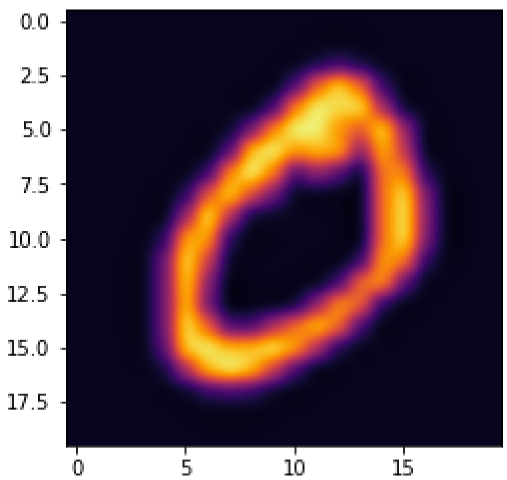
In [1]:

```python
# -*- coding: utf-8 -*-

import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
from scipy.io import loadmat
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
import copy
from matplotlib import cm
from matplotlib.animation import FuncAnimation
import scipy.optimize
import networkx as nx
```

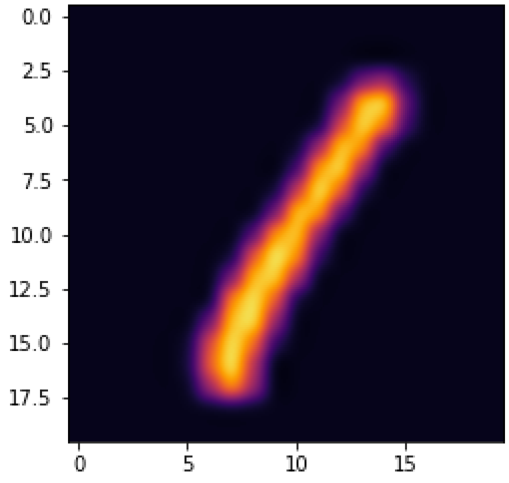
In [3]:

```
# task 1
# Загрузите данные ex4data1.mat из файла.
data = loadmat('G:/Labs/bsuir-labs/11cem/ml/lab04/data/ex4data1.mat')

x = data['X']
y = data['y']

def show_digit(x):
    plt.imshow(np.array(np.split(x, 20)).T, interpolation='gaussian', cmap='inferno')
    plt.show()


def show_database(x, y):
    digits = set()
    i = 0
    while len(digits) != 10:
        if y[i][0] not in digits:
            print(y[i][0])
            show_digit(x[i])
            digits.add(int(y[i][0]))
        i += 1
        continue

show_database(x, y)

weights = loadmat('G:/Labs/bsuir-labs/11cem/ml/lab04/data/ex4weights.mat')
```

