Chapter 5

Dynamic Networks and Deep Learning

5.1. Dynamic systems and dynamic NNets

5.2. Autoencoders

5.3. Convolutional Neural Networks (CNN)

5.4. Long Short-Term Memory NN (LSTM)

5.5. Generative NNs & Transformers

5.6. Conclusions

5.1 Dynamic systems and memory in dynamic NN

(Hagan, Chapter 14, nn_ug 2023b Matlab)

Many systems of practical importance may be described by linear difference equations, at instant *t* or *k* (equivalent notations)

$$y(k) = b_0 u(k) + b_1 u(k-1) + \ldots + b_m u(k-m) - a_1 y(k-1) - a_2 y(k-2) - \ldots - a_n y(k-n)$$

$$y(t) = b_0 u(t) + b_1 u(t-1) + \ldots + b_m u(t-m) - a_1 y(t-1) - a_2 y(t-2) - \ldots - a_n y(t-n)$$

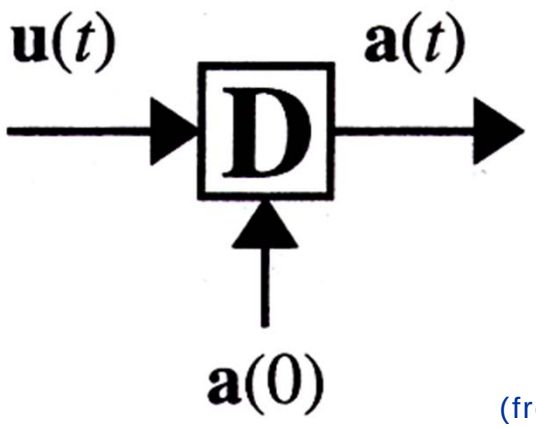
It can be said that the system has memory of size *n* in the output and memory of size *m* in the input.

The coefficients of the difference equations are the *a*'s and the *b*'s.

This equation can be implemented by a linear neuron without bias (as the ADALINE without bias).

Delays and Integrators

Delay

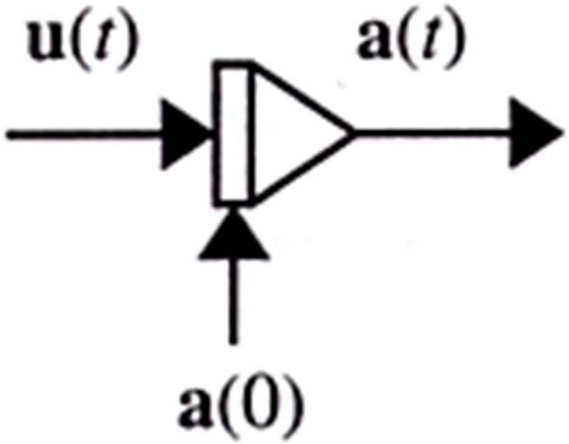


(from DL Toolbox Manual)

$$a(t) = u(t-1)$$

$$a(0) \triangleq \text{initial condition}$$

Integrator

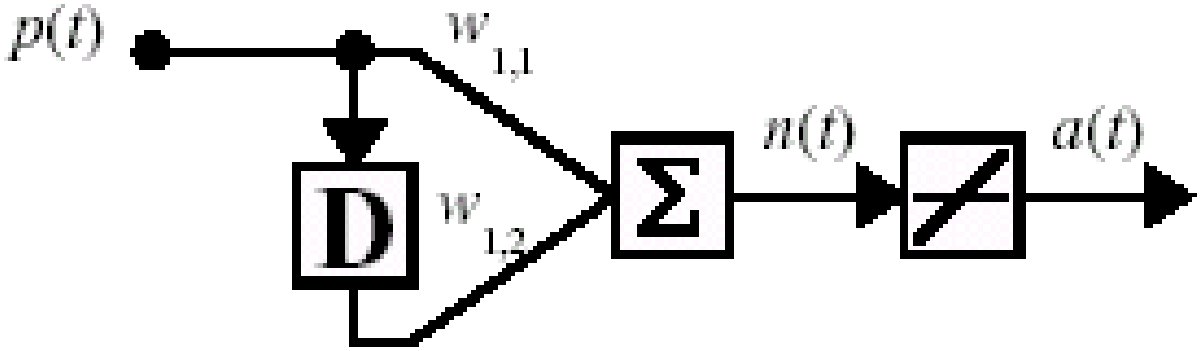


$$a(t) = \int_0^t u(\tau)\,d\tau + a(0)$$

$$a(0) \triangleq \text{initial condition}$$

Dynamic network with delays

Inputs Linear neuron

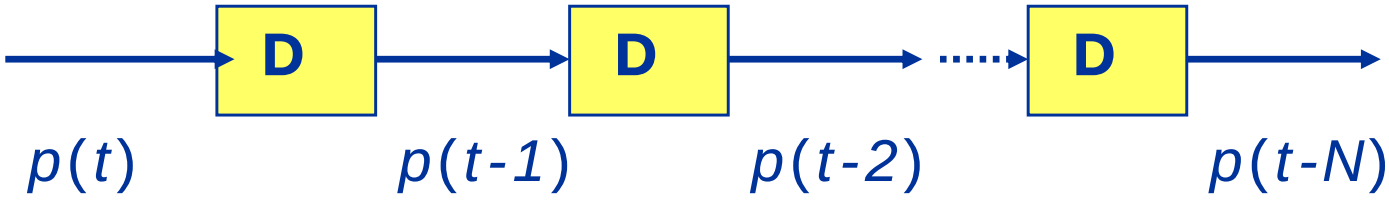


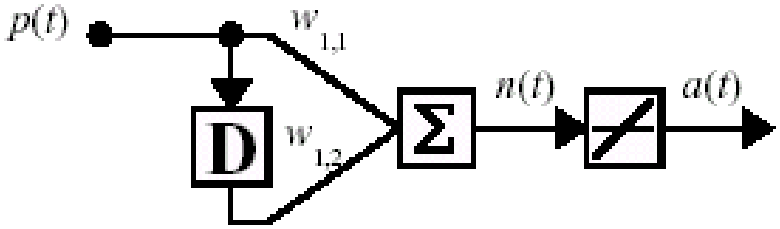
(from DL Toolbox ug, Chap 24.)

$$a(t) = w_{11}\,p(t) + w_{12}\,p(t-1)$$

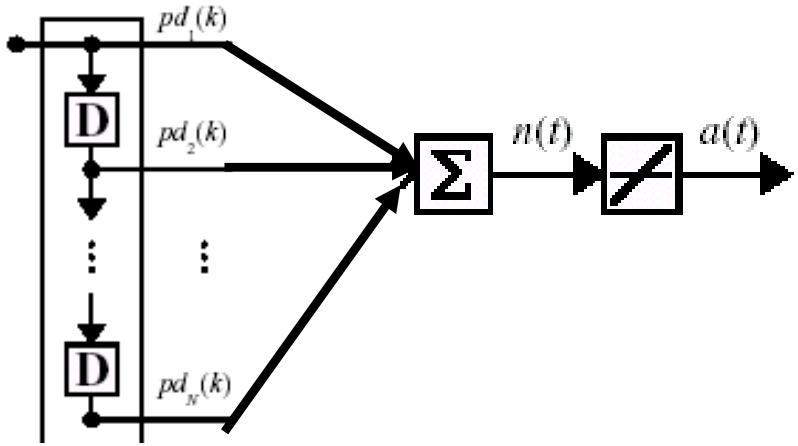
The output at instant t depends on what happened at instant t-1 (the neuron has memory)

Block of pure time delays (Delay)



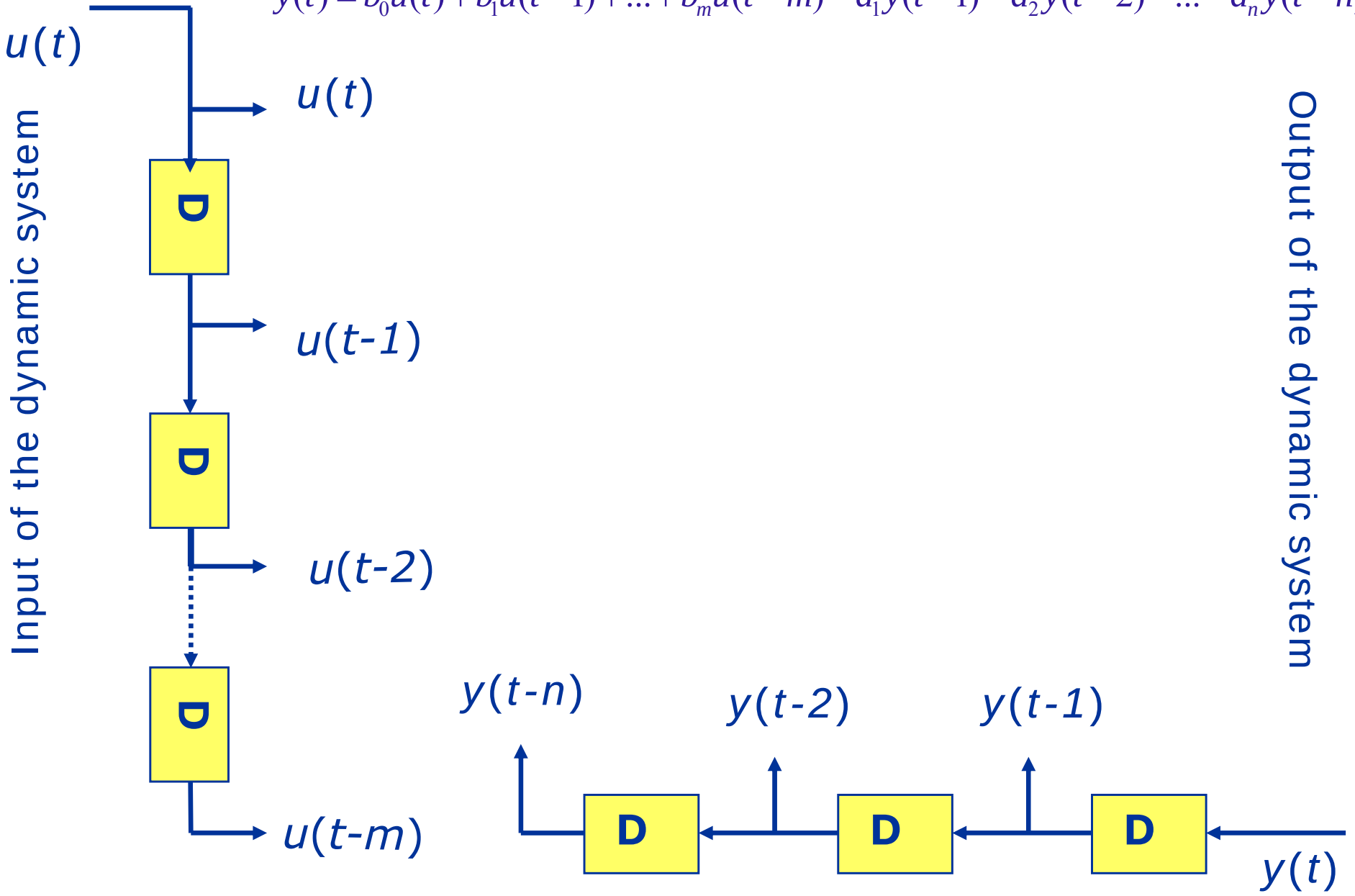


from DL Toolbox U.G.

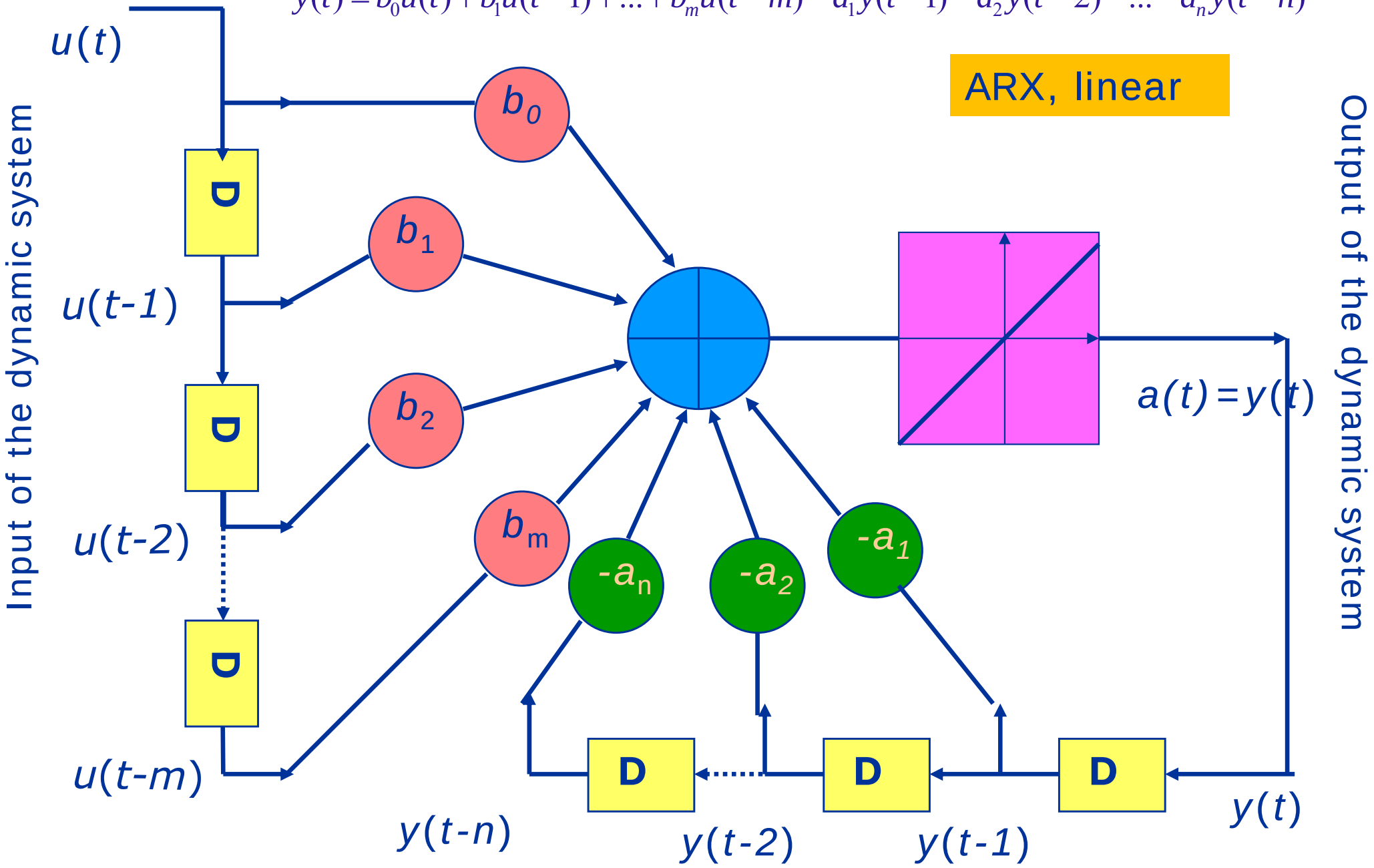


$$n(t) = w_{11}\,p(t) + w_{12}\,p(t-1) \qquad n(t) = w_{11}\,p(t) + w_{12}\,p(t-1) + ... + w_{1N}\,p(t-N)$$

$$y(t) = b_0 u(t) + b_1 u(t-1) + \ldots + b_m u(t-m) - a_1 y(t-1) - a_2 y(t-2) - \ldots - a_n y(t-n)$$

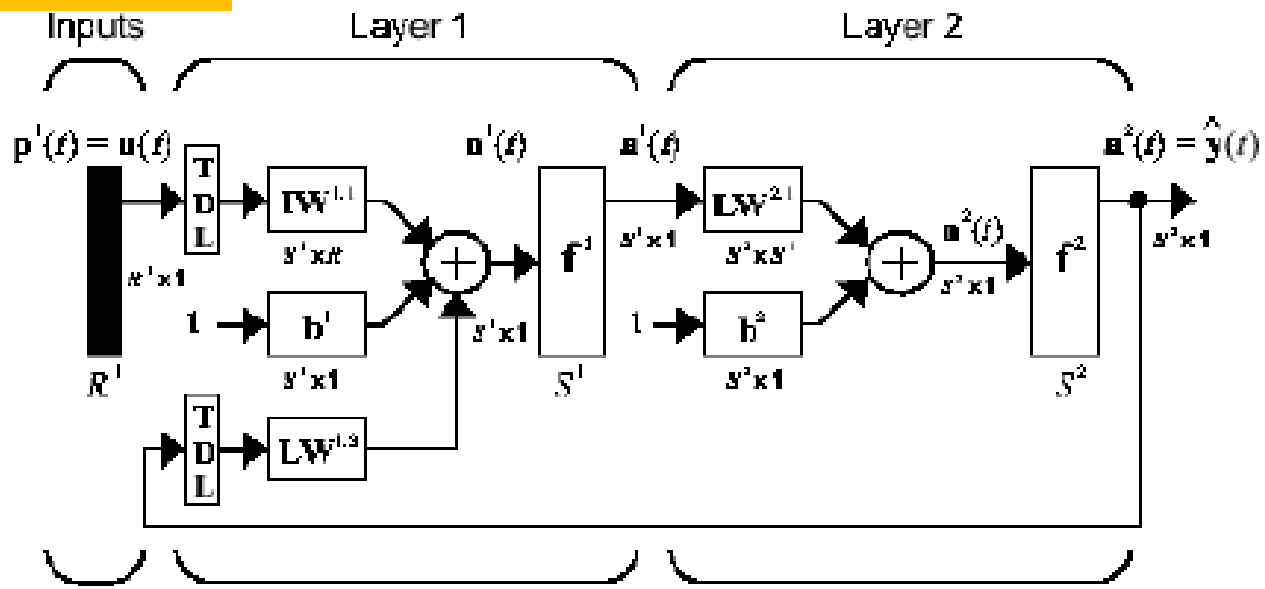


$$y(t) = b_0 u(t) + b_1 u(t-1) + \ldots + b_m u(t-m) - a_1 y(t-1) - a_2 y(t-2) - \ldots - a_n y(t-n)$$



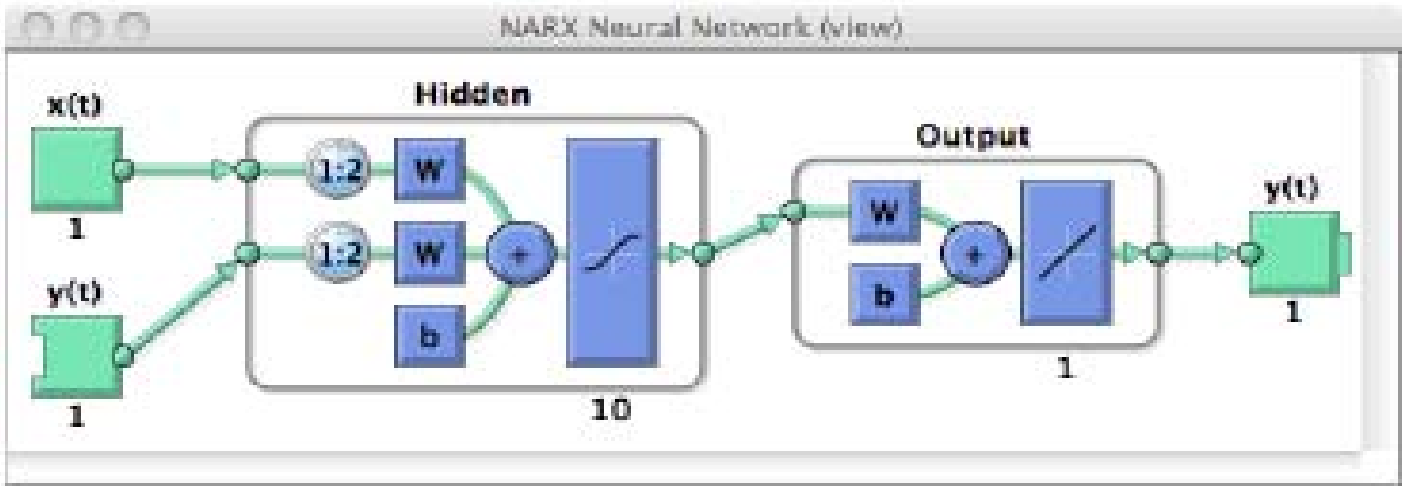
NARX, non linear

$$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n_y), u(t-1), u(t-2), \ldots, u(t-n_u))$$



(p. 24-18 nnet_ug 2023b)

