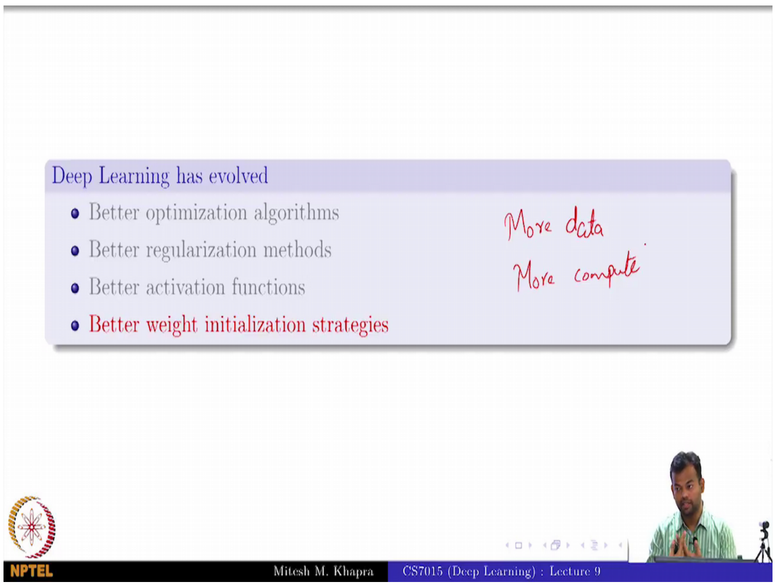
Deep Learning

Prof. Mitesh M Khapra
Department of Computer Science Engineering
Indian Institute of Technology, Madras

Module – 9.4
Lecture – 09
Better Initialization Strategies

So, in this module we will talk about Better Initialization Strategies.

(Refer Slide Time: 00:17)



Deep Learning has evolved

- Better optimization algorithms
- Better regularization methods
- Better activation functions
- Better weight initialization strategies

More data
More compute

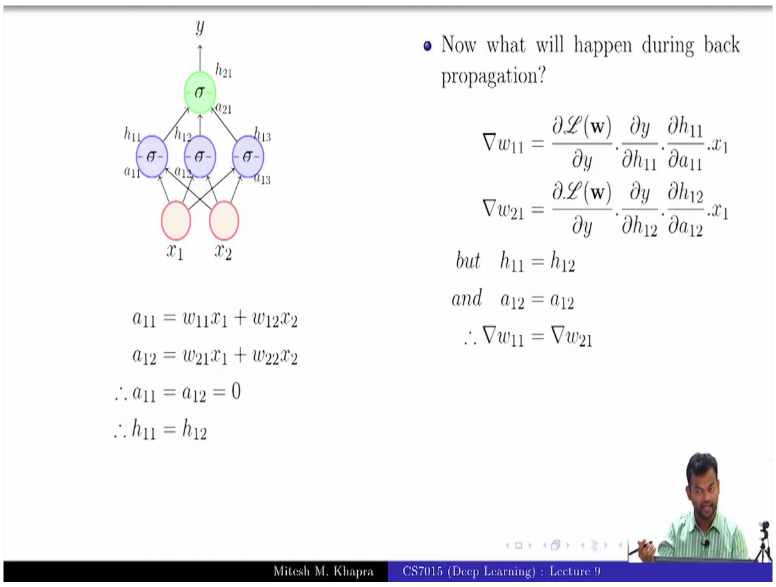
So, this is where we are in the story right. We saw that deep learning has evolved and at least these are the 4 things which have happened. So, we have by the way this slide is incomplete. What are the other things which have happened actually which you already said in the beginning 2 more things which are not technical, but which happened more data right.

And more compute, but these are not really technical in the sense that, I mean this just happened we have large amounts of data; that means, more data means, what if you are more data for training. You would have complex networks, but not over fit right because, you have so many, so much of data right. And more compute of course, it speeds up some of these matrix computations which happen.

So, remember in a deep neural network most of the things which are doing are matrix, matrix operations right. You are taking are that is what exactly you did in your back propagation assignment. You did a lot of matrix vector computations and so on, and the advent of g p use this. Became, very very fast rate orders of magnitude fast.

So, this 2 which are here as nothing much to talk about that is just something we all understand, what has happened they. And so now, we will talk about better weight initialization strategies.

(Refer Slide Time: 01:27)



$a_{11} = w_{11}x_1 + w_{12}x_2$

$a_{12} = w_{21}x_1 + w_{22}x_2$

$\therefore a_{11} = a_{12} = 0$

$\therefore h_{11} = h_{12}$

- Now what will happen during back propagation?

$$\nabla w_{11} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} . \frac{\partial y}{\partial h_{11}} . \frac{\partial h_{11}}{\partial a_{11}} . x_1$$

$$\nabla w_{21} = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} . \frac{\partial y}{\partial h_{12}} . \frac{\partial h_{12}}{\partial a_{12}} . x_1$$

but $h_{11} = h_{12}$

and $a_{12} = a_{12}$

$\therefore \nabla w_{11} = \nabla w_{21}$

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, let us start with this question right. We will take this network and we will ask this question what happens if we initialize all the weights to 0. I like it when you all try to visualize it and. So, you have to see what happens right. So, let us start with a 1 1, which is w 1 1 x 1 plus w 1 2 x 2.

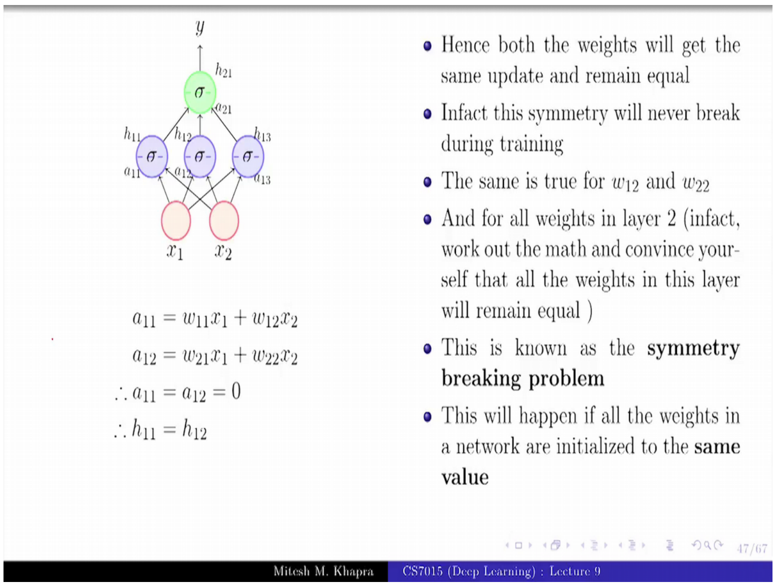
So, I always start small. I do not try to see what will happen everywhere, just start with one neuron and see what happens take a 1 2. What would a one month's value b? If all the weights are initialized to 0 0 and a 1 2 again 0 right and same for a 1 3 it is all the way all the neurons in this layer are going to be 0, till is it.

So; that means, they will all get the same activation. So, if the a's are same the h s are also going to be same and that is obvious irrespective of what non-linearity you use now what will happen during back propagation what will delta w 1 1 b. This again I do not know why you do this ok, anyway that will be raiser into x 1 right.

