NPTEL

NPTEL ONLINE CERTIFICATION COURSE

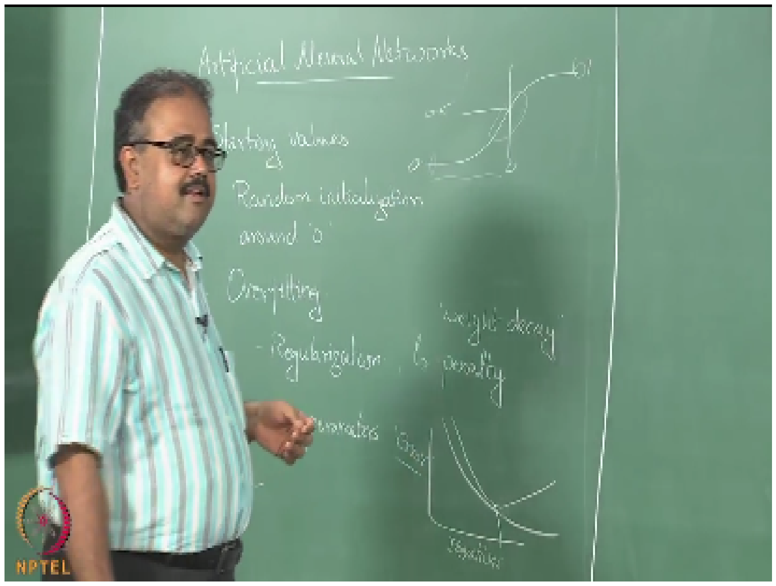
Introduction to Machine Learning

Lecture 35

Prof. Balaraman Ravindran
Computer Science and Engineering
Indian Institute of Technology Madras

Artificial Neural Networks IV – Initialization, Training and Validation

(Refer Slide Time: 00:21)



The first thing concerns the starting value of the weights right, so you have all this α and β you have this whole set of parameters in a neural network right, so we talked about gradient descent but gradient descent starts at some point in the weight space right, so you need to have an initial guess for what your α and your β should be right. So what should you do what are the good guess? set them all to1 is it yeah, so setting all of them to the same value whether it is one or sometimes people say this 0 right setting all of them to the same value is usually not a good choice right.

More often than not you will end up in some weird part of your gradient space and you will find it hard to get out right it is very rare that you are going to come up with a actual solution where all the weights are the same right. So if you were going to start off with a solution where all the weights are same right it is going to be hard to specialize right, so typically what you do is you douse random initialization, but there is one more constraint let us be feeling really lonely one more constraint to the random initialization.

So what do you think that would be the constraint yeah of course you do not really want to have infinite weights yeah, so but then what should the min and max is, so you really want your weights to be rather small okay? So just think about what is the implication of having really small weights what is the implication of having really small weights, so you remember your sigmoid right, so if your weights are really small where do you expect the outputs to lie? You know your α transpose x or your β transpose z will lie somewhere in this region because this is where 0 is right.

So this is 0.5 or 0 depending on whether you are using tan HR the sigmoid, so if you are using the sigmoid, so right around 0 you will have this 0.5, so if you look at this region this is almost a linear region correct. It is almost a linear region so I would really like to start off my network so that most of the outputs of the neurons are in the linear region. Why is that? Can you think of it and you have enough information to answer this question. The gradient will be larger right so the gradient will be larger if you are somewhere around here.

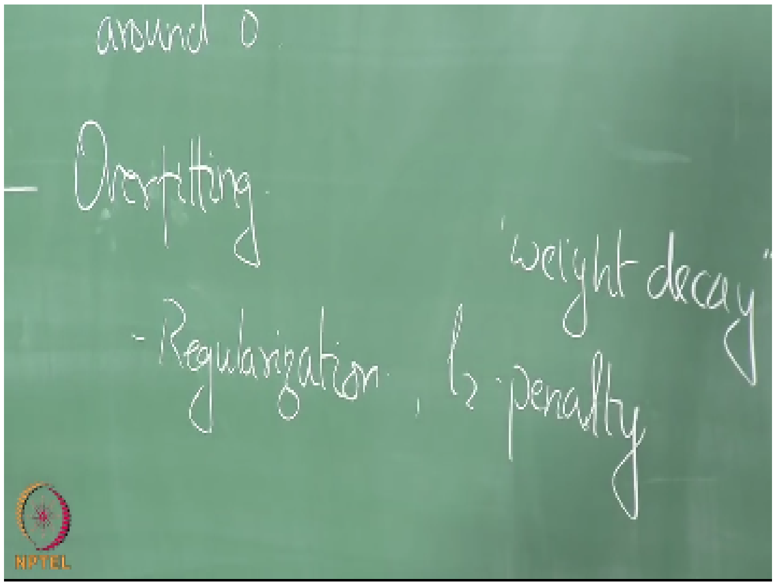
So even small changes in the input space or small changes in the weights will actually cause a large change in the output right, so if you end up going somewhere here or somewhere here right you can see that you are already saturated right, so you really have to drag yourself all the way here to see any change. So if you are somewhere in the middle you are more sensitive right to what the input right you are more sensitive to the weight changes right. So all of this helps you learn more rapidly, so this is one of the reasons you start off with random initializations around 0.

Of course you can make everything zero but then that will put you in a very weird part of the part of the search space right. So it is good to have some kind of randomness so that each wait can specialize to different things okay. It used to be the big bane of neural networks over fitting so why is that the case? You know all of you remember what water fitting is right, sorry yeah so I mean people remember what over fitting is yes, the training examples right there essentially you are fitting the parameters very closely to the training examples.

So you are not able to do any kind of generalization to unseen examples right and the reason why neural networks do this over fitting is exactly because they have too many parameters you have so many weights here. So if you remember we actually counted the number of parameters in a neural network right you know that M times 3 +1 + whatever K time’s n / M + 1 well that is a huge number of parameters. So it is very easy for you to over fit so you have to be careful about it, so there are two ways of avoiding over fitting.

So can you think of what are the two ways of avoiding over fitting one we already know regularization right, so one way of avoiding over fitting is regularization, so what you do here is you essentially add a quadratic penalty for the weights right so you do a norm α squared +norm β squared and then you try to find the gradient with respect to that and then you try to minimize things, right it becomes a little bit more complex and if you add a squared error penalty right if you add a squared error penalty it is sometimes called weight decay right because it makes your weights go towards zero. So sometimes called weight decay, so can you add a $l_1$penalty

(Refer Slide Time: 07:16)



You could you could add anything I mean so it is it just makes it a little bit more complex but the question is does it in do sparsity right, they said another way of avoiding over fitting yeah you can type parameters together to avoid over fitting that is good but in which wherever is new tied together? In fact one of the ways that deep learning actually has been made efficient despite doing this tying of parameters right but then the architectures that look at are very complex right so you could tie parameters together and try to reduce this okay.

So let me put that as a different kind of thing but it is actually a form of regularizing but it is yet another approach which is a purely empirical way of doing things which is to do what is called validation right I actually mentioned this in one of the earlier lectures right, so you train on a training set and then you have a validation set and then people remember I drew even a picture right, so as you are training so the error on your training set keeps going down right but the error on the test set or the validation set will initially go down, at some point it will start going up right maybe not that dramatically but go up nevertheless.

Like that right so the point here is where your right solution is okay, so I am putting it in quotes right solution okay, so what is the x-axis and what is they-axis on this figure? Whatever is your mission of error it could be miss classification error or whatever right, so fiction you ideally want miss classification error if it is the regression it is a prediction error whatever and the x-axis is usually iterations right but you could also think of having a figure like this for complexity right.

