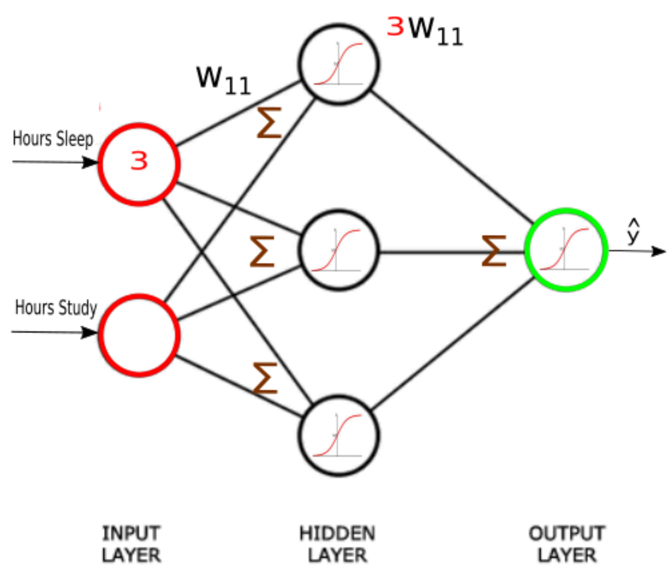


Our network has 2 inputs, 3 hidden units, and 1 output.

```
#Import code from last time
%pylab inline
from part1 import *
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
print X.shape, y.shape
```

```
(3, 2) (3, 1)
```

```
X
```

```
array([[ 0.3,  1. ],
       [ 0.5,  0.2],
       [ 1. ,  0.4]])
```

```
y
```

```
array([[ 0.75],
       [ 0.82],
       [ 0.93]])
```

This time we'll build our network as a python class.

The **init()** method of the class will take care of instantiating constants and variables.

```
class NeuralNetwork(object):
    def __init__(self):
        self.inputLayerSize = 2
        self.hiddenLayerSize = 3
        self.outputLayerSize = 1

    def forwardPropagation(self, X):
        #Propagate inputs though network
        pass
```


Quick Preview (/python/scikit-learn/scikit_machine_learning_qu

scikit-learn : Data Preprocessing I - Missing / Categorical data) (/python/scikit-learn/scikit_machine_learning_Da Missing-Data-Categorical-Data.php)

scikit-learn : Data Preprocessing II - Partitioning a dataset / Feature scaling / Feature Selection / Regularization (/python/scikit-learn/scikit_machine_learning_Da II-Datasets-Partitioning-Feature-scaling-Feature-Selection-Regularization.php)

scikit-learn : Data Preprocessing III - Dimensionality reduction vis Sequential feature selection / Assessing feature importance via random forests (/python/scikit-learn/scikit_machine_learning_Da III-Dimensionality-reduction-via-Sequential-feature-selection-Assessing-feature-importance-via-random-forests.php)

scikit-learn : Data Compression via Dimensionality Reduction I - Principal component analysis (PCA) (/python/scikit-learn/scikit_machine_learning_Da _PCA.php)

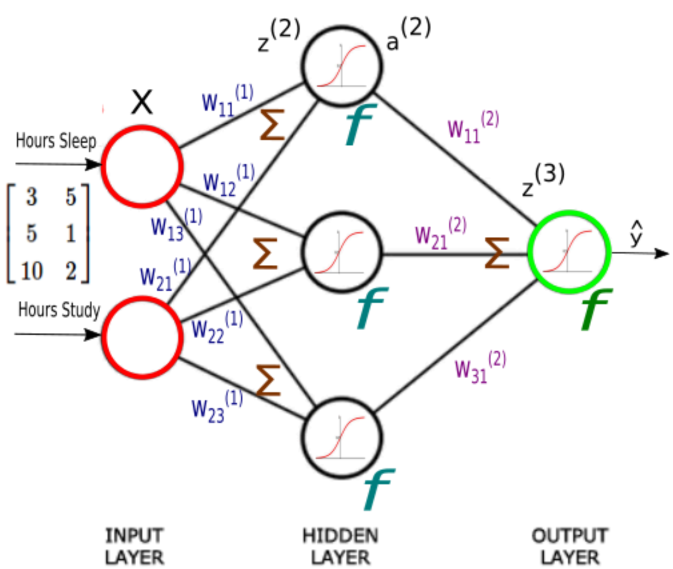
scikit-learn : Data Compression via Dimensionality Reduction II - Linear Discriminant Analysis (LDA) (/python/scikit-learn/scikit_machine_learning_Da

