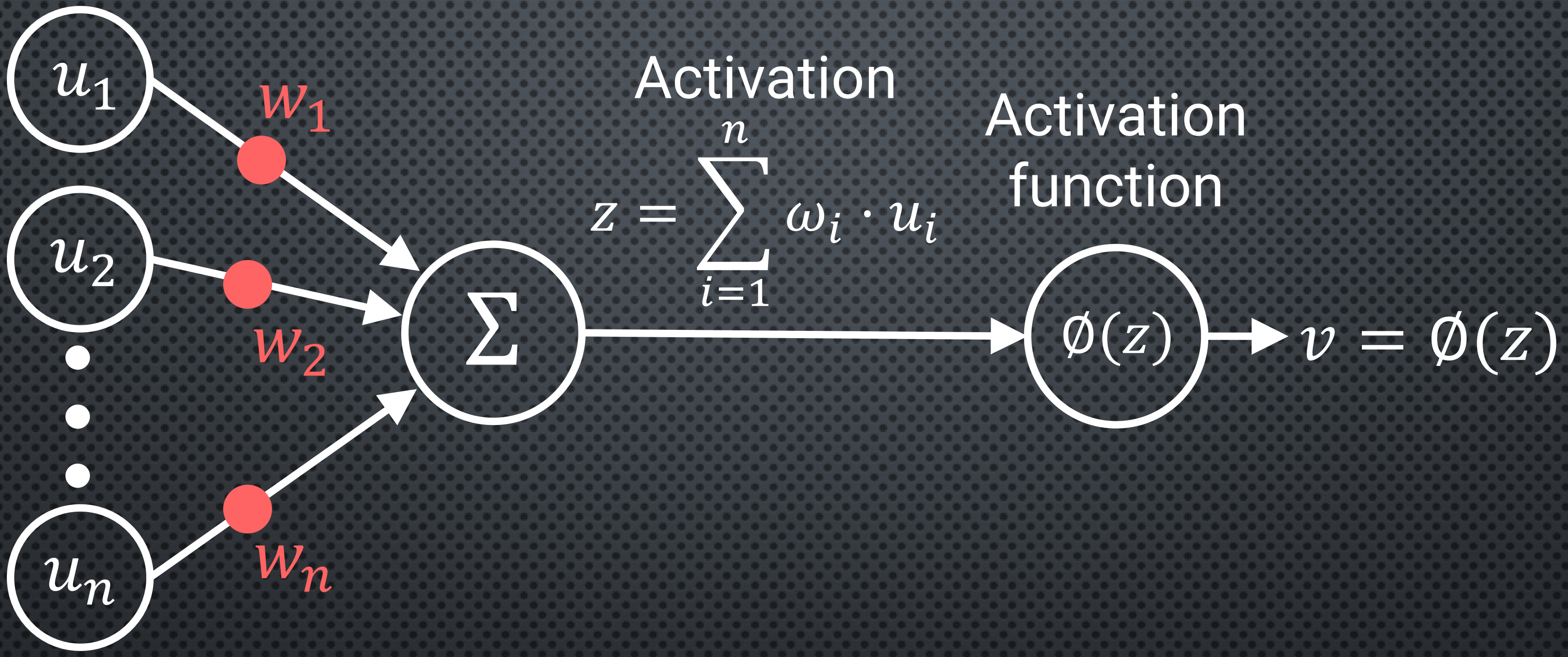
5 Supervised and unsupervised learning in artificial neural brains

Learning in perceptrons



Perceptrons have 2 sets of parameters - weights and activation function, and both affect the output.

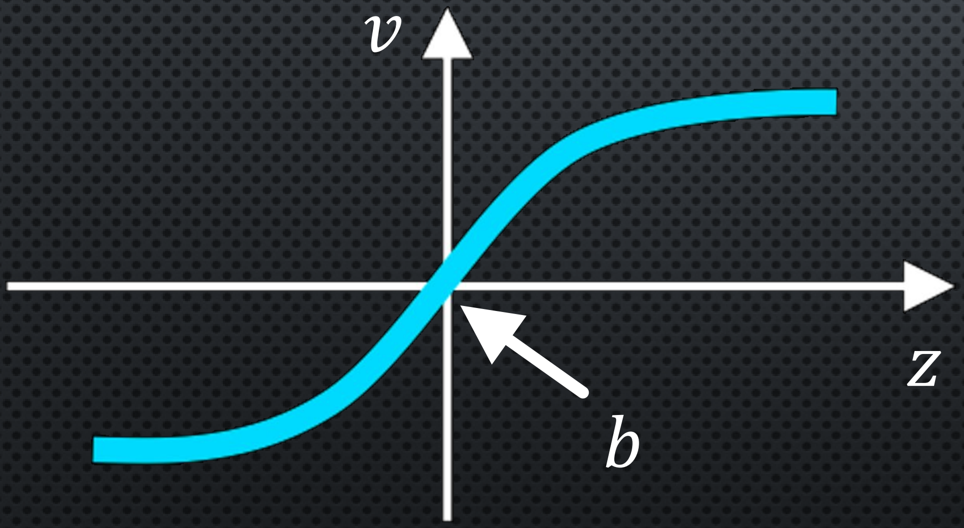
Question: So which one should we modify to learn?

