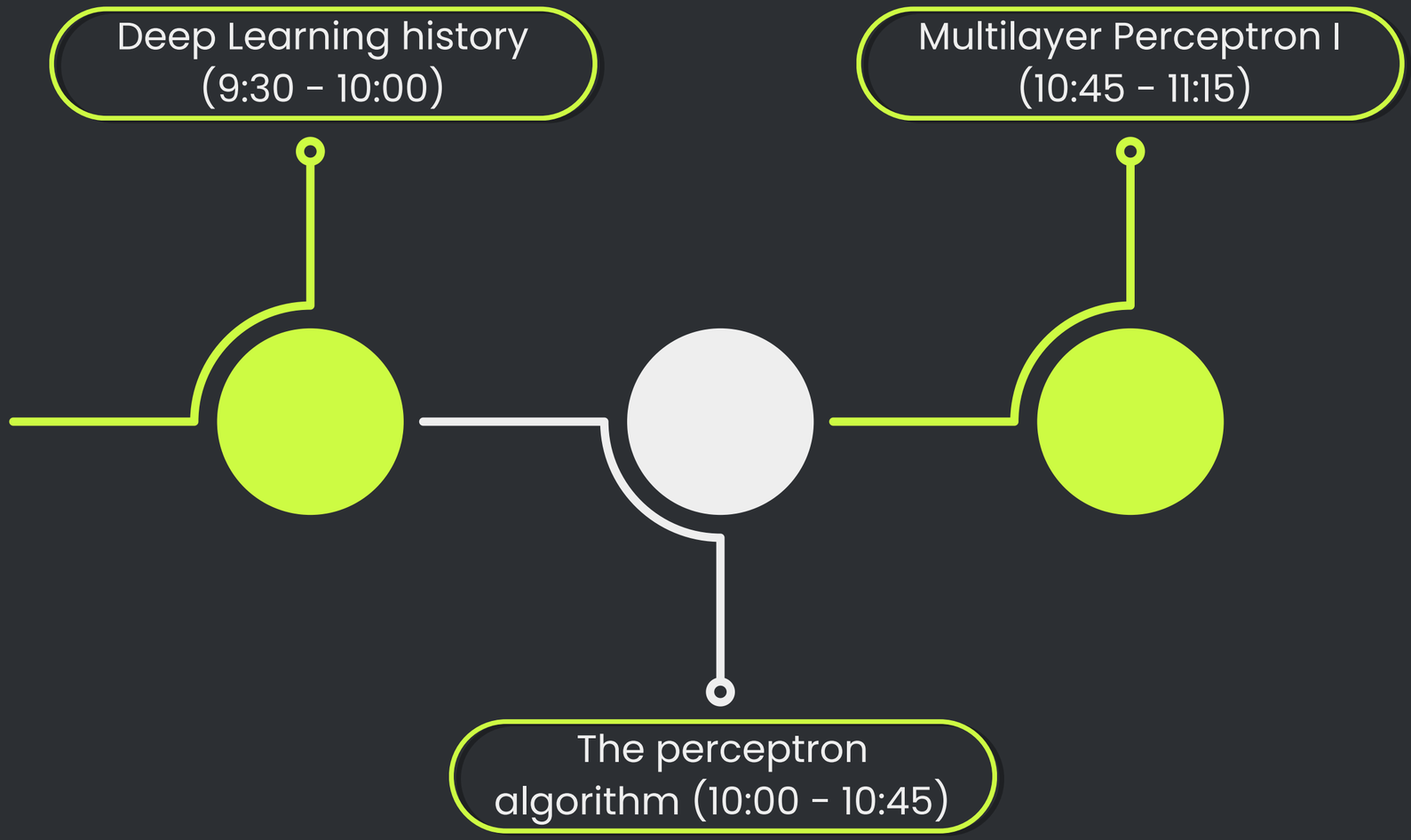
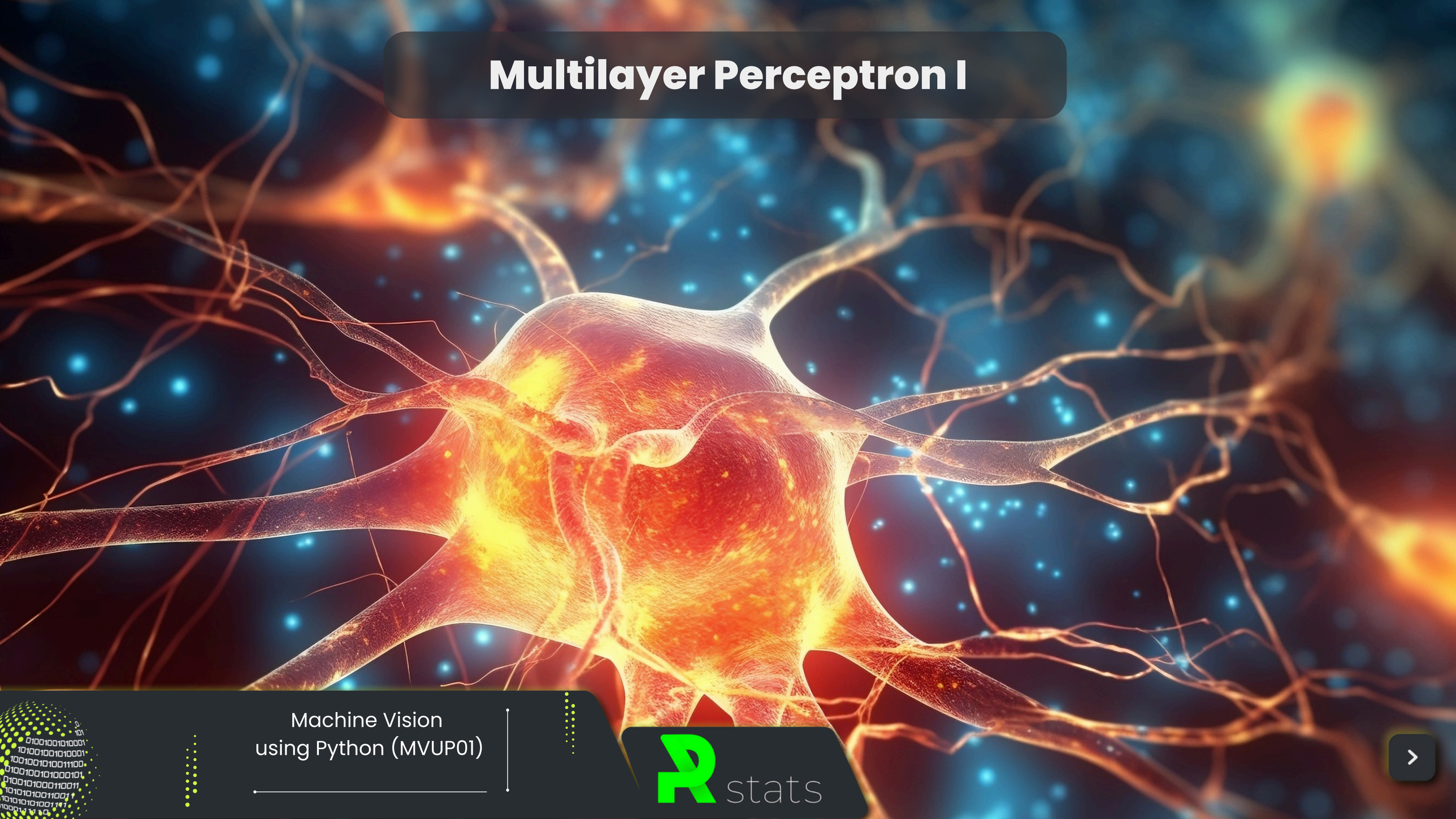
Morning Session (9:30 - 12:00)



Multilayer Perceptron I



Machine Vision
using Python (MVUP01)

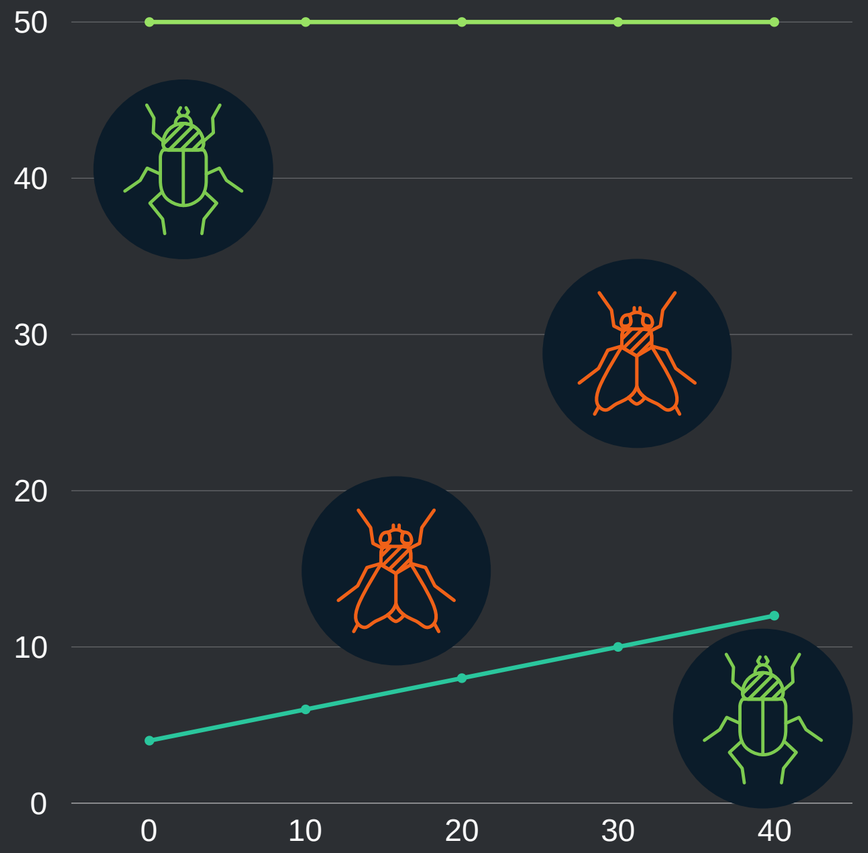
stats

Multilayer Perceptron I

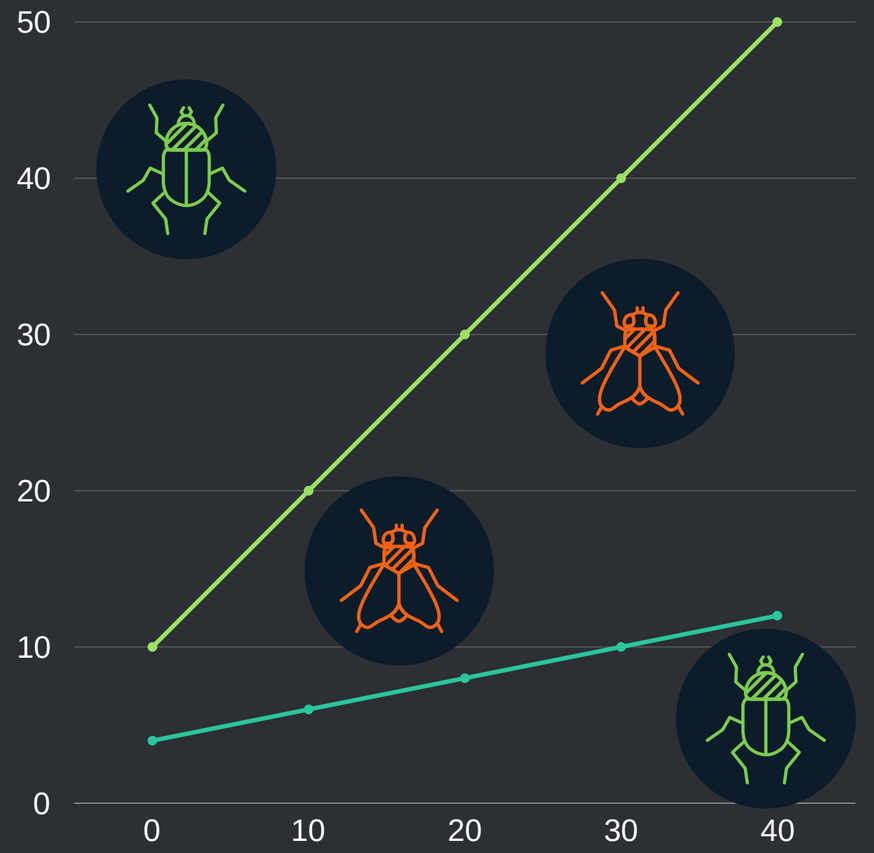
Machine Vision
using Python (MVUP01)

