Google Cloud

Optimization

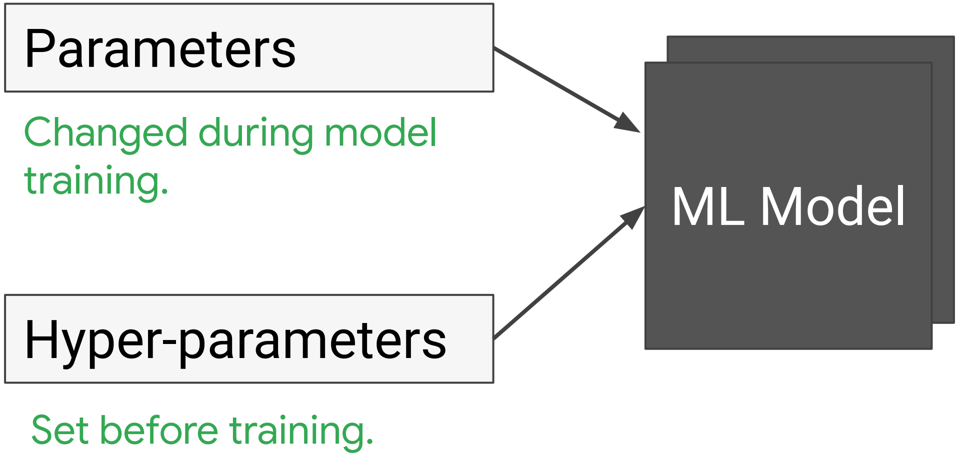
Agenda

Defining ML Models

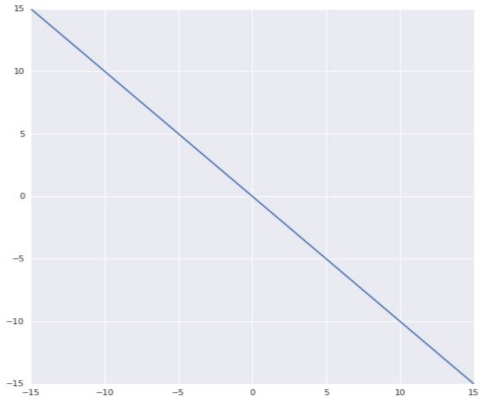
Introducing Loss Functions

TensorFlow Playground

ML models are mathematical functions with parameters and hyper-parameters



Linear models have two types of parameters: Bias and weight



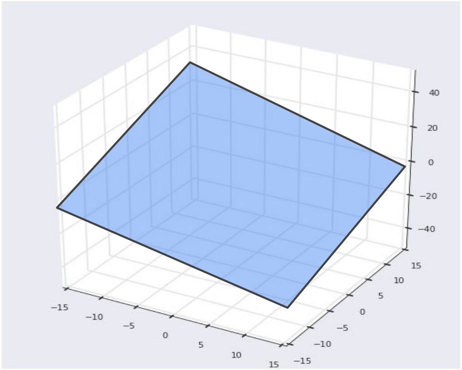
Linear

Output | Bias Term | Input | Weight

$$y = \boxed{b} + x \times \boxed{m}$$

$$y = \boxed{b} + X \times \boxed{w}$$

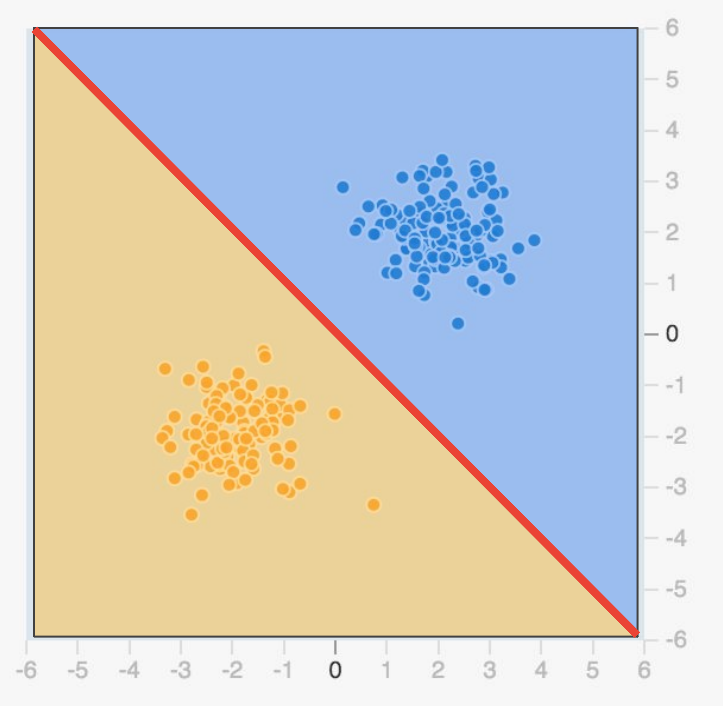
Model Parameters



Hyperplane

How can linear models classify data?

Classification explained graphically.



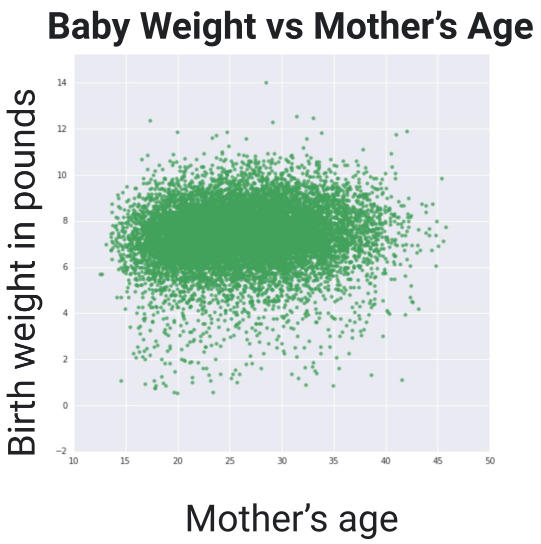
How do we predict a baby's health *before* they are born?

Which of these could be a *feature* in your model?

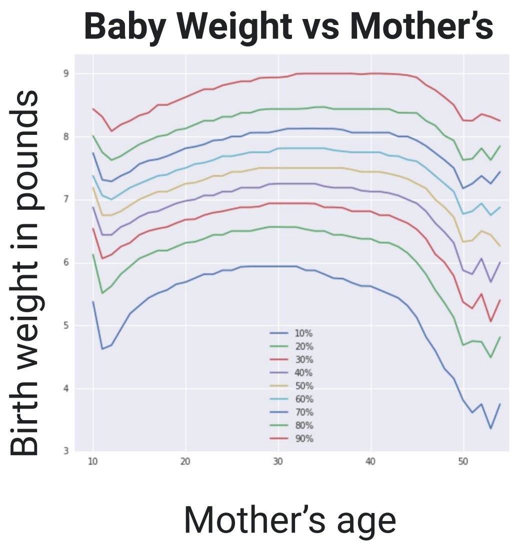
A. Mother's Age
B. Birth Time
C. Baby Weight

Which could be a *label?*

Exploring the data visually



Scatterplots are made from samples of large datasets rather than from the whole dataset.



