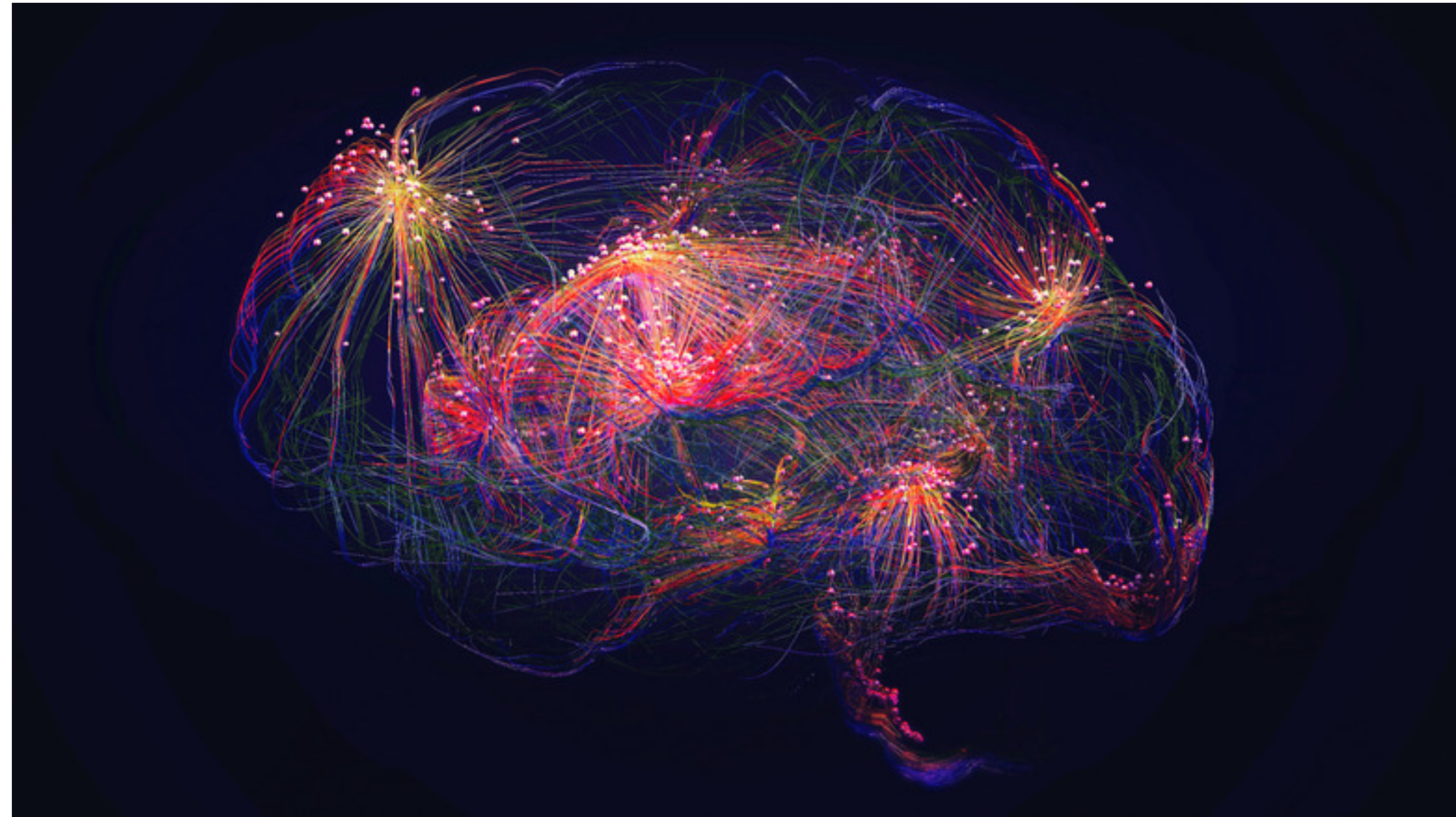


# Artificial Neural Networks



## Artificial Neural Networks inspired by brain research

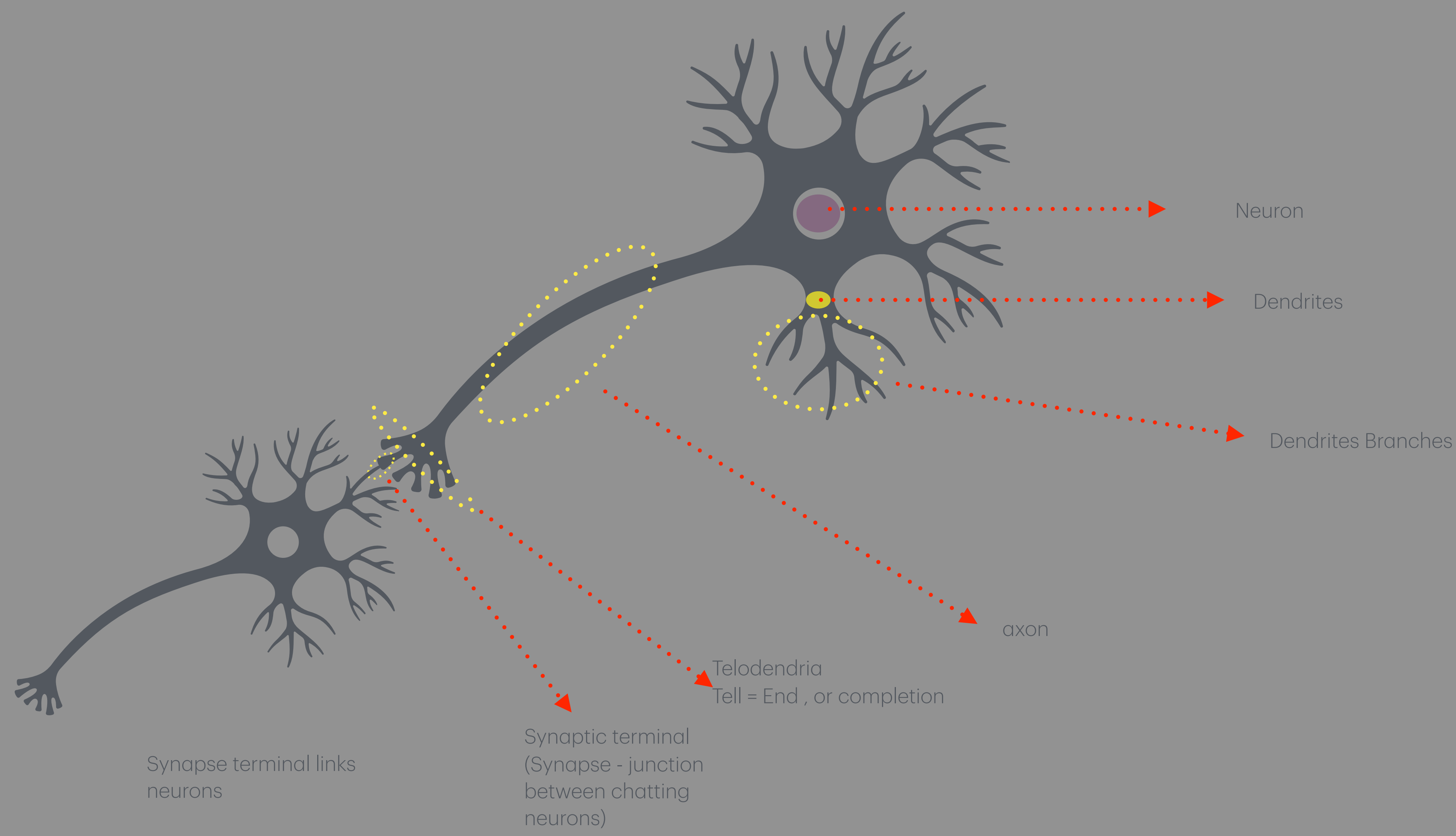
Speech Recognition

Google Images (Classification)

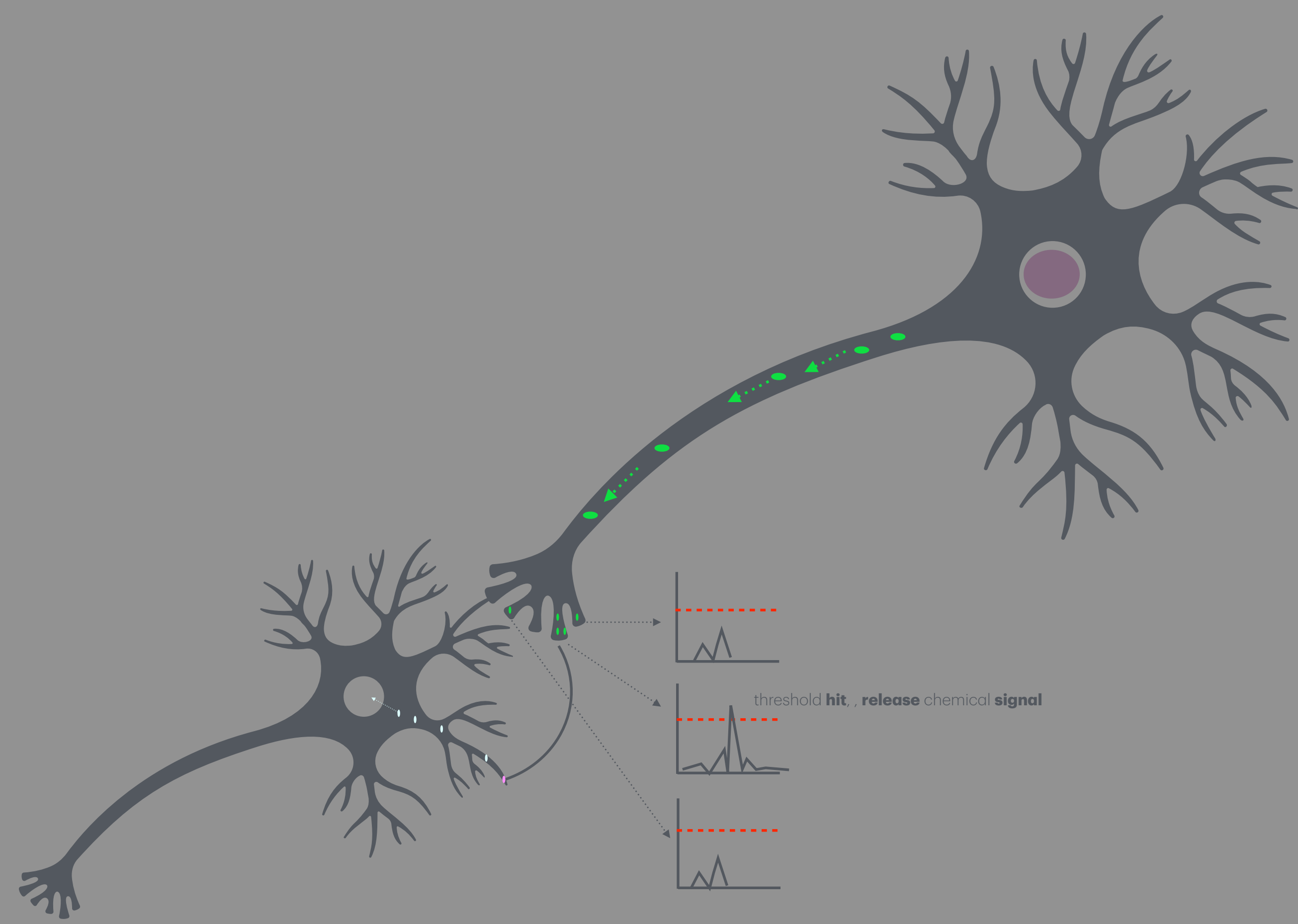
Go (DeepMind)

Recommendation Systems  
(Youtube)

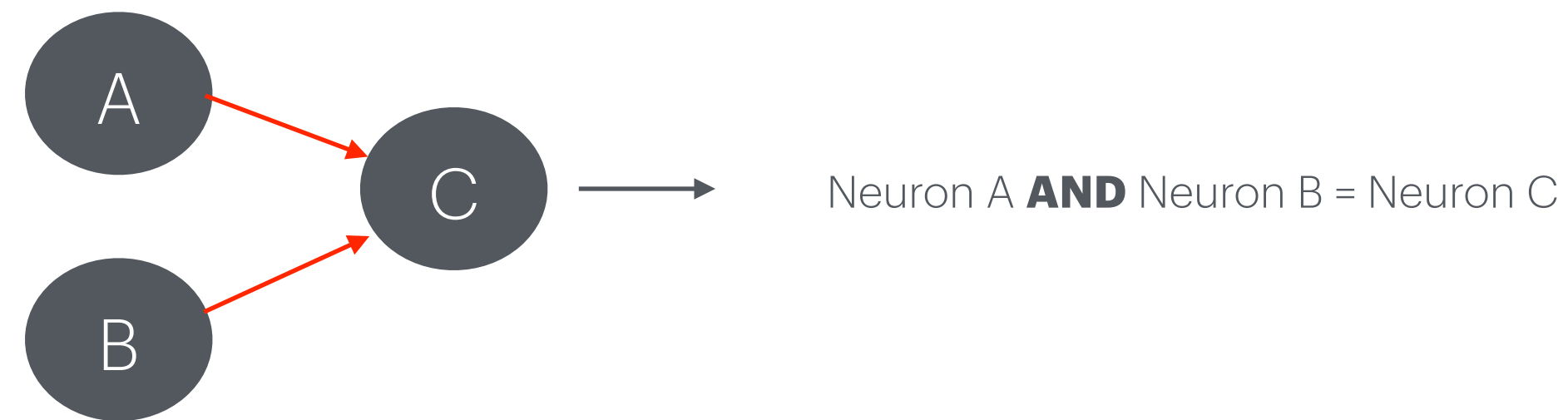
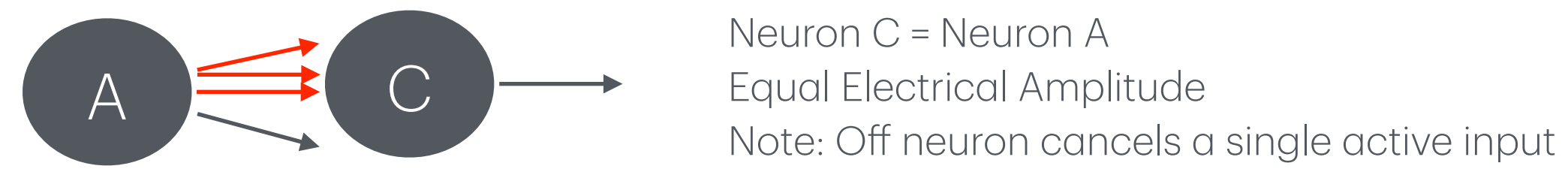
# Biological Neuron



# Biological Neuron



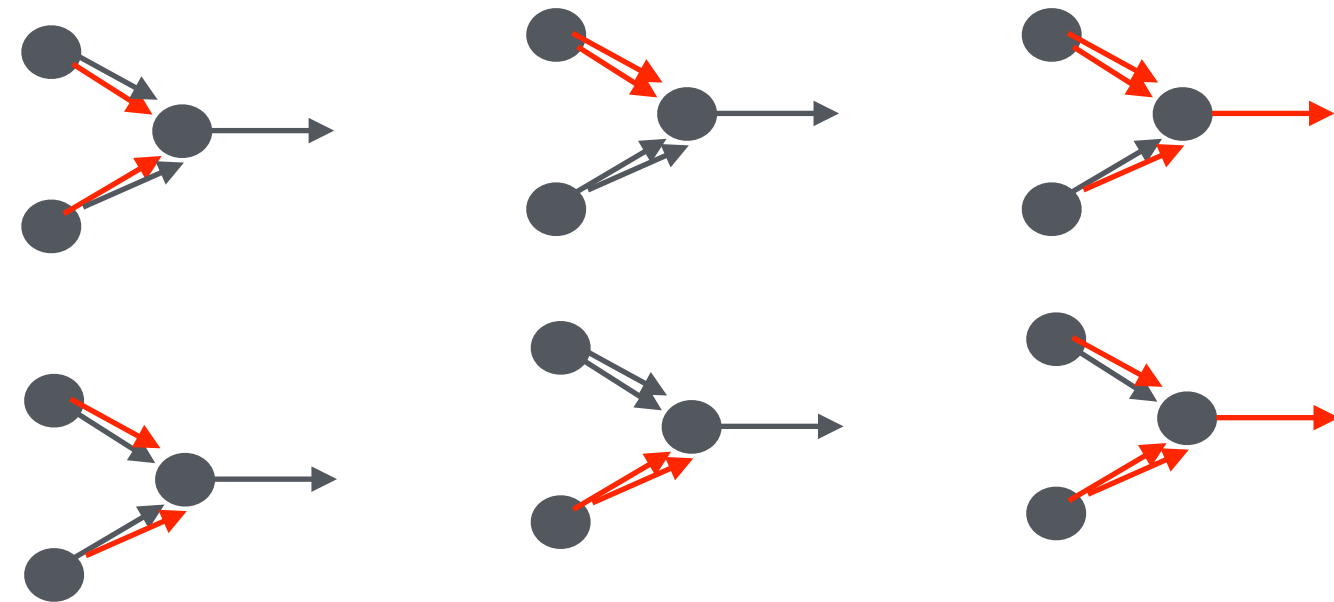
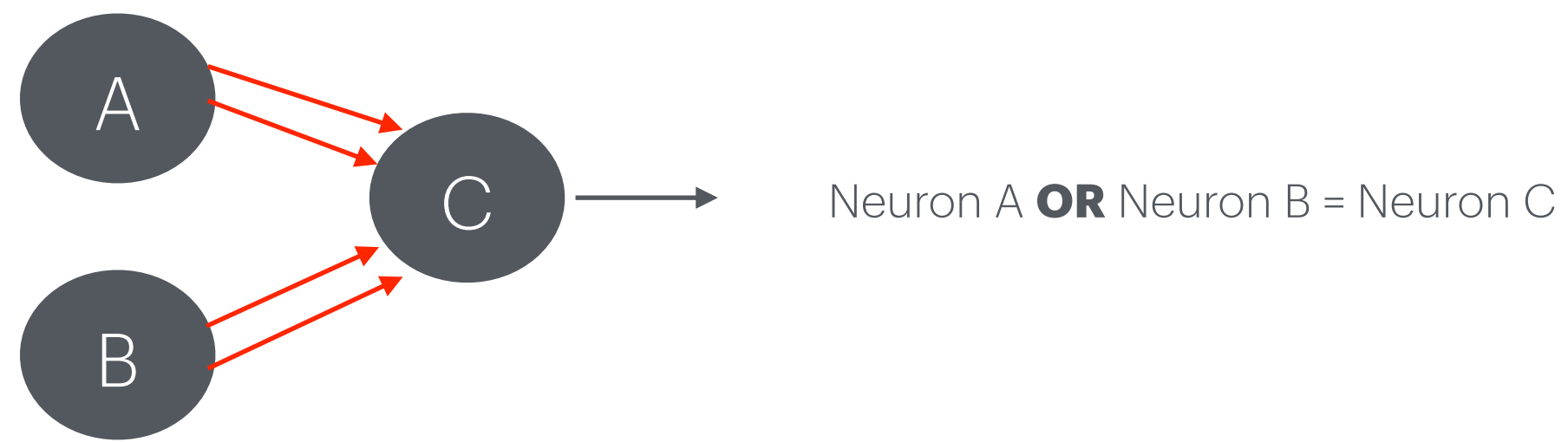
# Logical Computation: Artificial Neuron



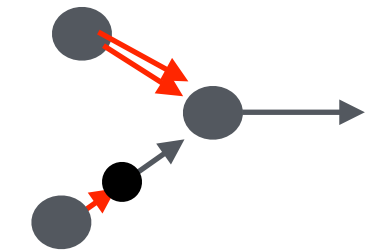
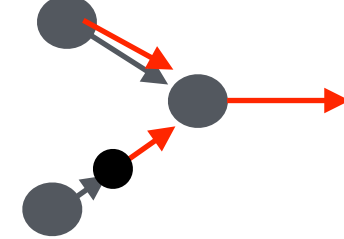
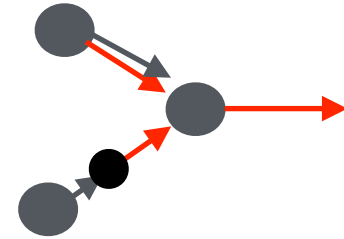
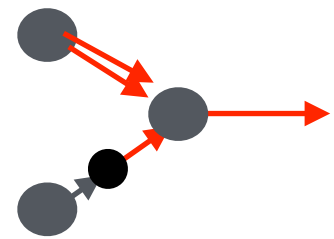
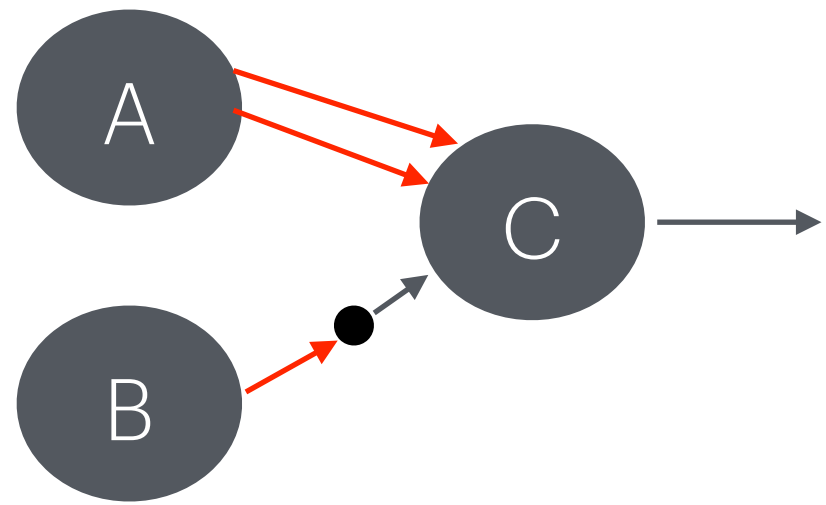
Note: Neuron activated  
when at least two inputs  
are active

Note: consumer neuron  
C sums all the inputs

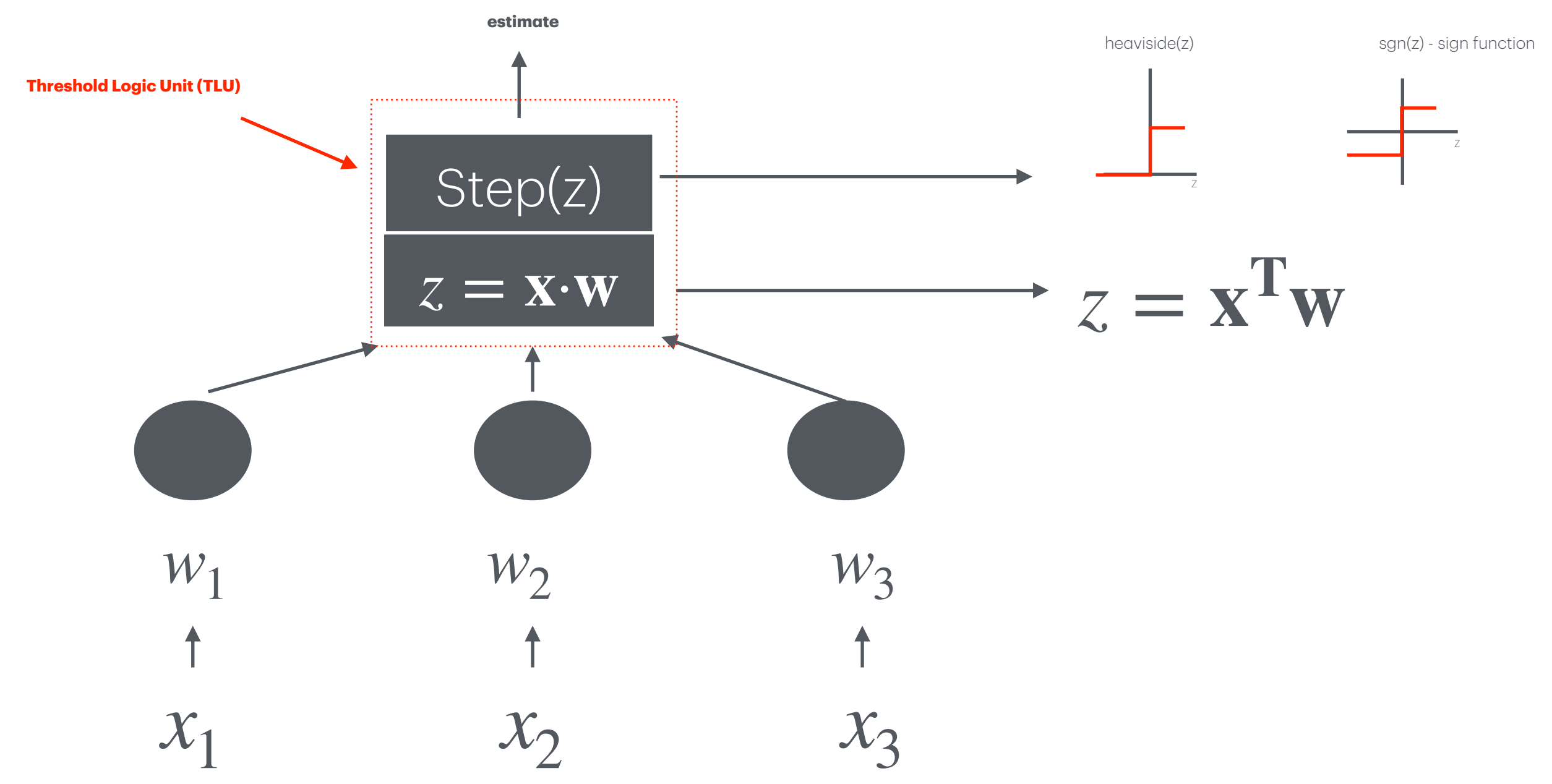
# Logical Computation: Artificial Neuron



