In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.utils import to_categorical
```

```
2025-02-20 00:14:29.302685: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may se
e slightly different numerical results due to floating-point round-off errors from different computation or
ders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-02-20 00:14:29.361082: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is
optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

In [2]:
```
def init_params(nx, nh, ny):
    return {
        'w1': np.random.normal(0, 0.3, (nx, nh)),
        'b1': np.zeros((1, nh)),
        'w2': np.random.normal(0, 0.3, (nh, ny)),
        'b2': np.zeros((1, ny)),
    }
```

In [3]:
```
def softmax(x):
    x_expo = np.exp(x - np.max(x, axis=1, keepdims=True))
    return x_expo / np.sum(x_expo, axis=1, keepdims=True)
```

In [4]:
```
def forward(params, x):
    a1 = x
    z1 = a1 @ params["w1"] + params["b1"]
    a2 = np.tanh(z1)
    z2 = a2 @ params["w2"] + params["b2"]
    a3 = softmax(z2)
    return a3, {
        "z1": z1, "a2": a2, "z2": z2, "a3": a3
    }
```

In [5]:
```
def loss_accuracy(yhat, y):
    m = y.shape[0]
    loss = -np.sum(y * np.log(yhat + 1e-10)) / m
```

```
    accuracy = np.mean(np.argmax(yhat, axis=1) == np.argmax(y, axis=1))
    return loss, accuracy
```

In [6]:

```
%%latex

# backward propagation

## we have

$$ z_1 = W_1x_1 $$
$$ a_1 = \sigma(z_1) = \sigma(W_1x_1) = \tanh(W_1x_1) $$
$$ z_2 = W_2a_2 = w_2 \sigma(w_1x_1)  $$
$$ a_2 = \hat{y} = \sigma(z_2) = \sigma(w_2 \sigma(w_1x_1)) = \frac{1}{1 + e^{-z_2}} $$
$$ J = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}, y) $$

## $$ \frac{\partial J}{\partial w_2} = ? $$

$$ \frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial 
$$ \frac{\partial J}{\partial \hat{y}} = \frac{\partial [\frac{-1}{n} \sum_{i=1}^{n} (y\log(\hat{y}) + (1-
$$ \frac{\partial \hat{y}}{\partial z_2} = \frac{\partial (\frac{1}{1+\exp(-z_2)})}{\partial z_2} = \frac{
$$ \frac{\partial z_2}{\partial w_2} = \frac{\partial (w_2a_1)}{\partial w_2} = a_1 $$

### so
$$ \frac{\partial J}{\partial w_2} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}\hat{y}(1-\hat{y})a_1 = (\hat{y

## $$ \frac{\partial J}{\partial w_1} = ? $$

$$ \frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial 
$$ \frac{\partial J}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} $$
$$ \frac{\partial \hat{y}}{\partial z_2} = \hat{y}(1-\hat{y}) $$
$$ \frac{\partial z_2}{\partial a_1} = \frac{\partial (w_2a_1)}{\partial a_1} = w_2 $$
$$ \frac{\partial a_1}{\partial z_1} = \frac{\partial (\sigma(z_1))}{\partial z_1} = 1 - \tanh²(z_1) = 1 -
$$ \frac{\partial z_1}{\partial w_1} = \frac{\partial (w_1x_1)}{\partial w_1} = x_1 $$

### so
$$ \frac{\partial J}{\partial w_1} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \hat{y}(1-\hat{y}) w_2 (1 - a_
```

backward propagation ## we have

$$z_1 = W_1 x_1$$

$$a_1 = \sigma(z_1) = \sigma(W_1 x_1) = \tanh(W_1 x_1)$$

$$z_2 = W_2 a_2 = w_2 \sigma(w_1 x_1)$$

$$a_2 = \hat{y} = \sigma(z_2) = \sigma(w_2 \sigma(w_1 x_1)) = \frac{1}{1 + e^{-z_2}}$$

