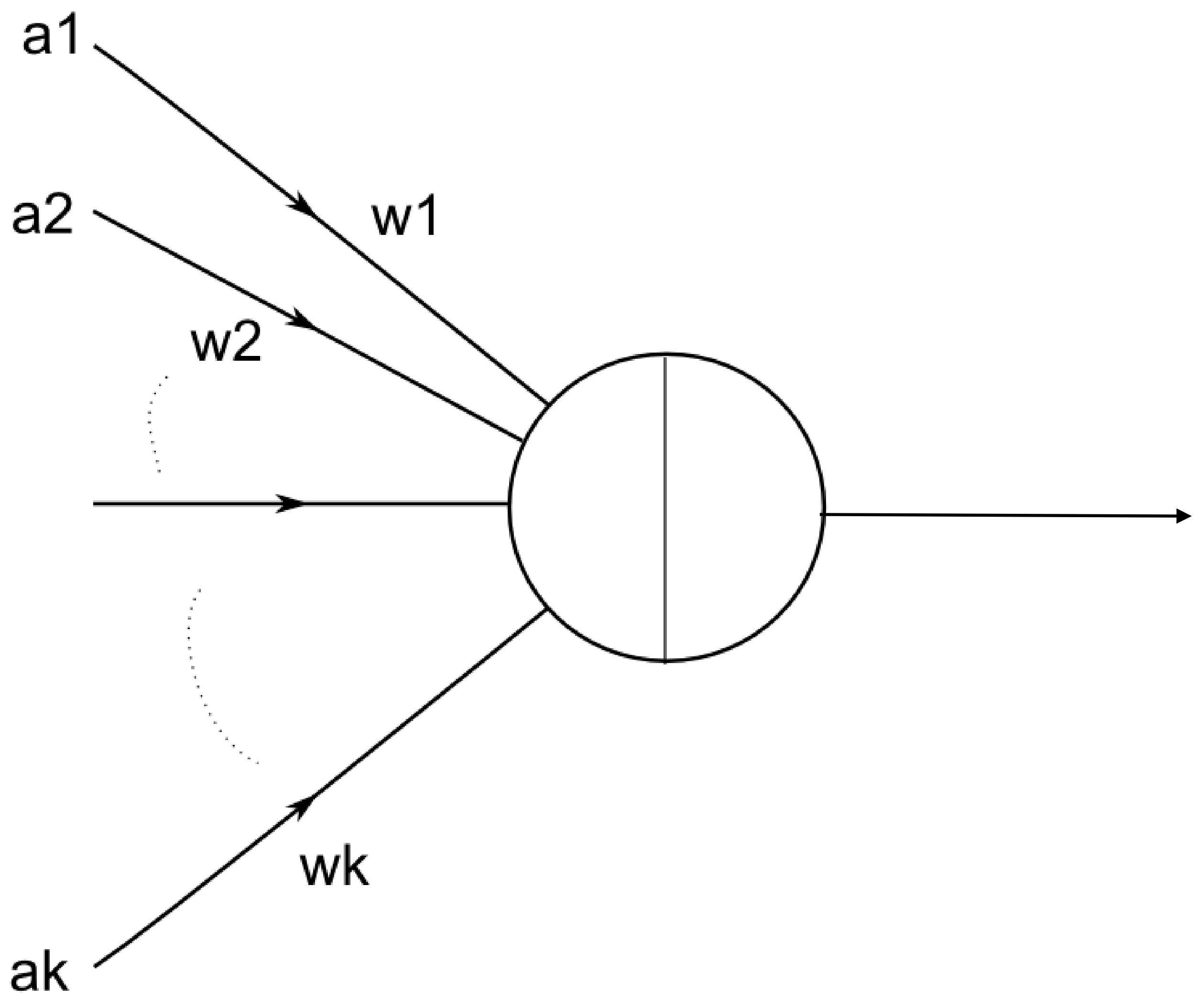
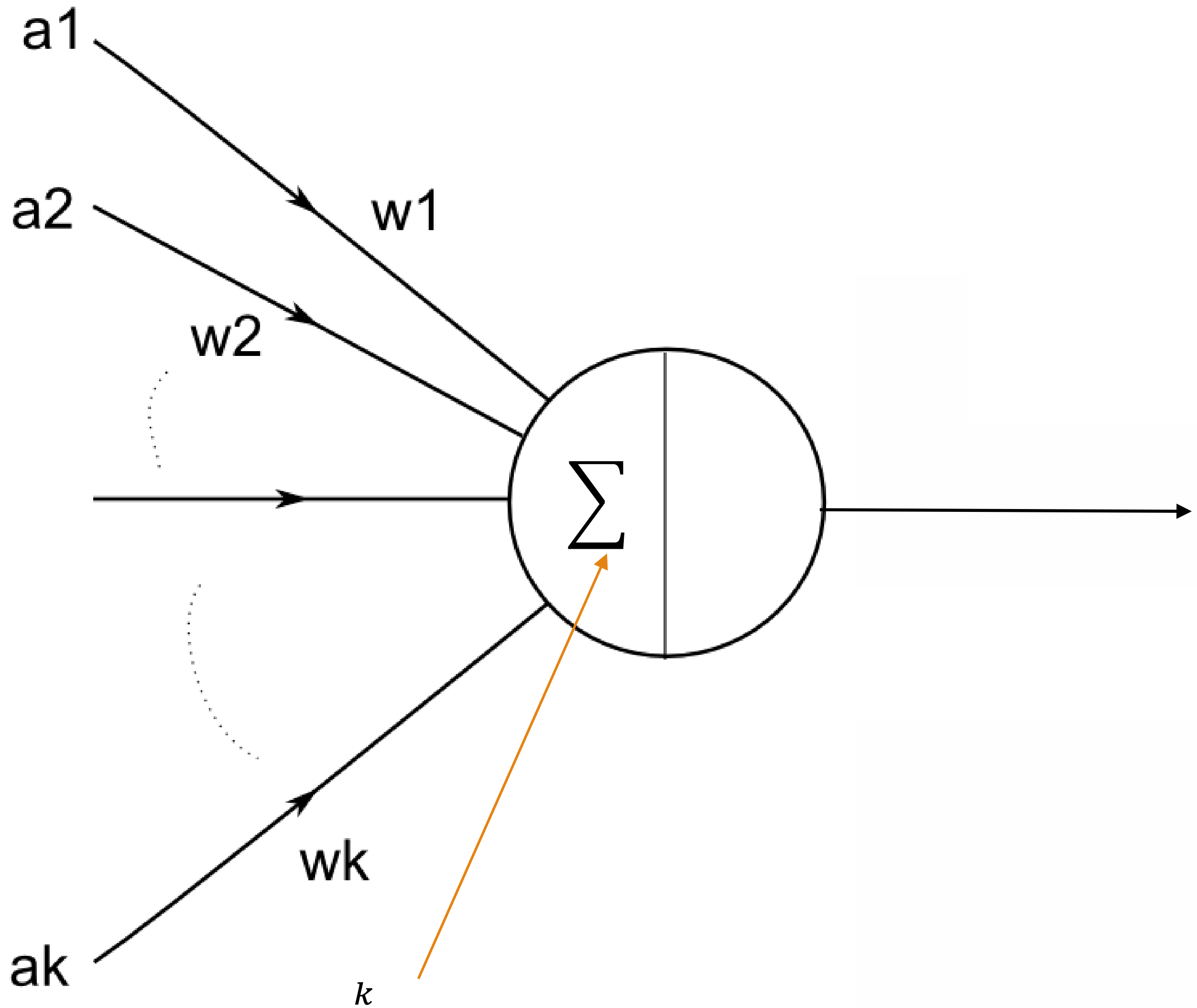
The Backpropagation Algorithm

MUSTAFA HAJIJ

Perceptron



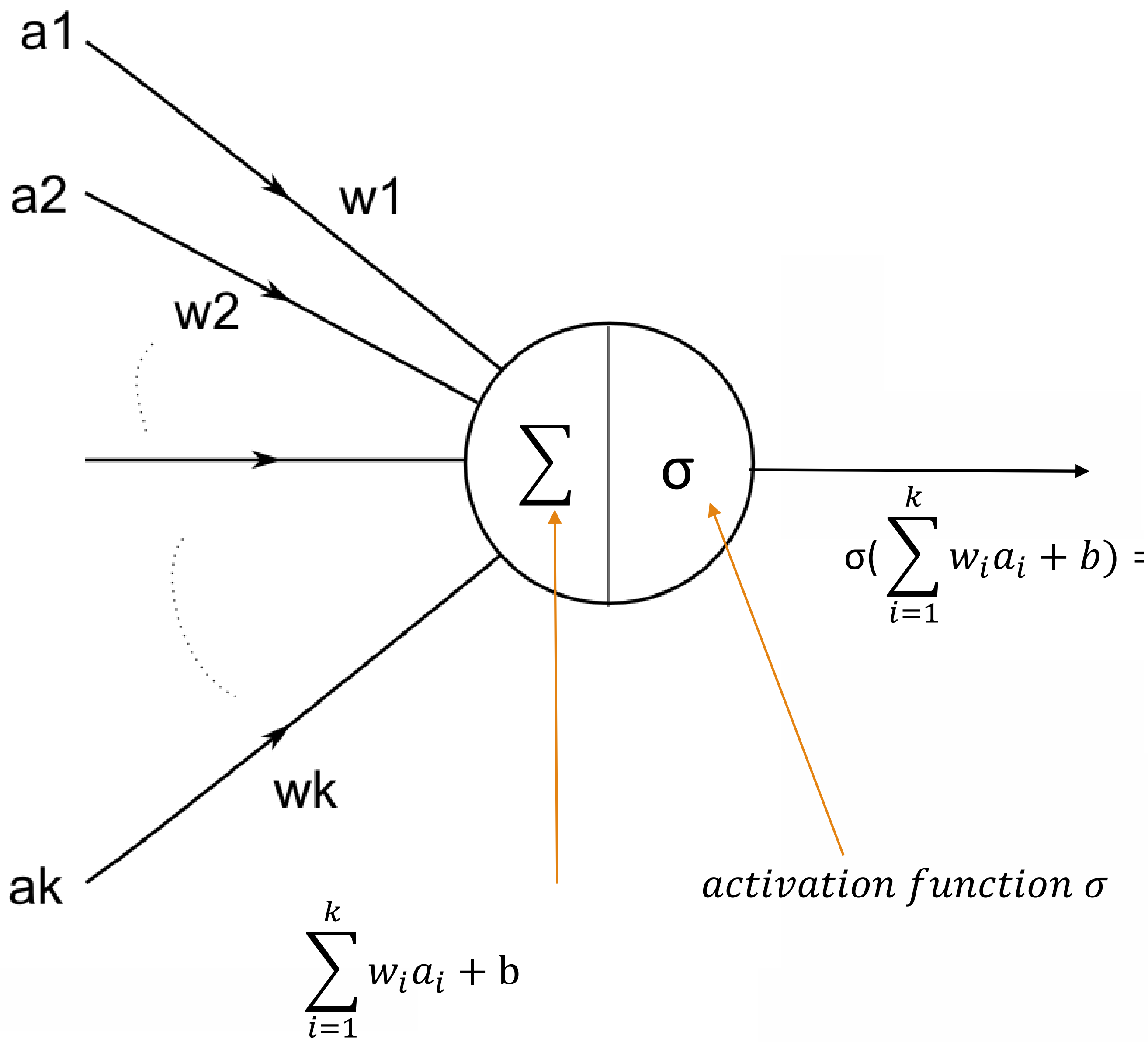
Perceptron



$$\sum_{i=1}^{k} w_i a_i + \mathrm{b}$$

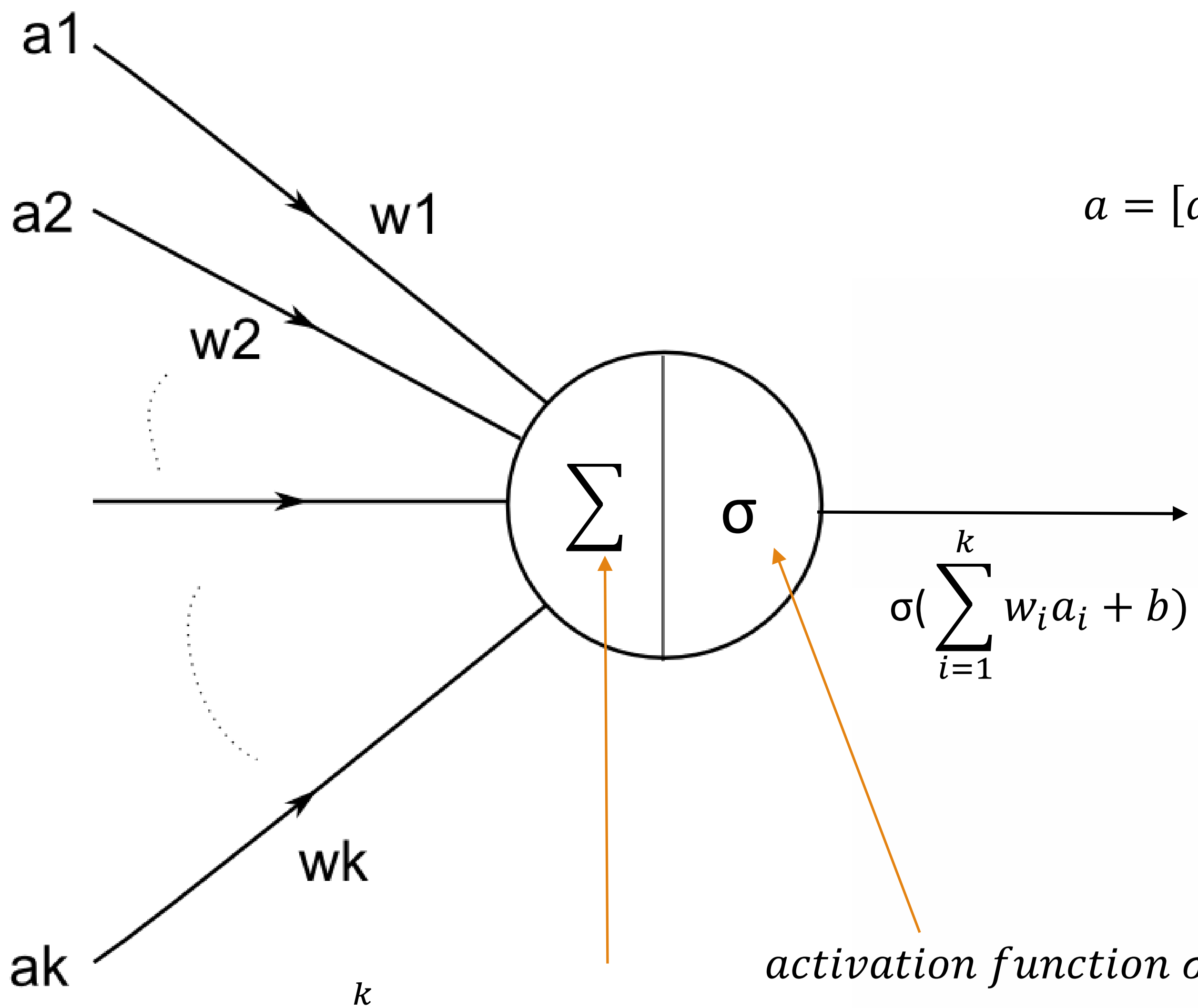
b is called a bias term.

Perceptron


a1
a2
ak
w1
w2
wk
$\sum$
σ
$\sigma(\sum_{i=1}^{k} w_i a_i + b)$ =
$\sum_{i=1}^{k} w_i a_i + \mathrm{b}$
$activation\ function\ \sigma$

Perceptron

$a = [a_1, \dots, a_k]^T$ $\quad w^T = [w_1, \dots, w_k]$



$$\sigma\left(\sum_{i=1}^{k} w_i a_i + b\right) = \sigma(w^T a + b)$$

activation function σ

$$\sum_{i=1}^{k} w_i a_i + \mathrm{b}$$

Training Perceptron

Given a collection of points $(x_1, y_1), \ldots., (x_n, y_n)$ where
x_i is a points in R^d and y_i is a label that takes values in $\{-1, +1\}$

Training Perceptron

Given a collection of points $(x_1, y_1), \ldots., (x_n, y_n)$ where
x_i is a points in R^d and y_i is a label that takes values in $\{-1, +1\}$

We want to choose $w = [w_1, \ldots w_d]$ and b such that the hyperplane determined by the $wx + b = 0$ separates the points x_i according to their labels. In other words, we want to choose the plane $wx + b = 0$ so that all points with positive sign on one side and all points with negative sign on the other side.

Training Perceptron

Given a collection of points $(x_1, y_1), \ldots., (x_n, y_n)$ where
x_i is a points in R^d and y_i is a label that takes values in $\{-1, +1\}$

We want to choose $w = [w_1, \ldots w_d]$ and b such that the hyperplane determined by the $wx + b = 0$ separates the points x_i according to their labels. In other words, we want to choose the plane $wx + b = 0$ so that all points with positive sign on one side and all points with negative sign on the other side.

As usual we have to define a cost function and a notion of error.

General Gradient Decent Algorithm

First we will need to know what gradient decent means.

Suppose that we are given a differentiable function $f(w_1, \ldots, w_d)$

General Gradient Decent Algorithm

First we will need to know what gradient decent means.

Suppose that we are given a differentiable function $f(w_1, \dots, w_d)$

Want to find $w_1, \dots, w_d$ such that $f(w_1, \dots, w_d)$ is minimal

General Gradient Decent Algorithm

First we will need to know what gradient decent means.

Suppose that we are given a differentiable function $f(w_1, \ldots, w_d)$

Want to find $w_1, \ldots, w_d$ such that $f(w_1, \ldots, w_d)$ is minimal. Gradient decent gives a way to find a local minimal for f

General Gradient Decent Algorithm

First we will need to know what gradient decent means.

Suppose that we are given a differentiable function $f(w_1, \ldots, w_d)$

Want to find $w_1, \ldots, w_d$ such that $f(w_1, \ldots, w_d)$ is minimal. Gradient decent gives a way to find a local minimal for f

Outline :

(1) Initiate $w_1, \ldots, w_d$ randomly
(2) keep changing $w_1, \ldots, w_d$ until hopefully $f(w_1, \ldots, w_d)$ is minimal

General Gradient Decent Algorithm

First we will need to know what gradient decent means.

Suppose that we are given a differentiable function $f(w_1, \dots, w_d)$

Want to find $w_1, \dots, w_d$ such that $f(w_1, \dots, w_d)$ is minimal. Gradient decent gives a way to find a local minimal for f

Outline :

(1) Initiate $w_1, \dots, w_d$ randomly
(2) keep changing $w_1, \dots, w_d$ until hopefully $f(w_1, \dots, w_d)$ is minimal

But how exactly do we change $w_1, \dots, w_d$?

General Gradient Decent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.

General Gradient Decent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.

(1) Initiate $w_1, \ldots, w_d$ randomly

General Gradient Decent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.

(1) Initiate $w_1, \ldots, w_d$ randomly
(2) Repeat until convergence :
    (1) For every i in range(1,d):
        $(1) w_i \coloneqq w_i - q \frac{\partial f}{\partial w_i}$ (here we do simultaneous update for the parameters w_i)

General Gradient Decent Algorithm

Key idea : gradient of f goes in the direction at which f maximally change.

(1) Initiate $w_1, \dots, w_d$ randomly
(2) Repeat until convergence :
    (1) For every i in range(1,d):
        $(1) w_i := w_i - q \frac{\partial f}{\partial w_i}$ (here we do simultaneous update for the parameters w_i)

Gradient decent asserts that the values of the function f when we update as described above are non-increasing :

$$f(old\ w_i) \geq f(new\ w_i)$$

Training Perceptron

Now back to training a perceptron. We need some facts.

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points $x_1 \; and \; x_2$ on the plane $w^T x + b = 0$, we have

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$w^T x_1 + b$=$w^T x_2 + b$=0

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$w^T x_1 + b$=$w^T x_2 + b$=0

Hence

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$$w^T x_1 + b = w^T x_2 + b = 0$$

Hence

$$w^T (x_1 - x_2) = 0$$

Hence the vector w^T is orthogonal to $(x_1 - x_2)$

Training Perceptron

Now back to training a perceptron. We need some facts.

For any points x_1 and x_2 on the plane $w^T x + b = 0$, we have

$w^T x_1 + b$=$w^T x_2 + b$=0

Hence

$w^T (x_1 - x_2)$=0

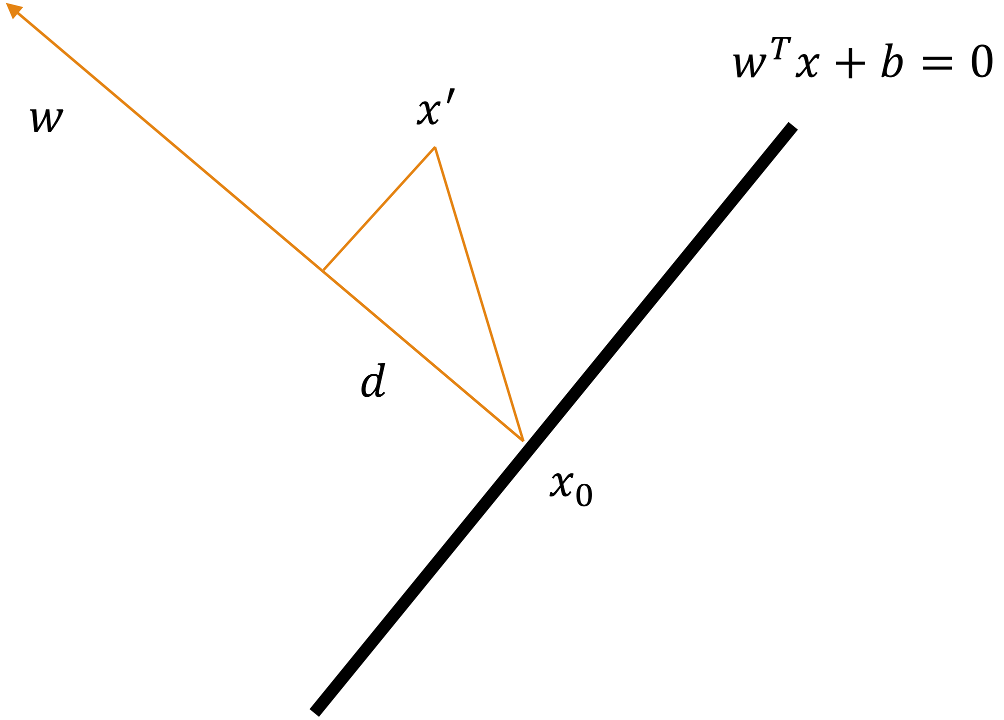
Hence the vector w^T is orthogonal to $(x_1 - x_2)$

Moreover, for any x_0 on the plane $w^T x + b = 0$ we have

$$b = -w^T x_0$$

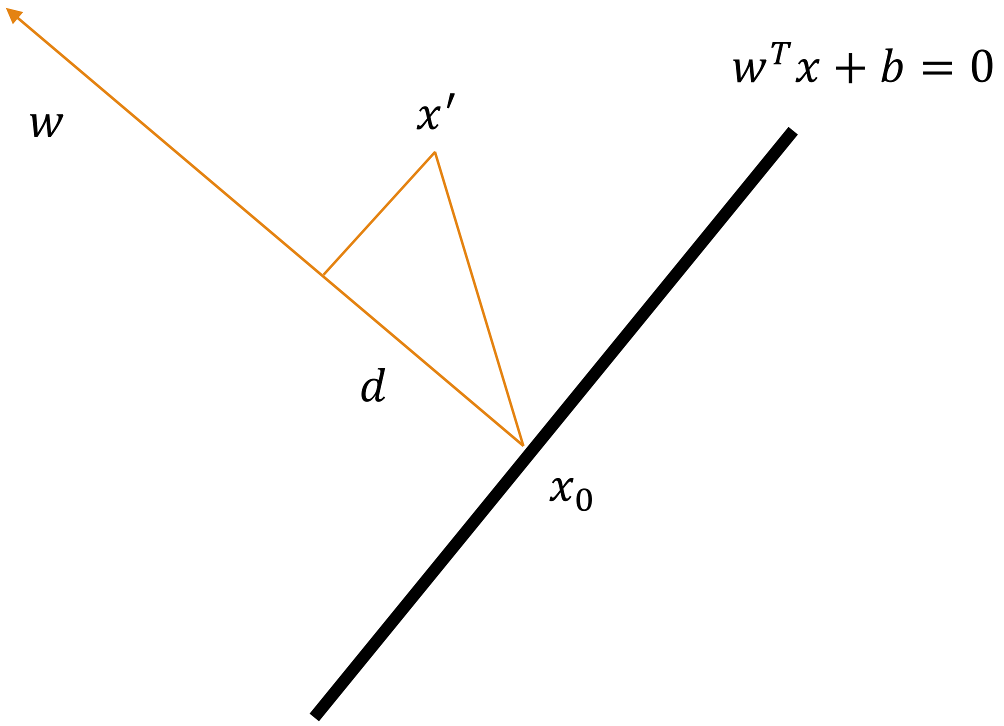
Training Perceptron

$$d = w^T(x' - x_0) = w^T x' - w^T \mathrm{x}_0 = w^T x' + b$$



Training Perceptron

$$d = w^T(x' - x_0) = w^T x' - w^T \mathrm{x}_0 = w^T x' + b$$



So if we have a point and we want to see where it is located on with respect to the plan, then all we have to do is to plug it in the equation of the plane.

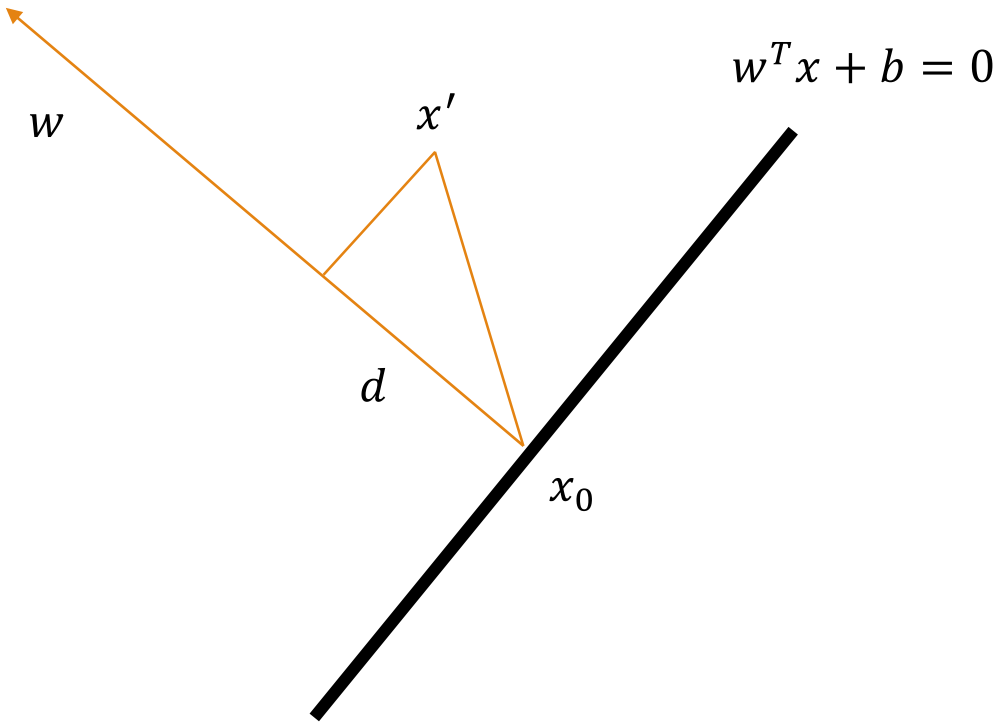
Training Perceptron

Write

$$d_i = y_i(w^T x_i + b)$$

Where (x_i, y_i) is a training example

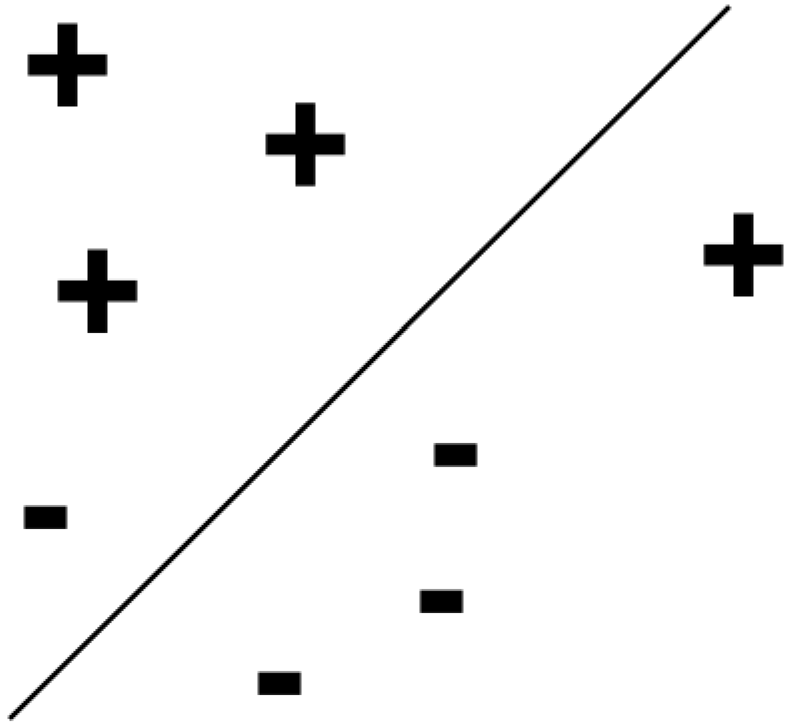
Note that $d_i \geq 0$



Training Perceptron

Define

$$error(w, b) \coloneqq -\sum_M y_i(w^T x_i + b_0\ )$$

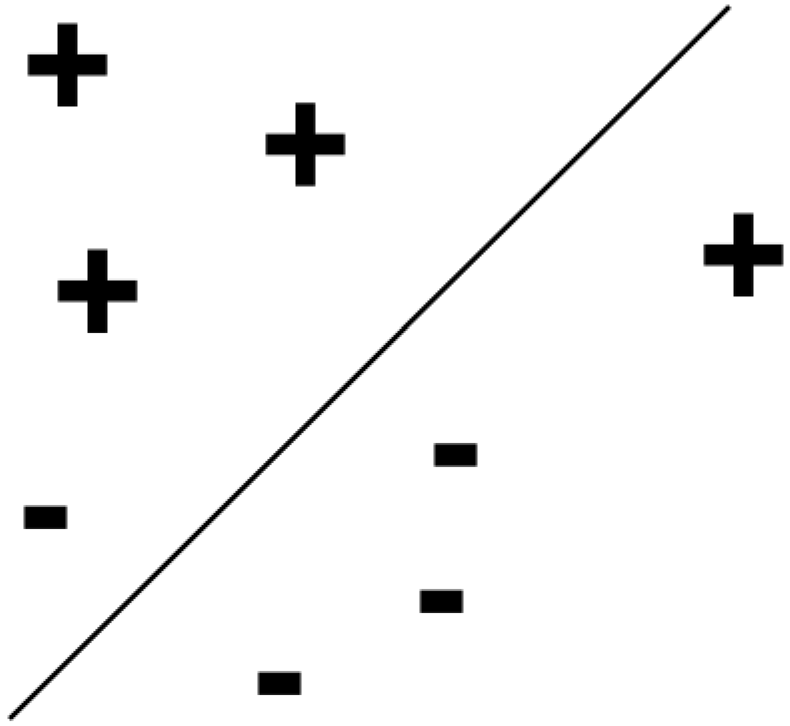


Training Perceptron

Define

$$error(w, b) := -\sum_M y_i(w^T x_i + b_0)$$

Where M is the set of misclassified points



Training Perceptron

Define

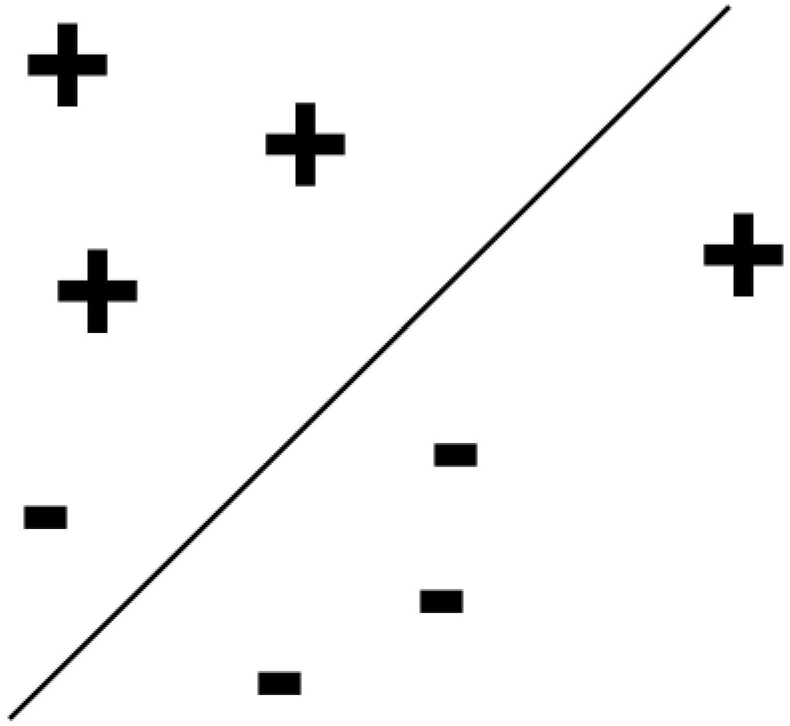
$$error(w, b) \coloneqq -\sum_M y_i(w^T x_i + b_0\ )$$

Where M is the set of misclassified points

We want to apply gradient decent on the function $error(w, b)$

$$\frac{\partial\ error(w, b)}{\partial\ w} = \sum_M y_i x_i$$

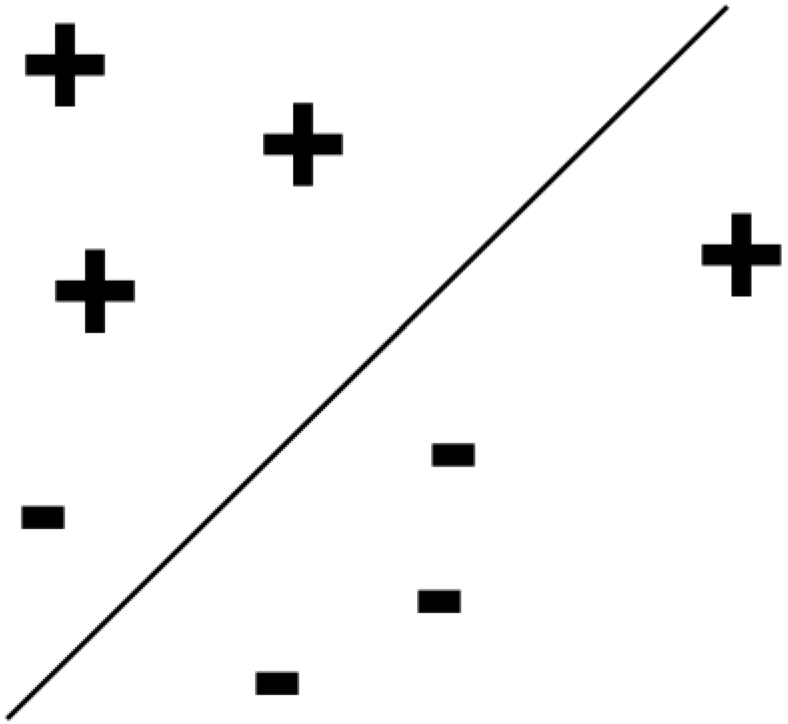
$$\frac{\partial\ error(w, b)}{\partial\ b} = \sum_M y_i$$



Training Perceptron

To train a perceptron

(1) Assign the weights w randomly

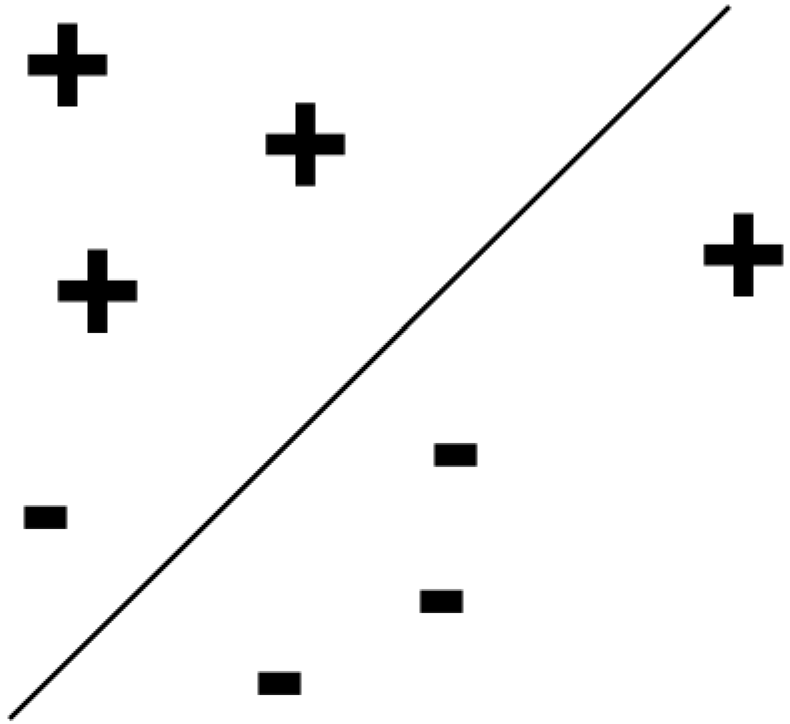


$$error(w, b) \coloneqq -\sum_M y_i(w^T x_i + b_0)$$

Training Perceptron

To train a perceptron

1. Assign the weights w randomly
2. Repeat until convergence



$$error(w, b) := -\sum_M y_i(w^T x_i + b_0)$$

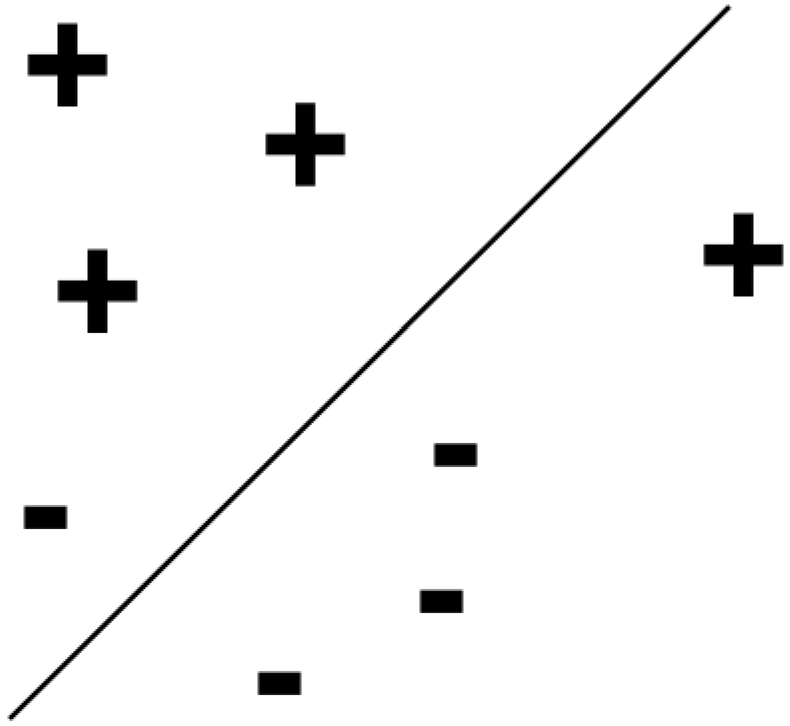
Training Perceptron

To train a perceptron

(1) Assign the weights w randomly
(2) Repeat until convergence

$$w_{new} := w_{old} - q \frac{\partial\, error(w,b)}{\partial\, w}$$

$$b_{new} := b_{old} - q \frac{\partial\, error(w,b)}{\partial\, b}$$



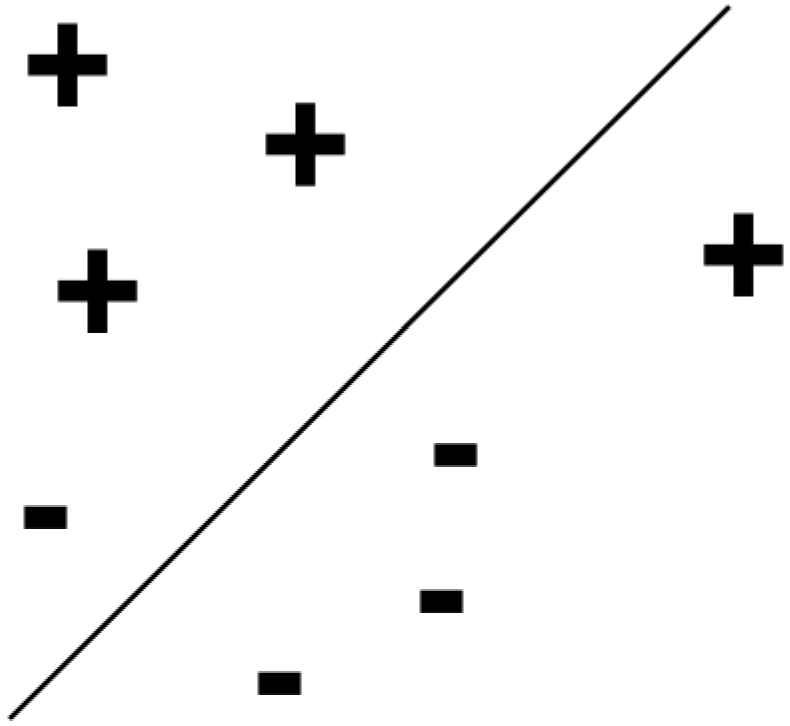
$$error(w,b) := -\sum_M y_i (w^T x_i + b_0)$$

Training Perceptron

To train a perceptron

(1) Assign the weights w randomly
(2) Repeat until convergence

$$w_{new} := w_{old} - q \frac{\partial \, error(w,b)}{\partial \, w}$$
$$b_{new} := b_{old} - q \frac{\partial \, error(w,b)}{\partial \, b}$$



if the examples are linearly separable then the above model classifies the points

$$error(w,b) := -\sum_M y_i(w^T x_i + b_0)$$

Training Perceptron

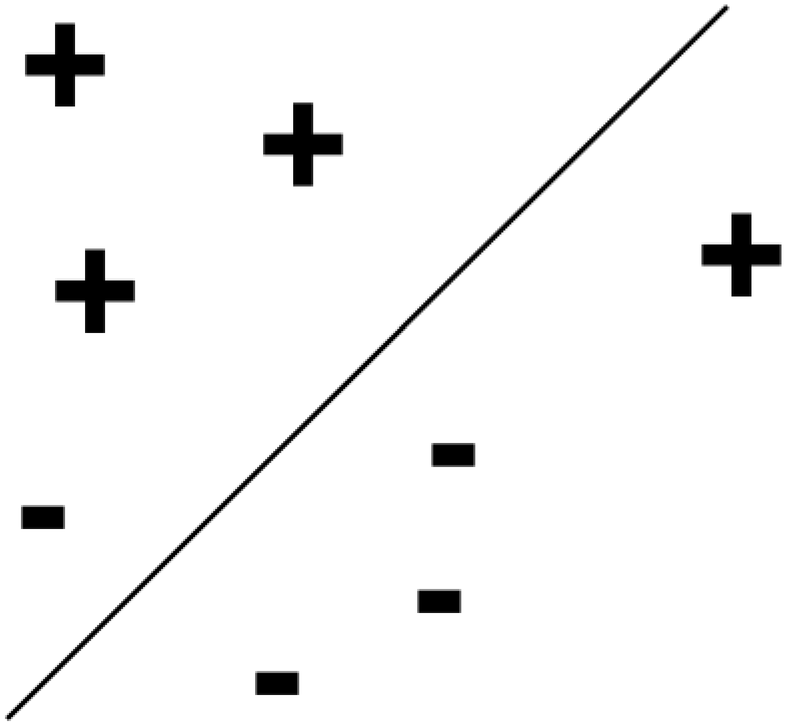
To train a perceptron

(1) Assign the weights w randomly
(2) Repeat until convergence

$$w_{new} := w_{old} - q y_i x_i$$

$$b_{new} := b_{old} - q x_i$$

Stochastic gradient decent



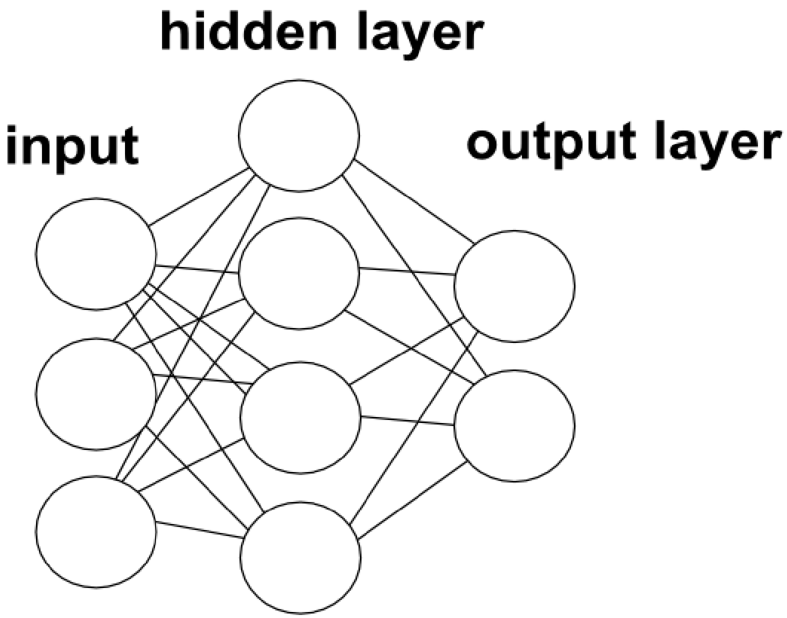
if the examples are linearly separable then the above model classifies the points

$$error(w, b) \coloneqq -\sum_M y_i (w^T x_i + b_0)$$

Neural Network

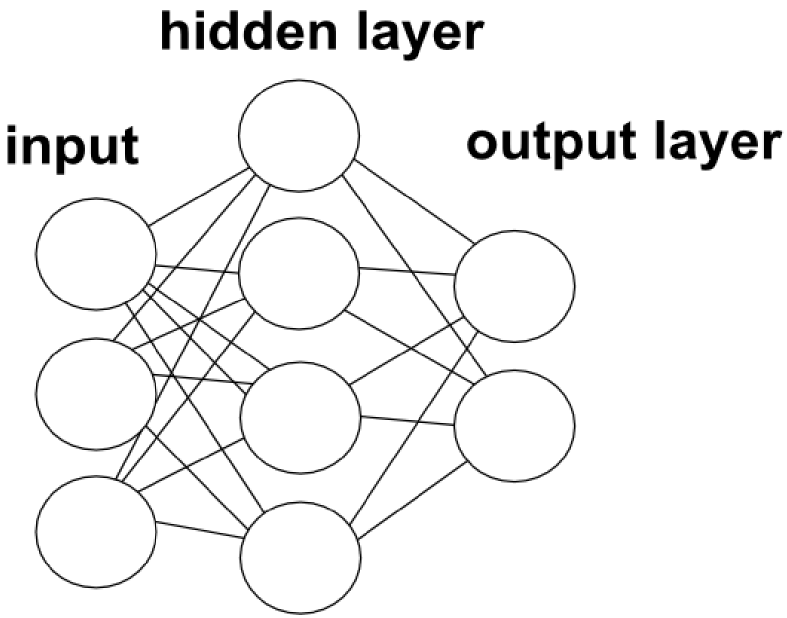
Perceptron is the building block of a neural network. Clearly there are some data that cannot be classified using a single perceptron.

