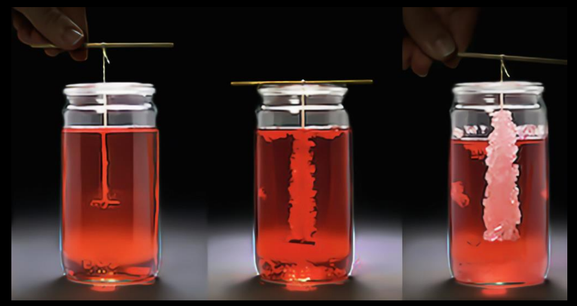
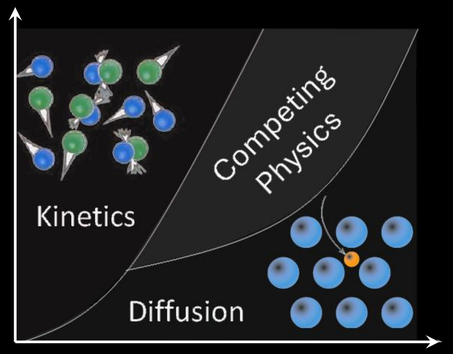
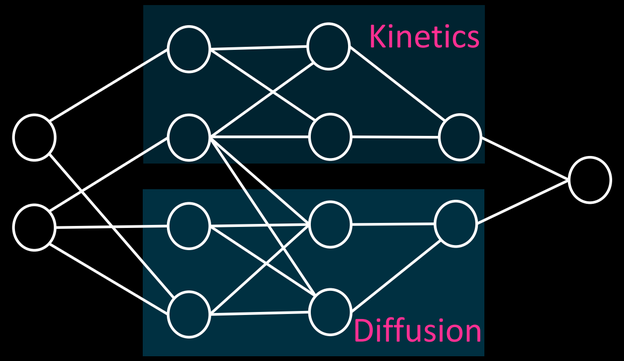
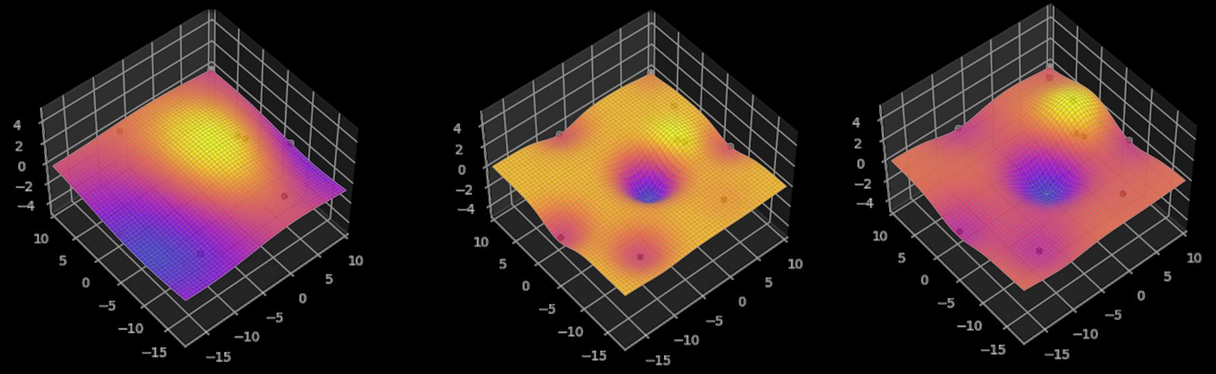
Physics Informed Machine Learning Introduction





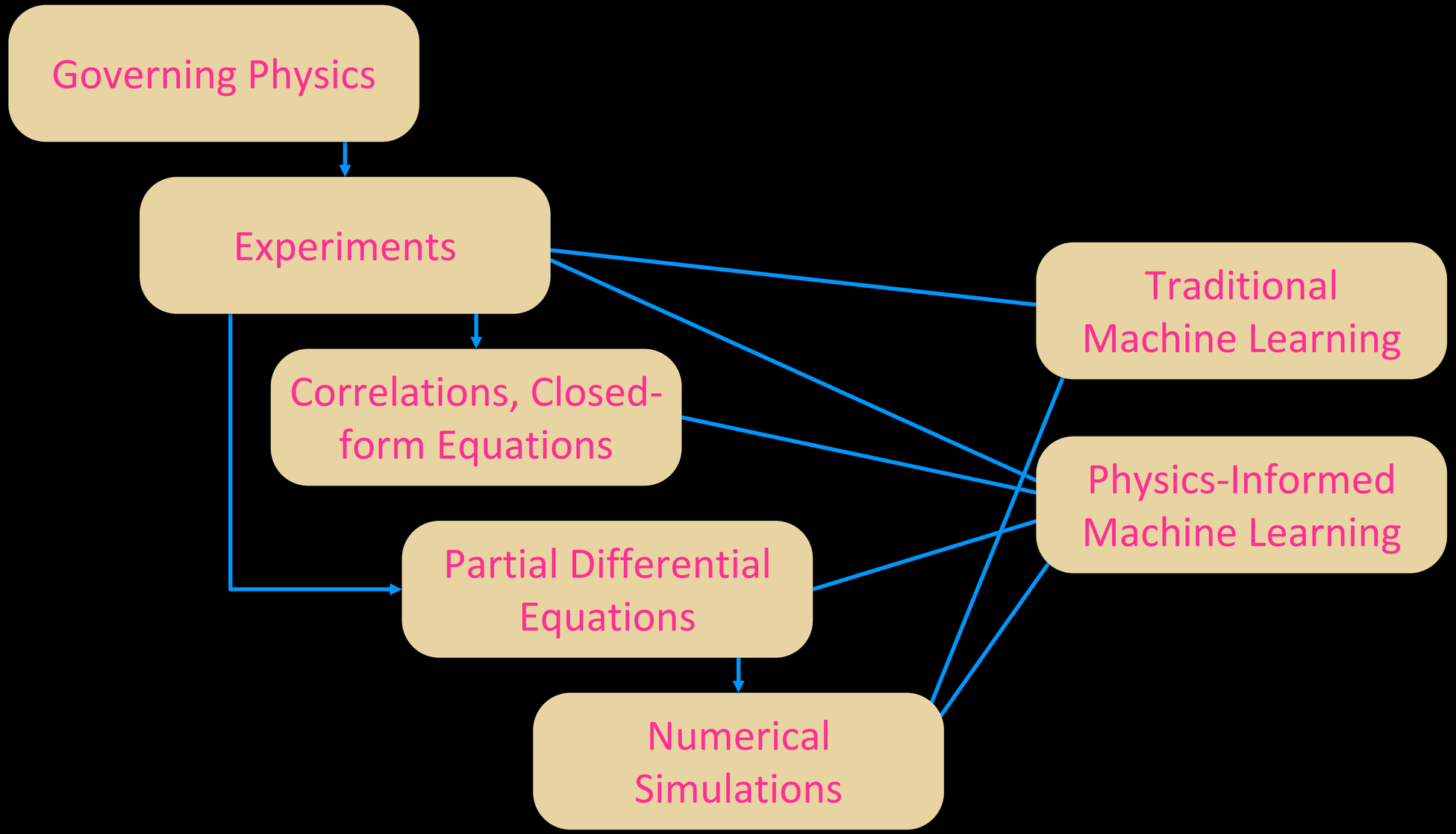




Navid Zobeiry, Associate Professor
Email: navidz@uw.edu
Materials Science & Engineering
https://composites.uw.edu/AI/

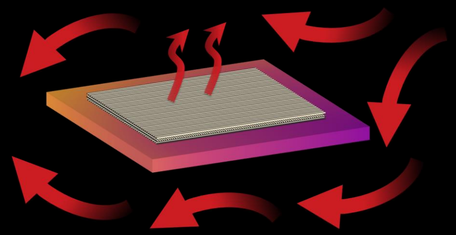
UNIVERSITY of WASHINGTON

Data Sources and Machine Learning Methods

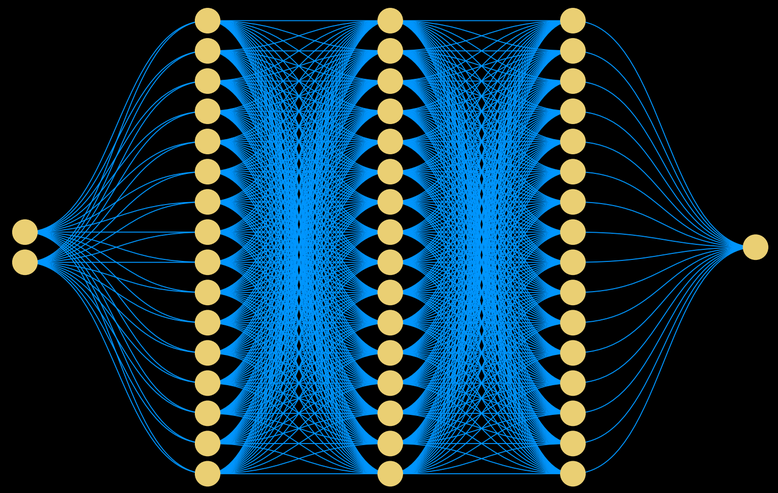


Case Studies and Machine Learning Algorithms

Case Study I: Heat Transfer

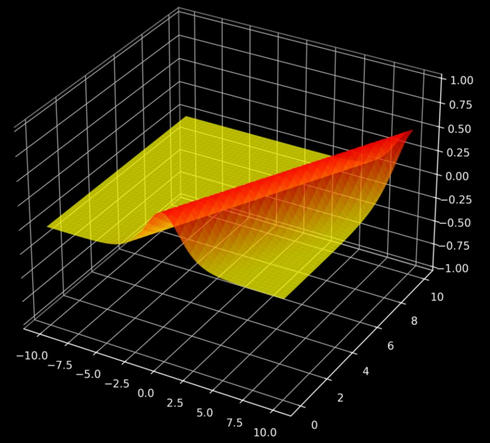


Neural Networks (NN)



Case Study II: Chemical Reactions

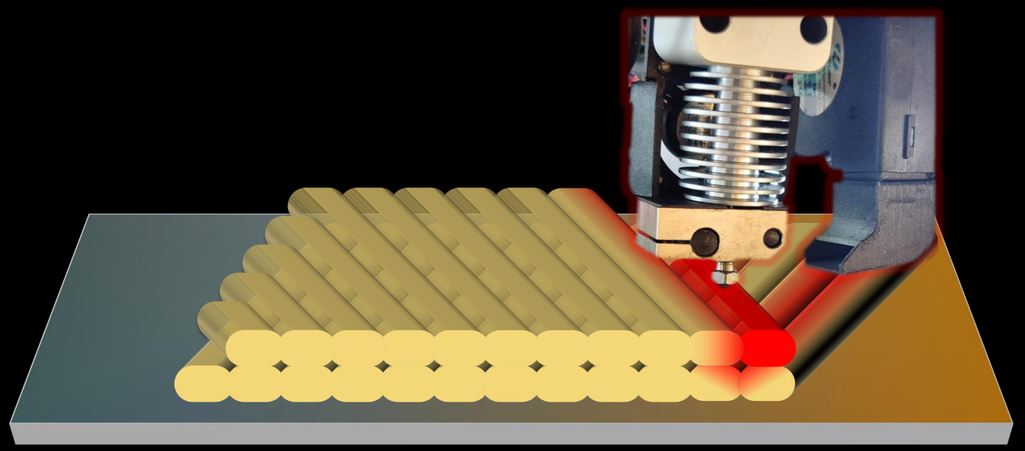
Sparse Identification of Nonlinear Dynamics (SINDy)



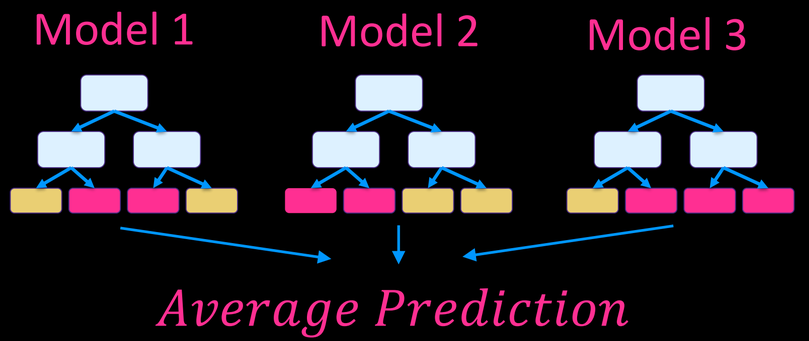
$$\frac{\mathrm{d}x}{dt} \approx c_1 x^{0.67}(1-x)^2 + c_2 x^{0.75}(1-x)^3$$

Case Studies and Machine Learning Algorithms

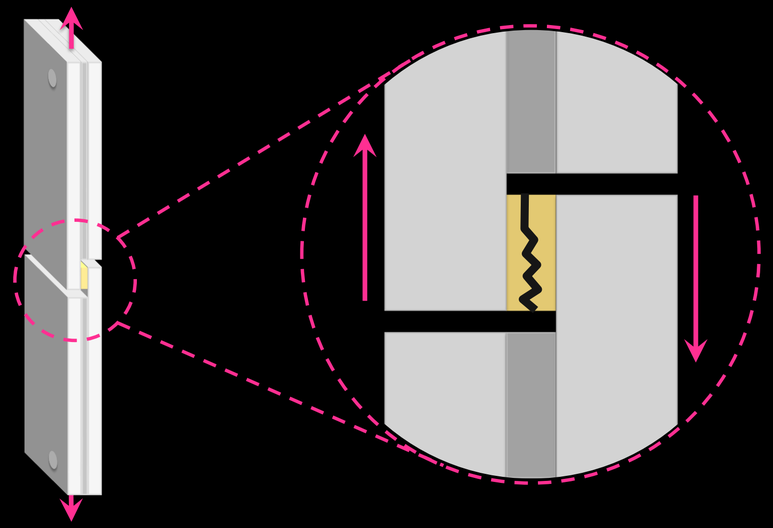
Case Study III: 3D Printing



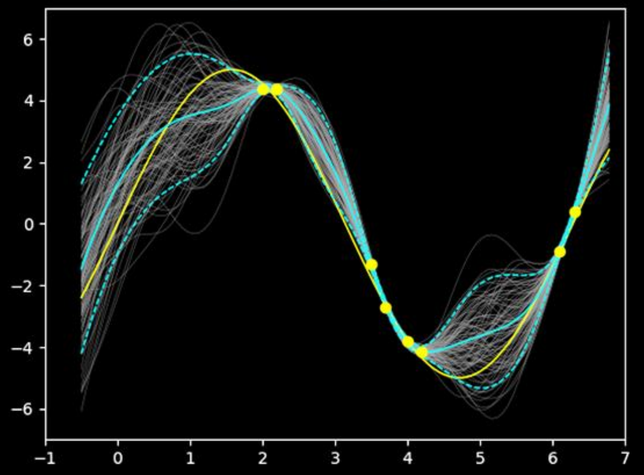
Ensemble Methods (Bagging and Boosting)



Case Study IV: Adhesive Failure



Gaussian Process Regression (GPR)



Reviewed Machine Learning Algorithms

Deterministic Regression

- Linear Regression
- Decision Tree
- Sparse Identification of Nonlinear Dynamics (SINDy)
- Neural Networks (NN)

