Neural Networks

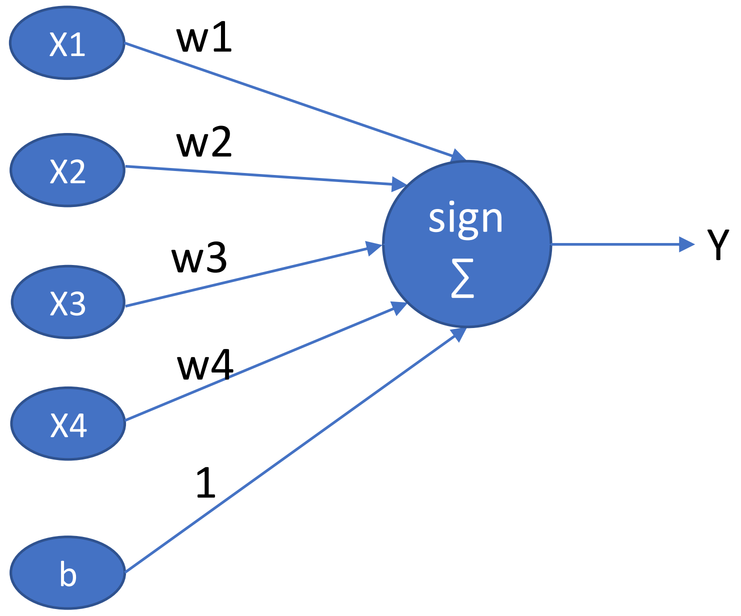
MLFA

Linear classifier as Neural Network

- A linear classifier computes a weighted sum of the features and a bias
- Then runs the "sign" function on the result
- $Y = \text{sign}(w.x + b)$

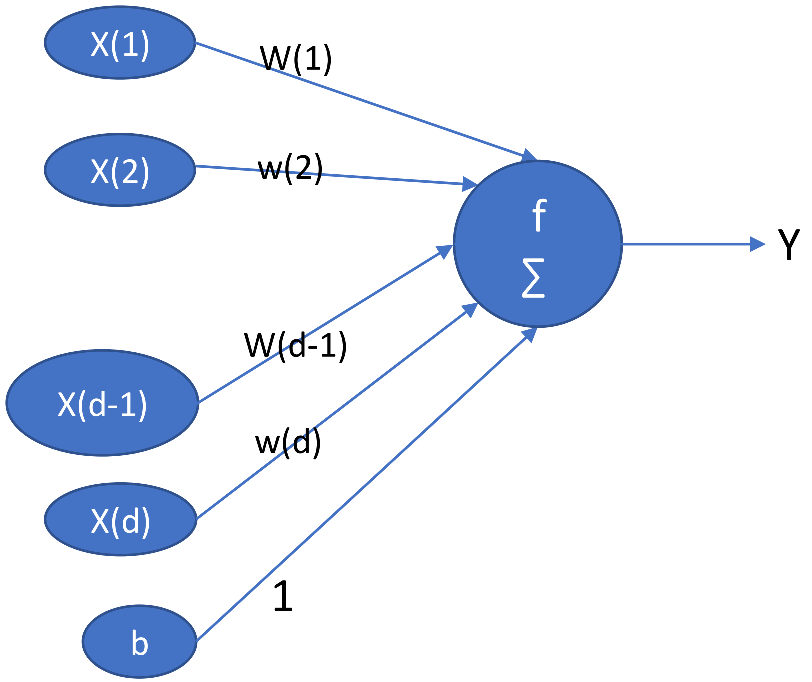
Linear classifier as Neural Network

- A linear classifier computes a weighted sum of the features and a bias
- Then runs the “sign” function on the result
- Y = sign(w.x + b)
- We can represent this by a graph
- Each input dimension: one input node
- Each input node connected to output node
- Each connection edge carries a weight



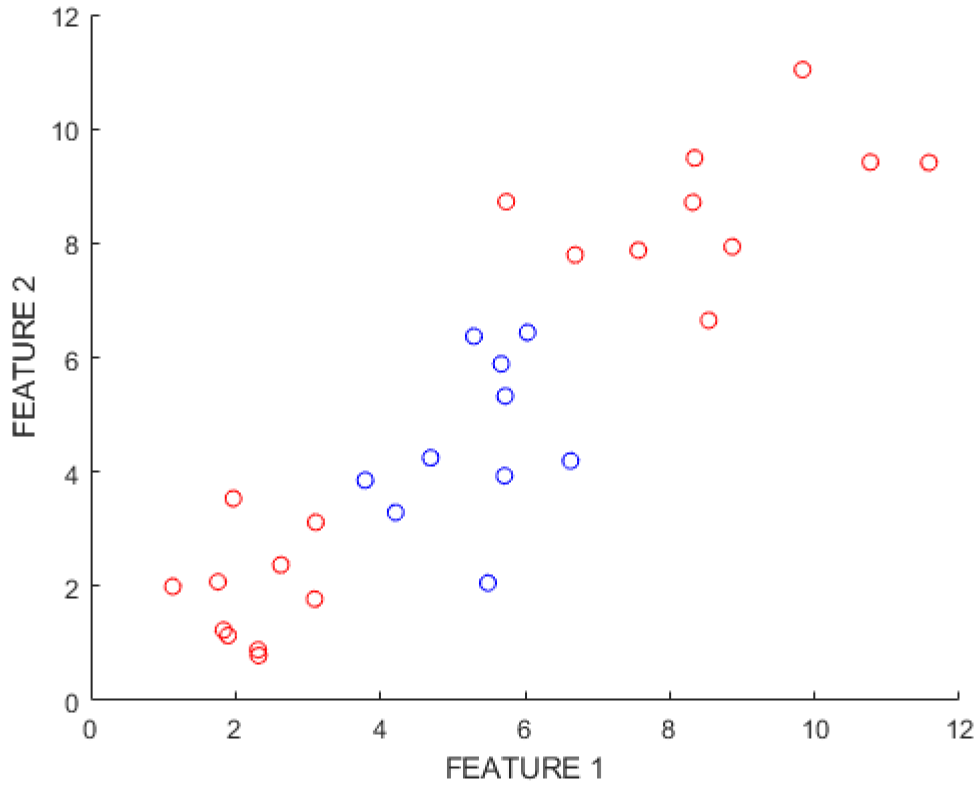
Non-Linear classifier as Neural Network

- A linear classifier computes a weighted sum of the features and a bias
- Then runs the "sign" function on the result
- Y = f(w.x + b), where f is non-linear function
- We can represent this by a graph
- Each input dimension: one input node
- Each input node connected to output node
- Each connection edge carries a weight



Multi-linear classifier

- A single Linear classifier is often not enough!

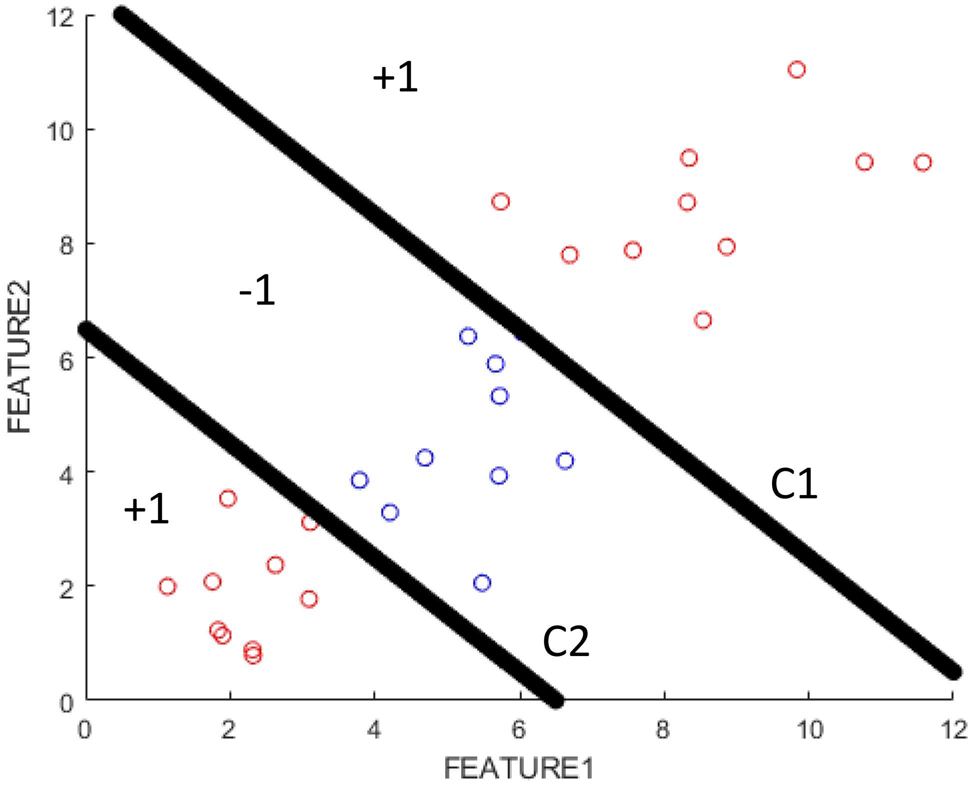


Multi-linear classifier

- A single Linear classifier is often not enough!
- We may need a combination of linear classifiers!

