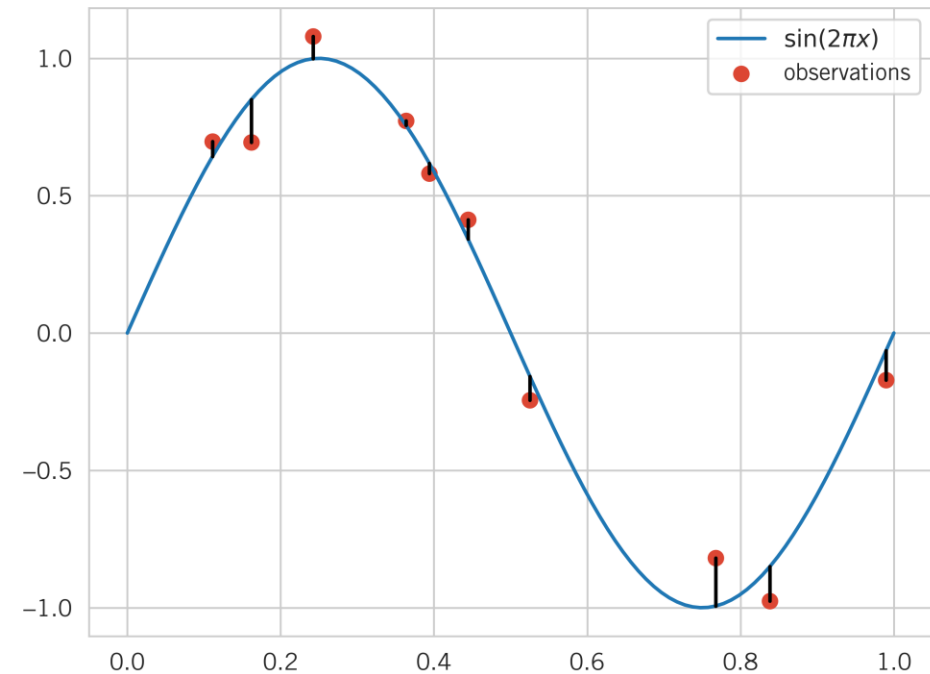


# **NSERC CREATE for BioZone Machine Learning Bootcamp**

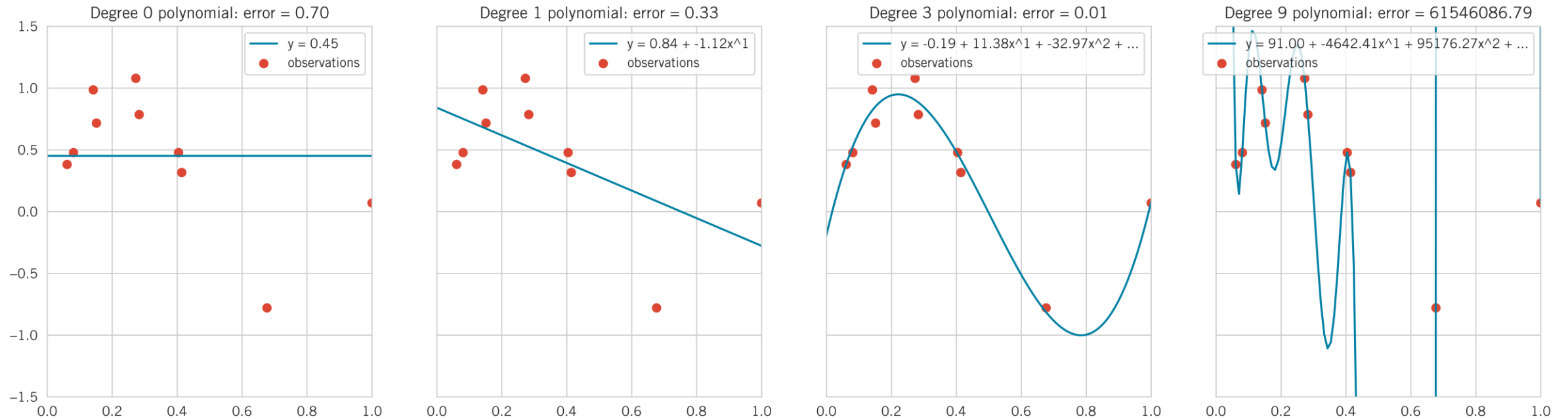
Regularization and Hyperparameter Tuning