So, remember that the gradient is always proportional to the input. And you have somewhere along the lines along the chain rule. You have this h 1 1 and a 1 1 ok. Just remember that and what would delta at gradient of w 2 1 b is that fine.

Now, can you see some things on the left hand side and make some comments on the gradients. We have seen that a 1 1 is same as a 1 2 and h 1 1 is same as h 1 2; that means, these gradients are going to be equal right; that means, the weight started off at the same value. They are going to get the same updates and again remain at the same or different value, but this same right. Then of course, move from where you started will not be 0 anymore, but they will all be at the same value.

(Refer Slide Time: 03:24)


y
h_{21}
σ
a_{21}
h_{11}
h_{12}
h_{13}
a_{11}
a_{12}
a_{13}
x_1
x_2

$a_{11} = w_{11}x_1 + w_{12}x_2$

$a_{12} = w_{21}x_1 + w_{22}x_2$

$\therefore a_{11} = a_{12} = 0$

$\therefore h_{11} = h_{12}$

- Hence both the weights will get the same update and remain equal
- Infact this symmetry will never break during training
- The same is true for w_{12} and w_{22}
- And for all weights in layer 2 (infact, work out the math and convince yourself that all the weights in this layer will remain equal)
- This is known as the **symmetry breaking problem**
- This will happen if all the weights in a network are initialized to the **same value**

47/67

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

Both the weights will get updated with the same value and they will remain equal so, but fine as I keep training they will move away from each other right. This is what I told you, when you feed in the first example take a both the weights remain the same, but now if you feed another example and you keep feeding batches there. There is no dearth of data that, you have and eventually these weights will move away from each other.

The update is the same again. The weights are the same again the same situation will hold right. Again your w 1 1 x 1 plus w 1 2 x 2 is going to be the same as w 2 and x 1 plus w 2 2 x 2 and the same argument repeats.

How many if you get this,. So, once you initialize the weights to 0 in all subsequent iterations, the weights are going to remain the same. I mean, they will move away from 0, but they will all be equal and this symmetry will never break during training. So, what actually is happening in terms of the capacity of the network this is same as.

Student: Single line tying the weights.

The same as tying the weights so, this symmetry will never break during training. So, asking what is the net effect which is happening?

So, you have so many weights in your layer, but all of them are moving together will so in essence. You do not have the same freedom as you have with n different weights right. Here, in some sense unintentionally tied them because, it started off with the same value. Now you are all moving at the same values rate you are all going to the same value. So, you do not really have the amount of freedom that you would actually expect with n different parameters all of you get this.

And the same is true for w 1 2 and w 2 2 also which are the weights connected to the second neuron and this is. In fact, true for all the weights in layer 2 you can actually mathematically verify it; that means, whatever this small analysis that I did here. Just go back and do it for all the weights in the network and you will see that all of them. If you are going to initialize them to 0 all of them are going to remain equally.

This is known as this symmetry breaking problem this is are known problem. This is existed much before 2006 and so on. If we initialize all the weights to 0 you will have the symmetry breaking problem is there. Anything sacrosanct about 0 or would this happen even if you initializer to same, but non 0 values and that should have been cleared from the iteration right. Because, after the first iteration we were at non 0 weights and after that the story repeated right.

So, even if you initialize it to non zero weights the same story is going to report repeat so; that means, as long as you initialize all the weights to the same value. You are going to end up with this symmetry breaking problem ok, which is not good. So, what is it that we have learnt about initializing weights?

Student: (Refer Time: 6:32).

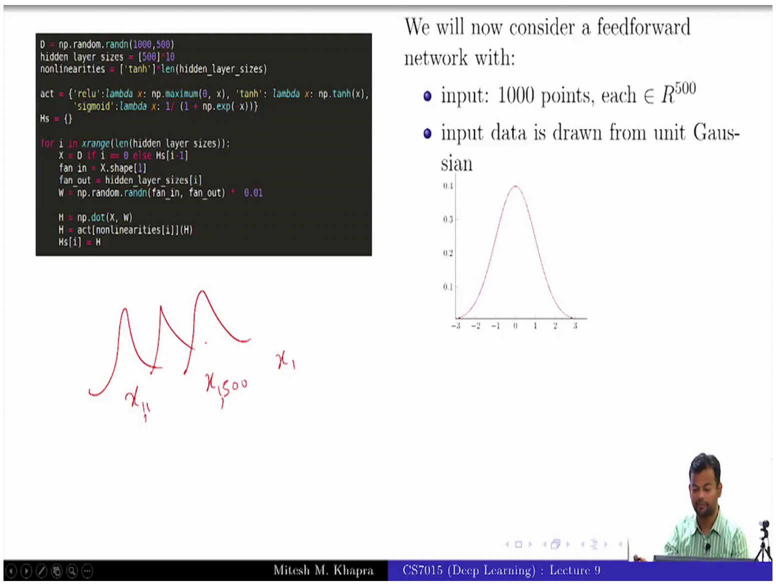
Definitely do not initialize all weights to 0, definitely do not realize them to the same value ok. This is the first thing that you have learned. So, we are seeing different ways of not making the light bulb and then, we will come to a way of making it. So, 0 and equalist no bad yes, some weights will not get updates in that case right.

So, then that that should be fine so, that is the other thing I wanted to make at some point right, these 4 things right initialization optimization regularization and activation function. These are not independent things they are all tied to each other.

So, as you said now if you use regularization then probably you could be a bit careless with the initialization. Even if you had initialize the weights together drop out would have ensured that, some of these weights are not active at a particular training instance; that means, they will not get weight updates; that means, they will move away from the other weights right.

So, that is this is not that only one of these things can be done right. You are going to use a combination of these things, but while analyzing them, we will just look at one of these things. Assuming that, the others are not being, right so will assume that we are not using drop order anything is that fine.

(Refer Slide Time: 07:01)



```
D = np.random.randn(1000,500)
hidden layer sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden layer sizes)):
    X = D if i == 0 else Hs[i-1]
    fan in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) *  0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

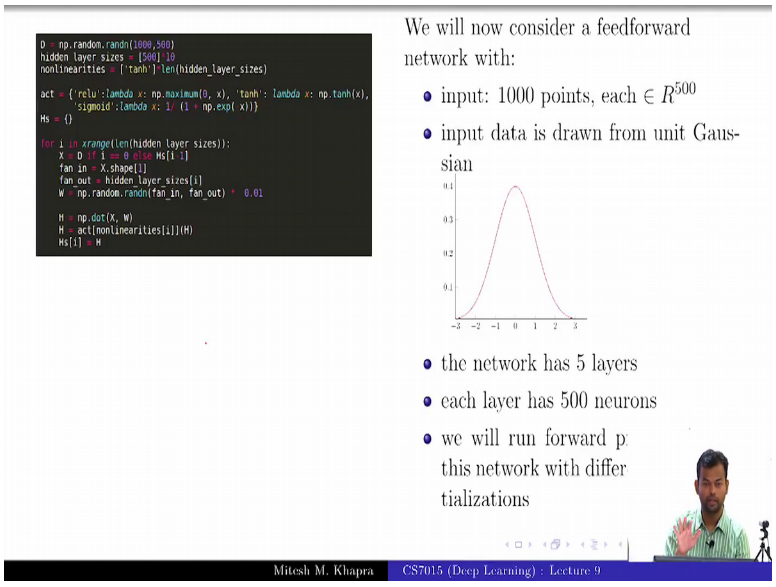
We will now consider a feedforward network with:

- input: 1000 points, each $\in R^{500}$
- input data is drawn from unit Gaussian

So, this at least this in practice you are not supposed to initialize the weights to 0s and equal values that is what we have learned so far. Now for the rest of the to convince you about some other weight initialization methods. What I am going to do is? I am going to take a feed forward network, where you have as input some thousand points? Each of this point is 500 dimensional. And the input data is drawn from a unit Gaussian. What I mean by that is? You have this x 1 2 x 1 500 rate for the data instance one.

So, all of these 500 dimensions come from a unit Gaussian is that fine. So, this comes from a unit Gaussian. This comes from a unit Gaussian and so on ok. That is what I am going to assume?

(Refer Slide Time: 07:50)



