The Principles of Deep Learning Theory

Dan Roberts

MIT & Salesforce

January 13, 2022

Based on *The Principles of Deep Learning Theory* w/ Yaida and Hanin, 2106.10165, to be published by Cambridge University Press in 2022.

Goals

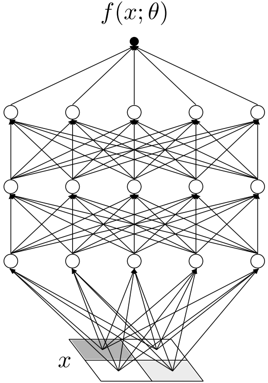
The goal of this talk is to theoretically analyze *deep* neural networks of *finite width*. In particular, we'll

(i) explain at a high level our approach, and

(ii) analyze a simple model of representation learning in nonlinear models.

Neural Networks

A **neural network** is a recipe for computing a function built out of many computational units called **neurons**:



Neurons are then organized in parallel into **layers**, and *deep* neural networks are those composed of multiple layers in sequence.

Neural Networks Abstracted

For the moment, let's ignore the detailed structure and focus on a general parameterized function,

$$f(x; \theta),$$

where x is the **input** to the function and θ is a vector of a large number of **parameters** controlling the shape of the function.

The Theoretical Minimum

Our goal is to analyze the *trained* network function:

$$f(x;\theta^{\star})\,.$$

The Theoretical Minimum

Our goal is to analyze the *trained* network function:

$$f(x;\theta^\star)\,.$$

One way to see the kinds of technical problems that we'll encounter in pursuit of this goal is to *Taylor expand* our trained network function $f(x;\theta^\star)$ around the initialized value of the parameters θ

$$f(x;\theta^\star) = f(x;\theta) + (\theta^\star-\theta)\,\frac{df}{d\theta} + \frac{1}{2}\,(\theta^\star-\theta)^2\,\frac{d^2 f}{d\theta^2} + \dots\,,$$

where $f(x;\theta)$ and its derivatives on the right-hand side are all evaluated at initialized value of the parameters.

The Theoretical Minimum: Problem 1

In general, the Taylor series contains an infinite number of terms

$$f\,, \quad \frac{df}{d\theta}\,, \quad \frac{d^2 f}{d\theta^2}\,, \quad \frac{d^3 f}{d\theta^3}\,, \quad \frac{d^4 f}{d\theta^4}\,, \quad \ldots\,,$$

and in principle we need to compute them all.

The Theoretical Minimum: Problem 2

Since the parameters θ are randomly sampled from $p(\theta)$, each time we initialize our network we get a different function $f(x;\theta)$, and we need to determine the mapping:

$$p(\theta) \to p\left(f, \frac{df}{d\theta}, \frac{d^2 f}{d\theta^2}, \dots\right) .$$

This means that each term f, $df/d\theta$, $d^2 f/d\theta^2$, ... , in the Taylor expansion is really a *random function* of the input x, and this joint distribution will have intricate statistical dependencies.

The Theoretical Minimum: Problem 3

The learned value of the parameters, $\theta^\star$, is the result of a complicated training process. In general, $\theta^\star$ is not unique and can depend on *everything*:

$$\theta^\star \equiv [\theta^\star]\left(\theta,\, f,\, \frac{df}{d\theta},\, \frac{d^2 f}{d\theta^2},\, \ldots;\, \text{learning algorithm};\, \text{training data}\right).$$

Determining an *analytical* expression for $\theta^\star$ must take "*everything*" into account.

Goal, restated

If we could solve all three of these problems, then we'd have a *distribution* over trained network functions

$$p(f^\star) \equiv p\Big(f(x;\theta^\star)\Big|\,\text{learning algorithm; training data}\Big)\,,$$

now conditioned in a simple way on the learning algorithm and the data we used for training.

Goal, restated

If we could solve all three of these problems, then we'd have a *distribution* over trained network functions

$$p(f^\star) \equiv p\Big(f(x;\theta^\star)\Big|\,\text{learning algorithm; training data}\Big)\,,$$

now conditioned in a simple way on the learning algorithm and the data we used for training.

- A framework for analyzing $p(f^\star)$ would let us *understand* AI systems and then let us use that knowledge to *improve* them.

Goal, restated

If we could solve all three of these problems, then we'd have a *distribution* over trained network functions

$$p(f^\star) \equiv p\Big(f(x;\theta^\star)\Big|\,\text{learning algorithm; training data}\Big)\,,$$

now conditioned in a simple way on the learning algorithm and the data we used for training.

- A framework for analyzing $p(f^\star)$ would let us *understand* AI systems and then let us use that knowledge to *improve* them.

The development of a method for the analytical computation of $p(f^\star)$ *should be a main goal of a theory of deep learning.*

Fine, Structure

Solving our three problems for general $f(x;\theta)$ is not tractable, so we need to use the particular **structure** of neural-network function.

Fine, Structure

Solving our three problems for general $f(x;\theta)$ is not tractable, so we need to use the particular **structure** of neural-network function.

- A starting point is the infinite-width limit

$$\lim_{n\to\infty} p(f^{\star})\,,$$

which gives an expression for the fully-trained distribution, in terms of a **Gaussian distribution** with a nonzero mean.

[Neal, Lee/Bahri/..., Matthews/..., Jacot/..., ...]

Fine, Structure

Solving our three problems for general $f(x;\theta)$ is not tractable, so we need to use the particular **structure** of neural-network function.

- A starting point is the infinite-width limit

$$\lim_{n\to\infty} p(f^\star)\,,$$

  which gives an expression for the fully-trained distribution, in terms of a **Gaussian distribution** with a nonzero mean.

  [Neal, Lee/Bahri/..., Matthews/..., Jacot/..., ...]

- We can find an *effective* description using **perturbation theory**, expanding in the inverse layer width, $\epsilon \equiv 1/n$:

$$p(f^\star) \equiv p^{\{0\}}(f^\star) + \frac{p^{\{1\}}(f^\star)}{n} + O\left(\frac{1}{n^2}\right)\,.$$

  (The details are in *The Principles of Deep Learning Theory*.)

Statistics vs. Dynamics

Stepping back, Problems 1 and 2 are about *initialization* ***statistics***:

$$p(\theta) \to p\left(f, \frac{df}{d\theta}, \frac{d^2 f}{d\theta^2}, \ldots\right) .$$

- Understanding this *ensemble* is essential for understanding *generalization* given different hyperparameter choices.

Statistics vs. Dynamics

Stepping back, Problems 1 and 2 are about *initialization* ***statistics***:

$$p(\theta) \to p\left(f, \frac{df}{d\theta}, \frac{d^2 f}{d\theta^2}, \ldots\right) .$$

- Understanding this *ensemble* is essential for understanding *generalization* given different hyperparameter choices.

Problem 3 is about the *training* ***dynamics***:

$$\theta^\star \equiv [\theta^\star]\left(\theta,\ f,\ \frac{df}{d\theta},\ \frac{d^2 f}{d\theta^2}, \ldots; \text{learning algorithm; training data}\right) .$$

- For now, we will try understand the *algorithm dependence* and *data dependence* of solutions for a very general class of machine learning models.

