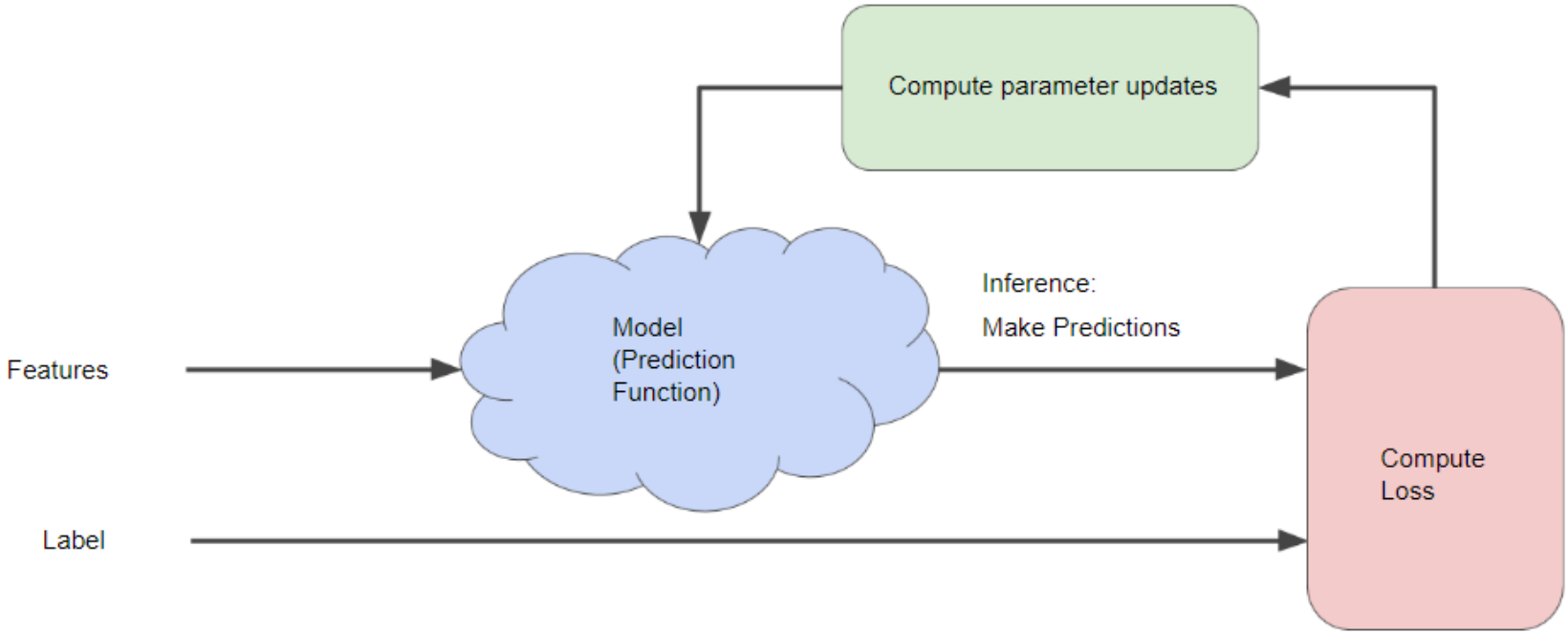
Machine Learning (Lecture 2)

UEM/IEM Summer 2018

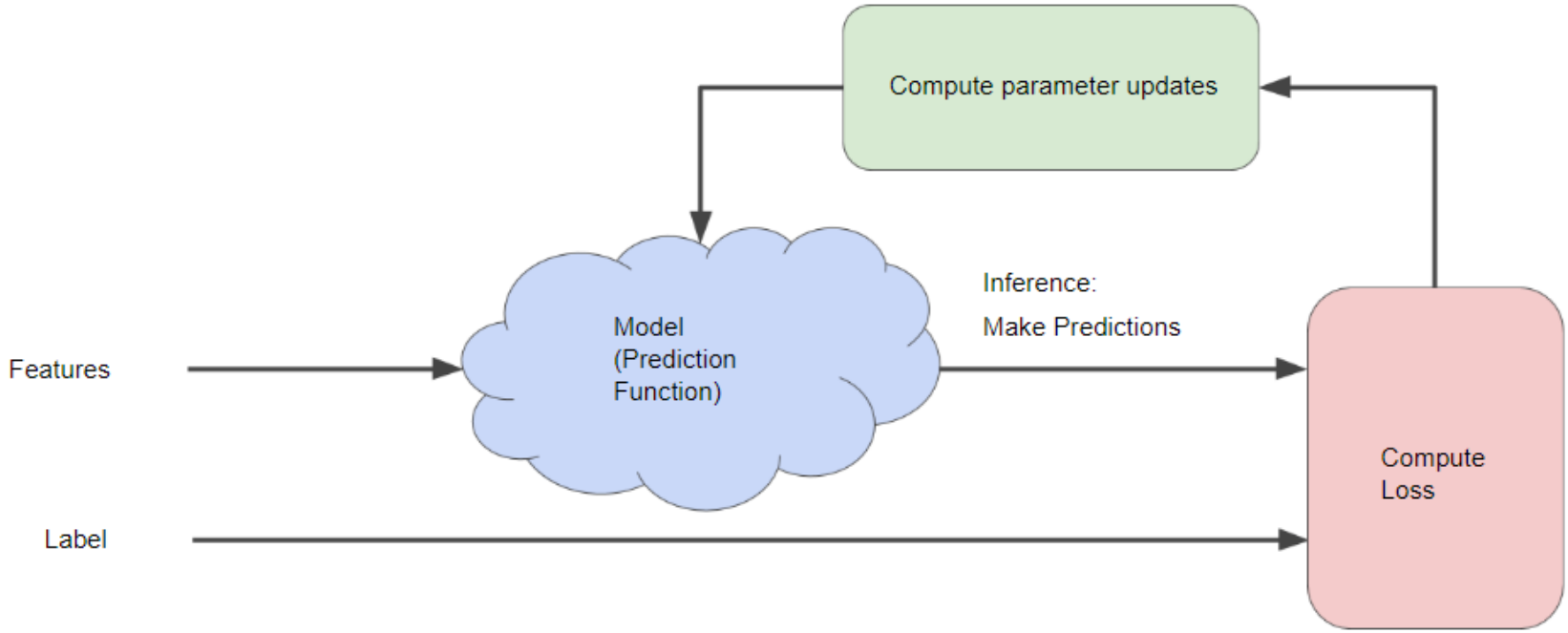
Reducing Loss: Gradient Descent

- Let's replace the green box, "Computer parameter updates" with something more substantial.



