Отчёт по лабораторной работе №4 "Нейронные сети"

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy.io import loadmat
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

1. Загрузите данные ex4data1.mat из файла.

In [2]:
```python
mat = loadmat('ex4data1.mat')
X_train, y_train = mat['X'], mat['y']
y_train = y_train.reshape(y_train.shape[0])
y_train = np.where(y_train != 10, y_train, 0)
```

2. Загрузите веса нейронной сети из файла ex4weights.mat, который содержит две матрицы Θ(1) (25, 401) и Θ(2) (10, 26). Какова структура полученной нейронной сети?

Нейронная сеть состоит из трёх слоёв: входной слой содержит 400 нейронов, выходной - 10 нейронов, один промежуточный слой - 25 нейронов.

In [3]:
```python
mat_weights = loadmat('ex4weights.mat')
theta1 = mat_weights['Theta1']
theta2 = mat_weights['Theta2']
s_L = [400, 25, 10]
```

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

In [4]:
```python
def sigmoid(z):
    return 1 / (1 + np.e ** (-z))

def insert_ones(x):
    if len(x.shape) == 1:
        return np.insert(x, 0, 1)
    return np.column_stack((np.ones(x.shape[0]), x))

def unroll(weights):
    result = np.array([])

    for theta in weights:
        result = np.concatenate((result, theta.flatten()))

    return result

def roll(weights):
    weights = np.array(weights)
    thetas = []
    left = 0

    for i in range(len(s_L) - 1):
        x, y = s_L[i + 1], s_L[i] + 1
        right = x*y
        thetas.append(weights[left:left + right].reshape(x, y))
        left = right

    return thetas
```

In [5]:
```python
def forward_prop(x, thetas, cache=False):
    cur_activation = x.copy()
    activations = [cur_activation]

    for theta_i in thetas:
        temp_a = insert_ones(cur_activation)
        z_i = theta_i.dot(temp_a.T).T
        cur_activation = sigmoid(z_i)
        if cache:
            activations.append(cur_activation)

    return activations if cache else cur_activation
```

4. Вычислите процент правильных классификаций на обучающей выборке. Сравните полученный результат с логистической регрессией.

In [6]:
```python
def accuracy(hyp, y):
    return 1 - ((np.count_nonzero(hyp.argmax(axis=1) - y) / y.shape[0]))
```

In [7]:
```python
weights = [theta1, theta2]
hypotesis = forward_prop(X_train, weights)
acc = accuracy(hypotesis, y_train)
print(f"Accuracy on training set: {acc}")
```

```
Accuracy on training set: 0.0011999999999999789
```

Как видим, точность вычислений очень низкая. Это связано с тем, что исходные веса не позволяют добиться хорошего результата. А значит нейронную сеть стоит обучить для вычисления наиболее оптимальных весов.

Заметим, что логистическая регресия давала нам точность 0.9588.

5. Перекодируйте исходные метки классов по схеме one-hot.

In [8]:
```python
def to_one_spot(y, num_classes=10):
    y_one_spot = np.zeros((y.shape[0], num_classes))

    for i, y_i in enumerate(y):
        y_one_spot[i][y_i] = 1

    return y_one_spot
```

In [10]:
```python
y_one_spot = to_one_spot(y_train)
y_train.shape, y_one_spot.shape
```

Out[10]:
```
((5000,), (5000, 10))
```

6. Реализуйте функцию стоимости для данной нейронной сети.

In [11]:
```python
ONE = 1.0 + 1e-15

def cost_func(X, y, weights):
    total_cost = 0
    K = y.shape[1]
    hyp = forward_prop(X, weights)
    for k in range(K):
        y_k, hyp_k = y[:, k], hyp[:, k]
        cost_trues =  y_k * np.log(hyp_k)
        cost_falses = (1 - y_k) * np.log(ONE - hyp_k)
        cost = cost_trues + cost_falses
        total_cost += cost
    return -total_cost.sum() / y.shape[0]
```

7. Добавьте L2-регуляризацию в функцию стоимости.

