Q1

a

mean = a * mu + b
var = a * sigma^2

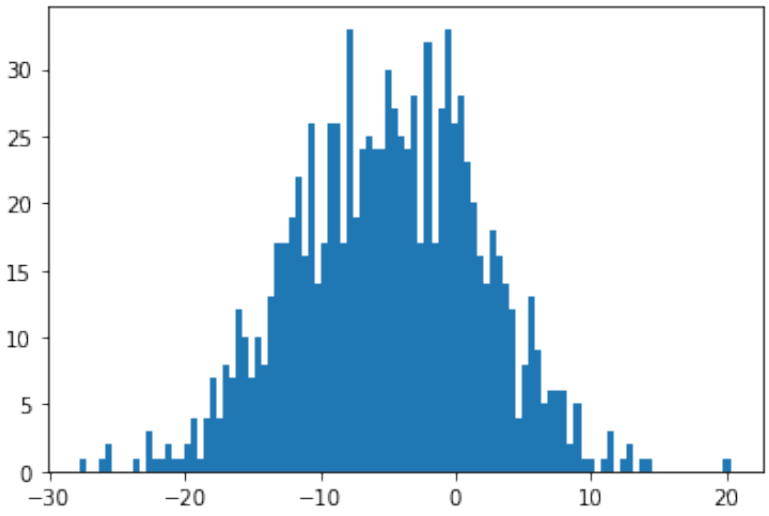
b

a = 1 / sigma^2
b = - 1/sigma^2 * u

c

In [28]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
import math


r = np.random.normal(-5, 7, 1000)
plt.hist(r, bins=100)
plt.show()
```



Q2

see other pdf

Q3

a

need 3 params

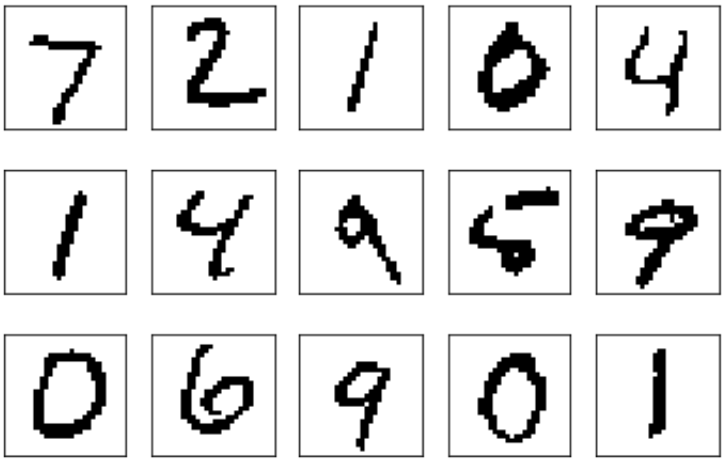
In [11]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_da

## function to plot images in grid
def show_images(images, rows, cols):
    for i in range(rows * cols):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(images[i], cmap=plt.cm.gray_r)
        plt.xticks(())
        plt.yticks(())
    plt.show()

# convert to 0/1 (instead of 0-255)
x_train_int = [np.round(1.0*i/256) for i in x_train]
x_test_int = [np.round(1.0*i/256) for i in x_test]

## Uncomment below to see a few images
print('A few example images:')
show_images(x_test_int, 3, 5)
```

```
A few example images:
```



b

No. Because a stroke has has a width and a length, meaning that for an given pixel, xi, be to black, it is likely that its adjacant pixels are also black

c

In [81]:
```
def p(xi_index, xi_val, y):
    class_size = 0
    pixel_match_count = 0
    for i in range(len(x_train_int)):
        if y_train[i] == y:
            class_size += 1
            if x_train_int[i].flatten()[xi_index] == xi_val:
                pixel_match_count += 1
    prob_smoothed = (pixel_match_count + .7) / (class_size + .7)
    return math.log(prob_smoothed)
```

In [70]:
```
# def MAP_y(x):
#     p_x_given_each_y = np.zeros((10))
#     # partition x_train into subsets according to true label
#     x_partitions = [[], [], [], [], [], [], [], [], [], []]
#     for i in range(60000):
#             x_partitions[y_train[i]] += [i]
#     print("finished partitioning")
#     # iterate through each y
#     for y in range(10):
#         print("in class y =", y)
#         p_xi_given_y = np.zeros((len(x_partitions[y])))
#         # iterate through each pixel of image x
#         for i in range(784):
#             pixel_match_count = 0
#             expected = x.flatten()[i]
#             for index in x_partitions[y]:
#                 sample = x_train_int[index]
#                 if (sample.flatten())[i] == expected:
#                     pixel_match_count += 1
#             p_xi_given_y = (pixel_match_count + .7) / len(x_partitio
#         p_x_given_each_y[y] = np.sum(np.log(p_xi_given_y))
#     print(p_x_given_each_y)
#     return  np.argmax(p_x_given_each_y)
```

d

In [145]:
```
# iterate through p matrix
P_MX_1 = []
p_x_given_each_y = np.zeros((10))
# partition x_train into subsets according to true label
x_partitions = [[], [], [], [], [], [], [], [], [], []]
for i in range(60000):
            x_partitions[y_train[i]] += [i]
for y in range(10):
    P_MX_y = np.zeros((28, 28))
    for i in range(len(x_partitions[y])):
        P_MX_y = np.add(P_MX_y, x_train_int[x_partitions[y][i]])
    P_MX_y = np.add(P_MX_y, .6)
    P_MX_y = np.divide(P_MX_y, len(x_partitions[y])+.6)
    P_MX_y += np.log(P_MX_y)
    P_MX_1 += [P_MX_y]

P_MX_1 = np.array(P_MX_1)
print(P_MX.shape)
```

```
(10, 28, 28)
```

In [146]:
```
# iterate through p matrix
P_MX_0 = []
p_x_given_each_y = np.zeros((10))
# partition x_train into subsets according to true label
x_partitions = [[], [], [], [], [], [], [], [], [], []]
for i in range(60000):
            x_partitions[y_train[i]] += [i]
for y in range(10):
    P_MX_y = np.zeros((28, 28))
    P_MX_y = np.add(P_MX_y, len(x_partitions[y]))
    for i in range(len(x_partitions[y])):
        P_MX_y = np.subtract(P_MX_y, x_train_int[x_partitions[y][i]])
    P_MX_y = np.add(P_MX_y, .6)
    P_MX_y = np.divide(P_MX_y, len(x_partitions[y])+.6)
    P_MX_y += np.log(P_MX_y)
    P_MX_0 += [P_MX_y]

P_MX_0 = np.array(P_MX_0)
print(P_MX.shape)
```

```
(10, 28, 28)
```

In [147]:
```
def MAP(x):
    p = np.zeros((10))
    for y in range(10):
        p_x_given_y = 0
        for i in range(28):
            for j in range(28):
                if x[i][j] == 1:
                    p_x_given_y += P_MX_1[y, i, j]
                else:
                    p_x_given_y += P_MX_0[y, i, j]
        p[y] = p_x_given_y
    return np.argmax(p)
```

In [150]:
```
t_size = len(x_test_int)
err_count = 0
for i in range(t_size):
    y_head = MAP(x_test_int[i])
    y_label = y_test[i]
    if y_head != y_label:
        err_count += 1
risk = err_count / t_size
print("classification error rate = ", risk)
```

```
classification error rate =  0.1773
```