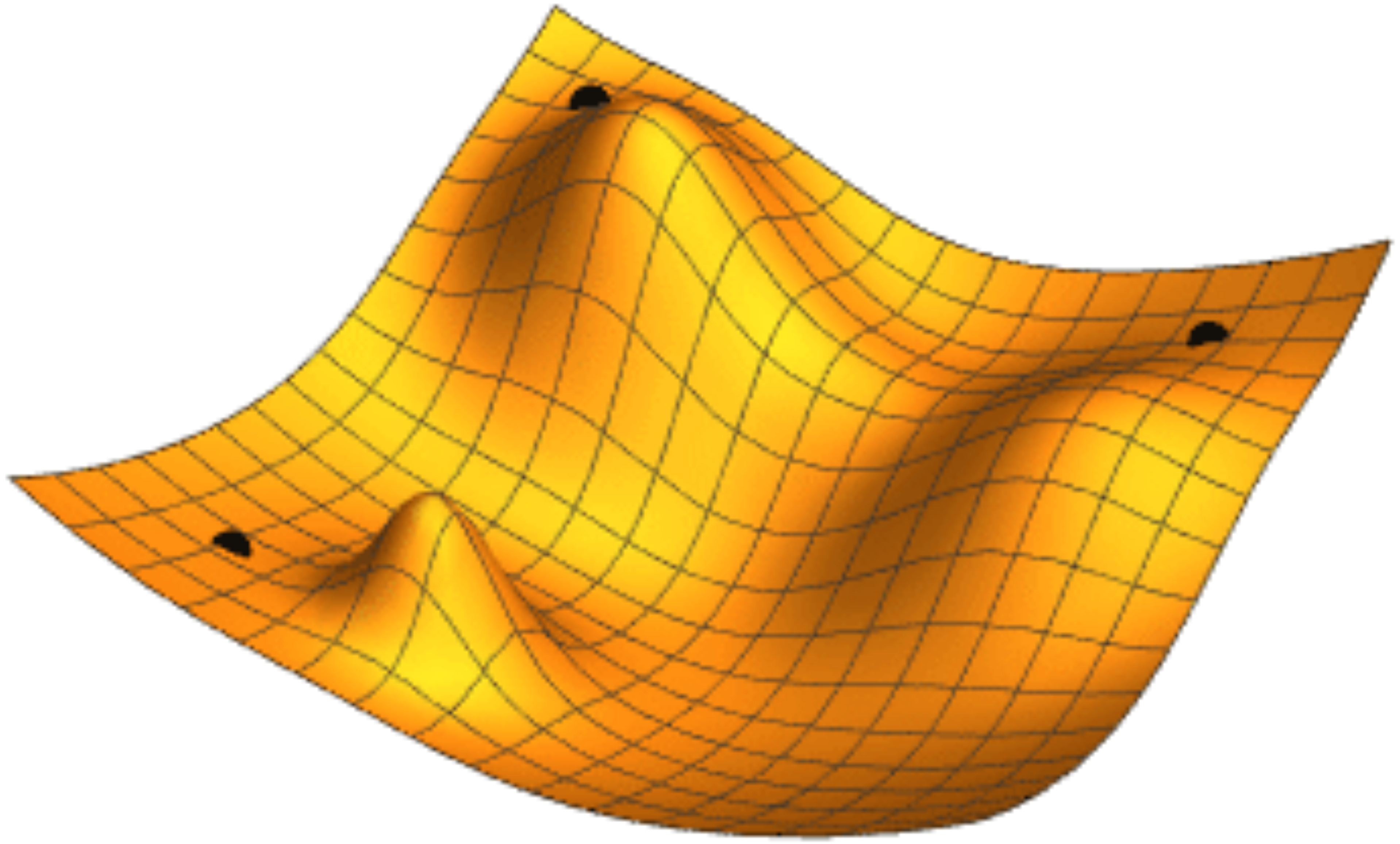
Neural Tangent Kernels, Finite and Infinite

ISI Winter School on Deep Learning
February 2023

Danica Sutherland
cs.ubc.ca/~dsuth/; these slides are under "talks" section

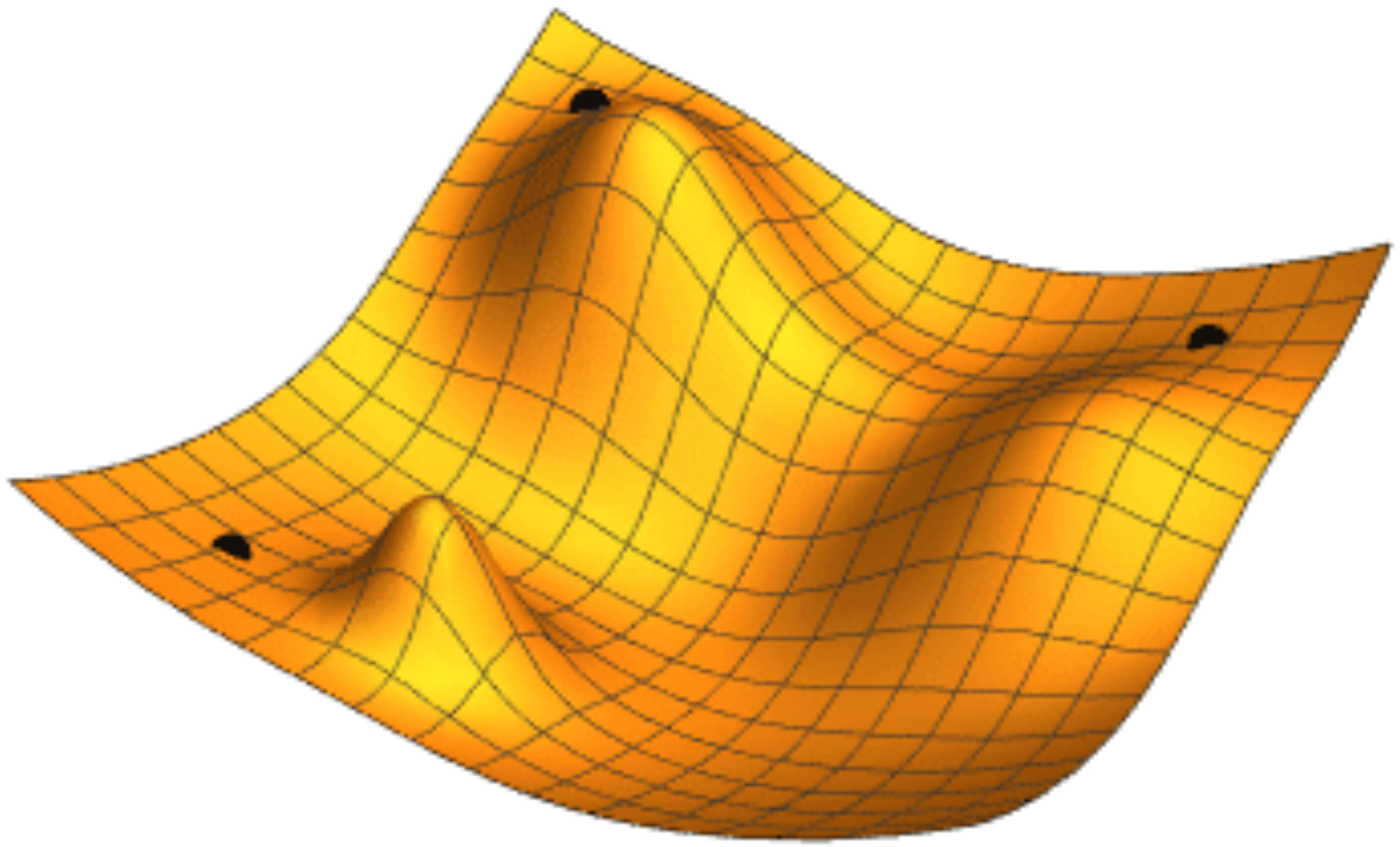
What happens when training a neural net?

- We use **gradient descent** (or similar) to try to find the best network



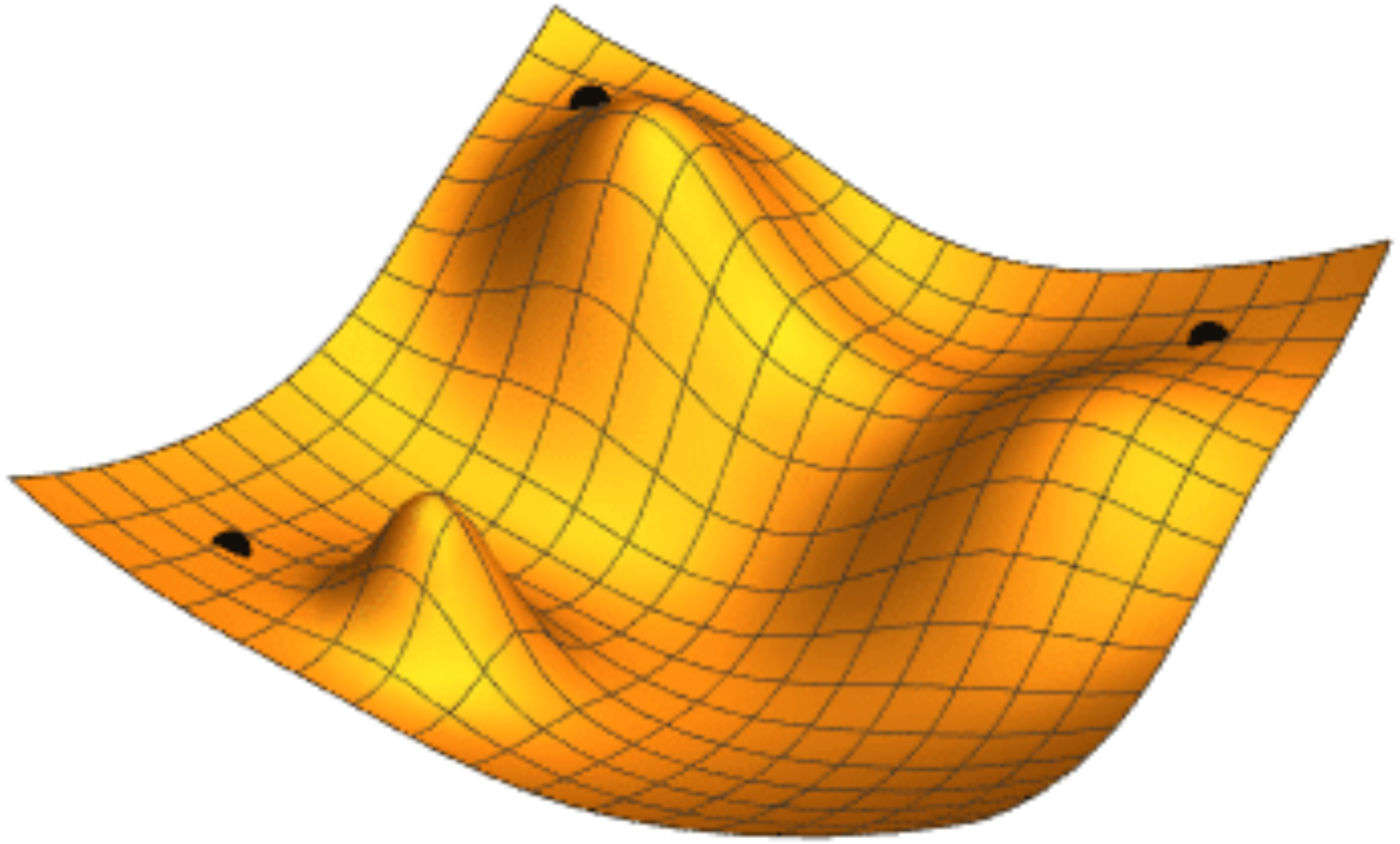
What happens when training a neural net?

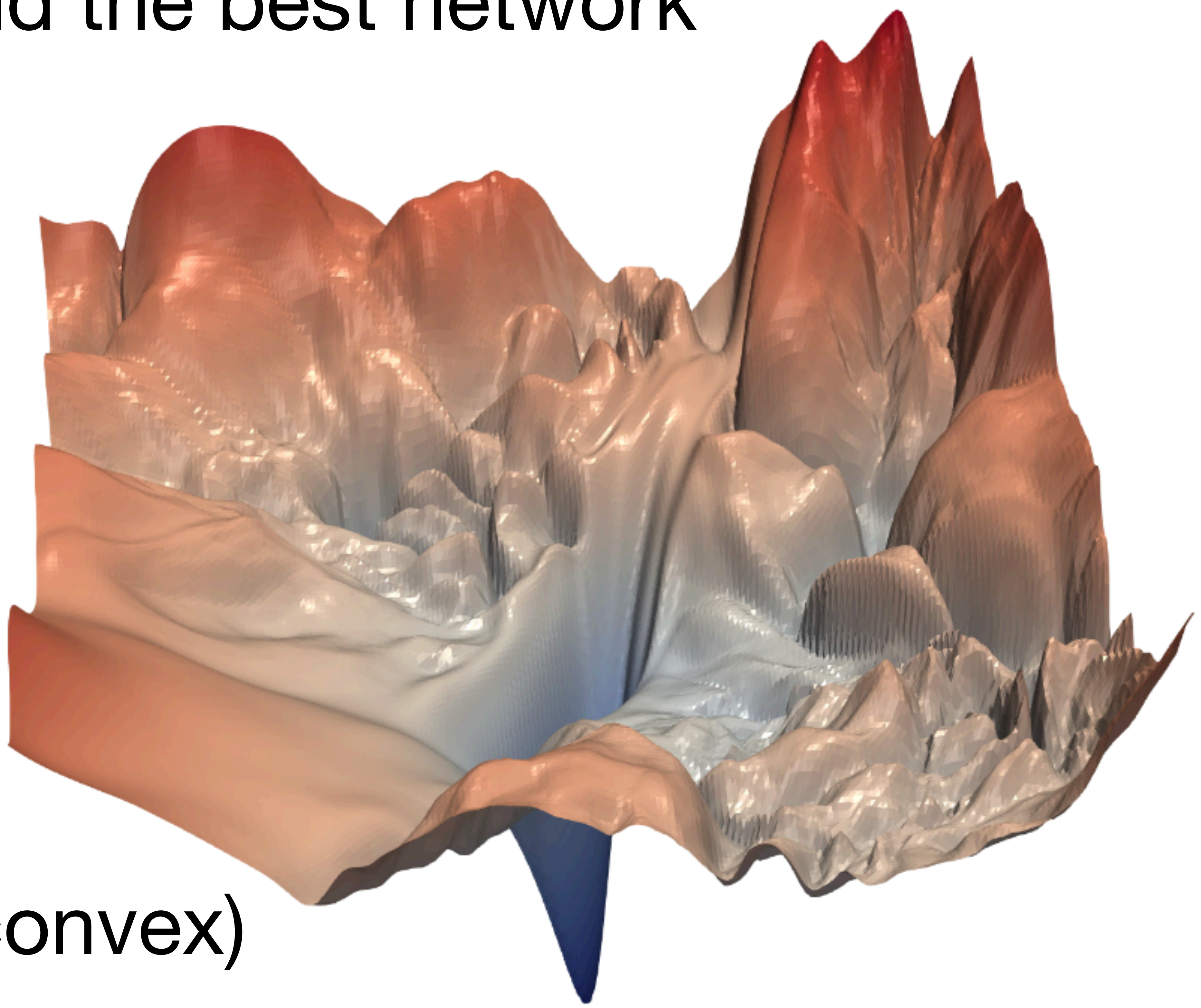
- We use **gradient descent** (or similar) to try to find the best network



What happens when training a neural net?

- We use **gradient descent** (or similar) to try to find the best network





- Loss landscape might be **complicated** (is non-convex)
- Where do we actually end up?
- **Neural tangent kernel** theory lets us approximate this process

Does training neural nets work?

Does training neural nets work?

- Gradient descent will find a **stationary point**: one where gradient = 0

Does training neural nets work?

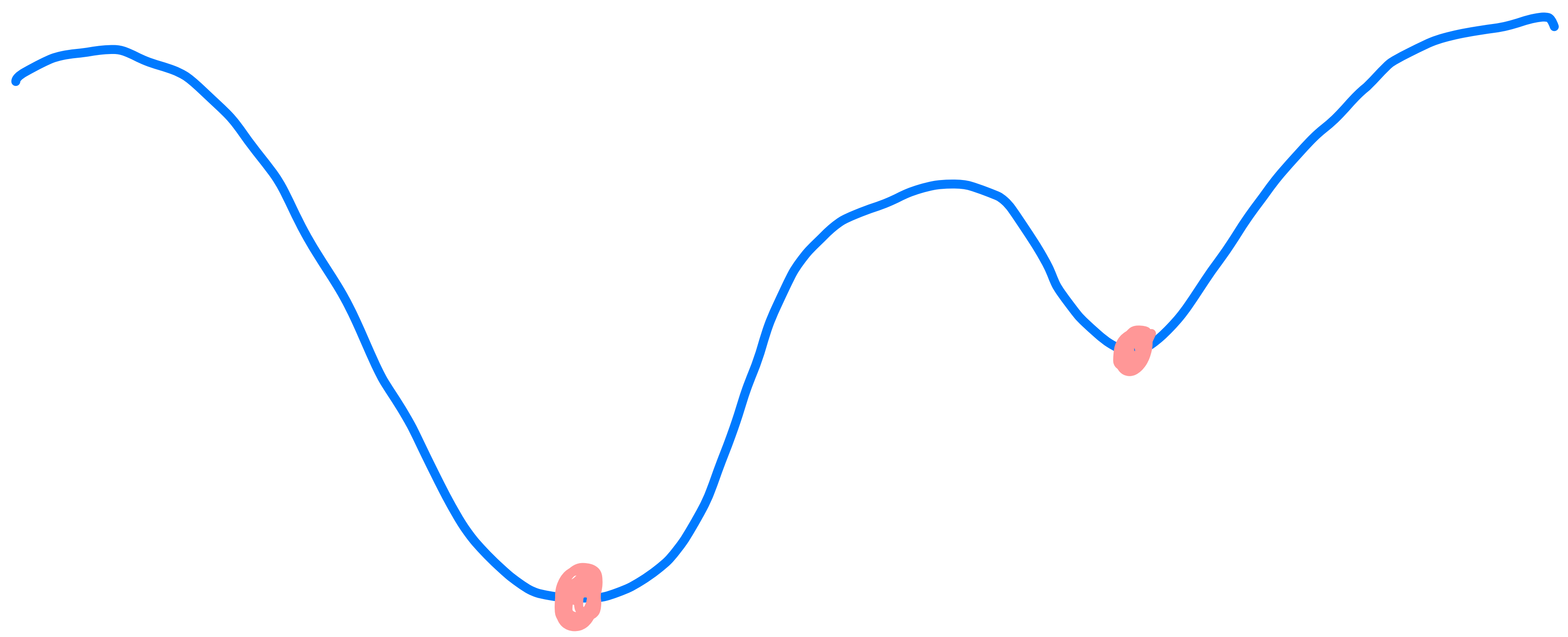
- Gradient descent will find a **stationary point**: one where gradient = 0
  - Could be a global minimum, a local minimum, or a saddle point

Does training neural nets work?

- Gradient descent will find a **stationary point**: one where gradient = 0
  - Could be a global minimum, a local minimum, or a saddle point
- Bad local minima do exist

Sub-Optimal Local Minima Exist for Neural Networks with Almost All Non-Linear Activations

Tian Ding* Dawei Li † Ruoyu Sun ‡



arXiv:1911.01413

Does training neural nets work?

- Gradient descent will find a **stationary point**: one where gradient = 0
  - Could be a global minimum, a local minimum, or a saddle point
- Bad local minima do exist
- But does SGD find them?

Sub-Optimal Local Minima Exist for Neural Networks with Almost All Non-Linear Activations

Tian Ding* Dawei Li † Ruoyu Sun ‡

arXiv:1911.01413

Does training neural nets work?

- Gradient descent will find a **stationary point**: one where gradient = 0
  - Could be a global minimum, a local minimum, or a saddle point
- Bad local minima do exist
- But does SGD find them?

Sub-Optimal Local Minima Exist for Neural Networks with Almost All Non-Linear Activations

Tian Ding* Dawei Li † Ruoyu Sun ‡

- Several papers around 2018-19 showed:
  - If the network is *very overparameterized* (width $\gg N$, possibly $\to \infty$)
  - and we use an appropriate random initialization
  - with square loss
  - then (S)GD finds a global minimum

Does training neural nets work?

- Gradient descent will find a **stationary point**: one where gradient = 0
  - Could be a global minimum, a local minimum, or a saddle point
- Bad local minima do exist
- But does SGD find them?

Sub-Optimal Local Minima Exist for Neural Networks with Almost All Non-Linear Activations

Tian Ding* Dawei Li † Ruoyu Sun ‡

- Several papers around 2018-19 showed:
  - If the network is *very overparameterized* (width $\gg N$, possibly $\to \infty$)
  - and we use an appropriate random initialization
  - with square loss
  - then (S)GD finds a global minimum
- Implicit in these papers:
  - Behaviour of deep nets converges to kernel ridge regression with the **neural tangent kernel**

arXiv:1911.01413

Problem setting

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x
  - $\mathbf{w}$ is all of the parameters of a deep net, all stacked together

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x
  - $\mathbf{w}$ is all of the parameters of a deep net, all stacked together
  - $\mathbf{x}$ is one particular input, e.g. 

