UNIT -II

BPN AND BAM
Back Propagation Network

- Rumelhart (early 80’s), Werbos (74),..., explosion of neural net interest
- Multi-layer supervised learning
- Able to train multi-layer perceptrons (and other topologies)
- Uses differentiable sigmoid function which is the smooth (squashed) version of the threshold function
- Error is propagated back through earlier layers of the network
Multi-Layer Perceptrons

In contrast to perceptrons, multilayer networks can learn not only multiple decision boundaries, but the boundaries may be nonlinear. The typical architecture of a multi-layer perceptron (MLP) is shown below.

Output nodes

Internal nodes

Input nodes

To make nonlinear partitions on the space we need to define each unit as a nonlinear function (unlike the perceptron). One solution is to use the sigmoid unit. Another reason for using sigmoids are that they are continuous unlike linear thresholds and are thus differentiable at all points.
**Back-Propagation Algorithm**

Multi-layered perceptrons can be trained using the back-propagation algorithm described next.

**Goal:** To learn the weights for all links in an interconnected multilayer network.

We begin by defining our measure of error:

\[ E(W) = \frac{1}{2} \sum_d \sum_k (t_{kd} - o_{kd})^2 \]

Where:
- \( k \) varies along the output nodes and \( d \) over the training examples.

The idea is to use again a gradient descent over the space of weights to find a global minimum (no guarantee).

**Algorithm:**

1. Create a network with \( nin \) input nodes, \( n_{hidden} \) internal nodes, and \( n_{out} \) output nodes.
2. Initialize all weights to small random numbers.
3. Until error is small do:
   - Propagate example \( X \) forward through the network
   - Propagate errors backward through the network

**Forward Propagation**

Given example \( X \), compute the output of every node until we reach the output layer.

**Internal**

**Input**

**Example**

**Compute sigmoid function**
Backward Propagation

A. For each output node \( k \) compute the error:
\[
\delta_k = O_k (1-O_k)(t_k - O_k)
\]

B. For each hidden unit \( h \), calculate the error:
\[
\delta_h = O_h (1-O_h) \sum_k W_{kh} \delta_k
\]

C. Update each network weight:
\[
W_{ji} = W_{ji} + \Delta W_{ji}
\]

A momentum term, depending on the weight value at last iteration, may also be added to the update rule as follows. At iteration \( n \) we have the following:
\[
\Delta W_{ji} (n) = \eta \delta_j X_{ji} + \alpha \Delta W_{ji} (n)
\]

Where \( \alpha (0 \leq \alpha \leq 1) \) is a constant called the momentum.

1. It implements a gradient descent search over the weight space.
2. It may become trapped in local minima.
3. In practice, it is very effective.
4. How to avoid local minima?

Remarks on Back-propagation

b) Use stochastic gradient descent.
c) Use different networks with different initial values for the weights.

Multi-layered perceptrons have high representational power. They can represent the following:

1. Boolean functions. Every boolean function can be represented with a network having two layers of units.
2. Continuous functions. All bounded continuous functions can also be approximated with a network having two layers of units.
3. Arbitrary functions. Any arbitrary function can be approximated with a network with three layers of units.
Generalization and overfitting:

One obvious stopping point for backpropagation is to continue iterating until the error is below some threshold; this can lead to overfitting.

Overfitting can be avoided using the following strategies.

- Use a validation set and stop until the error is small in this set.
- Use 10-fold cross validation.
- Use weight decay; the weights are decreased slowly on each iteration.

Applications of Neural Networks

Neural networks have broad applicability to real-world business problems. They have already been successfully applied in many industries.

Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs including:

- sales forecasting
- industrial process control
- customer research
- data validation
- risk management
- target marketing
Applications of BPN-Paint Quality Assessment

Figure 3.10 The BPN system is constructed to perform paint-quality assessment. In this example, the BPN was merely a software simulation of the network described in the text. Inputs were provided to the network through an array structure located in system memory by a pointer argument supplied as input to the simulation routine.
Applications of BPN-4:1 Video Data Compression

Figure 3.7 This BPN will do four-to-one data compression.
Applications of BPN-4:1 Video Data Compression
Backpropagation Learning

Notations:

- Weights: two weight matrices:
  - from input layer (0) to hidden layer (1)
  - from hidden layer (1) to output layer (2)
  - weight from node 1 at layer 0 to node 2 in layer 1
- Training samples: pair of
  \[ \{(x_p, d_p) \mid p = 1, \ldots, P\} \]
  so it is supervised learning
- Input pattern:
  \[ x_p = (x_{p,1}, \ldots, x_{p,n}) \]
- Output pattern:
  \[ o_p = (o_{p,1}, \ldots, o_{p,k}) \]
- Desired output:
  \[ d_p = (d_{p,1}, \ldots, d_{p,k}) \]
- Error:
  \[ l_{p,j} = o_{p,j} - d_{p,j} \]
  sum square error
  \[ = \sum_{p=1}^{P} \sum_{j=1}^{K} (l_{p,j})^2 \]
  This error drives learning (change the weights
     \[ W^{(1,0)} \] and \[ W^{(2,1)} \])
• Sigmoid function again:
  – Differentiable:

\[
S(x) = \frac{1}{1 + e^{-x}}
\]

\[
S'(x) = -\frac{1}{(1 + e^{-x})^2} \cdot (1 + e^{-x})'
\]

\[
= -\frac{1}{(1 + e^{-x})^2} \cdot (-e^{-x})
\]

\[
= \frac{1}{1 + e^{-x}} \cdot e^{-x}
\]

\[
= \frac{S(x)}{1 + e^{-x}}
\]

  – When \( |\text{net}| \) is sufficiently large, it moves into one of the two saturation regions, behaving like a threshold or ramp function.

• Chain rule of differentiation

If \( z = f(y), y = g(x), x = h(t) \) then

\[
\frac{dz}{dt} = \frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dt} = f'(y)g'(x)h'(t)
\]
Backpropagation Learning

• **Forward computing:**
  – Apply an input vector \( x \) to input nodes
  – Computing output vector \( x^{(1)} \) on hidden layer
    \[
    x_j^{(1)} = S(\text{net}_j^{(1)}) = S(\sum_i w_{j,i}^{(1,0)} x_i)
    \]
  – Computing the output vector \( o \) on output layer
    \[
    o_k = S(\text{net}_k^{(2)}) = S(\sum_j w_{k,j}^{(2,1)} x_j^{(1)})
    \]
  – The net is said to be a map from input \( x \) to output \( o \)

• **Objective of learning:**
  – reduce sum square error
    \[
    \sum_{p=1}^{P} \sum_{j=1}^{K} (l_{p,j})^2
    \]
  for the given \( P \) training samples as much as possible (to zero if possible)
Backpropagation Learning

• **Idea of BP learning:**
  – Update of weights in $w^{(2, 1)}$ (from hidden layer to output layer):
    delta rule as in a single layer net using sum square error
  – Delta rule is not applicable to updating weights in $w^{(1, 0)}$ (from input and hidden layer) because we don’t know the desired values for hidden nodes
  – **Solution:** Propagating errors at output nodes down to hidden nodes, these computed errors on hidden nodes drives the update of weights in $w^{(1, 0)}$ (again by delta rule), thus called error **BACKPROPAGATION (BP)** learning
  – How to compute errors on hidden nodes is the key
  – Error backpropagation can be continued downward if the net has more than one hidden layer
Backpropagation Learning