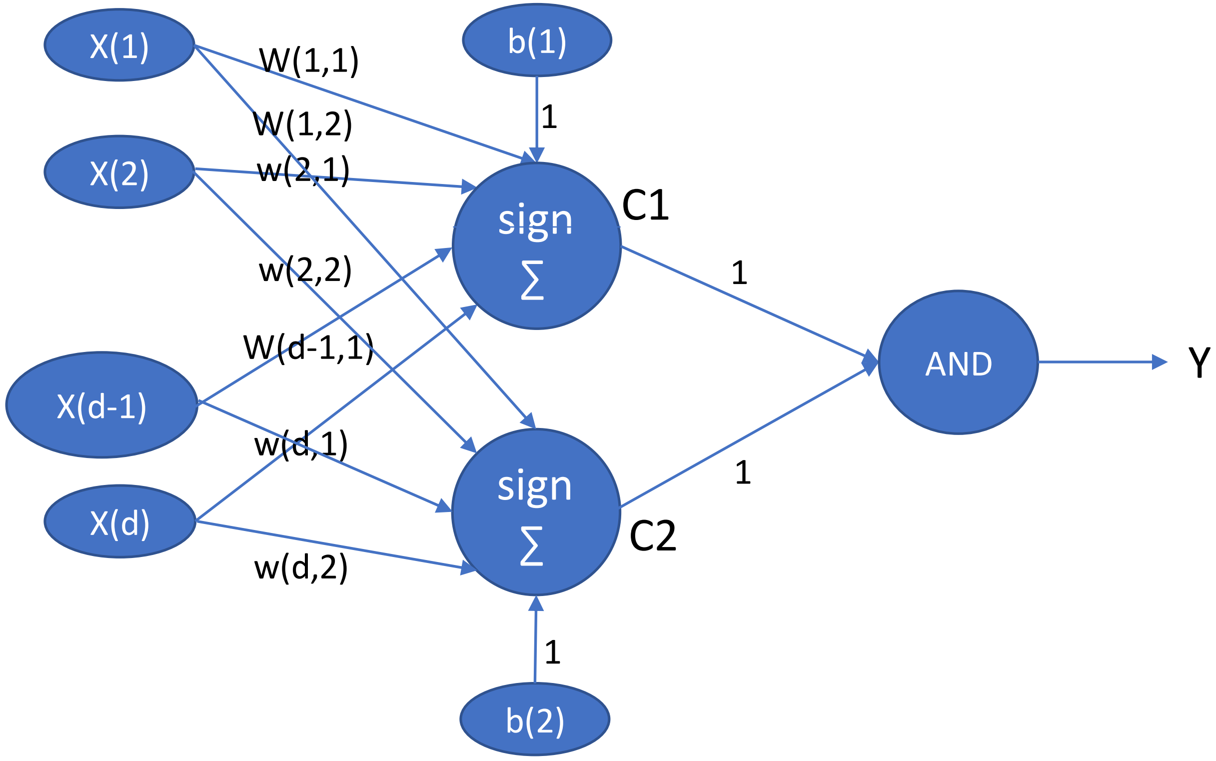
If C1:+1 OR C2:+1, Final:+1

If C1:-1 AND C2:-1, Final:-2



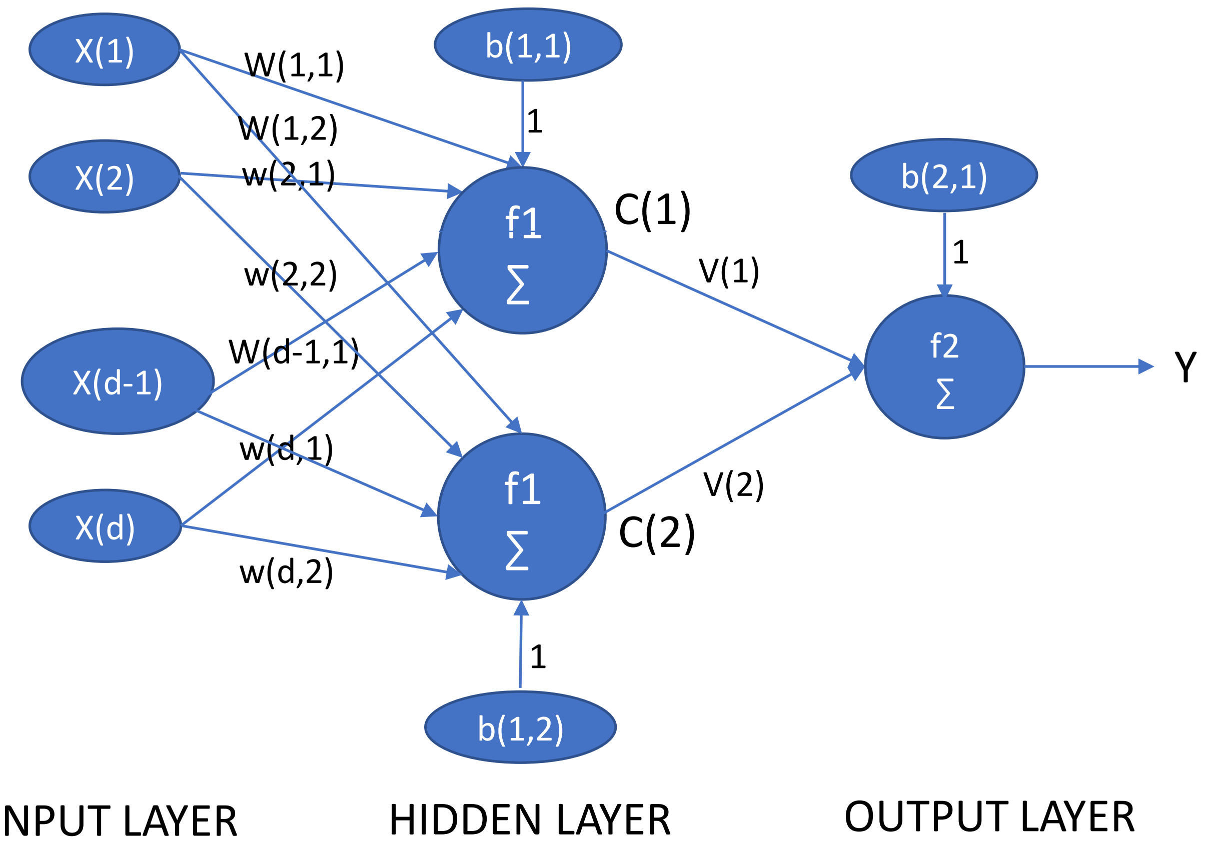
Multi-layer Neural Network

- We fuse both linear classifiers in the same neural network
- Multi-layer Neural Network!
- $C1=sign(w_1.x+b_1)$
- $C2=sign(w_2.x+b_2)$
- Y=C1 AND C2

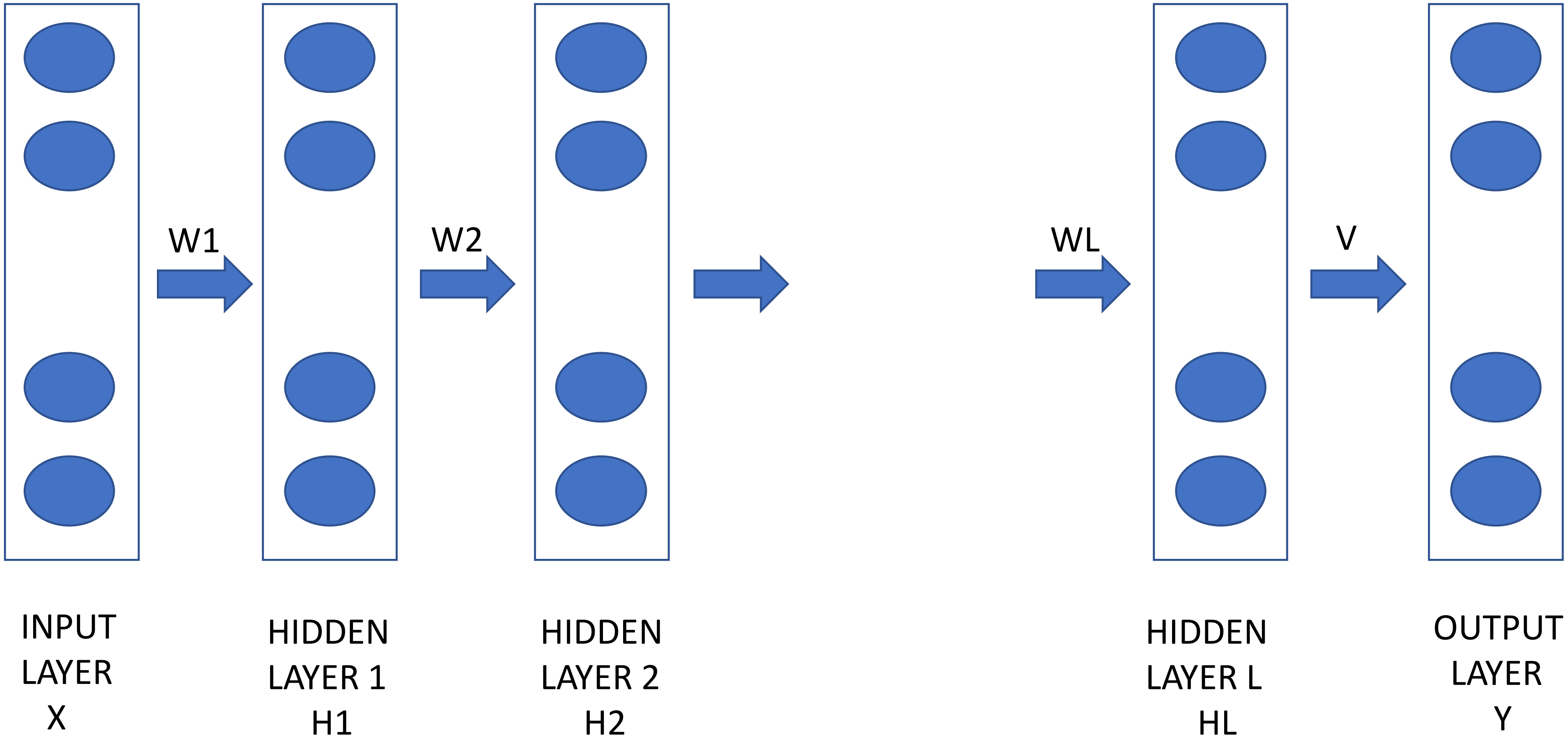


Multi-layer Neural Network

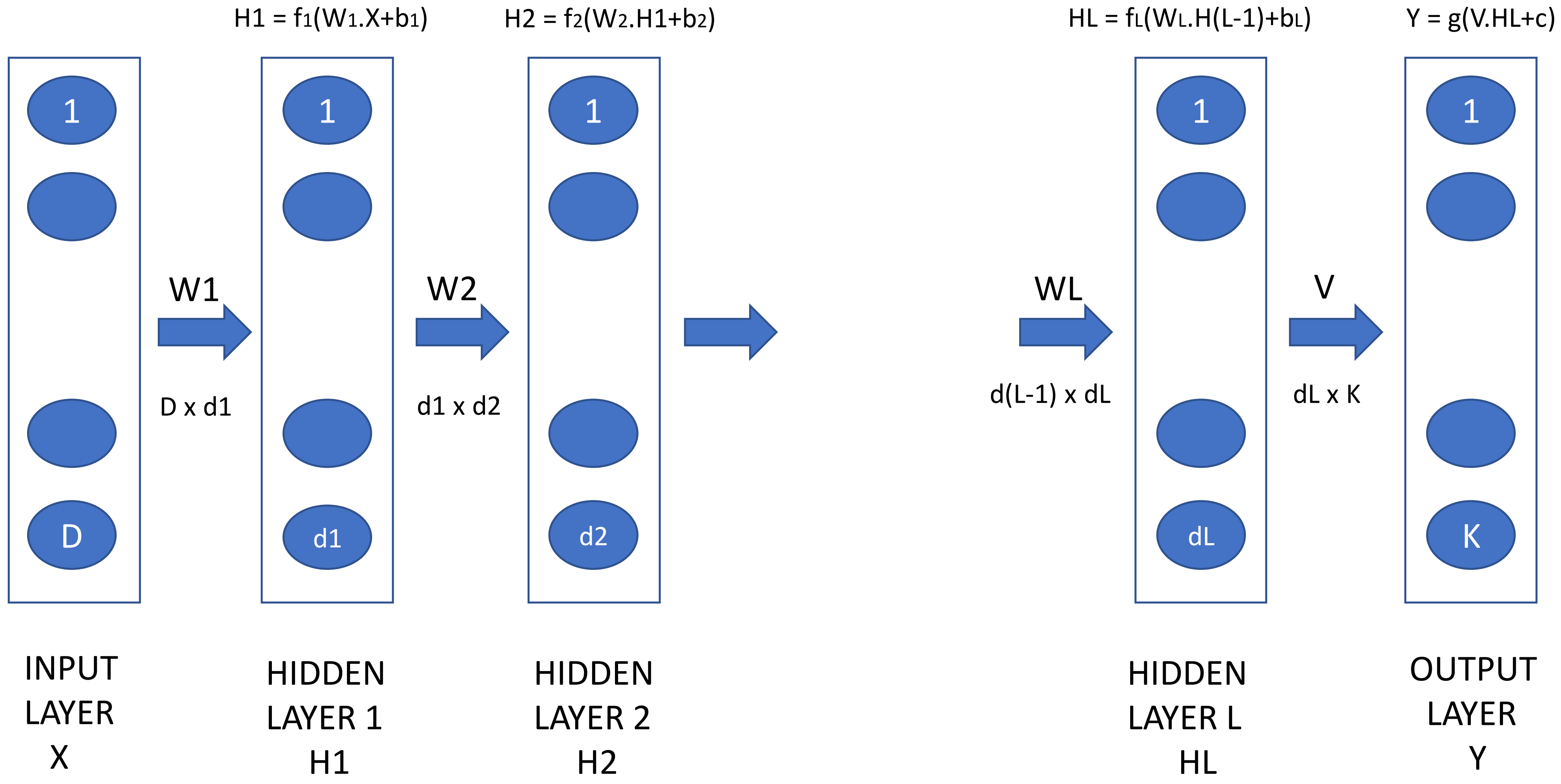
- We fuse both linear classifiers in the same neural network
- Multi-layer Neural Network!
- $C(1)=f1(w_1.x+b_{11})$
- $C(2)=f1(w_2.x+b_{12})$
- $Y=f2(v.c+b_{21})$



