

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


A quick recap of training deep neural networks
Lecture – 09
Greedy Layerwise Pre-training, Better activation functions, Better weight initialization methods, Batch normalization

Welcome to lecture 9 of CS7015. Today we will talk about Greedy Layer wise Pre training, Better activation functions, Better weight initialization methods, and Batch normalization. So, today's lecture is more like tips and tricks to make deep learning work right.

So, when you are actually experimenting with deep learning in practice, what are some of the things that you need to take keep in mind. And it is also my way of connecting the history that we saw to where we are today right. So, there were certain things which we saw in the history, and now I will try to bring those back and connect to where we are headed from here right; where we have reached today and where we are headed from here.

So, that is with that in module 1, I will do a very quick recap of training neural networks and not take more than 5 minutes and I need it for a specific purpose.

(Refer Slide Time: 01:06)



• We already saw how to train this network

$$w = w - \eta \nabla w \text{ where,}$$
$$\nabla w = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w}$$
$$= (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x$$

• What about a wider network with more inputs:

$$w_1 = w_1 - \eta \nabla w_1$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_3 = w_3 - \eta \nabla w_3$$

where, $\nabla w_i = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x_i$

3/67

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

So, we already saw how to train such a very shallow neural network, what was the learning algorithm gradient descent and this was the update rule right. In particular, I wanted you to notice that the gradient actually depends on the input ok.

So, when you compute the gradient formula you have this multiplication by x. So, it is proportional to the input and this is one fact that we will use it; at least a couple of phases in the lecture today. So, this was a very shallow single neuron network, what if we have a wider network; Still which algorithm?

Student: Gradient (Refer Time: 01:40).

Gradient descent ok and we just have these 3 different formulae. And for each of these formulae note that the gradient or rather this gradient depends on the input that you are feeding in ok, I did not keep this in mind.

(Refer Slide Time: 01:52)

• What if we have a deeper network ?

• We can now calculate ∇w_1 using chain rule:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_1} = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} \cdot \frac{\partial y}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$= \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} * \dots * h_0$$

• In general,

$$\nabla w_i = \frac{\partial \mathcal{L}(\mathbf{w})}{\partial y} * \dots * h_{i-1}$$

• Notice that ∇w_i is proportional to the corresponding input h_{i-1} (we will use this fact later)

$a_i = w_i h_{i-1}; h_i = \sigma(a_i)$
 $a_1 = w_1 * x = w_1 * h_0$

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

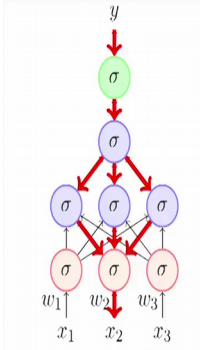
And what if you have a deeper network? So, we saw a very shallow network, we saw a wide network and I am showing you a deep network what will you do? Again gradient descent.

But you will apply the chain rule for computing the gradients. And again here in general you will notice that for any of these weights w 1, w 2, w 3 the gradient formula will have this h i minus 1, what is h i minus 1?

Student: (Refer Time: 02:17).

Input from the previous layer right, and h_0 is the actual input. So, the gradient at any layer is actually proportional to the input from the previous layer and this could either be the input from the hidden layer or the actual input.

(Refer Slide Time: 02:34)

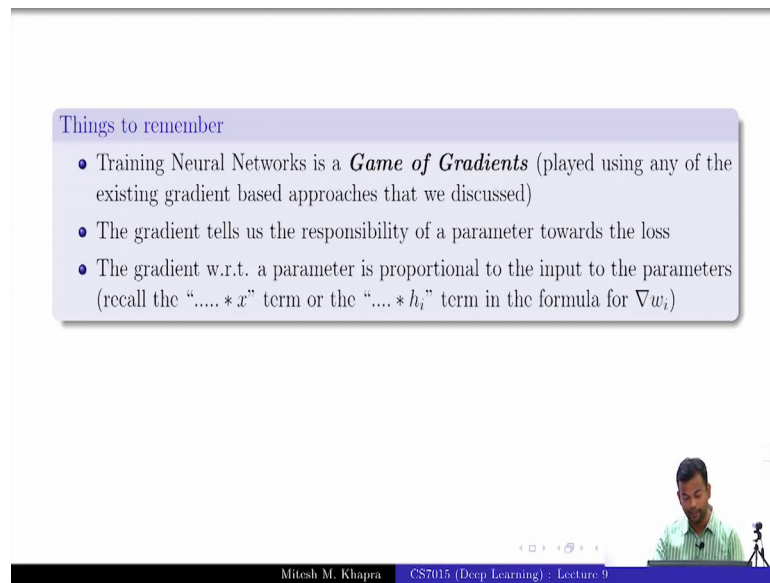


- What happens if we have a network which is deep and wide?
- How do you calculate $\nabla w_2 = ?$
- It will be given by chain rule applied across multiple paths (We saw this in detail when we studied back propagation)

