The Principles of Deep Learning Theory

Dan Roberts

MIT & Salesforce

January 13, 2022

The goal of this talk is to theoretically analyze \textit{deep} neural networks of \textit{finite width}. In particular, we’ll

\begin{enumerate}
  \item explain at a high level our approach, and
  \item analyze a simple model of representation learning in nonlinear models.
\end{enumerate}
Neural Networks

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

\[ f(x; \theta) \]

Neurons are then organized in parallel into layers, and deep neural networks are those composed of multiple layers in sequence.
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 \( \theta \) is a vector of a large number of parameters controlling the shape of the function.
Our goal is to analyze the \textit{trained} network function:

\[ f(x; \theta^*) \]
Our goal is to analyze the trained network function:

\[ f(x; \theta^*) . \]

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 \( f(x; \theta^*) \) around the initialized value of the parameters \( \theta \)

\[
f(x; \theta^*) = f(x; \theta) + (\theta^* - \theta) \frac{df}{d\theta} + \frac{1}{2} (\theta^* - \theta)^2 \frac{d^2f}{d\theta^2} + \ldots ,
\]

where \( f(x; \theta) \) and its derivatives on the right-hand side are all evaluated at initialized value of the parameters.
In general, the Taylor series contains an infinite number of terms
\[ f, \quad \frac{df}{d\theta}, \quad \frac{d^2f}{d\theta^2}, \quad \frac{d^3f}{d\theta^3}, \quad \frac{d^4f}{d\theta^4}, \quad \ldots, \]
and in principle we need to compute them all.
The Theoretical Minimum: Problem 2

Since the parameters $\theta$ 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:

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

This means that each term $f$, $df/d\theta$, $d^2f/d\theta^2$, $\ldots$, in the Taylor expansion is really a random function of the input $x$, and this joint distribution will have intricate statistical dependencies.
The learned value of the parameters, $\theta^*$, is the result of a complicated training process. In general, $\theta^*$ is not unique and can depend on everything:

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

Determining an analytical expression for $\theta^*$ 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

\[ p(f^*) \equiv p(f(x; \theta^*) \mid \text{learning algorithm; training data}) , \]

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

\[ p(f^*) \equiv p\left(f(x; \theta^*) \middle| \text{learning algorithm; training data}\right), \]

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

- A framework for analyzing \( p(f^*) \) 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

\[
p(f^*) \equiv p(f(x; \theta^*) \mid \text{learning algorithm; training data}),
\]

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

- A framework for analyzing \( p(f^*) \) 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 \( p(f^*) \) 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

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

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

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

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, $\epsilon \equiv 1/n$: 

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

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

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

\[ p(\theta) \rightarrow p\left(f, \frac{df}{d\theta}, \frac{d^2f}{d\theta^2}, \ldots \right). \]

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:

\[ p(\theta) \rightarrow p\left(f, \frac{df}{d\theta}, \frac{d^2f}{d\theta^2}, \ldots\right). \]

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

Problem 3 is about the training dynamics:

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

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

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

- \( i = 1, \ldots, n_{\text{out}} \) 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 \( \theta \) 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 \( \theta \) 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 \( \theta \) 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.
Instead, we might design a fixed basis of feature functions $\phi_j(x)$ that are meant to fit more complicated functions:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_f} W_{ij} \phi_j(x_\delta)$$
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 $\phi_j(x)$. 
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 $\phi_j(x)$. 
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 \( W_{i0} \equiv b_i \).
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.)
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

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

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**

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

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

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

\[
0 = \frac{d\mathcal{L}_A}{dW_{ij}} \bigg|_{W=W^*}
\]
Linear Regression

Supervised learning with a linear model is linear regression

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

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

▶ We could solve by direct optimization:

