Backpropagation via Computation Graphs

A Complete Worked Example for a 2-Layer Neural Network

1. Overview

This document walks through backpropagation for a simple 2-layer neural network (2 inputs, 2 hidden units, 1 output). We treat the network as a **computation graph** where values flow forward and gradients flow backward.

2. The Computation Graph

The network can be visualised as a chain of operations:

```
x --> z1 --> a1 --> z2 --> a2 --> L
      (W1)   (s)    (W2)   (s)   (loss)
```

More explicitly, in vector form:

```
x --> z1 = W1*x + b1 --> a1 = sigma(z1)
                                  |
                                  v
                     z2 = W2*a1 + b2 --> a2 = sigma(z2) --> L
```

3. Forward Pass: Values Flow Left to Right

During the forward pass, we compute and **store** all intermediate values. These cached values will be needed during backpropagation.

Step-by-step computations:

Step	Node	Computation
1	Linear (hidden)	z1 = W1*x + b1
2	Activation (hidden)	a1 = sigma(z1)
3	Linear (output)	z2 = W2*a1 + b2
4	Activation (output)	a2 = sigma(z2)
5	Loss	L = (1/2)*(a2 - y)^2

4. Backward Pass: Gradients Flow Right to Left

Each node asks: *"How does the loss change if I change slightly?"* We apply the chain rule at each node, propagating gradients backward.

4.1 Loss to Output Activation

Starting at the loss node:

```
dL/da2 = a2 - y
```

4.2 Output Activation to Output Pre-activation

Since a2 = sigma(z2), by the chain rule:

```
da2/dz2 = sigma'(z2)
```

Combining these gives the **output error signal**:

```
delta2 = dL/dz2 = (dL/da2) * (da2/dz2) = (a2 - y) * sigma'(z2)
```

4.3 Output Pre-activation to Weights, Bias, and Hidden Activations

Since z2 = W2*a1 + b2, the gradient branches to three destinations:

Destination	Gradient Formula
Weight W2	dL/dW2 = delta2 * a1^T
Bias b2	dL/db2 = delta2
Hidden activation a1	dL/da1 = W2^T * delta2

4.4 Hidden Activation to Hidden Pre-activation

Since a1 = sigma(z1):

```
da1/dz1 = sigma'(z1)
```

Applying the chain rule node-by-node gives the **hidden error signal**:

```
delta1 = dL/dz1 = (W2^T * delta2) [elementwise*] sigma'(z1)
```

This is where delta1 comes from: it is the gradient at the z1 node, computed by propagating delta2 backward through the weight matrix and element-wise multiplying by the local derivative of the activation.

4.5 Hidden Pre-activation to First-Layer Parameters

Since z1 = W1*x + b1:

Parameter	Gradient Formula
Weight W1	dL/dW1 = delta1 * x^T

Bias b1	dL/db1 = delta1

5. Summary of Error Signals

Symbol	Node	Formula	Meaning
delta2	z2	(a2 - y) * sigma'(z2)	Output layer error
delta1	z1	(W2^T * delta2) [elem*] sigma'(z1)	Hidden layer error

6. Key Insights

delta1 is not arbitrary

It is precisely dL/dz1 - the gradient of the loss with respect to the hidden layer's pre-activation. It arises naturally from traversing the computation graph backward.

Backprop is reverse graph traversal

The forward pass computes values; the backward pass computes derivatives. Same graph, opposite direction.

Every delta corresponds to a node

Each error signal delta_k represents the gradient at a specific pre-activation node z_k. This is why error signals propagate - they follow the graph structure.

> *"Backpropagation is just the chain rule applied to a computation graph, one node at a time, from right to left."*