In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```python
train_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.csv"
train_df = pd.read_csv(train_url) #training set
test_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv"
test_df = pd.read_csv(test_url) #test set
```

In [3]:
```python
train_df.isna().sum()
# So Age, Cabin and Embarked have missing values
```

Out[3]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [4]:
```python
test_df.isna().sum()
```

Out[4]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [5]:
```python
train_df.info()
test_df.info()
# So we will have to impute the missing values
# Seem like Cabin missing TOO MUCH values, so we will drop it later
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

In [6]:

```
# Fill values with median !!FROM TRAINING DATA!!
train_df["Age"] = train_df["Age"].fillna(train_df["Age"].median())
test_df["Age"] = test_df["Age"].fillna(train_df["Age"].median())

# T8: Median age of training data is
train_df["Age"].median()
```

Out[6]:

```
28.0
```

In [7]:

```
# Fill values with most common value !!FROM TRAIN_dfING DATA!!
train_df["Embarked"] = train_df["Embarked"].fillna(train_df["Embarked"].value_count
test_df["Embarked"] = test_df["Embarked"].fillna(train_df["Embarked"].value_counts(

# T9: Most common port of embarked is
train_df["Embarked"].value_counts().idxmax()
```

Out[7]:

```
'S'
```

In [8]:

```
# Fare and PClass seems to be correlated, so we will use PClass to impute Fare
test_df["Fare"] = test_df["Fare"].fillna(train_df.groupby("Pclass")["Fare"].transfo
```

In [9]:

```
embarked_categories = dict([(k,i) for i,k in enumerate(train_df["Embarked"].astype(
train_df["EmbarkedClass"] = train_df["Embarked"].map(embarked_categories)
test_df["EmbarkedClass"] = test_df["Embarked"].map(embarked_categories)
```

In [10]:

```
sex_categories = dict([(k,i) for i,k in enumerate(train_df["Sex"].astype('category'
train_df["SexClass"] = train_df["Sex"].map(sex_categories)
test_df["SexClass"] = test_df["Sex"].map(sex_categories)
```

In [11]:

```
train_df.isna().sum(),test_df.isna().sum()

# Data is quite cleaned!!
```

Out[11]:

```
(PassengerId        0
 Survived           0
 Pclass             0
 Name               0
 Sex                0
 Age                0
 SibSp              0
 Parch              0
 Ticket             0
 Fare               0
 Cabin            687
 Embarked           0
 EmbarkedClass      0
 SexClass           0
 dtype: int64,
 PassengerId        0
 Pclass             0
 Name               0
 Sex                0
 Age                0
 SibSp              0
 Parch              0
 Ticket             0
 Fare               0
 Cabin            327
 Embarked           0
 EmbarkedClass      0
 SexClass           0
 dtype: int64)
```

In [12]:

```
train_data = np.array(train_df[["Pclass","SexClass","Age","EmbarkedClass"]].values,
train_label = np.array(train_df["Survived"].values,dtype=np.int8).reshape(-1,1)
train_data
```

```
Out[12]: array([[ 3.,  1., 22.,  2.],
       [ 1.,  0., 38.,  0.],
       [ 3.,  0., 26.,  2.],
       ...,
       [ 3.,  0., 28.,  2.],
       [ 1.,  1., 26.,  0.],
       [ 3.,  1., 32.,  1.]], dtype=float32)
```