# Regularization

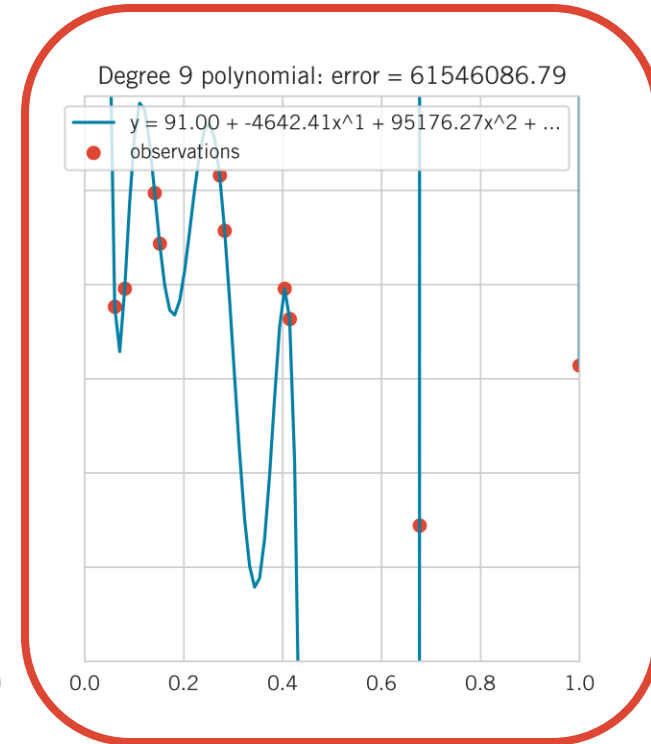
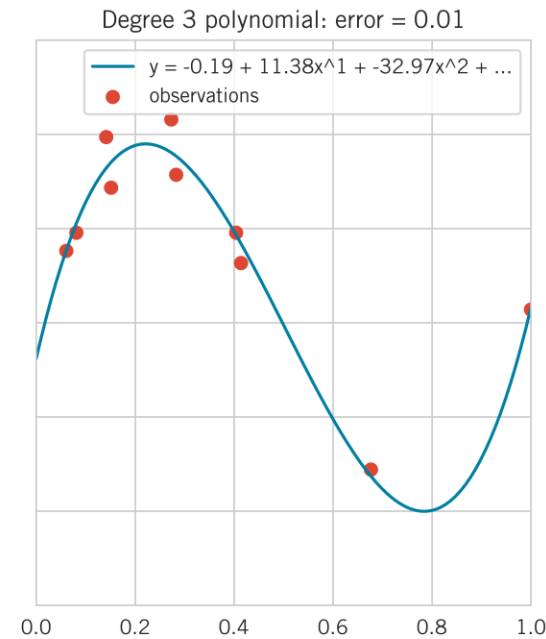
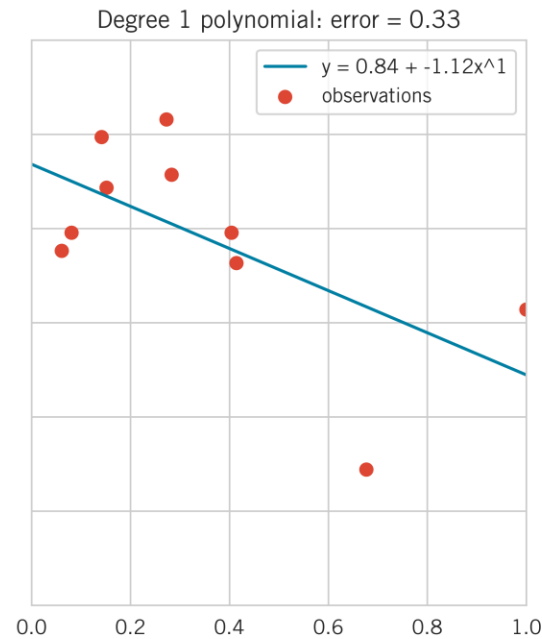
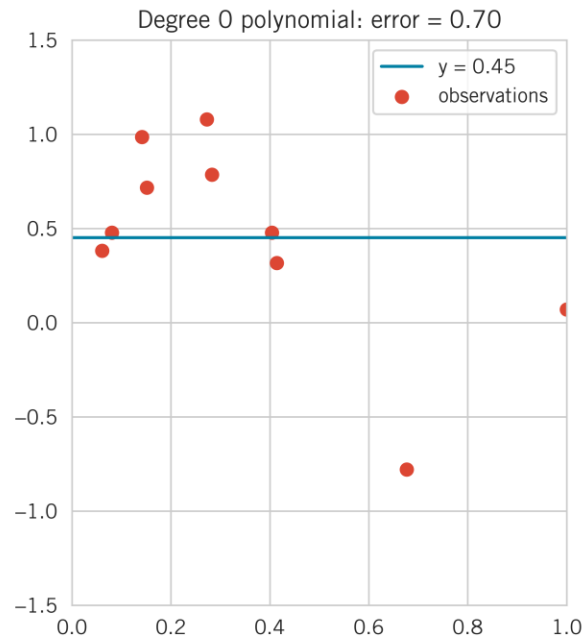
- Start with data points  $(x, y)$  generated by adding noise to  $\sin(2\pi x)$
- Each noisy point has a residual: the difference between the point and the true value
- $E(y_{obs}) = \frac{1}{n} \sum_{n=1}^N (y_{obs} - y)^2$
- Mean Squared Error