The idea of neural network is to stack together multiple layers of perceptrons in order to be able to learn more complicated functions



Neural Network

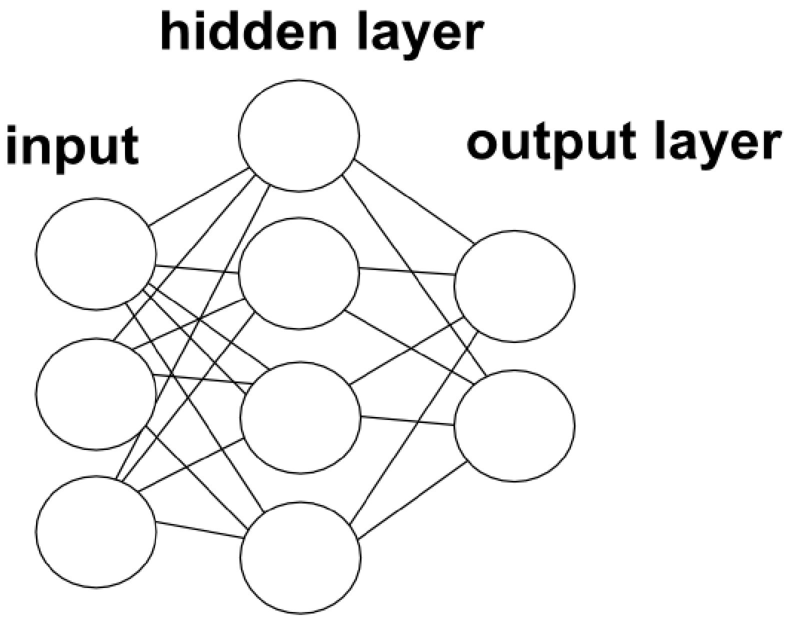
Mathematically, a neural network is a function f that takes x as input and produces an output y=f(x)



Neural Network

Mathematically, a neural network is a function f that takes x as input and produces an output y=f(x)

The training of a neural network means to tune the weights in all layers so that the output of the function f matches the label of x. The process of updating the weights for a feedforward neural network is called *backpropagation*.



Feedforward Neural Network

How do we compute a feedforward neural network on an input x ?

Feedforward Neural Network

Start with an input $x = a^{(0)}$. In the picture, this is represented by the first layer of nodes. We will call this layer 0.

$$x = a^{(0)}$$

Feedforward Neural Network

We apply the weight $W^{(1)}$ coming from the edges between layer 0 and layer 1 and add the biases and then apply the Activation function on the resulting vector coordinate-wise.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})}$$

$W^{(1)}$: Edges between layer 0 and layer 1
$a^{(0)}$: input
$b^{(1)}$: biases applied to layer 1
σ : activation function

Feedforward Neural Network

We will call the output of this computation $a^{(1)}$. This is now represented by the nodes in layer 1.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}}$$

$W^{(1)}$: Edges between layer 0 and layer 1

$a^{(0)}$: input

$b^{(1)}$: biases applied to layer 1

σ : activation function

Feedforward Neural Network

Repeat.

$$x = a^{(0)} \longrightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}} \boxed{\sigma(W^{(2)}a^{(1)}+b^{(2)})} \xrightarrow{a^{(2)}}$$

$W^{(2)}$: Edges between layer 1 and layer 2

$a^{(1)}$: input from layer 1

$b^{(2)}$: biases applied to layer 2

σ : activation function

Feedforward Neural Network

Until you finish the neural network and get the final output.

$$x = a^{(0)} \rightarrow \boxed{\sigma(W^{(1)}a^{(0)}+b^{(1)})} \xrightarrow{a^{(1)}} \boxed{\sigma(W^{(2)}a^{(1)}+b^{(2)})} \xrightarrow{a^{(2)}} \boxed{\sigma(W^{(3)}a^{(2)}+b^{(3)})} \dashrightarrow$$

$$\dashrightarrow \boxed{\sigma(W^{(n)}a^{(n-1)}+b^{(n)})} \rightarrow a^{(n)}=y$$

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify data.

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

How do we adjust the weights ? As before, we define a notion of cost function (which will be a function with respect to all weights in the neural network) and then we try to minimize that function using the gradient decent algorithm

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

How do we adjust the weights ? As before, we define a notion of cost function (which will be a function with respect to all weights in the neural network) and then we try to minimize that function using the gradient decent algorithm

Training a Neural Network

Now suppose that we are given a binary labeled data as before and we want to use neural network to classify this data.

The advantages of the neural network over the perceptron is that neural network would be able to define a much more complicated decision boundary which ultimately give us more ability to classify data.

To start working with neural network we initiate the weight of the network randomly and then we test if the output that we obtain from the network matches the label of the input. Most likely, the output obtained this way will not be useful with the initial random weight.

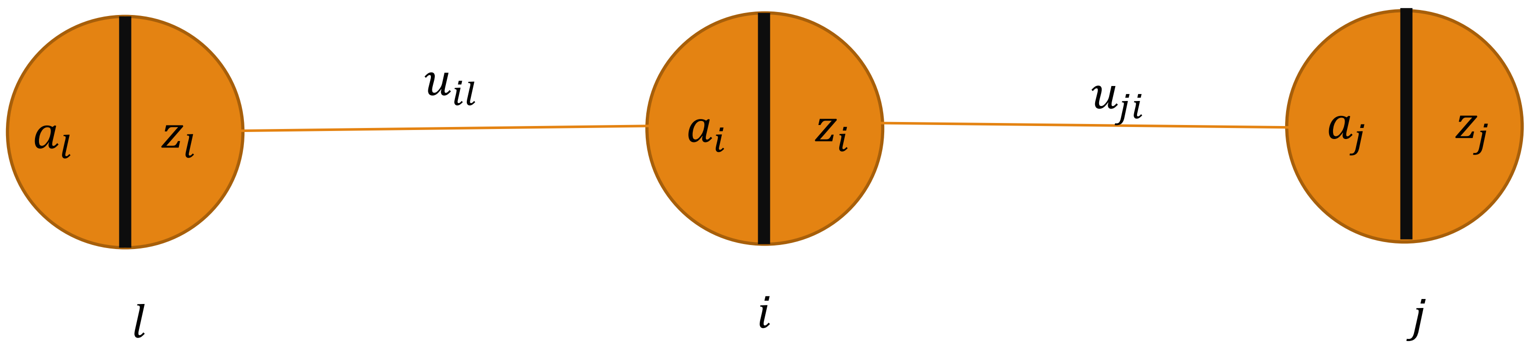
We need to adjust the weights of the network so that it classifies the data correctly. This is what we mean by *training the network*.

How do we adjust the weights ? As before, we define a notion of cost function (which will be a function with respect to all weights in the neural network) and then we try to minimize that function using the gradient decent algorithm

The process of updating the weights for a feedforward neural network is called *backpropagation*.
Which we will present next.

Backpropagation

To understand backpropagation we will consider the following simplified neural network:

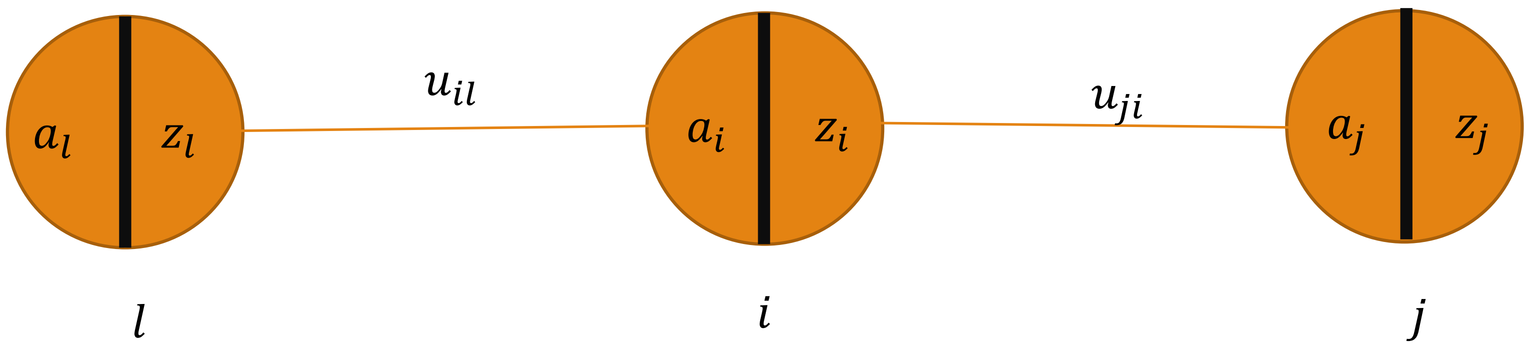


$z_i = \sigma(a_i)$ where σ is smooth function

$$a_i = \sum_l u_{il} z_l$$

Backpropagation

To understand backpropagation we will consider the following simplified neural network:

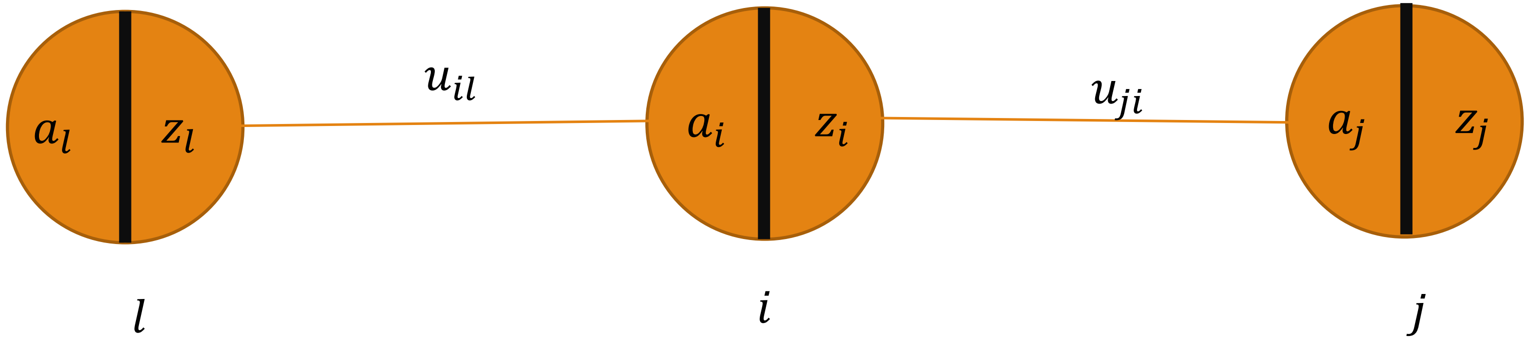


$z_i = \sigma(a_i)$ where σ is smooth function

$$a_i = \sum_l u_{il} z_l$$

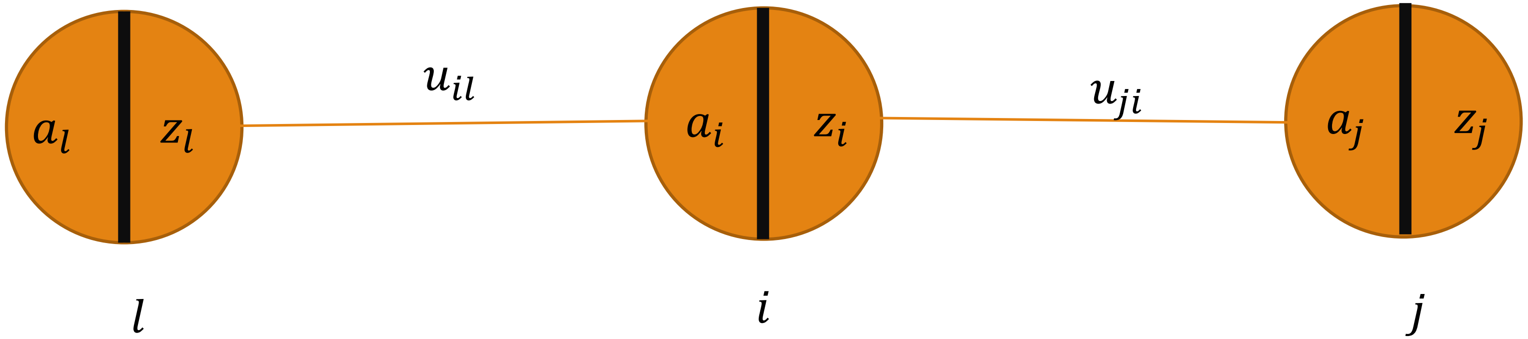
Suppose that we have $\{(x_i, y_i)\}^n{}_{i=1}$. We want to train the neural network so that it classifies the data. Suppose we have only have a single output and denote that by y'.
In other words, y' is the answer generated by the network and y is the desired output. For simplicity suppose that the cost function is $C_0 = (y - y')^2$.

Backpropagation



Goal : compute $\frac{\partial\, C_0}{\partial\, u_{il}}$

Backpropagation



Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

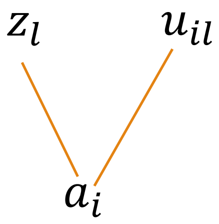
Recall :

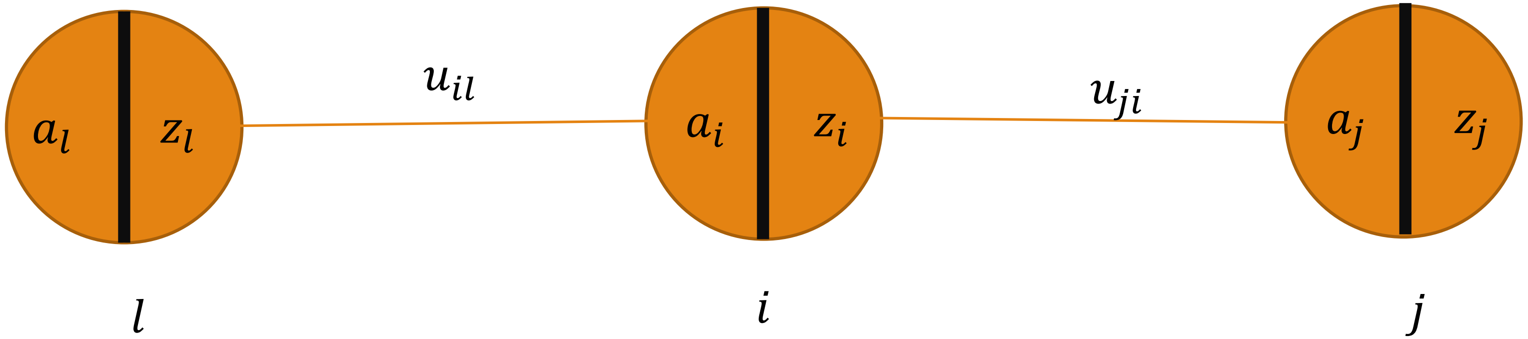
$z_i = \sigma(a_i)$

$a_i = \sum_l u_{il} z_l$

$C_0 = (y - y')^2.$

Backpropagation





Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

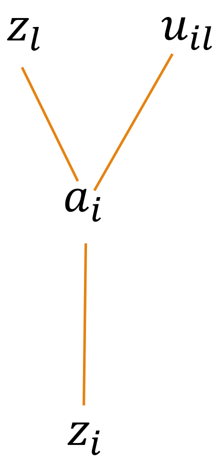
Recall :

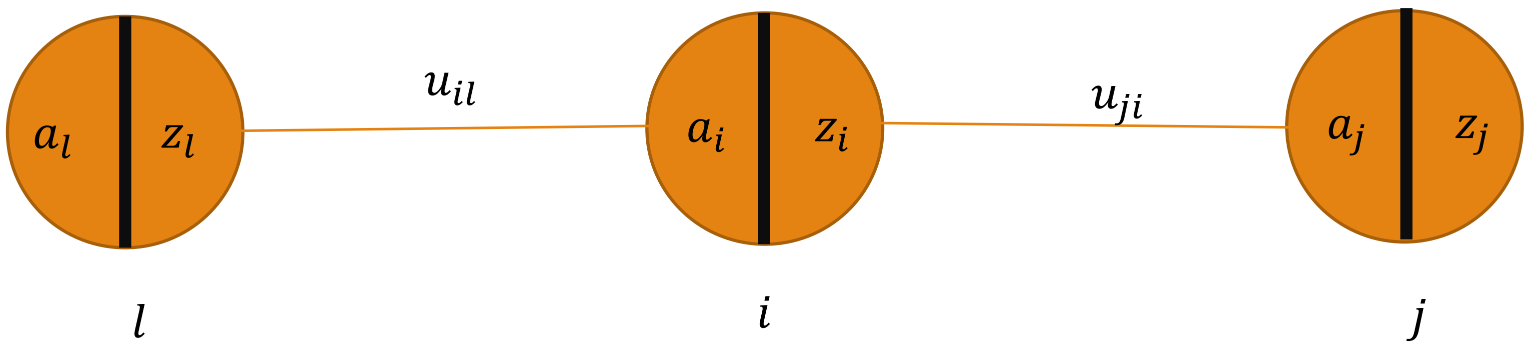
$z_i = \sigma(a_i)$

$a_i = \sum_l u_{il} z_l$

$C_0 = (y - y')^2.$

Backpropagation





Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

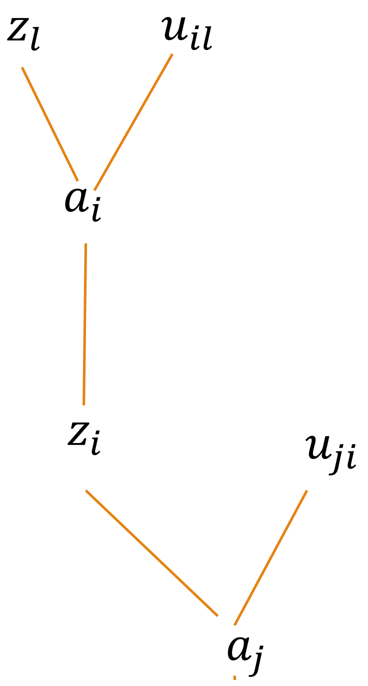
Recall :

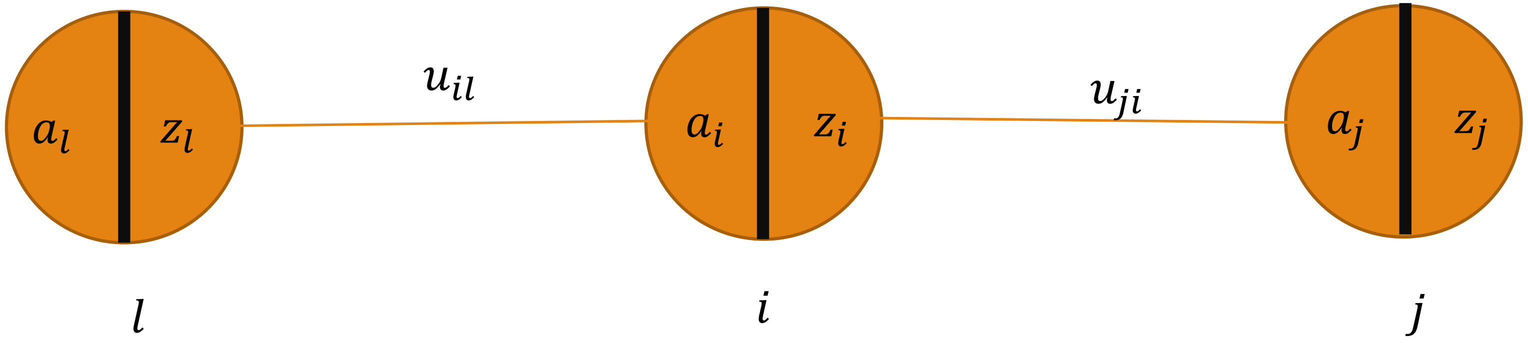
$z_i = \sigma(a_i)$

$a_i = \sum_l u_{il} z_l$

$C_0 = (y - y')^2.$

Backpropagation





Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

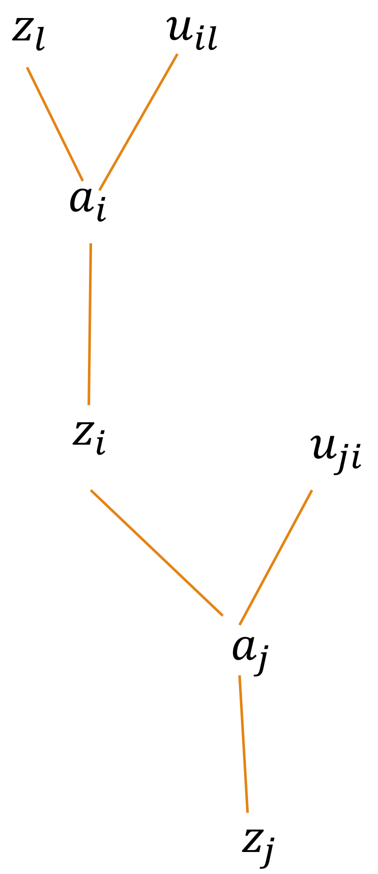
Recall :