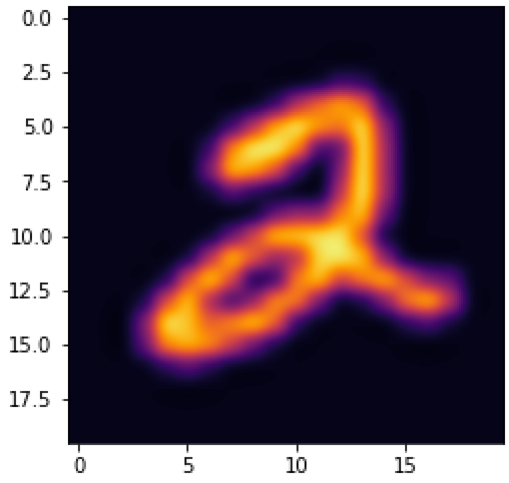
10



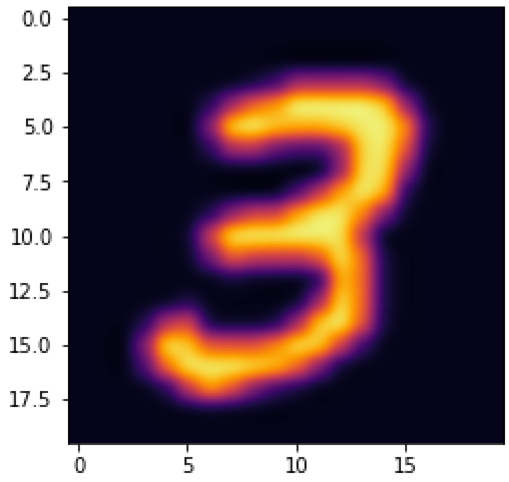
1


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15

2


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15

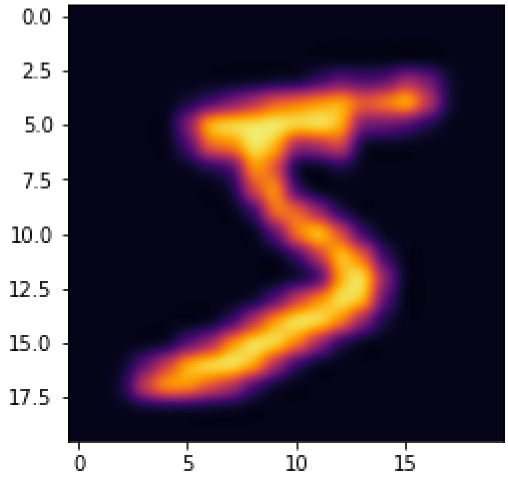
3


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15

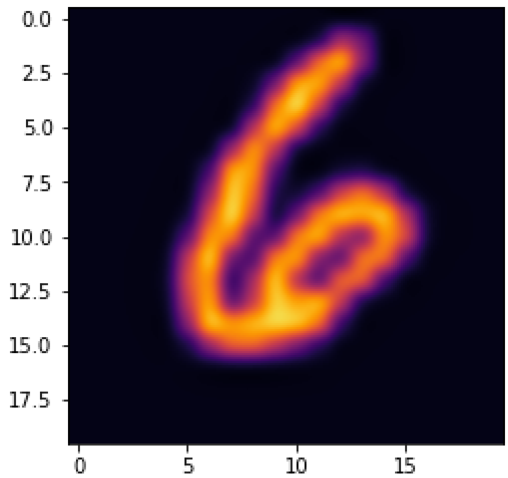
4


0.0
2.5
5.0

5

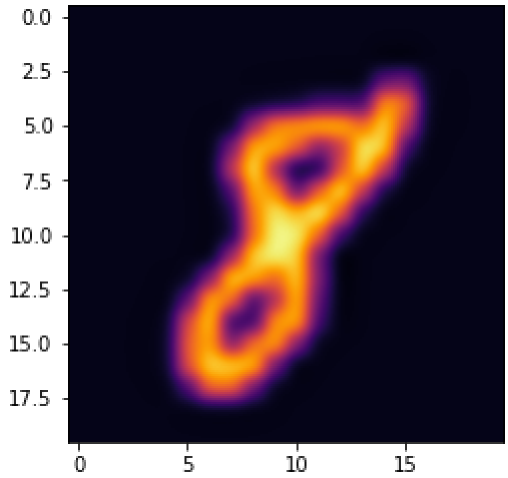

0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15

6

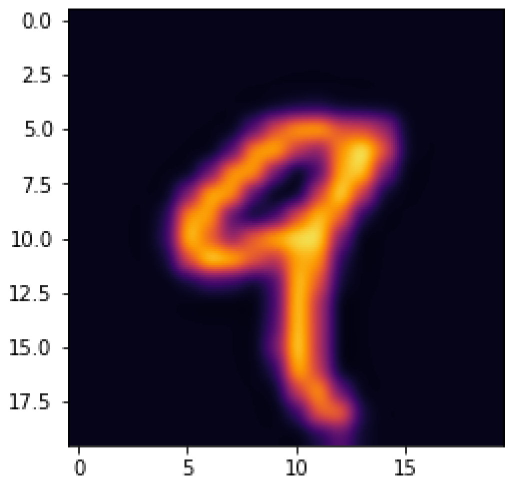

0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15

7

8



9



In [4]:

```
# task 2
# Загрузите веса нейронной сети из файла ex4weights.mat, который содержит две матрицы Θ(1
theta_1 = weights['Theta1']
theta_2 = weights['Theta2']

def show_head(array):
    print(array[:5])

def show_tail(array):
    print(array[5:])

show_tail(theta_1)

print("Network structure layer 1 = {}, layer 2 = {}".format(theta_1.shape, theta_2.shape)
print("input neuron layer: count {}+one bias neuron={} equal to features count(pixel coun
print("hidden neuron layer count {}+one bias neuron={} sets analytically".format(theta_1.
print("output neuron layer count {} equals to count of classes [1..10]".format(theta_2.sh
```

```
[[-5.97941174e-01 -7.76628103e-09  1.07444370e-08 ...  6.95713350e-05
  -1.04026165e-05 -5.65769223e-10]
 [ 1.54559442e-01 -6.38021750e-09 -6.05473425e-09 ...  7.52345604e-05
  -2.38386990e-06 -6.85497348e-09]
 [-3.37225729e-02  8.05170531e-09  5.42028087e-09 ...  3.57743751e-06
   2.87857120e-07  1.09306392e-08]
 ...
 [-1.83220638e-01 -8.89272060e-09 -9.81968100e-09 ...  2.35311186e-05
  -3.25484493e-06  9.02499060e-09]
 [-7.02096331e-01  3.05178374e-10  2.56061008e-09 ... -8.61759744e-04
   9.43449909e-05  3.83761998e-09]
 [-3.50933229e-01  8.85876862e-09 -6.57515140e-10 ... -1.80365926e-06
  -8.14464807e-06  8.79454531e-09]]
Network structure layer 1 = (25, 401), layer 2 = (10, 26)
input neuron layer: count 400+one bias neuron=401 equal to features count(pi
xel count)
hidden neuron layer count 25+one bias neuron=26 sets analytically
output neuron layer count 26 equals to count of classes [1..10]
```

In [5]:

```
# task 3
# Реализуйте функцию прямого распространения с сигмоидом в качестве функции активации.

# sigmoid
def activation(x):
