In [1]:
```
from sklearn import datasets
x, y = datasets.make_blobs(n_samples=150,n_features=2, centers=2,cluster_std=1.05,random_state=2)
```

In [2]:
```
import tensorflow as tf
```

In [3]:
```
from keras.datasets import mnist
```

In [4]:
```
(x_train, y_train),(x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11490434/11490434 [==============================] - 19s 2us/step
```

In [5]:
```
print("x_train shape", x_train.shape)
print("y_train shape", y_train.shape)
print("x_test shape", x_test.shape)
print("y_test shape", y_test.shape)
```

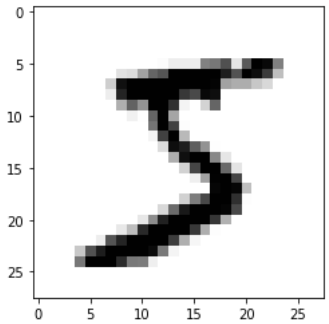
```
x_train shape (60000, 28, 28)
y_train shape (60000,)
x_test shape (10000, 28, 28)
y_test shape (10000,)
```

In [6]:
```
import numpy as np
```

In [7]:
```
import matplotlib.pyplot as plt
```

In [8]:
```
plt.imshow(x_train[0], cmap='binary')
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x176a5a4c670>
```



In [9]:
```
print(set(y_train))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [10]:
```
from keras.utils import to_categorical

y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)
```

In [11]:
```
print("y_train encoded shape ", y_train_encoded.shape)
print("y_test encoded shape ", y_test_encoded.shape)
```

```
y_train encoded shape  (60000, 10)
y_test encoded shape  (10000, 10)
```

In [12]:
```
# preprocessing

x_train_reshaped = np.reshape(x_train, (60000, 784))
x_test_reshaped = np.reshape(x_test, (10000, 784))
```

In [13]:
```
print(x_train_reshaped.shape)
print()
```

```
(60000, 784)
```

In [14]:
```
# display pixel value
print(set(x_train_reshaped[0]))
```

```
{0, 1, 2, 3, 9, 11, 14, 16, 18, 23, 24, 25, 26, 27, 30, 35, 36, 39, 43, 45, 46, 49, 55, 56, 64, 66, 70, 78, 80, 81, 82, 90, 93, 94, 107, 108, 114, 119, 126, 127, 130, 132, 133, 135, 136, 139, 148, 150, 154, 156, 160, 166, 170, 171, 172, 175, 182, 183, 186, 187, 190, 195, 198, 201, 205, 207, 212, 213, 219, 221, 225, 226, 229, 238, 240, 241, 242, 244, 247, 249, 250, 251, 252, 253, 255}
```

In [15]:
```
# Data normalization
import math
x_mean = np.mean(x_train_reshaped)
x_std = np.std(x_train_reshaped)
epsilon = pow(math.e,-10)
```

In [21]:
```
x_train_norm=(x_train_reshaped-x_mean)/(x_std+epsilon)
```

In [22]:
```
x_test_norm=(x_test_reshaped-x_mean)/(x_std+epsilon)
```

In [32]:
```
#Creating a model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model=Sequential()
model.add(Dense(128,activation='relu',input_shape=(784,)))
model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
```

In [37]:
```
#compiling model
model.compile(optimizer='sgd',loss='categorical_crossentropy',metrics=['accuracy'])
#Training the model
model.fit(x_train_norm , y_train_encoded,epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3726 - accuracy: 0.8909
Epoch 2/3
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1831 - accuracy: 0.9459
Epoch 3/3
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1379 - accuracy: 0.9599
```

Out[37]:
```
<keras.callbacks.History at 0x176aa8babf0>
```

In [39]:
```
#Evaluating the model
loss,accuracy=model.evaluate(x_test_norm,y_test_encoded)
print('Testset Accuracy=',accuracy*100)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.1266 - accuracy: 0.9599
Testset Accuracy= 95.99000215530396
```

In [47]:
```
model.predict([x_test_norm])
```

```
313/313 [==============================] - 0s 1ms/step
```

Out[47]:
```
array([[1.93954784e-05, 4.96380210e-07, 1.04715553e-04, ...,
        9.97893512e-01, 1.60243217e-05, 1.53789253e-04],
       [4.19750031e-05, 5.97508915e-04, 9.97321427e-01, ...,
        4.29165103e-09, 3.27578688e-04, 5.74905545e-09],
       [1.24418057e-05, 9.91768122e-01, 2.76199426e-03, ...,
        1.77408976e-03, 1.34497217e-03, 6.25114131e-04],
       ...,
       [1.11243615e-07, 1.40273828e-08, 2.40297226e-07, ...,
        7.94289881e-05, 2.96815939e-04, 5.34727471e-04],
       [1.46710272e-05, 4.70879022e-05, 7.05091225e-06, ...,
        6.90600416e-07, 7.89845549e-03, 2.99538897e-06],
       [2.40739537e-05, 6.86454143e-07, 1.47090628e-04, ...,
        2.66403943e-09, 1.96499468e-06, 2.13132111e-07]], dtype=float32)
```