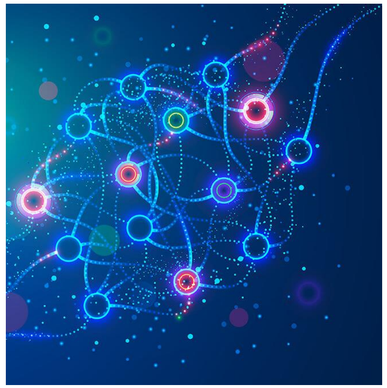
Neural Network and Deep Learning

Chapter 6



Prepared for SMSTU | Presented by Rajan Adhikari | 2023 November

Agendas

- Introduction to Neuron
- Mathematical Implementation of Neuron
- Feed Forward Neural Network
- Back Propagation
- Deep Neural Network (aka Deep Learning)
- Types of ANN
- Applications of Neural Network

Notations

m - no. of observations

n - no. of features

x - scalar

x - vector

X - Matrix

W - Weights

B - bias

b - Bias vector

z - Activation potential

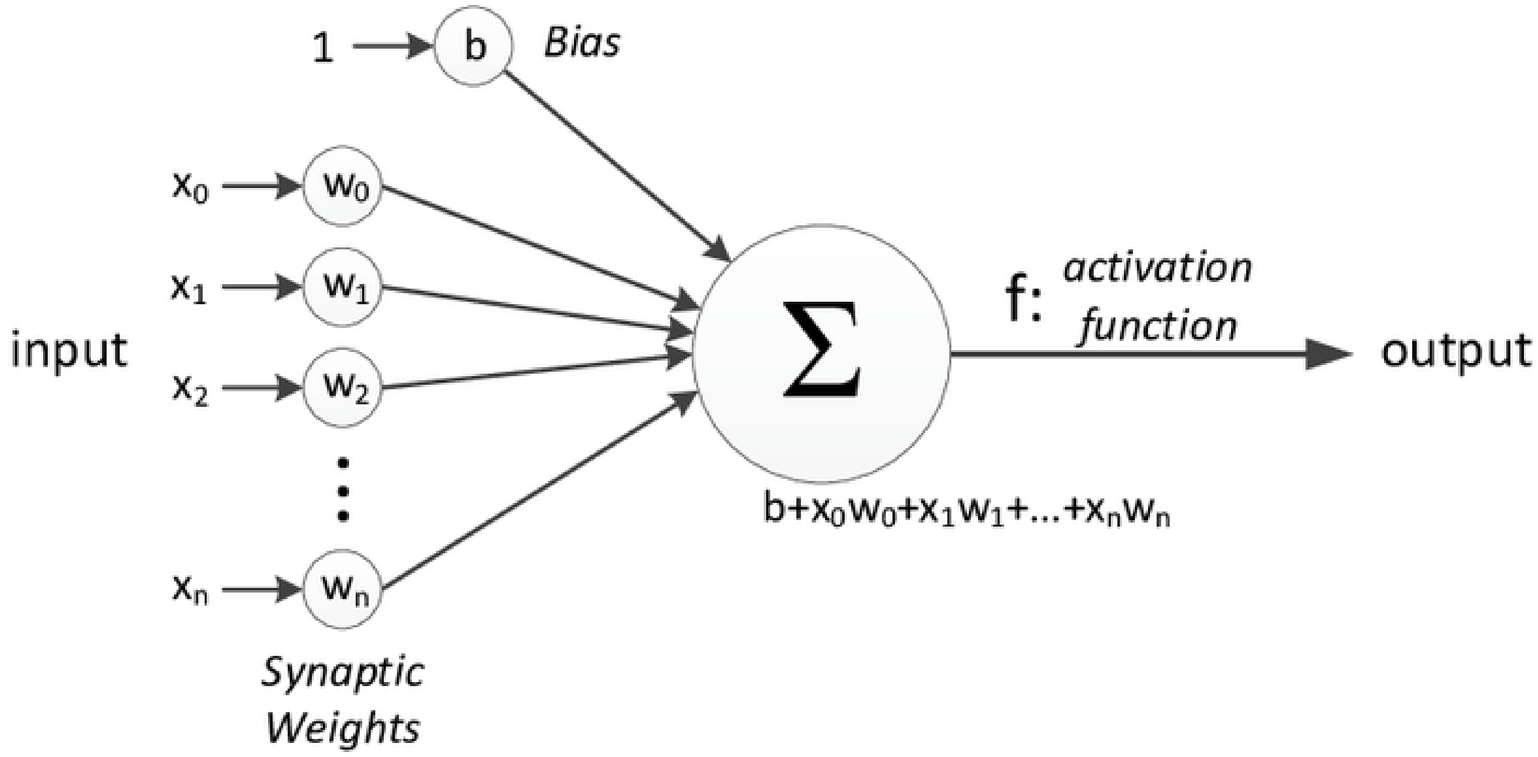
a - Activation vector

y - output vector

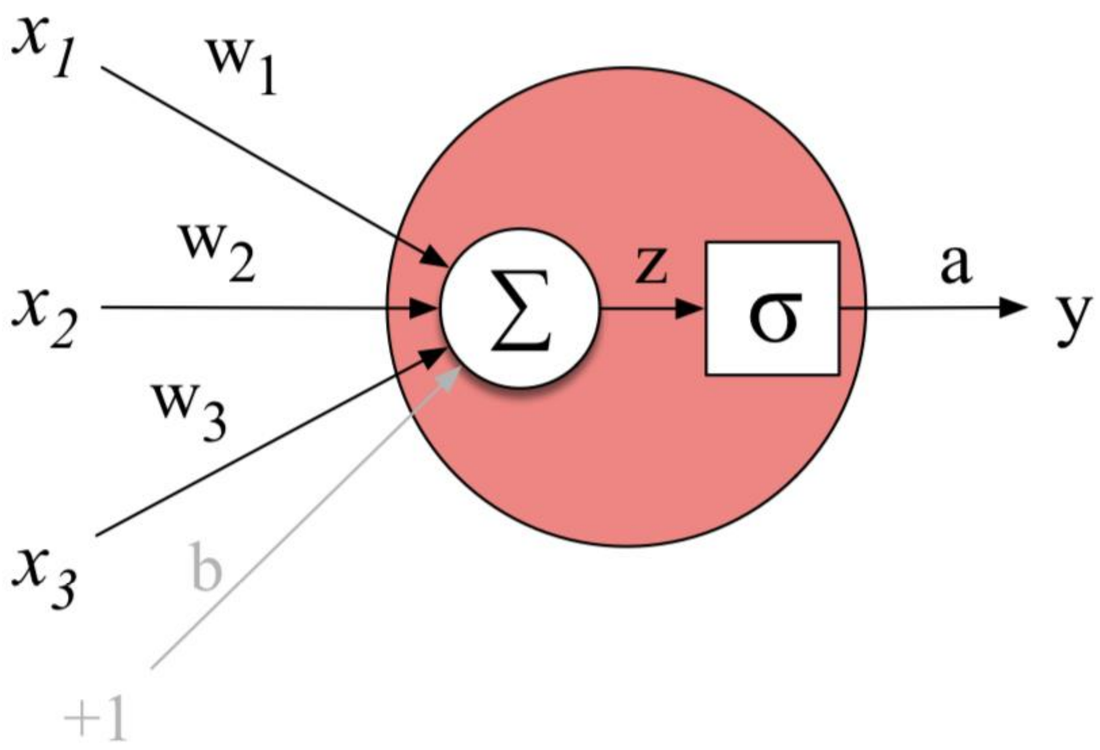
L - Layered

g() / f() - activation function

Mathematical model of neuron



Let's know the neurone



For single training example

$$z = b + \sum w_i x_i$$

or

$$z = b + \vec{w} \cdot \vec{x}$$

and

$$y = a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Numerically,

Calculation within neuron is:

$$\mathbf{w} = [0.2, 0.3, 0.9]$$
$$b = 0.5$$

What would this unit do with the following input vector:

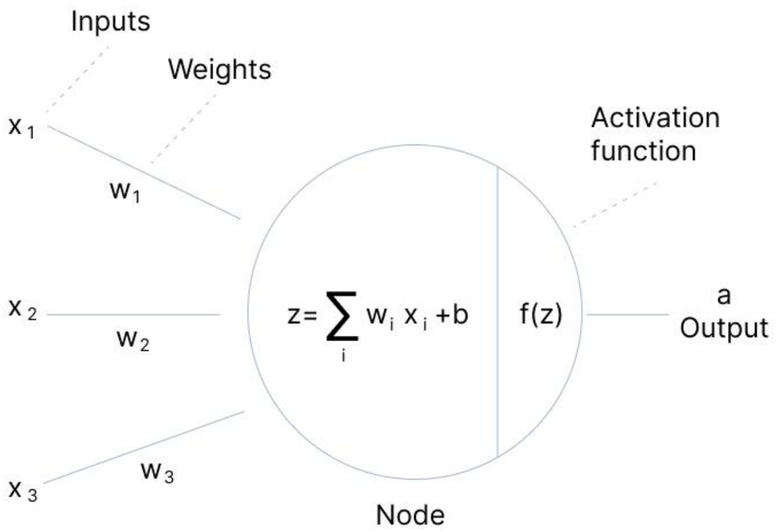
$$\mathbf{x} = [0.5, 0.6, 0.1]$$

The resulting output y would be:

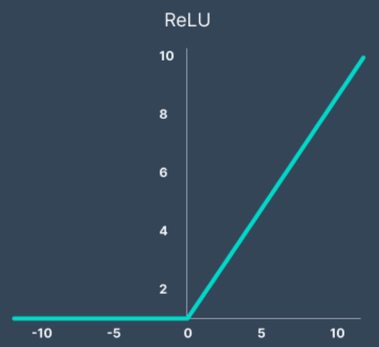
$$y = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} = \frac{1}{1 + e^{-(.5*.2 + .6*.3 + .1*.9 + .5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

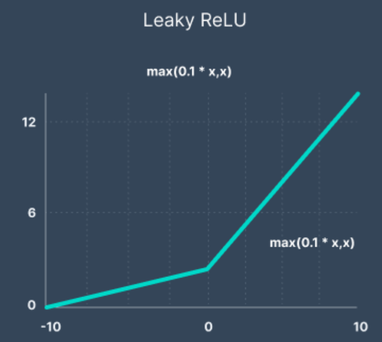
Activation Functions

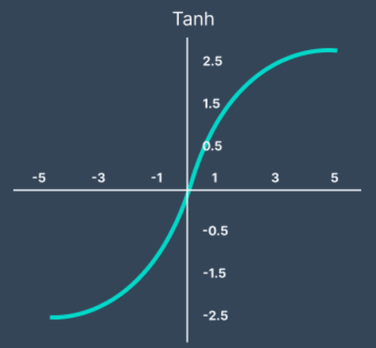
- Non - linear transformation in linear function