• **Generalized delta rule:**
  - Consider sequential learning mode: for a given sample \((x_p, d_p)\)
    \[
    E = \sum_k (l_{p,k})^2
    \]
  - Update of weights by gradient descent
    For weight in \(w^{(2,1)}\):
    \[
    \Delta w_{k,j}^{(2,1)} \propto (- \partial E / \partial w_{k,j}^{(2,1)})
    \]
    For weight in \(w^{(1,0)}\):
    \[
    \Delta w_{j,i}^{(1,0)} \propto (- \partial E / \partial w_{j,i}^{(1,0)})
    \]
  - Derivation of update rule for \(w^{(2,1)}\):
    since \(E\) is a function of \(l_k = d_k - o_k\), \(d_k - o_k\) is a function \(\text{net}_k^{(2)}\)
    and \(\text{net}_k^{(2)}\) is a function of \(w_{k,j}^{(2,1)}\) by chain rule
    \[
    \frac{\partial E}{\partial w_{k,j}^{(2,1)}} = \frac{\partial E}{\partial (d_k - o_k)} \frac{\partial (d_k - o_k)}{\partial \text{net}_k^{(2)}} \frac{\partial \text{net}_k^{(2)}}{\partial w_{k,j}^{(2,1)}}
    \]
    \[
    = 2(d_k - o_k) S'(\text{net}_k^{(2)}) x_j^{(1)}.
    \]
Backpropagation Learning

- Derivation of update rule for $w_{j,i}^{(1,0)}$

  consider hidden node $j$:
  weight $w_{j,i}^{(1,0)}$ influences $net_j^{(1)}$
  it sends $S(net_j^{(1)})$ to all output nodes

\[ E = \sum_k (d_k - o_k)^2, \quad o_k = S(net_k^{(2)}), \quad net_k^{(2)} = \sum_j x_j^{(1)} w_{k,j}^{(2,1)}, \]
\[ x_j^{(1)} = S(net_j^{(1)}), \quad net_j^{(1)} = \sum_i x_i w_{j,i}^{(1,0)} \]

by chain rule, we have

\[ \frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^{K} \left\{ -2(d_k - o_k) S'(net_k^{(2)}) w_{k,j}^{(2,1)} S'(net_j^{(1)}) x_i \right\} \]
Backpropagation Learning

– Update rules:

for outer layer weights $w^{(2, 1)}$:

$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = -2(d_k - o_k)S'(net_k^{(2)})x_j^{(1)}$$

$$\Delta w_{k,j}^{(2,1)} = \eta \times \delta_k \times x_j^{(1)}$$

where

$$\delta_k = (d_k - o_k)S'(net_k^{(2)})$$

for inner layer weights $w^{(1, 0)}$:

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^{K} \left\{ -2(d_k - o_k)S'(net_k^{(2)}) \cdot w_{k,j}^{(2,1)} \cdot S'(net_j^{(1)}) \cdot x_i \right\}$$

$$\Delta w_{j,i}^{(1,0)} = \eta \times \mu_j \times x_i$$

where

$$\mu_j = \left( \sum_k \delta_k w_{k,j}^{(2,1)} \right) S'(net_j^{(1)})$$

Weighted sum of errors from output layer
Algorithm Backpropagation;

Start with randomly chosen weights;

while MSE is unsatisfactory and computational bounds are not exceeded, do

for each input pattern $x_p$, $1 \leq p \leq P$,

Compute hidden node inputs ($net_{p,j}^{(1)}$);
Compute hidden node outputs ($x_{p,j}^{(1)}$);
Compute inputs to the output nodes ($net_{p,k}^{(2)}$);
Compute the network outputs ($o_{p,k}$);

Modify outer layer weights:

$$ \Delta w_{k,j}^{(2,1)} = \eta (d_{p,k} - o_{p,k}) S'(net_{p,k}^{(2)}) x_{p,j}^{(1)} $$

