Java Graphics Report

Ben Crabbe

University of Bristol, UK

May 22, 2015

1 Introduction

For my research project I am working with neural networks using a high level framework. To get some experience before I begin properly I have decided to use this assignment to explore some of the issues with training neural networks. I hope it will also provide some good OO design practise.

A neural network consists of layers of one or more units or 'neurons'. Each neuron recieves inputs from each of the neurons in the layer immediate below it. It computes a weighted sum of these inputs and applies some non linear 'activation' function, producing that neurons output.

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \tag{1}$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \tag{2}$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \tag{3}$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \tag{4}$$

2 Implementation

Input layers need to share get output function with layers but that is it, probably not worth using inheritance?

```
1
2  import java.util.*;
3
4  class Record
5  {
6     public class RecordException extends Exception
7     {
8        static final long serialVersionUID = 42L;
9        public RecordException()
10       {
11          super();
12       }
13
14       public RecordException(String message)
15       {
16          super(message);
17       }
18
19       public RecordException(String message, Throwable cause)
20       {
21         super(message, cause);
22       }
23
24       public RecordException(Throwable cause)
25       {
26          super(cause);
27       }
28    }
29    class Attribute
30    {
31       String name;
```

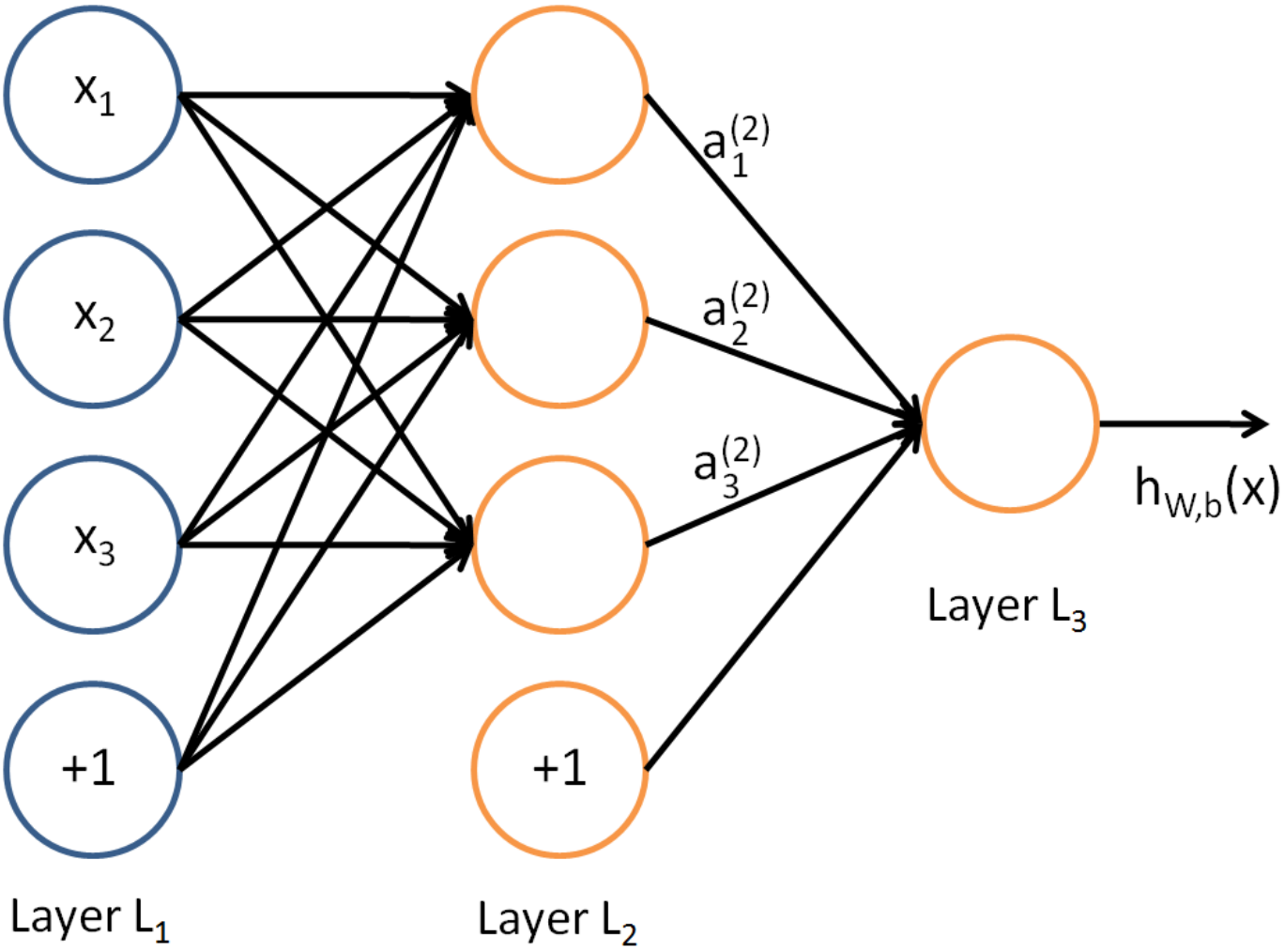

Figure 1

```
      String value;
33
      Attribute(String nameInput, String valueInput)
35    {
         name = nameInput;
37       value = valueInput;
      }
39  }
    Set<Attribute> data;
41
    Record() {
43    data = new LinkedHashSet<Attribute>();
    }
45
  void addField(String fieldName, String fieldValue) throws RecordException
47  {
      for(Attribute field: data) {
49       if(field.name==fieldName) {
            throw new RecordException("There already exists a field named " + fieldName + "in record.
    ");
51       }
      }
53    Attribute newField = new Attribute(fieldName,fieldValue);
      this.data.add(newField);
55  }

57  String getFieldValue(String fieldName) throws RecordException
  {
59    for(Attribute field: data) {
         if(field.name==fieldName) {
61          return field.value;
         }
63    }
      throw new RecordException("No field named " + fieldName + " in record.");
65  }

67  void setFieldValue(String fieldName, String fieldValue) throws RecordException
  {
69    for(Attribute field: data) {
         if(field.name==fieldName) {
71          field.value = fieldValue;
            return;
73       }
      }
75    throw new RecordException("No field named " + fieldName+ " in record.");
  }
77
  int countFields()
79  {
      return data.size();
81  }

83  public static void main(String[] args)
  {
85    Record row = new Record();
      row.testRecord();
87  }

89  void testRecord()
  {
91    try {
         this.addField("Field1","value1");
93       this.addField("Field2","value2");
         this.setFieldValue("Field2","value2.2");
95       System.out.println(this.getFieldValue("Field1"));
         System.out.println(this.getFieldValue("Field2"));
```

```
            System.out.println( this.countFields() );
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

3 Tables

I decided that the main data on the attribute names, types etc should be held by the table since every row is the same, so I split the Attribute class into two: Attribute and AttributeValue. Attributes hold the name (and other things in the future), AttributeValues, which are an inner class to Tuple (I renamed Record to fit better with the relational model) hold the values and a reference to the Attribute class it belongs to. This way the values are still directly searchable by attribute name, but the name is stored in only 1 place.

I also got rid of my custom exception. It wouldn't let me pass multiple string types to store as a message, and it didn't seem to gain much to be worth writing a bunch more constructors for.

```
import java.util.*;

class Table
{
    String name;