scikit-learn : Data Compression via Dimensionality Reduction III - Nonlinear mappings via kernel principal component

Variables

Code Symbol	Math Symbol	Definition	Dimensions
X	X	Input Data, each row in an example	(numExamples, inputLayerSize)
y	y	target data	(numExamples, outputLayerSize)
W1	$W^{(1)}$	Layer 1 weights	(inputLayerSize, hiddenLayerSize)
W2	$W^{(2)}$	Layer 2 weights	(hiddenLayerSize, outputLayerSize)
z2	$z^{(2)}$	Layer 2 activation	(numExamples, hiddenLayerSize)
a2	$a^{(2)}$	Layer 2 activity	(numExamples, hiddenLayerSize)
z3	$z^{(3)}$	Layer 3 activation	(numExamples, outputLayerSize)

$$z^{(2)} = XW^{(1)} \tag{1}$$
$$a^{(2)} = f\left(z^{(2)}\right) \tag{2}$$
$$z^{(3)} = a^{(2)}W^{(2)} \tag{3}$$
$$\hat{y} = f\left(z^{(3)}\right) \tag{4}$$



Each input value in matrix X should be multiplied by a corresponding **weight** and then added together with all the other results for each neuron.

$z^{(2)}$ is the activity of our second layer and it can be calculated as the following:

$$z^{(2)} = XW^{(1)} \tag{1}$$

$$= \begin{bmatrix} 3 & 5 \\ 5 & 1 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} 3W_{11}^{(1)} + 5W_{21}^{(1)} & 3W_{12}^{(1)} + 5W_{22}^{(1)} & 3W_{13}^{(1)} + 5W_{23}^{(1)} \\ 5W_{11}^{(1)} + W_{21}^{(1)} & 5W_{12}^{(1)} + W_{22}^{(1)} & 5W_{13}^{(1)} + W_{23}^{(1)} \\ 10W_{11}^{(1)} + 2W_{21}^{(1)} & 10W_{12}^{(1)} + 2W_{22}^{(1)} & 10W_{13}^{(1)} + 2W_{23}^{(1)} \end{bmatrix}$$

Note that each entry in z is a sum of weighted inputs to each hidden neuron. z is 3×3 matrix, one row for each sample, and one column for each hidden unit.

Activation function - sigmoid

Now that we have the **activities** for our second layer, $z^{(2)} = XW^{(1)}$, we need to apply the activation function.

We'll independently apply the **sigmoid** function to each entry in matrix z:

```
class NeuralNetwork(object):
    def __init__(self):
        self.inputLayerSize = 2
        self.hiddenLayerSize = 3
        self.outputLayerSize = 1

    def forwardPropagation(self, X):
        #Propagate inputs though network
        pass

    def sigmoid(self, z):
        #Apply sigmoid activation function to scalar, vector, or matrix
        return 1/(1+np.exp(-z))
```

By using numpy we'll apply the activation function element-wise, and return a result of the same dimension as it was given:

NN_k-nearest-neighbors-algorithm.php)

scikit-learn : Support Vector Machines (SVM) (/python/scikit-learn/scikit_machine_learning_Su

scikit-learn : Support Vector Machines (SVM) II (/python/scikit-learn/scikit_machine_learning_Su

Flask with Embedded Machine Learning I : Serializing with pickle and DB setup (/python/Flask/Python_Flask_Eml

