October 19, 2017 Lecture 10 , Oct. 10, 2017

Neural Networks

Lecturer: Justin Domke *Scribe: Lakshmi Vikraman*

1 Summary

Last lecture we talked about classsification models as well as the loss functions used in these models. As we have seen the main difference between classification and regression models are the loss functions. (eg: in case of decision trees , gini entropy for classification vs squared error for regression). Today we are going to talk about neural network models and automatic differentiation techniques.

2 Neural Networks

2.1 Motivation

In case of applications where there is a lot of data , what we are trying to predict might be very complex. Some examples of this are applications such as Machine Translation, Speech Recognition and Image Classification. In such a situation what we need are models which have high capacity as well as high scalability or efficiency. Before we go into more details on how neural network models solve this, let us understand a fundamental algorithm in Machine Learning.

2.2 Gradient Descent Algorithm

The goal of the algorithm is minimize $R(\theta)$ where

$$R(\theta) = \sum_{i=1}^{N} L(Y_i, f(X_i))$$

where f is the predictor , θ parameterise function f and L denotes loss .

Algorithm 1 Gradient Descent Algorithm

1: **repeat**
2: Calculate $\frac{dR}{d\theta}$
3: $\theta \leftarrow \theta - \gamma \frac{dR}{d\theta}$
4: **until** convergence

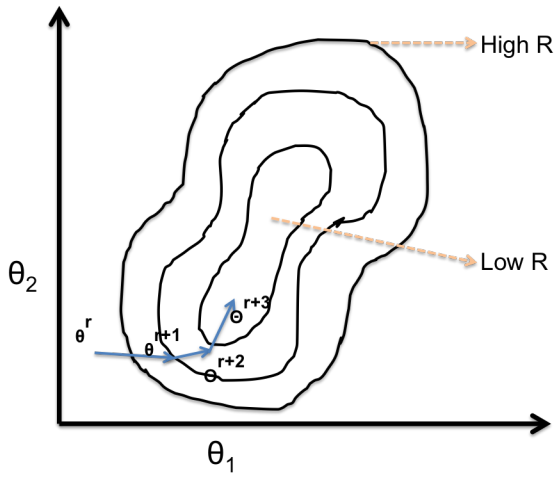

Figure 2.1: Gradient Descent

The figure 2.1 shows how gradient descent works in a 2D space. γ is the step size. Smaller values of γ is more likely to converge but it would be slow.

How do we calculate $\frac{dR}{d\theta}$. We can calculate it for a single point $\frac{dL(Y,f(X)}{d\theta}$ and then sum for all datapoints. It is expensive to perform this using Calculus , hence we are going to use a different algorithm called "Automatic Differentiation" (also called as "Autodiff" or "Backprop"). This is not the same as symbolic differentiation.

2.3 Automatic Differentiation

Given a differentiable function $f(X) : R^n \rightarrow R$, this algorithm automatically computes $\nabla f(X) : R^n \rightarrow R^n$. Some properties of this algorithm are :

- These algorithms are very general.
- ∇f computation takes the same time(up to constants) as f itself.
- This is a key technology in deep learning.
- Some examples where this is used are in packages such as TensorFlow, Theano etc.

Let us look at an example to explain this further. Let f be a function and x $\in R$.

$$f(x) = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$$

The derivative of the function f is given below.

$$\frac{df}{dx} = \exp(\exp(x) + \exp(x)^2)(\exp(x) + 2\exp(x)^2) + \cos(\exp(x) + \exp(x)^2)(\exp(x) + 2\exp(x)^2)$$

This blows up when there are complicated functions. Instead we are going to define intermediate variables as shown below.

$$a = \exp(x)$$
$$b = a^2$$
$$c = a + b$$
$$d = \exp(c)$$
$$e = \sin(c)$$
$$f = d + e$$

We will use expression graph to picture this.

.