    return 1 / (1 + np.exp(-x))


def println(message):
    print("\n{0}".format(message))

# функция прямого распространения
def predict(x, theta_1, theta_2, return_hidden_layer = False):
    # theta_1 and theta_2 is neural network config for lay1 and lay2
    ex_x = np.hstack((np.ones((len(x), 1)), x)) #расширение матрицы для умножения с первы
    lay_1_out = activation(ex_x.dot(theta_1.T)) #векторное умножение с последующим примен
    ex_lay_1_out = np.hstack((np.ones((len(lay_1_out), 1)), lay_1_out)) #расширение резул
    lay_2_out = activation(ex_lay_1_out.dot(theta_2.T)) #векторное умножение с последующи
    return ex_lay_1_out if return_hidden_layer else lay_2_out

predictions = predict(x, theta_1, theta_2)
```

In [6]:

```python
# task 4
# Вычислите процент правильных классификаций на обучающей выборке. Сравните полученный ре
# from run at 07/11/2019 - prediction: 95.98

def show_prediction_quality(predictions, y):
    digits = [[] for _ in range(10)]
    overall_stat = []
    for index, value in enumerate(y.squeeze()):
        is_predicted_correctly = value == np.argmax(predictions[index]) + 1
        digits[value - 1].append(is_predicted_correctly)
        overall_stat.append(is_predicted_correctly)
    for index, digits_prediction_results in enumerate(digits):
        correct = len([item for item in digits_prediction_results if item == True])
        whole = len(digits_prediction_results)
        percentage = round(100 * correct / whole, 1)
        print("{0}: {1} out of {2} = {3}%".format(index, correct, whole, percentage))
    print("prediction quality: {}%".format(
        round(100 * len([item for item in overall_stat if item == True]) / len(overall_st

println("PREDEFINED NETWORK")
show_prediction_quality(predictions, y)
print("logistic regression prediction quality = 95.98%")
print("newural network has better result")
```

```
PREDEFINED NETWORK
0: 491 out of 500 = 98.2%
1: 485 out of 500 = 97.0%
2: 480 out of 500 = 96.0%
3: 484 out of 500 = 96.8%
4: 492 out of 500 = 98.4%
5: 493 out of 500 = 98.6%
6: 485 out of 500 = 97.0%
7: 491 out of 500 = 98.2%
8: 479 out of 500 = 95.8%
9: 496 out of 500 = 99.2%
prediction quality: 97.5%
logistic regression prediction quality = 95.98%
newural network has better result
```

