Applied Machine Learning

Mini Project – Neural Network

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

PyTorch tutorial: https://pytorch.org/tutorials/

Neural Network

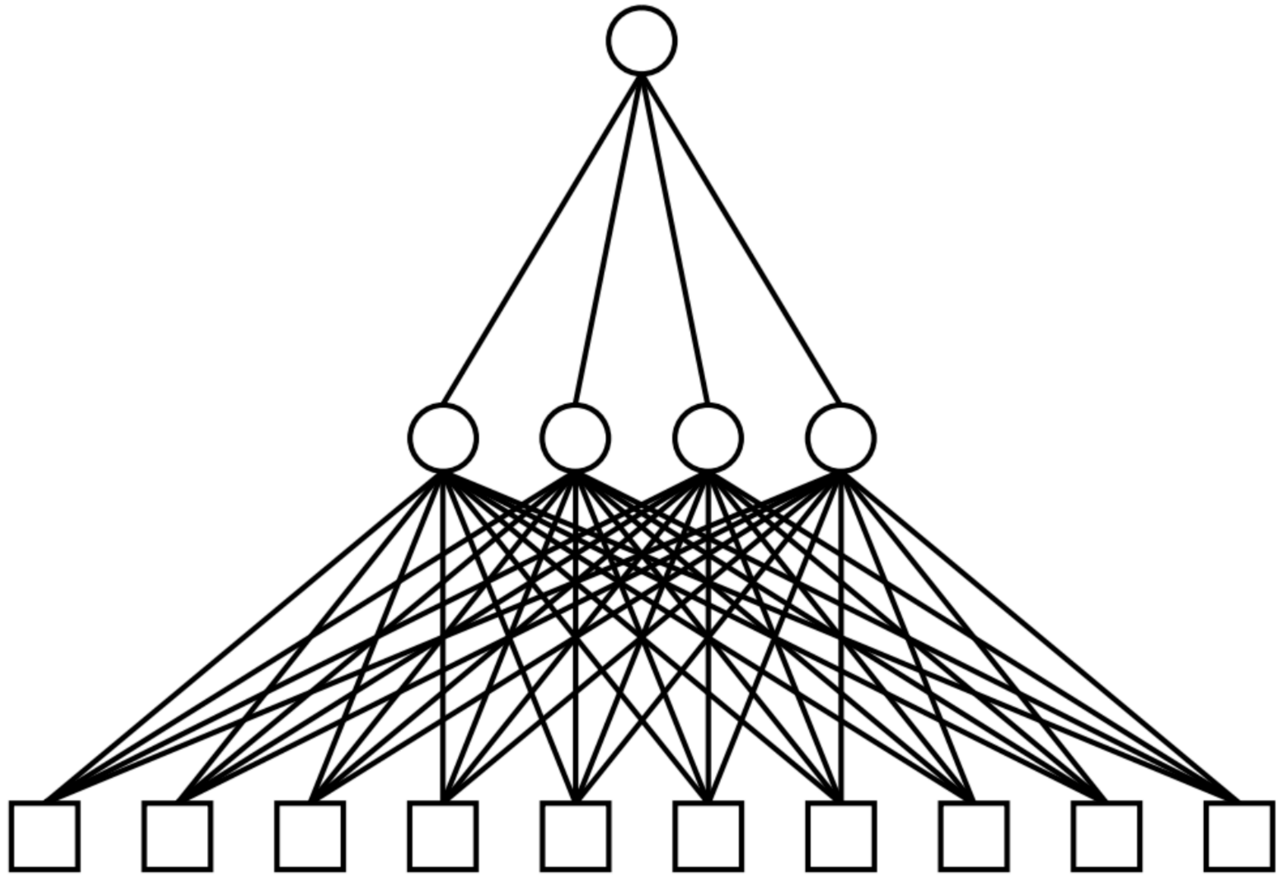
Output units a_i

$W_{j,i}$

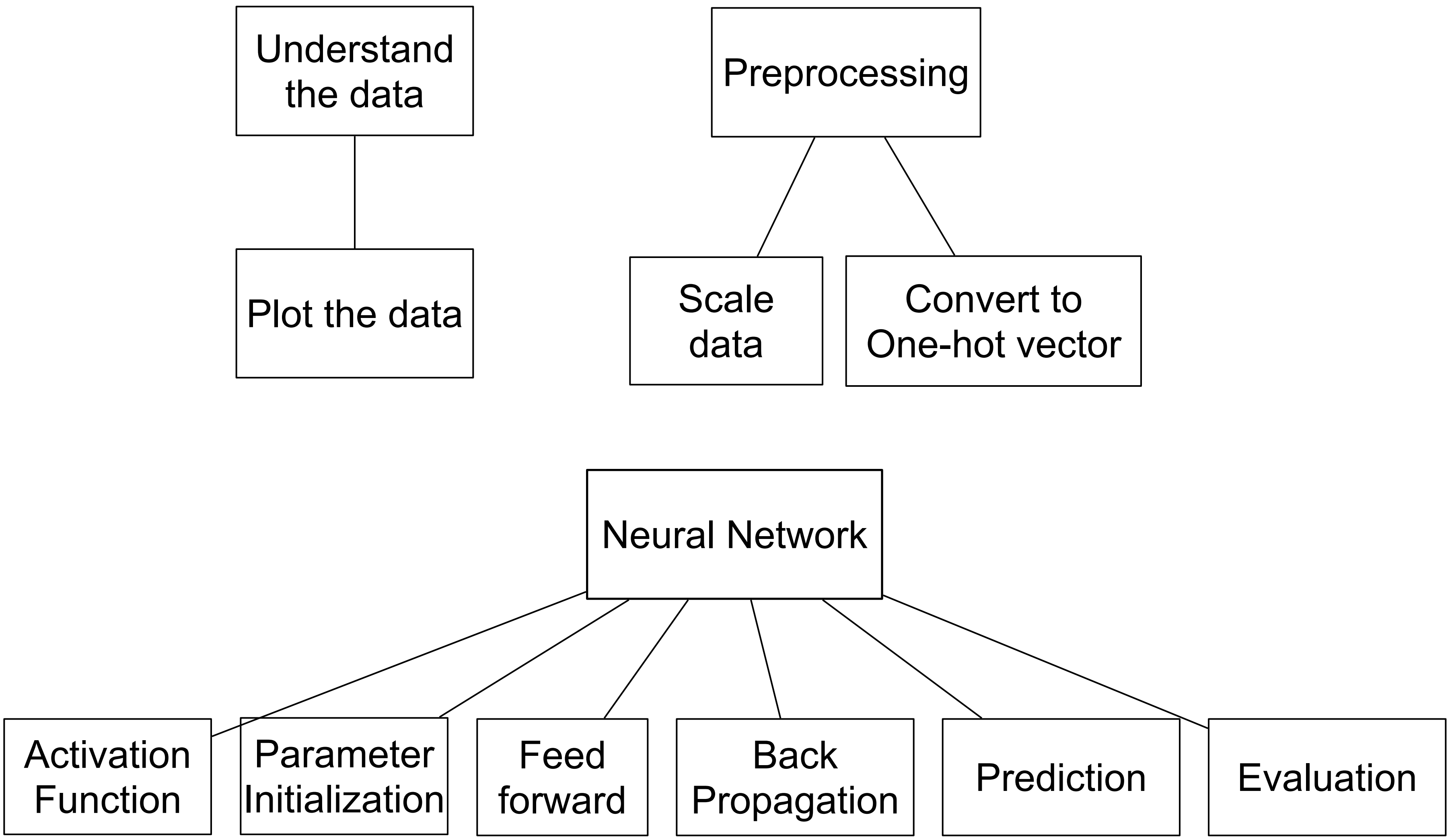
Hidden units a_j

$W_{k,j}$

Input units a_k



Main Modules for Neural Network



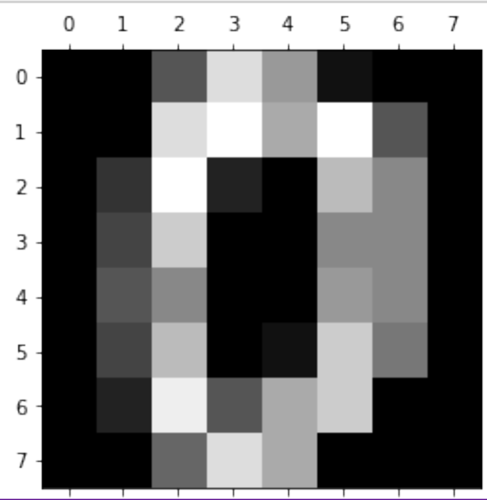
Main Modules for Neural Network

Understand the data

Plot the data

In [2]:
```python
# load all the digits (img)

# load the data from the digit (img)

print("The shape of the digits dataset:")
print(digits.data.shape)
# plot the digits
# using .gray()

# and .matshow() with argument digit.images[xx]

# plt.show()

# get the gt for this digit img

print(y[0:1])
print(X[0,:])
```

Example result:



Main Modules for Neural Network

Preprocessing

Scale data

- The training features range from 0 to 15.
- To help the algorithm converge, we will scale the data to have a mean of 0 and unit variance using *StandardScaler* from sklearn.preprocessing

