# Machine Learning
# Lecture 1: The Big Picture

Reid Wyde

# Course Outline

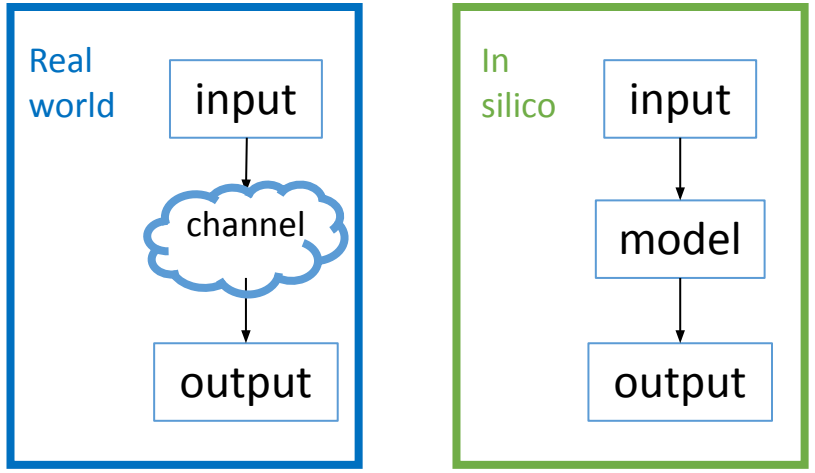
- Lecture 1 – The Big Picture
- Lecture 2 – Linear Algebra
- Lecture 3 – Probability and Statistics
- Lecture 4 – Supervised Models I
- Lecture 5 – Supervised Models II
- Lecture 6 – Unsupervised Models
- Lecture 7 – Deep Learning Intro I: Training and Optimizers
- Lecture 8 – Deep Learning Intro II: Computer Vision
- Lecture 9 – Deep Learning Intro III: Sequence Learning
- Lecture 10 – Reinforcement Learning
- Lecture 11 – Neural Architecture Search (AutoML)
- Lecture 12 – Graph Neural Networks

# Lecture 1 Outline

- Machine Learning as Task-Performance-Experience
- Supervised vs Unsupervised Learning
- Generalization
- Partitioning Data, Train-Test Split
- Data Leakage
- Learning Curves, Underfitting, Overfitting, Bias-Variance Tradeoff
- Data Augmentation
- Parameters and Hyperparameters
- Partitioning Data II: Train-Validate-Test Split
- K-Folds Cross Validation
- Summary
- Closing Thoughts
- Extra Resources

# Task-Performance-Experience

- An algorithm is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$ if its performance at tasks in $T$ (as measured by $P$) improves with experience $E$ (Michell 1997).
- For our purposes, algorithms are models
  - Models are function approximators
    - They can themselves be elementary functions or compositions of elementary functions
  - Examples:
    - Linear maps / linear transforms / affine transforms
    - Polynomials
    - Exponentials
    - Decision Trees
    - Neural Networks
    - And many more!