In [7]:

```
# task 5
# Перекодируйте исходные метки классов по схеме one-hot.

# onehot means
# 1 equals to [1 0 0 0 0 0 0 0 0 0]
# 2 equals to [0 1 0 0 0 0 0 0 0 0]
# 0 equals to [0 0 0 0 0 0 0 0 0 1]

def one_hot_convert(y):
    new_y = []
    for answer in y.squeeze():
        onehot = np.zeros(10)
        onehot[answer - 1] = 1
        new_y.append(onehot)
    return np.array(new_y)

one_hot_y = one_hot_convert(y)
```

In [8]:

```
# task 6
# Реализуйте функцию стоимости для данной нейронной сети.

def cost_with_devide(predictions, one_hot_y):
    batch_cost_accumulate = 0
    for y_pred, y_act in zip(predictions, one_hot_y):  # predicted and real answers
        record_cost_accumulate = sum((y_act-y_pred)**2)/len(y_act)  # 10
        batch_cost_accumulate += record_cost_accumulate
    batch_cost_accumulate /= len(one_hot_y) # 5000
    return batch_cost_accumulate

def cost(predictions, one_hot_y):
    batch_cost_accumulate = 0
    for y_pred, y_act in zip(predictions, one_hot_y):  # predicted and real answers
        record_cost_accumulate = sum((y_act-y_pred)**2)#/len(y_act)  # 10
        batch_cost_accumulate += record_cost_accumulate
    # batch_cost_accumulate /= len(one_hot_y) # 5000
    return batch_cost_accumulate



def get_wrong_prediction(y):
    import random
    new_y = []
    for _ in y.squeeze():
        onehot = np.zeros(10)
        onehot[random.randint(0,9)] = 1
        new_y.append(onehot)
    return np.array(new_y)
```

In [9]:

```
# task 7
# Добавьте L2-регуляризацию в функцию стоимости.

# not layer 2
# y should be one hotted
def l2cost(theta_1,theta_2, x,y, lambda_=10000):
     predictions = predict(x, theta_1, theta_2)
     return sum(sum((predictions - y) ** 2) * lambda_ / (2 * len(y)))

print("Predefined thetas cost: {}".format(cost(predictions, one_hot_y)))


print("Predefined thetas l2 cost: {}".format(l2cost(theta_1,theta_2, x,one_hot_y)))
```

```
Predefined thetas cost: 304.6618826303785
Predefined thetas l2 cost: 304.6618826303791
```

In [10]:

```
# task 8
# Реализуйте функцию вычисления производной для функции активации.
def sigmoid_speed(x):
    return np.exp(-x) / ((1 + np.exp(-x)) ** 2)


def create_random_theha(shape):
    import random
    theta = []
    for _ in range(shape[0]):
        theta.append(np.array([random.uniform(-1, 1) for _ in range(shape[1])]))
    return np.array(theta)
```

In [13]:

```
# task 9
# Инициализируйте веса небольшими случайными числами.

println("RANDOM THETA SET")
rand_tetha_1 = create_random_theha((25, 401))
rand_tetha_2 = create_random_theha((10, 26))

trained_predictions = predict(x, rand_tetha_1, rand_tetha_2)
show_prediction_quality(trained_predictions, y)
```

```
RANDOM THETA SET
0: 0 out of 500 = 0.0%
1: 42 out of 500 = 8.4%
2: 0 out of 500 = 0.0%
3: 0 out of 500 = 0.0%
4: 137 out of 500 = 27.4%
5: 41 out of 500 = 8.2%
6: 0 out of 500 = 0.0%
7: 29 out of 500 = 5.8%
8: 1 out of 500 = 0.2%
9: 54 out of 500 = 10.8%
prediction quality: 6.1%
```