# Polynomial Regression



# Polynomial Regression

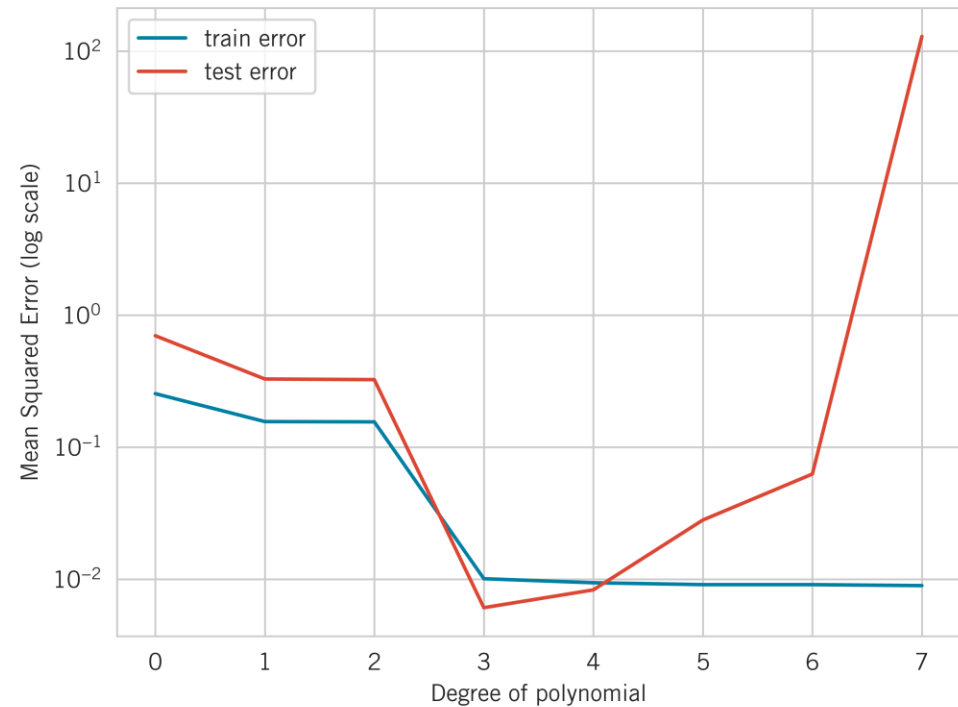


Underfitting: The model is not complex enough for the data

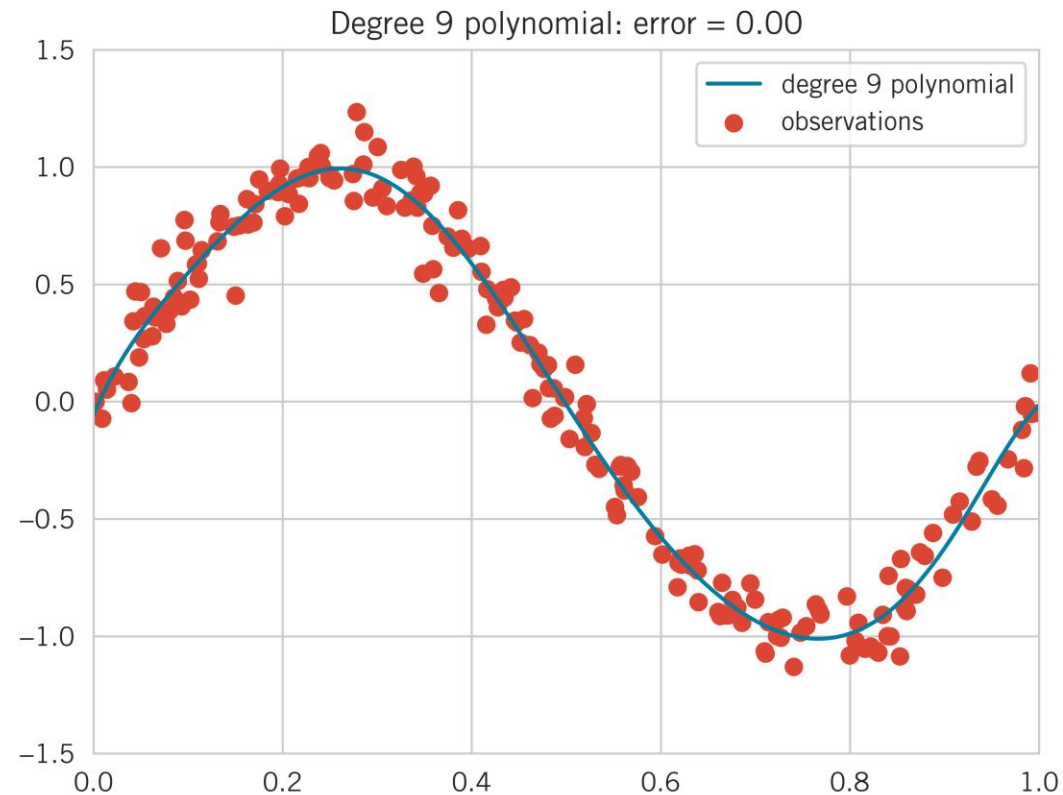
Overfitting: the model is too complex for the data

# Error as a function of degree of polynomial

- Overfitting occurs when the test performance decouples from the train performance
- Train error will typically trend toward zero as the model gets more complex
- With a complex enough model, it can “memorize” every training sample