```
D = np.random.randn(1000,500)
hidden layer sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x: np.maximum(0, x), 'tanh': lambda x: np.tanh(x),
       'sigmoid':lambda x: 1/ (1 + np.exp(-x))}
Hs = {}

for i in xrange(len(hidden layer sizes)):
    X = D if i == 0 else Hs[i-1]
    fan in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) *  0.01

    H = np.dot(X, W)
    H = act[nonlinearities[i]](H)
    Hs[i] = H
```

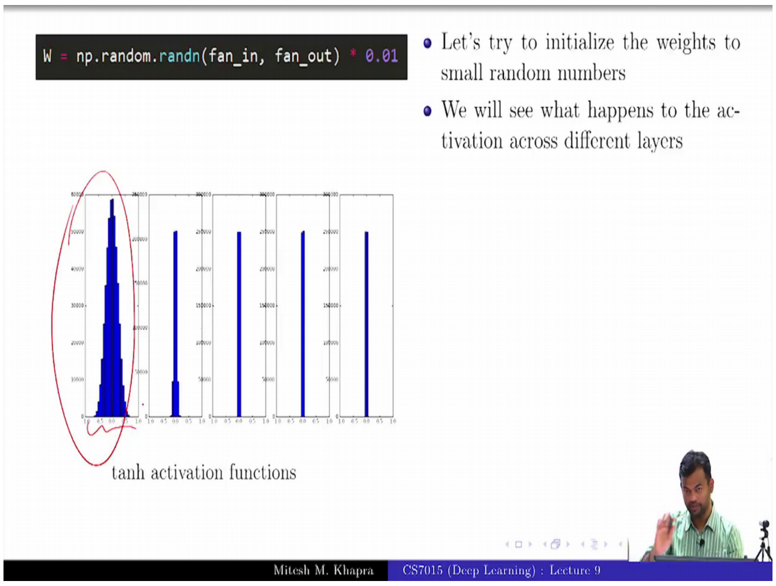
We will now consider a feedforward network with:

- input: 1000 points, each $\in R^{500}$
- input data is drawn from unit Gaussian
- the network has 5 layers
- each layer has 500 neurons
- we will run forward p this network with differ tializations

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

And the network has 5 layers each layer has 500 neurons. The input is 500 neurons each of the 5 layers is also 500 neurons. And now, we will run forward propagation no backward propagation, no loss nothing and I am not even giving you an objective. This is just some input and I just want to see, what happens up to the last layer? I am not even bothered about the actual last layer; that means, I am not trying to minimize any cost entropy squared error loss anything.

(Refer Slide Time: 08:18)



```
W = np.random.randn(fan_in, fan_out) * 0.01
```

- Let's try to initialize the weights to small random numbers
- We will see what happens to the activation across different layers

tanh activation functions

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, let us try a few initialization strategies. So, we realize 0 is not good realize equal is not good. So, let us try some random initializations, but small weights and this is my way of randomly initializing with small weights.

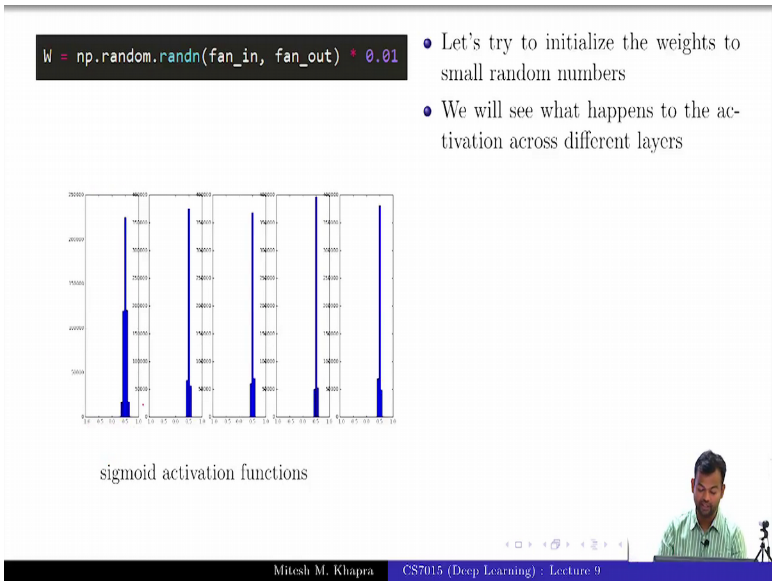
So, my W is a matrix of size fan in into fan out rate, which is n cross n ok the number of weights coming in and out rate. So, n cross n and I am drawing from a uniform distribution and then, multiplying it by point 0 1 which ensures that, all the weights are very small you get the setup now with this, I am going to start with the input and then keep doing these transformation.

So, I will do W transpose X plus b pass it through a sigmoid and do this 5 times. Because, I have 5 deep layers now, this is what happens to the activations across the 5 layers? So, the first layer remember that we had drawn from a unit Gaussian right. So, that is what the data input data looks like. So, this is the first layer which is the input data basically and then this is what happens across the different layers.

So, what is actually happening and this is for the tan h activation function. There is no variance in the output of. So, this tells me, so this basically tells me that, for all the neurons what is the average value that, I am getting right and I should ideally get some histogram that for some neurons I am getting the value minus 1 for some neurons minus 0.9 0.8 and so on, but what this is telling me is as, I keep progressing across the layers all the neurons have very similar values and they are all close to 0.

This is what actually happens in practice; I have just actually run it and computed the histogram.

(Refer Slide Time: 10:02)



