Neural Networks

Backpropagation

Dr. Petr Musilek

Department of Electrical and Computer Engineering
University of Alberta

Fall 2019

The Multilayer Perceptron (MLP)

- Training of synaptic weights is difficult when dealing with more than 1 neuron, let alone multiple layers of neurons
- MLP uses a training method called backpropagation (BP) learning which uses (again) the idea of gradient descent to determine weight modifications to minimize error.
- This is the same concept as used in ADALINE, but
  a) the actual system output is used (rather than the product of linear combination with a neuron); and
  b) the concept is extended backward into the system to allow for adjustment of cascaded layer weights

The Multilayer Perceptron (MLP)

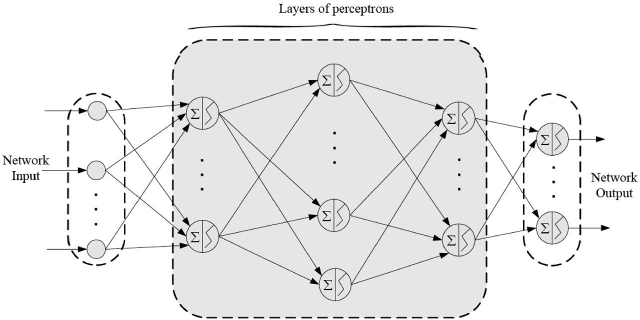
- Training of synaptic weights is difficult when dealing with more than 1 neuron, let alone multiple layers of neurons
- MLP uses a training method called backpropagation (BP) learning which uses (again) the idea of gradient descent to determine weight modifications to minimize error.
- This is the same concept as used in ADALINE, but
  a) the actual system output is used (rather than the product of linear combination with a neuron); and
  b) the concept is extended backward into the system to allow for adjustment of cascaded layer weights

The Multilayer Perceptron (MLP)

- Training of synaptic weights is difficult when dealing with more than 1 neuron, let alone multiple layers of neurons
- MLP uses a training method called backpropagation (BP) learning which uses (again) the idea of gradient descent to determine weight modifications to minimize error.
- This is the same concept as used in ADALINE, but
  a) the actual system output is used (rather than the product of linear combination with a neuron); and
  b) the concept is extended backward into the system to allow for adjustment of cascaded layer weights

MLP Topology

- The networks of this type are feedforward and contain one or more hidden layers
- The input layer is often referred to as a *buffer layer* and the weights of its inputs are held at unity



Cumulative error

$$E_c = \sum_{k=1}^{n} E(k) = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q} \left[t_i(k) - o_i(k)\right]^2$$

Cummulative error

Cycling through the iterations (training data)

Cycling through the output neurons

Note:

The outer sum (over k, training data) is used only for *batch training*. When performing *on–line training*, it is not included (next slide).

Cumulative error

$$E_c = \sum_{k=1}^{n} E(k) = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q}\left[t_i(k) - o_i(k)\right]^2$$

Cummulative error

Cycling through the iterations (training data)

Cycling through the output neurons

Note:

The outer sum (over k, training data) is used only for *batch training*. When performing *on–line training*, it is not included (next slide).

Cumulative error

$$E_c = \sum_{k=1}^{n} E(k) = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q} \left[t_i(k) - o_i(k)\right]^2$$

Cummulative error

Cycling through the iterations (training data)

Cycling through the output neurons

Note:

The outer sum (over k, training data) is used only for *batch training*. When performing *on–line training*, it is not included (next slide).

Cumulative error

$$E_c = \sum_{k=1}^{n} E(k) = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q}\left[t_i(k) - o_i(k)\right]^2$$

Cummulative error

Cycling through the iterations (training data)

Cycling through the output neurons

Note:

The outer sum (over k, training data) is used only for *batch training*. When performing *on–line training*, it is not included (next slide).

Cumulative error

$$E_c = \sum_{k=1}^{n} E(k) = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q}\left[t_i(k) - o_i(k)\right]^2$$

Cummulative error

Cycling through the iterations (training data)

Cycling through the output neurons

Note:

The outer sum (over k, training data) is used only for *batch training*. When performing *on–line training*, it is not included (next slide).

Training: batch vs. on–line

Training can be stated as an optimization problem: finding set of weights that minimizes error E.

- In *on–line* training, weight modifications are made pattern by pattern.

$$\min_{w} E(k) = \frac{1}{2}\sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

- In *batch* (or off–line) training, all training patterns are presented to the system before weights are updated.

$$\min_{w} E_c = \frac{1}{2}\sum_{k=1}^{n}\sum_{i=1}^{q} [t_i(k) - o_i(k)]^2$$

On–line training

Similarly as before (in the case of ADALINE):

$$\Delta \mathbf{w}^{(l)} = -\eta \nabla_{\mathbf{w}^{(l)}} E = -\eta \frac{\partial E(k)}{\partial \mathbf{w}^{(l)}}$$

Using chain rule:

$$\Delta \mathbf{w}^{(l)} = \Delta w_{ij}^{(l)} = -\eta \left[\frac{\partial E(k)}{\partial o_i^{(l)}} \right] \left[\frac{\partial o_i}{\partial tot_i^{(l)}} \right] \left[\frac{\partial tot_i}{\partial w_{ij}^{(l)}} \right]$$

$\frac{\partial E(k)}{\partial o_i^{(l)}}$ – how the error varies with a change in the output