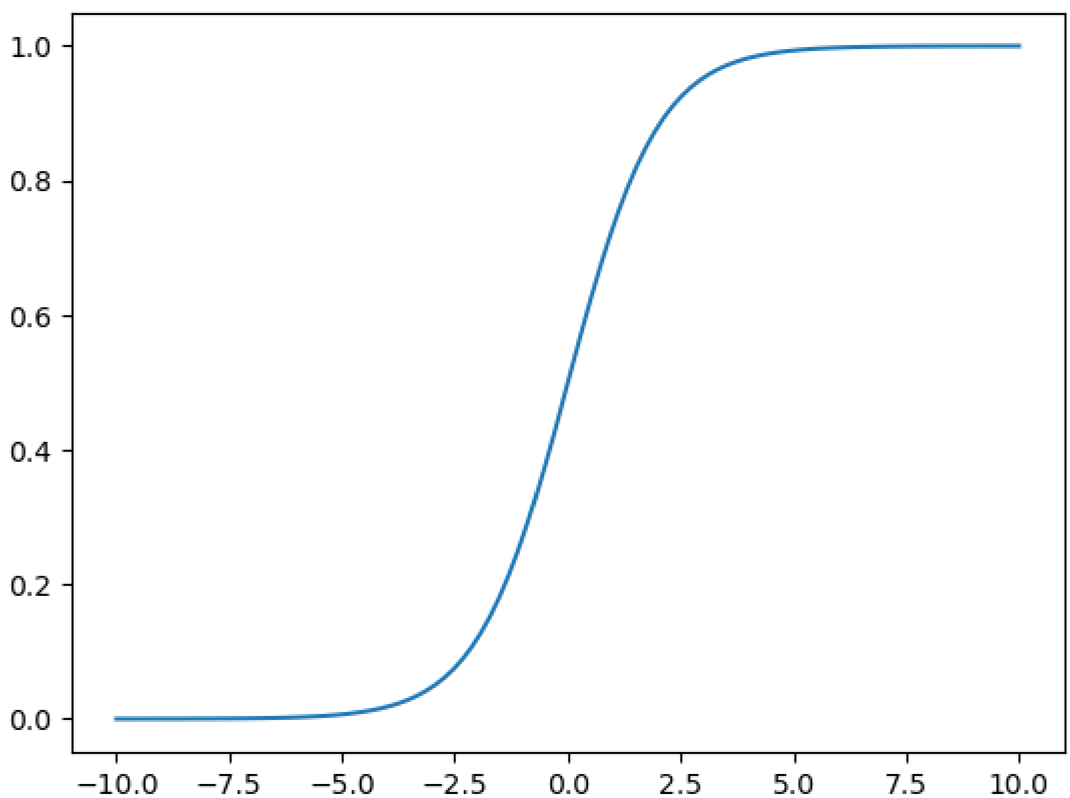
```
In [13]: test_data = np.array(test_df[["Pclass","SexClass","Age","EmbarkedClass"]].values,dt
test_data
```

```
Out[13]: array([[ 3. ,  1. , 34.5,  1. ],
       [ 3. ,  0. , 47. ,  2. ],
       [ 2. ,  1. , 62. ,  1. ],
       ...,
       [ 3. ,  1. , 38.5,  2. ],
       [ 3. ,  1. , 28. ,  2. ],
       [ 3. ,  1. , 28. ,  0. ]], dtype=float32)
```

```
In [14]: def sigmoid(z): return 1/(1+np.exp(-z))

x = np.linspace(-10,10,100)
z = sigmoid(x)
plt.plot(x,z)
```

```
Out[14]: [<matplotlib.lines.Line2D at 0x7f93fad7f3e0>]
```




```
In [15]: class LogisticRegression:
  def  __init__(self,train_x,train_y,test_x,learning_rate=1e-3) -> None:
    self.train_x = train_x
    self.train_y = train_y
```

```
    self.pred = np.zeros_like(train_y)
    self.pred_class = np.zeros_like(train_y)
    self.test_x = test_x
    self.cost,self.grad = self.cost_function()
    self.learning_rate = learning_rate
    self.W = self.__random_init_param()
    self.test_accuracies = []

  def __random_init_param(self):
    #size of X is [m, n] where m=sample, n=features
    W = np.random.randn(len(self.train_x[0]), 1) # +1 for Bias term
    return W

  def __sigmoid(self,z): return 1/(1+np.exp(-z))

  def __add_bias(self,X):
    Bias = np.ones((len(X), 1))
    res = np.concatenate((Bias, X), axis=1)
    return res

  def cost_function(self):
    if type(self.pred) == "None" : return None
    m = len(self.train_y)
    loss = np.dot(-self.train_y.T, np.log(self.__sigmoid(self.pred)+1e-10))-np.dot(
    cost = (1/m) * loss
    grad = (1/m) * (np.dot(self.train_x.T, (self.__sigmoid(self.pred)-self.train_y)
    return cost.astype('float64'), grad.astype('float64')

  def predict(self, X=None,sigmoid=True):
    if X is None : X = self.train_x
    h = np.dot(X, self.W)
    return self.__sigmoid(h) if sigmoid else h

  def train_accuracy(self):
    return np.squeeze(np.squeeze((sum(self.train_y == self.pred_class)/len(self.tra


  def step(self):
    #update parameters
    self.W = self.W - self.learning_rate * self.grad
    self.grad = 0

  def train(self,epoch=int(1e+8),interuption_step=10,logging_step=None):
    if logging_step is None : logging_step = epoch**0.5
    loss_step = 0
    best_W = self.W
    for i in range(1,epoch+1):
      self.pred = self.predict(None,False)
      self.pred_class = np.where(self.__sigmoid(self.pred) >= 0.5, 1, 0)
      cost, grad = self.cost_function()
      self.grad = grad
      self.step()
      if i%logging_step == 0: print(f"Epoch : {i}/{epoch}, Train Accuracy :{self.tr
      if cost.item() < self.cost.item() :
        loss_step = 0
        best_W = self.W
```

```python
      else :
        loss_step += 1
        if loss_step == interuption_step :
          print(f"Loss is increasing, stop training at epoch {i}")
          self.W = best_W
          break
      self.cost = cost

  def test(self):
    self.test_pred = self.predict(self.test_x)
    return self.test_pred
```