# Logical Computation: Artificial Neuron

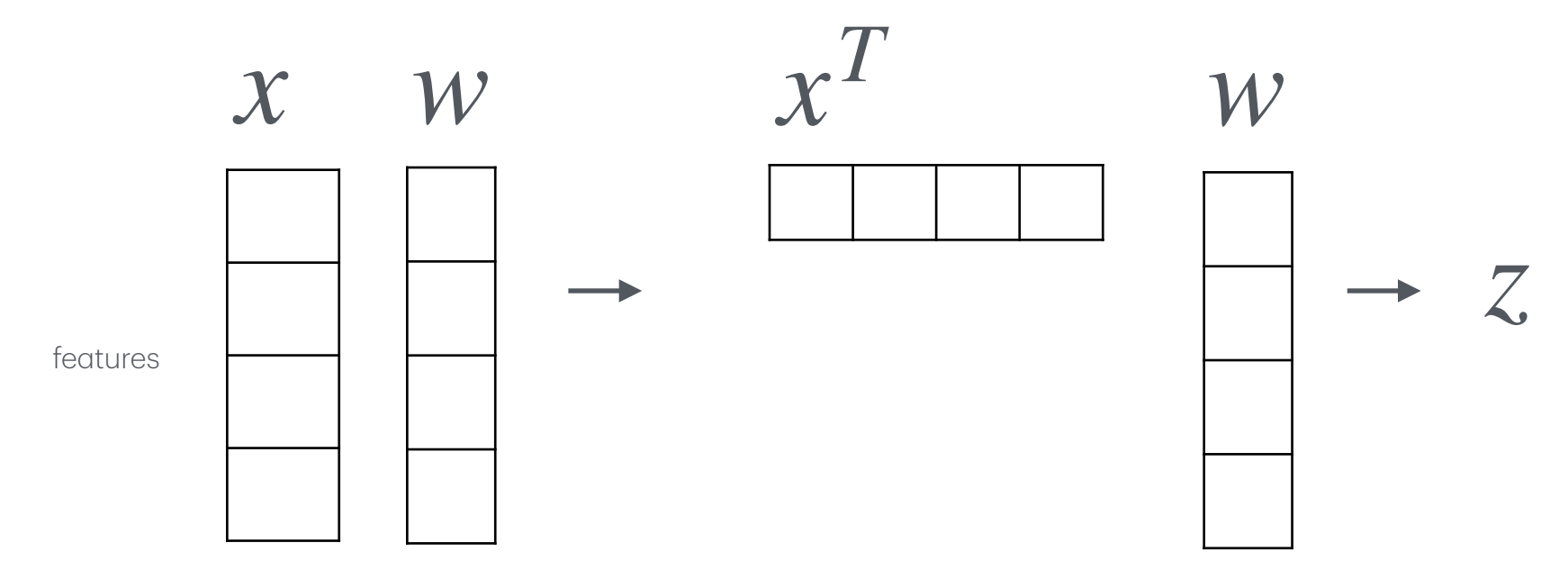
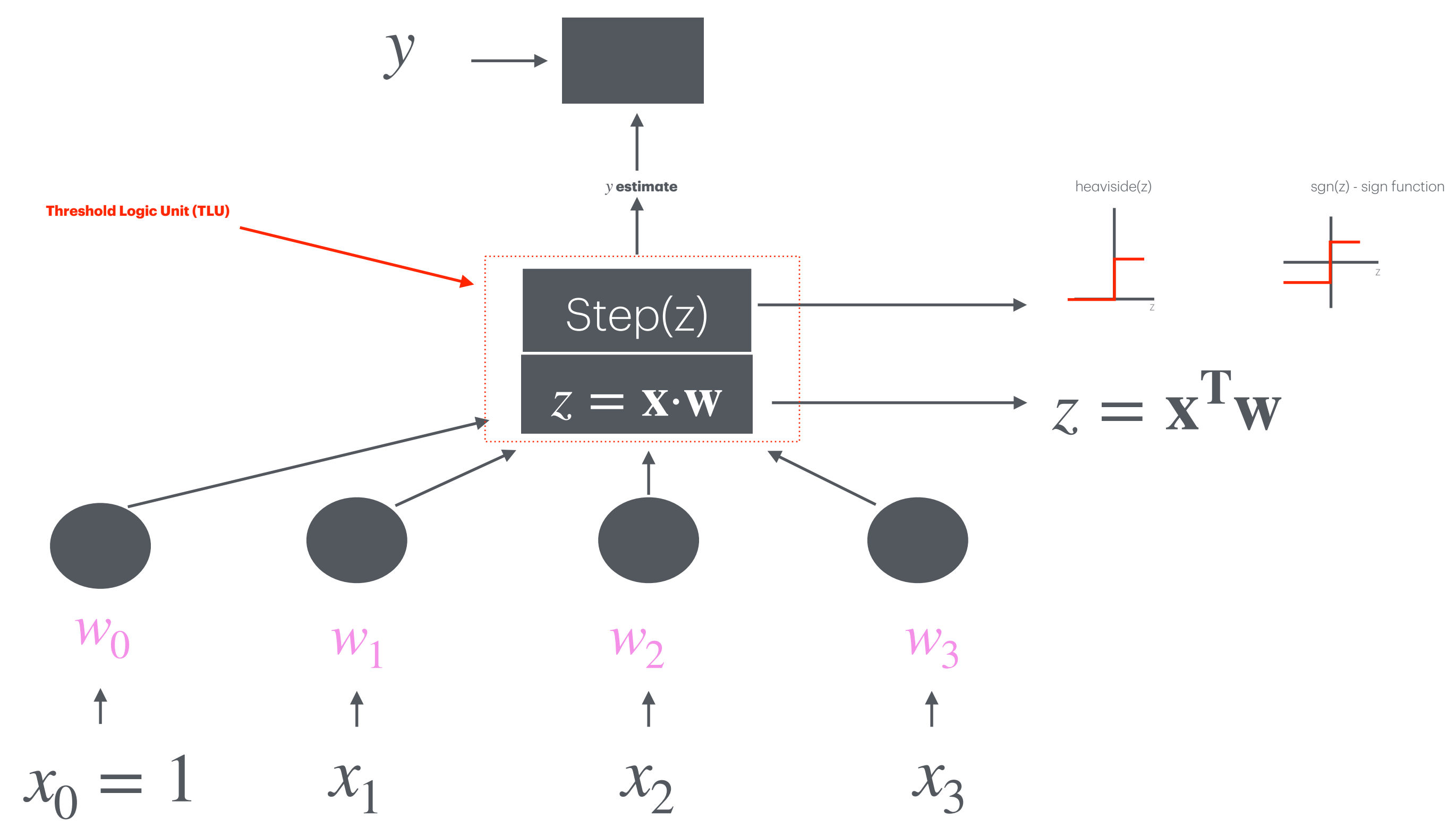


# Logical Computation: Perception





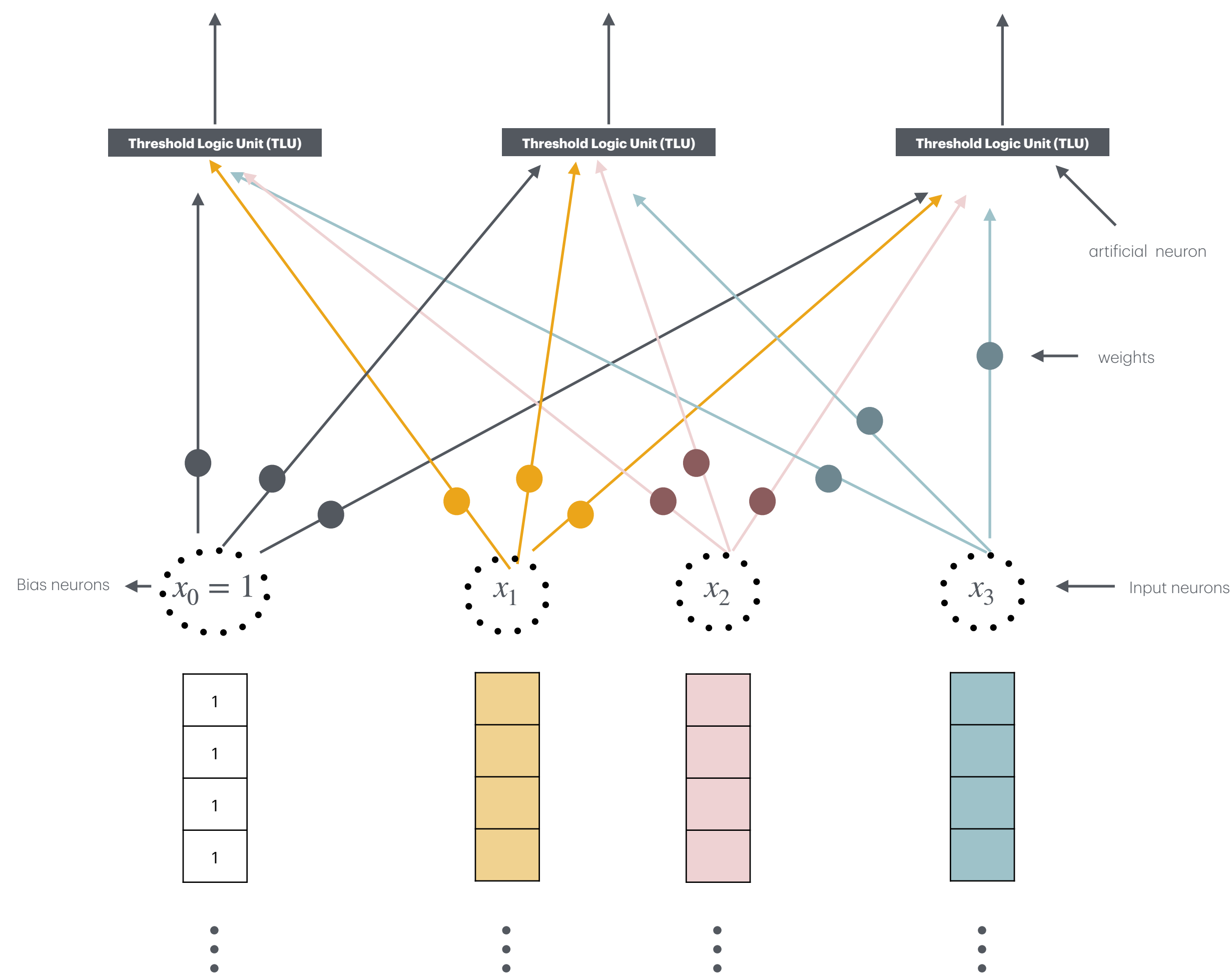
# Logical Computation: Perception



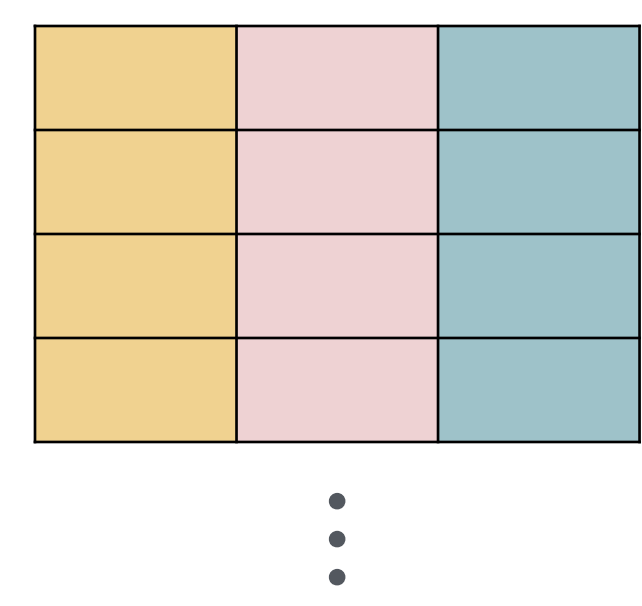
Find right values for weights that lower MSE with expectations

Used for simple linear classification (1D, 2D features )

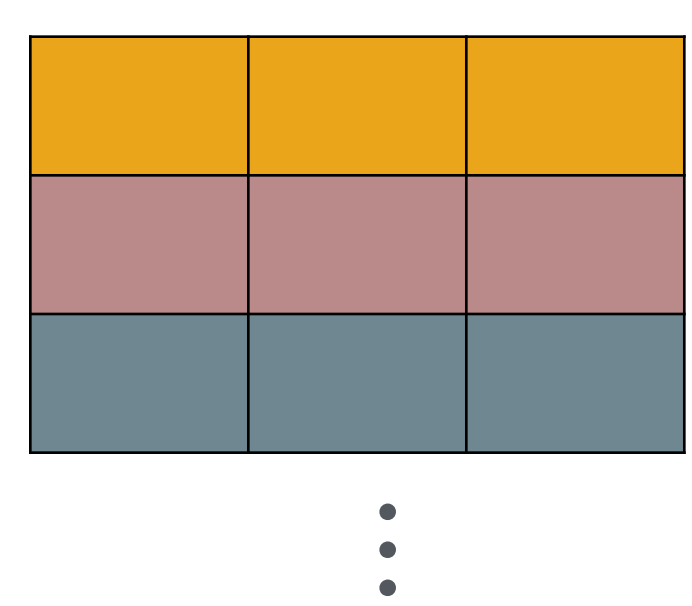
# Logical Computation: Multi-Perception



X



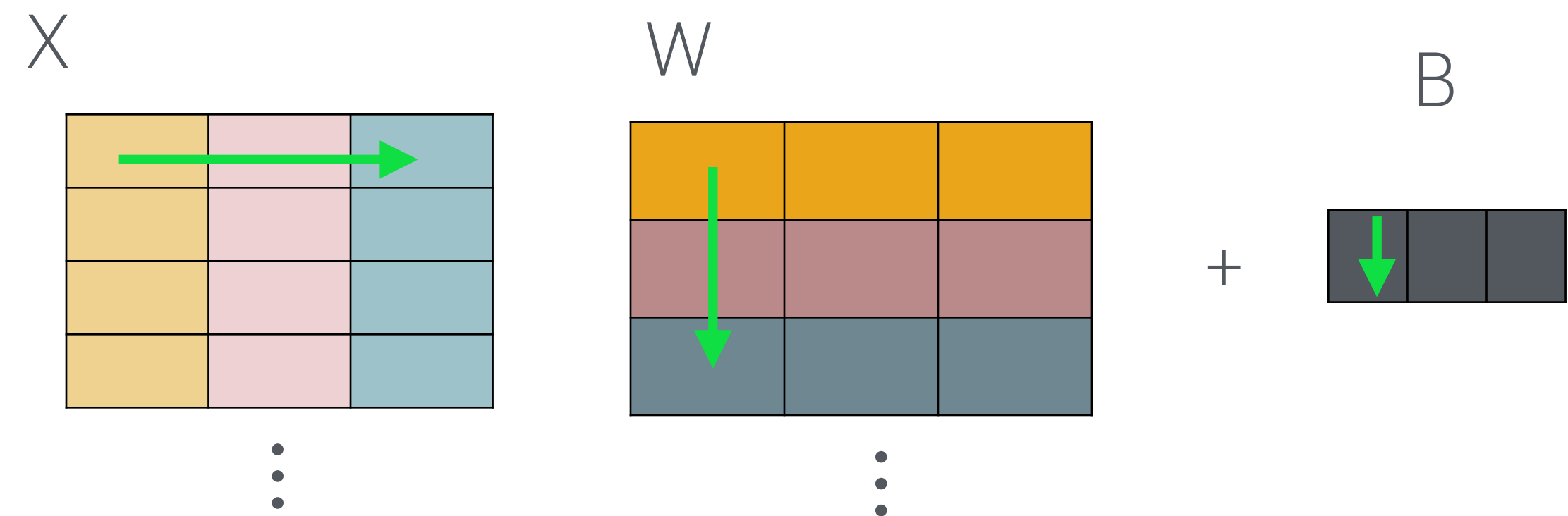
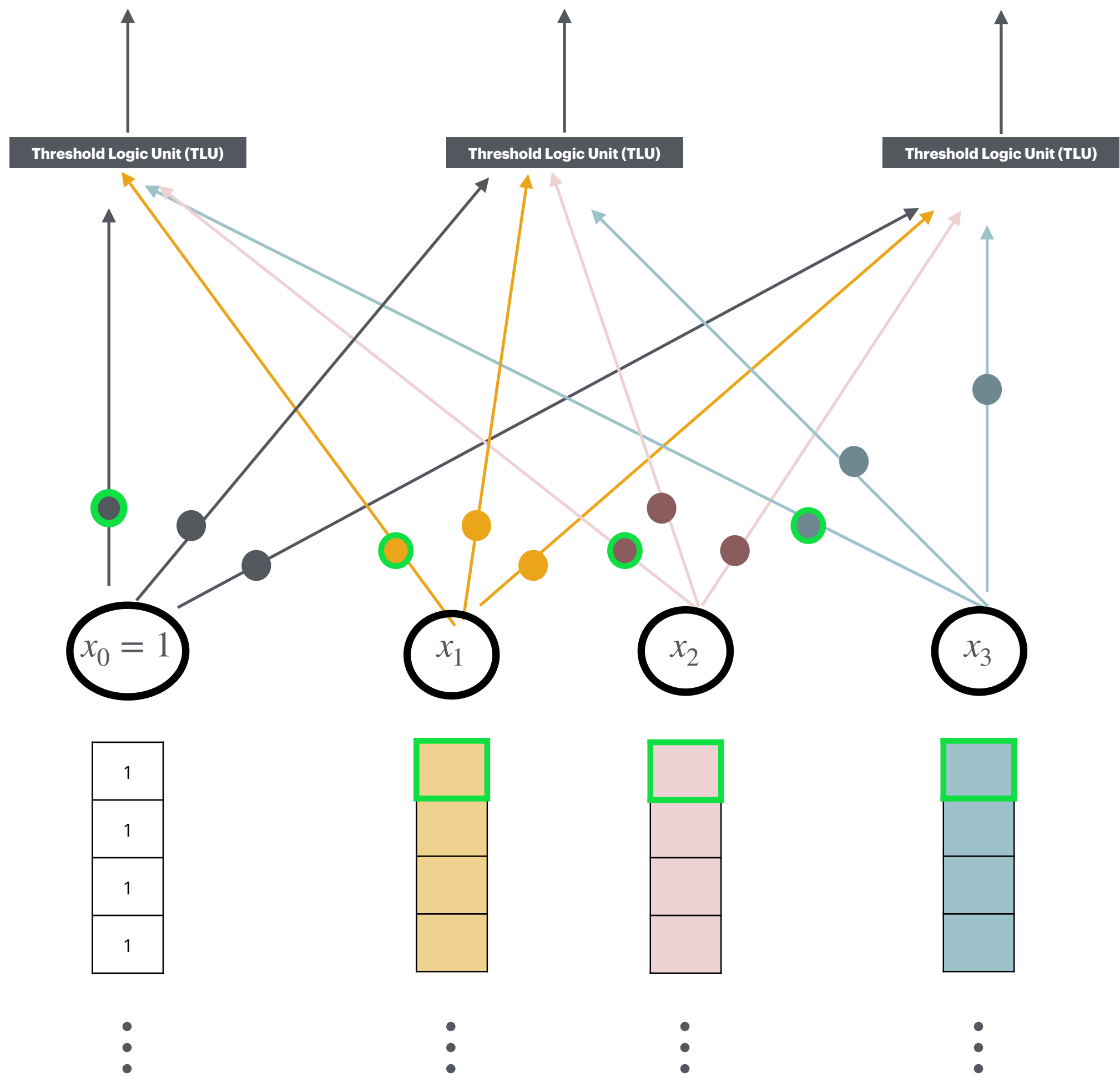
W



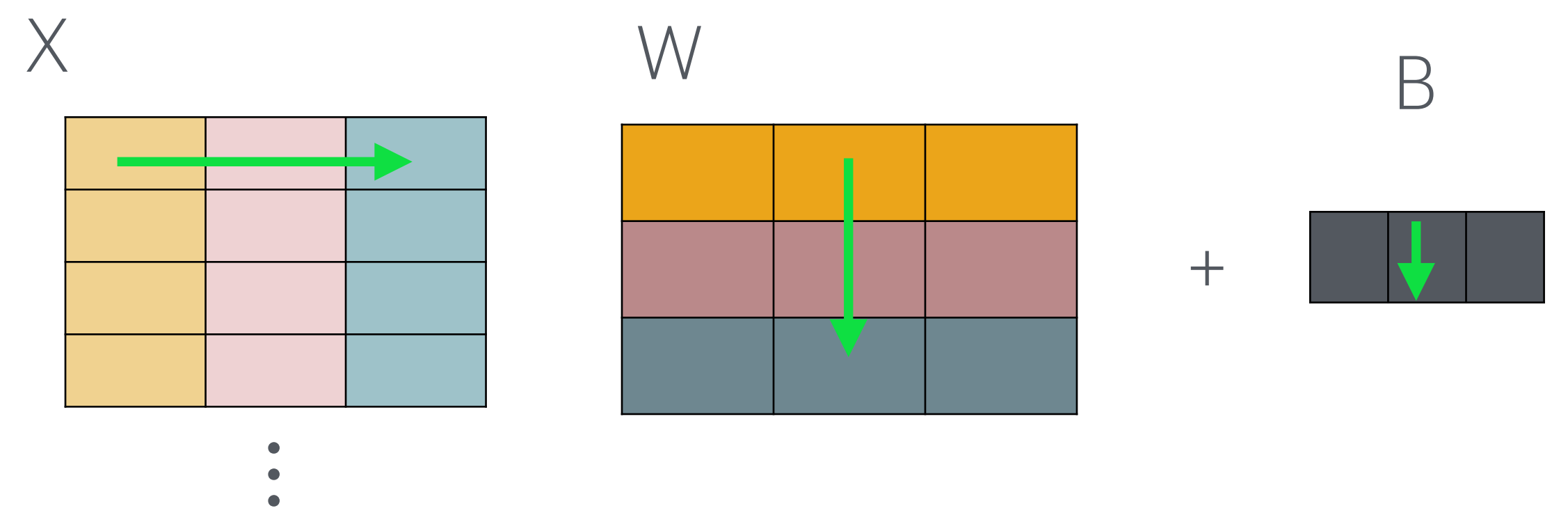
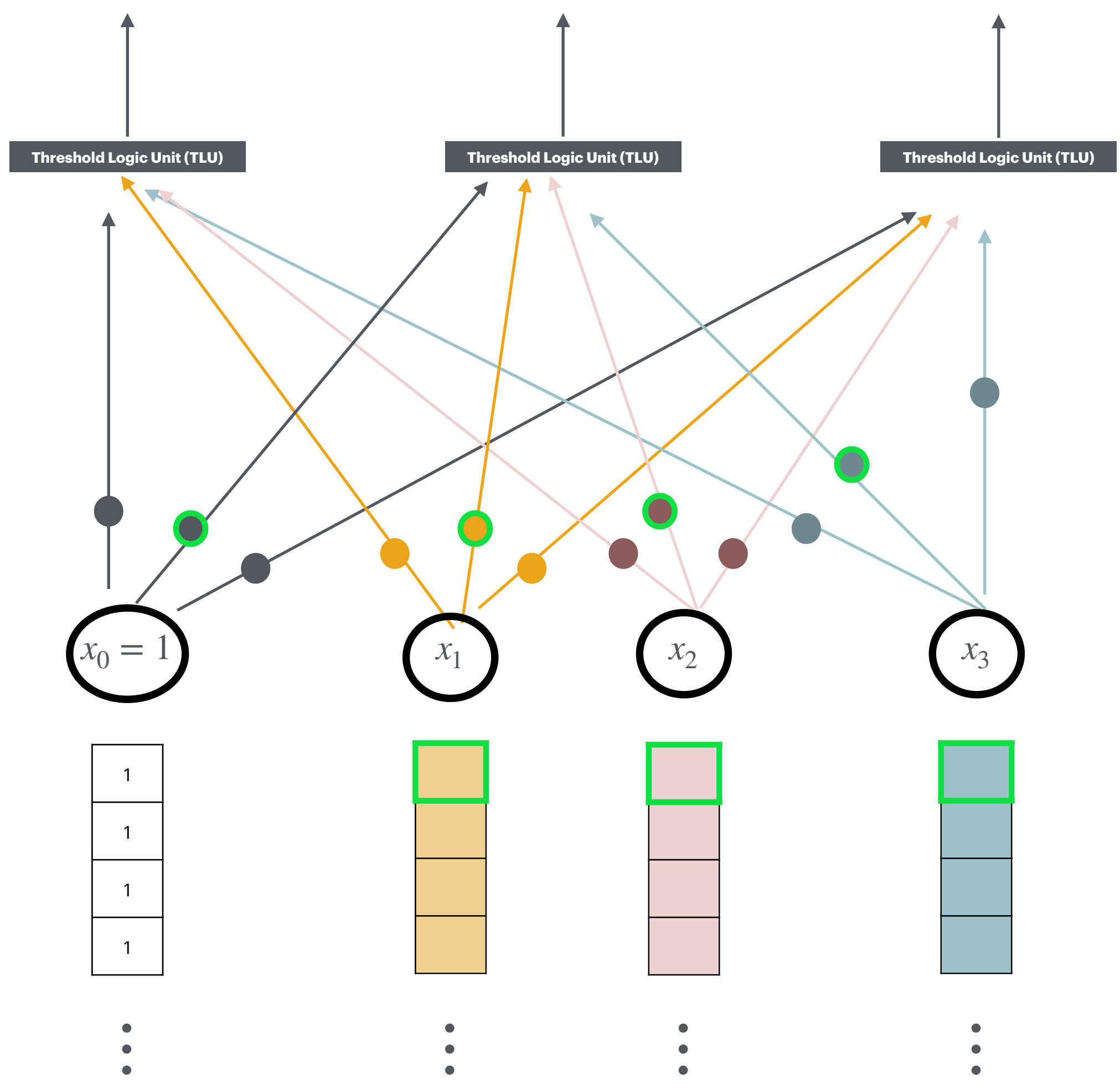
B



