# Machine Learning and Photonics

## Week 1: Part A

Ergun Simsek

University of Maryland Baltimore County
*simsek@umbc.edu*

Introduction to ENEE 691 MLP

January 31, 2023

# Instructor and Course

**Instructor:** Dr. Ergun Simsek
**E-mail:** simsek@umbc.edu
**Office:** ITE 325K
**www:** `https://www.csee.umbc.edu/~simsek/`

**Course Webpage:** `https://dil.umbc.edu/mlp`
**Lecture Day and Time:** Tuesdays, 10 am – 12:30 pm
**Mode of Instruction:** Online
**WWW:** Online meetings will be held in Blackboard Collaborate
**Office Hours:** Tuesdays 1 pm – 3 pm

# Course Description

**Course Description:** This 3-credit graduate course provides an introduction to (i) using machine learning (ML) to solve forward problems such as scattering and wave propagation, (ii) inverse photonics device design, and (iii) how photonics can help with accelerating some generic tasks in ML such as training and inference of high-dimensional data. Students will use traditional ML algorithms to solve forward problems and design various types of neural networks to solve different types of inverse photonic problems.

# Learning Objectives

- Develop some fundamental skills numerical methods (e.g. differentiation, integration, and interpolation)
- Apply exploratory data analysis to a given photonic dataset
- Understand the main differences among fundamental ML methods, choose the appreciate ones to solve basic forward problems in photonics, implement them, and evaluate their accuracies,
- Create physics-inspired neural networks to solve both forward and inverse problems in photonics,
- Understand how adjoint methods and generative adversarial networks design photonic devices,
- Develop a fundamental understanding of how photonics can enable faster computations, and
- Identify the main challenges of ML for solving photonic problems and using photonics for ML research.

# Required Text

No textbook is required. Lecture notes will be provided on a weekly basis. The papers that we will review can be downloaded from this page.

If you need a reference book on Phyton programming, ML or Photonics side, please consider these recommendations:

- Python Programming and Numerical Methods: A Guide For Engineers And Scientists by Kong, Siauw, and Bayen
- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Geron Aurelien
- Fundamentals of Photonics 2nd Edition by Bahaa E. A. Saleh and Malvin Carl Teich

However, we will read lots of papers which can be found at `https://dil.umbc.edu/mlp/` along with some other important materials and links.

# Grading

- Participation: %20
- Homework (4-6 assignments): %30
- Project I (Solving a forward problem with ML): %20
- Project II (Solving an inverse design problem): %30

Note that

- Homework assignments and projects can be completed using either Matlab or Python. The final version of the code you submit should be in good working condition, e.g. it should work from beginning to end without the instructor's or TA's input.
- The use of git and GitHub is highly recommended.
- Group projects are more than welcome.

# How about you?

I will call your name according to the list of participants.
Please unmute yourself and tell us

- What are you studying? (Degree and Program)
- How many years have you been studying at UMBC?
- Are you good at MATLAB or Python Programming?
- Have already taken courses/done research on Photonics or Machine Learning?
- Any questions you have about this class

# Some other Notes

- Since this will be an online class, please turn on your webcams and mute your microphones while you are not talking.
- **Late Work Policy:** For late homework/project submissions, 10 points will be deduced for each day late and late submissions will not be accepted if overdue by more than 6 days.
- **Attendance Policy:** Regular class attendance is required and necessary for students to understand many of the topics covered. Students must be on time for class. If missed a class, it is the responsibility of the student to find out the materials covered.
  - If you are going to miss a class, please inform the instructor in advance and do not forget to watch the lecture recording later.
- No class on March 28th (I will be attending a conference).

# Machine Learning and Photonics

## Week 1: Part B

Ergun Simsek

University of Maryland Baltimore County
*simsek@umbc.edu*

Why Should We Study Machine Learning and Photonics?

