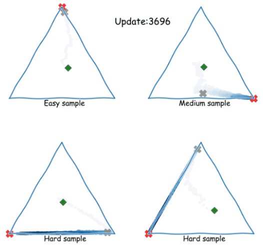
Local Learning Dynamics
Help Explain (Post-)Training Behaviour

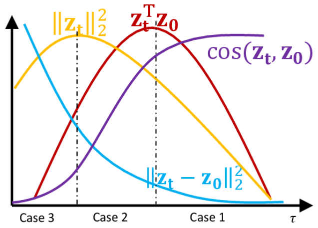
Danica J. Sutherland (she)
University of British Columbia (UBC) / Alberta Machine Intelligence Institute (Amii)

Knowl. dist. analysis
[ICLR-22]



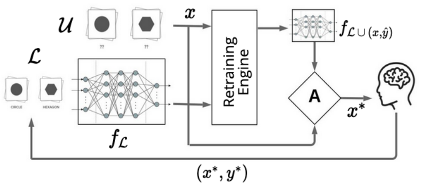
Yi Ren
Shangmin Guo

Finetuning analysis
[ICLR-23]



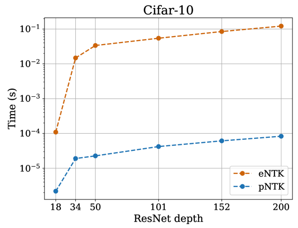
Yi Ren
Shangmin Guo
Wonho Bae

Active learning
[NeurIPS-22]



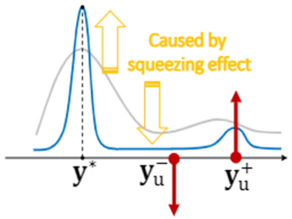
M. Amin Mohamadi
Wonho Bae

Pseudo-NTK
[ICML-23]



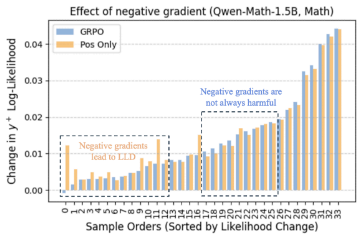
M. Amin Mohamadi
Wonho Bae

DPO/etc analysis
[ICLR-25]



Yi Ren

GRPO analysis+fix
[arXiv]



Wenlong Deng
Yi Ren
Muchen Li
Xiaoxiao Li
Christos Thrampoulidis

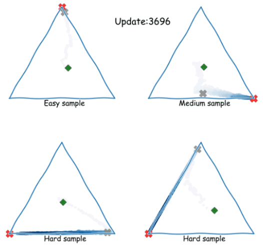
Mila – June 2025

HTML version

Local Learning Dynamics
Help Explain (Post-)Training Behaviour

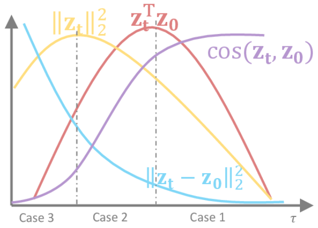
Danica J. Sutherland (she)
University of British Columbia (UBC) / Alberta Machine Intelligence Institute (Amii)

Knowl. dist. analysis
[ICLR-22]



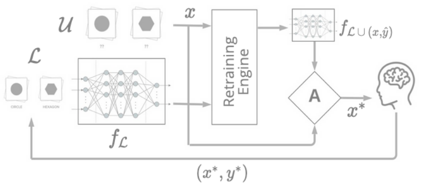
Yi Ren
Shangmin Guo

Finetuning analysis
[ICLR-23]



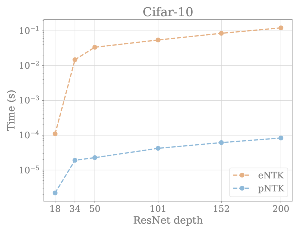
Yi Ren
Shangmin Guo
Wonho Bae

Active learning
[NeurIPS-22]



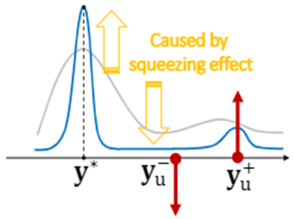
M. Amin Mohamadi
Wonho Bae

Pseudo-NTK
[ICML-23]



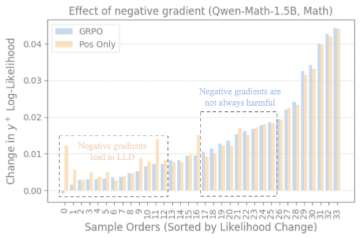
M. Amin Mohamadi
Wonho Bae

DPO/etc analysis
[ICLR-25]



Yi Ren

GRPO analysis+fix
[arXiv]



Wenlong Deng
Yi Ren
Muchen Li
Xiaoxiao Li
Christos Thrampoulidis

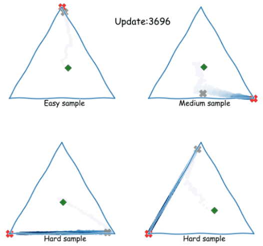
Mila – June 2025

HTML version

Local Learning Dynamics
Help Explain (Post-)Training Behaviour

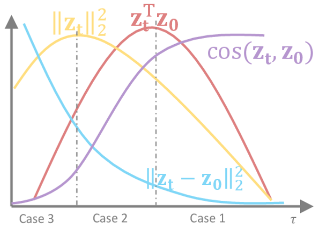
Danica J. Sutherland (she)
University of British Columbia (UBC) / Alberta Machine Intelligence Institute (Amii)

Knowl. dist. analysis
[ICLR-22]



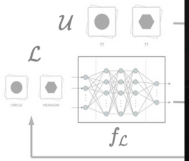
Yi Ren
Shangmin Guo

Finetuning analysis
[ICLR-23]



Yi Ren
Shangmin Guo
Wonho Bae

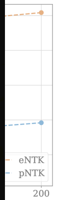
Active
[Neu



M. Amin
Won

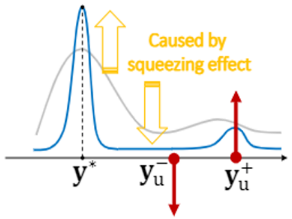
Yi (Joshua) Ren

TK



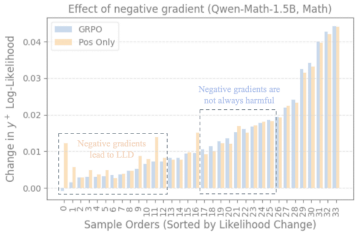
madi
e

DPO/etc analysis
[ICLR-25]



Yi Ren

GRPO analysis+fix
[arXiv]



Wenlong Deng
Yi Ren
Muchen Li
Xiaoxiao Li
Christos Thrampoulidis

Mila – June 2025

HTML version

LLM "post-training"

- Language models (e.g. GPT-2)
  - Scrape up a ton of the internet (usually illegally)
  - Train a big Transformer for next-token prediction

LLM “post-training”

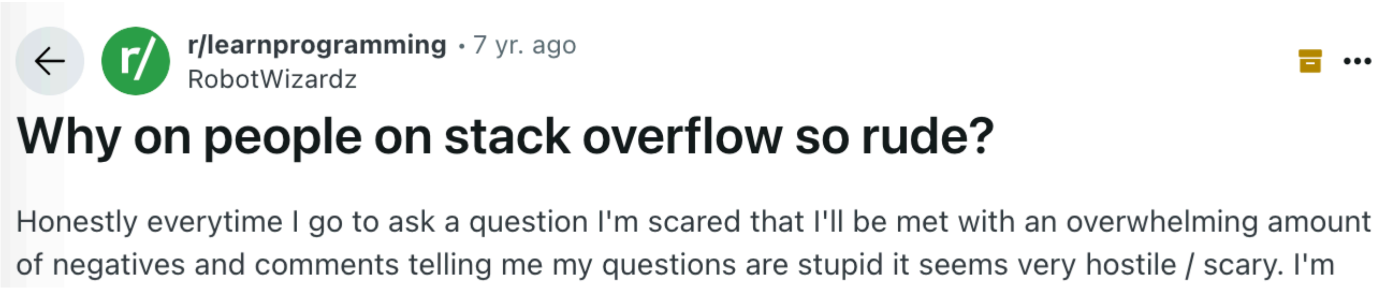
- Language models (e.g. GPT-2)
  - Scrape up a ton of the internet (usually illegally)
  - Train a big Transformer for next-token prediction
  - Super-useful as component of lots of things

LLM “post-training”

- Language models (e.g. GPT-2)
  - Scrape up a ton of the internet (usually illegally)
  - Train a big Transformer for next-token prediction
  - Super-useful as component of lots of things…but not necessarily what we want itself

LLM “post-training”

- Language models (e.g. GPT-2)
  - Scrape up a ton of the internet (usually illegally)
  - Train a big Transformer for next-token prediction
  - Super-useful as component of lots of things...but not necessarily what we want itself

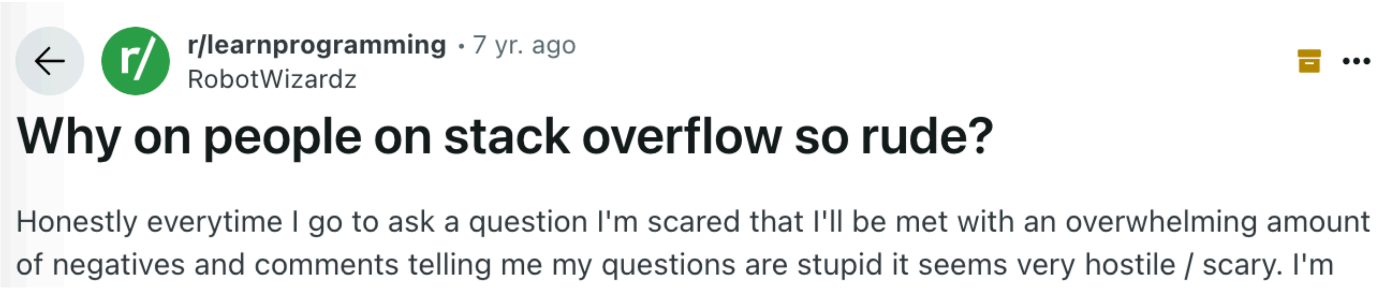

r/learnprogramming • 7 yr. ago
RobotWizardz

Why on people on stack overflow so rude?

Honestly everytime I go to ask a question I'm scared that I'll be met with an overwhelming amount of negatives and comments telling me my questions are stupid it seems very hostile / scary. I'm

LLM “post-training”

- Language models (e.g. GPT-2)
  - Scrape up a ton of the internet (usually illegally)
  - Train a big Transformer for next-token prediction
  - Super-useful as component of lots of things...but not necessarily what we want itself



- Turning a language model into a chatbot (e.g. ChatGPT):
  - Run “supervised fine-tuning” on a dataset of chatbot-like interactions
  - Run “preference optimization”: given prompt x, say A, not B

Surprises in LLM post-training

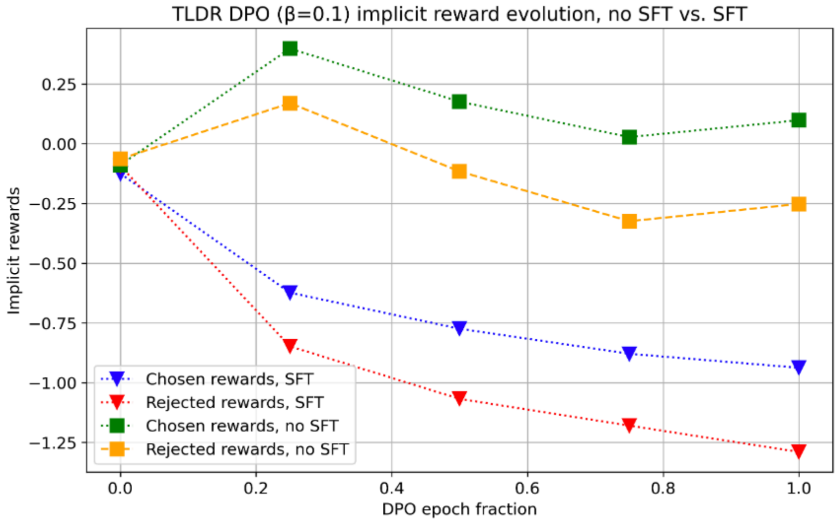
- Preference optimization: “given prompt x, say A, not B”
- Common algorithm: Direct Preference Optimization [RSM+ NeurIPS-23]

Surprises in LLM post-training

- Preference optimization: “given prompt x, say A, not B”
- Common algorithm: Direct Preference Optimization [RSM+ NeurIPS-23]
- Weird things can happen here!

Surprises in LLM post-training

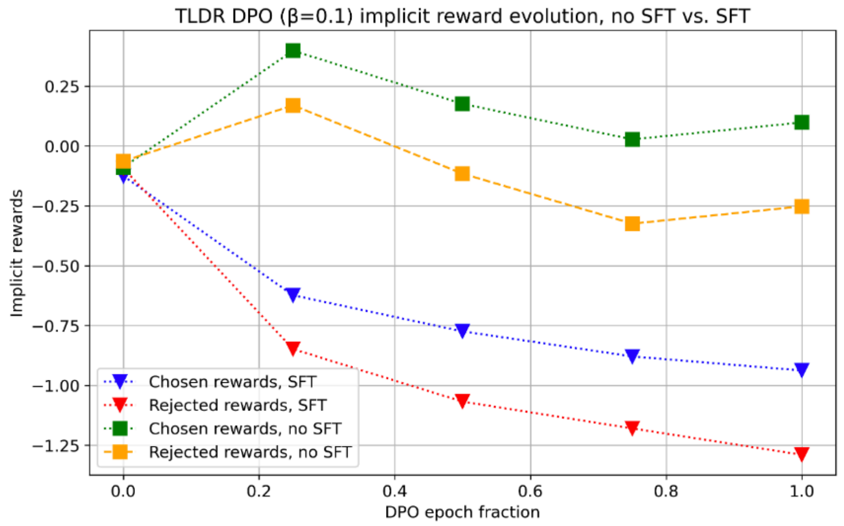
- Preference optimization: “given prompt x, say A, not B”
- Common algorithm: Direct Preference Optimization [RSM+ NeurIPS-23]
- Weird things can happen here!



- Even in the best case, “too much” DPO hurts [RHPF CoLM-24]

Surprises in LLM post-training

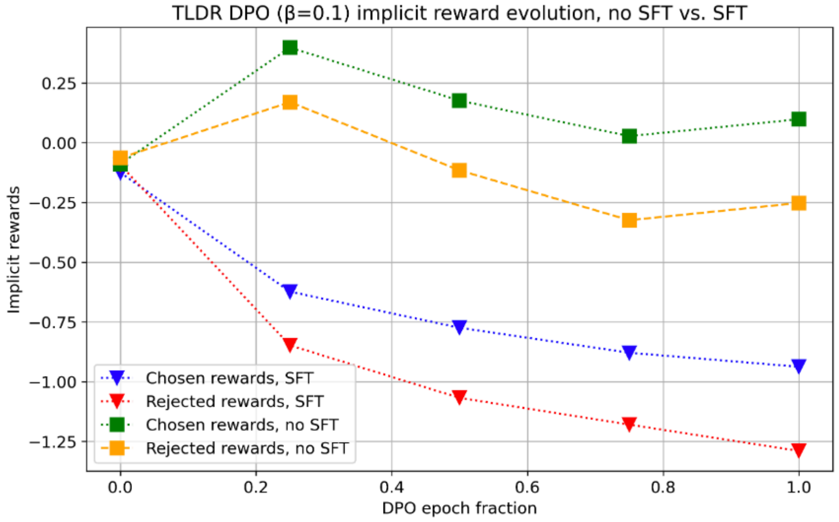
- Preference optimization: “given prompt x, say A, not B”
- Common algorithm: Direct Preference Optimization [RSM+ NeurIPS-23]
- Weird things can happen here!



- Even in the best case, “too much” DPO hurts [RHPF CoLM-24]
- Makes B way less likely, but eventually, model almost always says some C

Surprises in LLM post-training

- Preference optimization: "given prompt x, say A, not B"
- Common algorithm: Direct Preference Optimization [RSM+ NeurIPS-23]
- Weird things can happen here!



- Even in the best case, "too much" DPO hurts [RHPF CoLM-24]
- Makes B way less likely, but eventually, model almost always says some C
- There are some workarounds, but…why?

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions
  - "Implicit regularization" studies which one GD/SGD/Adam/... eventually converges to

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions
  - “Implicit regularization” studies which one GD/SGD/Adam/… eventually converges to
    - “Eventually” can take a *really* long time

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions
  - “Implicit regularization” studies which one GD/SGD/Adam/… eventually converges to
    - “Eventually” can take a *really* long time
- We'll take a related, but more qualitative approach

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions
  - “Implicit regularization” studies which one GD/SGD/Adam/… eventually converges to
    - “Eventually” can take a *really* long time
- We'll take a related, but more qualitative approach
- What does each step do to the model?

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions
  - “Implicit regularization” studies which one GD/SGD/Adam/... eventually converges to
    - “Eventually” can take a *really* long time
- We'll take a related, but more qualitative approach
- What does each step do to the model?
  - Mapping from parameters to “what the model does” is complicated

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions
  - "Implicit regularization" studies which one GD/SGD/Adam/… eventually converges to
    - "Eventually" can take a *really* long time
- We'll take a related, but more qualitative approach
- What does each step do to the model?
  - Mapping from parameters to "what the model does" is complicated
  - **When I take an SGD step to "learn" $f(x_i) \approx y_i$,**
    what happens to my predictions on $\tilde{x}$?

Learning dynamics

- Most theoretical analyses in this area ask: what do optimal solutions look like?
  - Turns out the loss function is very underspecified; there are many optimal solutions
  - “Implicit regularization” studies which one GD/SGD/Adam/… eventually converges to
    - “Eventually” can take a *really* long time
- We'll take a related, but more qualitative approach
- What does each step do to the model?
  - Mapping from parameters to “what the model does” is complicated
  - **When I take an SGD step to “learn” $f(x_i) \approx y_i$, what happens to my predictions on $\tilde{x}$?**
    - Also been called “local elasticity” [HS ICLR-20]

Learning dynamics

- “Learning dynamics” of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change

Learning dynamics

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- Suppose $\mathbf{z} = z(x)$, $\mathbf{f} = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_t(x_i, y_i)$:

Learning dynamics (Taylor's version)

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- Suppose $\mathbf{z} = z(x)$, $\mathbf{f} = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_t(x_i, y_i)$:

$$\underbrace{\mathbf{f}_{t+1}(\tilde{x})}_{k \times 1} - \underbrace{\mathbf{f}_t(\tilde{x})}_{k \times 1} = \underbrace{\left(\nabla_{\mathbf{w}} \mathbf{f}(\tilde{x})|_{\mathbf{w}_t}\right)}_{k \times p} \underbrace{\left(\mathbf{w}_{t+1} - \mathbf{w}_t\right)}_{p \times 1} + \mathcal{O}\left(\|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2\right)$$

Learning dynamics (Taylor's version)

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- Suppose $\mathbf{z} = z(x)$, $\mathbf{f} = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_t(x_i, y_i)$:

$$\underbrace{\mathbf{f}_{t+1}(\tilde{x})}_{k\times 1} - \underbrace{\mathbf{f}_t(\tilde{x})}_{k\times 1} = \underbrace{\left(\nabla_{\mathbf{w}} \mathbf{f}(\tilde{x})|_{\mathbf{w}_t}\right)}_{k\times p} \underbrace{\left(\mathbf{w}_{t+1} - \mathbf{w}_t\right)}_{p\times 1} + \mathcal{O}\left(\|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2\right)$$

$$= \underbrace{\left(\nabla_{\mathbf{w}} \mathbf{f}(\tilde{x})|_{\mathbf{w}_t}\right)}_{k\times p} \left(-\eta \underbrace{\nabla_{\mathbf{w}} \mathcal{L}(x_i, y_i)|_{\mathbf{w}_t}}_{1\times p}\right)^{\mathsf{T}} + \mathcal{O}(\eta^2)$$

Learning dynamics (Taylor's version)

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- Suppose $\mathbf{z} = z(x)$, $\mathbf{f} = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_t(x_i, y_i)$:

$$\underbrace{\mathbf{f}_{t+1}(\tilde{x})}_{k\times 1} - \underbrace{\mathbf{f}_t(\tilde{x})}_{k\times 1} = \underbrace{(\nabla_{\mathbf{w}} \mathbf{f}(\tilde{x})|_{\mathbf{w}_t})}_{k\times p} \underbrace{(\mathbf{w}_{t+1} - \mathbf{w}_t)}_{p\times 1} + \mathcal{O}\left(\|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2\right)$$

$$= \underbrace{(\nabla_{\mathbf{w}} \mathbf{f}(\tilde{x})|_{\mathbf{w}_t})}_{k\times p} \left(-\eta \underbrace{\nabla_{\mathbf{w}} \mathcal{L}(x_i, y_i)|_{\mathbf{w}_t}}_{1\times p}\right)^{\mathsf{T}} + \mathcal{O}(\eta^2)$$

$$= -\eta \underbrace{(\nabla_{\mathbf{z}} \mathbf{f}(\tilde{x})|_{\mathbf{z}_t})}_{k\times k} \underbrace{(\nabla_{\mathbf{w}} \mathbf{z}(\tilde{x})|_{\mathbf{w}_t})}_{k\times p} \underbrace{(\nabla_{\mathbf{w}} \mathbf{z}(x_i)|_{\mathbf{w}_t})^{\mathsf{T}}}_{p\times k} \underbrace{(\nabla_{\mathbf{z}} \mathcal{L}(x_i, y_i)|_{\mathbf{z}_t})^{\mathsf{T}}}_{k\times 1} + \mathcal{O}(\eta^2)$$

Learning dynamics (Taylor's version)

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- Suppose $\mathbf{z} = z(x)$, $\mathbf{f} = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_t(x_i, y_i)$:

$$\underbrace{\mathbf{f}_{t+1}(\tilde{x})}_{k\times 1} - \underbrace{\mathbf{f}_t(\tilde{x})}_{k\times 1} = \underbrace{\left(\nabla_{\mathbf{w}}\mathbf{f}(\tilde{x})|_{\mathbf{w}_t}\right)}_{k\times p}\underbrace{\left(\mathbf{w}_{t+1} - \mathbf{w}_t\right)}_{p\times 1} + \mathcal{O}\left(\|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2\right)$$

$$= \underbrace{\left(\nabla_{\mathbf{w}}\mathbf{f}(\tilde{x})|_{\mathbf{w}_t}\right)}_{k\times p}\left(-\eta\underbrace{\nabla_{\mathbf{w}}\mathcal{L}(x_i, y_i)|_{\mathbf{w}_t}}_{1\times p}\right)^{\mathsf{T}} + \mathcal{O}(\eta^2)$$

$$= -\eta\underbrace{\left(\nabla_{\mathbf{z}}\mathbf{f}(\tilde{x})|_{\mathbf{z}_t}\right)}_{k\times k}\underbrace{\left(\nabla_{\mathbf{w}}\mathbf{z}(\tilde{x})|_{\mathbf{w}_t}\right)}_{k\times p}\underbrace{\left(\nabla_{\mathbf{w}}\mathbf{z}(x_i)|_{\mathbf{w}_t}\right)^{\mathsf{T}}}_{p\times k}\underbrace{\left(\nabla_{\mathbf{z}}\mathcal{L}(x_i, y_i)|_{\mathbf{z}_t}\right)^{\mathsf{T}}}_{k\times 1} + \mathcal{O}(\eta^2)$$

$$= -\eta \quad \mathcal{A}_t(\tilde{x}) \qquad \mathcal{K}_t(\tilde{x}, x_i) \qquad \mathcal{G}_t(x_i, y_i) \qquad + \mathcal{O}(\eta^2)$$

Learning dynamics

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- To start: $\mathbf{z} = h_\theta(x)$, $f = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^N \mathcal{L}_t(x_i, y_i)$:

$$\mathbf{f}_{t+1}(\tilde{x}) - \mathbf{f}_t(\tilde{x}) = -\eta\, \mathcal{A}_t(\tilde{x})\, \mathcal{K}_t(\tilde{x}, x_i)\, \mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

Learning dynamics

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- To start: $\mathbf{z} = h_\theta(x)$, $f = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_t(x_i, y_i)$:

$$\mathbf{f}_{t+1}(\tilde{x}) - \mathbf{f}_t(\tilde{x}) = -\eta\, \mathcal{A}_t(\tilde{x})\, \mathcal{K}_t(\tilde{x}, x_i)\, \mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

- $\mathcal{G}_t(x_i, y_i) = \nabla_{\mathbf{z}} \mathcal{L}(x_i, y_i)\big|_{\mathbf{z}_t}$: how much do I need to change my x_i prediction?
  - For square loss with $\sigma(\mathbf{z}) = \mathbf{z}$, $\mathcal{G}_t = \mathbf{f}_t(x_i) - y_i$: how wrong was I before?
  - For cross-entropy on logits, $\mathcal{G}_t = \mathrm{Softmax}(\mathbf{z}_t(x_i))_{y_i} - e_{y_i}$: how wrong was I before?