In [14]:

```
# task 10
#Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети.

def revert_error_spread_train(x, rand_tetha_1, rand_tetha_2, y):
    for _ in tqdm(range(500)):
        predicted = predict(x, rand_tetha_1, rand_tetha_2)
        lay_1_predicted = predict(x, rand_tetha_1, rand_tetha_2, return_hidden_layer=True
        l2error = y - predicted
        l2delta = l2error * sigmoid_speed(predicted)
        l1error = np.dot(l2delta, rand_tetha_2)
        l1delta = l1error * sigmoid_speed(lay_1_predicted)
        l1delta = np.delete(l1delta, 0, 1)
        ex_x = np.hstack((np.ones((len(x), 1)), x))  # расширение матрицы для умножения с
        rand_tetha_1 += np.dot(ex_x.T, l1delta).T
        rand_tetha_2 += np.dot(lay_1_predicted.T, l2delta).T
    return rand_tetha_1, rand_tetha_2


trained_tetha_1, trained_tetha_2 = revert_error_spread_train(x, rand_tetha_1, rand_tetha_
```

```
  0%|
| 0/500 [00:00<?, ?it/s]c:\users\harwister\appdata\local\programs\python\pyt
hon36\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning: overflow en
countered in exp
  import sys
100%|██████████████████████████████████████████████████
████████| 500/500 [00:30<00:00, 16.04it/s]
```

```
RANDOM THETA SET
0: 31 out of 500 = 6.2%
```

```
1: 0 out of 500 = 0.0%
2: 0 out of 500 = 0.0%
3: 53 out of 500 = 10.6%
4: 0 out of 500 = 0.0%
5: 0 out of 500 = 0.0%
6: 146 out of 500 = 29.2%
7: 7 out of 500 = 1.4%
8: 0 out of 500 = 0.0%
9: 0 out of 500 = 0.0%
prediction quality: 4.7%

TRAINED THETA SET
0: 266 out of 500 = 53.2%
1: 248 out of 500 = 49.6%
2: 363 out of 500 = 72.6%
3: 341 out of 500 = 68.2%
4: 314 out of 500 = 62.8%
5: 438 out of 500 = 87.6%
6: 342 out of 500 = 68.4%
7: 424 out of 500 = 84.8%
8: 314 out of 500 = 62.8%
9: 432 out of 500 = 86.4%
prediction quality: 69.6%
```

In [15]:

```
println("TRAINED THETA SET")
trained_predictions = predict(x, trained_tetha_1, trained_tetha_2)
show_prediction_quality(trained_predictions, y) #todo uncomment
```

```
TRAINED THETA SET

c:\users\harwister\appdata\local\programs\python\python36\lib\site-packages
\ipykernel_launcher.py:7: RuntimeWarning: overflow encountered in exp
  import sys

0: 326 out of 500 = 65.2%
1: 470 out of 500 = 94.0%
2: 195 out of 500 = 39.0%
3: 313 out of 500 = 62.6%
4: 82 out of 500 = 16.4%
5: 26 out of 500 = 5.2%
6: 334 out of 500 = 66.8%
7: 161 out of 500 = 32.2%
8: 0 out of 500 = 0.0%
9: 0 out of 500 = 0.0%
prediction quality: 38.1%
```

In [16]:

```
# task 11
# Для того, чтобы удостоверится в правильности вычисленных значений градиентов используйт
def gradient(x, y, base_theta_1, base_theta_2, lambda_=0, alpha=0.001, iterations = 100):
    for _ in range(iterations):
        ex_x = np.hstack((np.ones((len(x), 1)), x))  # расширение матрицы для умножения с
        layer1 = activation(np.dot(ex_x, base_theta_1.T))
        ex_lay_1_out = np.hstack((np.ones((len(layer1), 1)), layer1))  # расширение резул
        layer2 = activation(np.dot(ex_lay_1_out, base_theta_2.T))
        layer2delta = (layer2 - y) * (layer2 * (1-layer2))
        layer1delta = np.dot(layer2delta, base_theta_2) * (ex_lay_1_out * (1-ex_lay_1_out
        l2_regularization = 1 - lambda_ / len(x)
        base_theta_2 = base_theta_2 * l2_regularization - alpha * np.dot(ex_lay_1_out.T,
        layer1delta = np.delete(layer1delta, 0, 1)
        base_theta_1 = base_theta_1 * l2_regularization - alpha * np.dot(ex_x.T, layer1de
    return np.array([base_theta_1, base_theta_2])


trained_tetha_1, trained_tetha_2 = gradient(x, one_hot_y, rand_tetha_1, rand_tetha_2, lam
println("TRAINED Alpha THETA SET")
trained_predictions = predict(x, trained_tetha_1, trained_tetha_2)
show_prediction_quality(trained_predictions, y)
```

```
c:\users\harwister\appdata\local\programs\python\python36\lib\site-packages
\ipykernel_launcher.py:7: RuntimeWarning: overflow encountered in exp
  import sys


TRAINED Alpha THETA SET
0: 326 out of 500 = 65.2%
1: 470 out of 500 = 94.0%
2: 195 out of 500 = 39.0%
3: 313 out of 500 = 62.6%
4: 82 out of 500 = 16.4%
5: 28 out of 500 = 5.6%
6: 334 out of 500 = 66.8%
7: 161 out of 500 = 32.2%
8: 0 out of 500 = 0.0%
9: 0 out of 500 = 0.0%
prediction quality: 38.2%
```

