CI P4 – Neural Networks

March 31, 2017

1 Neural networks in Python using Scikit Learn

- We will be using an implementation of Multilayer Perceptron back-ported from the next version of scikit-learn
- Documentation: http://scikit-learn.org/dev/modules/neural_networks_supervised.html
- This implementation consists of two classes:
- `MLPRegressor` for regression
- `MLPClassifier` for classification

```
In [1]: from sklearn.neural_network import MLPRegressor, MLPClassifier
```

1.1 Regression with neural networks

```
In [2]: # In the assignment, the data loading function is provided
        # Here, we use the example dataset that comes with scikit learn
        from sklearn.datasets import load_boston
        boston = load_boston()
        print(boston.data.shape)
        print(boston.target.shape)
```

```
(506, 13)
(506,)
```

```
In [3]: print(boston.DESCR)
```

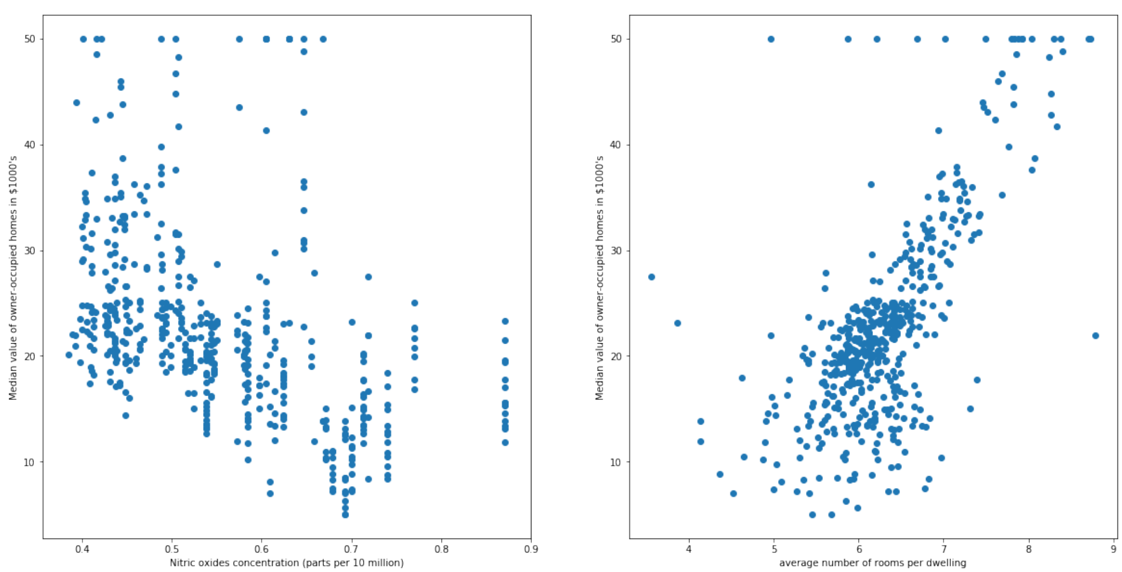
```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target
```

```
    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 other
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing


This dataset was taken from the StatLib library which is maintained at Carnegie Mel

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that addr
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data a
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proce
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
```

