Gradient Descent

Goal

The goal of gradient descent is to minimize a given function $J(\theta)$ by iteratively updating the parameters θ in the direction of the steepest descent.

Algorithm

1. Initialize the parameters θ (weights) randomly or with zeros.
2. Choose a learning rate η (a small positive value).
3. Repeat until convergence:
   - Compute the gradient of the loss function with respect to the parameters:
   $$\nabla J(\theta) = \frac{\partial J(\theta)}{\partial \theta}$$
   - Update the parameters:
   $$\theta := \theta - \eta \nabla J(\theta)$$

Derivation of the Gradient for Linear Regression

Imagine there is an objective function $J(\theta) = (y - xw)^2$. The gradient descent algorithm will iteratively adjust the weight w to minimize the error between the predicted value xw and the actual value y.

The derivative of J with respoect to w_i can be found using the chain rule:
The chain rule multiplies the derivative of the outer function by the derivative of the inner function:

$$\frac{\partial J}{\partial w_i} = 2(y - xw)(-x_i)$$

Cleaned up: this is

$$\frac{\partial J}{\partial w_i} = -2x_i(y - xw)$$

where xw is the predicted value $\hat{y}$.

So the gradient looks like this:

$$\nabla J(w) = \begin{bmatrix} -2x_1(y - \hat{y}) \\ -2x_2(y - \hat{y}) \\ \vdots \\ -2x_n(y - \hat{y}) \end{bmatrix}$$

which is the same as

$$\nabla J(w) = -2X^T(y - \hat{y})$$

where X is the feature matrix.

Sanity check: The dimensions of X^T are $d \times N$ and the dimensions of $(y - \hat{y})$ are $N \times 1$, so the resulting dimensions of the gradient are $d \times 1$, which matches the dimensions of the weight vector w.

Online vs. Batch Gradient Descent

Online gradient descent

You calculate the gradient for each individual data point and update the weights immediately. This can lead to faster convergence in some cases, especially with large datasets, but it can also introduce more noise into the updates.

Pros:

- Updates the model as data arrives
- Requires less memory since it processes one data point at a time
- Less likely to overfit

Cons:

- Takes longer time to converge because it updates weights more frequently (one observation at a time)

Batch gradient descent

Computes average of the gradients of some set of observations and updates the weights after processing the entire batch. This approach tends to be more stable and can lead to smoother convergence.
Pros:

- More holistic/global
- Faster convergence (generally)

Cons:

- More memory intensive because it requirees storing all data points in memory
- Can overfit

Mini-batch gradient descent (Stochastic Gradient Descent)

A compromise between online and batch gradient descent. It processes small batches of data points at a time, updating the weights after each batch. This approach balances the benefits of both methods, providing a good trade-off between convergence speed and stability. In this case, **batch size is a hyperparameter** that you can tune based on your specific problem and dataset.

Leveraging Linear Algebra

Our full least squares objective function looks like this:

$$J = \frac{1}{N}\sum_{i=1}^{N}(Y_i - x_i w)^2$$

Our batch gradient is:

$$\nabla J(w) = \frac{\partial J}{\partial w} = \frac{1}{N}\sum_{j=1} X_j^T(\hat{Y}_j - Y_j)$$

Design Features

Z-Scoring

Should we **z-score** our data? Z-scoring is the process of standardizing features by removing the mean and scaling to unit variance. This is particularly important for gradient descent because it ensures that all features contribute equally to the distance calculations, preventing features with larger scales from dominating the learning process. Z-scoring can lead to faster convergence and improved model performance.

To z-score a feature, you can use the formula:

$$z = \frac{(x - \mu)}{\sigma}$$

Where μ is the mean of the feature and σ is the standard deviation. The standard deviation is calculated as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Other design features

- Learning rate (smaller results in lower convergent but larger rates can overjump minima). If the value of the objective function is oscilating, this is a sign that the learning rate is too high.
- Batch size (smaller batches have less overfitting but take longer to converge)
- Number of iterations (epochs) - can be decided by max number of iterations or convergence criteria (e.g., change in objective function is less than some small value)
- Termination criteria (when to stop training so as to avoid overfitting)

Dealing with Overfitting

General Approach

- Get more data so we can generalize better
- Get more data for training via cross-validation
- Dont use all the features (feature selection)

Algorithmic Approaches

- Regularization (adding a penalty term to the loss function to discourage complex models)
- Use a validation set to determne the number of epochs and perhaps stop early
- Do stochastic or mini-batch gradient descent instead of full batch gradient descent