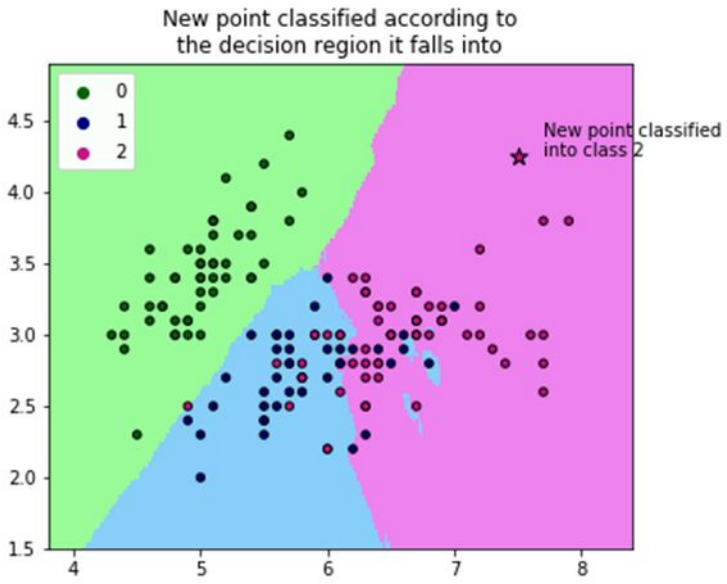


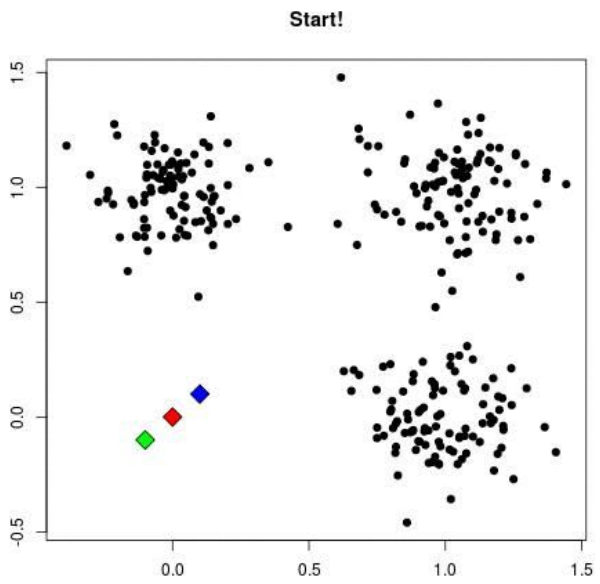
# Algorithms: Supervised vs Unsupervised

- Supervised: Input-output pairs exist
  - There's a right answer ("label") and we want our model to predict as close to the right answer as possible.
  - E.g. classification, regression, etc.
- Unsupervised: Only data exists
  - Operate on the data without there being a right answer to compare to.
  - E.g. PCA, ICA, clustering, etc.
- Self-Supervised: Data exists, input-output pairs can be created automatically
  - E.g. taking clean audio and adding noise to it, so that the clean audio is the output and the noisy audio is the input
- **The vast majority of all real world data is unlabeled**

# Task: What we want the model to **do**

- Examples:
  - We want the model to **classify** this contact as **target or non-target**
  - We want the model to **predict** the contact's **location**
  - We want the model to **group** similar contacts together
  - We want the model to **love** us unconditionally



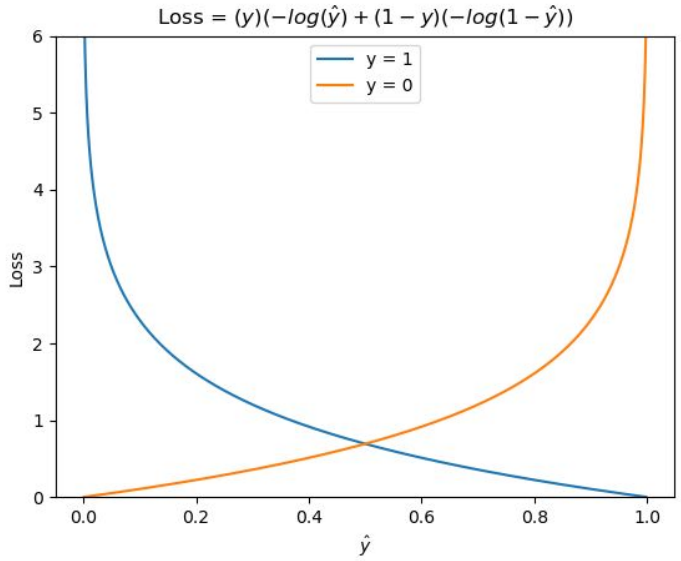

# Tasks, cont’d

- The type of task determines what kind of model output you want
  - Classification -> single discrete value (label), or probability distribution over labels
  - Regression -> single real value (or tensor), or probability distribution over real values (or tensors)
  - Transcription / translation -> text, or symbols, or language tokens
  - Anomaly detection (special case of classification) -> Binary (yes/no)
  - Synthesis / sampling / denoising -> output same data type as input
- **The structure of the task (in part) determines the structure of the model**

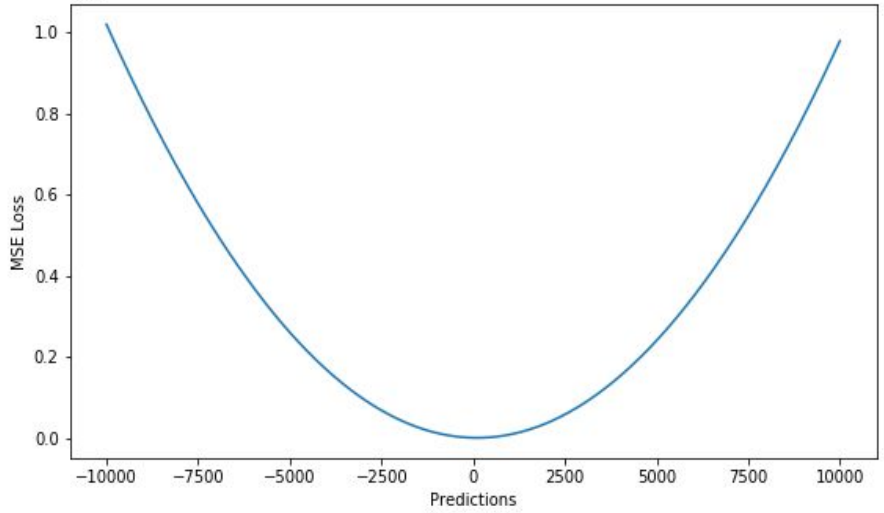
# Performance Measure: How to **judge** the model

