Software Engineering for Learning-Enabled Systems

Michigan State University
College of Engineering

CSE435: Software Engineering

Guest Lecturer

Michael Austin Langford

November 10, 2021
Introduction

- Teaching Assistant, Ph.D. student (5th year)
  - Systems for Trusted AI / Robustness & Resiliency
- M.S./B.S. Computer Science
  - Computer Vision & Graphics
- Prior experience
  - Private Industry, Military
- Areas of interest
  - Software Engineering for AI
  - AI for Software Engineering
  - Evolutionary Computation
  - Autonomous Systems with Deep Learning

Michael Austin Langford
Introduction

Motivation

- **Artificial Intelligence** is central to autonomy.
  - Autonomous systems must *predict, plan, and categorize* their surroundings
  - Almost *one-third of IT professionals* report businesses using AI [Bishop 2021]
  - Autonomous solutions *spurred by global lockdown* restrictions [Geylani 2020]
  - *Consumer anxiety* over widely publicized failures of AI services [Slingerland 2021]
Introduction

Overview

Traditional Software Systems
- Well-defined *sequence of instructions*
- Common *design patterns* exist
- Logic can be verified by *formal methods*

Learning-Enabled Systems (LESs)
- Optimized/refined to *fit given data*
- Enables solutions to *complex tasks*
- Decision logic *not easy to interpret*

*What software challenges are unique to Learning-Enabled Systems?*
Introduction

Objectives

- Introduce *concepts* of machine learning and deep learning
- Introduce the *limitations* of deep learning and data-driven systems
- Introduce *software engineering challenges* for LESs
Learning-Enabled Systems (LESs)

Learning Methods

- **Types of Learning**
  - **Supervised Learning**
    - Training examples are labeled with expected outcomes
  - **Unsupervised Learning**
    - Data is automatically organized by visible characteristics

- **Modes of Learning**
  - **Offline Learning**
    - System is trained at design time and fixed at run time
  - **Online Learning**
    - System learns and can change behavior at run time

---

Netflix Recommendation Engine
(Unsupervised Learning)
[Chong 2020]

Zillow Zestimates
(Supervised Learning)
[Zillow 2021]

Clusters users together based on preferences.

Real estate estimates by analyzing past listings.
Learning-Enabled Systems (LESs)
Supervised Learning

- **Labeled training data**
  - Collection of *data pairs* comprised of *example inputs* ($x$) and *expected outputs* ($t$)
  - **Examples:**
    - **Real Estate Estimates** ($f: \mathbb{R}^n \rightarrow \mathbb{R}^1$)
      - **Input ($x$):** [zip, acreage, floorspace, etc.]
      - **Output ($t$):** dollar value
        - ($x = [48910, 0.25, 1025, ...], t = \$95000.00$)
        - ($x = [48912, 0.33, 1530, ...], t = \$210000.00$)
        - ($x = [48910, 0.20, 950, ...], t = \$75000.00$)
    - **Handwritten Numbers** [Deng 2012] ($f: \mathbb{R}^{1024} \rightarrow \mathbb{R}^{10}$)
      - **Input ($x$):** image
      - **Output ($t$):** digit label
        - ($x = \begin{bmatrix}0\end{bmatrix}, t = \text{“0”}$)
        - ($x = \begin{bmatrix}3\end{bmatrix}, t = \text{“3”}$)
        - ($x = \begin{bmatrix}2\end{bmatrix}, t = \text{“2”}$)
        - ($x = \begin{bmatrix}4\end{bmatrix}, t = \text{“4”}$)
        - ($x = \begin{bmatrix}3\end{bmatrix}, t = \text{“3”}$)
        - ($x = \begin{bmatrix}4\end{bmatrix}, t = \text{“4”}$)

**Learning Task:** learn relationship between input and output ($f(x) = t$)
Learning-Enabled Systems (LESs)

Data Considerations

- **Generalization Assumption**
  - Assumes that a *general trend can be derived* from a finite set of examples

