Learning representations by back-propagating errors

Group 05

Madhushani H. K. E/15/209
Maliththa K.H.H. E/15/220
Muthucumarana P.N.N. E/15/233
Sankalpana W.A.P.C. E/15/325
The Authors

- David E. Rumelhart
  - Stanford University

- Geoffrey E. Hintont
  - University of Toronto

- Ronald J. Williams
  - Northeastern University

in

- 1986
What does mean by ‘Backpropagation’?

- The term *backpropagation* strictly refers only to the algorithm for computing the gradient.
- In machine learning, *backpropagation* is a widely used algorithm for training feedforward neural networks.
- Generalizations of backpropagation exists for other Artificial Neural Networks (ANNs).
- These classes of algorithms are all referred to generically as "backpropagation".
- The practice of fine-tuning the weights of a neural net
  - based on the error rate obtained in the previous epoch

- Activations of the input units are propagated forward to output layer through the connecting weights.
Why ‘Backpropagation’?

- Computes the gradient of the loss function with respect to the weights of the network for a single input–output example
- Does so efficiently, unlike a naive direct computation of the gradient
- Proper tuning of the weights ensures lower error rates
- This efficiency makes it feasible to use gradient methods
- Avoid redundant calculations of intermediate terms in the chain rule
- Image recognition and speech recognition
Back-propagating error

- Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of ANNs using gradient descent.
- Backpropagation is analogous to calculating the delta rule for a multilayer feedforward network.

backpropagation requires three things:

- Dataset
- A feedforward neural network,
- An error function
\[
\begin{pmatrix}
x_0 \\
\vdots \\
x_j \\
\vdots 
\end{pmatrix}
\]
\[
\begin{pmatrix}
y_0 \\
\vdots \\
y_j \\
\vdots 
\end{pmatrix}
\]
How Backpropagation Works


1. Input layer
2. Hidden layer(s)
3. Output layer
4. Difference in desired values
5. Backprop output layer

E/15/325 Chalani
Back-Propagation Learning Procedure

- The total input $x_j$, to unit $j$
  
  $y_i$ - output of unit $i$ connected to unit $j$
  $w_{ji}$ - weight connecting unit $i$ to unit $j$

- Resulting value is passed through a sigmoid function

\[
x_j = \sum_i y_i w_{ji}
\]

\[
y_j = \frac{1}{1 + e^{-x_j}}
\]
- Aim is to find a set of weights that gives the output vector produced by the network, same as the desired output vector.
- The total error comparing the actual and desired output vector:

\[
E = \frac{1}{2} \sum_c \sum_j (y_{j,c} - d_{j,c})^2
\]

c - index over cases (input-output pairs),
j - index over output units
y - actual state of an output unit
d - desired state.
- To minimize E by gradient descent → partial derivative of E
Let's Differentiate the above equation,

\[ \frac{\partial E}{\partial y_j} = y_j - d_j \]

Let's apply the chain rule to compute

\[ \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \frac{dy_j}{dx_j} \]
Let's differentiate the above equation to get the value of $dy_i/ dx$, and substituting gives

$$y_j = \frac{1}{1 + e^{-x_j}}$$

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} y_j (1 - y_j)$$

Let's compute how the error by changing these states and weights. For a weight $w_{ji}$, from $i$ to $j$ the derivative is

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial w_{ji}}$$

$$= \frac{\partial E}{\partial x_j} y_i$$
For the output of the $i$th unit the contribution to $aE/ay$:

$$\frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial y_i} = \frac{\partial E}{\partial x_j} w_{ji}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} w_{ji}$$
Ways of using $\frac{\partial E}{\partial w}$

1. Change the weights after every input-output case.

2. Accumulate $\frac{\partial E}{\partial w}$ over all the input-output cases before changing the weights.
   - An alternative scheme

3. Change each weight by an amount proportional to the accumulated $\frac{\partial E}{\partial w}$
   - The simplest version of gradient descent

$$\Delta w = -\varepsilon \frac{\partial E}{\partial w}$$
An improved version

\[ \Delta w(t) = -\varepsilon \frac{\partial E}{\partial w(t)} + \alpha \Delta w(t - 1) \]

- t - incremented by 1 for each sweep through the whole set of input-output cases
- alpha - is an exponential decay factor between 0 and 1
Detection of symmetry

use an intermediate layer

- weights that are symmetric about the middle of the input vector are equal in magnitude and opposite in sign.

Advantage:

- Both hidden units will receive a net input of 0 from the input units

- Weights on each side of the midpoint are in the ratio 1:2:4.
Storing the information in family trees

Set of propositions using the 12 relationships:

- (colin has-father james)
- (colin has-mother victoria)
- (james has-wife victoria)
- (charlotte has-brother colin)
- (victoria has-brother arthur)
- (charlotte has-uncle arthur)
Network
Receptive fields

- Trained on 100 of the 104 possible triples
  - White rectangles - excitatory weights
  - Black rectangles - inhibitory weights
  - Area of the rectangle - encodes the magnitude of the weight.
The layered network

Layered nets → Iterative nets.

Two complications arise in performing the mapping:

- Needs to store the history of output states of each unit
- Corresponding weights between different layers must have the same value

A set of corresponding weights
Thank You!
Q & A