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

▶ We could solve by gradient descent:

$$
W_{ij}(t + 1) = W_{ij}(t) - \eta \left. \frac{d \mathcal{L}_A}{dW_{ij}} \right|_{W=W_{ij}(t)}.
$$
Let us introduce a new $N_D \times N_D$-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 $N_A$-by-$N_A$-dimensional submatrix of the kernel evaluated on the training set as $\tilde{k}_{\tilde{\alpha}_1 \tilde{\alpha}_2}$ with a tilde. This lets us write its inverse as $\tilde{k}_{\tilde{\alpha}_1 \tilde{\alpha}_2}$, which satisfies

$$\sum_{\tilde{\alpha}_2 \in A} \tilde{k}_{\tilde{\alpha}_1 \tilde{\alpha}_2} \tilde{k}_{\tilde{\alpha}_2 \tilde{\alpha}_3} = \delta_{\tilde{\alpha}_1 \tilde{\alpha}_3}.$$
Linear Models and Kernel Methods

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

- **parameter space – linear regression**

  \[ z_i(x_{\hat{\beta}}; \theta^*) = \sum_{j=0}^{n_f} W_{ij}^* \phi_j(x_{\hat{\beta}}) \]

- **sample space – kernel methods**

  \[ z_i(x_{\hat{\beta}}; \theta^*) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in A} k_{\hat{\beta}\tilde{\alpha}_1} \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_2} y_i; \tilde{\alpha}_2. \]
Two forms of a solution for a **linear model**:

- **parameter space – linear regression**

  \[ z_i(x_\beta; \theta^*) = \sum_{j=0}^{n_f} W_{ij} \phi_j(x_\beta) \]

- **sample space – kernel methods**

  \[ z_i(x_\beta; \theta^*) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in A} k_{\tilde{\beta}\tilde{\alpha}_1} \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_2} y_i; \tilde{\alpha}_2 \cdot \]

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.
Linear Regression goes back to Legendre and Gauss.

- “Three Problems” are tractable and can analyze completely.
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.
Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear both in $x$ and $\theta$. 
Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear both in $x$ and $\theta$.

(i) The generalization of the $\phi(x)$ are inherited from the network and are random at the beginning of training.
Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear both in $x$ and $\theta$.

(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.
Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear both in $x$ and $\theta$.

(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.
Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear \textit{both in }x\textit{ and }\theta\textit{.}

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

\[ \phi(x) \rightarrow \phi(x; \theta^*) . \]
Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear \textit{both} in $x$ and $\theta$.

(ii) The effective features \textit{evolve} over the course of training:

$$\phi(x) \rightarrow \phi(x; \theta^*) .$$

- No longer just fitting a curve with a fixed basis!
Deep learning extends this classic paradigm in 2 important ways: neural networks are typically nonlinear both in $x$ and $\theta$.

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

$$\phi(x) \rightarrow \phi(x; \theta^*) .$$

- 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:

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

▶ 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**:

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

- It’s nonlinear because it’s quadratic in the weights: \( W_{ij_1} W_{ij_2} \).
- \( \epsilon \ll 1 \) is a small parameter that controls the size of the deformation.
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.
- We’ve introduced \((n_f + 1)(n_f + 2)/2\) meta feature functions, \( \psi_{j_1j_2}(x) \), with two feature indices.
Quadratic Models

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

$$z_i(x_\delta; \theta + d\theta) = z_i(x_\delta; \theta) + \sum_{j=0}^{n_f} dW_{ij} \left[ \phi_j(x_\delta) + \epsilon \sum_{j_1=0}^{n_f} W_{ij_1} \psi_{j_1j}(x_\delta) \right]$$

$$+ \frac{\epsilon}{2} \sum_{j_1,j_2=0}^{n_f} dW_{ij_1} dW_{ij_2} \psi_{j_1j_2}(x_\delta).$$
Quadratic Models

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

$$z_i(x_\delta; \theta + d\theta) = z_i(x_\delta; \theta) + \sum_{j=0}^{n_f} dW_{ij} \left[ \phi_j(x_\delta) + \epsilon \sum_{j_1=0}^{n_f} W_{ij_1} \psi_{j_1j}(x_\delta) \right]$$

$$+ \frac{\epsilon}{2} \sum_{j_1, j_2=0}^{n_f} dW_{ij_1} dW_{ij_2} \psi_{j_1j_2}(x_\delta).$$