- **“Garbage In, Garbage Out”**
  - Data-driven systems are highly *dependent on quality of input & training data*

- **Cognitive Bias** [Calikli 2010]
  - Collected data can be *misrepresentative of reality*
    - *Availability Bias*: emphasis given to *most readily available data*
    - *Confirmation Bias*: emphasis given to *data with positive results*
    - *Selection Bias*: emphasis given to an *incorrect data distribution*
Learning-Enabled Systems (LESs)

Learning Model Considerations

- Occam’s Razor (Law of Parsimony)
  - Favor solutions with the fewest assumptions
  - Simpler solutions are typically easier to test and less sensitive to data bias

- Model Complexity
  - How complex should our model be?
  - Underfitting data leads to poor predictions
  - Overfitting data leads to poor generalizations

How can we determine if a model generalizes well?
Learning-Enabled Systems (LESs)

Machine Learning Algorithms [Bishop 2006]

- Algorithms that can refine system behavior in response to data
  - **Linear Regression**
    - Linear relationship between input variables and observed output values
  - **Decision Trees**
    - Tree of decisions represented as nodes with branching outcomes
  - **Support Vector Machines (SVMs)**
    - Hyperplane that can separate data according to relevant data features.
  - **Artificial Neural Networks (ANNs)** [Haykin 1998]
    - Multi-layered perceptron inspired by neuron activity (linear transformations + activations).
Artificial Neural Networks (ANNs) [Haykin 1998]

Brief History

- **Single Layer Perceptron** (1958)
  - *Frank Rosenblatt*, Cornell Aeronautical Laboratory
  - Method to classify *linearly separable* data

- **Multilayer Perceptron** (1960s-1980s)
  - Classifies *non-linearly separable* data by linking perceptrons

- **Dark period** (1990s)
  - Methods to train ANNs are *difficult and computationally expensive*
  - SVMs gain popularity

- **Deep Neural Networks** (1998 to Present)
  - More *efficient methods to train* deep networks (*supported by GPUs*)
  - Godfathers of Deep Learning: *Geoffrey Hinton, Yann LeCun*, and *Yoshua Bengio*
  - *Dozens of layers*, millions of trainable parameters
Artificial Neural Networks (ANNs) [Haykin 1998]

Linearly Separable Data (Single Layer Perceptron)

- Data is **easiest to classify** when *linearly separable*

1. Define **linear transformation** (standard form)
   \[
   f(x_1, x_2) = x_1 - 2x_2 + 2
   \]

2. Define non-linear **activation function**
   \[
   \sigma(f(x_1, x_2)) = \{f(x_1, x_2) \geq 0 \? 0 : 1\}
   \]

3. Combine to create **single layer perceptron**
   \[
   y = \sigma(f(x_1, x_2)) = \{x_1 - 2x_2 + 2 \geq 0 \? 0 : 1\}
   \]

4. **Classify** based on final output
   \[
   y \geq 0 \? \bigcirc : \times
   \]
Artificial Neural Networks (ANNs) [Haykin 1998]

Non-linearly Separable Data

- Classic XOR problem [Karajgi 2020]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_1 \oplus x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

No single line exists that can separate X’s and O’s.

Output of XOR function is not linearly separable.

Can be separated by chaining together multiple linear separators.
Artificial Neural Networks (ANNs) [Haykin 1998]

Non-linearly Separable Data (Multilayer Perceptron)

- Classic **XOR** problem

\[ f_{1a} \]
\[ f_{1b} \]

- **Multiple Layers of Perceptrons**
  - First layer: determines which side point resides (both vertically and horizontally)
  - Second layer: determines which corner

\[ y = \sigma(W_2 \sigma(W_1 x)) \]

Weight constants determine where lines are drawn.

Chain of Matrix Computations
Artificial Neural Networks (ANNs) [Haykin 1998]

Deep Learning [Goodfellow 2016]

- More complex datasets and problem spaces require many layers
  - Each consecutive layer discerns higher order relationships
- Deep Neural Networks (DNNs)
  - Not unusual to see DNNs with dozens of layers and millions of weight values

How do we figure out the proper number of layers?

What about the size of each layer?

How do we figure out the proper weight values?

What about network topology?

There are numerous design considerations & configuration variables (hyperparameters).
Deep Neural Networks (DNNs) [Goodfellow 2016]

Training Process

- **Analogy**: network of branching pipes with valves
  - Each valve can open/close to adjust flow. How to find best valve settings?
- **Weight optimization (gradient descent)**
  - “Training” phase *tweaks weight values* to determine settings with *minimal error*
- **Gradient Descent with Back Propagation**
  - Start with random weight settings (*weight initialization*)
  - Feed in some inputs with known expected outputs (*labeled training data*)
  - Compare actual output of network to expected output (*objective loss function*)
  - Compute *gradient derivative* to determine direction to minimize error
  - Repeat at each layer in reverse to adjust weights (*backpropagation*)
Deep Neural Networks (DNNs) [Goodfellow 2016]

Training Algorithm

- Train by **comparing error** for each example in dataset over **multiple epochs**

```
train_dnn(dnn, dataset, criterion, n_epochs)
1  dnn = init_weights(network)
2  optimizer = init_optimizer(dnn)
3  for i in n_epochs:
4      for x, t in dataset:
5          y = dnn(x)
6          error = criterion(y, t)
7          dnn = backpropagation(dnn, optimizer error)
```

- Initialize with *random* weight values.
- Each *epoch* is an iteration over the **entire dataset**.
- **Compare DNN output to ground truth** for each given dataset entry.
- Compute gradient of error w.r.t. weights and **update weights**.
Deep Neural Networks (DNNs) [Goodfellow 2016]

Training Optimization

- **Gradient Descent Optimization**
  - Assumes a smooth *concave gradient landscape* to global minimum
  - *Millions of weights* are involved (*numerous dimensions*)
  - Landscape is *extremely complex*
  - The *Exploration vs. Exploitation* dilemma

- **Stochastic Gradient Descent (SGD)**
  - Introduce *randomness* to weight adjustments

- **Adaptive Gradient Descent Methods (Adam, Adadelta, RMSprop)**
  - *Changing magnitude* of weight adjustments over time
Deep Neural Networks (DNNs) [Goodfellow 2016]

Training Progress

- Weights for DNN continue to be adjusted over a number of *epochs*
- Plot of loss over the number of epochs follows a *decaying curve*
- Training terminates when *objective loss* (error) *converges to a minimum*

*When training phase completes, DNN is good at handling training examples.*

*But what about overfitting?*

*Loss is high with random weights*

*Training has converged. Diminishing returns.*

*How do we determine if the DNN is any good at non-training data?*
Deep Neural Networks (DNNs)  

Validation

- **Test data** can be used to estimate generalizability
  - Data must be *independent and identically distributed* (i.i.d.)
  - Must be careful about *data contamination*
- **Validation data** may be used to optimize hyperparameters
  - Typically taken from a subset of training data (*k-fold cross validation*)
- **Dataset partitioning**
  - Divide dataset into *training*, *validation*, and *test* data (80 / 20 split)

Assumes that test data is representative of real data.
Deep Neural Networks (DNNs) [Goodfellow 2016]

Network Architectures

- Numerous architectures (network topologies) have been devised
- **Feedforward Neural Networks**
  - Most common, data follows a *direct path from input to output* layer
- **Convolutional Neural Networks (CNNs)**
  - Tailored for processing *image data* more efficiently
  - LeNet, AlexNet, VGG, ResNet, Inception, etc.
- **Recurrent Neural Networks (RNNs)**
  - Tailored for processing *sequences of data* (time series)
  - Loops output of layer back into input (*state memory*)
  - LSTMs, GRUs, MGUs, etc.
Deep Neural Networks (DNNs) [Goodfellow 2016]

Convolutional Neural Networks (CNNs)

- Normal DNNs are **fully-connected**
  - Perceptrons are a *linear combination of all input variables*
  - Images contain *many pixels* (32 x 32 image = 1024 input variables)
- CNNs are only **locally-connected**
  - Performs a *linear combination of neighboring pixels* (for each pixel)
  - Output from each unit in layer is a **feature map** of image

**Image Convolution**

- Common *image processing* filtering operation
- Applies a moving window *kernel* to each pixel
- Convolutions can be used to **enhance/isolate features** of image
Convolutional Neural Networks (CNNs)

Common Tasks

- **Image Recognition** [Krizhevsky 2009]
  - Images are *classified into categories* based on imaged object
  
  ![Image Classification](image)

- **Predictions** are given as a *probability distribution* over possible categories

- **Performance** is measured by the *accuracy* (% correct) of predictions
Convolutional Neural Networks (CNNs)

Demonstration

- **ConvNetJS** [Karpathy 2014]
  - Deep learning in your browser
  - **Visualizes each layer** of a CNN in real time
  - Completely implemented in **javascript**

Useful tool for understanding CNNs.

http://cs.stanford.edu/people/karpathy/convnetjs/
Convolutional Neural Networks (CNNs)

Common Tasks

- **Object Detection** [Lin 2017] [Waymo 2020]
  - Multiple objects are located within an image with bounding boxes
  
  - Performance is measured by the precision and recall of detections

![Object Detection Diagram](image)
Convolutional Neural Networks (CNNs)

Example Applications

- **Automatic Machine Translation** (Google Translate) [Good 2015]
  - Text is located, translated, and *rendered in target language*

![Source Image (Swedish)](image1)

![Isolated Words](image2)

![Language Translation](image3)

![Rendered Translation](image4)
Convolutional Neural Networks (CNNs)

Example Applications

- **Image-to-Image Translation (Pix2Pix)** [Isola 2016] (github)
  - *Transforms the appearance* of an image from one form to another

- B&W to Color
- Labels to Facade
- Day to Night
- Edges to Photo

*These techniques require many images in both the source and target domain.*

*Be cautious of hype. Often examples are “cherry picked.”*
Convolutional Neural Networks (CNNs)
Example Applications

- **Image-to-Image Translation (Pix2Pix)** [Isola 2016](https://github.com) (github)
  - **Transforms the appearance** of an image from one form to another

- **Ideal Example: Sketch to Cat**
- **Uncanny Valley: Sketch to Human?**
- **Novel Example: Sketch to Beholder**

*These techniques require many images in both the source and target domain.*

*Be cautious of hype. Often examples are “cherry picked.”*
Convolutional Neural Networks (CNNs)

Example Applications

- **StyleGAN** (Nvidia) [Karras 2019] (github)
  - Generates photorealistic *synthetic faces in high resolution*
  - These are not real people.

- **DeepFakes** [Toews 2020]
  - Realistic media generated from *inauthentic sources*
  - Anyone can be Obama. [Choi 2017]

*DeepFakes have potential to devastate society.*
Software Engineering Challenges

Maturity of the Field

- **Research has made impressive leaps** with DNN applications
  - Rate of ML papers published **surpasses Moore’s Law** [Dean 2019] (arxiv)
  - Provides interesting **solutions to hard problems**

- **From Theory to Practice**
  - Research typically targets **toy problems** and **simple benchmarks**
  - Issues with **reproducibility** [Ivie 2018]
  - **Only 40%** IEEE publications have practices to ensure reproducibility [Goodrich 2021]

---

Be careful about overpromising results.
Software Engineering Challenges

**Design Challenges**

- **Hyperparameters** – there are numerous *design variables*
  - **Dataset variables**: data format, dataset distribution, etc.
  - **Network variables**: # of layers, size of layers, activation types, etc.
  - **Training variables**: # of epochs, loss functions, optimization methods, etc.