$$J = \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}, y)$$

##

$$\frac{\partial J}{\partial w_2} = ?$$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial w_2}$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial [\frac{-1}{n} \sum_{i=1}^{n} (y \log(\hat{y}) + (1-y) \log(1-\hat{y}))]}{\partial \hat{y}} = \frac{-n}{n} (\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}) = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z_2} = \frac{\partial (\frac{1}{1+\exp(-z_2)})}{\partial z_2} = \frac{\exp(-z_2)}{[1 + \exp(-z_2)]^2} = \hat{y}(1-\hat{y})$$

$$\frac{\partial z_2}{\partial w_2} = \frac{\partial (w_2 a_1)}{\partial w_2} = a_1$$

so

$$\frac{\partial J}{\partial w_2} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \hat{y}(1-\hat{y}) a_1 = (\hat{y} - y) a_1$$

##

$$\frac{\partial J}{\partial w_1} = ?$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z_2} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial z_2}{\partial a_1} = \frac{\partial (w_2 a_1)}{\partial a_1} = w_2$$

$$\frac{\partial a_1}{\partial z_1} = \frac{\partial (\sigma(z_1))}{\partial z_1} = 1 - \tanh^2(z_1) = 1 - {a_1}^2$$

$$\frac{\partial z_1}{\partial w_1} = \frac{\partial (w_1 x_1)}{\partial w_1} = x_1$$

so

$$\frac{\partial J}{\partial w_1} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \hat{y}(1 - \hat{y}) w_2 (1 - {a_1}^2) x_1 = (\hat{y} - y) w_2 (1 - {a_1}^2) x_1$$

In [7]:

```
def backward(x, params, outputs, y):
    m = y.shape[0]
    d_z2 = outputs["a3"] - y
    d_w2 = outputs["a2"].T @ d_z2 / m
    d_b2 = np.sum(d_z2, axis=0, keepdims=True) / m
    d_a2 = d_z2 @ params["w2"].T
    d_z1 = d_a2 * (1 - np.tanh(outputs["a2"])**2)
    d_w1 = x.T @ d_z1 / m
    d_b1 = np.sum(d_z1, axis=0, keepdims=True) / m
    return {
        "d_w1": d_w1,
        "d_b1": d_b1,
        "d_w2": d_w2,
        "d_b2": d_b2
    }
```

In [8]:
```python
def sgd(params, grads, eta):

    params["w1"] = params["w1"] - eta * grads["d_w1"]
    params["b1"] = params["b1"] - eta * grads["d_b1"]
    params["w2"] = params["w2"] - eta * grads["d_w2"]
    params["b2"] = params["b2"] - eta * grads["d_b2"]

    return params
```

training steps

In [9]:
```python
def training_steps(x_train, y_train, x_test, y_test, nx, nh, ny, epochs, batch_size, eta):
    params = init_params(nx, nh, ny)
    loss_history = []
    accuracy_history = []

    for epoch in range(epochs):
        # random data
        permutation = np.random.permutation(x_train.shape[0])
        x_train = x_train[permutation]
        y_train = y_train[permutation]

        for i in range(0, x_train.shape[0], batch_size):
            # load batch
            x_batch = x_train[i:i + batch_size]
            y_batch = y_train[i:i + batch_size]

            # forward
            yhat, outputs = forward(params, x_batch)

            # compute loss/accuracy
            loss, accuracy = loss_accuracy(yhat, y_batch)

            # backward
            grads = backward(x_batch, params, outputs, y_batch)

            # sgd
            sgd(params, grads, eta)
```

```python
        yhat_test, _ = forward(params, x_test)
        test_loss, test_acc = loss_accuracy(yhat_test, y_test)
        loss_history.append(test_loss)
        accuracy_history.append(test_acc)

        print(f"epoch {epoch+1}/{epochs} ### loss: {test_loss:.4f}, ### accuracy: {test_acc:.4f}")

    return params, loss_history, accuracy_history
```

In [10]:
```python
def plot_history(history, name):
    plt.plot(history, label=name)
    plt.xlabel("epoch")
    plt.ylabel(name)
    plt.legend()
```

In [11]:
```python
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [12]:
```python
x_train = x_train.reshape(x_train.shape[0], -1) / 255.0
x_test = x_test.reshape(x_test.shape[0], -1) / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

In [13]:
```python
nx = x_train.shape[1]
nh = 128
ny = y_train.shape[1]
epochs=50
batch_size=128
eta=0.1
```

