JIMMA UNIVERSITY
INSTITUTE OF TECHNOLOGY
FACULTY OF COMPUTING

Program: COMPUTER SCIENCE

Assignment:data mining

TITLE:RECURRENT NEURAL NETWORKS

Group name

Name	IDNO
Nigatu Kifle	RU 0536/08
Ramadan Mohammed	RU0590/08
Nebil Mohammed	RU1334/07

Abstract

This paper provides guidance to some of the concepts surrounding recurrent neural networks. Contrary to feedforward networks, recurrent networks can be sensitive, and be adapted to past inputs. Backpropagation learning is described for feedforward networks, adapted to suit our (probabilistic) modeling needs, and extended to cover recurrent networks. The aim of this brief paper is to set the scene for applying and understanding recurrent neural networks.

Introduction

it is well known that conventional feedforward neural networks can be used to approximate *any* spatially finite function given a (potentially very large) set of hidden nodes.

That is, for functions which have a *fixed* input space there is always a way of encoding these functions as neural networks.For a two-layered network, the mapping consists of two steps, $y(t) = G(F(x(t)))$*:* (1)

We can use automatic learning techniques such as backpropagation to find the weights of the network (G and F) if sufficient samples from the function is available. Recurrent neural networks are fundamentally different from feedforward architectures in the sense that they not only operate on an input space but also on an internal *state* space – a trace of what already has been processed by the network. This is equivalent to an Iterated Function System (IFS; see (Barnsley, 1993) for a general introduction to IFSs; (Kolen, 1994) for a neural network perspective) or a Dynamical System (DS; see e.g. (Devaney, 1989) for a general introduction to dynamical systems; (Tino et al., 1998; Casey, 1996) for neural network perspectives). The state space enables the representation (and learning) of temporally/sequentially extended dependencies over unspecified (and potentially infinite)

Intervals according to
$y(t) = G(s(t))$ (2)
$s(t) = F(s(t ¡ 1); x(t))$*:* (3)

To limit the scope of this paper and simplify mathematical matters we will assume that the network operates in discrete time steps (it is perfectly possible to use continuous time instead). It turns out that if we further assume that weights are at least rational and continuous output functions are used, networks are capable of representing *any* Turing Machine (again assuming

that any number of hidden nodes are available). This is important since we then know that all that can be computed, can be processed1 equally well with a discrete time recurrent neural network. It

Artificial Neural Network(ANN) uses the processing of the brain as a basis to develop algorithms that can be used to model complex patterns and prediction problems. Lets begin by first understanding how our brain processes information: In our brain, there are billions of cells called neurons, which processes information in the form of electric signals. External information/stimuli is received by the dendrites of the neuron, processed in the neuron cell body, converted to an output

and passed through the Axon to the next neuron. The next neuron can choose to either accept it or reject it depending on the strength of the signal.

Neural Networks and Data Mining

An Artificial Neural Network, often just called a neural network, is a mathematical model inspired by biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. In most cases a neural network is an adaptive system that changes its structure during a learning phase. Neural networks are used to model complex relationships between inputs and outputs or to find patterns in data.

The inspiration for neural networks came from examination of central nervous systems. In an artificial neural network, simple artificial nodes, called "neurons", "neuroses", "processing elements" or "units", are connected together to form a network which mimics a biological neural network.

There is no single formal definition of what an artificial neural network is. Generally, it involves a network of simple processing elements that exhibit complex global behavior determined by the connections between the processing elements and element parameters. Artificial neural networks are used with algorithms designed to alter the strength of the connections in the network to produce a desired signal flow.

Neural networks are also similar to biological neural networks in that functions are performed collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which various units are assigned. The term "neural network" usually refers to models employed in statistics, cognitive psychology and artificial intelligence. Neural network models which emulate the central nervous system are part of theoretical neuroscience and computational neuroscience.

In modern software implementations of artificial neural networks, the approach inspired by biology has been largely abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks or parts of neural networks (such as artificial neurons) are used as components in larger systems that combine both adaptive and nonadaptive elements. While the more general approach of such adaptive systems is more suitable for real-world problem solving, it has far less to do with the traditional artificial intelligence connectionist models.

What they do have in common, however, is the principle of non-linear, distributed, parallel and local processing and adaptation. Historically, the use of neural networks models marked a paradigm shift in the late eighties from high-level (symbolic) artificial intelligence, characterized by expert systems with knowledge embodied in if-then rules, to lowlevel (sub-symbolic) machine learning, characterized by knowledge embodied in the parameters of a dynamical system.

Description of basic algorithm

Actual Algorithm:

1. Initialize the weights in the network (often randomly)
2. Repeat

* for each example e in the training set do

1. neural-net-output(network, e) ; forward pass
2. T = teacher output for e
3. Calculate error (T - O) at the output units
4. Compute delta_ wi for all weights from hidden layer to output layer ; backward pass
5. Compute delta_ wi for all weights from input layer to hidden layer ;

backward pass continued

6. Update the weights in the network

* end

3. until all examples classified correctly or stopping criterion satisfied
4. return(network)

Applications

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical.

