIFAR-10	84.03%
	[Wang *et al.*, 2020]	ResNet-18	PGD_{20}	8/255	CIFAR-10	55.45%
	[Kannan *et al.*, 2018]	InceptionV3	PGD_{10}	16/255	ImageNet	27.90%
	[Mao *et al.*, 2019]	Wide ResNet	PGD_{20}	8/255	CIFAR-10	50.03%
Curriculum	[Zhang *et al.*, 2020]	Wide ResNet	PGD_{20}	16/255	CIFAR-10	49.86%
	[Cai *et al.*, 2018]	DenseNet-161	PGD_{7}	8/255	CIFAR-10	69.27%
	[Wang *et al.*, 2019]	8-Layer ConvNet	PGD_{20}	8/255	CIFAR-10	42.40%
Ensemble	[Pang *et al.*, 2019]	Wide ResNet	PGD_{10}	0.005	CIFAR-100	32.10%
	[Kariyappa and Qureshi, 2019]	ResNet-20	PGD_{30}	0.09/1	CIFAR-10	46.30%
	[Yang *et al.*, 2020a]	ResNet-20	PGD_{20}	0.01/1	CIFAR-10	52.40%
Adaptive ϵ	[Balaji *et al.*, 2019]	ResNet-152	PGD_{1000}	8/255	ImageNet	59.28%
	[Ding *et al.*, 2020]	Wide ResNet	PGD_{100}	8/255	CIFAR-10	47.18%
	[Cheng *et al.*, 2020]	Wide ResNet	PGD_{20}	8/255	CIFAR-10	73.38%
Semi-Unsupervised	[Alayrac *et al.*, 2019]	Wide ResNet	FGSM	8/255	CIFAR-10	62.18%
	[Carmon *et al.*, 2019]	Wide ResNet	PGD_{10}	8/255	CIFAR-10	63.10%
	[Zhai *et al.*, 2019]	Customized ResNet	PGD_{7}	8/255	CIFAR-10	42.48%
	[Hendrycks *et al.*, 2019]	Wide ResNet	PGD_{20}	0.3/1	ImageNet	50.40%
Efficient	[Shafahi *et al.*, 2019]	Wide ResNet	PGD_{100}	8/255	CIFAR-10	46.19%
	[Wong *et al.*, 2020]	ResNet-50	PGD_{40}	2/255	ImageNet	43.43%
	[Andriushchenko and Flammarion, 2020]	ResNet-50	PGD_{50}	2/255	ImageNet	41.40%
	[Kim *et al.*, 2021]	PreActResNet-18	FGSM	8/255	CIFAR-10	50.50%
	[S. and Babu, 2020]	Wide ResNet	PGD_{40}	8/255	MNIST	88.51%
	[Song *et al.*, 2019]	Customized ConvNet	PGD_{20}	4/255	CIFAR-10	58.10%
	[Vivek and Babu, 2020]	Wide ResNet	PGD_{100}	0.3/1	MNIST	90.03%
	[Huang *et al.*, 2020]	Wide ResNet	PGD_{20}	8/255	CIFAR-10	45.80%
	[Zhang *et al.*, 2019a]	Wide ResNet	PGD_{20}	8/255	CIFAR-10	47.98%
Others	[Dong *et al.*, 2020]	Wide ResNet	PGD_{20}	8/255	CIFAR-100	29.40%
	[Wang and Zhang, 2019]	Wide ResNet	CW_{200}	4/255	CIFAR-10	60.30%
	[Zhang and Wang, 2019]	Wide ResNet	PGD_{20}	8/255	CIFAR-100	47.20%
	[Pang *et al.*, 2020]	Wide ResNet	PGD_{500}	8/255	CIFAR-10	60.75%
	[Lee *et al.*, 2020]	PreActResNet-18	PGD_{20}	8/255	Tiny ImageNet	20.31%
Benchmark	[Madry *et al.*, 2018]	ResNet-50	PGD_{20}	8/255	CIFAR-10	45.80%

RobustBench

A standardized benchmark for adversarial robustness

The goal of **RobustBench** is to systematically track the *real* progress in adversarial robustness. There are already more than 3'000 papers on this topic, but it is still unclear which approaches really work and which only lead to overestimated robustness. We start from benchmarking common corruptions, ℓ_∞- and ℓ_2-robustness since these are the most studied settings in the literature. We use AutoAttack, an ensemble of white-box and black-box attacks, to standardize the evaluation (for details see our paper) of the ℓ_p robustness and CIFAR-10-C for the evaluation of robustness to common corruptions. Additionally, we open source the RobustBench library that contains models used for the leaderboard to facilitate their usage for downstream applications.

To prevent potential overadaptation of new defenses to AutoAttack, we also welcome external evaluations based on *adaptive attacks*, especially where AutoAttack flags a potential overestimation of robustness. For each model, we are interested in the best known robust accuracy and see AutoAttack and adaptive attacks as complementary.

News:

- **May 2022:** We have extended the common corruptions leaderboard on ImageNet with 3D Common Corruptions (ImageNet-3DCC). ImageNet-3DCC

Available Leaderboards

CIFAR-10 (ℓ_∞) | CIFAR-10 (ℓ_2) | CIFAR-10 (Corruptions) | CIFAR-100 (ℓ_∞) | CIFAR-100 (Corruptions)

ImageNet (ℓ_∞) | ImageNet (Corruptions: IN-C, IN-3DCC)

Leaderboard: CIFAR-10, $\ell_\infty = 8/255$, untargeted attack

Show 15 entries Search: Papers, architectures, ve

Rank	Method	Standard accuracy	AutoAttack robust accuracy	Best known robust accuracy	AA eval. potentially unreliable	Extra data	Architecture	Venue
1	Fixing Data Augmentation to Improve Adversarial Robustness *66.56% robust accuracy is due to the original evaluation (AutoAttack + MultiTargeted)*	92.23%	66.58%	66.56%	×	☑	WideResNet-70-16	arXiv, Mar 2021
2	Improving Robustness using Generated Data *It uses additional 100M synthetic images in training. 66.10% robust accuracy is due to the original evaluation (AutoAttack + MultiTargeted)*	88.74%	66.11%	66.10%	×	×	WideResNet-70-16	NeurIPS 2021
3	Uncovering the Limits of Adversarial Training against Norm-Bounded Adversarial Examples *65.87% robust accuracy is due to the original evaluation (AutoAttack + MultiTargeted)*	91.10%	65.88%	65.87%	×	☑	WideResNet-70-16	arXiv, Oct 2020

Slide courtesy
Nikos Tsilivis

Algorithm 1: Adversarial KIP

Input: A training dataset $\mathcal{D}_{\text{train}} = \{\mathcal{X}, \mathcal{Y}\}$.

Output: A new dataset $\mathcal{D}_{\text{rob}}$.

Sample data $\mathcal{S} = \{\mathcal{X}_S, \mathcal{Y}_S\}$ from $\mathcal{D}_{\text{train}}$;

for i $\leftarrow 1$ **to** epochs **do**

  Sample data $\mathcal{T} = \{\mathcal{X}_T, \mathcal{Y}_T\}$ from $\mathcal{D}_{\text{train}}$;

  for j $\leftarrow 1$ **to** pgd_steps **do**

    $\mathcal{X}_T \leftarrow \mathcal{X}_T + \alpha \cdot \text{sign}(\nabla_{\mathcal{X}_T} \mathcal{L}_{\text{ce}}(K_{\mathcal{X}_T \mathcal{X}_S} K_{\mathcal{X}_S \mathcal{X}_S}^{-1} \mathcal{Y}_S, \mathcal{Y}_T))$;

    $\mathcal{X}_T \leftarrow \Pi_{\mathcal{B}_\epsilon}(\mathcal{X}_T)$;

  $\mathcal{X}_S \leftarrow \mathcal{X}_S - \lambda \nabla_{\mathcal{X}_S} \mathcal{L}(K_{\mathcal{X}_T \mathcal{X}_S} K_{\mathcal{X}_S \mathcal{X}_S}^{-1} \mathcal{Y}_S, \mathcal{Y}_T)$;

  $\mathcal{Y}_S \leftarrow \mathcal{Y}_S - \lambda \nabla_{\mathcal{Y}_S} \mathcal{L}(K_{\mathcal{X}_T \mathcal{X}_S} K_{\mathcal{X}_S \mathcal{X}_S}^{-1} \mathcal{Y}_S, \mathcal{Y}_T)$;