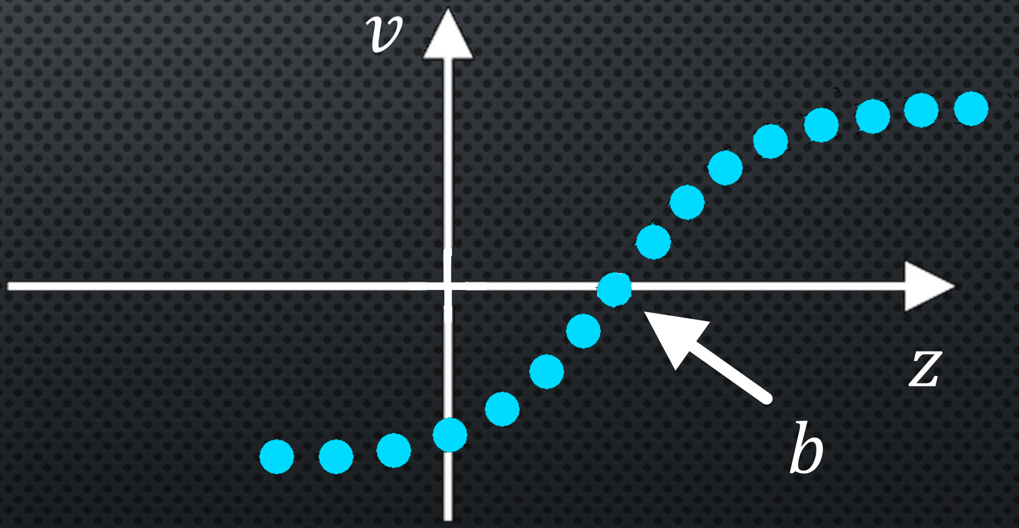
Answer: Ideally, both should be modified.

Introducing *bias*: a way to describe the activation function

- Bias is the point on the z-axis at which $v = 0$
- By changing bias, one can shift the activation function to the left or right.

Example: Sigmoid activation function $v = \dfrac{1}{1 + e^{-S(z-b)}}$ where b is the bias and S determines the slope of linear part.





Question: So how do we decide what should be the value of bias b?

Answer: We could learn it by considering b as another “weight”.

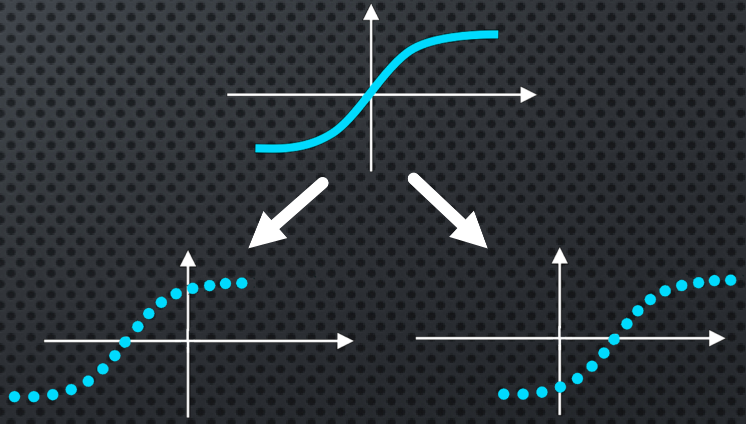
Bias b as a weight

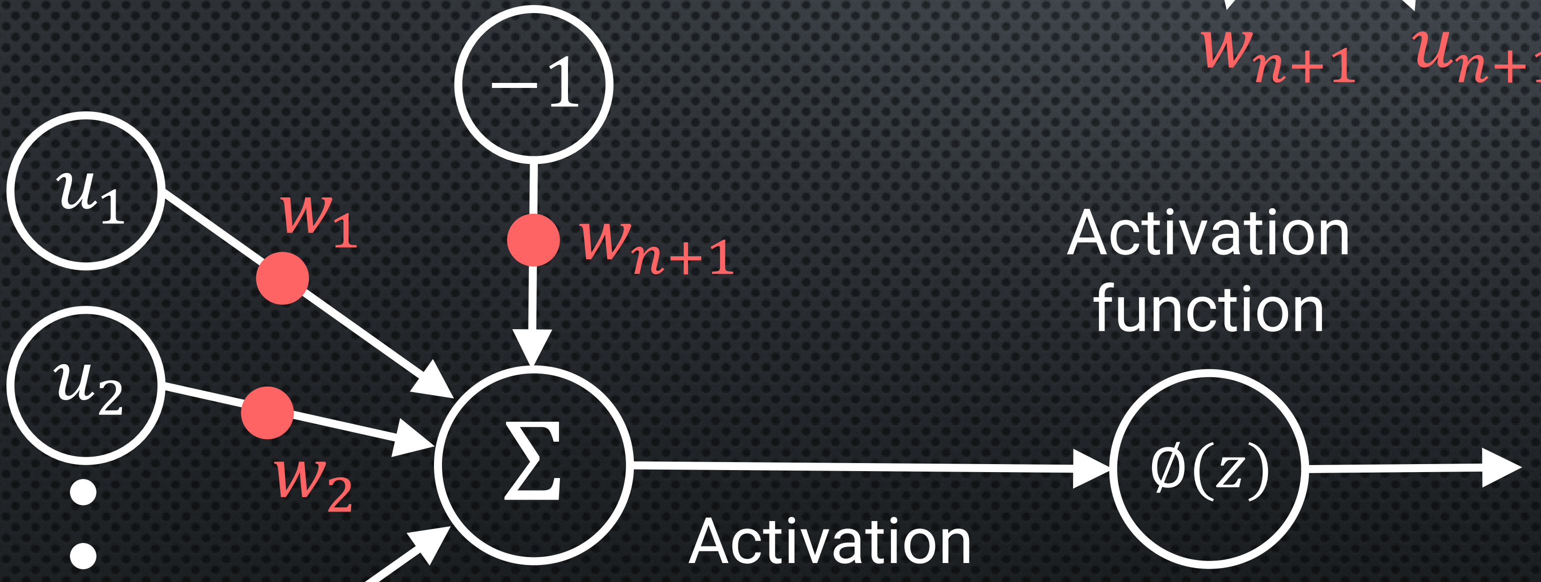
$$z = w_1 u_1 + w_2 u_2 + \cdots + w_n u_n$$

$$\rightarrow (z - b) = w_1 u_1 + w_2 u_2 + \cdots + w_n u_n - b$$

$$\rightarrow (z - b) = w_1 u_1 + w_2 u_2 \ldots + w_n u_n + \underbrace{b}_{w_{n+1}}\ \underbrace{(-1)}_{u_{n+1}}$$

This small change sets the AF in the centre **before learning**. By learning w_{n+1}, we can shift the activation function automatically.



