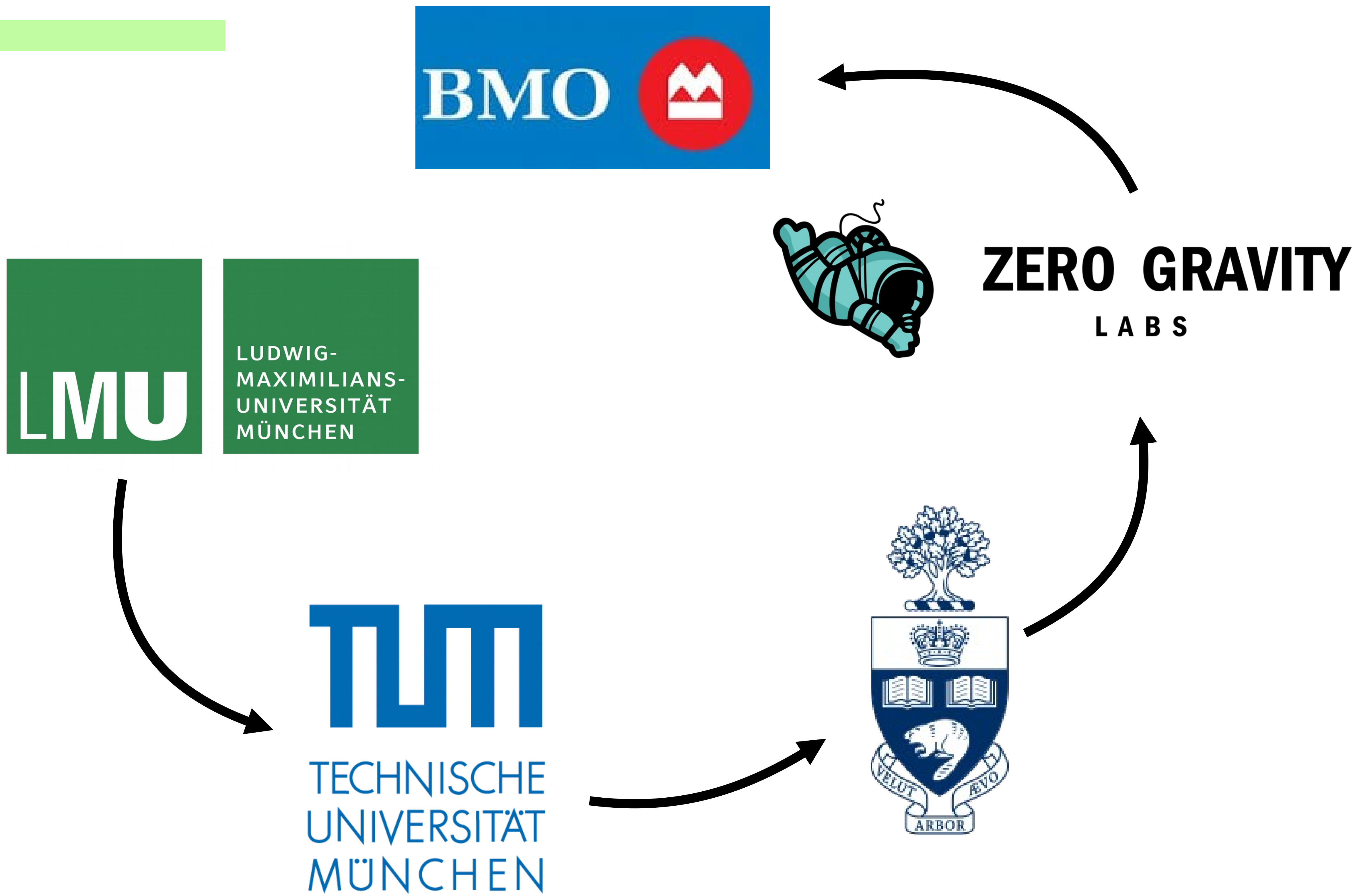
Learning representations by back-propagating errors

- Florian Goebels

December 12, 2018

florian.goebels@gmail.com

About me


BMO
LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN
LMU
TECHNISCHE
UNIVERSITÄT
MÜNCHEN
VELUT
ARBOR
ÆVO
ZERO GRAVITY
LABS

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts

Step—by—Step guide to back prop

Some terminology

- **X input matrix**
- **W hidden weight matrix**
- **Z hidden state**
- **A activations**

σ **Activation functio**

Perceptron

$$F_l(x) = \sigma(xW^l)$$

Multi-layer perceptron

$$\boxed{X} \xrightarrow{F_1} \boxed{a^1} \xrightarrow{F_2} \boxed{a^2}$$

Why activation function

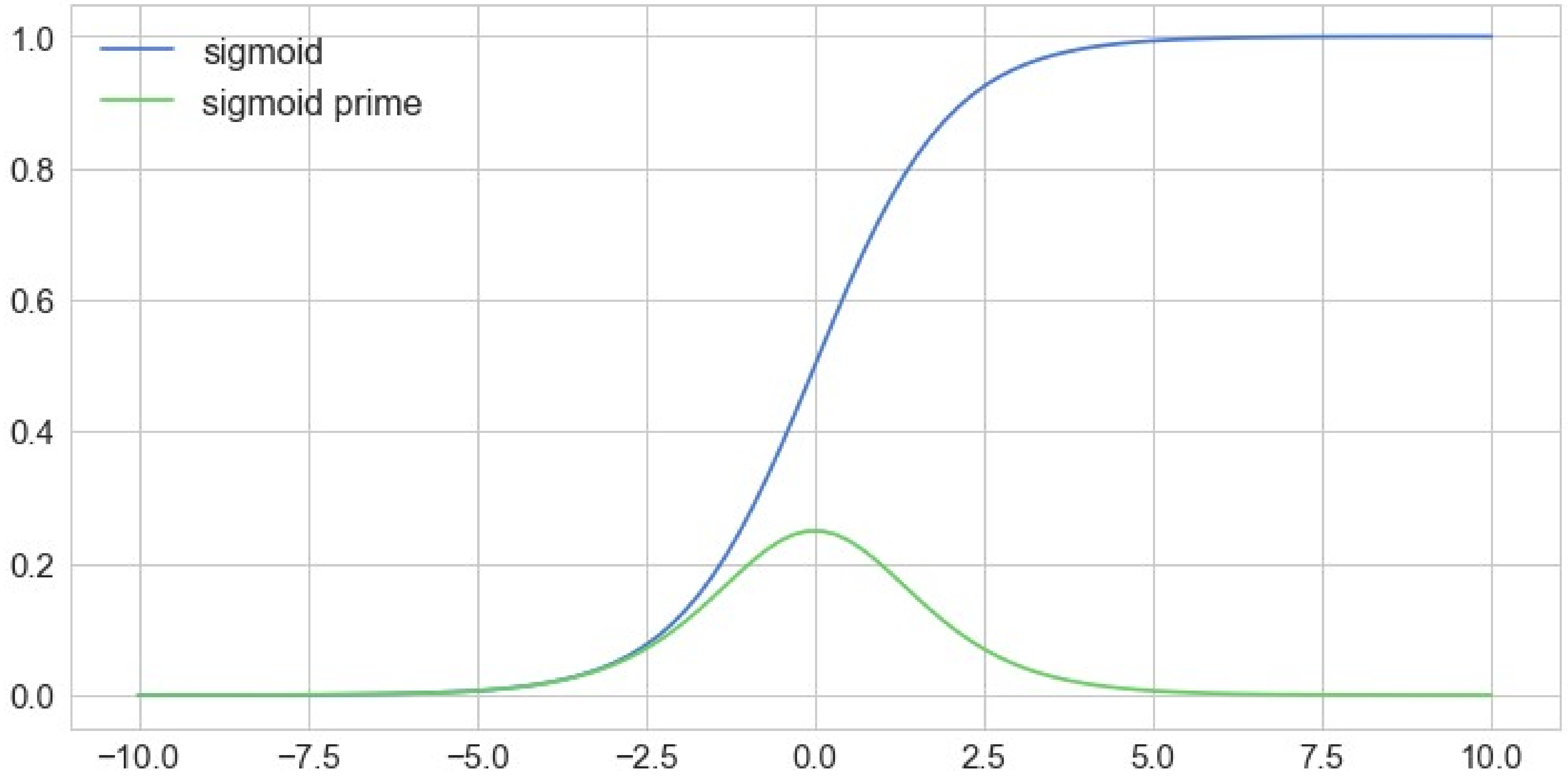
1. **Capture non-linear relationships**

2. **Easy to calculate**

3. **Easy derivatives**

Sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$



Forward propagation

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \tag{1}$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

$$y_j = \frac{1}{1+e^{-x_j}} \tag{2}$$

Neural network

Name	Days raining	Kdp	P(Rain)
Mon	3	5	0.99
Tue	1	6	0.59
Wend	0	8	0.19
	Features/Data		**Target**

