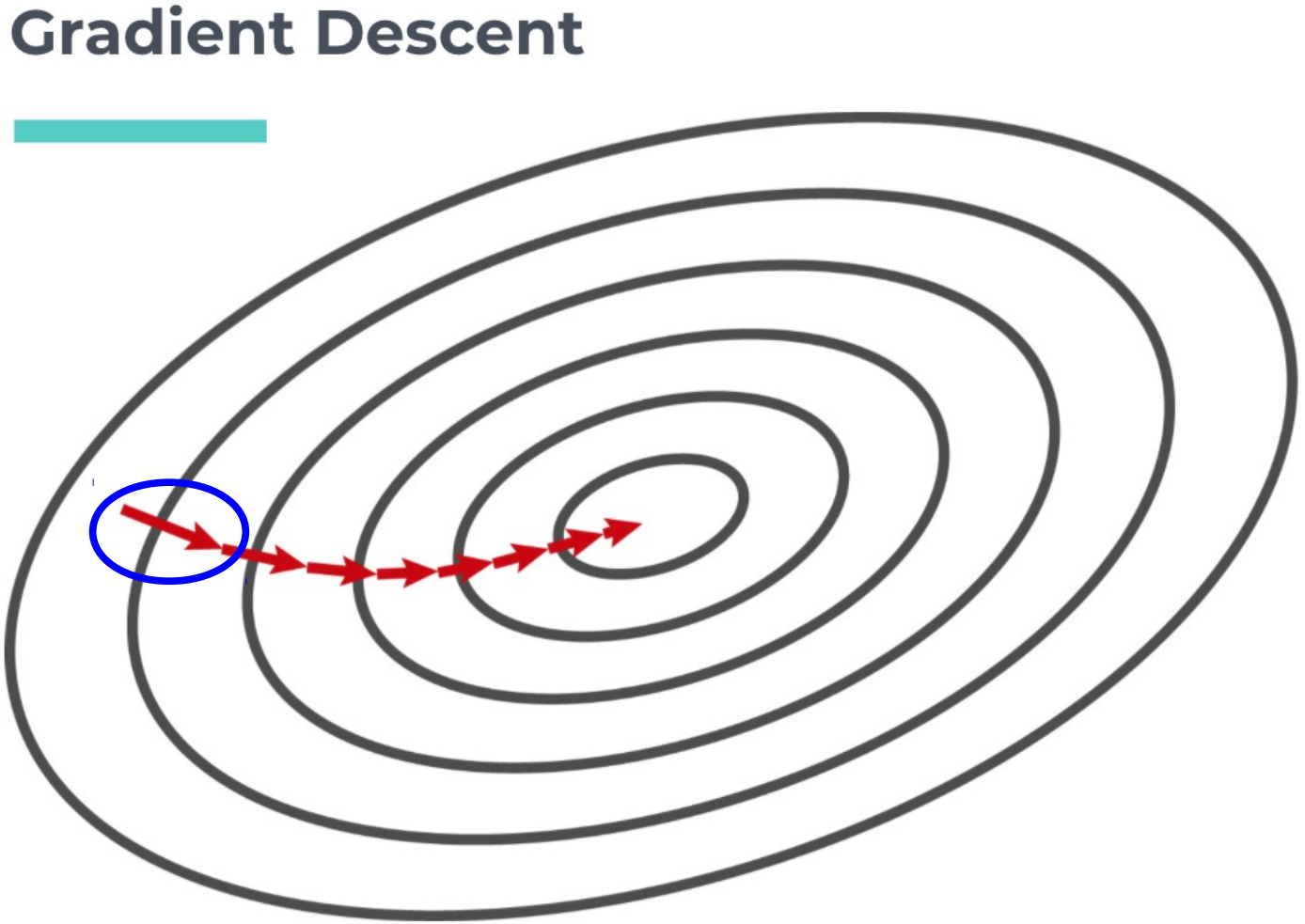
Improving the Performance of Backpropagation Learning

