Gradient Descent and Logistic Regression

Gradient Descent

Recall from calculus that the gradient of a function is a vector of partial derivatives.

If *y* is a scalar and *x* = *[x_0,...,x_{n-1}]* is a vector, the gradient of *y* with respect to *x* is:

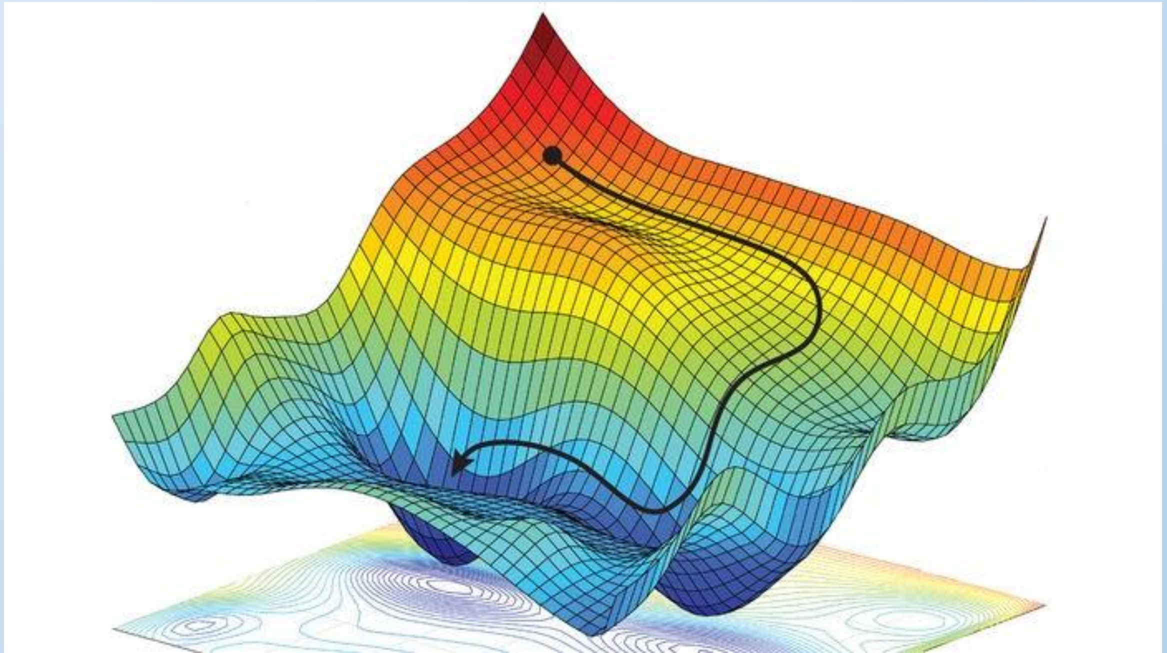
$$\nabla(y) = \langle \frac{\partial y}{\partial x_0}, \frac{\partial y}{\partial x_1}, \cdots, \frac{\partial y}{\partial x_{n-1}} \rangle$$

Intuitively, ∇(y) is the direction of maximum increase in y with respect to x. That is, the vector of infinitesimal changes in x that will results in the largest increase in y.