Neural network

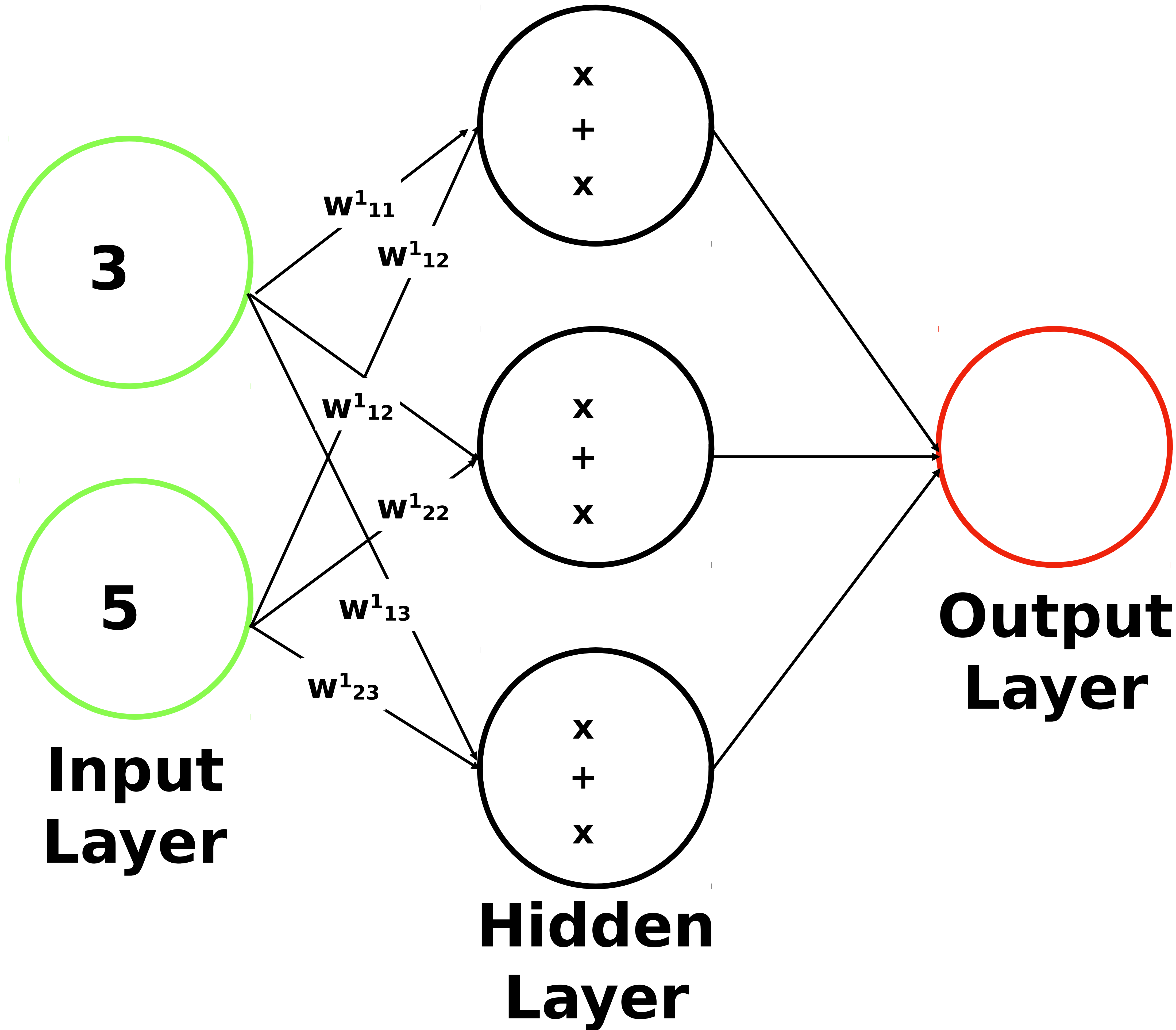
Days raining	Kdp
3	5
1	6
0	8

Neural network

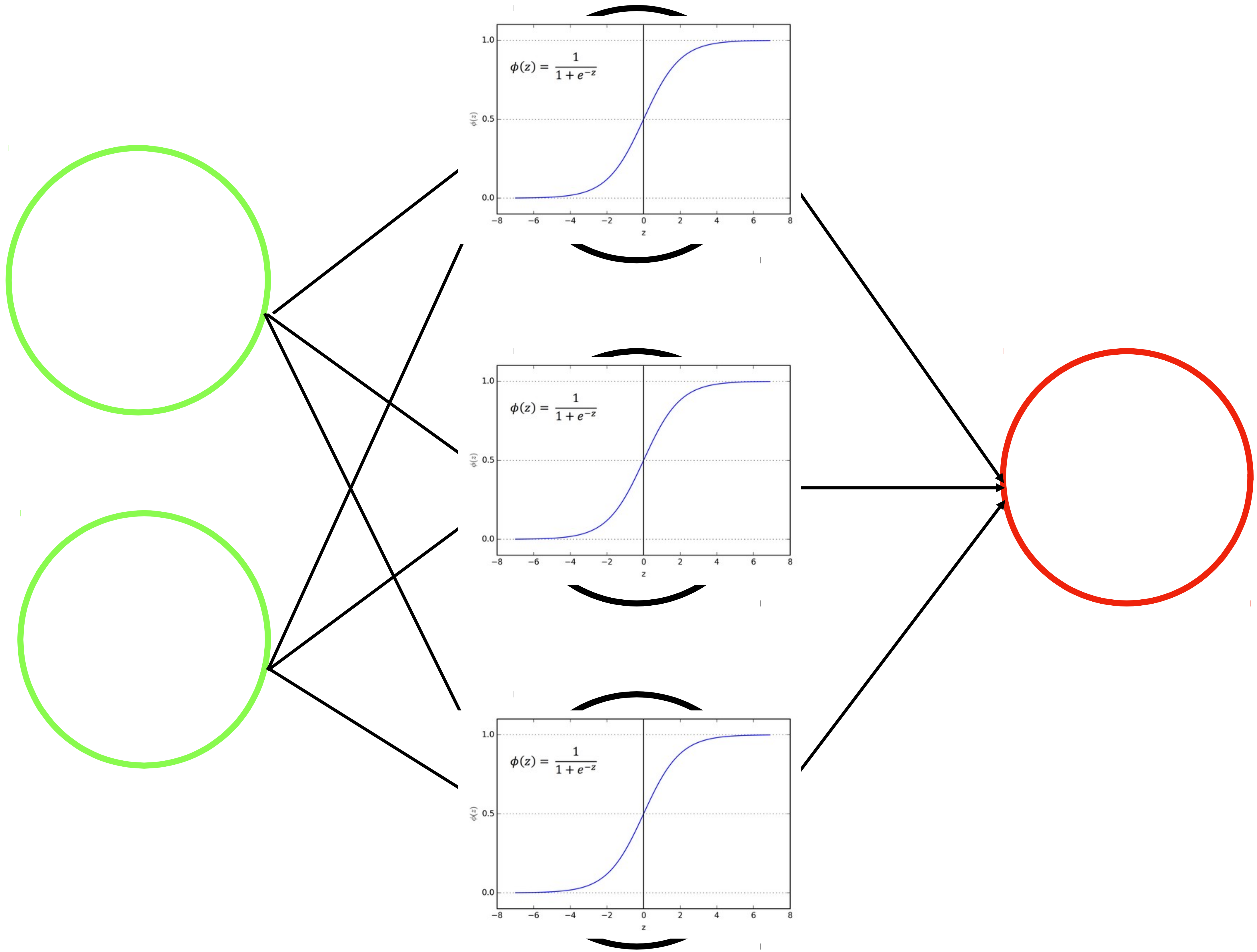
Days raining	Kdp
3	5
1	6
0	8

$$\begin{bmatrix} 3 & 5 \\ 1 & 6 \\ 0 & 8 \end{bmatrix}$$

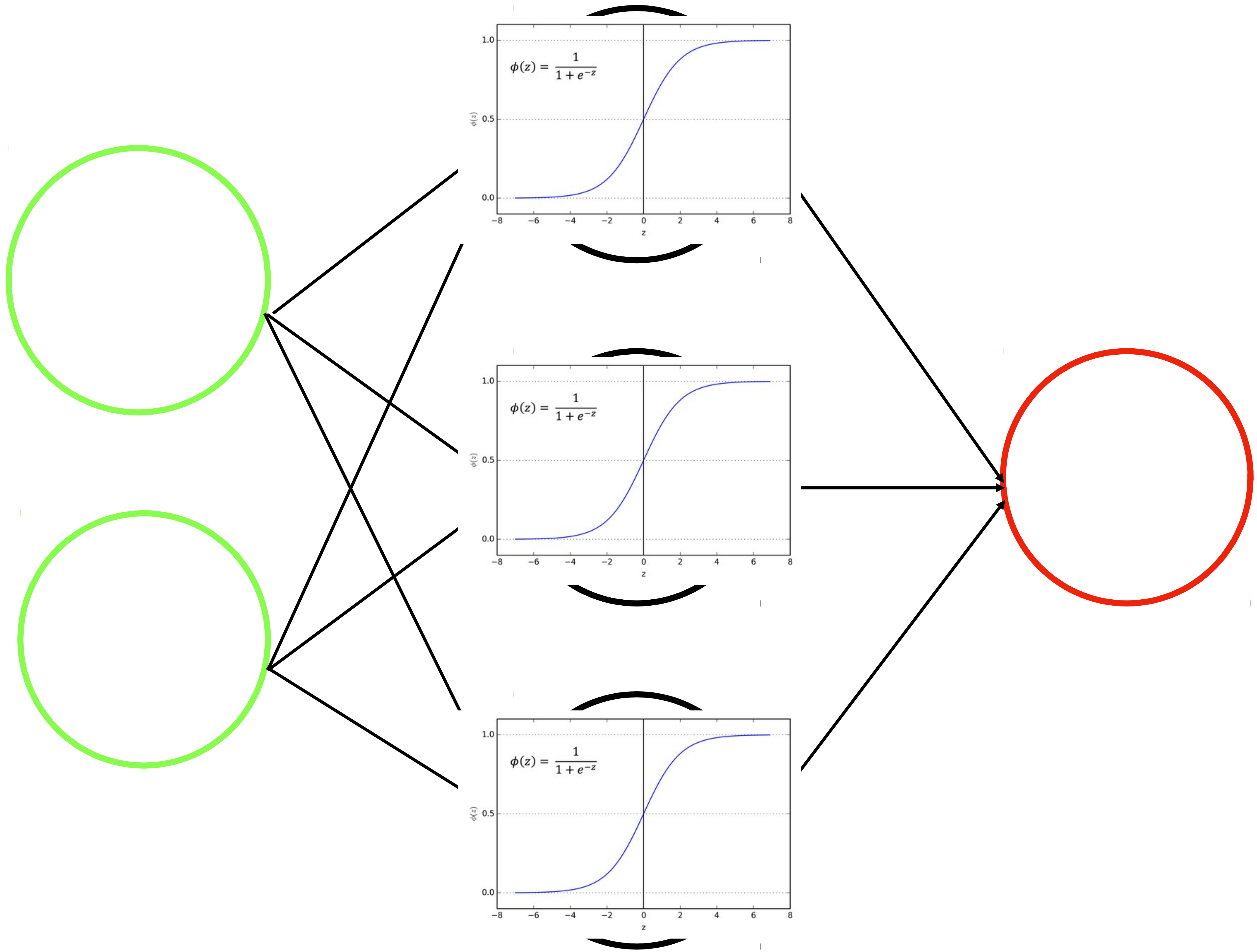
$$\begin{bmatrix} 3 & 5 \\ 1 & 6 \\ 0 & 8 \end{bmatrix}$$



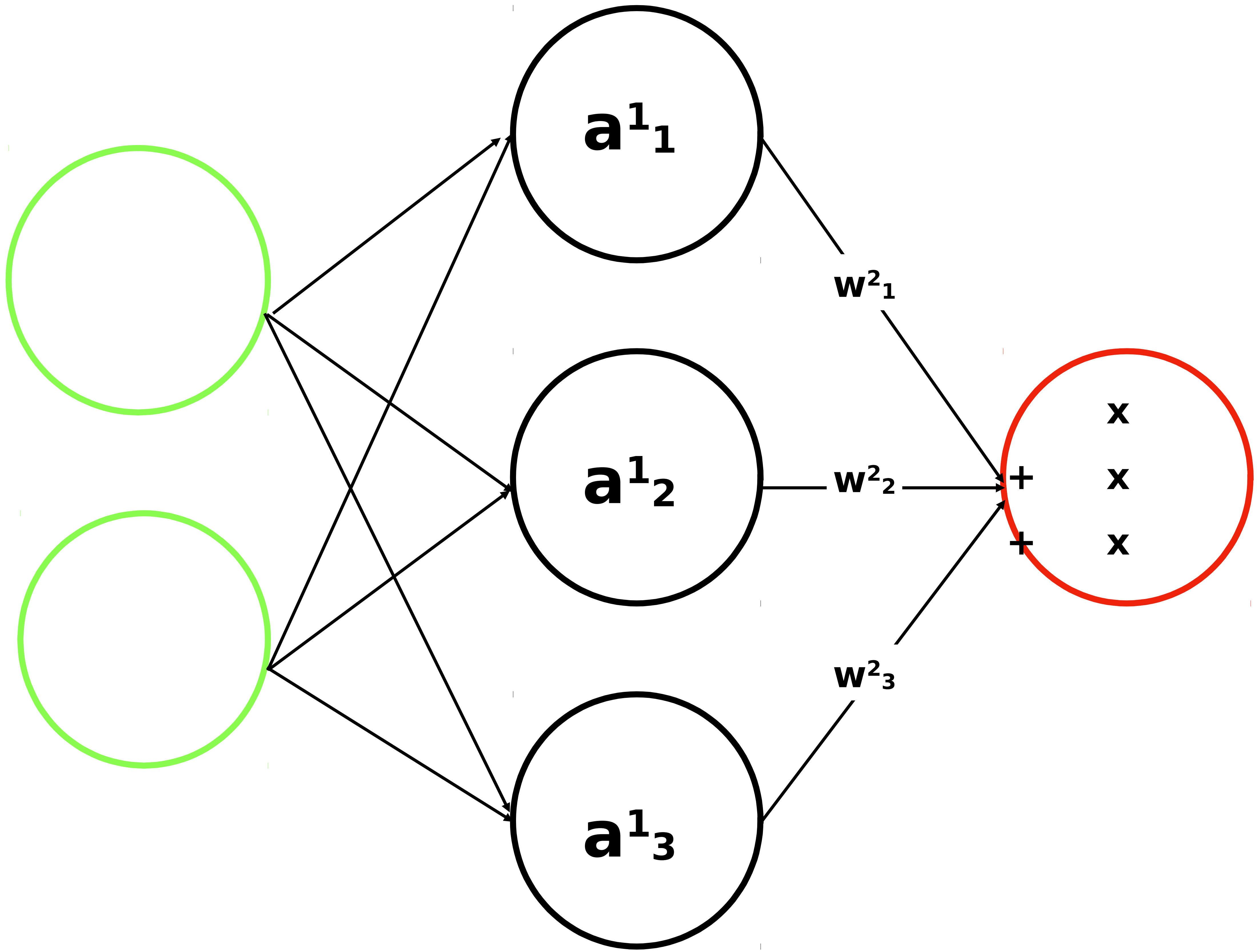
$$\begin{bmatrix} 1 & 6 \\ 0 & 8 \end{bmatrix}$$



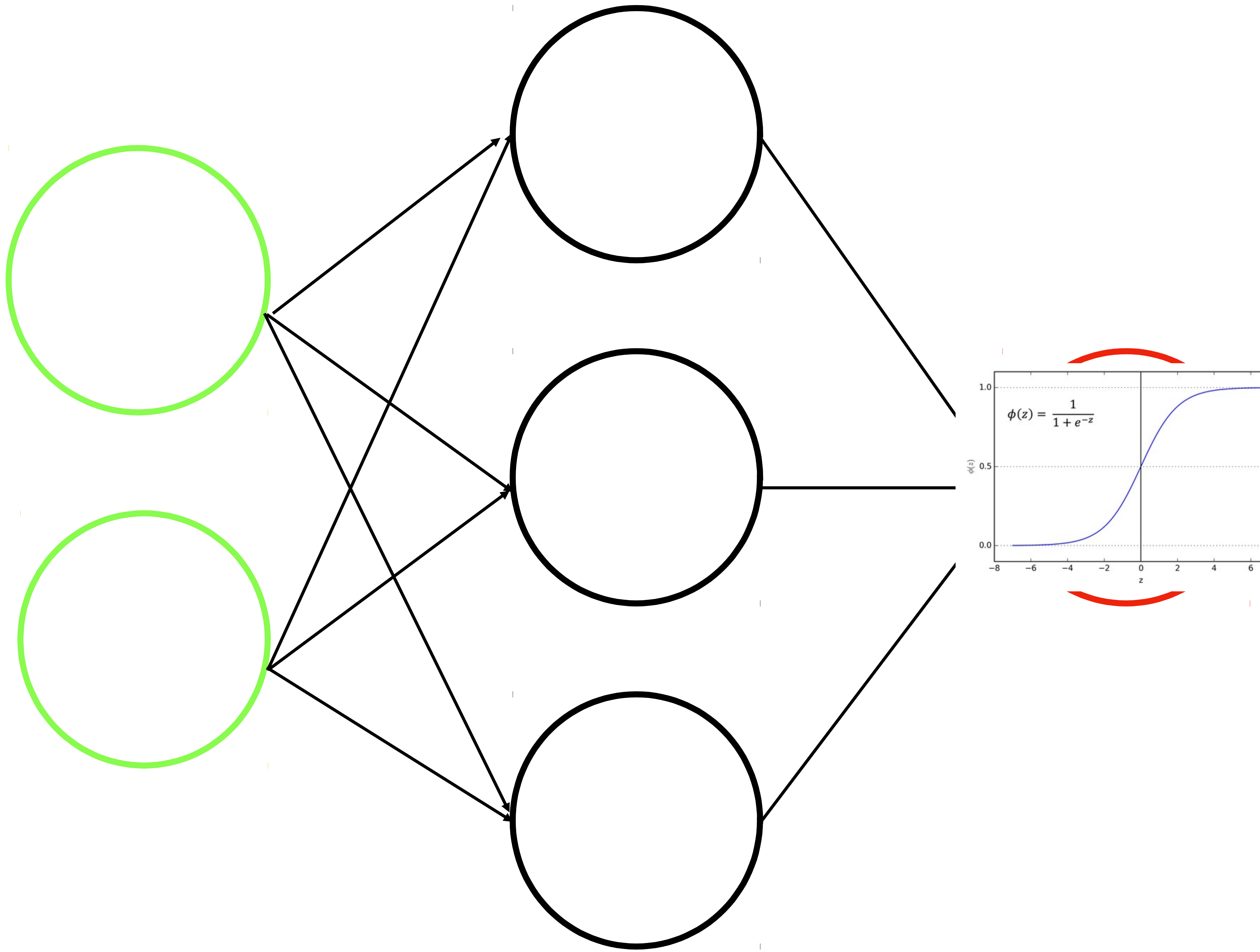
$$\begin{bmatrix} 1 & 6 \\ 0 & 8 \end{bmatrix}$$



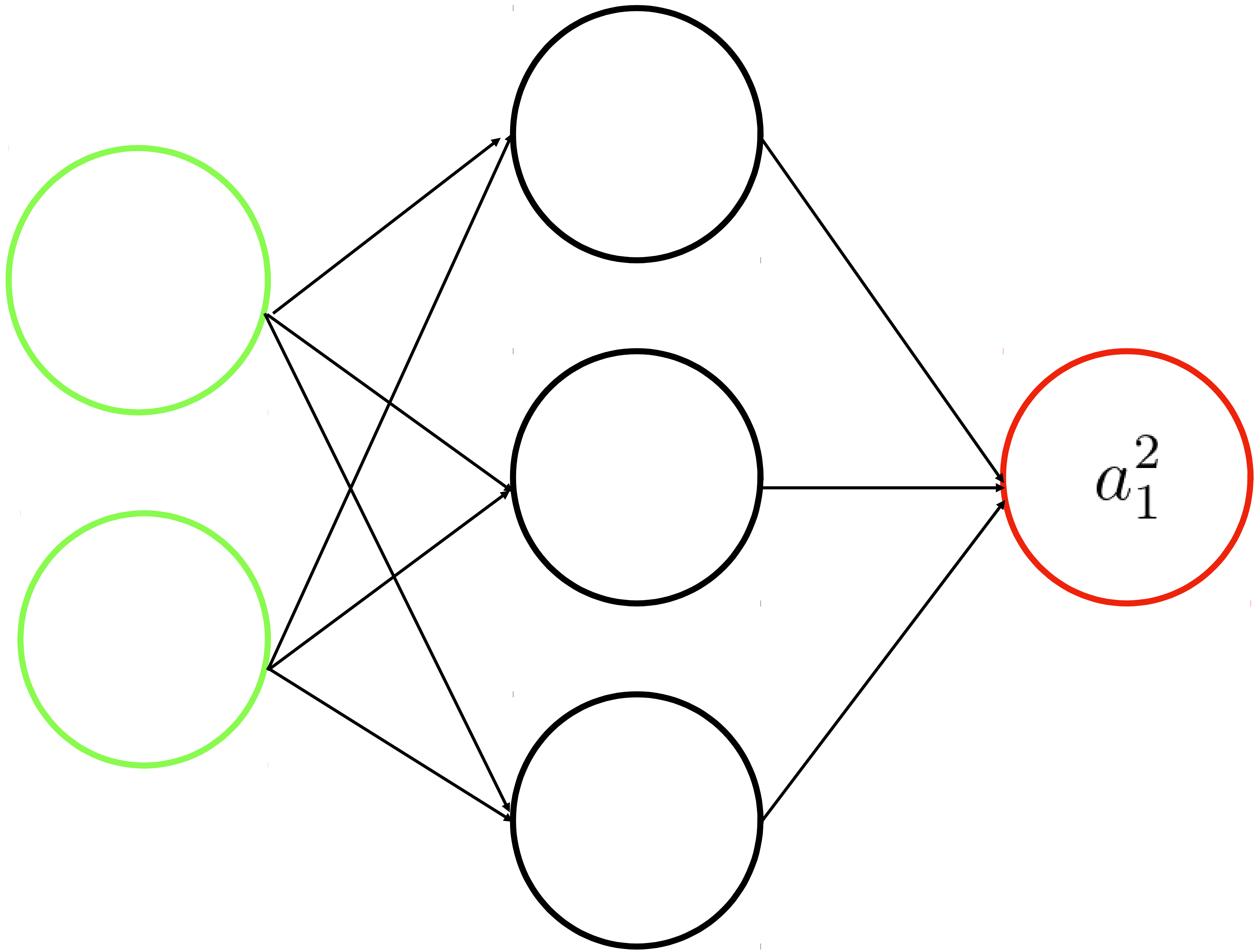
$$\begin{bmatrix} 1 & 6 \\ 0 & 8 \end{bmatrix}$$

