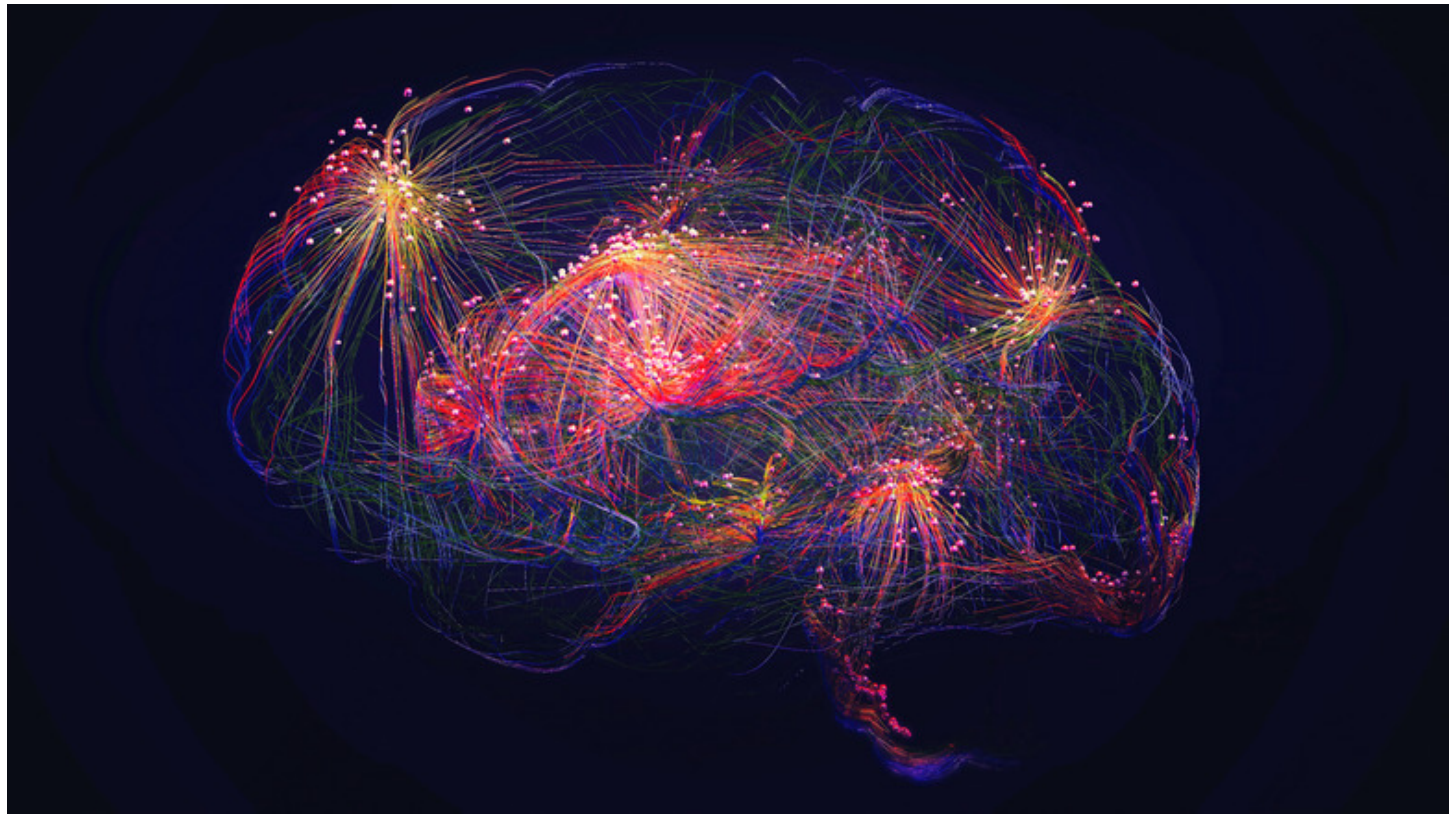
Artificial Neural Networks



Artificial Neural Networks inspired by brain research

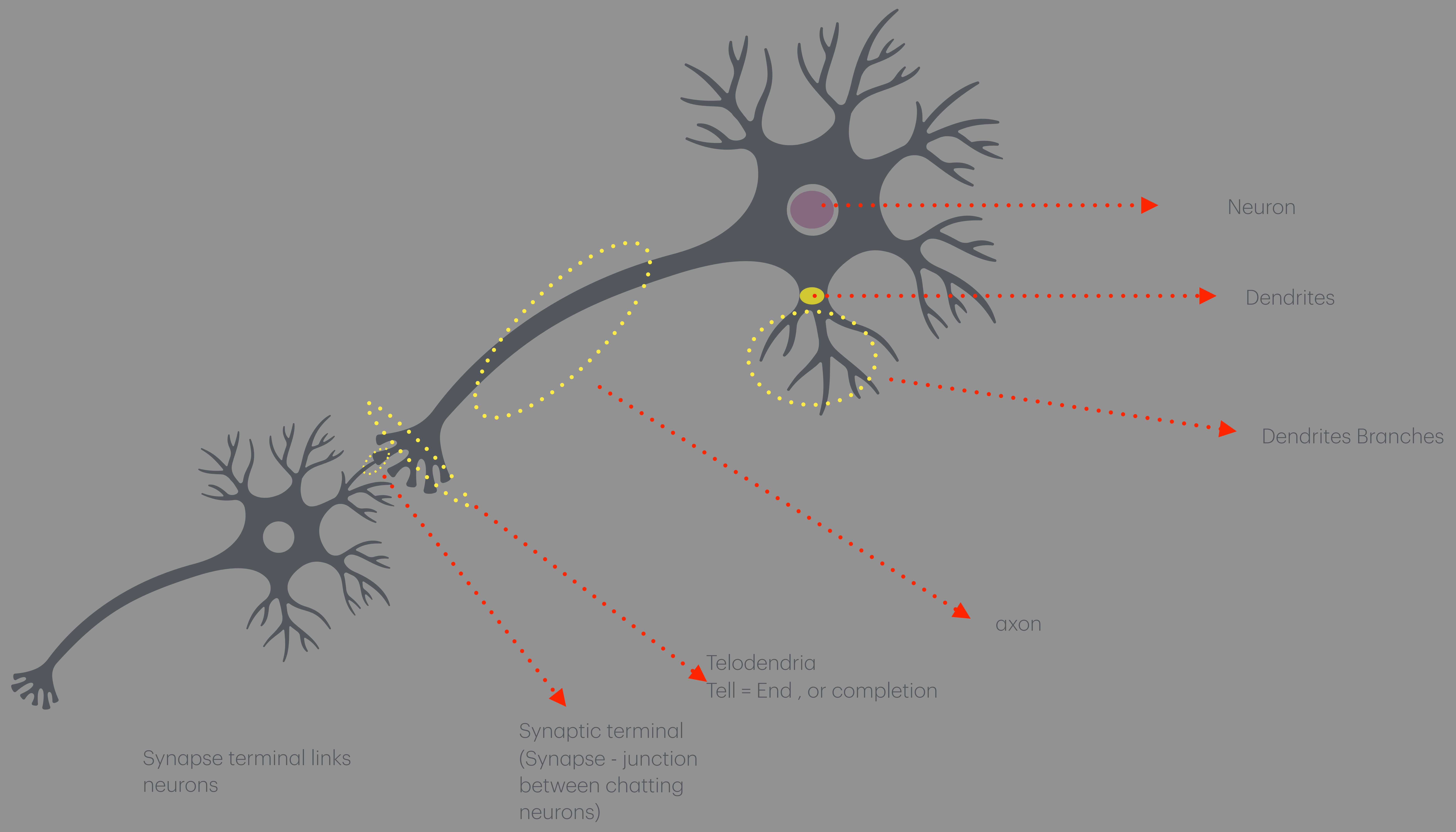
Speech Recognition

Google Images (Classification)

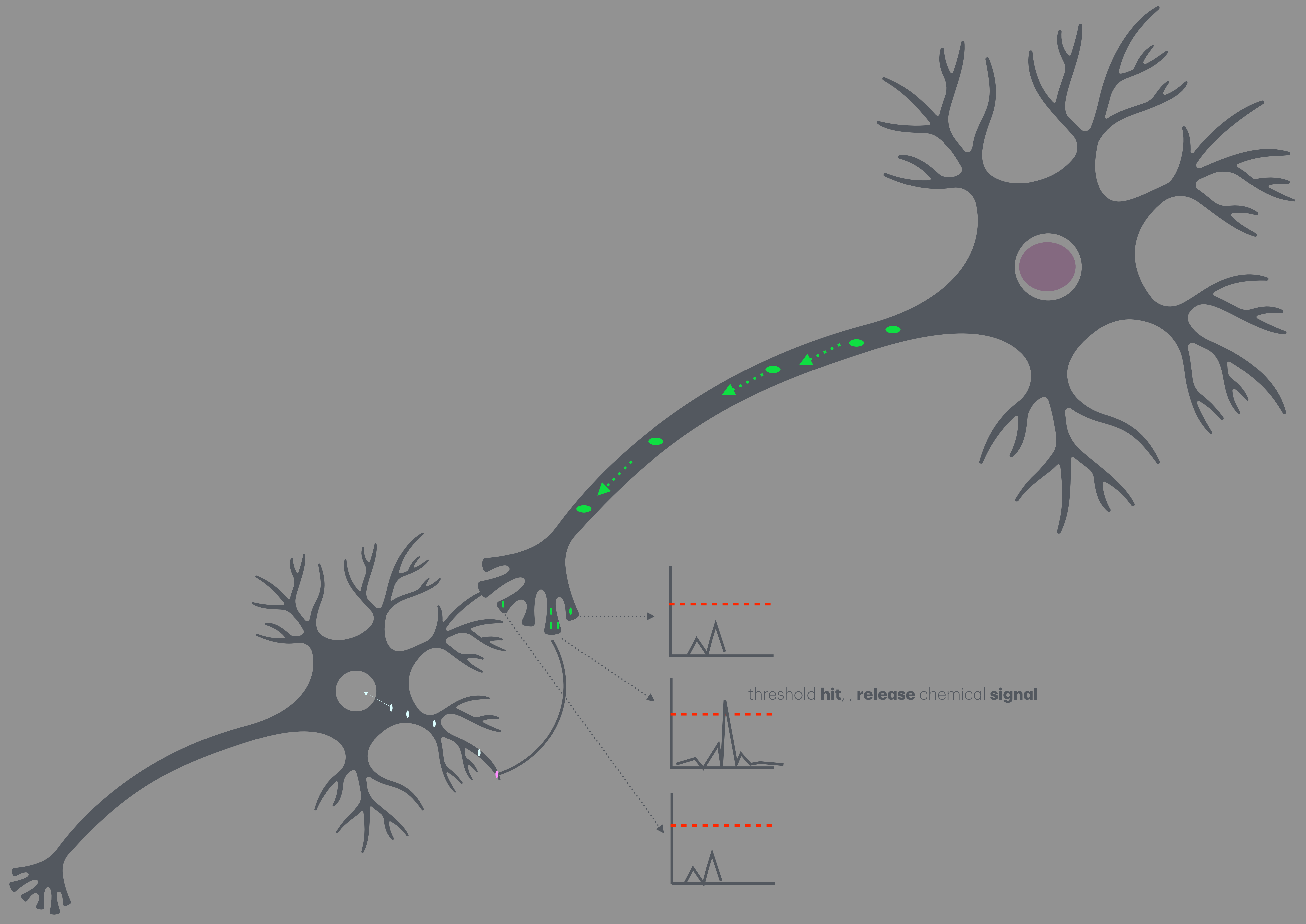
Go (DeepMind)

Recommendation Systems (Youtube)

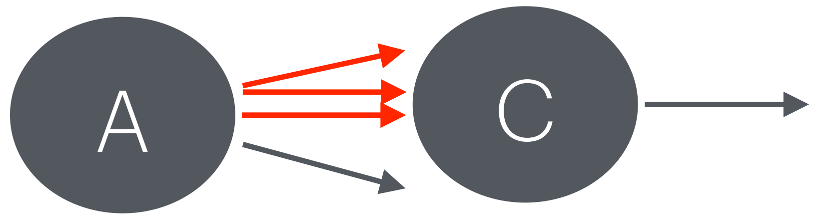
Biological Neuron



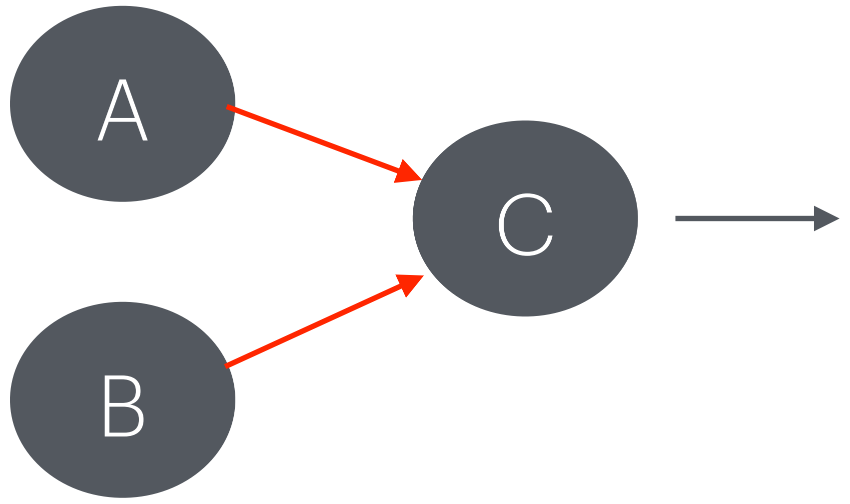
Biological Neuron



Logical Computation: Artificial Neuron



Neuron C = Neuron A
Equal Electrical Amplitude
Note: Off neuron cancels a single active input

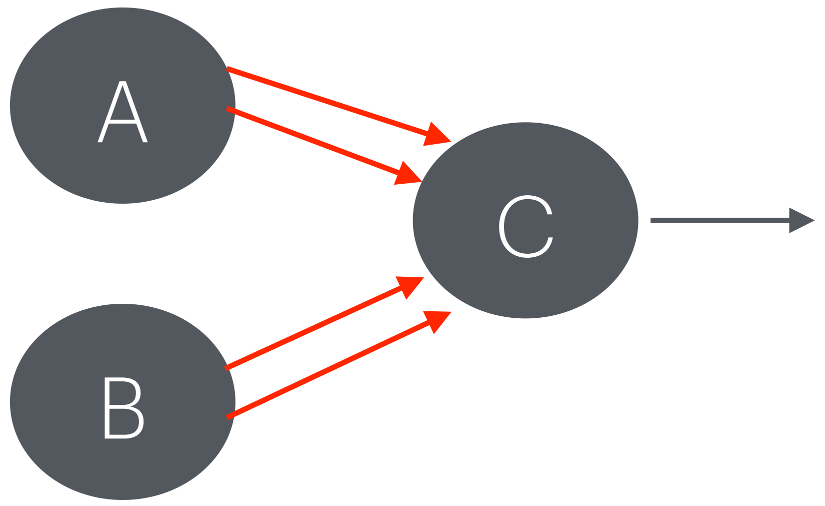


Neuron A **AND** Neuron B = Neuron C

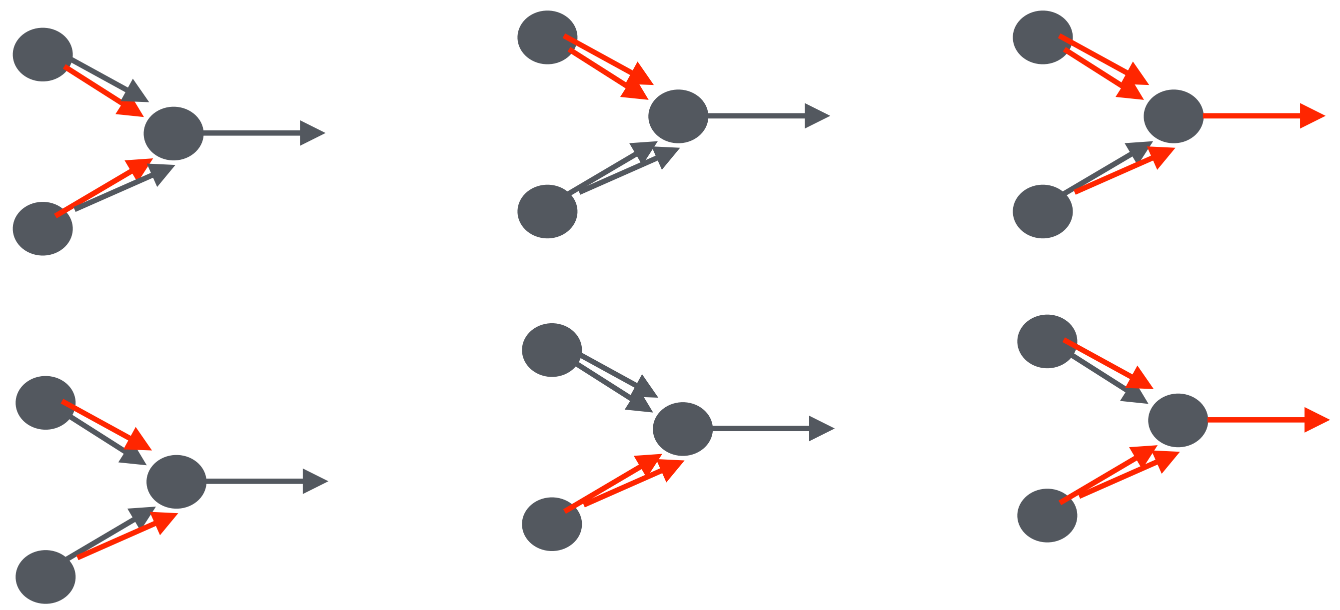
Note: Neuron activated when at least two inputs are active