Learning dynamics

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- To start: $\mathbf{z} = h_\theta(x)$, $f = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^N \mathcal{L}_t(x_i, y_i)$:

$$\mathbf{f}_{t+1}(\tilde{x}) - \mathbf{f}_t(\tilde{x}) = -\eta\, \mathcal{A}_t(\tilde{x})\, \mathcal{K}_t(\tilde{x}, x_i)\, \mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

- $\mathcal{G}_t(x_i, y_i) = \nabla_{\mathbf{z}} \mathcal{L}(x_i, y_i)\big|_{\mathbf{z}_t}$: how much do I need to change my x_i prediction?
  - For square loss with $\sigma(\mathbf{z}) = \mathbf{z}$, $\mathcal{G}_t = \mathbf{f}_t(x_i) - y_i$: how wrong was I before?
  - For cross-entropy on logits, $\mathcal{G}_t = \mathrm{Softmax}(\mathbf{z}_t(x_i))_{y_i} - e_{y_i}$: how wrong was I before?
- $\mathcal{A}_t(\tilde{x}) = \nabla_{\mathbf{z}} \sigma(\mathbf{z}_t)$ just "converts" prediction changes
  - If $\sigma(\mathbf{z}) = \mathbf{z}$, $\mathcal{A}_t$ is the identity; if $\sigma = \log \mathrm{Softmax}$, $\mathcal{A}_t = \mathbf{I}_k - \mathbf{1}_k\, \pi_t(\tilde{x})^\mathsf{T}$

Learning dynamics

- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- To start: $\mathbf{z} = h_\theta(x)$, $f = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^N \mathcal{L}_t(x_i, y_i)$:

$$\mathbf{f}_{t+1}(\tilde{x}) - \mathbf{f}_t(\tilde{x}) = -\eta\, \mathcal{A}_t(\tilde{x})\, \mathcal{K}_t(\tilde{x}, x_i)\, \mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

- $\mathcal{G}_t(x_i, y_i) = \nabla_{\mathbf{z}} \mathcal{L}(x_i, y_i)|_{\mathbf{z}_t}$: how much do I need to change my x_i prediction?
  - For square loss with $\sigma(\mathbf{z}) = \mathbf{z}$, $\mathcal{G}_t = \mathbf{f}_t(x_i) - y_i$: how wrong was I before?
  - For cross-entropy on logits, $\mathcal{G}_t = \mathrm{Softmax}(\mathbf{z}_t(x_i))_{y_i} - e_{y_i}$: how wrong was I before?
- $\mathcal{A}_t(\tilde{x}) = \nabla_{\mathbf{z}} \sigma(\mathbf{z}_t)$ just "converts" prediction changes
  - If $\sigma(\mathbf{z}) = \mathbf{z}$, $\mathcal{A}_t$ is the identity; if $\sigma = \log \mathrm{Softmax}$, $\mathcal{A}_t = \mathbf{I}_k - \mathbf{1}_k\, \pi_t(\tilde{x})^\mathsf{T}$
- $\mathcal{K}_t(\tilde{x}, x_i) = (\nabla_{\mathbf{w}} \mathbf{z}(\tilde{x})|_{\mathbf{w}_t})(\nabla_{\mathbf{w}} \mathbf{z}(x_i)|_{\mathbf{w}_t})^\mathsf{T}$ is $k \times k$ empirical neural tangent kernel of $\mathbf{z}$

Learning dynamics

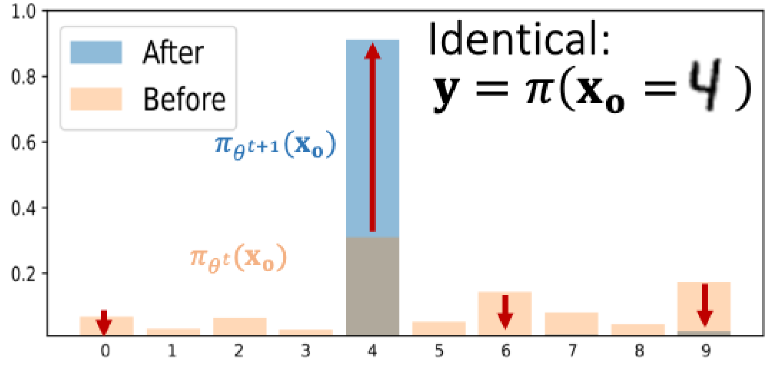
- "Learning dynamics" of a model: $f_t(\tilde{x})$ for some fixed $\tilde{x}$ as params $\mathbf{w}_t$ change
- To start: $\mathbf{z} = h_\theta(x)$, $f = \sigma(\mathbf{z})$, "plain" SGD on $\frac{1}{N}\sum_{i=1}^N \mathcal{L}_t(x_i, y_i)$:

$$\mathbf{f}_{t+1}(\tilde{x}) - \mathbf{f}_t(\tilde{x}) = -\eta\, \mathcal{A}_t(\tilde{x})\, \mathcal{K}_t(\tilde{x}, x_i)\, \mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

- $\mathcal{G}_t(x_i, y_i) = \nabla_{\mathbf{z}} \mathcal{L}(x_i, y_i)\big|_{\mathbf{z}_t}$: how much do I need to change my x_i prediction?
  - For square loss with $\sigma(\mathbf{z}) = \mathbf{z}$, $\mathcal{G}_t = \mathbf{f}_t(x_i) - y_i$: how wrong was I before?
  - For cross-entropy on logits, $\mathcal{G}_t = \mathrm{Softmax}(\mathbf{z}_t(x_i))_{y_i} - e_{y_i}$: how wrong was I before?
- $\mathcal{A}_t(\tilde{x}) = \nabla_{\mathbf{z}} \sigma(\mathbf{z}_t)$ just "converts" prediction changes
  - If $\sigma(\mathbf{z}) = \mathbf{z}$, $\mathcal{A}_t$ is the identity; if $\sigma = \log \mathrm{Softmax}$, $\mathcal{A}_t = \mathbf{I}_k - \mathbf{1}_k\, \pi_t(\tilde{x})^\mathsf{T}$
- $\mathcal{K}_t(\tilde{x}, x_i) = (\nabla_{\mathbf{w}} \mathbf{z}(\tilde{x})\big|_{\mathbf{w}_t})(\nabla_{\mathbf{w}} \mathbf{z}(x_i)\big|_{\mathbf{w}_t})^\mathsf{T}$ is $k \times k$ empirical neural tangent kernel of $\mathbf{z}$
  - If x_i, $\tilde{x}$ are "dissimilar" (small eNTK), stepping on (x_i, y_i) barely changes $\tilde{x}$ prediction
  - If x_i, $\tilde{x}$ are "similar" (large eNTK), makes $\tilde{x}$ prediction more like y_i

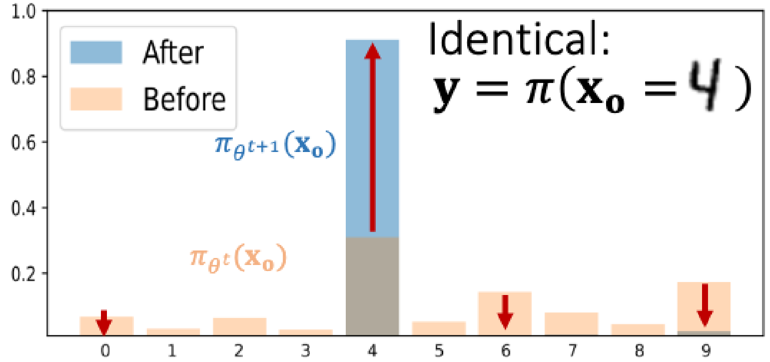
Example: learning dynamics on MNIST

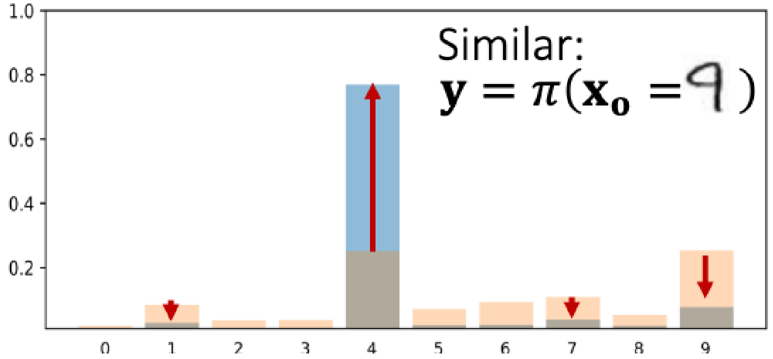
$$\log \pi_{t+1}(\tilde{x}) - \log \pi_t(\tilde{x}) \approx -\eta \mathcal{A}_t(\tilde{x}) \mathcal{K}_t(\tilde{x}, x_i) \mathcal{G}_t(x_i, y_i)$$



Example: learning dynamics on MNIST

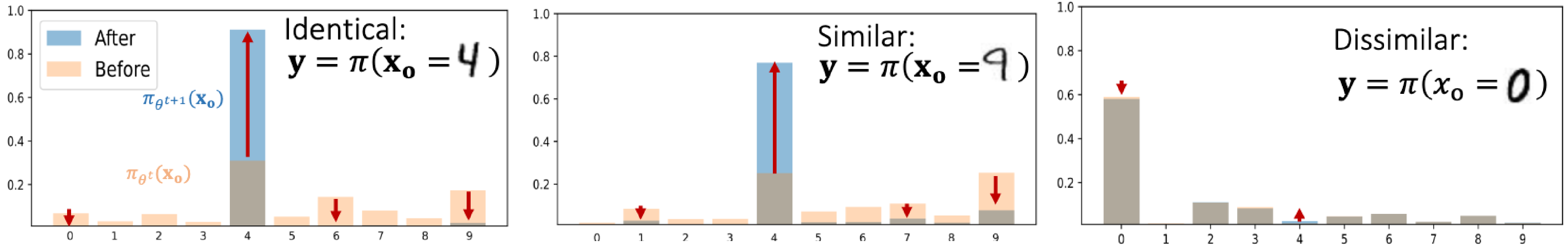
$$\log \pi_{t+1}(\tilde{x}) - \log \pi_t(\tilde{x}) \approx -\eta \mathcal{A}_t(\tilde{x}) \mathcal{K}_t(\tilde{x}, x_i) \mathcal{G}_t(x_i, y_i)$$





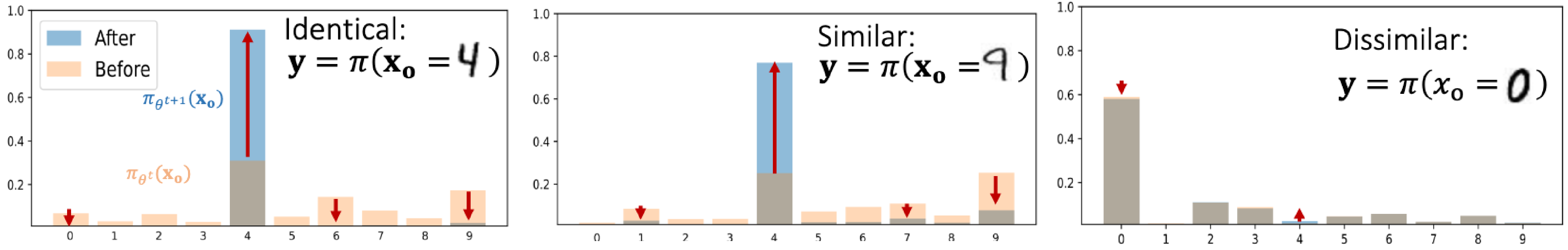
Example: learning dynamics on MNIST

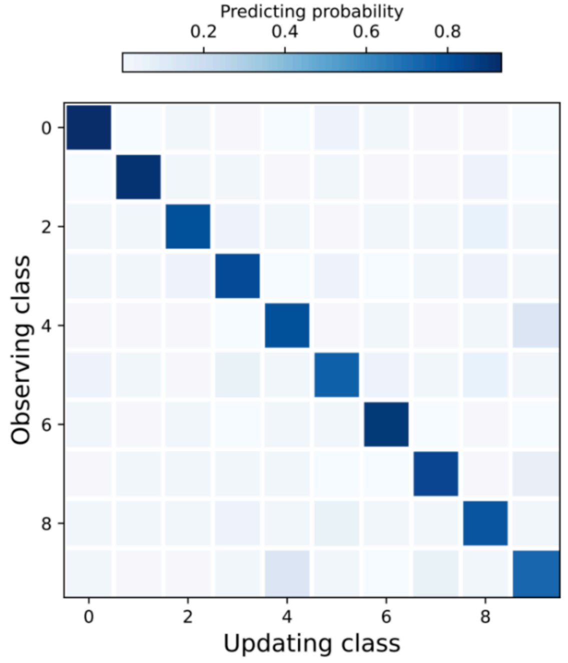
$$\log \pi_{t+1}(\tilde{x}) - \log \pi_t(\tilde{x}) \approx -\eta \mathcal{A}_t(\tilde{x})\, \mathcal{K}_t(\tilde{x}, x_i)\, \mathcal{G}_t(x_i, y_i)$$



Example: learning dynamics on MNIST

$$\log \pi_{t+1}(\tilde{x}) - \log \pi_t(\tilde{x}) \approx -\eta \mathcal{A}_t(\tilde{x}) \mathcal{K}_t(\tilde{x}, x_i) \mathcal{G}_t(x_i, y_i)$$





But wait...aren't NTKs an unrealistic approximation?

But wait...aren't NTKs an unrealistic approximation?


Ben Recht @beenwrekt · Jan 19
Replying to @KameronDHarris and @deepcohen
This! I spent a lot of time digging into NTKs, and I'm still not sure the math tells us much.
2 2 811

Lorenzo Rosasco @lrntzrsc · Jan 19
Replying to @beenwrekt @KameronDHarris and @deepcohen
I am more pessimistic than this, I am not sure NTKs say much at all.
3 5 638

But wait...aren't NTKs an unrealistic approximation?


Ben Recht @beenwrekt · Jan 19
Replying to @KameronDHarris and @deepcohen
This! I spent a lot of time digging into NTKs, and I'm still not sure the math tells us much.
2 2 811

Lorenzo Rosasco @lrntzrsc · Jan 19
Replying to @beenwrekt @KameronDHarris and @deepcohen
I am more pessimistic than this, I am not sure NTKs say much at all.
3 5 638

A quick aside: the “NTK regime” and infinite limits

- Full-batch GD:

$$f_{t+1}(\tilde{x}) - f_t(\tilde{x}) = -\frac{\eta}{N}\sum_{i=1}^{N} \mathcal{A}_t(\tilde{x})\,\mathcal{K}_t(\tilde{x}, x_i)\mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

- Full-batch GD: "stacking things up",

$$f_{t+1}(\tilde{x}) - f_t(\tilde{x}) = -\frac{\eta}{N}\sum_{i=1}^{N} \mathcal{A}_t(\tilde{x})\,\mathcal{K}_t(\tilde{x}, x_i)\mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

$$= -\frac{\eta}{N}\underbrace{\mathcal{A}_t(\tilde{x})}_{k\times k}\,\underbrace{\mathcal{K}_t(\tilde{x}, \mathbf{X})}_{k\times kN}\,\underbrace{\mathcal{G}_t(\mathbf{X}, \mathbf{y})}_{kN\times 1} + \mathcal{O}(\eta^2)$$

- Full-batch GD: "stacking things up",

$$f_{t+1}(\tilde{x}) - f_t(\tilde{x}) = -\frac{\eta}{N}\sum_{i=1}^{N} \mathcal{A}_t(\tilde{x})\,\mathcal{K}_t(\tilde{x}, x_i)\mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

$$= -\frac{\eta}{N}\underbrace{\mathcal{A}_t(\tilde{x})}_{k\times k}\,\underbrace{\mathcal{K}_t(\tilde{x}, \mathbf{X})}_{k\times kN}\,\underbrace{\mathcal{G}_t(\mathbf{X}, \mathbf{y})}_{kN\times 1} + \mathcal{O}(\eta^2)$$

- Observation I: If f is "wide enough" with any usual architecture+init* [Yang+Litwin 2021], $\mathcal{K}_t(\cdot, \mathbf{X})$ is roughly constant through training

- Full-batch GD: "stacking things up",

$$f_{t+1}(\tilde{x}) - f_t(\tilde{x}) = -\frac{\eta}{N} \sum_{i=1}^{N} \mathcal{A}_t(\tilde{x}) \mathcal{K}_t(\tilde{x}, x_i) \mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$

$$= -\frac{\eta}{N} \underbrace{\mathcal{A}_t(\tilde{x})}_{k \times k} \underbrace{\mathcal{K}_t(\tilde{x}, \mathbf{X})}_{k \times kN} \underbrace{\mathcal{G}_t(\mathbf{X}, \mathbf{y})}_{kN \times 1} + \mathcal{O}(\eta^2)$$

- Observation I: If f is "wide enough" with any usual architecture+init* [Yang+Litwin 2021], $\mathcal{K}_t(\cdot, \mathbf{X})$ is roughly constant through training
  - For square loss, $\mathcal{L}(\mathbf{X}, \mathbf{y}) = f_t(\mathbf{X}) - \mathbf{y}$: dynamics agree with kernel regression!