Graph representing groups of data, specifically, quantiles.

Equation for a linear model tying mother's age and baby weight

The slope of the line is given by w1.

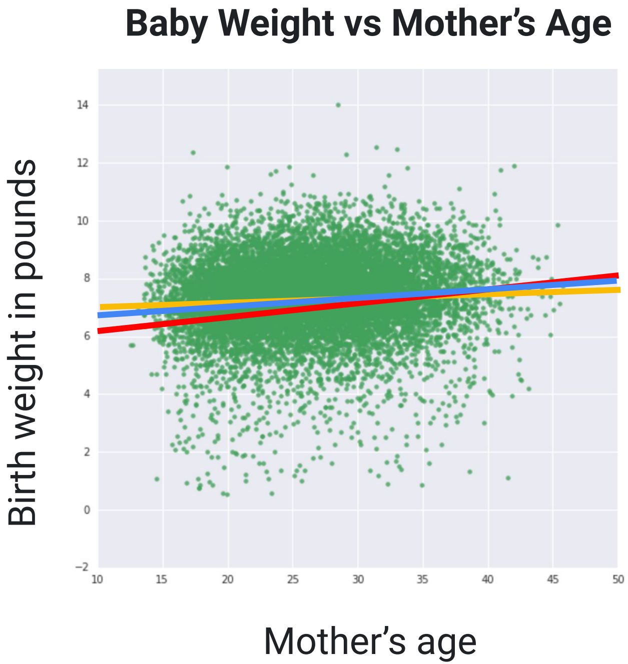
$$y = w_1 x_1 + b$$

- x_1 is the **feature** (e.g. mother's age)
- w_1 is the **weight** for x_1

`Line: y= .02x + 6.83`

`Line: y= .03x + 6.49`

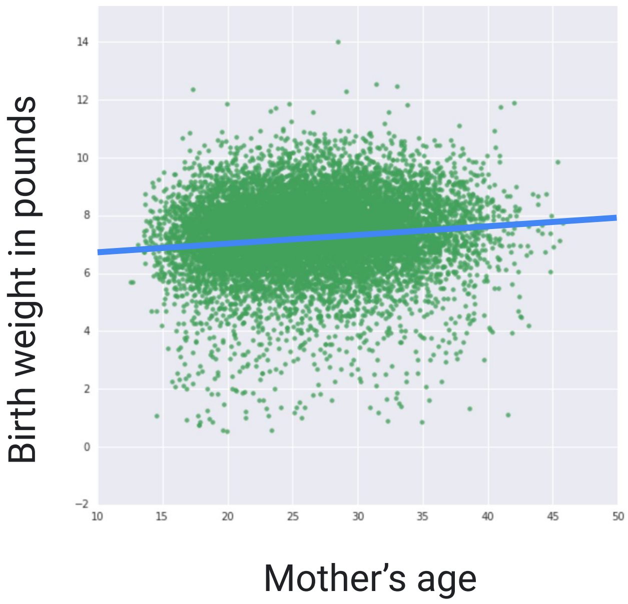
`Line: y= .01x + 7.14`



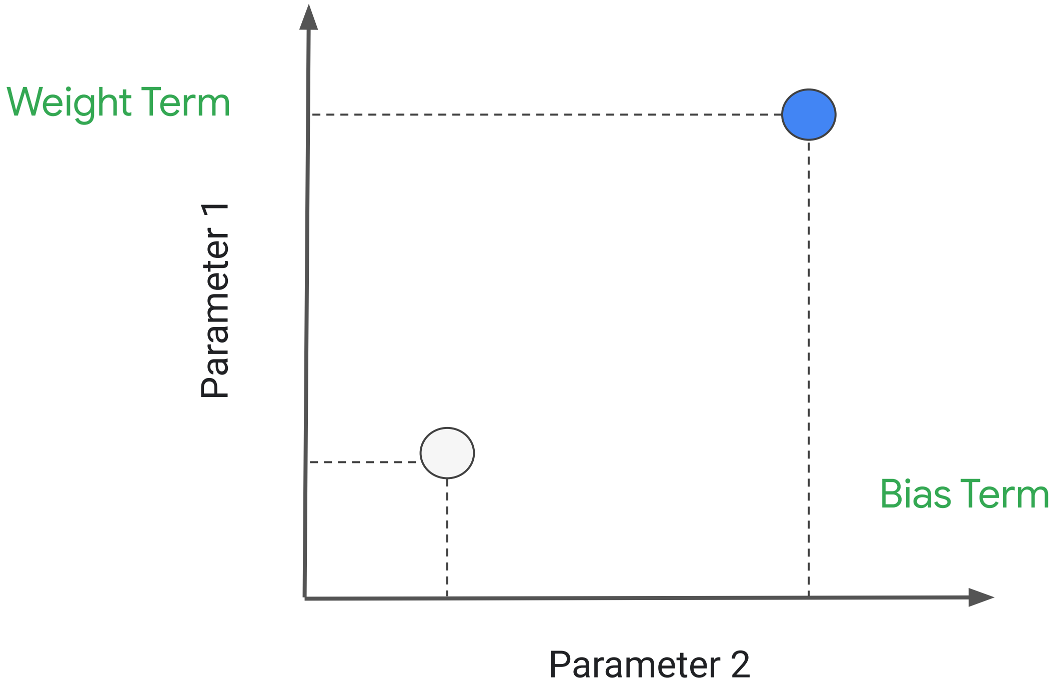
Can’t we just solve the equation using all the data?

When an analytical solution is no longer an option, you use gradient descent.