- **Traditional software has the benefit of decades of experience**
  - **Design patterns** and **best practices** have been established

*We are still establishing design patterns and best practices for Learning-Enabled Systems.*
Software Engineering Challenges

Validation Challenges

- Validations against an i.i.d. dataset is not enough
  - Assumes that all relevant run-time scenarios have been captured
  - Should only be treated as one piece of a more complete test strategy

- AI systems must be able to handle deviations from known data
  - Beyond accuracy, precision, and recall, we need to know the robustness of DNNs to new contexts

- Users need more information to trust AI
  - For real applications, we need more than a simple prediction
  - We need to know the margin of error (confidence intervals)
  - We need to understand how the final output was derived (interpretability)
Software Engineering Challenges

**Trusted AI**

- **Trust** has become a significant challenge for *safety-critical applications*
- Industry has responded by establishing **Trusted AI Guidelines**

<table>
<thead>
<tr>
<th>Aerospace Corp. Trusted AI Framework</th>
<th>Deloitte Trustworthy AI</th>
<th>IBM Trustworthy AI</th>
<th>Microsoft Responsible AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>Privacy</td>
<td>Privacy</td>
<td>Privacy &amp; Security</td>
</tr>
<tr>
<td>Confidence/Uncertainty</td>
<td>Responsible/Accountable</td>
<td>Robustness</td>
<td>Accountability</td>
</tr>
<tr>
<td>Adversarial Robustness</td>
<td>Safe/Secure</td>
<td>Transparency</td>
<td>Reliability &amp; Safety</td>
</tr>
<tr>
<td>Interpretability</td>
<td>Robust/Reliable</td>
<td>Explainability</td>
<td>Transparency</td>
</tr>
<tr>
<td>Familiarity</td>
<td>Transparent/Explainable</td>
<td>Fairness</td>
<td>Fairness</td>
</tr>
<tr>
<td>Fairness</td>
<td>Fair/Impartial</td>
<td></td>
<td>Inclusiveness</td>
</tr>
</tbody>
</table>

[Slingerland 2021] [Deloitte 2021] [IBM 2021] [Microsoft 2021]

Looking beyond the hype, we need more rigorous standards.
Software Engineering Challenges
Misunderstandings

- **DNNs can easily latch onto superficial details**
  - Typical DNNs *do not derive any semantic model* of the problem space
  - DNNs latch onto *surface statistical regularities* (patterns that appear often)

- **Example Scenario**
  - All training images of cats contain a specific cloud pattern in the background
  - DNN associates “cloud pattern” with “cats”
  - DNNs simply *isolate the features most useful for separating* classifications
  - *No sophisticated understanding* of what a cat is beyond pattern matching

*It is easy for the average user to project more intelligence to DNNs.*
Software Engineering Challenges

Robustness

- **Uncertainty** remains with how robust DNNs are to new data
  - Robustness: how well DNN performs as inputs deviate from training/validation
  - Not safe to assume a small deviation will result in similar results

- **Adversarial Examples** [Szegedy 2013]
  - Inputs specifically tailored to exploit and confuse a DNN
  - Can be generated with humanly imperceptible deviations

DNNs can latch onto patterns beyond our senses.

Robust DNNs are less sensitive to these sorts of insignificant disturbances.
Software Engineering Challenges
Understanding Adversarial Examples

- **Adversarial examples** are created in a similar manner to training DNNs
  - *Training DNN*: weights adjusted to minimize error for a fixed input
  - *Adversarial Examples*: weights are fixed and input is adjusted to maximize error with constraint to keep changes to input minimal

- **Defenses against adversarial examples**
  - *Adversarial training*: add adversarial examples to training set
    - *Whack-a-mole approach*, not very effective
  - *Training Noise*: methods to add noise to training to make DNN less sensitive
  - *Input Denoising*: methods to remove noise from inputs at run time

Adversarial attacks remain a real threat to any outward-facing DNN.
Software Engineering Challenges
Environmental Uncertainty

- **Beyond adversarial attacks** there remains uncertainty with how DNNs respond to *unseen environmental phenomena* [Langford 2021]