- Full-batch GD: "stacking things up",

$$f_{t+1}(\tilde{x}) - f_t(\tilde{x}) = -\frac{\eta}{N} \sum_{i=1}^{N} \mathcal{A}_t(\tilde{x}) \mathcal{K}_t(\tilde{x}, x_i) \mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$
$$= -\frac{\eta}{N} \underbrace{\mathcal{A}_t(\tilde{x})}_{k \times k} \underbrace{\mathcal{K}_t(\tilde{x}, \mathbf{X})}_{k \times kN} \underbrace{\mathcal{G}_t(\mathbf{X}, \mathbf{y})}_{kN \times 1} + \mathcal{O}(\eta^2)$$

- Observation I: If f is "wide enough" with any usual architecture+init* [Yang+Litwin 2021], $\mathcal{K}_t(\cdot, \mathbf{X})$ is roughly constant through training
    - For square loss, $\mathcal{L}(\mathbf{X}, \mathbf{y}) = f_t(\mathbf{X}) - \mathbf{y}$: dynamics agree with kernel regression!
    - $f_t(\tilde{x}) \xrightarrow{t \to \infty} \mathcal{K}_0(\tilde{x}, \mathbf{X}) \mathcal{K}_0(\mathbf{X}, \mathbf{X})^{-1} (\mathbf{y} - f_0(\mathbf{X})) + f_0(\tilde{x})$

- Full-batch GD: "stacking things up",

$$f_{t+1}(\tilde{x}) - f_t(\tilde{x}) = -\frac{\eta}{N}\sum_{i=1}^{N}\mathcal{A}_t(\tilde{x})\,\mathcal{K}_t(\tilde{x}, x_i)\mathcal{G}_t(x_i, y_i) + \mathcal{O}(\eta^2)$$
$$= -\frac{\eta}{N}\underbrace{\mathcal{A}_t(\tilde{x})}_{k\times k}\,\underbrace{\mathcal{K}_t(\tilde{x}, \mathbf{X})}_{k\times kN}\,\underbrace{\mathcal{G}_t(\mathbf{X}, \mathbf{y})}_{kN\times 1} + \mathcal{O}(\eta^2)$$

- Observation I: If f is "wide enough" with any usual architecture+init* [Yang+Litwin 2021], $\mathcal{K}_t(\cdot, \mathbf{X})$ is roughly constant through training
  - For square loss, $\mathcal{L}(\mathbf{X}, \mathbf{y}) = f_t(\mathbf{X}) - \mathbf{y}$: dynamics agree with kernel regression!
  - $f_t(\tilde{x}) \xrightarrow{t\to\infty} \mathcal{K}_0(\tilde{x}, \mathbf{X})\,\mathcal{K}_0(\mathbf{X}, \mathbf{X})^{-1}(\mathbf{y} - f_0(\mathbf{X})) + f_0(\tilde{x})$

- Observation II: As f becomes "infinitely wide" with any usual architecture+init* [Yang 2019], $\mathcal{K}_0(x_1, x_2) \xrightarrow{a.s.} \mathrm{NTK}(x_1, x_2)$, independent of the random $\mathbf{w}_0$

Infinite NTKs are great

- Infinitely-wide neural networks have very simple behaviour!
  - No need to worry about bad local minima, optimization complications, …

Infinite NTKs are great

- Infinitely-wide neural networks have very simple behaviour!
  - No need to worry about bad local minima, optimization complications, …
  - Understanding “implicit bias” of wide nets $\approx$ understanding NTK norm of functions

Infinite NTKs are great

- Infinitely-wide neural networks have very simple behaviour!
  - No need to worry about bad local minima, optimization complications, ...
  - Understanding “implicit bias” of wide nets $\approx$ understanding NTK norm of functions
- Can compute NTK exactly for many architectures
  - `github.com/google/neural-tangents`

Infinite NTKs are great

- Infinitely-wide neural networks have very simple behaviour!
  - No need to worry about bad local minima, optimization complications, ...
  - Understanding "implicit bias" of wide nets $\approx$ understanding NTK norm of functions
- Can compute NTK exactly for many architectures
  - `github.com/google/neural-tangents`
- A great kernel for many kernel methods!
  - Using in SVMs was then-best overall method across many small-data tasks [Arora+ 2020]

Infinite NTKs are great

- Infinitely-wide neural networks have very simple behaviour!
  - No need to worry about bad local minima, optimization complications, …
  - Understanding “implicit bias” of wide nets $\approx$ understanding NTK norm of functions
- Can compute NTK exactly for many architectures
  - `github.com/google/neural-tangents`
- A great kernel for many kernel methods!
  - Using in SVMs was then-best overall method across many small-data tasks [Arora+ 2020]
  - Good results in statistical testing [Jia+ 2021], dataset distillation [Nguyen+ 2021], clustering for active learning batch queries [Holzmüller+ 2022], …

But (infinite) NTKs aren't "the answer"

- Computational expense:
  - Poor scaling for large-data problems: typically n^2 memory and n^2 to n^3 computation
    - CIFAR-10 has $n = 50\,000$, $k = 10$: an $nk \times nk$ matrix of `float64`s is 2 terabytes!

But (infinite) NTKs aren't “the answer”

- Computational expense:
  - Poor scaling for large-data problems: typically n^2 memory and n^2 to n^3 computation
    - CIFAR-10 has $n = 50\,000$, $k = 10$: an $nk \times nk$ matrix of `float64`s is 2 terabytes!
    - ILSVRC2012 has $n \approx 1\,200\,000$, $k = 1\,000$: 11.5 *million* terabytes (exabytes)

But (infinite) NTKs aren't "the answer"

- Computational expense:
  - Poor scaling for large-data problems: typically n^2 memory and n^2 to n^3 computation
    - CIFAR-10 has $n = 50\,000$, $k = 10$: an $nk \times nk$ matrix of `float64`s is 2 terabytes!
    - ILSVRC2012 has $n \approx 1\,200\,000$, $k = 1\,000$: 11.5 *million* terabytes (exabytes)
  - For deep/complex models (especially CNNs), each pair very slow / memory-intensive

But (infinite) NTKs aren't "the answer"

- Computational expense:
  - Poor scaling for large-data problems: typically n^2 memory and n^2 to n^3 computation
    - CIFAR-10 has $n = 50\,000$, $k = 10$: an $nk \times nk$ matrix of `float64`s is 2 terabytes!
    - ILSVRC2012 has $n \approx 1\,200\,000$, $k = 1\,000$: 11.5 *million* terabytes (exabytes)
  - For deep/complex models (especially CNNs), each pair very slow / memory-intensive
  - Attention is even harder to handle

But (infinite) NTKs aren't "the answer"

- Computational expense:
  - Poor scaling for large-data problems: typically n^2 memory and n^2 to n^3 computation
    - CIFAR-10 has $n = 50\,000$, $k = 10$: an $nk \times nk$ matrix of `float64`s is 2 terabytes!
    - ILSVRC2012 has $n \approx 1\,200\,000$, $k = 1\,000$: 11.5 *million* terabytes (exabytes)
  - For deep/complex models (especially CNNs), each pair very slow / memory-intensive
  - Attention is even harder to handle
- Practical performance:
  - Typically performs worse than GD for "non-small-data" tasks (MNIST and up)

But (infinite) NTKs aren't “the answer”

- Computational expense:
  - Poor scaling for large-data problems: typically n^2 memory and n^2 to n^3 computation
    - CIFAR-10 has $n = 50\,000$, $k = 10$: an $nk \times nk$ matrix of `float64`s is 2 terabytes!
    - ILSVRC2012 has $n \approx 1\,200\,000$, $k = 1\,000$: 11.5 *million* terabytes (exabytes)
  - For deep/complex models (especially CNNs), each pair very slow / memory-intensive
  - Attention is even harder to handle
- Practical performance:
  - Typically performs worse than GD for “non-small-data” tasks (MNIST and up)
- Theoretical limitations:
  - NTK “doesn't do feature learning”:
    - $\mathcal{K}$, activations in the net don't change much [Chizat+ 2019] [Yang/Hu 2021]

But (infinite) NTKs aren't "the answer"

- Computational expense:
  - Poor scaling for large-data problems: typically n^2 memory and n^2 to n^3 computation
    - CIFAR-10 has $n = 50\,000$, $k = 10$: an $nk \times nk$ matrix of `float64`s is 2 terabytes!
    - ILSVRC2012 has $n \approx 1\,200\,000$, $k = 1\,000$: 11.5 *million* terabytes (exabytes)
  - For deep/complex models (especially CNNs), each pair very slow / memory-intensive
  - Attention is even harder to handle
- Practical performance:
  - Typically performs worse than GD for "non-small-data" tasks (MNIST and up)
- Theoretical limitations:
  - NTK "doesn't do feature learning":
    - $\mathcal{K}$, activations in the net don't change much [Chizat+ 2019] [Yang/Hu 2021]
  - We now know many problems where gradient descent on an NN $\gg$ *any* kernel method
    - Cases where GD error $\to 0$, any kernel is *barely* better than random [Malach+ 2021]

What can we learn from empirical NTKs?

What can we learn from empirical NTKs?

- As a theoretical tool for local understanding:
  - Why DPO breaks
  - Why GRPO does weird stuff + how to fix
  - Fine-grained explanation for early stopping in knowledge distillation
  - How you should fine-tune models
- As a practical tool for approximating “lookahead” in active learning
- Plus: efficiently approximating $\mathcal{K}$s for large output dimensions k, with guarantees

What can we learn from empirical NTKs?

- As a theoretical tool for local understanding:
  - Why DPO breaks
  - Why GRPO does weird stuff + how to fix
  - Fine-grained explanation for early stopping in knowledge distillation
  - How you should fine-tune models
- As a practical tool for approximating “lookahead” in active learning
- Plus: efficiently approximating $\mathcal{K}$s for large output dimensions k, with guarantees

Adapting to the LLM setting

- First problem: we don't classify a full response at a time, we do it token-by-token

Adapting to the LLM setting

- First problem: we don't classify a full response at a time, we do it token-by-token
- Once we've framed it correctly, this is fine: stack prompt+response into $\chi_i = [x_i, y_i]$

Adapting to the LLM setting

- First problem: we don't classify a full response at a time, we do it token-by-token
- Once we've framed it correctly, this is fine: stack prompt+response into $\chi_i = [x_i, y_i]$
- Change in $\log \pi(\tilde{y}_m \mid \tilde{x}, \tilde{y}_{:m-1})$ based on token-by-token update of x_i, y_i is

$$[\Delta \log \pi_t(\tilde{y} \mid \tilde{\chi})]_m = -\sum_{l=1}^{L_i} \eta [\mathcal{A}_t(\tilde{\chi})]_m [\mathcal{K}_t(\tilde{\chi}, \chi_i)]_{m,l} [\mathcal{G}_t(\chi_i)]_l + \mathcal{O}(\eta^2)$$

Adapting to the LLM setting

- First problem: we don't classify a full response at a time, we do it token-by-token
- Once we've framed it correctly, this is fine: stack prompt+response into $\chi_i = [x_i, y_i]$
- Change in $\log \pi(\tilde{y}_m \mid \tilde{x}, \tilde{y}_{:m-1})$ based on token-by-token update of x_i, y_i is

$$[\Delta \log \pi_t(\tilde{y} \mid \tilde{\chi})]_m = -\sum_{l=1}^{L_i} \eta [\mathcal{A}_t(\tilde{\chi})]_m [\mathcal{K}_t(\tilde{\chi}, \chi_i)]_{m,l} [\mathcal{G}_t(\chi_i)]_l + \mathcal{O}(\eta^2)$$

- Second problem: we can't check all possible output probabilities anymore

Adapting to the LLM setting

- First problem: we don't classify a full response at a time, we do it token-by-token
- Once we've framed it correctly, this is fine: stack prompt+response into $\chi_i = [x_i, y_i]$
- Change in $\log \pi(\tilde{y}_m \mid \tilde{x}, \tilde{y}_{:m-1})$ based on token-by-token update of x_i, y_i is

$$[\Delta \log \pi_t(\tilde{y} \mid \tilde{\chi})]_m = -\sum_{l=1}^{L_i} \eta [\mathcal{A}_t(\tilde{\chi})]_m [\mathcal{K}_t(\tilde{\chi}, \chi_i)]_{m,l} [\mathcal{G}_t(\chi_i)]_l + \mathcal{O}(\eta^2)$$

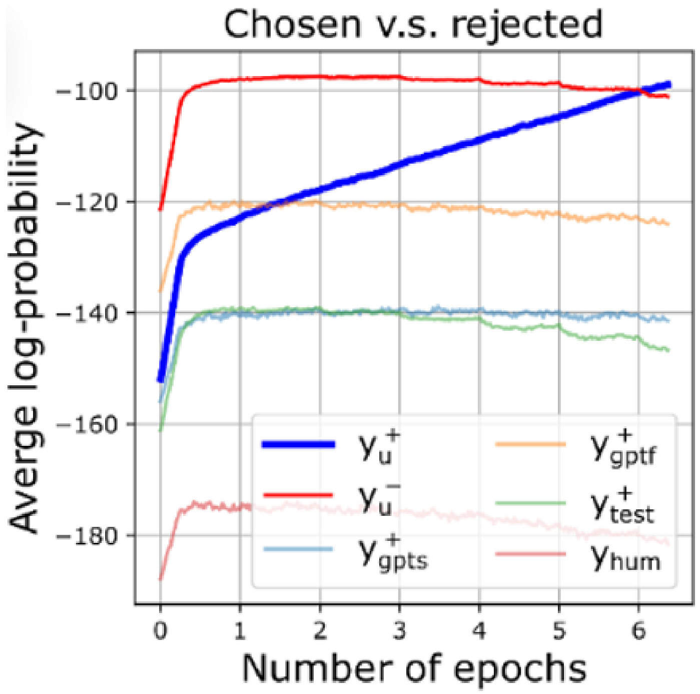
- Second problem: we can't check all possible output probabilities anymore
- Workaround: track some informative possible responses
  - The dataset responses, rephrases, similar strings with different meanings
  - Irrelevant responses in training set, random sentences...

LLM supervised fine-tuning

- SFT makes dispreferred answers more likely

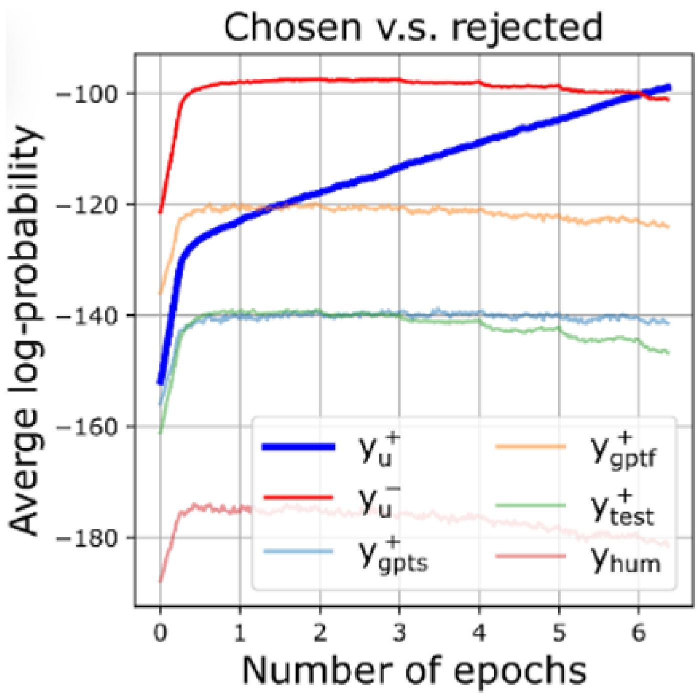
LLM supervised fine-tuning

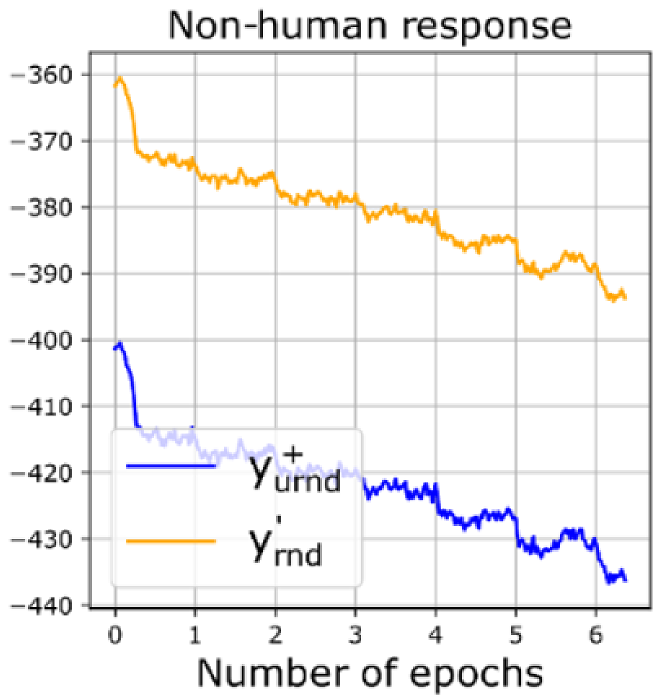
- SFT makes dispreferred answers more likely
- …because they're "similar enough" to the preferred ones
- $\mathcal{K}$ is reasonably large; $\mathcal{G}$ starts big (pulls up), gets small (pulls up less)



LLM supervised fine-tuning

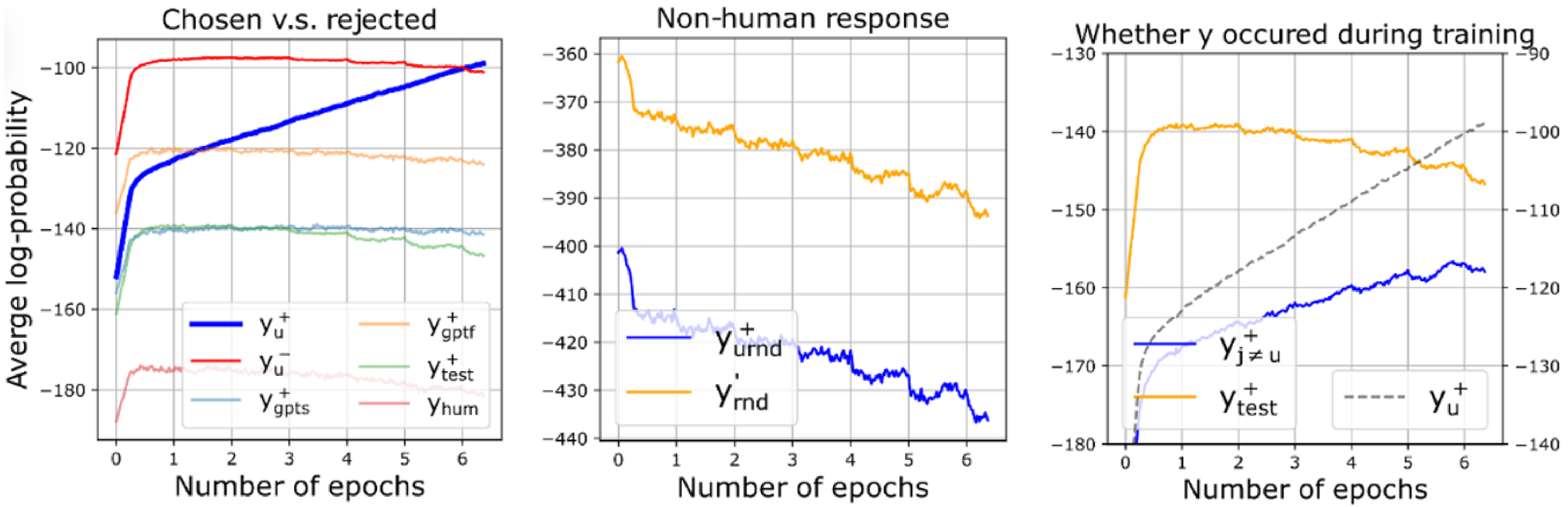
- SFT makes dispreferred answers more likely
- ...because they're “similar enough” to the preferred ones
- $\mathcal{K}$ is reasonably large; $\mathcal{G}$ starts big (pulls up), gets small (pulls up less)
- Ungrammatical responses just go down; $\mathcal{K}$ is small, so no upwards pressure





LLM supervised fine-tuning

- SFT makes dispreferred answers more likely
- …because they're "similar enough" to the preferred ones
- $\mathcal{K}$ is reasonably large; $\mathcal{G}$ starts big (pulls up), gets small (pulls up less)
- Ungrammatical responses just go down; $\mathcal{K}$ is small, so no upwards pressure
- Also makes answers to different questions more likely…one form of hallucination?



