backpropagation

April 15, 2019

1 Backpropagation

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network.Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout the network's layers. It is commonly used to train deep neural networks.

```
In [1]: from math import exp
        from random import seed
        from random import random
```

```
In [2]: # Initialize a network
        def initialize_network(n_inputs, n_hidden, n_outputs):
            network = list()
            hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range
            network.append(hidden_layer)
            output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range
            network.append(output_layer)
            return network
```

```
In [3]: # Calculate neuron activation for an input
        def activate(weights, inputs):
            activation = weights[-1]
            for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
            return activation
```

```
In [4]: # Transfer neuron activation
        def transfer(activation):
            return 1.0 / (1.0 + exp(-activation))
```

```
In [5]: # Forward propagate input to a network output
        def forward_propagate(network, row):
            inputs = row
            for layer in network:
                new_inputs = []
                for neuron in layer:
```

```
                activation = activate(neuron['weights'], inputs)
                neuron['output'] = transfer(activation)
                new_inputs.append(neuron['output'])
            inputs = new_inputs
        return inputs
```

In [6]:
```
# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)
```

In [7]:
```
# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                    errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

In [8]:
```
# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']
```

In [9]:
```
# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))
```

```
                backward_propagate_error(network, expected)
                update_weights(network, row, l_rate)
            print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
```

In [10]:
```
# Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

```
>epoch=0, lrate=0.500, error=6.365
>epoch=1, lrate=0.500, error=5.557
>epoch=2, lrate=0.500, error=5.291
>epoch=3, lrate=0.500, error=5.262
>epoch=4, lrate=0.500, error=5.217
>epoch=5, lrate=0.500, error=4.899
>epoch=6, lrate=0.500, error=4.419
>epoch=7, lrate=0.500, error=3.900
>epoch=8, lrate=0.500, error=3.461
>epoch=9, lrate=0.500, error=3.087
>epoch=10, lrate=0.500, error=2.758
>epoch=11, lrate=0.500, error=2.468
>epoch=12, lrate=0.500, error=2.213
>epoch=13, lrate=0.500, error=1.989
>epoch=14, lrate=0.500, error=1.792
>epoch=15, lrate=0.500, error=1.621
>epoch=16, lrate=0.500, error=1.470
>epoch=17, lrate=0.500, error=1.339
>epoch=18, lrate=0.500, error=1.223
>epoch=19, lrate=0.500, error=1.122
[{'weights': [-0.9766426647918854, 1.0573043092399, 0.7999535671683315], 'output': 0.054299270
[{'weights': [1.4965066037208181, 1.770264295168642, -1.28526000789383], 'output': 0.246982887
```

1.1 References:

1. https://en.wikipedia.org/wiki/Backpropagation
2. https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
3. https://www.youtube.com/watch?v=aircAruvnKk&list=PLLMP7TazTxHrgVk7w1EKpLBIDoC50QrPS&y
4. http://neuralnetworksanddeeplearning.com/chap2.html