27/03/2020

Adding var momentum to weight update

We have seen that each network weight w_{ji} is updated using the following

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

where δ_j varies in it's formula, based on whether it is the weight update of an output unit or a hidden unit.

To recapitulate

For output unit k, $\delta_k \leftarrow O_k(1-O_k)(t_k - O_k)$

For hidden unit h, $\delta_h \leftarrow O_h(1-O_h)\sum_{k \in \text{outputs}} w_{kh}\delta_k$ (I)

In essence, the weights are updated in each iteration ~~but~~ by an amount Δw_{ji}

η is the learning rate, whose role we had already seen earlier.

It has been seen that in regions where the gradient does not change, the weight update can be improved by addition another term, α, $(0 \leq \alpha \leq 1)$ which is a constant value – called momentum.

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)$$

where n is the current iteration and $n-1$ is

the previous iteration.

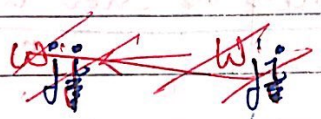
So, by changing the weight based on not only δ_j, but depending upon the previous weight update in the previous iteration, the algorithm avoids getting stuck in a local minima.

The effect of the momentum term (α) is to perturb or catalyse movement in regions where the weight update would have otherwise become stagnant.

What happens when there are many hidden layers? How do we update weights in such networks?

The Backpropagation algorithm generalizes to networks of arbitrary depth.

The weight update rule



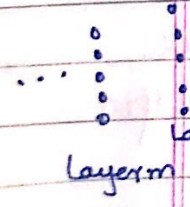
$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where $\Delta w_{ji} = \eta \delta_j x_{ji}$

only changes in the way δ_j is computed in each layer for deep networks.

Generally, for a hidden layer m

for a unit r



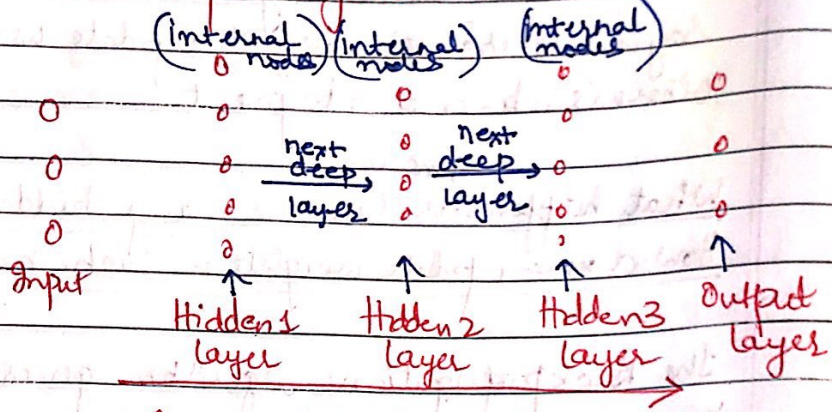
$$\delta_r = O_r(1-O_r) \sum_{s \in \text{layer } m+1} w_{sr} \delta_s \quad \text{(II)}$$

Notice its similarity to (I)

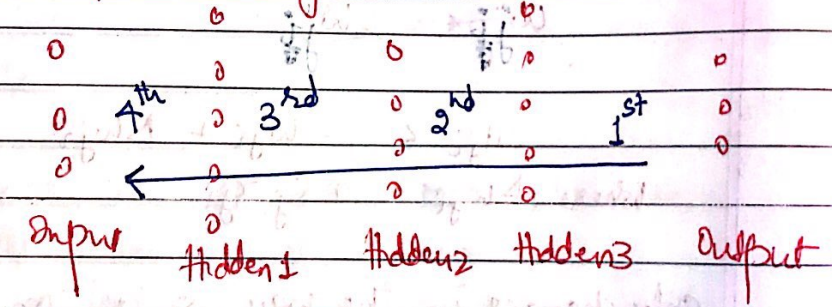
This implies that apart from the output layer, whose units' error term is calculated as

$$\delta_k \leftarrow O_k(1-O_k)(t_k - O_k)$$

All the other layers (hidden layers) follow a cascading procedure of calculating the error term based on equation (II) where each layer's term is dependent upon the next deeper layer's δ.



Sequence of calculating the error terms



It was also mentioned earlier, that it is not necessary that nodes of one layer will be connected to each node of the next layer.

So, for any internal unit (units that are not output units)

$$\delta_i = O_i(1-O_i)\sum_{s \in Downstream(i)} w_{si}\delta_s$$

I hope you remember what a Downstream set is from the last set of notes.

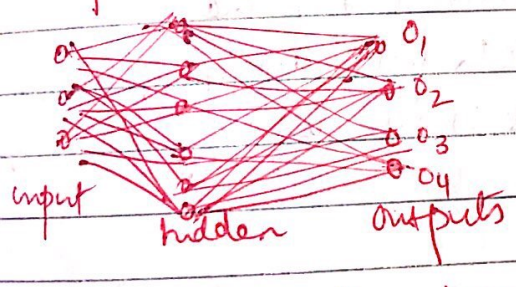
Does the Backpropagation algorithm learn the global optima or gets stuck in the local minima?

The BP algorithm employs gradient descent to minimize the squared error between the network output values and the target values for these outputs.