In [4]:
```python
%matplotlib inline
import matplotlib.pyplot as plt

#Feature order: 'CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT
```

```
plt.figure(figsize=(20,10))

nox_concentrations = boston.data[:, 4]
house_prices = boston.target

ax = plt.subplot(121)
ax.scatter(nox_concentrations, house_prices)
ax.set_xlabel("Nitric oxides concentration (parts per 10 million)")
ax.set_ylabel("Median value of owner-occupied homes in $1000's")

rooms_per_dwelling = boston.data[:, 5]
house_prices = boston.target

ax = plt.subplot(122)
ax.scatter(rooms_per_dwelling, house_prices)
ax.set_xlabel("average number of rooms per dwelling")
ax.set_ylabel("Median value of owner-occupied homes in $1000's")
```

```
Out[4]: <matplotlib.text.Text at 0x7fb4bcde7668>
```




```
In [5]: # In the assignment, the data is already split up
        from sklearn.model_selection import train_test_split
        import numpy as np

        X = np.array([rooms_per_dwelling, nox_concentrations]).T
        y = house_prices
        print("Dataset shape (X, y)      :", X.shape, y.shape)
        ## Split the data into a testing and training set (20% of the data for test
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
print("Training set shape (X, y):", X_train.shape, y_train.shape)
```

```
Dataset shape (X, y)      : (506, 2) (506,)
Training set shape (X, y): (404, 2) (404,)
```

```
In [6]: ## Initialize the neural network
        n_hidden_neurons = 20
        nn = MLPRegressor(activation='logistic', solver='lbfgs', hidden_layer_sizes
        nn  # Important parameters -- activation, algorithm, alpha, hidden_layer_si
```

```
Out[6]: MLPRegressor(activation='logistic', alpha=0.0001, batch_size='auto',
               beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
               hidden_layer_sizes=(20,), learning_rate='constant',
               learning_rate_init=0.001, max_iter=200, momentum=0.9,
               nesterovs_momentum=True, power_t=0.5, random_state=None,
               shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
               verbose=False, warm_start=False)
```

```
In [7]: ## Train the network
        nn.fit(X_train, y_train)
```

```
Out[7]: MLPRegressor(activation='logistic', alpha=0.0001, batch_size='auto',
               beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
               hidden_layer_sizes=(20,), learning_rate='constant',
               learning_rate_init=0.001, max_iter=200, momentum=0.9,
               nesterovs_momentum=True, power_t=0.5, random_state=None,
               shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
               verbose=False, warm_start=False)
```

```
In [8]: ## Calculate and print the MSE
        from sklearn.metrics import mean_squared_error
        train_mse = mean_squared_error(y_train, nn.predict(X_train))
        test_mse = mean_squared_error(y_test, nn.predict(X_test))
        print("Training MSE:", train_mse)
        print("Testing MSE: ", test_mse)
```

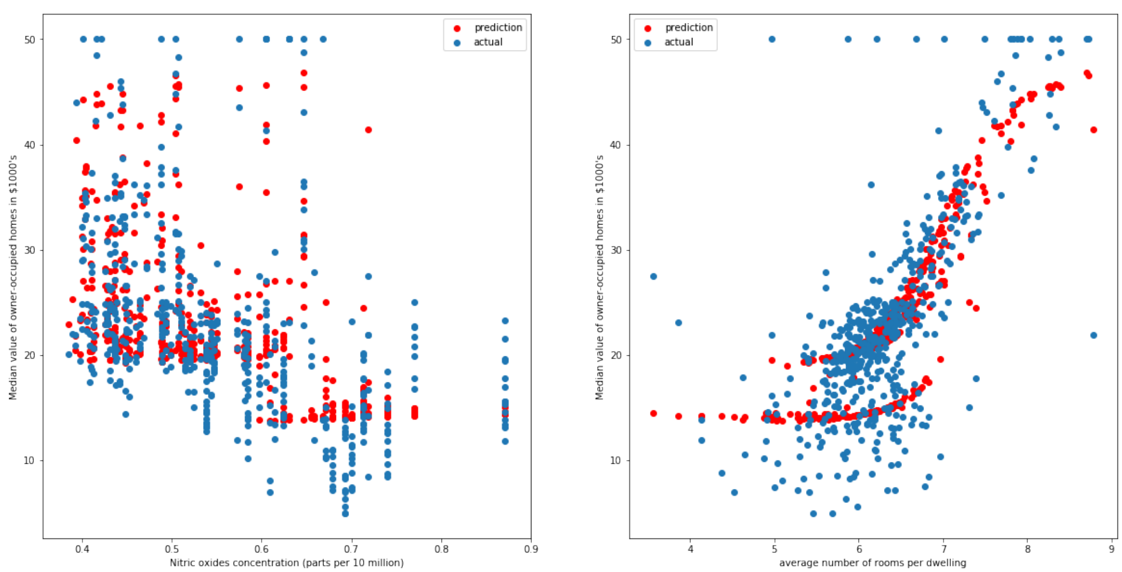
```
Training MSE: 27.7628351225
Testing MSE:  22.2361742609
```

```
In [9]: ## Predict the house prices for the entire data set
        predictions = nn.predict(X)
```

```
In [10]: ## Plot network predictions and actual values
         plt.figure(figsize=(20,10))

         house_prices_prediction = predictions
```

```
house_prices_actual = boston.target

nox_concentrations = boston.data[:, 4]

ax = plt.subplot(121)
ax.scatter(nox_concentrations, house_prices_prediction, color='r', label='
ax.scatter(nox_concentrations, house_prices_actual, label='actual')
ax.set_xlabel("Nitric oxides concentration (parts per 10 million)")
ax.set_ylabel("Median value of owner-occupied homes in $1000's")
plt.legend()

rooms_per_dwelling = boston.data[:, 5]

ax = plt.subplot(122)
ax.scatter(rooms_per_dwelling, house_prices_prediction, color='r', label='
ax.scatter(rooms_per_dwelling, house_prices_actual, label='actual')

ax.set_xlabel("average number of rooms per dwelling")
ax.set_ylabel("Median value of owner-occupied homes in $1000's")
plt.legend()
```

```
Out[10]: <matplotlib.legend.Legend at 0x7fb4acf683c8>
```