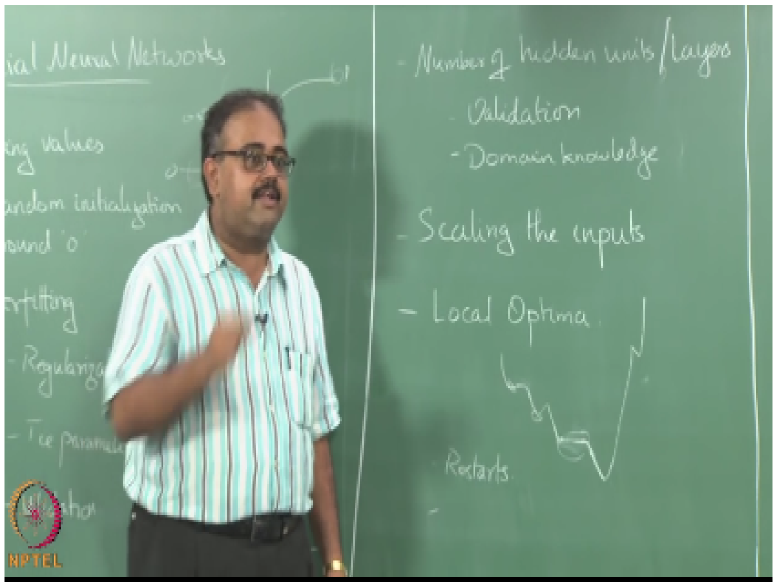
So you could say that I am going to keep adding more and more neurons right or more say I can keep expanding my M right I can keep adding more neurons in the hidden layer right.

I cannot change the new neurons in the input layer I cannot change in your notes in the output layer usually right because that is their depend on the problem that I am solving right when I can keep increasing the neurons in the hidden layer right but then these are more or less you know standard techniques for doing, I mean for avoiding over fitting not necessarily tailor to neural networks right. So you should remember that there was about a decade and a half when people worked a lot with neural networks in between right, they came up with many techniques for avoiding this kind of over fitting.

And they have explored many variations on parameter tying right and also many different kinds of regularizing so there are some really interestingly named algorithms for avoiding word fitting. so one that I particularly like is to be called optimal brain damage, essentially the idea was to remove weights from the network right so you train the neural network and then try to find out the sensitivity of the output with respect to certain weights okay. If I am changing this weight how much do the outputs change you know how much does the error change right.

So weights that have low sensitivity or right or yeah weights that exhibit low sensitivity on the error right well then remove and then you just retrained we keep doing this. So that way you are removing the number of parameters reducing the number of parameter heavily but we are not affecting the output too much. So like that there are many variations on it but these are the three things right that we have to think about.

(Refer Slide Time: 11:57)



Related to the over fitting question you see how do you figure out the number of hidden units and layers. Like the very expensive way of doing it is to do a similar validation kind of a setup right keep increasing the number of layers and then our number of neurons and check that out it is incredibly hard right and so people came up with automatic pruning techniques right, people came up with ways of growing your neural network. So they start off by having one neuron so something very similar to your forward feature selection or what is the thing a stage by stage wise selection right, so people remember stage wise feature selection what did he do?

At stepwise stage wise or something different see you later right so you could do the same thing with neural networks right, you start by training a single neuron all right and then what you do is once that neuron actually starts making some predictions right then you train another neuron that actually nullifies the prediction error right. Now you know a third neuron that adds up both of these and gives you the output right, so you could do something like that right so you do not have to make a decision as to how many neurons you are going to put in from the beginning right as you go along you just keep training more and more.

But the problem is such a network was that it will not look like your layered architecture right I will start off with 1 neuron okay that will give an output then I will add the other neuron then they will go so all of this gets the input directly, then I add another neuron right so the layer architecture is gone no wall right. So now how many layers this is our two or three you know so this neuron seems to be at the third layer right but it is connected directly to a neuron which is certainly first layer because it is taking inputs from here.

But then well you do not have to be very dogmatic about having the standard three layered architecture if you remember when introduced the standard three layer architecture I said there are a lot of different deviations from this that people have proposed right and will not be looking at most of those in detail. So these kinds of networks where you are actually trying to minimize the residual at every point they are called cascade correlation networks, so there are. So the most statistically sound ways to just do validation, what is the other way? To do it is kind of a cheat it is slightly better what can you do you can take an educated guess using domain knowledge.

I said you have some information about how complex the system is and then you can use ideas from that you can then try to see okay one layer two layers three layers right, so a lot of deep learning network that nowadays happen essentially it is more empirically driven you try one layer okay see what is the best you can do right see thoughts the best in performance, as you can get and then you try to add another layer and see if we can improve that another layer another layer until you are happy that you are performing well right. Of course you just cannot train the network into the ground you have to always make sure that you are not over fitting it but you can still this right okay.