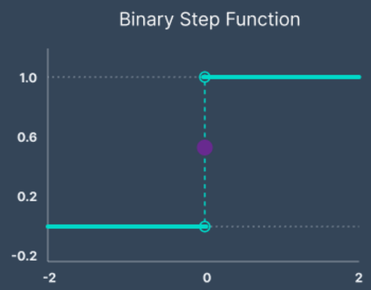


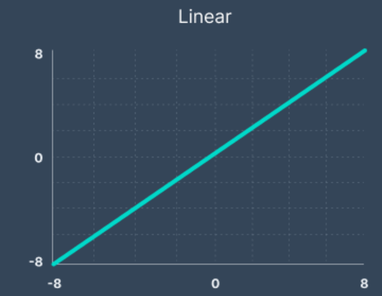
V7 Labs

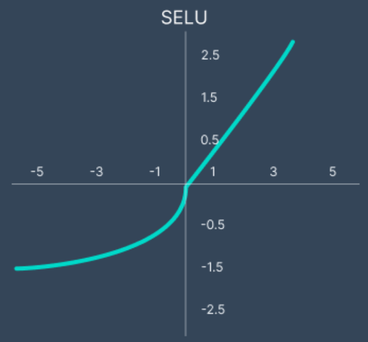


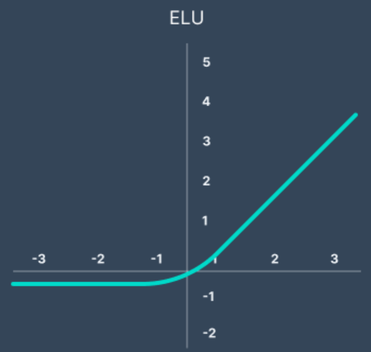


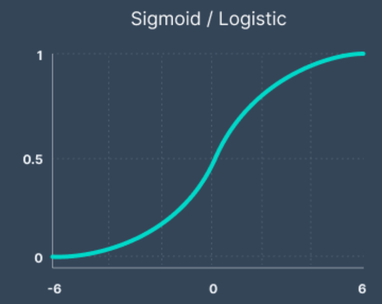


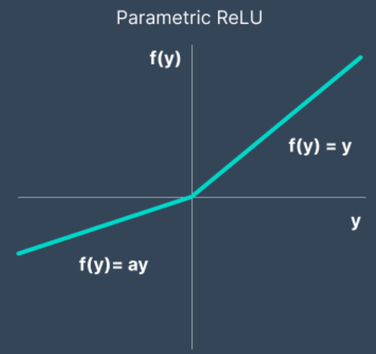






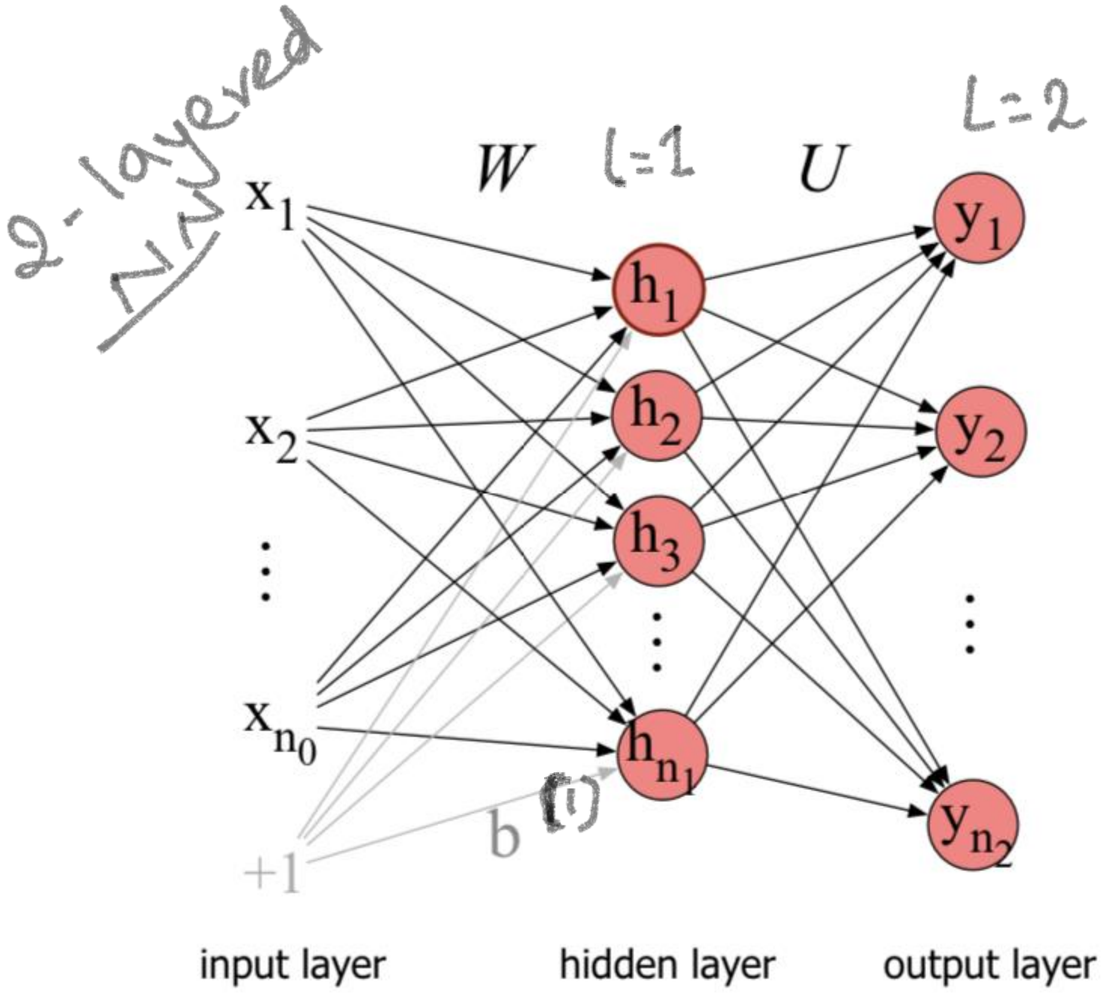






Feed Forward (2 - layered) Neural Network

Computation in 1st layer



$n_1 \times n_0$ matrix

$$\vec{h} = g(W\vec{x} + \vec{b}^{[1]})$$

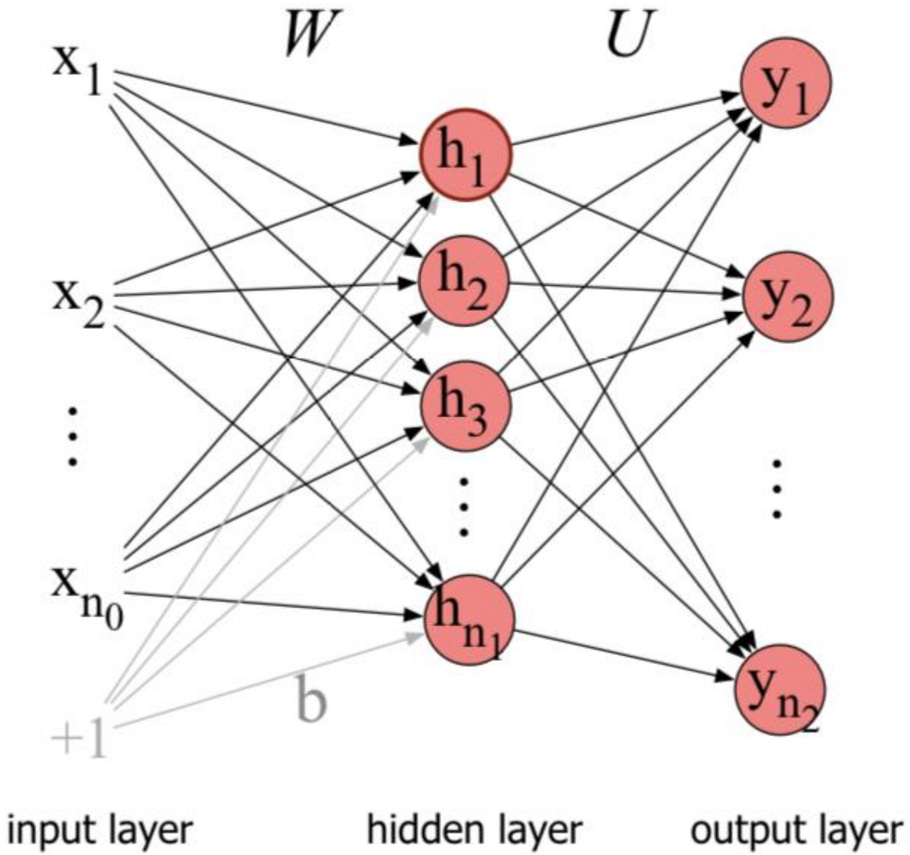
$n_1 \times 1$ vector

$n_1 \times 1$ vector

$n_0 \times 1$ vector

Feed Forward (2 - layered) Neural Network

Computation in Second layer



Similarly,

$$\vec{y} = g(U\vec{h} + \vec{b}^{[2]})$$

$n_2 \times 1$

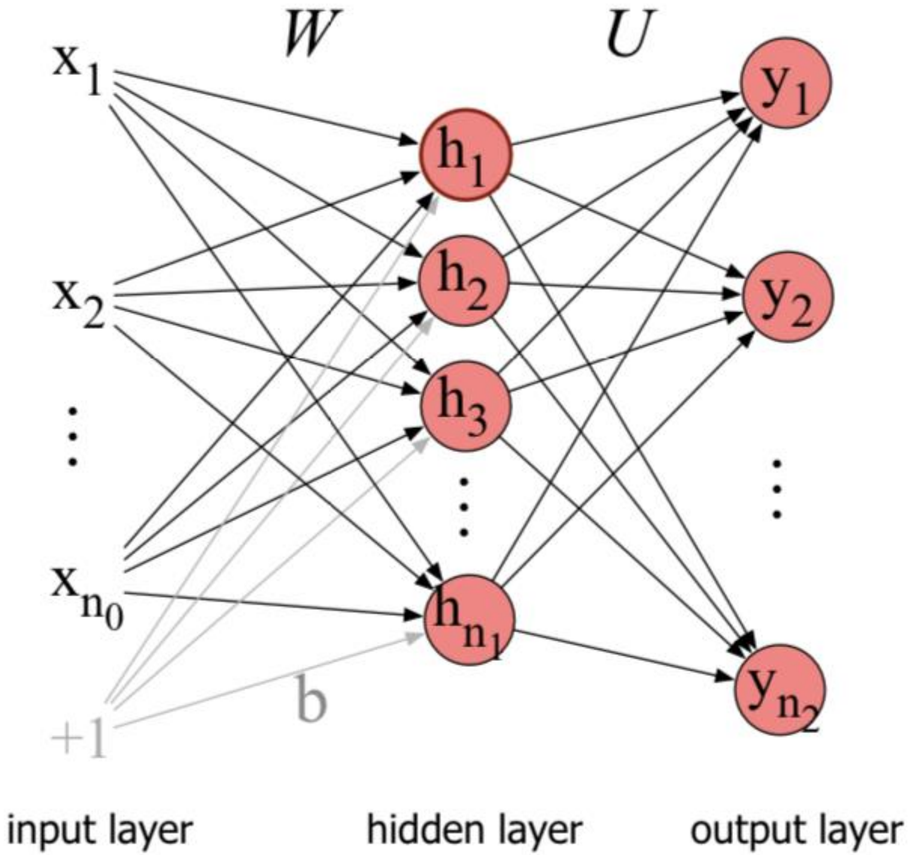
$\vec{y}$: $n_2 \times 1$ vector

$\vec{h}$: $n_1 \times 1$ vector

U: $n_2 \times n_1$

Feed Forward (2 - layered) Neural Network

In gist

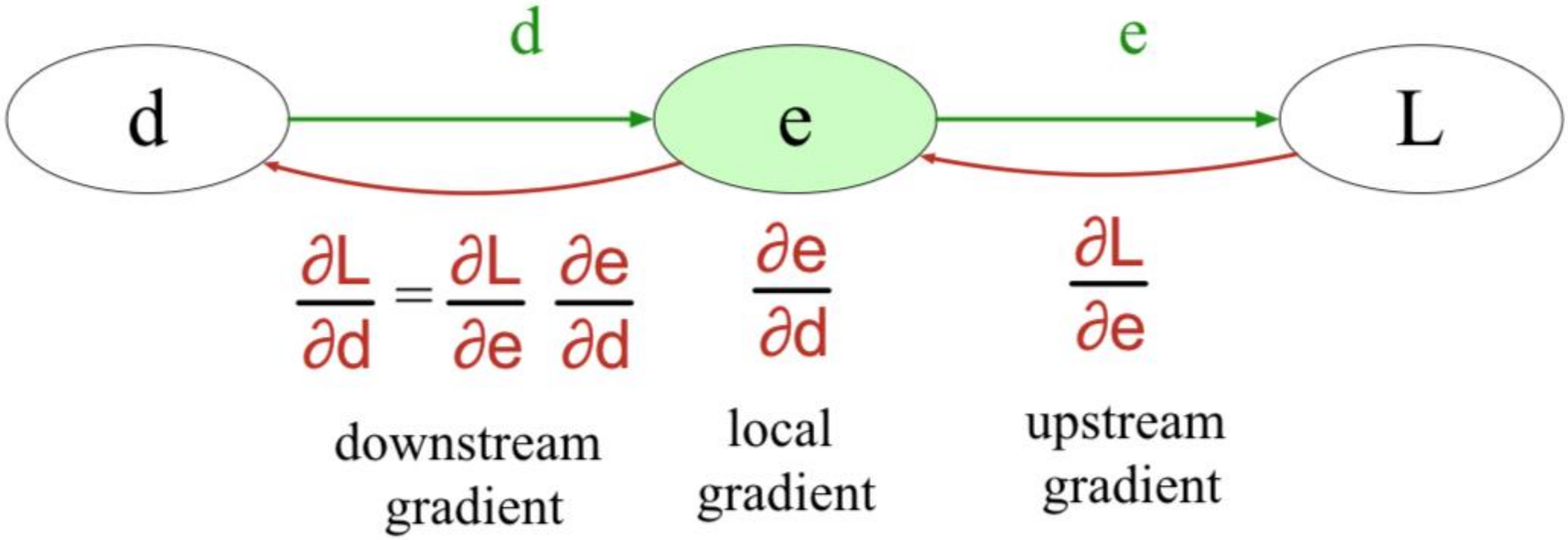


Feed Forward (L - layered) Neural Network

- Generalizing the feed forward NN to L layered

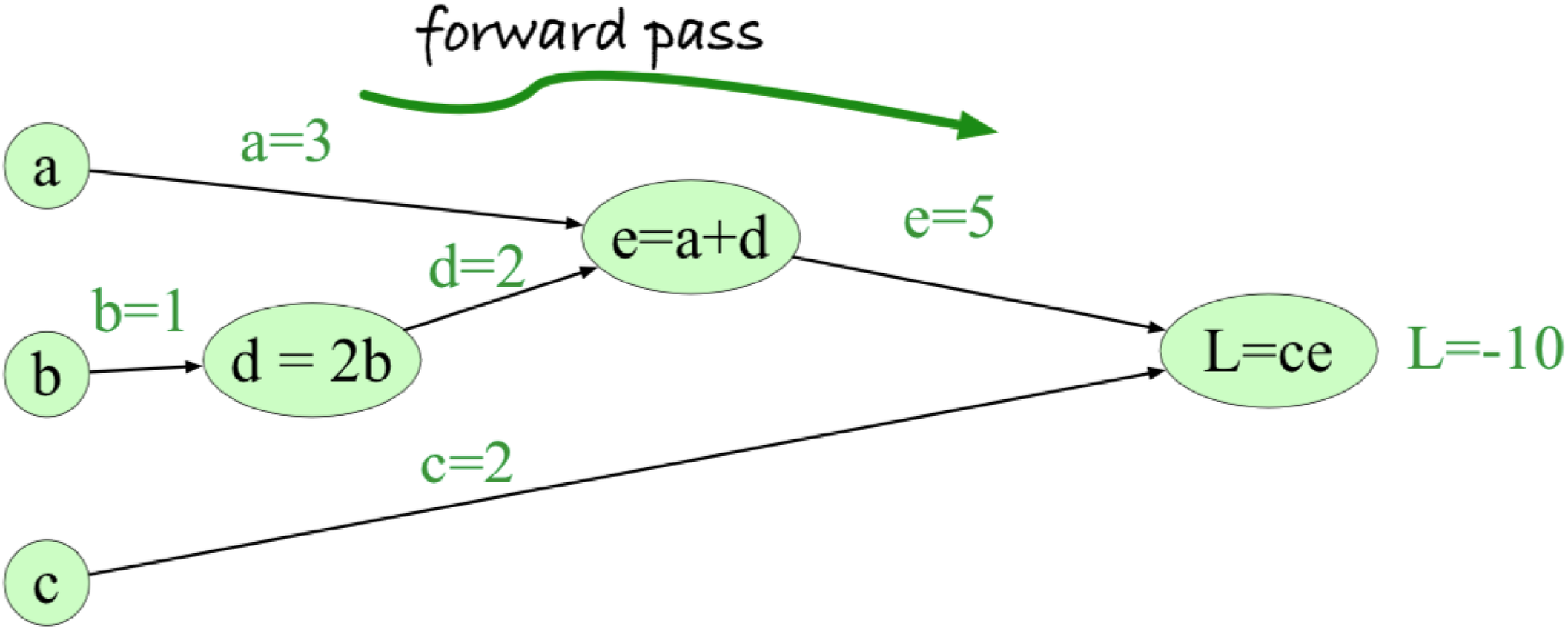
Back Propagation

Forward and Back propagation in simple computational graph



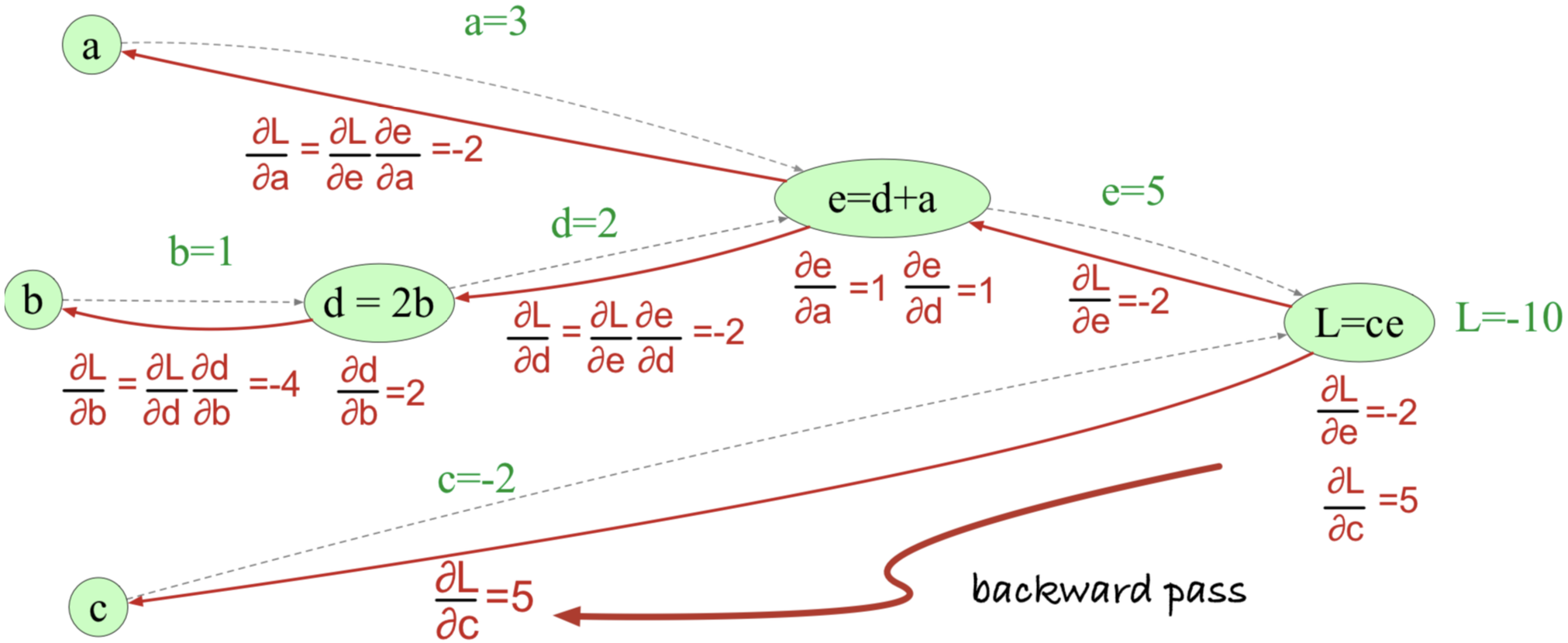
Back Propagation

Let's expand the the concept of computation graph to the complex one.



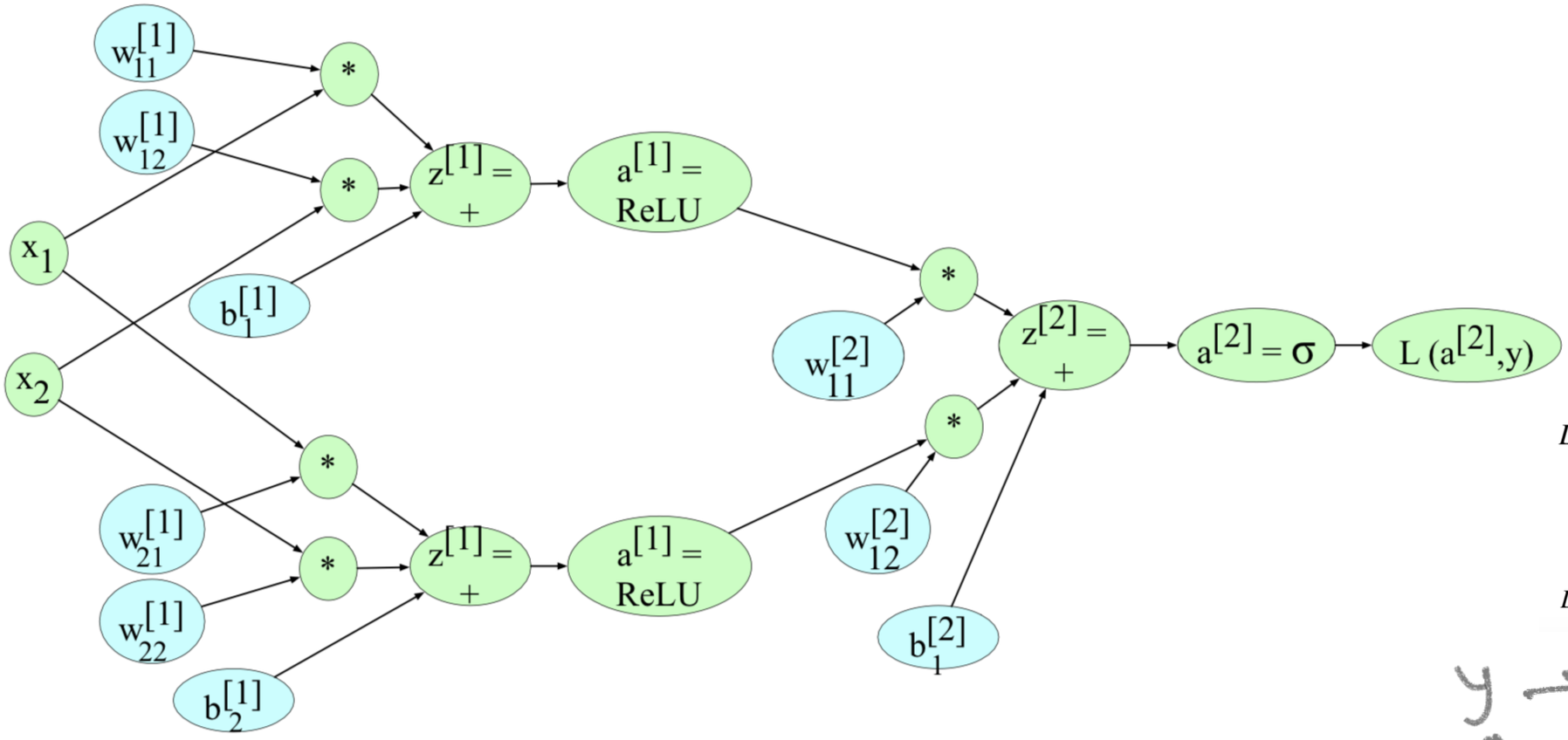
Back Propagation

Now, propagate the Loss in backward direction



Back Propagation

We take only two input features for our ease



$$L_{CE}(\hat{y}, y) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$$

Or we can write

$$L_{CE}(a^{[2]}, y) = -\left[y \log a^{[2]} + (1-y) \log(1-a^{[2]})\right]$$

$y \rightarrow$ actual label

$\hat{y} \rightarrow$ predicted label

Back Propagation

Compute derivative at each unit of each layer and further propagate backward

$$\frac{\partial L}{\partial a^{[2]}} = a^{[2]}(1-a^{[2]})\left(y\frac{\partial \log(a^{[2]})}{\partial a^{[2]}} + (1-y)\frac{\partial \log(1-a^{[2]})}{\partial a^{[2]}}\right)$$

$$= -\left(\left(y\frac{1}{a^{[2]}}\right) + (1-y)\frac{1}{1-a^{[2]}}(-1)\right)$$

$$= -\left(\frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}}\right)$$

Finally, we can use the chain rule:

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a^{[2]}}\frac{\partial a^{[2]}}{\partial z}$$

$$= -\left(\frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}}\right)a^{[2]}(1-a^{[2]})$$

