Deep networks training and generalization: insights from linearization

Thesis defense

Thomas George

Département d'Informatique et de Recherche Opérationnelle

April 20th, 2023

Deep networks

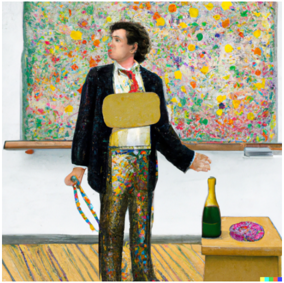


Figure: "A PhD student defending his thesis in the style of Gustav Klimt" generated by DALL·E

A range of applications

- Text-to-image generation
  DALL·E, Midjourney
- Language models
  ChatGPT
- Automation
  Self-driving cars, agriculture, industrial tools, medical tools, text translation...
- Science
  Prediction of protein folding structure
- Image classification
  first empirical success of deep learning, simple enough (easy training), yet exhibits interesting and unexplained properties

Parametric models

$$y = f_{\mathbf{w}}(x)$$

- Inspired by the structure of brains: a network of many simple computational units
- Learning: find correct values for parameters $\mathbf{w}$ = synapses weights

Learning with deep networks

Empirical risk minimization

Training dataset of examples $\mathcal{D}$, loss function ℓ

$$f_{\text{trained}} = \arg\min_{\mathbf{w}} L\left(\mathbf{w}, \ell, \mathcal{D}\right)$$

- Generalization to unseen examples
- Underspecification
- Analysis tools
- **Qualitative** and **quantitative** theoretical principles

Outline

Methodology and tools

1. NNGeometry (PyTorch Ecosystem Day 2021)

Train fast

2. EKFAC (NeurIPS 2018)

Insights into generalization properties

3. Lazy vs Hasty (TMLR 2022)
4. NTK Alignment (AISTATS 2021)

Proposed model

Outcome predictor f_{trained} is the result of an **iterative algorithm**:

$$f_{\mathbf{w}_T} = f_{\mathbf{w}_0} + \sum_{t=0}^{T-1} \underbrace{f_{\mathbf{w}_t + \delta \mathbf{w}_t} - f_{\mathbf{w}_t}}_{:= \delta f_t}$$

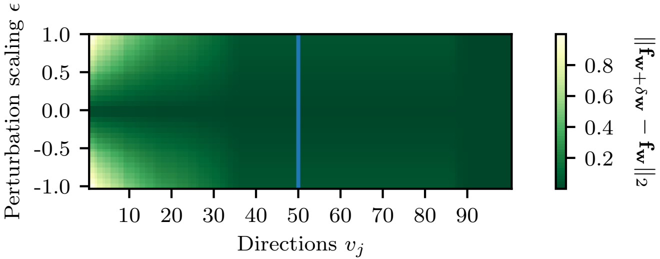
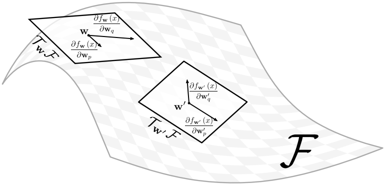


Figure: Effect on $f_{\mathbf{w}}$ of perturbations $\delta \mathbf{w} = \epsilon v_j$ on a partially trained VGG19 network on CIFAR10. **(left)** Top directions **(right)** directions are chosen at random.

$\Rightarrow$ Parameter space to function space mapping is very ill-conditioned

Proposed model: linearization



Locally to each iteration, **linearize** using 1st order Taylor expansion:

$$\delta f_t(x) \approx \langle \nabla_{\mathbf{w}} f_{\mathbf{w}_t}(x), \delta \mathbf{w}_t \rangle$$

