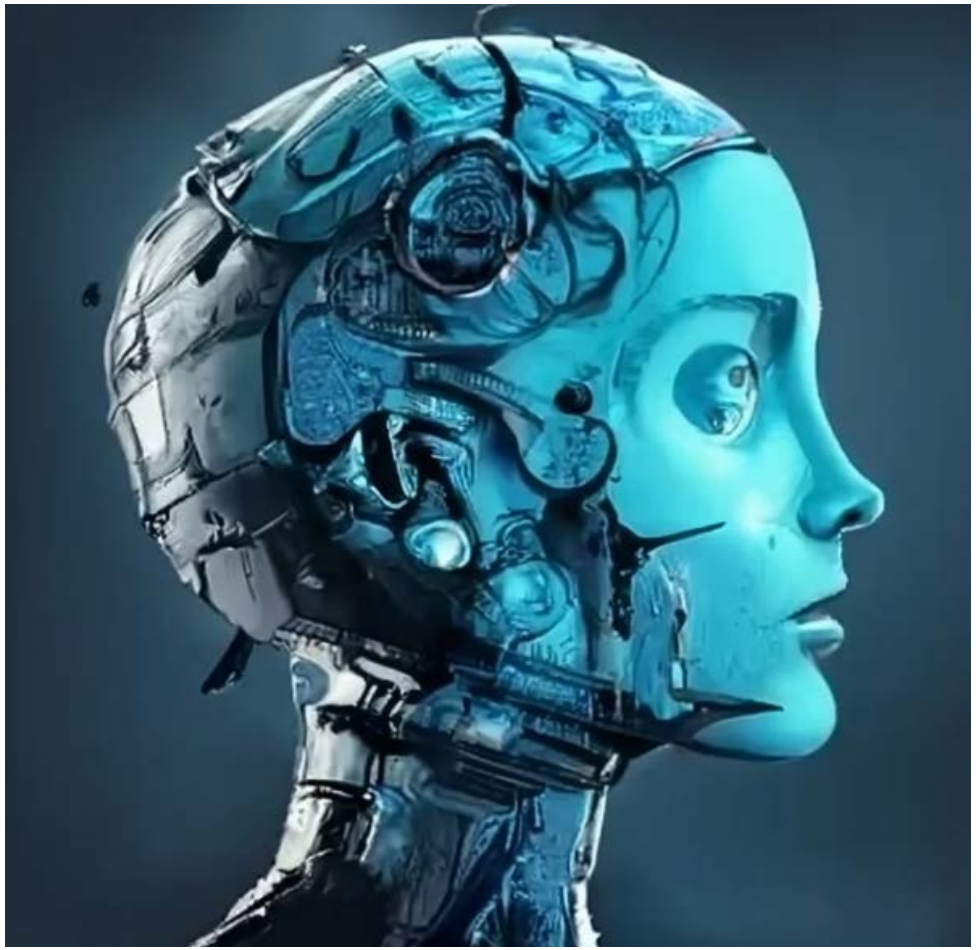
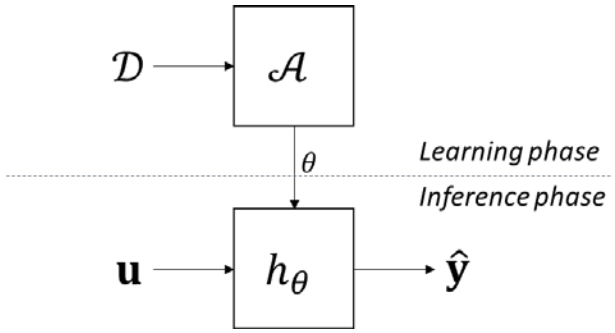
Machine Intelligence

Linear Regression Using Gradient Descent

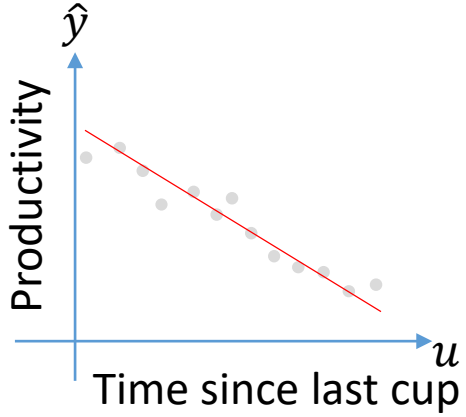
Dr. Cory Merkel



Linear Regression in 2D



- A.K.A. fitting a straight line to data



- $\hat{y} = h_\theta(u) = mu + b, \theta = \{m, b\}$
- $\mathcal{D} = \left\{\left(u^{(p)}, y^{(p)}\right)\right\}$
  - We will call the number of samples in our dataset m
- What is $\mathcal{A}$???

Linear Regression Using Linear Algebra

- Recall that we can perform linear regression on data with N independent variables and M dependent variables using the pseudoinverse:

$$\mathbf{W} = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{Y}$$

Where $\mathbf{U}$ is the $n \times N$ matrix of n independent variable observations, $\mathbf{Y}$ is the $n \times M$ matrix of n dependent variable observations, and $\mathbf{W}$ is the $N \times M$ matrix of coefficients. This equation gives us a least squares estimate to the solution of

$$\mathbf{Y} = \mathbf{UW}$$

- What if $(\mathbf{U}^T\mathbf{U})^{-1}$ isn't invertible or is too large to be inverted in a reasonable amount of time?