    List<Attribute> tableHeading;// attribute names, types, constraints etc
    Set<Tuple> tableBody;// a set of tuples

    Table(String name)
    {
        this.name = name;
        tableHeading = new ArrayList<Attribute>();//columns
        tableBody = new LinkedHashSet<Tuple>();//rows
    }

    Table(String name, List<String> fieldNames)
    {
        this.name = name;
        tableHeading = new ArrayList<Attribute>();
        tableBody = new LinkedHashSet<Tuple>();
        for(String newField: fieldNames) {
            try {
                addAttribute(newField);
            } catch(Exception e) {
                System.out.println(e.getMessage());
            }
        }
    }

    boolean attributeExists(String attributeName)
    {
        for(Attribute a: tableHeading) {
            if(a.name==attributeName) {
                return true;
            }
        }
        return false;
    }

    void addAttribute(String attributeName) throws Exception
    {
        if(attributeExists(attributeName)){
            throw new Exception("attribute " + attributeName + " already exists");
        }
```

```
45          Attribute newAtt = new Attribute(attributeName);
            tableHeading.add(newAtt);
47      }

49      int countFields()
        {
51         return tableHeading.size();
        }
53
        int countRows()
55      {
            return tableBody.size();
57      }

59      List<String> getFieldNames()
        {
61          List<String> fieldNames = new ArrayList<String>();
            for(Attribute field: tableHeading) {
63              fieldNames.add(field.name);
            }
65          return fieldNames;
        }
67
        void editAttributeName(String oldName, String newName) throws Exception
69      {
            if(attributeExists(oldName)) {
71              Attribute editMyName = getAttributeFromName(oldName);
                editMyName.name = newName;
73          }
            else {
75              throw new Exception("Could not edit attribute name. No attribute named " +
                                    oldName + " in table.");
77          }
        }
79
        private Attribute getAttributeFromName(String name) throws Exception
81      {
            for(Attribute field: tableHeading) {
83              if(field.name==name) {
                    return field;
85              }
            }
87          throw new Exception("Could not get attribute. None named " + name + " in table.");
        }
89
        void addTuple(String... values) throws Exception
91      {
            if(values.length != tableHeading.size()) {
93              throw new Exception("addTuple() for Table " + name + " expects " +
                                    tableHeading.size() + " data values. It received " +
95                                  values.length);
            }
97          else {
                Tuple newTuple = new Tuple(tableHeading, Arrays.asList(values));
99              tableBody.add(newTuple);
            }
101     }

103     void deleteTuple(String attributeName, String value) throws Exception
        {
105         if(attributeExists(attributeName)) {
                Attribute attributeToDelete = getAttributeFromName(attributeName);
107             try {
                    for(Tuple row: tableBody) {
109                     if(row.getAttributeValue(attributeName)==value) {
                            tableBody.remove(row);
```

```
111                         return;
                        }
113                 }
                } catch(Exception e) {
115                 throw new Exception("deleteTuple() there are no tuples where " +
                                        attributeName + " has a value of " + value );
                }
117         } else {
                throw new Exception("deleteTuple()  No attribute named " + attributeName +
                                    " in table " + name);
119         }
        }
123
        static void is(Object x, Object y)
125     {
            if (x == y) return;
127         if (x != null && x.equals(y)) return;
            throw new Error("Error:\n" + x + "\n" + y);
129     }

131     void testTable()
        {
133
        }
135
        public static void main(String[] args)
137     {
            Table t = new Table("testTable");
139         t.testTable();
        }
141

143 }
```