- Quantitative measure of model performance
- A.k.a. Objective function, loss function
- Different performance measures for different tasks
  - Cross-entropy (a.k.a. log-loss) for classification
  - Mean squared error (MSE) for regression
  - Many more: Accuracy, precision, recall, specificity, F1, AUC, logMSE, etc…

Cross-entropy




Mean Squared Error

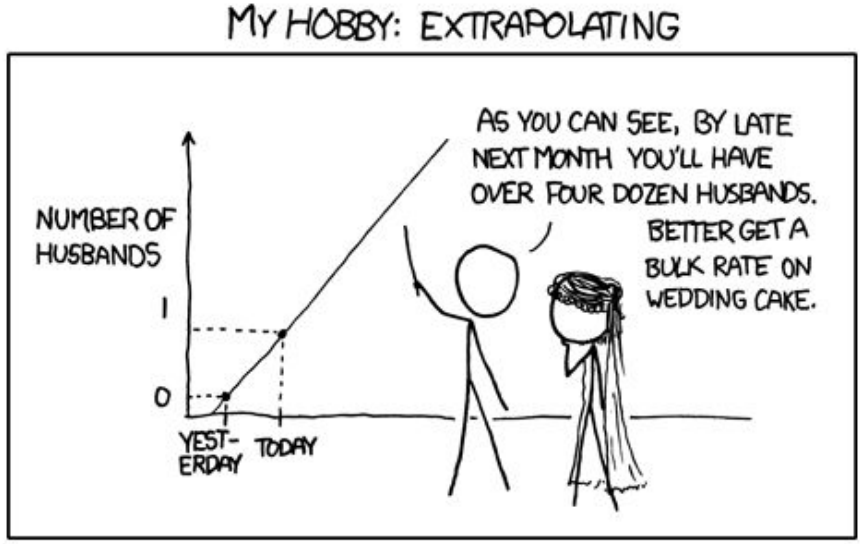


# Experience: How you **train** the model

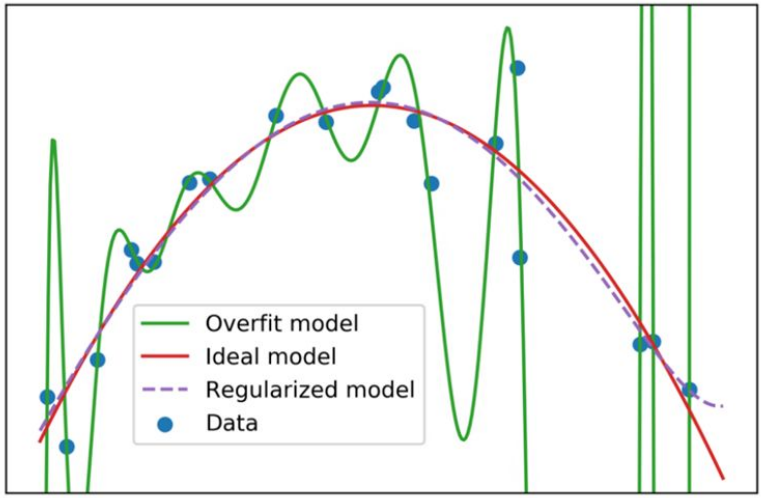
- The experience requires a model and data (and a performance measure P)
- The data can be the entire dataset or any subset of it
- The model parameters are adjusted to (better) fit the data of this experience (i.e. to perform better with respect to P)
- Examples (where P is MSE):
  - Fitting a gaussian distribution to a dataset's mean and variance
  - Fitting a line to points in the Cartesian plane
  - Adjusting the coefficients of an FIR adaptive filter after receiving another audio sample