```
W = np.random.randn(fan_in, fan_out) * 0.01
```

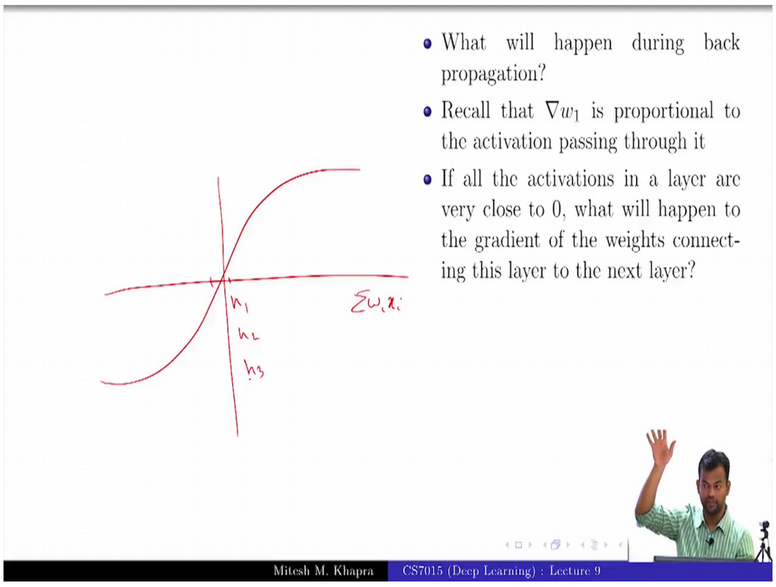
- Let's try to initialize the weights to small random numbers
- We will see what happens to the activation across different layers

sigmoid activation functions

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

And if I use sigmoid activation functions, again something similar all the values tend to be close to the center which is 0.5 right so, this is 0.5 and although I had started with a nice Gaussian distribution.

(Refer Slide Time: 10:15)



- What will happen during back propagation?
- Recall that ∇w_1 is proportional to the activation passing through it
- If all the activations in a layer are very close to 0, what will happen to the gradient of the weights connecting this layer to the next layer?

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

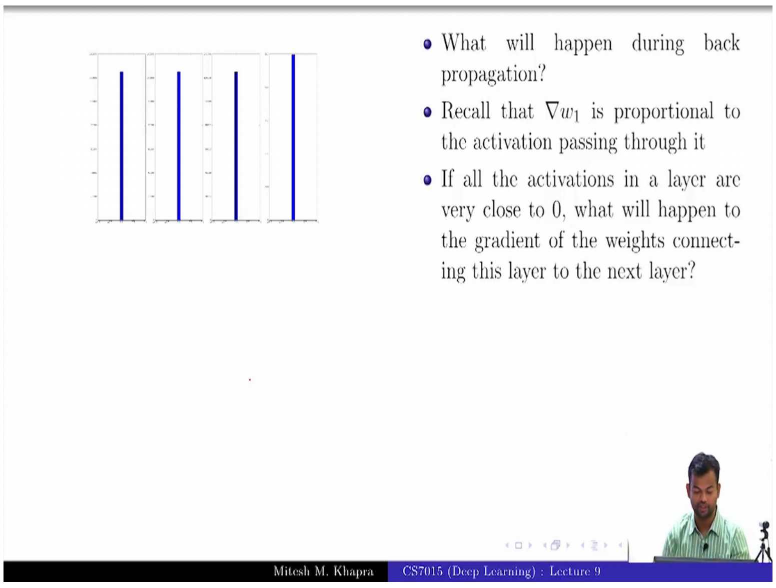
Now, what will happen during back propagation? So, do not try to think for now that why, this happens I am just telling you have actually run the code. And this is what happens? Now given that, this has happened. What will happen during back propagation? So, all the activations in a layer are very close to 0, all the gradients are going to be close to 0; that means, no gradients are going to flow back; that means, which problem are we dealing with vanishing gradient problem.

So, if you initialize your weights to very small values and this is easy to see in the case of tan h. So, for tan h this is my function right and this is 0. Now remember that, this is w I summation w I x I, if all my weights are close to 0 or very small values. What is summation w I x I going to be it is going to lie somewhere here right.

So, all these inputs are actually going to be very close to 0. Now if my inputs are going to be close to 0, I know that during back propagation at some point my gradient is proportional to the input. That, I have given and when I say input here I mean layer 1, layer 2, layer 3 and so on right. How many if you get this so far? Ok.

So; that means, all my inputs are very close to 0, now my gradients are actually proportional to the input. So, all my gradients are also going to be close to 0; that means, my gradients are vanishing right because, remember that across 5 layers you will have these products of gradients right. All of them are very close to 0. So, you will end up with something very close to 0 raise to 5. How many if you get this? Right, so our gradients are going to vanish.

(Refer Slide Time: 12:06)

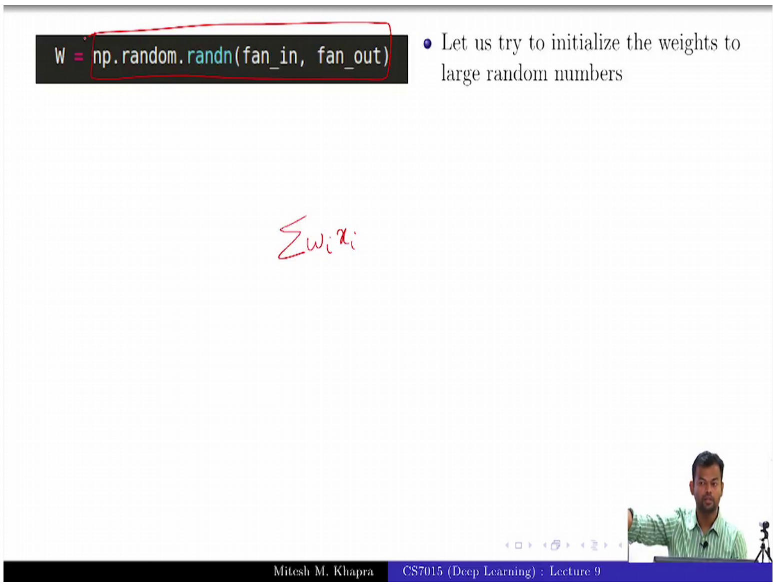


- What will happen during back propagation?
- Recall that ∇w_1 is proportional to the activation passing through it
- If all the activations in a layer are very close to 0, what will happen to the gradient of the weights connecting this layer to the next layer?

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

If you do this very small initialization of the weights and that is exactly what is happening? So, this is the histogram for the gradients and I see that, all my gradients are actually very close to 0; that means, no effective training is happening. My weights are not receiving any updates this is what happens in practice. If you initialize your weights to very small values ok.

(Refer Slide Time: 12:25)



```
W = np.random.randn(fan_in, fan_out)
```

- Let us try to initialize the weights to large random numbers

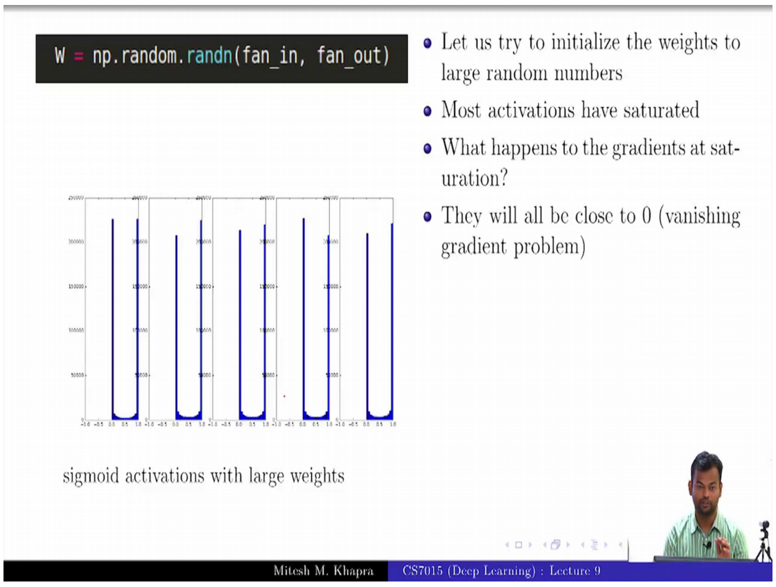
$\sum w_i x_i$

Now, let us try to do the opposite of this very small values did not work. So, let me try large values and for large values I just sample from the uniform distribution. I will get some numbers between 0 to 1. Now can you guess what will happen? Remember summation w I x I all your weights are large. So, why am I saying that number between 0 to 1 is actually large? It is not by all practical because; this is going to give me this function is actually going to give me numbers between 0 to 1. Why am I calling them large weights?

Student: (Refer Time: 12:54).

No, I will I just talked about the weights, assume there is no biases, how many of you get that answer? Remember there are 500 neurons. So, if you have 500 small values, that summation is going to be still large right. If you all of these are 0.4 or 0.5. Which still looks small, but if you have 250 of these or if, you have 500 of these the resultant sum could be somewhere of the order of 250 right. And that is very large because, if you pass that to a sigma and neuron, what will happen saturation right. So, you get this why I am calling these weights as large? Ok.

(Refer Slide Time: 13:30)



```
W = np.random.randn(fan_in, fan_out)
```

- Let us try to initialize the weights to large random numbers
- Most activations have saturated
- What happens to the gradients at saturation?
- They will all be close to 0 (vanishing gradient problem)

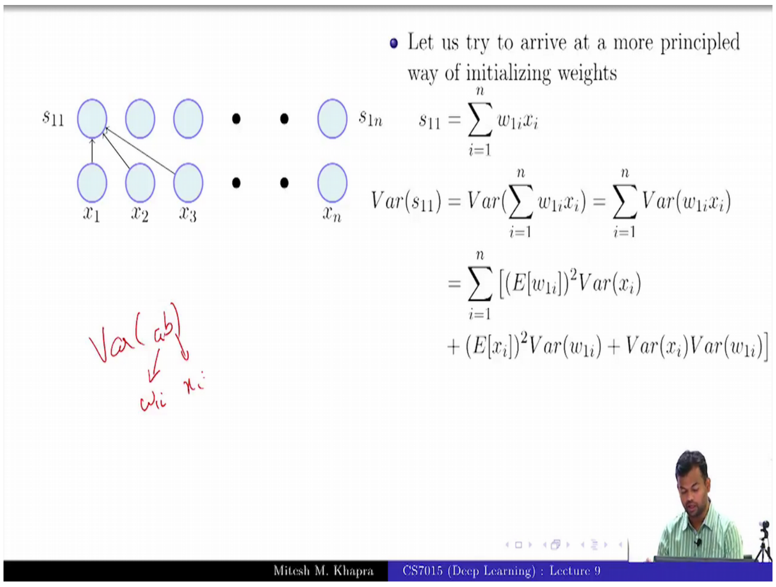
sigmoid activations with large weights

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, and this is actually what happens. So, when I have these tan h activations across all the 5 layers. I observe that my neurons saturate I either get minus 1 as the output or plus 1 as the output. And same thing happens, if I use sigmoid activations I either get 0 as the output or I get 1 as the output right. Neurons saturated means, what will happen gradients will vanish right.

So, even if you initialize the weights to very large values all your gradients are going to be close to 0 because, they are going to vanish and again you have a problem. So, what have we seen so far? 0 is not good, equal is not good, small weights is not good, large weight is not good. Then, what do we do go home ok.

(Refer Slide Time: 14:15)



- Let us try to arrive at a more principled way of initializing weights

$$s_{11} = \sum_{i=1}^{n} w_{1i}x_i$$

$$Var(s_{11}) = Var(\sum_{i=1}^{n} w_{1i}x_i) = \sum_{i=1}^{n} Var(w_{1i}x_i)$$

$$= \sum_{i=1}^{n} [(E[w_{1i}])^2 Var(x_i)$$
$$+ (E[x_i])^2 Var(w_{1i}) + Var(x_i)Var(w_{1i})]$$

So, let us see what to do? So, let us try to arrive at a more principled way of initializing weights and this again do not should the messenger. I am going to give you a proof under certain assumptions ok. So, just bear with me, I just tell you what those assumptions are going to be as we go along. So, as I said right.

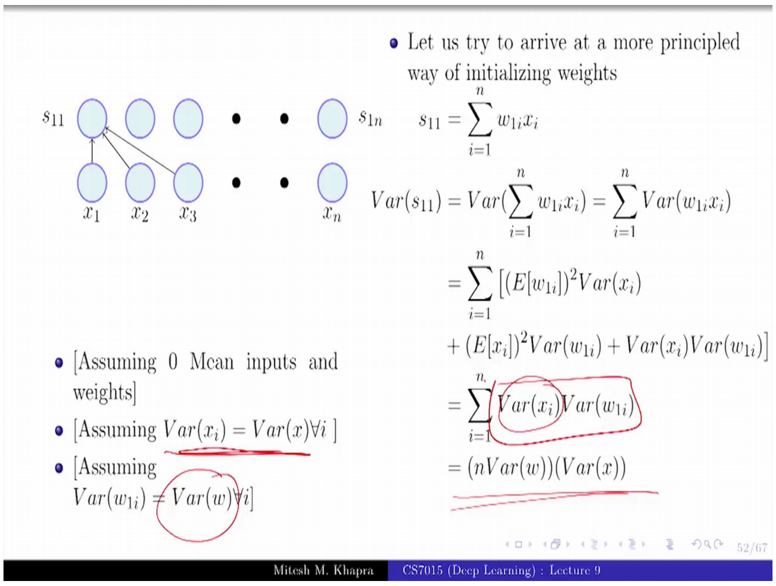
So, I mean you would argue that in practice these assumptions do not hold true, but at least they give us some insights into, what is happening rate? What is the overall idea behind? What is being proposed? So, let us start with that. So now, consider this deep neural network and I am just considering the first layer of it. Where I have this neuron s 1 1 and I am talking about things before the activation.

So, I know that s 1 1 is equal to this quantity right. So, all the incoming weights to the first neuron which is w 1 i into x i. Now for some reason, I am not telling you why I am interested in the variance of this. Can you tell me why I am interested in the variance? What did you see in the previous examples? There was no variance right, there was hardly any variance. So, let us see what happens if you compute the variance of this.

So, I am just taking the variance formula a variance of a sum is equal to the sum of the variances right. This is of the form variance of a into b, where a is w 1 i and b is x i. What is the formula for this? Or if you know it or do not know it do not care. So, this is the formula ok.

So, this is the generic formula for variance of a into b. Where you have to assume that a is w 1 i and b is xi. So, this is just a formula, there is no trick. Here, no matter I mean, no nothing fancy here just apply the formula for variance of a b and substitute a is equal to w 1 i and b is equal to x i.

(Refer Slide Time: 16:07)



s_{11}
s_{1n}
x_1
x_2
x_3
x_n
Mitesh M. Khapra
CS7015 (Deep Learning) : Lecture 9
52/67

- Let us try to arrive at a more principled way of initializing weights

$$s_{11} = \sum_{i=1}^{n} w_{1i}x_i$$

$$Var(s_{11}) = Var(\sum_{i=1}^{n} w_{1i}x_i) = \sum_{i=1}^{n} Var(w_{1i}x_i)$$

$$= \sum_{i=1}^{n} [(E[w_{1i}])^2 Var(x_i)$$
$$+ (E[x_i])^2 Var(w_{1i}) + Var(x_i)Var(w_{1i})]$$
$$= \sum_{i=1}^{n} Var(x_i)Var(w_{1i})$$
$$= (nVar(w))(Var(x))$$

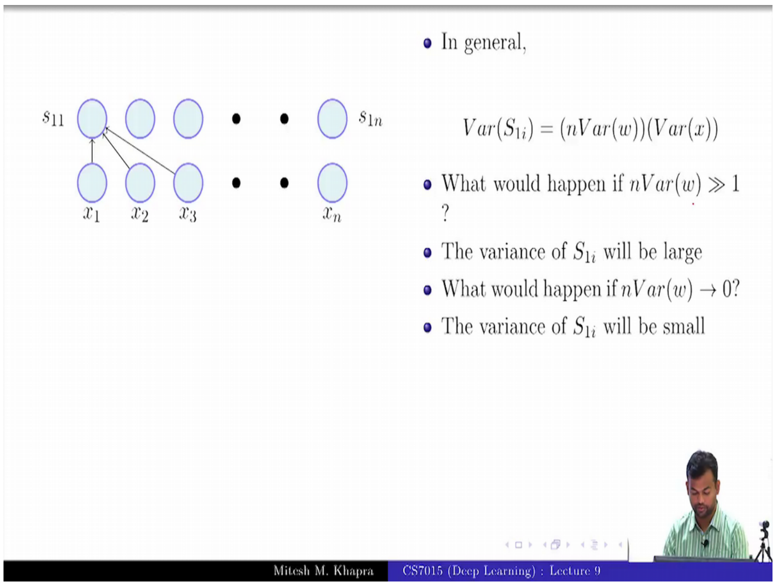
- [Assuming 0 Mean inputs and weights]
- [Assuming $Var(x_i) = Var(x) \forall i$]
- [Assuming $Var(w_{1i}) = Var(w) \forall i$]

Now, I will assume that all my inputs are 0 mean fine. We have been assuming that, forever and all my weights are also from 0 mean ok. What is the effect of that which quantities will disappear? This will disappear because mean as 0 means the expected value of the weight is 0. So, the square of that is 0 an expected value of the input is 0 the square of that is 0 ok.

So, what am I left with summation? Where I variance of x I into variance of w 1 I? Ok now I am going to assume that the variance of x I is equal to the variance of x; that means, it is the same for all the i's. So, I add this remember I add these 500 inputs.

So, I am assuming that for all the inputs the variance is the same. They all come from a similar variance distribution. And I am also going to make the same assumption for the weights fine and then I end up with this neat formula that the variance of s 1 1 is equal to n times the variance of w into variance of x right. Because, I assumed that all these terms are equal and there are n such terms. Everyone is fine with the match so far with the assumptions that we have.

(Refer Slide Time: 17:17)



- In general,

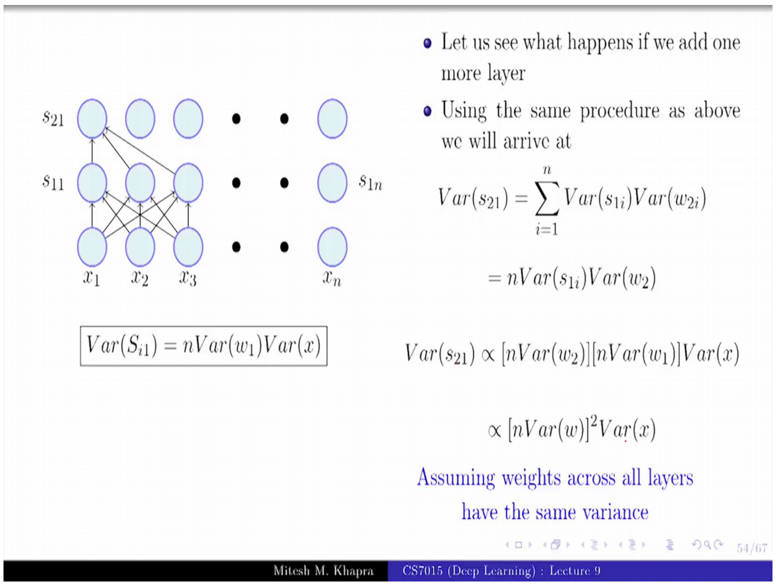
$$Var(S_{1i}) = (nVar(w))(Var(x))$$

- What would happen if $nVar(w) \gg 1$?
- The variance of S_{1i} will be large
- What would happen if $nVar(w) \to 0$?
- The variance of S_{1i} will be small

So, in general for any of these neurons right instead of just s 1 1, I could take any s 1 i and this is what the variance is going to be variance would turn out to be? Because? I have assumed that all the weights and all the inputs come out from the same variance distribution from a distribution having the same variance ok.

Now, let us what would happen if this quantity is very, very greater than 1? The variance of s 1 i would be very large right. And what would happen if this variance tends to 0 variance would be very low. So, I am just giving you 2 extremes to build the intuition and let us see what we are going to do with that intuition right.

(Refer Slide Time: 18:01)



$$\boxed{Var(S_{i1}) = nVar(w_1)Var(x)}$$

- Let us see what happens if we add one more layer
- Using the same procedure as above we will arrive at

$$Var(s_{21}) = \sum_{i=1}^{n} Var(s_{1i})Var(w_{2i})$$

$$= nVar(s_{1i})Var(w_2)$$

$$Var(s_{21}) \propto [nVar(w_2)][nVar(w_1)]Var(x)$$

$$\propto [nVar(w)]^2 Var(x)$$

Assuming weights across all layers have the same variance

54/67

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

