L4 Linear Regression Using Gradient Descent

Stochastic Gradient Descent

- How this optimization algorithm works
  - Each training instance is shown to the model one at a time.
  - The model makes a prediction for a training instance
  - the error is calculated and
  - the model is updated in order to reduce the error for the next prediction.
- This procedure can be used to find the set of coefficients in a model that result in the smallest error for the model on the training data.
- In each iteration the coefficients, called weights (w) in machine learning language are updated using the equation:
- w = w - alpha * delta
- w is the coefficient or weight being optimized
- alpha is a learning rate that we must configure (e.g. 0.1)
- gradient is the error for the model on the training data attributed to the weight.

Gradient Descent Iteration #1

x	y
1	1
2	3
4	3
3	2
5	5

- Start with values of 0.0 for both coefficients
- B0 = 0.0
- B1 = 0.0
- y = 0.0 + 0.0 * x
- We can calculate the error for a prediction as follows:
- error = p(i) - y(i)
- p(i) is the prediction for the i'th instance in our dataset and
- y(i) is the i'th output variable for the instance in the dataset.
- We can now calculate the predicted value for y using our starting point coefficients for the first training instance: x = 1; y = 1.
- p(i) = 0.0 + 0.0 * 1=0

Stochastic Gradient Descent

- Stochastic Gradient Descent can be used to learn (search) the coefficients for a simple linear regression model by minimizing the error on a training dataset.
- Gradient Descent is the process of minimizing a function by following the gradients(slope) of the cost function.
- This involves knowing the form of the cost as well as the derivative so that from a given point we know the gradient and can move in that direction
  - e.g. downhill towards the minimum value.
- In machine learning we can use a technique that evaluates and update the coefficients every iteration called stochastic gradient descent to minimize the error of a model on our training data.

Simple Linear Regression with Stochastic Gradient Descent

- The coefficients used in simple linear regression can be found using stochastic gradient descent.
- Stochastic gradient descent is not used to calculate the coefficients for linear regression in practice unless the dataset prevents traditional Ordinary Least Squares being used (e.g. a very large dataset).
- linear regression provides a useful exercise for practicing stochastic gradient descent which is an important algorithm used for minimizing cost functions by machine learning algorithms.
- As stated our linear regression model is defined as follows:
- y = B0 + B1 * x
- Apply to the same training data

Gradient Descent Iteration #1

x	y
1	1
2	3
4	3
3	2
5	5

- Using the predicted output, calculate error:
- error = p(i)-y(i) = (0 - 1) =-1
- We can now use this error in our equation for gradient descent to update the weights.
- We will start with updating the intercept.
- We can say that B0 is accountable for all of the error.
- This is to say that updating the weight will use just the error as the gradient.
- We can calculate the update for the B0 coefficient :w = w - alpha * delta
- B0(t + 1) = B0(t) - alpha * error

Gradient Descent Iteration #1

x	y
1	1
2	3
4	3
3	2
5	5

- B0(t + 1) - the updated version of the coefficient we will use on the next training instance
- B0(t) - the current value for B0
- alpha - our learning rate and
- error - the error we calculate for the training instance.
- Note: Use a small learning rate of 0.01 and plug the values into the equation to work out the new and optimized value of B0
- B0(t + 1) = B0(t) - alpha * error
- B0(t + 1) = 0.0 – 0.01 *-1.0 = 0.01
- Now, update the value for B1.
- Use the same equation with one small change. The error is filtered by the input that caused it.
- We can update B1 using the equation: B1(t + 1) = B1(t) - alpha * error * x

Gradient Descent Iteration #1

x	y
1	1
2	3
4	3
3	2
5	5

- This process must be repeated for the remaining 4 instances from our dataset.
- One pass through the training dataset is called an epoch.
- Calculate 20 iterations or 4 epochs
- 4 complete epochs of the training data being exposed to the model and updating the coefficients.
- list of all of the values for the coefficients over the 20 iterations
- Note: This step is repeated until we reach a stopping condition: either a specified number of steps or the algorithm is within a certain tolerance margin

Gradient Descent Iteration #1 - #20 Python Program

```
x= [1, 2, 4, 3, 5, 1, 2, 4, 3, 5, 1, 2, 4, 3, 5, 1, 2, 4, 3, 5]
y=[1, 3, 3, 2, 5, 1, 3, 3, 2, 5, 1, 3, 3, 2, 5, 1, 3, 3, 2, 5]
B0=0
B1=0
alpha=0.01
for i in range(0, 20):
    pi= B0+B1*x[i]
    error = pi-y[i]
    B0=B0-alpha*error
    B1=B1-alpha*error*x[i]
    print(i+1, " ", "x=", x[i], " ", "y=", y[i], " " , "B0 = ", B0, " ", "B1= ", B1, " ", "error=", error)
```