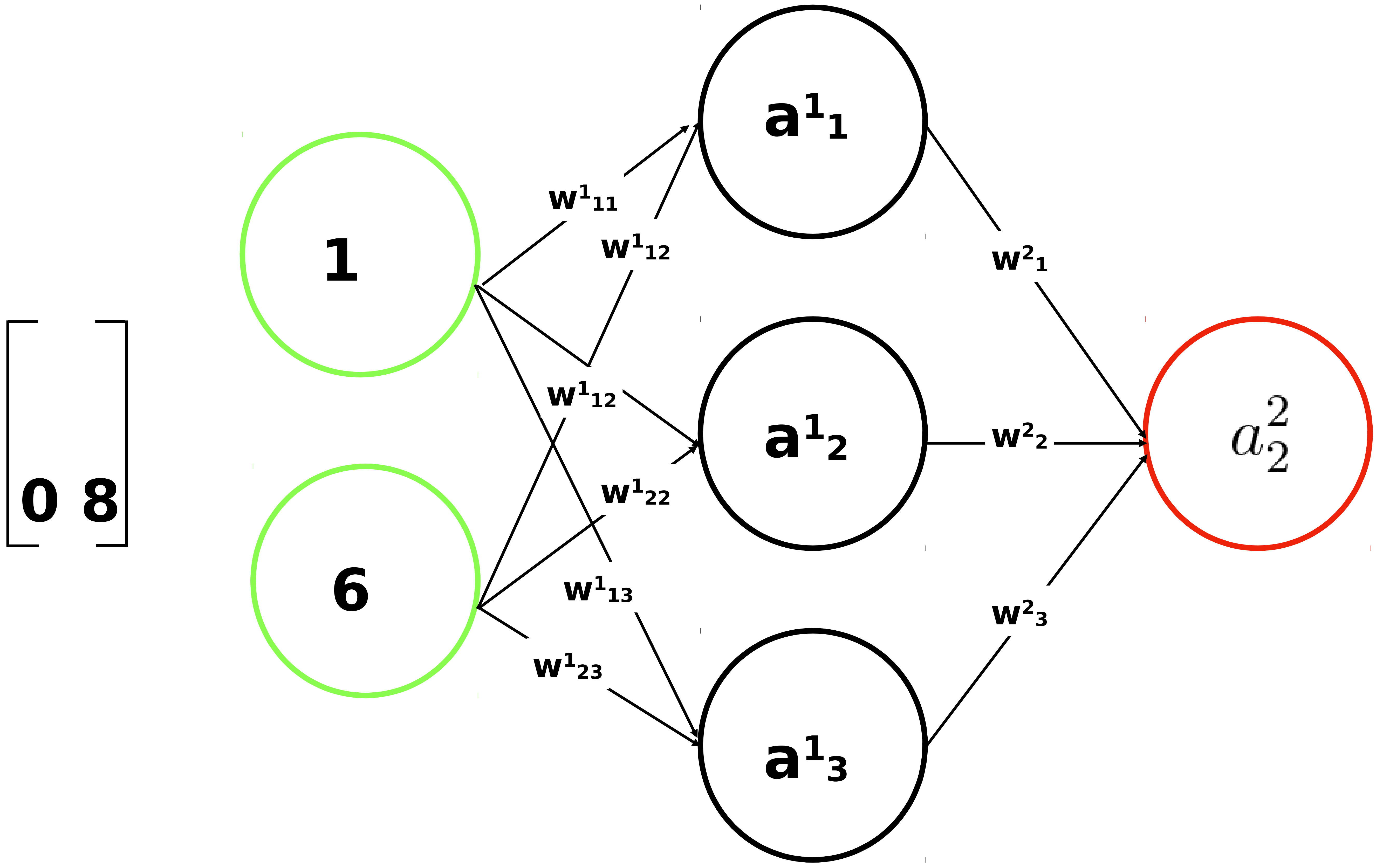


$$\begin{bmatrix} \mathbf{1} & \mathbf{6} \\ \mathbf{0} & \mathbf{8} \end{bmatrix}$$



$$\begin{bmatrix} 1 & 6 \\ 0 & 8 \end{bmatrix}$$

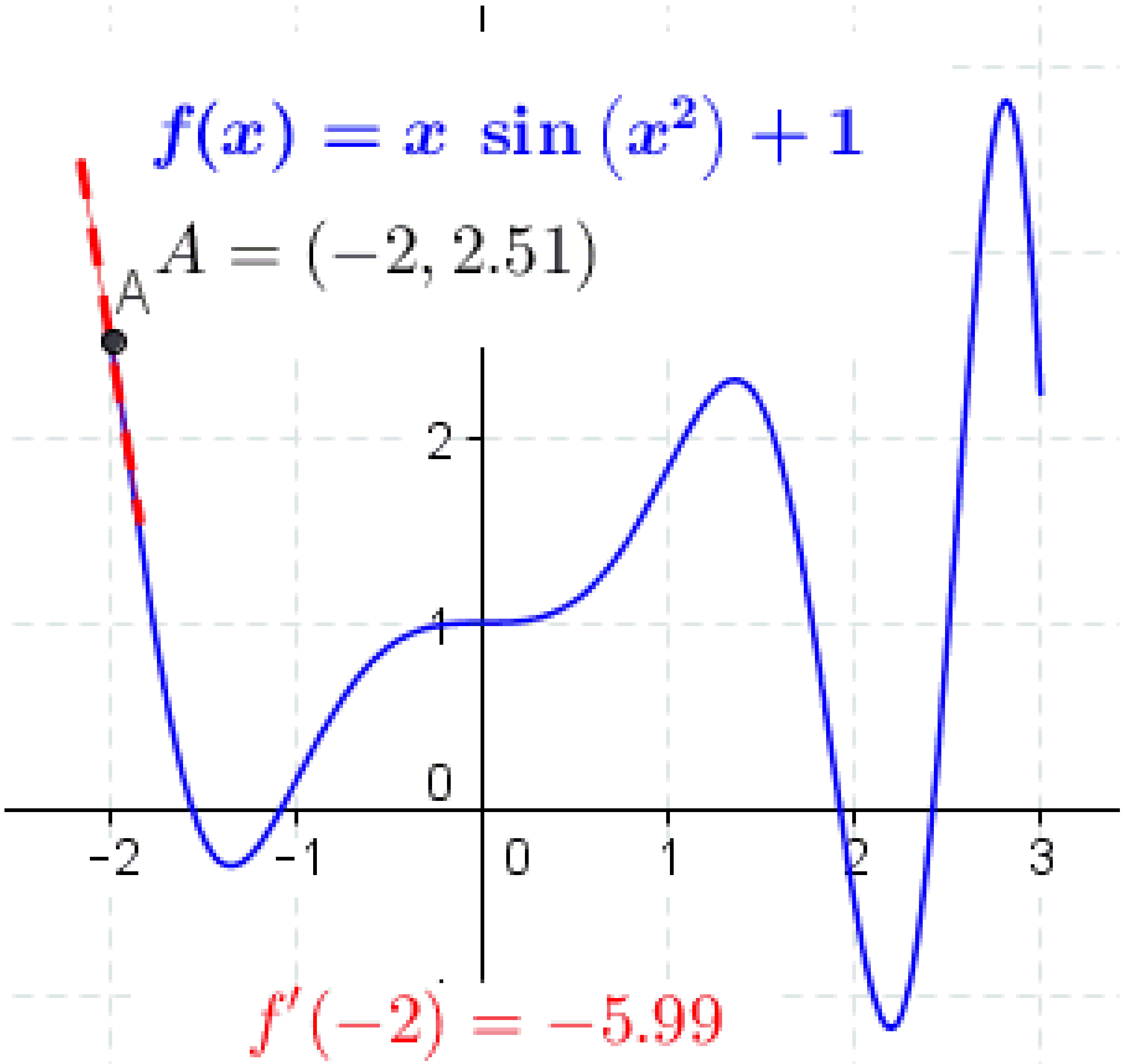




0 8
1
6
w¹₁₁
w¹₁₂
w¹₁₂
w¹₂₂
w¹₁₃
w¹₂₃
a¹₁
a¹₂
a¹₃
w²₁
w²₂
w²₃
a_2^2

Gradient descent

Derivative



Partial derivatives

$$f(x, y) = x^2 + xy + y^2$$

$$\frac{\partial f}{\partial x}(x, y) = 2x + y$$

$$\frac{\partial f(x, y)}{\partial y} = y + 2y$$

$$\nabla f(a) = [\frac{\partial f}{\partial x_1}(a), \ldots, \frac{\partial f}{\partial x_n}(a))]$$

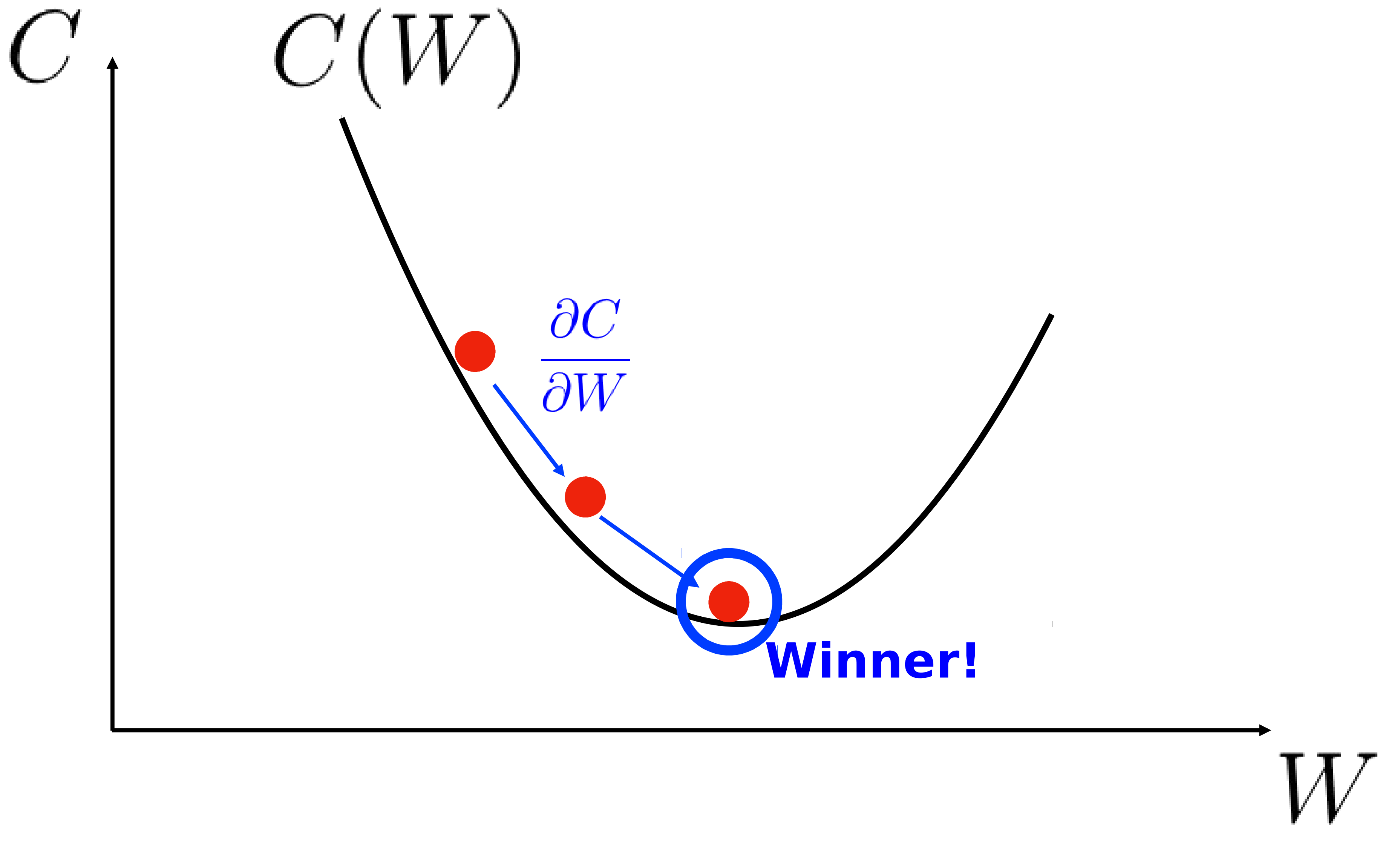
Chain rule

$$g(x) = y; f(y) = z \quad \frac{\partial f(g(x))}{\partial x} =?$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} * \frac{\partial y}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x} =$$