Probabilistic Regression

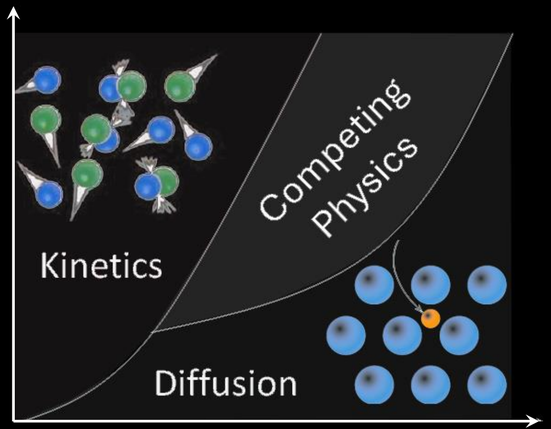
- Gaussian Process Regression (GPR)

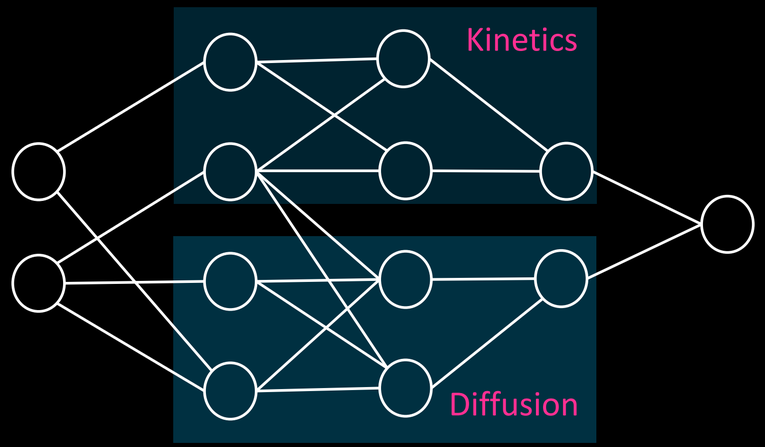
Ensemble Methods

- Bagging
  - Random Forest
- Boosting
  - Gradient Boosting Machine (GBM)
  - XGB Regressor

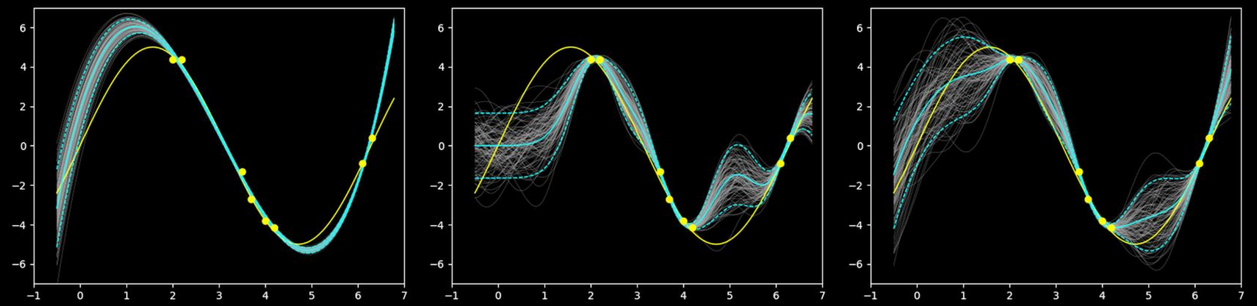
Physics Informed Machine Learning Techniques

- **Physics-informed Domain Transformation**





- **Physics-informed Model features and Parameters**



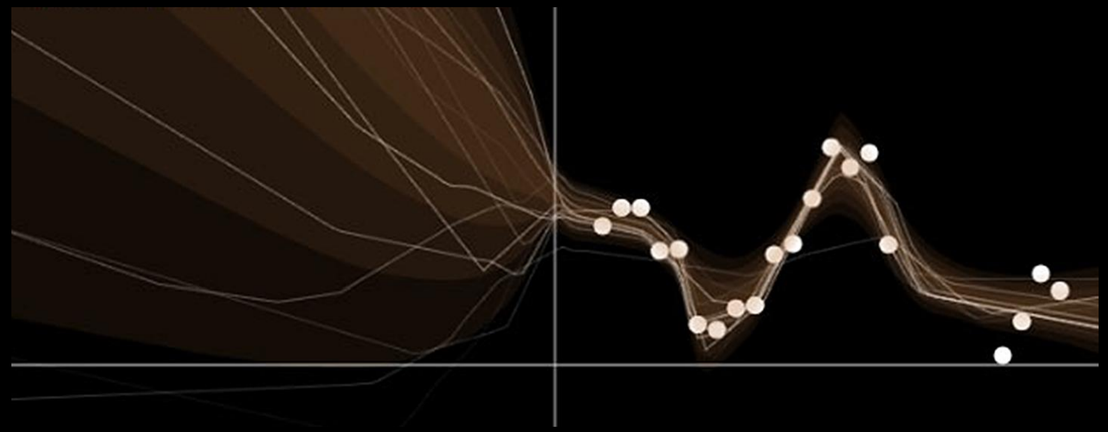
- **Physics-informed Loss**

$$Cost = \sum Err^2 + \sum \lambda_{Phy} Loss_{Phy}$$

Squares of Errors Physics-based Loss

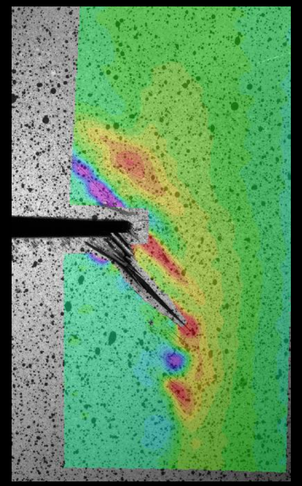
Small Data

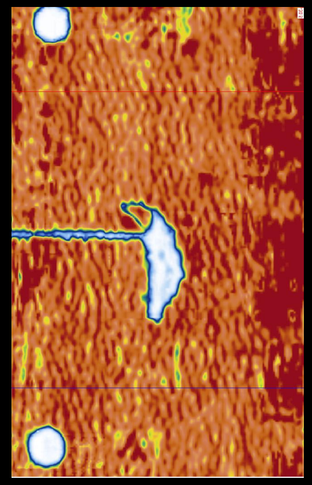
- Why small datasets? Difficult or expensive to generate large data in physical problems
- Why not small datasets?
  - Easily affected by noise and test errors
  - Cannot generalize the solution: Brittle AI

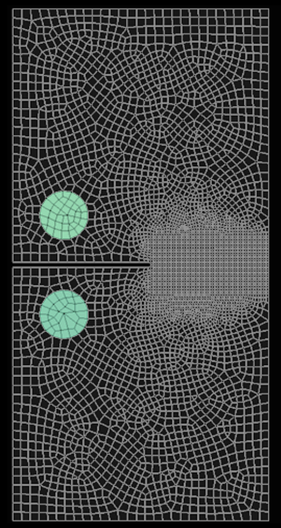


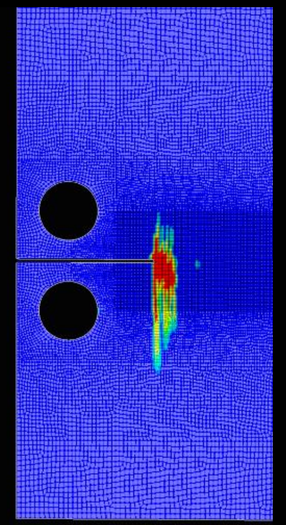
Sources of Large Data

- Factory data
- High through-put testing
- Simulation data
- Rich Data
  - Images
  - Time Series









Simple Regression Example

- We are conducting tests by changing two paramaters while measuring performance using a hidden function:

$$Z = f(x, y), \qquad x, y \in [-18,10]$$

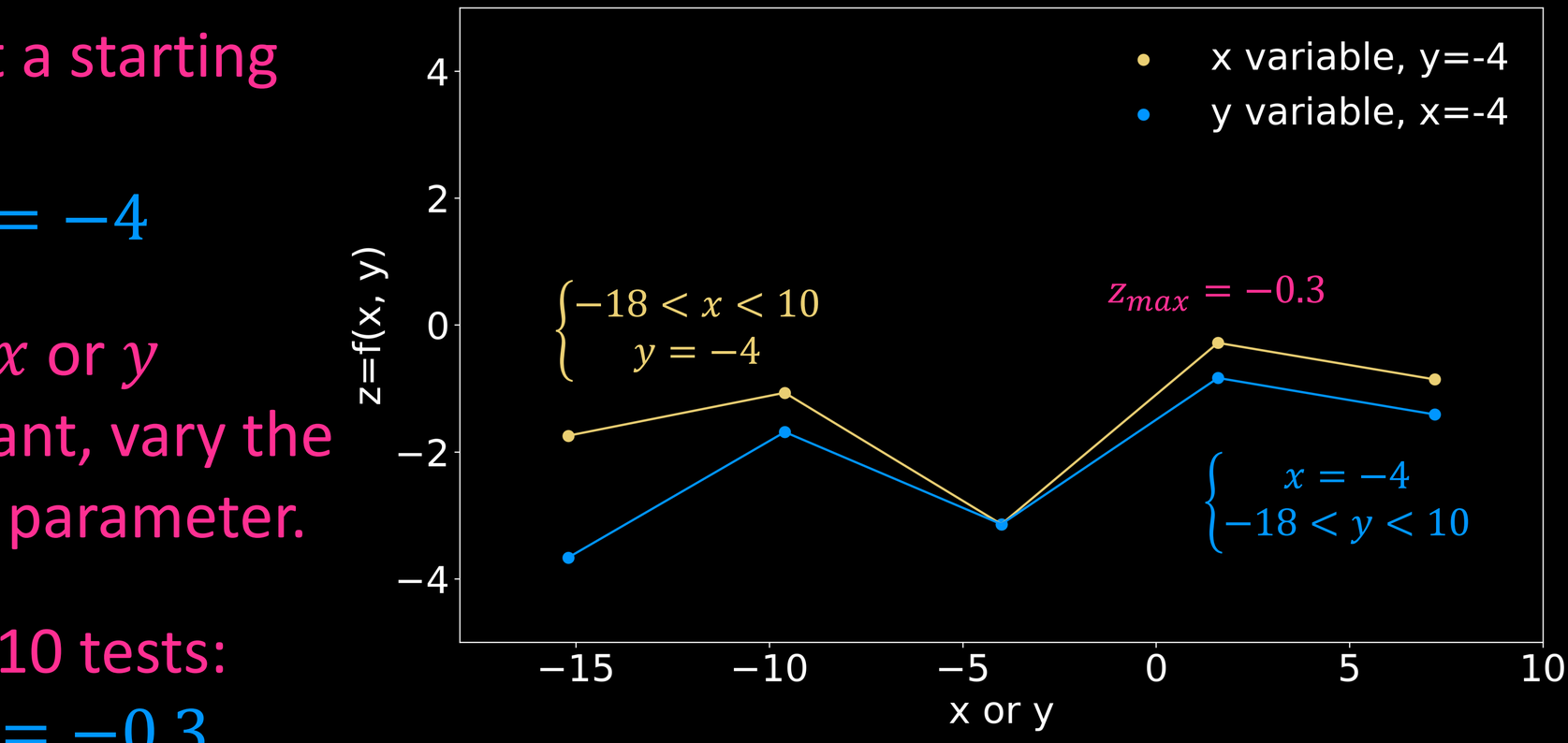
- Our goal is to identify parameter values that increase performance beyond a specified threshold:

$$Z > 1.75 \rightarrow x, y = ?$$

intro/ML_intro.py

OFAT: One-factor-at-a-time

1. Select a starting point:
   $x, y = -4$
2. Keep x or y constant, vary the other parameter.
3. After 10 tests:
   $z_{max} = -0.3$