Modify weights between input & hidden nodes:

$$ \Delta w_{j,i}^{(1,0)} = \eta \sum_k \left( (d_{p,k} - o_{p,k}) S'(net_{p,k}^{(2)}) w_{k,j}^{(2,1)} \right) S'(net_{p,j}^{(1)}) x_{p,i} $$

end-for

end-while.
Backpropagation Learning

• **Pattern classification:**
  - Two classes: 1 output node
  - N classes: binary encoding (log N) output nodes
    better using N output nodes, a class is represented as 
    \((0,\ldots, 0,1, 0,\ldots, 0)\)
  - With sigmoid function, nodes at output layer will never be 1 or 0, but either \(1-\epsilon\) or \(\epsilon\).
  - Error reduction becomes slower when moving into saturation regions (when \(\epsilon\) is small).
  - For fast learning, for a given error bound \(\epsilon\),
    set error \(l_{p,k} = 0\) if \(|d_{p,k} - o_{p,k}| \leq \epsilon\)
  - When classifying a input \(x\) using a trained BP net, classify it to the \(k^{th}\) class if with \(d_k > d_l\) for all \(l \neq k\)
Backpropagation Learning

• **Pattern classification**: an example
  – Classification of myoelectric signals
    * Input pattern: 3 features (NIF, VT, RR), normalized to real values between 0 and 1
    * Output patterns: 2 classes: (success, failure)
  – Network structure: 2-5-3
    * 3 input nodes, 2 output nodes,
    * 1 hidden layer of 5 nodes
    * $\eta = 0.95$, $\alpha = 0.4$ (momentum)
  – Error bound $\varepsilon = 0.05$
  – 332 training samples
  – Maximum iteration = 20,000
  – When stopped, 38 patterns remain misclassified
Hopfield Memories is of 2 types
- Discrete Hopfield NNs
- Continuous Hopfield NNs

Associative Memories is of 2 types
- Hopfield Memory
- Bidirectional Memory
Figure 3.3 The three-layer BPN architecture follows closely the general network description given in Chapter 1. The bias weights, \( \theta_{jk} \), and \( \theta_{ik} \), and the bias units are optional. The bias units provide a fictitious input value of 1 on a connection to the bias weight. We can then treat the bias weight (or simply, bias) like any other weight. It contributes to the net-input value to the unit, and it participates in the learning process like any other weight.
recordBPN =
    INUNITS : "layer; {locate input layer}
    OUTUNITS : "layer; {locate output units}
    LAYERS : "layer[]; {dynamically sized network}
        alpha,
        eta : float;
    end record;

Figure 3.11 The BPN data structure is shown without the arrays for the error and last_delta terms for clarity. As before, the network is defined by a record containing pointers to the subordinate structures, as well as network-specific parameters. In this diagram, only three layers are illustrated, although many more hidden layers could be added by simple extension of the layer.ptr array.
Feedforward/Feedback NNs

Feedforward NNs
- The connections between units do not form cycles.
- Usually produce a response to an input quickly.
- Most feedforward NNs can be trained using a wide variety of efficient algorithms.

Feedback or recurrent NNs
- There are cycles in the connections.
- In some feedback NNs, each time an input is presented, the NN must iterate for a potentially long time before it produces a response.
- Usually more difficult to train than feed forward NNs.
Supervised-Learning NNs

• **Feedforward NNs**
  – Perceptron
  – Adaline, Madaline
  – Backpropagation (BP)
  – Artmap
  – Learning Vector Quantization (LVQ)
  – Probabilistic Neural Network (PNN)
  – General Regression Neural Network (GRNN)

• **Feedback or recurrent NNs**
  – Brain-State-in-a-Box (BSB)
  – Fuzzy Cognitive Map (FCM)
  – Boltzmann Machine (BM)
  – Backpropagation through time (BPTT)
Unsupervised Learning NNs

- **Feedforward NNs**
  - Learning Matrix (LM)
  - Sparse Distributed Associative Memory (SDM)
  - Fuzzy Associative Memory (FAM)
  - Counterprogation (CPN)

