

Deep Learning
Prof. Mithesh M. Khapra
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 8.11
Lecture – 08
Dropout

(Refer Slide Time: 00:15)

Other forms of regularization

- l_2 regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- **Dropout**

The slide also features the NPTEL logo, a small video inset of the professor, and a footer with the text 'Mithesh M. Khapra CS7015 (Deep Learning) : Lecture 8'.

So, in this module we will look at dropout now.

(Refer Slide Time: 00:17)

The slide displays three neural network diagrams on the left, each representing a different instance of a model. On the right, there is a list of points:

- Typically model averaging (bagging ensemble) always helps
- Training several large neural networks for making an ensemble is prohibitively expensive
- Option 1: Train several neural networks having different architectures (obviously expensive)
- Option 2: Train multiple instances of the same network using different training samples (again expensive)
- Even if we manage to train with option 1 or option 2, combining several models at test time is infeasible in real time applications

The slide also features a footer with the text 'Mithesh M. Khapra CS7015 (Deep Learning) : Lecture 8' and a page number '71/84'.

So, the intuition that we have developed in the previous module which was about ensemble methods, is what that is that ensemble makes sense in most cases. Because, you do not expect the errors of these k models that, you are using to be perfectly correlated. And we saw that, whenever they are not perfectly correlated you are going to get some advantage.

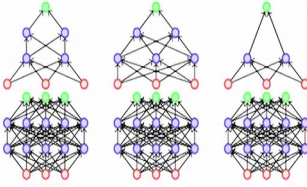
Now, how do you do this in the context of neural networks? So remember, what was bagging multiple instances of the same network trained on different subsets of the data? What is the problem with this in the context of neural networks? Each of these neural networks is very complex training, each of these is going to take time and I going to train k of them is that, fine right.

So you decide ok. Sorry, so, one option that you have is you train several different neural networks having different architectures right, but this is going to be expensive because, you have to train k of them. The other option that you have is, you train the same network, but on different subsets of the data this is also going to be expensive.

So, whatever on sampling techniques you can think, if in the think of in the context of neural networks which are essentially. These 2 techniques different architectures and take an ensemble or train the same architecture on different subsets of the data both of them are going to be expensive right.

So now how do you go about it? And it is not just training time, expensive it even if we manage to train it at test time again. When you are given a test instance you have to pass it through all of these complex neural networks, each of which is going to take some computation, and then take the ensemble of the outputs right. So, even at test time it is expensive it is not just that, that training time it is expensive ok.

(Refer Slide Time: 01:57)

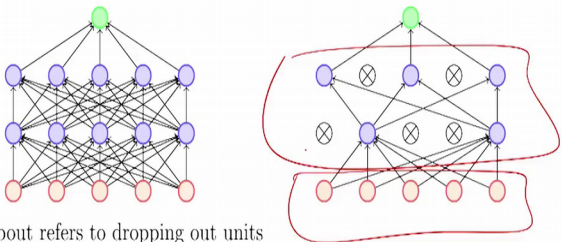


- Dropout is a technique which addresses both these issues.
- Effectively it allows training several neural networks without any significant computational overhead.
- Also gives an efficient approximate way of combining exponentially many different neural networks.

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 8

So now, dropout is a technique which addresses both these issues, which issues? Train time computation as well as test time, computation. So, it effectively allows training several neural network architectures without any significant computational overhead. So, we will see how that works and it just not training time as I said it also allows us to do this quickly at test time.

(Refer Slide Time: 02:21)



- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically $p = 0.5$) for hidden nodes and $p = 0.8$ for visible nodes

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 8

So again let us see, so again here ok. I will get to it when I know. So, drop out actually refers to dropping out units from the neural network ok.

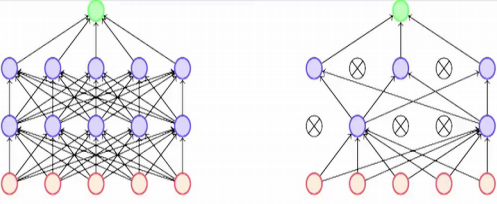
So, this is my original neural network and I am just talking about one neural network, forget about ensembles just one neural network is what I have. Now what dropout says this. You dropout some units from this neural network; that means, dropout some neurons and when I dropout some neurons, I am also going to drop out the incoming and the outgoing edges. Otherwise, where are they headed right, so I am just dropping out.

So basically, what is effectively happening here? I am getting a new network architecture right. At least that is clear that is what dropout effectively does, but I have already made a case that I do not want so many architectures. That because, it is a headed to train all of them and again a test time I have to pass it through all of them right.

