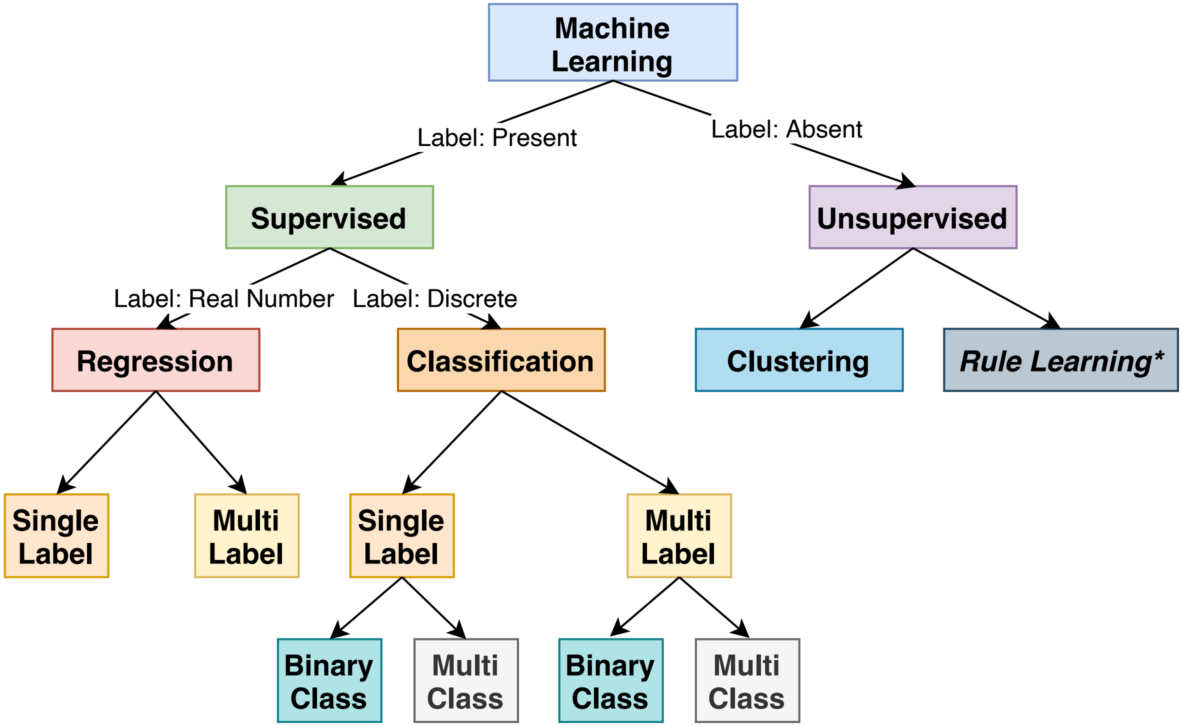
Models of Regression

Machine Learning Techniques

Dr. Ashish Tendulkar

IIT Madras

Regression



It is a supervised learning problem where the output label is a real number.

Two types of regression

Single label regression

Single output label: $y \in \mathbf{R}$

e.g. Predict temperature for tomorrow based on weather features of past days

Multi label regression

Multiple output labels - total labels k: $\mathbf{y} \in \mathbf{R}^k$

e.g. Predict temperature for next 5 days based on weather features of past days

What will be covered in this module?

- Linear regression
- Polynomial regression
- Regularization

All concepts will be discussed for single label regression problem set up.

In the end, we will extend them to multi-label regression problem set up.

Linear Regression

It is a machine learning algorithm that predicts real valued output label based on linear combination of input features.

- Input: Feature vector $\mathbf{x}_{m \times 1}$.
- Label: $y \in \mathbb{R}$, which is a scalar. (*Single label set up)*

Part I: Linear Regression

Data

Data

A set of n ordered pairs of a feature vector, $\mathbf{x}$ and a label y representing examples. We denote it by D:

$$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$$

Features

$\mathbf{X}$ is a feature matrix corresponding to n training examples, each represented with m features and has shape $n \times m$. In this matrix, each feature vector is transposed and represented as a row.

$$\mathbf{X}_{n\times m} = \begin{bmatrix} - \left(\mathbf{x}^{(1)}\right)^T - \\ - \left(\mathbf{x}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{x}^{(n)}\right)^T - \end{bmatrix}$$

Feature vector for i-th training example $\mathbf{x}^{(i)}$ can be obtained by $\mathbf{X}[i]$

Labels

For single label problem, $\mathbf{y}$ is a label vector of shape $n \times 1$.

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

The i-th entry in this vector, $\mathbf{y}[i]$ gives label for i-th example, which is denoted by $y^{(i)} \in \mathbb{R}$

Data: Example

$$D = \{(\mathbf{X}, \mathbf{y})\} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$$

Sample House Pricing data is presented below.

$$\mathbf{X}_{n \times m} = \begin{bmatrix} - & \left(\mathbf{x}^{(1)}\right)^T & - \\ - & \left(\mathbf{x}^{(2)}\right)^T & - \\ & \vdots & \\ - & \left(\mathbf{x}^{(n)}\right)^T & - \end{bmatrix}$$

$$\mathbf{X}_{4 \times 3} = \begin{array}{c|ccc|l} & \text{Area} & \text{No. of Floors} & \text{Age} & \\ \hline & 1000 & 1 & 15 & \text{House 1} \\ & 1500 & 2 & 25 & \text{House 2} \\ & 1100 & 3 & 10 & \text{House 3} \\ & 2000 & 1 & 40 & \text{House 4} \end{array}$$

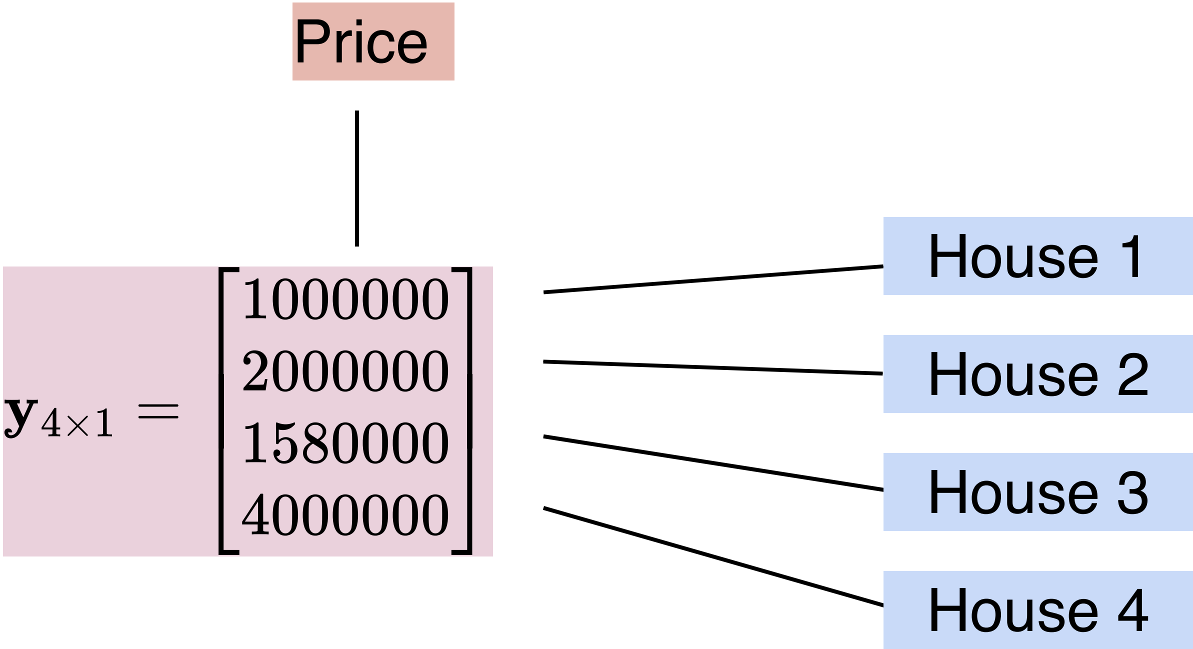
Features (m=3)

Examples (n=4)

Data: Example

House Pricing data, labels (price in Rs.)

