Temporal Convolutional Networks

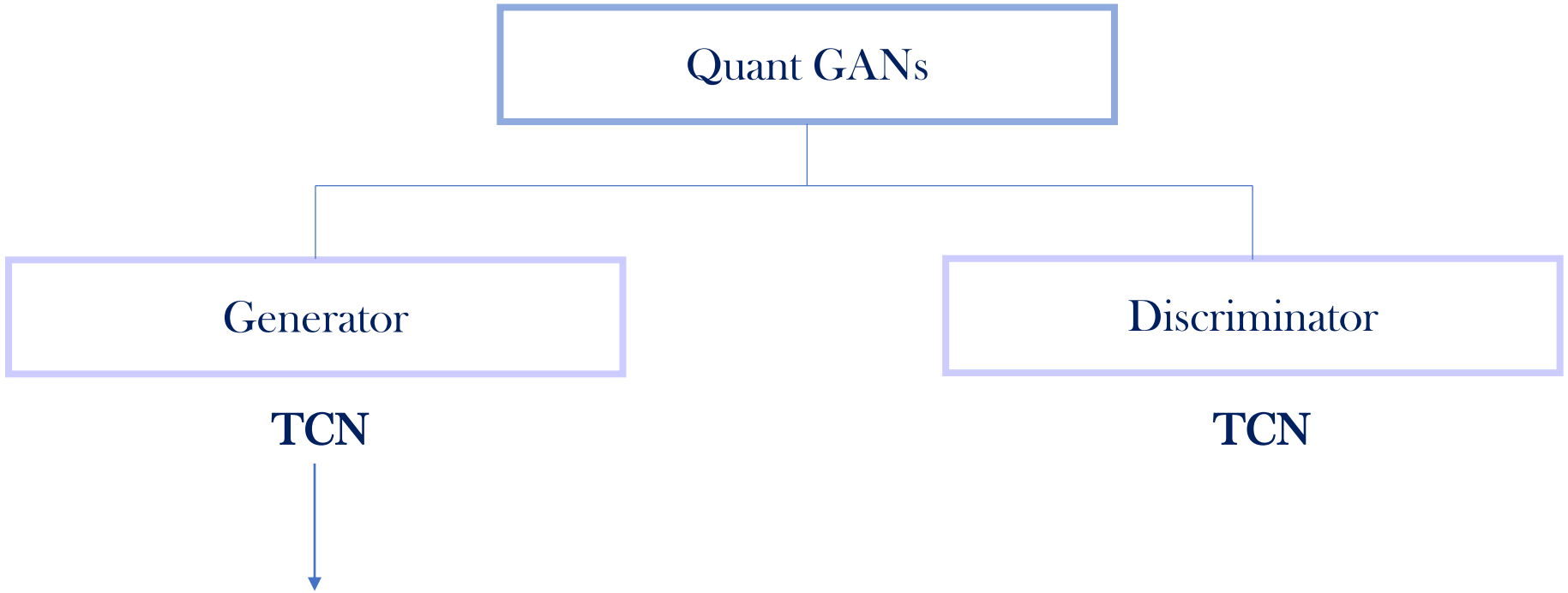
Hyelin Choi

Department of Mathematics

Sungkyunkwan University

April 15, 2024

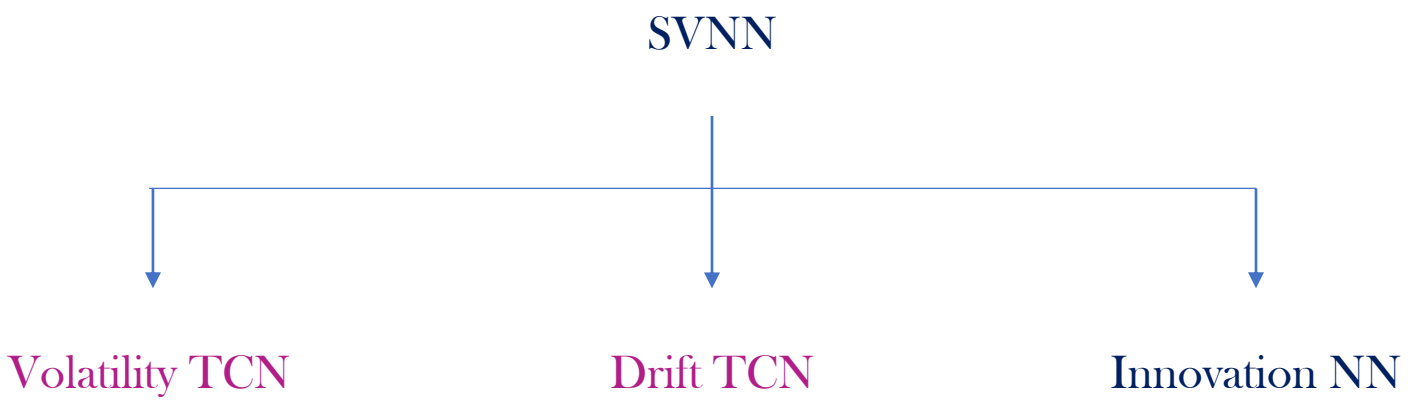
Temporal Convolutional Networks



Generate financial time series

(Log-return Neural Process)

Temporal Convolutional Networks



Temporal Convolutional Networks

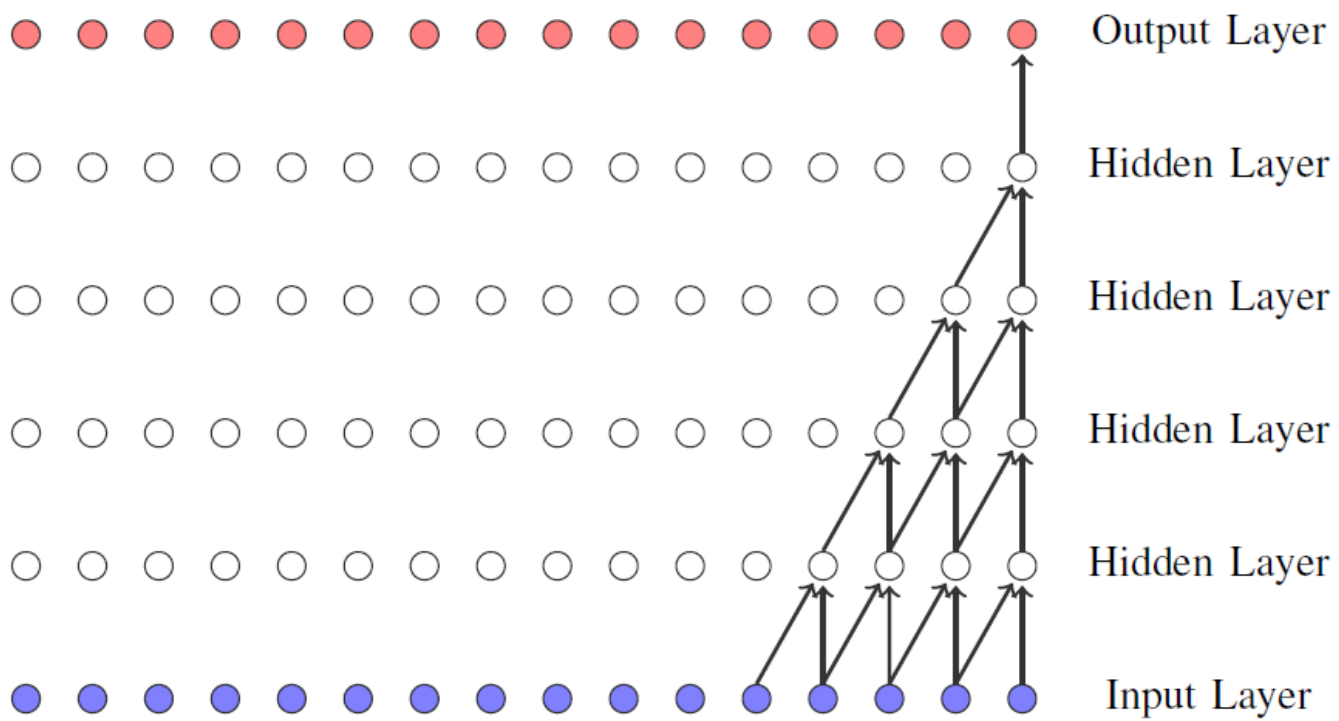
Sequence-related modeling task  financial time series