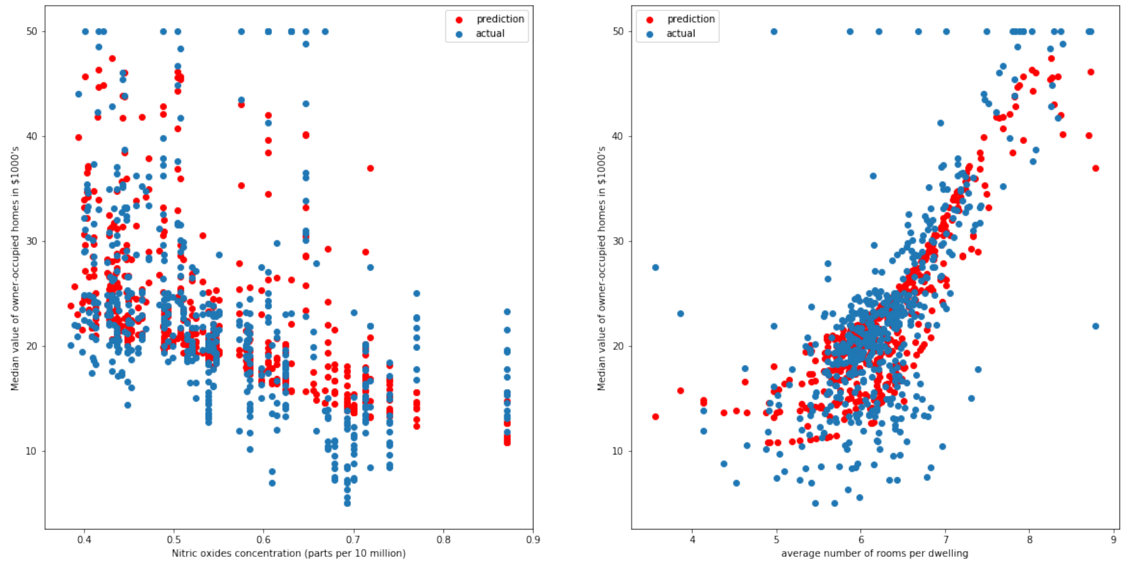
1.1.1 Experiment with parameters

```
In [11]: ## Initialize the neural network
         n_hidden_neurons = 20
         nn = MLPRegressor(activation='logistic', solver='lbfgs',
                           hidden_layer_sizes=(n_hidden_neurons,), random_state=1)
```

```
## Train the network
nn.fit(X_train, y_train)

## Calculate and print the MSE
from sklearn.metrics import mean_squared_error
train_mse = mean_squared_error(y_train, nn.predict(X_train))
test_mse = mean_squared_error(y_test, nn.predict(X_test))
print("Training MSE:", train_mse)
print("Testing MSE: ", test_mse)

## Predict the house prices for the entire data set
predictions = nn.predict(X)

## Plot network predictions and actual values
plt.figure(figsize=(20,10))

house_prices_prediction = predictions
house_prices_actual = boston.target

nox_concentrations = boston.data[:, 4]

ax = plt.subplot(121)
ax.scatter(nox_concentrations, house_prices_prediction, color='r', label='
ax.scatter(nox_concentrations, house_prices_actual, label='actual')
ax.set_xlabel("Nitric oxides concentration (parts per 10 million)")
ax.set_ylabel("Median value of owner-occupied homes in $1000's")
plt.legend()

rooms_per_dwelling = boston.data[:, 5]

ax = plt.subplot(122)
ax.scatter(rooms_per_dwelling, house_prices_prediction, color='r', label='
ax.scatter(rooms_per_dwelling, house_prices_actual, label='actual')

ax.set_xlabel("average number of rooms per dwelling")
ax.set_ylabel("Median value of owner-occupied homes in $1000's")
plt.legend()
```

```
Training MSE: 29.8548485819
Testing MSE:  26.4834499233
```

