AI-Systems
Machine Learning Lifecycle

Joseph E. Gonzalez
Co-director of the RISE Lab
jegonzal@cs.berkeley.edu
Objectives For Today

- Introduce the machine learning lifecycle
  - Challenges and Opportunities
  - Science vs Engineering

- Review Key Concepts in Readings
  - Hyperparameters
  - Model Pipelines, Features, and Feature Engineering
  - Warm Starting and Fine Tuning
  - Feedback Loops, Retraining and Continuous Training

- Important context for papers and what to expect
What is the Machine Learning Lifecycle?

Model Development
- Offline Training Data
- Data Collection → Cleaning & Visualization → Feature Eng. & Model Design → Training & Validation

Training
- Training Pipelines → Trained Models
- Live Data → Validation

Inference
- Prediction Service
- Logic
- Feedback
- Data Scientist
- Data Engineer
- Data Engineer
Model Development

Data Collection → Cleaning & Visualization

Training & Validation ← Feature Eng. & Model Design

Identifying potential sources of data

Joining data from multiple sources

Addressing missing values and outliers

Plotting trends to identify anomalies

Data Collection

Cleaning & Visualization

Training & Validation

Feature Eng. & Model Design

Identifying potential sources of data

Joining data from multiple sources

Addressing missing values and outliers

Plotting trends to identify anomalies

Help!
Model Development

- **Data Collection**
- **Cleaning & Visualization**
- **Training & Validation**
- **Feature Eng. & Model Design**

**Identifying** potential sources of data

**Joining** data from multiple sources

**Addressing** missing values and outliers

**Plotting** trends to identify anomalies

Data Scientist

Data Engineer

Help!

Happy to JOIN
In Data Science, 80% of time spent prepare data, 20% of time spent complain about need for prepare data.
Andréj Karpathy (Tesla Auto Pilot Team)

How many of you have ever worked with real data?
Model Development

Identifying potential sources of data

Joining data from multiple sources

Addressing missing values and outliers

Plotting trends to identify anomalies

Data Collection → Cleaning & Visualization → Data Collection

Training & Validation ← Feature Eng. & Model Design ← Training & Validation

Help! ← Data Scientist

Happy to JOIN ← Data Engineer

Offline Training Data
Model Development

Building informative features functions

Designing new model architectures

Tuning hyperparameters

Validating prediction accuracy

Data Collection → Cleaning & Visualization

Training & Validation ← Feature Eng. & Model Design

I was born for this!

Data Scientist

Offline Training Data
Features and Feature Engineering

- **Features**: properties or characteristic of the input

- **Click Prediction Example**: Information about User and Content

  Model: \[ f_\theta(x) \rightarrow y \]

  Simple Logistic Model: ("Perceptron")
  \[ f_\theta(x) = \sigma \left( \sum_{k=1}^{d} \theta_k \phi_k(x) \right), \quad \sigma(t) = \frac{1}{1 + e^{-t}} \]

  Useful features?
  - User Features:
    - age, gender, and click history
  - Product Features:
    - Price, popularity, and description...
  - Combined (Cross) features:
    - \( I(20 < \text{age} < 30, \text{male, "xbox" in desc})... \)

- Features function extract numeric properties from \( x \)

  e.g., Recurrent NN output..

  e.g., Language Model Embedding

  Hand coded features

* Technically the original perceptron used a 0/1 non-linearity but this is a common abuse of terminology.
Additional Notes on Features

- **Feature Joins**: combine multiple data source in a feature

- **Feature Reuse**: good features can aid in many tasks
  - Example: product embeddings, user tags, ...

- **Predictions as Features**: predictions for one task (e.g., products in an image) can be useful features for another (e.g., ad targeting)

- **Feature Tables/Caches**: features are often pre-computed and cached
  - Requires tracking data and compute and feature versions

- **Dynamic Features**: features can often be modified faster than models
  - Useful for addressing fast changing dynamics (e.g., user preferences can be encoded in click history features).
  - Issue: resulting potential covariate shift can be problematic
Hyperparameters

- the parameters and more generally configuration details that are not directly determined through training
- set by hand or tuned using cross validation
- why not learn directly?

Find the Hyperparameters:

Objective:

\[
\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L_{\alpha}(f_{\theta}(x_i), y_i) + \lambda R(\theta)
\]

Training Algorithm

\[
u(t) \leftarrow \beta u(t-1) + \eta \sum_{i \in B} \nabla_{\theta} (L_{\alpha}(f_{\theta}(x_i), y_i))\bigg|_{\theta(t)}
\]

Architecture is sometimes treated as separate from hyperparameters

Can be learned...
Offline Training Data

Data Collection → Cleaning & Visualization

Training & Validation ← Feature Eng. & Model Design

Model Development Technologies

- Jupyter
- TensorFlow
- Keras
- PyTorch
- MXNet
- DMLC
- XGBoost
- Matplotlib
- Caffe2
- Pandas
- NumPy
- Apache Spark
- DASK
- HIVE
What is the output of Model Development

Offensive Training Data

Data Collection → Cleaning & Visualization → Feature Eng. & Model Design → Training & Validation

Reports & Dashboards

(Insights …)

Trained Model

Bad Idea
Why is it a **Bad Idea** to directly produce trained models from model development?

With just a trained model we are **unable to**

1. **retrain** models with new data
2. track data and code for **debugging**
3. capture **dependencies** for deployment
4. audit training for **compliance** (e.g., GDPR)
What is the output of Model Development

Offline Training Data → Data Collection → Cleaning & Visualization

Training & Validation ← Feature Eng. & Model Design

Trained Models → Reports & Dashboards

(Insights …)

Bad Idea
What is the output of Model Development

Offline Training Data

Data Collection → Cleaning & Visualization → Training & Validation

Feature Eng. & Model Design

Reports & Dashboards

(INSIGHTS …)

Training Pipelines
Training Pipelines Capture the **Code** and **Data Dependencies**

- Description of how to train the model from data sources

![Diagram of training pipelines]

**Software Engineering Analogy**

- Training Pipelines → Code
- Trained Models → Binaries
What is the output of Model Development

Data Collection → Cleaning & Visualization

Training & Validation ← Feature Eng. & Model Design

Reports & Dashboards

(Insights …)

Offline Training Data

Training Pipelines
Training models at scale on live data

Retraining on new data

Automatically validate prediction accuracy

Manage model versioning

Requires minimal expertise in machine learning
Training Technologies

Workflow Management:
- Apache Airflow
- Azkaban
- Lugi
- Oozie

Scalable Training:
- PyTorch
- mxnet
- TensorFlow
- Horovod
- Apache Spark
- XGBoost

Live Data ➔ Training Pipelines ➔ Trained Models ➔ Validation ➔ Data Engineer

Trained Models

Data Engineer

Live

Workflow Management:
Warm Start Training

Loss Surface for a two-parameter model.

Stochastic Gradient Descent
Warm Starts Training

1) New training data arrives and changes the loss surface.

2) Instead of starting over from random weights, start at previous solution.

Stochastic Gradient Descent

Works well if data is changing slowly.
More challenging for model changes.

Loss Surface for a two-parameter model.
Additional Thoughts on Warm Starting

- A form of **transfer learning** across time.
- Useful for situations where **new data** is arriving
  - Data distribution is not changing rapidly (but changing...)
- **Issues:**
  - Need a mechanism to set learning rates appropriately
    - Typically start much smaller
  - Could get stuck in suboptimal solution for non-convex settings
    - Though this is true in general
  - **Catastrophic forgetting:** if you only train on new data may degrade model on old data
    - Can address by continuing to train on old data
Fine Tuning

- Using small learning rates to train pre-trained or partially pre-trained model for a new dataset or prediction task.
- Enables both faster training and improved accuracy
Open Problems

Context & Composition
Context  How, What, & Who?

- **How** was the model or data created?
- **What** is the latest or best version?
- **Who** is responsible? (blame...)

Partial Solution

Training Pipelines

Track relationships between

1. **Code** versions ✔️
2. **Model & Data** versions
3. **People** (versions?)

**Context**
Composition

Models are being composed to solve new problems
Composition

Models are being composed to solve new problems

Puppy Detector

Yes

Ball Detector

Yes

Cuteness Detector

Cute!
Composition

Models are being composed to solve new problems

![Diagram of composition of models to detect cuteness]
Composition

Models are being composed to solve new problems

Data Scientist → Puppy Detector → Ball Detector → Cuteness Detector → Cute!

Wrong but helpful… Still Correct
Models are being composed to solve new problems

Composition

Data Scientist

Puppy Detector

No

Reasonable Improvement

Cuteness Detector

Not Cute!

Degradation in accuracy

Ball Detector

Yes
Composition

Models are being composed to solve new problems

Need to track composition and validate **end-to-end accuracy**.

Need **unit** and **integration** testing for models.
Model Development

Data Collection → Cleaning & Visualization → Feature Eng. & Model Design → Training & Validation

Training Pipelines → Trained Models

Live Data → Validation

Data Scientist → Data Engineer
Goal: make predictions in ~10ms under bursty load

Complicated by Deep Neural Networks ➔ New ML Algorithms and Systems
Inference Technologies

Prediction Service

Logic

Query

Prediction

End User Application

Feedback

Data Engineer

PredictionIO

mxnet

TensorFlow
Incorporating Feedback

- **Model updates**: retrained as new data arrives
  - Periodically: leverage batch processing and validation
    - Model could be out-of-date for extended periods of time
  - Continuously (online learning): most fresh model
    - Needs validation, learning rates? … complicated

- **Feature updates**: new data may change features
  - Example: update click history for a user → new predictions
  - Can be more robust than online learning
Feedback Cycles

- Models can **bias the data** they collect
  - Example: content recommendation
  - Future models may reflect earlier model bias

- **Exploration – Exploitation Trade-off**
  - **Exploration**: observe diverse outcomes
  - **Exploitation**: leverage model to take predicted best action

- **Solutions**
  - **Randomization (ε-greedy)**: occasionally ignore the model
  - **Bandit Algorithms/Thompson Sampling**: optimally balance exploration and exploitation → active area of research
We will cover each phase in more detail throughout the semester but this week we focus on managing the entire process.
Objectives For Today

- Introduce the machine learning lifecycle
  - Challenges and Opportunities
  - Science vs Engineering

- Review Key Concepts in Readings
  - Hyperparameters
  - Model Pipelines, Features, and Feature Engineering
  - Warm Starting and Fine Tuning
  - Feedback Loops, Retraining and Continuous Training

- Important Context for Papers and what to expect.
Reading for the Week

- **Hidden Technical Debt in Machine Learning Systems**
  - NeurIPS’15, widely cited
  - Provides an overview of the challenges from Google

- **TFX: A TensorFlow-Based Production-Scale Machine Learning Platform**
  - KDD’17, now part of https://www.tensorflow.org/tfx (sort of)
  - Google’s solution to the challenges in the first paper

- **Towards Unified Data and Lifecycle Management for Deep Learning**
  - ICDE’17, Video Demo
  - An alternative database community solution
Related Systems Efforts

- Doing Machine Learning the Uber Way: Five Lessons From the First Three Years of Michelangelo
- Introducing FBLearner Flow: Facebook’s AI backbone
- KubeFlow: Kubernetes Pipeline Orchestration Framework
- DeepBird: Twitter’s ML Deployment Framework
- MLflow: A System to Accelerate the Machine Learning Lifecycle
- Data Engineering Bulletin on the Machine Learning Lifecycle
  - Full disclosure: I was the editor
Hidden Technical Debt in Machine Learning Systems

- **Technical Debt:** long term development and maintenance costs incurred by expedient design decisions

- **Key Idea:** machine learning deployments often incur substantial technical debt (compared to traditional software)

- **Contribution:** this paper characterizes the forms of technical debt and alludes to possible compensating actions
TFX: A TensorFlow-Based Production-Scale Machine Learning Platform

- Describes solutions to many of the problem outlined in the technical debt paper.

- **Key Idea:** Adapt best practices for software development to address machine learning lifecycle
  - empathetic to the reality of “machine learning developers”

- **Contributions:** actual system, interesting ideas around data and model validation, schema enforcement, and meaningful errors.
Towards Unified Data and Lifecycle Management for Deep Learning

- Describes a system (ModelHub) for managing, querying, and manipulating models and their related metadata.

- **Key Idea(?):** Model lifecycle management combines code and data (parameters) → a natural API would then combine version control commands with SQL-like querying.

- **Solution:** Combines a git-like client API with a SQL-like querying interface to enable basic actions and more complex queries.
  - Leverages optimizations to store model weights more efficiently.
  - (necessary?)
What to think about when reading

- How does the work differentiate between engineering and research challenges?
- What innovations in machine learning are needed?
- What are the key research challenges proposed and addressed?
- Are the proposed solutions too opinionated
  - Would they require top down mandates for adoption?
  - Would you use these systems?
  - Are they sufficiently flexible to support innovation?
Done!