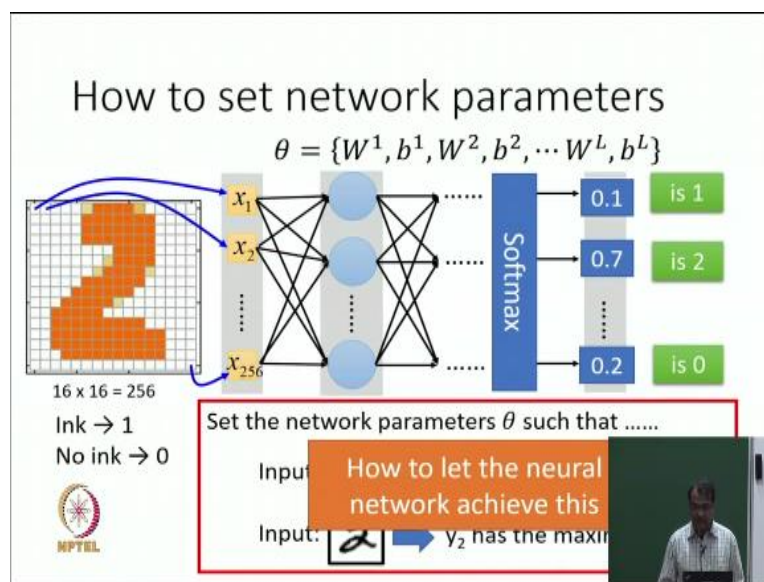


Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology-Delhi

Lecture - 87
Deep Learning: Differentiable Loss Function

So now we come to the more important aspect of deep learning, more important or very important, whatever. We have defined this architecture, we have defined the full forward propagation.

(Refer Slide Time: 00:31)



Which basically means, give me an image, I will convert it into a bit vector and send it to my many layers, I will apply a softmax, I will get a probability distribution, I will pick the maximum one and output that as my answer. This is what is my architecture for this particular problem. This is called an architecture because I have a certain set of layers.

I have a certain functions in there certain activations all of that is part of my architecture. But what is not known in this architecture? The weights are not known. What are the weights? We have defined a symbol for the weights, what are the weights? w i's and b i's, right? Because at every step I am doing $w \times$ plus b or w output of the previous layer plus b nonlinearity. Now how do I want to set these parameters?

Let us let me use the term θ for all the parameters cumulatively. θ is the superset or the set of all parameters. How do I want to set this parameter θ ? I want to set it such that it makes correct predictions, as simple as that. And what is this notion of correct predictions? Well, I am given some training data. So I am given that in my training data that I have this image and for this image the output should be 1.

I am given this image and for this image the output should be 2. This is given to me. My goal is to find the parameters θ that allow me to achieve this okay. Question. Okay, question was asked, given this input should y_1 be maximum or should it be 1. If we are okay with maximum, why are we okay with maximum because the way we are going to do the final prediction is that whatever has the maximum y_i , we will just output that i as the answer.

So even if it is not 1 it is close to 1 or it is just maximum even if it is 0.6 as long as everything else is less, we will still make the right prediction so we will be happy. And that might still be okay because it is possible that I have a certain symbol which is difficult even for people. How many times like somebody wrote one like this and it looks like seven. These things have happened.

So sometimes people may be confused, the training data may be ambiguous. The training data may have may be incorrect. All they may be noise of various kinds in various parts of the process. So we do not want to push it to one and you know force the networks to learn more than what we need. Because by forcing it, we might end up making the network less generalizable, right.

So we are satisfied as long as y_i is maximum. Right. So what is your name? Chanakya. Chanakya says what happens when y_i is less than 0.5 if it is maximum, let us say because it is a 10 class classification problem. Let us say everything is point 0.05 except or let us say everything is 0.8 0.08 except one is 0.28. What would we output? 0.28. The one which has 0.28. That is what we are going to output.

Are we happy with it? Probably not, right? We may not be happy with it. And when might these situations occur? When I do not even give a digit as input. I draw a

random character and give it as input just to test the system. Okay, which digit will you recommend now? I am giving you the symbol x. So if you start playing with neural network like that, obviously the neural network will say look, I only know my world is about digits.