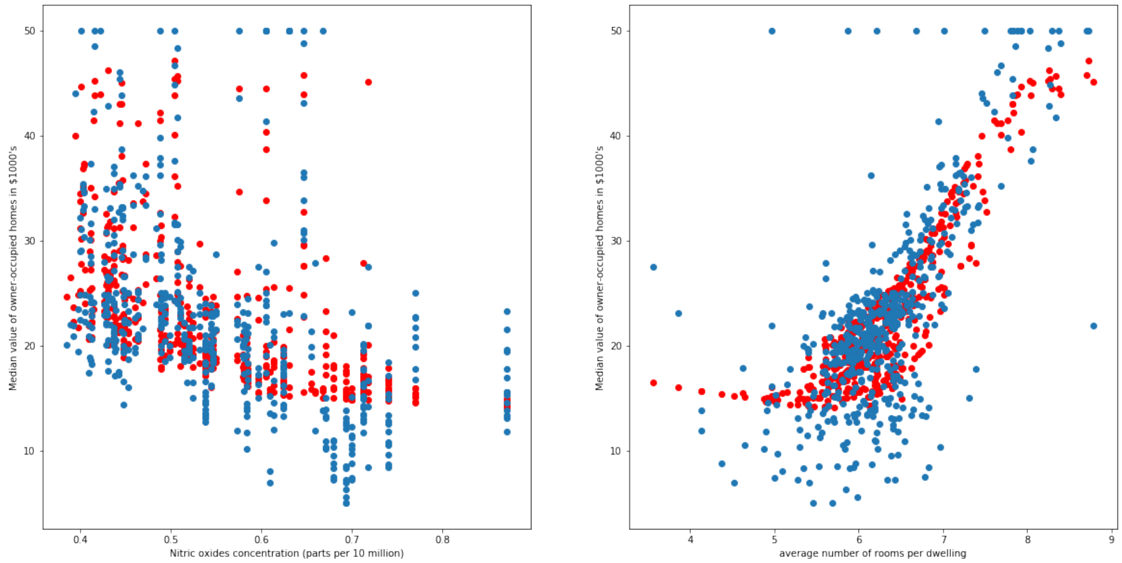
```
Out[11]: <matplotlib.legend.Legend at 0x7fb4acf41b38>
```