$\mathcal{D}_{\text{rob}} \leftarrow (\mathcal{X}_S, \mathcal{Y}_S)$

Backpropagate rob. error through data

Attack Kernel

Experiments

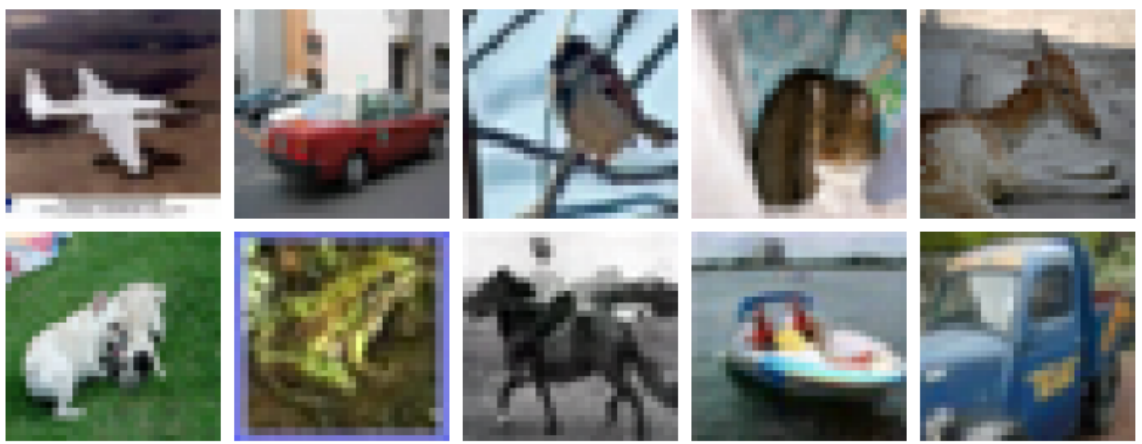
- Use a simple kernel (Fully Connected) to learn a new dataset.
- Train Neural Nets on this dataset with standard training.

	Adv KIP		AT Baseline	
Architecture	Clean (%)	PGD-20 (%)	Clean (%)	PGD-20 (%)
Simple CNN	72.10 ± 0.10	67.03 ± 0.24	58.07	31.49
AlexNet	68.87 ± 0.76	49.06 ± 0.63	44.35	24.41
VGG11	74.88 ± 0.45	53.18 ± 10.32	69.65	24.68

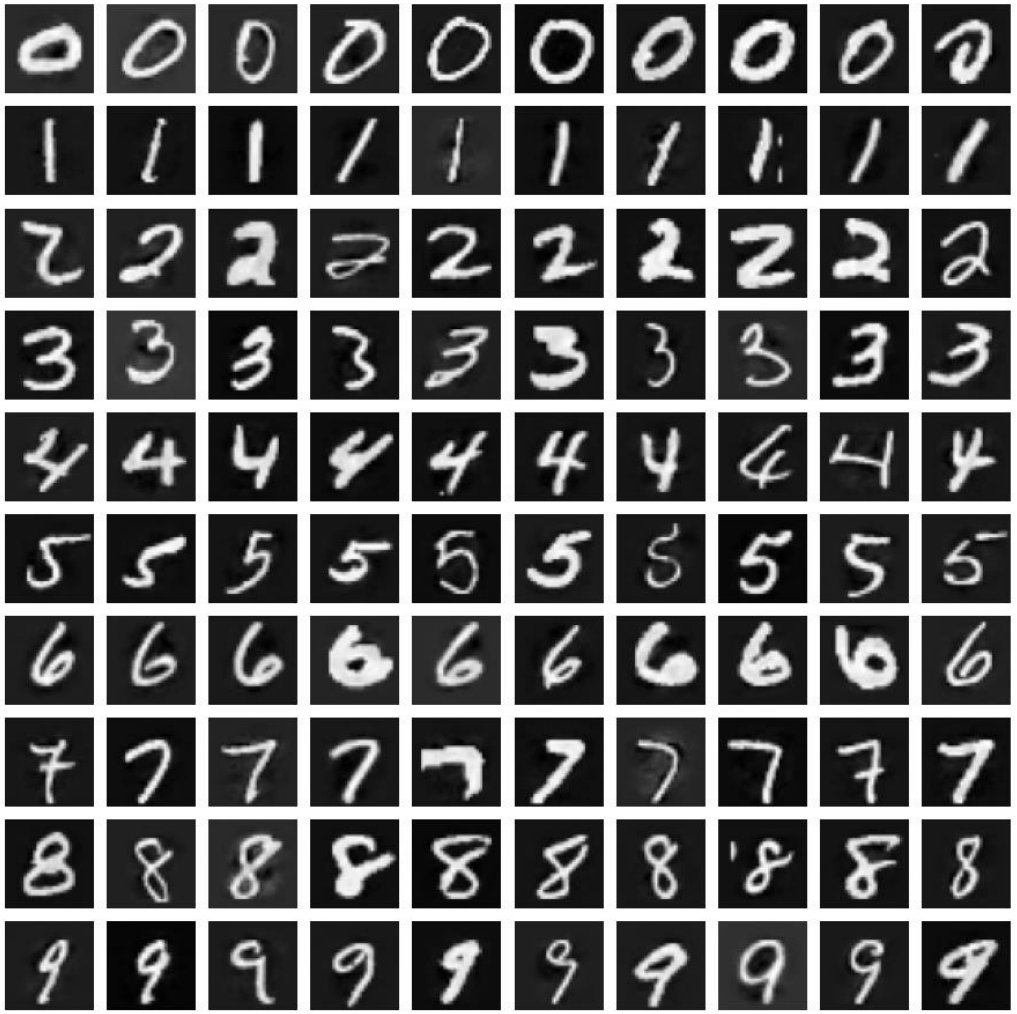
Table: Test accuracies of several convolutional architectures trained on a distilled CIFAR-10 dataset. Setting: CIFAR-10, ℓ_∞ adversary, $\epsilon = 8/255$, no data augmentation.

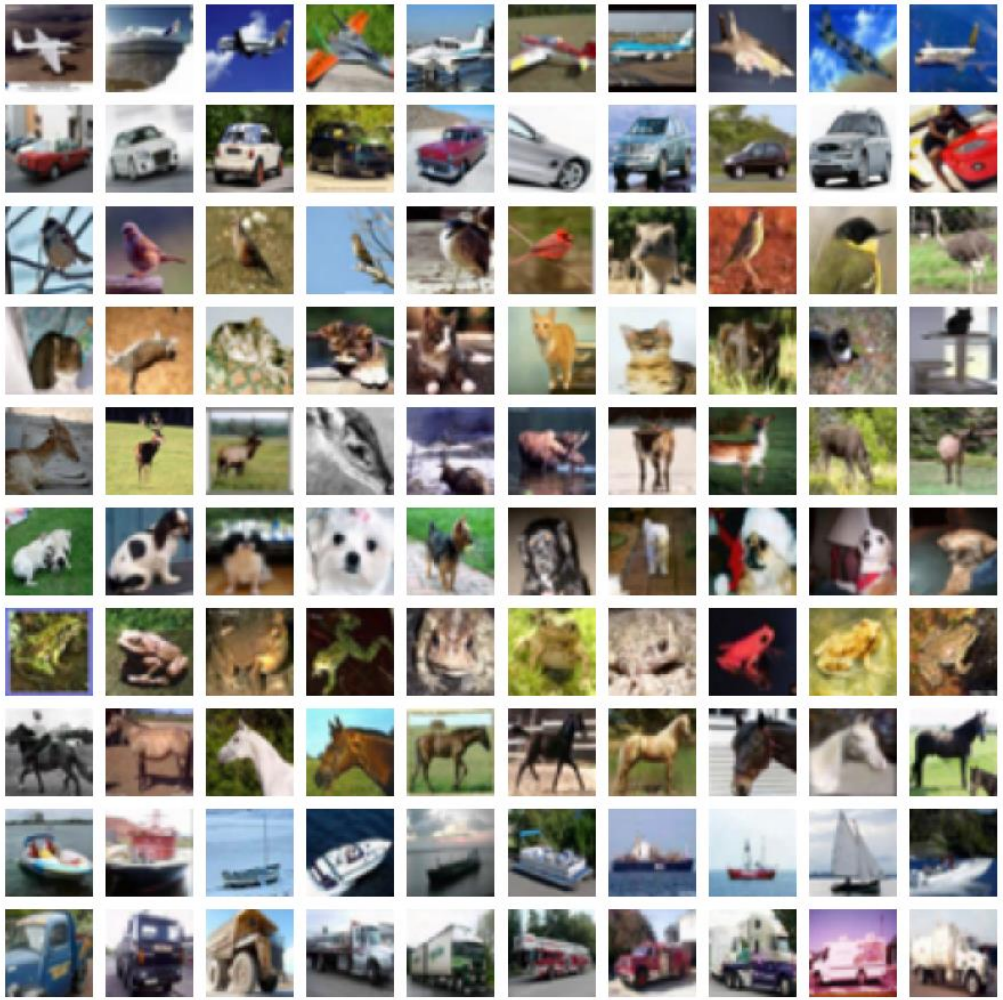
Slide courtesy
Nikos Tsilivis

Models achieve astonishing robustness to PGD attacks!