I only know that much. So I will output a probability distribution. It is possible that all the b_i 's are low. But then when you do softmax, they all may become relatively uniform. In which case what will my system output? It will still output the number which has the digit which has the highest probability. If we want to allow for such a possibility, we would allow for a no output as one of the outputs. None of the 10.

And then we will have to train it for it. So we will give it random images. And we will say, look this is not anything from 0 to 9, this is not anything from 0 to 9, it is none of the digits. And then we will hope that one of the answers would be higher probability. But for now we are just going to enforce that the maximum should be what we are looking for okay. Even if it is less than 0.5. What if two y_i 's have maximum value, right?

That is another question. What happens? First of all, can it happen? Well, it can happen with an extremely low probability because these are weights which are completely trained by the systems or the weights may not be exactly the same. Because of it, they may be small variations in fourth digit fifth, digit somewhere, and so you will output that. But more importantly, suppose two of the digits are 0.45 or 0.44 or whatever it is, what does that tell you?

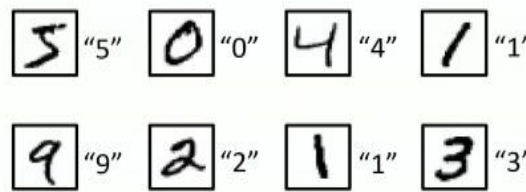
That tells you that the model is confused. That tells you that look the model is really confused whether this is this digit or that digit and then use that information downstream. You can ignore that particular digit, you can ask humans for that particular digit, right? You can do whatever you want to do with it, right. So but for now again, this is deep learning 101, right?

So all your questions are sort of valid, but in practice, you we just comfortable saying I want the max to be the correct answer. And I want to find those parameters θ , which let me achieve this.



(Refer Slide Time: 06:53)

Training Data

- Preparing training data: images and their labels

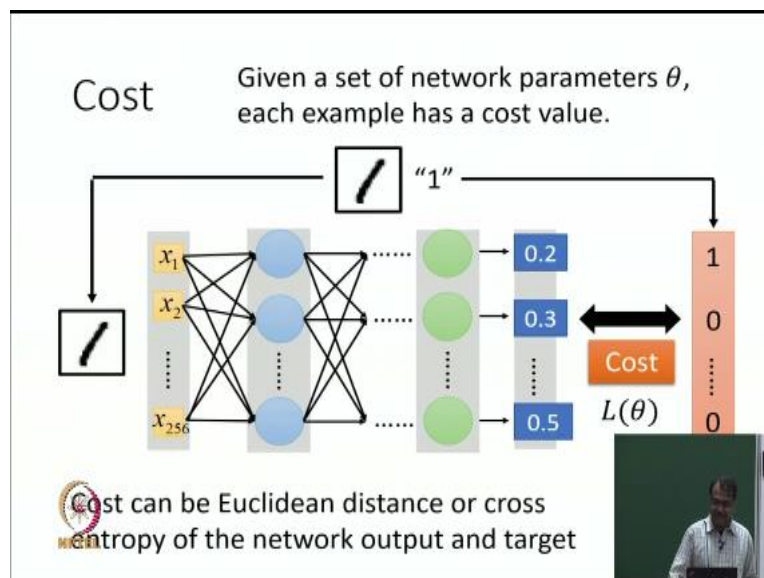


Using the training data to find the network parameters.



Now I am given some training data. So I am given this kind of training data. We will call this x, y pairs. So this is my x 1×1 . This input should lead 5 to 5, this input should lead to 0, this input should lead to 4 and so on so forth. And now I need to somehow use this training data to find the best network parameters. And the fundamental concept that we are missing is what is called the loss function.

(Refer Slide Time: 07:19)



So what actually happens is that you have some parameters, current parameters. You take a particular training example as input, give it to this neural network and have the neural network produce a probability distribution. So it produces a probability

distribution. Now I will say that given a set of network parameters, the probability distribution that I output, I can compare with the true label.

The true label would be the y_1 for the example at hand like for this it will be 1 and then I would say how costly is this prediction. How bad is this prediction. Am I satisfied or am I not satisfied. For example, this is my target, probability distribution and this is the computed probability distribution. Now are you satisfied? No, you are not satisfied. So what do you want to say?

