Foundations of Artificial Intelligence

CS472/3
Lecture #39

Bart Selman

Slide CS472–1

Backpropagation

In order to learn multi-layer neural nets,
we need another learning algorithm.
(invented ca. 1984)
A multi-layer net has one or more hidden layers.

We will consider the **backpropagation** algorithm for training such networks.
See also R&N. Here we will present a more detailed example.

Slide CS472–2

Backpropagation

Slide CS472–3

Based on Nilsson (Stanford)

Note that in the perceptron case, we looked at the output value, compared it to the desired value and changed the weights accordingly.

We want to do something similar in the multi-layer case but one difficulty is to determine the error in the outputs of the hidden units.
Luckily, we can approximate those errors by "backpropagating" the final output error.
To do so, we need an activition function that is differentiable.

Slide CS472–4

We use the **sigmoid** function to compute our output values.

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

The derivative of $f(x)$ is:

$f'(x) = f(x) \times (1 - f(x))$

Slide CS472–5

Figure of sigmoid.

Note: largest derivative at $x = 0$
That's where neuron is most sensitive to weight changes (effect of changes is well "controlled").

Slide CS472–6

Output value of neuron is simply the weighted sum of its input “pushed” through the sigmoid.

$f(s) = \frac{1}{1+e^{-s}}$, where
$s = \sum_{k=1}^{k=n+1} w_k \times x_k$

Again, we assume the threshold is replaced by an extra input fixed at 1. Inputs: 0 or 1.

Slide CS472–7

Figure 3.5 from Nilsson.
Needed to introduce notation.

Slide CS472–8

Setup and notation:

k-layer network.
Input vector:
$\vec{X^{(0)}} =< x_1^{(0)}, x_2^{(0)}, \ldots, x_{m_0}^{(0)} >$
The first layer has m_1 units and its output is:
$\vec{X^{(1)}} =< f_1^{(1)}, f_2^{(1)}, \ldots, f_{m_1}^{(1)} >$
The weights for the ith unit in the first layer are given by:
$\vec{W_i^{(1)}} =< w_{1,i}^{(1)}, w_{2,i}^{(1)}, \ldots, w_{m_0,i}^{(1)} >$

These outputs $\vec{X^{(1)}}$ are fed into the second layer of m_2 units.
The number of units in the j-th layer is m_j.

In general, the weight vector of unit i in the j-th layer is $\vec{W_i^{(j)}}$ with components $w_{l,i}^j$ for $l = 1, m_{(j-1)} + 1$.
Note that the previous layer had $m_{(j-1)}$ units (and thus outputs) all connected to each unit in the j-th layer. We use one additional weight and fixed input to model the threshold. (Not given on previous slide.)

The weighted sum of the inputs to i-th sigmoid unit in the j-th layer is denoted by $s_i^{(j)}$

The output is $f_i^{(j)} = \frac{1}{1+e^{-s_i^{(j)}}}$

We have:

$s_i^{(j)} = \vec{X}^{(j-1)} \, . \, \vec{W}^{(j)}$

We used the vector dot product, i.e.,

$s_i^{(j)} = \sum_{l=1}^{l=m_{(j-1)}+1} x_l^{(j-1)} \times w_{l,j}^{(j)}$

$x_l^{(j-1)}$ is the output of unit l in the previous layer. It's connected with weight $w_{l,i}^{(j)}$ to the i-th unit in the j-th layer.

Slide CS472–11

Note: again, concerning the "+1" here, we assume that an extra "1" is added to the input vector and an extra weight, to model the threshold.

Finally, the k-th layer is the output layer.

It has a single unit with input $s^{(k)}$ (weighted sum) and output value $f^{(k)} = f$.

Slide CS472–12

The objective of the backprogation algorithm is
to minimize the output error on each example.

That is, we want to minimize:
$(L - f)^2$
Where, L the label (0 or 1) of the input example under consideration and f is the output of the network given that example.

Note: we will update the weights after each example
An alternative approach considers the combined error over the total training set and update after seeing all examples. In the limit the approaches are the same.

Slide CS472–13

A key observation is that the error $(L - f)^2$ is only a function of the weights.
Note: the number of units etc. is fixed.
Also, the inputs are fixed, since we are considering a particular example.

The idea is now to do a **gradient** descent in the weight space to minimize the error.

The derivation (basically calculus) is somewhat involved but not difficult.

Slide CS472–14

Here we just give the results of the calculation.

First, the weight adjustment for the single unit
in the output layer is given by:

$$\vec{W}^{(k)} \leftarrow \vec{W}^{(k)} + \alpha \times (d - f) \times f \times (1 - f) \times \vec{X}^{(k-1)}$$

Note: f is the output of the unit; d is desired output.
This rule is analogous to the perceptron rule, except
that we are now using the sigmoid.
The $(d - f)$ is the error signal.
α is the learning rate (a constant chosen by the user).
$f \times (1 - f)$ comes from the derivative of the sigmoid.
$\vec{X}^{(k-1)}$ is the input to the unit under consideration.

Slide CS472–15

So, again we're adding (subtracting) the input vector,
depending on $(d - f)$.
Please check for yourself that the correction is
in the right direction!
We're basically going to do something similar
for the weights in the hidden layer. The only problem
is that we don't have a direct measure of the error
in the outputs on the hidden units.
However, we can estimate those errors by considering
the constribution (an estimate) of each unit to the
final output value (layer k).

Slide CS472–16

Starting with the final layer and moving backwards,
we compute for the i-th unit in the j-th layer:

$\delta_i^{(j)} = f_i^j(1 - f_i^j) \sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)} w_{i,l}^{(j+1)}$
Note that $f_i^{(j)}$ is the output value of the unit.
So, the δ for a unit in the i-th layer is
computed by considering the delta in the $i+1$-th layer.