Machine Learning Models

$$f(x; \theta)$$

Machine Learning Models

$$f(x;\theta) \equiv z(x;\theta)$$

Machine Learning Models

$$f(x;\theta) \equiv z_i(x;\theta)$$

- $i = 1, \ldots, n_{\text{out}}$ is a **vectorial index**

Machine Learning Models

$$f(x;\theta) \equiv z_i(x_\delta;\theta)$$

- $i = 1, \ldots, n_{\text{out}}$ is a **vectorial index**
- $\delta \in \mathcal{D}$ is a **sample index**

Machine Learning Models

$$f(x;\theta) \equiv z_{i;\delta}(\theta)$$

- $i = 1, \dots, n_{\text{out}}$ is a **vectorial index**
- $\delta \in \mathcal{D}$ is a **sample index**

A Familiar Example

The simplest machine learning model is a **linear model**:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta}\,.$$

A Familiar Example

The simplest machine learning model is a **linear model**:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta}\,.$$

- Linear in both the parameters $\theta = \{b_i, W_{ij}\}$ and the input x_j.

A Familiar Example

The simplest machine learning model is a **linear model**:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta}\,.$$

- Linear in both the parameters $\theta = \{b_i, W_{ij}\}$ and the input x_j.
- The *linear* in *linear model* takes its name from the dependence on the parameters θ and not the input x.

A Familiar Example

The simplest machine learning model is a **linear model**:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta}\,.$$

- Linear in both the parameters $\theta = \{b_i, W_{ij}\}$ and the input x_j.
- The *linear* in *linear model* takes its name from the dependence on the parameters θ and not the input x.
- The linearity in x means this model can only approximate functions that are linear transformations of the input.

A Familiar Example

The simplest machine learning model is a **linear model**:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta}\,.$$

- Linear in both the parameters $\theta = \{b_i, W_{ij}\}$ and the input x_j.
- The *linear* in *linear model* takes its name from the dependence on the parameters θ and not the input x.
- The linearity in x means this model can only approximate functions that are linear transformations of the input.
- By another name: a one-layer (zero-hidden layer) network.

(Generalized) Linear Models

Instead, we might design a fixed basis of **feature functions** $\phi_j(x)$ that are meant to *fit* more complicated functions:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

(Generalized) Linear Models

Instead, we might design a fixed basis of **feature functions** $\phi_j(x)$ that are meant to *fit* more complicated functions:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_f} W_{ij}\,\phi_j(x_\delta)$$

- In this context, much of the complicated modeling work goes into the construction of these feature functions $\phi_j(x)$.

(Generalized) Linear Models

Instead, we might design a fixed basis of **feature functions** $\phi_j(x)$ that are meant to *fit* more complicated functions:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

- In this context, much of the complicated modeling work goes into the construction of these feature functions $\phi_j(x)$.
- We can still think of this model as a one-layer neural network, but now we pre-process x with the function $\phi_j(x)$.

(Generalized) Linear Models

Instead, we might design a fixed basis of **feature functions** $\phi_j(x)$ that are meant to *fit* more complicated functions:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

- Here, we've subsumed the bias vector into the weight matrix by setting $\phi_0(x) \equiv 1$ and $W_{i0} \equiv b_i$.

(Generalized) Linear Models

Instead, we might design a fixed basis of **feature functions** $\phi_j(x)$ that are meant to *fit* more complicated functions:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

- Here, we've subsumed the bias vector into the weight matrix by setting $\phi_0(x) \equiv 1$ and $W_{i0} \equiv b_i$.
- (e.g. for a 1-dimensional function we might pick a basis $\phi_j(x) = \{1, x, x^2, x^3\}$ and fit cubic curves.)

(Generalized) Linear Models

Instead, we might design a fixed basis of **feature functions** $\phi_j(x)$ that are meant to *fit* more complicated functions:

$$z_i(\theta) = W_{i0} + W_{i1}x + W_{i2}x^2 + W_{i3}x^3$$

- Here, we've subsumed the bias vector into the weight matrix by setting $\phi_0(x) \equiv 1$ and $W_{i0} \equiv b_i$.
- (e.g. for a 1-dimensional function we might pick a basis $\phi_j(x) = \{1, x, x^2, x^3\}$ and fit cubic curves.)

Linear Regression

Supervised learning with a *linear model* is **linear regression**

$$\mathcal{L}_{\mathcal{A}}(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij} \phi_j(x_{\tilde{\alpha}}) \right]^2 ,$$

where $y_i \equiv f_i(x)$ is an observed true output or *label*.

Linear Regression

Supervised learning with a *linear model* is **linear regression**

$$\mathcal{L}_{\mathcal{A}}(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij} \phi_j(x_{\tilde{\alpha}}) \right]^2 ,$$

where $y_i \equiv f_i(x)$ is an observed true output or *label*.

- We could solve by **direct optimization**:

$$0 = \left. \frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}} \right|_{W=W^\star} .$$

Linear Regression

Supervised learning with a *linear model* is **linear regression**

$$\mathcal{L}_{\mathcal{A}}(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij} \phi_j(x_{\tilde{\alpha}}) \right]^2 ,$$

where $y_i \equiv f_i(x)$ is an observed true output or *label*.

- We could solve by **direct optimization**:

$$0 = \left. \frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}} \right|_{W = W^\star} .$$

- We could solve by **gradient descent**:

$$W_{ij}(t+1) = W_{ij}(t) - \eta \left. \frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}} \right|_{W_{ij} = W_{ij}(t)} .$$

The Kernel

Let us introduce a new $N_{\mathcal{D}} \times N_{\mathcal{D}}$-dimensional symmetric matrix:

$$k_{\delta_1 \delta_2} \equiv k(x_{\delta_1}, x_{\delta_2}) \equiv \sum_{j=0}^{n_f} \phi_j(x_{\delta_1})\, \phi_j(x_{\delta_2}) \ .$$

As an inner product of features, the **kernel** $k_{\delta_1 \delta_2}$ is a measure of similarity between two inputs $x_{i;\delta_1}$ and $x_{i;\delta_2}$ in *feature space*.

We'll also denote an $N_{\mathcal{A}}$-by-$N_{\mathcal{A}}$-dimensional submatrix of the kernel evaluated on the training set as $\widetilde{k}_{\tilde{\alpha}_1 \tilde{\alpha}_2}$ with a tilde. This lets us write its **inverse** as $\widetilde{k}^{\tilde{\alpha}_1 \tilde{\alpha}_2}$, which satisfies

$$\sum_{\tilde{\alpha}_2 \in \mathcal{A}} \widetilde{k}^{\tilde{\alpha}_1 \tilde{\alpha}_2} \widetilde{k}_{\tilde{\alpha}_2 \tilde{\alpha}_3} = \delta^{\tilde{\alpha}_1}_{\ \tilde{\alpha}_3} \, .$$

Linear Models and Kernel Methods

Two forms of a solution for a **linear model**:

- *parameter space – linear regression*

$$z_i(x_{\dot\beta};\theta^\star) = \sum_{j=0}^{n_f} W_{ij}^\star \phi_j(x_{\dot\beta})$$

- *sample space – kernel methods*

$$z_i(x_{\dot\beta};\theta^\star) = \sum_{\tilde\alpha_1,\tilde\alpha_2\in\mathcal{A}} k_{\dot\beta\tilde\alpha_1}\widetilde{k}^{\tilde\alpha_1\tilde\alpha_2} y_{i;\tilde\alpha_2}\,.$$

Linear Models and Kernel Methods

Two forms of a solution for a **linear model**:

- *parameter space – linear regression*

$$z_i(x_{\dot{\beta}}; \theta^\star) = \sum_{j=0}^{n_f} W_{ij}^\star \phi_j(x_{\dot{\beta}})$$

- *sample space – kernel methods*

$$z_i(x_{\dot{\beta}}; \theta^\star) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in \mathcal{A}} k_{\dot{\beta}\tilde{\alpha}_1} \widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} \,.$$

Features of this model, expressed as $\phi_j(x)$ or $k_{\delta_1\delta_2}$, are *fixed*.