In [18]:

```
# task 12
# Добавьте L2-регуляризацию в процесс вычисления градиентов.
# task 14
# Обучите нейронную сеть с использованием градиентного спуска
# task 15
# Вычислите процент правильных классификаций на обучающей выборке.
# task 17
# Подберите параметр регуляризации. Как меняются изображения на скрытом слое в зависимост

trained_tetha_1, trained_tetha_2 = gradient(x, one_hot_y, rand_tetha_1, rand_tetha_2, lam
println("TRAINED L2 Regularized THETA SET")
trained_predictions = predict(x, trained_tetha_1, trained_tetha_2)
show_prediction_quality(trained_predictions, y)

trained_tetha_1, trained_tetha_2 = gradient(x, one_hot_y, rand_tetha_1, rand_tetha_2, lam
println("TRAINED L2 Regularized THETA SET")
trained_predictions = predict(x, trained_tetha_1, trained_tetha_2)
show_prediction_quality(trained_predictions, y)


hidden_layer = predict(x, trained_tetha_1, trained_tetha_2, return_hidden_layer=True)
```

```
c:\users\harwister\appdata\local\programs\python\python36\lib\site-packages
\ipykernel_launcher.py:7: RuntimeWarning: overflow encountered in exp
  import sys


TRAINED L2 Regularized THETA SET
0: 326 out of 500 = 65.2%
1: 470 out of 500 = 94.0%
2: 195 out of 500 = 39.0%
3: 313 out of 500 = 62.6%
4: 82 out of 500 = 16.4%
5: 28 out of 500 = 5.6%
6: 334 out of 500 = 66.8%
7: 161 out of 500 = 32.2%
8: 0 out of 500 = 0.0%
9: 0 out of 500 = 0.0%
prediction quality: 38.2%

TRAINED L2 Regularized THETA SET
0: 326 out of 500 = 65.2%
1: 469 out of 500 = 93.8%
2: 198 out of 500 = 39.6%
3: 315 out of 500 = 63.0%
4: 82 out of 500 = 16.4%
5: 28 out of 500 = 5.6%
6: 334 out of 500 = 66.8%
7: 161 out of 500 = 32.2%
8: 0 out of 500 = 0.0%
9: 0 out of 500 = 0.0%
prediction quality: 38.3%
```

```
TRAINED Alpha THETA SET
0: 480 out of 500 = 96.0%
1: 380 out of 500 = 76.0%
2: 408 out of 500 = 81.6%
3: 406 out of 500 = 81.2%
```

```
4: 294 out of 500 = 58.8%
5: 454 out of 500 = 90.8%
6: 404 out of 500 = 80.8%
7: 297 out of 500 = 59.4%
8: 355 out of 500 = 71.0%
9: 466 out of 500 = 93.2%
prediction quality: 78.9%

TRAINED L2 Regularized THETA SET
0: 481 out of 500 = 96.2%
1: 388 out of 500 = 77.6%
2: 413 out of 500 = 82.6%
3: 414 out of 500 = 82.8%
4: 299 out of 500 = 59.8%
5: 461 out of 500 = 92.2%
6: 416 out of 500 = 83.2%
7: 314 out of 500 = 62.8%
8: 374 out of 500 = 74.8%
9: 471 out of 500 = 94.2%
prediction quality: 80.6%
```

In [19]:

```python
# task 13
# Проверьте полученные значения градиента.
print("Regularization has a positive effect on learning")

# task 16
# Визуализируйте скрытый слой обученной сети.

plt.matshow(hidden_layer[:100].T)
plt.show()
```

```
Regularization has a positive effect on learning
```