$$f'(g(x)) * g'(x)$$

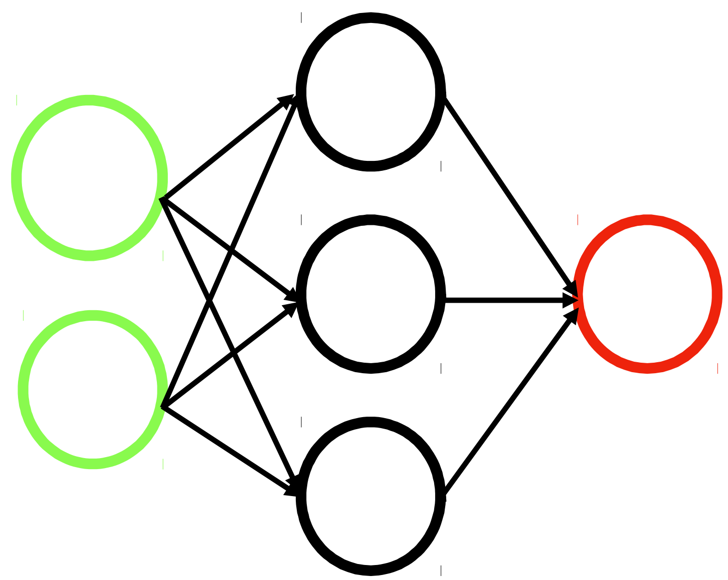
Gradient descent



Backward propagation

Actually predicting data

$$\begin{bmatrix} 3 & 5 \\ 9 & 6 \\ 7 & 8 \end{bmatrix}$$

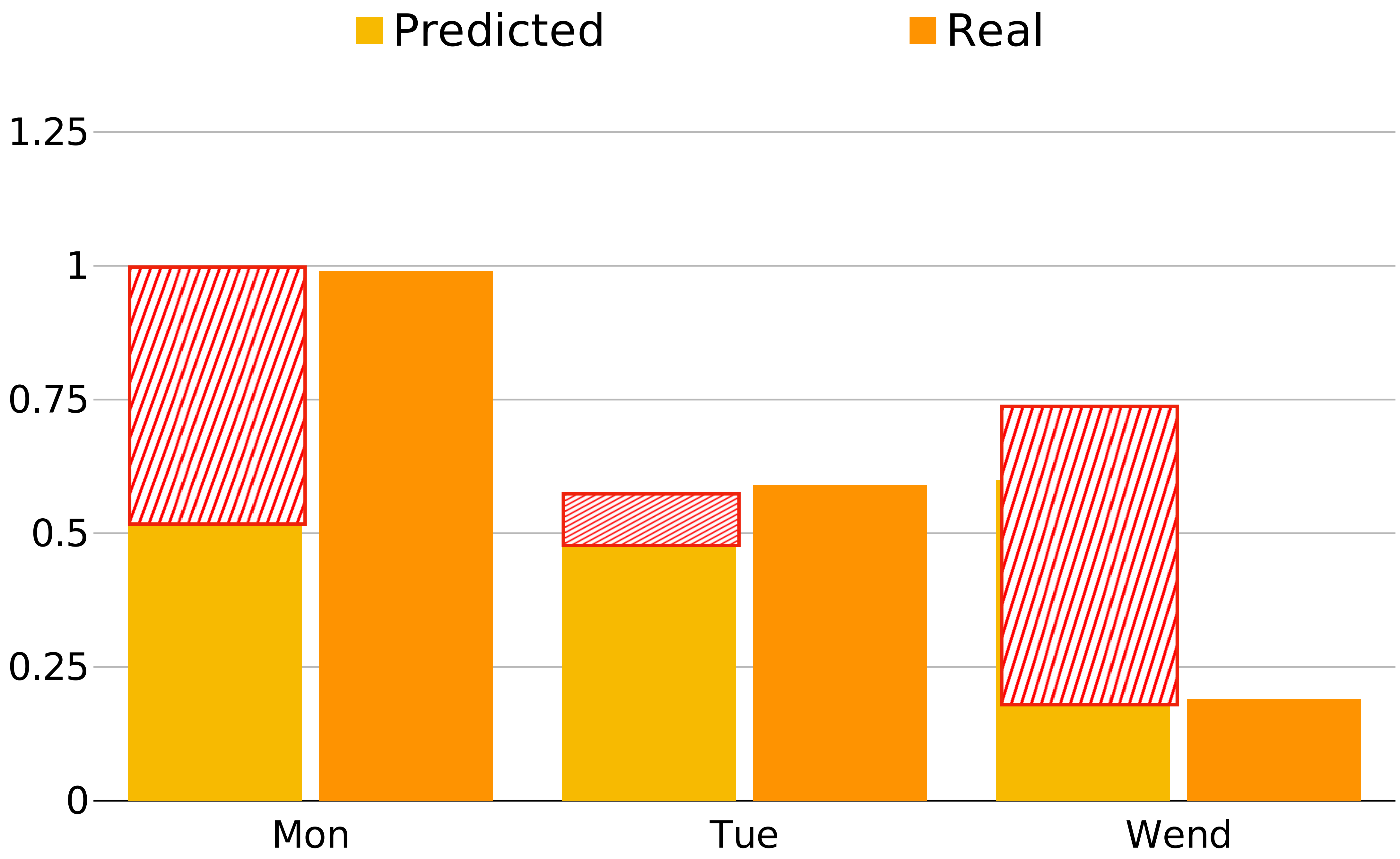


Predicted values

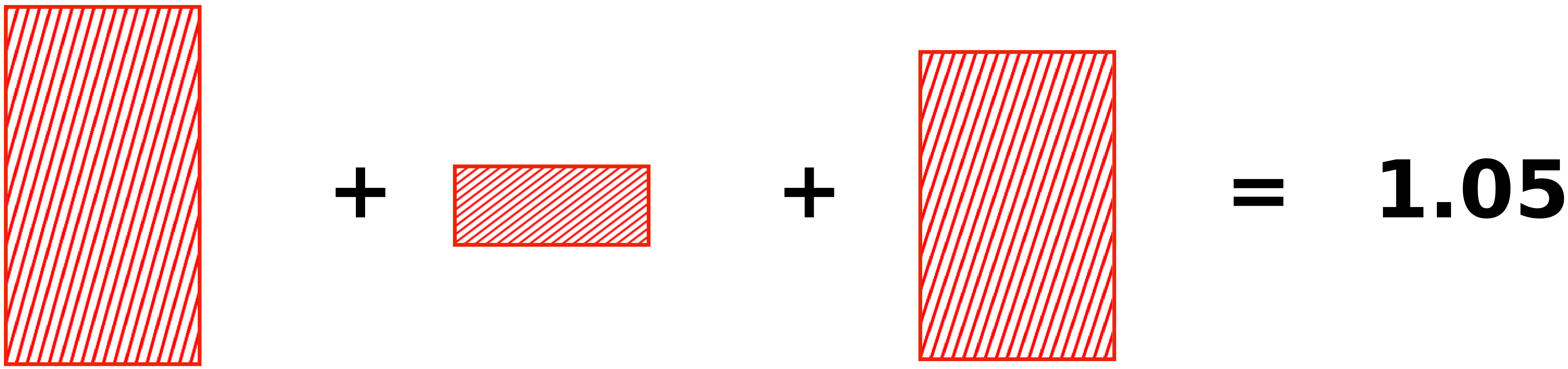
$$\begin{bmatrix} 0.52 \\ 0.49 \\ 0.6 \end{bmatrix}$$

Real values

$$\begin{bmatrix} 0.99 \\ 0.59 \\ .19 \end{bmatrix}$$


Predicted
Real
1.25
1
0.75
0.5
0.25
0
Mon
Tue
Wend

Total Error



Back propagation

$$\frac{\partial C}{\partial W} = ?$$

$$W_1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \end{bmatrix}$$

Back propagation

$$\frac{\partial C}{\partial W} = ?$$

$$\frac{\partial C}{\partial W_1} = \begin{bmatrix} \frac{\partial C}{\partial w_{11}^1} & \frac{\partial C}{\partial w_{12}^1} & \frac{\partial C}{\partial w_{13}^1} \\ \frac{\partial C}{\partial w_{21}^1} & \frac{\partial C}{\partial w_{22}^1} & \frac{\partial C}{\partial w_{23}^1} \end{bmatrix}$$

$$\frac{\partial C}{\partial W_2} = \begin{bmatrix} \frac{\partial C}{\partial w_1^2} \\ \frac{\partial C}{\partial w_2^2} \\ \frac{\partial C}{\partial w_3^2} \end{bmatrix}$$

BP: Last Layer

the output units. Differentiating equation (3) for a particular case, c, and suppressing the index c gives

$$\partial E/\partial y_j = y_j - d_j \tag{4}$$

We can then apply the chain rule to compute $\partial E/\partial x_j$

$$\partial E/\partial x_j = \partial E/\partial y_j \cdot dy_j/dx_j$$

Differentiating equation (2) to get the value of dy_j/dx_j and substituting gives

$$\partial E/\partial x_j = \partial E/\partial y_j \cdot y_j(1 - y_j) \tag{5}$$

This means that we know how a change in the total input x to an output unit will affect the error. But this total input is just a linear function of the states of the lower level units and it is also a linear function of the weights on the connections, so it is easy to compute how the error will be affected by changing these states and weights. For a weight w_{ji}, from i to j the derivative is

$$\begin{aligned}\partial E/\partial w_{ji} &= \partial E/\partial x_j \cdot \partial x_j/\partial w_{ji}\\ &= \partial E/\partial x_j \cdot y_i \end{aligned} \tag{6}$$

and for the output of the i^{th} unit the contribution to $\partial E/\partial y_i$

Back propagation

$$\frac{\partial C(y, a^2)}{\partial W^2} = ?$$

Back propagation last layer

Remember chain rule!

$$g(x) = y; f(y) = z \qquad f'(g(x)) * g'(x)$$

Here:

$$g(x) = \sigma(z^L) = a^L$$

$$f(y) = C(y, a^L)$$

Back propagation last layer

$$
\begin{aligned}
C'(y, a^L) &= C'(y, \sigma(z^L)) \\
&= C'(y, a^L) * \sigma'(z^L)
\end{aligned}
$$

Back propagation

$$\frac{\partial C(y, a^2)}{\partial W^2} = \frac{\partial C(y, a^2)}{\partial a^2} * \frac{\partial a^2}{\partial W^2}$$

Back propagation

$$\frac{\partial C(y, a^2)}{\partial W^2} = \frac{\partial C(y, a^2)}{\partial a^2} * \frac{\partial \sigma(z^2)}{\partial W^2}$$

Back propagation

$$\frac{\partial C(y, a^2)}{\partial W^2} = \frac{\partial C(y, a^2)}{\partial a^2} * \frac{\partial \sigma(z^2)}{\partial z^2} * \frac{\partial z^2}{\partial W^2}$$

Cost derivative

$$f(\hat{y}, \hat{x}) = \frac{1}{2} \sum_{i=0}^{n} (y_i - x_i)^2$$

$$\frac{\partial f(\hat{y}, \hat{x})}{\partial x_j} = \frac{\partial \frac{1}{2} \sum_{i=0}^{n} (y_i - x_i)^2}{\partial x_j}$$

$$\frac{\partial f(\hat{y}, \hat{x})}{\partial x_j} = \frac{1}{2} \sum_{i=0}^{n} \frac{\partial (y_i - x_i)^2}{\partial x_j}$$

Cost derivative

$$\frac{\partial f(\hat{y}, \hat{x})}{\partial x_j} = \frac{1}{2} \sum_{i=0}^{n} \frac{\partial (y_i - x_i)^2}{\partial x_j}$$

$$\frac{\partial f(\hat{y}, \hat{x})}{\partial x_j} = \sum_{i=0}^{n} (y_i - x_i) * \frac{\partial (y_i - x_i)}{\partial x_j}$$

$i \neq j$

$i = j$

$$\frac{\partial y_i - x_i}{\partial x_j} = 0$$

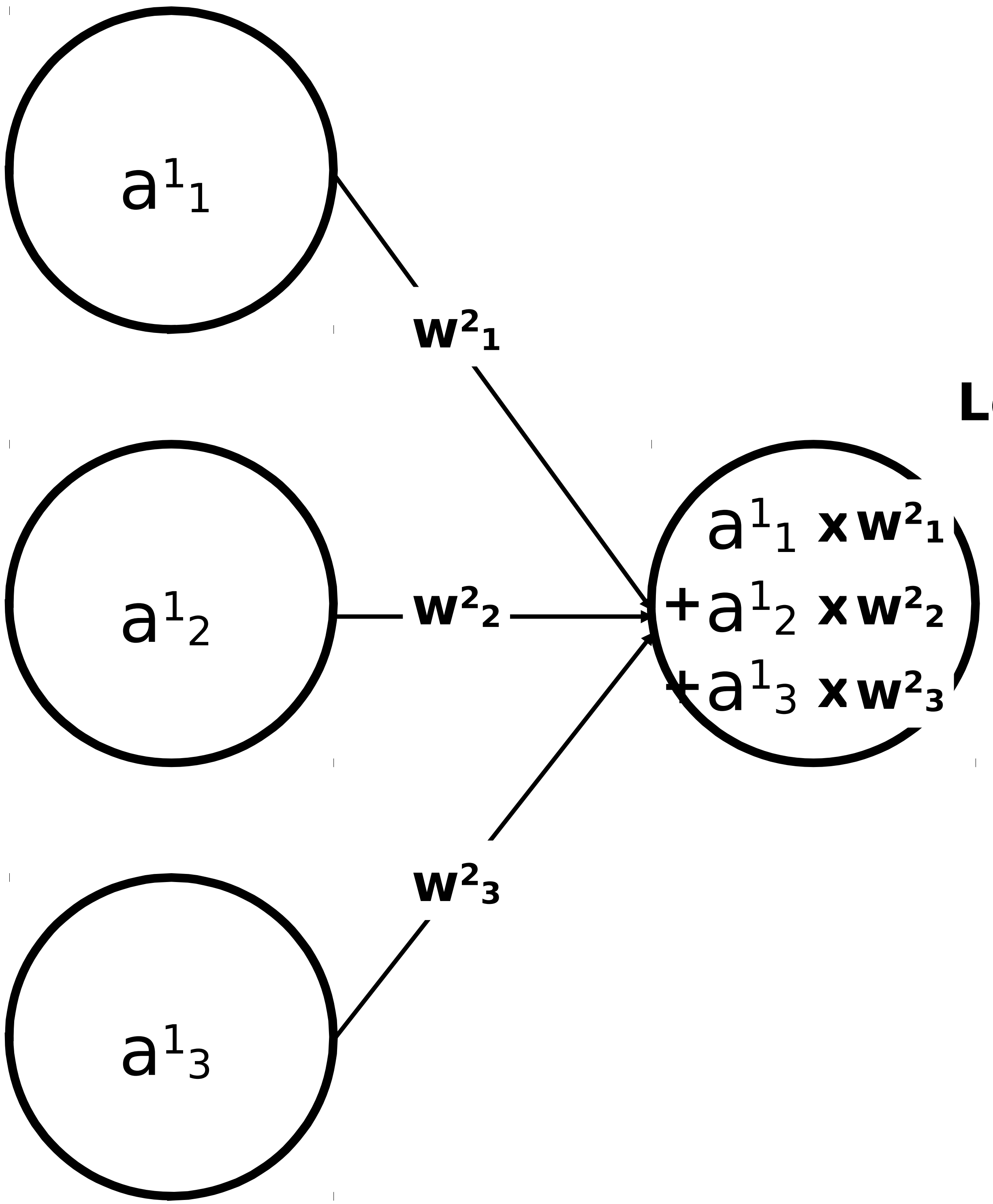
$$\frac{\partial y_j - x_j}{\partial x_j} = -1$$

Back propagation

$$\frac{\partial C(y, a^2)}{\partial W^2} = \frac{\partial C(y, a^2)}{\partial a^2} * \frac{\partial \sigma(z^2)}{\partial z^2} * \frac{\partial z^2}{\partial W^2}$$

$$\frac{\partial C(y, a^2)}{\partial W^2} = \underbrace{(a^2 - y) * \sigma'(z^2)}_{\delta^2} * \frac{\partial z^2}{\partial W^2}$$