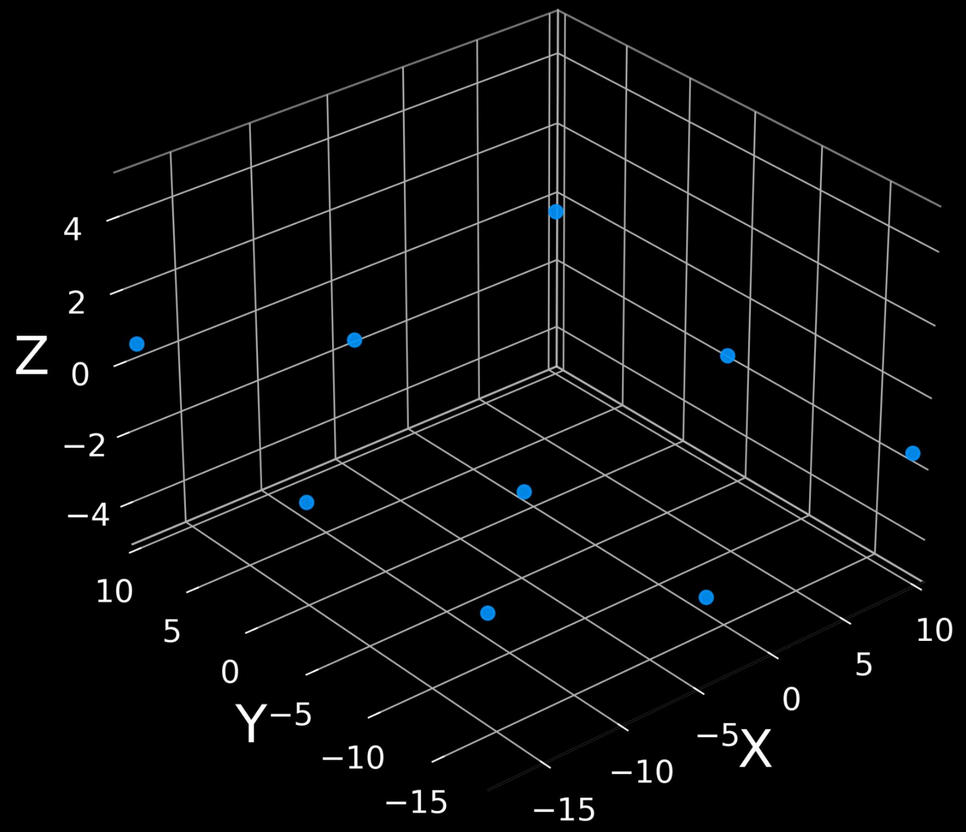
intro/ML_intro.py

DOE: Design of Experiments

1. Following the central composite design (CCD), we will select 9 points at the corners and centers of our domain.

2. Upon evaluating these 9 points:

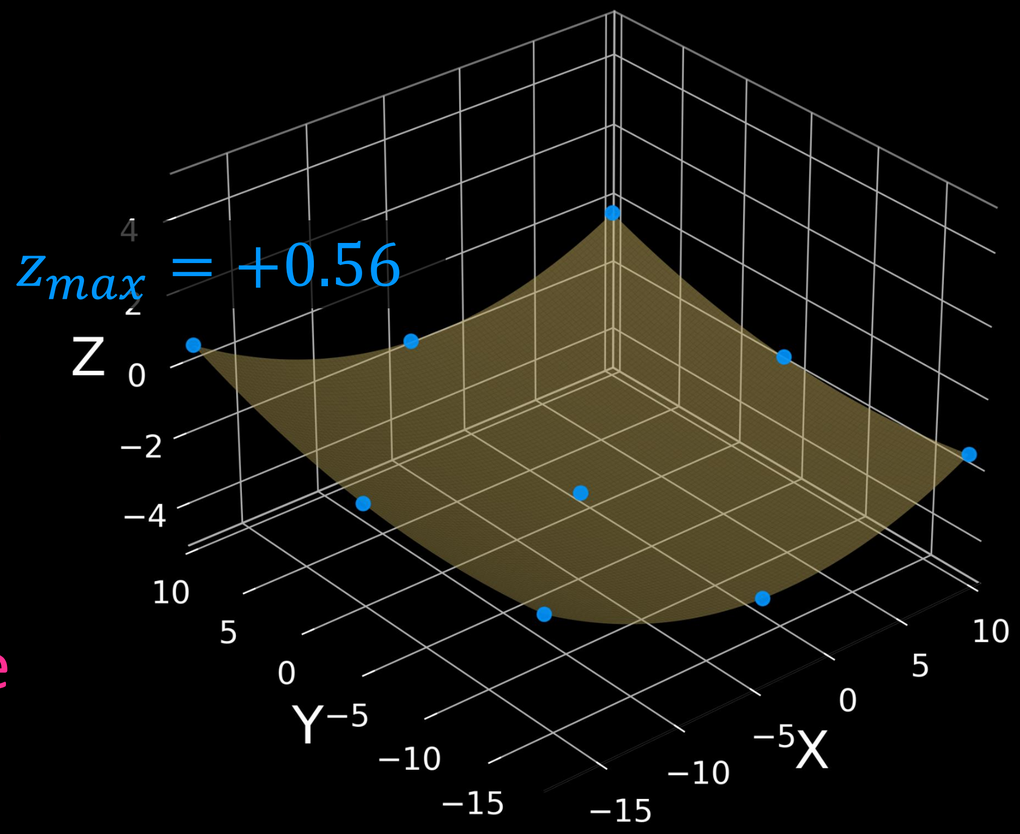
$$z_{max} = +0.56$$



intro/ML_intro.py

DOE: Design of Experiments

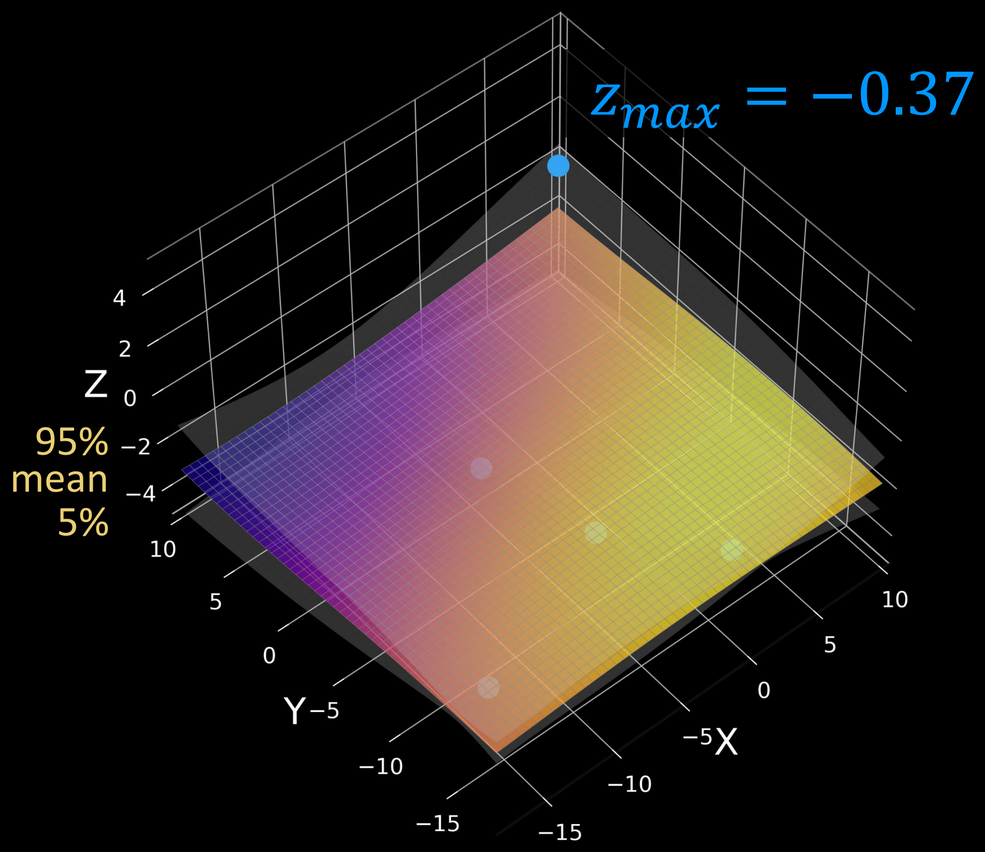
3. We follow the response surface methodology (RSM) to fit a second order polynomial.
4. This does not change the maximum value.
5. Next step? Further divide the domain into smaller zones?



intro/ML_intro.py

Machine Learning: an Iterative Fitting Approach

1. Initialization: Select 4-5 random points for evaluation.
2. Training: Use Gaussian Process Regression (GPR), a probabilistic ML method, to fit a response surface and predict the mean and confidence intervals from limited data.



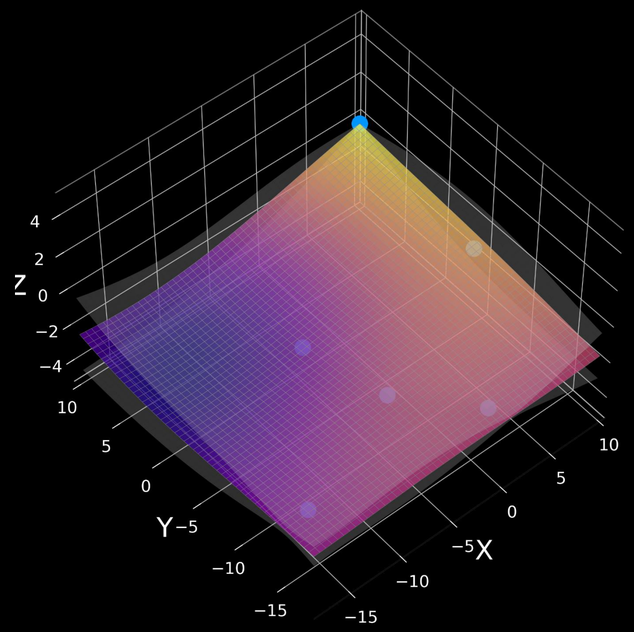
intro/ML_intro.py

Machine Learning: an Iterative Fitting Approach

3. Afterward, the GPR algorithm iteratively learns the response surface, aiming to locate optimal points at each step.

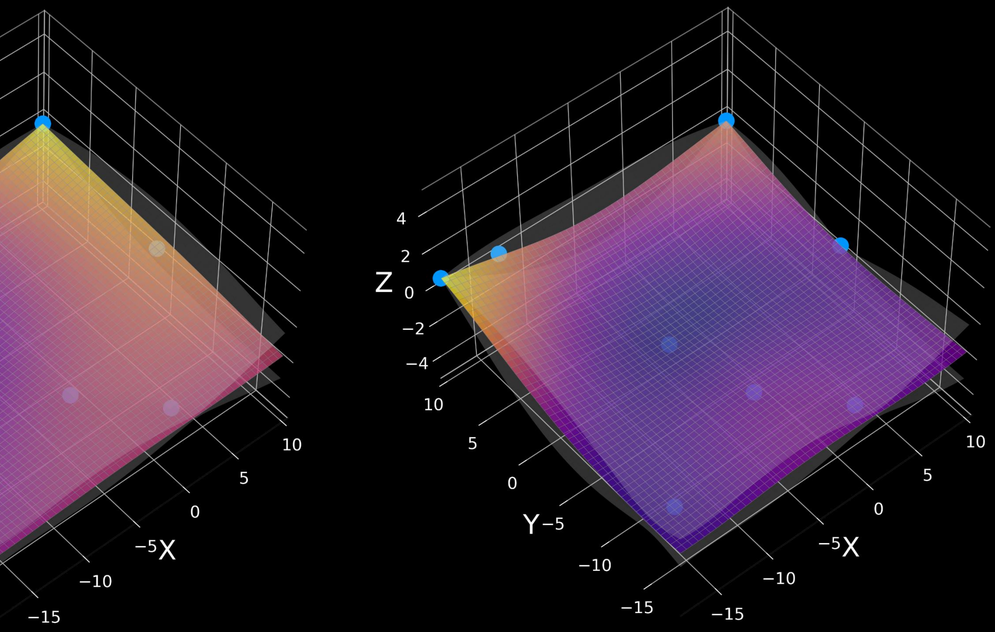
Iteration #6

$z_{max} = -0.37$



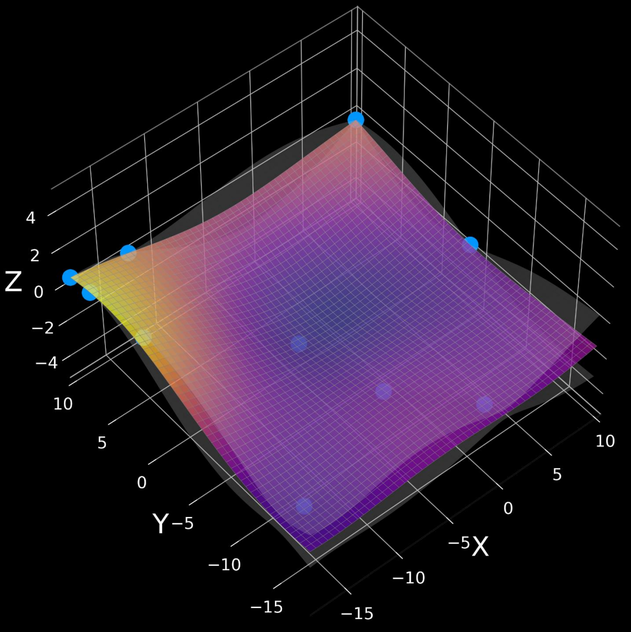
Iteration #8

$z_{max} = 0.56$



Iteration #10

$z_{max} = 1.17$



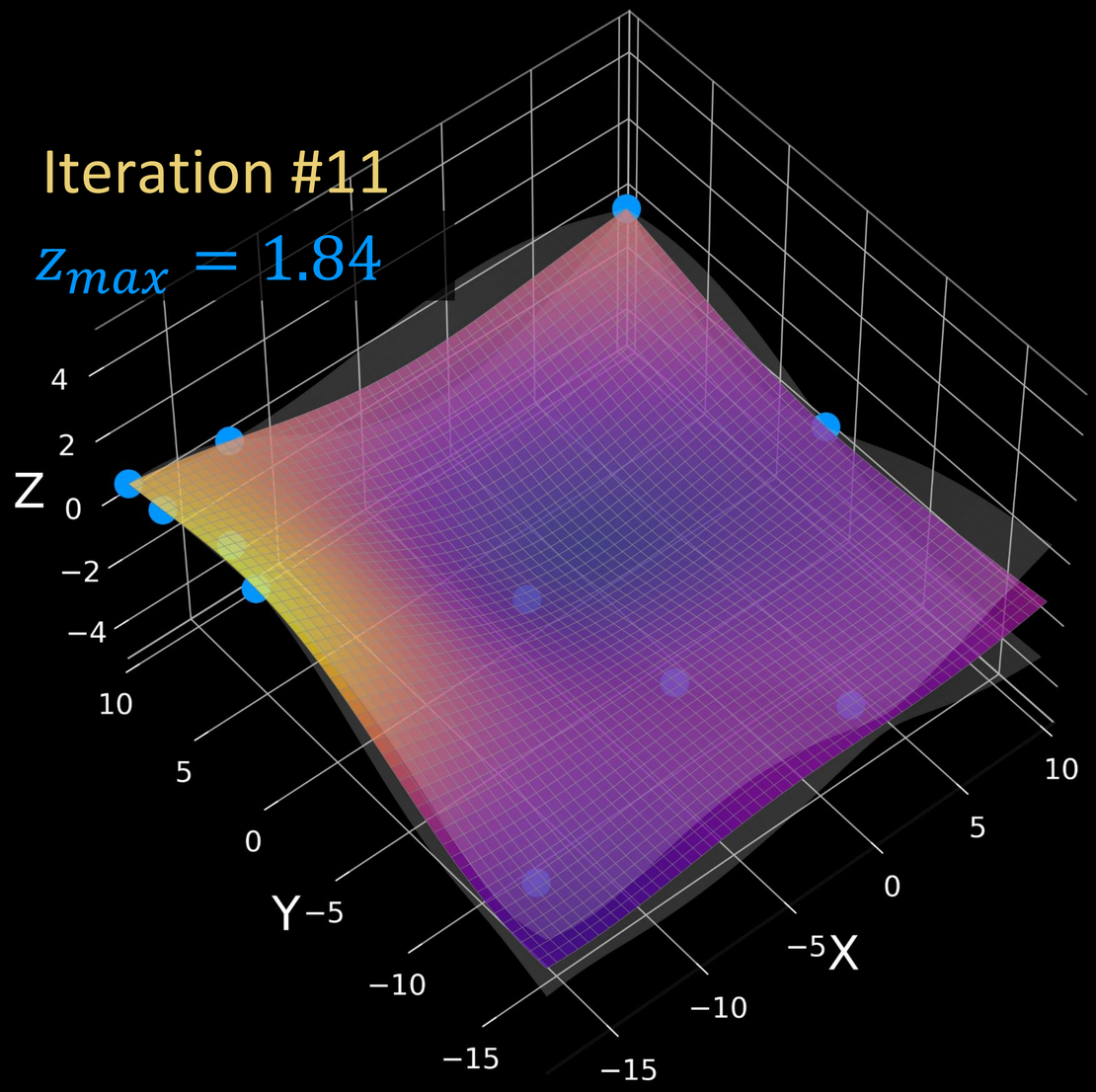
Machine Learning: an Iterative Fitting Approach

4. After 11 iterations, it finds a location.

- Depending on initialization and model assumptions, the number of iterations to find a solution can vary.