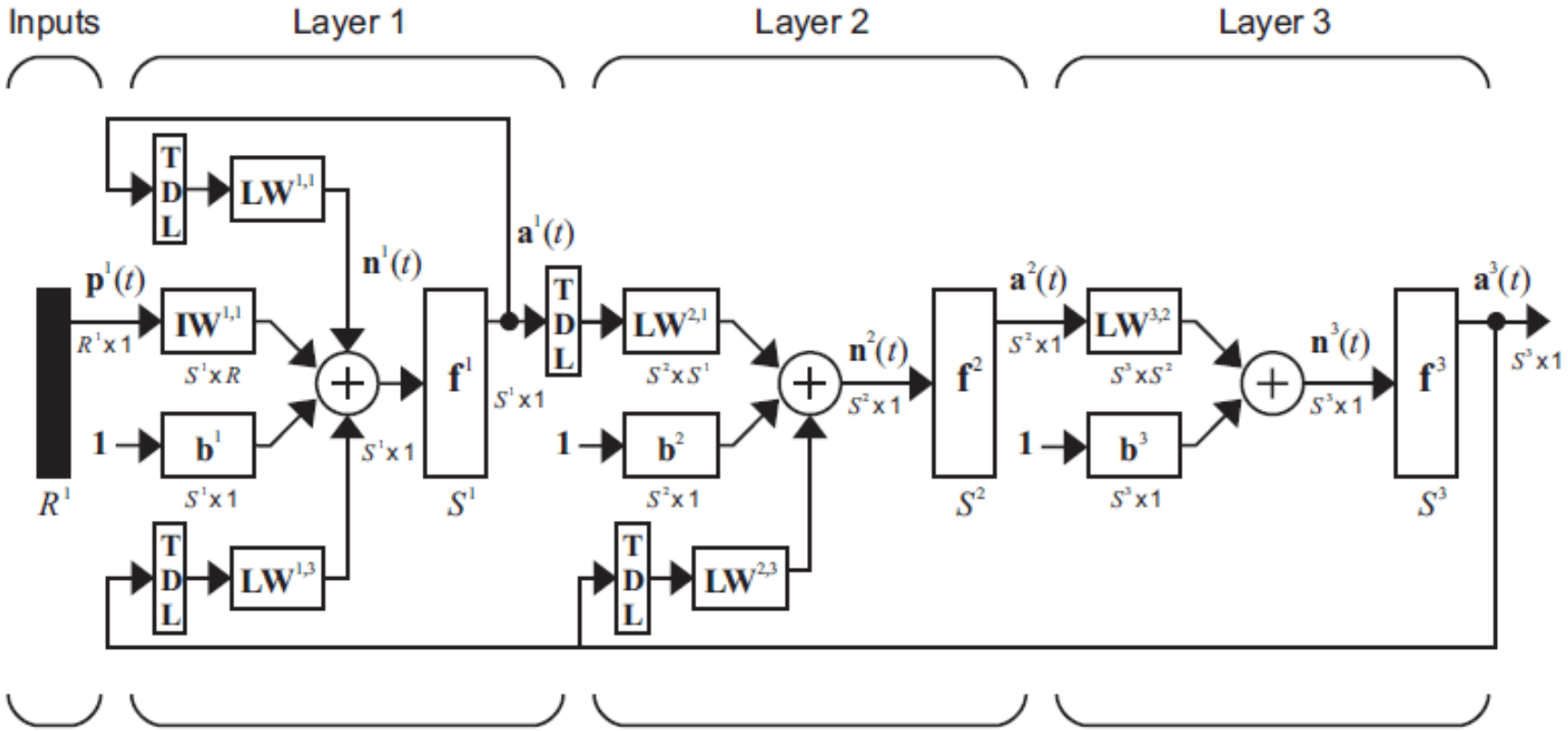
narx_net = narxnet(d1,d2,10)



Other dynamic architectures

Layered Digital Dynamic Network (LDDN)

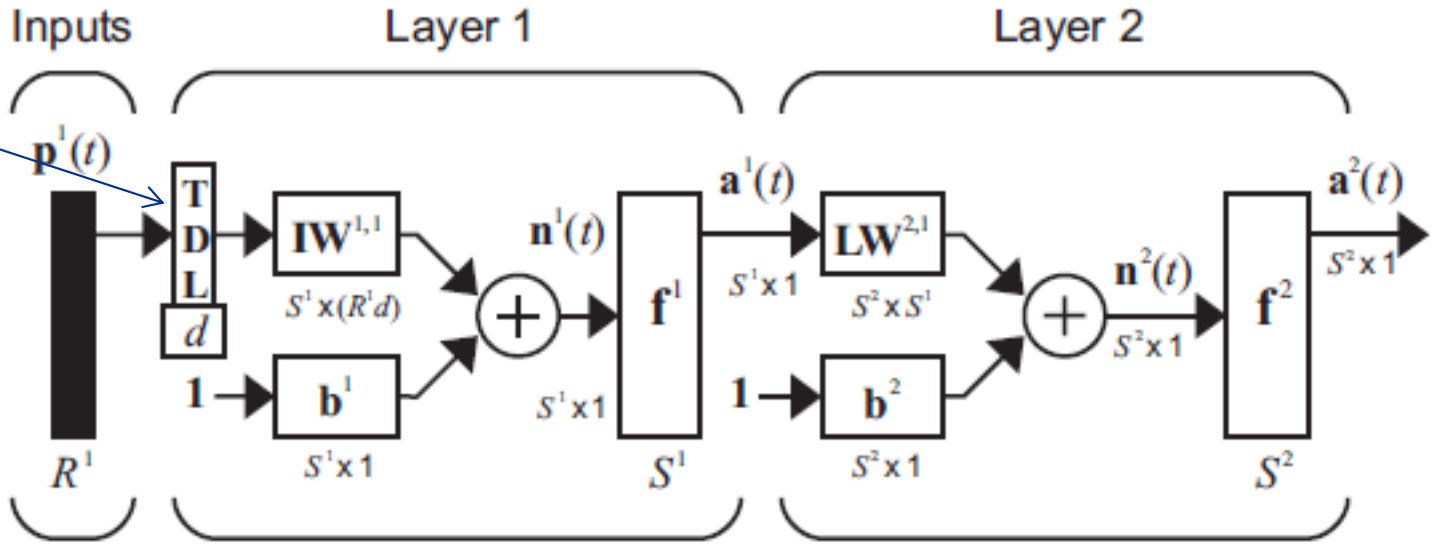
Nnet_ug2023a, page 24-10



Focused Time-Delay Neural Network

timedelaynet

Taped-delay line



Nnet_ug2023a, Chapt. 24-12

ftdnn_net = timedelaynet([1:8],10)

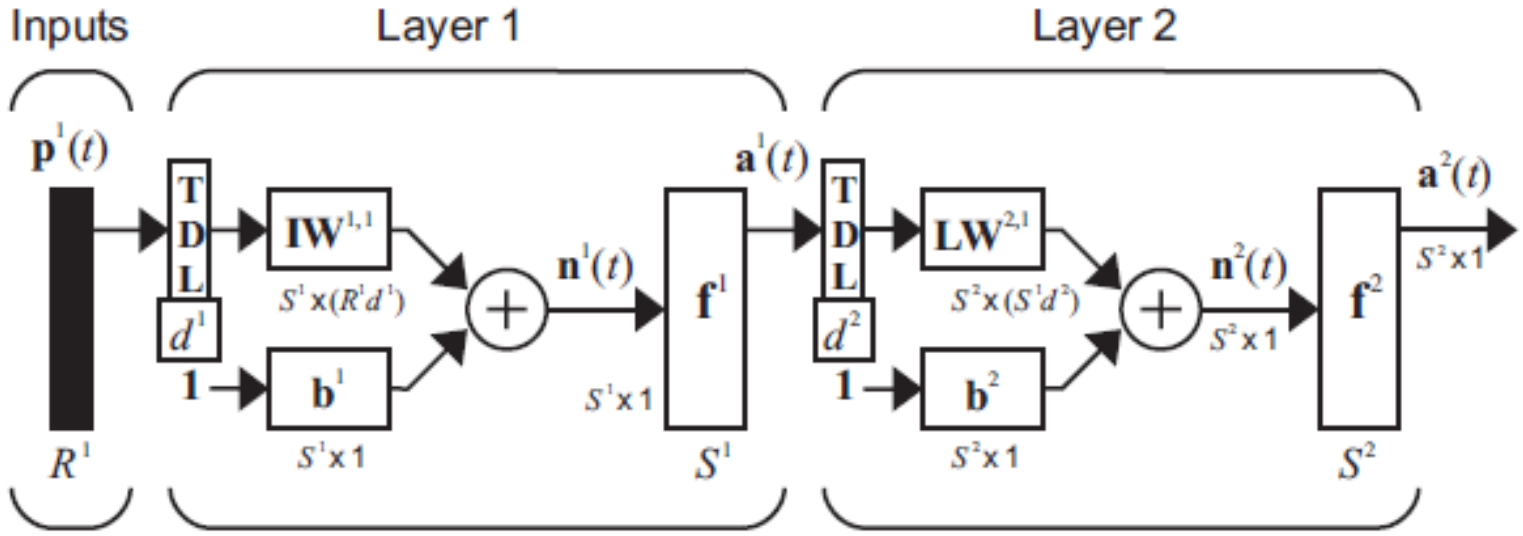
Good for forecasting of temporal series

(Example in p. 24-12 nn_ug 2020a)

Does not require dynamic retropropagation

Distributed Time-Delay Neural Network

distdelaynet



nn-ug 2023 Cap. 24-16

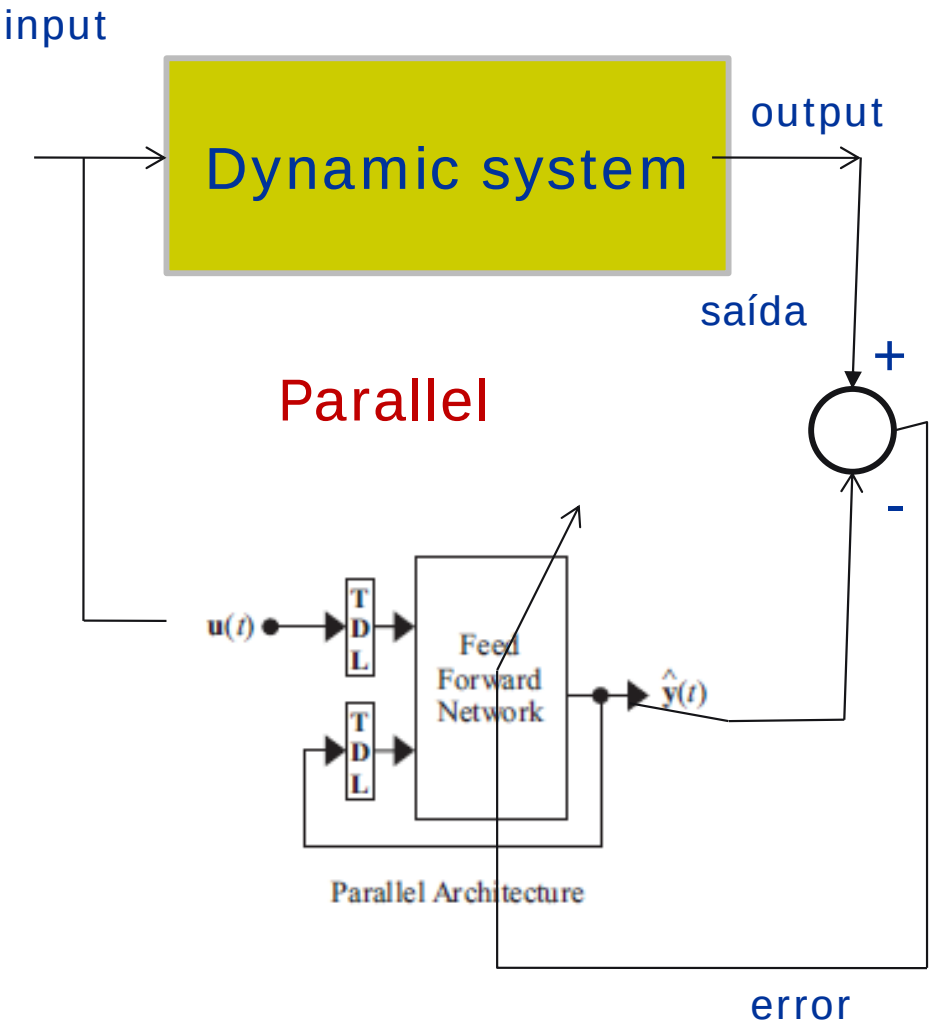
Exemple: p. 24-16 dlug 2022a

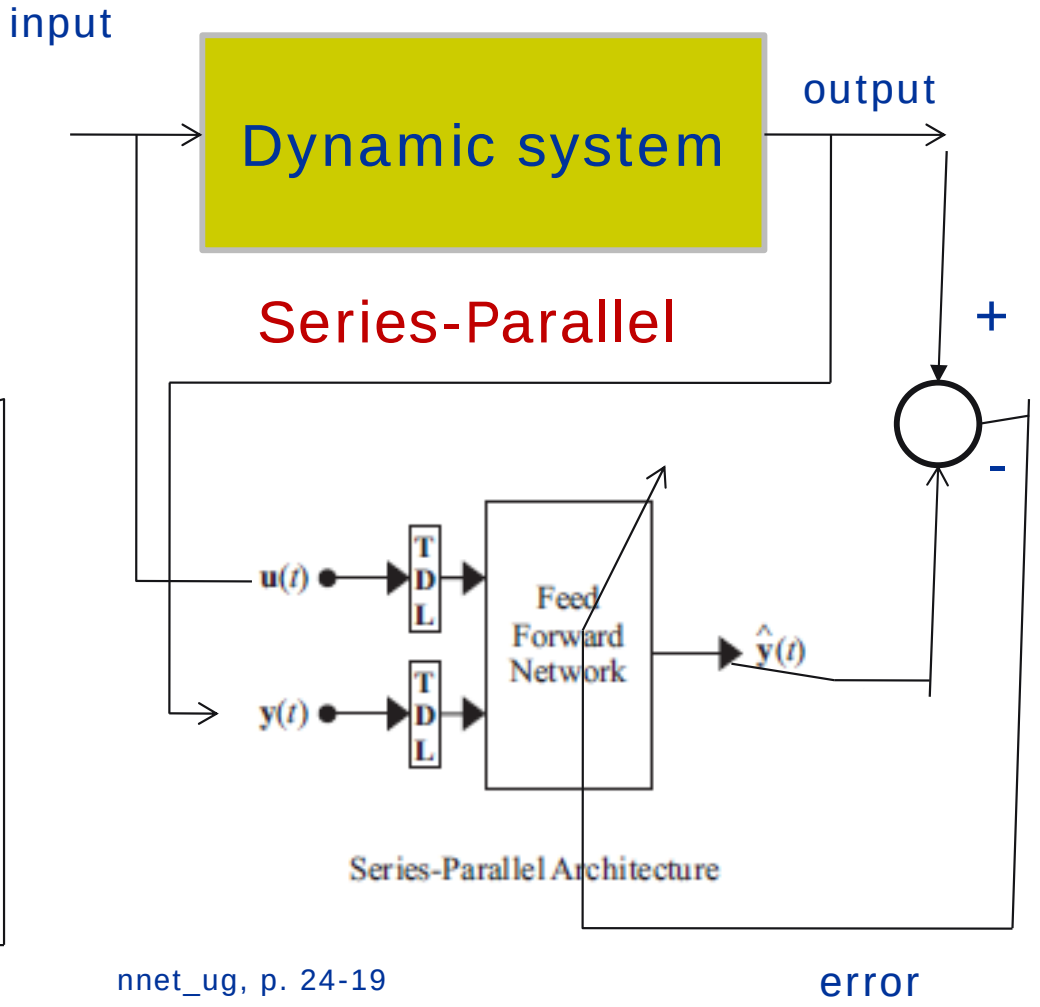
dtdnn_net = distdelaynet({d1,d2},5);

Alternative Assemblies

$$\hat{y}_k = f(\hat{y}_{k-1}, \hat{y}_{k-2}, \ldots, \hat{y}_{k-n}, u_{k-1}, u_{k-2}, \ldots, u_{k-m})$$

$$\hat{y}_k = f(y_{k-1}, y_{k-2}, \ldots, y_{k-n}, u_{k-1}, u_{k-2}, \ldots, u_{k-m})$$



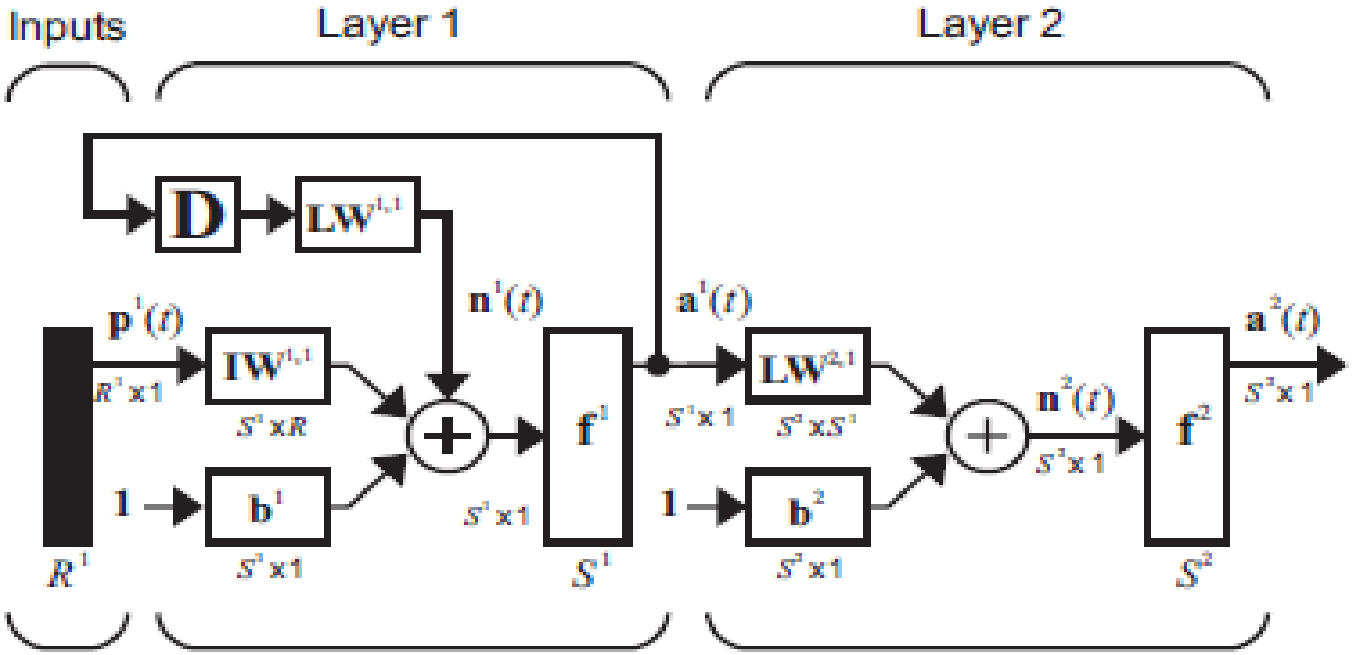


nnet_ug, p. 24-19

Layer-Recurrent Network

p. 24-26 nnet_ug 2023b

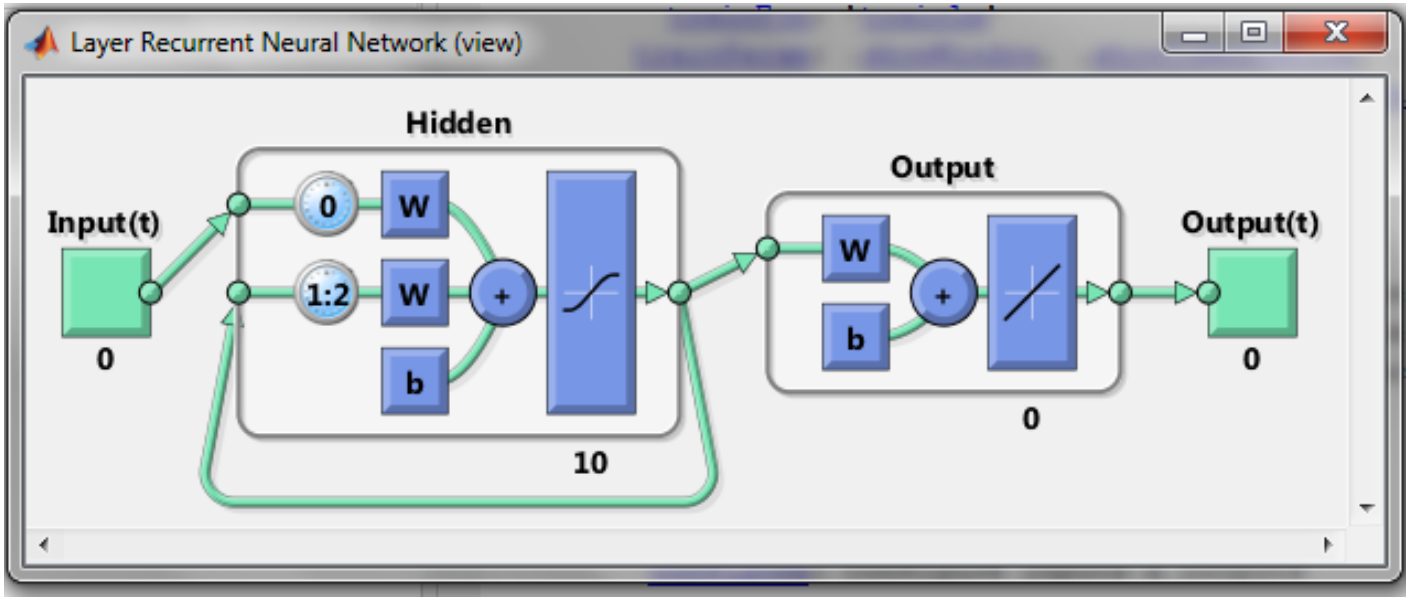
layrecnet



Feedback with one or more delays in all layers (can have any number), except the last one.

net=layrecnet(layerDelays,hiddenSizes,trainFcn)

```
net = layrecnet(1:2,10);
```

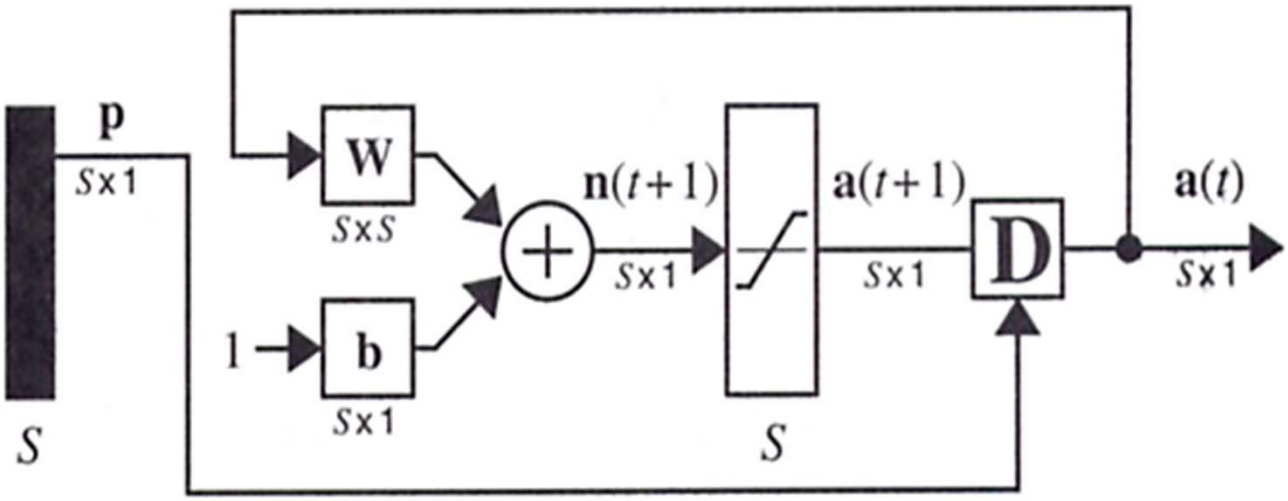


Training: gradient based methods
Ex. p. 24-26 nnet_ug 2022a

Recurrent Network

Recurrent layer

Initial condition



(from DL Toolbox Manual)