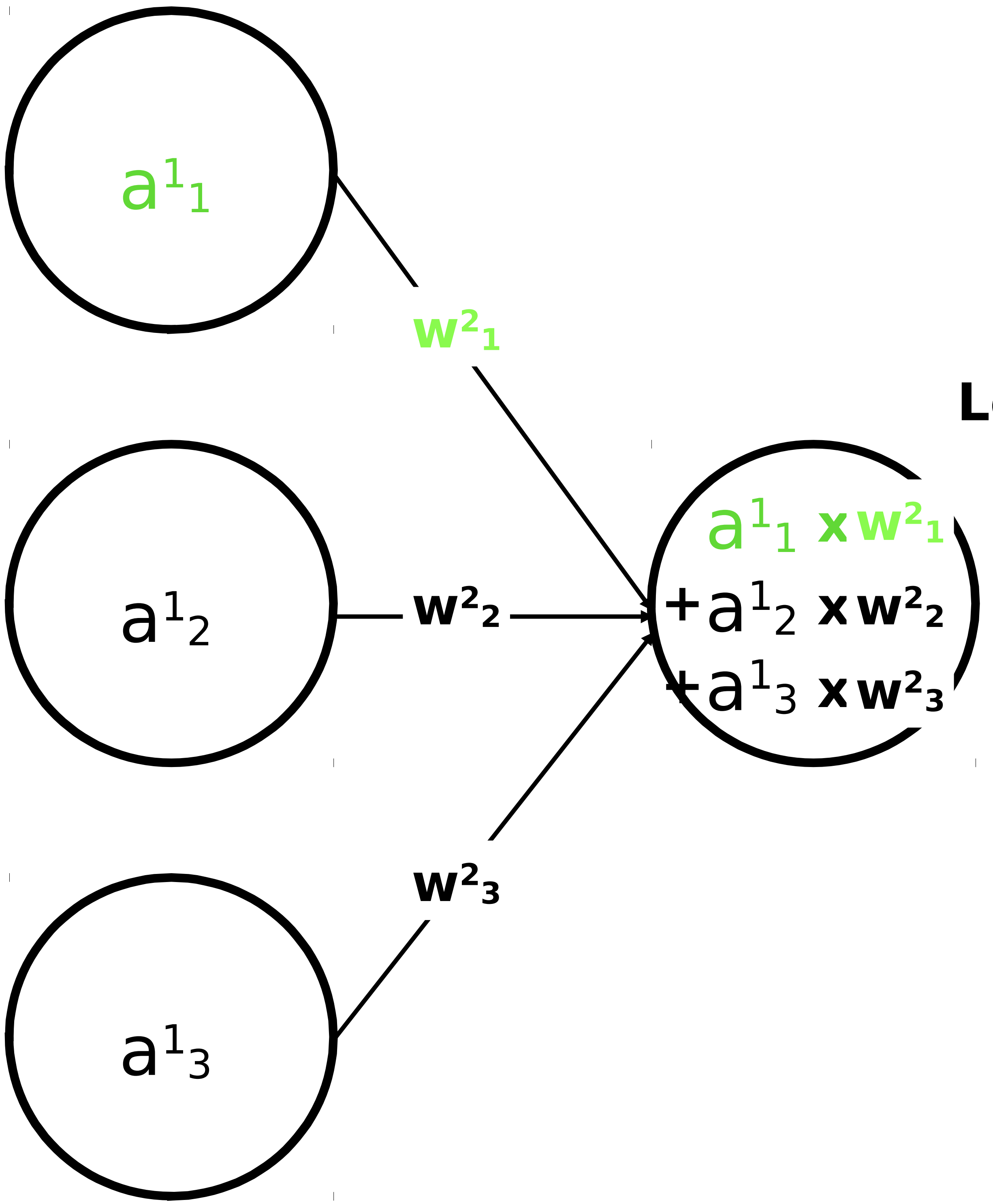
Back propagation



$$\frac{\partial z^2}{\partial W^2} = \frac{\partial a^2 W^2}{\partial W^2}$$

Let’s look at a single synapse

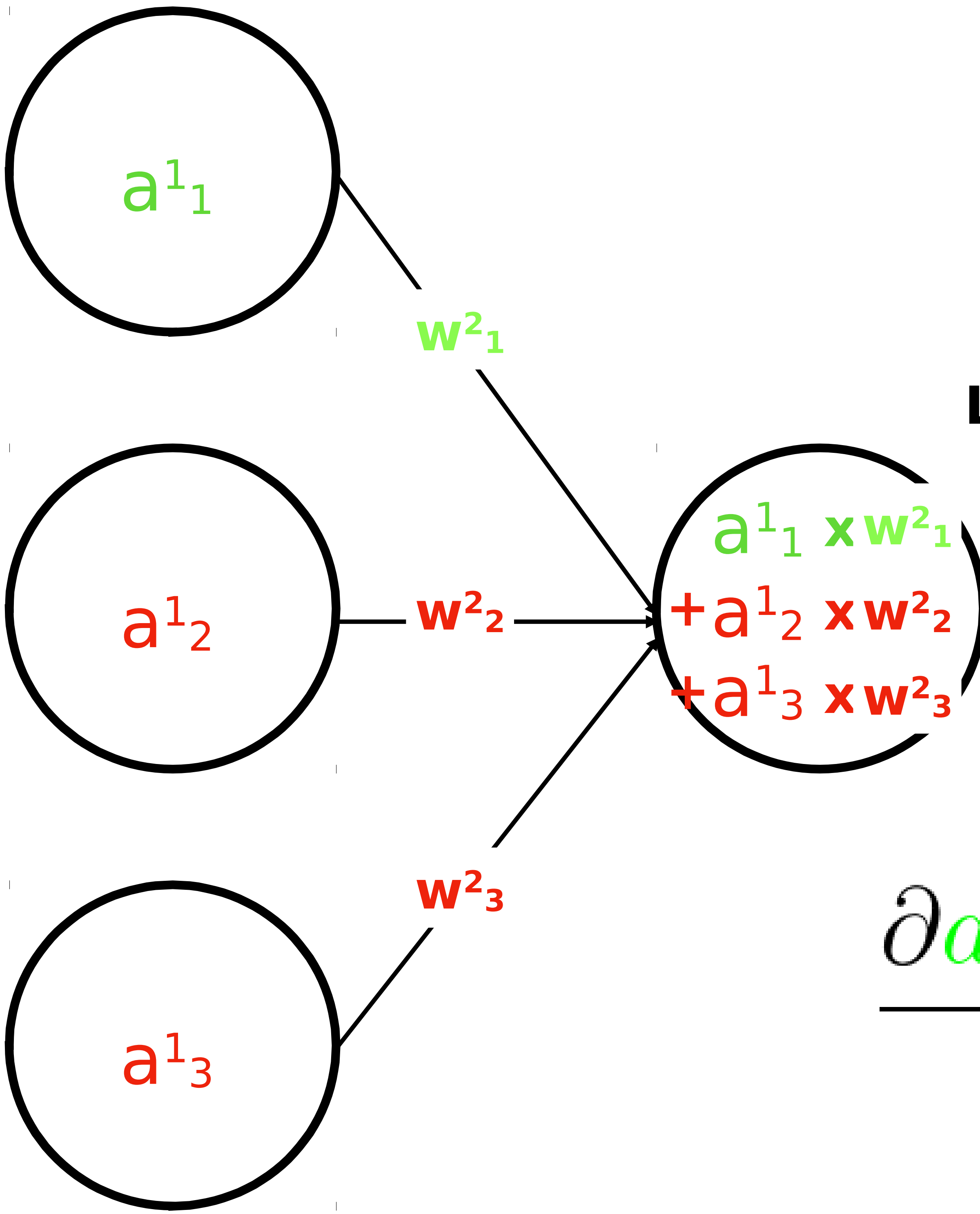
Back propagation



$$\frac{\partial z^2}{\partial W^2} = \frac{\partial a^2 W^2}{\partial W^2}$$

Let’s look at a single synapse

Back propagation



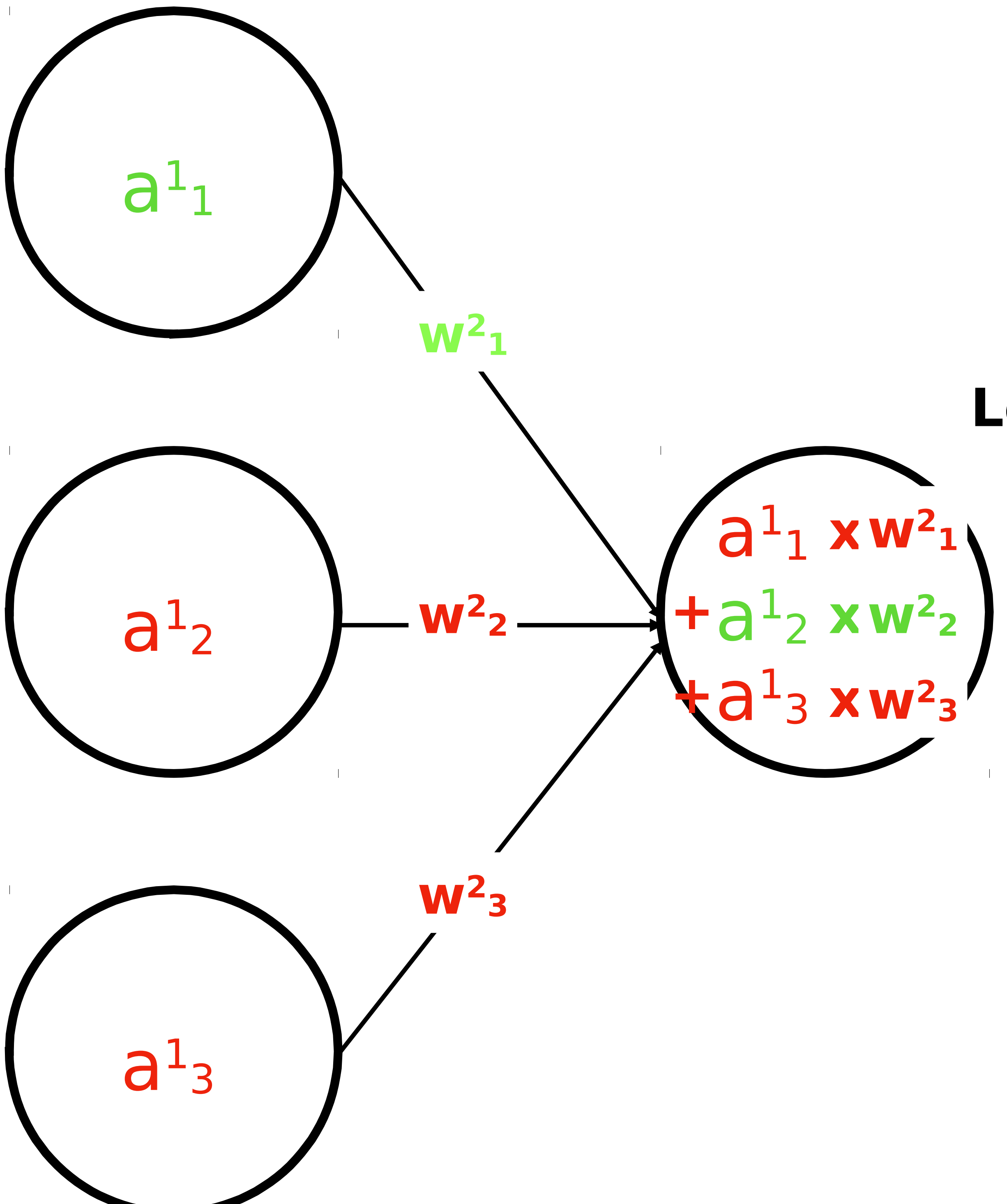
$$\frac{\partial z^2}{\partial W^2} = \frac{\partial a^2 W^2}{\partial W^2}$$

Let’s look at a single synapse

$$\frac{\partial S + a^1_1 * w^2_1}{\partial w^2_1} =$$

$$\frac{\partial a^1_1 * w^2_1}{\partial w^2_1} = a^1_1$$

Back propagation

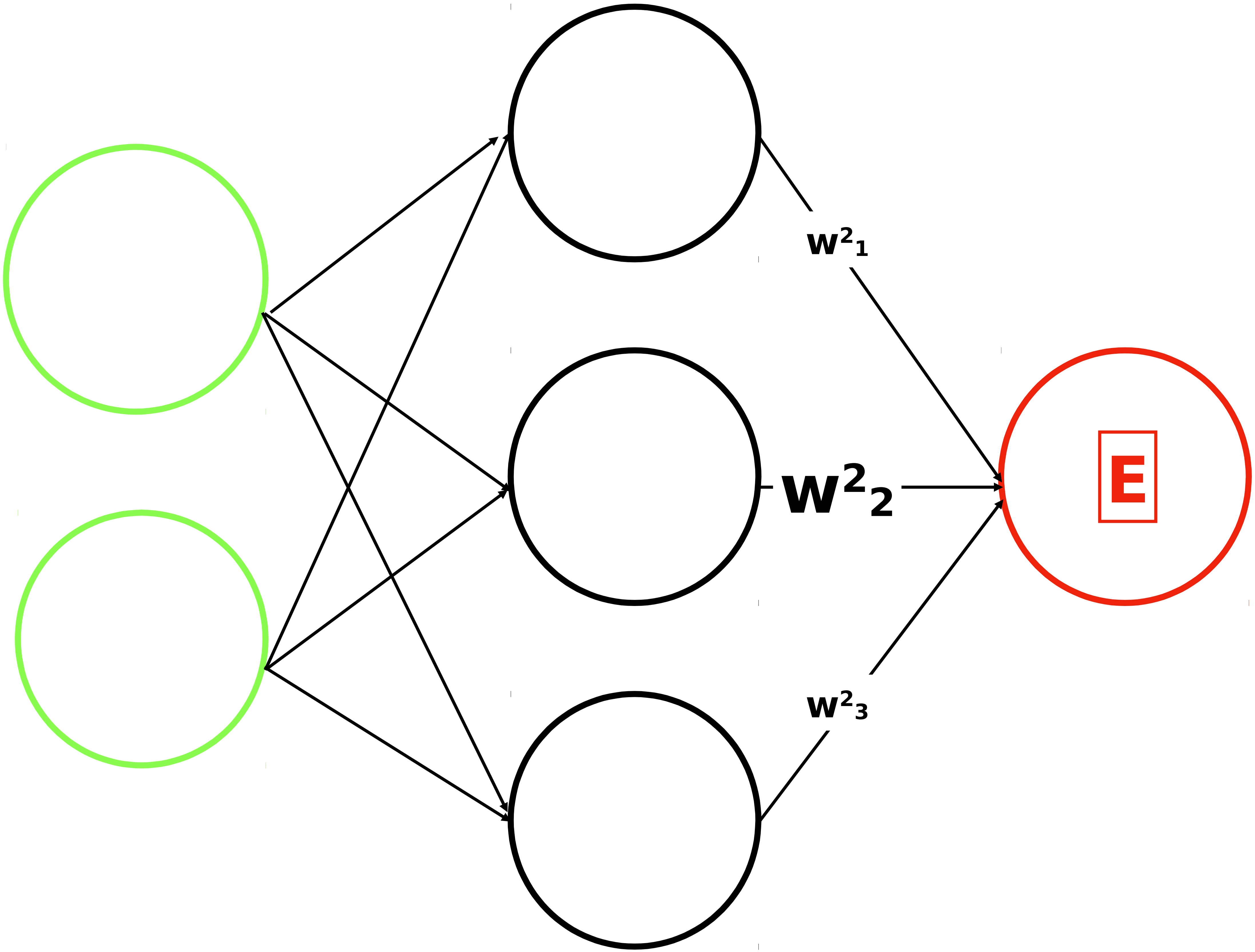


$$\frac{\partial z^2}{\partial W^2} = \frac{\partial a^2 W^2}{\partial W^2}$$

Let's look at a single synapse

$$\frac{\partial S + a_2^1 * w_2^2}{\partial w_2^2} =$$

$$\frac{\partial a_2^1 * w_2^2}{\partial w_2^2} = a_2^1$$



w^2_1
w^2_2
w^2_3
E

Back propagation

$$\frac{\partial C(y, a^2)}{\partial W^2} = (a^2 - y) * \sigma'(z^2) * \frac{\partial z^2}{\partial W^2}$$

$$\frac{\partial C(y, a^2)}{\partial W^2} = (a^1)^T * (a^2 - y) * \sigma'(z^2)$$

The formulas

$$\delta^2 = (a^2 - y) * \sigma'(z^2)$$

$$\frac{\partial C(y, a^2)}{\partial W^2} = (a^1)^T * \delta^2$$

BP: Middle Layer

We have now seen how to compute $\partial E/\partial y$ for any unit in the penultimate layer when given $\partial E/\partial y$ for all units in the last layer. We can therefore repeat this procedure to compute this term for successively earlier layers, computing $\partial E/\partial w$ for the weights as we go.

One way of using $\partial E/\partial w$ is to change the weights after every input–output case. This has the advantage that no separate memory is required for the derivatives. An alternative scheme, which we used in the research reported here, is to accumulate $\partial E/\partial w$ over all the input–output cases before changing the weights. The simplest version of gradient descent is to change each weight by an amount proportional to the accumulated $\partial E/\partial w$

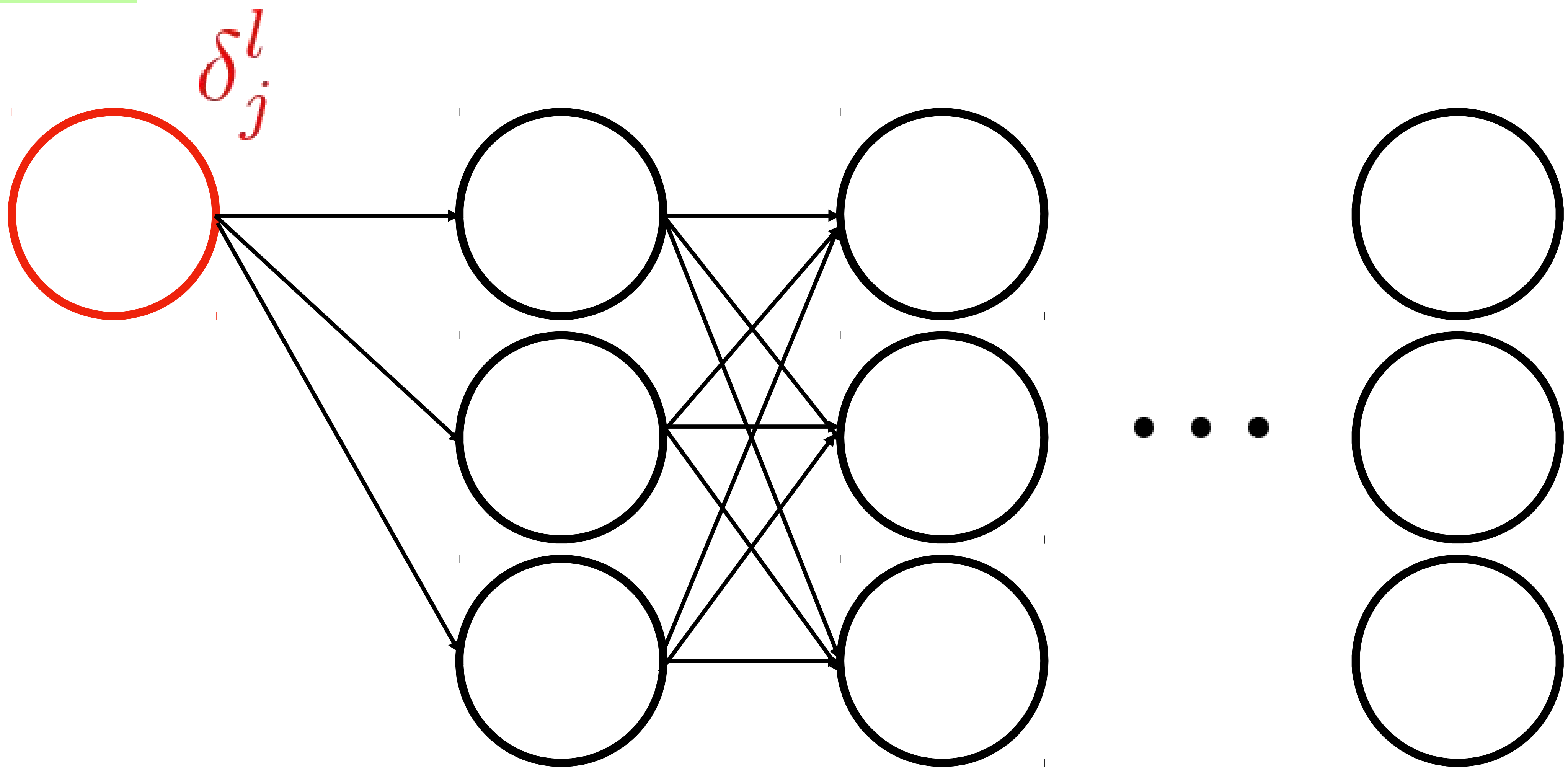
$$\Delta w = -\varepsilon \partial E/\partial w \tag{8}$$