Frameworks: Linear Models vs. Deep Learning

Linear Regression goes back to Legendre and Gauss.

Frameworks: Linear Models vs. Deep Learning

Linear Regression goes back to Legendre and Gauss.

- "Three Problems" are tractable and can analyze completely.

Frameworks: Linear Models vs. Deep Learning

Linear Regression goes back to Legendre and Gauss.

- "Three Problems" are tractable and can analyze completely.
- Just "curve fitting" so naively unlikely to be useful for AI.

Frameworks: Linear Models vs. Deep Learning

Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear *both* in x and θ.

Frameworks: Linear Models vs. Deep Learning

Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear *both* in x and θ.

(i) The generalization of the $\phi(x)$ are inherited from the network and are *random* at the beginning of training.

Frameworks: Linear Models vs. Deep Learning

Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear *both* in x and θ.

(i) The generalization of the $\phi(x)$ are inherited from the network and are *random* at the beginning of training.

- Practitioners can design a network to have certain nice properties – like including convolutions for translation-invariant data – rather than having to pick a basis of functions.

Frameworks: Linear Models vs. Deep Learning

Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear *both* in x and θ.

(i) The generalization of the $\phi(x)$ are inherited from the network and are *random* at the beginning of training.

- Practitioners can design a network to have certain nice properties – like including convolutions for translation-invariant data – rather than having to pick a basis of functions.
- Understanding the particular basis requires a calculation.

Frameworks: Linear Models vs. Deep Learning

Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear *both* in x and θ.

(ii) The effective features *evolve* over the course of training:

$$\phi(x) \to \phi(x; \theta^\star) \, .$$

Frameworks: Linear Models vs. Deep Learning

Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear *both* in x and θ.

(ii) The effective features *evolve* over the course of training:

$$\phi(x) \to \phi(x; \theta^\star) .$$

- No longer just fitting a curve with a fixed basis!

Frameworks: Linear Models vs. Deep Learning

Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear *both* in x and θ.

(ii) The effective features *evolve* over the course of training:

$$\phi(x) \to \phi(x; \theta^\star) \,.$$

- No longer just fitting a curve with a fixed basis!
- Such **feature learning** is only a property of *nonlinear* models.

Nonlinear Models

To go beyond the linear paradigm, let's slightly *deform* it to get a **nonlinear model**, specifically a **quadratic model**:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta) + \frac{\epsilon}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1} W_{ij_2}\psi_{j_1 j_2}(x_\delta)$$

Nonlinear Models

To go beyond the linear paradigm, let's slightly *deform* it to get a **nonlinear model**, specifically a **quadratic model**:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta) + \frac{\epsilon}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1}W_{ij_2}\psi_{j_1 j_2}(x_\delta)$$

- It's nonlinear because it's quadratic in the weights: $W_{ij_1}W_{ij_2}$.

Nonlinear Models

To go beyond the linear paradigm, let's slightly *deform* it to get a **nonlinear model**, specifically a **quadratic model**:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta) + \frac{\epsilon}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1}W_{ij_2}\psi_{j_1j_2}(x_\delta)$$

- It's nonlinear because it's quadratic in the weights: $W_{ij_1}W_{ij_2}$.
- $\epsilon \ll 1$ is small parameter that controls the size of the deformation.

Nonlinear Models

To go beyond the linear paradigm, let's slightly *deform* it to get a **nonlinear model**, specifically a **quadratic model**:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta) + \frac{\epsilon}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1}W_{ij_2}\psi_{j_1 j_2}(x_\delta)$$

- It's nonlinear because it's quadratic in the weights: $W_{ij_1}W_{ij_2}$.
- $\epsilon \ll 1$ is small parameter that controls the size of the deformation.
- We've introduced $(n_f+1)(n_f+2)/2$ **meta feature functions**, $\psi_{j_1 j_2}(x)$, with *two* feature indices.

Quadratic Models

To familiarize ourselves with this model, let's make a small change in the model parameters $W_{ij} \to W_{ij} + dW_{ij}$:

$$z_i(x_\delta; \theta + d\theta) = z_i(x_\delta; \theta) + \sum_{j=0}^{n_f} dW_{ij} \left[\phi_j(x_\delta) + \epsilon \sum_{j_1=0}^{n_f} W_{ij_1} \psi_{j_1 j}(x_\delta) \right] + \frac{\epsilon}{2} \sum_{j_1, j_2=0}^{n_f} dW_{ij_1} dW_{ij_2} \psi_{j_1 j_2}(x_\delta).$$

Quadratic Models

To familiarize ourselves with this model, let's make a small change in the model parameters $W_{ij} \to W_{ij} + dW_{ij}$:

$$z_i(x_\delta; \theta + d\theta) = z_i(x_\delta; \theta) + \sum_{j=0}^{n_f} dW_{ij} \left[\phi_j(x_\delta) + \epsilon \sum_{j_1=0}^{n_f} W_{ij_1} \psi_{j_1 j}(x_\delta) \right] + \frac{\epsilon}{2} \sum_{j_1, j_2=0}^{n_f} dW_{ij_1} dW_{ij_2} \psi_{j_1 j_2}(x_\delta).$$

Let us make a shorthand for the quantity in the square bracket,

$$\phi^{\mathrm{E}}_{ij}(x_\delta; \theta) \equiv \frac{dz_i(x_\delta; \theta)}{dW_{ij}} = \phi_j(x_\delta) + \epsilon \sum_{k=0}^{n_f} W_{ik} \psi_{kj}(x_\delta)\,,$$

which is an **effective feature function**.

Effective Feature Learning

The quadratic model $z_i(x_\delta; \theta)$ behaves *effectively* as if it has a parameter-dependent feature function, $\phi^{\mathrm{E}}_{ij}(x_\delta; \theta)$.

Effective Feature Learning

The quadratic model $z_i(x_\delta;\theta)$ behaves *effectively* as if it has a parameter-dependent feature function, $\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)$.

- The $\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)$ *learns* with update dW_{ik}:

$$\phi^{\mathrm{E}}_{ij}(x_\delta;\theta+d\theta) = \phi^{\mathrm{E}}_{ij}(x_\delta;\theta) + \epsilon\sum_{k=0}^{n_f} dW_{ik}\,\psi_{kj}(x_\delta)\,.$$

Effective Feature Learning

The quadratic model $z_i(x_\delta;\theta)$ behaves *effectively* as if it has a parameter-dependent feature function, $\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)$.

- The $\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)$ *learns* with update dW_{ik}:

$$\phi^{\mathrm{E}}_{ij}(x_\delta;\theta+d\theta) = \phi^{\mathrm{E}}_{ij}(x_\delta;\theta) + \epsilon \sum_{k=0}^{n_f} dW_{ik}\, \psi_{kj}(x_\delta)\,.$$

- For comparison, for the linear model we'd have:

$$z_i(x_\delta;\theta+d\theta) = z_i(x_\delta;\theta) + \sum_{j=0}^{n_f} dW_{ij}\, \phi_j(x_\delta)$$

Effective Feature Learning

The quadratic model $z_i(x_\delta;\theta)$ behaves *effectively* as if it has a parameter-dependent feature function, $\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)$.