# Dealing with overfitting (1): more data



# Dealing with overfitting (2): regularization

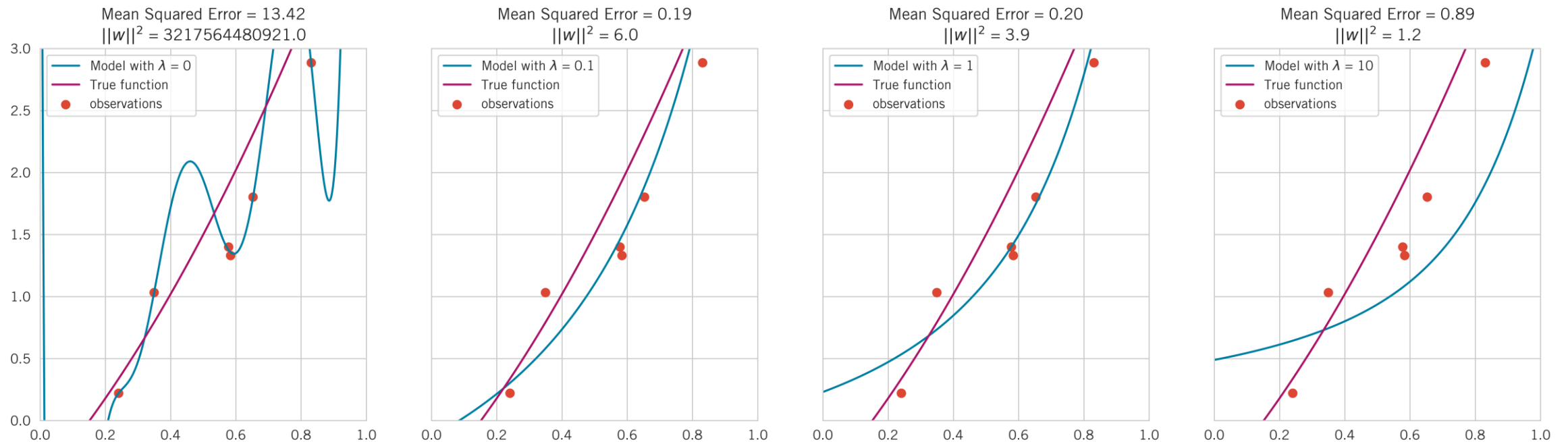
- For some model  $f(x, \mathbf{w})$  where  $x$  is the input and  $w$  are the parameters:

- $E(f) = \underbrace{\frac{1}{n} \sum_{n=1}^N (f(x, \mathbf{w}) - y)^2}_{\text{Mean Squared Error}} + \underbrace{\lambda \|\mathbf{w}\|^2}_{\text{Penalty on the size of parameters}}$

Mean Squared Error

Penalty on the size of parameters

# Effect of regularization

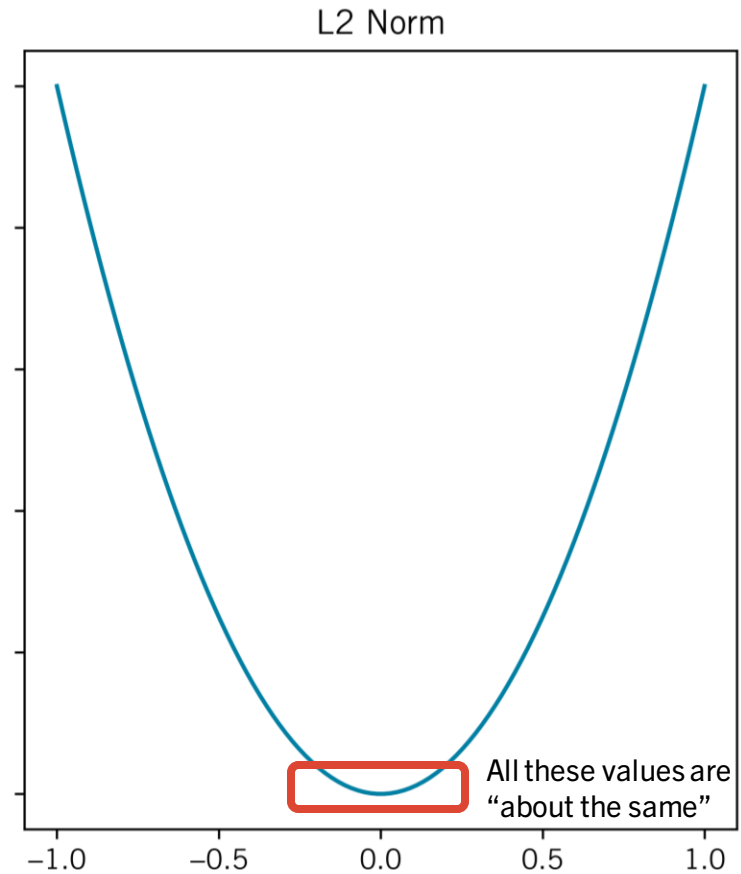
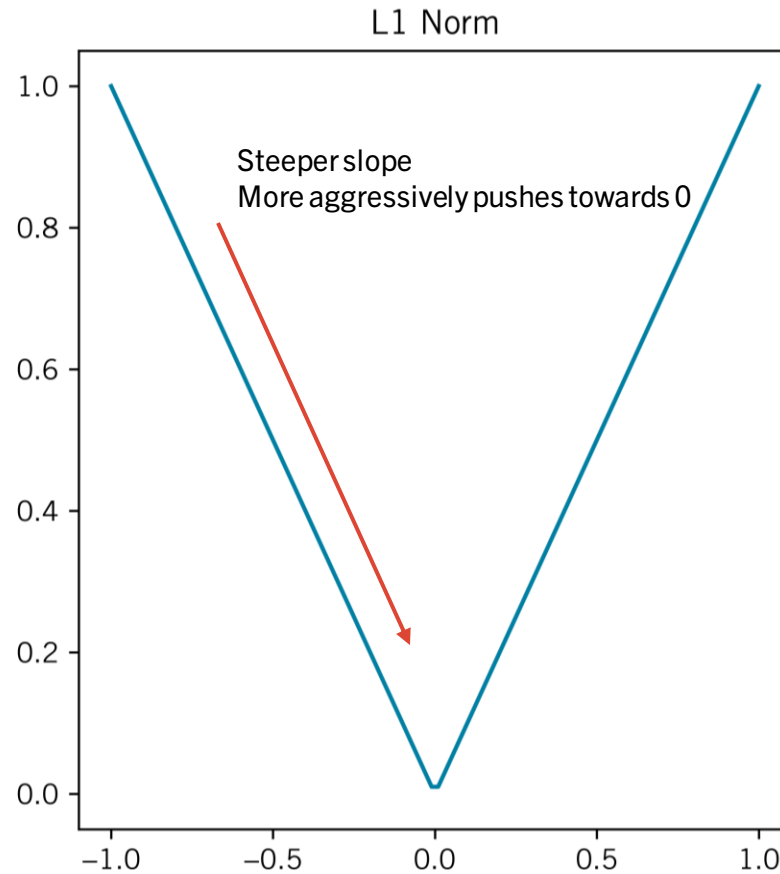