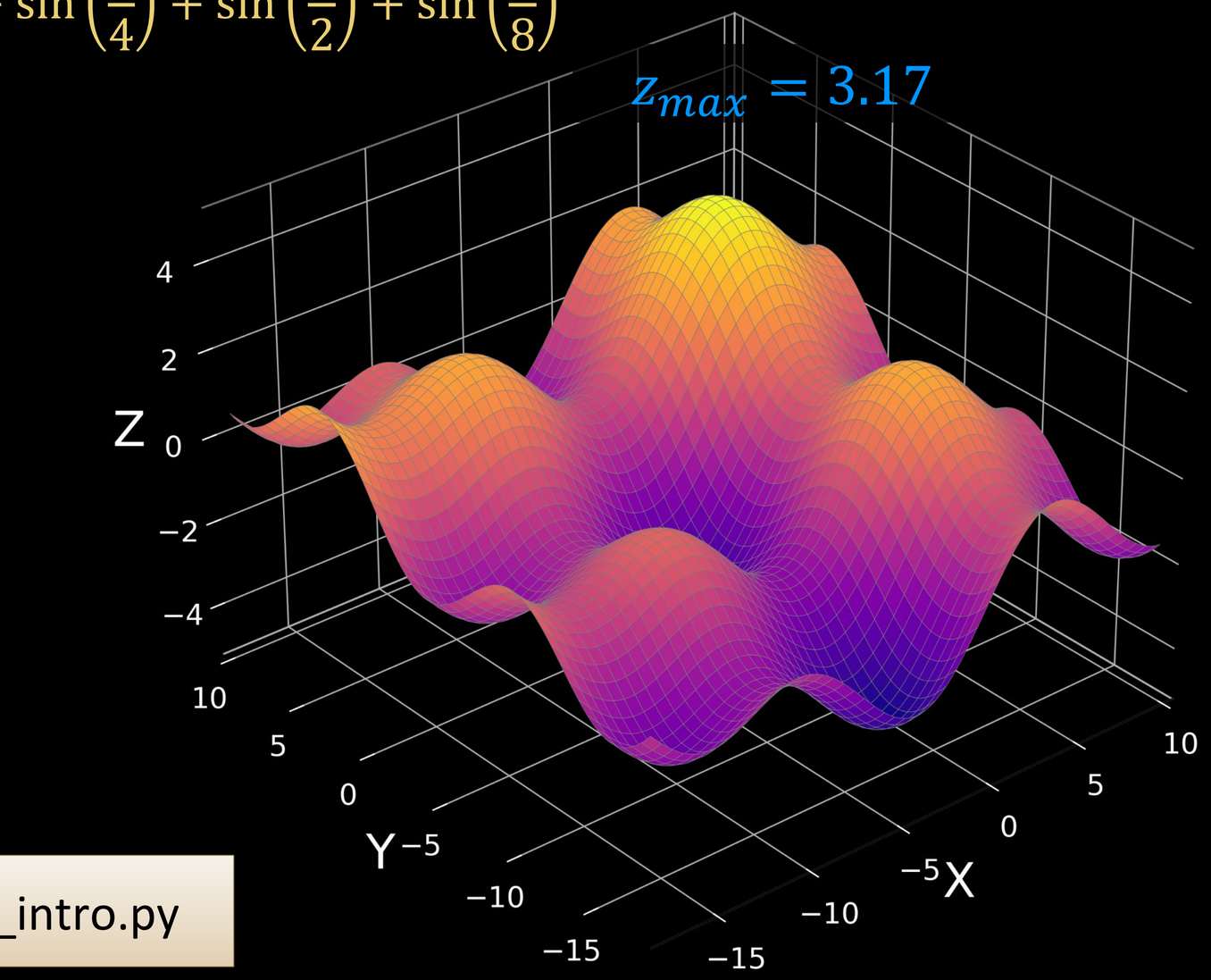
- This is a theory-agnostic ML approach

intro/ML_intro.py



True Hidden Response

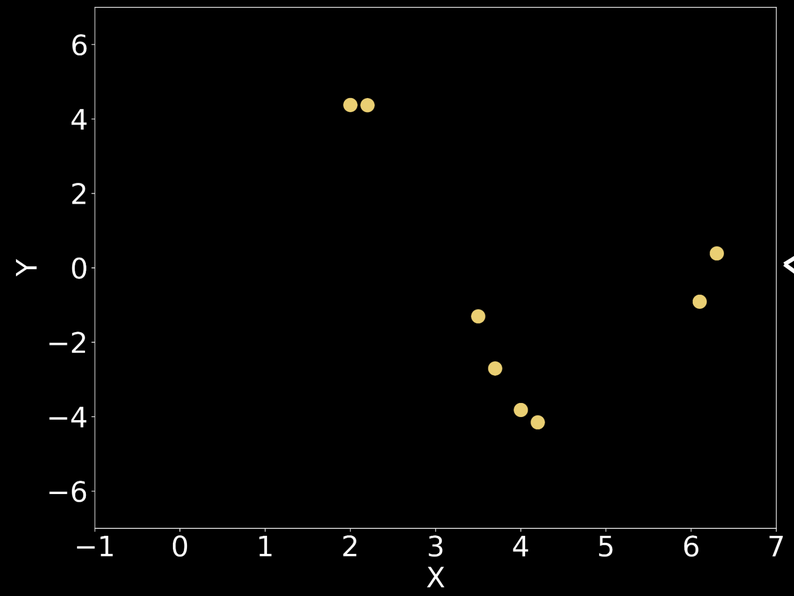
$$Z = \sin\left(\frac{x}{2}\right) + \sin\left(\frac{x}{4}\right) + \sin\left(\frac{y}{2}\right) + \sin\left(\frac{y}{8}\right)$$



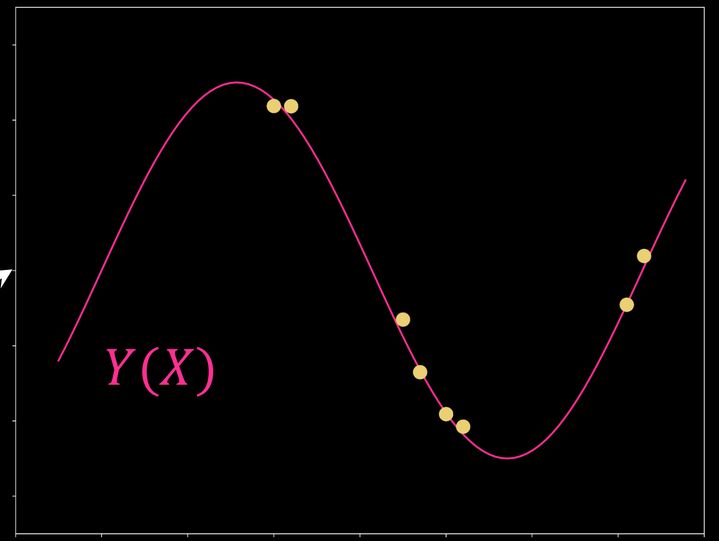
intro/ML_intro.py

Machine Learning: A Regression Perspective

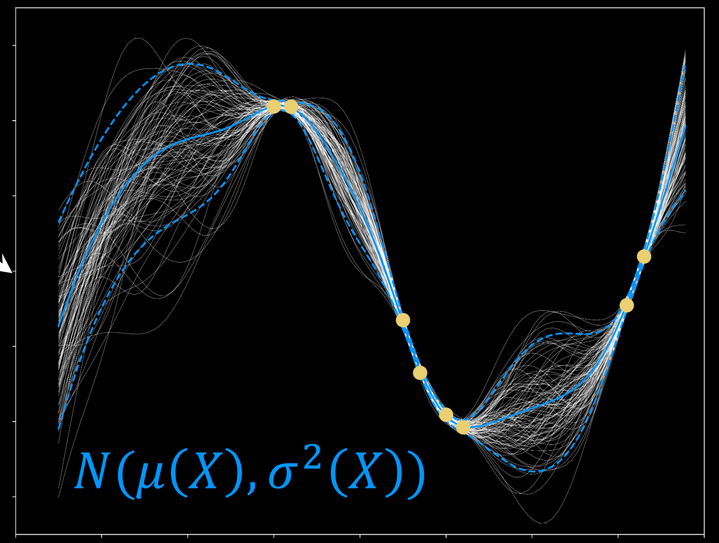
Data



Deterministic Regression (e.g., Neural Networks)

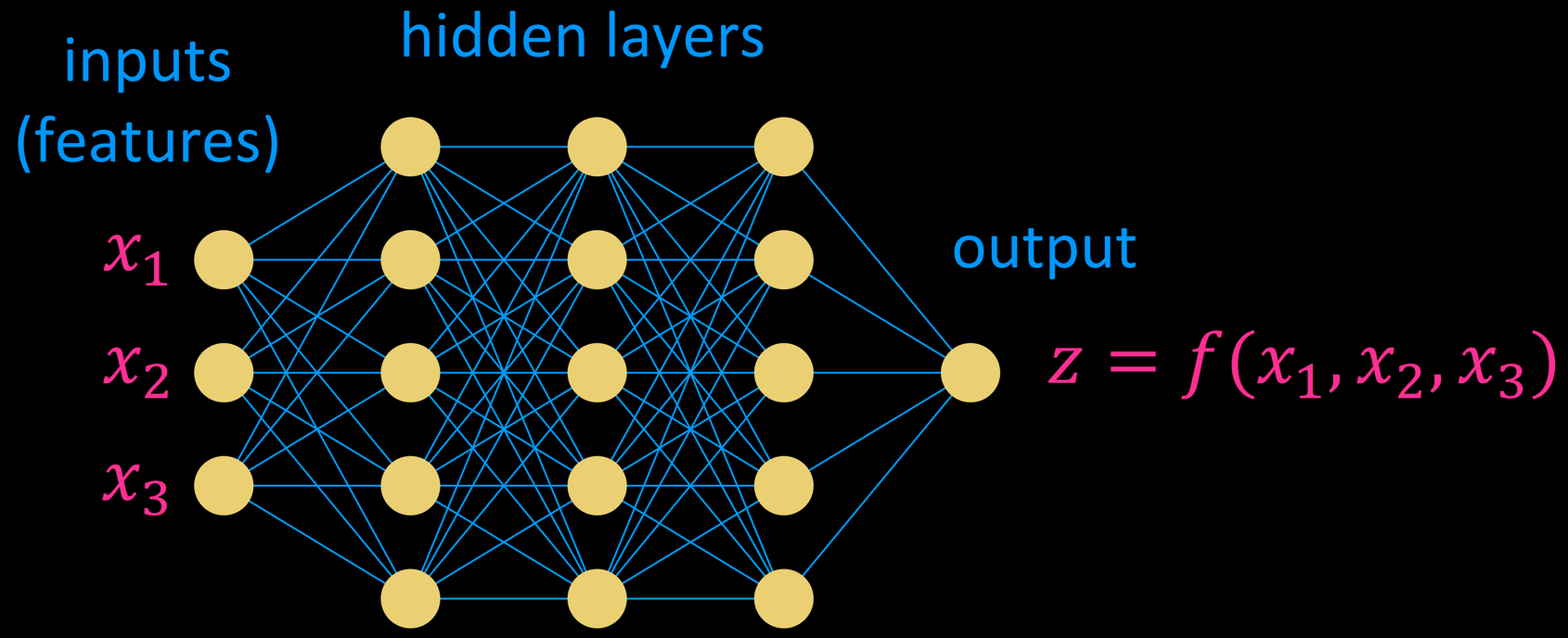


Probabilistic Regression (e.g., GPR)

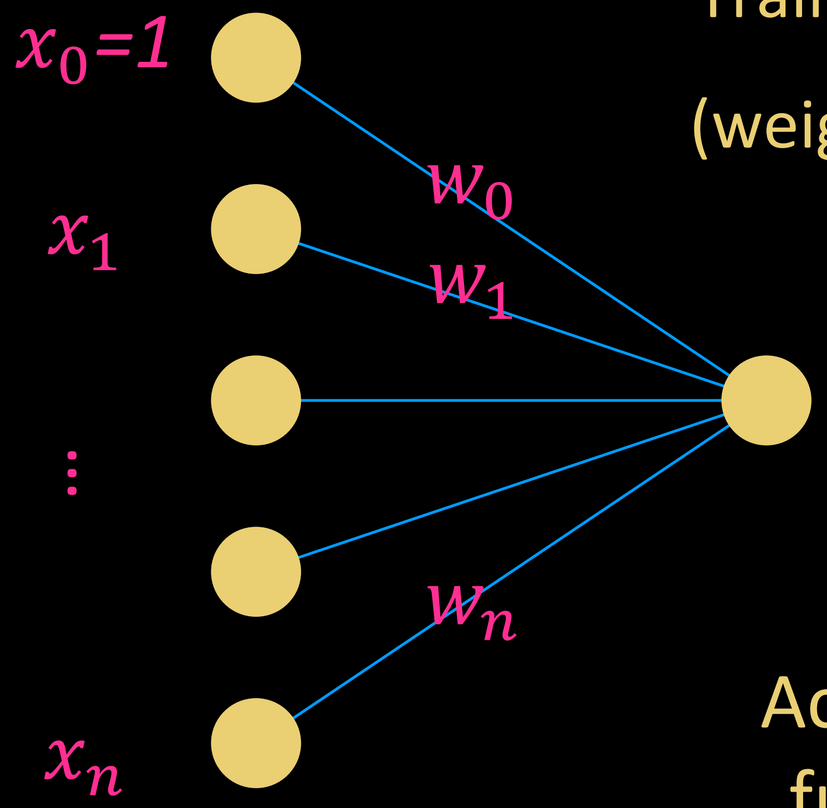


Introduction: Neural Network (NN)

- **Basically, a complex closed-form equation to fit high dimensional dataset**



Basic Calculations in a Layer



Training means fitting the constants (weights and biases) of the equation

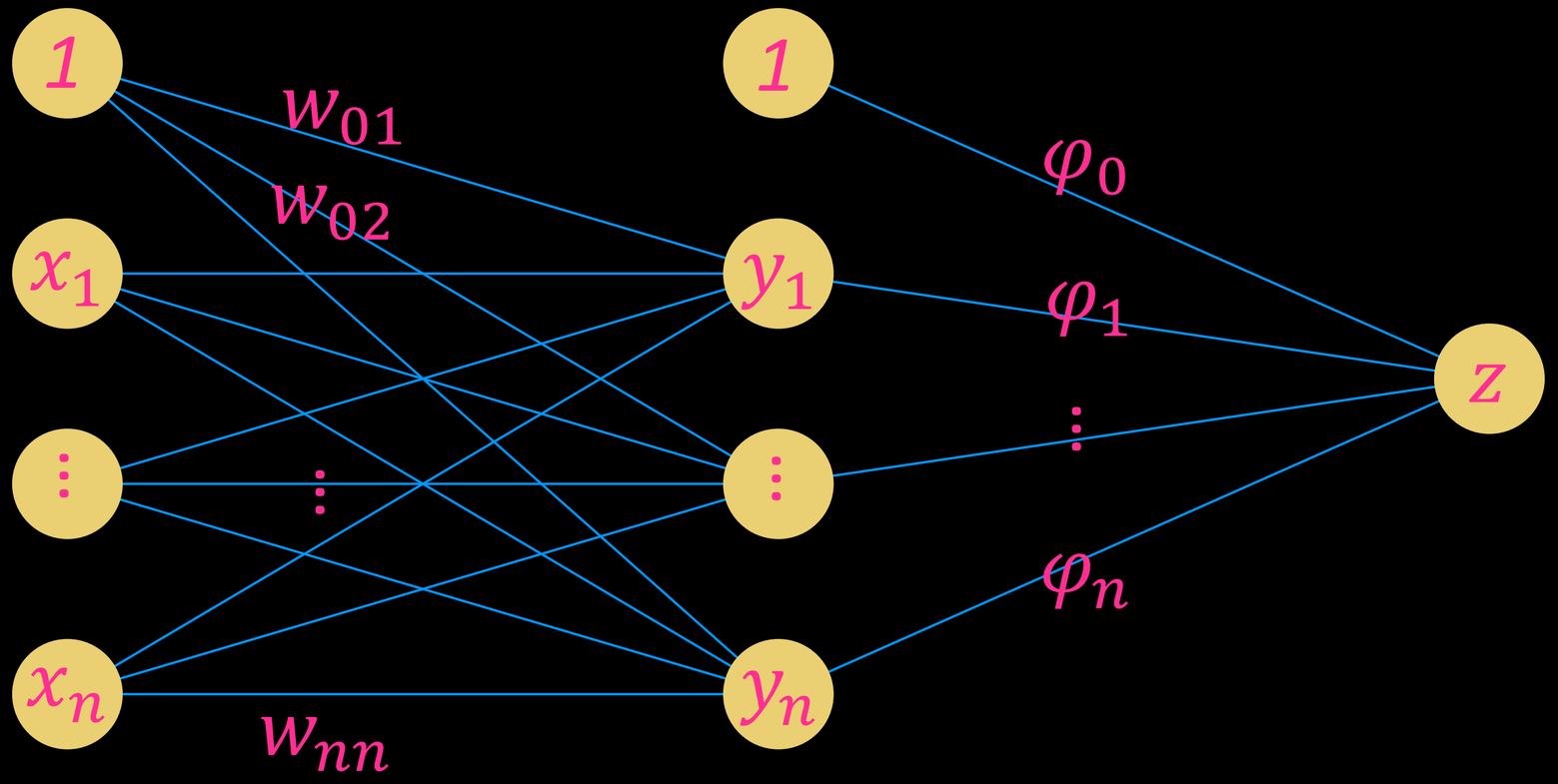
$$z = f\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