Note: consumer neuron C sums all the inputs

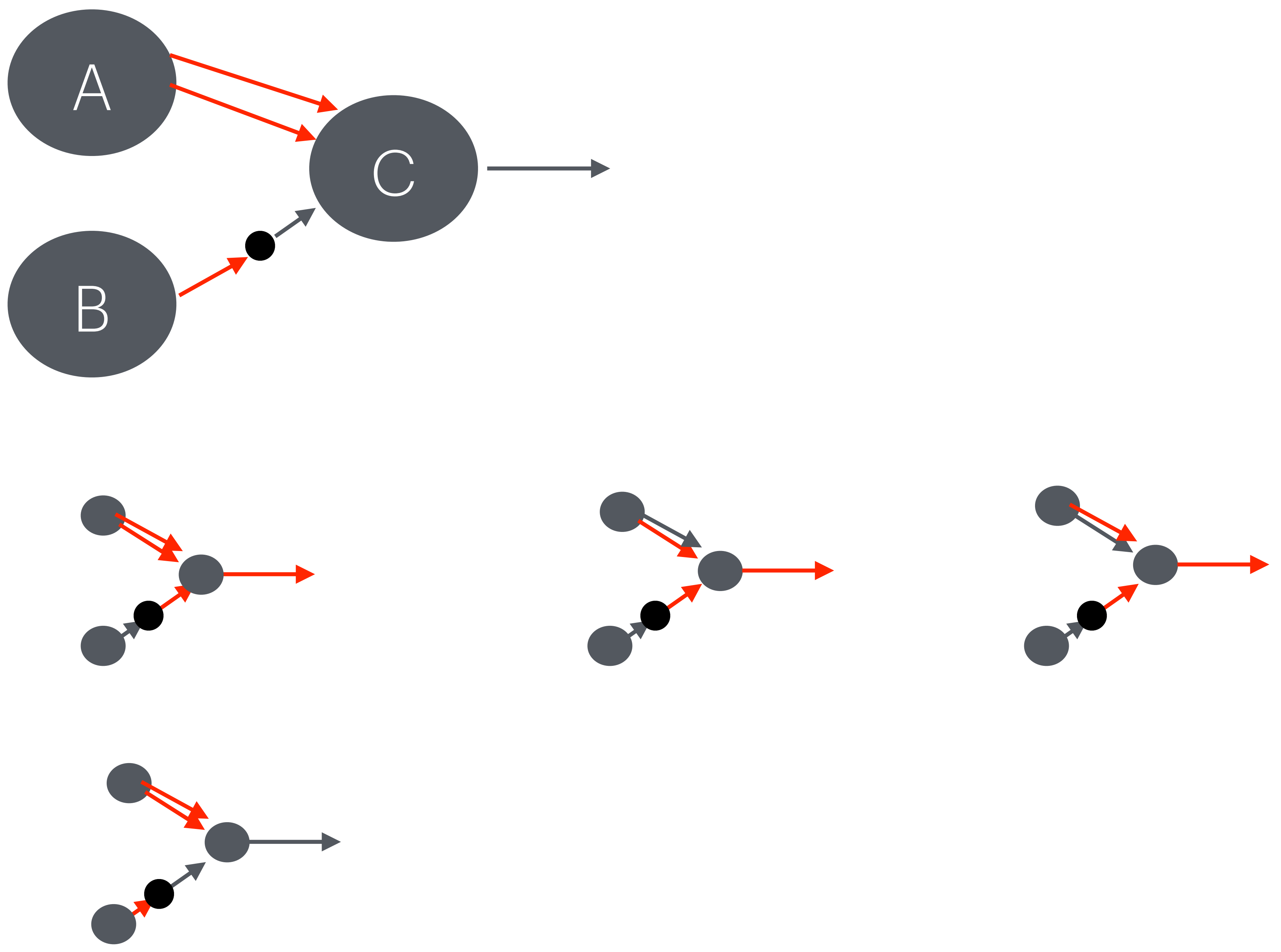
Logical Computation: Artificial Neuron



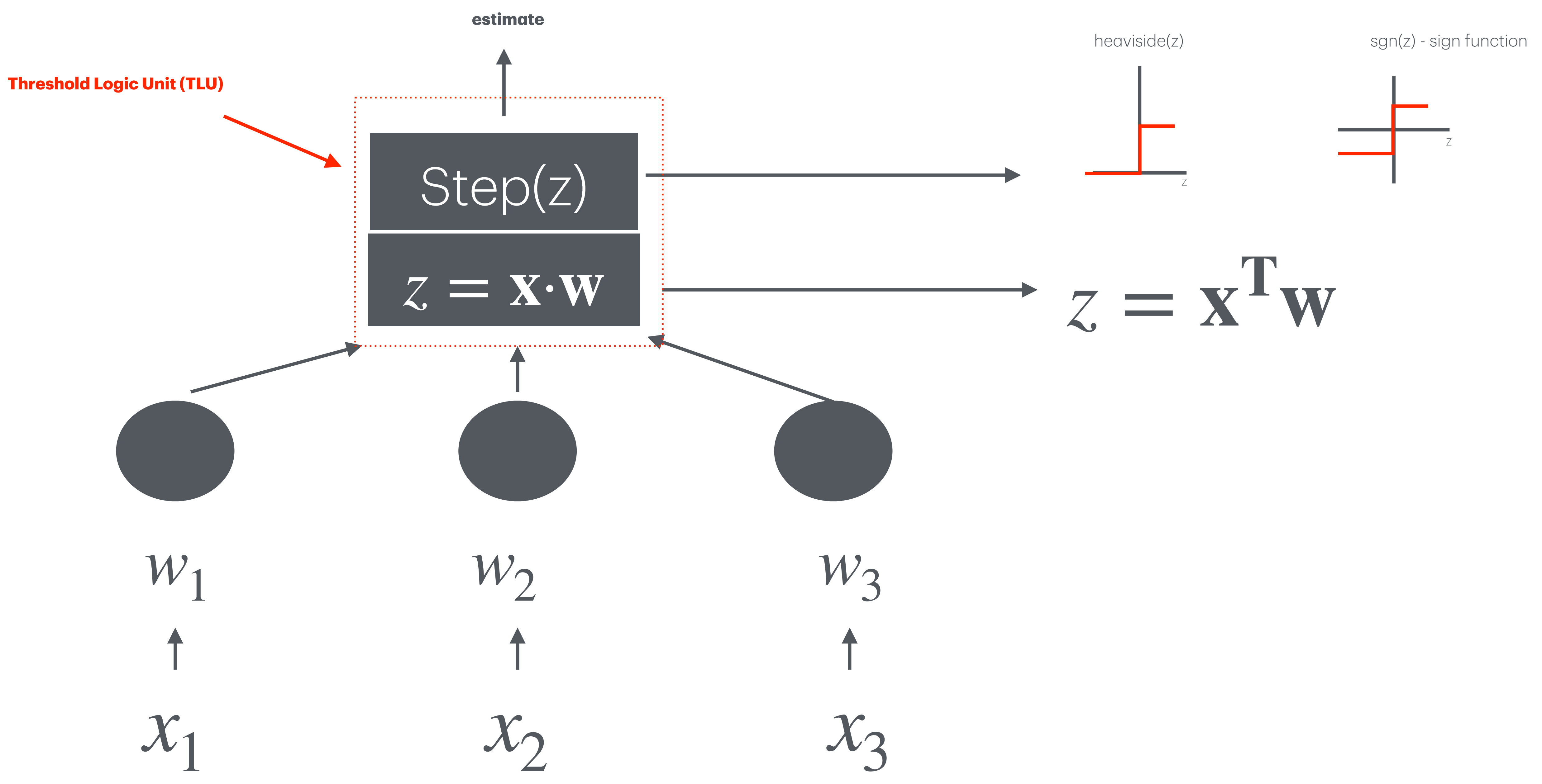
Neuron A **OR** Neuron B = Neuron C



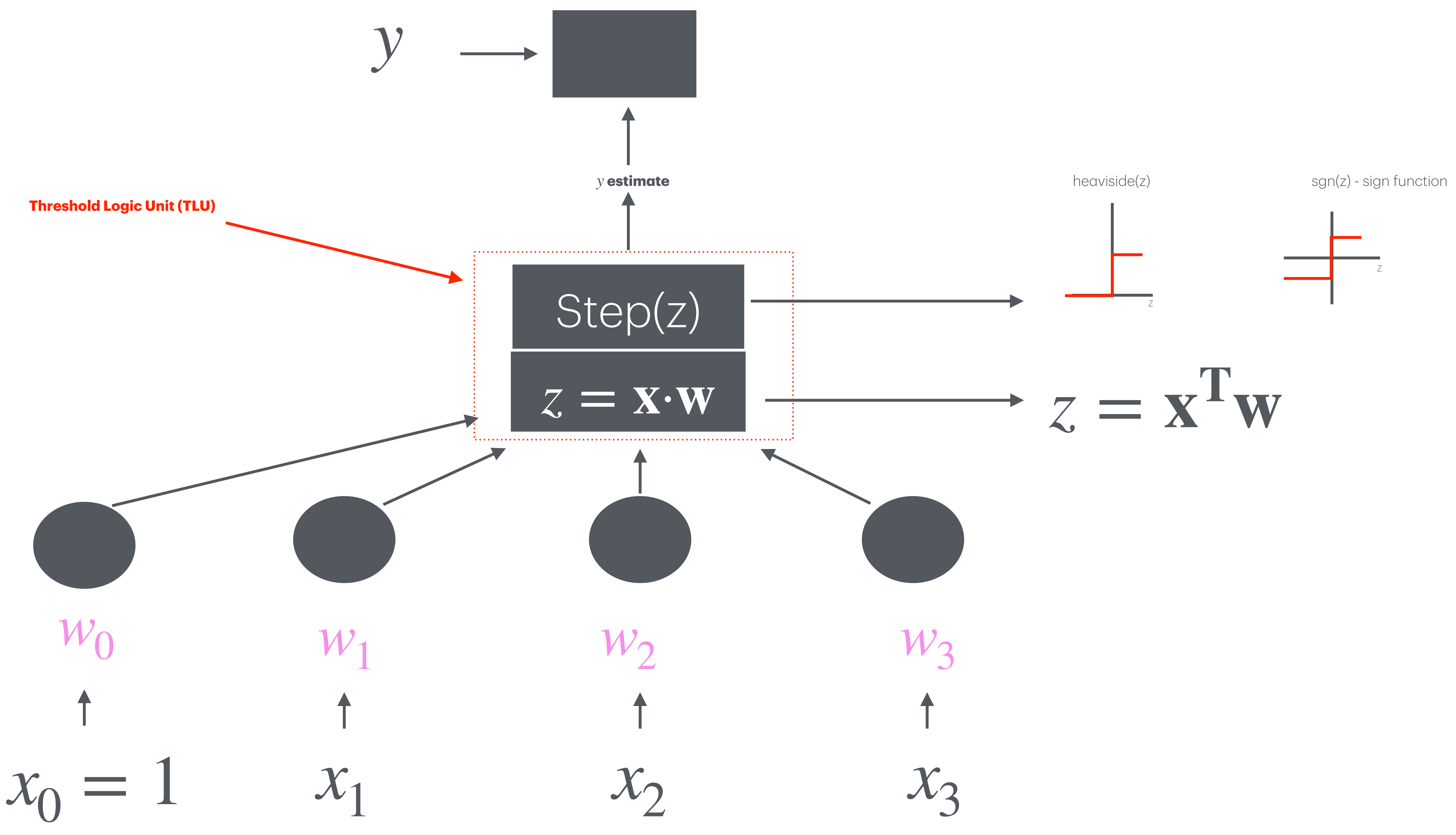
Logical Computation: Artificial Neuron

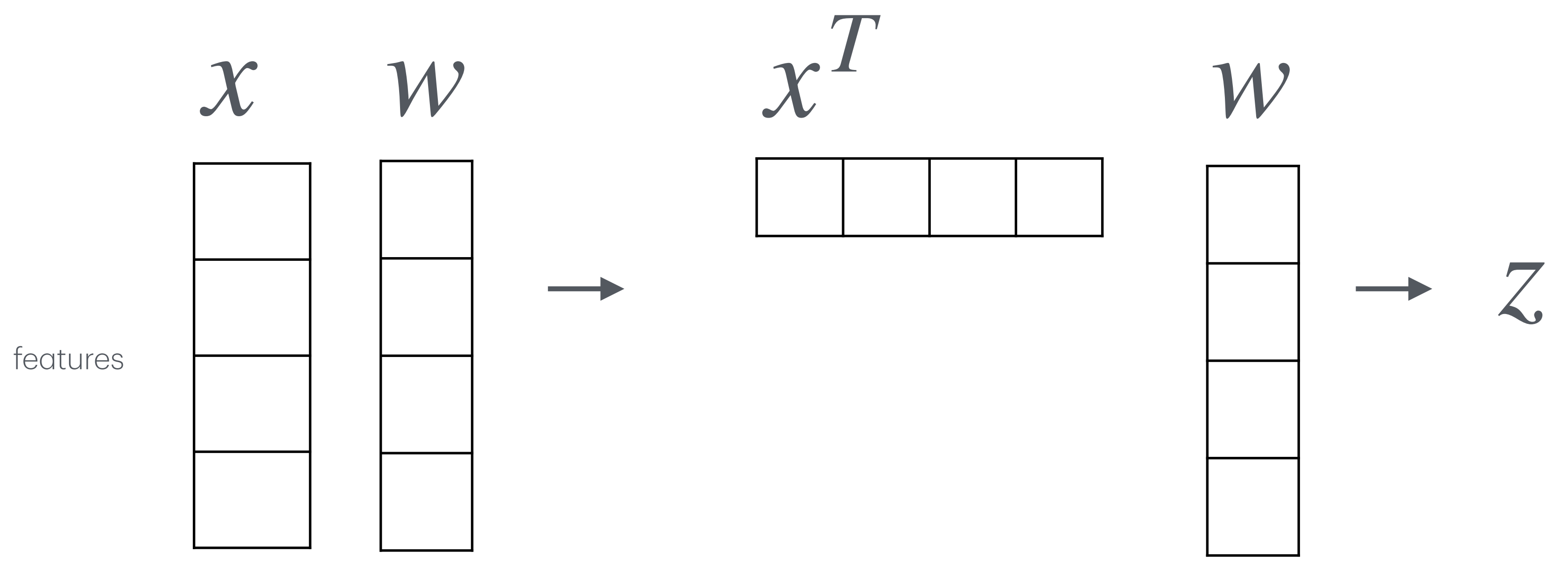


Logical Computation: Perception



Logical Computation: Perception

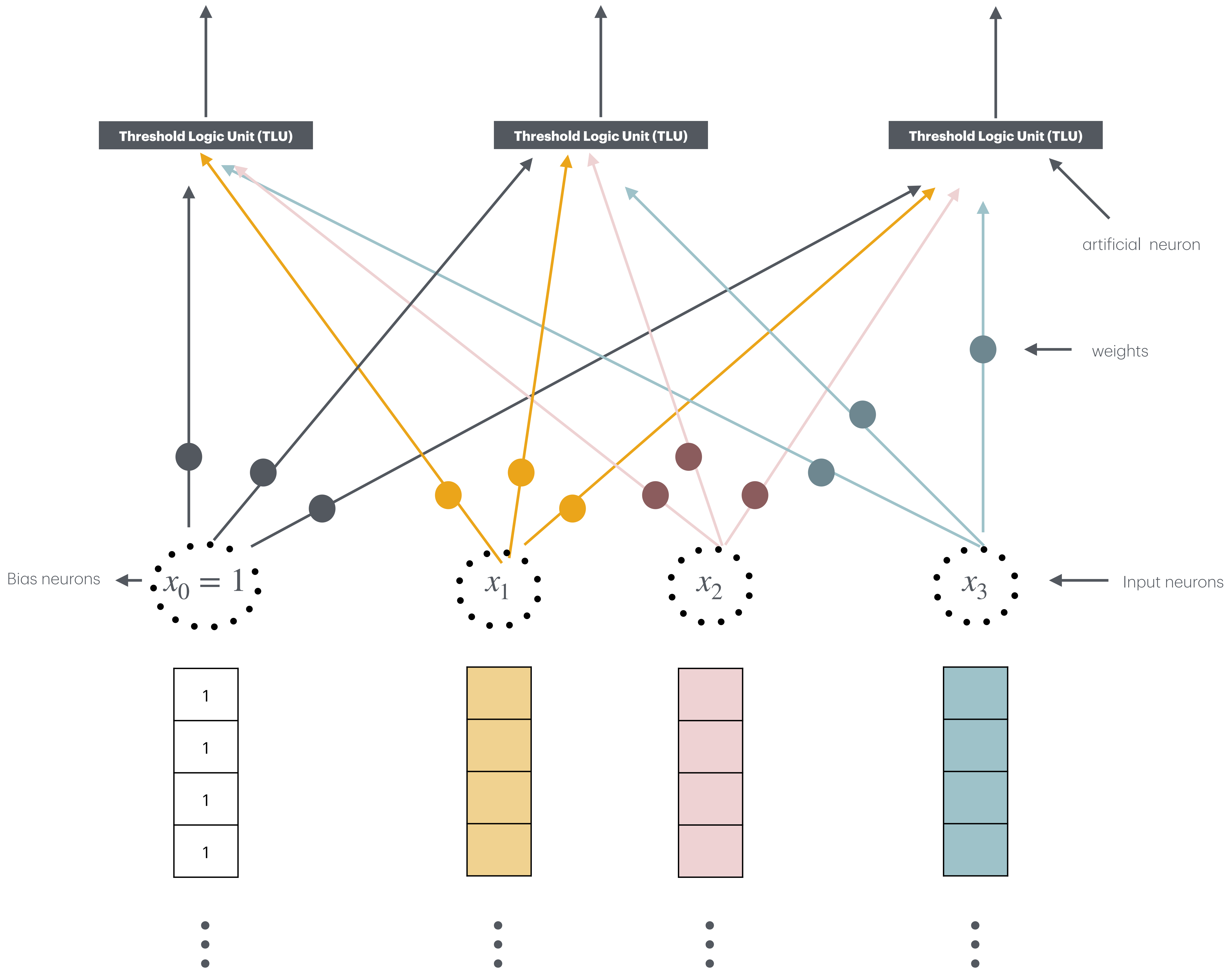


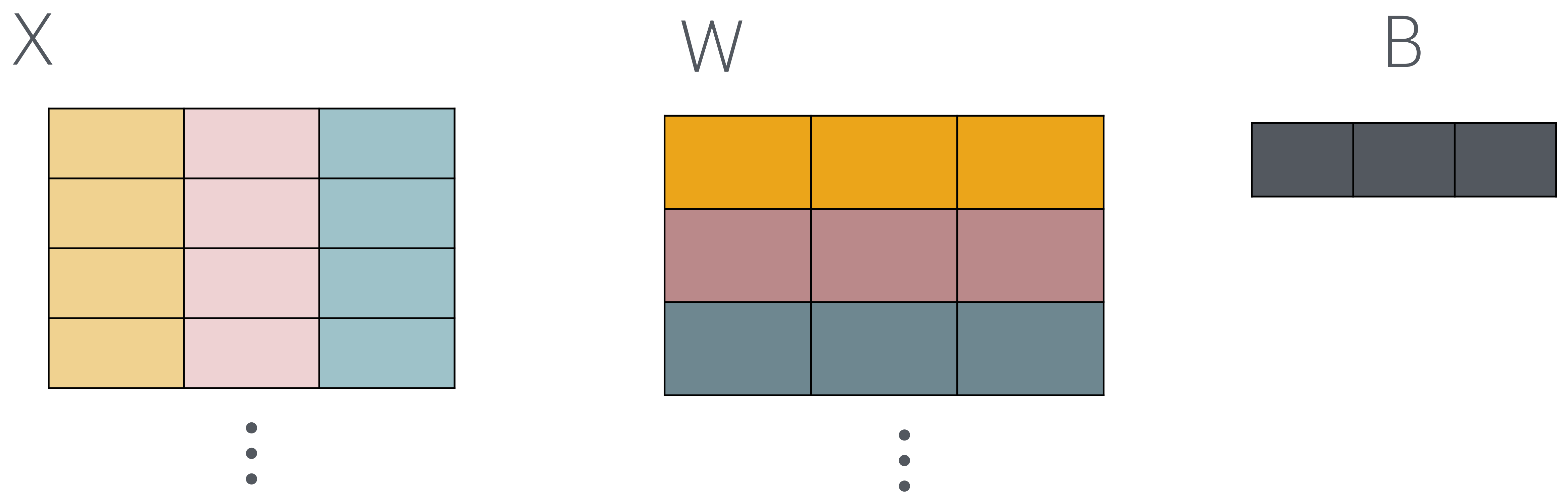


Find right values for weights that lower MSE with expectations

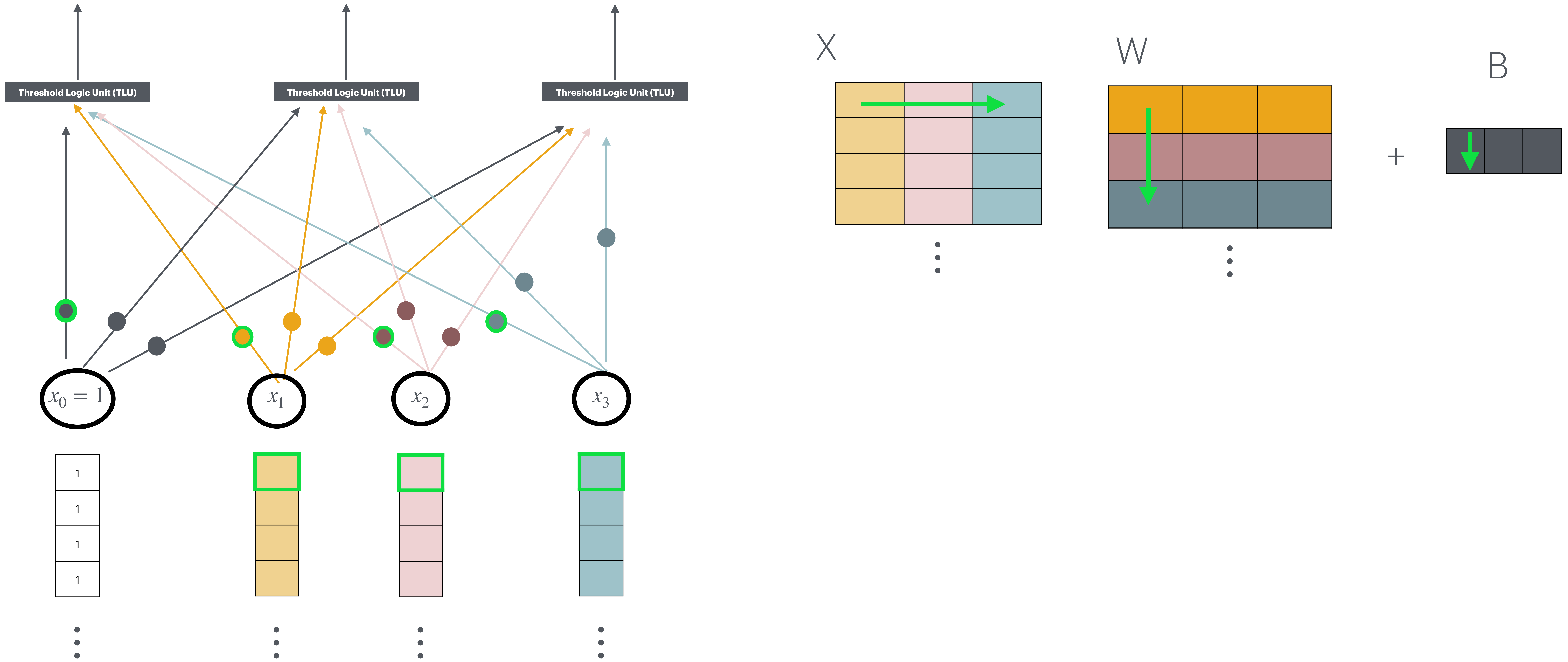
Used for simple linear classification (1D, 2D features)

Logical Computation: Multi-Perception

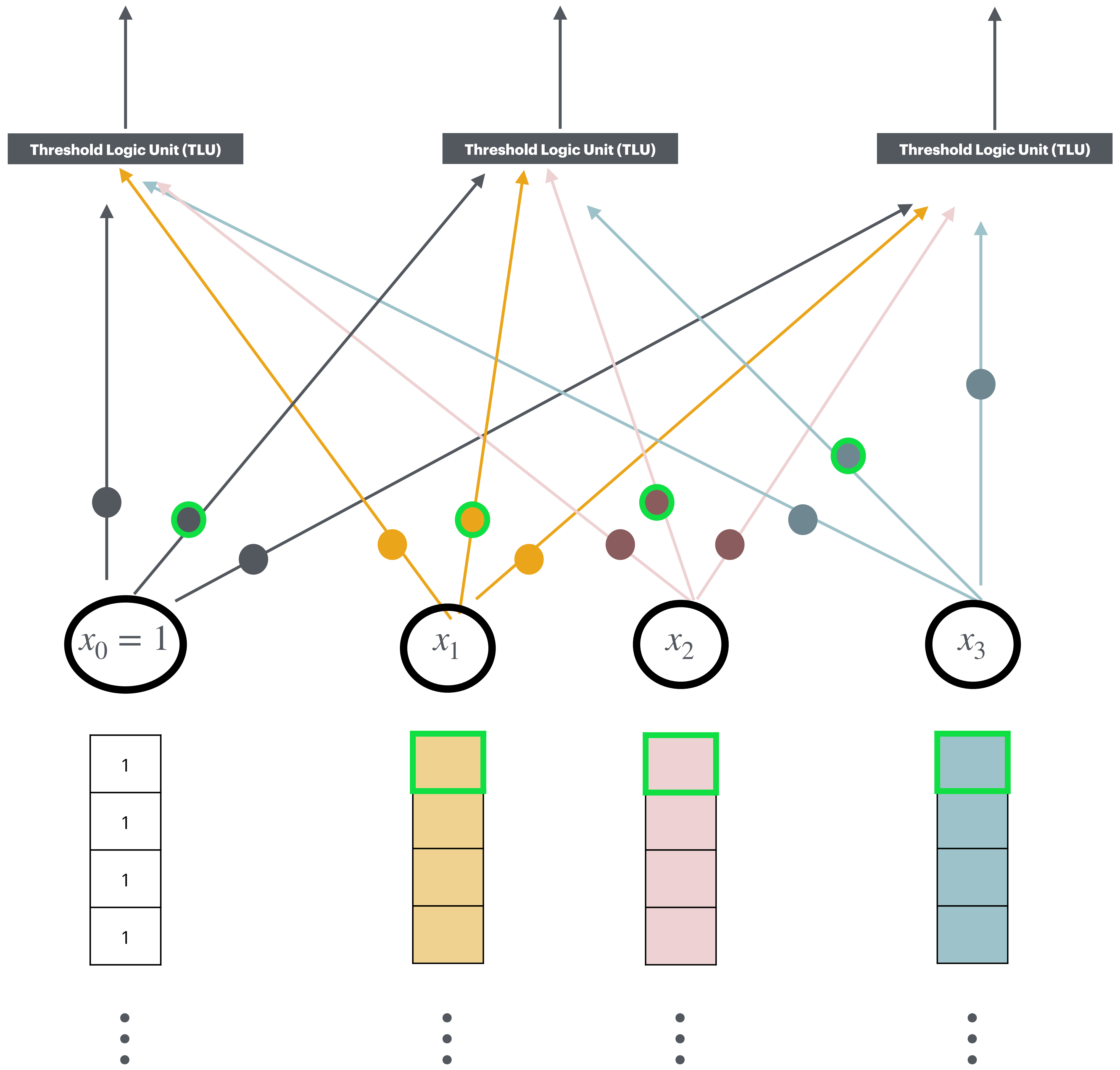


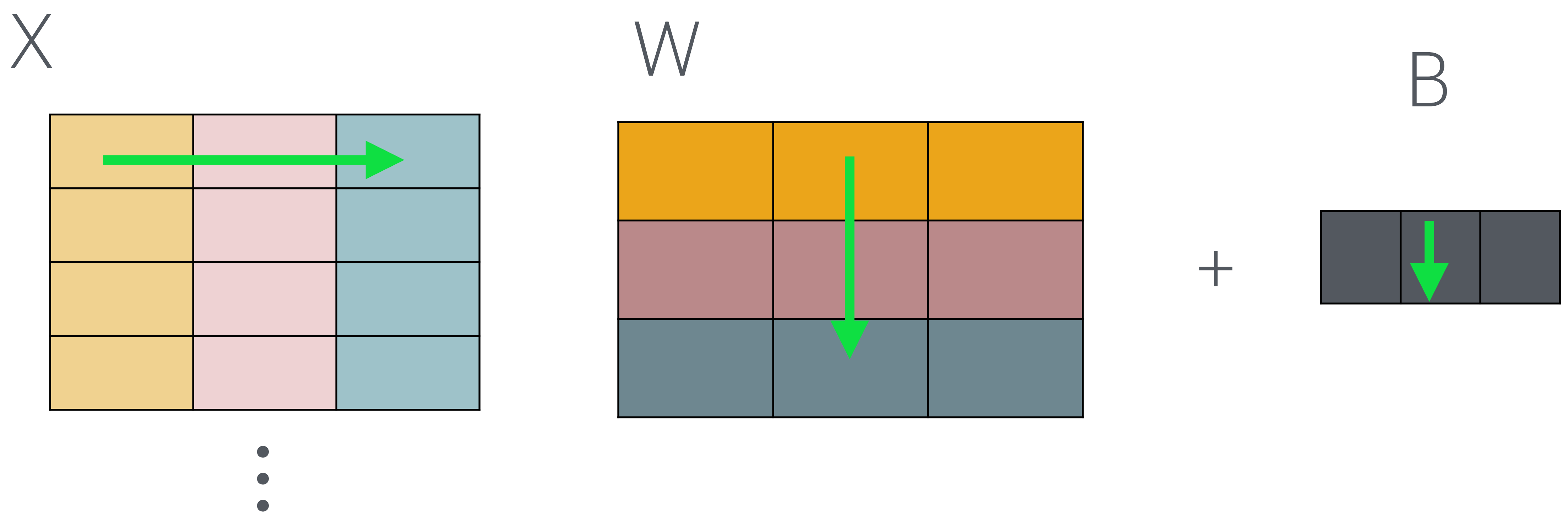


Logical Computation: Perception

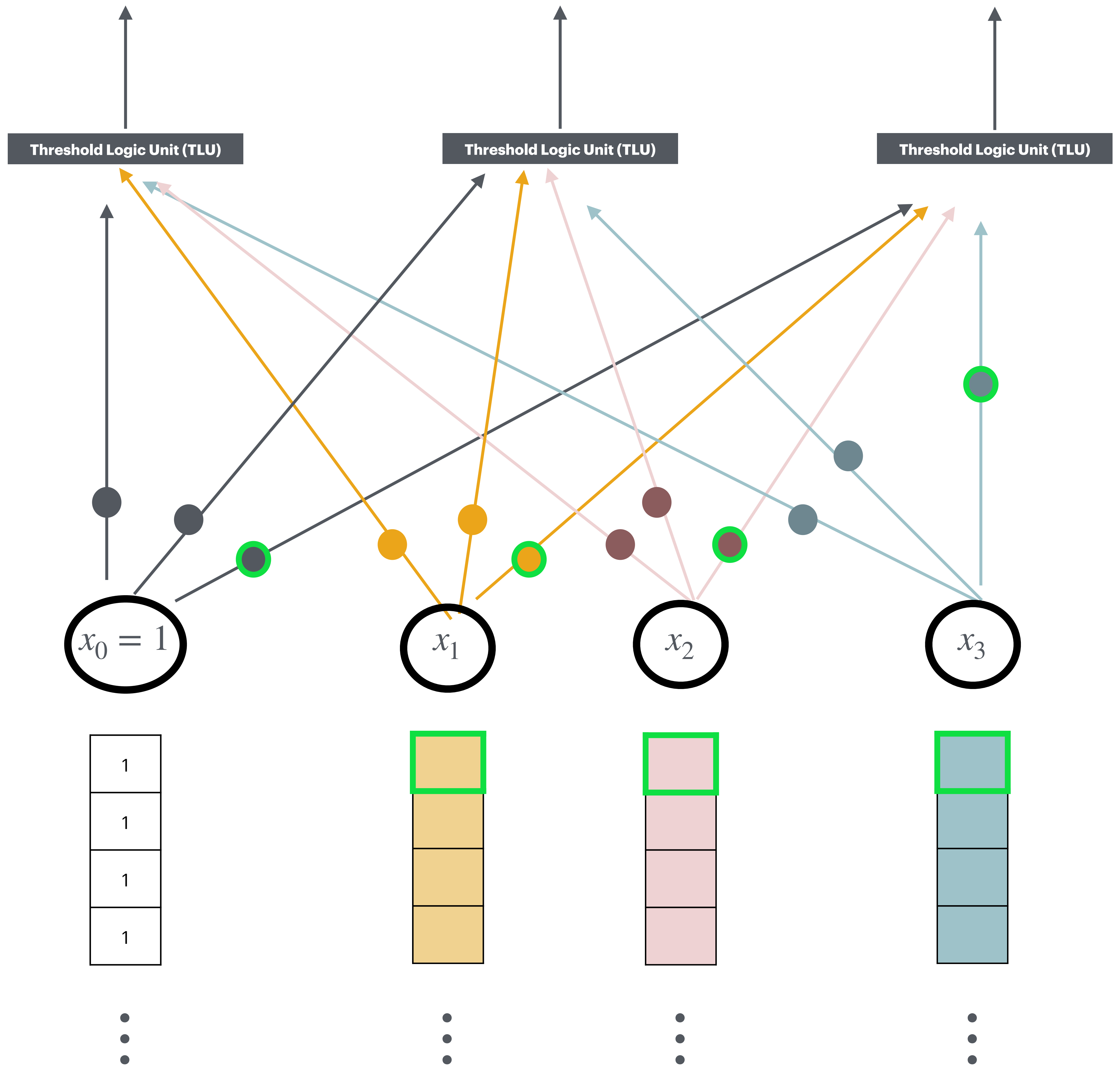


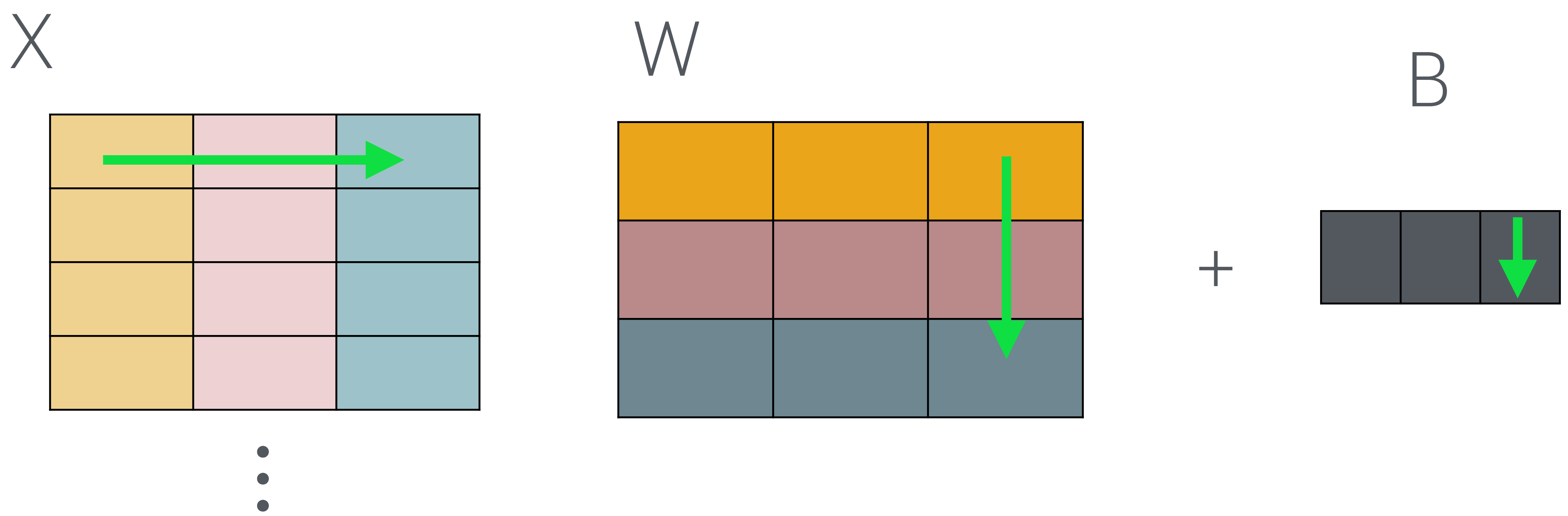
Logical Computation: Perception



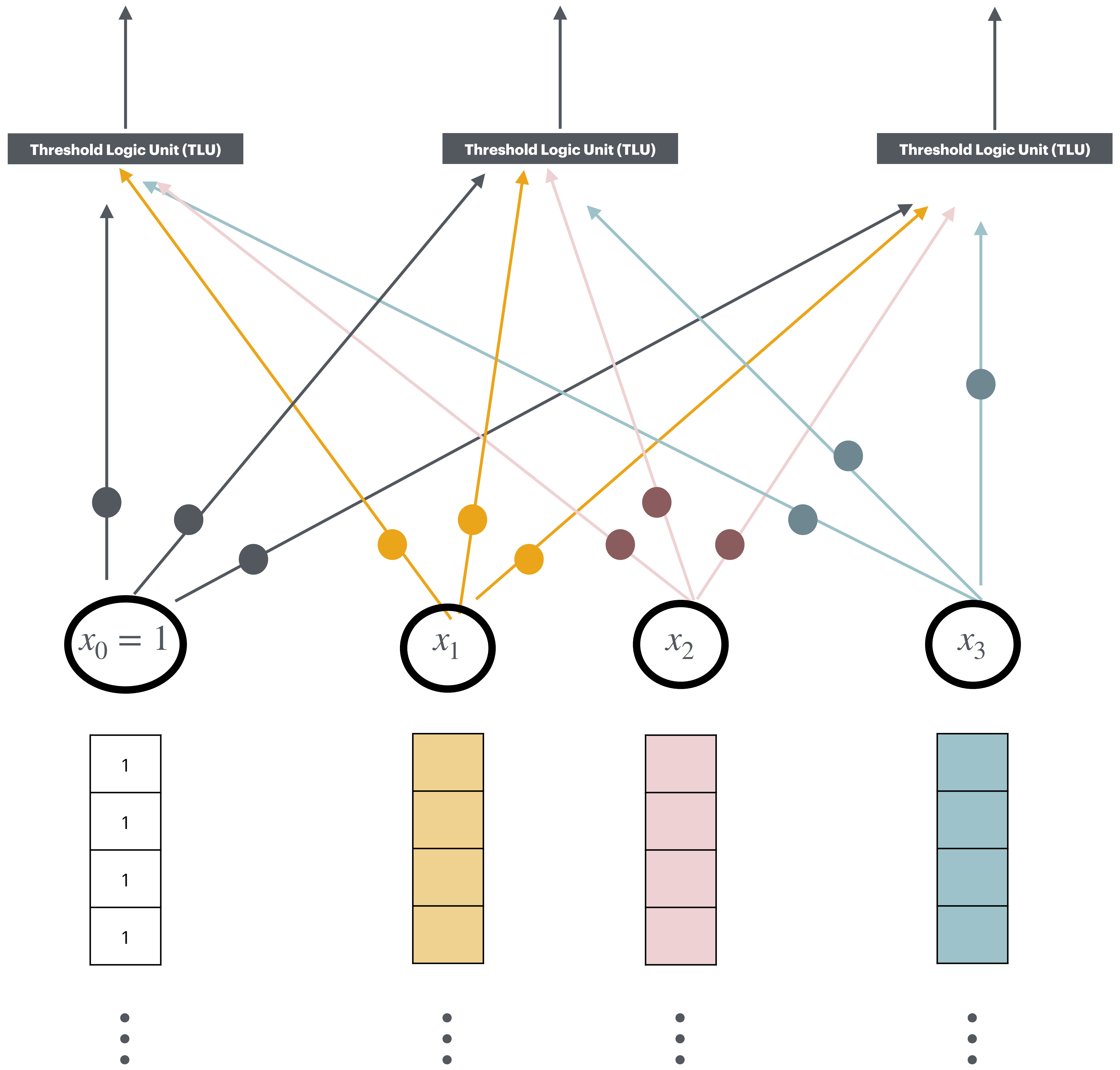


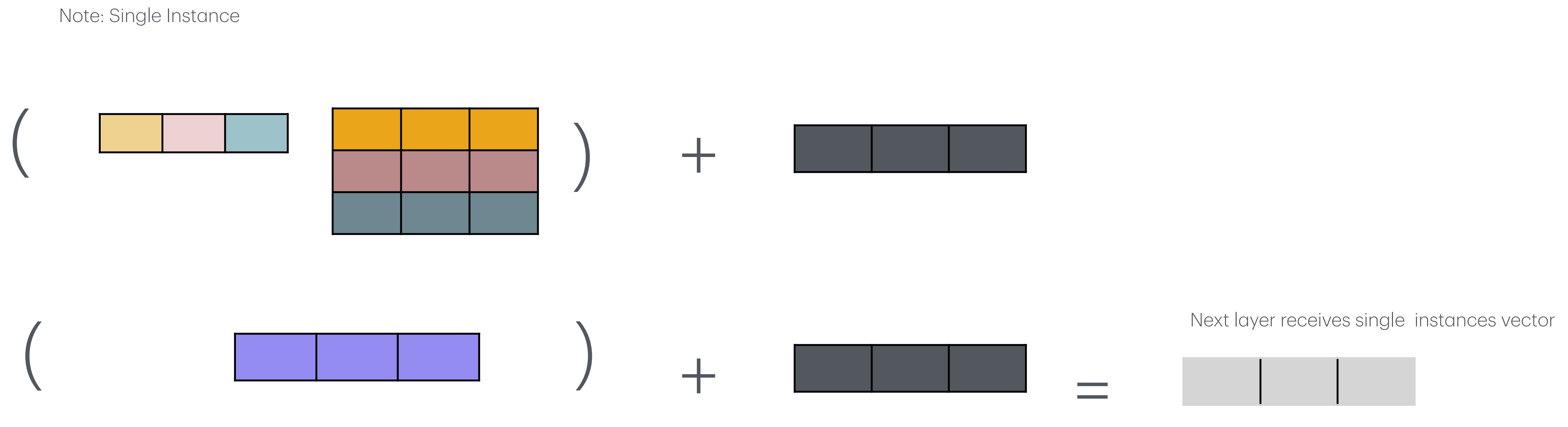
Logical Computation: Perception



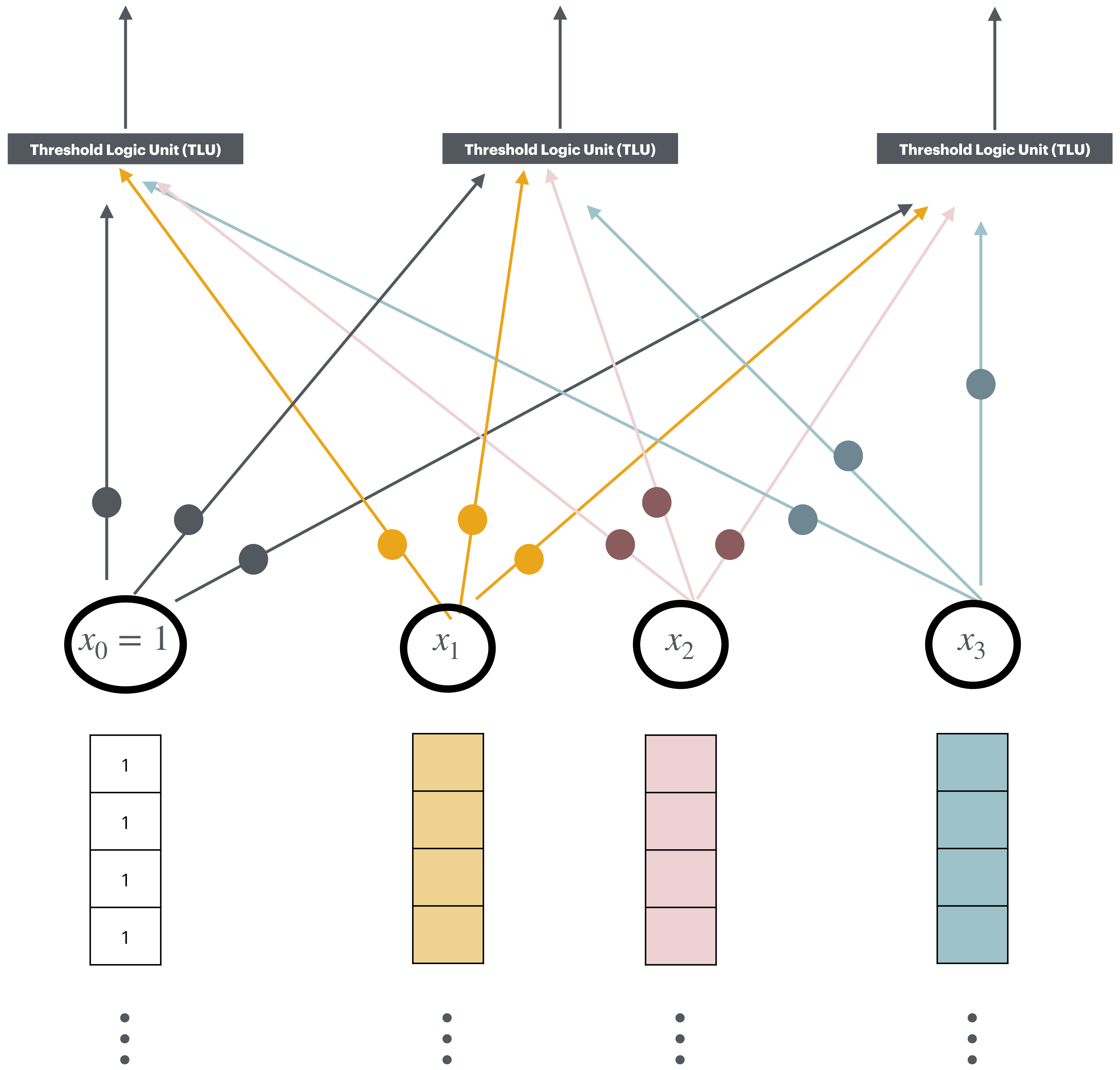


Logical Computation: Perception

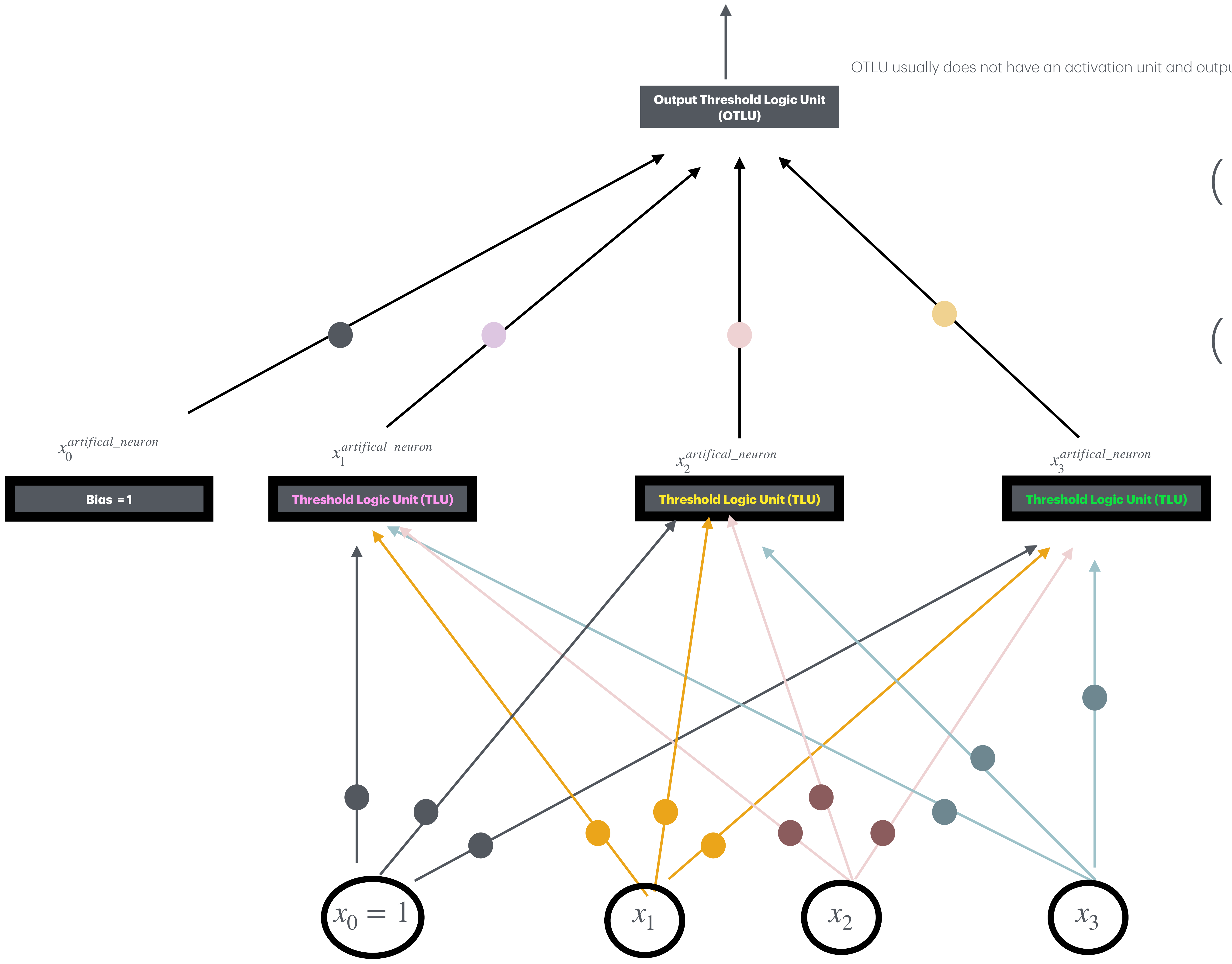


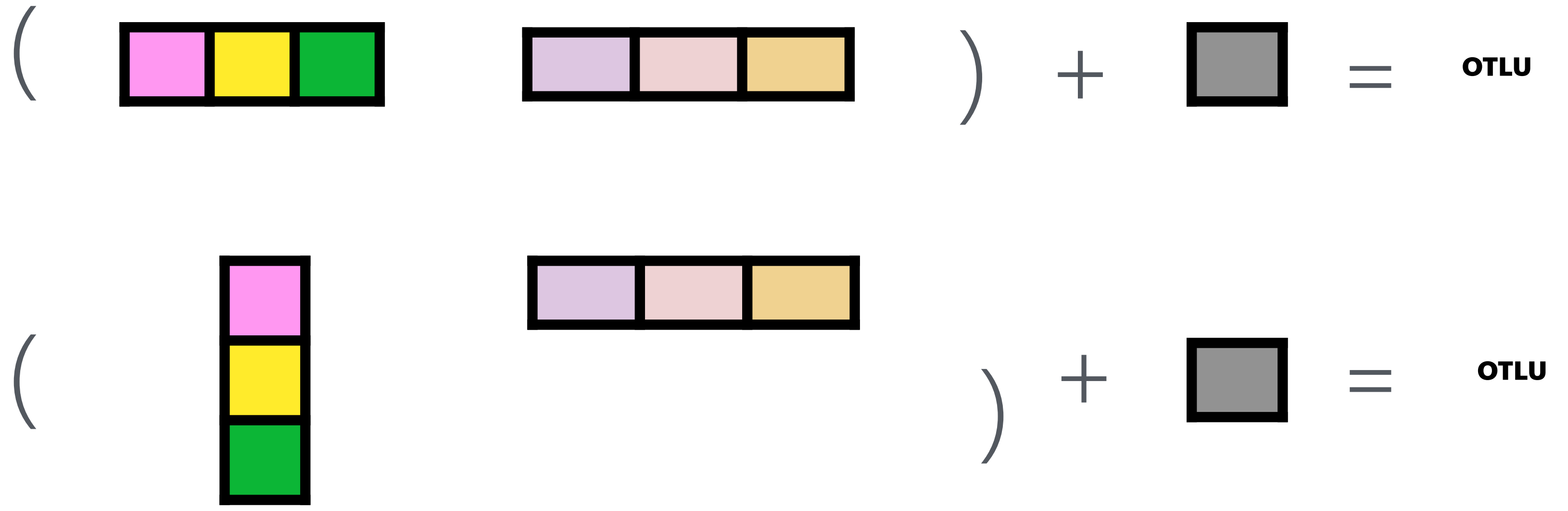


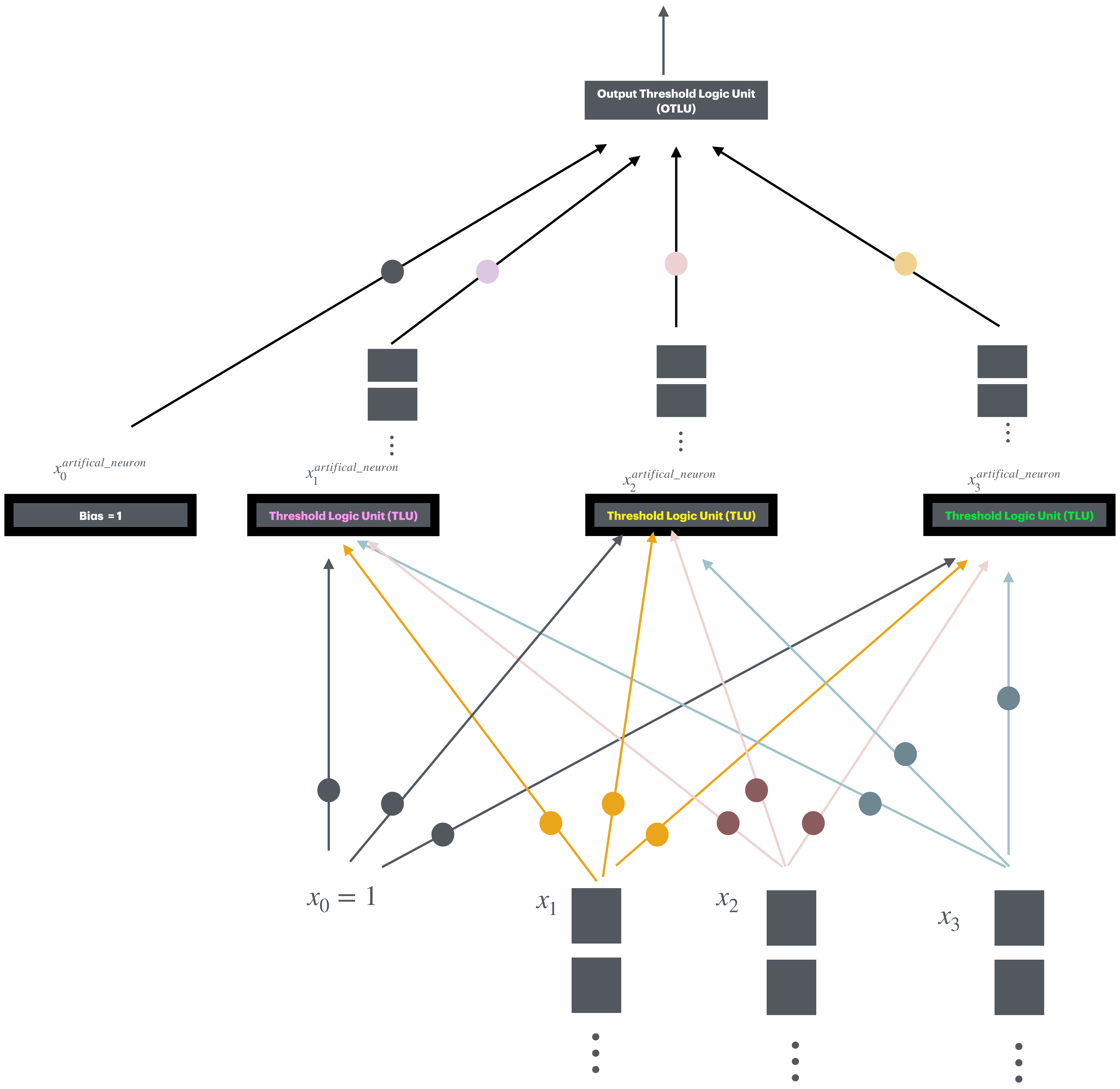
Logical Computation: Perception



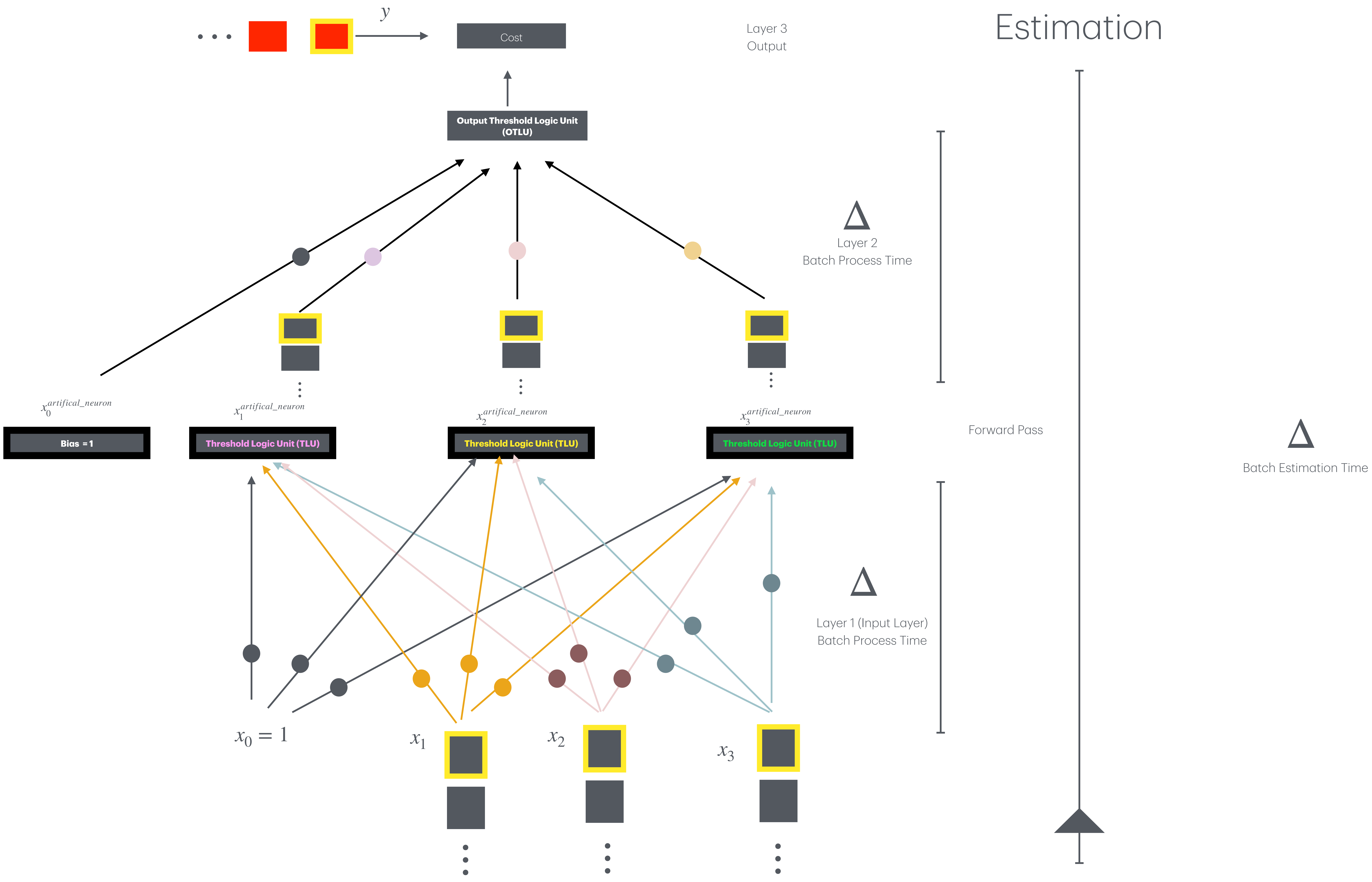
OTLU usually does not have an activation unit and outputs the weighted sum



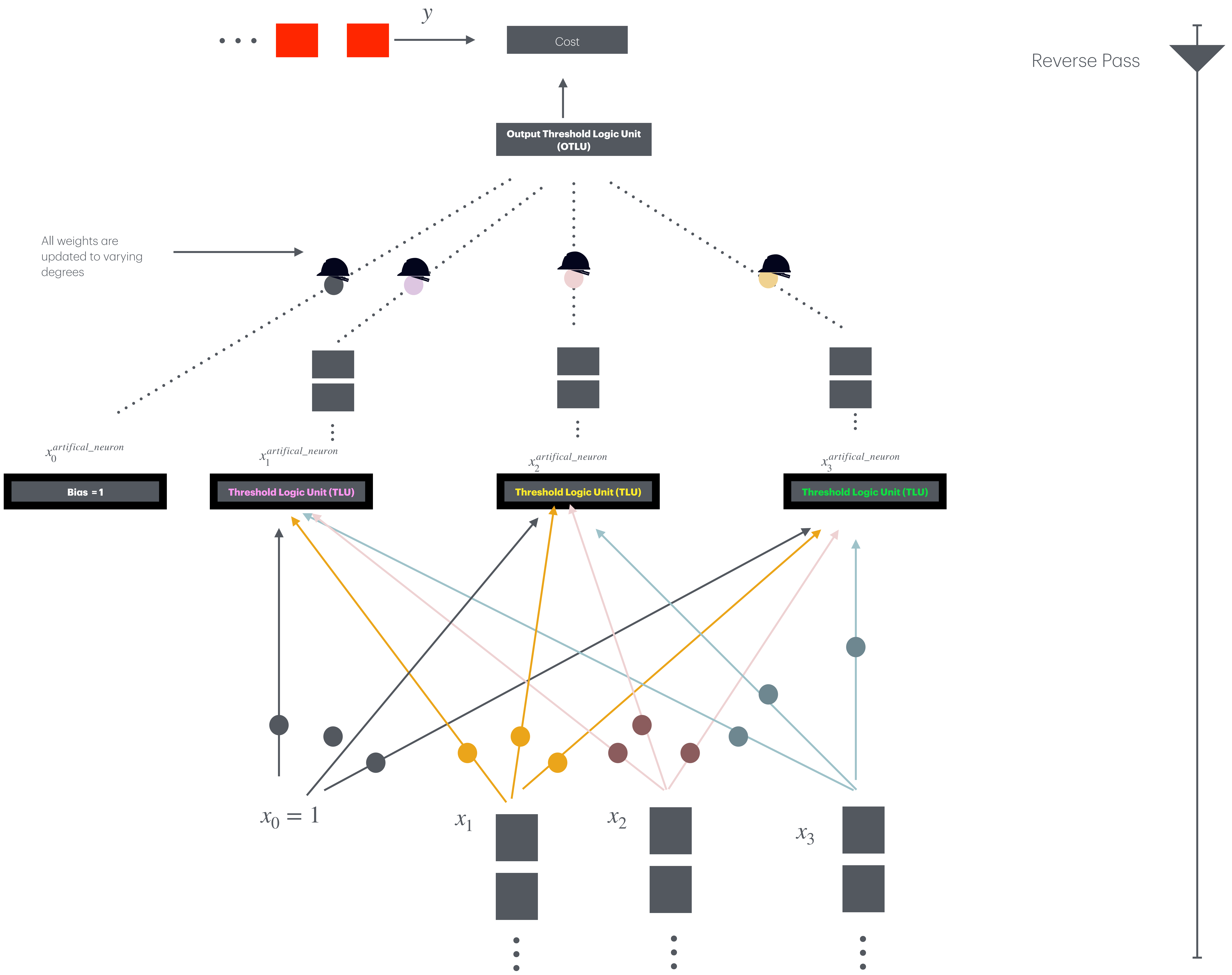




Networks handles mini-batch at a time



y
Cost
Layer 3
Output
Estimation
Output Threshold Logic Unit (OTLU)
Δ
Layer 2
Batch Process Time
$x_0^{artifical\_neuron}$
