What is Machine Learning?

- AI versus ML versus DL
- ML as an inverse problem
- ML Inference
Part 1: Introduction to Machine Learning

- **Introduction to Machine Learning**
  - What is machine learning?
    - AI versus machine learning versus Deep Learning
    - Machine learning and inference
    - Single layer neural networks
  - Loss functions
    - Supervised training
    - Loss minimization
    - Minimum mean squared error (MMSE) loss function
    - Inference versus training
  - Gradient descent optimization
    - Definition of gradient descent
    - Computing the function and loss gradient
    - Convex sets and functions
    - Local minimum, saddle points, and global minimum
    - Optimization theorems
  - Training and Generalization
    - Overfitting, underfitting, and generalization error
    - Training data, validation data, and testing data
    - Interpreting training and validation error
  - Probability and Estimation
    - Probability, random variables, marginal and conditional expectation
    - Frequentist and Bayesian estimation, ML and MAP estimators
    - The bias/variance tradeoff
AI versus ML versus DL

- **Artificial Intelligence (AI):**
  - Create computers behave intelligently – like humans
  - Alan Turning 1950: Turning test
  - Broadly defined:
    - Artificial general intelligence and Strong AI: Data on Star Trek
    - Weak AI and the Chinese room (Searle 1980): Exhibits intelligent behavior but not conscious
    - Narrow AI: Task specific AI such as Siri

- **Machine Learning (ML):**
  - Train an algorithm to reproduce answers from data

- **Deep Learning (DL):**
  - A particularly successful ML method based on deep sequences of neural networks
Inverse Problems

- Determine some unknown quantity from available data.

- $x$ what we would like to know
- $y$ what we can measure or observe

This is an inverse problem:
- Computer vision, sensing, demodulation, speech recognition, etc.
- Business analytics: What movie will the customer like best?
- Science: What is the structure of this particle?
Goal of Machine Learning (ML): Solve this inverse problem

Observations:
- The answer, $\hat{x}$, is usually not equal to the unknown, $x$.
- But hopefully, $\hat{x}$ is close to $x$.

Questions:
- How do we compute the inverse?
- Is there a best inverse?

Mick Jagger’s Theorem:
- You can’t always get what you want. But if you try sometimes, you might fine, you get what you need.
- **Machine Learning approach**

  ![Diagram]

  - Inference Function
  - $y$ \rightarrow $\hat{x}$
  - $y$ \text{ data} \rightarrow \text{Inference Function} \rightarrow \hat{x} \text{ estimate}
  - $\theta$ \text{ parameters}

- **Comments**
  - Adjust parameter $\theta$ to achieve the “best” or at least a “good” answer
  - $\hat{x}$ has a “hat” because it is an estimate (i.e., a guess) of the true unknown $x$.
  - $y$, $\hat{x}$, and $\theta$ are usually finite dimensional vectors.
ML: Mathematical Inference

- Mathematical representation of ML inference

\[ \hat{x} = f_\theta(y) \]

- Questions:
  - What family of functions do we choose for \( f_\theta(\cdot) \)?
  - What is the dimension of \( \theta \in \mathbb{R}^P \)?
  - What is our goal?
  - How do we determine the best value of \( \theta \)?
Single Layer Neural Networks

- Mathematical representation
- Graphical representation
- One hot encoding
- Activation Functions
What ML Function should we use?

Possible choices
- Support vector machines (SVD)
- Radial basis functions (RBF)
- Gaussian mixture functions

Neural Networks
- Very high capacity/model order
- Easy to train with modern analytical/computational tools.
- Shallow neural networks: Use one easy-to-train layer.
- Deep neural networks: Train a hierarchical stack of layers
Single Layer Dense NN

- Single layer NN, graphically

\[
y \rightarrow Ay \rightarrow \sigma(\cdot) \rightarrow B \rightarrow \hat{x}
\]