In [14]:
```python
nx
```

Out[14]:
```
784
```

In [15]:
```python
ny
```

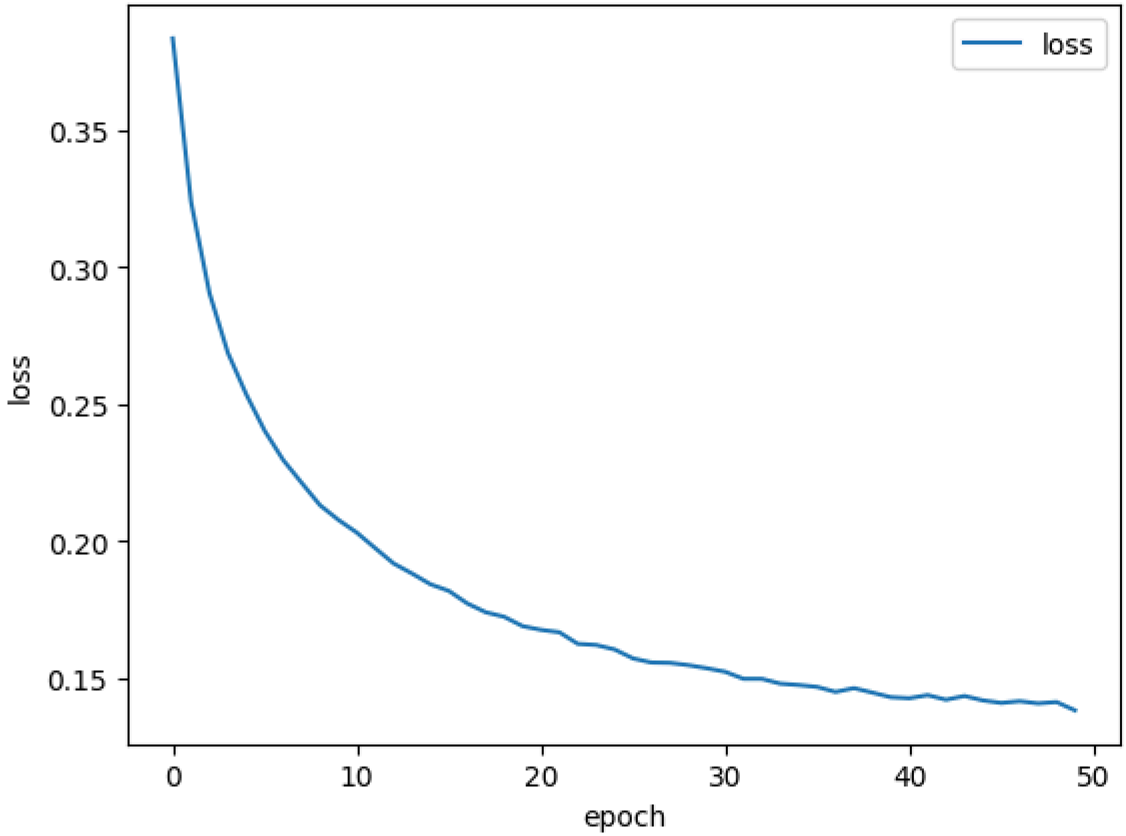
Out[15]:
```
10
```

In [16]:
```python
params, loss_history, accuracy_history = training_steps(x_train, y_train, x_test, y_test, nx, nh, ny, epocl
```