$x_1^{artifical\_neuron}$
$x_2^{artifical\_neuron}$
$x_3^{artifical\_neuron}$
Bias = 1
Threshold Logic Unit (TLU)
Threshold Logic Unit (TLU)
Threshold Logic Unit (TLU)
Forward Pass
Δ
Batch Estimation Time
Δ
Layer 1 (Input Layer)
Batch Process Time
$x_0 = 1$
x_1
x_2
x_3

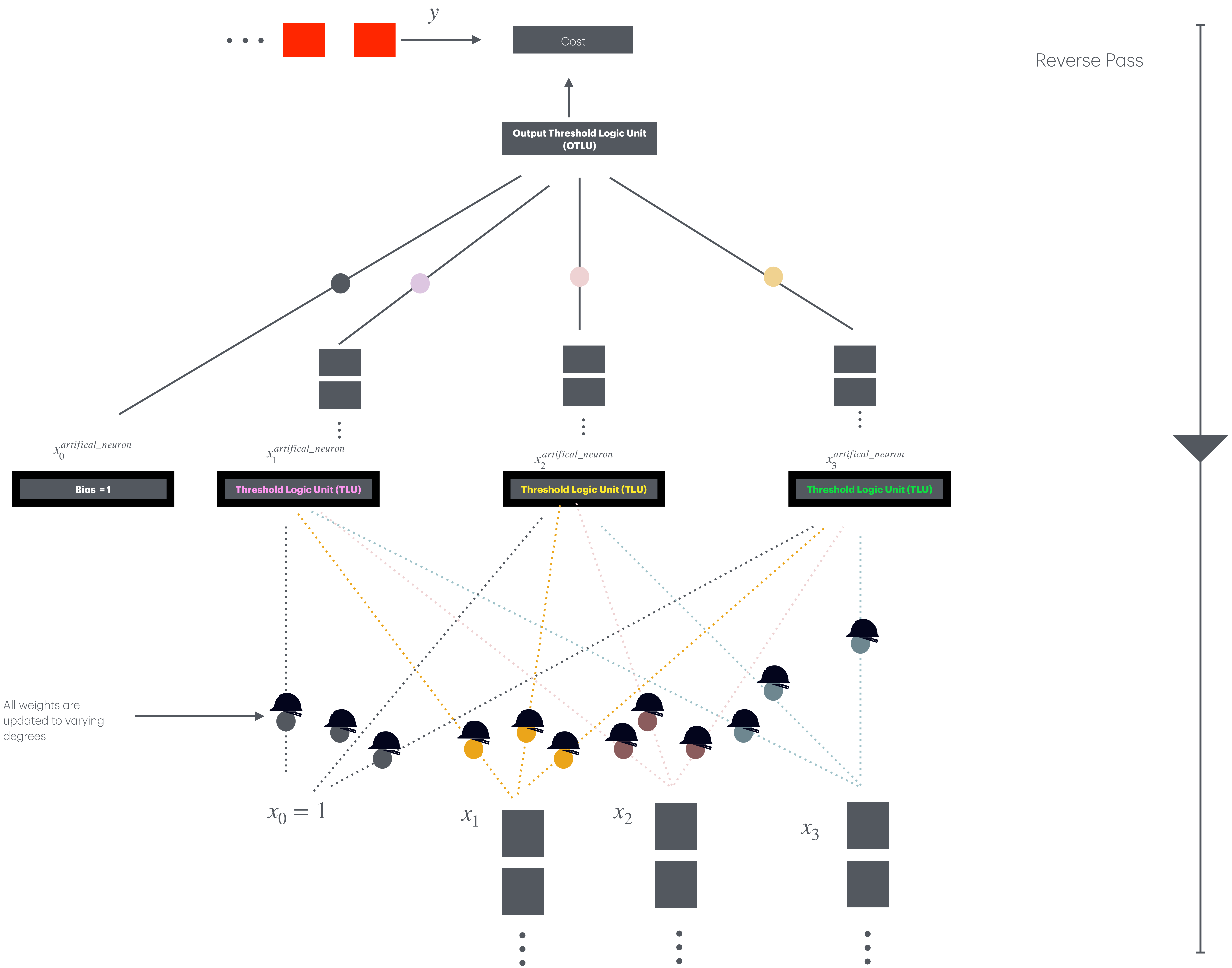


How much **layer 2 connections** contributed to high cost (i.e. high error)

Cost/Error gradients are measured across connections (weights) $\boxed{\frac{Cost}{W_{some\_connection}}}$

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

Note: Input batch persistence is required for reverse algorithm



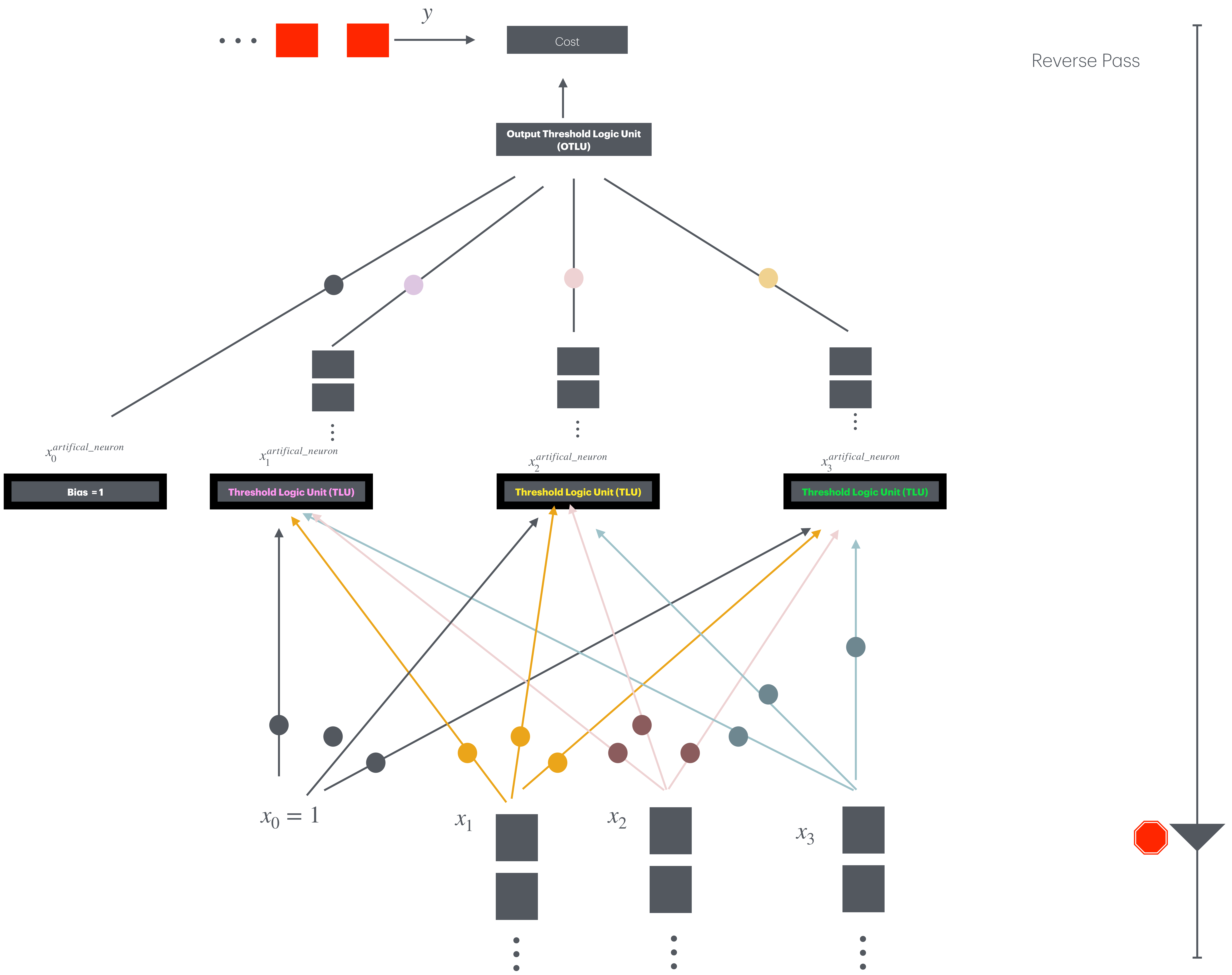
How much **layer 1 connections** contributed to high cost (i.e. high error)

Cost/Error gradients are measured across connections (weights)

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

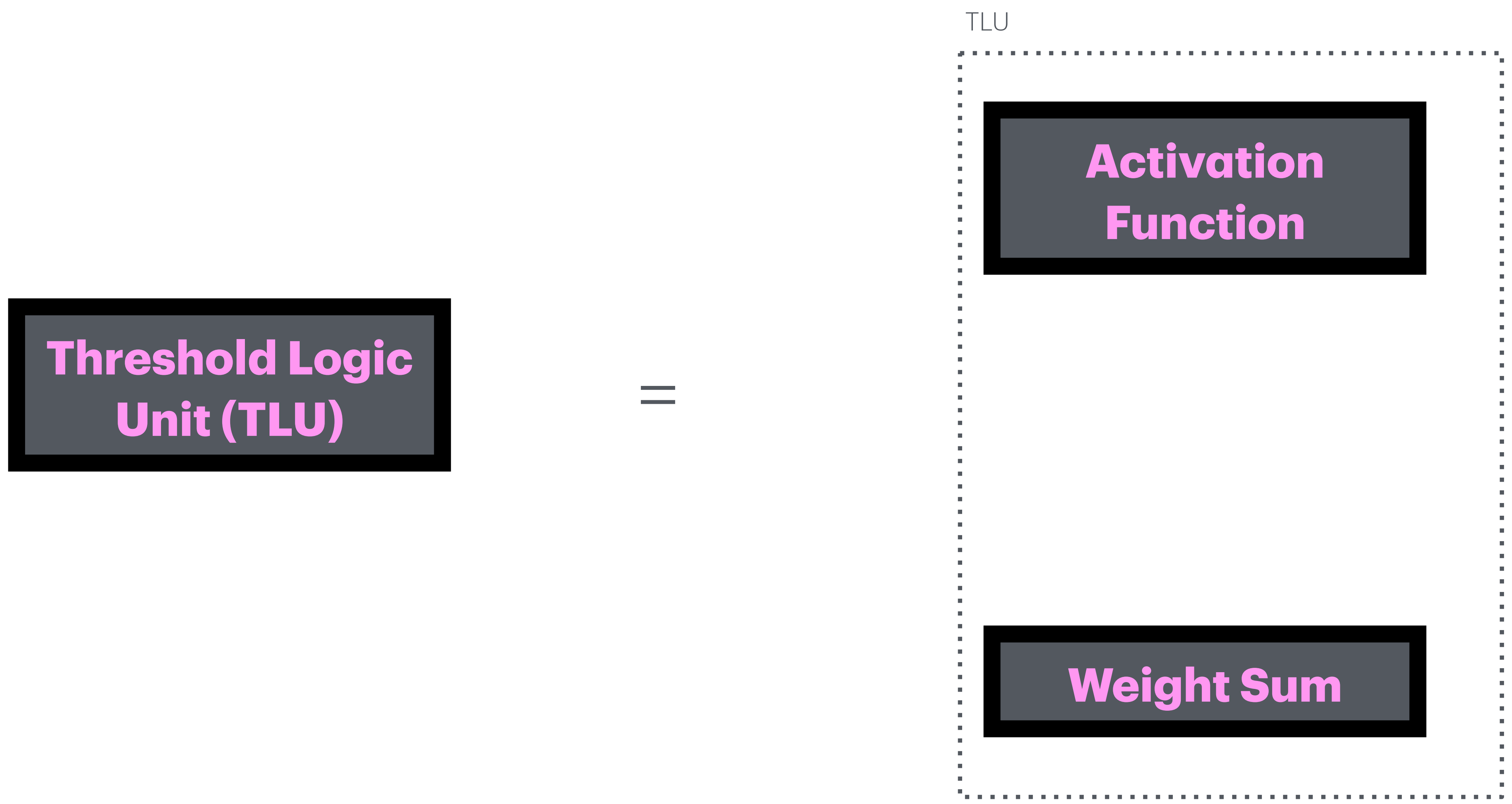
Gradient Descent performed on all connections (weights) using error gradients

