Vectorization

(SPEECH)
>>

(DESCRIPTION)
Text, Basics of Neural Network Programming. Vectorization. Website, deep learning, dot, A.I.

(SPEECH)
Welcome back. Vectorization is basically the art of getting rid of explicit folders in your code.

In the deep learning era safety in deep learning in practice, you often find yourself training on relatively large data sets, because that's when deep learning algorithms tend to shine.

And so, it's important that your code very quickly because otherwise, if it's running on a big data set, your code might take a long time to run then you just find yourself waiting a very long time to get the result.

So in the deep learning era, I think the ability to perform vectorization has become a key skill.

Let's start with an

(DESCRIPTION)
New slide, What is vectorization?

(SPEECH)
example.

So, what is Vectorization?

In logistic regression you need to compute Z equals W transpose X plus B, where W was this column vector and X is also this vector.

Maybe there are very large vectors if you have a lot of features.

So, W and X were both these R and no R, NX dimensional vectors.

So, to compute W transpose X, if you had a non-vectorized implementation, you would do something like Z equals zero.

And then for I in range of X.

So, for I equals 1, 2 NX, Z plus equals W I times XI.

And then maybe you do Z plus equal B at the end.

So, that's a non-vectorized implementation.

Then you find that that's going to be really slow.

In contrast, a vectorized implementation would just compute W transpose X directly.

In Python or a numpy, the command you use for that is Z equals np.W, X, so this computes W transpose X.

And you can also just add B to that directly.

And you find that this is much faster.

Let's actually illustrate this with a little

(DESCRIPTION)
Jupiter is opened. The file is Vectorization demo.

(SPEECH)
demo.

So, here's my Jupiter notebook in which I'm going to write some Python code.

So, first, let me import the numpy library to import.

Send P. And so, for example, I can create A as an array as follows.

Let's say print A.

Now, having written this chunk of code, if I hit shift enter, then it executes the code.

So, it created the array A and it prints it out.

Now, let's do the Vectorization demo.

I'm going to import the time libraries, since we use that, in order to time how long different operations take.

Can they create an array A?

Those random thought round.

This creates a million dimensional array with random values.

b = np.random.rand.

Another million dimensional array.

And, now, tic=time.time, so this measure the current time, c = np.dot (a, b).

toc = time.time.

And this print, it is the vectorized version.

It's a vectorize version.

And so, let's print out.

Let's see the last time, so there's toc - tic x 1000, so that we can express this in milliseconds.

So, ms is milliseconds.

I'm going to hit Shift Enter.

So, that code took about three milliseconds or this time 1.5, maybe about 1.5 or 3.5 milliseconds at a time.

It varies a little bit as I run it, but looks like maybe on average it's taking like 1.5 milliseconds, maybe two milliseconds as I run this.

All right.

Let's keep adding to this block of code.

That's not implementing non-vectorize version.

Let's see, c = 0, then tic = time.time.

Now, let's implement a folder.

For I in range of 1 million, I'll pick out the number of zeros right.

C += (a,i) x (b, i), and then toc = time.time.

Finally, print more than explicit full loop.

The time it takes is this 1000 x toc - tic + "ms" to know that we're doing this in milliseconds.

Let's do one more thing.

Let's just print out the value of C we compute it to make sure that it's the same value in both cases.

I'm going to hit shift enter to run this and check that out.

In both cases, the vectorize version and the non-vectorize version computed the same values, as you know, 2.50 to 6.99, so on.

The vectorize version took 1.5 milliseconds.

The explicit for loop and non-vectorize version took about 400, almost 500 milliseconds.

The non-vectorize version took something like 300 times longer than the vectorize version.

With this example you see that if only you remember to vectorize your code, your code actually runs over 300 times faster.

Let's just run it again.

Just run it again.

Yeah. Vectorize version 1.5 milliseconds seconds and the four loop.

So 481 milliseconds, again, about 300 times slower to do the explicit four loop.

If the engine x slows down, it's the difference between your code taking maybe one minute to run versus taking say five hours to run.

And when you are implementing deep learning algorithms, you can really get a result back faster.

It will be much faster if you vectorize your code.

(DESCRIPTION)
Return to slide What is vectorization?

(SPEECH)
Some of you might have heard that a lot of scaleable deep learning implementations are done on a GPU or a graphics processing unit.

But all the demos I did just now in the Jupiter notebook where actually on the CPU.

And it turns out that both GPU and CPU have parallelization instructions.

They're sometimes called SIMD instructions.

This stands for a single instruction multiple data.

But what this basically means is that, if you use built-in functions such as this np.function or other functions that don't require you explicitly implementing a for loop.

It enables Phyton Pi to take much better advantage of parallelism to do your computations much faster.

And this is true both computations on CPUs and computations on GPUs.

It's just that GPUs are remarkably good at these SIMD calculations but CPU is actually also not too bad at that.

Maybe just not as good as GPUs.

You're seeing how vectorization can significantly speed up your code.

The rule of thumb to remember is whenever possible, avoid using explicit four loops.

Let's go onto the next video to see some more examples of vectorization and also start to vectorize logistic regression.