How to Implement the Backpropagation Algorithm From Scratch In Python

by **Jason Brownlee** on November 7, 2016 in **Code Machine Learning Algorithms From Scratch**

The backpropagation algorithm is the classical feed-forward artificial neural network.

It is the technique still used to train large deep learning networks.

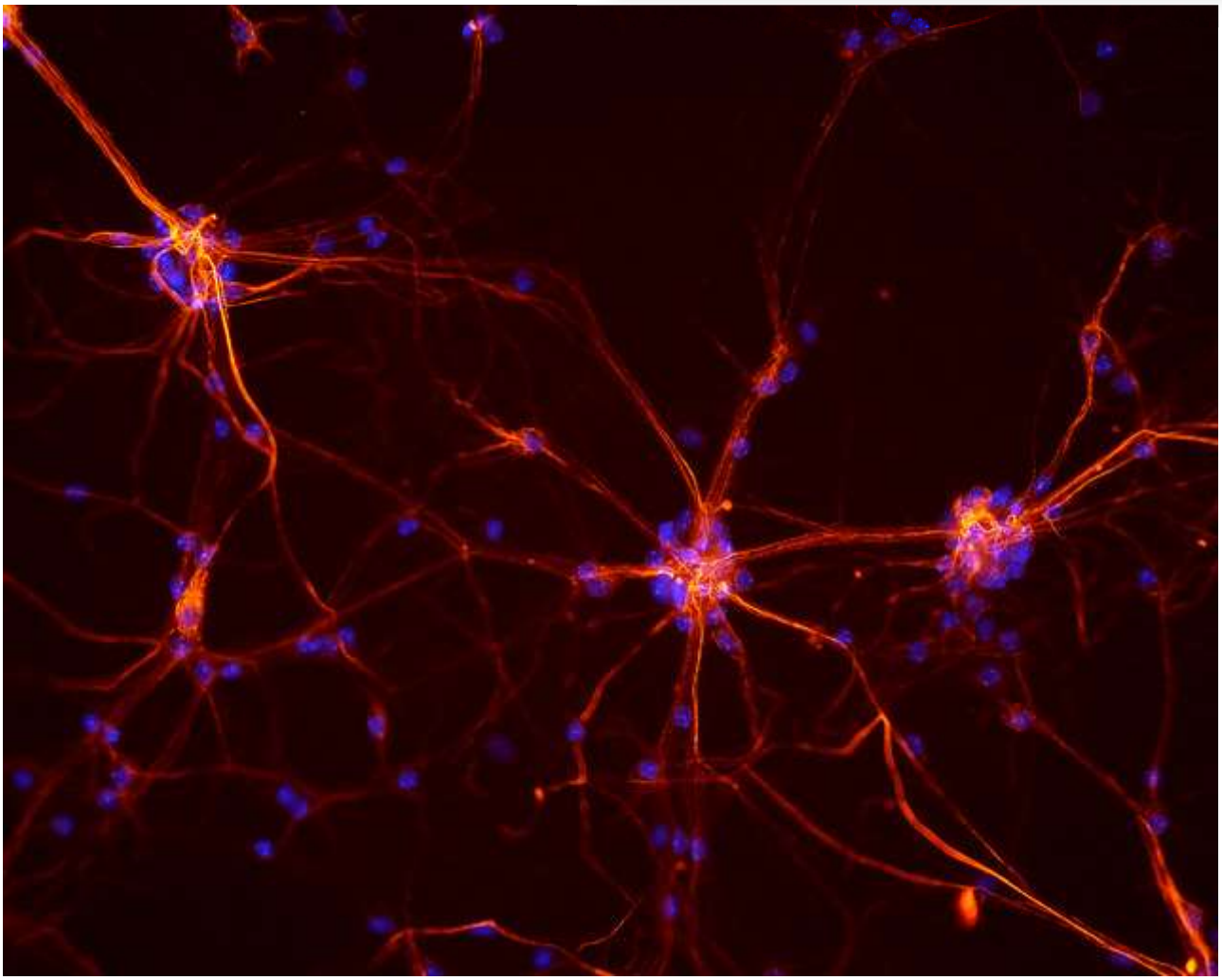
In this tutorial, you will discover how to implement the backpropagation algorithm from scratch with Python.

After completing this tutorial, you will know:

- How to forward-propagate an input to calculate an output.
- How to back-propagate error and train a network.
- How to apply the backpropagation algorithm to a real-world predictive modeling problem.

Let's get started.

- **Update Nov/2016**: Fixed a bug in the activate() fu
- **Update Jan/2017**: Changed the calculation of fol integer. Fixes issues with Python 3.
- **Update Jan/2017**: Updated small bug in update_
- **Update Apr/2018**: Added direct link to CSV datas
- **Update Aug/2018**: Tested and updated to work w



How to Implement the Backpropagation Algorithm From Scratch In Python
Photo by NICHD, some rights reserved.

Description

This section provides a brief introduction to the Backpropagation Algorithm and the Wheat Seeds dataset that we will be using in this tutorial.

Backpropagation Algorithm

The Backpropagation algorithm is a supervised learni om the field of Artificial Neural Networks.

Feed-forward neural networks are inspired by the info lled a neuron. A neuron accepts input signals via its dendr ell body. The axon carries the signal out to synapses, wh ll's dendrites.

The principle of the backpropagation approach is to m weightings of input signals to produce an expected ou sed

learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state.

Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer.

Backpropagation can be used for both classification and regression problems, but we will focus on classification in this tutorial.

In classification problems, best results are achieved when the network has one neuron in the output layer for each class value. For example, a 2-class or binary classification problem with the class values of A and B. These expected outputs would have to be transformed into binary vectors with one column for each class value. Such as [1, 0] and [0, 1] for A and B respectively. This is called a one hot encoding.

Wheat Seeds Dataset

The seeds dataset involves the prediction of species given measurements seeds from different varieties of wheat.

There are 201 records and 7 numerical input variables. It is a classification problem with 3 output classes. The scale for each numeric input value vary, so some data normalization may be required for use with algorithms that weight inputs like the backpropagation algorithm.

Below is a sample of the first 5 rows of the dataset.

```
15.26,14.84,0.871,5.763,3.312,2.221,5.22,1
14.88,14.57,0.8811,5.554,3.333,1.018,4.956,1
14.29,14.09,0.905,5.291,3.337,2.699,4.825,1
13.84,13.94,0.8955,5.324,3.379,2.259,4.805,1
16.14,14.99,0.9034,5.658,3.562,1.355,5.175,1
```

Using the Zero Rule algorithm that predicts the most common class value, the baseline accuracy for the problem is 28.095%.

You can learn more and download the seeds dataset f

Download the seeds dataset and place it into your cur
seeds_dataset.csv.

The dataset is in tab-separated format, so you must c et
program.

Update, download the dataset in CSV format directly:

- Download Wheat Seeds Dataset

Tutorial

This tutorial is broken down into 6 parts:

1. Initialize Network.
2. Forward Propagate.
3. Back Propagate Error.
4. Train Network.
5. Predict.
6. Seeds Dataset Case Study.

These steps will provide the foundation that you need to implement the backpropagation algorithm from scratch and apply it to your own predictive modeling problems.

1. Initialize Network

Let's start with something easy, the creation of a new network ready for training.

Each neuron has a set of weights that need to be maintained. One weight for each input connection and an additional weight for the bias. We will need to store additional properties for a neuron during training, therefore we will use a dictionary to represent each neuron and store properties by names such as '**weights**' for the weights.

A network is organized into layers. The input layer is really just a row from our training dataset. The first real layer is the hidden layer. This is followed by the output layer that has one neuron for each class value.

We will organize layers as arrays of dictionaries and treat the whole network as an array of layers.

It is good practice to initialize the network weights to small random numbers. In this case, will we use random numbers in the range of 0 to 1.

Below is a function named **initialize_network()** that creates a new neural network ready for training. It accepts three parameters, the number of inputs, the number of neurons to have in the hidden layer and the number of outputs.

You can see that for the hidden layer we create **n_hid**
has **n_inputs + 1** weights, one for each input column

You can also see that the output layer that connects t
n_hidden + 1 weights. This means that each neuron i
neuron in the hidden layer.

```
1 # Initialize a network
2 def initialize_network(n_inputs, n_hidden, n_o
3     network = list()
4     hidden_layer = [{'weights':[random() for i
5     network.append(hidden_layer)
6     output_layer = [{'weights':[random() for i
```

```
    network.append(output_layer)
    return network
```

Let's test out this function. Below is a complete example that creates a small network.

```
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_hidder
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_output
    network.append(output_layer)
    return network

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

Running the example, you can see that the code prints out each layer one by one. You can see the hidden layer has one neuron with 2 input weights plus the bias. The output layer has 2 neurons, each with 1 weight plus the bias.

```
[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]}]
[{'weights': [0.2550690257394217, 0.49543508709194095]}, {'weights': [0.4494910647887381, 0.651
```

Now that we know how to create and initialized a network, let's see how we can use it to calculate an output.

2. Forward Propagate

We can calculate an output from a neural network by propagating an input signal through each layer until the output layer outputs its values.

We call this forward-propagation.

It is the technique we will need to generate predictions during training that will need to be corrected, and it is the method we will need after the network is trained

We can break forward propagation down into three pa

1. Neuron Activation.
2. Neuron Transfer.
3. Forward Propagation.

2.1. Neuron Activation

The first step is to calculate the activation of one neur

The input could be a row from our training dataset, as
outputs from each neuron in the hidden layer, in the ca

Neuron activation is calculated as the weighted sum of the inputs. Much like linear regression.

```
activation = sum(weight_i * input_i) + bias
```

Where **weight** is a network weight, **input** is an input, **i** is the index of a weight or an input and **bias** is a special weight that has no input to multiply with (or you can think of the input as always being 1.0).

Below is an implementation of this in a function named **activate()**. You can see that the function assumes that the bias is the last weight in the list of weights. This helps here and later to make the code easier to read.

```
# 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
```

Now, let's see how to use the neuron activation.

2.2. Neuron Transfer

Once a neuron is activated, we need to transfer the activation to see what the neuron output actually is.

Different transfer functions can be used. It is traditional to use the sigmoid activation function, but you can also use the tanh (hyperbolic tangent) function to transfer outputs. More recently, the rectifier transfer function has been popular with large deep learning networks.

The sigmoid activation function looks like an S shape, it's also called the logistic function. It can take any input value and produce a number between 0 and 1 on an S-curve. It is also a function of which we can easily calculate the derivative (slope) that we will need later when backpropagating error.

We can transfer an activation function using the sigmoid function as follows:

```
output = 1 / (1 + e^(-activation))
```

Where **e** is the base of the natural logarithms (Euler's number).

Below is a function named **transfer()** that implements

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

Now that we have the pieces, let's see how they are u

2.3. Forward Propagation

Forward propagating an input is straightforward.

We work through each layer of our network calculating om one layer become inputs to the neurons on the next la

Below is a function named **forward_propagate()** that implements the forward propagation for a row of data from our dataset with our neural network.

You can see that a neuron's output value is stored in the neuron with the name '**output**'. You can also see that we collect the outputs for a layer in an array named **new_inputs** that becomes the array **inputs** and is used as inputs for the following layer.

The function returns the outputs from the last layer also called the output layer.

```
# 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
```

Let's put all of these pieces together and test out the forward propagation of our network.

We define our network inline with one hidden neuron that expects 2 input values and an output layer with two neurons.

```
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['wei
            neuron['output'] = transfer(activ
            new_inputs.append(neuron['output'
        inputs = new_inputs
    return inputs

# test forward propagation
network = [[{'weights': [0.13436424411240122,
        [{'weights': [0.2550690257394217, 0.4
row = [1, 0, None]
output = forward_propagate(network, row)
print(output)
```

Running the example propagates the input pattern [1, 0] and produces an output value that is printed. Because the output layer has two neurons, we get a list of two numbers as output.

The actual output values are just nonsense for now, but next, we will start to learn how to make the weights in the neurons more useful.

```
[0.6629970129852887, 0.7253160725279748]
```

3. Back Propagate Error

The backpropagation algorithm is named for the way in which weights are trained.

Error is calculated between the expected outputs and the outputs forward propagated from the network. These errors are then propagated backward through the network from the output layer to the hidden layer, assigning blame for the error and updating weights as they go.

The math for backpropagating error is rooted in calculus, but we will remain high level in this section and focus on what is calculated and how rather than why the calculations take this particular form.

This part is broken down into two sections.

1. Transfer Derivative.
2. Error Backpropagation.

3.1. Transfer Derivative

Given an output value from a neuron, we need to calculate it's slope.

We are using the sigmoid transfer function, the derivative of which can be calculated as follows:

```
derivative = output * (1.0 - output)
```

Below is a function named **transfer_derivative()** that implements this equation.

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

Now, let's see how this can be used.

3.2. Error Backpropagation

The first step is to calculate the error for each output n
propagate backwards through the network.

The error for a given neuron can be calculated as follo

```
error = (expected - output) * transfer_derivat
```

Where **expected** is the expected output value for the
transfer_derivative() calculates the slope of the neur

and

This error calculation is used for neurons in the output layer. The expected value is the class value itself. In the hidden layer, things are a little more complicated.

The error signal for a neuron in the hidden layer is calculated as the weighted error of each neuron in the output layer. Think of the error traveling back along the weights of the output layer to the neurons in the hidden layer.

The back-propagated error signal is accumulated and then used to determine the error for the neuron in the hidden layer, as follows:

```
error = (weight_k * error_j) * transfer_derivative(output)
```

Where **error_j** is the error signal from the **j**th neuron in the output layer, **weight_k** is the weight that connects the **k**th neuron to the current neuron and output is the output for the current neuron.

Below is a function named **backward_propagate_error()** that implements this procedure.

You can see that the error signal calculated for each neuron is stored with the name 'delta'. You can see that the layers of the network are iterated in reverse order, starting at the output and working backwards. This ensures that the neurons in the output layer have 'delta' values calculated first that neurons in the hidden layer can use in the subsequent iteration. I chose the name 'delta' to reflect the change the error implies on the neuron (e.g. the weight delta).

You can see that the error signal for neurons in the hidden layer is accumulated from neurons in the output layer where the hidden neuron number **j** is also the index of the neuron's weight in the output layer **neuron['weights'][j]**.

```
# 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
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - n
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * tra
```

Let's put all of the pieces together and see how it work

We define a fixed neural network with output values a e complete example is listed below.

