In [1]:
```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data.dataloader as dataloader
import torch.optim as optim

from torch.utils.data import TensorDataset
from torchvision import transforms
from torchvision.datasets import MNIST

import matplotlib.pyplot as plt
import time
```

```
/Users/yao/anaconda3/envs/eecs690/lib/python3.6/site-packages/tqdm/auto.py:22: TqdmWarnin
g: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readt
hedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Load the mnist database into train and test sets.

In [14]:
```
train = MNIST('./data', train=True, download=True, transform=transforms.ToTensor())
test = MNIST('./data', train=False, download=True, transform=transforms.ToTensor())
```

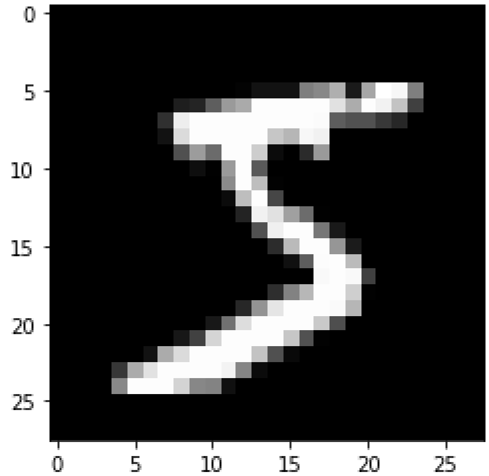
In [26]:
```
train_data = train.train_data
train_data = train.transform(train_data.numpy())

print('[Train]')
print(' - Numpy Shape:', train.train_data.numpy().shape)
print(' - Tensor Shape:', train.train_data.size())
print(' - min:', torch.min(train_data))
print(' - max:', torch.max(train_data))
print(' - mean:', torch.mean(train_data))
print(' - std:', torch.std(train_data))
print(' - var:', torch.var(train_data))
```

```
[Train]
 - Numpy Shape: (60000, 28, 28)
 - Tensor Shape: torch.Size([60000, 28, 28])
 - min: tensor(0.)
 - max: tensor(1.)
 - mean: tensor(0.1307)
 - std: tensor(0.3081)
 - var: tensor(0.0949)
```

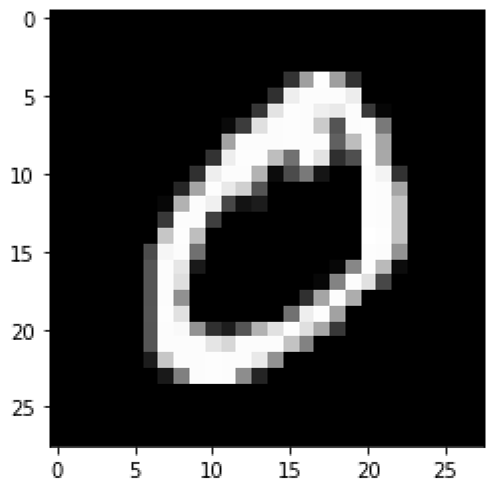
In [28]:
```
# Visualize a training instance with matplotlib
plt.imshow(train.train_data.numpy()[0], cmap='gray')
```

Out[28]:
```
<matplotlib.image.AxesImage at 0x7f8f36285908>
```



In [29]:
```
plt.imshow(train.train_data.numpy()[1], cmap='gray')
```

Out[29]: <matplotlib.image.AxesImage at 0x7f8f36361320>



Question 1:

Construct the dataloader for both training and testing sets. Set batch size to be 64 for now.

In [ ]:
```

```

Question 2:

Create a sequential model neural network that has 2 hidden layers that are 32 neurons wide. The final shape should be: Inupt, 32, 32, 10. This will be your baseline model. This network should use the Stochastic Gradient Descent optimizer. It should use the ReLU activation on each layer except the last one which needs to be a log_softmax layer. The loss function should be categorical crossentroppy and the metric to use should be accuracy. Batch size is 64, epochs is 30, learning_rate is 0.01. You will use the testing dataset as your validation data for the model in the fit command. The accuracy you will be asked about for the rest of the lab is the validation accuracy. You will also compare every future model against this one.

Hint: Think about what shape the data needs to be in for a neural network. The default shape for a MNIST image is 28x28 pixles. The input of image needs to be flattened.

In [ ]:

```

```

Question 3:

Now create another network that has the shape: Input, 64, 64, 10. How does increasing the number of neurons per hidden state affect accuracy and training speed? Does it make sense to increase the number of neurons in each layer? If so, why and by how much?

In [ ]:

```

```

Question 4:

Now create a network with the shape: Input, 32, 32, 10 again. This time adjust the SGD optimizer and set the learning rate to 0.01, the decay rate to 0.000001, and momentum rate to 0.9. How does this model compare to the two before it in terms of accuracy and computation speed? Is using momentum in our models a good idea?

In [ ]:

```

```

Question 5:

Now create a network with the shape: Input, 32, 32, 10. This time we will adjust the batch size. Run one model with a batch size of 128 and again with a batch size of 32. How does batch size effect computation speed and accuracy? Why do you think that is?

In [ ]:

```

```

Question 6:

Now we are going to add another hidden layer. Create a network with the shape: Input, 32, 32, 32, 10. How did adding another layer effect computation time and accuracy? Why do you think that is?

In [ ]:

```

```

Question 7:

Now create a network with the shape: Input, 128, 128, 10. This time we are going to introduce a dropout (https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html). After the input and both hidden layers add a dropout layer. Set the dropout value to 0.5. How does this effect accuracy and computation time? Think about what dropout is doing to our network. Does it make sense to use droput and how many neurons are being used in each layer during training? Does this effect training speed?

In [ ]:

Question 8:

Think about all of the networks you have made thus far. Combine the positive aspects of each network before and create the ‘best’ network you can. Give your accuracy and parameters of your network, visualize the loss curve according to epoch number.

In [ ]: