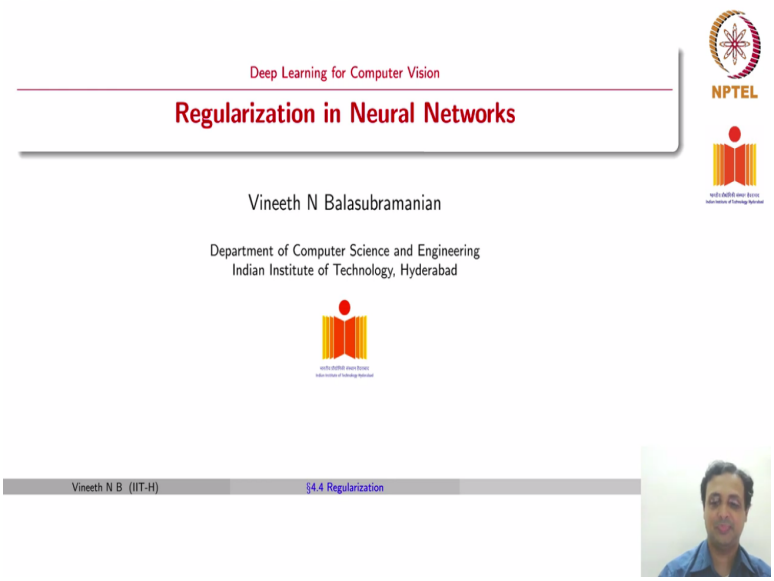
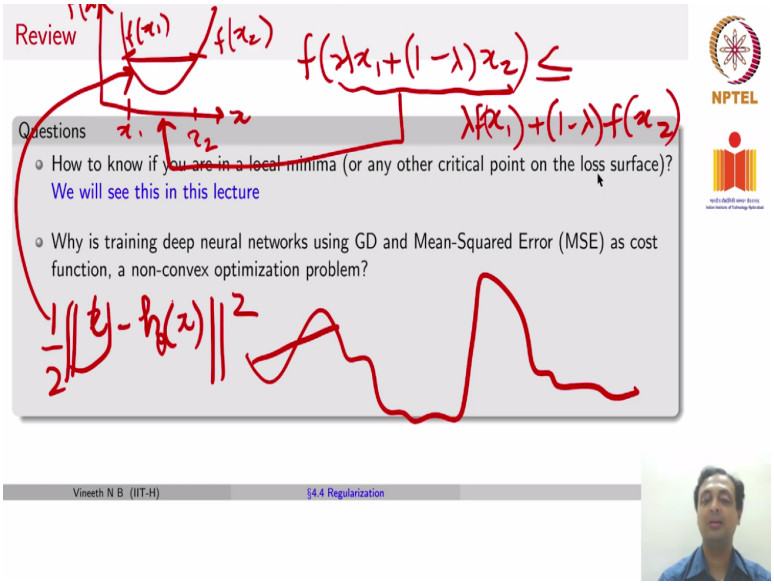
Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 27
Regularization in Neural Networks

(Refer Slide Time: 00:14)



In this lecture, we continue and talk about Regularization in Neural Networks, which is an important topic to obtain good test set performance with these models.

(Refer Slide Time: 00:33)



Before we start, let us review some of the questions we left behind in the previous lecture. The first question was, how to know if you are in a local minimum, or any other critical point on the loss surface, when you train Neural Networks. All the algorithms that we spoke about so far, backpropagation Gradient Descent, all the variants of Gradient Descent always had this clause in its while loop put follow, which said until convergence or until stopping criterion is achieved.

What should be the stopping criterion? Hope you had a chance to think about it. We are not going to talk about the answer now. But we are going to talk about it over the course of this lecture a little later in this lecture. The second question that we left behind was why is training deep Neural Networks using Gradient Descent and Mean Squared error as cost function, a non convex optimization problem?

I think we should probably spend some time here. Hope you tried to answer this question. And you have a good understanding of convex and non convex functions, but let us very briefly review that before answering this question. So, remember, this is a simple example of a convex function and a convex function says that if your x axis was here, and here was your $f(x)$ axis, a convex function says the definition says that if you have a point x_1, and if you have a point x_2, this is $f(x_1)$ and this is $f(x_2)$.

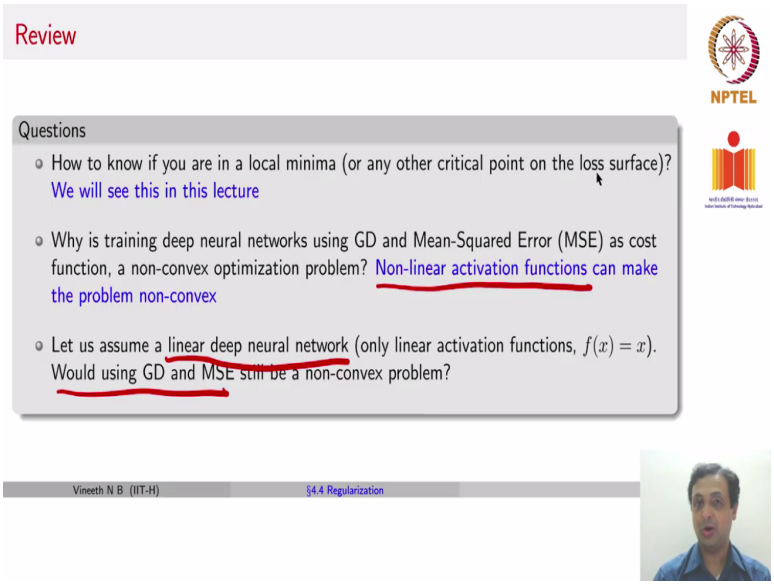
Then all that we say is that this line connecting $f(x_1)$ and $f(x_2)$ will always lie above the curve. So, mathematically speaking, $f(\lambda x_1 + (1 - \lambda)x_2) <= \lambda f(x_1) + (1 - \lambda)f(x_2)$

That is the definition of a convex function. So, clearly, a non-convex function is one which is not convex. So, a non-convex function as an example could be a surface something like this. So, you can clearly see that you will find lines such as these for instance, where the curve does not necessarily lie below the line at all times. Now, let us come back to our question as to why is training deep Neural networks using Gradient Descent and Mean Squared Error as cost function, a non convex optimization problem?

One thing that you have to first understand here before answering this question is the nature of the cost function Mean Squared error itself. Remember, Mean Squared Error you all know is given by let us just take let us use the same, let us assume that y was the label and say $h(x)$ was the output of the Neural Network, Mean Squared Error tries to minimize the square of these terms and of course, you take a mean, half can be added for simplicity, you of course, take a mean of these quantities, quantities, that is Mean Squared Error.

Now one look at it, you can say that Mean Squared Error will be a Convex function, it is is a quadratic function, which means it could be very similar to what we drew here and it could turn out to be a convex function. Now, in this context, let us ask this question, then why is training Deep Neural Networks using Gradient Descent and Mean Squared Error, a non convex optimization problem.

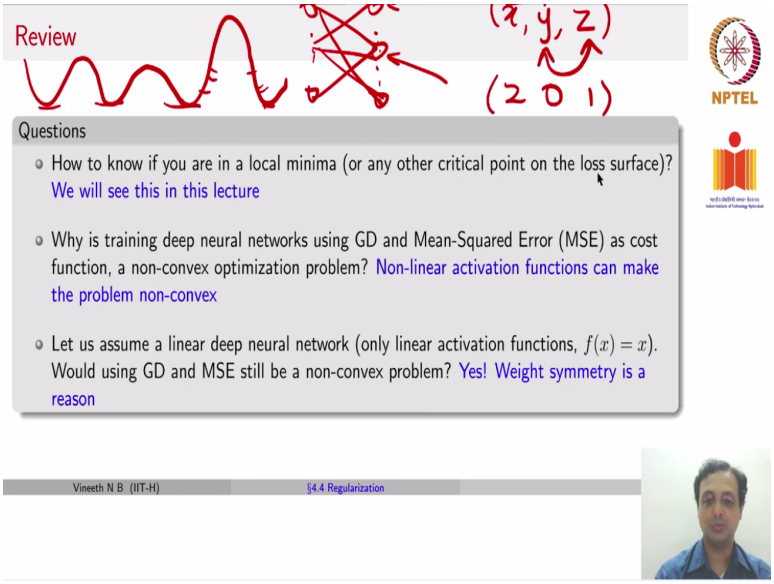
(Refer Slide Time: 05:27)



The reason is you could have nonlinear activation functions in your Neural Network that can make the problem non convex with respect to the weights, okay, then your counter question is okay, let us assume now a linear deep neural network, where you do not use any nonlinear activation function at all, no sigmoid, no tanh, no relu so on and so forth, you only use linear activation functions, which is as good as saying no, no additional activation function other than the Neural Network itself.

Now, would training a Neural Network using Gradient Descent and Mean Squared Error be a convex optimization problem or non convex optimization problem? From the answer I said earlier, it should be a convex optimization problem, because the cost function is Mean Squared Error. And I just said that nonlinear activation functions are what could make the problem non convex. So, which means, if you have linear activations, it should be convex.

(Refer Slide Time: 06:35)



But the answer is no, it is still not convex. Or other Yes, for non convexity? A reason for that is the phenomenon of Weights Symmetry. There are many other symmetries in a Neural Network. But Weights Symmetry is at least one reason. What does this mean? What does Weight Symmetry mean? Remember that, when we talk about an error surface, we have to give an index to each weight in the Neural Network, that inner surface has to have a certain dimensionality to it.

So, the first dimension would be the first weight in the first layer connecting to the first weight in the second layer. The second dimension would be that first weight in the first layer, connecting to the second weight, the second layer, so on and so forth. That is how you index all your weights, and you build your error surface. Now, let us take any two layers in a Neural Network.

And let us connect them using some set of weights. It is obviously fully connected. But these are some set of weights. Now, in a trained Neural Network, what I could do now is take this neuron here, and this neuron, neuron, in the second layer, and simply swap them, and also swap all the weights coming into those neurons. Once again, you have a trained model, you are no longer learning it, you are fully trained.

But what I am going to do now is take a couple of these neurons and simply swap them and also swap all the weights coming into these neurons, will the output change? No, because the weights

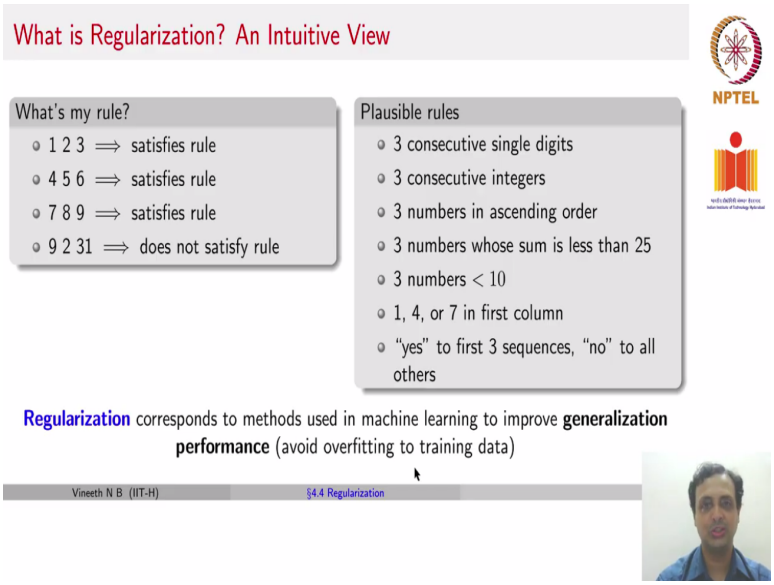
are the same, it simply changed the ordering of the weights in one of the layers. But on the error surface, this corresponds to a very different position. Let us take an example of three dimensions. Let us say we had x dimension, y dimension and z dimension. And let us say we had values which are 2, 1 and 0 in these three dimensions. What we have d1 now is swapped, and Y. That is what we did for some of the neurons and hence some of the weights.

(0, 1, 2), (2, 1, 0) and (2, 0, 1) are different points in three dimensional space. Similarly, for a Neural Network, if you swap neurons, after you swap these points are different weight configurations, because of the index to construct the error surface. But because of the symmetry of the Neural Network, they would give exactly the same output, exactly the same error.

Which means these two weight configurations, as I said, we take a train model, which means it is already at a critical point. Let us see a minimum. We are not saying that here is another weight configuration, which I am giving you which also will give you the same minimum which is going to be far away because (2, 1, 0) and (2, 0, 1) are not necessarily close by in the grid, there are many points in between (2, 1, 0) and (2, 0, 1), if you plotted them in three dimensional space.

Which means there is another point further away, which also has the same minimum value of error and this is just one swap of a neuron. For a Neural Network, assuming you have a million neurons, you could swap a million times. So, which means there are going to be many similar weight configurations, which all have the same error, which clearly tells us that the error surface must be non-convex. It is something like this. All these points have the same error, which can happen only if your error surface is non convex, because there are many other points in between those minimums.

(Refer Slide Time: 11:07)



Let us move on to Regularization, which is the focus of this lecture. Let us first start with an Intuitive understanding of Regularization. Most of you may have perhaps covered this in an introductory Machine Learning course, but let us briefly review this, and then talk about how Regularization is done in deep Neural Networks or Neural Networks.

Let us take a very simple example. Let us take these sets of values 1 2 3, we say that it satisfies some rule 4 5 6, it satisfies the same rule 7 8 9 also satisfies the same rule. 9 2 31 does not satisfy the rule. The question for you now is what is the rule? If you observe carefully, this is very similar to the Machine Learning setting that we have. Imagine now that these are all data points.

And we are given in training data that certain data points have a certain label and certain other data points have another label. And our job is to find out what is the model that tells us whether something satisfies plus 1 or plus 2. Coming back to this example, what could be the rule here, which satisfies these first three, and does not satisfy the set of values. A simple thought process should tell you that there are many possibilities.

You could say three consecutive single digits. You could say three consecutive integers, you could say three numbers in ascending order, three numbers whose sum is less than 25, three numbers less than 10. It should have 1 4 or 7 in the first column, or for all you know, you can

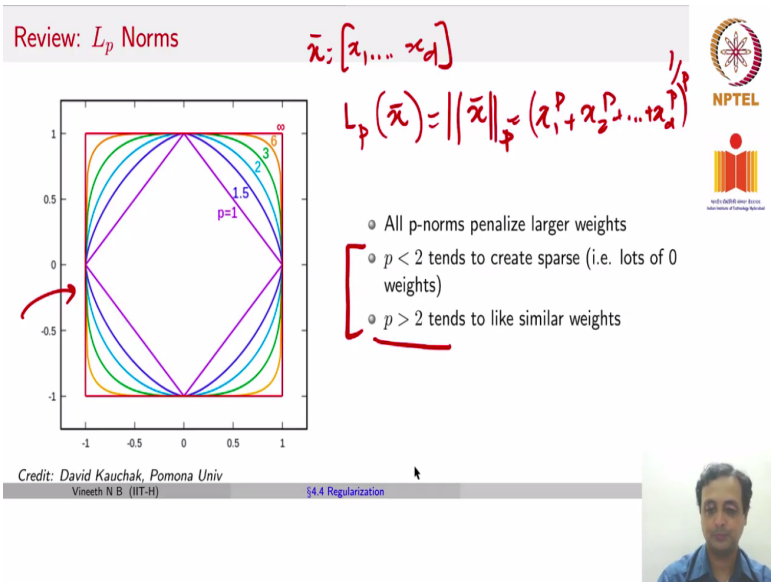
just say, say Yes to the first three sequences, and No to all others, which is also correct in this particular case.

Clearly, the last two cases would not have been the answers that you came up with intuitively. Why is that so? The reason is, even as humans, we are always looking for a rule that can generalize well if I gave you newer data points. I never even mentioned to you at this time that I am going to give you newer data points. But still, it is human tendency to always come up with rules that can generalize well to data that we may see tomorrow.

And this is exactly what we want to do with Neural Networks, or Machine Learning in general. And we call this process Regularization, where we use methods and Machine Learning to improve generalization performance and avoid overfitting to training data. So, we do not want to necessarily come up with a rule that fits only the training data.

But we want to come up with a rule that can do well, maybe on data that we will see tomorrow, which is what we mean by test set performance or generalization performance. Sometimes even at the cost of not getting hundred percent accuracy on your training data set. Let us see now how we can do this with Neural Networks.

(Refer Slide Time: 14:36)



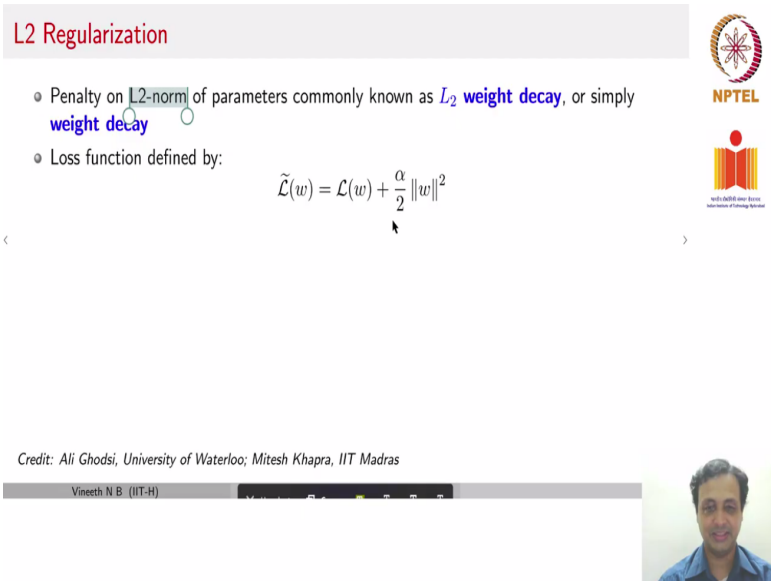
We need one of the concepts to be able to follow some of the discussions in this lecture, which is the concept of L_p norms. Hopefully you are all aware of this, but let us quickly review them and

move forward. Remember, L_p norm is formally written as $L_p(\bar{x})$. Let us assume that x is a vector of the d dimensions $[x_1, x_2, \cdots, x_d]$. $L_p(\bar{x})$ is given by or also denoted by $||\bar{x}||_p$ is given by $(x_1^{\ p} + x_2^{\ p} + \cdots x_d^{\ p})^{1/p}$.