Gradient Descent

- We found our linear algebra solution by minimizing a loss function:

$$\mathbf{W}^* = \arg\min_{\theta} \mathcal{L}(\mathcal{D}; \theta)$$

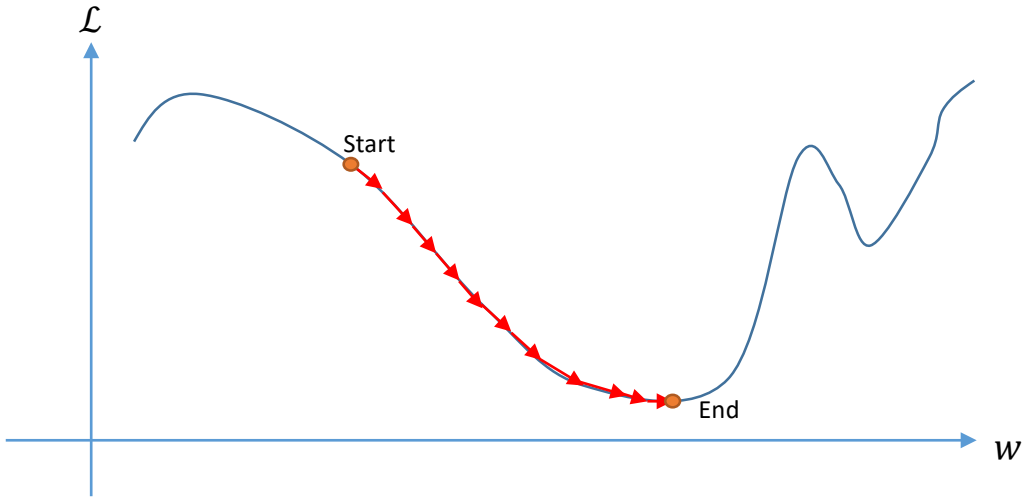
- Is there another way that we can find the minimum without using matrix inverses?

- If you are hiking, what is the fastest way to get to the bottom of a valley?
  - Head in the direction of the steepest descent
  - But make sure to not go too fast, otherwise you might get hurt!

Gradient Descent

Stop condition can be:
- Specified number of iterations
- Specified value of $\mathcal{L}$
- Specified change in loss between iterations

- Gradient descent is a general optimization procedure that finds places where the gradient of a function is equal to zero



Initialize θ (e.g. to random values)
while **stop_condition** = False:

$$\theta_i := \theta_i - \alpha \nabla_{\theta_i} \mathcal{L}(\mathcal{D}; \theta) \; \forall \; i$$

Step size or *learning rate*

Gradient operator

- Iteratively update your independent variable(s) in the *opposite* direction of the gradient until you reach a point where the gradient is close to zero or you reach a pre-specified number of iterations
  - Note that the gradient is a vector that points in the *ascending* direction of the function, which is why we need a minus sign when we update our variables

Linear Regression with Gradient Descent

- Now, let's solve our linear regression problem using gradient descent
- Remember our loss function:

$$\mathcal{L} = \frac{1}{2n}\sum_{p=1}^{n}\left(y^{(p)} - \hat{y}^{(p)}\right)^2 = \frac{1}{2n}\sum_{p=1}^{n}\left(y^{(p)} - \left[w_N u_N^{(p)} + w_{N-1}u_{N-1}^{(p)} + \cdots + w_1 u_1^{(p)} + w_o\right]\right)^2$$

  - We add a 2 in the denominator so it will cancel out once we take the derivative

$$\nabla_{w_i}\mathcal{L}(\mathcal{D};\mathbf{w}) \equiv \frac{\partial\mathcal{L}(\mathcal{D};\mathbf{w})}{\partial w_i} = -\frac{1}{n}\sum_{p=1}^{n}\left(y^{(p)} - \hat{y}^{(p)}\right)u_i^{(p)} = \frac{1}{n}\sum_{p=1}^{n}\left(\hat{y}^{(p)} - y^{(p)}\right)u_i^{(p)}$$

- Now, we just need to take small steps in each w according to the gradient:

$$w_i \coloneqq w_i - \alpha\frac{1}{n}\sum_{p=1}^{n}\left(\hat{y}^{(p)} - y^{(p)}\right)u_i^{(p)}$$

Example Gradient Descent Over 2 Dimensions

- Recall our life expectancy vs. smoking habits example.