Note: Input batch persistence is required for reverse algorithm



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**

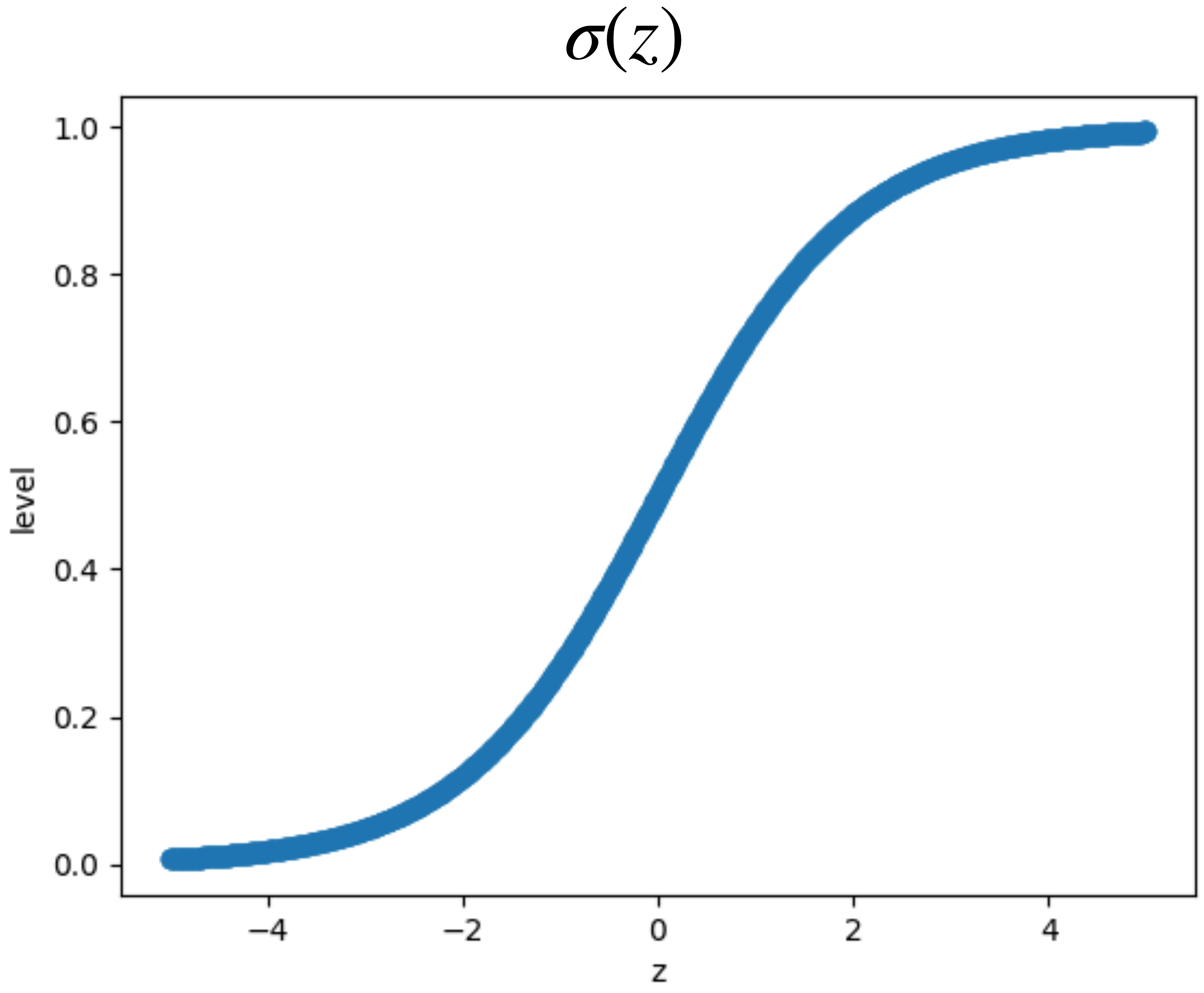
Nonlinear Regression/Classifiers :

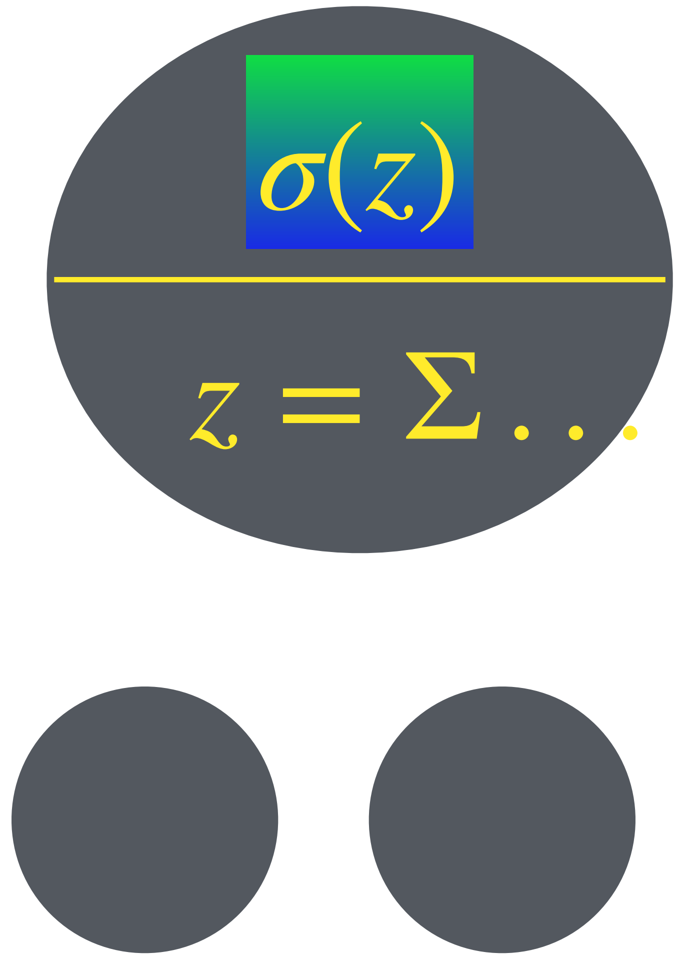
- **Sigmoid Function**
- **Hyperbolic Tangent Function**
- **Rectified Linear Unit Function**

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

Activation Function

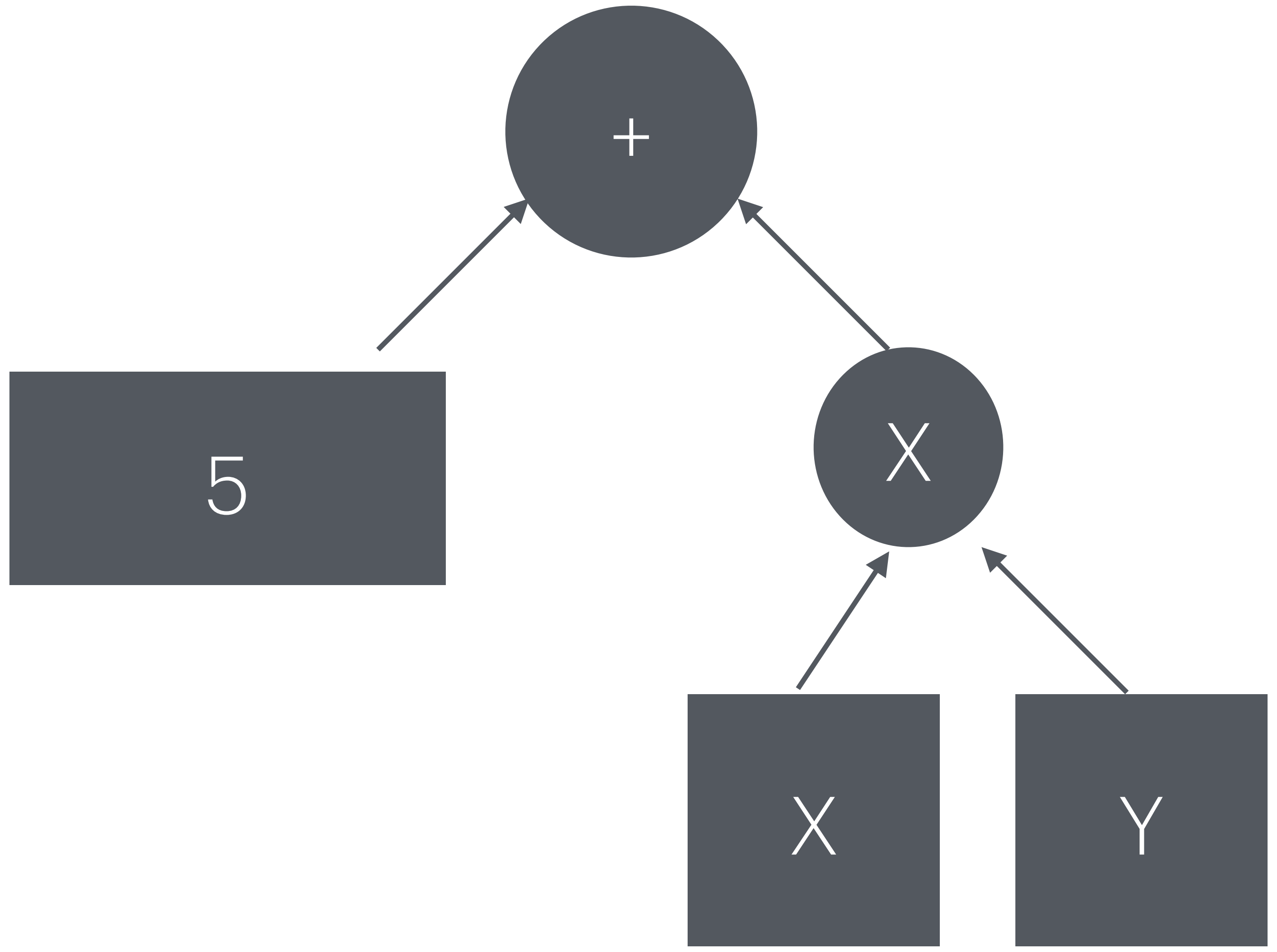
Biological neurons have been observed to implement a roughly **sigmoid activation** function





Forward-Mode AutoDifferentiation

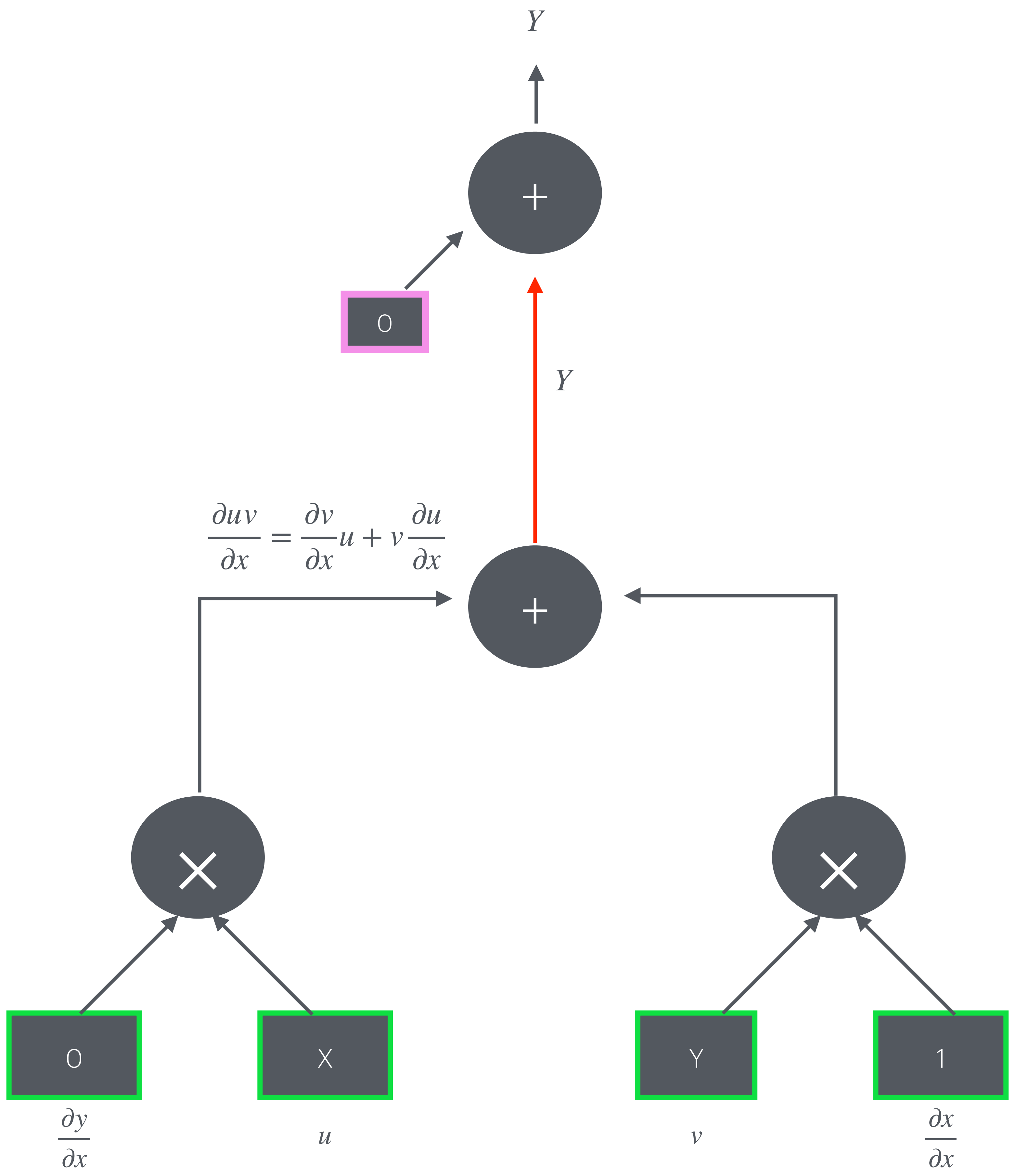
$$g(x, y) = 5 + xy$$



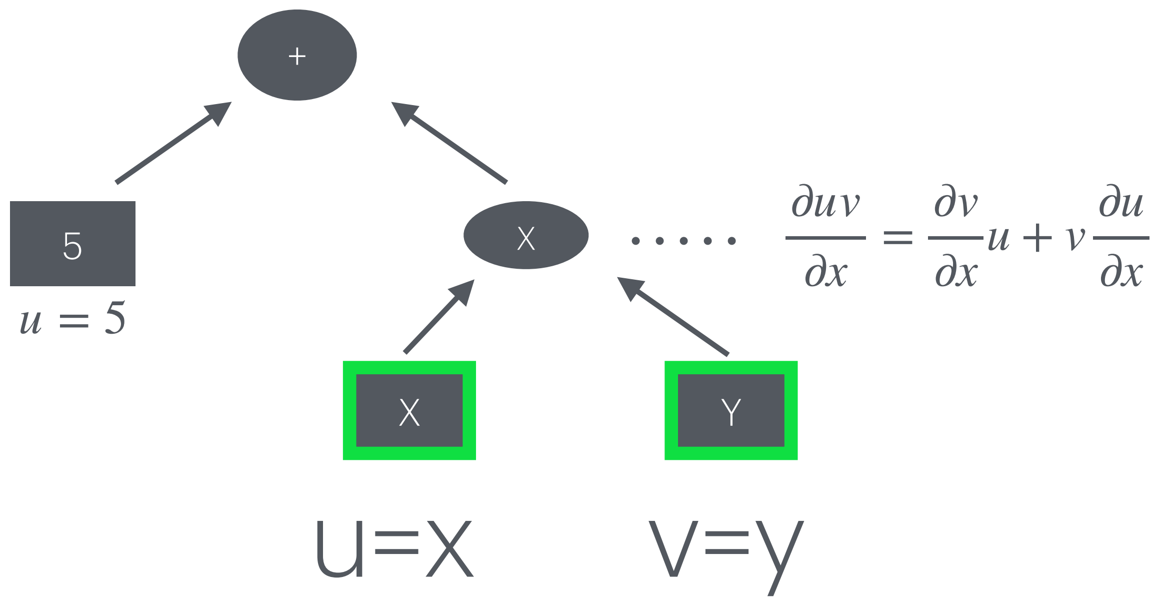
Forward-Mode AutoDifferentiation

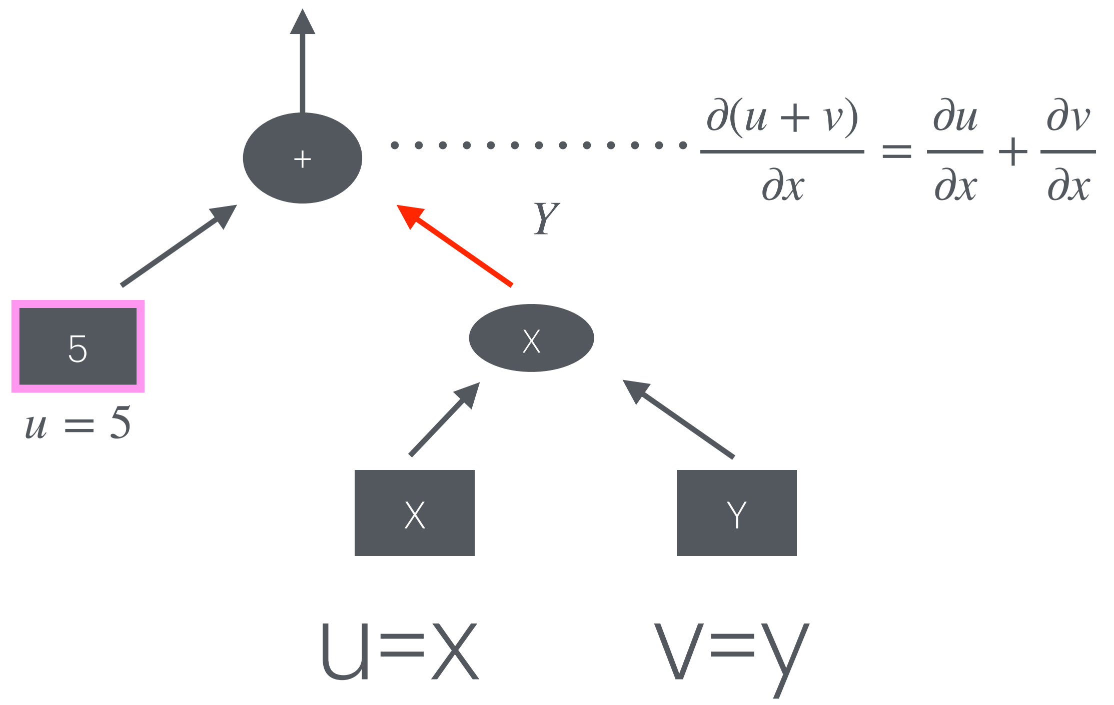
$$g(x, y) = 5 + xy \qquad \text{Partial Derivative } \frac{g(x, y)}{\partial x}$$

Symbolic Differentiation (created from AutoDiff)