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

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

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, $\phi_{ij}^E(x_\delta; \theta)$. For comparison, for the linear model we'd have:

$z_i(x_\delta; \theta + d\theta) = z_i(x_\delta; \theta) + n_f \sum_{j=0} \sum_{k} dW_{ik} \psi_{kj}(x_\delta)$. 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.
Effective Feature Learning

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

The $\phi_{ij}^E(x_\delta; \theta)$ learns with update $dW_{ik}$:

$$
\phi_{ij}^E(x_\delta; \theta + d\theta) = \phi_{ij}^E(x_\delta; \theta) + \epsilon \sum_{k=0}^{n_f} dW_{ik} \psi_{kj}(x_\delta).
$$
Effective Feature Learning

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

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

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

- 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, $\phi_{ij}^E(x_\delta; \theta)$.

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

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

- 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 \textit{quadratic regression}:

\[
\mathcal{L}_A(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in A} \sum_{i=1}^{n_{out}} \left[ y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij} \phi_j(x_{\tilde{\alpha}}) - \frac{\epsilon}{2} \sum_{j_1,j_2=0}^{n_f} W_{ij_1} W_{ij_2} \psi_{j_1j_2}(x_{\tilde{\alpha}}) \right]^2.
\]
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:

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

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

$$0 = \frac{d\mathcal{L}_A}{dW_{ij}} \bigg|_{W=W^*},$$

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**:

\[
\mathcal{L}_A(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in A} \sum_{i=1}^{n_{\text{out}}} \left[ y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij} \phi_j(x_{\tilde{\alpha}}) - \frac{\epsilon}{2} \sum_{j_1,j_2=0}^{n_f} W_{ij_1} W_{ij_2} \psi_{j_1j_2}(x_{\tilde{\alpha}}) \right]^2 .
\]

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

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

This will find a minimum in practice.
Quadratic Model Gradient Descent Dynamics

The weights will update as

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

$$= W_{ij}(t) - \eta \sum_{\tilde{\alpha}} \phi_{ij;\tilde{\alpha}}^E(t) (z_{i;\tilde{\alpha}}(t) - y_{i;\tilde{\alpha}}).$$

While the model and effective features update as

$$z_{i;\delta}(t + 1) = z_{i;\delta}(t) + \sum_j dW_{ij}(t) \phi_{ij;\delta}^E(t)$$

$$+ \frac{\epsilon}{2} \sum_{j_1,j_2} dW_{ij_1}(t) dW_{ij_2}(t) \psi_{j_1j_2}(x_{\delta}),$$

$$\phi_{ij;\delta}^E(t + 1) = \phi_{ij;\delta}^E(t) + \epsilon \sum_{k=0}^{n_f} dW_{ik}(t) \psi_{kj}(x_{\delta}).$$
Aside: Effective Kernel

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

$$k_{ii;\delta_1\delta_2}(\theta) \equiv \sum_{j=0}^{n_f} \phi_{ij}^E(x_{\delta_1}; \theta) \phi_{ij}^E(x_{\delta_2}; \theta),$$

which measures a parameter-dependent similarity between two inputs $x_{\delta_1}$ and $x_{\delta_2}$ using our effective features $\phi_{ij}^E(x_{\delta}; \theta)$. 
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 \( x_{\delta_0} \), \( \mu_{\delta_0 \delta_1 \delta_2} \) computes a different feature-space inner product between the two inputs, \( x_{\delta_1} \) & \( x_{\delta_2} \).
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.
- For a fixed input $x_{\delta_0}$, $\mu_{\delta_0 \delta_1 \delta_2}$ computes a different feature-space inner product between the two inputs, $x_{\delta_1}$ & $x_{\delta_2}$.
- Due to the inclusion of $\epsilon$ 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

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

\[ = W_{ij}(t) - \eta \sum_{\tilde{\alpha}} \phi^E_{ij;\tilde{\alpha}}(t) \left( z_{i;\tilde{\alpha}}(t) - y_{i;\tilde{\alpha}} \right). \]

