Neural Networks for Computing

Perceptron

Hikmat Farhat

September 18, 2017

Introduction

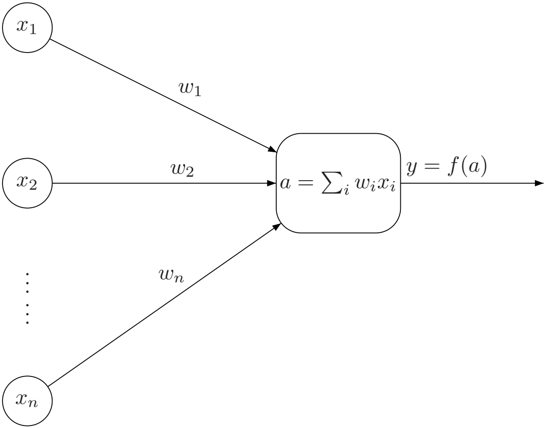
- Early days of AI computers solved problems that humans found difficult: playing chess, solving graph problems.
- The challenge is to solve problems that are easy for humans: identifying objects.
- Typically, learning from experience and recognizing patterns.
- One AI approach is to model the knowledge in the world in formal languages. When this is done AI can use inference rules to deduce properties of the world.
- Hard-coded knowledge is difficult to build. Better have the machine learn by itself.

Nerual Networks

- One approach for making machines(software, algorithms) learn is Neural Networks
- There are many variants of neural networks
  - Feedforward
  - Convolution
  - recurrent
  - etc.
- All of those models are more or less similar
- They use the same building block: perceptron or neuron (in the broader sense)
- So a neural network is a network (or graph) of connected perceptrons

Perceptron

- A perceptron is a non-linear computational unit inspired by the human neuron.



Perceptron

- The perceptron has n inputs $x_1, \ldots, x_n$ and each input x_i has an associated weight w_i.
- First step it computes the quantity $\sum_i w_i x_i$.
- Then adds the *bias* b (not shown in the figure) to obtain

$$a = \sum_i w_i x_i + b$$

- Finally it applies a nonlinear function f to the result

$$y = f(a)$$

Geometric Interpretation

- Consider the equation

$$a = \sum_i w_i x_i + b$$

- It can be written in vector form as

$$a = w^T \cdot x + b$$

- Where $\cdot$ is the dot product and w^T is the transpose of

$$\begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$

- i.e $w^T = [w_1 \dots w_n]$

Geometric Interpretation

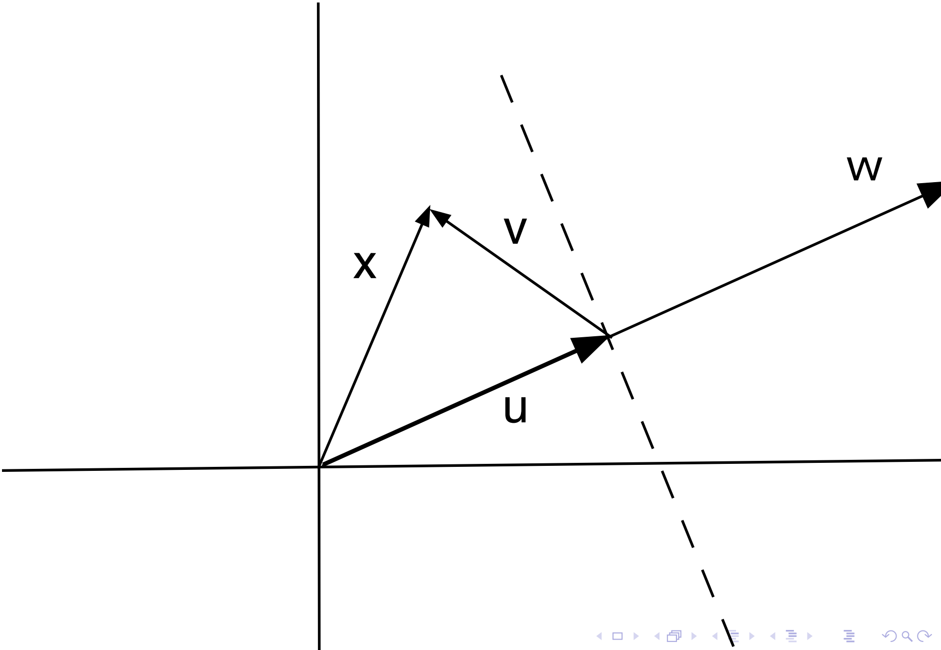
- Consider the plane normal to the vector w.
- Let u be the vector parallel to w and of length $b/|w|$.
- Any vector x can be written as $x = u + v$ so