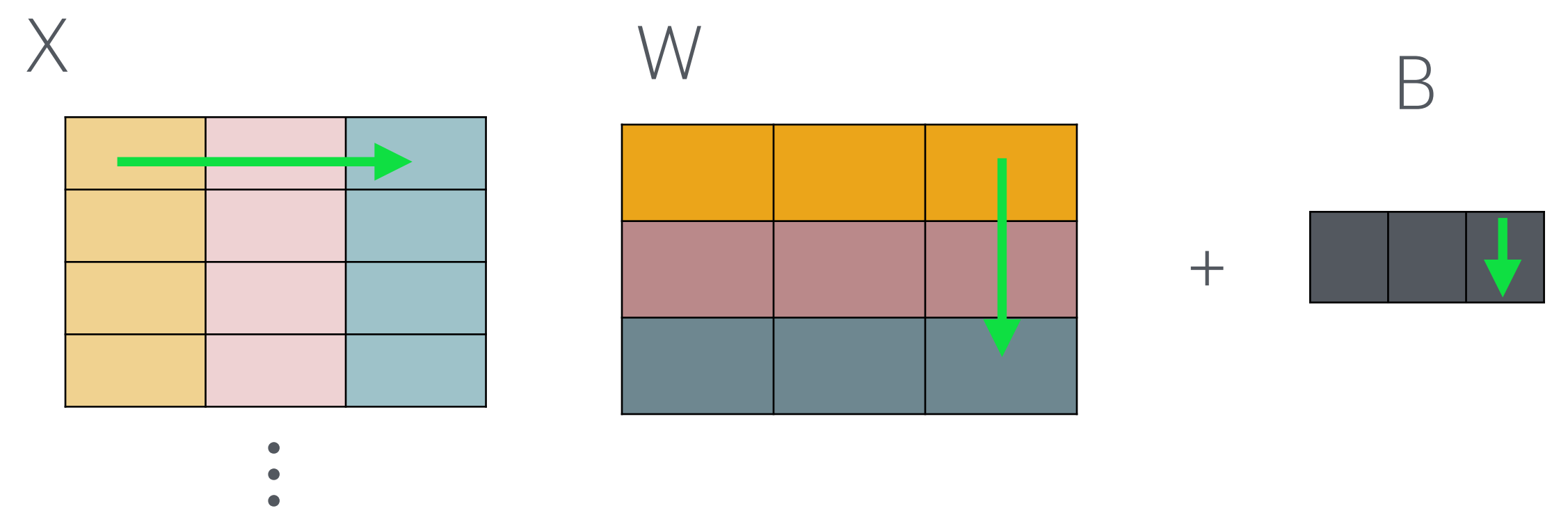
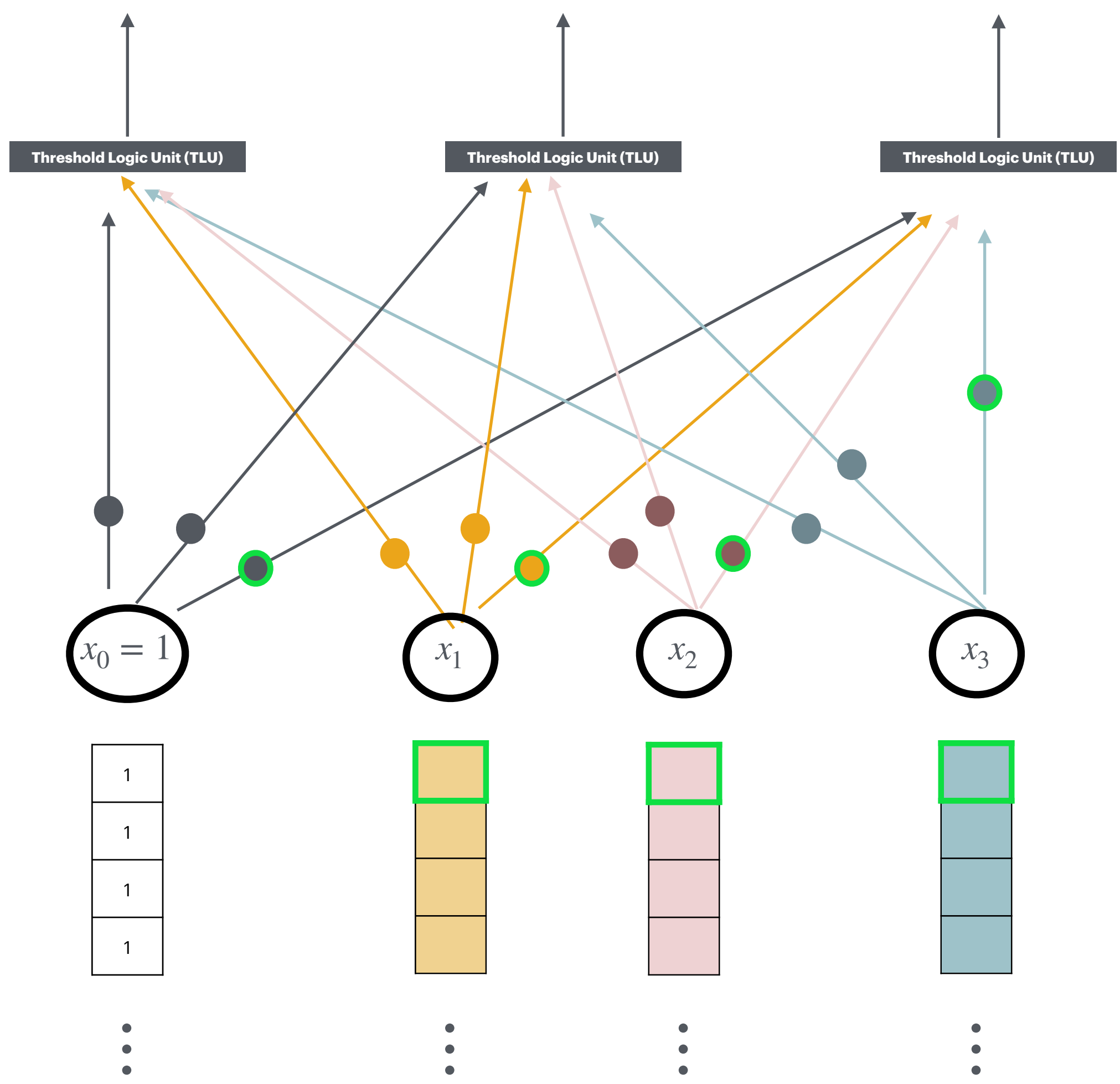
# Logical Computation: Perception



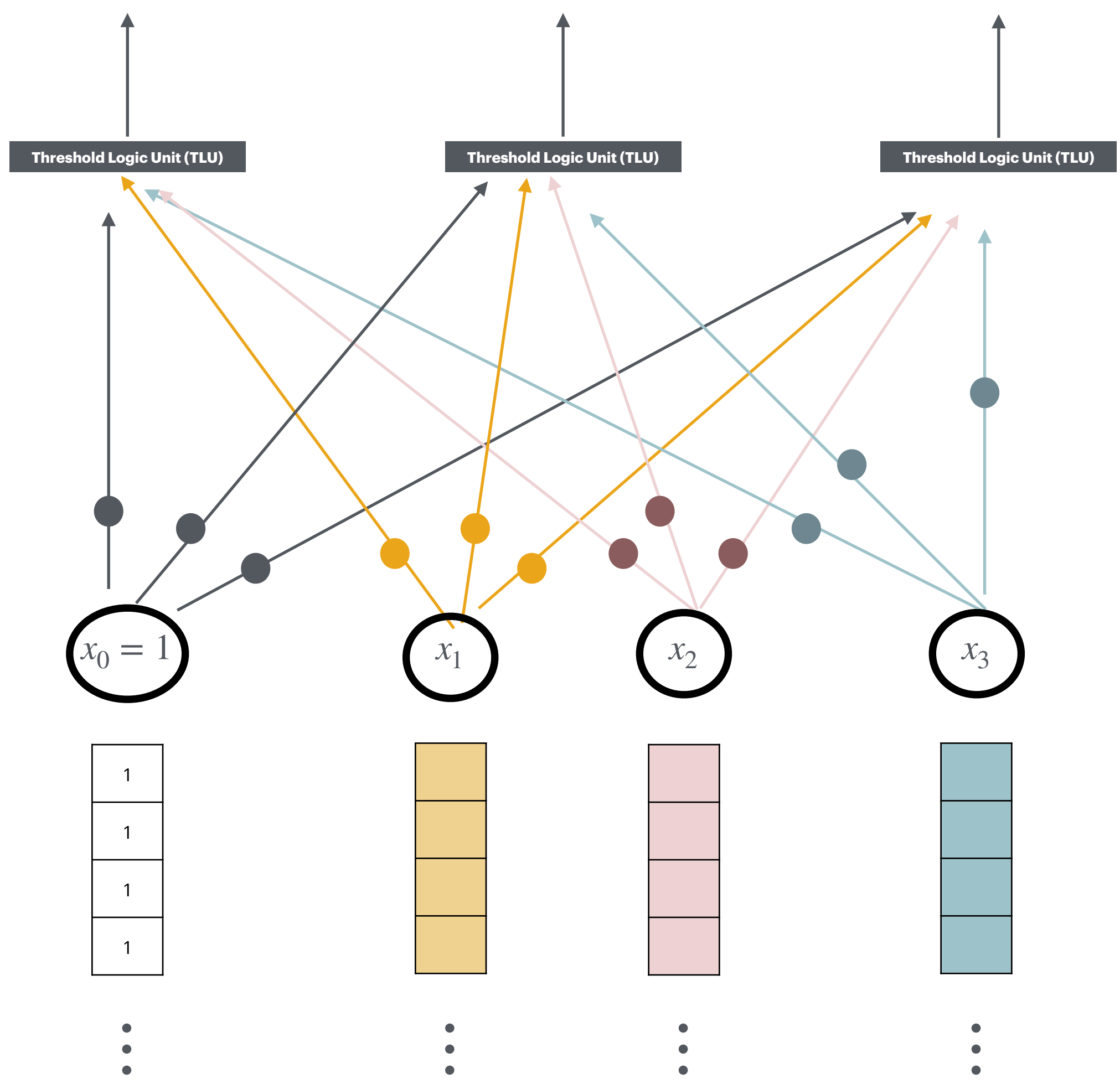
# Logical Computation: Perception



# Logical Computation: Perception



# Logical Computation: Perception

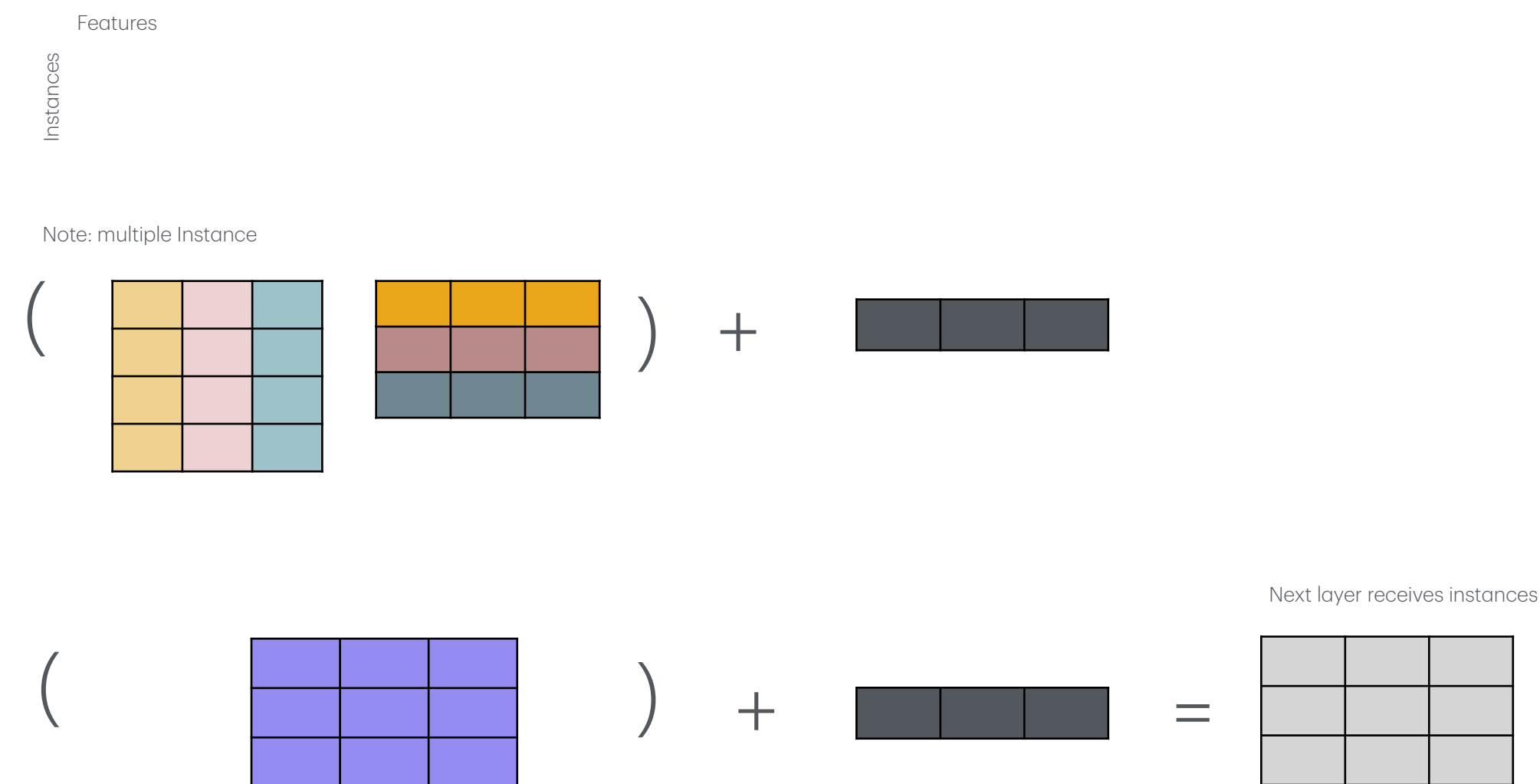
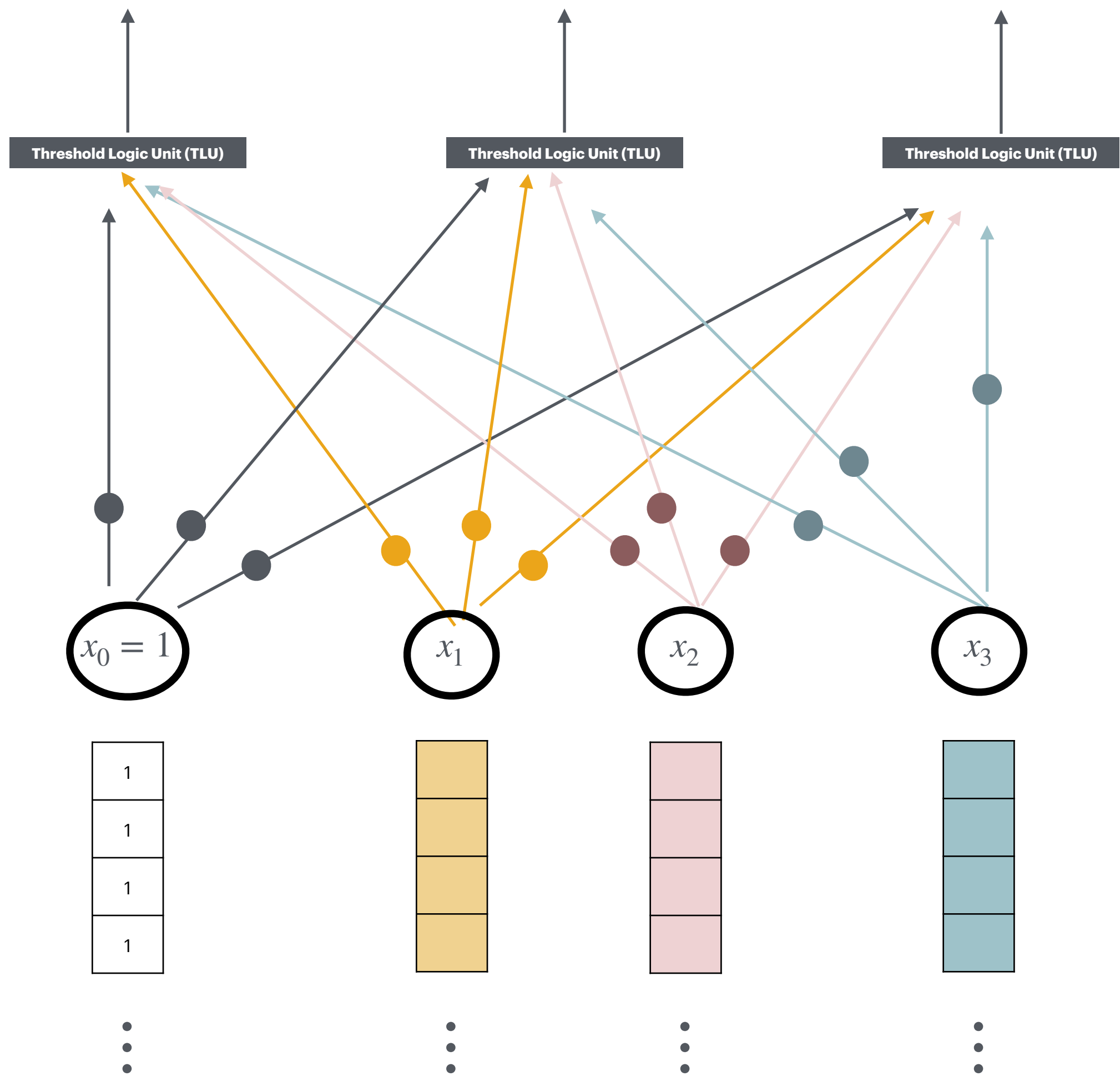


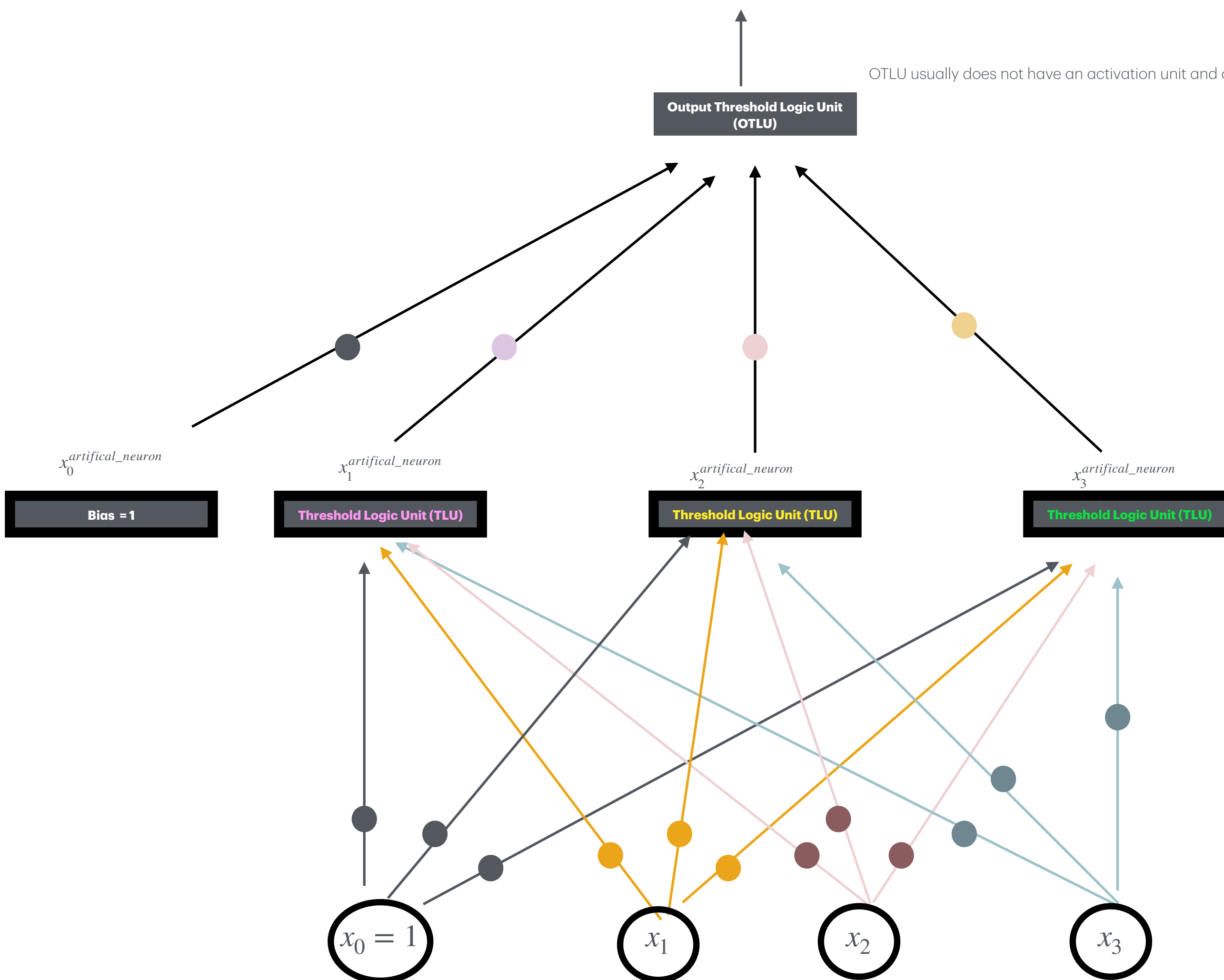
Note: Single Instance

$$\left( \begin{array}{|c|c|c|} \hline \text{yellow} & \text{pink} & \text{blue} \\ \hline \end{array} \begin{array}{|c|c|c|} \hline \text{yellow} & \text{yellow} & \text{yellow} \\ \hline \text{pink} & \text{pink} & \text{pink} \\ \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \end{array} \right) + \begin{array}{|c|c|c|} \hline \text{dark gray} & \text{dark gray} & \text{dark gray} \\ \hline \end{array}$$
$$\left( \begin{array}{|c|c|c|} \hline \text{blue} & \text{blue} & \text{blue} \\ \hline \end{array} \right) + \begin{array}{|c|c|c|} \hline \text{dark gray} & \text{dark gray} & \text{dark gray} \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline \text{light gray} & \text{light gray} & \text{light gray} \\ \hline \end{array}$$