So, I need to still fill that gap, but drop out says that, drop some units and you will get a new architecture. But how does that simplify life, we will see that. And now each node is actually retained with a fixed probability for the hidden nodes and even further input nodes.

So, then we were not wrong in actually dropping out the visible node. Because, you can do dropout at the visible nodes also ok. Anyways yeah so for, the hidden units you would drop them with a probability 50 percent and the input units you will drop them with a probability of 20 percent typically it again is some hyper parameter that you will have to tune, but typically this is what you will do and I hope you see that dropping nodes from the hidden unit. From the input unit is same as corrupting the input data right it is same as adding noise to the input data is that fine ok.

(Refer Slide Time: 03:58)



- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed? 2^n
- Of course, this is prohibitively large and we cannot possibly train so many networks
- **Trick:** (1) Share the weights across all the networks
(2) Sample a different network for each training instance
- Let us see how?

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 8

So this is the idea, now let us see how to actually implement this idea. So, suppose a neural network has n nodes using the dropout idea each node can be retained or dropped, an example in the above case I have dropped some 5 nodes to get a thinned network.

So, if there are n nodes what are the total number of thin networks that I can get from it? And so; that means, I can get 2 raise to n different neural networks. Am I happy about this, or sad about this? Sad, there is just too many neural networks. How can I train them actually right?

So how do I do this? I am just creating a lot of suspense without giving you the answer ok. So, first trick is, share the weights across all these networks ok. We will see what; that means, and the second trick is sample a different network for each training instance ok. None of which is clear at this point, I can see I can read your faces I am good at it ok. So, let us see how to do that.

(Refer Slide Time: 04:50)

- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active

So, we initialize all the parameters of the network randomly or whatever may be used and start training. When I start training, I will pick up the first training instance or the mini batch or whatever I am doing we apply dropout resulting in this network ok.

What will I do and they forward prop carlo, forward propagation right ok. Now, we compute the loss and back propagate how? Some weights are missing rate how do I do back propagation now. I have deliberately dropped up some of these connections, they did not participate in the forward propagation. This back propagate, which are the parameters which will update now? Only the ones which actually participated right.

So, I will just do back propagation. Just look at the red arrows, I will just do it over the paths which are actually present in my network fair enough right. That is what you meant by normally ok, that is normal ok. So, I will just do it over the weights which actually participated that is fair enough that is the only thing you could; obviously, do.

(Refer Slide Time: 05:51)

- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 8

Now, I take the second instance, again I apply dropout and quite naturally I will get a different thinned network as you see the figure 3 in this slide ok. What would I will do now?

Student: Forward propagation.

Forward propagation, then compute the loss [FL] back propagate to compute the loss ok, and then.

Student: Back propagate.

Back propagate again back propagate only to the.

Student: Active nodes.

Active nodes, so these other nodes which will get activated ok. So, what is happening here, is now trying to relate it to what we were doing in bagging right; where we are trying to train these different networks on different subsets of the training data right. Do you see something similar happening here? There are many such thinned networks; each time I am sampling a different network and updating it right.

So, it is equivalent to training these large number of networks on different subsets of the data right, but then the problem is that some of these networks may never even get

sampled. There are 2^n of those, my amount of data is definitely to be less than 2^n .

So, some of these networks might just not even get sampled. Then, what is happening? Or they would get sampled very rarely right. For example, what is the probability that again? I will end up with the same network we are computing it ok. Good, it is very less ok, I am fine with that at 730 right.

So, it is a very less right. So, it is quite likely that this network will never be sampled again; that means, for that network the parameters are getting updated very few times, am I fine with it. Yes, I am. Why? Because the same weights will get updated for a different network; I am just using the same weight matrix throughout remember that, my W matrix or W_1 , W_2 is the same throughout.

It is just that at different depth subsets different instances; I am just touching some portions of this W_1 and I am not touching the other portions of W_1 . So now, what would happen, so I have shown you 2 training instances right. What would happen to the weights which were active for the first training instance as well as, the second training instance? It will get updated twice and which are active only once?

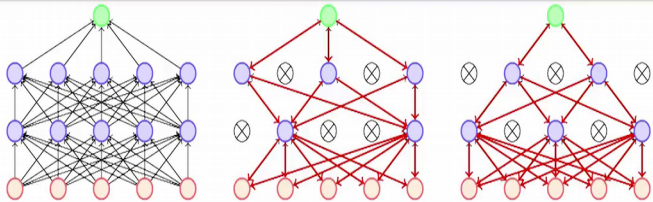
Student: (Refer Time: 07:59).

Only once right. So, over a period of time many of these weights are shared across all these networks that, I am sampling right. So, even though a particular network is sampled only a few times, its weights will get updated many times via these other networks which are similar to it. Do you get that, how many of you get this? Ok; good, so what is happening, I will just repeat that I have just one weight matrix. I am sampling a thinned out network which only uses some of these weights.

So for that training instance, I will update those weights. Now, I know that the likelihood of the same network getting sampled again is very less, but I do not care about it because, I could sample a different network, but I am sure that some of these weights will again repeat in that right. And in that, I told they will get updated. So, even though each of these networks is seemingly getting very few updates. Overall, all the weights shared by these networks are getting updated as much as they should be is it fine?

Everyone gets this idea? Ok, fine and while I am also taking care that, similar things like early stopping or weight regularization l_2 regularization where, I am not allowing a single weight to continuously grow or something. Otherwise because, these weights will be off for some networks. Is that fine? You see the connection between early stopping, l_2 regularization and this is that ok?

(Refer Slide Time: 09:18)



- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights
- If the weight was active for both the training instances then it would have received two updates by now
- If the weight was active for only one of the training instances then it would have received only one updates by now
- Each thinned network gets trained rarely (or even never) but the parameter sharing ensures that no model has untrained or poorly trained parameters

Mitesh M. Khurana CS7015 (Deep Learning) : Lecture 8 76/84

And so, each thinned network gets trained rarely or sometimes even never, but I am not worried about it. Because, it is weights will get updated through some of these other thin networks.

(Refer Slide Time: 09:27)

- What happens at test time?
- Impossible to aggregate the outputs of 2^n thinned networks
- Instead we use the full Neural Network and scale the output of each node by the fraction of times it was on during training

This is all finite training time. At training time what is happening is this is one of these blue guys introduce on with the probability p ; that means, the weights going out of it, who are available with a probability p right and other times they were not available.

Now, what do I do, it test time? I cannot let me finish this ok. I cannot take an ensemble of d ok, the answer would have been that, at test time instantiate all these 2 raised to n networks pass the training passed the test example, through all of them and then take an ensemble right, but of course, that is probabilistically expensive. So, what will I do at test time? What is the simple trick that I will do? So, he says that just use this network.

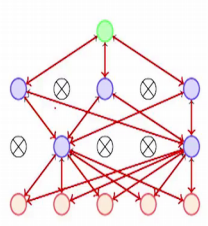
And just use the final matrix, that you had no, but then you have guessing out of the 2 raised in the sample, some small number of those and do it. Actually, dropout uses something very simple than this. What it says is, that each of my nodes was present only p fraction of the times in the training data ok; that means, one way of looking at it is that. So, imagine that you could think of this as the analogy is that all these nodes are participating in a discussion right where they trying to see how to do this job properly, but with probability p they all sleep off right.

So, at the end of the meeting, you will trust each of them only with probability p . So, that is the simple trick with dropout uses; it says that just scale their weights by p . Because, that is how much I trust this node; it only participated in p fraction of the decisions. So, that is the confidence that I have in it.

So, if it is saying that with W_1 weight, do this I will only do it with p into W_1 weight does that make sense. And there is again a squared egg with vacuum kind of explanation for this ok. Which was there in the quiz, last year which is very convoluted, it does not really give you the true picture because, you can derive some math and so, that this is mathematically proper, but that again works in very specific conditions, but at least if you get the intuition that is fine that what we are saying is that. These nodes will leave an active a few number of times. So, I will only trust them that much and I will just scale their weights by that factor.

So, at test time I will just pass my test instance through one network, which is the full network with the weights scaled according to the rule which, I just said that is exactly what dropout does. How many if you understand this? And the final interpretation of dropout right.

(Refer Slide Time: 12:03)



- Dropout essentially applies a masking noise to the hidden units
- Prevents hidden units from co-adapting
- Essentially a hidden unit cannot rely too much on other units as they may get dropped out any time
- Each hidden unit has to learn to be more robust to these random dropouts

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 8

So, what dropout actually does is, we will apply some kind of masking noise to the hidden units right. Since the same as seeing that you are computing the hidden unit, but then you are masking it off ok.

So, what is the effect of this? I will give you the answer and I like; I like you to think about it. The answer is that it prevents the neurons from becoming lazy. What do lazy people do? They depend on others yeah actually yeah; they depend on others now. So, let

me answer that give the answer for this and then tell me whether that is still contradict ok.

So, let us see right consider this layer of neurons. All of these are collectively responsible for what happens to this guy right; Now you see, what I mean by neurons becoming lazy? I could just see ok. I will not give my input these other neurons will take care of it right, they will adjust their weights.

