Neural Networks Project

What is this project all about?

- Artificial Neural Networks (ANNs) are one of the most versatile tools in the Machine Learning toolbox.
- Neural Nets are an attempt to model the way our own brains work. Such a network is a large number (maybe billions) of ”neurons,” each of which is incredibly simple.
- A neuron takes many inputs, combines them in some way, and generates an output. This output is either a final observable value, or it is the input to another neuron.

What are they good for?

- Mostly, neural nets are used for classification of some sort.
- Sounds a bit limiting!
- But the number of outputs can be large (can distinguish between many classes) and “deep learning” which is basically the use of multi-layer neural nets, with potentially lots of outputs,
- Neural nets are used for speech recognition, 3-D scene recognition (autonomous vehicles) and many, many other things!

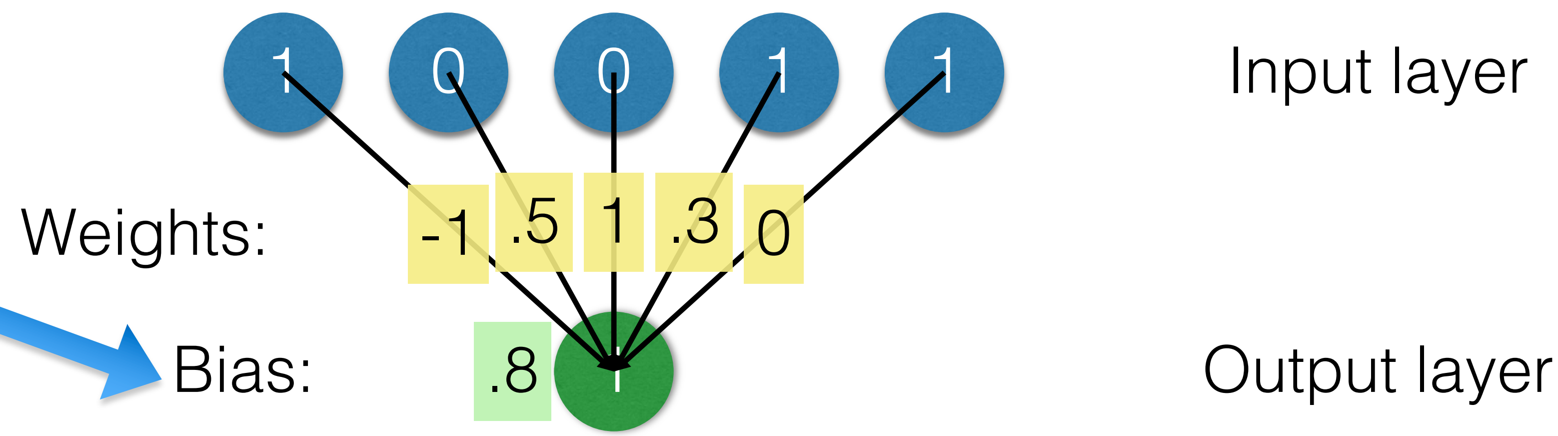
Perceptron Classifier

- What is a Perceptron?
  - A *perceptron* is a very simple form of ANN
    - with neurons arranged in layers such that its “feed-forward” connections are formed only between layers (not between neurons of the same layer)
    - each neuron outputs a binary signal (the “message”)
    - one of the first types of neural net to be built (1957)
    - and able to *classify* input sets with one or more (orthogonal) labels
    - for each neuron, output is calculated as follows:
      - $f(x) = 1$ if $w \cdot x + b > 0$; otherwise 0 — dot product
      - where ***w*** is the weight vector of connections from the previous layer, and
      - where ***x*** is the (binary) set of outputs at the previous layer,
      - and b is the bias

Training the perceptron

- Two-layer perceptron:

can also think of bias as being the weight of an additional input connection with value constrained to be 1