Flask with Embedded Machine Learning II : Basic Flask App (/python/Flask/Python_Flask_Eml

Flask with Embedded Machine Learning III : Embedding Classifier (/python/Flask/Python_Flask_Eml

Flask with Embedded Machine Learning IV : Deploy (/python/Flask/Python_Flask_Eml

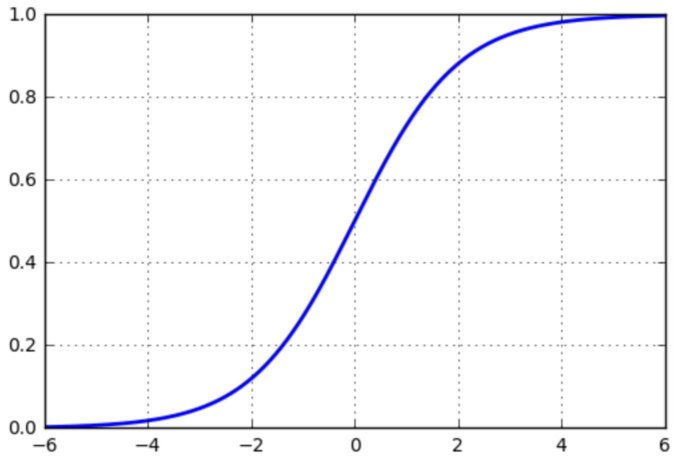
Flask with Embedded Machine Learning V : Updating the classifier (/python/Flask/Python_Flask_Eml

scikit-learn : Sample of a spam comment filter using SVM - classifying a good one or a bad one (/python/scikit-learn/scikit_learn_Support_Vecto

MACHINE LEARNING ALGORITHMS

Batch gradient descent algorithm (/python/python_numpy_batch_g

```
NN = NeuralNetwork()
testInput = np.arange(-6, 6, 0.01)
plot(testInput, NN.sigmoid(testInput), linewidth= 2)
grid(1)
```



Let's see how the **sigmoid()** takes an input and how returns the result:

The following calls for the **sigmoid()** with args : a number (scalar), 1-D (vector), and 2-D arrays (matrix).

```
NN.sigmoid(1)
```

```
0.7310585786300049
```

```
NN.sigmoid(np.array([-1,0,1]))
```

```
array([ 0.26894142,  0.5       ,  0.73105858])
```

```
NN.sigmoid(np.random.randn(3,3))
```

```
array([[ 0.61389443,  0.75267267,  0.65807184],
       [ 0.37843207,  0.42583846,  0.39238314],
       [ 0.24758466,  0.59707024,  0.88510656]])
```

Weight-matrices : $W^{(1)}$ and $W^{(2)}$

We initialize our weight matrices ($W^{(1)}$ and $W^{(2)}$) in our **__init__()** method with random numbers.

```
class NeuralNetwork(object):
    def __init__(self):
        #Define Hyperparameters
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3

        #Weights (parameters)
        self.W1 = np.random.randn(self.inputLayerSize, self.hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize, self.outputLayerSize)
```


Single Layer Neural Network - Perceptron model on the Iris dataset using Heaviside step activation function (/python/scikit-learn/Perceptron_Model_with_Iri

Batch gradient descent versus stochastic gradient descent (SGD) (/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php)

Single Layer Neural Network - Adaptive Linear Neuron using linear (identity) activation function with batch gradient descent method (/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron.php)

Single Layer Neural Network : Adaptive Linear Neuron using linear (identity) activation function with stochastic gradient descent (SGD) (/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron-with-Stochastic-Gradient-Descent.php)

VC (Vapnik-Chervonenkis) Dimension and Shatter (/python/scikit-learn/scikit_machine_learning_VC

Bias-variance tradeoff (/python/scikit-learn/scikit_machine_learning_Bi variance-Tradeoff.php)

Logistic Regression (/python/scikit-learn/logistic_regression.php)