While the model and effective features update as

\[ z_{i;\delta}(t + 1) = z_{i;\delta}(t) + \sum_j dW_{ij}(t) \phi^E_{ij;\delta}(t) \]

\[ + \frac{\epsilon}{2} \sum_{j_1,j_2} dW_{ij_1}(t) dW_{ij_2}(t) \psi_{j_1j_2}(x_\delta), \]

\[ \phi^E_{ij;\delta}(t + 1) = \phi^E_{ij;\delta}(t) + \epsilon \sum_{k=0}^{n_f} dW_{ik}(t) \psi_{kj}(x_\delta). \]
The model predictions will update as

\[ z_{i;\delta}(t + 1) = z_{i;\delta}(t) - \eta \sum_{\tilde{\alpha}} k^{E}_{i;\delta\tilde{\alpha}}(t) \epsilon_{i;\tilde{\alpha}}(t) + \frac{\eta^2}{2} \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2} \mu \delta \tilde{\alpha}_1 \tilde{\alpha}_2 \epsilon_{i;\tilde{\alpha}_1}(t) \epsilon_{i;\tilde{\alpha}_2}(t) + \ldots, \]

while the effective kernel will update as

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

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

\[ z_{i;\beta}(\infty) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in A} k_{\beta \tilde{\alpha}_1} \tilde{k} \tilde{\alpha}_1 \tilde{\alpha}_2 y_i;\tilde{\alpha}_2 \]

\[ + \sum_{\tilde{\alpha}_1, \ldots, \tilde{\alpha}_4 \in A} \left[ \mu \tilde{\alpha}_1 \beta \tilde{\alpha}_2 - \sum_{\tilde{\alpha}_5, \tilde{\alpha}_6 \in A} k_{\beta \tilde{\alpha}_5} \tilde{k} \tilde{\alpha}_5 \tilde{\alpha}_6 \mu \tilde{\alpha}_1 \tilde{\alpha}_6 \tilde{\alpha}_2 \right] Z_A \tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4 y_i;\tilde{\alpha}_3 y_i;\tilde{\alpha}_4 \]

\[ + \sum_{\tilde{\alpha}_1, \ldots, \tilde{\alpha}_4 \in A} \left[ \mu \beta \tilde{\alpha}_1 \tilde{\alpha}_2 - \sum_{\tilde{\alpha}_5, \tilde{\alpha}_6 \in A} k_{\beta \tilde{\alpha}_5} \tilde{k} \tilde{\alpha}_5 \tilde{\alpha}_6 \mu \tilde{\alpha}_6 \tilde{\alpha}_1 \tilde{\alpha}_2 \right] Z_B \tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4 y_i;\tilde{\alpha}_3 y_i;\tilde{\alpha}_4 \]

where the **algorithm projectors** are given by

\[ Z_A^{\tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4} \equiv \tilde{k} \tilde{\alpha}_1 \tilde{\alpha}_3 \tilde{k} \tilde{\alpha}_2 \tilde{\alpha}_4 - \sum_{\tilde{\alpha}_5} \tilde{k} \tilde{\alpha}_2 \tilde{\alpha}_5 X_{\parallel} \tilde{\alpha}_1 \tilde{\alpha}_5 \tilde{\alpha}_3 \tilde{\alpha}_4 \],

\[ Z_B^{\tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4} \equiv \tilde{k} \tilde{\alpha}_1 \tilde{\alpha}_3 \tilde{k} \tilde{\alpha}_2 \tilde{\alpha}_4 - \sum_{\tilde{\alpha}_5} \tilde{k} \tilde{\alpha}_2 \tilde{\alpha}_5 X_{\parallel} \tilde{\alpha}_1 \tilde{\alpha}_5 \tilde{\alpha}_3 \tilde{\alpha}_4 + \frac{\eta}{2} X_{\parallel} \tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4 \].
Here, an **inverting tensor** is implicitly defined:

\[
\delta_{\tilde{\alpha}_1 \tilde{\alpha}_5} \delta_{\tilde{\alpha}_2 \tilde{\alpha}_6} = \sum_{\tilde{\alpha}_3, \tilde{\alpha}_4 \in A} X_{\|}^{\tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4} \frac{1}{\eta} \left[ \delta_{\tilde{\alpha}_3 \tilde{\alpha}_5} \delta_{\tilde{\alpha}_4 \tilde{\alpha}_6} - (\delta_{\tilde{\alpha}_3 \tilde{\alpha}_5} - \eta k_{\tilde{\alpha}_3 \tilde{\alpha}_5}) (\delta_{\tilde{\alpha}_4 \tilde{\alpha}_6} - \eta k_{\tilde{\alpha}_4 \tilde{\alpha}_6}) \right]
\]

\[
= \sum_{\tilde{\alpha}_3, \tilde{\alpha}_4 \in A} X_{\|}^{\tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4} \left( k_{\tilde{\alpha}_3 \tilde{\alpha}_5} \delta_{\tilde{\alpha}_4 \tilde{\alpha}_6} + \delta_{\tilde{\alpha}_3 \tilde{\alpha}_5} k_{\tilde{\alpha}_4 \tilde{\alpha}_6} - \eta k_{\tilde{\alpha}_3 \tilde{\alpha}_5} k_{\tilde{\alpha}_4 \tilde{\alpha}_6} \right)
\]
z_{i;\tilde{\beta}}(\infty) = \sum_{\tilde{\alpha}_1,\tilde{\alpha}_2 \in A} k_{\tilde{\beta}\tilde{\alpha}_1} \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} \\
+ \sum_{\tilde{\alpha}_1,\ldots,\tilde{\alpha}_4 \in A} \left[ \mu_{\tilde{\alpha}_1\tilde{\beta}\tilde{\alpha}_2} - \sum_{\tilde{\alpha}_5,\tilde{\alpha}_6 \in A} k_{\tilde{\beta}\tilde{\alpha}_5} \tilde{k}_{\tilde{\alpha}_5\tilde{\alpha}_6} \mu_{\tilde{\alpha}_1\tilde{\alpha}_6\tilde{\alpha}_2} \right] Z_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 A} \left[ \mu_{\tilde{\beta}\tilde{\alpha}_1\tilde{\alpha}_2} - \sum_{\tilde{\alpha}_5,\tilde{\alpha}_6 \in A} k_{\tilde{\beta}\tilde{\alpha}_5} \tilde{k}_{\tilde{\alpha}_5\tilde{\alpha}_6} \mu_{\tilde{\alpha}_6\tilde{\alpha}_1\tilde{\alpha}_2} \right] Z_B^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} y_{i;\tilde{\alpha}_3} y_{i;\tilde{\alpha}_4}

where the algorithm projectors are given by

\[ Z_A^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} \equiv \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_3} \tilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_4} - \sum_{\tilde{\alpha}_5} \tilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_5} X_{\parallel}^{\tilde{\alpha}_1\tilde{\alpha}_5\tilde{\alpha}_3\tilde{\alpha}_4}, \]

\[ Z_B^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4} \equiv \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_3} \tilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_4} - \sum_{\tilde{\alpha}_5} \tilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_5} X_{\parallel}^{\tilde{\alpha}_1\tilde{\alpha}_5\tilde{\alpha}_3\tilde{\alpha}_4} + \frac{\eta}{2} X_{\parallel}^{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}. \]
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:

\[
Z_A \tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4 = 0, \quad Z_B \tilde{\alpha}_1 \tilde{\alpha}_2 \tilde{\alpha}_3 \tilde{\alpha}_4 = \frac{1}{2} \tilde{k}_1 \tilde{\alpha}_3 \tilde{k}_2 \tilde{\alpha}_4.
\]
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:

\[
Z_{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}^A = 0, \quad Z_{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}^B = \frac{1}{2} \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_3} \tilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_4}.
\]