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$



$$\mathbf{y}_{4\times1} = \begin{bmatrix} 1000000 \\ 2000000 \\ 1580000 \\ 4000000 \end{bmatrix}$$

Data: Example

$$D = \{(\mathbf{X}_{4\times 3}, \mathbf{y}_{4\times 1})\} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{4}$$

$$\mathbf{X}_{4\times 3} = \begin{array}{ccc} \text{Area} & \text{No. of Floors} & \text{Age} \\ \end{array} \atop \begin{bmatrix} 1000 & 1 & 15 \\ 1500 & 2 & 25 \\ 1100 & 3 & 10 \\ 2000 & 1 & 40 \end{bmatrix}$$

$$\mathbf{y}_{4\times 1} = \begin{array}{cc} \text{Price} & \\ \begin{bmatrix} 1000000 \\ 2000000 \\ 1580000 \\ 4000000 \end{bmatrix} & \begin{array}{l} \text{House 1} \\ \text{House 2} \\ \text{House 3} \\ \text{House 4} \end{array} \end{array}$$

Model

Model

Label y is computed as a linear combination of features $x_1, x_2, ..., x_m$:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$$

Here $w_0, w_1, w_2, \ldots, w_m$ are weights

- Each feature x_j has a corresponding weight w_j.
- Let's introduce a dummy feature $x_0 = 1$ for weight w_0:

$$y = w_0 x_0 + w_1 x_1 + \ \ldots + w_m x_m$$

It can be written compactly as

$$y = \sum_{i=0}^{m} w_i x_i$$

How does the model look like?

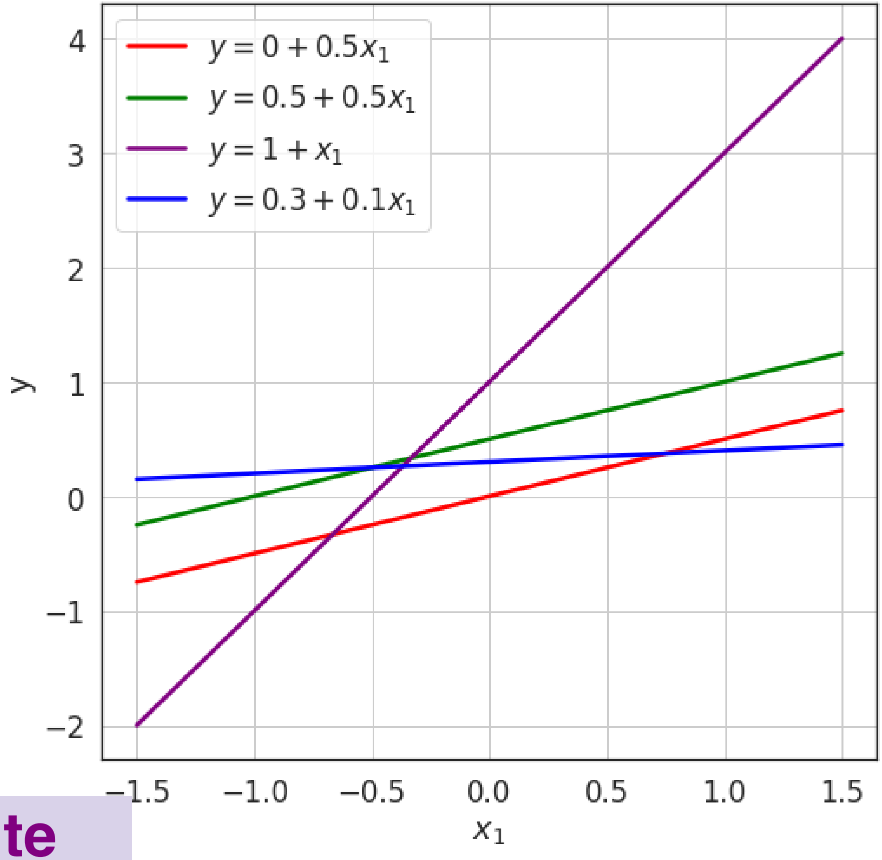
Take a simple model with a single feature x_1:

$$y = w_0 + w_1 x_1$$

# model parameters (2): w_0, w_1

Each new combination of w_0, w_1 leads to a new model.

Total # of possible models = infinite



Model: geometry and # parameters depends on # features

# Features (m)	Model	Geometry	# Params (# w)
1	$y = w_0 + w_1 x_1$	line	2
2	$y = w_0 + w_1 x_1 + w_2 x_2$	plane	3
≥ 3	$y = \sum_{i=0}^{m} w_i x_i$	hyperplane in (m +1)-D	$m + 1$

Model: Vectorized Form

$$y = \sum_{j=0}^{m} w_j x_j = \mathbf{w}^T \mathbf{x}$$

Exercise: Check $\mathbf{w}^T\mathbf{x} = \sum w_j x_j$

Solve vector matrix product and verify it corresponds to the linear combination of features.

All weights $w_0, w_1, \ldots, w_m$ can be arranged in a vector $\mathbf{w}$ with shape $(m+1) \times 1$

$$\mathbf{w}_{(m+1)\times 1} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

Exercise: Check $\mathbf{w}^T\mathbf{x} = \sum w_j x_j$

The features for an i-th example can also be represented as a feature vector $\mathbf{x}^{(i)}$. Note that we add a special feature x_0.

$$\mathbf{x}^{(i)}_{(m+1)\times 1} = \begin{bmatrix} x_0 \\ x^{(i)}_1 \\ \vdots \\ x^{(i)}_m \end{bmatrix}$$

Setting $x_0 = 1$

$$\mathbf{x}^{(i)}_{(m+1)\times 1} = \begin{bmatrix} 1 \\ x^{(i)}_1 \\ \vdots \\ x^{(i)}_m \end{bmatrix}$$

Exercise: Check $\mathbf{w}^T\mathbf{x} = \sum w_j x_j$

Let's multiply these vectors to obtain $y^{(i)}$:

$$\begin{aligned} y^{(i)} &= \mathbf{w}^T\mathbf{x} \\ &= \left(\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}\right)^T \times \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix} \\ &= \begin{bmatrix} w_0 & w_1 & \ldots & w_m \end{bmatrix} \times \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix} \end{aligned}$$

Exercise: Check $\mathbf{w}^T \mathbf{x} = \sum w_j x_j$

$$y^{(i)} = \begin{bmatrix} w_0 & w_1 & \dots & w_m \end{bmatrix} \times \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$$

$$y^{(i)} = w_0 \times \ 1 + w_1 \times x_1^{(i)} + \dots + w_m \times \ x_m^{(i)}$$

$$y^{(i)} = w_0 x_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_m x_m^{(i)}$$

$$\boxed{y^{(i)} = \sum_{j=0}^{m} w_j {x_j}^{(i)}}$$

Vectorized implementation: Multiple Examples

A weight vector $\mathbf{w}$ with shape $(m+1) \times 1$.

A feature matrix $\mathbf{X}$ has shape $(n, m+1)$ containing features for n examples.

$$\mathbf{w}_{(m+1)\times 1} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

$$\mathbf{X}_{n\times(m+1)} = \begin{bmatrix} - \left(\mathbf{x}^{(1)}\right)^T - \\ - \left(\mathbf{x}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{x}^{(n)}\right)^T - \end{bmatrix}$$

The label vector $\widehat{\mathbf{y}}$ containing labels for all the n examples can be computed as follows:

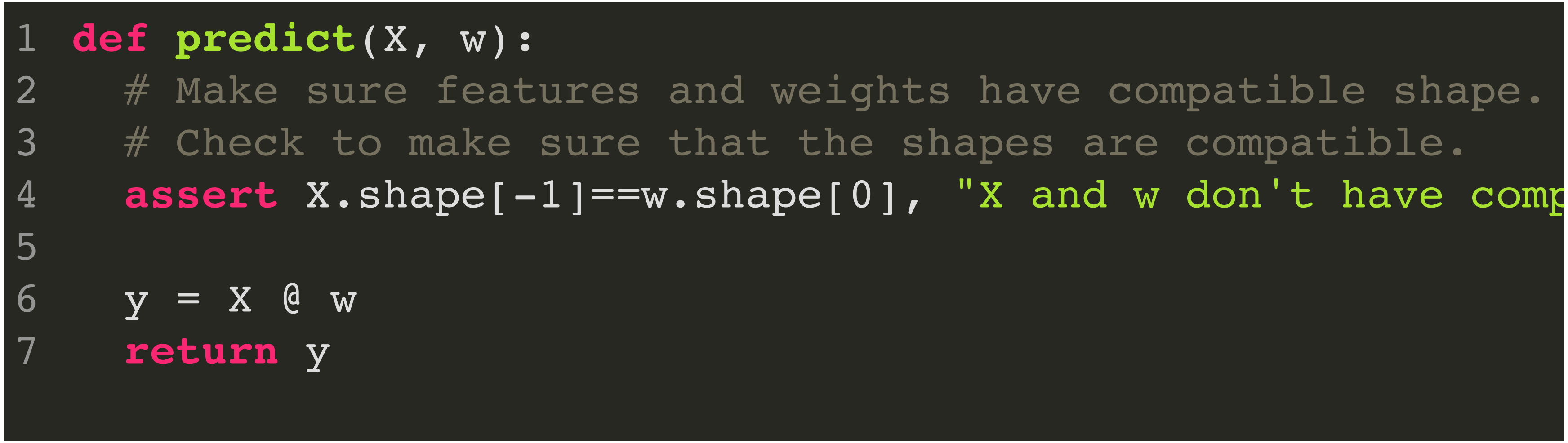
$$\widehat{\mathbf{y}}_{n\times 1} = \mathbf{X}_{n\times(m+1)}\mathbf{w}_{(m+1)\times 1}$$

$$\widehat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

Model Inference

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

Implement the model inference in vectorized form:



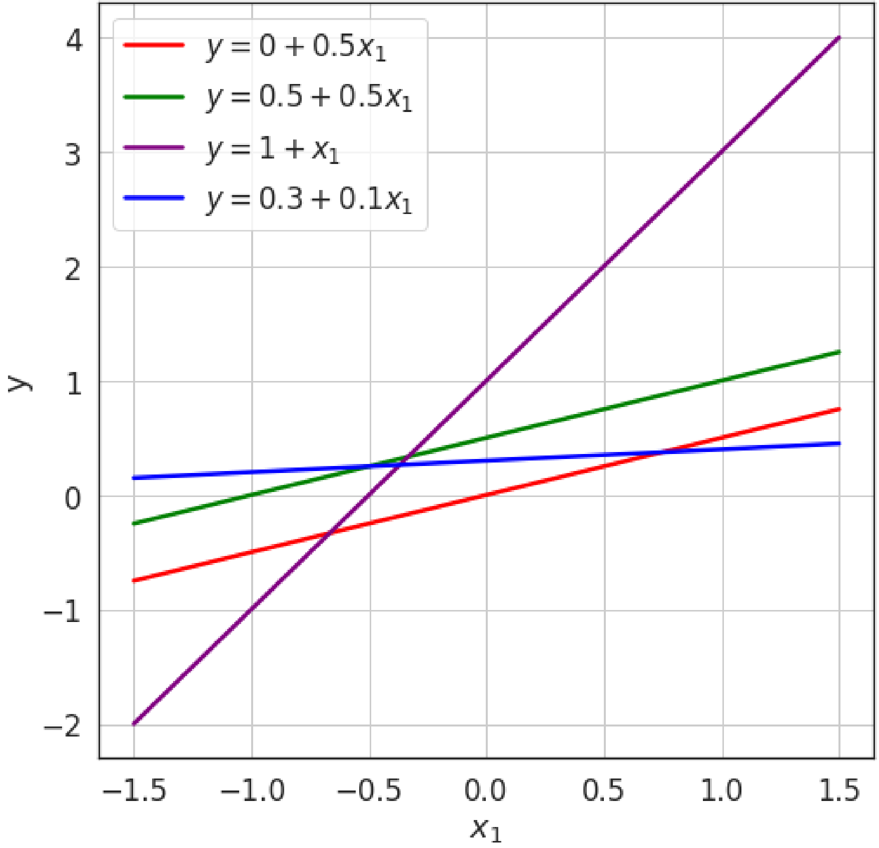
```
def predict(X, w):
  # Make sure features and weights have compatible shape.
  # Check to make sure that the shapes are compatible.
  assert X.shape[-1]==w.shape[0], "X and w don't have comp

  y = X @ w
  return y
```

Choosing best-fit model

- Let's choose a simple model for exploring this question:

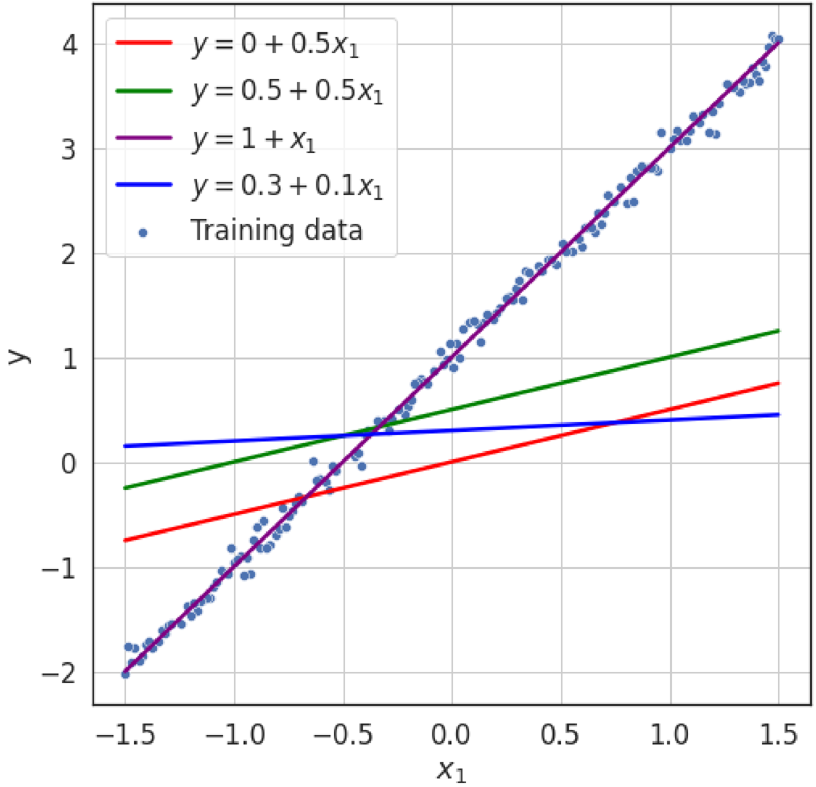
$$y = w_0 + w_1 x_1$$

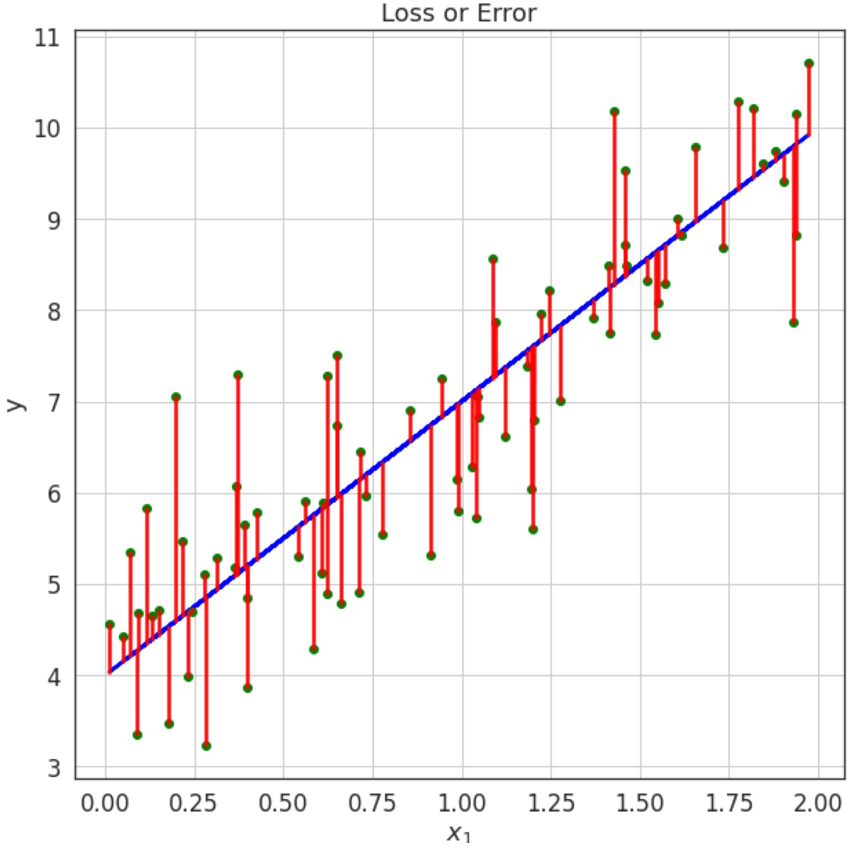
- As we vary (w_0, w_1), we get a new model.