Tangent Features	**Fisher Information Matrix**	**Neural Tangent Kernel** (Jacot et al., 2018)
$\phi_{\mathbf{w}_t}(x) := \nabla_{\mathbf{w}} f_{\mathbf{w}_t}(x)$	$F_{\mathbf{w}} = \mathbb{E}_x \left[\phi_{\mathbf{w}}(x) \phi_{\mathbf{w}}(x)^\top \right]$	$k_{\mathbf{w}}(x, x') = \langle \phi_{\mathbf{w}}(x), \phi_{\mathbf{w}}(x') \rangle$
$\mathbf{\Phi}_{\mathbf{w}} = \begin{pmatrix} \phi_{\mathbf{w}}(x_1)^\top \\ \vdots \\ \phi_{\mathbf{w}}(x_n)^\top \end{pmatrix}$	$\mathbf{F}_{\mathbf{w}} = \frac{1}{n} \mathbf{\Phi}_{\mathbf{w}}^\top \mathbf{\Phi}_{\mathbf{w}}$	$\mathbf{K}_{\mathbf{w}} = \mathbf{\Phi}_{\mathbf{w}} \mathbf{\Phi}_{\mathbf{w}}^\top$
$(n \times d)$	$(d \times d)$	$(n \times n)$

For a set of n examples, and a network with d parameters
$n \sim 10^4 - 10^{10}$ $d \sim 10^7 - 10^{14}$

Linearization toolbox: NNGeometry (George, 2021)

A PyTorch library for computing tangent features, FIMs, NTKs.

Example: compute a vector-Fisher-vector product $\mathbf{v}^\top F \mathbf{v}$:

using a KFAC Fisher

```
F_kfac = FIM(model=model,
             loader=loader,
             representation=PMatKFAC,
             n_output=10)

v = PVector.from_model(model)

vTMv = F_kfac.vTMv(v)
```

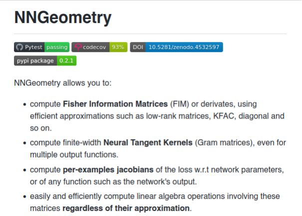
using implicit computation

```
F_full = FIM(model=model,
             loader=loader,
             representation=PMatImplicit,
             n_output=10)

v = PVector.from_model(model)

vTMv = F_full.vTMv(v)
```

- implicit operations
- approximate objects
  KFAC (Martens and Grosse, 2015), EKFAC (George et al., 2018), Quasi-diagonal (Ollivier, 2015), ...



github.com/tfjgeorge/nngeometry

Fast Approximate Natural Gradient Descent in a Kronecker-factored Eigenbasis

NeurIPS 2018

TG*, César Laurent*, Xavier Bouthillier, Nicolas Ballas, Pascal Vincent

Motivation

Train fast

- Ill-conditioning
- Millions of parameters to be optimized
  cure can be worse than disease (e.g. large improvement but slow iterations)

Usual (Euclidean) gradient descent

Iterate steps in steepest* descent direction
* as measured by the Euclidean norm $\|d\mathbf{w}\|^2 = \langle d\mathbf{w}, d\mathbf{w}\rangle$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla \hat{L}$$

Natural gradient descent (Amari, 1998)

Iterate steps in steepest* descent direction
* as measured by the Fisher norm $\|d\mathbf{w}\|^2_{F_\mathbf{w}} = \langle d\mathbf{w}, F_\mathbf{w} d\mathbf{w}\rangle$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta F_\mathbf{w}^{-1} \nabla \hat{L}$$

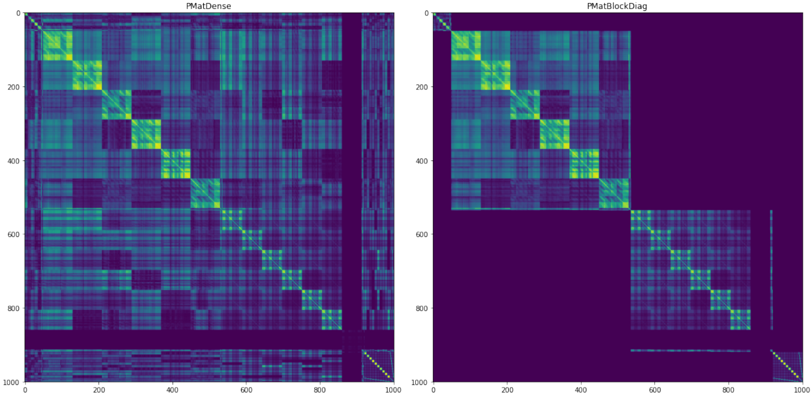
- mitigates ill-conditioning of the tangent features
- closely resembles Newton's method

but...