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x
  - $\mathbf{w}$ is all of the parameters of a deep net, all stacked together
  - $\mathbf{x}$ is one particular input, e.g. 
  - $f(\mathbf{x}; \mathbf{w})$ is the output of the network, e.g. $[0.0002, \cdots, 0.8735, \cdots, 0.0001]$

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x
  - $\mathbf{w}$ is all of the parameters of a deep net, all stacked together
  - $\mathbf{x}$ is one particular input, e.g. 
  - $f(\mathbf{x}; \mathbf{w})$ is the output of the network, e.g. $[0.0002, \cdots, 0.8735, \cdots, 0.0001]$
- Have a labeled dataset $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x
  - $\mathbf{w}$ is all of the parameters of a deep net, all stacked together
  - $\mathbf{x}$ is one particular input, e.g. 
  - $f(\mathbf{x}; \mathbf{w})$ is the output of the network, e.g. $[0.0002, \cdots, 0.8735, \cdots, 0.0001]$
- Have a labeled dataset $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$

[0 0 1 0 0 0]

- Per-element *loss function* $\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\log(\hat{\mathbf{y}} \cdot \mathbf{y})$, $\ell(\hat{y}, y) = \frac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2$, etc

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x
  - $\mathbf{w}$ is all of the parameters of a deep net, all stacked together
  - $\mathbf{x}$ is one particular input, e.g. 
  - $f(\mathbf{x}; \mathbf{w})$ is the output of the network, e.g. $[0.0002, \cdots, 0.8735, \cdots, 0.0001]$
- Have a labeled dataset $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$
- Per-element *loss function* $\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\log(\hat{\mathbf{y}} \cdot \mathbf{y})$, $\ell(\hat{y}, y) = \frac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2$, etc
- Training loss $L_S(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i)$

ImageNet: n11939491_daisy.JPEG

Problem setting

- Notation for this talk: $f(x; \mathbf{w})$ is a function with parameters $\mathbf{w}$ evaluated at x
  - $\mathbf{w}$ is all of the parameters of a deep net, all stacked together
  - $\mathbf{x}$ is one particular input, e.g. 
  - $f(\mathbf{x}; \mathbf{w})$ is the output of the network, e.g. $[0.0002, \cdots, 0.8735, \cdots, 0.0001]$
- Have a labeled dataset $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$
- Per-element *loss function* $\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\log(\hat{\mathbf{y}} \cdot \mathbf{y})$, $\ell(\hat{y}, y) = \frac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2$, etc
- Training loss $L_S(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i)$
- Choose $\mathbf{w}$ to minimize L_S with (stochastic) gradient descent

ImageNet: n11939491_daisy.JPEG

One step of gradient descent

- **Full-batch** gradient descent, **square loss on scalars**: $L_S(\mathbf{w}) = \dfrac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2$

One step of gradient descent

- **Full-batch** gradient descent, **square loss on scalars**: $L_S(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}} \ell(f(\mathbf{x}_i; \mathbf{w}), y_i)\Big|_{\mathbf{w}_t}^{\top}$$

One step of gradient descent

- **Full-batch** gradient descent, **square loss on scalars**: $L_S(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}}\ell(f(\mathbf{x}_i; \mathbf{w}), y_i)\Big|_{\mathbf{w}_t}^{\top}$$

$$= \mathbf{w}_t - \frac{\eta}{N}\sum_{i=1}^{N}\left(\left[\nabla_{\hat{y}}\ell(\hat{y}, y_i)\Big|_{\hat{y}=f(\mathbf{x}_i,\mathbf{w}_t)}\right]\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]\right)^{\top}$$

One step of gradient descent

- **Full-batch** gradient descent, **square loss on scalars**: $L_S(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}} \ell(f(\mathbf{x}_i; \mathbf{w}), y_i)\Big|_{\mathbf{w}_t}^{\top}$$

$$= \mathbf{w}_t - \frac{\eta}{N}\sum_{i=1}^{N}\left(\left[\nabla_{\hat{y}} \ell(\hat{y}, y_i)\Big|_{\hat{y}=f(\mathbf{x}_i, \mathbf{w}_t)}\right]\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]\right)^{\top}$$

$$\mathbf{w}_{t+1} - \mathbf{w}_t = -\frac{\eta}{N}\sum_{i=1}^{N}\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]^{\top}\left(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\right)$$

One step of gradient descent

- **Full-batch** gradient descent, **square loss on scalars**: $L_S(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{N}\sum_{i=1}^{N} \nabla_{\mathbf{w}} \ell(f(\mathbf{x}_i; \mathbf{w}), y_i)\Big|_{\mathbf{w}_t}^{\top}$$

$$= \mathbf{w}_t - \frac{\eta}{N}\sum_{i=1}^{N}\left(\left[\nabla_{\hat{y}}\ell(\hat{y}, y_i)\Big|_{\hat{y}=f(\mathbf{x}_i,\mathbf{w}_t)}\right]\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]\right)^{\top}$$

$$\mathbf{w}_{t+1} - \mathbf{w}_t = -\frac{\eta}{N}\sum_{i=1}^{N}\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]^{\top}\left(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\right)$$

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N}\sum_{i=1}^{N}\left\langle \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})\Big|_{\mathbf{w}_t}, \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right\rangle\left(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\right)$$

One step of gradient descent *in function space*

- **Full-batch** gradient descent, **square loss**: $L_S(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i,\mathbf{w}) - y_i\right)^2$

$$\mathbf{w}_{t+1} - \mathbf{w}_t = -\frac{\eta}{N}\sum_{i=1}^{N}\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i,\mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]^{\top}\left(f(\mathbf{x}_i,\mathbf{w}_t) - y_i\right)$$

- What does that do to $f(\mathbf{x};\mathbf{w}_t)$?

One step of gradient descent *in function space*

- **Full-batch** gradient descent, **square loss**: $L_S(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2$

$$\mathbf{w}_{t+1} - \mathbf{w}_t = -\frac{\eta}{N}\sum_{i=1}^{N}\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]^{\top}\left(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\right)$$

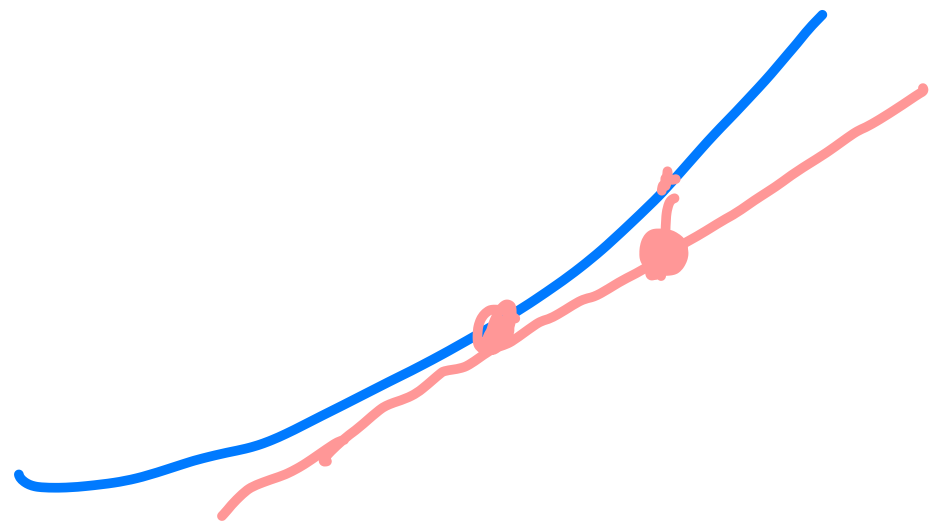
- What does that do to $f(\mathbf{x}; \mathbf{w}_t)$?

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N}\sum_{i=1}^{N}\left\langle \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})\Big|_{\mathbf{w}_t}, \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w})\Big|_{\mathbf{w}_t}\right\rangle\left(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\right)$$

One step of gradient descent *in function space*

- **Full-batch** gradient descent, **square loss**: $L_S(\mathbf{w}) = \dfrac{1}{2N}\sum_{i=1}^{N}\big(f(\mathbf{x}_i, \mathbf{w}) - y_i\big)^2$

$$\mathbf{w}_{t+1} - \mathbf{w}_t = -\frac{\eta}{N}\sum_{i=1}^{N}\left[\nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}_t)\Big|_{\mathbf{w}_t}\right]^{\top}\big(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\big)$$



- What does that do to $f(\mathbf{x}; \mathbf{w}_t)$?

$$f(\mathbf{x}; \mathbf{w}_{t+1}) = f(\mathbf{x}; \mathbf{w}_t) + \left[\nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})\Big|_{\mathbf{w}_t}\right](\mathbf{w}_{t+1} - \mathbf{w}_t) + \mathcal{O}(\|\mathbf{w}_{t+1} - \mathbf{w}_t\|^2)$$

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N}\sum_{i=1}^{N}\left\langle \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})\Big|_{\mathbf{w}_t}, \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w})\Big|_{\mathbf{w}_t}\right\rangle \big(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\big)$$

NTK regime

- Defining a function $k_{\mathbf{w}}(\mathbf{x}, \mathbf{x}') = \left\langle \nabla f(\mathbf{x}; \mathbf{w})|_{\mathbf{w}}, \nabla f(\mathbf{x}'; \mathbf{w})|_{\mathbf{w}} \right\rangle$, we just showed

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N}\sum_{i=1}^{N} k_{\mathbf{w}_t}(\mathbf{x}, \mathbf{x}_i)\big(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\big)$$

$f(x; w_0)$

$\vdots$

$f(x; w_T)$

NTK regime

- Defining a function $k_{\mathbf{w}}(\mathbf{x}, \mathbf{x}') = \left\langle \nabla f(\mathbf{x}; \mathbf{w})\big|_{\mathbf{w}}, \nabla f(\mathbf{x}'; \mathbf{w})\big|_{\mathbf{w}} \right\rangle$, we just showed

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \sum_{i=1}^{N} k_{\mathbf{w}_t}(\mathbf{x}, \mathbf{x}_i) \big(f(\mathbf{x}_i, \mathbf{w}_t) - y_i \big)$$

- "**NTK regime**" is when $k_{\mathbf{w}_t} \approx k_0$ throughout training. If so, we have

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \sum_{i=1}^{N} k_0(\mathbf{x}, \mathbf{x}_i) \big(f(\mathbf{x}_i, \mathbf{w}_t) - y_i \big)$$

NTK regime

- Defining a function $k_{\mathbf{w}}(\mathbf{x}, \mathbf{x}') = \left\langle \nabla f(\mathbf{x}; \mathbf{w})\big|_{\mathbf{w}}, \nabla f(\mathbf{x}'; \mathbf{w})\big|_{\mathbf{w}} \right\rangle$, we just showed

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \sum_{i=1}^{N} k_{\mathbf{w}_t}(\mathbf{x}, \mathbf{x}_i)\big(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\big)$$

- "**NTK regime**" is when $k_{\mathbf{w}_t} \approx k_0$ throughout training. If so, we have

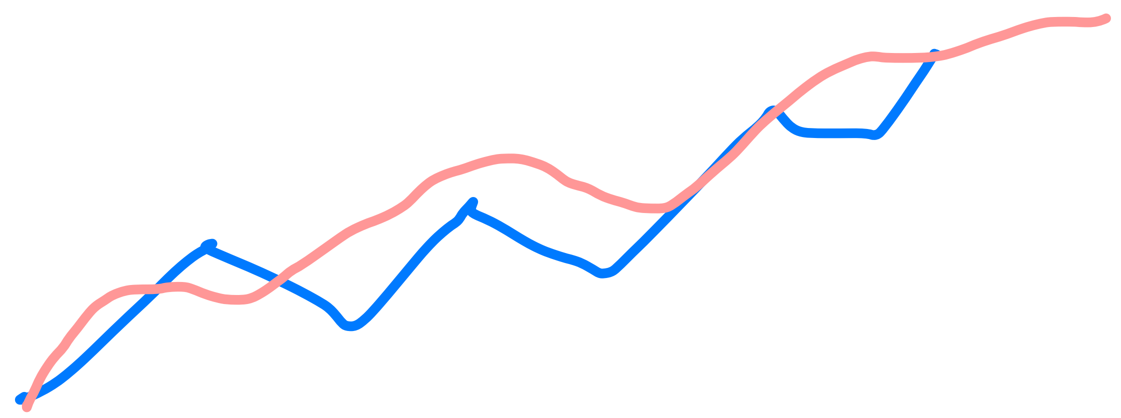
$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \sum_{i=1}^{N} k_0(\mathbf{x}, \mathbf{x}_i)\big(f(\mathbf{x}_i, \mathbf{w}_t) - y_i\big)$$

- Let $\mathbf{k}_0(\mathbf{x}) = \begin{bmatrix} k_0(\mathbf{x}, \mathbf{x}_1) & \cdots & k_0(\mathbf{x}, \mathbf{x}_N) \end{bmatrix} \in \mathbb{R}^{1 \times N}$,
  $\mathbf{f}_t = \begin{bmatrix} f(\mathbf{x}_1; \mathbf{w}_t) & \cdots & f(\mathbf{x}_N; \mathbf{w}_t) \end{bmatrix} \in \mathbb{R}^N$,
  $\mathbf{y} = \begin{bmatrix} y_1 & \cdots & y_N \end{bmatrix} \in \mathbb{R}^N$:

Kwt

$$f(\mathbf{x}; \mathbf{w}_{t+1}) - f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t - \mathbf{y})$$

Linearized solution



- If $k_{\mathbf{w}_t} \approx k_0$ throughout training,

  and we take a continuous limit instead of discrete steps (**gradient flow**),

$$\frac{\mathrm{d}}{\mathrm{d}t} f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t - \mathbf{y})$$

Linearized solution

- If $k_{\mathbf{w}_t} \approx k_0$ throughout training,
  and we take a continuous limit instead of discrete steps (**gradient flow**),

$$\frac{\mathrm{d}}{\mathrm{d}t} f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t - \mathbf{y})$$

- Let's define an explicit approximation: $f^{lin}(\mathbf{x}; \mathbf{w}) = f(\mathbf{x}; \mathbf{w}_0) + \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w}) \Big|_{\mathbf{w}_0}$

Linearized solution

- If $k_{\mathbf{w}_t} \approx k_0$ throughout training,
  and we take a continuous limit instead of discrete steps (**gradient flow**),
  $$\frac{\mathrm{d}}{\mathrm{d}t} f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t - \mathbf{y})$$
- Let's define an explicit approximation: $f^{lin}(\mathbf{x}; \mathbf{w}) = f(\mathbf{x}; \mathbf{w}_0) + \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w}) \Big|_{\mathbf{w}_0}$
- The differential equation $\frac{\mathrm{d}}{\mathrm{d}t} f^{lin}(\mathbf{x}; \mathbf{w}_t) = -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t^{lin} - \mathbf{y})$ has a closed-form solution:
  letting $(\mathbf{K}_0)_{ij} = k_0(\mathbf{x}_i, \mathbf{x}_j)$,
  $$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N} \mathbf{K}_0} \right) (\mathbf{y} - \mathbf{f}_0) + \overbrace{f_0(\mathbf{x})}^{f(x;w_0)}$$

Linearized solution

- If $k_{\mathbf{w}_t} \approx k_0$ throughout training,
  and we take a continuous limit instead of discrete steps (**gradient flow**),
$$\frac{\mathrm{d}}{\mathrm{d}t} f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t - \mathbf{y})$$

- Let's define an explicit approximation: $f^{lin}(\mathbf{x}; \mathbf{w}) = f(\mathbf{x}; \mathbf{w}_0) + \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})\Big|_{\mathbf{w}_0}$
- The differential equation $\frac{\mathrm{d}}{\mathrm{d}t} f^{lin}(\mathbf{x}; \mathbf{w}_t) = -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t^{lin} - \mathbf{y})$ has a closed-form solution:
  letting $(\mathbf{K}_0)_{ij} = k_0(\mathbf{x}_i, \mathbf{x}_j)$,
$$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) (\mathbf{y} - \mathbf{f}_0) + f_0(\mathbf{x})$$
  - As $t \to \infty$, if $\mathbf{K}_0$ is full-rank (usual case), $e^{-\frac{\eta t}{N}\mathbf{K}_0} \to \mathbf{0}$,
    $$f^{lin}(\mathbf{x}; \mathbf{w}_t) \to \underbrace{\mathbf{k}_0(\mathbf{x})}_{1\times N}\, \underbrace{\mathbf{K}_0^{-1}}_{N\times N} \underbrace{(\mathbf{y} - \mathbf{f}_0)}_{N\times 1} + \underbrace{f_0(\mathbf{x})}_{1\times 1}$$

Linearized solution

- If $k_{\mathbf{w}_t} \approx k_0$ throughout training,
  and we take a continuous limit instead of discrete steps (**gradient flow**),
  $$\frac{\mathrm{d}}{\mathrm{d}t} f(\mathbf{x}; \mathbf{w}_t) \approx -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t - \mathbf{y})$$
- Let's define an explicit approximation: $f^{lin}(\mathbf{x}; \mathbf{w}) = f(\mathbf{x}; \mathbf{w}_0) + \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w}) \Big|_{\mathbf{w}_0}$
- The differential equation $\frac{\mathrm{d}}{\mathrm{d}t} f^{lin}(\mathbf{x}; \mathbf{w}_t) = -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x})(\mathbf{f}_t^{lin} - \mathbf{y})$ has a closed-form solution:
  letting $(\mathbf{K}_0)_{ij} = k_0(\mathbf{x}_i, \mathbf{x}_j)$,
  $$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) (\mathbf{y} - \mathbf{f}_0) + f_0(\mathbf{x})$$
  - As $t \to \infty$, if $\mathbf{K}_0$ is full-rank (usual case), $e^{-\frac{\eta t}{N}\mathbf{K}_0} \to \mathbf{0}$,
    $f^{lin}(\mathbf{x}; \mathbf{w}_t) \to \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$
- This is the same formula as gradient flow for **kernel regression** – back to this soon!

A wide, shallow network

- Start with depth 2, scalar output: $$f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\,\sigma(\mathbf{w}_j\cdot\mathbf{x})$$

A wide, shallow network

- Start with depth 2, scalar output: $f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$
  - $\mathbf{w}_j \in \mathbb{R}^d$ is part of the vector of all parameters, $\mathbf{w} \in \mathbb{R}^D$ for $D = md$

A wide, shallow network

- Start with depth 2, scalar output: $f(\mathbf{x}; \mathbf{w}) = \dfrac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$
  - $\mathbf{w}_j \in \mathbb{R}^d$ is part of the vector of all parameters, $\mathbf{w} \in \mathbb{R}^D$ for $D = md$
  - The a_j are fixed signs in $\{-1, 1\}$, for maximum simplicity

A wide, shallow network

- Start with depth 2, scalar output: $f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$
  - $\mathbf{w}_j \in \mathbb{R}^d$ is part of the vector of all parameters, $\mathbf{w} \in \mathbb{R}^D$ for $D = md$
  - The a_j are fixed signs in $\{-1, 1\}$, for maximum simplicity
- Then $k_{\mathbf{w}}(\mathbf{x}_1, \mathbf{x}_2) = \left\langle \nabla_{\mathbf{w}} f(\mathbf{x}_1; \mathbf{w}), \nabla_{\mathbf{w}} f(\mathbf{x}_1; \mathbf{w}) \right\rangle$

A wide, shallow network

- Start with depth 2, scalar output: $f(\mathbf{x};\mathbf{w}) = \dfrac{1}{\sqrt{m}}\displaystyle\sum_{j=1}^{m} a_j\,\sigma(\mathbf{w}_j \cdot \mathbf{x})$
    - $\mathbf{w}_j \in \mathbb{R}^d$ is part of the vector of all parameters, $\mathbf{w} \in \mathbb{R}^D$ for $D = md$
    - The a_j are fixed signs in $\{-1, 1\}$, for maximum simplicity
- Then $k_{\mathbf{w}}(\mathbf{x}_1, \mathbf{x}_2) = \left\langle \nabla_{\mathbf{w}} f(\mathbf{x}_1;\mathbf{w}), \nabla_{\mathbf{w}} f(\underset{2}{\mathbf{x}_1};\mathbf{w}) \right\rangle$

$$= \left\langle \begin{bmatrix} a_1\mathbf{x}_1\sigma'(\mathbf{w}_1\cdot\mathbf{x}_1)/\sqrt{m} \\ \vdots \\ a_m\mathbf{x}_1\sigma'(\mathbf{w}_m\cdot\mathbf{x}_1)/\sqrt{m} \end{bmatrix} \begin{array}{l} \}d \\ \}d \\ \}d \end{array}, \begin{bmatrix} a_1\mathbf{x}_2\sigma'(\mathbf{w}_1\cdot\mathbf{x}_2)/\sqrt{m} \\ \vdots \\ a_m\mathbf{x}_2\sigma'(\mathbf{w}_m\cdot\mathbf{x}_2)/\sqrt{m} \end{bmatrix} \right\rangle$$

A wide, shallow network

- Start with depth 2, scalar output: $f(\mathbf{x};\mathbf{w}) = \dfrac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \,\sigma(\mathbf{w}_j \cdot \mathbf{x})$
  - $\mathbf{w}_j \in \mathbb{R}^d$ is part of the vector of all parameters, $\mathbf{w} \in \mathbb{R}^D$ for $D = md$
  - The a_j are fixed signs in $\{-1,1\}$, for maximum simplicity
- Then $k_{\mathbf{w}}(\mathbf{x}_1,\mathbf{x}_2) = \left\langle \nabla_{\mathbf{w}} f(\mathbf{x}_1;\mathbf{w}), \nabla_{\mathbf{w}} f(\mathbf{x}_{\cancel{1}2};\mathbf{w}) \right\rangle$

$$= \left\langle \begin{bmatrix} a_1 \mathbf{x}_1 \sigma'(\mathbf{w}_1 \cdot \mathbf{x}_1)/\sqrt{m} \\ \vdots \\ a_m \mathbf{x}_1 \sigma'(\mathbf{w}_m \cdot \mathbf{x}_1)/\sqrt{m} \end{bmatrix}, \begin{bmatrix} a_1 \mathbf{x}_2 \sigma'(\mathbf{w}_1 \cdot \mathbf{x}_2)/\sqrt{m} \\ \vdots \\ a_m \mathbf{x}_2 \sigma'(\mathbf{w}_m \cdot \mathbf{x}_2)/\sqrt{m} \end{bmatrix} \right\rangle$$

$$= \mathbf{x}_1^\top \mathbf{x}_2 \left[\frac{1}{m} \sum_{j=1}^{m} a_j^2 \sigma'(\mathbf{w}_j \cdot \mathbf{x}_1) \sigma'(\mathbf{w}_j \cdot \mathbf{x}_2) \right]$$

A wide, shallow network

- Start with depth 2, scalar output: $f(\mathbf{x};\mathbf{w}) = \dfrac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(\mathbf{w}_j \cdot \mathbf{x})$
  - $\mathbf{w}_j \in \mathbb{R}^d$ is part of the vector of all parameters, $\mathbf{w} \in \mathbb{R}^D$ for $D = md$
  - The a_j are fixed signs in $\{-1,1\}$, for maximum simplicity
- Then $k_{\mathbf{w}}(\mathbf{x}_1, \mathbf{x}_2) = \left\langle \nabla_{\mathbf{w}} f(\mathbf{x}_1;\mathbf{w}), \nabla_{\mathbf{w}} f(\mathbf{x}_1;\mathbf{w}) \right\rangle$

$$= \left\langle \begin{bmatrix} a_1 \mathbf{x}_1 \sigma'(\mathbf{w}_1 \cdot \mathbf{x}_1)/\sqrt{m} \\ \vdots \\ a_m \mathbf{x}_1 \sigma'(\mathbf{w}_m \cdot \mathbf{x}_1)/\sqrt{m} \end{bmatrix}, \begin{bmatrix} a_1 \mathbf{x}_2 \sigma'(\mathbf{w}_1 \cdot \mathbf{x}_2)/\sqrt{m} \\ \vdots \\ a_m \mathbf{x}_2 \sigma'(\mathbf{w}_m \cdot \mathbf{x}_2)/\sqrt{m} \end{bmatrix} \right\rangle$$

$$= \mathbf{x}_1^\top \mathbf{x}_2 \left[\frac{1}{m} \sum_{j=1}^{m} \underbrace{a_j^2}_{1} \sigma'(\mathbf{w}_j \cdot \mathbf{x}_1) \sigma'(\mathbf{w}_j \cdot \mathbf{x}_2) \right] \xrightarrow{m \to \infty} x_1^\top x_2 \, \mathbb{E}_{\mathbf{w}} \left[\sigma'(\mathbf{w}^\top x_1) \, \sigma'(\mathbf{w}^\top x_2) \right]$$

if the $\mathbf{w}_j$ are i.i.d. random

arccos kernel

For $\|\mathbf{x}_1\| = 1 = \|\mathbf{x}_2\|$, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \nu\mathbf{I})$ for any $\nu > 0$, and $\sigma(z) = \mathrm{ReLU}(z) = \max(z,0)$, the **NTK at initialization** is

$$(\mathbf{x}_1 \cdot \mathbf{x}_2)\mathbb{E}_{\mathbf{w}}[\sigma'(\mathbf{w} \cdot x_1)\, \sigma'(\mathbf{w} \cdot x_2)] = (\mathbf{x}_1 \cdot \mathbf{x}_2)\left(\frac{1}{2} - \frac{1}{2\pi}\arccos(\mathbf{x}_1 \cdot \mathbf{x}_2)\right)$$

$$\sigma'(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

$$\Pr_{w}(w \cdot x_1 \geq 0 \text{ and } w \cdot x_2 \geq 0)$$

arccos kernel

For $\|\mathbf{x}_1\| = 1 = \|\mathbf{x}_2\|$, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \nu \mathbf{I})$ for any $\nu > 0$, and $\sigma(z) = \mathrm{ReLU}(z) = \max(z, 0)$, the **NTK at initialization** is

$$(\mathbf{x}_1 \cdot \mathbf{x}_2)\mathbb{E}_{\mathbf{w}}[\sigma'(\mathbf{w} \cdot x_1)\,\sigma'(\mathbf{w} \cdot x_2)] = (\mathbf{x}_1 \cdot \mathbf{x}_2)\left(\frac{1}{2} - \frac{1}{2\pi}\arccos(\mathbf{x}_1 \cdot \mathbf{x}_2)\right)$$

This kernel has nice properties: it's **universal** on $\{x \in \mathbb{R}^{d+1} : \|x\| = 1,\, x_{d+1} = 1/\sqrt{2}\}$

NTK at initialization

- It’s generally true (with a much more complicated proof) that
  - for essentially any neural network architecture (CNNs, RNNs, GNNs, …)
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,

Tensor Programs I:
Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes

Greg Yang*
Microsoft Research AI
gregyang@microsoft.com

NTK at initialization

- It’s generally true (with a much more complicated proof) that
  - for essentially any neural network architecture (CNNs, RNNs, GNNs, …)
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - it holds that the “empirical NTK” $k_{\mathbf{w}}$ converges almost surely to $\mathbb{E}_{\mathbf{w}} k_{\mathbf{w}}$

Tensor Programs I:
Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes

---

Greg Yang*
Microsoft Research AI
gregyang@microsoft.com

NTK at initialization

- It's generally true (with a much more complicated proof) that
  - for essentially any neural network architecture (CNNs, RNNs, GNNs, …)
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - it holds that the "empirical NTK" $k_{\mathbf{w}}$ converges almost surely to $\mathbb{E}_{\mathbf{w}} k_{\mathbf{w}}$
    - Convergence might be slow, though!

