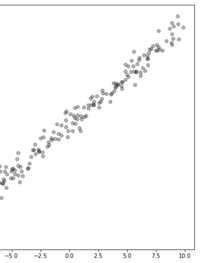
Visualization of Gradient Descent algorithm based on Linear Regression problem

1. Input points

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

path = '/content/drive/My Drive/Colab Notebooks/assignment 3/data.csv'
data = np.genfromtxt(path, delimiter=',')

x_data = np.array(data[:, 0], dtype=np.float32)
y_data = np.array(data[:, 1], dtype=np.float32)
x_data = x_data.reshape(-1,1)
y_data = y_data.reshape(-1,1)

plt.figure(figsize=(8, 8))
plt.scatter(x_data, y_data, c='black', alpha=0.3)
plt.show()
```

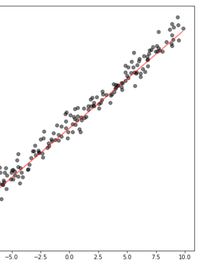


1. Linear regression reslut

In [3]:
```
# define model architecture.
import torch
from torch.autograd import Variable

class linearRegression(torch.nn.Module):
    def __init__(self, inputSize, outputSize):
        super(linearRegression, self).__init__()
        self.linear = torch.nn.Linear(inputSize, outputSize)

    def forward(self, x):
        out = self.linear(x)
        return out

# instantiate the model
inputDim = 1    # takes variable 'x'
outputDim = 1   # takes variable 'y'
learningRate = 0.01
epoches = 100

model = linearRegression(inputDim, outputDim)
#### for GPU #####
if torch.cuda.is_available():
    model.cuda()

# initialize the loss(Mean Squared Error) and optimization(stochastic Gradient Descent)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learningRate)

# begin to train model
for epoch in range(epoches):
    # Converting inputs and labels to Variable
    if torch.cuda.is_available():
        inputs = Variable(torch.from_numpy(x_data).cuda())
        labels = Variable(torch.from_numpy(y_data).cuda())
    else:
        inputs = Variable(torch.from_numpy(x_data))
        labels = Variable(torch.from_numpy(y_data))

    # Clear gradient buffers because we don't want any gradient from previous eopch to carry
 forward, don't want to cummulate gradients
    optimizer.zero_grad()

    # get output from the model, given the inputs
    outputs = model(inputs)

    # get loss for the predicted output
    loss = criterion(outputs, labels)
    #print(loss)
    # get gradients w.r,t to parameters
    loss.backward()

    # update parameters
    optimizer.step()

    #print('eopch {}, loss {}'.format(epoch, loss.item()))

# model is trained, test it
with torch.no_grad(): # don't need gradients in the testing phase
    if torch.cuda.is_available():
        predicted = model(Variable(torch.from_numpy(x_data).cuda())).cpu().data.numpy()
    else:
        predicted = model(Variable(torch.from_numpy(x_data))).data.numpy()
    #print(predicted)

plt.clf()
plt.figure(figsize=(8, 8))
plt.plot(x_data, y_data, 'ko', label='True data', alpha=0.5)
plt.plot(x_data, predicted, 'r-', label='Predictions', alpha=0.5)
plt.legend(loc='best')
plt.show
```

Out[3]:
```
<function matplotlib.pyplot.show>
```

```
<Figure size 432x288 with 0 Axes>
```



1. Plot the energy surface

In [0]:
```
for i in loss:
    print('p1 = ', i[0], ', p2 = ', i[1], ' ', i[2], ' ', i[3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-af51f5e46af0> in <module>()
----> 1 for i in loss:
      2     print('p1 = ', i[0], ', p2 = ', i[1], ' ', i[2], ' ', i[3])

/usr/local/lib/python3.6/dist-packages/torch/tensor.py in __iter__(self)
    460         # map will interleave them.)
    461         if self.dim() == 0:
--> 462             raise TypeError('iteration over a 0-d tensor')
    463         if torch._C._get_tracing_state():
    464             warnings.warn('Iterating over a tensor might cause the trace to be incorr
ect.'

TypeError: iteration over a 0-d tensor
```

In [0]:
```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np

fig = plt.figure(figsize=(8,8))
ax = fig.gca(projection='3d')
X = np.arange(-30,30,0.5)
Y = np.arange(-30,30,0.5)
Z = loss.item()/(2*x_data.size)

surf = ax.plot_surface(X,Y,Z, cmap=cm.coolwarm,linewidth=0,antialiased=False)
#X, Y, Z = axes3d.get_test_data(0.5)
#ax.plot_surface(X, Y, Z, rstride = 8, cstride=8, alpha = 0.9)
#cset = ax.contour(X, Y, Z, zdir='z', offset = -100, cmap=cm.coolwarm)
#cset = ax.contour(X, Y, Z, zdir='x', offset = -30, cmap=cm.coolwarm)
#cset = ax.contour(X, Y, Z, zdir='y', offset = 30, cmap=cm.coolwarm)

ax.set_xlabel('X')
ax.set_xlim(-30,30)
ax.set_ylabel('Y')
ax.set_ylim(-30,30)
ax.set_zlabel('Z')
ax.set_zlim(-100,100)

plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-33-89c0b2b5ea7d> in <module>()
     10 Z = loss.item()/(2*x_data.size)
     11
---> 12 surf = ax.plot_surface(X,Y,Z, cmap=cm.coolwarm,linewidth=0,antialiased=False)
     13 #X, Y, Z = axes3d.get_test_data(0.5)
     14 #ax.plot_surface(X, Y, Z, rstride = 8, cstride=8, alpha = 0.9)

/usr/local/lib/python3.6/dist-packages/mpl_toolkits/mplot3d/axes3d.py in plot_surface(self,
 X, Y, Z, norm, vmin, vmax, lightsource, *args, **kwargs)
   1494             had_data = self.has_data()
   1495
-> 1496             if Z.ndim != 2:
   1497                 raise ValueError("Argument Z must be 2-dimensional.")
   1498             if np.any(np.isnan(Z)):

AttributeError: 'float' object has no attribute 'ndim'
```