1.2 Warm start

In [12]:
```
from IPython import display
from sklearn.metrics import mean_squared_error,log_loss,accuracy_score


## Initialize the neural network
n_hidden_neurons = 20
max_iterations = 2000
nn = MLPRegressor(activation='logistic', solver='lbfgs',
                      hidden_layer_sizes=(n_hidden_neurons,), random_state

fig = plt.figure(figsize=(20,10))
for i in range(max_iterations):
    ## Train the network
    nn.fit(X_train, y_train)

    if i % 50 == 0 or i < 10:

        ## Calculate and print the MSE

        train_mse = mean_squared_error(y_train, nn.predict(X_train))
        test_mse = mean_squared_error(y_test, nn.predict(X_test))
        print("Iteration:", i)
        print("Training MSE:", train_mse)
        print("Testing MSE: ", test_mse)

        ## Predict the house prices for the entire data set
```

```
            predictions = nn.predict(X)

            ## Plot network predictions and actual values
            display.clear_output(wait=True)

            house_prices_prediction = predictions
            house_prices_actual = boston.target

            nox_concentrations = boston.data[:, 4]

            ax = plt.subplot(121)
            plt.gca().cla()
            ax.scatter(nox_concentrations, house_prices_prediction, color='r',
            ax.scatter(nox_concentrations, house_prices_actual, label='actual'
            ax.set_xlabel("Nitric oxides concentration (parts per 10 million)"
            ax.set_ylabel("Median value of owner-occupied homes in $1000's")

            rooms_per_dwelling = boston.data[:, 5]

            ax = plt.subplot(122)
            plt.gca().cla()
            ax.scatter(rooms_per_dwelling, house_prices_prediction, color='r',
            ax.scatter(rooms_per_dwelling, house_prices_actual, label='actual'

            ax.set_xlabel("average number of rooms per dwelling")
            ax.set_ylabel("Median value of owner-occupied homes in $1000's")

            display.display(plt.gcf())
    display.clear_output(wait=True)
```



1.3 Classification with Neural Networks

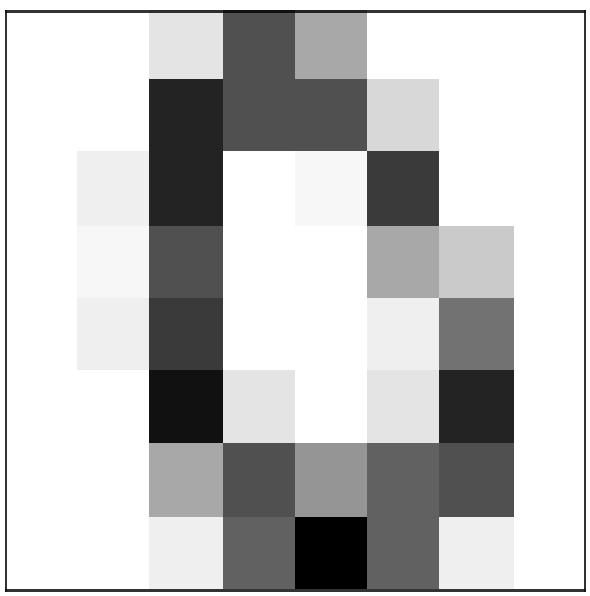
In [13]:
```
from sklearn.datasets import load_digits
digits = load_digits()

# from sklearn.datasets import fetch_mldata
# digits = fetch_mldata('MNIST original')

print(digits.data.shape)
digits.data[5].shape
```

```
(1797, 64)
```

Out[13]:
```
(64,)
```

In [14]:
```
## Show a random digit
IMAGE_DIM = (8, 8)
plt.figure()
plt.imshow(digits.data[np.random.randint(1797)].reshape(*IMAGE_DIM), inter
plt.xticks([])
plt.yticks([])
```

Out[14]:
```
([], <a list of 0 Text yticklabel objects>)
```



In [15]:
```
from sklearn.model_selection import train_test_split
import numpy as np
```