```
# Calculate the derivative of an neuron outpu
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'])

# test backpropagation of error
network = [[{'output': 0.7105668883115941, 'weights': [0.13436424411240122, 0.8474337369372327,
        [{'output': 0.6213859615555266, 'weights': [0.2550690257394217, 0.49543508709194095]},
expected = [0, 1]
backward_propagate_error(network, expected)
for layer in network:
    print(layer)
```

Running the example prints the network after the backpropagation of error is complete. You can see that error values are calculated and stored in the neurons for the output layer and the hidden layer.

```
[{'output': 0.7105668883115941, 'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618
[{'output': 0.6213859615555266, 'weights': [0.2550690257394217, 0.49543508709194095], 'delta': -
```

Now let's use the backpropagation of error to train the network.

4. Train Network

The network is trained using stochastic gradient descent.

This involves multiple iterations of exposing a training dataset to the network and for each row of data forward propagating the inputs, backpropagating the e

This part is broken down into two sections:

1. Update Weights.
2. Train Network.

4.1. Update Weights

Once errors are calculated for each neuron in the netw
can be used to update weights.

Network weights are updated as follows:

```
weight = weight + learning_rate * error * inpu
```

Where **weight** is a given weight, **learning_rate** is a parameter that you must specify, **error** is the error calculated by the backpropagation procedure for the neuron and **input** is the input value that caused the error.

The same procedure can be used for updating the bias weight, except there is no input term, or input is the fixed value of 1.0.

Learning rate controls how much to change the weight to correct for the error. For example, a value of 0.1 will update the weight 10% of the amount that it possibly could be updated. Small learning rates are preferred that cause slower learning over a large number of training iterations. This increases the likelihood of the network finding a good set of weights across all layers rather than the fastest set of weights that minimize error (called premature convergence).

Below is a function named **update_weights()** that updates the weights for a network given an input row of data, a learning rate and assume that a forward and backward propagation have already been performed.

Remember that the input for the output layer is a collection of outputs from the hidden layer.

```
# 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']
```

Now we know how to update network weights, let's see how we can do it repeatedly.

4.2. Train Network

As mentioned, the network is updated using stochastic gradient descent.

This involves first looping for a fixed number of epochs and within each epoch updating the network for each row in the training dataset.

Because updates are made for each training pattern, t ors were accumulated across an epoch before updating th gradient descent.

Below is a function that implements the training of an ning dataset, learning rate, fixed number of epochs and an

The expected number of output values is used to trans ot encoding. That is a binary vector with one column for rk. This is required to calculate the error for the output lay

You can also see that the sum squared error between the expected output and the network output is accumulated each epoch and printed. This is helpful to create a trace of how much the network is learning and improving each epoch.

```
# 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))
```

We now have all of the pieces to train the network. We can put together an example that includes everything we've seen so far including network initialization and train a network on a small dataset.

Below is a small contrived dataset that we can use to test out training our neural network.

```
X1          X2          Y
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
```

Below is the complete example. We will use 2 neurons in the hidden layer. It is a binary classification problem (2 classes) so there will be two neurons in the output layer. The network will be trained for 20 epochs with a learning rate of 0.5, which is high because we are training for so few iterations.

```
from math import exp
from random import seed
from random import random

# Initialize a network
def initialize_network(n_inputs, n_hidden, n
    network = list()
    hidden_layer = [{'weights':[random() for
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for
    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

# Train a network for a fixed number of epoc
def train_network(network, train, l_rate, n_
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(netw
            expected = [0 for i in range(n_o
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-o
            backward_propagate_error(network
            update_weights(network, row, l_r
        print('>epoch=%d, lrate=%.3f, 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)
```

Running the example first prints the sum squared error each training epoch. We can see a trend of this error decreasing with each epoch.

Once trained, the network is printed, showing the learned weights. Also still in the network are output and delta values that can be ignored. We could update our training function to delete these data if we wanted.

```
>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': 0.02998030
[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275], 'output': 0.2364879
```

Once a network is trained, we need to use it to make p

5. Predict

Making predictions with a trained neural network is ea

We have already seen how to forward-propagate an ir do to make a prediction. We can use the output values th belonging to each output class.

It may be more useful to turn this output back into a cr ne class value with the larger probability. This is also call

Below is a function named **predict()** that implements this procedure. It returns the index in the network output that has the largest probability. It assumes that class values have been converted to integers starting at 0.

```
# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))
```

We can put this together with our code above for forward propagating input and with our small contrived dataset to test making predictions with an already-trained network. The example hardcodes a network trained from the previous step.

The complete example is listed below.

```
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

# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

# Test making predictions with the network
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]]
network = [[{'weights': [-1.482313569067226,
    [{'weights': [2.5001872433501404, 0.78872
for row in dataset:
    prediction = predict(network, row)
    print('Expected=%d, Got=%d' % (row[-1], p
```

Running the example prints the expected output for each record in the training dataset, followed by the crisp prediction made by the network.

It shows that the network achieves 100% accuracy on this small dataset.

```
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
```

Now we are ready to apply our backpropagation algorithm to a real world dataset.

6. Wheat Seeds Dataset

This section applies the Backpropagation algorithm to the wheat seeds dataset.

The first step is to load the dataset and convert the loaded data to numbers that we can use in our neural network. For this we will use the helper function **load_csv()** to load the file, **str_column_to_float()** to convert string numbers to floats and **str_column_to_int()** to convert the class column to integer values.

Input values vary in scale and need to be normalized to the range of 0 and 1. It is generally good practice to normalize input values to the range of the chosen transfer function, in this case, the sigmoid function that outputs values between 0 and 1. The **dataset_minmax()** and **normalize_dataset()** helper functions were used to normalize the input values.

We will evaluate the algorithm using k-fold cross-validation with 5 folds. This means that 201/5=40.2 or 40 records will be in each fold. We will use the helper functions **evaluate_algorithm()** to evaluate the algorithm with cross-validation and **accuracy_metric()** to calculate the accuracy of predictions.

A new function named **back_propagation()** was developed to manage the application of the Backpropagation algorithm, first initializing a network, [illegible]ne trained network to make predictions on a test dataset.

The complete example is listed below.

```
# Backprop on the Seeds Dataset
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
```

```
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 10

# Evaluate an algorithm using a cross valida
def evaluate_algorithm(dataset, algorithm, n
    folds = cross_validation_split(dataset,
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
```

```
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores

# 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] -
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * tr

# 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 n
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_ra
            neuron['weights'][-1] += l_rate
```

```
# 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):
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)

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

# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
    n_inputs = len(train[0]) - 1
    n_outputs = len(set([row[-1] for row in train]))
    network = initialize_network(n_inputs, n_hidden, n_outputs)
    train_network(network, train, l_rate, n_epoch, n_outputs)
    predictions = list()
    for row in test:
        prediction = predict(network, row)
        predictions.append(prediction)
    return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'seeds_dataset.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# normalize input variables
minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
# evaluate algorithm
n_folds = 5
l_rate = 0.3
n_epoch = 500
n_hidden = 5
scores = evaluate_algorithm(dataset, back_pr
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)
```

A network with 5 neurons in the hidden layer and 3 ne
network was trained for 500 epochs with a learning ra
trial and error, but you may be able to do much better.

Running the example prints the average classification accuracy on each fold as well as the average performance across all folds.

You can see that backpropagation and the chosen configuration achieved a mean classification accuracy of about 93% which is dramatically better than the Zero Rule algorithm that did slightly better than 28% accuracy.

```
Scores: [92.85714285714286, 92.85714285714286, 97.61904761904762, 92.85714285714286, 90.4761904
Mean Accuracy: 93.333%
```

Extensions

This section lists extensions to the tutorial that you may wish to explore.

- **Tune Algorithm Parameters.** Try larger or smaller networks trained for longer or shorter. See if you can get better performance on the seeds dataset.
- **Additional Methods.** Experiment with different weight initialization techniques (such as small random numbers) and different transfer functions (such as tanh).
- **More Layers.** Add support for more hidden layers, trained in just the same way as the one hidden layer used in this tutorial.
- **Regression.** Change the network so that there is only one neuron in the output layer and that a real value is predicted. Pick a regression dataset to practice on. A linear transfer function could be used for neurons in the output layer, or the output values of the chosen dataset could be scaled to values between 0 and 1.
- **Batch Gradient Descent.** Change the training procedure from online to batch gradient descent and update the weights only at the end of each epoch.

Did you try any of these extensions?
Share your experiences in the comments below.

Review

In this tutorial, you discovered how to implement the Backpropagation algorithm from scratch

Specifically, you learned:

- How to forward propagate an input to calculate a
- How to back propagate error and update network
- How to apply the backpropagation algorithm to a

Do you have a
Ask your questions in the comments b

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

‹ How To Implement Learning Vector Quantization From Scratch With Python

How To Implement The Decision Tree Algorithm From Scratch In Python ›

373 Responses to *How to Implement the Scratch In Python*

Talk Data To Me November 7, 2016 at 9:28 pm #

That's what I was looking for. Write a neural n
you very much!

Jason Brownlee November 8, 2016 at 9:51 am #

REPLY

I'm glad to hear it!

WB February 20, 2018 at 3:07 pm #

REPLY

I experienced the following applying the Backpropagation algorithm to the wheat seeds dataset. I am wondering how to resolve the errors? Thank you

---

ValueError Traceback (most recent call last)
in ()
184 dataset = load_csv(filename)
185 for i in range(len(dataset[0])-1):
–> 186 str_column_to_float(dataset, i)
187 # convert class column to integers
188 str_column_to_int(dataset, len(dataset[0])-1)

in str_column_to_float(dataset, column)
20 def str_column_to_float(dataset, column):
21 for row in dataset:
—> 22 row[column] = float(row[column].strip())
23
24 # Convert string column to integer

ValueError: could not convert string to float:

Jason Brownlee February 21, 2018 at 6:35 am #

REPLY

Are you using Python 2?

wb February 21, 2018 at 2:51 pm #

Yes I am

harshith October 5, 2018 at 8:28 pm #

hi bro whass up

Mike Harney March 5, 2018 at 9:53 am #

REPLY

Hi wb, I'm on 3.6 and I found the same issue. Maybe you can answer this Jason, but it looks like the some of the data is misaligned in the sample. When opened in Excel, there are many open spaces followed by data jutted out to an extra column. I assume this is unintentional, and when I corrected the spacing, it appeared to work for me.

Jason Brownlee March 6, 2018 at 6:08 am #

REPLY

The code was written and tested with Python 2.7.

JU April 23, 2018 at 7:24 am #

Mike is right – the dataset from the UCI website is slightly defective: It has two tabs in some places where there should be only one. This needs to be corrected during the conversion to CSV. In Excel the easiest way is to use the text importer and then click the "Treat consecutive delimiters as one" checkbox.

Jason Brownlee April 23, 2018 at 7:37 am #

Here is the dataset ready to use:
https://raw.githubusercontent.com/jbrownlee/Datasets/master/wheat-seeds.csv

Alexis Batyk August 29, 2018 at 6:22 am #

REPLY

[SOLVED]
i have the same issue with

https://raw.githubusercontent.com/jbrownlee/D

there is still dirty that csv

use a text editor -> select search and replace t

Jason Brownlee August 29, 2018 a

I don't have such problems on Py

Jackson Scott October 1, 2018 at 9:08 am #

REPLY

thanks, this worked for me as well. The csv file had some tabbed over and others correct.

Deng October 14, 2018 at 5:50 pm #

REPLY

The data in the seeds_dataset file contains the backspace key, and it is ok to reset the data

MO November 8, 2016 at 9:26 am #

REPLY

where can i see your data set, i want to see how it looked like

Jason Brownlee November 8, 2016 at 10:01 am #

REPLY

Hi MO.

The small contrived dataset used for testing is listed inline in the post in section 4.2

The dataset used for the full example is on the UCI ML repository, linked in the section titled “Wheat Seeds Dataset”. Here is the direct link:
http://archive.ics.uci.edu/ml/datasets/seeds

prakash November 11, 2016 at 12:40 am #

REPLY

in two class classification for 0 the expected value is [1,0] for 1 its is [0,1].
how will be the output vectors for more than two class?

Jason Brownlee November 11, 2016 at 10:02

Hi prakash,

For multi-class classification, we can extend the or

Three class values for “red”, “green” “blue” can be
1, 0, 0 for red
0, 1, 0 for green
0, 0, 1 for blue

I hope that helps.

Rakesh November 13, 2016 at 3:41 pm #

REPLY

Hi, Jason.
You've mentioned that there are 3 output classes.
How do we check the values which come under the 3 classes / clusters?
Could we print the data which fall under each class?

Jason Brownlee November 14, 2016 at 7:35 am #

REPLY

Hi Rakesh,

The data does belong to 3 classes. We can check the skill of our model by comparing the predicted classes to the actual/expected classes and calculate an accuracy measure.

Alex November 16, 2016 at 12:35 pm #

REPLY

I'm confused why the activation method iterates from 0 to len(inputs) – 1 instead of from 0 to len(weights) – 1. Am I missing something?

Jason Brownlee November 17, 2016 at 9:47 am #

REPLY

Hi Alex,

The length of weights is the length of the input + 1 (to accommodate the bias term).

We add the bias term first, then we add the weighte[illegible]

Does that help?

Alex November 17, 2016 at 12:29 pm #

When I step through the code above [illegible] correctly generate the output for the single hid[illegible] processed when determining the outputs for th[illegible] function 'for i in range(len(inputs)-1):', when th[illegible] output node for class=0, since 'inputs' has a si[illegible] node), 'len(inputs) – 1' equals 0 so the for loop[illegible] to read 'for i in range(len(weights) -1):' Does th[illegible]

I'm just trying to make sure I don't fundamentally misunderstand something and improve this post for other readers. This site has been really, really helpful for me.

Jason Brownlee November 18, 2016 at 8:27 am #

REPLY

I'm with you now, thanks for helping me catch-up.

Nice spot. I'll fix up the tutorial.

Update: Fixed. Thanks again mate!

Tomasz Panek November 21, 2016 at 1:23 am #

REPLY

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

In this fragment:

```
for j in range(len(inputs)-1):
neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
neuron['weights'][-1] += l_rate * neuron['delta']
```

If inputs length = 1, you are not updating weights, it's correct? You are updating only bias, because in hidden layer is only one neuron.

Tomasz November 21, 2016 at 1:34 am #

Hello. In method update_weight you are doing
you aren't updating weights. It's correct? Hidden layer
updated

Jason Brownlee November 22, 2016 at 6:54

Hi Tomasz,

The assumption here is that the input vector always contains at least one input value and an output value, even if the output is set to None.

You may have found a bug though when updating the layers. I'll investigate and get back to you.

Jason Brownlee January 3, 2017 at 10:17 am #

REPLY

Thanks Tomasz, this was indeed a bug.

I have updated the update_weights() function in the above code examples.

Jerry Jones October 16, 2018 at 8:18 am #

REPLY

I don't understand how update_weights updates the NN. There is no global variable or return from the function. What am I missing?

Jason Brownlee October 16, 2018 at 2:33 pm #

The weights are passed in by reference and modified in place.

This is an advanced tutorial, I'd recommend using Keras for beginners.

Michael December 13, 2016 at 4:15 am #

REPLY

Hi, Thanks for the tutorial, I'm doing a backpropagation project at the moment so its been really useful.

I was a little confused on the back-propagation error calculation function. Does "if i != len(network)-1:" mean that if the current layer isn't the output layer then this fo
layer is an output layer?

Jason Brownlee December 13, 2016 at 8:08

Glad to hear it Michael.

The line means if the index i is not equal to the ind
then run code inside the block.

Michael January 5, 2017 at 7:53 am #

REPLY

I have another question.
Would it be possible to extend the code from this tutorial and create a network that trains using the MNIST handwritten digit set? using a input unit to represent each pixel in the image. I'm also not sure whether/how I could use feature extractors for the images.

I have a project where I have to implement the Backpropagation algorithm with possibly the MNIST handwritten digit training set.

I hope my question makes sense!

Jason Brownlee January 5, 2017 at 9:42 am #

REPLY

Sure Michael, but I would recommend using a library like Keras instead as this code is not written for performance.

Load an image as a long list of pixel integer values, convert to floats and away you go. No feature extraction needed for a simple MLP implementation. You should get performance above 90%.

Calin January 6, 2017 at 10:40 pm #

REPLY

Hi Jason,

Great post!

I have a concern though:

In train_network method there are these two lines of code:

```
expected = [0 for i in range(n_outputs)]
expected[row[-1]] = 1
```

Couldn't be the case that expected[row[-1]] = 1 will thro
set which is a subset of the dataset and row basically c

Jason Brownlee January 7, 2017 at 8:37 am

Hi Calin,

If I understand you correctly, No. The n_outputs va
values.

Maybe put some print() statements in to help you b

Calin January 7, 2017 at 9:48 pm #

REPLY

Hmm..I ran the entire code (with the csv file downloaded from http://archive.ics.uci.edu/ml/datasets/seeds), added some breakpoints and this is what I got after a few iterations:

n_outputs = 168
row[-1] = 201

which is causing IndexError: list assignment index out of range.

Adriaan January 11, 2017 at 4:27 am #

REPLY

I've got the same error, That my list assignment index is out of range

Jason Brownlee January 11, 2017 at 9:29 am #

Sorry to hear that, did you try running the updated code?

Ivan January 16, 2017 at 10:28 am #

This is error of csv read. Try to reformat it with commas. For me it worked

Jason Brownlee January 16, 2017 at 10:45 am #

What was the problem and fix exactly Ivan?

Bob February 5, 2017 at 10:59 am

The data file (http://archive.ic
databases/00236/seeds_dataset.txt) h
— removing the double tabs and chan

Thanks for the good article.

Jason Brownlee February 6, 2

Thanks for the note Bob.

Rowen Bruce October 20, 2018 at 8:52 pm #

updated code

Adriaan January 11, 2017 at 5:50 am #

REPLY

I've had the same error at the 'train_network' function. Is your dataset fine? I've had some problems because the CSV file wasn't loaded correctly due to my regional windows settings. I've had to adjust my settings and everything worked out alright.

http://superuser.com/questions/783060/excel-save-as-csv-options-possible-to-change-comma-to-pipe-or-tab-instead

Stanley January 8, 2017 at 3:15 pm #

REPLY

Thanks for such a good article.

Just one question: in the equation "weight = weight + learning_rate * error * input", why there is an "input"? IMO it should be: "weight = weight + learning_rate * error"?

Jason Brownlee January 9, 2017 at 7:47 am #

REPLY

The var names and explanation are correct.

The update equation is:

```
1 weight = weight + learning_rate * error
```

For the input layer the input are the input data, for

Madwadasa January 13, 2017 at 3:31 am #

Jason,

Thanks for the code and post.
Why is "expected" in expected = [0 for i in range(n_out
Should not the o/p values be taken as expected when
i.e for example in case of Xor should not 1 be taken as

Jason Brownlee January 13, 2017 at 9:16 am #

REPLY

Hi Madwadasa,

Expected is a one-hot encoding. All classes are “0” expect the actual class for the row which is marked as a “1” on the next line.

Michael January 19, 2017 at 3:44 am #

REPLY

Hello, I have a couple more questions. When training the network with a dataset, does the error at each epoch indicate the distance between the predicted outcomes and the expected outcomes together for the whole dataset? Also when the mean accuracy is given in my case being 13% when I used the MNIST digit set, does this mean that the network will be correct 13% of the time and would have an error rate of 87%?

Jason Brownlee January 19, 2017 at 7:38 am #

REPLY

Hi Michael,

The epoch error does capture how wrong the algorithm is on all training data. This may or may not be a distance depending on the error measure used. RMSE is technically not a distance measure, you could use Euclidean distance if you like, but I would not recommend it.

Yes, in generally when the model makes predictions your understanding is correct.

Bernardo Galvão January 24, 2017 at 3:51 am #

REPLY

Hi Jason,

in the excerpt regarding error of a neuron in a hidden l

“Where error_j is the error signal from the jth neuron in
the kth neuron to the current neuron and output is the

is the k-th neuron a neuron in the output layer or a neu
current neuron, are you referring to the neuron in the c

Appreciate your work!

Bernardo

anonymous February 1, 2017 at 1:42 am #

It would have been better if recall and precision were printed. Can somebody tell me how to print them in the above code.

Jason Brownlee February 1, 2017 at 10:51 am #

REPLY

You can learn more about precision and recall here:
https://en.wikipedia.org/wiki/Precision_and_recall

kehinde kolade February 6, 2017 at 8:29 pm #

REPLY

Hello Jason, great tutorial, I am developer and I do not really know much about this machine learning thing but I need to extend this your code to incorporate the Momentum aspect to the training, can you please explain how I can achieve this extension?

Jason Brownlee February 7, 2017 at 10:14 am #

REPLY

Sorry, I don't have the capacity to write or spell out this change for you.

My advice would be to read a good book on the topic, such as Neural Smithing: http://amzn.to/2Id9ds0

ibrahim February 18, 2017 at 2:21 am #

REPLY

Hi Jason,
I have my own code written in C++, which works similar to your code. My intention is to extend my code to convolutional deep neural nets, and i have actually written the convolution, Relu and pooling functions however i could not begin to apply the backpropagation i have used in my shallow neural net, to the convolutional deep net, cause i really cant imagine the
the convolutional layers and the standard shallow laye
source for this issue however i always come to the poi
given for shallow nets that i applied already. Can you p

Jason Brownlee February 18, 2017 at 8:42 a

I"d love to guide you but I don't have my
not best placed to help at the moment.

I'd recommend reading code from existing open so

Good luck with your project.

matias February 22, 2017 at 3:34 pm #

REPLY

Thank you, I was looking for exactly this kind of ann algorith. A simple thank won't be enough tho lol

Jason Brownlee February 23, 2017 at 8:52 am #

REPLY

I'm glad it helped.

The best way to help is to share the post with other people, or maybe purchase one of my books to support my ongoing work:
http://machinelearningmastery.com/products

Manohar Katam February 26, 2017 at 3:40 pm #

REPLY

Great one! .. I have one doubt .. the dataset seeds contains missing features/fields for some rows.. how you are handling that …

Jason Brownlee February 27, 2017 at 5:49 am #

REPLY

You could set the missing values to 0, you could remove the rows with missing values, you could impute the missing values with mean column values, etc.

Try a few different methods and see what results in the best performing models.

Manohar Katam March 1, 2017 at 2:59 p

What if I have canonical forms like "m
work even with string data..

Jason Brownlee March 2, 2017 at 8

Hi Manohar,

No, you will need to convert them to intege

Wissal ARGOUBI February 27, 2017 at 11:12 pm #

REPLY

Great job! this is what i was looking for ! thank you very much .
However i already have a data base and i didn't know how to make it work with this code how can i adapt it on my data
Thank you

Jason Brownlee February 28, 2017 at 8:10 am #

REPLY

This process will help you work through your predictive modeling problem:
http://machinelearningmastery.com/start-here/#process

Shweta Gupta March 5, 2017 at 4:37 am #

REPLY

Thanks for such a great article..
I have one question, in update_weights why you have used weight=weight+l_rate*delta*input rather than weight=weight+l_rate*delta?

Jason Brownlee March 6, 2017 at 10:55 am #

REPLY

You can learn more about the math in the book on the topic.

I recommend Neural Smithing: http://amzn.to/2ld9ds0

Sittha March 13, 2017 at 1:23 pm #

REPLY

Thanks for a good tutorial.
I have some IndexError: list assignment index out of ra
separator.

Jason Brownlee March 14, 2017 at 8:11 am #

What is the full error you are getting?

Did you copy-paste the full final example and run it

Sittha March 24, 2017 at 3:36 am #

line 151 :
expected[row[-1]] = 1
IndexError : list assignment index out of range

Jason Brownlee March 24, 2017 at 8:00 am #

REPLY

Is this with a different dataset?

Benji Weiss May 11, 2017 at 5:31 am #

if it is a different dataset, what do i need to do to not get this error

Karan March 16, 2017 at 6:26 pm #

REPLY

The dataset that was given was for training the network. Now how do we test the network by providing the 7 features without giving the class label(1,2 or 3) ?

Jason Brownlee March 17, 2017 at 8:27 am #

REPLY

You will have to adapt the example to fit the model on all of the training data, then you can call predict() to make predictions on new data.

Karan March 19, 2017 at 7:43 pm #

REPLY

Ok Jason, i'll try that and get back to

Karan March 19, 2017 at 7:48 pm #

Just a suggestion for the people who would b
training their network, make sure you add an IF loop a
if minmax[i][1]!=minmax[i][0]

This is because your own dataset might contain same
divide by zero error.

Jason Brownlee March 20, 2017 at 8:16 am #

REPLY

Thanks for the tip Karan.

Li Qun March 25, 2017 at 5:45 pm #

REPLY

Thanks jason for the amazing posts of your from scratch pyhton implementations! i have learned so much from you!

I have followed through both your naive bayes and backprop posts, and I have a (perhaps quite naive) question:

what is the relationship between the two? did backprop actually implement bayesian inference (after all, what i understand is that bayesian = weights being updated every cycle) already? perhaps just non-gaussian? so.. are non-gaussian PDF weight updates not bayesian inference?

i guess to put it simply : is backpropagation essentially a bayesian inference loop for an n number of epochs?

I came from the naive bayes tutorial wanting to implement backpropagation together with your naive bayes implementation but got a bit lost along the way.

sorry if i was going around in circles, i sincerely hope someone would be able to at least point me on the right direction.

Jason Brownlee March 26, 2017 at 6:11 am #

REPLY

Great question.

No, they are both very different. Naive bayes is a direct use of the probabilities and bayes theorem. The neural net is approximating a mapping function from inputs and outputs – a very different approach that does not directly use the joint probability.

Chiraag March 26, 2017 at 10:10 pm #

How did you decide that the number of folds v f this number. Thank You.

Jason Brownlee March 27, 2017 at 7:54 am #

In this case, it was pretty arbitary.

Generally, you want to split the data so that each fold is representative of the dataset. The objective measure is how closely the mean performance reflect the actual performance of the model on unseen data. We can only estimate this in practice (standard error?).

Li Qun March 27, 2017 at 10:19 pm #

REPLY

Dear Jason,

thank you for the reply! I read up a bit more about the differences between Naive Bayes (or Bayesian Nets in general) and Neural Networks and found this Quora answer that i thought was very clear. I'll put it up here to give other readers a good point to go from:

https://www.quora.com/What-is-the-difference-between-a-Bayesian-network-and-an-artificial-neural-network

TL:DR :
– they look the same, but every node in a Bayesian Network has meaning, in that you can read a Bayesian network structure (like a mind map) and see what's happening where and why.
– a Neural Network structure doesn't have explicit meaning, its just dots that link previous dots.
– there are more reasons, but the above two highlighted the biggest difference.

Just a quick guess after playing around with backpropagation a little: the way NB and backprop NN would work together is by running Naive Bayes to get a good 'first guess' of initial weights that are then run through and Neural Network and Backpropagated?

Jason Brownlee March 28, 2017 at 8:23 am #

REPLY

Please note that a Bayesian network and naive bayes are very different algorithms.

Melissa March 27, 2017 at 10:54 pm #

REPLY

Hi Jason,
Further to this update:

Update Jan/2017: Changed the calculation of fold_size
Fixes issues with Python 3.

I'm still having this same problem whilst using python 3
an error at line 75 saying 'list object has no attribute 'su

Any help would be very much appreciated.
Overall this code is very helpful. Thank you!

Jason Brownlee March 28, 2017 at 8:24 am #

Sorry to hear that, did you try copy-paste the complete working example from the end of the post and run it on the same dataset from the command line?

Melissa March 28, 2017 at 9:29 am #

REPLY

Yes I've done that, but still the same problem!

david March 29, 2017 at 6:16 am #

REPLY

Hello jason,

please i need help on how to pass the output of the trained network into a fuzzy logic system if possible a code or link which can help understand better. Thank you

Aditya April 2, 2017 at 3:57 pm #

REPLY

Awesome Explanation

Jason Brownlee April 4, 2017 at 9:05 am #

REPLY

Thanks!

Raunak Jain April 6, 2017 at 5:20 pm #

REPLY

Hello Jason

I m getting list assignment index out or range error. Ho

Jason Brownlee April 9, 2017 at 2:37 pm #

The example was developed for Python 2

Marco April 6, 2017 at 9:37 pm #

Thanks but I think python is not a good choice

Jason Brownlee April 9, 2017 at 2:40 pm #

REPLY

I think it is a good choice for learning how backprop works.

What would be a better choice?

Agrawal April 6, 2017 at 9:38 pm #

REPLY

Hey, Jason Thanks for this wonderful lecture on Neural Network.

As I am working on Iris Recognition, I have extracted the features of each eye and store it in .csv file, Can u suggest how further can I build my Backpropagation code.
As when I run your code I am getting many errors.
Thank you

Jason Brownlee April 9, 2017 at 2:40 pm #

REPLY

This process will help you work through your modeling problem:
http://machinelearningmastery.com/start-here/#process

Jack April 7, 2017 at 3:42 pm #

REPLY

Could you please convert this iterative implementation into matrix implementation?

Jason Brownlee April 9, 2017 at 2:52 pm #

REPLY

Perhaps in the future Jack.

Jk April 12, 2017 at 5:04 am #

Hi Jason,

In section 4.1 , may you please explain why you used

Thanks

Jason Brownlee April 12, 2017 at 7:58 am #

REPLY

Yes. By default we are back-propagating the error of the expected output vs the network output (inputs = row[:-1]), but if we are not the output layer, propagate the error from the previous layer in the network (inputs = [neuron['output'] for neuron in network[i – 1]]).

I hope that helps.

JK April 13, 2017 at 3:59 am #

REPLY

Thanks for your respond. I understand what you said , the part I am no understanding is the [:-1] . why eliminating the last list item ?

Jason Brownlee April 13, 2017 at 10:10 am #

REPLY

It is a range from 0 to the second last item in the list, e.g. (0 to n-1)

Amer April 6, 2018 at 7:22 am #

REPLY

Because the last Item in the weights array is the biass

Prem Puri April 12, 2017 at 8:18 pm #

REPLY

In function call, def backward_propagate_error(network, expected):
how much i understand is , it sequentially pass upto
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'])
My question is which value is used in neuron['delta']

Jason Brownlee April 13, 2017 at 10:01 am #

delta is set in the previous code block. It i

Nishu March 25, 2018 at 11:32 am #

REPLY

I'm sorry, but I still can't find the location where delta is set and hence, the code gives error. Where is the delta set for the first time?

Prem Puri April 14, 2017 at 3:20 am #

REPLY

Thanks very much!

Jason Brownlee April 14, 2017 at 8:54 am #

REPLY

You're welcome.

youssef oumate April 26, 2017 at 4:53 pm #

REPLY

Hi Jason

Thank you very much for this awesome implementation of neural network,
I have a question for you : I want to replace the activation function from Sigmoid to RELU . So, what are the changes that I should perform in order to get correct predictions?

Jason Brownlee April 27, 2017 at 8:34 am #

REPLY

I think just a change to the transfer() and transfer_derivative() functions will do the trick.

youssef oumate April 27, 2017 at 10:17

Awesome !

Thank you so much

Jason Brownlee April 28, 2017 at 7

You're welcome.

Yahya Alaa April 30, 2017 at 2:38 am #

REPLY

Hi Jason,
Thank you very much for this wonderful implementation of Neural Network, it really helped me a lot to understand neural networks concept,