$$\text{life expectency} = m \times \#\frac{cigarettes}{day} + b$$

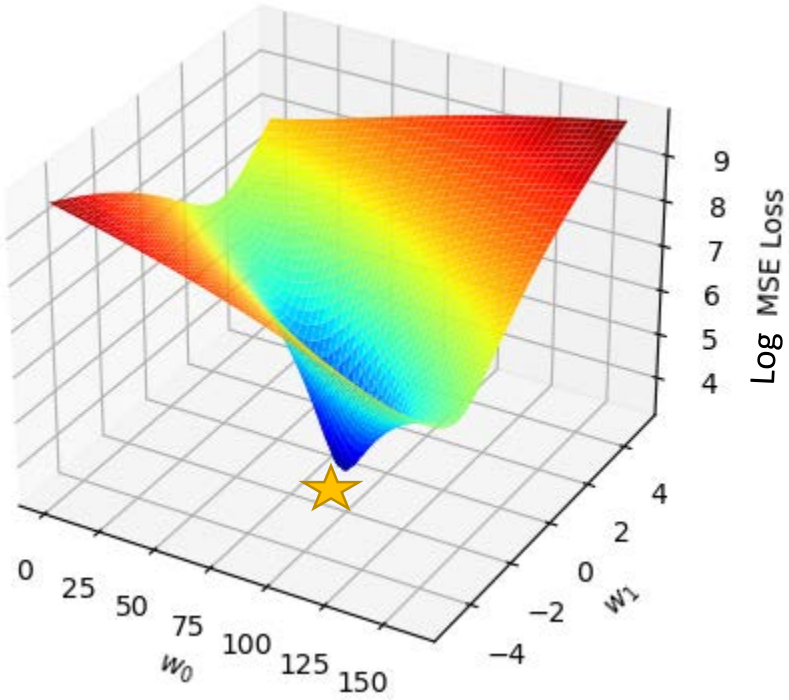
Or

$$\hat{y} = w_1 u + w_0$$

1. Initialize w_1 and w_o to random values
2. Iteratively update w_1 and w_0:

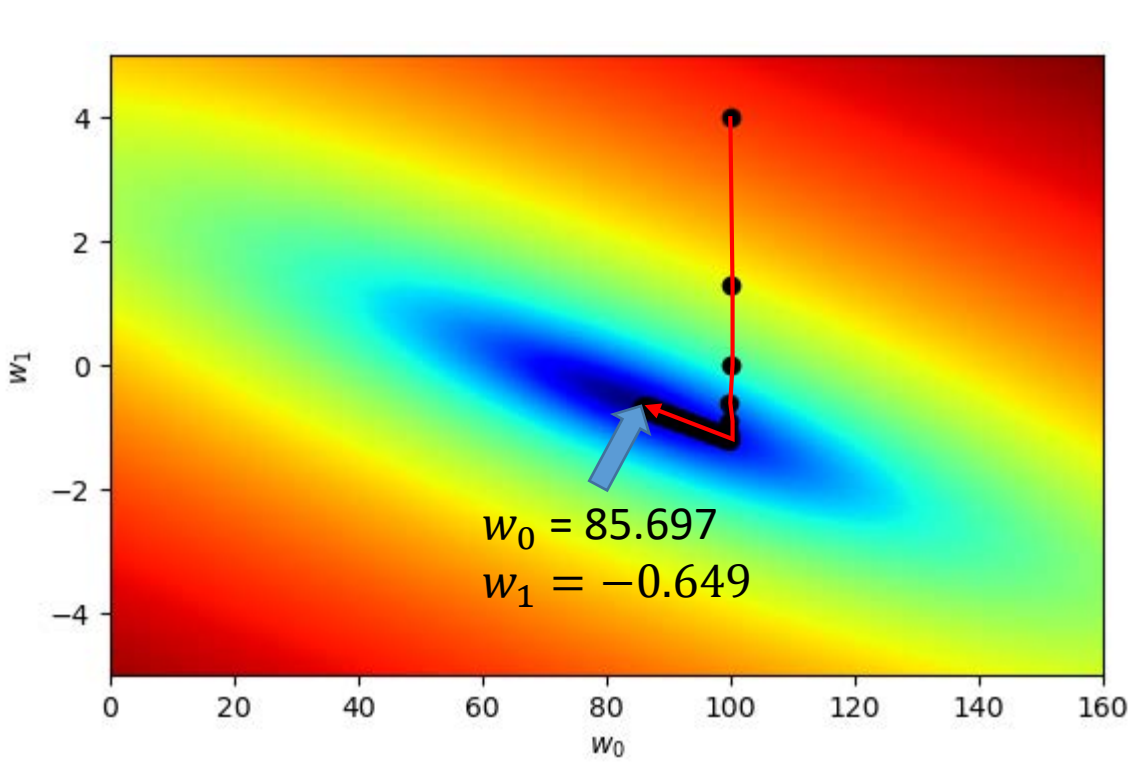
$$w_1 \coloneqq w_1 - \alpha \frac{1}{n} \sum_{p=1}^{n} \left(\hat{y}^{(p)} - y^{(p)}\right) u^{(p)}$$

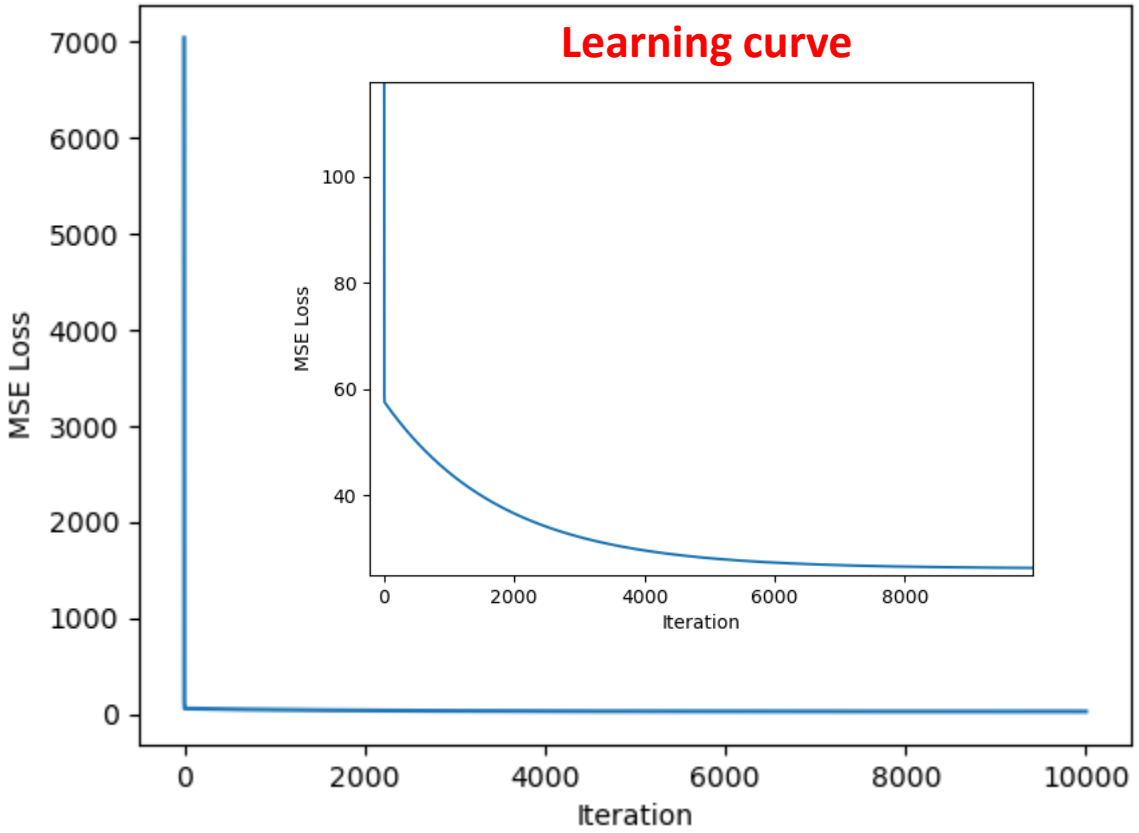
$$w_0 \coloneqq w_0 - \alpha \frac{1}{n} \sum_{p=1}^{n} \left(\hat{y}^{(p)} - y^{(p)}\right)$$