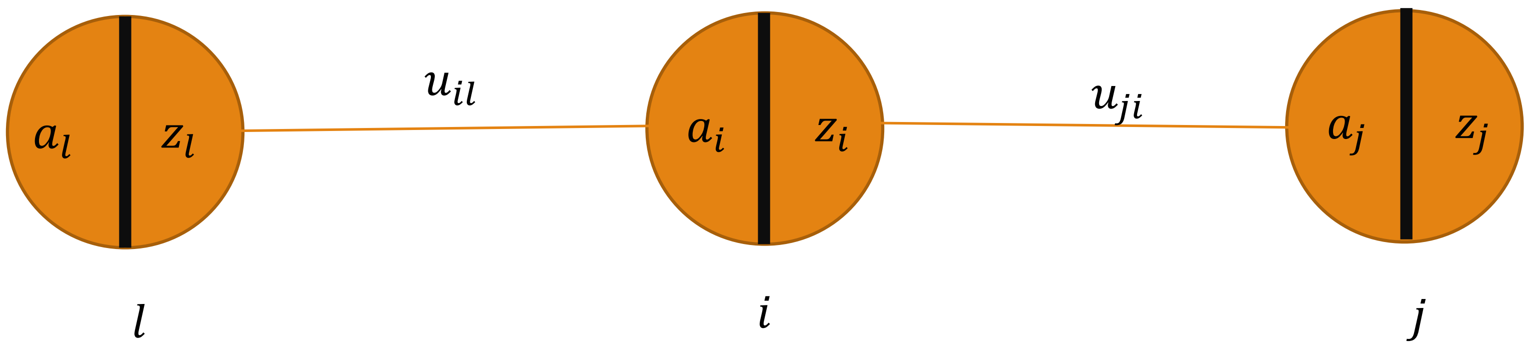
$z_i = \sigma(a_i)$

$a_i = \sum_l u_{il} z_l$

$C_0 = (y - y')^2.$

Backpropagation





Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

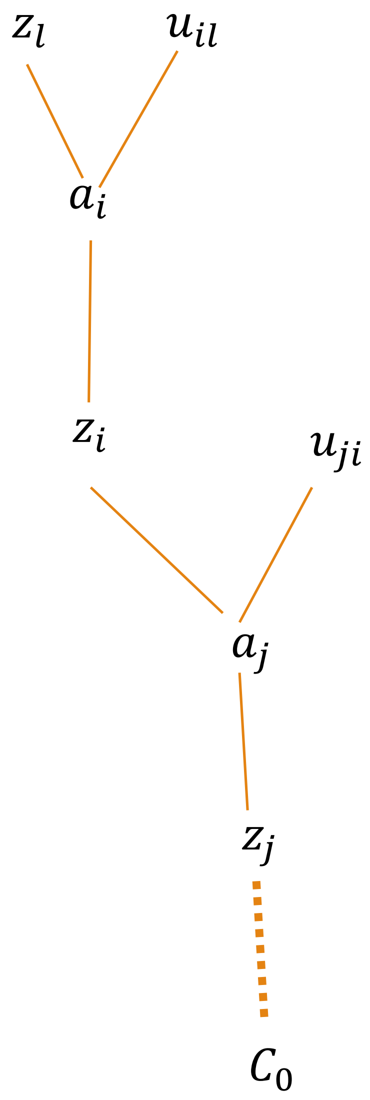
Recall :

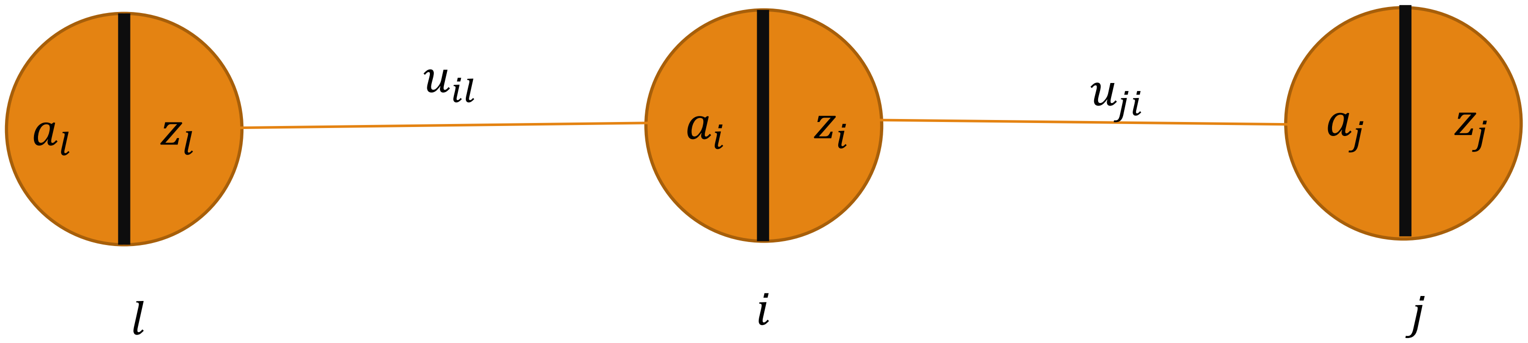
$z_i = \sigma(a_i)$

$a_i = \sum_l u_{il} z_l$

$C_0 = (y - y')^2.$

Backpropagation





Goal : compute $\dfrac{\partial C_0}{\partial u_{il}}$

Recall :

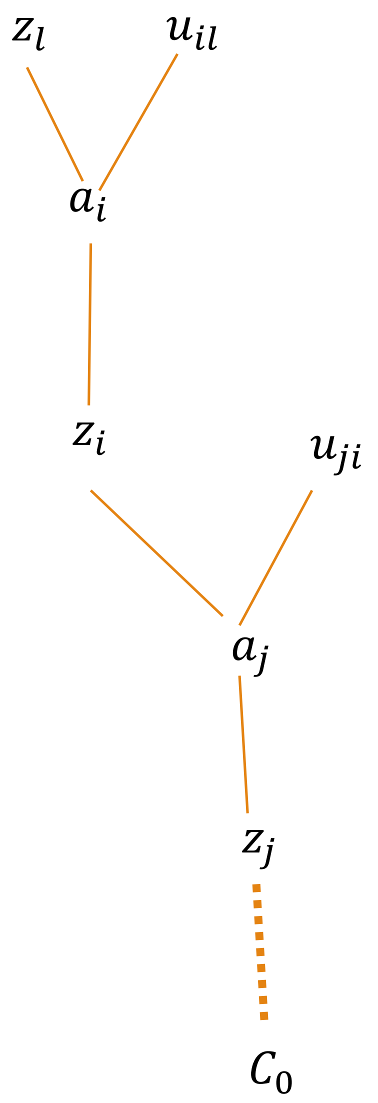
$z_i = \sigma(a_i)$

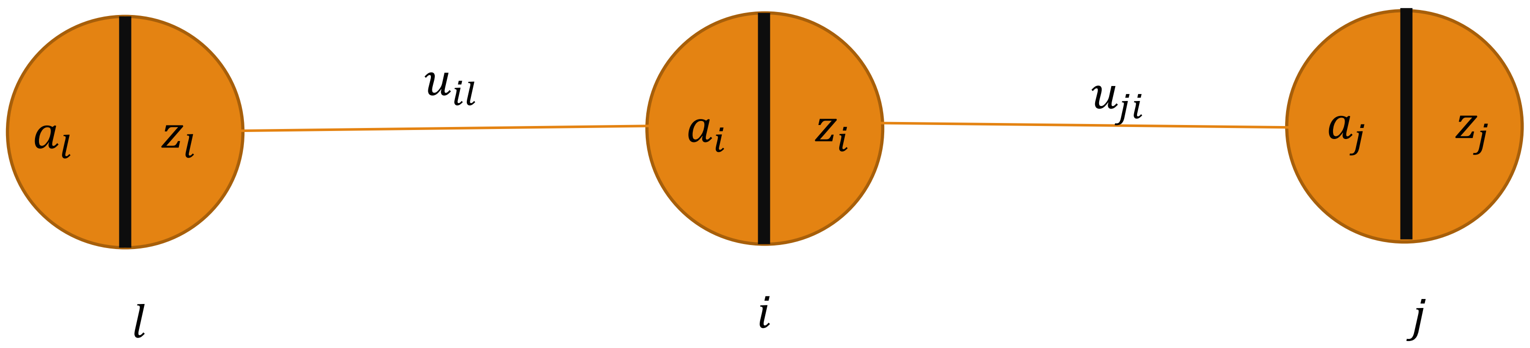
$a_i = \sum_l u_{il} z_l$

$C_0 = (y - y')^2.$

This explains the variable dependency.

Backpropagation





Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

We now can take the derivative of C_0 with respect to a specific weight u_{il}

Recall :

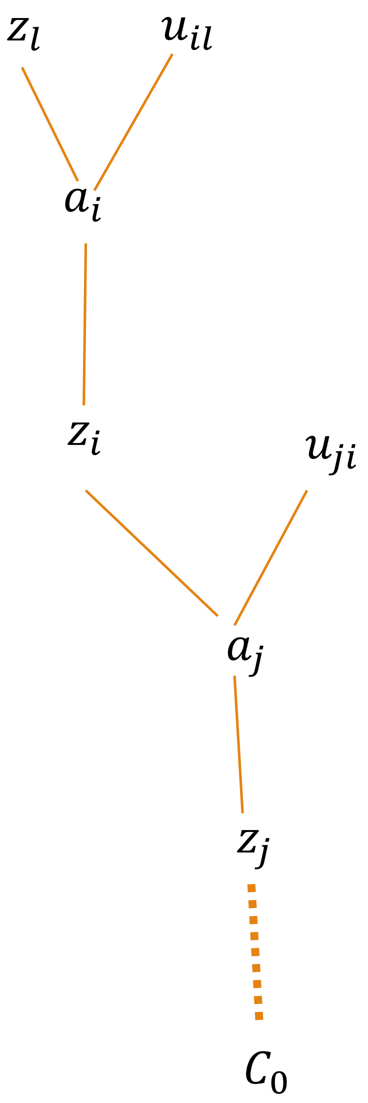
$z_i = \sigma(a_i)$

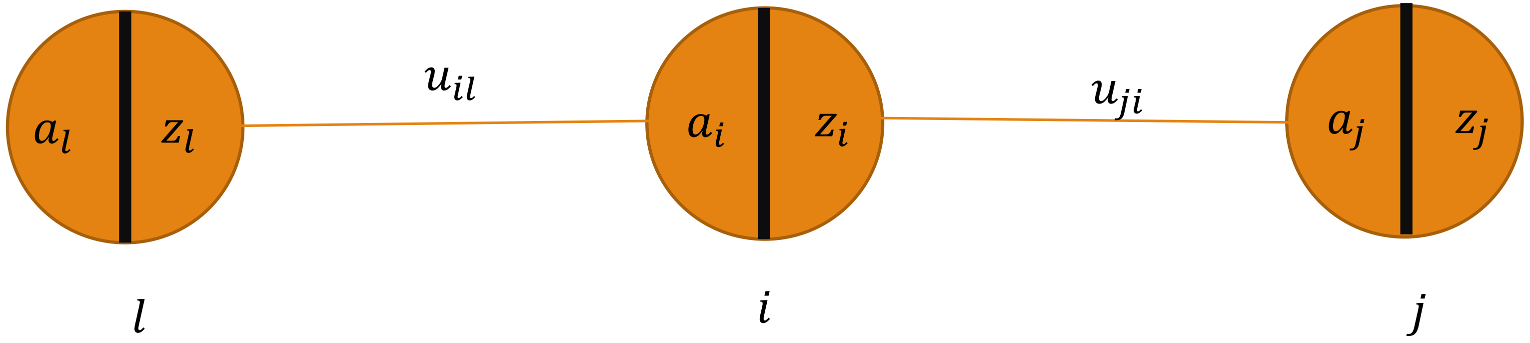
$a_i = \sum_l u_{il} z_l$

$C_0 = (y - y')^2.$

This explains the variable dependency.

Backpropagation





Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

$$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\frac{\partial\, a_i}{\partial\, u_{il}}$$ Chain rule

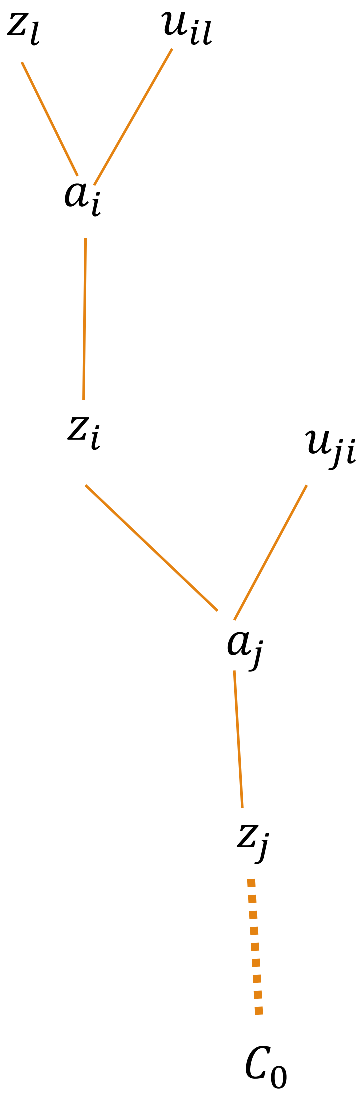
Recall :

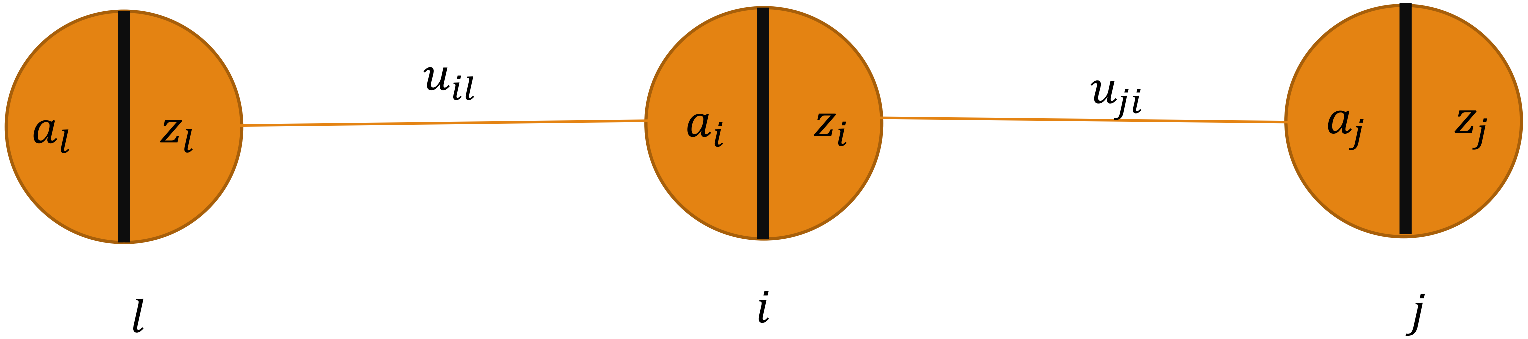
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





Goal : compute $\dfrac{\partial\, C_0}{\partial\, u_{il}}$

$$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\boxed{\frac{\partial\, a_i}{\partial\, u_{il}}}$$ Chain rule

$$\boxed{\frac{\partial\, a_i}{\partial\, u_{il}}} = z_l$$

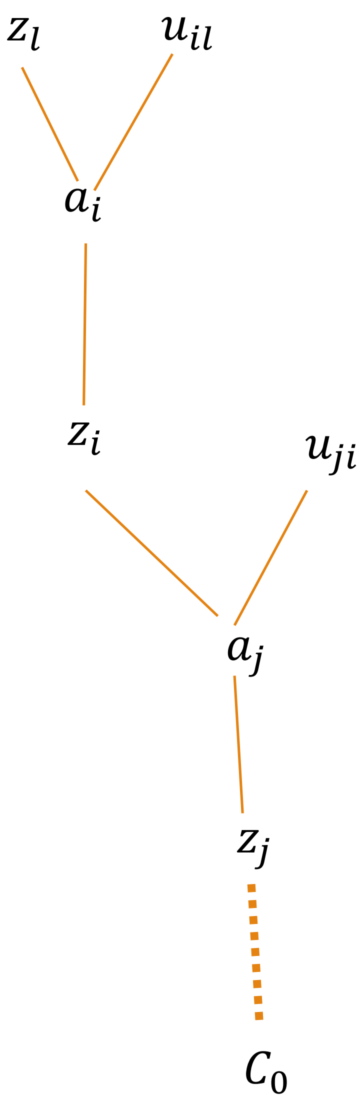
Recall :

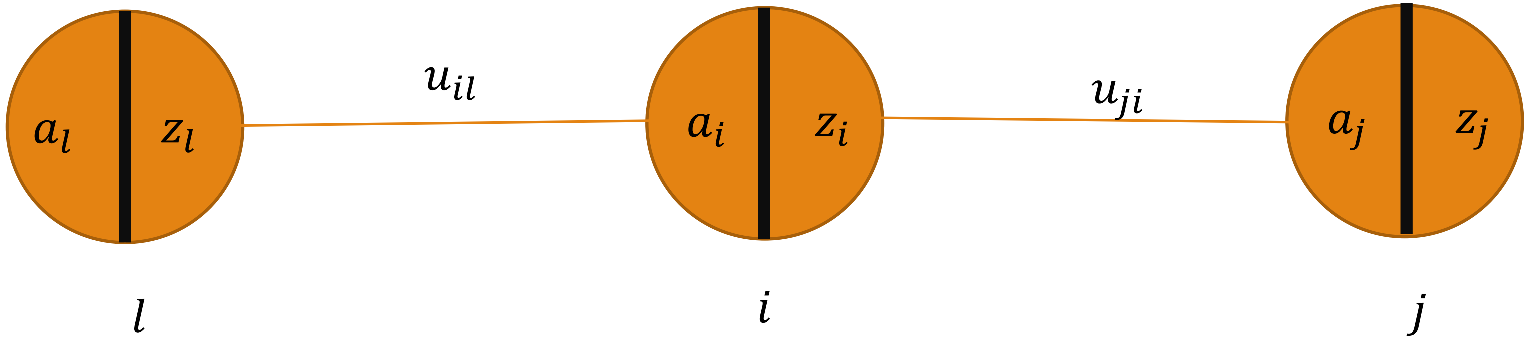
$z_i = \sigma(a_i)$

$$a_i = \sum_l u_{il} z_l$$

$C_0 = (y - y')^2.$

Backpropagation





Goal : compute $\frac{\partial C_0}{\partial u_{il}}$

$$\frac{\partial C_0}{\partial u_{il}} = \frac{\partial C_0}{\partial a_i}\boxed{\frac{\partial a_i}{\partial u_{il}}}$$

Chain rule

$$\boxed{\frac{\partial a_i}{\partial u_{il}}} = z_l$$

We do not know $\frac{\partial C_0}{\partial a_i}$ so we call it for now δ_i

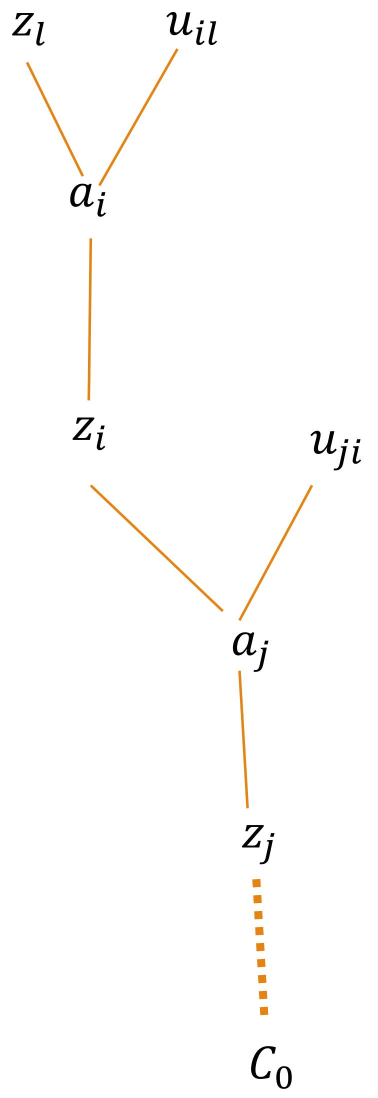
Recall :

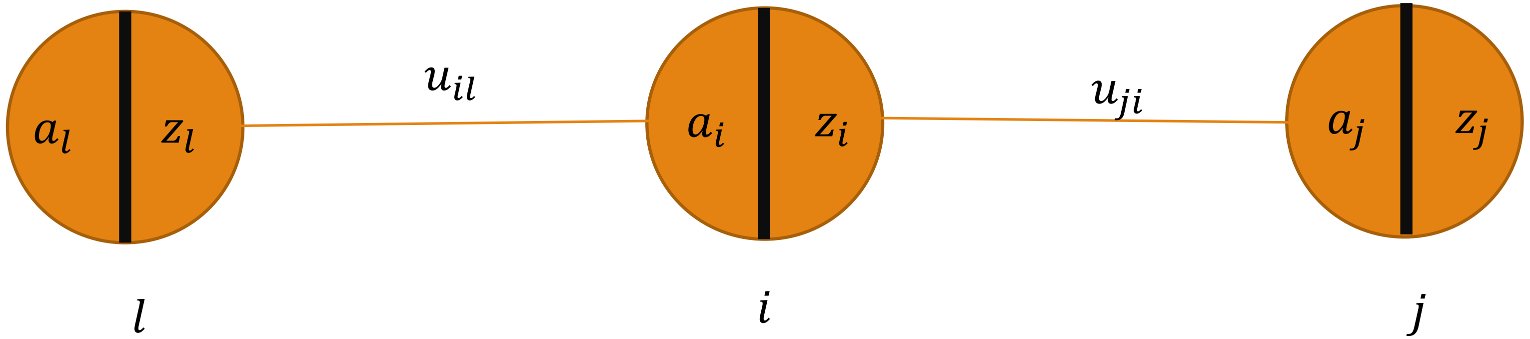
$z_i = \sigma(a_i)$

$$a_i = \sum_l u_{il} z_l$$

$C_0 = (y - y')^2.$

Backpropagation





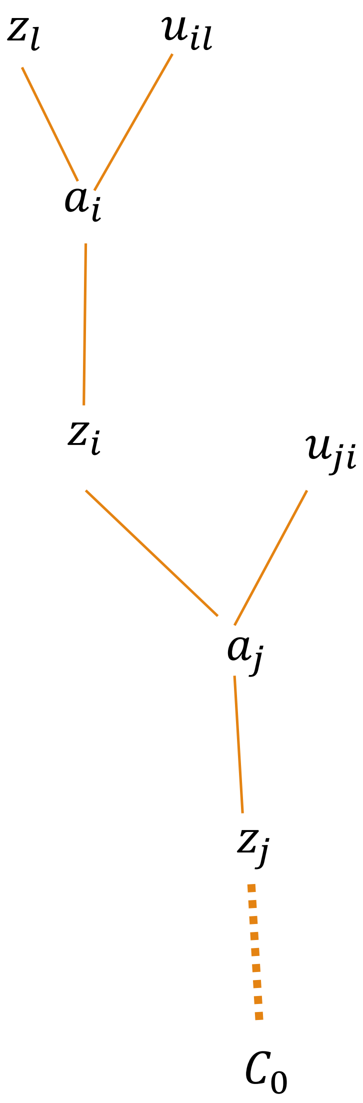
$$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\frac{\partial\, a_i}{\partial\, u_{il}}$$

