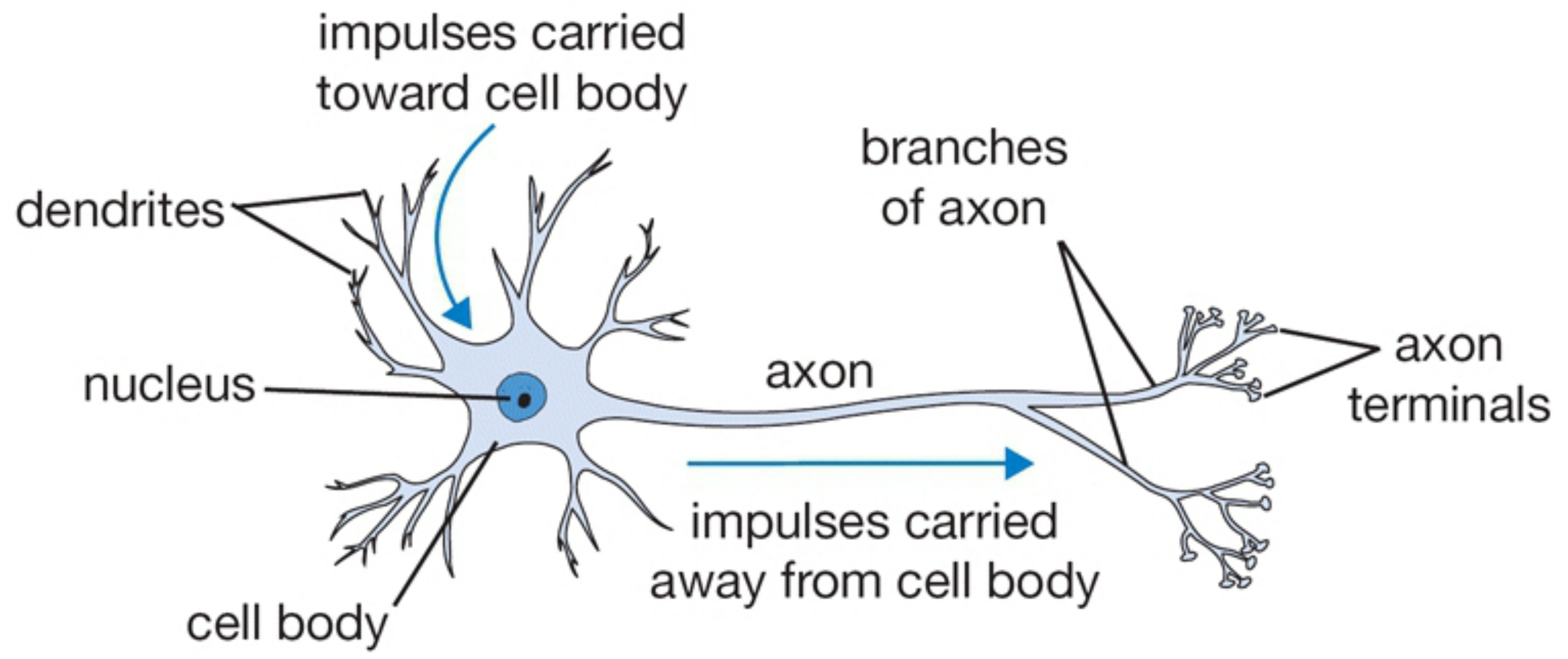
Neural network

Insop

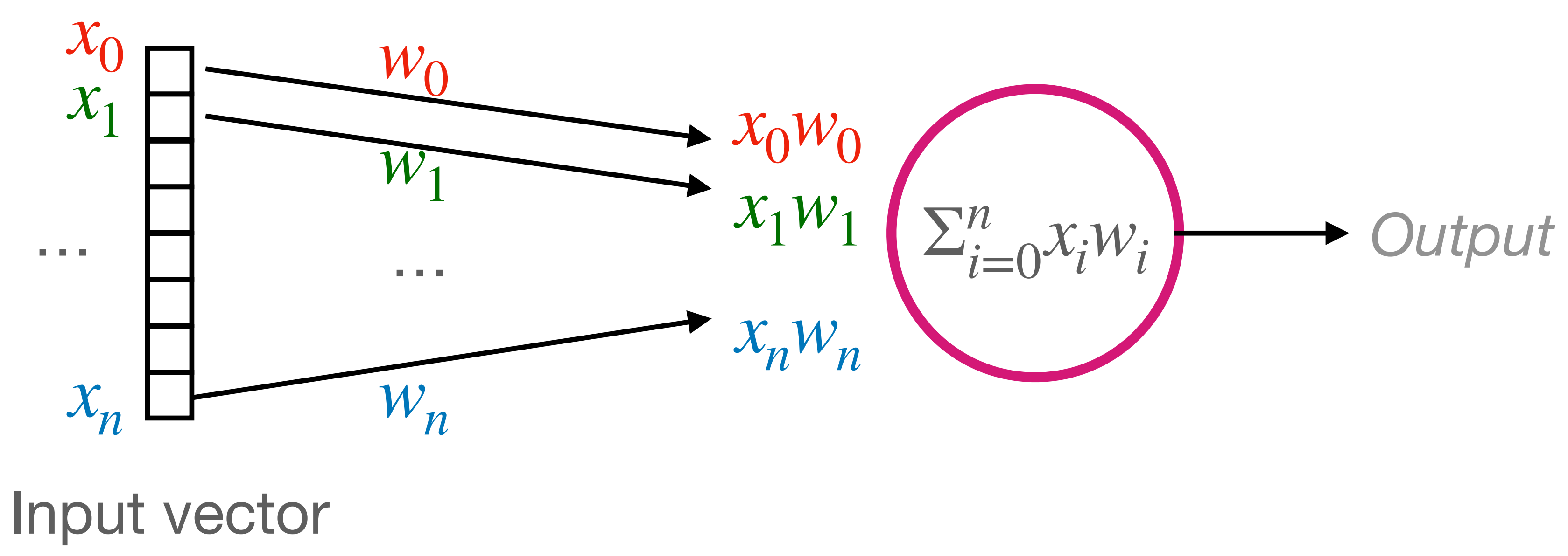
Perceptron

Biological motivation

- Perceptron: (*roughly*) biologically motivated model
- Takes sum of the weighted inputs
- Apply non-linear operation to the sum to determine output