Constructions

Dilated causal convolutions = Causal convolutions + Dilated convolutions

Temporal Convolutional Networks

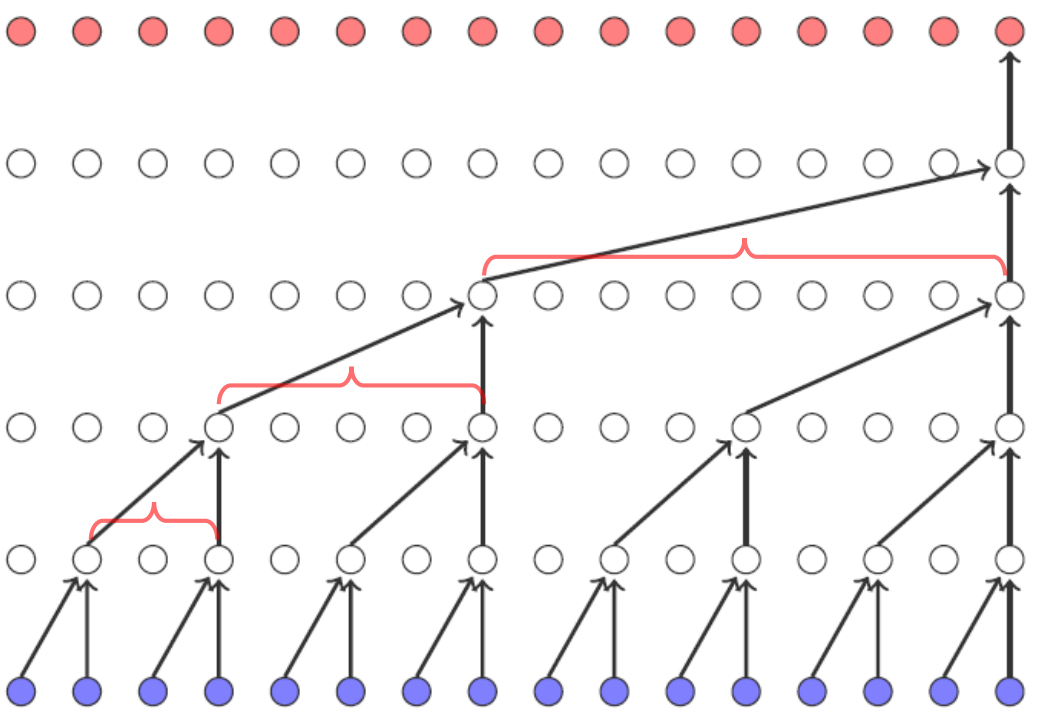
Causal convolutions are convolutions, where output only depends on past sequence elements.



Temporal Convolutional Networks

Dilated convolutions are convolutions 'with holes'.

- Dilation factor D
- Kernel size K=2



$D_3 = 8 = 2^3$

$D_2 = 4 = 2^2$

$D_1 = 2 = 2^1$

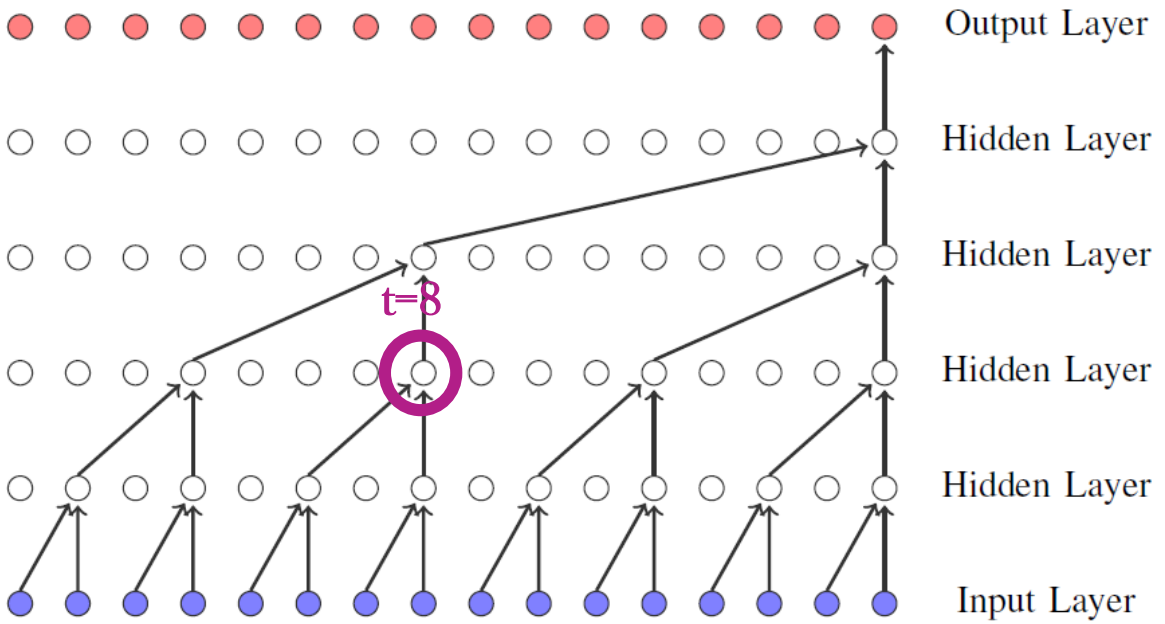
Temporal Convolutional Networks

Definition 3.5 ($*_D$ operator)**.** Let $X \in \mathbb{R}^{N_I \times T}$ be an N_I-variate sequence of length T and $W \in \mathbb{R}^{K \times N_I \times N_O}$ a tensor. Then for $t \in \{D(K-1)+1, \ldots, T\}$ and $m \in \{1 \ldots, N_O\}$ the operator $*_D$, defined by

$$(W *_D X)_{m,t} := \sum_{i=1}^{K} \sum_{j=1}^{N_I} W_{i,j,m} \cdot X_{j,t-D(K-i)} \ ,$$

is called *dilated causal convolutional operator* with *dilation* D and kernel size K.

Temporal Convolutional Networks



D=2

K=2

t=8

$$(W *_D X)_{m,8} = \sum_{i=1}^{2} \sum_{j=1}^{N_I} W_{i,j,m} \cdot X_{j,8-2(2-i)}$$

$\longrightarrow X_{j,6}, X_{j,8}$

$$(W *_D X)_{m,t} := \sum_{i=1}^{K} \sum_{j=1}^{N_I} W_{i,j,m} \cdot X_{j,t-D(K-i)} \ ,$$

Temporal Convolutional Networks

Definition 3.6 (Causal convolutional layer)**.** Let W be as in Definition 3.5 and $b \in \mathbb{R}^{N_O}$. A function

$$w : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times (T - D(K-1))}$$

defined for $t \in \{D(K-1)+1, \ldots, T\}$ and $m \in \{1, ..., N_O\}$ by

$$w(X)_{m,t} := (W *_D X)_{m,t} + b_m$$

is called *causal convolutional layer with dilation* D.

Temporal Convolutional Networks

Remark 3.7. The quadruple (N_I, N_O, K, D) will be called the *arguments* of a causal convolution w and represent the *input dimension*, *output dimension*, *kernel size*, and *dilation*, respectively.