You can make out that if you put p as 2 you'll get your standard Euclidean, Euclidean norm or L2 norm. Similarly, you can have L_0 norm, L_1 norm and so on and so forth till L_∞. You can actually have to be any number for that matter any, any, any positive number for that matter, non negative number. So, you see here diagrammatically you can see that this is how the unit ball of each of these L_p norms look like.

If you took the unit ball of L_1 norm, it looks like a rhombus, the unit ball of L_2 norm will look like a circle and L_∞ norm will look like a square, L_∞ norm given by max of a value max of the vector, max element of that, of that vector. This is important because this helps us understand certain aspects of Regularization that we will visit. So, a couple of points to keep in mind here is that when p is less than 2, it tends to create sparse weights. And when p is greater than 2, or greater than or equal to 2 for that matter, it tends to create similar ways. We will see why this is the case in some time from now.

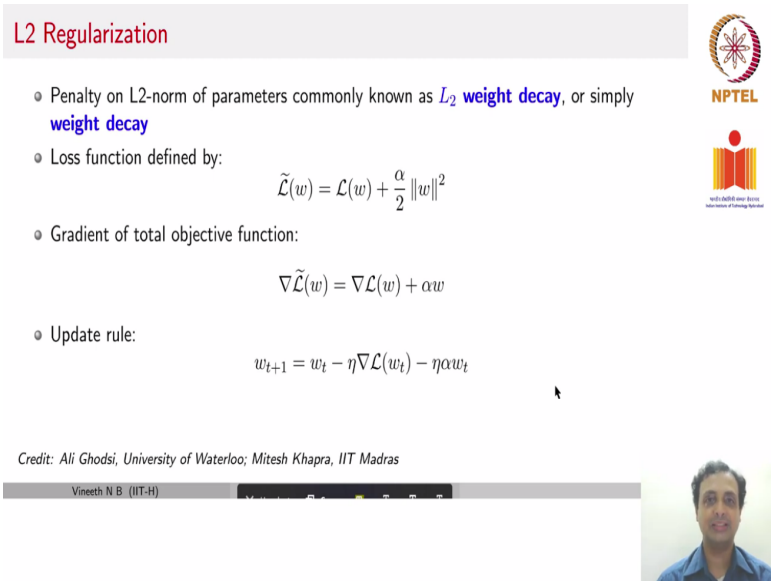
(Refer Slide Time: 16:55)



The most popular Regularization method for training, not just Neural Networks, even other Machine Learning models is known as L_2 Regularization which imposes a penalty on the L_2 norm of the parameters, which is the reason it is also known as L_2 weight decay, or simply weight decay. In this case, the loss function that we have for a Neural Network is upended by another term, which has the L_2 norm of the weights.

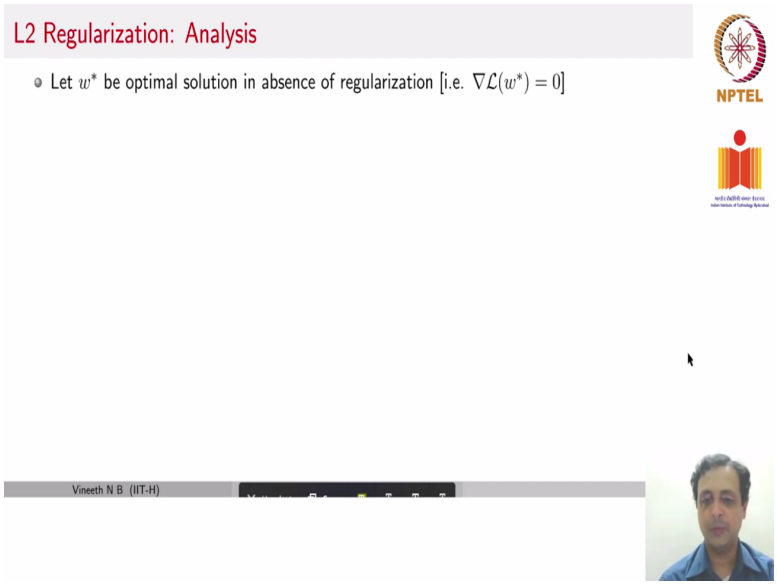
This loss function can be any loss function used to train your Neural Network. So far, we have seen Mean Squared Error, so you can assume its Mean Squared Error. But in future, we will see other loss functions that you can use to train your Neural Network. And the L_2 penalty is added to any of those loss functions. So, the α here denotes how much importance you want to give to penalizing the L_2 norm of the weights. And the by 2 here is for mathematical simplicity.

(Refer Slide Time: 18:08)



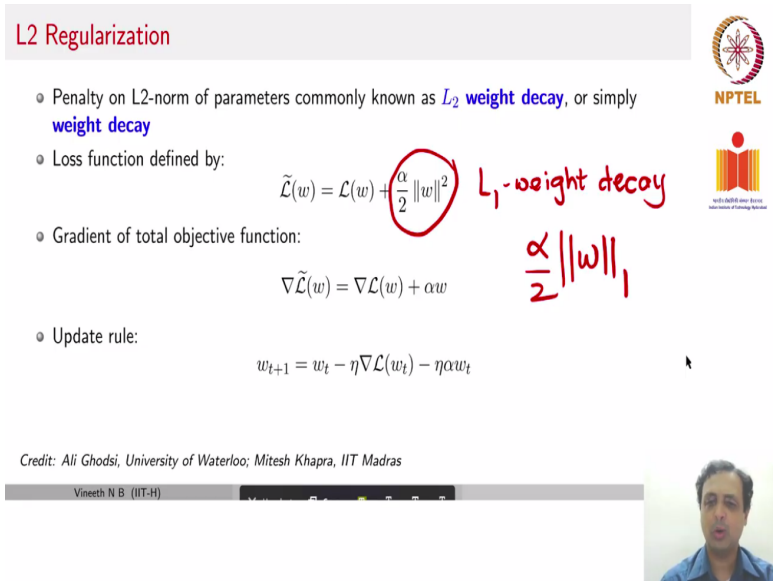
So, the gradient now of the total objective function will look like $\nabla\widetilde{L(w)} = \nabla L(w) + \alpha w$. This new objective function with the L2 weight decay term is equal $\nabla L(w)$, which was your original loss function, plus αw. And you differentiate this with respect to w. 2 and 2 get cancelled, and you are left with αw. So, your Gradient Descent update rule is going to look like $w_{t+1} = w_t - \eta\nabla L(w_t) - \eta\alpha w_t$. That is going to be your new Gradient Descent rule for your regularized loss function.

(Refer Slide Time: 19:07)



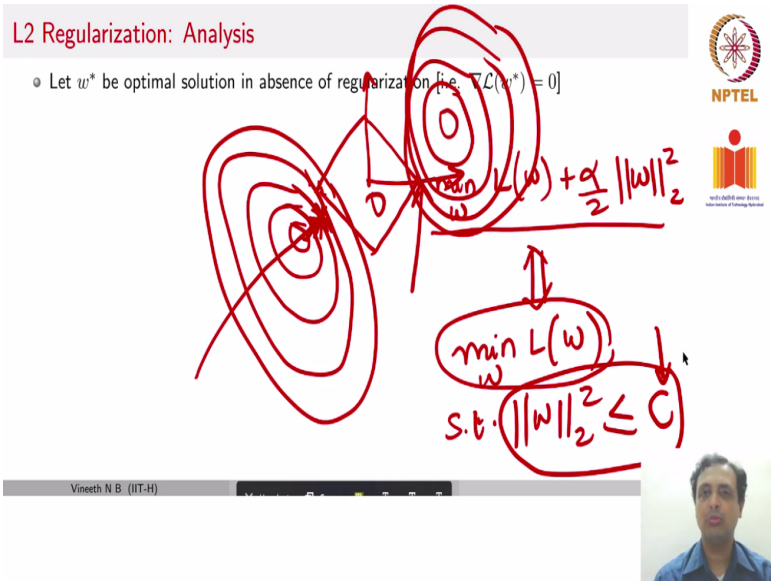
Let us do some analysis of this Regularization to understand what is really happening. Let us first talk about it conceptually, intuitively, and then we will go over one mathematical way of looking at what is going on in Regularization. We just said some time back that the shape of the L_2 norm ball can help you understand what you are doing with Regularization. Let us let me add one point here.

(Refer Slide Time: 19:40)



That you could also minimize other norms of w in this term. You could have L_1 weight decay which is also a common approach which enforces sparsity in weights in the Neural Network. By sparsity we mean many of the weights in the Neural Network will be forced to go to 0, in L_1 weight decay, we would say this is going to be with respect to L_1 norm. And remember the L_1 norm of any vector is the sum of the absolute values in that vector. This is also possible.

(Refer Slide Time: 20:19)



Let us try to see what would be the effect on these different cases. Remember, in the last lecture, we talked about contour plots. So, you have these contour plots of your error surface with respect to the weights, piece of contour plots such as these, if you recall, which will cross sections of your error surface something like this. This is the contour plot of your error surface, which is the term without a regularizer.

