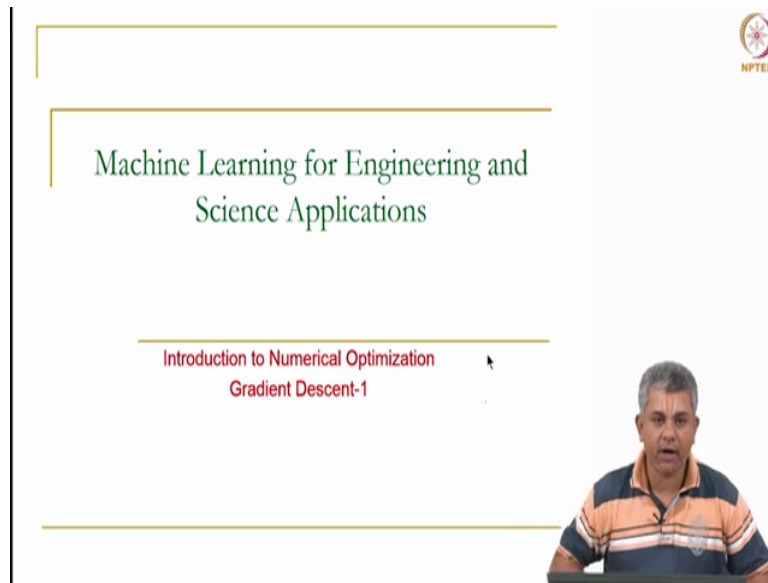


**Machine Learning for Engineering and Science Applications**  
**Professor Dr. Balaji Srinivasan**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Madras**  
**Introduction to Numerical Optimization Gradient Descent - 1**

(Refer Slide Time: 0:15)



In this video we will be looking at an introduction to numerical optimization. So far the optimization we had been looking at was essentially theoretical optimization we were just looking at analytical expressions. In this video we will see an introduction to how we can do the same thing numerically and specifically we will be looking at an algorithm called gradient descent which is sort of the work horse for most of deep learning.

(Refer Slide Time: 0:40)

The slide is titled "Need for Numerical Optimization" and features the NPTEL logo in the top right corner. It contains two 3D surface plots of functions  $f(x_1, x_2)$ . To the right, a diagram shows a black box with an input  $w$  and an output  $J(w)$ . A red arrow points to the box with the handwritten text "Analytical expression not known". Below the plots and diagram is a list of bullet points:

- Optimization we saw so far was analytical.
- This requires explicit expressions for the objective function in terms of the features (variables).
  - Example:  $J(w) = w_1^2 + w_2^2 + w_3^2 + 4$
- However, usually we only know the function as a "black" box.
  - In machine learning this "black box" is our Machine Learning Model (e.g. Neural network)
- So, we have to develop numerical (rather than analytical techniques)

A video inset in the bottom right shows a man speaking.

So why is it that we need numerical optimization? We were looking at shapes of this sort, so far what we were looking at was a case where suppose you have some  $f$  of  $x$  and let us say  $x$  is a vector and has two components  $x_1$  and  $x_2$ . In that case if you knew  $f$  of  $x$  as an analytical function of  $x_1$  and  $x_2$ , then you could use you know various ideas such as setting gradient of  $f$  equal to 0 and you have standard methodologies to find out what the appropriate minimum or maximum is.

However, most of the cases what happens is we do not have explicit expressions. So you do not really know what  $f$  is, so an explicit expression would be something of the sort  $J$  of  $w$  is  $w_1$  square plus  $w_2$  square plus  $w_3$  square plus 4. A small note for starting from this video I will start talking of optimization in terms of  $J$  and  $w$  because that is the notation we will ultimately use when we go to deep learning.

So usually what we know is the function only as a black box so that is some  $w$  comes in or some  $x$  comes in and some  $f$  comes out. So similarly some  $w$  comes in and some  $J$  comes out you do not really know analytical expression is unknown. So this is a proper black box, we will see a couple of sub cases of this later on in this video, but generally what happens is you know let us say  $x_1$  is 1,  $x_2$  is 2 and it suddenly tells you that  $J$  or  $f$  is 5.

