

# Backpack

## Functional design

P.P. Elfferich  
email: pp@dia.uva.nl

1st February 2006

### 1 Introduction

Backpack will be a toolkit for training multilayer feedforward neural networks using back-propagation. Users will be able to configure all necessary settings through a graphical interface and then proceed with the actual training of the network and the storage of the trained network.

To be able to determine which configuration items are necessary for this program I have researched back-propagation and translated it into pseudo-code while keeping track of any parameters. I have also looked into ways of improving training by acceleration, generalization, preprocessing and postprocessing. The resulting total list of parameters was then used to design a prototype of the graphical interface.

### 2 Network definition

Multilayer feedforward neural networks can be defined with the following parameters.

#### 2.1 Layers

The basic shape of the neural network is defined by the number of layers and the number of neurons per layer. There are at least two layers (input and output).

#### 2.2 Neurons

The character of the network is largely defined by the type of neurons used in the network.

### 2.2.1 Neuron transfer function

The most important property of a neuron is its neuron transfer function. The most common functions used are:

- Hard-limit<sup>1</sup>

$$f(x) = \begin{cases} 1 & \text{if } (x > 0) \\ 0 & \text{else} \end{cases}$$
$$f'(x) = 1$$

- Linear

$$f(x) = x$$
$$f'(x) = 1$$

- Sigmoid

$$f(x) = \frac{1}{1+e^{-x}}$$
$$f'(x) = f(x) \times (1 - f(x))$$

- Hyperbolic tangent

$$f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$
$$f'(x) = 1 - f(x)^2$$

### 2.2.2 Neuron threshold

Neurons can have a threshold, this means they have an extra weight, attached to a constant input of -1, whose value is added to the sum of input values during propagation.

## 2.3 Connections

The way the neurons in the different layers are connected can limit the capabilities of the network but it can also increase the speed of converging.

### 2.3.1 Fully or partially connected

Most networks are fully connected and are thereby functionally only limited by the shape of the network and the type of neurons. In fully connected networks, each node in each layer is connected to each node in the following layer. Obviously, in large networks this will result in huge numbers of connections.

### 2.3.2 Deep connections

Another feature which can be used by some networks is 'deep connections'. For example nodes in hidden layers can be connected to the input layer for extra inputs.

---

<sup>1</sup>Note that this transfer function can't be differentiated, but I think  $f'(x) = 1$  works for calculating the 'gradient' in back-propagation

### 2.3.3 Shared weights

Most networks don't use shared weights, which means that each connection has its own weight. In some cases it might be more optimal to share certain weights between certain connections.

## 3 Basic back-propagation

Back-propagation can be used as a learning algorithm for multilayer feedforward networks as follows.

### 3.1 Parameters

$\alpha$  = learning rate

*max epochs* = maximum number of epochs

*error goal* = performance value goal

*max time* = maximum training time

### 3.2 Propagation

Propagation of input to output:

1. Input values are set as output values of the first (input) layer
2. For each layer from the second to the last
  - a) For each neuron in this layer
    - i. For each connection to this neuron from the lower layer
      - A. Add to the sum of inputs: *lower output value*  $\times$  *connection weight*
      - ii. Subtract this neuron's threshold value from the sum of inputs
      - iii. Calculate this neuron's output value using its transfer function:  $f(\sum inputs)$

### 3.3 Back-propagation

Back-propagation of the error from output to input layer:

1. For each neuron in the last (output) layer
  - a) Calculate this neuron's error: *target value*  $-$  *output value*
  - b) Calculate this neuron's gradient using the derivative of this neuron's transfer function:  $f'(x) \times error$   
Where  $x = \sum inputs$  and  $f(x) = output value$  (efficient for some derivatives)
2. For each layer from the second last to the first

- a) For each neuron in this layer
  - i. For each connection from this neuron to the upper layer
    - A. Add to the sum of errors:  $connection\ weight \times upper\ neuron\ gradient$
    - ii. Calculate this neuron's gradient like at step 1b:  $f'(x) \times \sum errors$
- 3. For each layer from the first to the second last
  - a) For each neuron in this layer
    - i. For each connection from this neuron to the upper layer
      - A. Calculate the new weight by adding:  $\alpha \times lower\ output\ value \times upper\ gradient$
      - B. Calculate the upper neuron's new threshold by adding:  $\alpha \times -1 \times gradient$

## 3.4 Training

### 3.4.1 Parameters

*file train* = filename from which to load the training data<sup>2</sup>

*file out* = filename for storing output of test set

*file error* = filename for storing the performance value after each epoch

*file weights* = filename for storing the final weight configuration of the trained network

*test ratio* = percentage of training data to be used for testing

### 3.4.2 Method

1. Load the training data from *file train* and set aside *test ratio* percent of that data for testing
2. Clean up any stale files (*file out*, *file error*, *file weights*)
3. Build the neural network according to the configuration and initialize all weights with random values between -1 and 1
4. For each training case
  - a) Propagate case input values
  - b) Back-propagate based on desired case output values
5. Calculate the performance value (mean sum of squared errors) of the test set:

$$\frac{1}{N} \times \sum_{i=1}^N (target\ i - output\ i)^2$$

Where N is the number of cases in the test set.

6. Add the current performance value to *file error*

---

<sup>2</sup>In most files like this one, data is stored as simple tab-separated text which can be imported and exported from programs like Excel.

7. Return to step 4 unless one of the following stop conditions is met:
  - a)  $epoch > max\ epochs^3$
  - b)  $sum\ of\ squared\ errors < error\ goal$
  - c)  $time\ used > max\ time$
8. Store the weights in *file weights* and calculate the output for all test data and store that in *file out*

## 4 Accelerated back-propagation

There are a number of different methods to accelerate training during back-propagation.

### 4.1 Momentum

Momentum uses the fact that most of the times subsequent corrections to weights are in the same direction. The previous weight correction, multiplied by a constant, is added to weight corrections. With the right momentum constant this helps skip local minima in the error gradient and also helps to stabilize the network training.

#### 4.1.1 Parameters

$\beta$  = momentum constant (typically 0.95)

#### 4.1.2 Method

Store previous weight corrections for each node and replace the weight correction formula by:  $\beta \times previous\ weight\ correction + \alpha \times lower\ output\ value \times upper\ gradient$

### 4.2 Adaptive learning rate

An effective way to accelerate convergence is to increase the learning rate  $\alpha$ , but this may cause instability and as a result the network may become oscillatory. By following some simple rules and continuously adapting the learning rate during training we can however reap the benefits of accelerated convergence without risking the danger of instability.

#### 4.2.1 Parameters

$max\ ratio$  = maximum error ratio (typically 1.04)

$factor\ dec$  = decrease factor (typically 0.7)

$factor\ inc$  = increase factor (typically 1.05)

---

<sup>3</sup>The epoch number is the number of times the complete training set has been trained.

### 4.2.2 Method

Keep track of the last epoch's mean sum of squared errors. In the training process described at 3.4.2, insert an extra step below step 5:

- Calculate the new learning rate
  - If the current mean sum of squared errors exceeds the previous value by more than a factor *maxratio* then  $\alpha$  becomes:  $\alpha \times \text{factor dec}$
  - If the current mean sum of squared errors is less than the previous value then  $\alpha$  becomes:  $\alpha \times \text{factor inc}$

## 5 Improving generalization

One of the problems that can occur during neural network training is called overfitting. The error on the training set is driven to a very small value, but when new data is presented to the network the error is large. The network has memorized the training examples, but it has not learned to generalize to new situations. The following techniques can be used to improve generalization.

### 5.1 Regularization

When using regularization the performance formula is changed to also take into account the size of the weights being used in the network. This way, networks with small weights will perform better guiding the training process to a more generalized configuration.

#### 5.1.1 Parameters

*regratio* = regularization ratio (typically 0.5)

#### 5.1.2 Method

Replace the performance formula by:

$$\frac{1 - \text{regratio}}{N} \times \sum_{i=1}^N (\text{target } i - \text{output } i)^2 + \frac{\text{regratio}}{n} \times \sum_{j=1}^n \text{weight}_j^2$$

### 5.2 Early stopping

When a network begins to overfit the training data, the error of the test set will start to rise. When the error increases for a specified number of iterations, the training is stopped, and the weights at the minimum of the error are returned.

#### 5.2.1 Parameters

*maxnrinc* = maximum number of iterations of rising error

### 5.2.2 Method

Keep track of the minimum performance value and the corresponding weights. Also count the number of times the performance value has risen in a row. In the training process described at 3.4.2, insert an extra step below step 5:

- If the performance value has dropped to a new minimum then store all weights
- If the performance value has risen *maxnrinc* times in a row then stop training and recover stored weights

## 6 Preprocessing and postprocessing

Neural network training can be made more efficient if certain preprocessing steps are performed on the network inputs and targets.

### 6.1 Minimum and maximum

Before training, it is often useful to scale the inputs and targets so that they always fall within a specified range.

#### 6.1.1 Parameters

*data min* = minimum value to scale to  
*data max* = maximum value to scale to

#### 6.1.2 Method

In the training process described at 3.4, prepend an extra step before step 4:

- Scale all training and test data so that the minimum and maximum will equal *data min* and *data max*
- Store the original minimum and maximum values for both the inputs and the outputs

If you want to subsequently use the network with any new data make sure to scale and unscale it using the stored minimum and maximum values.