- $F_\mathbf{w}$ is a $d \times d$ matrix, $d \sim 10^7 - 10^{14}$
- memory cost $\mathcal{O}\left(d^2\right)$, computational cost $\mathcal{O}\left(d^3\right)$

Fisher Information Matrix: structure

Figure: Fisher Information Matrix of a small depth-4 convolutional network on MNIST



Dense matrix $\Rightarrow$ Block diagonal matrix

Fisher Information Matrix: KFAC (Martens and Grosse, 2015)

For a linear layer...

$$g = \underset{(k \times m)}{\mathbf{W}} a$$

we have

$$F_{\text{layer}} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} a a^\top \otimes \nabla_g \nabla_g^\top$$

$$F_{\text{layer}} \approx \underbrace{\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} a a^\top}_{:=A \quad (m \times m)} \otimes \underbrace{\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \nabla_g \nabla_g^\top}_{:=G \quad (k \times k)} \text{ (Martens and Grosse, 2015)}$$

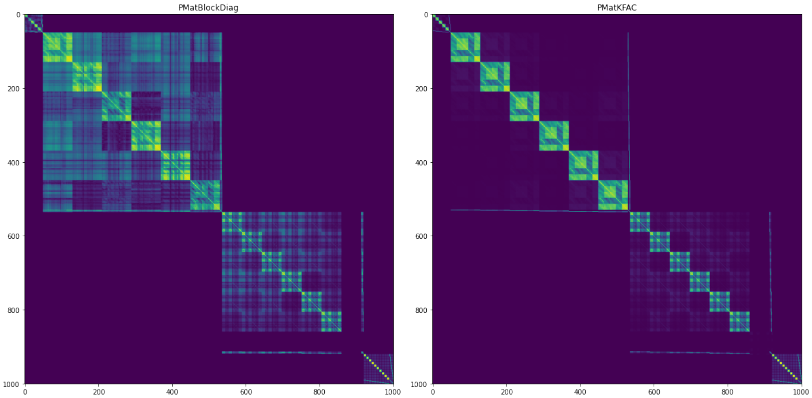
Property of Kronecker products:

$$\underset{(mk \times mk)}{(A \otimes G)^{-1}} = \underset{(m \times m)}{A^{-1}} \otimes \underset{(k \times k)}{G^{-1}}$$

Cost goes from $\mathcal{O}\left((mk)^3\right)$ to $\mathcal{O}\left(m^3\right) + \mathcal{O}\left(k^3\right)$

Fisher Information Matrix: structure

Figure: Fisher Information Matrix of a small depth-4 convolutional network on MNIST



Block diagonal matrix ⇒ KFAC matrix

Kronecker-factored eigenbasis

$$F_{\text{KFAC}} = A \otimes G$$

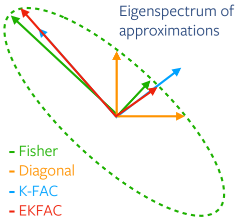
SVD of A and G:

$$A = U_A \Lambda_A U_A^\top \qquad G = U_G \Lambda_G U_G^\top$$

$$F_{\text{KFAC}} = \underbrace{(U_A \otimes U_G)}_{:=U_{\text{KFE}}} (\Lambda_A \otimes \Lambda_G) (U_A \otimes U_G)^\top$$

- A cheap basis that (approx.) diagonalizes F_{layer}
- The exact diagonal in the KFE costs $\mathcal{O}\left((m+k)\,|\mathcal{D}|\right)$

$$S = \text{vec}\left(\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \left(U_A^\top a \left(U_G^\top \nabla_y\right)^\top\right)^2\right)$$