Reducing Loss: Gradient Descent

- Let's replace the green box, "Computer parameter updates" with something more substantial.



Reducing Loss: Gradient Descent

- If we calculate the loss for all possible values of w_1, the resulting plot of loss vs. w1 will always be convex for the kind of regression problems we've been examining, looks like the following:

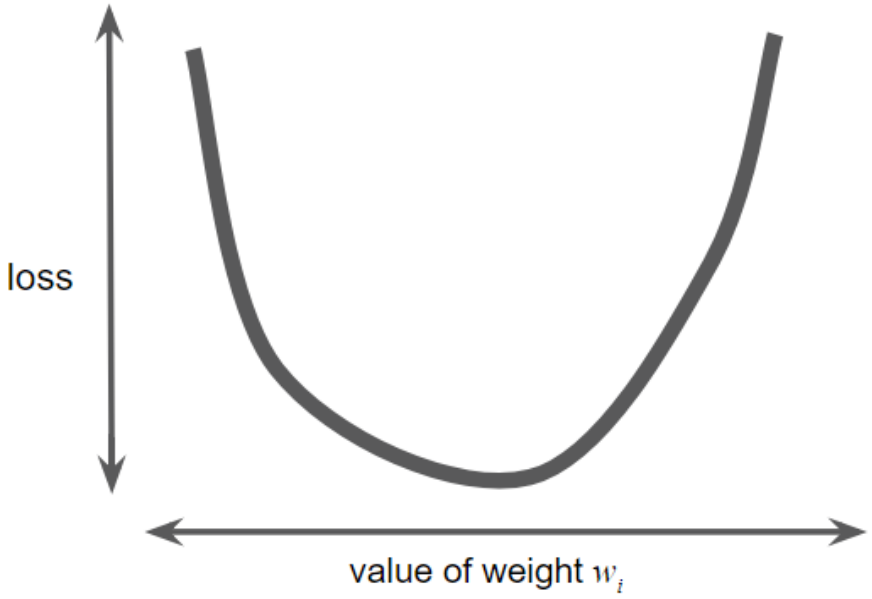


Figure 1. Regression problems yield convex loss vs weight plots.

Reducing Loss: Gradient Descent

- Convex problems have only one minimum; that is, only one place where the slope is exactly 0. That minimum is where the loss function converges.
- Calculating the loss function for every conceivable value of w_1 over the entire data set would be an inefficient way of finding the convergence point. Let's examine a better mechanism—very popular in machine learning—called **gradient descent**.

Reducing Loss: Gradient Descent

- The first stage in gradient descent is to pick a starting value (a starting point) for w_1.
- The starting point doesn't matter much; therefore, many algorithms simply set w_1 to 0 or pick a random value. The following figure shows that we've picked a starting point slightly greater than 0:

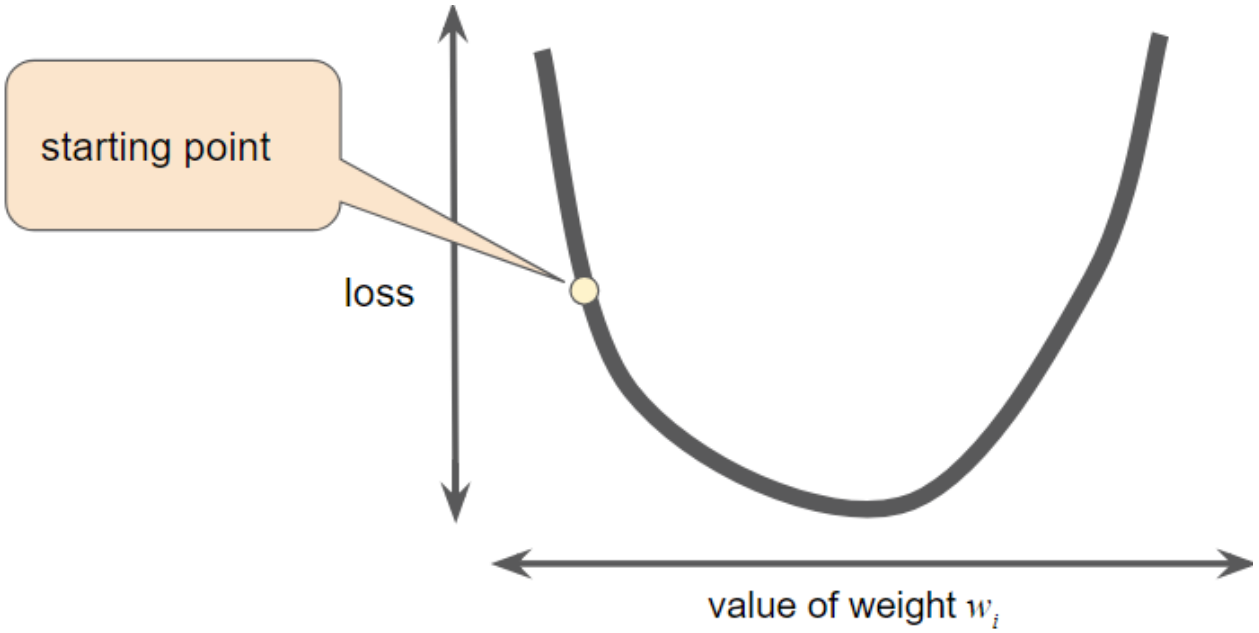


Figure 2. A starting point for gradient descent.

Reducing Loss: Gradient Descent

- The gradient descent algorithm then calculates the gradient of the loss curve at the starting point.
- In Figure 2, the gradient of loss is equal to the derivative (slope) of the curve, and tells you which way is "warmer" or "colder."
- When there are multiple weights, the **gradient** is a vector of partial derivatives with respect to the weights.
- Note that a gradient is a vector, so it has both of the following characteristics:
  - a direction
  - a magnitude

Reducing Loss: Gradient Descent

- The gradient always points in the direction of steepest increase in the loss function.
- The gradient descent algorithm takes a step in the direction of the negative gradient in order to reduce loss as quickly as possible.

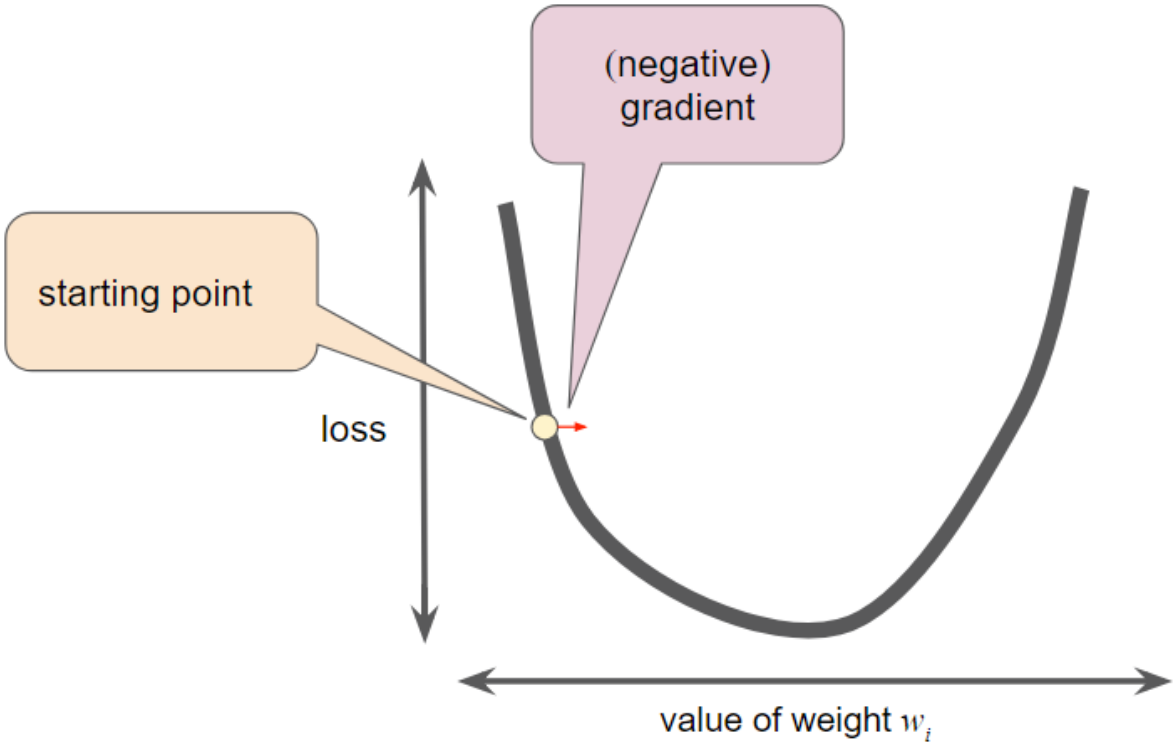


Figure 3. Gradient descent relies on negative gradients.

Reducing Loss: Gradient Descent

- To determine the next point along the loss function curve, the gradient descent algorithm adds some fraction of the gradient's magnitude to the starting point as shown in the following figure:

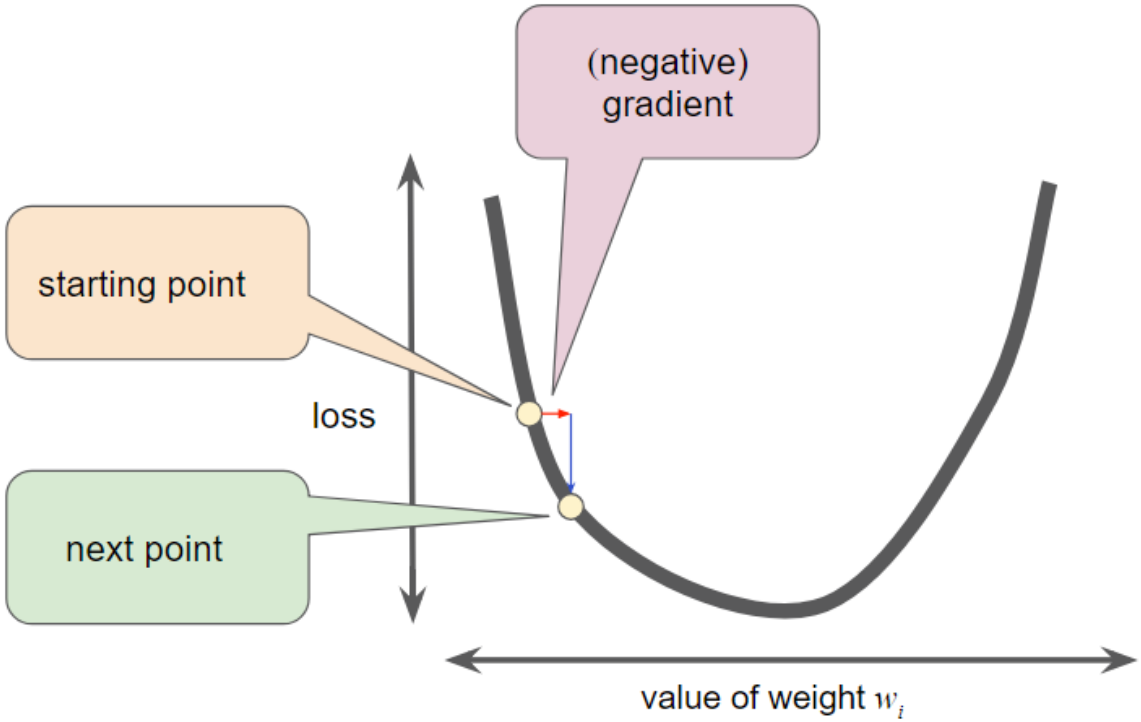


Figure 4. A gradient step moves us to the next point on the loss curve.

- The gradient descent then repeats this process, edging ever closer to the minimum.

Reducing Loss: Learning Rate

- The gradient vector has both a direction and a magnitude. Gradient descent algorithms multiply the gradient by a scalar known as the **learning rate** (also sometimes called **step size**) to determine the next point.
- For example, if the gradient magnitude is 2.5 and the learning rate is 0.01, then the gradient descent algorithm will pick the next point 0.025 away from the previous point.

Reducing Loss: Learning Rate

- **Hyperparameters** are the knobs that programmers tweak in machine learning algorithms.
- Most machine learning programmers spend a fair amount of time tuning the learning rate.
- If you pick a learning rate that is too small, learning will take too long:

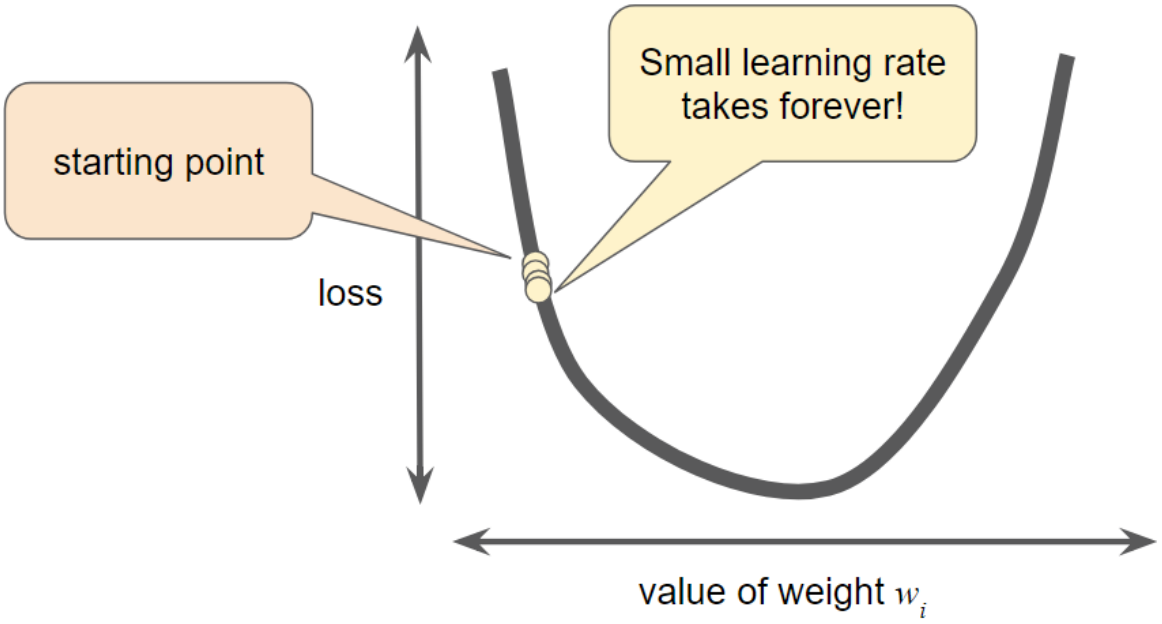


Figure 5. Learning rate is too small.

Reducing Loss: Learning Rate

- Conversely, if you specify a learning rate that is too large, the next point will perpetually bounce haphazardly across the bottom of the well like a quantum mechanics experiment gone horribly wrong:

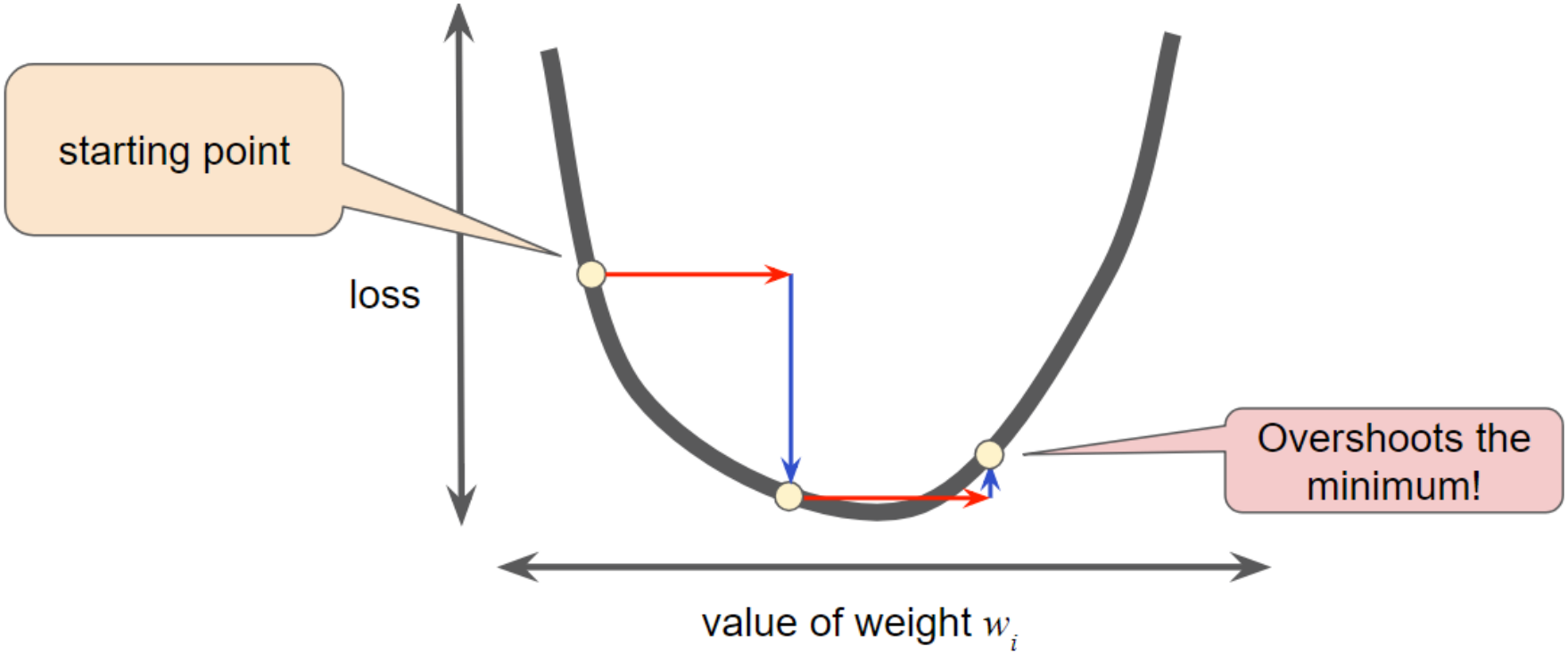


Figure 6. Learning rate is too large.

Reducing Loss: Learning Rate

- There is a Goldilocks (just-right) learning rate for every regression problem.
- The Goldilocks value is related to how flat the loss function is.
- If you know the gradient of the loss function is small then you can safely try a larger learning rate, which compensates for the small gradient and results in a larger step size.

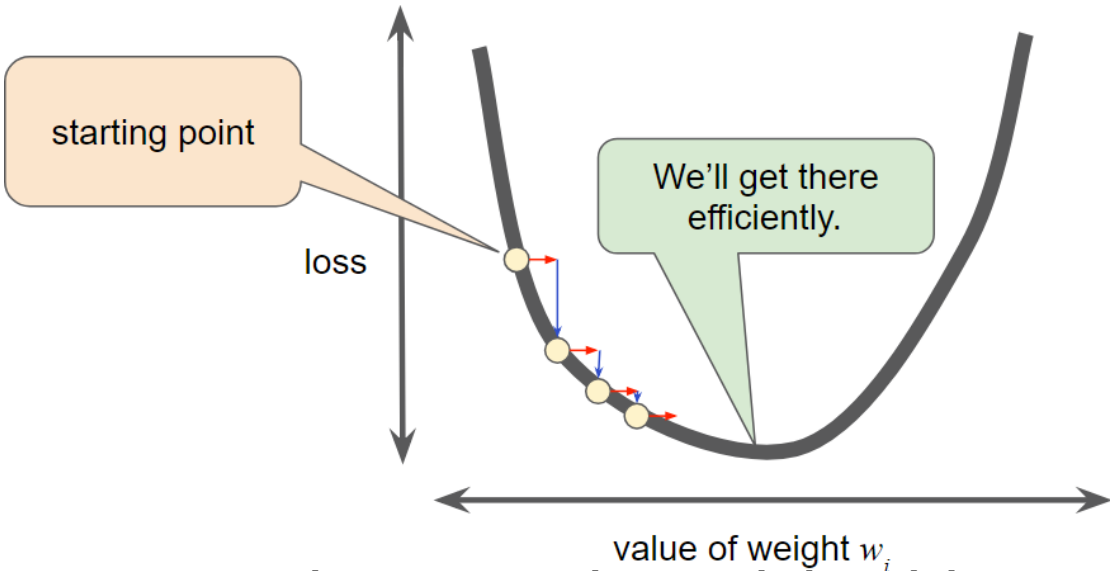


Figure 7. Learning rate is just right.

Optimizing Learning Rate

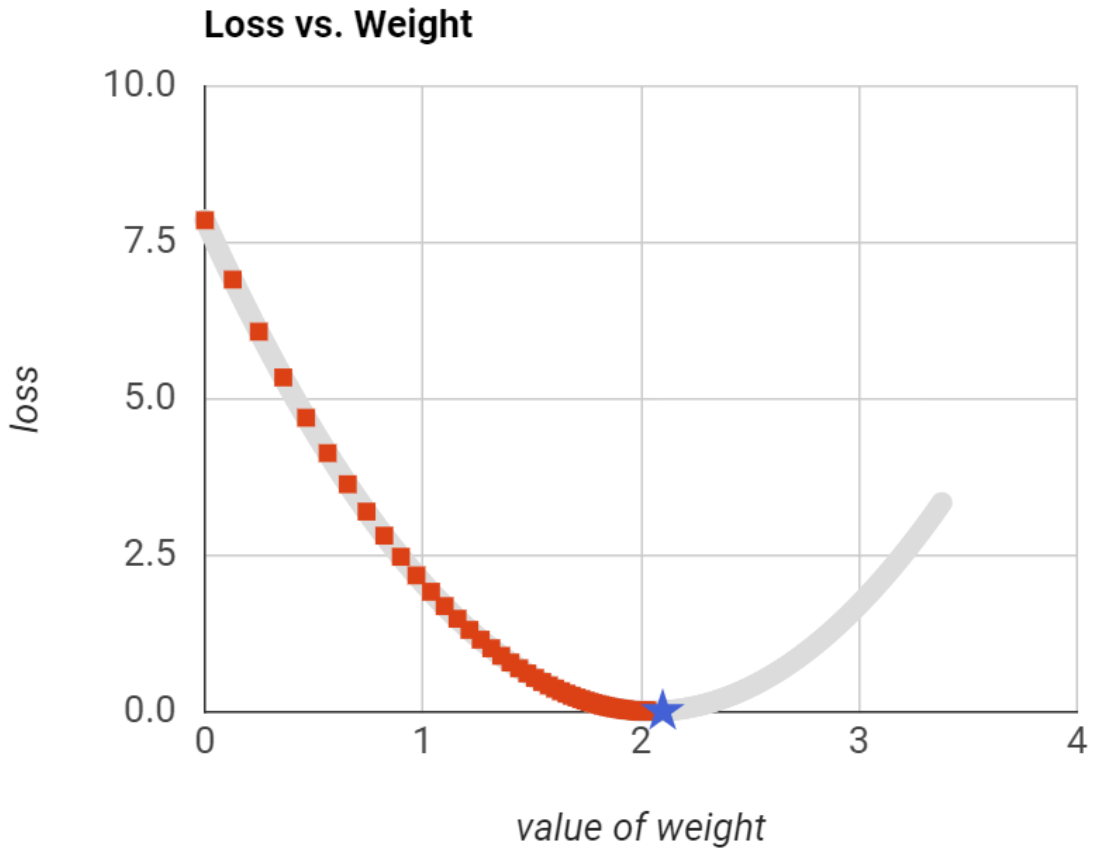
- Experiment with different learning rates and see how they affect the number of steps required to reach the minimum of the loss curve.

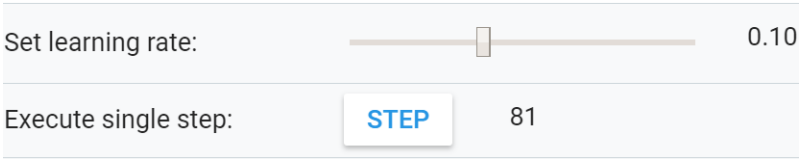
Optimizing Learning Rate

- Set a learning rate of 0.1 on the slider. Keep hitting the STEP button until the gradient descent algorithm reaches the minimum point of the loss curve. How many steps did it take?

Optimizing Learning Rate

- Set a learning rate of 0.1 on the slider. Keep hitting the STEP button until the gradient descent algorithm reaches the minimum point of the loss curve. How many steps did it take?





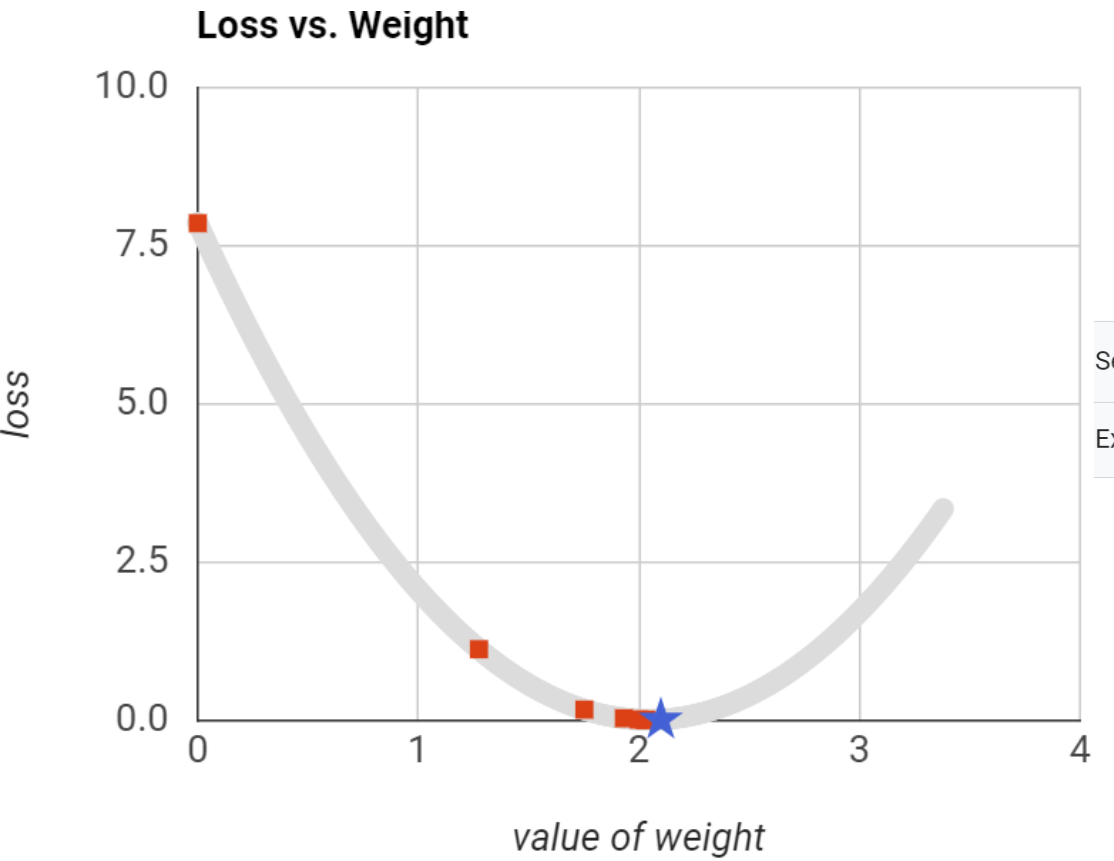
81 Steps

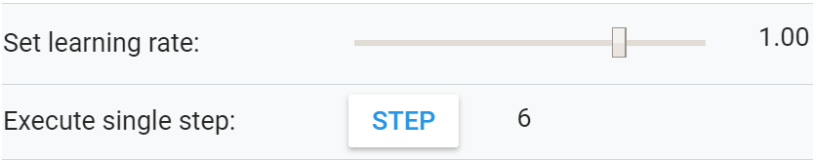
Optimizing Learning Rate

- Can you reach the minimum more quickly with a higher learning rate? Set a learning rate of 1, and keep hitting STEP until gradient descent reaches the minimum. How many steps did it take this time?

Optimizing Learning Rate

- Can you reach the minimum more quickly with a higher learning rate? Set a learning rate of 1, and keep hitting STEP until gradient descent reaches the minimum. How many steps did it take this time?





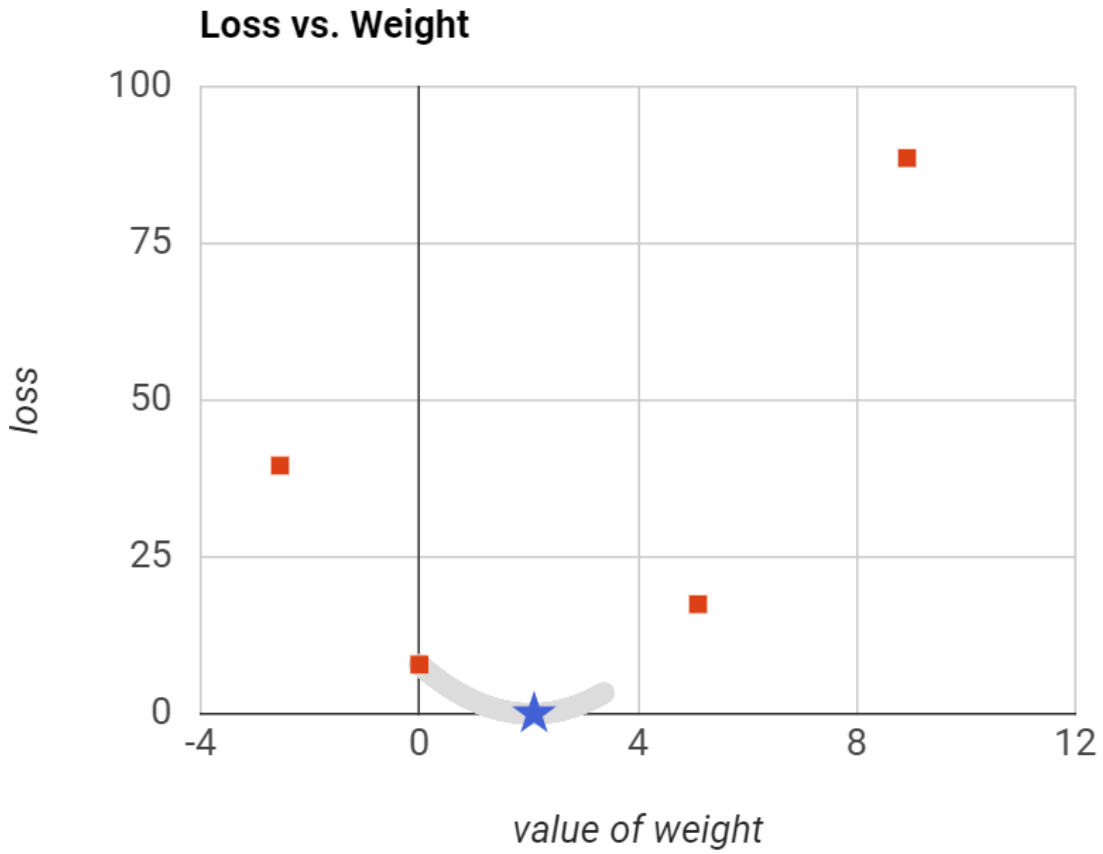
6 Steps

Optimizing Learning Rate

- How about an even larger learning rate. Reset the graph, set a learning rate of 4, and try to reach the minimum of the loss curve. What happened this time?