Tensor Programs I:
Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes

Greg Yang*
Microsoft Research AI
gregyang@microsoft.com

NTK at initialization

- It's generally true (with a much more complicated proof) that
  - for essentially any neural network architecture (CNNs, RNNs, GNNs, …)
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - it holds that the "empirical NTK" $k_{\mathbf{w}}$ converges almost surely to $\mathbb{E}_{\mathbf{w}} k_{\mathbf{w}}$
    - Convergence might be slow, though!
- Can compute $\mathbb{E}_{\mathbf{w}} k_{\mathbf{w}}$ with dynamic programming: github.com/google/neural-tangents

Tensor Programs I:
Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes

Greg Yang*
Microsoft Research AI
gregyang@microsoft.com

How good is the approximation?

- Remember that we **linearized f around $\mathbf{w}_0$**:

$$f^{lin}(\mathbf{x};\mathbf{w}) = f(\mathbf{x};\mathbf{w}_0) + \left[\nabla_{\mathbf{w}} f(\mathbf{x},\mathbf{w})\Big|_{\mathbf{w}_0}\right](\mathbf{w}-\mathbf{w}_0) \approx f(\mathbf{x};\mathbf{w})$$

  - Linear in $\mathbf{w}$ but usually **not** linear in $\mathbf{x}$!
- Let's return to our simple $f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$ case
- How close is f^{lin} to f for the $\mathbf{w}$ we see during training?

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$f^{lin}(\mathbf{x}; \mathbf{w}) = f(\mathbf{x}; \mathbf{w}_0) + \langle \nabla f(\mathbf{x}; \mathbf{w}_0), \mathbf{w} - \mathbf{w}_0 \rangle$$

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$\begin{aligned} f^{lin}(\mathbf{x}; \mathbf{w}) &= f(\mathbf{x}; \mathbf{w}_0) + \langle \nabla f(\mathbf{x}; \mathbf{w}_0), \mathbf{w} - \mathbf{w}_0 \rangle \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{x} \cdot (\mathbf{w}_j - \mathbf{w}_{0,j}) \right] \end{aligned}$$

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$\begin{aligned}
f^{lin}(\mathbf{x}; \mathbf{w}) &= f(\mathbf{x}; \mathbf{w}_0) + \langle \nabla f(\mathbf{x}; \mathbf{w}_0), \mathbf{w} - \mathbf{w}_0 \rangle \\
&= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{x} \cdot (\mathbf{w}_j - \mathbf{w}_{0,j}) \right] \\
&= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{w}_{0,j} \cdot \mathbf{x} \right] + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{w}_j \cdot \mathbf{x} \right)
\end{aligned}$$

$$f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\,\sigma(\mathbf{w}_j\cdot\mathbf{x})$$

$$\begin{aligned}
f^{lin}(\mathbf{x};\mathbf{w}) &= f(\mathbf{x};\mathbf{w}_0) + \langle\,\nabla f(\mathbf{x};\mathbf{w}_0), \mathbf{w}-\mathbf{w}_0\rangle \\
&= \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\left[\sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) + \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\,\mathbf{x}\cdot(\mathbf{w}_j-\mathbf{w}_{0,j})\right] \\
&= \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\left(\underbrace{\left[\sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\,\mathbf{w}_{0,j}\cdot\mathbf{x}\right]}_{=\,0\text{ for ReLU: }\sigma(z)=z\sigma'(z)} + \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{w}_j\cdot\mathbf{x}\right)
\end{aligned}$$

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$\begin{aligned}
f^{lin}(\mathbf{x}; \mathbf{w}) &= f(\mathbf{x}; \mathbf{w}_0) + \langle \nabla f(\mathbf{x}; \mathbf{w}_0), \mathbf{w} - \mathbf{w}_0 \rangle \\
&= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{x} \cdot (\mathbf{w}_j - \mathbf{w}_{0,j}) \right] \\
&= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\underbrace{\left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{w}_{0,j} \cdot \mathbf{x} \right]}_{= 0 \text{ for ReLU: } \sigma(z) = z\sigma'(z)} + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{w}_j \cdot \mathbf{x} \right)
\end{aligned}$$

$$f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$$

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$\begin{aligned}
f^{lin}(\mathbf{x}; \mathbf{w}) &= f(\mathbf{x}; \mathbf{w}_0) + \langle \nabla f(\mathbf{x}; \mathbf{w}_0), \mathbf{w} - \mathbf{w}_0 \rangle \\
&= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{x} \cdot (\mathbf{w}_j - \mathbf{w}_{0,j}) \right] \\
&= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\underbrace{\left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{w}_{0,j} \cdot \mathbf{x} \right]}_{= 0 \text{ for ReLU: } \sigma(z) = z\sigma'(z)} + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{w}_j \cdot \mathbf{x} \right)
\end{aligned}$$

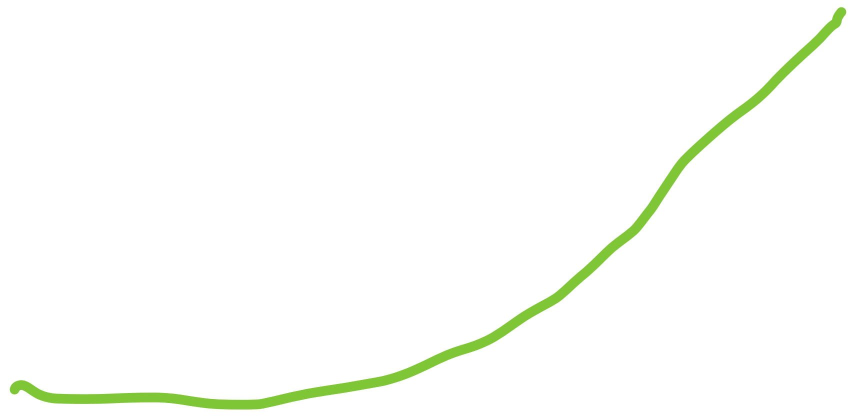
$$f_{W_0}(x; W) = \langle \nabla f(x; W_0), W \rangle$$

We'll see shortly that $f - f^{lin}$ *shrinks* as m grows

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$f^{lin}(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \, \mathbf{w}_{0,j} \cdot \mathbf{x} \right] + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{w}_j \cdot \mathbf{x} \right)$$

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(\mathbf{w}_j \cdot \mathbf{x})$$



$$f^{lin}(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \, \mathbf{w}_{0,j} \cdot \mathbf{x} \right] + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{w}_j \cdot \mathbf{x} \right)$$

If σ is **β-smooth** (meaning $|\sigma''(z)| \leq \beta$ for all z), $|a_j| \leq 1$, and $\|x\| \leq 1$:

$$f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\,\sigma(\mathbf{w}_j\cdot\mathbf{x})$$

$$f^{lin}(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\left(\left[\sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\,\mathbf{w}_{0,j}\cdot\mathbf{x}\right] + \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{w}_j\cdot\mathbf{x}\right)$$

If σ is β**-smooth** (meaning $|\sigma''(z)| \le \beta$ for all z), $|a_j| \le 1$, and $\|x\| \le 1$:

$$\left|f(\mathbf{x};\mathbf{w}) - f^{lin}(\mathbf{x};\mathbf{w})\right| \le \frac{1}{\sqrt{m}}\sum_{j=1}^{m}|a_j|\left|\sigma(\mathbf{w}_j\cdot\mathbf{x}) - \sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{x}\cdot(\mathbf{w}_j - \mathbf{w}_{0,j})\right|$$

$$f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \, \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$f^{lin}(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \, \mathbf{w}_{0,j} \cdot \mathbf{x} \right] + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{w}_j \cdot \mathbf{x} \right)$$

If σ is **β-smooth** (meaning $|\sigma''(z)| \leq \beta$ for all z), $|a_j| \leq 1$, and $\|x\| \leq 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r-s)| = \left| \int_r^s \sigma''(z)(s-z)\mathrm{d}z \right|$$

$$\left| f(\mathbf{x};\mathbf{w}) - f^{lin}(\mathbf{x};\mathbf{w}) \right| \leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \left| \sigma(\mathbf{w}_j \cdot \mathbf{x}) - \sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{x} \cdot (\mathbf{w}_j - \mathbf{w}_{0,j}) \right|$$

$$f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\,\sigma(\mathbf{w}_j\cdot\mathbf{x})$$

$$f^{lin}(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\left(\left[\sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\,\mathbf{w}_{0,j}\cdot\mathbf{x}\right] + \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{w}_j\cdot\mathbf{x}\right)$$

If σ is **β-smooth** (meaning $|\sigma''(z)| \leq \beta$ for all z), $|a_j| \leq 1$, and $\|x\| \leq 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r-s)| = \left|\int_r^s \sigma''(z)(s-z)\mathrm{d}z\right| \leq \frac{\beta}{2}(r-s)^2$$

$$\left|f(\mathbf{x};\mathbf{w}) - f^{lin}(\mathbf{x};\mathbf{w})\right| \leq \frac{1}{\sqrt{m}}\sum_{j=1}^{m}|a_j|\left|\sigma(\mathbf{w}_j\cdot\mathbf{x}) - \sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{x}\cdot(\mathbf{w}_j - \mathbf{w}_{0,j})\right|$$

$$f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\,\sigma(\mathbf{w}_j\cdot\mathbf{x})$$

$$f^{lin}(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\left(\left[\sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\,\mathbf{w}_{0,j}\cdot\mathbf{x}\right] + \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{w}_j\cdot\mathbf{x}\right)$$

If σ is β**-smooth** (meaning $|\sigma''(z)|\leq\beta$ for all z), $|a_j|\leq 1$, and $\|x\|\leq 1$:

$$|\sigma(r)-\sigma(s)-\sigma'(s)(r-s)| = \left|\int_r^s \sigma''(z)(s-z)\mathrm{d}z\right| \leq \frac{\beta}{2}(r-s)^2$$

$$\left|f(\mathbf{x};\mathbf{w}) - f^{lin}(\mathbf{x};\mathbf{w})\right| \leq \frac{1}{\sqrt{m}}\sum_{j=1}^{m}|a_j|\left|\sigma(\mathbf{w}_j\cdot\mathbf{x}) - \sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{x}\cdot(\mathbf{w}_j-\mathbf{w}_{0,j})\right|$$

$$\leq \frac{1}{\sqrt{m}}\sum_{j=1}^{m}\frac{1}{2}\beta(\mathbf{w}_j\cdot\mathbf{x}-\mathbf{w}_{0,j}\cdot\mathbf{x})^2$$

$$f(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\,\sigma(\mathbf{w}_j\cdot\mathbf{x})$$

$$f^{lin}(\mathbf{x};\mathbf{w}) = \frac{1}{\sqrt{m}}\sum_{j=1}^{m} a_j\left(\left[\sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\,\mathbf{w}_{0,j}\cdot\mathbf{x}\right] + \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{w}_j\cdot\mathbf{x}\right)$$

If σ is **β-smooth** (meaning $|\sigma''(z)|\le\beta$ for all z), $|a_j|\le 1$, and $\|x\|\le 1$:

$$|\sigma(r)-\sigma(s)-\sigma'(s)(r-s)| = \left|\int_r^s \sigma''(z)(s-z)\mathrm{d}z\right| \le \frac{\beta}{2}(r-s)^2$$

$$\left|f(\mathbf{x};\mathbf{w}) - f^{lin}(\mathbf{x};\mathbf{w})\right| \le \frac{1}{\sqrt{m}}\sum_{j=1}^{m}|a_j|\left|\sigma(\mathbf{w}_j\cdot\mathbf{x}) - \sigma(\mathbf{w}_{0,j}\cdot\mathbf{x}) - \sigma'(\mathbf{w}_{0,j}\cdot\mathbf{x})\mathbf{x}\cdot(\mathbf{w}_j-\mathbf{w}_{0,j})\right|$$

$$\le \frac{1}{\sqrt{m}}\sum_{j=1}^{m}\frac{1}{2}\beta(\mathbf{w}_j\cdot\mathbf{x}-\mathbf{w}_{0,j}\cdot\mathbf{x})^2 \le \frac{\beta}{2\sqrt{m}}\sum_{j=1}^{m}\|\mathbf{w}_j-\mathbf{w}_{0,j}\|^2\|\mathbf{x}\|^2$$

$$f(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(\mathbf{w}_j \cdot \mathbf{x})$$

$$f^{lin}(\mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x})\, \mathbf{w}_{0,j} \cdot \mathbf{x} \right] + \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{w}_j \cdot \mathbf{x} \right)$$

If σ is **β-smooth** (meaning $|\sigma''(z)| \leq \beta$ for all z), $|a_j| \leq 1$, and $\|x\| \leq 1$:

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r - s)| = \left| \int_r^s \sigma''(z)(s - z) \mathrm{d}z \right| \leq \frac{\beta}{2}(r - s)^2$$

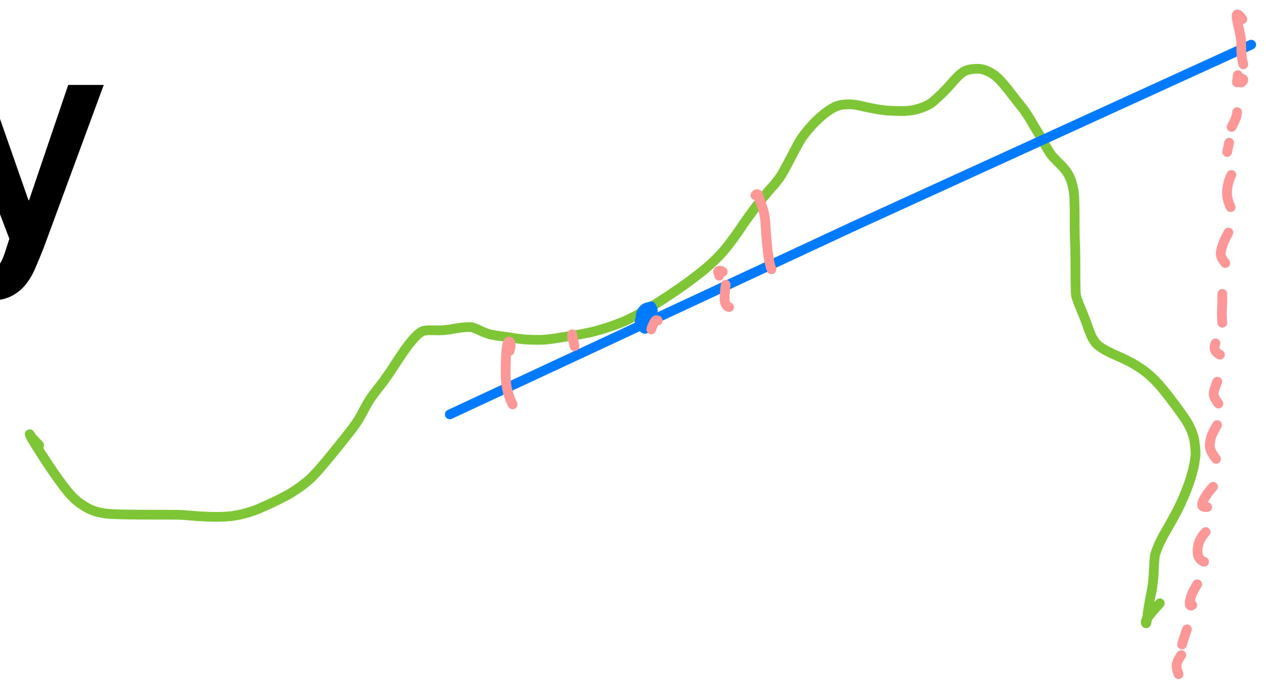
$$\left| f(\mathbf{x}; \mathbf{w}) - f^{lin}(\mathbf{x}; \mathbf{w}) \right| \leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \left| \sigma(\mathbf{w}_j \cdot \mathbf{x}) - \sigma(\mathbf{w}_{0,j} \cdot \mathbf{x}) - \sigma'(\mathbf{w}_{0,j} \cdot \mathbf{x}) \mathbf{x} \cdot (\mathbf{w}_j - \mathbf{w}_{0,j}) \right|$$

$$\leq \frac{1}{\sqrt{m}} \sum_{j=1}^{m} \frac{1}{2} \beta (\mathbf{w}_j \cdot \mathbf{x} - \mathbf{w}_{0,j} \cdot \mathbf{x})^2 \leq \frac{\beta}{2\sqrt{m}} \sum_{j=1}^{m} \|\mathbf{w}_j - \mathbf{w}_{0,j}\|^2 \|\mathbf{x}\|^2 \leq \frac{\beta}{2\sqrt{m}} \|\mathbf{w} - \mathbf{w}_0\|_F^2$$

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $||\mathbf{x}|| \leq 1$, $\quad |f(x; \mathbf{w}) - f^{lin}(x; \mathbf{w})| \leq \dfrac{\beta}{2\sqrt{m}} ||\mathbf{w} - \mathbf{w}_0||^2$

Linearization quality



- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $||\mathbf{x}|| \leq 1$, $\quad |f(x; \mathbf{w}) - f^{lin}(x; \mathbf{w})| \leq \dfrac{\beta}{2\sqrt{m}} ||\mathbf{w} - \mathbf{w}_0||^2$
  - This holds for *any* $\mathbf{w}$ and $\mathbf{w}_0$, but only for this shallow case

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $||\mathbf{x}|| \leq 1$, $\quad |f(x; \mathbf{w}) - f^{lin}(x; \mathbf{w})| \leq \dfrac{\beta}{2\sqrt{m}} ||\mathbf{w} - \mathbf{w}_0||^2$
  - This holds for *any* $\mathbf{w}$ and $\mathbf{w}_0$, but only for this shallow case
  - So if $||\mathbf{w}_t - \mathbf{w}_0||^2 \ll \dfrac{2}{\beta}\sqrt{m}$, approximation is "good enough"

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $||\mathbf{x}|| \leq 1,$ $\quad |f(x; \mathbf{w}) - f^{lin}(x; \mathbf{w})| \leq \dfrac{\beta}{2\sqrt{m}} ||\mathbf{w} - \mathbf{w}_0||^2$
  - This holds for *any* $\mathbf{w}$ and $\mathbf{w}_0$, but only for this shallow case
  - So if $||\mathbf{w}_t - \mathbf{w}_0||^2 \ll \dfrac{2}{\beta}\sqrt{m}$, approximation is "good enough"
  - $||\mathbf{w}_t - \mathbf{w}_0||^2$ is bounded as $m \to \infty$, **if** kernel is always full-rank

Linearization quality

- For a two-layer net with β-smooth hidden activations, second-layer weights $\leq 1/\sqrt{m}$ with linear activation, then for any $\|\mathbf{x}\| \leq 1, \quad |f(x;\mathbf{w}) - f^{lin}(x;\mathbf{w})| \leq \dfrac{\beta}{2\sqrt{m}}\|\mathbf{w} - \mathbf{w}_0\|^2$
  - This holds for *any* $\mathbf{w}$ and $\mathbf{w}_0$, but only for this shallow case
  - So if $\|\mathbf{w}_t - \mathbf{w}_0\|^2 \ll \dfrac{2}{\beta}\sqrt{m}$, approximation is "good enough"
  - $\|\mathbf{w}_t - \mathbf{w}_0\|^2$ is bounded as $m \to \infty$, **if** kernel is always full-rank
    - $\|\mathbf{w}_t - \mathbf{w}_0\| = \|-\Phi(\Phi\Phi^\top)^{-1}(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0})(\mathbf{f}_0 - \mathbf{y})\|$ where Φ stacks $\phi(\mathbf{x}_i)$

$$\leq \|\Phi(\Phi\Phi^\top)^{-1}\|_{op}\, \|(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0})\|_{op}\, \|\mathbf{f}_0 - \mathbf{y}\|$$

$$\leq \frac{1}{\sigma_{\min}(\Phi)} \cdot 1 \cdot \|\mathbf{f}_0 - \mathbf{y}\| \text{ for large } t$$

Full training with any architecture

- It's generally true (with a much more complicated proof) that
  - for essentially any neural network architecture (CNNs, RNNs, GNNs, …)
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - for small SGD step sizes η,
  - then for any $\mathbf{x}$ and t, it holds that $f_t(\mathbf{x}; w_t)$ converges almost surely to $f_t^{lin}(\mathbf{x}; w_t)$