Next layer receives single instances vector

# Logical Computation: Perception



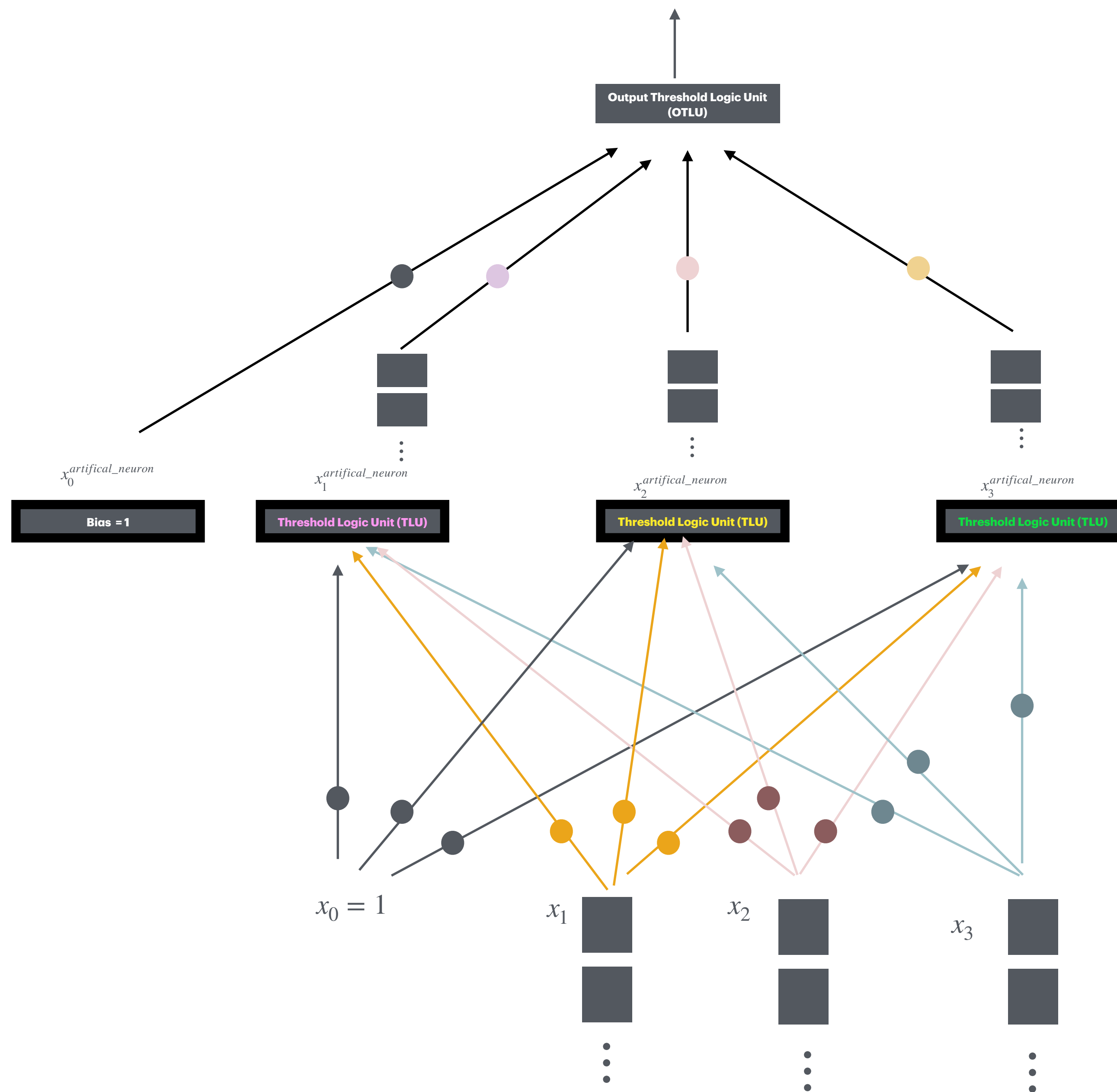


OTLU usually does not have an activation unit and outputs the weighted sum

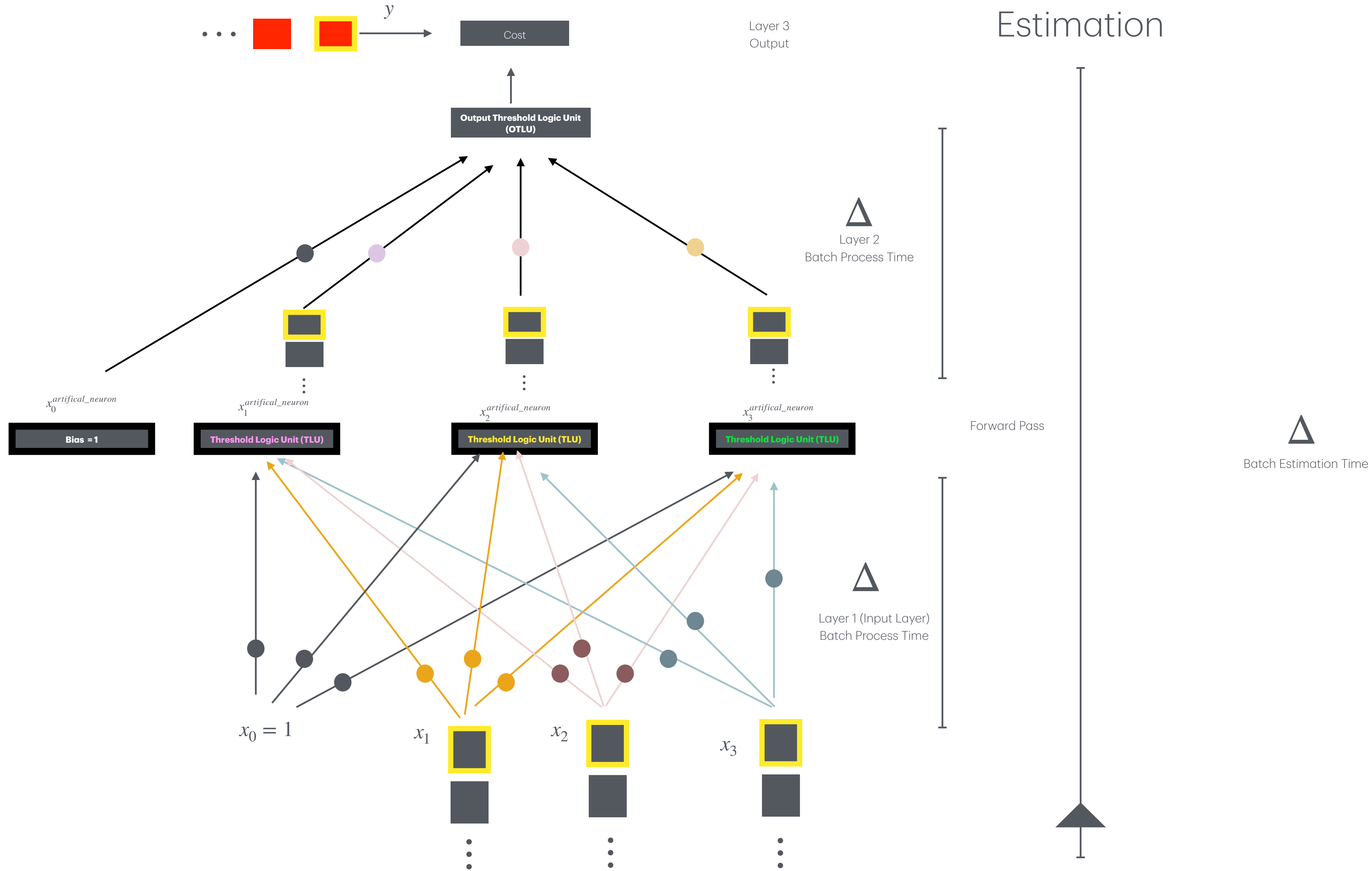
$$\begin{pmatrix} \text{pink} & \text{yellow} & \text{green} \\ \text{purple} & \text{pink} & \text{yellow} \end{pmatrix} + \text{grey} = \text{OTLU}$$

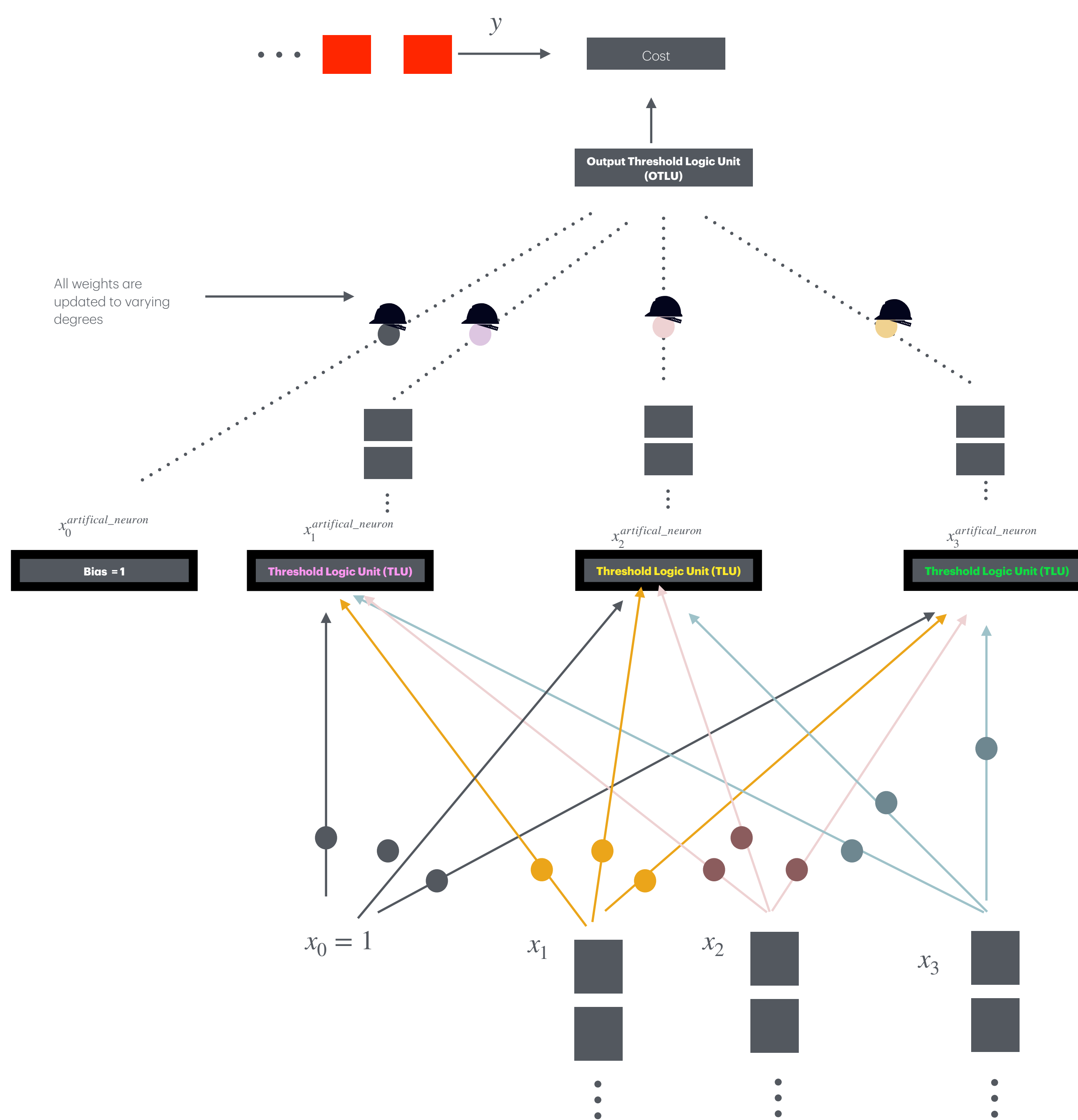
$$\begin{pmatrix} \text{pink} \\ \text{yellow} \\ \text{green} \end{pmatrix} + \text{grey} = \text{OTLU}$$





Networks handle mini-batch at a time





Reverse Pass

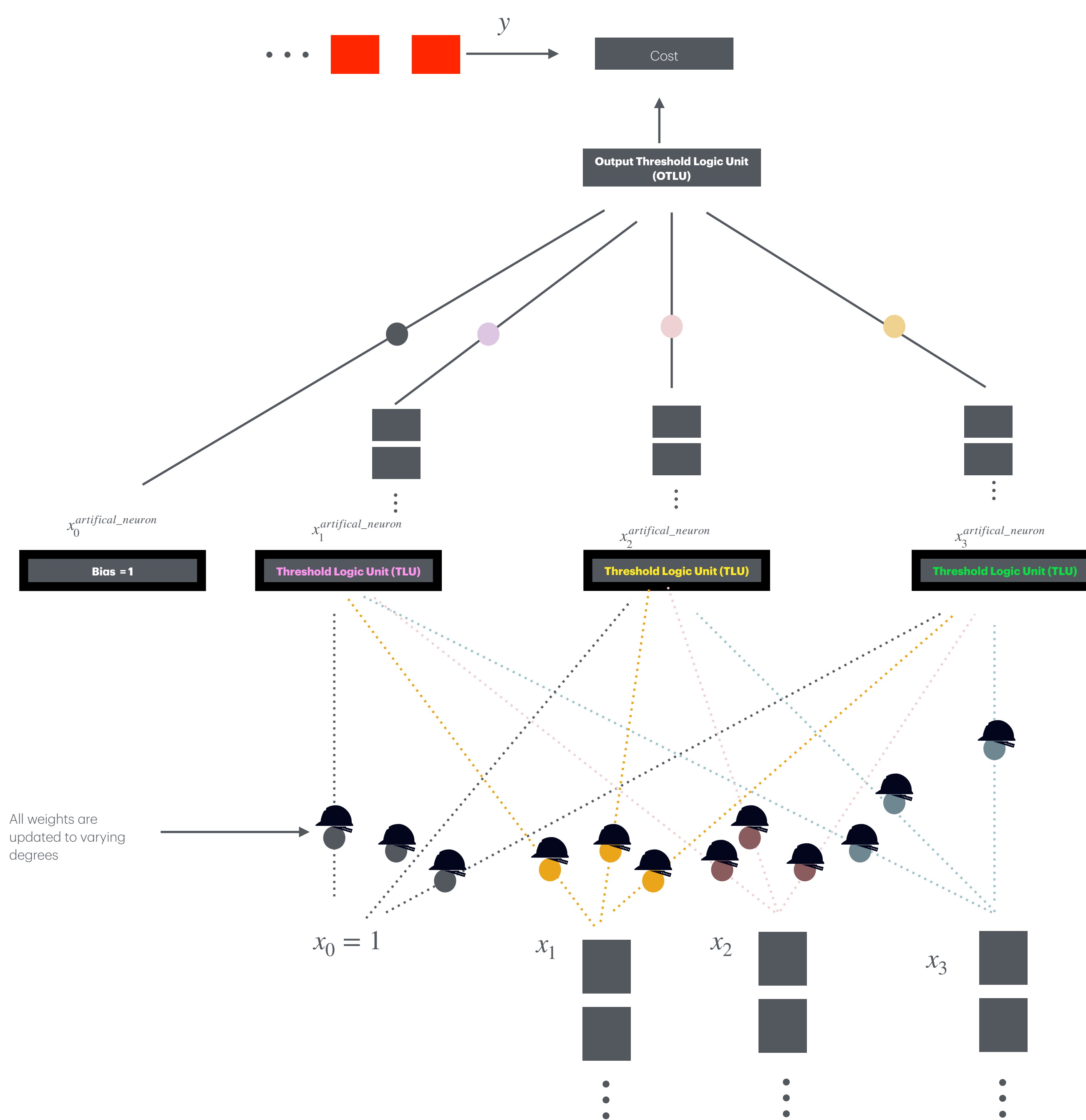
How much **layer 2 connections** contributed to high cost (i.e. high error)

Cost/Error gradients are measured across connections (weights)

$$\frac{Cost}{w_{some\_connection}}$$

Gradient Descent performed on all connections (weights) using error gradients

Note: Input batch persistence is required for reverse algorithm



Reverse Pass

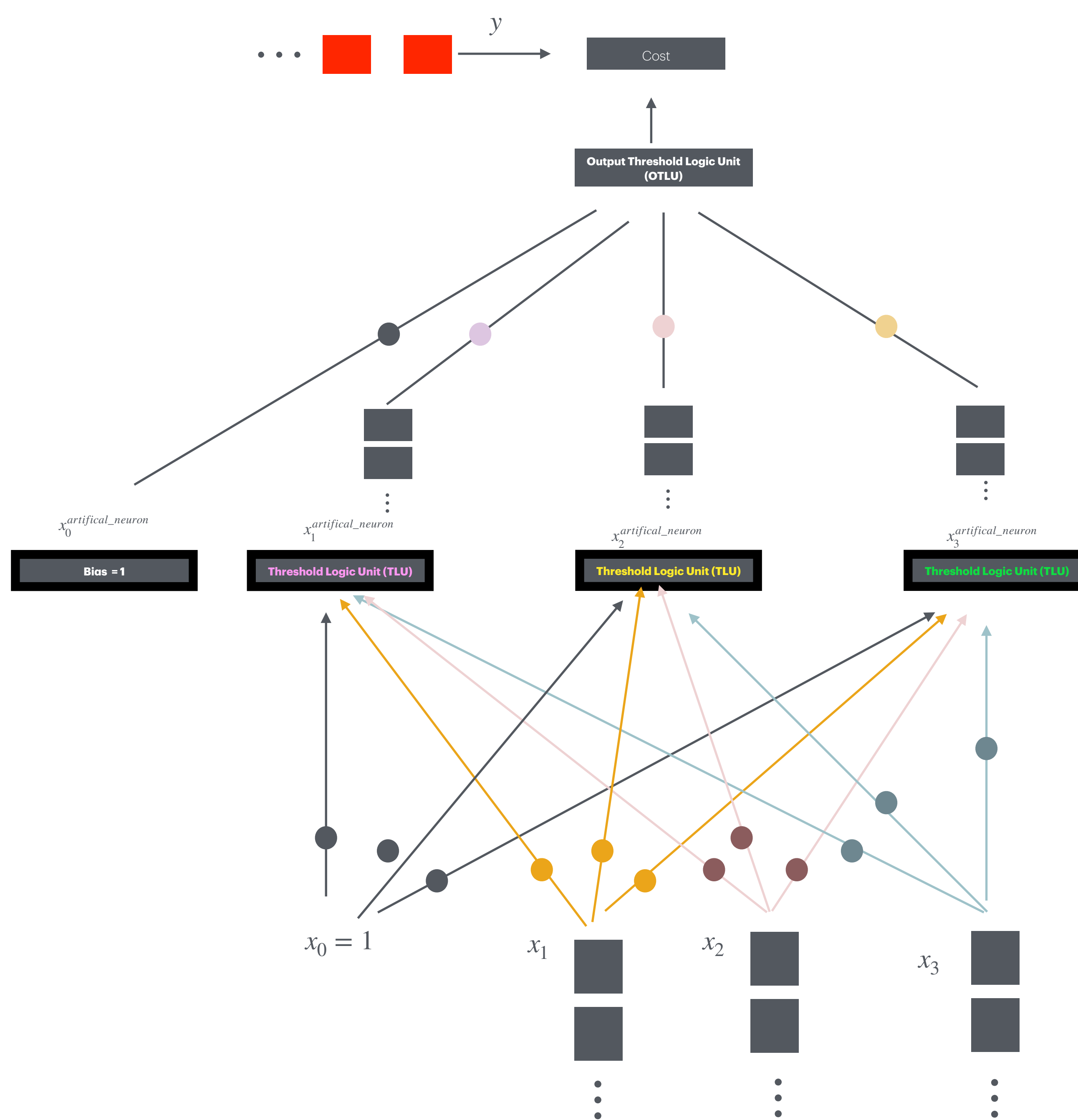
How much **layer 1 connections** contributed to high cost (i.e. high error)

Cost/Error gradients are measured across connections (weights)

$$\frac{Cost}{W_{some\_connection}}$$

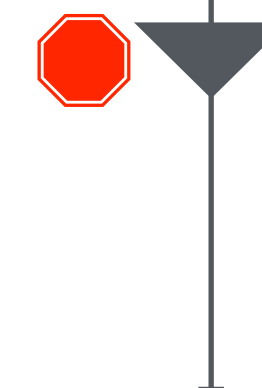
Gradient Descent performed on all connections (weights) using error gradients

Note: Input batch persistence is required for reverse algorithm



Reverse Pass

**Stop** after performing weight update using Gradient Descent on all the connections in each layer



: Activation Function



Activation Function:

Linear Regression/Classifiers :

- **Heaviside**
- **Sign Function**

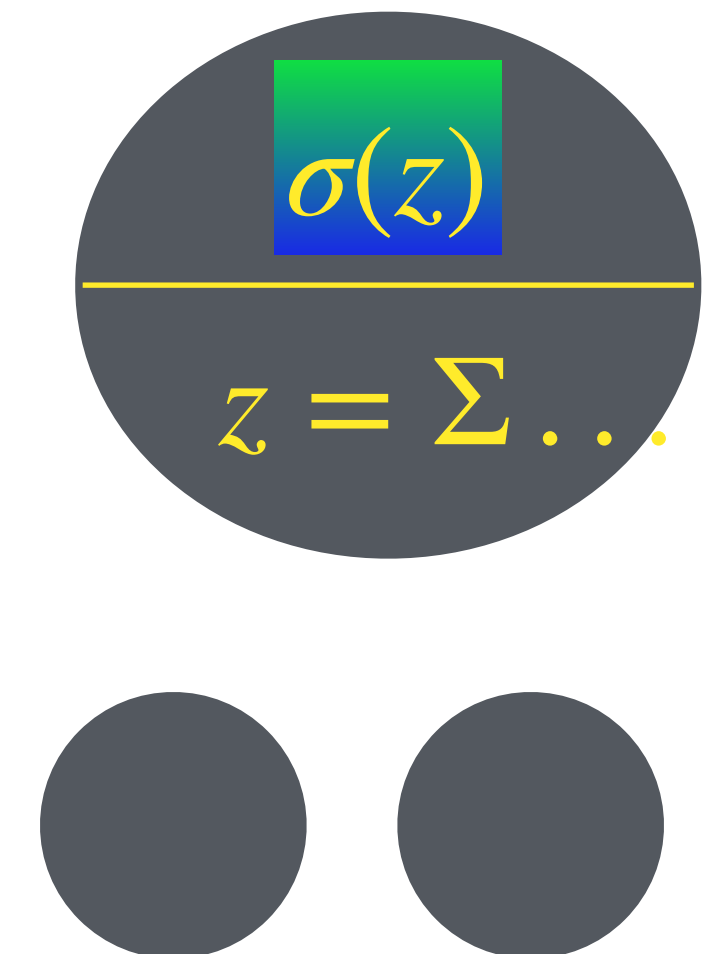
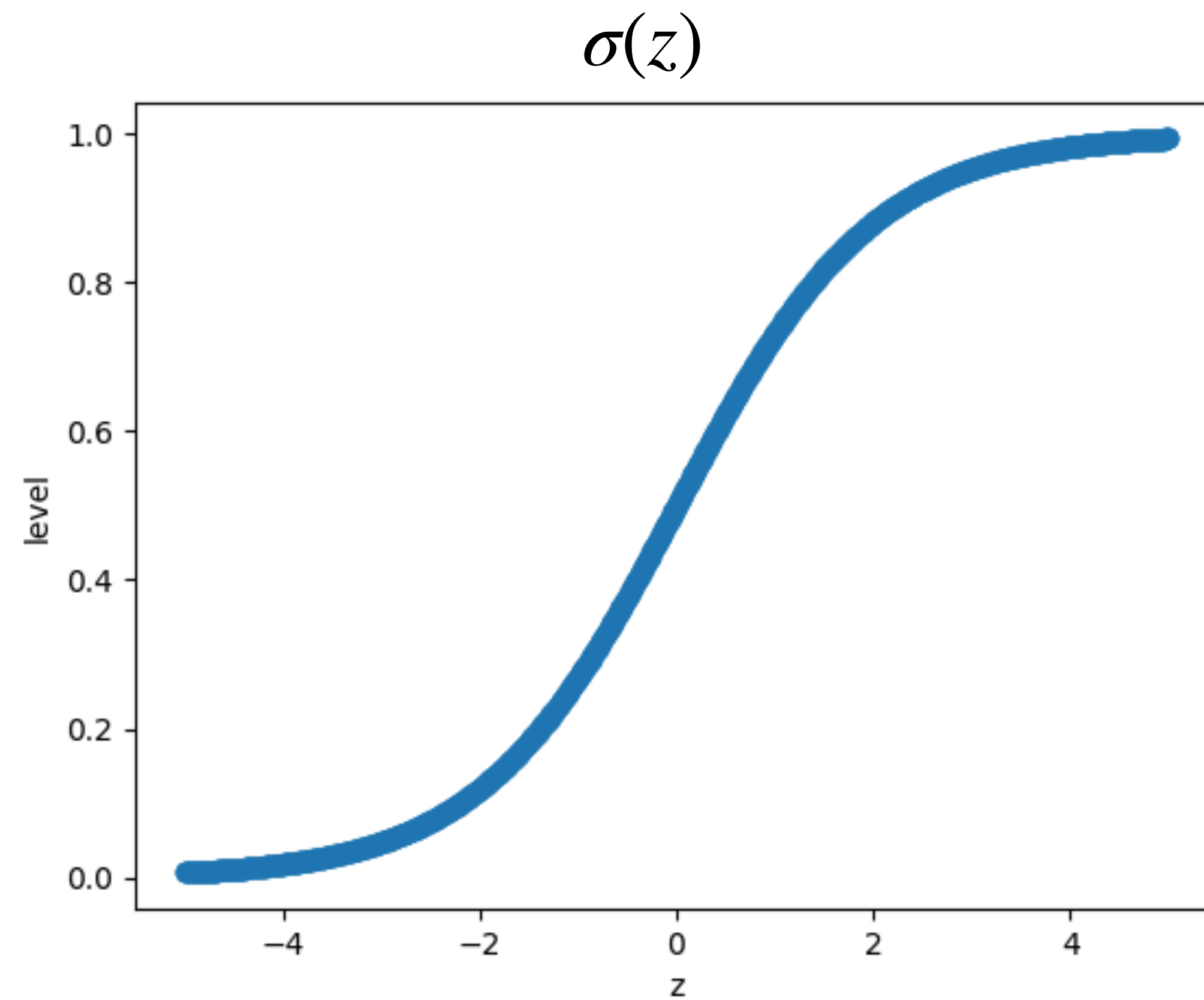
Nonlinear Regression/Classifiers :

- **Sigmoid Function**
- **Hyperbolic Tangent Function**
- **Rectified Linear Unit Function**

