Mini-batch gradient descent

- Vectorization allows you to efficiently compute on m examples. But m if is really large that can still be slow. A solution to this is only ingest a small fixed amount of examples (1000, for example) and use each one to iteratively compute the errors.
- Notation:
  - $x^{(i)}$ ith example in the test set
  - $Z^{[l]}$ lth layer in the neural network
  - X^{i} ith batch in the mini-batch test set. $X^{i}.\mathrm{shape} = (n_x, m)$

Understanding mini-batch gradient descent

- With the batch the cost should decrease on every iteration
- With the mini batch you're using just a small sample of the data and while it still decrease on time the graph cost x mini batch iteration is much noiser
- On both extremes if size = m then $(X^{i}, Y^{i}) = (X, Y)$ and if size = 1 then $(X^{i}, Y^{i}) = (x^{(i)}, y^{i})$
- In practice size must be between 1 and m to ensure that it really converges
- With size = m it takes too long but the advantages is that the convergence has much less noise
- With stochastic small values you add too much noise and also loses the advantages of vectorization
- The in-between has both advantages (vectorization on memory and quick conversion) with reduced disadvantages
- But how do I choose my batch size? If the training set is small use it all. If it's bigger use a mini-batch size closest to a given n where size = 2^{n}

Exponentially weighted averages

- $V_t = \beta V_{t-1} + (1-\beta)\theta_t$
- Small β values implies shorter windows of influence, quick response regressions and much noiser curves
- Long β values implies in longer windows of influence, delayed regressions and smoother curves

Understanding exponentially weighted averages

- The impact of beta changes much simply because the equation is recursive. Recursively the importance of the β value decreases exponentially
- Implementing the recursive code we don't really care for all the V's, just the current day V value

Bias correction in exponentially weighted averages

- As $V_0 = 0$ we have no prior info on the starting days until the regression achieves the minimum number of days
- To correct this simply uses $\frac{V_t}{1-\beta^t}$ which, on the given number of days, will simple make no effect anymore but in the starting days will serve as normalizing factor

Gradient descent with momentum

- In one sentence, the basic idea is to compute an exponentially weighted average of your gradients, and then use that gradient to update your weights instead.
- Momentum add the moving averages equation to the derivative calculations adding the impact of previous iterations

Momentum:

```
Compute dW and db on current mini-batch
```

$$V_{dW} = \beta V_{dW} + (1-\beta)dW$$

$$V_{db} = \beta V_{db} + (1-\beta)db$$

$$W = W - \alpha V_{dW}$$

$$b = b - \alpha V_{db}$$

- With this algorithm, with a few iterations you find that the gradient descent with momentum ends up eventually just taking steps that are much smaller oscillations in the vertical direction, but are more directed to just moving quickly in the horizontal direction.
- The $(1-\beta)$ can be seen as acceleration and the β as the velocity on that specific point
- The bias correction can be implemented but as with just few steps it'll correct itself it makes not much sense to many to implement it
- With these equations β becomes a new hyperparameter which directly interfere on the learning rate α

RMSProp (Root Means Square Propagation)

RMSProp:

```
Compute dW and db on current mini-batch
```

$$S_{dW} = \beta S_{dW} + (1-\beta)dW^2$$ `<- element-wise operation`

$$S_{db} = \beta S_{db} + (1-\beta)db^2$$ `<- element-wise operation`

$$W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon}$$

$$b = b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$$

- Intuition is that in dimensions where you're getting these oscillations, you end up computing a larger sum. A weighted average for these squares and derivatives, and so you end up dumping out the directions in which there are these oscillations.
- The ε is present in the denominator to avoid division by 0

Adam optimization algorithm

- Not many of the proposed optimization algorithms generalized well for many problems and were not widespreadly used.
- Adam optimization is basically a combination of both Momentum and RMSProp algorithms
- In a regular Adam implementation you'll also have bias correction implemented

