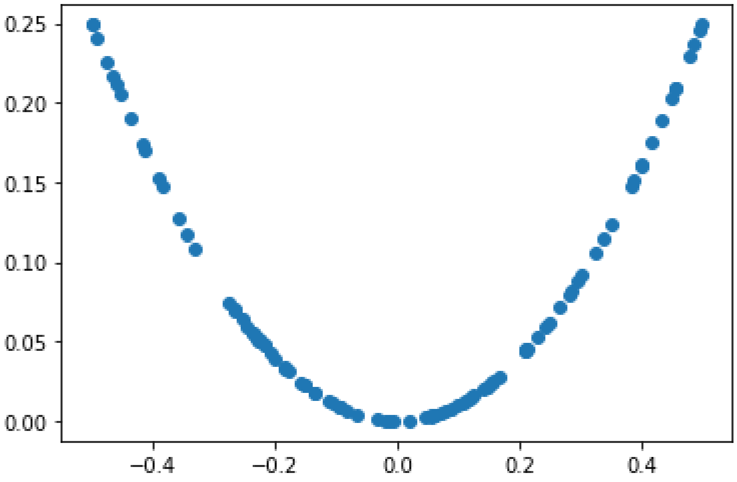
Question 4 and 6

```
from numpy.random import rand
from numpy.random import randn

from numpy import hstack
from numpy import zeros
from numpy import ones

from matplotlib import pyplot

from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
```

Data-Set

```
# generate real randoms sample from x^2
def generate_real_samples(n=100):
  # generate random inputs in [-0.5, 0.5]
  X1 = rand(n) - 0.5
  # generate outputs X^2 (quadratic)
  X2 = X1 * X1
  # stack arrays
  X1 = X1.reshape(n, 1)
  X2 = X2.reshape(n, 1)
  X = hstack((X1, X2))
  # generate class labels
  y = ones((n, 1))
  return X, y


# generate samples
data , y = generate_real_samples()
# plot samples
pyplot.scatter(data[:, 0], data[:, 1])
pyplot.show()
```



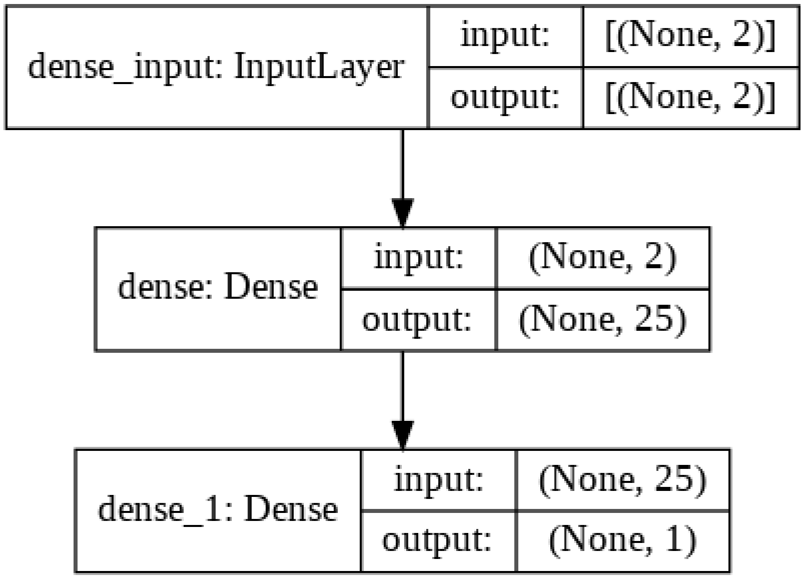
A simple dataset has been created in the form of x^2 and has been given a class value true

Discriminator

```
# define the standalone discriminator model
def define_discriminator(n_inputs=2):
  model = Sequential()
  model.add(Dense(25, activation='relu', kernel_initializer='he_uniform', input_dim=n_inputs))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```

```
# define the discriminator model
model = define_discriminator()
# summarize the model
model.summary()
# plot the model
plot_model(model, to_file='discriminator_plot.png', show_shapes=True, show_layer_names=True)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 25)                75        
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 26        
=================================================================
Total params: 101
Trainable params: 101
Non-trainable params: 0
_________________________________________________________________
```



A simple MLP-3 model has been made for the discriminator with 2 inputs, a relu activation in the hidden layer (25 neurons) and sigmoid activation in the output layer.

Training the discriminator with fake samples created by us

```
# generate n fake samples with class labels
def generate_fake_samples(n):
  # generate inputs in [-1, 1]
  X1 = -1 + rand(n) * 2
  # generate outputs in [-1, 1]
  X2 = -1 + rand(n) * 2
  # stack arrays
  X1 = X1.reshape(n, 1)
  X2 = X2.reshape(n, 1)
  X = hstack((X1, X2))
  # generate class labels
  y = zeros((n, 1))
  return X, y
```

Fake samples are created that closely resemble the dataset and are given the class value false

```
# train the discriminator model
def train_discriminator(model, n_epochs=1000, n_batch=128):
  half_batch = int(n_batch / 2)
  # run epochs manually
  for i in range(n_epochs):
    # generate real examples
    X_real, y_real = generate_real_samples(half_batch)
    # update model
    model.train_on_batch(X_real, y_real)
    # generate fake examples
    X_fake, y_fake = generate_fake_samples(half_batch)
    # update model
    model.train_on_batch(X_fake, y_fake)
    # evaluate the model
    _, acc_real = model.evaluate(X_real, y_real, verbose=0)
    _, acc_fake = model.evaluate(X_fake, y_fake, verbose=0)
    print(i, acc_real, acc_fake)