Non-Linear activation functions can be used on linearly models as well

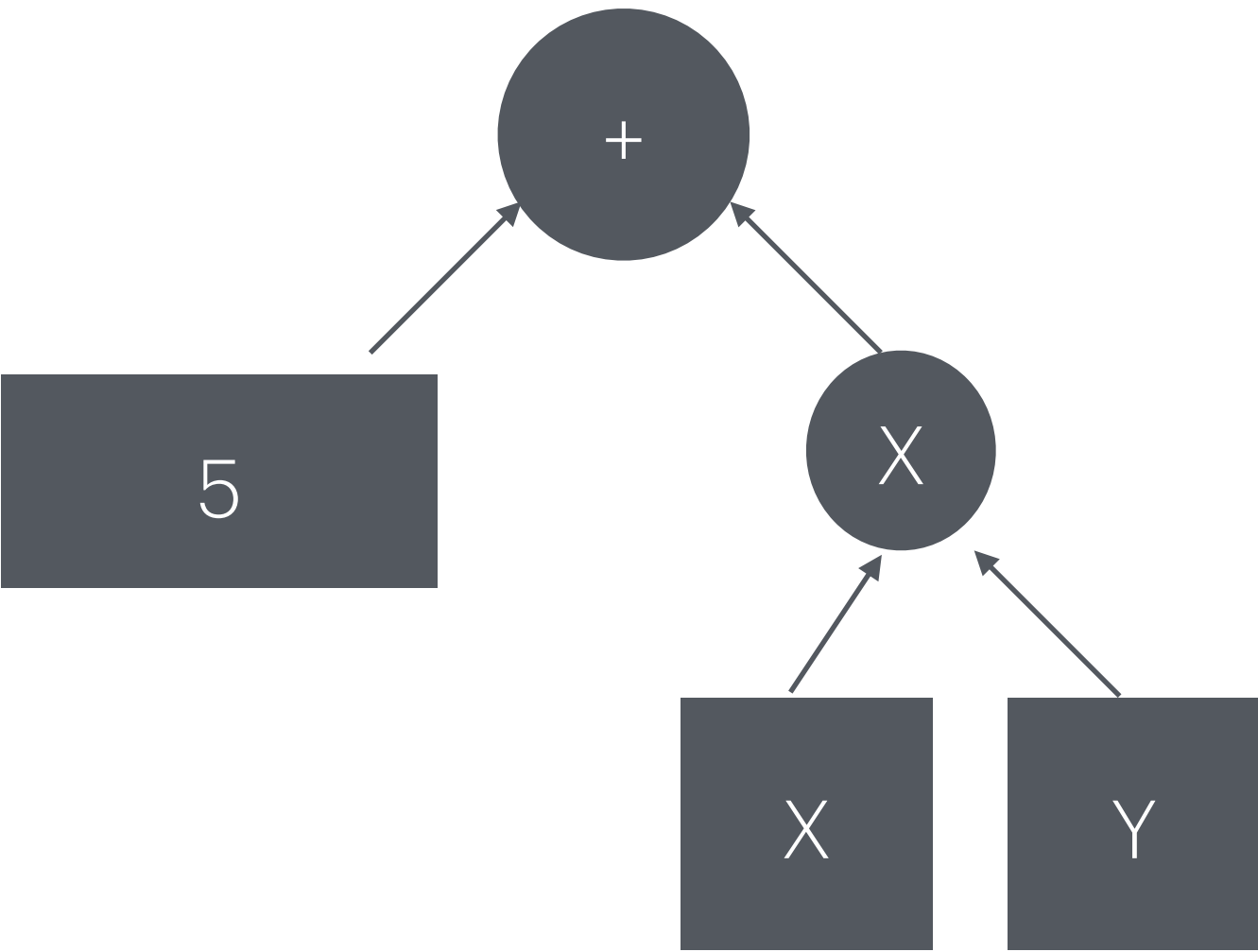
# Activation Function

**Biological neurons** have been observed to implement a roughly **sigmoid activation** function



# Forward-Mode AutoDifferentiation

$g(x, y) = 5 + xy$



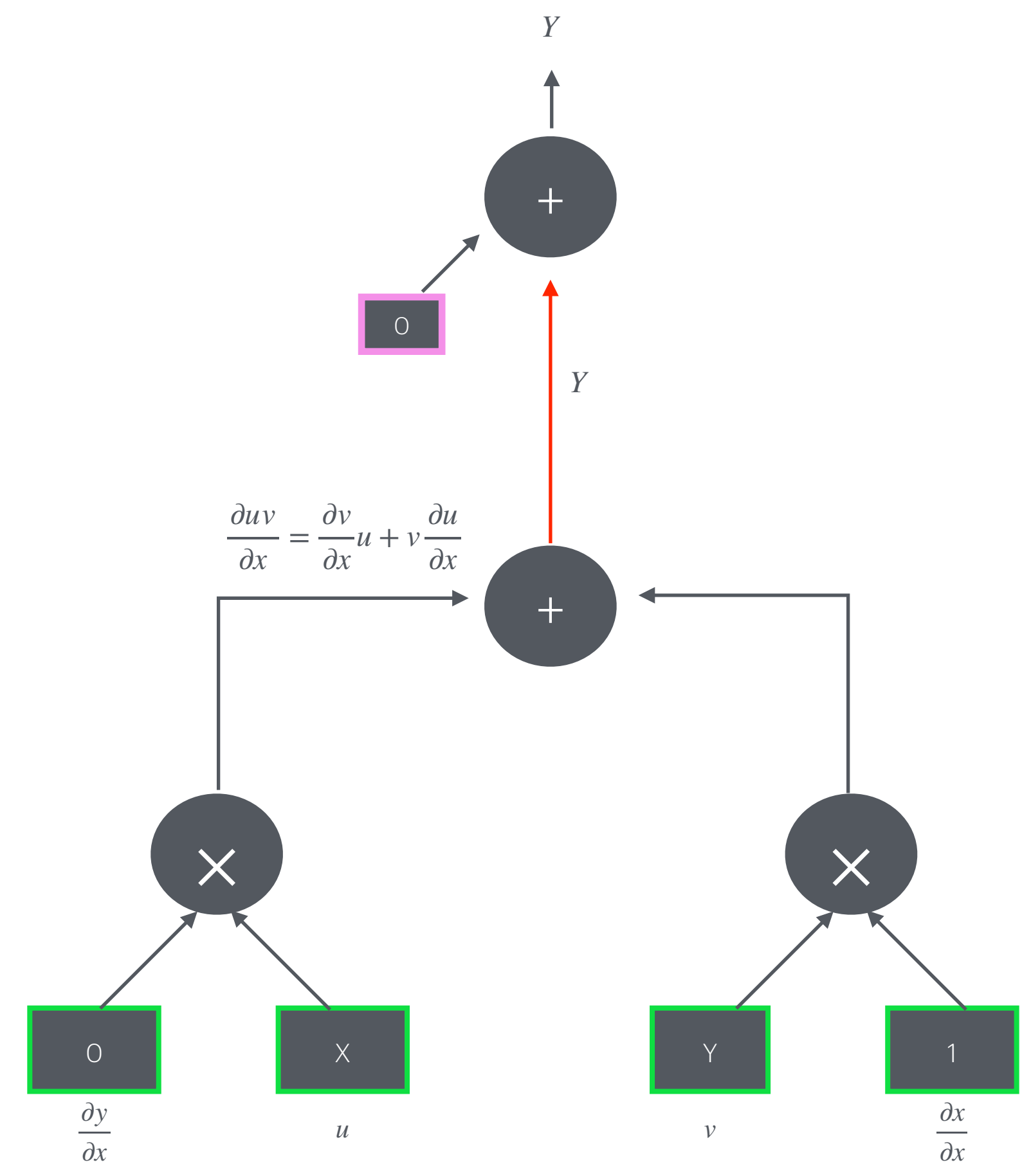


Forward-Mode AutoDifferentiation

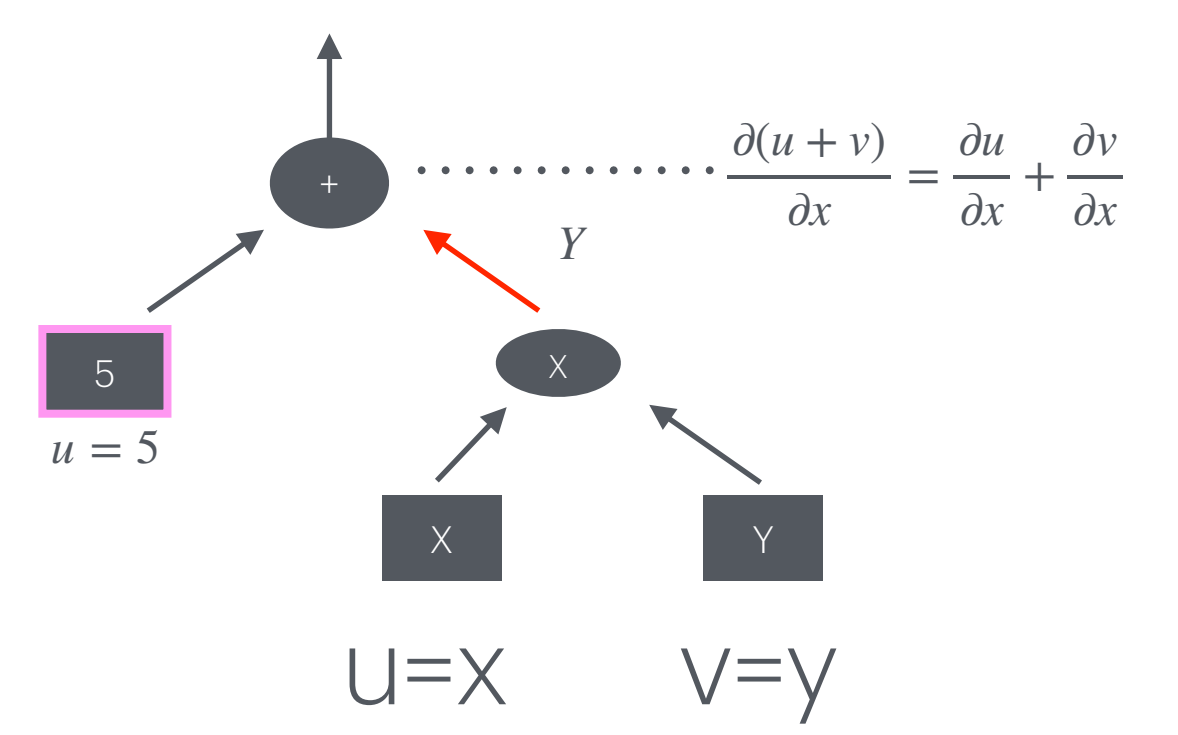
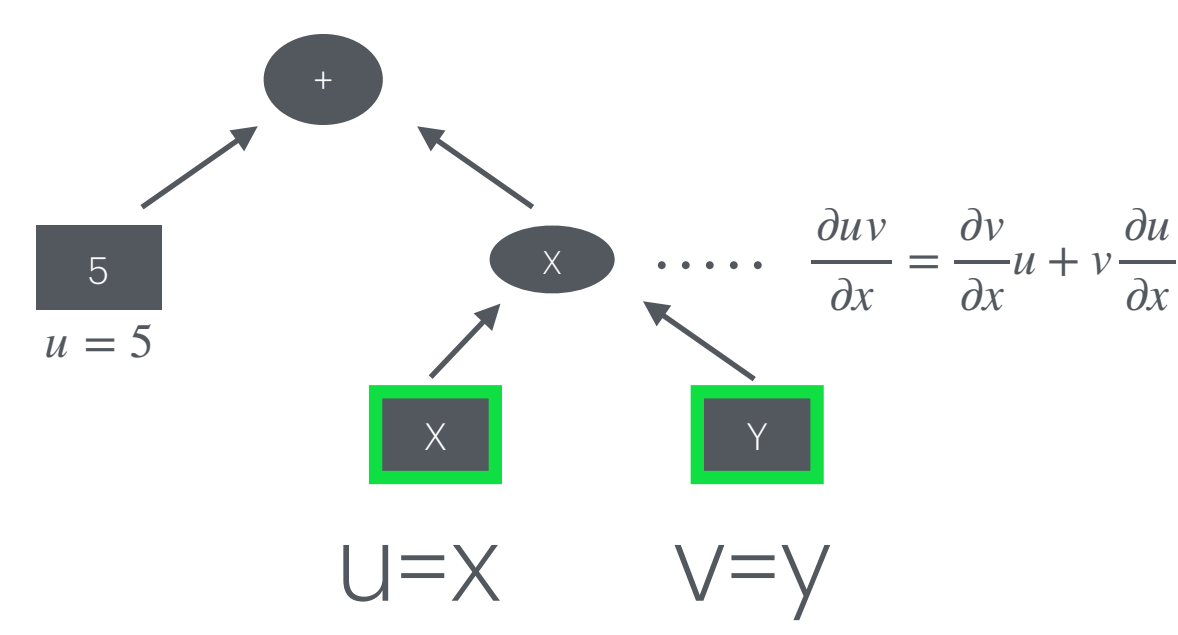
$g(x, y) = 5 + xy$

Partial Derivative  $\frac{g(x, y)}{\partial x}$

Symbolic Differentiation (created from AutoDiff)

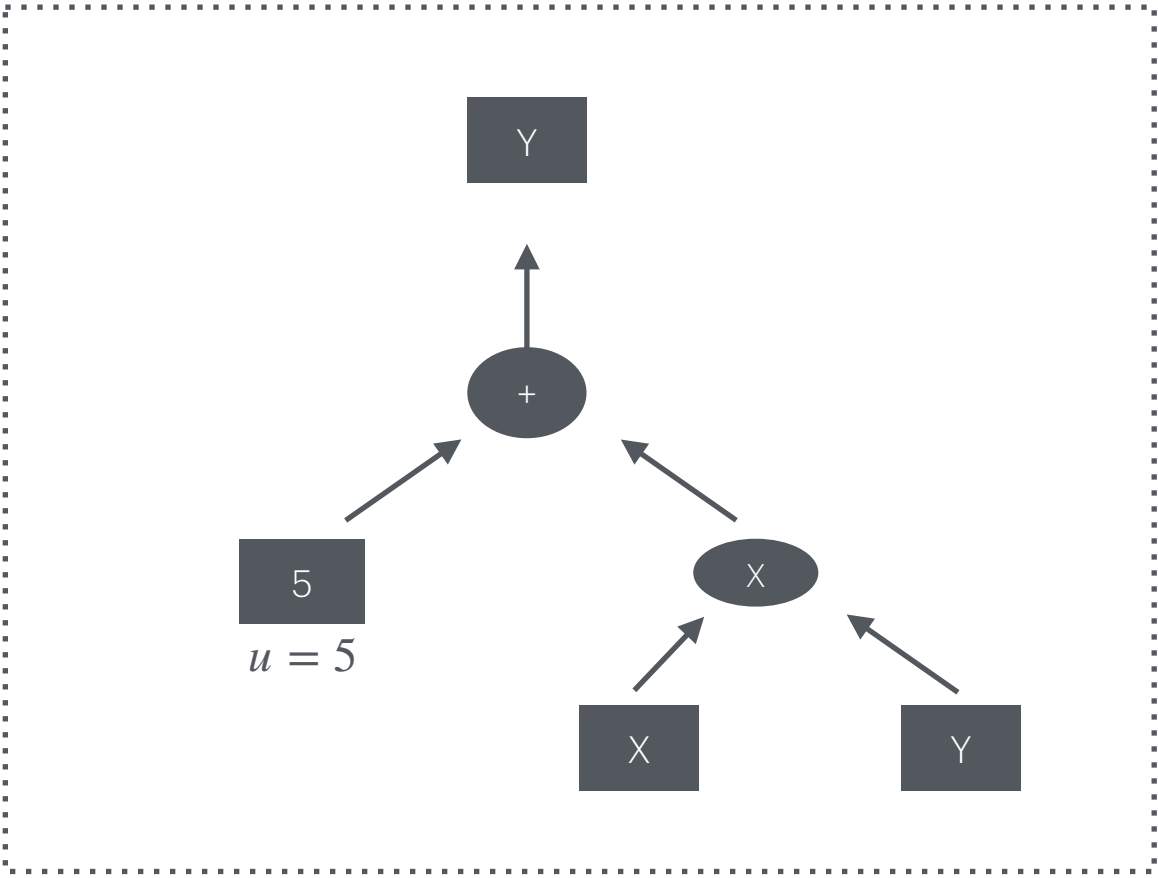
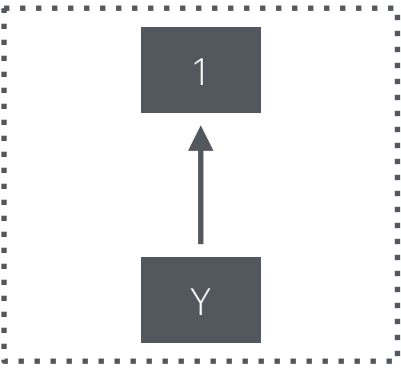
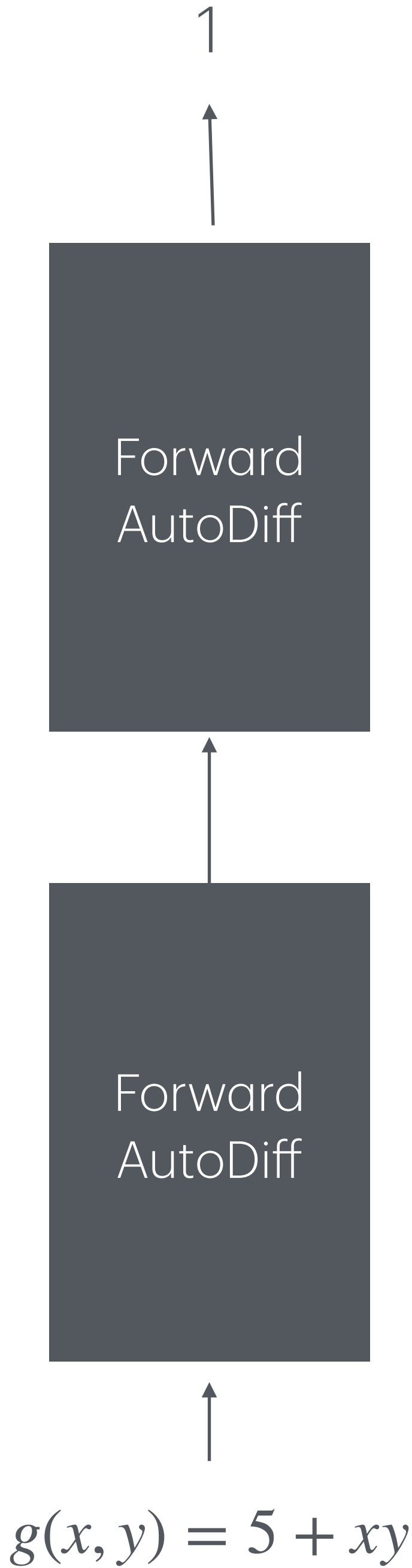


AutoDiff Computation Graphs



# Forward-Mode AutoDifferentiation

Chain Forward Mode AutoDiffs



$$\frac{\partial f(3,4)}{\partial x} = ?$$

$$f(3 + \epsilon, 4) = f(3, 4) + f'(3, 4)\epsilon$$

$$f(3 + \epsilon, 4) = f(3, 4) + \frac{\partial f(3, 4)}{\partial x}\epsilon$$

Run forward diff to find real and  $\epsilon$  components

If  $\epsilon$  is a infinitesimal number with  $\epsilon^2 = 0$ , dual numbers can be used to solve forward-mode autodiff

Rule:  $h(a + \epsilon) = h(a) + h'(a)\epsilon$

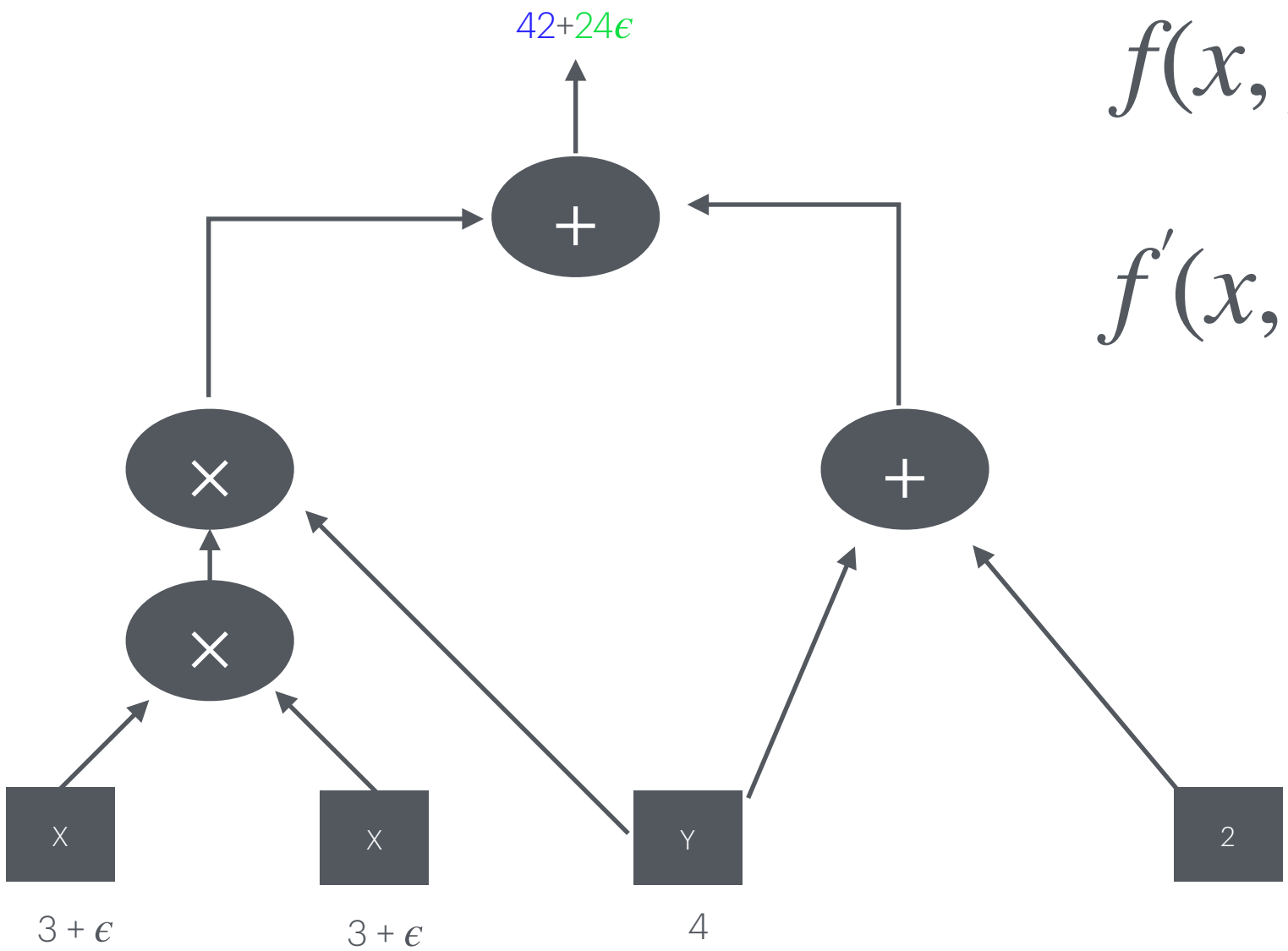


$$f(x, y) = x^2y + y + 2$$

$$f(x, y) = 42 \text{ real component}$$

$$f'(x, y) = 24 \text{ } \epsilon \text{ component}$$

Note:  $\epsilon^2 = 0$

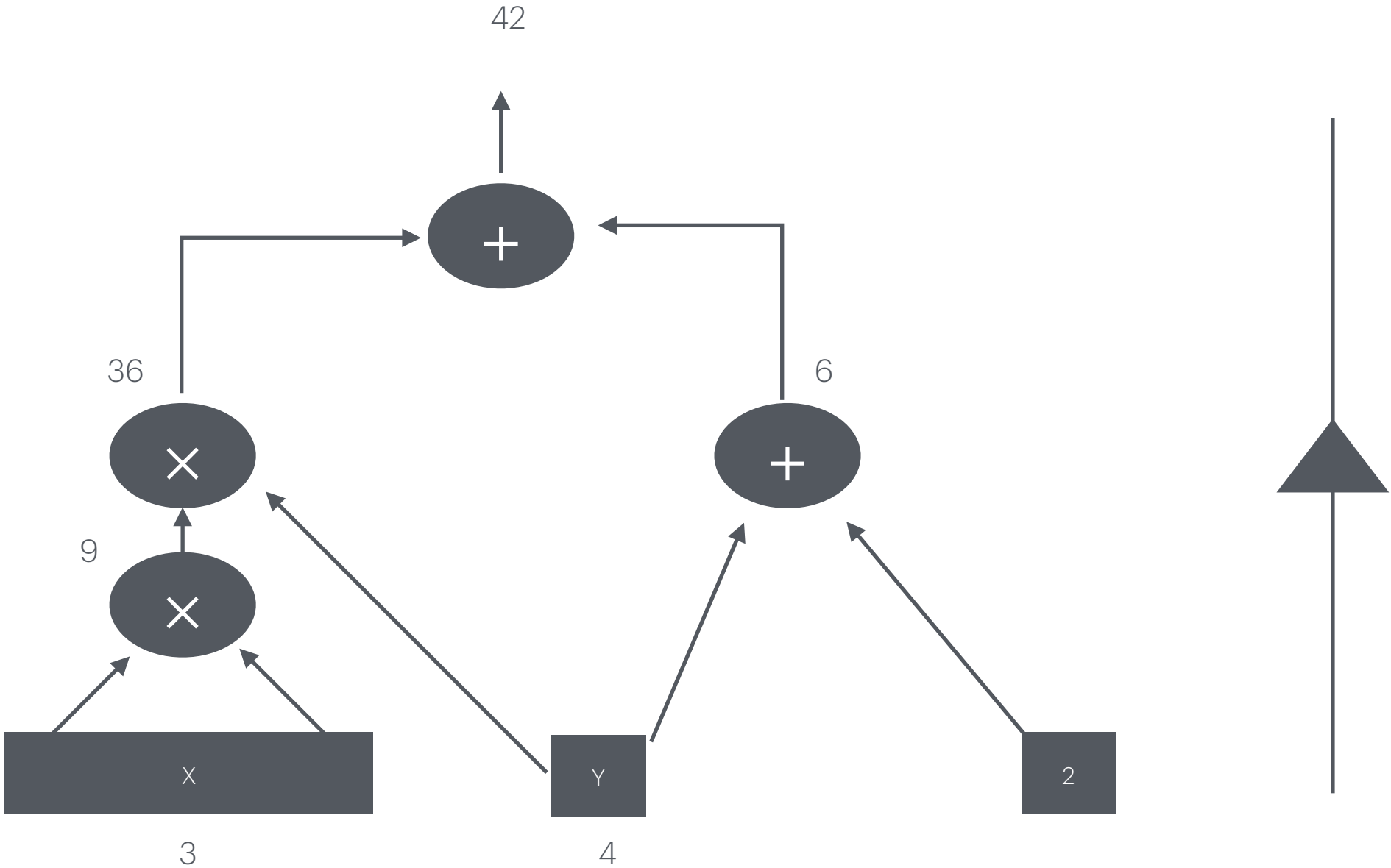


Partial derivative with respect to  $y$  requires the same process

# Reverse-Mode AutoDifferentiation

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

$\frac{\partial f(3,4)}{\partial x} = ?$



$n_7$  change caused by  $n_5$

$f(x, y)$  change caused by  $n_7$

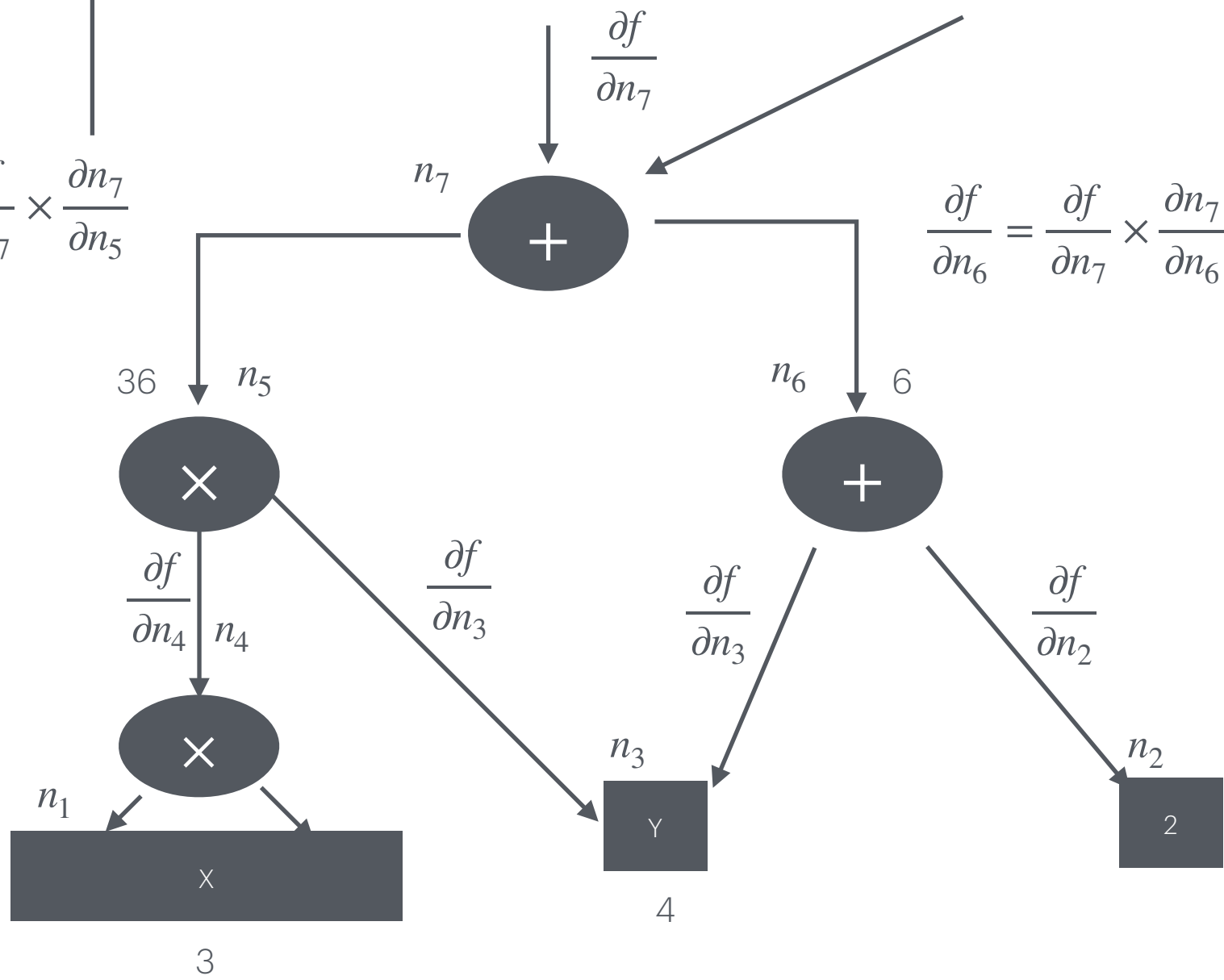
$f(x, y)$  change caused by  $n_5$

$$\frac{\partial f}{\partial n_5} = \frac{\partial f}{\partial n_7} \times \frac{\partial n_7}{\partial n_5}$$