Activation function

bias

weights

Building a NN model

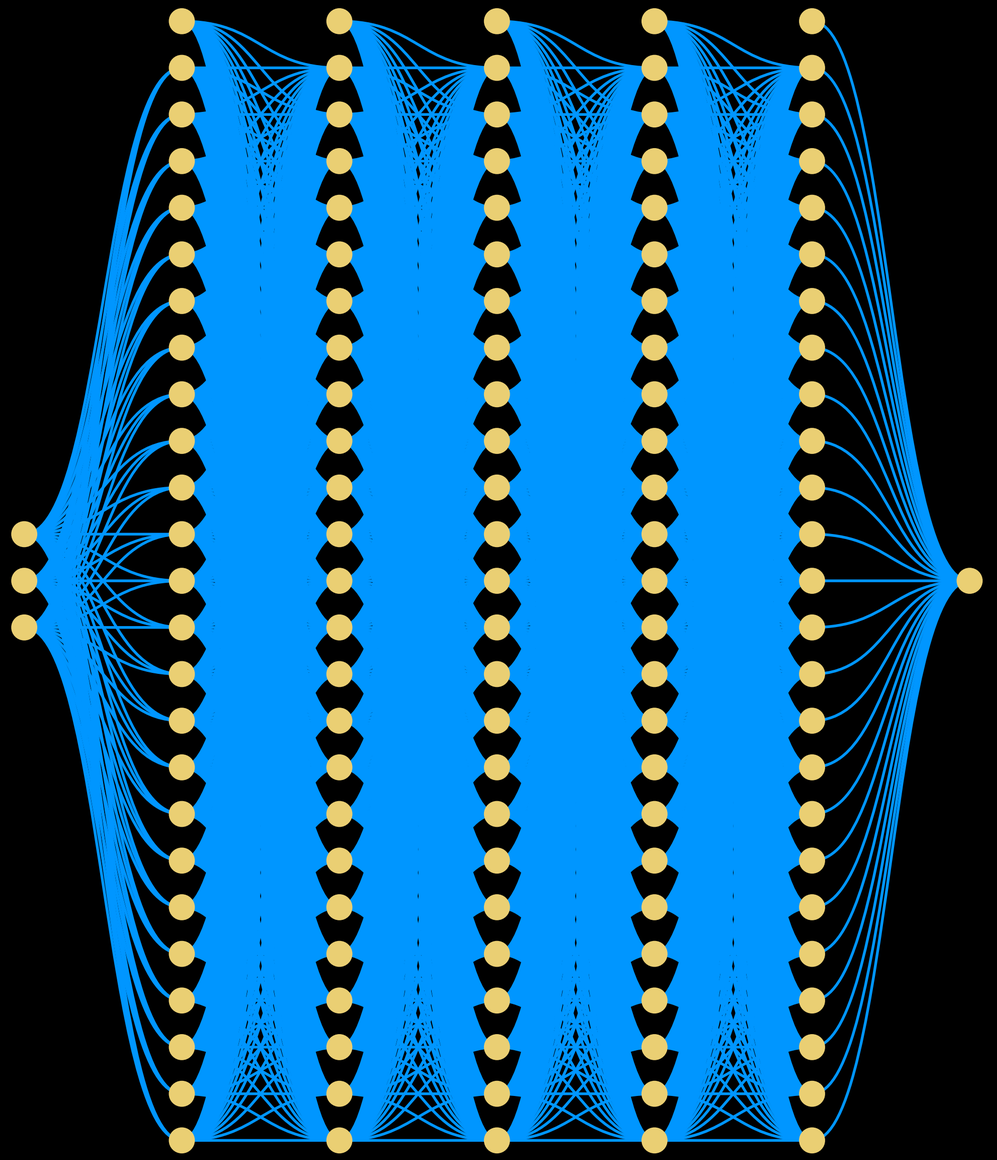


$$z = f\left(\varphi_0 + \sum_{i=1}^{m} \varphi_i y_i\right) = f\left(\varphi_0 + \sum_{i=1}^{m} \varphi_i f\left(w_{0i} + \sum_{j=1}^{n} w_{ji} x_i\right)\right)$$

Building a Deep NN model

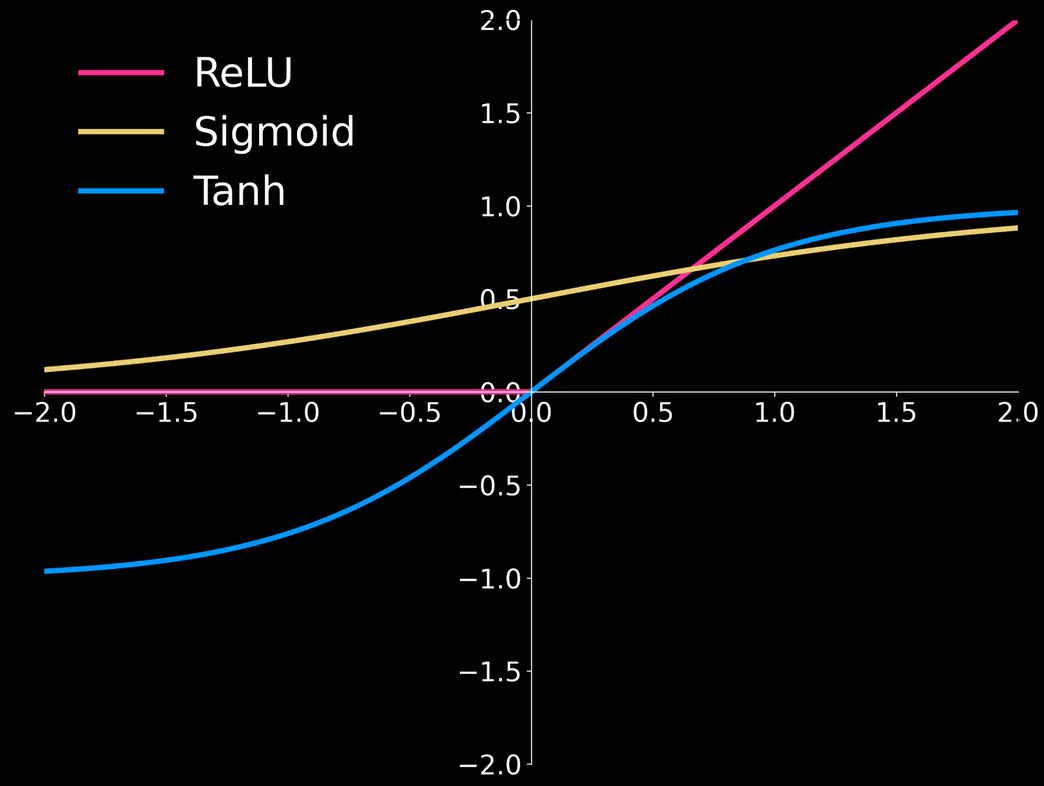
- **Model Summary**
  - 3 inputs
  - 5 hidden layers
  - 25 nodes/hidden layer
  - 1 output
- **Constants to fit**
  - 2,600 weights
  - 101 biases
  - Total: 2,701
- **Model equation**

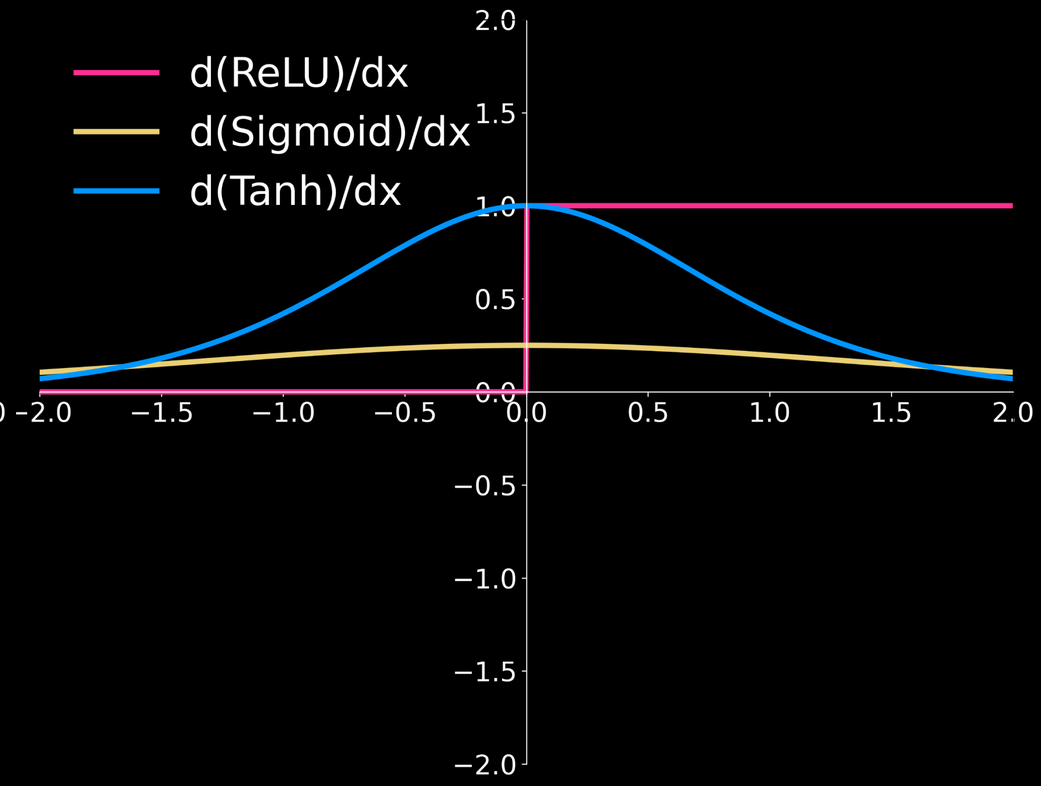
$$z = f\left(f\left(f\left(f\left(f\left(x_1, x_2, x_3\right)\right)\right)\right)\right)$$



Standard Activation Functions

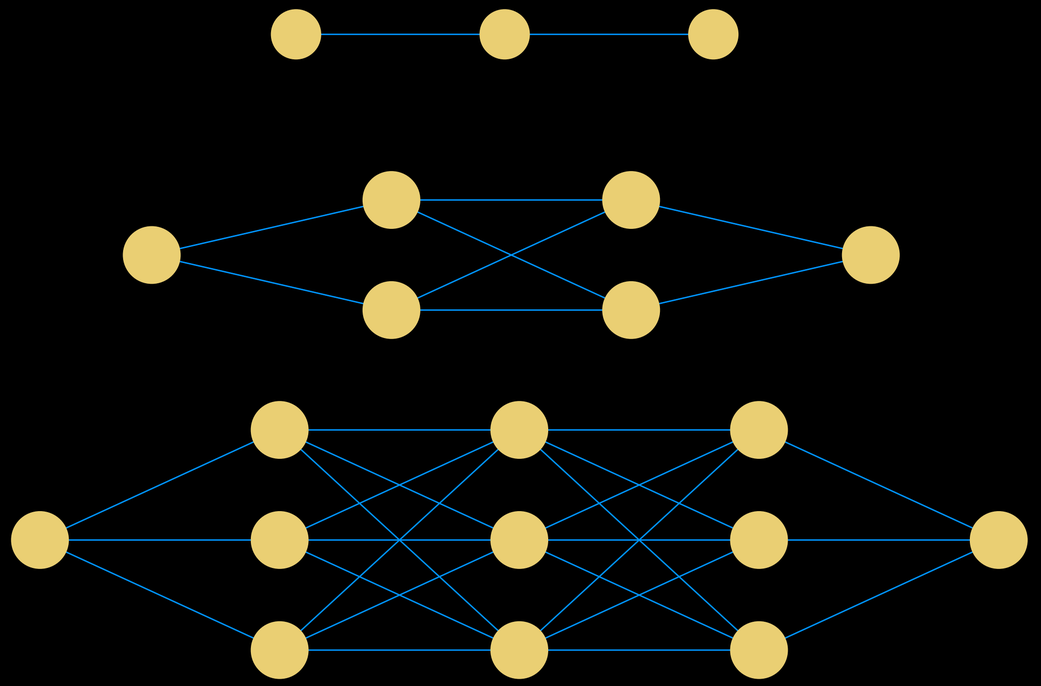
$$ReLU(z) = \max(0, z) \qquad \text{Tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad Sigmoid(z) = \frac{1}{1 + e^{-z}}$$

