CS 556 Mathematical Foundations of Machine Learning

Linear Regression

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

In Praise of Linear Models!

- Despite its simplicity, the linear model has distinct advantages in terms of its interpretability and often shows good predictive performance.
- In this lecture, we discuss (1) optimization, (2) some ways in which the simple linear model can be improved, by replacing ordinary least squares fitting with some alternative fitting procedures.

Regression Task

- Example: given the gender and height information, predict the weight.

Data sample	Gender	Height (inches)	Y: Weight (pounds)
1	Male	73	190
2	Male	67	165
3	Female	65	130

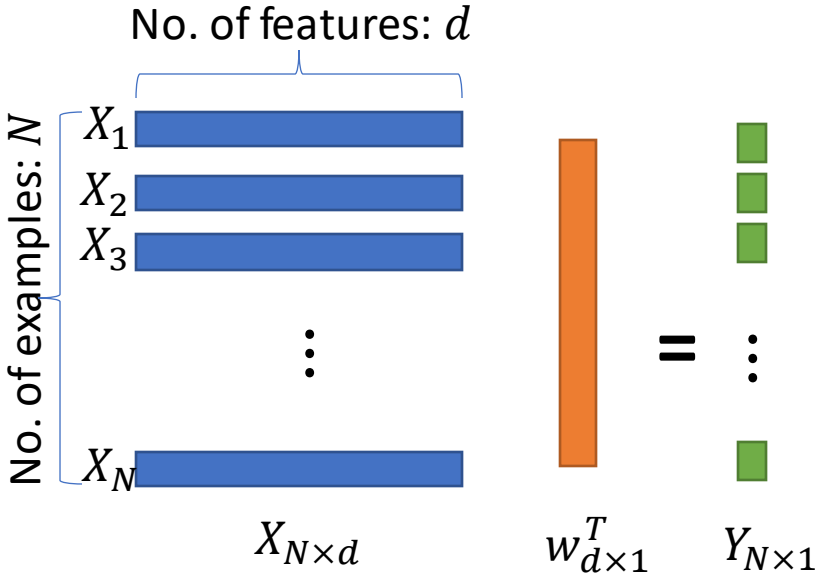
Features/Attributes (X) **Output (Y)**

$$y \approx f(x) = \sum_{i=1}^{d} w_i x_i = x w^T$$

Linear Regression

- Input: d-dimensional $x = (x_1, x_2, \ldots, x_d)$.
- Output: Real value output y.
- Regression Model:

$$y \approx f(x) = \sum_{i=1}^{d} w_i x_i = x w^T$$



where $w = (w_1, w_2, \ldots, w_d)$ are parameters (weights/coefficients).

- Training Data: $(X_1, Y_1), (X_2, Y_2), \cdots, (X_N, Y_N)$ used for the estimation of the parameters w.

The Learning Problem

- Hypothesis class: Consider some restricted set F of mappings $f: X \rightarrow Y$ from input data to output.
- Estimation: Find the best estimate of mapping $\hat{f} \in F$ based on the training data.
- Evaluation: Measure how well $\hat{f}$ generalizes to the testing data (unseen examples), i.e., whether $\hat{f}(x_{new})$ agrees with y_{new}.

Estimation Criterion

- Loss function $Loss(x, y, w)$: a function of w that quantifies how well (more accurately, how badly) you would be if you use w to make a prediction on x when the correct output is y.
- Objective: the estimation problem can be solved by minimizing the loss function.
- Valid for any parameterized class of mapping from examples to predictions.
- Valid when the predictions are discrete labels or real valued as long as the loss is defined properly.