- Mathematically, \( \hat{x} = B \sigma(Ay + b) \) where
  - \( A \in \mathbb{R}^{N_1 \times N_y} \) is a matrix of multiplicative weights.
  - \( b \in \mathbb{R}^{N_1} \) is a column vector of additive offsets.
  - \( \sigma: \mathbb{R}^{N_1} \rightarrow \mathbb{R}^{N_1} \) is a point-wise activation function.
  - \( B \in \mathbb{R}^{N_x \times N_1} \) is a matrix of multiplicative weights.
  - Typical activation function: Logistic sigmoid

\[
\sigma_i(z) = \frac{1}{1 + e^{-z_i}}
\]
Point-Wise Activation Functions

- Logistic sigmoid function
  \[ \sigma_i(z) = \frac{1}{1 + e^{-z_i}} \]

- Rectified linear unit (ReLU)
  \[ \sigma_i(z) = \begin{cases} 
  0 & \text{if } z_i \leq 0 \\ 
  z_i & \text{if } z_i > 0 
\end{cases} \]

- Leaky ReLU
  \[ \sigma_i(z) = \begin{cases} 
  \alpha z_i & \text{if } z_i \leq 0 \\ 
  z_i & \text{if } z_i > 0 
\end{cases} \]
Point-Wise Activation Function

- Point-wise activation function, \( \sigma: \mathbb{R}^N \rightarrow \mathbb{R}^N \)

\[
\begin{align*}
z &\quad \mapsto \quad \sigma(\cdot) \\
&\quad \mapsto \quad y
\end{align*}
\]

\[
\begin{align*}
z_0 &\quad \mapsto \quad \sigma_0(\cdot) \\
&\quad \mapsto \quad y_0 \\
\vdots \\
z_1 &\quad \mapsto \quad \sigma_1(\cdot) \\
&\quad \mapsto \quad y_0
\end{align*}
\]
Gradient of Point-Wise Activation Function

- Gradient Matrix is:
  - Diagonal
  - Sparse (most entries are zero)
  - Fast to compute and apply

$$\nabla \sigma(z) = \begin{bmatrix} \frac{\partial \sigma_i(z)}{\partial z_j} & & & \end{bmatrix} = \begin{bmatrix} d_0 & d_1 & 0 & & \cdots & 0 & d_N \end{bmatrix}$$

$$d_i = \frac{\partial \sigma(z_i)}{\partial z_i}$$
Single Layer NN: Abstract Form

- Single layer NN,

Mathematical representation is

\[ f_{\theta}(y) = B\sigma(Ay + b) \]

where \( \theta = (A, B, b) \) is the set of all NN parameters.
**Single Layer NN Flow Diagram**

- Example for $N_y = 3$, $N_1 = 4$, and $N_x = 2$.

- **Approximation theorem:**
  - Cybenko 1989, "Approximation by Superpositions of Sigmoidal Functions" ⇒
    Any function can be approximated by a single layer neural network!
  - But number of hidden layers might be huge!!
One-Hot Encoding for Classification

- A method to encode the class of an object
  - A vector $y \in \mathbb{R}^M$ needs to be classified into one of $M$ possible classes.

- Standard encoding:
  - $\hat{x} \in \{0, \ldots, M - 1\}$ each value represents a different class

- One-hot encoding:
  - $\hat{x} \in \mathbb{R}^M$ s.t.
    $\hat{x}_i = \begin{cases} 1 & \text{if class} = i \\ 0 & \text{if class} \neq i \end{cases}$

- Example: For $M = 5$, and class=3, then $\hat{x} = [0, 0, 1, 0, 0]$
Continuous Encoding for Classification

- Define an $M$-dimensional simplex as

$$S^M = \left\{ x \in \mathbb{R}^M : \forall i, x_i \geq 0 \text{ and } 1 = \sum_{i=0}^{M-1} x_i \right\}$$

  - Then $\hat{x} \in S^5 \subset \mathbb{R}^5$
  - Like a probability density for each class

- Advantage:
  - Continuous function on a convex set
  - Makes optimization easier
  - Allows for representation of probability densities

- Example: For $M = 5$, and class=3, then $\hat{x} = [p_0, p_1, p_2, p_3, p_4]$
Softmax Activation Functions

• Softmax

\[ \sigma_i(z) = \frac{e^{z_i}}{\sum_j e^{z_j}} \]

– Joint activation function
– Notice that \( \sigma_i(z) \in S^N \subset \mathbb{R}^N \).
– It can be interpreted as a probability density.
Gradient of Softmax Function