- The $\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)$ *learns* with update dW_{ik}:

$$\phi^{\mathrm{E}}_{ij}(x_\delta;\theta+d\theta)=\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)+\epsilon\sum_{k=0}^{n_f} dW_{ik}\,\psi_{kj}(x_\delta)\,.$$

- For comparison, for the linear model we'd have:

$$z_i(x_\delta;\theta+d\theta)=z_i(x_\delta;\theta)+\sum_{j=0}^{n_f} dW_{ij}\,\phi_j(x_\delta)$$

Thus quadratic model has a *hierarchical structure*, where the features evolve as if they are described by a linear model and the model's output evolves in a more complicated nonlinear way.

Quadratic Regression

Supervised learning a quadratic model doesn't have a particular name, but if it did, we'd all probably agree that its name should be **quadratic regression**:

$$\mathcal{L}_{\mathcal{A}}(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij}\, \phi_j(x_{\tilde{\alpha}}) - \frac{\epsilon}{2} \sum_{j_1, j_2 = 0}^{n_f} W_{ij_1} W_{ij_2} \psi_{j_1 j_2}(x_{\tilde{\alpha}}) \right]^2 .$$

Quadratic Regression

Supervised learning a quadratic model doesn't have a particular name, but if it did, we'd all probably agree that its name should be **quadratic regression**:

$$\mathcal{L}_{\mathcal{A}}(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij}\, \phi_j(x_{\tilde{\alpha}}) - \frac{\epsilon}{2} \sum_{j_1, j_2 = 0}^{n_f} W_{ij_1} W_{ij_2} \psi_{j_1 j_2}(x_{\tilde{\alpha}}) \right]^2 .$$

The loss is now *quartic* in the parameters, and in general

$$0 = \left. \frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}} \right|_{W = W^\star} ,$$

doesn't give analytical solutions or a tractable practical method.

Quadratic Regression

Supervised learning a quadratic model doesn't have a particular name, but if it did, we'd all probably agree that its name should be **quadratic regression**:

$$\mathcal{L}_{\mathcal{A}}(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij}\, \phi_j(x_{\tilde{\alpha}}) - \frac{\epsilon}{2} \sum_{j_1, j_2 = 0}^{n_f} W_{ij_1} W_{ij_2} \psi_{j_1 j_2}(x_{\tilde{\alpha}}) \right]^2 .$$

The loss is now *quartic* in the parameters, but we can optimize with *gradient descent*:

$$W_{ij}(t+1) = W_{ij}(t) - \eta \frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}} \bigg|_{W_{ij} = W_{ij}(t)} .$$

This will find a minimum in practice.

Quadratic Model Gradient Descent Dynamics

The weights will update as

$$
\begin{aligned}
W_{ij}(t+1) &= W_{ij}(t) - \eta \left.\frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}}\right|_{W_{ij}=W_{ij}(t)} \\
&= W_{ij}(t) - \eta \sum_{\tilde{\alpha}} \phi^{\mathsf{E}}_{ij;\tilde{\alpha}}(t)\,\left(z_{i;\tilde{\alpha}}(t) - y_{i;\tilde{\alpha}}\right).
\end{aligned}
$$

While the model and effective features update as

$$
\begin{aligned}
z_{i;\delta}(t+1) =& z_{i;\delta}(t) + \sum_{j} dW_{ij}(t)\,\phi^{\mathsf{E}}_{ij;\delta}(t) \\
&+ \frac{\epsilon}{2} \sum_{j_1,j_2} dW_{ij_1}(t)\,dW_{ij_2}(t)\,\psi_{j_1 j_2}(x_\delta), \\
\phi^{\mathsf{E}}_{ij;\delta}(t+1) =& \phi^{\mathsf{E}}_{ij;\delta}(t) + \epsilon \sum_{k=0}^{n_f} dW_{ik}(t)\,\psi_{kj}(x_\delta).
\end{aligned}
$$

Aside: Effective Kernel

To better understand this from the dual sample-space picture, let's analogously define an **effective kernel**

$$k^{\mathrm{E}}_{ii;\delta_1\delta_2}(\theta) \equiv \sum_{j=0}^{n_f} \phi^{\mathrm{E}}_{ij}(x_{\delta_1};\theta)\,\phi^{\mathrm{E}}_{ij}(x_{\delta_2};\theta)\,,$$

which measures a parameter-dependent similarity between two inputs x_{δ_1} and x_{δ_2} using our *effective features* $\phi^{\mathrm{E}}_{ij}(x_\delta;\theta)$.

Aside 2: Meta Kernel

Another important object worth defining we call the **meta kernel**:

$$\mu_{\delta_0\delta_1\delta_2} \equiv \sum_{j_1,j_2=0}^{n_f} \epsilon\, \psi_{j_1 j_2}(x_{\delta_0})\, \phi_{j_1}(x_{\delta_1})\, \phi_{j_2}(x_{\delta_2}).$$

Aside 2: Meta Kernel

Another important object worth defining we call the **meta kernel**:

$$\mu_{\delta_0\delta_1\delta_2} \equiv \sum_{j_1,j_2=0}^{n_f} \epsilon\, \psi_{j_1 j_2}(x_{\delta_0})\, \phi_{j_1}(x_{\delta_1})\, \phi_{j_2}(x_{\delta_2}).$$

- This is a *parameter-independent* tensor given entirely in terms of the fixed $\phi_j(x)$ and $\psi_{j_1 j_2}(x)$ that define the model.

Aside 2: Meta Kernel

Another important object worth defining we call the **meta kernel**:

$$\mu_{\delta_0\delta_1\delta_2} \equiv \sum_{j_1,j_2=0}^{n_f} \epsilon\, \psi_{j_1 j_2}(x_{\delta_0})\, \phi_{j_1}(x_{\delta_1})\, \phi_{j_2}(x_{\delta_2}).$$

- This is a *parameter-independent* tensor given entirely in terms of the fixed $\phi_j(x)$ and $\psi_{j_1 j_2}(x)$ that define the model.
- For a fixed input x_{δ_0}, $\mu_{\delta_0\delta_1\delta_2}$ computes a different feature-space inner product between the two inputs, x_{δ_1} & x_{δ_2}.

Aside 2: Meta Kernel

Another important object worth defining we call the **meta kernel**:

$$\mu_{\delta_0\delta_1\delta_2} \equiv \sum_{j_1,j_2=0}^{n_f} \epsilon\, \psi_{j_1 j_2}(x_{\delta_0})\, \phi_{j_1}(x_{\delta_1})\, \phi_{j_2}(x_{\delta_2}).$$

- This is a *parameter-independent* tensor given entirely in terms of the fixed $\phi_j(x)$ and $\psi_{j_1 j_2}(x)$ that define the model.
- For a fixed input x_{δ_0}, $\mu_{\delta_0\delta_1\delta_2}$ computes a different feature-space inner product between the two inputs, x_{δ_1} & x_{δ_2}.
- Due to the inclusion of ϵ into the definition of $\mu_{\delta_0\delta_1\delta_2}$, we should think of it as being parametrically small too.