# L1 and L2 regularization

- L1 Regularization (aka LASSO):  $\lambda \sum ||w||$ 
  - Tends to produce sparse solutions, where some coefficients are zero
  - Useful for feature selection, as unimportant features are ignored
  - Less stable when multiple correlated features exist
- L2 Regularization (aka Ridge):  $\lambda \sum w^2$ 
  - Tends to distribute weights evenly and doesn't push coefficients to zero.
  - More stable solution where multiple correlated predictors exist, will include all of them.
  - Tends to perform better when all features are relevant.

# L1 and L2 regularization

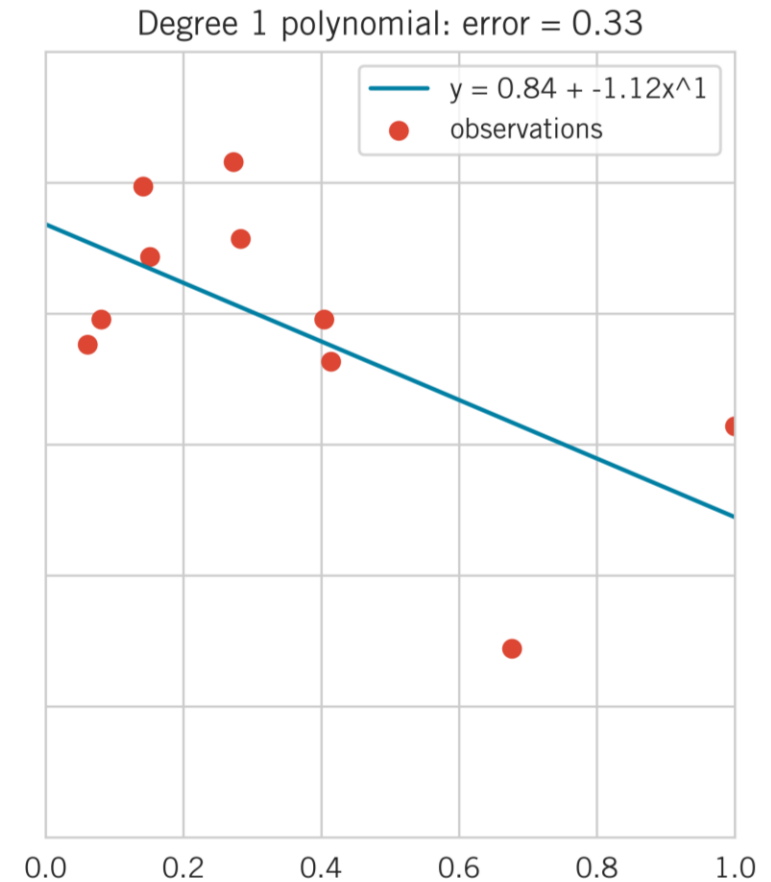


# Bias / Variance Tradeoff

- Regularization handles overfitting, a high variance problem.
- Overfitting: model too sensitive to training data specifics.
- Conversely, underfitting represents high bias.
- The goal: balance bias (flexibility to learn) and variance (ability to generalize).