Baby Weight vs Mother’s Age



Searching in parameter-space



Agenda

Defining ML Models

Introducing Loss Functions

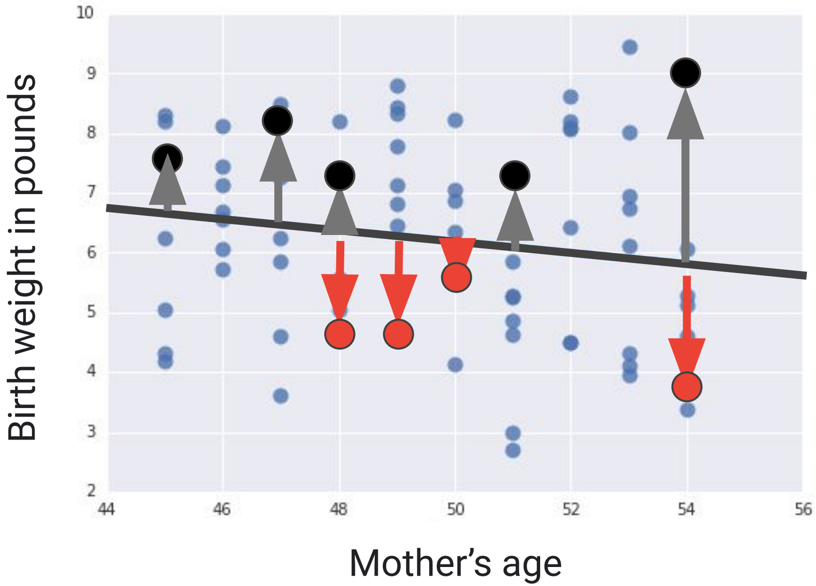
TensorFlow Playground

Compose a loss function by calculating errors

Error = actual (true) - predicted value
Compute the errors:

```
+0.70
+1.10
+0.65
-1.20
-1.15
+1.10
+3.09
-2.10
```

Each error makes sense. How about all the errors added together?



One loss function metric is Root Mean Squared Error (RMSE)

1 Get the errors for the training examples.

2 Compute the squares of the error values.

3 Compute the mean of the squared error values.

+0.70	0.49	2.51
+1.10	1.21	
+0.65	0.42	
-1.20	1.44	
-1.15	1.32	
+1.10	1.21	
+3.09	9.55	
-2.10	4.41	

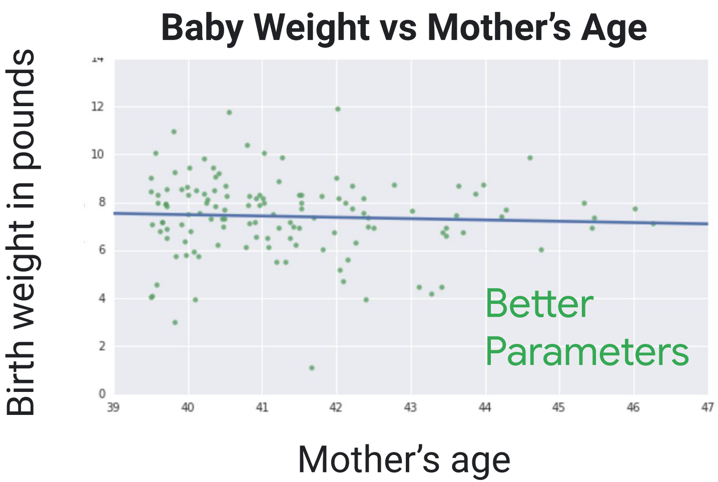
$$\sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2}$$

$\hat{Y}_i$ predicted value

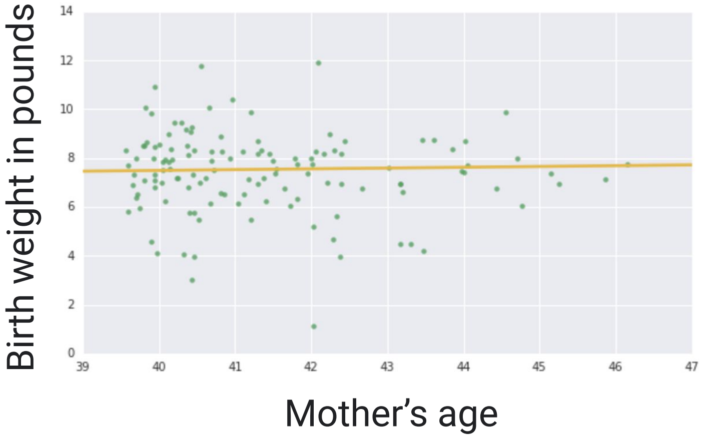
Y_i labeled value

4 Take a square root of the mean. **1.58**

Lower RMSE indicates a better performing model



RMSE=.145

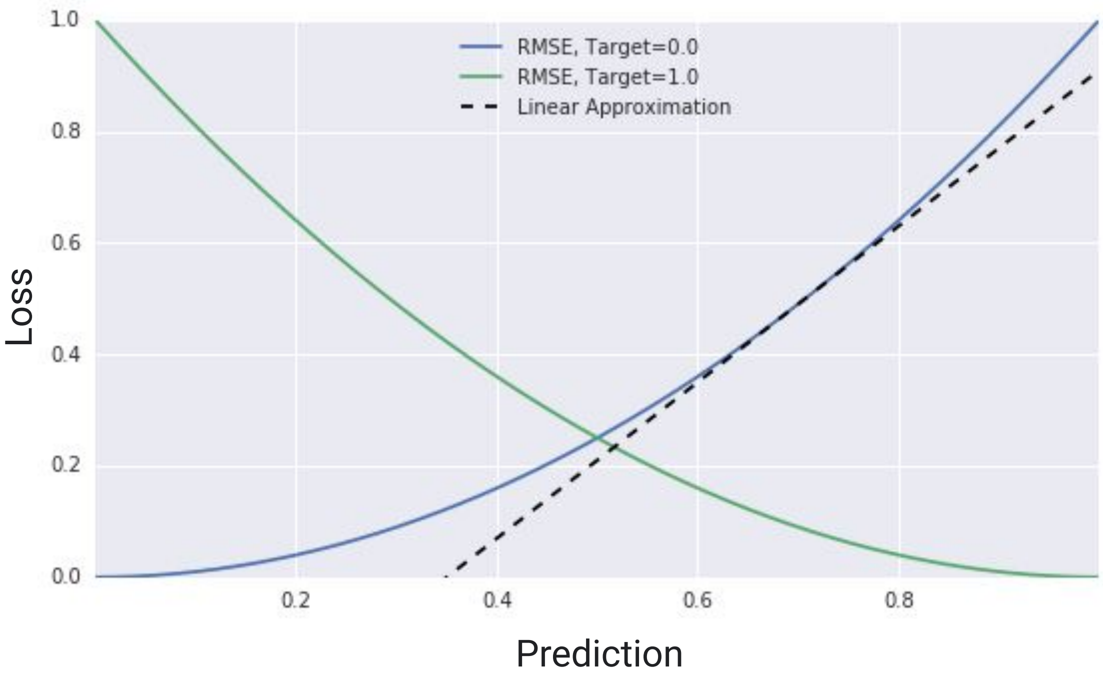


RMSE=.149

Need a way to find the best values for weight and bias.