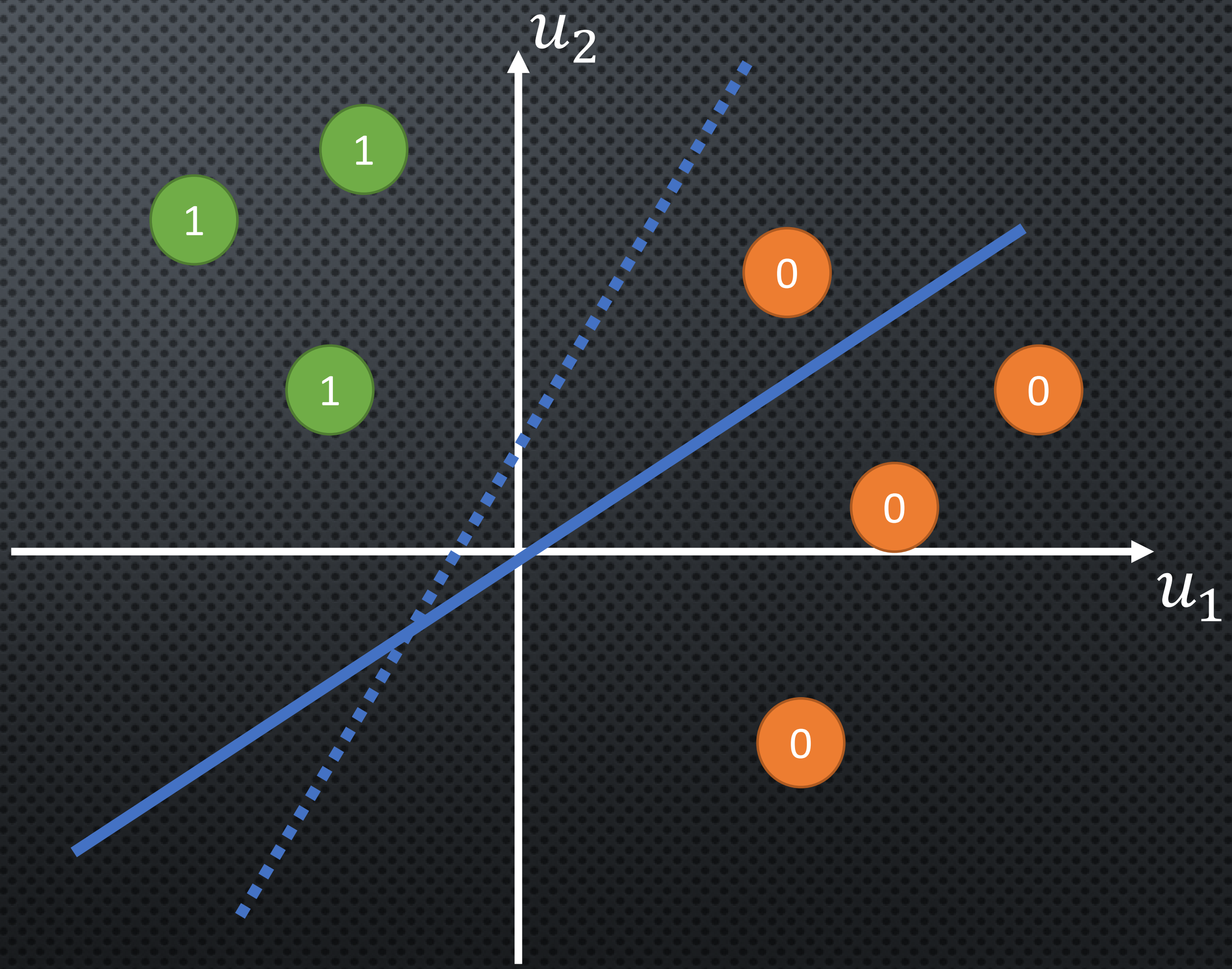


$$z = \sum_{i=1}^{n+1} \omega_i \cdot u_i$$

$$v = \frac{1}{1 + e^{-S(z-b)}}$$

Why shift the activation function?

