**What you should remember**:

* The difference between gradient descent, mini-batch gradient descent and stochastic gradient descent is the number of examples you use to perform one update step.
* You have to tune a learning rate hyperparameter αα.
* With a well-turned mini-batch size, usually it outperforms either gradient descent or stochastic gradient descent (particularly when the training set is large).

**2 - Mini-Batch Gradient descent**

Let's learn how to build mini-batches from the training set (X, Y).

There are two steps:

* **Shuffle**: Create a shuffled version of the training set (X, Y) as shown below. Each column of X and Y represents a training example. Note that the random shuffling is done synchronously between X and Y. Such that after the shuffling the ithith column of X is the example corresponding to the ithith label in Y. The shuffling step ensures that examples will be split randomly into different mini-batches.
* **Partition**: Partition the shuffled (X, Y) into mini-batches of size mini\_batch\_size (here 64). Note that the number of training examples is not always divisible by mini\_batch\_size. The last mini batch might be smaller, but you don't need to worry about this. When the final mini-batch is smaller than the full mini\_batch\_size, it will look like this:

**What you should remember**:

* Shuffling and Partitioning are the two steps required to build mini-batches
* Powers of two are often chosen to be the mini-batch size, e.g., 16, 32, 64, 128.

## 3 - Momentum

Because mini-batch gradient descent makes a parameter update after seeing just a subset of examples, the direction of the update has some variance, and so the path taken by mini-batch gradient descent will "oscillate" toward convergence. Using momentum can reduce these oscillations.

Momentum takes into account the past gradients to smooth out the update. We will store the 'direction' of the previous gradients in the variable vv. Formally, this will be the exponentially weighted average of the gradient on previous steps. You can also think of vv as the "velocity" of a ball rolling downhill, building up speed (and momentum) according to the direction of the gradient/slope of the hill.

* The velocity is initialized with zeros. So the algorithm will take a few iterations to "build up" velocity and start to take bigger steps.
* If β=0β=0, then this just becomes standard gradient descent without momentum.

**How do you choose**ββ**?**

* The larger the momentum ββ is, the smoother the update because the more we take the past gradients into account. But if ββ is too big, it could also smooth out the updates too much.
* Common values for ββ range from 0.8 to 0.999. If you don't feel inclined to tune this, β=0.9β=0.9 is often a reasonable default.
* Tuning the optimal ββ for your model might need trying several values to see what works best in term of reducing the value of the cost function JJ.

**What you should remember**:

* Momentum takes past gradients into account to smooth out the steps of gradient descent. It can be applied with batch gradient descent, mini-batch gradient descent or stochastic gradient descent.
* You have to tune a momentum hyperparameter ββ and a learning rate αα.

**4 - Adam**

Adam is one of the most effective optimization algorithms for training neural networks. It combines ideas from RMSProp (described in lecture) and Momentum.

**How does Adam work?**

1. It calculates an exponentially weighted average of past gradients, and stores it in variables vv (before bias correction) and vcorrectedvcorrected (with bias correction).
2. It calculates an exponentially weighted average of the squares of the past gradients, and stores it in variables ss (before bias correction) and scorrectedscorrected (with bias correction).
3. It updates parameters in a direction based on combining information from "1" and "2".

The update rule is, for l=1,...,Ll=1,...,L:

|  |  |  |
| --- | --- | --- |
| **optimization method** | **accuracy** | **cost shape** |
| Gradient descent | 79.7% | oscillations |
| Momentum | 79.7% | oscillations |
| Adam | 94% | smoother |

Momentum usually helps, but given the small learning rate and the simplistic dataset, its impact is almost negligeable. Also, the huge oscillations you see in the cost come from the fact that some minibatches are more difficult thans others for the optimization algorithm.

Adam on the other hand, clearly outperforms mini-batch gradient descent and Momentum. If you run the model for more epochs on this simple dataset, all three methods will lead to very good results. However, you've seen that Adam converges a lot faster.

Some advantages of Adam include:

* Relatively low memory requirements (though higher than gradient descent and gradient descent with momentum)
* Usually works well even with little tuning of hyperparameters (except αα)