Real-life applications

The tasks artificial neural networks are applied to tend to fall within the following broad categories:

- Function approximation, or regression analysis,
  including time series prediction, fitnessap proximation and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind source separation and compression.
- Robotics, including directing manipulators, Computer numerical control.

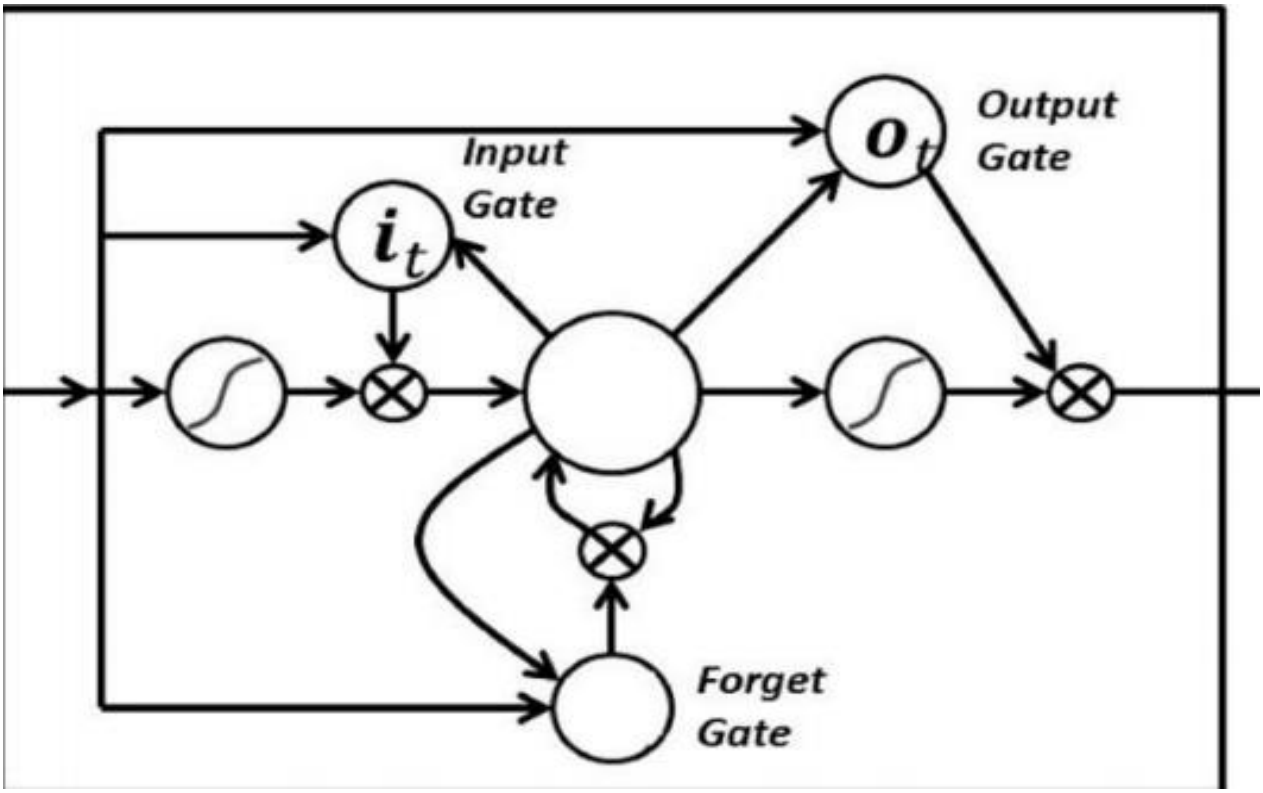
Application areas include system identification and control (vehicle control, process control, natural resources management), quantum chemistry, game-playing and decision making (backgammon, chess, poker), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications (automated trading systems), data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.

Working with Neural Network Toolbox Like its counterpart in the biological nervous system, a neural network can learn and therefore can be trained to find solutions, recognize patterns, classify data, and forecast future events. The behavior of a neural network is defined by the way its individual computing elements are connected and by the strength of those connections, or weights. The weights are automatically adjusted by training the network according to a specified learning rule until it performs the desired task correctly. Neural Network Toolbox includes command-line functions and graphical tools for creating, training, and simulating neural networks. Graphical tools make it easy to develop neural networks for tasks such as data fitting (including time-series data), pattern recognition, and clustering. After creating your networks in these tools, you can automatically generate MATLAB code to capture your work and automate tasks.

Long-Short Term Memory

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between. The units of an LSTM are used as building units for the layers of a RNN, which is then often called an LSTM network. LSTM's enable RNN's to remember their inputs over a long period of time.This is because LSTM's contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory. This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information

(e.g. if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm.This simply means that it learns over time which information is important and which not. In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate). You can see an illustration of a RNN with its three gates below:



The gates in a LSTM are analog, in the form of sigmoid, meaning that they range from 0 to 1. The fact that they are analog, enables them to do backpropagation with it. The problematic issues of vanishing gradients is solved through LSTM because it keeps the gradients steep enough and therefore the training relatively short and the accuracy high.

