

## Neural networks

Friday, 21 May 2021 04:35

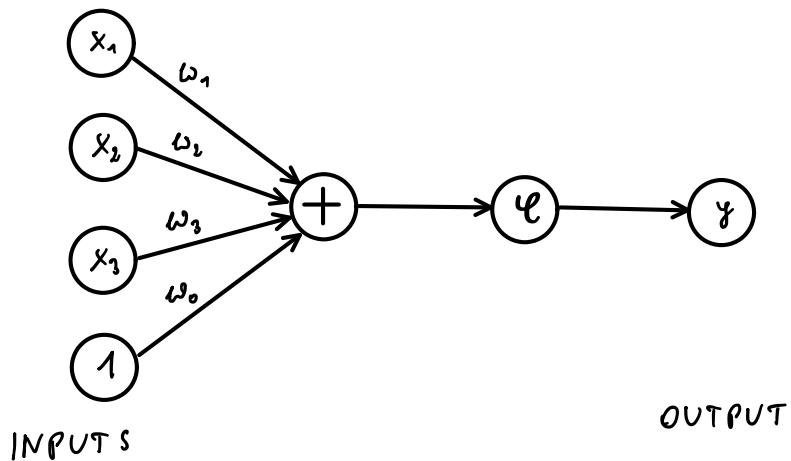
A neural network (NN), just as any other ML model, represents a function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ :

$$y = f(x|\theta)$$

dependent on some parameters  $\theta$ . It is also characterized by its typically large number of parameters and its unique training method - backpropagation.

### Neurons

A neuron has the form



and performs the following operation

$$y = \varphi \left( \sum_{k=1}^n w_k x_k + w_0 \right)$$

where  $\varphi$  is a non-decreasing differentiable function (except for possibly one point) and is called an activation function.

Examples of activation functions:

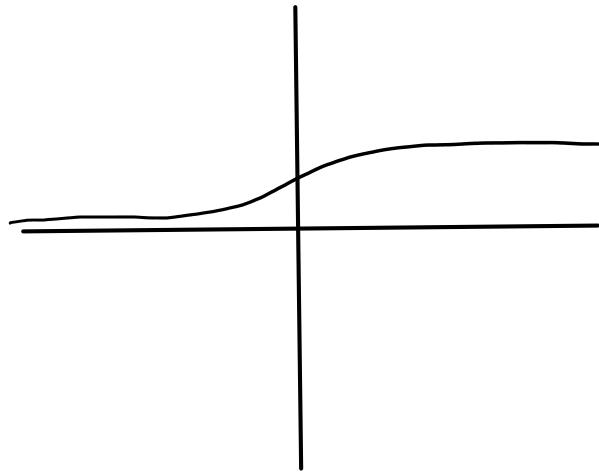
$$-\varphi(x) = ax + b \quad (\text{linear})$$

(with a linear activation function  
NN becomes Linear Regression)

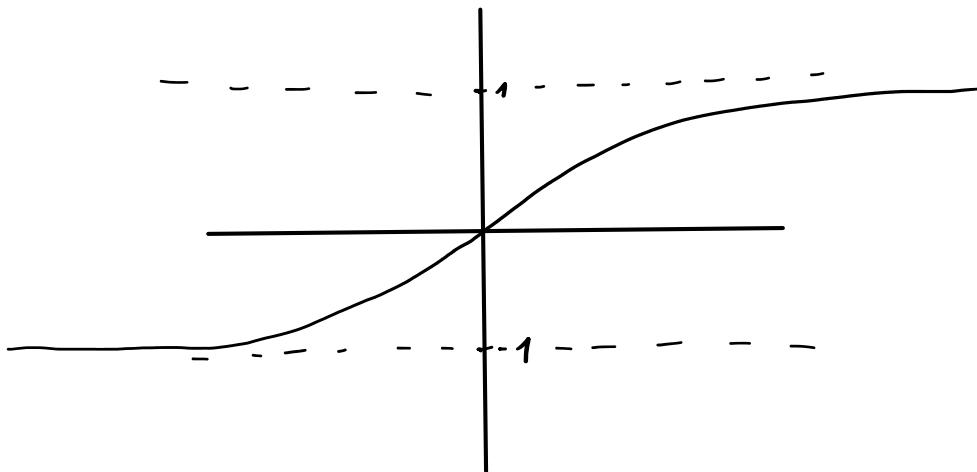
(favorite activation)

-  $\ell(x) = \frac{1}{1 + e^{-x}}$  (sigmoid or logistic curve)

(favorite activation function in the 80's and 90's)

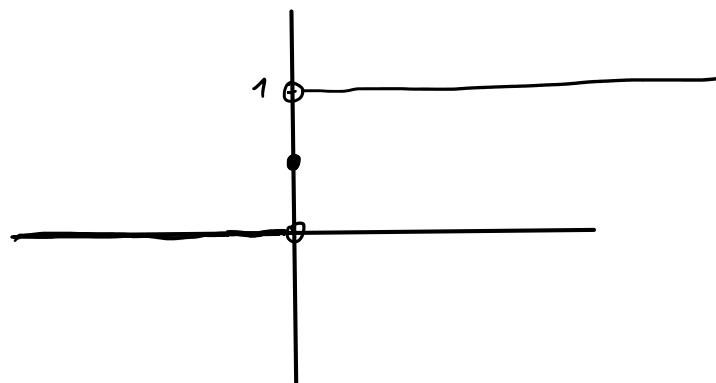


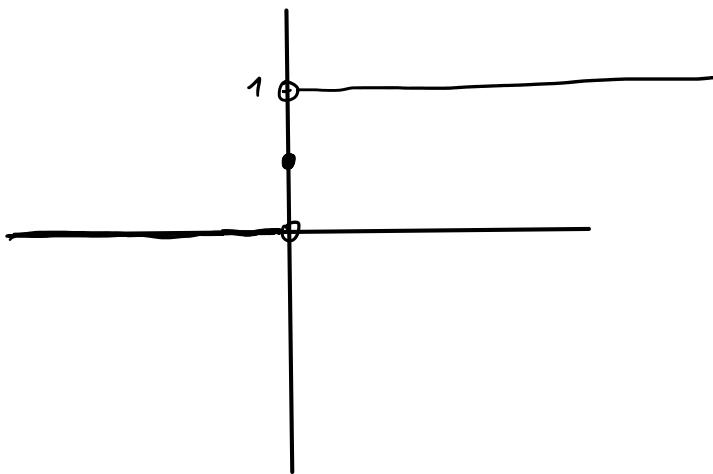
-  $\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$  (hyperbolic tangent)



-  $\varphi(x) = \begin{cases} 0 & : x < 0 \\ 0.5 & : x = 0 \\ 1 & : x > 0 \end{cases}$  (Heaviside function)

(this function is closest to how the brain neurons actually work, but was used only in the early days of neural networks as it makes training very hard)

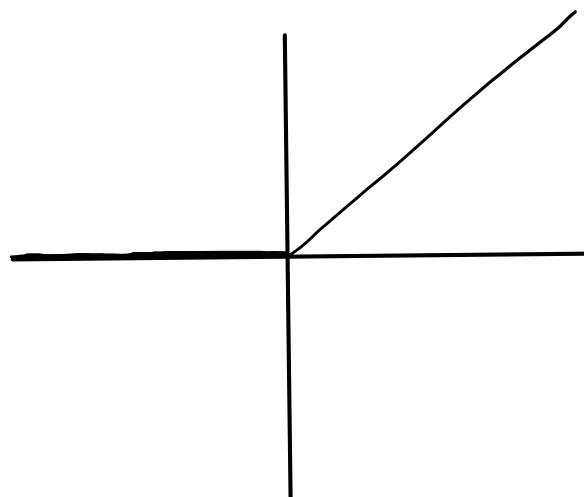




useful only on ...  
early days of  
neural networks  
as it makes  
training very  
hard )

$$- \varphi(x) = \begin{cases} x & : x > 0 \\ 0 & : x \leq 0 \end{cases} \quad (\text{Rectified Linear Unit - ReLu})$$