stats

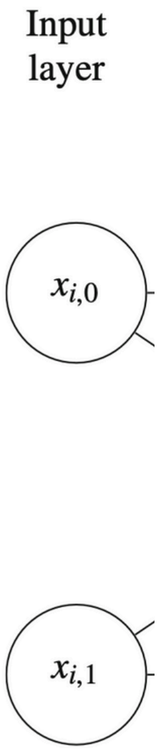
Multilayer Perceptron I



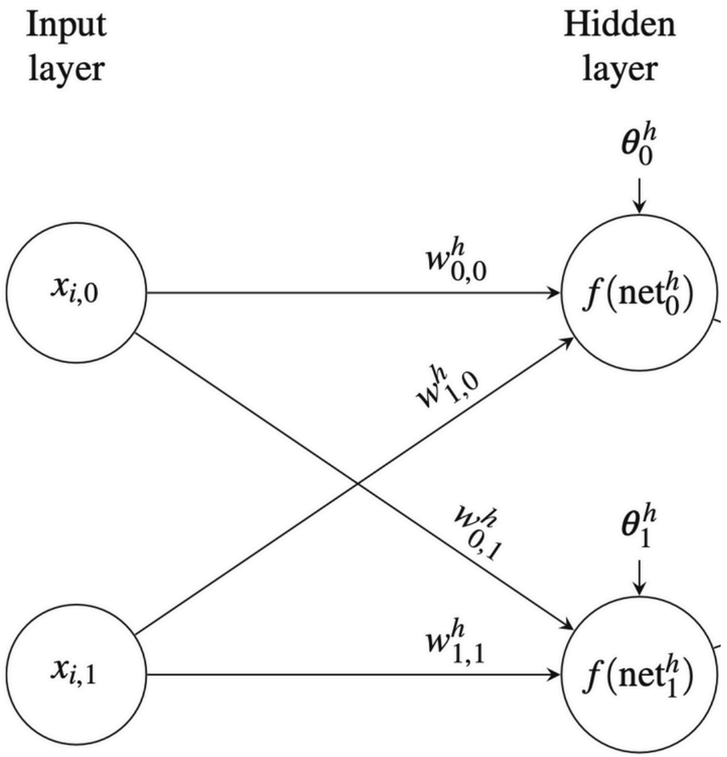
Multilayer Perceptron I



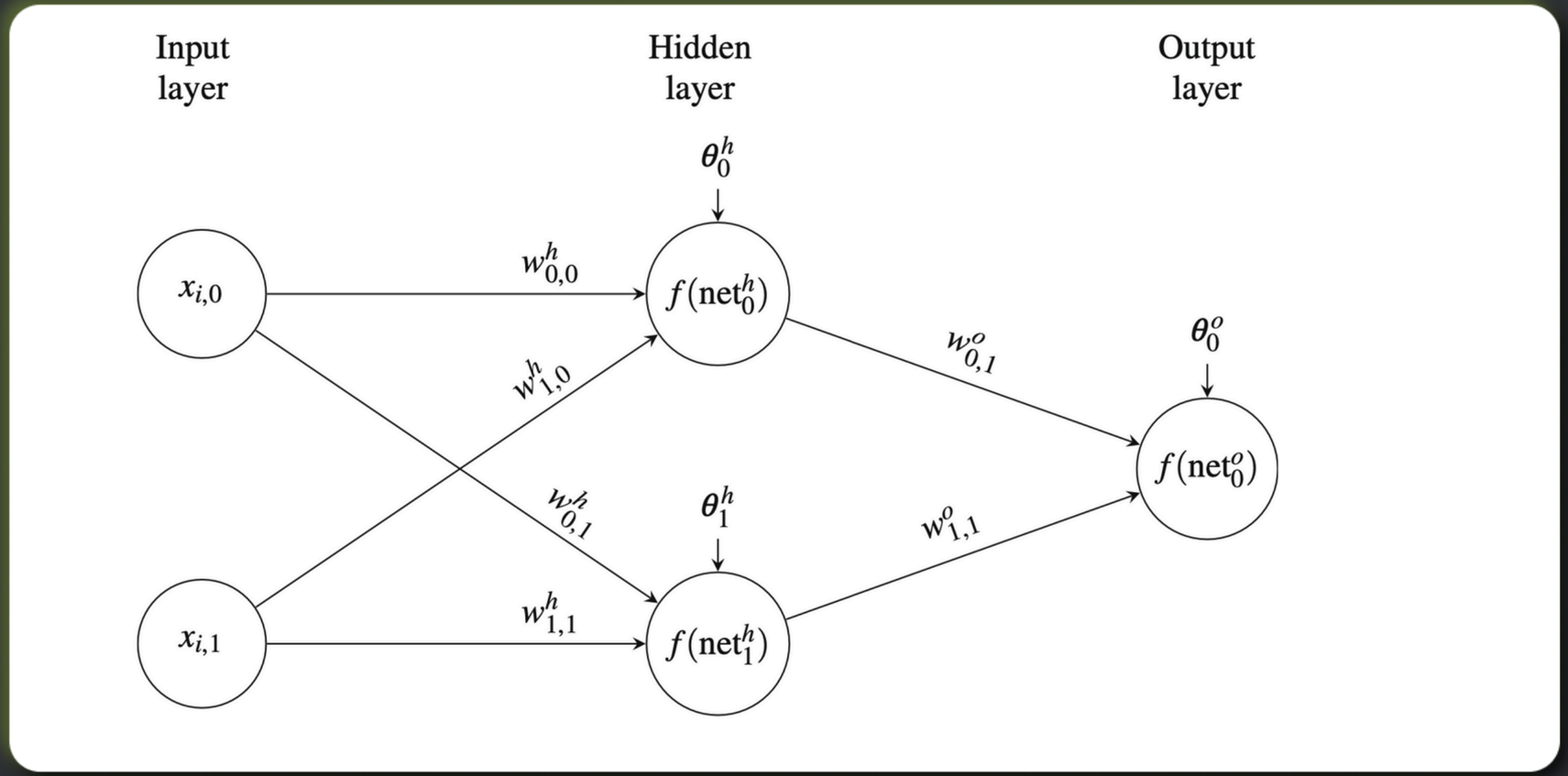