So, when we add the regularizes, what are we doing, we are now saying you want to minimize over the set of weights, your loss with respect to the weights, as well as some constant times you are 2 norm of the weights or 1 norm of the weights in certain cases, 2 norm of your weight, weight square.

This is equivalent to saying that you want to minimize your loss function such that your 2 norm of your weights is less than a certain quantity, you can come up with some number, there is some constancy. These formulations are equivalent to a certain degree under certain conditions. From

an optimization standpoint, why are we saying this? What this means now is the minimum of this new object to function is equivalent to the minimum of this original objective function subject to the 2 norm being lesser than some value.

If you recall, this is nothing but a norm ball of the weights. And it is not the unit normal ball now, it is a C norm ball that a unit norm ball is when the 2 norm of the weights lies within 1 or all the weights lie within, within a norm of 1 around the, around the center, around the origin. Now, we just C norm ball as a measure C, which should be less than. So, which means this, the solution here will be the intersection of let us assume that the C norm ball was something like this.

Let us assume that that is where the origin was. Let us assume that here is the origin and the C norm ball is somewhere here. You can see now the C norm ball intersects the error surface at some point, that is now going to be the minimum of this new function, not the point at the center, because that is outside the C norm ball. So, you have to find a minimum somewhere here, which is 1, which is minimum for both the loss and lies within the C norm ball, which is what you are trying to do.

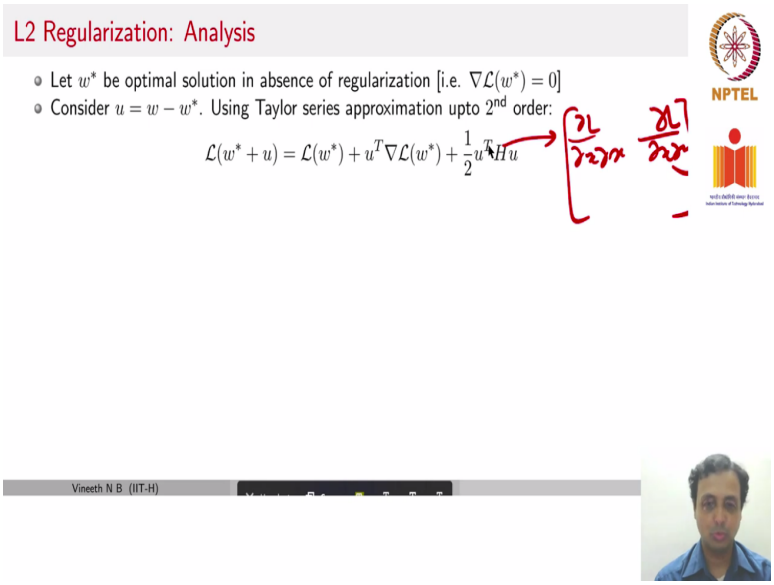
So, when you have your L_2 norm, your norm ball is going to look like a sphere, and you intersect with the error surface and you get a certain solution. However, if you had L_1 weight decay, then in that particular case, your norm ball is going to be a rhombus, not a square not, not a sphere. So, a rhombus has sharp edges, which means it is very likely that you will find minimum along points where 1 of the axis is 0.

You can probably see this in a slightly different way to let us try to draw the error surface elsewhere just to make a point clear. So, we want to find the intersection of the error surface and the 1 norm ball. And the intersection will typically happen at one of these corners. And what happens at one of these corners in L_1 norm ball, the other weight is going to be 0. And in a high dimensional rhombus, there may be many weights that go to 0.

And that is why enforcing L_1 weight decay will enforce sparsity in your weights. Both are valid regularizers. In one case, you are reducing the 2 norm of the weights. In the other case, you are

reducing 1 norm of the weights but the effect can be slightly different. Let us now also see it as to what we are trying to do from a mathematical perspective.

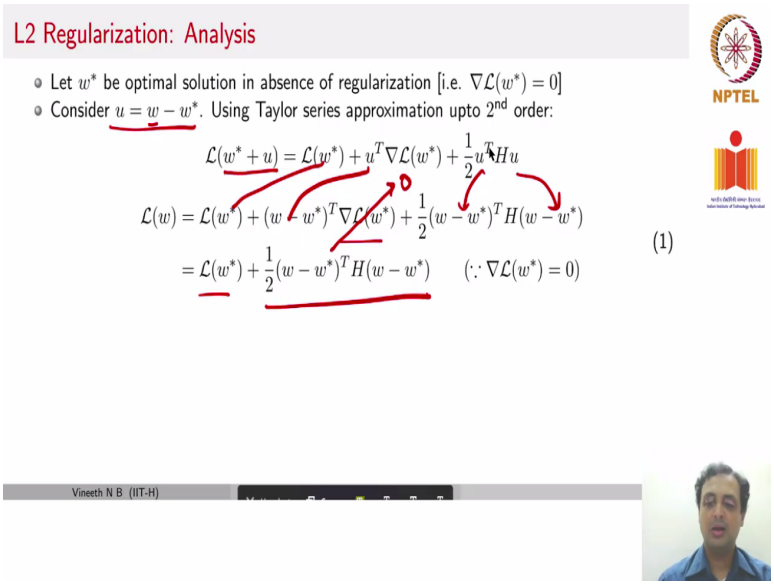
(Refer Slide Time: 24:49)



Let us assume that we have an optimal solution for your original problem. That is called that w^* So, which means $\nabla L(w^*) = 0$. So, w^* was our optimal solution before Regularization, only with the original loss function. Let us now consider a term $u = w - w^*$, let us write a Taylor series expansion of loss function at $w^* + u$.

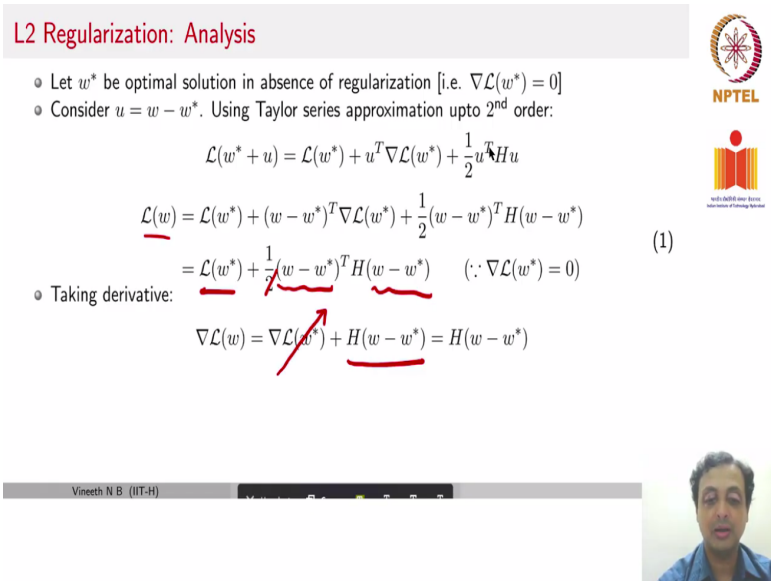
By Taylor series expansion this is given by $L(w^*) + u^T\nabla L(w^*) + 1/2(u^THu)$, where H is the Hessian of the loss with respect to your weights, what is the Hessian, the matrix of all second partial derivatives you can have matrix H is a matrix of second partial derivatives you can have $\partial^2 L/\partial x^2$ or $\partial^2 L/\partial x\partial y$, so on and so forth. If you had many more weights, you can have that many that is the size of your Hessian. The other if you had 1 million weights, the size of your Hessian will be 1 million times 1 million. It is definitely a large matrix to compute.

(Refer Slide Time: 26:19)



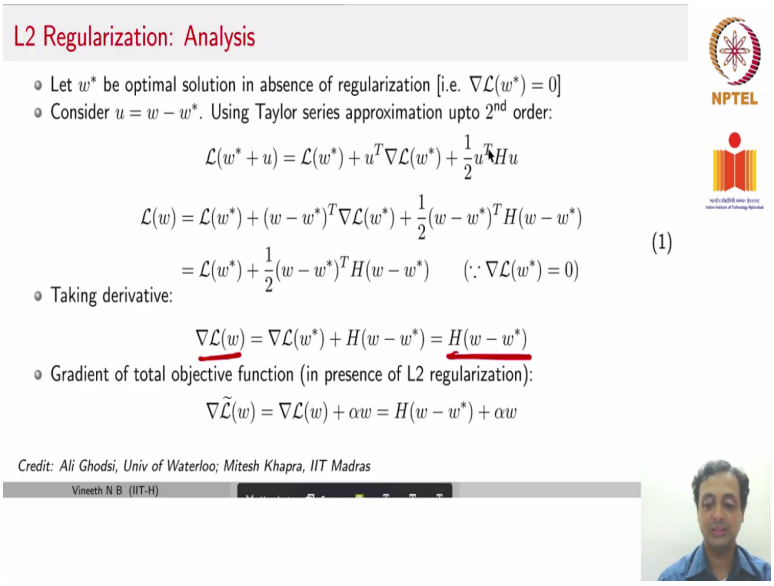
But let us move on with this mathematical discussion. So, which means $L(w^* + u)$ is actually w. So, I can say $L(w) = L(w^*) + (w - w^*)\nabla L(w^*) + 1/2(w - w^*)^T H(w - w^*)$. We now know that $\nabla L(w^*) = 0$, because that is our assumption of what w^* is, it is an optimal point for your original loss function. Which means we are left only with the first and the third term. $L(w) = L(w^*) + 1/2(w - w^*)^T H(w - w^*)$.

