Week 1 – AI511-ML

Aniruddh Kishore Budhgavi
aniruddh.kishore@iiitb.ac.in

1 Sept 2021

1 Introduction

Welcome to AI511 Machine Learning 2021! In this document, we will revise some of the basic concepts of machine learning and also look at some of the terms associated with the subject. Do not worry if a lot of it feels foreign to you, or if you can't quite pin something down. A lot of this will become clear once you start training your own models.
After the definitions, we will have a look at your first machine learning model – Linear Regression – done in two ways. In one method, we use the closed-form solution of the Least Squares problem. In another method, we use Gradient Descent.

Part I
Basic concepts and terminology of Machine Learning

2 What is machine learning?

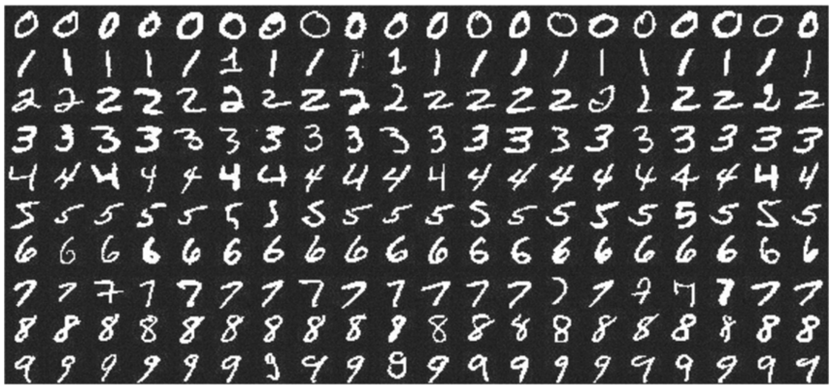

Figure 1: Examples from the MNIST handwritten digits dataset

Consider that you wish to recognize handwritten digits from 28-by-28 greyscale images. A traditional approach might be to write a program where there are a bunch of rules specified. For instance, a lone circle or oval could be a zero, a straight line with nothing else could be a one, a small circle with a downward line to the right could be nine... and so on.

The issue with this approach is that it is terribly complicated, and it's difficult to find the right rules. In general, such methods do not work well for this task.

The **Machine Learning** approach to this would be: Create a program which sees a large number of examples of handwritten digits with the labels given (that is, we tell it which ones are 0, which ones are 1, ...). Let the program figure out what rules it needs internally to distinguish between the different digits.

That's the core of Machine Learning. It's about creating systems which learn from data, instead of coding by hand the rules needed for decision-making.

3 The three classes of machine learning

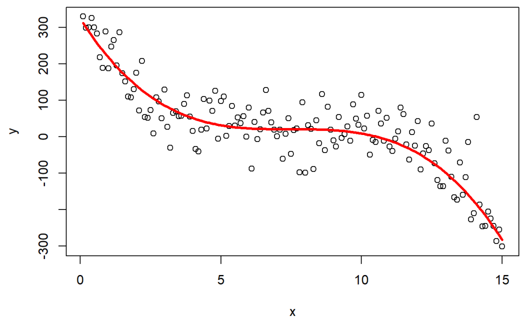

Figure 2: A supervised learning example – polynomial regression. The algorithm is given the input (x) and the label (y) for a bunch of data points, and learns to predict the label for any new point.

- **Supervised learning:** Here, it's about learning to *predict* something. Is that a cat photo or a dog? Is that a street light? What is the expected price of a 2BHK 1000 square foot house on MG Road? Does this chest X-ray contain a tumour?
  This is called supervised learning because there is someone who is telling the machine what the actual ground truth is for the training data.