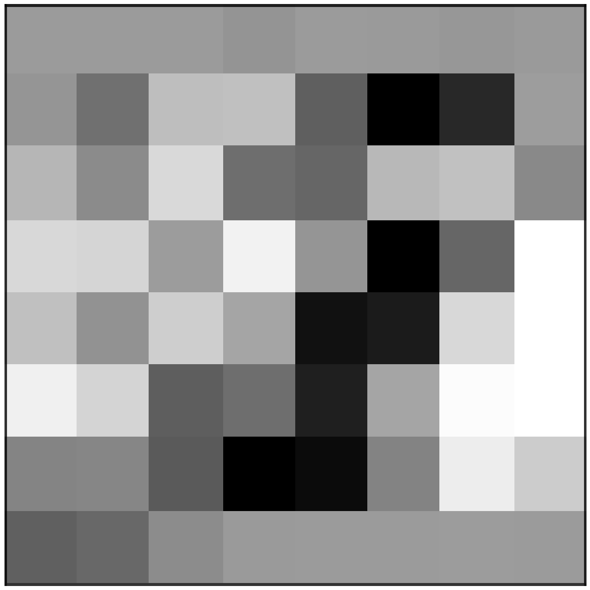
```
X = digits.data
y = digits.target
print("Dataset shape (X, y)      :", X.shape, y.shape)
## Split the data into a testing and training set (20% of the data for tes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
print("Training set shape (X, y):", X_train.shape, y_train.shape)
```

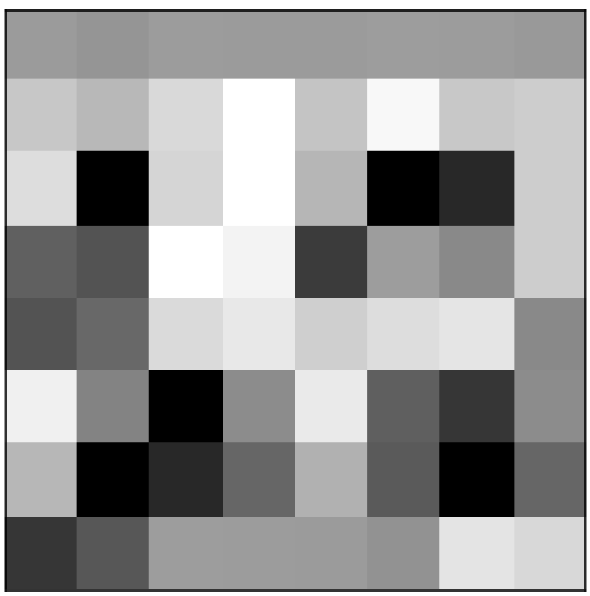
```
Dataset shape (X, y)      : (1797, 64) (1797,)
Training set shape (X, y): (1437, 64) (1437,)
```

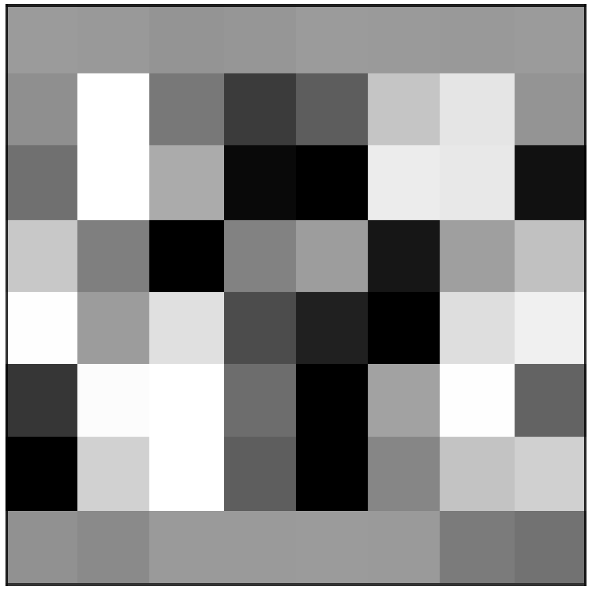
In [16]:
```
## Initialize the neural network
nn = MLPClassifier(activation='logistic', solver='adam',hidden_layer_sizes

## Train the network
nn.fit(X_train, y_train)

## Calculate and print the MSE
from sklearn.metrics import mean_squared_error, accuracy_score
train_mse = mean_squared_error(y_train, nn.predict(X_train))
test_mse = mean_squared_error(y_test, nn.predict(X_test))
print("Training MSE:", train_mse)
print("Testing MSE: ", test_mse)

test_accuracy = accuracy_score(y_test, nn.predict(X_test))
print("Test accuracy: ", test_accuracy)
```

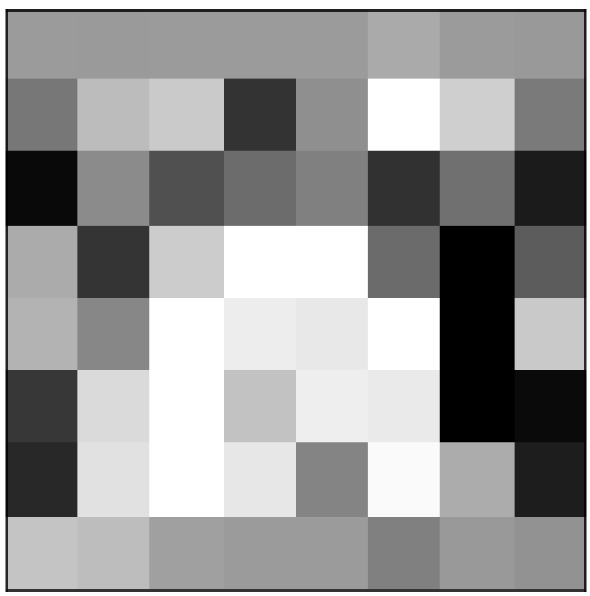
```
Training MSE: 0.310368823939
Testing MSE:  0.177777777778
Test accuracy:  0.980555555556
```

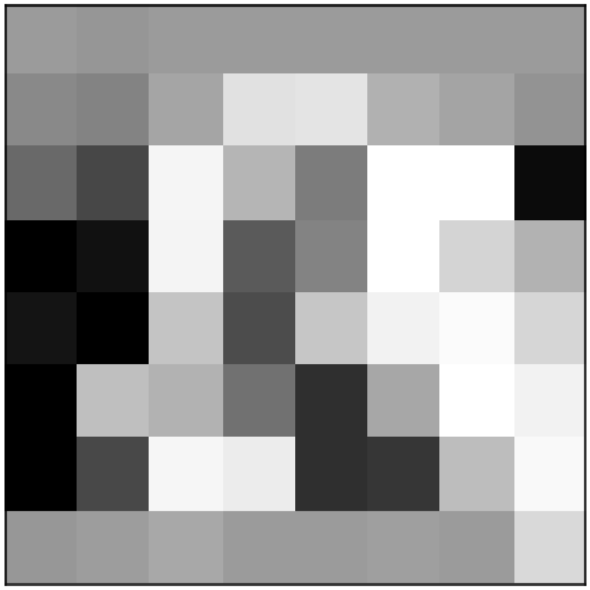
In [17]:
```
hidden_layer_weights = nn.coefs_[0]
for hidden_neuron_num in range(hidden_layer_weights.shape[1])[:10]:
        plt.figure()
        vmin, vmax = hidden_layer_weights.min(), hidden_layer_weights.max(
        plt.imshow(hidden_layer_weights[:, hidden_neuron_num].reshape(*IMA
                   vmin=.5 * vmin, vmax=.5 * vmax, interpolation='none')
        plt.xticks(())
        plt.yticks(())
plt.close()
```