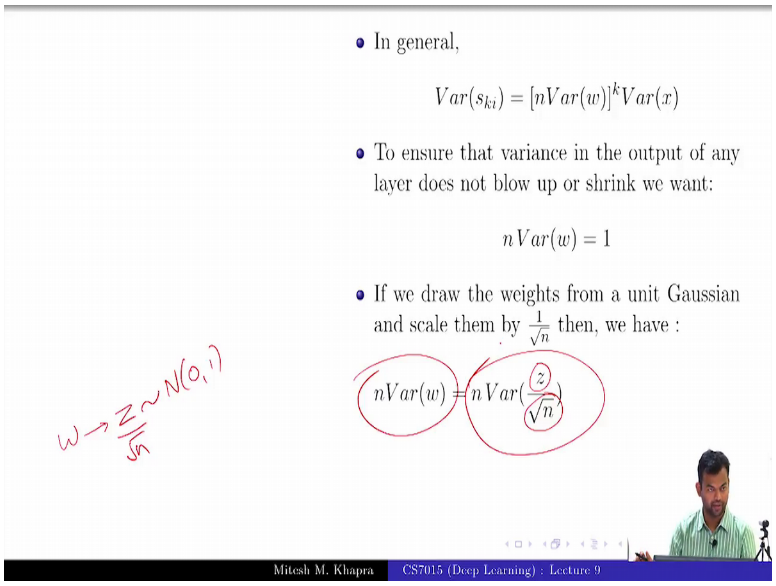
Now, let me add one more layer and see. So, I have added one more layer and using the same procedure as above. He will arrive at variance of s 2 1 is actually given by this formula and actually what has happened here is, that this is s i had x i earlier, but now instead of x ii have s 1 i because those are the inputs to this layer right.

So, this is exactly the formula that we had arrived at earlier. Assuming, 0 mean and the same variance for all the weights and the inputs and I am s arriving in the same formula for the next layer where, instead of x i have s 1 i everyone is fine with this ok. How many if you get this? Ok fine.

So, this will result in this quantity ok, but I already had a formula for a variance of s 1 i. What was that n into this quantity? So, I will just substituted there. So, I can say that variance of s 2 1 is actually equal to this. I just substituted this value.

So, that turns out to be, I have a square here when I have 2 here. So, you see where I am headed with this? What will be the variance of s k i. This raised to k and is on everyone gets this ok. I can just continue the same analysis and, I have assumed that these weights and always are the same variance right.

(Refer Slide Time: 19:29)



- In general,

$$Var(s_{ki}) = [nVar(w)]^k Var(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

$$nVar(w) = 1$$

- If we draw the weights from a unit Gaussian and scale them by $\frac{1}{\sqrt{n}}$ then, we have :

$$nVar(w) = nVar(\frac{z}{\sqrt{n}})$$

$$w \rightarrow \frac{z}{\sqrt{n}} \quad z \sim N(0,1)$$

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, in general I can say this ok. Now can you tell me something about when would this variance vanish when n variance of w is.

Student: Less than 1.

Less than one ok, and which is the thing that we should aim for you would want this quantity to be equal to one. In which case it will neither blow up nor shrink fine. So, so it to ensure that the variance is the output of any layer does not blow up or shrink. We should ensure that n into variance of w is equal to 1 right.. So, what is this, this list take a minute to understand this. I am saying that, I am going to initialize my weights.

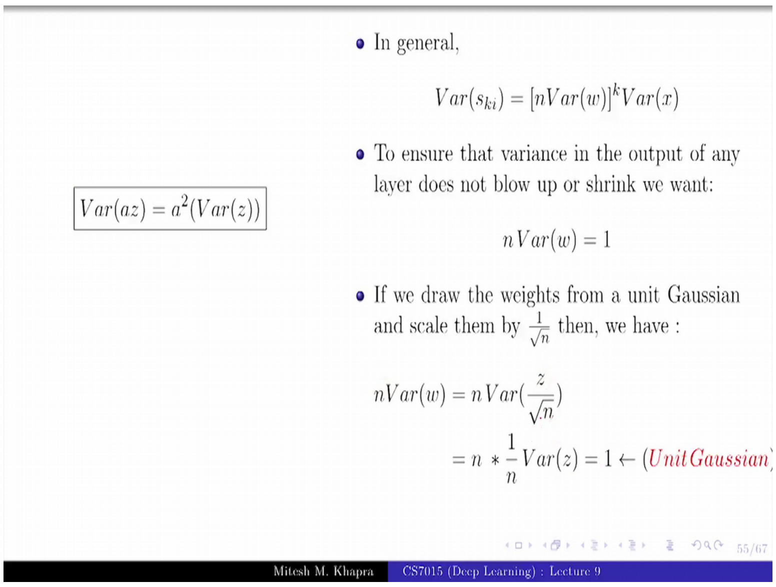
So, I should initialize them in such a way that, the weights are coming from some distribution like we saw that the distribution was a uniform distribution from where, I was drawing the weights.

So, they are coming from some distribution I should try to draw them from a distribution such that, this condition holds. If this condition holds then across layers, my activations will not blow up or shrink though that is exactly. What was happening in the earlier case when I was doing those bad initializations with small values and large values. All if you, get the idea ok. How many if you get this whatever I say said? Ok good.

So, let us see how to do that. So, what I am going to do is? I am going to consider a random variable z ok. Where is it z comes from a normal distribution ok and I am going to scale it is value, I will draw from there and then I am going to scale it by 1 by square root of n. What is n number of neurons in each layer right? Here it is the same across all layers, but it could also be different. So, I am considering a particular layer and n is the number of neurons in that layer.

And now if w is actually equal to z by square root of n then, I can write that n into variance of w is actually equal to this quantity. Everyone is fine with this? There is no trickery here, I am just saying that, why I am doing this is not clear that will become clear, but at least what I am doing is clear? I am drawing the weights I am taking a random variable z which comes from a normal distribution and then, I am setting my weights to whatever values I have drawn. I just divide them by the square root of n.

(Refer Slide Time: 21:49)



$\boxed{Var(az) = a^2(Var(z))}$

- In general,

$$Var(s_{ki}) = [nVar(w)]^k Var(x)$$

- To ensure that variance in the output of any layer does not blow up or shrink we want:

$$nVar(w) = 1$$

- If we draw the weights from a unit Gaussian and scale them by $\frac{1}{\sqrt{n}}$ then, we have :

$$nVar(w) = nVar(\frac{z}{\sqrt{n}})$$

$$= n * \frac{1}{n} Var(z) = 1 \leftarrow (UnitGaussian$$

55/67

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

Now, let us see, what is variance of a z a square into variance of z? Hey that is a basic formula all of us know this. So now, what is variance of z by one in z into 1 by square root of n 1 by n into variance of z right. So, the n and n cancel and what is variance of z. What did I assume about z it came from a normal distribution 0 mean and unit variance?

So, variance of z is 1; that means, this quantity n variance of w is going to be 1. If I have initialized my weights such that, they are equal to this right and now, do you see whether the weights are very small very large or what are they some now they made the weights dependent on the number of neurons.

So, if you have very large number of neurons, you are drawing drawing weights such that or you are initializing weights such that, it is some normal variable divided by the square root of n right. So now, when you do this summation w I x I your summation cannot blow up because, you have already divided it by n.

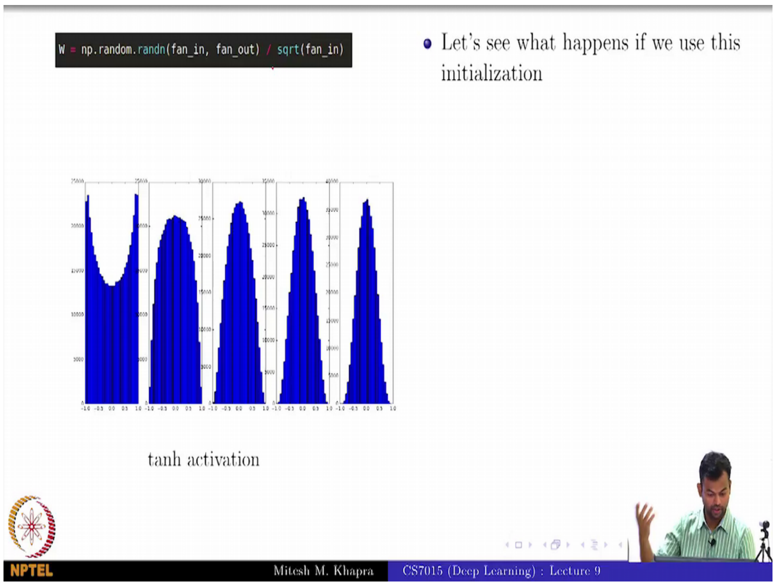
How many if you get this? So, this is a standard way used for initializing weights. How many if you tried this for your back propagation assignment? Why did you try this ah?

Student: (Refer Time: 23:12).

Because, you are having some problems with saturation I guess right. So, this is how you should initialize your weights. This is more or less the standard technique and some variance of this right. Because, instead of n you would have this fan in and fan in out it, how many weights are coming in? And how many weights are going out?

So, you make it proportional to the square root of n into k or something like that right. So, but in general this idea right of course, this proof we arrived at it with lot of assumptions, but we at least got to some principle way of initializing weights and this is a largely used standard this and some variants of it right.

(Refer Slide Time: 23:43)