The base case is the output layer k:
$\delta^{(k)} = (d - f) \times f \times (1 - f)$
I.e., the real output error times the gradient.
This values is propagated backwards to get error measures
for the internal units (times the gradient again).

Slide CS472–17

Finally, using these δ's, we can compute the
weight updates for the hidden units:

$\vec{W}_i^{(j)} \leftarrow \vec{W}_i^{(j)} + \alpha \times \delta_i^j \times \vec{X}^{(j-1)}$

Verify that this rule is consistent with the
update rule for the final node. (Check $j = k$;
drop i, since only one unit in layer.)

Slide CS472–18

Although, the rules are somewhat intuitive, only by doing the full derivation, can one explain all the terms.

Let us consider an example of the procedure in action.

Slide CS472–19

Figure 3.6 Nilsson.

Slide CS472–20

We have two layers ($k = 2$).
Two units in first layer, with a total
of three inputs ($x_1^{(0)}, x_2^{(0)}, x_3^{(0)}$).
$x_3^{(0)}$ will be fixed to 1.
The two units in the first layer are connected
to a node in the final layer. This node
has three inputs: ($x_1^{(1)}, x_2^{(1)}, x_3^{(1)}$).
$x_1^{(1)}$ is the output of the 1st unit in the 1st layer.
$x_2^{(1)}$ is the output of the 2nd unit in the 2nd layer.
$x_3^{(1)}$ is fixed at 1.
Note the given weights in the figure.

Slide CS472–21

We want to train the network to capture the
following patterns:

ex 1.) $x_1^{(0)} = 1, x_2^{(0)} = 0, x_3^{(0)} = 1, d = 0$
ex 2.) $x_1^{(0)} = 0, x_2^{(0)} = 0, x_3^{(0)} = 1, d = 1$
ex 3.) $x_1^{(0)} = 0, x_2^{(0)} = 1, x_3^{(0)} = 1, d = 0$
ex 4.) $x_1^{(0)} = 1, x_2^{(0)} = 1, x_3^{(0)} = 1, d = 1$

Again, d is the desired output; $x_3^{(0)}$ is the fixed unit.
Let's consider the update after the first pattern.

Slide CS472–22

Ex 1.) gives input vector $< 1, 0, 1 >$, which leads via the sigmoid to the following output values:

$f_1^{(1)} = \frac{1}{1+e^{-2}} = 0.881$
$f_2^{(1)} = \frac{1}{1+e^{-0}} = 0.5$
$f = \frac{1}{1+e^{-(3\times 0.881+(-2)\times 0.5-1)}} = 0.665$

Slide CS472–23

We now compute the values for the δ's.
First the base case:

$\delta^{(2)} = (0 - 0.665) \times 0.665 \times (1 - 0.665) = -0.148$

Backpropagating through the weights gives:

$\delta_1^{(1)} = 0.881 \times (1 - 0.881) \times (-0.148 \times 3) = -0.047$
$\delta_2^{(1)} = 0.5 \times (1 - 0.5) \times (-0.148 \times -2) = 0.074$

Double-check at least one of these!

Slide CS472–24

After computing the δ's, we can now update the weights. (Use learning rate $\alpha = 1$.) We get for the new weights:

$\vec{W}_1^{(1)} =< 1.953, -2.0, -0.047 >$

$\vec{W}_2^{(1)} =< 1.074, 3.0, -0.926 >$

$\vec{W}^{(2)} =< 2.870, -2.074, -1.148 >$

Slide CS472–25

Aside: the original weights were:

$\vec{W}_1^{(1)} =< 2.0, -2.0, 0.0 >$

$\vec{W}_2^{(1)} =< 1.0, 3.0, -1.0 >$

$\vec{W}^{(2)} =< 3.0, -2.0, -1.0 >$

Slide CS472–26

Let's do an example calculation of the first weight vector.

$\overrightarrow{W_1}^{(1)} =< 1.953, -2.0, -0.047 >$

$$w_{1,1}^{(1)} = w_{1,1}^{(1)} + (1 \times \delta_1^1 \times x_1^{(0)})$$
$$= 2 + (1 \times (-0.047) \times 1) = 1.953$$

$$w_{1,2}^{(1)} = w_{1,2}^{(1)} + (1 \times \delta_1^1 \times x_2^{(0)})$$
$$= -2 + (1 \times (-0.047) \times 0) = -2.0$$

$$w_{1,3}^{(1)} = w_{1,3}^{(1)} + (1 \times \delta_1^1 \times x_3^{(0)})$$
$$= 0 + (1 \times (-0.047) \times 1) = -0.047$$

Slide CS472–27

Let's do the calculation of the third weight vector.

$\overrightarrow{W}(2) =< 2.870, -2.074, -1.148 >$ $w_1^{(2)} = w_1^{(2)} + (1 \times \delta^2 \times x_1^{(1)})$
$$= 3 + (1 \times (-0.148) \times 0.881)$$
$$= 2.870$$

$$w_2^{(2)} = w_2^{(2)} + (1 \times \delta^2 \times x_2^{(1)})$$
$$= -2 + (1 \times (-0.148) \times 0.5)$$
$$= -2.074$$

$$w_3^{(2)} = w_3^{(2)} + (1 \times \delta^2 \times x_3^{(1)})$$
$$= -1 + (1 \times (-0.148) \times 1)$$
$$= -1.148$$

Slide CS472–28

Note that the weights $w_{1,2}^{(1)}$ and $w_{2,2}^{(1)}$
don't change. Why?

How does the output value (f) of final unit change?
Is this in the right direction?