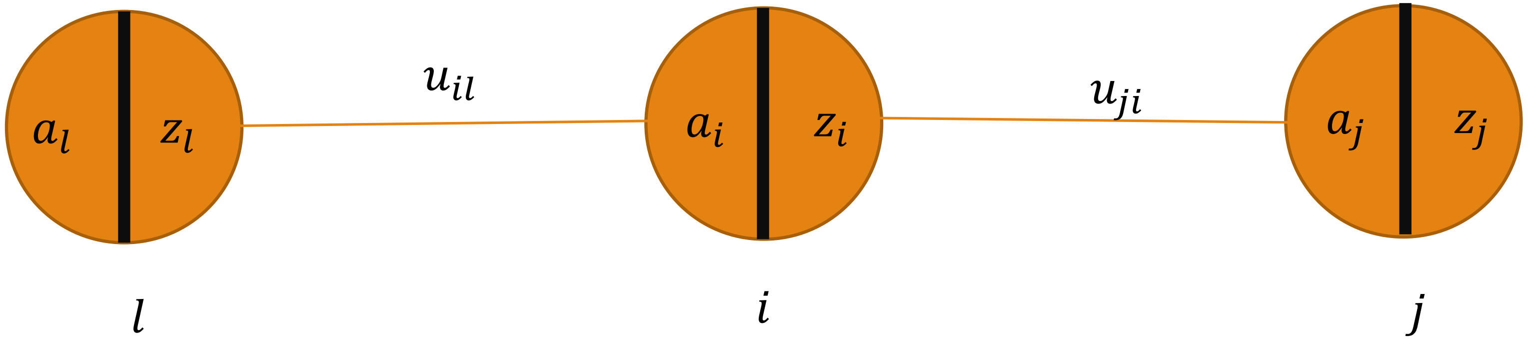
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





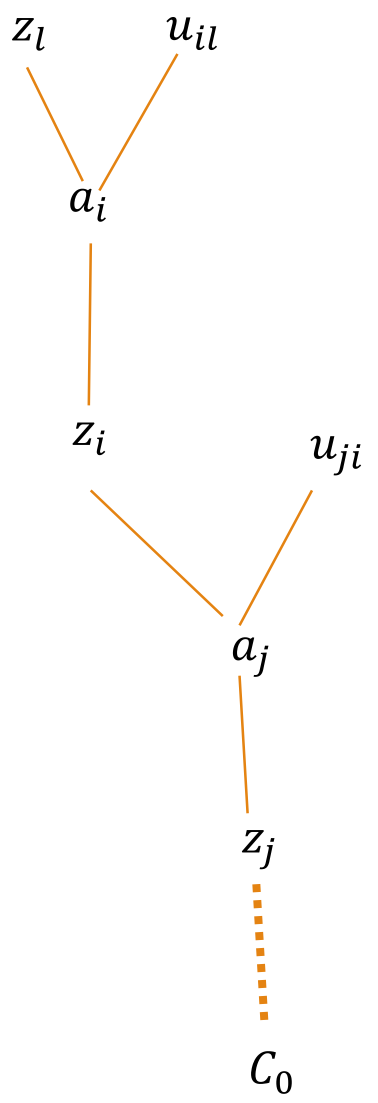
$$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\boxed{\frac{\partial\, a_i}{\partial\, u_{il}}}$$ where $$\boxed{\frac{\partial\, a_i}{\partial\, u_{il}}} = z_l$$

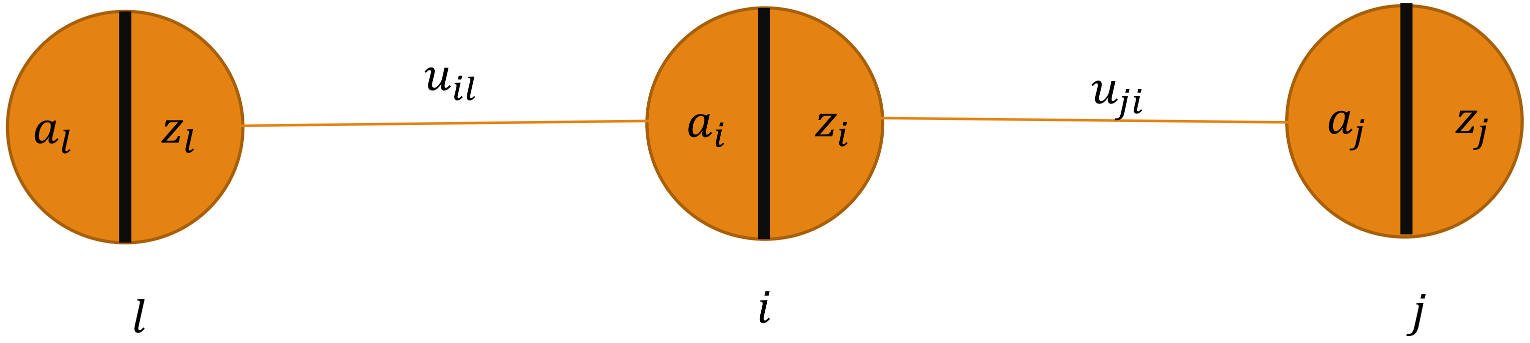
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$\frac{\partial\, C_0}{\partial\, u_{il}} = \boxed{\frac{\partial\, C_0}{\partial\, a_i}}\frac{\partial\, a_i}{\partial\, u_{il}} \quad \text{where} \quad \frac{\partial\, a_i}{\partial\, u_{il}} = z_l$$

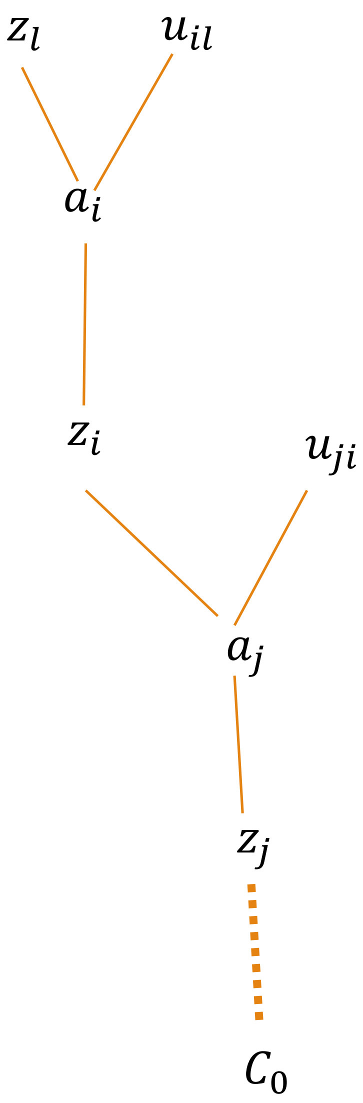
Want to compute $\boxed{\frac{\partial\, C_0}{\partial\, a_i}}$

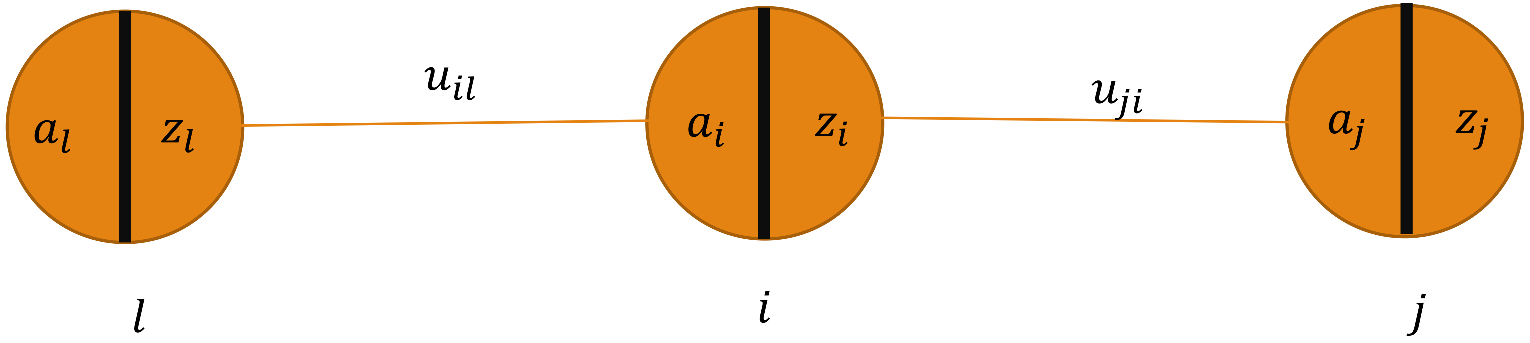
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$\frac{\partial\, C_0}{\partial\, u_{il}} = \boxed{\frac{\partial\, C_0}{\partial\, a_i}} \frac{\partial\, a_i}{\partial\, u_{il}}$$ where $$\frac{\partial\, a_i}{\partial\, u_{il}} = z_l$$

Want to compute $\boxed{\frac{\partial\, C_0}{\partial\, a_i}}$

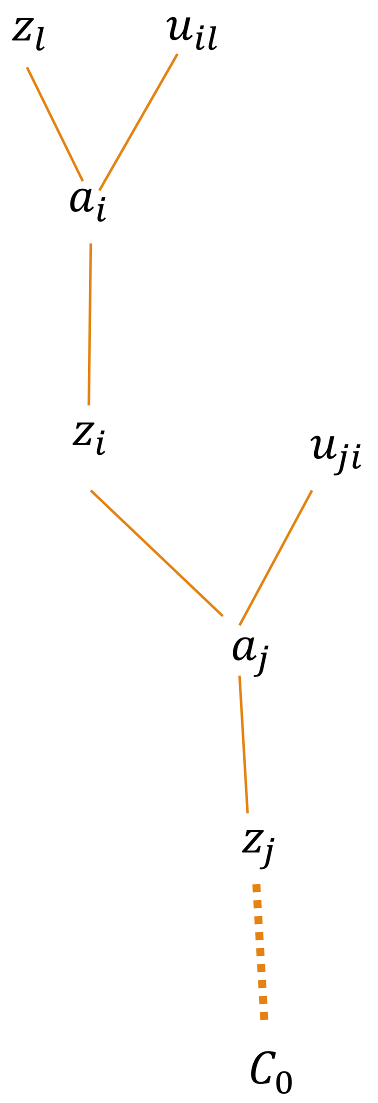
$$\boxed{\frac{\partial\, C_0}{\partial\, a_i}} = \sum_j \frac{\partial\, C_0}{\partial\, a_j} \frac{\partial\, a_j}{\partial\, a_i}$$

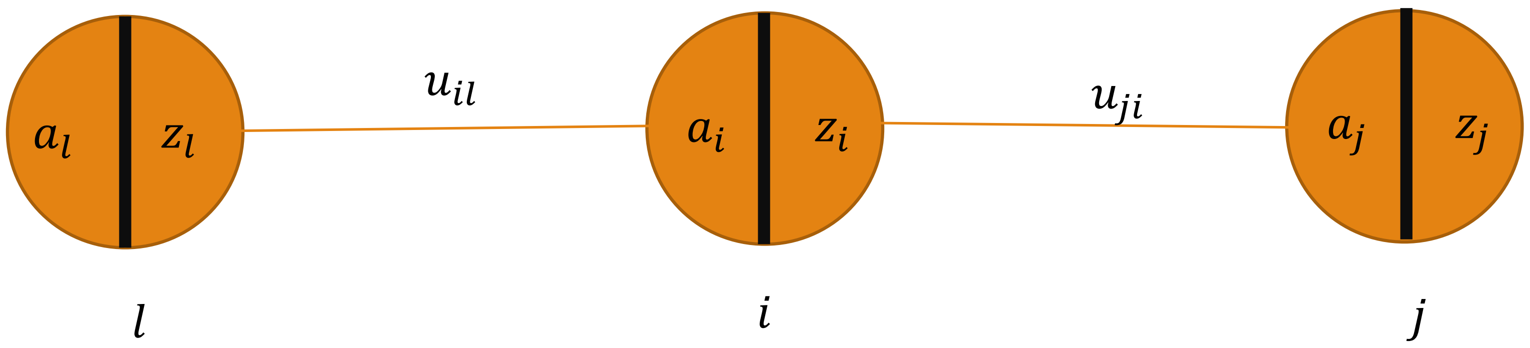
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}} \frac{\partial a_i}{\partial u_{il}} \quad \text{where} \quad \frac{\partial a_i}{\partial u_{il}} = z_l$$

Want to compute $\boxed{\frac{\partial C_0}{\partial a_i}}$

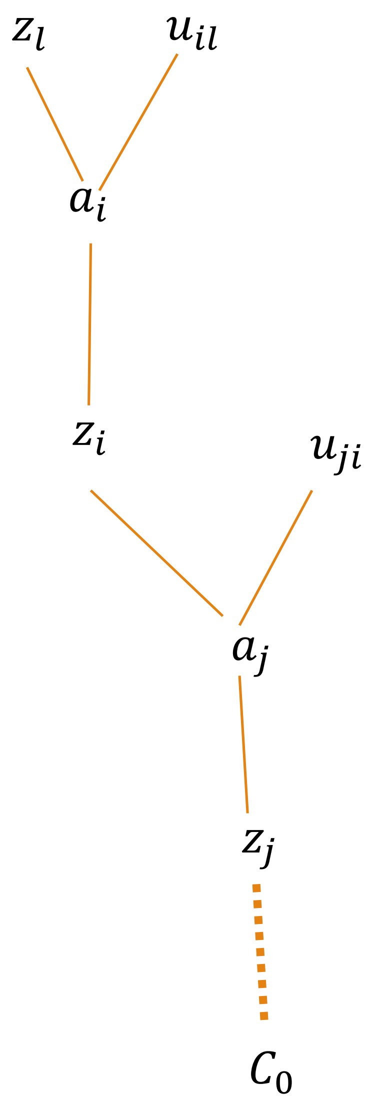
$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i}$$

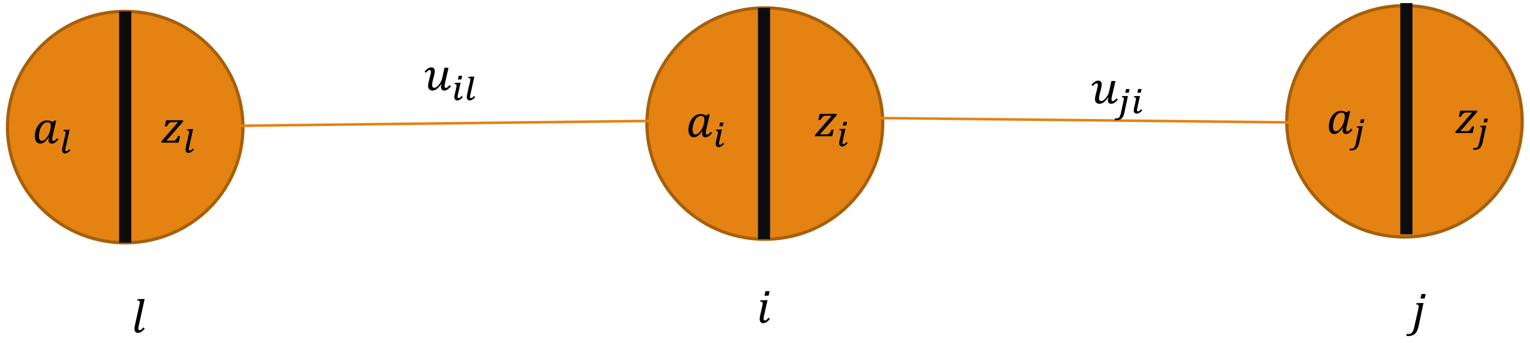
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

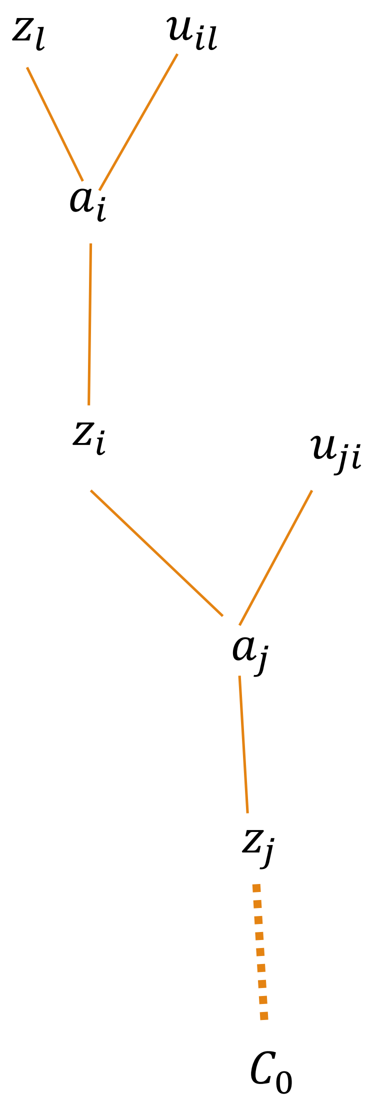
$$C_0 = (y - y')^2.$$

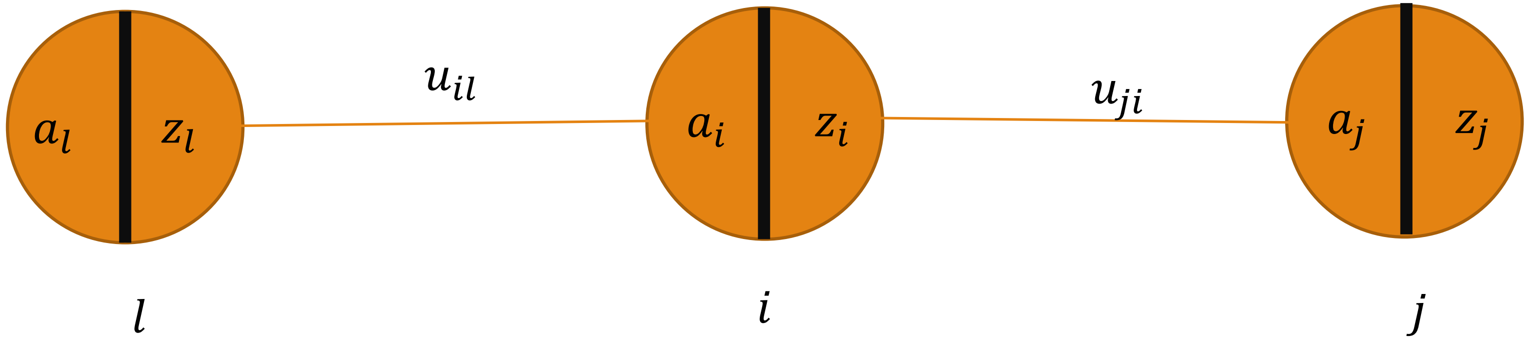
$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}} \frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

Want to compute $\boxed{\frac{\partial C_0}{\partial a_i}}$

$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i}$$

Backpropagation





$z_i = \sigma(a_i)$

$a_i = \sum_l u_{il} z_l$

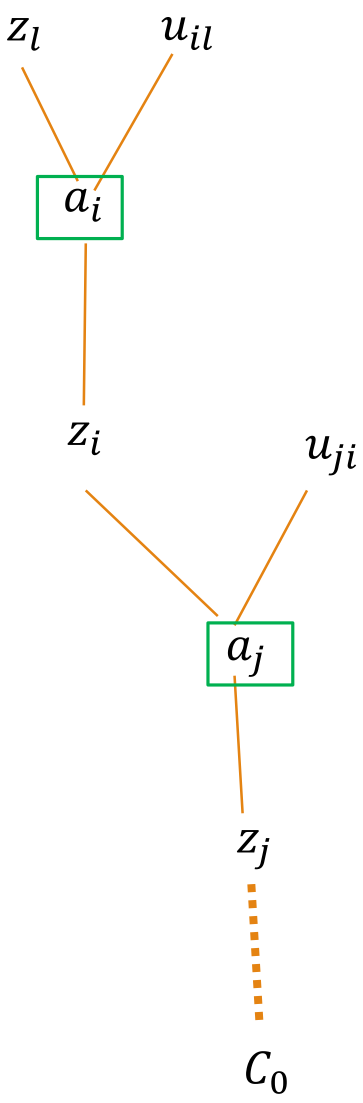
$C_0 = (y - y')^2.$

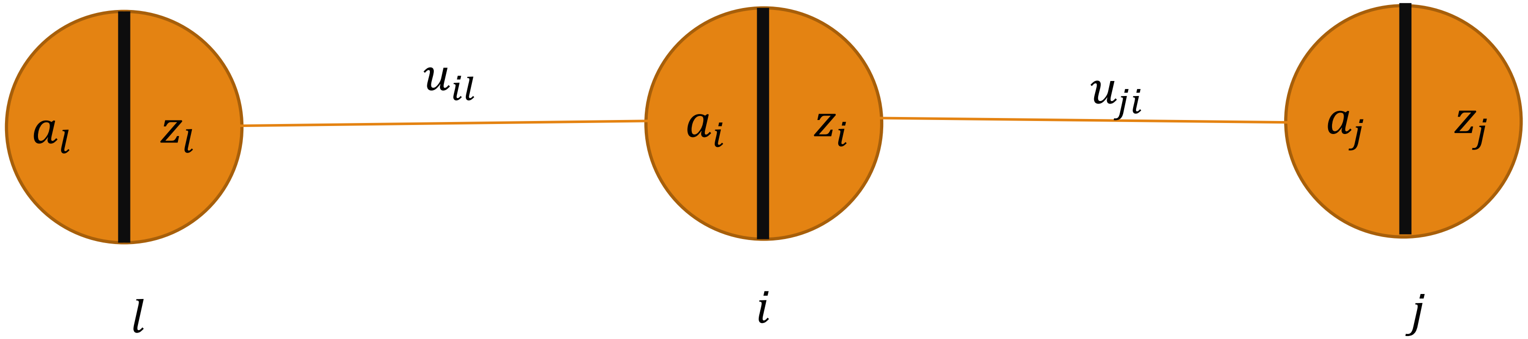
$\frac{\partial\, C_0}{\partial\, u_{il}} = \boxed{\frac{\partial\, C_0}{\partial\, a_i}} \frac{\partial\, a_i}{\partial\, u_{il}}$ where $\frac{\partial\, a_i}{\partial\, u_{il}} = z_l$