Multilayer Perceptron I



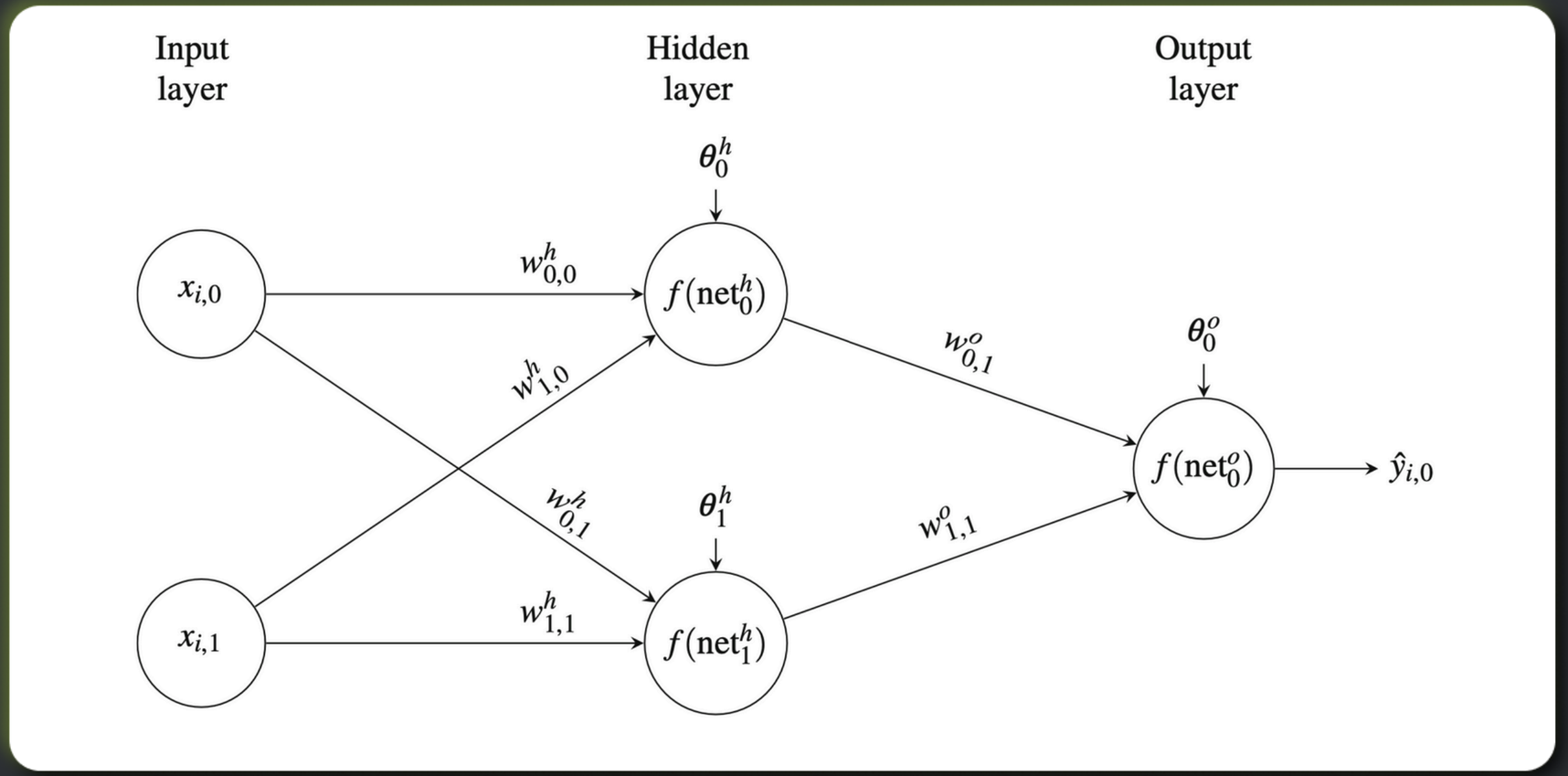
Multilayer Perceptron I



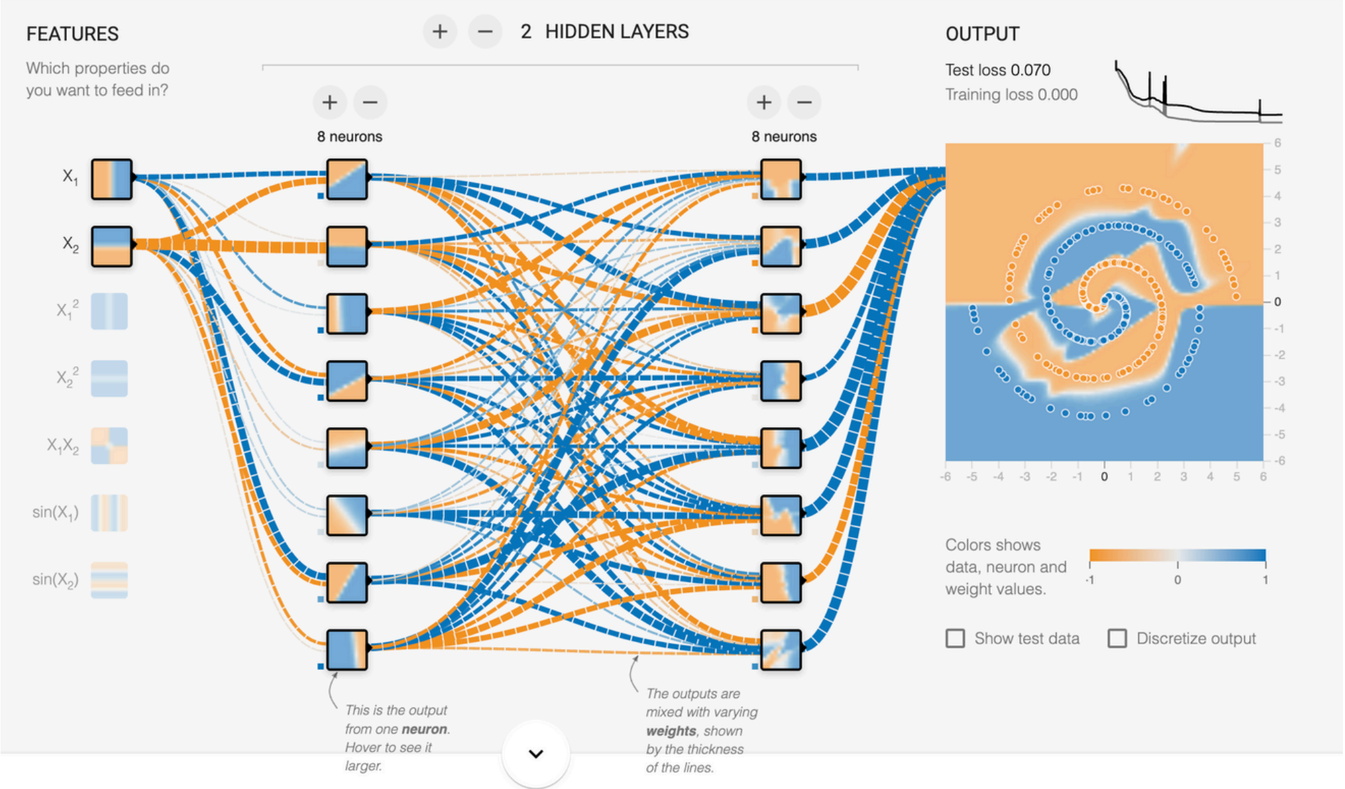
Multilayer Perceptron I



Multilayer Perceptron I

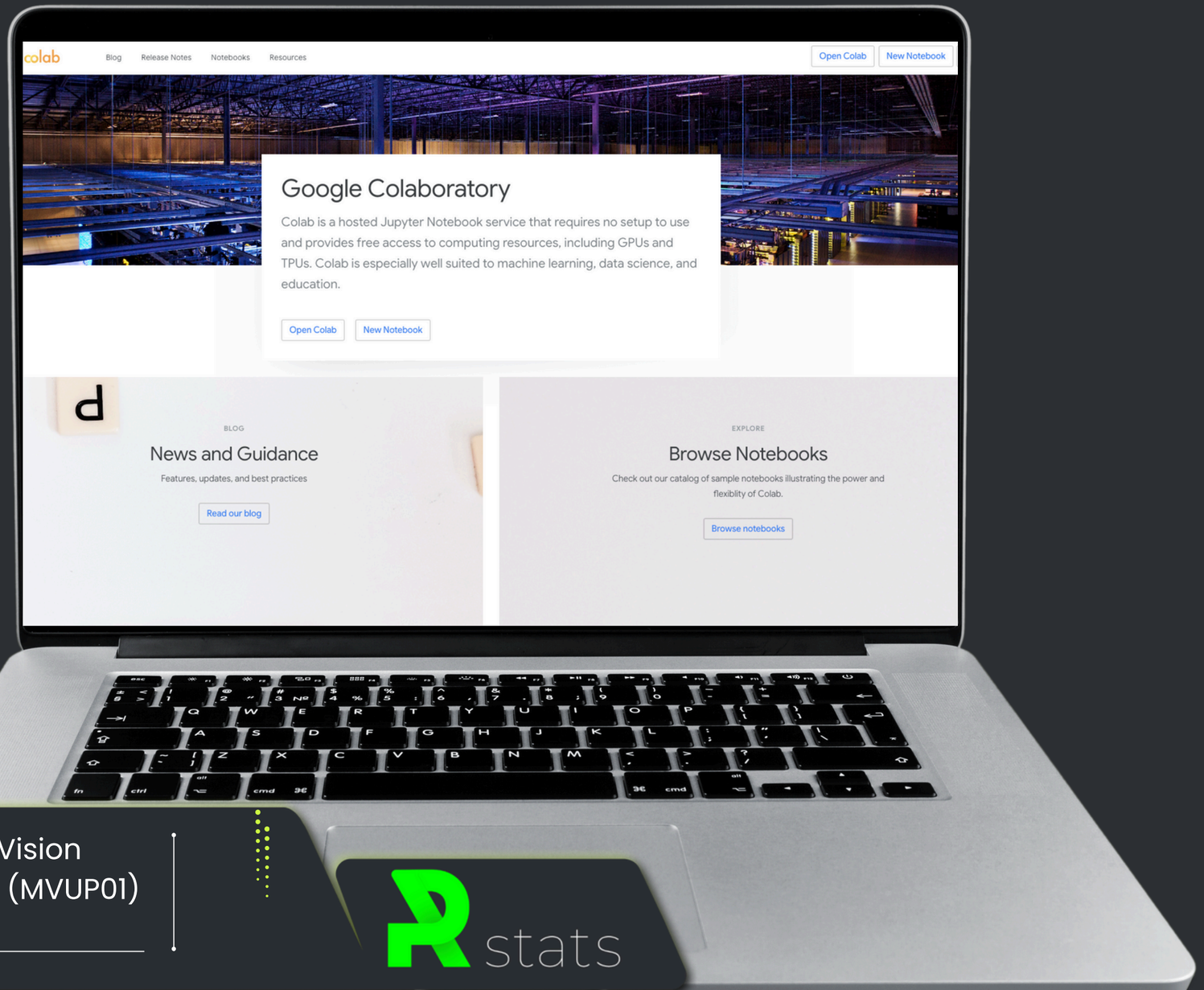


Multilayer Perceptron I



Fundamentals of Digital Images

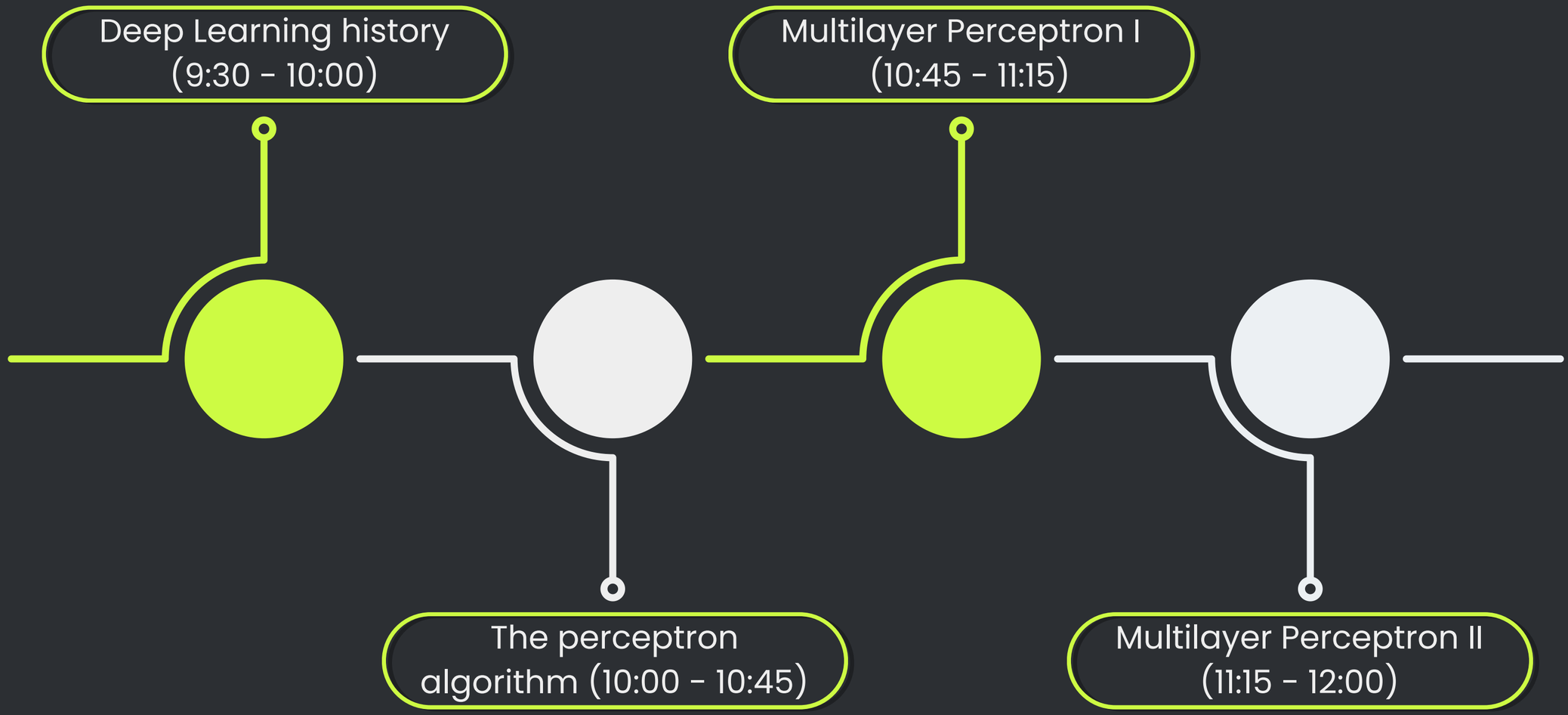
Hands-on activity:



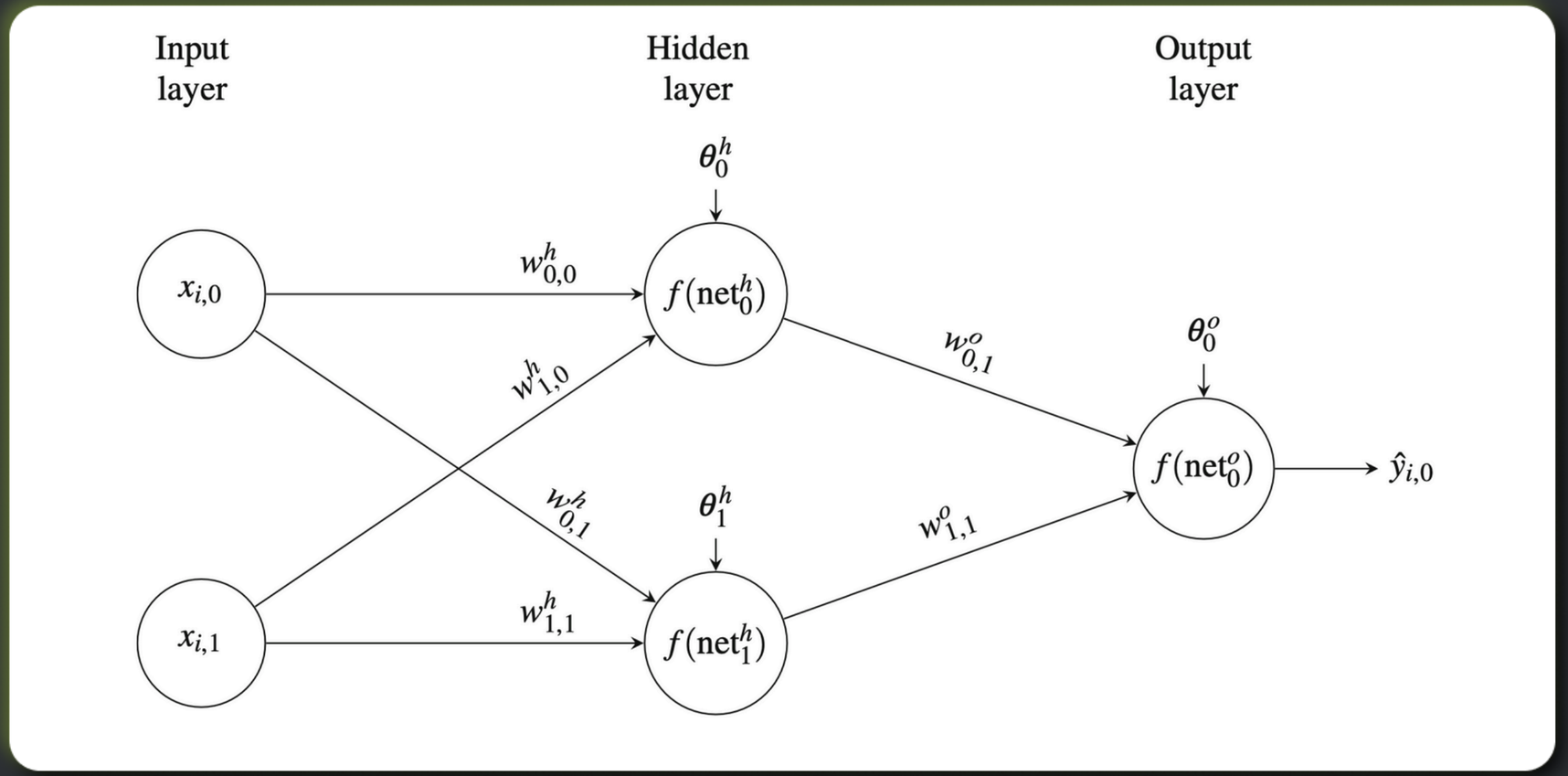
Machine Vision
using Python (MVUP01)

stats

Morning Session (9:30 - 12:00)



Multilayer Perceptron I



Multilayer Perceptron I

We formalize MLP with the squared-error function:

$$E_i^2 = (y_i - f(\text{net}_i))^2 , \tag{1.19}$$

in which $f(\text{net}_i)$ corresponds to the output MLP produces for example i. Rewriting term $f(\text{net}_i)$:

Multilayer Perceptron I

We formalize MLP with the squared-error function:

$$E_i^2 = (y_i - f(\text{net}_i))^2 , \tag{1.19}$$

in which $f(\text{net}_i)$ corresponds to the output MLP produces for example i. Rewriting term $f(\text{net}_i)$:

$$E_i^2 = \sum_k \left(y_{i,k} - f_k^o\left(\text{net}_{i,k}^o\right)\right)^2$$

Multilayer Perceptron I

We formalize MLP with the squared-error function:

$$E_i^2 = (y_i - f(\text{net}_i))^2 , \tag{1.19}$$

in which $f(\text{net}_i)$ corresponds to the output MLP produces for example i. Rewriting term $f(\text{net}_i)$:

$$E_i^2 = \sum_k \left(y_{i,k} - f_k^o\left(\text{net}_{i,k}^o\right)\right)^2 = \left(y_{i,k} - f_k^o\left[\sum_j f_j^h(\text{net}_j^h) w_{k,j}^o + \theta_k^o\right]\right)^2 , \tag{1.20}$$

Multilayer Perceptron I

1. Equation (1.21) to adapt every weight $w^h_{j,l}$ connecting input neuron l to hidden neuron j;
2. Equation (1.22) to adapt every weight $w^o_{k,j}$ connecting hidden neuron j to output neuron k.