$\frac{\partial o_i}{\partial tot_i^{(l)}}$ – how the output varies with a change to the neuron inputs

$\frac{\partial tot_i}{\partial w_{ij}^{(l)}}$ – how the total input vary with changes to the specific weight

On–line training

Similarly as before (in the case of ADALINE):

$$\Delta \mathbf{w}^{(l)} = -\eta \nabla_{\mathbf{w}^{(l)}} E = -\eta \frac{\partial E(k)}{\partial \mathbf{w}^{(l)}}$$

Using chain rule:

$$\Delta \mathbf{w}^{(l)} = \Delta w_{ij}^{(l)} = -\eta \left[\frac{\partial E(k)}{\partial o_i^{(l)}} \right] \left[\frac{\partial o_i}{\partial tot_i^{(l)}} \right] \left[\frac{\partial tot_i}{\partial w_{ij}^{(l)}} \right]$$

$\frac{\partial E(k)}{\partial o_i^{(l)}}$ – how the error varies with a change in the output

$\frac{\partial o_i}{\partial tot_i^{(l)}}$ – how the output varies with a change to the neuron inputs

$\frac{\partial tot_i}{\partial w_{ij}^{(l)}}$ – how the total input vary with changes to the specific weight

Backpropagation algorithm

For the output layer, $l = L$, we can write:

$$\begin{aligned}
\left[\frac{\partial E(k)}{\partial o_i^{(L)}}\right] &= \frac{\partial}{\partial o_i^{(L)}}\frac{1}{2}(t_i - o_i^{(L)})^2 = -1 \cdot (t_i - o_i^{(L)}) \\
\left[\frac{\partial o_i^{(L)}}{\partial tot_i^{(L)}}\right] &= \frac{\partial}{\partial tot_i^{(L)}}\left(f(tot_i^{(L)})\right) \\
\left[\frac{\partial tot_i^{(L)}}{\partial w_{ij}^{(L)}}\right] &= x_{ij}^{(L)} = o_j^{(L-1)}
\end{aligned}$$

algorithm ...

$$\left[\frac{\partial o_i^{(L)}}{\partial tot_i^{(L)}}\right] = \frac{\partial}{\partial tot_i^{(L)}}(\text{activation function}(tot_i^{(L)}))$$

For sigmoidal function (nicely differentiable - necessary for BP)

$$\begin{aligned} f'(tot_i^{(L)}) = \frac{\partial}{\partial tot_i^{(L)}}\left(\frac{1}{1+e^{-tot_i}}\right) &= -\frac{1}{(1+e^{-tot_i^{(L)}})^2}\cdot(-e^{-tot_i^{(L)}}) \\ &= \left(\frac{1}{1+e^{-tot_i}}\right)\cdot\left(\frac{1+e^{-tot_i^{(L)}}-1}{1+e^{-tot_i^{(L)}}}\right) \\ &= f(tot_i^{(L)})\cdot(1-f(tot_i^{(L)})) \end{aligned}$$

Put together: $\Delta w_{ij}^{(L)} = \eta(t_i - o_i^{(L)})[\, f'(tot_i^{(L)})]o_j^{(L-1)}$

So far so good, but:

This is only useful for the output layer neurons!

Error propagation

The approach now is to isolate the terms that allow us to propagate the error back into the layers before the output.

$$
\begin{aligned}
\text{For output layer:}\quad & \Delta w_{ij}^{(L)} = \eta \delta_i^{(L)} o_j^{(L-1)} \\
\text{where:}\quad & \delta_i^{(L)} = (t_i - o_i^{(L)}) f'(tot_i^{(L)}) \\
\text{For hidden layer(s):}\quad & \Delta w_{ij}^{(l)} = \eta \delta_i^{(l)} o_j^{(l-1)} \\
\text{where:}\quad & \delta_i^{(l)} = f'(tot_i^{(l)}) \sum_{p=1}^{n_i} \delta_p^{(l+1)} w_{p_i}^{(l+1)}
\end{aligned}
$$

δ is called *error signal*. Because error signals of layer $(l+1)$ are used to determine the error signal of neurons in layer (l), this algorithm is called (error) *backpropagation*.

BP algorithm summary

1. Initialize weights to small random values;
2. Select a piece of training data;
3. Propagate the input through the network;
4. Calculate the total cumulative error to this iteration

   $$E_c = E_c + E(k)$$

   and the error signal for the output layer neurons [sigmoid]

   $$\delta_i = (t_i - o_i)o_i(1 - o_i)$$

5. Update the output layer weights $\Delta w_{ij} = \eta \delta_i o_j$ and proceed backward using [sigmoid]:

$$\delta_i = o_i^l (1 - o_i^l) \sum_{p=1}^{n_i} \delta_p^{(l+1)} w_{p_i}^{(l+1)}$$

6. Repeat with next training data
7. If the cumulative error, E_c is within tolerances, terminate. Otherwise, continue with another epoch (Steps 2–6)

Batch learning

- For batch learning, weights are only updated at the end of an epoch.
- However, individual $\Delta w_{ij}^{(l)}(k)$ are calculated for each pattern k using the same procedure as in online learning.
- At the end of the epoch (all patterns presented) , weights are updated by

$$\Delta w_{ij}^{(l)} = \sum_{k=1}^{n} \Delta w_{ij}^{(l)}(k)$$

(the sum of incremental weight updates for all patterns)