Tensor Programs IIb:
Architectural Universality of Neural Tangent Kernel Training Dynamics

---

Greg Yang [1*] **Etai Littwin** [2*]

Recap of overall theory

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:

$$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$

Recap of overall theory

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:

$$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N} \mathbf{K}_0} \right) (\mathbf{y} - \mathbf{f}_0) + f_0(\mathbf{x})$$

  - and so as $t \to \infty$, (S)GD on the network converges to kernel regression

$$\hat{f}(\mathbf{x}) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} (\mathbf{y} - \mathbf{f}_0) + f_0(\mathbf{x})$$

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix} = \underbrace{K_0 K_0^{-1}}_{I} (y - f_0) + f_0 = y - f_0 + f_0 = y$$

$$K_0 = \begin{bmatrix} K_0(x_1, x_1) & \cdots & K_0(x_1, x_N) \\ \vdots & \ddots & \vdots \\ K_0(x_N, x_1) & \cdots & K_0(x_N, x_N) \end{bmatrix}$$

Recap of overall theory

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:
    $$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$
  - and so as $t \to \infty$, (S)GD on the network converges to kernel regression
    $$\hat{f}(\mathbf{x}) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$
    - predictions on training set: $\mathbf{K}_0 \mathbf{K}_0^{-1}(\mathbf{y} - \mathbf{f}_0) + \mathbf{f}_0 = \mathbf{y} - \mathbf{f}_0 + \mathbf{f}_0 = \mathbf{y}$

Vector outputs

- Network can have $O > 1$ outputs
  - $\phi_{\mathbf{w}}(\mathbf{x}) = \nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})|_{\mathbf{w}} \in \mathbb{R}^{O \times D}, \quad k_{\mathbf{w}}(\mathbf{x}_1, \mathbf{x}_2) = \phi_{\mathbf{w}}(\mathbf{x}_1)\phi_{\mathbf{w}}(\mathbf{x}_2)^\top \in \mathbb{R}^{O \times O}$
  - Usually write $\mathbf{K}_0 \in \mathbb{R}^{NO \times NO}$, $\mathbf{k}_0(\mathbf{x}) \in \mathbb{R}^{1 \times NO}$, $\mathbf{y} \in \mathbb{R}^{NO}$

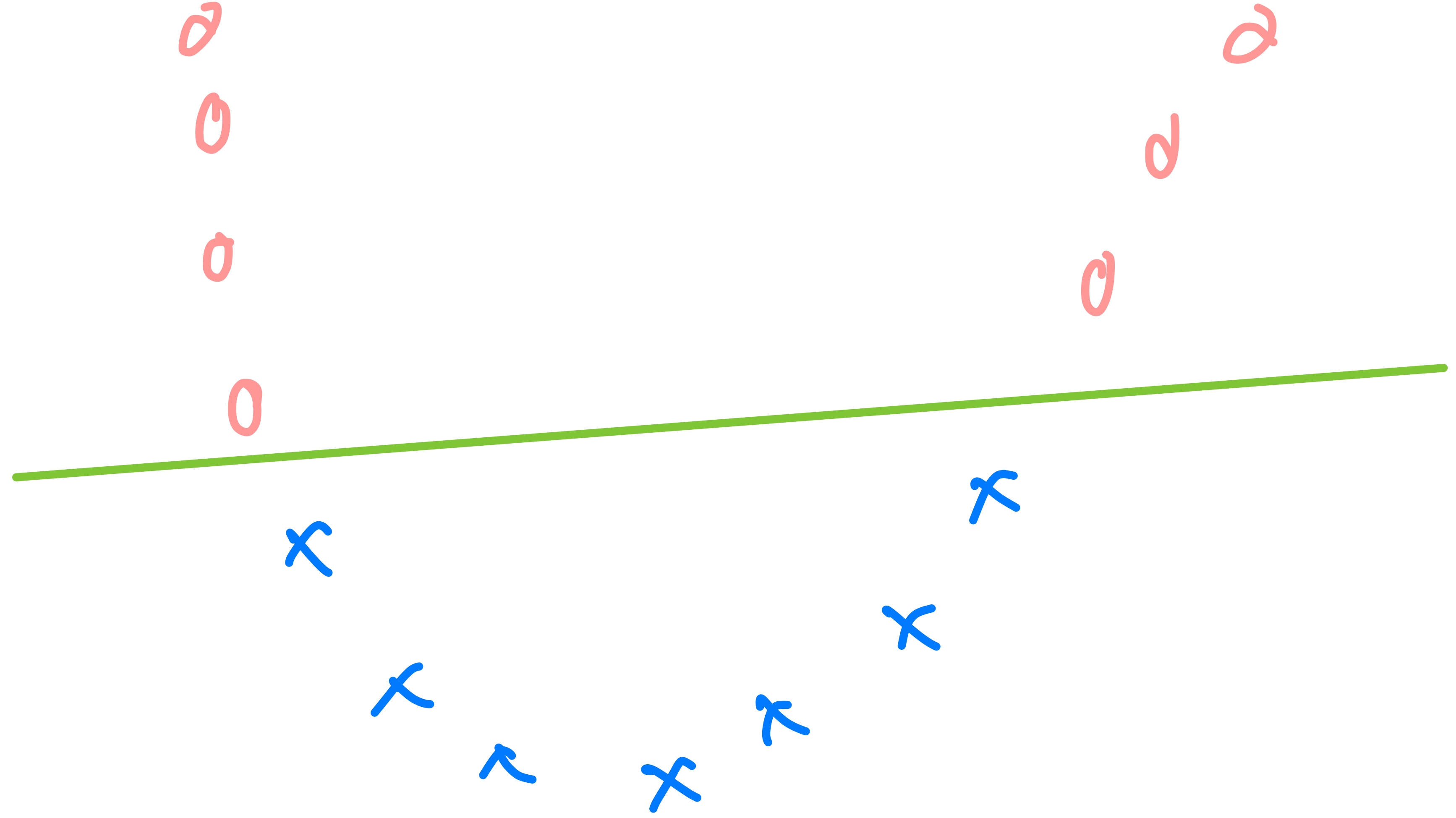
    Then $\quad f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x}) \in \mathbb{R}^O$
  -

Other loss functions

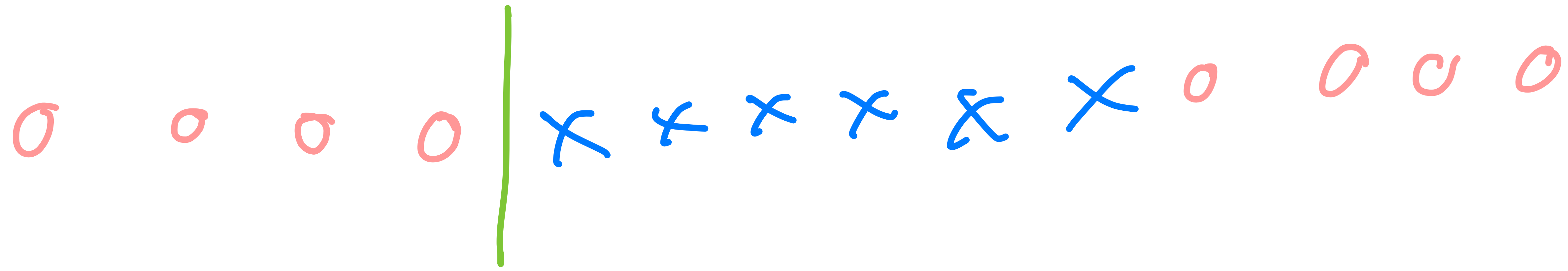
- Can use other losses than square loss; get same kind of ODE

$$\frac{\mathrm{d}}{\mathrm{d}t} f^{lin}(\mathbf{x}; \mathbf{w}_t) = -\frac{\eta}{N} \mathbf{k}_0(\mathbf{x}) \left[\nabla_{\hat{\mathbf{y}}} L_S(\hat{\mathbf{y}}, \mathbf{y}_i) \Big|_{\hat{\mathbf{y}}=f(\mathbf{x}_i, \mathbf{w}_t)} \right]_{i=1}^{N}$$

  - Doesn't necessarily have a closed form anymore
  - Square loss isn't such a bad loss, even for classification!
    - Hui and Belkin (2020)



Kernel regression



Kernel methods

- Kernel models are **linear models in feature space**
- A usual real-valued linear model is $f(\mathbf{x}; \mathbf{w}) = \langle \mathbf{x}, \mathbf{w} \rangle$ $= x \cdot w = x^T w$
- Kernel models are $f(\mathbf{x}; \mathbf{w}) = \langle \phi(\mathbf{x}), \mathbf{w} \rangle$ for some fixed **feature map** $\phi(\mathbf{x})$
  - For example, polynomial features like $\phi(\mathbf{x}) = \left(1, x, x^2\right)$
  - Can in general map to any Hilbert space, not just $\mathbb{R}^D$
    - More on this later!
- The **kernel function** is $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$
  - *Not the same as* kernel density estimation, convolutional kernels, kernel of a linear map (null space), the Linux kernel, …
- The space of functions $f(\mathbf{x}; \mathbf{w}) = \langle \phi(\mathbf{x}), \mathbf{w} \rangle$ for all possible $\mathbf{w}$ is known as a **reproducing kernel Hilbert space (RKHS)**

Kernel regression

OLS: $\varphi(x) = x$
NTK $\varphi_{w_t}(x) = \nabla_w f(x; w)|_{w_t}$

- For a fixed ϕ, minimize $L_S(\mathbf{w}) = \dfrac{1}{2N}\sum_{i=1}^{N}(\langle\phi(\mathbf{x}_i), \mathbf{w}\rangle - \mathbf{y}_i)^2$

$n \times D$

$\Phi = \begin{bmatrix} \varphi(x_1) \\ \vdots \\ \varphi(x_n) \end{bmatrix}$

$K = \Phi \Phi^T$

- If $D > N$, there are typically infinitely many $\mathbf{w}$ with $L_S(\mathbf{w}) = 0$
- One strategy: start at some $\mathbf{w}_0$ and do gradient descent until you hit one
  - If $\eta < \dfrac{2}{\sigma_{\max}(X)^2}$, gradient descent converges to (proof via SVD)

$$\hat{\mathbf{w}} = \underset{\mathbf{w}:\boldsymbol{\Phi}\mathbf{w}=\mathbf{y}}{\operatorname{argmin}}\|\mathbf{w} - \mathbf{w}_0\| = \underbrace{\boldsymbol{\Phi}^\top\mathbf{K}^{-1}\mathbf{y}}_{\Phi^\dagger y} + (\mathbf{I} - \boldsymbol{\Phi}^\top\mathbf{K}^{-1}\boldsymbol{\Phi})\mathbf{w}_0$$

$\varphi(x)^T \Phi = \begin{bmatrix} \varphi(x)^T\varphi(x_1) \\ \vdots \\ \varphi(x)^T\varphi(x_n) \end{bmatrix} = K(x)$

- Predictions are $\langle\varphi(\mathbf{x}), \hat{\mathbf{w}}\rangle = \mathbf{k}(\mathbf{x})\,\mathbf{K}^{-1}\mathbf{y} + f_0(\mathbf{x}) - \overset{K(x)}{\cancel{\mathbf{K}}}\mathbf{K}^{-1}\mathbf{f}_0$
- i.e. exactly $\hat{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x})\,\mathbf{K}^{-1}\left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$ from before

Kernel ridge regression

- The more common way to choose a solution is **ridge regression**: for $\lambda > 0$,

$$
\begin{aligned}
\hat{\mathbf{w}}_\lambda &= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} (\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle - y_i)^2 + \lambda \|\mathbf{w} - \mathbf{w}_0\|^2 \\
&= \mathbf{w}_0 + \boldsymbol{\Phi}^\top (\mathbf{K} + N\lambda \mathbf{I})^{-1} (\mathbf{y} - \boldsymbol{\Phi}^\top \mathbf{w}_0) \\
\langle \hat{\mathbf{w}}_\lambda, \phi(x) \rangle &= f_0(x) + \mathbf{k}(x) (\mathbf{K} + N\lambda \mathbf{I})^{-1} (\mathbf{y} - \mathbf{f}_0)
\end{aligned}
$$

Kernel ridge regression

- The more common way to choose a solution is **ridge regression**: for $\lambda > 0$,

$$\hat{\mathbf{w}}_\lambda = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{N}(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - y_i)^2 + \lambda\|\mathbf{w} - \mathbf{w}_0\|^2$$

$$= \mathbf{w}_0 + \mathbf{\Phi}^\top(\mathbf{K} + N\lambda\mathbf{I})^{-1}(\mathbf{y} - \mathbf{\Phi}^\top\mathbf{w}_0)$$

$$\langle \hat{\mathbf{w}}_\lambda, \phi(x)\rangle = f_0(x) + \mathbf{k}(x)\,(\mathbf{K} + N\lambda\mathbf{I})^{-1}(\mathbf{y} - \mathbf{f}_0)$$

- An equivalent view, **kernel ridge regression**:
  - The RKHS $\mathscr{H}$ is a Hilbert space, and so has a norm $\|f\|_{\mathscr{H}}$

$$\hat{f}_\lambda = \operatorname*{argmin}_{f \in \mathscr{H}} \frac{1}{N}\sum_{i=1}^{N}(f(\mathbf{x}_i) - y_i)^2 + \lambda\|f - f_0\|^2_{\mathscr{H}}$$

  - RKHS norm for $f(x) = \langle \mathbf{w}, \phi(x)\rangle$ is $\|f\|_{\mathscr{H}} = \|\mathbf{w}\|$

Kernel ridge regression

- The more common way to choose a solution is **ridge regression**: for $\lambda > 0$,

$$\hat{\mathbf{w}}_\lambda = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{N} (\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - y_i)^2 + \lambda \|\mathbf{w} - \mathbf{w}_0\|^2$$

$$= \mathbf{w}_0 + \mathbf{\Phi}^\top (\mathbf{K} + N\lambda \mathbf{I})^{-1}(\mathbf{y} - \mathbf{\Phi}^\top \mathbf{w}_0)$$

$$\langle \hat{\mathbf{w}}_\lambda, \phi(x)\rangle = f_0(x) \, + \, \mathbf{k}(x)\,(\mathbf{K} + N\lambda\mathbf{I})^{-1}(\mathbf{y} - \mathbf{f}_0)$$

- An equivalent view, **kernel ridge regression**:
  - The RKHS $\mathscr{H}$ is a Hilbert space, and so has a norm $\|f\|_{\mathscr{H}}$

$$\hat{f}_\lambda = \operatorname*{argmin}_{f \in \mathscr{H}} \frac{1}{N}\sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 + \lambda \|f - f_0\|_{\mathscr{H}}^2$$

  - RKHS norm for $f(x) = \langle \mathbf{w}, \phi(x)\rangle$ is $\|f\|_{\mathscr{H}} = \|\mathbf{w}\|$
- Note: equivalent to use $\mathbf{w}_0 = \mathbf{0}$ / $f_0(\mathbf{x}) = 0$ but fit **residuals** $y_i - f_0(\mathbf{x}_i)$

Kernel “ridgeless” regression

- Running small-LR gradient descent from $\mathbf{w}_0$ gives same predictions as $\lim_{\lambda \to 0} \hat{f}_\lambda$:

$$\hat{f}_\lambda(x) = \mathbf{k}(x)\,(\mathbf{K} + N\lambda\mathbf{I})^{-1}(\mathbf{y} - \mathbf{f}_0) + f_0(x)$$
$$\hat{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x})\,\mathbf{K}^{-1}\left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$

Kernel "ridgeless" regression

- Running small-LR gradient descent from $\mathbf{w}_0$ gives same predictions as $\lim_{\lambda \to 0} \hat{f}_\lambda$:

$$\hat{f}_\lambda(x) = \mathbf{k}(x)\,(\mathbf{K} + N\lambda\mathbf{I})^{-1}(\mathbf{y} - \mathbf{f}_0) + f_0(x)$$
$$\hat{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x})\,\mathbf{K}^{-1}\left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$

- We know some stuff about kernel predictors, e.g. what kinds of functions they can learn without overfitting
  - (it's the functions with small $\|f - f_0\|_{\mathscr{H}}$)

Infinite NTKs are infinite-dimensional

- The $\mathbf{w}$ for an empirical NTK is in $\mathbb{R}^D$, with D the total number of parameters

Infinite NTKs are infinite-dimensional

- The $\mathbf{w}$ for an empirical NTK is in $\mathbb{R}^D$, with D the total number of parameters
- As width $m \to \infty$, $D \to \infty$ as well

Infinite NTKs are infinite-dimensional

- The $\mathbf{w}$ for an empirical NTK is in $\mathbb{R}^D$, with D the total number of parameters
- As width $m \to \infty$, $D \to \infty$ as well
  - But $k_{\mathbf{w}}$ doesn't change too much as $m \to \infty$!

Infinite NTKs are infinite-dimensional

- The $\mathbf{w}$ for an empirical NTK is in $\mathbb{R}^D$, with D the total number of parameters
- As width $m \to \infty$, $D \to \infty$ as well
  - But $k_{\mathbf{w}}$ doesn't change too much as $m \to \infty$!
- We can still do stuff in infinite-dimensional RKHSes!

Infinite NTKs are infinite-dimensional

- The $\mathbf{w}$ for an empirical NTK is in $\mathbb{R}^D$, with D the total number of parameters
- As width $m \to \infty$, $D \to \infty$ as well
  - But $k_{\mathbf{w}}$ doesn't change too much as $m \to \infty$!
- We can still do stuff in infinite-dimensional RKHSes!
  - Gaussian kernel $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$ is also infinite-dim

Infinite NTKs are infinite-dimensional

- The $\mathbf{w}$ for an empirical NTK is in $\mathbb{R}^D$, with D the total number of parameters
- As width $m \to \infty$, $D \to \infty$ as well
  - But $k_{\mathbf{w}}$ doesn't change too much as $m \to \infty$!

- We can still do stuff in infinite-dimensional RKHSes!
  - Gaussian kernel $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$ is also infinite-dim
  - We just only use kernel form: $\hat{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x})\,\mathbf{K}^{-1}\left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$

Infinite NTKs are infinite-dimensional

- The $\mathbf{w}$ for an empirical NTK is in $\mathbb{R}^D$, with D the total number of parameters
- As width $m \to \infty$, $D \to \infty$ as well
  - But $k_{\mathbf{w}}$ doesn't change too much as $m \to \infty$!

- We can still do stuff in infinite-dimensional RKHSes!
  - Gaussian kernel $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$ is also infinite-dim
  - We just only use kernel form: $\hat{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x})\,\mathbf{K}^{-1}\left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$
  - **Representer theorem** implies that

$$\hat{f}(\mathbf{x}) - f_0(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = \mathbf{k}(\mathbf{x}) \cdot \boldsymbol{\alpha} \quad \text{for some } \boldsymbol{\alpha} \in \mathbb{R}^N$$

Uses and limitations of infinite NTKs

Great method for small-data tasks

Classifier	Friedman Rank	Average Accuracy	P90	P95	PMA
NTK	**28.34**	**81.95%±14.10%**	**88.89%**	**72.22%**	**95.72% ±5.17%**
NN (He init)	40.97	80.88%±14.96%	81.11%	65.56%	94.34% ±7.22%
NN (NTK init)	38.06	81.02%±14.47%	85.56%	60.00%	94.55% ±5.89%
RF	33.51	81.56% ±13.90%	85.56%	67.78%	95.25% ±5.30%
Gaussian Kernel	35.76	81.03% ± 15.09%	85.56%	**72.22%**	94.56% ±8.22%
Polynomial Kernel	38.44	78.21% ± 20.30%	80.00%	62.22%	91.29% ±18.05%

Table 1: Comparisons of different classifiers on 90 UCI datasets. P90/P95: the number of datasets a classifier achieves 90%/95% or more of the maximum accuracy, divided by the total number of datasets. PMA: average percentage of the maximum accuracy.

Good for distinguishing distributions

CIFAR	ME	SCF	C2ST-S	C2ST-L	M-O	M-D	SRF	**SCNTK**
2000	0.588	0.171	0.452	0.529	0.316	0.744	0.440	**0.805**

Table 5. SCNTK for outlier detection. SCNTK, CNTK with all relu activations (CNTK-relu), and naive Gaussian KDE are compared for the outlier detection task with CIFAR10 and SVHN datasets. With a fixed kernel, SCNTK shows a promising results for OOD detection in both settings.