```
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)
```

What do n_inputs and n_outputs refer to? According to the small dataset used in this section, is n_inputs only 2 and n_outputs only 2 (0 or 1) or I am missing something?

Jason Brownlee April 30, 2017 at 5:31 am #

REPLY

Input/outputs refers to the number of input and output features (columns) in your data.

Yahya Alaa May 3, 2017 at 1:42 pm #

REPLY

Is the program training the network for 500 epochs for each one of the k-folds and then testing the network with the testing data set?

Jason Brownlee May 4, 2017 at 8:02 am #

REPLY

Hi Yahya,

5-fold cross validation is used.

That means that 5 models are fit and evaluated on 5 different hold out sets. Each model is trained for 500 epochs.

I hope that makes things clearer Yahya.

Yahya Alaa May 4, 2017 at 8:17 am #

Yes you made things clear to me,
I have two other questions,
How to know when to stop training the net
How to choose the number of neurons in t

Jason Brownlee May 5, 2017 at 7:27 am #

You can use early stopping, to save network weights when the skill on a validation set stops improving.

The number of neurons can be found through trial and error.

Yahya Alaa May 6, 2017 at 8:48 am #

I am working on a program that recognizes handwritten digits, the dataset is consisting of pictures (45*45) pixels each, which is 2025 input neurons, this causes me a problem in the activation function, the summation of (weight[i] * input[i]) is big, then it gives me always a result of (0.99 -> 1) after putting the value of the activation function in the Sigmoid function, any suggestions?

Jason Brownlee May 7, 2017 at 5:31 am #

I would recommend using a Convolutional Neural Network rather than a Multilayer Perceptron.

morok April 30, 2017 at 3:56 am #

REPLY

In section 3.2. Error Backpropagation, where did output numbers came from for testing backpropagation

'output': 0.7105668883115941
'output': 0.6213859615555266
'output': 0.6573693455986976

Perhaps from outputs on test forward propagation [0.6
> derivative = output * (1.0 – output), problem is they d

thanks!

Awesome article!!!

Jason Brownlee April 30, 2017 at 5:34 am #

In that example, the output and weights w
the "delta" in those outputs.

Massa November 25, 2017 at 7:36 am #

REPLY

hello Dr Jason…

I was wondering …

n_outputs = len(set([row[-1] for row in dataset]))

this line, how does it give the number of output features?
when I print it gives the number of the dataset(number of rows, not columns)

Jason Brownlee November 25, 2017 at 10:25 am #

REPLY

The length of the set of values in the final column.

Perhaps this post will help with Python syntax:
https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/

Massa November 26, 2017 at 4:02 am #

REPLY

but I thought it gives the number of outputs…I mean the number of neurons in the output layer.

here it's giving the number of the dataset ….if I have 200 input/output pairs it prints 200

so I am confused…how would it be?

Jason Brownlee November 26, 2017 at 7:35 am #

If there are two class values, it should print 2. It should not print the number of examples.

Umamaheswaran May 8, 2017 at 9:49 pm #

Hi Jason,

I am using the MNIST data set to implement a handwri
be needed to get a perfomance above 90%.

Jason Brownlee May 9, 2017 at 7:42 am #

I would recommend using a CNN on MNIST. See this tutorial:
http://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

Huyen May 9, 2017 at 6:32 pm #

REPLY

Hi Jason,

Your blog is totally awesome not only by this post but also for the whole series about neural network. Some of them explained so much useful thing than others on Internet. They help me a lot to understand the core of network instead of applying directly Keras or Tensorflow.

Just one question, if I would like to change the result from classification to regression, which part in back propagation I need to change and how?

Thank you in advance for your answer

Jason Brownlee May 10, 2017 at 8:46 am #

REPLY

Thanks Huyen.

You would change the activation function in the output layer to linear (e.g. no transform).

TGoritsky May 12, 2017 at 12:41 am #

REPLY

Hi Jason,

I am playing around with your code to better understand how the ANN works. Right now I am trying to do predictions with a NN, that is trained on my own dataset, but the program returns me one class label for all rows in a test dataset. I understand, that normalizing dataset should help, but it doesn`t work (I am using your minmax and normalize_dataset functions). Also, i
dataset?
Here is the code (sorry for lack of formatting):
def make_predictions():
dataset = [[29,46,107,324,56,44,121,35,1],
[29,46,109,327,51,37,123,38,1],
[28,42,107,309,55,32,124,38,1],
[40,112,287,59,35,121,36,1],
[27,43,129,306,75,41,107,38,1],
[28,38,127,289,79,40,109,37,1],
[29,37,126,292,77,35,100,34,1],
[30,40,87,48,77,51,272,80,2],
[26,37,88,47,84,44,250,80,2],
[29,39,91,47,84,46,247,79,2],

```
[28,38,85,45,80,47,249,78,2],
[28,36,81,43,76,50,337,83,2],
[28,34,75,41,83,52,344,81,2],
[30,38,80,46,71,53,347,92,2],
[28,35,72,45,64,47,360,101,2]]
network = [[{'weights': [0.09640510259345969, 0.37923370996257266, 0.5476265202749506,
0.9144446394025773, 0.837692750149296, 0.5343300438262426, 0.7679511829130964,
0.5325204151469501, 0.06532276962299033]}],
[{'weights': [0.040400453542770665, 0.13301701225112483]}, {'weights': [0.1665525504275246,
0.5382087395561351]}, {'weights': [0.26800994395551214, 0.3322334781304659]}]]
# minmax = dataset_minmax(dataset)
# normalize_dataset(dataset, minmax)
for row in dataset:
prediction = predict(network, row)
print('Expected=%d, Got=%d' % (row[-1], prediction))
```

Jason Brownlee May 12, 2017 at 7:43 am #

REPLY

I would suggest exploring your problem with the Keras framework:
http://machinelearningmastery.com/start-here/#deeplearning

Tomo May 18, 2017 at 6:22 pm #

REPLY

Hi Jason!
In the function "backward_propagate_error", when you do this:

neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

The derivative should be applied on the activation of that neuron, not to the output . Am I right??

neuron['delta'] = errors[j] * transfer_derivative(activate(neuron['weights'], inputs))

And inputs is:
inputs = row[-1]
if i != 0:
inputs = [neuron['output'] for neuron in self.network[i-1]

Thank you! The post was really helpful!

Tina May 26, 2017 at 3:49 am #

Hello Jason!

This is a very interesting contribution to the community 🙂
Have you tried using the algorithm with other activation functions?
I tried with Gaussian, tanh and sinx, but the accuracy was not that high, so I think that I omitted something. What I altered were the activation functions and the derivatives. Is there something else that needs to be changed?

Jason Brownlee June 2, 2017 at 11:49 am #

REPLY

Sigmoid was the defacto standard for many years because it performs well on many different problems.

Now the defacto standard is ReLU.

Manu June 6, 2017 at 8:50 pm #

REPLY

Sigmoid and ReLU are transfer functions right ?
Activation function is just the sum of all weights and inputs

Jason Brownlee June 7, 2017 at 7:12 am #

REPLY

You are correct, but in some frameworks, transfer functions are called activation functions:
https://keras.io/activations/

vishwanathan May 27, 2017 at 8:08 pm #

REPLY

Thanks for the great post. Here is some obser
ward propagate you are not taking all the weights and
understand. I was under the impression that the delta f
for neuron in network[i + 1]:
error += (neuron['weights'][j] * neuron['delta'])

vishwanathan May 27, 2017 at 8:14 pm #

I understand you do not want to take in th
neuron. I kind of get stumped on bias.

vishwanathan May 27, 2017 at 9:12 pm #

REPLY

Thanks for the great article. In the backward propagate, the delta value is applied for each weight across the neuron and the error is summed. I am curious why is the delta not applied to individual weights of the neuron and the error summed for that neuron. Can you please clarify?

Josue May 29, 2017 at 3:12 am #

REPLY

Why don't you split the data into TrainData and TestData, like 80% of the dataset for training and 20% for testing, because if you train with 100% of rows of the dataset and then test some rows of the dataset the accuracy will be good . But if you put new data on the seeds.csv the model will work with less accuracy, Right?

Jason Brownlee June 2, 2017 at 12:16 pm #

REPLY

You can, k-fold cross validation generally gives a better estimate of model performance.

Once we have the estimate and choose our model, we can fit the final model on all available data and make predictions on new data:
http://machinelearningmastery.com/train-final-machine-learning-model/

Josue May 29, 2017 at 11:08 am #

REPLY

Thanks for the post! I have a question about cross-validation. The dataset of seeds is perfect for 5 folds but for a dataset of 211? I'll have uniformly sized subset right? (211/5) Can you give me a suggestion how I could handle that ?
Thanks in advanced.

Jason Brownlee June 2, 2017 at 12:20 pm #

One way is that some records can be disc

Sebastián May 30, 2017 at 9:35 am #

Thanks so much for the tutorial. It was really h

Jason Brownlee June 2, 2017 at 12:31 pm #

REPLY

I'm glad it helped.

Manu June 10, 2017 at 9:00 pm #

REPLY

Hello Jason,

any advice on how to handle multi-classifier problems when the classes have high cardinality ?
I'm thinking about input data of search engines linked to choosen urls.

Jason Brownlee June 11, 2017 at 8:25 am #

REPLY

Ouch, consider modeling it as regression instead (e.g. a rating or recommender system).

Manuel June 13, 2017 at 1:17 am #

REPLY

Ok thank you very much Jason.
But it wont work with searches unseen by the algorithm.
I red something in the books "Programming collective intelligence" about a neural net from scratch for this king of problem but I don't understang how it works for the moments…

Jason Brownlee June 13, 2017 at 8:23 am #

REPLY

Consider focusing on one measure/metric that really matters in your domain, then try a suite of framings of the problem and different algorithms to get a feeling for what might work best.

Yash June 18, 2017 at 6:21 pm #

I am not able to understand the above code.S

Jason Brownlee June 19, 2017 at 8:43 am #

Which part do you not understand exactly

Tathagat June 21, 2017 at 3:20 pm #

REPLY

Hey Jason..am a novice in machine learning..have a small question…how can I track the timesteps involved in the algorithm with accordance with the code?

Jason Brownlee June 22, 2017 at 6:04 am #

REPLY

What do you mean by time steps?

bazooka June 29, 2017 at 6:52 am #

REPLY

Hi, Jason. I am so confused, in the result, why there are 4 set of [output,weight,delta]

like this:
[{'output': 0.9999930495852168, 'weights': [0.9315463130784808, 1.0639526745114607, 0.9274685127907779], 'delta': -4.508489650980804e-09}, {'output': 0.9992087809233077, 'weights': [-2.4595353900551125, 5.153506472345162, -0.5778256160239431], 'delta': 1.940550145482836e-06}]
[{'output': 0.01193860966265472, 'weights': [2.3512725698865053, -8.719060612965613, 1.944330467290268], 'delta': -0.0001408287858584854}, {'output': 0.988067899681387, 'weights': [-2.2568526798573116, 8.720113230271012, -2.0392501730513253], 'delta': 0.0001406761850156443}]

after the backpropagation we find the optimal weights to get minimum error, what does these 4 group means?
E

Jason Brownlee June 29, 2017 at 7:48 am #

REPLY

That is the internal state of the whole trair

hassan June 29, 2017 at 7:30 am #

hi Jason
thanks for your code and good description here, i like i
i run your example code and encounter with an error s
the error is:
expected[row[-1]] = 1
IndexError: list assignment index out of range

how i can fix this error?

Jason Brownlee June 29, 2017 at 7:49 am #

REPLY

The code was written for Python 2.7, confirm that this is your Python version.

Also confirm that you have copied the code exactly.

Jerome July 5, 2017 at 9:20 pm #

REPLY

Dear Jason,

i have this question about Back Propagate Error

1- derivative sigmoid = output * (1.0 – output)
That is ok

2- error = (expected – output) * transfer_derivative(output)
Ok but it also means that error == 0 for output = 1 whatever the expected is because transfer_derivative(1) ==0

So, whatever the expected , error is nil if output is 1 …
Is there something rotten here?

Thanks

Jerome

wddddds July 10, 2017 at 10:01 pm #

REPLY

Thank you Jason, It’s a great tutorial and really helpful for me!

But I have to say that trying to reimplement your code strongly increased my ability of debugging 🙂

Jason Brownlee July 11, 2017 at 10:32 am #

Thanks.

Victor July 17, 2017 at 7:50 pm #

Hi Jason,

Thanks for sharing your code. I’m a PhD candidate in
weights update in section 4.1:

weight = weight + learning_rate * error * input

Should not it be as follows?

weight = weight – learning_rate * error * input

Thanks again for sharing this.

Regards,
Victor.

Víctor August 4, 2017 at 11:07 pm #

REPLY

I didn't say anything, my mistake in understanding.

Thanks again for sharing your work.

vishnu priya July 22, 2017 at 4:26 pm #

REPLY

Hi..
Thanks for ur coding. It was too helpful. can u suggest me how to use this code for classifying tamil characters. i have tried in cnn and now i need to compare the result with bpn. can u pls suggest me.

thank you

Jason Brownlee July 23, 2017 at 6:20 am #

REPLY

Perhaps this tutorial on classifying with a CNN would be more useful to you:
http://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

vishnu priya July 23, 2017 at 4:06 pm #

Thank you sir. With this tutorial i have implem
687.203 sir. i dnt know what to do sir. can u help me si

Thank you

Jason Brownlee July 24, 2017 at 6:49 am #

What is the problem exactly?

Vishnupriya July 24, 2017 at 4:53 pm #

REPLY

Classification of Tamil characters sir. I have 144 different classes. I have taken 7 glcm features of each character and I need to train this features in backpropagation and predict the character to which class it belongs.

Jason Brownlee July 25, 2017 at 9:34 am #

REPLY

Sound like a great project!

codeo July 26, 2017 at 5:37 pm #

REPLY

Hi, so I wasn't following this tutorial when implementing my neural network from scratch, and mine is in JavaScript. I just need help with the theory. How do I calculate the error for each node in the net so that I can incrementally change the weights? Great tutorial btw

codeo July 26, 2017 at 6:38 pm #

REPLY

Hahaha nevermind, it was my code
Multidimensional arrays and stuff boggle the mind hah

Jason Brownlee July 27, 2017 at 7:56 am #

REPLY

Glad to hear you worked it out.

PRABHAKARAN M July 31, 2017 at 4:31 pm #

[6.38491205 5.333345 4.81565798 5.435522
6.19728301 0 1]
Dear sir,
the above mentioned numerical values are extracted fr
occurrence matrix [10 inputs and 1 output]. This datase
same data set as[.csv] file can be used as the input for
and can i get the output as image ? for example if i giv
to get the caries affected teeth as the output for the giv

Jason Brownlee August 1, 2017 at 7:51 am #

REPLY

That sounds like a great problem. It may be possible.

I would recommend using deep CNNs.

Perhaps this tutorial will give you some ideas on how to get started:
http://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

You may want to look at some papers on object localization in images. I don't have material on it sorry.

PRABHAKARAN M July 31, 2017 at 4:32 pm #

REPLY

can i get the example code for dental caries detection using deep Convolutional Neural Network for the given dataset as x ray images.

Jason Brownlee August 1, 2017 at 7:52 am #

REPLY

I do not have sample code for this problem, sorry.

John August 1, 2017 at 3:26 am #

REPLY

Very nice explanation, thank you.
I have some questions.

1) weight = weight + learning_rate * error * input

Do I really need to multiply it with input ? For example here
http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.h
that…

2) Is your method same as in http://home.agh.edu.pl/~
i think yes, but again, Im not sure and Im confused by

3) What is exactly loss function in your example (I usua
other explanations), not transfer function derivation)? I
around …

4) momentum and weight decay. In your example, you
decay and add calculated momentum (to weight updat
weight update as w + deltaW, so again I'm mega confu

Sorry for dumb questions, … math is not my strong side, so many things which can be inferred by math sense are simply hidden for me.

John August 1, 2017 at 3:30 am #

REPLY

*substract both and weight update as w + deltaW, so again

I found above sentence as nonsense, must be side effect of my confusion …

Jason Brownlee August 1, 2017 at 8:12 am #

REPLY

Hang in there.

Pick one tutorial and focus on it. Jumping from place to place will make things worse for sure.

Jason Brownlee August 1, 2017 at 8:10 am #

REPLY

Hi John, good questions.

According to my textbook, yes.
I can't speak for random sites on the internet sorry.

Loss is prediction error. You can change this to other forms like MAE or MSE.

No decay or momentum in this example. Easy to add if you want. There are many ways to dial in the learning process. No hard and fast rules, just some norms that people reuse.

Parminder Kaur August 6, 2017 at 7:50 pm #

REPLY

A VERY GOOD TUTORIAL SIR…
Sir i am implementing remote sensed image classificat
I am not finding good resources on constructing featur
and number of neurons in hidden layer.
Any resources you know, can help me?

Thanks

Jason Brownlee August 7, 2017 at 8:41 am #

The CNN will perform feature extraction a
on the data to see if it helps the network.

The number of layers and neurons/filters per layer must be found using trial and error. It is common to copy the designs from other papers as a starting point.

I hope that helps.

pero August 9, 2017 at 1:11 am #

REPLY

Nice tutorial, very clean and readable code. =) thank you!

Jason Brownlee August 9, 2017 at 6:37 am #

REPLY

Thanks pero.

Vatandas August 15, 2017 at 3:28 am #

REPLY

1. I expect that this code is deep learning (many hidden layer) but not. One sentence is easy ("you can add more hidden layer as explained") but to do is not as easy as you said.

2. I think your code is wrong.
neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
but
Error = Target – ActivatedOutputNode
Delta = Error * Derivative(NONActivatedOutputNode)

I mean you use the same 'output' variable both error and delta. But in error it must be activated one, in delta it must be NONactivated one.

8CG_256 August 18, 2017 at 9:02 am #

REPLY

Nice tutorial, very clean code and beginner-fri

Jason Brownlee August 18, 2017 at 4:36 pm

Thanks, I'm glad you found it useful!

8CG_256 August 18, 2017 at 9:26 pm #

I only have one slight issue: I implemented this in Ruby and I tried to train it using the IRIS dataset, keeping the network simple (1 input layer, 1 hidden layer, 1 output layer) and after decreasing for a while the error rate keeps increasing. I tried lowering the learning rate, even making it dynamic so it decreases whenever the error increases but it doesn't seem to help. Could you give me some advice? P.S sorry for my bad English

Jason Brownlee August 19, 2017 at 6:19 am #

REPLY

Here is an example of backprop I developed in Ruby:
http://cleveralgorithms.com/nature-inspired/neural/backpropagation.html

Derek Martins August 22, 2017 at 9:22 pm #

REPLY

Hi Jason, I enjoy so much your tutorials. Can you do a tutorial implementing BackPropagation Through Time? Thanks man.

Jason Brownlee August 23, 2017 at 6:50 am #

REPLY

Thanks for the suggestion.

I have a few posts on the general topic, for example:
https://machinelearningmastery.com/gentle-introduction-backpropagation-time/

Anubhav Singh August 24, 2017 at 1:08 pm #

REPLY

Hello Jason,

Thank you for the great tutorial!

I would like to know how I can obtain the weight*input f

I've been trying these lines –