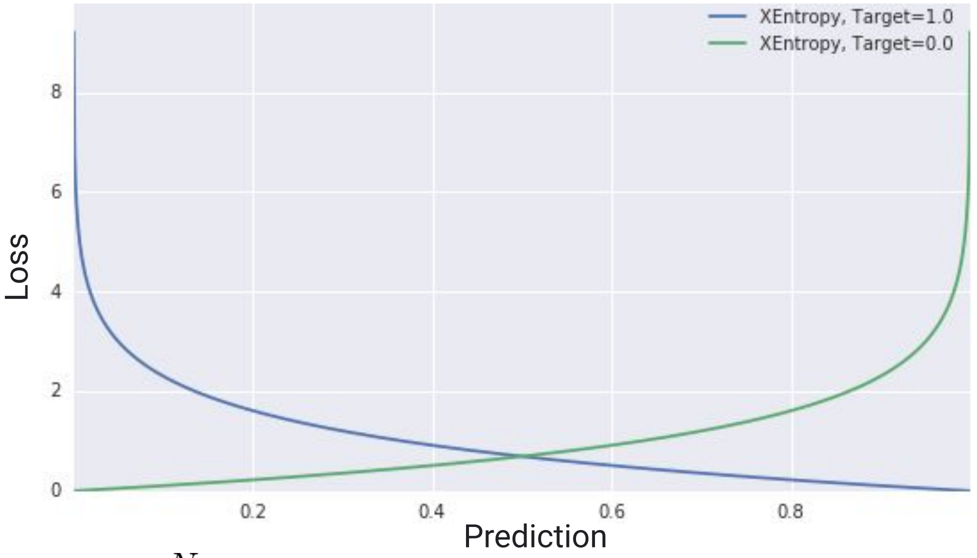
Problem: RMSE doesn't work as well for classification

RMSE doesn't penalize bad classifications appropriately.



Problem: RMSE doesn't work as well for classification

Bad classifications are penalized appropriately.



$$\frac{-1}{N} \times \sum_{1}^{N} y_i \times log(\hat{y}_i) + (1 - y_i) \times log(1 - \hat{y}_i)$$

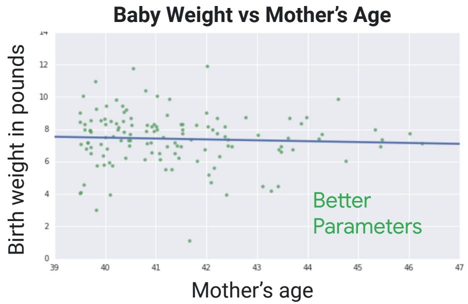
Computing cross-entropy loss

$$\frac{-1}{N} \times \sum_{1}^{N} \overbrace{y_i \times log(\hat{y}_i)}^{\text{Positive term}} + \overbrace{(1-y_i) \times log(1-\hat{y}_i)}^{\text{Negative term}}$$

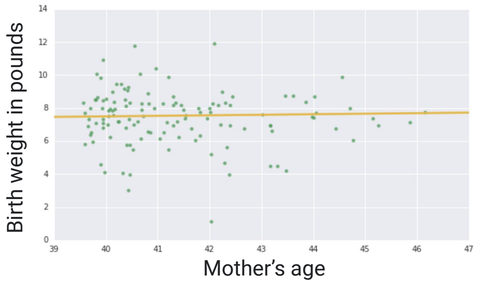
X	Y_i	$\hat{Y}_i$
	1	.7
	0	.2

$$\left(\begin{array}{c} (1.0*log(.7)) + \cancel{(1-1.0)*log(1-.7)} \\ + \\ \cancel{(0.0*log(.2))} + (1-0.0)*log(1-.2)) \end{array} \right) *(-½)=.13$$

From loss functions to gradient descent



RMSE=.145



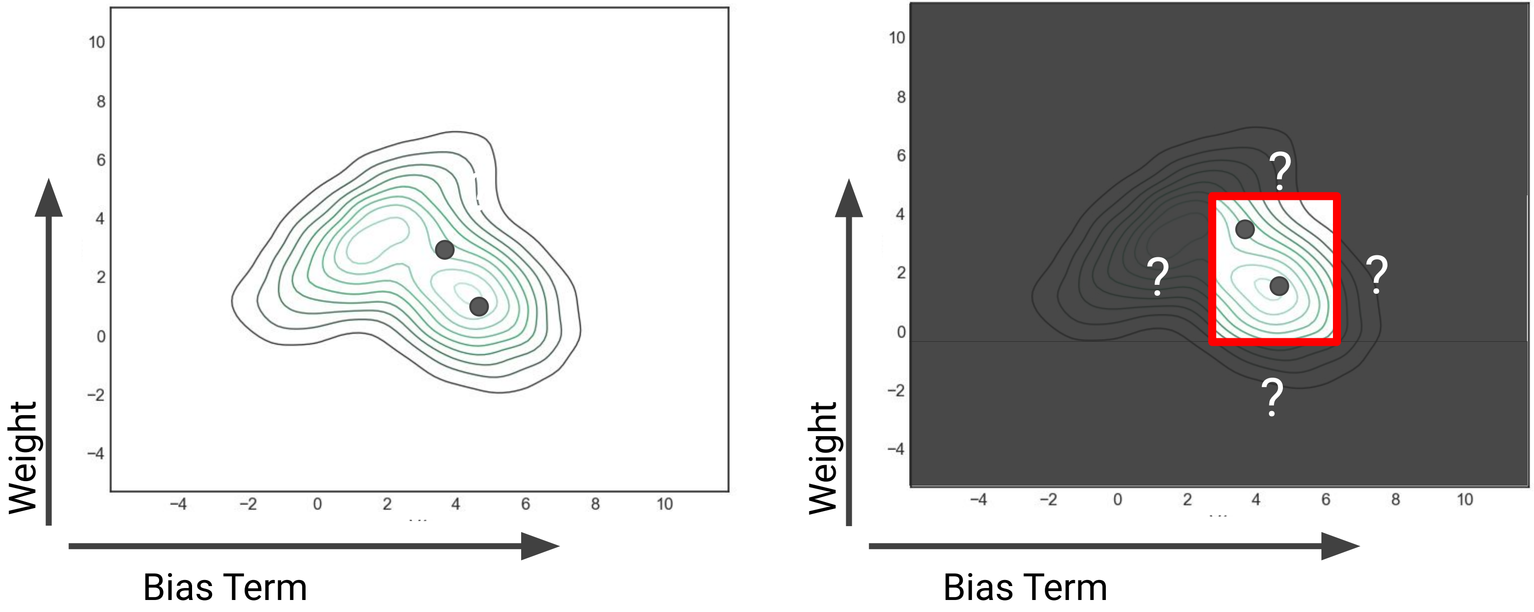
RMSE=.149

X	Y_i	$\hat{Y}_i$
	1	.7
	0	.2

$$\frac{-1}{N} \times \sum_{1}^{N} \overbrace{y_i \times log(\hat{y}_i)}^{\text{Positive term}} + \overbrace{(1 - y_i) \times log(1 - \hat{y}_i)}^{\text{Negative term}}$$

$$\left(\begin{array}{c} (1.0*log(.7) + \cancel{(1-1.0)*log(1-.7)} \\ + \\ (\cancel{0.0*log(.2)} + (1-0.0)*log(1-.2)) \end{array} \right) *(-½)=.13$$

Loss functions lead to loss surfaces



Finding the bottom

Which direction should I head?