Want to compute $\boxed{\frac{\partial\, C_0}{\partial\, a_i}}$

$$\boxed{\frac{\partial\, C_0}{\partial\, a_i}} = \sum_j \frac{\partial\, C_0}{\partial\, a_j} \frac{\partial\, a_j}{\partial\, a_i} = \sum_j \delta_j \frac{\partial\, a_j}{\partial\, a_i} = \sum_j \delta_j \boxed{\frac{\partial\, a_j}{\partial\, a_i}}$$

Backpropagation





$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

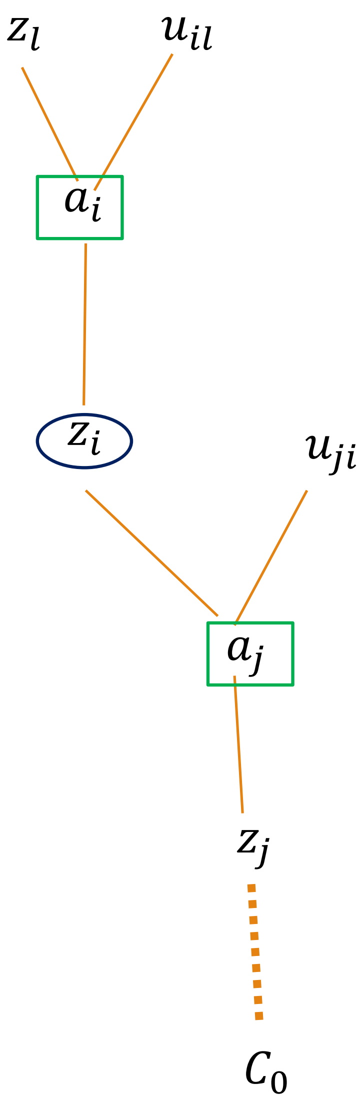
$$\frac{\partial C_0}{\partial u_{il}} = \frac{\partial C_0}{\partial a_i} \frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

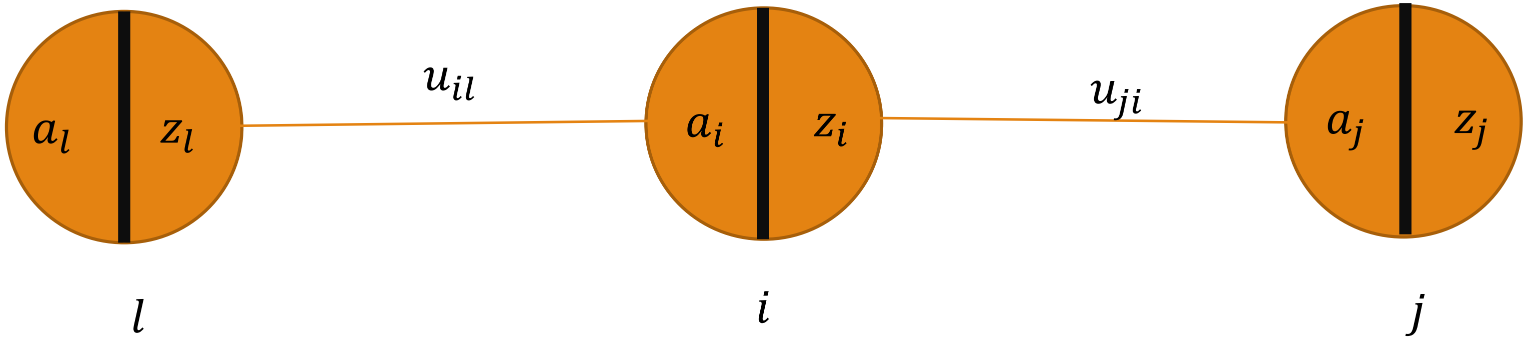
Want to compute $\frac{\partial C_0}{\partial a_i}$

$$\frac{\partial C_0}{\partial a_i} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i}$$

$$\frac{\partial a_j}{\partial a_i}$$

Backpropagation





$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}}\frac{\partial a_i}{\partial u_{il}}$$

where $\frac{\partial a_i}{\partial u_{il}} = z_l$

Want to compute $\boxed{\frac{\partial C_0}{\partial a_i}}$

$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j}\frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}}$$

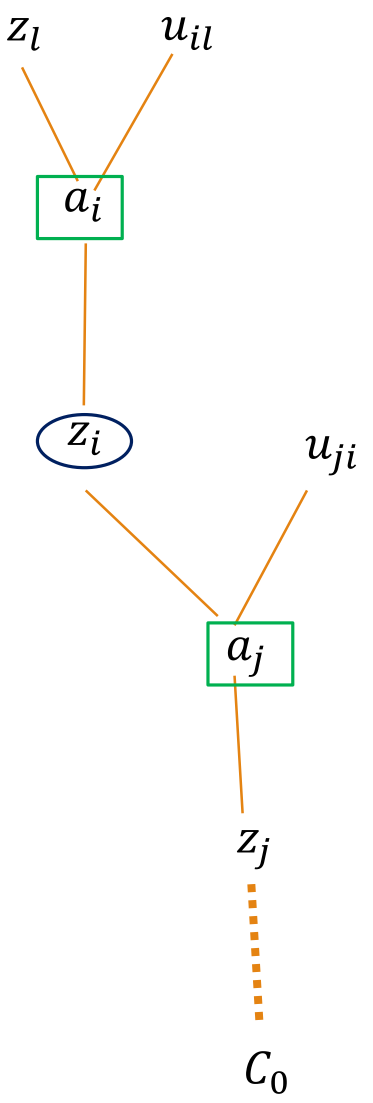
$$\boxed{\frac{\partial a_j}{\partial a_i}} = \frac{\partial a_j}{\partial z_i}\frac{\partial z_i}{\partial a_i}$$

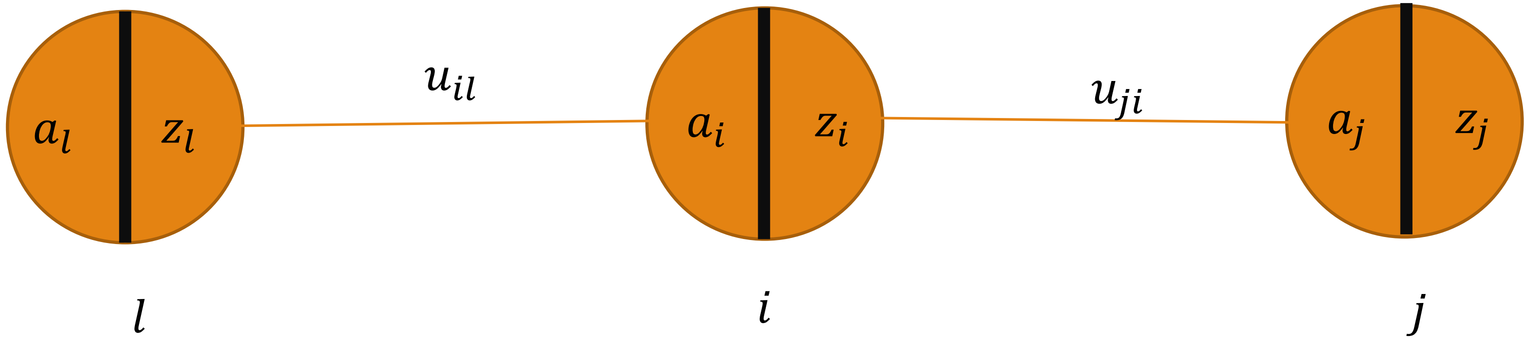
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}} \frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

Want to compute $\boxed{\frac{\partial C_0}{\partial a_i}}$

$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}}$$

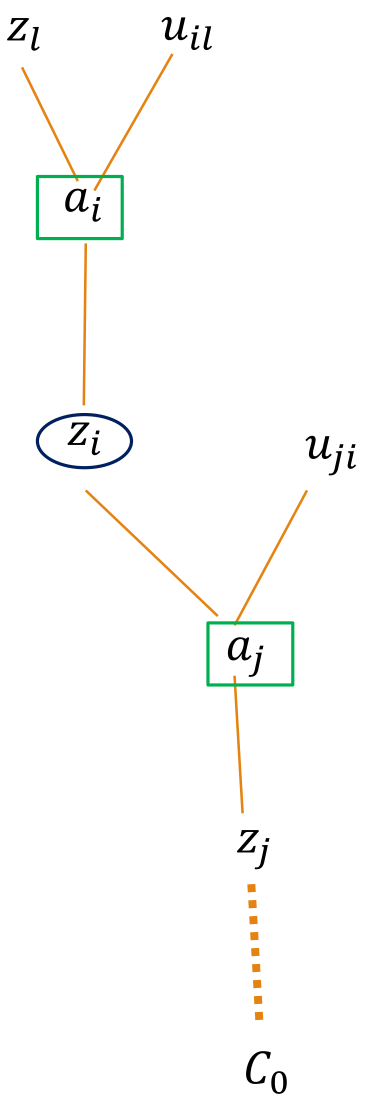
$$\boxed{\frac{\partial a_j}{\partial a_i}} = \boxed{\frac{\partial a_j}{\partial z_i}} \frac{\partial z_i}{\partial a_i}$$

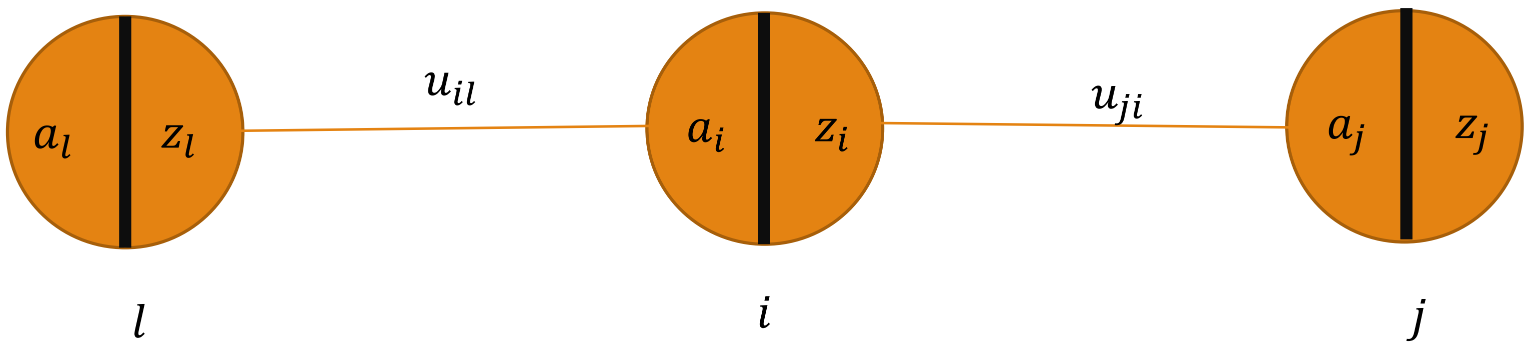
$$z_i = \sigma(a_i)$$

$$\boxed{a_i = \sum_l u_{il} z_l}$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$\frac{\partial C_0}{\partial u_{il}} = \frac{\partial C_0}{\partial a_i}\frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

Want to compute $\frac{\partial C_0}{\partial a_i}$

$$\frac{\partial C_0}{\partial a_i} = \sum_j \frac{\partial C_0}{\partial a_j}\frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i}$$

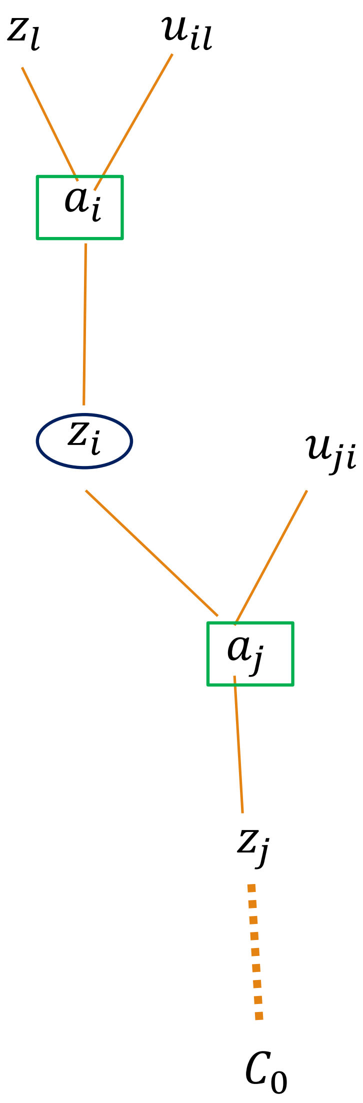
$$\frac{\partial a_j}{\partial a_i} = \frac{\partial a_j}{\partial z_i}\frac{\partial z_i}{\partial a_i}$$

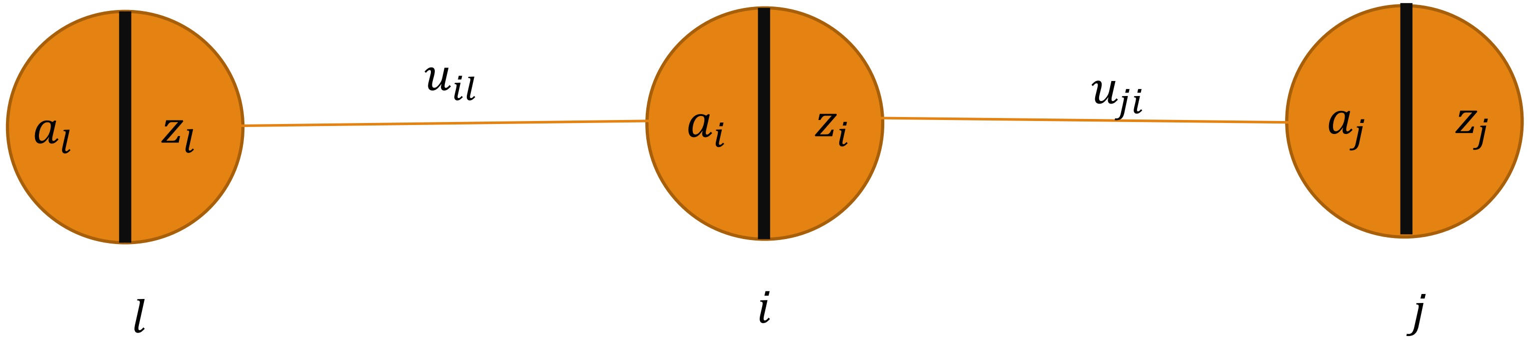
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$\boxed{z_i = \sigma(a_i)}$$

$$\boxed{a_i = \sum_l u_{il} z_l}$$

$$C_0 = (y - y')^2.$$

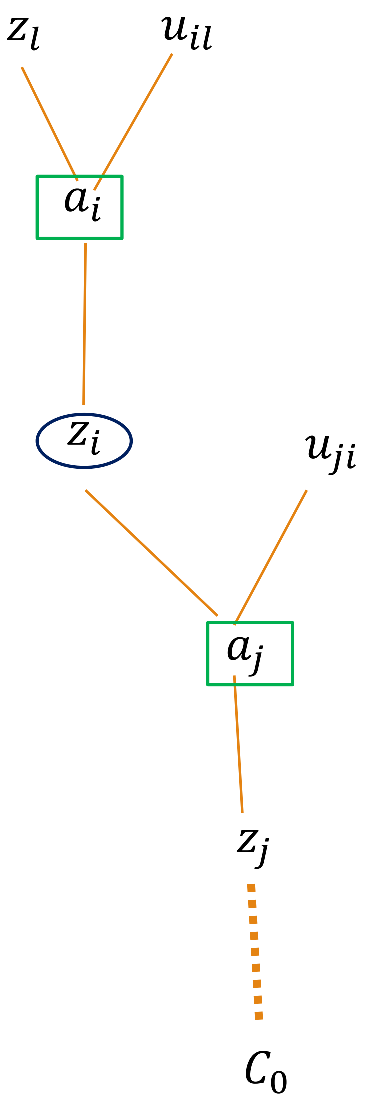
$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}} \frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

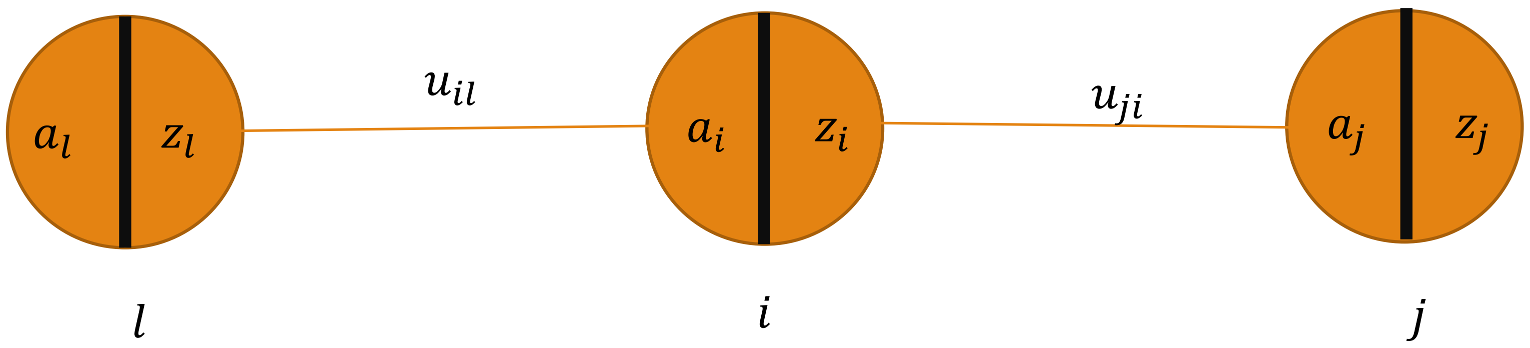
Want to compute $\boxed{\dfrac{\partial C_0}{\partial a_i}}$

$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}}$$

$$\boxed{\frac{\partial a_j}{\partial a_i}} = \boxed{\frac{\partial a_j}{\partial z_i}} \boxed{\frac{\partial z_i}{\partial a_i}} = u_{ji} \sigma'(a_i)$$

Backpropagation





$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}} \frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

$$\boxed{z_i = \sigma(a_i)}$$

$$\boxed{a_i = \sum_l u_{il} z_l}$$

$$C_0 = (y - y')^2.$$

Want to compute $\boxed{\frac{\partial C_0}{\partial a_i}}$

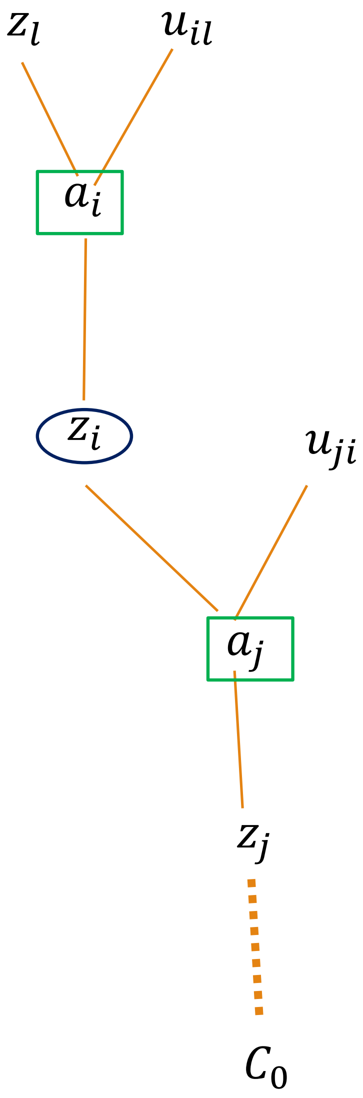
$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}}$$

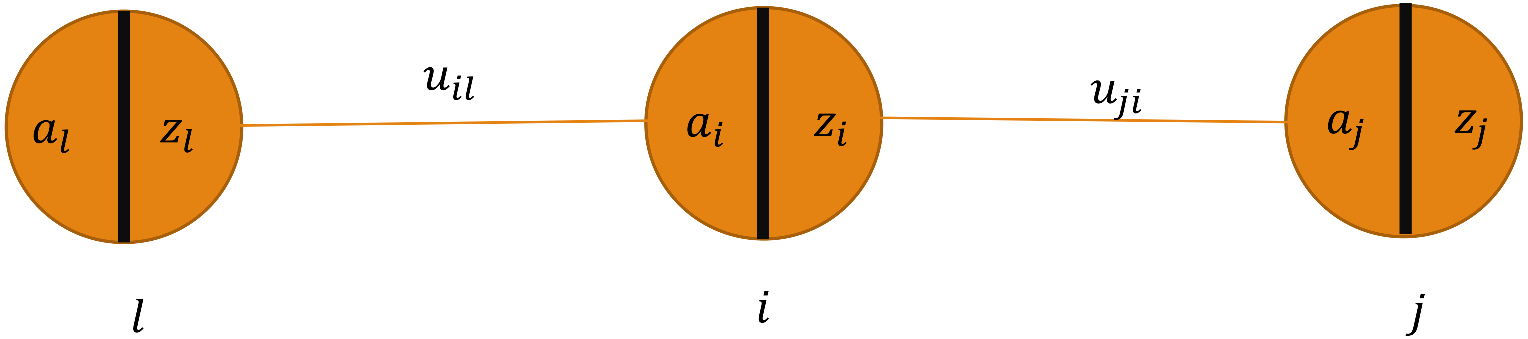
$$\boxed{\frac{\partial a_j}{\partial a_i}} = \boxed{\frac{\partial a_j}{\partial z_i}} \boxed{\frac{\partial z_i}{\partial a_i}} = u_{ji} \sigma'(a_i)$$

Hence

$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}}$$

Backpropagation





$$\boxed{z_i = \sigma(a_i)}$$

$$\boxed{a_i = \sum_l u_{il} z_l}$$

$$C_0 = (y - y')^2.$$

$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}} \frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