advKIP Data [Tsilivis, Su, K ‘22]





Neural Tangent generalization attacks [Yuan&Wu ICML'22]

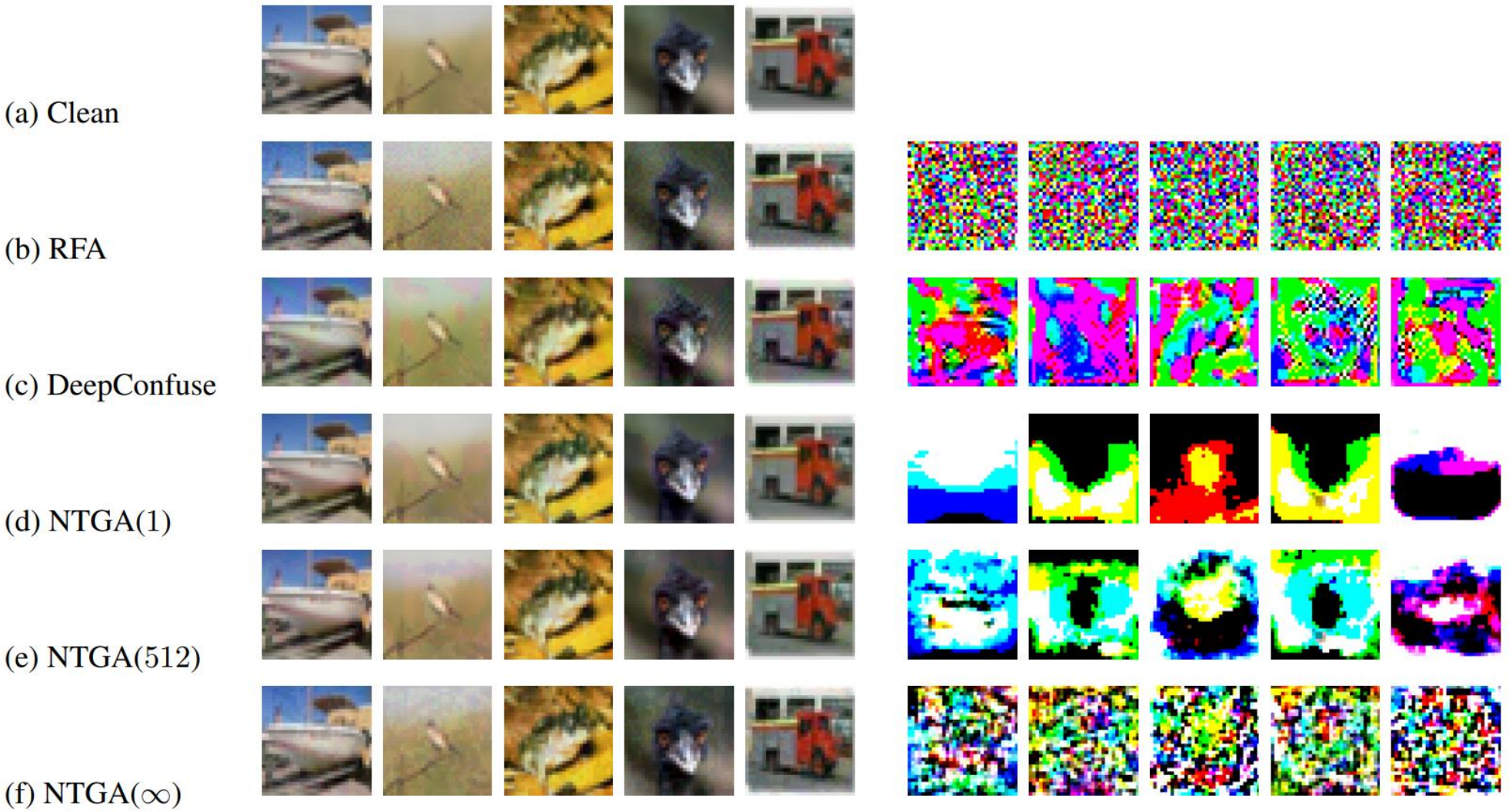
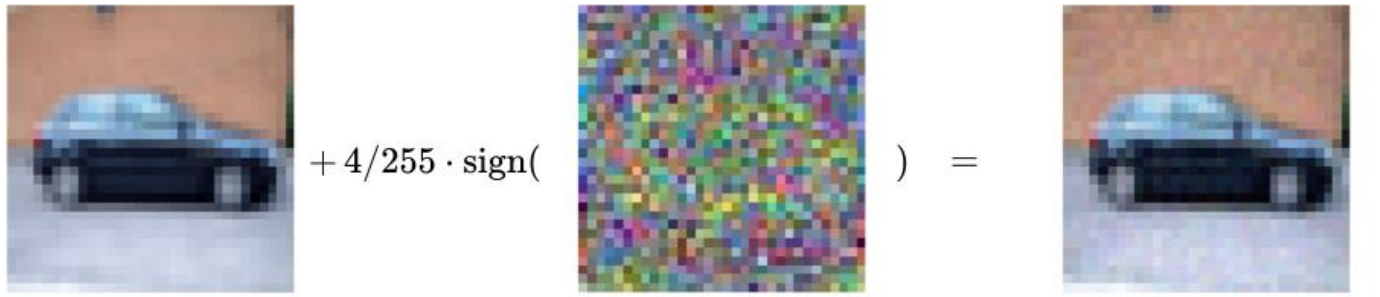
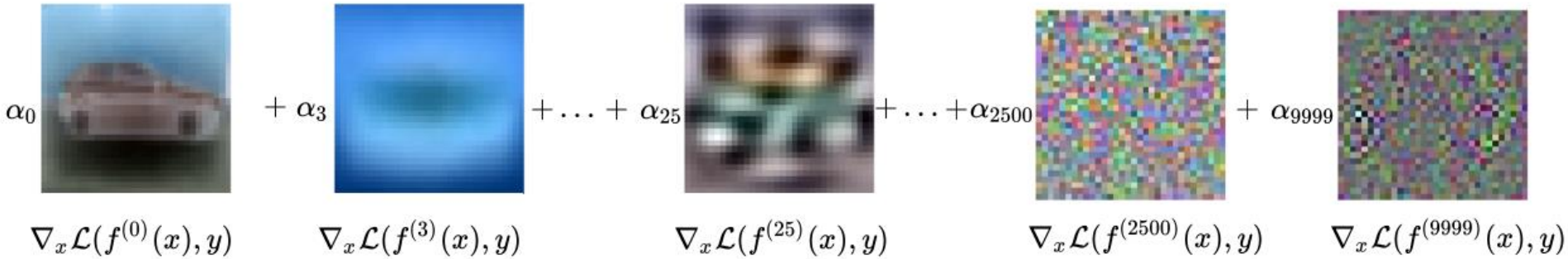


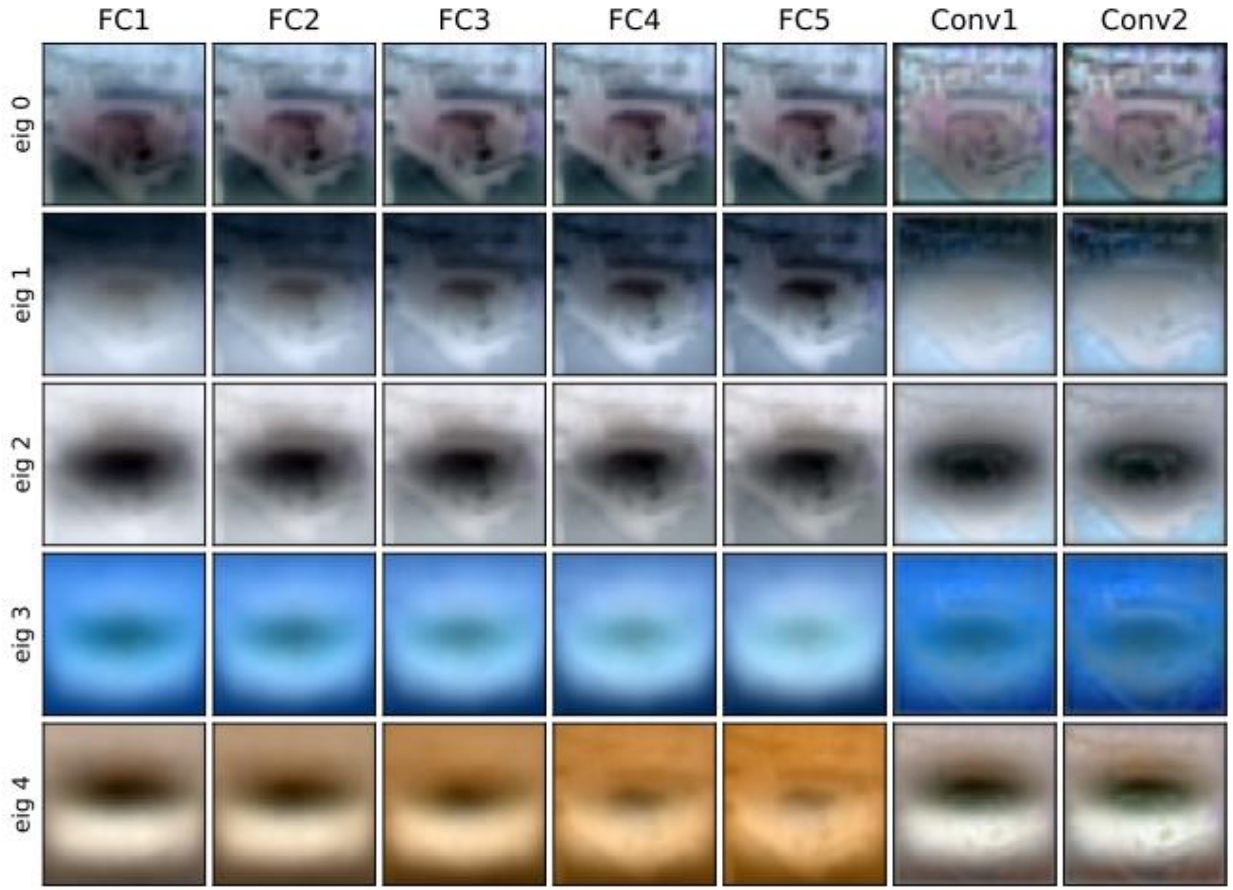
Figure 3. Visualization of some poisoned CIFAR-10 images (left) and their normalized perturbations (right).

“Visualizing features” [N. Tsilivis, JK’22]



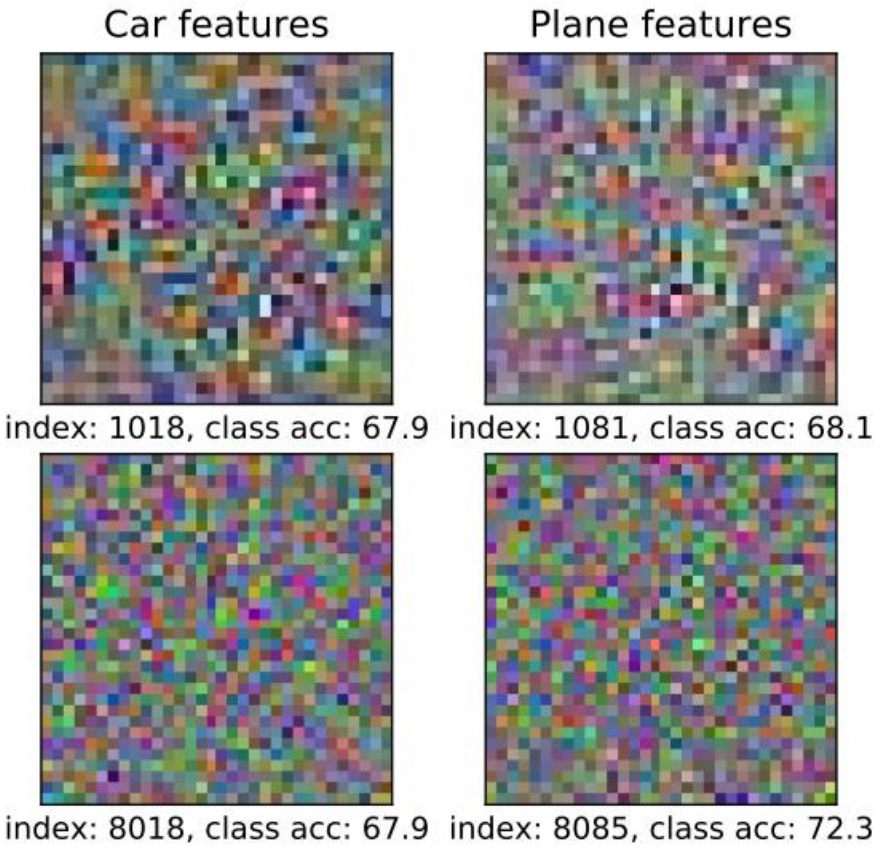


“Visualizing features” [N. Tsilivis, JK’22]



Top 5 features for 7 different kernel architectures for a car image from CIFAR trained on {car, plain} images

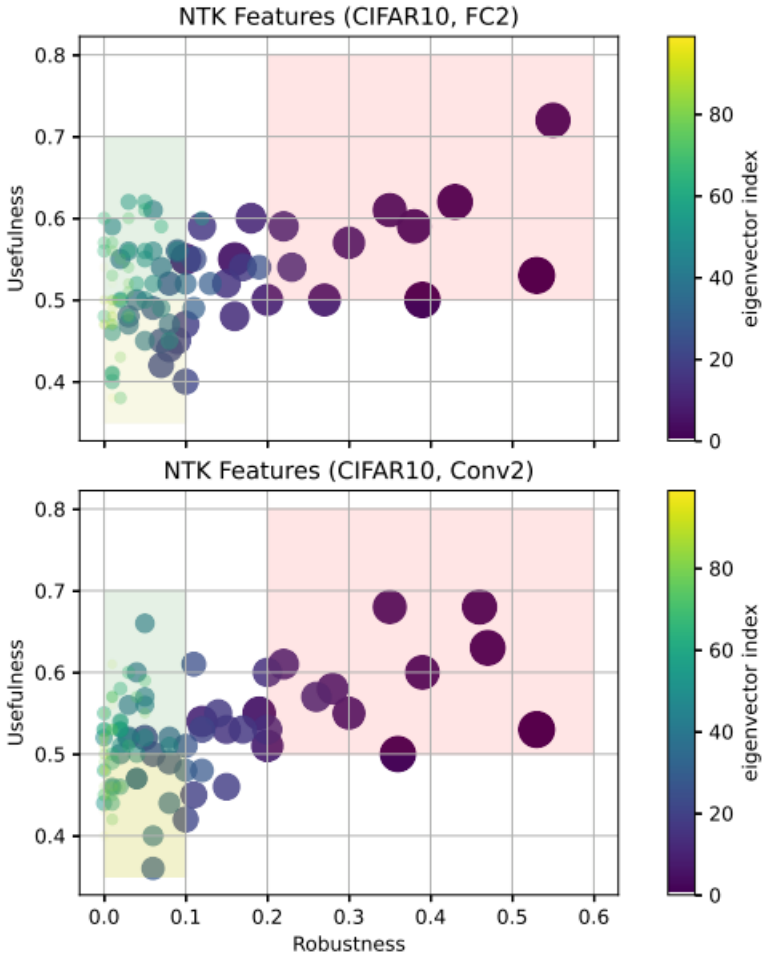
“Visualizing features” [N. Tsilivis, JK’22]