Direct Preference Optimization (DPO)

$$\mathcal{L}_t^{\text{DPO}}(x_i, y_i^+, y_i^-) = \log \sigma \left(\beta \left[\log \frac{\pi_t(y_i^+ \mid x_i)}{\pi_{\text{ref}}(y_i^+ \mid x_i)} - \log \frac{\pi_t(y_i^- \mid x_i)}{\pi_{\text{ref}}(y_i^- \mid x_i)} \right] \right)$$

Direct Preference Optimization (DPO)

$$\mathcal{L}_t^{\mathrm{DPO}}(x_i, y_i^+, y_i^-) = \log \sigma\left(\beta\left[\log \frac{\pi_t(y_i^+ \mid x_i)}{\pi_{\mathrm{ref}}(y_i^+ \mid x_i)} - \log \frac{\pi_t(y_i^- \mid x_i)}{\pi_{\mathrm{ref}}(y_i^- \mid x_i)}\right]\right)$$

which gives that $[\Delta \log \pi_t(\tilde{y} \mid \tilde{\chi})]_m$ is about $\quad \mathcal{G}_t^{\mathrm{DPO}}(\boldsymbol{\chi}) = \beta(1-\sigma(\ldots))(\pi_t(\mathbf{y} \mid \boldsymbol{\chi}) - e_{\mathbf{y}})$

$$-\eta[\mathcal{A}_t(\tilde{\chi})]_m \left(\sum_{l=1}^{L_i}[\mathcal{K}_t(\tilde{\chi}, \chi_i^+)]_{m,l}[\mathcal{G}_t^{\mathrm{DPO}}(\chi_i^+)]_l - \sum_{l=1}^{L_i}[\mathcal{K}_t(\tilde{\chi}, \chi_i^-)]_{m,l}[\mathcal{G}_t^{\mathrm{DPO}}(\chi_i^-)]_l\right)$$

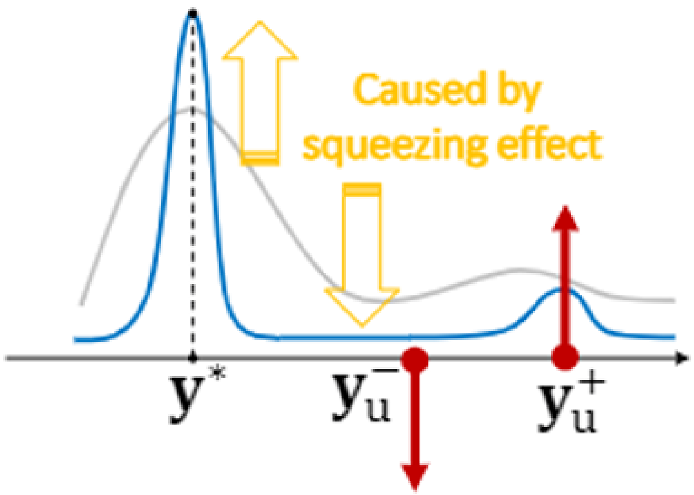
Direct Preference Optimization (DPO)

$$\mathcal{L}_t^{\mathrm{DPO}}(x_i, y_i^+, y_i^-) = \log \sigma \left(\beta \left[\log \frac{\pi_t(y_i^+ \mid x_i)}{\pi_{\mathrm{ref}}(y_i^+ \mid x_i)} - \log \frac{\pi_t(y_i^- \mid x_i)}{\pi_{\mathrm{ref}}(y_i^- \mid x_i)} \right] \right)$$

which gives that $[\Delta \log \pi_t(\tilde{y} \mid \tilde{\chi})]_m$ is about $\mathcal{G}_t^{\mathrm{DPO}}(\chi) = \beta(1 - \sigma(\ldots))(\pi_t(\mathbf{y} \mid \chi) - e_{\mathbf{y}})$

$$-\eta [\mathcal{A}_t(\tilde{\chi})]_m \left(\sum_{l=1}^{L_i} [\mathcal{K}_t(\tilde{\chi}, \chi_i^+)]_{m,l} [\mathcal{G}_t^{\mathrm{DPO}}(\chi_i^+)]_l - \sum_{l=1}^{L_i} [\mathcal{K}_t(\tilde{\chi}, \chi_i^-)]_{m,l} [\mathcal{G}_t^{\mathrm{DPO}}(\chi_i^-)]_l \right)$$

This negative gradient can do really weird things:



Negative gradients and the squeezing effect

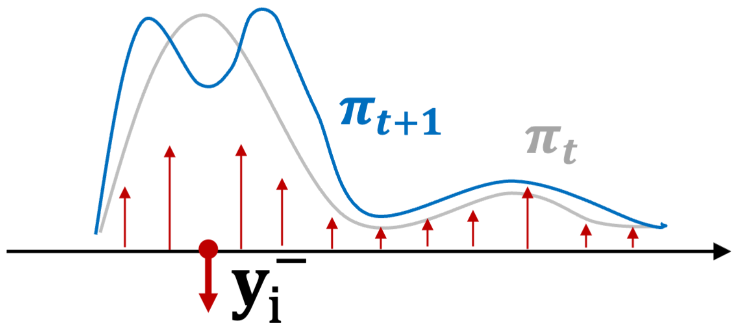
$$\pi(y \mid x) = \frac{\exp(z(x)_y)}{\exp(z(x)_y) + \exp(z(x)_{y^*}) + \dots}$$

- To decrease $\log \pi((y_i^-)_m \mid [\chi_i^-]_{:m})$, decrease numerator and increase denominator

Negative gradients and the squeezing effect

$$\pi(y \mid x) = \frac{\exp(z(x)_y)}{\exp(z(x)_y) + \exp(z(x)_{y^*}) + \dots}$$

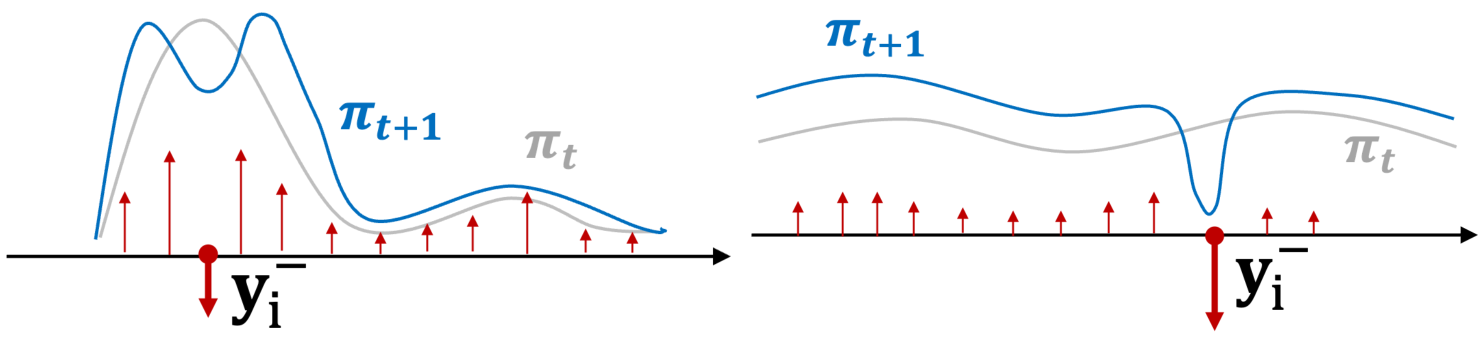
- To decrease $\log \pi((y_i^-)_m \mid [\chi_i^-]_{:m})$, decrease numerator and increase denominator



Negative gradients and the squeezing effect

$$\pi(y \mid x) = \frac{\exp(z(x)_y)}{\exp(z(x)_y) + \exp(z(x)_{y^*}) + \dots}$$

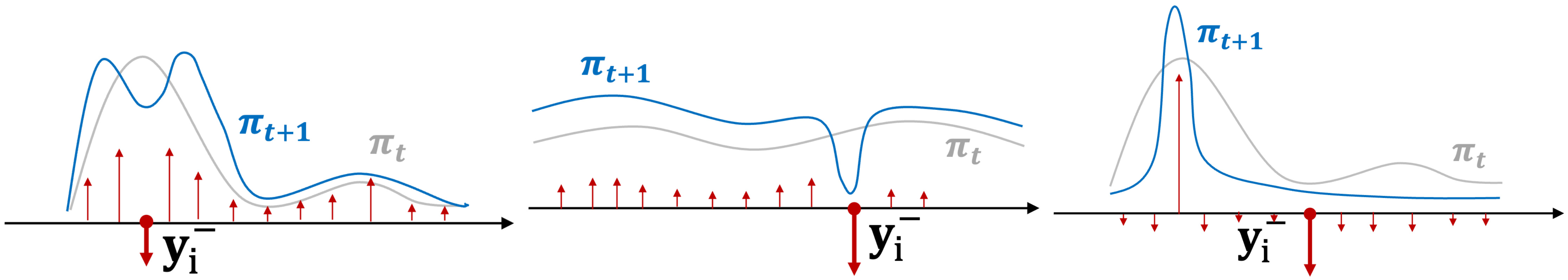
- To decrease $\log \pi((y_i^-)_m \mid [\chi_i^-]_{:m})$, decrease numerator and increase denominator



Negative gradients and the squeezing effect

$$\pi(y \mid x) = \frac{\exp(z(x)_y)}{\exp(z(x)_y) + \exp(z(x)_{y^*}) + \dots}$$

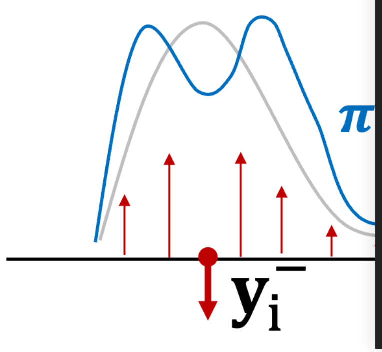
- To decrease $\log \pi((y_i^-)_m \mid [\chi_i^-]_{:m})$, decrease numerator and increase denominator
- If $z(x)_{y^*}$ is big, dominates the sum: increasing it is almost as effective as decreasing $z(x)_y$

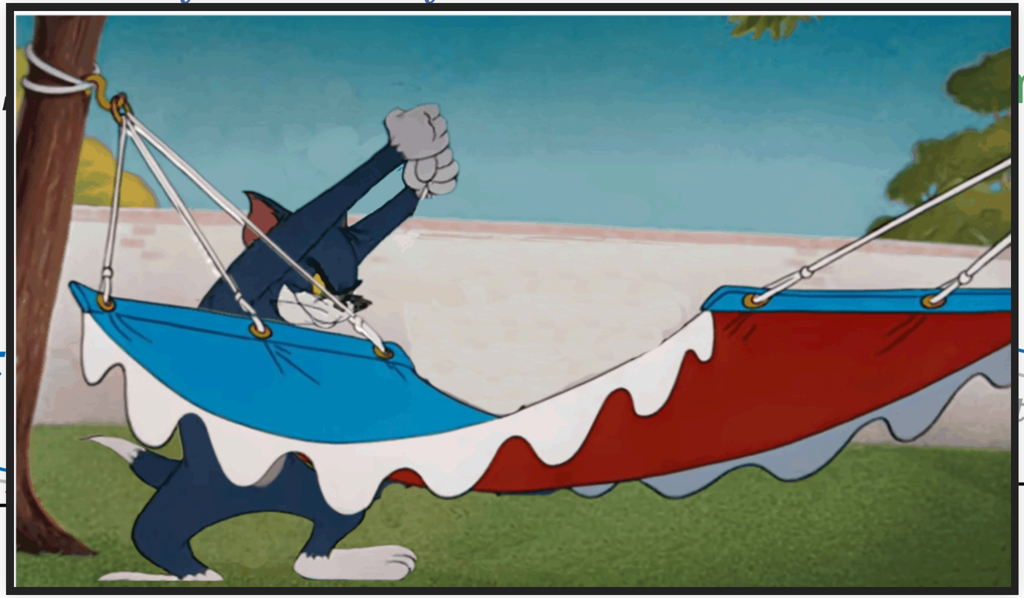


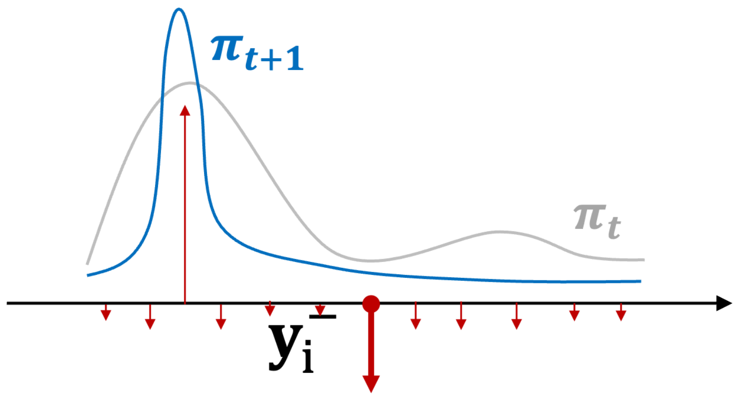
Negative gradients and the squeezing effect

$$\pi(y \mid x) = \frac{\exp(z(x)_y)}{\exp(z(x)_y) + \exp(z(x)_{y^*}) + \dots}$$

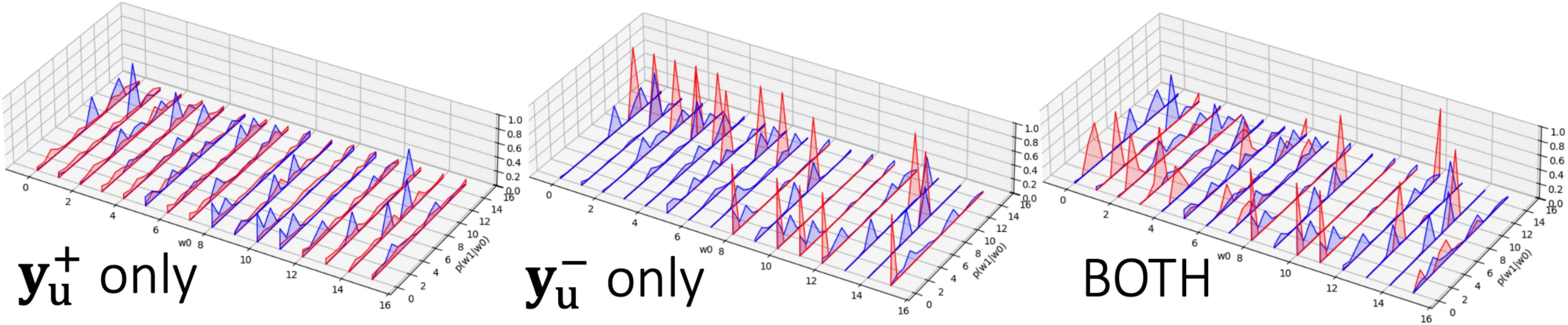
- To decrease $\log \pi((y_i^-)_m \mid [\chi_i^-]_{:m})$, decrease numerator and increase denominator
- If $z(x)_{y^*}$ is big, … nost as effective as decreasing $z(x)_y$



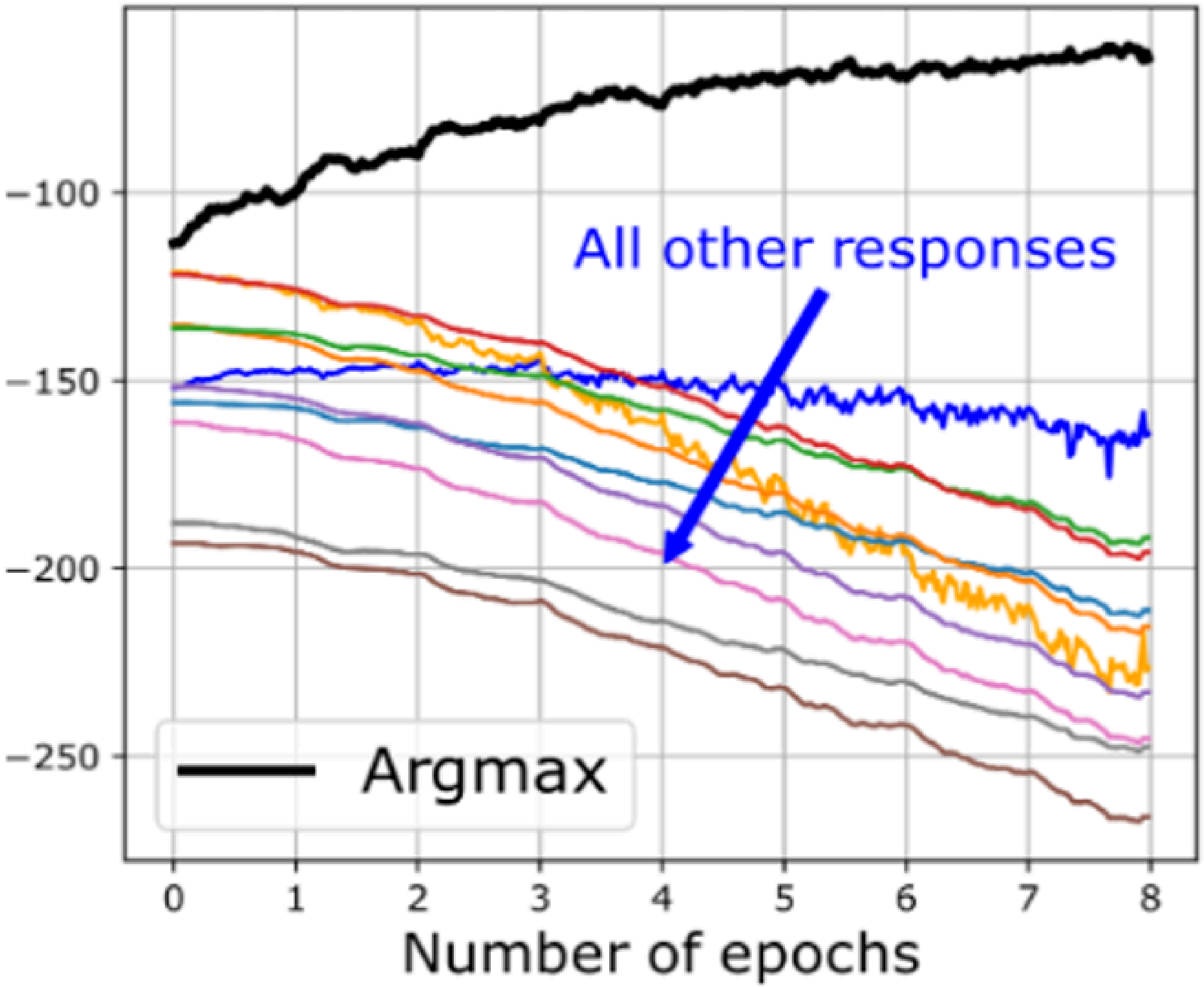




Positive gradients cancel out...*in the positive context*



Squeezing effect accumulates over time

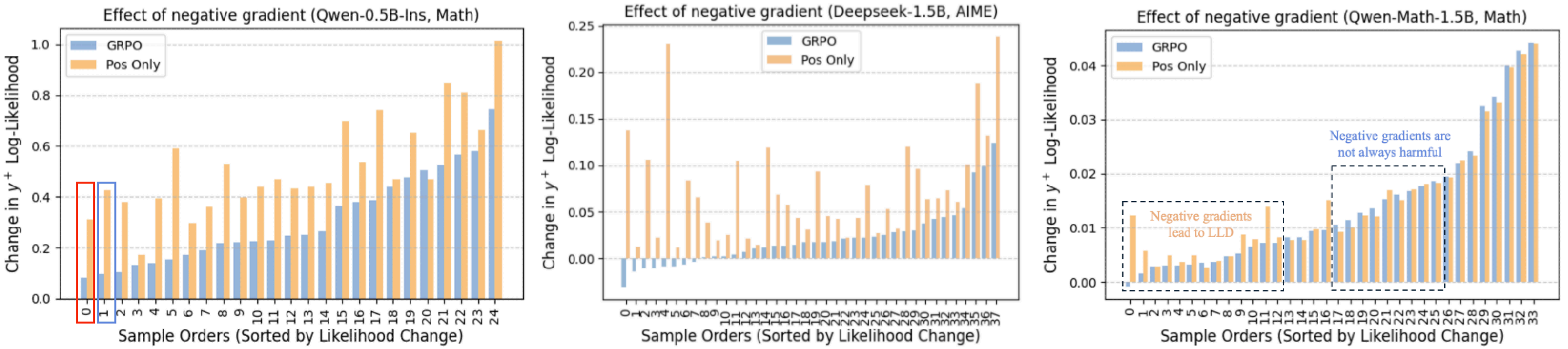


What can we learn from empirical NTKs?

- As a theoretical tool for local understanding:
  - Why DPO breaks
  - Why GRPO does weird stuff + how to fix
  - Fine-grained explanation for early stopping in knowledge distillation
  - How you should fine-tune models
- As a practical tool for approximating “lookahead” in active learning
- Plus: efficiently approximating $\mathcal{K}$s for large output dimensions k, with guarantees

Group Relative Policy Optimization (GRPO) [DeepSeekMath 24]

- Similar to a "group-wise" version of DPO; negative gradients have similar effect!



(a) Qwen-0.5B-Instruct (b) Deepseek-1.5B (c) Qwen-Math-1.5B

Figure 1: We show that negative gradients can lead to small or reduced likelihood change of positive samples in GRPO. The log-likelihood gains achieved by `Pos Only` training (orange) are significantly higher than those from GRPO (blue) for Qwen-0.5B-Ins (a) and Deepseek-1.5B (b). In Qwen-Math-1.5B (c), samples with small or reduced $\Delta(\boldsymbol{x})$ (left) are primarily influenced by negative gradients, as evidenced by their larger $\Delta(\boldsymbol{x})$ in the `Pos Only` setup. However, some samples on the right show smaller $\Delta(\boldsymbol{x})$ than in GRPO, indicating that negative gradients are not always harmful.

Negative token hidden rewards

Down-weight penalties on tokens that are probably okay

Base model + Method	AIME24	AMC	MATH500	Minerva	Olympiad	Avg.
Qwen2.5-Math-1.5B						
Base	3.3	20.0	39.6	7.7	24.9	19.10
GRPO	13.3	57.5	**71.8**	29.0	34.1	41.14
Pos Only	10.0	57.5	70.6	30.1	31.0	39.84
NTHR	**16.7**	**57.5**	70.8	**30.5**	**34.2**	**41.94**
Qwen2.5-0.5B-Ins						
Base	0.0	2.5	33.4	4.4	7.0	9.46
GRPO	0.0	7.5	33.8	**9.2**	8.1	11.72
NTHR	0.0	**10.0**	**36.6**	8.1	**8.6**	**12.66**
Qwen2.5-1.5B-Ins						
Base	0.0	22.5	53.0	19.1	20.7	23.06
GRPO	3.3	32.5	57.2	18.8	**23.0**	26.96
NTHR	**6.7**	**35.0**	**58.8**	**21.0**	20.9	**28.48**
Qwen2.5-Math-1.5B (deepscaler)						
Base	3.3	20.0	39.6	7.7	24.9	19.10
GRPO	10.0	42.5	72.4	**32.4**	**31.9**	37.80
NTHR	**16.7**	**47.5**	**73.2**	29.4	31.4	**39.60**
Qwen2.5-3B						
Base	10.0	37.5	58.6	26.1	24.6	31.36
GRPO	6.7	35.0	**66.6**	31.2	**29.9**	33.88
NTHR	**10.0**	**47.5**	65.6	**31.6**	26.8	**36.30**

Table 2: Results across selected math benchmarks for different Qwen2.5 models and methods. NTHR consistently provides average performance gains on various models.

What can we learn from empirical NTKs?

- As a theoretical tool for local understanding:
  - Why DPO breaks
  - Why GRPO does weird stuff + how to fix
  - Fine-grained explanation for early stopping in knowledge distillation
  - How you should fine-tune models
- As a practical tool for approximating “lookahead” in active learning
- Plus: efficiently approximating $\mathcal{K}$s for large output dimensions k, with guarantees

Better supervisory signal implies better learning

- Classification: target is $\mathcal{L}_P = \mathbb{E}_{(x,y)}\, \mathcal{L}(x, y) = \mathbb{E}_x\, \mathbb{E}_{y|x}\, \ell_y(f(x))$
- Normally: see $\{(x_i, y_i)\}$, minimize

$$\mathcal{L}_{\mathbf{X},\mathbf{y}} = \frac{1}{N} \sum_{i=1}^{N} \ell_{y_i}(f(x_i))$$

Better supervisory signal implies better learning

- Classification: target is $\mathcal{L}_P = \mathbb{E}_{(x,y)}\, \mathcal{L}(x, y) = \mathbb{E}_x\, \mathbb{E}_{y|x}\, \ell_y(f(x))$

- Normally: see $\{(x_i, y_i)\}$, minimize

$$\mathcal{L}_{\mathbf{X},\mathbf{y}} = \frac{1}{N}\sum_{i=1}^{N} \ell_{y_i}(f(x_i)) = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{k} \mathbb{I}(y_i = c)\ell_c(f(x_i))$$

Better supervisory signal implies better learning

- Classification: target is $\mathcal{L}_P = \mathbb{E}_{(x,y)}\, \mathcal{L}(x, y) = \mathbb{E}_x\, \mathbb{E}_{y|x}\, \ell_y(f(x))$
- Normally: see $\{(x_i, y_i)\}$, minimize ($\vec{\ell}(\hat{y}) \in \mathbb{R}^k$ is vector of losses for all possible labels)

$$\mathcal{L}_{\mathbf{X},\mathbf{y}} = \frac{1}{N}\sum_{i=1}^{N} \ell_{y_i}(f(x_i)) = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{k} \mathbb{I}(y_i = c)\ell_c(f(x_i)) = \frac{1}{N}\sum_{i=1}^{n} e_{y_i} \cdot \vec{\ell}(f(x_i))$$

Better supervisory signal implies better learning

- Classification: target is $\mathcal{L}_P = \mathbb{E}_{(x,y)}\, \mathcal{L}(x,y) = \mathbb{E}_x\, \mathbb{E}_{y|x}\, \ell_y(f(x))$
- Normally: see $\{(x_i, y_i)\}$, minimize ($\vec{\ell}(\hat{y}) \in \mathbb{R}^k$ is vector of losses for all possible labels)

$$\mathcal{L}_{\mathbf{X},\mathbf{y}} = \frac{1}{N}\sum_{i=1}^{N} \ell_{y_i}(f(x_i)) = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{k} \mathbb{I}(y_i = c)\ell_c(f(x_i)) = \frac{1}{N}\sum_{i=1}^{n} e_{y_i} \cdot \vec{\ell}(f(x_i))$$

- Potentially better scheme: see $\{(x_i, p_i^{tar})\}$, minimize $L^{tar}(f) = \frac{1}{N}\sum_{i=1}^{N} p_i^{tar} \cdot \vec{\ell}(f(x_i))$

Better supervisory signal implies better learning

- Classification: target is $\mathcal{L}_P = \mathbb{E}_{(x,y)}\,\mathcal{L}(x,y) = \mathbb{E}_x\,\mathbb{E}_{y|x}\,\ell_y(f(x))$
- Normally: see $\{(x_i, y_i)\}$, minimize ($\vec{\ell}(\hat{y}) \in \mathbb{R}^k$ is vector of losses for all possible labels)

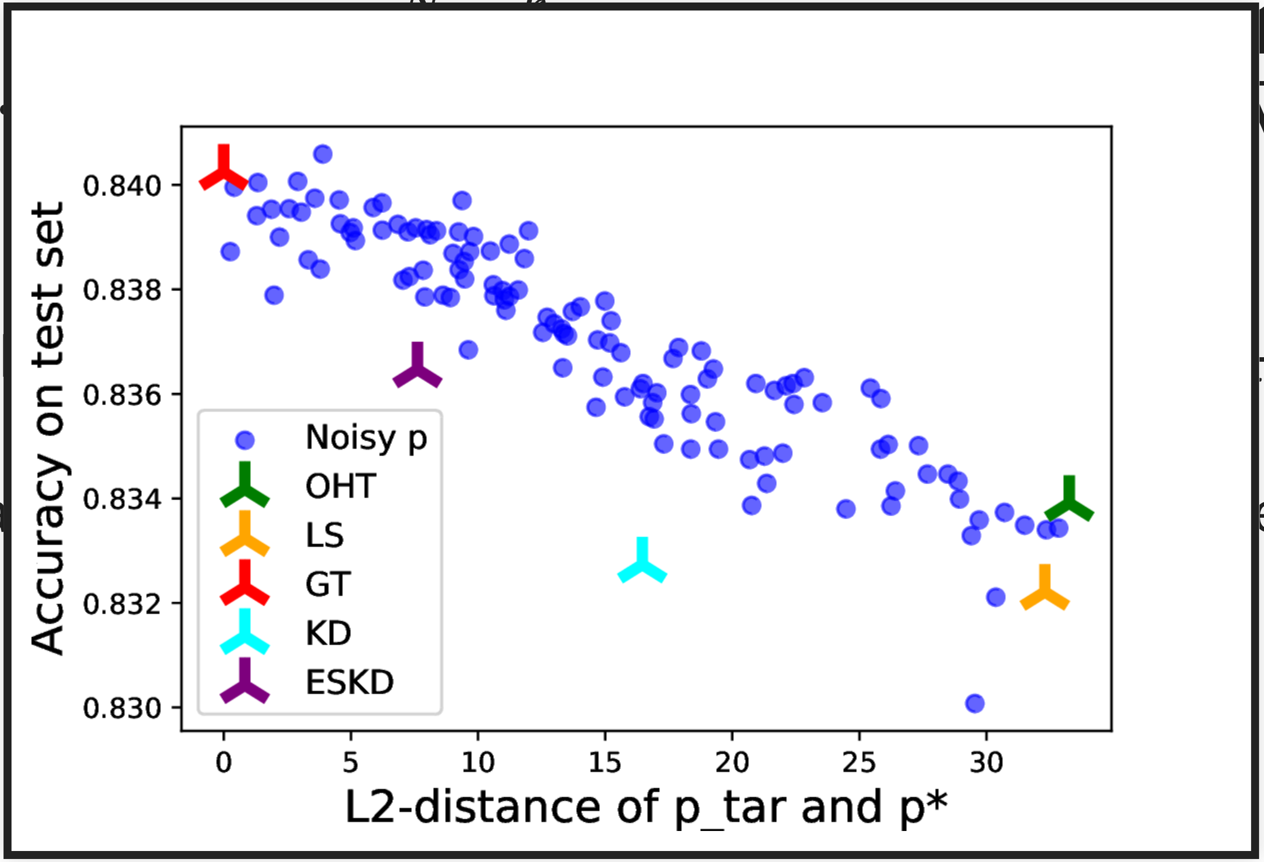
$$\mathcal{L}_{\mathbf{X},\mathbf{y}} = \frac{1}{N}\sum_{i=1}^{N}\ell_{y_i}(f(x_i)) = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{k}\mathbb{I}(y_i = c)\ell_c(f(x_i)) = \frac{1}{N}\sum_{i=1}^{n} e_{y_i}\cdot\vec{\ell}(f(x_i))$$

- Potentially better scheme: see $\{(x_i, p_i^{tar})\}$, minimize $L^{tar}(f) = \frac{1}{N}\sum_{i=1}^{N} p_i^{tar}\cdot\vec{\ell}(f(x_i))$
  - Can reduce variance if $p_i^{tar} \approx p_i^*$, the true conditional probabilities

Better supervisory signal implies better learning

- Classification: target is $\mathcal{L}_P = \mathbb{E}_{(x,y)}\,\mathcal{L}(x,y) = \mathbb{E}_x\,\mathbb{E}_{y|x}\,\ell_y(f(x))$
- Normally: see $\{(x_i, y_i)\}$, minimize ($\vec{\ell}(\hat{y}) \in \mathbb{R}^k$ is vector of losses for all possible labels)

$$\mathcal{L}_{\mathbf{X},\mathbf{y}} = \frac{1}{N}\sum_{i=1}^{N}\ell_{y_i}(\quad \ldots \quad \sum_{i=1}^{n} e_{y_i}\cdot\vec{\ell}(f(x_i))$$



- Potentially better sc … $\sum_{i=1}^{N} p_i^{tar}\cdot\vec{\ell}(f(x_i))$
  - Can reduce varia … es

Knowledge distillation

- Process:
  - Train a teacher $f^{teacher}$ on $\{(x_i, y_i)\}$ with standard ERM, $L(f)$
  - Train a student on $\{(x_i, f^{teacher}(x_i))\}$ with L^{tar}
- Usually $f^{student}$ is “smaller” than $f^{teacher}$

Knowledge distillation

- Process:
  - Train a teacher $f^{teacher}$ on $\{(x_i, y_i)\}$ with standard ERM, $L(f)$
  - Train a student on $\{(x_i, f^{teacher}(x_i))\}$ with L^{tar}
- Usually $f^{student}$ is “smaller” than $f^{teacher}$
- But “self-distillation” (using the same architecture), often $f^{student}$ outperforms $f^{teacher}$!

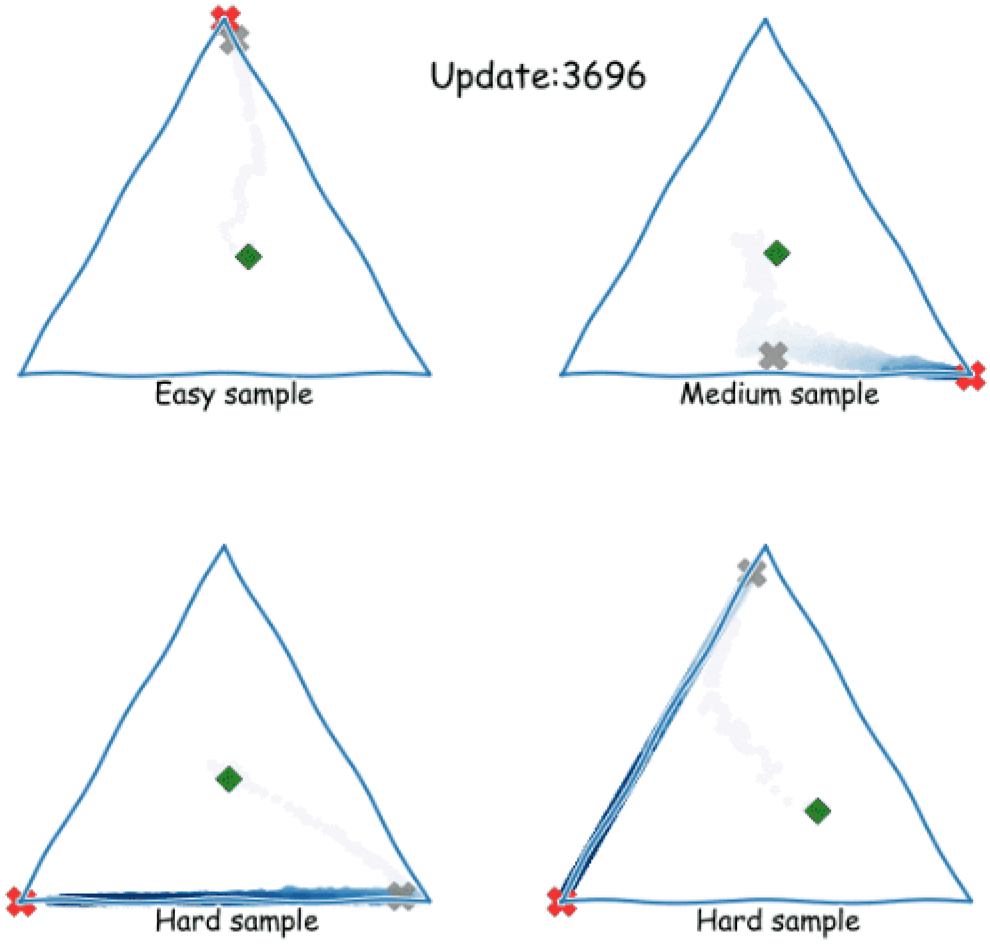
Knowledge distillation

- Process:
  - Train a teacher $f^{teacher}$ on $\{(x_i, y_i)\}$ with standard ERM, $L(f)$
  - Train a student on $\{(x_i, f^{teacher}(x_i))\}$ with L^{tar}
- Usually $f^{student}$ is “smaller” than $f^{teacher}$
- But “self-distillation” (using the same architecture), often $f^{student}$ outperforms $f^{teacher}$!
- One possible explanation: $f^{teacher}(x_i)$ is closer to p_i^* than sampled y_i

Knowledge distillation

- Process:
  - Train a teacher $f^{teacher}$ on $\{(x_i, y_i)\}$ with standard ERM, $L(f)$
  - Train a student on $\{(x_i, f^{teacher}(x_i))\}$ with L^{tar}
- Usually $f^{student}$ is “smaller” than $f^{teacher}$
- But “self-distillation” (using the same architecture), often $f^{student}$ outperforms $f^{teacher}$!
- One possible explanation: $f^{teacher}(x_i)$ is closer to p_i^* than sampled y_i
- But why would that be?

Zig-Zagging behaviour in learning



Plots of (three-way) probabilistic predictions: × shows p_i^*, × shows y_i

eNTK explains it

- Let $q_t(\tilde{x}) = \mathrm{softmax}(f_t(\tilde{x})) \in \mathbb{R}^k$; for cross-entropy loss, one SGD step gives us

$$q_{t+1}(\tilde{x}) - q_t(\tilde{x}) = \eta\, \mathcal{A}_t(\tilde{x})\, \mathcal{K}_{\mathbf{w}_t}(\tilde{x}, x_i)\, (p_i^{tar} - q_t(x_i)) + \mathcal{O}(\eta^2)$$

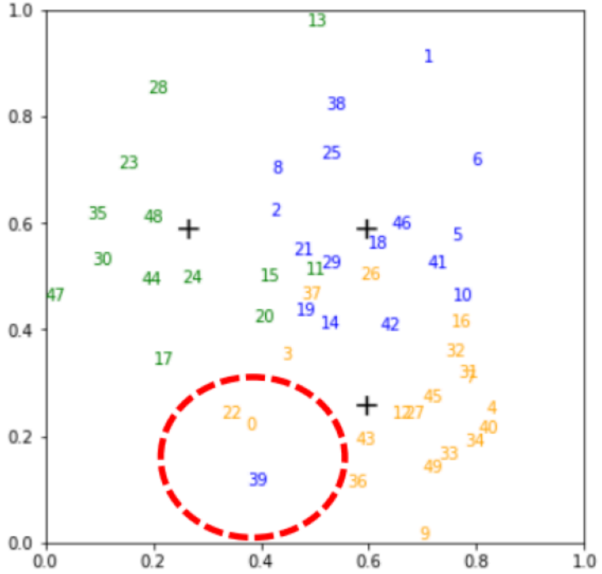
$\mathcal{A}_t(\tilde{x}) = \mathrm{diag}(q_t(\tilde{x})) - q_t(\tilde{x}) q_t(\tilde{x})^\mathsf{T}$ is the covariance of a $\mathrm{Categorical}(q_t(\tilde{x}))$

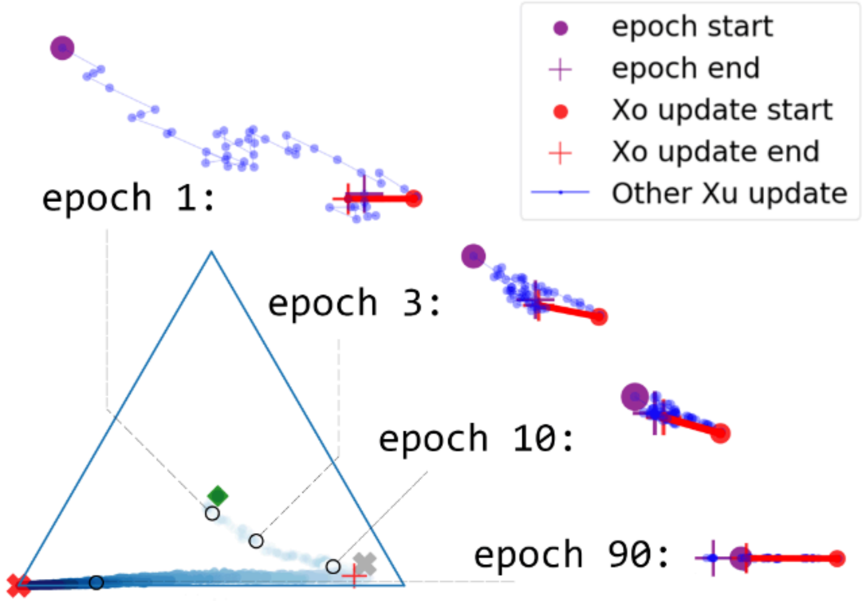
eNTK explains it

- Let $q_t(\tilde{x}) = \operatorname{softmax}(f_t(\tilde{x})) \in \mathbb{R}^k$; for cross-entropy loss, one SGD step gives us

$$q_{t+1}(\tilde{x}) - q_t(\tilde{x}) = \eta\, \mathcal{A}_t(\tilde{x})\ \mathcal{K}_{\mathbf{w}_t}(\tilde{x}, x_i)\,(p_i^{tar} - q_t(x_i)) + \mathcal{O}(\eta^2)$$

$\mathcal{A}_t(\tilde{x}) = \operatorname{diag}(q_t(\tilde{x})) - q_t(\tilde{x})q_t(\tilde{x})^\mathsf{T}$ is the covariance of a $\operatorname{Categorical}(q_t(\tilde{x}))$



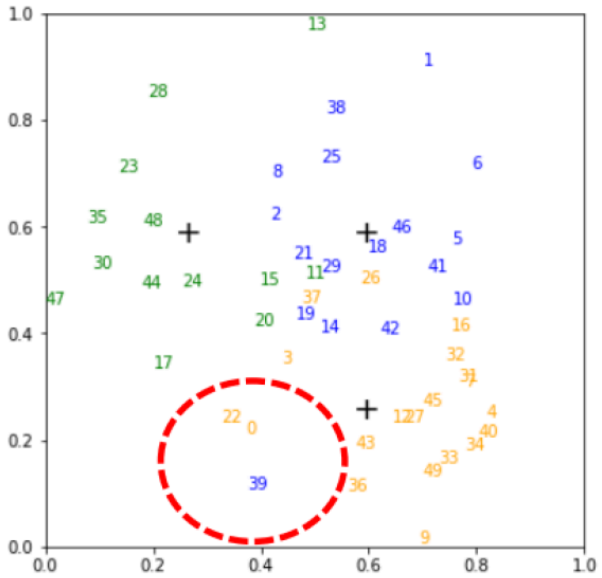


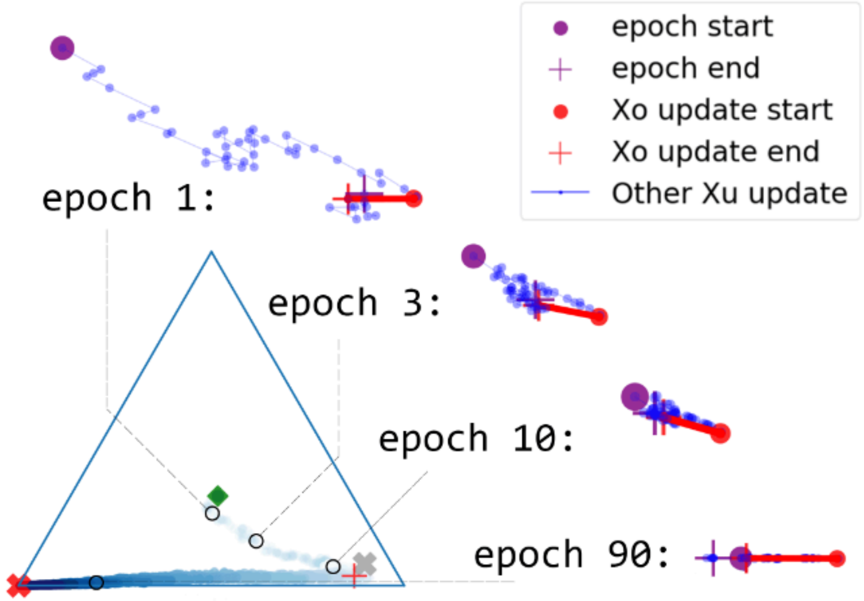
eNTK explains it

- Let $q_t(\tilde{x}) = \mathrm{softmax}(f_t(\tilde{x})) \in \mathbb{R}^k$; for cross-entropy loss, one SGD step gives us

$$q_{t+1}(\tilde{x}) - q_t(\tilde{x}) = \eta\, \mathcal{A}_t(\tilde{x})\; \mathcal{K}_{\mathbf{w}_t}(\tilde{x}, x_i)\, (p_i^{tar} - q_t(x_i)) + \mathcal{O}(\eta^2)$$

$\mathcal{A}_t(\tilde{x}) = \mathrm{diag}(q_t(\tilde{x})) - q_t(\tilde{x}) q_t(\tilde{x})^\mathsf{T}$ is the covariance of a $\mathrm{Categorical}(q_t(\tilde{x}))$



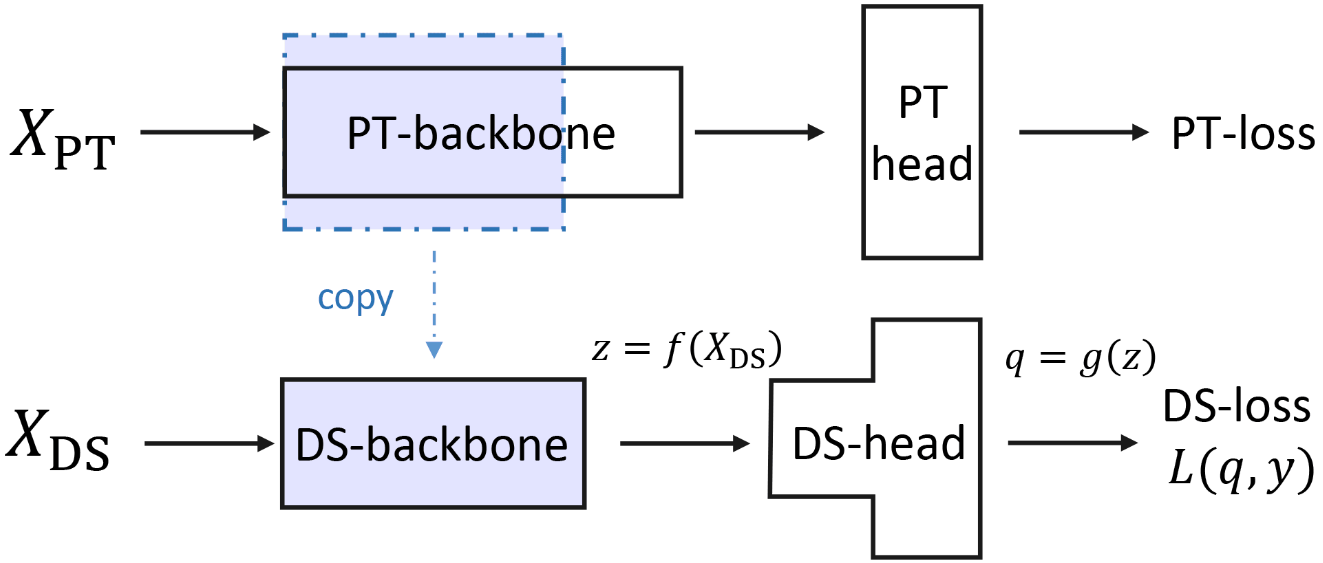


- Improves distillation (esp. with noisy labels) to take moving average of $q_t(x_i)$ as p_i^{tar}

What can we learn from empirical NTKs?

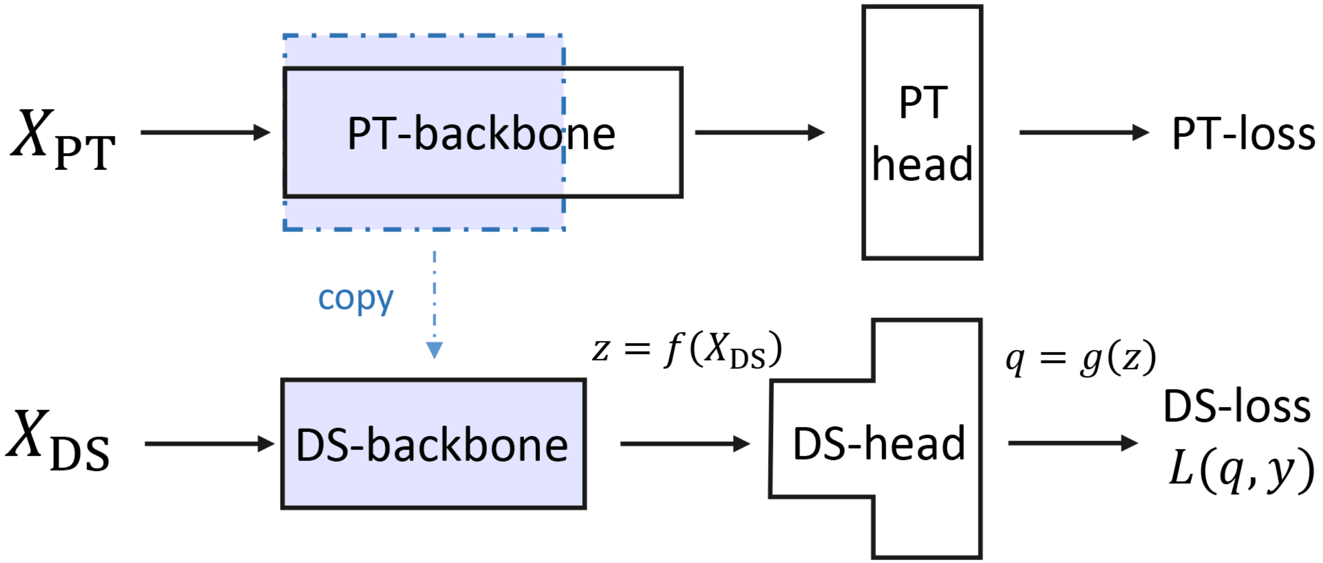
- As a theoretical tool for local understanding:
  - Why DPO breaks
  - Why GRPO does weird stuff + how to fix
  - Fine-grained explanation for early stopping in knowledge distillation
  - How you should fine-tune models
- As a practical tool for approximating “lookahead” in active learning
- Plus: efficiently approximating $\mathcal{K}$s for large output dimensions k, with guarantees

Fine-tuning



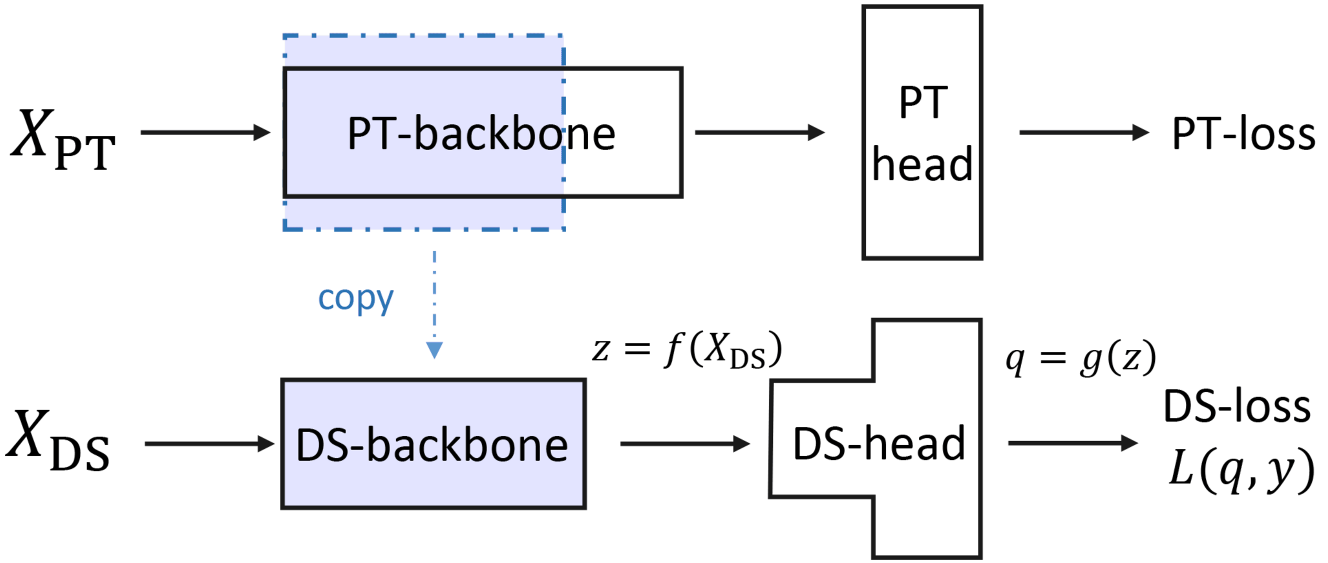
- Pretrain, re-initialize a random head, then adapt to a downstream task. Two phases:
  - Head probing: only update the head $g(z)$
  - Fine-tuning: update head $g(z)$ and backbone $z = f(x)$ together

Fine-tuning



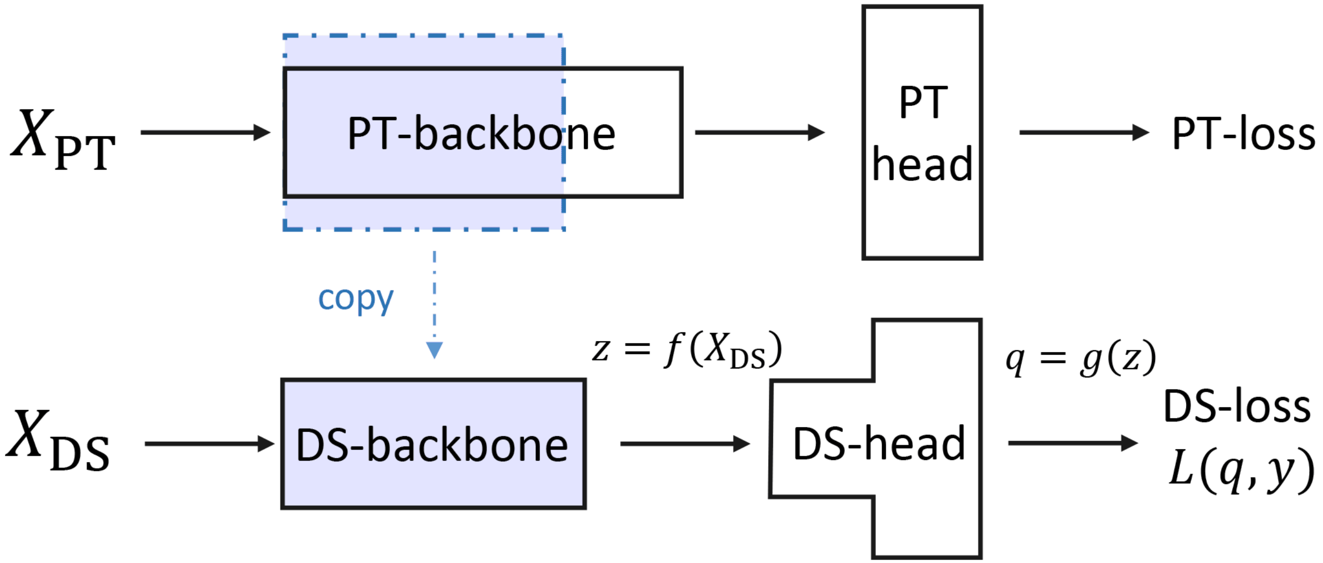
- Pretrain, re-initialize a random head, then adapt to a downstream task. Two phases:
  - Head probing: only update the head $g(z)$
  - Fine-tuning: update head $g(z)$ and backbone $z = f(x)$ together
- If we only fine-tune: noise from random head might break our features!

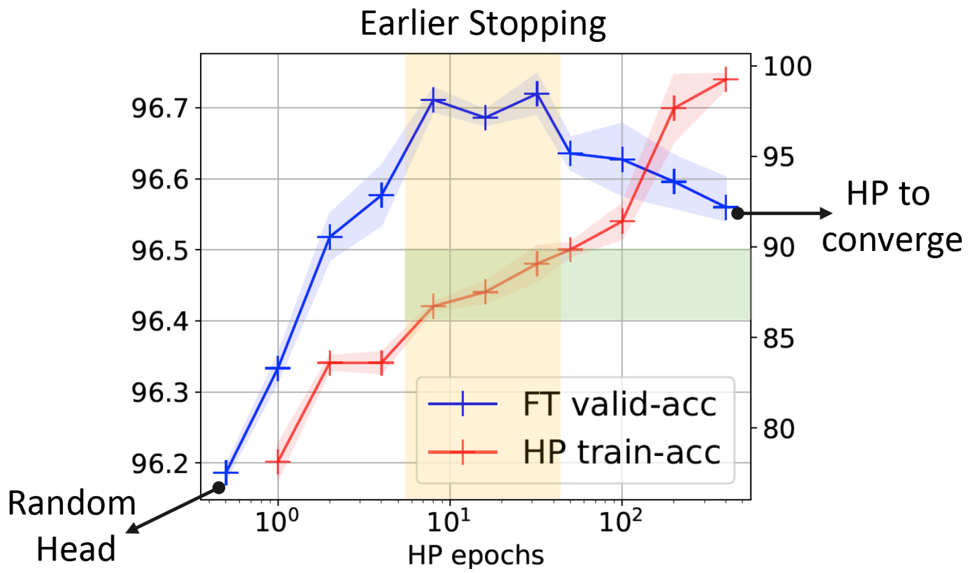
Fine-tuning



- Pretrain, re-initialize a random head, then adapt to a downstream task. Two phases:
  - Head probing: only update the head $g(z)$
  - Fine-tuning: update head $g(z)$ and backbone $z = f(x)$ together
- If we only fine-tune: noise from random head might break our features!
- If we head-probe to convergence: might already fit training data and not change features!

Fine-tuning





- Pretrain, re-initialize a random head, then adapt to a downstream task. Two phases:
  - Head probing: only update the head $g(z)$
  - Fine-tuning: update head $g(z)$ and backbone $z = f(x)$ together
- If we only fine-tune: noise from random head might break our features!
- If we head-probe to convergence: might already fit training data and not change features!

How much do we change our features?

- Same kind of decomposition with backbone features $z = f(x)$, head $q = \mathrm{softmax}(g(z))$:

$$z_{t+1}(\tilde{x}) - z_t(\tilde{x}) = \frac{\eta}{N} \sum_{i=1}^{N} \underbrace{\mathcal{K}^f_{\mathbf{w}_t}(\tilde{x}, x_i)}_{\text{eNTK of backbone}} \underbrace{(\nabla_z q_t(x_i))^\mathsf{T}}_{\text{direction of head}} \underbrace{(e_{y_i} - q_t(x_i))}_{\text{“energy”}} + \mathcal{O}(\eta^2)$$

How much do we change our features?

- Same kind of decomposition with backbone features $z = f(x)$, head $q = \operatorname{softmax}(g(z))$:

$$z_{t+1}(\tilde{x}) - z_t(\tilde{x}) = \frac{\eta}{N} \sum_{i=1}^{N} \underbrace{\mathcal{K}^f_{\mathbf{w}_t}(\tilde{x}, x_i)}_{\text{eNTK of backbone}} \underbrace{(\nabla_z q_t(x_i))^\mathsf{T}}_{\text{direction of head}} \underbrace{(e_{y_i} - q_t(x_i))}_{\text{“energy”}} + \mathcal{O}(\eta^2)$$

- If initial “energy”, e.g. $\mathbb{E}_{x_i, y_i} \left\| e_{y_i} - p_0(x_i) \right\|$, is small, features don't change much

How much do we change our features?

- Same kind of decomposition with backbone features $z = f(x)$, head $q = \mathrm{softmax}(g(z))$:

$$z_{t+1}(\tilde{x}) - z_t(\tilde{x}) = \frac{\eta}{N} \sum_{i=1}^{N} \underbrace{\mathcal{K}^f_{\mathbf{w}_t}(\tilde{x}, x_i)}_{\text{eNTK of backbone}} \underbrace{(\nabla_z q_t(x_i))^\mathsf{T}}_{\text{direction of head}} \underbrace{(e_{y_i} - q_t(x_i))}_{\text{“energy”}} + \mathcal{O}(\eta^2)$$

- If initial “energy”, e.g. $\mathbb{E}_{x_i, y_i} \left\| e_{y_i} - p_0(x_i) \right\|$, is small, features don't change much
- If we didn't do any head probing, “direction” is very random, especially if g is rich

How much do we change our features?

- Same kind of decomposition with backbone features $z = f(x)$, head $q = \mathrm{softmax}(g(z))$:

$$z_{t+1}(\tilde{x}) - z_t(\tilde{x}) = \frac{\eta}{N} \sum_{i=1}^{N} \underbrace{\mathcal{K}^f_{\mathbf{w}_t}(\tilde{x}, x_i)}_{\text{eNTK of backbone}} \underbrace{(\nabla_z q_t(x_i))^\mathsf{T}}_{\text{direction of head}} \underbrace{(e_{y_i} - q_t(x_i))}_{\text{“energy”}} + \mathcal{O}(\eta^2)$$

- If initial “energy”, e.g. $\mathbb{E}_{x_i, y_i} \| e_{y_i} - p_0(x_i) \|$, is small, features don't change much
- If we didn't do any head probing, “direction” is very random, especially if g is rich
- Specializing to simple linear-linear model, can get insights about trends in z

How much do we change our features?

- Same kind of decomposition with backbone features $z = f(x)$, head $q = \mathrm{softmax}(g(z))$:

$$z_{t+1}(\tilde{x}) - z_t(\tilde{x}) = \frac{\eta}{N} \sum_{i=1}^{N} \underbrace{\mathcal{K}^f_{\mathbf{w}_t}(\tilde{x}, x_i)}_{\text{eNTK of backbone}} \underbrace{(\nabla_z q_t(x_i))^\mathsf{T}}_{\text{direction of head}} \underbrace{(e_{y_i} - q_t(x_i))}_{\text{“energy”}} + \mathcal{O}(\eta^2)$$

- If initial “energy”, e.g. $\mathbb{E}_{x_i, y_i} \left\| e_{y_i} - p_0(x_i) \right\|$, is small, features don't change much
- If we didn't do any head probing, “direction” is very random, especially if g is rich
- Specializing to simple linear-linear model, can get insights about trends in z
- Recommendations from paper:
  - Early stop during head probing (ideally, try multiple lengths for downstream task)
  - Label smoothing can help; so can more complex heads, but be careful

How good will our fine-tuned features be? [Wei/Hu/Steinhardt 2022]

What can we learn from empirical NTKs?

- As a theoretical tool for local understanding:
  - Why DPO breaks
  - Why GRPO does weird stuff + how to fix
  - Fine-grained explanation for early stopping in knowledge distillation
  - How you should fine-tune models
- As a practical tool for approximating “lookahead” in active learning
- Plus: efficiently approximating $\mathcal{K}$s for large output dimensions k, with guarantees

Pool-based active learning

Pool-based active learning

Pool-based active learning

Pool-based active learning

Pool-based active learning

Approximate retraining with local linearization

- Given $f_{\mathcal{L}}$ trained on labeled data $\mathcal{L}$, approximate $f_{\mathcal{L} \cup \{(x_i, y_i)\}}$ with local linearization

Approximate retraining with local linearization

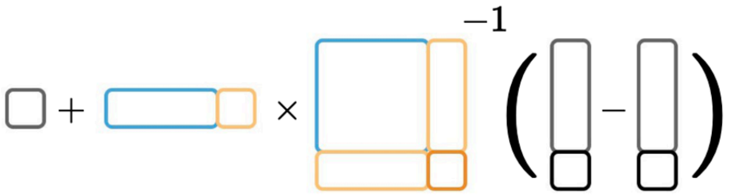
- Given $f_{\mathcal{L}}$ trained on labeled data $\mathcal{L}$, approximate $f_{\mathcal{L}\cup\{(x_i,y_i)\}}$ with local linearization

$$f_{\mathcal{L}\cup\{(x_i,y_i)\}}(\tilde{x}) \approx f_{\mathcal{L}}(\tilde{x}) + \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\tilde{x}, \begin{bmatrix}\mathbf{X}_{\mathcal{L}} \\ x_i\end{bmatrix}\right) \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\begin{bmatrix}\mathbf{X}_{\mathcal{L}} \\ x_i\end{bmatrix}, \begin{bmatrix}\mathbf{X}_{\mathcal{L}} \\ x_i\end{bmatrix}\right)^{-1} \left(\begin{bmatrix}\mathbf{y}_{\mathcal{L}} \\ y_i\end{bmatrix} - f_{\mathcal{L}}\left(\begin{bmatrix}\mathbf{X}_{\mathcal{L}} \\ x_i\end{bmatrix}\right)\right)$$

Approximate retraining with local linearization

- Given $f_{\mathcal{L}}$ trained on labeled data $\mathcal{L}$, approximate $f_{\mathcal{L}\cup\{(x_i,y_i)\}}$ with local linearization

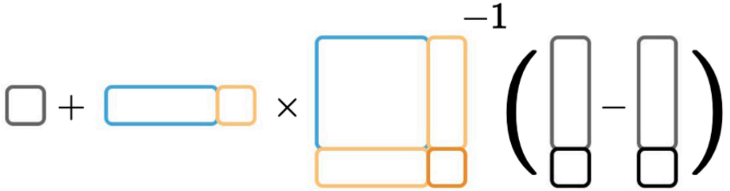
$$f_{\mathcal{L}\cup\{(x_i,y_i)\}}(\tilde{x}) \approx f_{\mathcal{L}}(\tilde{x}) + \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\tilde{x}, \begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right) \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}, \begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right)^{-1} \left(\begin{bmatrix} \mathbf{y}_{\mathcal{L}} \\ y_i \end{bmatrix} - f_{\mathcal{L}}\left(\begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right)\right)$$

  - Rank-one updates for efficient computation: schema 

Approximate retraining with local linearization

- Given $f_{\mathcal{L}}$ trained on labeled data $\mathcal{L}$, approximate $f_{\mathcal{L}\cup\{(x_i,y_i)\}}$ with local linearization

$$f_{\mathcal{L}\cup\{(x_i,y_i)\}}(\tilde{x}) \approx f_{\mathcal{L}}(\tilde{x}) + \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\tilde{x}, \begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right) \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}, \begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right)^{-1} \left(\begin{bmatrix} \mathbf{y}_{\mathcal{L}} \\ y_i \end{bmatrix} - f_{\mathcal{L}}\left(\begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right)\right)$$

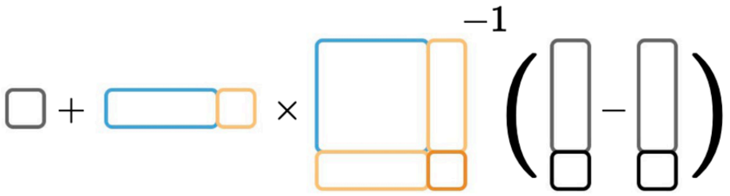
  - Rank-one updates for efficient computation: schema 

- We prove this is exact for infinitely wide networks
  - $f_0 \to f_{\mathcal{L}} \to f_{\mathcal{L}\cup\{(x_i,y_i)\}}$ agrees with direct $f_0 \to f_{\mathcal{L}\cup\{(x_i,y_i)\}}$

Approximate retraining with local linearization

- Given $f_{\mathcal{L}}$ trained on labeled data $\mathcal{L}$, approximate $f_{\mathcal{L}\cup\{(x_i,y_i)\}}$ with local linearization