Objective of Linear Regression

- Prediction: for one data example (x, y), the prediction is:

$$f_w(x) = xw^T$$

- Residual: difference between the prediction and the target.
- Squared loss: $Loss(x, y, w) = (y - f_w(x))^2$; smaller values of loss indicates that $f_w(x)$ is a good approximation of y.
- Example:
  - $w = [2, -1], x = [2,0], y = -1, L(x, y, w) = 25$

Objective of Linear Regression

- Loss is easy to minimize if there is only one example.
- Overall training loss: on the training data D_{train} with N examples:

$$\min_{w \in R^d} TrainLoss(w) = \min_{w \in R^d} \frac{1}{N} \sum_{(x,y) \in D_{train}} L(x, y, w)$$

- Key: learn the parameters to make the global tradeoffs.
- If $Loss(x, y, w) = (y - f_w(x))^2$, the residual sum of squares (RSS) is defined as $RSS = \sum_{(x,y) \in D_{train}} L(x, y, w)$.

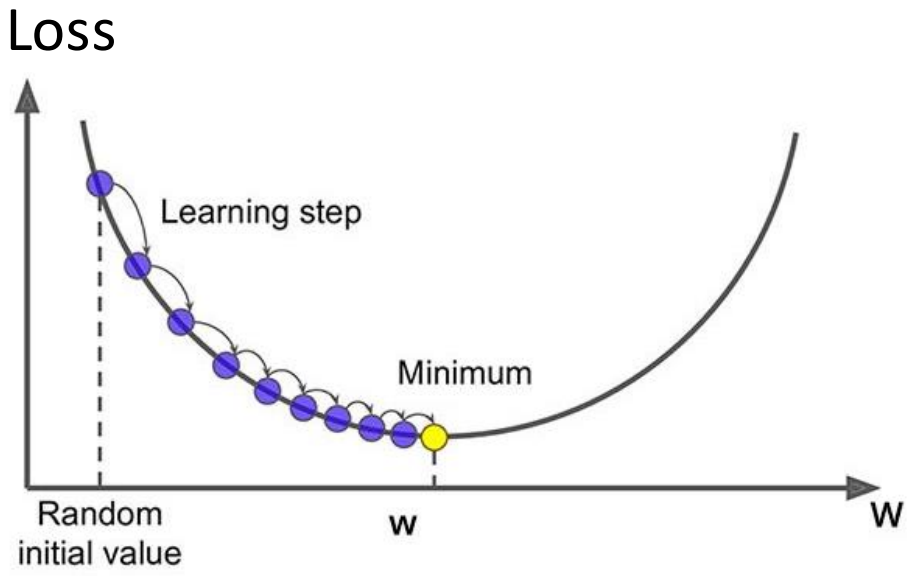
Regression Loss Functions

- Squared loss: $L_2(x, y, w) = (y - f_w(x))^2$
  - The overall loss will be mean squared error (MSE)
  - Root mean squared error (RMSE): $RMSE = \sqrt{MSE}$

- Absolute loss: $L_1(x, y, w) = |y - f_w(x)|$
  - The overall loss will be mean absolute error (MAE)

How to Optimize: Gradient Descent

- A first-order iterative optimization algorithm for finding the local minimum of a differentiable function.
- To get the parameter that minimizes the objective function, we iteratively move in the opposite direction of the gradient of the function.
- The size of each step is determined by a parameter which is called Learning Rate.



How to Optimize: Gradient Descent

- Objective: $TrainLoss(w) = \min_{w \in R^d} \frac{1}{N} \sum_{(x,y) \in D_{train}} (y - xw^T)^2$
- Gradient with respect to w: the direction that increases the loss the most.

$$\nabla_w TrainLoss(w) = \frac{1}{N} \sum_{(x,y) \in D_{train}} 2(y - xw^T)\, x$$

- Gradient descent update:

$$w \leftarrow w - \eta * \nabla_w TrainLoss(w)$$

Step size Gradient

Gradient Descent

```
1: procedure BATCH GRADIENT DESCENT
2:     for i in range(epochs) do
3:         g^(i)(w) = evaluate_gradient(TrainLoss, data, w)
4:         w = w − learning_rate ∗ g^(i)(w)
```

Some Limitations:

- Each iteration requires going over all training examples (Batch gradient descent) – expensive and slow when have lots of data.
- Intractable for datasets that don't fit in memory.
- Guaranteed to converge to the global minimum for convex functions, but may end up at a local minimum for non-convex functions.