(this is the most  
commonly used  
activation function  
in the modern days)

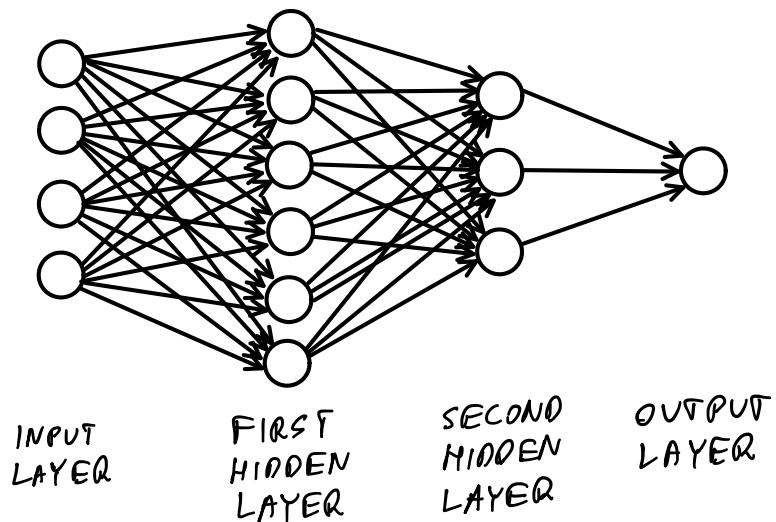


A neuron can be thought of as the most basic non-linear model built on top of a linear model.

A neuron is a primary building block of a neural network.

### Structure of a neural network

Every neural network consists of layers of neurons where the outputs of the previous layer form the input for the next layer.



### Fully connected layer

Layers with all possible connections to the previous layer are called **fully connected layers**.

A neural network which consists of fully connected layers is called **artificial neural network (ANN)**.

There exist other types of layers :

- dropout
- normalization (e.g. batch, spectral)
- convolution
- deconvolution
- recurrent
- embedding

There are also many neural network architectures.

The most common include :

- artificial neural networks (ANN)
- convolutional neural networks (CNN)
- recurrent neural networks (RNN)
- long short-term memory (LSTM)
- encoder - decoder
- generative adversarial networks (GAN)

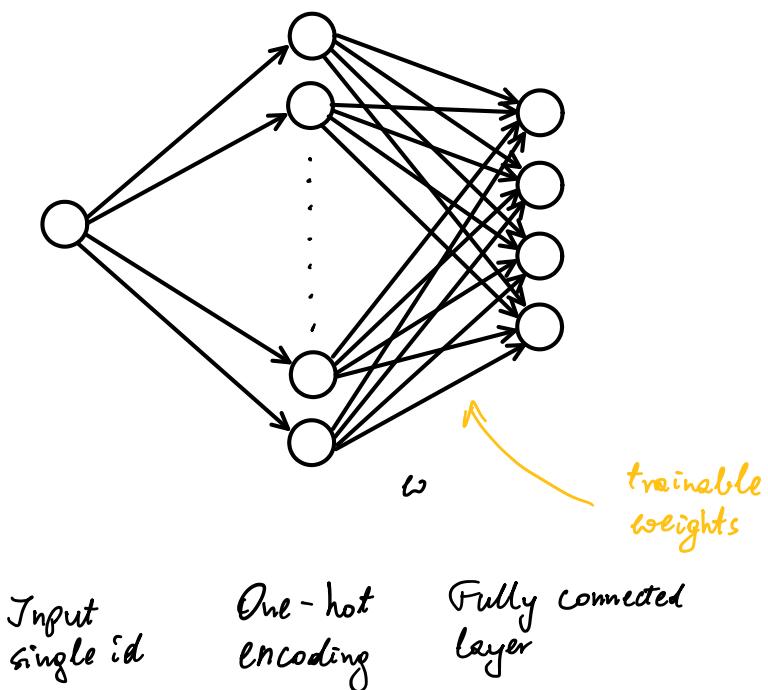
ANN with just one layer is called **perception**.

ANN with more than one layer is called  
**multi-layer perception (MLP)**

A neural network with many layers is called  
**deep neural network (DNN)**

In many real-world applications many types of layers and architectures are mixed into a larger network.

## Embedding layer



## Training a neural network

Learning the parameters of a neural network is always by minimizing a loss function.

The minimization process is typically done using backpropagation.

The two most common loss functions are:

- mean squared error (MSE) - for regression problems
- cross-entropy loss - for classification problems

- mean squared error (MSE) - for regression problems
- cross-entropy loss - for classification problems

### MSE

$$\text{loss}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x_n, y_n) \in \mathcal{D}} (y_n - f(x_n | \theta))^2$$

### Cross-entropy

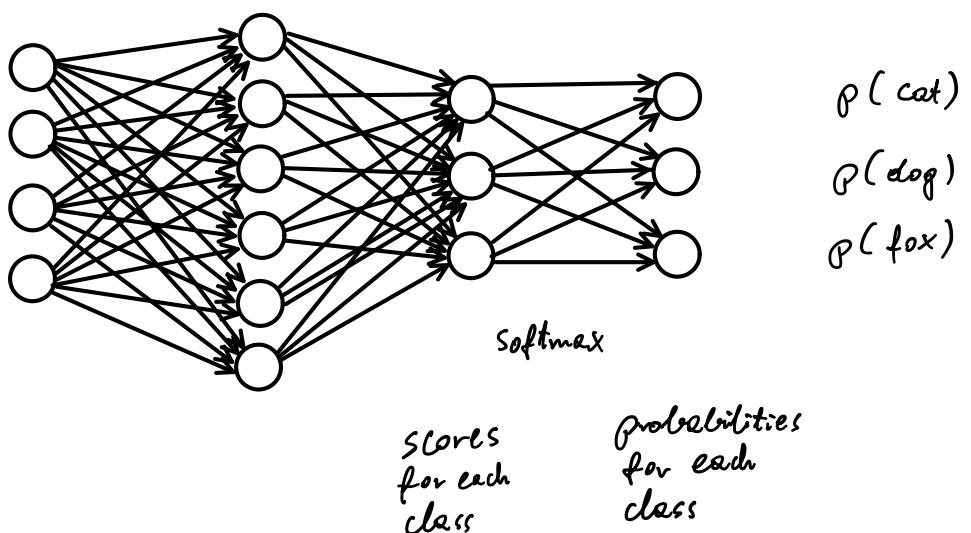
$$\text{loss}(\theta) = \sum_{(x_n, y_n) \in \mathcal{D}} \left( -\sum_c y_{nc} \ln(f_c(x_n | \theta)) \right) \quad y_{nc} \in \{0, 1\}$$

In the case of cross-entropy the neural network must return probabilities for all classes in the output layer.

Typically this is achieved by applying softmax to the final layer of a neural network:

$$p(c) = \frac{e^{f(c | \theta)}}{\sum_{c'} e^{f(c' | \theta)}}$$

where  $c$  is an index of a neuron in the final layer



## Backpropagation

Backpropagation relies on stochastic gradient descent to minimize the loss, which means in every step it shifts every network parameter in the direction opposite to the loss derivative with respect to that parameter, i.e.

$$\theta_i^{(n)} = \theta_i^{(n-1)} - \alpha \frac{\partial \text{loss}}{\partial \theta_i}$$

But the network parameters are in different layers and the formula for calculating the value of the network and then the loss can be very complex, hence also the derivative will be extremely complex.

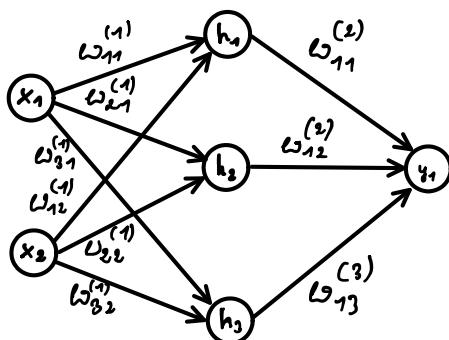
The clever idea in backpropagation is how to calculate those derivatives and parameter updates efficiently.