January 31, 2023

# What is Photonics?

*An interdisciplinary field of study that deals with the generation, detection, transmission, manipulation, and, control of light.*

Two questions:

- Where do we use photonics?
- What are the current challenges of photonics (think from an application point of view)?

# Most Important or Fundamental

**Photonics Components**

- Lasers
- Photodetectors
- Fiber optics
- Modulators
- Sensors
- Amplifiers
- Optical Frequency Combs
- Mirrors, lenses, and filters

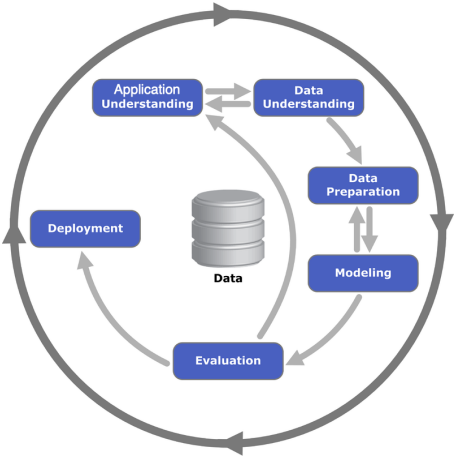
**Materials & Elements**

- Metals
- Semiconductors
- Dielectrics
- Nonlinear optical materials
- Metamaterials and photonic crystals
- Quantum dots

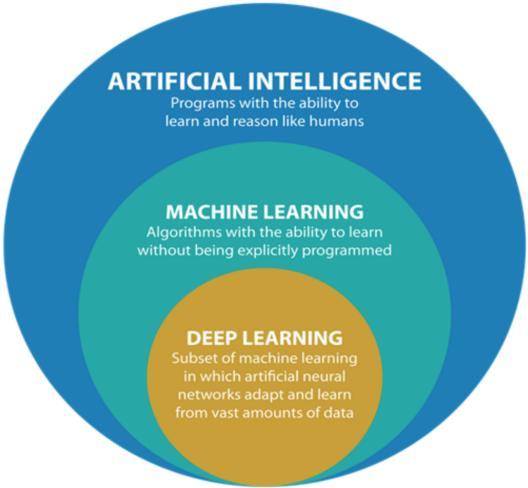
# What is Data Science?

*Data science is a field that combines knowledge and tools from computer science, statistics, and domain expertise in order to extract insights and knowledge from data.*

Life-cycle of a typical data science project.



# What is Machine Learning?




*Machine Learning is a subcategory of artificial intelligence, that refers to the process by which computers develop pattern recognition, or the ability to continuously learn from and make predictions based on "data", then make adjustments without being specifically programmed to do so.*

# The Goal of ML

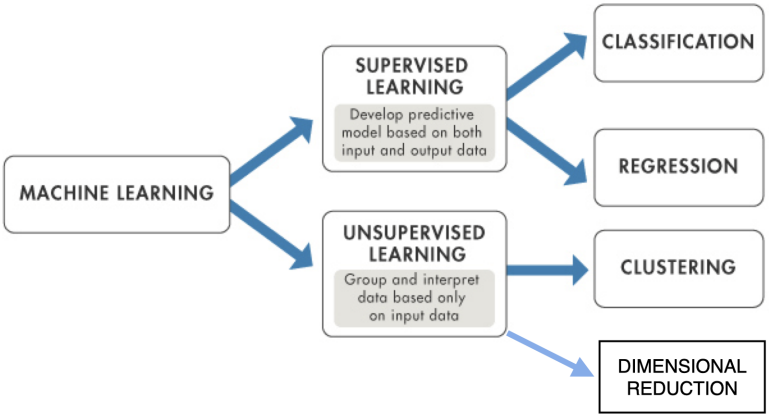
The goal of machine learning is to design general-purpose methodologies to *extract valuable patterns from data*, ideally without much domain-specific expertise.

Learning can be introduced as a method to automatically find patterns and structure in data by optimizing the parameters of the model.