Want to compute $\boxed{\dfrac{\partial C_0}{\partial a_i}}$

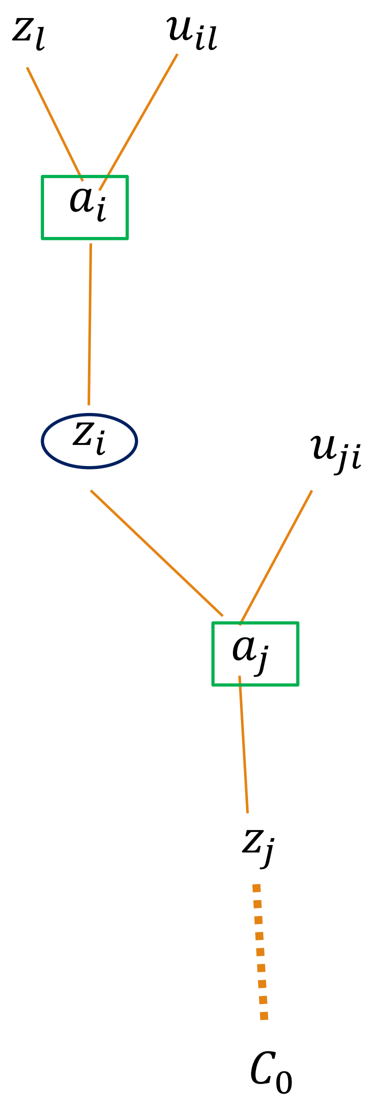
$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}}$$

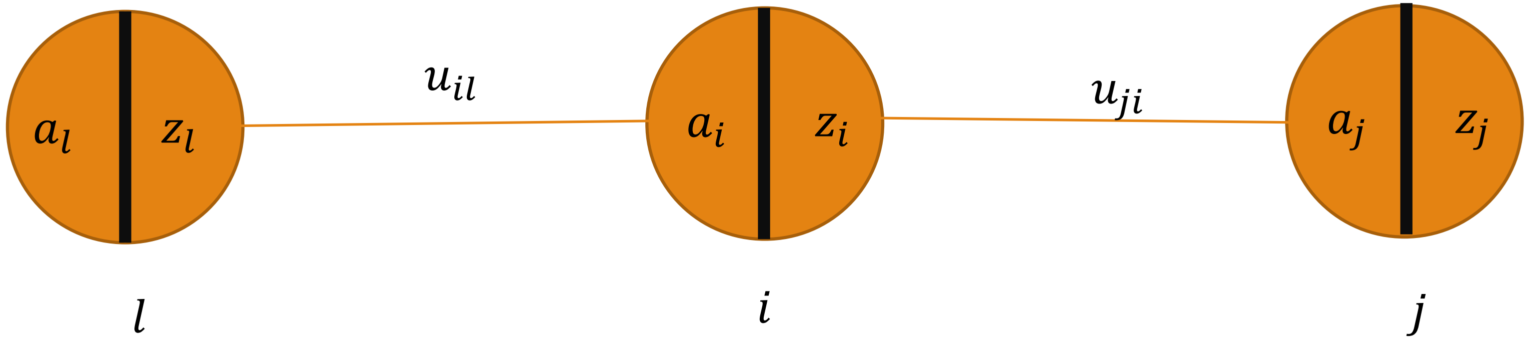
$$\boxed{\frac{\partial a_j}{\partial a_i}} = \boxed{\frac{\partial a_j}{\partial z_i}} \boxed{\frac{\partial z_i}{\partial a_i}} = u_{ji} \sigma'(a_i)$$

Hence

$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}} = \sum_j \delta_j u_{ji} \sigma'(a_i)$$

Backpropagation





$$\boxed{z_i = \sigma(a_i)}$$

$$\boxed{a_i = \sum_l u_{il} z_l}$$

$$C_0 = (y - y')^2.$$

$$\frac{\partial C_0}{\partial u_{il}} = \boxed{\frac{\partial C_0}{\partial a_i}} \frac{\partial a_i}{\partial u_{il}}$$ where $$\frac{\partial a_i}{\partial u_{il}} = z_l$$

Want to compute $\boxed{\frac{\partial C_0}{\partial a_i}}$

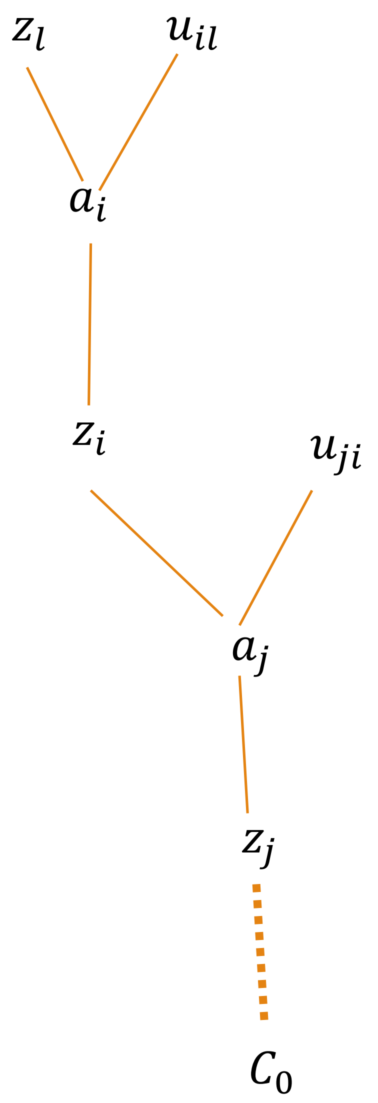
$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \frac{\partial C_0}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}}$$

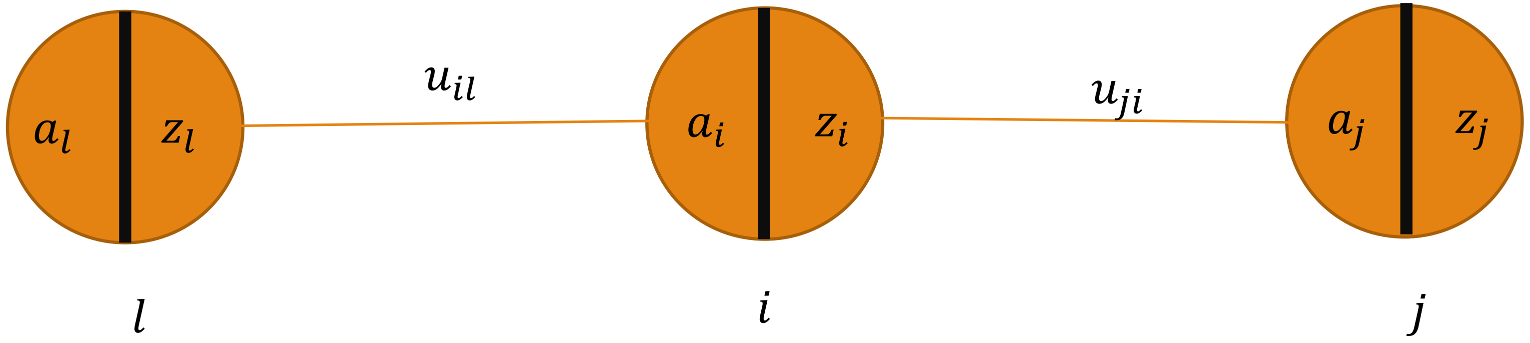
$$\boxed{\frac{\partial a_j}{\partial a_i}} = \boxed{\frac{\partial a_j}{\partial z_i}} \boxed{\frac{\partial z_i}{\partial a_i}} = u_{ji} \sigma'(a_i)$$

Hence

$$\boxed{\frac{\partial C_0}{\partial a_i}} = \sum_j \delta_j \boxed{\frac{\partial a_j}{\partial a_i}} = \sum_j \delta_j u_{ji} \sigma'(a_i) = \sigma'(a_i) \sum_j \delta_j u_{ji}$$

Backpropagation





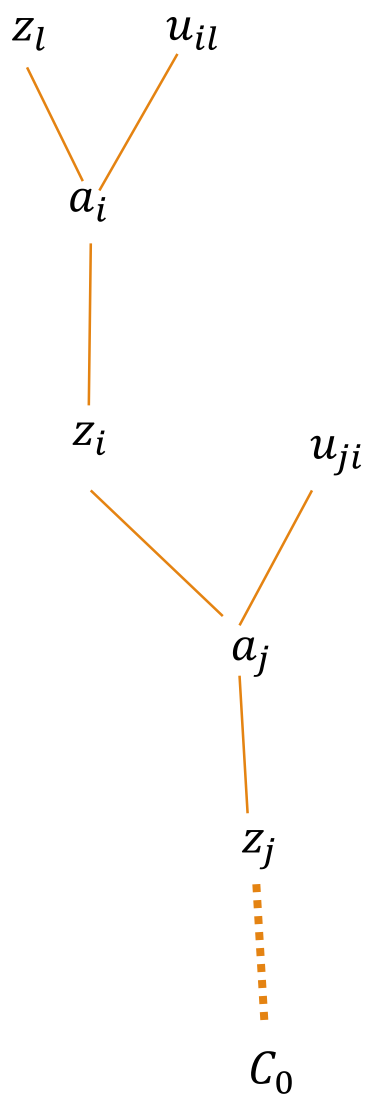
In summary $$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\frac{\partial\, a_i}{\partial\, u_{il}}$$

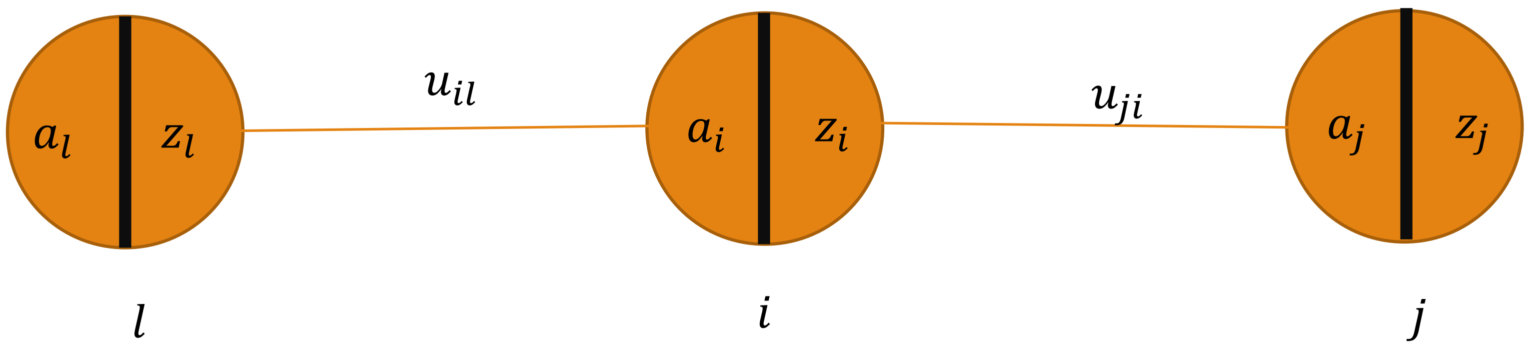
$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

Backpropagation





$$z_i = \sigma(a_i)$$

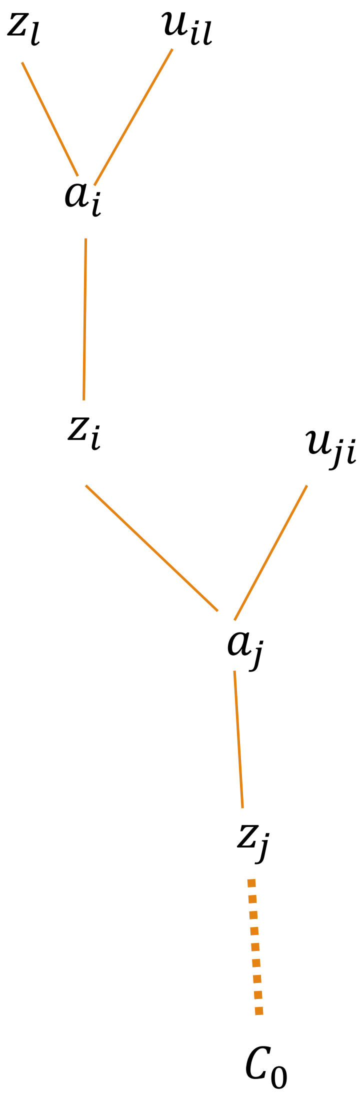
$$a_i = \sum_l u_{il} z_l$$

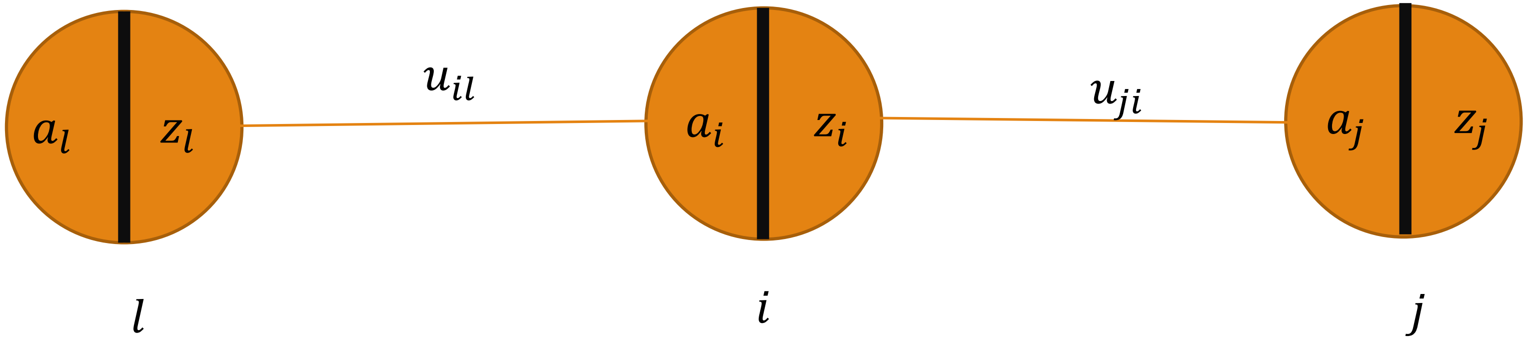
$$C_0 = (y - y')^2.$$

In summary $$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i} \frac{\partial\, a_i}{\partial\, u_{il}}$$

where $$\frac{\partial\, a_i}{\partial\, u_{il}} = z_l$$

Backpropagation





$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

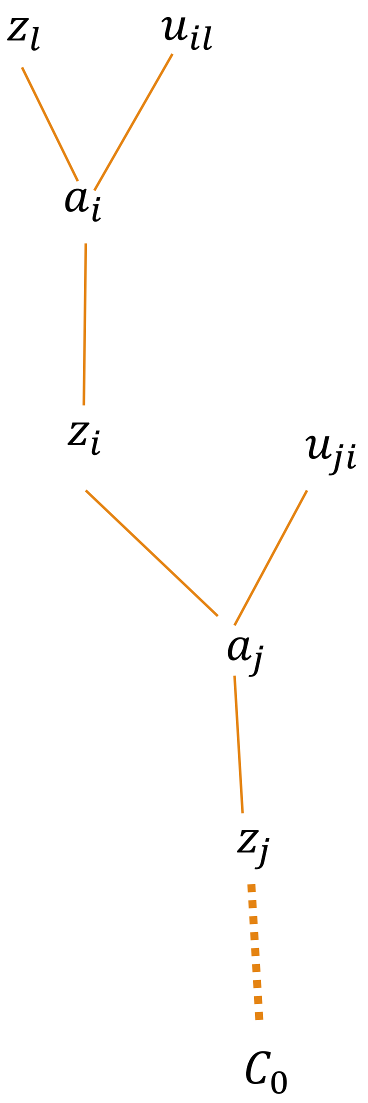
$$C_0 = (y - y')^2.$$

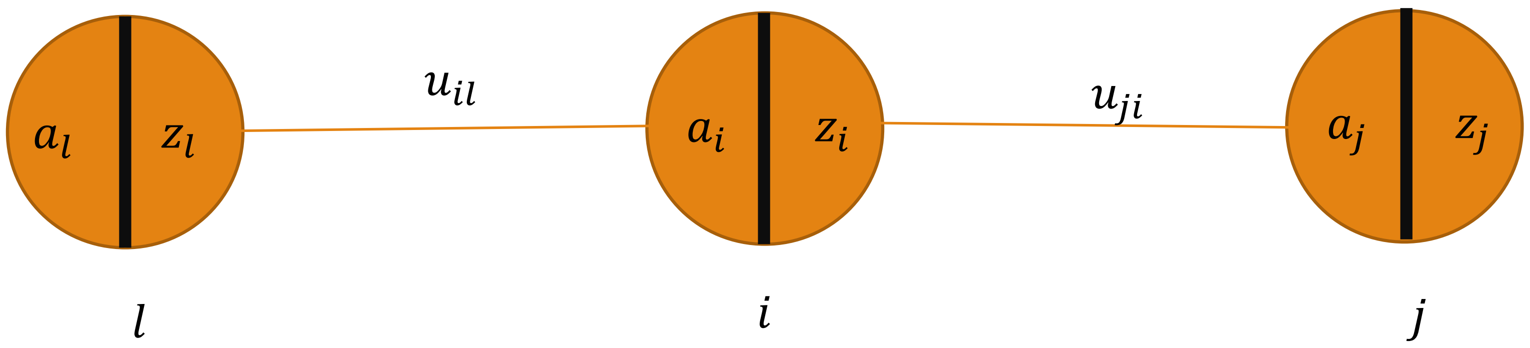
In summary $\dfrac{\partial\, C_0}{\partial\, u_{il}} = \dfrac{\partial\, C_0}{\partial\, a_i} \dfrac{\partial\, a_i}{\partial\, u_{il}}$

where $\dfrac{\partial\, a_i}{\partial\, u_{il}} = z_l$

$$\delta_i = \frac{\partial\, C_0}{\partial\, a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) = \sigma'(a_i) \sum_j \delta_j u_{ji}$$

Backpropagation





$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

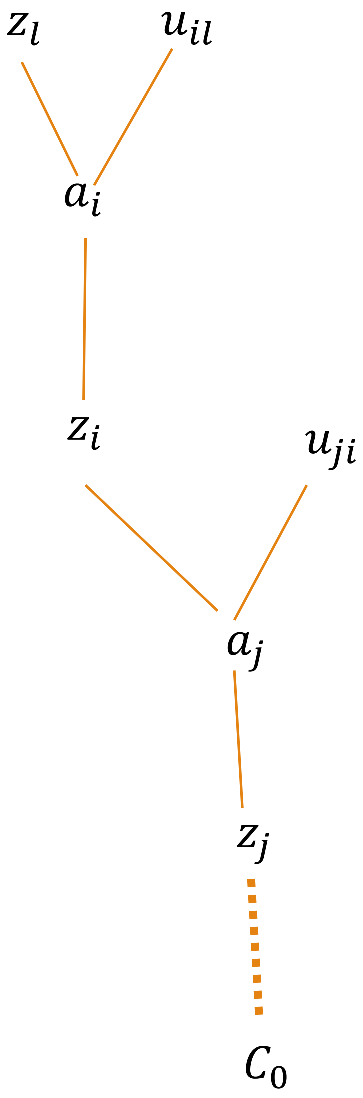
In summary $$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\frac{\partial\, a_i}{\partial\, u_{il}}$$

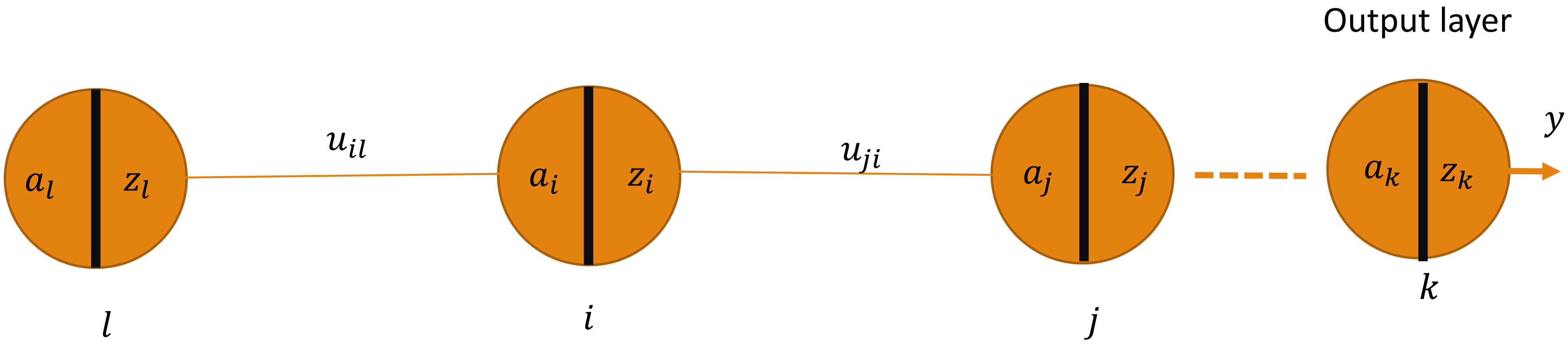
where $$\frac{\partial\, a_i}{\partial\, u_{il}} = z_l$$

$$\delta_i = \frac{\partial\, C_0}{\partial\, a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) \; = \sigma'(a_i) \sum_j \delta_j u_{ji}$$

If we have the δ_j we can compute δ_i hence the name backpropagation

Backpropagation





$$\frac{\partial C_0}{\partial u_{il}} = \frac{\partial C_0}{\partial a_i}\frac{\partial a_i}{\partial u_{il}} = \delta_i z_l$$

$$\delta_i = \frac{\partial C_0}{\partial a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) = \sigma'(a_i) \sum_j \delta_j u_{ji} \qquad (*)$$