# define the discriminator model
model = define_discriminator()
# fit the model
train_discriminator(model)
```

```
941 1.0 0.859375
942 1.0 0.828125
943 1.0 0.875
944 1.0 0.90625
945 1.0 0.90625
946 1.0 0.890625
947 1.0 0.875
948 1.0 0.8125
949 1.0 0.875
950 1.0 0.84375
951 1.0 0.875
952 1.0 0.796875
953 1.0 0.9375
954 1.0 0.96875
955 1.0 0.921875
956 1.0 0.84375
957 1.0 0.890625
958 1.0 0.84375
959 1.0 0.90625
960 1.0 0.90625
961 1.0 0.875
962 1.0 0.890625
963 1.0 0.859375
964 1.0 0.859375
965 1.0 0.875
966 1.0 0.84375
```

```
967 1.0 0.890625
968 1.0 0.859375
969 1.0 0.90625
970 1.0 0.875
971 1.0 0.90625
972 1.0 0.875
973 1.0 0.90625
974 1.0 0.84375
975 1.0 0.859375
976 1.0 0.84375
977 1.0 0.859375
978 1.0 0.828125
979 1.0 0.921875
980 1.0 0.84375
981 1.0 0.828125
982 1.0 0.90625
983 1.0 0.828125
984 1.0 0.875
985 1.0 0.796875
986 1.0 0.828125
987 1.0 0.890625
988 1.0 0.859375
989 1.0 0.921875
990 1.0 0.84375
991 1.0 0.921875
992 1.0 0.8125

993 1.0 0.890625
994 1.0 0.84375
995 1.0 0.84375
996 1.0 0.953125
997 1.0 0.90625
998 1.0 0.9375
999 1.0 0.890625
```

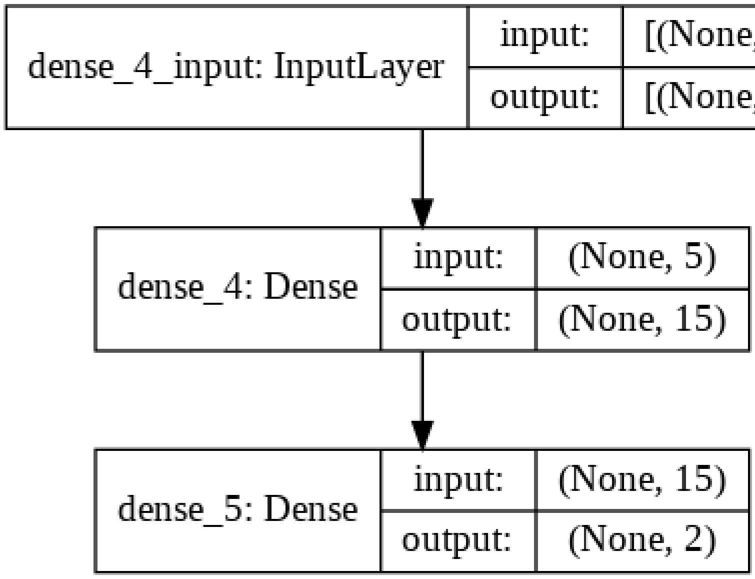
From the training it can be seen that the model is 100% accurate with the dataset and can to a great extent guess when a fake sample is given to the model

Generator

```
# define the standalone generator model
def define_generator(latent_dim, n_outputs=2):
  model = Sequential()
  model.add(Dense(15, activation='relu', kernel_initializer='he_uniform', input_dim=latent_di
  model.add(Dense(n_outputs, activation='linear'))
  return model


# define the generator model
model = define_generator(5)
# summarize the model
```

```
model.summary()
# plot the model
plot_model(model, to_file='generator_plot.png', show_shapes=True, show_layer_names=True)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 15)                90        
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 32        
=================================================================
Total params: 122
Trainable params: 122
Non-trainable params: 0
_________________________________________________________________
```

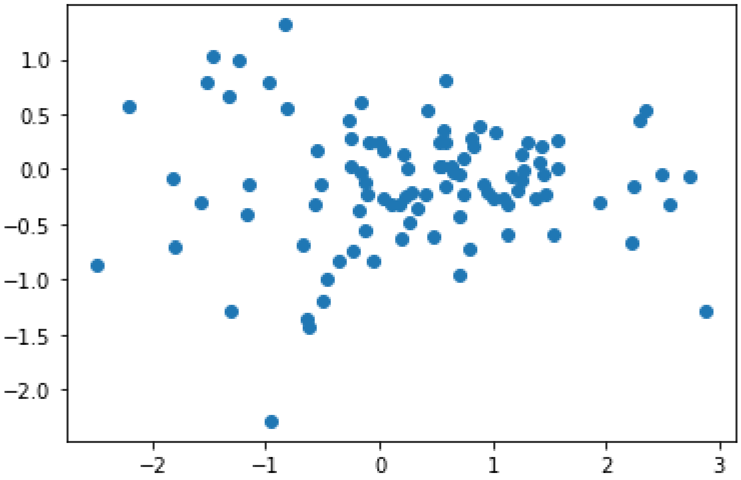


A simple MLP-3 model has been made for the generator with 5 inputs, a relu activation in the hidden layer (15 neurons) and no non linearity in the output layer.

generator used to generate data

```
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n):
  # generate points in the latent space
  x_input = randn(latent_dim * n)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n, latent_dim)
  return x_input
```

```
# use the generator to generate n fake examples and plot the results
def generate_fake_samples(generator, latent_dim, n):
  # generate points in latent space
  x_input = generate_latent_points(latent_dim, n)
  # predict outputs
  X = generator.predict(x_input)
  # plot the results
  pyplot.scatter(X[:, 0], X[:, 1])
  pyplot.show()