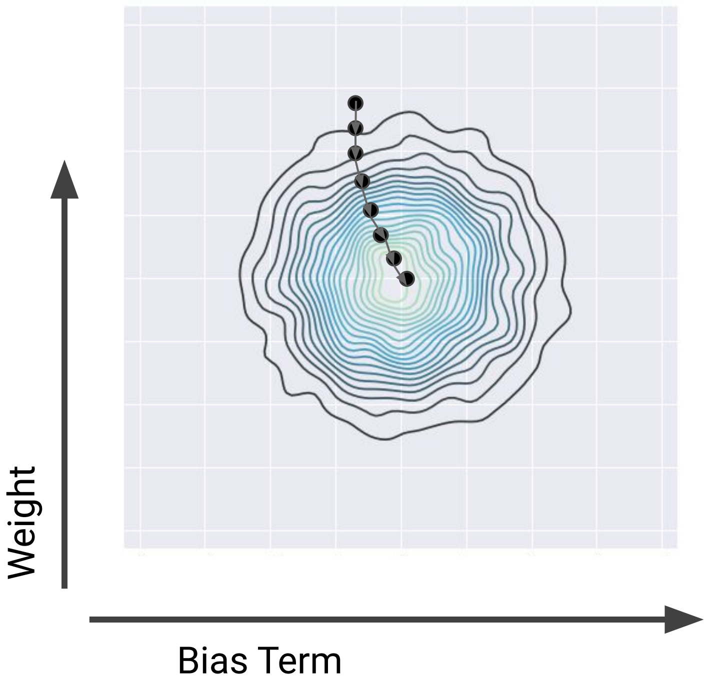
How large or small a step?

A simple algorithm to find the minimum

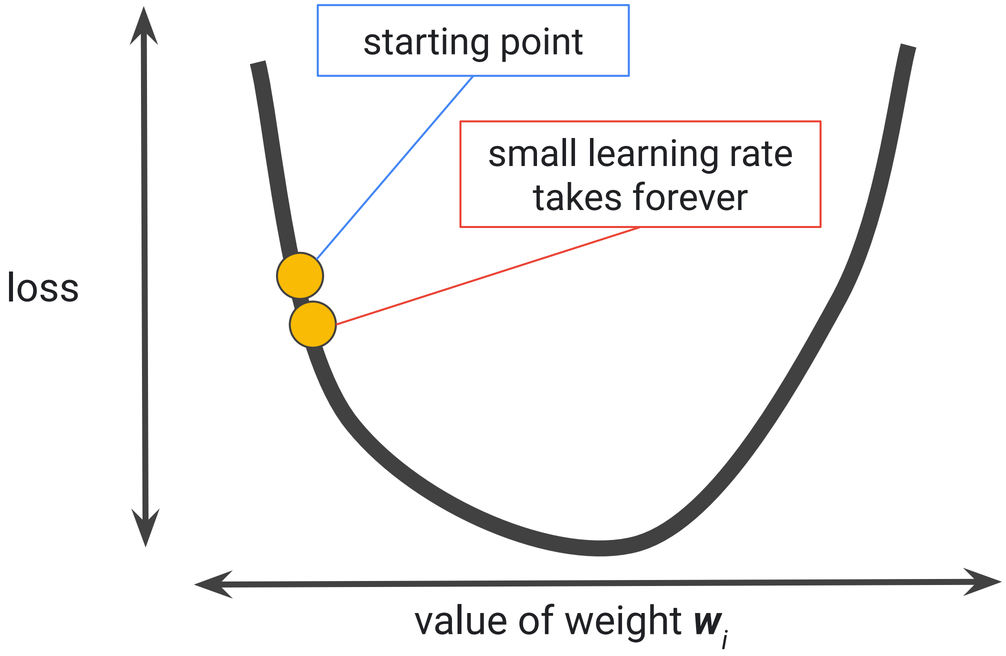
```
while loss is > Epsilon:
    direction = computeDirection()
    for i in range(weights.size):
        weights[i] = weights[i] + stepSize * direction[i]
    loss = computeLoss()
```

Epsilon = A tiny Constant

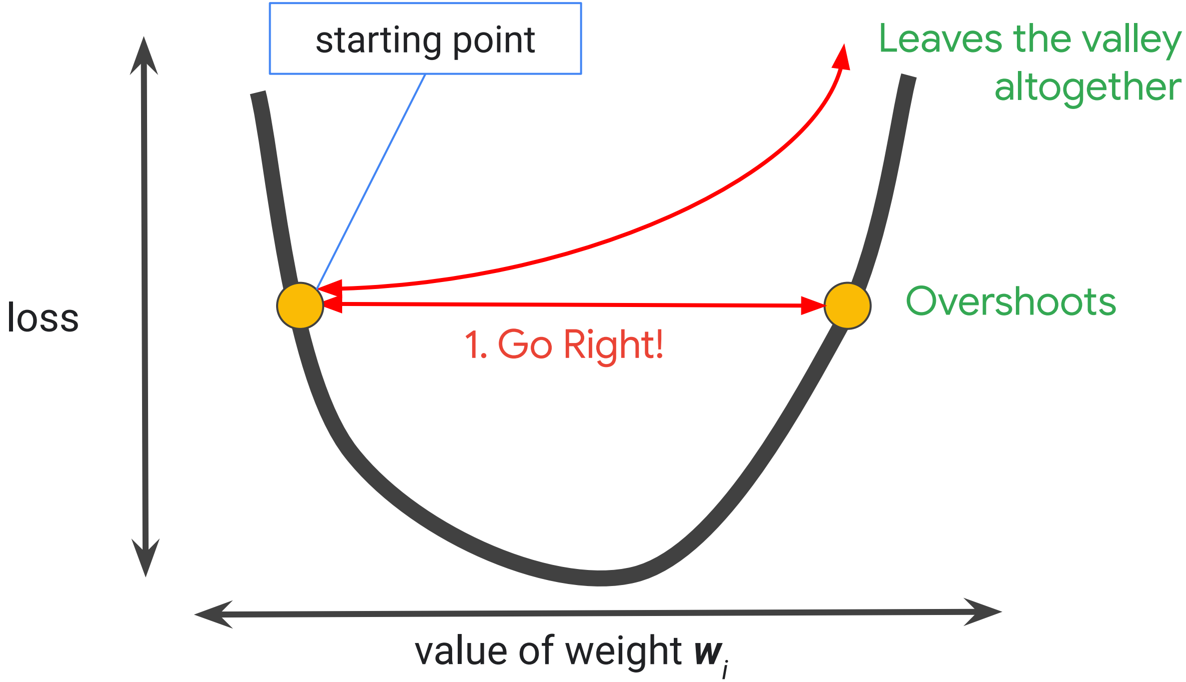
Search for a minima by descending the gradient