Theorem

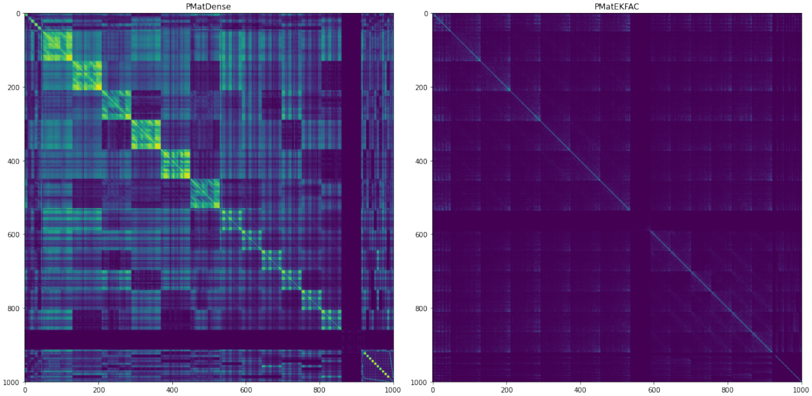
S is the optimal diagonal scaling in the KFE (in the sense of the Frobenius norm)

Corollary

EKFAC better approximates the FIM than KFAC

Fisher Information Matrix: structure

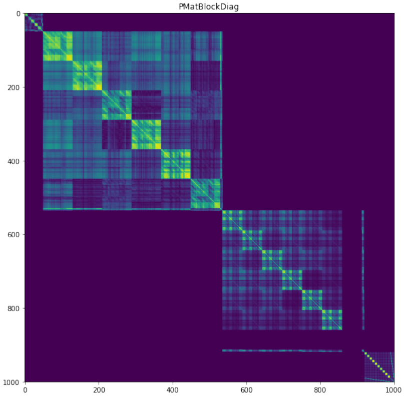
Figure: Fisher Information Matrix of a small depth-4 convolutional network on MNIST



Dense matrix ⇒ Same matrix projected in KFE

Fisher Information Matrix: structure

Figure: Fisher Information Matrix of a small depth-4 convolutional network on MNIST



Block diagonal matrix ⇒ EKFAC matrix

Training curves

Algorithm and implementation

- (amortized every T iterations) re-evaluate the KFE
- (at every iteration)
  1. project everything in the KFE
  2. perform a diagonal method in the KFE
  3. project back in parameter space (=update)

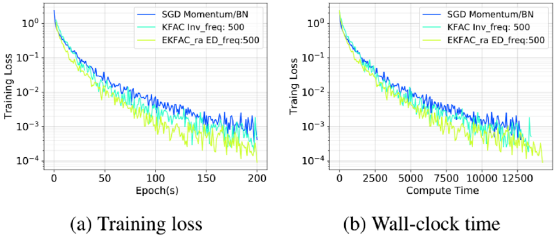


Figure: Training curves of a Resnet34 on CIFAR10 using different optimizers

EKFAC summary

- Approximate of the Fisher Information Matrix
  - Optimization
  - Pruning (Wang et al., 2019)
  - Continual learning (Liu et al., 2018)
- Accelerates training (per iteration/per wall-clock time)

Lazy vs hasty: linearization in deep networks impacts learning schedule based on example difficulty

TMLR 2022



TG, Guillaume Lajoie, Aristide Baratin

Motivation

Generalization properties of f_{trained}

- Compare linear (lazy) regime (with fixed tangent features $\phi_{\mathbf{w}_0}$) to full regime
  model for which we have theoretical results, vs full training trajectory
- Which examples are used to learn?
  Typical examples vs corner cases

Outline

1. Empirical results with different ways of measuring example *difficulty*
2. Analytical insights on a simplified tractable model

Setup

Use scalar coefficient α to modulate non-linearity (Chizat et al., 2019)

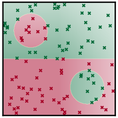
$$f_{\alpha,\mathbf{w}} = \alpha \left(f_{\mathbf{w}} - f_{\mathbf{w}_0} \right) \text{ and learning rate } \eta_\alpha = \frac{\eta}{\alpha^2}$$

- $\alpha \gg 1$: linearized regime
- $\alpha = 1$: standard regime
- $\alpha < 1$: "super adaptive" regime

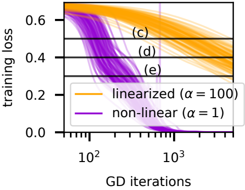
Monitor "non-linearity": NTK alignment, Repr. kernel alignment, sign similarity