# Generalization (Not Memorization!)

- A useful model doesn't memorize training data, it "learns" from the experience, so it can perform well on data it hasn't yet experienced
- We want the model to **generalize** to **out of distribution data** (i.e. non-training data)
- Often we need a lot of data to create a model that generalizes well




source: Randall Munroe, xkcd.com



# Partitioning Data, Train-Test Split

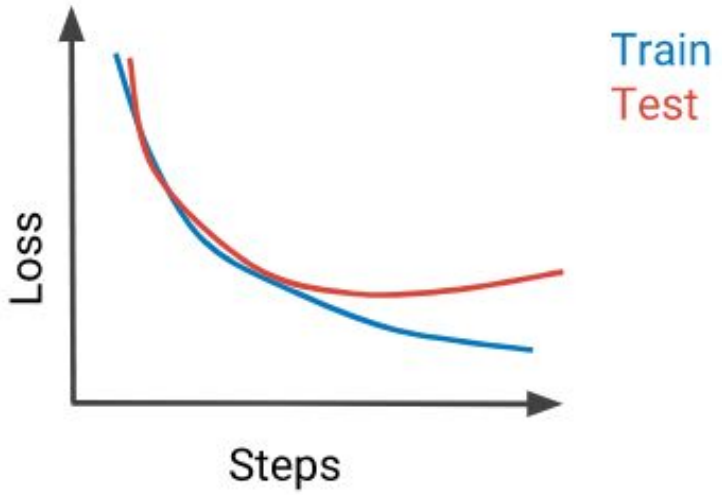
- In order to evaluate model performance on data it hasn't experienced, we don't train on all of our dataset
- The data we don't train on is called the "test set"
- The default is an 80/20 train-test split, but it varies by problem
- Our model's performance on the test set is the best predictor of how well it will generalize to unseen data
- Often we must be careful about how we partition our data, or else we can bias our prediction of our model's performance; this is called data leakage

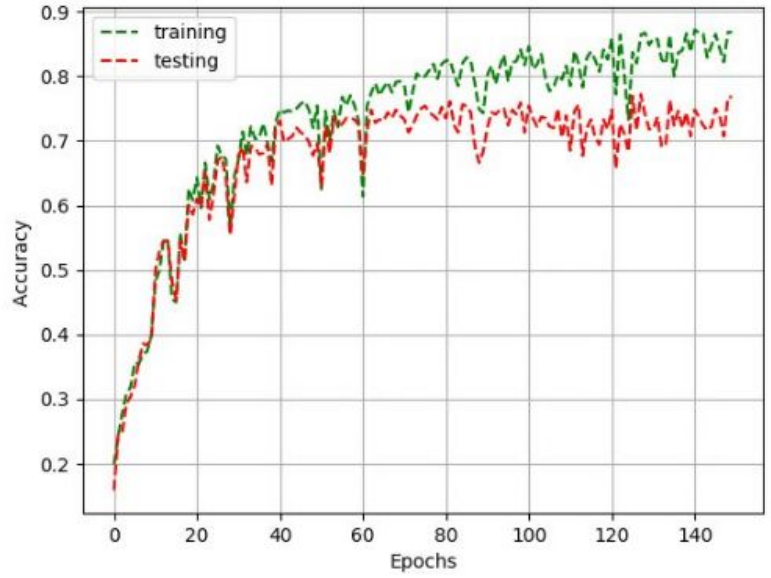
# Data Leakage

- Data leakage occurs when the model is allowed to train on the testing data
- This biases our estimate of how well the model generalizes, because it will overperform on the testing data compared to new data from the real world
- Example:
  - You have a dataset of audio samples and surveys from users who used a customer support line
  - There are multiple audio samples per user
  - We are training a model to classify the mood of the user
  - **We want this model to generalize well to new users**
  - If we partition our dataset such that **datapoints from a given user end up in both training and testing** data, we have allowed our testing data to **leak** into our training data (it's being tested on a user it's already experienced)
  - Solution: we should partition our data with respect to users, not samples

# Learning Curves

- A learning curve is a plot of the model's performance vs experience
- Learning curves allow us to analyze many things:
  - The model's capacity (best performance on training data)
  - How quickly the model learns
  - How much data the model needs to perform well
  - How well the model generalizes



# Underfitting, Overfitting, Bias, Variance

- A model that doesn't perform well on the training or testing data is "high bias"
  - The model has "underfit" the training data
- A model that performs well on the training data, but not testing data is "high variance"
  - The model has "overfit" the training data
- A model that performs well on testing data, but not training data is "spooky"
  - It's likely the model must be trained more to analyze its performance; check code for bugs

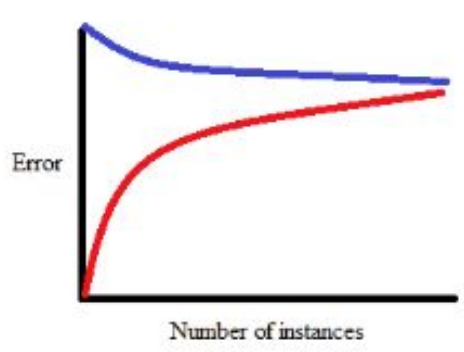



Image #1 (high bias)