Source: link



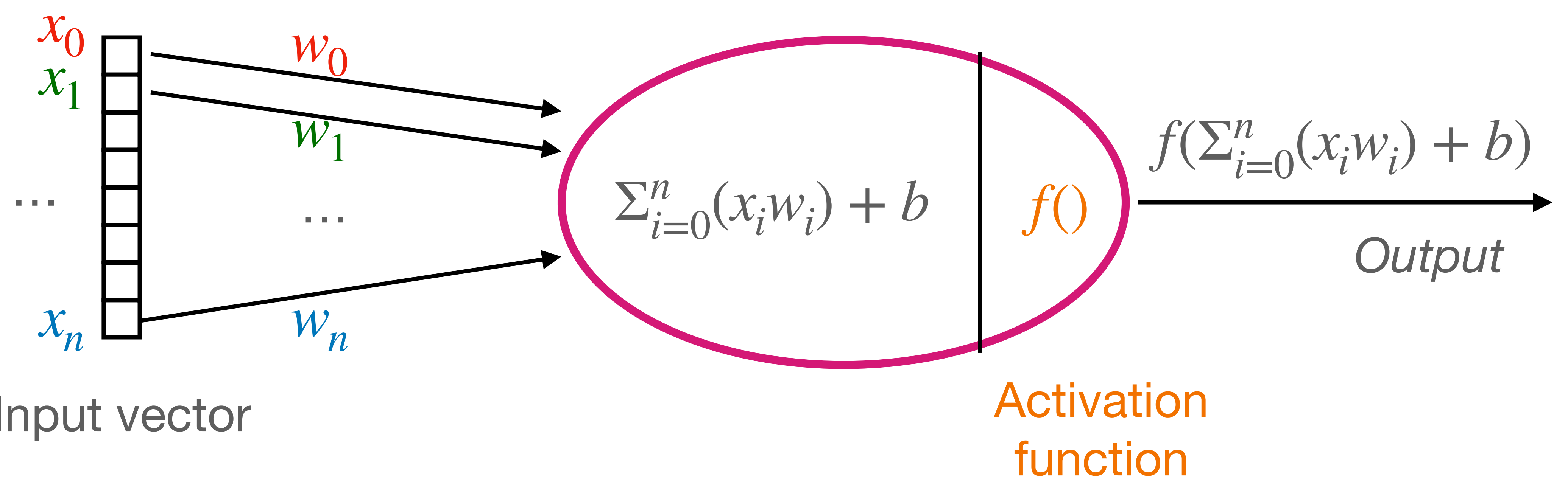
output= 0 if $\Sigma_{i=0}^{n} x_i w_i \leq$ threshold

= 1 if $\Sigma_{i=0}^{n} x_i w_i \geq$ threshold

Single neuron

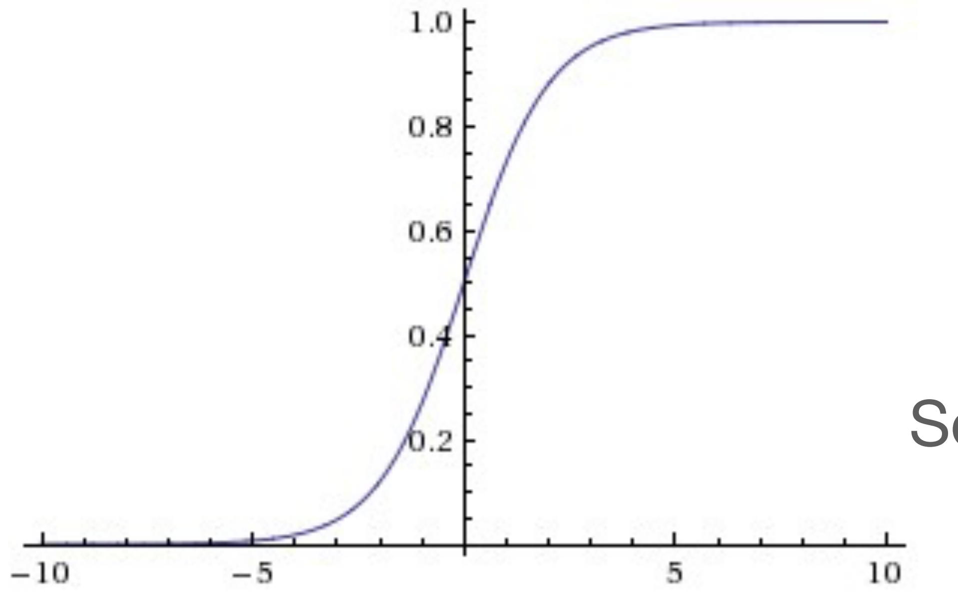
Neuron of Neural network

- More generic form: added bias (b)
- Use non-linear activation function $f()$
    - Types of *non-linear functions*: sigmoid, tanh, *ReLU*



$$[w_0 x_0 + w_1 x_1 + .. + w_n x_n] = [w_0 \quad w_{01} \quad \dots \quad w_n] \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix}$$