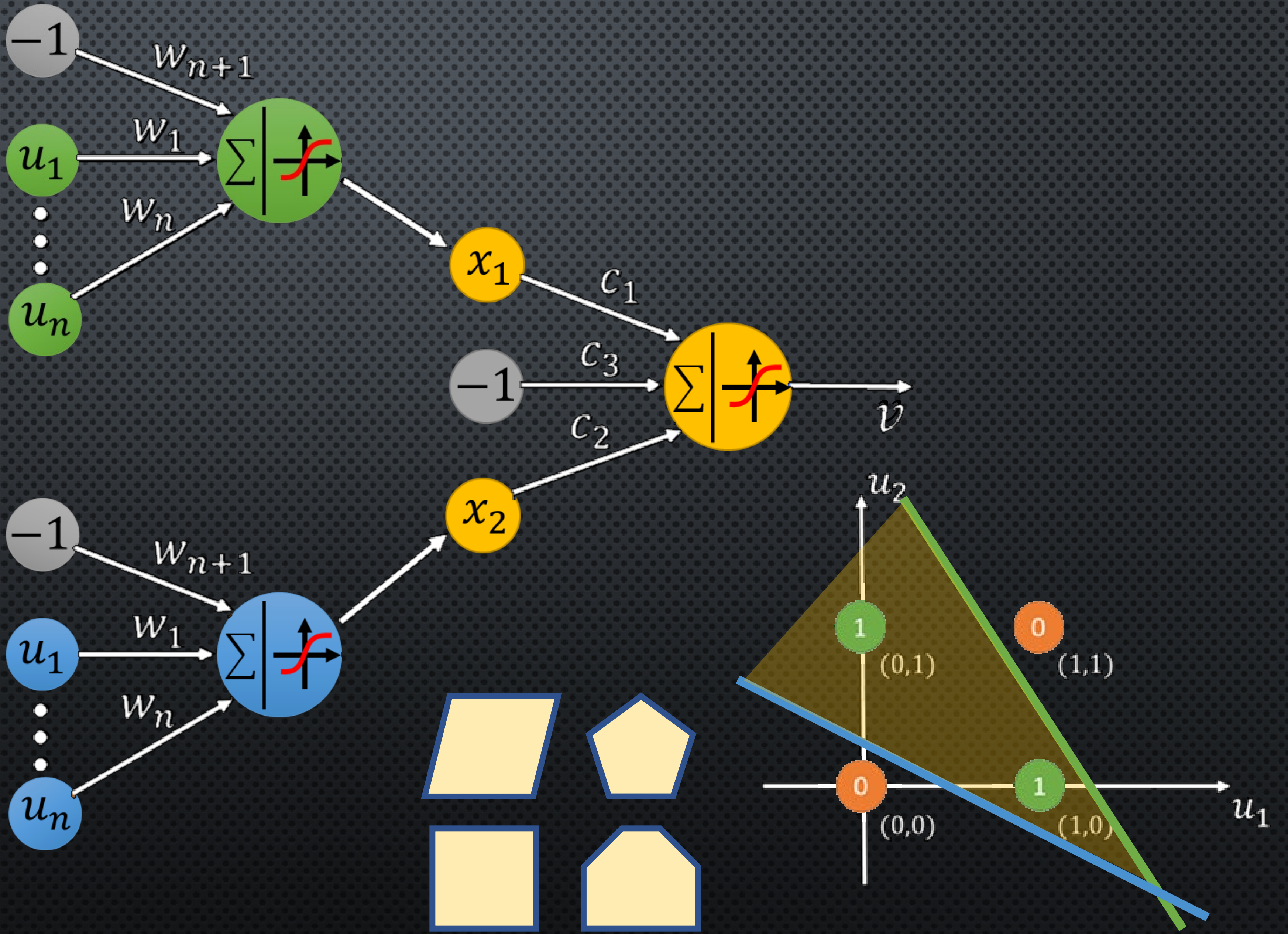
- The blue line is called the decision boundary.
- Decision boundary separates inputs into different classes (for a classification problem). Here the classes are “0” and “1”, but it can also be labels such as “cat”, “dog” etc.
- By changing $w_1, w_2, \dots, w_n, w_{n+1}$ by learning, the decision boundary can be shifted to adapt to new input data.



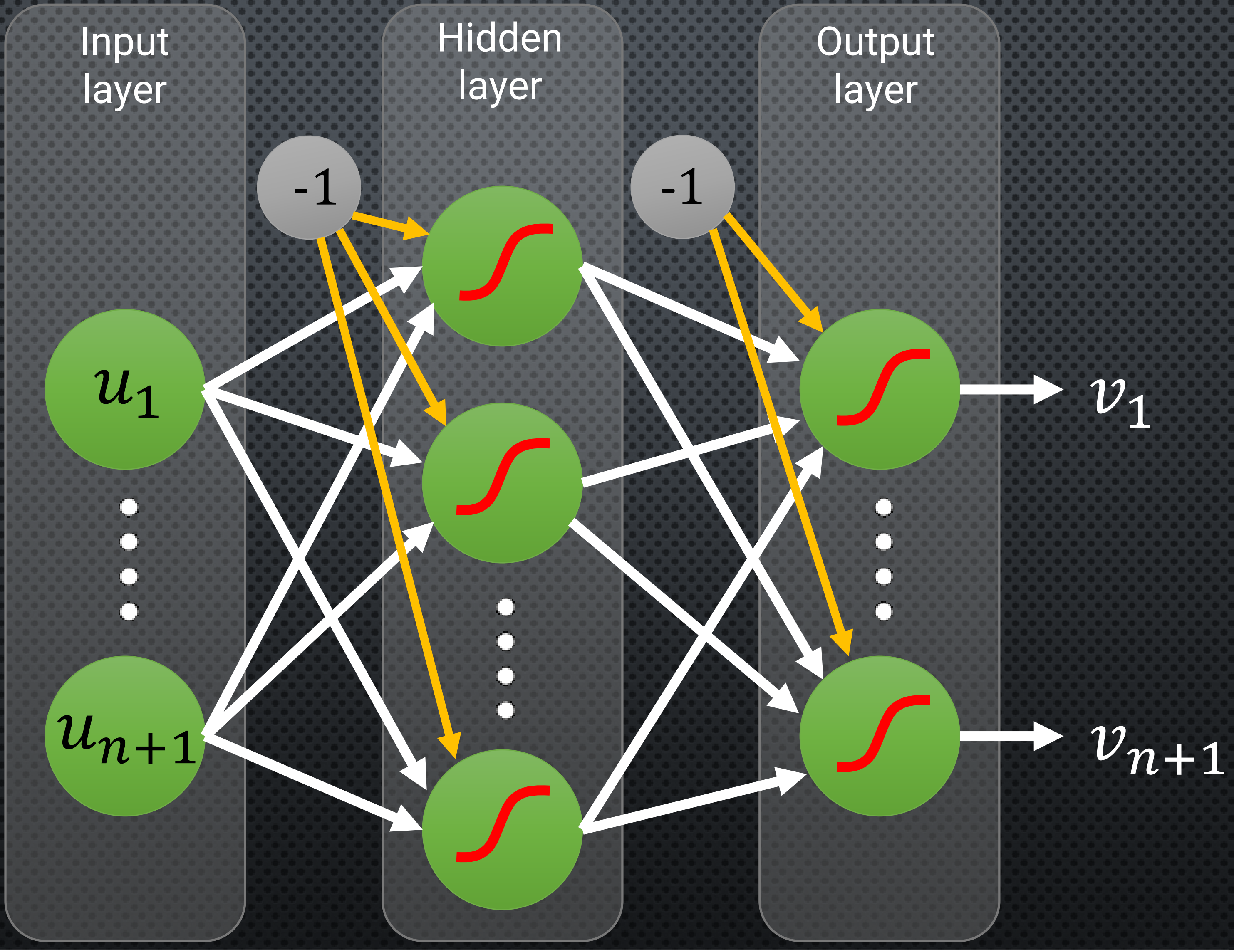
Equation for blue line:
$w_1 u_1 + w_2 u_2 \dots + w_n u_n + b(-1) = (z - b)$