Similarly, anytime you given  $x_1$  and  $x_2$  is able to give you a  $J$  or an  $f$ , but you still want to optimize it. In such case the methods that we use so far are not really usable. So in this case in the case of deep learning this black box is typically a neural network or something of that sort. So what we want to find out something that can deal directly with numbers rather than

with analytical expressions and that is why you need numerical optimization as against analytical optimization.

(Refer Slide Time: 2:55)

The slide is titled "Iterative optimization -- Fundamental idea" and features the NPTEL logo in the top right corner. A flowchart in the center shows a black box representing a function. An input  $w^{(k)}$  enters from the left, and an output  $J(w)$  exits to the right. From  $J(w)$ , an arrow points to  $\nabla_w J$ . A red feedback loop arrow starts from  $\nabla_w J$  and points back to the black box, indicating an iterative process. Below the flowchart is a list of bullet points:

- We want to drive  $\nabla_w J$  to  $\vec{0}$  but we do not have an analytical expression.
- Iterative Process
- Guess for  $w$
- Run through the black box and find value of  $J(w)$ 
  - This value may be obtained through a program instead of an expression
- Find  $\nabla_w J$ 
  - We will discuss methods for determining  $\nabla_w J$  numerically in later videos
- If  $\nabla_w J = 0$ , we stop, else we need to take a new guess
  - More precisely, improve our guess
- A very common method for improving guess is called **Gradient Descent**

In the bottom right corner of the slide, there is a small inset image of a man in a striped polo shirt looking at a laptop screen.

So here is a simple idea what you want to do is you want to drive the gradient of the function that you are trying to minimize or maximize to 0. So remember for this we are using the notation the function is  $J$  and the variable that we are optimizing over we are calling that  $w$ , so  $\text{grad } J w$ , we want this to go to 0 specifically the 0 vector because remember gradient is a vector, but we do not have an analytical expression for  $J$ .

So the iterative process is as follows, you take a guess so this is always the iterative process in anything not just optimization whichever variable you are trying to find out whether it is a linear system of equations, whatever system of equations you are trying to solve or optimizing you do not know  $w$  which is optimum. So you take a guess, okay. So here the super script  $k$  refers to the iteration number we will see a few examples later on in the slide.

So you take a guess, so you run through the black box this gives you a value this gives you a value of  $J$ , this might or might not be the optimal value, if you are a really good guesser you will automatically get right value but generally you will not, okay then you find out gradient of  $w$ . So now a question might arise if you only have a function as a black box how are you going to find out gradient of  $w$ , we will discuss this in the subsequent video but assume that you have a method of finding not only  $J$  but also gradient of  $J$  as a number.

Now suppose this gradient turns out to be 0 you stop if not you take a guess you take a new guess. Now how do you take a new guess? Will you guess this randomly? No it turns out that

there are specific methods to find out improved guesses, based on the  $w$  you got and the grad  $J$  you got you can actually get a better guess. So this method of improving your guess is what is called gradient descent.

(Refer Slide Time: 5:02)

**Gradient Descent (Scalar case)**

- Our task is to improve our guess for  $w$  such that we move from a region of higher gradient to a region of lower gradient
- For scalar (i.e. one component)  $w$ , this is easy
 
$$w^{new} = w^{old} - \alpha \left( \frac{dJ}{dw} \right)$$

So let us take the example of a gradient descent in a simple scalar case, simple scalar case like  $J$  of  $w$  where  $w$  is now a scalar a single number. So let us say it looks like this we did may not know what the specific function is? We just need to know that we are trying to get here. Now let us say this is the actual  $w$  optimum but your guess is this let me call it  $w_1$  or  $w_0$  this is our guess.