```
for layer in network:
new_inputs = []
for neuron in layer:
activation = activate(neuron['weights'], inputs)
neuron['output'] = transfer(activation)
new_inputs.append(neuron['output'])
```

but the activation variable here is a single value…what (number of hidden layers), I should get N*5 (N = numb activation…

Kindly help 🙂

Thank you!

Jose Panakkel August 25, 2017 at 10:45 am #

REPLY

Dear Jason,

I have a question on the delta calculation at the output layer, where the primary value is the difference between the neuron output and the expected output. And we are then multiplying this difference with the transfer_derivative. where transfer_derivative is a function of neuron's output.

My question is, is it correct to find the difference between the neuron's output and the expected output?

In this case of the example, you have chosen digital outputs [0,1] and hence it may not have come up .. but my point is…
one is already subjected to a transfer function, and one is not.

The neuron's output is always subjected to a transfer function and hence will be in a specific range, say -.5 to +.5 or something..
But the expected output is the user's choice .. isnt it?
user can have an expected value of say 488.34, for some stock price learning.. then is it still correct to find this primary difference between the expected output and the neuron output, at the output layer delta calculation?

shoulnt the expected output also be subjected to the same transfer function before finding the difference? Or the otherway, like shoulnt the neuron ouptut be subjected to a reverse transfer function before comparing with the expected output directly?

Thanks and Regards,
Jose Panakkel

RealUser404 September 6, 2017 at 1:36 pm #

Hello Jason, great tutorial that helped me a lo

I have a question concerning the back-propagation : w
desired gradient for the output (in the case of an actor-
How can I change your backprop function to make it w

Jason Brownlee September 7, 2017 at 12:49 pm #

REPLY

Sorry, I don't follow, perhaps you can restate your question with an example?

user28 September 8, 2017 at 9:26 pm #

REPLY

Hi Jason , thank you for providing this tutorial. I'm confused of how can I implement the same backpropagation algorithm with output not binary. Since I noticed that your example has binary output. Like predicting for stock price given the open, high, low and close values. Regards.

Jason Brownlee September 9, 2017 at 11:55 am #

REPLY

Use a library like Keras. Start here:
https://machinelearningmastery.com/start-here/#deeplearning

Lewis September 11, 2017 at 2:11 am #

REPLY

Hi Jason,

great article. I have an interest in NN but I am not that good at python.

Want I wanted to try was to withhold say 5 rows from the dataset and have the trained network predict the results for those rows. these is is different from what I think the example does which is rolling predictions with the learning. Removing 5 rows from the dataset is of course easy but my pitiful attempts at predicting with unseen data like below fail ((I guess network is not in scope at the end): any help appreciated!

```
# predict unseen data
unseendataset = [[12.37,13.47,0.8567,5.204,2.96,3.919,5.001],
[12.19,13.2,0.8783,5.137,2.981,3.631,4.87],
[11.23,12.88,0.8511,5.14,2.795,4.325,5.003],
[13.2,13.66,0.8883,5.236,3.232,8.315,5.056],
[11.84,13.21,0.8521,5.175,2.836,3.598,5.044],
[12.3,13.34,0.8684,5.243,2.974,5.637,5.063]]

for row in unseendataset:
prediction2 = predict(network, row)
print('Predicted=%d' % (prediction2))
```

Jason Brownlee September 11, 2017 at 12:0

I would recommend starting with Keras ra

Start here:
https://machinelearningmastery.com/start-here/#deeplearning

Karim September 14, 2017 at 1:27 pm #

REPLY

Hi Jason, I am trying to generalize your implementation to work with a variable number of layers and nodes. However, whenever I try to increase the number of nodes too much it stops working (the network freezes at one error rate and all output nodes are active, i.e. giving 1). Although the code would work if I decreased the layers and the errors will go down.
Is there something I am missing when using too many layers? The concepts should be the same.

I trained a network with 4 layers: [14,10,10,4] and it worked.
I trained a network with 4 layers [14,100,40,4] and it is stuck. Same dataset.

My code is here if you are looking in more details:
https://github.com/KariMagdy/Implementing-a-neural-network

Thanks

Jason Brownlee September 15, 2017 at 12:10 pm #

REPLY

What problem do you get exactly?

Laksh October 4, 2017 at 11:11 pm #

REPLY

Hi, Jason Brownlee,
can we extend this code for 2 or more hidden layers ?

Jason Brownlee October 5, 2017 at 5:24 am

Sure.

dsliver33 October 9, 2017 at 1:52 pm #

Dear Mr. Brownlee,

I'm trying to alter the code to represent a regression pr
layer). As far as I know, the main part of the code that
rewritten the code as below:

```
# Forward propagate input to a network output
def forward_propagate_regression(network, row):
    inputs = row
    new_inputs = []
    #gets the 1st layer, applies sigmoid activation
    hiddenlayer = network[0]
    for neuron in hiddenlayer:
        activation = activate(neuron['weights'], inputs)
        neuron['output'] = transfer(activation)
        new_inputs.append(neuron['output'])
    inputs = new_inputs
    #gets the last layer, applies linear activation
    outputlayer = network[-1]
    for neuron in outputlayer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = activation
            new_inputs.append(neuron['output'])
    inputs = new_inputs
    return inputs
```

With this code, I'm getting an "OverflowError: (34, 'Result too large')" error. Could you please tell what I'm doing wrong? All the other parts of the code are as you've written.

Jason Brownlee October 9, 2017 at 4:47 pm #

REPLY

What did you change exactly? Can you highlight the change for me?

Also, try using pre tags.

dsliver33 October 10, 2017 at 4:08 am #

REPLY

(I don't know how to highlight the change, sorry!)

I got the hidden layer (network[0]), and I applied your algorithm (calculate activation, transfer the activation to the output, append that to a new list called "new_inputs").

After that, I get the output layer (network[-1]), I calculate the activation with the "new_inputs", but I do NOT apply the sigmoid transfer function (sc
appended to a new list, which is set to be the r

Would that be the best way to remove the sign
regression, instead of a classification?

Jason Brownlee October 10, 2017

Sounds good. I don't have any gc
calculations to help spot where it is going v

You may want to consider moving to an open source neural net library, such as Keras:
https://machinelearningmastery.com/start-here/#deeplearning

Liam McGoldrick October 26, 2017 at 5:23 am #

REPLY

I am having the same issue with mine. i made alterations and they are just the same as yours. Did you find a solution?

Liam McGoldrick October 26, 2017 at 5:27 am #

REPLY

I GOT IT TO WORK!!! You have to normalize your output data. Then you can apply the transfer function to the output layer just the same! After that it will work!

Chris October 12, 2017 at 11:27 am #

REPLY

Hi Jason, nice posting and it really helps a lot
for j in range(len(layer)):
neuron = layer[j]
neuron[‘delta’] = errors[j] * transfer_derivative(neuron[‘output’])
Should the neuron[‘output’] be the output of the activation function instead of the transfer function here?

Asad October 14, 2017 at 3:24 pm #

REPLY

hi jason, nice post its really helps alot.

please tell me how we can change the neuron in hidden layer and in output layer?
and what will be the result when we change the neuron
in this tutorial u take one hidden layer,so can we use m

please tell me i m waiting

Jason Brownlee October 15, 2017 at 5:19 am

Perhaps you would be better served by st
https://machinelearningmastery.com/tutorial-first-n

dsliver33 October 16, 2017 at 2:27 pm #

Dear Mr. Brownlee,

I'm trying to adapt the code to support many hidden layers. I've adapted the code as below, with a new input called "n_layers", to insert N hidden layers in the network.