This method does not converge as rapidly as methods which make use of the second derivatives, but it is much simpler and can easily be implemented by local computations in parallel hardware. It can be significantly improved, without sacrificing the simplicity and locality, by using an acceleration method in which the current gradient is used to modify the velocity of the point in weight space instead of its position

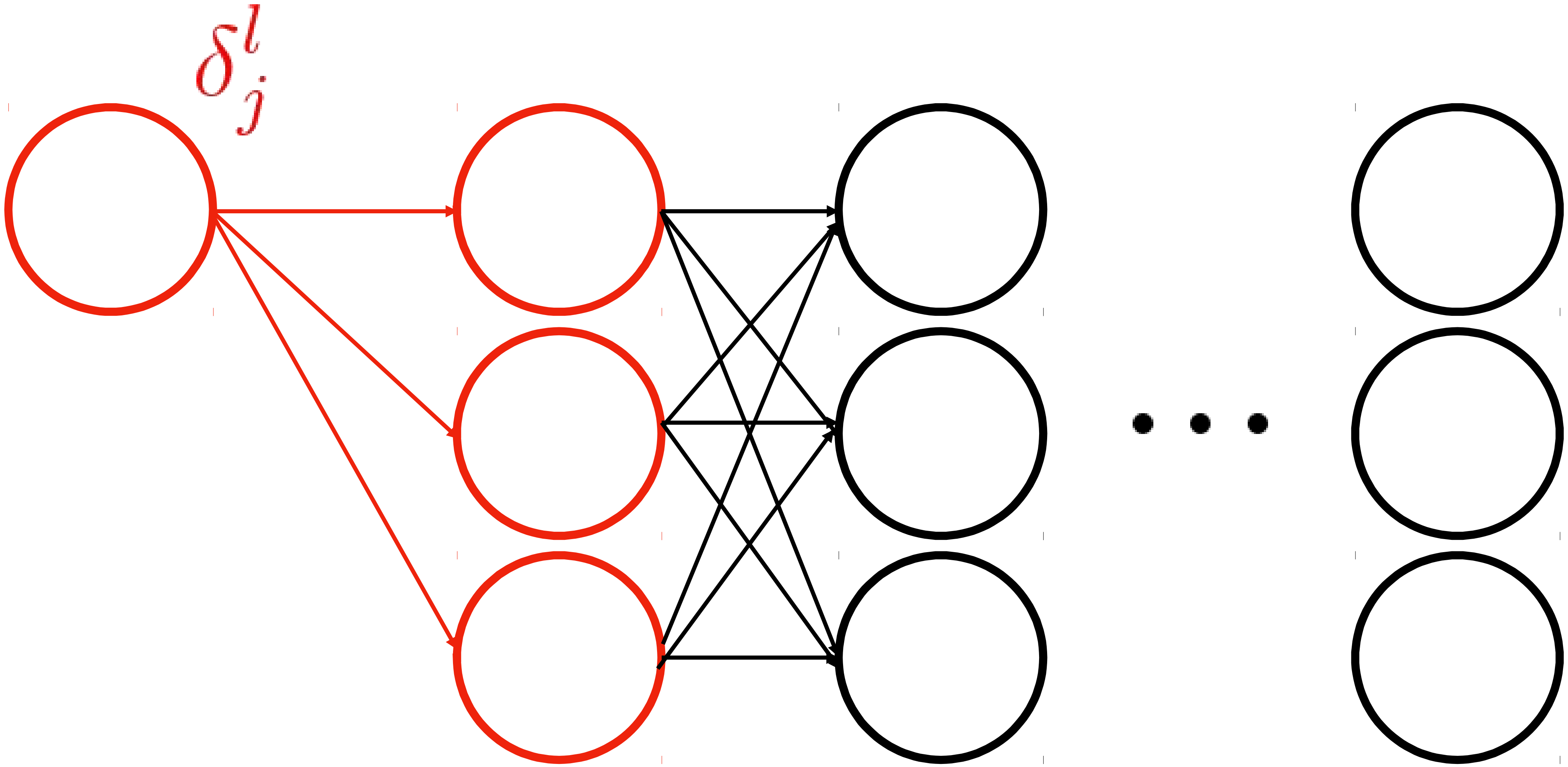
$$\Delta w(t) = -\varepsilon \partial E/\partial w(t) + \alpha \Delta w(t-1) \tag{9}$$

