Backpropagation Through Time Algorithm for Multi-Class Classification using Recurrent Neural Networks

Mahapara Ashraf¹, Umar Yaqoob Khan²
Department of Computer Science
University of Kashmir, Hazratbal, Srinagar, J&K, India

Abstract:
During recent past Deep Learning bought evolutionary changes with it which helped in achieving great results to model high level abstractions in data through the use of model architectures. In this paper we are working on deep learning model which is used to extract the better features from IRIS flower which is a multivariate dataset having 150 observations to solve the multiclassification problem. A basic recurrent neural network (RNN) is used to enhance the work for multiclass classification where “back propagation through time algorithm” is used to perform analysis and learning by taking various parameters into account. It is considered as a building block to work in the pattern recognition and classification environment. It contains 3 different classes of 50 objects each, where each class is an iris flower which have names as Iris (Setosa, Versicolour, Virginica). The database needs no preprocessing except that the multiclass problem of Iris dataset used has categorical data and to work with categorical data one-hot vector encoding is used to identify the class uniquely.

Keywords: Backpropagation Through Time, Classification, Deep Learning, Iris Flower, one-hot vector, Recurrent Neural Network.

I. INTRODUCTION
Most of the deep learning algorithms are based on artificial neural networks using deep layered structures. However, the simple learning algorithm which were successful for training shallow layered structures failed to train deep layered structure. Deep learning is broader family of machine learning methods which supports learning data representations, as a critical task-specific algorithms. It's a group of ML techniques that uses stacked layers of transformation trainable from the start to the end. Each layer is represented as a series of neurons to keep extracting higher and better level features until final layer essentially makes a call about what the input shows. The more layers the network has the better features it will learn. Deep learning architectures are applied to fields like image processing, natural language analysis, big data analysis, internet search, computer vision, pattern classification speech recognition and drug design. Many applications using deep learning are already developed and in some cases produce results superior than human experts. Deeply Layered networks were developed as a Machine Learning approach to house complex input-output mappings. From the study of pattern recognition traditional ML methods successfully solve problems and explore the construction of algorithms through building a model where final value is a simple function of input data. In Deep learning models the typical models include Autoencoder (AE), Convolutional Neural Network(CNN), Deep Belief Networks and Recurrent Neural network(RNN). The main purpose of this paper is to demonstrate a novel method of developing Recurrent Neural network based classifier which classifies the Iris Flower database and save the states.

II. RELATED WORK
A. Recurrent Neural Networks (RNNs)
Recurrent Neural Networks (RNNs) represent a class of artificial neural network which are specialized in forming connections between units thus forming directed cycles with each other, the internal states help them in processing dynamic temporal behavior of network. RNNs are specialized in using their internal memory to process arbitrary sequences of inputs. In basic neural networks, we assume all inputs and outputs perform feature extraction independently RNNs are called recurrent because for every element of a sequence, the output is being dependent on the previous computations where its nodes directionally connect with each other into a loop. This internal state can handle any input timing sequence. The internal memory captures information of what has been calculated,[1] RNN makes use of history, simply termed as ‘memory’ of previous inputs to persist in the network's internal state, this greatly influences the output of the network. RNN’s take current input along with what they have perceived previously in time. In every time step, the RNN receives an input, updates its hidden state, and makes a prediction based on the inputs provided.

Figure 1. A recurrent neural network with timesteps.

The Recurrent Neural Network (RNN) uses high dimensional hidden state and nonlinear prediction methodology to achieve more accuracy in predictions and in lesser steps unlike multilayer perceptron. An MLP (multilayer perceptron) uses only input provided (does not use history as inputs) and maps
the input to output vectors by adjusting the weights as per the output in the process of learning, whereas an RNN uses history of previous inputs to each output.[2] Difference between MLP and RNN In RNN we have self loops and feedback loops unlike MLP.

Figure.2. MLP and RNN model with same number of nodes

B. Backpropagation in RNN
Certain recurrent networks depend on a gradient-based technique to adjust the weights during training. The algorithms used for such cases are termed as backpropagation through time. Backpropagation through time is basically an extension of backpropagation which updates weights like LSTM during the process of learning. Time in backpropagation through time is defined by an ordered series of calculations to determine the derivatives and an application of the derivative chain rule. Adding a time element only extends the series of functions for derivatives calculated with the chain rule. Errors are then calculated and accumulated for each timestep. Background Backpropagation through time-like basic backpropagation-is used most often in pattern recognition today. In some applications-such as speech recognition or submarine detection-our classification at time t will be more accurate if we can account for what we saw at earlier times. Even though the training set still fits the same format as above, we want to use a more powerful class of networks to do the classification; we want the output of the network at time t to account for variables at earlier times.[5] The Introduction cited several examples where such “memory” of previous time periods is very important. For eg; it is easier to recognize moving objects if our network accounts for changes in the scene from the time t - 1 to time t, which requires memory of time t - 1. Many of the best pattern recognition algorithms involve a kind of “relaxation” approach where the representation of the world at time t is based on an adjustment of the representation at time t - 1; this requires memory of the internal network variables for time t - 1. Example of a Recurrent Network Backpropagation can be applied to any system with a well defined order of calculations, even if those calculations depend on past calculations within the network itself. For the sake of generality, every neuron is potentially allowed to have history by using input values from any of the neurons at the two previous time periods (including, of course, the input neurons). Process of unfolding over time can be used to translate a recurrent network into a feed-forward network:

III. METHODOLOGY

The connections of hidden layers in RNN is iteration cycle, which has weight connection between the current state and previous state. The network is based on recurrent neural network (RNN) model where RNN contains at least one feedback connection, so the activations can flow and form a loop thus enabling the networks to do timing processing and pattern learning, e.g., perform sequence recognition. In Recurrent architectures the timing information signals can be accurately expressed and have correlation of nodes. RNN with cycle of iterative function stores timing related information effectively. The connection of hidden layers in RNN is iteration cycle, which has weight connection between the current state and previous state, potentially every neuron connected with others, and may also have stochastic activation functions. For simple architectures and activation functions the power of learning the feature extraction and analysis of the temporal behavior can be achieved using stochastic gradient descent functions leading to the back-propagation algorithm in feed-forward networks for weight updation and error propagation. When the activations are stochastic, stimulated approaches are suggested for improvement.[6] The unique design of RNN makes information of recent input events stored by the feedback connection, which is called short term memory. RNN uses output at last time-step to calculate output at hidden layers. The output at each time-step defines the state at that time-step.[3]

C. Training RNN
To imagine how weights would be updated in case of a recurrent neural network, might be a bit of a challenge. To understand and visualize the back propagation, unroll the network across different time steps. Unlike a feed forward propagation, we are figuratively going back across different time steps to change and update the weights, hence it is called as back propagation through time (BPTT). The Recurrent (loop nodes) neuron in this case are just taking the immediate previous state into consideration and are converted into feedforward network by unrolling across time steps. For longer and complex sequences it can involve multiple such states. [4]

Figure.3. Recurrent network converted into a feed-forward network

D. BPTT Algorithm
1. Read the features of dataset as input train and the class as output train.
2. Determine number of nodes in each layer.
3. Initialize the weight matrices with random values at different layers, learning rate and biases.
4. Also initialize the matrices that will be used to update weights.
5. Using forward pass calculate the predicted output values and save them for backpropagation.
Forward pass:

For \( j = 1 \) to timesteps do

\[
\text{layer}_h(t) = \text{sigmoid}(\text{input} \ast w_{ih} + \text{layer}_h(t-1) \ast w_{hh} - b_h)
\]

\[
\text{layer}_k(t) = \text{sigmoid}(\text{layer}_h(t) \ast w_{hk} - b_k)
\]

Also calculate error and gradient at output layer:

\[
\text{layer}_k\_error = \text{output} \ast \text{layer}_k(t)
\]

\[
\text{layer}_k\_delta(t) = \text{layer}_k\_error \ast \text{diffsig}(\text{layer}_k(t))
\]

\[
\text{layer}_h\_copy = \text{layer}_h(t)
\]

6. Using backpropagation calculate the deltas for input and hidden layer

Backward pass:

\[
J = 1 \text{ to } \text{timesteps do}
\]

\[
\text{layer}_h\_error = \text{layer}_h\_delta(t+1) \ast w_{hh} + \text{layer}_k\_delta(t) \ast w_{ih}
\]

\[
\text{layer}_h\_delta = \text{layer}_h\_error \ast \text{diffsig}(\text{layer}_h)
\]

Accumulate the values in weight update matrices.

\[
\text{w} \_\text{update}_{ih} = \text{w} \_\text{update}_{ih} + \alpha \ast \text{input} \ast \text{layer}_h\_delta
\]

\[
\text{w} \_\text{update}_{hh} = \text{w} \_\text{update}_{hh} + \alpha \ast \text{layer}_h(t-1) \ast \text{layer}_h\_delta
\]

\[
\text{w} \_\text{update}_{hk} = \text{w} \_\text{update}_{hk} + \alpha \ast \text{layer}_h(t) \ast \text{layer}_k\_delta
\]

End for

7. Update the weights using this weight update matrices.

IV. PROBLEM DESCRIPTION AND DESIGN

- In this project we are using “iris flower dataset” an online available dataset and downloaded from machine learning repository. This dataset has features given as input and the network is trained to classify the attributes into three classes of iris flower data set.
- It has 150 instances, which are equally separated between the three classes, contain the following four numeric attributes:
  - Sepal length, Sepal width, Petal length, Petal width