So I did not explicitly mention this while talking about the numerical training but you can kind of imagine, wherever that I am going to be using the data asset is rate as a real valued vector right I would have to worry about scale, so if I have one variable that has a very large range another variable which has a very small range, at the variable with a large rate it is obviously going to dominate my gradient computation. If you remember the gradient has a X the x ml component to it right that is the input variable is part of the gradient. So if the variable some of the variables can have a very large range and some of the variables might have a very small range and the large range variables will dominate the computation.

This numerically by being large they are going to dominate the computation right, so we do not want that to happen whether they are actually needed or not just by being numerically large right they will dominate the computation, so we essentially make sure that all the variables will have the same range right. So we talked about this in couple of other scenarios also but in this case again it is important so this is something which people typically forget. When they are using either neural networks or SVM's you try you take the raw data right and you just try to run it through a neural network or run it through an SVM and then produce a classifier right and quite often things do not look work that well right.

You might find some reported results that are much better than what you are getting by using is SVM, nine times out of ten okay the reason to fold with STL is a two-fold with neural networks one thing you forgot to scale the input okay, the STM's what happened? So you have those kernel functions we talked about right you forgot to tune the parameters of the kernel function you just took the kernel function as it is and you are trying to use it so that the performance will be bad, so you have to tune the parameters of the kernel function and you have to scale the inputs, if you do not scale the inputs sometimes the performance can be arbitrarily bad.

And this is a problem which SVM do not have and that is one of the reasons they became so much more popular than your networks in late 90s and early 2000s right, so the neural network error surface is fairly complex. So what do I mean by error surface? So what is it so error surface what will be the x axis the y axis the z axis whatever can you describe mathematically, what the error surface is? with respect to what aha on the what parameter is not the inputs okay, so the error surface is something which people have difficulty okay there is areas on I am making a fuss out of it.

The error surface is the function of the error with respect to the parameters okay, so as I change my α and β how does the error change okay, so that is that is the error surface that we are talking about right and so how does the error surface how will it look like for the case of SVM and minimizing something quadratic they are right in terms of β right, so it is actually very nice quadratic thing, so it always has a single Optima right when the optimal hyper plane formulation the very nice thing about it is it has got one Optima and then if you run the optimizer on it you will always get that solution right.

The error cell phase for neural networks if you think about it it's got those stupid exponents in there right your sigmoid there is the derivative of, your sigmoid is in there so the error surface is going to look incredibly complicated right. So it is going to have lots of little valleys right the error surface is going to look something like this, ever even look something like this okay, so if I am doing gradient descent I might come here and get stuck, that looks like I mean whatever direction I tried to go there is increasing. So I might say okay this is the good place to be so I might just stop there right.

I could get stuck here what about here huh very slow or not at all because the gradient is 0 I mean if you are in the middle of a point the gradient is zero because it is flat it is flat, I declared that to be flat okay and so the gradient is zero at that point right and there you go okay and so you might not just move right you are essentially drifting around there you and whatever happens you are not able to make any progress. So the error surface can become really complicated like this right and this is just on one dimension right, they say this is a no one it is a single neuron with one input that is what we have drawn.

So imagine this generalized to a very large dimensional space right mp + mk dimensions right, so the surface can be really complex and again the plethora of solutions were getting out of local optima right, so we are not going to get into most of those and let us tell you one very practical way of doing it essentially do restarts right. So you start off with some random initialization close to 0 right you do gradient descent until you do not change weights very much. Remember those weights right remember those weights and remember the performance. No reinitialize the network again close to zero random weights close to zero.

You say different random see please all right and then rerun the experiments right and again you will go off to some other optima remember those and keep doing it. There are other techniques which people use right, so they can make the, you know there is something cleverer gradient descent techniques right which all of you to get over these local optima not all of them but at least some of the shallow or local optima it allows you to get over easily. So for example this is a shallow local optima right with a little bit of effort I can actually get over and how do you provide their effort.

