Mini-batch gradient descent

(SPEECH)
Hello,

(DESCRIPTION)
Text, Optimization Algorithms. Mini-batch gradient descent. Website, deep learning, dot, A.I.

(SPEECH)
and welcome back.

In this week, you learn about optimization algorithms that will enable you to train your neural network much faster.

You've heard me say before that applying machine learning is a highly empirical process, is highly iterative process.

In which you just had to train a lot of models to find one that works really well.

So, it really helps to really train models quickly.

One thing that makes it more difficult is that Deep Learning does not work best in a regime of big data.

We are able to train neural networks on a huge data set and training on a large data set is just slow.

So, what you find is that having fast optimization algorithms, having good optimization algorithms can really speed up the efficiency of you and your team.

So, let's get started by talking about mini-batch gradient descent.

You've

(DESCRIPTION)
New slide, Batch vs. mini-batch gradient descent. Text, Vectorization allows you to efficiently compute on M examples.

(SPEECH)
learned previously that vectorization allows you to efficiently compute on all m examples, that allows you to process your whole training set without an explicit formula.

That's why we would take our training examples and stack them into these huge matrix capsule Xs.

X1, X2, X3, and then eventually it goes up to X, M training samples.

And similarly for Y this is Y1 and Y2, Y3 and so on up to YM.

So, the dimension of X was an X by M and this was 1 by M. Vectorization allows you to process all M examples relatively quickly if M is very large then it can still be slow.

For example what if M was 5 million or 50 million or even bigger.

With the implementation of gradient descent on your whole training set, what you have to do is, you have to process your entire training set before you take one little step of gradient descent.

And then you have to process your entire training sets of five million training samples again before you take another little step of gradient descent.

So, it turns out that you can get a faster algorithm if you let gradient descent start to make some progress even before you finish processing your entire, your giant training sets of 5 million examples.

In particular, here's what you can do.

Let's say that you split up your training set into smaller, little baby training sets and these baby training sets are called mini-batches.

And let's say each of your baby training sets have just 1,000 examples each.

So, you take X1 through X1,000 and you call that your first little baby training set, also call the mini-batch.

And then you take home the next 1,000 examples.

X1,001 through X2,000 and then X1,000 examples and come next one and so on.

I'm going to introduce a new notation I'm going to call this X superscript with curly braces, 1 and I am going to call this, X superscript with curly braces, 2.

Now, if you have 5 million training samples total and each of these little mini batches has a thousand examples, that means you have 5,000 of these because you know 5,000 times 1,000 equals 5 million.

Altogether you would have 5,000 of these mini batches.

So it ends with X superscript curly braces 5,000 and then similarly you do the same thing for Y.

You would also split up your training data for Y accordingly.

So, call that Y1 then this is Y1,001 through Y2,000.

This is called, Y2 and so on until you have Y5,000.

Now, mini batch number T is going to be comprised of X, T and Y, T. And that is a thousand training samples with the corresponding input output pairs.

Before moving on, just to make sure my notation is clear, we have previously used superscript round brackets I to index in the training set so X I, is the I training sample.

We use superscript, square brackets L to index into the different layers of the neural network.

So, ZL comes from the Z value, the L layer of the neural network and here we are introducing the curly brackets T to index into different mini batches.

So, you have XT, YT and to check your understanding of these, what is the dimension of XT and YT?

Well, X is an X by M. So, if X1 is a thousand training examples or the X values for a thousand examples, then this dimension should be MX by 1,000 and X2 should also be an X by 1,000 and so on.

So, all of these should have dimension MX by 1,000 and these should have dimension 1 by 1,000.

To explain the name of this algorithm, batch gradient descent, refers to the gradient descent algorithm we have been talking about previously.

Where you process your entire training set all at the same time.

And the name comes from viewing that as processing your entire batch of training samples all at the same time.

I know it's not a great name but that's just what it's called.

Mini-batch gradient descent in contrast, refers to algorithm which we'll talk about on the next slide and which you process is single mini batch XT, YT at the same time rather than processing your entire training set XY the same

(DESCRIPTION)
New slide, Mini-batch gradient descent.

(SPEECH)
time.

So, let's see how mini-batch gradient descent works.

To run mini-batch gradient descent on your training sets you run for T equals 1 to 5,000 because we had 5,000 mini batches as high as 1,000 each.

What are you going to do inside the for loop is basically implement one step of gradient descent using XT comma YT.

It is as if you had a training set of size 1,000 examples and it was as if you were to implement the overall you are already familiar with but just on this little training set size of M equals 1,000 rather than having an explicit for loop over all 1,000 examples, you would use vectorization to process all 1,000 examples sort of all at the same time.

Let us write this out first, you implemented for a prop on the inputs.

So just on XT and you do that by implementing Z1 equals W1.

Previously, we would just have X there, right?

But now you are processing the entire training set, you are just processing the first mini-batch so that it becomes XT when you're processing mini-batch T. Then you will have A1 equals G1 of Z1, a capital Z since this is actually a vectorizing connotation and so on until you end up with AL, answer is GL of ZL and then this is your prediction.

And you notice that here you should use a vectorized implementation.

It's just that this vectorized implementation processes 1,000 examples at a time rather than 5 million examples.

Next you compute the cost function J which I'm going to write as one over 1,000 since here 1,000 is the size of your little training set.

Sum from I equals one through L of really the loss of YI and this notation for clarity, refers to examples from the mini batch XT YT.

And if you're using regularization, you can also have this regularization term.

Move it to the denominator times sum of L, Frobenius on the way makes it a square.

Because this is really the cost on just one mini-batch, I'm going to index as cost J with a superscript T in curly braces.

You notice that everything we are doing is exactly the same as when we were previously implementing gradient descent except that instead of doing it on XY, you're not doing it on XT YT.

Next, you implement that prop to compute gradients with respect to JT, you are still using only XT YT and then you update the weights W, read WL gets updated as WL minus alpha D WL and similarly for B.

This is one pass through your training set using mini-batch gradient descent.

The code I have written down here is also called doing one epoch of training and epoch is a word that means a single pass through the training set.

Whereas with batch gradient descent, a single pass through the training allows you to take only one gradient descent step.

With mini-batch gradient descent, a single pass through the training set, that is one epoch, allows you to take 5,000 gradient descent steps.

Now of course you want to take multiple passes through the training set which you usually want to, you might want another for loop for another while loop out there.

So you keep taking passes through the training set until hopefully you converge with approximately converge.

When you have a lost training set, mini-batch gradient descent runs much faster than batch gradient descent and that's pretty much what everyone in Deep Learning will use when you're training on a large data set.

In the next video, let's delve deeper into mini-batch gradient descent so you can get a better understanding of what it is doing and why it works so well.