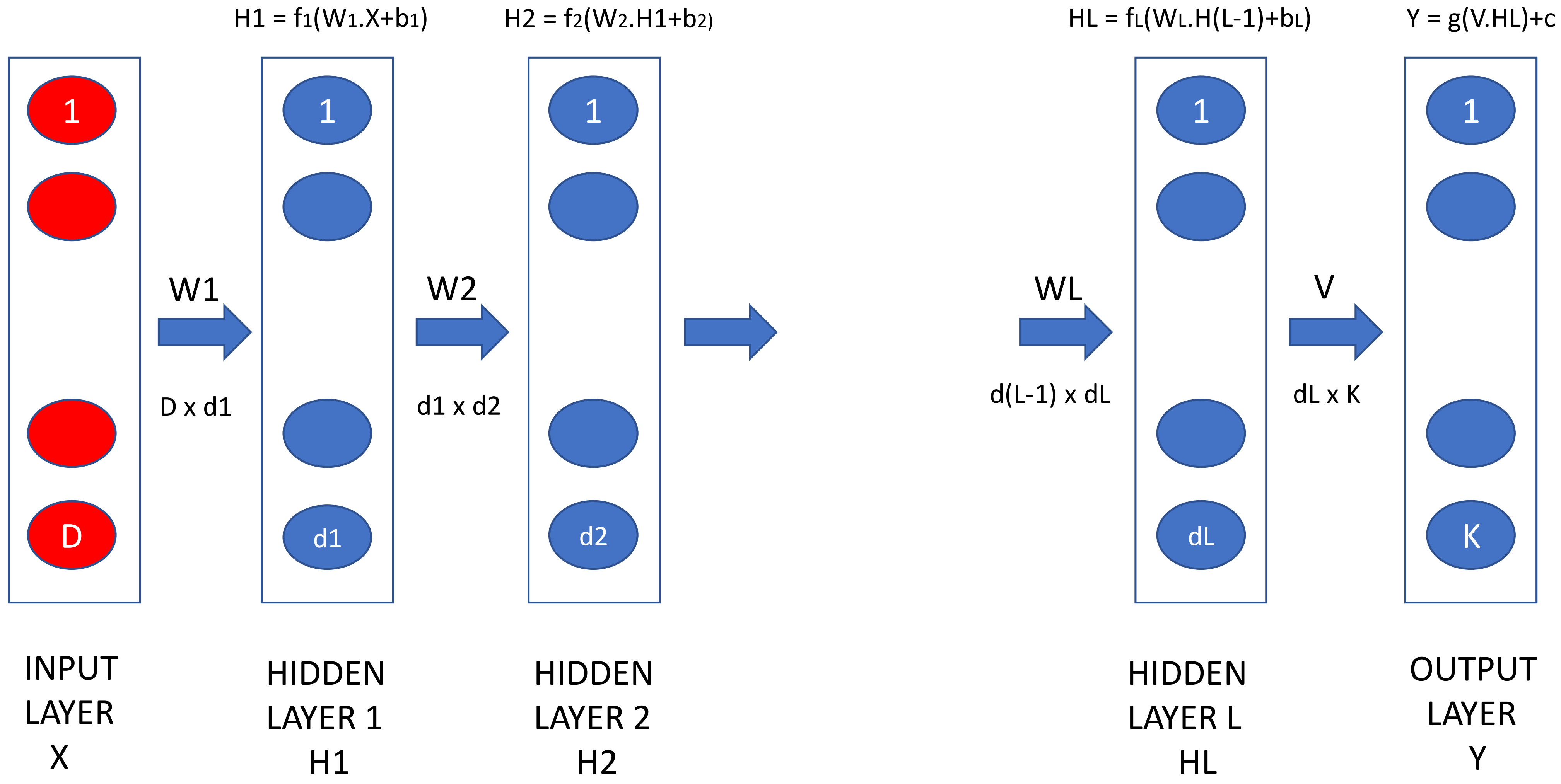
Multi-layer Neural Network



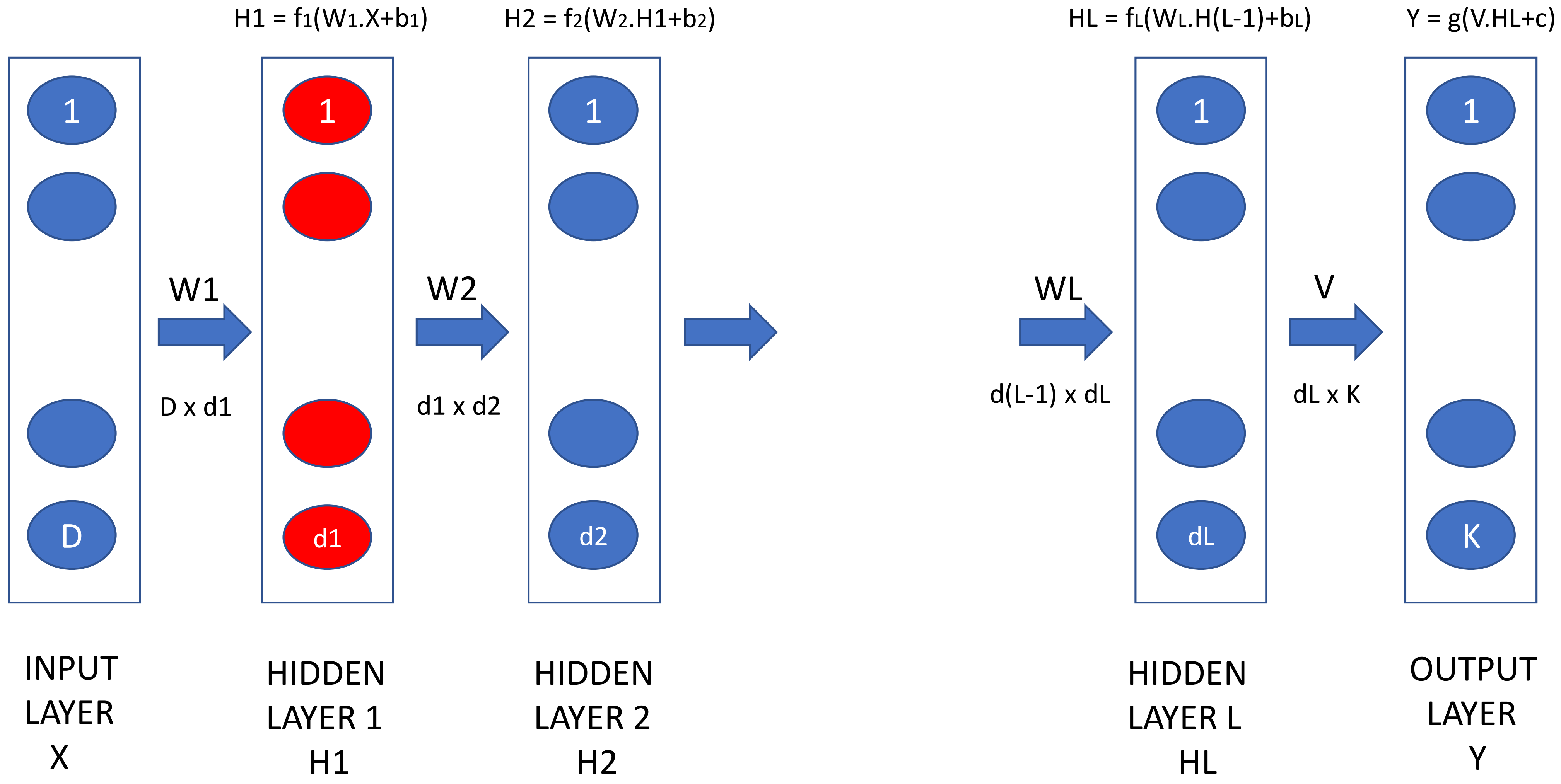
Multi-layer Neural Network



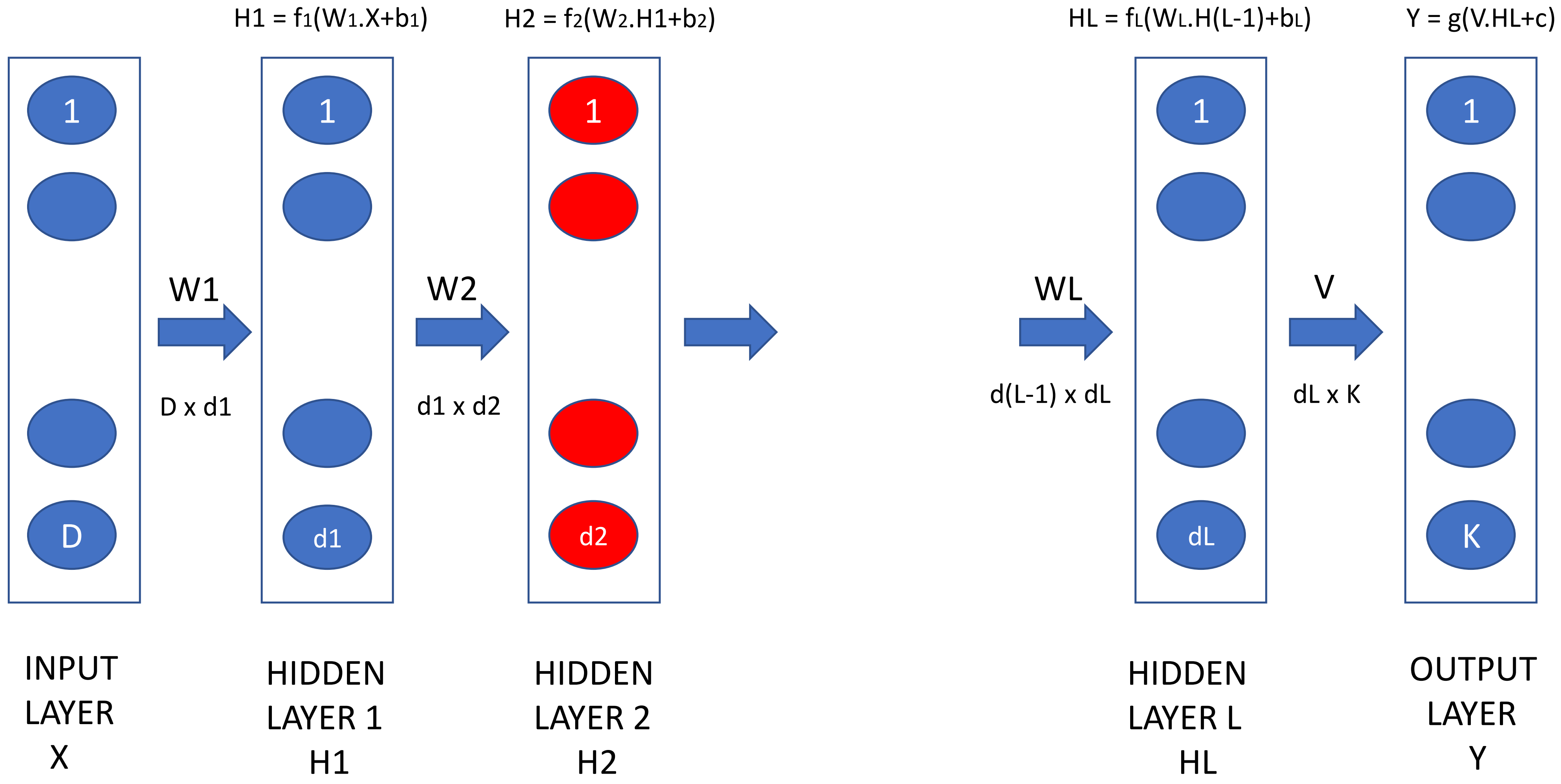
Feedforward operation



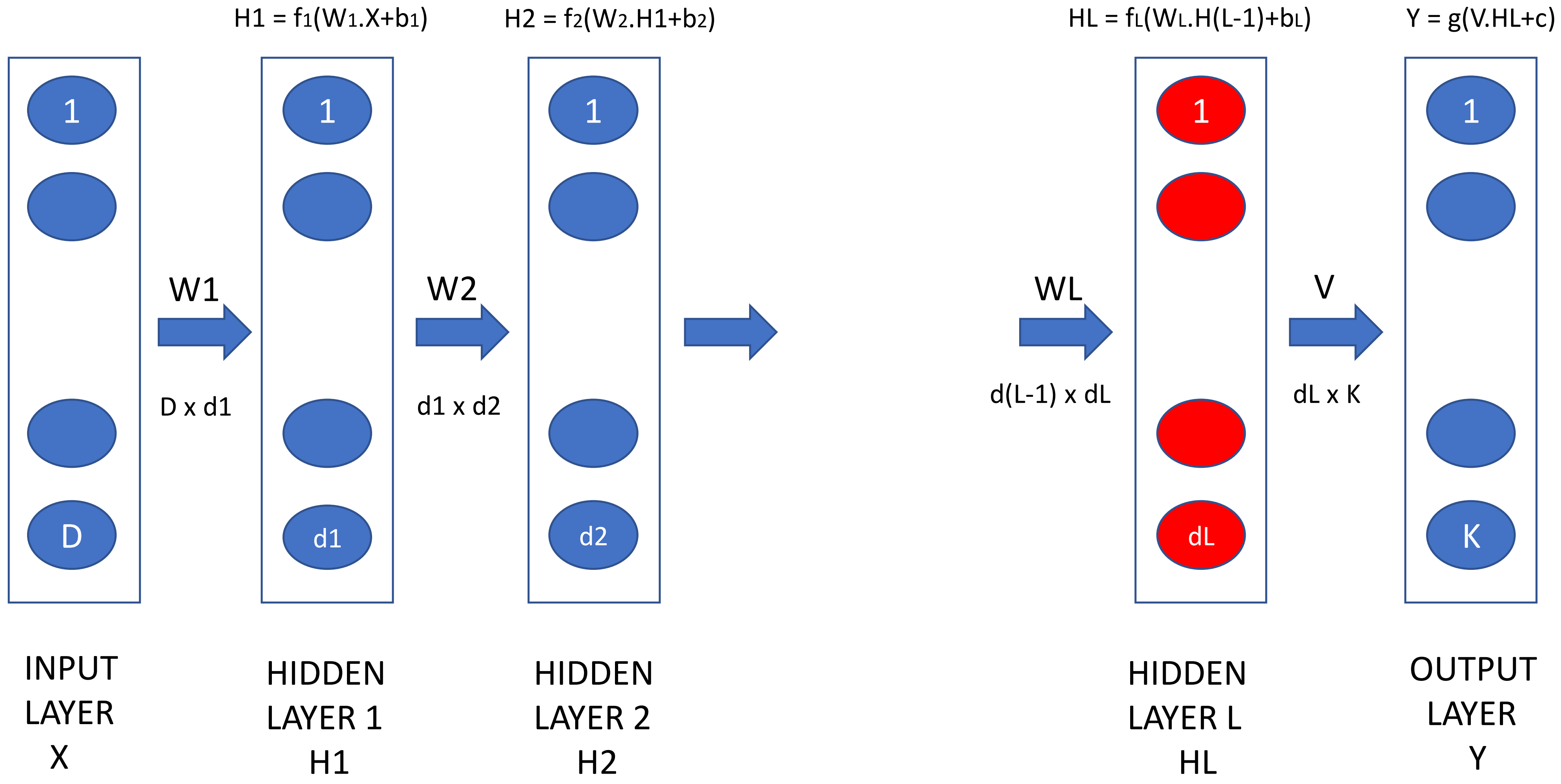
Feedforward operation



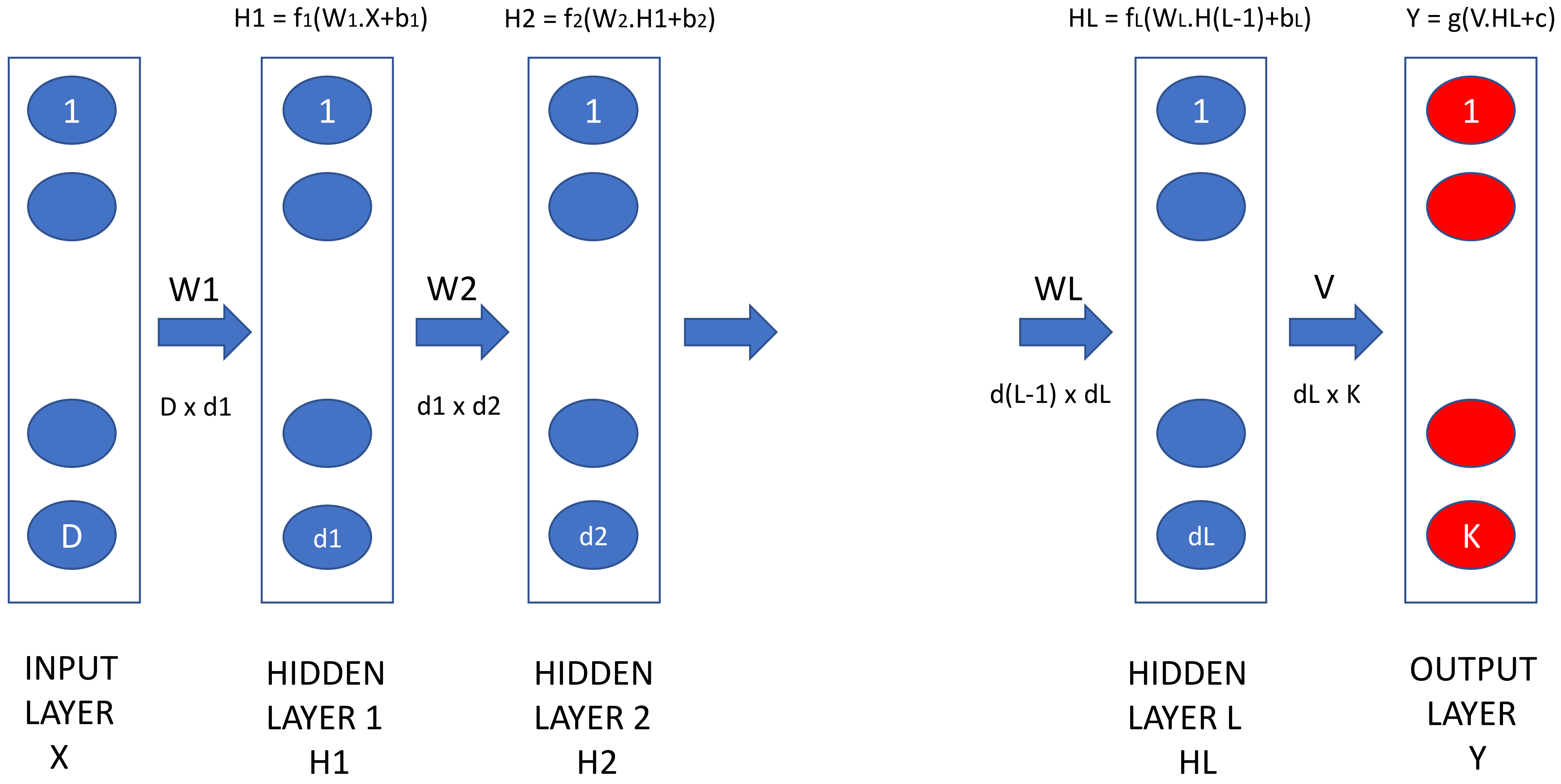
Feedforward operation



Feedforward operation



Feedforward operation



Role of the hidden layers

- Neural networks have the following parameters

    1) Number of hidden layers

    2) Number of units in each hidden layer

    3) Types of activation functions in the hidden layers

- These are chosen by the network designer
- The hidden layers represent complex functions of the input vector
- The functions are non-linear due to the activations

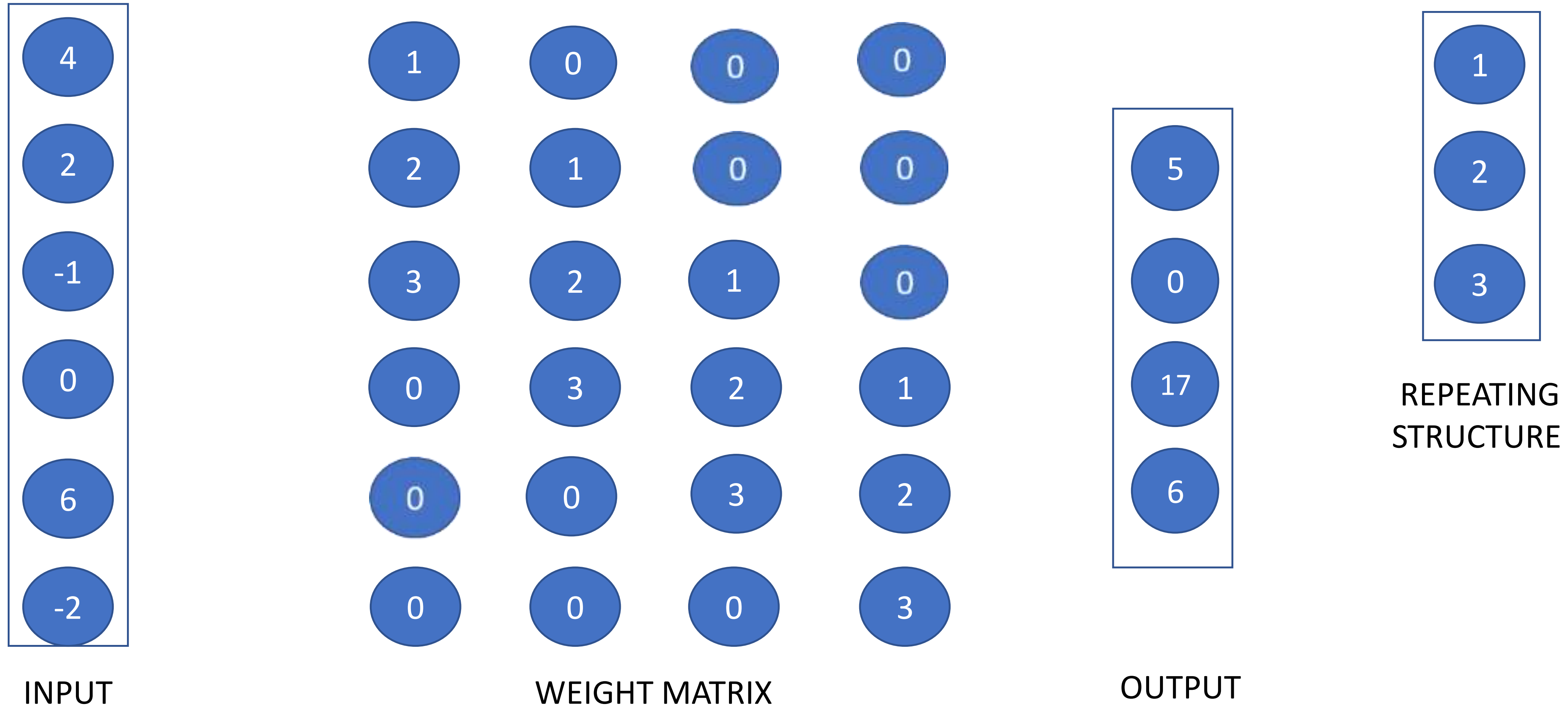
Convolutional Neural Network

- Some neural networks have “Special” structures
- There are sparse connections between adjacent layers (except the last layer)
- Many edges between two layers have “shared weights”
- This reduces number of parameters, and helps to capture local properties of the input
- Especially suitable for “structured” inputs such as images

Convolution Operation

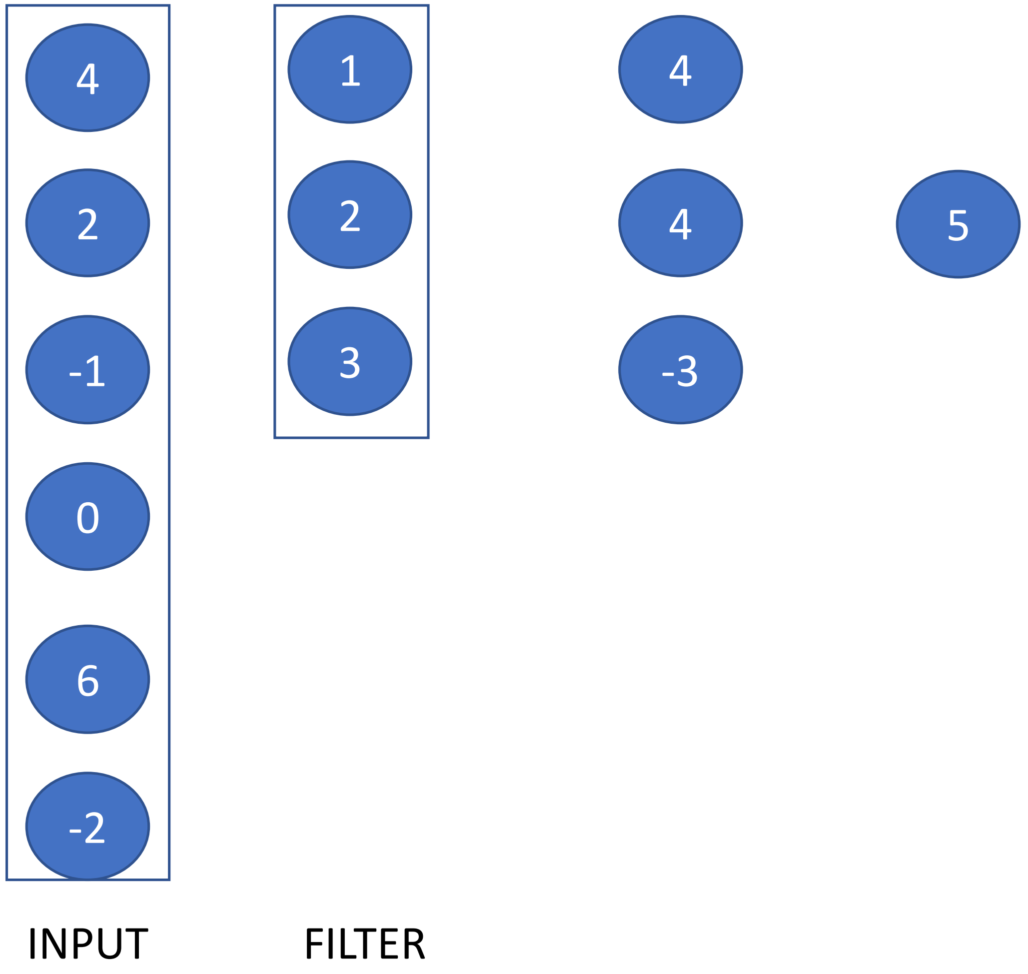
- A dot product operation with a special type of weight vector
- The weight vector has repeating structures and many empty values

Convolution Operation



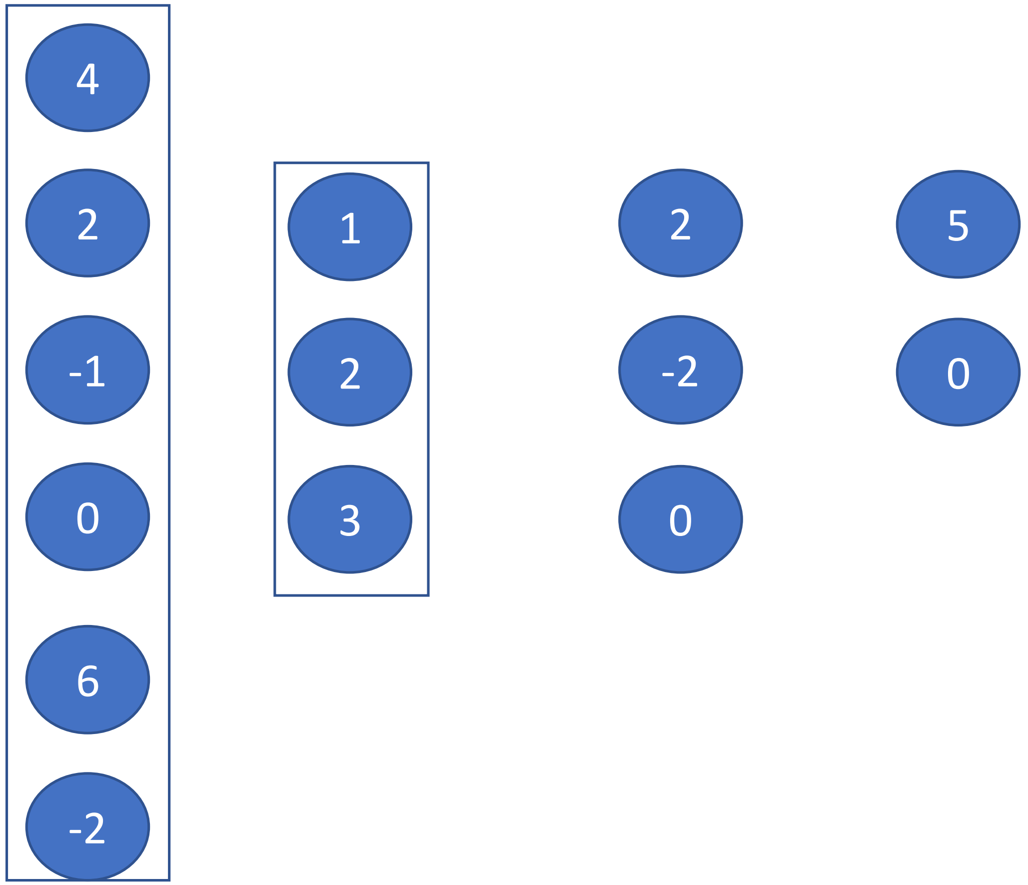
Convolution Operation

- Can be looked upon as sliding the repeating structure along the input

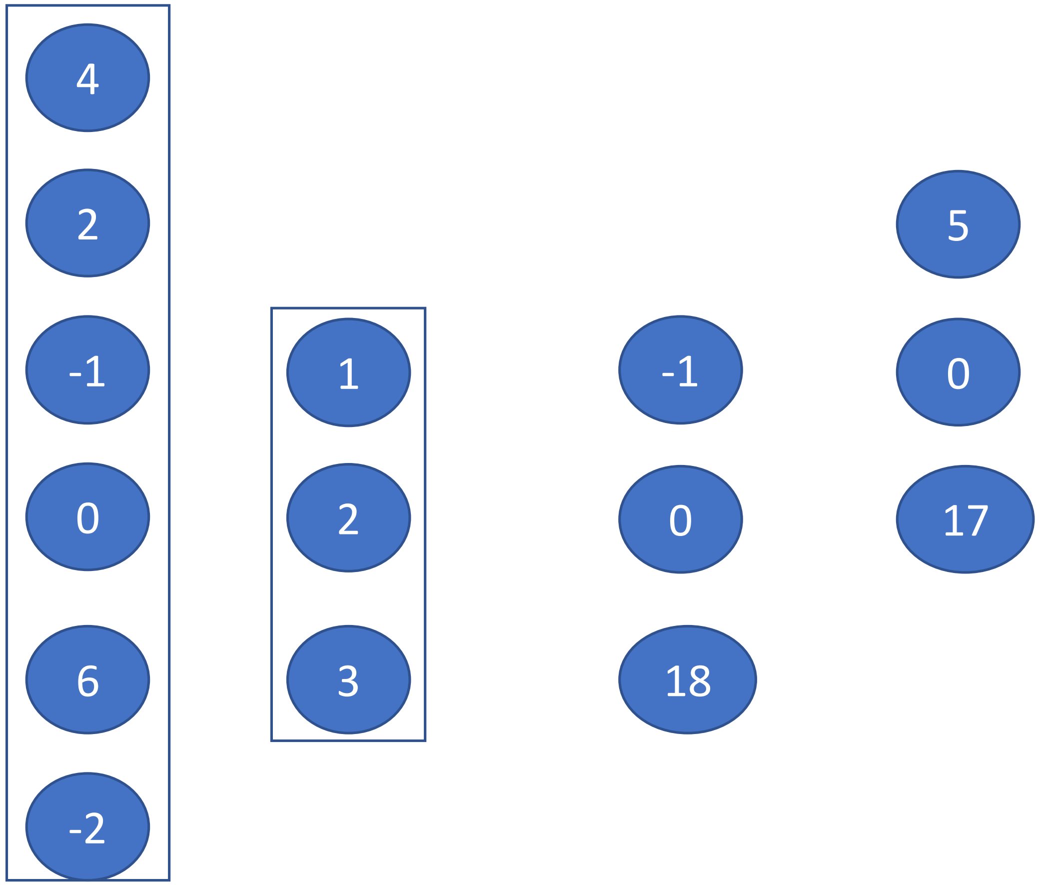


Convolution Operation

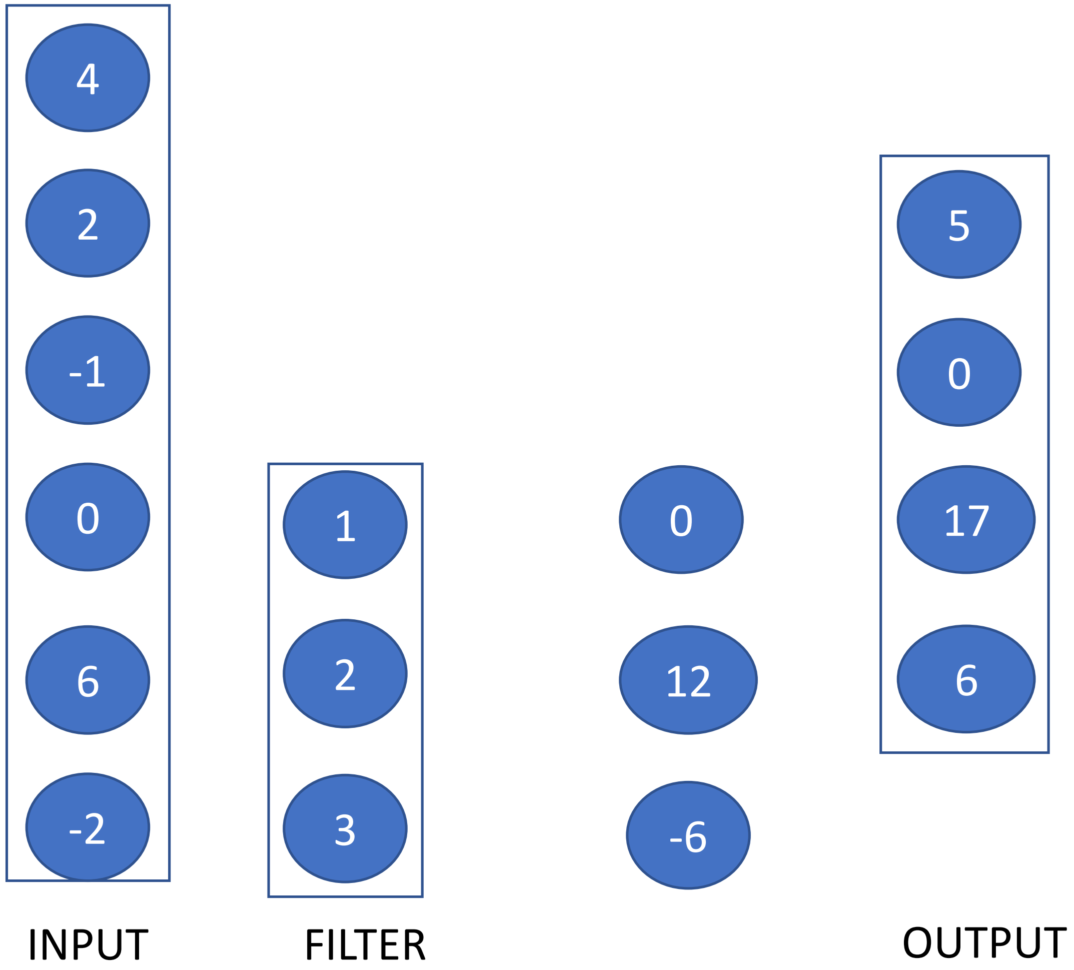
STRIDE: 1



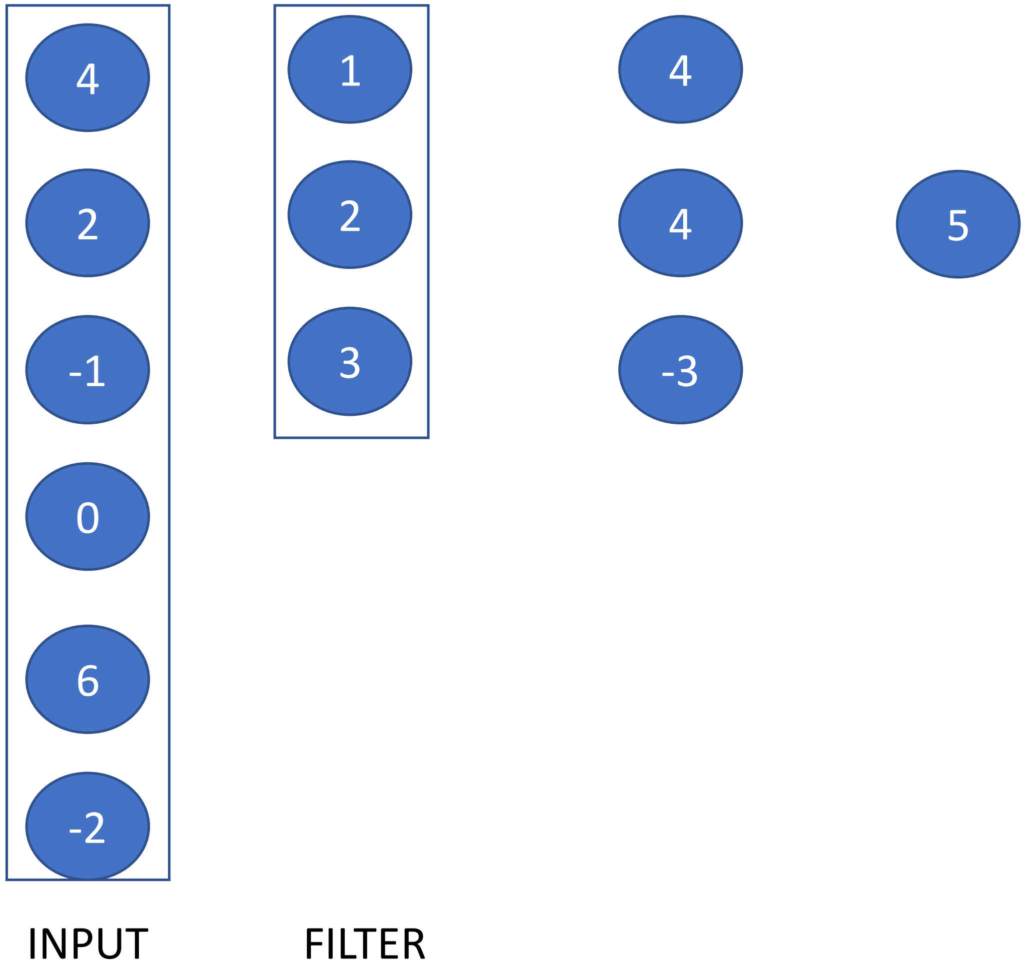
Convolution Operation



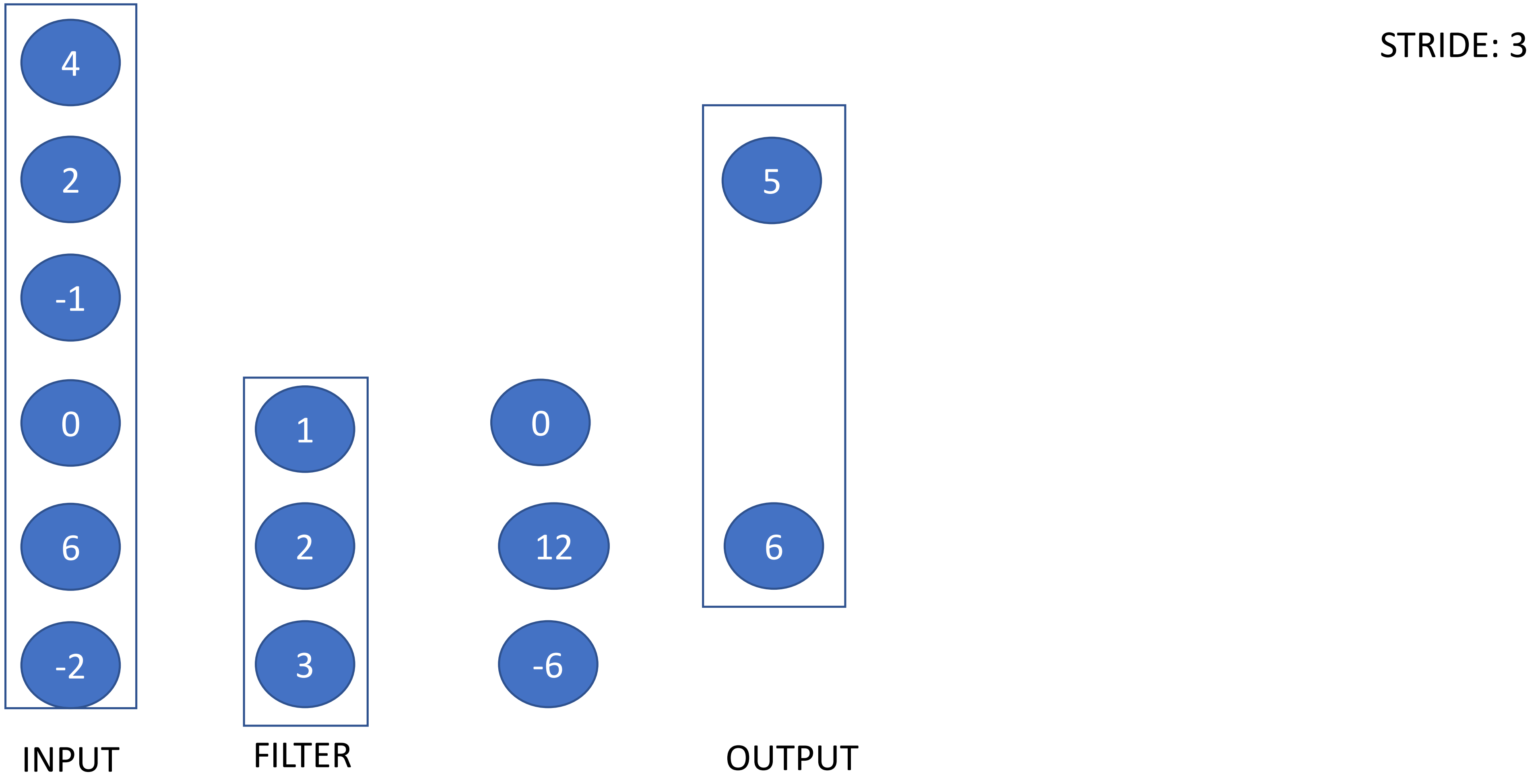
Convolution Operation



Convolution Operation



Convolution Operation



Convolution Operation

- The same operation can be done for matrix input and matrix filter
- The filter will move first row-wise and then column-wise
- Row-stride and column-stride will be specified

0	0	0	0
0	1	1	1
0	1	2	2
0	1	2	1

*

1	0	0
0	0	0
0	0	-1

=

0	0	0	0
0	-2	?	0
0	?	?	0
0	0	0	0

Pooling Operation

- Blockwise operation on a vector or matrix
- Operation: max (most common), mean, sum
- Block size: user input

Max-Pooling Operations

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Block size:2
Stride: 2

6	8
3	4

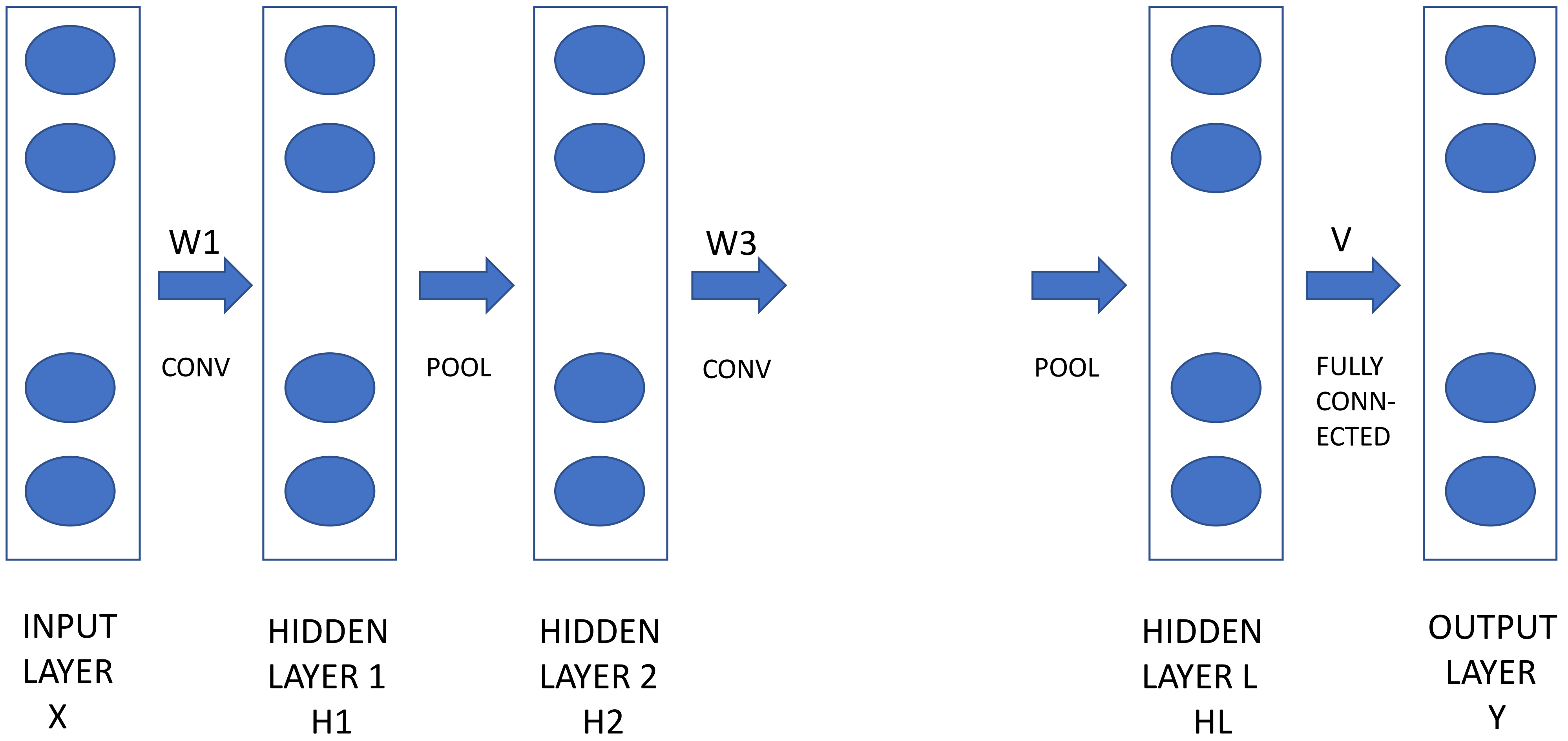
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Block size:3
Stride: 1

7	8
7	8

Convolutional Neural Network

- A convolutional neural network has many "convolution layers"
- Usually, each convolutional layer is followed by a pooling layer



Convolutional Neural Network

- In large neural networks, each convolution layer involves multiple convolution operations with multiple filters on the same input!
- This results in creation of “feature maps”

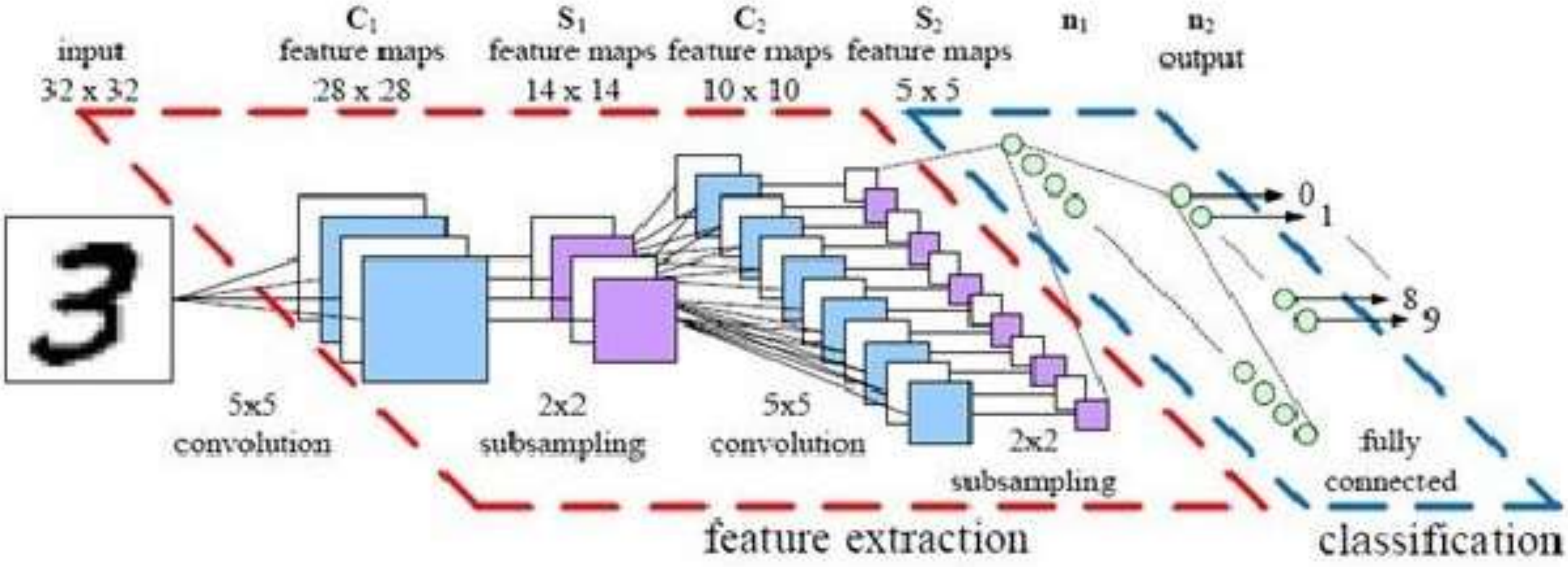


Image Source: Google Images

Parameters of a Neural Network

- Number of hidden layers
- Number of nodes in each hidden layer
- Number of connections across layers
- Activation functions at each layer
- Weights of the connections

Specified by designer

Learnt from data

Activation functions

- Activation functions are specific to each layer
- They are non-linear so that the network represents a non-linear function
- Most common: Sigmoid, tanh and RELU

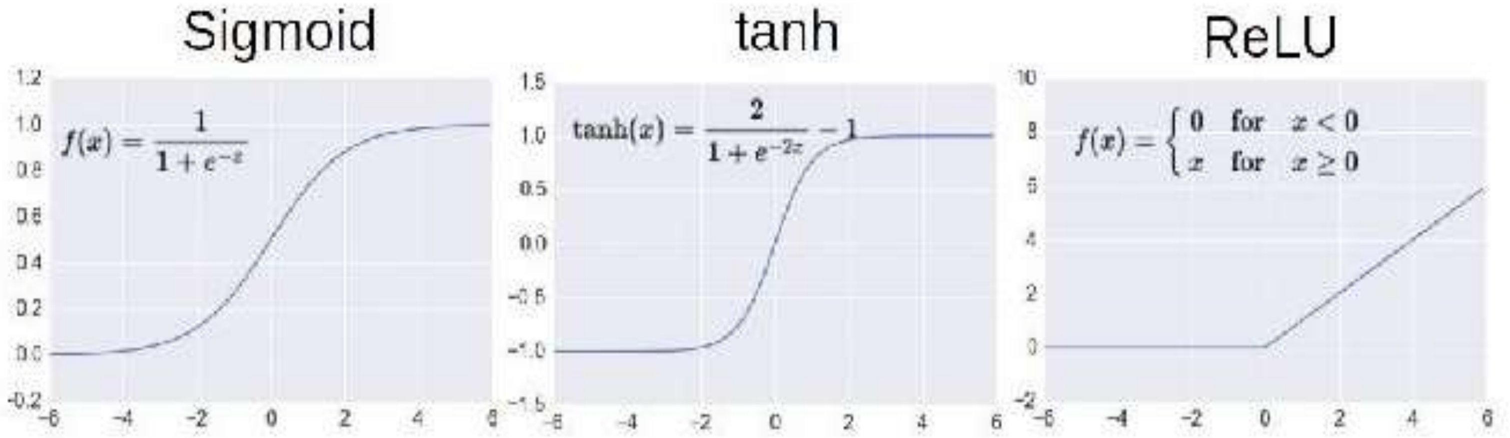


Image Source: Google Images

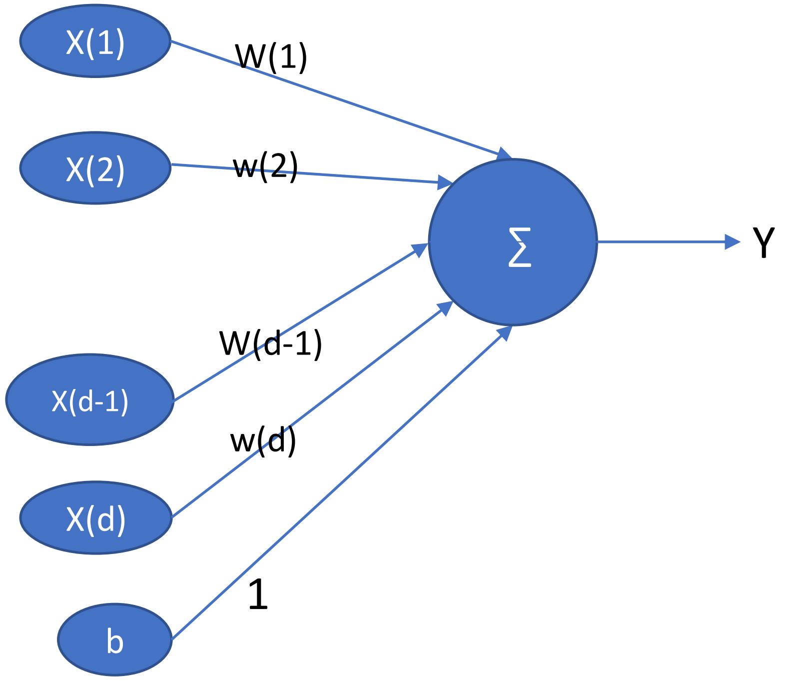
Learning the parameters

- The weights of the neural network are estimated from data
- The output of the network is compared with expected output during training phase
- Comparison of outputs via loss function
- Weights of the network adjusted to minimize this loss function
- Gradient descent used to adjust the weights
- Required: derivative of loss function w.r.t. each weight!

Parameter learning in simple Neural Network

- Consider a simple neural network
- $Y = W.X+b$
- $L(Y,t) = (Y-t)^2 = (w.x + b - t)^2 = (\sum_i w_i x_i + b - t)^2$

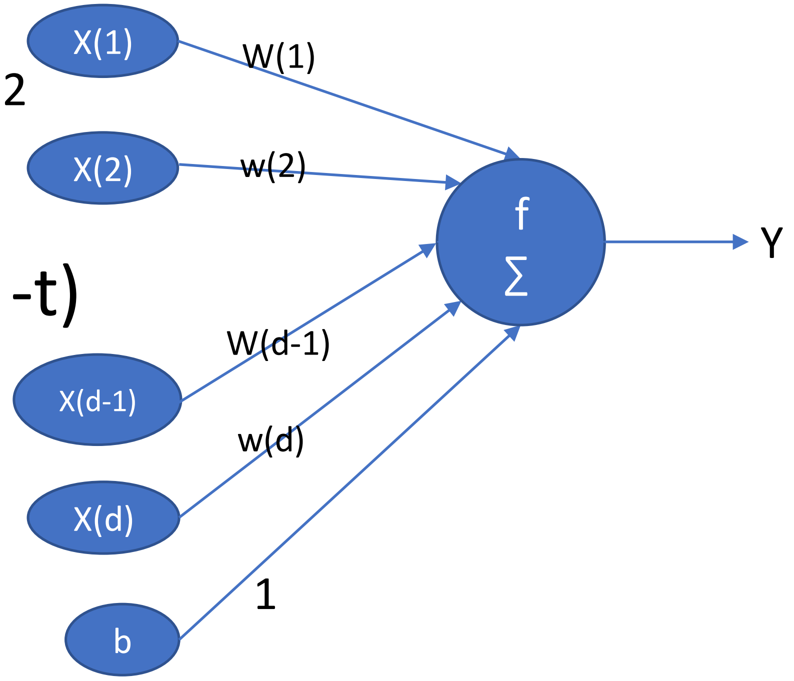
- Derivative $\Delta L(w_j) = 2x_j(\sum_i w_i x_i + b - t)$
- $W_j = W_j - \alpha \Delta L(w_j)$
- [Repeat for all dimensions ‘j’]



Parameter learning in simple Neural Network

- Consider a simple neural network
- $Y = W.X+b$
- $L(Y,t) = (Y-t)^2 = (f(w.x + b) - t)^2 = (f(\sum_i w_i x_i + b) - t)^2$

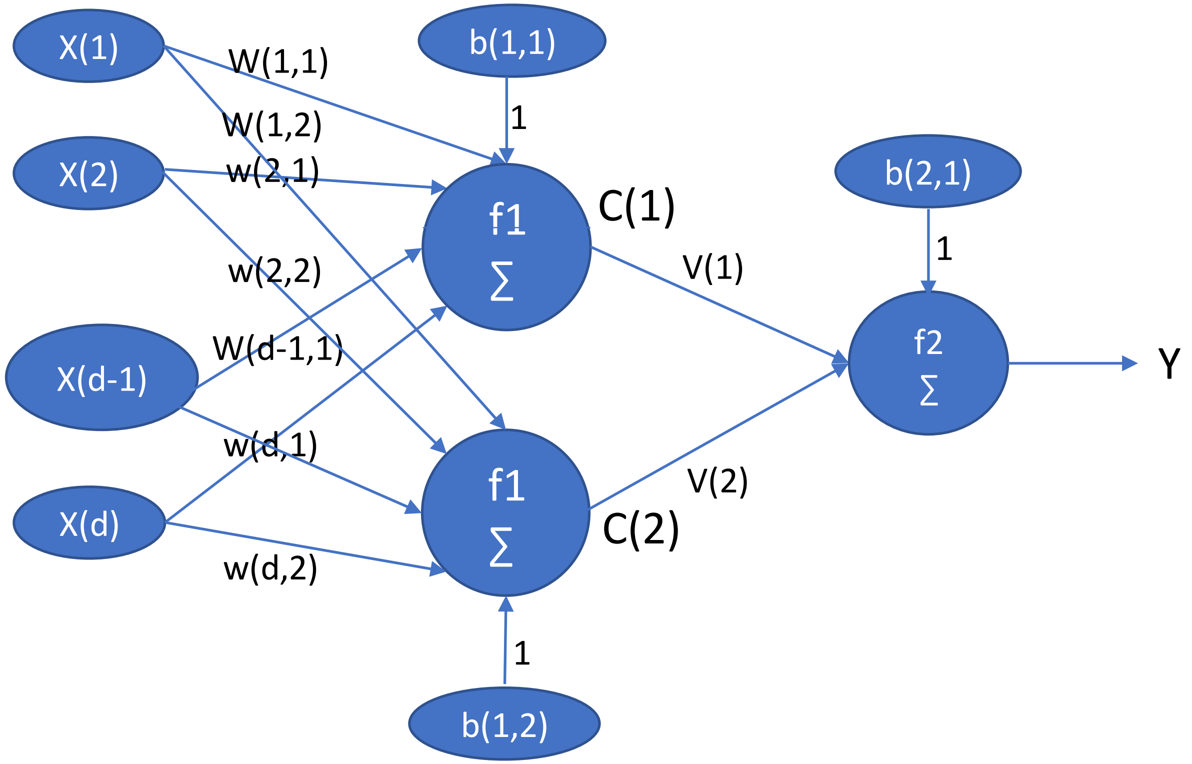
- Derivative $\Delta L(w_j) = 2x_j\, f'(\sum_i w_i x_i + b)\,(f(\sum_i w_i x_i + b) - t)$
- $W_j = W_j - \alpha \Delta L(w_j)$
- [Repeat for all dimensions 'j']



What if the network is deeper?

- For deep networks, the hidden weights too need to be updated!
- Weights are updated turnwise, from

output layer to input layer

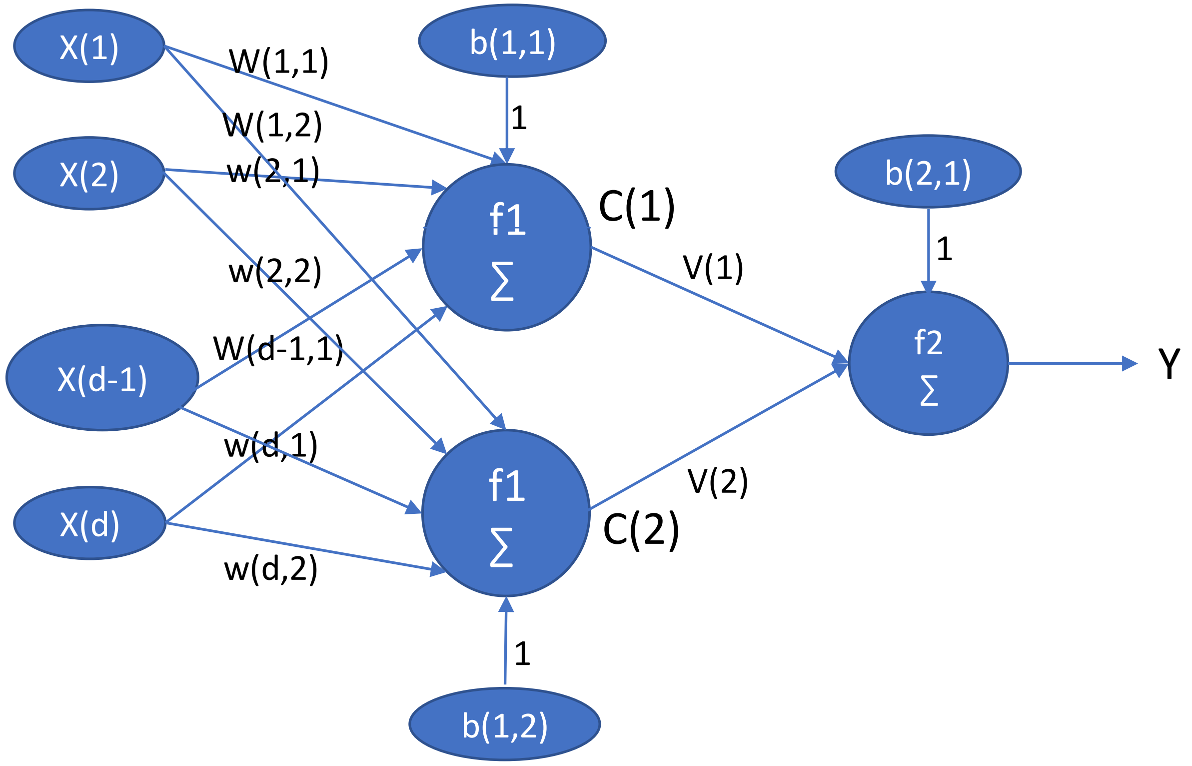


What if the network is deeper?

- For deep networks, the hidden weights too need to be updated!
- Weights are updated turnwise, from

output layer to input layer

- First update v
- Loss $L = (f_2(v.c+b_{21})-t)^2$
- $\Delta L(v_j) = 2c_j\, f_2'(v.c+b)\, (f_2(v.c+b) - t)$

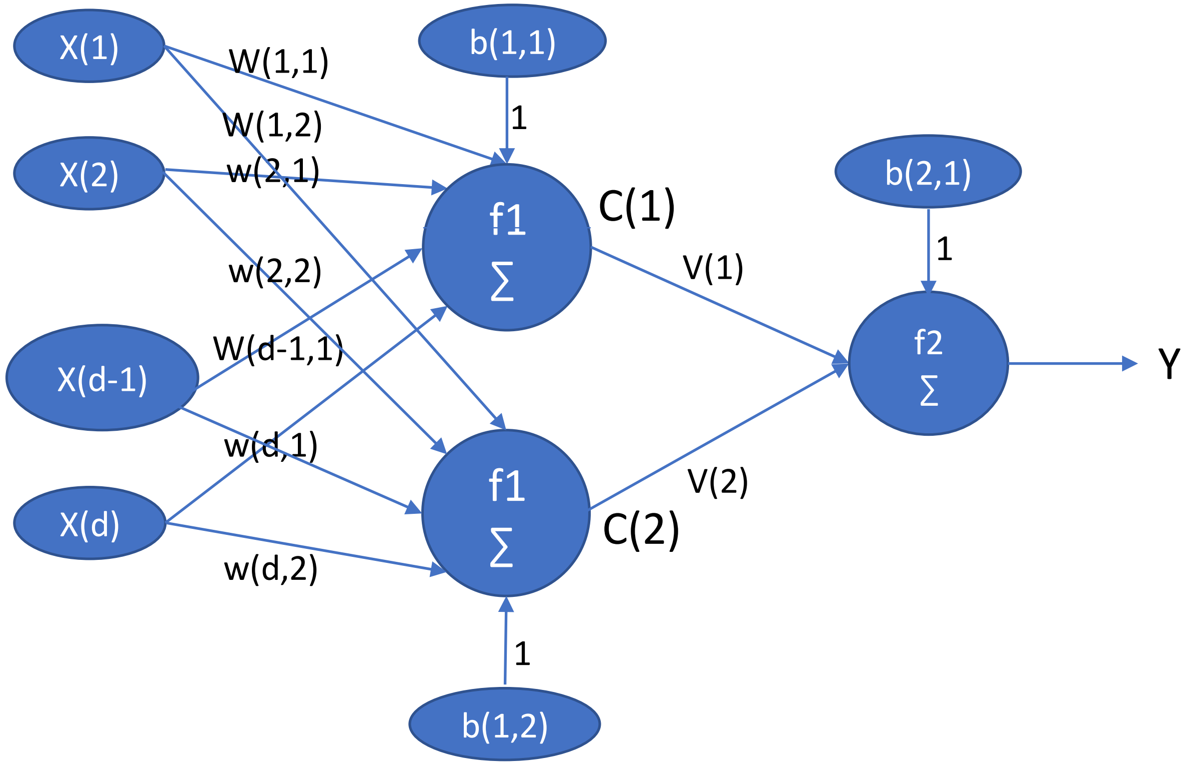


What if the network is deeper?

- For deep networks, the hidden weights too need to be updated!
- Weights are updated turnwise, from

output layer to input layer

- First update v
- Loss $L = (f_2(v.c+b_{21})-t)^2$
- $\Delta L(v_j) = 2c_j\, f_2'(v.c+b)\,(f_2(v.c+b) - t)$
- Next update w
- $c_j = f_1(x.w_j + b_j) = f_1(\sum_i w_{ij} x_{ij} + b_j)$
- $\Delta L(w_{ij})$: calculate using chain rule!

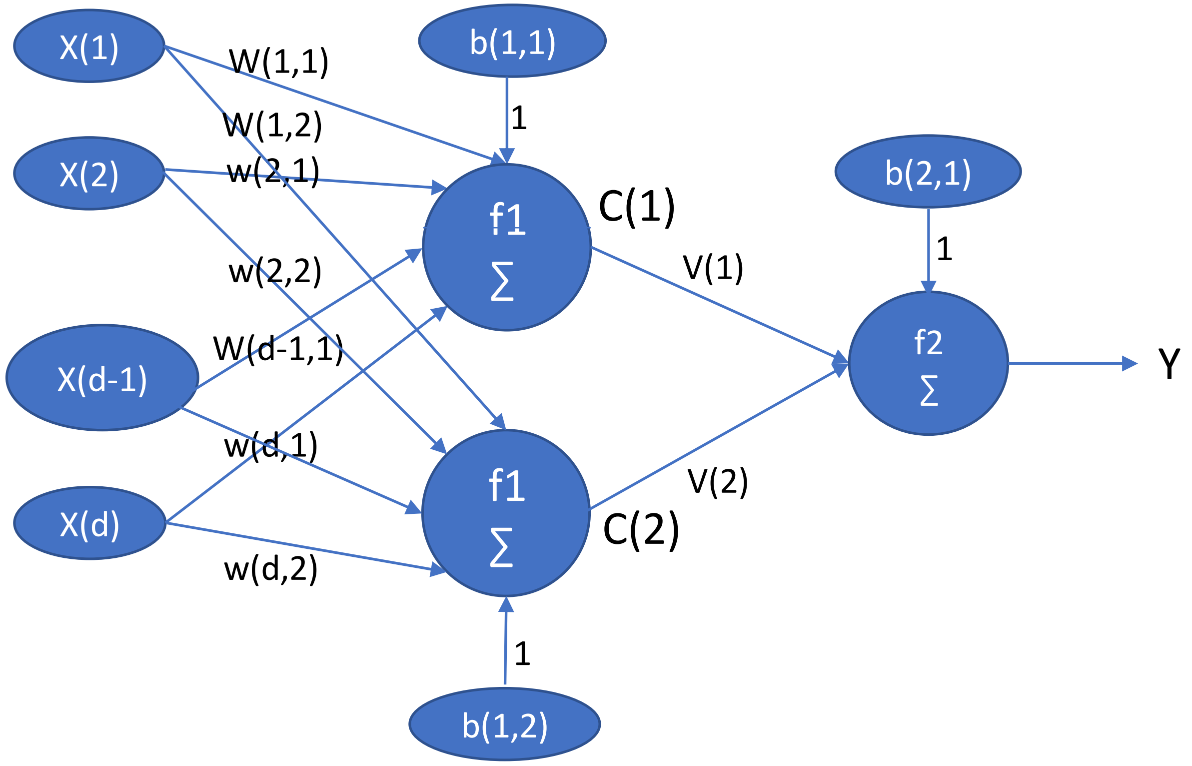


What if the network is deeper?

- For deep networks, the hidden weights too need to be updated!
- Weights are updated turnwise, from

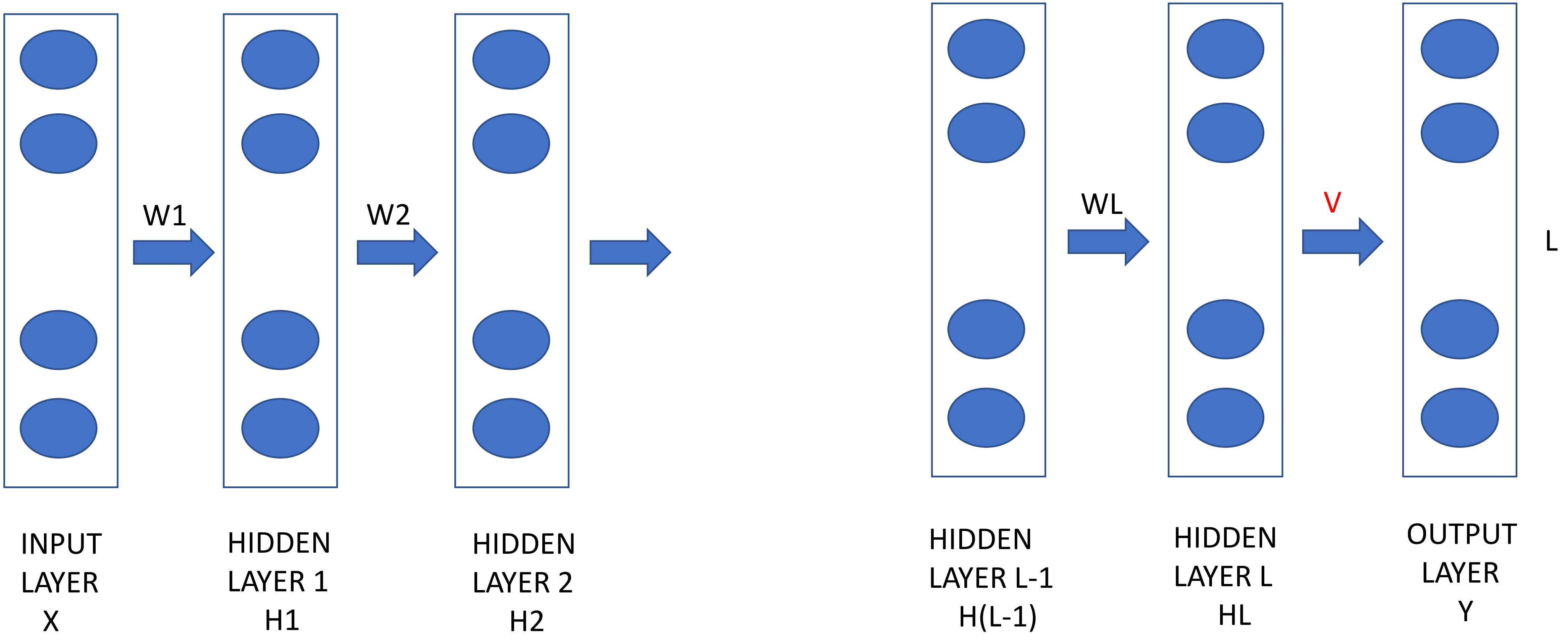
output layer to input layer

- First update v
- Loss $L = (f_2(v.c+b_{21})-t)^2$
- $\Delta L(v_j) = 2c_j\, f_2'(v.c+b)\,(f_2(v.c+b) - t)$
- Next update w
- $c_j = f_1(x.w_j + b_j) = f_1(\sum_i w_{ij} x_{ij} + b_j)$
- $\Delta L(w_{ij})$: calculate using chain rule!



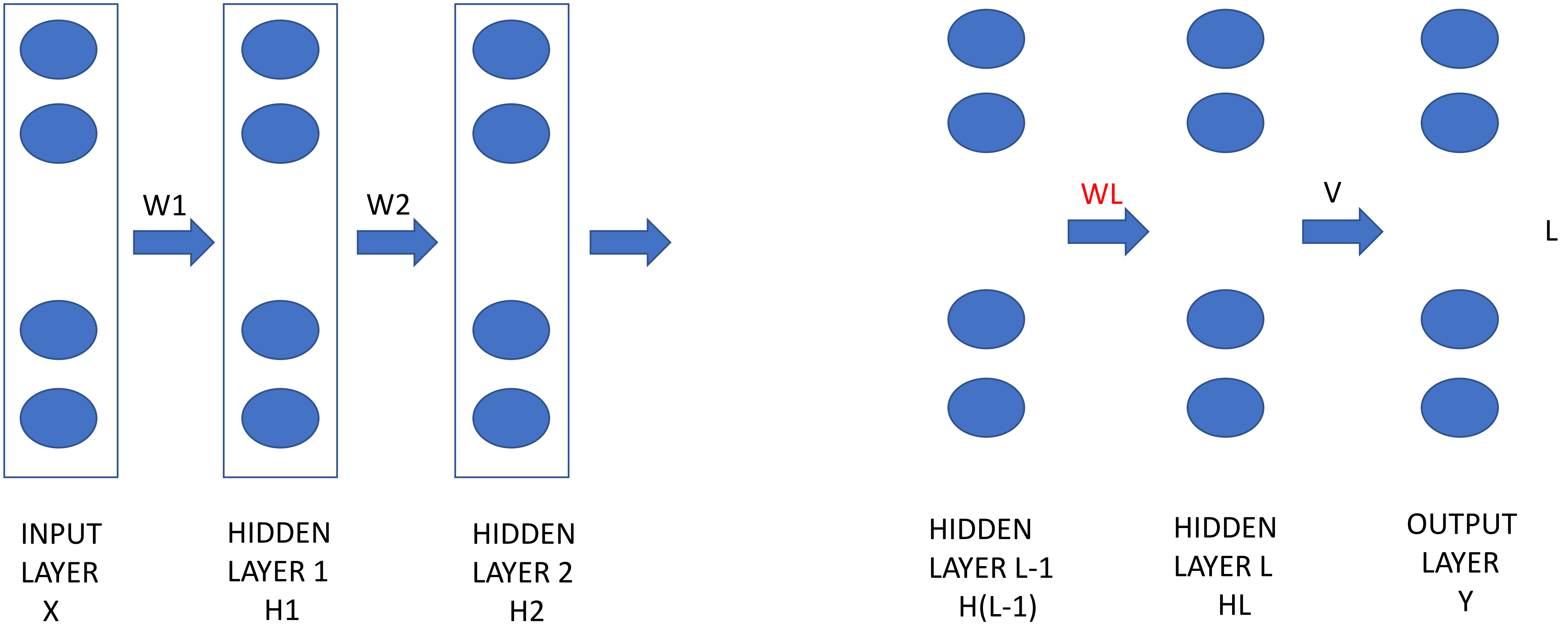
Backpropagation

- Calculate gradients of outermost parameters w.r.t. loss function
- Update these parameters



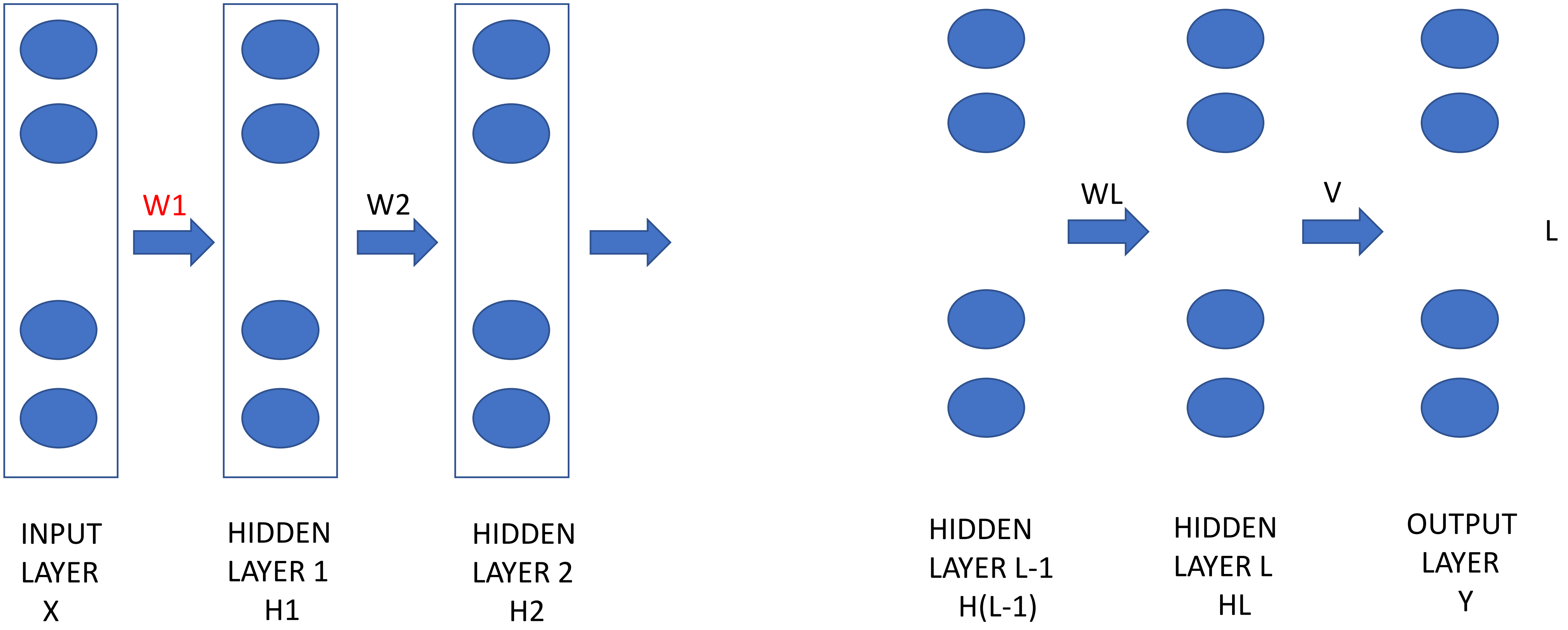
Backpropagation

- Calculate gradients of next set of parameters
- Use chain rule and re-use the values already calculated



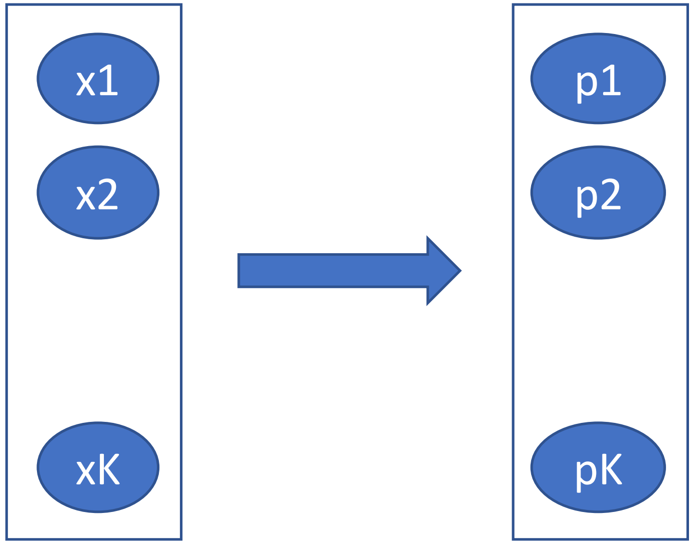
Backpropagation

- Continue updating weights in successive layers
- The gradients flow backwards towards input layer



Probabilistic Classification by Neural Network

- In case of a K-classification problem, we need K output nodes
- These represent the probability of each class for the given input
- K-dimensional input from the last hidden layer converted into probability distribution
- Softmax function:



pi = exp(xi)/S where
S = exp(x1) + exp(x2) + …… + exp(xK)

exp makes all elements of x positive
Dividing by S makes their sum to 1

Probabilistic Classification by Neural Network

- How to train for K-classification?
- Compare with expected output – one-hot vector indicating true label
- Loss function: cross-entropy (compares two probability distributions)

- $H(p,q) = - \sum_x p(x)\log(q(x))$ [p: softmax output, q: expected one-hot output]
- Weights updated to minimize this function!

Python implementation

```
In [1]: import matplotlib.pyplot as plt
        #plot the first image in the dataset
        #plt.imshow(X_train[0],cmap='gray')
```

```
In [0]: #check image shape
        X_train[0].shape

Out[4]: (28, 28)
```

```
In [0]: #reshape data to fit model
        X_train = X_train.reshape(60000,28,28,1)
        X_test = X_test.reshape(10000,28,28,1)
```

```
In [0]: from keras.utils import to_categorical
        #one-hot encode target column
        y_train = to_categorical(y_train)
        y_test = to_categorical(y_test)

Out[6]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

```
In [0]: from keras.models import Sequential
        from keras.layers import Dense, Conv2D, Flatten,MaxPool2D
        #create model
        model = Sequential()
        #add model layers
        model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
        model.add(Conv2D(32, kernel_size=3, activation='relu'))
        model.add(MaxPool2D((2,2)))
        model.add(Flatten())
        model.add(Dense(10, activation='softmax'))
```

```
In [0]: #compile model using accuracy to measure model performance
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
In [0]: #train the model
        model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)
```

Node drop-out

- A neural network needs "regularizer" to prevent overfitting
- Regularizer: reduces model complexity
- Dropout: randomly ignore a few hidden nodes with all their connections
- During each round of training, each node is dropped with a probability
- Drop probability can vary across layers
- This forces the remaining nodes to take greater "responsibility" of prediction

Choice: wider or deeper?

- A neural network designer has two choices: deep or wide
- Deep: many hidden layer
- Wide: few hidden layers, with more nodes per layer
- Usually deep networks are preferred, as they allow computational units to be reused

Image Source: Google Images

Thank You!