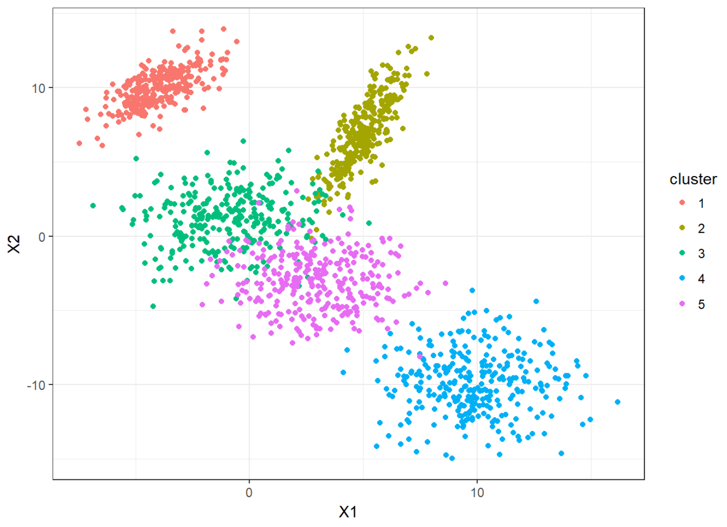

Figure 3: An example of unsupervised learning – k-means clustering. The algorithm is given the set of data points, and returns the clusters.

- **Unsupervised learning:** This relates to *finding patterns in unlabeled data*. Given these genes, can we hierarchically organize the genetic information? Is there some underlying structure to the data we have?
  This is called unsupervised learning because there is no ”teacher” to correct mistakes – no ground truth labels. All we have is unlabeled data.

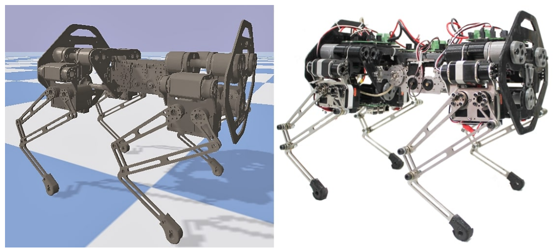

Figure 4: IISc's Stoch 2 robot. It is trained to walk and also to climb slopes, using reinforcement learning.

- **Reinforcement learning:** This branch of machine learning is about learning how to control an agent in order to maximize some reward. Popular examples are agents that learn how to play video games, agents that can self-drive cars, or the controller code in walking robots.

4 Some jargon:

1. **Model:** The model is what performs the specific machine learning task – it is what is trained, and what is later used during inference. In supervised learning, a model can be thought of as a function which takes in some input (the input data) and returns a predicted label. So if you wanted to classify handwritten digits, the model would accept a 28-by-28 greyscale image (suppose), and return a value from 0-9 corresponding to the handwritten digit.

2. **Training:** This is the process during which the model learns how to become good at the specified task.

3. **Model parameters vs hyperparameters:**
   Parameters are the variables internal to the model. These are modified during the training process by the learning algorithm. These are also known as the model weights. Typically, the developer does not manually modify these weights.
   Hyperparameters can be thought of as knobs the developer can adjust which influence the training process. Things like the number of iterations to run training for, the learning rate, the batch size are choices which YOU have to make and are levers which you directly have control over during the training process. The distinction between parameter and hyperparameter will become clear once you see examples of models.

4. **Regression vs classification:** Classification and regression are the two classes of supervised learning tasks.
   In classification, the goal is literally to classify stuff into discrete, finite, previously-known categories.Examples: Is this a cat or a dog? Does this person have COVID, or not? Which out of 0 to 9 is this handwritten digit?
   In regression, the goal is to predict a continuous variable.Examples: What is the relation between population density and incidence of COVID-19? How does the price of a house change depending on the number of bedrooms, the carpet area, and the number of car parks?

Part II
Your First Machine Learning Model: Linear Regression

5 Introduction: Straight-line Linear Regression

Let's take a simple example. You've just graduated from IIITB and have gotten a top offer from an MNC's India R&D branch. Rather than spending your generous joining bonus on frivolities, you've decided that it would be better to invest that money on a small one-bedroom apartment in the vicinity of your office – so that you can save on rent and on commuting (after all, Bangalore traffic is terrible).

Now, let's say you have visited 20 different flats for sale – some larger than others, and some more expensive. Let's say that you plot their square footage on the x-axis and their price on the y-axis.