Example Gradient Descent Over 2 Dimensions

- Starting at a random point ($w_0 = 100, w_1 = 4$):

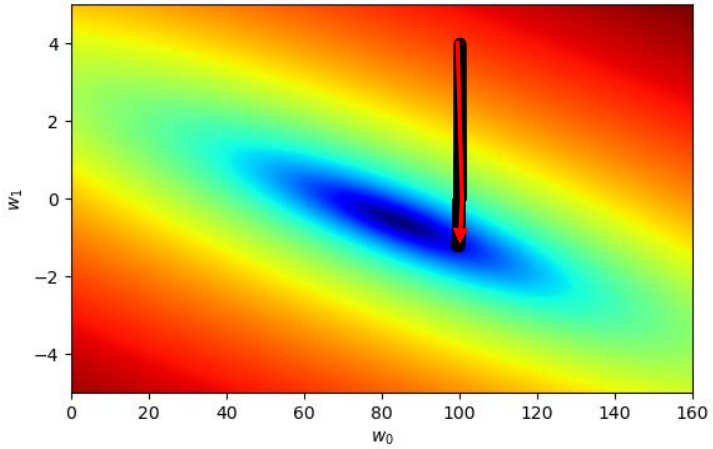


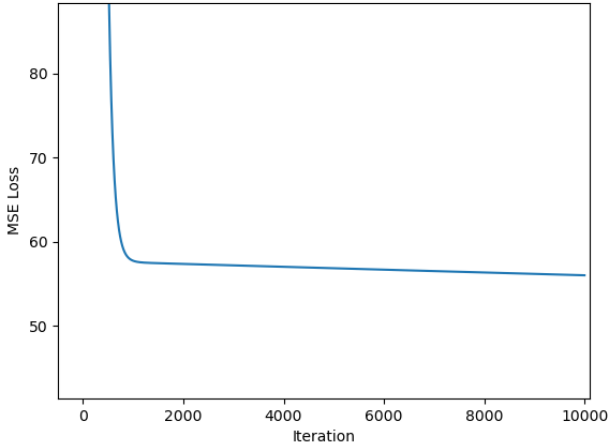


- ***Gradient is perpendicular to contour lines, but notice in this case the range of the vertical and horizontal axes are very different, which is why the trajectory does not look like it's perpendicular.***

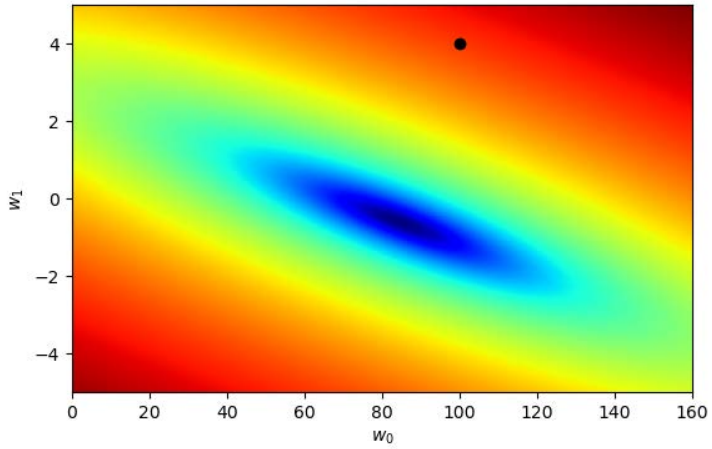
How Do We Chose α?

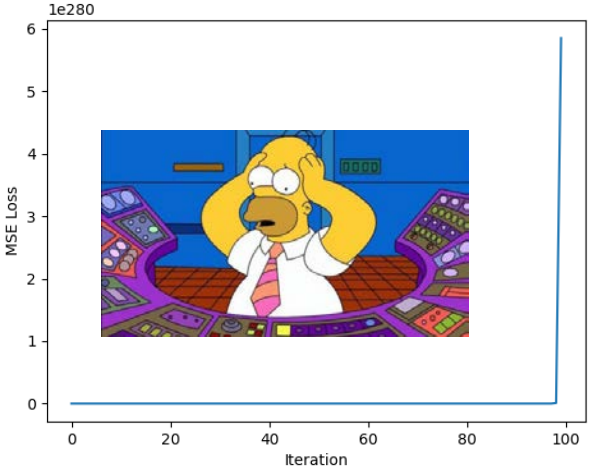
- If α is too small:
  - Convergence will take a very long time
  - May not reach an optimal result





- If α is too big
  - Loss may oscillate or explode





- α can be chosen empirically by picking an initial value $0.001 \leq \alpha \leq 0.1$ and then increasing/decreasing based on learning curve

Basing Weight Updates on Fewer Observations

- The gradient of our loss is

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{1}{n} \sum_{p=1}^{n} \left(\hat{y}^{(p)} - y^{(p)} \right) u_i^{(p)}$$

- This means we need to evaluate our model n times (once for each observation) before we update the parameters
- Can we estimate the gradient using fewer observations?

$$\frac{\partial \mathcal{L}}{\partial w_i} \approx \frac{1}{n'} \sum_{p=1}^{n' < n} \left(\hat{y}^{(p)} - y^{(p)} \right) u_i^{(p)}$$

