Обучение нейронной сети для распознавания рукописных цифр на изображениях из базы MNIST

Импорт основных модулей

- Импортируется стандартный класс **numpy** для работы с векторами и матрицами
- Импортируется стандартный класс **time** для замера времени работы скриптов
- Выставляется путь к папке с кодом
- Импортируется класс **Img** для хранения и обработки изображения
- Импортируется класс **Imgs** для хранения массива всех изображений
- Импортируется класс **LayerFC**, представляющий один полносвязный слой нейронной сети
- Импортируется класс **NeuralNetwork**, представляющий нейронную сеть и функцию **nn_load** для загрузки параметров сохраненной на диске сети

In [2]:
```
import numpy as np
import time
import sys
sys.path.append('./../')

from cap_solver.image.img import Img
from cap_solver.image.imgs import Imgs
from cap_solver.neural_network.layer_fc import LayerFC
from cap_solver.neural_network.neural_network import NeuralNetwork,
nn_load
```

Функция, переводящая символ ($0, 1, \ldots, 9$ ***) на изображении в вектор длины*** 10

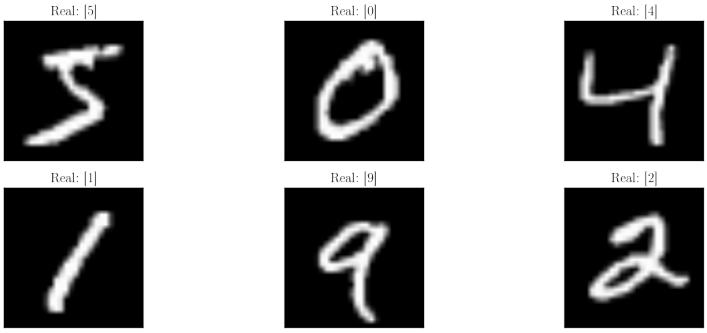
In [3]:
```
def fsmb2vec(s):
    ''' Symbol may contain 0-9 numbers. '''
    v = np.zeros((10, 1))
    v[int(s)] = 1.
    return v
```

Загрузка изображений

- Инициализация класса, хранящего изображения и выставление опции печати промежуточной информации
- Распаковка изображений из архива (база mnist из 70000 распознанных рукописных цифр)
- Отрисовка шести первых изображений для примера

In [4]:
```
Ims = Imgs(fsmb2vec=fsmb2vec, verb=True)
Ims.load('./data/mnist/mnist.pkl.gz', dtype='mnist')
Ims.show(n=6, c=3, figsize=(13, 5), figsize_sub=(4, 4))
```

```
Total time   (sec.):    3.9
Time per img (sec.):    0.0001
Total number of img:     70000
```

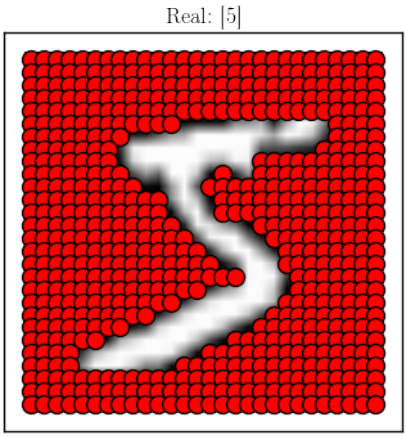


Анализ изображений (на примере первого изображения в базе)

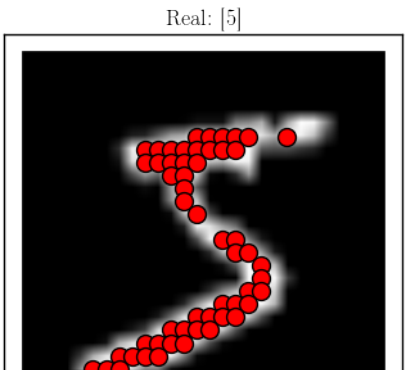
- Вычисление и отображение (в виде списка из пар: номер цвета - количество пикселей) гистограммы цветов (функция calc_hist)
- Явное отображение (в виде красных кругов) первых двух по распространенности на изображении цветов (функция show_colors)