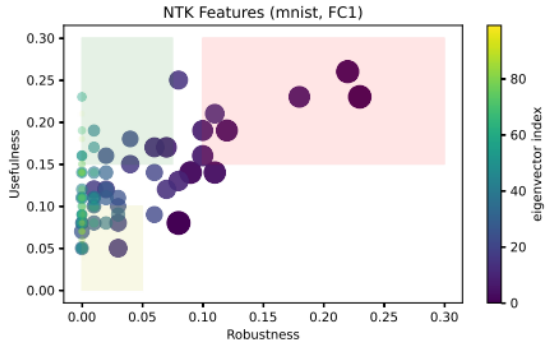
Non-robust, useful features earlier and later in the spectrum, CIFAR (car/pane)

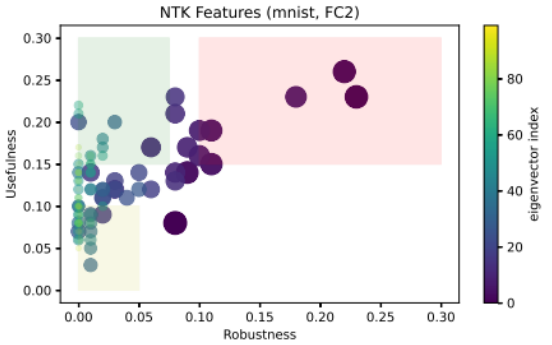
“Visualizing features” [N. Tsilivis, JK’22]

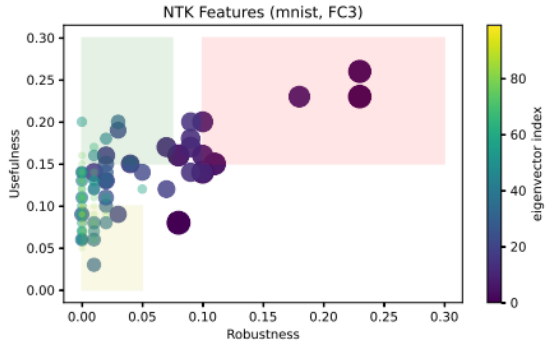
CIFAR (car/plane)

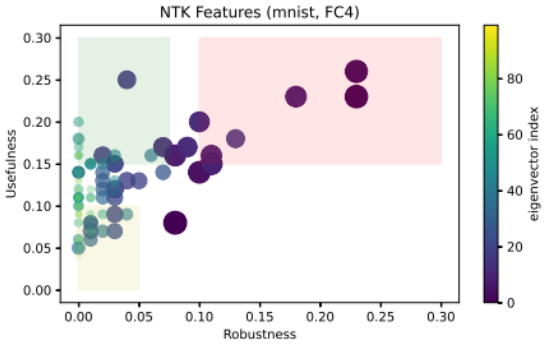


MNIST









Robustness – usefulness graphs

“Robustness lies at the top” [N. Tsilivis, JK’22]

- Illustrates Robustness-Accuracy tradeoff: useful, robust features are learned first, followed by use-ful non-robust ones
- Ties in well with studies showing that low frequency functions are fitted first and provide favorable generalization properties
- Robust features alone are not enough

NKT in practice?

NTK for Neural Architecture Search (<AutoML)

- NAS : automate the process of developing neural architectures for a given dataset
- First NAS techniques trained 1000s of architectures to completion: 1000s GPUs
- Search heuristics: RL based, Evolutionary algorithms, Bayesian Optimization, …
- Need to efficiently evaluate candidate architectures → surrogate metrics

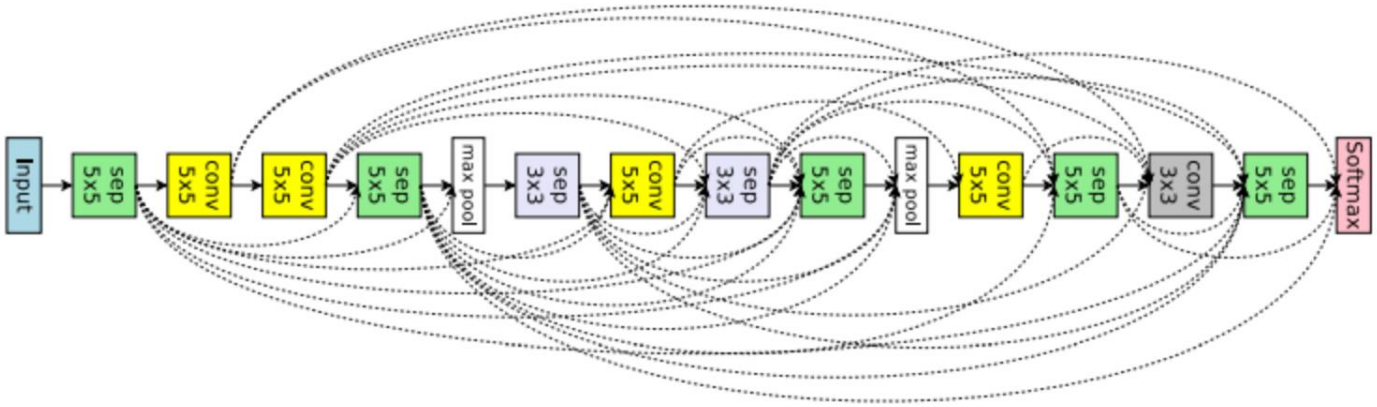
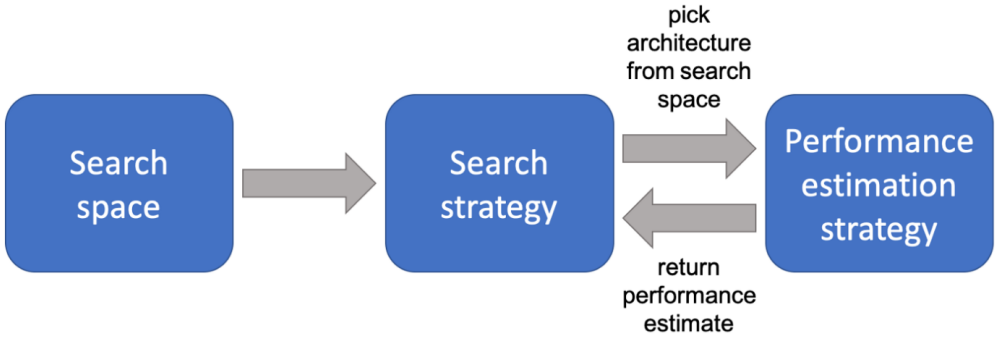


Figure 7. ENAS's discovered network from the macro search space for image classification.

Pham et al. ICML'18



T. Elsken, J.H. Metzen and F. Hutter, Neural Architecture Search: A Survey, Journal of Machine Learning Research, 2019

NTK for Neural Architecture Search (<AutoML)

- Validation requires training on large data, hyperparameter tuning, …
- A large number of surrogate metrics have been proposed for validation performance
- [Chen et al. ICML'21] "NEURAL ARCHITECTURE SEARCH ON IMAGENET IN FOUR GPU HOURS: A THEORETICALLY INSPIRED PERSPECTIVE":
  - Uses condition number κ of empirical NTK at initialization
- [Xu et al. ICML'21] "Knas: Green neural architecture search"
  - Uses Frobenius norm of NTK as a proxy for smallest eigenvalue (n^2 not n^3)
  - Sometimes use mean of NTK entries as proxy
- [Park, Lee et al. '20] "Towards NNGP-guided Neural Architecture Search"
  - Use NNGP and Monte Carlo Sampling to approximate training performance

NTK for Neural Architecture Search (<AutoML)

- [Mok et al. CVPR'22] "Demystifying the Neural Tangent Kernel from a Practical Perspective: Can it be trusted for Neural Architecture Search without training?"
  - Note that NTK changes significantly for small architectures
  - Propose to use LGA (Label Gradient Alignment) after 1,3 or 5 epochs
  - Hypothesize that trainable architectures align labels with gradients rapidly

LGA for model selection for fine tuning [Deshpande et al, ‘21]

- Which model to chose for fine tuning on idiosyncratic data?
- Use LGA on initial empirical NTK for a small sample of tuning data to decide
- Idea of why it works: The less fine tuning is required, the less the weights change and the closer the fine tuning is to linear