- Gradient Matrix is:
  - Dense matrix (most or all entries are non-zero)
  - Usually slow to compute, but some tricks in this case

\[
[\nabla \sigma(z)]_{i,j} = \frac{1}{\sum_k e^{z_k}} \left( e^{z_i} \delta_{i,j} - \frac{e^{z_i} e^{z_j}}{\sum_k e^{z_k}} \right)
\]

\[
\nabla \sigma(z) = \frac{1}{\sum_k e^{z_k}} 
\begin{bmatrix}
    e^{z_0} & 0 & \cdots & 0 \\
    0 & e^{z_1} & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & e^{z_N}
\end{bmatrix}
- \frac{1}{(\sum_k e^{z_k})^2} 
\begin{bmatrix}
    e^{z_0} & e^{z_1} & \cdots & e^{z_N} \\
    e^{z_1} & \cdots & \cdots & \cdots \\
    \vdots & \vdots & \ddots & \vdots \\
    e^{z_N} & \cdots & \cdots & e^{z_N}
\end{bmatrix}
\]
The Loss Function

- Measuring supervised training error
- Mathematical representation
- Parameter estimation through loss minimization
- Inference and training as inverse problems
### Training:

- Measure $x$ too! This produces training data.
  - Collecting training data is application specific
  - It can be difficult, expensive, or even impossible
- Select $\theta$ so that $\epsilon = x - \hat{x}$ is small
- How do we measure “small”?
Find lots of training data
- (\(x_k, y_k\)) for \(k = 0, \ldots, K - 1\)

Define a loss function \(L(\theta)\)

Pick \(\theta\) to minimize \(L(\theta)\)
The ML Loss Function

- What is a loss function?
  - The loss function is a measure of training error
  - So for example, \( x_k \in \mathbb{R}^{N_x} \)

\[
Loss = L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - \hat{x}_k \|^2 = \frac{1}{K} \sum_{k=0}^{K-1} \sum_{i=0}^{N_y-1} (x_{k,i} - \hat{x}_{k,i})^2
\]

But wait! Gobbligood Alert!

- What I really mean is

\[
Loss = L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - f_\theta(y_k) \|^2
\]
Loss Function Properties

Facts:

- Usually called mean squared error (MSE). But technically, MSE is really defined as
  \[ L_{MSE}(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - f_\theta(\gamma_k) \|^2 \]

- When \( L(\theta) = 0 \), then for all \( k \), \( x_k = f_\theta(\gamma_k) \)
Parameter Estimation using Loss Minimization

\[ \theta^* = \arg\min_\theta \{ L_{MSE}(\theta) \} = \arg\min_\theta \left\{ \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - f_\theta(y_k) \|^2 \right\} \]

- Estimate \( \theta^* \) by minimizing loss

**What does this mean?**

\[ \theta^* = \arg\min_\theta \{ l_{MSE}(\theta) \} \]

---

**Question: What is the lowest point in the USA?**

Death Valley = \( \arg\min_{\theta \in \text{USA}} [\text{Altitude}(\theta)] \)

Badwater Basin = \( \arg\min_{\theta \in \text{USA}} [\text{Altitude}(\theta)] \)

- \( -282 \) feet = \( \min_{\theta \in \text{USA}} [\text{Altitude}(\theta)] \)

- “\( \min \)” returns the minimum value
- “\( \arg \min \)” returns the parameter that minimizes the value
ML Supervised Training

- Generate training data:
  - Collect $K$ training pairs, $(x_0, y_0), (x_1, y_1), \ldots, (x_{K-1}, y_{K-1})$

- Estimate parameter:
  - $\theta^* = \min_{\theta} \{ l_{\text{MSE}}(\theta) \}$

- This is also an inverse problem!
  - Trying to find the “hidden” or “latent” value of $\theta$. 
Complete ML System

Supervised training:
- Off-line and often slow
- Requires expensive training data

Online inference:
- On-line and usually needs to be fast

\[
y \rightarrow \hat{x} = f_{\theta^*}(y) \rightarrow \hat{x}
\]
Two Inverse Problems in ML

- **Inference inverse problem:**
  - Estimate the unknown, \(\hat{x}\), from the available measurements, \(y\).

- **Training inverse problem:**
  - Estimate the unknown parameter, \(\theta^*\), from the training pairs, \((x_n, y_n)|_{n=0}^{N-1}\).

```
```

```