$$w \cdot x = w \cdot u + w \cdot v$$

- Since w and u are parallel then $w \cdot u = |w||u| = b$ therefore for any point x we have

$$w \cdot x = b + w \cdot v$$

- All points on the plane are such that $w \cdot v = 0$, to one side of the plane are such that $w \cdot v < 0$ and the other side are such that $w \cdot v > 0$



w
v
x
u

First Example

- Given a set of points belonging to classes $\mathcal{C}_1$ and $\mathcal{C}_1$ we would like an algorithm that learns to classify correctly any point.
- To do so we assume we have two sets $\mathcal{H}_1 \subseteq \mathcal{C}_1$ and $\mathcal{H}_2 \subseteq \mathcal{C}_2$ that are properly labeled.
- The algorithm learns from the data in $\mathcal{H}_1$ and $\mathcal{H}_2$ to obtain values for w and b.
- After the learning stage the values of w and b obtained in the learning phase the algorithm will be used to predict to which class a given input point belongs to .

Notational Simplifications

- It is convenient to extend the inputs from $x_0, \ldots x_n$ to $x_0, \ldots, x_n, x_{n+1}$ and $w_0, \ldots, w_n$ to $w_0, \ldots, w_n, w_{n+1}$
- where $x_{n+1} = 1$ and $w_{n+1} = -b$.
- In this case the equation becomes

$$w^T \cdot x = 0$$

Perceptron Learning Algorithm

- As a first example we will use a simple algorithm to learn from a 2-dimensional data.
- Given an input x and a current value for w and b the algorithm works as follows
  1. If $x \in \mathcal{H}_1$ and $w \cdot x \leq 0$ do nothing
  2. If $x \in \mathcal{H}_2$ and $w \cdot x \geq 0$ do nothing
  3. If $x \in \mathcal{H}_1$ and $w \cdot x > 0$ then $w = w - x$.
  4. If $x \in \mathcal{H}_2$ and $w \cdot x < 0$ then $w = w + x$.
  5. We will use another simplification. All points $x \in \mathcal{H}_1$ we convert them to $x = -x$.
  6. This way we will have only two rules: 2 and 4.

Introduction to Python

- Python is a dynamically typed language i.e. variables are bound to objects at execution time.
- It is interpreted which makes ideal it for prototyping.
- It is open source
- You can start the interpreter by typing *python* on the prompt.
- You exit the interpreter by either typing *quit()* or CONTROL-D on Unix or CONTROL-Z on Windows

Simple Examples

```
>>>x=12
>>>y=3
>>>x*y
36
>>>y="hello" #y is bound to a different object
>>>print y
hello
>>>x=" there"#x is bound to a different object
>>>y+x #concatenate two strings
'hello there'
>>>y[0]
'h'
```

Lists

```
>>>x=[1,2,3]
>>>x[0]
1
>>>x[0:2] # the last index is not included
[1,2]
>>>y=[['a',5,"hello"],[3,"b","there"]]
>>>y[0]
['a', 5, 'hello']
>>>y[0][1]
5
```

Control Structures

```
>>>x=[7,12,3]
>>>if len(x) > 4 :
...   print "larger"
...else:
...   print "smaller"
...
smaller
>>> for i in x:
...      print i
...
7
12
3
>>>
```

List Comprehension

- List comprehensions are convenient in creating lists

```
>>>x=[i**2 for i in range(5)]
>>>x
[0,1,4,9,16]
>>>x[1]=44 #lists are not immutable
>>>x
[0,44,4,9,16]
>>>x=[i**2 for i in range(7) if i%2==0 or i%3==0]
>>>x
[0,4,9,16,36]
```

Perceptron Algorithm in Python

- We will use the perceptron learning algorithm to "learn" the logical AND function
- Since there are only 4 possible inputs for the AND function we iterate many times over the input.
- We read the data from the file having the following format

```
4
0 0 0
0 1 0
1 0 0
1 1 1
```

- The first value denotes the number of lines (four).
- Subsequent lines contain the input (the first two values) and the output of the AND function.