```
# Initialize a network with "n_layers" hidden layers
def initialize_network3(n_inputs, n_hidden, n_layers, n_outputs):
network = list()
for i in range(n_layers):
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)]} for i in range(n_outputs)]
network.append(output_layer)
return network
```

When I try to run the code, it shows the error below. Do you have any idea why?

```
in backward_propagate_error(network, expected)
78 error = 0.0
79 for neuron in network[i + 1]:
—> 80 error += (neuron['weights'][j] * neuron['delta'])
81 errors.append(error)
82 else:

IndexError: list index out of range
```

dna_remaps February 3, 2018 at 10:43 pm #

REPLY

This took me a minute to figure out myself.

You need to add a conditional after your first layer to make sure your subsequent hidden layer weights have the proper dimensions (n_hidden+1, n_hidden)

```
for i in range(n_layers):
hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
if i > 0:
hidden_layer = [[{'weights':[random() for i in range(
network.append(hidden_layer)
```

Arijit Mukherjee October 17, 2017 at 1:40 am #

Hi,

In the output/last layer when we are calculating the ba
derivative with the (expected-output)?? transfer derivat
update should be only (expected-output)*previous_lay
Thanks

Tanoh Henry October 18, 2017 at 8:54 pm #

REPLY

Really good article. Thanks a lot.
Need a little bit of clarification.
For backward propagation starting at the output layer,
you get the error by appending to errors expected[j] – neuron['output'].
Isn't Error = 0.5 * sum(errors)?
and then using this sum of errors for back-propagation?
Thanks.

Liam October 21, 2017 at 5:41 am #

REPLY

Thanks for the tutorial! I am trying to modify your code to do a regression model and I am stuck. I have an input data set (4 columns and many rows) and a single variable output data set (in range of tens of thousands). I fed them into the train procedure and I get an error when it reaches "expected = [0 for i in range(n_outputs)]" in the train portion. The error reads "only length-1 arrays can be converted to Python scalar". Now I understand this is because of the intended purpose for the code was a categorization problem but I am wondering what I would need to modify to get this to work? Any help would go a long way as I have been stuck on this issue for some time now.

Thanks, and again wonderful tutorial!

Jason Brownlee October 21, 2017 at 5:45 am #

REPLY

Perhaps start with Keras, it will be much easier for you:
https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/

Sam October 26, 2017 at 11:15 pm #

Hi
I am implementing a 2 layer neural network with 100 h
in the next using your code. Implement sigmoid activat
model on the MNIST dataset subset.
But it is always giving same prediction.
[0.99999999986772, 0.99999999994584]
Expected=0, Got=1
[0.99999999986772, 0.99999999994584]
Expected=1, Got=1
[0.99999999986772, 0.99999999994584]

```
Expected=1, Got=1
[0.99999999986772, 0.99999999994584]
Expected=1, Got=1
[0.99999999986772, 0.99999999994584]
Expected=1, Got=1
[0.99999999986772, 0.99999999994584]
Expected=0, Got=1
[0.99999999986772, 0.99999999994584]
Expected=0, Got=1
[0.99999999986772, 0.99999999994584]
Expected=1, Got=1
[0.99999999986772, 0.99999999994584]
Expected=0, Got=1
[0.99999999986772, 0.99999999994584]
Expected=0, Got=1
[0.99999999986772, 0.99999999994584]
Expected=0, Got=1
[0.99999999986772, 0.99999999994584]
Expected=0, Got=1
```

Jason Brownlee October 27, 2017 at 5:20 am #

REPLY

I would recommend using a framework like Keras:
https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

Matthias April 15, 2018 at 2:57 am #

REPLY

Weights should be initialized with normally distributed random values. Try using random.gauss for weight initialization.

John October 28, 2017 at 6:52 pm #

help, I dont know why i got this error.

Traceback (most recent call last):
File “a.py”, line 185, in
for i in range(len(dataset[0])-1):
TypeError: ‘NoneType’ object has no attribute ‘__getite

Jason Brownlee October 29, 2017 at 5:52 am #

REPLY

You cannot have the “-1” within the call to len()

João Costa November 12, 2017 at 12:46 pm #

REPLY

Hey Jason, thanks for your post!

This is helping me a lot with a college work. But in this NN, how can I set manually not the number of input neuros, the input values?

For example, if I have 1 input neuro, I wan’t to set this value to 0.485.

Best regards!

Jason Brownlee November 13, 2017 at 10:11 am #

REPLY

Sorry, I don’t follow.

Perhaps you’d be better off using a library like Keras:
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

yesta November 17, 2017 at 8:21 am #

REPLY

Hi, Jason
Thank you for this amazing tutorial!

I have a question that may be out of the topic. How do you call models or type of DL models where you feed a model with new test data in order to make the model adaptive to the environment?

Thank you.

Jason Brownlee November 17, 2017 at 9:31

Yes, you can update a model after it has b

Nil December 6, 2017 at 7:47 pm #

Hi, Dr. Jason,

I have been studying how to develop a neural network from scratch and this tutorial is the main one I have been following because it is helping me so much.
I have a doubt: When I study the theory I see the neural network scheme carrying only the weights and bias. And here in practice I see that the network is also carrying the output values and the delta i.e (weights, bias, output and delta). Will the final model be saved like this? with the latter (weights, bias, output and delta)? would this be the rule in practice?

I would appreciate it if you could help with this issue so that I could get out of where I left off.

Your posts are really very good there is where I find my way in to learning in Machine Learning.

Best Regards

Jason Brownlee December 7, 2017 at 7:51 am #

REPLY

The final model (e.g. trained) only needs to perform the forward pass.

Nil December 8, 2017 at 5:50 am #

REPLY

Understood.
Thank you Dr. Jason

MohamedElshazly December 8, 2017 at 9:30 pm #

REPLY

Hi , there's something i don't understand :

```
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
```

wouldn't the last line be out of range because the curre
thanks in advance

Jason Brownlee December 9, 2017 at 5:41 a

No, because of the "if" check on the 4th li

Olu December 9, 2017 at 3:17 am #

Hi Mr Brownlee,

Thank you for your tutorial. The training for the example worked however when I try to implement the code for the Wheat Seeds Dataset I get an error from my line 210:

for i in range(len(dataset[0]) – 1):
str_column_to_float(dataset, i)

The error is: IndexError: list index out of range

Can you please explain why it is (dataset[0])? Does (dataset[0]) means the 1st column in the dataset?

Jason Brownlee December 9, 2017 at 5:45 am #

REPLY

Yes, I recommend learning more about Python arrays here:
https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/

The example was written for Python 2.7, confirm your Python version.

Jonesy December 12, 2017 at 3:17 pm #

REPLY

Hello Jason,

Fantastic stuff here. I had a question about the network configuration. This 2 input, 2 hidden and 2 output seems a bit odd to me. I'm used to seeing 2, 2, 1 for XOR – can you explain why you have two output nodes and how they work with the max function? I think it would better explain this line for me in train():

expected[row[-1]] = 1

And lastly, why would one choose this configuration over a 2, 2, 1.

Thanks!

Jason Brownlee December 12, 2017 at 4:12

The final model has the shape [7, 5, 3]. P
for each of the 3 classes in the dataset.

Configuration was chosen via trial and error. There
neural network.

Finally, you can learn more about array indexing in
https://machinelearningmastery.com/index-slice-re

Mohamed Elshazly December 17, 2017 at 9:34 p

Hi Jason

In the tain_network function the line “expected[row[-1]] = 1” what i understand is that you take the Y value of every row (which is either 0 or 1) and use it as an index in the expected array and you change the value at that index to 1 ,First i don’t know if i understand that correctly in the first place or not but if so, Wouldn’t the modification to the expected array be locked down to just only the first and second index because “expected[row[-1]] = 1” would only be expected[0] or expected[1] ? and how would that help in our algorithm .

looking forward to your response and thanks for the Great Tutorial

Jason Brownlee December 18, 2017 at 5:22 am #

REPLY

We are one hot encoding the variable.

Learn more about it here:
https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

MohamedElshazly December 18, 2017 at 4:55 pm #

REPLY

HI again Jason

If I’m implementing this algorithm in python 3 what should i change in expected[row[-1]]=1 in order for it to work because I’m having this error : list assignment index out of range
thanks in advance

Jason Brownlee December 19, 2017 at 5:15 am #

REPLY

I don’t know off the cuff, I will look into porting the example to Py3 in the new year.

Tushar December 19, 2017 at 5:29 pm #

You are just awesome Jason. You are adding
graduate schools do in the US.

Thanks a ton!

Jason Brownlee December 20, 2017 at 5:39

Thanks Tushar.

mark January 6, 2018 at 3:29 am #

REPLY

Wow, thanks for your codes. I have a question, what if I want to add regularisation term like L2 during back propagation, what should i do?

Jason Brownlee January 6, 2018 at 5:55 am #

REPLY

I would recommend moving to a platform like Keras:
https://machinelearningmastery.com/start-here/#deeplearning

mark January 7, 2018 at 5:12 pm #

REPLY

Thanks for replying. I know the keras and have been using keras for a while. But in the problem I am focusing on, I need to make changes on the back propagation. That's why I didn't use keras.
So let's go back to my original question, is the error term the cost function? Thanks.

Jason Brownlee January 8, 2018 at 5:42 am #

REPLY

Sorry, I cannot work-through adding regularization to this tutorial for you.

Mojo January 20, 2018 at 7:04 pm #

REPLY

Hello Jeson,
Thanks for the informative tutorial. I have a question.
if i want to change the error equation and as well as th
output layer. How can i change it?
Hope you will reply in a short time.

Regards,
Mojo

Jason Brownlee January 21, 2018 at 9:08 am

I would recommend using a library instead

Here's how to get started with Keras:
https://machinelearningmastery.com/start-here/#deeplearning

Aliya Anil February 16, 2018 at 9:04 pm #

REPLY

Hi Jason,

It was indeed a very informative tutorial. Could you please explain the need for seed(1) in the code?

Jason Brownlee February 17, 2018 at 8:44 am #

REPLY

I am trying to tie down the random number generator so that you get the same results as me.

Learn more here:
https://machinelearningmastery.com/randomness-in-machine-learning/

Raj February 19, 2018 at 7:10 am #

REPLY

Hey there,
Been following your tutorial and I'm having problems with using my dataset with it. The outputs of the hidden neurons appear to only be exactly 1 constantly. I'm not sure what's wrong exactly or how to fix it but its resulting in the network not learning at all. Please let me know if you can help.
Thanks,
Raj

Jason Brownlee February 19, 2018 at 9:10 am #

REPLY

Perhaps try to get the code and data in th
for your own problem.

Generally, I would recommend using a library like
from scratch as a learning exercise.

Aliya Anil February 20, 2018 at 4:37 pm #

Hi,

I tried the first code in the tutorial with 4-parameter dat
Could you explain the reason?

Thanks,
Aliya

Jason Brownlee February 21, 2018 at 6:36 am #

REPLY

If you are looking to develop a neural network for your own data, I would recommend the Keras library:
https://machinelearningmastery.com/start-here/#deeplearning

Nik March 2, 2018 at 1:54 pm #

REPLY

Dear Jason,

Can I use the codes for handwritten digits recognition? If yes, are there any special recommendations what to change in the codes or I can use them with no changes?

Thanks,
Nik

Jason Brownlee March 2, 2018 at 3:25 pm #

REPLY

I would recommend this tutorial:
https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

Nik March 2, 2018 at 7:51 pm #

REPLY

Yes, I have seen that tutorial. But is th
would like to understand why they work so wel

Jason Brownlee March 3, 2018 at

Yes, you can. You can develop a

I cannot write the modification for you, sor

Filoingko March 4, 2018 at 2:58 am #

Hi,

How can I use this trained network to predict another data set.

Thank you.

Jason Brownlee March 4, 2018 at 6:04 am #

REPLY

The code in this tutorial is to teach you about backprop, not for use on real problems. If you are working through a problem, I'd recommend using Keras.

Jean-Michel Richer March 5, 2018 at 9:27 pm #

REPLY

Dear Jason,
I have tried to use your code on a simple XOR example but get a result of [0, 0, 1, 1] instead of [0,1,1,0]
Scores: [0.0]
Mean Accuracy: 0.000%

The input xor.csv file is
0,0,0
0,1,1
1,0,1
1,1,0

For this I have modified the evaluate_algorithm function to:

```
def evaluate_algorithm_no_fold(dataset, algorithm, *args):
scores = list()
predicted = algorithm(dataset, dataset, *args)
print(predicted)
accuracy = accuracy_metric(dataset, predicted)
scores.append(accuracy)
return scores
```

and call the function like this:

```
scores = evaluate_algorithm_no_fold(dataset, back_pr
```

Would you have some explanation because I can not f
Best regards,
JM

Jason Brownlee March 6, 2018 at 6:12 am #

Perhaps the model requires tuning to you

Tanveer March 5, 2018 at 9:29 pm #

REPLY

Thank You So Much Jason !! Wonderful Tutorial. THANKS Much !!

Jason Brownlee March 6, 2018 at 6:12 am #

REPLY

You're welcome.

Mojo March 9, 2018 at 10:06 pm #

REPLY

If i want to calculate the training accuracy and F-measure and want to change the activation function, how i can do it?

Jason Brownlee March 10, 2018 at 6:28 am #

REPLY

Perhaps you would be better off using scikit-learn and Keras instead.

Fahad March 12, 2018 at 8:03 pm #

REPLY

Is there something wrong with this code in case of using MINIST data? I tried to change the structured of the data to be compatible with the code, but it gave me a huge error and the error did not decrees during all training steps

Jason Brownlee March 13, 2018 at 6:25 am #

REPLY

The code was not developed for MNIST. H
http://machinelearningmastery.com/handwritten-dig
python-keras/

Fahad March 13, 2018 at 4:06 pm #

Thanks Jason for your response. I want to ap
structure of the data to be each row as a vector of 784
huge error and does not decrees at all.

I am trying to develop some algorithm for enhancing of learning, hence, I need to deal with the procedure as step by step. So keras or any other library does not help.

Thanks again Jason

Jason Brownlee March 14, 2018 at 6:17 am #

REPLY

Perhaps update the code to use numpy, it will be much faster.

kelvin March 15, 2018 at 2:39 am #

REPLY

Hi Mr Brownlee,

Can you teach me how to plot the errors per epochs (validation error) and accuracy for both training and validation in your scratch network?

kelvin March 15, 2018 at 2:44 am #

REPLY

I only can find the training error but not validation error in the code. For the accuracy, I plot a graph have a straight line only.

Jason Brownlee March 15, 2018 at 6:33 am #

REPLY

Yes, you can do it easily in Keras here:
https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

kelvin March 15, 2018 at 12:19 pm #

Is there any possible way to do it on y
code save the training error, validation error, tr
plot the graph myself since your scratch mode

Jason Brownlee March 15, 2018 at

Yes, perhaps change it from CV t
each dataset at the end of each epoch. Sa

Jack March 15, 2018 at 12:11 pm #

REPLY

Can I use this model for regression problem? For example us this model for boston house-prices dataset?

Jason Brownlee March 15, 2018 at 2:50 pm #

REPLY

Sure, some changes would be required, such as the activation in the output layer would need to be linear.

Nabil March 15, 2018 at 3:49 pm #

REPLY

Are you using MSE?

Jason Brownlee March 16, 2018 at 6:09 am #

REPLY

As mentioned in the post, we are reporting accuracy for the classification problem.

Olu March 19, 2018 at 11:23 pm #

REPLY

In the train section,

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            backward_propagate_error(networ
            update_weights(network, row, l_
```

Can you please explain how this expected[row[-1]] = 1 created.