(Refer Slide Time: 27:16)



Let us take its gradient now. The gradient will be given by $\nabla L(w) = \nabla L(w^*) + H(w - w^*) = H(w - w^*)$.

(Refer Slide Time: 27:51)



(Refer Slide Time: 28:05)

L2 Regularization

- Penalty on L2-norm of parameters commonly known as L_2 **weight decay**, or simply **weight decay**
- Loss function defined by:

$$\tilde{\mathcal{L}}(w) = \mathcal{L}(w) + \frac{\alpha}{2}\|w\|^2$$

L_1-weight decay

$\frac{\alpha}{2}\|w\|_1$

- Gradient of total objective function:

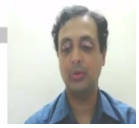
$$\nabla\tilde{\mathcal{L}}(w) = \nabla\mathcal{L}(w) + \alpha w$$

- Update rule:

$$w_{t+1} = w_t - \eta\nabla\mathcal{L}(w_t) - \eta\alpha w_t$$

Credit: Ali Ghodsi, University of Waterloo; Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H)



Going back to this equation that we have on the previous slide, let us write that out.

(Refer Slide Time: 28:12)

L2 Regularization: Analysis

- Let w^* be optimal solution in absence of regularization [i.e. $\nabla\mathcal{L}(w^*) = 0$]
- Consider $u = w - w^*$. Using Taylor series approximation upto 2nd order:

$$\mathcal{L}(w^* + u) = \mathcal{L}(w^*) + u^T\nabla\mathcal{L}(w^*) + \frac{1}{2}u^THu$$

$$\mathcal{L}(w) = \mathcal{L}(w^*) + (w - w^*)^T\nabla\mathcal{L}(w^*) + \frac{1}{2}(w - w^*)^TH(w - w^*) \quad (1)$$

$$= \mathcal{L}(w^*) + \frac{1}{2}(w - w^*)^TH(w - w^*) \qquad (\because \nabla\mathcal{L}(w^*) = 0)$$

- Taking derivative:

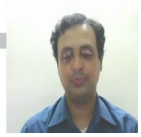
$$\nabla\mathcal{L}(w) = \nabla\mathcal{L}(w^*) + H(w - w^*) = H(w - w^*)$$

- Gradient of total objective function (in presence of L2 regularization):

$$\nabla\tilde{\mathcal{L}}(w) = \nabla\mathcal{L}(w) + \alpha w = H(w - w^*) + \alpha w$$

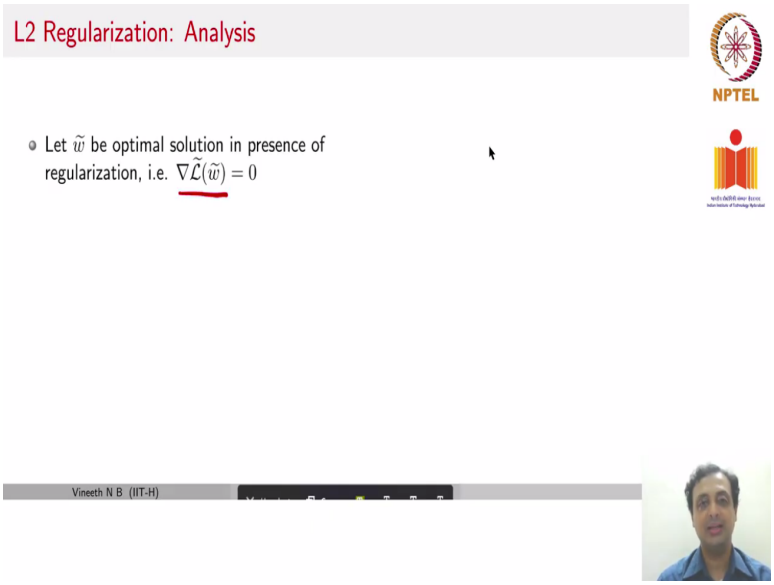
Credit: Ali Ghodsi, Univ of Waterloo; Mitesh Khapra, IIT Madras

Vineeth N B (IIT-H)



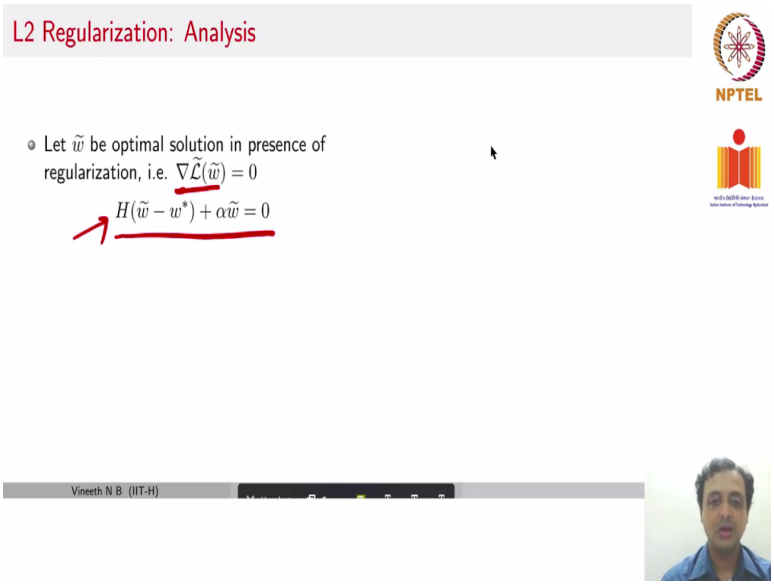
Here, $\nabla\tilde{L}(w) = \nabla L(w) + \alpha w$, that is what we saw earlier. So, $\nabla\tilde{L}(w) = H(w - w^*) + \alpha w$. Why all this? What do we want to do?

(Refer Slide Time: 28:30)



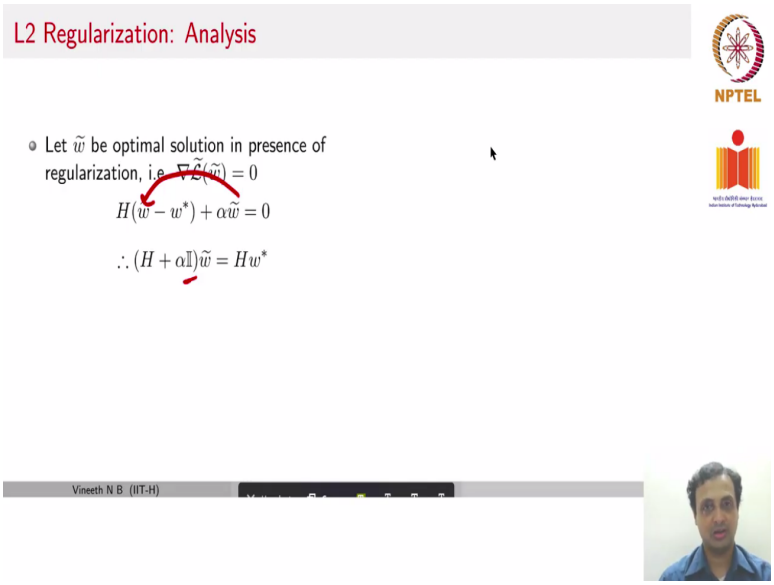
Let us now consider $\tilde{w}$ to be the optimal solution in the presence of regularization, which means $\nabla\tilde{L}(w) = 0$.

(Refer Slide Time: 28:46)



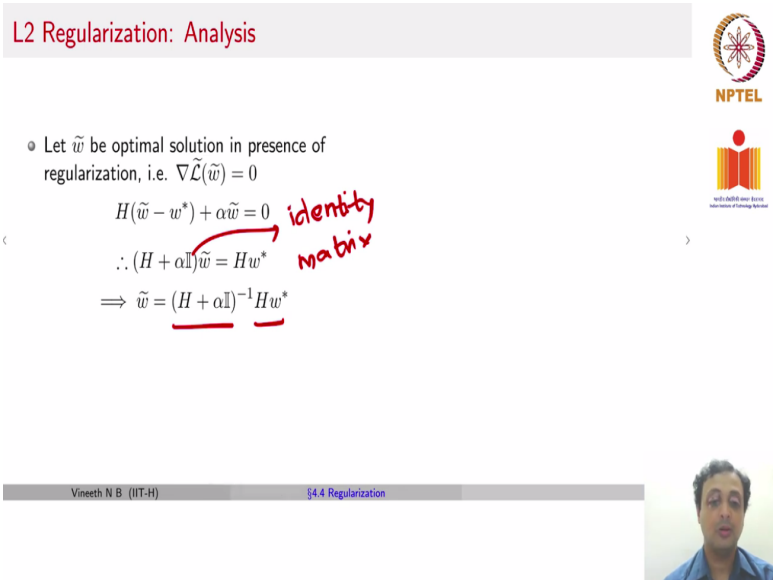
Which means $H(\tilde{w} - w^*) + \alpha\tilde{w} = 0$, because we just showed on the previous slide that $\nabla\tilde{L}$ can be written this way. Since that 0, this also ought to be 0.

