Neural networks (Ch. 12)

Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high. Get some experts and minimize the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin.
Neural network: feed-forward

You try Bat on this: \{WB=0, LE=-1, CH=1\}

Assume (for now) output = sum input

if Output(Node 5) > 0, guess mammal
Neural network: feed-forward

Output is -7, so bats are not mammal... Oops...

if Output(Node 5) > 0, guess mammal
Neural network: feed-forward

In fact, this is no better than our 1 node NN

This is because we simply output a linear combination of weights into a linear function (i.e. if f(x) and g(x) are linear... then g(x)+f(x) is also linear)

Ideally, we want a activation function that has a limited range so large signals do not always dominate
Neural network: feed-forward

One commonly used function is the sigmoid:

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

The neural network is as good as it's structure and weights on edges

Structure we will ignore (more complex), but there is an automated way to learn weights

Whenever a NN incorrectly answer a problem, the weights play a “blame game”...
- Weights that have a big impact to the wrong answer are reduced
Back-propagation

To do this blaming, we have to find how much each weight influenced the final answer

Steps:
1. Find total error
2. Find derivative of error w.r.t. weights
3. Penalize each weight by an amount proportional to this derivative
Back-propagation

Consider this example: 4 nodes, 2 layers

This node as a constant bias of 1
Node 1: $0.15 \times 0.05 + 0.2 \times 0.1 + 0.35$ as input thus it outputs (all edges) $S(0.3775)=0.59327$
Eventually we get: \( \text{out}_1 = 0.7513, \text{out}_2 = 0.7729 \)

Suppose wanted: \( \text{out}_1 = 0.01, \text{out}_2 = 0.99 \)
Back-propagation

We will define the error as: 
\[ \sum_i (\text{correct}_i - \text{output}_i)^2 \]
(you will see why shortly)

Suppose we want to find how much \( w_5 \) is to blame for our incorrectness

We then need to find: 
\[ \frac{\partial \text{Error}}{\partial w_5} \]

Apply the chain rule:
\[ \frac{\partial \text{Error}}{\partial \text{out}_1} \cdot \frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} \cdot \frac{\partial \text{In}(N_3)}{\partial w_5} \]
**Back-propagation**

\[ \text{Error} = \sum_i (\text{correct}_i - \text{output}_i)^2 \]

\[
\frac{\partial \text{Error}}{\partial \text{out}_1} = - (\text{correct}_1 - \text{out}_1) \\
= - (0.01 - 0.7513) = 0.7413
\]

\[
\frac{\partial S(\text{In}(N_3))}{\partial \text{In}(N_3)} = S(\text{In}(N_3)) \cdot (1 - S(\text{In}(N_3))) \\
= 0.7513 \cdot (1 - 0.7513) = 0.1868
\]

\[
\frac{\partial \text{In}(N_3)}{\partial w_5} = \frac{\partial w_5 \cdot \text{Out}(N_1) + w_6 \cdot \text{Out}(N_2) + b_2 \cdot 1}{\partial w_5} \\
= \text{Out}(N_1) = 0.5932
\]

Thus, \[
\frac{\partial \text{Error}}{\partial w_5} = 0.7413 \cdot 0.1868 \cdot 0.5932 = 0.08217
\]
Back-propagation

In a picture we did this:

Now that we know $w_5$ is 0.08217 part responsible, we update the weight by:

$$w_5 \leftarrow w_5 - \alpha \times 0.08217 = 0.3589 \text{ (from 0.4)}$$

$\alpha$ is learning rate, set to 0.5
Back-propagation

Updating this $w_5$ to $w_8$ gives:

- $w_5 = 0.3589$
- $w_6 = 0.4067$
- $w_7 = 0.5113$
- $w_8 = 0.5614$

For other weights, you need to consider all possible ways in which they contribute.
Back-propagation

For $w_1$ it would look like:

\[
\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} \ast \frac{\partial \text{out}_{h1}}{\partial \text{net}_{h1}} \ast \frac{\partial \text{net}_{h1}}{\partial w_1}
\]

\[
\frac{\partial E_{\text{total}}}{\partial \text{out}_{h1}} = \frac{\partial E_{o1}}{\partial \text{out}_{h1}} + \frac{\partial E_{o2}}{\partial \text{out}_{h1}}
\]