$$w^h_{j,l}(t+1) = w^h_{j,l}(t) - \eta \frac{\partial E_i^2}{\partial w^h_{j,l}} \tag{1.21}$$

$$w^o_{k,j}(t+1) = w^o_{k,j}(t) - \eta \frac{\partial E_i^2}{\partial w^o_{k,j}} \tag{1.22}$$

Multilayer Perceptron I

To complement, Eqs. (1.23) and (1.24) provide the GD method to adapt the free variable θ for every neuron at either the hidden or the output layer along iterations:

$$\theta_j^h(t+1) = \theta_j^h(t) - \eta \frac{\partial E_i^2}{\partial \theta_j^h}, \tag{1.23}$$

$$\theta_k^o(t+1) = \theta_k^o(t) - \eta \frac{\partial E_i^2}{\partial \theta_k^o}. \tag{1.24}$$

Multilayer Perceptron I

Despite the Gradient Descent equations are the same for weights in both layers as well as for thetas, the partial derivatives:

$$\frac{\partial E_i^2}{\partial w_{j,l}^h}, \quad \frac{\partial E_i^2}{\partial w_{k,j}^o}, \quad \frac{\partial E_i^2}{\partial \theta_j^h}, \text{ and } \frac{\partial E_i^2}{\partial \theta_k^o}$$

change. Consequently, we must compute the partial derivatives for this problem XOR to exemplify the MLP formulation, however this solution still supports a general-purpose algorithm.

Multilayer Perceptron I

The partial derivatives for the output layer are found in advance, once they are simpler to obtain:

$$\frac{\partial E_i^2}{\partial w_{k,j}^o} = \frac{\partial \sum_k (y_{i,k} - \hat{y}_{i,k})^2}{\partial w_{k,j}^o}$$

Multilayer Perceptron I

The partial derivatives for the output layer are found in advance, once they are simpler to obtain:

$$\begin{aligned}\frac{\partial E_i^2}{\partial w_{k,j}^o} &= \frac{\partial \sum_k (y_{i,k} - \hat{y}_{i,k})^2}{\partial w_{k,j}^o} \\ &= \frac{\sum_k \left(y_{i,k} - f_k^o \left(\sum_j f_j^h(\text{net}_j^h) w_{k,j}^o + \theta_k^o\right)\right)^2}{\partial w_{k,j}^o},\end{aligned} \tag{1.25}$$

Multilayer Perceptron I

The differentiation employs the chain rule as follows:

$$\frac{\partial E_i^2}{\partial w_{k,j}^o} = \sum_k 2\left(y_{i,k} - \hat{y}_{i,k}\right) \frac{\partial \left(y_{i,k} - \hat{y}_{i,k}\right)}{\partial w_{k,j}^o},$$

as $y_{i,k}$ is a constant defining the expected class to be produced by output neuron k, assuming zero while deriving in terms of $w_{k,j}^o$:

$$\frac{\partial E_i^2}{\partial w_{k,j}^o} = \sum_k 2(y_{i,k} - \hat{y}_{i,k}) \frac{\partial - \hat{y}_{i,k}}{\partial w_{k,j}^o},$$

Multilayer Perceptron I

$$\frac{\partial E_i^2}{\partial w_{k,j}^o} = \sum_k 2(y_{i,k} - \hat{y}_{i,k}) \frac{\partial - \hat{y}_{i,k}}{\partial w_{k,j}^o},$$

thus, we still need to solve the derivative $\frac{\partial - \hat{y}_{i,k}}{\partial w_{k,j}^o}$. As the reader may recall, $\hat{y}_{i,k} = f_k^o(\text{net}_{i,k}^o)$, i.e., the output value produced by output neuron k is the result of the sigmoid function, defined in Eq. (1.18):

Multilayer Perceptron I

$$\frac{\partial - \hat{y}_{i,k}}{\partial w^o_{k,j}} = \frac{\partial - f^o_k(\text{net}^o_{i,k})}{\partial w^o_{k,j}}$$

Multilayer Perceptron I

$$\frac{\partial - \hat{y}_{i,k}}{\partial w^o_{k,j}} = \frac{\partial - f^o_k(\text{net}^o_{i,k})}{\partial w^o_{k,j}}$$

$$= \frac{\partial - (1 + e^{-\text{net}^o_{i,k}})^{-1}}{\partial w^o_{k,j}}$$

Multilayer Perceptron I

$$
\begin{aligned}
\frac{\partial - \hat{y}_{i,k}}{\partial w^o_{k,j}} &= \frac{\partial - f^o_k(\text{net}^o_{i,k})}{\partial w^o_{k,j}} \\
&= \frac{\partial - (1 + e^{-\text{net}^o_{i,k}})^{-1}}{\partial w^o_{k,j}} \\
&= \frac{\partial - \left(1 + e^{-\left[\sum_j f^h_j(\text{net}^h_j) w^o_{k,j} + \theta^o_k\right]}\right)^{-1}}{\partial w^o_{k,j}}.
\end{aligned}
$$

Multilayer Perceptron I

We know the sigmoid function (Eq. (1.18)) has the following derivative in terms of net:

$$\frac{\partial f(\text{net})}{\partial \text{net}} = f(\text{net})(1 - f(\text{net})),$$

Multilayer Perceptron I

We know the sigmoid function (Eq. (1.18)) has the following derivative in terms of net:

$$\frac{\partial f(\text{net})}{\partial \text{net}} = f(\text{net})(1 - f(\text{net})),$$

what simplifies our formulation:

Multilayer Perceptron I

We know the sigmoid function (Eq. (1.18)) has the following derivative in terms of net:

$$\frac{\partial f(\text{net})}{\partial \text{net}} = f(\text{net})(1 - f(\text{net})),$$

what simplifies our formulation:

$$\frac{\partial - f_k^o(\text{net}_{i,k}^o)}{\partial w_{k,j}^o} = -\left[f_k^o(\text{net}_{i,k}^o)(1 - f_k^o(\text{net}_{i,k}^o)) \frac{\partial \text{net}_{i,k}^o}{\partial w_{k,j}^o} \right],$$

Multilayer Perceptron I

what simplifies our formulation:

$$\frac{\partial - f_k^o(\text{net}_{i,k}^o)}{\partial w_{k,j}^o} = -\left[f_k^o(\text{net}_{i,k}^o)(1 - f_k^o(\text{net}_{i,k}^o)) \frac{\partial \text{net}_{i,k}^o}{\partial w_{k,j}^o} \right],$$

so, we still need to find the following partial derivative:

$$\frac{\partial \text{net}_{i,k}^o}{\partial w_{k,j}^o} = \frac{\partial \sum_j f_j^h(\text{net}_{i,j}^h) w_{k,j}^o + \theta_k^o}{\partial w_{k,j}^o} = f_j^h(\text{net}_{i,j}^h).$$