Consider the case of MSE loss

$$\text{loss}(\theta) = \sum_{(x_n, y_n) \in D} (y_n - f(x_n | \theta))^2$$

(We omit  $\frac{1}{101}$  as does not affect the optimization problem)

and a one hidden layer network



$$\Theta = (\omega_{11}^{(1)}, \omega_{12}^{(1)}, \omega_{21}^{(1)}, \omega_{22}^{(1)}, \omega_{31}^{(1)}, \omega_{32}^{(1)}, \omega_{11}^{(2)}, \omega_{12}^{(2)}, \omega_{13}^{(2)})$$

We can denote

$$\omega^{(1)} = \begin{bmatrix} \omega_{11}^{(1)} & \omega_{12}^{(1)} \\ \omega_{21}^{(1)} & \omega_{22}^{(1)} \\ \omega_{31}^{(1)} & \omega_{32}^{(1)} \end{bmatrix} \quad \omega^{(2)} = \begin{bmatrix} \omega_{11}^{(2)} \\ \omega_{12}^{(2)} \\ \omega_{13}^{(2)} \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \end{bmatrix}$$

Then

$$h = \varphi(\omega^{(1)} x)$$

$$y = \varphi(\omega^{(2)} h) = \underbrace{\varphi(\omega^{(2)} \varphi(\omega^{(1)} x))}_{\text{function composition}}$$

We can use the chain rule to calculate the derivative.

In the case of bigger networks the formula just contains more identical terms

$$y = \varphi(\omega^{(n)} \varphi(\omega^{(n-1)} \varphi(\dots \omega^{(2)} \varphi(\omega^{(1)} x) \dots))$$

We have

$$\frac{\partial \text{loss}(\theta)}{\partial \omega_{ij}^{(2)}} = \frac{\partial}{\partial \omega_{ij}^{(2)}} (y_n - f(x_n | \theta))^2$$

$$= 2(y_n - f(x_n | \theta)) \cdot \left(-\frac{\partial}{\partial \omega_{ij}^{(2)}} f(x_n | \theta)\right)$$

$$= -2 e \frac{\partial}{\partial \omega_{ij}^{(2)}} \varphi(\omega^{(2)} h)$$

$$e = (y_n - f(x_n | \theta))$$

$$\begin{aligned}
 &= -2 e \frac{\partial}{\partial \omega_{ij}^{(2)}} \varphi(\omega^{(2)} h) \quad e = (\gamma_n - f(x_n | \theta)) \\
 &= -2 e \frac{\partial}{\partial \omega_{ij}^{(2)}} \varphi(\omega_{11}^{(2)} h_1 + \omega_{12}^{(2)} h_2 + \omega_{13}^{(2)} h_3) \\
 &= -2 e \varphi'(\omega^{(2)} h) \cdot \frac{\partial}{\partial \omega_{ij}^{(2)}} (\omega_{11}^{(2)} h_1 + \omega_{12}^{(2)} h_2 + \omega_{13}^{(2)} h_3)
 \end{aligned}$$

Consider  $i=1, j=1$ . Then

$$\begin{aligned}
 \frac{\partial \text{loss}(\theta)}{\partial \omega_{11}^{(2)}} &= -2 e \varphi'(\omega^{(2)} h) \cdot \frac{\partial}{\partial \omega_{11}^{(2)}} (\omega_{11}^{(2)} h_1 + \omega_{12}^{(2)} h_2 + \omega_{13}^{(2)} h_3) \\
 &= -2 e \varphi'(\omega^{(2)} h) \cdot h_1
 \end{aligned}$$

Therefore

$$\begin{bmatrix} \Delta \omega_{11}^{(2)} \\ \Delta \omega_{12}^{(2)} \\ \Delta \omega_{13}^{(2)} \end{bmatrix} = -2 \begin{bmatrix} \frac{\partial \text{loss}(\theta)}{\partial \omega_{11}^{(2)}} \\ \frac{\partial \text{loss}(\theta)}{\partial \omega_{12}^{(2)}} \\ \frac{\partial \text{loss}(\theta)}{\partial \omega_{13}^{(2)}} \end{bmatrix} = 2 \cancel{2} e \underbrace{\varphi'(\omega^{(2)} h)}_{\substack{\text{single} \\ \text{value}}} \cdot \underbrace{\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}}_{\substack{\text{single} \\ \text{value} \\ 3 \times 1}}$$

For the sigmoid function

$$\varphi'(x) = \varphi(x)(1 - \varphi(x))$$

hence

$$\varphi'(\omega^{(2)} h) = \underbrace{\varphi(\omega^{(2)} h)}_{\substack{\text{these are the outputs} \\ \text{from the first layer}}} (1 - \underbrace{\varphi(\omega^{(2)} h)}_{\substack{\text{these are the outputs} \\ \text{from the first layer}}})$$

Therefore for the second layer

for the second layer

'I therefore for the second layer

$$\begin{bmatrix} \Delta \omega_{11}^{(2)} \\ \Delta \omega_{12}^{(2)} \\ \Delta \omega_{13}^{(2)} \end{bmatrix} = \lambda \nabla [\text{error-term}] \begin{bmatrix} \text{gradient in} \\ \text{this layer result} \end{bmatrix} \begin{bmatrix} \text{previous} \\ \text{layer} \\ \text{result} \end{bmatrix}$$

for the second layer  
these are just  $1 \times 1$  matrices

In other words we propagate the error term backwards into the network with weights derived from the previous layer and this layer results

Symbolically it could be expressed as

$$\Delta \omega = f^{-1}(\text{error-term} \mid \psi(x \mid \theta))$$

To calculate the first layer parameter updates compare

$$y = \psi(\omega^{(2)} h)$$

$$y = \psi(\omega^{(2)} \psi(\omega^{(1)} x))$$

For the second layer we had

$$\frac{\partial \text{loss}(\theta)}{\partial \omega_{ij}^{(2)}} = -\lambda e \frac{\partial}{\partial \omega_{ij}^{(2)}} \psi(\omega^{(2)} h)$$

Hence for the first layer

$$\frac{\partial \text{loss}(\theta)}{\partial \omega_{ij}^{(1)}} = -\lambda e \frac{\partial}{\partial \omega_{ij}^{(1)}} \psi(\omega^{(2)} \psi(\omega^{(1)} x))$$

We have

$$\frac{\partial}{\partial \omega_{ij}^{(1)}} \psi(\omega^{(2)} \psi(\omega^{(1)} x)) = \underbrace{\psi'(\omega^{(2)} \psi(\omega^{(1)} x))}_{\rightarrow} \frac{\partial}{\partial \omega_{ij}^{(1)}} (\omega^{(2)} \psi(\omega^{(1)} x))$$

$$\frac{\partial \ell}{\partial w_{ij}^{(1)}} \varphi(\omega^{(2)} \varphi(\omega^{(1)} x)) = \varphi'(\omega^{(2)} \varphi(\omega^{(1)} x)) \frac{\partial \varphi}{\partial w_{ij}^{(1)}}$$

This expression is equal to  $\varphi'(\omega^{(2)} h)$  and we already calculated it for updates in the second layer

Again we can "propagate" information backward

Then

$$\frac{\partial}{\partial w_{ij}^{(1)}} (\omega^{(2)} \varphi(\omega^{(1)} x)) = \frac{\partial}{\partial w_{ij}^{(1)}} \left[ \begin{bmatrix} \omega_{11}^{(2)}, \omega_{12}^{(2)}, \omega_{13}^{(2)} \end{bmatrix} \varphi \left( \begin{bmatrix} \omega_{11}^{(1)}, \omega_{12}^{(1)}, \omega_{13}^{(1)} \\ \omega_{21}^{(1)}, \omega_{22}^{(1)}, \omega_{23}^{(1)} \\ \omega_{31}^{(1)}, \omega_{32}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right) \right]$$

$$= \frac{\partial}{\partial w_{ij}^{(1)}} \left[ \begin{bmatrix} \omega_{11}^{(2)}, \omega_{12}^{(2)}, \omega_{13}^{(2)} \end{bmatrix} \begin{bmatrix} \varphi(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2) \\ \varphi(\omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2) \\ \varphi(\omega_{31}^{(1)} x_1 + \omega_{32}^{(1)} x_2) \end{bmatrix} \right]$$

$$= \frac{\partial}{\partial w_{ij}^{(1)}} \left( \omega_{11}^{(2)} \varphi(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2) + \omega_{12}^{(2)} \varphi(\omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2) + \omega_{13}^{(2)} \varphi(\omega_{31}^{(1)} x_1 + \omega_{32}^{(1)} x_2) \right)$$

For  $i=1, j=1$  we have

$$\frac{\partial}{\partial w_{11}^{(1)}} (\omega^{(2)} \varphi(\omega^{(1)} x))$$

$$= \frac{\partial}{\partial w_{11}^{(1)}} \left( \omega_{11}^{(2)} \varphi(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2) + \omega_{12}^{(2)} \varphi(\omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2) + \omega_{13}^{(2)} \varphi(\omega_{31}^{(1)} x_1 + \omega_{32}^{(1)} x_2) \right)$$

$$= \omega_{11}^{(2)} \varphi'(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2) \cdot \frac{\partial}{\partial w_{11}^{(1)}} (\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2)$$

$$= \omega_{11}^{(2)} \varphi'(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2) x_1$$

Hence

$$\frac{\partial}{\partial \omega_{ij}^{(1)}} \varphi(\omega^{(2)} \varphi(\omega^{(1)} x)) = \varphi'(\omega^{(2)} h) \omega_{ij}^{(2)} \varphi'(h_i) x_j$$

and

$$\frac{\partial \text{loss}(\theta)}{\partial \omega_{ij}^{(1)}} = -2 e \varphi'(\omega^{(2)} h) \omega_{ij}^{(2)} \varphi'(h_i) x_j$$

Therefore

$$\begin{bmatrix} \Delta \omega_{11}^{(1)} & \Delta \omega_{12}^{(1)} \\ \Delta \omega_{21}^{(1)} & \Delta \omega_{22}^{(1)} \\ \Delta \omega_{31}^{(1)} & \Delta \omega_{32}^{(1)} \end{bmatrix} = -\lambda \begin{bmatrix} \frac{\partial \text{loss}(\theta)}{\partial \omega_{11}^{(1)}} & \frac{\partial \text{loss}(\theta)}{\partial \omega_{12}^{(1)}} \\ \frac{\partial \text{loss}(\theta)}{\partial \omega_{21}^{(1)}} & \frac{\partial \text{loss}(\theta)}{\partial \omega_{22}^{(1)}} \\ \frac{\partial \text{loss}(\theta)}{\partial \omega_{31}^{(1)}} & \frac{\partial \text{loss}(\theta)}{\partial \omega_{32}^{(1)}} \end{bmatrix} = -2 \lambda e \underbrace{\varphi'(\omega^{(2)} h) \cdot \omega^{(2)} \circ \varphi'(h)}_{\substack{\text{single value} \\ \text{3x1}}} \underbrace{x^T}_{\substack{\text{1x2} \\ \text{3x2}}}$$

Again we just propagate the error term  $e$  backwards into the network. We also "propagate" the gradient from the next layer to the previous layer.

Symbolically

$$\begin{bmatrix} \Delta \omega_{11}^{(1)} & \Delta \omega_{12}^{(1)} \\ \Delta \omega_{21}^{(1)} & \Delta \omega_{22}^{(1)} \\ \Delta \omega_{31}^{(1)} & \Delta \omega_{32}^{(1)} \end{bmatrix} = -2 \lambda \left[ \text{error-term} \right] \left[ \begin{array}{c} \text{gradient on the next layer result} \\ \text{gradient in this layer result} \end{array} \right] \left[ \begin{array}{c} \text{weighted} \\ \text{in this layer result} \end{array} \right] \left[ \begin{array}{c} \text{previous} \\ \text{layer result} \end{array} \right]$$

All the necessary values for backpropagation can be calculated in the forward network run except for the error term. But the error term is the same for each weight update, therefore all weight updates can be calculated simultaneously.