Optimizing Learning Rate

- How about an even larger learning rate. Reset the graph, set a learning rate of 4, and try to reach the minimum of the loss curve. What happened this time?



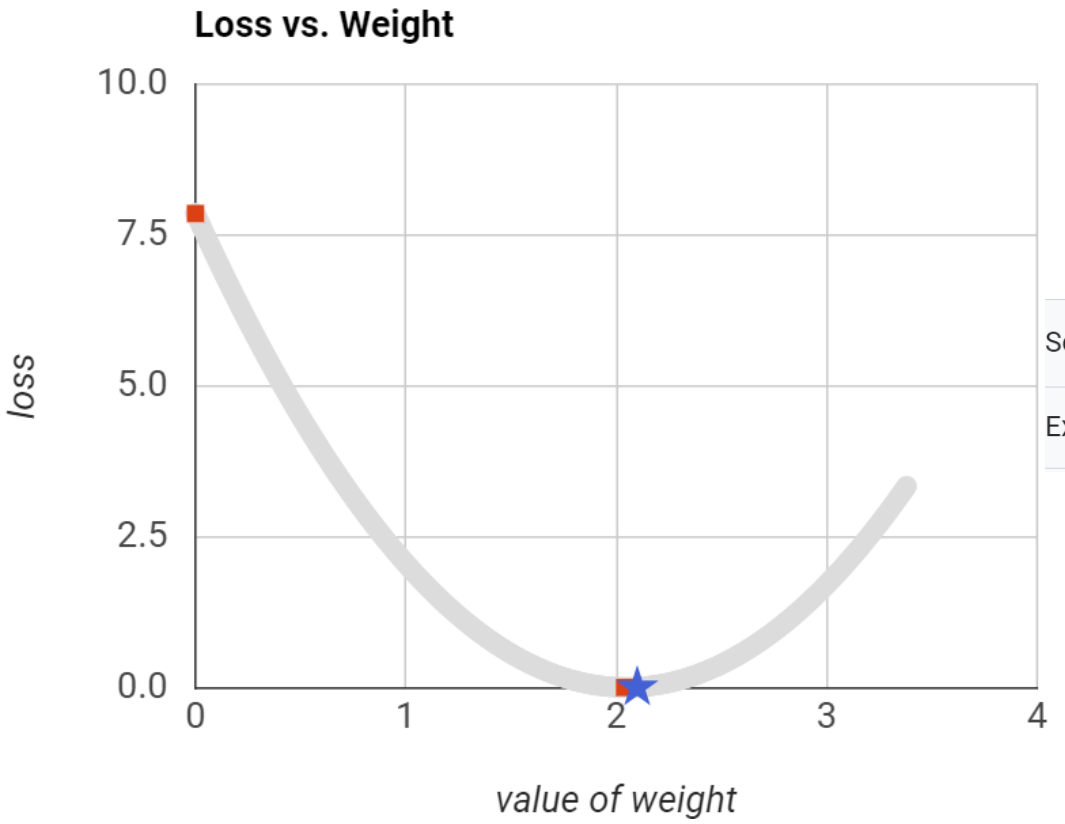
Gradient descent **never reaches the minimum**. As a result, steps progressively increase in size. Each step jumps back and forth across the bowl, climbing the curve instead of descending to the bottom.

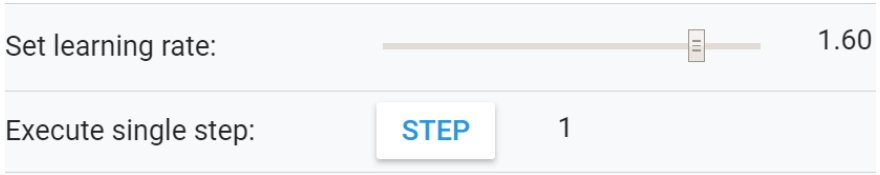
Optimizing Learning Rate

- Can you find the Goldilocks learning rate for this curve, where gradient descent reaches the minimum point in the fewest number of steps? What is the fewest number of steps required to reach the minimum?

Optimizing Learning Rate

- Can you find the Goldilocks learning rate for this curve, where gradient descent reaches the minimum point in the fewest number of steps? What is the fewest number of steps required to reach the minimum?





Goldilocks earning rate: 1.6 with just 1 step

Reducing Loss: Stochastic Gradient Descent

- In gradient descent, a **batch** is the total number of examples you use to calculate the gradient in a single iteration.
- So far, we've assumed that the batch has been the entire data set.
- When working at Google scale, data sets often contain billions or even hundreds of billions of examples.
- Furthermore, Google data sets often contain huge numbers of features. Consequently, a batch can be enormous. A very large batch may cause even a single iteration to take a very long time to compute.

Reducing Loss: Stochastic Gradient Descent

- A large data set with randomly sampled examples probably contains redundant data.
- In fact, redundancy becomes more likely as the batch size grows.
- Some redundancy can be useful to smooth out noisy gradients, but enormous batches tend not to carry much more predictive value than large batches.

Reducing Loss: Stochastic Gradient Descent

- What if we could get the right gradient *on average* for much less computation?
- By choosing examples at random from our data set, we could estimate (albeit, noisily) a big average from a much smaller one.
- **Stochastic gradient descent** (**SGD**) takes this idea to the extreme--it uses only a single example (a batch size of 1) per iteration. Given enough iterations, SGD works but is very noisy. The term "stochastic" indicates that the one example comprising each batch is chosen at random.

Reducing Loss: Stochastic Gradient Descent

- **Mini-batch stochastic gradient descent** (**mini-batch SGD**) is a compromise between full-batch iteration and SGD.
- A mini-batch is typically between 10 and 1,000 examples, chosen at random. Mini-batch SGD reduces the amount of noise in SGD but is still more efficient than full-batch.
- To simplify the explanation, we focused on gradient descent for a single feature. Rest assured that gradient descent also works on feature sets that contain multiple features.