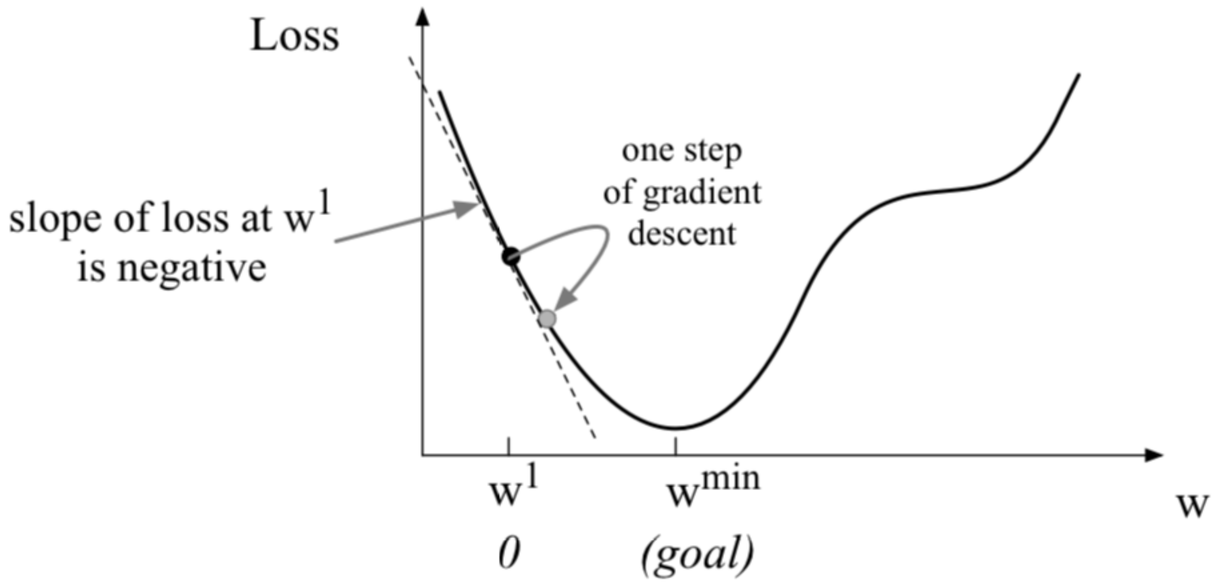
$$= a^{[2]} - y$$

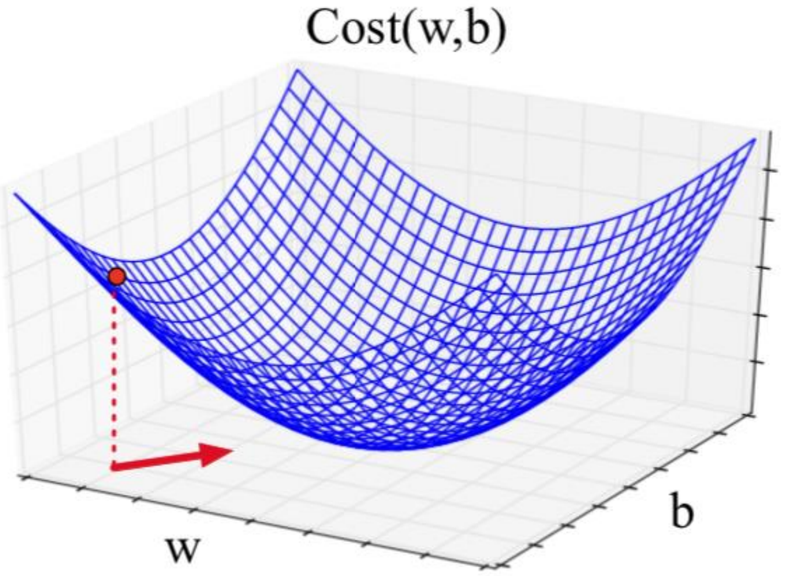
Optimization

Weights & Biases Update Rule

- Optimizers are used for loss minimization and we use update rule as:

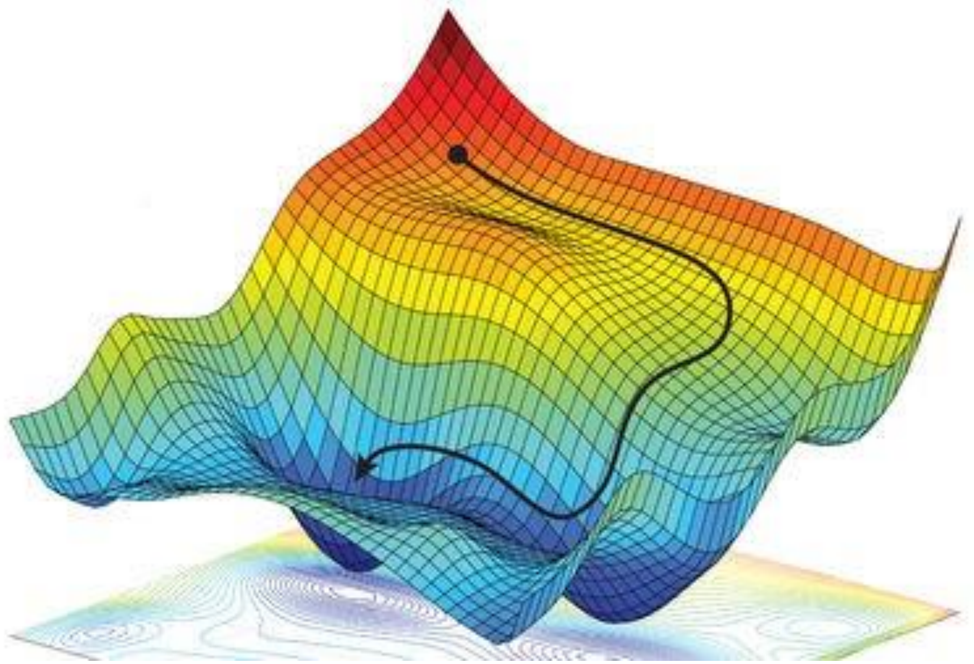
$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

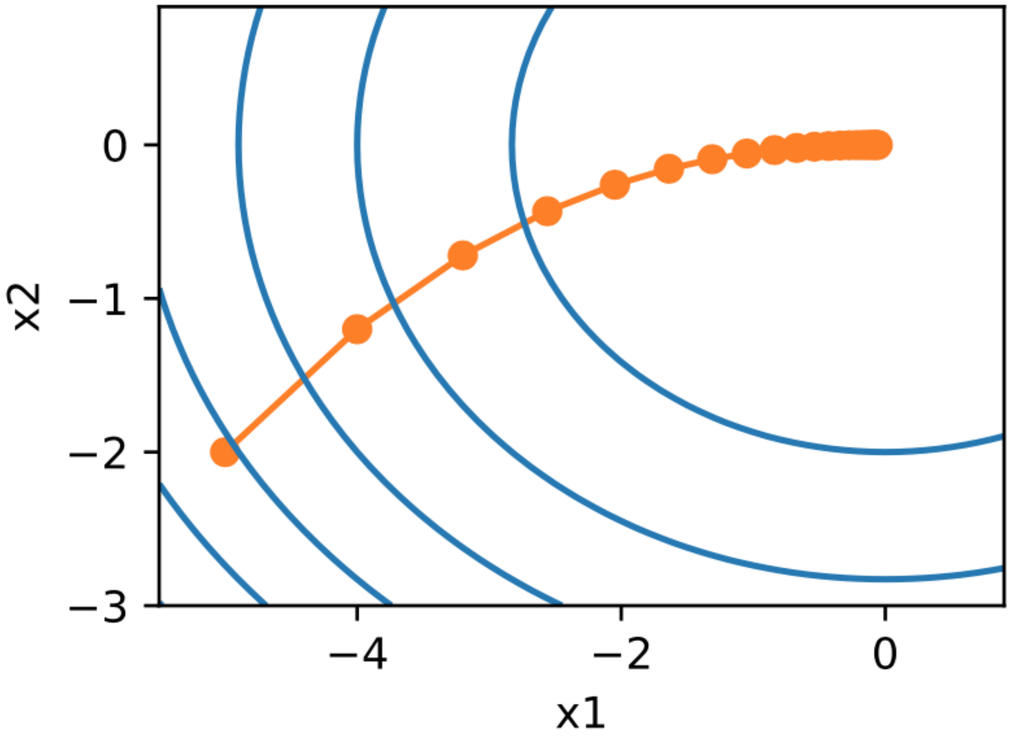




Optimization

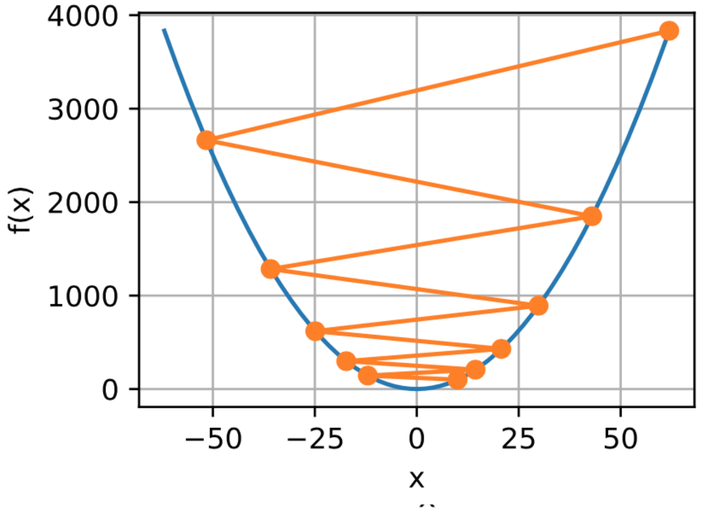
Gradient Descent

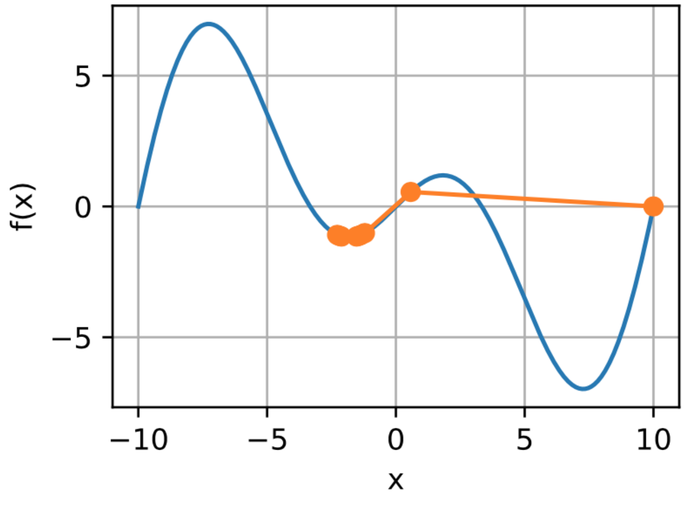


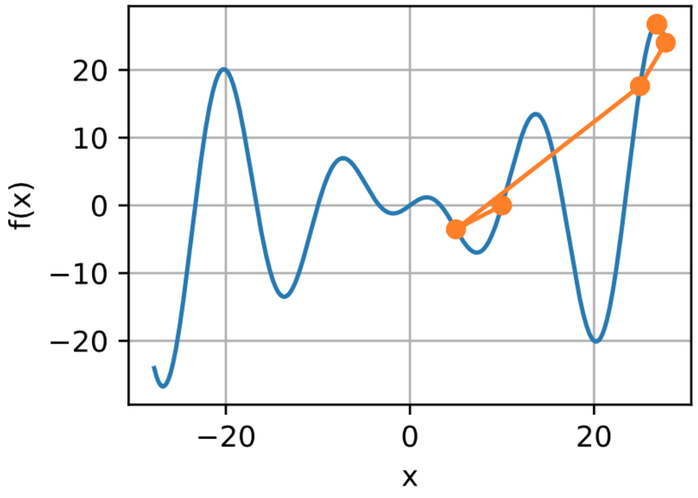


Optimization

Learning Rate

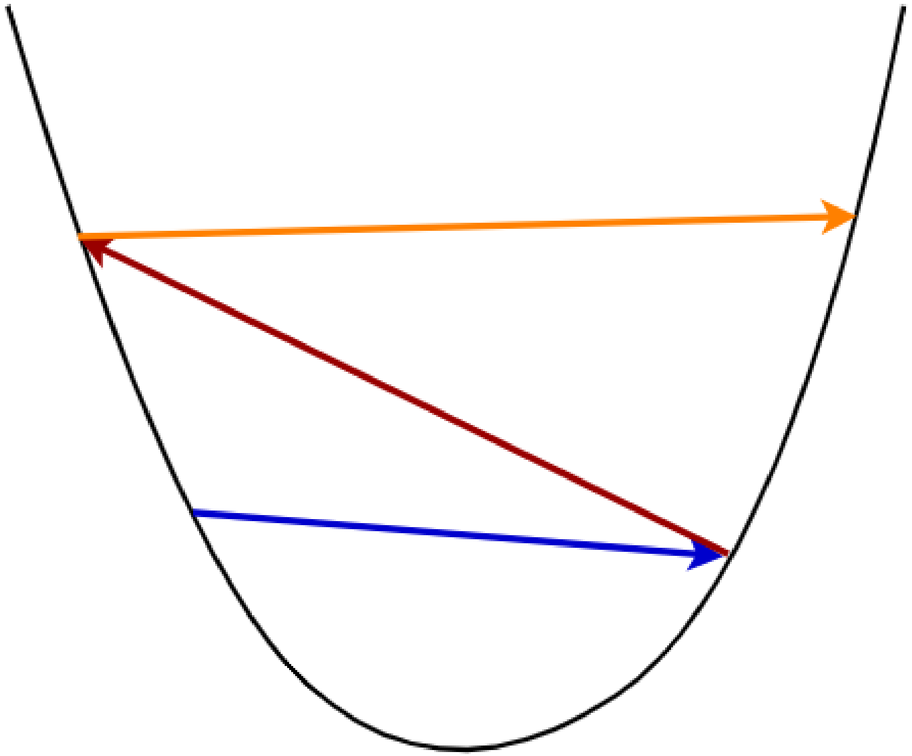




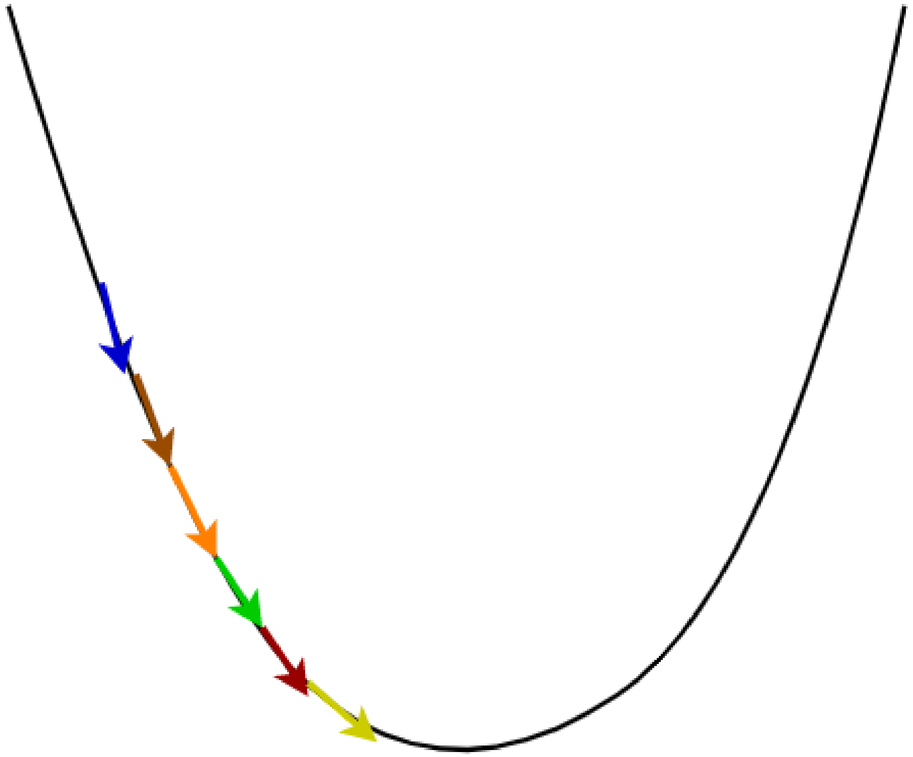


Optimization

Learning Rate

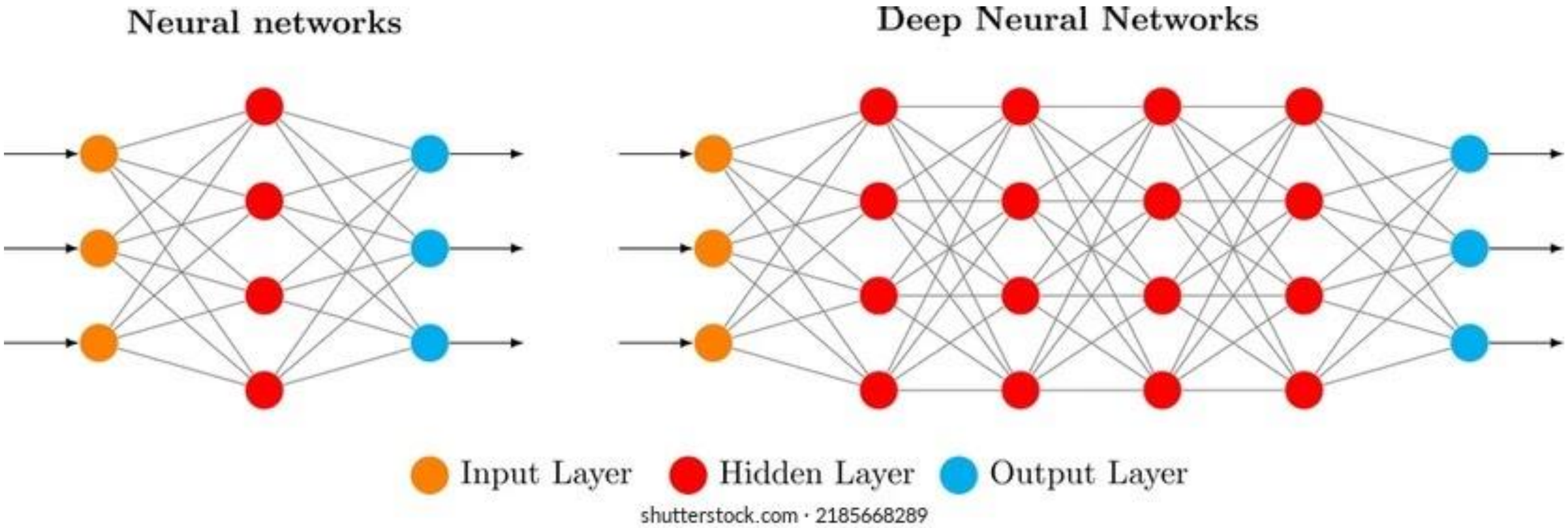


High Learning Rate



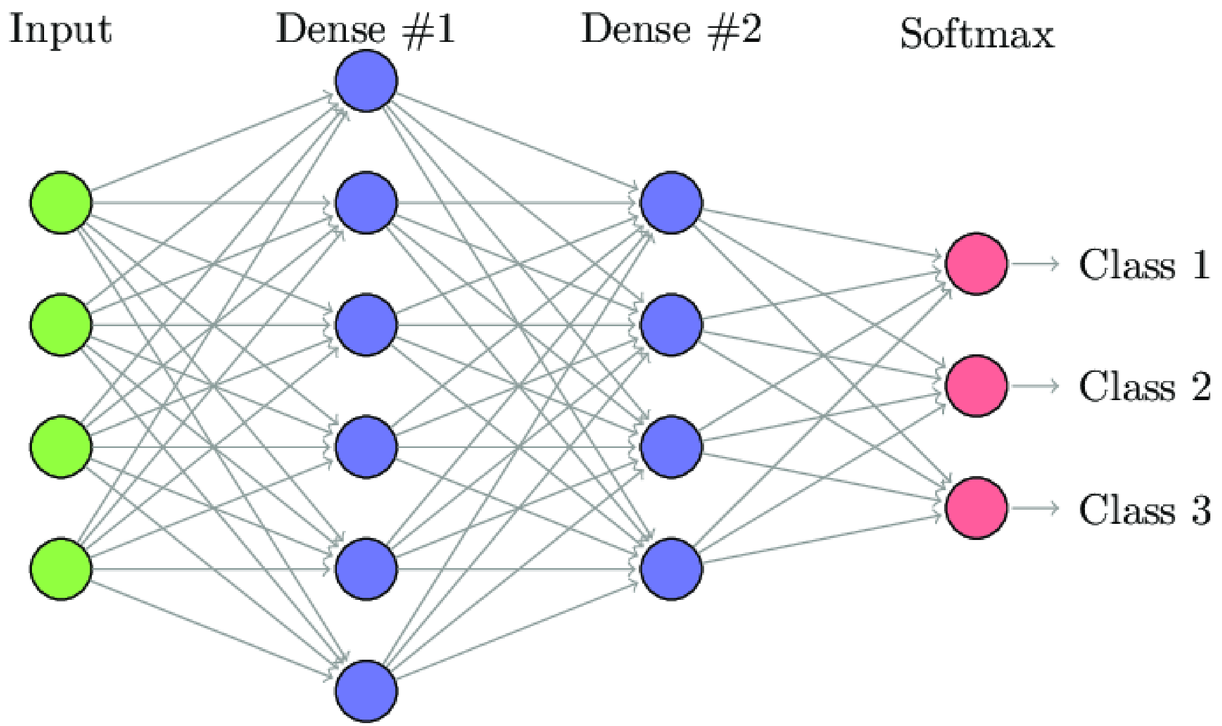
Low Learning Rate

Deep Neural Network



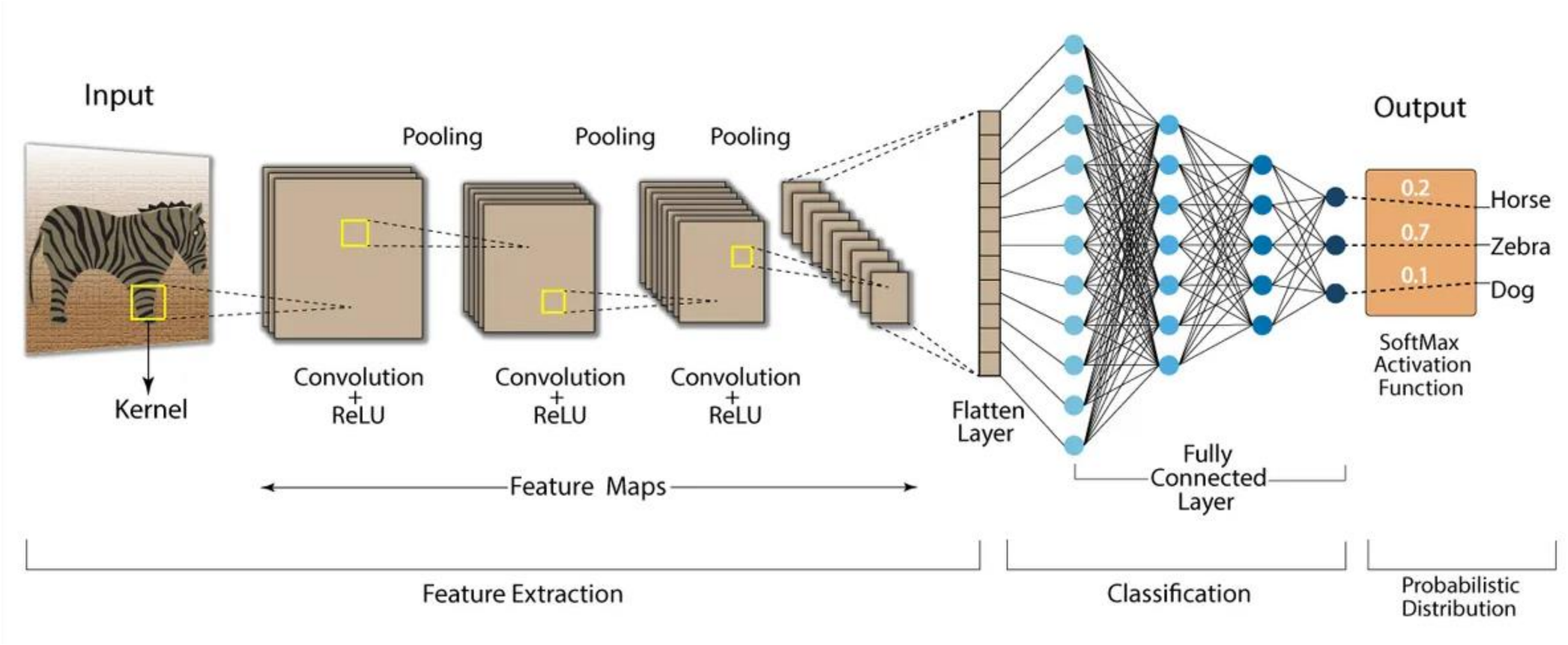
Types of Neural Network

- Fully Connected Neural Network



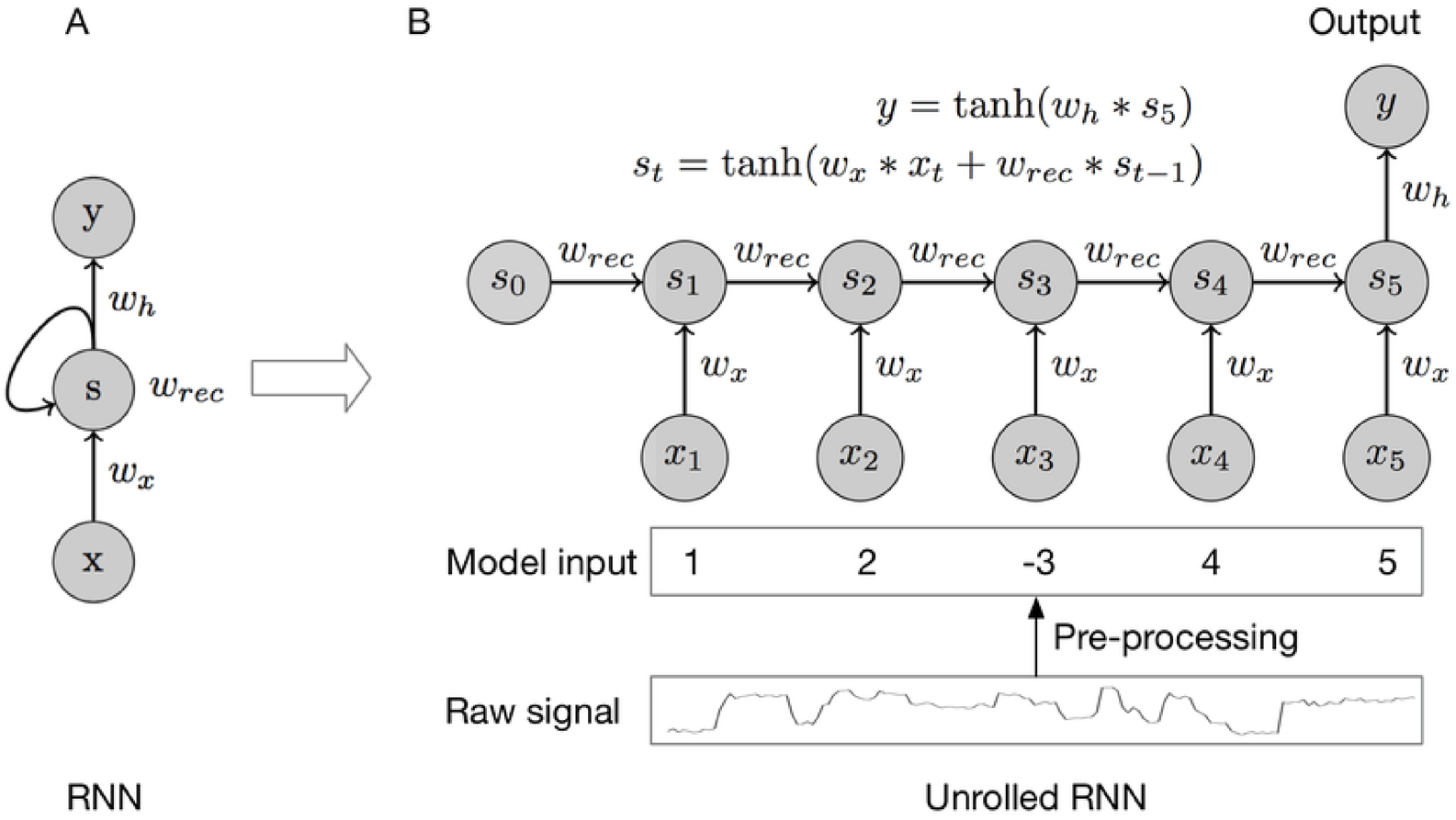
Types of Neural Network

- Convolutional Neural Network



Types of Neural Network

- Recurrent Neural Network

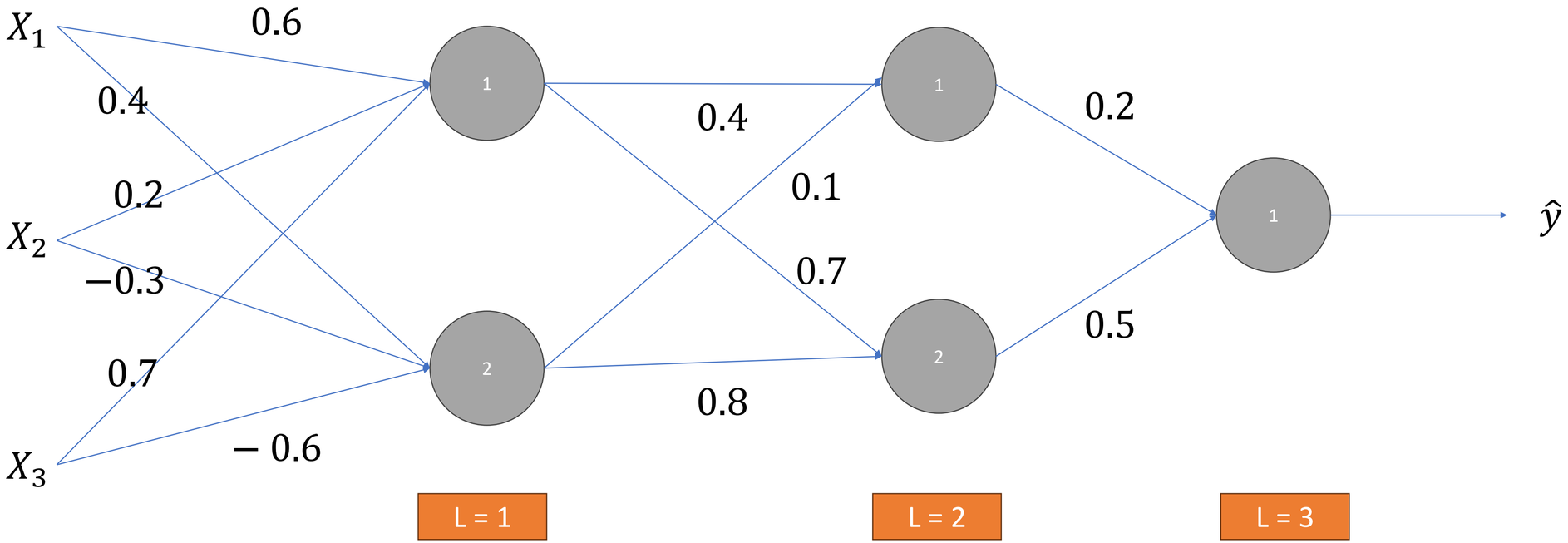


Application of Neural Network

- Signature Verification & Handwriting recognition
- Facial Recognition
- Weather Forecasting
- Voice Recognition System
- Computer Imagination
- Stock Market Prediction
- Language Translation
- Question Answering
- Natural Language Understanding
- Natural Language Generation
- Image Captioning
- Video Commentary
- Gene Segmentation
- Disease Prediction
- Medication Research
- Security Monitoring
- Virtual Learning Platform

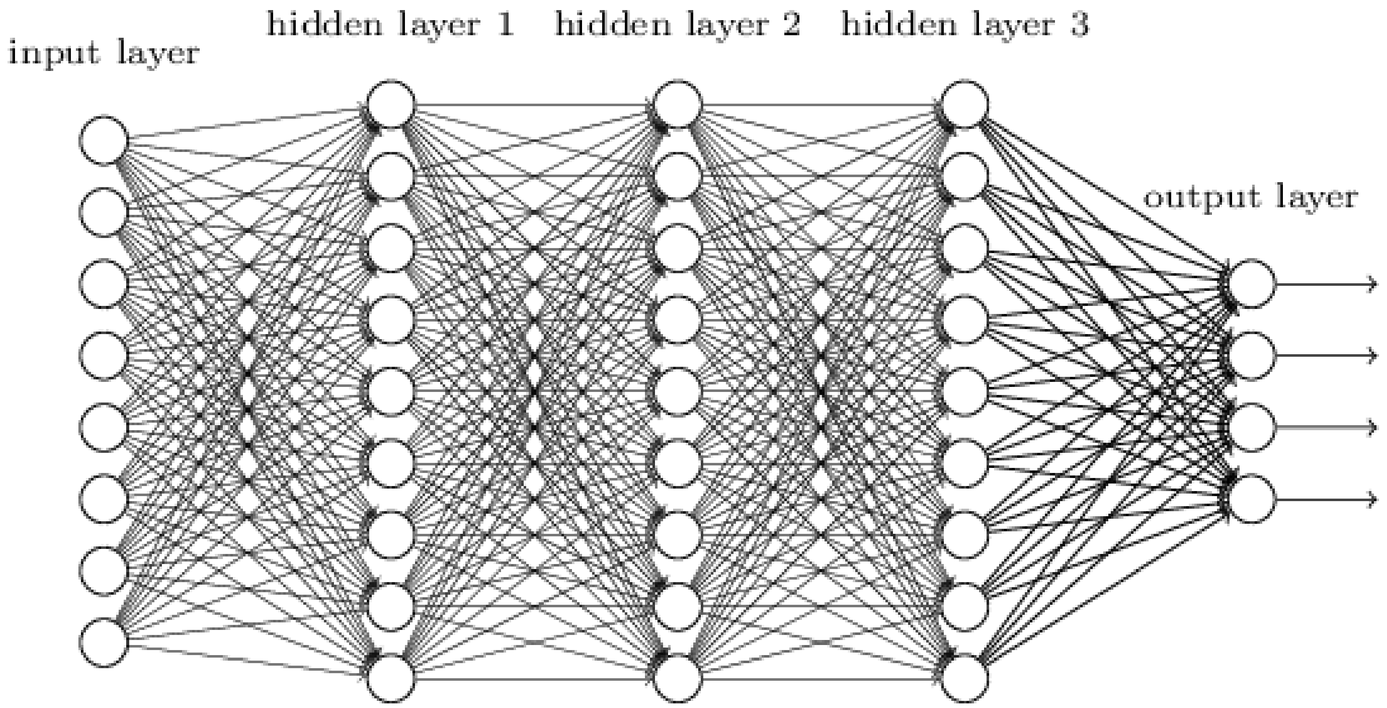
Backpropagation Numerical - 1

- Consider a neural network as below and (1, 0, 1) be the two training examples with the output as 1. Compute the weight updated using back propagation algorithm. Ignore bias in all neuron i.e. b = 0.



Smaller Network: CNN

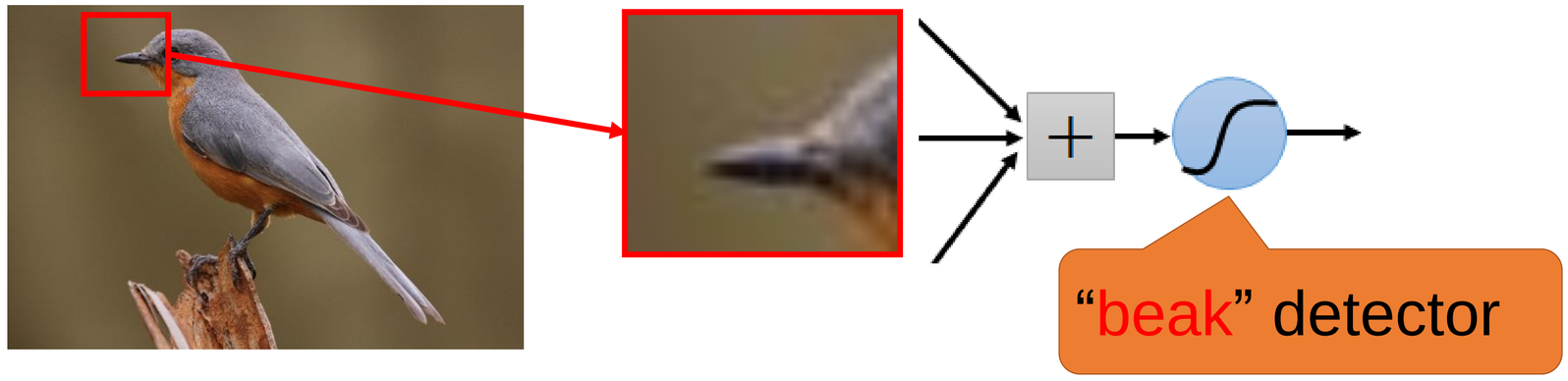
- We know it is good to learn a small model.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?



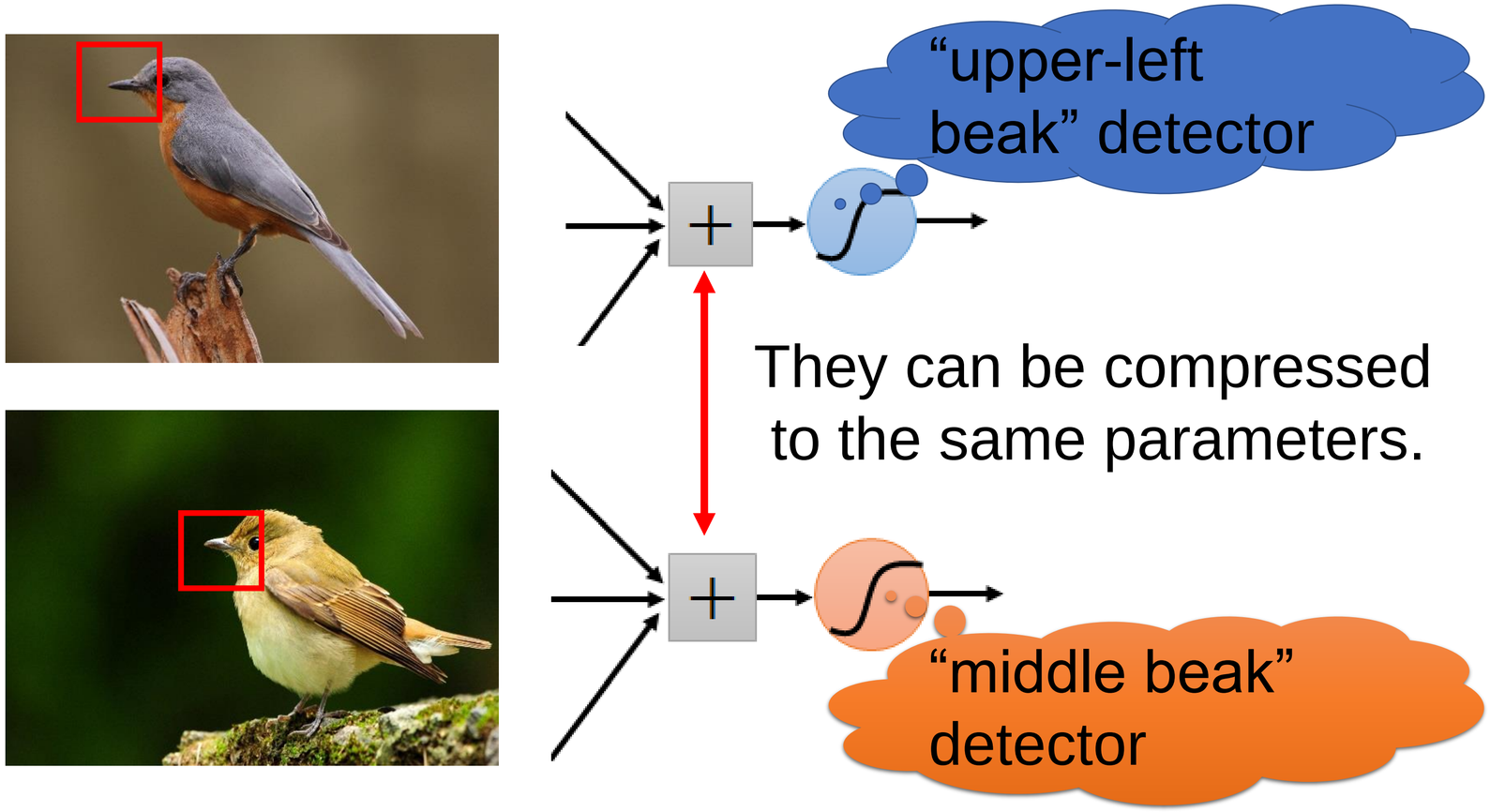
Consider learning an image:

- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters

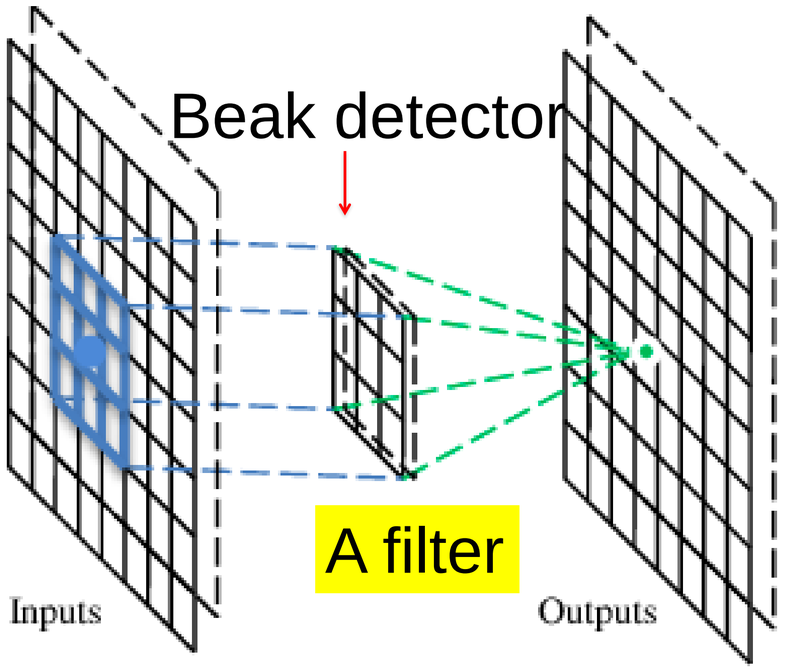


Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors
and each detector must “move around”.



A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

These are the network parameters to be learned.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product → 3

-1

6 x 6 image

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

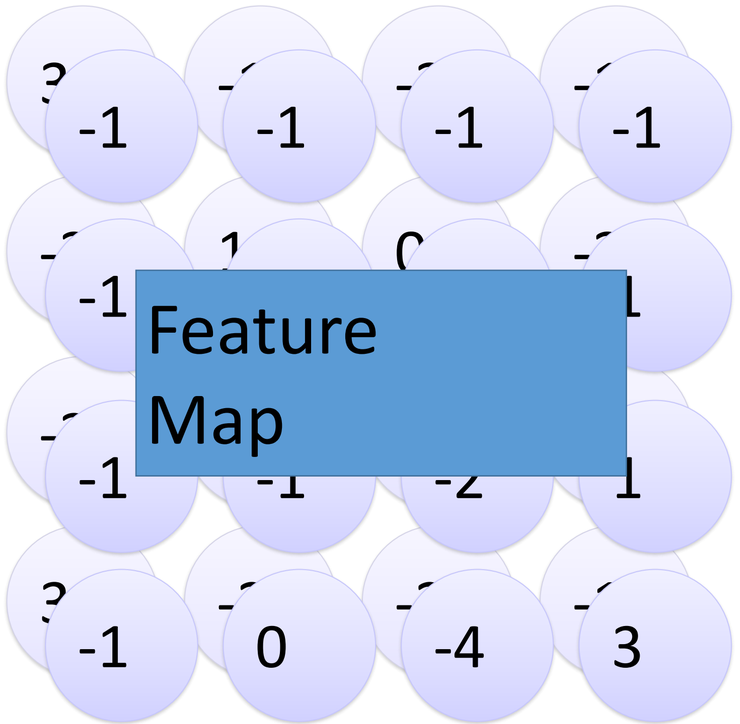
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

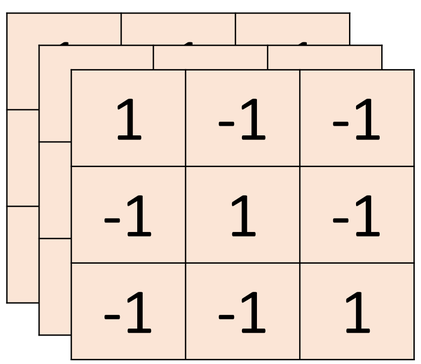
6 x 6 image

Repeat this for each filter

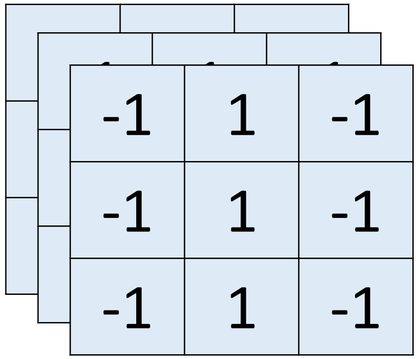


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

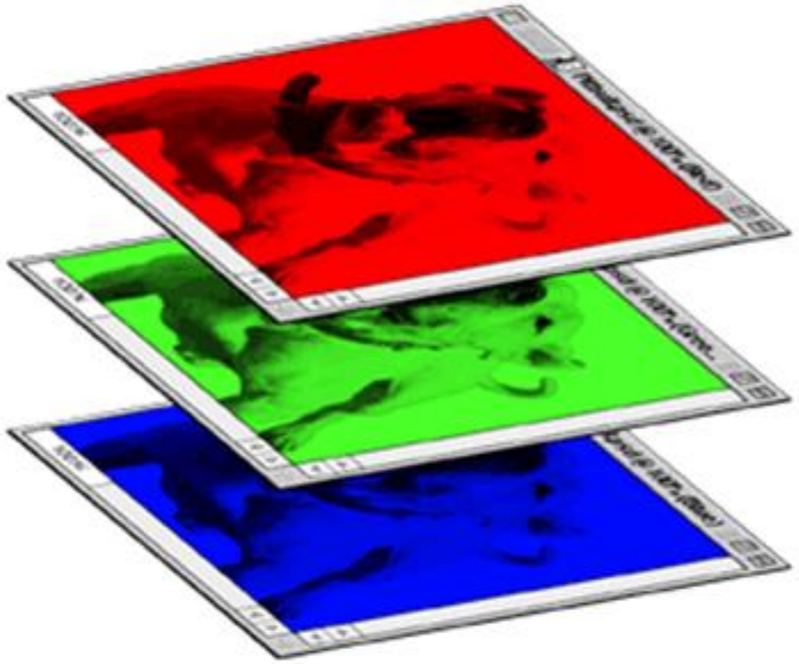


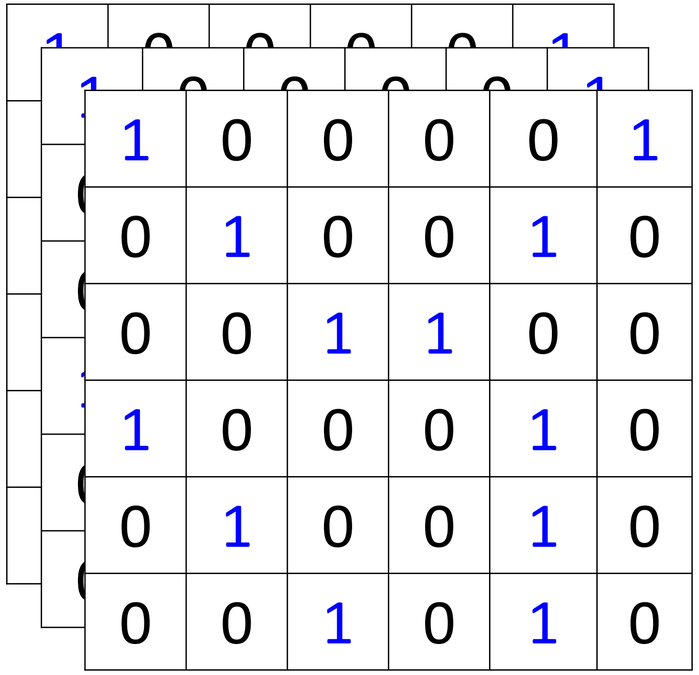
Filter 1



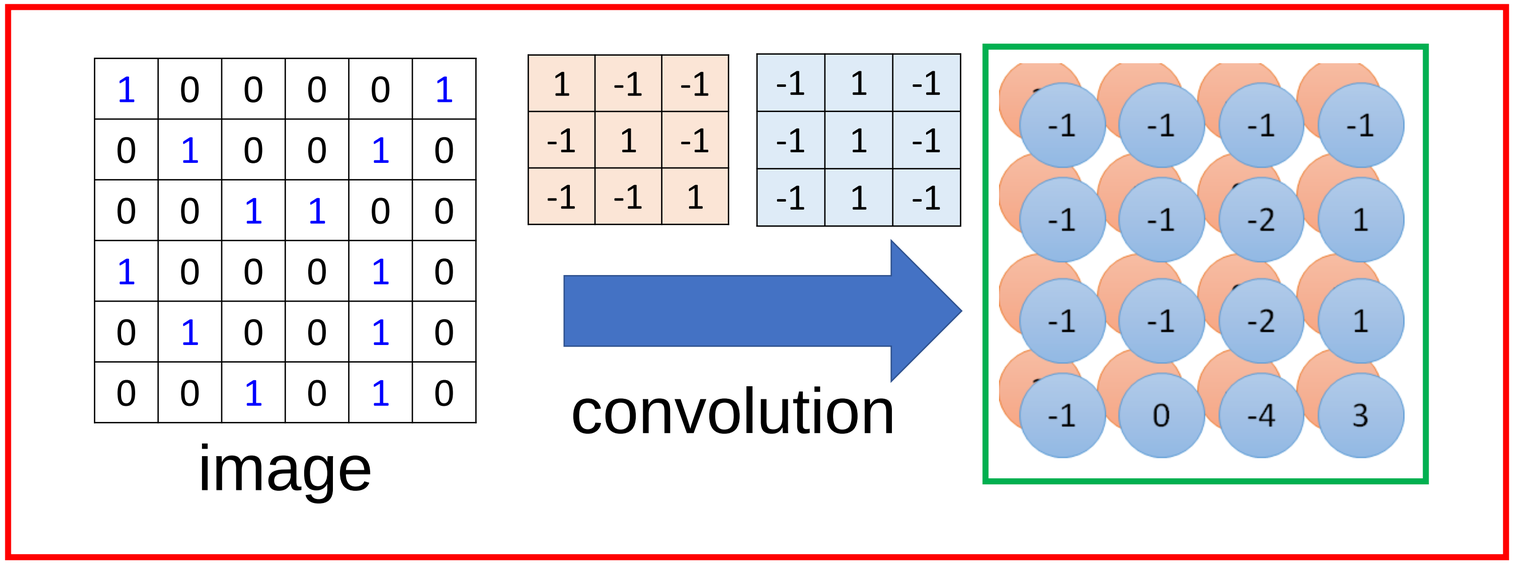
Filter 2

Color image



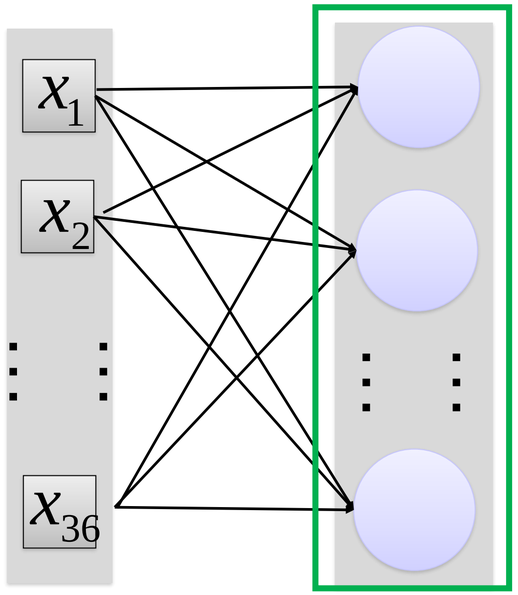


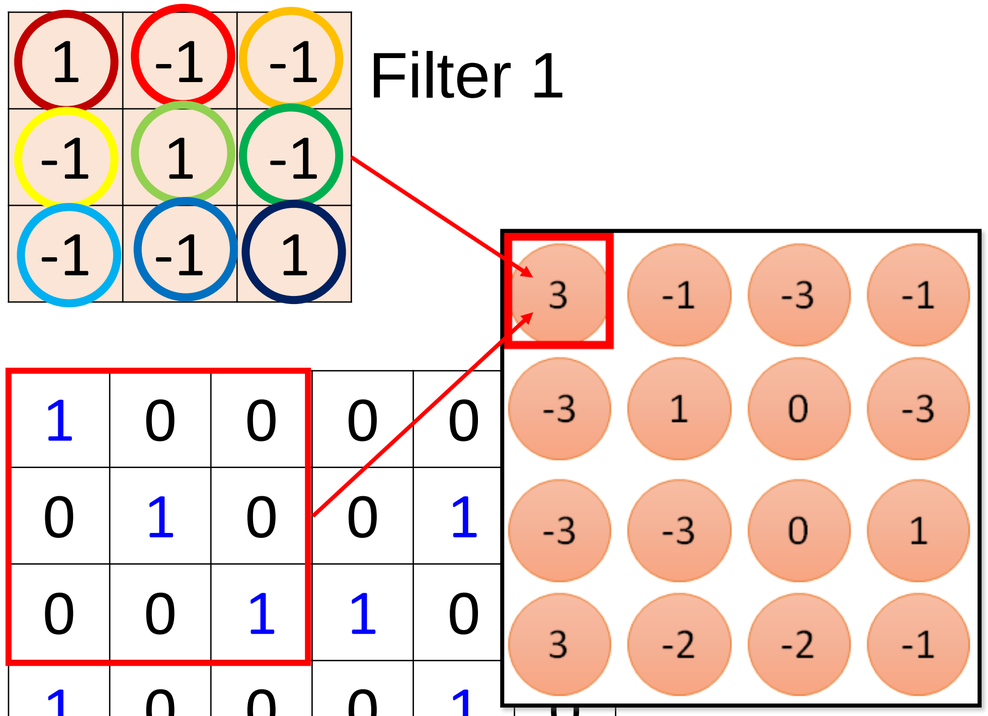
Convolution v.s. Fully Connected



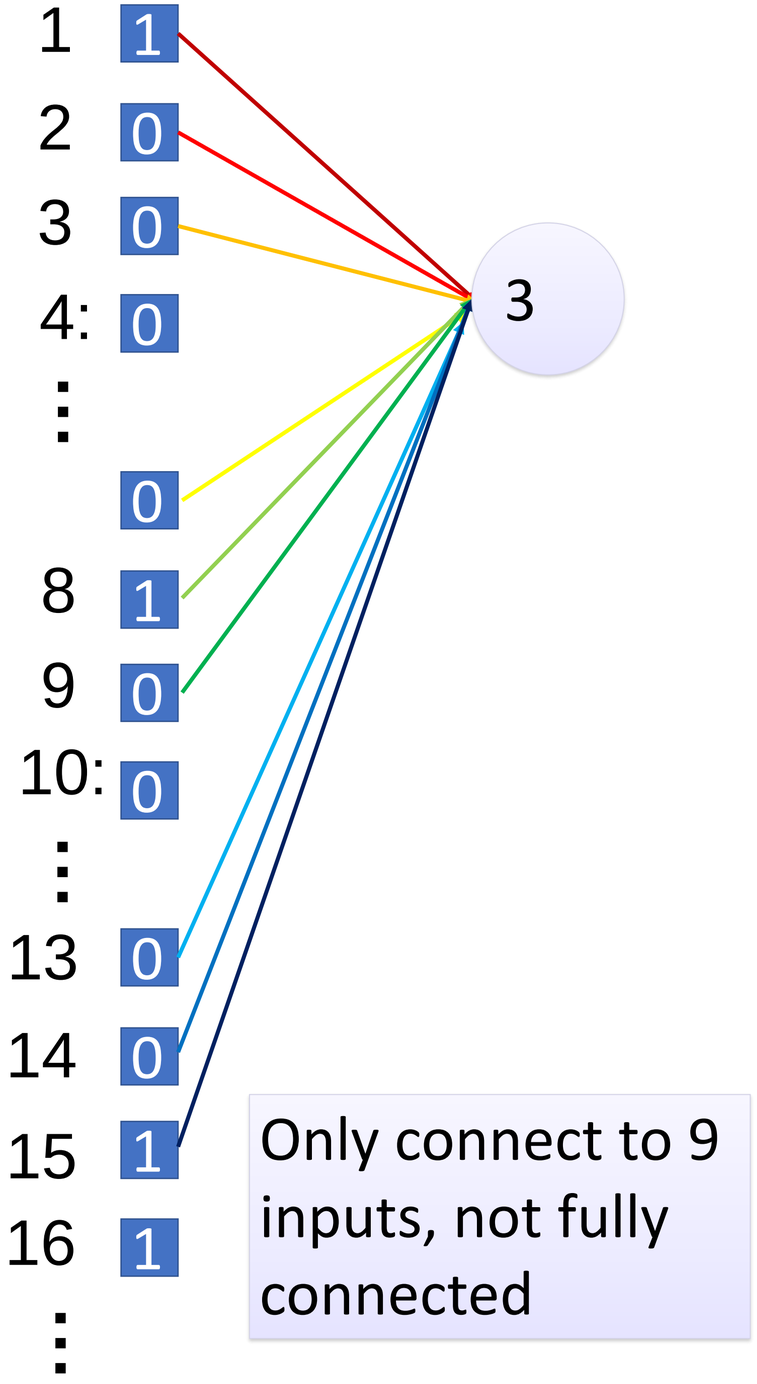
Fully-connected

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

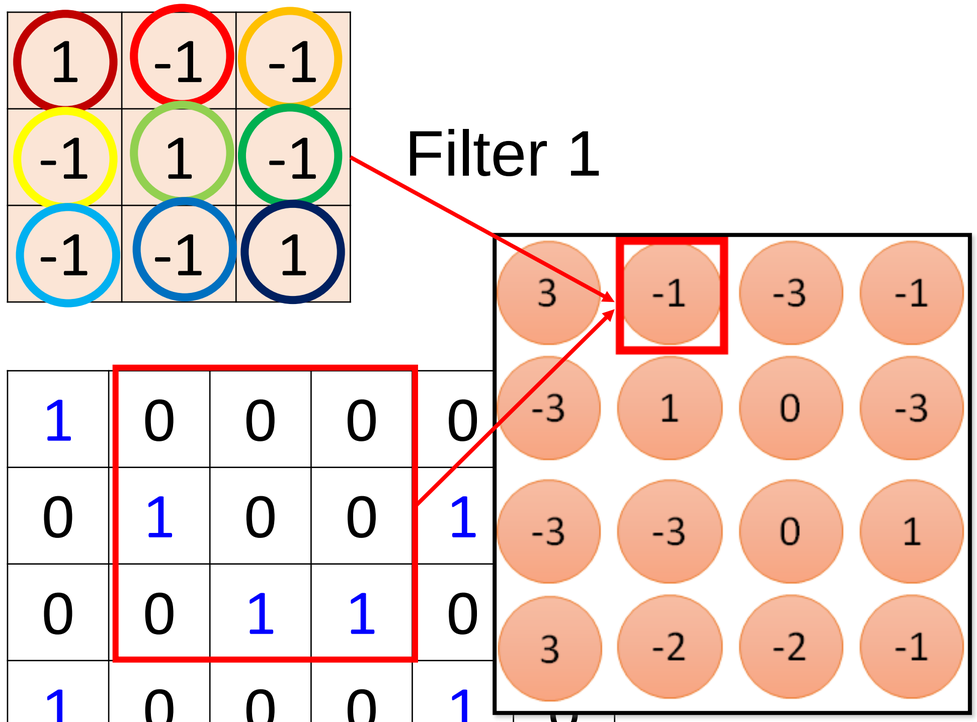




fewer parameters!

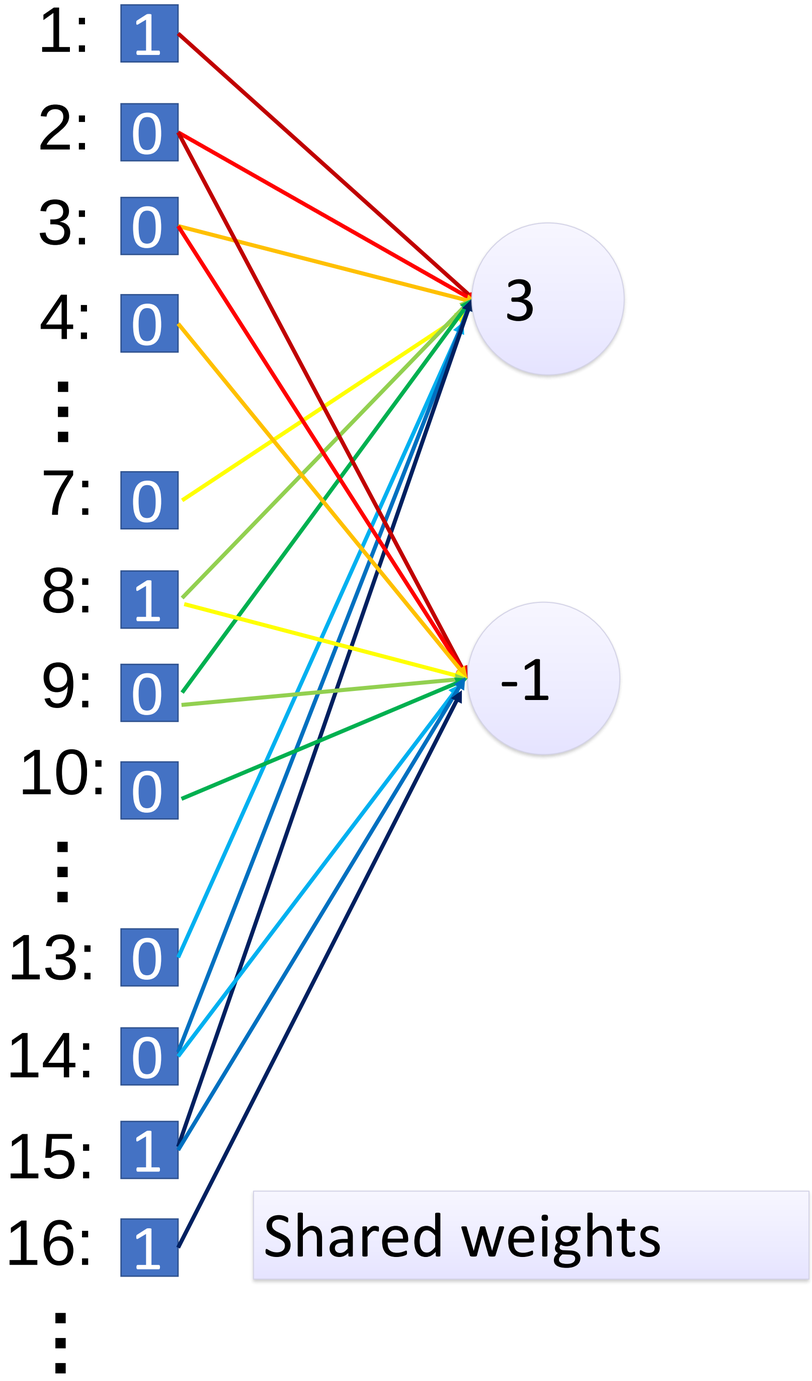


Only connect to 9 inputs, not fully connected

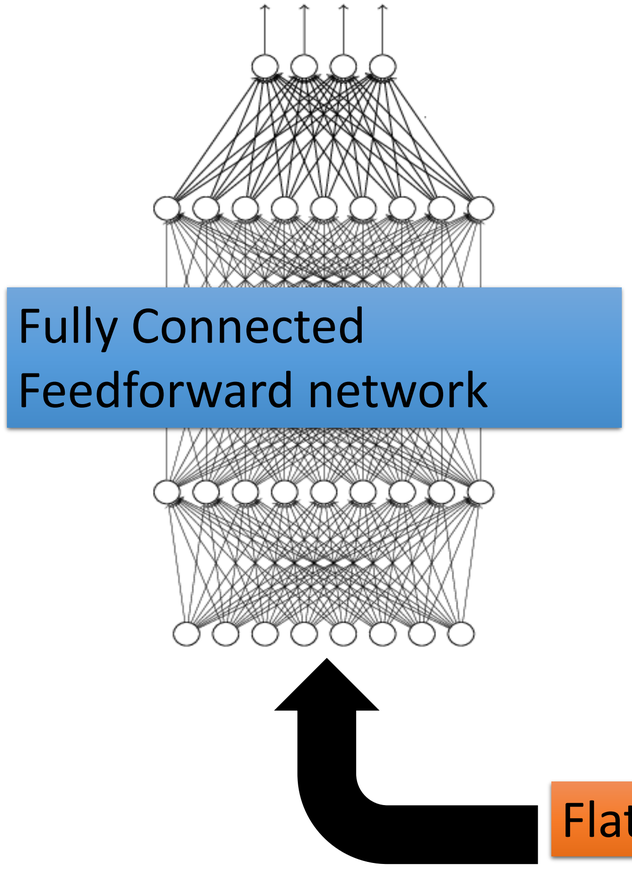


Fewer parameters

Even fewer parameters



The whole CNN



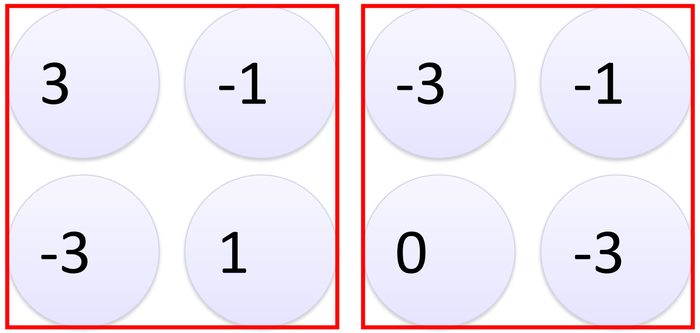
Max Pooling

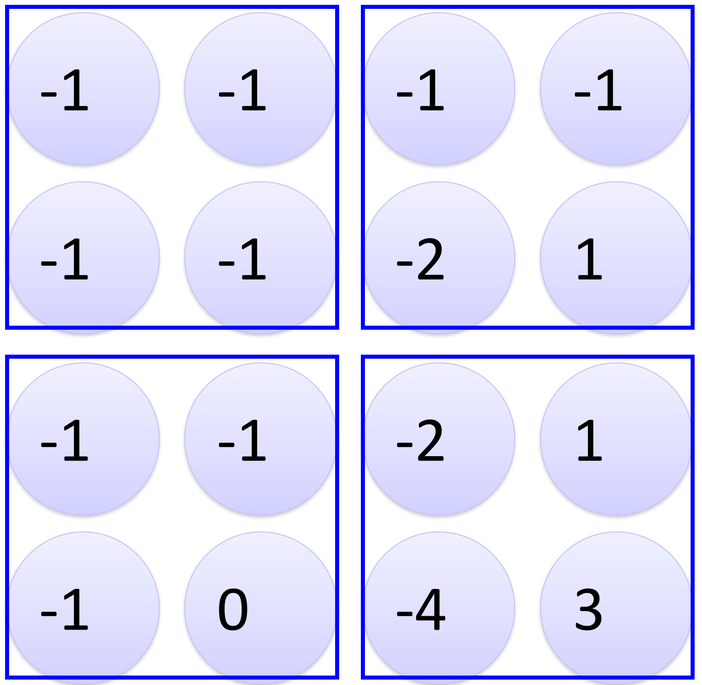
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

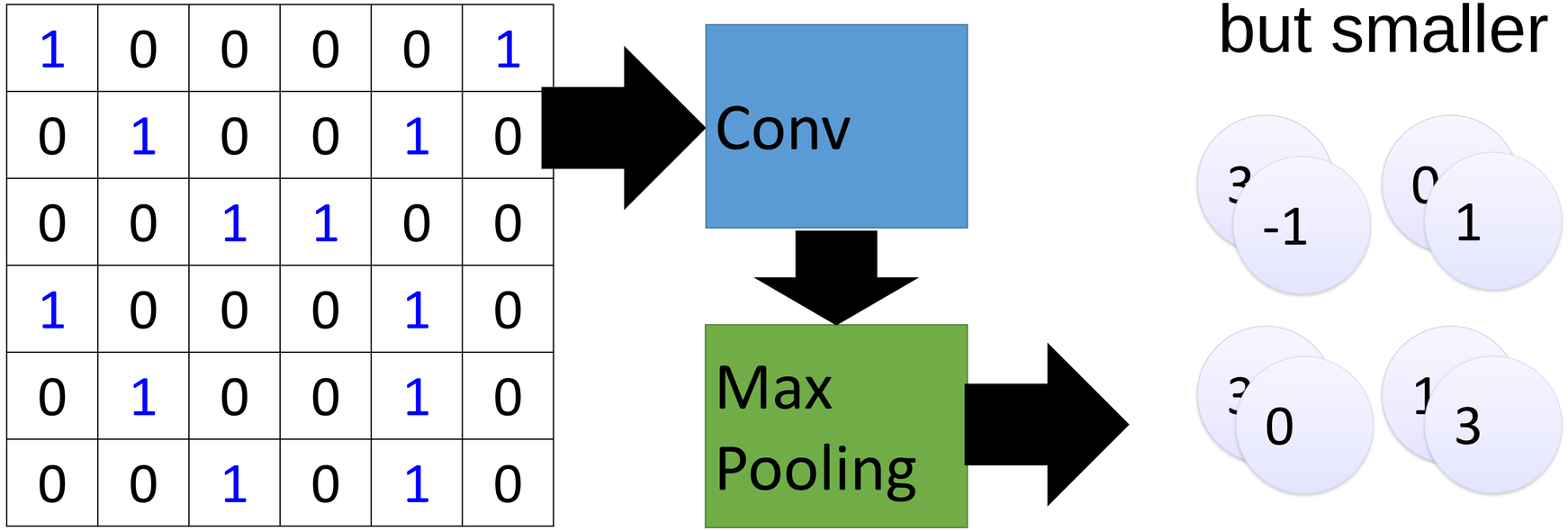
We can subsample the pixels to make image smaller

→ fewer parameters to characterize the image

A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

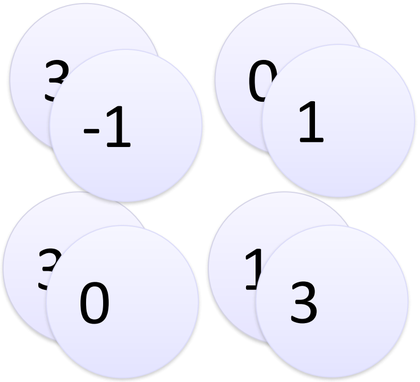
Max Pooling



6 x 6 image

Each filter is a channel

The whole CNN



Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Smaller than the original image

The number of channels is the number of filters

The whole CNN

Convolution

Max Pooling

A new image

Convolution

Max Pooling

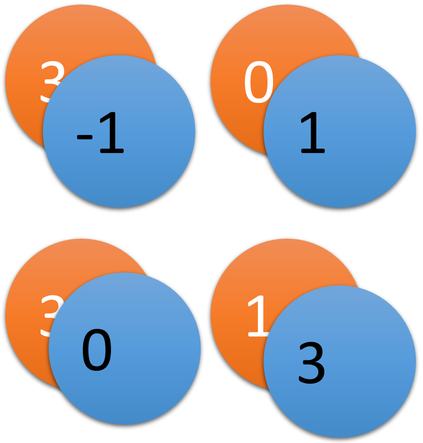
A new image

Flattened

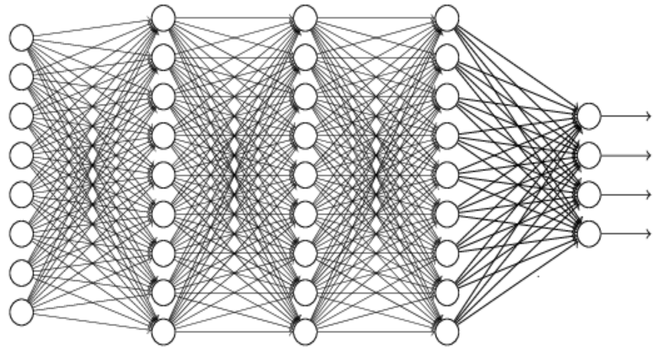
Fully Connected
Feedforward network

cat dog ……

Flattening

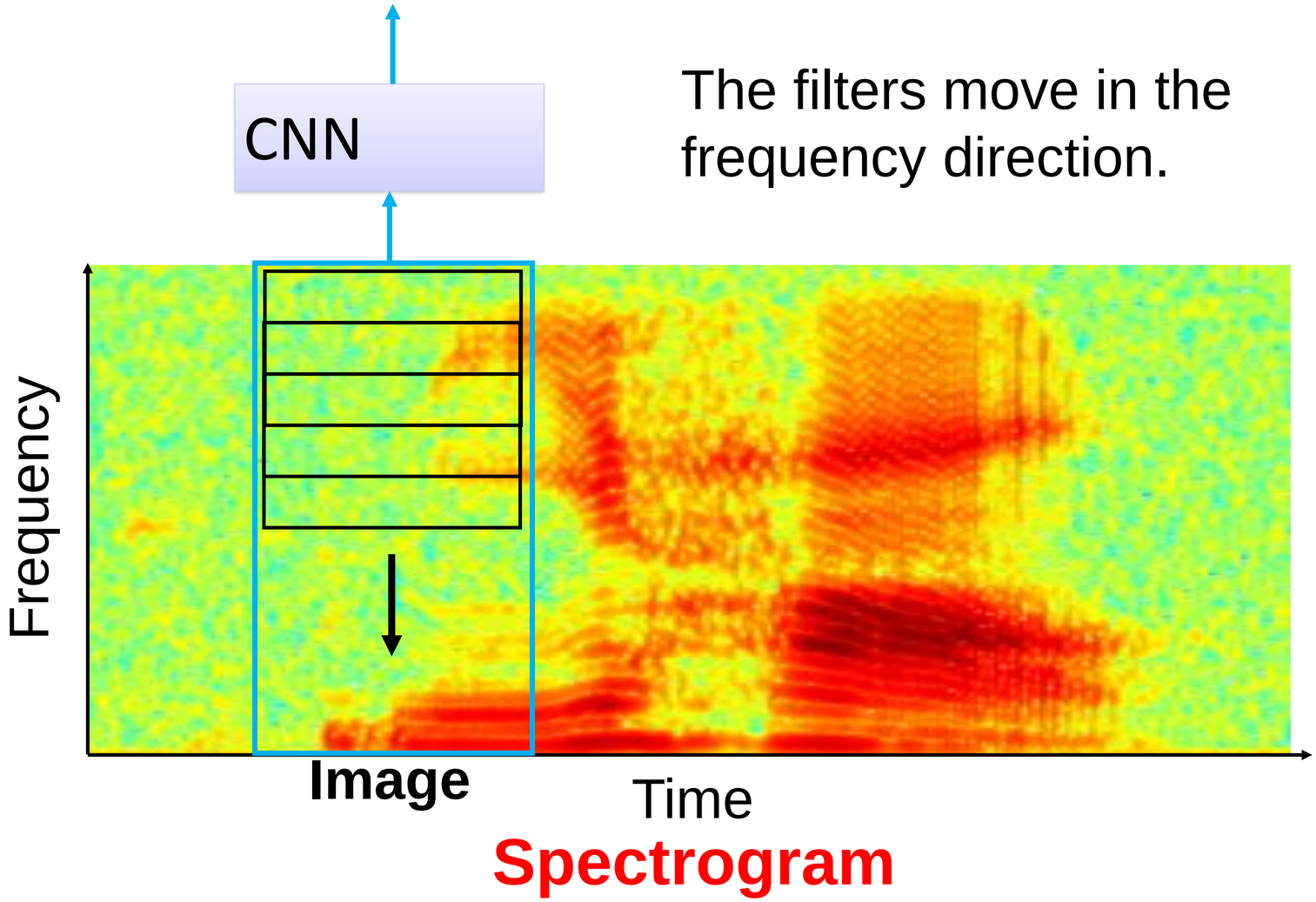


Flattened

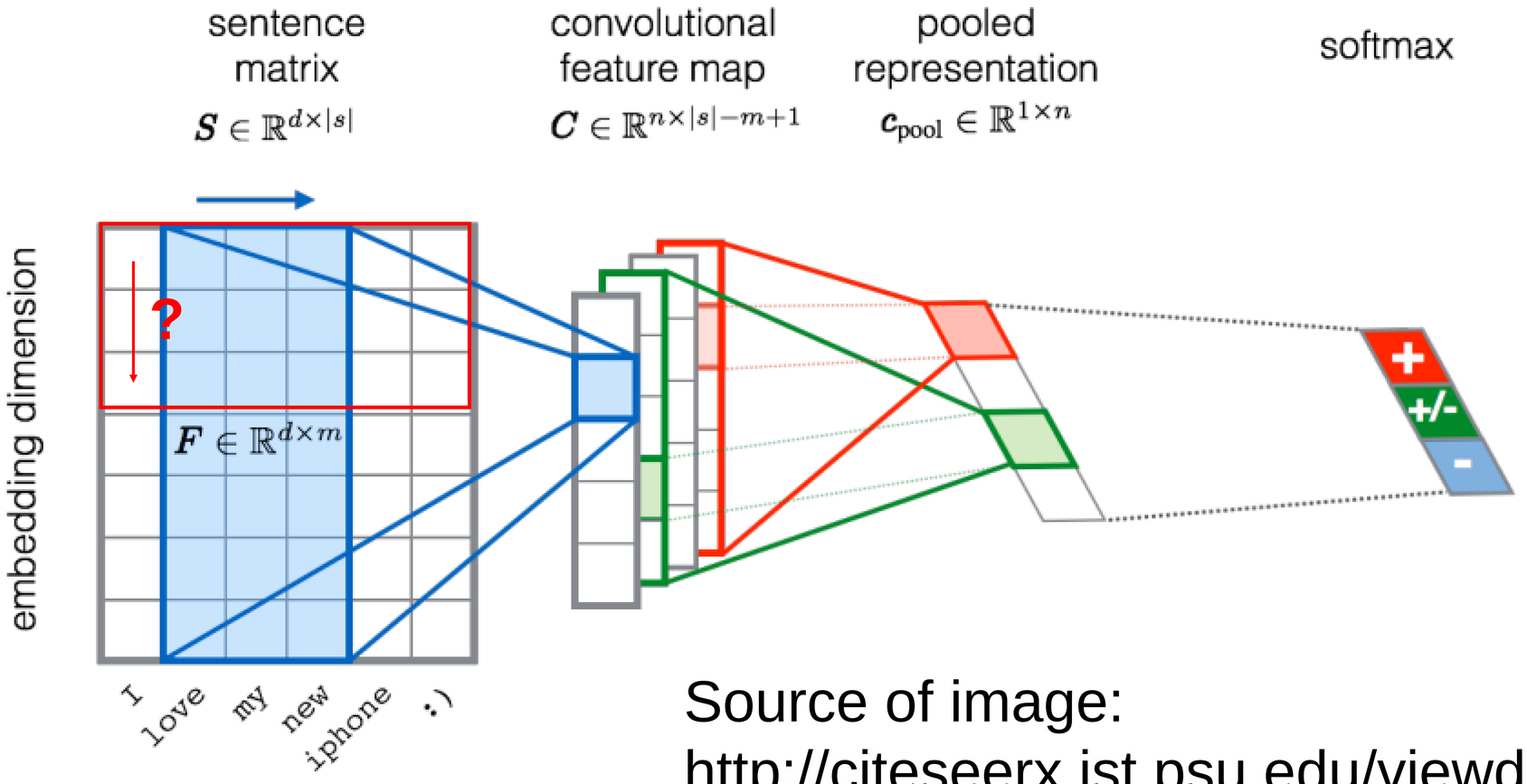


Fully Connected
Feedforward network

CNN in speech recognition



CNN in text classification



Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

References

- CS 230 - Convolutional Neural Networks Cheatsheet (stanford.edu)

CNN Numerical - 1

1. An input image has been converted into a matrix of size 12 X 12 along with a filter of size 3 X 3 with a Stride of 1. Determine the size of the convoluted matrix.

Solution:

Size of Image = 12 x 12 i.e. h = w = 12

Size of filter = 3 x 3 i.e. f = 3

Stride (s) = 1 and padding (p) = 0

Then,

$$h_{new} = \frac{h_{old} - f + 2p}{s} + 1 = \frac{12 - 3 + 0}{1} + 1 = 10$$

Thus, the resulting convoluted matrix is 10 x 10.

CNN Numerical - 2

2. Consider a Convolutional Neural Network having three different convolutional layers in its architecture as –

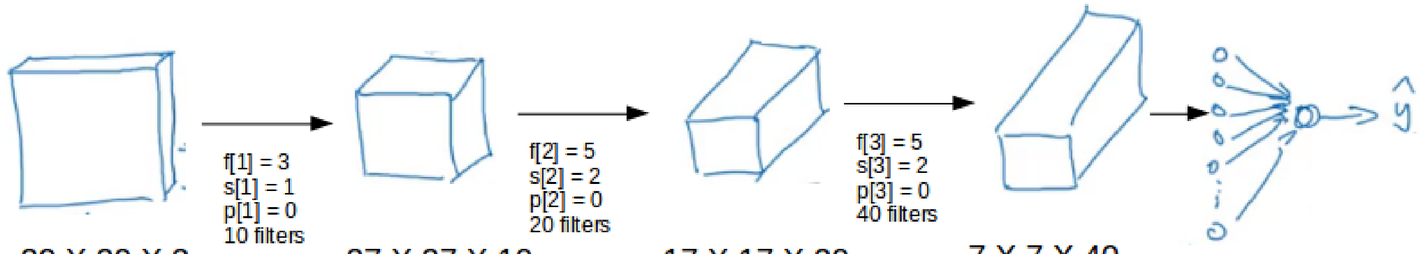
Layer-1: Filter Size – 3 X 3, Number of Filters – 10, Stride – 1, Padding – 0

Layer-2: Filter Size – 5 X 5, Number of Filters – 20, Stride – 2, Padding – 0

Layer-3: Filter Size – 5 X5 , Number of Filters – 40, Stride – 2, Padding – 0

If we give the input a 3-D image to the network of dimension 39 X 39, then determine the dimension of the vector after passing through a fully connected layer in the architecture.

Solution:



nh[0] = nw[0] = 39
nc[0] = 3

nh[1] = nw[1] = 37
nc[1] = 10

nh[2] = nw[2] = 17
nc[1] = 20

nh[1] = nw[1] = (n + 2p – f) / s + 1
nh[1] = nw[1] = (39 + 0 – 3) / 1 + 1
nh[1] = nw[1] = 37

CNN Numerical - 3

Execute a convolution using below given filter in any image of size 6 x 6. And then apply average pooling of size 2 x 2 to the Output.

$$\text{filter} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Now, apply padding of size 1 to the image on all four side and again apply the above given filter. Then apply max pooling of size 3 x 3 on the output.

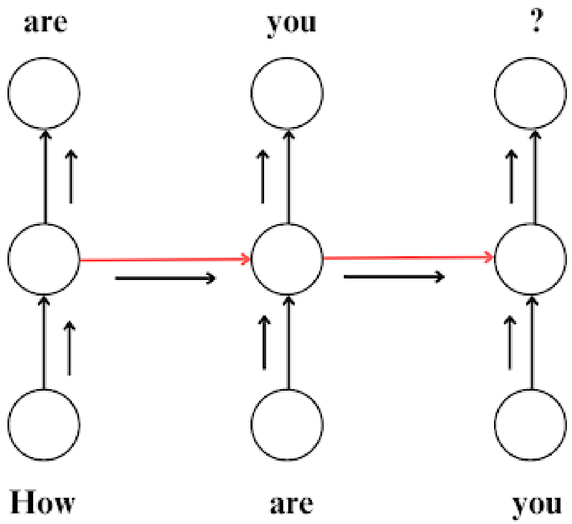
CNN Numerical - 4

- Consider a 4 bit grey level image with resolution 6 x 6 and a 3 x 3 filter whose all diagonal elements are 1 and non-diagonal elements are zero. Compute feature map and then compute pooled feature map using 3 x 3 window. Use Max Pooling.

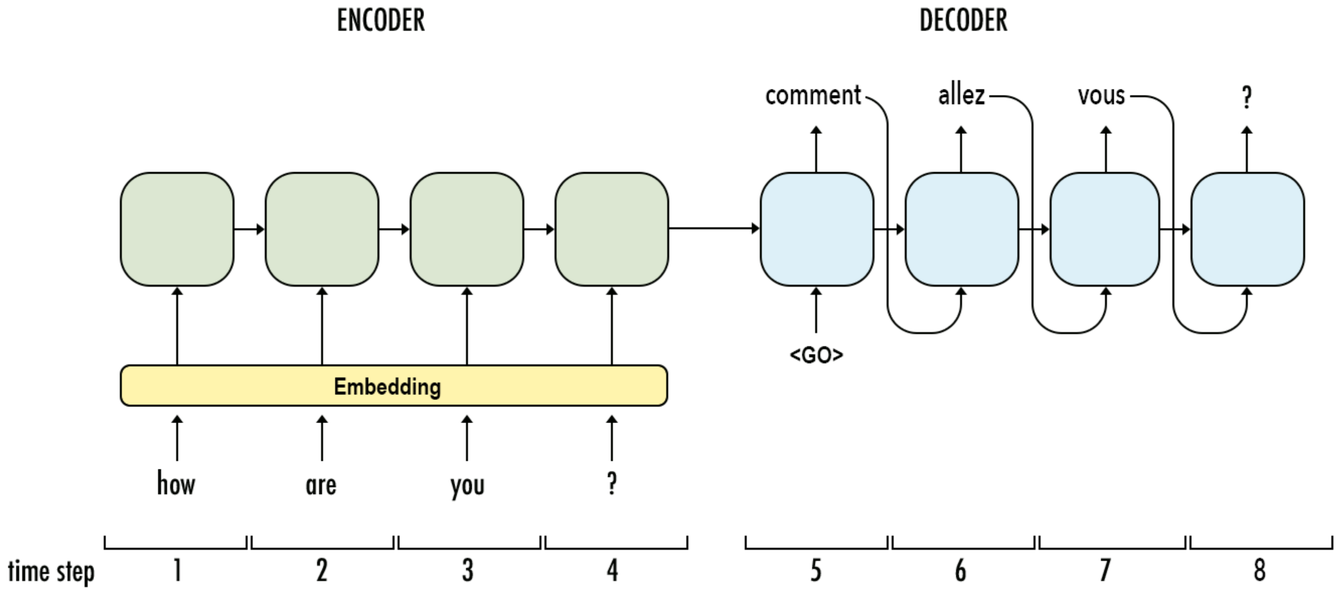
Recurrent Neural Network

- A recurrent neural network (RNN) is a type of artificial neural network which uses data that occurs in sequence.
- These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate.
- Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn.
- They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output.

Recurrent Neural Network

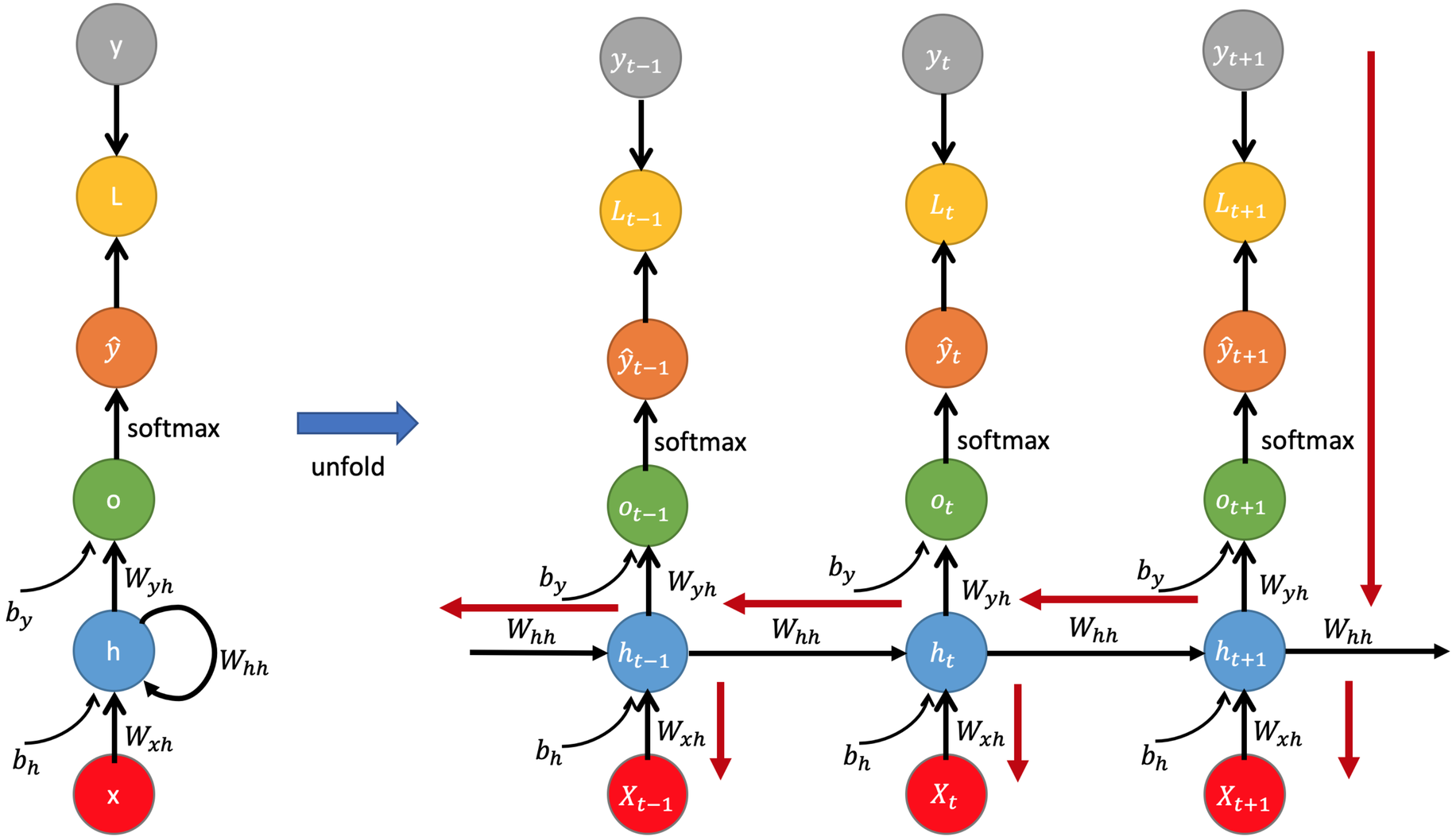


Sentence Generation

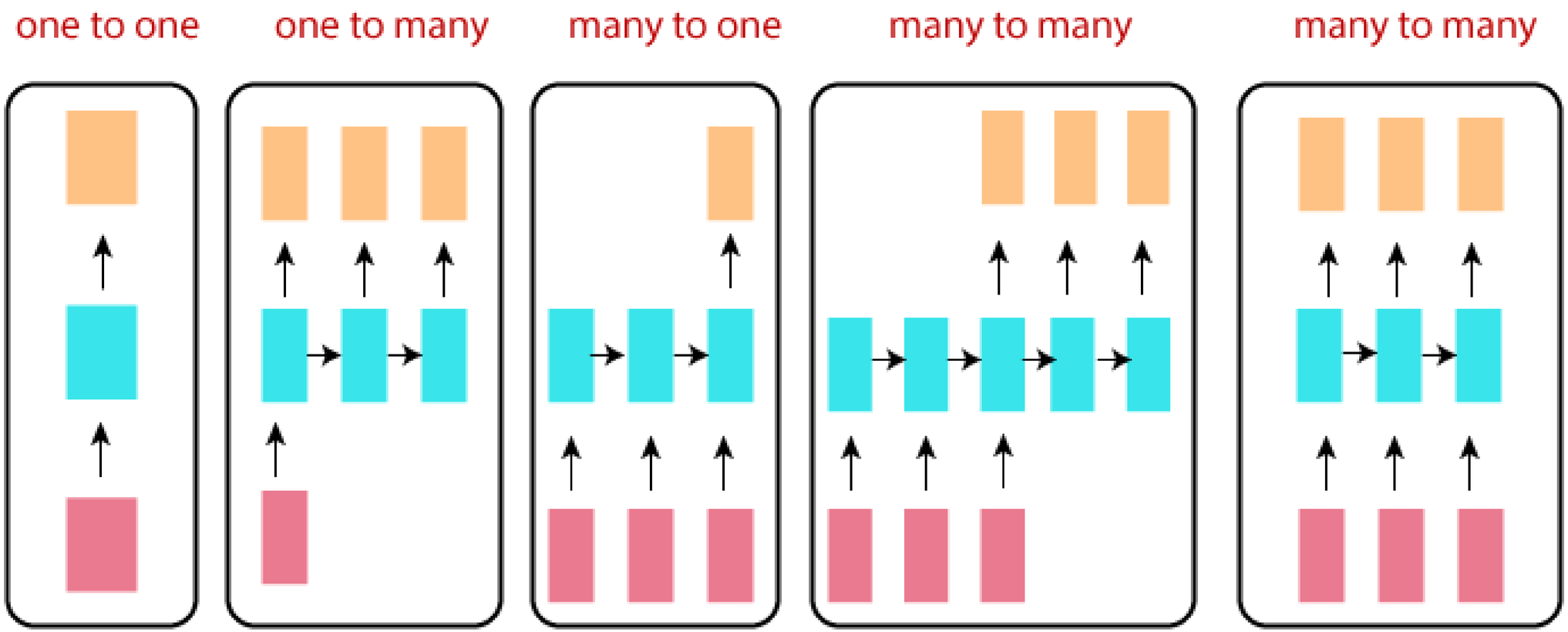


Language Translation

Recurrent Neural Network



Types of RNN



RNN Forward Propagation – Mathematically

- Mathematically, the form of transition function and output function in RNN is given as:

In hidden layer,

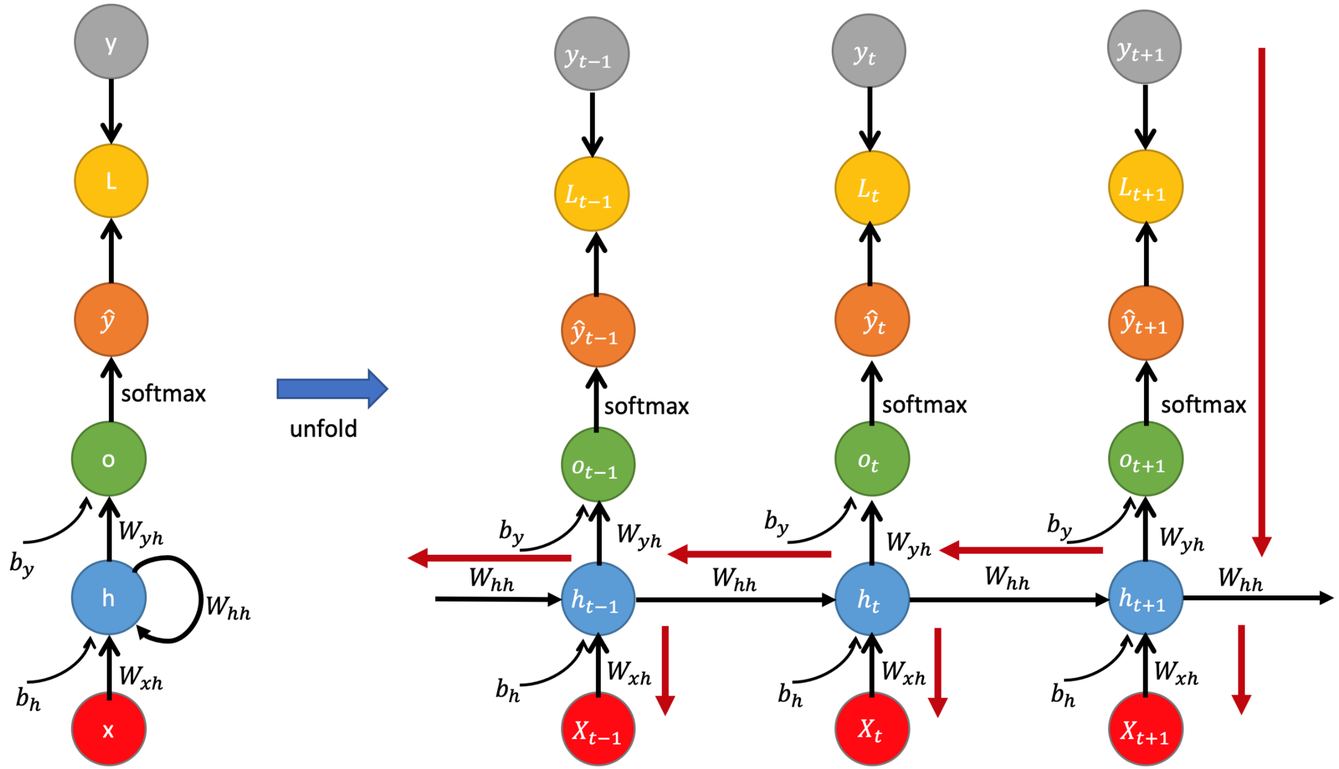
$$h_t = g_h(X_t, h_{t-1})$$

$$h_t = g_h(W_{xh}.X_T + W_{hh}.h_{t-1} + b_h)$$

For the Output

$$\hat{y}_t = g_o(h_t)$$

$$\hat{y}_t = g_h(W_{yh}.h_t + b_y)$$



*** I have used h instead of a for action value only to align notation with figures.*

RNN Forward Propagation – Mathematically

Where W_{xh}, W_{hh} and W_{yh} are weight matrices for the input, recurrent connections and the output, respectively. h_{t-1}, h_t, h_{t+1} are the hidden states in the hidden layer of each time step t.

And, g_h and g_o are element wise non linear function. Usually, when comes to RNN, we use the ***Tanh (Hyperbolic tangent)*** activation function in the hidden units of each time step. This is because Tanh activation maps inputs between -1 and +1 which helps to manage issues like the vanishing gradient problem

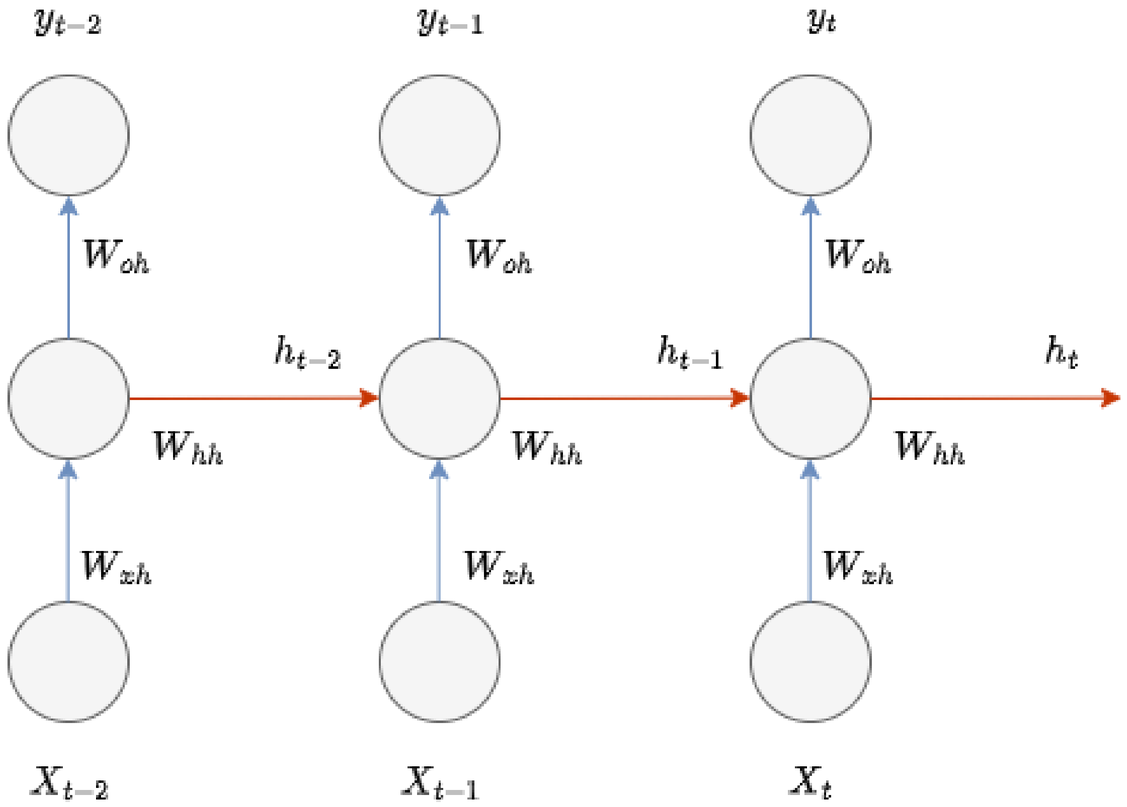
$$Tanh = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

Back Propagation through time (BPTT)

- Recurrent Neural Networks specialize in processing sequences of data, introducing the concept of time steps, where each step corresponds to a specific moment in the sequence. This temporal aspect enables RNNs to excel in tasks involving sequential data like text, speech, and time series.
- A time step in a Recurrent Neural Network (RNN) refers to a specific moment or instance in a sequence of data being processed by the network. In the context of sequential data, such as text, speech, or time series, the RNN processes one element of the sequence at each time step.
- For RNNs to learn sequential data, a variant of the backpropagation algorithm known as "Backpropagation Through Time" (BPTT) is used.

BPTT

- Consider we have only three time sequence in the network…



BPTT...

- Consider we are performing regression on share market data, thus our loss function would be Mean Squared Error (MSE).

$$E = \frac{1}{T}\sum_{t=1}^{T}(y - \hat{y}_t)^2$$

And to propagate loss backward to adjust weights and biases we need to find the derivative of Error.

$$\frac{\partial E}{\partial \hat{y}_t} = d\hat{y}_t$$

BPTT...

Next, we compute derivative of Error with respect to h_t,

$$\frac{\partial E}{\partial h_t} = \frac{\partial E}{\partial \hat{y}_t} \cdot \frac{\partial y_t}{\partial h_t} = dh_t$$

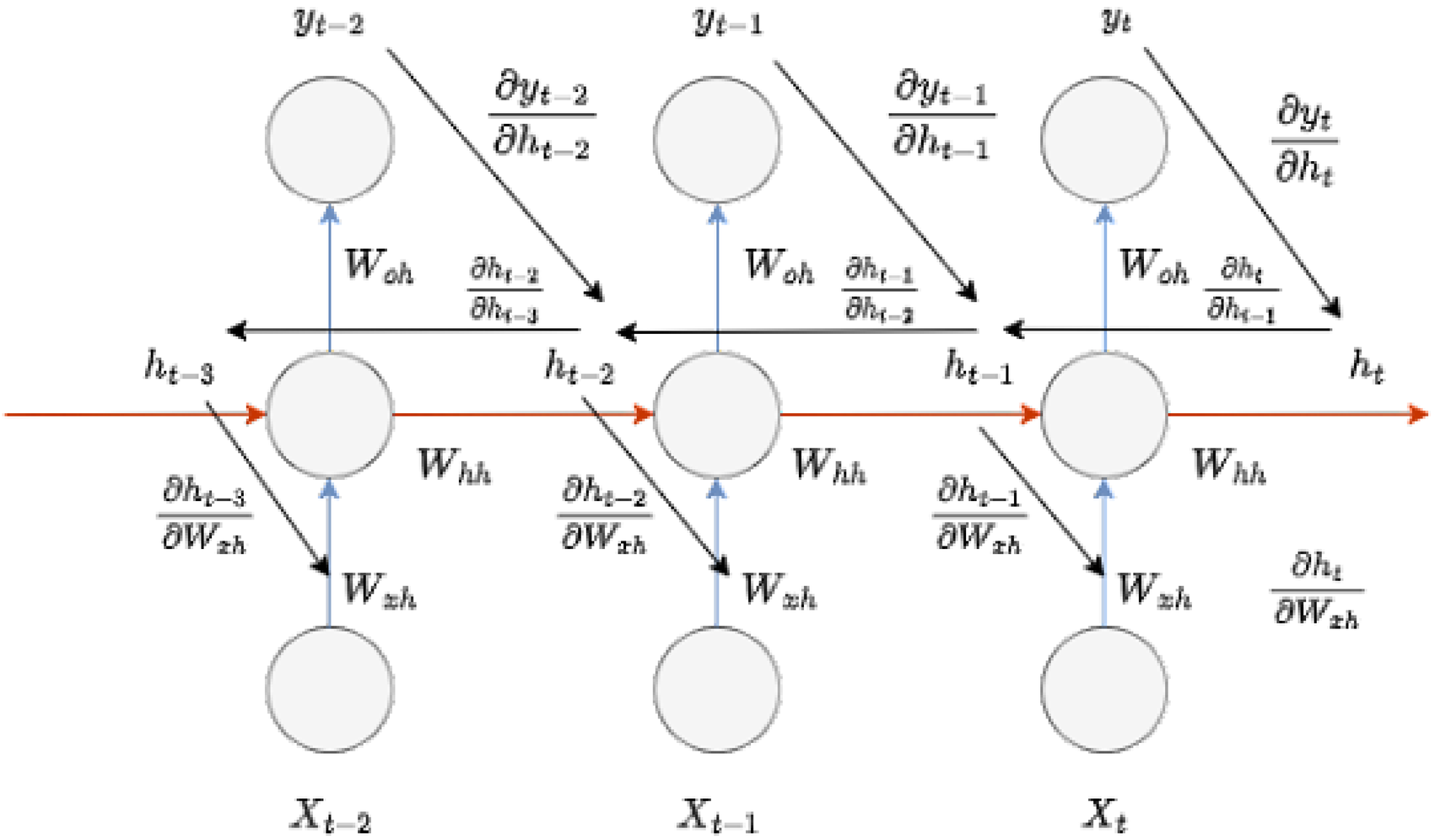
And, next derivate with respect to W_{yh},

$$\frac{\partial E}{\partial W_{yh}} = \frac{\partial E}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial W_{yh}}$$

And so on…

BPTT...

And the backpropagation continues.....



BPTT...

To make it simpler, lets compute derivative of loss w.r.t. W_{xh},

i) considering a single time step,

$$\frac{\partial E}{\partial W_{xh}} = \frac{\partial E}{\partial y_t} . \frac{\partial y_t}{\partial h_t} . \frac{\partial h_t}{\partial h_{t-1}} . \frac{\partial h_{t-1}}{\partial W_{xh}}$$

ii) Considering 2 time step,

$$\frac{\partial E}{\partial W_{xh}} = \frac{\partial E}{\partial y_t} . \frac{\partial y_t}{\partial h_t} . \frac{\partial h_t}{\partial h_{t-1}} . \frac{\partial h_{t-1}}{\partial W_{xh}} + \frac{\partial E}{\partial y_{t-1}} . \frac{\partial y_t}{\partial h_t} . \frac{\partial h_t}{\partial h_{t-1}} . \frac{\partial h_{t-1}}{\partial h_{t-2}} . \frac{\partial h_{t-2}}{\partial W_{xh}}$$

BPTT...

iii) Similarly, considering three time steps

$$\frac{\partial E}{\partial W_{xh}} = \frac{\partial E}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_{xh}} + \frac{\partial E}{\partial y_{t-1}}\frac{\partial y_{t-1}}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial W_{xh}} + \frac{\partial E}{\partial y_{t-2}}\frac{\partial y_{t-2}}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial h_{t-3}}\frac{\partial h_{t-3}}{\partial W_{xh}}$$

This shows us that with increase in time step, the gradient also accumulate in earlier steps. Thus, generalizing it

$$\frac{\partial E}{\partial W_{xh}} = \sum_{t=0}^{T} \frac{\partial E}{\partial y_{T-t}}\frac{\partial y_{T-t}}{\partial h_T}\left(\prod_{j=T-t+1}^{T}\frac{\partial h_{T-j+1}}{\partial h_{T-j}}\right)\frac{\partial h_{T-t-1}}{\partial W_{xh}}$$

BPTT...

Continuing the same,
We obtain the derivate of Error w.r.t. Weights in hidden state i.e. W_{hh}

$$\frac{\partial E}{\partial W_{hh}} = \sum_{t=0}^{T} \frac{\partial E}{\partial y_{T-t}} \frac{\partial y_{T-t}}{\partial h_T} \left(\prod_{j=T-t+1}^{T} \frac{\partial h_{T-j+1}}{\partial h_{T-j}} \right) \frac{\partial h_{T-t-1}}{\partial W_{hh}}$$

And, we obtain the derivative of Error w.r.t. Weights in Output layer i.e. W_{oh}

$$\frac{\partial E}{\partial W_{oh}} = \sum_{t=0}^{T} \frac{\partial E}{\partial y_{T-t}} \frac{\partial y_{T-t}}{\partial h_T} \left(\prod_{j=T-t+1}^{T} \frac{\partial h_{T-j+1}}{\partial h_{T-j}} \right) \frac{\partial h_{T-t-1}}{\partial W_{oh}}$$

BPTT...

Similarly, the derivative of Error w.r.t. bias in hidden states b_h

$$\frac{\partial L}{\partial b_h} = \sum_{t=0}^{T} \frac{\partial E}{\partial y_{T-t}} \frac{\partial y_{T-t}}{\partial h_{T-t}} \frac{\partial h_{T-t}}{\partial b_h}$$

And, the derivative of Error w.r.t. bias in output layer b_o

$$\frac{\partial L}{\partial b_o} = \sum_{t=0}^{T} \frac{\partial L}{\partial y_{T-t}} \frac{\partial y_{T-t}}{\partial b_o}$$

Updating Weights and bias

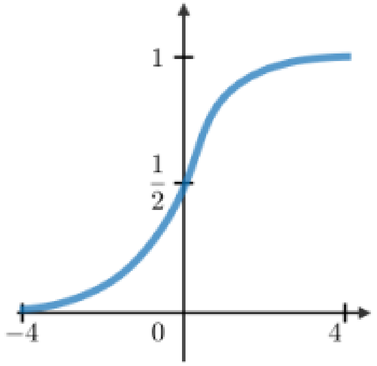
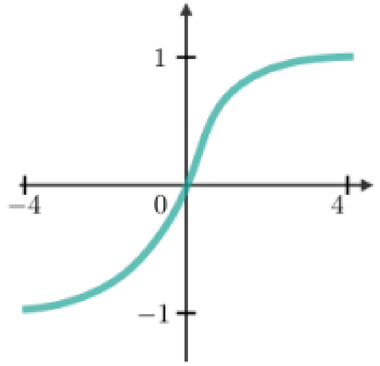
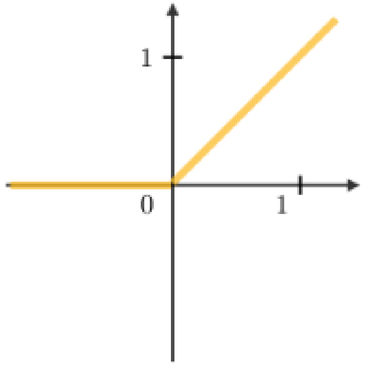
Now similar to other neural networks, we can apply gradient descent to update weights and bias as:

$$(\boldsymbol{W}, \boldsymbol{b}) = (\boldsymbol{W}, \boldsymbol{b}) \; - \alpha \frac{\partial L}{\partial (\boldsymbol{W}, \boldsymbol{b})}$$

Vanishing/Exploding Gradient

- The vanishing and exploding gradient phenomena are often encountered in the context of RNNs.
- The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

Commonly used activation functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

References

- CS 230 - Recurrent Neural Networks Cheatsheet (stanford.edu)