Also it has been mentioned in the last week's notes that since we are considering networks with multiple output units rather than single units, the error E can be re-defined to sum the errors over all of the network units for all the training examples

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

outputs → set of output units in the network



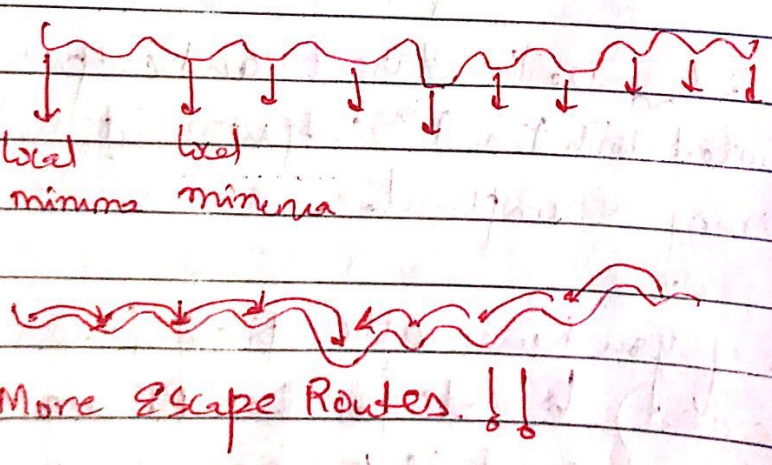
t_{kd} & o_{kd} are the target and output values associated with the k^{th} output unit and training example d.

Those of you have attempted (even if not succeeded) to code for the BP algorithm would understand this terminology easily.

Now the error surface can have multiple local minima in contrast to the single minimum parabolic error surface. Though this may imply that gradient descent will lead to convergence to local minima, in practise BP has been found to produce global optima for many applications.

Why is this so? Why is BP not afflicted with getting stuck in some local minima? There are some theories suggested for these.

Theory 1 — Network with large number of weights correspond to error surfaces in very high dimension. Each weight contributing to each dimension. It is argued that the more weights in the network, more are the escape routes for the gradient descent to fall away from the local minima w.r.t a single weight.



Theory B. Consider the following —

time →

(iteration) 1, 2, 3, . . . last

(weights) near zero, near zero, slightly more, larger

(Error function) Approx linear, approx linear, nonlinear network function

As the number of iterations increase, the weights that were initially very small (near zero) were contributing very very less to the network and hence the network function (in terms of error function) was approximately linear in its inputs.

$o = \frac{1}{1+e^{-\sum w_i x_i}}$

Only after some time has elapsed (iterations have elapsed), weights have had the time to grow (in dimension) [PLEASE CHECK YOUR VARIABLES IN THE BACKPROPAGATION CODE for EACH WEIGHT MATRIX!].

It is then that the weights contribute to represent highly nonlinear network functions.

It is now that many local minima exist and it is now that the problem of having struck in the a local minima may be anticipated.

But in time, another vital thing has happened, error has decreased over time. And mostly the weights have reached global optima.

So, though complex error surfaces do make us fear local minima, the theories above and many more, try to reason out how BP gives results that are acceptable for so many real world problems.

Common Methods used to prevent the Local Minima problem

The following are the common attempts made by us to skirt the issues of local minima

1. Momentum — Adding the momentum term to the weight update rule, helps avoid the local minima. (Details in the previous pages)

2. Use Stochastic gradient descent.
The stochastic method descends a different error surface for each training example. Since we shift from a training example to the next, the different error surfaces typically have different local minima, making the problem of getting stuck in the local minima less likely.

3. Say DU CS Department is holding a quiz on ML and I have to send a team of 4 students. I have taught (TRAINED) you all equally, I make 20 ~~de~~ groups of 4 students each with you 14 students.
I hope you understand this. I can have a student in more than one group. Say
{1,2,4,7} {2,5,7,9} {1,8,11,13} {4,5,7,3}
.....
Got it? Now I make different mock tests or even the same and evaluate the 20 groups. And I send my best performing group to DUCSD.

In the same vein, in NN, we can train multiple networks with the same data, initializing each network with random weights.
What does this mean?

Multiple networks

Say a 400 x 10X10X10 network
Say I have 1000 images of 20x20 pixels having hand-written digits.
So Input nodes are 400 (Fixed)
Output nodes are 10 (Fixed - for each digit)
The other are change-able
Say we have 2 hidden layers, each with 10 hidden Nodes
[How do I decide as to how many hidden layers I should have & how many nodes in each - is a question I will take up later.]
So I repeat, I have a
400X10X10X10 network
and I will have a set of 3 weight matrices.
I will randomly initialize them (like you did in BP) and then as the iterations progress, you search the optimum weight vectors for this network. (*)

Now there are 2 things to this.
One → I take the same architecture (400X10X10X0) and start with a random set of weight matrices. It will search it's optimum value. If I compare this weight matrix with the one achieved in *, they will be different and so ~~the~~ will be their results (most often)

Two → I take a different architecture,
Say 100 × 15 × 10 × 10
or 100 × 20 × 10
or 100 × 7 × 11 × 10 etc
and get a final weight matrix.

So, whatever I do, my training samples remain the same (1000 images).
I test all of them with some 200 new images or different set of new images (say) and choose the one with the least error.

By this technique, we can get the architecture and the set of weights which will invariably perform best on such problems even with unseen data.

I had given you the FF question with the input data, the architecture and the optimum weights! Think.

Rather than sending my best performing team, I could have done it differently. (Though it does not apply for my students example).
Retain all the networks and form an ensemble or committee whose output is the possibly weighted average of the individual network outputs.

Prabhu Ram
28/05/2020