V. IMPLEMENTATION

Our proposed model is divided into parts:

1. Splitting Dataset: Iris flower data set is being divided into training and testing part according to the requirement of our project so it can give us the best classification results produced.
2. Formatting Data set: Iris dataset (150 instances, 5 attributes)
3. Learning Algorithm: Backpropagation through time is being used where we have took 2 hidden layers with different time steps and unrolled the same.
4. Training Network: out of 150 instances 120 are taken for the training part having 40 instances from each class as training part.
5. Testing network: out of 150 instances 10 instances are taken for the testing part from each class thus making it overall 30 instances for testing.
Categorical data can be termed as the variables that take a limited and defined values and usually contain label values rather than numeric values. The number of possible values is often limited to a fixed set. Categorical variables are often called nominal. E.g: A “color” variable with the values: “red”, “green” and “blue” or a “blood type” variable with the values: “A”, “B” and “O”. The Problem with Categorical data is some algorithms can work with categorical data directly. For example, decision trees can be created (learned) directly from the categorical data instead of converting the data to another form. In general, all the machine learning algorithms can’t operate on categorical data directly rather the data needs to be changed to some other forms usually numerical forms. This means that categorical data must be converted to a numerical form.

F. Splitting dataset and classification

The Iris flower data set contains overall 150 instances and 5 attributes.

1. sepal_length: Sepal length, in centimeters, used as input.
2. sepal_width: Sepal width, in centimeters, used as input.
3. petal_length: Petal length, in centimeters, used as input.
4. petal_width: Petal width, in centimeters, used as input.
5. setosa: Iris setosa, true or false, used as target.
6. versicolour: Iris versicolour, true or false, used as target.
7. virginica: Iris virginica, true or false, used as target.

G. Categorical Data

Categorical data can be termed as the variables that take a limited and defined values and usually contain label values rather than numeric values. The number of possible values is often limited to a fixed set. Categorical variables are often called nominal. E.g: A “color” variable with the values: “red”, “green” and “blue” or a “blood type” variable with the values: “A”, “B” and “O”. The Problem with Categorical data is some algorithms can work with categorical data directly. For example, decision trees can be created (learned) directly from the categorical data instead of converting the data to another form. In general, all the machine learning algorithms can’t operate on categorical data directly rather the data needs to be changed to some other forms usually numerical forms. This means that categorical data must be converted to a numerical form, however as already discussed, this is not mandatory in all the cases. In cases where categorical variable is an output variable, we may also require converting data to categorical form to show the relevant output. To Convert Categorical data to Numerical data this involves two steps:

Integer Encoding

One-Hot Encoding

1. Integer Encoding: As a first step, each unique category value is assigned an integer value. For example, “red” is 1, “green” is 2, and “blue” is 3. This is called a label encoding or an integer encoding and is easily reversible. For some variables, this may be enough. The integer values have a natural ordered relationship between each other, and machine learning algorithms may be able to understand and harness this relationship. For example, ordinal variables like the “place” example above would be a good example where a label encoding would be sufficient.

2. One-Hot Encoding: For some categorical variables, the integer encoding is not enough. Moreover, using the integer encoding technique, we allow the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories). In such cases, one hot encoding is useful which adds more meaning to the data. One-Hot Encoding is applied to the integer representation where the integer encoded variable is removed and a new binary variable is added for each unique integer value. In the “color” variable example, there are 3 categories and therefore 3 binary variables are needed. A “1” value is placed in the binary variable for the color and “0” values for the other colors. During the process of designing a multi class classifier, it is recommended to reshape the output attribute from a vector, this reshaped output contains values for each class separately which defines the class value for every object differently. This type of encoding is called as one hot vector.

<table>
<thead>
<tr>
<th>Class or label</th>
<th>One hot vector encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setosa</td>
<td>[1 0 0]</td>
</tr>
<tr>
<td>Versicolour</td>
<td>[0 1 0]</td>
</tr>
<tr>
<td>Virginica</td>
<td>[0 0 1]</td>
</tr>
</tbody>
</table>

Figure 7. splitting dataset into training and testing part

Figure 8. One hot vector coding

A. Software Requirement

Python 3 Jupyter Notebook. Users can easily install Python on their own computer and use the standard and extend library. Scalability: Programmers can write their code in C or C++ and run them in Python. It also helps to implement various regression, clustering and classification algorithms and is designed to interoperate with the Python numerical libraries NumPy and based on Python and it helps to figure out how to implement this algorithm in programming. NumPy, Scipy and Matplotlib In Python, there is no data type called array. In order to implement the data type of array with python, numpy and scipy are the essential libraries for analyzing and calculating data. They are all open source libraries. Numpy is mainly used for the matrix calculation. scipy is developed based on numpy and it is mainly used for scientific research. By using them in Python programming, they can be used with three simple commands:
>>> import numpy
>>> import scipy
>>> import pandas
Then Python will call the methods from numpy and scipy.
Matplotlib is a famous library for plotting in Python.