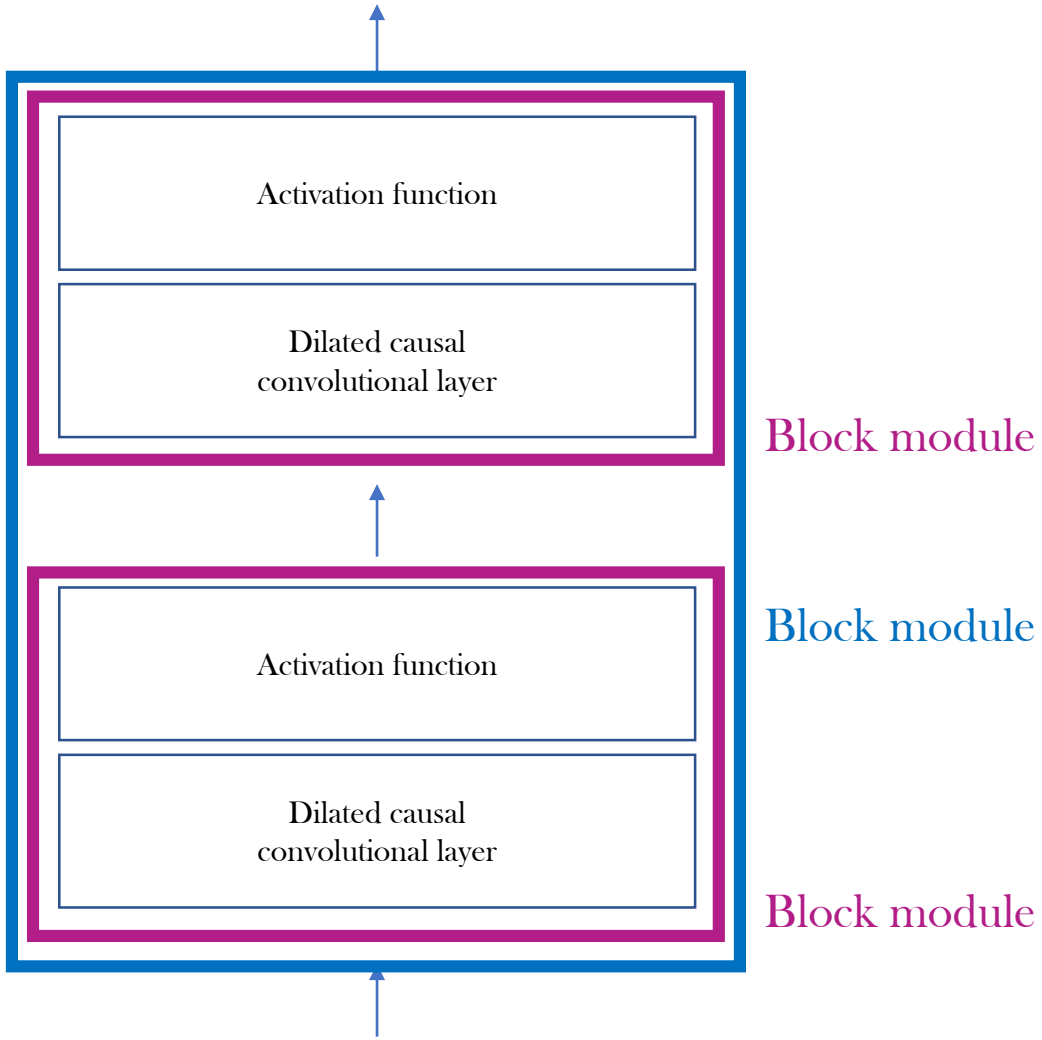
Temporal Convolutional Networks

Dilated causal convolutional layers + Activation functions

generalization

Block module

Temporal Convolutional Networks



Temporal Convolutional Networks

Definition 3.9 (Block module)**.** Let $S \in \mathbb{N}$. A function $\psi : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times (T-S)}$ that is Lipschitz continuous is called *block module* with arguments (N_I, N_O, S).

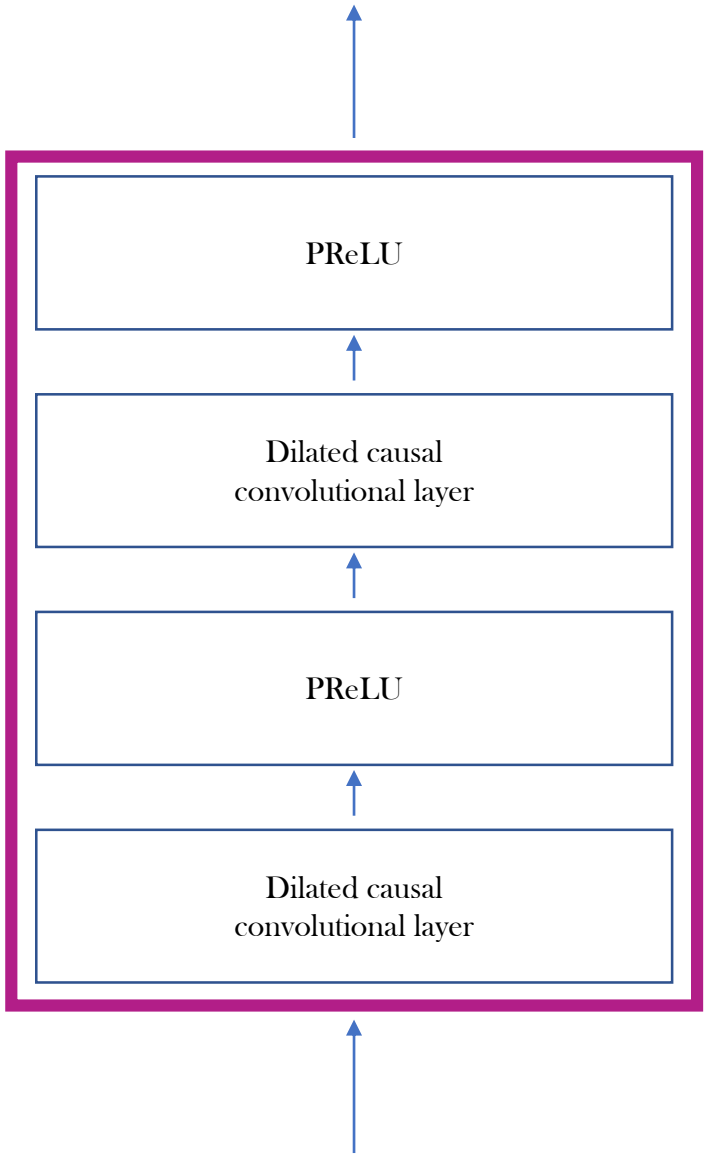
Temporal Convolutional Networks

Definition B.1 (Temporal block)**.** Let $N_I, N_H, N_O \in \mathbb{N}$ denote the input, hidden and output dimension and let $D, K \in \mathbb{N}$ denote the dilation and the kernel size. Furthermore, let w_1, w_2 be two dilated causal convolutional layers with arguments (N_I, N_H, K, D) and (N_H, N_O, K, D) respectively and let $\phi_1, \phi_2 : \mathbb{R} \to \mathbb{R}$ be two PReLUs. The function $f : \mathbb{R}^{N_I \times (2D(K-1)+1)} \to \mathbb{R}^{N_O}$ defined by

$$f(X) = \phi_2 \circ w_2 \circ \phi_1 \circ w_1(X)$$

is called *temporal block* with arguments (N_I, N_H, N_O, K, D).