Gradient Descent Iteration #1

x	y
1	1
2	3
4	3
3	2
5	5

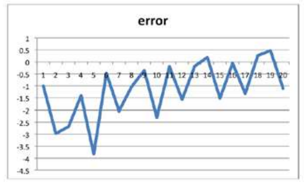
- B1(t + 1) - the update coefficient,
- B1(t) - the current version of the coefficient
- alpha - the same learning rate
- error - the same error calculated above and
- X - the input value.
- Plug in values into the equation and calculate the updated value for B1:
- B1(t + 1) = 0.0 – 0.01 * -1* 1 = 0.01
- We have finished the first iteration of gradient descent and we have updated our weights
- B0 = 0.01 and B1 = 0.01.

Gradient Descent Iteration #1 - #20

```
1  x= 1  y= 1  B0 =  0.01  B1=  0.01  error= -1
2  x= 2  y= 3  B0 =  0.039700000000000006  B1=  0.0694  error= -2.97
3  x= 4  y= 3  B0 =  0.066527  B1=  0.176708  error= -2.6827
4  x= 3  y= 2  B0 =  0.08056049  B1=  0.21880847  error= -1.403349
5  x= 5  y= 5  B0 =  0.1188144616  B1=  0.410078328  error= -3.8253971599999996
6  x= 1  y= 1  B0 =  0.123525533704  B1=  0.414789400104  error= -0.47110721040000003
7  x= 2  y= 3  B0 =  0.14399449036488  B1=  0.45572731342576  error= -2.046895666088
8  x= 4  y= 3  B0 =  0.1543254529242008  B1=  0.4970511636630432  error= -1.03309625593208
9  x= 3  y= 2  B0 =  0.1578706634850675  B1=  0.5076867953456433  error= -0.354521056086696
10  x= 5  y= 5  B0 =  0.18090761708293468  B1=  0.6228715633349792  error= -2.3036953597867162
11  x= 1  y= 1  B0 =  0.1828698252787553  B1=  0.6248337715308  error= -0.19622081958208615
12  x= 2  y= 3  B0 =  0.19854445159535197  B1=  0.6561830241639929  error= -1.5674626316596445
13  x= 4  y= 3  B0 =  0.20031168611283873  B1=  0.6632519622339399  error= -0.17672345174867665
14  x= 3  y= 2  B0 =  0.19841101038469214  B1=  0.6575499350495001  error= 0.19006757281465836
15  x= 5  y= 5  B0 =  0.2135494035283702  B1=  0.7332419007678904  error= -1.5138393143678073
16  x= 1  y= 1  B0 =  0.2140814904854076  B1=  0.7337739877249279  error= -0.05320869570373943
17  x= 2  y= 3  B0 =  0.22726519582605495  B1=  0.7601413984062226  error= -1.3183705340647367
18  x= 4  y= 3  B0 =  0.2245868879315455  B1=  0.7494281668281848  error= 0.2678307894509455
19  x= 3  y= 2  B0 =  0.2198581740473845  B1=  0.7352420251757018  error= 0.47287138841609977
20  x= 5  y= 5  B0 =  0.23089749104812557  B1=  0.7904386101794071  error= -1.1039317000741065
```

Simple Linear Regression Performance Versus Iteration

Plot of the error for each set of coefficients as the learning process unfolded.
A useful graph as it shows that error was decreasing with each iteration and starting to bounce around a bit towards the end.



Simple linear regression predictions for the training dataset

Final value of coefficients

- B0 = 0.230897491 and B1 = 0.79043861.
- Plug them into our simple linear Regression model and make prediction for each point in our training dataset.

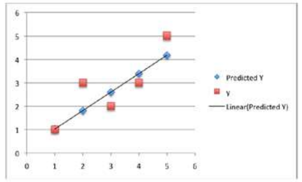
x	y
1	1
2	3
4	3
3	2
5	5

- Making Predictions
- We now have the coefficients for our simple linear regression equation.
- y = B0 + B1 * x
- y = 0.230897491 + 0.79043861 * x
- Problem: Try out the model by making predictions for our training data and plot these predictions as a line with our data. This gives a visual idea of how well the line models our data.