Following the

negative gradient

of a function



Gradient Descent

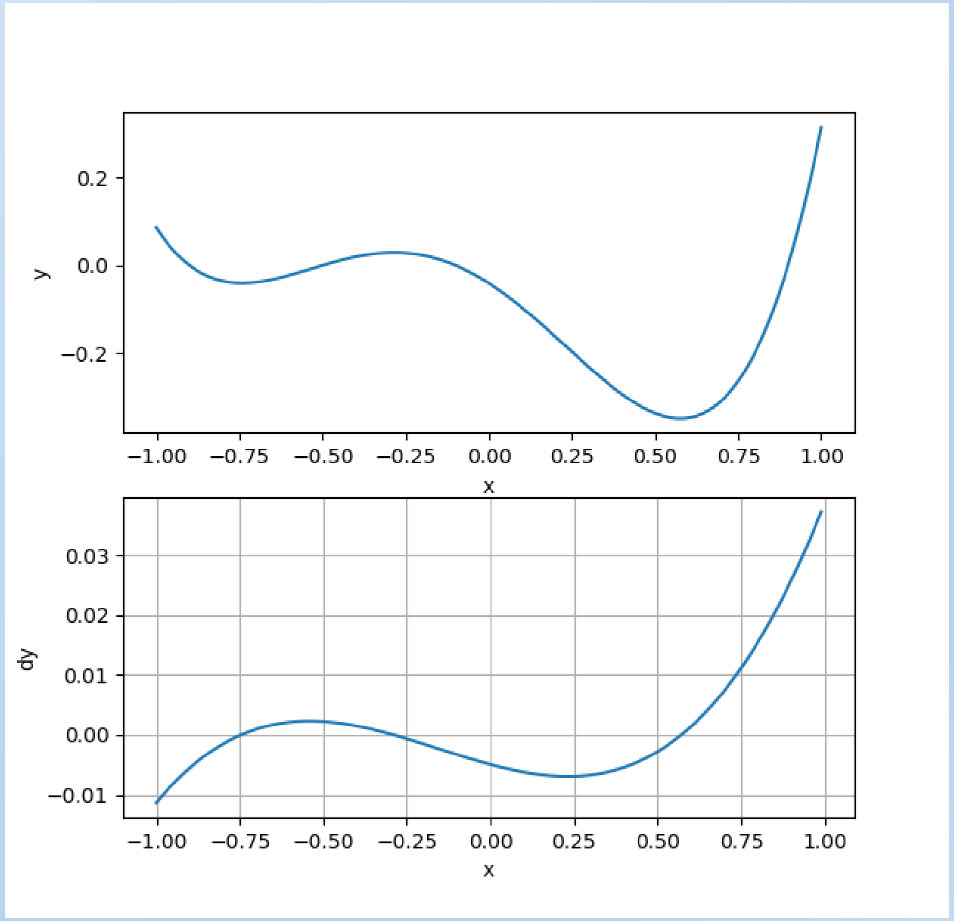
Reminder from calculus:

- To minimize or maximize a function in 1D, we find the point where its derivative is equal to zero.

Similarly,

- To minimize or maximize a function in multiple dimensions, we find the point where its gradient is equal to zero.

For some functions, maxima and minima can be found algebraically. For many others, no algebraic solution exists, so we have to search iteratively.



A function y =f(x) and its derivative

Gradient Descent

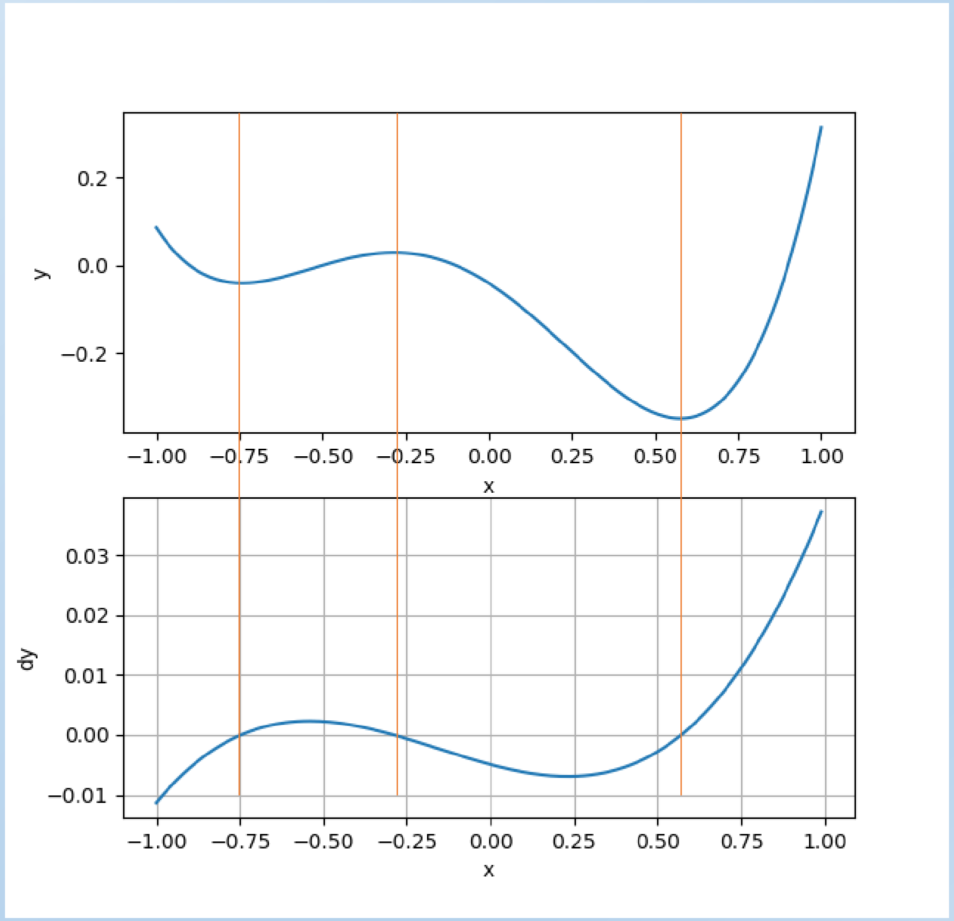
Reminder from calculus:

- To minimize or maximize a function in 1D, we find the point where its derivative is equal to zero.

Similarly,

- To minimize or maximize a function in multiple dimensions, we find the point where its gradient is equal to zero.

For some functions, maxima and minima can be found algebraically. For many others, no algebraic solution exists, so we have to search iteratively.



A function y =f(x) and its derivative
Point where dy/dx =0 correspond to maxima or minima in y

Gradient Descent

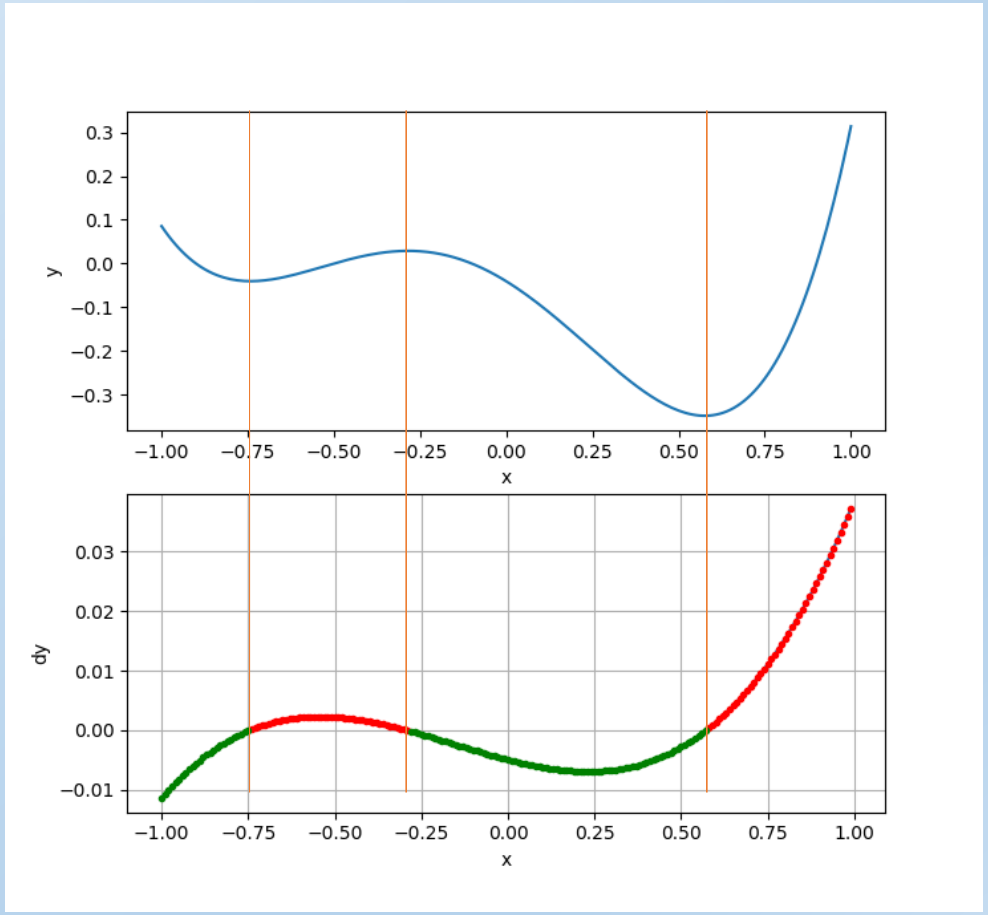
Reminder from calculus:

- To minimize or maximize a function in 1D, we find the point where its derivative is equal to zero.