▶ In the ODE limit, we get different predictions

\[
Z_{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}^A = Z_{\tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4}^B \equiv \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_3} \tilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_4} - \sum_{\tilde{\alpha}_5} \tilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_5} X_{\|} \tilde{\alpha}_1\tilde{\alpha}_5\tilde{\alpha}_3\tilde{\alpha}_4,
\]

\[
\sum_{\tilde{\alpha}_3, \tilde{\alpha}_4 \in A} X_{\|} \tilde{\alpha}_1\tilde{\alpha}_2\tilde{\alpha}_3\tilde{\alpha}_4 \left( \tilde{k}_{\tilde{\alpha}_3\tilde{\alpha}_5} \delta_{\tilde{\alpha}_4\tilde{\alpha}_6} + \delta_{\tilde{\alpha}_3\tilde{\alpha}_5} \tilde{k}_{\tilde{\alpha}_4\tilde{\alpha}_6} \right) = \delta_{\tilde{\alpha}_1\tilde{\alpha}_5} \delta_{\tilde{\alpha}_2\tilde{\alpha}_6},
\]
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 $\propto y_i;\bar{\alpha}_2$, and
▶ it also has a new quadratic piece $\propto y_i;\bar{\alpha}_1 y_i;\bar{\alpha}_2$. 
For simplicity, let’s pick the **direct optimization** solution:

\[ k_{ii;\delta_1\delta_2}^E(\theta^*) = k_{\delta_1\delta_2} + \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in A} (\mu_{\delta_1\delta_2}\tilde{\alpha}_1 + \mu_{\delta_2}\delta_1\tilde{\alpha}_1)\tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_2}y_{i;\tilde{\alpha}_2} + O(\epsilon^2) . \]

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

\[ k_{ii;\delta_1\delta_2}^{\#} \equiv \frac{1}{2} \left[ k_{\delta_1\delta_2} + k_{ii;\delta_1\delta_2}^E(\theta^*) \right] . \]

Now the nearly-kernel prediction formula can be compressed,

\[ z_i(x_\beta; \theta^*) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in A} k_{ii;\tilde{\beta}\tilde{\alpha}_1}^{\#} \tilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_2}y_{i;\tilde{\alpha}_2} + O(\epsilon^2) , \]

taking the form of a *kernel prediction*, but with the benefit of nontrivial feature evolution incorporated into the trained kernel.
The direct optimization solution in parameter space is

\[ z_i(x_\beta; \theta^*) = \sum_{j=0}^{n_f} W_{ij}^* \phi_j(x_\beta) + \frac{\epsilon}{2} \sum_{j_1,j_2=0}^{n_f} W_{ij_1}^* W_{ij_2}^* \psi_{j_1j_2}(x_\beta) \]

and the optimal parameters can decompose as

\[ W_{ij}^* \equiv W_{ij}^F + W_{ij}^l, \]

where \( W_{ij}^F \) are the optimal parameters from the linear model.
Representation Learning as Regularization

The *direct optimization* solution in parameter space is

\[
    z_i(x_\beta; \theta^*) = \sum_{j=0}^{n_f} W^*_{ij} \phi_j(x_\beta) + \frac{\epsilon}{2} \sum_{j_1,j_2=0}^{n_f} W^*_{ij_1} W^*_{ij_2} \psi_{j_1j_2}(x_\beta)
\]

and the optimal parameters can decompose as

\[
    W^*_{ij} \equiv W^F_{ij} + W^l_{ij},
\]

where \(W^F_{ij}\) are the optimal parameters from the linear model.

▷ The \(O(\epsilon)\) tunings \(W^l_{ij}\) 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

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

and the optimal parameters can decompose as

$$W_{ij}^* \equiv W_{ij}^F + W_{ij}^l,$$

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

- The $O(\epsilon)$ tunings $W_{ij}^l$ ruin the **fine tuning** of the $W_{ij}^F$, 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

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

where \( \hat{\beta} \in B \) is a sample index in the test set, with

\[ z_{i;\hat{\beta}}(\epsilon) = z_{i;\hat{\beta}}^F + \epsilon z_{i;\hat{\beta}}^l + O(\epsilon^2). \]
Does feature learning help generalization?

