Day 6: Starting Vision

Week Overview

- Introduction to ML
- Linear Regression
- Gradient Descent
- Modeling in ML
- Single Layer Neural Networks (NN) + Universal Function Approximation
- Multi-Layer and Convolutional NN

Capstone: Face Recognition

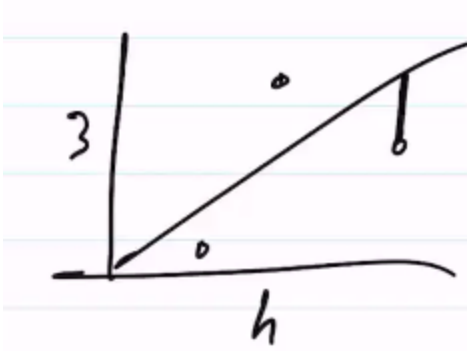
- matching faces to a database
    - clustering of faces

Introduction to ML

- The concept is to take some set or subset of observed data and use an intelligent system (that can learn or be trained) and output a prediction and/or decision.
- The manual approach: Provide data, context, and the desired outcome to a human component, who generates the rules for transforming an input into the output, which another human will make a program out of. New data can now be input into the program to generate its own predictions and/or decision.
    - Examples: audio capstone project, chess (Deep Blue), old chatbots, game NPCs
- ML approach: Provide data, context, and the desired outcome to a human component, who interprets the problem that is posed and attach it to an appropriate framework which outputs a set of rules. The rules can be passes to another human component to write a new computer program. From here, new data can be passed to the computer program and create new prediction and/or decisions.

Linear Regression

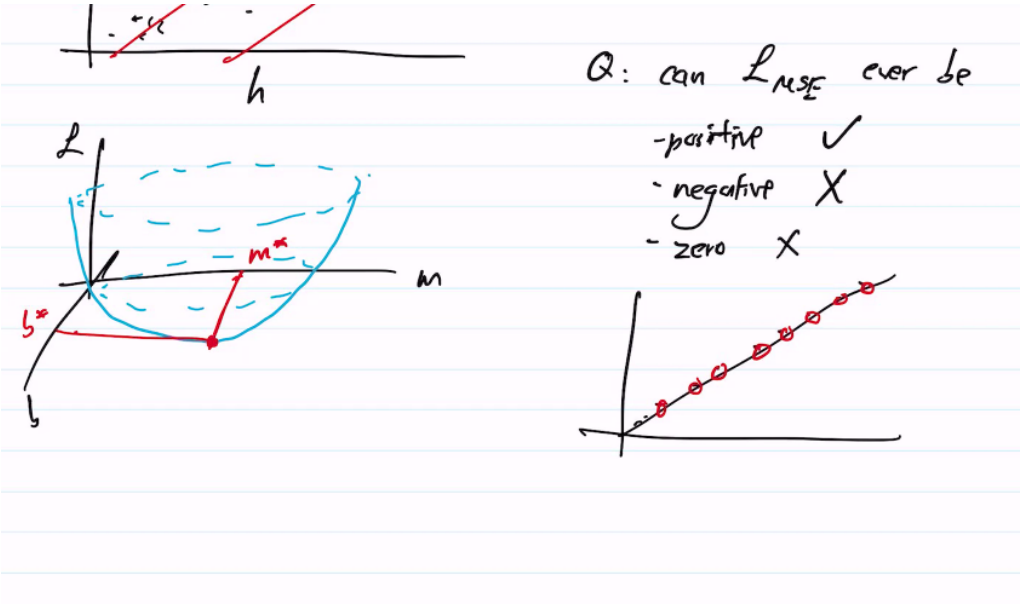
- A simple model which takes observed data, trains a model on the data, and outputs a prediction from that (eg.: learning wingspan from human height)
- Regression: estimate a continuous function that describes a noisy dataset
- Linear regression: Regression with a linear function
  - $y = mx + b$
  - $F(m, b; x) = mx + b \rightarrow$ predicted wingspan
  - How do we find good values for m and b?
    - What is good here?
      - “Good” = close to actual
      - minimizes error
    - Objective functions - some function that we use to measure the quality of a prediction



    - $res = |y^{(true)} - y^{(pred)}|$ is the objective function
    - a better objective function is $\sum_{n=0}^{N-1}(y_n^{(true)} - y_n^{(pred)})$
      - This is formatted as $L_{MSE}(m, b; (x_n, y_n^{(true)})_{n=0}^{N-1})$ which is $\frac{1}{N}(SSE)$ where SSE is “sum of squared errors”
    - The final notation is:

$$argmin_{m,b}(L_{MSE}(m, b; (x_n, y_n^{(true)})_{n=0}^{N-1})) = m^*, b^*$$

- $m*, b*$ = the parameters of the model F that minimize the loss function given the data it has access to (training data)
- SSE vs MSE (sum of squared errors vs mean squared error)
  - SSE is MSE multiplied by the number of data points present. As such, the only difference (visually) between the two is that SSE would grow much faster than MSE. There is not much other difference in how data is modeled **BUT** there is a difference in how loss is minimized (during training)
- The “least risky” model should be used (Empirical Risk Minimization)
  - fewest mistakes in existing data



- Assumptions behind L_{MSE}:
  - no outliers → RANSACK algorithm removes outliers
  - old/new data are collected under the same conditions/are from the same distribution
    - NBA player -/→ (golf players) but may apply to (WNBA players) or (42 year olds) etc.