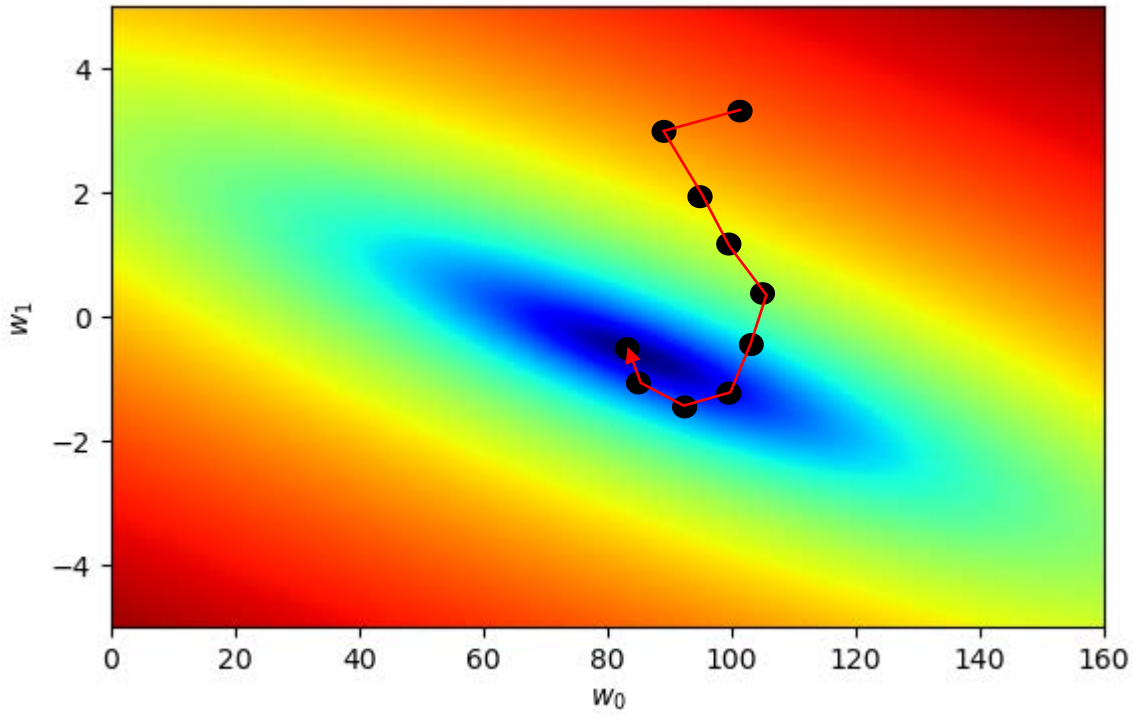
Basing Weight Updates on Fewer Observations

- When $n' = 1$, we call this stochastic gradient:

while stop_condition = False:

for p = 1 to n:

$$w_i := w_i - \alpha(\hat{y}^p - y^p)u_i^p$$



Noisy gradient estimate means we will not directly follow path of steepest descent

- But, now we have n parameter updates per iteration instead of just one!

Basing Weight Updates on Fewer Observations

- We can also break the dataset into “mini batches”
  - Better gradient estimate since we are using more data
  - Might be faster than full batch training because we have more parameter updates per iteration

while stop_condition = False:

  for batch = 1 to num_batches:

$$w_i := w_i - \frac{1}{n'} \sum_{p \in batch} (\hat{y}^p - y^p)\, u_i^p$$

- n' is the batch size

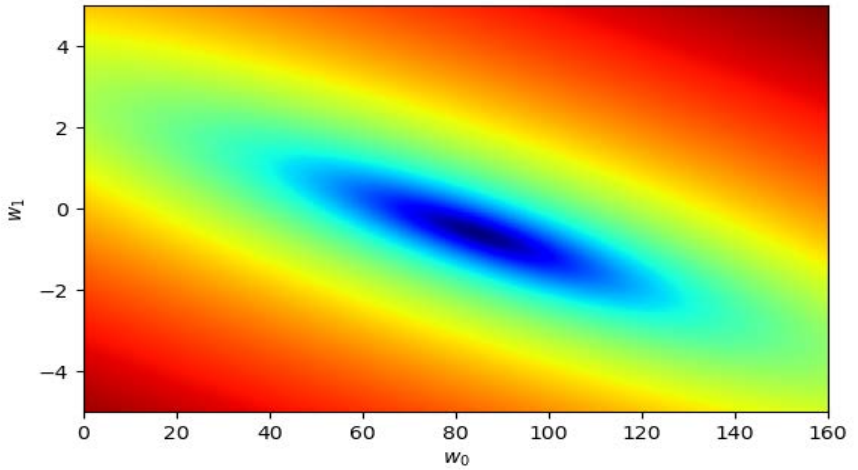
Feature Scaling

- Recall our smoking dataset:
  - We have 2 features
    - ❑Constant column of ones for the bias
    - ❑Cigarettes smoked per day

$$\mathbf{U} = \begin{bmatrix} & \textbf{Cigarettes smoked per day} \\ 1 & 4.00 \\ 1 & 4.98 \\ 1 & 4.00 \\ 1 & 7.93 \\ 1 & 11.03 \\ 1 & 14.06 \\ 1 & 17.17 \\ 1 & 18.80 \\ 1 & 22.97 \\ 1 & 22.97 \\ 1 & 25.02 \\ 1 & 25.92 \\ 1 & 28.94 \\ 1 & 34.91 \\ 1 & 48.00 \end{bmatrix}$$

- $\nabla_{\mathbf{w}}\mathcal{L} = \left[\frac{1}{n}\sum_{p\in batch}\left(\hat{y}^{(p)} - y^{(p)}\right), \quad \frac{1}{n}\sum_{p\in batch}\left(\hat{y}^{(p)} - y^{(p)}\right)u^{(p)}\right]$

- Note that the gradient will have a much larger component in the direction of w_1
  - Leads to the “skinny” valley in the loss landscape:
  - How can we make the valley more circular?