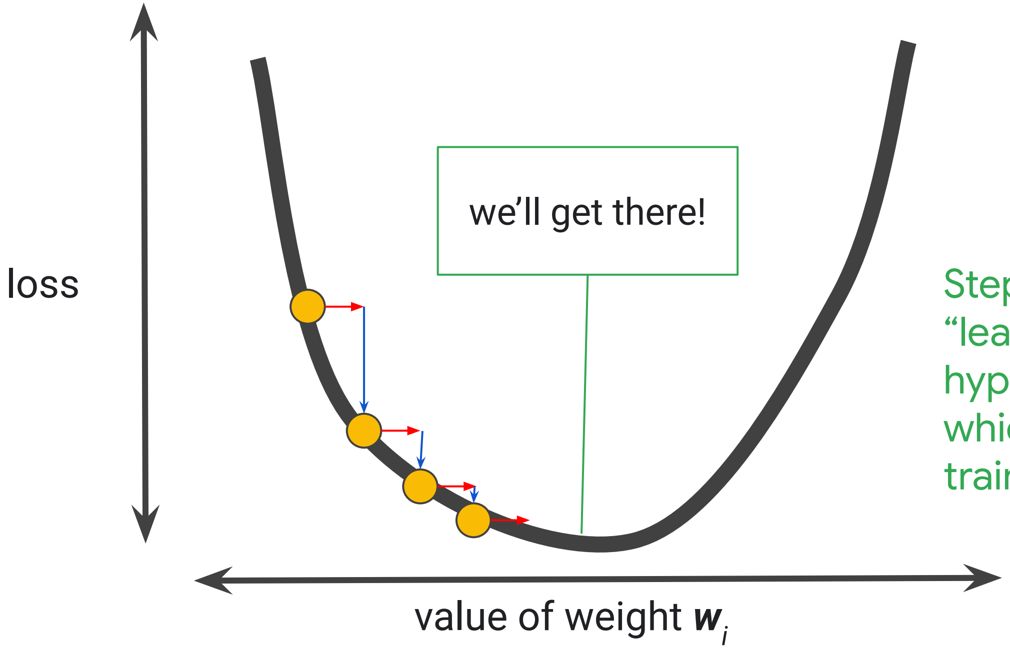
Small step sizes can take a very long time to converge



Large step sizes may never converge to the true minimum

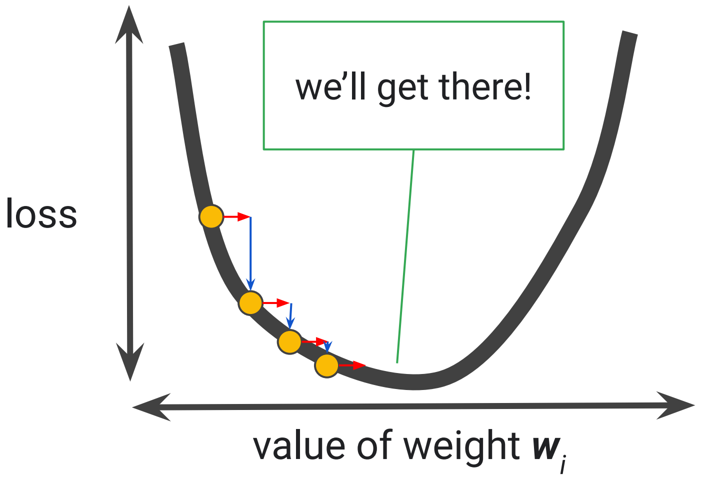


A correct and constant step size can be difficult to find

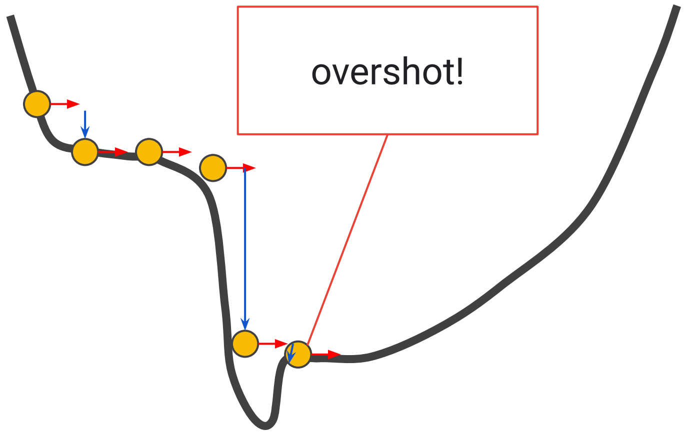


Step size or "learning rate" is a hyper-parameter which is set before training.

A correct and constant step size can be difficult to find

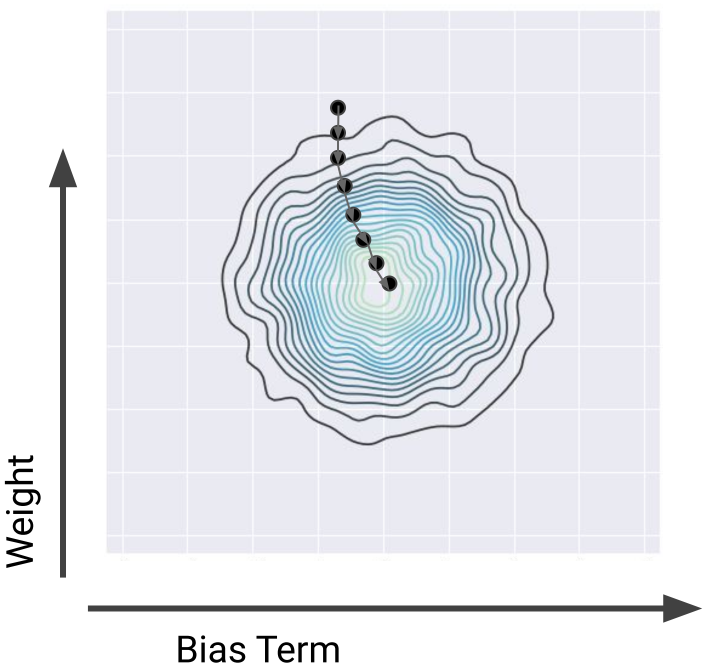


Step size or “learning rate” is a hyper-parameter which is set before training.