Evaluate data fitment of model

Loss or error function

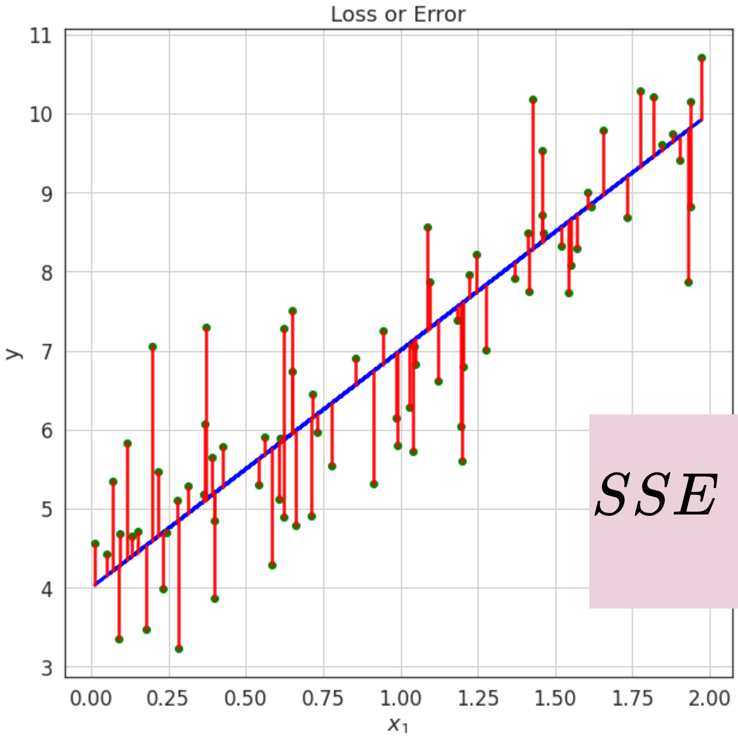




- Which one fits the training data better than all other models?
- We need a measure of fitment or mis-fitment.

Loss Function

Loss Function: Sum of Squared Error (SSE)



$$SSE = \sum_{i=1}^{n} \left(\text{predicted label}^{(i)} - \text{actual label}^{(i)} \right)^2$$

SSE is the sum of square of difference between actual and predicted labels for each training point.

Error at i-th training point

$$\begin{aligned} e^{(i)} &= (\text{predicted label} - \text{actual label})^2 \\ &= \left(h_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2 \\ &= \left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)^2 \end{aligned}$$

- The error $e^{(i)}$ is non-negative or $e^{(i)} \geq 0$.
- $e^{(i)} = 0$, whenever the actual label and the predicted label are identical.

Loss Function: SSE

The total loss $J(\mathbf{w})$ is sum of errors at each training point:

$$J(\mathbf{w}) = \sum_{i=1}^{n} e^{(i)}$$

We divide this by $\frac{1}{2}$ for mathematical convenience in later use:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^{(i)} = \frac{1}{2}\sum_{i=1}^{n}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)^2$$

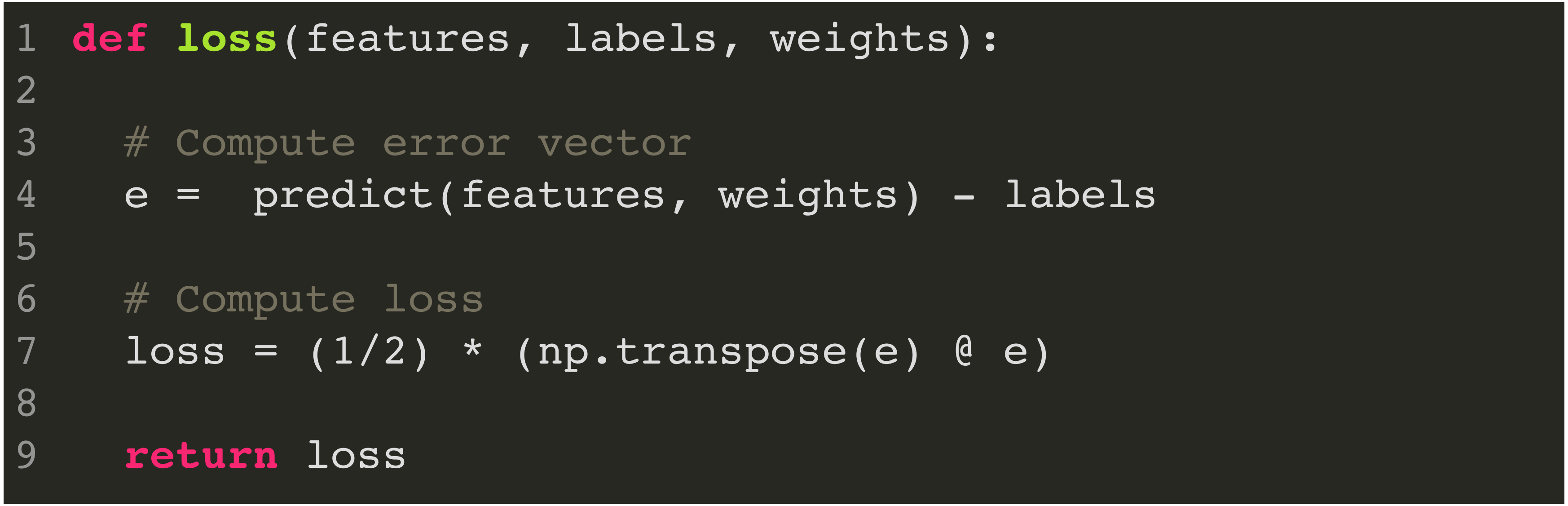
The loss is dependent on the value of $\mathbf{w}$ - as these values change, we get a new model, which will result in different prediction and hence different error at each training point.

SSE vectorization for efficient computation

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} \left(\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

$$= \tfrac{1}{2} \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right)$$

Recall, $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$



```python
def loss(features, labels, weights):

  # Compute error vector
  e =  predict(features, weights) - labels

  # Compute loss
  loss = (1/2) * (np.transpose(e) @ e)

  return loss
```

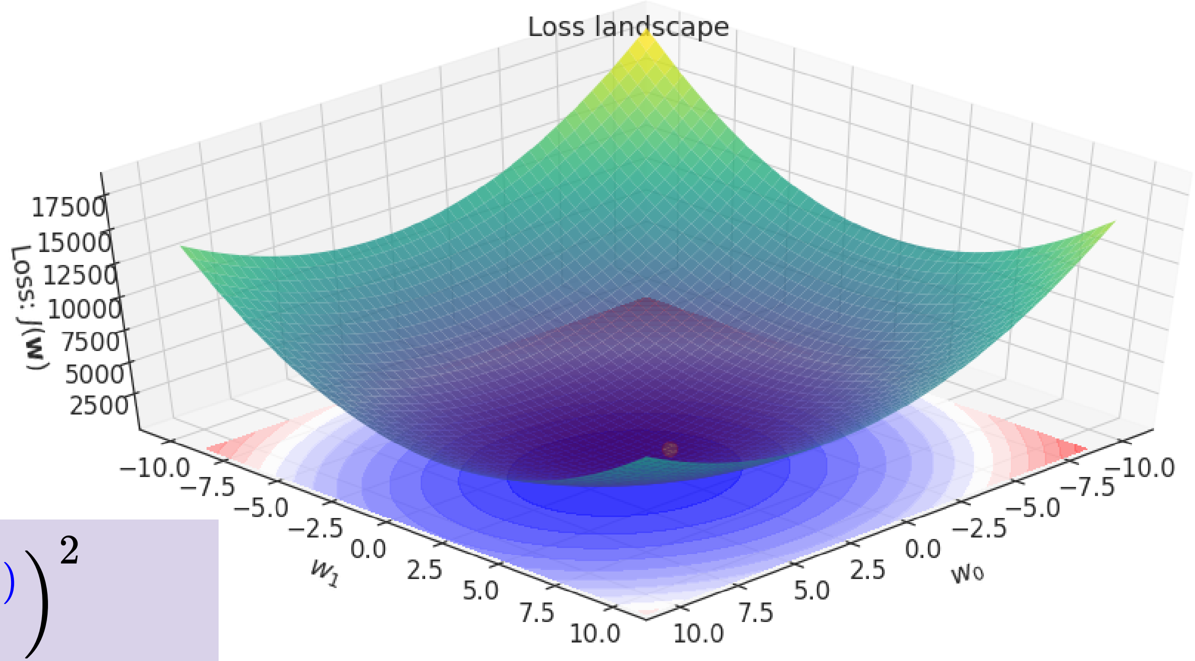
Loss function visualization

Model

$$h_{\mathbf{w}}(\mathbf{x}) : y = w_0 + w_1 x_1$$

Loss (SSE)

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(h_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$
$$= \frac{1}{2}\sum_{i=1}^{n}\left((w_0 + w_1 x_1^{(i)}) - y^{(i)}\right)^2$$



Duality of loss and model spaces

https://www.geogebra.org/material/iframe/id/evjby7mb/width/1020/height/500/border/888888/sfsb/true/smb/false/stb/false/stbh/false/ai/false/asb/false/sri/false/rc/false/ld/false/sdz/true/ctl/false

How do we find the model with the least loss or error?

Optimization

Optimization

Key calculation is the derivative of loss function $J(\mathbf{w})$ with respect to $\mathbf{w}$.