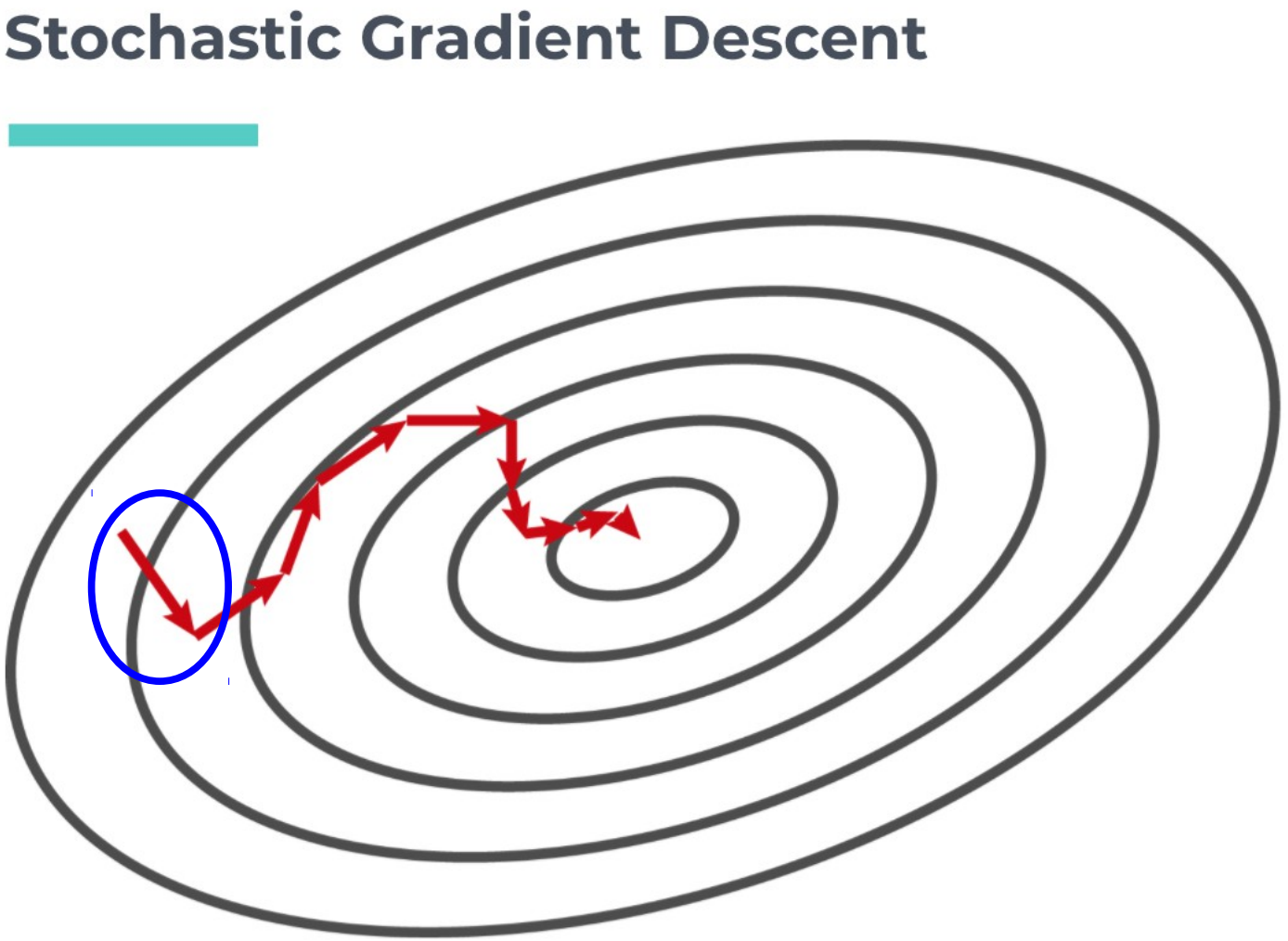
Weight Space Trajectory



Each update is based on **all *N* patterns** in the dataset

Source: Sam Abrahams (https://goo.gl/tKOvr7)

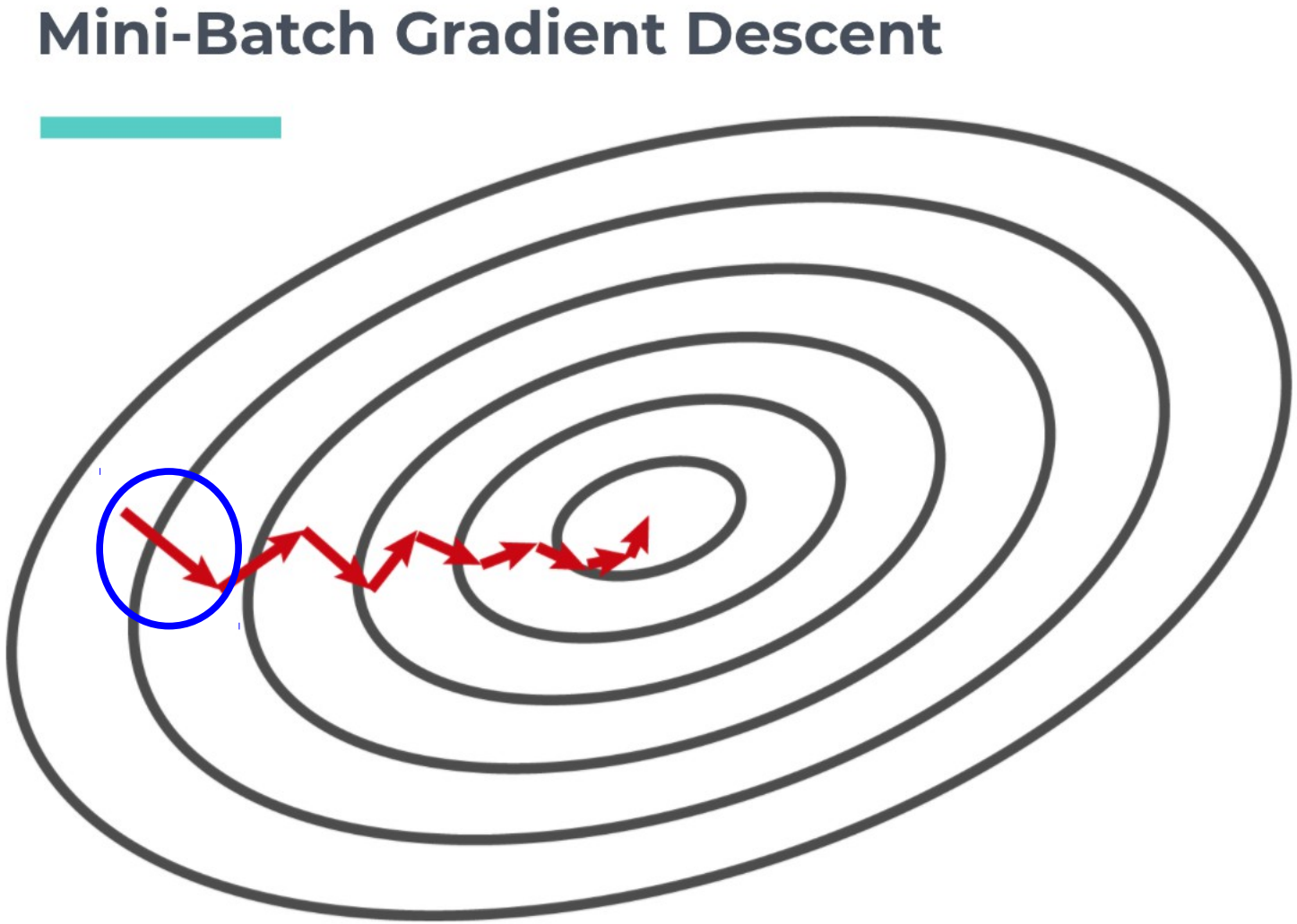
Weight Space Trajectory



Each update is based on **just 1 pattern** in the dataset

Source: Sam Abrahams (https://goo.gl/tKOvr7)

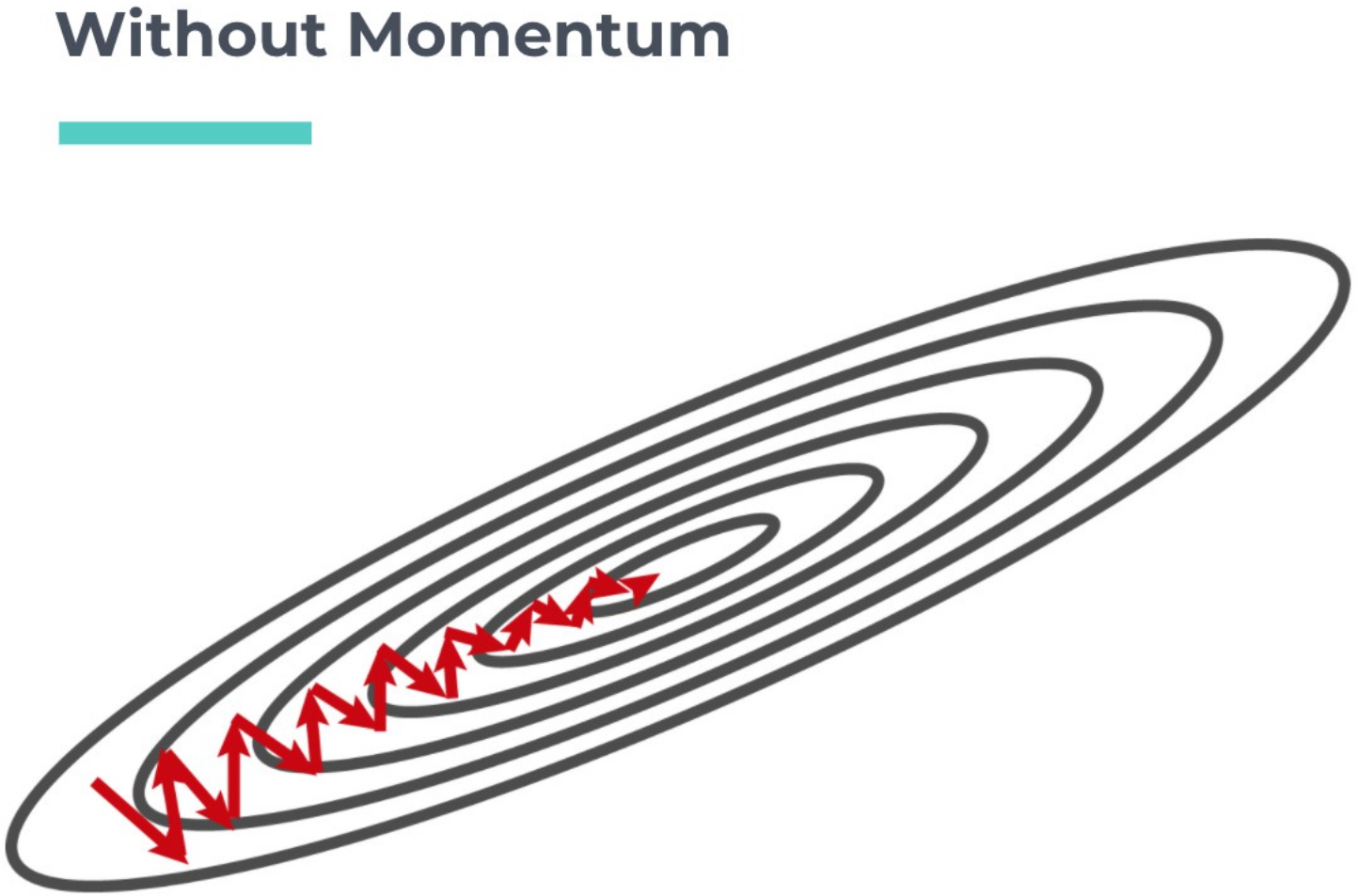
Weight Space Trajectory



Each update is based on a **mini-batch** of *m* patterns

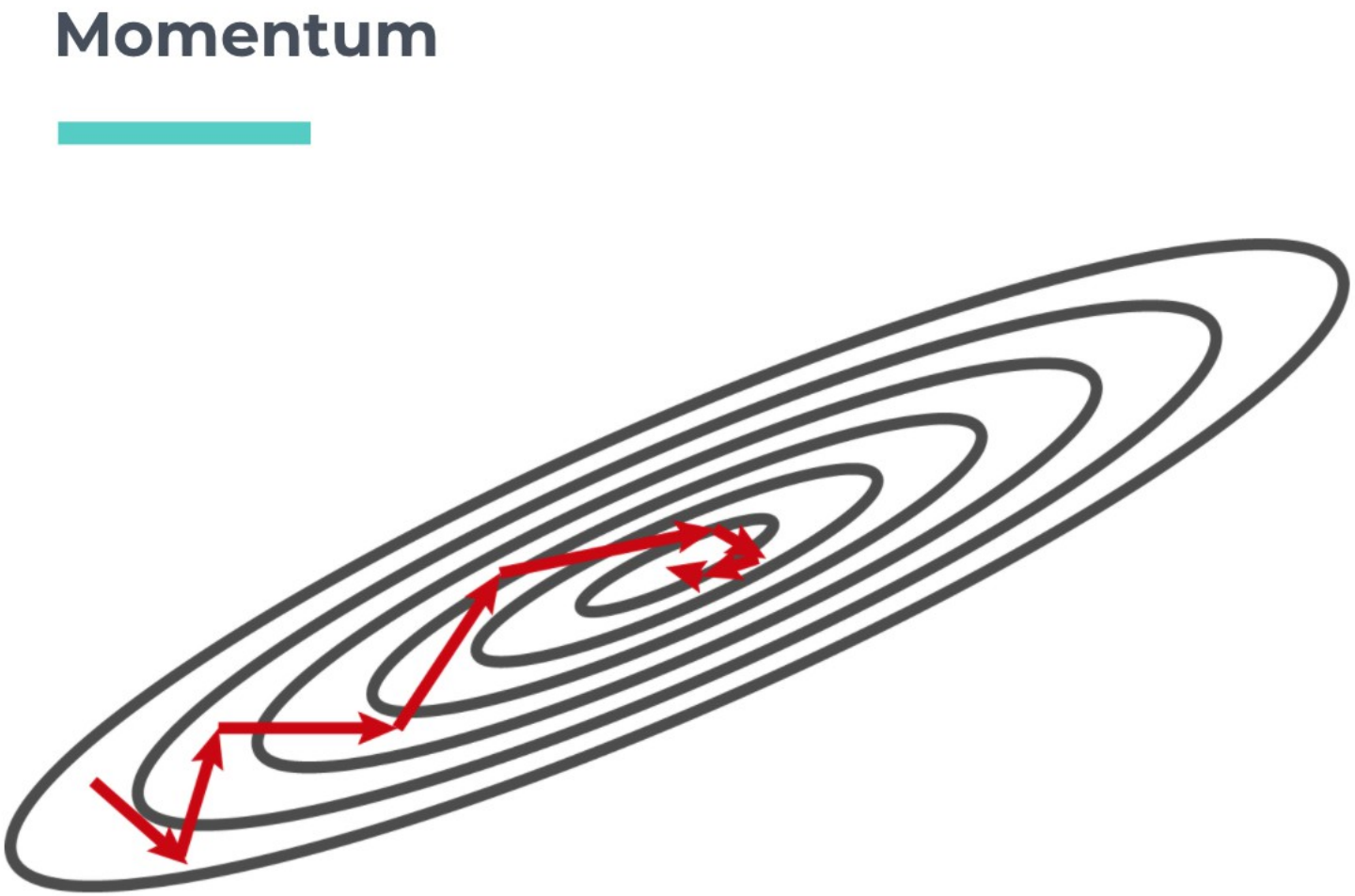
Source: Sam Abrahams (https://goo.gl/tKOvr7)

Weight Space Trajectory



Source: Sam Abrahams (https://goo.gl/tKOvr7)

Weight Space Trajectory



Source: Sam Abrahams (https://goo.gl/tKOvr7)

Adding Momentum

Momentum parameter $0 \leq \alpha \leq 1$ controls how much the **previous weight change** at time $t - 1$ contributes to the **current** amount of weight change at time t

$$\underbrace{\Delta w_{ij}(t)}_{\text{weight change on the current time step}} = \underbrace{-\eta\, \delta_i\, a_j}_{\text{weight change calculated from the current cost gradient}} + \underbrace{\alpha\, \Delta w_{ij}(t-1)}_{\text{amount the weight was changed on the previous time step}}$$

hidden → output weights

Adding Momentum

Momentum parameter $0 \leq \alpha \leq 1$ controls how much the **previous weight change** at time $t - 1$ contributes to the **current** amount of weight change at time t

$$\Delta w_{ij}(t) = -\eta\,\delta_i\,a_j + \alpha\,\Delta w_{ij}(t-1)$$ hidden → output weights

$$\Delta w_{jk}(t) = -\eta\,\delta_j\,x_k + \alpha\,\Delta w_{jk}(t-1)$$ input → hidden weights

$$\Delta b_i(t) = -\eta\,\delta_i + \alpha\,\Delta b_i(t-1)$$ output unit biases

$$\Delta b_j(t) = -\eta\,\delta_j + \alpha\,\Delta b_j(t-1)$$ hidden unit biases

Many Variations on Gradient Descent

- **SGD** (Stochastic Gradient Descent) with Momentum
- **Adagrad** (Adaptive Gradient Descent)
- **RMSprop**
- **Adam** (Adaptive Moment Estimation)
- **Nesterov** Accelerated Gradient
- **Nadam** (Nesterov-accelerated Adaptive Moment Estimation)

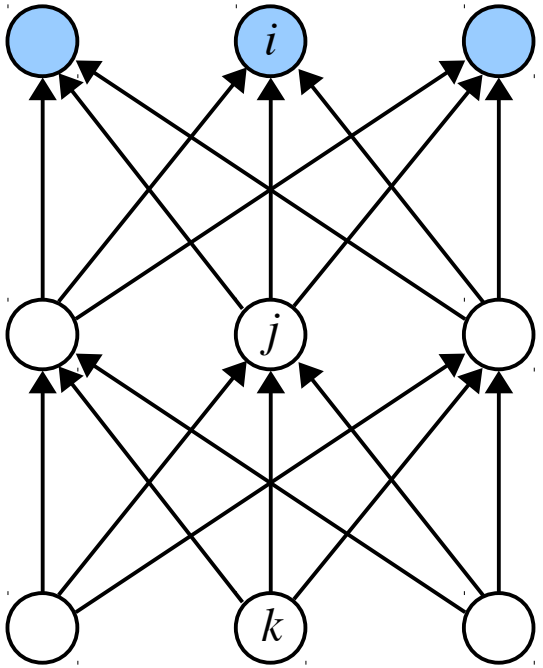
More info: https://ruder.io/optimizing-gradient-descent

Mean-Squared-Error Cost Function

$$C = \frac{1}{n}\left[\frac{1}{2}(y_1 - a_1)^2 + \frac{1}{2}(y_2 - a_2)^2 + \ldots + \frac{1}{2}(y_i - a_i)^2 + \ldots\right]$$

$y_1 \quad y_2 \cdots y_i$ ← target values

$a_1 \quad a_2 \cdots a_i$ ← output values



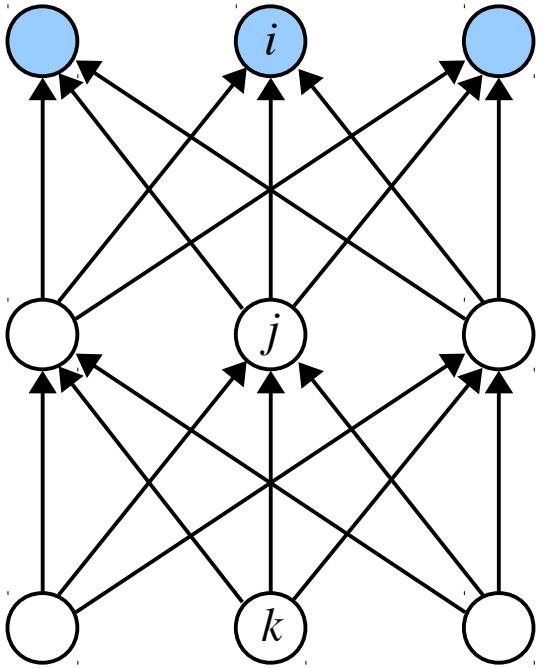
$$\frac{1}{n}\sum_i \frac{1}{2}(y_i - a_i)^2$$

Mean-Squared-Error Cost Function

$$C = \frac{1}{n}\left[\frac{1}{2}(y_1 - a_1)^2 + \frac{1}{2}(y_2 - a_2)^2 + \ldots + \frac{1}{2}(y_i - a_i)^2 + \ldots\right]$$

$y_1 \quad y_2 \cdots y_i$ ← target values

$a_1 \quad a_2 \cdots a_i$ ← output values



$$\frac{1}{n}\sum_i \frac{1}{2}(y_i - a_i)^2$$

Demo of learning with mean-squared-error

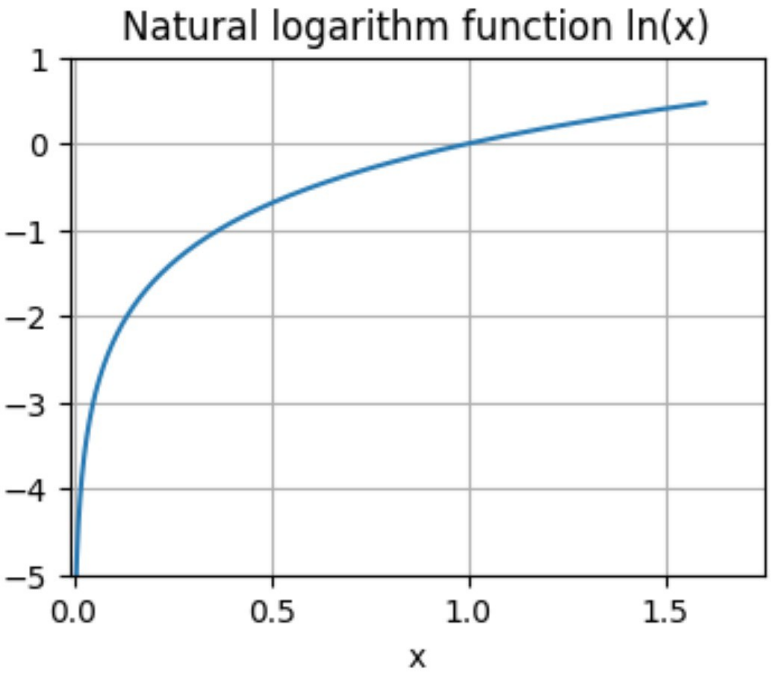
from *Neural Networks and Deep Learning*
by Michael Nielsen

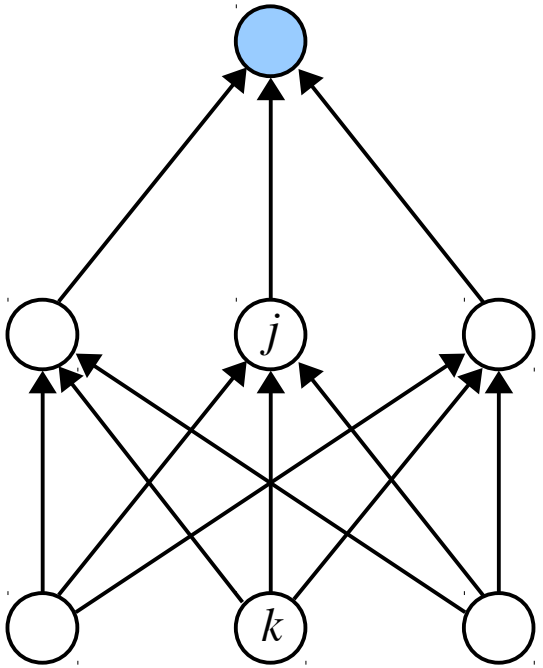
Binary Cross-Entropy Cost Function

$$C = -\left[\, y \ln a \;+\; (1-y) \ln(1-a) \,\right]$$

$y = 0$ or 1 ⟵ target value

$0 < a < 1$ ⟵ output value





$\ln a$ and $\ln(1-a)$ are **negative**

when $y = 0$ and $a \approx 0$ $\quad C \approx 0$

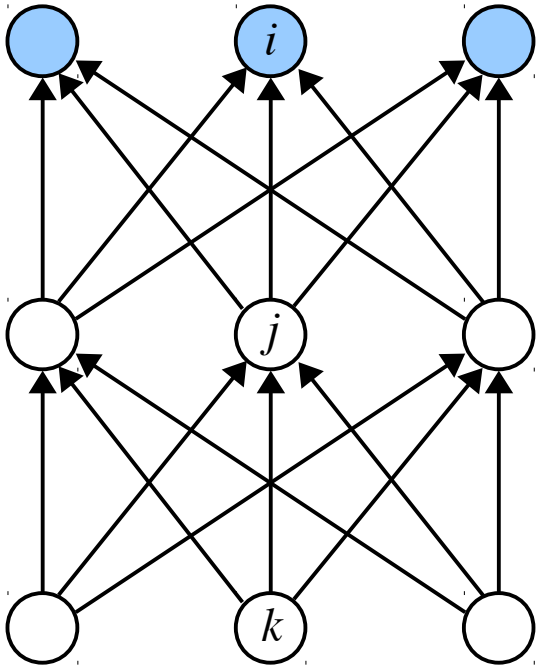
when $y = 1$ and $a \approx 1$ $\quad C \approx 0$

Categorical Cross-Entropy Cost Function

$$C = -\sum_i y_i \ln a_i + (1 - y_i) \ln(1 - a_i)$$

$y_1 \quad y_2 \quad \cdots \quad y_i$ ← target values

$a_1 \quad a_2 \quad \cdots \quad a_i$ ← output values



$$-\Big[\; y_1 \ln a_1 + (1 - y_1) \ln(1 - a_1)$$

$$+\; y_2 \ln a_2 + (1 - y_2) \ln(1 - a_2) \;+\; \ldots$$

$$+\; y_i \ln a_i + (1 - y_i) \ln(1 - a_i) \;+\; \ldots \;\Big]$$

Categorical Cross-Entropy Cost Function

$$C = -\sum_i y_i \ln a_i + (1 - y_i) \ln(1 - a_i)$$

$$-\Big[\; y_1 \ln a_1 + (1 - y_1) \ln(1 - a_1) \;+\; y_2 \ln a_2 + (1 - y_2) \ln(1 - a_2) \;+\; \ldots$$
$$+\; y_i \ln a_i + (1 - y_i) \ln(1 - a_i) \;+\; \ldots \;\Big]$$

$$\frac{\partial C}{\partial a_i} = -\left[\frac{y_i}{a_i} + \frac{(1 - y_i)}{(1 - a_i)} \cdot (-1)\right] = \frac{-y_i(1 - a_i)}{a_i(1 - a_i)} + \frac{a_i(1 - y_i)}{a_i(1 - a_i)}$$

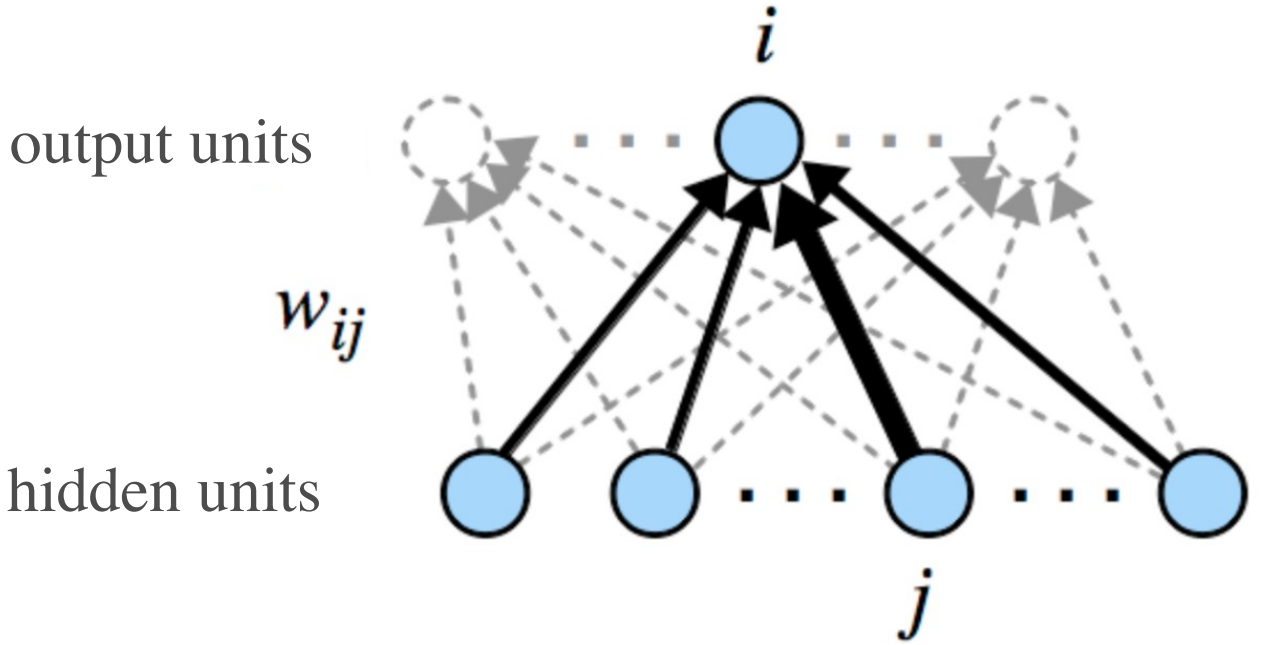
$$= \frac{-y_i + a_i y_i + a_i - a_i y_i}{a_i(1 - a_i)} = \boxed{\frac{a_i - y_i}{a_i(1 - a_i)}}$$

Hidden → Output Weights

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial z_i}{\partial w_{ij}} \times \frac{\partial a_i}{\partial z_i} \times \frac{\partial C}{\partial a_i}$$

different!

$$\boxed{\text{influence of } w_{ij} \text{ on } C} = \overbrace{a_j}^{\text{same as before}} \times \overbrace{a_i(1-a_i)}^{\text{same as before}} \times \frac{a_i - y_i}{a_i(1-a_i)}$$



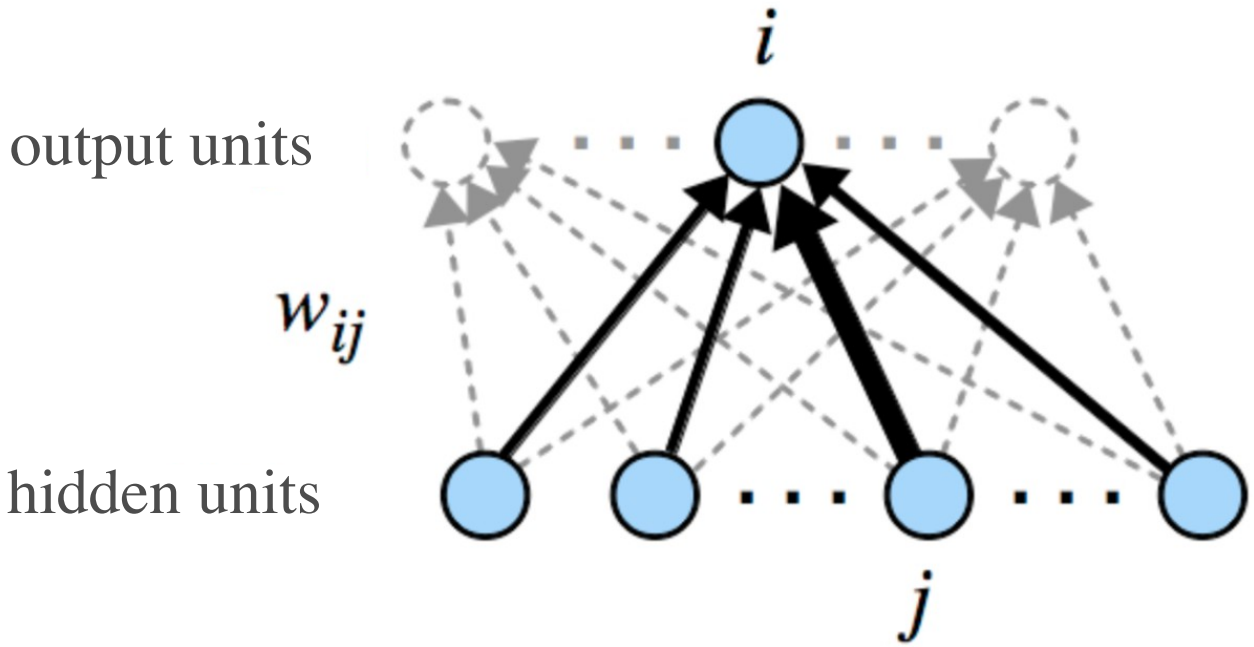
Hidden → Output Weights

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial z_i}{\partial w_{ij}} \times \frac{\partial a_i}{\partial z_i} \times \frac{\partial C}{\partial a_i}$$

different!

same as before same as before

$$\boxed{\text{influence of } w_{ij} \text{ on } C} = a_j \times \cancel{a_i(1-a_i)} \times \cancel{\frac{a_i - y_i}{a_i(1-a_i)}}$$

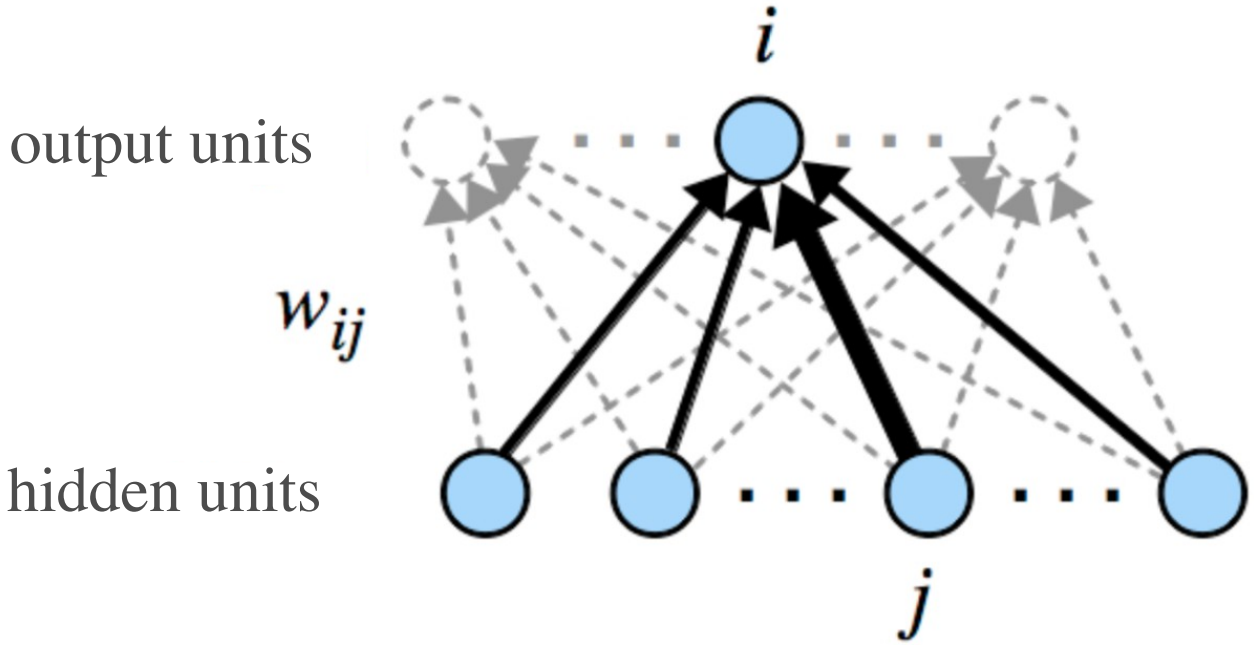


Hidden → Output Weights

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial z_i}{\partial w_{ij}} \times \frac{\partial a_i}{\partial z_i} \times \frac{\partial C}{\partial a_i}$$

$$\frac{\partial C}{\partial w_{ij}} = \boxed{(a_i - y_i)}\, a_j$$

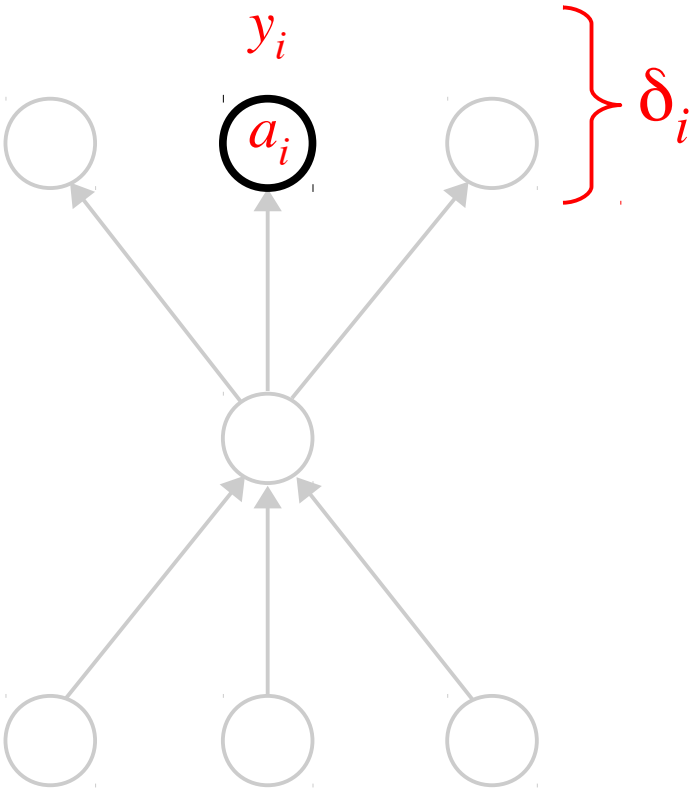
this is now δ_i

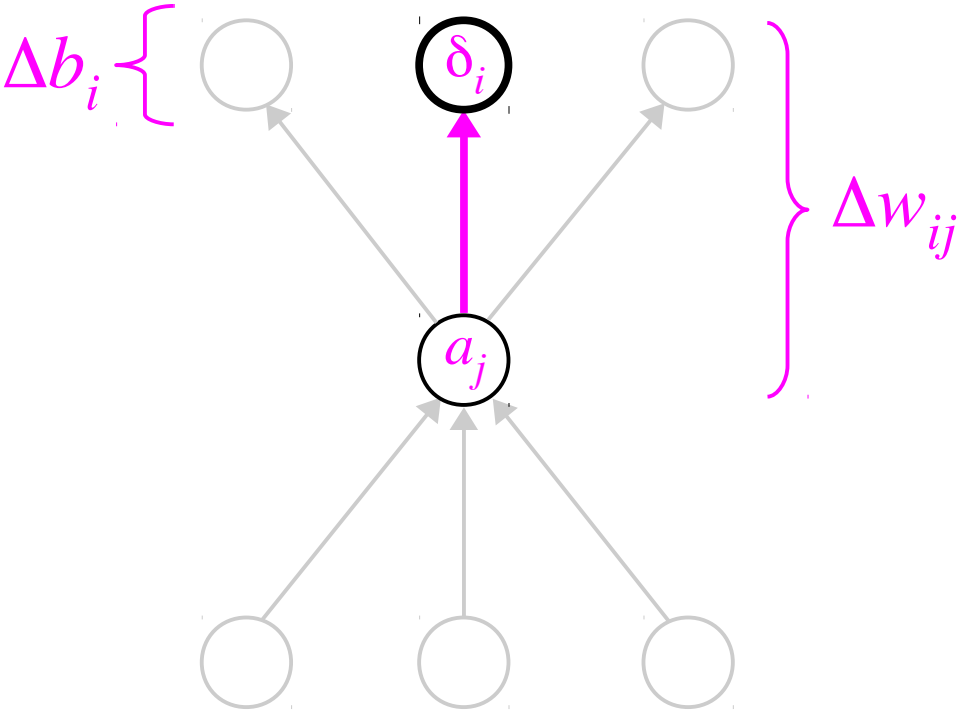


New Update Rule for Output Unit i

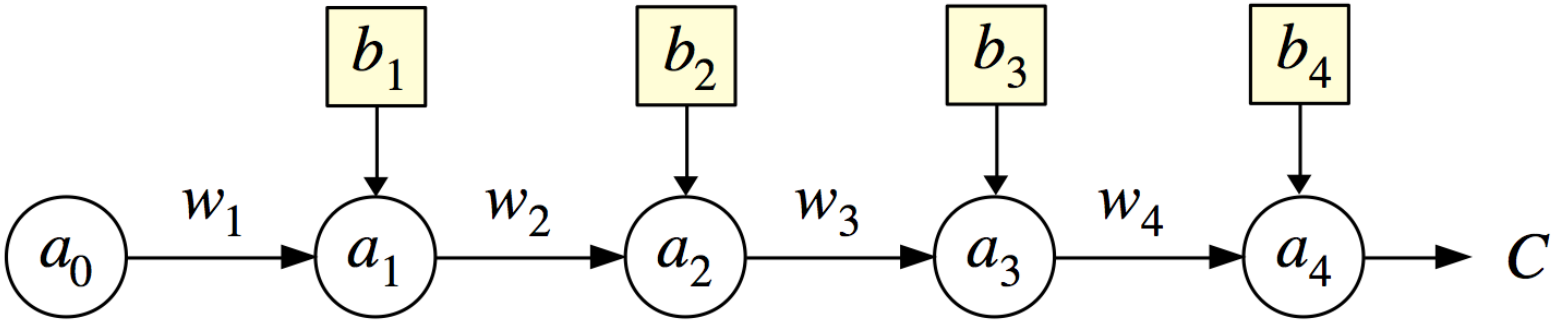
$$\delta_i = a_i - y_i \qquad 0 < \eta < 1$$

$$\Delta w_{ij} = -\eta\, \delta_i\, a_j \qquad \Delta b_i = -\eta\, \delta_i$$





The Problem of Vanishing Gradients



w_i = weight into the i^{th} neuron

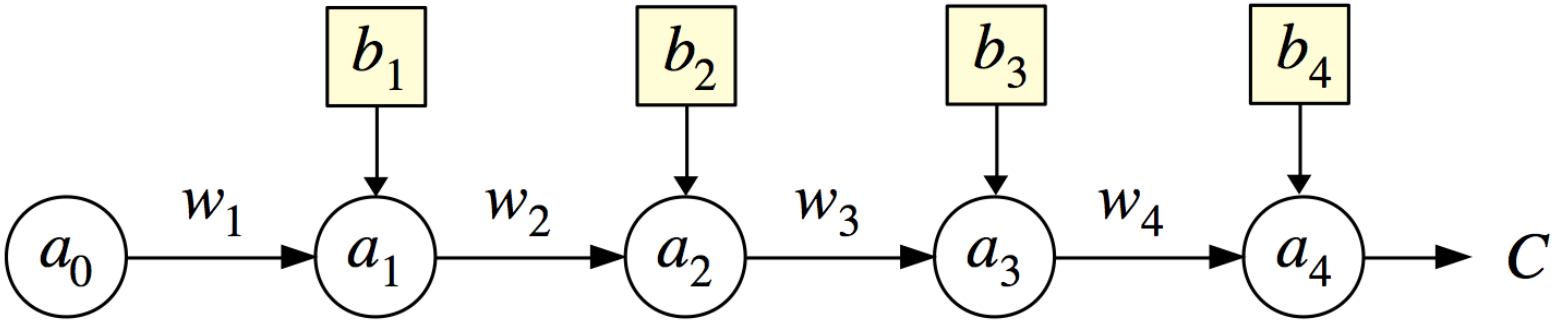
b_i = bias of the i^{th} neuron

z_i = sum of inputs into the i^{th} neuron = $w_i \cdot a_{i-1} + b_i$

a_i = output activation of the i^{th} neuron = $\sigma(z_i) = \dfrac{1}{1 + e^{-z_i}}$

$\dfrac{\partial a_i}{\partial z_i}$ = derivative of the activation function = $\sigma'(z_i) = a_i(1 - a_i)$

The Problem of Vanishing Gradients



$$\frac{\partial a_i}{\partial z_i} = \sigma'(z_i) = a_i(1 - a_i)$$

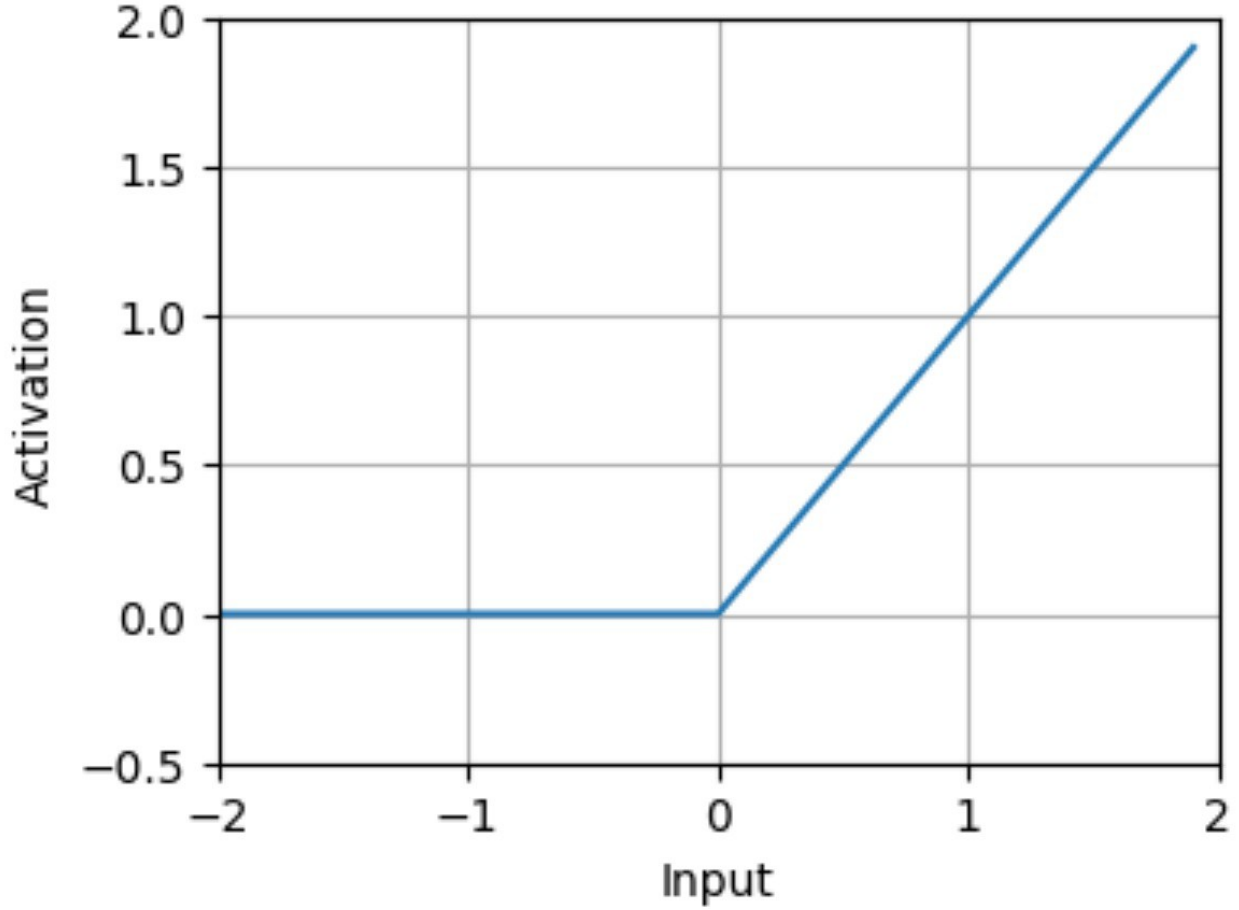
Update rule for bias b_1: $\Delta b_1 = -\eta \frac{\partial C}{\partial b_1}$

$$\frac{\partial C}{\partial b_1} = \frac{\partial z_1}{\partial b_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial C}{\partial a_4}$$

$$= 1 \times \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

What happens when hidden unit activations a_i are close to 0 or 1?

Rectified Linear Unit (ReLU)



$$\frac{\partial a_i}{\partial z_i} = \text{derivative of the ReLU function} = \begin{cases} 1 & \text{when input} > 0 \\ 0 & \text{when input} < 0 \end{cases}$$