- The two classes in this example are the ones that have output 0 and the ones that have output 1
- Therefore in the input we have three samples in the first class and one sample in the second class.
- Since the number of samples is very small we iterate repeatedly over them until the algorithm converges (no change).
- Running the code we see that the algorithm converges after 8 iterations.
- We plot the result with the points of the AND function.
- Recall that we are solving for $w_1x_1 + w_2x_2 + w_3x_3 = 0$. Since $x_3 = 1$ then we obtain an equation of the line

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_3}{w_2}$$

Utility functions

```
def add(w,v,row):
  for i in range(3):
    w[i]=w[i]+v[row][i]
```

```
def dotproduct(w,v,row):
  sum=0
  for i in range(3):
    sum=sum+w[i]*v[row][i]

  return sum

f=open("data.txt")
rows=int(next(f))
x=[[int(i) for i in line.split()] for line in f]
```

Main code

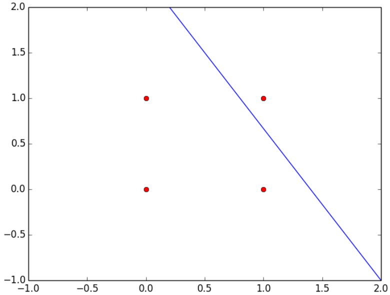
```
    for i in range(rows):
        if x[i][2]==0:
            x[i][0]=-x[i][0]
            x[i][1]=-x[i][1]
            x[i][2]=-1
        else:
            x[i][2]=1

w=[0.5,0.5,0.5]
changed=True
while changed:
    changed=False
    for i in range(rows):
        result=dotproduct(x,w,i)
        if result <=0:
            add(x,w,i)
            changed=True
print w
```