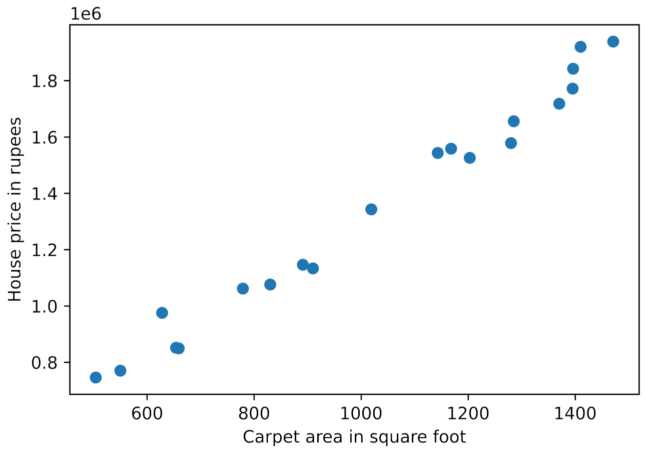

Figure 5: The data we have

There seems to be a linear relationship between the carpet area and the price. If you could model this relationship, you could glean more insight into the worth of a particular flat, whether it is over or underpriced, and how much you can negotiate.

We know that the data can be modeled by a straight line which kind of ”fits” the data points.

Specifically, if we consider the carpet areas of the houses to be $x_1, x_2, ..., x_n$ and the prices to be $y_1, y_2, ..., y_n$, we know that the relationship between any x_i and any y_i is roughly

$$y_i \approx w.x_i + b$$

This is what a linear regression model is. We assume that there is a linear relationship between the input variables x and the output label y, and now our objective is to find the appropriate values for w and b.

6 Capturing the ”Goodness” of a Model: The Loss Function

6.1 Intuition

Okay, so we know we have to find a straight line that fits the given data. Visually, we know that the below line *kind of* fits:

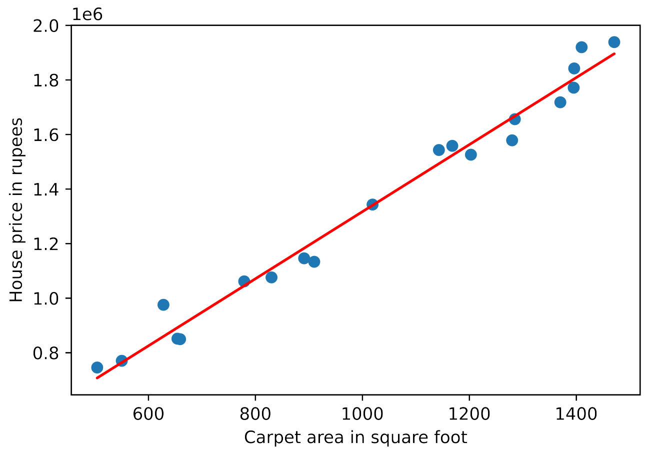

Figure 6: The data points with a line that ”fits” the data

But how do we perform this step computationally?

Look at this other model here:

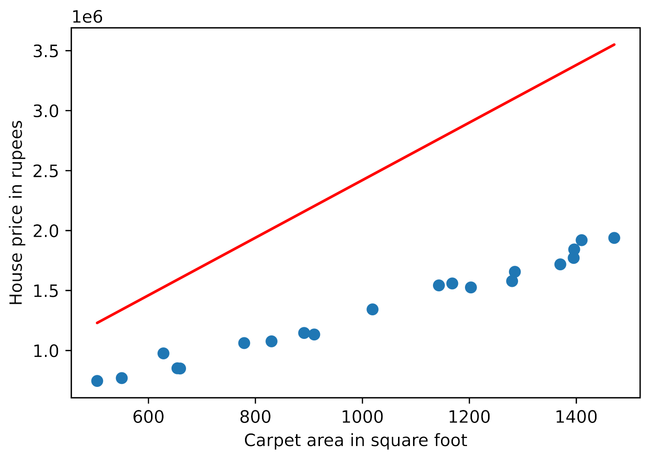

Figure 7: The data points with a "bad" line

We can see that the first model is *better* at explaining the data than the second one. But what does that look like, computationally? How do we capture the notion of one model being better than the other?

Intuitively, we feel that the first model is better than the second one because, *on average, the predicted label is closer to the true label for each input.* This gives us a hint as to how to progress: We say that the first model is better than the second one because it has a lower **mean squared error** on the given dataset.

6.2 The Mean Squared Error

Given a set of true labels (the answer) $y_1, y_2, ..., y_n$ and a matching set of predictions $\hat{y_1}, \hat{y_2}, ..., \hat{y_n}$, the mean squared error is defined as

$$J = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

You can intuitively see that a "bad" model has a large MSE and a "good" model has a small MSE on the training data.

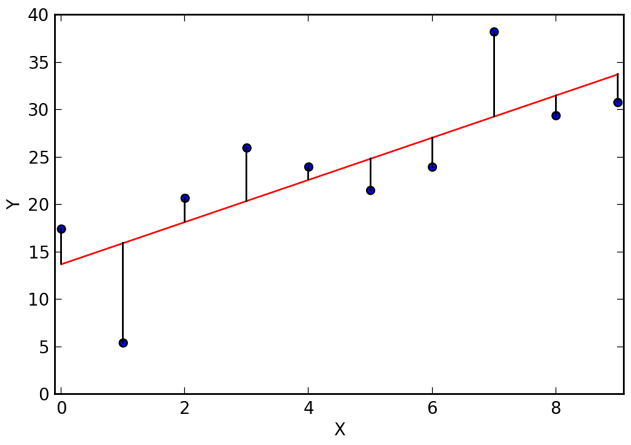

Figure 8: A visualization of the mean squared error. Note that the MSE measures the vertical distance between the line and the data points – NOT the *perpendicular* distance!!

Now, not only do we have a way to compare models, but if we can find a way to minimize the mean squared error with respect to the model parameters, we can find the best-fitting model for this dataset (among the class of straight-line models).

The mean squared error is one of many **loss functions** in machine learning. This idea, that you can capture the goodness of the model, and optimize it, is one that will keep showing up in machine learning, with linear regression, logistic regression and even in neural networks.

7 Two ways to optimize the cost

If we put in our model definition into the cost function, we get

$$J = \frac{1}{n}\sum_{i=1}^{n}(y_i - w.x_i - b)^2$$

This is a quadratic in w and b. This means that it has a single global minimum, and we can find this minimum using two ways – the first is the closed-form equation using the maxima-minima concept we learned in class 11, and the other is using an optimization method called gradient descent.

7.1 Closed-form solution: Least Squares

At the global minimum of this quadratic expression, we know that the partial derivatives of J with respect to w and b should be zero. Let us compute them.

To make things easier, let us consider that

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ ... \\ \hat{y_n} \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ ... & \\ x_n & 1 \end{bmatrix}$$

$$W = \begin{bmatrix} w \\ b \end{bmatrix}$$

$$\hat{Y} = XW$$

Putting all this into the MSE, we get:

$$\begin{aligned} J &= \frac{1}{n}(Y - \hat{Y})^T(Y - \hat{Y}) \\ &= \frac{1}{n}(Y - XW)^T(Y - XW) \\ &= \frac{1}{n}(Y^T - W^TX^T)(Y - XW) \\ &= \frac{1}{n}(Y^TY - W^TX^TY - Y^TXW + W^TX^TXW) \\ &= \frac{1}{n}(Y^TY - 2Y^TXW + W^TX^TXW) \end{aligned}$$

The last line occurs because $Y^TXW = W^TX^TY$. Remember that every term in J is a scalar!

Now, if we compute the gradient of J with respect to W and set that to zero, we will obtain the value of W which will minimize J.

$$\frac{\partial J}{\partial W} = \frac{1}{n}(-2X^TY + 2X^TXW) = 0$$

$$-2X^TY + 2X^TXW = 0$$
$$X^TXW = X^TY$$
$$\boxed{W = (X^TX)^{-1}X^TY}$$

There are two identities of matrix calculus we used above. The first is that

$$\frac{\partial Ax}{\partial x} = A^T$$

where A is a matrix and x is a vector. The second is that

$$\frac{\partial x^TAx}{\partial x} = 2Ax$$

where A is a $\underline{\text{symmetric}}$ matrix and x is a vector.

The above is known as the **Least-Squares Solution for Linear Regression**. This (and the gradient descent method) apply to higher dimensions as well – where every row of X is a vector, not a scalar, and where W has more dimensions than just 2. The only difference is that instead of fitting a straight line, we fit a **hyperplane**[1] in many dimensions.