Quadratic Model Gradient Descent Dynamics (Again)

The weights will update as

$$
\begin{aligned}
W_{ij}(t+1) &= W_{ij}(t) - \eta \left.\frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}}\right|_{W_{ij}=W_{ij}(t)} \\
&= W_{ij}(t) - \eta \sum_{\tilde{\alpha}} \phi^{\mathsf{E}}_{ij;\tilde{\alpha}}(t)\, \left(z_{i;\tilde{\alpha}}(t) - y_{i;\tilde{\alpha}}\right).
\end{aligned}
$$

While the model and effective features update as

$$
\begin{aligned}
z_{i;\delta}(t+1) =& z_{i;\delta}(t) + \sum_{j} dW_{ij}(t)\, \phi^{\mathsf{E}}_{ij;\delta}(t) \\
&+ \frac{\epsilon}{2} \sum_{j_1,j_2} dW_{ij_1}(t)\, dW_{ij_2}(t)\, \psi_{j_1 j_2}(x_\delta), \\
\phi^{\mathsf{E}}_{ij;\delta}(t+1) =& \phi^{\mathsf{E}}_{ij;\delta}(t) + \epsilon \sum_{k=0}^{n_f} dW_{ik}(t)\, \psi_{kj}(x_\delta).
\end{aligned}
$$

Quadratic Model Gradient Dynamics: Dual Sample Space

The *model predictions* will update as

$$
\begin{aligned}
& z_{i;\delta}(t+1) \\
=& z_{i;\delta}(t) - \eta \sum_{\tilde{\alpha}} k^{\mathrm{E}}_{ii;\delta\tilde{\alpha}}(t)\, \epsilon_{i;\tilde{\alpha}}(t) + \frac{\eta^2}{2} \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2} \mu_{\delta\tilde{\alpha}_1\tilde{\alpha}_2} \epsilon_{i;\tilde{\alpha}_1}(t)\, \epsilon_{i;\tilde{\alpha}_2}(t) + \ldots\,,
\end{aligned}
$$

while the *effective kernel* will update as

$$
k^{\mathrm{E}}_{ii;\delta_1\delta_2}(t+1) = k^{\mathrm{E}}_{ii;\delta_1\delta_2}(t) - \eta \sum_{\tilde{\alpha}} \left(\mu_{\delta_1\delta_2\tilde{\alpha}} + \mu_{\delta_2\delta_1\tilde{\alpha}}\right) \epsilon_{i;\tilde{\alpha}}(t) + \ldots\,,
$$

with the residual training error

$$
\epsilon_{i;\tilde{\alpha}}(t) \equiv z_{i;\tilde{\alpha}}(t) - y_{i;\tilde{\alpha}}\,.
$$

- These joint updates are coupled *difference equations*, and the first is *nonlinear* in the training error.

Solution

$$
\begin{aligned}
& z_{i;\dot\beta}(\infty) \\
= & \sum_{\tilde\alpha_1,\tilde\alpha_2\in\mathcal{A}} k_{\dot\beta\tilde\alpha_1}\widetilde{k}^{\tilde\alpha_1\tilde\alpha_2} y_{i;\tilde\alpha_2} \\
& + \sum_{\tilde\alpha_1,\ldots,\tilde\alpha_4\in\mathcal{A}} \left[\mu_{\tilde\alpha_1\dot\beta\tilde\alpha_2} - \sum_{\tilde\alpha_5,\tilde\alpha_6\in\mathcal{A}} k_{\dot\beta\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_5\tilde\alpha_6}\mu_{\tilde\alpha_1\tilde\alpha_6\tilde\alpha_2}\right] Z_{\mathrm{A}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} y_{i;\tilde\alpha_3}\, y_{i;\tilde\alpha_4} \\
& + \sum_{\tilde\alpha_1,\ldots,\tilde\alpha_4\in\mathcal{A}} \left[\mu_{\dot\beta\tilde\alpha_1\tilde\alpha_2} - \sum_{\tilde\alpha_5,\tilde\alpha_6\in\mathcal{A}} k_{\dot\beta\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_5\tilde\alpha_6}\mu_{\tilde\alpha_6\tilde\alpha_1\tilde\alpha_2}\right] Z_{\mathrm{B}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} y_{i;\tilde\alpha_3}\, y_{i;\tilde\alpha_4}
\end{aligned}
$$

where the **algorithm projectors** are given by

$$
\begin{aligned}
Z_{\mathrm{A}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} \equiv & \widetilde{k}^{\tilde\alpha_1\tilde\alpha_3}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_4} - \sum_{\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_5} X_{\mathrm{II}}^{\tilde\alpha_1\tilde\alpha_5\tilde\alpha_3\tilde\alpha_4}\,, \\
Z_{\mathrm{B}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} \equiv & \widetilde{k}^{\tilde\alpha_1\tilde\alpha_3}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_4} - \sum_{\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_5} X_{\mathrm{II}}^{\tilde\alpha_1\tilde\alpha_5\tilde\alpha_3\tilde\alpha_4} + \frac{\eta}{2} X_{\mathrm{II}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4}\,.
\end{aligned}
$$

Here, an **inverting tensor** is implicitly defined:

$$
\begin{aligned}
&\delta^{\tilde{\alpha}_1}_{\tilde{\alpha}_5}\delta^{\tilde{\alpha}_2}_{\tilde{\alpha}_6} \\
&= \sum_{\tilde{\alpha}_3,\tilde{\alpha}_4\in\mathcal{A}} X_{\mathrm{II}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}\frac{1}{\eta}\left[\delta_{\tilde{\alpha}_3\tilde{\alpha}_5}\delta_{\tilde{\alpha}_4\tilde{\alpha}_6} - (\delta_{\tilde{\alpha}_3\tilde{\alpha}_5} - \eta\widetilde{k}_{\tilde{\alpha}_3\tilde{\alpha}_5})(\delta_{\tilde{\alpha}_4\tilde{\alpha}_6} - \eta\widetilde{k}_{\tilde{\alpha}_4\tilde{\alpha}_6})\right] \\
&= \sum_{\tilde{\alpha}_3,\tilde{\alpha}_4\in\mathcal{A}} X_{\mathrm{II}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}\left(\widetilde{k}_{\tilde{\alpha}_3\tilde{\alpha}_5}\delta_{\tilde{\alpha}_4\tilde{\alpha}_6} + \delta_{\tilde{\alpha}_3\tilde{\alpha}_5}\widetilde{k}_{\tilde{\alpha}_4\tilde{\alpha}_6} - \eta\widetilde{k}_{\tilde{\alpha}_3\tilde{\alpha}_5}\widetilde{k}_{\tilde{\alpha}_4\tilde{\alpha}_6}\right) .
\end{aligned}
$$