Stochastic Gradient Descent (SGD)

- Considers only one point at a time to update weights.
- Not calculate the total error for whole data in one step, instead we calculate the error of each point and use it to update weights.
- Gradient descent update:

$$w \leftarrow w - \eta * \nabla_w TrainLoss(w)$$

Step size Gradient

- Stochastic Gradient Descent:

For each $(x, y) \in D_{train}$,

$$w \leftarrow w - \eta \nabla_w Loss(x, y, w)$$

Stochastic Gradient Descent (SGD)

```
1: procedure STOCHASTIC GRADIENT DESCENT
2:     for i in range(epochs) do
3:         np.random.shuffle(data)
4:         for example ∈ data do
5:             g^(i)(w) = evaluate_gradient(loss, example, w)
6:             w = w − learning_rate ∗ g^(i)(w)
```

- Shuffling: to ensure that each data point creates an "independent" change on the model, without being biased by the same points before them.
- Easy to fit in memory as only one data point needs to be processed at a time, thus, computationally less expensive.
- With a high variance that cause the objective function to fluctuate heavily.
- May never reach local minima and oscillate around it due to the fluctuations in each step.

Mini-batch Gradient Descent

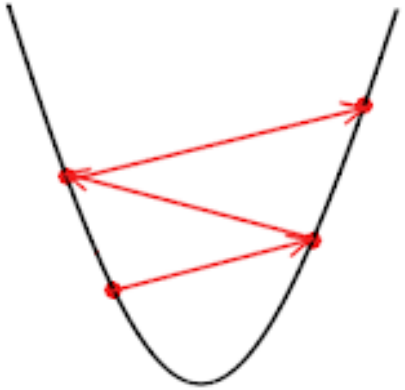
```
1: procedure MINI-BATCH GRADIENT DESCENT
2:     for i in range(epochs) do
3:         np.random.shuffle(data)
4:         for batch ∈ get_batches(data, batch_size=50) do
5:             g^(i)(w) = evaluate_gradient(loss, batch, w)
6:             w = w − learning_rate ∗ g^(i)(w)
```

- Instead of using the whole data for calculating gradient, we use only a mini-batch of it (batch size is a hyperparameter).
- Reduces the variance of the parameter updates, which can lead to more stable convergence.
- Can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

Learning Rate

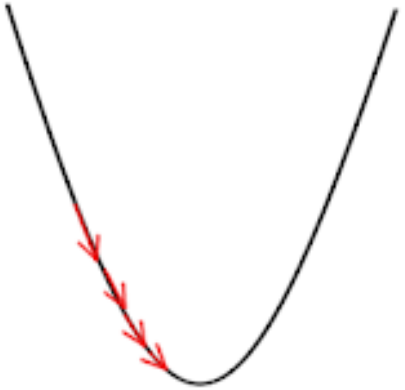
Choosing an appropriate value of learning rate is important because it helps in determining how much we have to descent in each iteration.

High Learning Rate



Causes drastic updates and leading to divergent behavior

Just Right



Swiftly reaches the minimum point

Too small

Requires many updates before reaching the minimum points

https://medium.com/analytics-vidhya/gradient-descent-why-and-how-e369950ae7d3

Summary

- Linear predictors:

$$f(x) = \sum_{i=1}^{d} w_i x_i = x w^T$$

- Loss minimization: learning as optimization

$$\min_{w \in R^d} TrainLoss(w)$$

- Optimization algorithm: gradient decent or other variants

$$w \leftarrow w - \eta * \nabla_w TrainLoss(w)$$

Slides

- Courtesy of Ping Wang.