where t is incremented by 1 for

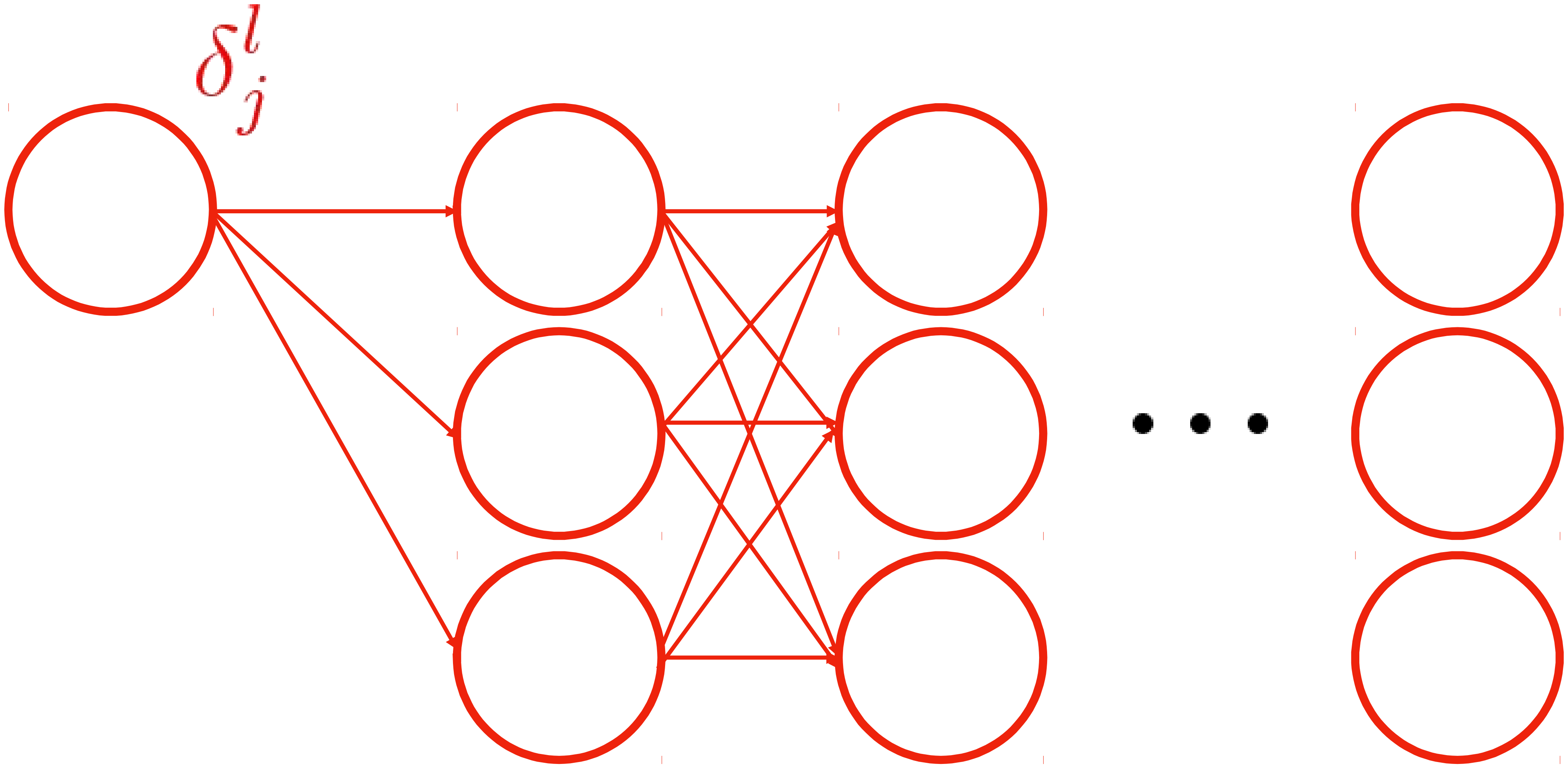
Back propagation



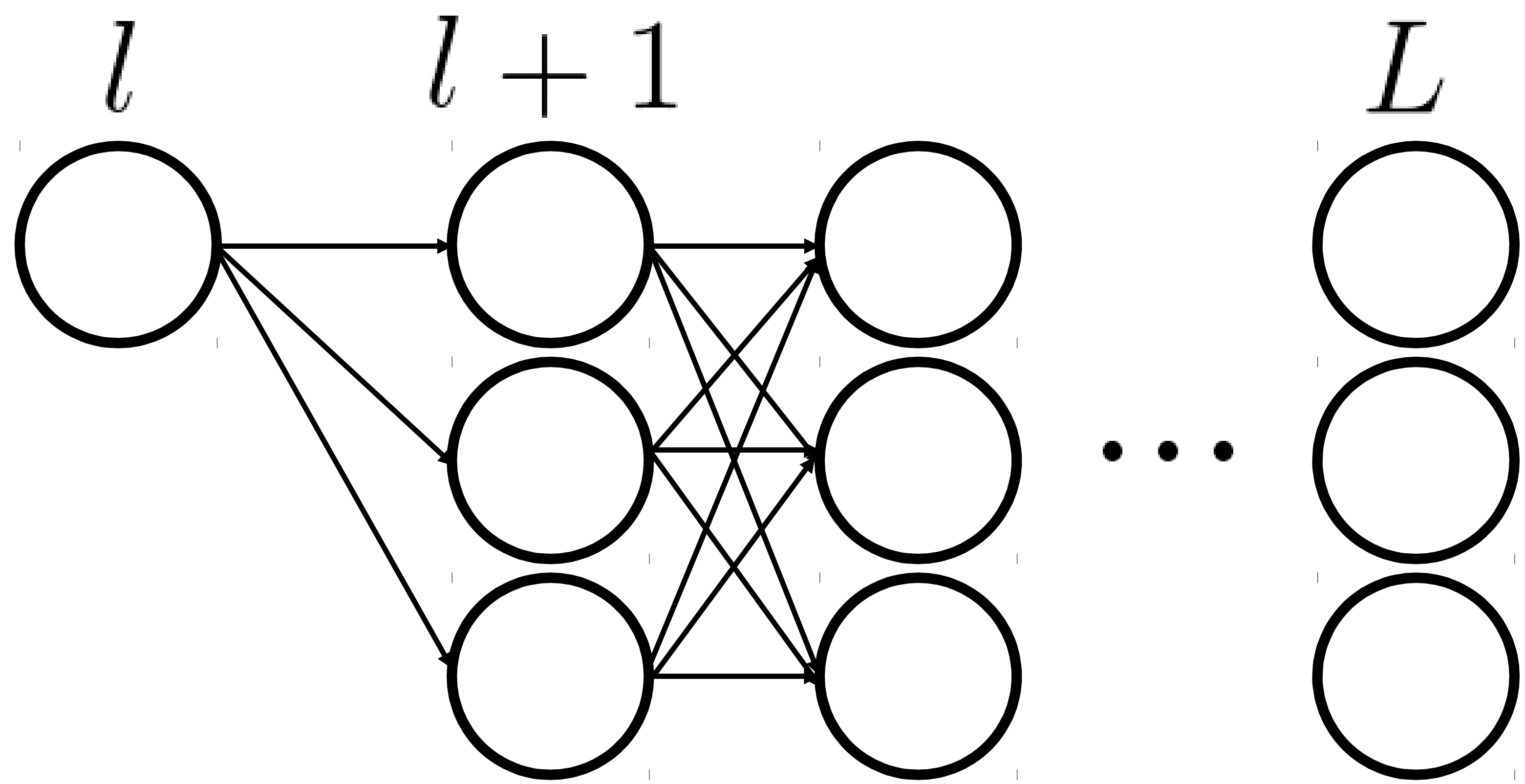
Back propagation



Back propagation



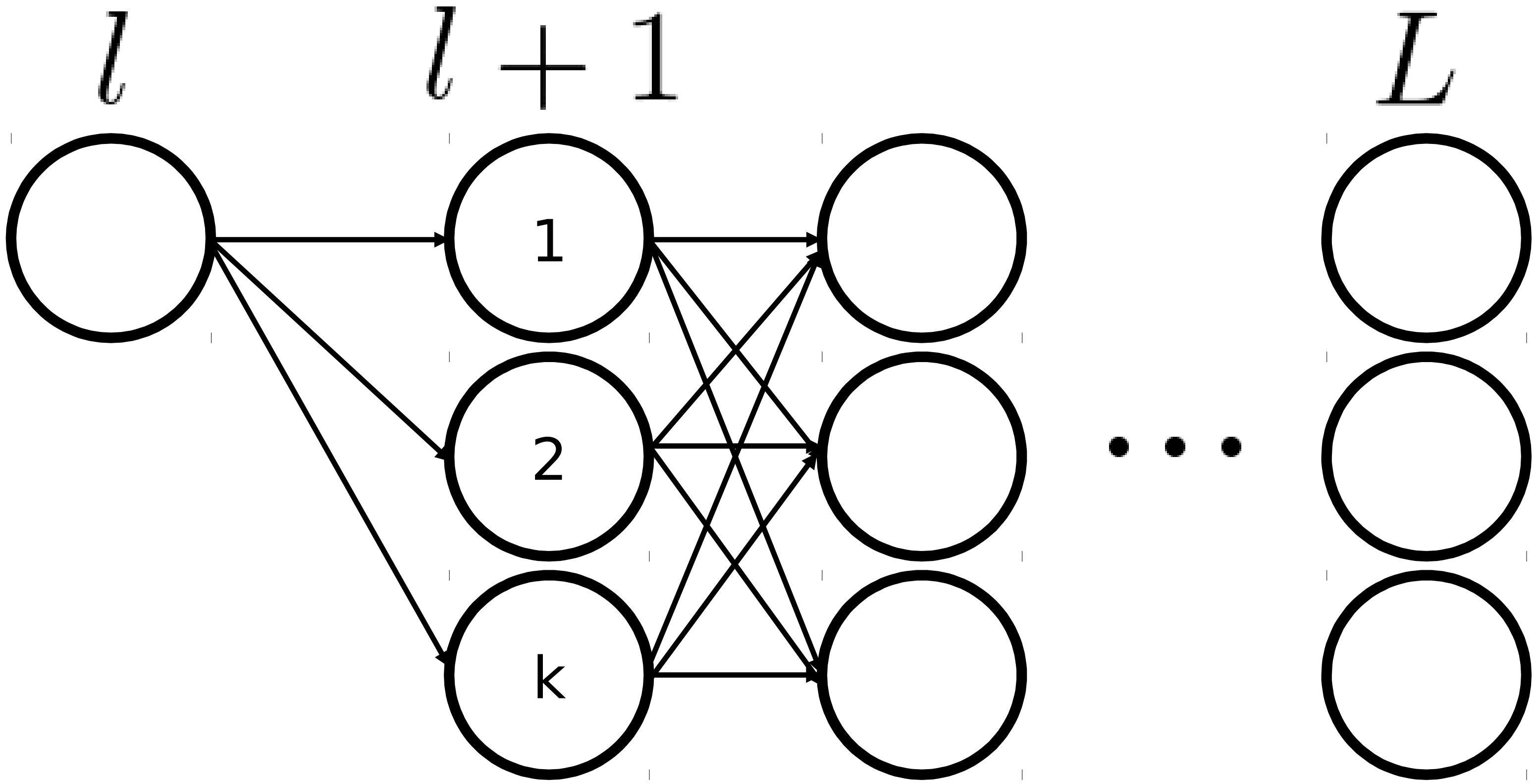
Back propagation any layer



$$\delta_j^l = ?$$

Back propagation any layer

$$\delta_j^L = (a_j^L - y_j) * \sigma'(z_j^L)$$

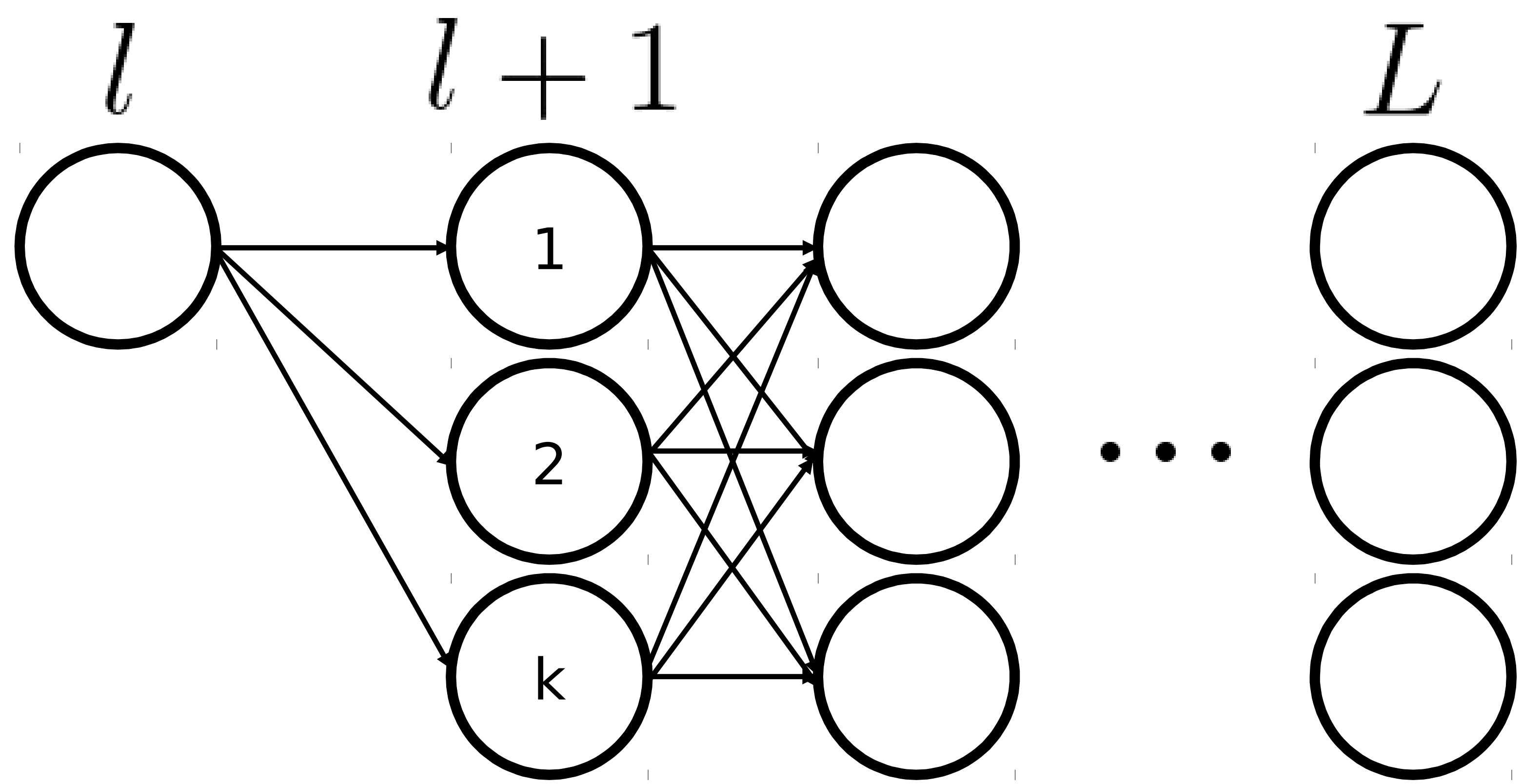


$$\delta_j^l = ?$$

Back propagation any layer

$$\delta_j^L = (a_j^L - y_j) * \sigma'(z_j^L)$$

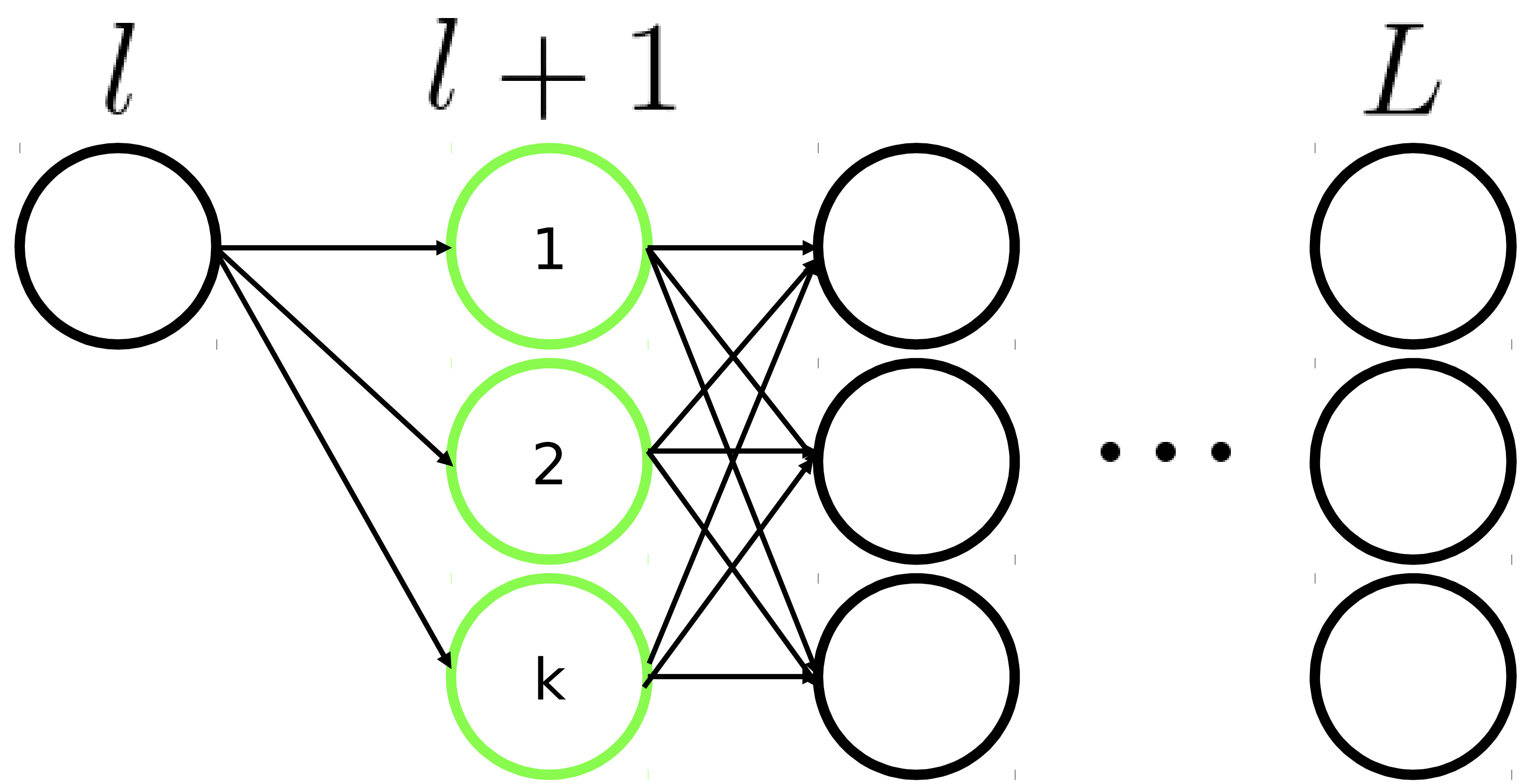
$$\delta_j^l = \sum_{n=0}^{k}$$



Back propagation any layer

$$\delta_j^L = (a_j^L - y_j) * \sigma'(z_j^L)$$

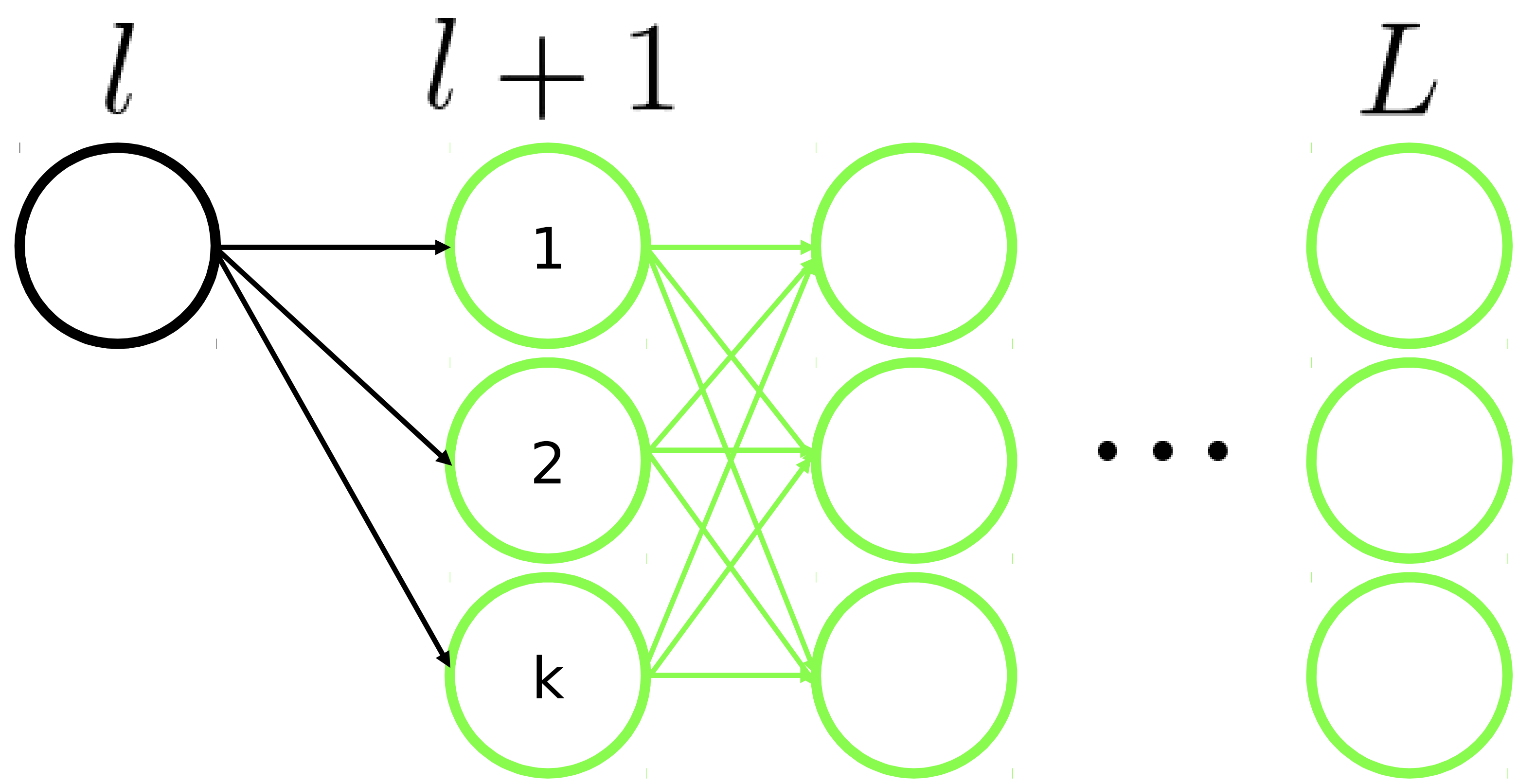
$$\delta_j^l = \sum_{n=0}^{k}$$



Back propagation any layer

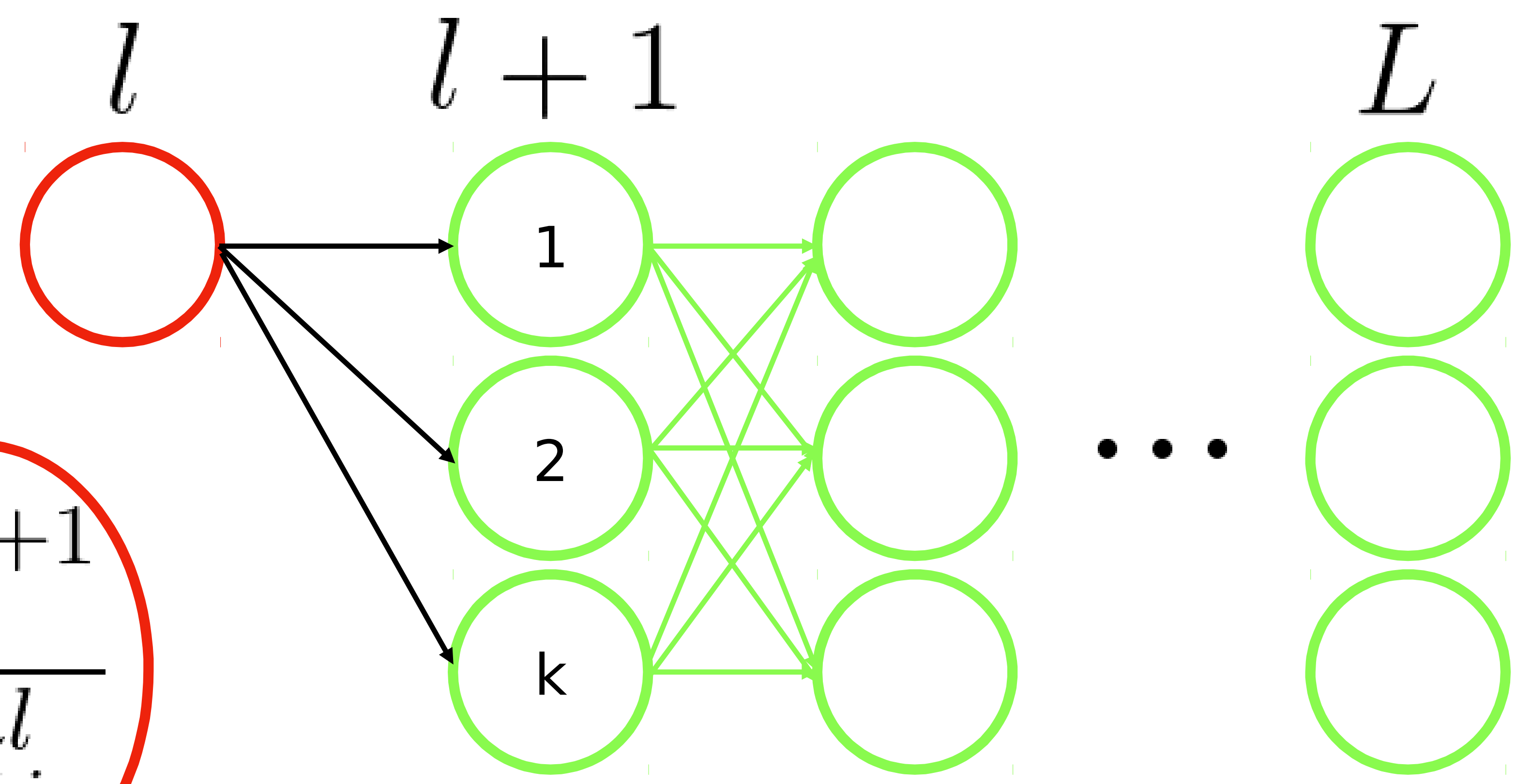
$$\delta_j^L = (a_j^L - y_j) * \sigma'(z_j^L)$$

