MACHINE LEARNING OUTLINE

JORDAN E DEHMEL

Pseudocode for finding the derivative of some node with respect to some weight w_k:

```
if w_k is in links:
    return links[w_k].previousActivation
else:
    out = 0
    for link in links:
        selfDer = derivative of this with respect to link
        out += selfDer * (derivative of link with respect to w_k)
    return out
```

Anatomy of an error node:

Method 1 (vector norm):

$$y = \sqrt{\sum (e_i - o_i)^2}$$

$$\frac{\partial y}{\partial w_k} = \frac{-\sum (e_i - o_i) \frac{\partial o_i}{\partial w_k}}{y}$$

Method 2 (easy derivative):

$$y = \sum (e_i - o_i)^2$$

$$\frac{\partial y}{\partial w_k} = -2 \sum (e_i - o_i) \frac{\partial o_i}{\partial w_k}$$

Anatomy of a regular node:

Sigmoid S:

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

$$S'(x) = S(x)(1 - S(x))$$

Sum f:

$$f(\vec{b}) = \sum v_i w_i + b$$

$$\frac{\partial f(\vec{b})}{\partial w_k} = \sum w_i \frac{\partial v_i}{\partial w_k}$$

(If w$_k iswithinitslistofweights, thederivativewillinsteadbeequaltov_i$)

Full equation:

$$y = S(f(\vec{b}))$$

$$\frac{\partial y}{\partial s} = S'(f(\vec{b}))(f'(\vec{b}))$$

With inputs v_n, weights w_i and bias b. Partial derivative taken with respect to later weight s.