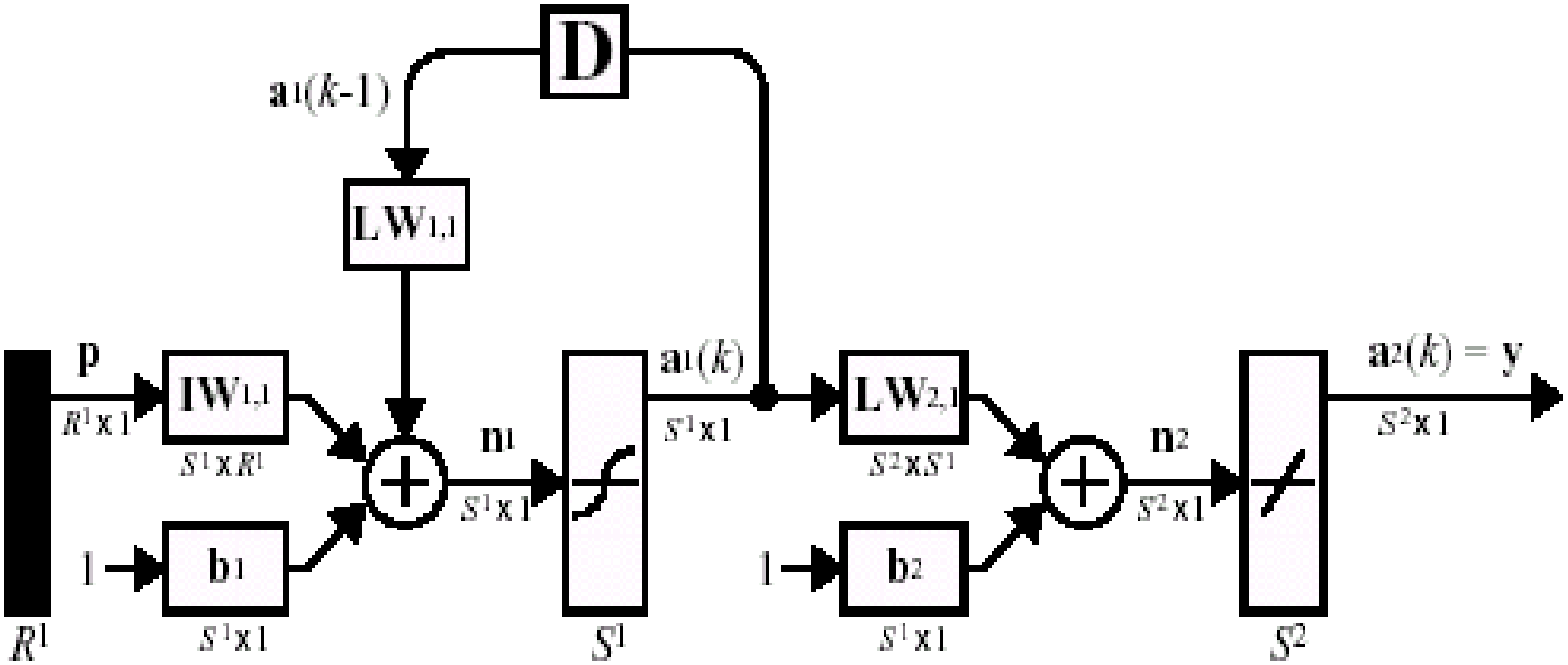
$$a(t+1) = satlins(W \times a(t) + b)$$

$$a(0) = p$$

Elman recurrent network

(historical, a particular case of Layer-Recurrent NN with 2 layers)

tansig recurrent layer output linear layer



$$a^1(k) = \text{tansig}(IW^{1,1}p + LW^{1,1}a^1(k-1) + b^1)$$

$$a^2(k) = \text{purelin}(LW^{2,1}a^1(k) + b^2)$$

Training the NN with memory:

- *Dynamic backpropagation*:

  - The computation of the gradient is more complex, because of the *feedback.*
  - Higher computational complexity.
  - Harder convergence: the trap of local minima.
  - In some architectures it is used RTRL (Real Time Recurrent Learning), BPTT (Back Propagation Through Time, Hagan 14-11).

- Static backpropagation, as in the NN without memory
  - Requires pre-organization of data to build the temporal sequences (in Matlab: *preparets).*
  - it works in some architectures.

Shallow networks: small number of layers

Deep networks: high number of layers

How small is small ?

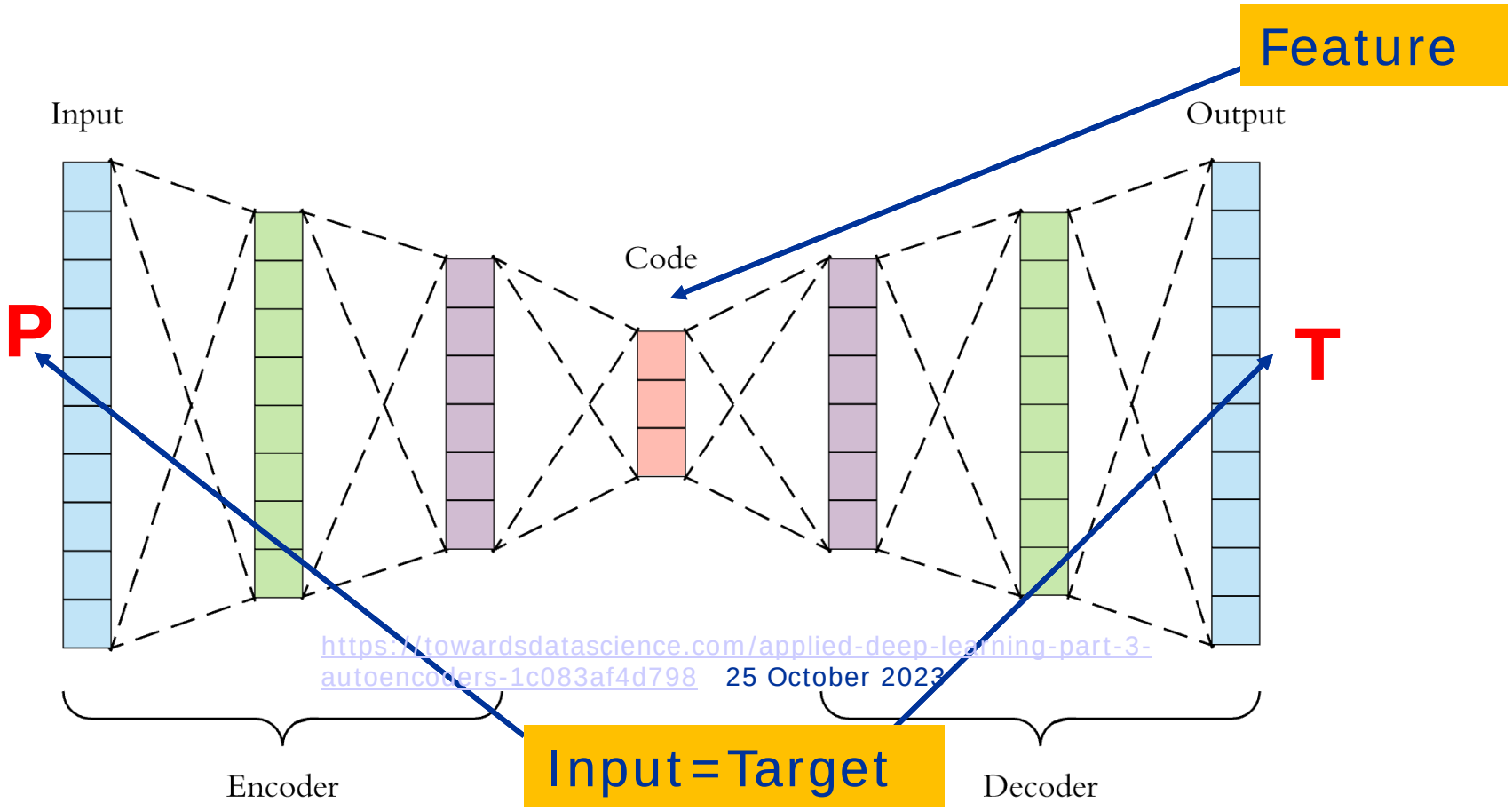
How high is high ?

... no threshold defined ....

5.2. Autoencoders

Several layers feedforward neural network.

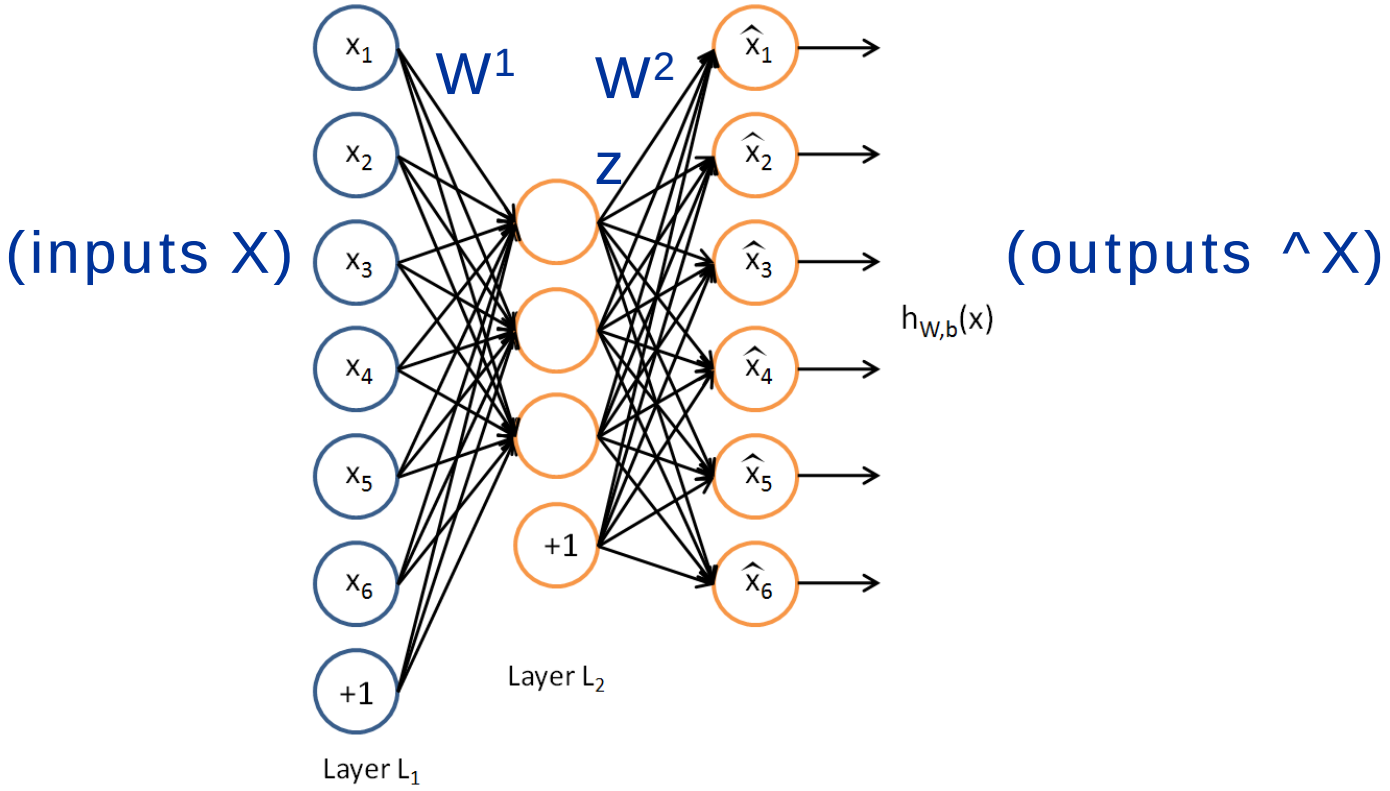
The output layer must reproduce the inputs: **T = P**



https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 25 October 2023

https://www.mathworks.com/help/nnet/autoencoders.html?s_tid=gn_loc_drop

Illustration with two layers (the Matlab implementation)



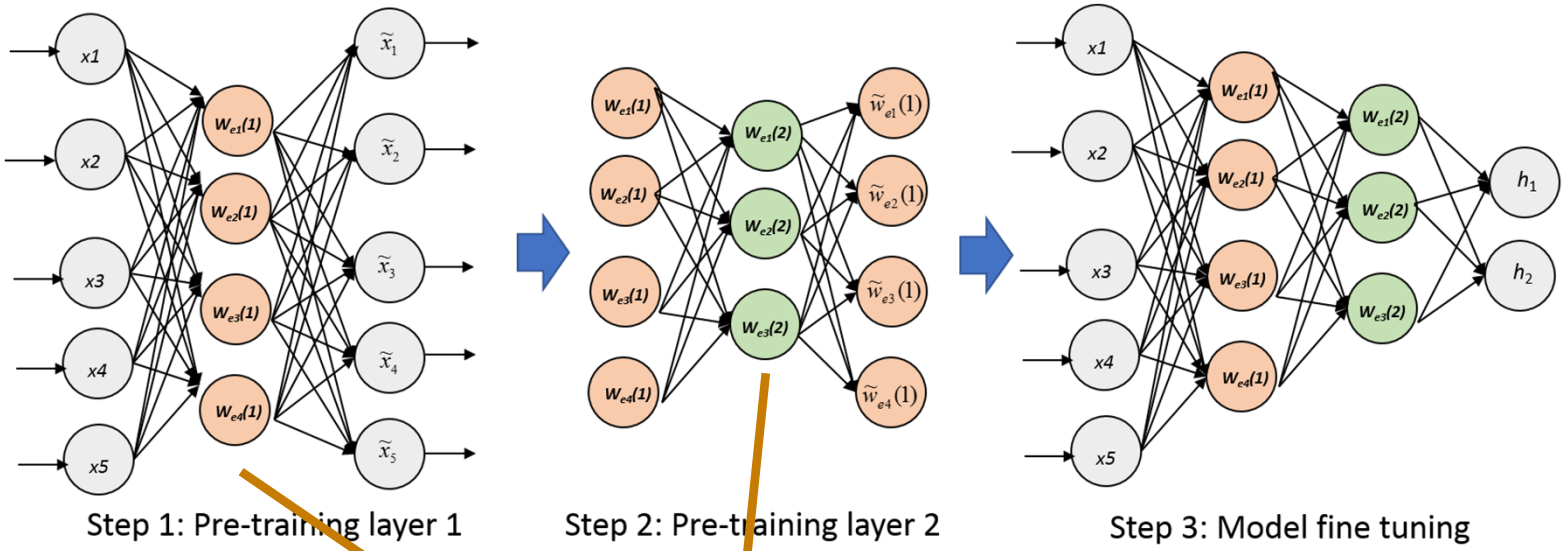
$z = \sigma^1(W^1 X + b^1)$ $\quad \hat{X} = \sigma^2(W^2 z + b^2)$

idealy $X = \hat{X}$

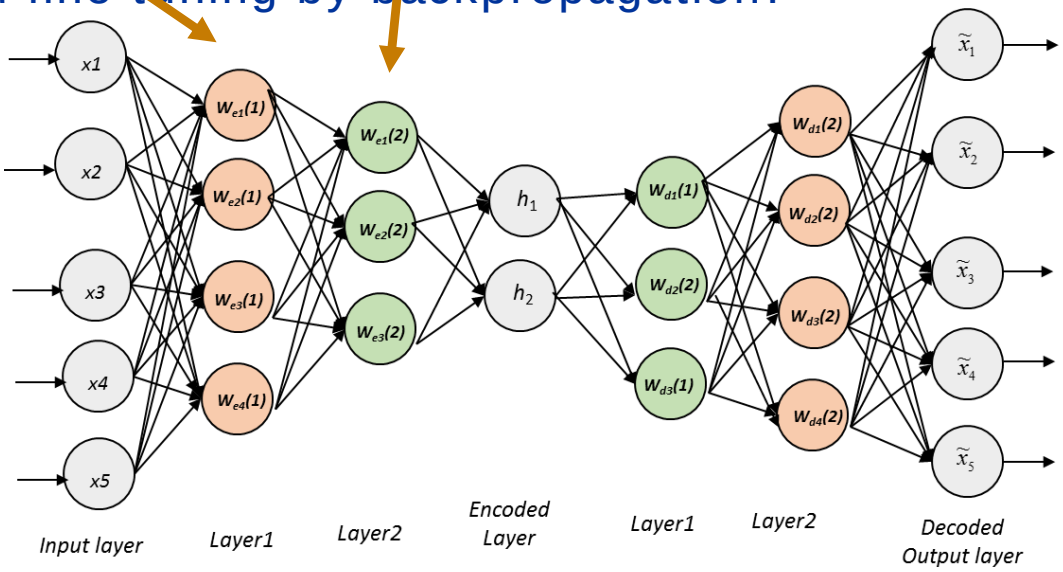
adapted from http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/ 25 October 2023

Stacked autoencoders

https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781787121089/4/ch04lvl1sec51/setting-up-stacked-autoencoders (25 October 2023)



Assemble and fine tuning by backpropagation:



Usually, the middle layers have less neurons than the input and output layers.

The output ^X must be equal to X, so with the middle variable z the input is “rebuilt”. This means that z contains sufficient information to reproduce the input , i.e., z is a highly representative **feature** of P. In the limit z may have dimension 1.

For example, if we have inputs with 20 dimensions, then these 20 dimensions may be reduced to one, the z, eventually without significative loss of information (ideal situation ...).

So, with autoencoders we can extract features from the data, reducing the dimension, giving the features to a classifier of reduced dimension, with better computing properties.

Regularization, prevents overfitting, improves generalization

The typical performance function used for training feedforward neural networks is the mean sum of squares of the network errors.

$$F = mse = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(t_i - \alpha_i)^2$$

Nnet_ug2023b, p. 29-29

It is possible to improve generalization if you modify the performance function by adding a term that consists of the mean of the sum of squares of the network weights and biases

$msereg = \gamma * msw + (1-\gamma) * mse$, where γ is the performance ratio, and

$$msw = \frac{1}{n}\sum_{j=1}^{n} w_j^2$$

γ is a regularization parameter fixed by the user, between 0 and 1

Sparsity, improves training

sparsity proportion: average of the activations of one neuron in the hidden layer for all the inputs of the training set; a small value (typically 0.05) means that this neuron will give near zero output for most of the inputs.
This constrain can be introduced in the cost function *J* (the same as *F* in previous slide) by

$$J_{\text{sparse}}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} \text{KL}(\rho||\hat{\rho}_j),$$

where ρ is the desired average firing, $\hat{\rho}$ is the obtained value, and β is the sparsity regularization parameter. KL means the Kullback-Leibler divergence and is given by the differentiable function

$$\text{KL}(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

for more https://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/ 17/10/2023

Training Autoencoders

pp 1-18 nnet_gs2018b

in Matlab 2023b type >help Autoencoder

Load dataset

```
[X,T] = wine_dataset;
```

(X=13 features, T=Target 3 classes)

Train autoencoder

reduce number of features to 10

```
hiddenSize = 10;
autoenc1 = trainAutoencoder(X,hiddenSize,...
    'L2WeightRegularization',0.001,...
    'SparsityRegularization',4,...
    'SparsityProportion',0.05,...
    'DecoderTransferFunction','purelin');
```

Extract the 10 features

```
features1 = encode(autoenc1,X);
```

Train a second autoencoder with features1 as input (there may be several autoencoder layers)

Reduce the number of features to 6

```
hiddenSize = 6 ;
autoenc2 = trainAutoencoder(features1,hiddenSize,...
    'L2WeightRegularization',0.001,...
    'SparsityRegularization',4,...
    'SparsityProportion',0.05,...
    'DecoderTransferFunction','purelin',...
    'ScaleData',false);
```

Extract the new 6 features

```
features2 = encode(autoenc2,features1);
```

Train a softmax layer with features2

```
softnet = trainSoftmaxLayer(features2,T,'LossFunction','crossentropy');
```

stack (put together) all layers

```
deepnet = stack(autoenc1,autoenc2,softnet);
```

Train the network on wine data

```
deepnet = train(deepnet,X,T);
```

See the classification results

```
wine_type = deepnet(X);
```

Analyze results with confusion matrix

```
plotconfusion(T,wine_type);
```

more in https://www.mathworks.com/help/nnet/autoencoders.html?s_tid=gn_loc_drop

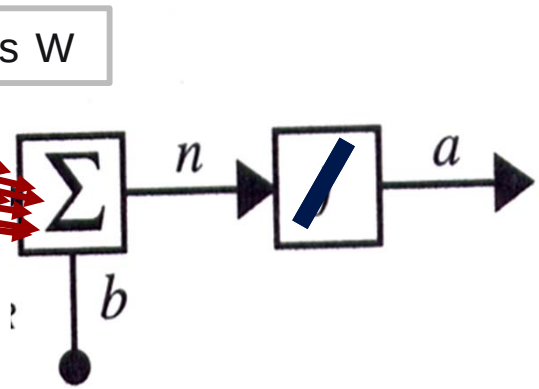
5.3- Convolutional Neural Networks

- multilayer feedforward
- many hidden layers
- different types of layers
- layers may not be fully connected; this fact reduces the number of weights to be learned
- very powerful for image analysis (are inspired by our visual cortex)

CNN- Convolution Layer

P, input data

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



W

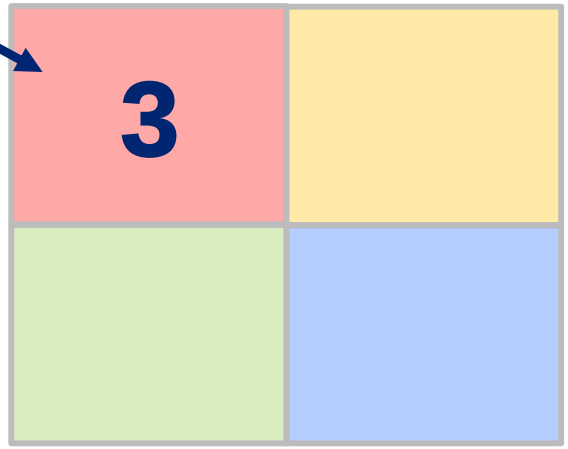
1	1	0
1	0	1
1	0	0

b **= 1**

WP+*b*=*a* (Hadamard multiplication)

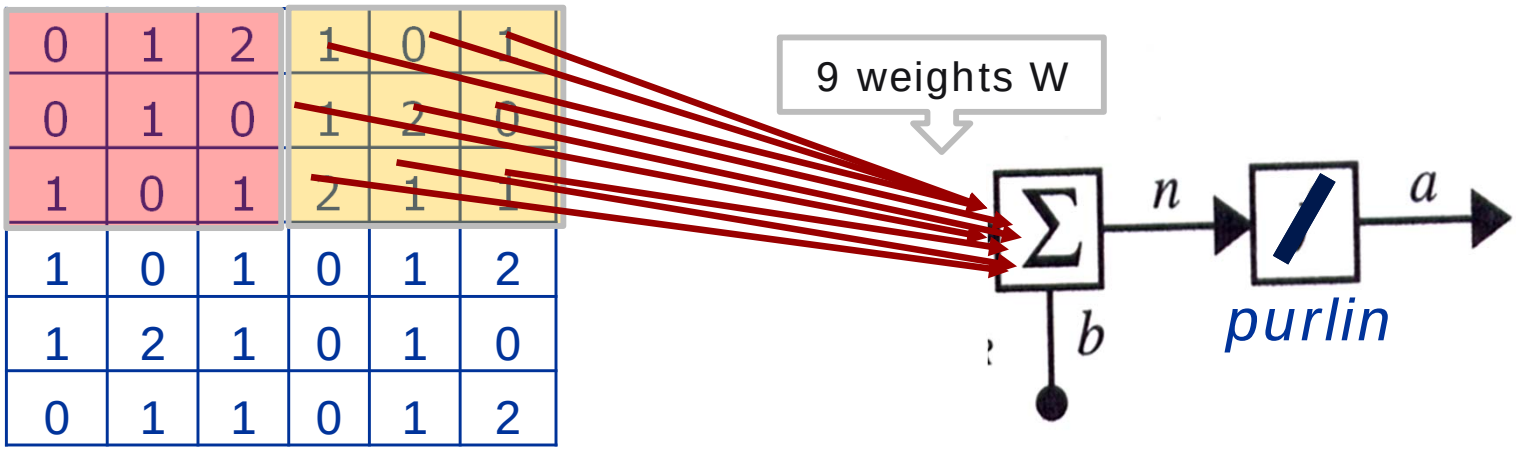
A

1x0+1x1+0x2+1x0+0x1+1x0+1x1+0x0+0x1+1= 3



The neuron covers a square 3x3, in this example. In the DL Toolbox it is defined by one scalar, ex. 3, if squared, by a vector if rectangular, ex. [3 5]).
The neuron is also called **filter**, or **kernel**.