Backpropagation Through Time

To understand the concept of Backpropagation Through Time you definitely have to understand the concepts of Forward and Back-Propagation first. I will not go into the details here because that would be way out of the limit of this blog post, so I will try to give you a definition of these concepts that is as simple as possible but allows you to understand the overall concept of backpropagation through time. In neural networks, you basically do Forward-Propagation to get the output of your model and check if this output is correct or incorrect, to get the error. Now you do Backward-Propagation, which is nothing but going backwards through your neural network to find the partial derivatives of the error with respect to the weights, which enables you to subtract this value from the weights.

Those derivatives are then used by Gradient Descent, an algorithm that is used to iteratively minimize a given function. Then it adjusts the weights up or down, depending on which decreases the error. That is exactly how a Neural Network learns during the training process.

So with Backpropagation you basically try to tweak the weights of your model, while training. Backpropagation Through Time (BPTT) is basically just a fancy buzz word for doing Backpropagation on an unrolled Recurrent Neural Network. Unrolling is a visualization and conceptual tool, which helps you to understand what's going on within the network. Most of the time when you implement a Recurrent Neural Network in the common programming frameworks, they automatically take care of the Backpropagation but you need to understand how it works, which enables you to troubleshoot problems that come up during the development process.

How they work

We will first discuss some important facts about the "normal" Feed Forward Neural Networks, that you need to know, to understand Recurrent Neural Networks properly.

But it is also important that you understand what sequential data is. It basically is just ordered data, where related things follow each other. Examples are financial data or the DNA sequence.

The most popular type of sequential data is perhaps Time series data, which is just a series of data points that are listed in time order

CONCLUSION

In this paper, we present research on data mining based on neural network. At present, data mining is a new and important area of research, and neural network itself is very suitable for solving the problems of data mining because its characteristics of good robustness, selforganizing adaptive, parallel processing, distributed storage and high degree of fault tolerance. The combination of data mining method and neural network model can greatly improve the efficiency of data mining methods, and it has been widely used. It also will receive more and more attention. RNNs allow for processing of variable length inputs and outputs by maintaining state information across time steps. Various Input-Output scenarios are possible (Single/Multiple). Vanilla RNNs are improved upon by LSTMs which address the vanishing gradient problem through the CEC! Exploding gradients are handled by gradient clipping.

There is rarely one right tool to use in data mining it is a question as to what is available and what gives the "best" results. Many articles, in addition to those mentioned in this paper, consider neural networks to be a promising data mining tool.

Artificial Neural Networks offer qualitative methods for business and economic systems that traditional quantitative tools in statistics and econometrics cannot quantify due to the complexity in translating the systems into precise mathematical functions. Hence, the use of neural networks

in data mining is a promising field of research especially given the ready availability of large mass of data sets and the reported ability of neural networks to detect and assimilate relationships between a large numbers of variables.

In most cases neural networks perform as well or better than the traditional statistical techniques to which they are compared. Resistance to using these “black boxes” is gradually diminishing as more researchers use them, in particular those with statistical backgrounds. Thus, neural networks are becoming very popular with data mining practitioners, particularly in medical research, finance and marketing.

This is because they have proven their predictive power through comparison with other statistical techniques using real data sets. Due to design problems neural systems need further research before they are widely accepted in industry. As software companies develop more sophisticated models with user-friendly interfaces the attraction to neural networks will continue to grow.

Part of code

Code PyBrain

```
import cybrain as cb
import numpy as np
from time import time
X =     [[0.0,0.0]];     Y = [[0.0]]
X.append([1.0,0.0]); Y.append([1.0])
X.append([0.0,1.0]); Y.append([1.0])
X.append([1.0,1.0]); Y.append([0.0])
X, Y = np.array(X), np.array(Y)
nnet = cb.Network()
Lin = cb.LinearLayer(2)
Lhidden = cb.LogisticLayer(2)
Lout = cb.LogisticLayer(1)
bias = cb.BiasUnit()
nnet.inputLayers = [Lin]
nnet.hiddenLayers = [Lhidden]
nnet.outputLayers = [Lout]
nnet.autoInputLayers = [bias]
Lin.fullConnectTo(Lhidden)
Lhidden.fullConnectTo(Lout)
bias.fullConnectTo(Lhidden)
bias.fullConnectTo(Lout)
rate = 0.1
nnet.setup()
batch = cb.FullBatchTrainer(nnet, X, Y, rate)
t1 = time()
batch.epochs(2000)
print "Time CyBrain {}".format(time()-t1)
for i in range(len(X)):
    print "{} ==> {}".format(X[i],
np.array(nnet.activate(X[i:i+1,:])))
```

Outputs

```
Time CyBrain 0.211654996872
[ 0.  0.] ==> [ '0.0365560102866' ]
[ 1.  0.] ==> [ '0.951081842587'  ]
[ 0.  1.] ==> [ '0.951928021684'  ]
[ 1.  1.] ==> [ '0.0332036251855' ]

Time PyBrain 7.03572702408
[ 0.  0.] ==> [  1.67662906e-08]
[ 1.  0.] ==> [ 0.99999998]
[ 0.  1.] ==> [ 0.99999998]
[ 1.  1.] ==> [  7.30255101e-09]
```