Adam:

```
Initialize $V_{dW} = 0, V_{db} = 0, S_{dW} = 0, S_{db} = 0$

Compute dW and db on current mini-batch

$V_{dW} = \beta_1 V_{dW} + (1 - \beta_1)dW$ <- "momentum"

$V_{db} = \beta_1 V_{db} + (1 - \beta_1)db$ <- "momentum"

$S_{dW} = \beta_2 S_{dW} + (1 - \beta_2)dW^2$ <- "rmsprop"

$V_{db} = \beta_2 S_{db} + (1 - \beta_2)db^2$ <- "rmsprop"

$V_{dW}^{corrected} = \frac{V_{dW}}{(1-\beta_1^t)}$

$V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_1^t)}$

$S_{dW}^{corrected} = \frac{S_{dW}}{(1-\beta_2^t)}$

$S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$

$W = W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}}+\varepsilon}$

$b = b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\varepsilon}$
```

Hyperparameters:

- α need to be tuned
- $\beta_1 = 0.9(dW)$
- $\beta_2 = 0.999(dW^2)$
- $\varepsilon = 10^{-8}$

Learning rate decay

- As you iterate your steps will be a little bit noise and never really converge. The intuition is that during the initial steps of learning, you could afford to take much bigger steps. But then as learning approaches converges, then having a slower learning rate allows you to take smaller steps.

Equations/Techniques to control the learning rate decay

- 1 epoch = 1 pass though the data (all the mini batches)
- $\alpha = \frac{1}{1+\text{decay}_{\text{rate}}*\text{epoch}_{\text{num}}}$
- $\alpha = 0.95^{\text{epoch}_{\text{num}}} * \alpha_0$
- $\alpha = \frac{k}{\sqrt{\text{epoch}_{\text{num}}}}\alpha_0$
- Discrete staircase (values of α change to determined values after n iteration making a descending staircase)
- Manual control

The problem of local optima

- A local optima is a place where the algorithm stops learning because it got stuck into the “bottom” of a multidimensional space
- In 2 dimensional space is pretty easy to “fall” into this kind of situations but as deep learning usually needs to deal with higher dimensional spaces to achieve such “bottom” all the dimensions would have to converge and have a 0 derivative on that point.
- As the number of dimension increases the probability that this would happen decreases as a chance that all the dimensions have 0 derivatives is roughly 2^n
- Plateaus are much bigger problems because the algorithm still move the error function in the right direction but much more slowly. In this scenarios Momentum, RMSProp and Adam can help speed up the training

Quiz

1. Which notation would you use to denote the 3rd layer’s activations when the input is the 7th example from the 8th minibatch?
   - ☑ $a^{[3]8(7)}$
   - ☐ $a^{[8]7(3)}$
   - ☐ $a^{[8]3(7)}$
   - ☐ $a^{[3]7(8)}$
2. Which of these statements about mini-batch gradient descent do you agree with?
   - ☐ You should implement mini-batch gradient descent without an explicit for-loop over different mini-batches, so that the algorithm processes all mini-batches at the same time (vectorization).
   - ☐ Training one epoch (one pass through the training set) using mini-batch gradient descent is faster than training one epoch using batch gradient descent.
   - ☑ One iteration of mini-batch gradient descent (computing on a single mini-batch) is faster than one iteration of batch gradient descent.
3. Why is the best mini-batch size usually not 1 and not m, but instead something in-between?
   - ☐ If the mini-batch size is m, you end up with stochastic gradient descent, which is usually slower than mini-batch gradient descent.
   - ☐ If the mini-batch size is 1, you end up having to process the entire training set before making any progress.
   - ☑ If the mini-batch size is 1, you lose the benefits of vectorization across examples in the mini-batch.
   - ☑ If the mini-batch size is m, you end up with batch gradient descent, which has to process the whole training set before making progress.