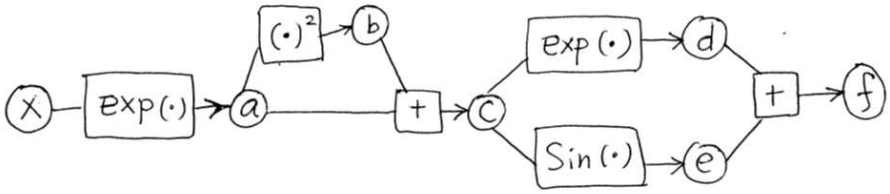

Figure 2.2: Example Network

These graphs are easy to understand and evaluate. They also unravel the internal structure of the functions.
In order to calculate the derivatives we use chain rule.

$$\frac{df}{dd} = 1$$
$$\frac{df}{de} = 1$$
$$\frac{df}{dc} = \frac{df}{dd}\frac{dd}{dc} + \frac{df}{de}\frac{de}{dc} = \frac{df}{dd}\exp(c) + \frac{df}{de}\cos(c)$$
$$\frac{df}{db} = \frac{df}{dc}\frac{dc}{db} = \frac{df}{dc}$$
$$\frac{df}{da} = \frac{df}{dc}\frac{dc}{da} + \frac{df}{db}\frac{db}{da} = \frac{df}{dc} + \frac{df}{db}2a$$
$$\frac{df}{dx} = \frac{df}{da}\frac{da}{dx} = \frac{df}{da}\exp(x)$$

We can work back from the end of the graph and compute the derivatives using chain rule.

Formally we can define this as follows :
Input : $x \in R^n$
Total number of nodes : $N > n$

Functions for each node : $g_i, n < i \leq N$
Parents of each node : $Pa(i) \geq 0$

In the given example we have
$n = 1$
$N = 7$
$g_1 = none$
$g_2 = exp$
$g_3 = ()^2$
$g_4 = +$
$g_5 = exp$
$g_6 = sin$
$g_7 = +$
$Pa(1) = none$
$Pa(2) = \{1\}$
$Pa(3) = \{2\}$
$Pa(4) = \{2, 3\}$
$Pa(5) = \{4\}$
$Pa(6) = \{4\}$
$Pa(7) = \{5, 6\}$
where the numbers refer to the following variables : $1 \rightarrow X$, $2 \rightarrow a$, $3 \rightarrow b$, $4 \rightarrow c$, $5 \rightarrow d$, $6 \rightarrow e$, $7 \rightarrow f$ in the expression graph.

The Forward Propagation algorithm is given below.

Algorithm 2 Forward Propagation

1: **for** i = $n + 1$,$n + 2$,....N **do**
2: $x_i \leftarrow g_i(x_{Pa(i)})$
return x_N

There are two versions for the Back Propagation algorithm. The two versions are given below. The first version has to be executed after running the Forward Propagation algorithm.

Algorithm 3 Back Propagation version 1

1: **Initialize:**
$\frac{df}{dx_N} \leftarrow 1$
2: **for** i = $N - 1$,$N - 2$,....1 **do**
3: $\frac{df}{dx_i} \leftarrow \sum_{j:i \in Pa(j)} \frac{df}{dx_j} \frac{dg_j}{dx_i}$
return $\frac{df}{dx_1}$,....., $\frac{df}{dx_n}$

Algorithm 4 Back Propagation version 2

Initialize:
$\frac{df}{dx_N} = 1$, $\frac{df}{dx_1} = \frac{df}{dx_2} = ...\frac{df}{dx_{N-1}} = 0$
for j = N,$N-1$,....$n+1$ **do**
  for all $i \in Pa(j)$ **do**
    $\frac{df}{dx_i} \leftarrow \frac{df}{dx_i} + \frac{df}{dx_j}\frac{dg_j}{dx_i}$
return $\frac{df}{dx_1}$,....., $\frac{df}{dx_n}$

In the second version , Node i looks at all its children and pulls the derivatives towards itself. In the first version Node j looks at all its parents and pushes the derivatives towards the parents. They do almost the same thing, but the difference is in the push/pull. The first version is easier to understand but is often not done in practice. The second version is more widely used, the reason being(in the first version) it is inconvenient to find all nodes which have a particular node as parent.