Result for AND function

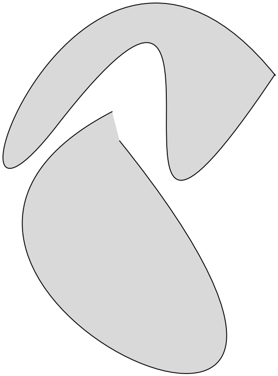
- The output of our simple program is [2.5,1.5,-3.5]. We can plot the result with the values of the AND function in python.

```
import matplotlib.pyplot as plt
def line(w1,w2,w3,x):
    y=[]
    for i in x:
      y.append(-w1/w2*i-w3/w2)
    return y
x=[0,1,2,3,4]
y=line(2.5,1.5,-3.5,x)
plt.plot(x,y)
plt.plot([0,0,1,1],[0,1,0,1],'ro')
plt.axis([-1,2,-1,2])
plt.show()
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Separable Classes

- We have applied the perceptron algorithm to a simple problem and saw that we obtained a results after 8 iterations.
- The question is: does it always converge (terminate)?
- The answer is a qualified yes: it will converge only if the two classes are **separable**.
- Below is a figure of two classes that are **not separable**



Algorithm Convergence

- We show that if the two classes are **separable** then the algorithm terminates after a finite number of steps.
- Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the two classes and let $\mathcal{H}_1 \subseteq \mathcal{C}_1$ and $\mathcal{H}_2 \subseteq \mathcal{C}_2$ be the training sets.
- Since we are assuming that $\mathcal{C}_1$ and $\mathcal{C}_2$ are separable then **there exists** w_0 such that

$$\begin{aligned} \forall x \in \mathcal{C}_1 \quad & w_0 \cdot x > 0 \\ \forall x \in \mathcal{C}_2 \quad & w_0 \cdot x < 0 \end{aligned}$$

- Let $x_0, \ldots, x_n$ be a sequence of inputs that were misclassified. From our perceptron algorithm we have (Note that for the second class we take the negative of x)

$$w(n+1) = w(n) + x(n)$$

- Iterating the above equation together with the initial condition of $w(0) = 0$ we get

$$w(n+1) = x(1) + \ldots + x(n)$$

- Let

$$\alpha = \min_{x \in \mathcal{H}_1} w_0 \cdot x$$

- Then $w_0 \cdot w(n+1) \geq \alpha n$

- Using Schwartz inequality

$$|w_0|^2|w(n+1)|^2 \geq (w_0 \cdot w(n+1))^2$$

- We get

$$|w(n+1)|^2 \geq \frac{\alpha^2 n^2}{|w_0|^2} \tag{1}$$

- On the other hand, since $w(n+1) = w(n) + x(n)$ then

$$|w(n+1)|^2 = |w(n)|^2 + |x(n)|^2 + 2w(n) \cdot x(n)$$

- But $x(1), \ldots, x(n)$ are misclassified so $w(n) \cdot x(n) \leq 0$ thus

$$\begin{aligned}
|w(n+1)|^2 &\leq |w(n)|^2 + |x(n)|^2 \\
|w(n)|^2 &\leq |w(n-1)|^2 + |x(n-1)|^2 \\
&\cdots\cdots \\
|w(2)|^2 &\leq |w(1)|^2 + |x(1)|^2 \\
|w(1)|^2 &\leq |w(0)|^2 + |x(0)|^2
\end{aligned}$$

- By adding the above inequalities we get

$$|w(n+1)|^2 \leq \sum_{i=1}^{n} |x(i)|^2$$

- Let $\beta = \max_{x\in\mathcal{H}} |x|^2$ then

$$|w(n+1)|^2 \leq \beta n \tag{2}$$

- Therefore we have obtained a lower bound for $|w(n+1)|^2$ in equation (1) and an upper bound in equation (2).
- Combining both equations we obtained a value for the maximum number of iterations

$$n_{max} = \frac{\beta |w_0|^2}{\alpha^2} \tag{3}$$

Vector Operations

- The python module numpy has builtin optimize versions for vector operations
- for example the dot product can be done directly in numpy
- The numpy package can take advantage of our hardware
- this is why we will use the numpy operations from now on

Example

```
import numpy as np
import time

a=np.random.rand(1000000)
b=np.random.rand(1000000)
start=time.time()
c=np.dot(a,b)
end=time.time()
print("vectorized version "+str(end-start))

