In [6]:

```
from termcolor import colored
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import IPython
%matplotlib inline

import keras
from keras.layers import Input, Dense, Dropout
from keras.layers import BatchNormalization
from keras.models import Model
from keras.optimizers import Adam
from keras import backend
from keras.models import load_model
from keras.layers import Dropout, Masking
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.text import hashing_trick
from keras.preprocessing.text import text_to_word_sequence
from sklearn.model_selection import train_test_split
```

The following presentation will demonstrate the use of two ML approaches for solving a classification problem.

The first model is a simple fully connected logistic regression model.

The second model is one I'm particularly excited about, the Generative Adversarial Network (GAN).

GAN's are constructed by pitting two networks against each other in a zero sum game. The Discriminator is trained to detect fake loan applications and labels whether an application is likely to be approved or rejected. The Generator is then fed noise to produce fake loan applications to the Discriminator with the objective of passing as a valid loan. As the Discriminator learns to detect fake applications it forces the Generator to learn more clever ways to pass a fake. Essentially, the Generator will learn the fundamental distribution of the dataset. Over time, each network reinforces the others weights to achieve an optimal equilibrium.

It is worth nothing that GAN's have been successful in several areas including datasets with limited labelled data and image generation.

Let's start by downloading and loading all HMDA data filtered by state of Connecticut for 2014-2016 limited to those intended for home purchase.

In [221]:

```
# Note: link to data and filter settings
# Note: https://www.consumerfinance.gov/data-research/hmda/explore#!/as_of_year=201

data = pd.read_csv('/home/shant/Downloads/hmda/hmda_lar.csv',low_memory=False,heade
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 171251 entries, 0 to 171250
Data columns (total 78 columns):
action_taken                      171251 non-null int64
action_taken_name                 171251 non-null object
agency_code                       171251 non-null int64
agency_abbr                       171251 non-null object
agency_name                       171251 non-null object
applicant_ethnicity               171251 non-null int64
applicant_ethnicity_name          171251 non-null object
applicant_income_000s             148174 non-null float64
applicant_race_1                  171251 non-null int64
applicant_race_2                  648 non-null float64
applicant_race_3                  36 non-null float64
applicant_race_4                  8 non-null float64
applicant_race_5                  9 non-null float64
applicant_race_name_1             171251 non-null object
applicant_race_name_2             648 non-null object
applicant_race_name_3             36 non-null object
applicant_race_name_4             8 non-null object
applicant_race_name_5             9 non-null object
applicant_sex                     171251 non-null int64
applicant_sex_name                171251 non-null object
application_date_indicator        171251 non-null int64
as_of_year                        171251 non-null int64
census_tract_number               170932 non-null float64
co_applicant_ethnicity            171251 non-null int64
co_applicant_ethnicity_name       171251 non-null object
co_applicant_race_1               171251 non-null int64
co_applicant_race_2               192 non-null float64
co_applicant_race_3               8 non-null float64
co_applicant_race_4               2 non-null float64
co_applicant_race_5               1 non-null float64
co_applicant_race_name_1          171251 non-null object
co_applicant_race_name_2          192 non-null object
co_applicant_race_name_3          8 non-null object
co_applicant_race_name_4          2 non-null object
co_applicant_race_name_5          1 non-null object
co_applicant_sex                  171251 non-null int64
co_applicant_sex_name             171251 non-null object
county_code                       171051 non-null float64
county_name                       171051 non-null object
denial_reason_1                   11674 non-null float64
denial_reason_2                   2545 non-null float64
denial_reason_3                   455 non-null float64
denial_reason_name_1              11674 non-null object
denial_reason_name_2              2545 non-null object
denial_reason_name_3              455 non-null object
edit_status                       21002 non-null float64
edit_status_name                  21002 non-null object
hoepa_status                      171251 non-null int64
hoepa_status_name                 171251 non-null object
```

```
lien_status                        171251 non-null int64
lien_status_name                   171251 non-null object
loan_purpose                       171251 non-null int64
loan_purpose_name                  171251 non-null object
loan_type                          171251 non-null int64
loan_type_name                     171251 non-null object
msamd                              162077 non-null float64
msamd_name                         162077 non-null object
owner_occupancy                    171251 non-null int64
owner_occupancy_name               171251 non-null object
preapproval                        171251 non-null int64
preapproval_name                   171251 non-null object
property_type                      171251 non-null int64
property_type_name                 171251 non-null object
purchaser_type                     171251 non-null int64
purchaser_type_name                171251 non-null object
respondent_id                      171251 non-null object
sequence_number                    171251 non-null int64
state_code                         171251 non-null int64
state_abbr                         171251 non-null object
state_name                         171251 non-null object
hud_median_family_income           170932 non-null float64
loan_amount_000s                   171251 non-null int64
number_of_1_to_4_family_units      170932 non-null float64
number_of_owner_occupied_units     170903 non-null float64
minority_population                170932 non-null float64
population                         170932 non-null float64
rate_spread                        4653 non-null float64
tract_to_msamd_income              170927 non-null float64
dtypes: float64(23), int64(21), object(34)
memory usage: 101.9+ MB
```