# size of the latent space
latent_dim = 5
# define the discriminator model
model = define_generator(latent_dim)
# generate and plot generated samples
generate_fake_samples(model, latent_dim, 100)
```



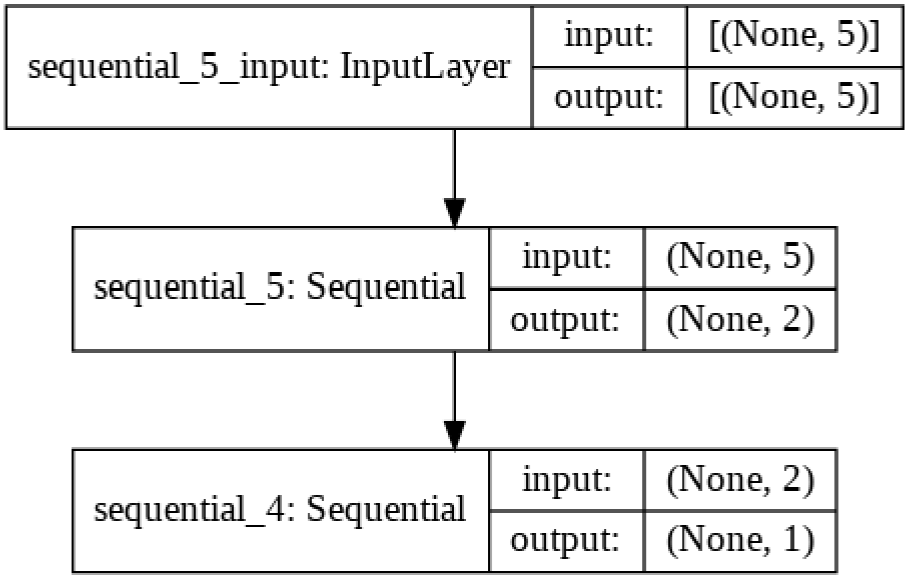
The generator will be trained with the help of the discriminator in the GAN which will be shown below.

GAN Model

```
def define_gan(generator, discriminator):
  # make weights in the discriminator not trainable
  discriminator.trainable = False
  # connect them
  model = Sequential()
  # add generator
  model.add(generator)
  # add the discriminator
  model.add(discriminator)
  # compile model
  model.compile(loss='binary_crossentropy', optimizer='adam')
  return model

# size of the latent space
```

```
latent_dim = 5
# create the discriminator
discriminator = define_discriminator()
# create the generator
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, discriminator)
# summarize gan model
gan_model.summary()
# plot gan model
plot_model(gan_model, to_file='gan_plot.png', show_shapes=True, show_layer_names=True)
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_5 (Sequential)    (None, 2)                 122
_________________________________________________________________
sequential_4 (Sequential)    (None, 1)                 101
=================================================================
Total params: 223
Trainable params: 122
Non-trainable params: 101
_________________________________________________________________
```



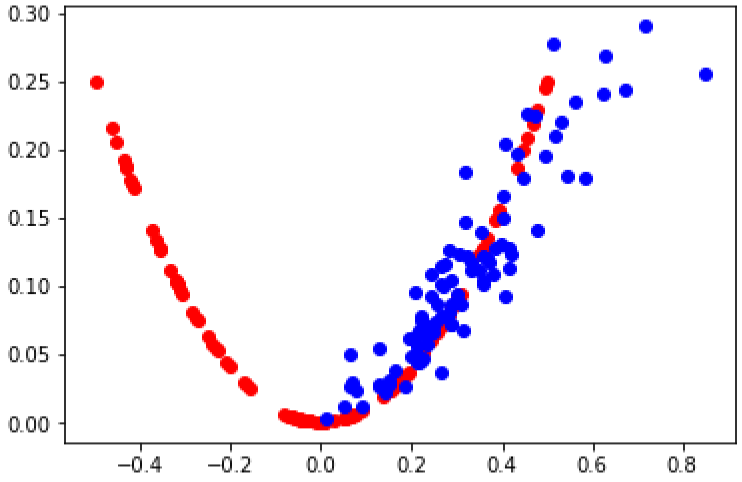
The GAN model is made by combining the generator and the discriminator from above

Training the evaluating the GAN Model

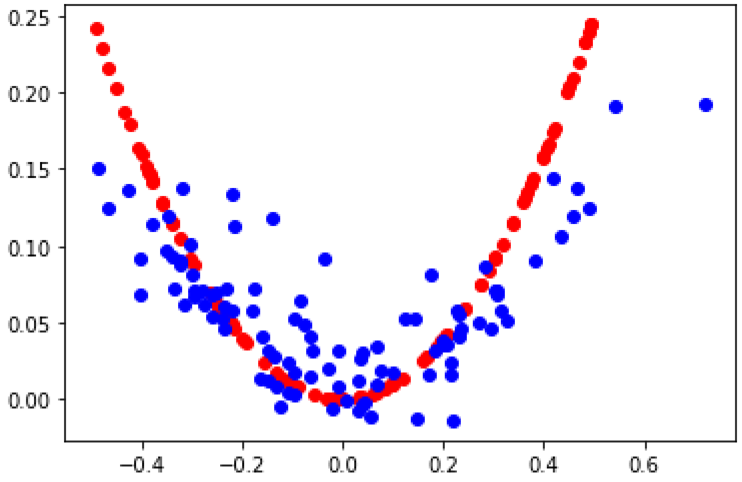
```
# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n):
  # generate points in latent space
  x_input = generate_latent_points(latent_dim, n)
  # predict outputs
  # predict outputs
  X = generator.predict(x_input)
  # create class labels
  y = zeros((n, 1))
  return X, y


# evaluate the discriminator and plot real and fake points
def summarize_performance(epoch, generator, discriminator, latent_dim, n=100):
  # prepare real samples
  x_real, y_real = generate_real_samples(n)
  # evaluate discriminator on real examples
  _, acc_real = discriminator.evaluate(x_real, y_real, verbose=0)
  # prepare fake examples
  x_fake, y_fake = generate_fake_samples(generator, latent_dim, n)
  # evaluate discriminator on fake examples
  _, acc_fake = discriminator.evaluate(x_fake, y_fake, verbose=0)
  # summarize discriminator performance
  print('epoch',epoch,'real accuracy: ', acc_real,' fake accuracy: ', acc_fake)
  # scatter plot real and fake data points
  pyplot.scatter(x_real[:, 0], x_real[:, 1], color='red')
  pyplot.scatter(x_fake[:, 0], x_fake[:, 1], color='blue')
  pyplot.show()