start=time.time()
c=0
for i in range(1000000):
    c+=a[i]*b[i]
end=time.time()
print("loop version "+str(end-start))
```

Learning to identify ships

- We will use a modified version of what we have learned to be able to identify images of ships
- The dataset that we will be using, CIFAR-10, contains images of 10 different types(classes) of objects.
- One of the classes is for ships. We will use our perceptron to learn to identify ships.
- Our model, we hope,once it finishes "learning", will output 1 if the input is an image of a ship and 0 otherwise

Sigmoid Function

- First we modify the output of our perceptron by using what is called the sigmoid function.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- The sigmoid function introduces non-linearity
- It is differentiable
- Being differentiable is an important property as we will see later.

Input images

- One way to represent images is by using a three-dimensional array
- The first two dimensions denote the pixels of the image
- The third dimension gives the Red-Green-Blue values of the each pixel
- In the case of the CIFAR-10 dataset representing images of 32x32 pixels, the image is flattened
- The first 1024 values are the Red values of the pixels, the next 1024 are the Green values and finally the last 1024 are the Blue values.
- So our input is a vector of dimension 3072 with values between 0 and 255

Input vector

- The CIFAR-10 dataset contains 60,0000 images of 10 classes with 6000 image for each class
- They are divided into training(50,000) and test (10,000) images. The training set is also divided into 5 batches (files)
- When we read the data from a given file we are reading 10,000 samples of vectors
- As you will see we will process them as a single batch rather than iterating 10,000 times.

- Let x^i be the input vector for sample i with size $n = 3072$
- Then m samples(vectors) are represented as a matrix

$$X = \begin{bmatrix} x_1^1 & x_1^2 & \dots & x_1^m \\ x_2^1 & x_2^2 & \dots & x_2^m \\ \dots & \dots & \dots & \dots \\ x_n^1 & x_n^2 & \dots & x_n^m \end{bmatrix}$$

- The output of our model becomes

$$\hat{y} = \sigma(w^T \cdot X + b)$$

- Where σ is the sigmoid function introduced earlier

- Sometimes it helps to visualize the above computation

$$\left[\hat{y}_1 \ldots \hat{y}_m\right] = \sigma\left(\left[w_1 \ldots w_n\right] \cdot \begin{bmatrix} x_1^1 & x_1^2 & \ldots & x_1^m \\ x_2^1 & x_2^2 & \ldots & x_2^m \\ \ldots & \ldots & \ldots & \ldots \\ x_n^1 & x_n^2 & \ldots & x_n^m \end{bmatrix} + b\right)$$

- So according to the above $\hat{y}_i$ is the output for the i^{th} sample.
- The function σ applied to a matrix is defined as the matrix obtained by applying the function to every element in it.
- Also adding b to a matrix is defined as adding b to every element.

Interpretation of the output

- Now we have a method to compute $\hat{y}$ given input x (if we can determine w and b).
- But what is exactly $\hat{y}$?
- Since we are dealing with a binary classification problem (ship or not-ship) then we interpret $\hat{y}$ as
- The probability that $y = 1$ given x.
- In other words, given a image represented by x, what is the probability that it is a ship?
- If,for example, $\hat{y} = 0.99$ so most probably x is an image of a ship
- If ,for example, $\hat{y} = 0.2$, it is unlikely that x is an image of a ship

Cost function

- To compute the "optimal" values of w and b we need to minimize the "error" of our prediction.
- For a given sample, x, the "error" of our prediction $\hat{y}$ depends on how "closely" it predicts the value of y, the label associated with the sample.
- In this example we will use the following function for the "difference" also called the cross-entropy

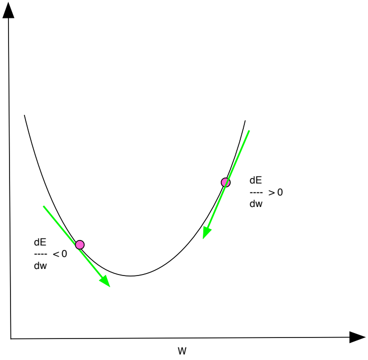
$$-\left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right)$$

- We compute the average error over all the samples as

$$E = -\frac{1}{m}\sum_{i=1}^{m} y^i \log \hat{y}^i + (1-y^i)\log(1-\hat{y}^i) = \frac{1}{m}\sum_{i=1}^{m} E^i$$

Gradient Descent

- Given the error E as defined previously we want to find w and b such that E is minimum
- We illustrate the gradient descent idea by using a curve in 2-d.



Gradient Descent

- From the figure above we see that if the derivative is positive then to find the minimum w should be decreased
- If the derivative is negative then w should be increased
- In both cases w is updated as

$$w = w - \alpha \frac{dE}{dw}$$

- Where α is a parameter than determines the rate of which w is updated