(Refer Slide Time: 29:08)



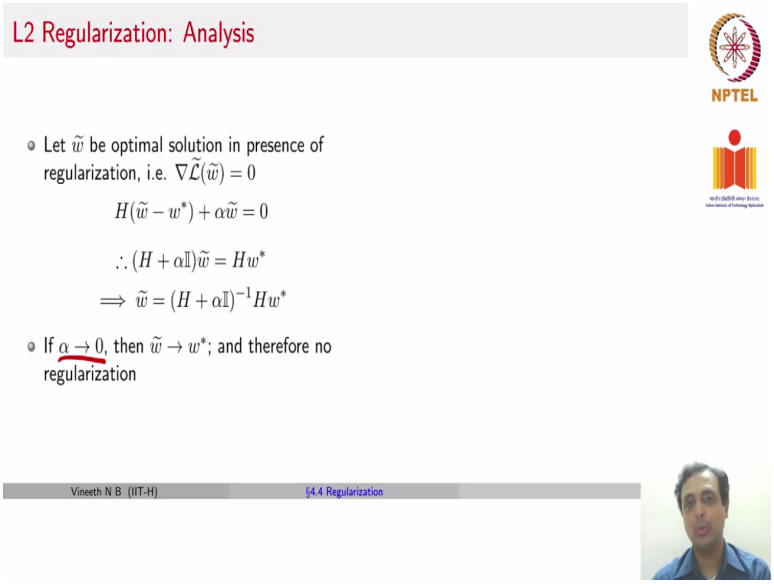
Now, let us rearrange the terms a little bit. And you can write it this way. Let us group all the $\tilde{w}$ terms. That would give us $H + \alpha I$, because when you add it, you have to make it a matrix. So, we have to put αI here to ensure that α is added to every diagonal element of H you are going to have $(H + \alpha I)\,\tilde{w} = Hw^*$.

(Refer Slide Time: 29:37)



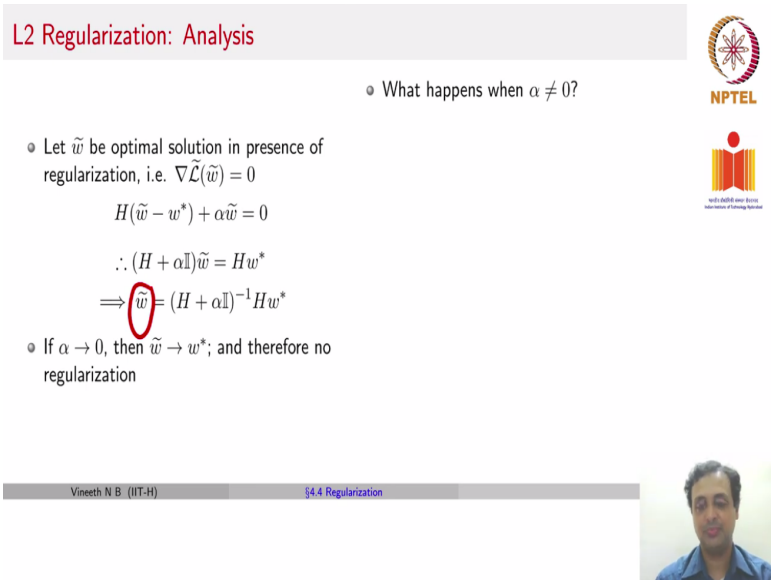
This means $\tilde{w}$, which is a solution of your regularized loss function is going to be given by $(H + \alpha I)^{-1} H w^*$. Just to remind you, I hear this notation means the identity matrix in case you do not you did not get that already. This is your identity matrix.

(Refer Slide Time: 30:12)



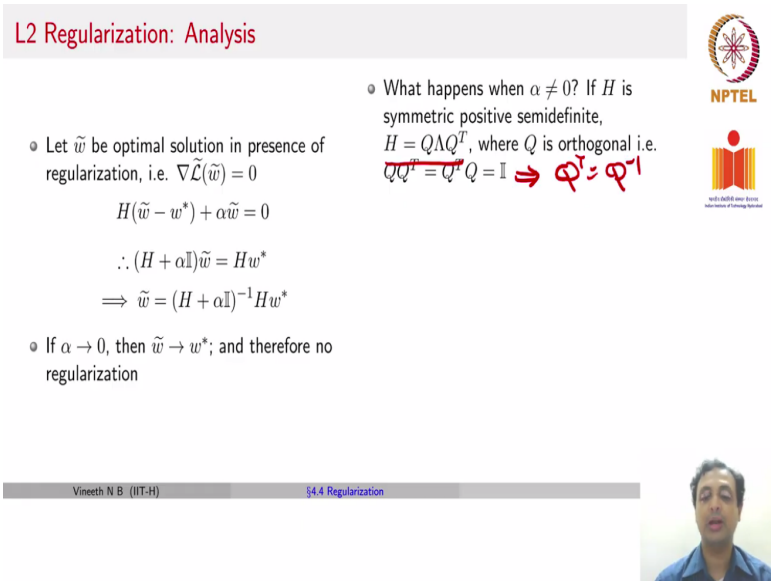
Now, in this expression, if α goes to 0, it is very evident that you would be left with $H^{-1} H w^*$ and $H^{-1} H = I$, or rather, $\tilde{w} = w^*$ itself? That is equivalent to doing no Regularization, of course, because if $\alpha = 0$, the coefficient of your L2 weight decay term goes to 0, and there is no Regularization.

(Refer Slide Time: 30:39)



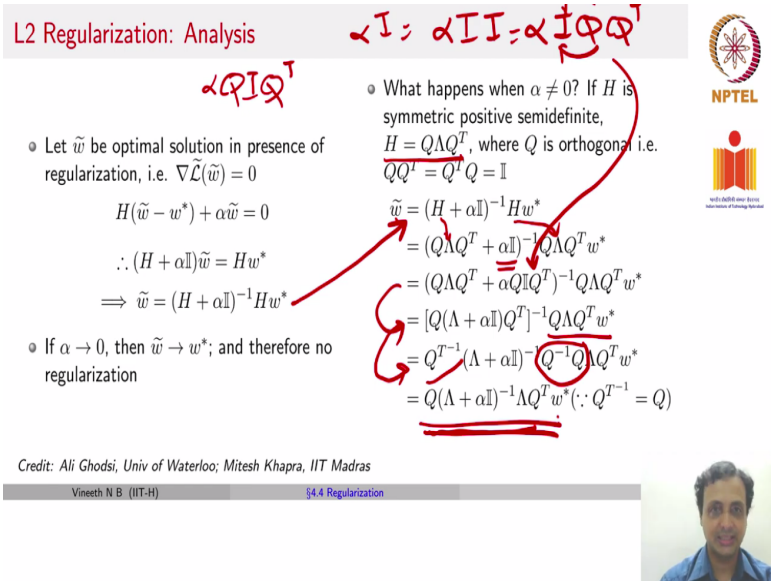
So, we are only concerned about the case when α is not equal to 0, what can we say about $\tilde{w}$. It is not trivial to say something in general about $\tilde{w}$, unless you assume some form for H.

(Refer Slide Time: 30:58)



So, to do that, let us assume that H is symmetric positive semidefinite. Positive semidefinite means that the Eigenvalues of H are all greater than or equal to 0. In such a scenario H can be decomposed as $Q\Lambda Q^T$ where Q is an orthogonal matrix Q, which means $Q^TQ = I$. This should also tell you that $Q^T = Q^{-1}$.

(Refer Slide Time: 31:39)

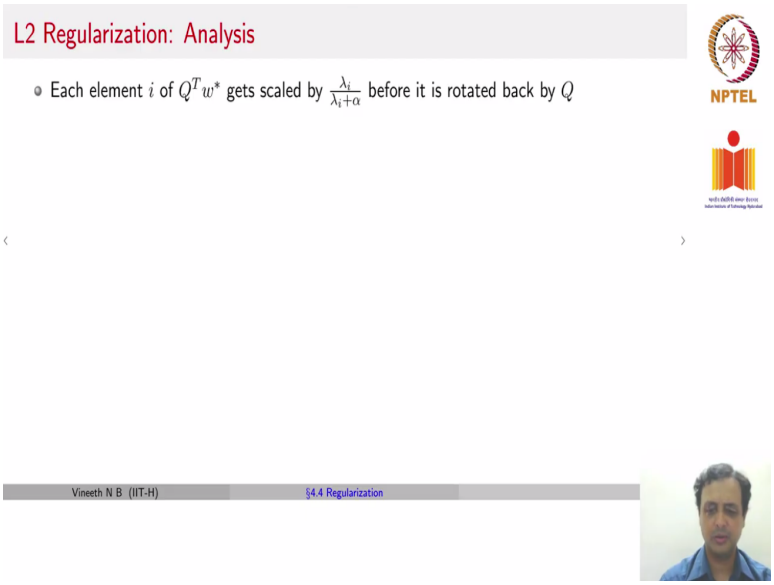


So, why do we do this with this, let us try to analyze what $\tilde{w}$ is? $\tilde{w} = (H + \alpha I)^{-1}Hw^*$. Finally, we get $\tilde{w} = Q(\Lambda + \alpha I)^{-1}\Lambda Q^T w^*$.