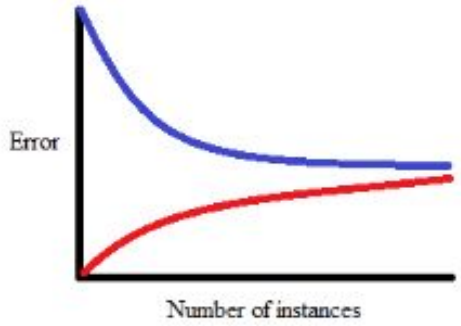



Image #2 (ideal)

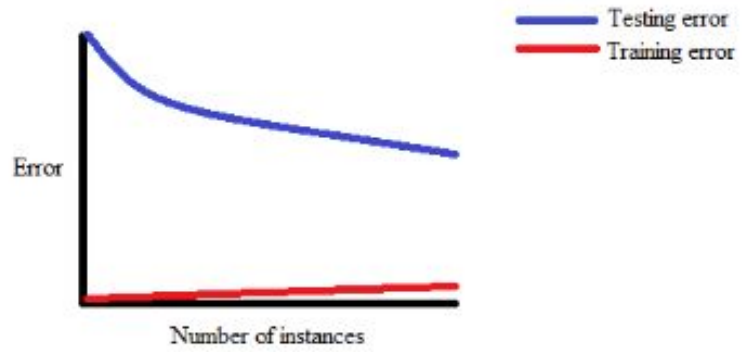
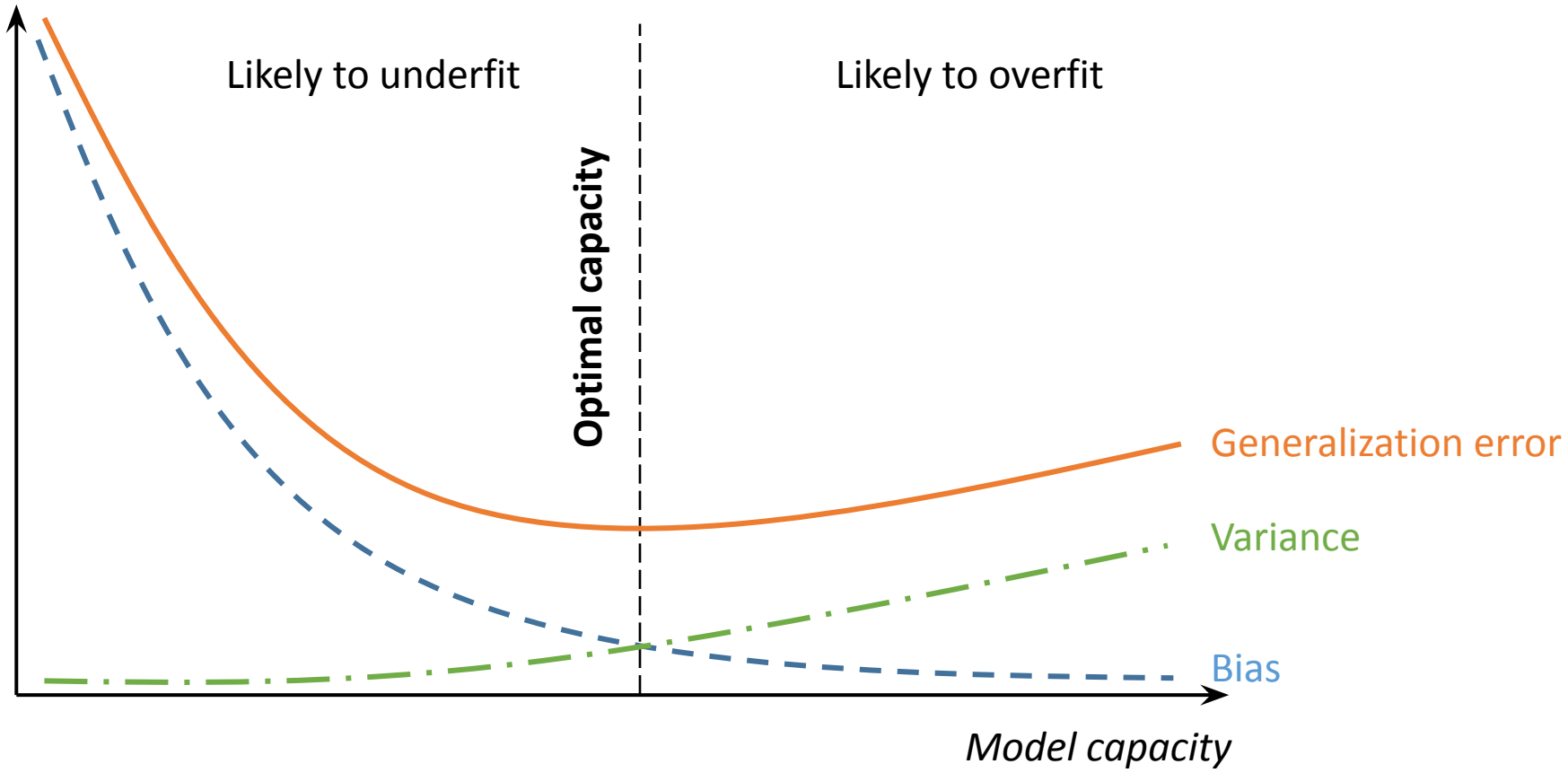