## 7 Parameter list

#	Parameter	Type	Default	Min	Max	Reference
1	propagator	choice	Standard B.P.	-	-	
2	nr of layers	int	3	2	-	2.1
3	nr of neurons per layer	int	1	1	-	2.1
4	transfer function per layer	choice	Sigmoid	-	-	2.2.1
5	threshold per layer	option	1	0	1	2.2.2
6	connector per layer	choice	FullConnector	-	-	2.3
7	learning rate (alpha)	float	0.5	0	2	3.1
8	max epochs	int	0	0	-	3.1
9	error goal	float	0.001	0	1	3.1
10	max time (seconds)	int	0	0	-	3.1
11	file train	filename	-	-	-	3.4.1
12	file out	filename	-	-	-	3.4.1
13	file error	filename	-	-	-	3.4.1
14	file weights	filename	-	-	-	3.4.1
15	test ratio	int	10	0	100	3.4.1
16	use momentum	option	1	0	1	4.1
17	momentum (beta)	float	0.95	0	2	4.1.1
18	use adaptive learning rate	option	1	0	1	4.2
19	max ratio	float	1.04	1	2	4.2.1
20	factor dec	float	0.7	0	1	4.2.1
21	factor inc	float	1.05	1	2	4.2.1
22	use regularization	option	0	0	1	5.1
23	reg ratio	float	0.5	0	1	5.1.1
24	use early stopping	option	0	0	1	5.2
25	max nr inc	int	4	1	-	5.2.1
26	use min max	option	1	0	1	6.1
27	data min	float	0.1	0	1	6.1.1
28	data max	float	0.9	0	1	6.1.1

The program will be designed to be easily extended with extra functionality. Parameters 1, 4 and 6 enable the user to choose a specific implementation of a functionality interface. Other implementations can be added by implementing the same interface, adding the class name to the appropriate list and recompiling.

Note in the following interface prototype that when the number of layers (parameter 2) is changed, the number of layer option sets below is also changed to match that number. The input layer doesn't need a transfer function and the output layer doesn't need a connector. Also note that the user will be able to load and save network configurations using a standard File menu.

## 8 Interface prototype

This is a graphical interface prototype with all preceding parameters marked by number. The following three screens will be implemented as tabs.

<b>Basic options</b> -1- Propagator <input type="checkbox"/> <input checked="" type="checkbox"/> -15- Test set ratio <input type="checkbox"/> <input type="text"/>		<b>Files</b> -11- Training data <input type="text"/> -12- Test output <input type="text"/> -13- Error log <input type="text"/> -14- Final weights <input type="text"/>	
<b>Network architecture</b> -2- Nr of layers <input type="text" value="3"/>			
<b>Input layer</b> -3- Nr of neurons <input type="checkbox"/> -5- Threshold <input checked="" type="checkbox"/> -4- Transfer func. <input type="text"/> -6- Connector <input type="text"/>		<b>Hidden layer 1</b> Nr of neurons <input type="checkbox"/> Threshold <input checked="" type="checkbox"/> Transfer func. <input type="text"/> Connector <input type="text"/>	
		<b>Output layer</b> Nr of neurons <input type="checkbox"/> Threshold <input checked="" type="checkbox"/> Transfer func. <input type="text"/> Connector <input type="text"/>	
<input type="button" value="Next"/>			

<b>Back-propagation</b>			
<b>Configuration</b> -7- Learning rate <input type="checkbox"/> <input type="text"/> -9- Error goal <input type="checkbox"/> <input type="text"/> -8- Max epochs <input type="checkbox"/> -10- Max time <input type="checkbox"/> seconds		<b>Generalization</b> -22- Regularization <input type="checkbox"/> -23- Reg. ratio <input type="checkbox"/> <input type="text"/> -24- Early stopping <input type="checkbox"/> -25- Max nr inc. <input type="checkbox"/>	
<b>Acceleration</b> -16- Use momentum <input checked="" type="checkbox"/> -17- Momentum <input type="checkbox"/> <input type="text"/> -18- Use A.L.R. <input checked="" type="checkbox"/> -19- Max ratio <input type="checkbox"/> <input type="text"/> -21- Factor inc <input type="checkbox"/> <input type="text"/> -20- Factor dec <input type="checkbox"/> <input type="text"/>		<b>Processing</b> -26- Scale min-max <input checked="" type="checkbox"/> -27- Data min <input type="checkbox"/> <input type="text"/> -28- Data max <input type="checkbox"/> <input type="text"/>	
<input type="button" value="Previous"/>		<input type="button" value="Next"/>	