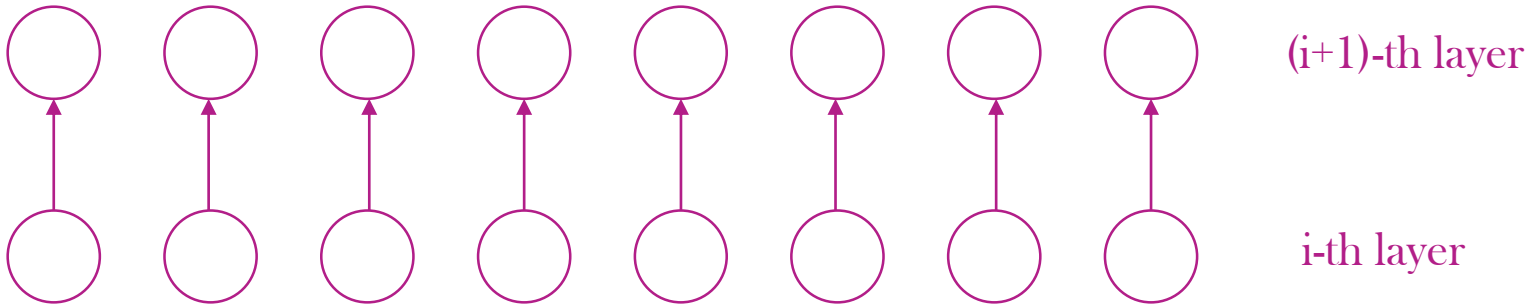
Temporal Convolutional Networks



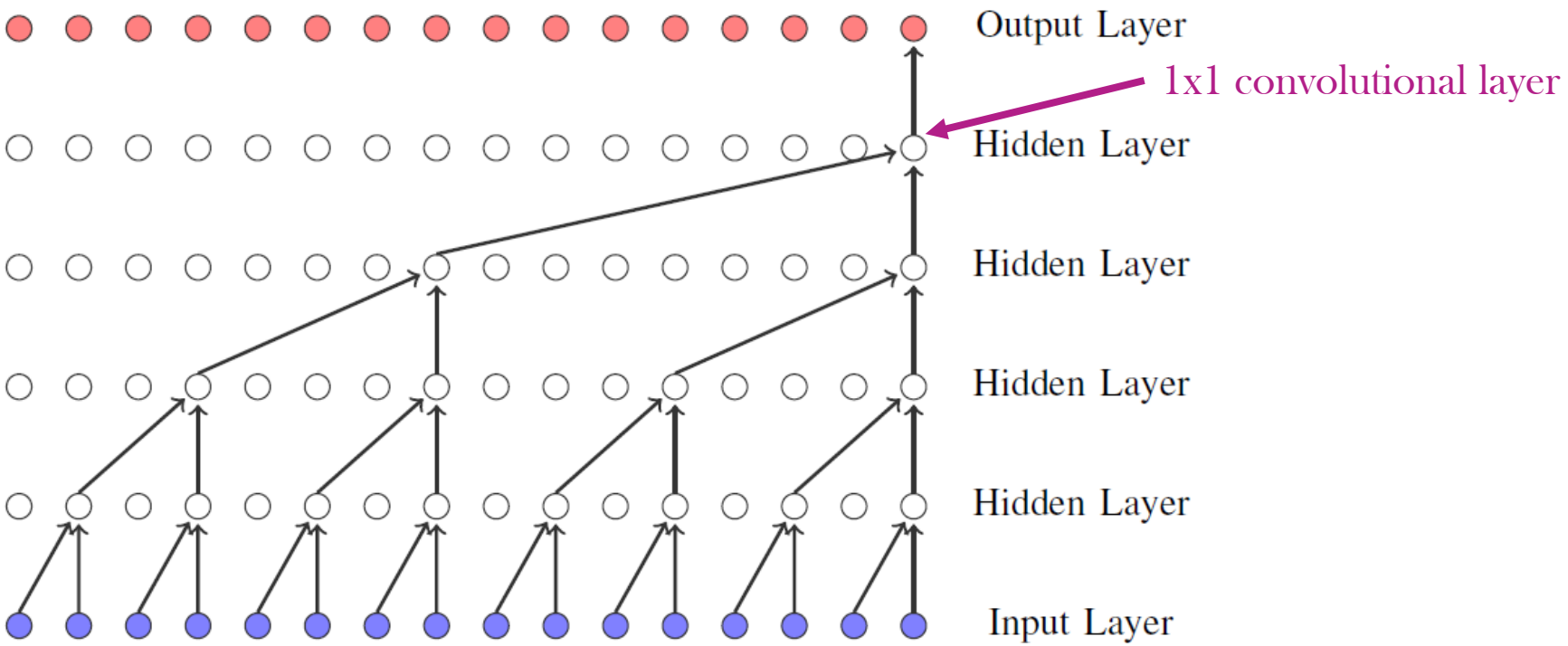
Block module

Temporal Convolutional Networks

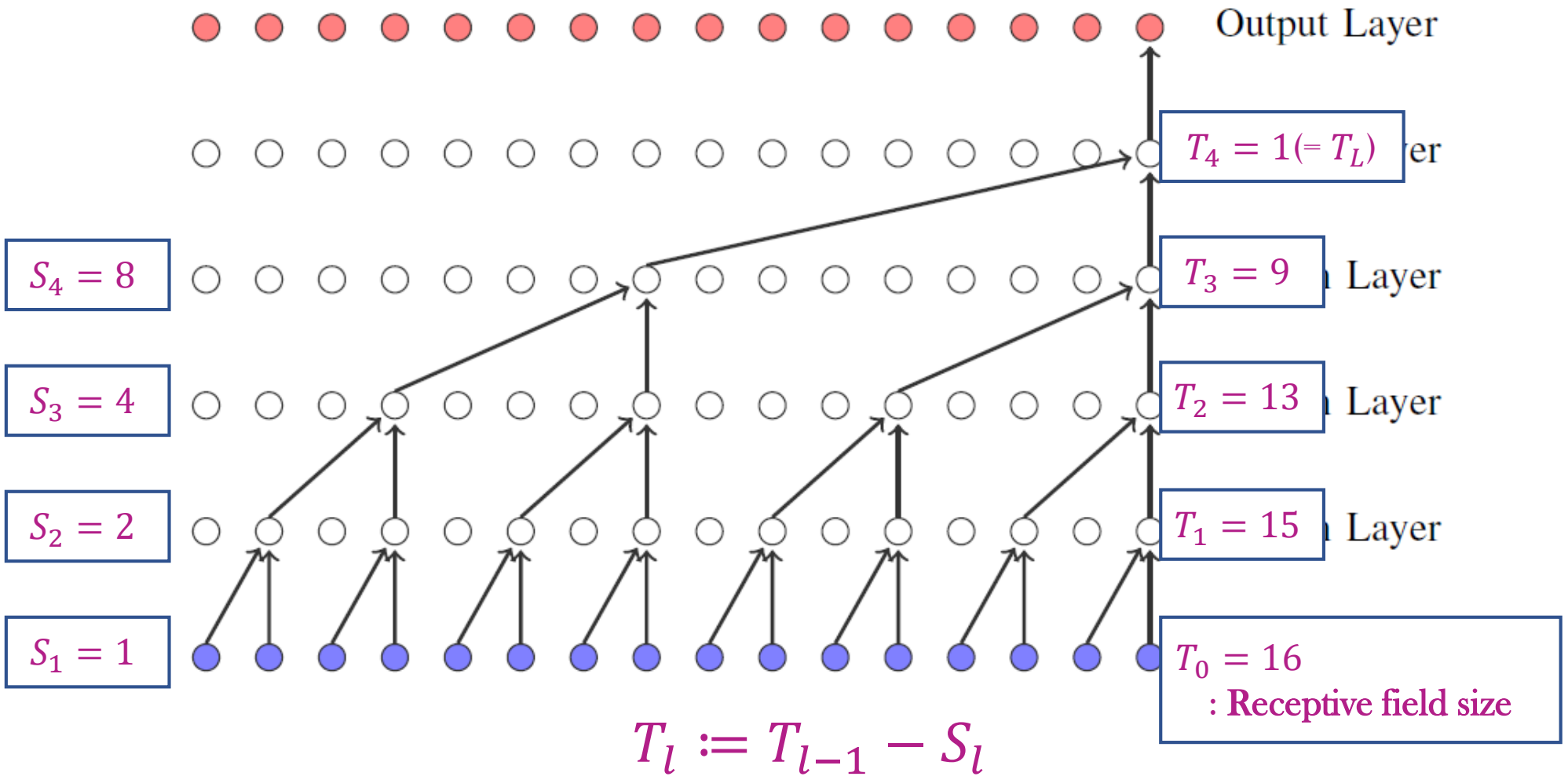
Example 3.8 (1×1 convolutional layer)**.** Let $X \in \mathbb{R}^{N_I \times T}$ be an N_I-variate sequence and $w : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times T}$ a causal convolutional layer with arguments $(N_I, N_O, 1, 1)$. We call such a layer a 1×1 *convolutional layer*.[2]



Temporal Convolutional Networks



Temporal Convolutional Networks



Temporal Convolutional Networks

Definition 3.10 (Temporal convolutional network)**.** Let $T_0, L, N_0, \ldots, N_{L+1} \in \mathbb{N}$. Moreover, for $l \in \{1, \ldots, L\}$ let $S_l \in \mathbb{N}$ such that $\sum_{l=1}^{L} S_l \leq T_0 - 1$. Hence, for $T_l := T_{l-1} - S_l$ it holds