Impact of training regime: a 2d toy dataset

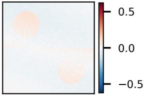
Figure: Yinyang dataset, 100 linear runs, 100 non-linear runs, 4 layers MLP



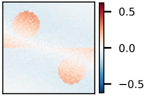
(a) task + example dataset



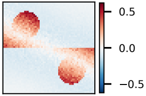
Normalize training progress



(c) $\Delta loss(x_{test})$ at training loss=0.5



(d) $\Delta loss(x_{test})$ at training loss=0.4



(e) $\Delta loss(x_{test})$ at training loss=0.3

$$\Delta\text{loss}\,(\cdot) = \text{loss}\, f_{\textbf{non-linear}}\,(\cdot) - \text{loss}\, f_{\textbf{linear}}\,(\cdot)$$

- (red) linear is better
- (blue) non-linear is better

⇒ Non-linear is better at learning larger areas of the same class

Impact of training regime: examples grouped by C-scores

C-score (Jiang et al., 2021)

$$C_{\mathcal{D},n}(x,y) = \mathbb{E}_{D \overset{n}{\sim} \mathcal{D}\setminus\{(x,y)\}}\left[\mathbb{P}\left(f_{\textbf{trained}}(x;D) = y\right)\right],$$

Likeliness that example x is correctly classified if it were not included in the training set

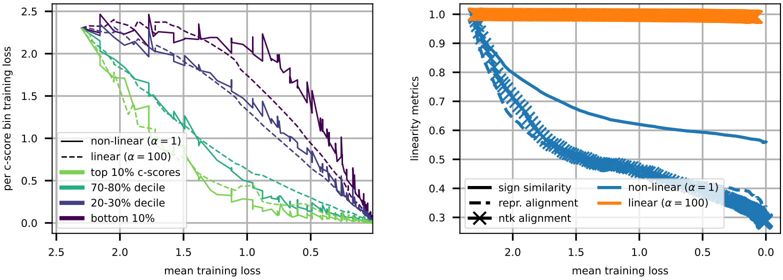


Figure: ResNet18 on CIFAR10. The training loss is monitored separately on groups of examples ranked by their C-scores.

⇒ Non-linear learns examples with high C-score faster

Impact of training regime: corrupted examples

Noisy labels

Part of the training examples are assigned a wrong label

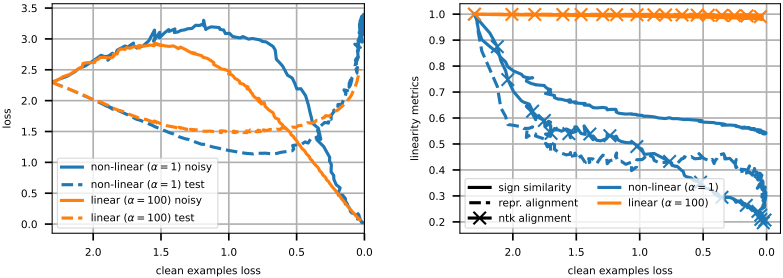


Figure: ResNet18 on CIFAR10. The training loss is monitored separately on clean and noisy examples.

⇒ Non-linear spends greater part ignoring noisy examples

Impact of training regime: spurious features

Spurious feature

A feature that is uninformative on the task, but easier to learn

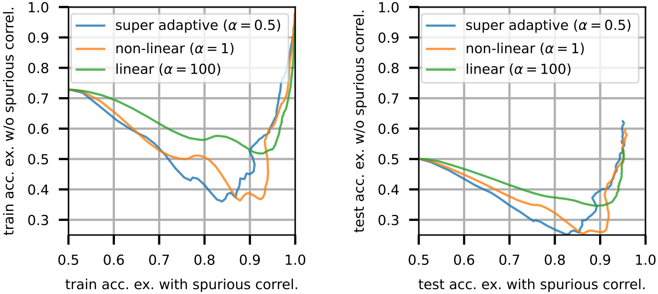


Figure: ResNet18 on Waterbirds. The training accuracy is monitored separately for examples that include the spurious feature, and examples without the spurious feature.