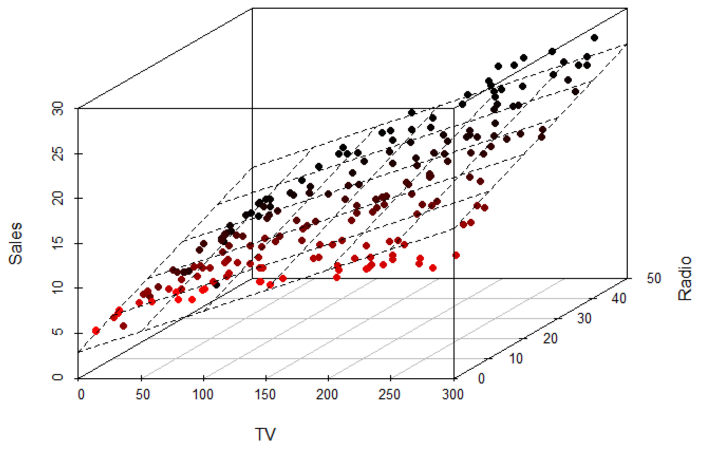

Figure 9: Linear regression in 3 dimensions – a 2-dimensional input, and a 1-dimensional (scalar) output

One interesting thing to think about is – when would this process of least squares fail? In particular, when does the inversion process fail?

[1]Strictly speaking, a hyperplane is a subspace, whereas here we just mean a $n-1$ dimensional flat object in an n dimensional vector space. Like a 2D plane in a 3D space.

7.2 Gradient descent

The idea of optimizing the loss function in order to obtain a model's weights is powerful, and is repeatedly used in machine learning. However, it is not possible to always find a neat closed-form solution for every combination of loss function and model.

In those cases (which, in fact, is most cases) we use numerical optimization methods. The most elementary one from which most machine learning optimization methods are derived is called gradient descent.

The idea behind gradient descent stems from a single fact. If we have the surface of a multivariate function (many inputs, one output), then

The gradient at a point of the function with respect to the inputs is in the direction of steepest increase of the function.

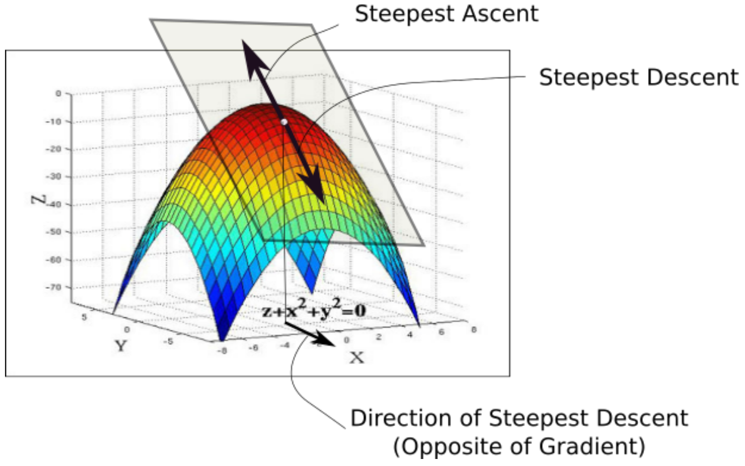

Figure 10: Visualization of gradient and direction of steepest ascent and descent

Therefore, if we wish to reach a minimum of the cost function, we could move in the direction of steepest descent, or, the negative of the gradient with respect to the weights. Of course, we cannot simply compute the gradient at a single point and forever keep moving in the same direction. The reason is simple: As soon as we move from that initial point, the gradient vector changes!

So we follow a cycle: Compute the gradient, move in the negative direction, recompute, and so on.

Another question is – *how much* should we move? When we are far from the minimum, we wish to make a large step so that we can use as few steps as possible – but when we are close to the minimum, we wish to make small steps so that we don't "overshoot" the minimum point. It's a bit like golf – when you are far from the target, you make large, powerful swings, but when you're close, you make short, precise putts.

Another piece of intuition is that, in general, we can assume that the gradient is large when we are far from the minimum, but is smaller when we are close to the absolute minimum. [2] Therefore, we could make the step size proportional to the magnitude of the gradient.

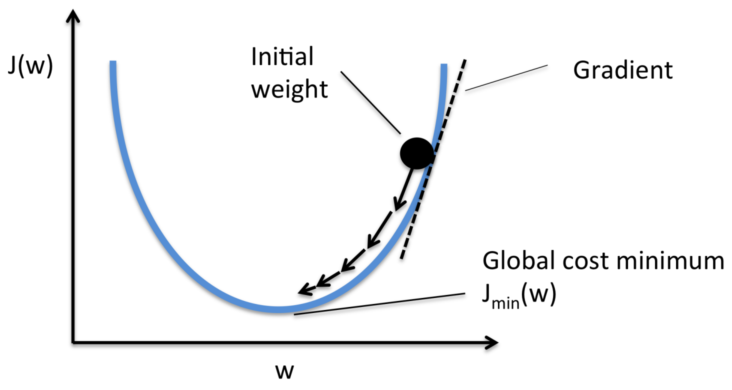

Figure 11: Notice how the magnitude of slope is high when you are far from the minimum but low when you are closer. Now expand this idea to multiple dimensions.