From decision boundaries to decision surfaces

- By adding a third perceptron as the next layer, we get a 2-dimensional decision surface.
- By adding more perceptrons in the first layer, we can draw more decision boundaries to enclose more complex decision surfaces.

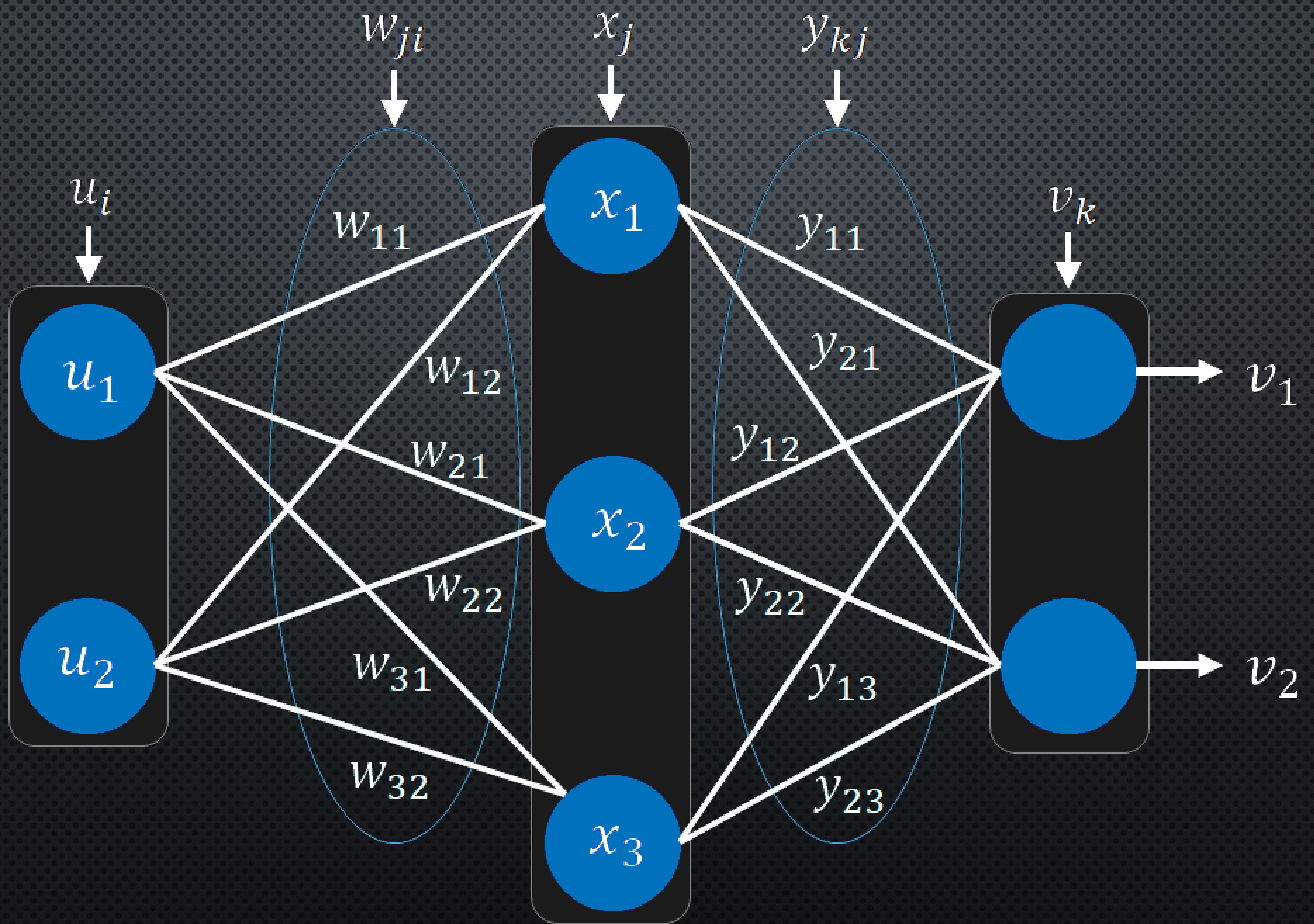


Multi-Layer Perceptron (MLP)