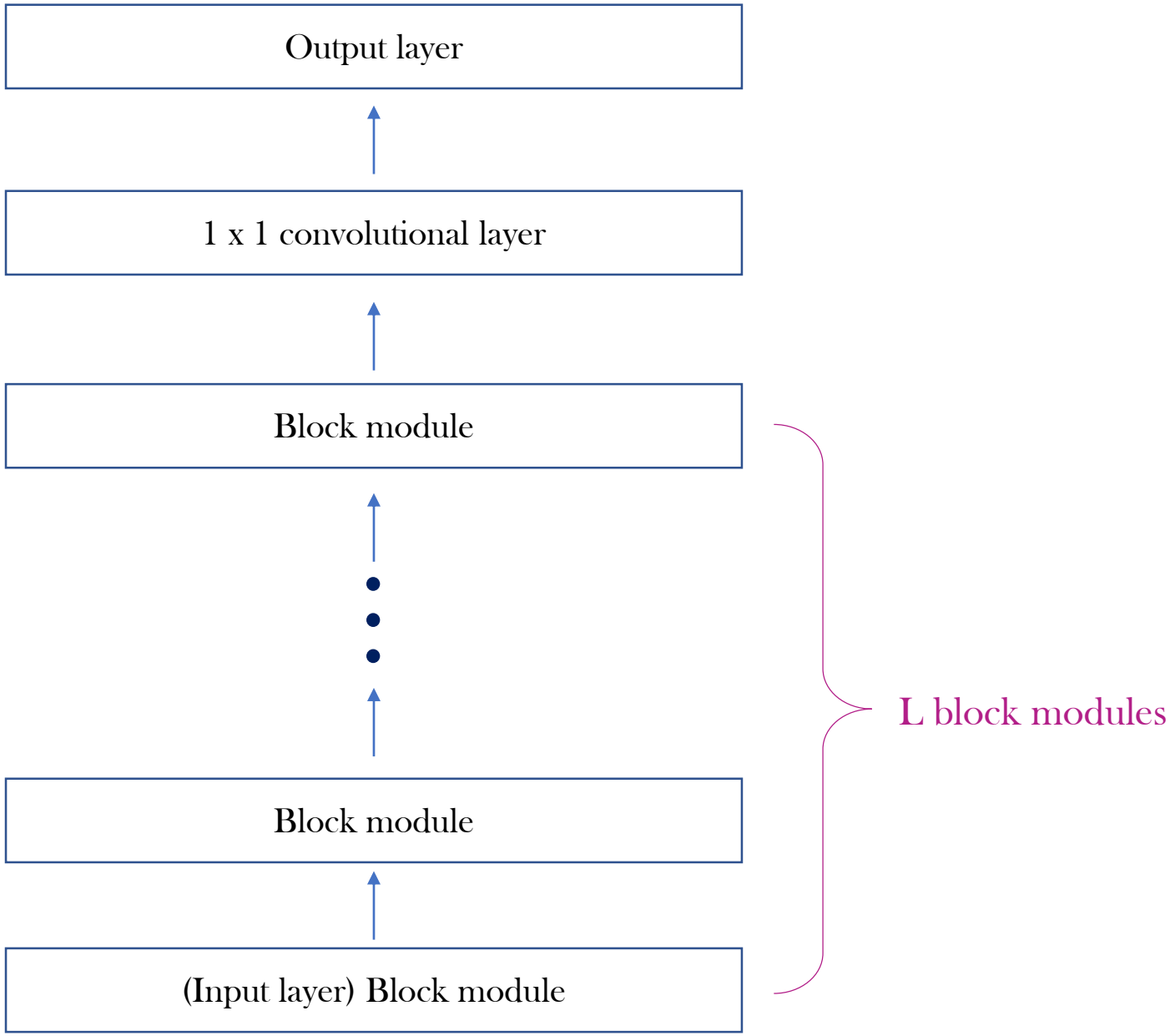
$$T_L = T_0 - \sum_{l=1}^{L} S_l \geq 1 \,. \tag{1}$$

Furthermore, let $\psi_l : \mathbb{R}^{N_{l-1} \times T_{l-1}} \to \mathbb{R}^{N_l \times T_l}$ for $l \in \{1, \ldots, L\}$ represent block modules and $w : \mathbb{R}^{N_L \times T_L} \to \mathbb{R}^{N_{L+1} \times T_L}$ a 1×1 convolutional layer. A function $f : \mathbb{R}^{N_0 \times T_0} \times \Theta \to \mathbb{R}^{N_{L+1} \times T_L}$, defined by

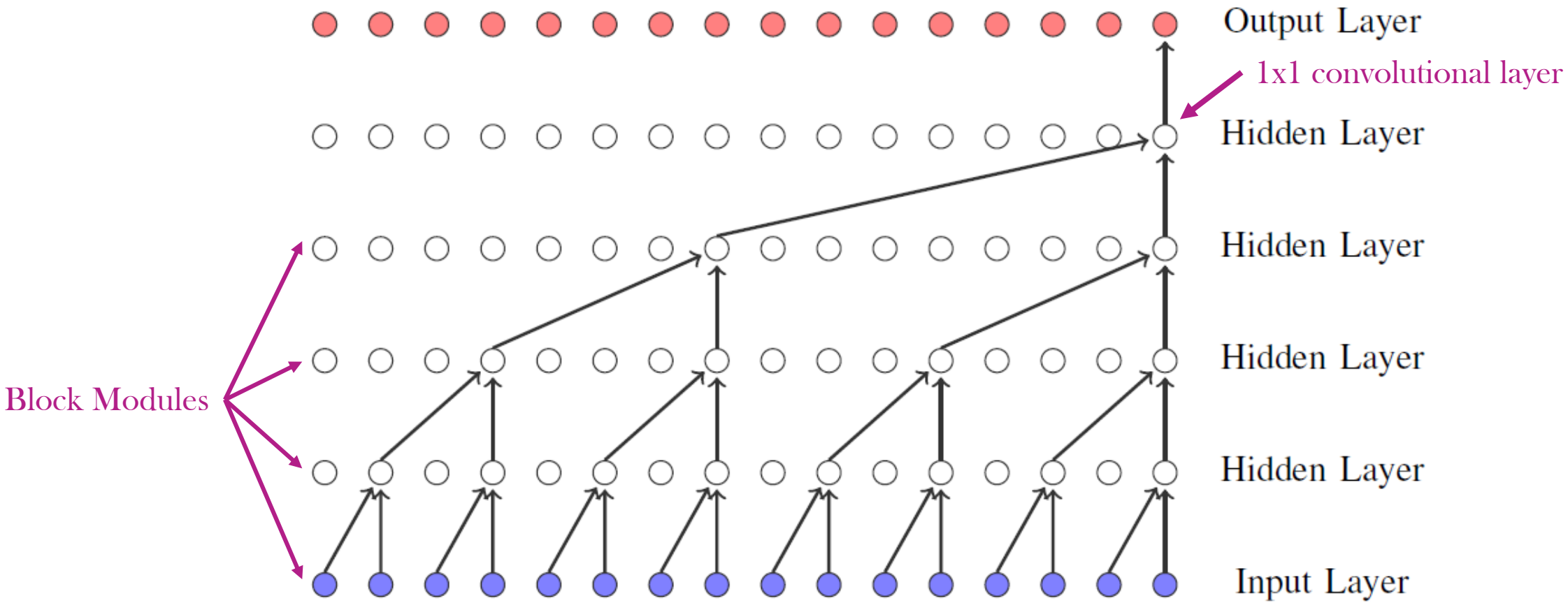
$$f(X, \theta) = w \circ \psi_L \circ \cdots \circ \psi_1(X) \,,$$

is called *temporal convolutional network* with L hidden layers. The class of TCNs with L hidden layers mapping from $\mathbb{R}^{d_0}$ to $\mathbb{R}^{d_1}$ will be denoted by $\mathrm{TCN}_{d_0, d_1, L}$ ($d_0 = N_0, d_1 = N_{L+1}$).