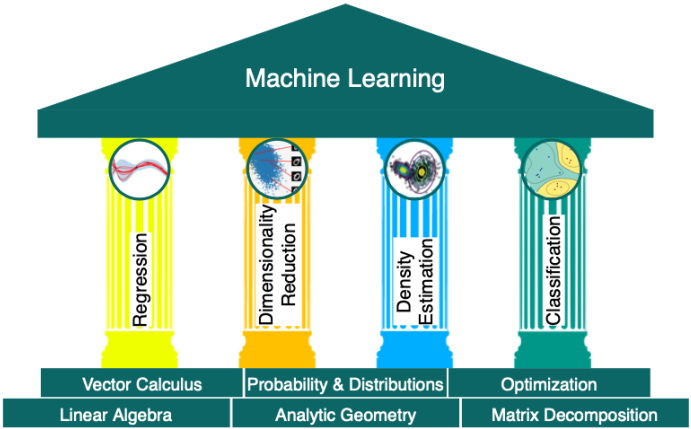
## Note the Difference

- Numerical optimization is focused on finding the best solution to a mathematical problem.
- ML is focused on training a computer system to make predictions or decisions based on data.

# Supervised vs Unsupervised Learning



# We Need Math!




Source: M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*, Cambridge University Press (2020)

# Before You Start Implementation

You should consider

- Power and Expressibility
- Interpretability
- Ease of Use
- Training Speed
- Prediction Speed

# What can ML do for Photonics?

- Photonic device performance prediction (before we fabricate or simulate them)
  - Linear Algebra based linear regression
  - Neural network based linear regression
- Categorizing photonic devices
  - k-Nearest neighbors
  - Random Forests
- Understanding the influence of design parameters on device performance (PCA, SVM)
- Inverse photonic design
  - Adjoint Method
  - Back to Back Neural Nets
  - Generative adversarial networks (GANs)
- Error reduction in optical communication
- Higher accuracy in optical sensing

# What can Photonics do for ML?

Photonics can help to improve the speed, accuracy, and scalability of machine learning algorithms, making them more useful for a wide range of applications.

For example,

- Optical computing
- Neuromorphic computing

# Tentative Schedule

- Week-1: Math Foundations (Series and Root finding)
- Week-2: Math Foundations (Numerical Differentiation and Integration)
- Week 3: Numerical Optimization
- Week 4: Linear Regression
- Week 5: Interpolation
- Weeks 6-8: Logistic Regression, kNNs, and Random Forests
- Weeks 8-11: Neural Networks
- Weeks 12-13: Inverse Design with the Adjoint Method
- Weeks 14-15: Photonics for ML

# Machine Learning and Photonics

## Math Foundations: Series

Ergun Simsek

University of Maryland Baltimore County
*simsek@umbc.edu*

The Power of Taylor Series

January 31, 2023

# Math is full of surprises!

What if I tell you that you can use this expression to calculate $\sin(x)$

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} \tag{1}$$

Most of the functions that we use every can be computed by sums of functions that are easy to compute, such as polynomials.

The starting point of all is the powerful Taylor's Theorem

# Sequence vs. Series

A **sequence** is an ordered set of numbers denoted by the list of numbers inside parentheses.

For example, $s = (s_1, s_2, s_3, \cdots)$ means $s$ is the sequence $s_1, s_2, s_3, \cdots$ and so on.

Note that here "ordered" means that $s_1$ comes *before* $s_2$, not that $s_1 < s_2$.

A **series** is the sum of a sequence up to a certain element.

An **infinite sequence** is a sequence with an infinite number of terms, and

An **infinite series** is the sum of an infinite sequence.

# Taylor Series

A **Taylor series expansion** is a representation of a function by an infinite series of polynomials around a point. Mathematically, the Taylor series of a function, $f(x)$, is defined as:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)(x-a)^n}{n!},$$

where $f^{(n)}$ is the $n^{\text{th}}$ derivative of $f$ and $f^{(0)}$ is the function $f$.