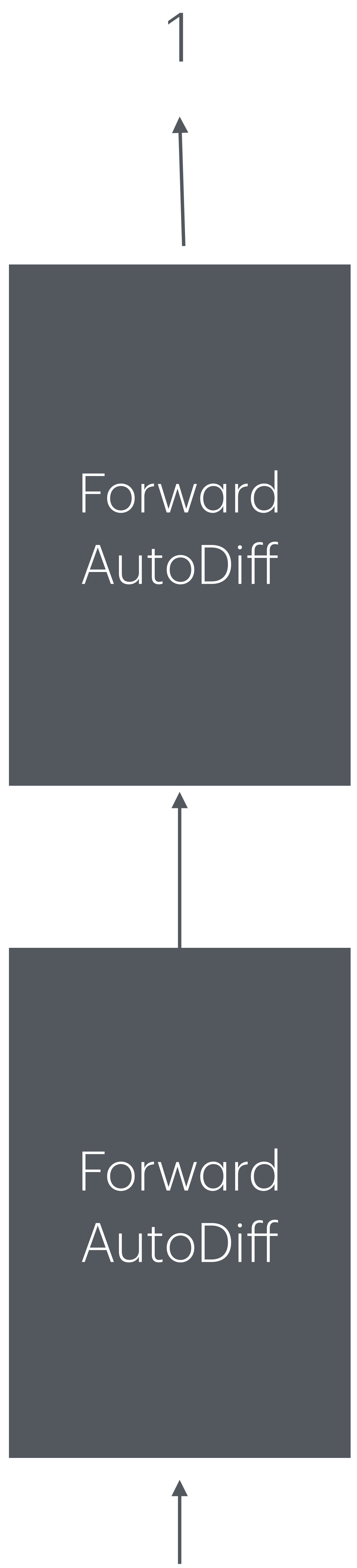
AutoDiff Computation Graphs



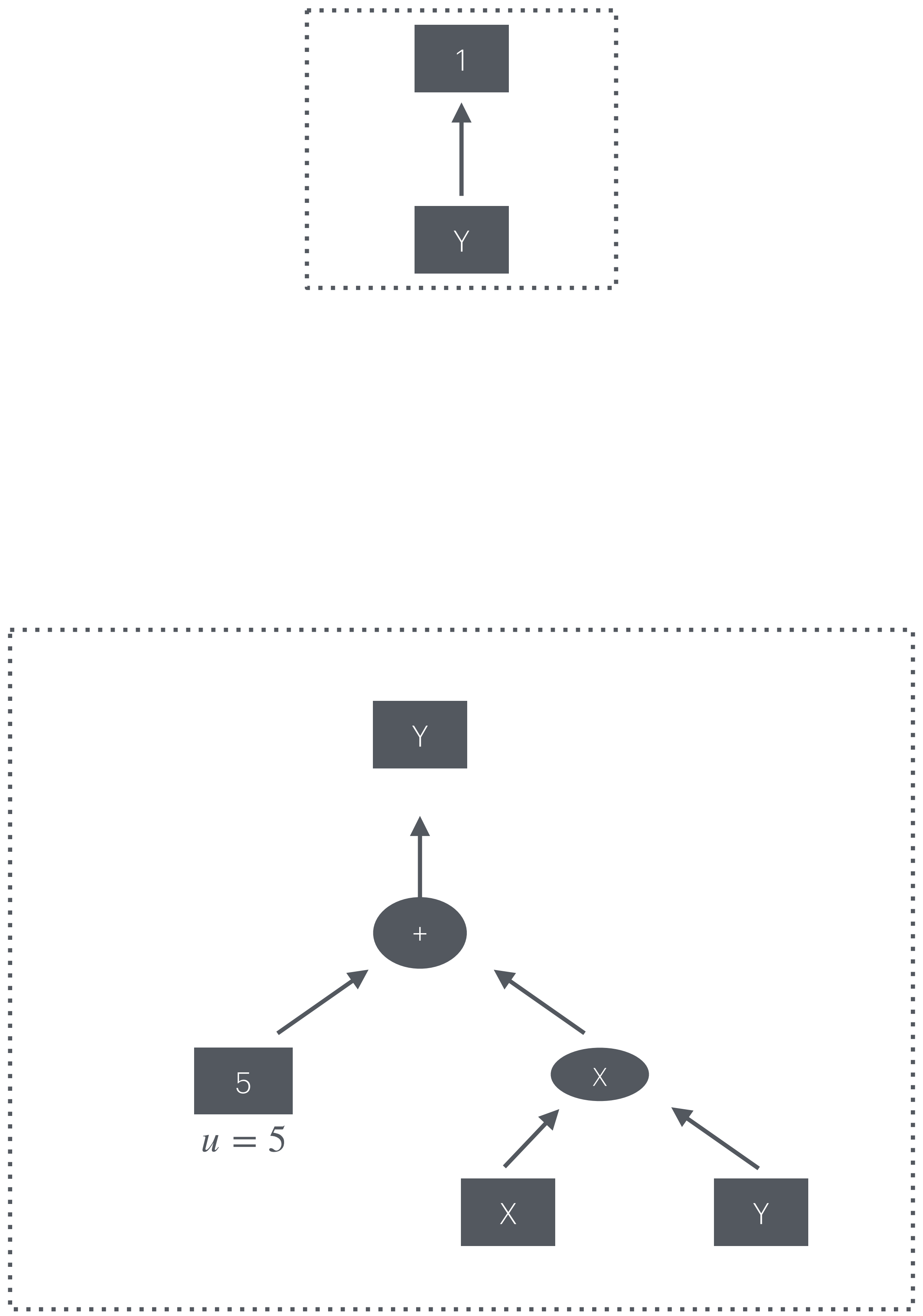


Forward-Mode AutoDifferentiation

Chain Forward Mode AutoDiffs



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



Forward-Mode Dual Number

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

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

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

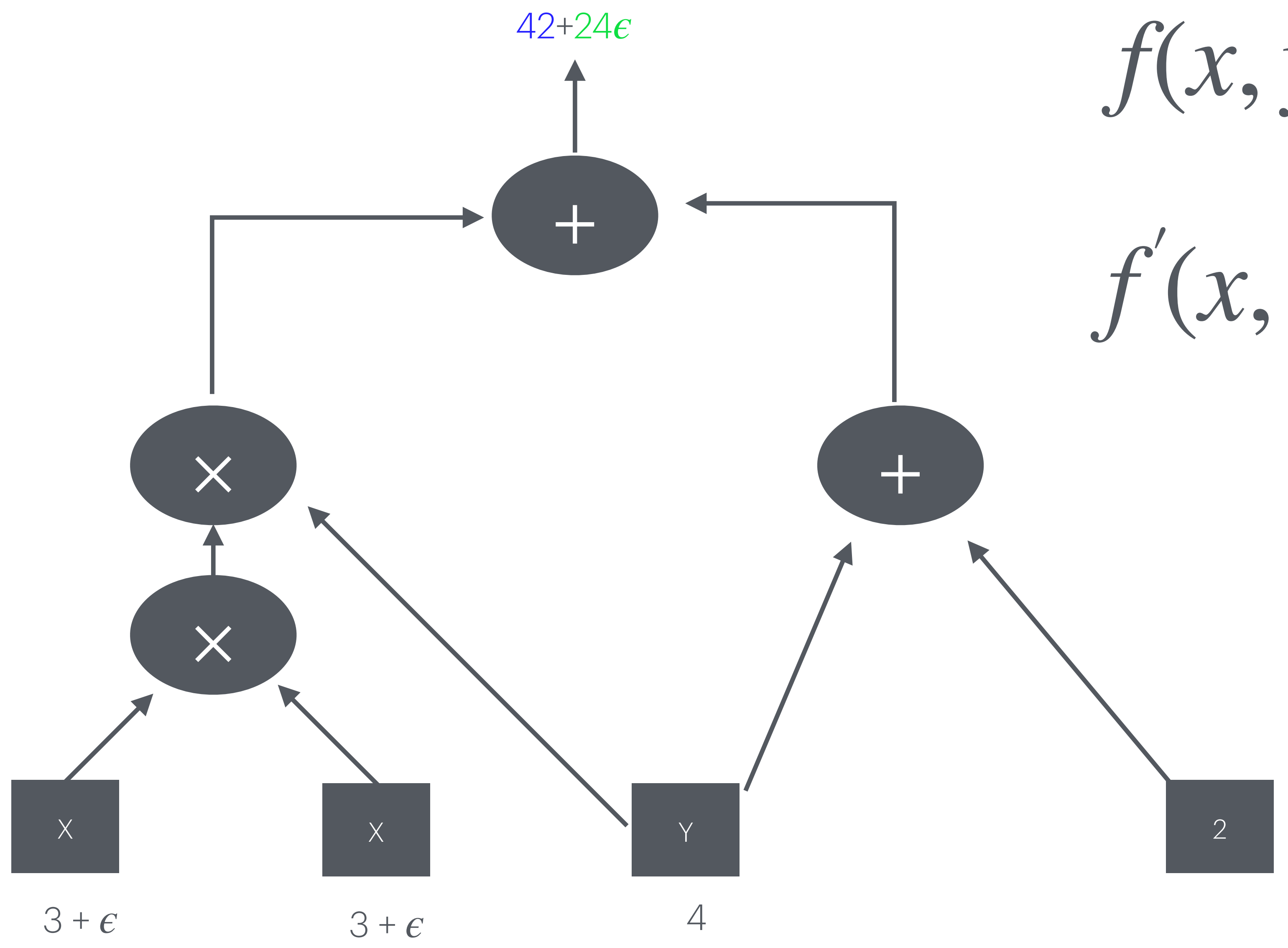
Rule: $h(a+\epsilon) = h(a) + h'(a)\epsilon$

value at point derivative at point

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

Run forward diff to find real and ϵ components

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



Note: $\epsilon^2 = 0$

$f(x,y) = 42$ real component

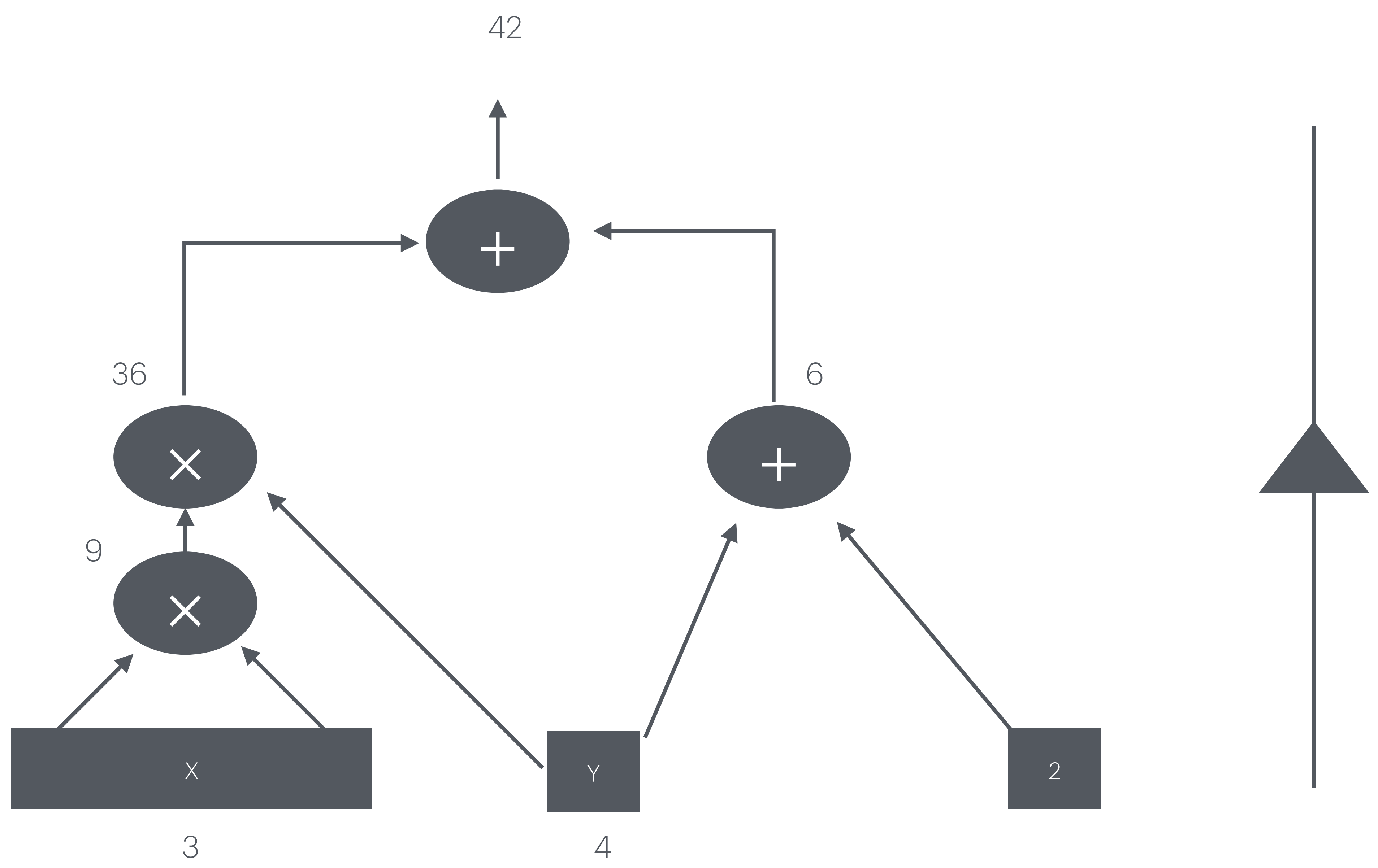
$f'(x,y) = 24$ ϵ component

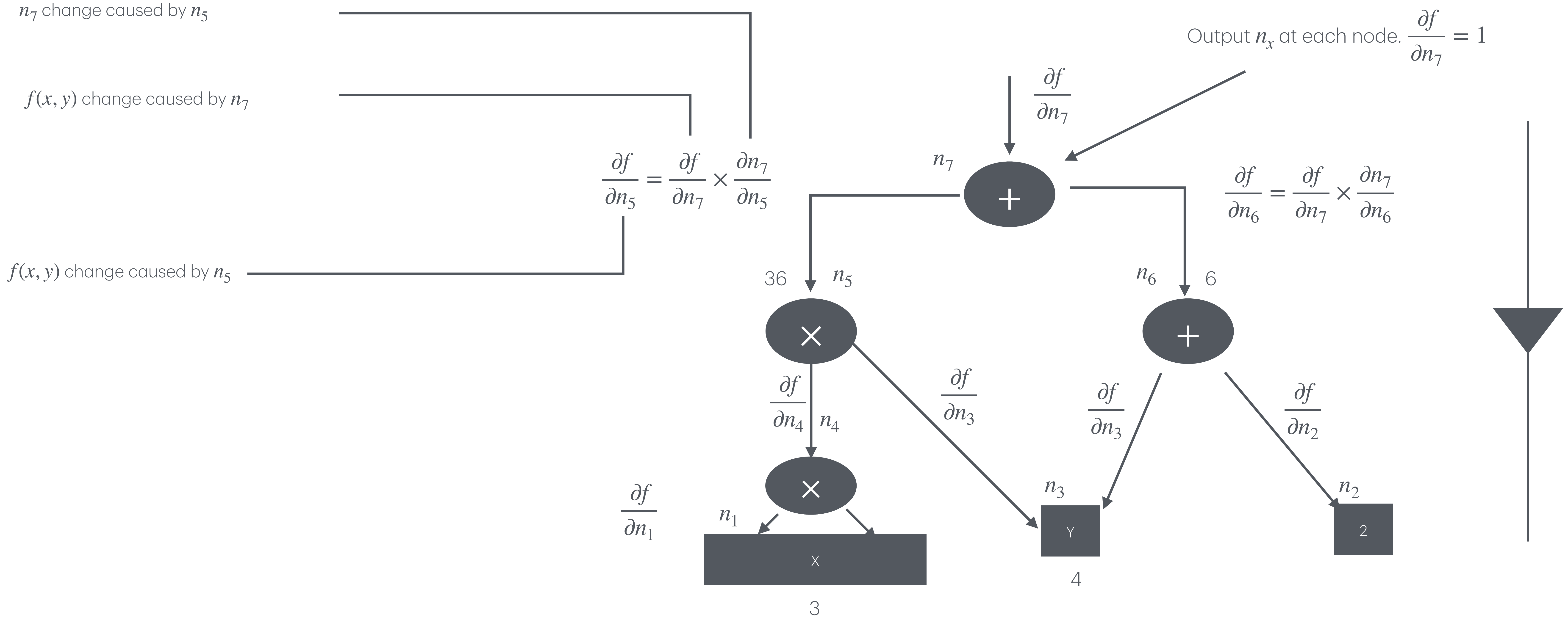
Partial derivative with respect to y requires the same process

Reverse-Mode AutoDifferentiation

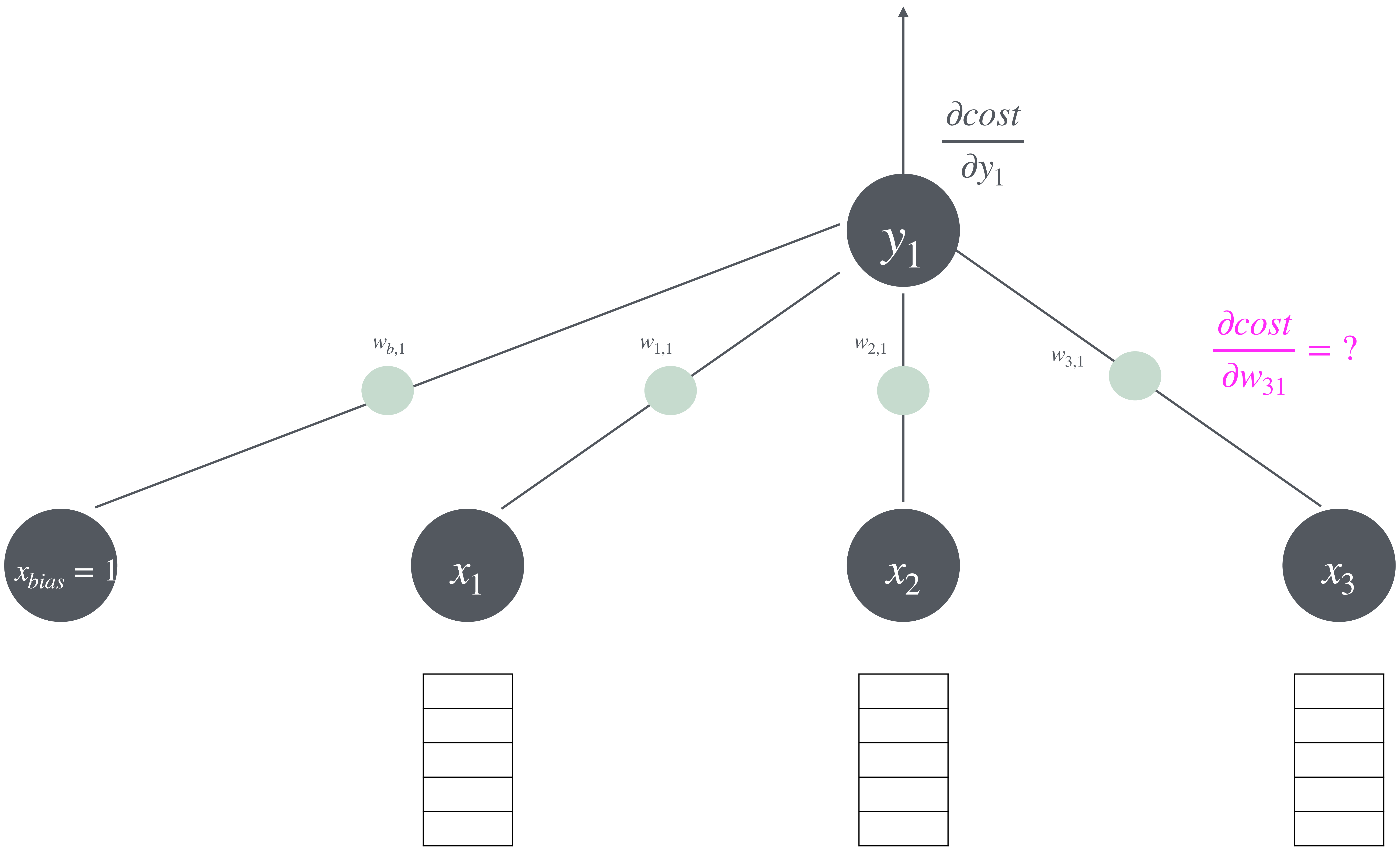
$$f(x, y) = x^2y + y + 2$$

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



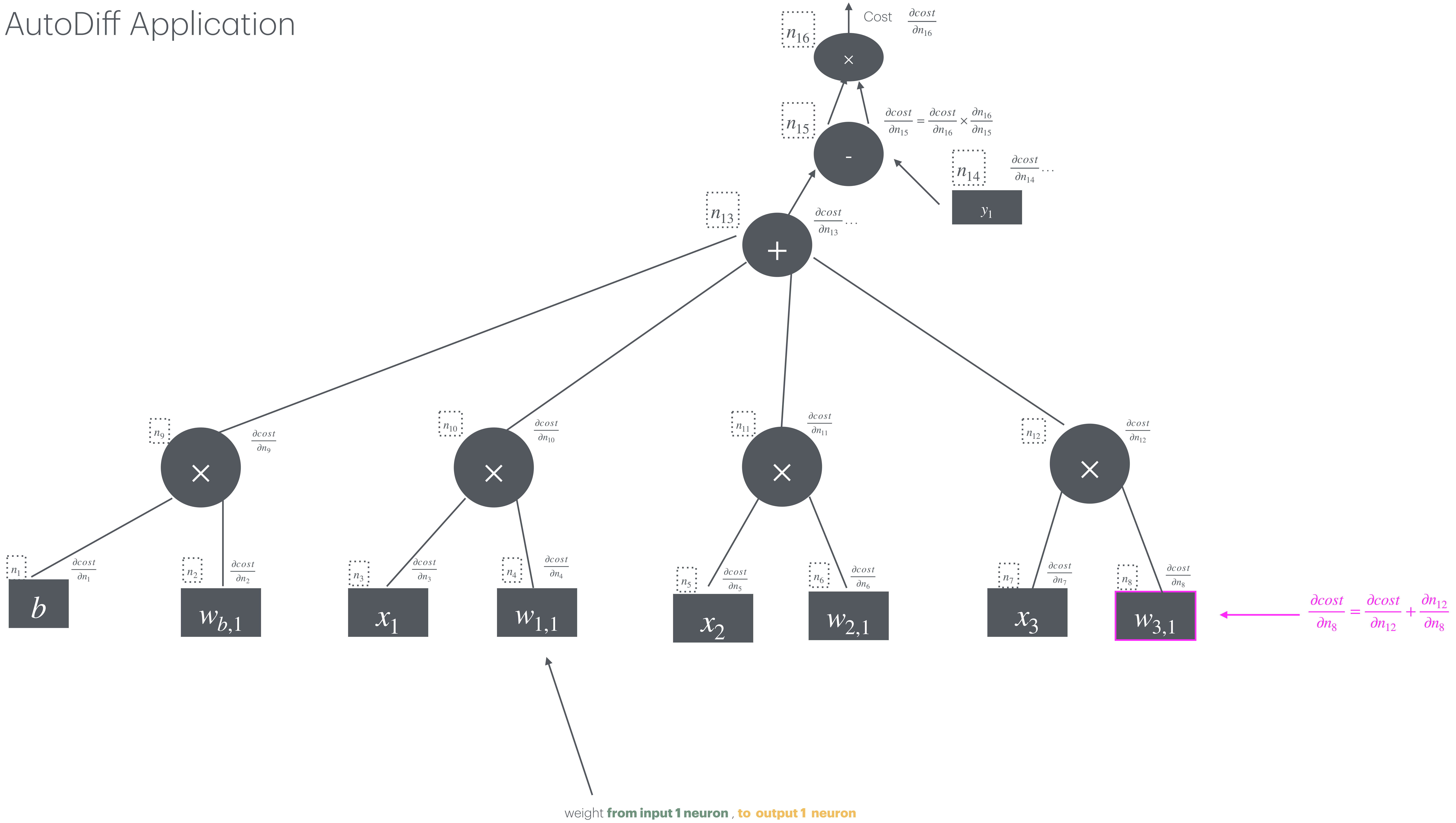


AutoDiff Application



How?