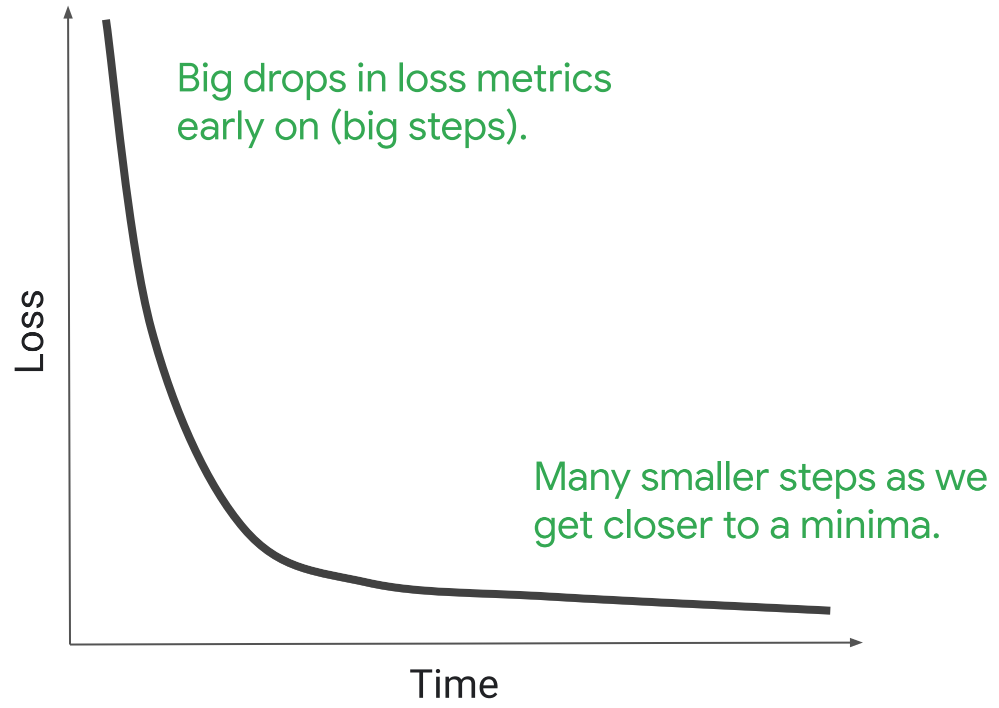
One size does not fit all models.

Are you done yet?

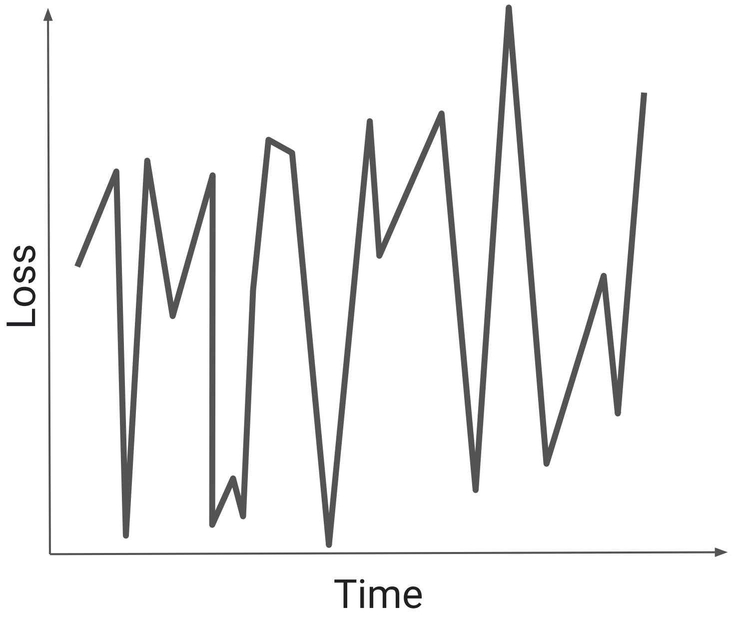


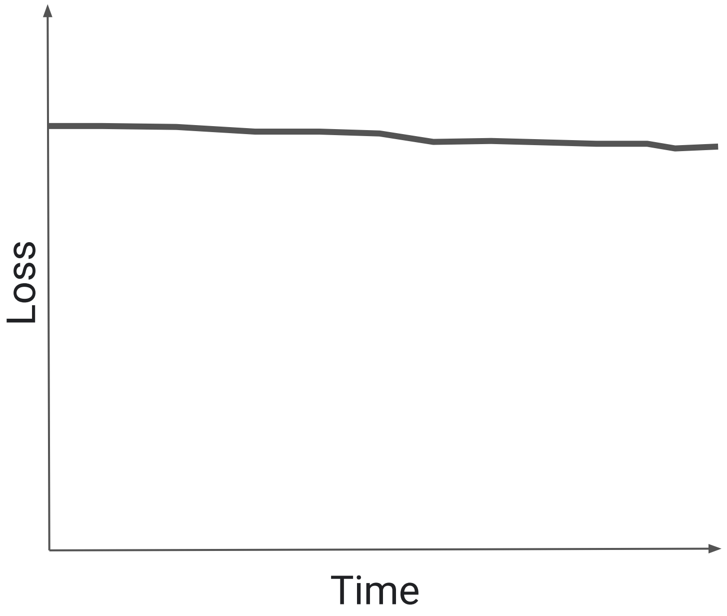

```
while loss is > Epsilon:
     derivative = computeDerivative()
     for i in range(weights.size):
          weights[i] = weights[i] - derivative[i]
     loss = computeLoss()
```