At first glance we notice that most of the data has a scalar value and an associated label denoted by column names ending with "_name".

There are several fields indicating the reason for denying the application. We will exclude these features since that information is relevant after the target value has been determined.

Respondent ID stands out as valuable information related to the mortgage issuer. Let's convert these text columns to scalar values by using pandas factorize method.

Let's remove all the redundant name data and define our features.

Let's also look at the dataframe description to see if we should pre-normalize the data or use Batch Normalization instead.

In [224]:

```
data['issuer'] = data[['respondent_id']].apply(lambda col: pd.factorize(col)[0])

features = [
    'agency_code','applicant_ethnicity','applicant_income_000s','applicant_race_1',
    'applicant_race_4','applicant_race_5','applicant_sex','application_date_indicat
    'co_applicant_ethnicity','co_applicant_race_1','co_applicant_race_2','co_applic
    'co_applicant_race_5','co_applicant_sex','county_code',
    'edit_status',
    'hoepa_status','hud_median_family_income','lien_status','loan_amount_000s','loa
    'msamd','number_of_1_to_4_family_units','number_of_owner_occupied_units','owner
    'property_type','purchaser_type','rate_spread',
    'issuer',
    'sequence_number','state_code','tract_to_msamd_income']
```

Next, let's explore the different values associated with actions_taken in the distribution and define our target Y. There are 8 unique responses in the dataset. Since we are only interested in predicting which applications will be approved let's consolidate the 8 values into 0 for denial, 1 for approval.

For the purposes of this exercise we will assume that the following actions will be considered an approval:

```
1: Loan originated
```

We will assume the following actions will be considered denial:

```
3: Application denied by financial institution
```

We will drop all other actions since it is not explicit with respect to the objective function.

In [225]:

```
for k,v in data.groupby(['action_taken','action_taken_name']):
    print(k,":\t Count:",len(v))
data = data[data.action_taken.isin([1,3])]
data['approved'] = data.action_taken.isin([1]).astype(int)
data = data.fillna(0)

print("")
print("Y values:")
for k,v in data.groupby(['approved','action_taken_name']):
    print(k,":\t Count:",len(v))
```

```
(1, 'Loan originated') :         Count: 99210
(3, 'Application denied by financial institution') :     Count: 14220

Y values:
(0, 'Application denied by financial institution') :     Count: 14220
(1, 'Loan originated') :         Count: 99210
```

Let's explore the feature description

In [226]:

```
pd.set_option('display.float_format', lambda x: '%.3f' % x)
# print("---- Min ----")
# print(data[features].min())
# print("---- Max ----")
# print(data[features].max())
print(data[features].describe())
```

```
       agency_code  applicant_ethnicity  applicant_income_000s  \
count   113430.000           113430.000             113430.000   
mean         6.559                2.021                129.000   
std          2.297                0.462                213.266   
min          1.000                1.000                  0.000   
25%          5.000                2.000                 56.000   
50%          7.000                2.000                 85.000   
75%          9.000                2.000                139.000   
max          9.000                4.000               9999.000   

       applicant_race_1  applicant_race_2  applicant_race_3  applicant
_race_4  \
count        113430.000        113430.000        113430.000        113
430.000   
mean              4.825             0.019             0.001          
  0.000   
std               0.910             0.297             0.071          
  0.030   
min               1.000             0.000             0.000          
  0.000   
25%               5.000             0.000             0.000          
  0.000   
50%               5.000             0.000             0.000          
  0.000   
75%               5.000             0.000             0.000          
  0.000   
max               7.000             5.000             5.000          
  5.000   

       applicant_race_5  applicant_sex  application_date_indicator  \
count        113430.000     113430.000                  113430.000   
mean              0.000          1.445                       0.000   
std               0.035          0.646                       0.000   
min               0.000          1.000                       0.000   
25%               0.000          1.000                       0.000   
50%               0.000          1.000                       0.000   
75%               0.000          2.000                       0.000   
max               5.000          4.000                       0.000   

                ...             owner_occupancy  population  preapproval
  \
count           ...                  113430.000  113430.000   113430.000

mean            ...                       1.082    4932.893        2.501

std             ...                       0.291    1580.722        0.599

min             ...                       1.000       0.000        1.000

25%             ...                       1.000    3741.000        2.000

50%             ...                       1.000    4810.000        3.000
```

```
75%            ...                        1.000    5974.000        3.000

max            ...                        3.000   10289.000        3.000


       property_type  purchaser_type  rate_spread     issuer  sequence
_number  \
count     113430.000      113430.000   113430.000 113430.000       113
430.000
mean           1.015           3.127        0.082     86.700        55
185.072
std            0.160           3.217        0.433    112.092       137
275.915
min            1.000           0.000        0.000      0.000
  1.000
25%            1.000           0.000        0.000     16.000
747.000
50%            1.000           2.000        0.000     51.000         4
534.000
75%            1.000           6.000        0.000    114.000        28
354.000
max            3.000           9.000        8.530    671.000      1241
460.000

       state_code  tract_to_msamd_income
count  113430.000             113430.000
mean        9.000                114.261
std         0.000                 40.851
min         9.000                  0.000
25%         9.000                 89.540
50%         9.000                109.730
75%         9.000                134.080
max         9.000                256.740

[8 rows x 41 columns]
```

The following code blocks are intended to be executed during/after training to evaluate performance. We will load the saved test results from disk and compute basic statistics.

Let's analyze the results!

In [154]:

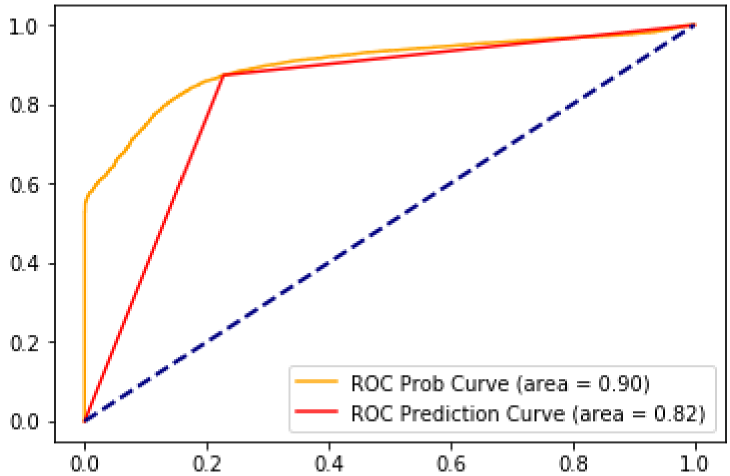
```
def analyze(filename):
    results = pd.read_csv('/home/shant/PycharmProjects/OWL/'+filename,low_memory=Fa
    mislabeled = results[results.approved != results.prediction]
    total = len(results)
    error = len(mislabeled)/total
    accuracy = 1 - error
    denied = results.loc[results.approved==0,'approved'].count()
    approved = results.loc[results.approved==1,'approved'].count()
    denied_tp = results.loc[(results.approved==0)&(results.prediction==0),'predicti
    approved_tp = results.loc[(results.approved==1)&(results.prediction==1),'predic
    roc_prob = roc_auc_score(results.approved, results.probs)
    roc_pred = roc_auc_score(results.approved, results.prediction)
    fpr,tpr,thresh = roc_curve(results.approved,results.probs)
    fpr_pred,tpr_pred,thresh_pred = roc_curve(results.approved,results.prediction)
    roc_auc = auc(fpr,tpr)
    roc_auc_pred = auc(fpr_pred,tpr_pred)
    print("Errors:            \t {:0.2f}%".format(error*100))
    print("Accuracy:          \t {:0.2f}%".format(accuracy*100))
    print("Denied             \t",denied)
    print("Approved           \t",approved)
    print("Denied True Pos    \t {} / {:0.2f}%".format(denied_tp,100*(denied_tp/de
    print("Approved True Pos  \t {} / {:0.2f}%".format(approved_tp,100*(approved_t
    print("AUC Probs.         \t {:0.2f}".format(roc_prob*100))
    print("AUC Predictions    \t {:0.2f}".format(roc_pred*100))
    plt.plot(fpr,tpr,color='orange',label='ROC Prob Curve (area = %0.2f)' % roc_auc
    plt.plot(fpr_pred,tpr_pred,color='red',label='ROC Prediction Curve (area = %0.2
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.legend()
```

First, the simple logistic regression model

In [227]:

```
analyze('results_best.txt')
```

```
Errors:                  13.89%
Accuracy:                86.11%
Denied                   2788
Approved                 19898
Denied True Pos          2154 / 77.26%
Approved True Pos        17381 / 87.35%
AUC Probs.               90.01
AUC Predictions          82.31
```

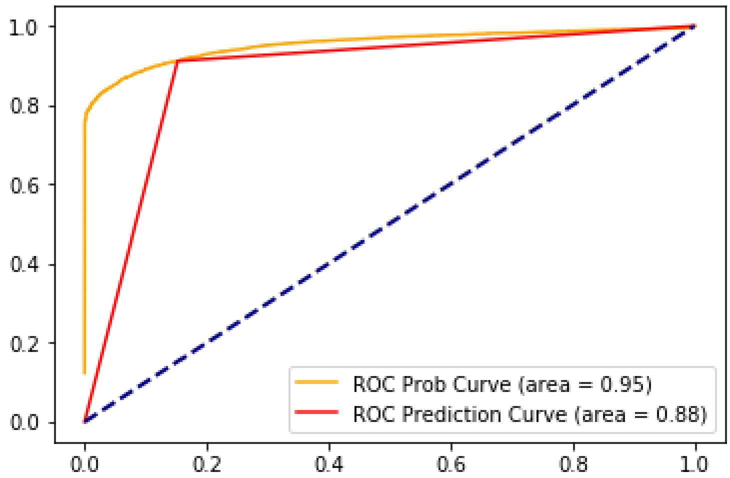


Let's do the same analysis for the GAN

In [233]:

```
analyze('results_gan_best.txt')
```

```
Errors:                  9.64%
Accuracy:                90.36%
Denied                   2788
Approved                 19898
Denied True Pos          2362 / 84.72%
Approved True Pos        18136 / 91.14%
AUC Probs.               95.45
AUC Predictions          87.93
```



The GAN stands out in this exercise for several reason.

```
AUC and accuracy are far more stable during training than the logistic regr
ession
While GAN's can be difficult/time consuming to train, it surpasses the logi
stic regression from the start
```