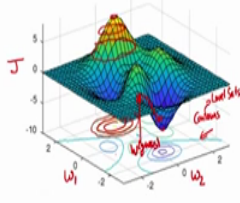
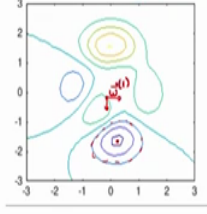
Now when you take this guess the  $J$  you get will be corresponding to this guess so this is  $J$ , this is  $w$  and we can automatically see that this is not optimum, why is this not optimum? Because at this point  $dJ/dw$  is not equal to 0. Now if you treat this as a game from this  $w$  you have two choices you can either move to the left or you can move to the right in order to improve your guess.

Now looking at this picture we automatically know that we have to move to the left, how do we know this? If you find out the slope at this point, so  $dJ/dw$  here is actually positive. If it is positive we know that 0 lies somewhere here, so you actually say  $w$  is  $w$  minus something, this something often is written as some  $\alpha$  (multiplying by) multiplied by the slope so that if you are here and the slope is negative you actually will go to the right so this is the simple idea behind gradient descent.

Our task is basically to improve our guess for  $w$ . For a scalar this is fairly straight forward, the new guess is the old guess minus alpha times  $(dJ/w) dw$  where alpha is an arbitrary parameter, okay it is a positive arbitrary parameter this parameter is often called the learning rate we will see that once again now.

(Refer Slide Time: 7:11)

### Gradient Descent (vector case)

- For the vector case, we rely on a theorem that says  
At any given point the **gradient gives the direction of steepest descent**
  - We will show a quick proof near the end of the video
- The general gradient descent algorithm is  

$$w^{new} = w^{old} - \alpha \nabla_w J$$
- $\alpha$  is called the **learning rate** is chosen by the user

So now let us take the more complicated vector case. So you have a whole manifold let us say once again you have  $J$ ,  $w_1$ ,  $w_2$  once again you guess we can see that the actual minimum is here, what is drawn here at the bottom are contours, recollect from our discussion of multi variable calculus the contours basically are collapsed. So if you think of this as a series of rings each of which is of constant value, if you collapse all of them this is the kind of contour that you will get, these contours are what are called level sets basically lines of equal value.

You would see this also in something like if you see the weather channel you will see this, you will see lines of constant pressure or lines of constant temperature which are moving around, so they are a representation of the function. So suppose here is just the contour drawn so on one line the value of  $J$  is constant. So we want to come to this point which is actually the minimum, but instead let us say I guess somewhere here this is my  $w$  guess so somewhere here is my guess the first guess is here.

Now I want to move in the right direction, now remember moving unlike 1D now is will be in two directions, okay. So the delta  $w$  or the change in  $w$  that you have give actually is a full vector, you have to say how much you want to move in  $x$  and how much you want to move in

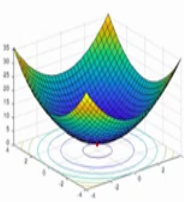
y direction, okay so you have to give both these. Luckily for us we have a nice theorem which says that we can move in the direction which decreases the maximum.

So for example if you are here you would like to move in this direction where the decrease is the sharpest so that you can think of this as a ball which is rolling downhill and you want to go as fast as possible to the bottom. So you would like to move in the direction where the change in J is the maximum or is the steepest and it turns out that the gradient gives exactly the direction that we are looking for, okay.

So we will show a quick proof of this actually not nearly at the end of this video but in the next video but we will show a quick proof of this. So the general gradient descent algorithm it turns out is a very simple generalization of the scalar case. The new w remember this is a vector is the old w minus alpha times grad J which is also a w, okay. So you take the steepest descent direction multiplied by a parameter just to adjust the size of the step and then move from there, this will become a little bit clearer as we go through a couple of examples alpha is a very important parameter call as I said earlier the learning rate this is something that we have to choose.

(Refer Slide Time: 10:30)

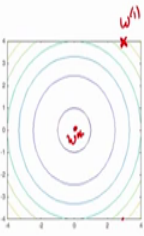
### Gradient Descent example



$$J(w) = w_1^2 + w_2^2 + 4$$

$$\nabla_w J(w) = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$

$\vec{w} = \vec{w} - \alpha \nabla_w J$   
 $w_1^{k+1} = w_1^k - \alpha (2w_1^k)$




Gradient Descent gives the iterative formula

$$w_1^{k+1} = w_1^k - \alpha (2w_1^k)$$

$$w_2^{k+1} = w_2^k - \alpha (2w_2^k)$$

We know that the actual minimum is at  $w = [0 \ 0]^T$   
 Let us start with an initial guess of  $w^0 = [3 \ 4]^T$   
 Let us see different cases for various choices of  $\alpha$   
 $\alpha = 2, 1, 0.1, 0.5$



So let us take an example let us take a very simple function for which we already know the minimum. So let us take the function J's w 1 square plus w 2 square plus 4 we know that the bottom is here the actual minimum is at 0 0, here are the contours which are drawn here, these are circles these are circles because J is a constant when w 1 square plus w 2 square is a constant which means these are circles centred at 0, okay.

So let us take the gradient of this analytical gradient is simply  $2w_1, 2w_2$  vector. So the iterative formula that we get for this is remember  $w$  vector new is  $w$  vector old minus alpha times grad  $J$  vector, this grad  $J$  has two components so which means  $w_1$  is  $w_1$  minus alpha times the first component which is  $2w_1$  as I have written before. I have denoted  $k$  plus 1 and  $k$  instead of old and new, so old I am calling  $k$ , new is called  $k$  plus 1 so that we can keep on iterating so start from the first guess to the second guess and so forth, okay.

Similarly, the second component also works the same way  $2w_2$   $k$  comes from the second component of the gradient so this is the iterative formula. We know that the actually minimum is at  $0, 0$  as we said, let us start with some random guess suppose we give a bad guess of 3, comma 4 so that on this curve is somewhere here so this is my first guess I want to go here this is the ideal  $w$  star.

So now how we can proceed is by actually choosing some value of this constant alpha this alpha so we will take 4 different choices just so that you can see a range of behaviours for what happens. So let us say alpha is we will start with alpha is 2, we will look at 1.1 and 0.5.

(Refer Slide Time: 12:48)

### Gradient Descent example

$$J(w) = w_1^2 + w_2^2 + 4$$

$$\nabla_w J(w) = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$

$$w_1^{k+1} = w_1^k - \alpha (2w_1^k)$$

$$w_2^{k+1} = w_2^k - \alpha (2w_2^k)$$

$w^0 = [3 \ 4]^T$      $\alpha = 2$

| Iteration (k) | $w^k$       | $\nabla_w J = 2[w_1 \ w_2]$ | $J(w^k)$ | $w^{k+1} = w^k - \alpha \nabla_w J$      |
|---------------|-------------|-----------------------------|----------|--|
| 0             | $[3 \ 4]$   | $[6 \ 8]$                   | 29       | $[3 \ 4] - 2 \cdot [6 \ 8] = [-9 \ -12]$ |
| 1             | $[-9 \ 12]$ | $[-18 \ 24]$                | 229      | $[27 \ 36]$                              |
| 2             | $[27 \ 36]$ | $[54 \ 72]$                 | 2029     | $[-81 \ 108]$                            |

Okay so let us start with alpha equal to 2 we are starting at 3 4 let us say alpha is equal to 2. So let us look at a simple table, so let us look at iteration 0 this is our initial guess, our  $w$  was 3, comma 4. I am not putting the transpose each time please understand that I am treating it as a row vector instead of column vector but works the same way. Now grad  $J$  is simply  $2w_1, 2w_2$  which is 6, comma 8,  $J$  the cost is 3 square plus 4 square plus 4 remember 3, comma 4 so this now comes to 29.

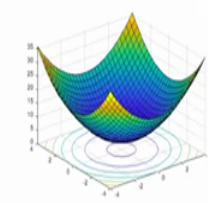
And now we can calculate  $w_{k+1}$  which is 3, comma 4 minus alpha which we choose to be 2 multiplied by 6, comma 8 and if you calculate it, it comes to minus 9 minus 12 so it has gone far away, okay so we started here and we have gone somewhere outside of the picture and you can see that this is actually not doing quite well I would like to come here but I have gone far away somewhere else but let us see how it goes further.

So suppose I start with minus 9 minus 12 and then proceed again using the same formula  $\text{grad } J$  is  $2w_1$   $2w_2$  which is minus 18 minus 24. If you calculate  $J$ ,  $J$  has actually increased ideally we would like  $J$  to always decrease this does not always happen in gradient descent but you can see that it has increased tremendously, okay. If you calculate  $w_{k+1}$  now it has come to 27 36 so all the way from here now you have gone somewhere else.

If you see in this picture you were you started somewhere here 3, comma 4 at this point and then we went somewhere far out, then we went somewhere else so we are actually going further and further away. So if I put 27 36, I see that my  $J$  has increased from 29 to 2029 and  $w$  has become worse. So this kind of process where intuitively we see that  $J$  is actually not coming down but is going further and further away from the actual solution even if you do not know the actual solution you can at least see that  $J$  is increasing so  $J$  our cost function is actually increasing and we are going at worse and worse places rather than better and better places. So this is a case which is a divergent case of alpha.

(Refer Slide Time: 15:35)

### Gradient Descent example

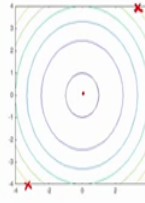


$$J(w) = w_1^2 + w_2^2 + 4$$

$$\nabla_w J(w) = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$


$$w_1^{k+1} = w_1^k - \alpha (2w_1^k)$$

$$w_2^{k+1} = w_2^k - \alpha (2w_2^k)$$



$w^0 = [3 \ 4]^T \quad \alpha = 1$

| Iteration (k) | $w^k$  | $\nabla_w J = 2[w_1 \ w_2]$ | $J$ | $w^{k+1} = w^k - \alpha \nabla_w J$     |
|---------------|--------|-----------------------------|-----|---|
| 0             | [3 4]  | [6 8]                       | 29  | $[3 \ 4] - 1 \cdot [6 \ 8] = [-3 \ -4]$ |
| 1             | [-3 4] | [-6 8]                      | 29  | [3 4]                                   |
| 2             | [3 4]  | [6 8]                       | 29  | [-3 -4]                                 |



So unhappy with this we try a slightly lower alpha it is always a good prescription to try lower alpha in case a higher alpha does not work. So once again we start here but instead of



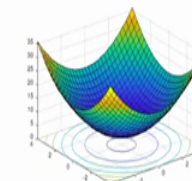
alpha equal to 2 if we use alpha equal to 1 we get minus 3 minus 4 which atleast seems a little bit better. So you started here 3 4 and we came here minus 3 minus 4 you wanted to come here maybe hopefully we will come back there.

So we now put minus 3 minus 4 the corresponding grad J is minus 6 minus 8, J unfortunately has not decreased because it is  $w_1$  square plus  $w_2$  square and if you try and find out what  $w$  at  $k+1$  is which is minus 3 minus 4 minus 1 times minus 6 minus 8 you will get 3 4, so now you are back here. So now we are sort of stuck in a cycle, it basically oscillates between two points 3 4 minus 3 minus 4, 3 4 so on and so forth it just goes back and forth, J does not decrease at all in fact in this case.

So this case is also not useful for us because we would actually like to systematically come towards the actual minimum so this is an example which does not converge at all.

(Refer Slide Time: 16:58)

### Gradient Descent example

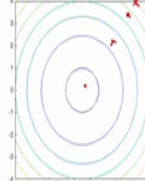


$$J(w) = w_1^2 + w_2^2 + 4$$

$$\nabla_w J(w) = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$

$$w_1^{k+1} = w_1^k - \alpha (2w_1^k)$$

$$w_2^{k+1} = w_2^k - \alpha (2w_2^k)$$



$w^0 = [3 \ 4]^T \quad \alpha = 0.1$

| Iteration (k) | $w^k$           | $\nabla_w J = 2[w_1 \ w_2]$ | J      | $w^{k+1} = w^k - \alpha \nabla_w J$         |
|---------------|-----------------|-----------------------------|--------|---|
| 0             | [3 4]           | [6 8]                       | 29     | $[3 \ 4] - 0.1 \cdot [6 \ 8] = [2.4 \ 3.2]$ |
| 1             | [2.4 3.2]       | [4.8 6.4]                   | 20     | [1.92 2.56]                                 |
| 2             | [1.92 2.56]     | [3.84 5.12]                 | 14.24  | [1.536 2.048]                               |
| 30            | [0.0007 0.0008] | ...                         | 4.0000 | ...   |

