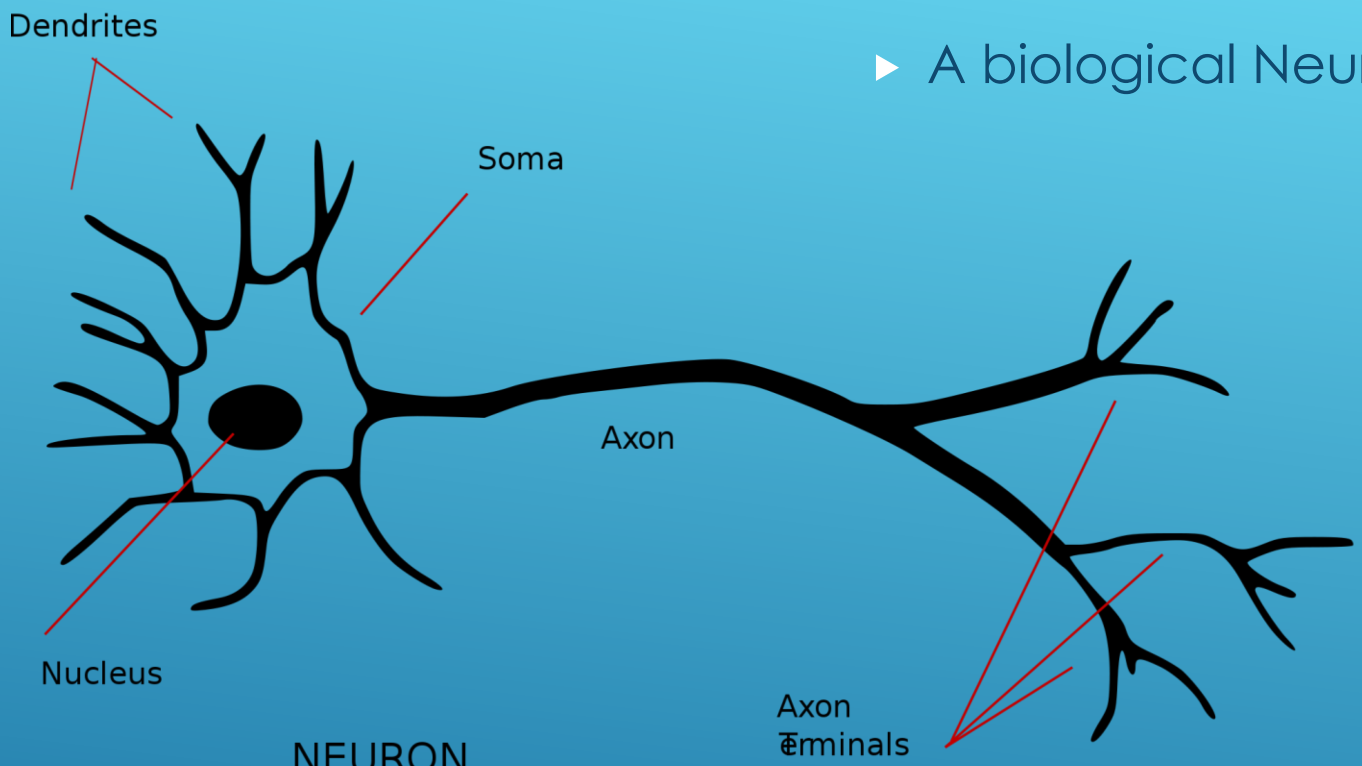
NEURAL NETWORKS

Power Hidden in Obscurity

“I'M GOING TO TEACH YOU IN THIS LECTURE HOW TO ANSWER QUESTIONS ABOUT NEUROBIOLOGY WITH AN 80% PROBABILITY THAT YOU WILL GIVE THE SAME ANSWER AS A NEUROBIOLOGIST.”

PROFESSOR PATRICK HENRY WINSTON

Comment to his class on Artificial Intelligence at Massachusets Institute of Technology.