Jason Brownlee March 20, 2018 at 6:23 am #

Good question.

expected is all zeros. row[-1] is the index of the cla in expected to 1.

Perhaps it is worth reading up on array indexing:
https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/

kmillen November 16, 2018 at 11:27 am #

REPLY

Jason,
This is an amazing piece of code that has been very beneficial.

Doesn't that mean that only expected[0] and expected[1] will ever be set to 1 for this test data?

Thank you,

Jason Brownlee November 16, 2018 at 1:58 pm #

REPLY

Sorry, I don't understand your question?

kmillen November 17, 2018 at 1:44 am #

If I understand Python (which I may not), row[-1] represents the last item in the row. Since the last value in each of the 10 rows is only either 0 or 1, expected[row[-1]] = 1 will only ever set expected[0] or expected[1] to the value of 1. Or, what am I missing?

Jason Brownlee November 17, 2018 at 5:50 am #

Are you referring to this: expected[row[-1]] = 1

If so:

"expected" is all zeros, e.g. [0, 0]
"row" is an example, e.g. […] where th

Therefore row[-1] is an index of either
that index as 1.

We have created a one hot vector.

kmillen November 17, 2018 at 2:54 am

Disregard my previous question;
again for this example.

kelvin March 21, 2018 at 2:14 am #

REPLY

Hi, I would like to use softmax as the activation function for output layer. However, I do not know how to write the code for the derivative of softmax. Can you show me the code how to change the sigmoid function from your code to softmax?

kelvin March 21, 2018 at 2:20 am #

REPLY

I do try few ways to change the sigmoid to softmax, however, all of them are not working. Can you show me how to create a softmax layer?

for transfer():
first case:

```
def transfer(input_value):
exp_scores = np.exp(input_value)
return exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
```

second case:

```
def transfer(input_value):
input_value -= np.max(input_value)
return np.exp(input_value) / np.sum(np.exp(input_value))
```

third case:

```
def transfer(input_value):
input_value -= np.max(input_value)
result = (np.exp(input_value).T / np.sum(np.exp(input_value))).T
return result
```

for transfer_derivative():
first case:

```
def transfer_derivative(output):
s = output.reshape(-1, 1)
return np.diagflat(s) – np.dot(s, s.T)
```

second case:

```
def transfer_derivative(output):
jacobian_m = np.diag(output)
for i in range(len(jacobian_m)):
for j in range(len(jacobian_m)):
if i == j:
jacobian_m[i][j] = output[i] * (1 – output[i])
else:
jacobian_m[i][j] = -output[i] * output[j]
return jacobian_m
```

Jason Brownlee March 21, 2018 at 6:39 am #

REPLY

Here you go:
https://en.wikipedia.org/wiki/Softmax_function#Artificial_neural_networks

Jason Brownlee March 21, 2018 at 6:38 am #

REPLY

Perhaps use Keras instead?

Suede March 29, 2018 at 6:40 am #

REPLY

hey Jason, this is very helpful. I have run the code but i keep on getting this error, can you please help me out? the error is:

NameError Traceback (most recent call last)
in ()
186 str_column_to_float(dataset, i)
187 # convert class column to integers
–> 188 str_columnto_int(dataset, len(dataset[0])-1)
189 # normalize input variables
190 minmax = dataset_minmax(dataset)

NameError: name ‘str_columnto_int’ is not defined

Jason Brownlee March 29, 2018 at 6:42 am #

REPLY

The code was written for Python 2.7, confirm that you are using this version of Python?

Fahri Güreşçi April 15, 2018 at 7:03 am #

The csv file is not working. Edited csv file > bi
you can use python 2 or 3
results:
python2 > Mean Accuracy: 95.238%
python3 > Mean Accuracy: 93.333%

Why different?

Jason Brownlee April 16, 2018 at 6:00 am #

REPLY

Wh is the CSV file not working? The link appears to work fine.

Good question re the difference, no idea. Perhaps small differences in the API? The code was written for Py2, so it may require changes for py3.

Also, see this post on the stochastic nature of ml algorithms:
https://machinelearningmastery.com/randomness-in-machine-learning/

Fahad April 18, 2018 at 8:58 pm #

REPLY

I have altered the code to work with MNIST (digit numbers) , the problem I have faced that forward_propagate function returns [1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1] for each instance !

Any help

Jason Brownlee April 19, 2018 at 6:30 am #

REPLY

Well done.

The model will require tuning for the problem.

Fahad April 19, 2018 at 7:09 am #

REPLY

Could you explain more in some details please.

Jason Brownlee April 19, 2018 at 2:45 pm #

What details exactly?

Fahad April 19, 2018 at 8:17 pm #

As I mentioned that forward_propagation func
alter to come over this problem

Jason Brownlee April 20, 2018 at 5:48 am #

Perhaps tune the model to your specific problem. I have some suggestions here:
http://machinelearningmastery.com/improve-deep-learning-performance/

Fahad April 23, 2018 at 5:34 am #

REPLY

I altered the code to work with XOR problem and it was working perfectly. Then, I altered the code to work with digit numbers MNIST, but as I told you there was a problem with the the forward_propagation function that it returned all outputs to be [1,1,1,…] instead of a probabilities for each output.
I think it is not an optimization problem, there is something wrong with the forward_propagate function.

Here is the code after alteration [it is working but with a fixed error during all training epochs

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
global network
global gl_errors

# Load a CSV file
def load_csv(filename):
dataset = list()
with open(filename, 'r') as file:
csv_reader = reader(file)
for row in csv_reader:
if not row:
continue
dataset.append(row)
return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
for row in dataset:
row[column] = float(row[column].strip())

def str_column_to_intX(dataset, column):
for row in dataset:
row[column] = int(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
class_values = [row[column] for row in dataset]
unique = set(class_values)
lookup = dict()
for i, value in enumerate(unique):
lookup[value] = i
for row in dataset:
```

```
row[column] = lookup[row[column]]
return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
minmax = list()
stats = [[min(column), max(column)] for column in zip(*dataset)]
return stats

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset):
for row in dataset:
for i in range(1,len(row)):
# row[i] = (row[i] – minmax[i][0]) / (minmax[i][1] – minmax[i][0])
if row[i]>10:
row[i]=1
else:
row[i]=0

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
dataset_split = list()
dataset_copy = list(dataset)
fold_size = int(len(dataset) / n_folds)
for i in range(n_folds):
fold = list()
while len(fold) epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))

# 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[1:]
i=0

for layer in network:
new_inputs = []
i+=1
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):
# err =0
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']

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
global network
network = list()
hidden_layer1 = [{'weights':[random() for i in range(n_i
network.append(hidden_layer1)
```

```
hidden_layer2 = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(100)]
network.append(hidden_layer2)

hidden_layer3 = [{'weights':[random() for i in range(100 + 1)]} for i in range(50)]
network.append(hidden_layer3)

output_layer = [{'weights':[random() for i in range(50 + 1)]} for i in range(n_outputs)]
network.append(output_layer)
return network

# Make a prediction with a network
def predict(network, row):
outputs = forward_propagate(network, row)
return outputs.index(max(outputs))

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'dataset/train2.csv'
dataset = load_csv(filename)

for i in range(1,len(dataset[0])):
str_column_to_float(dataset, i)

# convert class column to integers
str_column_to_int(dataset, 0)

normalize_dataset(dataset)

# evaluate algorithm
n_folds = 5
l_rate = 0.5
n_epoch = 100
n_hidden = 500

scores = evaluate_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(l
```

Jason Brownlee April 23, 2018 at 6:26 am #

Sorry, I don't have the capacity to debug t

Rahmad ars April 26, 2018 at 3:23 am #

Sir, can you help me?
this is my question..

https://stackoverflow.com/questions/50027886/need-help-for-check-my-backprop-ann-using-python

Jason Brownlee April 26, 2018 at 6:37 am #

REPLY

Perhaps you can summarize your question in a sentence or two?

Rahmad ars April 26, 2018 at 7:19 am #

REPLY

your original code sir, only have 1 hidden layer with 2 neurons. then, I modify it, so the ANN have 3 hidden layers, each consist of (128, 64, 32). and i have my own dataset, so i change it (the dataset and input neurons). when i run this code, everything looks fine but the error value is not changing…

here's the screen: https://i.stack.imgur.com/NQbNd.png

modified code: https://stackoverflow.com/questions/50027886/need-help-for-check-my-backprop-ann-using-python

thanks sir

Jason Brownlee April 26, 2018 at 2:59 pm #

REPLY

If you are struggling with the code, I would recommend not coding the algorithm from scratch.

Instead, I would recommend using a library like Keras. Here is a worked example: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

Fahad April 28, 2018 at 9:20 pm #

REPLY

I have the same problem of Rahmad

The same problem occurs when you change the origin
neuron (the error value does not change).

I know 31 hidden neuron is not a right number of neuro
the wrong when you increase the number of neurons.

Logically, it should be fine and the error value decreas
it is still working , when change it to 31 neurons it does

I think if this problem is fixed, then the problem of Rah

Jason Brownlee April 29, 2018 at 6:26 am #

Perhaps the model requires tuning to your specific problem (e.g. layers, nodes, activation function, etc.)

It might be better to use a library like Keras for your project:
https://machinelearningmastery.com/start-here/#deeplearning

Rocha May 2, 2018 at 9:58 pm #

REPLY

Hi dude, I'm stuck in this error, could you help me?

```
# 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
```

That line is giving me this: TypeError: list indices must be integers or slices, not str

Should be the python version? I'm using python 3…

Jason Brownlee May 3, 2018 at 6:33 am #

REPLY

This code was written for Python 2.7 sorry.

Kamrun Nahar Nisha May 8, 2018 at 4:00 pm #

hello.please help me.
I want to use breast cancer dataset instead of seed da

```
seed(1)
# load and prepare data
filename = 'seeds_dataset.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# normalize input variables
```

```
minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
# evaluate algorithm
n_folds = 5
l_rate = 0.3
n_epoch = 500
n_hidden = 5
scores = evaluate_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

In this part of your code I also want to print the error and it will be like

```
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
[{'output': 0.029980305604426185, 'weights': [-1.4688375095432327, 1.850887325439514,
1.0858178629550297], 'delta': -0.0059546604162323
[0.37711098142462157, -0.0625909894552989, 0.276
[{'output': 0.23648794202357587, 'weights': [2.515394
-0.9671565426390275], 'delta': -0.0427005927836458
[-2.5584149848484263, 1.0036422106209202, 0.4238
```

please tell me the code . Using breast cancer dataset
that's why I need your help immediately.

Jason Brownlee May 9, 2018 at 6:09 am #

I'm eager to help, but I do not have the capacity to outline the changes or write the code for you.

Akefar May 11, 2018 at 4:41 am #

REPLY

hi Jason,
I tried your code in my data set ,shape of my data is (576,16) .the problem is

IndexError: list assignment index out of range

is there any need to change your code for (576,16) data shape .
Thanks

IndexError Traceback (most recent call last)
in ()
195 n_epoch = 500
196 n_hidden = 1
–> 197 scores = evaluate_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
198 print('Scores: %s' % scores)
199 print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

in evaluate_algorithm(dataset, algorithm, n_folds, *args)
79 test_set.append(row_copy)
80 row_copy[-1] = None
—> 81 predicted = algorithm(train_set, test_set, *args)
82 actual = [row[-1] for row in fold]
83 accuracy = accuracy_metric(actual, predicted)

in back_propagation(train, test, l_rate, n_epoch, n_hidden)
171 n_outputs = len(set([row[-1] for row in train]))
172 network = initialize_network(n_inputs, n_hidden, n_outputs)
–> 173 train_network(network, train, l_rate, n_epoch, n_outputs)
174 predictions = list()
175 for row in test:

in train_network(network, train, l_rate, n_epoch, n_out
148 outputs = forward_propagate(network, row)
149 expected = [0 for i in range(n_outputs)]
–> 150 expected[row[-1]] = 1
151 backward_propagate_error(network, expected)
152 update_weights(network, row, l_rate)

IndexError: list assignment index out of range

Jason Brownlee May 11, 2018 at 6:39 am #

You may need to change your data to match the model or the model to match the data.

Pradeep May 14, 2018 at 3:31 am #

REPLY

Hi Jason, I tried your code on the same sample dataset, i am getting the following type error in the function activate. I am doing it in python3.6. hope to hear from you soon

Traceback (most recent call last):
File “neural_network.py”, line 94, in
train_network(network, dataset, 0.5, 20, n_outputs)
File “neural_network.py”, line 76, in train_network
outputs = forward_propagate(network, row)
File “neural_network.py”, line 31, in forward_propagate
activation = activate(neuron[‘weights’], inputs)
File “neural_network.py”, line 18, in activate
activation += weights[i] * inputs[i]
TypeError: can’t multiply sequence by non-int of type ‘float’

Jason Brownlee May 14, 2018 at 6:39 am #

REPLY

The code requires Python 2.7.

Deepak D May 17, 2018 at 6:49 pm #

REPLY

Hi Jason Brownlee,

I tried your code and experienced the some error applying the Backpropagation algorithm to the wheat seeds dataset. I am using python 2.7.

Error type:

File “C:\Python27\programs\backpropagation.py”, line
in str_column_to_float(dataset, i)

File “C:\Python27\programs\backpropagation.py”, line
in str_column_to_float row[column] = float(row[column]
ValueError: could not convert string to float:

Jason Brownlee May 18, 2018 at 6:22 am #

I am sorry to hear that, I have some sugg
https://machinelearningmastery.com/faq/single-faq

Dhanya Hegde May 19, 2018 at 1:53 am #

REPLY

Hey Jason! Great work. Really helpful. I didn't understand one part of your code. On what basis does predict function return the predicted value as 0 or 1, after taking the maximum of the two output neuron values?