Let us take a third case which much smaller alpha which is 0.1, okay. So if we go through the exercise now all that has changed from the previous two examples is the alpha that I have put which has become 0.1 and you see that this has become slightly better now so you have come to 2.4 3.2 somewhere here so we have got a little bit better atleast it looks promising and we can now check what happens as we do future calculations.

You will also notice that slowly now instead of either increasing or getting stuck at the same point J is actually decreasing. So I would recommend that you do this exercise yourself you will also see one such example problem being given in the assignments but you can see now

that from 1.9 to 2.5 it has come to 1.5 2.0 which is a little bit better once again okay so getting somewhere here so slowly we are approaching the origin.

Now if you keep on repeating the exercise so this is now the 30th iteration not just the second iteration so we wrote a code and if you see the 30th iteration you will see that it is actually getting quite close to 0, the cost has actually come very close to the minimal cost, why is 4 the minimal cost? If  $w_1$  and  $w_2$  were 0, the actual cost would be 4. So you are actually converging slowly and we have come somewhere here over the 30th iteration.

Now a couple of things are worth nothing here one is that we have not actually come to the total minimum 0 0 and in fact if you use alpha equal to 0.1 you will never really come there because it will only keep on multiplying by small factors there is no way that you can get 0 0 out of this. So you are actually only slowly converge, theoretically it will take infinite iterations in order to get to 0 0 but numerically we know that below machine epsilon it will anyway stop.

So if you need to find out the absolute minimum where you know your grad J goes to 0 you might actually need infinite iterations which is why we actually need a stopping criteria, we need to say something like okay I am happy with two decimal places of accuracy, we will see how to do that in the next video.

(Refer Slide Time: 19:12)

### Gradient Descent example

$$J(w) = w_1^2 + w_2^2 + 4$$

$$\nabla_w J(w) = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$

$$w_1^{k+1} = w_1^k - \alpha (2w_1^k)$$

$$w_2^{k+1} = w_2^k - \alpha (2w_2^k)$$

$w^0 = [3 \ 4]^T \quad \alpha = 0.5$

Converges rapidly  $w = w - \alpha \nabla J$


| Iteration (k) | $w^k$ | $\nabla_w J = 2[w_1 \ w_2]$ | J  | $w^{k+1} = w^k - \alpha \nabla_w J$<br><span style="color: red; font-size: small;">[0 0] - 0.5 * [6 8] = [0 0]</span> |
|---------------|-------|-----------------------------|----|---|
| 0             | [3 4] | [6 8]                       | 29 | [3 4] - 0.5 * [6 8] = [0 0] ✓   |
| 1             | [0 0] | [0 0]                       | 4  | [0 0]   |
| 2             | [0 0] | [0 0]                       | 4  | [0 0]   |

In the meantime, let us look at another alpha, alpha equal to 0.5 by now you should be familiar with the whole process. So if we put alpha equal to 0.5 you actually get 0 0 right at the first step. So you start here, we come here. Now what happens to the algorithm once it

comes to the right minimum? So if you come to 0 0 note that grad J is also 0 0 because this is the actual minimum, J is 4 of course and  $w_k$  plus 1 is 0 0 because it is 0 0 minus alpha times 0 0 so it is just there.


So in all future iterations it will always stay at 0 0, so this is an advantage at gradient descent because you have  $w$  is equal to  $w$  minus alpha grad J, the moment grad J goes to 0 you will actually  $w$  will stop there, of course you can have this at a false minimum something like a saddle point also it can get stuck, but we will see cases of that sort in the coming weeks. The important thing here is alpha equal to 0.5 actually converges quite rapidly.