Inlier	Outlier	Gaussian	CNTK-relu	SCNTK
CIFAR10	SVHN	0.82	0.71	0.85
SVHN	CIFAR10	0.20	0.51	0.80

Figure 1: The images generated by different methods on MNIST, CIFAR-10, and CelebA datasets given only 256 training images.

https://proceedings.mlr.press/v139/jia21a.html; https://openreview.net/pdf?id=_d2f3hRn0hT

Useful signal for trainability of architectures

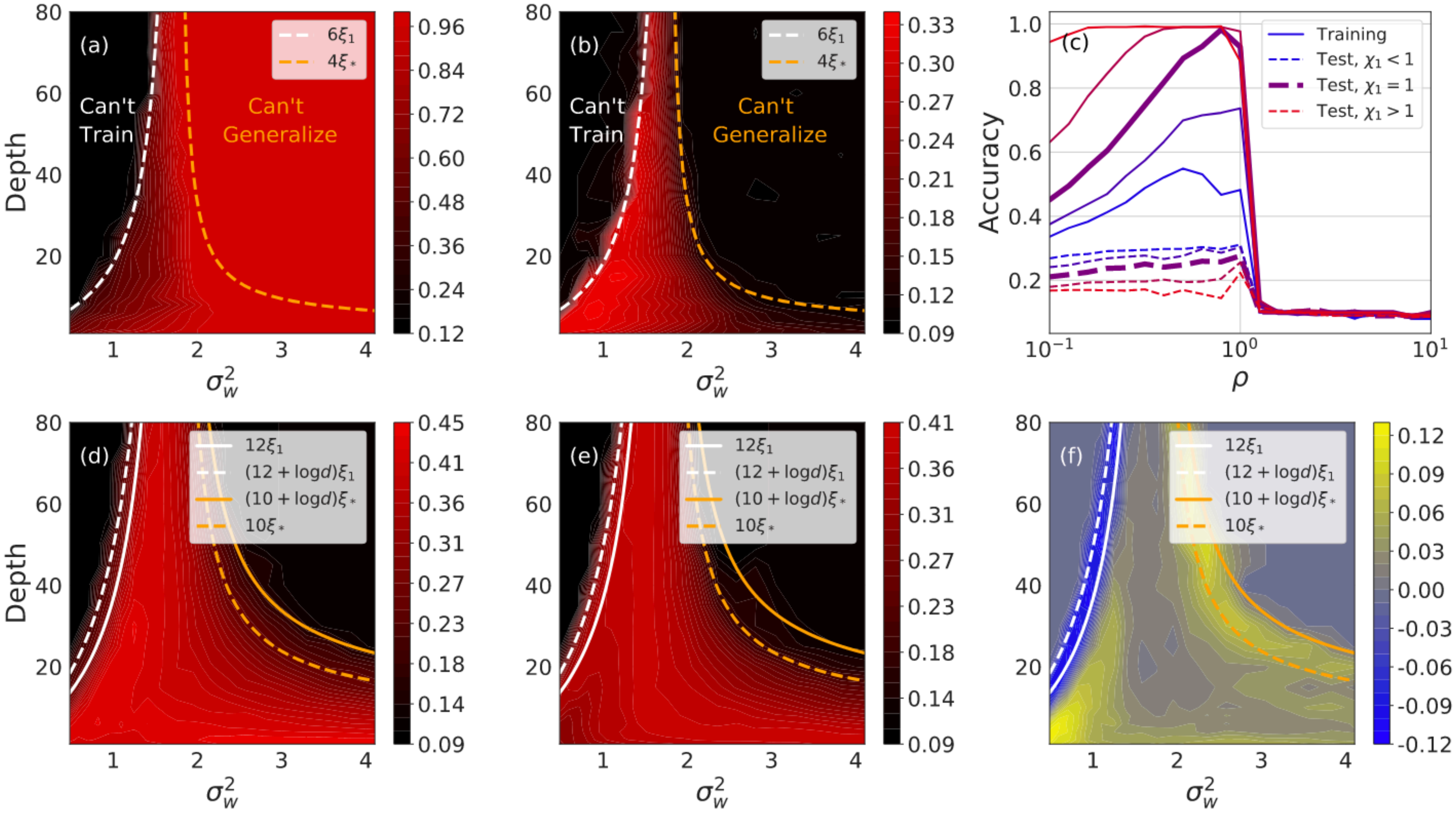


Figure 2. **Trainability and generalization are captured by $\kappa^{(l)}$ and $P(\Theta^{(l)})$.** (a,b) The training and test accuracy of CNN-F trained with SGD. The network is untrainable above the green line because $\kappa^{(l)}$ is too large and is ungeneralizable above the orange line because $P(\Theta^{(l)})$ is too small. (c) The accuracy vs learning rate for FCNs trained with SGD sweeping over the weight variance. (d,e) The test accuracy of CNN-P and CNN-F using kernel regression. (f) The difference in accuracy between CNN-P and CNN-F networks.

http://proceedings.mlr.press/v119/xiao20b/xiao20b.pdf

Drawback: computation

- For a scalar problem, the kernel matrix $\mathbf{K}$ is $N \times N$
  - Solving kernel regression exactly takes $\mathcal{O}(N^2)$ memory, ~$\mathcal{O}(N^3)$ time
  - Computing $\mathbf{K}$ is really slow / lots of memory for big architectures
    - Empirical NTK generally much faster, lower-memory than infinite NTK
  - With "normal" deep learning, everything is $\mathcal{O}(N)$

https://arxiv.org/abs/2106.07880

Drawback: computation

- For a scalar problem, the kernel matrix $\mathbf{K}$ is $N \times N$
  - Solving kernel regression exactly takes $\mathcal{O}(N^2)$ memory, ~$\mathcal{O}(N^3)$ time
  - Computing $\mathbf{K}$ is really slow / lots of memory for big architectures
    - Empirical NTK generally much faster, lower-memory than infinite NTK
  - With "normal" deep learning, everything is $\mathcal{O}(N)$
- One possible help: "sketching" approximations
  - $\hat{k}(\mathbf{x}_1, \mathbf{x}_2) = \psi(\mathbf{x}_1) \cdot \psi(\mathbf{x}_2)$ with $\psi(\mathbf{x}) \in \mathbb{R}^p$

https://arxiv.org/abs/2106.07880

Drawback: computation

- For a scalar problem, the kernel matrix $\mathbf{K}$ is $N \times N$
  - Solving kernel regression exactly takes $\mathcal{O}(N^2)$ memory, ~$\mathcal{O}(N^3)$ time
  - Computing $\mathbf{K}$ is really slow / lots of memory for big architectures
    - Empirical NTK generally much faster, lower-memory than infinite NTK
  - With "normal" deep learning, everything is $\mathcal{O}(N)$
- One possible help: "sketching" approximations
  - $\hat{k}(\mathbf{x}_1, \mathbf{x}_2) = \psi(\mathbf{x}_1) \cdot \psi(\mathbf{x}_2)$ with $\psi(\mathbf{x}) \in \mathbb{R}^p$

Table 1: Test accuracy and runtime to solve CNTK regression and its approximations on CIFAR-10. (*) means that the result is copied from Arora et al. [5].

	CNTKSketch (ours)			GradRF			Exact CNTK	CNN
Feature dimension	4,096	8,192	16,384	9,328	17,040	42,816		
Test accuracy (%)	67.58	70.46	72.06	62.49	62.57	65.21	70.47*	63.81*
Time (s)	780	1,870	5,160	300	360	580	> 1,000,000	

Table 2: MSE and runtime on large-scale UCI datasets. We measure the entire time to solve kernel ridge regression. (−) means Out-of-Memory error.

	MillionSongs		WorkLoads		CT		Protein	
# of data points (n)	467,315		179,585		53,500		39,617	
	MSE	Time (s)	MSE	Time (s)	MSE	Time (s)	MSE	Time (s)
RBF Kernel	–	–	–	–	35.37	59.23	18.96	46.45
RFF	109.50	231	4.034×10^4	53.0	48.20	15.2	19.72	12.1
NTK	–	–	–	–	30.52	72.10	20.24	76.93
NTKRF (ours)	94.27	95	3.554×10^4	35.7	46.91	2.12	20.51	4.3
NTKSketch (ours)	92.83	36	3.538×10^4	27.5	46.52	18.8	21.19	14.91

https://arxiv.org/abs/2106.07880

So, is deep learning just kernels?

So, is deep learning just kernels?

- **No.**

So, is deep learning just kernels?

- **No.**
- Real neural net optimization isn’t in the “NTK regime”

So, is deep learning just kernels?

- **No.**
- Real neural net optimization isn’t in the “NTK regime”
    - Best NTK models get ~70% accuracy on CIFAR-10, compared to 99+%

So, is deep learning just kernels?

- **No.**
- Real neural net optimization isn't in the "NTK regime"
  - Best NTK models get ~70% accuracy on CIFAR-10, compared to 99+%
- NTK regime *doesn't allow for feature learning* – the kernel doesn't change…

A problem NTKs can't learn

- Let's try to learn a single ReLU unit, $f^*(\mathbf{x}) = \mathrm{ReLU}(\langle w^*, \mathbf{x} \rangle + b^*)$
  - Some choice with $\|w^*\| = d^3$, $|b^*| \leq 6d^4 + 1$

On the Power and Limitations of Random Features for Understanding Neural Networks

Gilad Yehudai Ohad Shamir
Weizmann Institute of Science
{gilad.yehudai,ohad.shamir}@weizmann.ac.il

A problem NTKs can't learn

- Let's try to learn a single ReLU unit, $f^*(\mathbf{x}) = \mathrm{ReLU}(\langle w^*, \mathbf{x} \rangle + b^*)$
  - Some choice with $\|w^*\| = d^3$, $|b^*| \leq 6d^4 + 1$
  - Gradient descent can learn this with polynomially many samples

On the Power and Limitations of Random Features
for Understanding Neural Networks

Gilad Yehudai Ohad Shamir
Weizmann Institute of Science
{gilad.yehudai,ohad.shamir}@weizmann.ac.il

A problem NTKs can't learn

- Let's try to learn a single ReLU unit, $f^*(\mathbf{x}) = \mathrm{ReLU}(\langle w^*, \mathbf{x}\rangle + b^*)$
  - Some choice with $\|w^*\| = d^3$, $|b^*| \le 6d^4 + 1$
  - Gradient descent can learn this with polynomially many samples
  - Kernel-based methods require at least one of

On the Power and Limitations of Random Features
for Understanding Neural Networks

Gilad Yehudai Ohad Shamir
Weizmann Institute of Science
{gilad.yehudai,ohad.shamir}@weizmann.ac.il

A problem NTKs can't learn

- Let's try to learn a single ReLU unit, $f^*(\mathbf{x}) = \mathrm{ReLU}(\langle w^*, \mathbf{x} \rangle + b^*)$
  - Some choice with $\|w^*\| = d^3$, $|b^*| \leq 6d^4 + 1$
  - Gradient descent can learn this with polynomially many samples
  - Kernel-based methods require at least one of
    - exponentially many samples

On the Power and Limitations of Random Features
for Understanding Neural Networks

Gilad Yehudai Ohad Shamir
Weizmann Institute of Science
{gilad.yehudai,ohad.shamir}@weizmann.ac.il

A problem NTKs can't learn

- Let's try to learn a single ReLU unit, $f^*(\mathbf{x}) = \mathrm{ReLU}(\langle w^*, \mathbf{x}\rangle + b^*)$
  - Some choice with $\|w^*\| = d^3$, $|b^*| \le 6d^4 + 1$
  - Gradient descent can learn this with polynomially many samples
  - Kernel-based methods require at least one of
    - exponentially many samples
    - exponentially large RKHS norm (i.e. hard to learn)

On the Power and Limitations of Random Features
for Understanding Neural Networks

Gilad Yehudai Ohad Shamir
Weizmann Institute of Science
{gilad.yehudai,ohad.shamir}@weizmann.ac.il

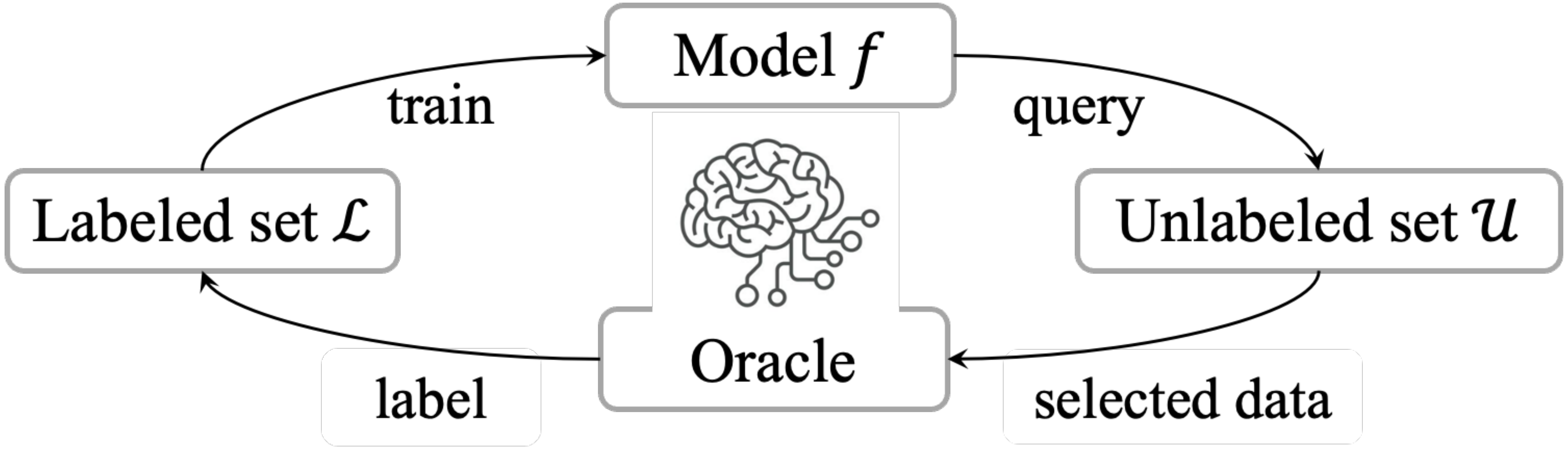
Quantifying the Benefit of Using Differentiable Learning over Tangent Kernels

Eran Malach Hebrew University of Jerusalem eran.malach@mail.huji.ac.il
Pritish Kamath Toyota Technological Institute at Chicago pritish@ttic.edu
Emmanuel Abbe EPFL emmanuel.abbe@epfl.ch
Nathan Srebro Toyota Technological Institute at Chicago nati@ttic.edu
Collaboration on the Theoretical Foundations of Deep Learning (deepfoundations.ai)

<table>
<tr><th colspan="2"></th><th>NTK at same Initialization</th><th>NTK at alternate randomized Initialization</th><th>NTK of arbitrary model or even an arbitrary Kernel</th></tr>
<tr><td colspan="2">GD with unbiased initialization ($\forall_x f_{\theta_0}(x) = 0$) ensures small error</td><td>▶ NTK edge $\geq \text{poly}^{-1}$ (Thm. 1)
▶ NTK edge can be $< \text{poly}^{-1}$ while GD reaches 0 loss (Separation 1)</td><td colspan="2">Edge with any kernel can be $< \text{poly}^{-1}$ while GD reaches 0 loss (Separation 2)</td></tr>
<tr><td rowspan="2">GD with arbitrary init. ensures small error</td><td>Kernel (or alt init) can depend on input dist. $\mathcal{D}_{\mathcal{X}}$</td><td rowspan="2">NTK edge can be $= 0$ while GD reaches arb. low loss (Separation 3)</td><td>▶ NTK edge $\geq \text{poly}^{-1}$ (Thm. 2)
▶ NTK edge can be $< \text{poly}^{-1}$ while GD reaches 0 loss (Separation 2)</td><td>Edge can be $< \text{poly}^{-1}$ while GD reaches 0 loss (Separation 2)</td></tr>
<tr><td>Dist-indep kernels</td><td colspan="2">edge with any kernel can be $< \exp^{-1}$ while GD reaches arb. low loss (Separation 4)</td></tr>
</table>

Uses of empirical NTKs

One application: active learning

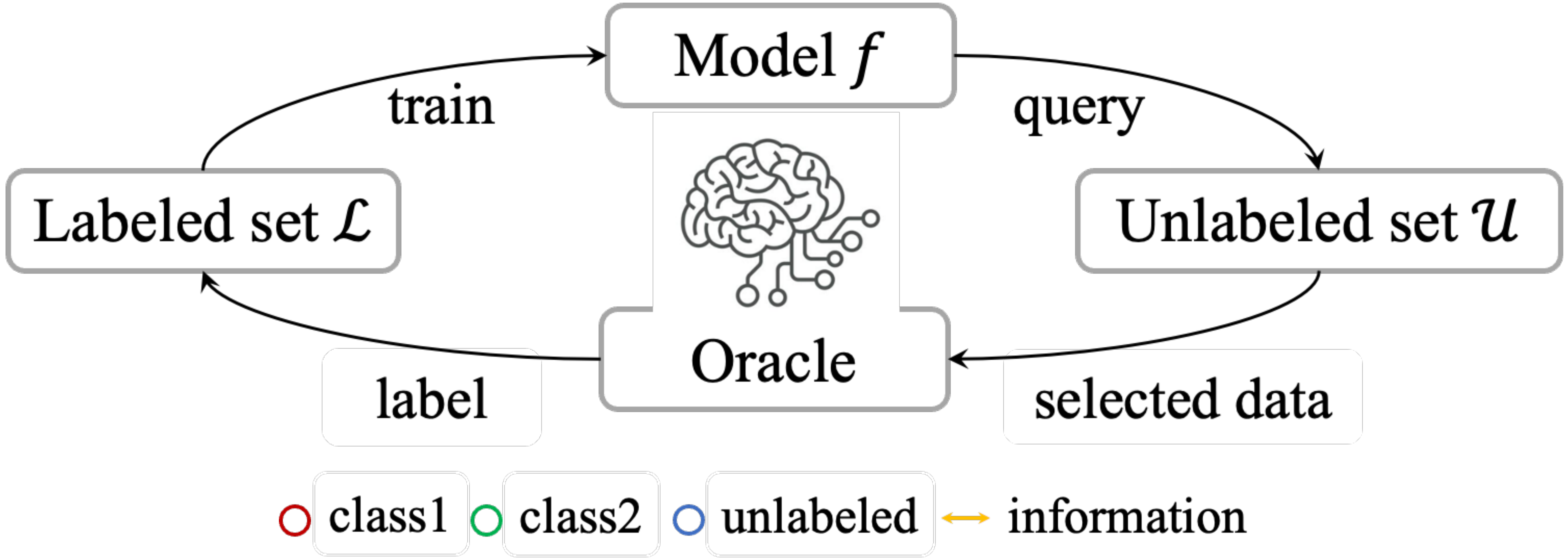


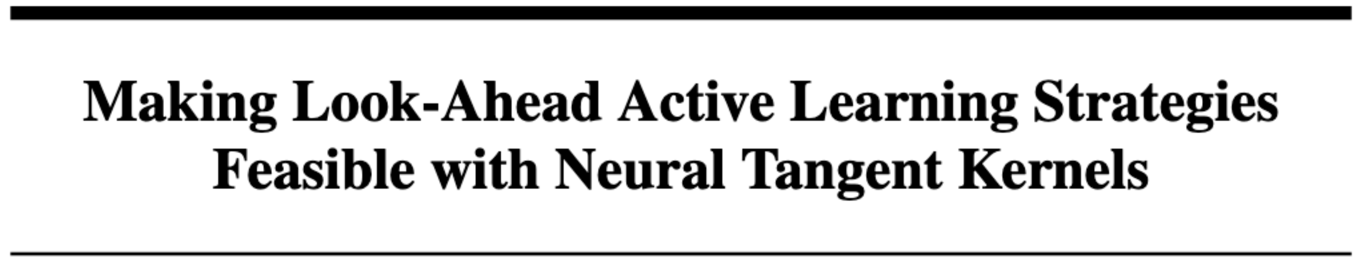
Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels

Mohamad Amin Mohamadi* University of British Columbia lemohama@cs.ubc.ca
Wonho Bae* University of British Columbia whbae@cs.ubc.ca
Danica J. Sutherland UBC & Amii dsuth@cs.ubc.ca

https://arxiv.org/abs/2206.12569

One application: active learning




Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels

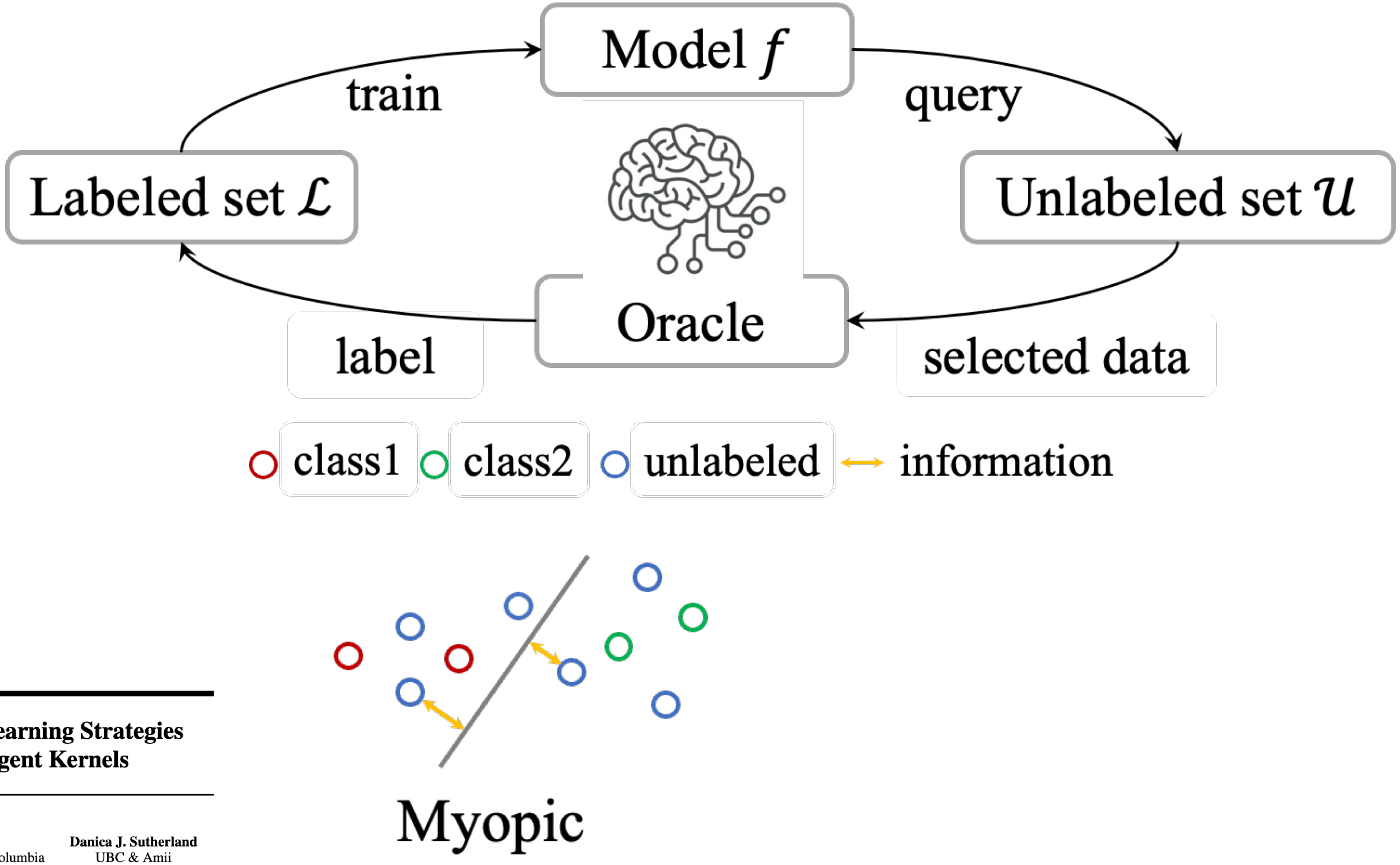
Mohamad Amin Mohamadi*
University of British Columbia
lemohama@cs.ubc.ca

Wonho Bae*
University of British Columbia
whbae@cs.ubc.ca

Danica J. Sutherland
UBC & Amii
dsuth@cs.ubc.ca

https://arxiv.org/abs/2206.12569

One application: active learning



Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels

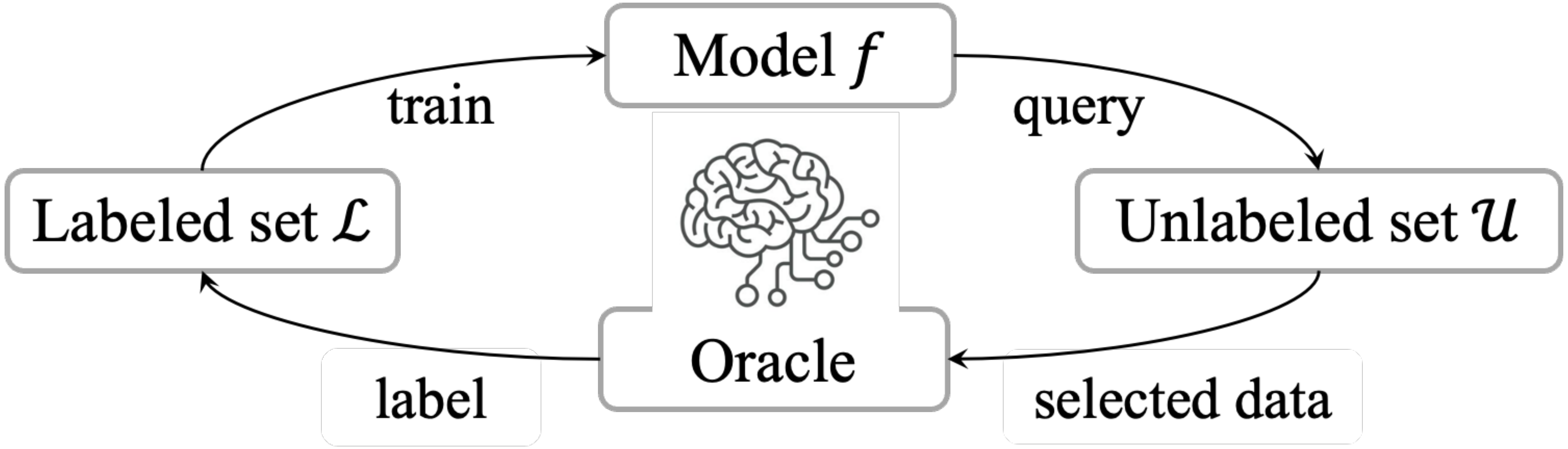
Mohamad Amin Mohamadi* University of British Columbia lemohama@cs.ubc.ca

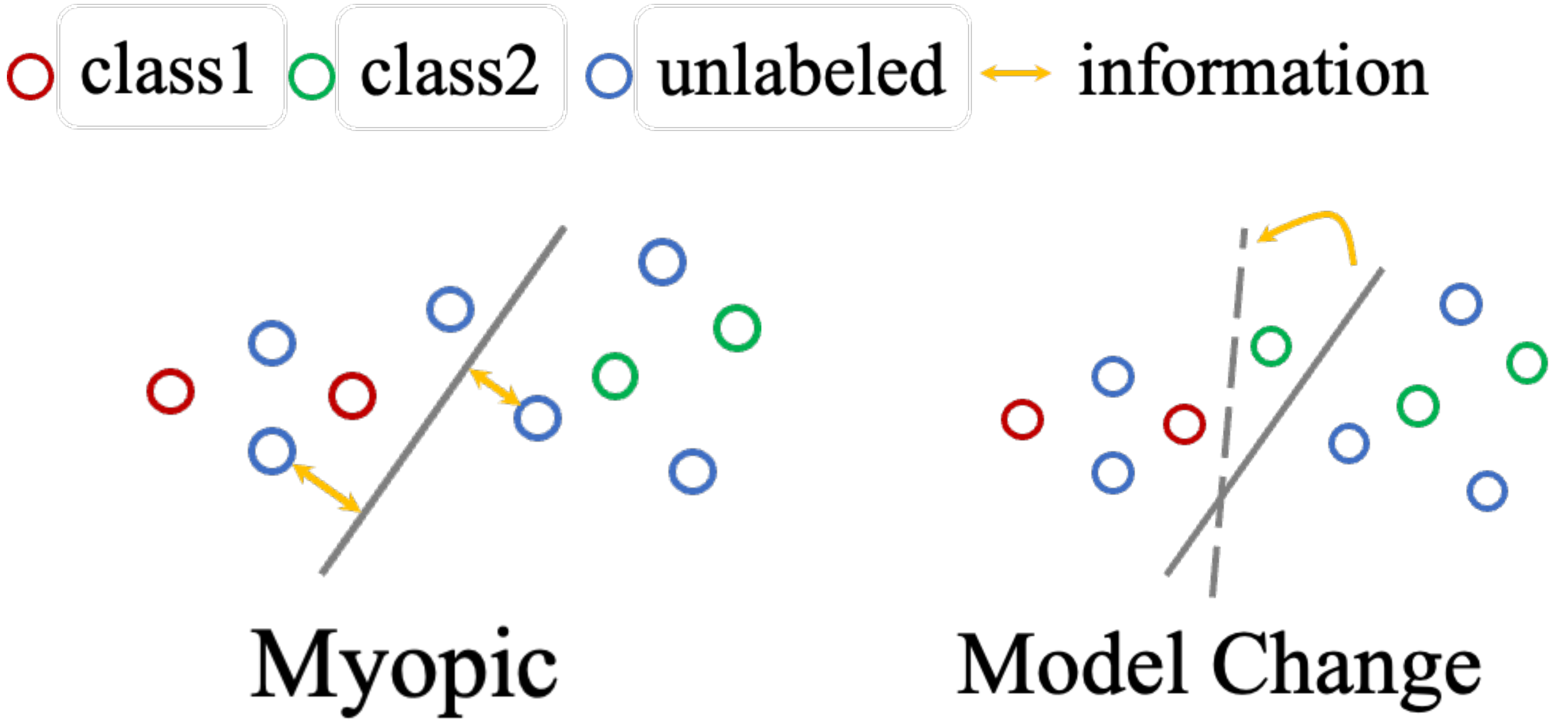
Wonho Bae* University of British Columbia whbae@cs.ubc.ca

Danica J. Sutherland UBC & Amii dsuth@cs.ubc.ca

https://arxiv.org/abs/2206.12569

One application: active learning





Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels

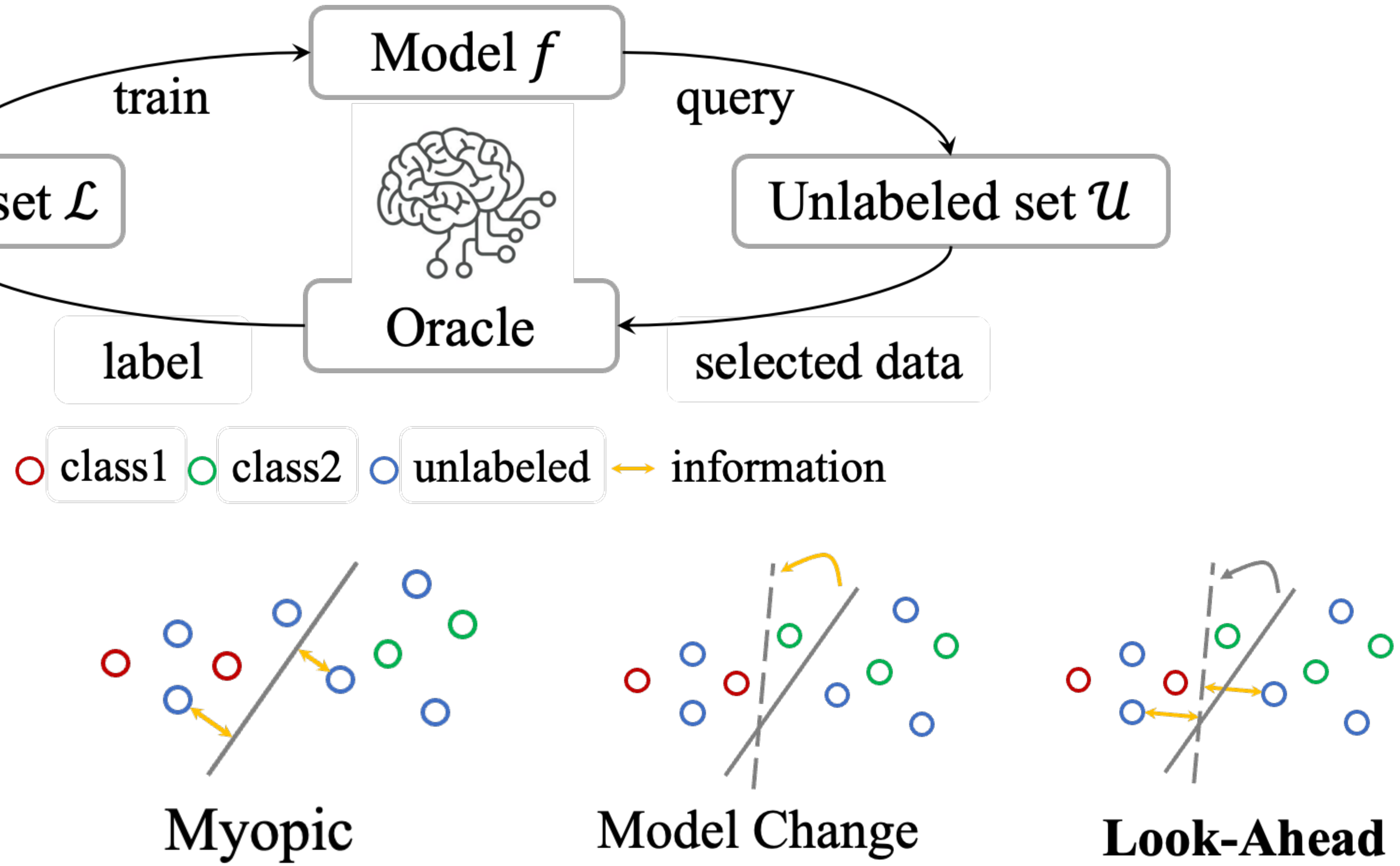
Mohamad Amin Mohamadi* University of British Columbia lemohama@cs.ubc.ca

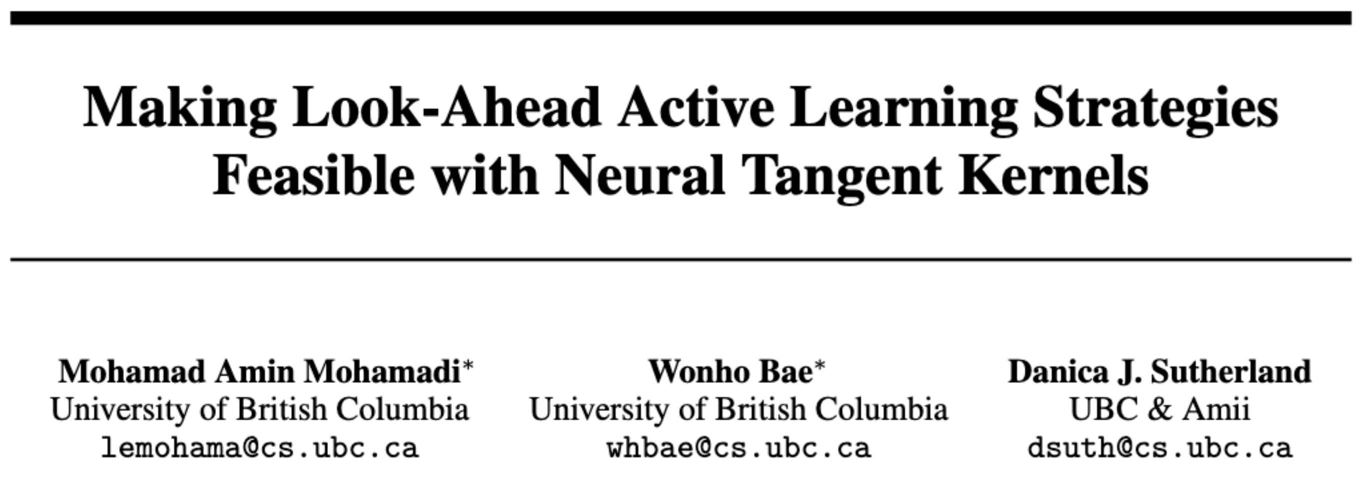
Wonho Bae* University of British Columbia whbae@cs.ubc.ca

Danica J. Sutherland UBC & Amii dsuth@cs.ubc.ca

https://arxiv.org/abs/2206.12569

One application: active learning




Making Look-Ahead Active Learning Strategies Feasible with Neural Tangent Kernels

Mohamad Amin Mohamadi* University of British Columbia lemohama@cs.ubc.ca
Wonho Bae* University of British Columbia whbae@cs.ubc.ca
Danica J. Sutherland UBC & Amii dsuth@cs.ubc.ca

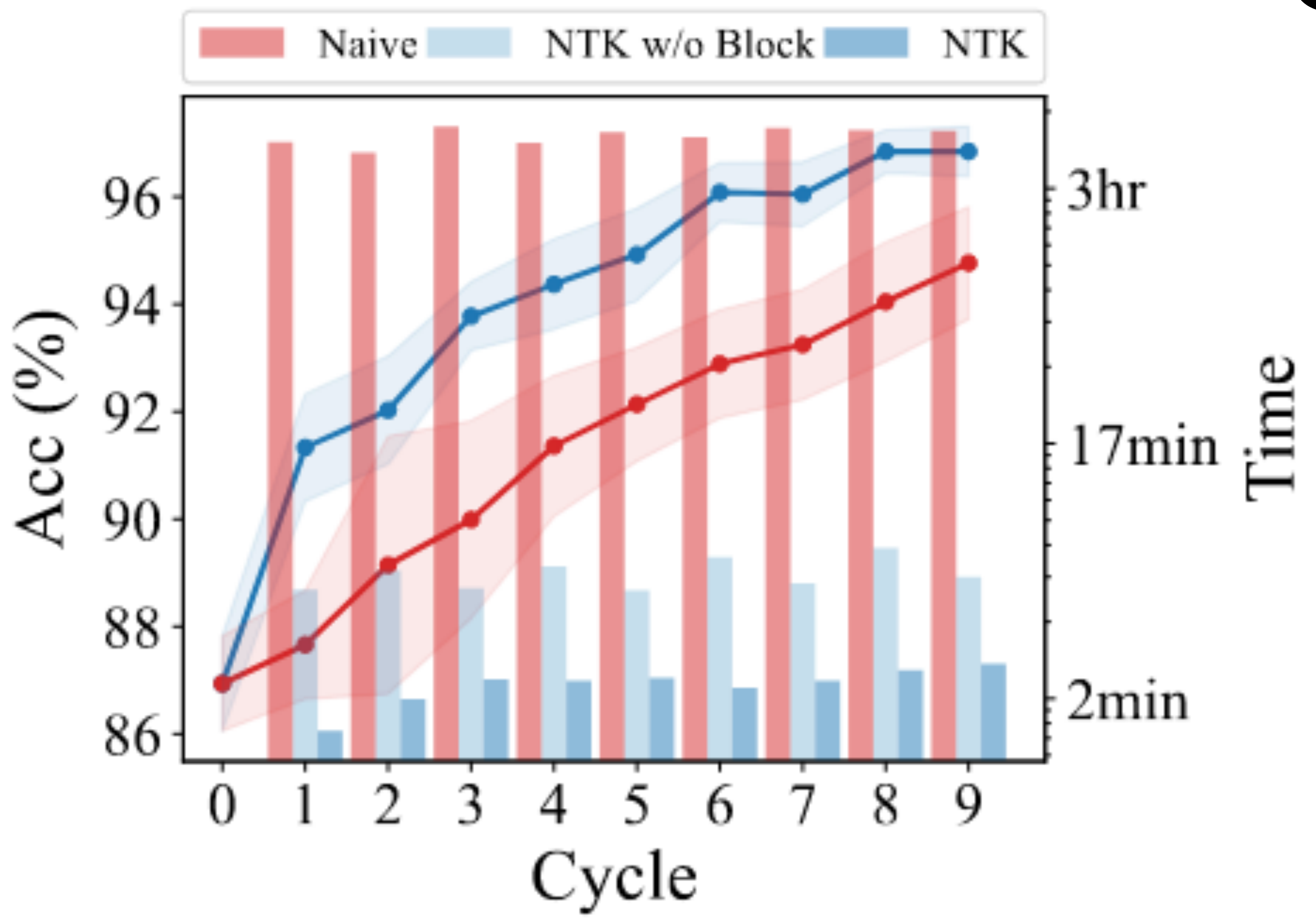
https://arxiv.org/abs/2206.12569

Active learning: look-ahead criteria

- Given a model $f(\mathbf{x}; \mathbf{w})$ trained on S
- What our model would be if we **retrained** on $S^+ = S \cup \{(\mathbf{x}^+, y^+)\}$?
  - Too expensive to actually retrain
- We can take a Taylor expansion around current weights $\mathbf{w}$
- Then ask questions about $f^{lin}(x; \mathbf{w}^+)$ to pick which point to query
  - e.g. measure $\sum_{\mathbf{x} \in \mathcal{U}} \|f^{lin}(\mathbf{x}; \mathbf{w}^+) - f(\mathbf{x}; \mathbf{w})\|$

https://arxiv.org/abs/2206.12569

Active learning with empirical NTKs

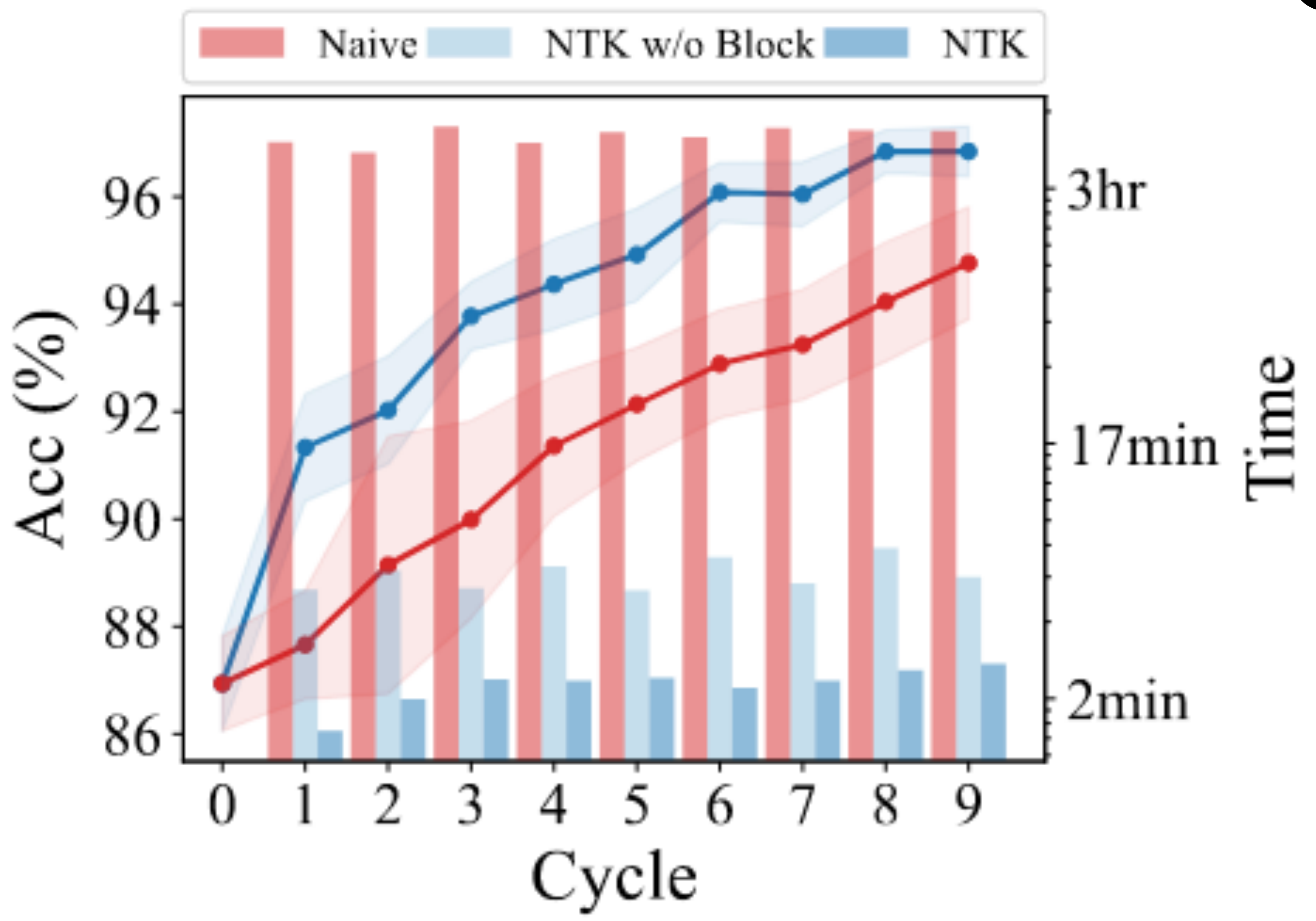


(a) Naïve look-ahead acquisition versus NTK approximation. Bars show runtime per cycle.

- Far faster + better performance than actually retraining with the new data

https://arxiv.org/abs/2206.12569

Active learning with empirical NTKs

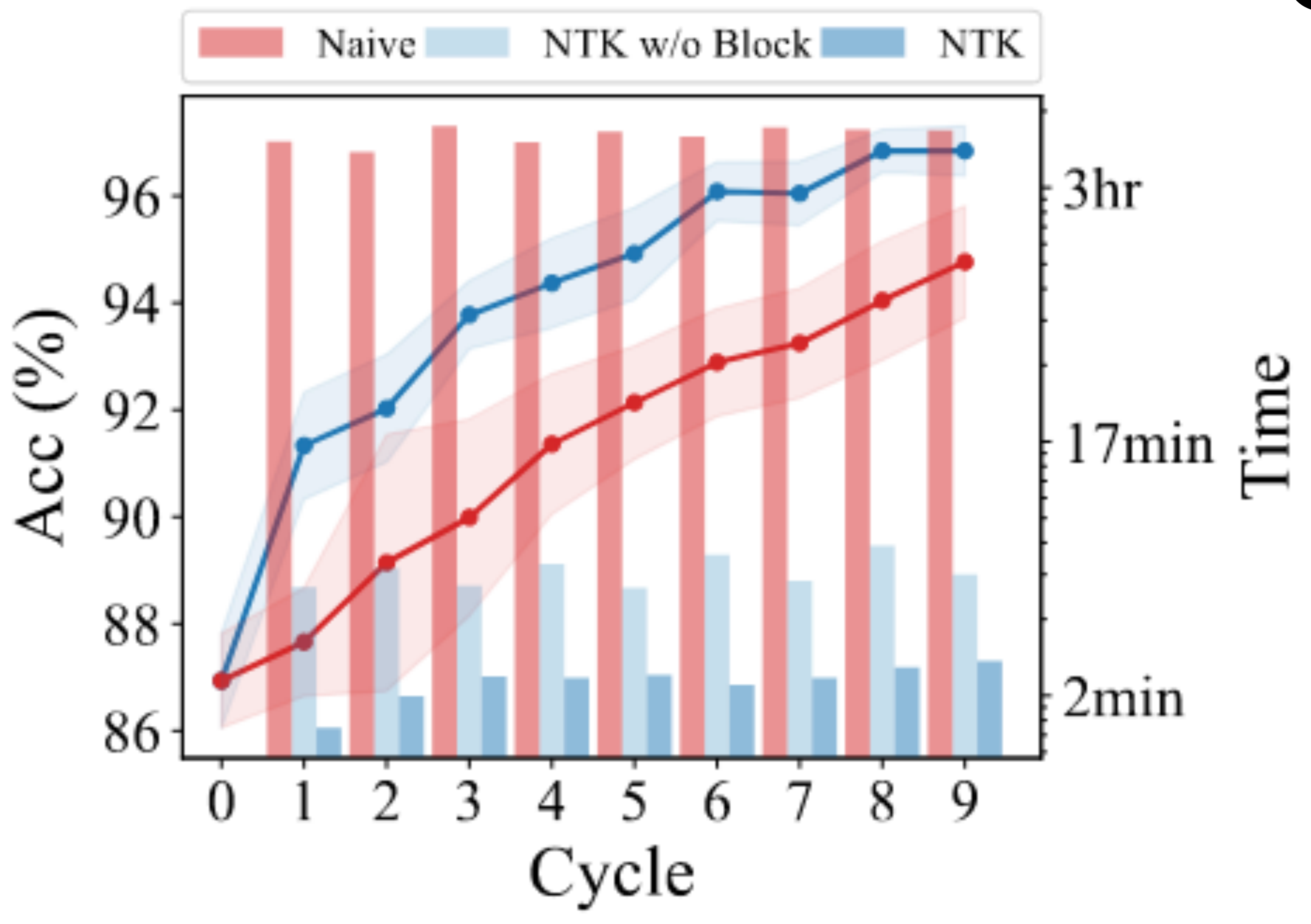


(a) Naïve look-ahead acquisition versus NTK approximation. Bars show runtime per cycle.