Feature Scaling

- Lots of options, but the basic idea is to try to normalize the range, mean, standard deviation, and other statistics so they are the same among all features
- Some examples:
  - Normalize the range of each feature between 0 and 1:

$$u_i^{(p)} := \frac{u_i^{(p)} - \min_p \left(u_i^{(p)}\right)}{\max_p \left(u_i^{(p)}\right)}$$

  - Make the mean = 0 and standard deviation = 1 for each feature:

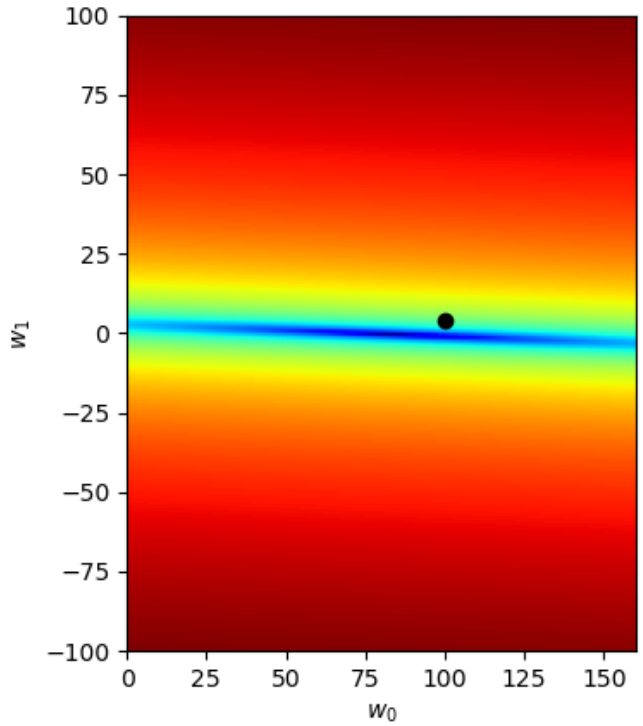
$$u_i^{(p)} := \frac{u_i^{(p)} - \mu_{u_i}}{\sigma_{u_i}}$$

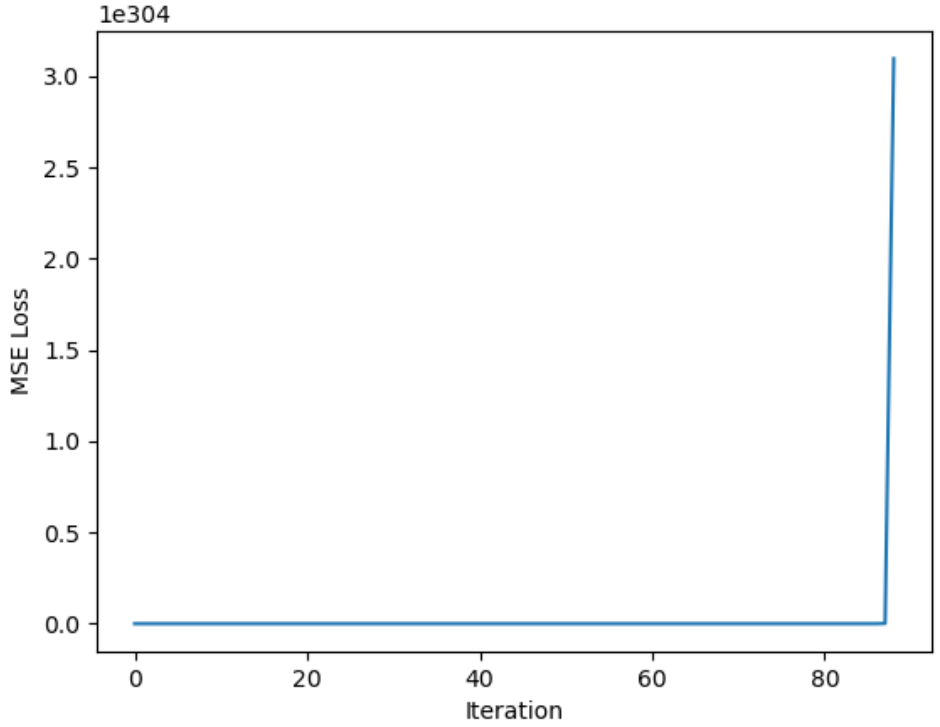
Mean value of feature i

Standard deviation of feature i

Without Feature Scaling

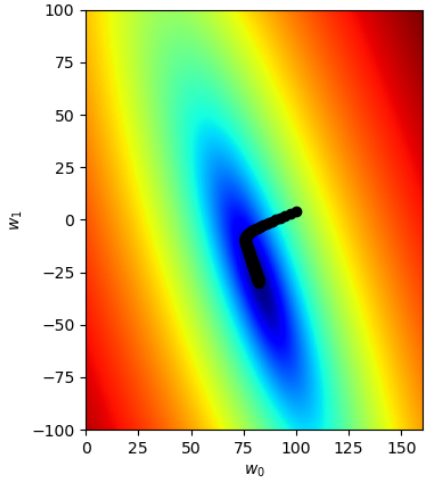
- Gradient descent doesn't converge
  - Would need to reduce learning rate significantly→very long time to complete algorithm

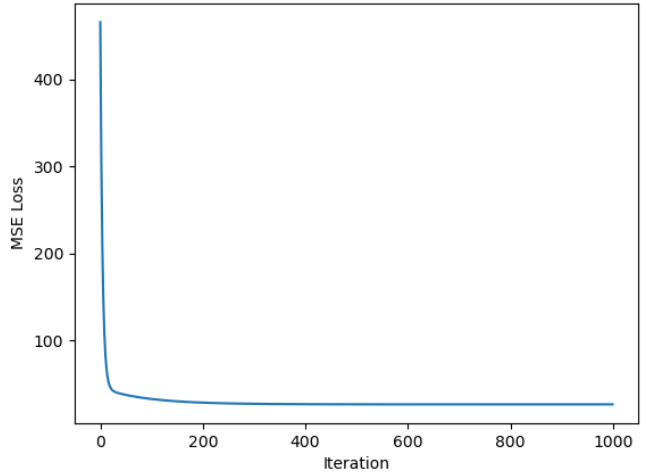




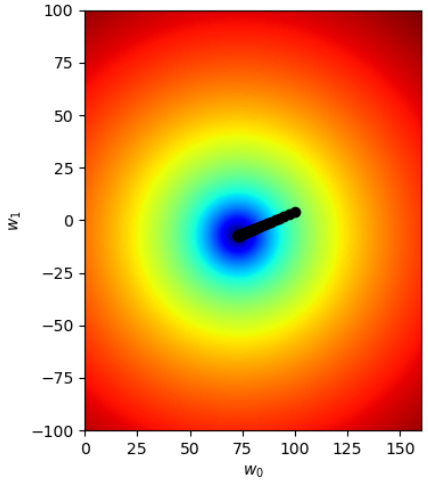
Feature Normalization Allows Faster Learning Rates → Faster Convergence

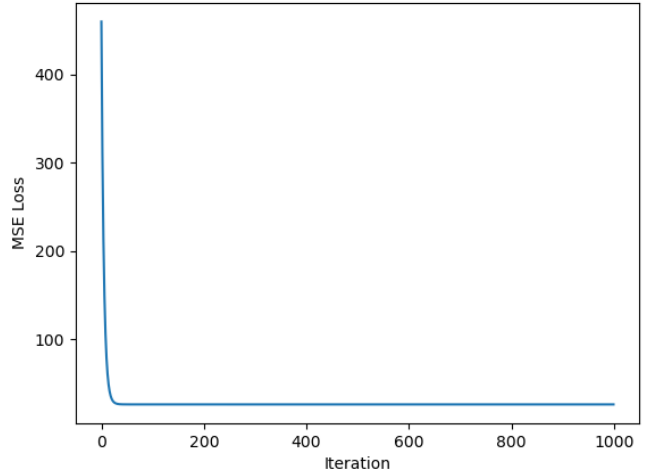
- Scale u to be between 0 and 1:





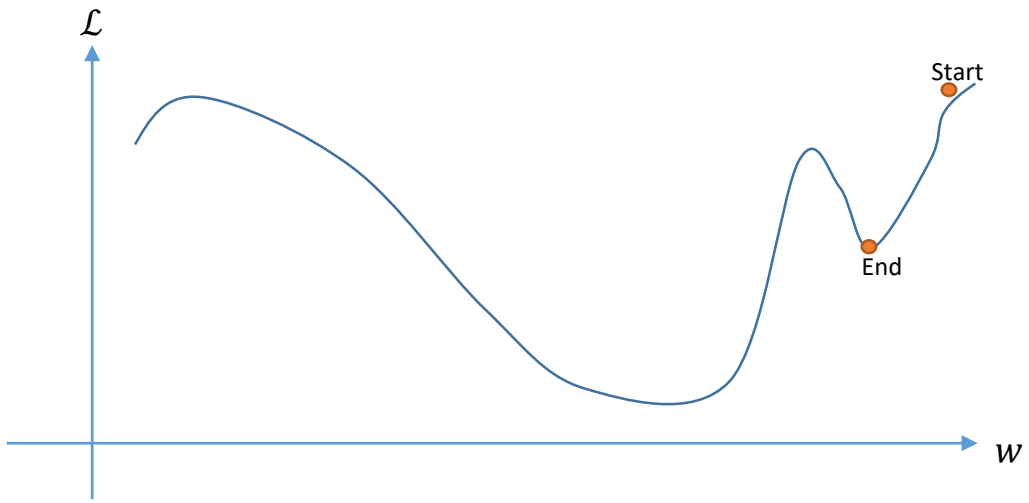
- Scale u to have mean 0 and std. 1:





Function Convexity

- Gradient descent could get stuck in “local optima” if the loss function is not *convex*



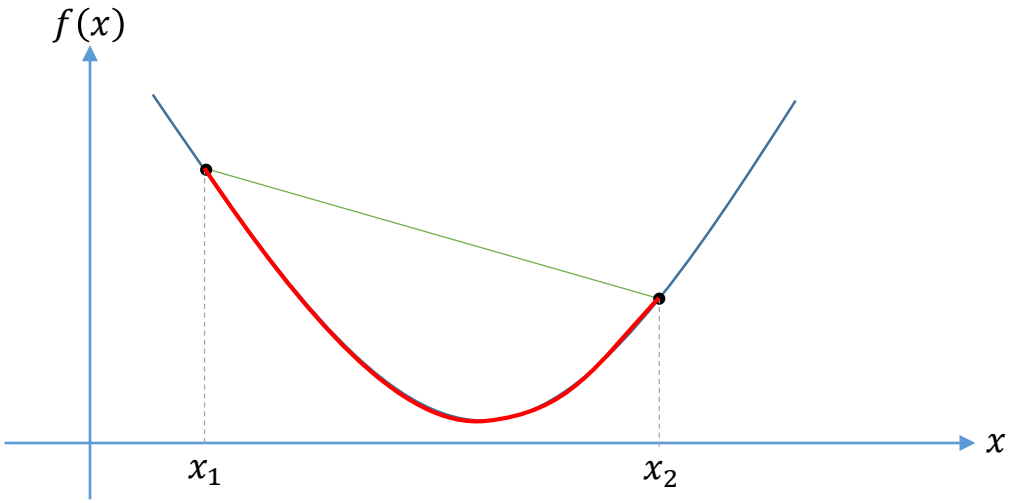
- A function $f(x)$ is convex iff, for all (x_1, x_2) and $0 \leq \lambda \leq 1$:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) - (1-\lambda) f(x_2)$$