Similarly,

- To minimize or maximize a function in multiple dimensions, we find the point where its gradient is equal to zero.

For some functions, maxima and minima can be found algebraically. For many others, no algebraic solution exists, so we have to search iteratively.



- dy/dx < 0: increasing x decreases y
- dy/dx > 0: decreasing x decreases y

Gradient Descent

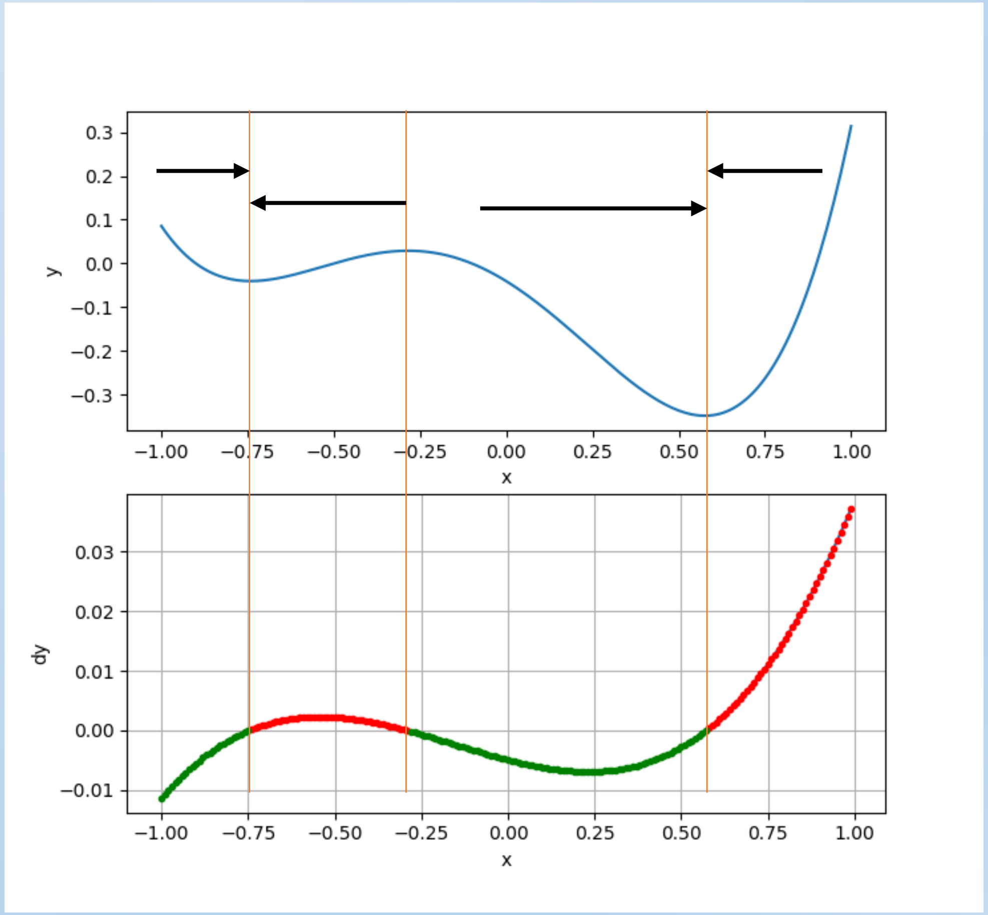
Reminder from calculus:

- To minimize or maximize a function in 1D, we find the point where its derivative is equal to zero.

Similarly,

- To minimize or maximize a function in multiple dimensions, we find the point where its gradient is equal to zero.

For some functions, maxima and minima can be found algebraically. For many others, no algebraic solution exists, so we have to search iteratively.