Consider the generalization error

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

where \( \hat{\beta} \in B \) is a sample index in the test set, with

\[ z_{i;\hat{\beta}}(\epsilon) = z_{i;\hat{\beta}}^F + \epsilon z_{i;\hat{\beta}}^I + O(\epsilon^2). \]

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

\[ \frac{d \mathcal{L}_B}{d \epsilon} = \sum_{\hat{\beta} \in B} \sum_{i=1}^{n_{\text{out}}} \frac{\partial \mathcal{L}_B(\epsilon)}{\partial z_{i;\hat{\beta}}} \frac{dz_{i;\hat{\beta}}}{d \epsilon} < 0 \]
Does feature learning help generalization?

Consider the generalization error

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

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

\[ z_{i;\beta}(\epsilon) = z_{i;\beta}^F + \epsilon z_{i;\beta}^I + O(\epsilon^2). \]

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

\[ \frac{d \mathcal{L}_B}{d\epsilon} = \sum_{\beta \in B} \sum_{i=1}^{n_{\text{out}}} \left( z_{i;\beta}(\epsilon) - y_{i;\beta} \right) \frac{dz_{i;\beta}}{d\epsilon} < 0 \]
Does feature learning help generalization?

Consider the generalization error

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

where $\beta \in B$ is a sample index in the test set, with

$$z_{i;\beta}(\epsilon) = z^F_{i;\beta} + \epsilon z^I_{i;\beta} + O(\epsilon^2).$$

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

$$\frac{d \mathcal{L}_B}{d\epsilon} = \sum_{\beta \in B} \sum_{i=1}^{n_{out}} (z_{i;\beta}(\epsilon) - y_{i;\beta}) \epsilon z^I_{i;\beta} + O(\epsilon^2) < 0.$$
Does feature learning help generalization?

Consider the generalization error

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

where \( \hat{\beta} \in B \) is a sample index in the test set, with

\[ z_{i;\hat{\beta}}(\epsilon) = z_{i;\hat{\beta}}^F + \epsilon z_{i;\hat{\beta}}^l + O(\epsilon^2). \]

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

\[ \frac{d\mathcal{L}_B}{d\epsilon} = \sum_{\beta \in B} \sum_{i=1}^{n_{out}} \left( z_{i;\hat{\beta}}^F - y_{i;\hat{\beta}} \right) \epsilon z_{i;\hat{\beta}}^l + O(\epsilon^2) < 0 \]
Does feature learning help generalization?

Consider the generalization error

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

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

$$
z_{i;\hat{\beta}}(\epsilon) = z_{i;\hat{\beta}}^F + \epsilon z_{i;\hat{\beta}}^l + O(\epsilon^2).
$$

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

$$
\frac{d\mathcal{L}_B}{d\epsilon} = \sum_{\hat{\beta} \in B} \sum_{i=1}^{n_{\text{out}}} \left( z_{i;\hat{\beta}}^F - y_{i;\hat{\beta}} \right) \epsilon z_{i;\hat{\beta}}^l + O(\epsilon^2) < 0
$$

*Depends on the initial training error and the nonlinear prediction.*
How Much?

Need to evaluate our solution to order $\epsilon^2$:

$$z_{i;\hat{\beta}}(\epsilon) = z_{i;\hat{\beta}}^F + \epsilon z_{i;\hat{\beta}}^I + \epsilon^2 z_{i;\hat{\beta}}^{II} + O(\epsilon^3).$$
How Much?

Need to evaluate our solution to order $\epsilon^2$:

$$z_{i;\hat{\beta}}(\epsilon) = z_{i;\hat{\beta}}^F + \epsilon z_{i;\hat{\beta}}^I + \epsilon^2 z_{i;\hat{\beta}}^{\|} + O(\epsilon^3).$$

Then, by calculating

$$0 = \left. \frac{d\mathcal{L}_B}{d\epsilon} \right|_{\epsilon \to \epsilon^*},$$

we can optimize the amount of feature learning:

$$\epsilon^* = \frac{- \sum_{\hat{\beta} \in B} \sum_{i=1}^{n_{out}} \left( z_{i;\hat{\beta}}^F - y_{i;\hat{\beta}} \right) z_{i;\hat{\beta}}^I}{\sum_{\hat{\beta} \in B} \sum_{i=1}^{n_{out}} \left[ \left( z_{i;\hat{\beta}}^I \right)^2 + \left( z_{i;\hat{\beta}}^F - y_{i;\hat{\beta}} \right) z_{i;\hat{\beta}}^{\|} \right]}.$$
How Much?