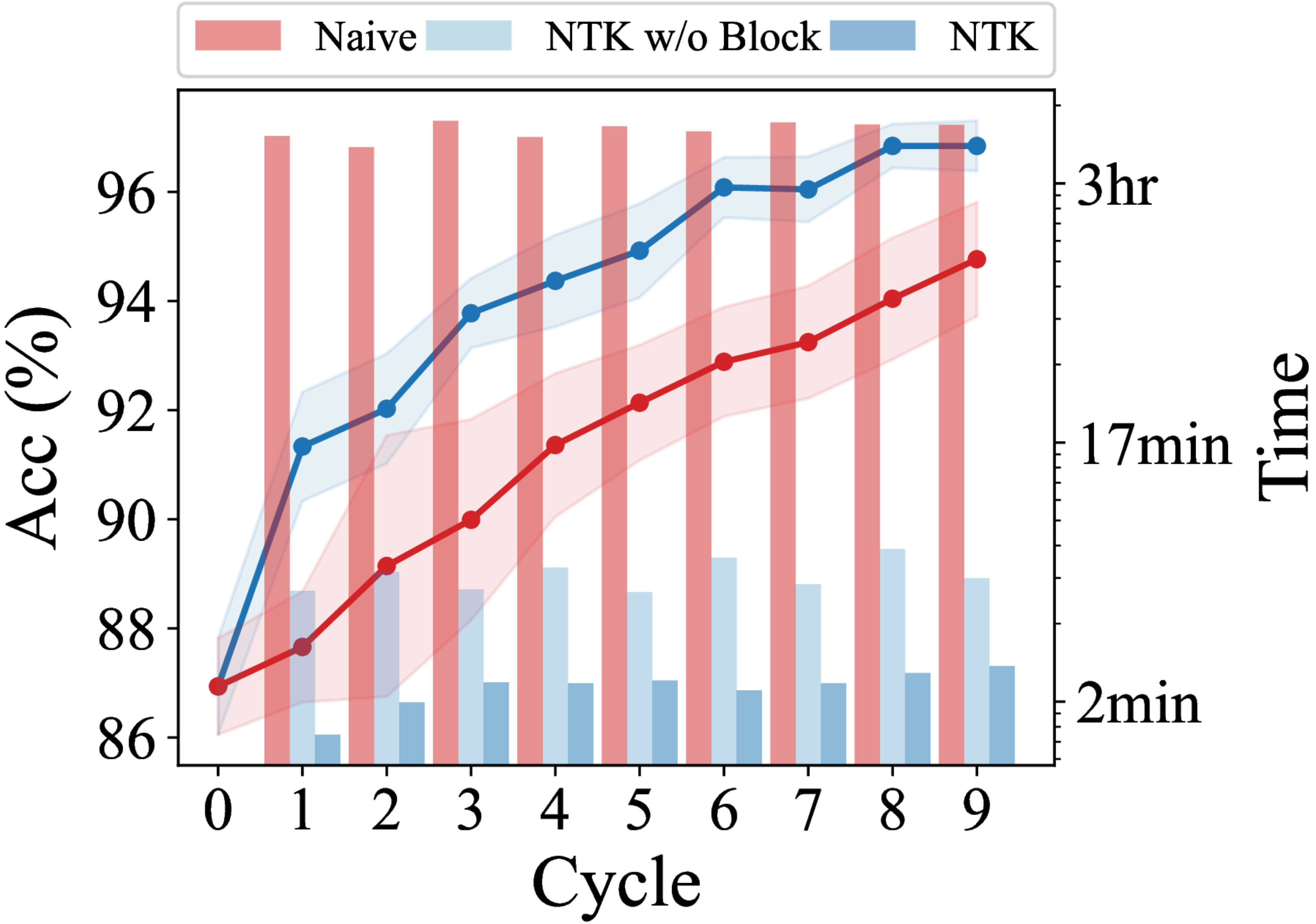
$$f_{\mathcal{L}\cup\{(x_i,y_i)\}}(\tilde{x}) \approx f_{\mathcal{L}}(\tilde{x}) + \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\tilde{x}, \begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right) \mathcal{K}_{\mathbf{w}_{\mathcal{L}}}\left(\begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}, \begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right)^{-1} \left(\begin{bmatrix} \mathbf{y}_{\mathcal{L}} \\ y_i \end{bmatrix} - f_{\mathcal{L}}\left(\begin{bmatrix} \mathbf{X}_{\mathcal{L}} \\ x_i \end{bmatrix}\right)\right)$$

  - Rank-one updates for efficient computation: schema 

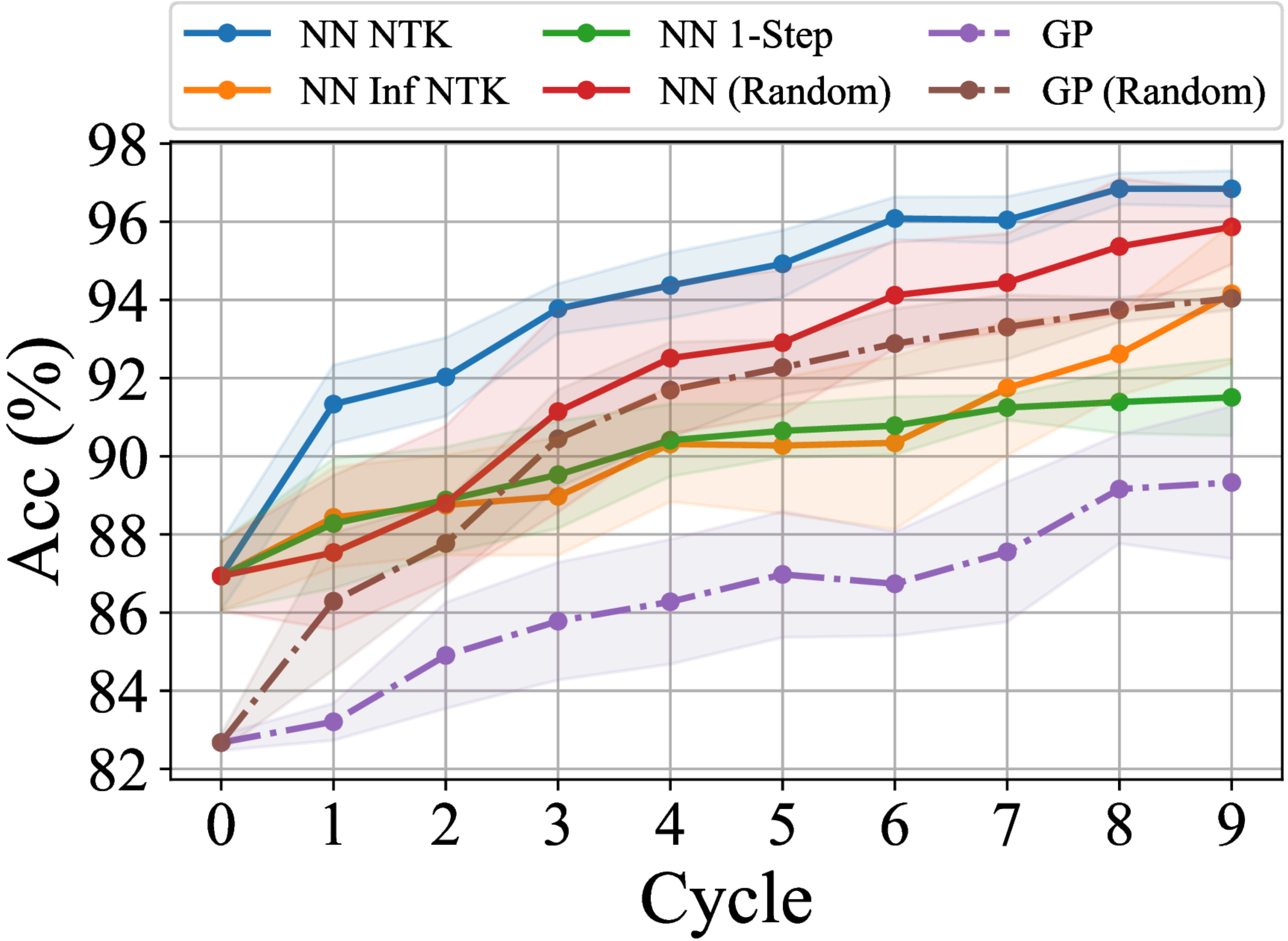
- We prove this is exact for infinitely wide networks
  - $f_0 \to f_{\mathcal{L}} \to f_{\mathcal{L}\cup\{(x_i,y_i)\}}$ agrees with direct $f_0 \to f_{\mathcal{L}\cup\{(x_i,y_i)\}}$

- Local approximation with eNTK “should” work much more broadly than “NTK regime”

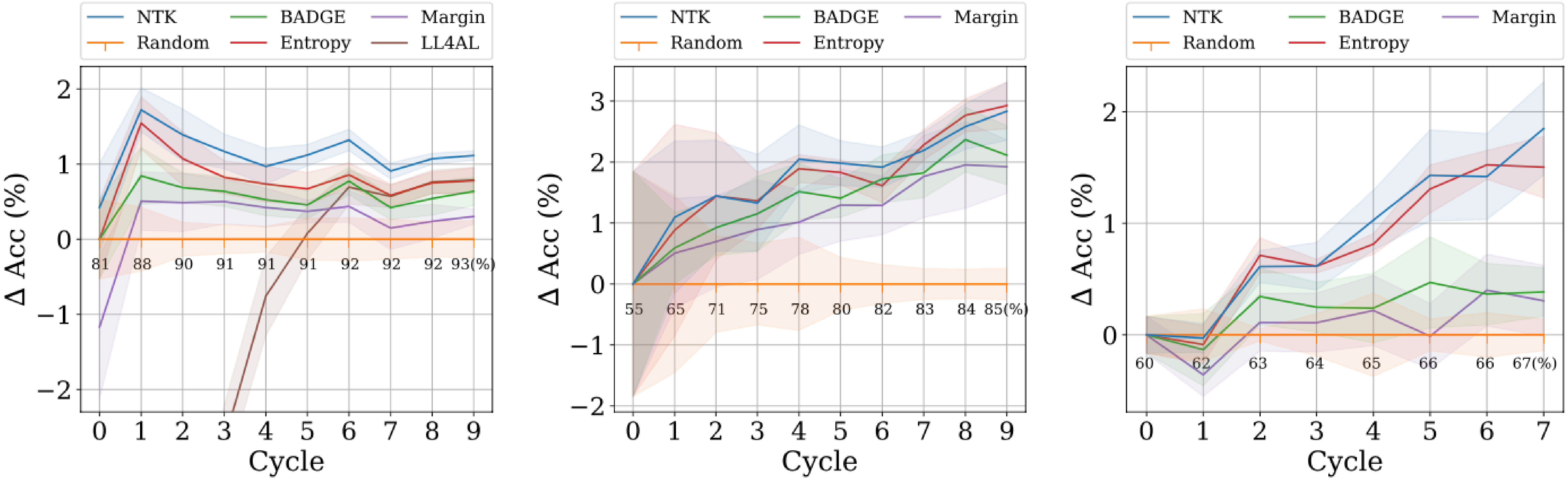
Much faster than SGD



Much more effective than infinite NTK and one-step SGD



Matches/beats state of the art



(a) SVHN: 1-layer WideResNet (b) CIFAR10: 2-layer WideResNet (c) CIFAR100: ResNet18

Figure 2: Comparison of the-state-of-the-art active learning methods on various benchmark datasets. Vertical axis shows difference from random acquisition, whose accuracy is shown in text.

Downside: usually more computationally expensive (especially memory)

Enables new interaction modes

What can we learn from empirical NTKs?

- As a theoretical tool for local understanding:
  - Why DPO breaks
  - Why GRPO does weird stuff + how to fix
  - Fine-grained explanation for early stopping in knowledge distillation
  - How you should fine-tune models
- As a practical tool for approximating “lookahead” in active learning
- Plus: efficiently approximating $\mathcal{K}$s for large output dimensions k, with guarantees

Approximating empirical NTKs

- I hid something from you on active learning (and Wei/Hu/Steinhardt fine-tuning) results…

Approximating empirical NTKs

- I hid something from you on active learning (and Wei/Hu/Steinhardt fine-tuning) results…
- With k classes, $\mathcal{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{kN \times kN}$ – potentially very big

Approximating empirical NTKs

- I hid something from you on active learning (and Wei/Hu/Steinhardt fine-tuning) results...
- With k classes, $\mathcal{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{kN \times kN}$ – potentially very big
- But actually, we know that $\mathbb{E}_{\mathbf{w}} \mathcal{K}_{\mathbf{w}}(x_1, x_2)$ is diagonal for most architectures
  - Let $\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = \underbrace{[\nabla_{\mathbf{w}} f_1(x_1)]}_{1 \times p} \underbrace{[\nabla_{\mathbf{w}} f_1(x_2))^{\mathsf{T}}}_{p \times 1}$.

$\mathbb{E}_{\mathbf{w}} \mathcal{K}_{\mathbf{w}}(x_1, x_2) = \mathbb{E}_{w} [\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2)] I_k$. $\mathrm{pNTK}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$ (no k!)

Approximating empirical NTKs

- I hid something from you on active learning (and Wei/Hu/Steinhardt fine-tuning) results...
- With k classes, $\mathcal{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{kN \times kN}$ – potentially very big
- But actually, we know that $\mathbb{E}_{\mathbf{w}}\, \mathcal{K}_{\mathbf{w}}(x_1, x_2)$ is diagonal for most architectures
  - Let $\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = \underbrace{[\nabla_{\mathbf{w}} f_1(x_1)]}_{1\times p}\underbrace{[\nabla_{\mathbf{w}} f_1(x_2)]^\mathsf{T}}_{p\times 1}$.

    $\mathbb{E}_{\mathbf{w}}\, \mathcal{K}_{\mathbf{w}}(x_1, x_2) = \mathbb{E}_{w}[\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2)] I_k$. $\quad \mathrm{pNTK}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$ (no k!)
  - Can also use "sum of logits" $\frac{1}{\sqrt{k}} \sum_{j=1}^{k} f_j$ instead of just "first logit" f_1

Approximating empirical NTKs

- I hid something from you on active learning (and Wei/Hu/Steinhardt fine-tuning) results...
- With k classes, $\mathcal{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{kN \times kN}$ – potentially very big
- But actually, we know that $\mathbb{E}_{\mathbf{w}}\, \mathcal{K}_{\mathbf{w}}(x_1, x_2)$ is diagonal for most architectures
  - Let $\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = \underbrace{[\nabla_{\mathbf{w}} f_1(x_1)]}_{1\times p}\underbrace{[\nabla_{\mathbf{w}} f_1(x_2)]^{\mathsf{T}}}_{p\times 1}$.

$$\mathbb{E}_{\mathbf{w}}\, \mathcal{K}_{\mathbf{w}}(x_1, x_2) = \mathbb{E}_w[\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2)] I_k. \qquad \mathrm{pNTK}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N} \text{ (no } k\text{!)}$$

  - Can also use "sum of logits" $\frac{1}{\sqrt{k}} \sum_{j=1}^{k} f_j$ instead of just "first logit" f_1
- Lots of work (including above) has used pNTK instead of $\mathcal{K}$
  - Often without saying anything; sometimes doesn't seem like they know they're doing it

Approximating empirical NTKs

- I hid something from you on active learning (and Wei/Hu/Steinhardt fine-tuning) results...
- With k classes, $\mathcal{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{kN \times kN}$ – potentially very big
- But actually, we know that $\mathbb{E}_{\mathbf{w}}\, \mathcal{K}_{\mathbf{w}}(x_1, x_2)$ is diagonal for most architectures
  - Let $\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = \underbrace{[\nabla_{\mathbf{w}} f_1(x_1)]}_{1\times p}\underbrace{[\nabla_{\mathbf{w}} f_1(x_2))^\mathsf{T}}_{p\times 1}$.

    $\mathbb{E}_{\mathbf{w}}\, \mathcal{K}_{\mathbf{w}}(x_1, x_2) = \mathbb{E}_w[\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2)] I_k$. $\quad \mathrm{pNTK}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$ (no k!)
  - Can also use "sum of logits" $\frac{1}{\sqrt{k}} \sum_{j=1}^{k} f_j$ instead of just "first logit" f_1
- Lots of work (including above) has used pNTK instead of $\mathcal{K}$
  - Often without saying anything; sometimes doesn't seem like they know they're doing it
- Can we justify this more rigorously?

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k\times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k \times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries
- If $v_{j,i} \sim \mathcal{N}(0, \sigma^2)$, then v_1 and $\frac{1}{\sqrt{k}} \sum_{j=1}^{k} v_j$ have same distribution

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k\times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries
- If $v_{j,i} \sim \mathcal{N}(0, \sigma^2)$, then v_1 and $\frac{1}{\sqrt{k}} \sum_{j=1}^{k} v_j$ have same distribution

$$\mathcal{K}_{\mathbf{w}}(x_1, x_2)_{jj'} = v_j^\mathsf{T} \, \mathcal{K}^{\phi}_{\mathbf{w}\setminus V}(x_1, x_2) \, v_{j'} + \mathbb{I}(j = j')\phi(x_1)^\mathsf{T} \phi(x_2)$$

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k\times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries
- If $v_{j,i} \sim \mathcal{N}(0, \sigma^2)$, then v_1 and $\frac{1}{\sqrt{k}} \sum_{j=1}^k v_j$ have same distribution

$$\mathcal{K}_{\mathbf{w}}(x_1, x_2)_{jj'} = v_j^\mathsf{T} \mathcal{K}^{\phi}_{\mathbf{w}\setminus V}(x_1, x_2)\, v_{j'} + \mathbb{I}(j = j')\phi(x_1)^\mathsf{T}\phi(x_2)$$

$$\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = {\color{red}v_1^\mathsf{T}}\, \mathcal{K}^{\phi}_{\mathbf{w}\setminus V}(x_1, x_2)\, {\color{red}v_1} + \phi(x_1)^\mathsf{T}\phi(x_2)$$

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k\times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries
- If $v_{j,i} \sim \mathcal{N}(0, \sigma^2)$, then v_1 and $\frac{1}{\sqrt{k}} \sum_{j=1}^{k} v_j$ have same distribution

$$\mathcal{K}_{\mathbf{w}}(x_1, x_2)_{jj'} = v_j^\mathsf{T} \mathcal{K}^{\phi}_{\mathbf{w}\setminus V}(x_1, x_2)\, v_{j'} + \mathbb{I}(j = j')\phi(x_1)^\mathsf{T}\phi(x_2)$$

$$\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = v_1^\mathsf{T} \mathcal{K}^{\phi}_{\mathbf{w}\setminus V}(x_1, x_2)\, v_1 + \phi(x_1)^\mathsf{T}\phi(x_2)$$

- We want to bound difference $\mathcal{K}(x_1, x_2) - \mathrm{pNTK}(x_1, x_2) I_k$

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k \times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries
- If $v_{j,i} \sim \mathcal{N}(0, \sigma^2)$, then v_1 and $\frac{1}{\sqrt{k}} \sum_{j=1}^k v_j$ have same distribution

$$\mathcal{K}_{\mathbf{w}}(x_1, x_2)_{jj'} = v_j^\top \mathcal{K}^\phi_{\mathbf{w}\setminus V}(x_1, x_2)\, v_{j'} + \mathbb{I}(j = j')\phi(x_1)^\top \phi(x_2)$$

$$\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = {\color{red}v_1^\top} \mathcal{K}^\phi_{\mathbf{w}\setminus V}(x_1, x_2)\, {\color{red}v_1} + \phi(x_1)^\top \phi(x_2)$$

- We want to bound difference $\mathcal{K}(x_1, x_2) - \mathrm{pNTK}(x_1, x_2) I_k$
  - Want $v_1^\top A v_1$ and $v_j^\top A v_j$ to be close, and $v_j^\top A v_{j'}$ small, for random v and fixed A

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k\times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries
- If $v_{j,i} \sim \mathcal{N}(0, \sigma^2)$, then v_1 and $\frac{1}{\sqrt{k}}\sum_{j=1}^k v_j$ have same distribution

$$\mathcal{K}_{\mathbf{w}}(x_1, x_2)_{jj'} = v_j^\top \mathcal{K}^{\phi}_{\mathbf{w}\setminus V}(x_1, x_2)\, v_{j'} + \mathbb{I}(j = j')\phi(x_1)^\top \phi(x_2)$$
$$\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = v_1^\top \mathcal{K}^{\phi}_{\mathbf{w}\setminus V}(x_1, x_2)\, v_1 + \phi(x_1)^\top \phi(x_2)$$

- We want to bound difference $\mathcal{K}(x_1, x_2) - \mathrm{pNTK}(x_1, x_2) I_k$
  - Want $v_1^\top A v_1$ and $v_j^\top A v_j$ to be close, and $v_j^\top A v_{j'}$ small, for random v and fixed A
  - Using Hanson-Wright: $\frac{\|\mathcal{K} - \mathrm{pNTK}\, I\|_F}{\|\mathcal{K}\|_F} \leq \frac{\left\|\mathcal{K}^{\phi}\right\|_F + 4\sqrt{h}}{\mathrm{Tr}(\mathcal{K}^{\phi})} k \log \frac{2k^2}{\delta}$

pNTK motivation

- Say $f(x) = V\phi(x)$, $\phi(x) \in \mathbb{R}^h$, and $V \in \mathbb{R}^{k\times h}$ has rows $v_j \in \mathbb{R}^h$ with iid entries
- If $v_{j,i} \sim \mathcal{N}(0, \sigma^2)$, then v_1 and $\frac{1}{\sqrt{k}} \sum_{j=1}^k v_j$ have same distribution

$$\mathcal{K}_{\mathbf{w}}(x_1, x_2)_{jj'} = v_j^\mathsf{T} \mathcal{K}^\phi_{\mathbf{w}\setminus V}(x_1, x_2)\, v_{j'} + \mathbb{I}(j = j')\phi(x_1)^\mathsf{T}\phi(x_2)$$
$$\mathrm{pNTK}_{\mathbf{w}}(x_1, x_2) = v_1^\mathsf{T} \mathcal{K}^\phi_{\mathbf{w}\setminus V}(x_1, x_2)\, v_1 + \phi(x_1)^\mathsf{T}\phi(x_2)$$

