Journey to the Bottom of the Valley

We know we'd like to move the ball to the bottom of the valley, but how do we accomplish this?

Intuitively, we want to push the ball downhill. And that makes sense, but when we're talking about our cost function, how do we know which way is downhill?

Luckily, the gradient provides this exact information.

Technically, the gradient actually points uphill, in the direction of **steepest ascent**. But if we put a – sign at the front this value, we get the direction of **steepest descent**, which is what we want.

You'll learn more about the gradient in a moment, but, for now, just think of it as a vector of numbers. Each number represents the amount by which we should adjust a corresponding weight or bias in the neural network. Adjusting all of the weights and biases by the gradient values reduces the cost (or error) of the network.

Got all that?

Great! Now we know where to push the ball. The next thing to consider is how much force should be applied to the *push*. This is known as the *learning rate*, which is an apt name since this value determines how quickly or slowly the neural network learns.

You might be tempted to set a really big learning rate, so the network learns really fast, right?

Be careful! If the value is too large you could overshoot the target and eventually diverge. Yikes!

Convergence. This is the ideal behaviour.

Divergence. This can happen when the learning rate is too large.

So what is a good learning rate, then?

This is more of a guessing game than anything else but empirically values in the range 0.1 to 0.0001 work well. The range 0.001 to 0.0001 is popular, as 0.1 and 0.01 are sometimes too large.

Here's the formula for gradient descent (pseudocode):

```
x = x - learning_rate * gradient_of_x
```

`x` is a parameter used by the neural network (i.e. a single weight or bias).

We multiply `gradient_of_x` (the uphill direction) by `learning_rate`(the force of the push) and then subtract that from `x` to make the push go downhill.

Awesome! Time to apply all this in a quiz.

Setup

For this quiz you'll complete `TODOs` in both the `f.py` and `gd.py` files.

Tasks:

- Set the `learning_rate` in `f.py`.
- Complete the gradient descent implementation in `gradient_descent_update` function in `gd.py`.

Notes:

- Setting the `learning_rate` to 0.1 should result in `x` -> 0 and `f(x)` -> 5 if you've implemented gradient descent correctly.

- Play around with different values for the learning rate. Try very small values, values close to 1, above 1, etc. What happens?