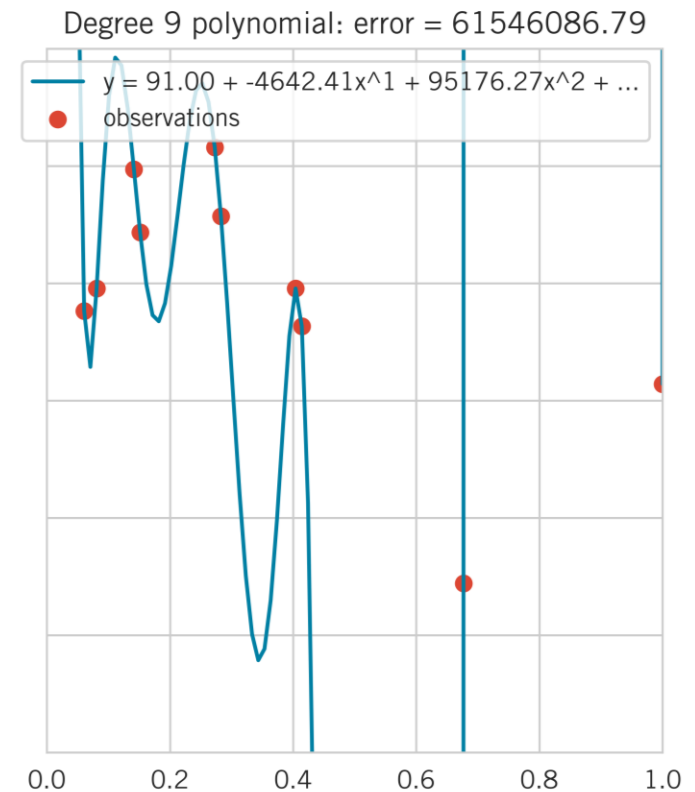
# Bias / Variance Tradeoff: Bias

- Bias: Error from assumptions the model makes about the data
- Linear model assumes the data is linear
- In general, a simpler model is making more assumptions → high bias



# Bias / Variance Tradeoff: Variance

- Variance: Algorithm's sensitivity to noise
- More complex models are more sensitive!
- High variance hurts generalization



# Bias / Variance Tradeoff

$$\text{Error} = \text{Noise} + \text{Bias} + \text{Variance}$$

- Noise
  - Random variations in data
- Bias
  - Error from assumptions the model makes about the data
  - Less complex model → more assumptions
- Variance
  - Algorithm's sensitivity to noise
  - More complex algorithms are more sensitive

# Parameters vs Hyperparameters

- Parameters:
  - Values learned from the data during training
  - In NNs, weights and biases
- Hyperparameters:
  - Settings affecting the structure or training process of the model
  - Not learned during training, but defined beforehand
  - Everything we've been discussing this afternoon

# Hyperparameter Tuning

- Before selecting our final model, we explore the space of possible configurations
- This exploration is known as hyperparameter tuning
- Hyperparameter tuning methods aim to find the combination of hyperparameters that yields the most predictive model.
- The aim is not only to improve model accuracy but also to prevent issues like overfitting and underfitting
- Note: Hyperparameter tuning can be time-consuming and computationally intensive, but the payoff is a more effective and reliable model.