- We want to bound difference $\mathcal{K}(x_1, x_2) - \mathrm{pNTK}(x_1, x_2) I_k$
  - Want $v_1^\mathsf{T} A v_1$ and $v_j^\mathsf{T} A v_j$ to be close, and $v_j^\mathsf{T} A v_{j'}$ small, for random v and fixed A
  - Using Hanson-Wright: $\frac{\lVert \mathcal{K} - \mathrm{pNTK}\, I \rVert_F}{\lVert \mathcal{K} \rVert_F} \le \frac{\lVert \mathcal{K}^\phi \rVert_F + 4\sqrt{h}}{\mathrm{Tr}(\mathcal{K}^\phi)} k \log \frac{2k^2}{\delta}$
  - Fully-connected ReLU nets at init., fan-in mode: numerator $\mathcal{O}(h\sqrt{h})$, denom $\Theta(h^2)$

pNTK's Frobenius error

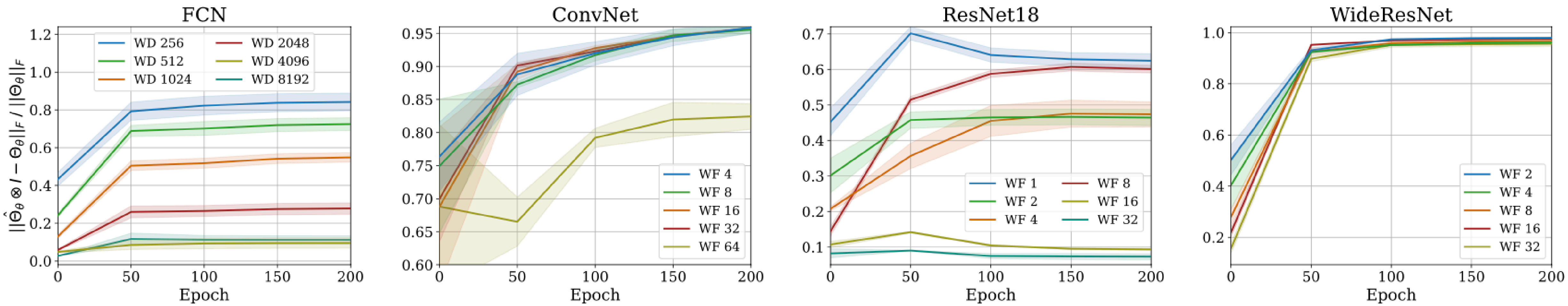


Figure 3: Evaluating the **relative difference of Frobenius norm of** $\Theta_\theta(\mathcal{D},\mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D},\mathcal{D}) \otimes I_O$ at initialization and throughout training, based on $\mathcal{D}$ being 1000 random points from CIFAR-10. Wider nets have more similar $\|\Theta_\theta\|_F$ and $\|\hat{\Theta}_\theta \otimes I_O\|_F$ at initialization.

pNTK's Frobenius error

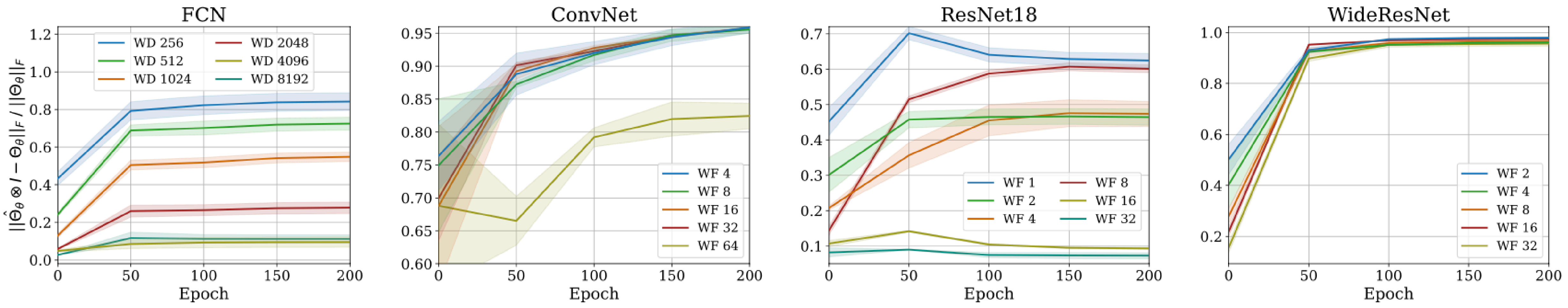


Figure 3: Evaluating the **relative difference of Frobenius norm of** $\Theta_\theta(\mathcal{D}, \mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D}, \mathcal{D}) \otimes I_O$ at initialization and throughout training, based on $\mathcal{D}$ being 1000 random points from CIFAR-10. Wider nets have more similar $\|\Theta_\theta\|_F$ and $\|\hat{\Theta}_\theta \otimes I_O\|_F$ at initialization.

Same kind of theorem / empirical results for largest eigenvalue,
and empirical results for $\lambda_{\min}$, condition number

Kernel regression with pNTK

- Reshape things to handle prediction appropriately:

$$\underbrace{f_{\mathcal{K}}(\tilde{x})}_{k\times 1} = \underbrace{f_0(\tilde{x})}_{k\times 1} + \underbrace{\mathcal{K}_{\mathbf{w}_0}(\tilde{x}, \mathbf{X})}_{k\times kN} \underbrace{\mathcal{K}_{\mathbf{w}_0}(\mathbf{X}, \mathbf{X})^{-1}}_{kN\times kN} \underbrace{(\mathbf{y} - f_0(\mathbf{X}))}_{kN\times 1}$$

$$\underbrace{f_{\text{pNTK}}(\tilde{x})}_{k\times 1} = \underbrace{f_0(\tilde{x})}_{k\times 1} + \left(\underbrace{\text{pNTK}_{\mathbf{w}_0}(\tilde{x}, \mathbf{X})}_{1\times N} \underbrace{\text{pNTK}_{\mathbf{w}_0}(\mathbf{X}, \mathbf{X})^{-1}}_{N\times N} \underbrace{(\mathbf{y} - f_0(\mathbf{X}))}_{N\times k}\right)^{\mathsf{T}}$$

- We have $\|f_{\mathcal{K}}(\tilde{x}) - f_{\text{pNTK}}(\tilde{x})\| = \mathcal{O}(\frac{1}{\sqrt{h}})$ again
  - If we add regularization, need to “scale” λ between the two

Kernel regression with pNTK

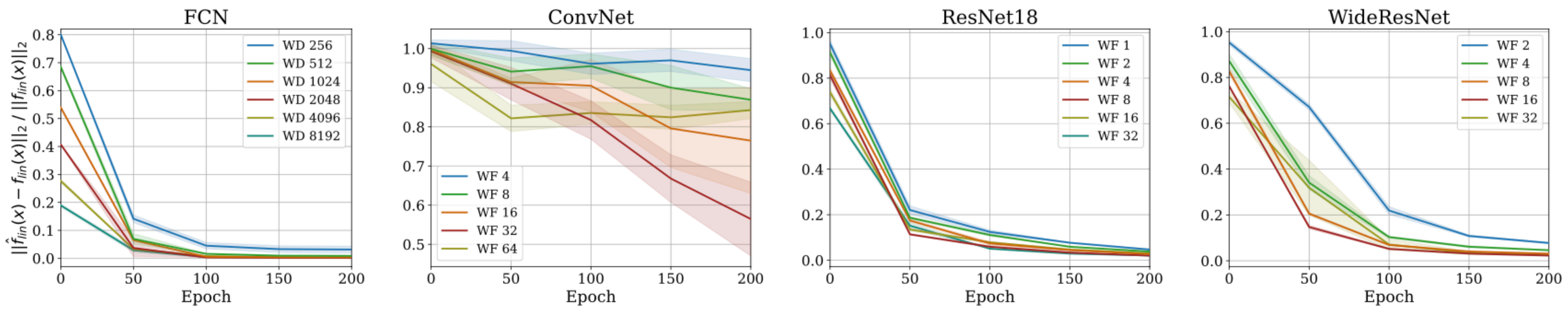


Figure 7: The **relative difference of kernel regression outputs**, (4) and (5), when training on $|\mathcal{D}| = 1000$ random CIFAR-10 points and testing on $|\mathcal{X}| = 500$. For wider NNs, the relative difference in $\hat{f}^{lin}(\mathcal{X})$ and $f^{lin}(\mathcal{X})$ decreases at initialization. Surprisingly, the difference between these two continues to quickly vanish while training the network.

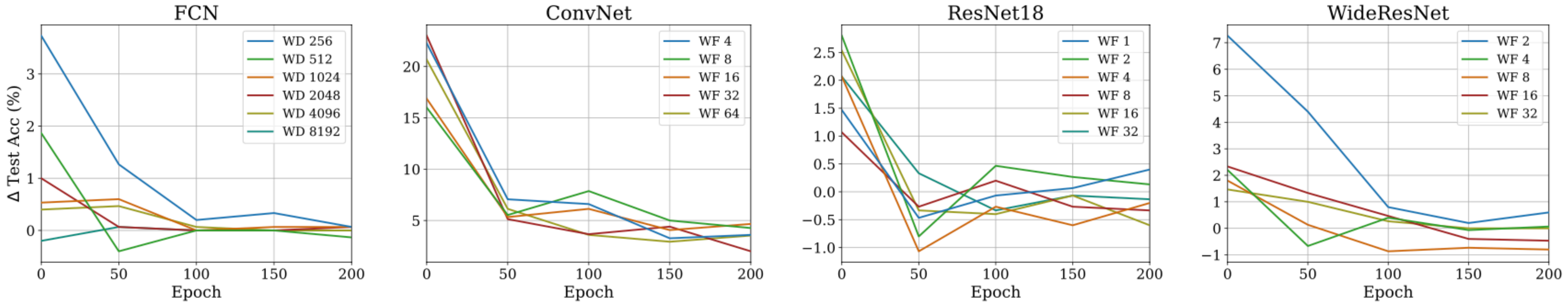


Figure 8: **Using pNTK in kernel regression** (as in Figure 7) **almost always achieves a higher test accuracy than using eNTK.** Wider NNs and trained nets have more similar prediction accuracies of $\hat{f}^{lin}$ and f^{lin} at initialization. Again, the difference between these two continues to vanish throughout the training process using SGD.

pNTK speed-up

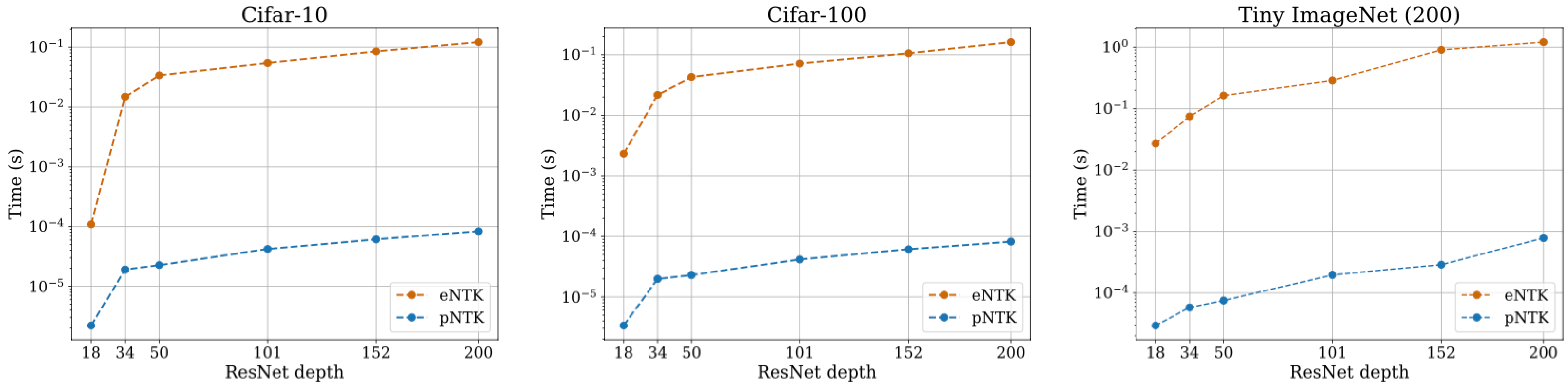
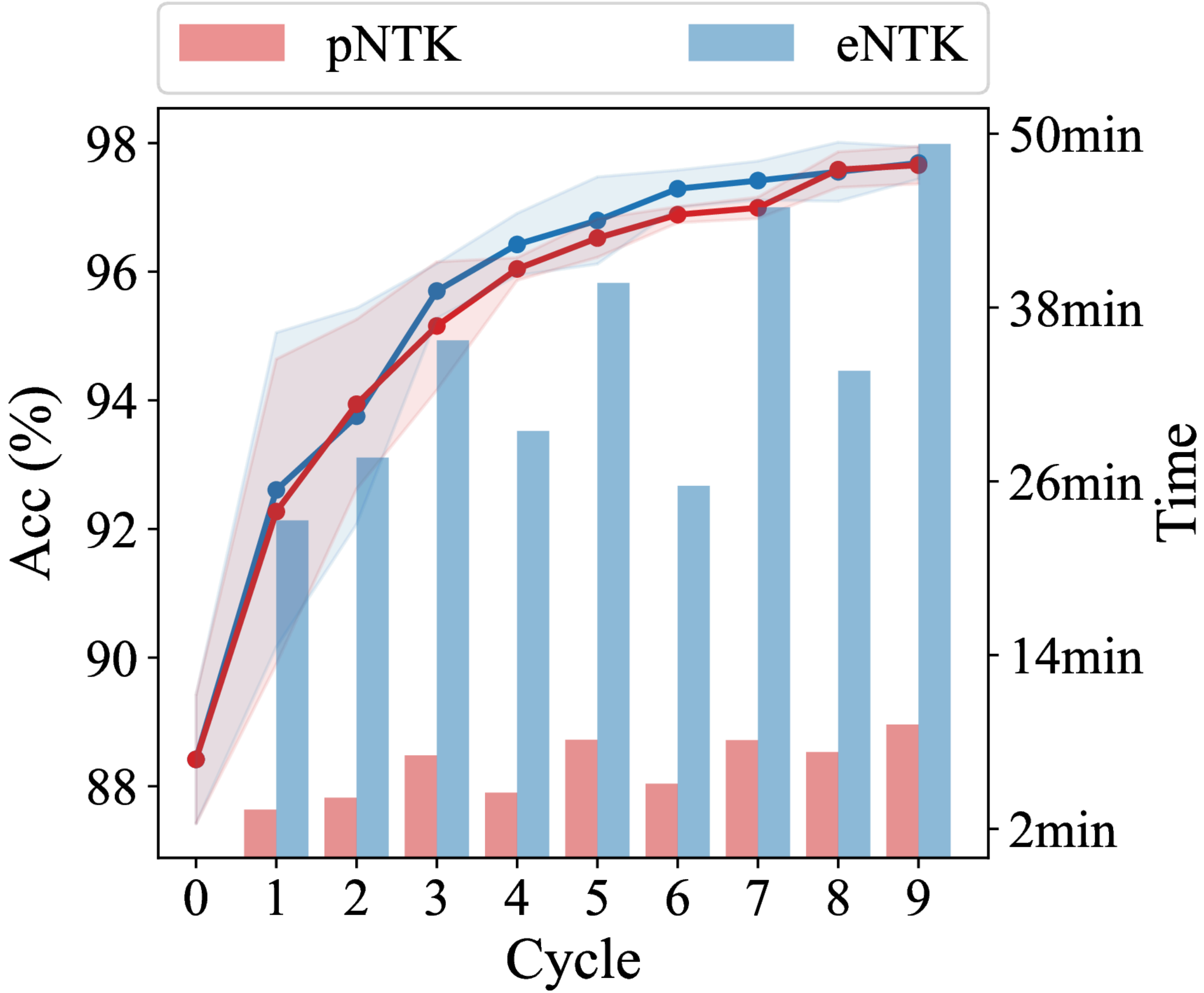


Figure 1: **Wall-clock time to evaluate the eNTK and pNTK** for one pair of inputs, across datasets and ResNet depths.

pNTK speed-up on active learning task



pNTK for full CIFAR-10 regression

- $\mathcal{K}(\mathbf{X}, \mathbf{X})$ on CIFAR-10: 1.8 terabytes of memory
- $\mathrm{pNTK}(\mathbf{X}, \mathbf{X})$ on CIFAR-10: 18 gigabytes of memory

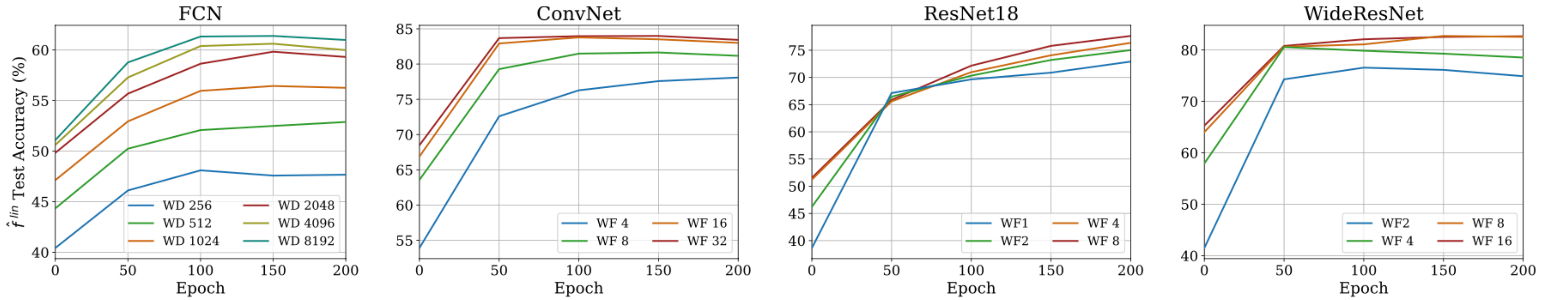


Figure 9: Evaluating the **test accuracy of kernel regression predictions using pNTK as in (5) on the full CIFAR-10 dataset**. As the NN's width grows, the test accuracy of $\hat{f}^{lin}$ also improves, but eventually saturates with the growing width. Using trained weights in computation of pNTK results in improved test accuracy of $\hat{f}^{lin}$.

pNTK for full CIFAR-10 regression

- $\mathcal{K}(\mathbf{X}, \mathbf{X})$ on CIFAR-10: 1.8 terabytes of memory
- $\mathrm{pNTK}(\mathbf{X}, \mathbf{X})$ on CIFAR-10: 18 gigabytes of memory

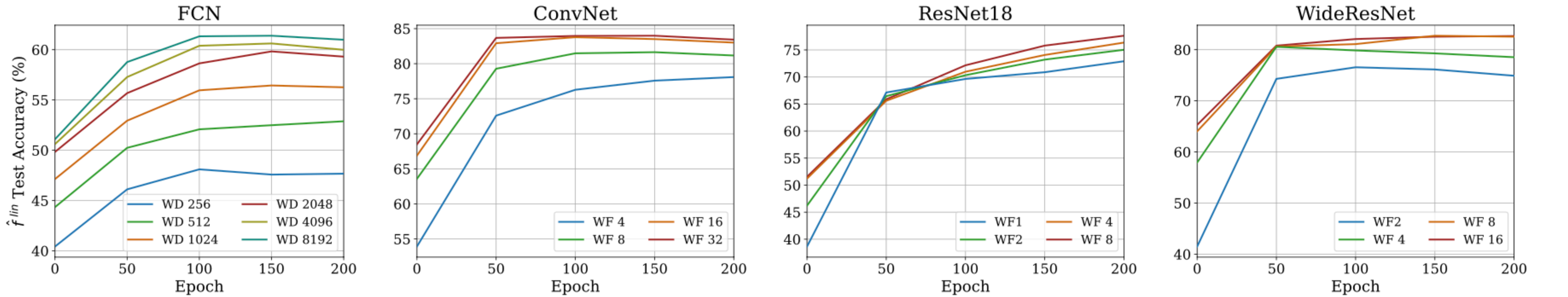


Figure 9: Evaluating the **test accuracy of kernel regression predictions using pNTK as in (5) on the full CIFAR-10 dataset**. As the NN's width grows, the test accuracy of $\hat{f}^{lin}$ also improves, but eventually saturates with the growing width. Using trained weights in computation of pNTK results in improved test accuracy of $\hat{f}^{lin}$.

- Worse than infinite NTK for FCN/ConvNet (where they can be computed, if you try hard)

pNTK for full CIFAR-10 regression

- $\mathcal{K}(\mathbf{X}, \mathbf{X})$ on CIFAR-10: 1.8 terabytes of memory
- $\mathrm{pNTK}(\mathbf{X}, \mathbf{X})$ on CIFAR-10: 18 gigabytes of memory

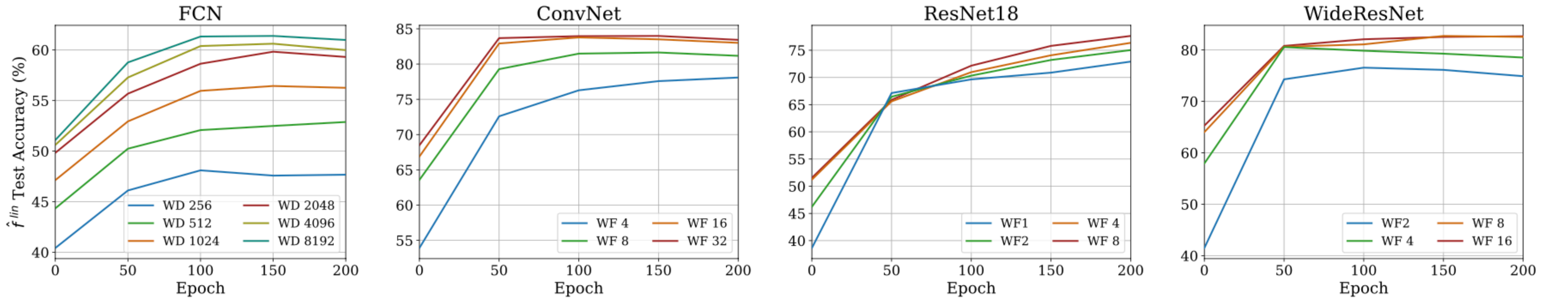


Figure 9: Evaluating the **test accuracy of kernel regression predictions using pNTK as in (5) on the full CIFAR-10 dataset**. As the NN's width grows, the test accuracy of $\hat{f}^{lin}$ also improves, but eventually saturates with the growing width. Using trained weights in computation of pNTK results in improved test accuracy of $\hat{f}^{lin}$.

- Worse than infinite NTK for FCN/ConvNet (where they can be computed, if you try hard)
- Way worse than SGD

Recap

eNTK is a good tool for intuitive understanding of the learning process

Ren, Guo, S. Better Supervisory Signals by Observing Learning Paths

Ren, Guo, Bae, S. How to prepare your task head for finetuning

Ren, S. Learning dynamics of LLM Finetuning

Deng, Ren, M. Li, S., X. Li, Thrampoulidis On the Effect of Negative Gradient in Group Relative Deep Reinforcement Optimization

eNTK is practically very effective at “lookahead” for active learning

Mohamadi*, Bae*, S. Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels

You should probably use pNTK instead of eNTK for high-dim output problems:

Mohamadi, Bae, S. A Fast, Well-Founded Approximation to the Empirical Neural Tangent Kernel