Activation function: Sigmoid function

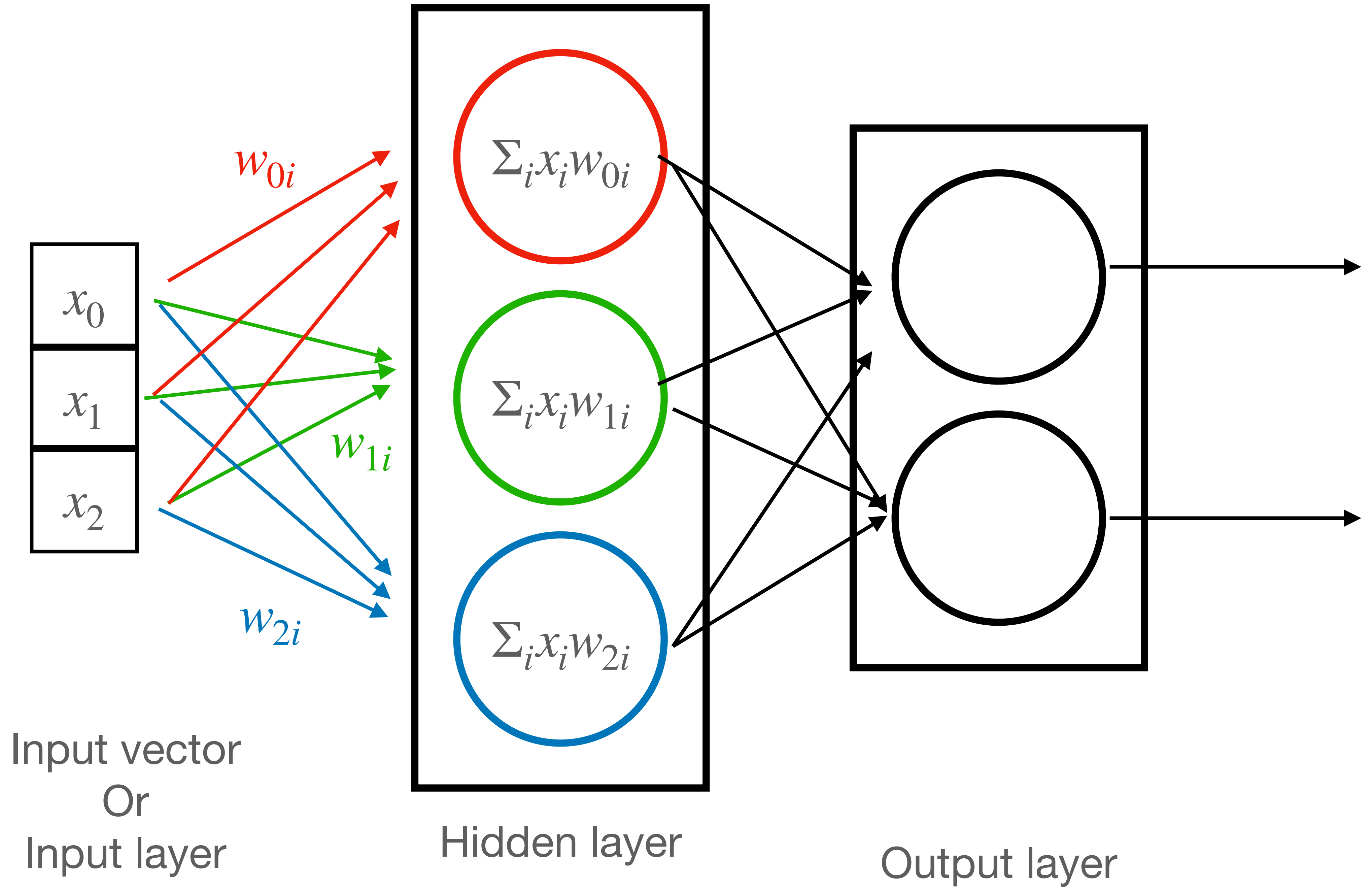


Source: link

```
class Neuron(object):
  def forward(self, inputs):
    input_weighted_sum = np.dot(input, weights) + bias
    activation_output = f(input_weighted_sum)
    return activation_output
```

Neural network

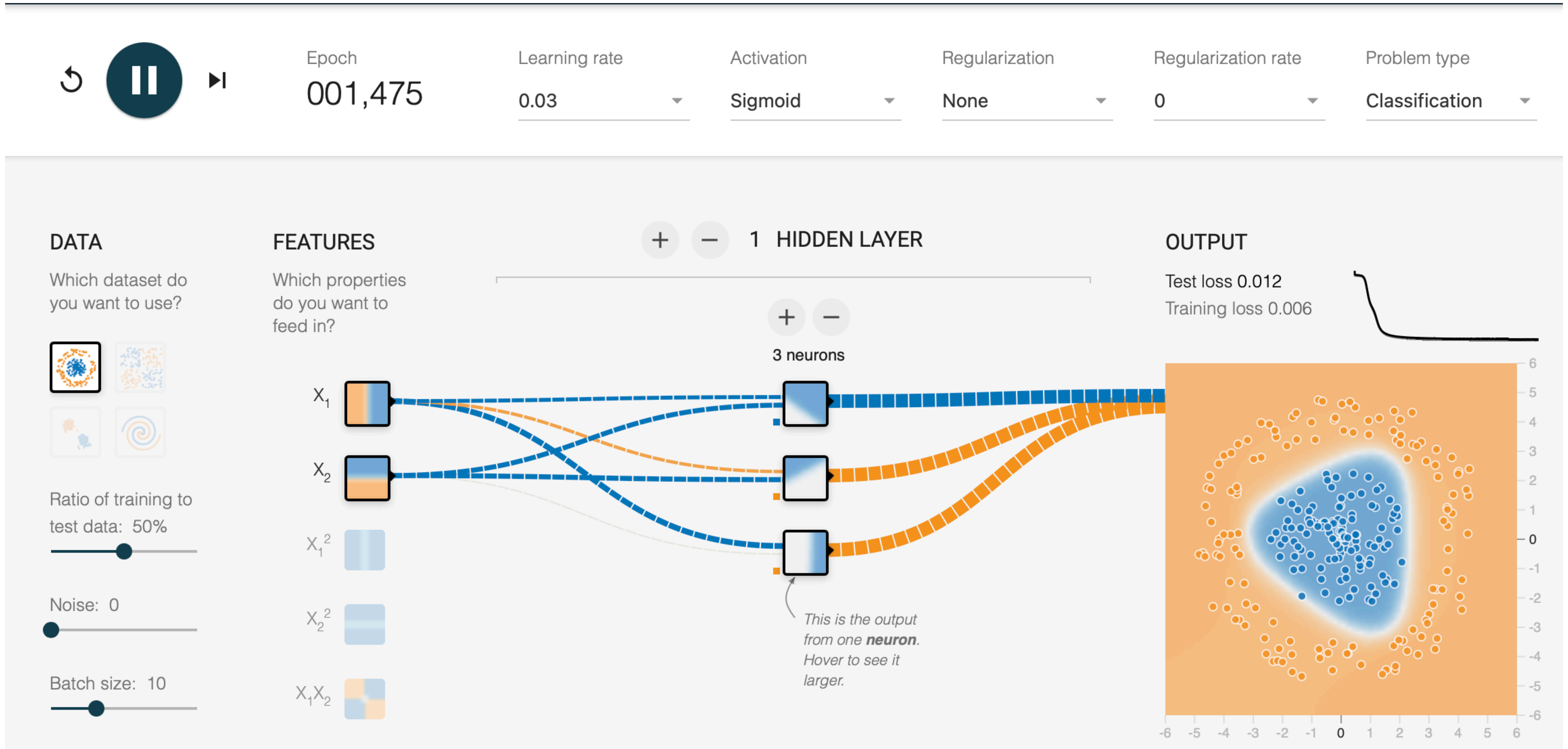
- Layer: multiple neurons
- Multiple layers: connecting many layers
- Neurons in each layers are connected
  - Also called FC (fully connected) layers or MLP (multi-layer perceptron)