Image #3 (high variance)

# Bias-Variance Tradeoff

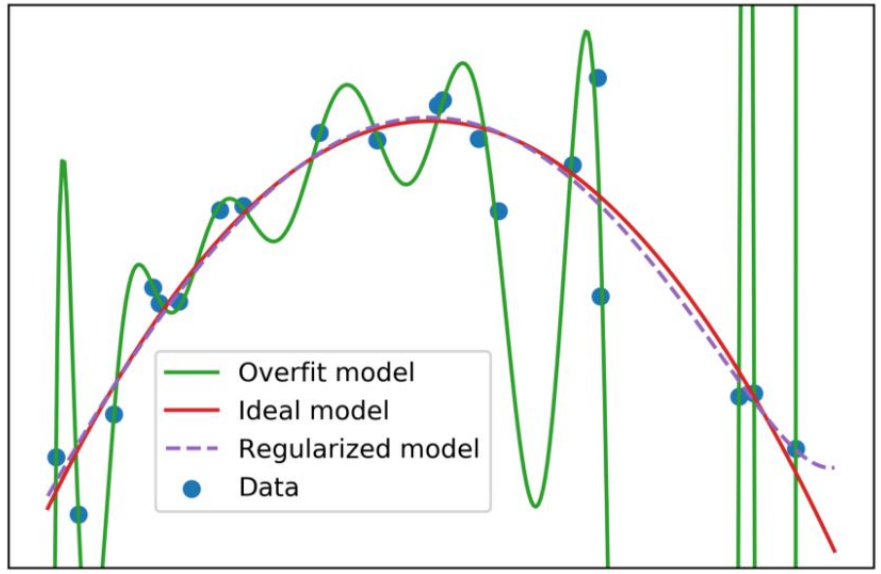
- In general it is difficult to construct a model that can achieve perfect performance on the training data, while still performing well on the testing data
  - The training data may have noise, which we don't want to model
- In fact, there is a general tension between model capacity and model generalization
  - This is known as the "bias-variance tradeoff"
  - This is why simpler models are almost always considered more robust

# Bias-Variance Tradeoff



# Addressing High Bias and Variance in Models

- A high bias model has low capacity
  - Increase model complexity until it can perform well on the training data
- A high variance model has poor generalization
  - Increase the size of your dataset (never a bad option, but can be expensive)
  - Regularize your model. Regularization is when you change the experience to avoid overfitting.
    - E.g. adding the square of the coefficients of your model to the loss function (L2 normalization, a.k.a. “leaky” LMS)
    - E.g. Limiting the degree of a polynomial model