Solution

$$
\begin{aligned}
& z_{i;\dot\beta}(\infty) \\
= & \sum_{\tilde\alpha_1,\tilde\alpha_2\in\mathcal{A}} k_{\dot\beta\tilde\alpha_1}\widetilde{k}^{\tilde\alpha_1\tilde\alpha_2} y_{i;\tilde\alpha_2} \\
& + \sum_{\tilde\alpha_1,\dots,\tilde\alpha_4\in\mathcal{A}} \left[\mu_{\tilde\alpha_1\dot\beta\tilde\alpha_2} - \sum_{\tilde\alpha_5,\tilde\alpha_6\in\mathcal{A}} k_{\dot\beta\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_5\tilde\alpha_6}\mu_{\tilde\alpha_1\tilde\alpha_6\tilde\alpha_2}\right] Z_{\mathrm{A}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} y_{i;\tilde\alpha_3}\, y_{i;\tilde\alpha_4} \\
& + \sum_{\tilde\alpha_1,\dots,\tilde\alpha_4\in\mathcal{A}} \left[\mu_{\dot\beta\tilde\alpha_1\tilde\alpha_2} - \sum_{\tilde\alpha_5,\tilde\alpha_6\in\mathcal{A}} k_{\dot\beta\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_5\tilde\alpha_6}\mu_{\tilde\alpha_6\tilde\alpha_1\tilde\alpha_2}\right] Z_{\mathrm{B}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} y_{i;\tilde\alpha_3}\, y_{i;\tilde\alpha_4}
\end{aligned}
$$

where the **algorithm projectors** are given by

$$
Z_{\mathrm{A}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} \equiv \widetilde{k}^{\tilde\alpha_1\tilde\alpha_3}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_4} - \sum_{\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_5} X_{\mathrm{II}}^{\tilde\alpha_1\tilde\alpha_5\tilde\alpha_3\tilde\alpha_4}\,,
$$

$$
Z_{\mathrm{B}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4} \equiv \widetilde{k}^{\tilde\alpha_1\tilde\alpha_3}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_4} - \sum_{\tilde\alpha_5}\widetilde{k}^{\tilde\alpha_2\tilde\alpha_5} X_{\mathrm{II}}^{\tilde\alpha_1\tilde\alpha_5\tilde\alpha_3\tilde\alpha_4} + \frac{\eta}{2} X_{\mathrm{II}}^{\tilde\alpha_1\tilde\alpha_2\tilde\alpha_3\tilde\alpha_4}\,.
$$

Nearly-Kernel Methods

When the prediction is computed in this way, we can think of it as a *nearly-kernel machine* or **nearly-kernel methods**.

Nearly-Kernel Methods

When the prediction is computed in this way, we can think of it as a *nearly-kernel machine* or **nearly-kernel methods**.

Unlike *kernel methods*, this depends on the *learning algorithm*.

Nearly-Kernel Methods

When the prediction is computed in this way, we can think of it as a *nearly-kernel machine* or **nearly-kernel methods**.

Unlike *kernel methods*, this depends on the *learning algorithm*.

- If we'd optimized by *direct optimization*, we'd have found:

$$Z_{\mathrm{A}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} = 0, \qquad Z_{\mathrm{B}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} = \frac{1}{2}\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_3}\widetilde{k}^{\tilde{\alpha}_2\tilde{\alpha}_4}.$$

Nearly-Kernel Methods

When the prediction is computed in this way, we can think of it as a *nearly-kernel machine* or **nearly-kernel methods**.

Unlike *kernel methods*, this depends on the *learning algorithm*.

- If we'd optimized by *direct optimization*, we'd have found:

$$Z_{\mathrm{A}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} = 0, \qquad Z_{\mathrm{B}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} = \frac{1}{2}\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_3}\widetilde{k}^{\tilde{\alpha}_2\tilde{\alpha}_4}.$$

- In the ODE limit, we get different predictions

$$Z_{\mathrm{A}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} = Z_{\mathrm{B}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} \equiv \widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_3}\widetilde{k}^{\tilde{\alpha}_2\tilde{\alpha}_4} - \sum_{\tilde{\alpha}_5}\widetilde{k}^{\tilde{\alpha}_2\tilde{\alpha}_5}X_{\mathrm{II}}^{\tilde{\alpha}_1\tilde{\alpha}_5\tilde{\alpha}_3\tilde{\alpha}_4},$$

$$\sum_{\tilde{\alpha}_3,\tilde{\alpha}_4\in\mathcal{A}} X_{\mathrm{II}}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}\left(\widetilde{k}_{\tilde{\alpha}_3\tilde{\alpha}_5}\delta_{\tilde{\alpha}_4\tilde{\alpha}_6} + \delta_{\tilde{\alpha}_3\tilde{\alpha}_5}\widetilde{k}_{\tilde{\alpha}_4\tilde{\alpha}_6}\right) = \delta^{\tilde{\alpha}_1}_{\tilde{\alpha}_5}\delta^{\tilde{\alpha}_2}_{\tilde{\alpha}_6},$$

Nearly-Kernel Methods

When the prediction is computed in this way, we can think of it as a *nearly-kernel machine* or **nearly-kernel methods**.

We again have two ways of thinking about the solution:

- we can use the optimal parameters to make predictions, or
- we can make *nearly-kernel predictions* in which the features, the meta features, and the model parameters do not appear.

Nearly-Kernel Methods

When the prediction is computed in this way, we can think of it as a *nearly-kernel machine* or **nearly-kernel methods**.

We again have two ways of thinking about the solution:

- we can use the optimal parameters to make predictions, or
- we can make *nearly-kernel predictions* in which the features, the meta features, and the model parameters do not appear.

Predictions are made by direct comparison with the training set:

- It has the kernel linear piece $\propto y_{i;\tilde{\alpha}_2}$, and
- it also has a new quadratic piece $\propto y_{i;\tilde{\alpha}_1} y_{i;\tilde{\alpha}_2}$.

Representation Learning

For simplicity, let's pick the **direct optimization** solution:

$$k^{\mathrm{E}}_{ii;\delta_1\delta_2}(\theta^\star) = k_{\delta_1\delta_2} + \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} \left(\mu_{\delta_1\delta_2\tilde{\alpha}_1} + \mu_{\delta_2\delta_1\tilde{\alpha}_1}\right)\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} + O\!\left(\epsilon^2\right) .$$

Then, we can define a ***trained kernel*** that averages between the *fixed* kernel and *dynamical* effective kernels:

$$k^{\sharp}_{ii;\delta_1\delta_2} \equiv \frac{1}{2}\left[k_{\delta_1\delta_2} + k^{\mathrm{E}}_{ii;\delta_1\delta_2}(\theta^\star)\right] .$$

Now the nearly-kernel prediction formula can be compressed,

$$z_i(x_{\dot{\beta}};\theta^\star) = \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} k^{\sharp}_{ii;\dot{\beta}\tilde{\alpha}_1} \widetilde{k^{\sharp}}_{ii}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} + O\!\left(\epsilon^2\right) ,$$

taking the form of a *kernel prediction*, but with the benefit of nontrivial feature evolution incorporated into the trained kernel.

Representation Learning as Regularization

The *direct optimization* solution in parameter space is

$$z_i(x_{\dot\beta};\theta^\star) = \sum_{j=0}^{n_f} W_{ij}^\star \phi_j(x_{\dot\beta}) + \frac{\epsilon}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1}^\star W_{ij_2}^\star \psi_{j_1 j_2}(x_{\dot\beta})$$

and the optimal parameters can decompose as

$$W_{ij}^\star \equiv W_{ij}^{\mathrm{F}} + W_{ij}^{\mathrm{I}}\,,$$

where W_{ij}^{F} are the optimal parameters from the linear model.