CNN- Convolution Layer



W

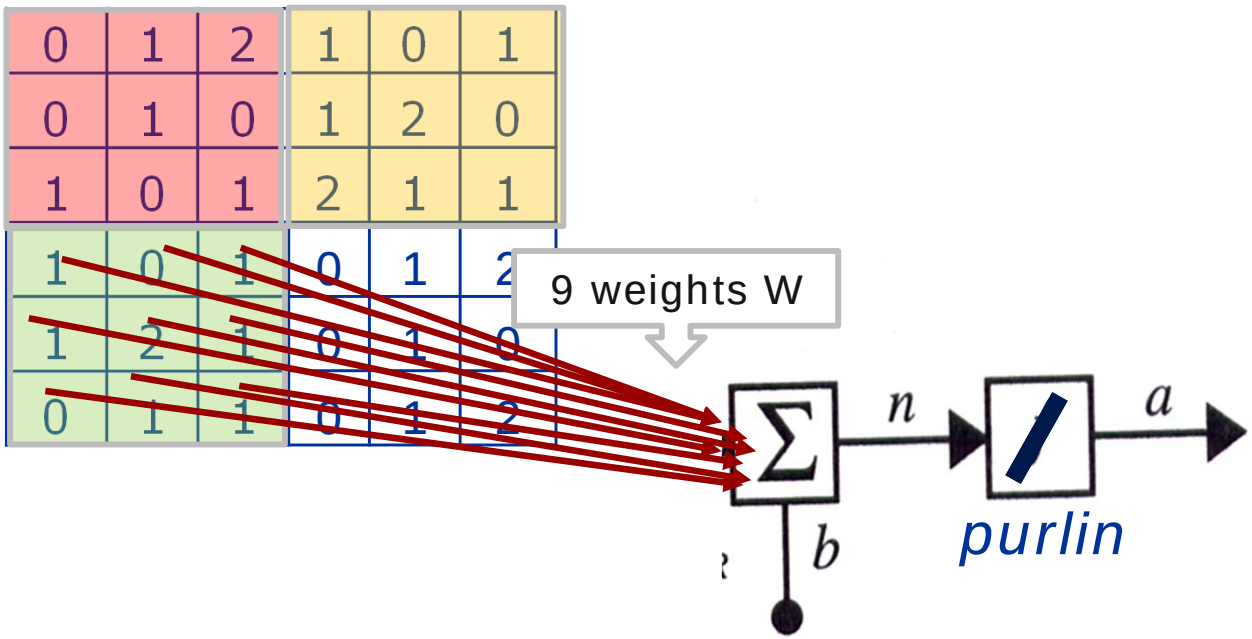
1	1	0
1	0	1
1	0	0

$b = 1$

A

3	5

CNN- Convolution Layer



W

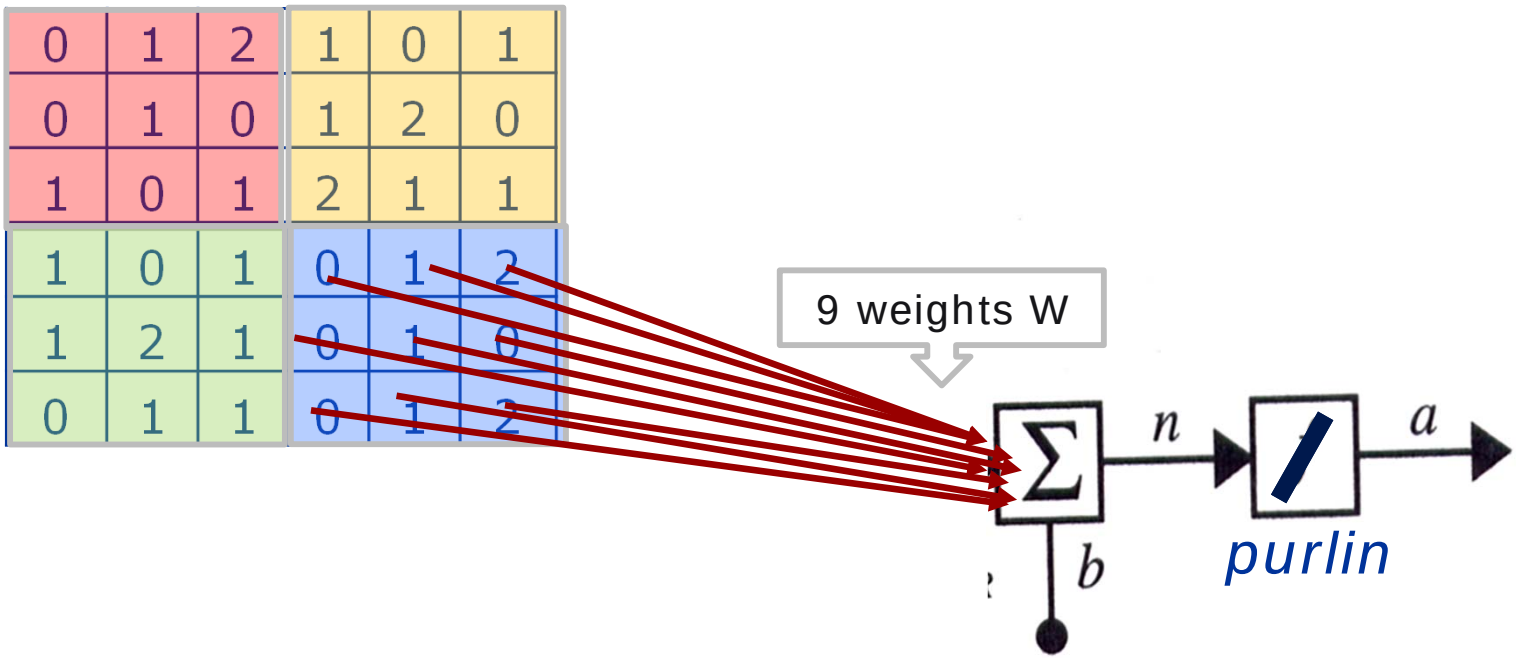
1	1	0
1	0	1
1	0	0

$b = 1$

A

3	5
4	

CNN- Convolution Layer



W

1	1	0
1	0	1
1	0	0

$b = 1$

A

3	5
4	2

CNN- Convolution Layer

P

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

W

1	**1**	**0**
1	**0**	**1**
1	**0**	**0**

$b = 1$

A

3	**5**
4	**2**

Convolution of the matrix P with the “filter” W (plus *b*)

The name CNN derives from this operation. The (first) convolutional layer is the matrix of the outputs of the filter. The filter is applied to a subregion of the input matrix. They may overlap. **The weights and bias are the same for all sub regions, i.e., there is only one neuron moving along the input matrix .**

CNN- Convolution Layer, moving step, stride

P

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

W

1	**1**	**0**
1	**0**	**1**
1	**0**	**0**

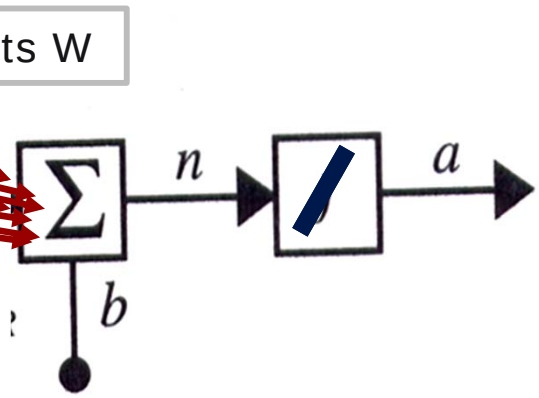
***b* = 1**

A

6A2 CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

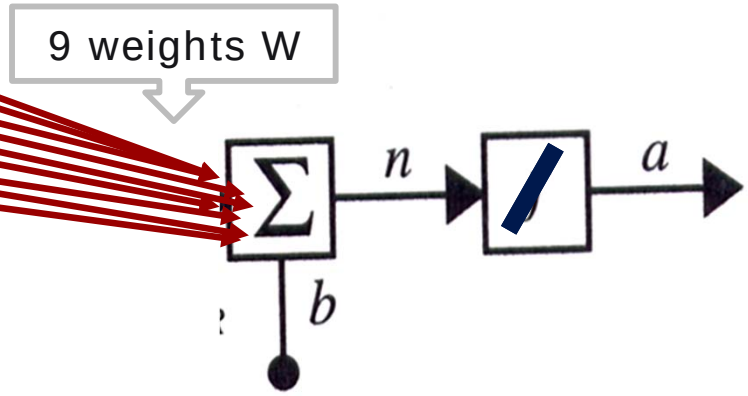
A

3			

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

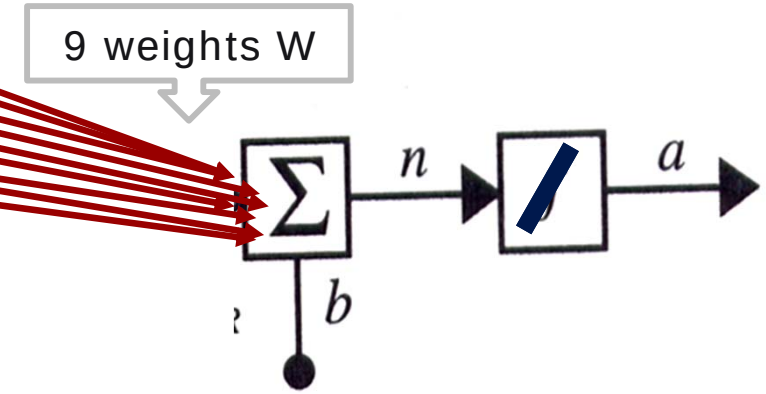
A

3	5		

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

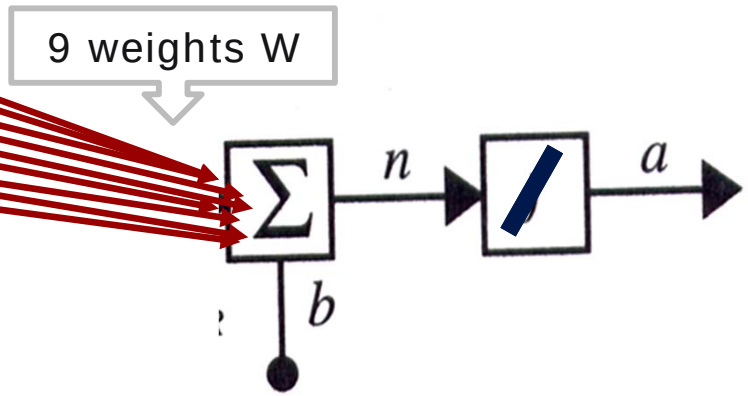
A

3	5		

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

b = 1

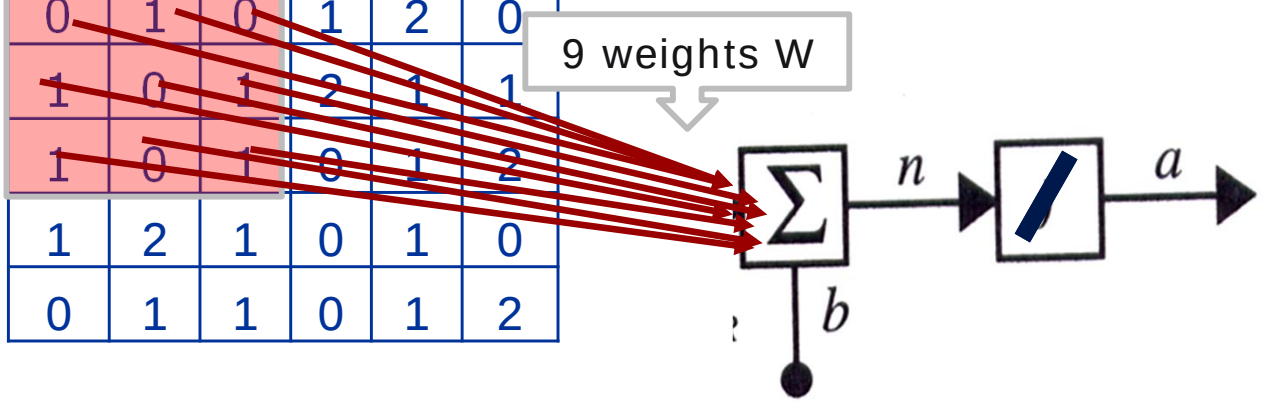
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

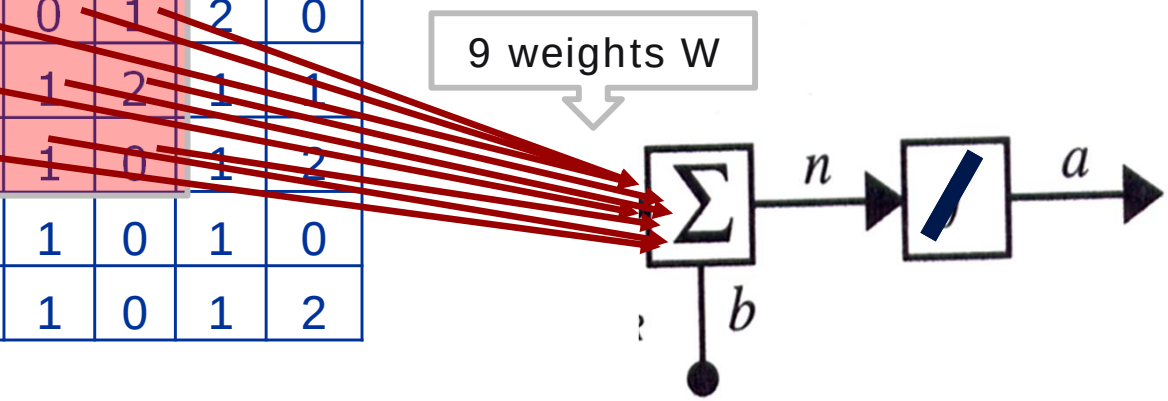
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

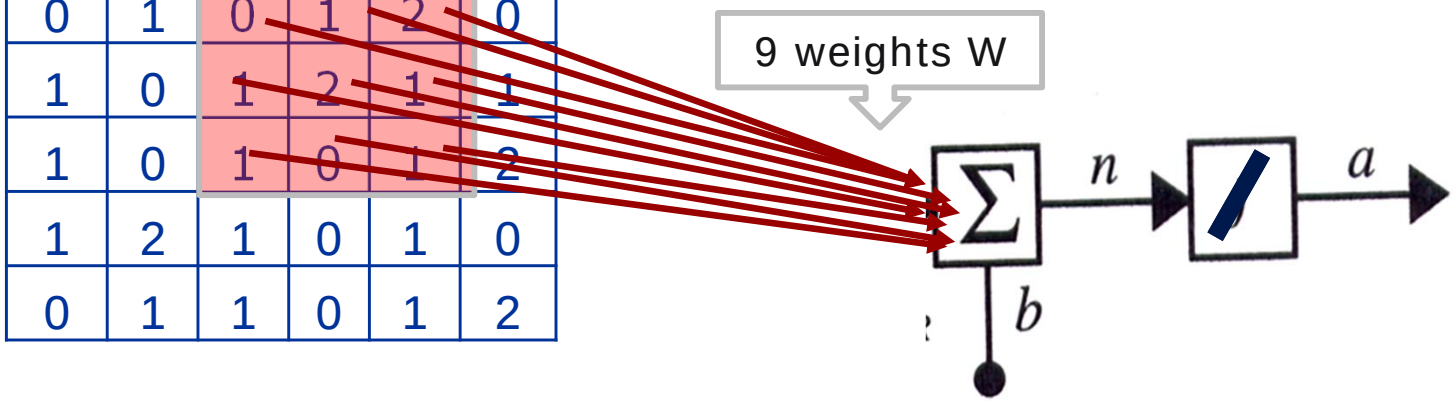
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

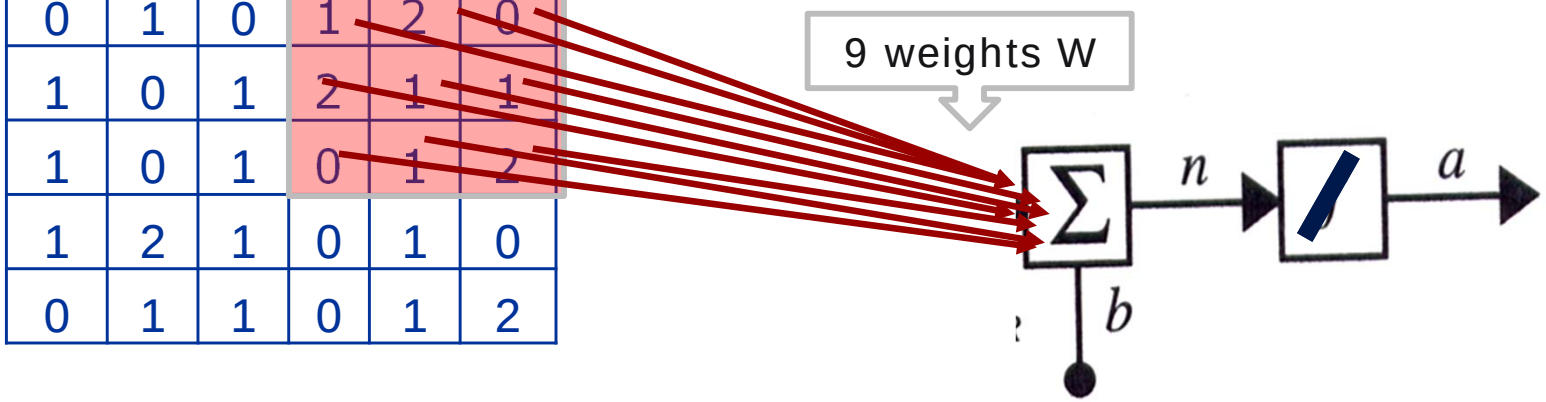
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

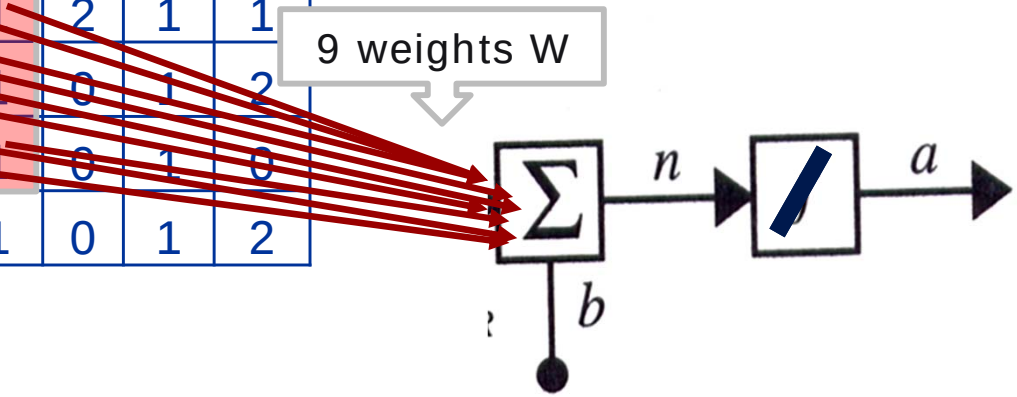
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