NEURON

- A biological Neuron

THE BASIC BUILDING BLOCK OF THE BRAIN

“AND NOW, I’M GOING TO SHOW YOU HOW TO ANSWER A QUESTION ABOUT NEUROBIOLOGY WITH 80% PROBABILITY YOU’LL GET IT RIGHT. JUST SAY, WE DON’T KNOW!”

PROFESSOR PATRICK HENRY WINSTON

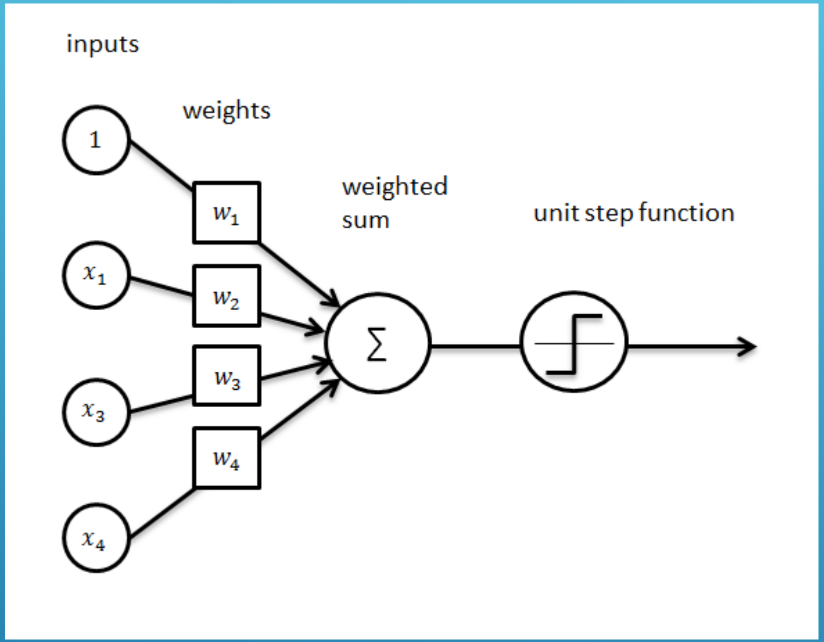
Comment to his class on Artificial Intelligence at Massachusets Institute of Technology.

IS NEURAL NETWORKS WORTH TEACHING.

- In 2010 MIT was considering dropping neural networks from the AI curriculum. The success was not very good.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton from the University of Toronto brought neural networks back to the forefront with a neural net with 60 million parameters in it.
- It made some mistakes but was remarkably accurate and brought neural networks back from near extinction.

Image from Imagenet Classification With Deep Convolutional Neural Networks.pdf

Perceptron



Emulates Neuron

- Has *m* inputs corresponding to features of Data Set.
- Weights each input and sums the value.
- Includes a bias of 1 and a weight
- Passes output through a sigmoid function.

NEURAL NETWORK PERCEPTRON

[Ahmet Taspinar: The Perceptron]

CLASS DATA

- Normalize data and output to an list of inputs
- Each point of classification data is assigned a numerical value from 0 to 1.

Executable File | 36 lines (32 sloc) | 1.24 KB

Raw | Blame | History

```python
#! /usr/bin/python3
## reads in file and writes out normalized data

def class_array():
    f = open('class.csv', 'r')
    read_array = []
    for line in f:
        line = line.strip()
        read_array.append(line.split(','))
    f.close()

#day        weekday = 0, saturday = .33, sunday =.66, holiday = 1;
#seasons    summer = 0, autumn = .33, winter =.66, spring = 1;
#wind       none = 0, normal = .5, high = 1;
#Rain       none = 0, slight = .5, heavy = 1;
#answer     cancelled = 0, very late = .33, late = .66, on time = 1;
    normalized_array = []
    entry=[]
    for list in read_array:
        for s in list:
            if s == 'weekday' or s == 'summer' or s == 'none' or s == 'cancelled':
                entry.append(0)
            elif s == 'saturday' or s == 'autumn' or s == 'very late':
                entry.append(.33)
            elif s == 'normal' or s == 'slight':
                entry.append(.5)
            elif s == 'sunday' or s == 'winter' or s == 'late':
                entry.append(.66)
            elif s == 'holiday' or s == 'spring' or s == 'on time' or s == 'heavy' or s == 'high':
                entry.append(1)
        if entry:
            normalized_array.append(entry)
        entry = []

    return normalized_array
```