Representation Learning as Regularization

The *direct optimization* solution in parameter space is

$$z_i(x_{\dot\beta};\theta^\star) = \sum_{j=0}^{n_f} W^\star_{ij}\phi_j(x_{\dot\beta}) + \frac{\epsilon}{2}\sum_{j_1,j_2=0}^{n_f} W^\star_{ij_1}W^\star_{ij_2}\psi_{j_1j_2}(x_{\dot\beta})$$

and the optimal parameters can decompose as

$$W^\star_{ij} \equiv W^{\mathrm{F}}_{ij} + W^{\mathrm{I}}_{ij}\,,$$

where W^{F}_{ij} are the optimal parameters from the linear model.

- The $O(\epsilon)$ tunings W^{I}_{ij} ruin the **fine tuning** of the W^{F}_{ij}, as they are constrained by the $\psi_{kj}(x)$ defined before training.

Representation Learning as Regularization

The *direct optimization* solution in parameter space is

$$z_i(x_{\dot\beta};\theta^\star) = \sum_{j=0}^{n_f} W^\star_{ij}\phi_j(x_{\dot\beta}) + \frac{\epsilon}{2}\sum_{j_1,j_2=0}^{n_f} W^\star_{ij_1}W^\star_{ij_2}\psi_{j_1j_2}(x_{\dot\beta})$$

and the optimal parameters can decompose as

$$W^\star_{ij} \equiv W^{\mathrm{F}}_{ij} + W^{\mathrm{I}}_{ij}\,,$$

where W^{F}_{ij} are the optimal parameters from the linear model.

- The $O(\epsilon)$ tunings W^{I}_{ij} ruin the **fine tuning** of the W^{F}_{ij}, as they are constrained by the $\psi_{kj}(x)$ defined before training.
- Assuming these $\psi_{kj}(x)$ are *useful*, we might expect that the quadratic model will overfit less and generalize better.

Does feature learning help generalization?

Consider the generalization error

$$\mathcal{L}_{\mathcal{B}}(\epsilon) = \frac{1}{2} \sum_{\dot{\beta}\in\mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\dot{\beta}} - z_{i;\dot{\beta}}(\epsilon) \right]^2 ,$$

where $\dot{\beta} \in \mathcal{B}$ is a sample index in the test set, with

$$z_{i;\dot{\beta}}(\epsilon) = z^{\mathrm{F}}_{i;\dot{\beta}} + \epsilon\, z^{\mathrm{I}}_{i;\dot{\beta}} + O\left(\epsilon^2\right) .$$

Does feature learning help generalization?

Consider the generalization error

$$\mathcal{L}_{\mathcal{B}}(\epsilon) = \frac{1}{2} \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\dot\beta} - z_{i;\dot\beta}(\epsilon) \right]^2 ,$$

where $\dot\beta \in \mathcal{B}$ is a sample index in the test set, with

$$z_{i;\dot\beta}(\epsilon) = z^{\mathsf{F}}_{i;\dot\beta} + \epsilon\, z^{\mathsf{I}}_{i;\dot\beta} + O\left(\epsilon^2\right) .$$

Then, we can see if the quadratic deformation helps by computing

$$\frac{d\mathcal{L}_{\mathcal{B}}}{d\epsilon} = \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \frac{\partial \mathcal{L}_{\mathcal{B}}(\epsilon)}{\partial z_{i;\dot\beta}} \frac{dz_{i;\dot\beta}}{d\epsilon} < 0$$

Does feature learning help generalization?

Consider the generalization error

$$\mathcal{L}_{\mathcal{B}}(\epsilon) = \frac{1}{2} \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\dot\beta} - z_{i;\dot\beta}(\epsilon) \right]^2 ,$$

where $\dot\beta \in \mathcal{B}$ is a sample index in the test set, with

$$z_{i;\dot\beta}(\epsilon) = z^{\mathsf{F}}_{i;\dot\beta} + \epsilon\, z^{\mathsf{I}}_{i;\dot\beta} + O\left(\epsilon^2\right) .$$

Then, we can see if the quadratic deformation helps by computing

$$\frac{d\mathcal{L}_{\mathcal{B}}}{d\epsilon} = \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left(z_{i;\dot\beta}(\epsilon) - y_{i;\dot\beta} \right) \frac{dz_{i;\dot\beta}}{d\epsilon} < 0$$

Does feature learning help generalization?

Consider the generalization error

$$\mathcal{L}_{\mathcal{B}}(\epsilon) = \frac{1}{2} \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\dot\beta} - z_{i;\dot\beta}(\epsilon) \right]^2 ,$$

where $\dot\beta \in \mathcal{B}$ is a sample index in the test set, with

$$z_{i;\dot\beta}(\epsilon) = z^{\mathrm{F}}_{i;\dot\beta} + \epsilon\, z^{\mathrm{I}}_{i;\dot\beta} + O\left(\epsilon^2\right) .$$

Then, we can see if the quadratic deformation helps by computing

$$\frac{d\mathcal{L}_{\mathcal{B}}}{d\epsilon} = \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left(z_{i;\dot\beta}(\epsilon) - y_{i;\dot\beta} \right) \epsilon\, z^{\mathrm{I}}_{i;\dot\beta} + O\left(\epsilon^2\right) < 0$$

Does feature learning help generalization?

Consider the generalization error

$$\mathcal{L}_{\mathcal{B}}(\epsilon) = \frac{1}{2} \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\dot\beta} - z_{i;\dot\beta}(\epsilon) \right]^2 ,$$

where $\dot\beta \in \mathcal{B}$ is a sample index in the test set, with

$$z_{i;\dot\beta}(\epsilon) = z^{\mathrm{F}}_{i;\dot\beta} + \epsilon\, z^{\mathrm{I}}_{i;\dot\beta} + O\left(\epsilon^2\right) .$$

Then, we can see if the quadratic deformation helps by computing

$$\frac{d\mathcal{L}_{\mathcal{B}}}{d\epsilon} = \sum_{\dot\beta \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left(z^{\mathrm{F}}_{i;\dot\beta} - y_{i;\dot\beta} \right) \epsilon\, z^{\mathrm{I}}_{i;\dot\beta} + O\left(\epsilon^2\right) < 0$$

Does feature learning help generalization?

Consider the generalization error

$$\mathcal{L}_{\mathcal{B}}(\epsilon) = \frac{1}{2} \sum_{\dot{\beta} \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left[y_{i;\dot{\beta}} - z_{i;\dot{\beta}}(\epsilon) \right]^2 ,$$

where $\dot{\beta} \in \mathcal{B}$ is a sample index in the test set, with

$$z_{i;\dot{\beta}}(\epsilon) = z^{\mathrm{F}}_{i;\dot{\beta}} + \epsilon \, z^{\mathrm{I}}_{i;\dot{\beta}} + O\left(\epsilon^2\right) .$$

Then, we can see if the quadratic deformation helps by computing

$$\frac{d\mathcal{L}_{\mathcal{B}}}{d\epsilon} = \sum_{\dot{\beta} \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left(z^{\mathrm{F}}_{i;\dot{\beta}} - y_{i;\dot{\beta}} \right) \epsilon \, z^{\mathrm{I}}_{i;\dot{\beta}} + O\left(\epsilon^2\right) < 0$$

Depends on the initial training error and the nonlinear prediction.

How Much?

Need to evaluate our solution to order ϵ^2:

$$z_{i;\dot\beta}(\epsilon) = z^{\mathrm{F}}_{i;\dot\beta} + \epsilon\, z^{\mathrm{I}}_{i;\dot\beta} + \epsilon^2\, z^{\mathrm{II}}_{i;\dot\beta} + O\left(\epsilon^3\right) \, .$$

How Much?