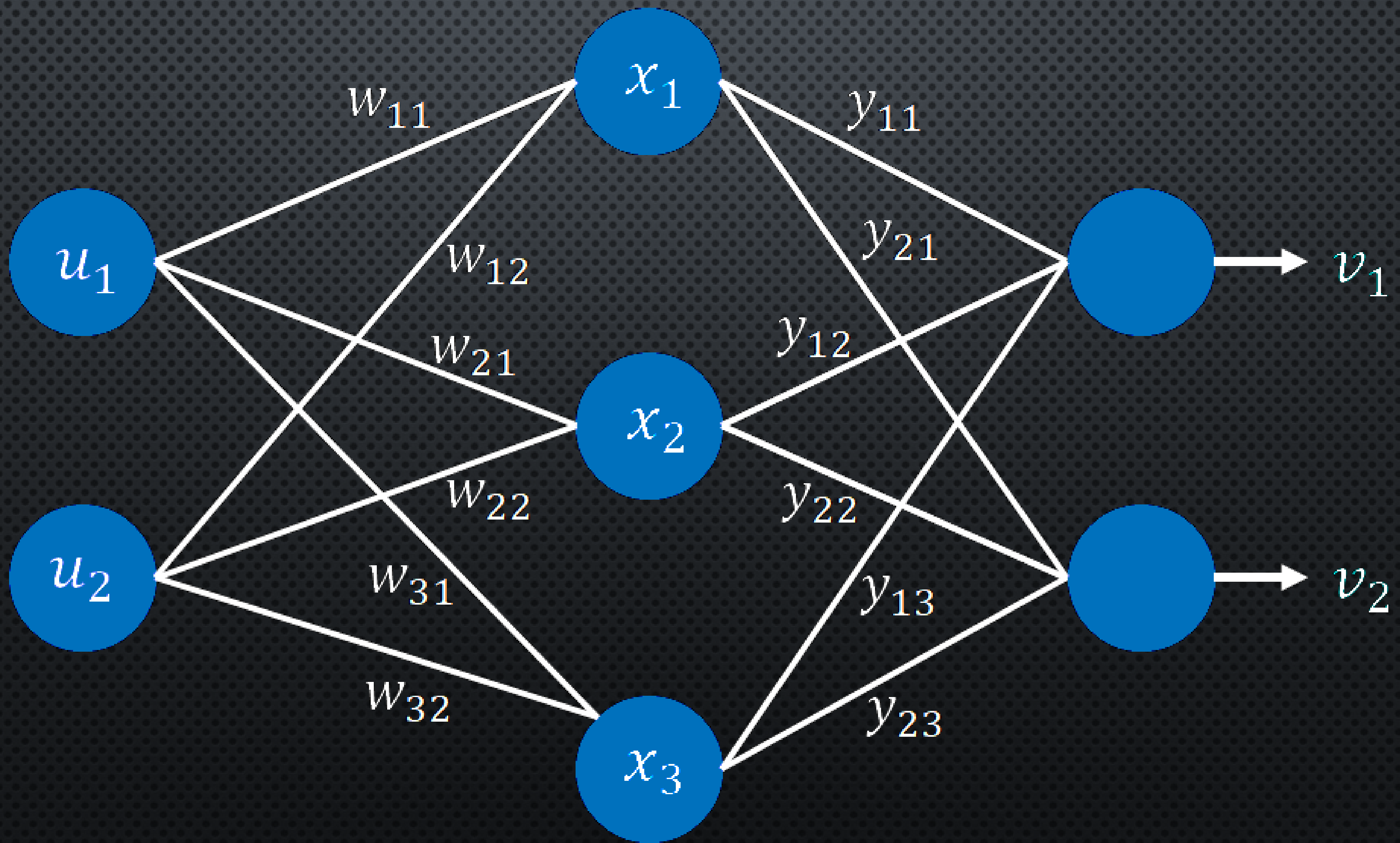
In deep learning neural networks, there is **more than one** hidden layer