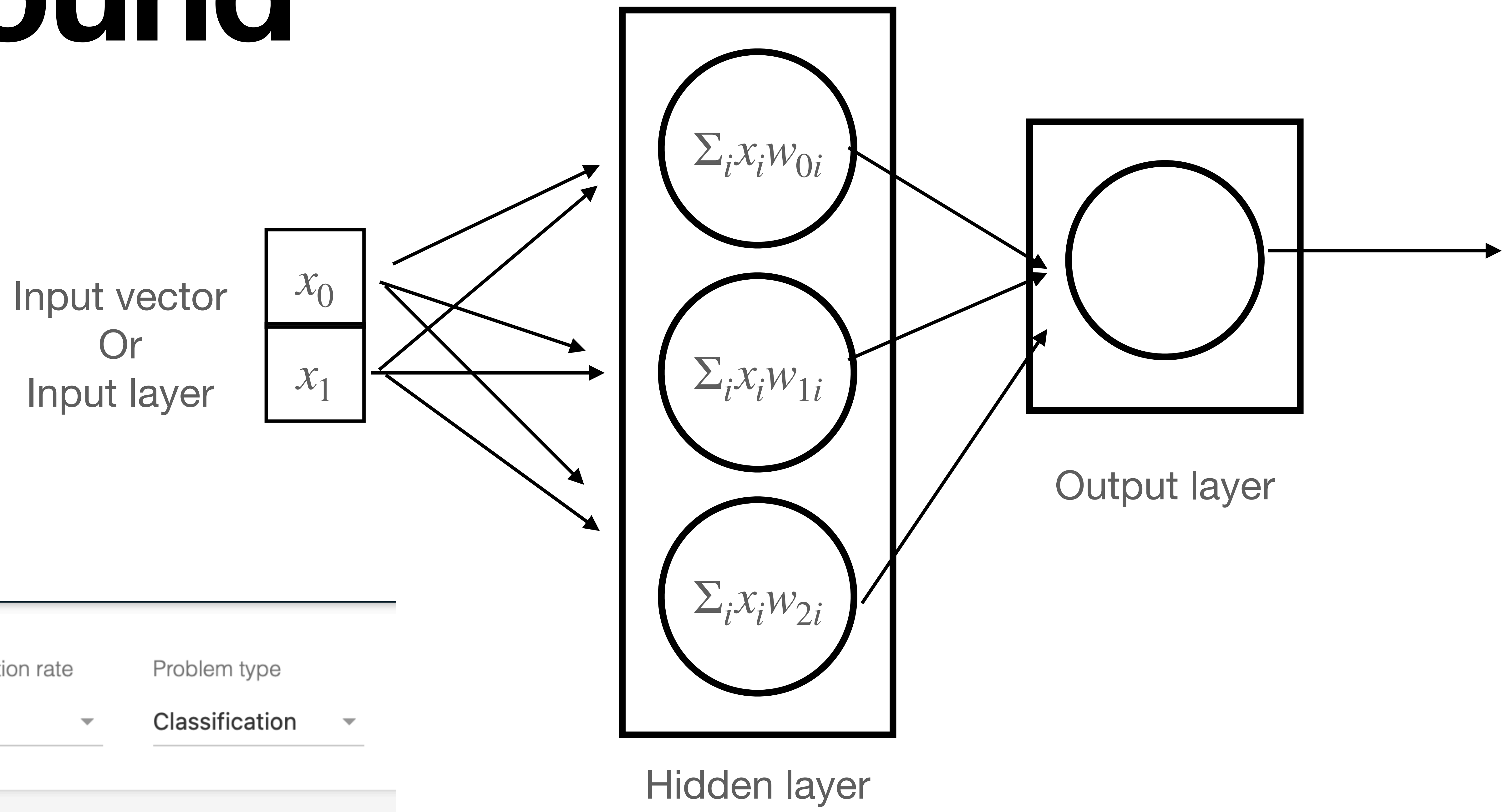


$$\begin{bmatrix} \boxed{w_{00}x_0 + w_{01}x_1 + w_{02}x_2} \\ \boxed{w_{10}x_0 + w_{11}x_1 + w_{12}x_2} \\ \boxed{w_{20}x_0 + w_{21}x_1 + w_{22}x_2} \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

Neural network playground

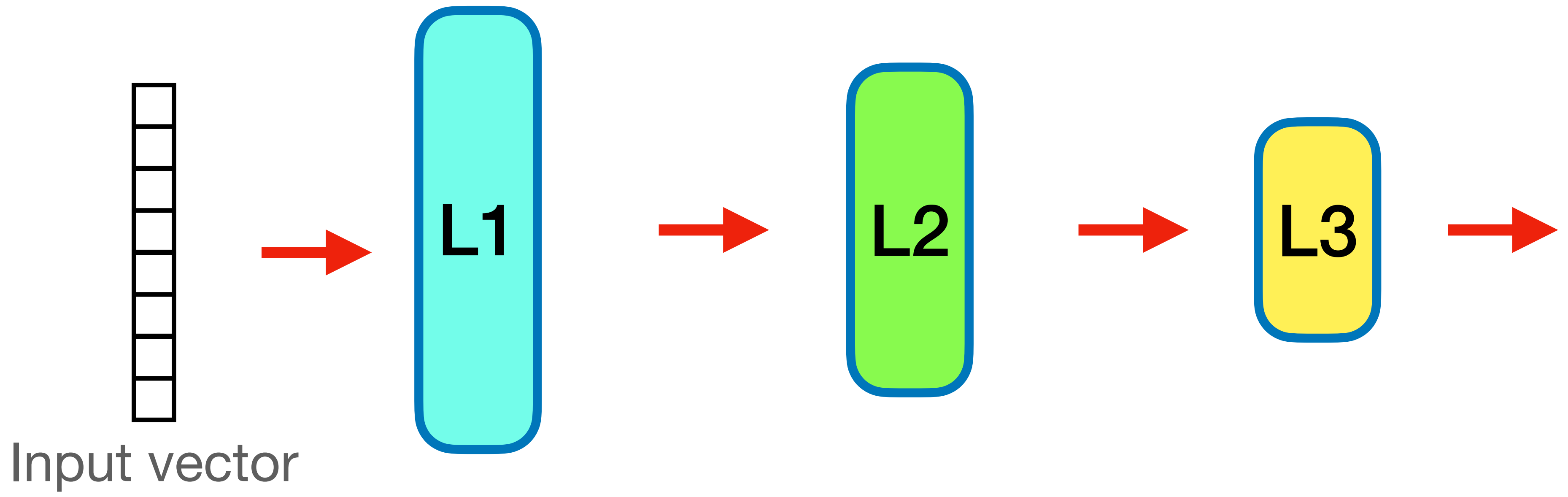
- Great way to gain intuition
- http://playground.tensorflow.org





Simple neural network example

- Simple 3 layers neural network
- In each layer:
  - Input: vector
  - Output: vector
  - Learned parameters: matrix
  - Operations:
    - Multiply the input vector with weight matrix
    - Apply element-wise non-linear operations: ReLU *