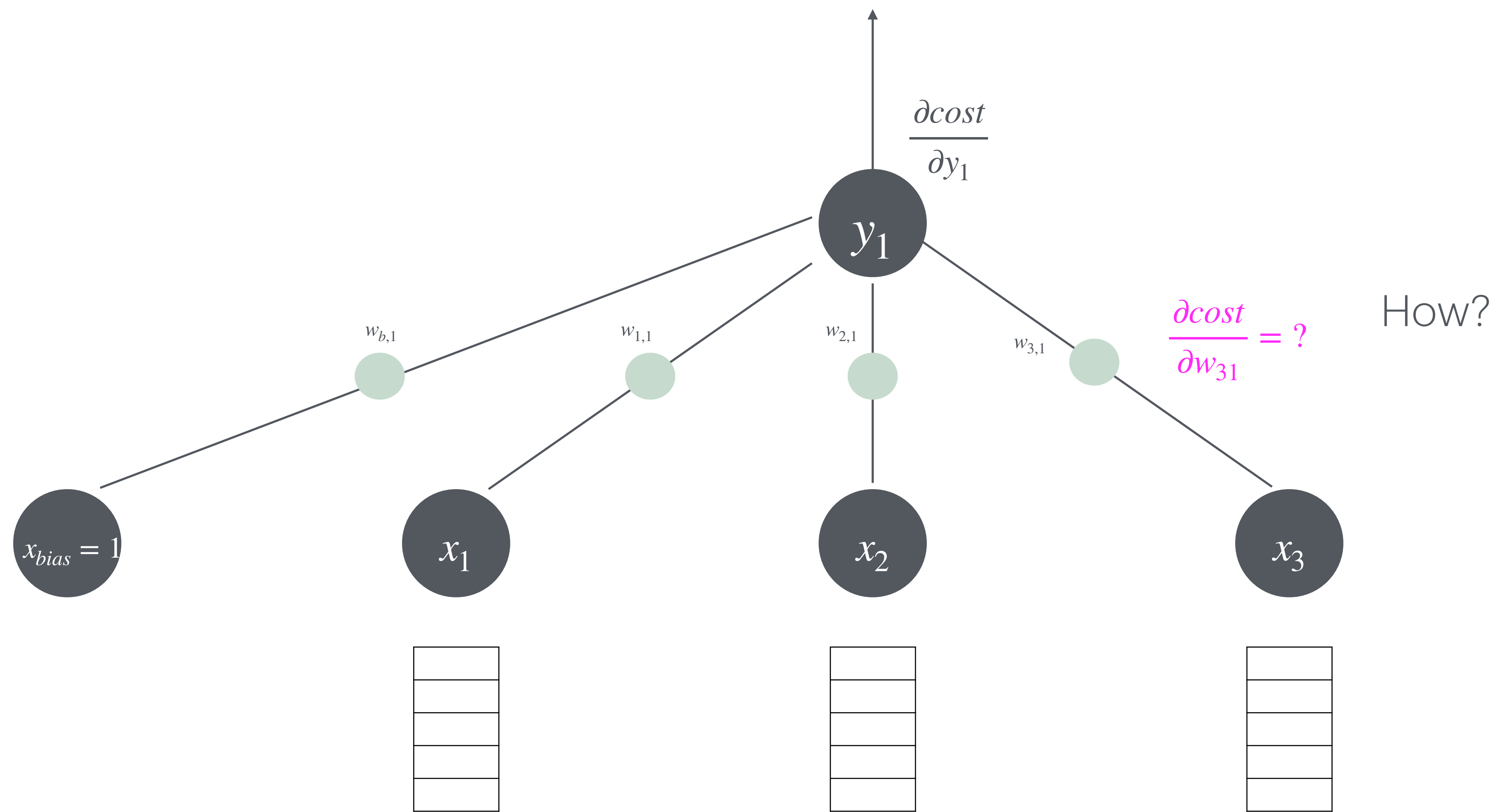
Output  $n_x$  at each node.  $\frac{\partial f}{\partial n_7} = 1$

$$\frac{\partial f}{\partial n_6} = \frac{\partial f}{\partial n_7} \times \frac{\partial n_7}{\partial n_6}$$

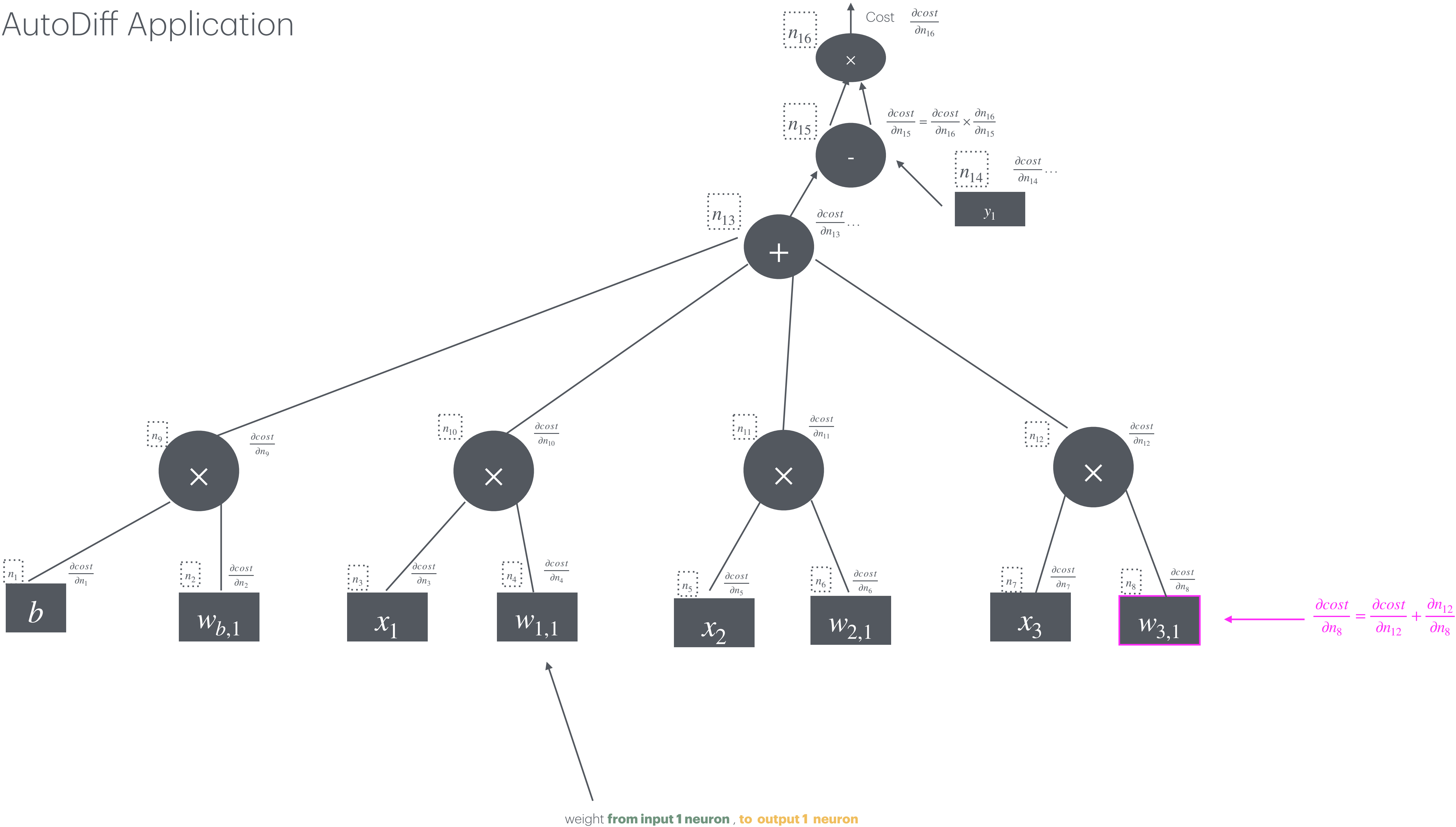
$$\frac{\partial f}{\partial n_1}$$



# AutoDiff Application



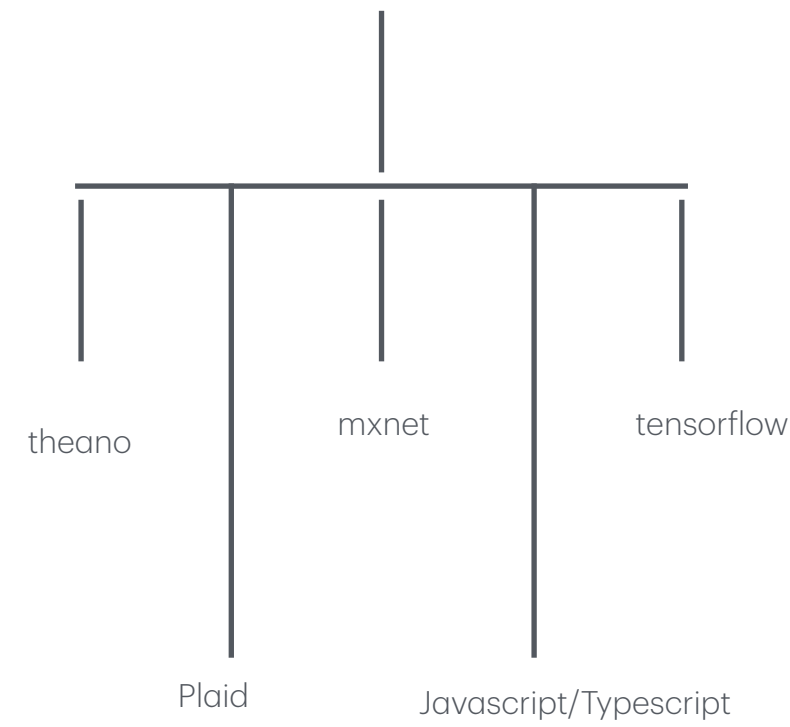
# AutoDiff Application



# Tensorflow

## Keras(API)

Keras(implementation )



from tensorflow import keras

## Libraries containing Keras

- ★ - Tensorflow
- Microsoft Cognitive Toolkit
- Theano
- Keras(API)
- Javascript/Typescript
- PlaidML
- Apple's Core ML
- Apache MXNet

## Tensorflow(API)

Keras(custom implementation )

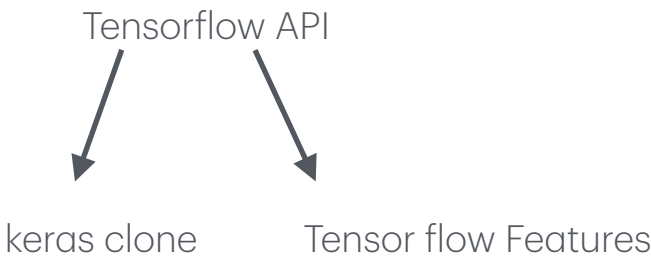
Import tensorflow.keras

## PyTorch(API)

Keras(custom implementation )

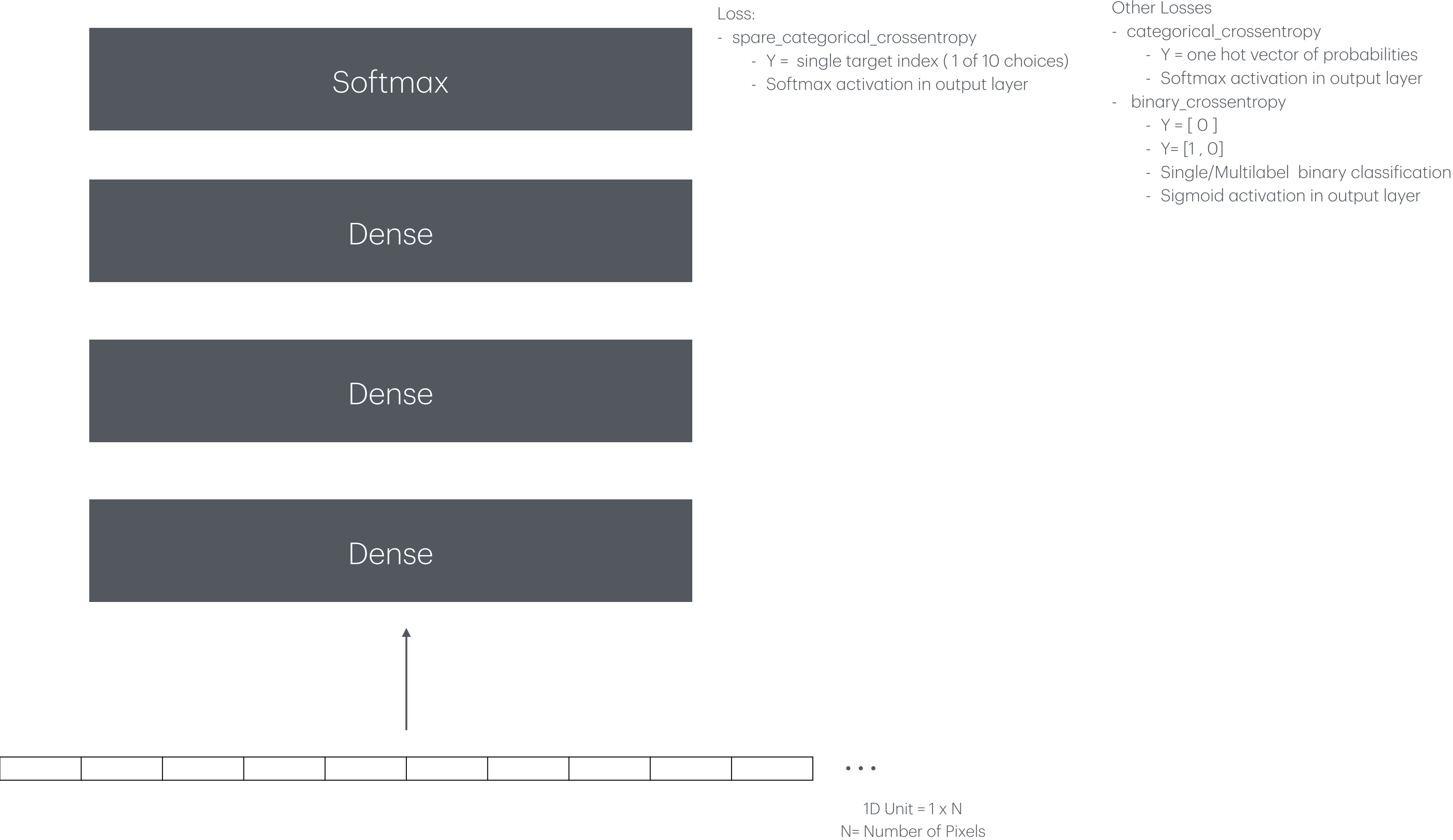
## Libraries containing custom Keras framework

- ★ - Pytorch
- ★ - Tensorflow



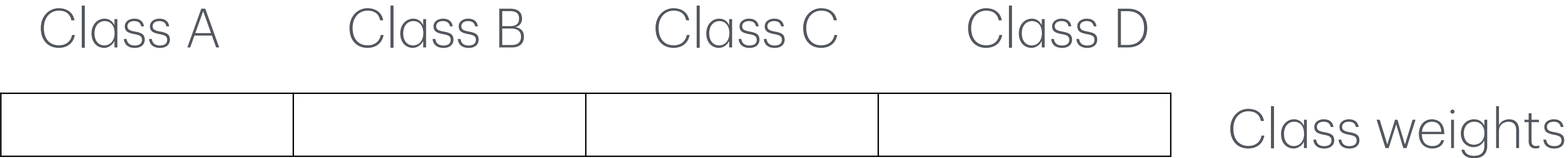
★ – popular

# Sequential Model: Classify Fashion MNIST

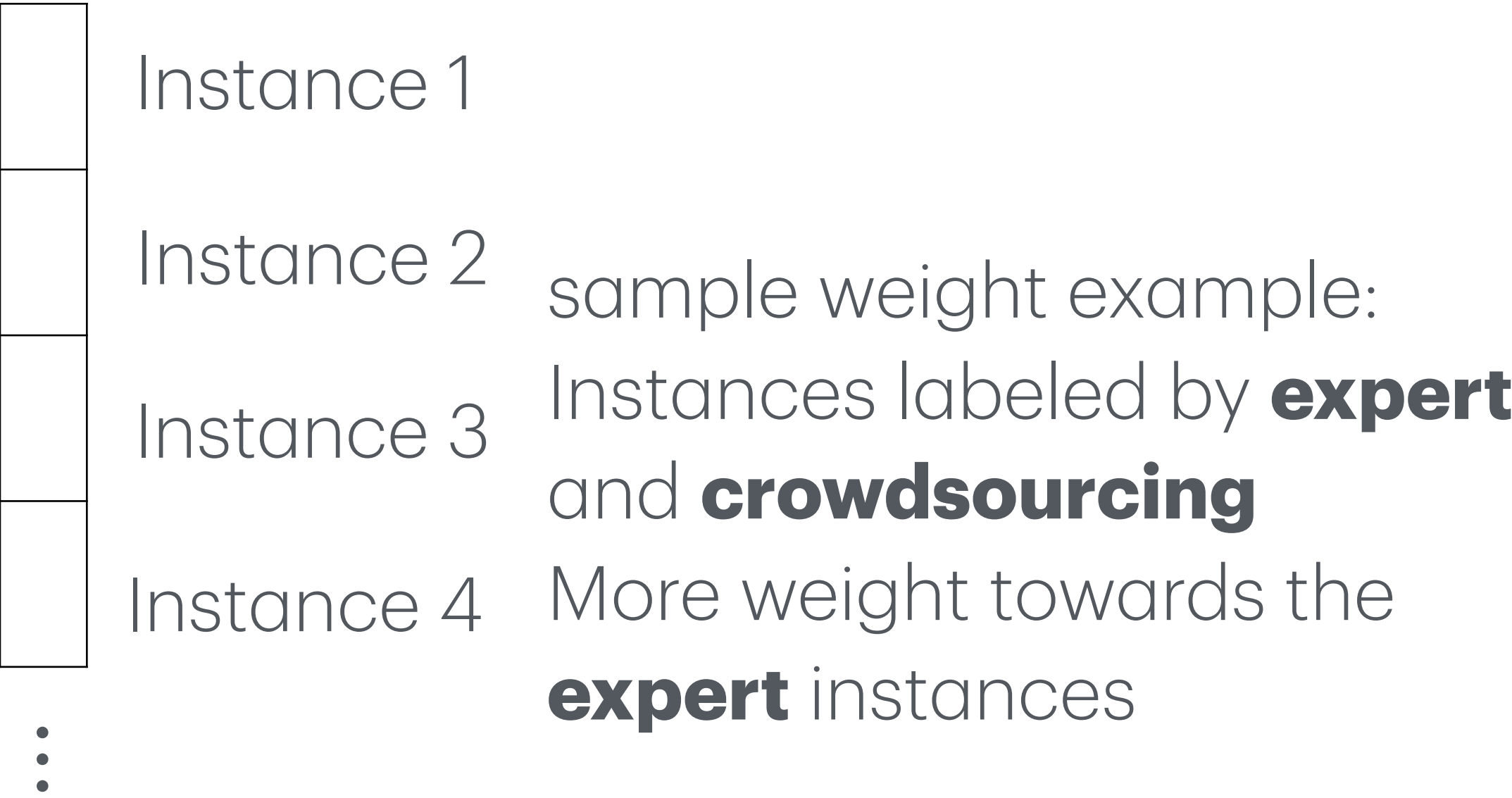




Dealing with skewed data



class weight example:  
Class A **overrepresented** in  
dataset. Give **more weight** to  
Class B,C, and D