Training a MLP: Notation



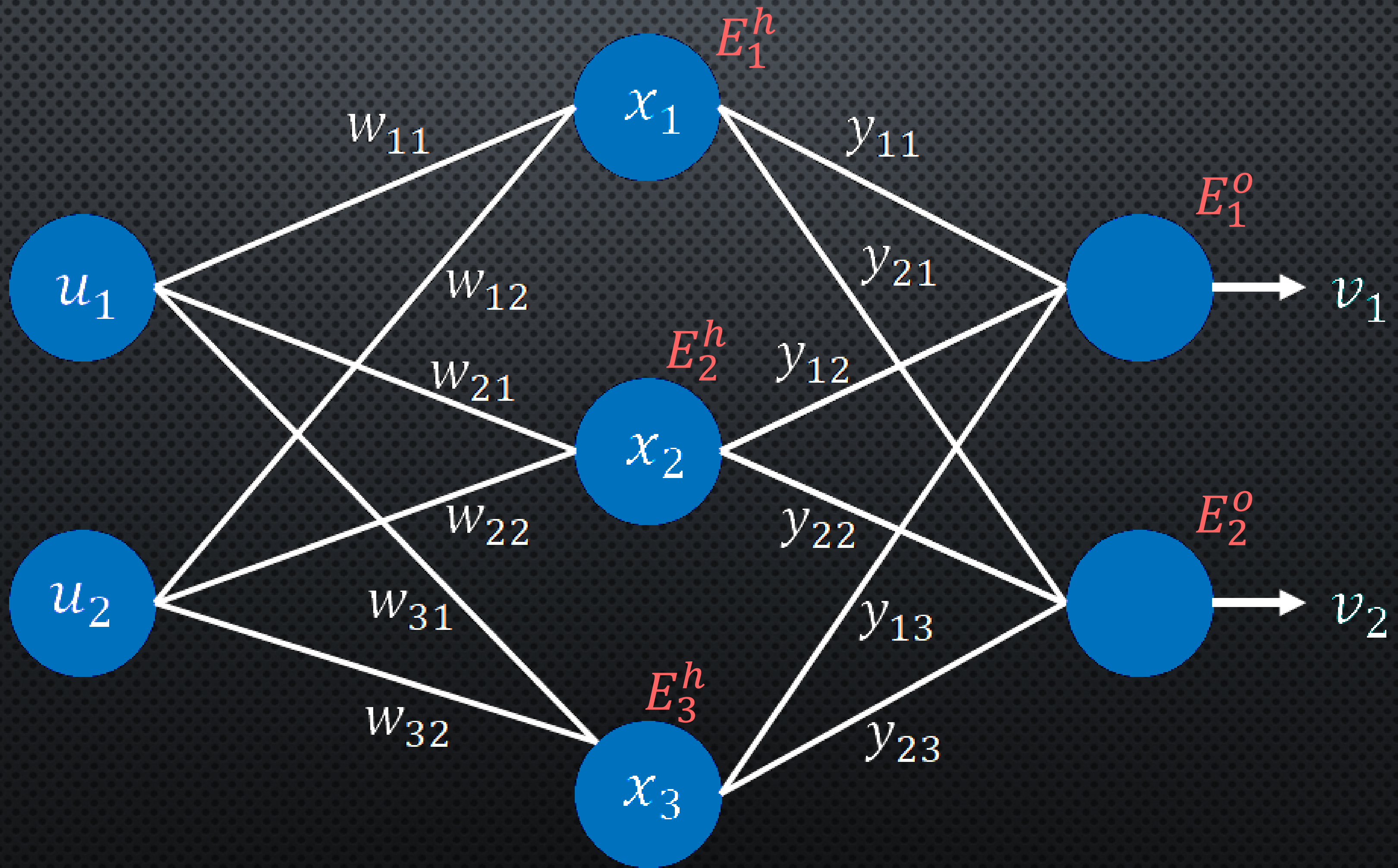
Step 1: Forward propagation

- Calculate output $\boldsymbol{x_j} = \sum_i u_i w_{ji}$ for all hidden neurons
- Calculate output $\boldsymbol{v_k} = \sum_j x_j y_{kj}$ for all output neurons



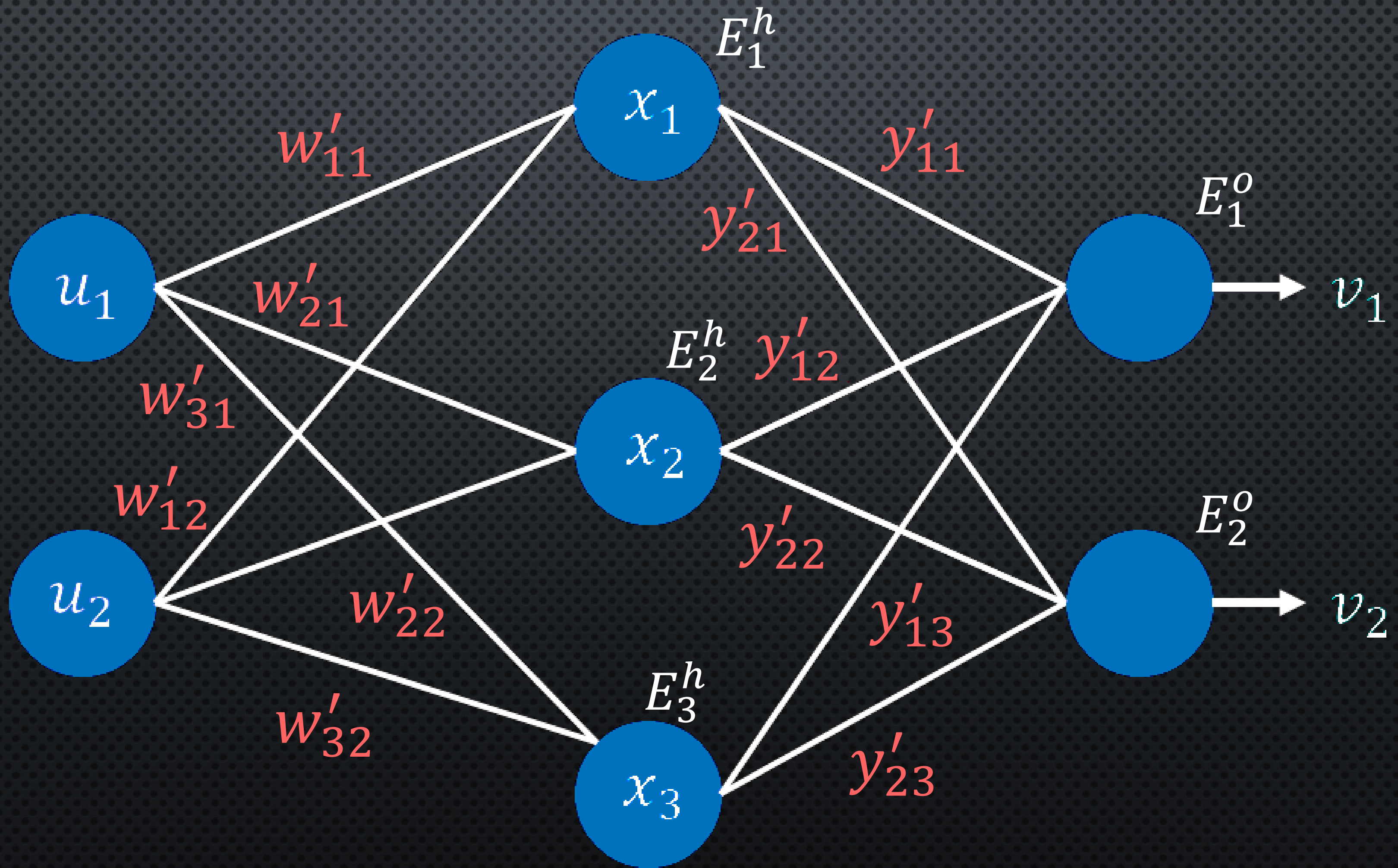
Step 2: Backpropagation

- Calculate error gradient for all output neurons, $\boldsymbol{E_k^o = v_k(1 - v_k)(t_k - v_k)}$
- Calculate error gradient for all hidden neurons, $\boldsymbol{E_j^h = x_j(1 - x_j)} \sum_k E_k^o \, y_{kj}$



Step 2: Backpropagation

- Update weights for the output neurons, $y'_{kj} = y_{kj} + \mu\, E^o_k\, x_j$
- Update weights for the hidden neurons, $w'_{ji} = w_{ji} + \mu\, E^h_j\, u_i$