Quiz

- When performing gradient descent on a large data set, which of the following batch sizes will likely be more efficient?

1. A small batch or even a batch of one example (SGD).
2. The full batch.

Quiz

- When performing gradient descent on a large data set, which of the following batch sizes will likely be more efficient?

1. A small batch or even a batch of one example (SGD).
2. The full batch.

First Steps with TensorFlow: Toolkit

- Tensorflow is a computational framework for building machine learning models.
- TensorFlow provides a variety of different toolkits that allow you to construct models at your preferred level of abstraction.
- You can use lower-level APIs to build models by defining a series of mathematical operations.
- Alternatively, you can use higher-level APIs (like tf.estimator) to specify predefined architectures, such as linear regressors or neural networks.

First Steps with TensorFlow: Toolkit

- The following figure shows the current hierarchy of TensorFlow toolkits:

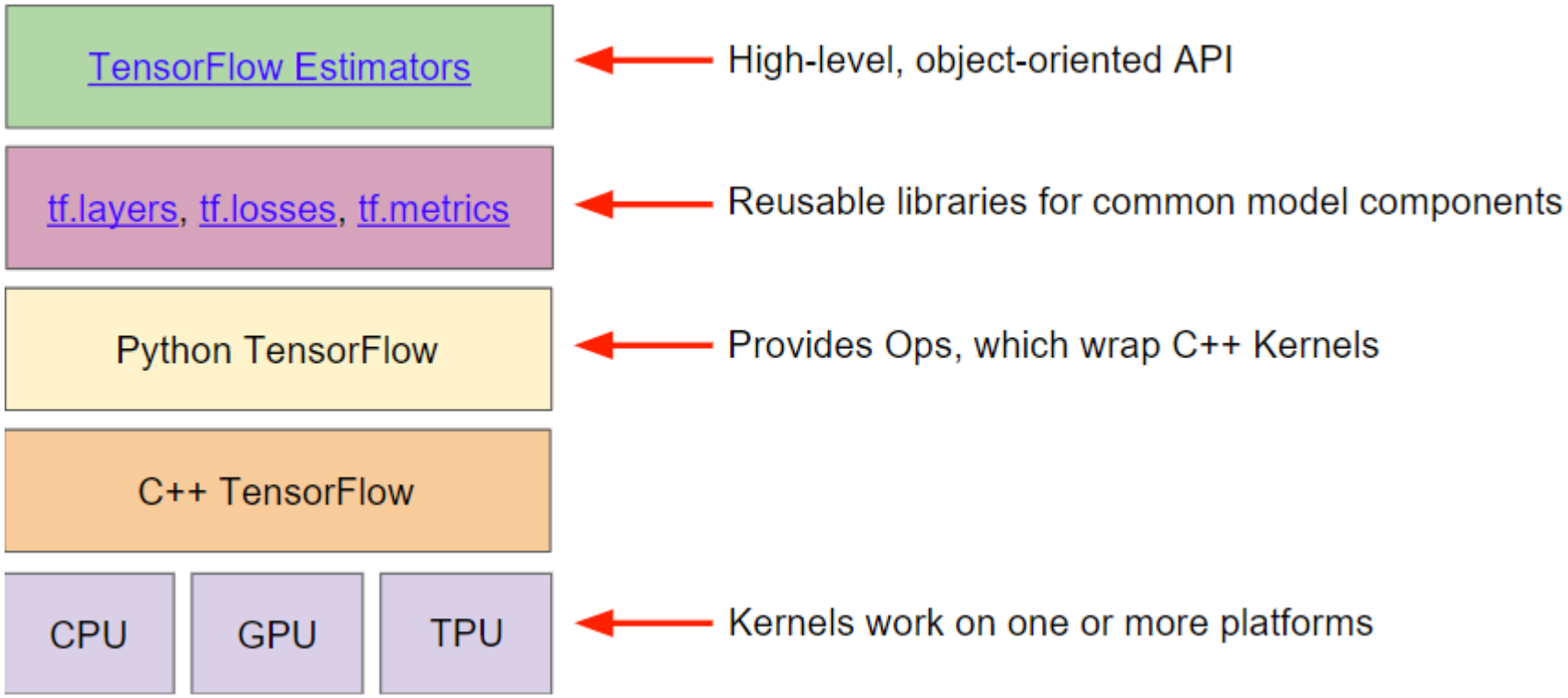


Figure 8. TensorFlow toolkit hierarchy.

First Steps with TensorFlow: Toolkit

- The following table summarizes the purposes of the different layers:

Toolkit(s)	Description
Estimator (tf.estimator)	High-level, OOP API.
tf.layers/tf.losses/tf.metrics	Libraries for common model components.
TensorFlow	Lower-level APIs

First Steps with TensorFlow: Toolkit

- TensorFlow consists of the following two components:
  - a graph protocol buffer
  - a runtime that executes the (distributed) graph
- These two components are analogous to Python code and the Python interpreter.
- Just as the Python interpreter is implemented on multiple hardware platforms to run Python code, TensorFlow can run the graph on multiple hardware platforms, including CPU, GPU, and TPU.

First Steps with TensorFlow: Toolkit

- Which API(s) should you use?
- You should use the highest level of abstraction that solves the problem.
- The higher levels of abstraction are easier to use, but are also (by design) less flexible.
- We recommend you start with the highest-level API first and get everything working.
- If you need additional flexibility for some special modeling concerns, move one level lower.
- Note that each level is built using the APIs in lower levels, so dropping down the hierarchy should be reasonably straightforward.

First Steps with TensorFlow: Toolkit

- Using tf.estimator dramatically lowers the number of lines of code.
- tf.estimator is compatible with the scikit-learn API.
- Scikit-learn is an extremely popular open-source ML library in Python, with over 100k users
- The pseudocode for a linear classification program implemented in tf.estimator:

```
import tensorflow as tf

# Set up a linear classifier.
classifier = tf.estimator.LinearClassifier(feature_columns)

# Train the model on some example data.
classifier.train(input_fn=train_input_fn, steps=2000)

# Use it to predict.
predictions = classifier.predict(input_fn=predict_input_fn)
```

Reference

- This lecture note has been developed based on the machine learning crash course at Google, which is under *Creative Commons Attribution 3.0 License*.