Let's look at a couple of useful identities for this:

Derivative of linear function: $\frac{\partial}{\partial \mathbf{w}}(\mathbf{w}\mathbf{x}) = \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{x}^T\mathbf{w}\right) = \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^T\mathbf{x}\right) = \mathbf{x}$

Similar to $\frac{d}{dw}(xw) = x$

Derivative of quadratic function: $\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^T\mathbf{X}\mathbf{w}\right) = 2\mathbf{X}\mathbf{w}$

Similar to $\frac{d}{dw}(xw^2) = 2xw$

Computing gradient of loss function

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left\{ \frac{1}{2} \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T \right) \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T\mathbf{X}^T - \mathbf{y}^T \right) \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T\mathbf{X}^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) - \mathbf{y}^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} \right) - \left(\mathbf{y}^T\mathbf{X}\mathbf{w} - \mathbf{y}^T\mathbf{y} \right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} - {\color{blue}\mathbf{y}^T\mathbf{X}\mathbf{w}} + \mathbf{y}^T\mathbf{y} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} - {\color{blue}\mathbf{w}^T\mathbf{X}^T\mathbf{y}} + \mathbf{y}^T\mathbf{y} \right\}
\end{aligned}
$$

Computing gradient of loss function

$$\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2}\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}\right\} \\
&= \frac{1}{2}\left\{\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X}\right\} - \frac{\partial}{\partial \mathbf{w}}\left\{2\mathbf{w}^T\mathbf{X}^T\mathbf{y}\right\} + \frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{y}^T\mathbf{y}\right\}\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 0\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y}\right\} \\
&= \frac{1}{2} \times 2 \times \left\{\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}\right\} \\
&= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} \\
&= \mathbf{X}^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)
\end{aligned}$$

How to obtain $\mathbf{w}$?

- Normal equation

Set the partial derivative to 0 and solve with analytical method to obtain the weight vector

- Gradient descent

Iteratively change the weights based on the partial derivative of the loss function until convergence (Iterative optimization)

Normal Equation

Let's set $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ to 0 and solve for $\mathbf{w}$:

$$\begin{aligned} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= 0 \\ \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} &= 0 \\ \mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\ \left(\mathbf{X}^T\mathbf{X}\right)\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\ \mathbf{w} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \end{aligned}$$

Recall $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}$

Gradient Descent: Non Vectorized

1. Randomly initialize the weight vector $\mathbf{w}$. One possible initialization can be: $w_0 = 0, w_1 = 0, \ldots, w_m = 0$.

Gradient Descent: Non Vectorized

2. Iterate until convergence:
   a. Calculate gradient of loss function w.r.t. the weights w_0, w_1 .... w_m, which are $\frac{\partial J(\mathbf{w})}{\partial w_0}$, $\frac{\partial J(\mathbf{w})}{\partial w_1}$ .... $\frac{\partial J(\mathbf{w})}{\partial w_m}$ respectively.

Gradient Descent: Non Vectorized

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \sum_{i=1}^{n}(w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)})x_0^{(i)}$$

$$= \sum_{i=1}^{n}(w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)})1$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \sum_{i=1}^{n}(w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)})x_1^{(i)}$$

$$\vdots$$

$$\frac{\partial J(\mathbf{w})}{\partial w_m} = \sum_{i=1}^{n}(w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)})x_m^{(i)}$$

Gradient Descent: Non Vectorized

b. Calculate updated parameters:

$$
\begin{aligned}
w_0(\textbf{new}) &:= \quad w_0 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_0} \\
w_1(\textbf{new}) &:= \quad w_1 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_1} \\
&\vdots \\
w_m(\textbf{new}) &:= \quad w_m - \alpha \frac{\partial J(\mathbf{w})}{\partial w_m}
\end{aligned}
$$

Here α is learning rate.

Gradient Descent: Non Vectorized

c. Update parameters simultaneously:

$$
\begin{aligned}
w_0 &= w_0(\text{new}) \\
w_1 &= w_1(\text{new}) \\
&\vdots \\
w_m &= w_m(\text{new})
\end{aligned}
$$

After this step, we have a new weight vector.

Gradient Descent: Non Vectorized

- Note that the weights are updated to their new values simultaneously in the last step of GD .

- Until then the old values are used for predicting labels of examples while calculating the parameter updates.

Gradient Descent: Vectorized

$$
\begin{aligned}
\mathbf{w}_{k+1} &:= \mathbf{w}_k - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \\
&:= \mathbf{w}_k - \alpha \left(\mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{Y} \right) \\
&:= \mathbf{w}_k - \alpha \mathbf{X}^T \left(\mathbf{X} \mathbf{w} - \mathbf{Y} \right) \\
&:= \mathbf{w}_k - \alpha \mathbf{X}^T \left(\text{predicted output} - \text{actual output} \right)
\end{aligned}
$$

This vectorized implementation will make sure that all the parameters are updated in one go.

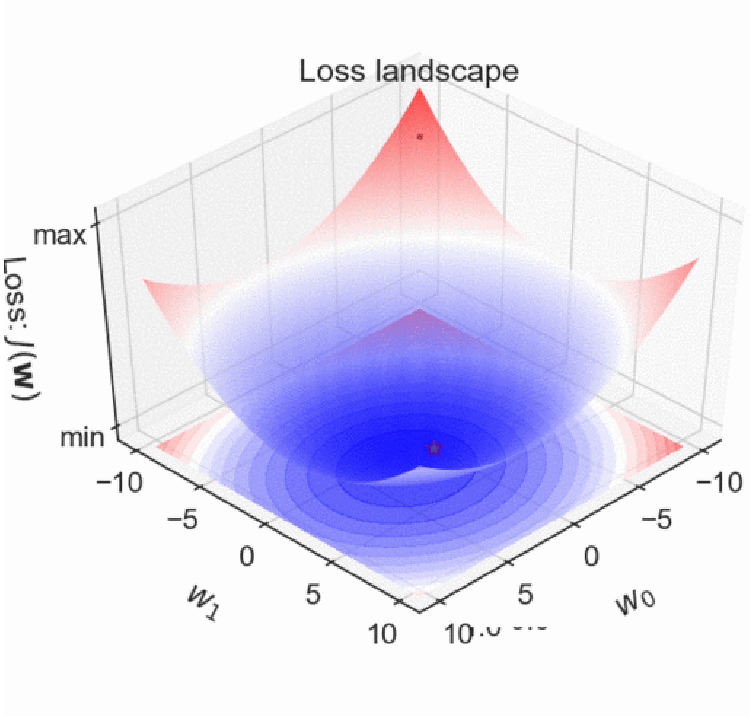
Gradient Descent Visualization

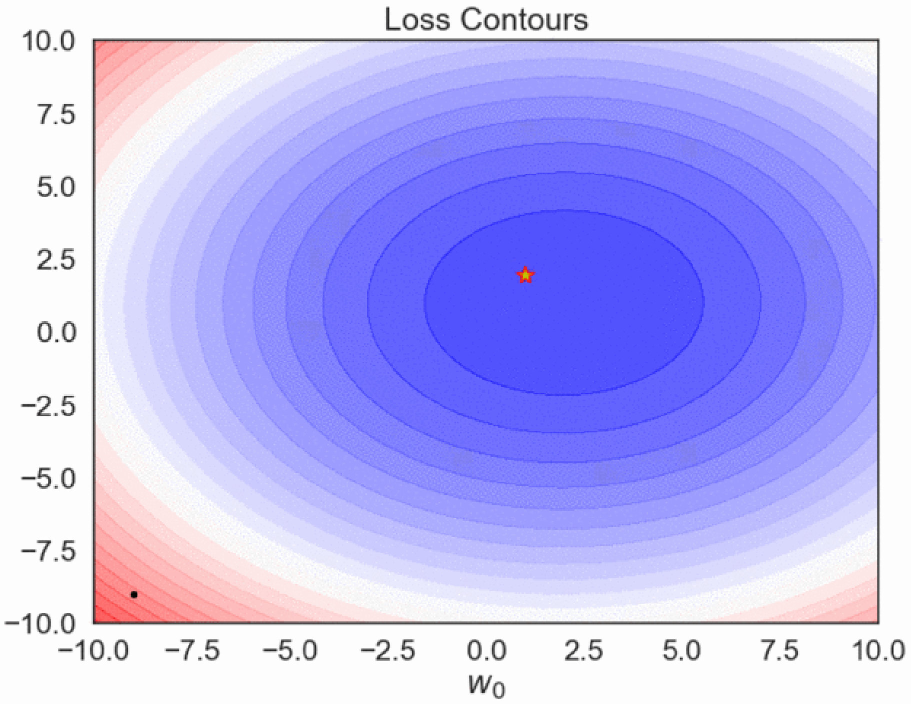
Model

$$h_{\mathbf{w}}(\mathbf{x}) : y = w_0 + w_1 x_1$$

Loss (SSE)

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} \left(\left(w_0 + w_1 x_1^{(i)} \right) - y^{(i)} \right)^2$$





Gradient Descent: Salient Features

- GD is a very generic optimization algorithm that can be used for learning weight vectors of most of the ML models.
- GD obtains optimal weight vector by making small changes to their values in each iteration proportional to the gradient of the loss function.
- Once GD reaches the minima of the loss function, we obtain the optimal weights corresponding to the minima.