You want to say look, I am not satisfied you should be given high penalty for such parameters. And now you have to define such a function which can give high penalty for such parameters. So one such function would be some simply squared loss and I do $1 - 0.2$ whole square + $0.3 - 0$ whole square + $0.5 - 0$ whole square and some that right. But sometimes it can be more interesting.

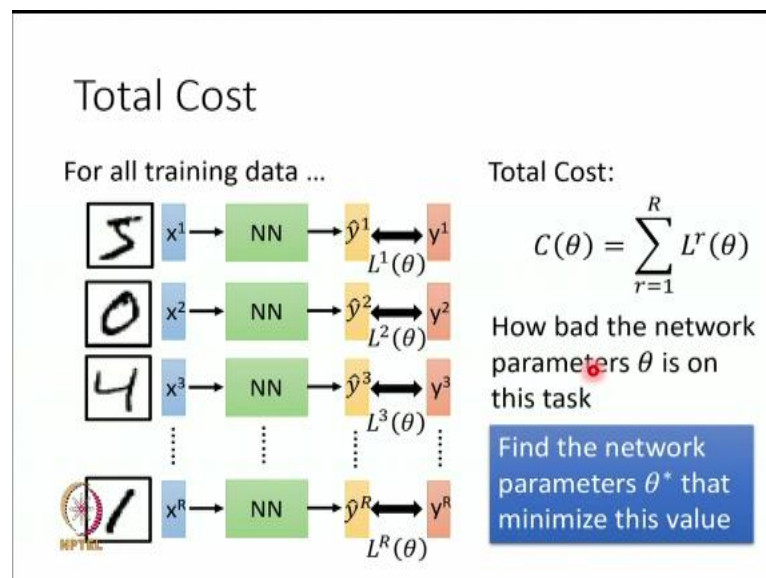
Squared loss does not work for probability distributions as well. We have something called the cross entropy function. For people who are interested they can go and read it out. It is the way to compare two probability distributions. So nevertheless, the point is that this computes a loss function, we will call l , a cost function we will call l .

And this l is a function of θ , because I use these θ s to compute this prediction, which I compared with the true to get the final loss. If I had a different θ , I will have a different loss. So finally, what do I want? I want those parameters θ that minimize the loss. So therefore, what have we done, we have taken our neural network and we have converted the training problem finding the parameter problem into an objective function that we can minimize.

Now that should not surprise you. Because even in Bayes nets that is sort of what we do. Even in RL when we did some of basis functions, that sort of what we did. We defined the squared loss and said let us take the derivative. That is exactly what is going to happen and again it should not surprise you also because I told you alt learning is nothing but some form of optimization.

And it will have a form like w_i is equal to w_i plus something. It is always going to have that form, w_i is equal to w_i 's plus learning rate times something, okay.

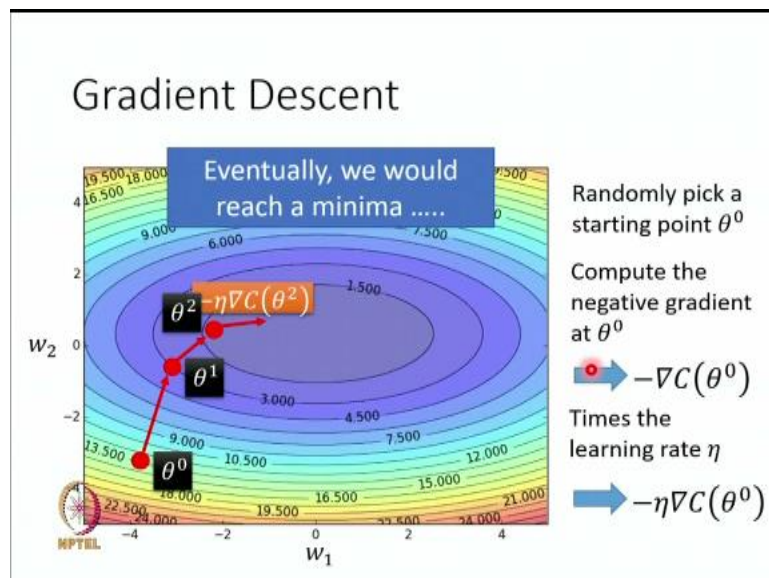
(Refer Slide Time: 10:21)



Now I am given many training data points and for these many training data points, I can compute many loss functions. So we use the term y_1 is the true label and \hat{y}_1 is the predicted label or predicted probability distribution. So for example, in this case, if the input is 4 that is x_3 . If the output whatever is the probability distribution that is the output that is \hat{y}_3 and then the final target is y_3 .