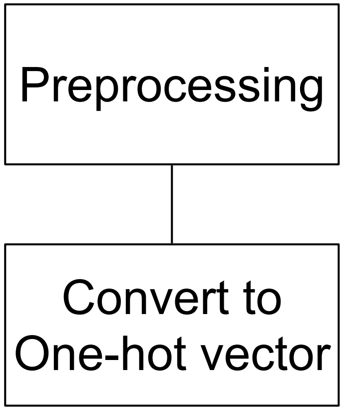
In [3]:
```
# use the stander lib to scale the data
# init the scaler

# fit the data to the scaler


# Looking the new features after scaling
```

Before scaling: [0. 0. 5. 13. 9. 1. 0. 0. 0. 0. 13. 15. 10. 15. 5. 0. 0. 3. 15. 2. 0. 11. 8. 0. 0. 4. 12. 0. 0. 8. 8. 0. 0. 5. 8. 0. 0. 9. 8. 0. 0. 4. 11. 0. 1. 12. 7. 0. 0. 2. 14. 5. 10. 12. 0. 0. 0. 0. 6. 13. 10. 0. 0. 0.]

After scaling:

```
array([ 0.        , -0.33501649, -0.04308102,  0.27407152, -0.66447751,
       -0.84412939, -0.40972392, -0.12502292, -0.05907756, -0.62400926,
        0.4829745 ,  0.75962245, -0.05842586,  1.12772113,  0.87958306,
       -0.13043338, -0.04462507,  0.11144272,  0.89588044, -0.86066632,
       -1.14964846,  0.51547187,  1.90596347, -0.11422184, -0.03337973,
        0.48648928,  0.46988512, -1.49990136, -1.61406277,  0.07639777,
        1.54181413, -0.04723238,  0.        ,  0.76465553,  0.05263019,
       -1.44763006, -1.73666443,  0.04361588,  1.43955804,  0.        ,
       -0.06134367,  0.8105536 ,  0.63011714, -1.12245711, -1.06623158,
        0.66096475,  0.81845076, -0.08874162, -0.03543326,  0.74211893,
        1.15065212, -0.86867056,  0.11012973,  0.53761116, -0.75743581,
       -0.20978513, -0.02359646, -0.29908135,  0.08671869,  0.20829258,
       -0.36677122, -1.14664746, -0.5056698 , -0.19600752])
```

Main Modules for Neural Network



- Our target is an integer in the range [0,..,9], so we will have 10 output neuron's in our network.
- If y=0, we want the output neurons to have the values (1,0,0,0,0,0,0,0,0,0)
- If y=2, we want the output neurons to have the values (0,0,1,0,0,0,0,0,0,0)
- Thus we need to change our target label accordingly so it is the same as our expectation for output of the neural network.

In [5]:
```
def convert_y_to_vect(y):
#     Our target is an integer in the range [0,..,9], so we will have 10 output neuron's in our network.

#     If  y=0 we want the output neurons to have the values (1,0,0,0,0,0,0,0,0,0)
#     If  y=1 we want the output neurons to have the values (0,1,0,0,0,0,0,0,0,0)
#     etc

#     Thus we need to change our target so it is the same as our hoped for output of the neural network.

#     If y=0$we change it into the vector (1,0,0,0,0,0,0,0,0,0)
#     If y=1 we change it into the vector (0,1,0,0,0,0,0,0,0,0)
#     etc

#     The code to covert the target vector.

    return
```

Main Modules for Neural Network

Neural Network

Activation Function

- Define the activation function for each neuron, e.g. sigmoid function, tanh function, ReLu
- Sigmoid Function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- Tanh function:

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

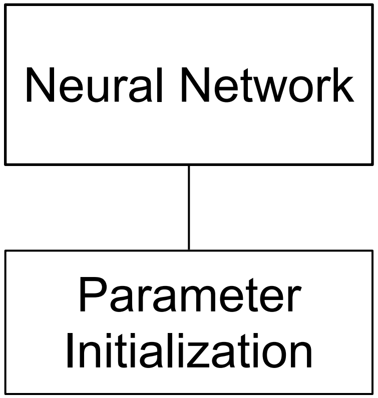
- ReLU (Rectified Linear Units) function:

$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

The activation function and its derivative

In [8]:
```
# We will use the sigmoid activation function:  f(z)={1}/{1+e^{-z}}
def f(z):
    return

# The deriviative of the sigmoid function is: $f'(z) = f(z)(1-f(z))$
def f_deriv(z):
    return
```