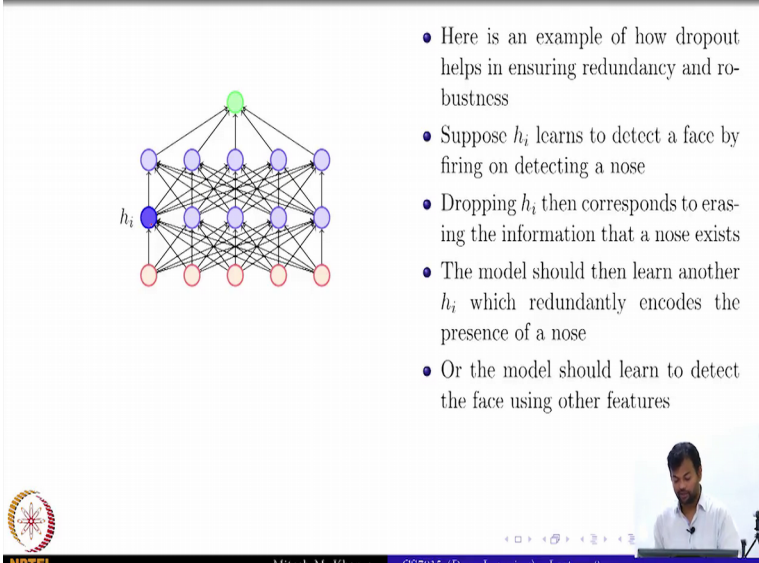
So, that they eventually, it will fire or not fire or whatever right you see that could happen, but now these neurons cannot rely on their neighbors. Because they do not know when their neighbors are going to ditch them right? They will suddenly drop off ok. And now I was waiting for my neighbor to actually do something and he is not going to do it. So, I have to be alert always do you get the analogy right.

So, these guys are collectively responsible for something and they know that some people in the collection are going to betray them. So, each of them has to be more careful; so, the more technical term for this is that does not allow the neurons to co adapted right.

So, it does not allow them to get into this mutual agreement that you take care of certain things, I will take care of certain things and together we will do the job right. You do question 1, I will do question 2, I am ok, it does not allow them to do this.

So, let us just concretize that intuition a bit for. So, essentially a hidden unit cannot rely too much on other units as they may get dropped out at any time. Each hidden neuron has to learn to be more robust right. It has to do the job as if it is the only guy responsible for the job and let us consider one of these neurons h_i .

(Refer Slide Time: 14:00)



- Here is an example of how dropout helps in ensuring redundancy and robustness
- Suppose h_i learns to detect a face by firing on detecting a nose
- Dropping h_i then corresponds to erasing the information that a nose exists
- The model should then learn another h_i which redundantly encodes the presence of a nose
- Or the model should learn to detect the face using other features

NPTEL
Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 8

And let us see that, a h_i learns to detect faces; sorry, it learns to detect a nose. So, I am trying to do face detection whether, an image is about a face or not and h_i is the feature which fires. If there is a face somewhere, if there is a nose somewhere in the image is that fine.

Now, if all these guys start acting lazily ok. This guy is going to detect a nose; that means, definitely face will be there. So, I do not need to do anything right, what would happen now? Suddenly this guy is going to go away dropped out; so then, these other guys need to do one of 2 things; either add redundancy; that means, one of them should also take responsibility for detecting a nose or do it in a different way. Take responsibility for detecting the lips or the eyes or some other part do you get that? Right because, you know that I cannot co adopted with my other neurons; I cannot say that ok, in these front facing faces you just detect the nose and will be done and we will all keep quiet right.

I do not know whether, you will do your job properly. So, I will have to add more redundancy. You detect a nose, I will also detect a nose or you detect a nose and I will detect something else which helps detecting the feature right. So, that is why these networks become more and more robust as you add this proper ok.

(Refer Slide Time: 15:26)

Recap

- l_2 regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout

NPTEL Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 8

So, that is all that I had to say I still do not know whether I have answered your question or not. All of them try to detect nose; see as long as that helps reducing the final loss it is fine. It is just the case that you would have some training images, where the nose is not visible maybe that person is drinking something right.

So, for at least for those training instances someone else has to take care that you detect from the other images right. Otherwise, a loss would not be 0 for that training instance. So, as long as you have some training instances see, if all your training instances can be detected just by detecting the nose. Then, there is nothing wrong in all of them trying to detect the nose. So, if the training it is like that it will happen, but the hope is the training data is not like that right; is that fine? So we will end here.