Temporal Convolutional Network



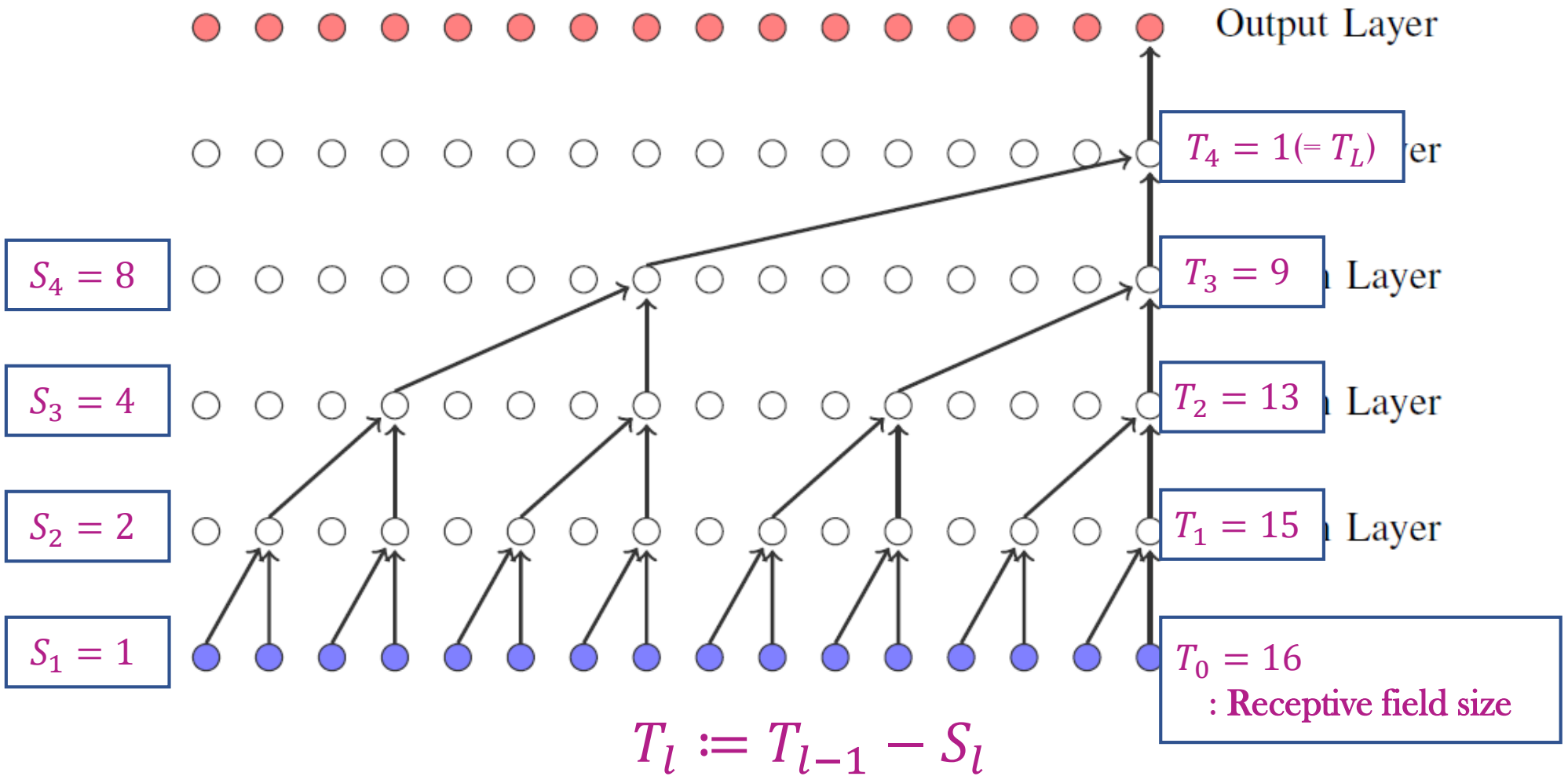
Temporal Convolutional Networks



Temporal Convolutional Networks

Definition 3.11 (Vanilla TCN)**.** Let $f \in \mathrm{TCN}_{N_0, N_{L+1}, L}$ such that for all $l \in \{1, \ldots, L\}$ each block module ψ_l is defined as a composition of a causal convolutional layer w_l with arguments (N_{l-1}, N_l, K_l, D_l) and an activation function ϕ, i.e. $\psi_l = \phi \circ w_l$. Then we call $f : \mathbb{R}^{N_0 \times T_0} \times \Theta \to \mathbb{R}^{N_{L+1} \times T_L}$ a *Vanilla TCN*. Moreover, if $D_l = D^{l-1}$ for all $l \in \{1, \ldots, L\}$, we call f a *Vanilla TCN with dilation factor* D. Whenever $K_l = K$ for all $l \in \{1, \ldots, L\}$, we say that f has *kernel size* K.

Temporal Convolutional Networks



Temporal Convolutional Networks

Definition 3.12 (Receptive field size). Let $f \in \mathrm{TCN}_{d_0,d_1,L}$ and let $S_1, \ldots, S_L$ be as in Definition 3.10. The constant

$$T^{(f)} := 1 + \sum_{l=1}^{L} S_l$$

is called *receptive field size* (*RFS*).

Receptive field size is the number of sequence elements that the TCN can capture.

Temporal Convolutional Networks

Skip Connections

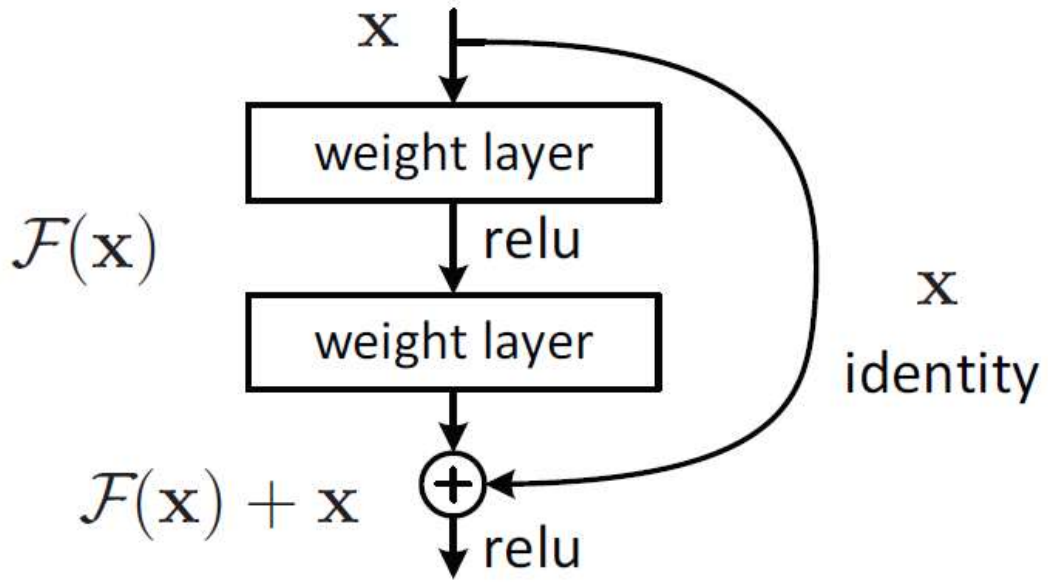


Figure 2. Residual learning: a building block.

Temporal Convolutional Networks

Definition 3.15 (TCN with skip connections)**.** Assume the notation from Definition 3.10 and for $N_{skip} \in \mathbb{N}$ let

$$\gamma_l : \mathbb{R}^{N_{l-l} \times T_{l-1}} \to \mathbb{R}^{N_l \times T_l} \times \mathbb{R}^{N_{skip} \times T_L} \quad \text{for } l \in \{1, \dots, L\}$$

denote block modules. Moreover, let γ be a block module with arguments $(N_{skip}, N_{L+1}, 0)$. If the output $Y \in \mathbb{R}^{N_{L+1} \times T_L}$ of a TCN $f : \mathbb{R}^{N_0 \times T_0} \times \Theta \to \mathbb{R}^{N_{L+1} \times T_L}$ is defined recursively by

$$\left(X^{(l)}, H^{(l)}\right) = \gamma_l \left(X^{(l-1)}\right) \quad \text{for } l \in \{1, \dots, L\}$$

$$Y = \gamma \left(\sum_{l=1}^{L} H^{(l)} \right) ,$$

where $X^{(0)} \in \mathbb{R}^{N_0 \times T_0}$, then f is called a *temporal convolutional network with skip connections.*