Multilayer Perceptron I

Connecting all terms, the update rule for weights at the output layer is:

$$w^o_{k,j}(t+1) = w^o_{k,j}(t) - \eta \frac{\partial E_i^2}{\partial w^o_{k,j}}$$

$$w^o_{k,j}(t+1) = w^o_{k,j}(t) - \eta\, 2(y_i - \hat{y}_{i,k}) \frac{\partial - \left[f^o_k(\text{net}^o_{i,k}) \right]}{\partial w^o_{k,j}}$$

Multilayer Perceptron I

$$w^o_{k,j}(t+1) = w^o_{k,j}(t)$$

$$- \eta\, 2(y_i - f^o_k(\text{net}^o_{i,k}))\left(-\left[f^o_k(\text{net}^o_{i,k})(1 - f^o_k(\text{net}^o_{i,k}))\right]\right)\frac{\partial \text{net}^o_{i,k}}{\partial w^o_{k,j}}$$

$$w^o_{k,j}(t+1) = w^o_{k,j}(t)$$

$$- \eta\, 2(y_i - f^o_k(\text{net}^o_{i,k}))\left(-\left[f^o_k(\text{net}^o_{i,k})(1 - f^o_k(\text{net}^o_{i,k}))\right]\right) f^h_j(\text{net}^h_{i,j}).$$

Multilayer Perceptron I

Similarly, the Gradient Descent formulated for theta from output neuron k is:

$$\theta_k^o(t+1) = \theta_k^o(t) - \eta\, 2(y_i - f_k^o(\text{net}_{i,k}^o))\left(-\left[f_k^o(\text{net}_{i,k}^o)(1 - f_k^o(\text{net}_{i,k}^o))\right]\right) 1,$$

Multilayer Perceptron I

Similarly, the Gradient Descent formulated for theta from output neuron k is:

$$\theta_k^o(t+1) = \theta_k^o(t) - \eta\, 2(y_i - f_k^o(\text{net}_{i,k}^o))\left(-\left[f_k^o(\text{net}_{i,k}^o)(1 - f_k^o(\text{net}_{i,k}^o))\right]\right) 1,$$

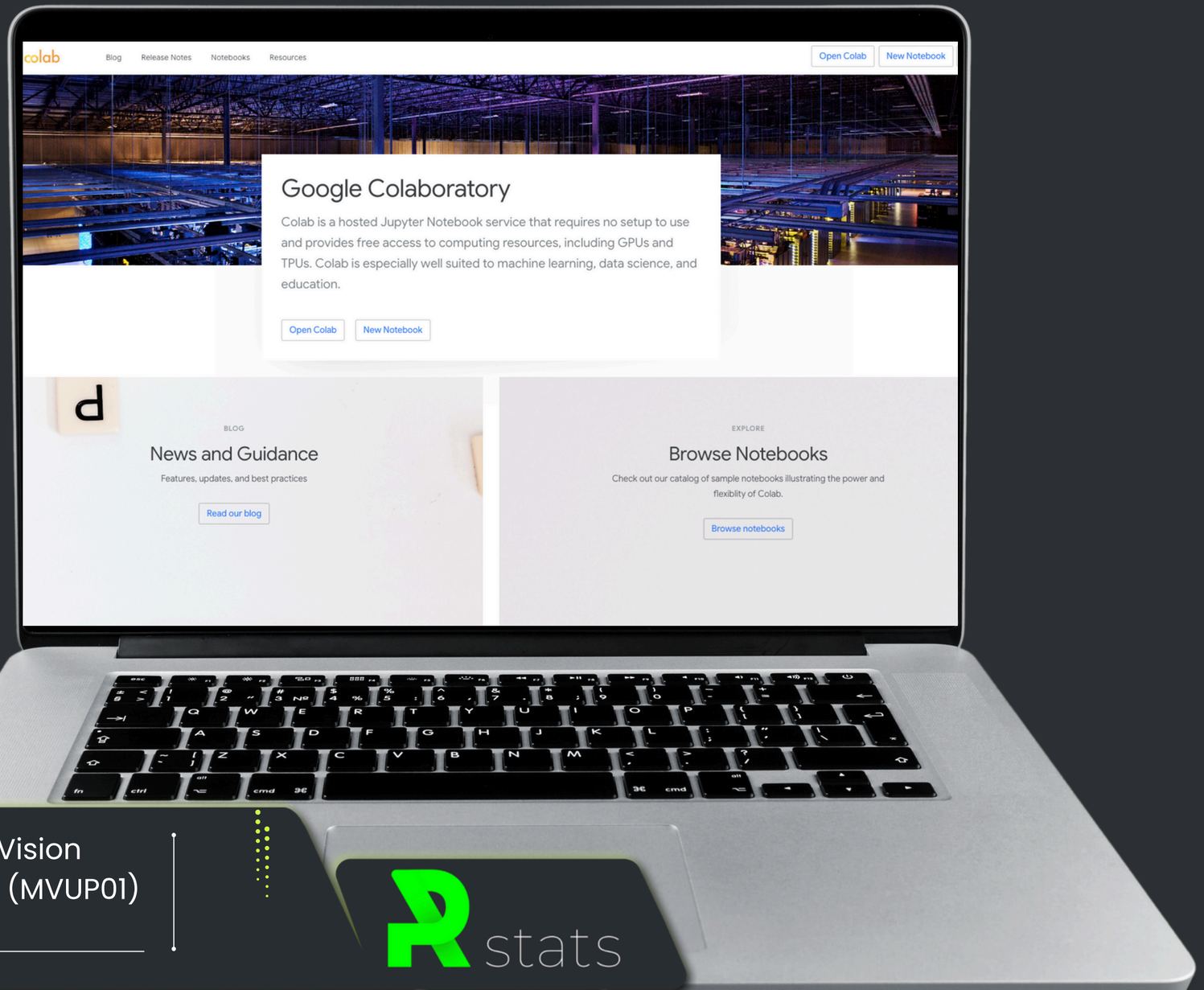
which has another term instead of $f_j^h(\text{net}_{i,j}^h)$, once the derivative:

$$\frac{\partial \text{net}_{i,k}^o}{\partial \theta_k^o} = \frac{\partial\left[\sum_j f_j^h(\text{net}_{i,j}^h) w_{k,j}^o + \theta_k^o\right]}{\partial \theta_k^o} = 1,$$

results in the number 1. Finally, we have the update rules for weights and thetas at the output layer.

Fundamentals of Digital Images

Hands-on activity:



Machine Vision
using Python (MVUP01)

stats

Lunch time