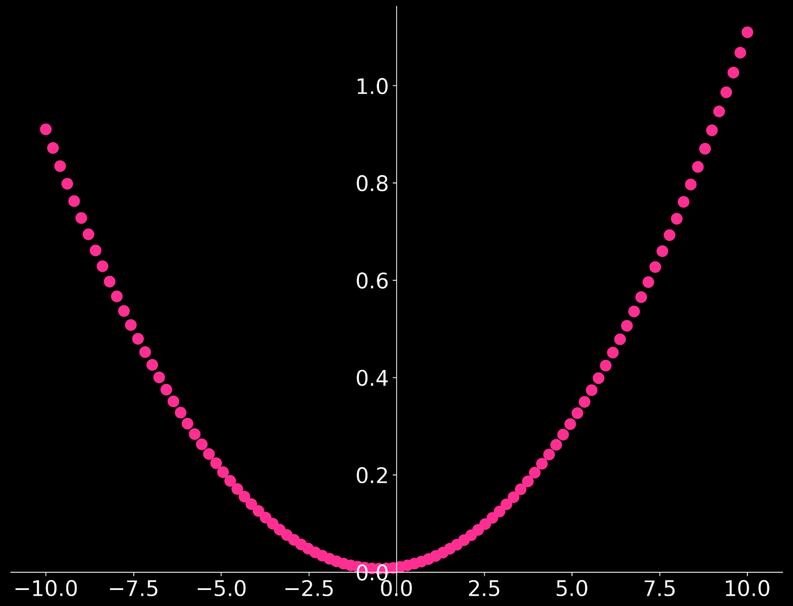




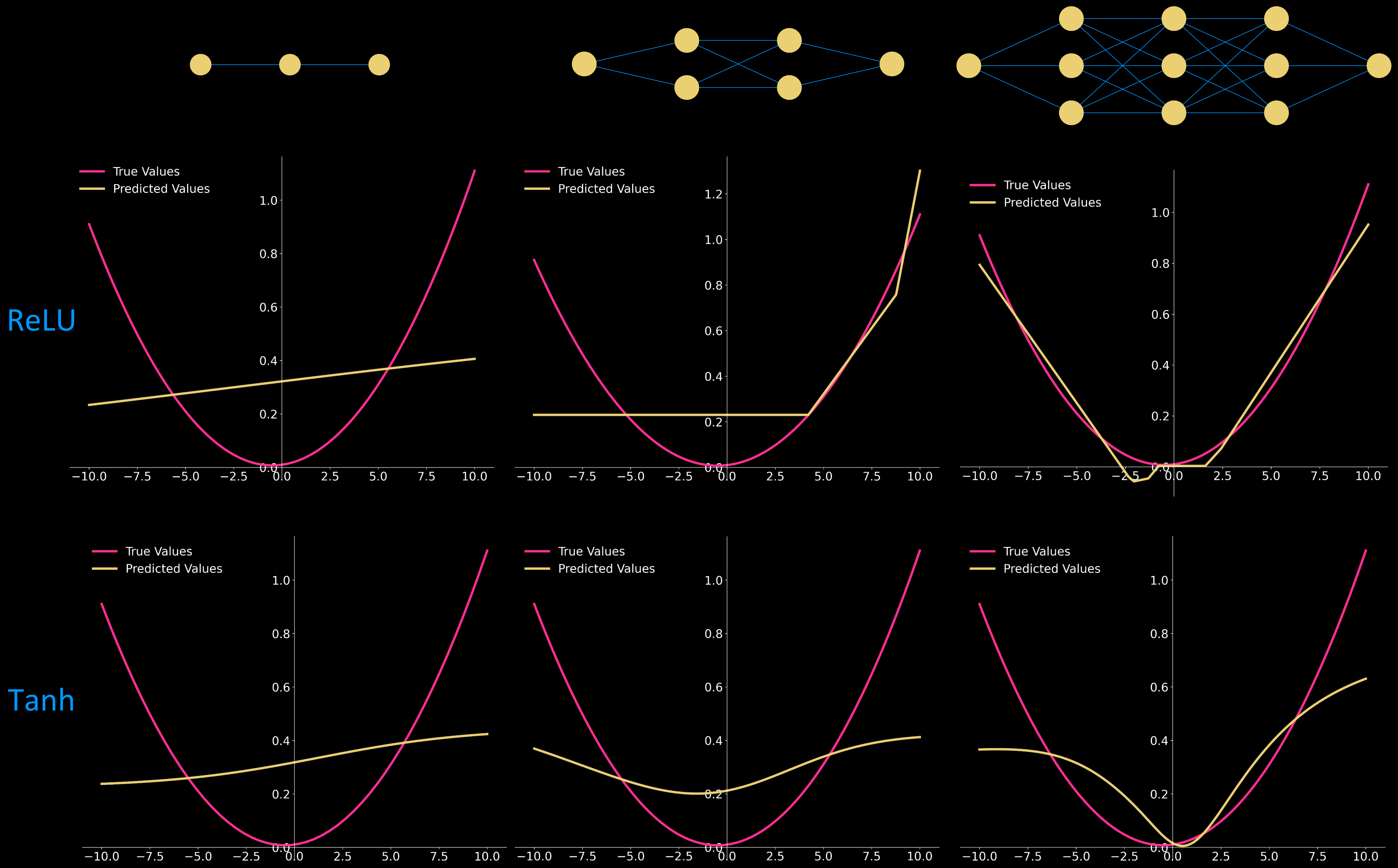
NN: A Simple Example

- We will train Neural Networks with different architectures to estimate a quadratic function:



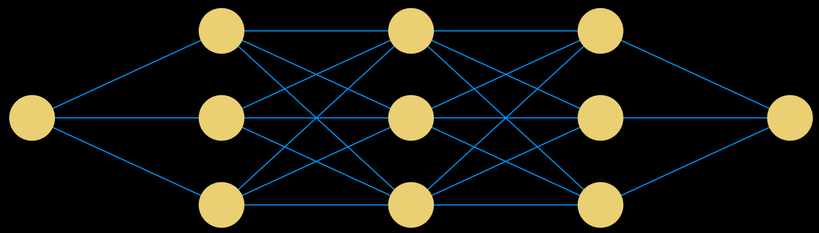


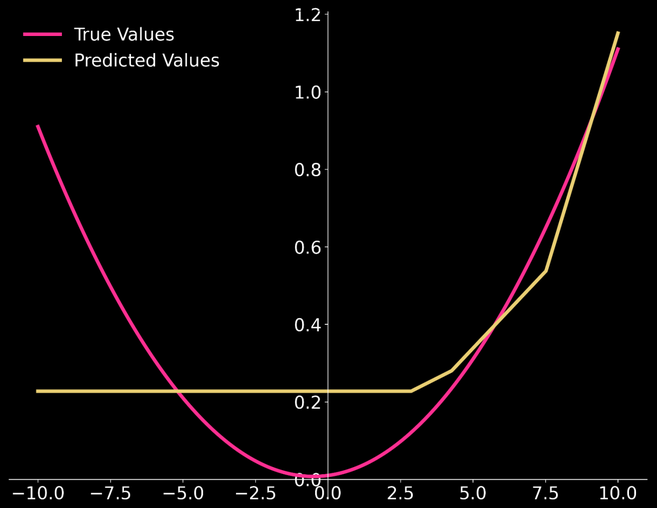
intro/NN_intro.py

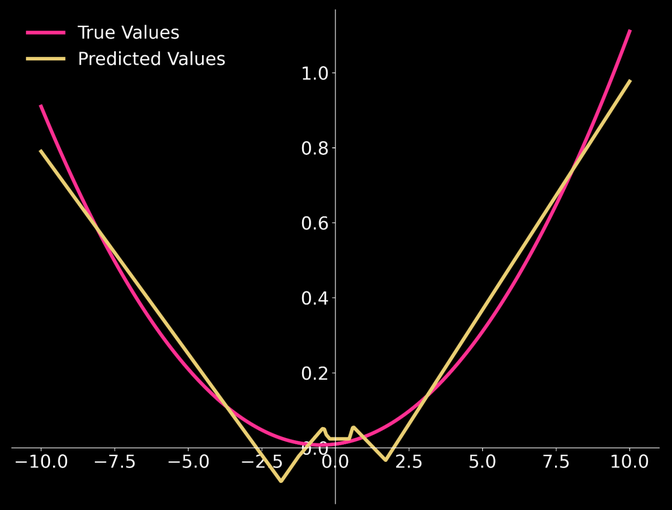

ReLU
Tanh
True Values
Predicted Values

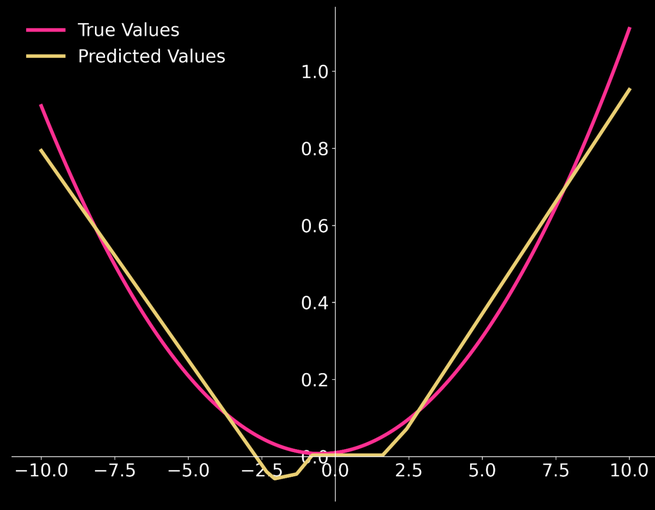
Non-Unique Solution?

- If we repeat the trainings, each time we might get a different solution. E.g., 3×3 architecture and ReLU function:



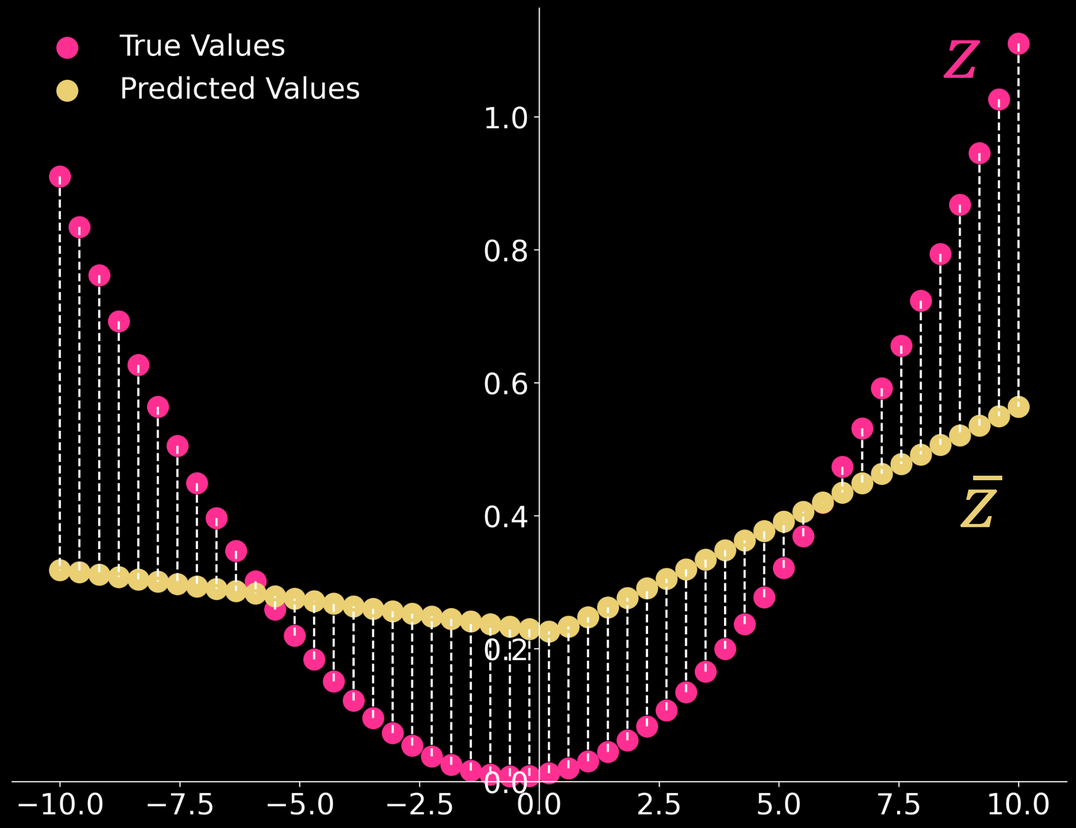






Training a NN: Loss

- To train a NN model, we need to minimize a cost (loss) function such as Root Mean Squared Error (RMSE):



$$RMSE = \sqrt{\frac{\sum(\bar{z}_i - z_i)^2}{N}}$$

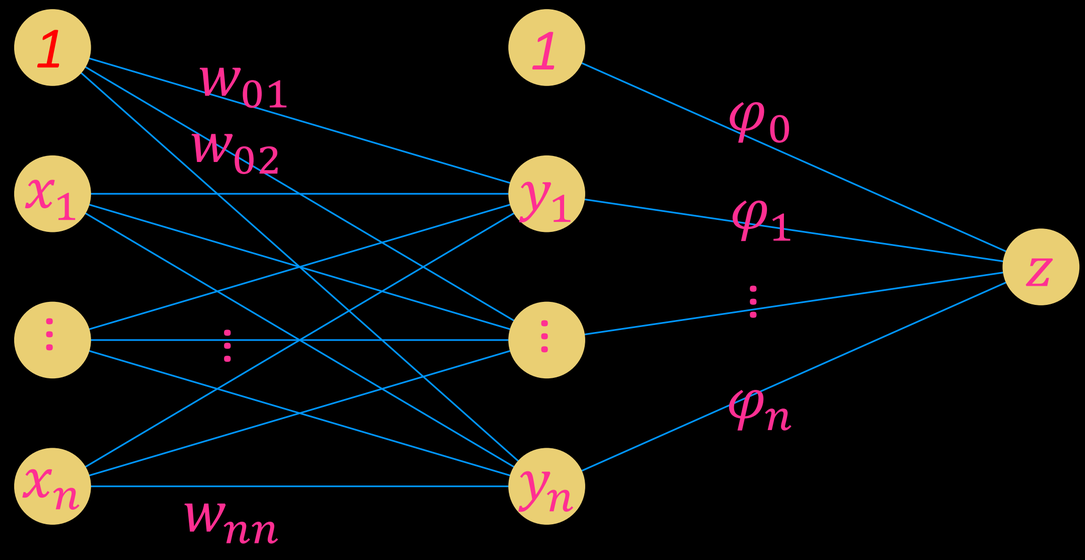
Backpropagation and Optimization Algorithms

1. In each step, calculate the derivative of the loss with respect to the weights using the backpropagation algorithm:

$$\frac{\partial Loss}{\partial w_{11}} = \frac{\partial Loss}{\partial z} \times \frac{\partial f}{\partial \Sigma_2} \times \varphi_1 \times \frac{\partial f}{\partial \Sigma_1} \times x_1$$

2. Then, adjust the weights using an optimization algorithm such as gradient descent (GD).

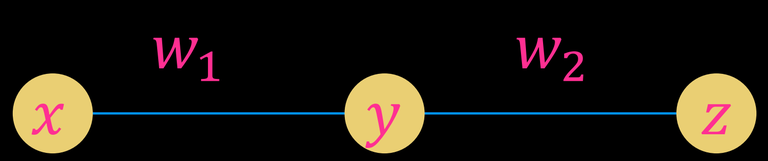
$$w_{11}^{k+1} = w_{11}^{k} - \eta \frac{\partial Loss}{\partial w_{11}^{k}}$$



$$z = f\left(\varphi_0 + \sum_{i=1}^{m} \varphi_i f\left(w_{0i} + \sum_{j=1}^{n} w_{ji} x_i\right)\right)$$

Why Non-Unique Solution?