In [5]:
```
hist = Ims[0].calc_hist(present=True)
for h in hist[:2]:
    print 'Color: %-3d | Count: %-d'%tuple(h)
    Ims[0].show_colors(colors=h, figsize=(4, 4))
```

```
C:  0->   618       | C:253->    54       | C: 18->     5       | C:
2->     3       | C: 11->     3       |
C:154->     3       | C:  1->     2       | C:136->     2       | C:
16->     2       | C: 39->     2       |
C:172->     2       | C:182->     2       | C:190->     2       | C:
64->     2       | C:195->     2       |
C:198->     2       | C: 80->     2       | C: 81->     2       | C:
82->     2       | C:219->     2       |
C: 93->     2       | C:225->     2       | C:241->     2       | C:2
47->     2       | C:249->     2       |
C:  3->     1       | C:132->     1       | C:133->     1       | C:1
35->     1       | C:  9->     1       |
C:130->     1       | C: 14->     1       | C:148->     1       | C:1
71->     1       | C:150->     1       |
C: 23->     1       | C: 24->     1       | C: 25->     1       | C:
26->     1       | C: 27->     1       |
C:156->     1       | C: 30->     1       | C:160->     1       | C:
35->     1       | C: 36->     1       |
C: 70->     1       | C:166->     1       | C:221->     1       | C:1
70->     1       | C: 43->     1       |
C: 45->     1       | C: 46->     1       | C:175->     1       | C:
49->     1       | C:183->     1       |
C: 56->     1       | C:186->     1       | C:187->     1       | C:
66->     1       | C:201->     1       |
C:119->     1       | C:205->     1       | C: 78->     1       | C:2
07->     1       | C:139->     1       |
C:213->     1       | C: 55->     1       | C: 90->     1       | C:
94->     1       | C:226->     1       |
C:251->     1       | C:212->     1       | C:229->     1       | C:1
07->     1       | C:108->     1       |
C:238->     1       | C:240->     1       | C:242->     1       | C:2
44->     1       | C:114->     1       |
C:250->     1       | C:255->     1       | C:252->     1       | C:1
26->     1       | C:127->     1       |
Color: 0   | Count: 618
```




```
Color: 253 | Count: 54
```

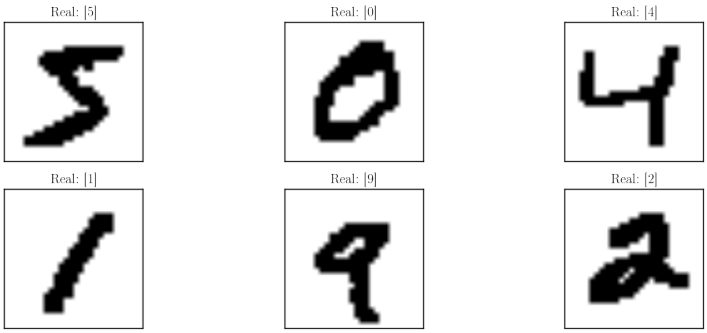


Бинаризация изображений

- На основе анализа гистограммы, наиболее представленный на изображении цвет считаем фоновым
- Все остальные цвета считаем соответствующими тексту
- Бинаризация: переводим цвет фона в нуль, а цвет текста в единицу (функция binarization)
- Отрисовка нескольких преобразованных изображений для примера

In [6]:
```
_t = time.time()
for Im in Ims:
    Im.binarization(colors_bg=[0])
print 'Total time   (sec.): %6.1f'%(time.time() - _t)
print 'Time per img (sec.): %9.4f'%((time.time() - _t)/Ims.len_)
Ims.show(n=6, r=2, figsize=(13, 5), figsize_sub=(4, 4))
```

```
Total time   (sec.):  123.5
Time per img (sec.):    0.0018
```



Подготовка данных для обучения сети

- Инициализация класса, представляющего данные
- Преобразование подготовленных изображений в матрицу входных и выходных векторов
- Разделение изображений на набор обучающих данных (50000), валидационных данных (0) и тестовых данных (20000)

In [7]:
```
_t = time.time()
X_all = Ims.get_matrix(var='arr')
Y_all = Ims.get_matrix(var='smb_real')
print X_all.shape
print Y_all.shape
N_trn = 50000
X_trn, Y_trn = X_all[:, :N_trn], Y_all[:, :N_trn]
X_tst, Y_tst = X_all[:, N_trn:], Y_all[:, N_trn:]
print 'Total time   (sec.): %6.1f'%(time.time() - _t)
print 'Time per img (sec.): %9.4f'%((time.time() - _t)/Ims.len_)
```

```
(784, 70000)
(10, 70000)
Total time   (sec.):    1.9
Time per img (sec.):    0.0000
```

Создание нейронной сети

- Инициализируется экземпляр класса нейронной сети и выставляется опция печати промежуточной информации
- Создаются три слоя с числами нейронов 784 (входной слой), 30 (внутренний слой), 10 (выходной слой)
- Гиперпараметр числа элементов в подвыборке выбран равным 10, а гиперпараметр скорости обучения выбран равным 3 (на основе ряда текстых расчетов по выбору оптимального значения гиперпараметров)

In [8]:
```
NN = NeuralNetwork(verb=True)
NN.add_layers(LayerFC(None, 784))
NN.add_layers(LayerFC(784 ,  30))
NN.add_layers(LayerFC( 30 ,  10))
NN.set_params(mb_size=10, eta=3.)
```

Обучение нейронной сети

- Выбрано 30 эпох для обучения
- Выходная информация на каждой эпохе обучения:
  - Номер эпохи
  - Время в секундах, затраченное на эпоху
  - Полное количество итераций обратного распространения в сети (с момента создания сети)
  - Количество данных в тестовом наборе
  - Качество сети после соотвествующей эпохи обучения на тестовом наборе данных (относительная доля неправильных результатов предсказания)

In [9]:
```
NN.learning(X_trn, Y_trn, X_tst, Y_tst, epochs=30)
```

```
Epoch #  1: T=    9.71; m=    50000; n_check=    20000; e_check=0.
106050
Epoch #  2: T=    7.81; m=   100000; n_check=    20000; e_check=0.
090200
Epoch #  3: T=    7.90; m=   150000; n_check=    20000; e_check=0.
083650
Epoch #  4: T=    8.70; m=   200000; n_check=    20000; e_check=0.
077900
Epoch #  5: T=    6.44; m=   250000; n_check=    20000; e_check=0.
074450
Epoch #  6: T=    6.51; m=   300000; n_check=    20000; e_check=0.
065950
Epoch #  7: T=    6.76; m=   350000; n_check=    20000; e_check=0.
067500
Epoch #  8: T=    7.67; m=   400000; n_check=    20000; e_check=0.
066400
Epoch #  9: T=    7.30; m=   450000; n_check=    20000; e_check=0.
063100

Epoch # 10: T=    7.39; m=   500000; n_check=    20000; e_check=0.
062700
Epoch # 11: T=    6.64; m=   550000; n_check=    20000; e_check=0.
065500
Epoch # 12: T=    6.99; m=   600000; n_check=    20000; e_check=0.
063800
Epoch # 13: T=    6.80; m=   650000; n_check=    20000; e_check=0.
062500
Epoch # 14: T=    8.26; m=   700000; n_check=    20000; e_check=0.
065350
Epoch # 15: T=    8.02; m=   750000; n_check=    20000; e_check=0.
062950
Epoch # 16: T=    7.40; m=   800000; n_check=    20000; e_check=0.
061300
Epoch # 17: T=    7.95; m=   850000; n_check=    20000; e_check=0.
062650
Epoch # 18: T=    6.88; m=   900000; n_check=    20000; e_check=0.
059500
Epoch # 19: T=    7.72; m=   950000; n_check=    20000; e_check=0.
059800
Epoch # 20: T=    6.94; m=  1000000; n_check=    20000; e_check=0.
060900
Epoch # 21: T=    7.96; m=  1050000; n_check=    20000; e_check=0.
059500
Epoch # 22: T=    8.90; m=  1100000; n_check=    20000; e_check=0.
058100
Epoch # 23: T=    8.78; m=  1150000; n_check=    20000; e_check=0.
058900
Epoch # 24: T=    7.82; m=  1200000; n_check=    20000; e_check=0.
062650
Epoch # 25: T=    9.76; m=  1250000; n_check=    20000; e_check=0.
061800
Epoch # 26: T=    8.44; m=  1300000; n_check=    20000; e_check=0.
059200
Epoch # 27: T=    6.83; m=  1350000; n_check=    20000; e_check=0.
061000
Epoch # 28: T=    6.27; m=  1400000; n_check=    20000; e_check=0.
062250
```

```
Epoch # 29: T=    8.46; m=  1450000; n_check=    20000; e_check=0.
060150
Epoch # 30: T=    9.92; m=  1500000; n_check=    20000; e_check=0.
058450
```

Комментарий: как следует из результатов, нейронная сеть после 30 эпох обучения дает менее 6% ошибок на тестовой выборке размера 20000 (качество предсказания может быть существенно улучшено путем лучшего выбора гиперпараметров сети).

Добавим еще один слой из 30 нейронов к сети и соответствующим образом модифицируем параметр скорости обучения и максимальное число эпох обучения

In [20]:
```
NN = NeuralNetwork(verb=True)
NN.add_layers(LayerFC(None, 784))
NN.add_layers(LayerFC(784 ,  30))
NN.add_layers(LayerFC( 30 ,  30))
NN.add_layers(LayerFC( 30 ,  10))
NN.set_params(mb_size=10, eta=1.5)
NN.learning(X_trn, Y_trn, X_tst, Y_tst, epochs=50)
```

```
Epoch #  1: T=    9.30; m=    50000; n_check=    20000; e_check=0.
122550
Epoch #  2: T=   12.27; m=   100000; n_check=    20000; e_check=0.
096000
Epoch #  3: T=    8.18; m=   150000; n_check=    20000; e_check=0.
091200
Epoch #  4: T=    7.97; m=   200000; n_check=    20000; e_check=0.
079100
Epoch #  5: T=    7.66; m=   250000; n_check=    20000; e_check=0.
076500
Epoch #  6: T=   10.33; m=   300000; n_check=    20000; e_check=0.
072750
Epoch #  7: T=    8.65; m=   350000; n_check=    20000; e_check=0.
072650
Epoch #  8: T=    7.59; m=   400000; n_check=    20000; e_check=0.
071700
Epoch #  9: T=   10.79; m=   450000; n_check=    20000; e_check=0.
071350
Epoch # 10: T=    7.56; m=   500000; n_check=    20000; e_check=0.
066400
Epoch # 11: T=    7.86; m=   550000; n_check=    20000; e_check=0.
067650
Epoch # 12: T=    7.54; m=   600000; n_check=    20000; e_check=0.
064750
Epoch # 13: T=    7.54; m=   650000; n_check=    20000; e_check=0.
061650
Epoch # 14: T=    7.74; m=   700000; n_check=    20000; e_check=0.
060700
Epoch # 15: T=    9.88; m=   750000; n_check=    20000; e_check=0.
061250
Epoch # 16: T=    9.33; m=   800000; n_check=    20000; e_check=0.
061150
Epoch # 17: T=    7.73; m=   850000; n_check=    20000; e_check=0.
061400
Epoch # 18: T=    7.69; m=   900000; n_check=    20000; e_check=0.
061150
Epoch # 19: T=    7.68; m=   950000; n_check=    20000; e_check=0.
060800
Epoch # 20: T=    7.54; m=  1000000; n_check=    20000; e_check=0.
059900
Epoch # 21: T=    7.55; m=  1050000; n_check=    20000; e_check=0.
061000
Epoch # 22: T=    7.55; m=  1100000; n_check=    20000; e_check=0.
062400
Epoch # 23: T=    7.55; m=  1150000; n_check=    20000; e_check=0.
058700
Epoch # 24: T=    7.58; m=  1200000; n_check=    20000; e_check=0.
059700
Epoch # 25: T=    9.09; m=  1250000; n_check=    20000; e_check=0.
056950
Epoch # 26: T=    8.40; m=  1300000; n_check=    20000; e_check=0.
057300
Epoch # 27: T=    7.85; m=  1350000; n_check=    20000; e_check=0.
058850
Epoch # 28: T=    7.52; m=  1400000; n_check=    20000; e_check=0.
057650
Epoch # 29: T=    7.54; m=  1450000; n_check=    20000; e_check=0.
058600
Epoch # 30: T=    7.78; m=  1500000; n_check=    20000; e_check=0.
058100
Epoch # 31: T=    7.56; m=  1550000; n_check=    20000; e_check=0.
059650
Epoch # 32: T=    8.04; m=  1600000; n_check=    20000; e_check=0.
057700
Epoch # 33: T=    8.95; m=  1650000; n_check=    20000; e_check=0.
057150
Epoch # 34: T=    9.23; m=  1700000; n_check=    20000; e_check=0.
056550
```

```
Epoch # 35: T=    9.57; m=  1750000; n_check=    20000; e_check=0.
056550
Epoch # 36: T=    8.76; m=  1800000; n_check=    20000; e_check=0.
056750
Epoch # 37: T=    9.53; m=  1850000; n_check=    20000; e_check=0.
056600
Epoch # 38: T=    8.09; m=  1900000; n_check=    20000; e_check=0.
059150
Epoch # 39: T=    8.55; m=  1950000; n_check=    20000; e_check=0.
058650
Epoch # 40: T=    8.83; m=  2000000; n_check=    20000; e_check=0.
057800
Epoch # 41: T=    9.11; m=  2050000; n_check=    20000; e_check=0.
057650
Epoch # 42: T=   10.49; m=  2100000; n_check=    20000; e_check=0.
056900
Epoch # 43: T=   11.11; m=  2150000; n_check=    20000; e_check=0.
055750
Epoch # 44: T=   11.01; m=  2200000; n_check=    20000; e_check=0.
056300
Epoch # 45: T=   10.49; m=  2250000; n_check=    20000; e_check=0.
055550
Epoch # 46: T=   10.89; m=  2300000; n_check=    20000; e_check=0.
057500
Epoch # 47: T=   10.48; m=  2350000; n_check=    20000; e_check=0.
057800
Epoch # 48: T=   10.40; m=  2400000; n_check=    20000; e_check=0.
056350
Epoch # 49: T=   10.05; m=  2450000; n_check=    20000; e_check=0.
055850
Epoch # 50: T=   10.56; m=  2500000; n_check=    20000; e_check=0.
057650
```

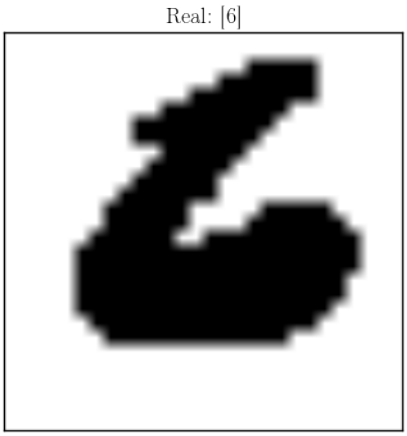
Комментарий: как видим, точность существенно не увеличилась.

Проверка качества предсказания на конкретном примере

- Выбираем одно изображение (последнее изображение из набора) для примера и получаем соответствующие интенсивности пикселей (x)
- Запускаем расчет посредством обученной нейронной сети и выводим результат (предсказание сети)
- Отрисовываем изображение

In [13]:
```
Im = Ims[-1]
x = Im.get_vector('arr')
a = NN.forward(x)
print 'The answer of the neural network is ', np.argmax(a)
Im.show(figsize=(4, 4))
```

```
The answer of the neural network is  6
```



Комментарий: обученная нейронная сеть выдала правильное предсказание (цифра 6).

Сохраняем обученную нейронную сеть в файл для возможности последующего использования

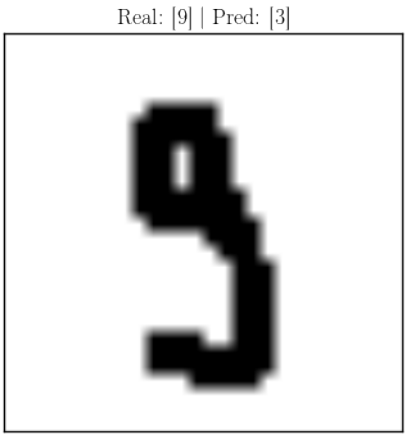
In [14]:
```
NN.save('./nn_saved/nn_mnist.p')
```

Загружаем (для проверки корректности опции сохранения) обученную нейронную сеть из файла и демонстрируем первый ошибочный результат распознавания

In [21]:
```
NN = nn_load('./nn_saved/nn_mnist.p')

for i, Im in enumerate(Ims):
    x = Im.get_vector('arr')
    a = NN.forward(x)
    a = np.argmax(a)
    Im.smb_calc = unicode(a)
    if not Im.symb_is_corr:
        print 'NN result is "%s" (real value is "%s"). Image number
is %d'%(Im.smb_calc, Im.smb_real, i+1)
        break
Im.show(figsize=(4, 4))
```

```
NN result is "3" (real value is "9"). Image number is 49
```



In [ ]: