0COSC4337_Back_Propagation

[1]:
```python
'''
Initialize Network.
Forward Propagate.
Back Propagate Error.
Train Network.
Predict.
'''
```

[1]: '\nInitialize Network.\nForward Propagate.\nBack Propagate Error.\nTrain Network.\nPredict.\n'

[2]:
```python
# 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(n_hidden)]
        network.append(hidden_layer)
        output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for␣
↪i in range(n_outputs)]
        network.append(output_layer)
        return network
```

[3]:
```python
from random import seed
from random import random

# 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(n_hidden)]
        network.append(hidden_layer)
        output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for␣
↪i in range(n_outputs)]
        network.append(output_layer)
        return network

seed(1)
```

```
network = initialize_network(2, 1, 2)
for layer in network:
        print(layer)
```

```
[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]}]
[{'weights': [0.2550690257394217, 0.49543508709194095]}, {'weights':
[0.4494910647887381, 0.651592972722763]}]
```

[4]:
```
'''
Neuron Activation.
Neuron Transfer.
Forward Propagation.
'''
```

[4]:
```
'\nNeuron Activation.\nNeuron Transfer.\nForward Propagation.\n'
```

[5]:
```
# 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
```

[6]:
```
# Transfer neuron activation
def transfer(activation):
        return 1.0 / (1.0 + exp(-activation))
```

[7]:
```
# 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
```

[8]:
```
from math import exp

# 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
```

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

# 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

# test forward propagation
network = [[{'weights': [0.13436424411240122, 0.8474337369372327, 0.
 ↪763774618976614]}],
                [{'weights': [0.2550690257394217, 0.49543508709194095]},␣
 ↪{'weights': [0.4494910647887381, 0.651592972722763]}]]
row = [1, 0, None]
output = forward_propagate(network, row)
print(output)
```

```
[0.6629970129852887, 0.7253160725279748]
```

[9]:
```
'''
Transfer Derivative.
Error Backpropagation.
'''
```

[9]:
```
'\nTransfer Derivative.\nError Backpropagation.\n'
```

[10]:
```
# Calculate the derivative of an neuron output
def transfer_derivative(output):
        return output * (1.0 - output)
```

[11]:
```
''' normal error
error = (expected - output) * transfer_derivative(output)
'''
```

[11]:
```
' normal error\nerror = (expected - output) * transfer_derivative(output)\n'
```

[12]:
```
'''backprop error
error = (weight_k * error_j) * transfer_derivative(output)
'''
```

[12]: 'backprop error\nerror = (weight_k * error_j) * transfer_derivative(output)\n'

[13]:
```
# 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'])
```

[14]:
```
# test backpropagation of error
network = [[{'output': 0.7105668883115941, 'weights': [0.13436424411240122, 0.
 ↪8474337369372327, 0.763774618976614]}],
                [{'output': 0.6213859615555266, 'weights': [0.2550690257394217,␣
 ↪0.49543508709194095]}, {'output': 0.6573693455986976, 'weights': [0.
 ↪4494910647887381, 0.651592972722763]}]]
expected = [0, 1]
backward_propagate_error(network, expected)
for layer in network:
        print(layer)
```

```
[{'output': 0.7105668883115941, 'weights': [0.13436424411240122,
0.8474337369372327, 0.763774618976614], 'delta': -0.0005348048046610517}]
[{'output': 0.6213859615555266, 'weights': [0.2550690257394217,
0.49543508709194095], 'delta': -0.14619064683582808}, {'output':
0.6573693455986976, 'weights': [0.4494910647887381, 0.651592972722763], 'delta':
0.0771723774346327}]
```

[ ]:
```
'''
Train the model nto two sections:

Update Weights.
Train Network.
'''
```

[15]:
```
# 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']
```

[16]:
```
# 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))
```

[17]:
```
from math import exp
from random import seed
from random import random

# 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(n_hidden)]
        network.append(hidden_layer)
        output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for␣
↪i in range(n_outputs)]
        network.append(output_layer)
        return network

# 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

# Transfer neuron activation
def transfer(activation):
        return 1.0 / (1.0 + exp(-activation))

# 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

# Calculate the derivative of an neuron output
def transfer_derivative(output):
        return output * (1.0 - output)

# 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'])

# 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']

# 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))

# 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=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
```

```
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.626
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472
>epoch=17, lrate=0.500, error=1.346
>epoch=18, lrate=0.500, error=1.233
>epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.850887325439514, 1.0858178629550297],
'output': 0.029980305604426185, 'delta': -0.0059546604162323625}, {'weights':
[0.37711098142462157, -0.0625909894552989, 0.2765123702642716], 'output':
0.9456229000211323, 'delta': 0.0026279652850863837}]
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275],
'output': 0.23648794202357587, 'delta': -0.04270059278364587}, {'weights':
[-2.5584149848484263, 1.0036422106209202, 0.42383086467582715], 'output':
0.7790535202438367, 'delta': 0.03803132596437354}]
```

[ ]:

```

```