# train the generator and discriminator
def train(g_model, d_model, gan_model, latent_dim, n_epochs=10000, n_batch=128, n_eval=2000):
  # determine half the size of one batch, for updating the discriminator
  half_batch = int(n_batch / 2)
  # manually enumerate epochs
  for i in range(n_epochs):
    # prepare real samples
    x_real, y_real = generate_real_samples(half_batch)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
    # update discriminator
    d_model.train_on_batch(x_real, y_real)
    d_model.train_on_batch(x_fake, y_fake)
    # prepare points in latent space as input for the generator
    x_gan = generate_latent_points(latent_dim, n_batch)
    # create inverted labels for the fake samples
    y_gan = ones((n_batch, 1))
    # update the generator via the discriminator's error
    gan_model.train_on_batch(x_gan, y_gan)
    # evaluate the model every n_eval epochs
    if (i+1) % n_eval == 0:
      summarize_performance(i, g_model, d_model, latent_dim)


# size of the latent space
latent_dim = 5
# create the discriminator
discriminator = define_discriminator()
```

```
# create the generator
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, discriminator)
# train model
train(generator, discriminator, gan_model, latent_dim)
```

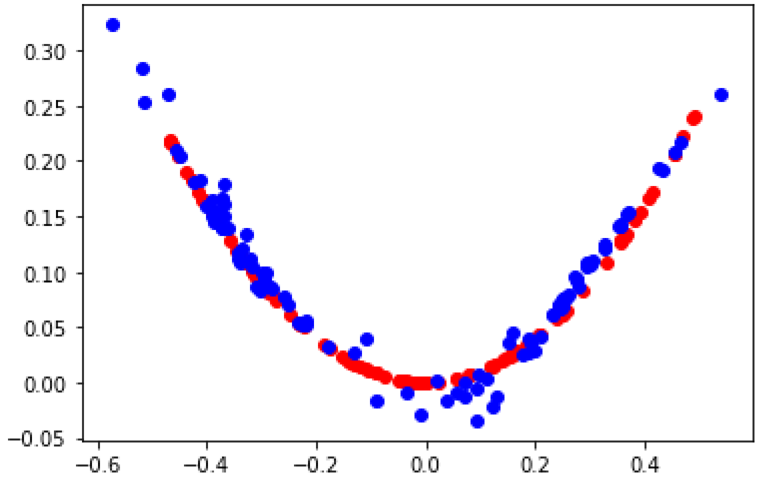
```
epoch 1999 real accuracy:  0.4699999988079071  fake accuracy:  0.8199999928474426
```



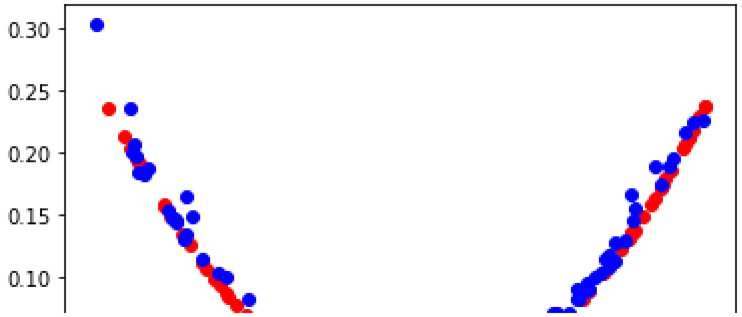
```
epoch 3999 real accuracy:  0.4300000071525574  fake accuracy:  0.6600000262260437
```

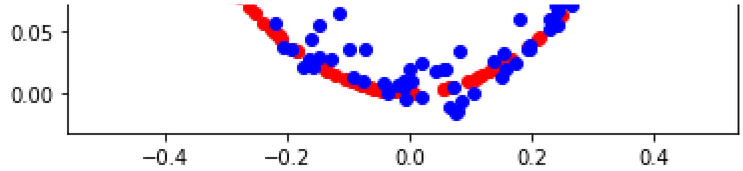


```
epoch 5999 real accuracy:  0.4099999964237213  fake accuracy:  0.47999998927116394
```

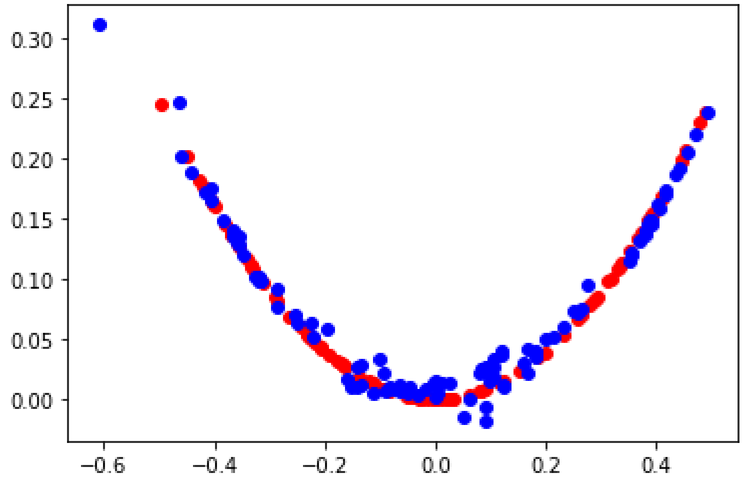


```
epoch 7999 real accuracy:  0.6600000262260437  fake accuracy:  0.4000000059604645
```