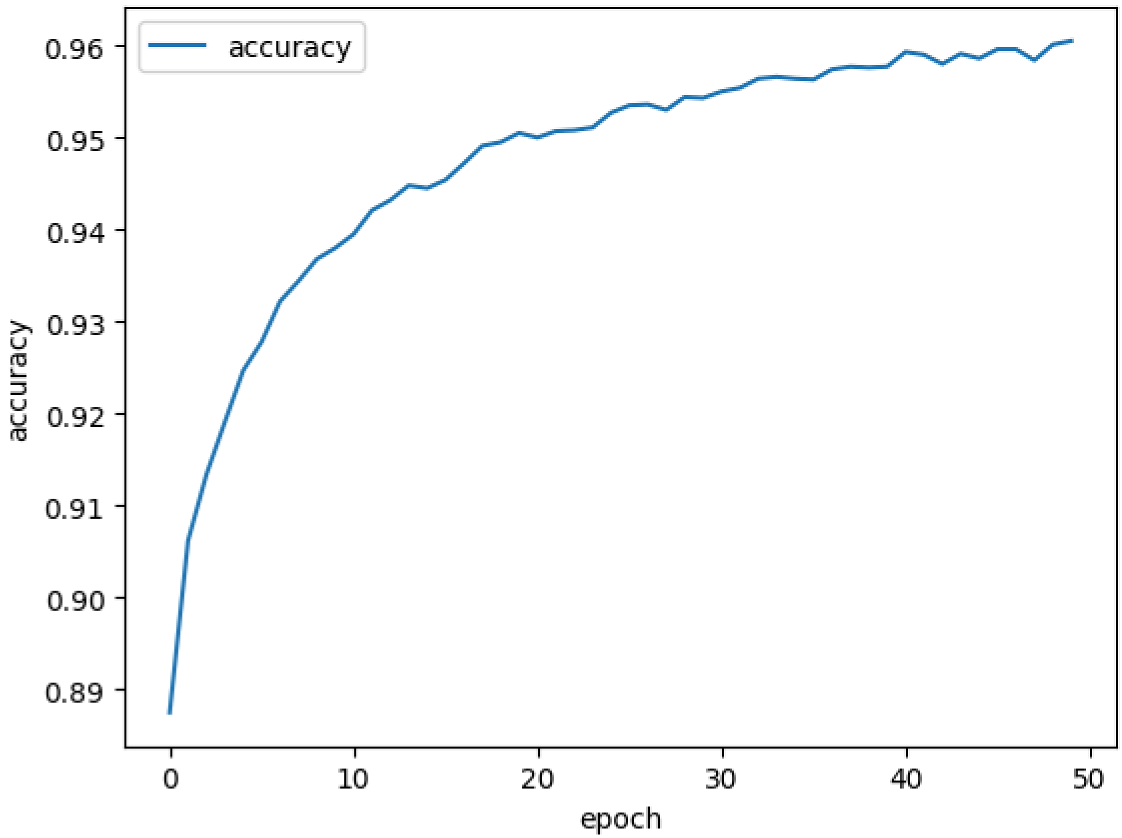
```
epoch 1/50 ### loss: 0.3833, ### accuracy: 0.8875
epoch 2/50 ### loss: 0.3235, ### accuracy: 0.9062
epoch 3/50 ### loss: 0.2903, ### accuracy: 0.9134
epoch 4/50 ### loss: 0.2686, ### accuracy: 0.9191
epoch 5/50 ### loss: 0.2534, ### accuracy: 0.9247
epoch 6/50 ### loss: 0.2402, ### accuracy: 0.9278
epoch 7/50 ### loss: 0.2296, ### accuracy: 0.9322
epoch 8/50 ### loss: 0.2213, ### accuracy: 0.9344
epoch 9/50 ### loss: 0.2131, ### accuracy: 0.9368
epoch 10/50 ### loss: 0.2077, ### accuracy: 0.9380
epoch 11/50 ### loss: 0.2031, ### accuracy: 0.9395
epoch 12/50 ### loss: 0.1974, ### accuracy: 0.9421
epoch 13/50 ### loss: 0.1918, ### accuracy: 0.9432
epoch 14/50 ### loss: 0.1882, ### accuracy: 0.9448
epoch 15/50 ### loss: 0.1843, ### accuracy: 0.9445
epoch 16/50 ### loss: 0.1818, ### accuracy: 0.9454
epoch 17/50 ### loss: 0.1773, ### accuracy: 0.9472
epoch 18/50 ### loss: 0.1741, ### accuracy: 0.9491
epoch 19/50 ### loss: 0.1724, ### accuracy: 0.9495
epoch 20/50 ### loss: 0.1690, ### accuracy: 0.9505
epoch 21/50 ### loss: 0.1676, ### accuracy: 0.9500
epoch 22/50 ### loss: 0.1668, ### accuracy: 0.9507
epoch 23/50 ### loss: 0.1626, ### accuracy: 0.9508
epoch 24/50 ### loss: 0.1621, ### accuracy: 0.9511
epoch 25/50 ### loss: 0.1605, ### accuracy: 0.9527
epoch 26/50 ### loss: 0.1572, ### accuracy: 0.9535
epoch 27/50 ### loss: 0.1557, ### accuracy: 0.9536
epoch 28/50 ### loss: 0.1556, ### accuracy: 0.9530
epoch 29/50 ### loss: 0.1548, ### accuracy: 0.9544
epoch 30/50 ### loss: 0.1536, ### accuracy: 0.9543
epoch 31/50 ### loss: 0.1524, ### accuracy: 0.9550
epoch 32/50 ### loss: 0.1498, ### accuracy: 0.9554
epoch 33/50 ### loss: 0.1498, ### accuracy: 0.9564
epoch 34/50 ### loss: 0.1480, ### accuracy: 0.9566
epoch 35/50 ### loss: 0.1475, ### accuracy: 0.9564
epoch 36/50 ### loss: 0.1468, ### accuracy: 0.9563
epoch 37/50 ### loss: 0.1450, ### accuracy: 0.9574
epoch 38/50 ### loss: 0.1464, ### accuracy: 0.9577
epoch 39/50 ### loss: 0.1448, ### accuracy: 0.9576
epoch 40/50 ### loss: 0.1431, ### accuracy: 0.9577
epoch 41/50 ### loss: 0.1427, ### accuracy: 0.9593
epoch 42/50 ### loss: 0.1438, ### accuracy: 0.9590
```

```
epoch 43/50 ### loss: 0.1422, ### accuracy: 0.9580
epoch 44/50 ### loss: 0.1435, ### accuracy: 0.9591
epoch 45/50 ### loss: 0.1419, ### accuracy: 0.9586
epoch 46/50 ### loss: 0.1410, ### accuracy: 0.9596
epoch 47/50 ### loss: 0.1416, ### accuracy: 0.9596
epoch 48/50 ### loss: 0.1408, ### accuracy: 0.9584
epoch 49/50 ### loss: 0.1413, ### accuracy: 0.9601
epoch 50/50 ### loss: 0.1383, ### accuracy: 0.9605
```