*When a CNN has only been trained/validated with images in clear conditions, how will it react to a raindrop on the camera lens?*

*It is not possible to include every real world corner case into a dataset!*
Software Engineering Challenges

Interpretability

- DNNs contain *many hidden layers* that are *trained as one*
  - Weights for *all layers are adjusted end-to-end* to minimize training error
  - The *output of a single layer in isolation makes no sense* without the other layers
  - CNN layers will not isolate *semantically relevant features* like “eyes” “mouth”

*How do we know what a DNN is looking at when it makes its decisions?*

- **Grad-CAM** [Selvaraju 2016]
  - *Highlights region* of image that had *most significant impact on final output*

*Users need these sorts of tools to trust DNNs.*
Software Engineering Challenges
Monitor and Control

- Run-time environments will present *new and unexpected conditions*

  - Systems must *continuously monitor capability* of learning components
  - Systems must *assess the trustworthiness* of learning components
  - Systems must *implement failsafes* for untrusted states

*How can we avoid these types of incidents?*

*How do we assess trustworthiness at run time?*
Software Engineering Challenges
Monitor and Control

- **Uber Accident Timeline Review** [NTSB 2019]

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Classification</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5.6 to 2.7</td>
<td>[Vehicle/Other]</td>
<td>Tracking history is reset with each change in classification</td>
</tr>
<tr>
<td>-2.6 to 2.5</td>
<td>[Bicycle]</td>
<td>Classification stabilizes. Path updated to traveling left through lane</td>
</tr>
<tr>
<td>-1.5</td>
<td>[Other]</td>
<td>Classification changes. Path updated to static</td>
</tr>
<tr>
<td>-1.2</td>
<td>[Bicycle]</td>
<td>Classification changes. Path updated to static but on SUV path.</td>
</tr>
<tr>
<td>-1.2 to 0.2</td>
<td>[Bicycle]</td>
<td>ADS cannot steer around, begins slowing down, notifies driver</td>
</tr>
<tr>
<td>-0.02</td>
<td>[Bicycle]</td>
<td>Driver takes control of steering, disengages ADS</td>
</tr>
</tbody>
</table>

**IMPACT**

+0.7 | Driver brakes.

Learning-Enabled Systems must include online risk and trust assessment.

Tracking history reset each time classification changed.
Software Engineering Challenges
Monitor and Control

- **Behavior Oracles** to assess impact of *known unknowns* [Langford 2021]
  - *Known unknowns* – phenomena with *unknown impact* but can be simulated
  - *Simulation* to introduce the phenomena into existing datasets
  - *Evolutionary Computation* to generate relevant contexts of phenomenon
  - *Machine Learning* to create a generalized predictive model

**Example** How will DNN respond to dust clouds?

**Automated Assessment** (Simulation + Evolution)

**Image Input**

**Behavior Oracle**

**Run-Time Assessment**

- **Perceived Context** dust density: 0.22
dust intensity: 0.19

  **Inferred Behavior** “little impact”

- **Perceived Context** dust density: 0.60
dust intensity: 0.59

  **Inferred Behavior** “compromised”
Software Engineering Challenges

Monitor and Control

- Behavior oracle in action [Langford 2021]
  - Enable assessment of run-time context
  - Outputs the perceived context to help locate the source of interference
  - Outputs inferred behavior to determine capability of learning component

Behavior oracles can help determine applicability of learning components.
Conclusion

Summary

▪ Advances in Deep Learning have enormous promise.
  ▪ We live in exciting times, but **be careful about hype** and overpromising
  ▪ A huge leap from research on **toy problems to real-world** applications
  ▪ **Software engineering is difficult** for learning-enabled systems!

**State-of-the-art systems are good at what they know but are bad at knowing what they know.**

▪ **Emphasis is shifting to bi-directional trust for AI systems.**
  ▪ Tools/techniques to **enable users to understand** AI decision-making
  ▪ **AI must understand limitations** and when **human expertise** is required
References

References

- **[IBM 2021]** Trustworthy AI, IBM, 2021.
References