⇒ Non-linear is more prone to learning spurious correlations

Analytical insights

Setup

Matrix of inputs $\mathbf{X}$ $(n \times d)$, with labels $\mathbf{y}$ (n).
We minimize the MSE of the linear model $f_\theta(x) = \theta^\top x$:

1. **(linearly)** directly with parameters θ
2. **(non-linearly)** with quadratic parameterization $\theta := \frac{1}{2}\sum_{\lambda=1}^{d} w_\lambda^2 \mathbf{v}_\lambda$

$$\mathbf{X} = \mathbf{U}\mathbf{M}\mathbf{V}^\top := \sum_{\lambda=1}^{r_X} \sqrt{\mu_\lambda}\,\mathbf{u}_\lambda \mathbf{v}_\lambda^\top$$

Solution (gradient flow)

Linear training regime

$$\theta_\lambda(t) = \theta_\lambda^* + e^{-\mu_\lambda \theta_\lambda^0 t}(\theta_\lambda^0 - \theta_\lambda^*)$$

Non-linear training regime

$$\theta_\lambda(t) = \theta_\lambda^* + \frac{\theta_\lambda^*}{\left(e^{2\tilde{y}_\lambda t} - 1\right)\theta_\lambda^0 + \theta_\lambda^*}(\theta_\lambda^0 - \theta_\lambda^*)$$

Analytical insights

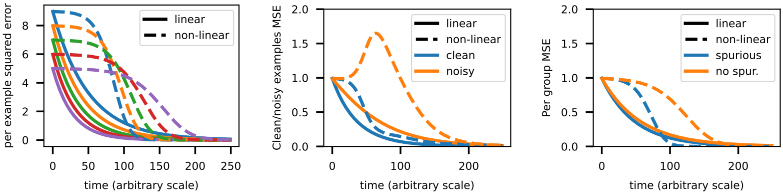


Figure: **(left)** Different input/label correlation (mirrors C-scores exp.) **(middle)** Label noise (mirrors exp. with noisy labels) **(right)** Spurious correlations (mirrors spurious features exp.)

Lazy vs hasty summary

- Embedded curriculum learning mechanism
- Depending on the task, can be beneficial or detrimental
- Quadratic analytical model that mirrors empirical observations

Implicit Regularization via Neural Feature Alignment

AISTATS 2021

Aristide Baratin*, **TG***, César Laurent, R Devon Hjelm, Guillaume Lajoie, Pascal Vincent, Simon Lacoste-Julien

Motivation

Generalization properties of f_{trained}

- implicit regularization mechanism
  e.g. previous paper
- generalization bounds
  non-vacuous, that actually capture empirical observations

Recall

- tangent features are ill-conditioned
  feature importance
- compared to linear training regime, full training dynamics amplify the implicit bias towards easy examples

This work: shed light on a dynamical effect on the tangent features?

Outline

1. Empirical study of the tangent features during training
2. Complexity measure

Tangent features: dynamical spectral bias

Question

How does the conditioning of the tangent features $\phi_{\mathbf{w}}$ evolve during training?

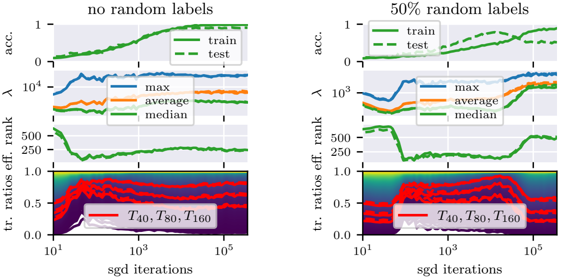


Figure: NTK **eigenvalues** (max, mean and median), **effective rank** and **trace ratios** during training of a VGG19 network on CIFAR10.

⇒ Dynamical stretching along a few directions as training progresses

Tangent features alignment

Question

Do tangent features *adapt* to the task?