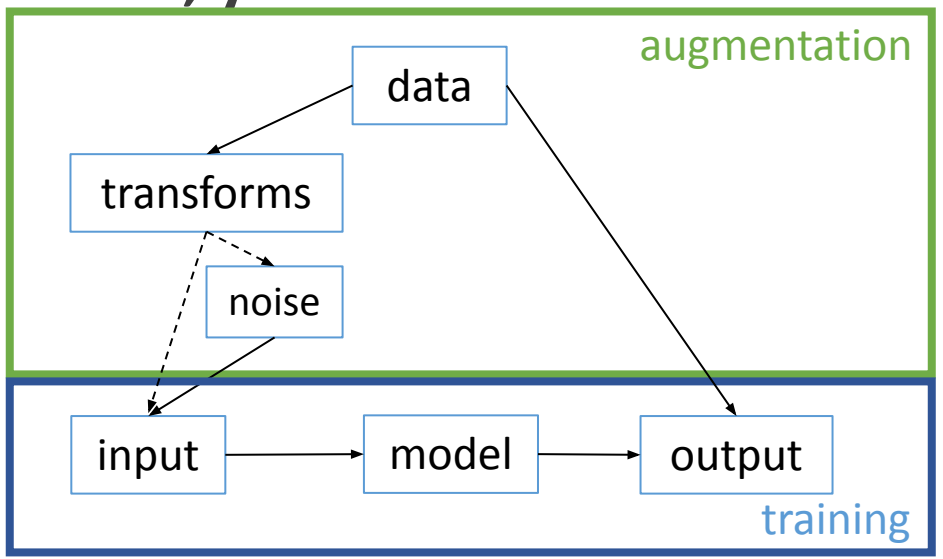


# Common Early Debugging Steps

- Count and verify your train/test ratio
- Visualize input and output pairs before, during, and after training
- Review process for data leakage
- **Testing data should receive the same pre-model processing as training data!**
  - We want the training data statistics and the testing data statistics to be as close as possible, so the model has the best chance at generalization

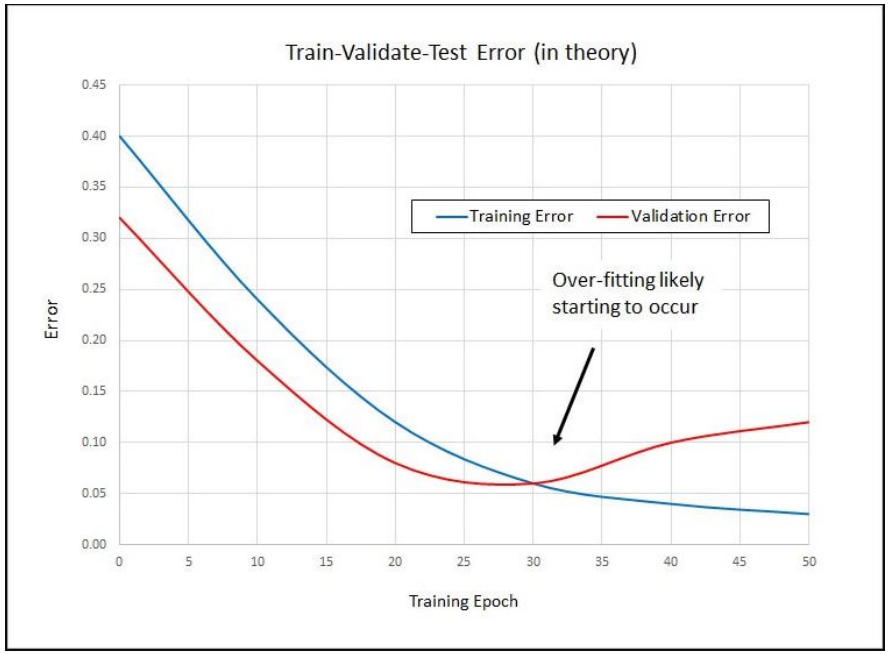
# Data Augmentation

- Collecting data can be expensive
- Real world data is often noisier than perfect simulations, which hurts generalization
- It is possible to artificially inflate the size of a dataset and make the model more robust at the same time through data augmentation
  - It can also prevent overfitting
- Data augmentation is when you train on transformed versions of the original dataset
- **If your augmentation includes a noise element, your model becomes robust to that kind of noise**
- Examples:
  - Adding white noise to image or audio data
  - Randomly cropping image data
  - Adding reverb to audio data
  - Flipping images on an axis