Jason Brownlee May 19, 2018 at 7:44 am #

REPLY

The summation of the activation is passed through a sigmoid transfer function.

Dhanya Hegde May 21, 2018 at 3:37 am #

REPLY

I didn't understand this part of the code

outputs.index(max(outputs)

Is one hot encoding used or binary classification?
If so, how is the actual mapping done?
And when is the iteration process stopped?

Jason Brownlee May 21, 2018 at 6:35 am #

REPLY

As stated in the text above the code, it returns an integer for the class with the largest probability.

Ionut May 27, 2018 at 12:05 am #

Hi,

I'm a beginner in neural networks and I don't understar
Can anyone explain me what x1, x2 and y means?

Jason Brownlee May 27, 2018 at 6:45 am #

Input 1, input 2 and output.

Perhaps start here:
https://machinelearningmastery.com/start-here/#al

Rishik Mani May 28, 2018 at 7:26 am #

REPLY

Hi Jason, thank you for the highly informative post. But could you please clarify me upon this petty little issue.

In section 4.2 Train network, you considered n_inputs = len(dataset[0]) – 1. Why did you put a -1 here, while the number of the inputs should exactly be of the length of the dataset.

Jason Brownlee May 28, 2018 at 2:32 pm #

REPLY

To exclude the output variable from the number of inputs.

Samih Eisa June 2, 2018 at 9:13 pm #

REPLY

Thank you, jason.

Jason Brownlee June 3, 2018 at 6:22 am #

REPLY

You're welcome.

Kie Woo Nam June 5, 2018 at 3:01 am #

REPLY

```
# 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
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_
            neuron['weights'][-1] += l_rate
```

Hi,

I guess I'm likely mistaken, so please but when i != 0, i
second time?

So, shouldn't it be "inputs = [neuron['output'] for neuron

If I'm wrong, I'll read the code again more carefully, so

Jason Brownlee June 5, 2018 at 6:46 am #

REPLY

No. There are more weights than inputs and the -1 index of the weights is the bias.

Kie Woo Nam June 5, 2018 at 7:06 pm #

REPLY

Ah, right. Now I see it from “output_layer = [{‘weights’:[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]”.

Thank you for your quick reply.

Jason Brownlee June 6, 2018 at 6:39 am #

REPLY

No problem.

Thomas Specht July 10, 2018 at 4:46 am #

REPLY

Hi Jason,

Great tutorial to get into ML coding. I have one question:

What library would you recommend for projects and why? I want to use NN for regression problems.

Jason Brownlee July 10, 2018 at 6:53 am #

REPLY

I recommend Keras because it is computationally efficient, fast for development and fun:
https://machinelearningmastery.com/start-here/#de

Hugo B. August 13, 2018 at 1:29 am #

Hi Jason! Thank you for this tutorial.
I try to implement the batch learning, but I have some

– Computing the errors:
Do I have to accumulate the errors (‘delta’) in backwar
an average according to the number of back-propagati

– Updating the weights:
In train_network(), I call update_weights() for each epoch, but I don't know which row(s) of train (dataset) I have to used. Currently I use only one row: train[0].

elizabeth August 14, 2018 at 2:40 am #

REPLY

Hello Sir
Thank you so much for this article.
Can you please tell me how i can solve this error:

File "mlp8.py", line 186, in
str_column_to_float(dataset, i)
File "mlp8.py", line 22, in str_column_to_float
row[column] = float(row[column].strip())
ValueError: could not convert string to float:

thankyou

Jason Brownlee August 14, 2018 at 6:22 am #

REPLY

I have some suggestions here:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

STEPHEN OLADEJI August 28, 2018 at 10:35 pm #

REPLY

Dear Prof,

Thanks for this tutorial,
Sir, I've gone through a lot of your project online and they are very superb. God bless you, sir.
Sir. my question is as followings.
1. I noticed that one WEKA 3.6, Artificial Immune Syste
because there is no research prospect in the algorithm
2. I want to write python version for AIRS, CSCA, Gen
what I write is correct

Jason Brownlee August 29, 2018 at 8:11 am

I have an implementation here that you ca
http://wekaclassalgos.sourceforge.net/

Tamara September 11, 2018 at 5:36 am #

REPLY

Thank you so much for this tutorial.
How to see the results of work a trained network in Python?

Jason Brownlee September 11, 2018 at 6:34 am #

REPLY

What do you mean exactly?

ritu September 23, 2018 at 9:49 am #

REPLY

How should I modify to code to always run for one output neuron in the output layer?
eg. if the output class consist of only 2 output classes ‘1’ and ‘2’ , as per the above code 2 neurons will be created within the output layer, but what if I wanted a neural network to just have one neuron in the output layer.

Jason Brownlee September 24, 2018 at 6:09 am #

REPLY

If you are having trouble with this tutorial, I would encourage you to use Keras to develop neural network models.

You can get started here:
https://machinelearningmastery.com/start-here/#python

Parva September 27, 2018 at 9:04 am #

REPLY

Why is there just one output coming from laye
outputs one from each neuron?

[{‘output’: 0.7105668883115941, ‘weights’: [0.1343642
0.763774618976614], ‘delta’: -0.00053480480466105
[{‘output’: 0.6213859615555266, ‘weights’: [0.2550690
-0.14619064683582808}, {‘output’: 0.65736934559869
0.651592972722763], ‘delta’: 0.0771723774346327}]

Jason Brownlee September 27, 2018 at 2:47

Which step exactly are you having trouble

Brian September 28, 2018 at 1:17 am #

REPLY

Thank you for this example. It has helped me get past the block I had with the mathematical based descriptions and differential calculus related to back propagation.
As I have been focusing on the back propagation portion of this example I have come up with an alternative version of the 'backward_propagate_error' function that I think is a much more succinct and logical way to write this function.

Please find below

```
# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        for j in range(len(layer)):
            fromNeuron = layer[j]
            error = 0.0
            if i != len(network)-1:                          #This identifies all
                for toNeuron in network[i + 1]:
                    error += (toNeuron['weights'][j] * toNeuron['delta'])
            else:                                            #This is the last (o
                error = expected[j] - fromNeuron['output']
            fromNeuron['error'] = error
            fromNeuron['delta'] = error * transfer_derivative(fromNeuron['output']
```

Jason Brownlee September 28, 2018 at 6:16 am #

REPLY

Cool!

Sorry, I don't have the capacity to review your code.

Brian October 4, 2018 at 2:59 am #

REPLY

Should not the formula for the error be
sum_error += sum([0.5*(expected[i]-outputs[i])**2 for i
as apposed to
sum_error += sum([(expected[i]-outputs[i])**2 for i in ra
To correctly back propagate with the derivative?

Jason Brownlee October 4, 2018 at 6:20 am

Why is that Brian?

KS October 10, 2018 at 2:15 am #

REPLY

Can you please tell me how to inplant tanh and ReLu?
I tried to find examples on the internet, but I couldn't find good examples.

I understand that these 2 codes need to be changed:

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

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

1. What is the code when using tanh?
2. What is the code when using ReLu?

Jason Brownlee October 10, 2018 at 6:15 am #

REPLY

Thanks for the suggestion, sorry, I don't have the capacity to make these changes for you.

moe October 10, 2018 at 7:08 pm #

REPLY

Thank you so much, 1 – 5 helped me to built my own generic network. Now i have a MLP network which i can easily adjust with a few parameter changes, wouldnt have been so easy with your example.

Jason Brownlee October 11, 2018 at 7:51 am #

REPLY

Well done!

Anwar October 11, 2018 at 3:35 am #

I am running the same code that u have provi
help me:-

IndexError Traceback (most recent call last)
in ()
16 n_epoch = 500
17 n_hidden = 5

```
—> 18 scores = evaluate_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
19 print('Scores: %s' % scores)
20 print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

in evaluate_algorithm(dataset, algorithm, n_folds, *args)
12 test_set.append(row_copy)
13 row_copy[-1] = None
—> 14 predicted = algorithm(train_set, test_set, *args)
15 actual = [row[-1] for row in fold]
16 accuracy = accuracy_metric(actual, predicted)

in back_propagation(train, test, l_rate, n_epoch, n_hidden)
4 n_outputs = len(set([row[-1] for row in train]))
5 network = initialize_network(n_inputs, n_hidden, n_outputs)
—-> 6 train_network(network, train, l_rate, n_epoch, n_outputs)
7 predictions = list()
8 for row in test:

in train_network(network, train, l_rate, n_epoch, n_outputs)
5 outputs = forward_propagate(network, row)
6 expected = [0 for i in range(n_outputs)]
—-> 7 expected[row[-1]] = 1
8 backward_propagate_error(network, expected)
9 update_weights(network, row, l_rate)
IndexError: list assignment index out of range
```

Jason Brownlee October 11, 2018 at 8:00 am #

REPLY

I have some suggestions here:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

KS October 12, 2018 at 11:33 am #

How to:
"Regression. Change the network so that there is only
predicted."

How to get 1 output layer?

Jason Brownlee October 13, 2018 at 6:06 am

Coding algorithms from scratch is not for

I strongly encourage you to use Keras, for example:
https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/

Jerry October 16, 2018 at 9:08 am #

REPLY

Ok, just following up. I still don't get how the backprop is updating:

```
        for row in train:
            outputs = forward_propagate(network, row, alpha)
            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, alpha)
            update_weights(network, row, l_rate)
```

1. No return statement or 'global network' for backward_propagate_error or update_network to actually incorporate the new weights.

My question, are you sure this uses back-propagation? How are the weights saved and updated for each epoch?

Jason Brownlee October 16, 2018 at 2:34 pm #

REPLY

The weights are passed in by reference and modified in place.

Perhaps using Keras would be a better fit Jerry:
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

Jerry November 10, 2018 at 11:53 pm #

REPLY

You learn something new everyday! I was always under the impression that values were immutable when passing to a function in Pytho

https://stackoverflow.com/questions/986006/ho

Thank you Jason! I'm really amazed on how a
your work are referenced often in my MSDS pr

One area that I think would also be beneficial i
interpretations. The majority of our work is not
of visualizing weights, activations, and determi

Jason Brownlee November 11, 201

Thanks, happy to help.

Why do you want to view/understand the dynamics of nodes in hidden layers?

ben October 31, 2018 at 9:47 pm #

REPLY

i think this is not correct:
sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])

shouldn´t this be:
sum_error += sum([abs(0.5*(expected[i]-outputs[i])**2) for i in range(len(expected))])

see https://goo.gl/iqHJf6 page 233 (5.11).

Jason Brownlee November 1, 2018 at 6:11 am #

REPLY

This implementation is based on the book “Neural Smithing”:
https://www.amazon.com/Neural-Smithing-Supervised-Feedforward-Artificial/dp/0262527014/ref=as_li_ss_tl?ie=UTF8&linkCode=sl1&tag=inspiredalgor-20&linkId=e3db0b57249093a94ebb073983bc8b4d&language=en_US

Ayhan October 20, 2018 at 1:43 am #

REPLY

Hey Jason, first thnx for the efforts.

Maybe one thing:
wouldnt it be better (especially for beginners to that topic) to at least use something like numpy which would at least let all the matrix calculation look a bit more compact (and therefore possible to concentrate on the real topic .. which is backprop i would have guessed)?

Ok the title sais ‘implement from scratch’ but I would say at some point getting the point towards backprop is maybe more important t
it was implemented from scratch (using nothing but pla

Greets

Jason Brownlee October 20, 2018 at 5:57 am

Thanks for the feedback.

This example is not intended for writing operationa
https://machinelearningmastery.com/start-here/#de

This tutorial is to teach how to develop a net (training and evaluation) without any dependencies other than the standard lib.

Vadim November 3, 2018 at 11:10 pm #

REPLY

Just Brilliant stuff

Jason Brownlee November 4, 2018 at 6:27 am #

REPLY

Thanks.

Himanshu November 14, 2018 at 10:38 pm #

REPLY

Hi Jason
how i can use forward and backward propagation in real time data. I mean data in which we have multiple field. Fields contains Numerical values , floating values , String Values.
For an example if i want to use this technique for Titanic data how can i use it.

How can i decide how many hidden layers should be there.
How can i decide for learning rate.
How can i decide for what should be the seed value.
How can i decide for weights for such huge data.

Jason Brownlee November 15, 2018 at 5:30 am #

REPLY

You must use careful experimentation to get answers to each of these questions.

Himanshu November 15, 2018 at 6:12 pm #

Hi Jason
I have few more questions on this.

First: why you assigned “activation = weights[-1]” , why
Second: why you are looping only for two values
for i in range(len(weights) – 1) though we have three v
Third: Why you have considered only two outputs here
output for layer one =.9643898158763548
output for layer two =.9185258960543243

output for layer three = .8094918973879515
output for layer two = .7734292563511262

why you considered only last two values why not first two or any other combination.

Four: here i guess some problem by mistake you have update wrong value
expected[row[-1]] = 1
after this line you have updated expected as [1,0] from [0,0]
and why we in this i have other question why we are updating this value.

Jason Brownlee November 16, 2018 at 6:12 am #

REPLY

Too many questions for one comment (I'm a simple human), one at a time and can you please reference specific examples/lines of code otherwise I can't help.

Karim November 18, 2018 at 8:14 am #

REPLY

Hi Jason — How can i add more hidden layers? Thankx

Jason Brownlee November 19, 2018 at 6:41 am #

REPLY

I recommend using Keras to develop your network:
https://machinelearningmastery.com/start-here/#deeplearning

Leave a Reply

Name (required)

Email (will not be published) (requ

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery!

Hi, I'm Jason Brownlee, PhD
I write tutorials to help developers (*like you*) get results with machine learning.

Read More

Code Algorithms From Scratch
No libraries, just simple Python code.

Click to Get Started Now!

POPULAR

How to Develop a Deep Learning Photo Caption Ge
NOVEMBER 27, 2017

How to Develop a Neural Machine Translation Syst
JANUARY 10, 2018

Difference Between Classification and Regression in Machin
DECEMBER 11, 2017

Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

How to Develop an N-gram Multichannel Convolutional Neural Network for Sentiment Analysis
JANUARY 12, 2018

So, You are Working on a Machine Learning Problem…
APRIL 4, 2018

Encoder-Decoder Models for Text Summarization in Keras
DECEMBER 8, 2017

How to Make Predictions with Keras
APRIL 9, 2018

You might also like…

- How to Install Python for Machine Learning
- Your First Machine Learning Project in Python
- Your First Neural Network in Python
- Your First Classifier in Weka
- Your First Time Series Forecasting Project