```
epoch 9999 real accuracy:  0.5600000023841858  fake accuracy:  0.3700000047683716
```



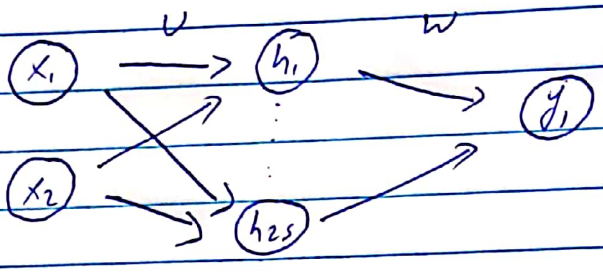
from the results of training we can see-

1. The plot shows us the the output of the generator plotted along side the dataset. We can see that in the start of training the generator was produces data that looked nothing like the dataset. As training proceeded we can see the incrmental improvements the generator made in producing data that behaved more like our dataset.
2. The accuracy of the discriminator is also printed above the plots with the respective epoch. We cen see at the start the discriminator is fairly able to categorize the fake data generated by the generator correctly. But as training proceeds and the generator is tuned better we can see a dip in accuracy of the discriminator being able categorize the fake data even though the accuracy of the real dataset increases.

✓ 9m 10s completed at 8:46 PM ● ×

5. GAN Model.

→ Phase - I - Discriminator.



Input layer.	hidden layer.	Output layer.
2 Neuron	25 Neuron	1 Neuron.

→ $X \in R^2$ $t \in Z^1$

→ $U \in R^{25 \times 2}$, $b \in R^{25}$, $W \in R^{1 \times 25}$, $C \in R^1$

→ Equation.

$g = UX + b.$
$h = \phi^1(g)$ $\phi^1()$ → RelU F'n.
$z = Wh + c$
$y = \phi^2(z)$ $\phi^2()$ → Sigmoid.

→ CFG.

$$\begin{tikzcd}
U \arrow[dr] & & W \arrow[dr] & & t \arrow[dr] & \\
X \arrow[r] & g \arrow[r, "\phi^1()"] & h \arrow[r] & z \arrow[r, "\phi^2()"] & y \arrow[r] & L \\
b \arrow[ur] & & c \arrow[ur] & & &
\end{tikzcd}$$

$$L_{CE} = \frac{1}{N}\sum_{1}^{N}\left[t\log(1+\exp(-z)) + (1-t)\log(1+\exp(z))\right].$$

-) Back Propagation

$$\bar{L} = \frac{dL}{dL} = 1.$$

$$\bar{z} = \bar{L}\cdot\frac{dL}{dz} = 1\cdot\frac{1}{N}(y-t)$$

$$\bar{w} = \bar{z}\cdot\frac{dz}{dw} = \frac{1}{N}(y-t)\cdot h.$$

$$\bar{c} = \bar{z}\cdot\frac{dz}{dc} = \frac{1}{N}(y-t)\cdot 1$$

$$\bar{h} = \bar{z}\cdot\frac{dz}{dh} = \frac{1}{N}(y-t)\cdot w.$$

$$\bar{g} = \bar{h}\,\frac{dh}{dg} = \frac{1}{N}(y-t)\cdot w\cdot\phi''(g)$$

~~$\phi''(g) = 1$ $g \geq 0$~~
~~0 oth~~
~~wise.~~

$$\phi''(y) = \begin{cases} 1 & y \geq 0. \\ 0 & \text{otherwise.} \end{cases}$$

$$\bar{v} = \bar{g}\cdot\frac{dy}{du} = \frac{1}{N}(y-t)\cdot w\cdot\phi''(y)\cdot x$$

$$\bar{b} = \bar{g}\cdot\frac{dg}{db} = \frac{1}{N}(y-t)\cdot w\cdot\phi'(g)$$

-) note -> $t = 1$ if input from dataset.
$t = 0$ if input from generator.

-) Weight update.

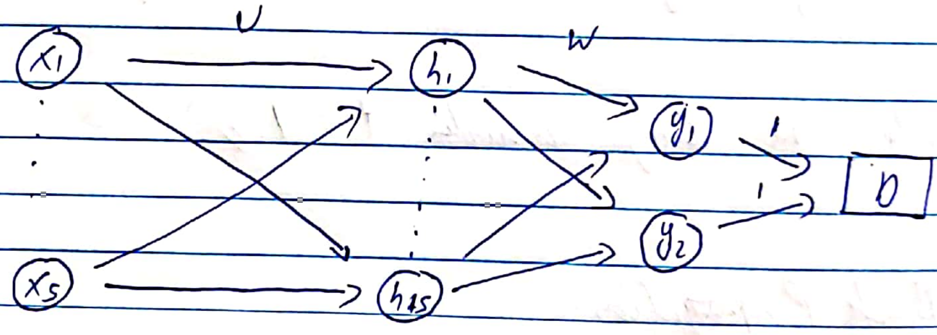
$U' = U - \alpha \bar{U}$

$b' = b - \alpha \bar{b}$.

$W' = W - \alpha \bar{W}$.

$C' = C - \alpha \bar{C}$.

-) Phase - 2 - Generator.



Input layer.	hidden layer.	Output layer.	Discriminator.
5 Neuron	15 Neurons	2-Neuron	

-) $X \in R^5$, $y \in R^2$

-) $U \in R^{15 \times 5}$, $b \in R^{15}$, $W \in R^{2 \times 15}$, $C \in R^2$

ⓧ -) The outputs of the generator are fed into the discriminator with no weight.

→ ~~CFG~~

→ Equation :

$g = Ux + b.$

$h = \phi(g)$ $\qquad\qquad \phi() \rightarrow$ ReLU F'n.

$y = Wh + c.$

→ CFG.

$$\begin{tikzcd}
U \arrow[dr] & & W \arrow[dr] & & t \arrow[dr] & \\
X \arrow[r] & g \arrow[r, "\phi()"] & h \arrow[r] & y \arrow[r, "1"] & \boxed{D} \arrow[r] & L. \\
b \arrow[ur] & & c \arrow[ur] & & &
\end{tikzcd}$$

$$L = \frac{1}{N}\sum_{1}^{N}\left[t \log\left(1 + \exp(-z_D)\right) + (1-t)\log\left(1 + \exp(z_D)\right)\right]$$

→ note → we always consider $t = 1$.

→ Back Propagation.

→ Considering the result we got in phase-1 Back - Propagation.

→ $\bar{X}_D = \bar{g}_D \cdot \frac{dg_D}{dx_D} = \bar{g}_D \cdot U_D$ (from discriminator equations).

→ $\bar{y} = \bar{X}_D.$

$\bar{h} = \bar{y} \cdot \frac{dy}{dh} = \bar{y} \cdot W.$

$$\bar{W} = \bar{y} \cdot \frac{dy}{dW} = \bar{y} \cdot h.$$

$$\bar{C} = \bar{y} \cdot \frac{dy}{dc} = \bar{y}$$

$$\bar{g} = \bar{h} \cdot \frac{dh}{dy} = \bar{h} \cdot \phi'(g).$$

$$\phi'(y) = \begin{cases} 1 & y \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

$$\bar{U} = \bar{g} \cdot \frac{dg}{du} = \bar{g} \cdot X.$$

$$\bar{b} = \bar{g} \, \frac{dg}{db} = \bar{g}$$

→ Weight Update

$W' = W - \alpha \bar{W}$

$C' = C - \alpha \bar{C}$

$U' = U - \alpha \bar{C}$

$b' = b \cdot \alpha \bar{b}.$

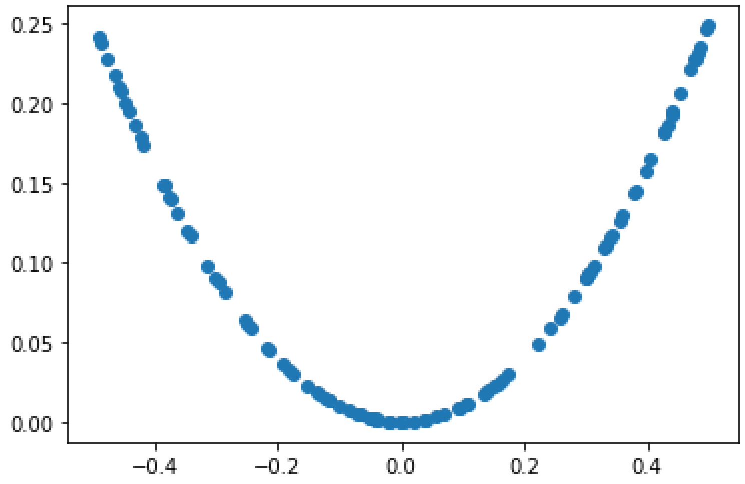
Question 7

```
from numpy.random import rand
from numpy.random import randn

import numpy as np

from numpy import hstack
from numpy import zeros
from numpy import ones

from matplotlib import pyplot
from sklearn.metrics import mean_squared_error
from scipy.special import expit as sigmoid

from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
from keras import backend as k
from keras import losses


def loss_stable(z,t,N):
  return (1./N) * np.sum(t*np.logaddexp(0,-z) + (1-t)*np.logaddexp(0,z))


def drelu(z):
  z[z<=0] = 0
  z[z>0] = 1
  return z
```

Dataset

```
# generate real randoms sample from x^2
def generate_real_samples(n=100):
  # generate random inputs in [-0.5, 0.5]
  X1 = rand(n) - 0.5
  # generate outputs X^2 (quadratic)
  X2 = X1 * X1
  # stack arrays
  X1 = X1.reshape(n, 1)
  X2 = X2.reshape(n, 1)
  X = hstack((X1, X2))
  # generate class labels
  y = ones(n)
  return X
```

```
    return x, y


# generate samples
data , y = generate_real_samples()
# plot samples
pyplot.scatter(data[:, 0], data[:, 1])
pyplot.show()
print(data.shape)
```



```
(100, 2)
```

Discriminator model

```
def forward_dis(X,U,b,W,c):
    G = np.dot(X, U.T)  + b
    H = G* (G>0)
    z = np.dot(H,W.T) + c
    y = sigmoid(z)

    return y,z,H,G
```

Generator model

```
def forward_gen(X,U,b,W,c):
    G = np.dot(X, U.T)  + b
    H = G* (G>0)
    y = np.dot(H,W.T) + c

    return y,H,G
```

Discriminator Backpropagation

```
def grad_decent_dis(x,t,U,b,W,c):
  N=x.shape[0]
  #U = np.random.randn(25,2)
  #b = np.zeros(25)
  #W = np.random.randn(25)
  #c = 0
  #num_steps = 50000
  alpha = 0.01
  #thresh=0.02
  #for step in range(num_steps):
  y,z,H,G = forward_dis(x,U,b,W,c)
  l= loss_stable(z,t,N)
  #if (l<thresh):
    #print('converged at step: ',step)
    #break
    #if (step % 1000==0):
      #print (step,' loss = ',l)

  E_bar = 1
  z_bar = (1./N) * (y - t)
  #y_bar = (1./N) * (y.T - t)
  #z_bar = y_bar * (y.T*(1-y.T))
  W_bar = np.dot(H.T,z_bar)
  c_bar = np.dot(z_bar, np.ones(N))
  H_bar = np.outer(z_bar, W.T )
  G_bar = H_bar * drelu(G)
  U_bar = np.dot(G_bar.T, x)
  b_bar = np.dot(G_bar.T , np.ones(N))