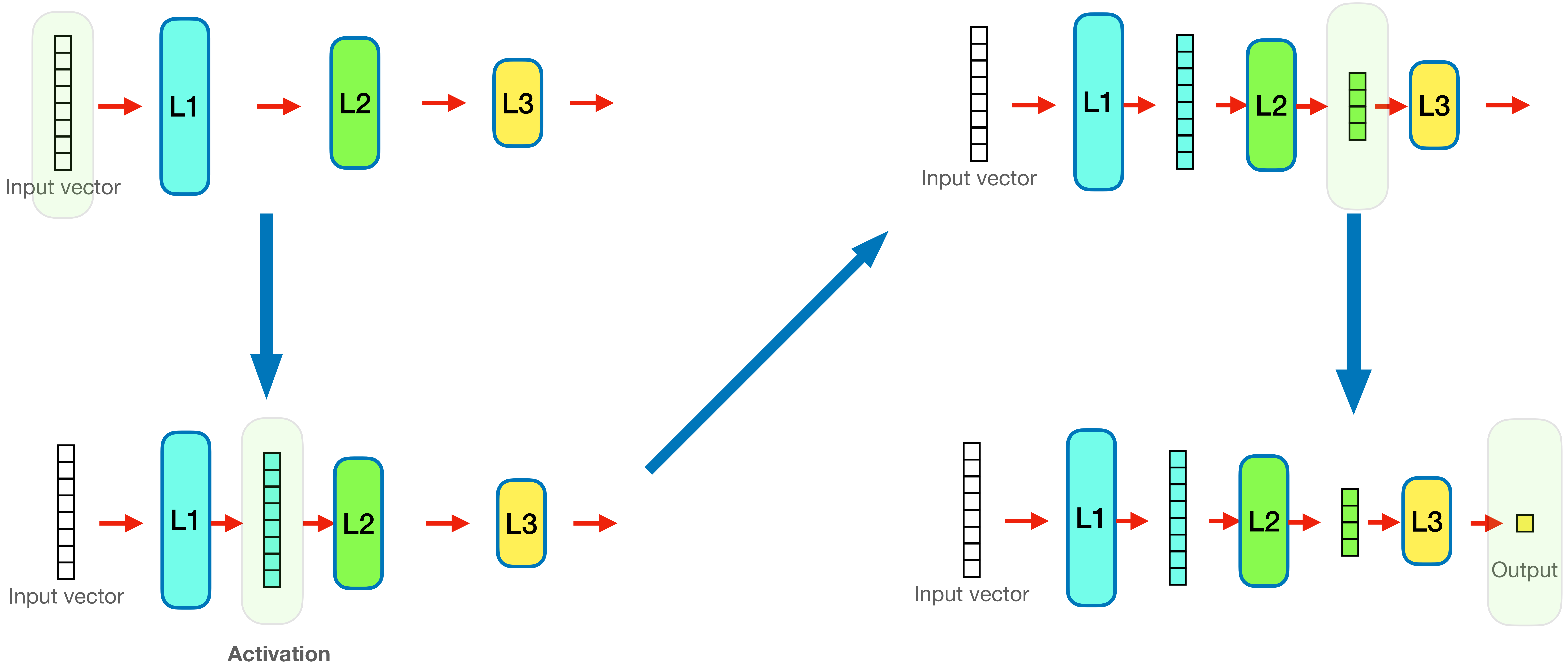
- Neural network training
  - Forward propagation
  - Backward propagation
  - Weight update

*Other types of non-linear ops: sigmoid, tanh

Source: Fundamentals of Scaling Out DL Training

Forward propagation

Example using a single input

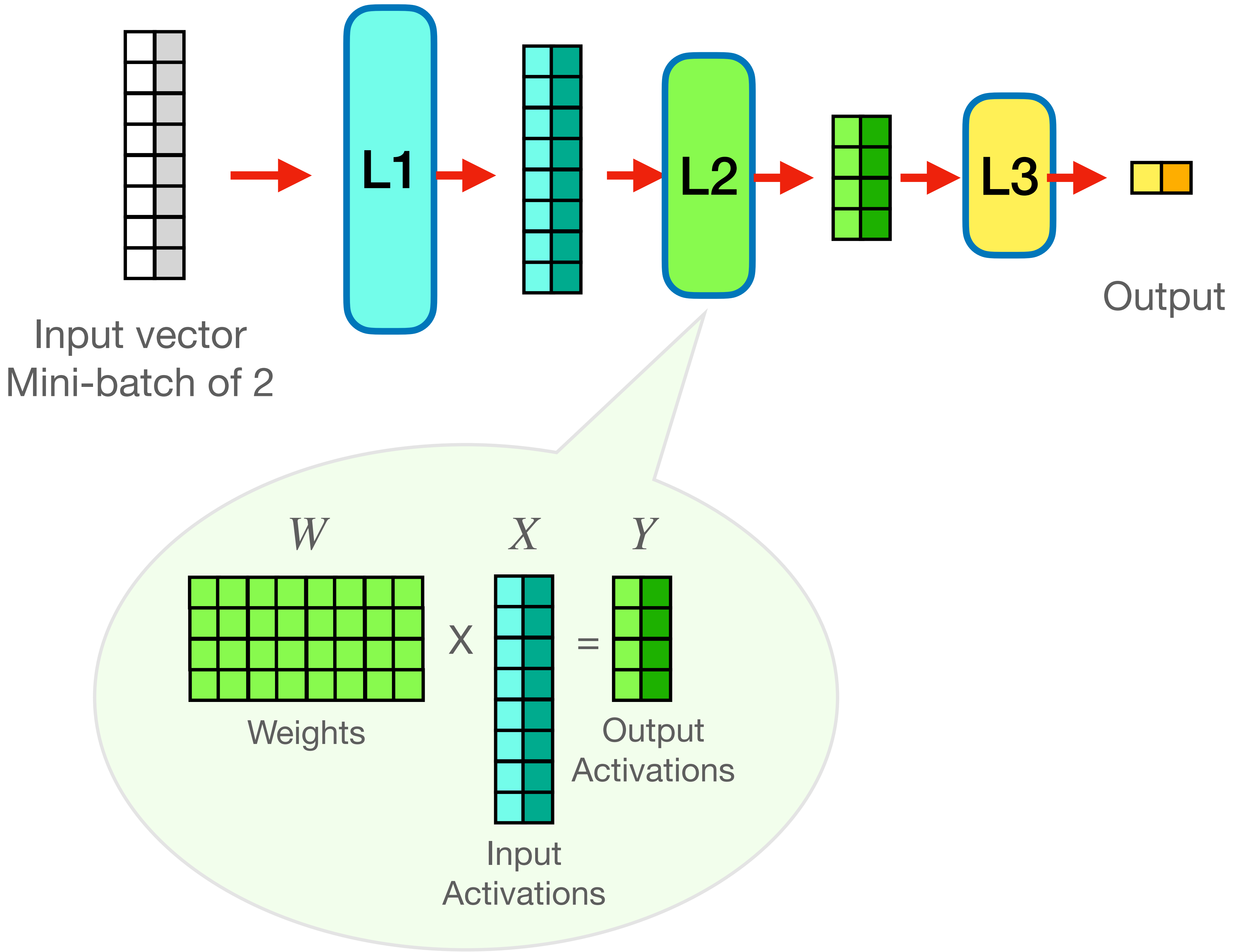


Source: Fundamentals of Scaling Out DL Training

Forward propagation

Mini-batch of 2 input vectors

- For mini-batch inputs
- Matrix-vector multiplication becomes matrix-matrix multiplication