- Clearly when $\frac{dE}{dw} = 0$ then w will not change since the minimum has been reached.

Computing the derivative

- To use gradient descent we need to compute the derivative $\frac{\partial E}{\partial w}$ and $\frac{\partial E}{\partial b}$.
- Note that since $E = \sum_{i=1} E^i$ it is enough to get the derivatives of E^i then average over the number of samples.
- Using the chain rule we can write

$$\frac{\partial E^i}{\partial w} = \frac{\partial E^i}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial w}$$

- We compute each term

$$\begin{aligned}\frac{\partial E^i}{\partial \hat{y}^i} &= \frac{\partial}{\partial \hat{y}^i}\left[-y^i \log \hat{y}^i - (1-y^i)\log(1-\hat{y}^i)\right] \\ &= -\frac{y^i}{\hat{y}^i} + \frac{1-y^i}{1-\hat{y}^i}\end{aligned}$$

- On the other hand let $z^i = w^T \cdot x^i + b$ and $\hat{y}^i = \sigma(z^i)$ then

$$\frac{\partial \hat{y}^i}{\partial w} = \frac{\partial \hat{y}^i}{\partial z^i}\frac{\partial z^i}{\partial w}$$

- we have

$$\frac{\partial \hat{y}^i}{\partial z^i} = \frac{\partial}{\partial z^i}\frac{1}{1+e^{-z^i}} = \frac{e^{-z^i}}{(1+e^{-z^i})^2}$$
$$= \frac{-1+(1+e^{-z^i})}{(1+e^{-z^i})^2} = \hat{y}^i - \hat{y^i}^2$$

- and since $\frac{\partial z^i}{\partial w} = x^i$ then

$$\frac{\partial \hat{y}^i}{\partial w} = (\hat{y^i} - \hat{y^i}^2) \cdot x^i$$
$$= \hat{y}^i(1 - \hat{y}^i) \cdot x^i$$

- Combining all partial results we get

$$\begin{aligned}\frac{\partial E^i}{\partial w} &= \left[-\frac{y^i}{\hat{y}^i} + \frac{1-y^i}{1-\hat{y}^i}\right]\left[\hat{y}^i(1-\hat{y}^i)\cdot x^i\right] \\ &= \left[-y^i(1-\hat{y}^i) + (1-y^i)\hat{y}^i\right]\cdot x^i \\ &= (\hat{y}^i - y^i)\cdot x^i\end{aligned}$$

- To get the derivatives of E we average over all samples

$$\frac{\partial E}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)x^i$$

- We can reuse all the partial computations and the fact that $\frac{\partial z^i}{\partial b} = 1$ to get

$$\frac{\partial E}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)$$

Implementation

- Typically when we apply the algorithm we would have m samples.
- Let x_j^i be the j^{th} input of the i^{th} sample. Similarly, y^i is the label of the i^{th} sample.
- Recall that we average over all samples

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{m} (\hat{y}^i - y^i) x_j^i$$

- In vector notation

$$dw = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \dots \\ \frac{\partial E}{\partial w_n} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} x_1^1 & \dots & x_1^m \\ x_2^1 & \dots & x_2^m \\ \dots & \dots & \dots \\ x_n^1 & \dots & x_n^m \end{bmatrix} \cdot \begin{bmatrix} \hat{y}^1 - y^1 \\ \hat{y}^2 - y^2 \\ \dots \\ \hat{y}^m - y^m \end{bmatrix}$$

Implementation

- Or using the transpose of $\left[(\hat{y}^m - y^1)\ldots(\hat{y}^m - y^m)\right]$

$$dw = X \cdot (\hat{y} - y)^T$$

- Similarly

$$\frac{\partial E}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^i - y^i)$$

- To summarize
- $\hat{y}$, y and w are row vectors, dw is a column vector then the formulas we need are (where α is the learning rate)

$$\hat{y} = \sigma(w \cdot x + b)$$

$$dw = \frac{1}{m} x \cdot (\hat{y} - y)^T$$

$$db = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^i - y^i)$$

$$b = b - \alpha db$$

$$w = w - \alpha dw^T$$

Python Implementation

- Let $A = \hat{y}$, *sigmoid* be the function that computes $\sigma(z)$,*np* is the numpy package,learning_rate=α then the previous equations can be written in Python as

```
A=sigmoid(np.dot(w,x)+b)
dw=np.dot(x,(A-Y).T)/m
db=np.sum(A-Y)/m
w=w-learning_rate*dw.T
b=b-learning_rate*db
cost=-np.sum(np.multiply(Y,np.log(A))
                +np.multiply((1-Y),np.log(1-A)))/m
```

- where np.multiply multiplies vectors component wise.

Python Implementation

- First we give the general architecture of the implementation using functions that will be defined later
- The algorithm works as follows
  1. Read learning input data into arrays X and Y and test input data into arrays X_{test} and Y_{test}.
  2. Since the data contains 10 classes and we are making a binary decision only i.e. ship or not ship we convert all data labeled ship to value 1 and all others to 0
  3. Define a function **propagate** to compute the output given an input. This function computes the "difference" between the output and the "true" output.
  4. Define a function **optimize** that does multiple iterations:each iteration uses the function **propagate**
  5. Once the iterations finish our computation produces values for w and b
  6. We use these computed values to test the accuracy of prediction for the test data