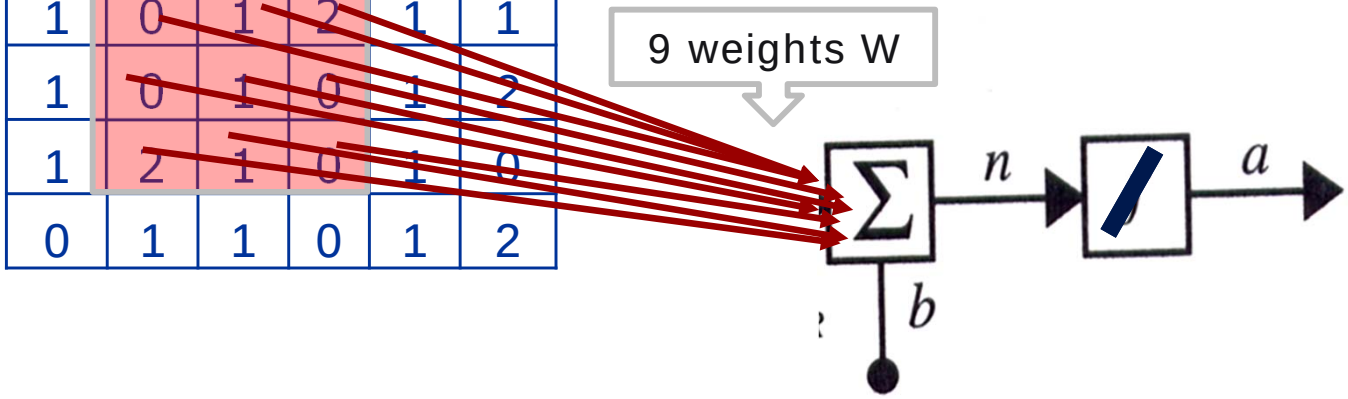
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

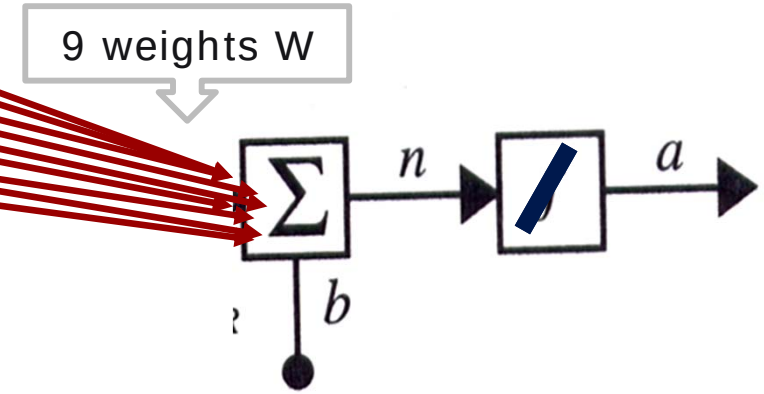
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

A

3	5	4	

CNN- Convolution Layer, moving step, stride

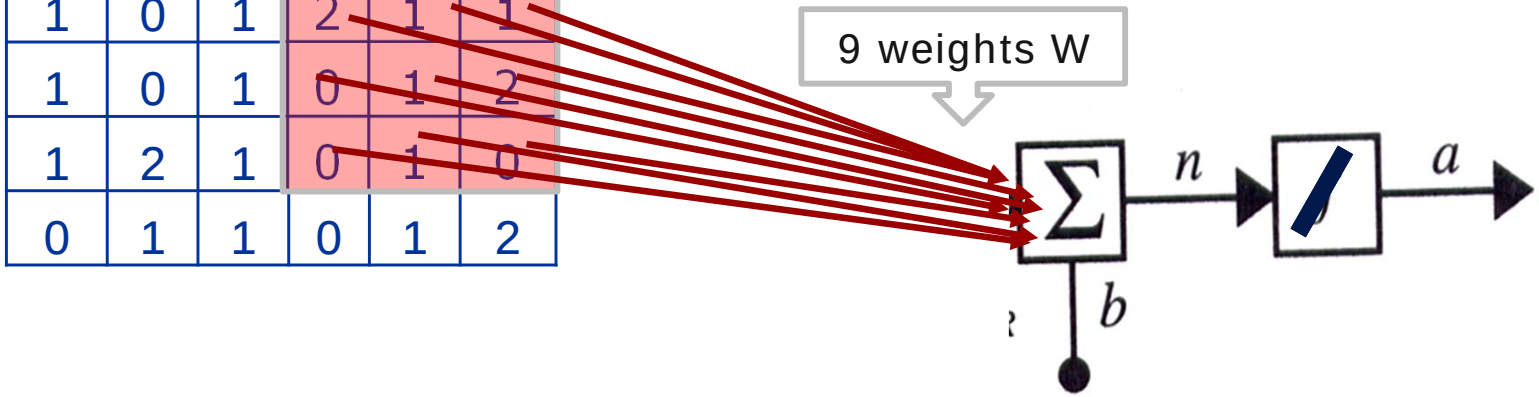
P

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

W

1	1	0
1	0	1
1	0	0

$b = 1$



A

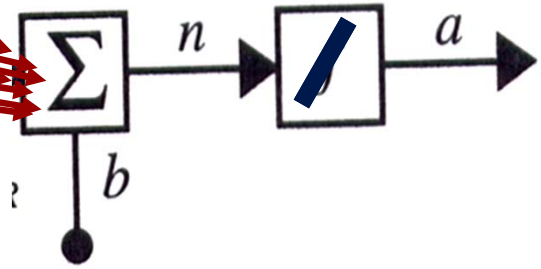
3	5	4	

CNN- Convolution Layer, moving step, stride

P

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

9 weights W



W

1	1	0
1	0	1
1	0	0

$b = 1$

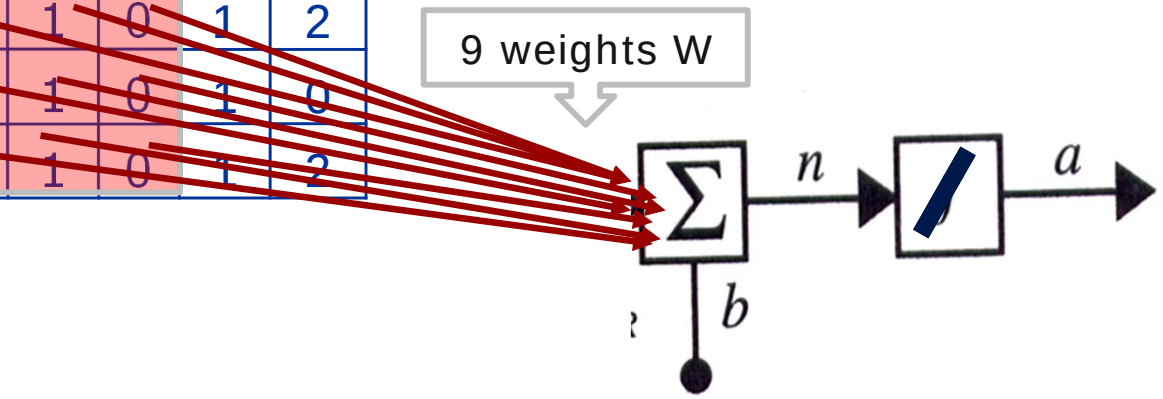
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

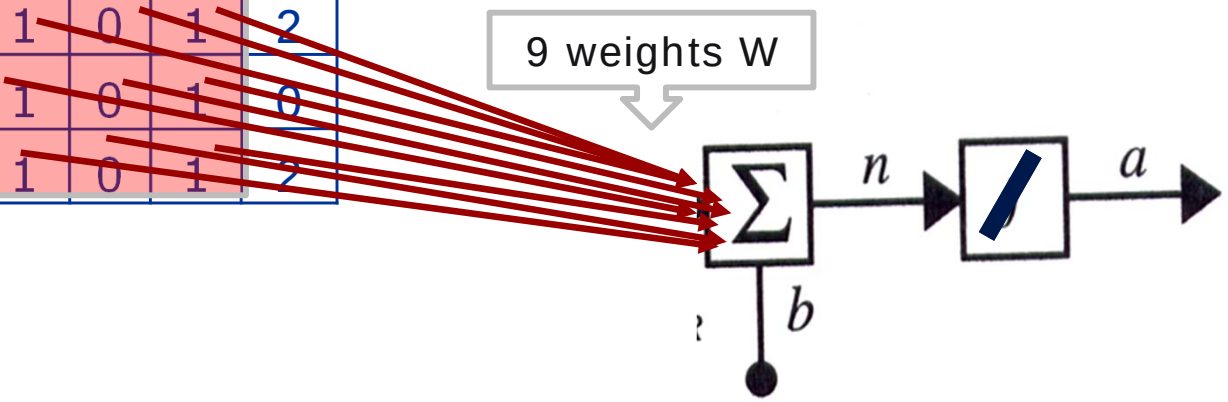
A

3	5	4	

CNN- Convolution Layer, moving step, stride

P

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



W

1	1	0
1	0	1
1	0	0

$b = 1$

A

3	5	4	

CNN- Convolution Layer, moving step, stride

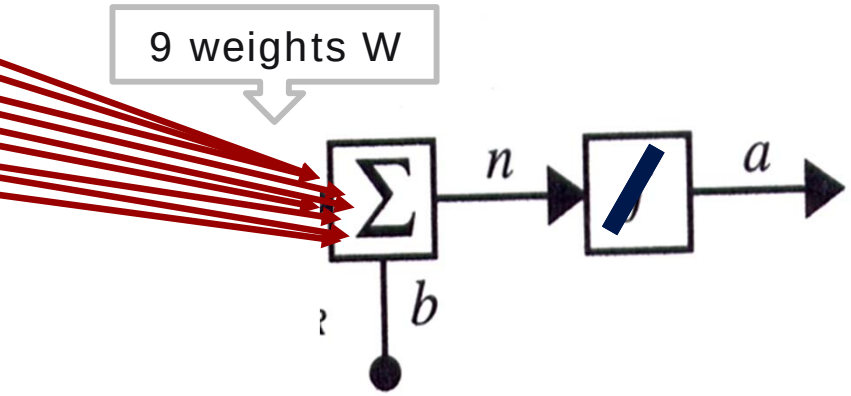
P

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

W

1	1	0
1	0	1
1	0	0

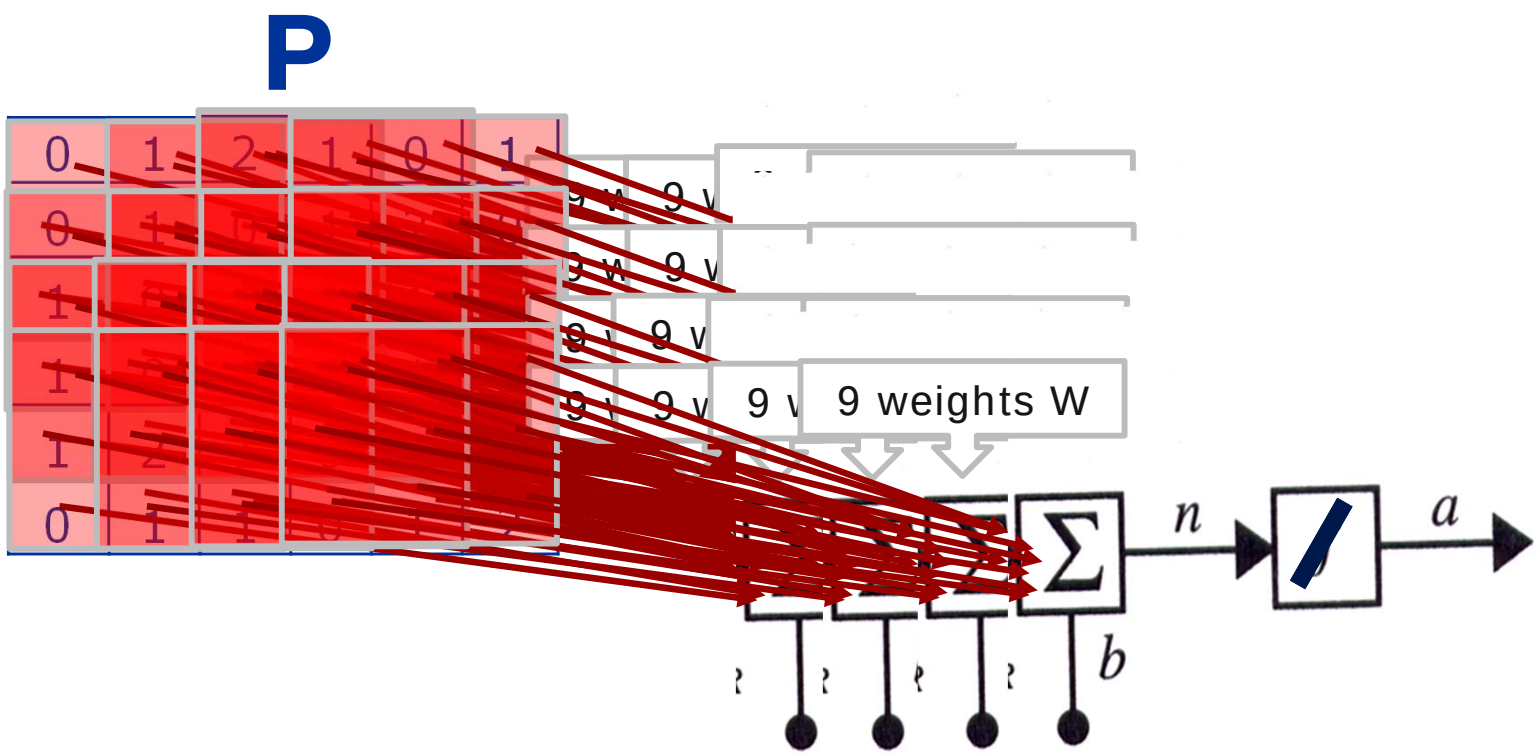
$b = 1$



A

3	5	4	

CNN- Convolution Layer, moving step, stride



W

1	1	0
1	0	1
1	0	0

$b = 1$

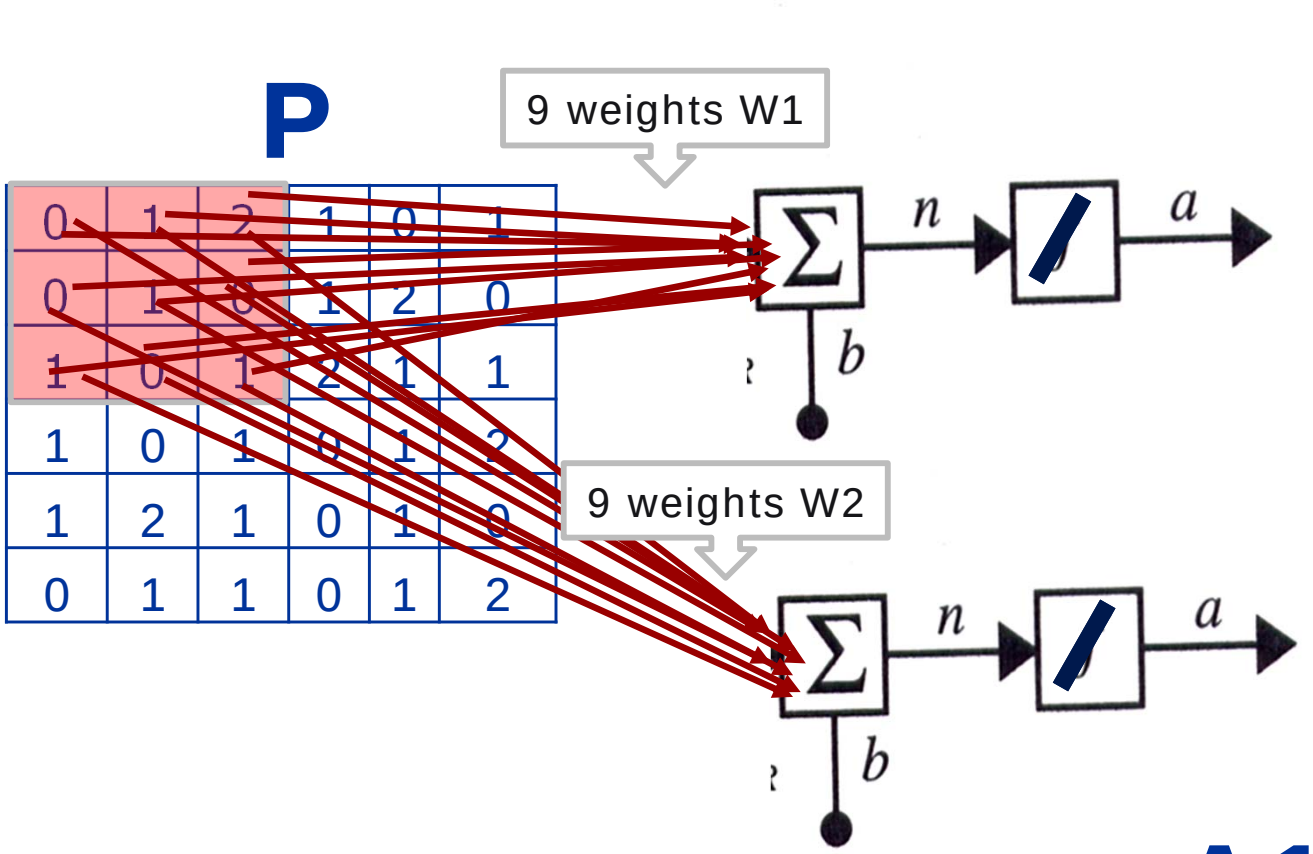
A

3	5	4	

(complete the table ...)

The matrix **A** is a **feature map** of the input matrix, extracted by the filter.
In the example the filter is of **size** 3x3 and the **stride** (moving step) is 1.

CNN- Convolution Layer, several filters



W1

1	1	0
1	0	1
1	0	0

W2

0	1	0
0	1	1
1	0	1

b1 = 1 ***b2 = 2***

Two filters !

A1

3			

A2

6			

There are two neurons moving along the input matrix.
We can convolute with several filters, obtaining several **feature maps**.

(zero) padding: zeros are introduced in lateral lines and/or columns:

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	0	1	2	1	0	1	**0**
0	0	1	0	1	2	0	**0**
0	1	0	1	2	1	1	**0**
0	1	0	1	0	1	2	**0**
0	1	2	1	0	1	0	**0**
0	0	1	1	0	1	2	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**

this allows to control the size of the output layer (i.e.., the size of the feature maps).

With a 3x3 filter and stride 1, gives a 6x6 output A

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

W1

1	1	0
1	0	1
1	0	0

b1 **= 1**

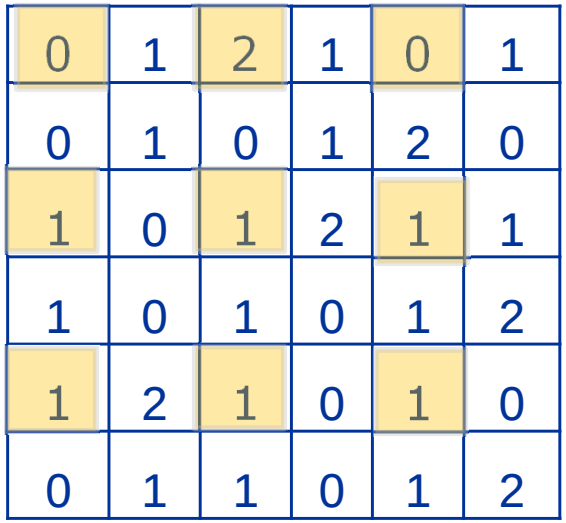
A1

2					

... complete !

Dilated convolution

Filters are expanded in input space by inserting spaces between the elements of the filter.
Increases the receptive field without increasing the number of weights.



W1

1	1	0
1	0	1
1	0	0

b1 **= 1**

A1

6	

Complete …

a 3x3 filter with dilation factor 2 (factor 1 no dilation) and stride 1

For more look at:

https://deeplearning.net/software/theano/tutorial/conv_arithmetic.html
(with animation)

https://keras.io/layers/convolutional/ 3 Oct 2023

https://cs231n.github.io/convolutional-networks/
(with animation) 3 Oct 2023

Output size=

$$(Input\ zize - ((Filter\ size - 1) * Dilation\ factor + 1) + 2 * Padding) / Stride + 1$$

must be an integer, if not part of the image will not be covered.

A CNN can have several convolutional layers in series.