# Taylor Series Expansion Example

Let's compute the Taylor series expansion for $f(x) = 5x^2 + 3x + 5$ around $a = 0$, and $a = 1$ and verify that $f$ and its Taylor series expansions are identical.

First compute derivatives analytically:

$$\begin{aligned} f(x) &= 5x^2 + 3x + 5 \\ f'(x) &= 10x + 3 \\ f''(x) &= 10 \end{aligned}$$

# Taylor Series Expansion Example (Cont...)

Around a = 0:

$$f(x) = \frac{5x^0}{0!} + \frac{3x^1}{1!} + \frac{10x^2}{2!} + 0 + 0 + \cdots = 5x^2 + 3x + 5$$

Around a = 1:

$$\begin{aligned} f(x) &= \frac{13(x-1)^0}{0!} + \frac{13(x-1)^1}{1!} + \frac{10(x-1)^2}{2!} + 0 + \cdots \\ &= 13 + 13x - 13 + 5x^2 - 10x + 5 = 5x^2 + 3x + 5 \end{aligned}$$

Note: The Taylor series expansion of any polynomial has finite terms because the $n^{\text{th}}$ derivative of any polynomial is 0 for $n$ large enough.

# Taylor Series Expansion Example 2

Let's write the Taylor series for $\sin(x)$ around the point $a = 0$.

$$f(x) = \frac{\sin(0)}{0!}x^0 + \frac{\cos(0)}{1!}x^1 + \frac{-\sin(0)}{2!}x^2 + \frac{-\cos(0)}{3!}x^3 + \frac{\sin(0)}{4!}x^4 + \frac{\cos(0)}{5!}x^5 + \cdots .$$

The expansion can be written compactly by the formula

$$f(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!},$$

which ignores the terms that contain $\sin(0)$ (i.e., the even terms).

# Taylor Series Expansion Example 2 (Cont...)

Let's use Python to plot the sin function along with the first, third, fifth, and seventh order Taylor series approximations. Note that this is the zeroth to third in the formula given earlier.

```
:
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')
```

# Taylor Series Expansion Example 2 (Cont...)

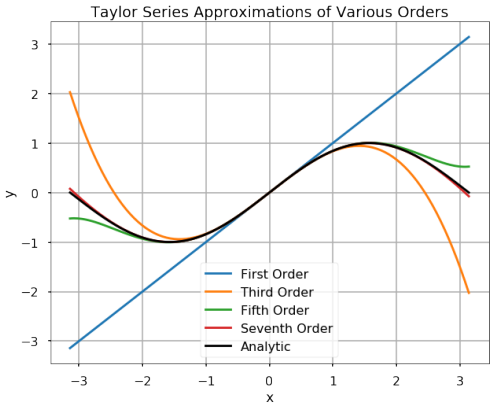
```
x = np.linspace(-np.pi, np.pi, 200)
y = np.zeros(len(x))

labels = ['First Order', 'Third Order', 'Fifth Order', 'Seventh Order']

plt.figure(figsize = (10,8))
for n, label in zip(range(4), labels):
    y = y + ((-1)**n * (x)**(2*n+1)) / np.math.factorial(2*n+1)
    plt.plot(x,y, label = label)

plt.plot(x, np.sin(x), 'k', label = 'Analytic')
plt.grid()
plt.title('Taylor Series Approximations of Various Orders')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

# Taylor Series Expansion Example 2 (Cont...)




The approximation approaches the analytic function quickly, even for $x$ not near to $a = 0$.

# Sources of Error

There are usually two sources of error

- **Round-off Errors** that are due to the inexactness in the representation of real numbers on a computer and the arithmetic operations done with them
- **Truncation Errors** that are due to the approximate nature of the method used

For example, when the function $f(x)$ is approximated with the Taylor series approximation, then

$$f(x) = f_n(x) + E_n(x)$$

where $E_n(x)$ represents the truncation error. With more terms we use, the approximation will be more close to the exact value, and the absolute value of $E_n(x)$ will decrease up to a certain limit, which is machine epsilon.

# Example

Approximate $e^2$ using different order of Taylor series, and print out the results.