NTK for Pruning

- Pruning at inference [LeCun, Denker, Solla ‘90: “Optimal Brain Damage”] …
- Pruning during or before training? …
- Lottery Ticket Hypothesis: There exist sparse subnetworks that when trained perform as well as the original

- [Liu & Zenke ICML’20: “Finding Sparse Networks Through Neural Tangent Transfer”]
  - Use NTK at initialization as surrogate to align training trajectories
  - Minimize Frobenius distance between the two kernels during IMP
- [Yang, Wang ‘22]: study NTK under random pruning of weights w.p. 1-p → rescaling of weights by $1/\sqrt{p}$

Empirical studies of the empirical NTK

- [Fort et al. NeurIPS'20] "Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the Neural Tangent Kernel"
  - study the empirical kernel, linearize at different epochs during training, relate to the loss landscape
  - NTK "rotates" rapidly early in training, then "stabilizes"
  - In parallel, linearizing after a few epochs yields (nearly) full performance
  - …

Empirical studies of the empirical NTK

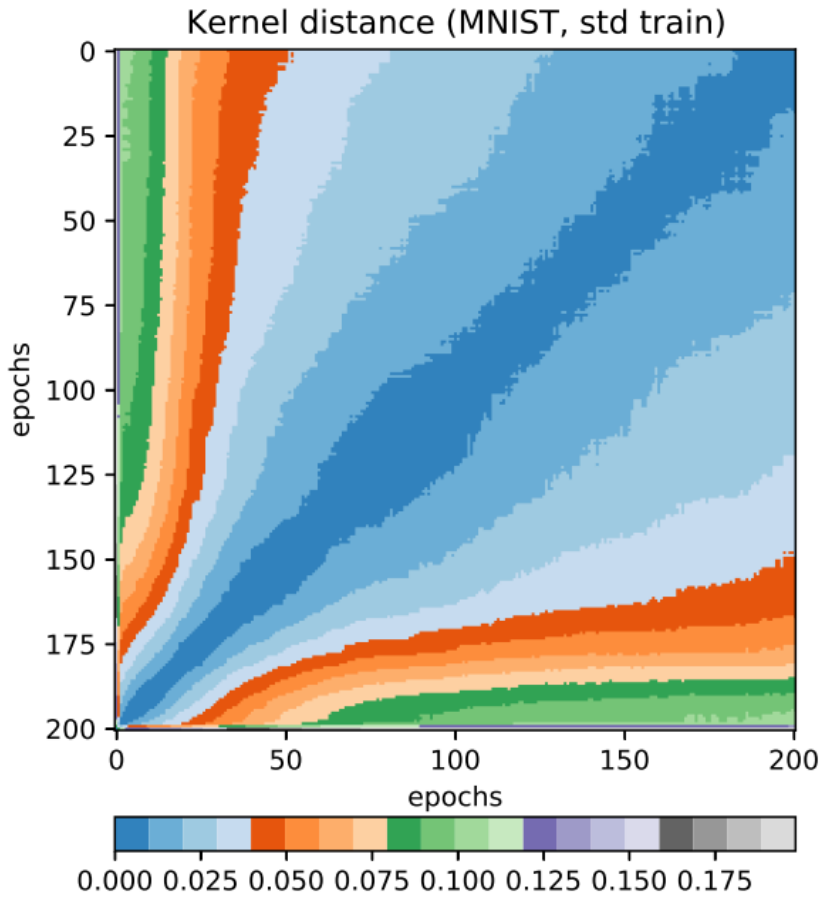
- [Ortiz-Jimenez et al. NeurIPS'21] "What can linearized neural networks actually say about generalization?"
  - Studies the "alignment" of kernel to labels, relates this to complexity and "inductive bias"
  - …

Empirical studies of the empirical NTK

- [Baratin et al. AISTATS'21] "Implicit Regularization via Neural Feature Alignment"
  - Studies the "alignment" of kernel to tasks, observe sharp increase in "anisotropy" (effective rank of the kernel decreases = "dominance" of top eigenvalues)
  - "Dynamic alignment" acts as implicit regularizer → heuristic complexity measure that correlates with generalization

Back to adversarial training (with “NKT lens” [Tsilivis, JK’22])

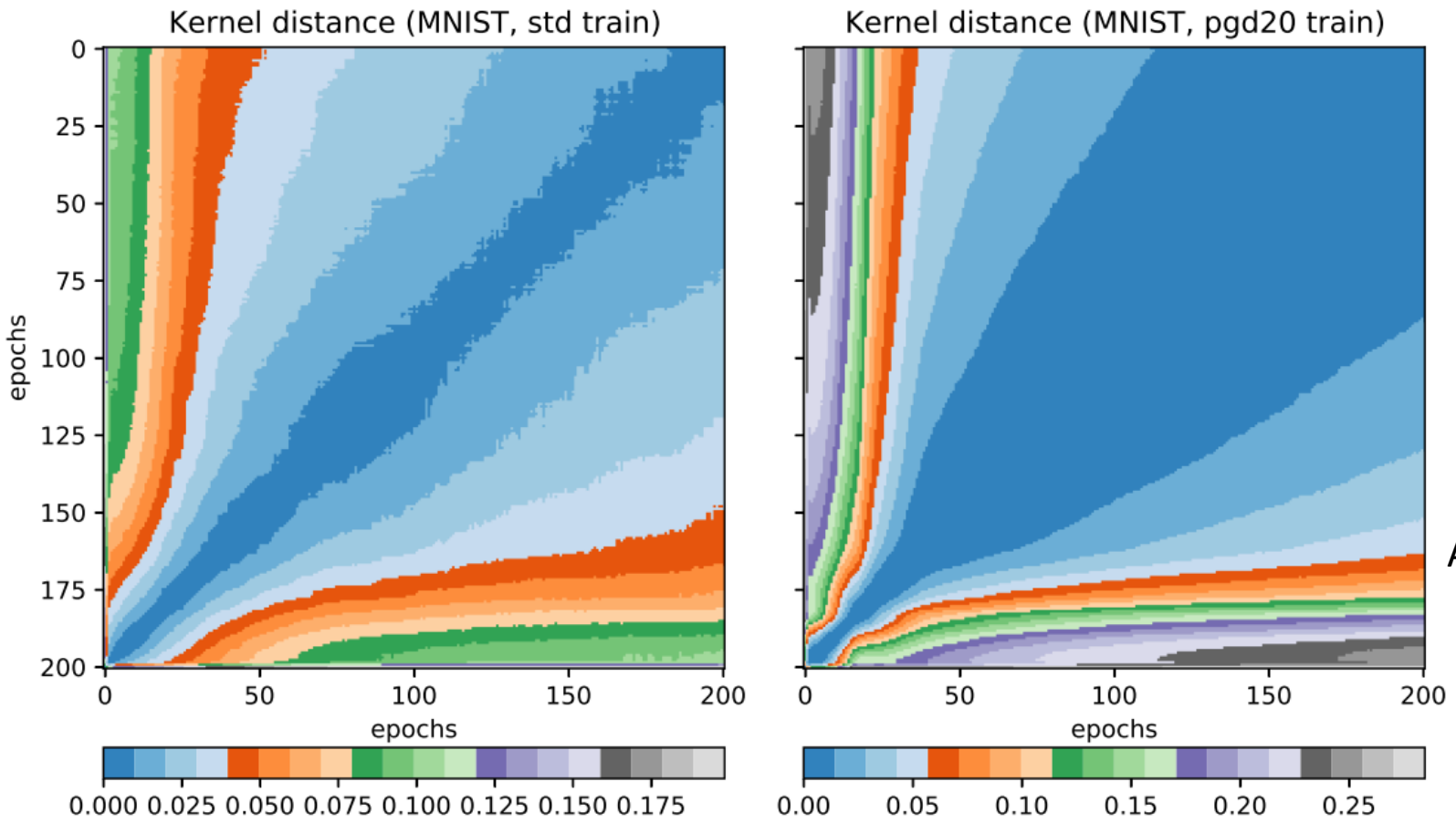
- Study kernel distance for standard and adversarial training a la [Fort et al. ‘20]



$$d(\Theta_i, \Theta_j) = 1 - \frac{\mathrm{Tr}(\Theta_i \Theta_j^\top)}{\sqrt{\mathrm{Tr}(\Theta_i \Theta_i^\top)}\sqrt{\mathrm{Tr}(\Theta_j \Theta_j^\top)}}$$