- **Feedback or Recurrent NNs**
  - Binary Adaptive Resonance Theory (ART1)
  - Analog Adaptive Resonance Theory (ART2, ART2a)
  - Discrete Hopfield (DH)
  - Continuous Hopfield (CH)
  - Discrete Bidirectional Associative Memory (BAM)
  - Kohonen Self-organizing Map/Topology-preserving map (SOM/TPM)
The Hopfield NNs

• In 1982, Hopfield, a Caltech physicist, mathematically tied together many of the ideas from previous research.
• A fully connected, symmetrically weighted network where each node functions both as input and output node.
• Used for
  – Associated memories
  – Combinatorial optimization
Figure 4.1 This figure shows the Hamming cube in three-dimensional space. The entire three-dimensional Hamming space, $H^3$, comprises the eight points having coordinate values of either $-1$ or $+1$. In this three-dimensional space, no other points exist.
Figure 4.11 This schematic illustrates the pattern of inhibitory connections between PEs for the TSP problem. Unit a illustrates the inhibition between units on a single row, unit b shows the inhibition within a single column, and unit c shows the inhibition of units in adjacent columns. The global inhibition is not shown.
Associative Memories

- An associative memory is a content-addressable structure that maps a set of input patterns to a set of output patterns.
- Two types of associative memory: autoassociative and heteroassociative.
- Auto-association
  - retrieves a previously stored pattern that most closely resembles the current pattern.
- Hetero-association
  - the retrieved pattern is, in general, different from the input pattern not only in content but possibly also in type and format.
Associative Memories

Auto-association

Hetero-association

memory

Niagara

Waterfall
Optimization Problems

- Associate costs with energy functions in Hopfield Networks
  - Need to be in quadratic form

- Hopfield Network finds local, satisfactory solutions, doesn't choose solutions from a set.

- Local optima, not global.
The Discrete Hopfield NNs
Figure 4.2  The BAM shown here has $n$ units on the $x$ layer, and $m$ units on the $y$ layer. For convenience, we shall call the $x$ vector the input vector, and call the $y$ vector the output vector. In this network, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$. All connections between units are bidirectional, with weights at each end. Information passes back and forth from one layer to the other, through these connections. Feedback connections at each unit may not be present in all BAM architectures.
BAM Energy landscape
Auto associative bam architecture

Figure 4.4 The autoassociative BAM architecture has an equal number of units on each layer. Note that we have omitted the feedback terms to each unit.
Single layer structure of bam

Figure 4.5 The autoassociative BAM can be reduced to a single-layer structure. Notice that, when the reduction is carried out, the feedback connections to each unit reappear.
Figure 4.6 This figure shows the Hopfield-memory architecture without the feedback connections to each unit. Eliminating these connections explicitly forces the weight matrix to have zeros on the diagonal. We have also added external input signals, $I_i$, to each unit.
Inhibitory connections between pES FOR TSP PROBLEM

Figure 4.11 This schematic illustrates the pattern of inhibitory connections between PE{s} for the TSP problem: Unit a illustrates the inhibition between units on a single row, unit b shows the inhibition within a single column, and unit c shows the inhibition of units in adjacent columns. The global inhibition is not shown.
Backpropagation Learning

• **Architecture:**
  - **Feedforward** network of at least one layer of non-linear hidden nodes, e.g., # of layers $L \geq 2$ (not counting the input layer)
  - Node function is differentiable
    - most common: sigmoid function
• **Learning:** supervised, error driven, generalized delta rule
• Call this type of nets BP nets
• The weight update rule
  (gradient descent approach)
• Practical considerations
• Variations of BP nets
• Applications
Review questions

• Discuss the role of generalized delta rule in neural networks.
• What is the importance of hamming distance in visualizing the data space.
• Discuss the pros and cons of BPN architecture.
• State the importance of BAM energy function.
• How the Hopfield memory model is useful for optimization problems.
References


• cortex.snowcron.com

• pcrochat.online.fr/webus/tutorial/BPN_tutorial.html/neural_networks.htm