Optimising Function Approximation using MLP Neural Network

Zarana Parekh (201301177), Henil Shah (201301436)

November 15, 2016

1 Problem Statement and Motivation

The most powerful feature of artificial neural networks (ANNs) is their universality. There is a type of neural network (NN) for computing any kind of function. This is true of ANNs even if we restrict the network to contain only a single hidden (intermediate) layer. Here we exploit this property of ANNs to determine the underlying relationship between an independent variable and one or more dependent variables from a finite input-output data set. We use the simplest framework of a multi-layer perceptron NN with only one hidden layer for function approximation.

ANNs are important for processing large volumes of complex information and deriving relation between them. In order to make the algorithm efficient, we use the gradient descent method for minimising the error function of the neural network. This will improve the rate of convergence and decrease its computationally complexity.

The gradient descent method is memory intensive because it performs computations on every data point for each iteration on the dataset and requires a large number of iterations to converge which makes it slow. To address this issue, we compare the performance of the traditional batch gradient descent method as described above with the stochastic gradient descent method which only uses one data point for computation in each iteration.

2 Background

2.1 Artificial Neural Networks

An ANN is a novel structure for storing exponential information and processing it. They have become increasingly popular because of their ability to extract patterns and detect trends from complex data for solving real-world problems whose algorithmic solutions are too complex to be found. Their applications include pattern recognition, data classification and forecasting.

In functional approximation, NN use layers of artificial neuron models for predictive modeling of complex tasks. Training data is used for learning the relation between the output variable (to be predicted) and the input variables. This generates a mapping used for future predictions. Further, features generated at each layer can be combined to form higher-order features that are more complex.

2.2 Artificial Neurons

These are the simplest computation units of any ANN. Given weighted input signals they produce an output signal using an activation function. Each neuron has a **bias**, an input of value 1 and must also be weighted.

The mapping of summed weighted input to the output of the neuron is represented through the **activation function**. Commonly used activation functions include linear, sigmoid (logistic) and hyperbolic tangent functions. Activation functions govern the threshold at which the neuron is activated and strength of the output signal. Non-linear activation functions are used to increase the capability of the NN to model complex functions.

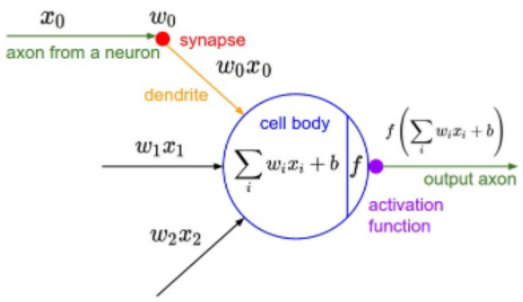
2.3 Multi Layer Perceptron

A multi layer perceptron NN (MLP) is the model of a NN with one or more layers between the input and output layers. It is a feedforward ANN which means that data only moves forward from the input to the output. MLPs are used to solve problems that are not linearly separable.

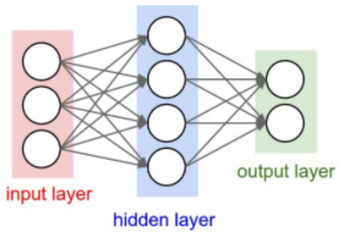
There are three layers in the network:

1. The **input or visible layer** which takes input from the dataset. Each value is taken in a neuron which passes it forward to the next layer of neurons.
2. The **hidden layer** which is not exposed to the input directly. These increase the computing power of the NN.
3. The **output layer** which outputs the estimated vector of values in the corresponding format required for the problem.

Each layer may contain one or more neurons depending on the nature of the problem.



(a) Mathematical model of a neuron



(b) A simple MLP NN with one hidden layer

3 Approach

3.1 Gradient Descent Method

Suppose we want to approximate a function

$$y = f(x)$$

The training dataset is supplied as input to the learning algorithm implemented using MLP. After training, a set of parameter values is assigned to each of the neurons and the approximating function thus generated is called the **hypothesis function**. It is denoted as $h_\theta(x)$. For the following explanation, we take an example hypothesis function $h_\theta(x) = \theta_0 + \theta_1 x$.

The error function is represented as:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

Here, m is the number of training data points. We have considered a mean squared error function here.