```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in)
```

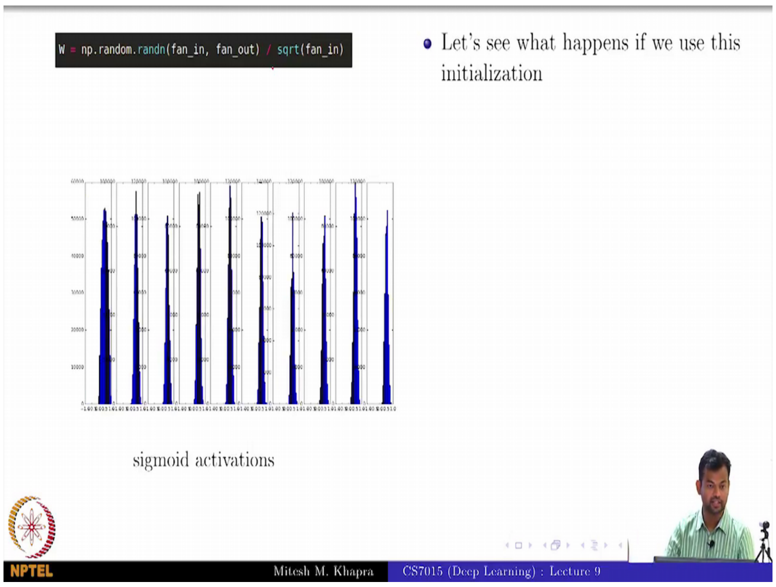
- Let's see what happens if we use this initialization

tanh activation

So now, let us see if I actually take the same network; that means 5 layers 500 neurons at every layer and then initialize it using this. So, this exactly what I had told you right that take it from a unit distribution sorry a normal distribution? And then, divided by the square root of the number of neurons in that layer and now let us see what happens across the 5 layers.

You see what happens we get this good variance in the activation functions. They are not all going to 0 or 1 or point 5 right. So, this solves the purpose for tan h activation and also for the sigmoid activations right.

(Refer Slide Time: 24:18)