In [16]:

```python
model = LogisticRegression(train_data,train_label,test_data,0.001)
model.train()
print("Cost :",model.cost_function()[0],"\tTrain accuracy :",model.train_accuracy()
```

```
Epoch : 10000/100000000, Train Accuracy :71.49%, Cost : 0.56660
Epoch : 20000/100000000, Train Accuracy :77.33%, Cost : 0.54443
Epoch : 30000/100000000, Train Accuracy :77.67%, Cost : 0.53633
Epoch : 40000/100000000, Train Accuracy :77.67%, Cost : 0.53302
Epoch : 50000/100000000, Train Accuracy :78.00%, Cost : 0.53162
Epoch : 60000/100000000, Train Accuracy :77.67%, Cost : 0.53100
Epoch : 70000/100000000, Train Accuracy :77.55%, Cost : 0.53073
Epoch : 80000/100000000, Train Accuracy :77.55%, Cost : 0.53061
Epoch : 90000/100000000, Train Accuracy :77.67%, Cost : 0.53056
Epoch : 100000/100000000, Train Accuracy :77.89%, Cost : 0.53053
Epoch : 110000/100000000, Train Accuracy :77.89%, Cost : 0.53052
Epoch : 120000/100000000, Train Accuracy :77.89%, Cost : 0.53052
Epoch : 130000/100000000, Train Accuracy :77.89%, Cost : 0.53051
Epoch : 140000/100000000, Train Accuracy :77.89%, Cost : 0.53051
Epoch : 150000/100000000, Train Accuracy :77.78%, Cost : 0.53051
Epoch : 160000/100000000, Train Accuracy :77.78%, Cost : 0.53051
Epoch : 170000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 180000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 190000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 200000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 210000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 220000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 230000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 240000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 250000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 260000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 270000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 280000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 290000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 300000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 310000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 320000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Epoch : 330000/100000000, Train Accuracy :77.67%, Cost : 0.53051
Loss is increasing, stop training at epoch 335870
Cost : [[0.53051089]]   Train accuracy : 77.665544332211
```

In [17]:

```python
res = model.test()
pred_class = np.where(res>=0.5,1,0)
test_df["Survived"] = pred_class
```

```
test_df[["PassengerId","Survived"]].to_csv("submission.csv",index=False)
test_df
```

Out[17]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875
...	...	...	...	...	...	...	...	...	...
413	1305	3	Spector, Mr. Woolf	male	28.0	0	0	A.5. 3236	8.0500
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500
416	1308	3	Ware, Mr. Frederick	male	28.0	0	0	359309	8.0500
417	1309	3	Peter, Master. Michael J	male	28.0	1	1	2668	22.3583

418 rows × 14 columns

T11: My submission score is 0.76076

submission.csv
Complete · 9h ago

0.76076

In [30]:
```
poly_features_train = np.array(train_df[["Pclass","SexClass","Age","EmbarkedClass"]
poly_features_test = np.array(test_df[["Pclass","SexClass","Age","EmbarkedClass"]].
for i in range(train_data.shape[1]):
  poly_features_train = np.concatenate((poly_features_train,train_data[:,i].reshape
  poly_features_test = np.concatenate((poly_features_test,test_data[:,i].reshape(-1
  for j in range(i,train_data.shape[1]):
    poly_features_train = np.concatenate((poly_features_train,train_data[:,i].resha
    poly_features_test = np.concatenate((poly_features_test,test_data[:,i].reshape(
```

In [31]:
```
poly_features_train.shape,poly_features_test.shape
```

Out[31]:
```
((891, 18), (418, 18))
```

In [45]:
```
model2 = LogisticRegression(poly_features_train,train_label,poly_features_test,3e-5
model2.train()
res2 = model2.test()
res2_class = np.where(res2>=0.5,1,0)
print("Cost :",model2.cost_function()[0],"\tTrain accuracy :",model2.train_accuracy
test_df["Survived"] = res2_class
test_df[['PassengerId','Survived']].to_csv('submission2.csv',index=False)
```

```
/tmp/ipykernel_294501/3797932207.py:18: RuntimeWarning: overflow encountered in exp
  def __sigmoid(self,z): return 1/(1+np.exp(-z))
Epoch : 10000/100000000, Train Accuracy :42.99%, Cost : 4.41231
Epoch : 20000/100000000, Train Accuracy :71.04%, Cost : 2.40492
Loss is increasing, stop training at epoch 23068
Cost : [[1.40367608]]   Train accuracy : 75.75757575757575
```