4. Suppose your learning algorithm's cost JJ, plotted as a function of the number of iterations, looks like this: Suppose your learning algorithm's cost JJ, plotted as a function of the number of iterations, looks like this: Which of the following do you agree with?
   - ☐ Whether you're using batch gradient descent or mini-batch gradient descent, something is wrong.
   - [✓] If you're using mini-batch gradient descent, this looks acceptable. But if you're using batch gradient descent, something is wrong.
   - ☐ If you're using mini-batch gradient descent, something is wrong. But if you're using batch gradient descent, this looks acceptable.
   - ☐ Whether you're using batch gradient descent or mini-batch gradient descent, this looks acceptable.
5. Suppose the temperature in Casablanca over the first three days of January are the same: Jan 1st: $\theta_1 = 10^oC$ Jan 2nd: $\theta_2 10^oC$ (We used Fahrenheit in lecture, so will use Celsius here in honor of the metric world.) Say you use an exponentially weighted average with $\beta = 0.5 \beta = 0.5$ to track the temperature: $v_0 = 0, v_t = \beta v_{t-1} + (1 - \beta)\theta_t$. If v_2 is the value computed after day 2 without bias correction, and $v_2^{corrected}$ is the value you compute with bias correction. What are these values? (You might be able to do this without a calculator, but you don't actually need one. Remember what is bias correction doing.) (**Notes**: v_2 is 7.5 because no bias correction is being made in this calculation. If the bias correction was used in all the steps both v_2 and its corrected counterpart would be 10)
   - ☐ $v_2 = 10, v_2^{corrected} = 7.5$
   - [✓] $v_2 = 7.5, v_2^{corrected} = 10$
   - ☐ $v_2 = 7.5, v_2^{corrected} = 7.5$
   - ☐ $v_2 = 10, v_2^{corrected} = 10$
6. Which of these is NOT a good learning rate decay scheme? Here, t is the epoch number.
   - ☐ $\alpha = \frac{1}{1+2*t}$
   - [✓] $\alpha = e^t \alpha_0$
   - ☐ $\alpha = \frac{1}{\sqrt{t}}$
   - ☐ $\alpha = 0.95^t \alpha_0$
7. You use an exponentially weighted average on the London temperature dataset. You use the following to track the temperature: $v_t = \beta v_{t-1} + (1 - \beta)\theta_t$. The red line below was computed using $\beta = 0.9$. What would happen to your red curve as you vary β? (Check the two that apply)
   - ☐ Decreasing β will shift the red line slightly to the right.
   - [✓] Increasing β will shift the red line slightly to the right.
   - [✓] Decreasing β will create more oscillation within the red line.
   - ☐ Increasing β will create more oscillations within the red line.
8. Consider this figure. These plots were generated with gradient descent; with gradient descent with momentum (β = 0.5) and gradient descent with momentum (β = 0.9). Which curve corresponds to which algorithm? (**Note**: On the equation $v_t = \beta v_{t-1}(1 - \beta)v_t$ as greater is the beta less variation we will see on the function. With a large beta the less varying function is the (3). Large betas means considering more the past and less the actual measurement. Small beta is prone to be noiser as it consider a shorter span of info to give the equation.)
   - ☐ (1) is gradient descent. (2) is gradient descent with momentum (large β) . (3) is gradient descent with momentum (small β)
   - ☐ (1) is gradient descent with momentum (small β), (2) is gradient descent with momentum (small β), (3) is gradient descent
   - ☐ (1) is gradient descent with momentum (small β). (2) is gradient descent. (3) is gradient descent with momentum (large β)

- ✓ (1) is gradient descent. (2) is gradient descent with momentum (small β). (3) is gradient descent with momentum (large β)

9. Suppose batch gradient descent in a deep network is taking excessively long to find a value of the parameters that achieves a small value for the cost function (W[1],b[1],…,W[L],b[L]). Which of the following techniques could help find parameter values that attain a small value for? (Check all that apply)
   - ✓ Try mini-batch gradient descent
   - ✓ Try better random initialization for the weights
   - ✓ Try tuning the learning rate α
   - ✓ Try using Adam
   - ☐ Try initializing all the weights to zero
10. Which of the following statements about Adam is False?
    - ☐ The learning rate hyperparameter α in Adam usually needs to be tuned.
    - ✓ Adam should be used with batch gradient computations, not with mini-batches.
    - ☐ We usually use "default" values for the hyperparameters β_1, β_2 and ε in Adam $(\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8})$
    - ☐ Adam combines the advantages of RMSProp and momentum