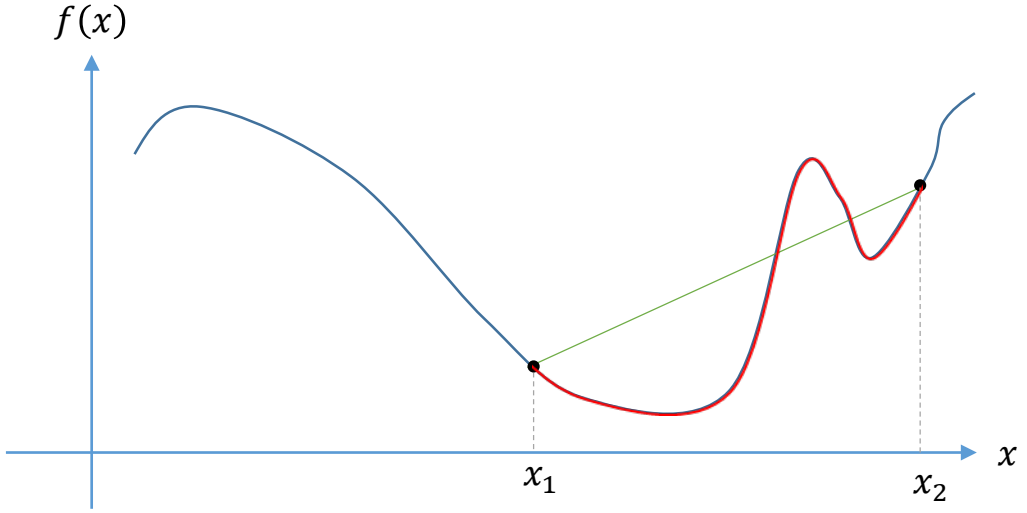
- If a function’s 2nd derivative is positive everywhere, then it is convex

Function Convexity

- Convexity: $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) - (1-\lambda) f(x_2)$



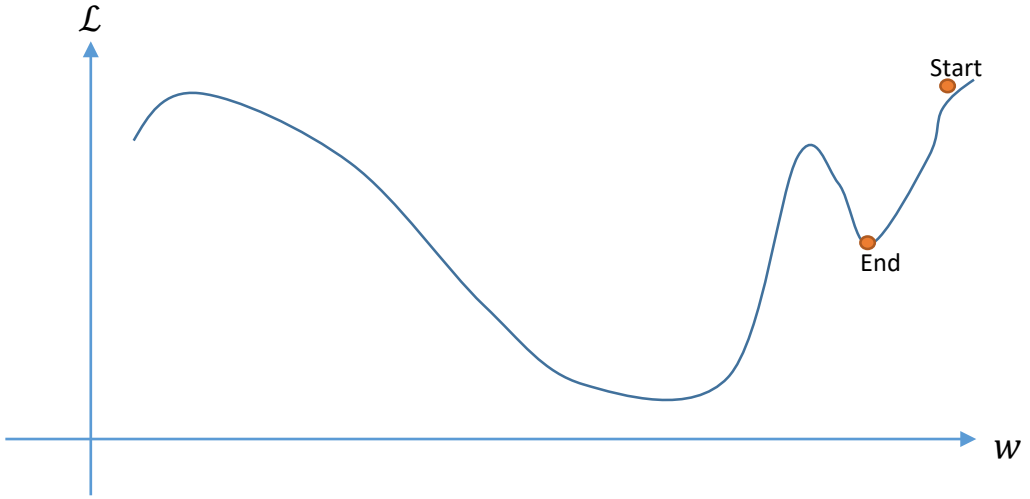
Convex: Green line segment lies above red function segment for any two values of x

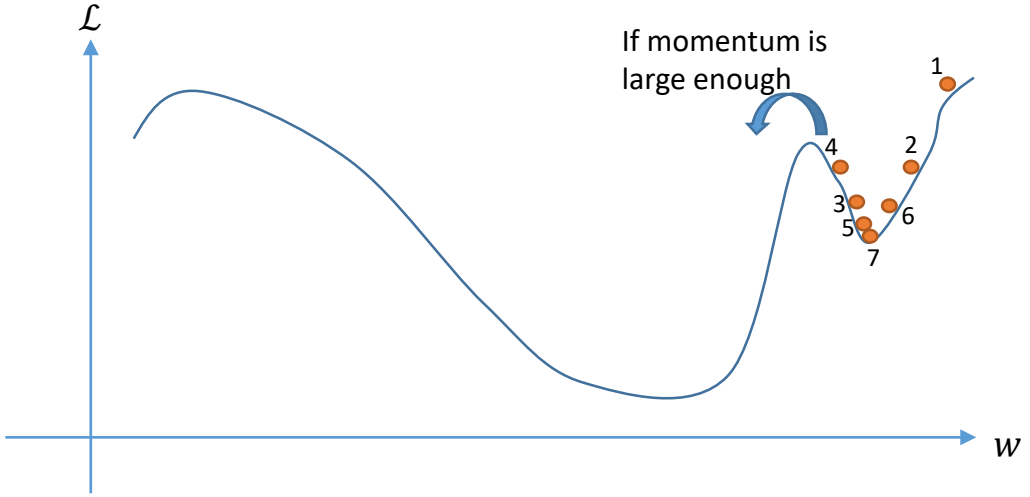


Not Convex: Parts of the function lie above the green line segment.

What if We Have a non-Convex Loss Function?

- Most of the time, we will be dealing with complicated loss functions, which may not be convex
- How can we avoid local minima?
- One technique is to use *momentum*
  - Think of a ball rolling down one side of a valley. It won't stop instantly at the bottom, but will roll a little bit up the other side of the valley and oscillate a couple times before settling:
  - If the momentum is large enough, we can escape local minima





Gradient Descent with Momentum

- We introduce a new constant μ called the momentum and use it to adjust how quickly we change the parameter updates:

$$w_i := w_i - \alpha \Delta w_i$$

$$\Delta w_i := \mu \Delta w_i + \frac{\partial \mathcal{L}}{\partial w_i}$$

- When $\mu = 0$, this is just the normal gradient descent
- When $\mu > 0$, we remember the previous update values and use them to determine the new one.