Other layers in CNN

After the convolution some operations are made to the output of the convolution layer:

(i)- ReLU layer

This layer applies to the convoluted output a threshold operation by which any value less than zero is set to zero, i.e., a rectified linear unit (ReLU, (the poslin of Chapter 4) is applied. The rationale is that CNNs have been developed for image processing, where the data are the pixels' intensity in [0 1], and it does not make sense that the feature maps have negative values. However, this in not useful in all applications. There are some variations:

$$\sigma(x) = \begin{cases} x, x \geq 0 \\ 0, x \prec 0 \end{cases}$$

ReLU

(poslin)

$$\sigma(x) = \begin{cases} x, x \geq 0 \\ \alpha x, x \prec 0 \end{cases}$$

leaky ReLU

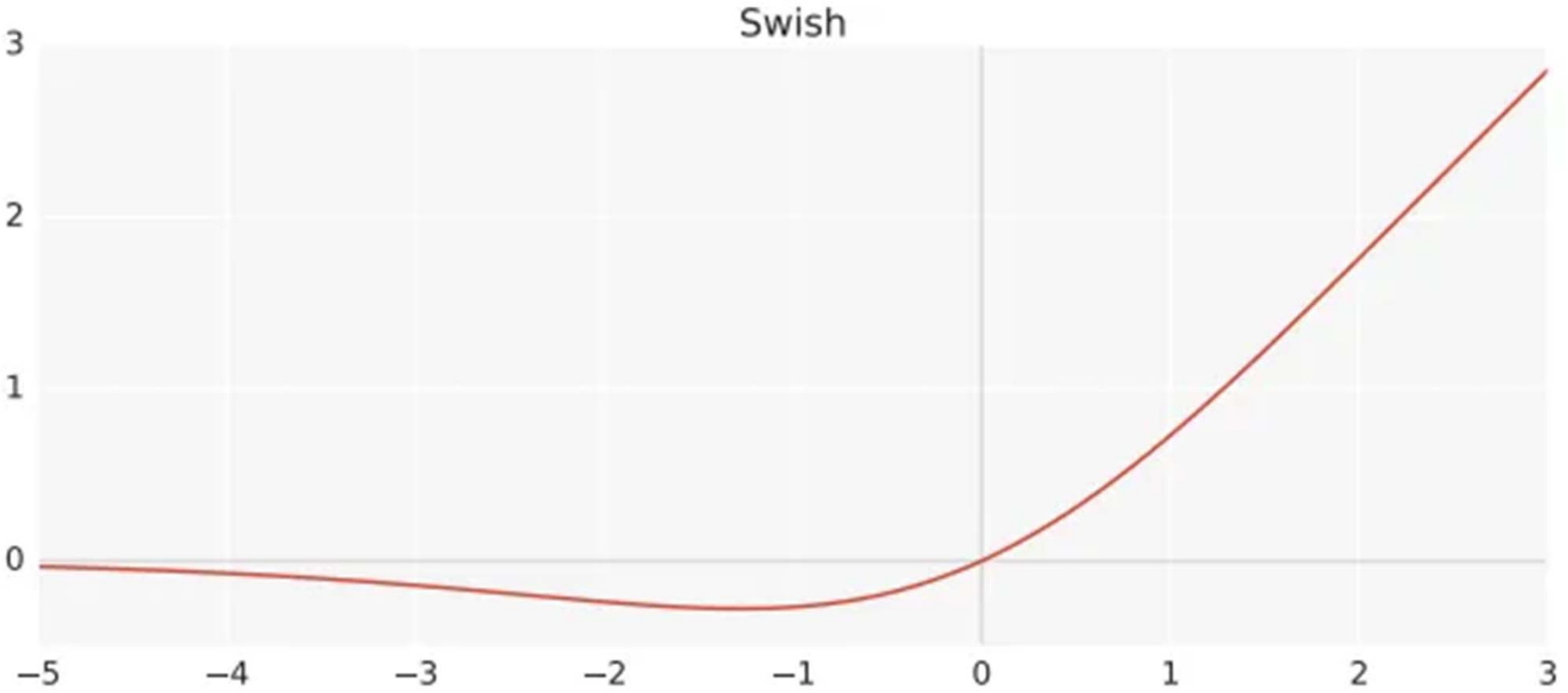
If alfa=1, it becomes the normal linear layer, which may be good when data can be negative or positive; by default is 0.01

$$\sigma(x) = \begin{cases} clipping\ ceiling, x \geq clipping\ ceiling \\ x, 0 \leq x \leq clipping\ ceiling \\ 0, x \prec 0 \end{cases}$$

clipped ReLU

(the positive satlin of chapt. 4)

(ii) swishLayer, can produce better resuts that reluLayer by taking into account some negative information, and it is a continuous function, easing backpropagation



https://medium.com/@neuralnets/swish-activation-function-by-google-53e1ea86f820
17 oct 2023

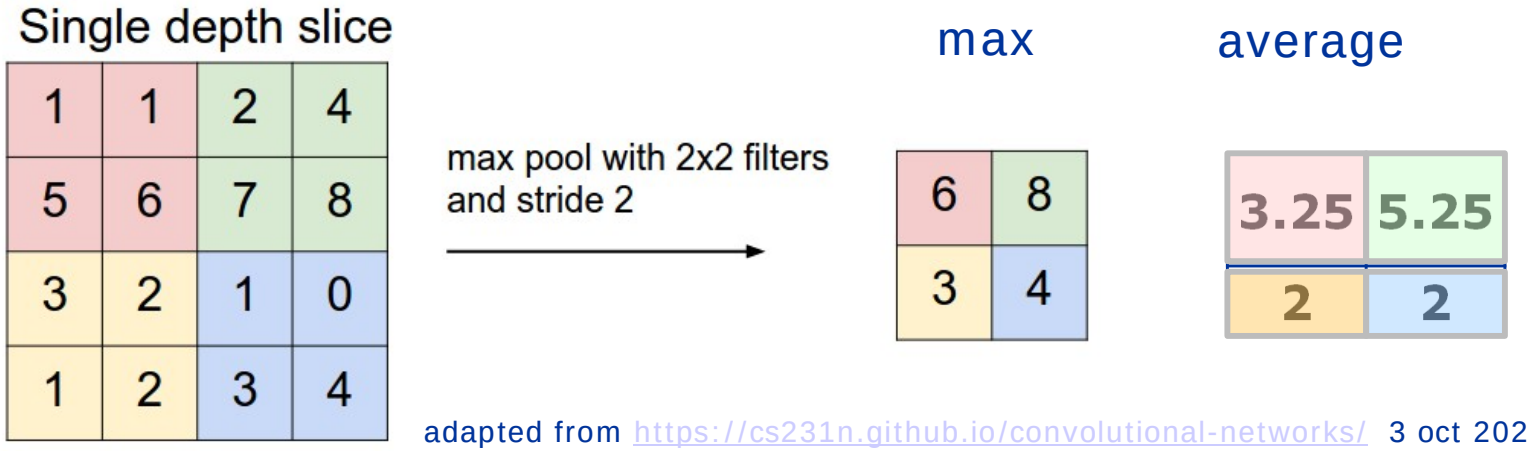
$$f(x) = \frac{x}{1+e^{x}} \qquad f(x) = x.sigmoid(x)$$

In Matlab 2023b: >help swishLayer

(iii) Max and average pooling layers

Down-sampling the convoluted layer, reducing the number of features, may be useful to lower the number of parameters to be learned in following layers.

Max pooling layers divide their input into rectangular pooling regions and compute the maximum of each region. Stride is also a parameter. *maxPooling2dLayer*



adapted from https://cs231n.github.io/convolutional-networks/ 3 oct 2023

Average pooling layers divide their input into rectangular pooling regions and compute the average of each region. Stride is also a parameter. *averagePooling2dLayer*

(iv) batch normalization layer

Normalization can favor training of the CNN. Usually, the inputs to the CNN are normalized to [-1 1], [0,1] or to zero mean and unit variance (whitening). But this normalization is lost in the intermediate layers, after all the calculations in convolution, pooling, etc.. So it can be convenient to normalize again the intermediate layers to zero mean and unit variance. This is done by a *batchnormalization layer.* The normalization is done in two steps:

- first compute $x^*=(x-E[x])/sqrt(var(x))$ for all x in the batch (a batch can be a single features map or a subset of all the features maps). Note that if the batch is composed by M images and each mage is PxQ, then the mean and variance are computed over the MxPxQ points.
- then compute $x^{**}= \gamma.x^*+\beta$; x^{**} is the final normalized value, γ and β are parameters learned per layer.

It has many advantages for the training stage; it increases the learning speed. See for example in https://github.com/aleju/papers/blob/master/neural nets/Batch_Normalization.md (**3** oct 2023) for a good synthesis.

(v) fully connected layers

fullyConnectedLayer

Follows the convolution and pooling layers. May be one or more.

They have usually ***purlin*** activation functions. All of its neurons connect to all the neurons in the previous layer; each neuron in the first fully connected layer connects through a weight to each cell of the features map issued from the last pooling layer, or the last convolution layer if it is the last). They have weights *W* and bias *b*.

For classification problems the output size of the last fully connected layer is equal to the number of classes of the data set.

The learning rate and the regularization parameters of these layers can be adjusted and have default values in the toolbox.

(vi) softmax layer

softmaxLayer

Follows the last fully connected layer (for classification problems). Applies a softmax function to the input:

If the input vector of this layer is *x* and the output vector is *y*, then the r^{th} component of the output is

$$y_r(x) = \frac{\exp(a_r(x))}{\sum_{j=1}^{k} \exp(a_j(x))} \quad \text{where} \quad a_r(x) = \ln(P(x, \theta \mid a_r) P(a_r))$$

It is known as the *normalized exponential* and verifies:

$$0 \le y_r \le 1 \quad \text{and} \quad \sum_{j=1}^{K} y_j = 1 \,, \; K \text{ is the number of classes}$$

y_r is the probability that the input *x* belongs to the class *r*

(vii) classification layer

Follows the softmax layer (in case of classification problems).

Takes as inputs the outputs of the softmax function and assigns each input to one of the *K* mutually exclusive classes.

For that it minimizes the cross-entropy function for a 1-of-K coding scheme:

$$loss = -\sum_{i=1}^{Q}\sum_{j=1}^{K} t_{ij} \ln y_{ij}$$

Q : number of training examples
K: number of classes
t_{ij} : target for the *i*th input, i.e., the indicator that the *i*th input belongs or not to the class *j* (it is 1 or 0).
y_{ij} : is the softmax output (*ln* natural logarithm, base *e)*

(viii) dropout layer

A dropout layer randomly sets input elements to zero with a given probability (ex. 0.5). This can be useful to prevent overfitting.

```
layer = dropoutLayer
layer = dropoutLayer(probability)
layer = dropoutLayer(___,'Name',Name)
```

```
layers = [ ... imageInputLayer([28 28 1])
convolution2dLayer(5,20)
reluLayer
dropoutLayer (0.4)
fullyConnectedLayer(10)
softmaxLayer
classificationLayer] ;
```

drops out about 40% of its input elements, that are the outputs of of the reluLayer.

see online help in matlab
or >help dropoutLayer

Defining the structure of a CNNet

The structure of a CNN is built by specifying which layers it will have and their characteristics. For example, in the DL Toolbox, page 1-22 nnetug 2023b, the object layers is created by the concatenation of several layers:

```
layers = [imageInputLayer([28 28 1])
          convolution2dLayer(5,20)
          reluLayer
          maxPooling2dLayer(2,'Stride',2)
          fullyConnectedLayer(10)
          softmaxLayer
          classificationLayer];
```

20 filters

filter 5x5

filter 2x2

10 classes

The number and types of layers depends on the problem; the more layers, more time and resources will take to train. The users can define their own layers (see DL user's guide Chapt.19 2023b).

Specification of the training options of a CNNet

The training options specify the used algorithms and more. Algorithms available in the toolbox are different variants of the stochastic gradient descent. For example

```
options = trainingOptions('sgdm');
```

specifies the stochastic gradient descent with momentum algorithm (solver).

For the complete list of options and their values see

>help trainingOptions

Defining the data for training

Let the input data be named XTrain and the output (target data) YTrain.

XTrain is an array of four dimensions (a tensor, ex. made up of colored images). For example, Xtrain (:,:,1,5) is the 5th matrix of data (black and white image, sequences, time-series for a time interval, etc.)

YTrain is a categorical vector containing the labels for each observation in XTrain, i.e, the label of the class to which each matrix in Xtrain belongs. Its dimension is equal to the size of the fourth dimension of Xtrain, ex. the number of images.

Training the CNNet

With the architecture defined and the options specified, the CNN is ready to be trained, if the data is available. This is done simply by, for example

```
convnet = trainNetwork(data,layers,options);
```

To prepare the data in good shape is a decisive step.

Read carefully the information in >*help trainNetwork*

Matrices with more that three dimensions are frequently named *tensors.* In the literature the inputs to a CNN are also frequently called tensors, because an array that has a collection of images is a matrix with four dimensions, a tensor.

Example of calls of CNN

trainedNet = trainNetwork(imds,layers,options) trains a network for image classification problems. imds stores the input image data (including the targets), layers defines the network architecture, and options defines the training options.

trainedNet = trainNetwork(mbds,layers,options) trains a network using the mini-batch datastore mbds, when it is impossible to consider all images at the same time for training. Use a mini-batch datastore to read out-of-memory data or to perform specific operations when reading batches of data.

trainedNet = trainNetwork(X,Y,layers,options) trains a network for image classification and regression problems. X, cell array, contains the predictor variables (the images) and Y, a categorical vector, contains the categorical labels or numeric responses (the labels of the class to which each image in X belongs)..

(adapted from the Matlab online reference page for trainNetwork)

Transfer learning

We can use a pre-trained network for a certain problem. There are numerous CNNs available (in Matlab, Keras, etc.), the most famous being the Google's net to classify images into 1000 classes, available in the DL Toolbox by

>> net=googlenet

It can be changed and retrained for another problem (for example to classify a collection of images into 10 classes), by changing the parameters of some layers. This can be easily done using the

>> deepNetworkDesigner

importing the googlenet to the designer and retraining it to our data.

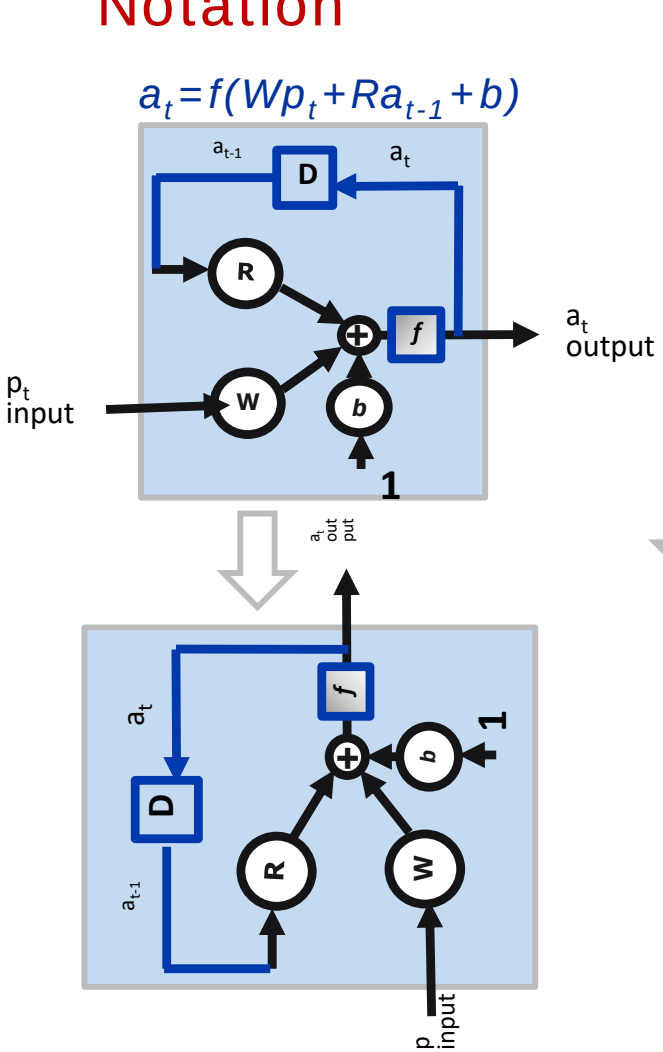
This transfers the initial learning of the net to the new net, and it is called transfer learning.

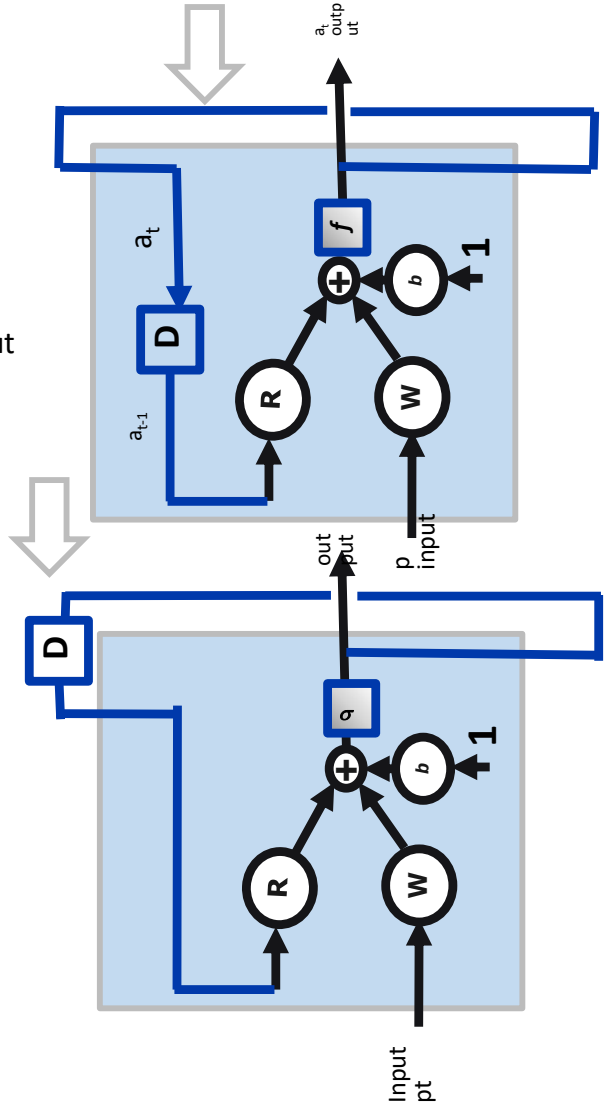
See the video https://www.mathworks.com/videos/interactively-modify-a-deep-learning-network-for-transfer-learning-1547157074175.html

5.4 LSTM Long Short Term Memory Networks

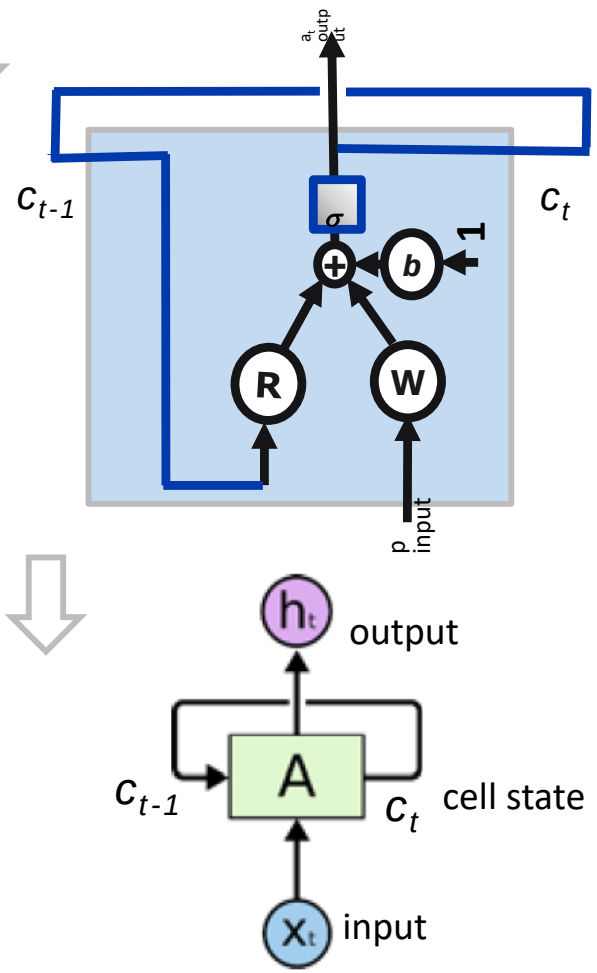
Notation

$a_t = f(Wp_t + Ra_{t-1} + b)$





$a_t = \sigma(p_t,\ a_{t-1}, b)$

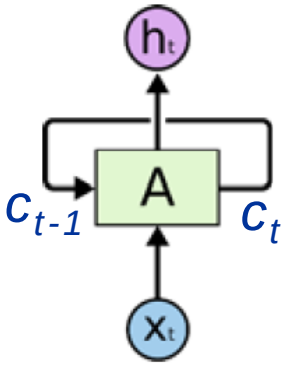


(new notation; delay is implicit; output and cell state can be different variables)

$h_t = \sigma(X_t,\ c_{t-1})$

LSTM Long Short Term Memory Networks

- LSTM are recurrent Neural Networks (RNN) spanning in space the memory in time.

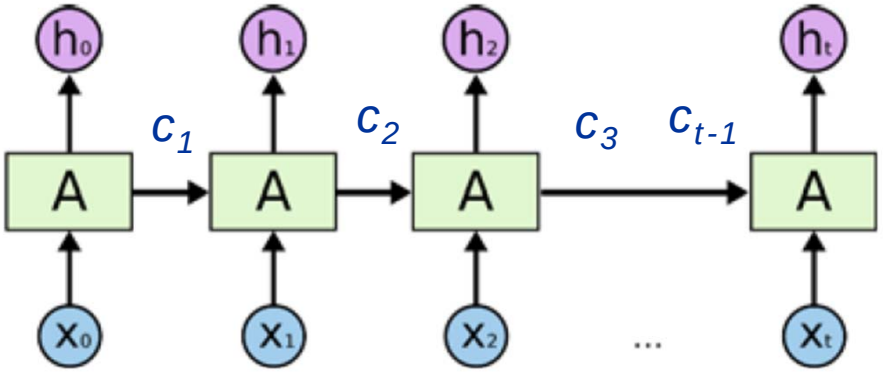


A recurrent NN concentrated in space has

$$h_t = \sigma(X_t, c_{t-1}) \text{ , } t = 0,1,2,\ldots \text{ and } c_{-1} = 0$$

c is the (internal) state of the NN.
h is the output (hidden state).