Simple linear regression predictions for the training dataset

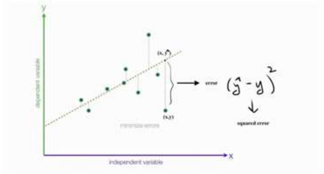
x	Prediction
1	1.021336101
2	1.811774711
4	3.392651932
3	2.602213322
5	4.183090542

Simple Linear Regression Predictions (x vs y in red and x vs prediction in blue)



RMSE

- calculate the RMSE for these predictions
- RMSE = 0.720626401.



Derive the update equation for simple linear regression using stochastic gradient descent (SGD)

- Derive the mean squared error (MSE) loss function for linear regression
- We have a dataset with input features *x* and corresponding target values *y*.
- The linear regression model predicts the target values using the equation: $y=w \cdot x+b$
  - *w* is the weight (slope) associated with the feature *x*.
  - b is the bias term (intercept).
- **Step 1: Define the Mean Squared Error (MSE) Loss Function for linear regression**

$$L(w,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (w \cdot x_i + b))^2$$

  - N is the number of training examples.
  - *xi* is the *i*th feature.
  - yi is the true label for the ith training example.
- **Step 2: Compute the Gradient and Update Coefficients using SGD**
- To update the weight (*w*) using SGD, we need to
  - compute the gradient of the loss function with respect to *w* and
  - update *w* in the opposite direction of the gradient to minimize the loss.
- The update equation for *w* is as follows w = w – alpha * dL/dw

Derive the update equation for simple linear regression using stochastic gradient descent (SGD)

- **Step 3: Compute the Gradient**
- compute the gradient of the mean squared error loss function with respect to *w*:
- Show derivation for the following:

Partial Derivative with Respect to w:

$$\frac{\partial L}{\partial w} = -\frac{2}{N}\sum_{i=1}^{N} x_i \cdot (y_i - (w \cdot x_i + b))$$

Partial Derivative with Respect to b:

$$\frac{\partial L}{\partial b} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - (w \cdot x_i + b))$$

- Setting the derivatives to zero and solving for *w* and *b*, we can find the optimal values that minimize the MSE loss function.

Derive the update equation for simple linear regression using stochastic gradient descent (SGD)

- **Step 4: Gradient Descent Perspective**
- Alternatively, we can view the minimization of the MSE loss function as an optimization problem. We use gradient descent to iteratively update the values of w and b in the opposite direction of the gradient to minimize the loss.
- The gradient of the MSE loss with respect to w and b is given by the partial derivatives above.
- Using gradient descent, we update w and b as follows

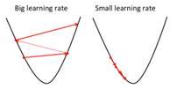
$$w = w - \alpha \cdot \frac{\partial L}{\partial w}$$
$$b = b - \alpha \cdot \frac{\partial L}{\partial b}$$

Understanding Gradient Descent

- The perfect analogy for the gradient descent algorithm that minimizes the cost-function and reaches its local minimum by adjusting the parameters is hiking down to the bottom of a mountain
- Need to make repetitive short steps till they make it to the bottom of the mountain
- Imagine a valley and a person with no sense of direction to get to the bottom of the valley.
- He goes down the slope and takes large steps when the slope is steep and small steps when the slope is less steep.
- He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

Understanding Learning rate - alpha (α)

- The learning rate determines how big the step would be on each iteration
- It is critical to have a good learning rate
  - if it is too large algorithm will not arrive at the minimum (moves from the point on the left all the way to the point on the right)
  - if it is too small algorithm will take forever to get there (gradient descents will work, but very slowly)
- Ex: we set alpha to be 0.01
- learning rate is a number between 0 and 1
  - 0 means we do not change our values at all
  - 1 means we subtract the entirety of our gradient
- B0(t + 1) = B0(t) - alpha * error
- B1(t + 1) = B1(t) - alpha * error * x



- x is the input value for the coefficient.
- B0 does not have an input.
- This coefficient is called the bias or the intercept and we can assume B0 always has an input value of 1.0.