In [12]:
```python
def cost_func_regularized(X, y, weights, reg_L=1):
    weights = roll(weights)
    reg = 0
    cost = cost_func(X, y, weights)

    for theta in weights:
        theta_R = theta[:, 1:]
        reg += (theta_R ** 2).sum()

    return cost + (reg_L / 2 / y.shape[0]) * reg
```

8. Реализуйте функцию вычисления производной для функции активации.

In [13]:
```python
def activation_der(act):
    return act * (1 - act)
```

9. Инициализируйте веса небольшими случайными числами.

Зададим значения случайных весов в диапазоне [-0.01, 0.01]

In [14]:
```python
INIT_EPS = 1e-2

def initialize_weights():
    weights = []

    for i in range(len(s_L) - 1):
        theta = np.random.random((s_L[i + 1], s_L[i] + 1)) * 2 * INIT_EPS - INIT_EPS
        weights.append(theta)

    return unroll(weights)
```

In [15]:
```python
init_weights = initialize_weights()
init_weights
```

Out[15]:
```
array([-0.00223469, -0.00677362, -0.00900332, ...,  0.00248924,
        0.00782102, -0.00829766])
```

10. Реализуйте алгоритм обратного распространения ошибки для данной конфигурации сети.

In [16]:
```python
def back_prop(X, y, weights, reg_L=0):
    M = y.shape[0]
    L = len(weights)
    act = forward_prop(X, weights, cache=True)
    Deltas = [np.zeros(theta.shape) for theta in weights]

    for i in range(M):
        delta_L = y[i] - act[-1][i]
        deltas = [delta_L]

        for l in reversed(range(1, L)):
            d = np.dot(weights[l].T, deltas[-1]) * activation_der(insert_ones(act[l][i]))
            deltas.append(d[1:])

        deltas = list(reversed(deltas))
        for l in range(L):
            Deltas[l] = Deltas[l] + np.dot(deltas[l].reshape((-1,1)), insert_ones(act[l][i]).reshape
((1, -1)))

    D = []
    for l, Delta_l in enumerate(Deltas):
        D_l = Delta_l / M
        D_l[:, 1:] += reg_L * weights[l][:, 1:]
        D.append(D_l)

    return D
```

In [17]:
```python
Deltas = back_prop(X_train, y_one_spot, weights)
```

11. Для того, чтобы удостоверится в правильности вычисленных значений градиентов используйте метод проверки градиента с параметром ε = 10-4.

In [18]:
```python
GRAD_EPS = 1e-4

def check_gradient(X, y, thetas, D_vec, edge=500):
    def J(theta):
        return cost_func_regularized(X, y, theta)

    N = min(len(thetas), edge)
    grad_approx = np.zeros(N)

    for i in range(N):
        theta_plus, theta_minus = thetas.copy(), thetas.copy()
        theta_plus[i] += GRAD_EPS
        theta_minus[i] -= GRAD_EPS
        grad_approx[i] = (J(theta_plus) - J(theta_minus)) / (2 * GRAD_EPS)

    return np.allclose(grad_approx, D_vec[:N], atol=1)
```

12. Добавьте L2-регуляризацию в процесс вычисления градиентов.

Регуляризация добавлена в метод `back_prop`.

13. Проверьте полученные значения градиента.

In [19]:
```python
check_gradient(X_train, y_one_spot, unroll(weights), unroll(Deltas))
```

Out[19]:
```
True
```

14. Обучите нейронную сеть с использованием градиентного спуска или других более эффективных методов оптимизации.

In [20]:
```python
def train(X, y, reg_L, l_rate=0.5, max_steps=1e+3, with_history=False):
    history = []
    cur_weights = initialize_weights()
    cur_loss = cost_func_regularized(X, y, cur_weights, reg_L)

    cur_step = 0
    while cur_step < max_steps:
        cur_step += 1
        new_weights = update_weights(X, y, cur_weights, l_rate, reg_L)
        new_loss = cost_func_regularized(X, y, new_weights, reg_L)

        if np.isnan(new_loss):
            break

        history.append((new_weights, new_loss))
        cur_weights = new_weights
        cur_loss = new_loss

    if with_history:
        return history

    return cur_weights


def update_weights(X, y, weights, l_rate, reg_L):
    gradient = unroll(back_prop(X, y, roll(weights), reg_L))
    gradient *= l_rate
    return weights + gradient
```

In [23]:
```python
grad_weights = train(X_train, y_one_spot, reg_L=0.003, l_rate=0.5)
```

15. Вычислите процент правильных классификаций на обучающей выборке.