```
import numpy as np
```

```
exp = 0
x = 2
for i in range(10):
    exp = exp + \
        ((x**i)/np.math.factorial(i))
    print(f'Using {i}-term, {exp}')

print(f'The true e^2 is: \n{np.exp(2)}')
```

# Example (Cont...)

```
Using 0-term, 1.0
Using 1-term, 3.0
Using 2-term, 5.0
Using 3-term, 6.333333333333333
Using 4-term, 7.0
Using 5-term, 7.266666666666667
Using 6-term, 7.355555555555555
Using 7-term, 7.3809523809523805
Using 8-term, 7.387301587301587
Using 9-term, 7.3887125220458545
The true e^2 is:
7.38905609893065
```

We can see the higher order we use to approximate the function at the value, the closer we are to the true value.

# Estimate Truncation Error

From the Taylor series, if we use only the first $n$ terms, we can see:

$$f(x) = f_n(x) + E_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)(x-a)^k}{k!} + E_n(x)$$

The $E_n(x)$ is the remainder of the Taylor series, or the truncation error that measures how far off the approximation $f_n(x)$ is from $f(x)$.

We can estimate the error using the **Taylor Remainder Estimation Theorem**:

If the function $f(x)$ has $n+1$ derivatives for all $x$ in an interval $I$ containing $a$, then, for each x in $I$, there exists $z$ between $x$ and $a$ such that

$$E_n(x) = \frac{f^{(n+1)}(z)(x-a)^{(n+1)}}{(n+1)!}$$

# Estimate Truncation Error (Cont...)

In many times, if we know $M$ is the maximum value of $|f^{(n+1)}|$ in the interval, we will have:

$$|E_n(x)| \leq \frac{M|x-a|^{(n+1)}}{(n+1)!}$$

Therefore, we can get a bound for the truncation error using this theorem

# Example: Error Limit Estimation

Estimate the remainder bound for the approximation using taylor series for $e^2$ using n = 9.

We know that $(e^x)' = e^x$, and, a = 0. Therefore, the error related to x = 2 is:

$$E_n(x) = \frac{f^{(9+1)}(z)(x)^{(9+1)}}{(9+1)!} = \frac{e^z 2^{10}}{10!}$$

Recall that $0 \leq z \leq 2$, and $e < 3$, we will have

$$|E_n(x)| \leq \frac{3^2 2^{10}}{10!} = 0.00254$$

Therefore, if we use Taylor series with n = 9 to approximate $e^2$, our absolute error should be less than 0.00254. Let's also verifty it below.

```
]: abs(7.3887125220458545-np.exp(2))

]: 0.0003435768847959153
```

# Round-off errors for Taylor series

Numerically, to add many terms in a sum, we should be mindful of numerical accumulation of errors that is due to floating point round-off errors.

**EXAMPLE:** Approximate $e^{-30}$ using different order of Taylor series, and print out the results.

```
exp = 0
x = -30
for i in range(200):
    exp = exp + \
        ((x**i)/np.math.factorial(i))

print(f'Using {i}-term, our result is {exp}')
print(f'The true e^2 is: {np.exp(x)}')
```

```
Using 199-term, our result is -8.553016433669241e-05
The true e^2 is: 9.357622968840175e-14
```

# What went wrong?

- When using negative large arguments, in order to get a small result, the Taylor series need alternating large numbers to cancel to achieve that.
- We need many digits of precision in the series to capture both the large and the small numbers with enough remaining digits to get the result in the desired output precision.