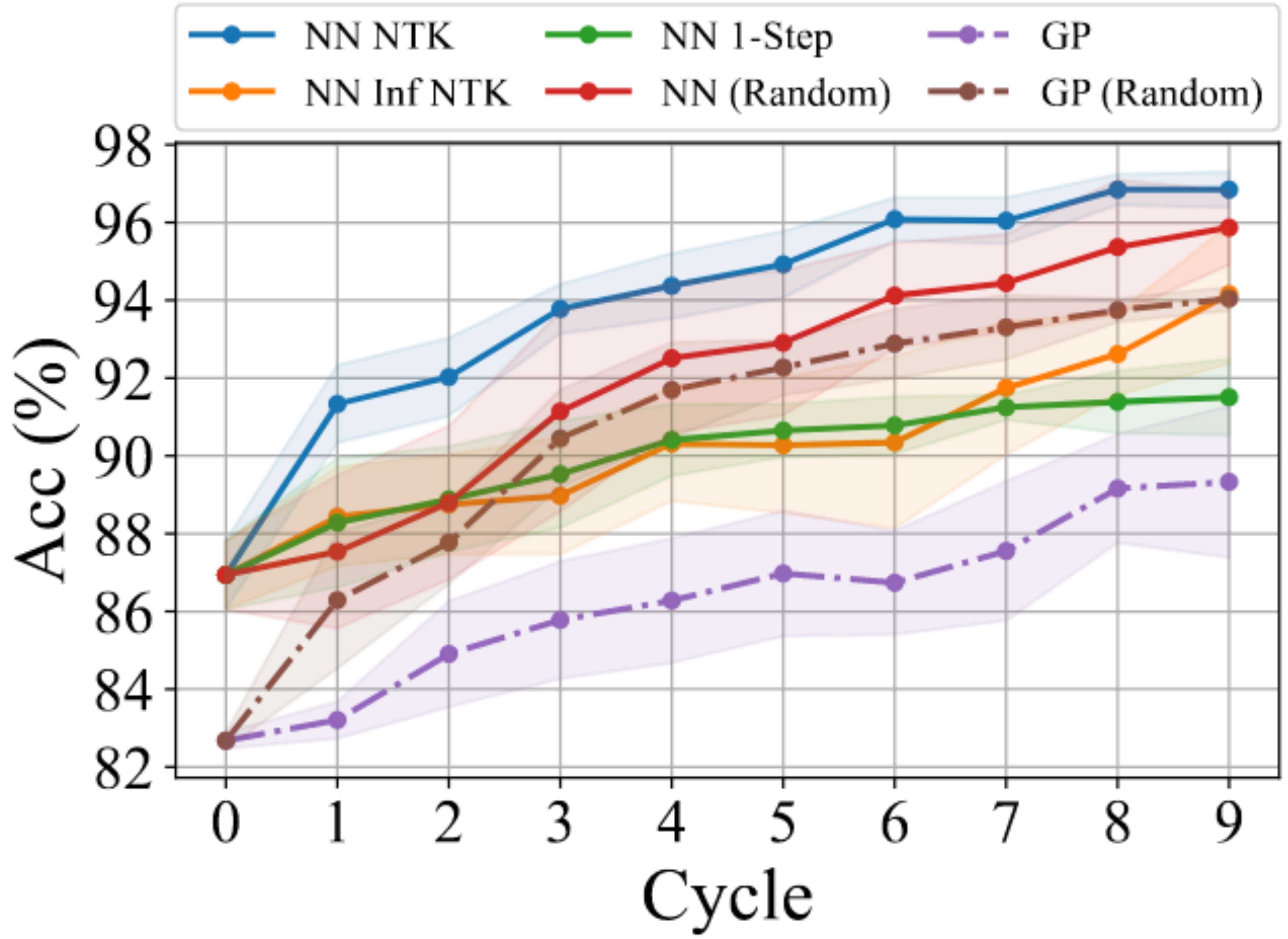
- Far faster + better performance than actually retraining with the new data
- Much better "understanding" of retraining behaviour than infinite NTK or one step of SGD

https://arxiv.org/abs/2206.12569

Active learning with empirical NTKs



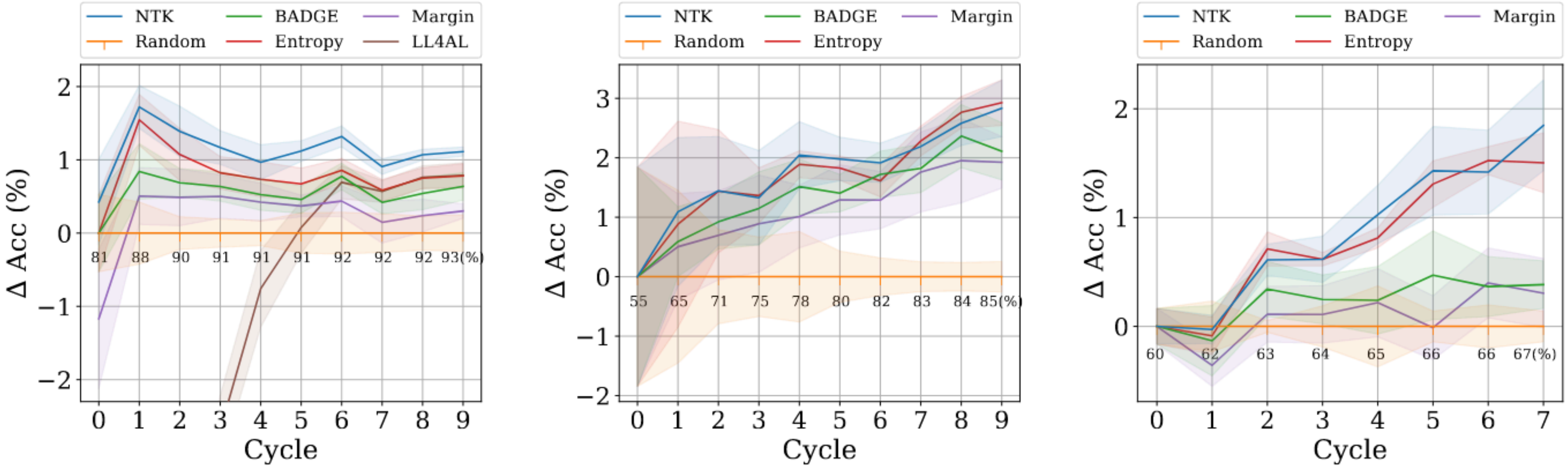
(a) Naïve look-ahead acquisition versus NTK approximation. Bars show runtime per cycle.



- Far faster + better performance than actually retraining with the new data
- Much better “understanding” of retraining behaviour than infinite NTK or one step of SGD

Active learning with empirical NTKs

- Matches or beats other active learning methods



(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.

Predicting generalization

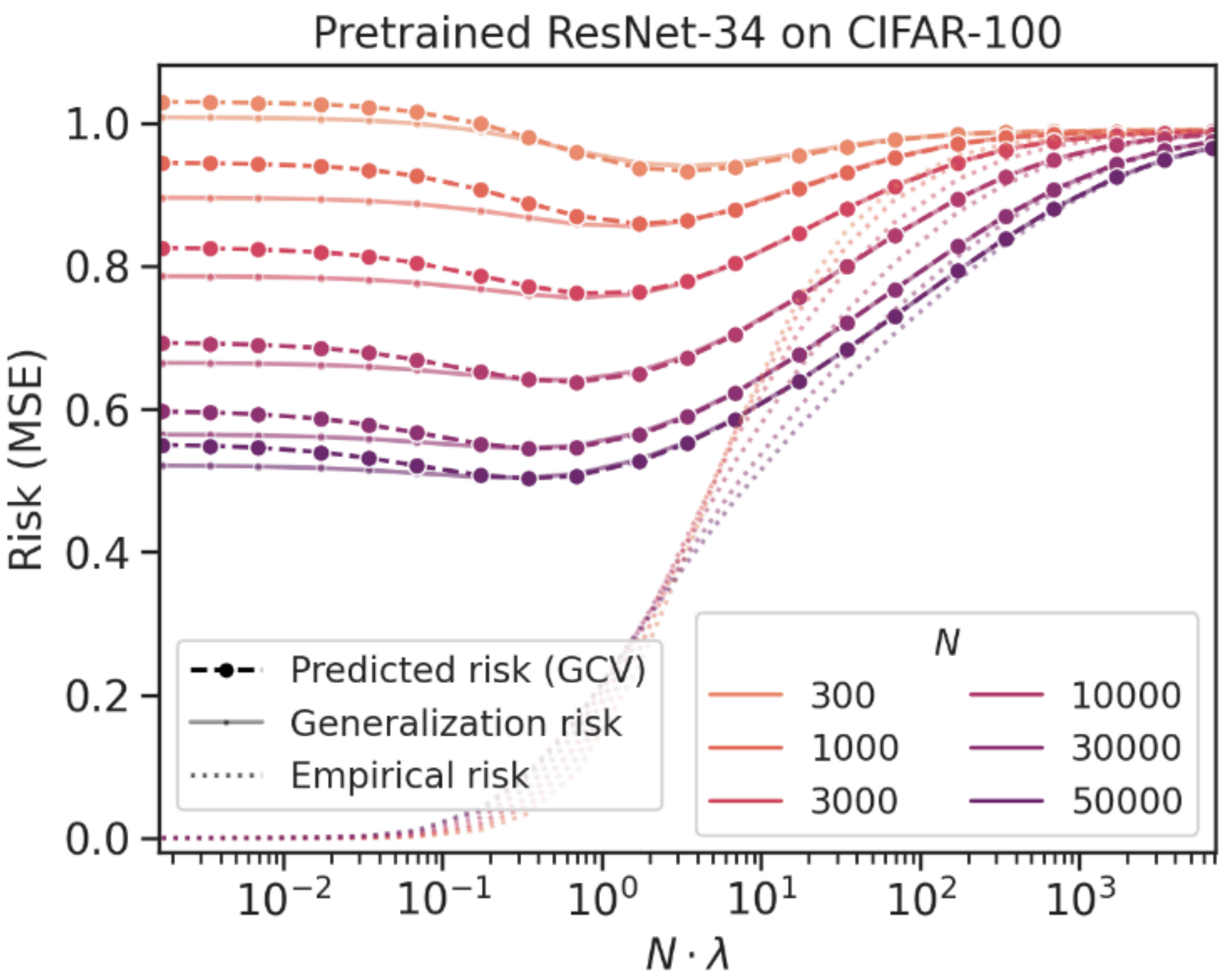


Figure 1. Predicted vs. actual generalization risk of a pretrained ResNet-34 empirical NTK on CIFAR-100 over dataset sizes N and ridge regularizations λ. Corresponding training risks are plotted in the background. The fit achieving the lowest MSE has 19.9% test error on CIFAR-100 (vs. 15.9% from finetuning the ResNet).

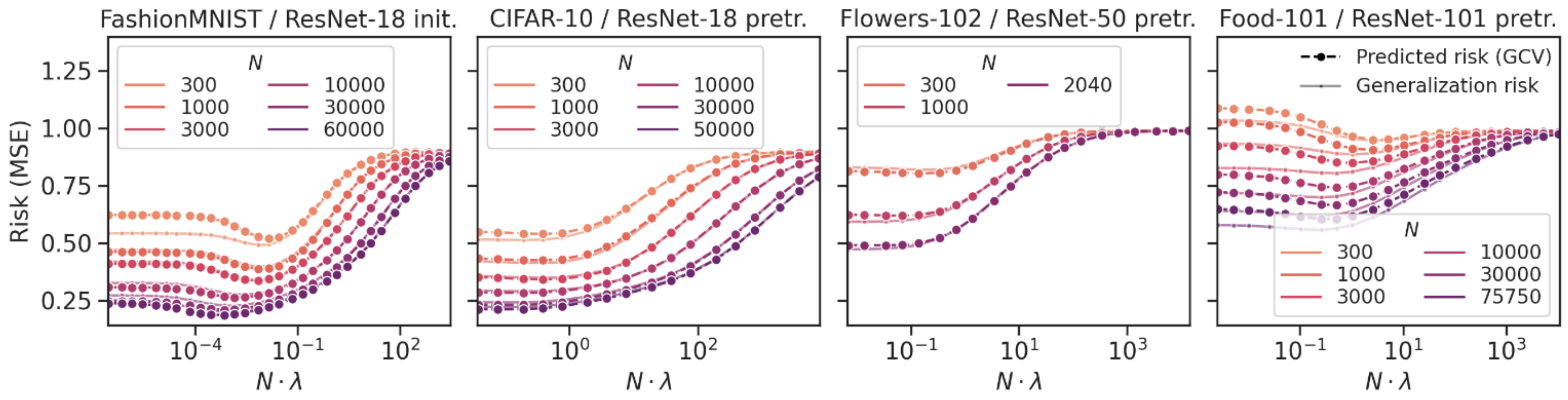


Figure 4. Generalization risk vs. the GCV prediction, for various datasets and networks, across sample sizes N and regularization levels λ.

- Uses generalized cross-validation to estimate how well a network will generalize on a new dataset after you fine-tune it

More Than a Toy: Random Matrix Models Predict How Real-World Neural Representations Generalize

Alexander Wei [1] Wei Hu [1] Jacob Steinhardt [1]

https://arxiv.org/abs/2203.06176

Computing empirical NTKs

- If we have O outputs, $\mathbf{K}$ is $NO \times NO$
  - CIFAR-10: $N = 60{,}000$, $O = 10$: 2.8 terabytes in memory
  - Imagenet: $N \approx 1{,}200{,}000$, $O = 1000$: 11,520,000 terabytes in memory

- For the infinite NTK, we can actually ignore the O part
  - Has form $\mathbf{K} \otimes \mathbf{I}$ because of the last layer – corresponds to doing an independent kernel regression for each component
  - CIFAR-10 becomes 29 gigabytes, ImageNet 11.52 terabytes

- For empirical NTK, we can get rid of the O too!

Pseudo-NTK

- $$\tilde{k}_{\mathbf{w}}(\mathbf{x}_1, \mathbf{x}_2) = \left[\nabla_{\mathbf{w}} \frac{1}{\sqrt{O}} \sum_{j=1}^{O} f_j(\mathbf{x}_1; \mathbf{w}) \right] \cdot \left[\nabla_{\mathbf{w}} \frac{1}{\sqrt{O}} \sum_{j=1}^{O} f_j(\mathbf{x}_2; \mathbf{w}) \right]$$
    - The kernel, its largest eigenvalue, and kernel regression outputs all converge to the full eNTK result at rate $\mathcal{O}(1/\sqrt{m})$

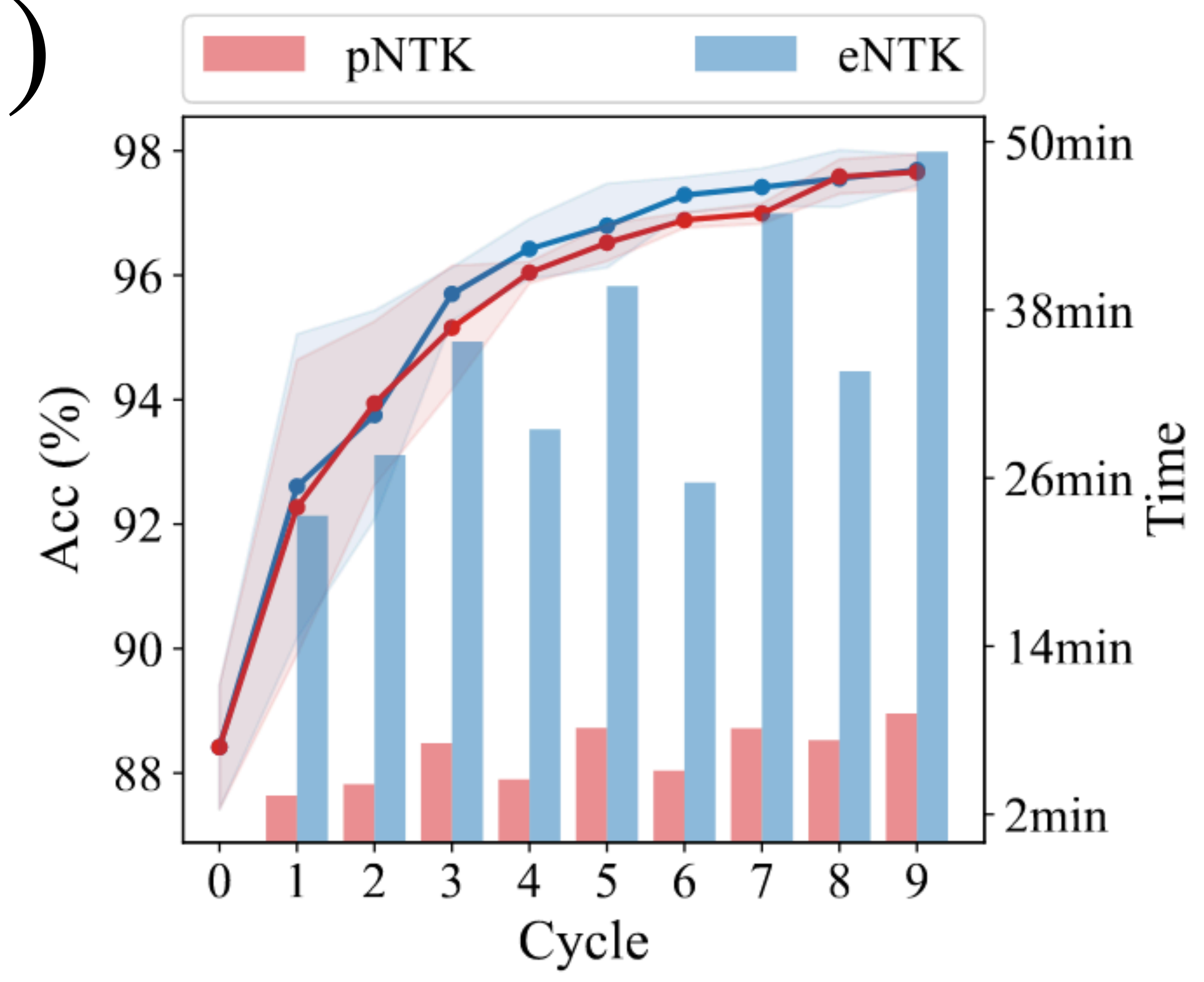


Figure 12: **Comparison of pNTK with eNTK on a look-ahead active learning task.** pNTK is much faster than eNTK without losing performance.

A Fast, Well-Founded Approximation to the Empirical Neural Tangent Kernel

Mohamad Amin Mohamadi [1] **Wonho Bae** [1] **Danica J. Sutherland** [1 2]

https://arxiv.org/abs/2206.12543

Pseudo-NTK: kernel approximation

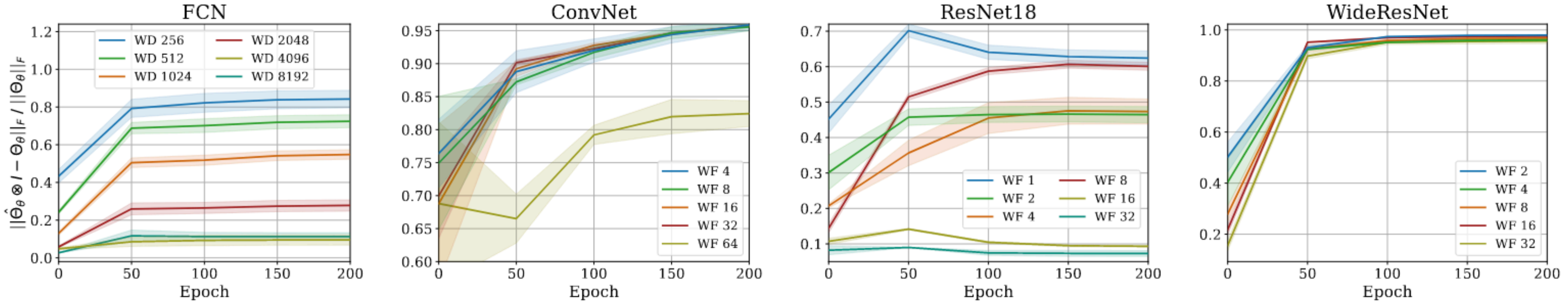


Figure 3: Evaluating the **relative difference of Frobenius norm of** $\Theta_\theta(\mathcal{D}, \mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D}, \mathcal{D}) \otimes I_O$ at initialization and throughout training, based on $\mathcal{D}$ being 1000 random points from CIFAR-10. Wider nets have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

https://arxiv.org/abs/2206.12543

Pseudo-NTK: regression results

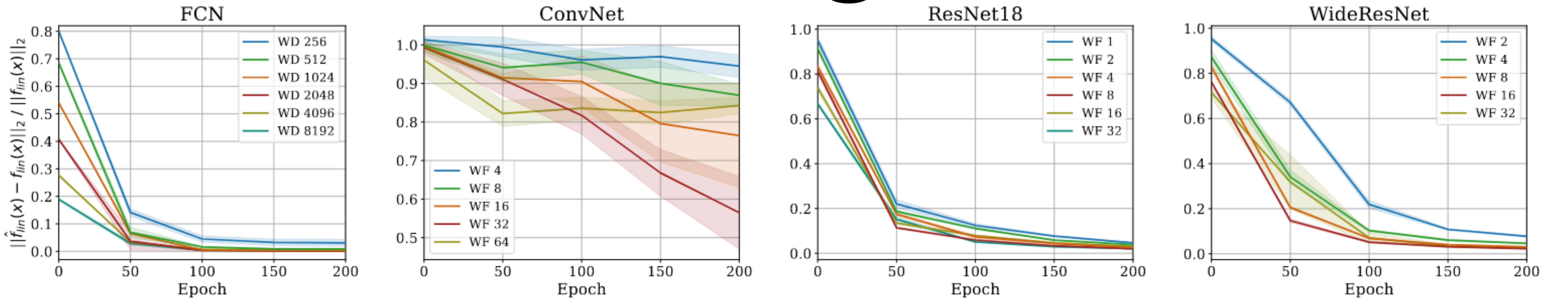


Figure 7: The **relative difference of kernel regression outputs**, (4) and (5), when training on $|\mathcal{D}| = 1000$ random CIFAR-10 points and testing on $|\mathcal{X}| = 500$. For wider NNs, the relative difference in $\hat{f}^{lin}(\mathcal{X})$ and $f^{lin}(\mathcal{X})$ decreases at initialization.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