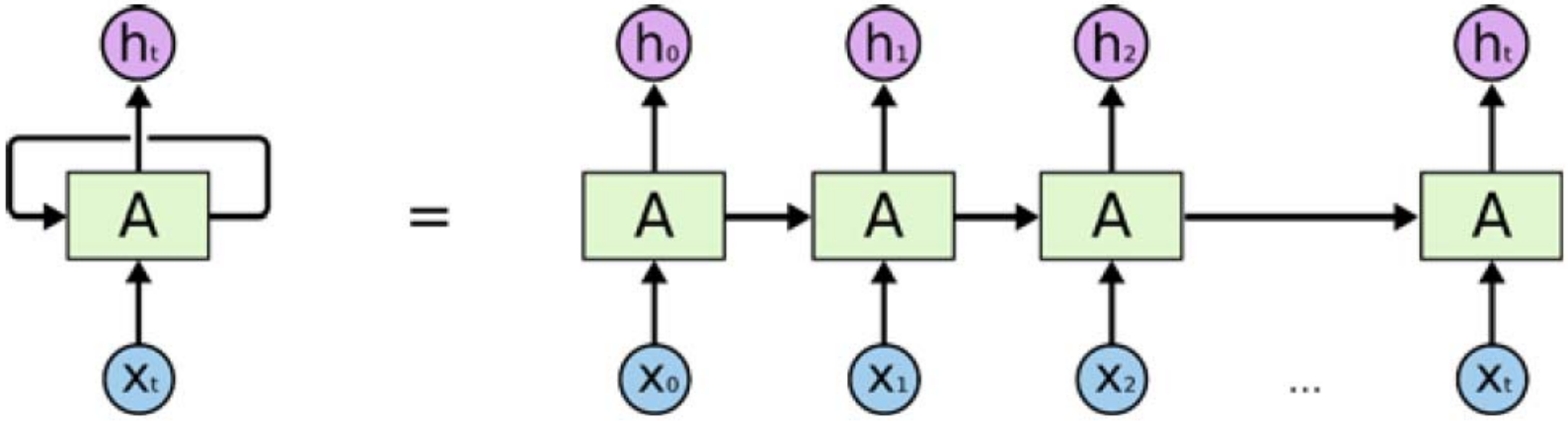
can be the same in RNN



unrolling the loop, we obtain an equivalent non-recurrent NN spanning in space as much as we need.

$$h_t = \sigma(X_t, c_{t-1}) \text{ , } t = 0,1,2,\ldots \text{ and } c_{-1} = 0$$

adapted from http://colah.github.io/posts/2015-08-Understanding-LSTMs/ 3 oct 2023

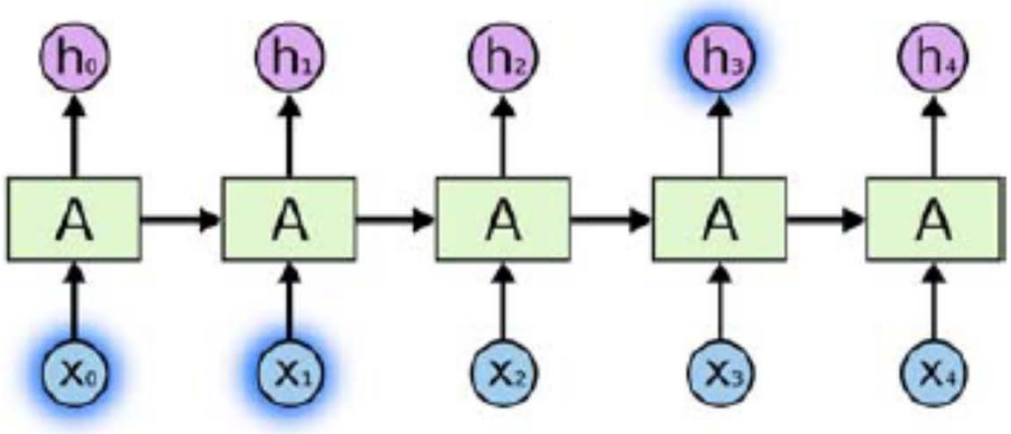


An unrolled recurrent neural network.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 1 nov 2023

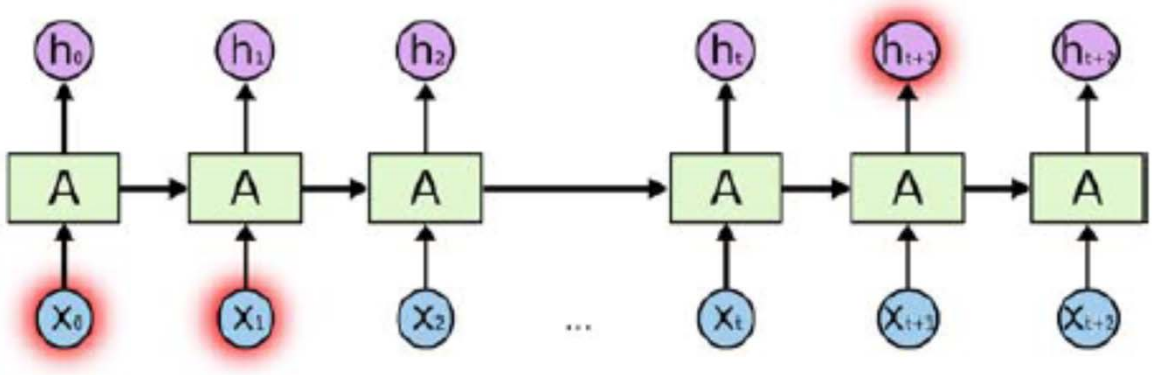
A RNN can be thought of as multiple copies of the same NN, each passing its state to a successor. This evidences that they are intimately related to sequences, ex. time series .
Note that after enrollment, formally there is no more feedback, but its effect remains.

That is the inspiration of LSTM.



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Short-term dependencies: h_3 depends on x_0, and x_1
RNN can do it.

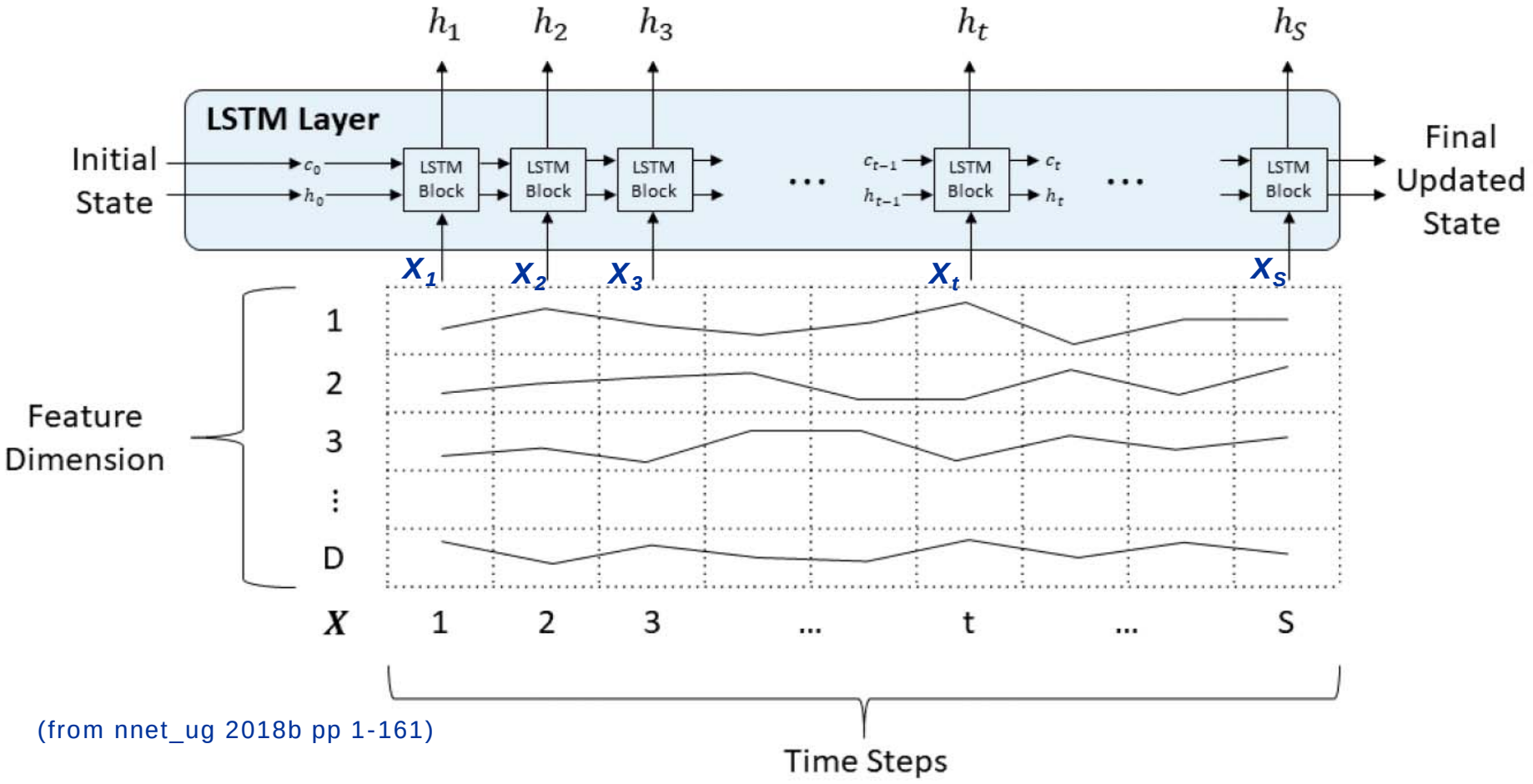


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long-term dependencies: h_{t+1} depends on x_0, x_1
RNN cannot do it.

LSTM Long Short Term Memory Networks

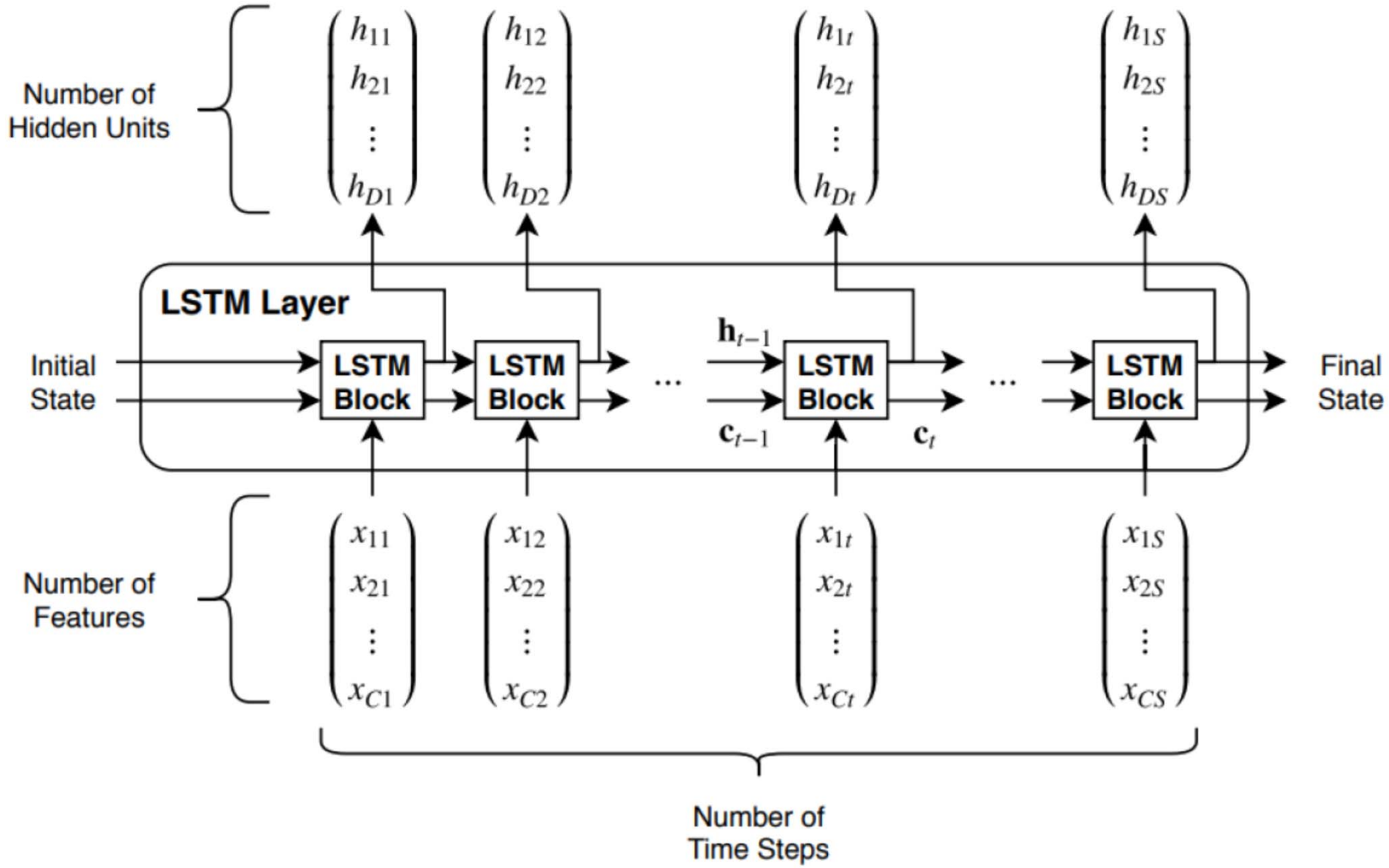
NN capable of learning long-term dependencies in sequences



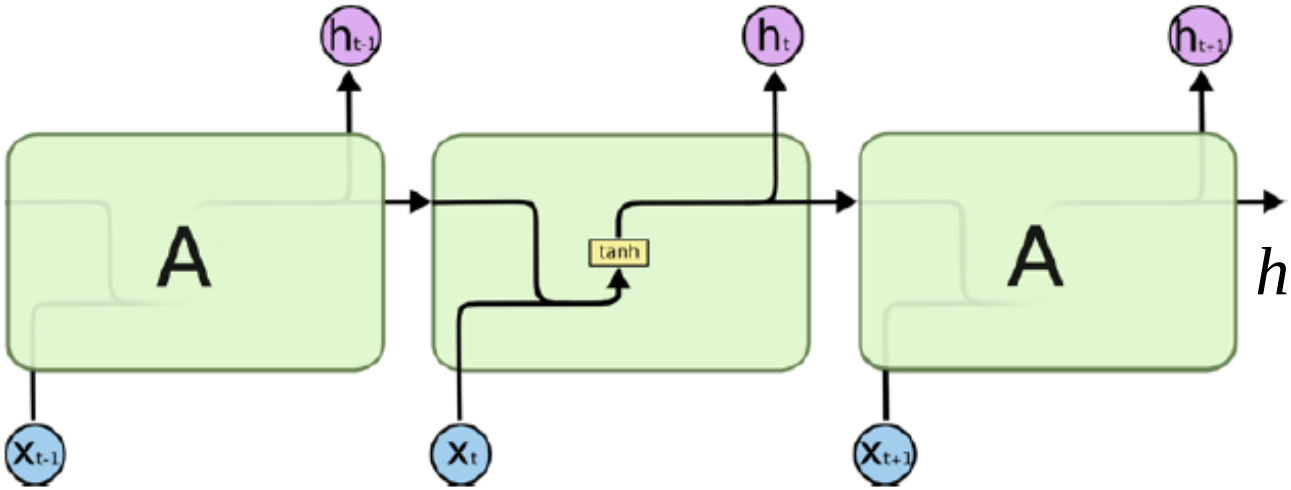
(from nnet_ug 2018b pp 1-161)

long term memory: because of the high number of serial blocks
short term memory: because each block uses only the previous state

(from nnet_ug 2023b, pp 1-111)



RNN Block

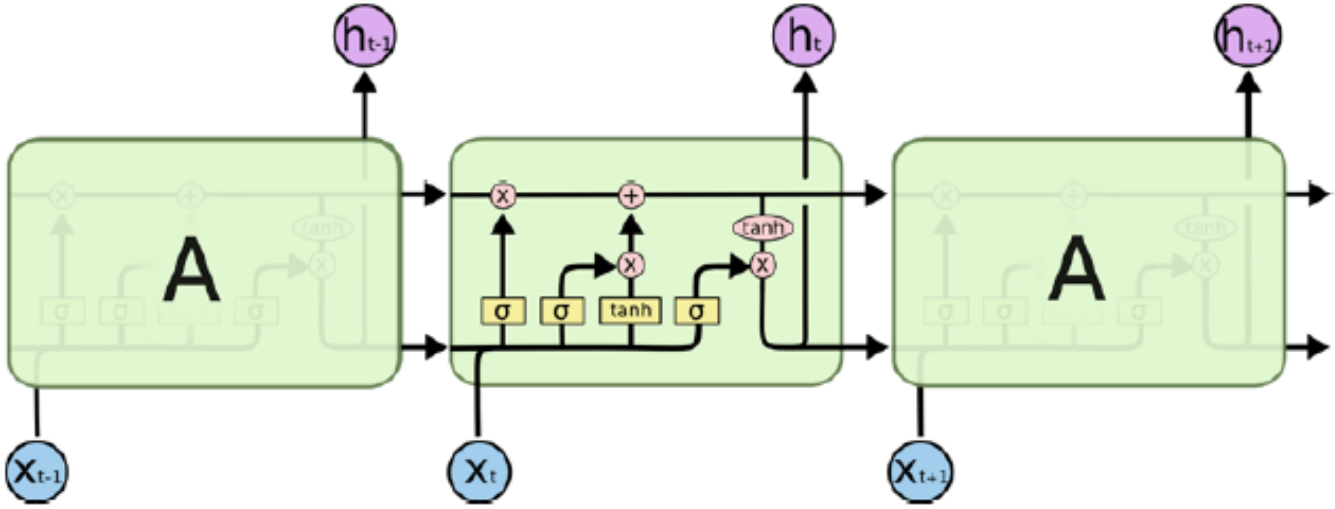


one single neuron, ex.

$$h(t) = \tanh(Wx_t + Rh_{t-1} + b)$$

LSTM Block

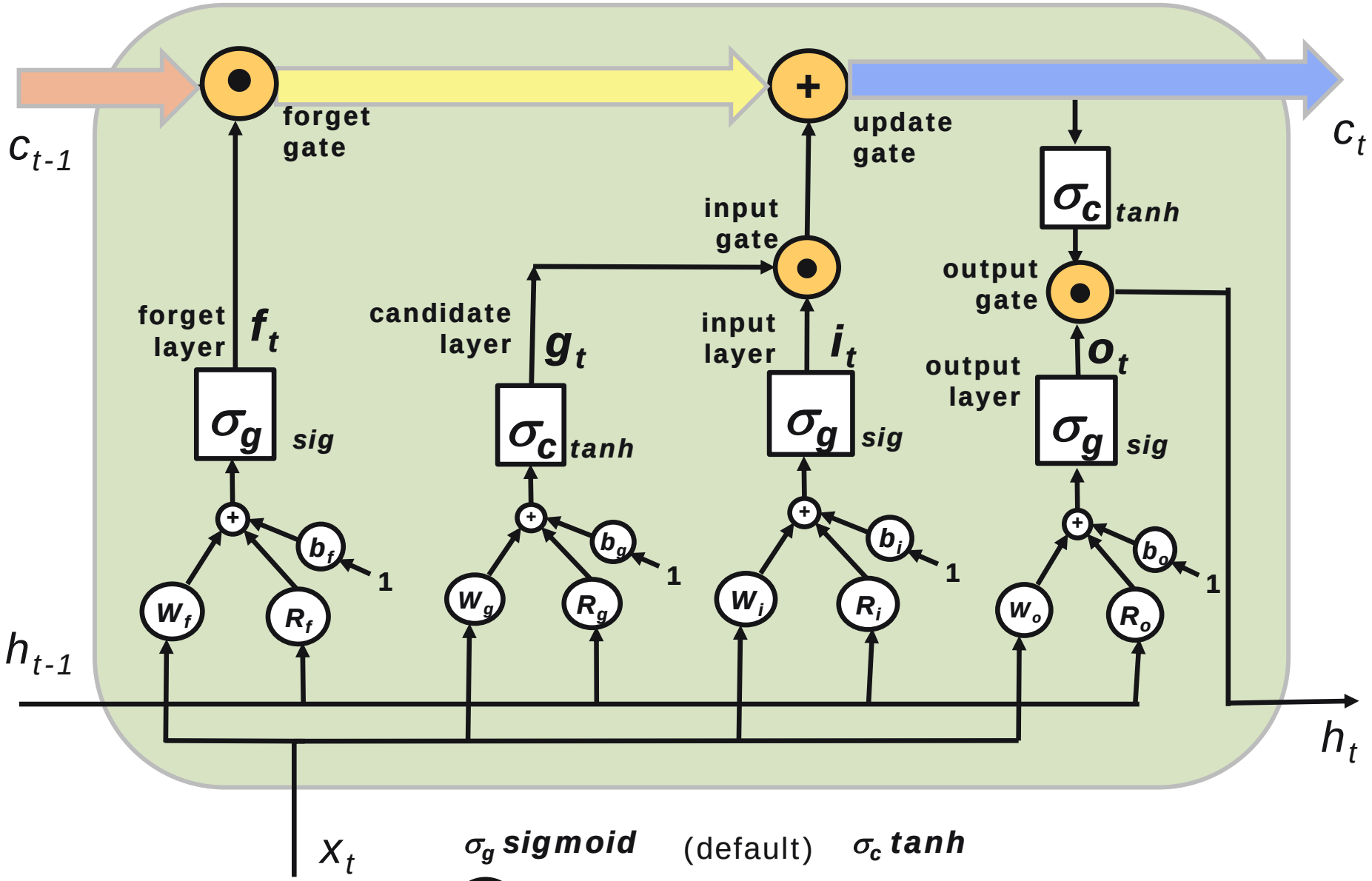
http://colah.github.io/posts/2015-08-Understanding-LSTMs/ 17 oct 2023



four interacting layers (*X* and *h* are vectors, the solid lines are vectorial, yellow blocks have several neurons in parallel)

LSTM Block in detail

solid lines are vectorial



LSTM Block

The block has four gates where element-wise operation are performed (addition, Hadamard multiplications).

- the **forget gate**, where f_t multiplies c_{t-1} by a forgetting factor between 0 and 1 to “forget” elements from the cell state,
- the input layer decides which values (of the state) will be updated (in [0 1]),
- the candidate layer gives a vector of new candidate values that could be added or subtracted to the state (in [-1 1]); which is added or subtracted is decided in the **input gate** where i_t multiplies g_t and the result is summed to the state in the **update gate,**
- the **output gate** where o_t multiplies the output of the state activation function,
- σ_c denotes the state activation function; by default, it is the hyperbolic tangent (*tanh*) to give output between -1 and 1,
- σ_g denotes the gate activation function; by default, it is the sigmoid function, to give outputs between 0 and 1.

Parameters to be learned:

- input weight matrix *W* (inputWeights)
- recurrent weight matrix R (recurrentWeights)
- bias *b* (Bias)

$$W = \begin{bmatrix} W_i \\ W_f \\ W_g \\ W_o \end{bmatrix}, R = \begin{bmatrix} R_i \\ R_f \\ R_g \\ R_o \end{bmatrix}, b = \begin{bmatrix} b_i \\ b_f \\ b_g \\ b_o \end{bmatrix},$$

(from nnug_2023b pp 1-113)

Component	Formula	
Input gate	$i_t = \sigma_g(W_i x_t + R_i h_{t-1} + b_i)$	σ_g ***sigmoid*** (default)
Forget gate	$f_t = \sigma_g(W_f x_t + R_f h_{t-1} + b_f)$	
Cell candidate	$g_t = \sigma_c(W_g x_t + R_g h_{t-1} + b_g)$	σ_c ***tanh*** (default)
Output gate	$o_t = \sigma_g(W_o x_t + R_o h_{t-1} + b_o)$	

$$h_t = o_t \odot \sigma_c(c_t) \qquad c_t = f_t \odot c_{t-1} + i_t \odot g_t .$$

Training LSTM NN

i) for classification

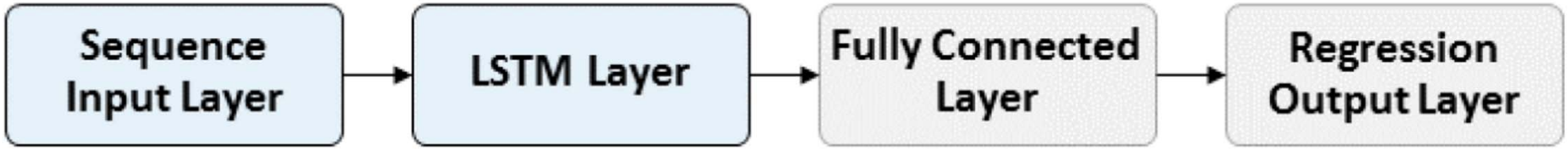
example:

(from nnug_2023b pp 1-103)

```
numFeatures = 12;
numHiddenUnits = 100;
numClasses = 9;
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

Training LSTM NN

ii) for regression



example:

(from nnug_2023b pp 1-103)

```
numFeatures = 12;
numHiddenUnits = 125;
numResponses = 1;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

iii) define training options

example:

```
maxEpochs = 100;
miniBatchSize = 27;

options = trainingOptions('adam', ...
'ExecutionEnvironment','cpu', ...
'GradientThreshold',1, ...
'MaxEpochs',maxEpochs, ...
'MiniBatchSize',miniBatchSize, ...
'SequenceLength','longest', ...
'Shuffle','never', ...
'Verbose',0, ...
'Plots','training-progress');
```

Possible values for solverName include:

'sgdm' - Stochastic gradient descent with momentum.
'adam' - Adaptive moment estimation (ADAM).
'rmsprop' - Root mean square propagation (RMSProp).

iv) train the NN

```
net = trainNetwork(XTrain,YTrain,layers,options);
```

v) test the NN

```
miniBatchSize = 27;

YPred = classify(net,XTest, ...
'MiniBatchSize',miniBatchSize, ...
'SequenceLength','longest');
```

(see Matlab online help “Sequence classification using deep learning”)

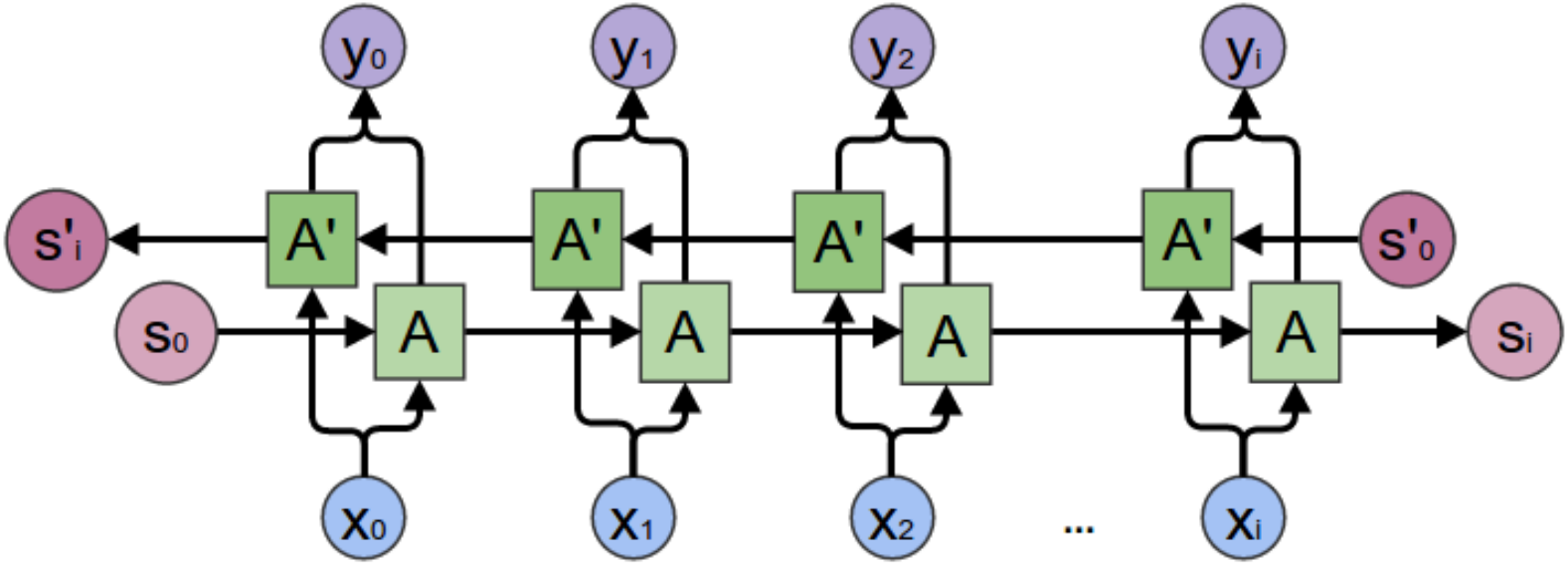
vi) bidirectional LSTM (BiLSTM) layer

In some applications, like natural language processing, automatic translation, etc., it may be useful to train the LSTM network with the complete time-series at each time step. At instant *k*, the learning algorithm uses all the data before *k* and after *k*. In a first run it goes from the first to the last sample, and in a second run it goes from the last to the first sample.

This includes the context influence in training. Translation, for example, depends strongly on the context (the correct translation of the first part of a sentence depends on the second part of it).

For this bi-directional LSTM, BiLSTM can be used.
After training it, the set of weights and bias have been obtained also with the context influence, so it is expected that translation in real time will be improved by BiLSTM.

https://paperswithcode.com/method/bilstm# 17/10/2023
Graves, Alex, Santiago Fernández, and Jürgen Schmidhuber. "Bidirectional LSTM networks for improved phoneme classification and recognition." Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005. Springer Berlin Heidelberg, 2005. 799-804 17/10/2023
https://en.wikipedia.org/wiki/Bidirectional_recurrent_neural_networks 17/10/2023
Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations
Eliyahu Kiperwasser, Yoav Goldberg, https://www.aclweb.org/anthology/Q16-1023/
https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66 17/10/2023

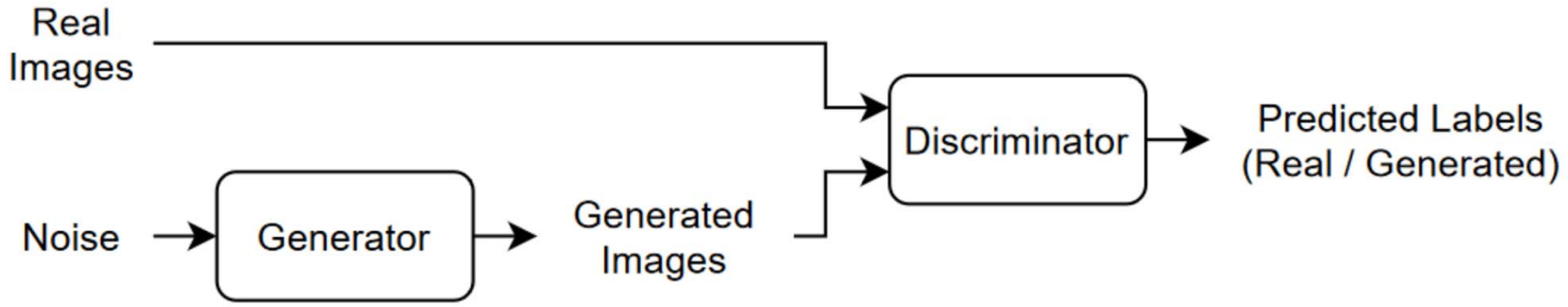


https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66 3 Oct 2023

5.5 Generative NNets&Transformers

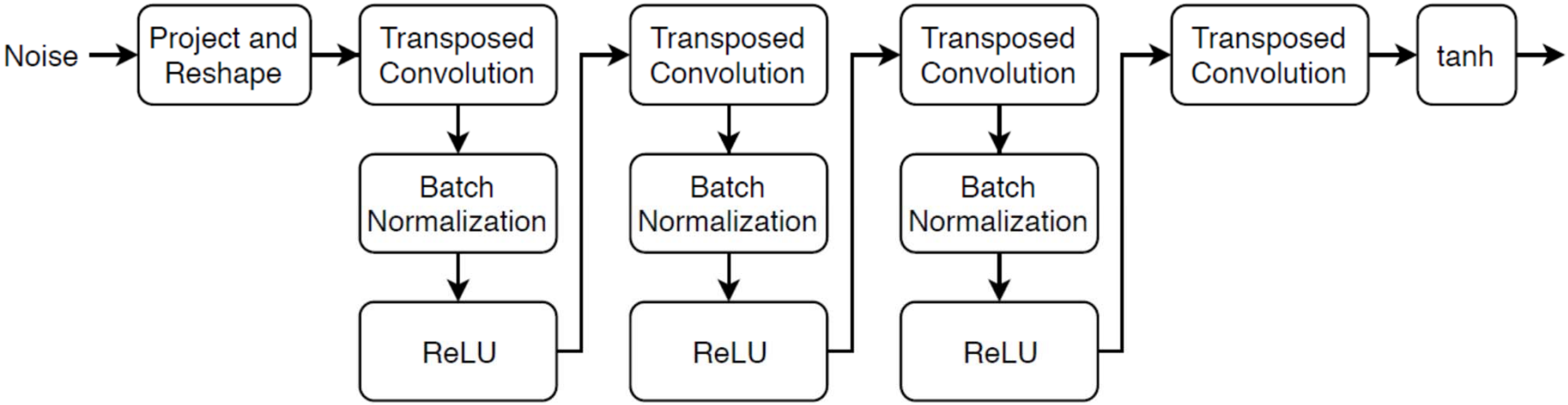
Generative AI is based on models trained with existing data, and then these trained models can generate new data that has similar characteristics but is not identical to the original data. They are used mostly for natural language processing.

The models are based on CNN's and LSTM's and biLSTM's. Generative Adversial Neural Netwok (GAN), composed by a generator and a discriminator are actually in great development.



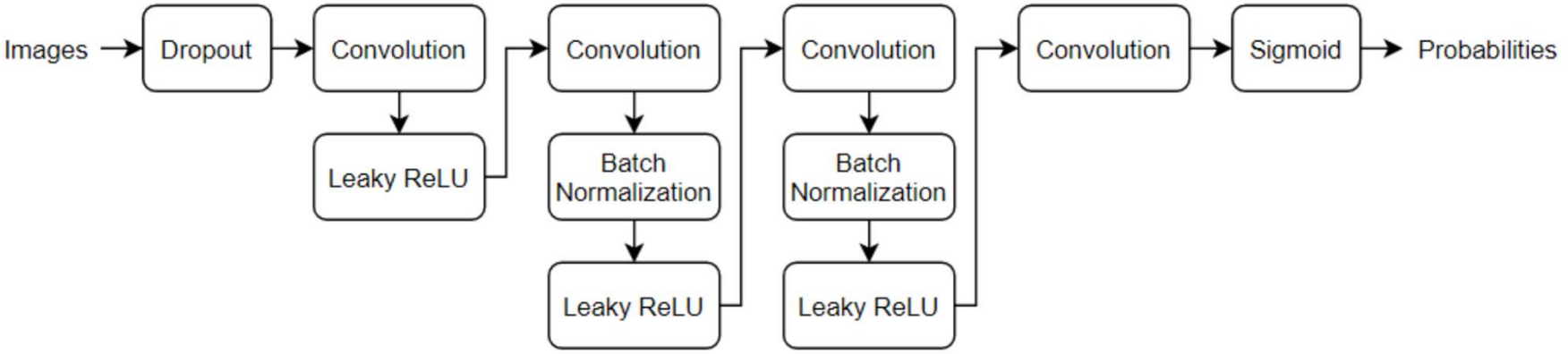
General scheme of a GAN, from nnug 2023b, pag. 3-74.

Generator



See nnug, 3-74

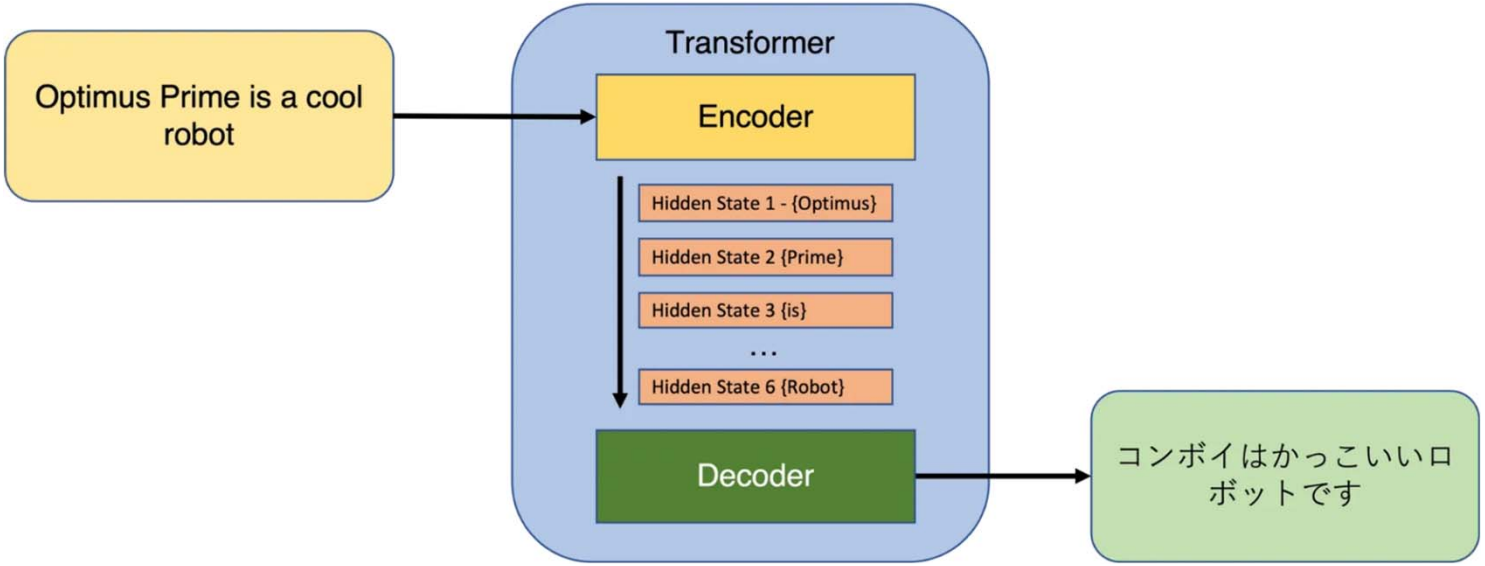
Discriminator



Transformers

Transformers are special artificial neural network architectures used to solve the problem of transduction or transformation of input sequences into output sequences in deep learning applications, namely in natural language processing, speech recognition, protein structure prediction, machine translation. They are based on encoders and decoders. They may have mechanisms of attention to better consider the context (past and future) of the words in a sentence. See more in

https://domino.ai/blog/transformers-self-attention-to-the-rescue 16 Oct 2023



Read the note “GenAI.pdf” generated by bing (ChatGTP3.5), and nnug 2023b, pages 3-72 to 3-79, with programming examples.

ChatGPT (Generative Pre-trained Transformer)

Version	**Training set**	**Number of parameters**	**Year**
GTP 1	84 Million webpages	117 Million	2017
GTP 2	40 Giga bytes of text	1,5 Billion	2019
GTP 3.5	570 Giga bytes of text, 300 Billion words	175 Billion	2020
GTP 4	??	Until 100 Trillion parameters	2023

GPT Model	Size (Parameters)	Release Date	Applications
GPT	1.5 billion	June 2017	Text generation, language translation, language modeling, text summarization
GPT-2	1.5 billion	February 2019	Text generation, language translation, language modeling, text summarization
GPT-3	175 billion	June 2020	Text generation, language translation, language modeling, text summarization, question answering, chatbots, automated content generation
CHAT-GPT	175 billion	June 2020	Chatbots, conversation generation
GPT-4	175 billion Can reach 100 Trillion	Not publicly released	Text generation, language translation, language modeling, text summarization, question answering, chatbots, automated content generation, customer service, education

https://indianexpress.com/article/technology/tech-news-technology/chatgpt-4-release-features-specifications-parameters-8344149 16 October 2023

Generative AI needs a set of ethical and societal rules to be useful to society.

From https://www.zdnet.com/article/the-5-biggest-risks-of-generative-ai-according-to-an-expert/ 17/10/2023:

1. Hallucinations

Hallucinations refer to the errors that AI models are prone to make because, although they are advanced, they are still not human and rely on training and data to provide answers.

If you've used an AI chatbot, then you have probably experienced these hallucinations through a misunderstanding of your prompt or a blatantly wrong answer to your question.

Also: ChatGPT's intelligence is zero, but it's a revolution in usefulness, says AI expert

Litan (https://www.gartner.com/en/newsroom/press-releases/2023-04-20-why-trust-and-security-are-essential-for-the-future-of-generative-ai). says the training data can lead to biased or factually incorrect responses, which can be a serious problem when people are relying on these bots for information.

"Training data can lead to biased, off-base or wrong responses, but these can be difficult to spot, particularly as solutions are increasingly believable and relied upon," says Litan

2. Deepfakes

A deepfake uses generative AI to create videos, photos, and voice recordings that are fake but take the image and likeness of another individual.
Perfect examples are the AI-generated viral photo of Pope Francis in a puffer jacket or the AI-generated Drake and the Weeknd song, which garnered hundreds of thousands of streams.
"These fake images, videos and voice recordings have been used to attack celebrities and politicians, to create and spread misleading information, and even to create fake accounts or take over and break into existing legitimate accounts," says Litan.
Also: How to spot a deepfake? One simple trick is all you need
Like hallucinations, deepfakes can contribute to the massive spread of fake content, leading to the spread of misinformation, which is a serious societal problem.

3. Data privacy

Privacy is also a major concern with generative AI since user data is often stored for model training. This concern was the overarching factor that pushed Italy to ban ChatGPT, claiming OpenAI was not legally authorized to gather user data.

"Employees can easily expose sensitive and proprietary enterprise data when interacting with generative AI chatbot solutions," says Litan. "These applications may indefinitely store information captured through user inputs, and even use information to train other models -- further compromising confidentiality."

Also: AI may compromise our personal information

Litan highlights that, in addition to compromising user confidentiality, the stored information also poses the risk of "falling into the wrong hands" in an instance of a security breach.

4. Cybersecurity

The advanced capabilities of generative AI models, such as coding, can also fall into the wrong hands, causing cybersecurity concerns.
"In addition to more advanced social engineering and phishing threats, attackers could use these tools for easier malicious code generation," says Litan.
Also: The next big threat to AI might already be lurking on the web
Litan says even though vendors who offer generative AI solutions typically assure customers that their models are trained to reject malicious cybersecurity requests, these suppliers don't equip end users with the ability to verify all the security measures that have been implemented.

5. Copyright issues

Copyright is a big concern because generative AI models are trained on massive amounts of internet data that is used to generate an output.
This process of training means that works that have not been explicitly shared by the original source can then be used to generate new content.
Copyright is a particularly thorny issue for AI-generated art of any form, including photos and music.
Also: How to use Midjourney to generate amazing images
To create an image from a prompt, AI-generating tools, such as DALL-E, will refer back to the large database of photos they were trained on. The result of this process is that the final product might include aspects of an artist's work or style that are not attributed to them.
Since the exact works that generative AI models are trained on are not explicitly disclosed, it is hard to mitigate these copyright issues.

Read also Generative AI: Advantages, Disadvantages, Limitations, and Challenges (fact.technology) 16 oct 2023

5.6 Conclusions

When information (data) is changing, being it from dynamical systems of from drifts in data streams, nnets models must be recursive, with memory. This can be obtained by shallow nets or by deep LSTM nets, depending on the complexity, the dimension and the quantity of data available.

CNNs have been developed for image classification, so the standard layers used on them have been conceived mainly for operations on images.
Many other problems can be reduced to image classification (for example a multidimensional time-series), but for more specific problems the great challenge of deep learning is to build appropriate layers adequate for new problems. Note that the CNN is trained by the backpropagation algorithm studied in Chapter 4 and also used for shallow nets.

Transfer learning can help in many situations, if cautiously made.

More advanced architectures, like Generative NNs and Transformers, allow to build powerful and useful applications, namely in natural language and image processing .

However ethical and societal rules should regulate the use of these (and future) techniques of Artificial Intelligence.

The **AI Safety Summit 2023**, November 1-2:

https://www.gov.uk/government/publications/ai-safety-summit-2023-the-bletchley-declaration 1 November 2023

The Bletchley declaration signed by 28 countries and the EU (https://www.reuters.com/technology/britain-brings-together-political-tech-leaders-talk-ai-2023-11-01/ , 1 Nov 2023)

Bibliography

Deep learning: see review papers in the course materials

Deep Learning Toolbox Users´s Guide, The Mathworks, 2023b.

Deep Learning Book, in Portuguese (Basil):

https://www.deeplearningbook.com.br/ 3/10/2023

https://keras.io/layers/convolutional/ 3/10/2023

https://cs231n.github.io/convolutional-networks/ 3/10/2023 (with animation)

Deep Learning, I. Goodfellow, Y. Bengio, A. Courville, MIT Press, 2016 (https://www.deeplearningbook.org 17/10/2023).

Links in the slides, active in 17/10/2023.