- dy/dx < 0: increasing x decreases y
- dy/dx > 0: decreasing x decreases y

Gradient Descent

Given training data X,y, we find the model parameters w as follows:

1. Define an error function E(X,y,w) – E must be **differentiable** with respect to w and E(X,y,w) == 0 must correspond to optimal model performance (perfect accuracy or zero MSE)
2. Derive analytical expression for J(w), the gradient of E with respect to w
3. Randomly initialize w
4. Repeat until minimum value of E(X,y,w) is found:
   a. Compute J(w) using the current value of w
   b. Change w to decrease E
      a. $w = - \lambda J(w)$ where λ, the learning rate, is a positive constant

Gradient Descent Example

Learning a linear regression model using gradient descent:

According to the linear regression model:

$pred(x_i) = x_i \cdot w \quad = \sum_{j=0}^{m-1} x_{ij} w_j$

We define the error as:

$E(X, y, w) = \frac{1}{2n} \sum_{i=0}^{n-1} (pred(x_i) - y_i)^2$ $\longrightarrow E(X, y, w)$ is the mean squared error

$E(X, y, w)$ == 0 when $pred(x_i) - y_i$ for all examples $[x_i, y_i]$ in the dataset

The gradient of the error function is:

$$\frac{d(E,y,w)}{dw} = \frac{1}{n} \sum_{i=0}^{n-1} \underbrace{(pred(x_i) - y_i)}_{\text{scalar}} \underbrace{\frac{d(pred(x_i))}{dw}}_{\text{vector of partial derivatives of length } m}$$

Gradient Descent Example

Learning a linear regression model using gradient descent:

Recall that the prediction is given by:

$$pred(x_i) = x_i \cdot w \quad = \sum_{j=0}^{m-1} x_{ij} w_j = \; x_{i0} w_0 + \; x_{i1} w1 + \; \cdots + \quad x_{im\,\_\,1} w_{m\,\_\,1}$$

$$\frac{d(pred(x_i))}{dw} = [\frac{\partial\, pred(x_i)}{\partial w_0}, \cdots, \frac{\partial\, pred(x_i)}{\partial wm_{\_1}}]$$

$$\frac{d(pred(x_i))}{dw} = \; [x_{i0}, x_{i1}, \ldots, x_{im\,\_\,1}]$$

Finally:

$$\left.\frac{d(E,y,w)}{dw} = \frac{1}{n} \sum_{i=0}^{n-1} (pred(x_i) - y_i) \, [\, x_{i0}, xi_1, \ldots, x_{im\,\_\,1}] \quad \right\}$$

Vector of length *m*

Gradient Descent – Python Code

Learning a linear regression model using gradient descent:

```
def gradient_descent(X,y,lr=0.01, tol =0.001,max_it = 1000):
    # It will return w that minimizes (Xw - y)**2
    # It will exit when the mean squared error is less than tol or the number of iterations reaches max_it
    n, m = X.shape                          # the training set has n examples and m attributes
    w = np.random.random((1,m))             # w is 1xm
    for i in range(max_it):
        pred = np.sum(X*w, axis =1)                # pred has len n
        error = (pred - y).reshape(-1,1)           # error is nx1
        mse = np.mean(error**2)                    # mse is a scalar
        if mse<tol:
            break
        gradient = np.mean(X*error,axis=0)         # gradient is 1xm
        w = w - lr*gradient
    return w
```

Logistic Regression

- Despite its name, logistic regression is a classification algorithm
- Simple and effective algorithm for binary classification
- Can be adapted to multiclass problems
- Works with real-valued attributes only
- One of the simplest examples of learning algorithms based on **gradient descent**
- Also known as single-layer perceptron
- Starting point for neural networks and deep learning

Logistic Regression for Binary Classification

We estimate the probability that example *x* belongs to class 1 using the logistic or sigmoid function:

$$p(c(x) == 1) = \sigma(x \cdot w) = \frac{1}{1 + e^{-x \cdot w}}$$

Since classes are binary, we apply a threshold to the probability to make a prediction:

$$pred(x) = \begin{cases} 1 & \text{if } p(c(x) == 1) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