Gradient Descent Convergence

- The weight vector changes by a very small margin in successive iterations or in last few iterations.

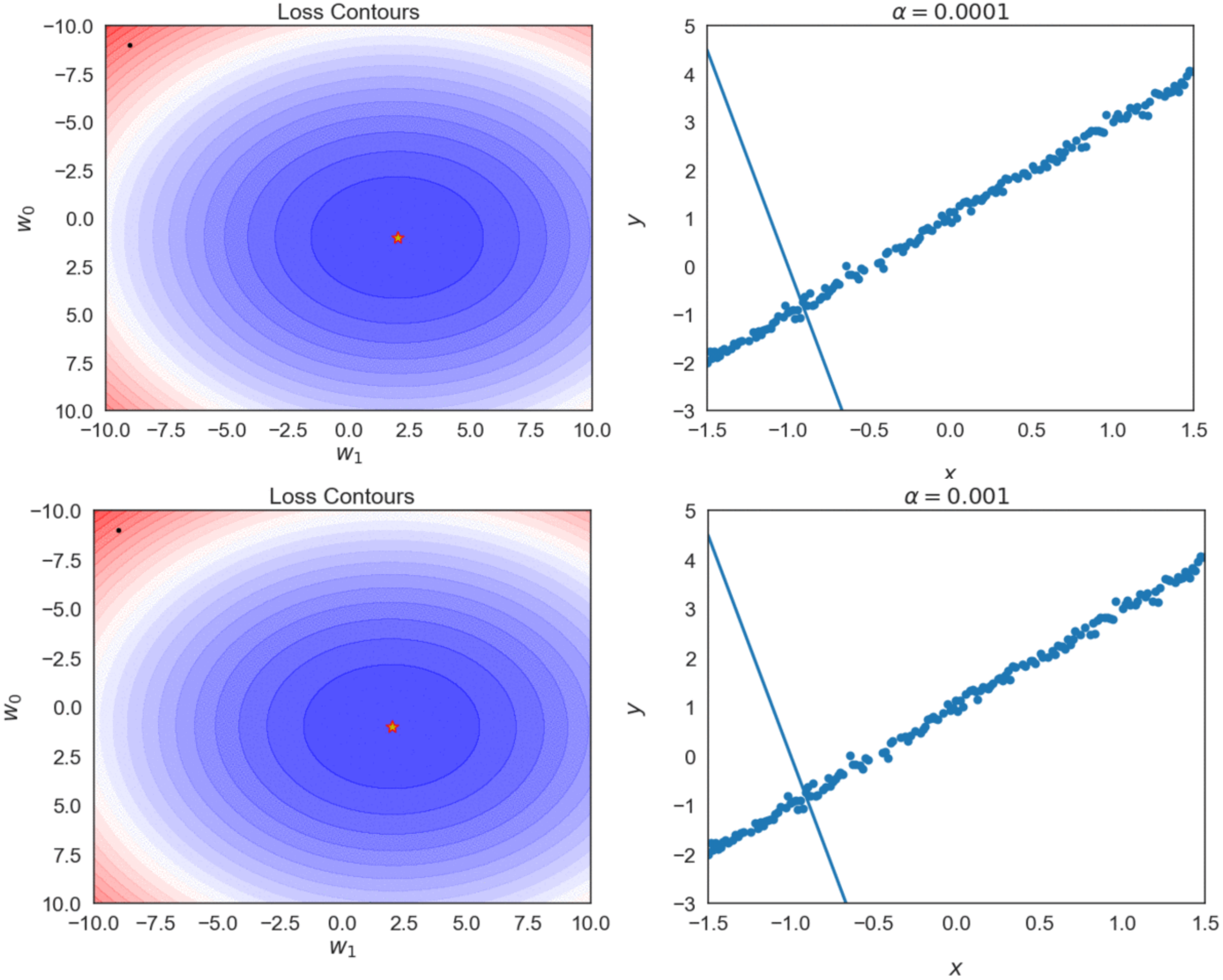
- After completing a fixed number of iterations.

Gradient Descent: Practical considereations

1. How do we set the learning rate α ?

2. How do we decide the number of iterations?

Fixing learning rate α

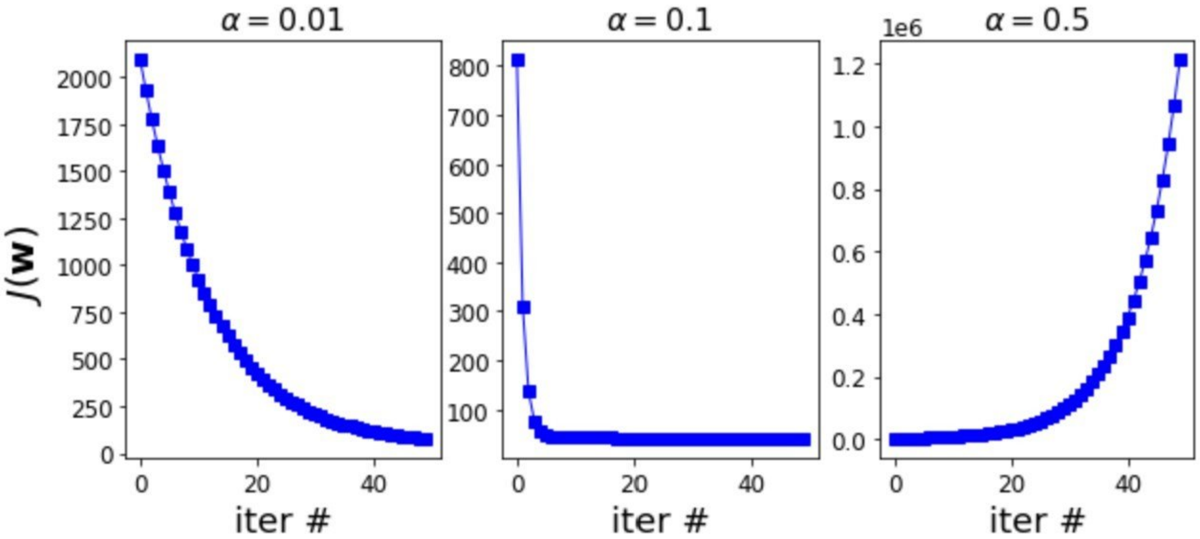


Try different values of α: $\{0.0001, 0.001, 0.01, 0.1, 1\}$

Top one, with $\alpha = 0.0001$ takes longer to reach optimal point, compared with $\alpha = 0.001$

Visualization based diagnosis is not possible for real-word problems with more features, learning curves are our best friends in general case.

Fixing α: Diagnosis based on learning curves



- After first 50 iterations, $\alpha = 0.1$(middle one) has the lowest loss. Loss decreases rapidly.
- The loss for $\alpha = 0.01$ slowly decreases.
- The loss in case of $\alpha = 0.5$ rather increases.

just right α

small α

too large α

Fixing α

- [Small α]
  - Loss either does not reduce or reduces very slowly after initial few iterations points to too small α.
  - Stop the training, increase α (typically by 10x), restart the training and again observe learning curves.

- [Too large α]
  - Loss increases rather than decreasing points to too large α.
  - Stop the training, decrease α (typically by 10x), restart the training and again observe the learning curves.

Fixing the number of iterations

- **Too few iterations:** not enough iterations to reach the optimal solution.
- **Too many iterations:** once we reach the optimal solution, we end up wasting computational cycles as subsequent runs hardly result in any changes in $\mathbf{w}$.
- In practice, we set the number of iterations to a sufficiently large number, but add a convergence criteria which terminates GD loop as soon as the gradient vector becomes smaller than some threshold ϵ.

Variations of GD

A couple of variations for faster weight updates and hence faster convergence. Note that GD uses all n training examples for weight updates:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) x_j^{(i)}$$

Mini-batch gradient descent (MBGD)

Uses $k << n$ examples for weight update in each iteration.

Stochastic gradient descent (SGD)

Uses $k = 1$ examples for weight update in each iteration.

Mini batch Gradient Descent (MBGD)

The key time consuming step in GD is the the gradient computation, which performs summation over all training examples:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) x_j^{(i)}$$

- This summation is carried out for each $m + 1$ weight in each iteration.
- Exploits the fact that training examples are independent and identically distributed thereby uses $k << n$ examples for faster updates.