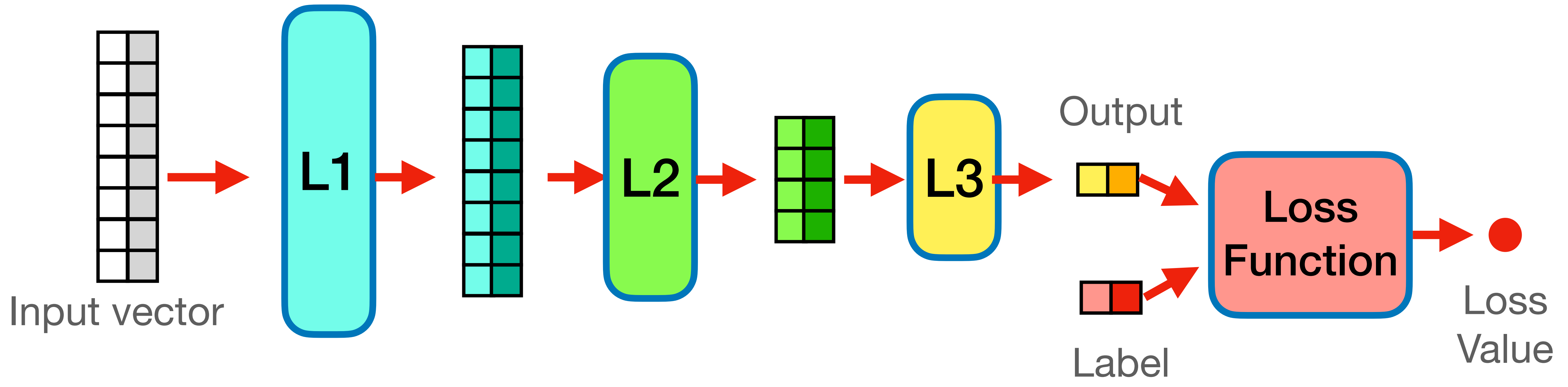


Source: Fundamentals of Scaling Out DL Training

Forward propagation

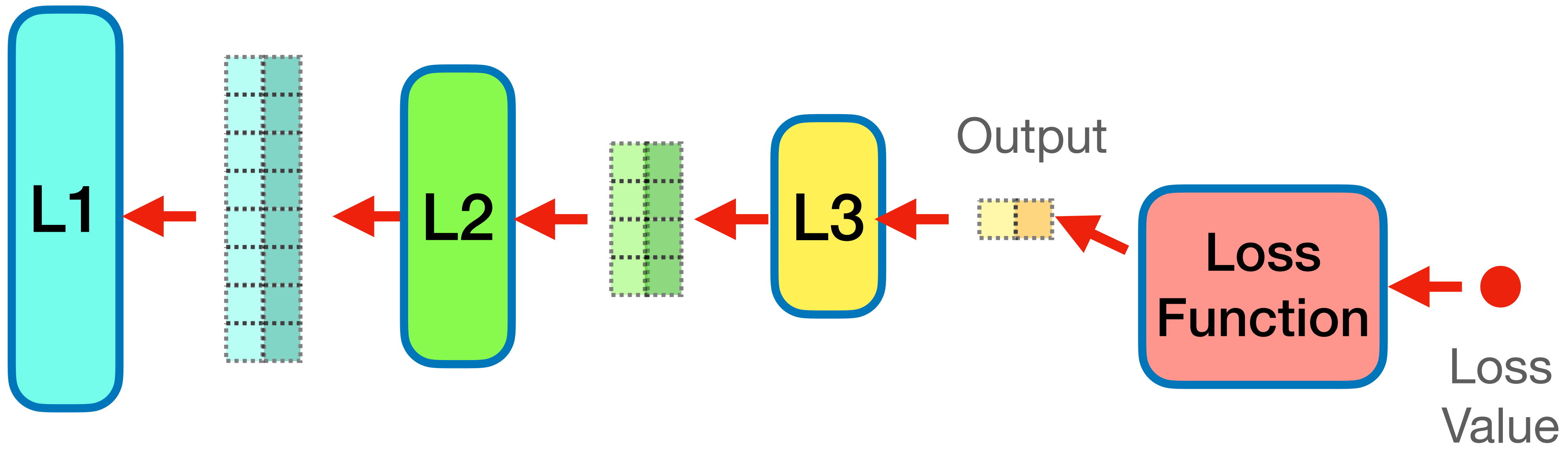
Compute loss

- Loss function: measure how ‘bad’ the neural network was
    - Compare the output to the label for each input
- Goal of the training: minimize the loss value
    - Update the weights to reduce the loss
    - Output will be close to the labels



