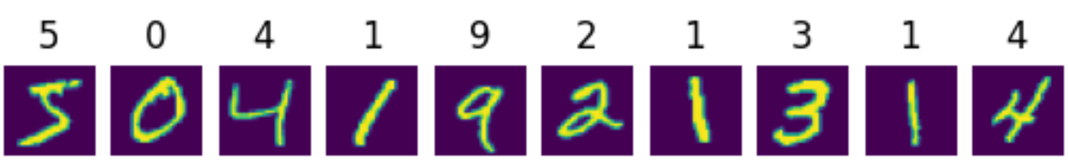
Questions 1

In [2]:
```python
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import numpy as np
import random

# for easier reading np
np.set_printoptions(precision=3,suppress=True)
```

Data

In [7]:
```python
# Prepare the data
from torchvision.datasets import MNIST

db_train = MNIST(root="./", train=True, transform=None, target_transform=Non
db_test = MNIST(root="./", train=False, transform=None, target_transform=Non
def to1hot(labels):
    """Converts an array of class labels into their 1hot encodings.
    Assumes that there are at most three classes."""
    return torch.eye(10)[labels]

for i in range(10):
    img, lbl = db_train[i]
    plt.subplot(1, 10, i+1)
    plt.imshow(img)
    plt.title(lbl)
    plt.axis(False)
```



In [8]:
```python
# Reading the dataset
def data_iter(batch_size, db):
    num_examples = len(db)

    # The examples are read at random, in no particular order
    indices = list(range(num_examples))
    random.shuffle(indices)
    indices = indices[:10000]
    for i in range(0, 10000, batch_size):
        X, Y = [], []
        for j in indices[i:i + batch_size]:
            img, lbl = db[j]
```

```
            # Process image
            img = torch.from_numpy(np.array(img))   # Convert PIL to numpy a
            img = img.view(28*28)                   # Image is 28x28. For ou
            img = img.float() / 255.                # Make the pixel values 

            lbl = torch.tensor(lbl)

            X.append(img), Y.append(lbl)
        yield torch.stack(X), torch.stack(Y)

# Check data reader
for X_batch, y_batch in data_iter(batch_size=10, db=db_train):
    print('X_batch', X_batch.shape, X_batch.dtype)
    print('y_batch', y_batch.shape, y_batch.dtype)
    break
```

```
X_batch torch.Size([10, 784]) torch.float32
y_batch torch.Size([10]) torch.int64
```

Model

In [9]:
```
# Define Model
from torch import nn
class Linear(nn.Module):
    def __init__(self, input_dim):
        super(Linear, self).__init__()
        self.layer1 = nn.Linear(input_dim, 10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.softmax(self.layer1(x))
        return x

# Check model
model = Linear(28*28)
for X_batch, y_batch in data_iter(batch_size=16, db=db_train):
    out_batch = model(X_batch)
    print('X_batch', X_batch.shape)
    print('out_batch', out_batch.shape)
    break
```

```
X_batch torch.Size([16, 784])
out_batch torch.Size([16, 10])
```

Training

In [10]:
```
# Optimization algorithms
def sgd(model, lr):
    """Minibatch stochastic gradient descent."""
    for p in model.parameters():
```

```
        p.data -= lr * p.grad
        p.grad = None

# Loss Functions and Accuracy Metric
def mse(y_hat, y):
    """MSE Loss."""
    loss_per_sample = (to1hot(y) - yhat).pow(2).sum(1)
    return loss_per_sample.mean()

def accuracy(y_hat, y):
    return (y_hat.argmax(dim=1) == y).float().mean()

# Check functions
yhat = torch.tensor([[0.2, 0.8, 0.1, 0, 0, 0, 0, 0, 0, 0],[0.6, 0.3, 0.1, 0,
y = torch.tensor([1, 1])
loss = mse(yhat, y)
acc = accuracy(yhat, y)
print(f'loss = {loss}    acc = {acc}')
```

```
loss = 0.4749999940395355    acc = 0.5
```

In [11]:

```
# Hyperparameters
lr = 0.01
batch_size = 16
num_epochs = 10
```

In [12]:

```
# Training
model = Linear(28*28)
# model = MLP(28*28)
for epoch in range(num_epochs):
    # Train for one epoch
    losses = []
    for X_batch, y_batch in data_iter(batch_size=batch_size, db=db_train):
        # Use model to compute predictions
        yhat = model(X_batch)
        l = mse(yhat, y_batch)  # Minibatch loss in `X_batch` and `y_batch`

        # Compute gradients by back propagation
        l.backward()

        # Update parameters using their gradient
        sgd(model, lr)

        losses.append(l.detach().item())

    # Measure accuracy on the test set
    acc = []
    for X_batch, y_batch in data_iter(batch_size=16, db=db_test):
        yhat = model(X_batch)
        acc.append(accuracy(yhat, y_batch))

    print(f"Epoch {epoch+1}: Train Loss {np.mean(losses):.3f} Test Accuracy
```

```
Epoch 1: Train Loss 0.706 Test Accuracy 0.753
Epoch 2: Train Loss 0.425 Test Accuracy 0.834
Epoch 3: Train Loss 0.324 Test Accuracy 0.860
Epoch 4: Train Loss 0.271 Test Accuracy 0.870
Epoch 5: Train Loss 0.247 Test Accuracy 0.878
Epoch 6: Train Loss 0.228 Test Accuracy 0.882
Epoch 7: Train Loss 0.225 Test Accuracy 0.888
Epoch 8: Train Loss 0.205 Test Accuracy 0.889
Epoch 9: Train Loss 0.205 Test Accuracy 0.892
Epoch 10: Train Loss 0.194 Test Accuracy 0.896
```