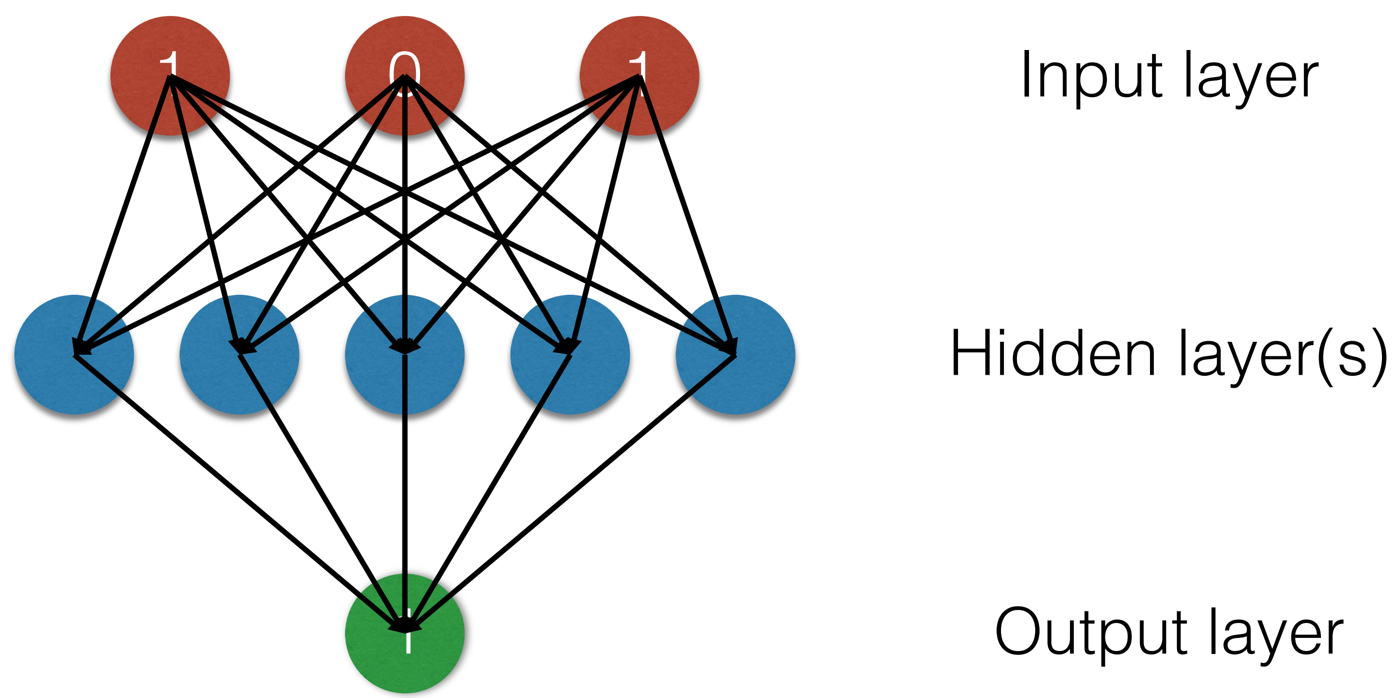


- The weights start out at random (and bias values usually zero):
    - As each new input set is labeled (supervised learning), the weights/bias are adjusted (by back-propagation of errors) until output values match as closely as possible.
    - Provided the input is *linearly separable**, the perceptron will converge (with some set of weights). The optimally stable solution is known as a *support vector machine*.

* if a hyperplane can be drawn separating the inputs

Solving non-linearly-separable problems

- Use a multi-layer perceptron:



- Now you can solve problems such as XOR

Back-propagation, etc.

- Remember our Newton-Raphson convergence from the first week or two?
  - Essentially, perceptrons use an N-dimensional version of the same thing where N is the number of weights (including bias) that must be adjusted and where the function is linear.
- In practice, perceptrons aren't strictly binary at each neuron
  - Hidden neurons use "sigmoid" function;
  - Output neurons use "softmax" function.

Perceptron code

```
package edu.neu.coe.scala.spark.nn
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.sql.Row
object PerceptronClassifier extends App {
    val conf = new SparkConf().setAppName("perceptron")
    val sc = new SparkContext(conf)
    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    val sparkHome = "/Applications/spark-1.5.1-bin-hadoop2.6/"
    val trainingFile = "data/mllib/sample_multiclass_classification_data.txt"
    // this is used to implicitly convert an RDD to a DataFrame.
    import sqlContext.implicits._
    // Load training data
    val data = MLUtils.loadLibSVMFile(sc, s"$sparkHome$trainingFile").toDF()
    // Split the data into train and test
    val splits = data.randomSplit(Array(0.6, 0.4), seed = 1234L)
    val train = splits(0)
    val test = splits(1)
    // specify layers for the neural network:
    // input layer of size 4 (features), two intermediate of size 5 and 4 and output of size 3 (classes)
    val layers = Array[Int](4, 5, 4, 3)
    // create the trainer and set its parameters
    val trainer = new MultilayerPerceptronClassifier()
            .setLayers(layers)
            .setBlockSize(128)
            .setSeed(1234L)
            .setMaxIter(100)
    // train the model
    val model = trainer.fit(train)
    // compute precision on the test set
    val result = model.transform(test)
    val predictionAndLabels = result.select("prediction", "label")
    val evaluator = new MulticlassClassificationEvaluator()
            .setMetricName("precision")
    println("Precision:" + evaluator.evaluate(predictionAndLabels))
}
```

input file: label followed by four input neuron values

```
1 1:-0.222222 2:0.5 3:-0.762712 4:-0.833333
1 1:-0.555556 2:0.25 3:-0.864407 4:-0.916667
1 1:-0.722222 2:-0.166667 3:-0.864407 4:-0.833333
1 1:-0.722222 2:0.166667 3:-0.694915 4:-0.916667
0 1:0.166667 2:-0.416667 3:0.457627 4:0.5
1 1:-0.833333 3:-0.864407 4:-0.916667
2 1:-1.32455e-07 2:-0.166667 3:0.220339 4:0.0833333
2 1:-1.32455e-07 2:-0.333333 3:0.0169491 4:-4.03573e-08
1 1:-0.5 2:0.75 3:-0.830508 4:-1
0 1:0.611111 3:0.694915 4:0.416667
```

result:

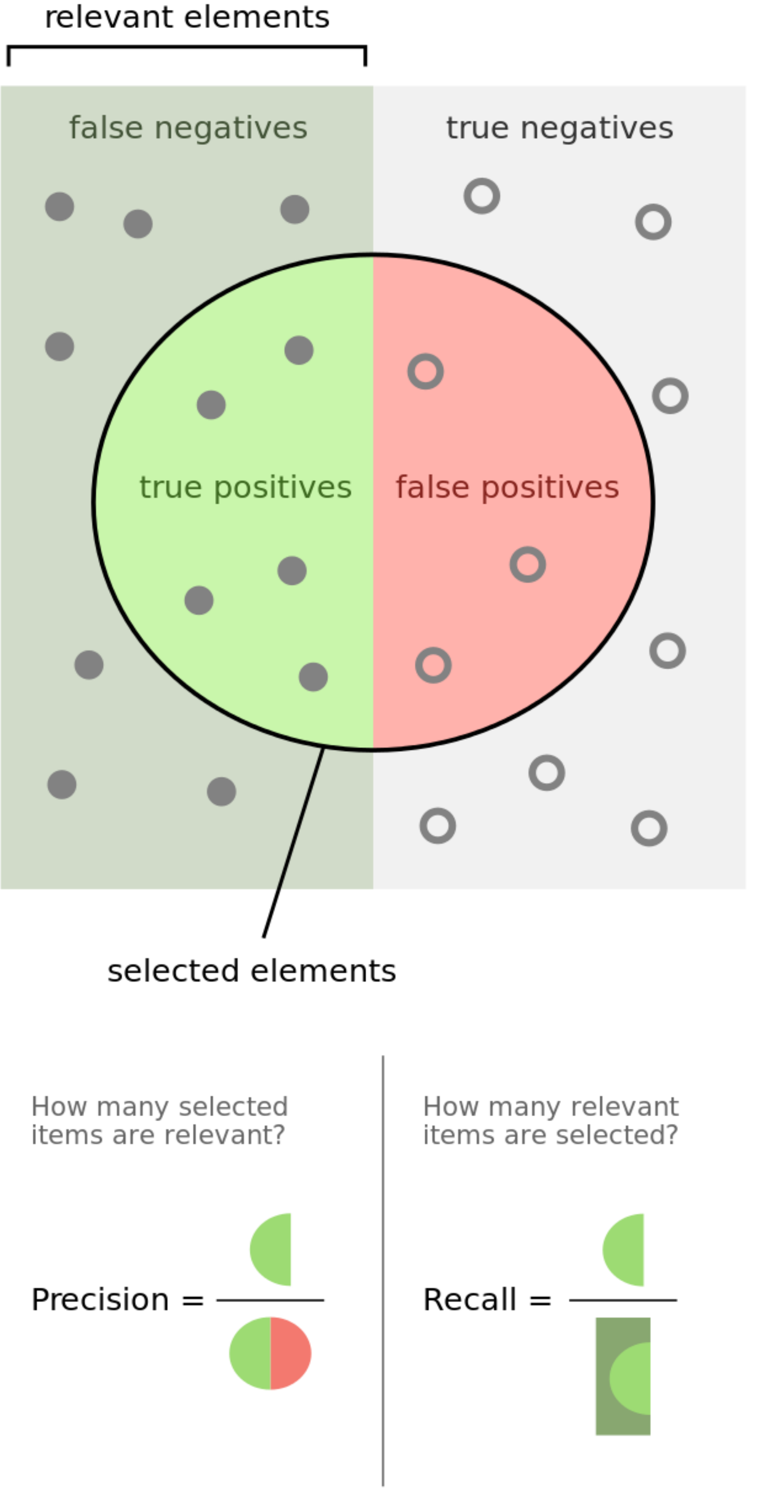
```
+----------+-----+
|prediction|label|
+----------+-----+
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  0.0|
|       2.0|  2.0|
|       0.0|  2.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       1.0|  1.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       2.0|  2.0|
|       1.0|  1.0|
|       0.0|  0.0|
|       0.0|  0.0|
+----------+-----+
only showing top 20 rows
```

Why Neural Nets?

- I believe that ANNs are primarily useful because they:
  - are very general;
  - do not require much advance modeling thought;
  - are resistant to mutual information (co-variance);
  - can be used with very large feature sets and training sets—
    - in fact, multi-layer ANNs essentially do the feature selection for you;
  - can be used to gain insights for better modeling.
- What to avoid:
  - without sophisticated optimization (gradient descent, etc.) algorithms, convergence may be slow—or worse
  - MLlib uses L-BFGS (limited memory version of Broyden–Fletcher–Goldfarb–Shanno algorithm)

Classification

- Measurements
  - Precision/Recall (applicable for two classes)
    - e.g. for a recall of 50%, we want a precision of 60%
  - Confusion matrix (for > two classes)



What you have to do

- Form up in pairs;
- Build a neural net implementation:
  - Code (proper project structure using maven);
  - Unit tests (important);
  - Committed to github.
- Create an application/dataset for it and demonstrate its capabilities for .
- Write a report discussing your findings
- Submit before the deadline. Aug 9th.