(Refer Slide Time: 20:30)



### Some lessons from the example

- It is possible for the gradient descent algorithm to
  - Diverge ( $\alpha=2$ )
  - Oscillate without diverging or converging ( $\alpha=1$ )
  - Converge slowly ( $\alpha=0.1$ )
  - Converge rapidly ( $\alpha=0.5$ )
- All these behaviors can manifest for the sample example depending on the learning rate  $\alpha$
- The choice of  $\alpha$  is part of algorithm design
- $\alpha$  is a *hyperparameter* – a parameter that must be set before learning begins

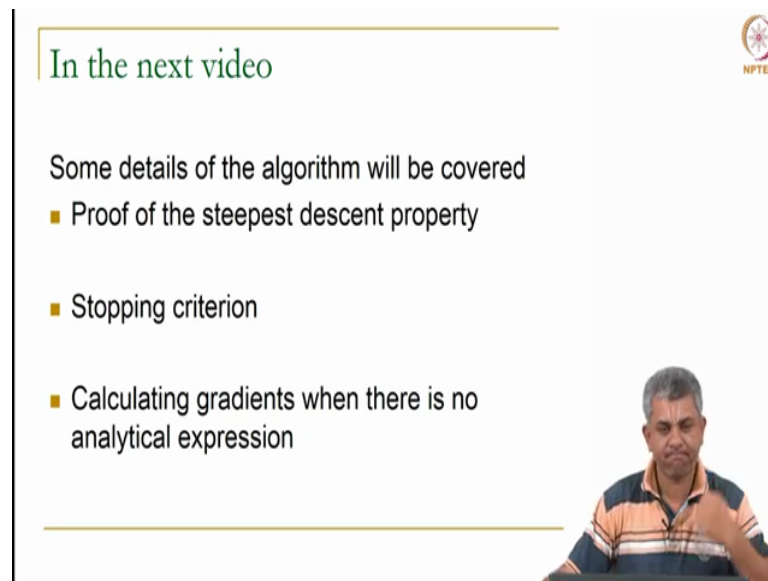


So what we have seen so far is that it is possible for the gradient descent algorithm to either diverge which we saw for alpha equal to 2, or it could oscillate without diverging or converging, it could converge slowly which we saw with alpha equal to 0.1 or it could converge quite rapidly which happened with alpha equal to 0.5, okay. In practical algorithms you will probably never see a case such as alpha equal to 0.5 where in one step you are going to get to the right answer, okay so that will almost never happen.

But typically you are going to see some manifestation of either slow convergence or fast convergence. So all this depend on the learning rate alpha, part of algorithm design what you will have to do as a user is to choose the right alpha. There are methods which have some variations on this which we will discuss in the coming weeks, but alpha is what is called a hyper parameter, okay.

A hyper parameter is a parameter that must be set before your learning algorithm actually starts, okay so even before you actually learn you will actually have to set some parameters alpha is just one such example. In fact an open problem typically in neural network and deep learning research is what is called calculation of hyper parameters, okay so design of hyper parameters and coming up with optimal hyper parameters.

(Refer Slide Time: 21:54)



The slide features a title 'In the next video' in green text at the top left. Below the title, there is a list of three bullet points: 'Some details of the algorithm will be covered', '■ Proof of the steepest descent property', '■ Stopping criterion', and '■ Calculating gradients when there is no analytical expression'. The NPTEL logo is in the top right corner. A small video inset of a man speaking is in the bottom right corner.

In the next video

Some details of the algorithm will be covered

- Proof of the steepest descent property
- Stopping criterion
- Calculating gradients when there is no analytical expression

In the next video what we will see is some of the details of gradient descent. For example, we will see a proof of the steepest descent property the fact that the gradient represents the direction of steepest descent. We will also look at the point that I mentioned briefly for the alpha equal to 0.1 case which is you need to decide when to stop the algorithm, you will never get actually to full minimum but you need to decide when to stop, okay.

And the third thing which we have to find out is finding out how to calculate gradients when there is no actual analytical expression for J available. So these are the three issues that we will be discussing in the next video, thank you.