In [24]:
```python
hypotesis = forward_prop(X_train, roll(grad_weights))
acc = accuracy(hypotesis, y_train)
print(f"Accuracy on training set: {acc}")
```

```
Accuracy on training set: 0.9416
```

16. Визуализируйте скрытый слой обученной сети.

In [25]:
```python
def plot_hidden_layer(X, w):
    hyp = forward_prop(X, roll(w), cache=True)
    print(f"Accuracy on training set: {accuracy(hyp[-1], y_train)}")
    hidden_layer = hyp[1]

    nums = list(range(150, 5000, 250))
    size = int(np.sqrt(hidden_layer.shape[1]))
    pictures = [hidden_layer[i].reshape((size, size)) for i in nums]

    fig, axs = plt.subplots(1, 20, figsize=(20, 0.85))
    for i, ax in enumerate(axs.flatten()):
        ax.pcolor(pictures[i], cmap=cm.gray)
        ax.axis('off')

    plt.show()
```

In [26]:
```python
plot_hidden_layer(X_train, grad_weights)
```

```
Accuracy on training set: 0.9416
```

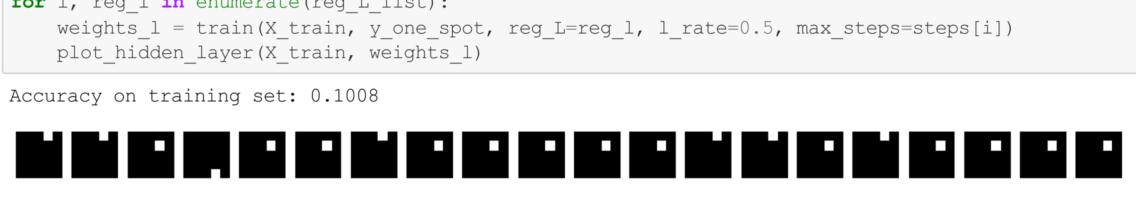

На изображениях представлено по два примера для каждого класса.

17. Подберите параметр регуляризации. Как меняются изображения на скрытом слое в зависимости от данного параметра?

In [27]:
```python
reg_L_list = [1, 0.3, 0.1, 0.03, 0.01, 0.003]
steps = [20, 50, 75, 400, 600, 1000]

for i, reg_l in enumerate(reg_L_list):
    weights_l = train(X_train, y_one_spot, reg_L=reg_l, l_rate=0.5, max_steps=steps[i])
    plot_hidden_layer(X_train, weights_l)
```

```
Accuracy on training set: 0.1008
```



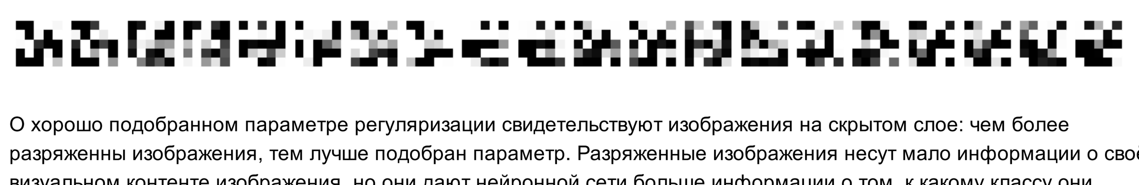
```
Accuracy on training set: 0.10019999999999996
```

```
Accuracy on training set: 0.4052
```

```
Accuracy on training set: 0.8568
```

```
Accuracy on training set: 0.9062
```

```
Accuracy on training set: 0.9412
```



О хорошо подобранном параметре регуляризации свидетельствуют изображения на скрытом слое: чем более разряженны изображения, тем лучше подобран параметр. Разряженные изображения несут мало информации о своём визуальном контенте изображения, но они дают нейронной сети больше информации о том, к какому классу они принадлежат. Также высокая точность нейронной сети свидетельствует о хорошо подобранном параметре регуляризации.