The idea is to minimize the average error function so that we obtain the best possible function approximation. For this, we choose θ_0, θ_1 so that the predicted function value is close to our training data point values.

In our neral netowrk simjld

That is,

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1) = \min_{\theta_0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2}$$

where $\frac{1}{2m} \sum\limits_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$ is the **cost function.**

For each parameter value we apply the gradient descent formula until the solution converges:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (j = 0, 1) \tag{3}$$

Here, α represents the learning rate. If its value is too small, the convergence rate can be slow and if it is too large, it can overshoot the minimum and the method may fail to converge or diverge.

The gradient descent method can converge to the minimum even if the learning rate is fixed. It will automatically take smaller steps as we approach the minimum.

We calculate $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ as follows:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{4}$$

3.2 Stochastic Gradient Method

The above method is called the **batch gradient descent** because each step of gradient descent uses all the training points. However, for a large dataset, this is an expensive procedure. It is also difficult to introduce new data in an 'online' setting. Hence, for scaling to bigger datasets the **stochastic gradient descent method** is used. It is a stochastic approximation of the gradient descent method which is used to overcome the high cost of running back propagation over the full training set and achieve convergence faster.

For stochastic gradient descent we modify our definitions of the hypothesis and cost functions as follows:

$$h_\theta(x) = \sum_{j=1}^{n} \theta_j x_j \tag{5}$$

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2 \tag{6}$$

$$J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)})) \tag{7}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j(i) \qquad (j = 0, 1) \tag{8}$$

where $\frac{\partial}{\partial \theta_j} cost(\theta, (x^{(i)}, y^{(i)})) = (h_\theta(x^{(i)}) - y^{(i)}) x_j(i)$. To improve the accuracy of our future predictions, we randomly shuffle the dataset before applying the stochastic gradient method.

3.3 Implementing back propagation

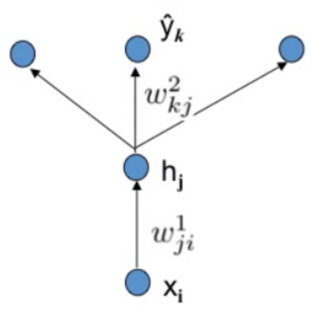


Figure 2: Back propagation in MLP

Here i is the input layer, j is hidden and k is the output layer. w_{mn} represent the weight vector between layer m and n. x_i is the input vector which makes $w_{ij}x$ input for the hidden layer. Here h_j is the output from hidden layer and y'_k is the predicted final output.

We measure the final error by comparing y'_k with the actual targeted output y_k.

Loss Function:

$$J_i(w) = \sum_k (y^{(i)} - \hat{y}^{(i)})^2 \tag{9}$$

Output layer:

$$\hat{y}^{(i)} = \sigma(s_k) = \sigma(\sum_j w^2_{kj} h_j) \tag{10}$$

Hidden layer:

$$h_j = \sigma(t_j) = \sigma(\sum_j w^1_{ji} x_i) \tag{11}$$

$$\frac{\partial J}{\partial w^2_{kj}} = -2(y^{(k)} - \hat{y}^{(k)})\sigma'(s_k)h_j \tag{12}$$

$$\frac{\partial J}{\partial w^1_{ji}} = \sum_k -2(y^{(k)} - \hat{y}^{(k)})\sigma'(s_k)w_{kj}\sigma'(t_j)x_i \tag{13}$$

$\beta^2_k = -2(y^{(k)} - \hat{y}^{(k)})\sigma'(s_k)$ propagates from one layer to another.

4 Results and Conclusion

To compare accuracy of both the methods, we tried approximating two time series using a multi layer perceptron network using same number of epochs. Both the time series had 3 input factors and one output.