And then there will be some loss between the two. And so I add over all the data points. This is typically done. We take each input as independent, and their contribution to the loss function is independent, okay? So find the network parameters θ or θ^* that minimize this particular loss function that is sort of our goal.

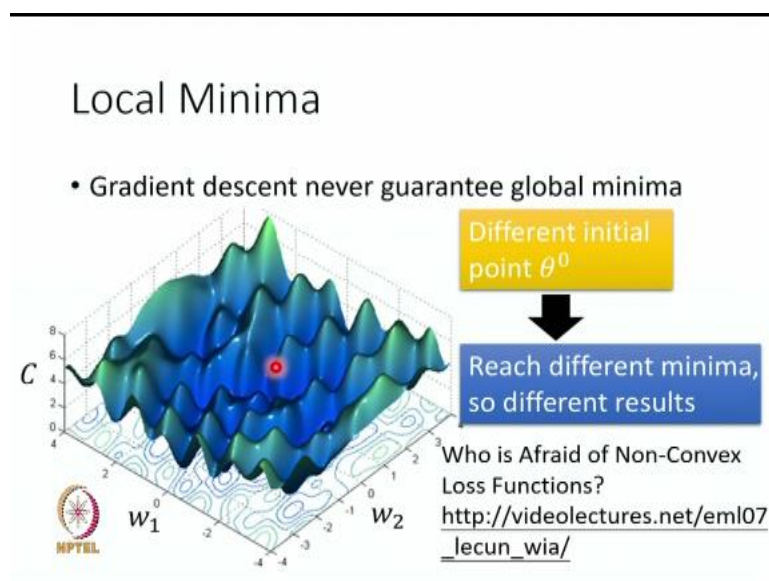
(Refer Slide Time: 11:29)



And we can do this by gradient descent, right. So how do we do this by gradient descent? We have some loss function, we have some parameters theta, we can compute the derivative, we start from any random theta and then we try to move in the direction of the gradient or opposite the direction of the gradient because you are minimizing and we keep doing this keep doing this keep doing this and eventually we will get to get somewhere close to the minima.

So all of learning modern learning continuous variable learning is nothing but some form of gradient descent. But of course, would it lead to global optima? In general no.

(Refer Slide Time: 12:16)



And in fact for neural networks almost never. And the reason is that because I have layers of these neural neurons, and each layer has an activation which is not

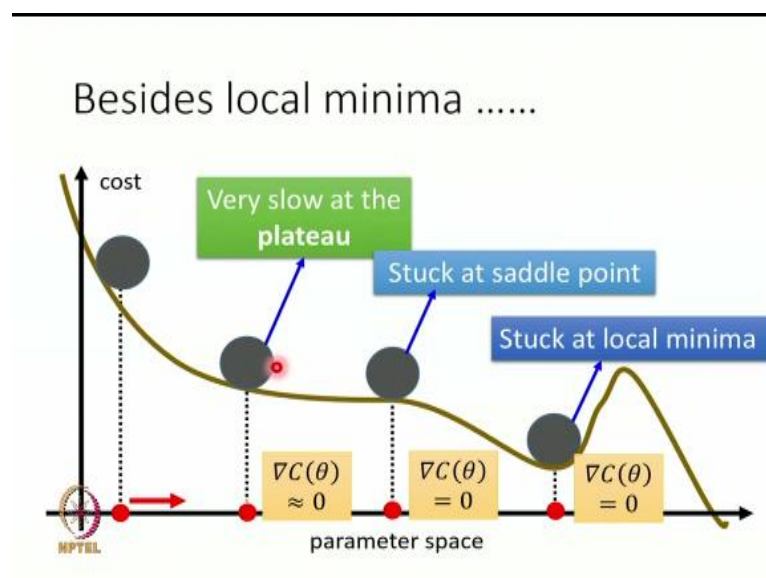
necessarily convex, but even if it is the combination of these convex because the output goes into the next and so on so forth makes a highly nonlinear function. And in fact, this is the typical nature of the nonlinear function.

And so the final theta that you get is some local optima, okay. Depending upon where you start, you reach some local optimum and you get different results. But we are not afraid of non-convex loss functions. And this is the just the beauty of AI. The theoretician says you cannot reach global maximum, we say fine. We will just do whatever we can with local maximum.

This has been there even in local search when we read local search in the AI class, and this is also true in the context of neural networks. So we come up with strategies where we can find better optimum or better optima, but we do not care if we do not. We just keep trying to improve it. We train it many times from different starting points, we do whatever we can, and finally, we get an theta star which is the best so far.

We just tell the application's people look we have trained a great neural network. Now you go and classify your images with it or whatever, okay.

(Refer Slide Time: 13:45)

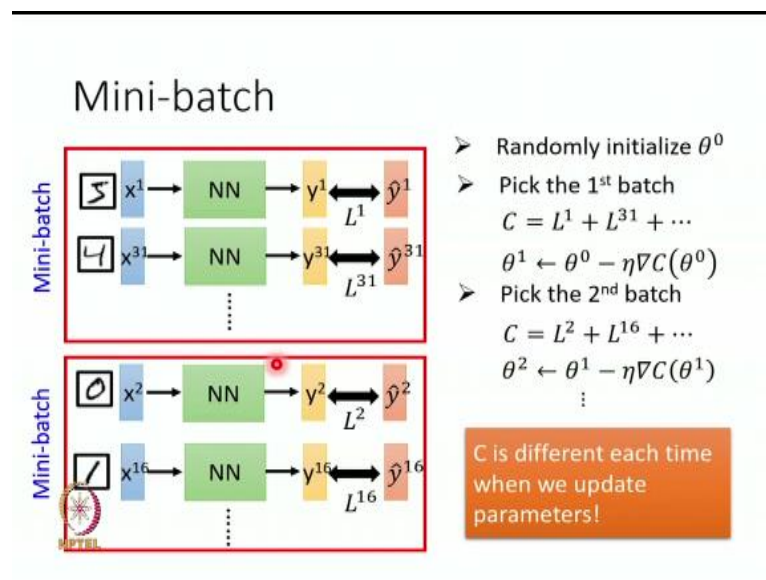


And it is not just local maximum or local minimum, pretty much not different from what we studied in the local search class. Yes, we could be stuck at local minimum, but we can also be stuck at saddle points. Saddle points are where we can make a

move, but you know the derivatives will be zero, we may be stuck a very, we may be very slow on plateaus and so on so forth.

And there are lots of you know even theoretical investigations that are going on, on how not to how this may not happen and what to do. We are not going to talk about this. You should at some point I encourage you to take a deep learning course where all these things which I am describing in two to three minutes will be described in one to two lectures, okay.

(Refer Slide Time: 14:31)



Then the next very important phenomenon is the phenomenon of mini-batch. Because we will typically have many parameters. We will have large training data points. If we have too many training data points, then going over all the training data and computing one loss function by doing summation over all and then taking one gradient step might be very slow. So what we do? We typically say okay, which GPU do I have?

I have a 32 Gb GPU or a 16 Gb GPU or a 8 Gb GPU. Okay, fine. How many training data points can I fit in my GPU so that each matrix computation gets done in like unit time, okay. So the system says, maybe 128. So you say fine, we will just divide the full data into pieces of 128 data points each, and we will call each of that group as a mini-batch. And then we will give this mini-batch to my GPU.

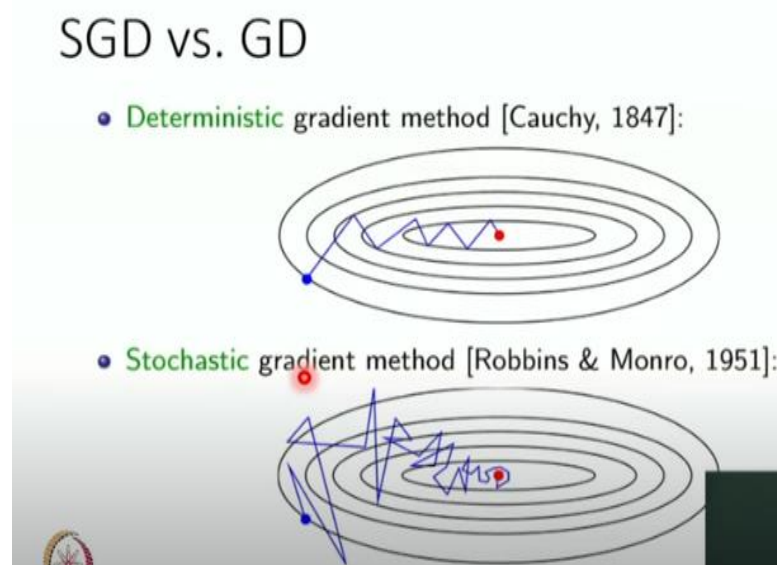
It will compute all the parameters, it will do take a gradient step only on those 128 data points. So now I have changed my theta. Then I take the next 128, give it to the GPU, then I get the next theta from theta 1 to theta 2. Then I give it the next then I go from theta 2 to theta 3. And I keep doing this. At some point I am done with my training data. What do I do? Start again.

Maybe if I am smart, I will re-divide them into a different mini-batch combination or I will just say, forget it, I will just give the one mini-batch again. Then the second, the same first mini-batch, the same second mini-batch. Going over training data once is called one epoch. One epoch goes over many mini-batches and does many gradient steps. Okay, so again just to go over the slides.

Randomly initialize theta 0, pick the first batch, compute the loss only on the data points that are in the first batch, do a theta step gradient step, then pick the second batch do the next same thing pick the next theta 2 and keep doing this. C is different each time when we update parameters. The data points obviously update is different.

And the extreme case of mini-batch is what is called a stochastic gradient descent where in the extreme, we only take one data point. But mini-batch is only somewhere in the middle, right.

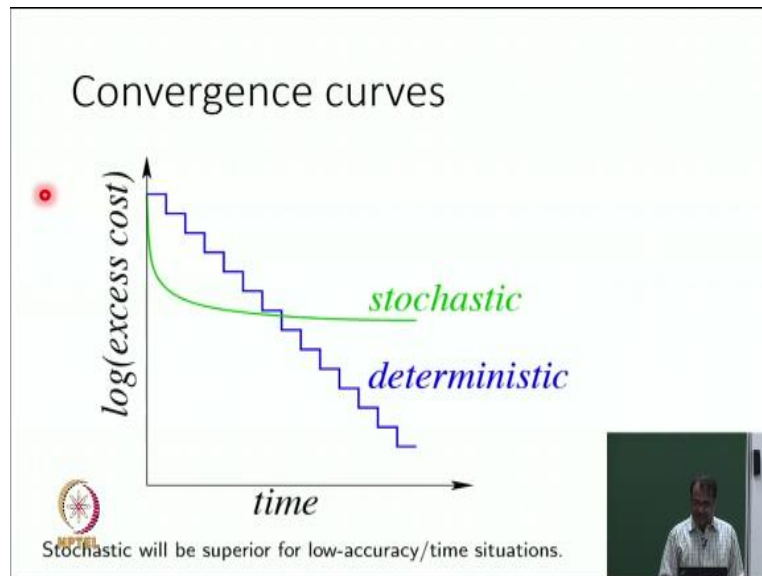
(Refer Slide Time: 16:58)



So if we really did gradient descent, what will happen is that our gradient would be better computed, obviously, because you would have gone over the full training data.

On the other hand, if I am doing stochastically, if I am only doing it on a small training data, then I may be going everywhere and I might sometimes be going in the wrong direction, because it is not giving me the correct gradient.