In [17]:
```
plot_history(loss_history, 'loss')
```



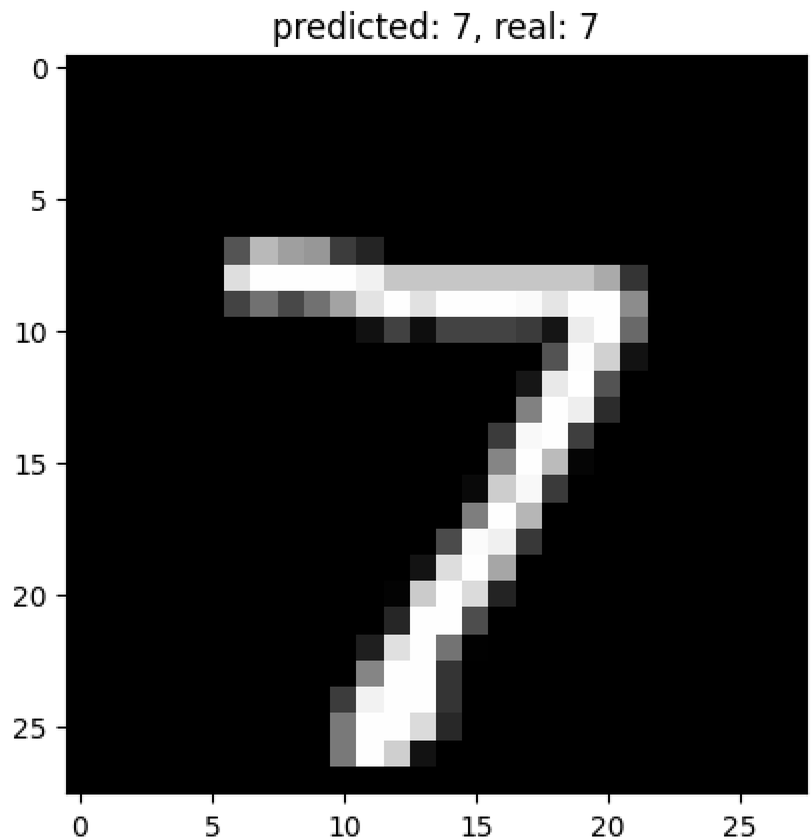
In [18]:
```
plot_history(accuracy_history, 'accuracy')
```