Need to evaluate our solution to order $\epsilon^2$:

$$z_{i;\hat{\beta}}(\epsilon) = z_{i;\hat{\beta}}^F + \epsilon z_{i;\hat{\beta}}^I + \epsilon^2 z_{i;\hat{\beta}}^{II} + O(\epsilon^3).$$

Then, by calculating

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

we can optimize the amount of feature learning:

$$\epsilon^* = \frac{- \sum_{\hat{\beta} \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left( z_{i;\hat{\beta}}^F - y_{i;\hat{\beta}} \right) z_{i;\hat{\beta}}^I}{\sum_{\hat{\beta} \in \mathcal{B}} \sum_{i=1}^{n_{\text{out}}} \left[ \left( z_{i;\hat{\beta}}^I \right)^2 + \left( z_{i;\hat{\beta}}^F - y_{i;\hat{\beta}} \right) z_{i;\hat{\beta}}^{II} \right]}$$

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:

\[
    z_i(x_\delta; \theta^*) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in \mathcal{A}} k^\#_{ii; \tilde{\delta}\tilde{\alpha}_1} k^\#_{ii} y_{i; \tilde{\alpha}_2} + O(\epsilon^2),
\]

\[
k^\#_{ii; \tilde{\delta}_1 \tilde{\delta}_2} \equiv \frac{1}{2} \left[ k_{\delta_1 \delta_2} + k^E_{ii; \tilde{\delta}_1 \tilde{\delta}_2}(\theta^*) \right].
\]
Quadratic Models vs. Deep Learning

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

  \[
  z_i(x_δ; θ^*) = \sum_{\tilde{α}_1, \tilde{α}_2 ∈ A} k_{ii; δ}^{\#} \tilde{α}_1 \tilde{α}_2 \sum_{\tilde{α}_1, \tilde{α}_2 ∈ A} y_{i; \tilde{α}_2} + O(\epsilon^2),
  \]

  \[
  k_{ii; δ}^{\#} \equiv \frac{1}{2} \left[ k_{δ_1 δ_2} + k_{ii; δ_1 δ_2}^E(θ^*) \right].
  \]

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

  \[
  z_i(x_δ; θ) = \sum_{j=0}^{n_f} W_{ij} φ_j(x_δ) + \frac{1}{2} \sum_{j_1, j_2=0}^{n_f} W_{ij_1} W_{ij_2} \psi_{j_1 j_2}(x_δ)
  \]

  \[
  + \frac{1}{6} \sum_{j_1, j_2, j_3=0}^{n_f} W_{ij_1} W_{ij_2} W_{ij_3} \psi_{j_1 j_2 j_3}(x_δ)
  \]
Quadratic Models vs. Deep Learning

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

\[
    z_i(x_\delta; \theta^*) = \sum_{\bar{\alpha}_1, \bar{\alpha}_2 \in A} k_{ii;\bar{\alpha}_1}^\# k_{ii}^\# \bar{\alpha}_1 \bar{\alpha}_2 y_{i;\bar{\alpha}_2} + O(\epsilon^2),
\]

\[
k_{ii;\delta_1 \delta_2}^\# \equiv \frac{1}{2} \left[ k_{\delta_1 \delta_2} + k_{ii;\delta_1 \delta_2}^E(\theta^*) \right].
\]

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

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

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

▶ 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:

\[ z_i(x_\delta; \theta^*) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in \mathcal{A}} k^\#_{ii;\delta\tilde{\alpha}_1} k^\#_{ii} y_{i;\tilde{\alpha}_2} + O(\epsilon^2), \]

\[ k^\#_{ii;\delta_1\delta_2} \equiv \frac{1}{2} \left[ k_{\delta_1\delta_2} + k^E_{ii;\delta_1\delta_2}(\theta^*) \right]. \]

- 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.