In [13]:

```python
# Evaluation
with torch.no_grad():
    yhat, y = [], []
    for X_batch, y_batch in data_iter(batch_size=16, db=db_test):
        yhat.append(model(X_batch))
        y.append(y_batch)

yhat = torch.cat(yhat, dim=0).argmax(dim=1)
y = torch.cat(y, dim=0)
cm = to1hot(y).T@to1hot(yhat)
print('CM = \n', cm.numpy())
```

```
CM =
 [[ 959.    0.    2.    3.    0.    0.    8.    1.    7.    0.]
 [   0. 1103.    3.    4.    1.    5.    4.    0.   15.    0.]
 [  13.    3.  887.   18.   16.    1.   20.   26.   37.   11.]
 [   6.    1.   22.  888.    1.   38.    6.   14.   21.   13.]
 [   2.    6.    5.    0.  897.    1.   13.    1.    9.   48.]
 [  19.   11.    7.   36.   22.  713.   23.    9.   43.    9.]
 [  20.    3.    6.    2.    7.   19.  895.    0.    6.    0.]
 [   4.   22.   34.    1.   13.    0.    1.  911.    5.   37.]
 [  12.    9.   14.   25.   11.   28.   15.   15.  828.   17.]
 [  13.    8.    7.   10.   42.   16.    2.   21.    8.  882.]]
```

Question 1-a

Loss Function:

$$L_{sq} = \frac{1}{2N}\sum_{i=1}^{N}(Z_2(i) - Y(i))^2$$

$$\delta_2^i = \frac{\partial L_{sq}}{\partial U_2(i)} = \frac{\partial L_{sq}}{\partial Z_2(i)} \cdot \frac{\partial Z_2(i)}{\partial U_2(i)}$$

$$\frac{\partial L_{sq}}{\partial Z_2(i)} = \frac{1}{N}\sum_{i=1}^{N}(Z_2(i) - Y(i))$$

$$\frac{\partial Z_2(i)}{\partial U_2(i)} = \sigma(U_2(i)) \cdot (1 - \sigma(U_2(i))) = Z_2(i) \cdot (1 -$$

$$\delta_2^i = \frac{1}{N}\sum_{i=1}^{N}(Z_2(i) - Y(i)) \cdot Z_2 \cdot (1 - Z_2)$$

$$\delta_1^i = \frac{\partial L_{sq}}{\partial U_1(i)} = \frac{\partial L_{sq}}{\partial Z_2(i)} \cdot \frac{\partial Z_2(i)}{\partial U_2(i)} \cdot \frac{\partial U_2(i)}{\partial Z_1(i)} \cdot \frac{\partial Z_1(i)}{\partial U_1(i)} = \delta_2^i \cdot \frac{\partial U_2(i)}{\partial Z_1(i)} \cdot \frac{\partial Z_1(i)}{\partial U_1(i)}$$

$$\frac{\partial U_2(i)}{\partial Z_1(i)} = W_2[1:]$$

$$\frac{\partial Z_1(i)}{\partial U_1(i)} = \frac{\partial Z_1(i)}{\partial U_1(i)} = \sigma(U_1(i)) \cdot (1 - \sigma(U_1(i))) = Z_1(i) \cdot (1 - Z_1(i))$$

$$\delta_1^i = \delta_2^i \cdot W_2[1:] \cdot \sigma(U_1(i)) \cdot (1 - \sigma(U_1(i)))$$

Question 1-b

$$\frac{\partial L_{sq}}{\partial W_2} = \frac{\partial L_{sq}}{\partial U_2} \cdot \frac{\partial U_2}{\partial W_2}$$

$$\frac{\partial L_{sq}}{\partial W_2} = \delta_2^i \cdot \begin{bmatrix} 1 & Z_1 \end{bmatrix}$$

$$\frac{\partial L_{sq}}{\partial W_1} = \frac{\partial L_{sq}}{\partial U_1} \cdot \frac{\partial U_1}{\partial W_1}$$

$$\frac{\partial L_{sq}}{\partial W_1} = \delta_1^i \cdot \begin{bmatrix} 1 & X \end{bmatrix}$$

Question 1-c

$$W_1^{t+1} \leftarrow W_1^t - \alpha \frac{\partial L_{sq}}{\partial W_1^t}$$

$$W_2^{t+1} \leftarrow W_2^t - \alpha \frac{\partial L_{sq}}{\partial W_2^t}$$

In [ ]:
```
X = torch.tensor([[1, 2], [2, -1], [3, 0]], dtype=torch.float32)
y = torch.tensor([[0], [1], [0]], dtype=torch.float32)
W1 = torch.tensor([[0.1, -0.1], [0.2, 0], [0, 0.3]], dtype=torch.float32, re
W2 = torch.tensor([[0.05], [-0.1], [0.2]], dtype=torch.float32, requires_gra
```