So think of it from a very dynamics perspective, so people have added something called momentum right, so if you have been moving in a particular direction I have been descending little bit a little bit a little bit okay do not stop just keep going in the direction for some more time right that is momentum right. So in this case these kinds of shallow things you can get over you know the gradient has become a slowed down significantly right this becomes 0 here but I will still be going in the same direction I went for a little while longer because I have momentum it is going to take me forward.

So that is going to get me out of this little valleys but if you are in a deep valley then still cannot get out right but so these kinds of tricks help right and then more recently with all the with deep learning, that one of the reasons that deep learning is become so popular most people do have very powerful gradient based techniques which allow you to navigate the error surface more efficiently a lot to avoid local optima, but allows you to navigate the surface more efficiently right good. So any questions so far, no why well I am going back to my zero yeah when I start restart a little again be small weights right.

I will be very far away from this is optima I have converged to after a lot of training, you maybe not see that, this is one thing which you should get your hands dirty then you will see what I mean. Even small changes in the starting weight configuration can lead you into very different Optima that so there is surface or so complex right and remember I am not just moving in one direction or the other I have a very large dimensional space in which I am moving right. So even though I am taking and I am constraining it to be around zero right, the volume that I can actually start in is very large because of the high dimensionality of the weight space right.

So and each random starting point can be very different because it is also possible that you start in the same location or start very close to the same location. I will end up with the same Optima but that is probability of that happening is very small especially with large networks. So if we do the D start will actually end up with somewhere else. Any other questions so we have till this point we have the exam rate just checking yeah, so that is the question nobody asks how many times do you restart right.

There when you have a budget you just say that okay I am going to restart this many times right and yeah you might have actually re absolute minima but you may still be doing research that is one of the reasons I told you to remember the weights right, it could very well be that the best weight best solution, you got could have been oh the first one and then all the fuel further restarts that you do could actually be leading to worse collisions right. So I am not guaranteeing that we do a restart we will get a better solution the restart just allows you to explore different local optima and pick the one that is best.

It all depends on your budget right I mean if it is it is as expensive as to train the network the first time around right and it is really expensive to train the network if you are doing deep neural networks because the number of parameters are really large runs into several hundred thousand right and therefore doing are start is expensive, we do fewer of them and I also tried to come up with other gradient descent techniques, that allow you to avoid local optima. The whole idea behind simulated annealing is that you would want with some probability of ignoring the gradient right.

So what the whole algorithm here says at every point follow the opposite direction of the gradient right, you would descend the gradient direction you find which is the maximum ascent direction you descend it right. The whole idea behind simulated annealing is to say that no I allow you to ignore the gradient, you can move in another direction in fact you can move in the direction off the gradient also if you want right. So the gradient opposite direction of the gradient is a choice that is given to you can choose that or you need not choose them. As the number of iterations become larger and larger the probability of you choosing the direction of the grade up the gradient direction is going to become higher and higher.

This is essentially what we call the temperature parameter the temperature parameter is very high right if you can think of this particle that is going to be jumping all over the place right, so if the temperature is very high you can move in whatever direction you want you are not necessarily constrained to following the gradient direction and that is the temperature becomes lower and lower you, then you are constrained to follow the direction of the gradient .So the reason is called temperature is because it is actually used in waddling physical systems right, so the so if you look at the Boltzmann distribution which people typically use in this context that is actually a parameter called temperature.

Which behaves very much like this right, so one another way to think of what the simulator and helium will do to your error surface it is like it will pull your error surface is falt if the temperature is very high it is like your inner surface is flat it essentially means I do not have any gradient information. I can move any which way I want right whatever direction I move it looks the same then I can this move randomly and then what happens I slowly start regaining the shape of the inner surface.

So what will happen is first the deepest dink will form okay the shallower ones will still be farther away and the deepest thing will the first tip that will appear in my error surface and likewise as I keep cooling it cooling it cooling it will completely go back to the original inner surface. That is one visual way of thinking about it I mean there are more formal ways of explaining why that visualization works but I do not want to get into simulated annealing today but that's another way of avoiding local optima okay done.

IIT Madras Production

Funded by
Department of Higher Education
Ministry of Human Resource Development
Government of India

www.nptel.ac.in