Temporal Convolutional Networks

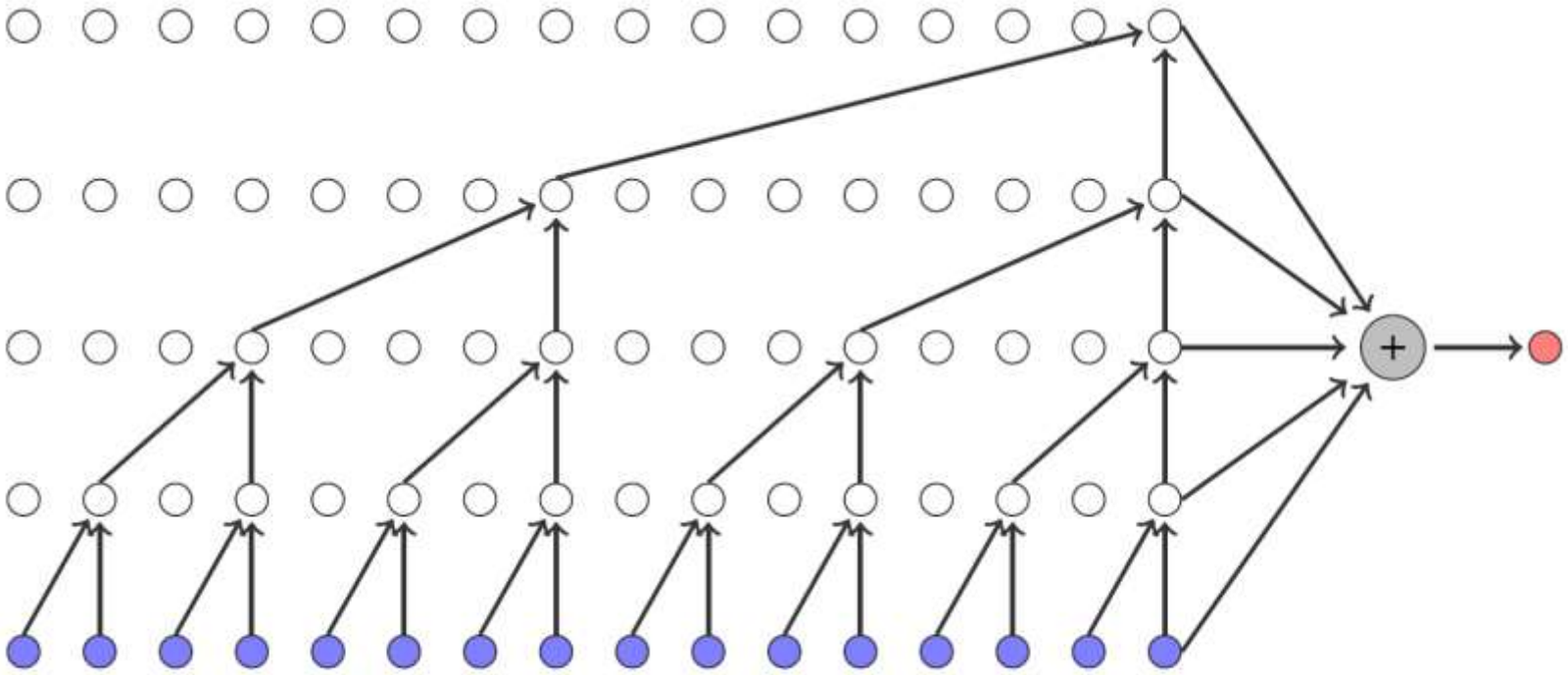
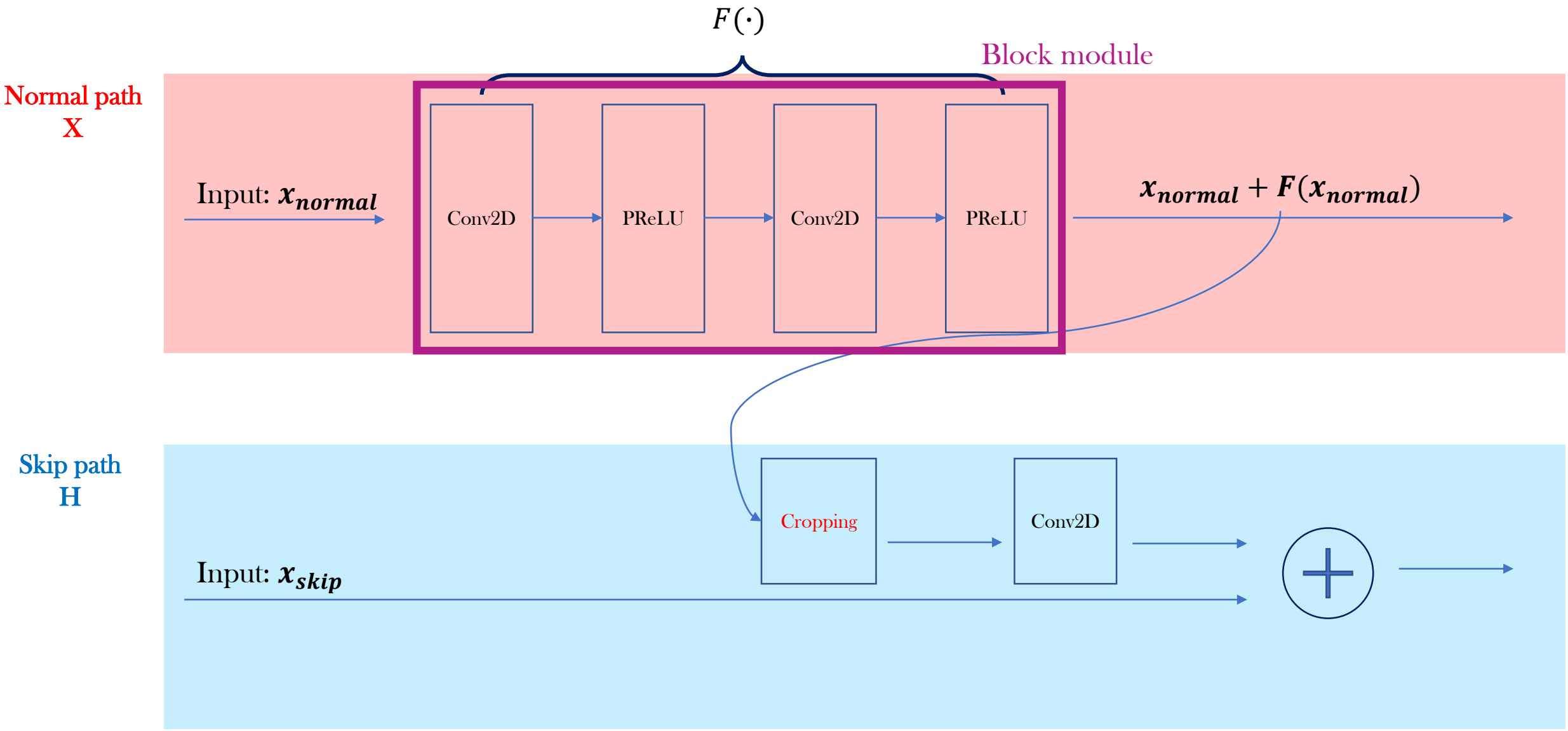


Figure 7: Vanilla TCN with skip connections.