# Hyperparameter Tuning: Validation Set

Training Set

Testing Set

# Hyperparameter Tuning: Validation Set

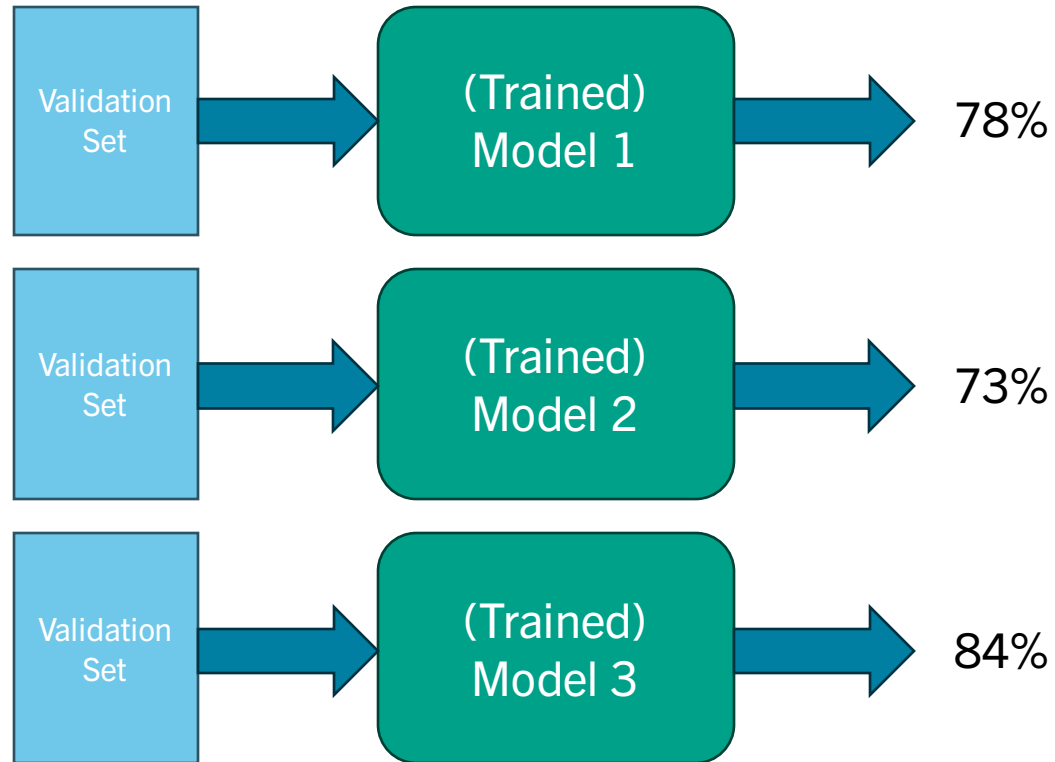


Training Set

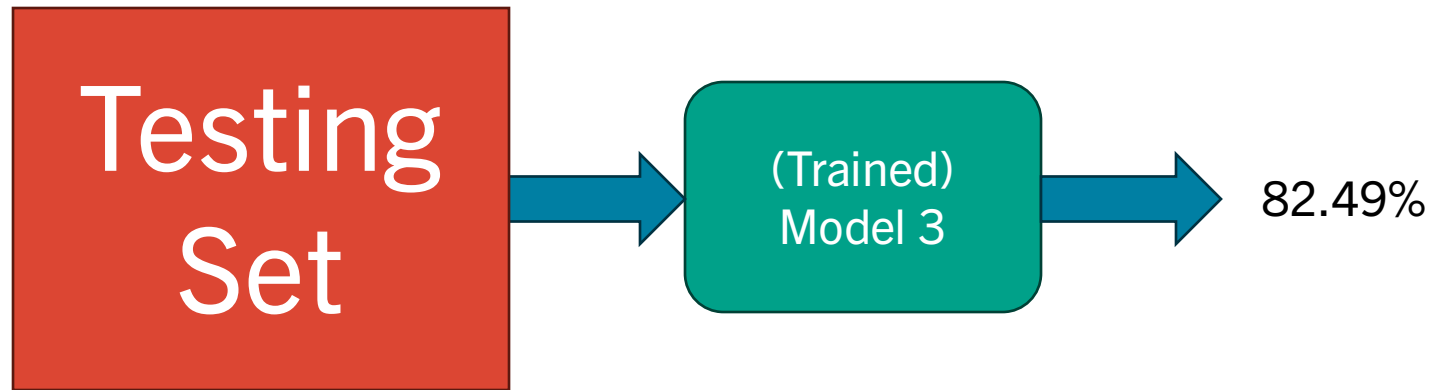
Validation  
Set

Testing Set

# Hyperparameter Tuning: Validation Set



# Hyperparameter Tuning: Validation Set



# Defining the Search Space

- Instead of a model, we can build a hypermodel
- Replace concrete definitions with range of acceptable values

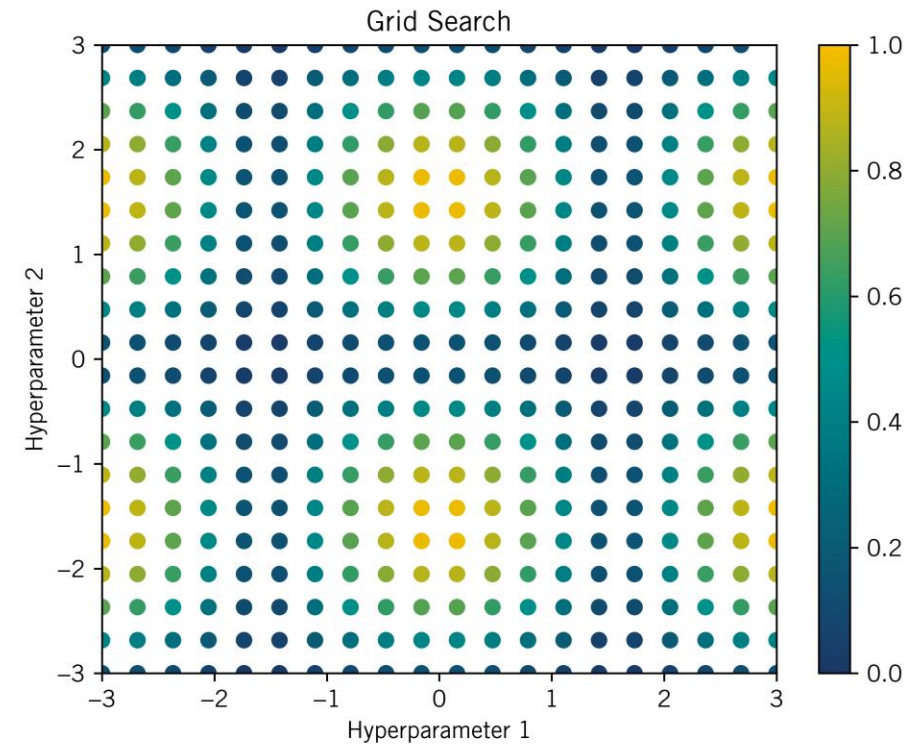
```
n_layers = 5
```

```
n_layers = hp.Int('layers', min_value=1, max_value=15) # Keras Tuner
```

```
n_layers = trial.suggest_int('layers', 1, 15) # Optuna
```