Back to adversarial training (with “NKT lens” [Tsilivis, JK’22])

- Study kernel distance for standard and adversarial training a la [Fort et al. ‘20]

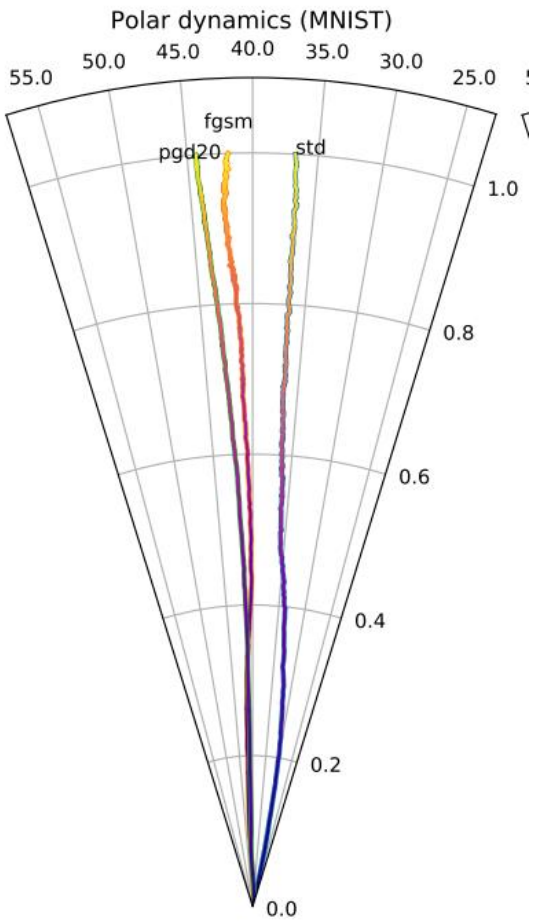


$$d(\Theta_i, \Theta_j) = 1 - \frac{\mathrm{Tr}(\Theta_i \Theta_j^\top)}{\sqrt{\mathrm{Tr}(\Theta_i \Theta_i^\top)}\sqrt{\mathrm{Tr}(\Theta_j \Theta_j^\top)}}$$

AT kernel becomes “lazy” much earlier

Back to adversarial training (with “NKT lens” [Tsilivis, JK’22])

- Study kernel rotation for standard and adversarial training



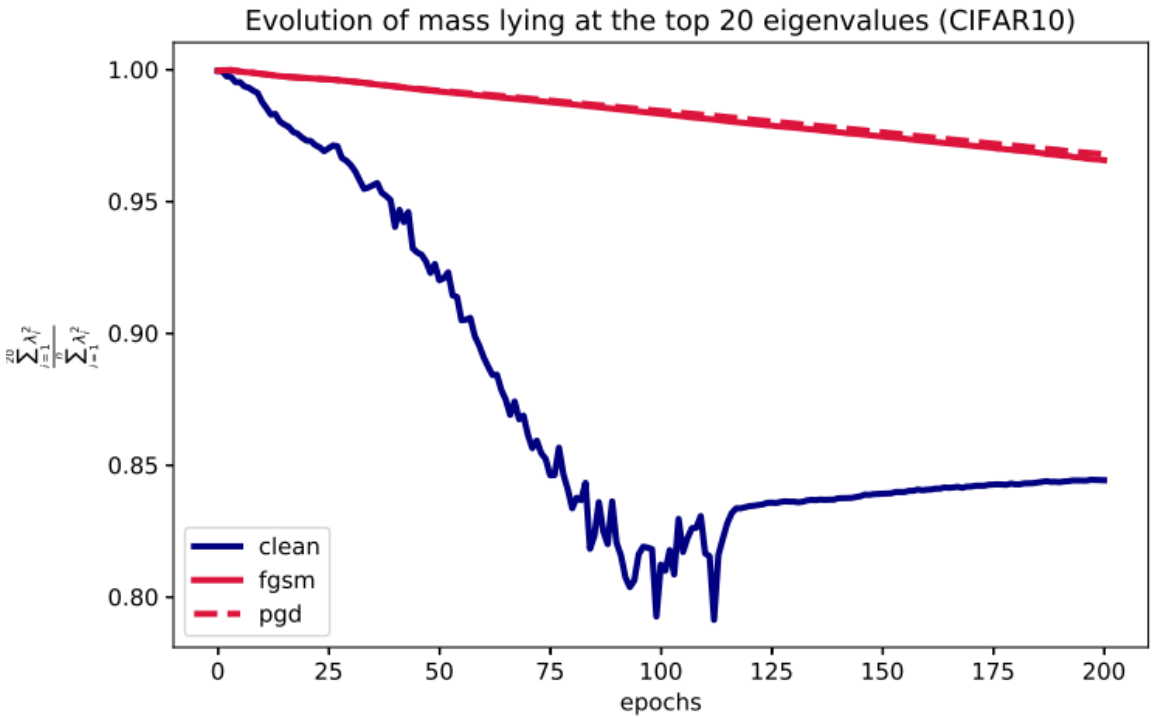
Polar coordinates:

$$r_t = \frac{\|\Theta_t - \Theta_0\|_F}{\|\Theta_f - \Theta_0\|_F}, \quad \theta_t = \arccos\left(1 - d(\Theta_t, \Theta_0)\right)$$

- STD kernel rotates early, then expands
- AT kernel rotates more, becomes “lazy” much earlier

Back to adversarial training (with “NKT lens” [Tsilivis, JK’22])

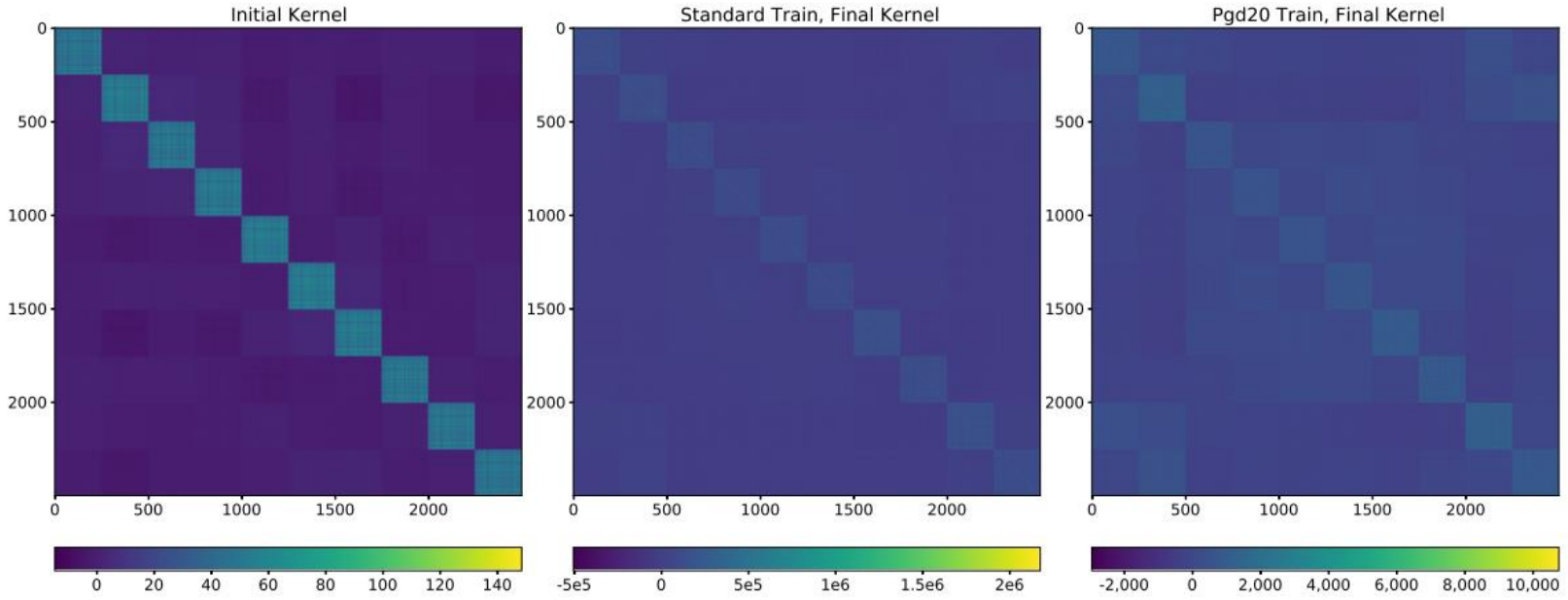
- Eigenvalue concentration for standard and adversarial training a la [Baratin et al. ‘21]