Need to evaluate our solution to order ϵ^2:

$$z_{i;\dot\beta}(\epsilon) = z^{\mathsf{F}}_{i;\dot\beta} + \epsilon\, z^{\mathsf{I}}_{i;\dot\beta} + \epsilon^2\, z^{\mathsf{II}}_{i;\dot\beta} + O\left(\epsilon^3\right) \, .$$

Then, by calculating

$$0 = \frac{d\mathcal{L}_{\mathcal{B}}}{d\epsilon}\bigg|_{\epsilon\to\epsilon^\star} \, ,$$

we can optimize the amount of feature learning:

$$\epsilon^\star = \frac{-\sum_{\dot\beta\in\mathcal{B}}\sum_{i=1}^{n_{\text{out}}}\left(z^{\mathsf{F}}_{i;\dot\beta} - y_{i;\dot\beta}\right)\, z^{\mathsf{I}}_{i;\dot\beta}}{\sum_{\dot\beta\in\mathcal{B}}\sum_{i=1}^{n_{\text{out}}}\left[\left(z^{\mathsf{I}}_{i;\dot\beta}\right)^2 + \left(z^{\mathsf{F}}_{i;\dot\beta} - y_{i;\dot\beta}\right)\, z^{\mathsf{II}}_{i;\dot\beta}\right]}$$

How Much?

Need to evaluate our solution to order ϵ^2:

$$z_{i;\dot\beta}(\epsilon) = z^{\mathsf{F}}_{i;\dot\beta} + \epsilon\, z^{\mathsf{I}}_{i;\dot\beta} + \epsilon^2\, z^{\mathsf{II}}_{i;\dot\beta} + O\left(\epsilon^3\right) .$$

Then, by calculating

$$0 = \frac{d\mathcal{L}_{\mathcal{B}}}{d\epsilon}\bigg|_{\epsilon\to\epsilon^\star} ,$$

we can optimize the amount of feature learning:

$$\epsilon^\star = \frac{-\sum_{\dot\beta\in\mathcal{B}}\sum_{i=1}^{n_{\text{out}}}\left(z^{\mathsf{F}}_{i;\dot\beta} - y_{i;\dot\beta}\right)\, z^{\mathsf{I}}_{i;\dot\beta}}{\sum_{\dot\beta\in\mathcal{B}}\sum_{i=1}^{n_{\text{out}}}\left[\left(z^{\mathsf{I}}_{i;\dot\beta}\right)^2 + \left(z^{\mathsf{F}}_{i;\dot\beta} - y_{i;\dot\beta}\right)\, z^{\mathsf{II}}_{i;\dot\beta}\right]}$$

This means that for different datasets and tasks, this will have different levels of importance.

Quadratic Models vs. Deep Learning

- Quadratic models are *minimal models* of feature learning:

$$
z_i(x_\delta;\theta^\star) = \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} k^\sharp_{ii;\delta\tilde{\alpha}_1} \widetilde{k^\sharp}_{ii}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} + O\Big(\epsilon^2\Big) \ ,
$$

$$
k^\sharp_{ii;\delta_1\delta_2} \equiv \frac{1}{2}\Big[k_{\delta_1\delta_2} + k^{\mathsf{E}}_{ii;\delta_1\delta_2}(\theta^\star)\Big] \ .
$$

Quadratic Models vs. Deep Learning

- Quadratic models are *minimal models* of feature learning:

$$z_i(x_\delta;\theta^\star) = \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} k^\sharp_{ii;\delta\tilde{\alpha}_1} \widetilde{k^\sharp}_{ii}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} + O\left(\epsilon^2\right) ,$$

$$k^\sharp_{ii;\delta_1\delta_2} \equiv \frac{1}{2}\left[k_{\delta_1\delta_2} + k^{\mathrm{E}}_{ii;\delta_1\delta_2}(\theta^\star)\right] .$$

- MLPs at large-but-finite width are *cubic models*

$$z_i(x_\delta;\theta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta) + \frac{1}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1}W_{ij_2}\psi_{j_1j_2}(x_\delta)$$
$$+ \frac{1}{6}\sum_{j_1,j_2,j_3=0}^{n_f} W_{ij_1}W_{ij_2}W_{ij_3}\Psi_{j_1j_2j_3}(x_\delta)$$

Quadratic Models vs. Deep Learning

- Quadratic models are *minimal models* of feature learning:

$$z_i(x_\delta;\theta^\star) = \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} k^\sharp_{ii;\delta\tilde{\alpha}_1}\widetilde{k^\sharp}_{ii}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} + O\left(\epsilon^2\right) \,,$$

$$k^\sharp_{ii;\delta_1\delta_2} \equiv \frac{1}{2}\left[k_{\delta_1\delta_2} + k^{\mathrm{E}}_{ii;\delta_1\delta_2}(\theta^\star)\right] \,.$$

- MLPs at large-but-finite width are *cubic models*

$$z_i(x_\delta;\theta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta) + \frac{1}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1}W_{ij_2}\psi_{j_1j_2}(x_\delta)$$
$$+\frac{1}{6}\sum_{j_1,j_2,j_3=0}^{n_f} W_{ij_1}W_{ij_2}W_{ij_3}\Psi_{j_1j_2j_3}(x_\delta)$$

- The amount of representation learning is set by the depth-to-width ratio, $\epsilon \equiv \frac{L}{n}$, with the depth L and width n.

Quadratic Models vs. Deep Learning

- Quadratic models are *minimal models* of feature learning:

$$z_i(x_\delta;\theta^\star) = \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2\in\mathcal{A}} k^\sharp_{ii;\delta\tilde{\alpha}_1} \widetilde{k^\sharp}_{ii}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} + O\left(\epsilon^2\right) ,$$

$$k^\sharp_{ii;\delta_1\delta_2} \equiv \frac{1}{2}\left[k_{\delta_1\delta_2} + k^{\mathrm{E}}_{ii;\delta_1\delta_2}(\theta^\star)\right] .$$

- MLPs at large-but-finite width are *cubic models*

$$z_i(x_\delta;\theta) = \sum_{j=0}^{n_f} W_{ij}\phi_j(x_\delta) + \frac{1}{2}\sum_{j_1,j_2=0}^{n_f} W_{ij_1}W_{ij_2}\psi_{j_1j_2}(x_\delta)$$

$$+ \frac{1}{6}\sum_{j_1,j_2,j_3=0}^{n_f} W_{ij_1}W_{ij_2}W_{ij_3}\Psi_{j_1j_2j_3}(x_\delta)$$

- The amount of representation learning is set by the depth-to-width ratio, $\epsilon \equiv \frac{L}{n}$, with the depth L and width n.
- The $\phi_j(x_\delta)$, $\psi_{j_1j_2}(x_\delta)$, $\Psi_{j_1j_2j_3}(x_\delta)$ are *random*.

Some Takeaways

- The deep learning framework makes it easy to define and train *nonlinear* models, letting us approximate functions that are often easy for humans to do – *is there a cat in that image?* – but hard for humans to program: a.k.a AI.
- These nonlinear models are much richer than classical statistical models such as linear regression.
- We can understand deep learning using "effective theory" tools to analyze large-but-finite-width networks.
- There are many more exciting "experimental" results that are waiting to be analyzed theoretically.

Thank You!

This slide is intentionally left blank.