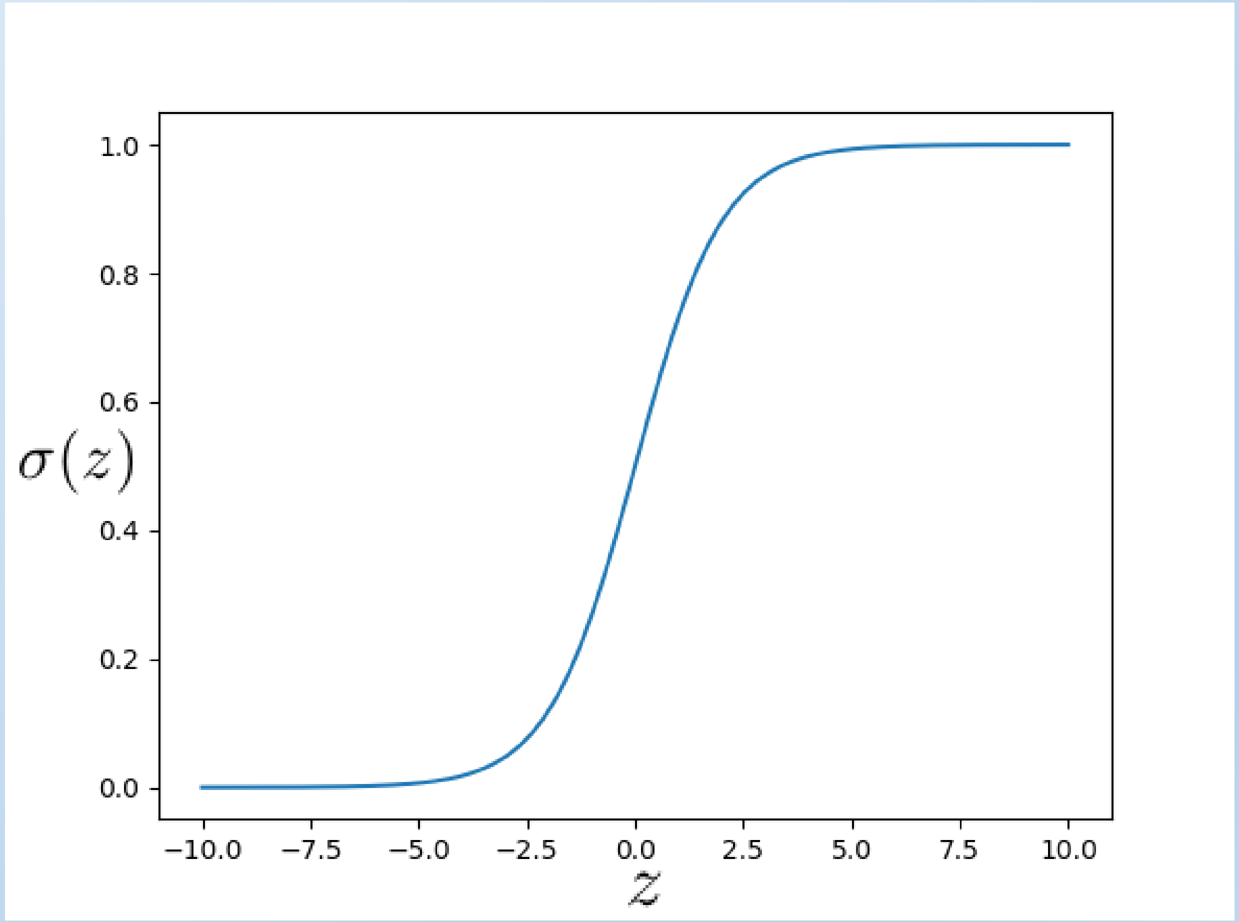
Where $\boldsymbol{x} \cdot \boldsymbol{w}$ is the dot product:

$\boldsymbol{x} \cdot \boldsymbol{w} = x_0 \cdot w_0 + x_1 \cdot w_1 + \ldots + x_{n-2} \cdot w_{n-2} + x_{n-1} \cdot w_{n-1}$

and the parameters $w_0, w_1, \ldots, w_{n-2}, w_{n-1}$ are learned from the training data

The Logistic Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Logistic Regression Learning

We define the error function as the mean squared error, as in linear regression – but consider the output before applying the threshold, since the actual prediction is not differentiable:

$E(X, y, w) = \frac{1}{2n} \sum_{i=0}^{n-1} (\sigma(x_i \cdot w) - y_i)^2$ $\longrightarrow E(X, y, w)$ is the mean squared error

Let's define a new variable $z_i = x_i \cdot w$

$$E(X, y, w) = \frac{1}{2n} \sum_{i=0}^{n-1} (\sigma(z_i) - y_i)^2$$

$$\frac{d(E,y,w)}{dw} = \frac{1}{2n} \sum_{i=0}^{n-1} 2\,(\sigma(z_i) - y_i)\,\frac{d\sigma(z_i)}{dz_i}\frac{dz_i}{dw}$$

$$\frac{d(E,y,w)}{dw} = \frac{1}{n} \sum_{i=0}^{n-1} (\sigma(z_i) - y_i)\,\frac{d\sigma(z_i)}{dz_i}\frac{dz_i}{dw}$$

$$\frac{d\sigma(z_i)}{dz_i} = \sigma(z_i)(1 - \sigma(z_i))$$

$$\frac{dz_i}{dw} = [x_{i0}, xi_1, \ldots, x_{im\,-\,1}]$$

Logistic Regression

Finally:

$$\frac{d(E,y,w)}{dw} = \frac{1}{n}\sum_{i=0}^{n-1} \underbrace{(\sigma(z_i) - y_i)\,\sigma(z_i)\big(1 - \sigma(z_i)\big)}_{\text{scalar}} \underbrace{[x_{i0}{}_{,}\; xi_1, \cdots, x_{im\ \_\ 1}]}_{\text{vector of partial derivatives of length } m}$$

Logistic Regression – Python Code

```
def sigma(X,w):
    z = np.sum(X*w, axis =1)
    return (1/(1+np.exp(-z)))


def logistic_regression_gd(X,y,lr=0.01, tol =0.001,max_it = 1000):
    n, m = X.shape               # the training set has n examples and m attributes
    w =  (np.random.random((1,m))-.5)*.01     # w is 1xm
    for i in range(1,max_it+1):
        s = sigma(X,w)               # s has len n
        error = (s - y)              # error is nx1
        if np.mean(Error**2)<tol: break
        g = (error*s*(1-s)).reshape(-1,1)
        gradient = np.mean(X*g,axis=0)    # gradient is 1xm
        w = w - lr*gradient
   return w
```

Logistic Regression

Multi-class classification

- Use a one-hot representation of the output
- Algorithms learns an array of size k-by-m, where k is the number of classes and m is the number of features
- The rest of the algorithm remains unchanged

Multiclass Logistic Regression – Python Code

```
def sigma(X,W):
    z = np.matmul(X,W.T)
    return (1/(1+np.exp(-z)))


def logistic_regression_gd(X,y,lr=0.01, tol =0.001,max_it = 1000,display_period=10):
    n, m = X.shape                              # the training set has n examples and m attributes
    y_oh = one_hot(y)                           # y_oh has n rows and k=num_classes columns
    W =  (np.random.random((y_oh.shape[1],m))-.5)*.001      # W is k-by-m
    for i in range(1,max_it+1):
        S = sigma(X,W)                          # S is n-by-k
        Error = (S - y_oh)                      #  Error is n-by-k
        if np.mean(Error**2)<tol: break
        G = (Error*S*(1-S)).T                   #  G is k-by-n
        Gradient = np.matmul(G,X)/n             # gradient is k-by-m
        W = W - lr*Gradient
    return W
```

Logistic Regression

Some Improvements:

- **Batch training:**
    - Randomly split training data into small groups – batches – and update weight matrix after processing each batch – more weight updates, randomness may help avoid local minima
- **Adaptive learning rate**:
    - Reduce learning rate after a fixed number of epochs
    - Reduce learning rate when MSE does not decrease after a certain number of epochs
- **Momentum**:
    - Keep a running estimate of the gradient, combining the previous estimate and the gradient computed from the current batch
- **Second order approximation of gradient**:
    - Use second derivatives to obtain a good approximation of gradient in a larger region
- **Label Smoothing:**
    - y_smooth = y_oh(α) + $^{(1-\alpha)}/_{\mathrm{n\_classes}}$ for $\alpha < 1$ - Notice that $\sigma(z) = 1$ when $z = \infty$