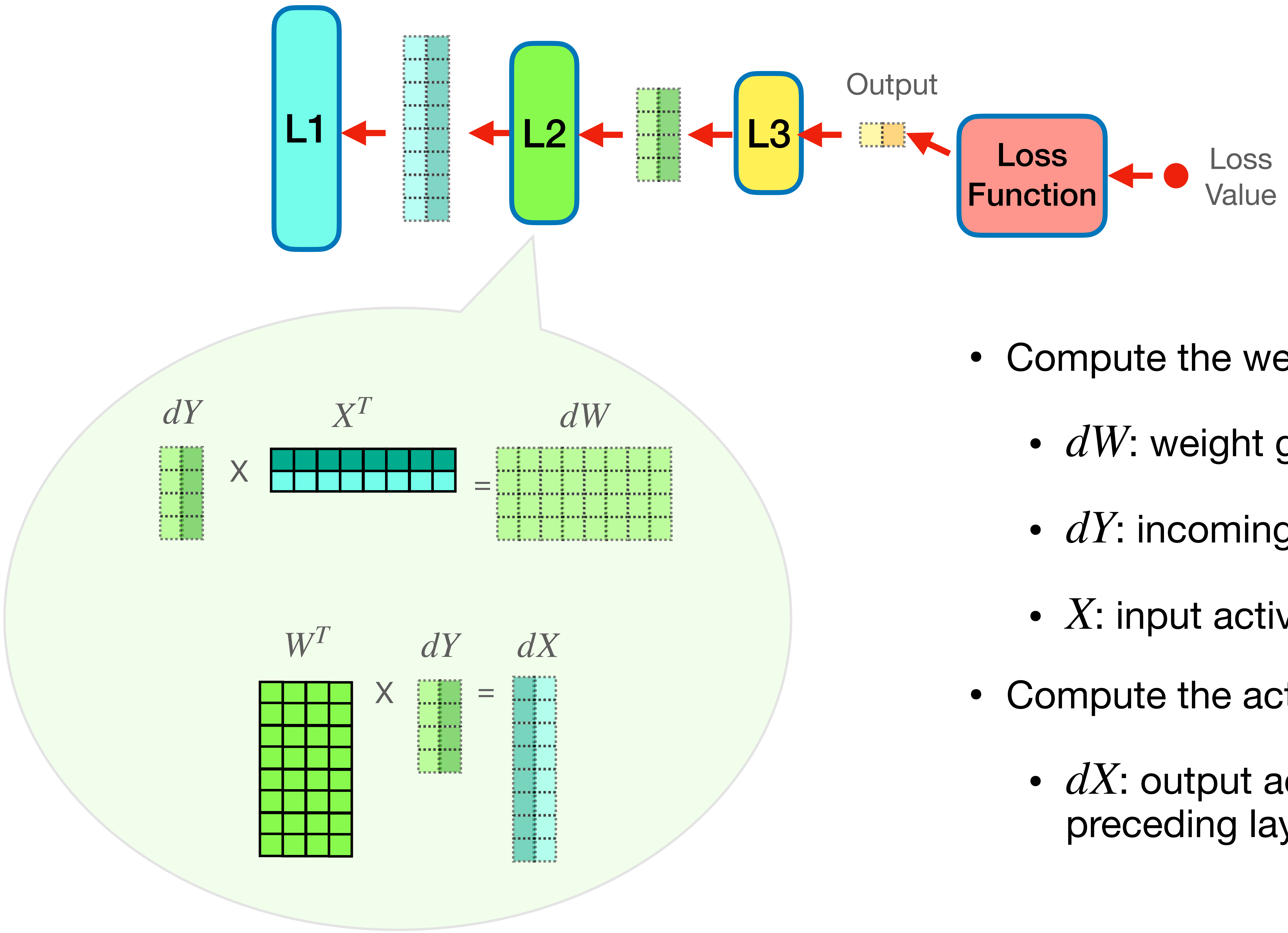
Backward propagation

- Goal: update the layer weights to minimize loss value
- Can be done by “backward propagating” loss through the layers
  - Each layer computes weight gradient, used to update the weights
  - Each layer computes activation gradient, to be back propagated to preceding layer



Source: Fundamentals of Scaling Out DL Training

Backward propagation



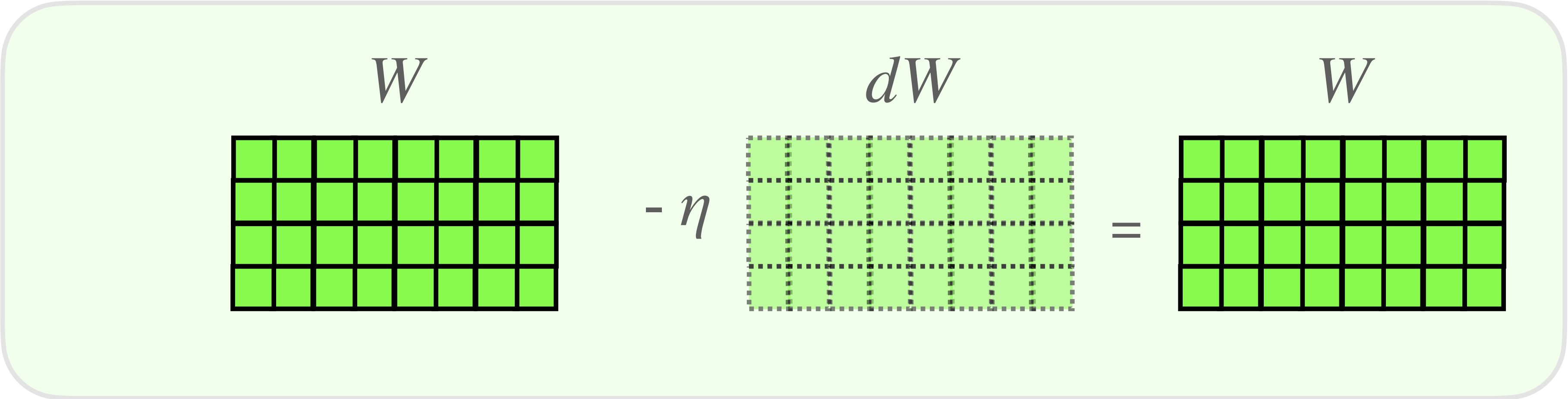
- Compute the weight gradient
  - dW: weight gradient (to update weights)
  - dY: incoming activation gradient
  - X: input activation (from forward propagation)
- Compute the activation gradient
  - dX: output activation gradient to back propagate to the preceding layer