Executable File | 32 lines (28 sloc) | 952 Bytes

Raw | Blame | History

```
#! /usr/bin/python3
## reads in file and writes out normalized data
def weather_array():
    f = open('weather.csv', 'r')
    read_array = []
    for line in f:
        line = line.strip()
        read_array.append(line.split(','))
    f.close()


#outlook rain = 0, overcast = .5, sun = 1;
#temperature cool = 0, mild = .5, hot = 1;
#humidity normal = 0, high = 1;
#windy  False = 0, true = 1;
#answer no = 0, yes = 1;
    normalized_array = []
    entry=[]
    for list in read_array:
        for s in list:
            if s == 'rainy' or s == 'cool' or s == 'normal' or s == 'FALSE' or s == 'no':
                entry.append(0)
            elif s == 'overcast' or s == 'mild':
                entry.append(.5)
            elif s == 'sunny' or s == 'hot' or s == 'high' or s == 'TRUE' or s == 'yes':
                entry.append(1)
        if entry:
            normalized_array.append(entry)
        entry = []

    return normalized_array
```

WEATHER DATA

- Normalize data and output to an list of inputs
- Each point of classification data is assigned a numerical value from 0 to 1.

```python
#returns layer of weights ranomized -.5 <= w <= .5
def randomize_node_weights(num):
    node = []
    for i in range(0,num):
        node.append(random.uniform(-.5,.5))

    return node

#returns fully populated neural net of n layers by m nodes/layer
def define_neural_net(n,m):
    layers = []
    for i in range(0,n):
        layers.append(randomize_node_weights(4))
    return layers

```

```python
#returns entry passed through node
def return_weighted_input(node=[],input=[]):
    weighted_value = 0
    weighted_input = []
    for i in range(0,len(node)):
        weighted_value += BIAS_VALUE
        for j in range(0,len(node)):
            #summation of weights to each node.
            weighted_value += (node[i]*input[j])
        weighted_input.append(1/(1+math.exp(-1*weighted_value)))
    return weighted_input
```

MY NEURAL NETWORK WAS LACKING

MY UNDERSTANDING CAME MUCH TO LATE

In trying to finish my program and understand Backward Propagation I came upon a few flaws with my design. I miscalculated the number of weights for each layer. Structurally my code has to change some to allow for proper weighting of each synapse. The next couple slides make clear a simple walk through of a proper neural net.

Step 0: Read input and output

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0																1	
1	0	1	1																1	
0	1	0	1																0	

Step 1: Initialize weights and biases with random values (There are methods to initialize weights and biases but for now initialize with random values)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08							0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68										0.25			1	
0	1	0	1	0.60	0.18	0.47										0.23			0	
				0.92	0.11	0.52														

Borrowed from Analyticsvidhya.com

STEPS THROUGH THE METHODS

Step 2: Calculate hidden layer input:

hidden_layer_input= matrix_dot_product(X,wh) + bh

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10				0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61				0.25			1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27				0.23			0	
				0.92	0.11	0.52														

Step 3: Perform non-linear transformation on hidden linear input

hiddenlayer_activations = sigmoid(hidden_layer_input)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25			1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23			0	
				0.92	0.11	0.52														

Step 4: Perform linear and non-linear transformation of hidden layer activation at output layer

output_layer_input = matrix_dot_product (hiddenlayer_activations * wout) + bout

output = sigmoid(output_layer_input)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	
				0.92	0.11	0.52														

Step 10: Update weight at both output and hidden layer

wout = wout + matrix_dot_product(hiddenlayer_activations.Transpose, d_output)*learning_rate

wh = wh+ matrix_dot_product(X.Transpose,d_hiddenlayer)*learning_rate

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.29	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.51														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

Learning Rate	0.1

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

delta output
0.035
0.033
-0.131

REFERENCES

- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-12a-neural-nets/uXt8qF2Zzfo.pdf
- http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- http://ataspinar.com/2016/12/22/the-perceptron/
- https://github.com/vfoley/neuralNet
- https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/