So, this is what you get as $\tilde{w}$. What are we doing with this seems to be a complex representation of w tilde up what do we want to do with it?

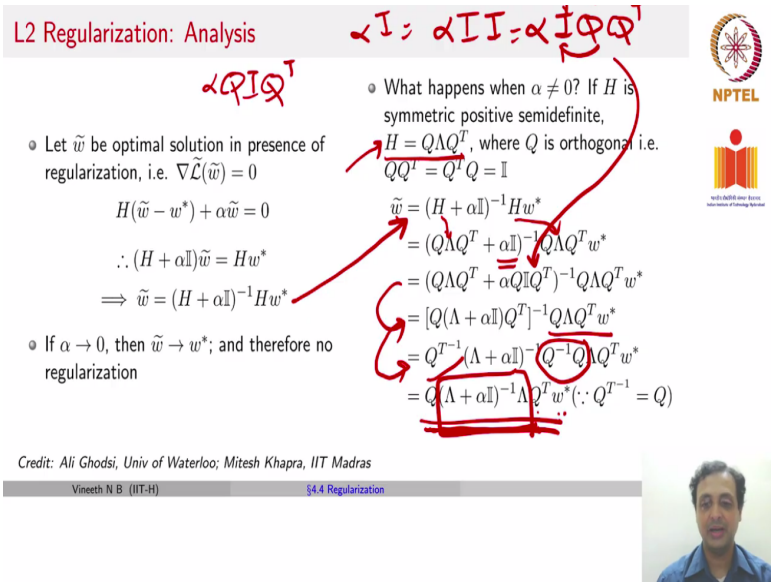
The main takeaway here is that $\tilde{w}$ is connected to w^*. The optimal solution of the regularized loss is connected to the optimal solution of the original loss by this transformation, the transformation that multiplies w^*. Let us try to assess this transformation more carefully.

(Refer Slide Time: 34:19)



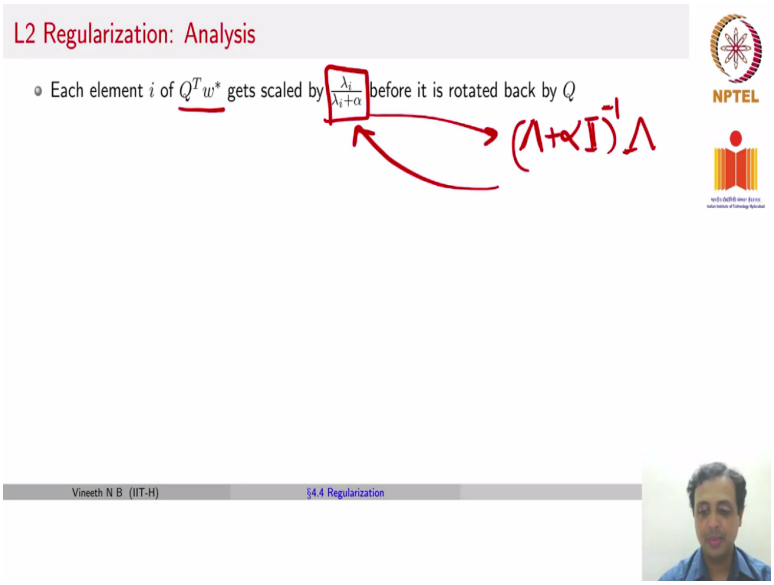
If you observe here,

(Refer Slide Time: 34:23)



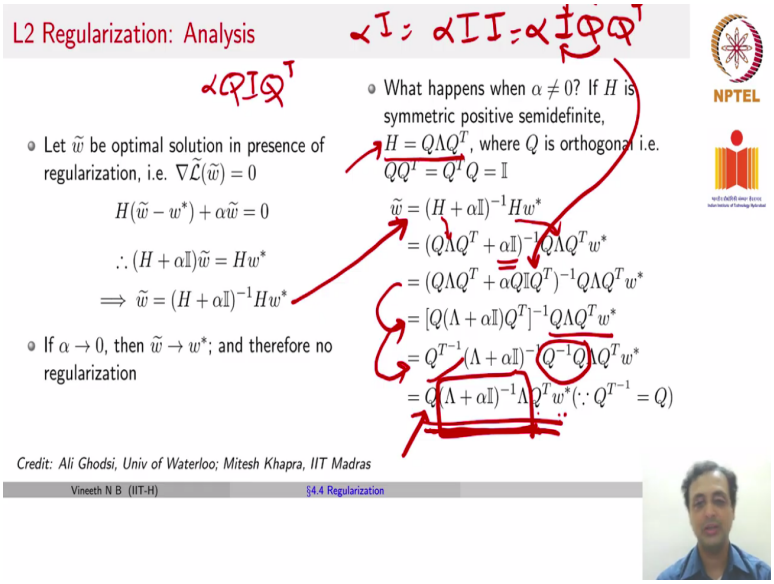
You first take w^* multiplied by a matrix Q^T which is equivalent to doing some transformation on w^*. And after you do that, you are multiplying by this term here. Remember Λ as a diagonal matrix, whenever you have a decomposition such as this Λ will be a diagonal matrix. So, you have $(\Lambda + \alpha I)^{-1}\Lambda$.

(Refer Slide Time: 35:01)



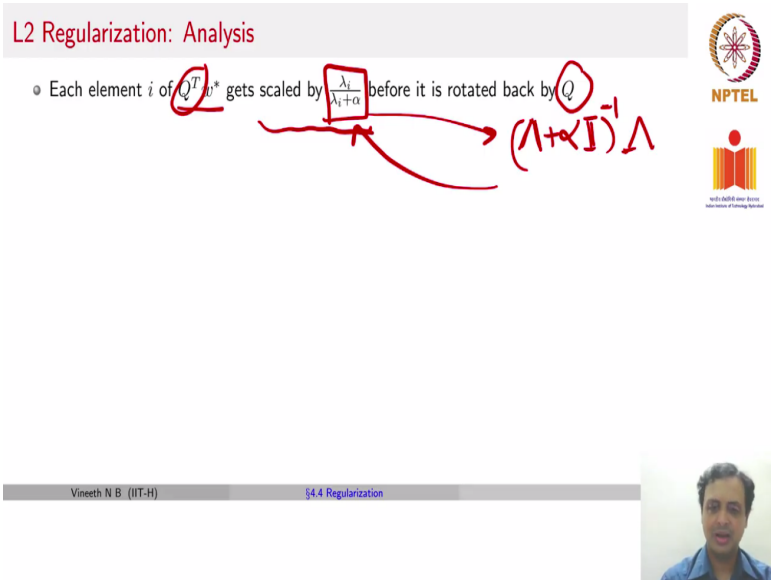
You now write this more succinctly as you take every element of $Q^T w^*$, it gets scaled by $\lambda_i/(\lambda_i + \alpha)$. How did we get this, we got this as one element of lambda, this is $(\Lambda + \alpha I)^{-1}\Lambda$, which is what we had on the earlier slide. So, this is just one term of that, because this is equivalent into So, these are going to be diagonal matrices, one of these terms will actually turn out to be $\lambda_i/(\lambda_i + \alpha)$, which means every element of you first taking w^*, transforming it by Q^T, scaling it by $\lambda_i/(\lambda_i + \alpha)$.

(Refer Slide Time: 35:50)



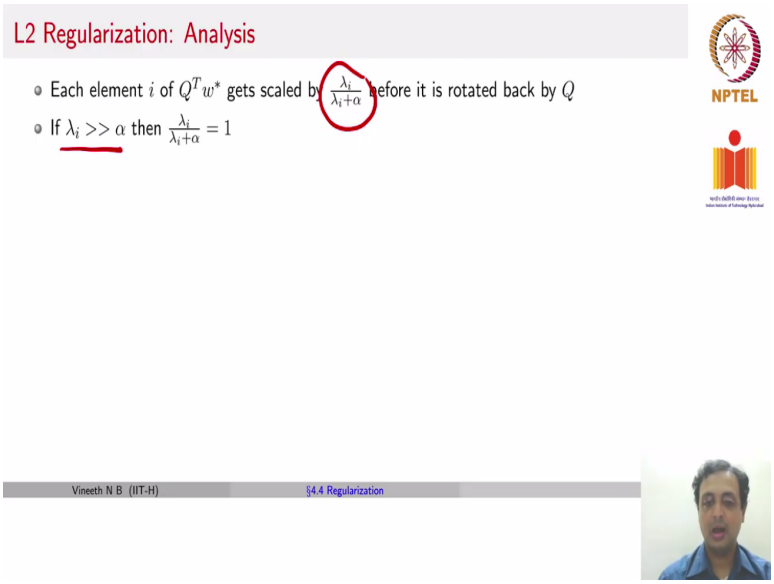
And then at the end, you are rotating it by Q again, Q is again, when we say rotate, we mean it is a transformation imposed by Q on the output.