Source: Fundamentals of Scaling Out DL Training

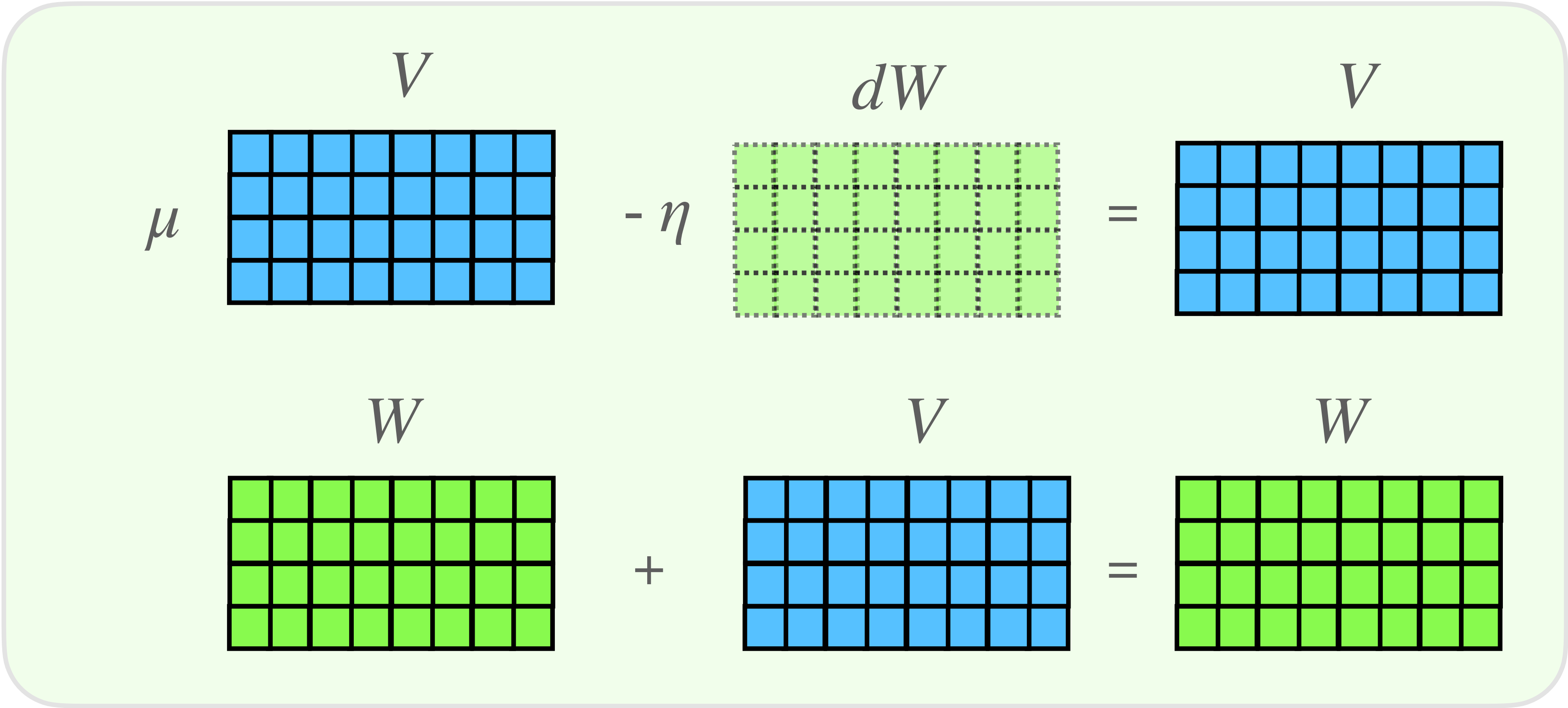
Weight update

- Optimization step
  - Types of optimization: SGD, Adam, Adagrdad, rmsprop
- Input:
  - Current network weights
  - Weight gradient from backward prop.
- Output: updated weights
- Operations:
  - Update each weight with corresponding weight gradient value
  - Advanced methods:
    - Maintain internal optimization state and use this state to update the weight
- Internal states of advanced method
  - 1 or 2 momentum
  - Each momentum is the same size as weight
    - Size of the state may need 2-6 times as the weight size

SGD: stochastic gradient descent



SGD with momentum, advanced method

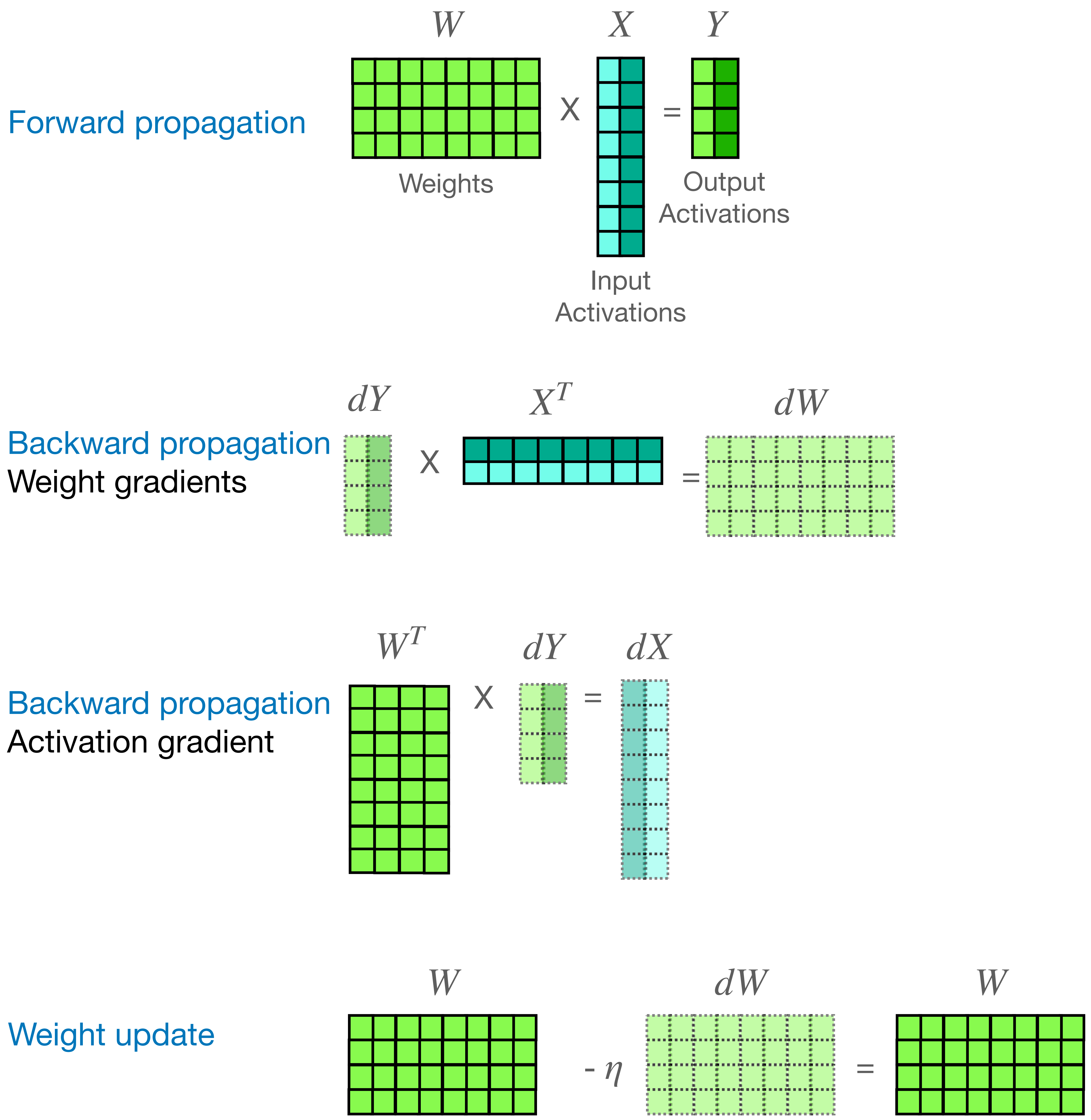


η: learning rate, also called step size

Source: Fundamentals of Scaling Out DL Training

Training summary

- Backward compute does ~2x of forward compute
- Backward propagation uses the activations computed during the forward propagation
  - X in the example, and this is computed by a preceding layer
  - This is a major fraction of memory used in training



- Example: Resnet50 training in FP16 at mini-batch size 256:
  - Requires ~ 15GB of memory
  - ~12 GB of the memory is used by activations

Source: Fundamentals of Scaling Out DL Training

Simple PyTorch neural network

- Open in Colab

References

- Excellent Neural network introduction book: Neural Networks and Deep Learning
- Simple neural network implementation
- Excellent neural network tutorial