Masters Course
Neural Networks

Summary
On
Week 2 & 3

Name: Noha Ahmad Darwish
Submitted to: Dr/Omar Nasr

- Training on large datasets is a very slow process since we have to process the entire training set before we take a single step in gradient descent.
- Batch gradient descent refers to processing the entire dataset at once which yields very slow updates in gradients.
- Mini batch gradient descent refers to processing M examples only from the dataset to achieve faster updates to the gradient.
- In Mini batch gradient descent $1<M<N$, where N is the size of the entire dataset and 1 is only one training example. If M = N, we get no speedup as this is the gradient descent. And if M = 1, we will update the gradients after each training example which is called SGD (Stochastic Gradient Descent).
- SGD is very noisy and it will never converge to the global minima yet on average it will lead us to the direction of the minimum.
- If the dataset size is less than 2000 then there is no need for mini batch gradient descent.
- If more than that, try to choose a size M where M is a power of 2, starting from 64, 128,... .
- One last point to keep in mind, the mini batch size (M) must fit in CPU/GPU memory used in the training phase.
- Exponentially weighted average technique is used for curve fitting and prediction problems.
- The prediction Y(n) = beta × Y(n-1) + (1-beta) X(n), where it can be thought of as averaging over 1/(1-beta) so if beta = 0.9 then we're averaging over the past 10 days.
- Bias correction is used to get a better estimate of the initial phase predictions' when beta is very large. That's achieved by replacing Y(n) by Y(n)/(1-beta^n).
- Bias correction is essential in case you're interested in the initial phase of the training.
- Gradient descent with momentum is based on the weighted average algorithm, explained before.
- When moving towards the minimum there are oscillations in the vertical direction that prevents us from increasing the learning rate to avoid overshooting.
- In GD with momentum, we use these set of eqns:
  1- vdw = beta × vdw + (1-beta) × dW
  2- vdb = beta × vdb + (1-beta) × db
  3- w = w - alpha × Vdw, b = b - alpha × vdb
- We can think of dW and db as the acceleration while Vdw and Vdb as the velocity and beta as the friction coefficient
- RMS prop stands for root mean square prop. Again, we try to decrease the overshoots and oscillations in the vertical direction by following theses equations:
  1- Sdw = beta × Sdw + (1-beta) × dW^2
  2- Sdb = beta × Sdb + (1-beta) × db^2
  3- w = w - alpha × (dW/sqrt(Sdw))
  4- b = b - alpha × (db/sqrt(Sdb))
- To understand how the previous set of eqns. help us achieve this, we shall keep in mind that dW is relatively small when compared to db (slope in vertical direction) so Sdb is relatively large thus dividing by it yields slow updates in the vertical direction

whereas Sdw is relatively small so we achieve faster updates in the horizontal direction.

- Note that the hyperparameter beta in weighted average is different from beta in the RMS prop.
- Adam optimization algorithm combines both the weighted average and the RMS prop as shown below:
  1- vdw = beta1 × vdw + (1-beta1) × dW
  2- vdb = beta1 × vdb + (1-beta1) × db
  3- Sdw = beta2 × Sdw + (1-beta2) × dW^2
  4- Sdb = beta2 × Sdb + (1-beta2) × db^2
  5- vdw(corrected) = vdw/(1-beta1^n)
  6- vdb(corrected) = vdb/(1-beta1^n)
  7- Sdw(corrected) = sdw/(1-beta2^n)
  8- Sdb(corrected) = sdb/(1-beta2^n)
  9- w = w - alpha × (vdw(correct)/sqrt(Sdw(correct))
  10- b = b - alpha × (vdb(correct)/sqrt(Sdb(correct))
- Adam stands for adaptive moment algorithm where dw is the 1st moment and dw^2 is the second moment.
- Usually, beta 1 = 0.9, beta 2 = 0.999, epsilon = 10 ^(-8)
- Learning rate decay help us prevent oscillations around the minimum and never reaching the exact point
- It can be achieved by several methods as discussed below:
  1- alpha = 1/(1 + decay_rate× epoch_no) × alpha0
  2- alpha = (m)^epoc_no × alpha0 where m < 1
  3- alpha = k / sqrt(epoch_no) × alpha0
  4- discrete staircase decay
- Since we have many hyperparameters, the tuning process turns out to be tough unless we set up some rules for prioritizing them.
- When tuning the hyperparameters follow this pattern from higher priority to lower priority :
  1- learning rate alpha
  2- hidden units, mini batch size, moving average coefficient beta (0.9 preferable)
  3- #layers, learning rate decay
  4- Adam optimization algorithm parameters beta1 (0.9), beta2 (0.99), epsilon (10^(-8))
- When choosing hyperparameters avoid using grid and instead use random values to increase the number of choices you have.
- If found that the values for a hyperparameter in a certain neighbourhood are doing quite well then zoom in to that neighbourhood and sample values more densely. This is called coarse to fine search.
- To choose an appropriate scale for the hyperparameters, we must first understand the minimum and the maximum value from which we are sampling. So a linear scale might not always work.
- For example if we need to choose a value for alpha between 0.0001 and 1 and we sampled in a linear scale, probably we will sample from 0.1 to 1 and neglect the

region from 0.0001 to 0.1. so in that case we can have a log scale to guarantee equal chance of sampling values from the entire space.

- In log scale we determine the start point a = log(0.0001) and b = log (1) so now on the log scale our space becomes [-4,0], so when we sample we try one of the values in the new space denoted by r. So alpha is 10 ^ r.
- In case of exponentially weighted averages, when choosing the hyperparameter beta we also use the log scale. If beta is sampled from 0.9 to 0.999, then on a linear scale some values maybe ignored but if we instead sampled 1-beta we have values from 0.1 to 0.001
  So now on a log scale r is in the range [-3,-1]
- Depending on the computational power we have 2 approaches. The first one is babysitting one model and watching it day by day and see it have to increase or decrease one of the hyperparameters while the other one is training different models at the same time and see which one works better.
- Babysitting one model (panda), training different models (caviar).
- Batch normalization is used to speed up learning where Znorm = (Z - mue)/sqrt(variance + epsilon)
  so now all the activations of the hidden units are centered around 0 with standard deviation of 1.
- One problem here is that maybe it's desirable for the hidden units to have a non zero mean and different variance so in that case we define
  Z'norm = gama × Znorm + beta where gamma and beta are learnable parameters of the model.
- Beta and gamma help us change the mean and the variance of the hidden units so now we have explicit parameters that the model can tune as well.
- In batch normalization b or bias is no more a learnable parameter as it will be cancelled since we center the data around zero.
- For multiple classes classification, we use softmax layer to output the conditional probability of each class given the input vector x.
- In softmax layer the output Z is processed as follows:
  1- t = e ^ (Zl)
  2- a = ti/sum(ti), where i is the number of the output nodes in the last layer.
- The local optima means that in all the directions the curve of the cost function is bending down which can hardly happen in a high dimensional space.
- We're more likely to run into saddle points rather than local optima in high dimensional space where at one direction the curve is bending down while at another direction its bending up and normally this is a zero gradient point.
- So we are unlikely to get stuck in a bad local optimum.
- Plateaus can make learning very slow as the slope and the gradients are very small values but algorithms like rms prop and ADAM can help speed up the learning.