```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in)
```

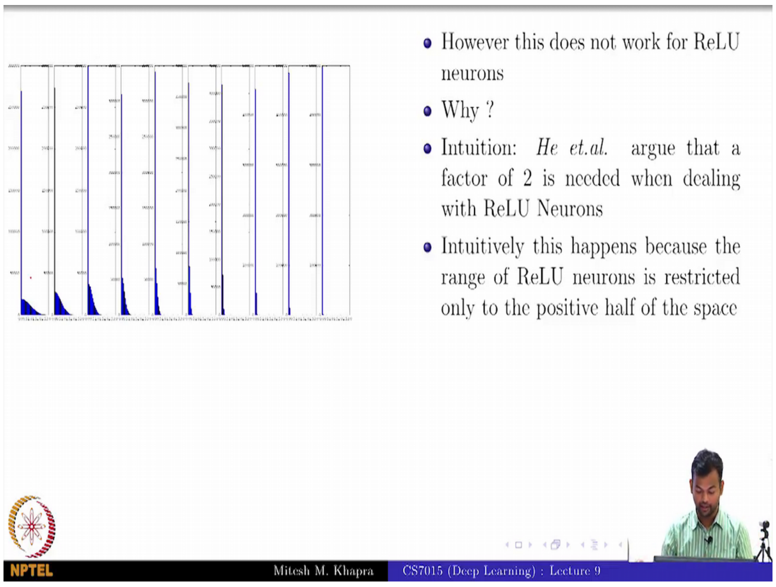
- Let's see what happens if we use this initialization

sigmoid activations

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

Do you see a good spread in the weights and remember actually for sigmoid? Although, these values look close to each other, but this is the 0 to 1 range. This is actually minus 1 to 0 which will not happen for sigmoid. So, within the 0 to 1 range you get a good spread if you initialize the weights this way.

(Refer Slide Time: 24:35)

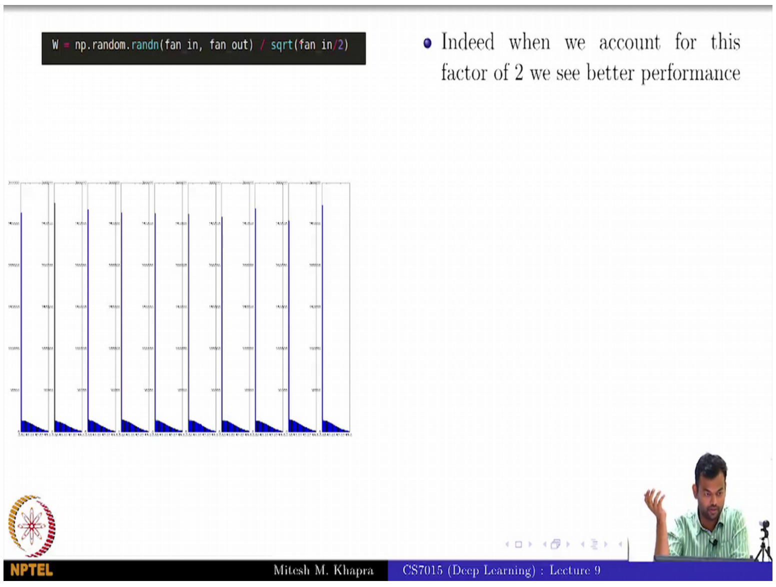


- However this does not work for ReLU neurons
- Why ?
- Intuition: *He et.al.* argue that a factor of 2 is needed when dealing with ReLU Neurons
- Intuitively this happens because the range of ReLU neurons is restricted only to the positive half of the space

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

But it turns out that, this initialization does not work for the ReLU function. In the ReLU function you still see this effect that, you started off with a good spread, but as you keep going across deps this spread disappears, why would that happen to someone gave an intuition for this and is again one of those heuristic e things. That in the case of ReLU you need to account for this divided by half. Because, half of the ReLU is not active right half of the ReLU is 0.

(Refer Slide Time: 25:10)



```
W = np.random.randn(fan_in, fan_out) / sqrt(fan_in/2)
```

- Indeed when we account for this factor of 2 we see better performance

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, you need to account for that fact and do this simple trick that, instead of taking the square root of the fan in you. Take the square root of fan in by 2 because, you know that half the times it is not going to produce any output right. So, that is a very simple heuristic that someone tried. And that leads to better activation functions better activations across all these layers right. So, as you see across all the layers the spread is good now so, the same idea ok. So now, you have a good way of initializing neurons.

So, this should help you in your future assignments fine. So, this is how, what you have learned about? How to initialize your weights? And it makes a lot of difference to, how your network will behave weight? And that is what the I was trying to show that by computing these activations across different layers? And I showed that as you change these initializations strategies you get better activations.