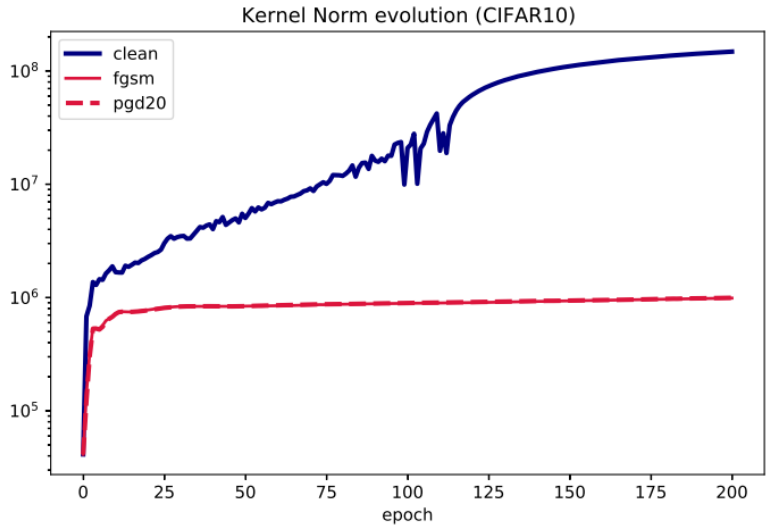


AT Kernel “learns to depend on the robust features at the top”

Back to adversarial training (with “NKT lens” [Tsilivis, JK’22])

- Kernel matrices and norms a la [Baratin et al. 21]

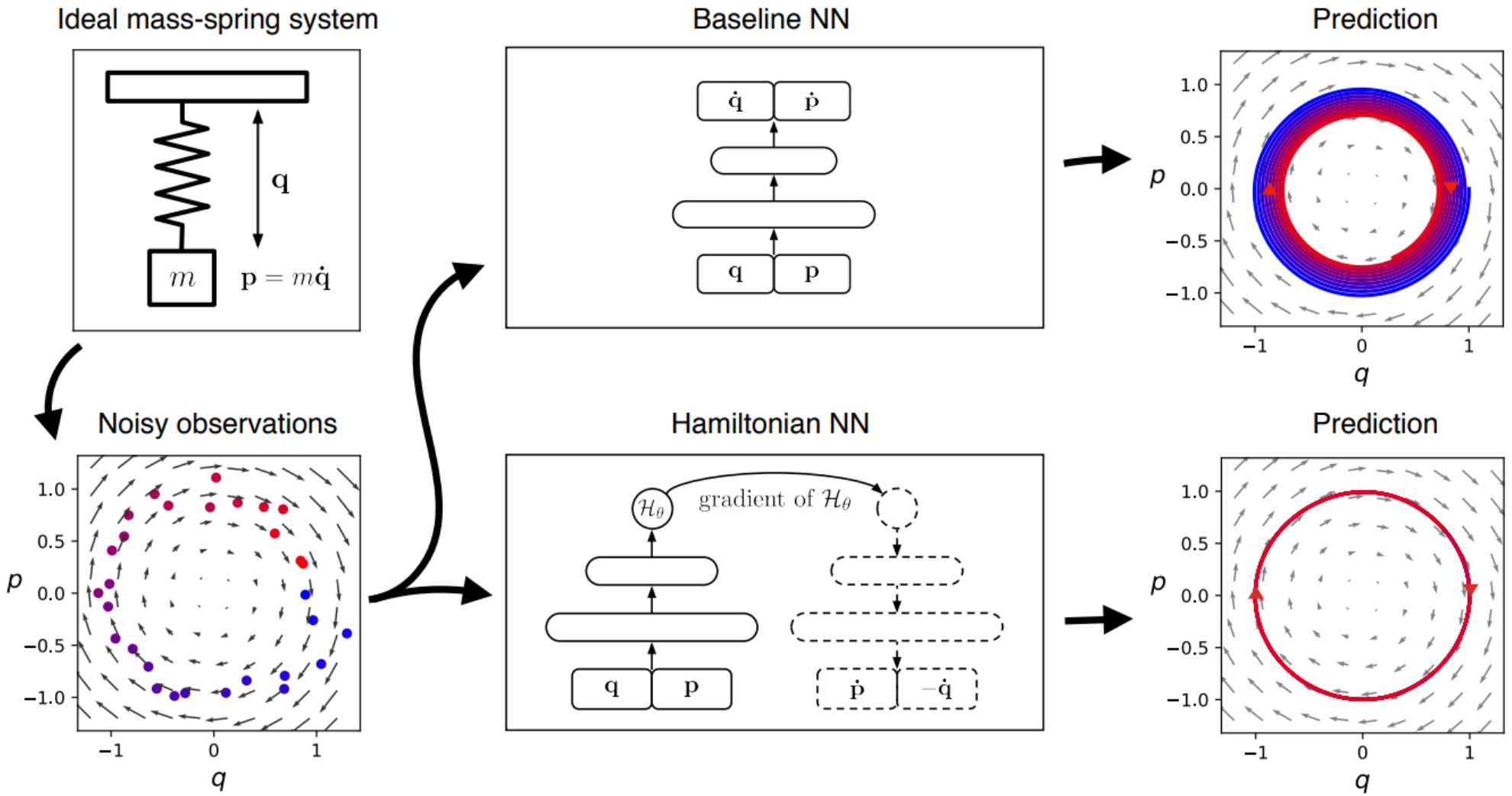




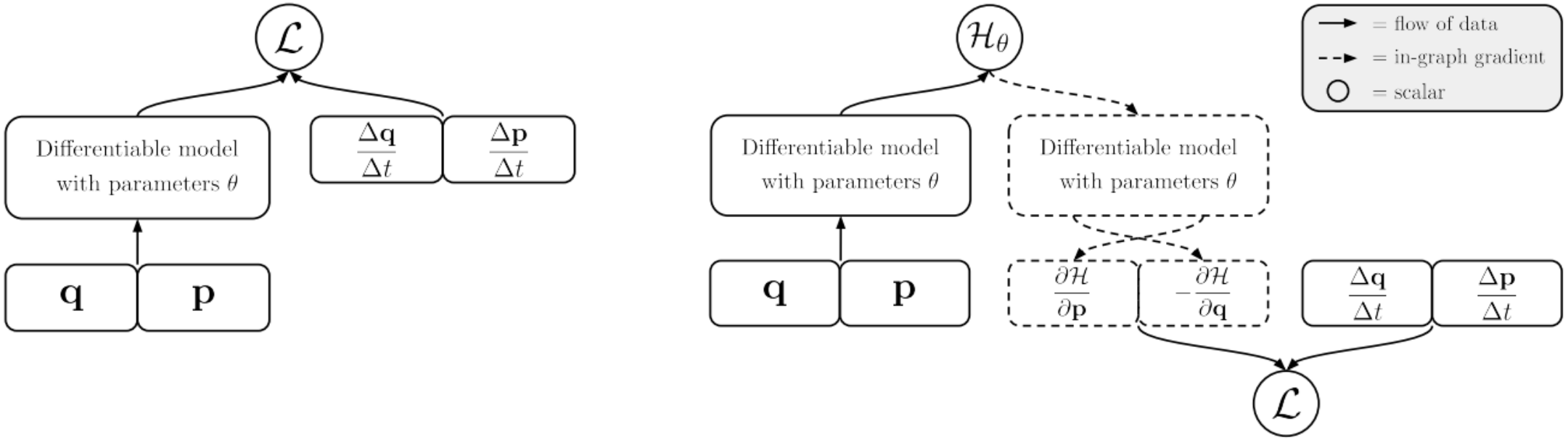
AT gives a more “conservative” kernel

Appendix (Material I didn’t get to)

Hamiltonian Neural Nets [Greydanus et al.'19]



Hamiltonian Neural Nets [Greydanus et al.'19]



(a) Baseline NN

(b) Hamiltonian NN

Hamiltonian Neural Nets [Greydanus et al.'19]