(Refer Slide Time: 17:22)



On the other hand, if you plot the time versus cost curve, then each gradient computation will be very slow, because you would have to go through the whole training data and after that, you will make one move. So for a long time, you will not do anything then you will make a move where your cost will increase or reduce. Then you for some time you will do gradient computation and then your cost will reduce.

So this is the kind of curve you will get when we are doing gradient descent. On the other hand, if you are doing stochastic gradient descent you will start one data point quick compute gradient move, another data point compute gradient move. You will be making very fast moves, but your cost will be decreasing slowly. So which one is better? Well in questions like this typically the answer is it depends.

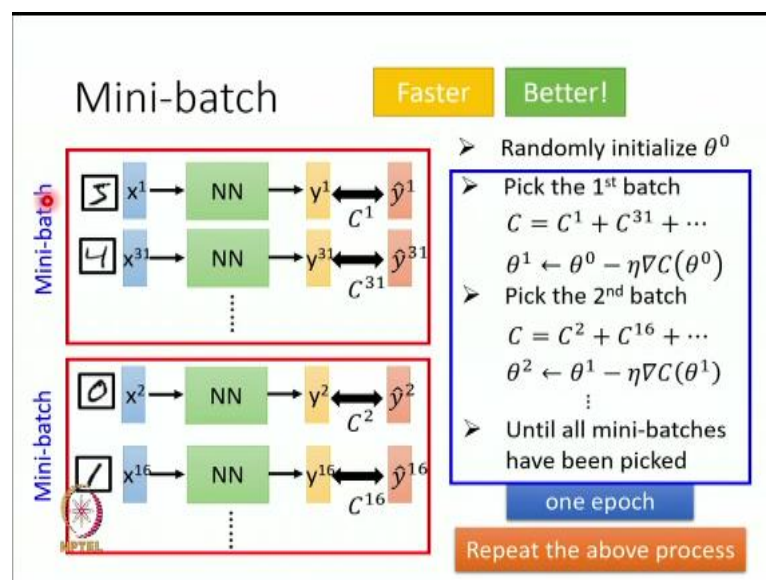
If you when you go to your interview next year you have a you know job interview and the person ask you an interesting question. Is A better or B? You must say always it depends, but you should not stop there. Almost everything you will be able to find a reason to like, same with people. They always have some good qualities.

So in the case of gradient descent versus stochastic gradient descent, it has been found that early on in the training, stochastic gradient descent is really better. But later down

the line, gradient descent is better. But because we have large training data points, we are almost never in this region of the curve. So we are almost always in this region of the curve. So in practice nobody uses full gradient descent.

They all do a mini-batch based on the GPU limitations and they do stochastic on that mini-batch, okay. A gradient descent in the batch and then stochastic across batches basically, okay.

(Refer Slide Time: 19:21)



And that is it. So then I think yeah, so this is faster. Until all mini-batches have been picked you do this and this is called one epoch and then after this you repeat the above process and you may shuffle your training data. Yes, question. Yes. So the question that Vaibhav asks is if I give you lots of GPUs, can you parallelize this computation? And the answer is yes, you can.

I would also say that a that programming becomes much harder for you know novices like you but also people have found that in some cases it really helps. In some cases, it is not clear how you deal with the different GPUs. Because what is actually happening is that now what you are saying is that theta 1 goes to theta 0 minus this is parallelized. So that means that your first processor sent you some theta 1, which it thinks should be the next theta 1.

Your second processor sent you another theta 1, which it thinks should be the right theta 1. And all your processors have sent too many theta ones. Now you need to

figure out how to combine them. Moreover, they may not do it synchronously, because one processor was faster for whatever reason one was slower, whatever, right? They may send you these θ_i 's asynchronously.

How to deal with this becomes a challenge. However, all of this said the most famous not necessarily now the state of the art but one of the most famous good deep learning algorithms for reinforcement learning is what is called A3C. And the first A stands for asynchronous. So essentially what you do is that, you know different GPUs are doing their computations, and they are sending the next version of their weights.

And you are asynchronously computing them and updating your weight vectors and hoping that everything converges. It is a complicated mess. But there is a lot of research that goes on into this area and this area is called distributed machine learning. Very important area, right. So extremely important in today's world. Okay.