- Consider a simple neural network:
  - 1 feature, 1 hidden node, 1 output
  - 2 weights to fit
  - Activation function: tanh
  - Loss function: mean squared error
  - 1 data point to fit: (0.5, 0.5)



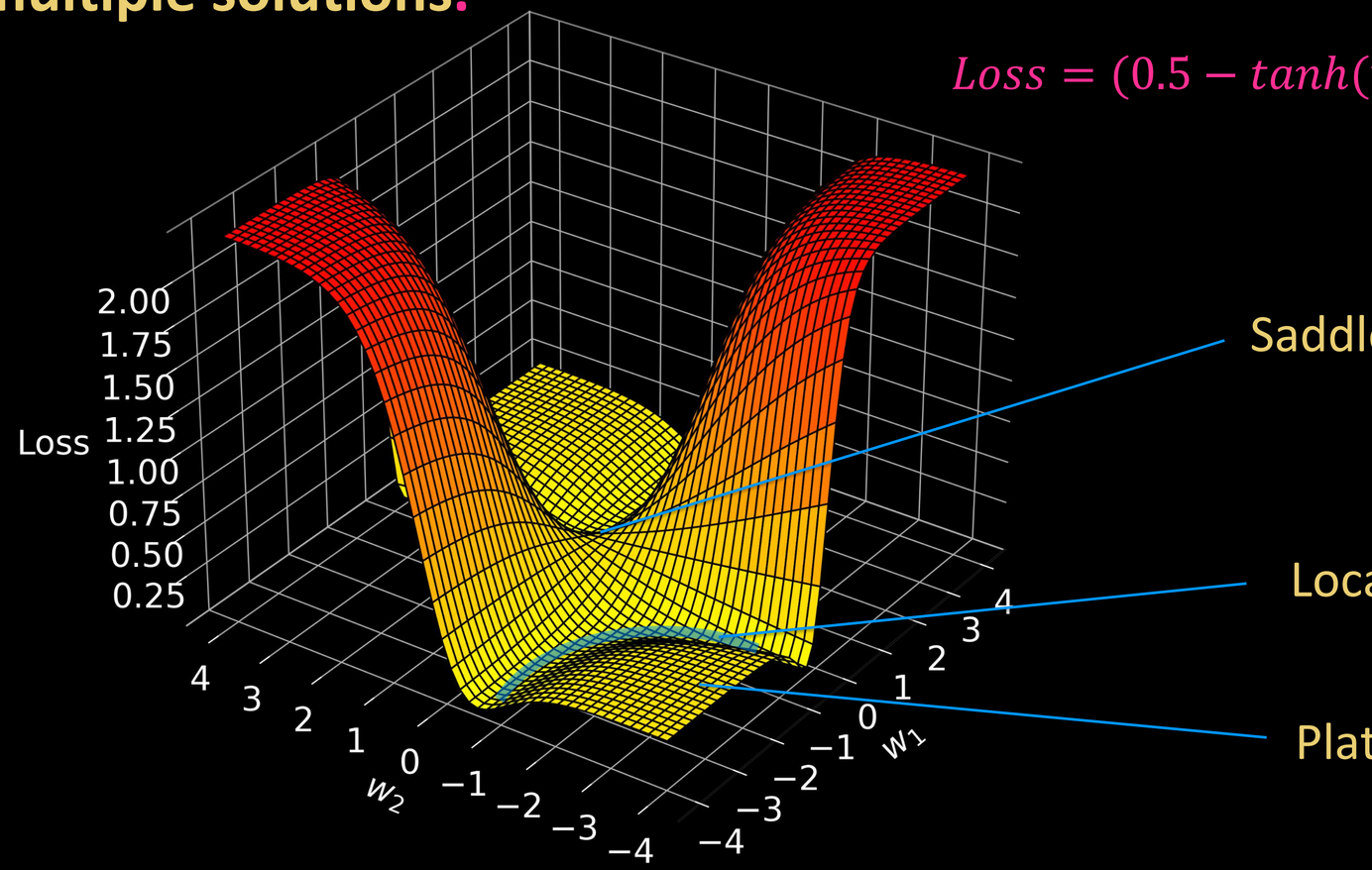
$$z = \tanh(w_2 y) = tanh(w_2\, tanh(w_1 x)$$

$$Loss = (0.5 - tanh(w_2\, tanh(0.5 w_1))^{\,2}$$

Why Non-Unique Solution?

- **The loss function often has many local minima near the global minimum. Weight initialization and the choice of optimization algorithm significantly impact the outcome, potentially leading to multiple solutions.**

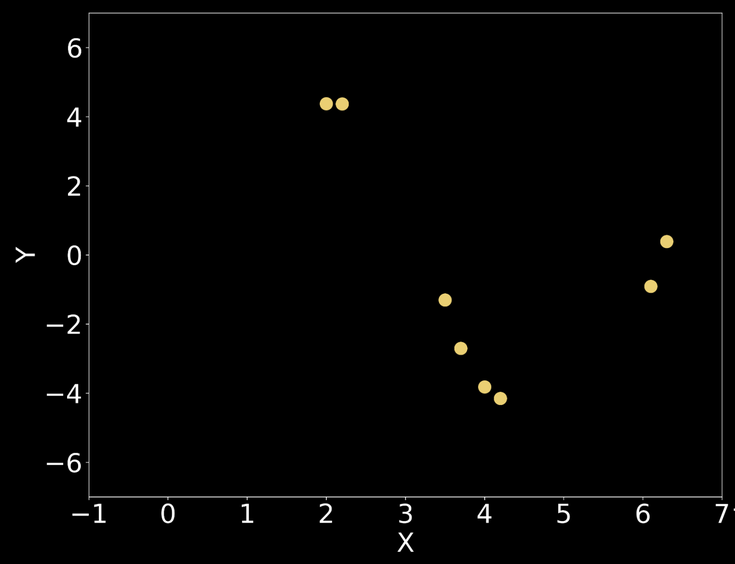
$$Loss = (0.5 - tanh(w_2\, tanh(0.5w_1))\,)^{\,2}$$



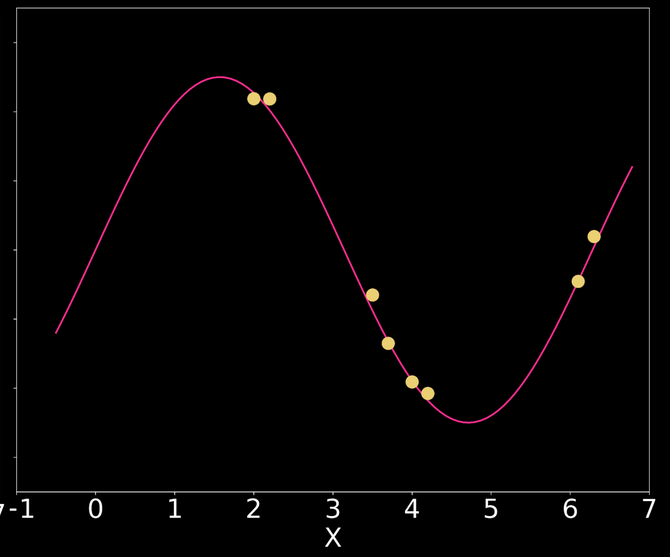
Introduction: Gaussian Process Regression (GPR)

- Consider a small dataset. GPR is a probabilistic machine learning approach that can identify the zone of the highest likelihood for the true response.

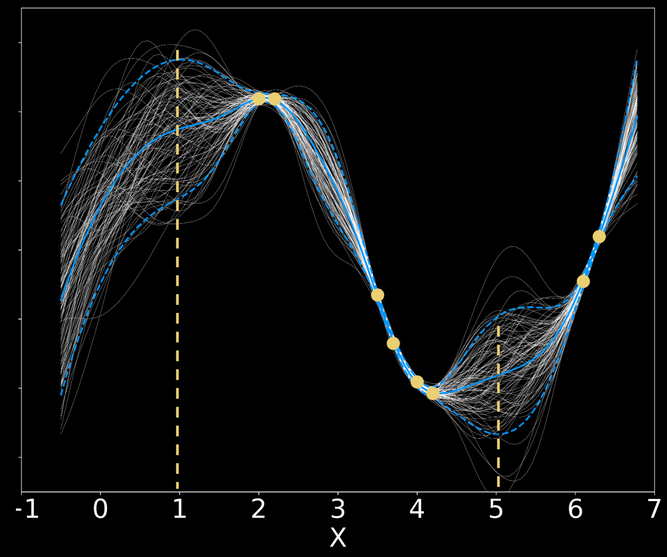
Small experimental data



Hidden True response



90% confidence interval



Potential Maximum

Potential Minimum

Introduction: Gaussian Process Regression (GPR)

- Given *n* examples of a function $D_n = (X_n, Y_n)$, what random functions could explain those observed values?

$$Y(X)|D_n$$

- We assume that any observation is modeled having a multivariate normal distribution and are completely described by their mean, μ, and covariance matrix,σ^2*:*

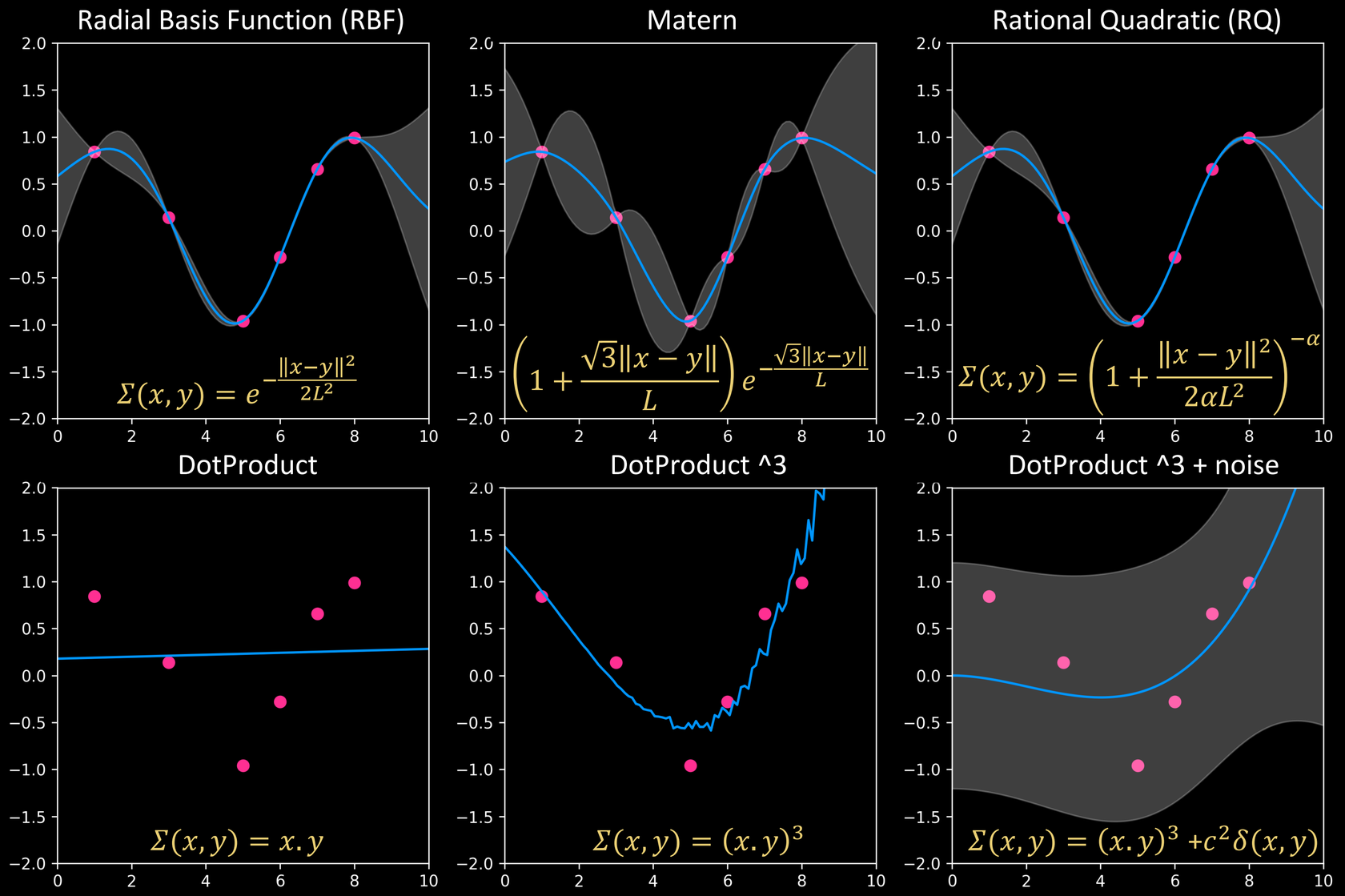
$$Y(x)|D_n \sim N(\mu(x), \sigma^2(x))$$

- We can show that for a selection of x values:

$$\mu(x) = \Sigma(x, \mathrm{X}_n) \times \Sigma(\mathrm{X}_n, \mathrm{X}_n)^{-1} \times Y_n$$ mean

$$\sigma^2(x) = \Sigma(x, x) - \Sigma(x, \mathrm{X}_n) \times \Sigma(\mathrm{X}_n, \mathrm{X}_n)^{-1} \times \Sigma(x, \mathrm{X}_n)^T$$ variance

Introduction: GPR Kernel

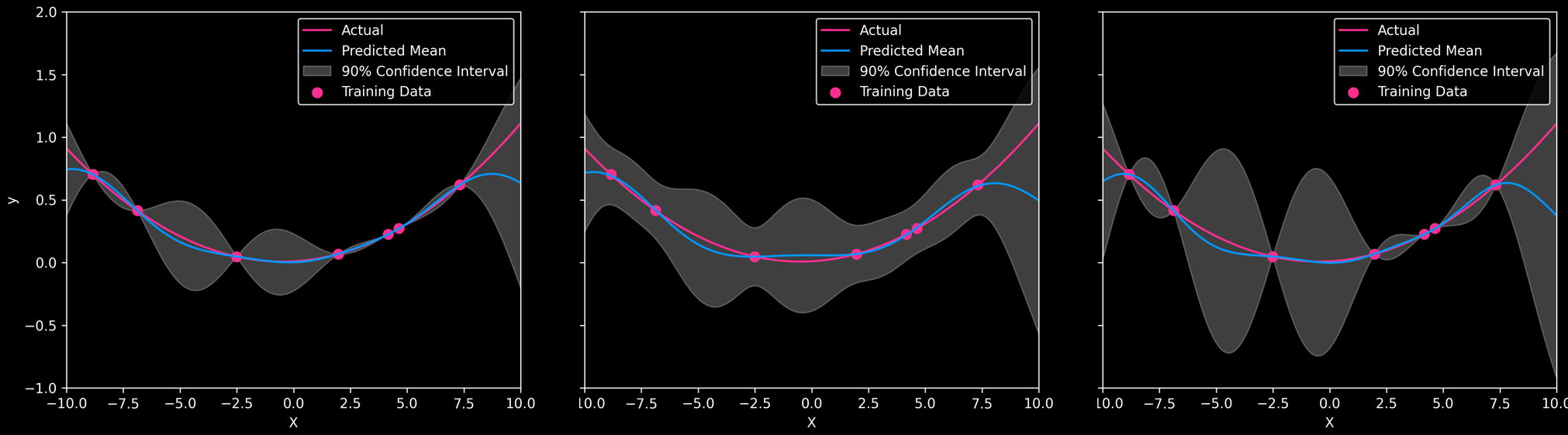


intro/GPR_kernel.py

Introduction: GPR Hyper-Parameters

$$\Sigma = RBF(L) + noise(c) \rightarrow \quad \Sigma(x,y) = e^{-\frac{\|x-y\|^2}{2L^2}} + c^2\delta(x,y)$$