It provides a series of API and it is suitable for making interactive mapping. In this case, we need to use it to find the best result visually. Preparing the data set for this paper, we used Iris flower and the data set can be found in UCI Machine Learning Repositor (Bache & Lichman 2013). In this paper, we use the famous Fisher’s Iris data set. The data set of Iris flower can be also found in the python library, 0 represents setosa, 1 represents versicolor, 2 represents virginica. In the process of preparing a training data set and a testing data set, the greatest problem is how to find the most appropriate way to divide the data set into training data set and testing data set. In some cases, by using sampling theory and estimation theory, we can separate the whole data set into training data set and testing data set. However, sometimes, the method would be changed. Different objects have different properties and attributes and a data set varies on basis of those differences. Thus, in this kind of situation, in order to achieve better results, the data set will be separated according to the property of attributes of the data set. Using Python to implement the program, for good implementation and good compatibility, Python version 3 will be in use. The Integrated Development Environment used in this case would be Jupyter Notebook.

B. Architecture of network
1. The model used has 4 input nodes for given 4 features.
2. Then we have hidden layer(s) with nodes that have self loop and feed their output as input to next layer.
3. At output layer we have 3 nodes as at output we have three values representing the unique code for each class.

![Figure 9](image)

**Figure 9. Architecture of the network**

4. In the model we are using sigmoid function as activation function and SGD (stochastic gradient descent) function for updating or optimizing the weights during training.
5. After training the model for 1600 iterations we test the model using test data and calculated the accuracy and error percentage.

B. Architecture of network
1. The model used has 4 input nodes for given 4 features.
2. Then we have hidden layer(s) with nodes that have self loop and feed their output as input to next layer.
3. At output layer we have 3 nodes as at output we have three values representing the unique code for each class.

**C. Coding snipets:**

```python
# compute sigmoid nonlinearity
def sigmoid(m):
    output = ((np.exp(m) - np.exp(-m)) / (np.exp(m) + np.exp(-m)))
    return output

# convert output of sigmoid function to its derivative
def sigmoid_output_to_derivative(output):
    return (1 - (output)**2)
```

Our nonlinearity sigmoid function as it is easy to evaluate and shows good results in this case and converting the output of this activation function to its derivative.

```python
# initialize neural network weights
synapse_0 = np.random.random((input_dim, hidden_dim))
synapse_1 = np.random.random((hidden_dim, output_dim))
synapse_h = np.random.random((hidden_dim, hidden_dim))

synapse_0_update = np.zeros_like(synapse_0)
synapse_1_update = np.zeros_like(synapse_1)
synapse_h_update = np.zeros_like(synapse_h)
```

This snippet shows that the weight matrix is connecting our input layer and our hidden layer. After merging several weight updates, we actually update the matrices. The (input_dim) are as rows and (hidden_dim) are as columns. The second weight matrix is connecting the hidden layer to the output layer, in which (hidden_dim) are as rows and (output_dim) as columns. The next weight matrix connects the hidden layer in the previous time-step to the hidden layer in the current timestep. It also connects the hidden layer in the current timestep to the hidden layer in the next timestep.

```python
imtest = pd.read_csv("C:/Users/LENOVO/Desktop/input_test.csv")
outtest = pd.read_csv("C:/Users/LENOVO/Desktop/output_test.csv")
```

This part of code shows that the division of input testing part and the output training part.
D. Screen shots

After evaluation this screenshot, it shows us the clear vision of desired and the predicted value of the values which are encoded in a hot vector form.

VII. CONCLUSION AND FUTURE WORK

Deep Learning is currently used in many fields which extracts abstract features and gives better feature classification. The RNN model used for pattern recognition and how RNN’s works in pattern recognition. Each hidden layer is characterized by its own weights, activation and biases which work independently. In order to combine the layer, the weight and bias should be same for these hidden layers and then can be unrolled in a recurrent layer. Now the recurrent neuron acts as a single neuron. In this manner the neuron saves state of previous input and combines with current input state whereas some relation remains preserved. The problem of multiclassification has been described by taking multivariate IRIS Flower Data Set in hand where BackPropagation Through Time (BPTT) Algorithm is applied when the RNN is being unrolled across time. Some problems which come across while training of deep neural networks using gradient-based learning methods is when more number of hidden layers are added it might expose the network to vanishing gradients and the exploding gradients which become a risk for the network to perform inaccurately. These problems will be considered for the future work and the possible solutions to overcome.

VIII. REFERENCES


[4]. Supervised Sequence Labelling with Recurrent Neural Networks, Alex Graves

[5]. PAUL J. WEBROS “Backpropagation through time: what it does and how to do it”