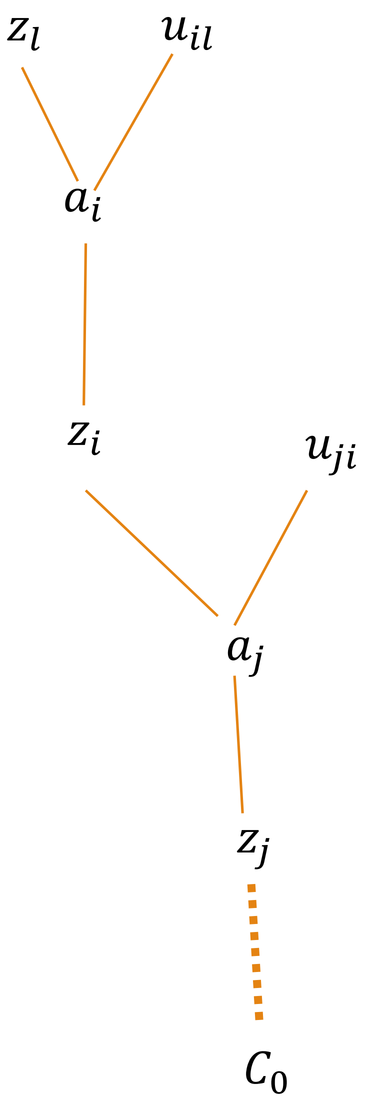
$$z_i = \sigma(a_i)$$

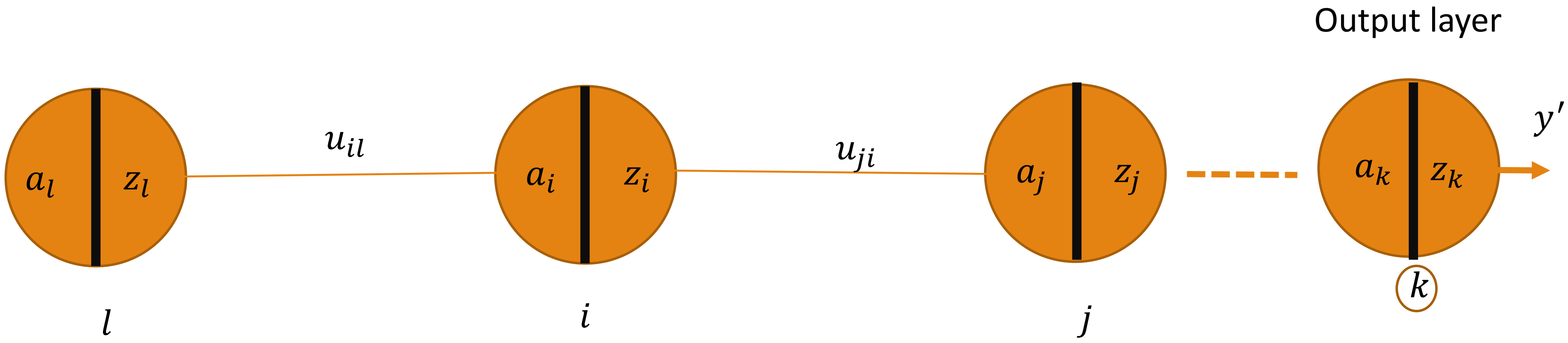
$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

$$z_k = y'$$

Backpropagation





$$\frac{\partial\ C_0}{\partial\ u_{il}} = \frac{\partial\ C_0}{\partial\ a_i}\frac{\partial\ a_i}{\partial\ u_{il}} = \delta_i z_l$$

$$\delta_i = \frac{\partial\ C_0}{\partial\ a_i} = \sum_j \delta_j u_{ji}\sigma'(a_i) = \sigma'(a_i)\sum_j \delta_j u_{ji} \qquad (*)$$

Compute $\delta_k = \dfrac{\partial\ C_0}{\partial\ a_k}$

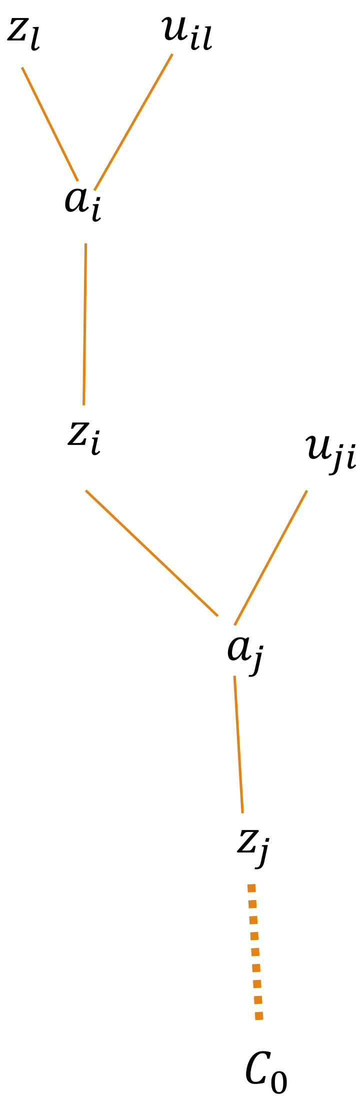
$$z_i = \sigma(a_i)$$

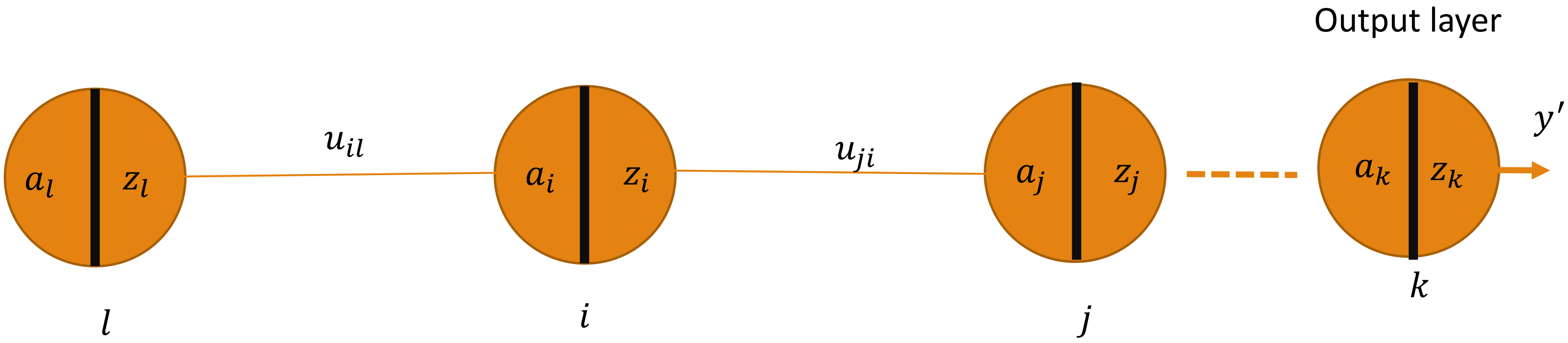
$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

$$\sigma(a_k) = z_k = y'$$

Backpropagation





$$\frac{\partial\ C_0}{\partial\ u_{il}} = \frac{\partial\ C_0}{\partial\ a_i}\frac{\partial\ a_i}{\partial\ u_{il}} = \delta_i z_l$$

$$\delta_i = \frac{\partial\ C_0}{\partial\ a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) \ = \sigma'(a_i) \sum_j \delta_j u_{ji} \qquad (*)$$

$$\delta_k = \boxed{\frac{\partial\ C_0}{\partial\ a_k}} = \frac{\partial\ C_0}{\partial z_k}\frac{\partial z_k}{\partial a_k}$$

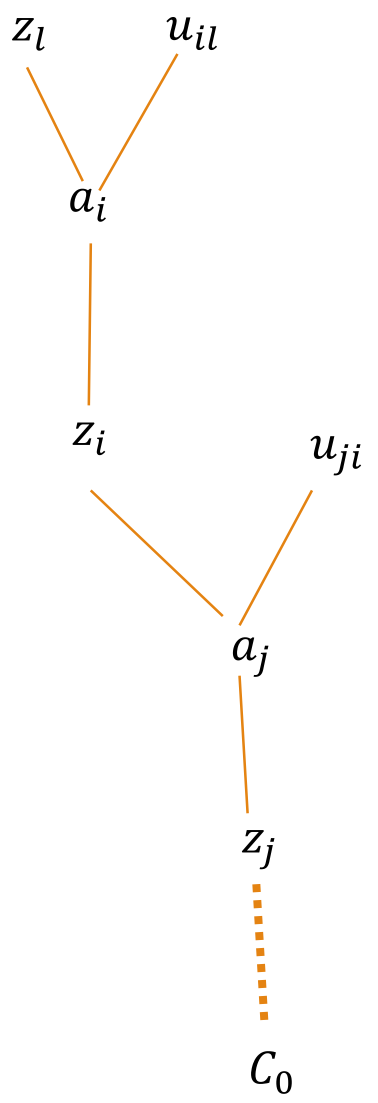
$$z_i = \sigma(a_i)$$

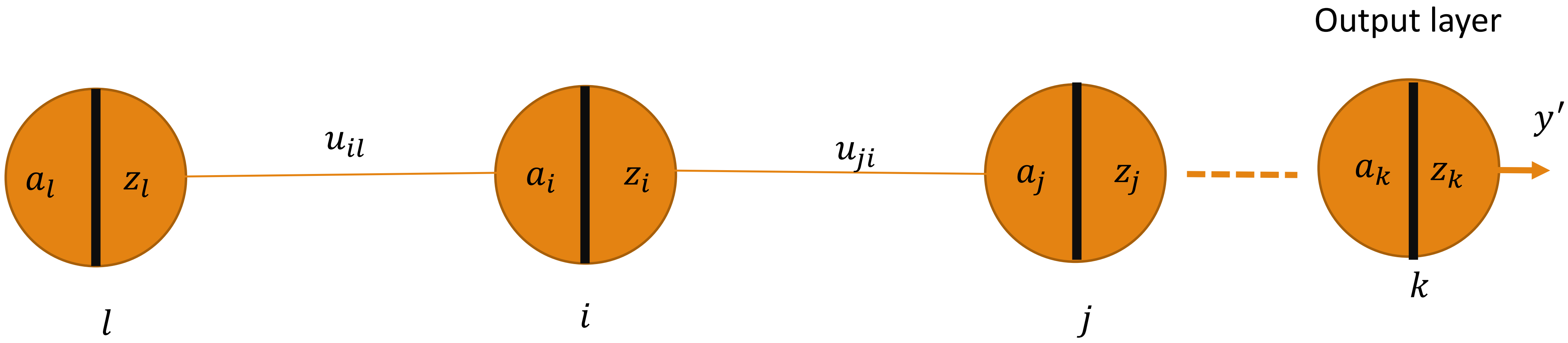
$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

$$\boxed{\sigma(a_k) = z_k = y'}$$

Backpropagation





$$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\frac{\partial\, a_i}{\partial\, u_{il}} = \delta_i z_l$$

$$\delta_i = \frac{\partial\, C_0}{\partial\, a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) \; = \sigma'(a_i) \sum_j \delta_j u_{ji} \qquad (*)$$

$$\delta_k = \frac{\partial\, C_0}{\partial\, a_k} = \frac{\partial\, C_0}{\partial z_k}\frac{\partial z_k}{\partial a_k} = \frac{\partial\, C_0}{\partial y'}\frac{\partial y'}{\partial a_k}$$

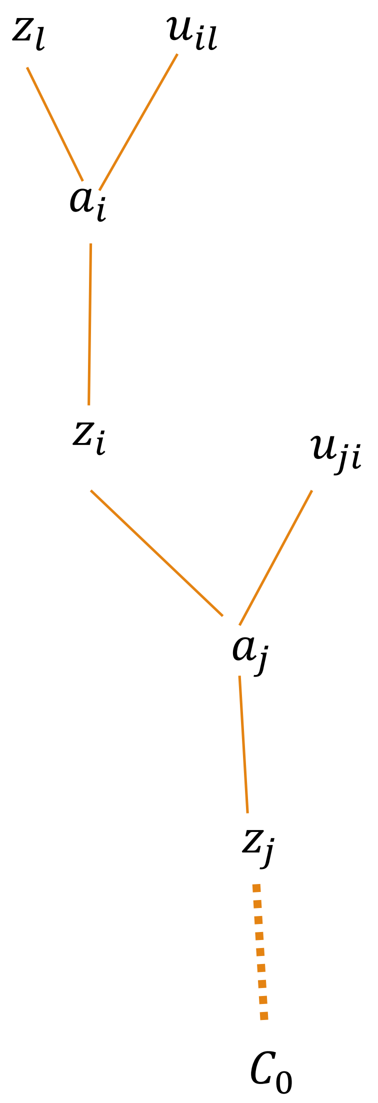
$$z_i = \sigma(a_i)$$

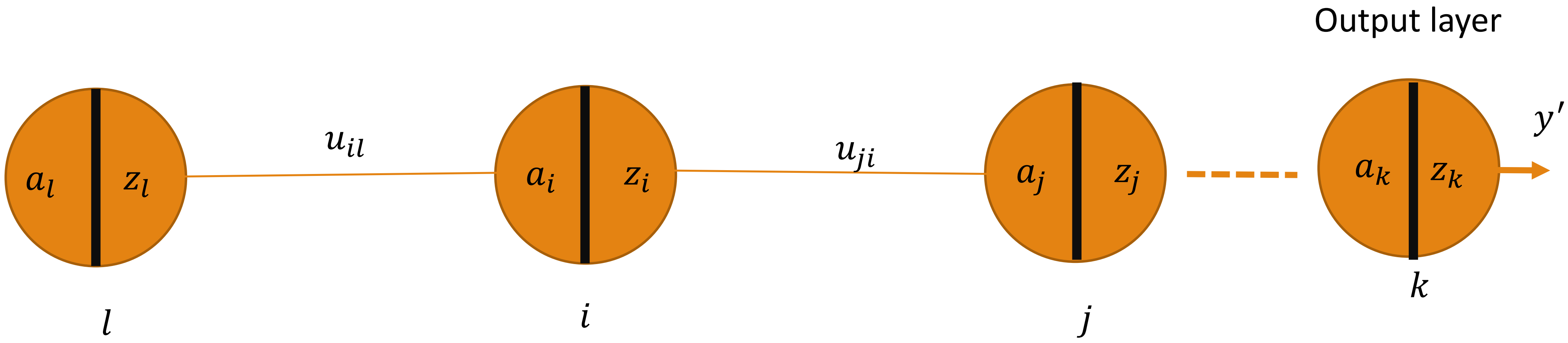
$$a_i = \sum_l u_{il} z_l$$

$$C_0 = (y - y')^2.$$

$$\sigma(a_k) = z_k = y'$$

Backpropagation





$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$\boxed{C_0 = (y - y')^2.}$$

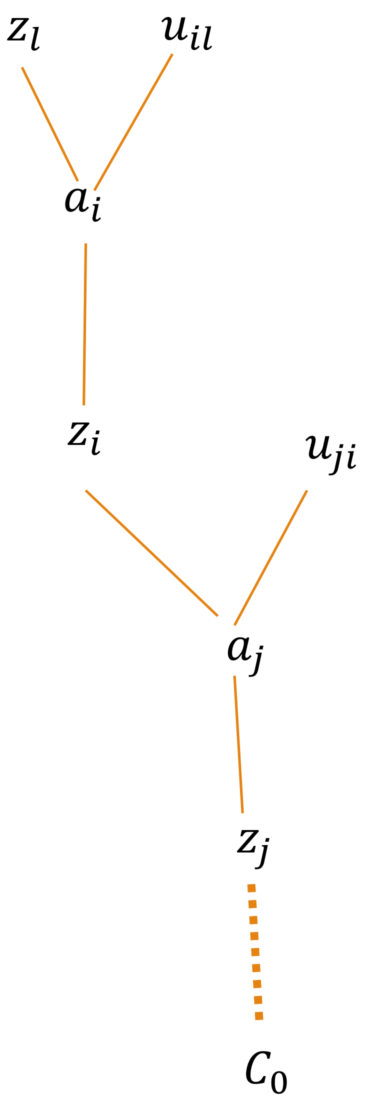
$$\boxed{\sigma(a_k) = z_k = y'}$$

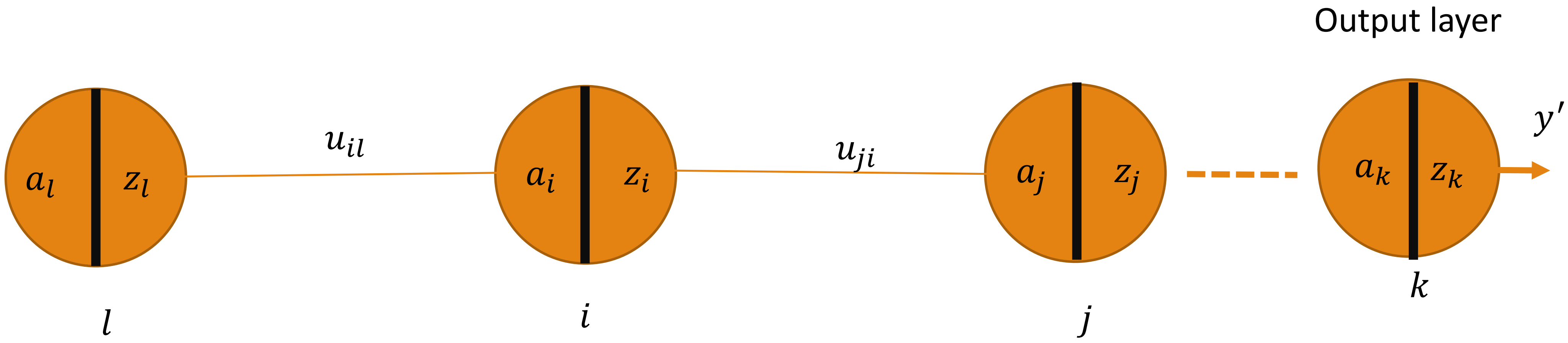
$$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\frac{\partial\, a_i}{\partial\, u_{il}} = \delta_i z_l$$

$$\delta_i = \frac{\partial\, C_0}{\partial\, a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) \ = \sigma'(a_i) \sum_j \delta_j u_{ji} \qquad (*)$$

$$\delta_k = \frac{\partial\, C_0}{\partial\, a_k} = \frac{\partial\, C_0}{\partial z_k}\frac{\partial z_k}{\partial a_k} = \boxed{\frac{\partial\, C_0}{\partial\, y'}}\boxed{\frac{\partial y'}{\partial a_k}} = \boxed{\frac{\partial\, (y - y')^2}{\partial\, y'}}\boxed{\sigma'(a_k)}$$

Backpropagation





$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$\boxed{C_0 = (y - y')^2.}$$

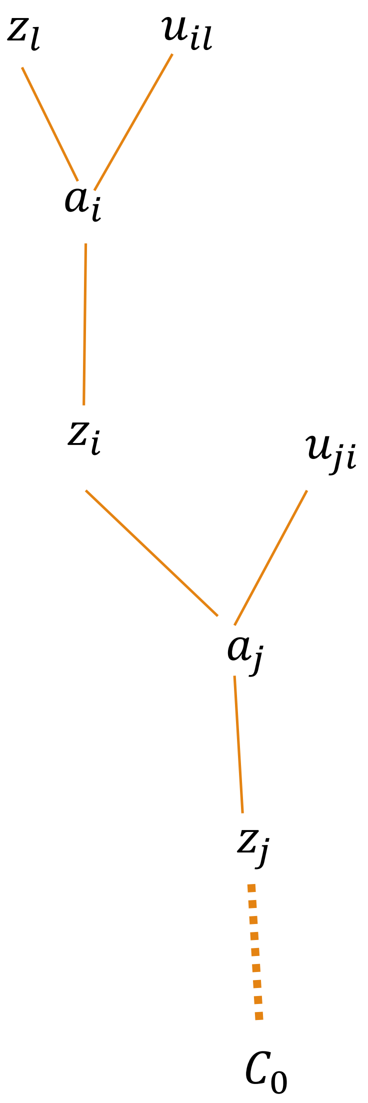
$$\boxed{\sigma(a_k) = z_k = y'}$$

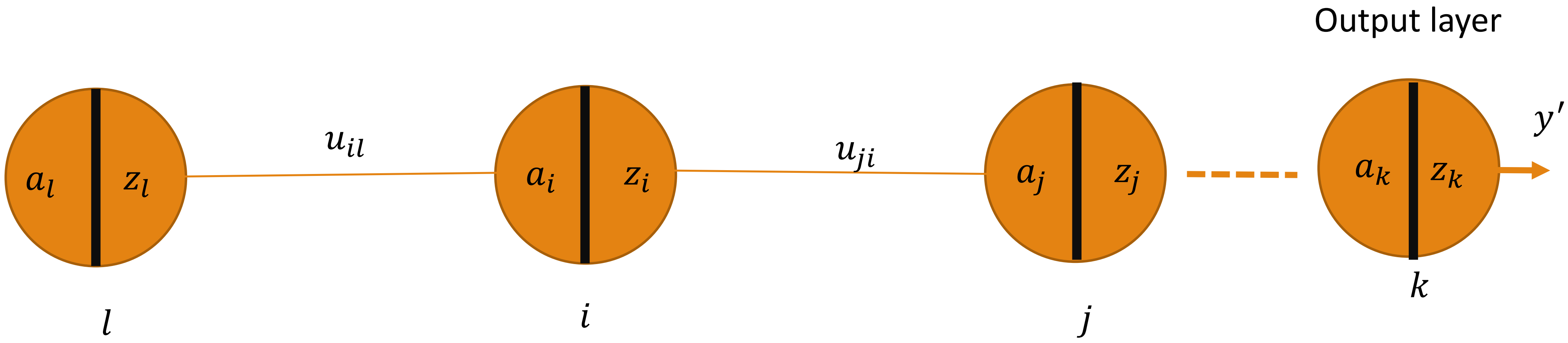
$$\frac{\partial C_0}{\partial u_{il}} = \frac{\partial C_0}{\partial a_i}\frac{\partial a_i}{\partial u_{il}} = \delta_i z_l$$

$$\delta_i = \frac{\partial C_0}{\partial a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) = \sigma'(a_i) \sum_j \delta_j u_{ji} \qquad (*)$$

$$\delta_k = \frac{\partial C_0}{\partial a_k} = \frac{\partial C_0}{\partial z_k}\frac{\partial z_k}{\partial a_k} = \boxed{\frac{\partial C_0}{\partial y'}}\boxed{\frac{\partial y'}{\partial a_k}} = \boxed{\frac{\partial (y - y')^2}{\partial y'}}\boxed{\sigma'(a_k)} = -2(y - y')\sigma'(a_k)$$

Backpropagation





$$z_i = \sigma(a_i)$$

$$a_i = \sum_l u_{il} z_l$$

$$\boxed{C_0 = (y - y')^2.}$$

$$\boxed{\sigma(a_k) = z_k = y'}$$

$$\frac{\partial\, C_0}{\partial\, u_{il}} = \frac{\partial\, C_0}{\partial\, a_i}\frac{\partial\, a_i}{\partial\, u_{il}} = \delta_i z_l$$

$$\delta_i = \frac{\partial\, C_0}{\partial\, a_i} = \sum_j \delta_j u_{ji} \sigma'(a_i) \;\; = \sigma'(a_i) \sum_j \delta_j u_{ji} \qquad (*)$$

$$\delta_k = \frac{\partial\, C_0}{\partial\, a_k} = \frac{\partial\, C_0}{\partial z_k}\frac{\partial z_k}{\partial a_k} = \boxed{\frac{\partial\, C_0}{\partial\, y'}}\boxed{\frac{\partial y'}{\partial a_k}} = \boxed{\frac{\partial\, (y - y')^2}{\partial\, y'}}\boxed{\sigma'(a_k)} = -2(y - y')\sigma'(a_k)$$

Knowing δ_k, we can compute all the δ_h that comes before it using the formula (*)

Backpropagation Algorithm

Input a set of examples $\{(x_i, y_i)\}^n{}_{i=1}$

(1) Arbitrary choose the weights randomly
(2) For each x_k in the training example set:
    (1) Feedforward: Apply x_k to the neural network and compute the output y_k'
    (2) Compute $\delta_k = -2(y - y')\sigma'(a_k)$
    (3) Compute each of $\delta_i = \sigma'(a_i)\sum_j \delta_j u_{ji}$
    (4) Compute $\frac{\partial\, C_0}{\partial\, u_{il}} = \delta_i z_l$
    (5) Gradient decent : Update the weights u_{il}:=u_{il}-q $\frac{\partial\, C_0}{\partial\, u_{il}}$

It is usually good to initiate the weights to small values.