Mini batch Gradient Descent (MBGD)

- The key idea is to use a small batch of examples for calculating the gradient in each iteration.
- For that the training set is partitioned into batches of $k << n$ examples.
- In each iteration, MBGD performs the weight update using k examples.
- It needs total of $\frac{n}{k}$ iterations to process the entire training set.
- One full pass over the training set is called an **epoch** and it has $\frac{n}{k}$ iterations and hence performs $\frac{n}{k}$ weight updates.

Mini batch Gradient Descent Algorithm

- Randomly initialize the parameter vector $\mathbf{w}$.
- Iterate until convergence:

  1. **for every batch**:

     1. Calculate gradient of loss function w.r.t. the weights.
     2. Set weights to their new values.
     3. Update weights simultaneously.

All steps are same as GD except *each step processes a small number of examples*.

Stochastic Gradient Descent (SGD)

When we use $k = 1$ in mini-batch GD, it is called stochastic gradient descent (SGD).

Total weight updates after processing full training set once:

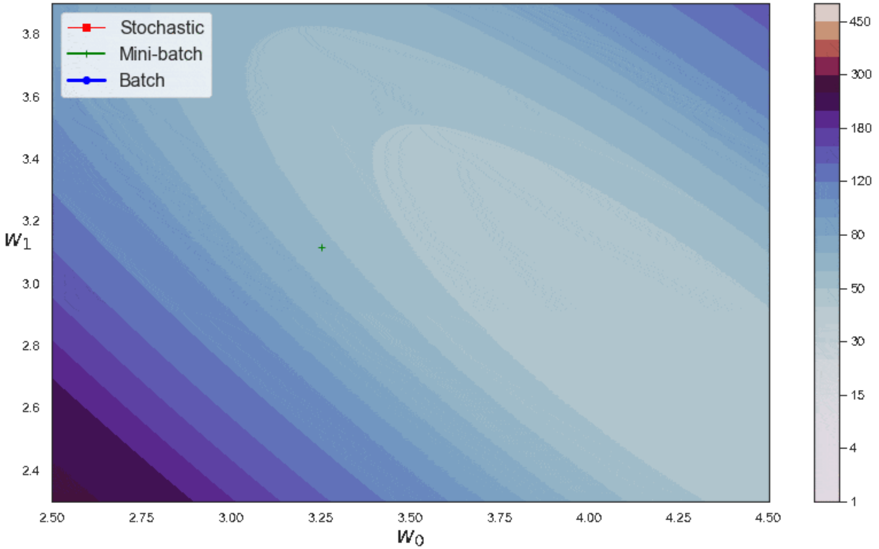
- GD: 1
- MBGD: $\frac{n}{k}$
- SGD: n

GD, MBGD and SGD: Convergence characteristics

- Plot the trajectory of optimization algorithm in the weights' space. In other words, plot the weights as obtained from the first to the last iteration in that order.

GD, MBGD and SGD: Convergence trajectories

Step by step trajectories



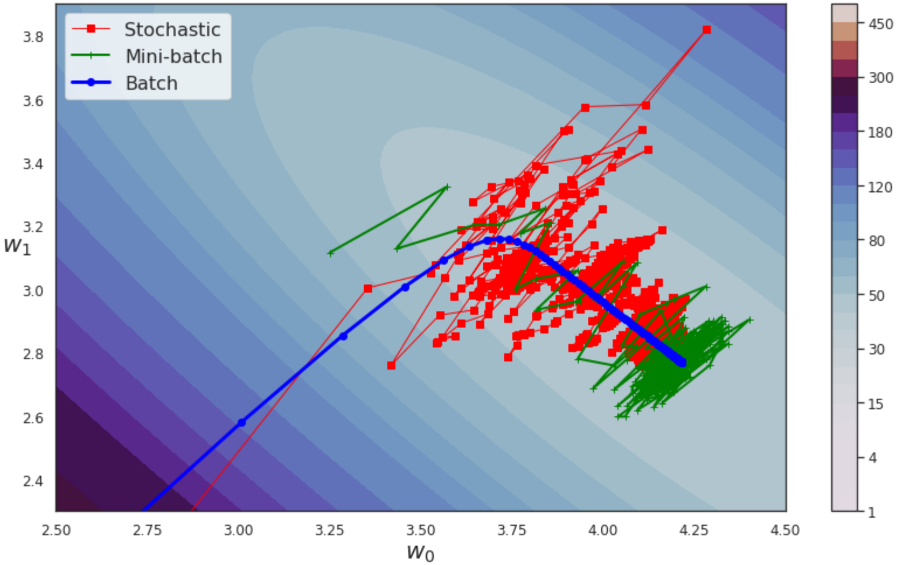
- Batch GD have a smooth path to the minima, though it takes longer time to reach there.

- SGD is more erratic in its path than mini-batch GD. Why?

SGD computes weight updates based on a single example as against k examples used in mini-batch GD.

GD, MBGD and SGD: Convergence trajectories

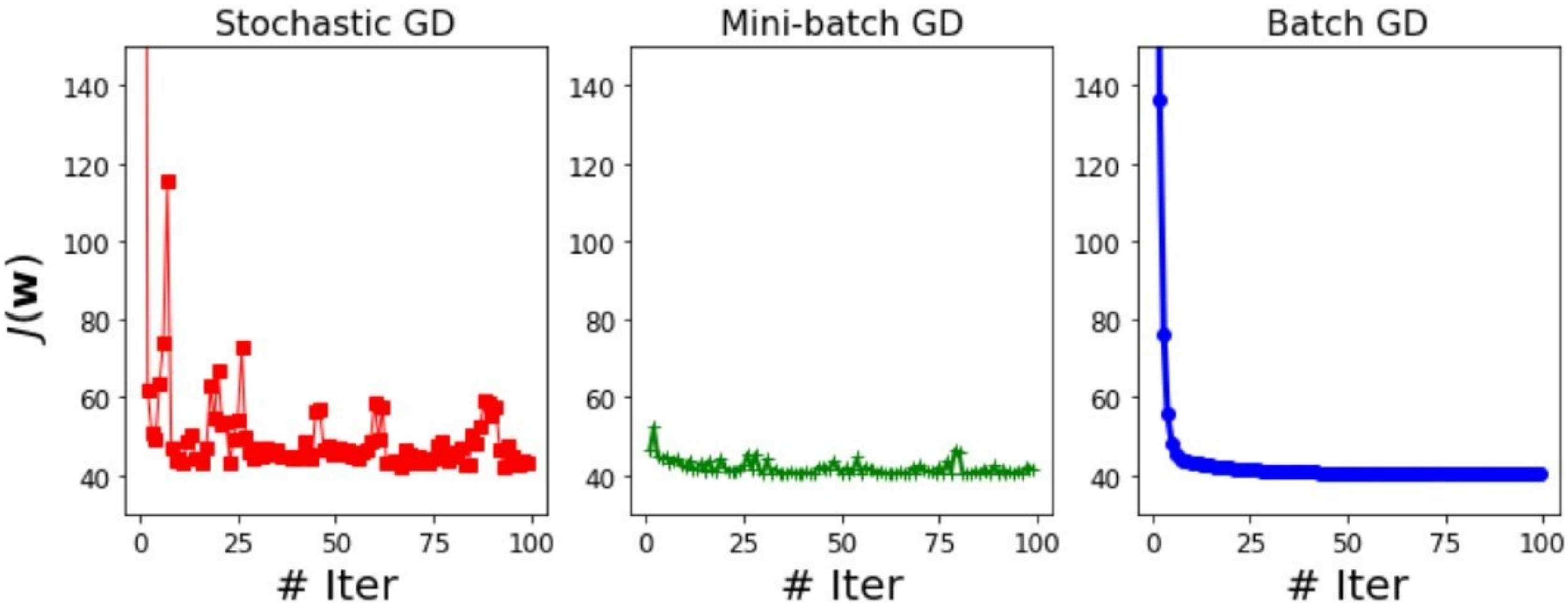
- Observe the trajectory of three variants and compare it with one another.