- How do we actually find m^* and b^*?

$$L = \frac{1}{N}\sum_{n=0}^{N-1}(y_n^{(true)} - (mx_n + b))^2$$

$$\frac{dL}{dm}|_{m=m^*} = 0 = \sum(2m^*x_n^2 + 2b^*x_n - 2x_ny_n)$$

$$\frac{dL}{db}|_{b=b^*} = 0 = \sum(2b^* + 2x^*x_n - 2y_n)$$

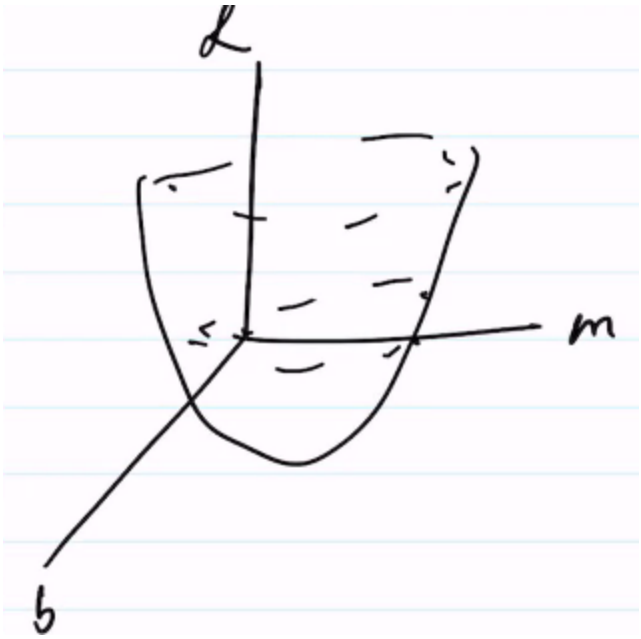
these differentiations yield:

$$m^* = \frac{\sum_{n=0}^{N-1}(n_xy_n) - \frac{1}{N}(\sum y_n)(\sum x_n)}{\sum(x_n^2) - \frac{1}{N}(\sum x_n)}$$

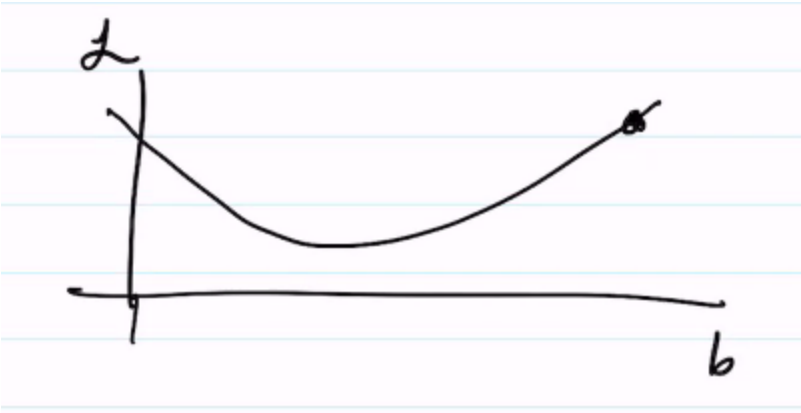
$$b^* = \frac{1}{N}\sum(y_n) - m^*\frac{1}{N}\sum(x_n)$$

Gradient Descent

- For L_{MSE} on a line, we had a nice closed for solution (can be represented by an algebraic expression we can solve), but this wouldn't apply for other problems (e.g: next chess move? am I looking at a cat?)
- Example loss landscape:



- Is it possible to find the minimum of the loss landscape using some numerical expression
  - Okay to find approximate solution for b^* and m^*
  - Assume we can measure the slope of the loss landscape at any point



- Start at some initial guess b_0
  1. Measure the slope at current location
  2. If slope ~ 0, stop and set $b^* = b_0$. Otherwise, take a “step” downhille
  3. Repeat steps 1 and 2