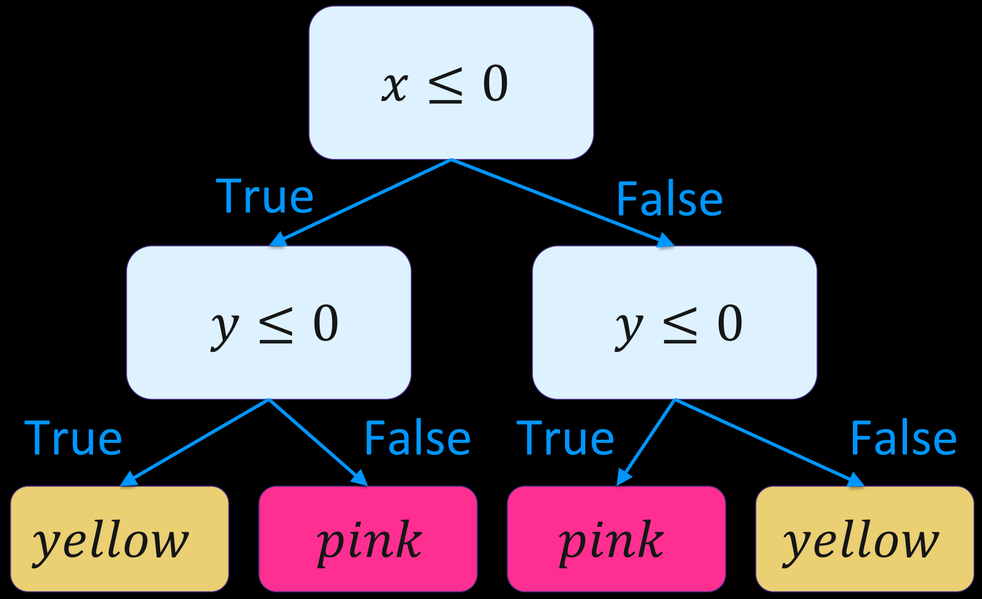
$L = 3, c = 0$ $L = 3, c = 0.01$ $L = 2.25, c = 0$

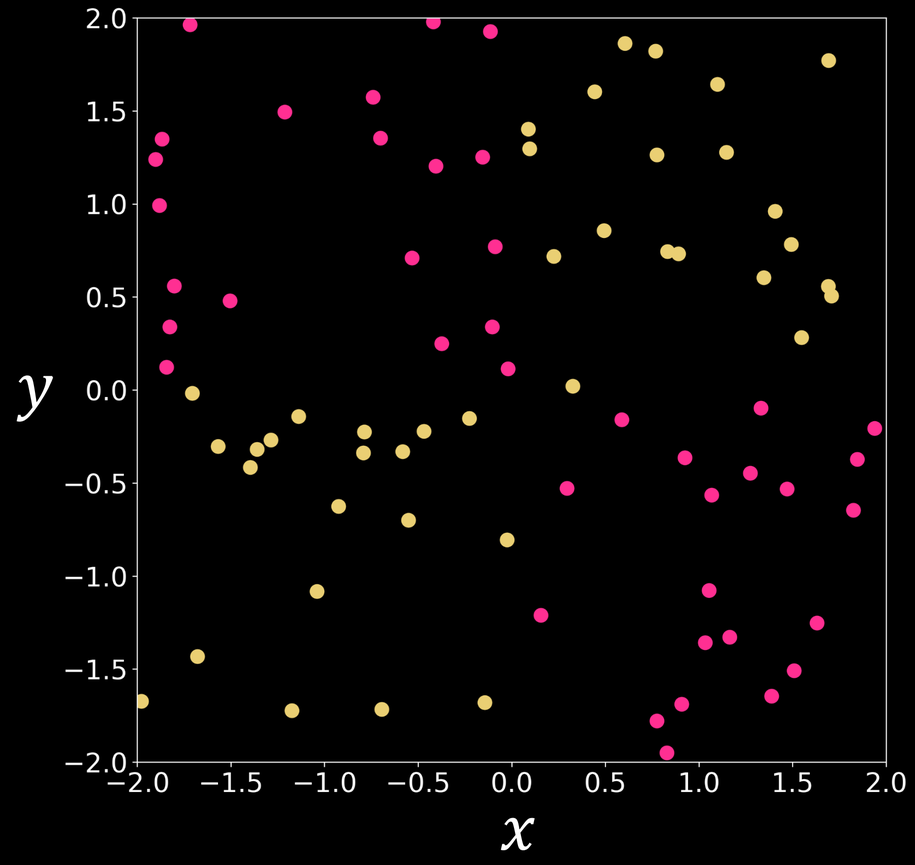


intro/GPR_intro.py

Introduction: Decision Tree

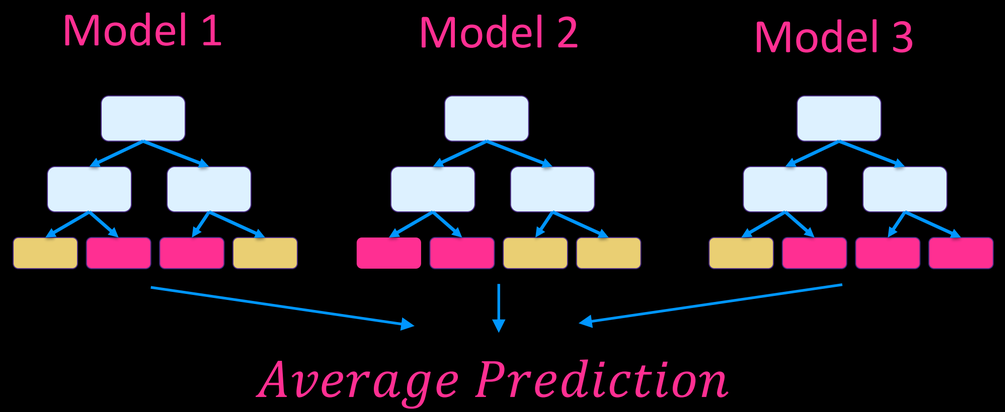
- How is the color of the circles determined by their coordinates?



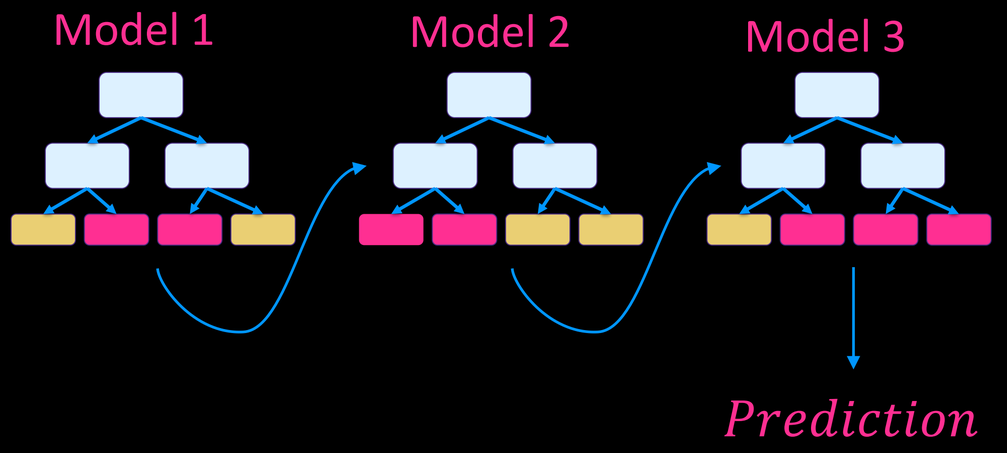


Introduction: Ensemble Methods

- **Bagging (Parallel)**
  - **Random Forest**

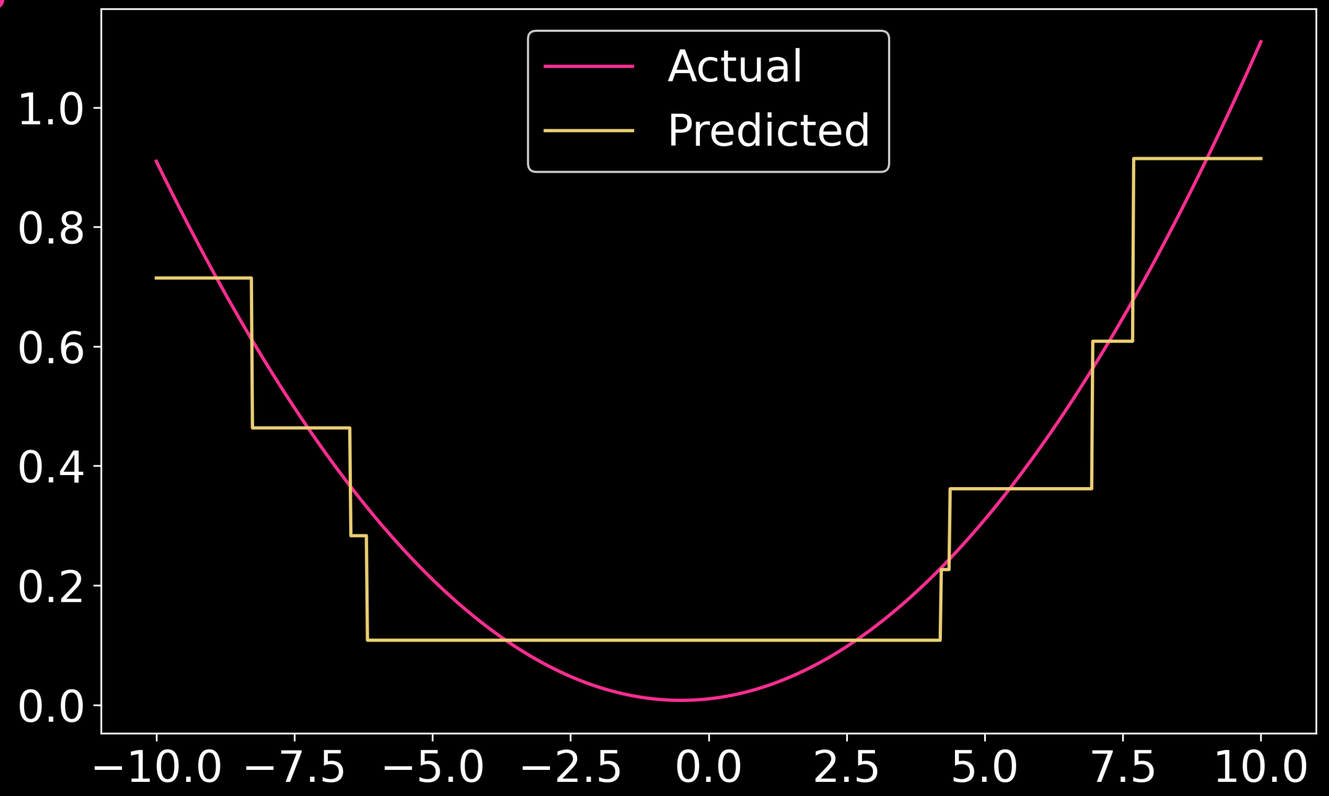


- **Boosting (Sequential)**
  - **Gradient Boosting Machine (GBM)**
  - **XGB Regressor**



Random Forest - Example

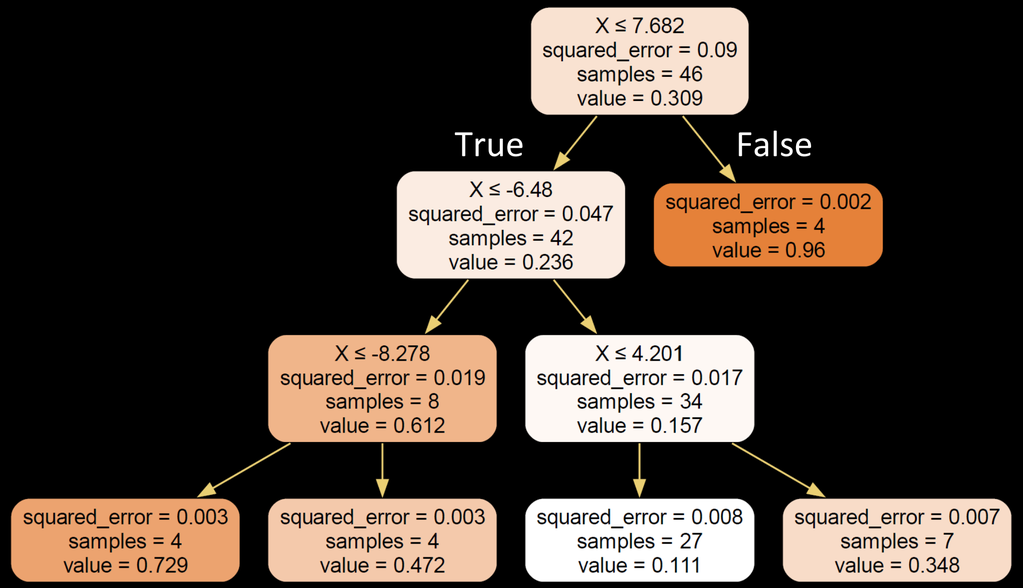
- **Hyper-parameters**
  - **n_estimators = 2**
  - **max_depth = 3**
  - **max_leaf_nodes = 5**



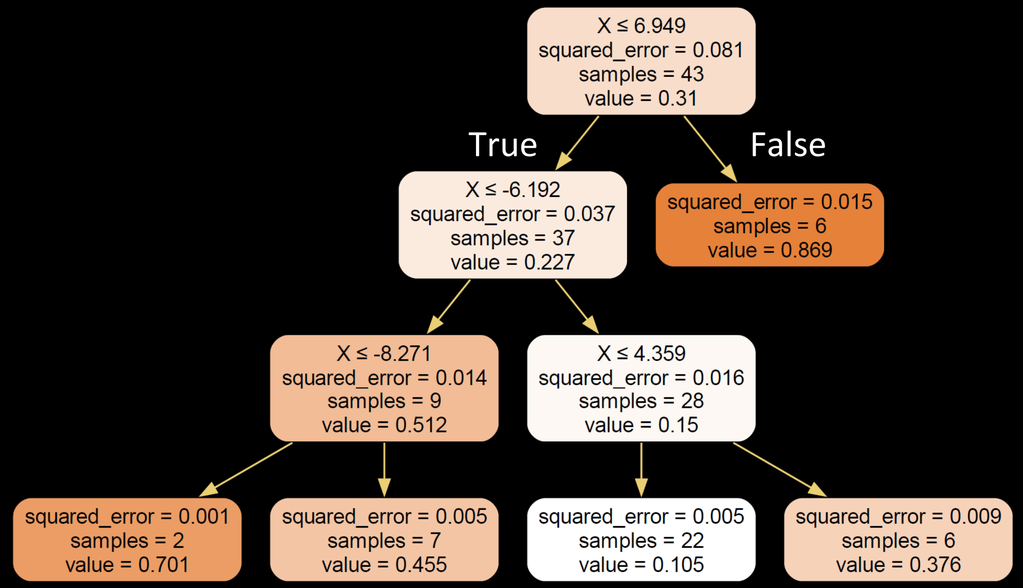
intro/RF_intro.py

Random Forest - Example

Estimator (Tree) #1



Estimator (Tree) #2



y_1 → $y = \frac{y_1 + y_2}{2}$ ← y_2

intro/RF_intro.py