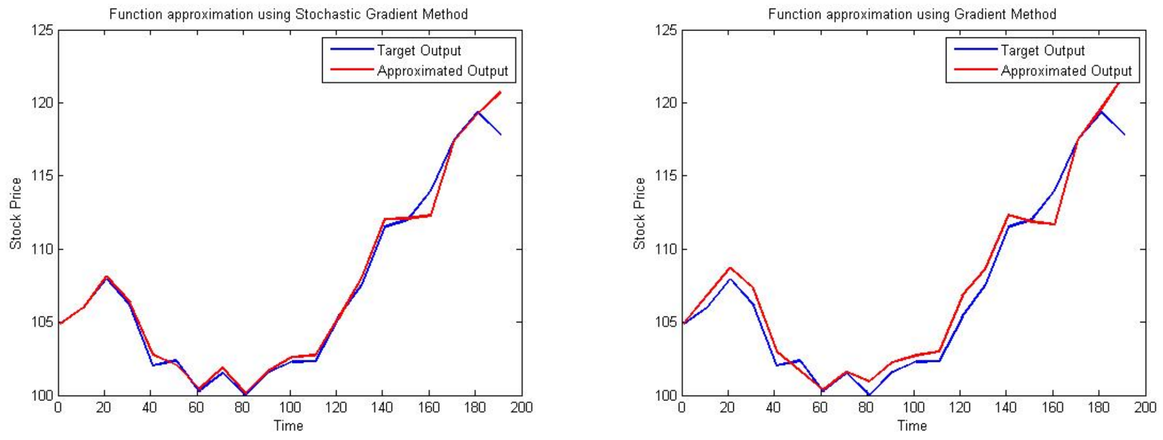
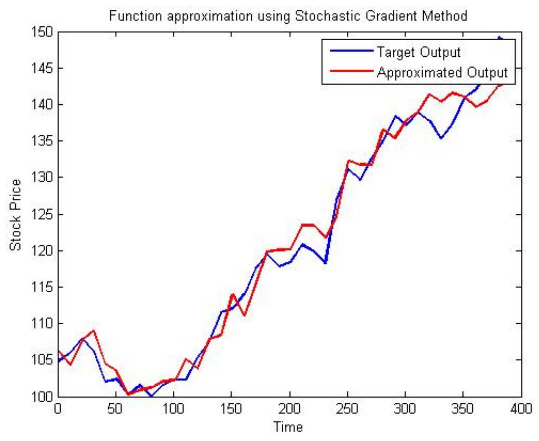


Figure 3: Time series approximation



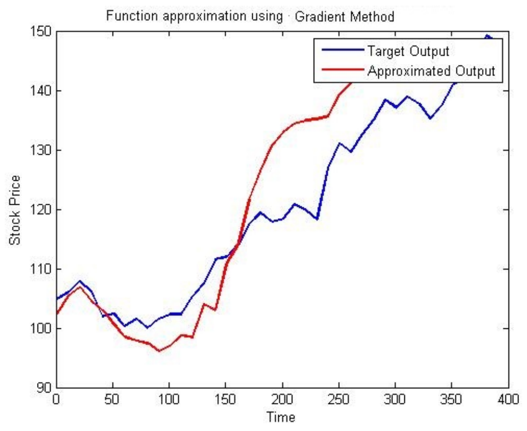


Figure 4: Complex Time series approximation

For normal time series functions, both methods performed well. Performance of stochastic gradient descent method was better than gradient descent, but the latter also performed reasonably well (as shown in figure 3). With stochastic gradient descent method, we could get the accuracy of gradient descent with one less neuron. But the maximum learning rate for which the method coverged was higher for gradient descent method compared to stochastic method. The number of epoch (iterations over dataset) was 60,000 for this experiment. The learning rate was 0.12 and number of hidden neurons was 7.

Method	**Mean Squared Error (Normalized)**
Gradient Descent Method	0.096
Stochastic Gradient Descent Method	0.077

Table 1: Details of normal time series prediction

Approximation of a complex time series requires a more sophisticated neural network than MLP. But we still tried to approximate one to observe behaviour of both the methods. As shown in figure 4, both the methods gave poor results, but stochastic gradient descent methods was still able to follow the trend of the time series. While results gained through gradient descent method was quite erratic.

Here the number of epoch was 100,000, learning rate 0.02 and number of hidden neurons was 4.

Method	**Mean Squared Error (Normalized)**
Gradient Descent Method	0.36
Stochastic Gradient Descent Method	0.22

Table 2: Details of complex time series prediction

5 Applications

The ability of NN to identify trends in data using function approximation and make appropriate assumptions has several applications including the following:

- sales forecasting	- industrial process control	- target marketing
- customer research	- data validation	- risk management