AutoDiff Application



Tensorflow

Keras(API)

Keras(implementation)

Libraries containing Keras

- ★ Tensorflow
- Microsoft Cognitive Toolkit
- Theano
- ★ Keras(API)
- ★ PyTorch

★ – popular

Tensorflow

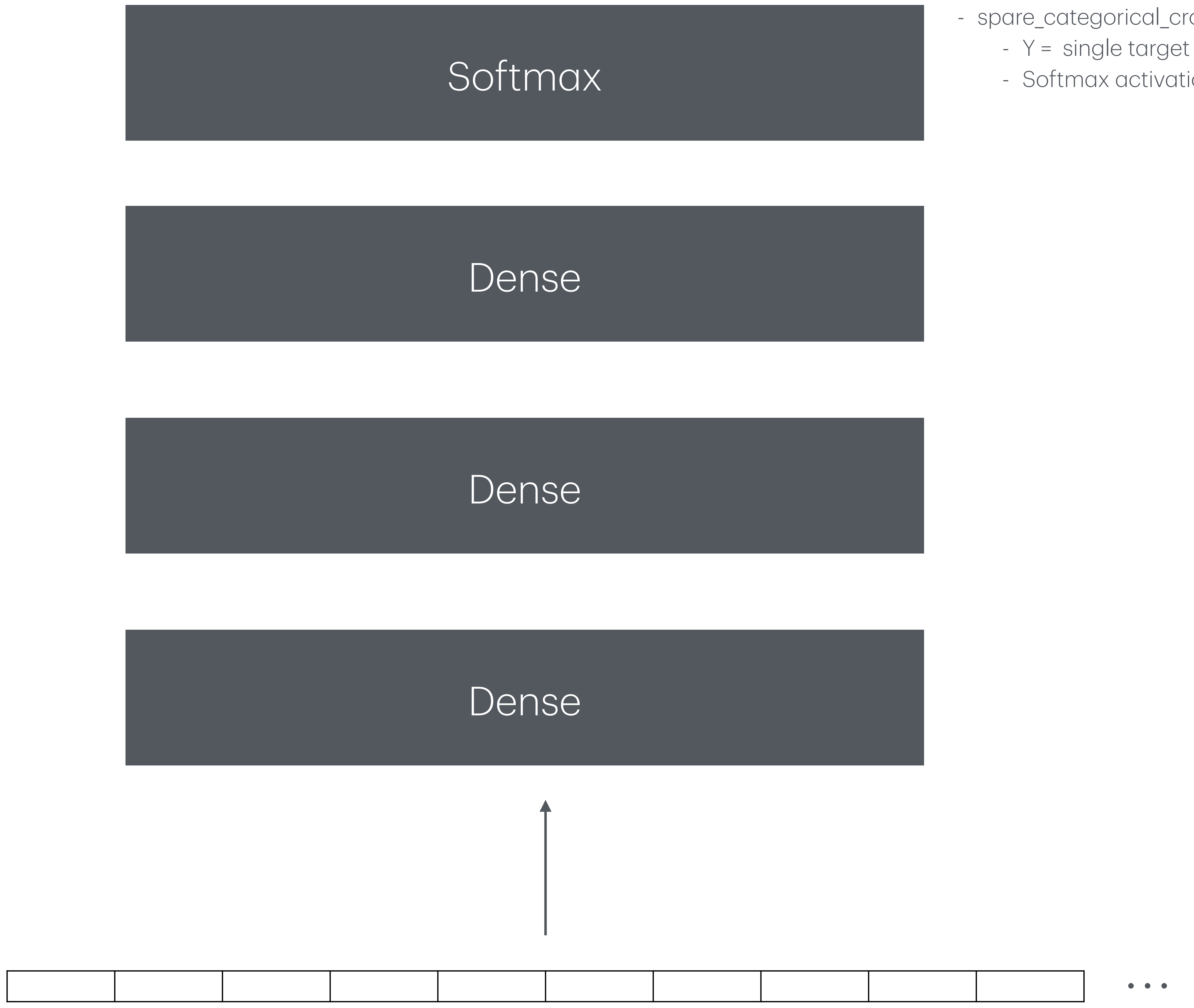
keras

Tensor flow Features

Others containing Keras

- Javascript/Typescript
- PlaidML
- Apple's Core ML
- Apache MXNet

Sequential Model: Classify Fashion MNIST



1D Unit = 1 x N
N= Number of Pixels

Loss:
- spare_categorical_crossentropy
  - Y = single target index (1 of 10 choices)
  - Softmax activation in output layer

Other Losses
- categorical_crossentropy
  - Y = one hot vector of probabilities
  - Softmax activation in output layer
- binary_crossentropy
  - Y = [0]
  - Y= [1 , 0]
  - Single/Multilabel binary classification
  - Sigmoid activation in output layer

Dealing with skewed data

Class A Class B Class C Class D

Class weights

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

Instance 1

Instance 2

Instance 3

Instance 4

⋮

Samples weights

sample weight example:
Instances labeled by **expert** and **crowdsourcing**
More weight towards the **expert** instances

Saving

- Functional
- Sequential

Declarative

Static Graph

save

clione

inspect

- Subclassing

Conditional loops

Varying shapes

Branching

~~save~~

~~clione~~

~~inspect~~

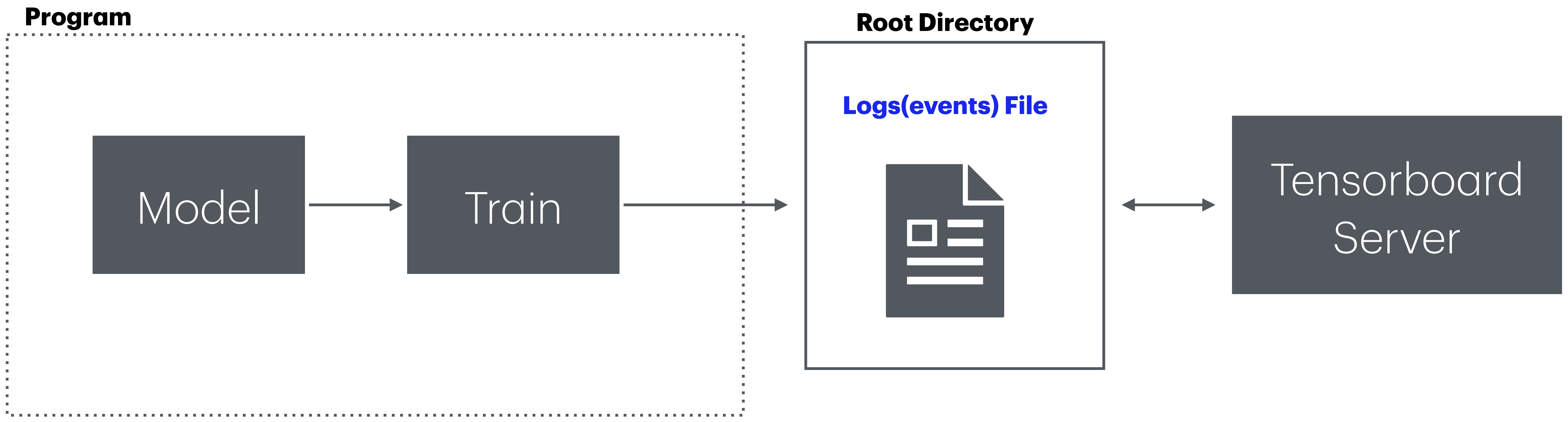
Keras cannot do, in this mode

Save and load model weights yourself

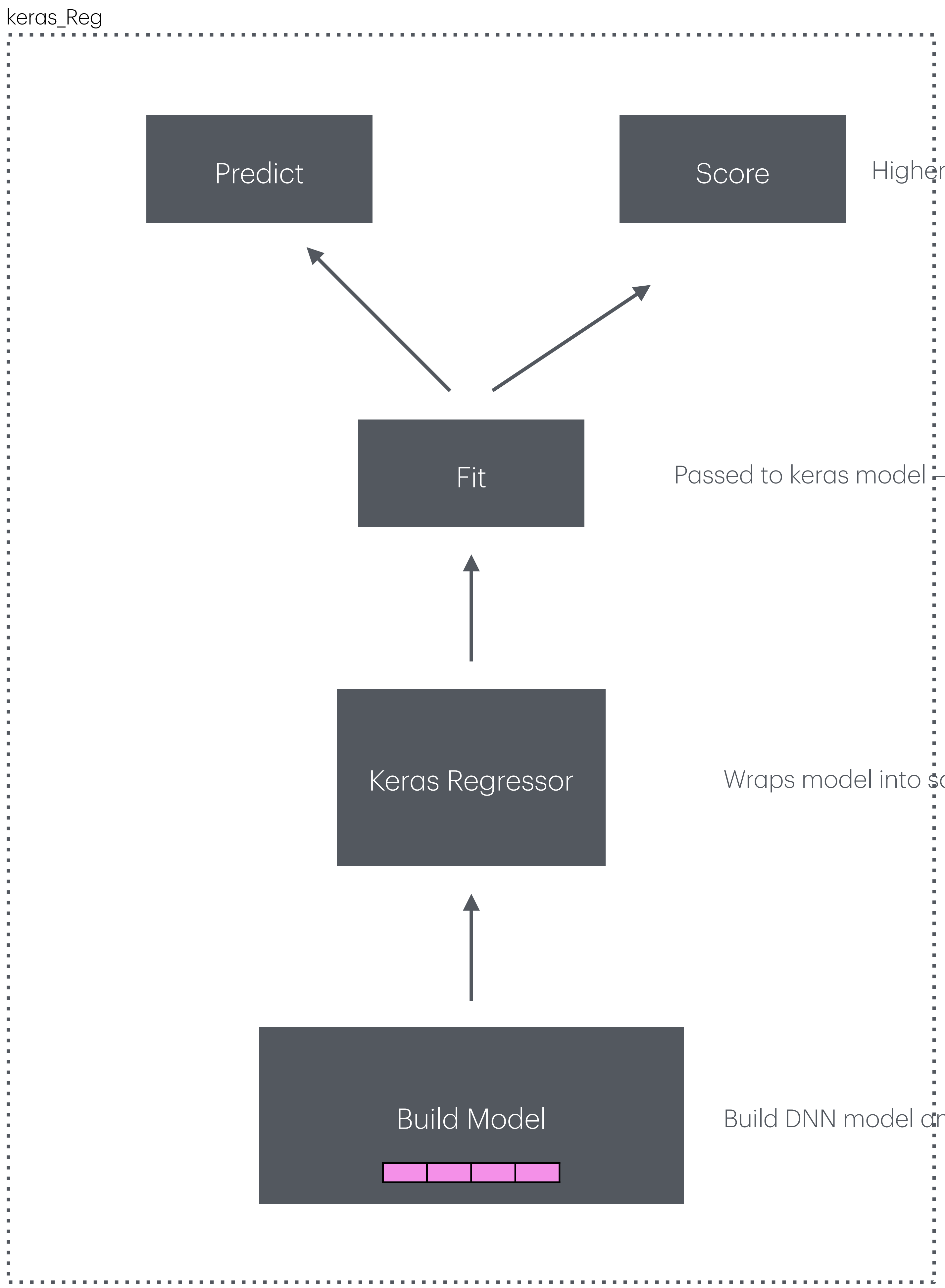
Saving


Architecture
Compile
- loss
- search/optimizer method (i.e. SGD)
Fit
Save
Export Artifacts- TF Serving
Save SOME_NAME.keras file

Tensorboard



Fine-Tuning Neural Networks



Note: Scikit-Learn wants scores not losses

Higher the better

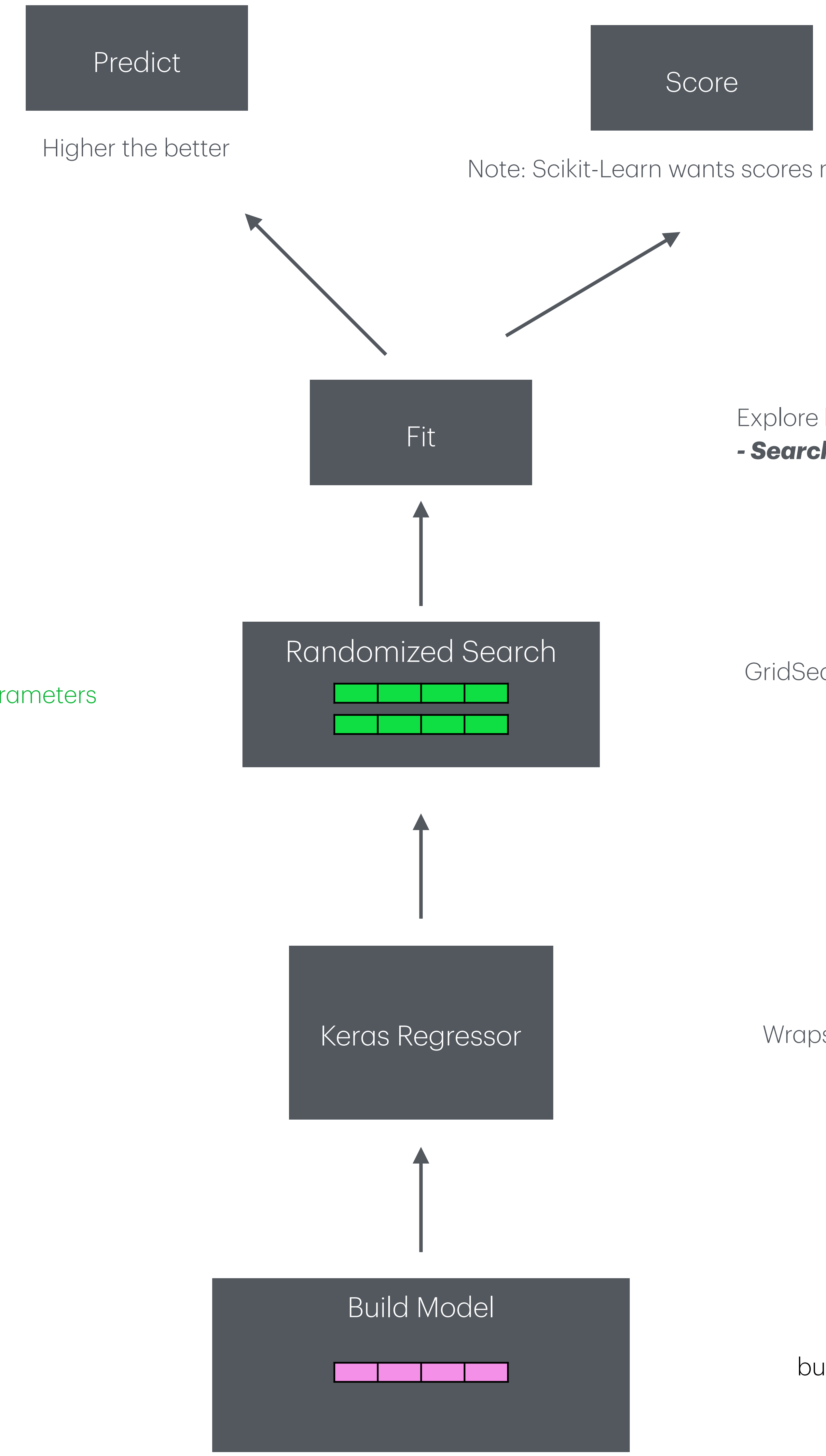
Passed to keras model —>Add training data, validation, callbacks, etc ...

Wraps model into scikit-learn regressor (e.g. scikit-learn classifier used to train or fit clustered data

Build DNN model and compile

build_model (Parameters)

Fine-Tuning Neural Networks



Higher the better

Note: Scikit-Learn wants scores not losses

Explore **hyper-parameters** and find optimal ***build_model***
- Search uses K-fold cross-validation

Hyperparameters

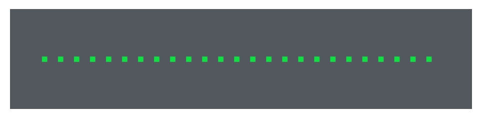
GridSearch could be used instead.

Wraps model into scikit-learn regressor (e.g. scikit-learn classifier used to train or fit clustered data

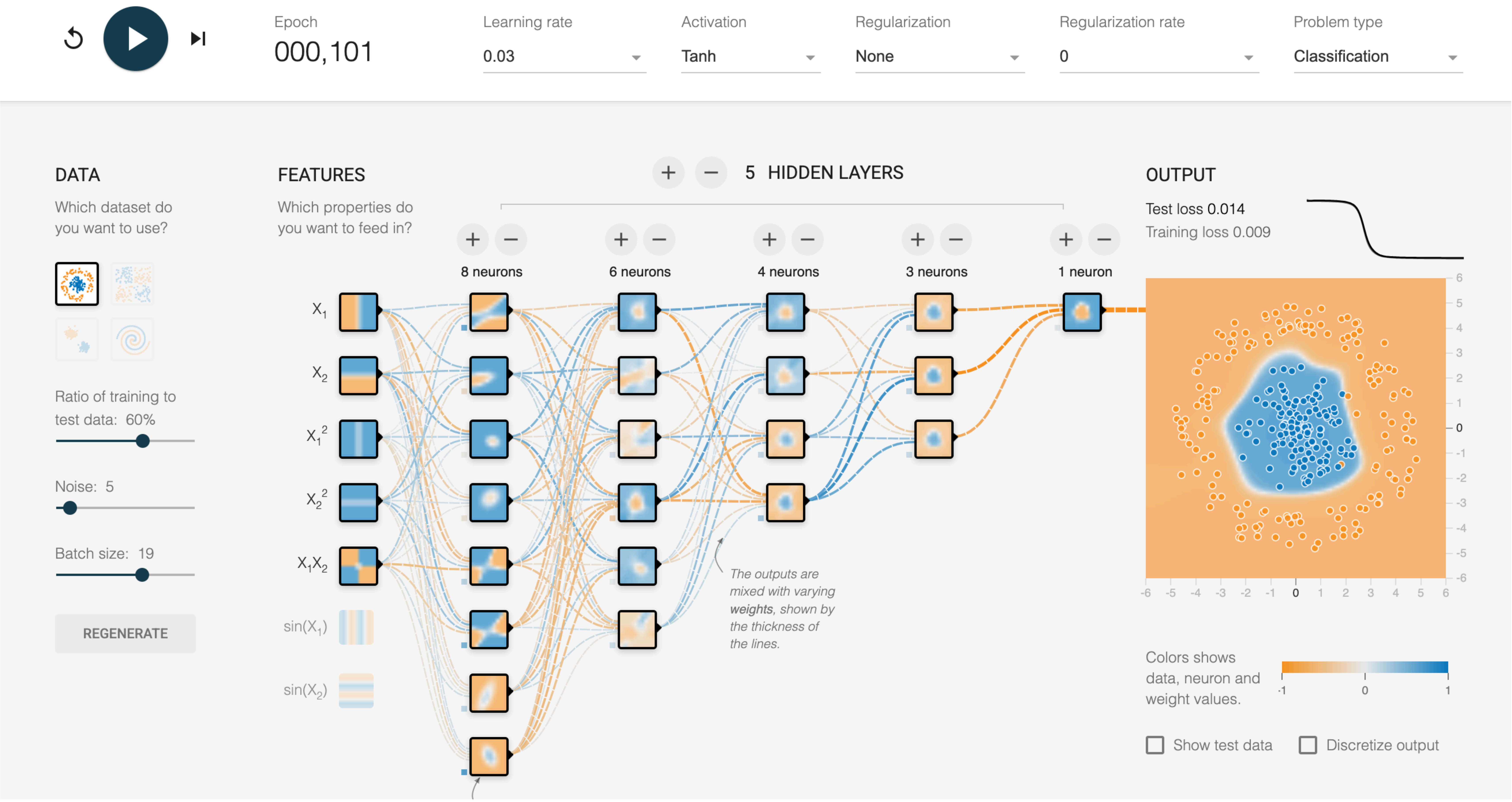
build_model (Parameters)

Networks learns in hierarchical way

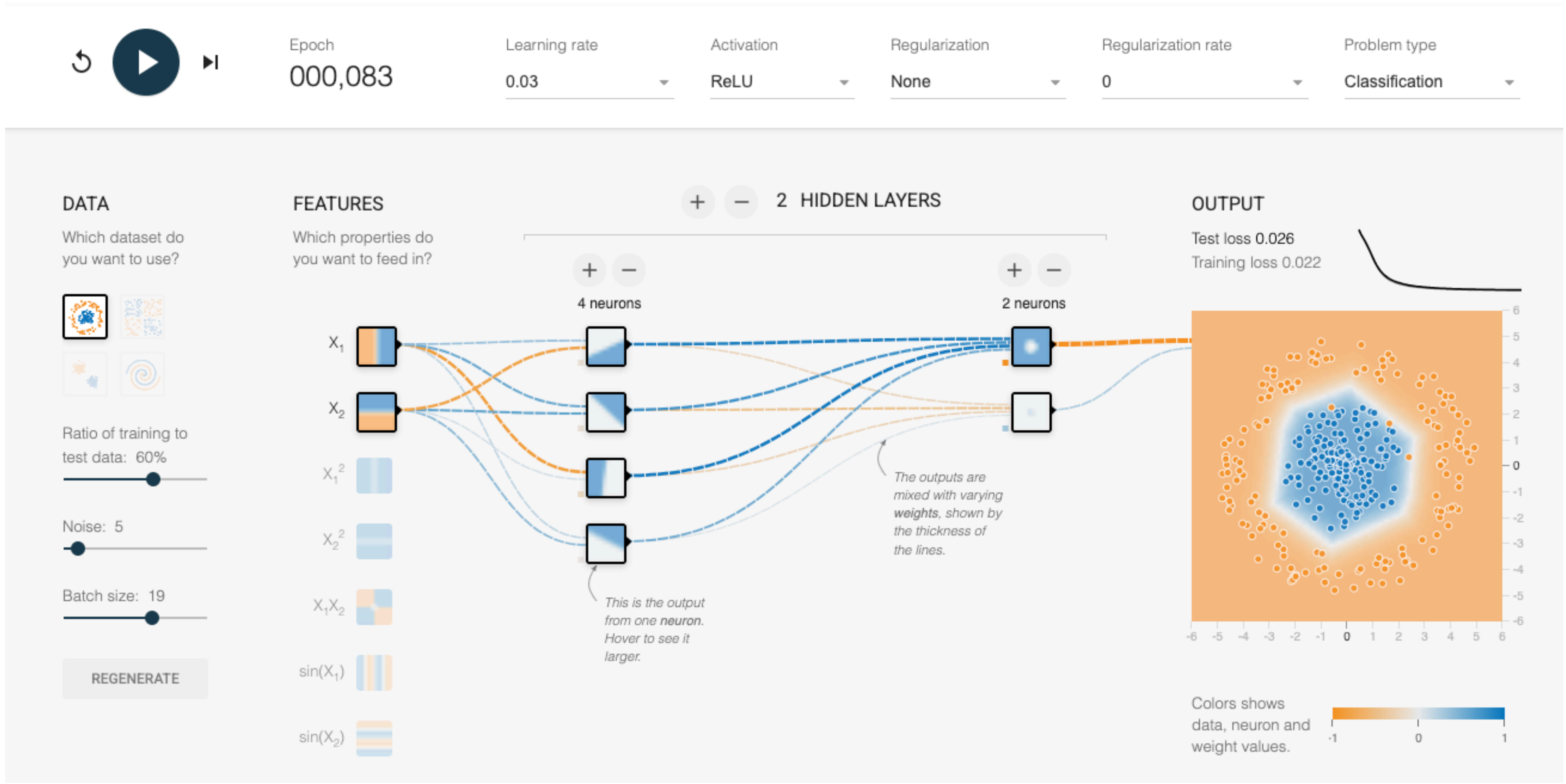
Complex problems deep networks
Have higher parameter efficiency.
Fewer neurons needed per layer