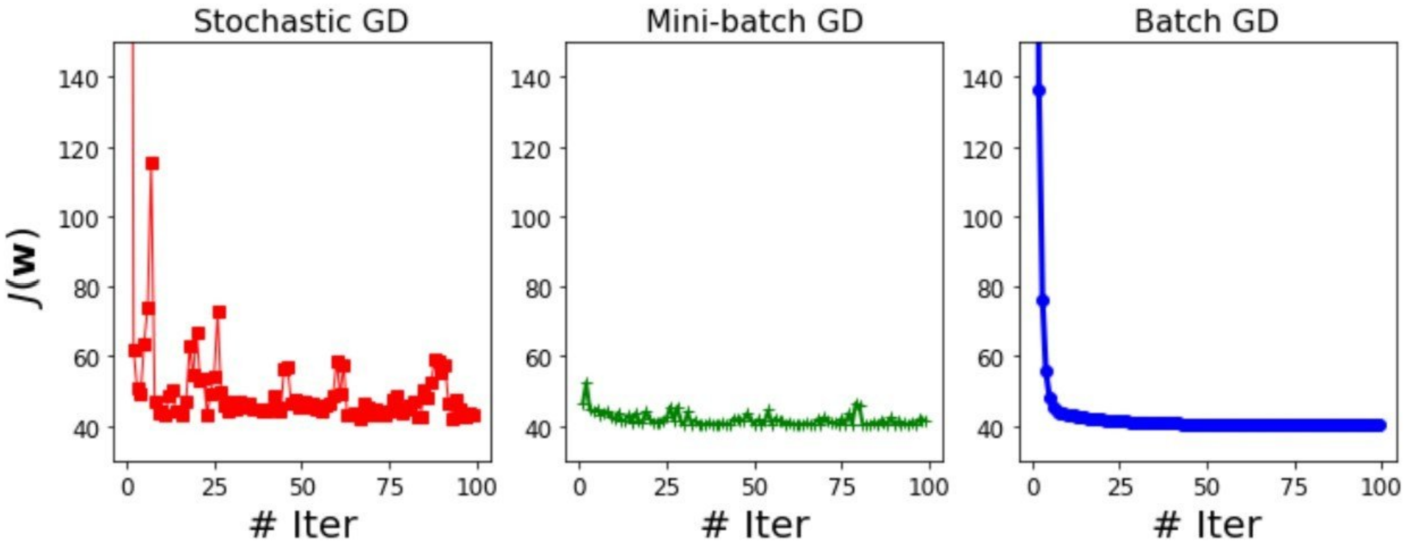
- All optimizers end up around the actual minima.
- Batch GD ends in the minima, while the other two end up around minima. They can reach the minima with appropriate learning schedule.

Visualization based diagnosis is not possible for real-word problems with more features, learning curves are our best friends in general case.

GD vs MBGD vs SGD: Learning curve comparison

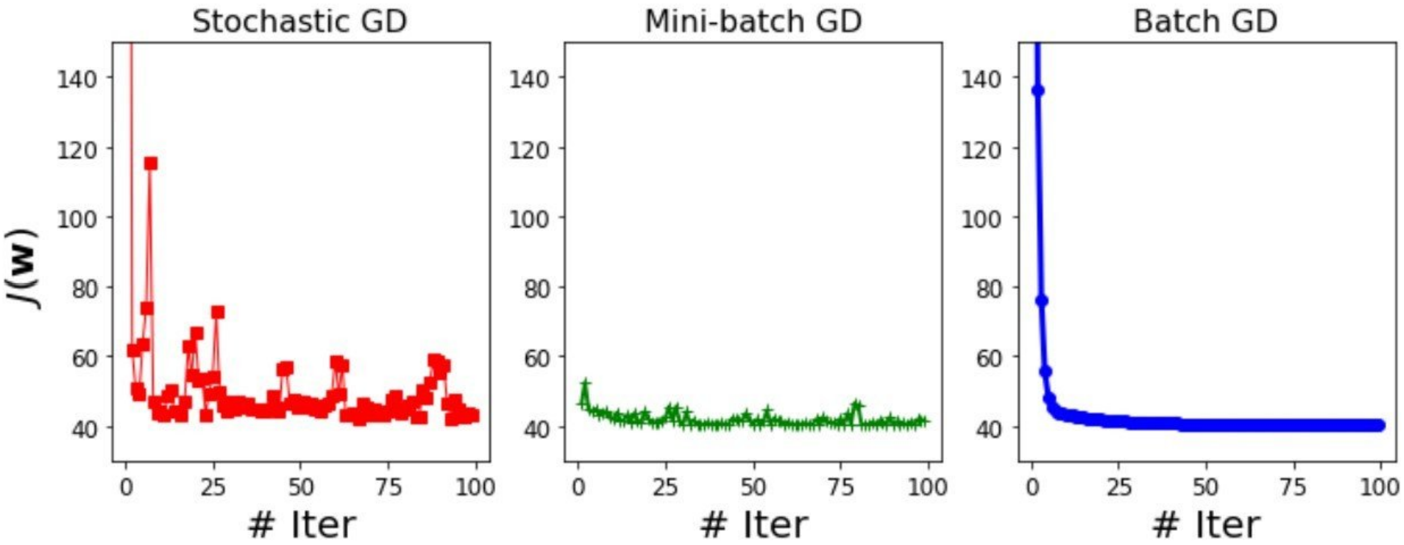


GD vs MBGD vs SGD: Learning curve comparison



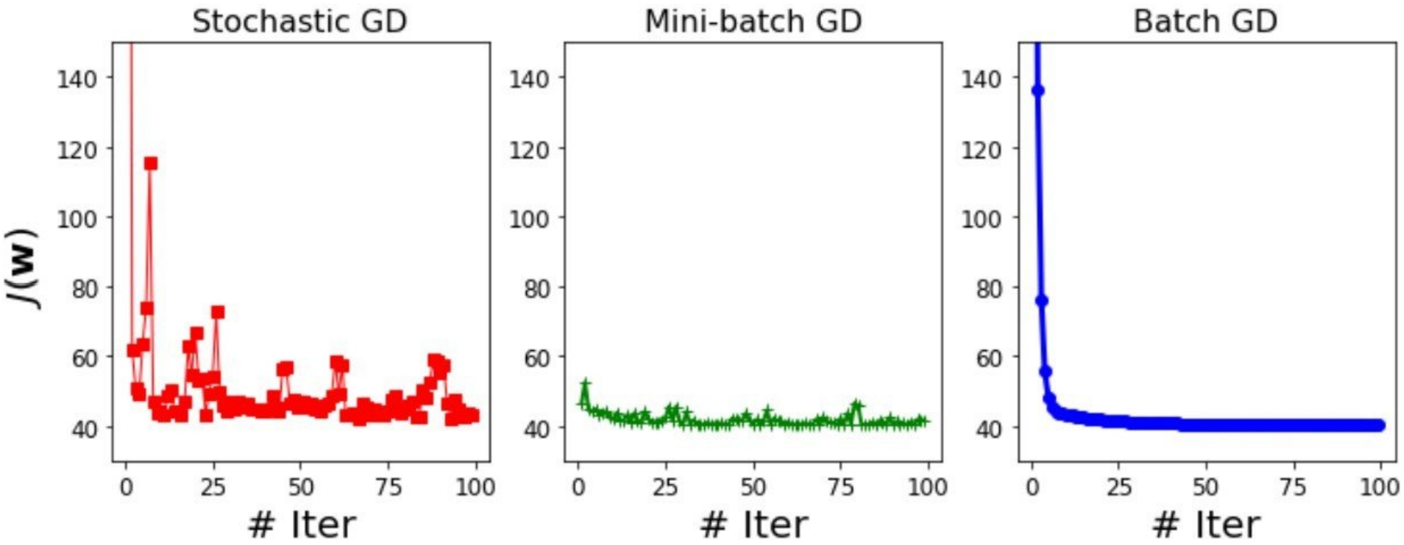
- Batch GD has the smoothest learning curve: the loss reduces continuously iterations after iterations.

GD vs MBGD vs SGD: Learning curve comparison



- The learning curves corresponding to mini-batch GD and SGD show ups and downs in loss.
- In some iteration, the loss reduces and then in the next one it goes up.
- Overall the losses are on downward trajectory.

GD vs MBGD vs SGD: Learning curve comparison



- As the batch size increases, the resulting learning curves become smoother for a given configuration of hyper-parameters like learning rate.

Evaluation

Evaluation

Uses **root-mean-squared-error (RMSE)** measure, which is derived from SSE.

$$\text{SSE} = \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$$

$$\begin{aligned}\text{RMSE} &= \sqrt{\frac{2}{n}\text{SSE}} \\ &= \sqrt{\frac{1}{n}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)} \\ &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2}\end{aligned}$$

Square-root is used to compare the error on the same unit and the same scale as that of the output label.

Division by n, the size of the dataset, enables us to compare model performance on datasets of difference sizes on the same footing.

Linear regression: Recap

(1) Data	Features and label that is real number.
(2) Model	Linear combination of features
(3) Loss function	Sum of squared error (SSE)
(4) Optimization procedure	(1) Normal eq. (2) GD/MBGD/SGD
(5) Evaluation	Root mean squared error (RMSE)

Linear regression: Recap

(1) Data	$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
(2) Model	$h_{\mathbf{w}} : \mathbf{y} = \mathbf{X}\mathbf{w}$
(3) Loss function	$J(\mathbf{w}) = \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{T}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$
(4) Optimization procedure	(1) Normal eq. (2) GD/MBGD/SGD
(5) Evaluation	$\mathrm{RMSE} = \sqrt{\frac{1}{n}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{T}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)}$