(Refer Slide Time: 35:58)



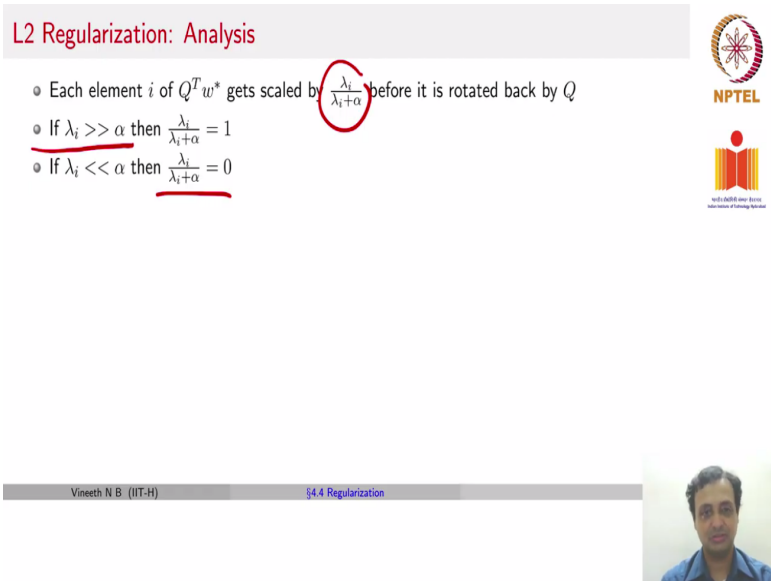
Now, you are initially transforming it by Q^T and later transforming it back by Q. So, in some sense, they will probably cancel out each other in terms of the effect, but in between, you are imposing a scaling on your weight vectors or on your optimal weight vector.

(Refer Slide Time: 36:20)



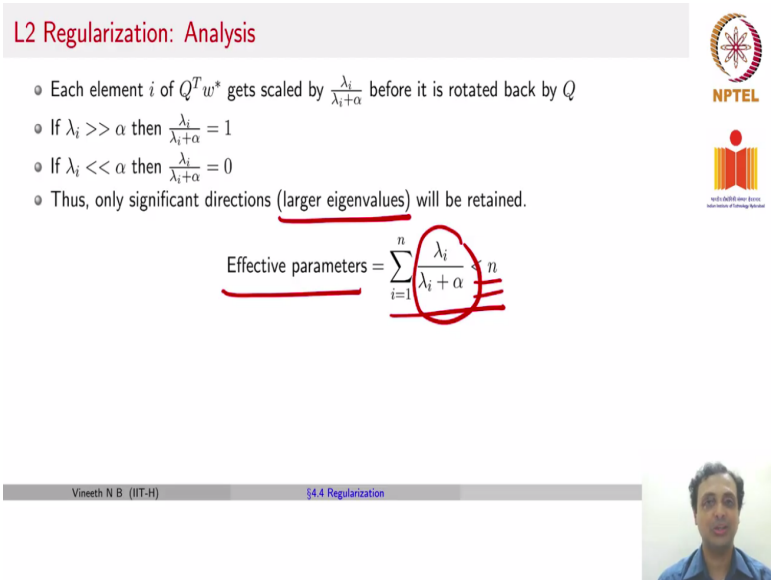
Let us see what that scaling means, the scaling says that if $\lambda_i >> \alpha$. This is going to become 1.

(Refer Slide Time: 36:31)



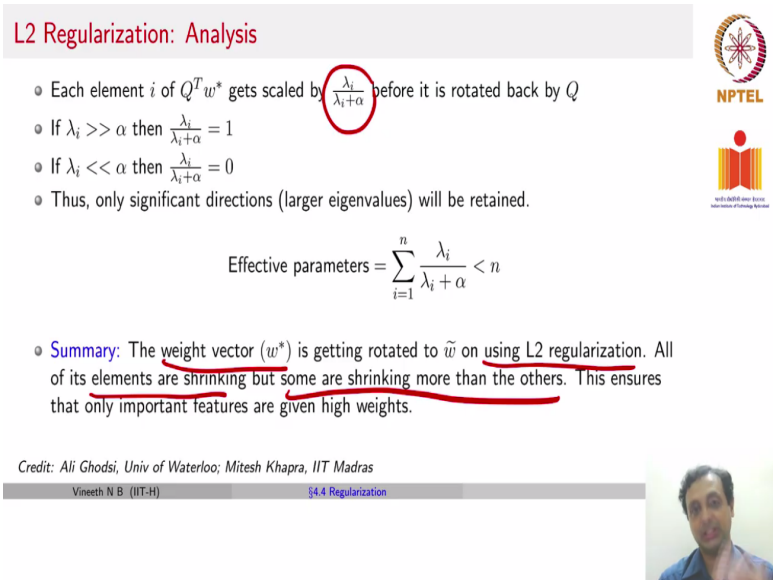
And if $\lambda_i << \alpha$, this will become 0 rather, the scaling is going to be a value only when $\lambda_i >> \alpha$, because in other cases, it is going to become 0. So, those components of w^* may just become 0 after this transformation.

(Refer Slide Time: 37:00)



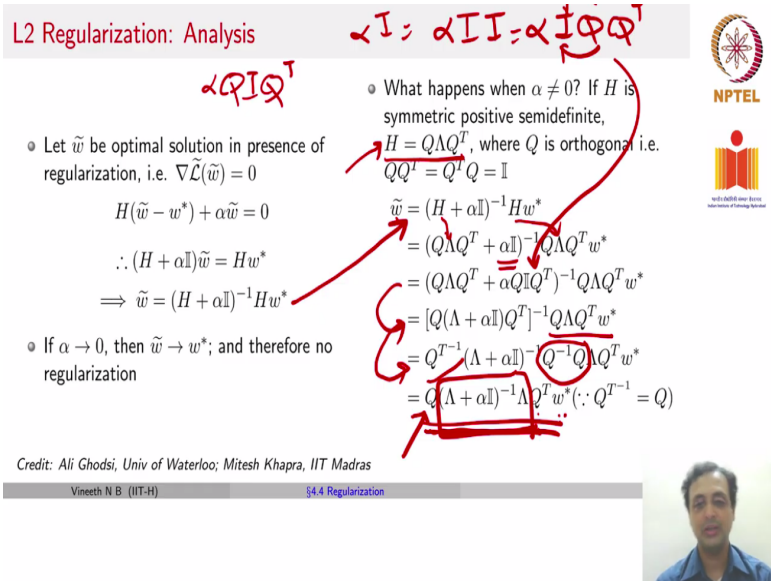
So, only when we have large Eigenvalues, the w^* components will be retained in $\tilde{w}$ which is the new solution for your regularized loss. And the total number of parameters is going to be given by $\sum_{i=1}^{n} \lambda_i/(\lambda_i + \alpha)$ which has to be less than n because $\lambda_i/(\lambda_i + \alpha)$ will be a quantity less than 1 less than or equal to 1. So, which means the sum for all n elements will be less than n. So, which means, even if you had n parameters or n components of your w^* vector, the effective parameters will definitely be less than the number of parameters divided by the number of components that you had or dimensions that you had in your w^*.

(Refer Slide Time: 37:52)



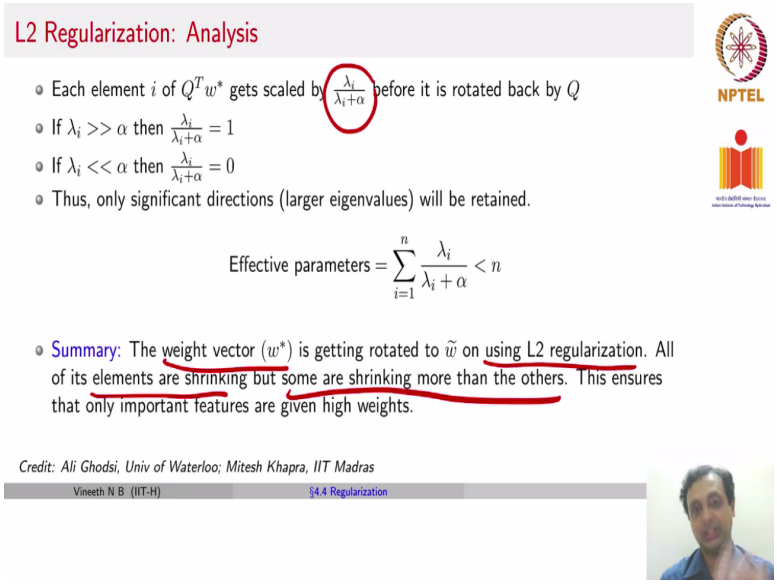
The summary here is that your original solution w^* is getting rotated to $\tilde{w}$. When you do L_2 Regularization, all of its elements shrink, because $\lambda_i/(\lambda_i + \alpha)$ is always less than 1 as long as α is a non-zero value. α is always going to be less than 1 which means all of its elements are shrinking, but some are shrinking more than the others depending on those Eigenvalues λ and what are those Eigenvalues of H.

(Refer Slide Time: 38:33)



But that is why we wrote it. A couple of slides back it is Eigenvalues of H then Hessian.

(Refer Slide Time: 38:37)



This ensures that only important features get high weights and other features may not get high weights. That is one way of understanding L_2 weight decay.