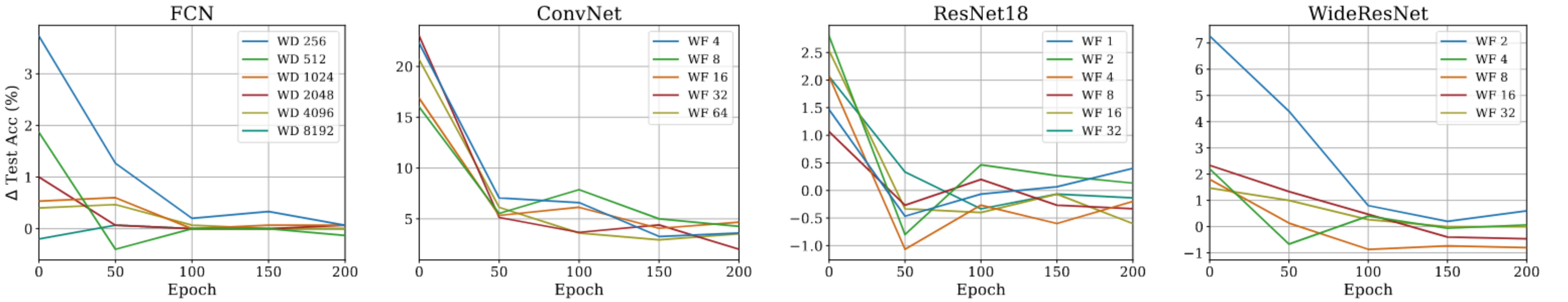


Figure 8: **Using pNTK in kernel regression** (as in Figure 7) **almost always achieves a higher test accuracy than using eNTK.** Wider NNs and trained nets have more similar prediction accuracies of $\hat{f}^{lin}$ and f^{lin} at initialization. Again, the difference between these two continues to vanish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

Pseudo-NTK on full CIFAR-10

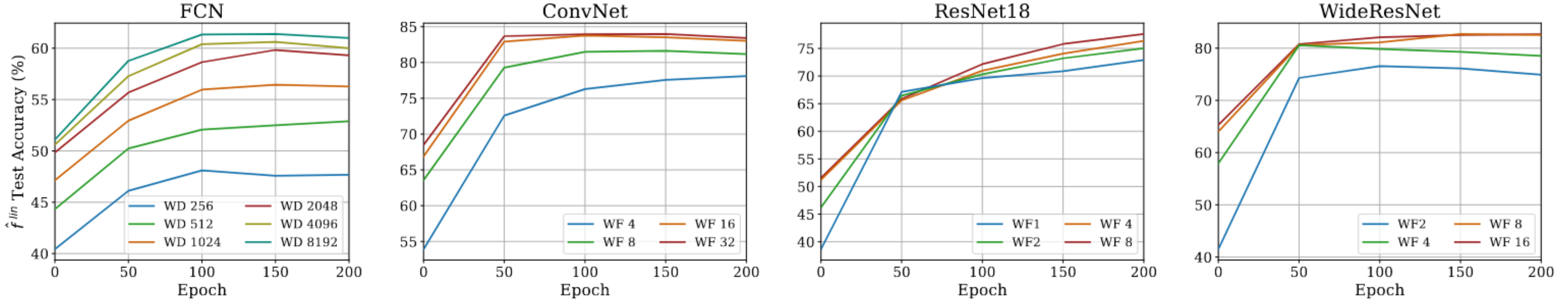


Figure 9: Evaluating the **test accuracy of kernel regression predictions using pNTK as in (5) on the full CIFAR-10 dataset**. As the NN's width grows, the test accuracy of $\hat{f}^{lin}$ also improves, but eventually saturates with the growing width. Using trained weights in computation of pNTK results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

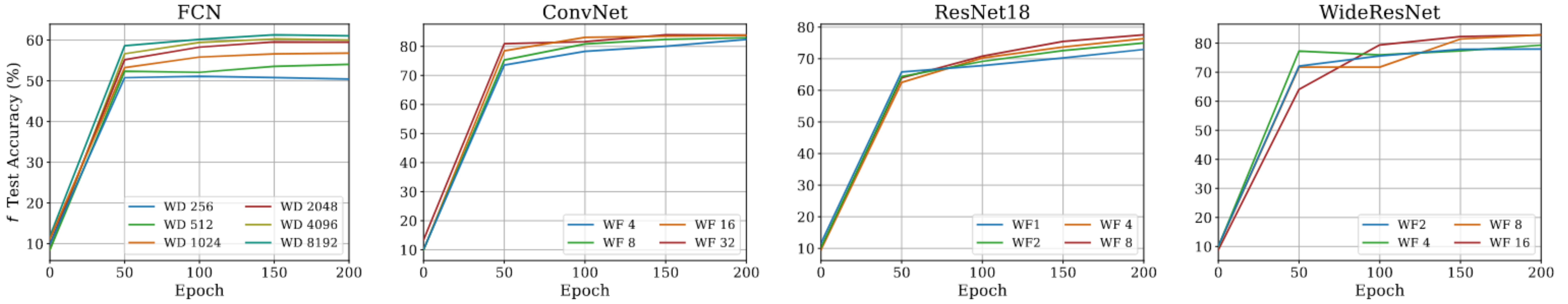


Figure 10: Evaluating the **test accuracy of model f throughout SGD training on the full CIFAR-10 dataset**. In contrast to $\hat{f}^{lin}$, the test accuracy of f does not significantly improve with growing width.

Understanding learning dynamics

- Interesting insights, worth reading – but based on NTKs from 500 samples only, since they didn't have the pNTK!
- In particular, it seems empirical NTK *does* meaningfully change later in the process than they were able to notice

Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the Neural Tangent Kernel

Stanislav Fort[1*] **Gintare Karolina Dziugaite**[2*] **Mansheej Paul**[1]
Sepideh Kharaghani[2] **Daniel M. Roy**[3,4] **Surya Ganguli**[1]
[1]Stanford University [2]Element AI [3]University of Toronto [4]Vector Institute

https://arxiv.org/abs/2010.15110

Understanding learning dynamics

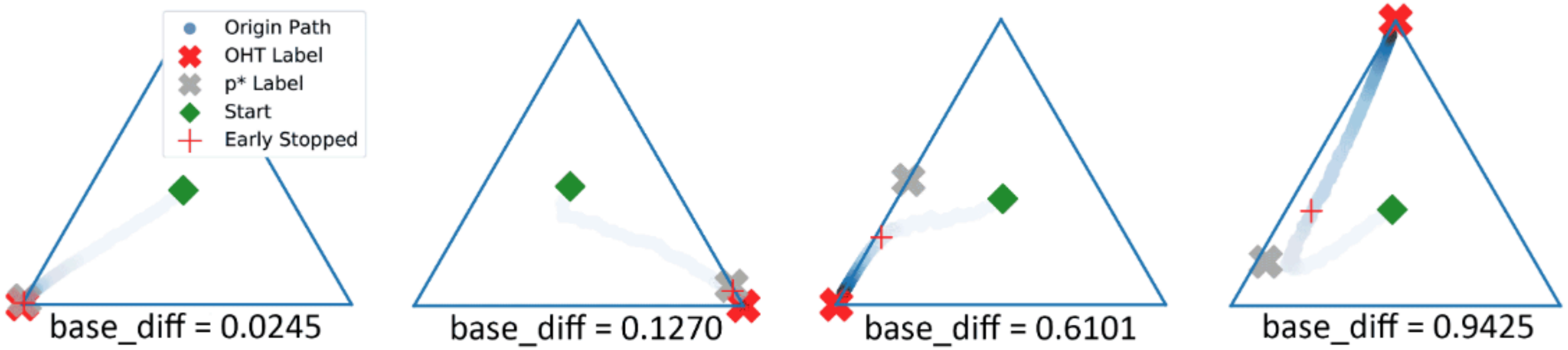
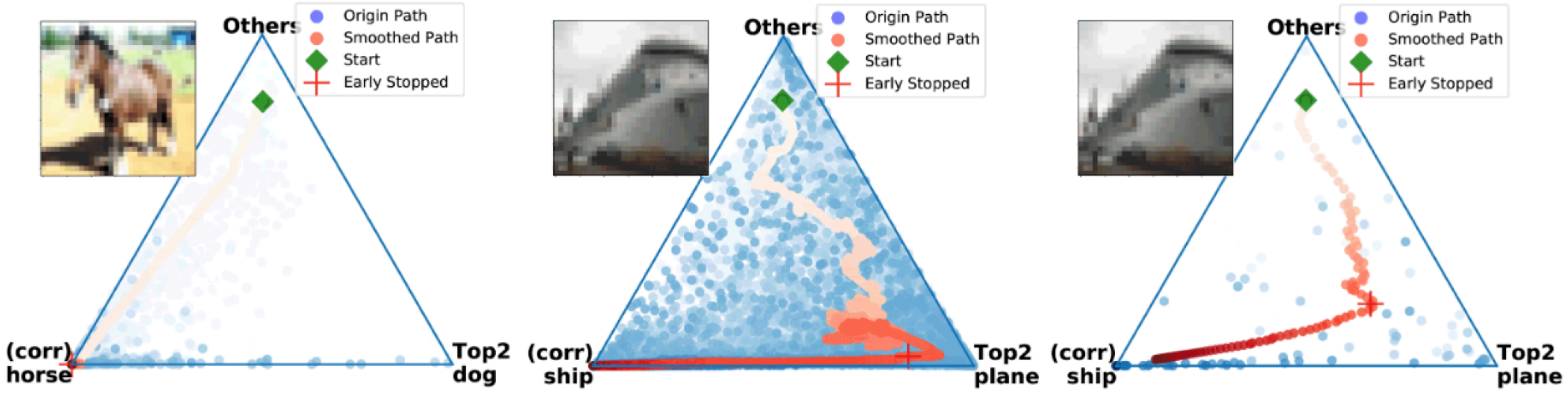


Figure 3: Learning path of samples with different base difficulty. Corners correspond to one-hot vectors. Colors represent training time: transparent at initialization, dark blue at the end of training.



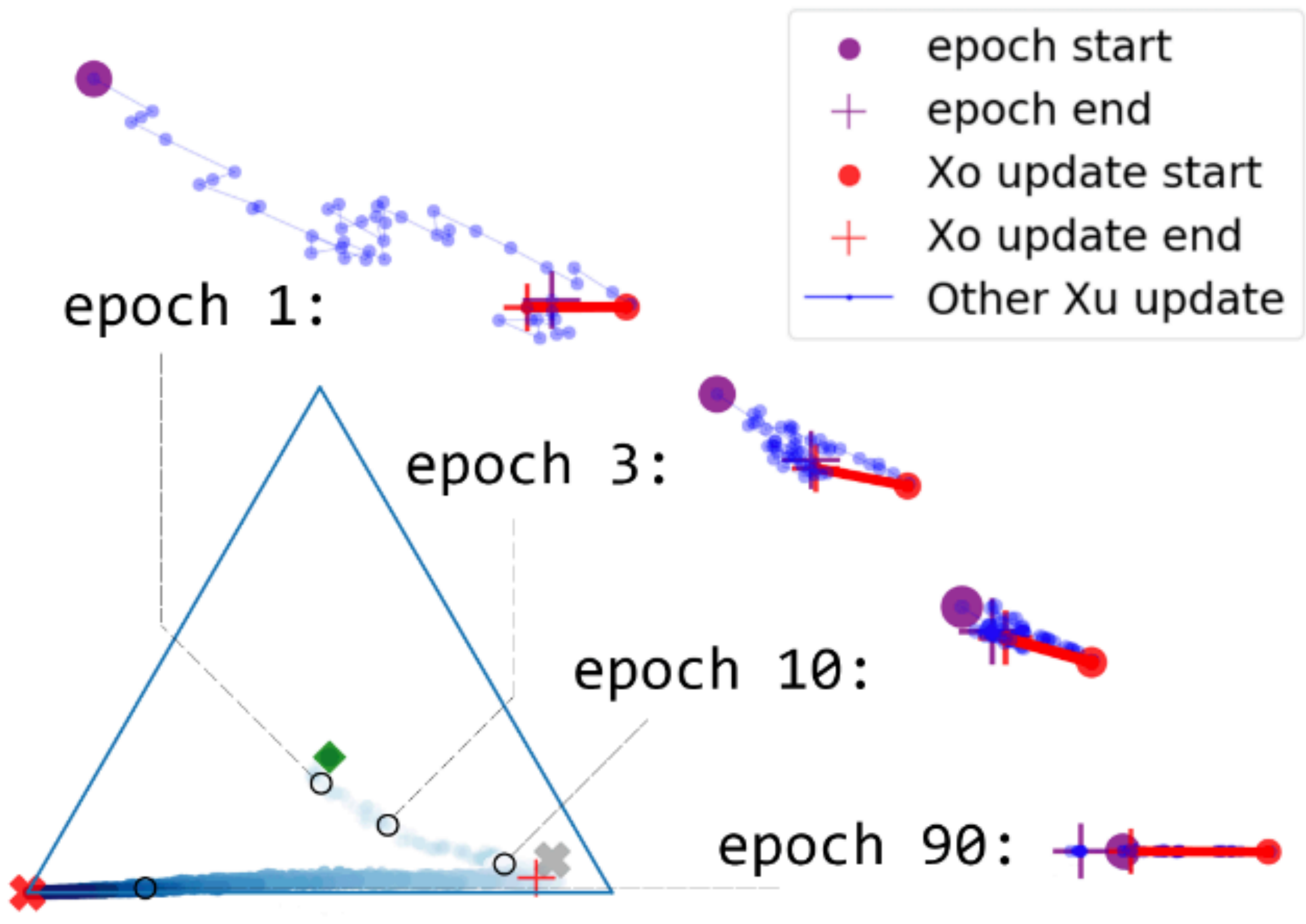


Figure 4: Updates of $\mathbf{q}(\mathbf{x}_o)$ over training.

BETTER SUPERVISORY SIGNALS
BY OBSERVING LEARNING PATHS

Yi Ren
UBC
renyi.joshua@gmail.com

Shangmin Guo
University of Edinburgh
s.guo@ed.ac.uk

Danica J. Sutherland
UBC and Amii
dsuth@cs.ubc.ca

Understanding learning dynamics

Proposition 1. *Let* $\mathbf{z}^t(\mathbf{x}) \triangleq f(\mathbf{w}^t, \mathbf{x})$ *denote the network output logits with parameters* $\mathbf{w}^t$*, and* $\mathbf{q}^t(\mathbf{x}) = \mathrm{Softmax}(\mathbf{z}^t(\mathbf{x}))$ *the probabilities. Let* $\mathbf{w}^{t+1} \triangleq \mathbf{w}^t - \eta \nabla_{\mathbf{w}} \left(\mathbf{p}_{tar}(\mathbf{x}_u)^\mathsf{T} \mathbf{L}(\mathbf{q}^t(\mathbf{x}_u))\right)$ *be the result of applying one step of SGD to* $\mathbf{w}^t$ *using the data point* $(\mathbf{x}_u, \mathbf{p}_{tar}(\mathbf{x}_u))$ *with learning rate* η*. Then the change in network predictions for a particular sample* $\mathbf{x}_o$ *is*

$$\mathbf{q}^{t+1}(\mathbf{x}_o) - \mathbf{q}^t(\mathbf{x}_o) = \eta\, \mathcal{A}^t(\mathbf{x}_o)\, \mathcal{K}^t(\mathbf{x}_o, \mathbf{x}_u)\, \left(\mathbf{p}_{tar}(\mathbf{x}_u) - \mathbf{q}^t(\mathbf{x}_u)\right) + \mathcal{O}(\eta^2 \|\nabla_{\mathbf{w}} \mathbf{z}(\mathbf{x}_u)\|_{\mathrm{op}}^2),$$

where $\mathcal{A}^t(\mathbf{x}_o) = \nabla_{\mathbf{z}} \mathbf{q}^t(\mathbf{x}_o)$ *and* $\mathcal{K}^t(\mathbf{x}_o, \mathbf{x}_u) = (\nabla_{\mathbf{w}} \mathbf{z}(\mathbf{x}_o)|_{\mathbf{w}^t})\, (\nabla_{\mathbf{w}} \mathbf{z}(\mathbf{x}_u)|_{\mathbf{w}^t})^\mathsf{T}$ *are* $K \times K$ *matrices.*

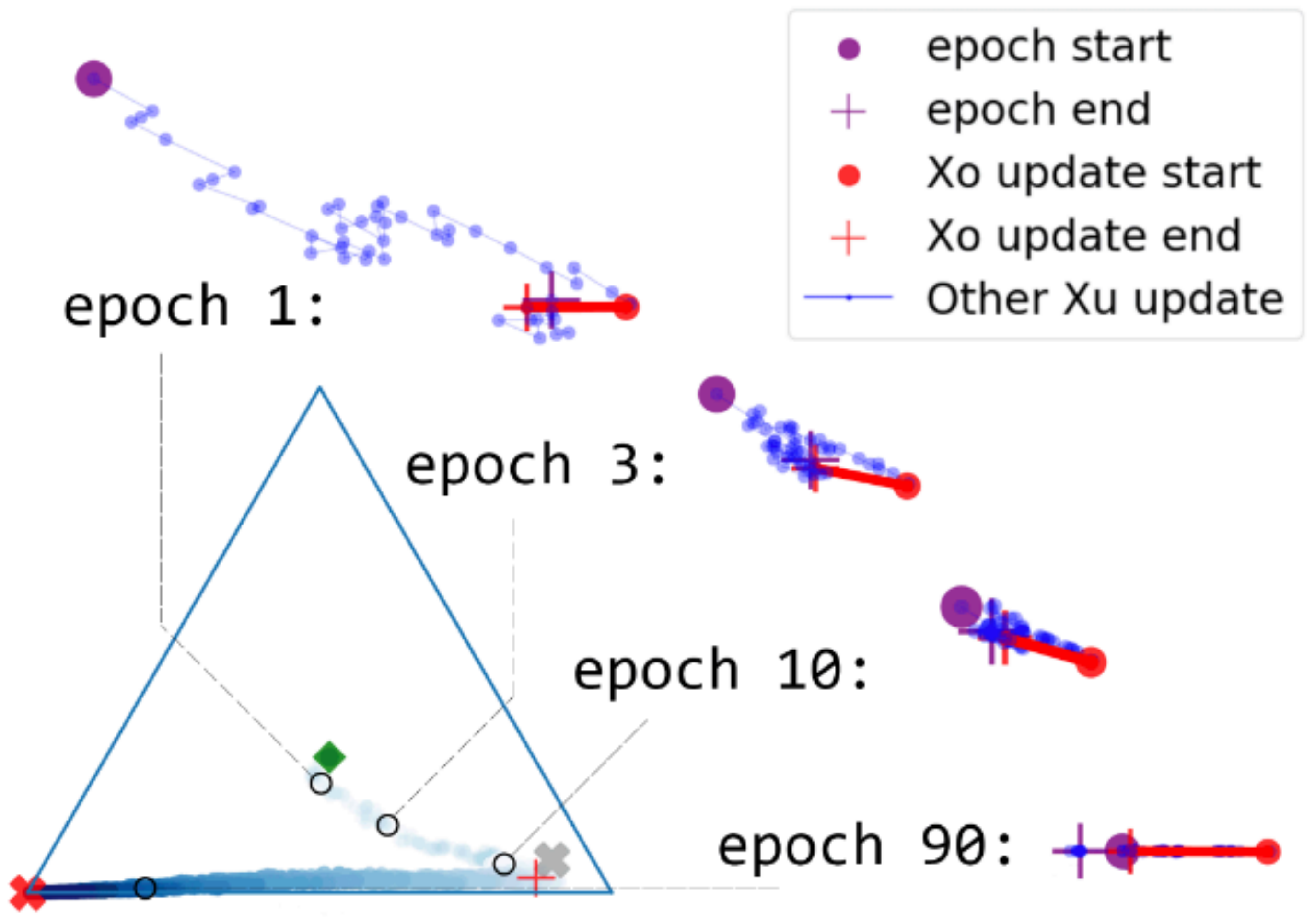


Figure 4: Updates of $\mathbf{q}(\mathbf{x}_o)$ over training.

Recap

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:

$$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right)\left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$

Recap

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:
$$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$
  - and so as $t \to \infty$, (S)GD on the network converges to kernel regression
$$\hat{f}(\mathbf{x}) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$

Recap

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:

$$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$

  - and so as $t \to \infty$, (S)GD on the network converges to kernel regression

$$\hat{f}(\mathbf{x}) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$

    - predictions on training set: $\mathbf{K}_0 \mathbf{K}_0^{-1}(\mathbf{y} - \mathbf{f}_0) + \mathbf{f}_0 = \mathbf{y} - \mathbf{f}_0 + \mathbf{f}_0 = \mathbf{y}$

Recap

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:
  $$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0} \right) \left(\mathbf{y} - \mathbf{f}_0 \right) + f_0(\mathbf{x})$$
  - and so as $t \to \infty$, (S)GD on the network converges to kernel regression
  $$\hat{f}(\mathbf{x}) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{y} - \mathbf{f}_0 \right) + f_0(\mathbf{x})$$
    - predictions on training set: $\mathbf{K}_0 \mathbf{K}_0^{-1}(\mathbf{y} - \mathbf{f}_0) + \mathbf{f}_0 = \mathbf{y} - \mathbf{f}_0 + \mathbf{f}_0 = \mathbf{y}$
- This **can't explain** all of real deep learning

Recap

- With essentially any architecture, using square loss, scalar outputs:
  - in the limit as the network becomes wider,
  - for appropriate Gaussian-distributed $\mathbf{w}$,
  - the NTK at initialization, $k_{\mathbf{w}_0}$, converges to its mean $\mathbf{E}_{\mathbf{w}} k_{\mathbf{w}}$,
  - **and** during training, $f(\mathbf{x}; \mathbf{w}_t)$ stays close to the linearized training result:
$$f^{lin}(\mathbf{x}; \mathbf{w}_t) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{I} - e^{-\frac{\eta t}{N}\mathbf{K}_0}\right) \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$
  - and so as $t \to \infty$, (S)GD on the network converges to kernel regression
$$\hat{f}(\mathbf{x}) = \mathbf{k}_0(\mathbf{x})\, \mathbf{K}_0^{-1} \left(\mathbf{y} - \mathbf{f}_0\right) + f_0(\mathbf{x})$$
    - predictions on training set: $\mathbf{K}_0 \mathbf{K}_0^{-1}(\mathbf{y} - \mathbf{f}_0) + \mathbf{f}_0 = \mathbf{y} - \mathbf{f}_0 + \mathbf{f}_0 = \mathbf{y}$
- This **can't explain** all of real deep learning
- But it's a useful tool, especially **local** approximations with **empirical NTK**