Temporal Convolutional Networks

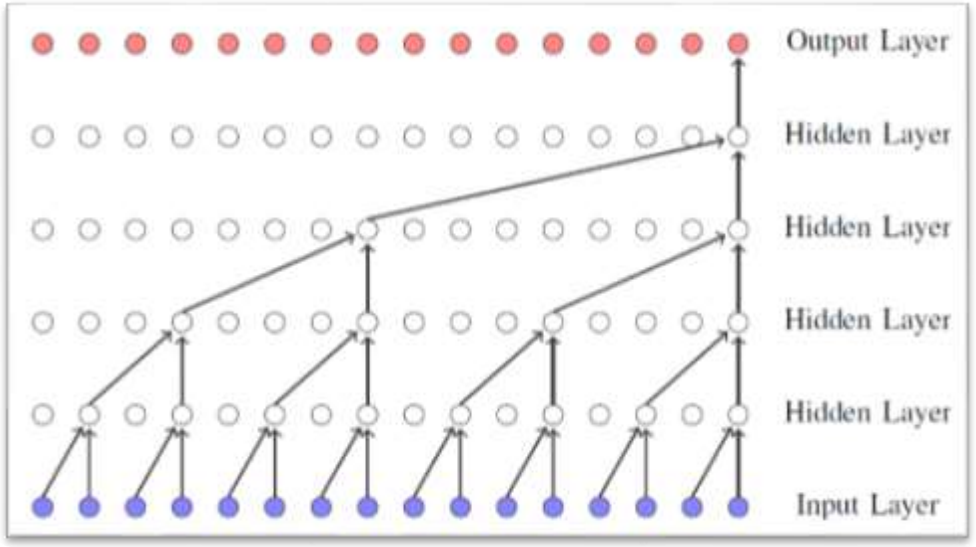


```python
def add_temporal_block(prev_layer, skip_layer, kernel_size, dilation, fixed_filters, moving_filters, n_series, rfs, block_size, use_batchNorm, cropping):
    """Creates a temporal block.
    Args:
        prev_layer (tensorflow.keras.layers.Layer): previous layer to attach to on standard path.
        skip_layer (tensorflow.keras.layers.Layer): previous layer to attach to on the skip path. Use None for intiation.
        kernel_size (int): kernel size along temporal axis of convolution layers within the temporal block.
        dilation (int): dilation of convolution layers along temporal axis within the temporal block.
        n_filters (int): Number of channels to use within the temporal block.
        n_series (int): Number of timeseries to model.
        rfs (int): Receptive field size of the network.
        block_size (int): Number of convolution layers to use within a temporal block.
        use_batchNorm (bool): Whether to use batch normalization (with renormalization).
    Returns:
        tuple of tensorflow.keras.layers.Layer: Output layers belonging to (normal path, skip path).
    """
    # Identity mapping so that we hold a valid reference to prev_layer
    block = Lambda(lambda x: x)(prev_layer)

    for _ in range(block_size):
        convs = []
        for _ in range(n_series):
            prev_block= Lambda(lambda x: x)(block)
            convs.append(SpectralNormalization(Conv2D(fixed_filters, (n_series, kernel_size), dilation_rate=(1, dilation)))(block))

        if len(convs) > 1:
            block = Concatenate(axis=1)(convs)
        else:
            block = convs[0]
        if moving_filters:
            block = Concatenate(axis=-1)([block, Conv2D(moving_filters, (1, kernel_size), dilation_rate=(1, dilation))(prev_block)])
        if use_batchNorm:
            block = BatchNormalization(axis=3, momentum=.9, epsilon=1e-5, renorm=True, renorm_momentum=.9)(block)

        block = PReLU(shared_axes=[2, 3])(block)

    # As layer output gets smaller, we need to crop less before putting output
    # on the skip path. We cannot infer this directly as tensor shapes may be variable.
    drop_left = block_size * (kernel_size - 1) * dilation
    cropping += drop_left

    if skip_layer is None:
        prev_layer = Conv2D(fixed_filters + moving_filters, 1)(prev_layer)
    # add residual connections
    out = Add()([Cropping2D(cropping=((0,0), (drop_left, 0)))(prev_layer), block])
    # crop from left side for skip path
    skip_out = Cropping2D(cropping=((0,0), (rfs-1-cropping, 0)))(out)
    # add current output with 1x1 conv to skip path
    if skip_layer is not None:
        skip_out = Add()([skip_layer, SpectralNormalization(Conv2D(fixed_filters + moving_filters, 1))(skip_out)])
    else:
        skip_out = SpectralNormalization(Conv2D(fixed_filters + moving_filters, 1))(skip_out)

    return PReLU(shared_axes=[2, 3])(out), skip_out, cropping
```

```python
def make_TCN(dilations, fixed_filters, moving_filters, use_batchNorm, one_series_output, sigmoid, input_dim, block_size=2):
    """Creates a causal temporal convolutional network with skip connections.
       This network uses 2D convolutions in order to model multiple timeseries co-dependency.
    Args:
        dilations (list, tuple): Ordered number of dilations to use for the temporal blocks.
        fixed_filters (int): Number of channels in the hidden layers corresponding fixed over series axis.
        moving_filters (int): Number of channels in the hidden layers moving over series axis.
        use_batchNorm (bool): Whether to use batch normalization in the temporal blocks. Includes batch Renormalization.
        one_series_output (bool): Whether to collapse the dimension of the series axis to 1 using an additional convolution layer.
        sigmoid (bool): Whether to append the sigmoid activation function at the output of the network.
        input_dim (list, tuple): Input dimension of the shape (number of timeseries, timesteps, number of features). Timesteps may be None for variable length timeseries.
        block_size (int): How many convolution layers to use within a temporal block. Defaults to 2.
    Returns:
        tensorflow.keras.models.Model: a non-compiled keras model.
    """
    rfs = receptive_field_size(dilations, block_size)
    n_series = input_dim[0]

    input_layer = Input(shape=input_dim)
    cropping = 0
    prev_layer, skip_layer, _ = add_temporal_block(input_layer, None, 1, 1, fixed_filters, moving_filters, n_series, rfs, block_size, use_batchNorm, cropping)

    for dilation in dilations:
        prev_layer, skip_layer, cropping = add_temporal_block(prev_layer, skip_layer, 2, dilation, fixed_filters, moving_filters, n_series, rfs, block_size, use_batchNorm, cropping)

    output_layer = PReLU(shared_axes=[2, 3])(skip_layer)
    output_layer = SpectralNormalization(Conv2D(fixed_filters + moving_filters, kernel_size=1))(output_layer)
    output_layer = PReLU(shared_axes=[2, 3])(output_layer)
    output_layer = SpectralNormalization(Conv2D(1, kernel_size=1))(output_layer)

    if one_series_output:
        output_layer = SpectralNormalization(Conv2D(1, (n_series, 1)))(output_layer)

    if sigmoid:
        output_layer = Activation('sigmoid')(output_layer)

    return Model(input_layer, output_layer)
```

```python
def receptive_field_size(dilations, block_size):
    """Non-exhaustive computation of receptive field size.
    Args:
        dilations (list, tuple): Ordered number of dilations of the network.
        block_size (int): Number of convolution layers in each temporal block of the network.
    Returns:
        int: the receptive field size.
    """
    return 1 + block_size * sum(dilations)
```

Temporal Convolutional Networks



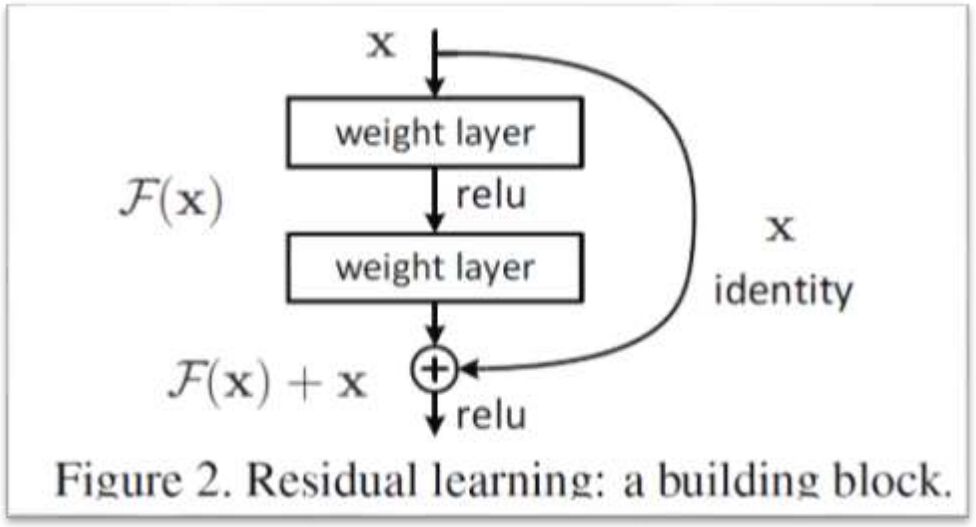


Figure 2. Residual learning: a building block.

- Advantages of TCN
  - ACF score
  - Leverage effect score

1) TCNs are able to capture long-range dependencies in sequences.

2) TCNs have an advantage of avoiding exponentially vanishing and exploding gradients.