Definition

Centered Kernel Alignment (Cortes et al., 2012)

$$\text{CKA}\left(\mathbf{K},\mathbf{K}'\right) = \frac{\text{Tr}\left(\mathbf{K}_c\mathbf{K}'_c\right)}{\|\mathbf{K}_c\|_F \|\mathbf{K}'_c\|_F}$$

$\mathbf{K}_c = C\mathbf{K}C$ is the centered kernel using $C = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$

- measures similarity between kernels
- similarity kernel/task when applied to the rank-one **target kernel** $\mathbf{K}_\mathbf{y} := \mathbf{y}\mathbf{y}^\top$

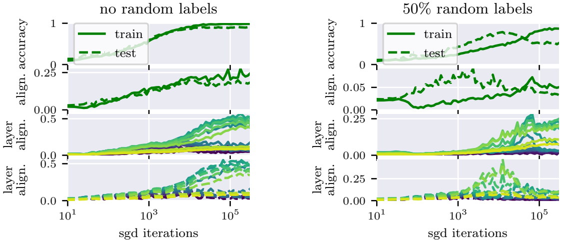


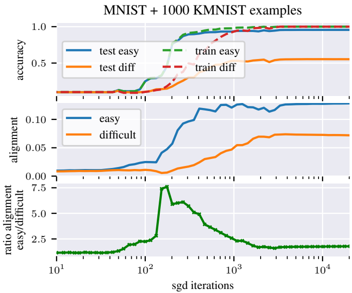
Figure: **Alignment** of **tangent features** to the **target kernel** while training a VGG19 architecture on CIFAR10.

$\Rightarrow$ The NTK increasingly aligns to the task as training progresses.

Tangent features: dynamical spectral bias

Question

Do tangent features *adapt* to the task?



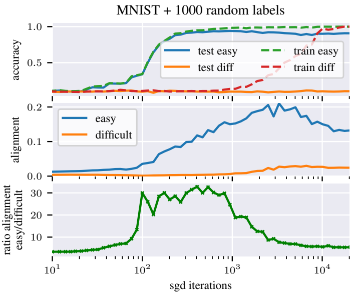


Figure: Alignment to *easy* and *difficult* examples. **(left)** 10.000 (easy) MNIST examples + 1.000 (difficult) kMNIST examples **(right)** 10.000 MNIST examples + 1.000 examples with randomly flipped labels

⇒ Faster alignment towards easier features.

Generalization guarantees

Uniform bounds: Rademacher complexity

with probability $\geq 1-\delta$, for functions $f \in \mathcal{F}$

$$\text{Test error}(f) \leq \text{Training error}(f) + \text{Complexity}(\mathcal{F}) + \sqrt{\frac{\log 1/\delta}{n}}$$

M-ball linear model (standard)

$$\mathcal{F}_M = \{f_{\mathbf{w}} = \langle \phi, \mathbf{w} \rangle : \|\mathbf{w}\| \leq M\}$$

$$\mathcal{R}_{\mathcal{S}}\left(\mathcal{F}_M\right) \leq (M/n)\,\|\mathbf{\Phi}\|_F$$

M-ellipsis linear model

Given any invertible matrix $\mathbf{A}$,

$$\mathcal{F}_M = \left\{f_{\mathbf{w}} = \langle \phi, \mathbf{w} \rangle : \left\|\mathbf{A}^{-1}\mathbf{w}\right\| \leq M\right\}$$

$$\mathcal{R}_{\mathcal{S}}\left(\mathcal{F}_M\right) \leq (M/n)\,\|\mathbf{\Phi}\mathbf{A}\|_F$$

Sum of linear steps using sequence of feature maps $\{\phi_t\}$

$$\mathcal{F}_{\mathbf{m}}^{\{\phi_t\}} = \left\{f_{\mathbf{w}} = \sum_t \langle \phi_t, \delta\mathbf{w}_t \rangle : \|\delta\mathbf{w}_t\| \leq m_t\right\}$$

$$\mathcal{R}_{\mathcal{S}}\left(\mathcal{F}_{\mathbf{m}}^{\{\phi_t\}}\right) \leq \sum_t (m_t/n)\,\|\mathbf{\Phi}_t\|_F$$