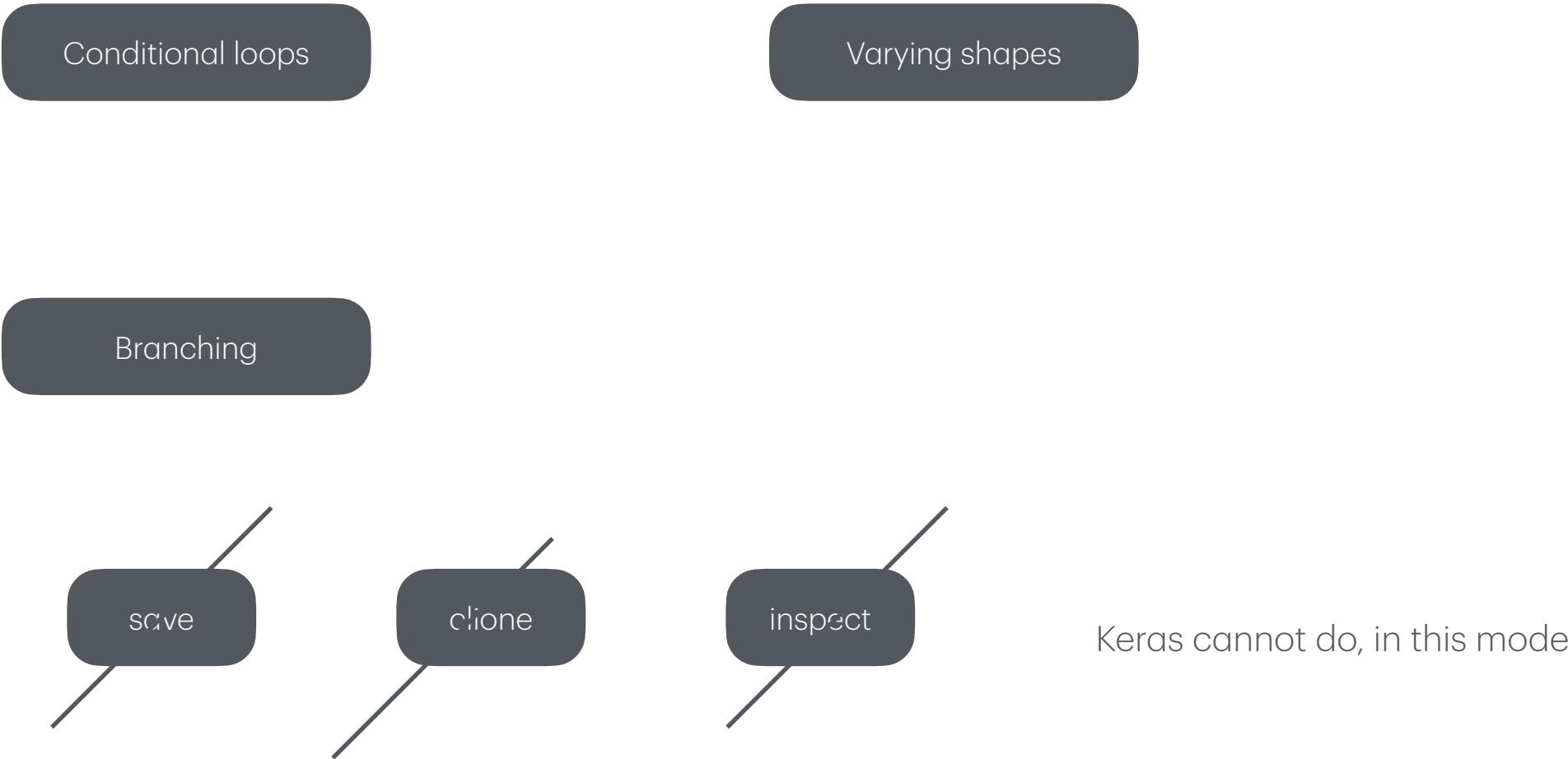
Samples weights

# Saving

- Functional
- Sequential



- Subclassing

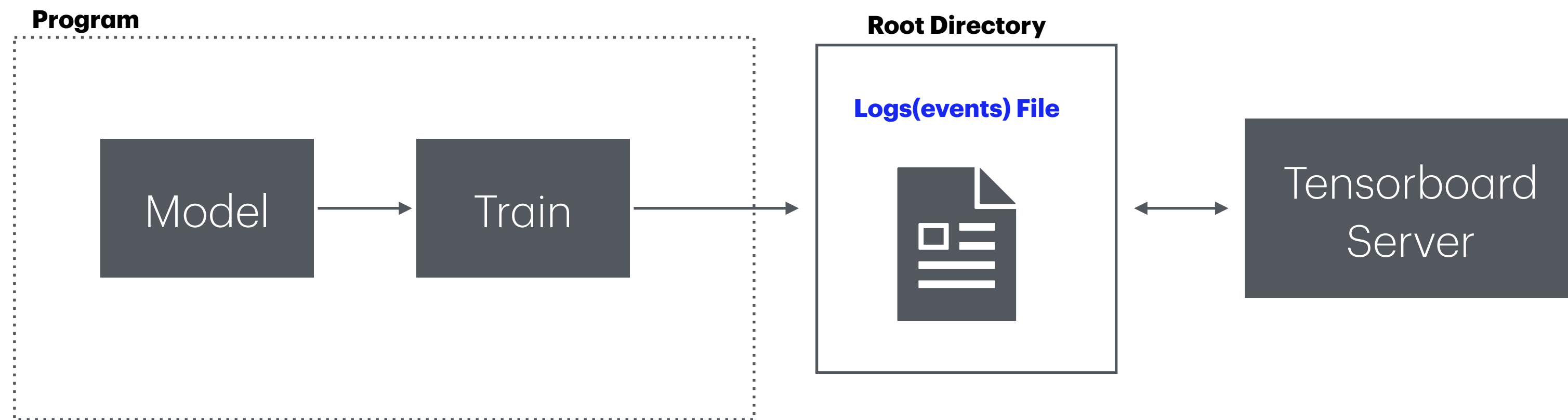


***Save and load model weights yourself***

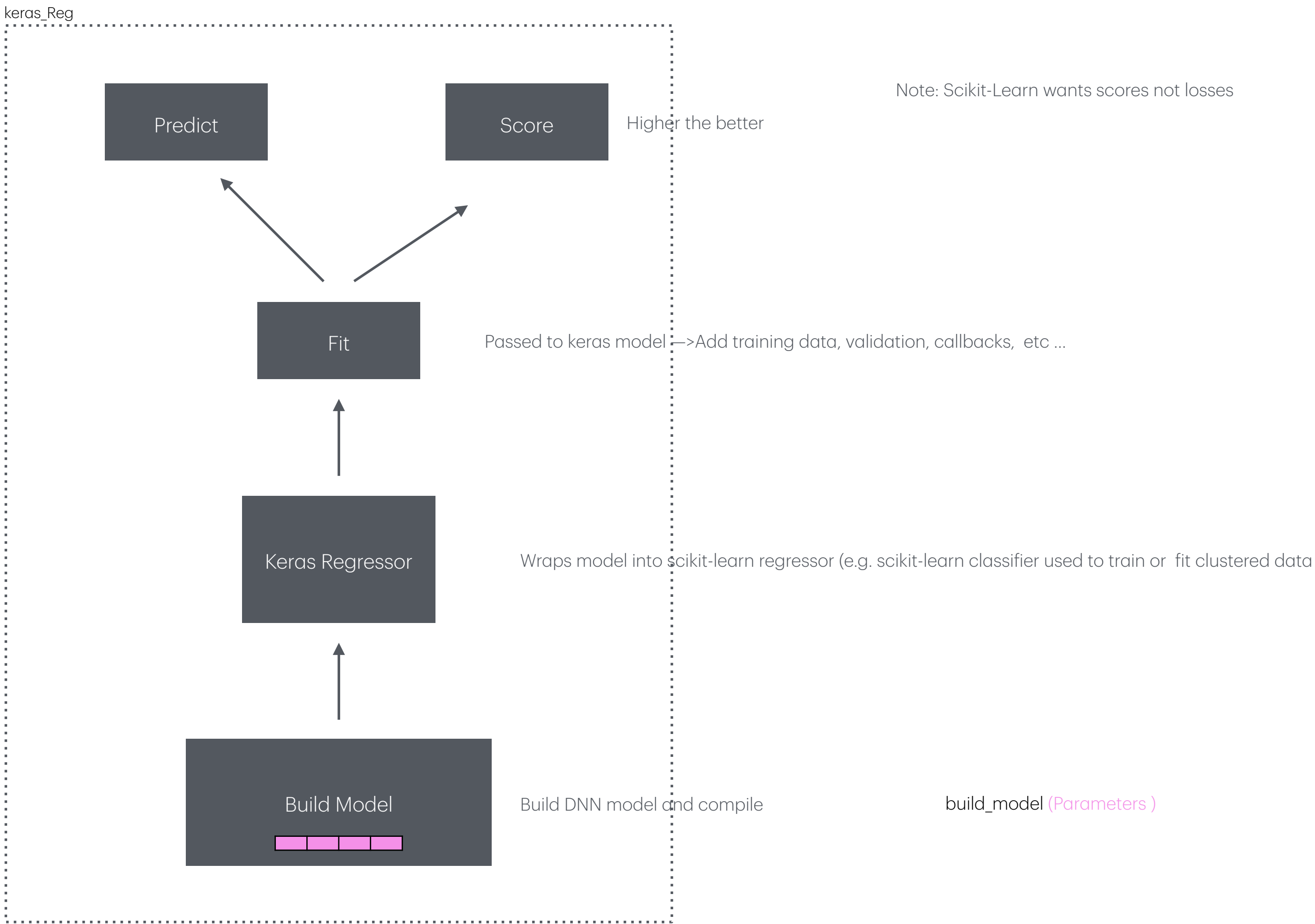
# Saving



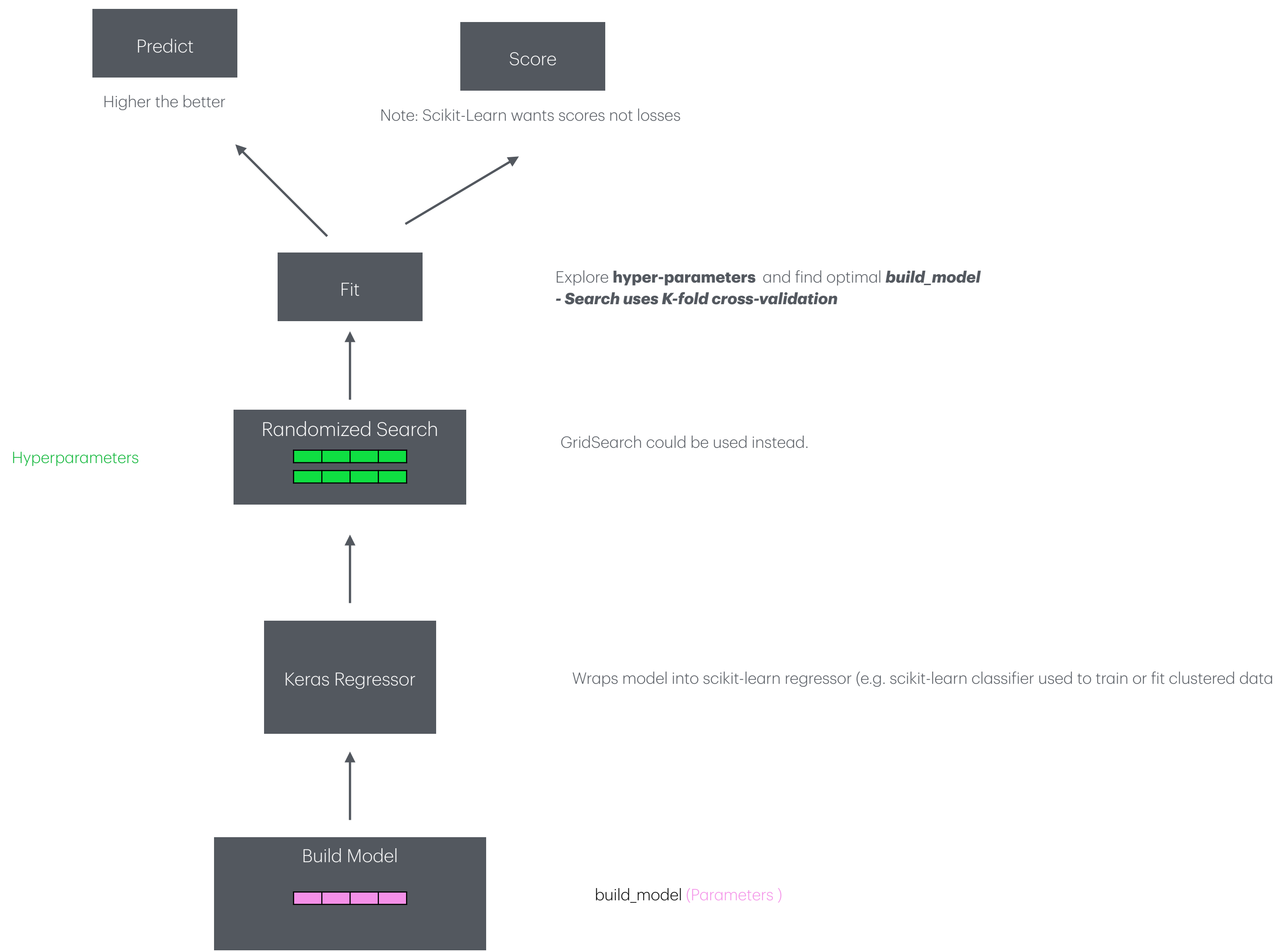
# Tensorboard



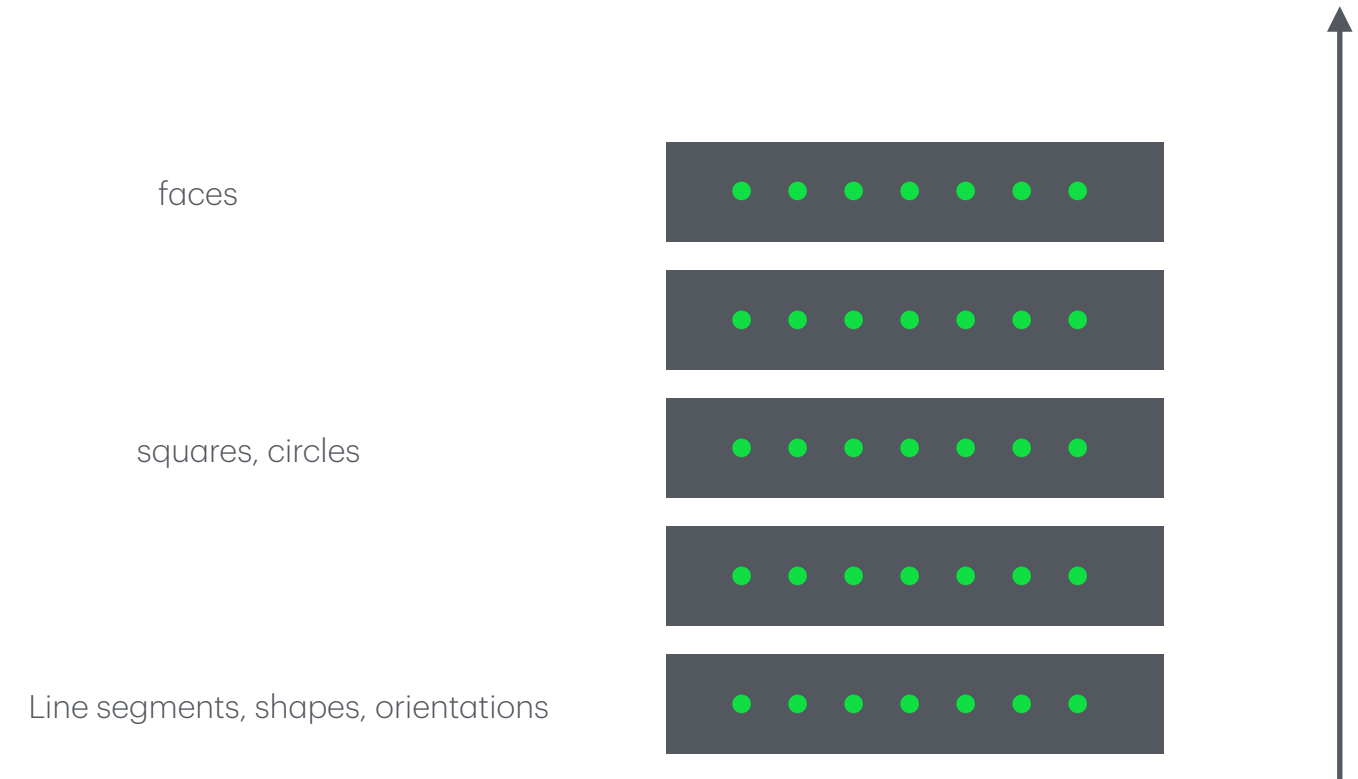
# Fine-Tuning Neural Networks



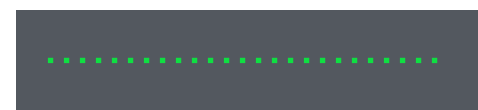
# Fine-Tuning Neural Networks



**Networks learn in hierarchical way**



Complex problems deep networks  
Have higher parameter efficiency.  
Fewer neurons needed per layer



Shallow network can solve many  
problems with enough neurons

↺

▶

▶

Epoch  
000,101

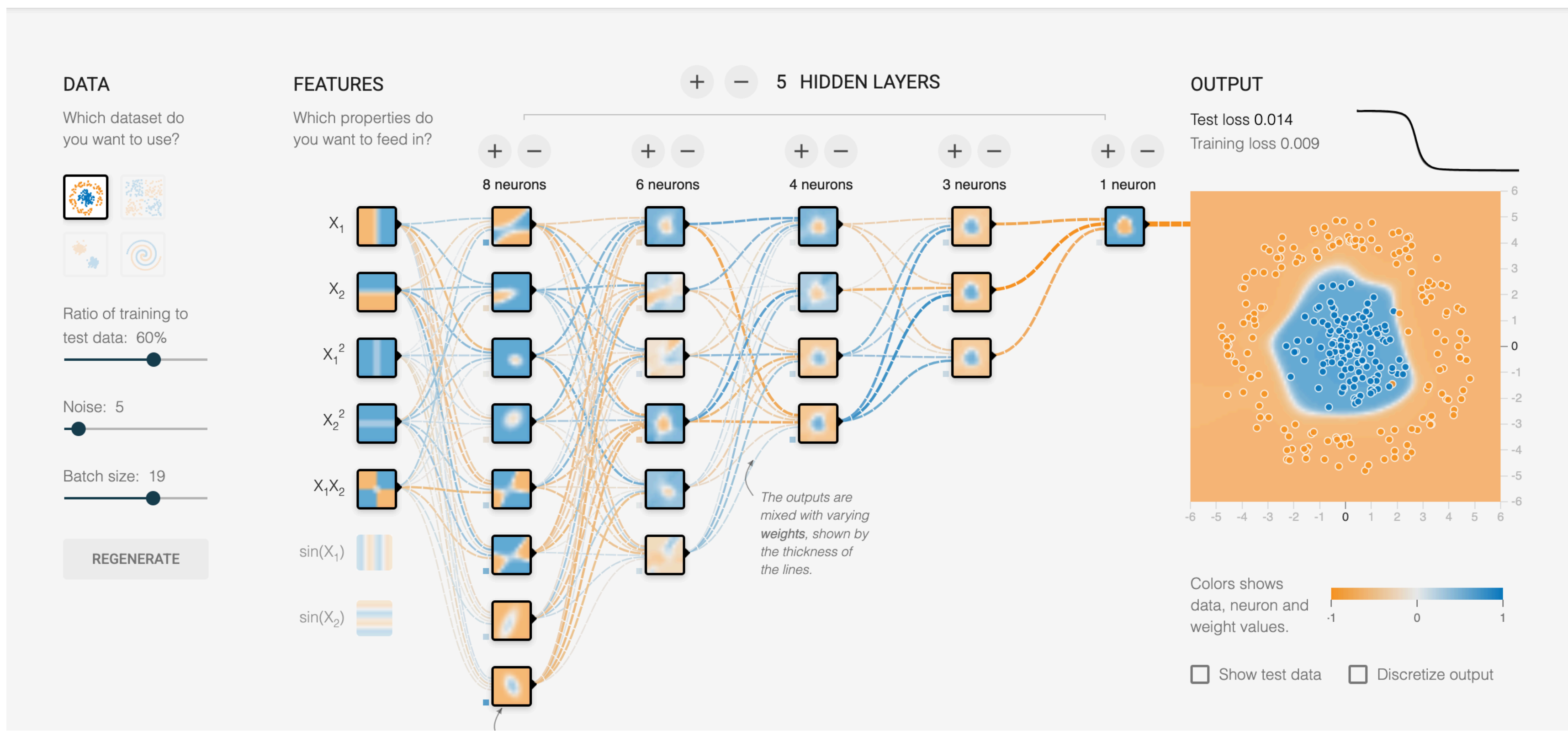
Learning rate  
0.03

Activation  
Tanh

Regularization  
None

Regularization rate  
0

Problem type  
Classification



DATA

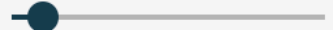
Which dataset do you want to use?



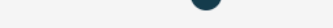
Ratio of training to test data: 60%



Noise: 5



Batch size: 19



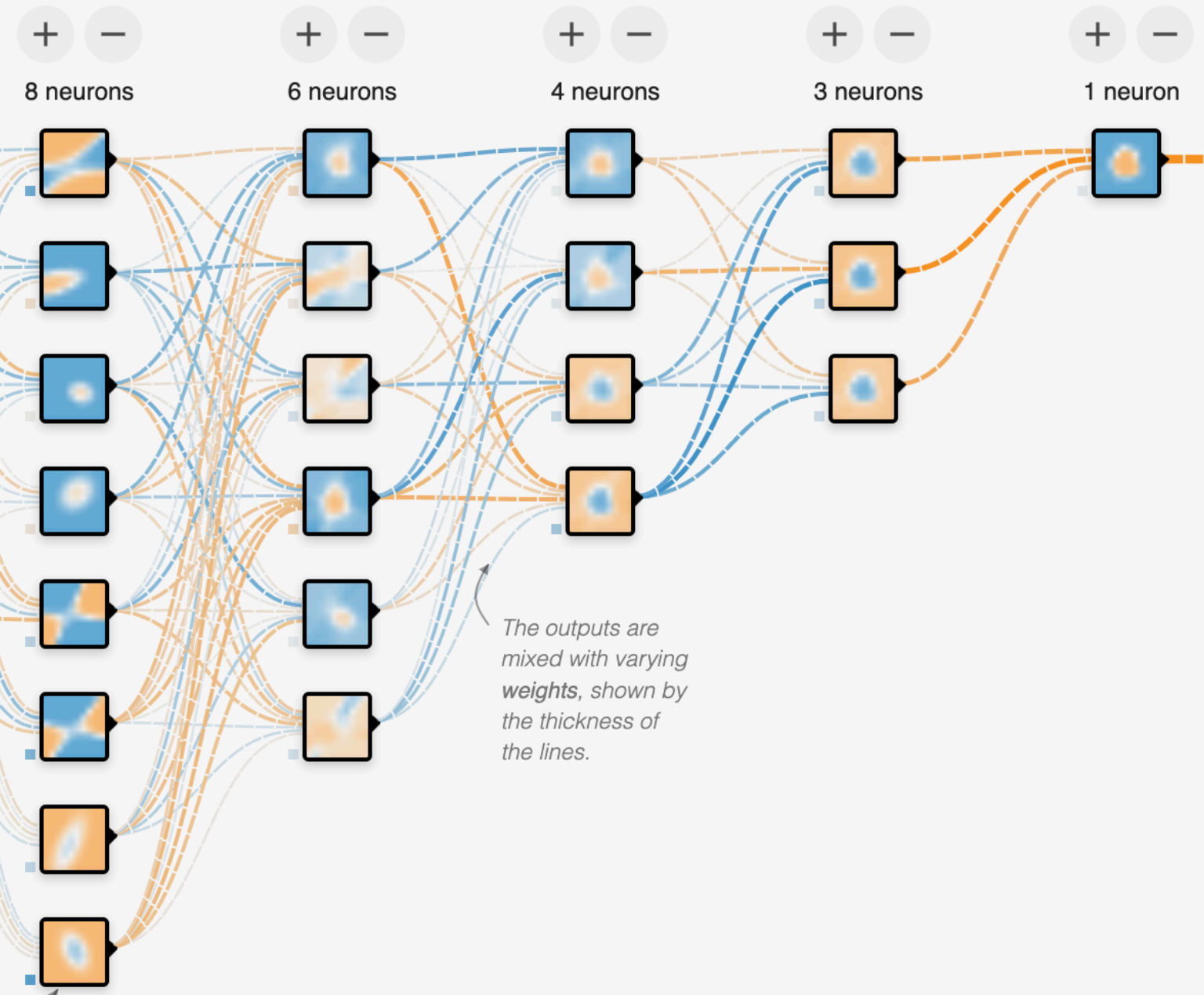
REGENERATE

FEATURES

Which properties do you want to feed in?

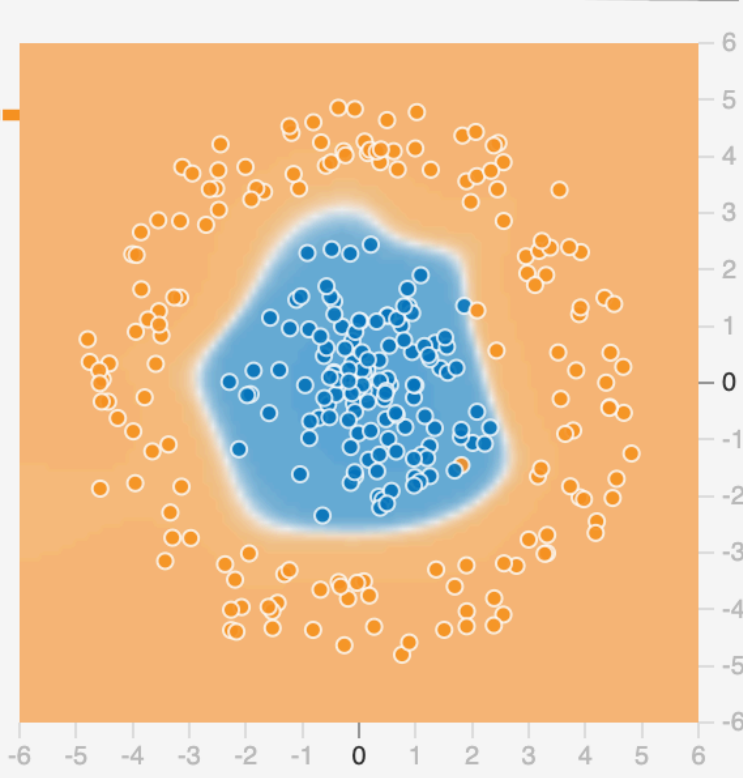
- $X_1$
- $X_2$
- $X_1^2$
- $X_2^2$
- $X_1X_2$
- $\sin(X_1)$
- $\sin(X_2)$