Understanding Stochastic Gradient Descent

- The idea behind stochastic gradient descent is iterating a weight update based on the gradient of loss function
- w(t+1)=w(t)-alpha*error
- The process should end when the weights stop modifying or their variation keeps itself under a selected threshold

Understanding Gradient Descent

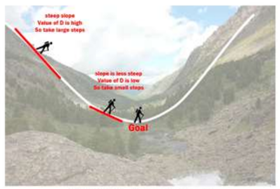


Illustration of how the gradient descent algorithm works

Stochastic Gradient Descent - Summary

- When there are one or more inputs we can use a process of optimizing the values of the coefficients by iteratively minimizing the error of the model on our training data.
- This operation is called Gradient Descent and works by starting with zero values for each coefficient.
- The sum of the squared errors are calculated for each pair of input and output values.
- A learning rate is used as a scale factor and the coefficients are updated in the direction towards minimizing the error.
- The process is repeated until a minimum sum squared error is achieved or no further improvement is possible.
- In practice, SGD is useful when we have a very large dataset either in the number of rows or the number of columns that may not fit into memory.

Stochastic Gradient Descent - Summary

- Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).
- Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

Gradient Descent Procedure

- The procedure starts off with initial values for the coefficient or coefficients for the function.
- These could be 0.0 or a small random value.
- coefficient = 0.0
- The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.
- cost = f(coefficient)
- cost = evaluate(f(coefficient))

Gradient Descent

- The goal of all supervised machine learning algorithms is to best estimate a target function (f) that maps input data (X) onto output variables (Y).
- This describes all classification and regression problems.
- Some machine learning algorithms have coefficients that characterize the algorithms estimate for the target function (f).
- Different algorithms have different representations and different coefficients
- but many of them require a process of optimization to find the set of coefficients that result in the best estimate of the target function.
- Common examples of algorithms with coefficients that can be optimized using gradient descent are Linear Regression and Logistic Regression.

Intuition for Gradient Descent

- Think of a large bowl.
- This bowl is a plot of the cost function (f)
- A random position on the surface of the bowl is the cost of the current values of the coefficients (cost).
- The bottom of the bowl is the cost of the best set of coefficients, the minimum of the function.
- The goal is to continue to try different values for the coefficients, evaluate their cost and select new coefficients that have a slightly better (lower) cost.
- Repeating this process enough times will lead to the bottom of the bowl and you will know the values of the coefficients that result in the minimum cost.

Gradient Descent Procedure

- The derivative of the cost is calculated.
- The derivative is a concept from calculus and refers to the slope of the function at a given point.
- We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.
- delta = derivative(cost)
- Now that we know from the derivative which direction is downhill, we can now update the coefficient values.
- A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.
- coefficient = coefficient - (alpha * delta)
- This process is repeated until the cost of the coefficients (cost) is 0.0 or no further improvements in cost can be achieved.

Gradient Descent

- The evaluation of how close a fit a machine learning model estimates the target function can be calculated a number of different ways
- The cost function involves evaluating the coefficients in the machine learning model by calculating a prediction for each training instance in the dataset and
- comparing the predictions to the actual output values and
- calculating a sum or average error (such as the Sum of Squared Residuals or SSR in the case of linear regression)

Gradient Descent

- From the cost function a derivative can be calculated for each coefficient
- so that it can be updated using exactly the update equation
- The cost is calculated for a machine learning algorithm over the entire training dataset

Applying Gradient Descent – Preparing data for Gradient Descent

- Plot Cost versus Time: Collect and plot the cost values calculated by the algorithm each iteration.
- The expectation for a well performing gradient descent run is a decrease in cost each iteration.
- If it does not decrease, try reducing our learning rate.
- Learning Rate: The learning rate value is a small real value such as 0.1, 0.001 or 0.0001.
- Try different values and see which works best.
- Rescale Inputs: The algorithm will reach the minimum cost faster if the shape of the cost function is not skewed and distorted.
- We can achieve this by rescaling all of the input variables (X) to the same range, such as between 0 and 1.

Gradient Descent

- Gradient descent can be slow to run on very large datasets.
- Because one iteration of the gradient descent algorithm requires a prediction for each instance in the training dataset, it can take a long time when we have many millions of instances.
- When we have large amounts of data, we can use a variation of gradient descent called stochastic gradient descent.
- In this variation, the gradient descent procedure is run
- but the update to the coefficients is performed for each training instance
- rather than at the end of the batch of instances (unlike batch gradient descent)

Applying Gradient Descent – Preparing data for Gradient Descent

- Few Passes: Stochastic gradient descent often does not need more than 1-to-10 passes through the training dataset to converge on good coefficients.
- Plot Mean Cost: The updates for each training dataset instance can result in a noisy plot of cost over time when using stochastic gradient descent.
- Taking the average over 10, 100, or 1000 updates can give a better idea of the learning trend for the algorithm.