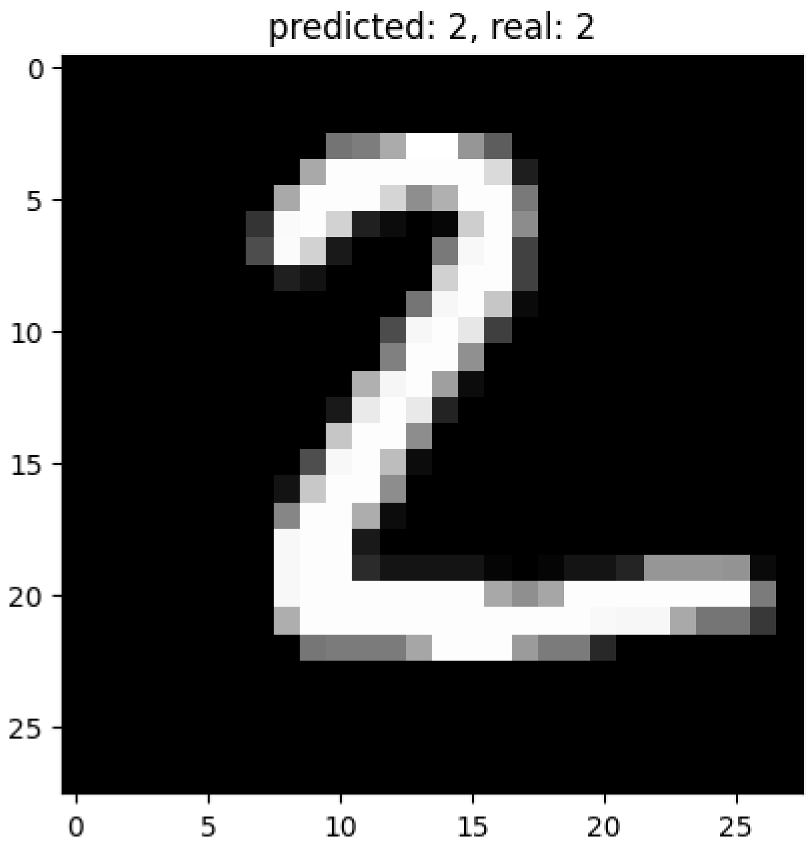


In [19]:
```
def hand_written_recognition(params, x_test, y_test, __range):
    for i in range(__range):
        image = x_test[i].reshape(28, 28)
        yhat, _ = forward(params, x_test[i].reshape(1, -1))
        predicted_label = np.argmax(yhat)
        real_label = np.argmax(y_test[i])

        plt.imshow(image, cmap='gray')
        plt.title(f"predicted: {predicted_label}, real: {real_label}")
        plt.show()
```

In [20]:
```
hand_written_recognition(params, x_test, y_test, 2)
```


predicted: 7, real: 7
0
5
10
15
20
25
0
5
10
15
20
25



In [21]:
```
import pandas as pd
```

In [22]:
```
mnist = pd.read_csv('./mnist.csv')
mnist.head()
```

Out[22]:

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23	28x24	28x25	28x2
0	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
4	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 785 columns

In [23]:
```
mnist.shape
```

Out[23]: (10000, 785)

In [24]:
```
mnist.describe()
```

Out[24]:

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	
count	10000.000000	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0	...	1000
mean	4.443400	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
std	2.895865	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	25

8 rows × 785 columns

In [25]:
```python
mnist.isna().sum()
```

Out[25]:
```
label     0
1x1       0
1x2       0
1x3       0
1x4       0
         ..
28x24     0
28x25     0
28x26     0
28x27     0
28x28     0
Length: 785, dtype: int64
```