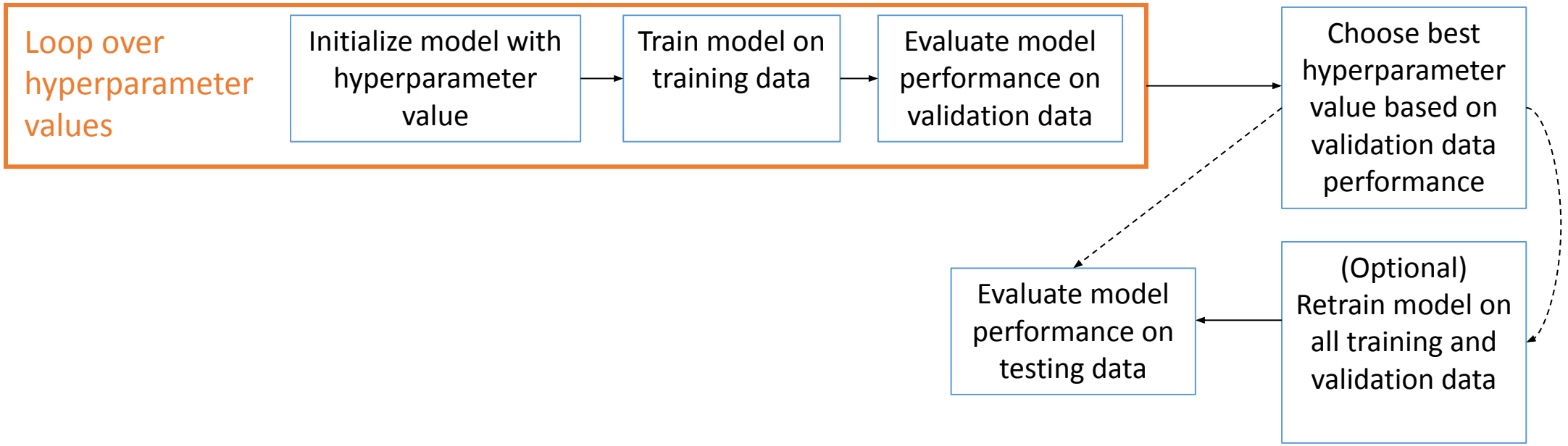
# Parameters and Hyperparameters

- Parameters of a model are automatically learnable
  - Values of parameters are learned from experience
  - Examples:
    - Coefficients of a polynomial
    - Weights of a neural network
    - The set of (μ, σ) pairs of a GMM
- Hyperparameters are not automatically learnable
  - Values of hyperparameters must be chosen apriori
  - Examples:
    - Maximum degree of a polynomial
    - Structure of a neural network
    - Number of Gaussians in a GMM
    - Type / amount of noise in data augmentation
    - **Optimal number of training epochs (to not overfit)**



# Partitioning Data II: Train-Validate-Test Split for Hyperparameter Search

- We can experiment with hyperparameters, but we want to avoid data leakage
- To experiment with hyperparameters, we divide data into training, validation, and testing
  - 60/20/20 and 70/10/20 divisions are common



# K-Folds Cross Validation

- Dividing data into train/test can result in labels that are unequally distributed between partitions
  - i.e. all of the examples for a given class can end up in training or testing, which would bias our model
- Evaluation of model performance on a single train-test split is just a point estimate of performance
  - We want a tight estimate of model generalization performance
- To address this, we repeat the process of training and testing many times on different splits of the data

  **Divide the data into K nearly equal subsets (a.k.a. folds)**

  **Loop i from 0 to K-1:**

  **Train model on all subsets except the $i^{th}$ subset**

  **Evaluate model on the $i^{th}$ subset**
- We get a tighter estimate of model generalization performance
- (For free) we get an estimate of the sensitivity of the model to different datasets

# Summary

- Machine learning as task-performance-experience (plus our model!)
- Goal: Generalize model to unseen data
- Train-test split to evaluate model generalization performance
- Avoid data leakage with careful data handling
- Use learning curves to evaluate models for underfitting and overfitting
- Use data augmentation to increase the size of a dataset, improve model robustness to noise, and prevent overfitting
- Experiment with hyperparameters using a train-validation-test split
- Use K-folds cross validation to better evaluate model performance and sensitivity

# Closing Thoughts

- There is no replacement for domain specific knowledge.
  - Data scientists must understand where their data comes from
- ML is not magic. Ultimately, it's all about the quality of your data.
  - That's why I prefer “data driven modeling” to “machine learning”
  - 90% of your effort is related to data engineering, not model building
- Many of these techniques have been derived independently multiple times.
  - There are strong overlaps between ML and other fields such as signal processing, physics, information theory, optimization, and optimal control
  - I encourage you to draw connections to the math you already know
- Occam's razor. Simpler is almost always better.
- Pareto principle: 80% of the gain comes from 20% of the effort, and 20% of the gain comes from 80% of the effort.

# Extra Resources

- Coursera *Machine Learning* Course by Andrew Ng
  - Introduction to supervised and unsupervised machine learning, covering many models, ends with basic neural networks
  - Original course was taught in octave (essentially MATLAB language)
  - Focus on practice and implementation, not just theory
- *Applied Linear Algebra: The Decoupling Principle* by Lorenzo Sadun
  - Application focused, very good for building intuition around linear transforms and vector spaces
  - Dr. Sadun also has many online video lectures which are of very high quality
- *Deep Learning* by Goodfellow, Bengio, and Courville
  - Concise and clear descriptions of many kinds of neural networks, with many citations for further research by the student
- *Reinforcement Learning* by Sutton and Barto
  - Detailed treatment of reinforcement learning from tabular methods and dynamic programming to modern deep neural network approaches