T12: My submission has acccuracy of model2 is worse than fisrt try

submission2.csv
Complete · 9h ago

0.7488

In [29]:
```
model3 = LogisticRegression(np.array(train_df[["SexClass","Age"]].values,dtype=np.f
model3.train(1000000)
print("Cost :",model.cost_function()[0],"\tTrain accuracy :",model.train_accuracy()
res3 = model3.test()
res3_class = np.where(res3>=0.5,1,0)
test_df["Survived"] = res3_class
test_df[['PassengerId','Survived']].to_csv('submission3.csv',index=False)
```

```
Epoch : 1000/1000000, Train Accuracy :61.62%, Cost : 0.60242
Epoch : 2000/1000000, Train Accuracy :61.62%, Cost : 0.58151
Epoch : 3000/1000000, Train Accuracy :78.68%, Cost : 0.56721
Epoch : 4000/1000000, Train Accuracy :78.68%, Cost : 0.55732
Epoch : 5000/1000000, Train Accuracy :78.68%, Cost : 0.55038
Epoch : 6000/1000000, Train Accuracy :78.68%, Cost : 0.54547
Epoch : 7000/1000000, Train Accuracy :78.68%, Cost : 0.54196
Epoch : 8000/1000000, Train Accuracy :78.68%, Cost : 0.53942
Epoch : 9000/1000000, Train Accuracy :78.68%, Cost : 0.53757
Epoch : 10000/1000000, Train Accuracy :78.68%, Cost : 0.53621
Epoch : 11000/1000000, Train Accuracy :78.68%, Cost : 0.53521
Epoch : 12000/1000000, Train Accuracy :78.68%, Cost : 0.53447
Epoch : 13000/1000000, Train Accuracy :78.68%, Cost : 0.53392
Epoch : 14000/1000000, Train Accuracy :78.68%, Cost : 0.53351
Epoch : 15000/1000000, Train Accuracy :78.68%, Cost : 0.53320
Epoch : 16000/1000000, Train Accuracy :78.68%, Cost : 0.53297
Epoch : 17000/1000000, Train Accuracy :78.68%, Cost : 0.53279
Epoch : 18000/1000000, Train Accuracy :78.68%, Cost : 0.53266
Epoch : 19000/1000000, Train Accuracy :78.68%, Cost : 0.53256
Epoch : 20000/1000000, Train Accuracy :78.68%, Cost : 0.53249
Epoch : 21000/1000000, Train Accuracy :78.68%, Cost : 0.53243
Epoch : 22000/1000000, Train Accuracy :78.68%, Cost : 0.53238
Epoch : 23000/1000000, Train Accuracy :78.68%, Cost : 0.53235
Epoch : 24000/1000000, Train Accuracy :78.68%, Cost : 0.53233
Epoch : 25000/1000000, Train Accuracy :78.68%, Cost : 0.53231
Epoch : 26000/1000000, Train Accuracy :78.68%, Cost : 0.53229
Epoch : 27000/1000000, Train Accuracy :78.68%, Cost : 0.53228
Epoch : 28000/1000000, Train Accuracy :78.68%, Cost : 0.53227
Epoch : 29000/1000000, Train Accuracy :78.68%, Cost : 0.53227
Epoch : 30000/1000000, Train Accuracy :78.68%, Cost : 0.53226
Epoch : 31000/1000000, Train Accuracy :78.68%, Cost : 0.53226
Epoch : 32000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 33000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 34000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 35000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 36000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 37000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 38000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 39000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 40000/1000000, Train Accuracy :78.68%, Cost : 0.53225
Epoch : 41000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 42000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 43000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 44000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 45000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 46000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 47000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 48000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 49000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 50000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 51000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 52000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 53000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 54000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 55000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 56000/1000000, Train Accuracy :78.68%, Cost : 0.53224
```

```
Epoch : 57000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 58000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 59000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 60000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 61000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 62000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 63000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 64000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 65000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 66000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 67000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 68000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 69000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 70000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 71000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 72000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 73000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 74000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 75000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 76000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 77000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 78000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 79000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 80000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 81000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 82000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 83000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 84000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 85000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 86000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 87000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 88000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 89000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 90000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 91000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 92000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 93000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 94000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 95000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 96000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 97000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 98000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 99000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 100000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 101000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 102000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Epoch : 103000/1000000, Train Accuracy :78.68%, Cost : 0.53224
Loss is increasing, stop training at epoch 103268
Cost : [[0.53051089]]   Train accuracy : 77.665544332211
```

T13: Seems a bit better?

12484	**Patrick Cho #2**	0.76555	10	21m

🙂 Your Best Entry!
Your submission scored 0.72488, which is not an improvement of your previous score. Keep trying!