$$\delta_j^l = \sum_{n=0}^{k} \frac{\partial C}{\partial z_k^{l+1}}$$



Back propagation any layer

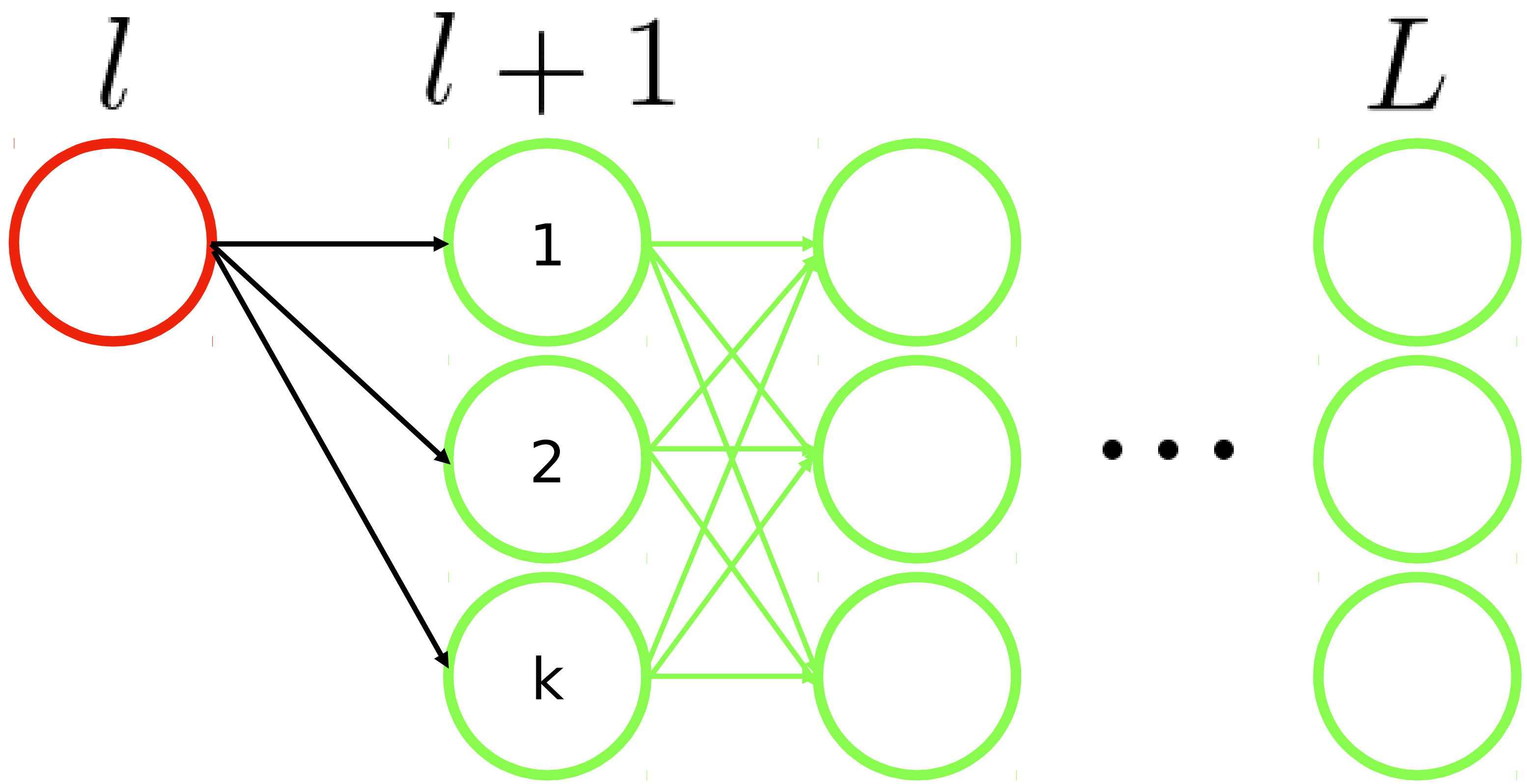
$$\delta_j^L = (a_j^L - y_j) * \sigma'(z_j^L)$$



$$\delta_j^l = \sum_{n=0}^{k} \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

Back propagation any layer

$$\delta_j^L = (a_j^L - y_j) * \sigma'(z_j^L)$$



$$\delta_j^l = \sum_{n=0}^{k} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

Back propagation any layer

$$\delta_j^l = \sum_{n=0}^{k} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$z_k^{l+1} = \sum_{i=0}^{n} a_i^l * w_{ik}^{l+1}$$

$$a_i^l = \sigma(z_i^l)$$

Zero for all $i \neq j$

$$\frac{\partial \sigma(z_i^l) * w_k^{l+1}}{\partial z_j^l}$$

Back propagation any layer

$$\frac{\partial \sigma(z_i^l) * w_k^{l+1}}{\partial z_j^l} \quad = w_{jk}^{l+1} \sigma'(z_j^l)$$

Back propagation any layer

$$\delta_j^l = \sum_{n=0}^{k} \delta_k^{l+1} * w_{jk}^{l+1} * \sigma'(z_j^l)$$

$$\delta^l = \delta^{l+1} * (W^{l+1})^T * \sigma'(z^l)$$

Back propagation any layer

$$\frac{\partial C(y, a^2)}{\partial W^1} = \quad \delta^1 * \frac{\partial z^1}{\partial W^1}$$

$$\frac{\partial C(y, a^2)}{\partial W^1} = (X)^T * \delta^1$$

Back propagation any layer

$$\delta^l = \delta^{l+1} * (W^{l+1})^T * \sigma'(z^l)$$

$$\delta^1 = \delta^2 * (W^2)^T * \sigma'(z^1)$$

$$\frac{\partial C(y, a^2)}{\partial W^1} = (X)^T * \delta^1$$

Now all together

$$\delta^2 = (a^2 - y) * \sigma'(z^2)$$

$$\frac{\partial C(y, a^2)}{\partial W^2} = (a^1)^T * \delta^2$$

$$\delta^1 = \delta^2 * (W^2)^T * \sigma'(z^1)$$

$$\frac{\partial C(y, a^2)}{\partial W^1} = (X)^T * \delta^1$$

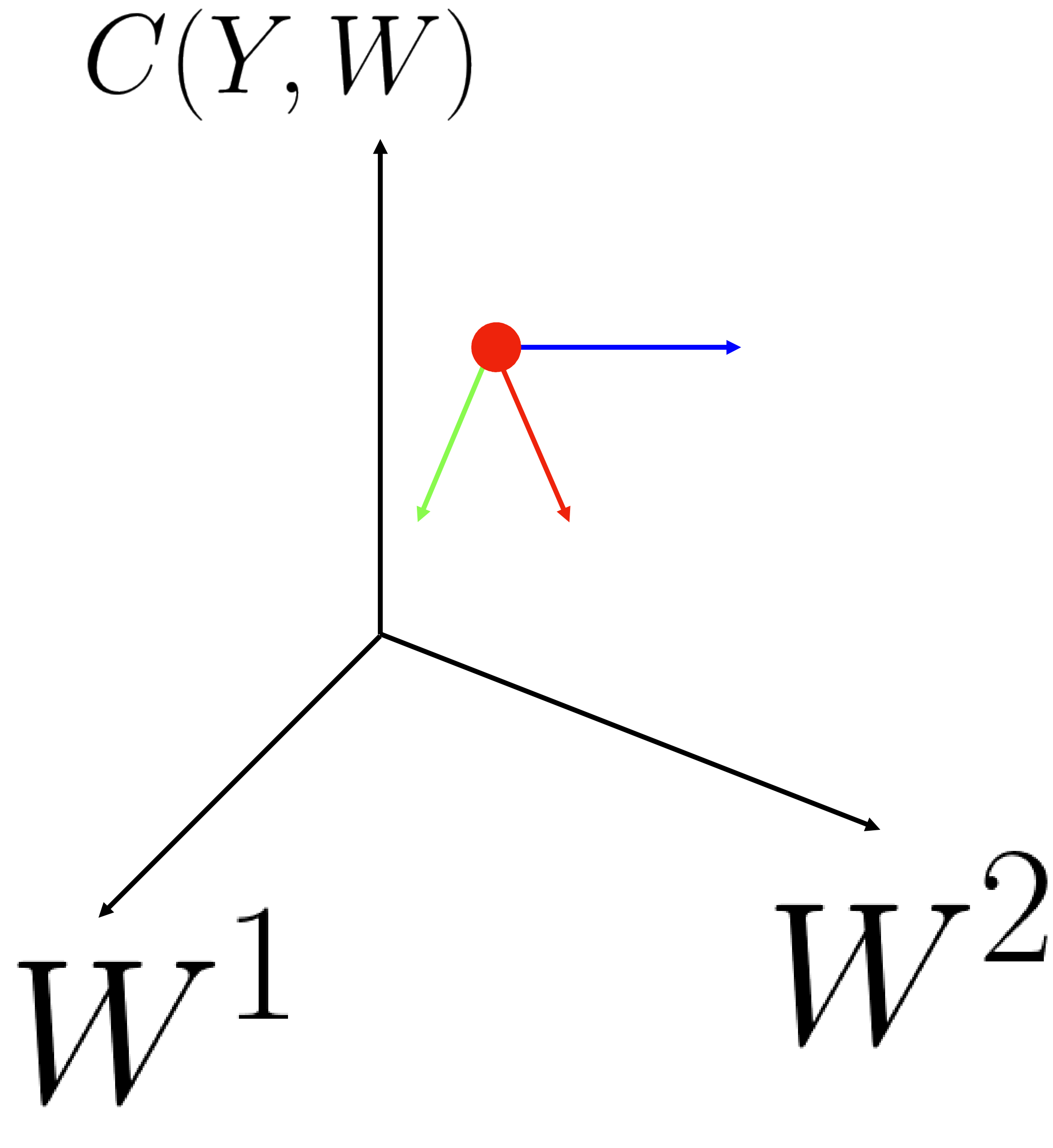
The gradient

$$\begin{bmatrix} \mathbf{3} & \mathbf{5} \\ \mathbf{1} & \mathbf{6} \\ \mathbf{0} & \mathbf{8} \end{bmatrix}$$

$$\frac{\partial C(y_1, a_1^2)}{\partial W}$$

$$\frac{\partial C(y_2, a_2^2)}{\partial W}$$

$$\frac{\partial C(y_3, a_3^2)}{\partial W}$$



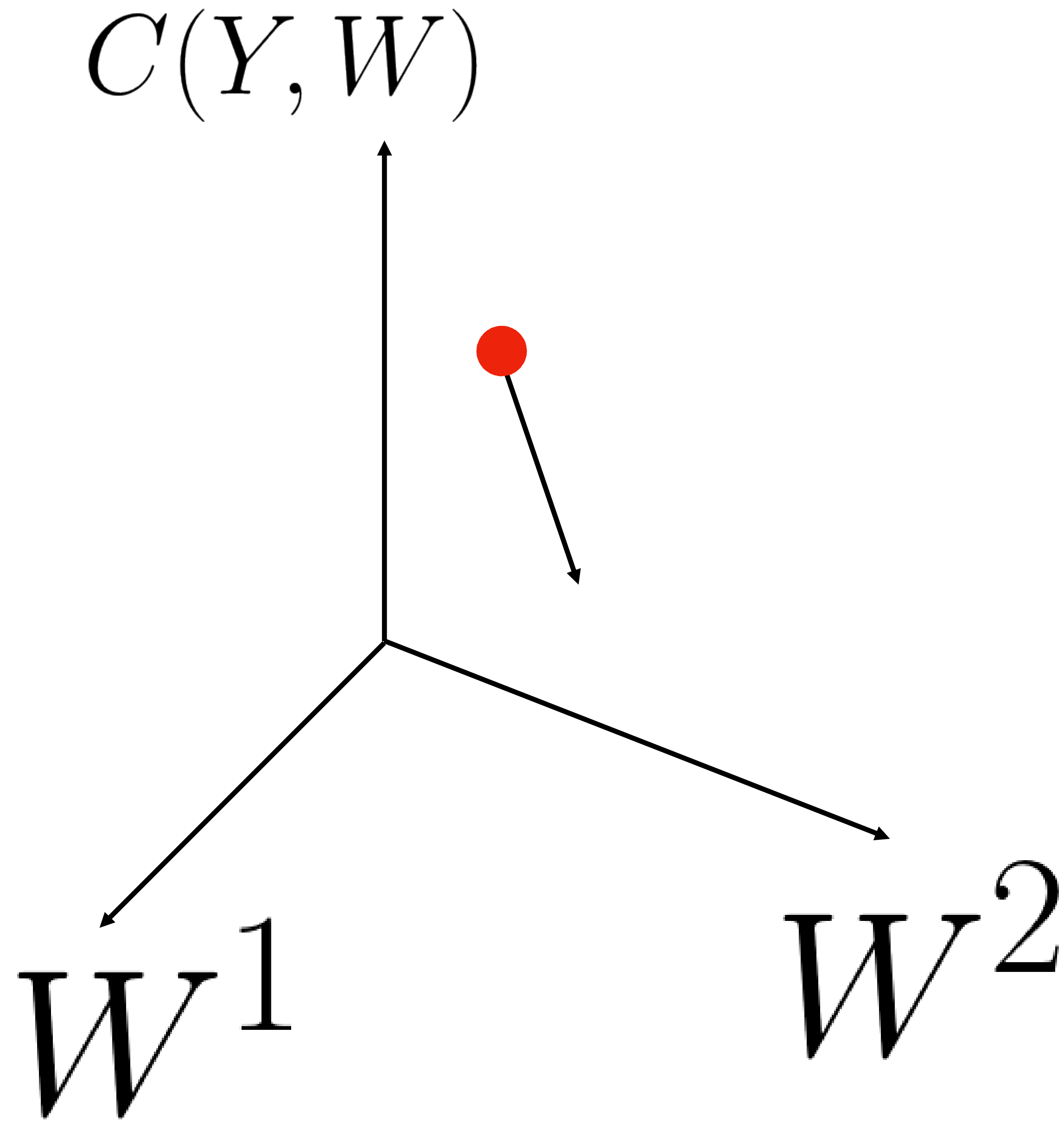
The gradient

$$\begin{bmatrix} \mathbf{3} & \mathbf{5} \\ \mathbf{1} & \mathbf{6} \\ \mathbf{0} & \mathbf{8} \end{bmatrix}$$

$$\frac{\partial C(y_1, a_1^2)}{\partial W}$$

$$\frac{\partial C(y_2, a_2^2)}{\partial W}$$

$$\frac{\partial C(y_3, a_3^2)}{\partial W}$$



Caveat

This method does not converge as rapidly as methods which make use of the second derivatives, but it is much simpler and can easily be implemented by local computations in parallel hardware. It can be significantly improved, without sacrificing the simplicity and locality, by using an acceleration method in which the current gradient is used to modify the velocity of the point in weight space instead of its position

Any Questions?

Thank you for your attention

Resources

Code: **https://iamtrask.github.io/2015/07/12/basic-python-networ**

Backprop: **https://www.youtube.com/watch?v=GlcnxUlrtek&t=299s**

Backprop detailed: **https://www.youtube.com/watch?v=gl3lfL-g5mA&t=1592s**