The algorithm (where M is the model, W, X, Y and $\hat{Y}$ are as before:

Algorithm 1 Gradient Descent

```
$W \leftarrow$ random
Costs $\leftarrow \phi$
**for** $i = 1$ to $n_i$ **do**
    $\hat{Y} \leftarrow M(W, X)$
    $C \leftarrow J(Y, \hat{Y})$
    $W \leftarrow W - \alpha \nabla_W C$
    Append $C$ to Costs
**end for**
```

n_i is the number of iterations (or **epochs**) and α is the **learning rate**. Both

[2]In general, this is always true only for convex cost function–model family pairs. In case of neural networks, this is only mostly true. In case of complex architectures, you may find that this doesn't always hold true. In those cases, we use other tricks, like gradient clipping. But for now, you can safely ignore this footnote.

are examples of **hyperparameters**, while W is a **parameter**. The learning rate tells us how big the step size should be, while the number of iterations tells us how many times we should execute gradient descent.

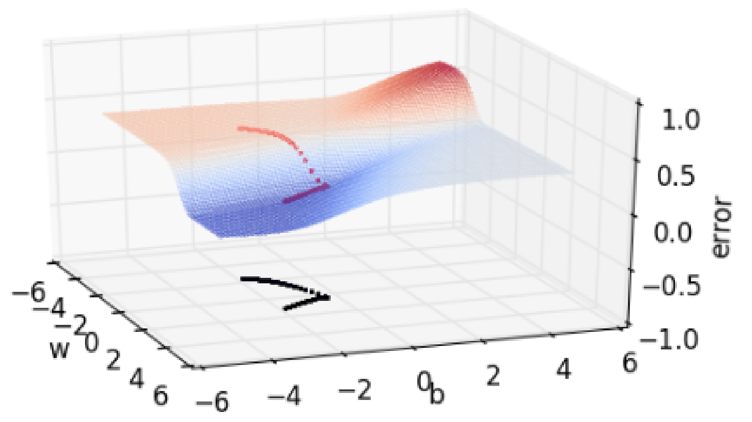

Figure 12: Another analogy for gradient descent is that it is like a ball rolling to the lowest point from the side of a hill.

If you're observant, you may notice that we don't actually need to compute C, the current cost, to compute $\nabla_W C$, the gradient of C with respect to W. However, it is pretty much idiomatic to compute the cost this way and to store it. The reason is that logging and then plotting the cost is one of the easiest ways of getting information to debug models. You typically want the cost to decrease throughout the learning process. When it reaches a plateau, that's when the number of iterations is enough. You should then test the model and see if it's working well (more on that later). If the cost suddenly explodes, then it means something is going wrong in the training. Your first instinct should be to try a smaller learning rate (anywhere from 10^0 to 10^{-6}, typically), and if that doesn't work, take a closer look at what is going on.

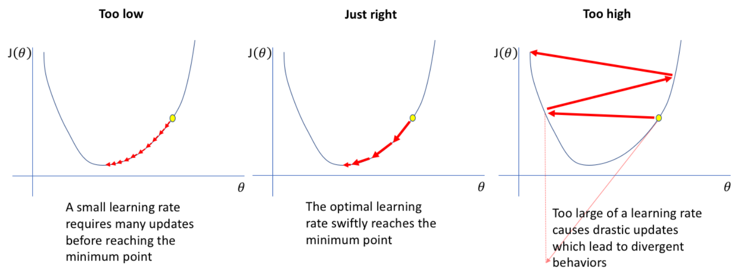
Too low
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
Just right
J(θ)
θ
The optimal learning rate swiftly reaches the minimum point
Too high
J(θ)
θ
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 13: The effect of learning rate on gradient descent.

That's it for now. Hopefully, this helps. Please give feedback on this document. We'll be implementing linear regression using Numpy and using Scikit-Learn in this week's lab.