  U -= alpha * U_bar
  b -= alpha * b_bar
  W -= alpha * W_bar
  c -= alpha * c_bar

  return U,b,W,c,l
```

Generator data generation

```
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n):
  # generate points in the latent space
  x_input = randn(latent_dim * n)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n, latent_dim)
  x_input = x_input.reshape(n, latent_dim)
  return x_input


def generate_fake_samples_gen(latent_dim, n, U,b,W,c):
  # generate points in latent space
  x_input = generate_latent_points(latent_dim, n)
  # predict outputs
  X,H,G = forward_gen(x_input,U,b,W,c)
  # create class labels
  y = zeros(n)
  return X, y, H,G,x_input
```

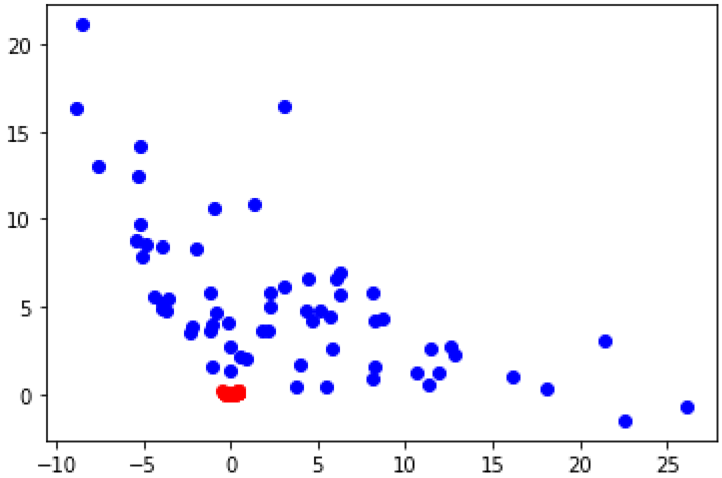
Generator Backpropagation

```
def grad_decent_gen(t,U,b,W,c,Ug,bg,Wg,cg,latent_dim,n):
  x,_,Hg,Gg, latent_x= generate_fake_samples_gen(latent_dim, n, Ug,bg,Wg,cg)
  N=x.shape[0]
  #U = np.random.randn(25,2)
  #b = np.zeros(25)
  #W = np.random.randn(25)
  #c = 0
  #num_steps = 50000
  alpha = 0.001
  #thresh=0.02
  #for step in range(num_steps):
  y,z,H,G = forward_dis(x,U,b,W,c)
  l= loss_stable(z,t,N)
  #if (l<thresh):
    #print('converged at step: ',step)
    #break
    #if (step % 1000==0):
      #print (step,' loss = ',l)

  z_bar = (1./N) * (y - t)
  H_bar = np.outer(z_bar, W.T )
  G_bar = H_bar * drelu(G)
  x_bar = np.dot(G_bar,U)
  z_bar = x_bar
  W_bar = np.dot(z_bar.T,Hg)
  c_bar = np.dot(np.ones(N),z_bar)
  H_bar = np.dot(z_bar, Wg )
  G_bar = H_bar * drelu(Gg)
  U_bar = np.dot(G_bar.T, latent_x)
  b_bar = np.dot(G_bar.T , np.ones(N))

  Ug -= alpha * U_bar
  bg -= alpha * b_bar
  Wg -= alpha * W_bar
  cg -= alpha * c_bar

  return Ug,bg,Wg,cg,l
```

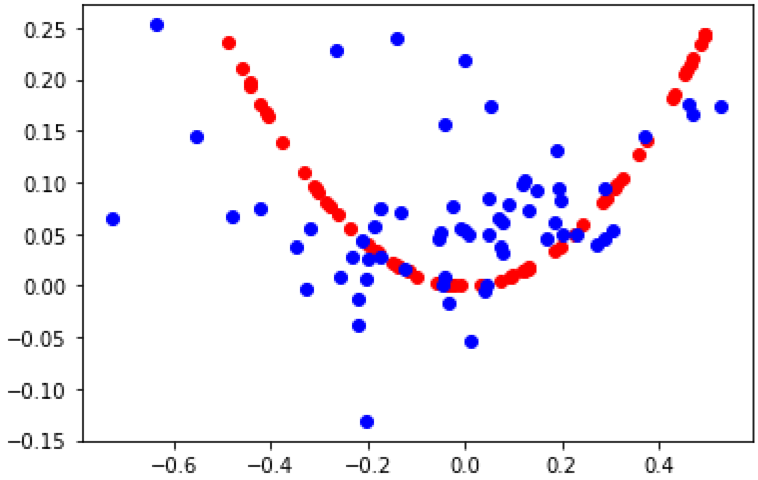
Final GAN model and training

```
n_epochs=20001
n_batch=128
l=0
latent_dim = 5

U = np.random.randn(25,2)
b = np.zeros(25)
W = np.random.randn(25)
c = 0

Ug = np.random.randn(15,5)
bg = np.zeros(15)
Wg = np.random.randn(2,15)
cg = np.zeros(2)