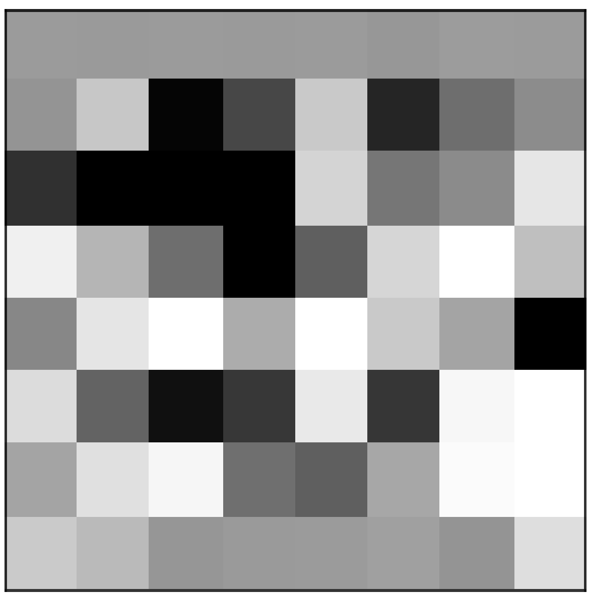


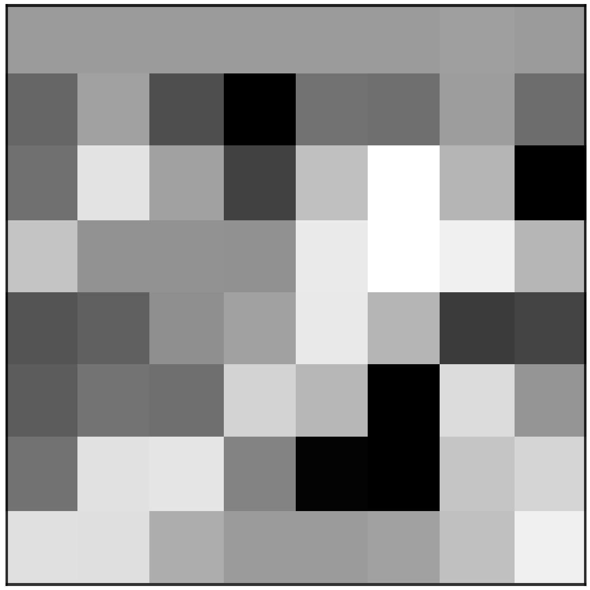


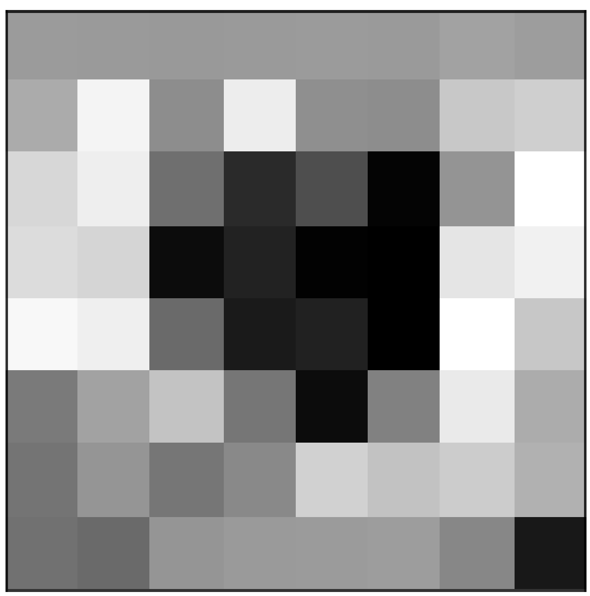


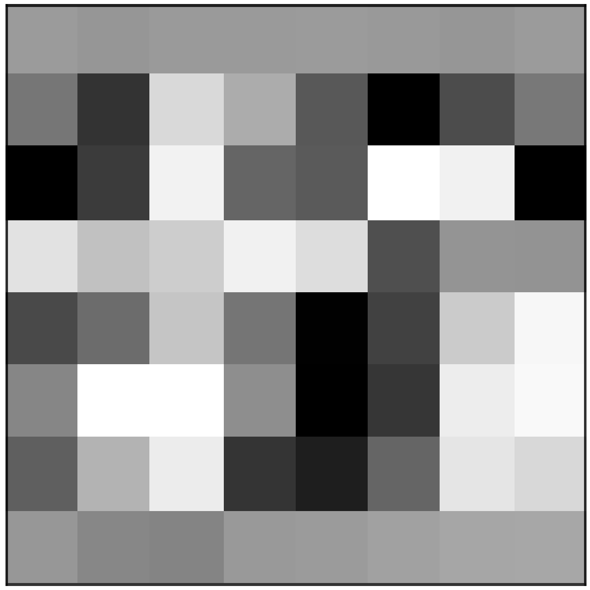












Experiment with values of alpha and n_hidden_neurons