- Or more mathematically: $b_n = b_o - \delta \frac{dL}{db}|_{b_o}$ where δ is the learning rate
  - If δ is very small, the model:
    - Learns slowly
    - Is prone to get stuck in a local minimum
    - May be more accurate
  - If δ is too big, the model:
    - May get out of local minima
    - May be unstable
- Example: $L(b) = b^2; \frac{dL}{db} = 2b; b_0 = 10, \delta = 0.3$ over 4 steps
  - Iteration #1: $b_n := 10 - (0.3)(2 * 10) = 4, b_o = 10$
  - Iteration #2: $b_n := 4 - (0.3)(2 * 4) = 1.6, b_o = 4$
  - Iteration #3: $b_n := 1.6 - (0.3)(2 * 1.6) = 0.64, b_o = 1.6$
  - Iteration #4: $b_n := 0.64 - (0.3)(2 * 0.64) = 0.256, b_o = 0.64$
- Generalizing to L_{MSE}
  - $m_n = m_o - \delta \frac{dL}{dm}|_{m_o, b_o}$
  - $b_n = b_o - \delta \frac{dL}{db}|_{m_o, b_o}$

  OR

  - $\begin{pmatrix} m_n \\ b_n \end{pmatrix} = \begin{pmatrix} m_n \\ b_n \end{pmatrix} - \delta \begin{pmatrix} \frac{dL}{dm} \\ \frac{dL}{db} \end{pmatrix}$
- Generalize to other models and objective functions
  - “model parameters” or “model weights” are represented by w

    $$F(m, b; x) -> F(w, w_2, ..., w_m; x) -> F((w_i)_{i=0}^{m-1}; x)$$

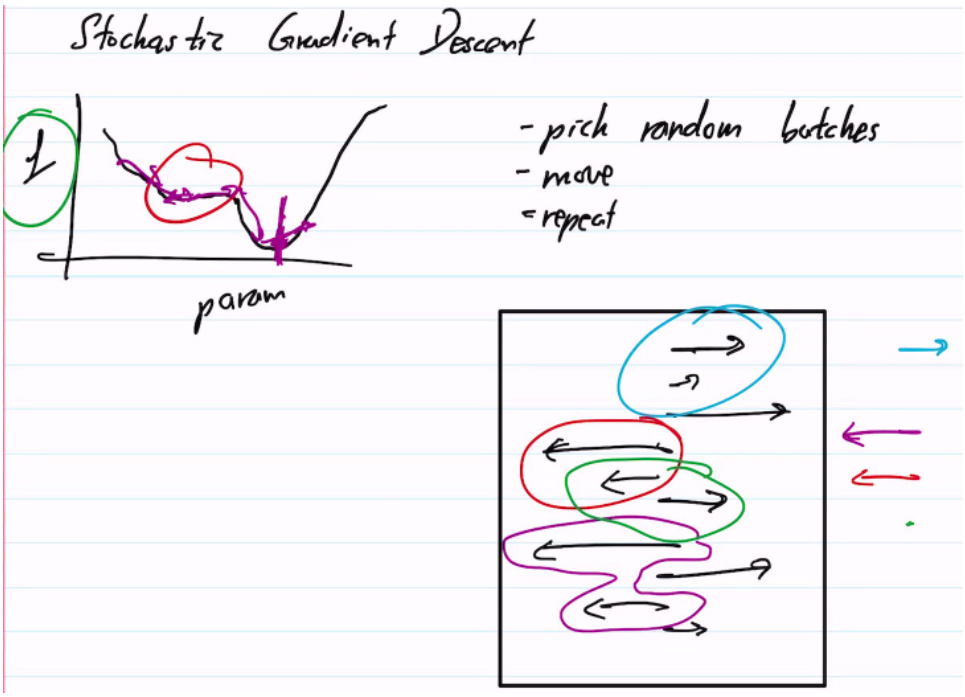
  - Ad loss we be represented as so, where there is some number for the loss of data point (x_i, y_i)

    $$L(w_o, w_1, ..., w_{m-1}; (x_i, y_i)_{i=0}^{N-1})$$

    - Batching: N_0 is the size of a “batch” of data
        - $L_b = \frac{1}{N_o} \sum_{i=0}^{N-1} L(y_i^{(pred)}, y_i^{(true)})$
        - As $N_0 \to N$, $L_b \to L$ (as the batch accumulates to reach the size of all data points, the loss for each batch gets closer and closer to the total loss)
        - Batching isn’t entirely necessary theoretically, but it is useful in the real world where computing power may not be high enough to handle all the data at one point
- Hyperparameters
    - δ (learning rate)
    - # of step sizes
    - Method for determining m_0, b_0

Stochastic Gradient Descent

- In contrast to regular Gradient Descent, Stochastic Gradient Descent does not use all the training data points. Instead it selects random batches of the data (which may have wildly different parameters) and uses these different parameters to move around on the loss plane.
- This has the benefit of making the steps slightly more noisy, therefore decreasing the chance of converging and remaining in a local min instead of a global min.



Autodifferentiation

Linear Regression

- Create a linear model
- Use `mygrad` to compute derivatives for gradient descent in order to optimize model parameters
- Fit the model on normalized and unnormalized data