for i in range(n_epochs):
  # generate real examples
  X_real, y_real = generate_real_samples(half_batch)
  # update discriminator model
  U,b,W,c,l=grad_decent_dis(X_real,y_real,U,b,W,c)
  if (i%4000==0):
    print('Discriminator loss real',i,':',l)
  # generate fake examples
  X_fake, y_fake,Hg,Gg,latent_x = generate_fake_samples_gen(latent_dim,half_batch,Ug,bg,Wg,cg
  # update discriminator model
  U,b,W,c,l=grad_decent_dis(X_fake,y_fake,U,b,W,c)
  if (i%4000==0):
    print('Discriminator loss fake',i,':',l)
  #train the generator model
  Ug,bg,Wg,cg,l = grad_decent_gen(np.ones(n_batch),U,b,W,c,Ug,bg,Wg,cg,latent_dim,n_batch)
  if (i%4000==0):
    print('Generator loss',i,':',l)
    pyplot.scatter(X_real[:, 0], X_real[:, 1], color='red')
    pyplot.scatter(X_fake[:, 0], X_fake[:, 1], color='blue')
    pyplot.show()
```

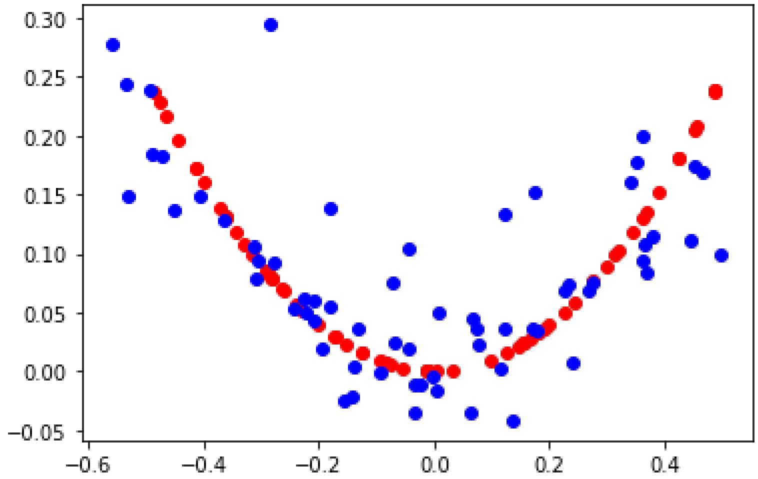
```
Discriminator loss real 0 : 0.4747555792634539
Discriminator loss fake 0 : 33.52022145779241
Generator loss 0 : 0.0009854603279658707
```



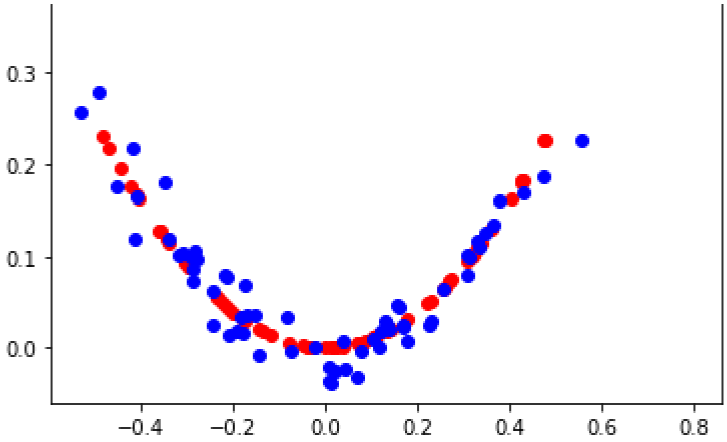
```
Discriminator loss real 4000 : 0.6742679308560113
Discriminator loss fake 4000 : 0.6622314971280567
Generator loss 4000 : 0.7711444227533917
```



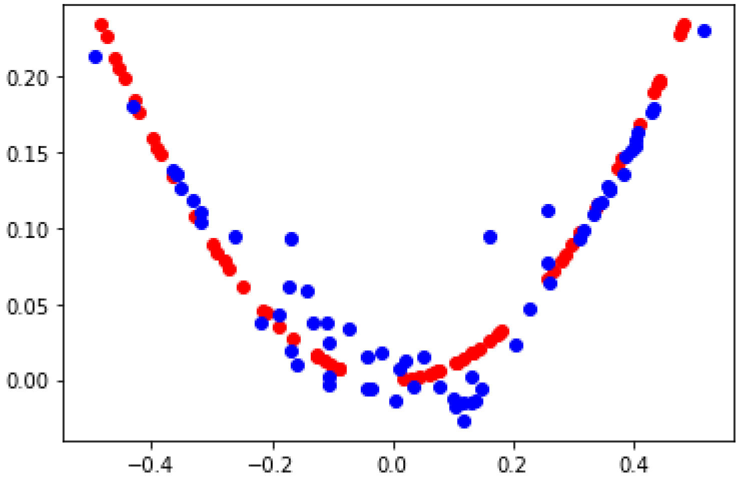
```
Discriminator loss real 8000 : 0.6719639277806612
Discriminator loss fake 8000 : 0.6901468099080383
Generator loss 8000 : 0.7899212772449655
```



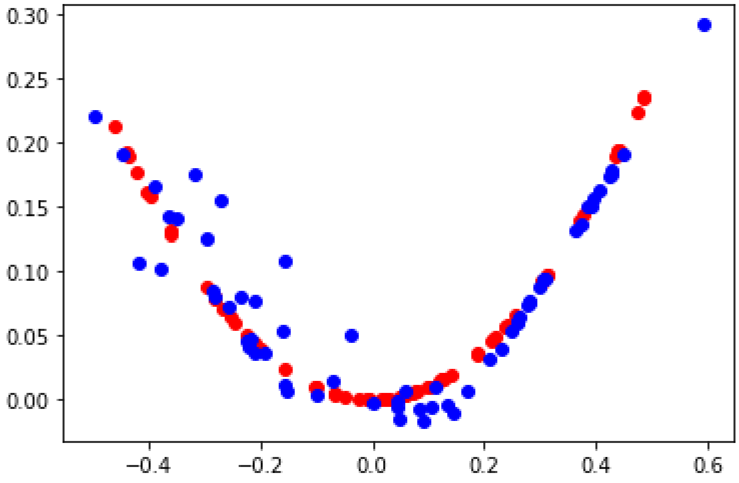
```
Discriminator loss real 12000 : 0.7074233669054396
Discriminator loss fake 12000 : 0.7004637099317828
Generator loss 12000 : 0.7304616141340543
```



```
Discriminator loss real 16000 : 0.7022284859361609
Discriminator loss fake 16000 : 0.7034814613068479
Generator loss 16000 : 0.7128372253233979
```



```
Discriminator loss real 20000 : 0.7072081649916047
Discriminator loss fake 20000 : 0.6987434467645222
Generator loss 20000 : 0.7161312557414106
```



We can see as we train the generator gets more competent in producing data that resembles the dataset

● ✕