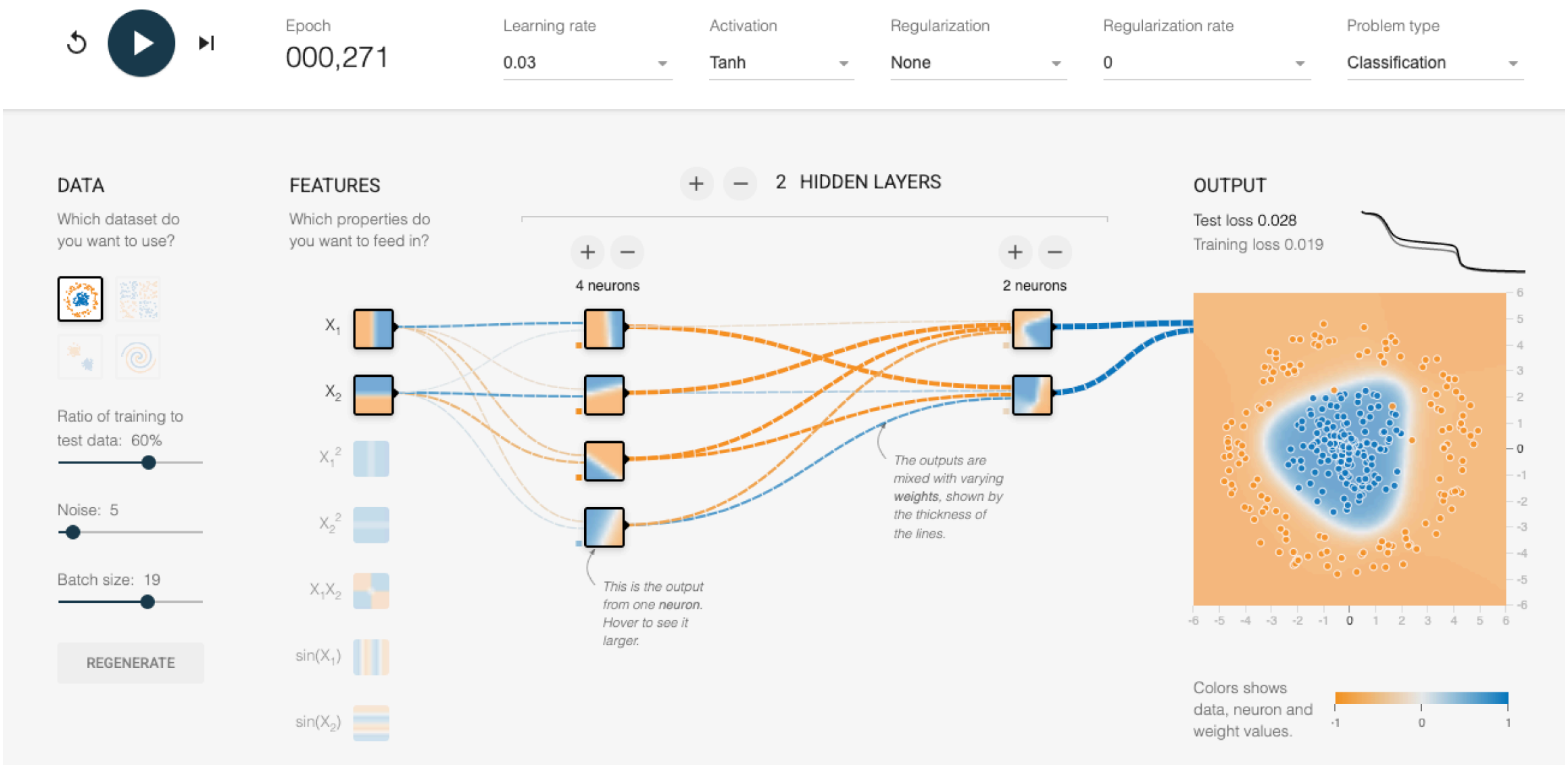
Shallow network can solve many
problems with enough neurons



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?
5 HIDDEN LAYERS
8 neurons
6 neurons
4 neurons
3 neurons
1 neuron
The outputs are mixed with varying weights, shown by the thickness of the lines.
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

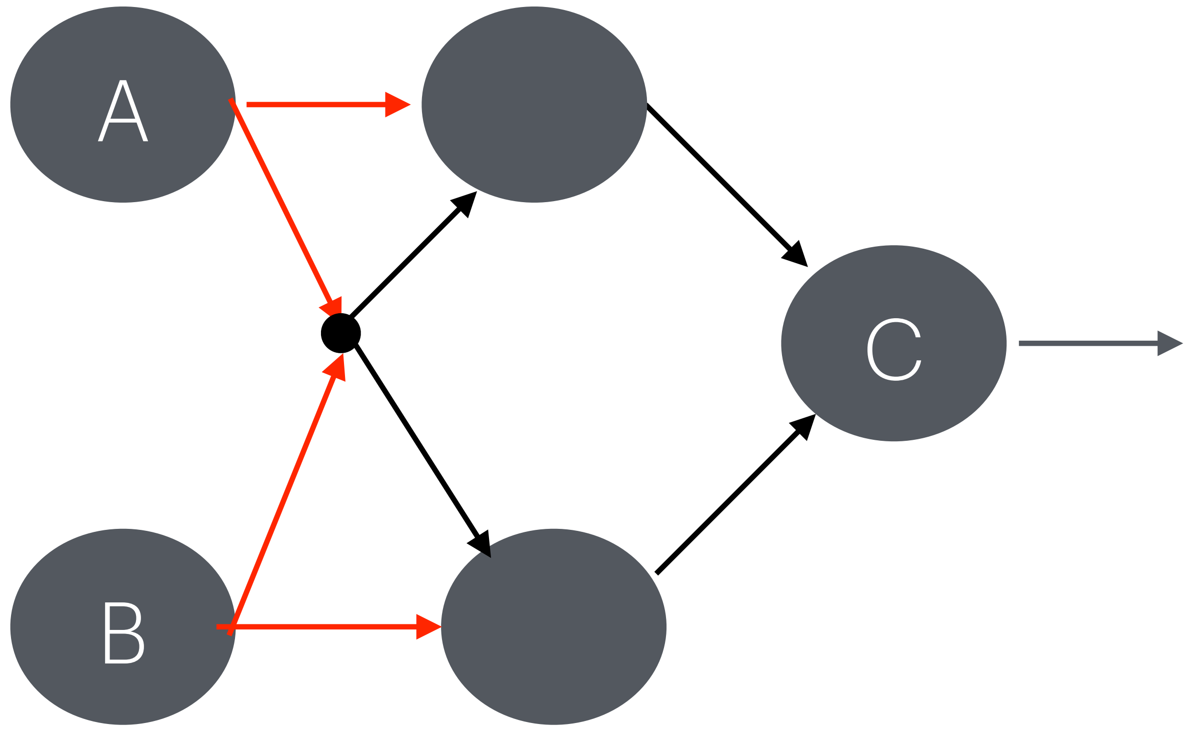


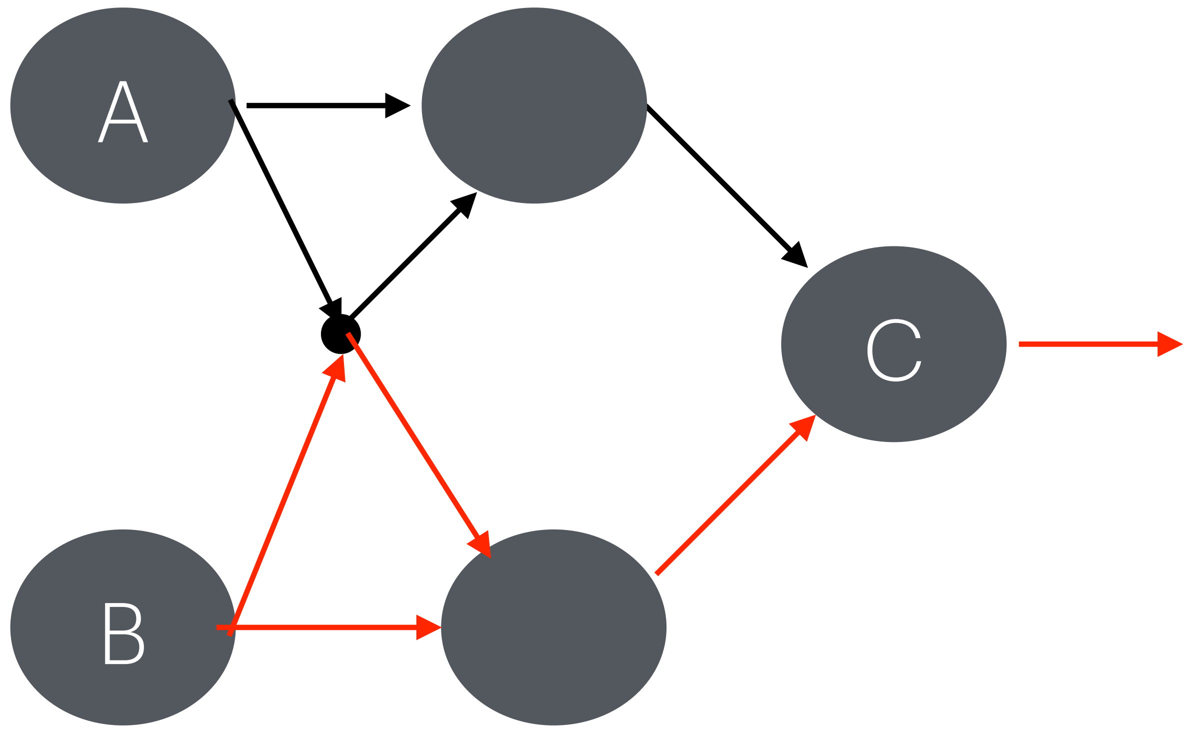
TANU takes time to converge on a solution

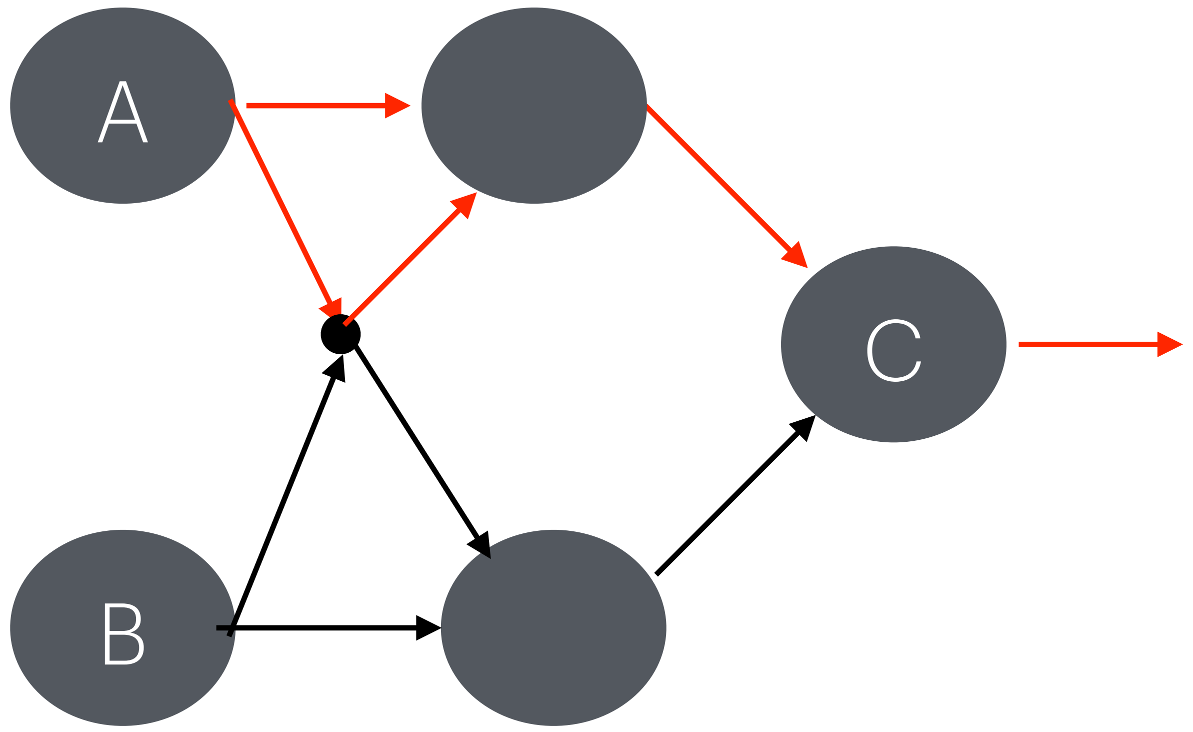
Exercise 2

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

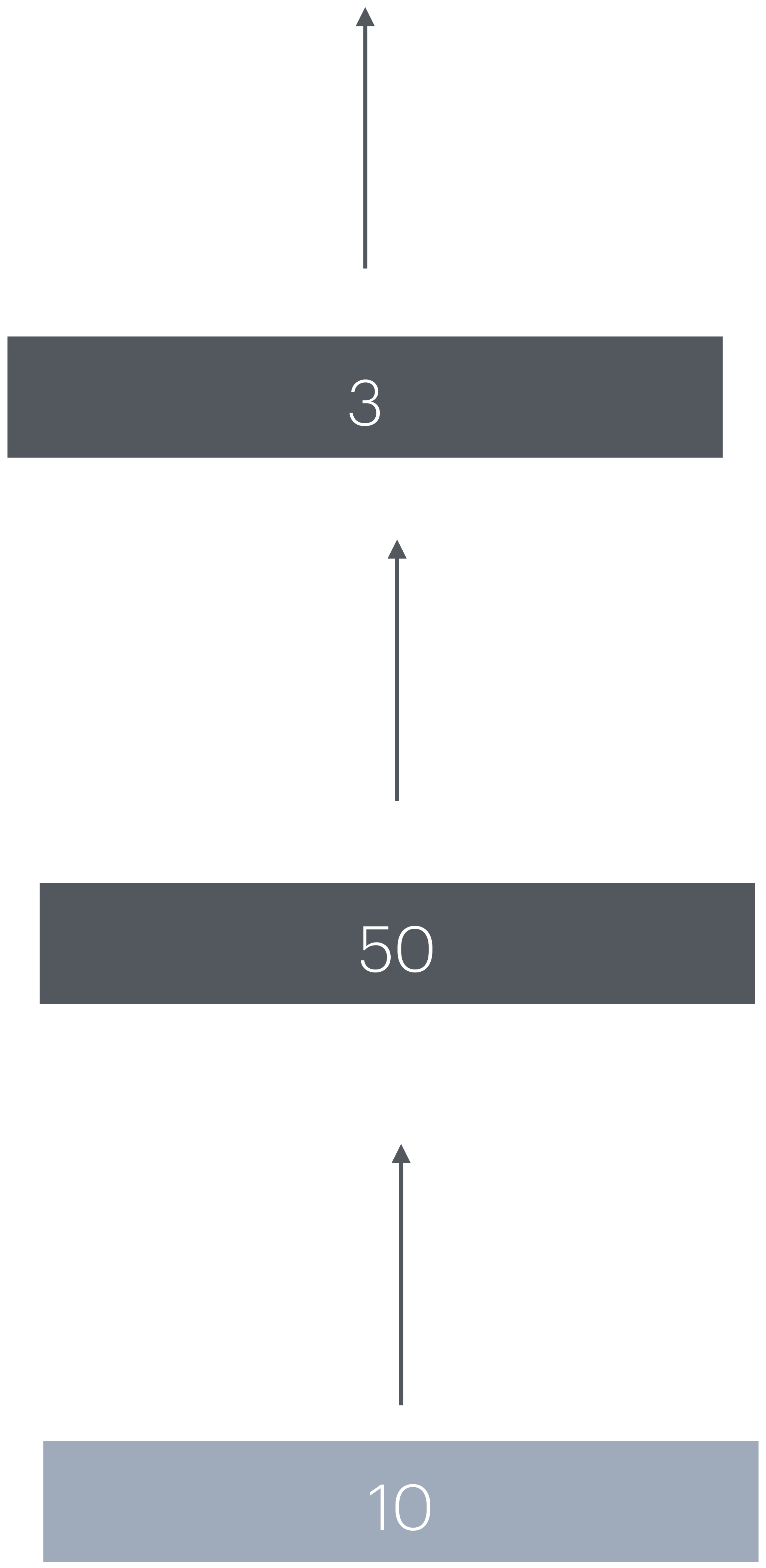
$(A \bigcup \neg B) \bigcup (\neg A \bigcup B)$







Exercise 3



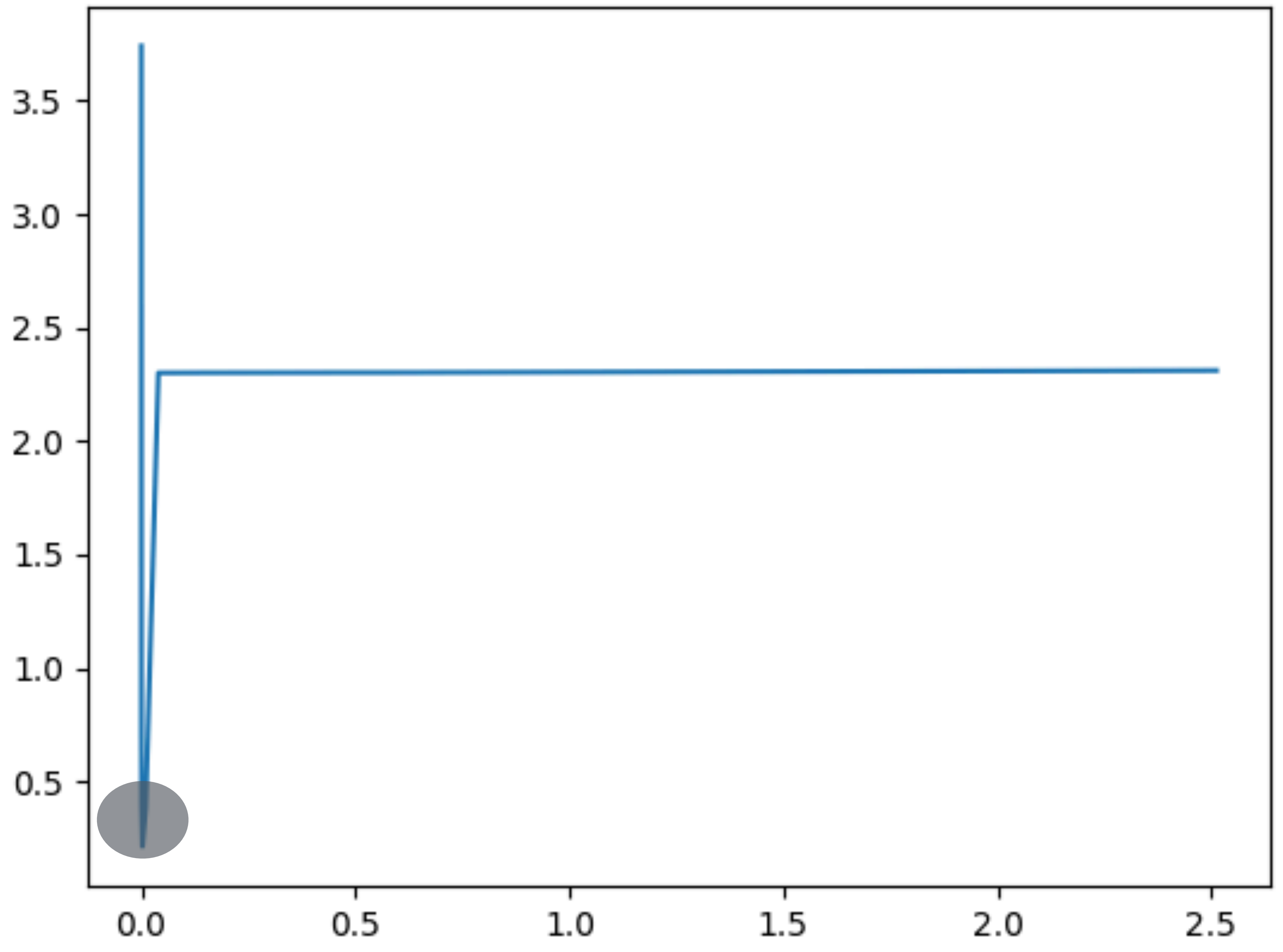
$$\mathbf{Y} = 1 \times 3$$

$$\mathbf{W_o} = 50 \times 3 \qquad \mathbf{b_o} = 1 \times 3$$

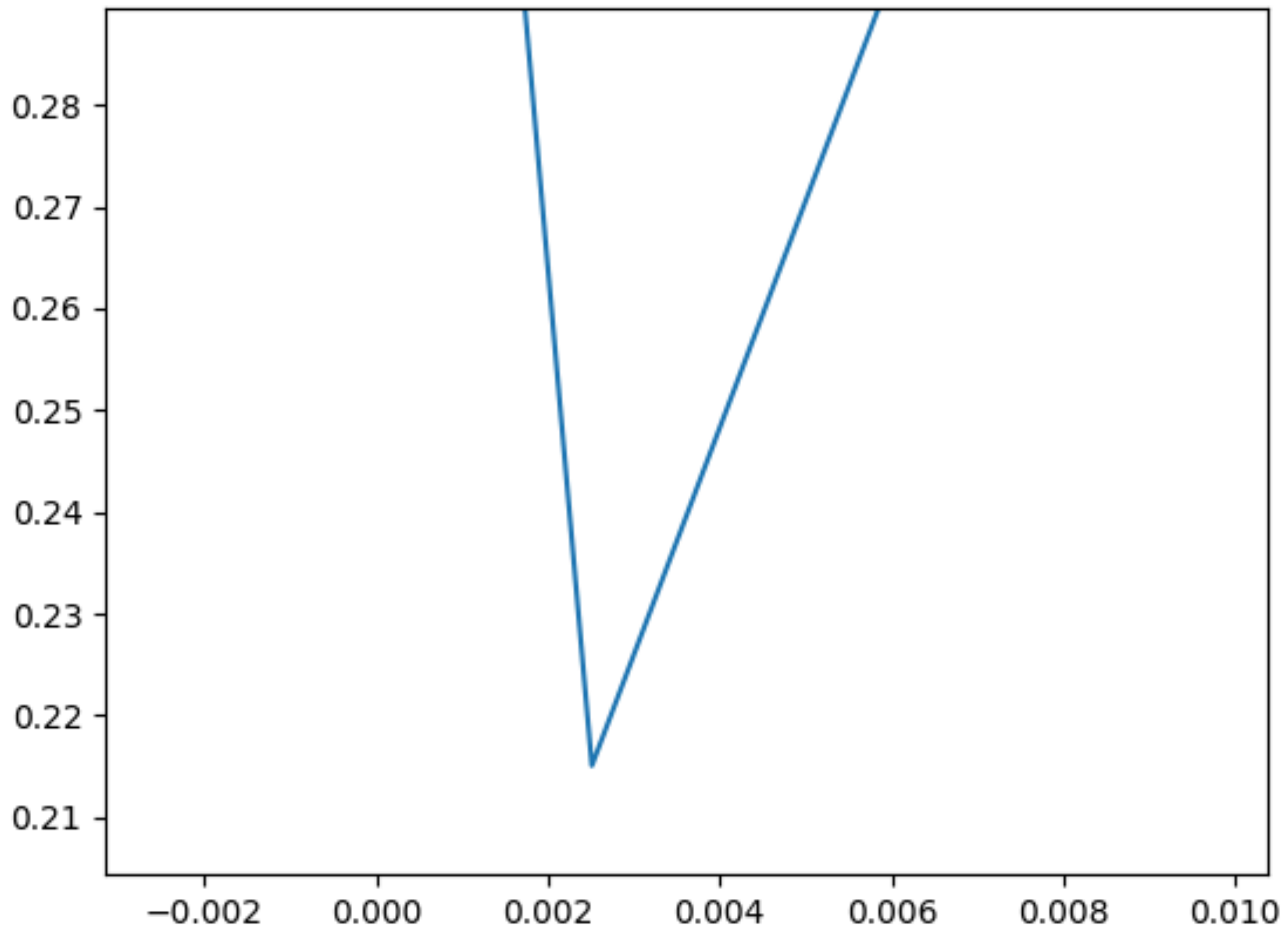
$$\mathbf{W_h} = 10 \times 50 \qquad \mathbf{b_h} = 1 \times 50$$

$$\mathbf{X} = N \times 10$$

Find Learning Rate



Gradually increase learning rate



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