$E_{\text{total}} = E_{o1} + E_{o2}$

(book describes how to dynamic program this)
Back-propagation

Specifically for $w_1$ you would get:

$$\frac{\partial Error}{\partial S(In(N_1))} = \frac{\partial Error_1}{\partial S(In(N_1))} + \frac{\partial Error_2}{\partial S(In(N_1))}$$

$$\frac{\partial S(In(N_1))}{\partial In(N_1)} = S(In(N_1)) \cdot (1 - S(In(N_1)))$$

$$= 0.5933 \cdot (1 - 0.5933) = 0.2413$$

$$\frac{\partial In(N_3)}{\partial w_5} = \frac{\partial w_1 \cdot In_1 + w_2 \cdot In_2 + b_1 \cdot 1}{\partial w_5}$$

$$= In_1 = 0.05$$

Next we have to break down the top equation...
Back-propagation

\[
\frac{\partial \text{Error}}{\partial S(In(N_1))} = \frac{\partial \text{Error}_1}{\partial S(In(N_1))} + \frac{\partial \text{Error}_2}{\partial S(In(N_1))}
\]

\[
\frac{\partial \text{Error}_1}{\partial S(In(N_1))} = \frac{\partial \text{Error}_1}{\partial S(In(N_3))} \cdot \frac{\partial S(In(N_3))}{\partial S(In(N_1))} \cdot \frac{\partial S(In(N_3))}{\partial S(In(N_1))}
\]

From before...

\[
\frac{\partial \text{Error}_1}{\partial S(In(N_3))} \cdot \frac{\partial S(In(N_3))}{\partial S(In(N_1))} = 0.7414 \cdot 0.1868 = 0.1385
\]

\[
\frac{\partial S(In(N_3))}{\partial S(In(N_1))} = \frac{\partial w_5 \cdot S(In(N_1)) + w_6 \cdot S(In(N_2)) + b_1 \cdot 1}{\partial S(In(N_1))}
\]

\[
= w_5 = 0.4
\]

Thus,

\[
\frac{\partial \text{Error}_1}{\partial S(In(N_1))} = 0.1385 \cdot 0.4 = 0.05540
\]
Back-propagation

Similarly for Error$_2$ we get:

$$\frac{\partial \text{Error}}{\partial S(In(N_1))} = \frac{\partial \text{Error}_1}{\partial S(In(N_1))} + \frac{\partial \text{Error}_2}{\partial S(In(N_1))}$$

$$= 0.05540 + -0.01905 = 0.03635$$

Thus, $$\frac{\partial \text{Error}}{\partial w_1} = 0.03635 \cdot 0.2413 \cdot 0.05 = 0.0004386$$

Update $$w_1 \leftarrow w_1 - \alpha \frac{\partial \text{Error}}{\partial w_1} = 0.15 - 0.5 \cdot 0.0004386 = 0.1498$$

You might notice this is small... This is an issue with neural networks, deeper the network the less earlier nodes update
Despite this learning shortcoming, NN are useful in a wide range of applications:

- Reading handwriting
- Playing games
- Face detection
- Economic predictions

Neural networks can also be very powerful when combined with other techniques (genetic algorithms, search techniques, ...).
NN examples

Examples:
https://www.youtube.com/watch?v=umRdt3zGgpU
https://www.youtube.com/watch?v=qv6UVOQ0F44
https://www.youtube.com/watch?v=xcIBoPuNliw
https://www.youtube.com/watch?v=0Str0Rdkxxo
https://www.youtube.com/watch?v=l2_CPB0uBkc
https://www.youtube.com/watch?v=0VTI1BBLydE
NN examples

AlphaGo/Zero has been in the news recently, and is also based on neural networks.

AlphaGo uses Monte-Carlo tree search guided by the neural network to prune useless parts.

Often limiting Monte-Carlo in a static way reduces the effectiveness, much like mid-state evaluations can limit algorithm effectiveness.
NN examples

Basically, AlphaGo uses a neural network to “prune” parts for a Monte-carlo search.