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

And finally, we saw this thin. So, we saw a wide network, we saw a thin network. Now we will see a wide network and a deep network right, sorry we saw earlier we saw a wide network and a deep network; now we see a wide and deep network. And here again you have compute the gradient by applying this chain rule across multiple paths and that is what we use and we call it back propagation. And remember again they are the same thing holds that the gradients at some point are proportional to the input at that clear everyone remembers that ok?

(Refer Slide Time: 03:07)



Things to remember

- Training Neural Networks is a *Game of Gradients* (played using any of the existing gradient based approaches that we discussed)
- The gradient tells us the responsibility of a parameter towards the loss
- The gradient w.r.t. a parameter is proportional to the input to the parameters (recall the “... * x ” term or the “... * h_i ” term in the formula for ∇w_i)

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

So, this is important. So, what we have is things to remember from, what we have seen so far is that; so, training neural networks is basically a game of gradients right. So, you compute the gradients and everything depends on those how will you update the weights and everything from there on is about the gradients.

And these gradients actually tell you the responsibility of the parameters towards the loss, and you appropriately update them. And we saw a variant way different sorry various variants of how to use the gradient. So, we saw the gradient descent, we saw nag momentum and all.

But in all of these the underlying core thing was to compute the gradient and then do some manipulations based on that. And the other key thing is that, the gradient at a particular layer depends on the input to that layer fine.

(Refer Slide Time: 03:48)

Learning representations by back-propagating errors
David E. Rumelhart*, Geoffrey E. Hinton†
A Ronald J. Williams*

* Institute for Cognitive Science, C-181, University of California, San Diego, La Jolla, California 92037, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, USA

We describe a new learning procedure, back-propagation, for networks of interconnected units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the network's output and the desired output. As a result of the weight adjustments, internal hidden units which have not got direct inputs or target error signals represent distributed features of the task domain, and the capabilities of the task set, acquired by the interactions of these units. The ability to create useful new features distinguishes back-propagation from simple, single-module units as the preprocessor-convergence procedure*.

- Backpropagation was made popular by Rumelhart et.al in 1986
- However when used for really deep networks it was not very successful
- In fact, till 2006 it was very hard to train very deep networks
- Typically, even after a large number of epochs the training did not converge

NPTEL Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9 7/07

So now let us go back and just retrospect a better and see what is it that we have learned so far. So, so far what I have taught you gradient descent oh sorry, back propagation, is something which was proposed way back in 1986 right.

So in fact, it was existing before that, but it was popularized by this work of Rumelhart and others in 1986 right. So, but then in the 1990s or early 2000; if back propagation already existed and we could train deep neural networks, then why did not we here so much about deep learning at that time? Of course, you guys were busy with school and all at that time, but why did the others or older people like me not hear about it? Right.

Student: Computational power.

=Computational power is that the only thing?

Student: (Refer Time: 04:35).

Computation and memory is are the only thing?

Student: Convergence

Who said convergence? Ok good. So, actually what happened right in the late 80s and early 90s and even early 2000, when you used back propagation to train really deep networks, it was not very successful. And what do I mean by not successful? Actually, what are the 2 things that could happen someone gave the answer already.

Student: (Refer Time: 05:00).

It does not converge right; that means, you do not reach the optimum solution right. In fact, till 2006 it was very hard to train very deep networks ok.

And typically even after a large number of epochs these networks did not converge; that means, they were still at a very high loss and although in principle everything is fine, you have a deep neural network, you have an algorithm that can train it, but you are still not being able to train it properly and you are not being able to make any practical use of that right.

So, that was the story till 2006. So, today is about what happened in 2006? What it led to in the next few years and then where we are currently right. So, that is the journey that we need to make ok. And that is why, we started off with this quick recap of back propagation because that is what I want to tell you that why did it not work earlier and where are we today.