5 HIDDEN LAYERS



OUTPUT

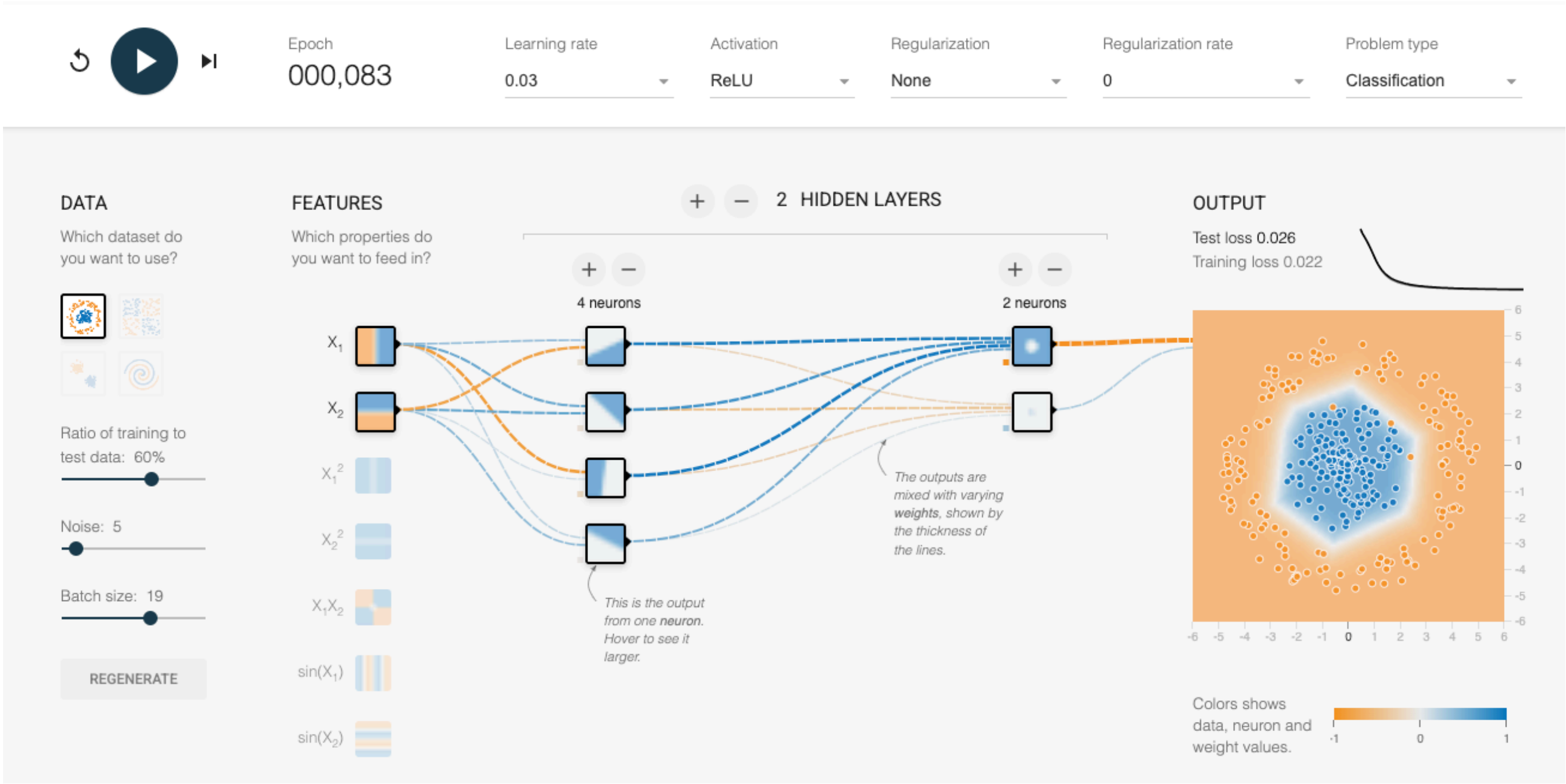
Test loss 0.014  
Training loss 0.009



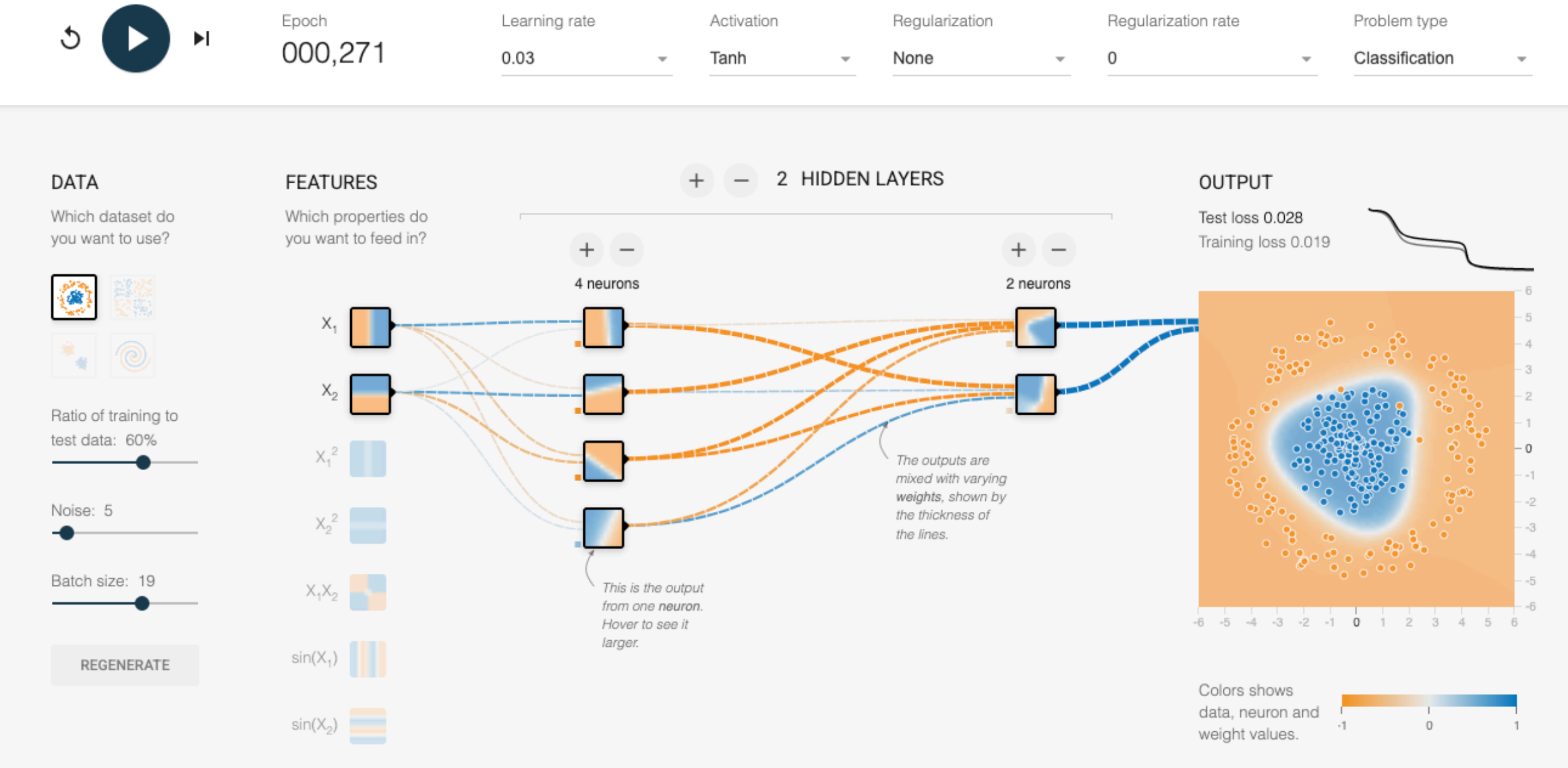
Colors shows data, neuron and weight values.

☐ Show test data   ☐ Discretize output





***ReLU faster,  
notice linear  
boundaries***

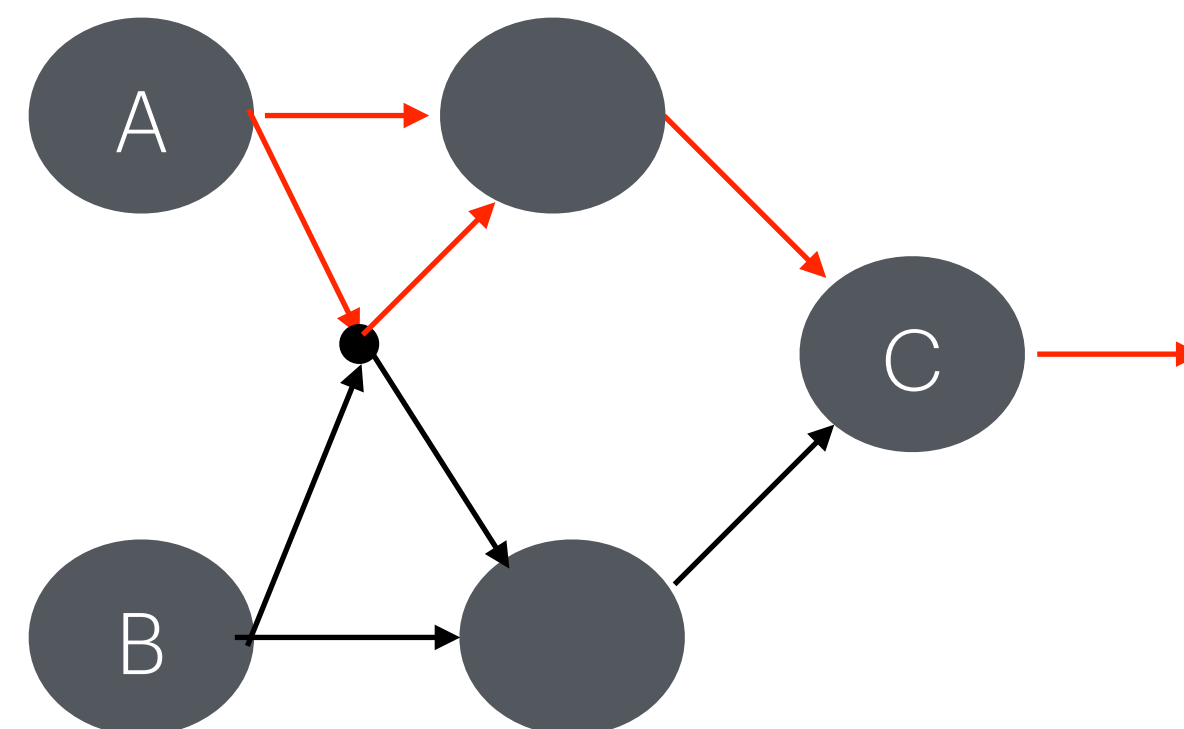
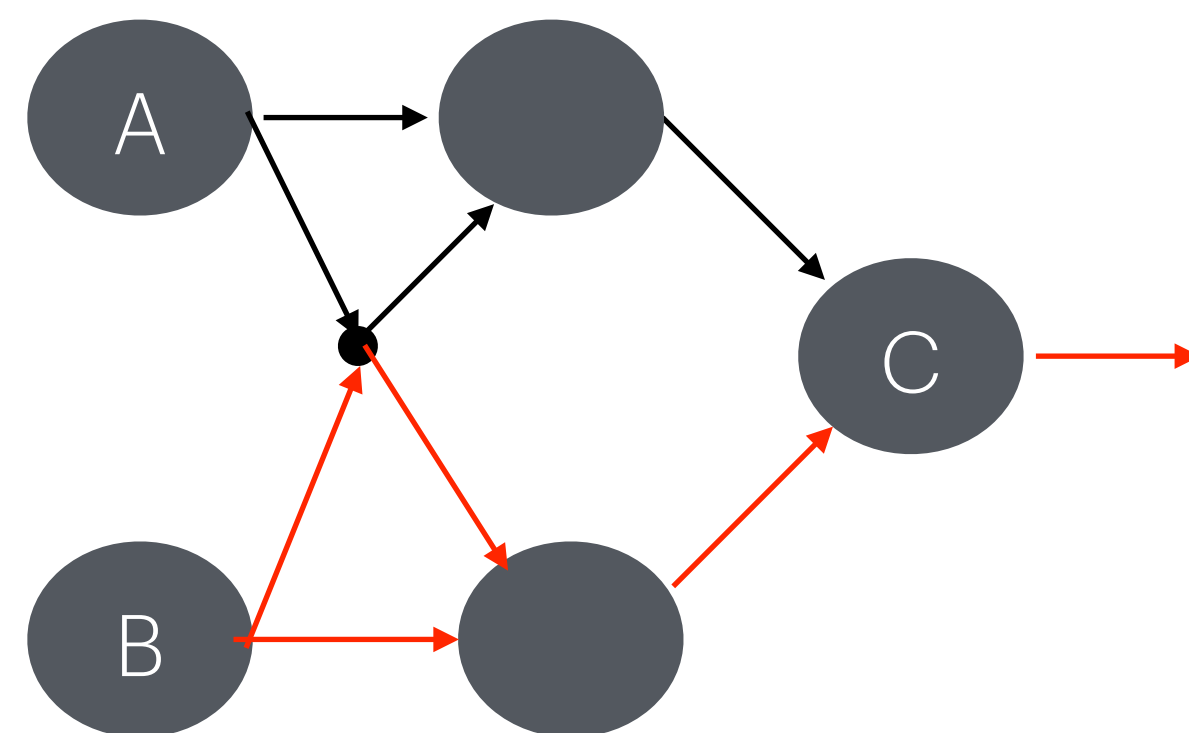
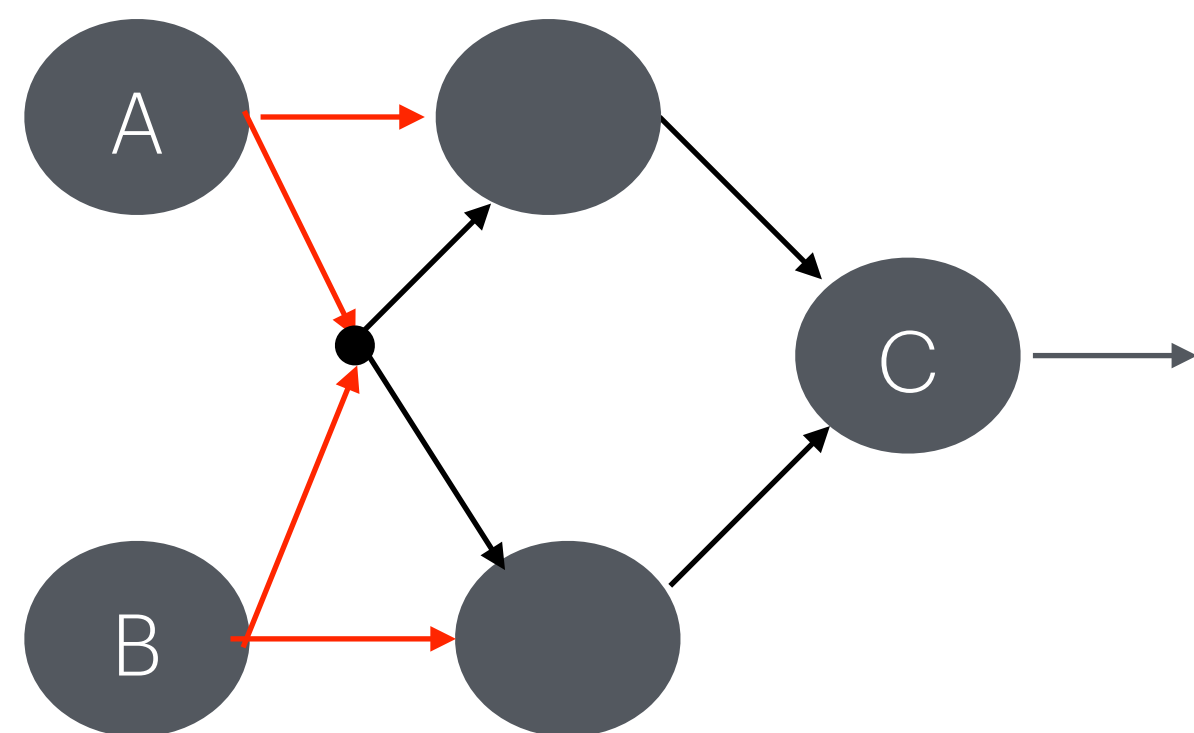


***TANU takes time  
to converge on a  
solution***

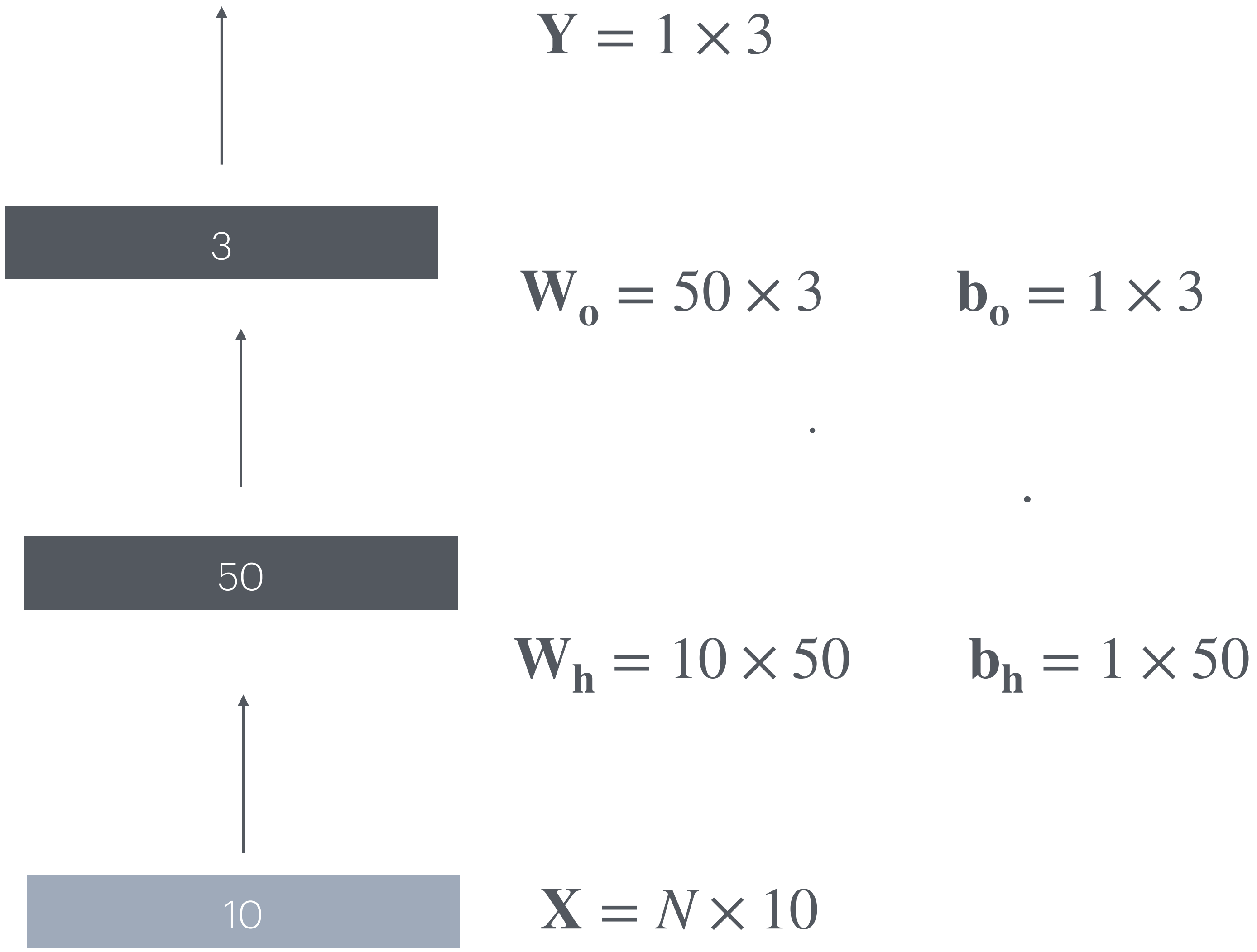
# Exercise 2

Draw an ANN using the original artificial neurons that computes  $A \oplus B$

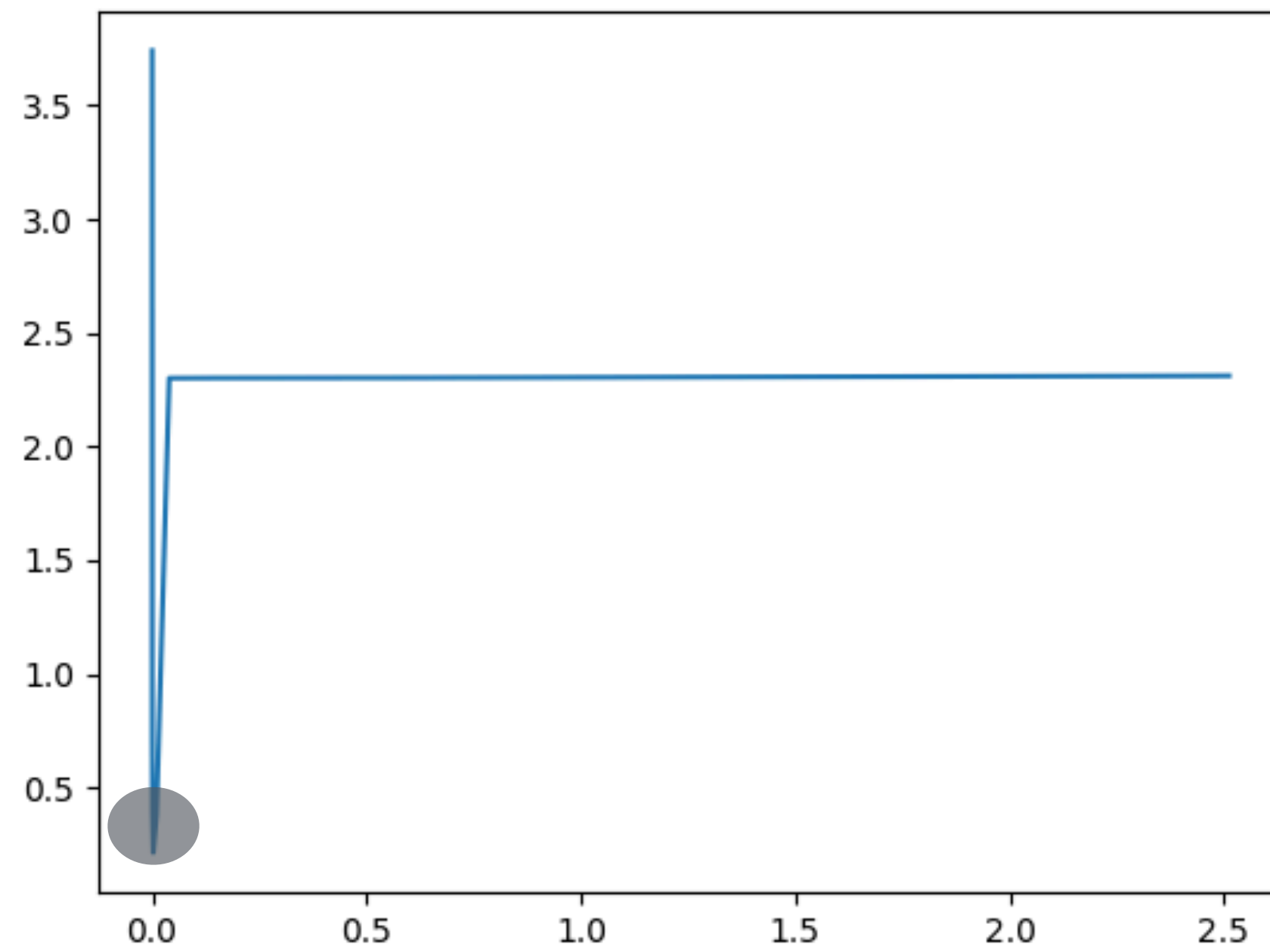
$$(A \cup \neg B) \cup (\neg A \cup B)$$



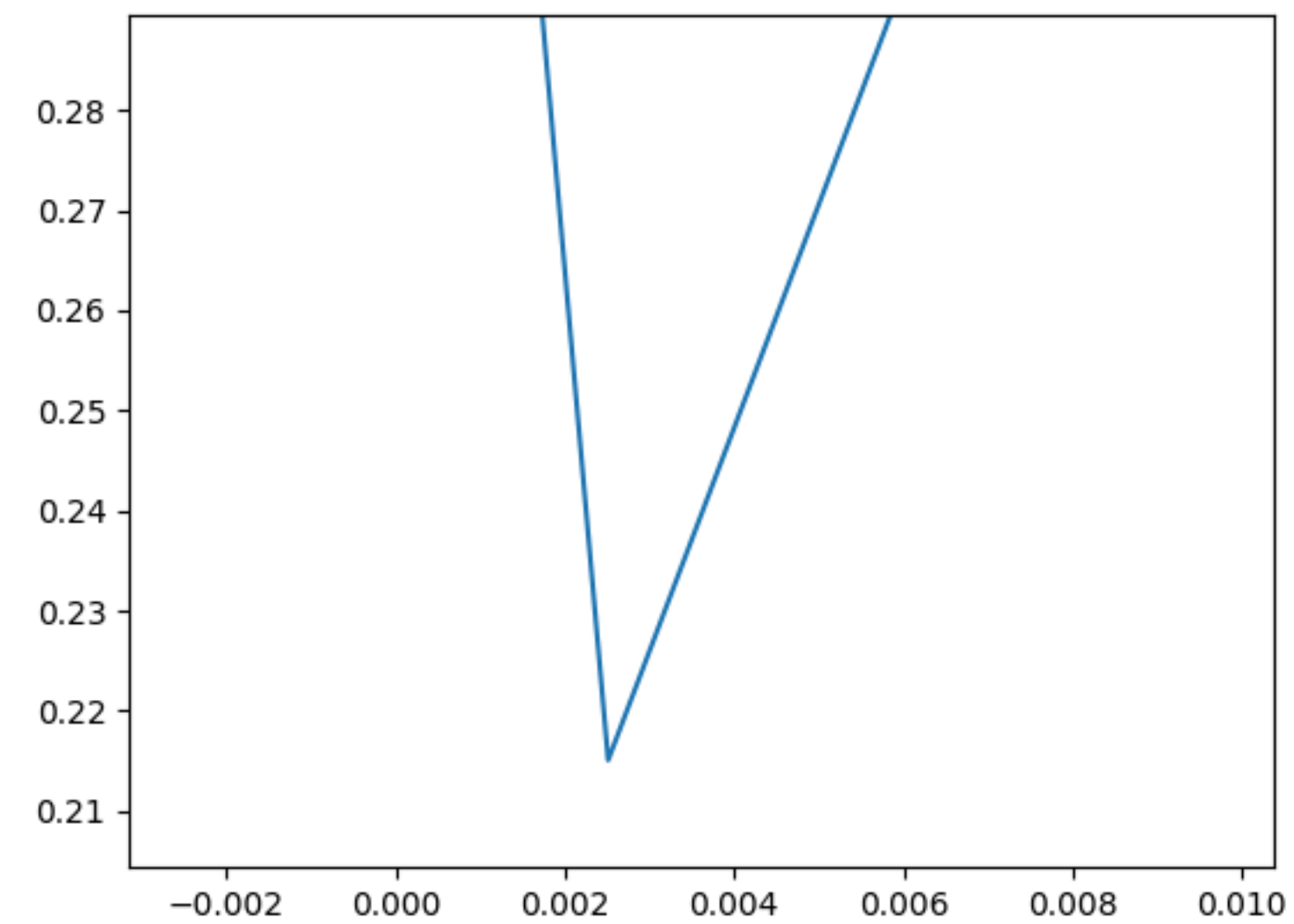
Exercise 3



## Find Learning Rate



Gradually increase learning rate



Region where loss decreases and immediately increases is an optimal learning rate