Complexity measure: empirical evaluation

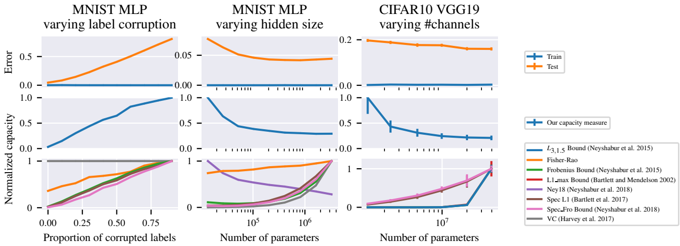


Figure: Complexity measure and test error as we vary **(left)** the proportion of corrupted labels, **(middle and right)** the number of parameters

$\Rightarrow$ Our complexity measure correlates with the test error.

NTK alignment summary

- Dynamical stretching and rotation of the NTK
- Implicit bias towards a few *relevant* directions
- Rademacher complexity measure that correlates with test error

Closing words

Linearization...

- EKFAC optimization algorithm
- Comparison between linear and non-linear training regimes
- Evolution of the NTK during training

Summary of contributions

- EKFAC approximate Fisher Information Matrix
  1. New method
  2. Optimality results
  3. Empirical evaluation of optimization algorithm
- Lazy vs Hasty
  1. Showed non-linear regime learns easy examples even faster than lazy regime (embedded curriculum learning mechanism)
  2. Consistent picture across 4 different easy/difficult setups
  3. Analytical model that mirrors experiments
- NTK dynamical alignment
  1. Showed empirical stretching and alignment to a few *relevant* directions
  2. Conjecture that it acts as an implicit regularization mechanism
  3. New Rademacher complexity measure
- NNGeometry (software library)
  1. Toolbox with unified API
  2. Scales to "large" networks with implicit operations

Questions

References I

Amari, Shun-Ichi (1998). “Natural gradient works efficiently in learning”. In: *Neural computation* 10.2, pp. 251–276.

Baratin, Aristide et al. (2020). *Implicit Regularization via Neural Feature Alignment*. arXiv: 2008.00938 [cs.LG].

Baratin, Aristide et al. (2021). “Implicit regularization via neural feature alignment”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 2269–2277.

Chizat, Lénaïc, Edouard Oyallon, and Francis Bach (2019). “On Lazy Training in Differentiable Programming”. In: *Advances in Neural Information Processing Systems* 32, pp. 2937–2947.

Cortes, Corinna, Mehryar Mohri, and Afshin Rostamizadeh (2012). “Algorithms for Learning Kernels Based on Centered Alignment”. In: *JMLR* 13.1, pp. 795–828.

George, Thomas (Feb. 2021). *NNGeometry: Easy and Fast Fisher Information Matrices and Neural Tangent Kernels in PyTorch*. Version v0.2.1.

George, Thomas, Guillaume Lajoie, and Aristide Baratin (2022). “Lazy vs hasty: linearization in deep networks impacts learning schedule based on example difficulty”. In: *arXiv preprint arXiv:2209.09658*.

George, Thomas et al. (2018). “Fast approximate natural gradient descent in a kronecker factored eigenbasis”. In: *Advances in Neural Information Processing Systems* 31.

References II

Ghorbani, Behrooz et al. (2020). "When do neural networks outperform kernel methods?" In: *arXiv preprint arXiv:2006.13409*.

Jacot, Arthur, Franck Gabriel, and Clément Hongler (2018). "Neural tangent kernel: Convergence and generalization in neural networks". In: *arXiv preprint arXiv:1806.07572*.

Liu, Xialei et al. (2018). "Rotate your networks: Better weight consolidation and less catastrophic forgetting". In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 2262–2268.

Martens, James and Roger Grosse (2015). "Optimizing neural networks with kronecker-factored approximate curvature". In: *International conference on machine learning*. PMLR, pp. 2408–2417.

Wang, Chaoqi et al. (2019). "Eigendamage: Structured pruning in the kronecker-factored eigenbasis". In: *International conference on machine learning*. PMLR, pp. 6566–6575.