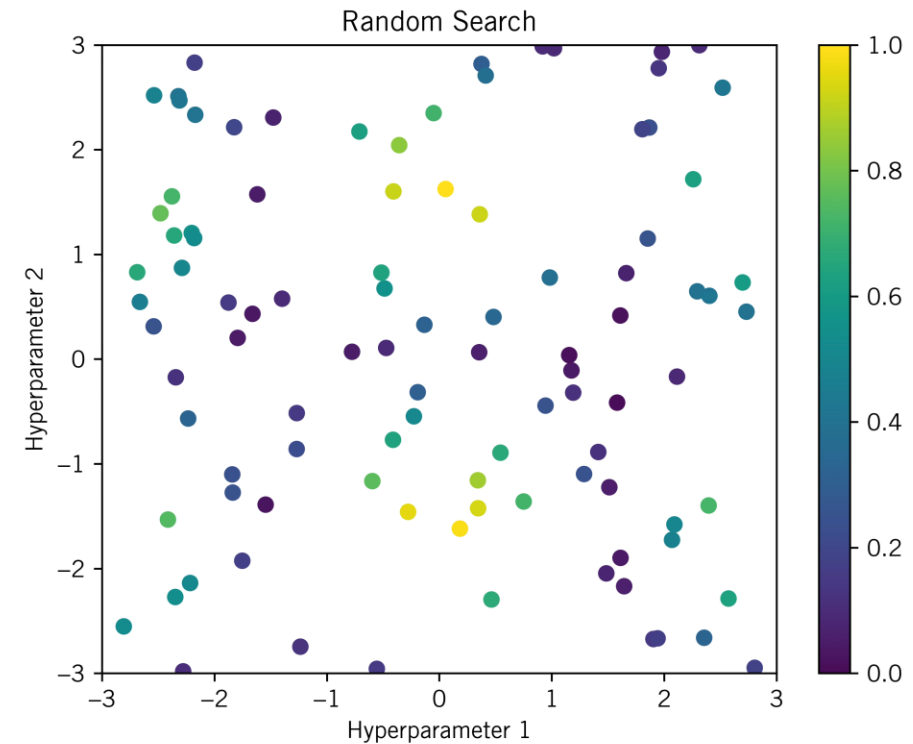
# Finding Optimal Parameters: Grid Search

- Naïve approach: try every possible combination
  - This is actually an accepted method!
  - Guaranteed to find the best combination in the defined space
  - Quickly becomes intractable with more parameters



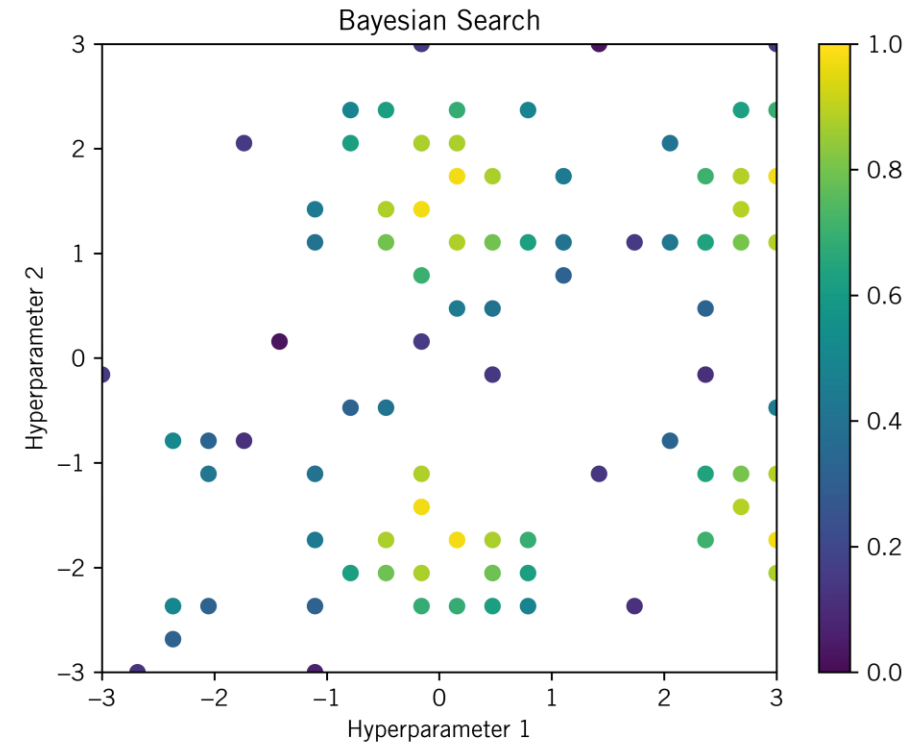
# Finding Optimal Parameters: Random Search

- A more efficient alternative to Grid Search.
- Randomly samples the defined space of hyperparameters.
- Not guaranteed to find the best combination, but often finds a good combination quickly.
- Particularly useful when dealing with a larger number of parameters



# Finding Optimal Parameters: Bayesian Search

- An advanced, intelligent approach to hyperparameter tuning.
- Uses information from past evaluations to choose the next parameters.
- Creates a probabilistic model mapping hyperparameters to a probability of a score on the objective function.
- Balances exploration (testing new, uncertain parameters) with exploitation (choosing parameters that look promising).
- Highly efficient, especially when evaluations are costly (e.g., tuning deep neural networks).





# Preparing for Hyperparameter Tuning

- **Start with a Reasonable Baseline:** Use known good configurations from the literature as a starting point, or use heuristics to choose a good initial configuration.
- **Scale Up Gradually:** Start with a smaller network or fewer epochs while tuning, then scale up once you've narrowed the hyperparameter range.
- **Focus on the Most Impactful Parameters:** Not all hyperparameters are created equal. Often, the learning rate, batch size, and number of layers will have a big impact on performance.

# Efficient Hyperparameter Tuning

- **Coarse to Fine Search:** Begin with a broad range and refine the search space as you identify promising regions.
- **Parallelize Hyperparameter Search:** If resources permit, train multiple models with different hyperparameters in parallel.
- **Use Automated Tuning if Possible:** Consider using automated tuning libraries, which can handle the tuning process more efficiently.
- **Record and Analyze Results:** Keep track of the performance for each set of hyperparameters. Visualization or analysis of these results can often yield insights and guide the search.