A typical loss curve



Troubleshooting a Loss Curve



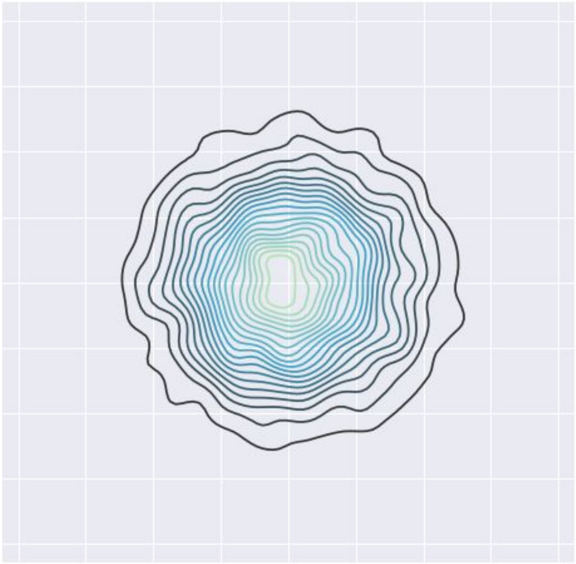


Adding a scaling hyperparameter

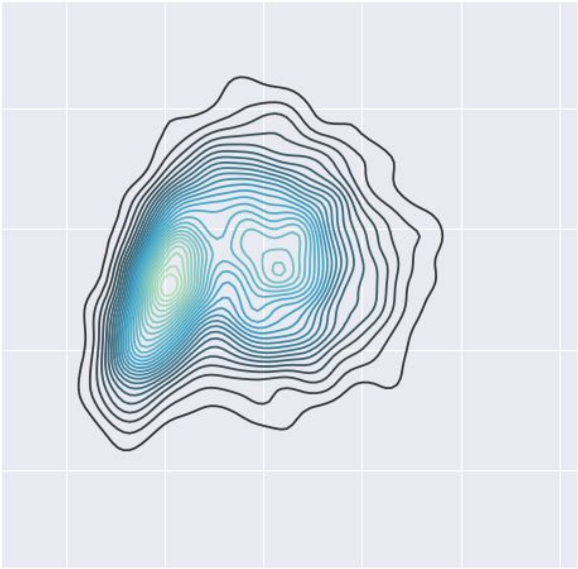
```
while loss is > Epsilon:
     derivative = computeDerivative()
     for i in range(weights.size):
          weights[i] = weights[i] - learning_rate * derivative[i]
     loss = computeLoss()
```

Problem: My model changes every time I retrain it

Loss Surface with a global minimum

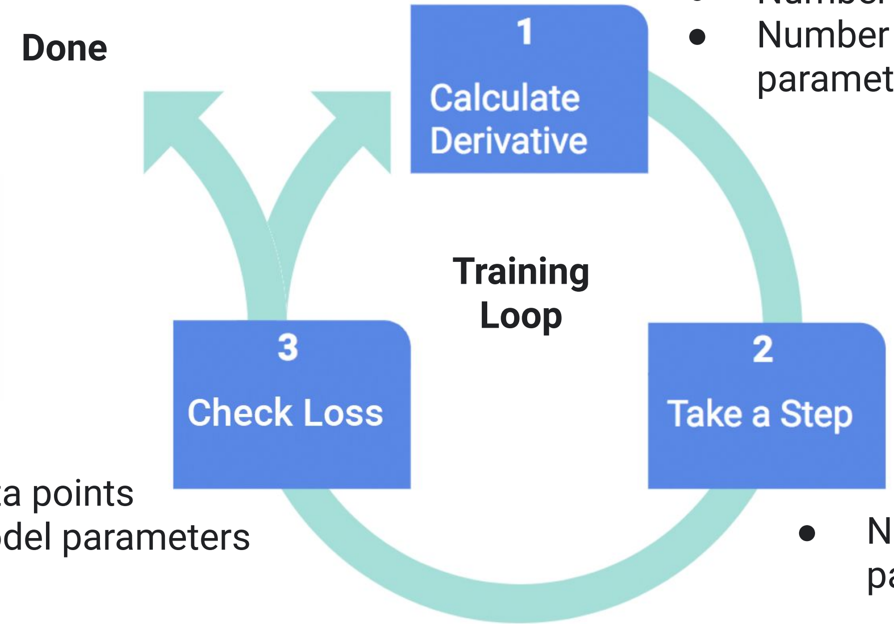


Loss Surface with more than one minima



Problem: Model training is still too slow

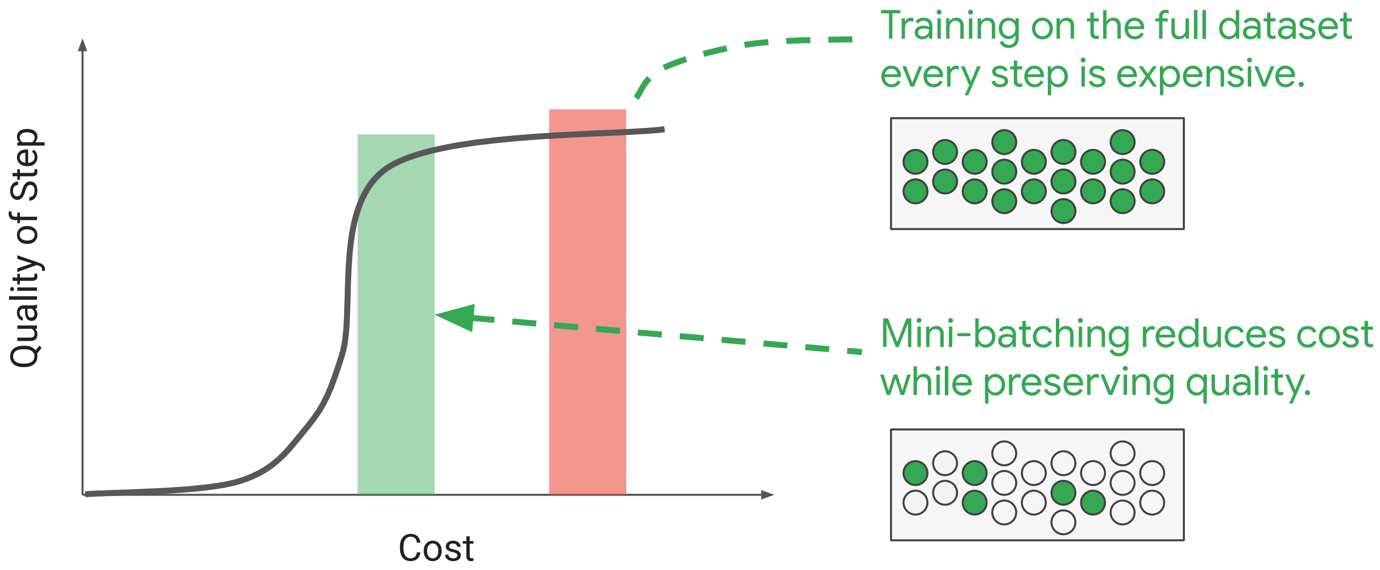
- Number of data points
- Number of model parameters



- Number of data points
- Number of model parameters
- Frequency

- Number of model parameters

Calculating the derivative on fewer data points



Typical values for batch size: 10 - 1000 examples.

Checking loss with reduced frequency

```
while loss is > Epsilon:
    derivative = computeDerivative()
    for i in range(weights.size):
        weights[i] = weights[i] - learning_rate * derivative[i]
    if readyToSampleLoss():
        loss = sampleLoss()
```

Popular implementations for readyToSampleLoss():

- Time-based (e.g., every hour)
- Step-based (e.g., every 1000 steps)

Agenda

Defining ML Models

Introducing Loss Functions

TensorFlow Playground

TensorFlow Playground Interface