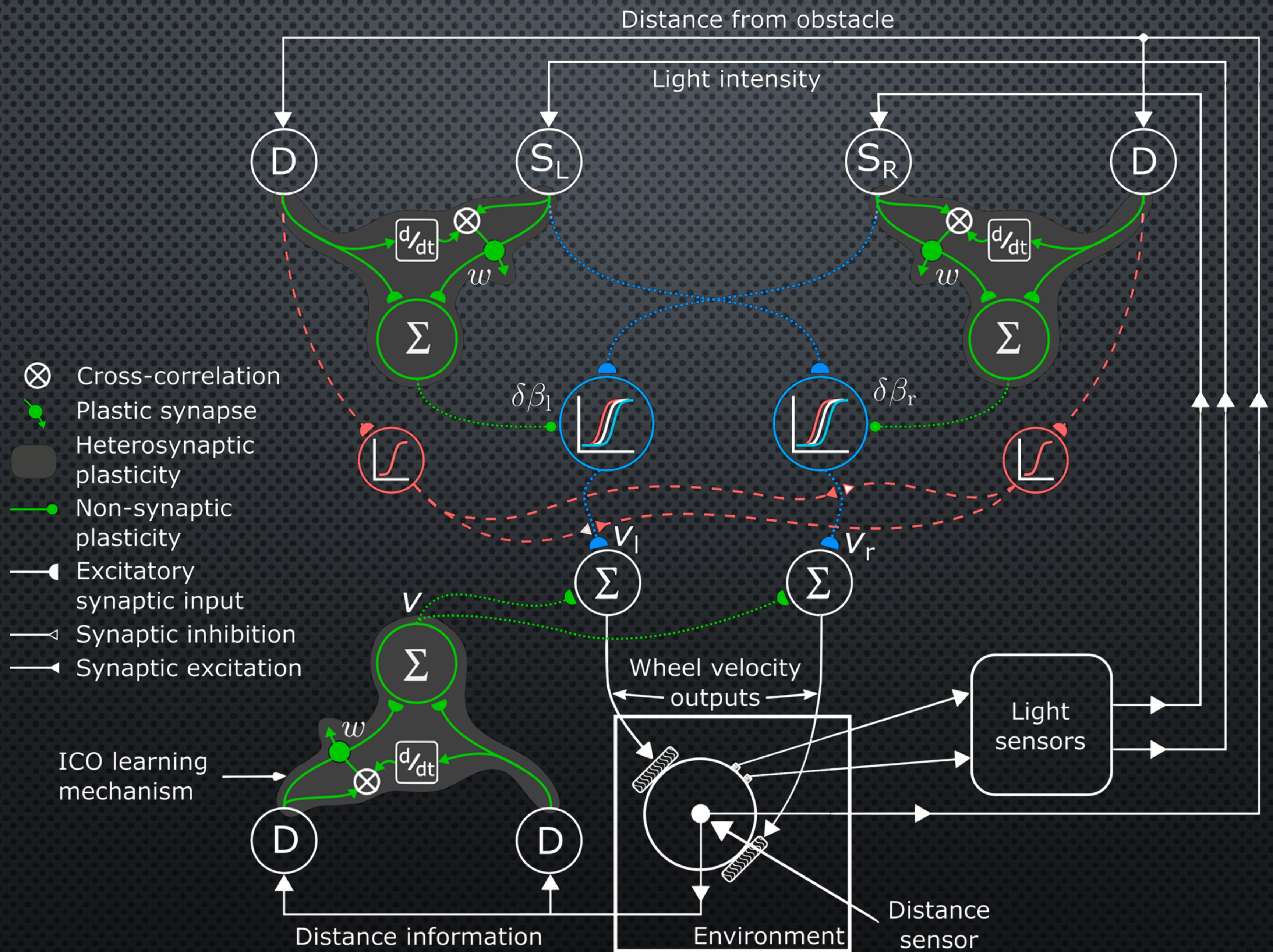
Training a MLP: backpropagation algorithm

- Divide dataset into two sets – training dataset (70% of total input data points) and testing dataset (30% of total input data points)
- Initialise all weights to random values between 0 and 1 (or -1 and +1)
- Step 1: Feed forward
  1. Randomly shuffle training dataset and select a (new) randomly chosen input data point
  2. Calculate output $\boldsymbol{x_j} = \sum_j \sum_i u_i \, w_{ji}$ for all hidden neurons, and $\boldsymbol{v_k} = \sum_k \sum_j x_j \, y_{kj}$ for all output neurons
- Step 2: Backpropagation
  1. Calculate error gradient for all output neurons, $\boldsymbol{E_k^o} = \boldsymbol{v_k}(1 - \boldsymbol{v_k})(\boldsymbol{t_k} - \boldsymbol{v_k})$
  2. Calculate error gradient for all hidden neurons, $\boldsymbol{E_j^h} = \boldsymbol{x_j}(1 - \boldsymbol{x_j}) \sum_k E_k^o \, y_{kj}$
  3. Update weights for the output neurons, $\boldsymbol{y'_{kj}} = \boldsymbol{y_{kj}} + \mu \, \boldsymbol{E_k^o} \, \boldsymbol{x_j}$
  4. Update weights for the hidden neurons, $\boldsymbol{w'_{ji}} = \boldsymbol{w_{ji}} + \mu \, \boldsymbol{E_j^h} \, \boldsymbol{u_i}$
- Repeat Step 1 and Step 2 until the error is very low or max. number of epochs is reached
- Test your trained network on testing dataset by repeating Step 1

Additional notes on training

- Training set: Used to adjust weights of the neural network
- Validation set: Used to minimize overfitting
- Testing set: Used only for testing the final solution
- Monte Carlo cross validation
  - Sub-sample data randomly into training and test sets (e.g., 70% and 30%)
- K-fold cross validation
  - Divide data into *k* subsets
  - Each time (in total *k* times) one of the subsets is used for testing and the rest *k-1* subsets are joined and used as a training set
- Leave-p-out cross validation
  - Use *p* data samples as training samples and the rest (*n-p*) data samples as test samples
  - Train and test $\frac{n!}{p! \cdot (n-p)!}$ times

Learning the activation function



Hungry for more?