Maximum Likelihood Estimation (MLE) (/python/scikit-learn/Maximum-

Likelyhood-Estimation-MLE.php)

Neural Networks with backpropagation for XOR using one hidden layer (/python/python_Neural_Networ

minHash (/Algorithms/minHash_Jaccard_S

tf-idf weight (/Algorithms/tf_idf_term_frequen

Natural Language Processing (NLP): Sentiment Analysis I (IMDb & bag-of-words) (/Algorithms/Machine_Learning_

Natural Language Processing (NLP): Sentiment Analysis II (tokenization, stemming, and stop words) (/Algorithms/Machine_Learning_

Natural Language Processing (NLP): Sentiment Analysis III (training & cross validation) (/Algorithms/Machine_Learning_

Natural Language Processing (NLP): Sentiment Analysis IV (out-of-core) (/Algorithms/Machine_Learning_

Locality-Sensitive Hashing (LSH) using Cosine Distance (Cosine Similarity) (/Algorithms/Locality_Sensitive_H

Implementing forward propagation

We now have our second formula for forward propagation, using our activation function(f), we can write that our second layer activity: $a^{(2)} = f\left(z^{(2)}\right)$. The $a^{(2)}$ will be a matrix of the same size (3×3):

$$a^{(2)} = f(z^{(2)}) \tag{2}$$

To finish **forward propagation** we want to propagate $a^{(2)}$ all the way to the output, $\hat{y}$.

All we have to do now is multiply $a^{(2)}$ by our second layer weights $W^{(2)}$ and apply one more activation function. The $W^{(2)}$ will be of size 3×1, one weight for each synapse:

$$z^{(3)} = a^{(2)} W^{(2)} \tag{3}$$

Multiplying $a^{(2)}$, a (3×3 matrix), by $W^{(2)}$, a (3×1 matrix) results in a 3×1 matrix $z^{(3)}$, the activity of our 3rd layer. The $z^{(3)}$ has three activity values, one for each sample.

Then, we'll apply our activation function to $z^{(3)}$ yielding our estimate of test score, $\hat{y}$:

$$\hat{y} = f\left(z^{(3)}\right) \tag{4}$$

Now we are ready to implement forward propagation in our **forwardPropagation()** method, using numpy's built in dot method for matrix multiplication:

```
class NeuralNetwork(object):
    def __init__(self):
        #Define Hyperparameters
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3

        #Weights (parameters)
        self.W1 = np.random.randn(self.inputLayerSize, self.hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize, self.outputLayerSize)

    def forwardPropagation(self, X):
        #Propagate inputs though network
        self.z2 = np.dot(X, self.W1)
        self.a2 = self.sigmoid(self.z2)
        self.z3 = np.dot(self.a2, self.W2)
        yHat = self.sigmoid(self.z3)
        return yHat

    def sigmoid(self, z):
        #Apply sigmoid activation function to scalar, vector, or matrix
        return 1/(1+np.exp(-z))
```

ARTIFICIAL NEURAL NETWORKS (ANN)

Getting estimate of test score

Now we have a class capable of estimating our test score given how many hours we sleep and how many hours we study. We pass in our input data (X) and get real outputs ($\hat{y}$).

```
X
```

```
array([[ 0.3,  1. ],
       [ 0.5,  0.2],
       [ 1. ,  0.4]])
```

```
NN = NeuralNetwork()
yHat = NN.forwardPropagation(X)
```

```
yHat
```

```
array([[ 0.56539674],
       [ 0.44717722],
       [ 0.54117384]])
```

```
y
```

```
array([[ 0.75],
       [ 0.82],
       [ 0.93]])
```

Note that our estimates ($\hat{y}$) looks quite terrible when compared with our target (y). That's because we have not yet trained our network, that's what we'll work on next article.

Next:

3. Gradient Descent (/python/scikit-learn/Artificial-Neural-Network-ANN-3-Gradient-Descent.php)

Machine Learning with scikit-learn