Main Modules for Neural Network

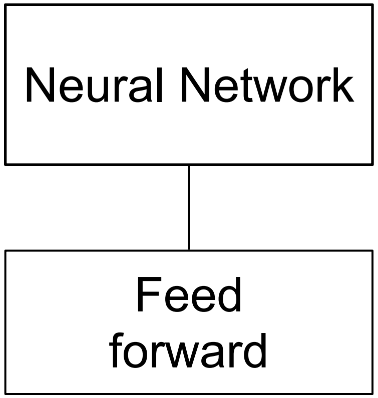


- We want to initialize weights in W to be different so that during back propagation the nodes on a level will have different gradients and thus have different update values.
- Assigns each weight a number uniformly drawn from [0.0,1.0) using *random_sample* from *numpy.random*
- Initialize $\triangle W, \triangle b$ as zeros

In [9]:
```python
def setup_and_init_weights(nn_structure):
    # The weights in W are different so that during back propagation the nodes on a level will have different gradients
    #creating a dictionary for wiehgts i.e. a set of key: value pairs

    #creating a dictionary for bias i.e. a set of key: value pairs

    for:
        # We want the weights to be small values, since the sigmoid is almost "flat" for large inputs.
        # Next is the code that assigns each weight a number uniformly drawn from $[0.0, 1.0)$.
        # The code assumes that the number of neurons in each level is in the python list *nn_structure*.
        # .random_sample return “continuous uniform” random floats in the half-open interval [0.0, 1.0).


    # Return weight and b
    return
```

In [10]:
```python
def init_tri_values(nn_structure):
    # Creating dlt_W and dlt_b to have the same size as W and b,and init the dlt_W, and dlt_b to 0


    # use for loop to init the dlt W and dlt b

        # you can use np.zeros
    return
```

Main Modules for Neural Network



- Given input x, W, b
- Calculate the value of f(W*x + b), where f is the activation function

In [11]:

```python
def feed_forward(x, W, b):
    # create a dictionary for holding the a values for all levels

    # create a dictionary for holding the z values for all the layers

    # for each layer
    for:

        # z^(l+1) = W^(l)*a^(l) + b^(l)

        # a^(l+1) = f(z^(l+1))

    return
```

Main Modules for Neural Network

Neural Network

Back Propagation

- Calculate the Mean Squared Error (MSE):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

- Calculate the $\triangle W, \triangle b$, update w, b

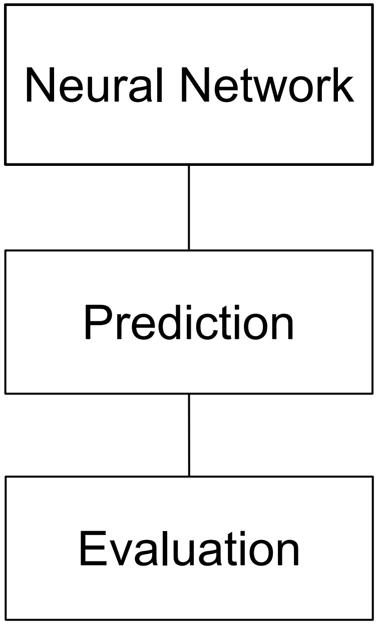
In [ ]:

```
def train_nn(nn_structure, X, y, iter_num=3000, alpha=0.25):
    # init W and b

    # init counter to 0

    # store the length of data

    # init a list to store the all costs

    print('Starting gradient descent for {} iterations'.format(iter_num))
    # while the counter is less than the max iterations:
    while:
        # print the iteration number for every 1000 iter
```

In [12]:

```
def calculate_out_layer_delta(y, a_out, z_out):
    # delta^(nl) = -(y_i - a_i^(nl)) * f'(z_i^(nl))
    return


def calculate_hidden_delta(delta_plus_1, w_l, z_l):
    # delta^(l) = (transpose(W^(l)) * delta^(l+1)) * f'(z^(l))
    return
```

Main Modules for Neural Network



- Given a testing sample, get the output of network using feed forward function with trained W, b
- Predict a label for given sample as the class with maximum output value
  - E.g. if we get output (0,0,0.2,0,0,0.5,0,0,0.3,0), then the predicted label should be 6
- Calculate the accuracy using *accuracy_score* from *sklearn.metrics*

In [13]:
```
def predict_y(W, b, X, n_layers):
    # store the length of data

    # init for prediction array

    # for each data:

        # feed forward

        # predict

    return
```

In [16]:
```
# get the prediction accuracy and print

print('Prediction accuracy is {}%'.format(accuracy_score(y_test, y_pred) * 100))
```

```
Prediction accuracy is 89.1515994437%
```

Pipeline for Neural Network