Classification and Regression

Part 2

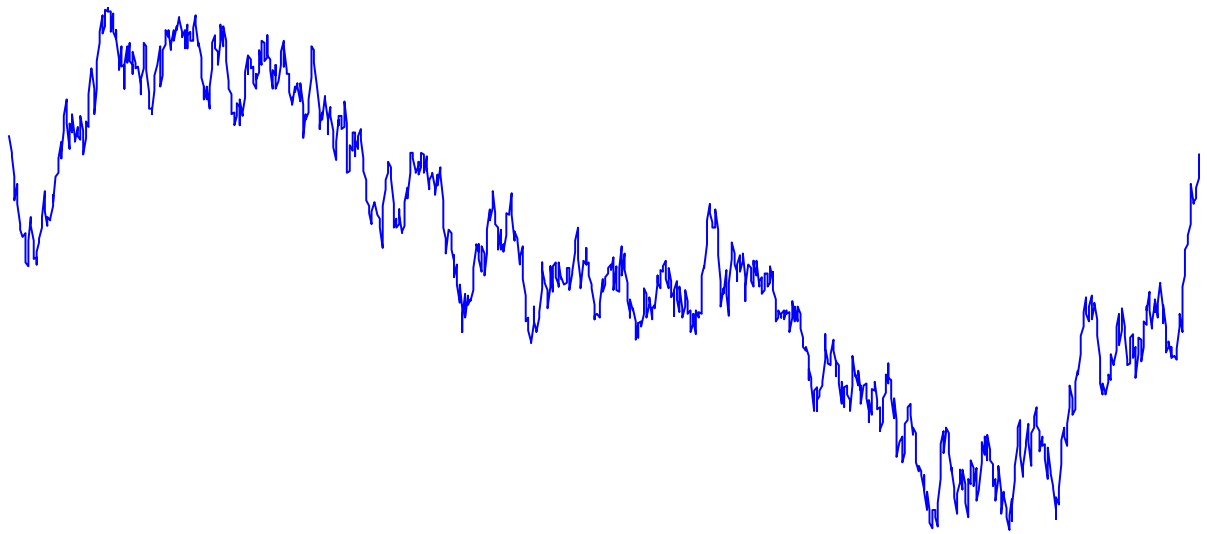
Riccardo Guidotti

Università di Pisa

Syllabus

- Deep-learning Models
- Kernel-based Models
- Ensemble Models

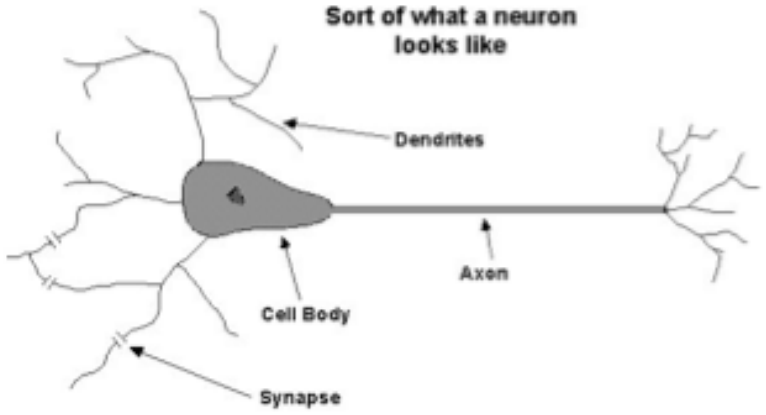
- Time Series Transformations

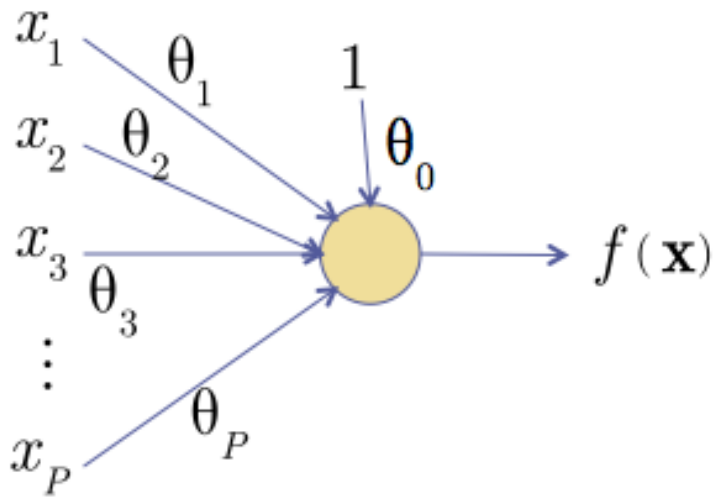


Deep-learning Models

The Neuron Metaphor

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node

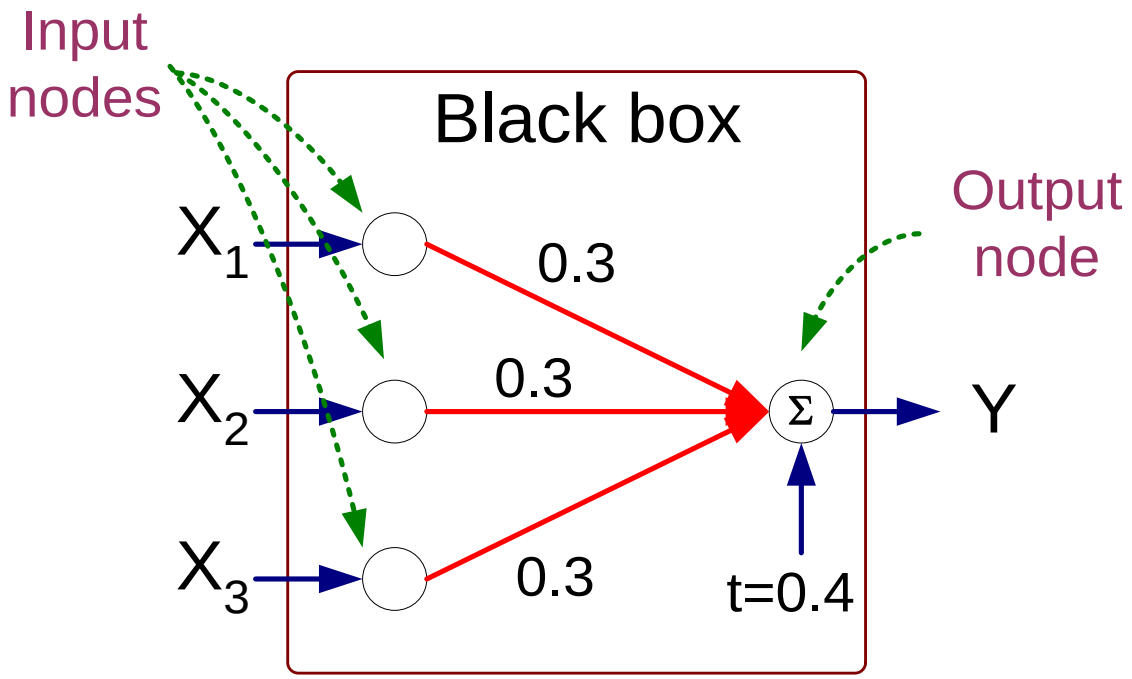




Neural Networks

NN - Neural Networks

- A NN models is an assembly of inter-connected nodes and weighted links.
- Output nodes sum up each of its input value according to the weights of its links.
- The node outcome is passed through an activation function that typically squeeze the input-weight dot product towards an extreme value used for the final prodiction.



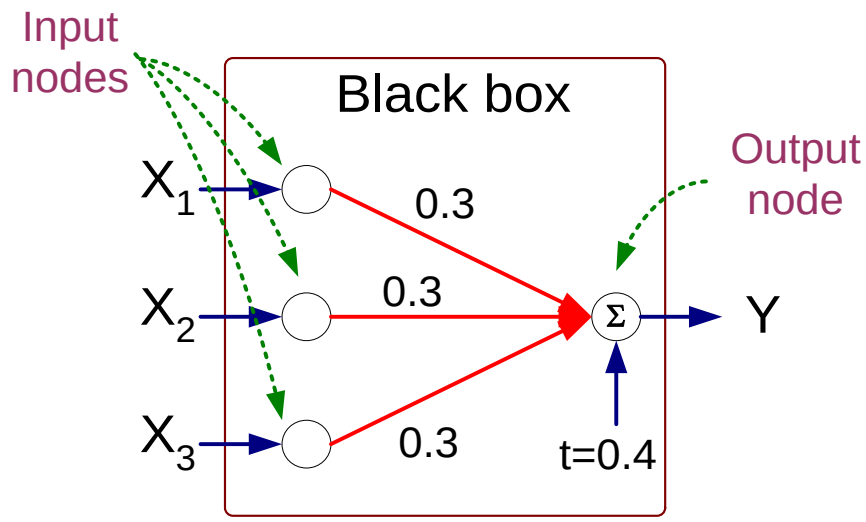
$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Characterizing a NN

- Input/Output signal may be:
  - Real value
  - Unipolar {0, 1}
  - Bipolar {-1, +1}

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



- **Weight** (w or sigma): ϑ_{ij} – strength of connection from unit *j* to unit *i*

- Learning amounts to **adjusting the weights** ϑ_{ij} by means of an **optimization algorithm** aiming to minimize a cost function.

- As in biological systems, training a NN model amounts to adapting the weights of the links until they fit the input output relationships of the underlying data.

Characterizing a NN

- The outcome of a node net_i of a NN is the dot product between the input and the weights.
- The bias b is a constant value that can be written as $\vartheta_{i0}x_0$ with $x_0 = 1$ and $\vartheta_{i0} = b$ such that

$$net_i = \sum_{j=0}^{n} \vartheta_{ij} x_j$$

- The function $f(net_i(x))$ applied to net_i is the unit's **activation function**.

Characterizing a NN

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

- **Linear Unit*:*** activation function *f* is the **identity** function, then the unit's output is just its net input.

where $\sigma(a) = a$

- **Linear Threshold Unit**: activation function *f* is the **sign** function.

where $\sigma(a) = \begin{cases} +1 \; a \geq 0 \\ -1 \; a < 0 \end{cases}$

- **Logistic Unit**: activation function *f* is the **sigmoid** function

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

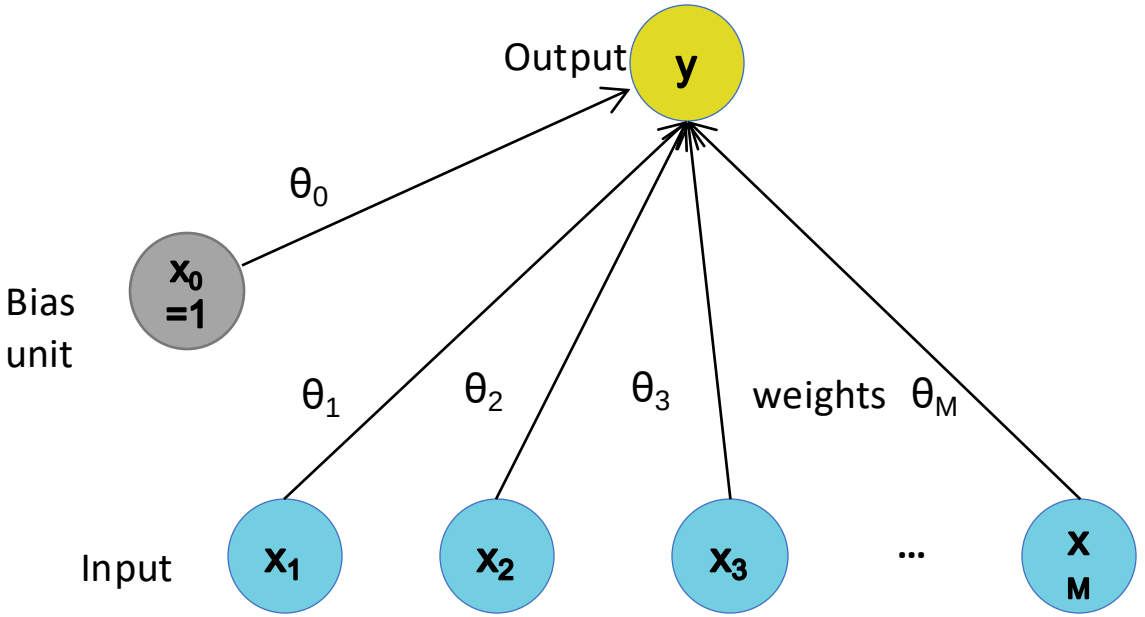
Single Layer Perceptron Network

- Contains only input and output nodes
- Activation function: $f = sign(w \bullet x)$
- Applying model is straightforward

$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

$X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = sign(0.2) = 1$



Learning Iterative Procedure

- During the training phase the weight parameters are adjusted until the outputs of the perceptron become consistent with the true outputs of the training examples.
- Initialize the weights $(w_0, w_1, \ldots, w_m)$
- Repeat
  - For each training example (x_i, y_i)
    - Compute $f(w^{(k)}, x_i)$
    - Update the weights: $w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$
- Until stopping condition is met

Iteration index

Learning rate

Perceptron Learning Rule

- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i \;\; ; \;\; \lambda : \text{learning rate}$$

- Intuition:
  - Update weight based on error: $e = \left[y_i - f(w^{(k)}, x_i)\right]$
  - If *y=f(x,w), e=0*: no update needed
  - If *y>f(x,w), e=2*: weight must be increased so that *f(x,w)* will increase
  - If *y<f(x,w), e=-2*: weight must be decreased so that *f(x,w)* will decrease

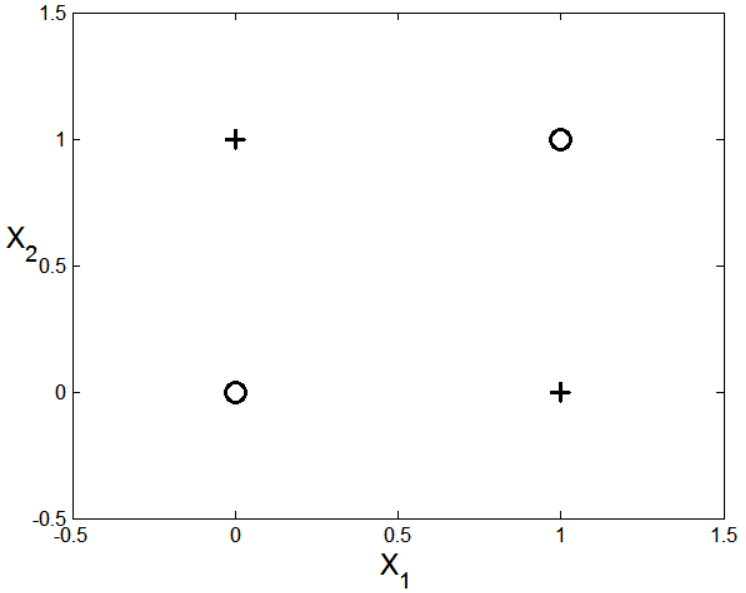
Nonlinearly Separable Data

- Since *f(w,x)* is a linear combination of input variables, decision boundary is linear.
- For nonlinearly separable problems, the perceptron fails because no linear hyperplane can separate the data perfectly.
- An example of nonlinearly separable data is the XOR function.
- Nonlinearly separable data problems can be solved by Deep NN

XOR Data

x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1

$$y = x_1 \oplus x_2$$

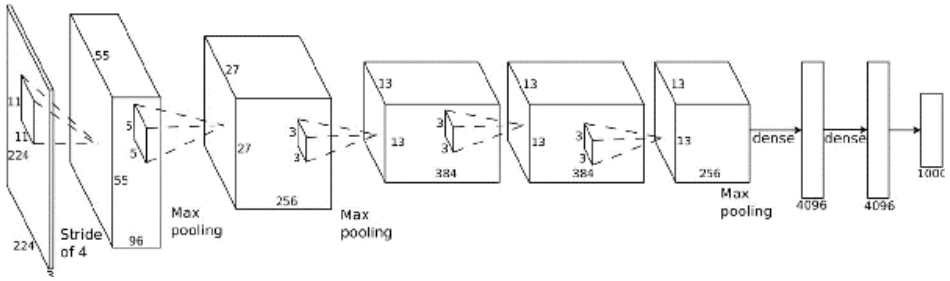


Deep Neural Networks

Deep Neural Networks - Why Now?

(Big) Data

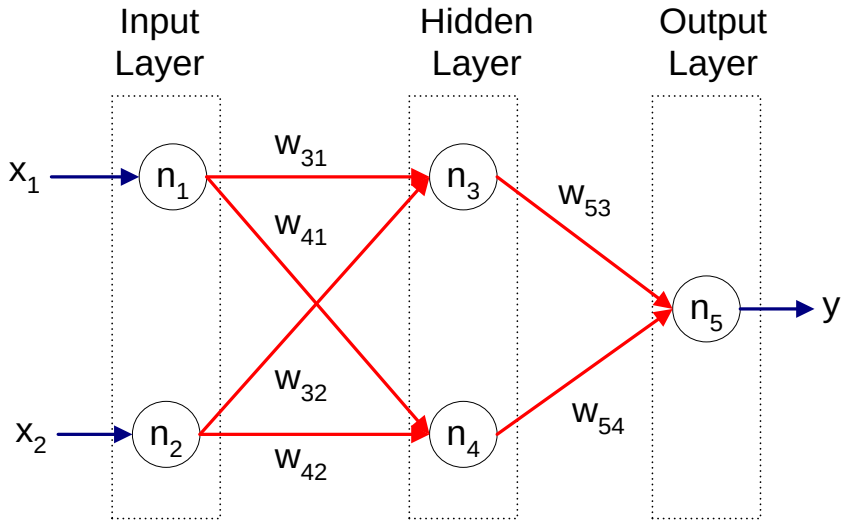
GPU



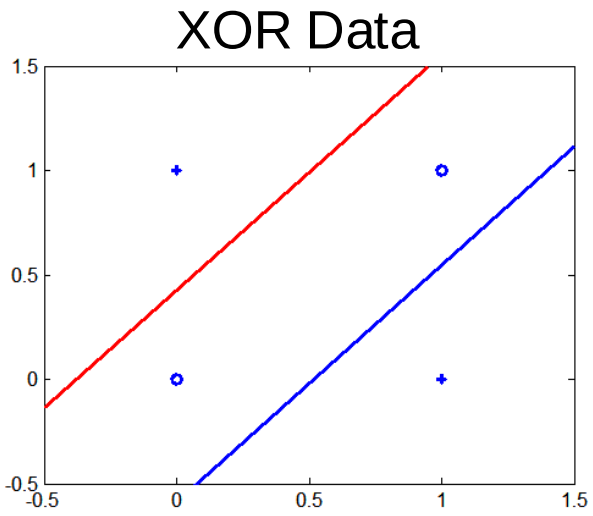
Theory

DNN - Multilayer Neural Network

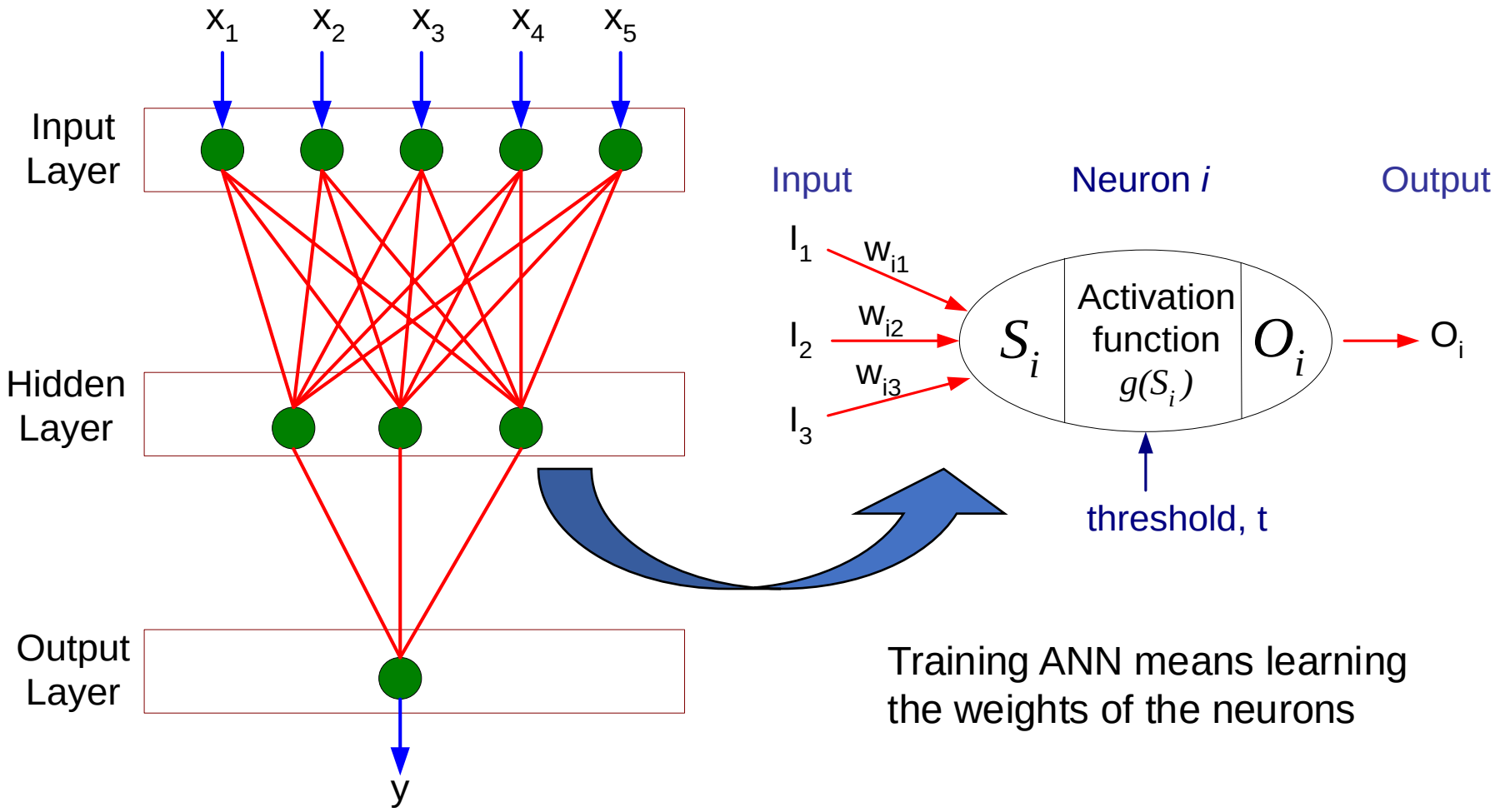
- **Hidden Layers**: intermediary layers between input and output layers.
- More general **activation functions** (sigmoid, linear, hyperbolic tangent, etc.).



- DNN can solve any type of predictive task involving nonlinear decision surfaces.
- A perceptron is single layer.
- We can think to each hidden node as a perceptron that tries to construct one hyperplane, while the output node combines the results to return the decision boundary.



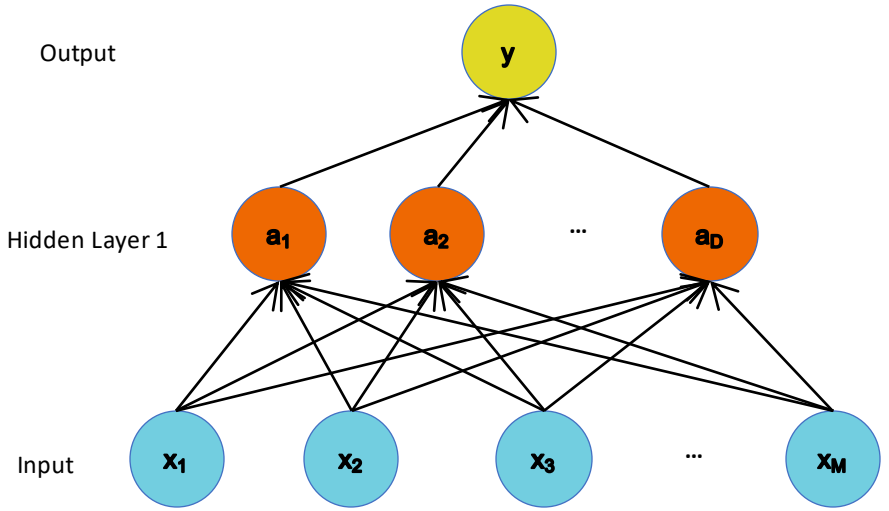
Training a DNN



Training ANN means learning the weights of the neurons

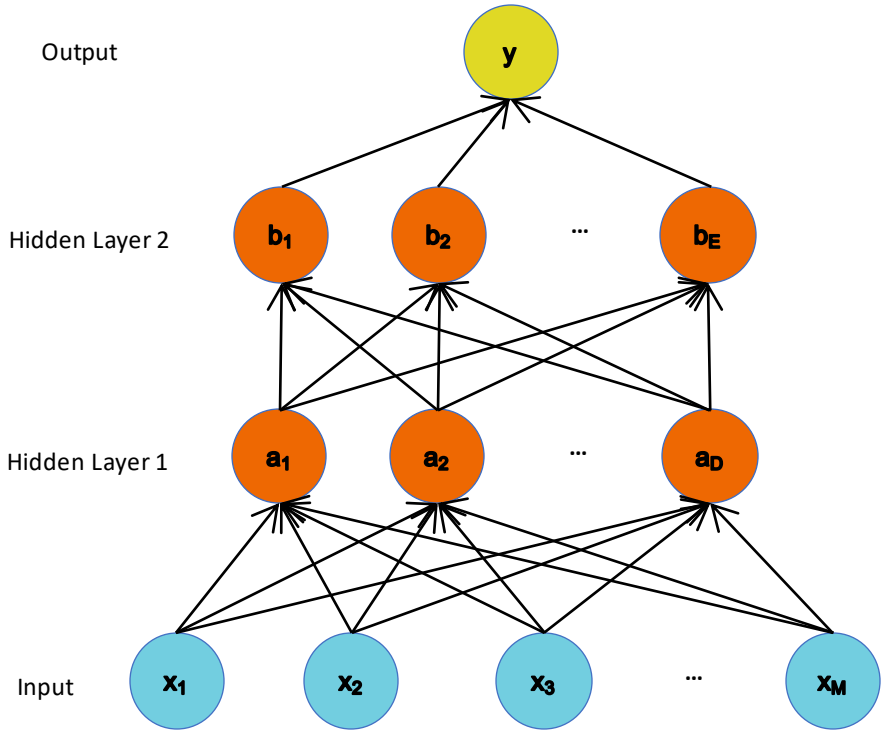
Deep Neural Networks

- One hidden layer



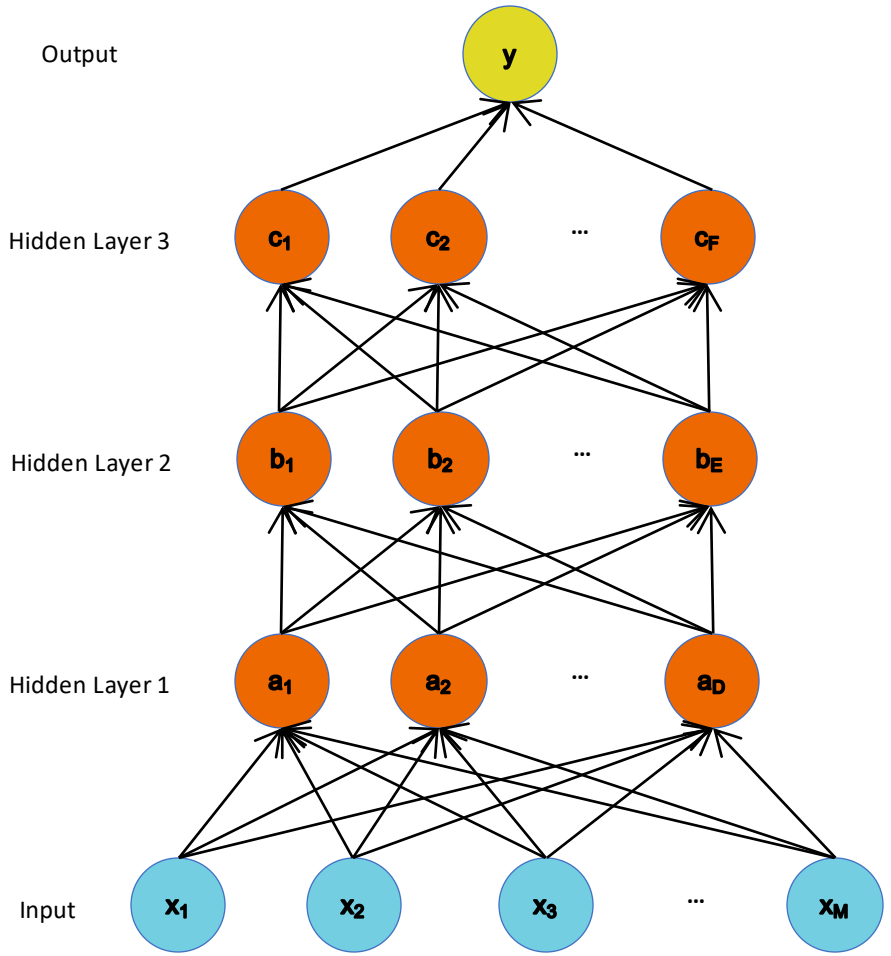
Deep Neural Networks

- Two hidden layers

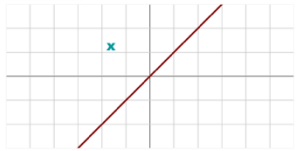
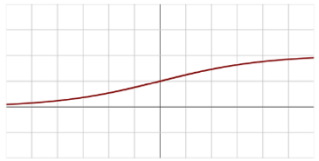
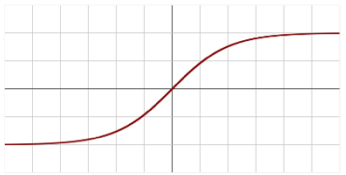


Deep Neural Networks

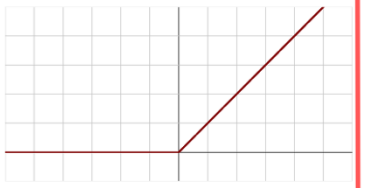
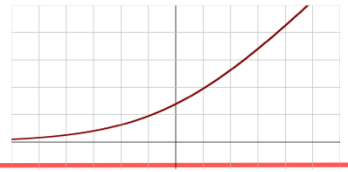
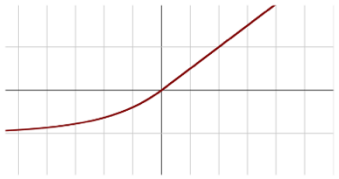
- As many layers as you want.
- Deep learning is way more than having NN with a lot of layers.
- Various types of activation functions.
- Various types of NN topology.



Activation Functions

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		f(x) = x	f'(x) = 1	(-∞, ∞)
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	{0, 1}
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	(0, 1)
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$f'(x) = 1 - f(x)^2$	(-1, 1)

Activation Functions

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	(0, 1)
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	(-1, 1)
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

Learning Multilayer Neural Network

- Can we apply perceptron learning to each node, including hidden nodes?
- Perceptron computes error *e = y-f(w,x)* and updates weights accordingly
- Problem: how to determine the true value of *y* for hidden nodes?
- Solution: Gradient Descent optimization

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - f(\sum_j w_j x_{ij}) \right)^2$$

Sum of Squared Residuals

Quadratic function from which we can find a global minimum solution

- Given a loss function
- Weight update:

$$w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$$

Slope of the Activation Function obtained as partial derivative by the Gradient Descent

- Activation functions *f* must be differentiable

Loss Functions for NN

- Regression:
  - Output Layer: One node with a linear activation unit.
  - Loss Function: Quadratic Loss (Mean Squared Error (MSE))
    - $\frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i-\hat{y}_i)^2$ (back $y_i-\hat{y}_i$)
- Classification:
  - Output Layer:
    - One node with a sigmoid activation unit (K=2, binary cross-entropy)
    - K nodes in a softmax layer (K>2, categorical cross-entropy)
  - Loss Function: Cross-entropy (i.e. negative log likelihood)
    - Binary: $-\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i\log(y_i)+(1-\hat{y}_i)\log(1-y_i)$ (back $\hat{y}_i/y_i+(1-\hat{y}_i)/(1-y_i)$)
    - Multiclass: $-\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{c}\hat{y}_{ik}\log(\hat{y}_{ik})$ (back $\hat{y}_{ik}/y_{ik}$)

Design Issues in a DNN

- Number of nodes in input layer
  - One input node per binary/continuous attribute
  - k or $log_2 k$ nodes for each categorical attribute with k values
- Number of nodes in output layer
  - One output for binary class problem
  - k or $log_2 k$ nodes for k-class problem
- Number of nodes in hidden layer
- Initial weights and biases

Problems with DNN

- DNN are universal approximators but could suffer from ***overfitting*** if the network is too large.
- Gradient descent may converge to ***local minimum***.
- Model building can be very time consuming, but testing can be very fast.
- Can handle redundant attributes because weights are automatically learnt.
- Sensitive to noise in training data.
- Difficult to handle missing attributes.

Dataset Should Normally be Split Into

- ***Training set:*** use to update the weights. Records in this set are repeatedly in random order. The weight update equation are applied after a certain number of records.

- ***Validation set:*** use to decide when to stop training only by monitoring the error and to select the best model configuration

- ***Test set:*** use to test the performance of the neural network. It should not be used as part of the neural network development and model selection cycle

Before Starting: Weight Initialization

- Choice of ***initial weight values is important as this decides starting position in weight space.*** That is, how far away from global minimum
  - Aim is to select weight values which produce midrange function signals
  - Select weight values randomly from uniform probability distribution
  - Normalize weight values so number of weighted connections per unit produces midrange function signal

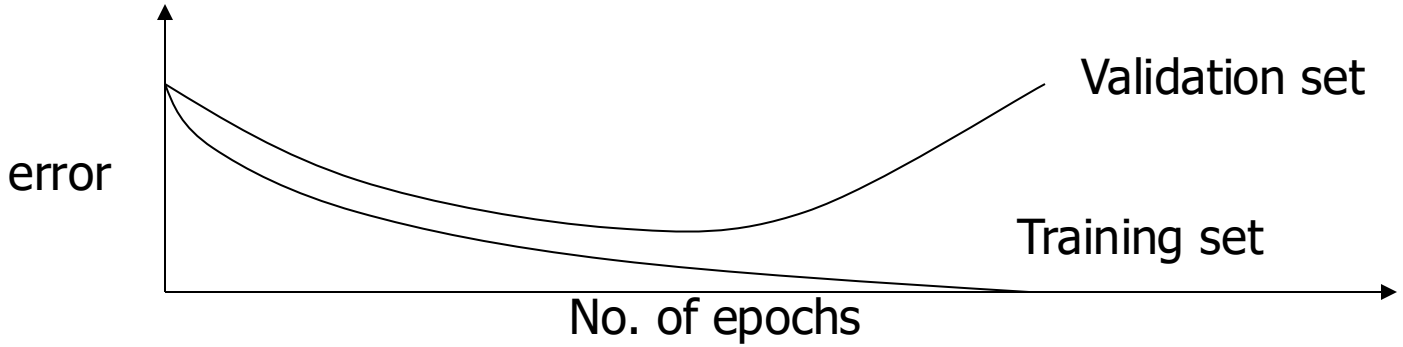
- Try different random initialization to
  - Assess robustness
  - Have more opportunities to find optimal results

Convergence Criteria

- Learning is obtained by repeatedly supplying training data and adjusting by backpropagation
  - Typically 1 training set presentation = **1 epoch**
- We need a stopping criteria to define convergence
  - Euclidean norm of the gradient vector reaches a sufficiently small value
  - Absolute rate of change in the average squared error per epoch is sufficiently small
  - **Validation for generalization performance: stop when generalization performance reaches a peak**

Early Stopping

- Running too many epochs may **overtrain** the network and result in **overfitting** and perform poorly in generalization
- Keep a hold-out validation set and test accuracy after every epoch. Maintain weights for best performing network on the validation set and stop training when error increases beyond this
- Always let the network run for some epochs before deciding to stop (**patience parameter**), then backtrack to best result



Model Selection

- **Too few hidden units** prevent the network from learning adequately fitting the data and learning the concept.
- **Too many hidden units** leads to overfitting, unless you regularize heavily (e.g. dropout, weight decay, weight penalties)
- Cross validation should be used to determine an appropriate number of hidden units by using the optimal validation error to select the model with optimal number of hidden layers and nodes.

Regularization

- Constrain the learning model to avoid overfitting and help improving generalization.
- Add **penalization terms** to the loss function that *punish* the model for excessive use of resources
  - Limit the **number of weights** that is used to learn a task
  - Limit the **total activation of neurons** in the network

$$E' = E(y, y^*) + \lambda R(\cdot)$$

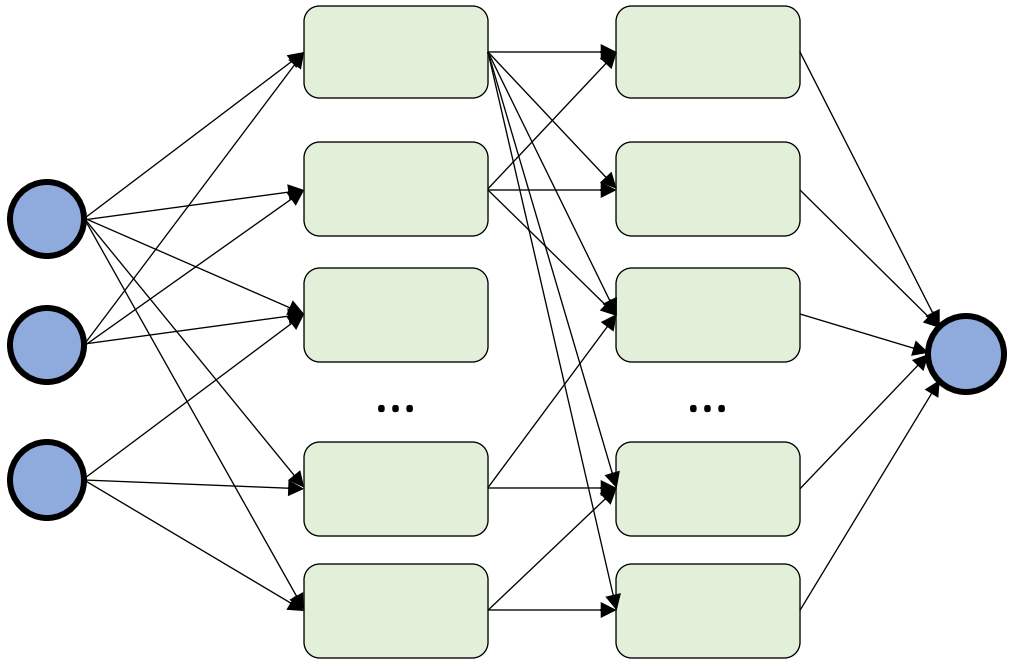
Hyperparameter to be chosen in model selection

$R(W_\theta)$ Penalty on parameters

$R(Z)$ Penalty on activations

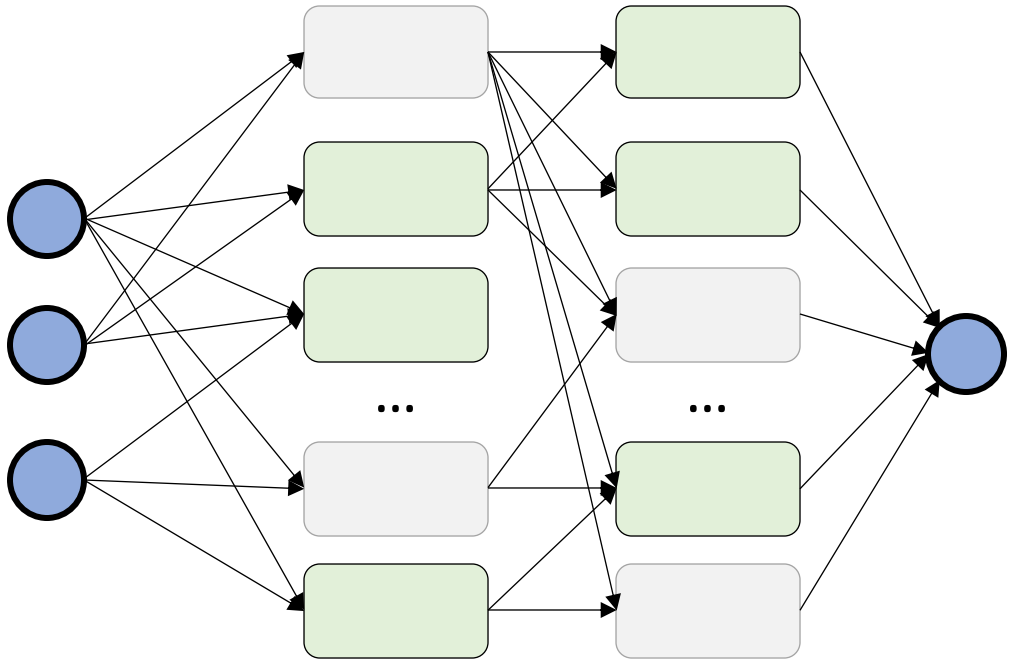
Dropout Regularization

Randomly disconnect units from the network during training



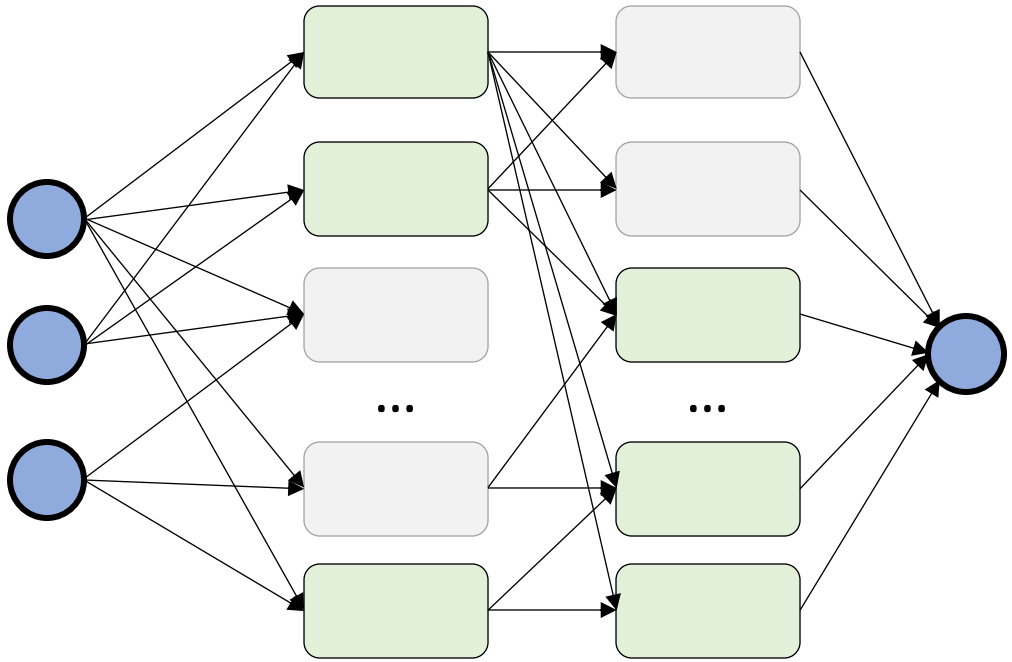
Dropout Regularization

Randomly disconnect units from the network during training



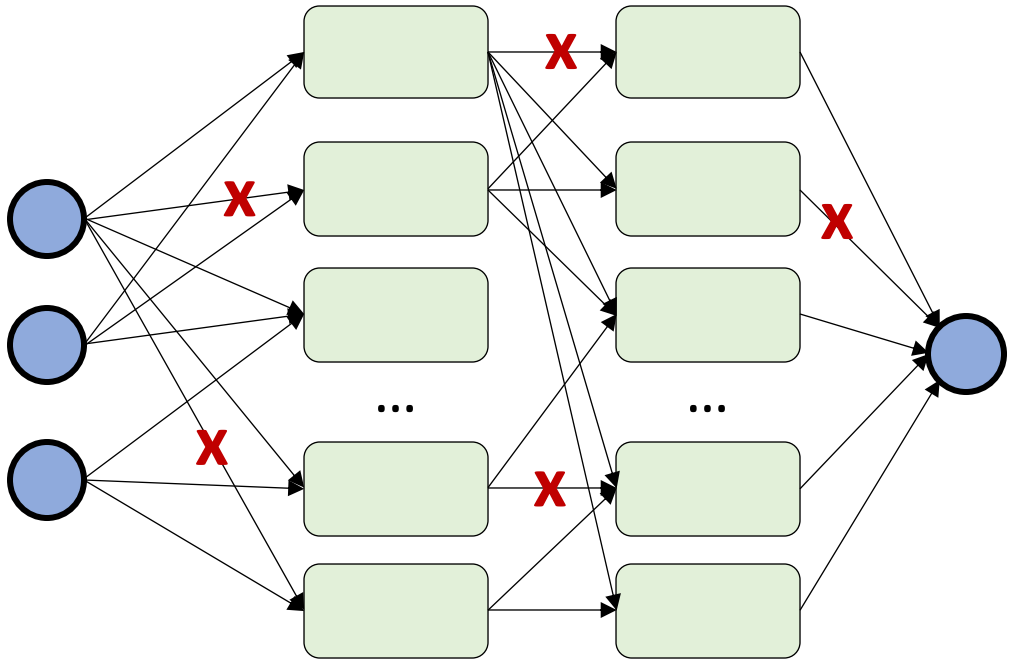
Dropout Regularization

Randomly disconnect units from the network during training



Dropout Regularization

Randomly disconnect units from the network during training



- Regulated by unit **dropping hyperparameter**
- Prevents unit **coadaptation**
- Committee machine effect
- Need to adapt **prediction phase**
- Used at prediction time gives **predictions with confidence intervals**

You can also **drop single connections** (dropconnect)

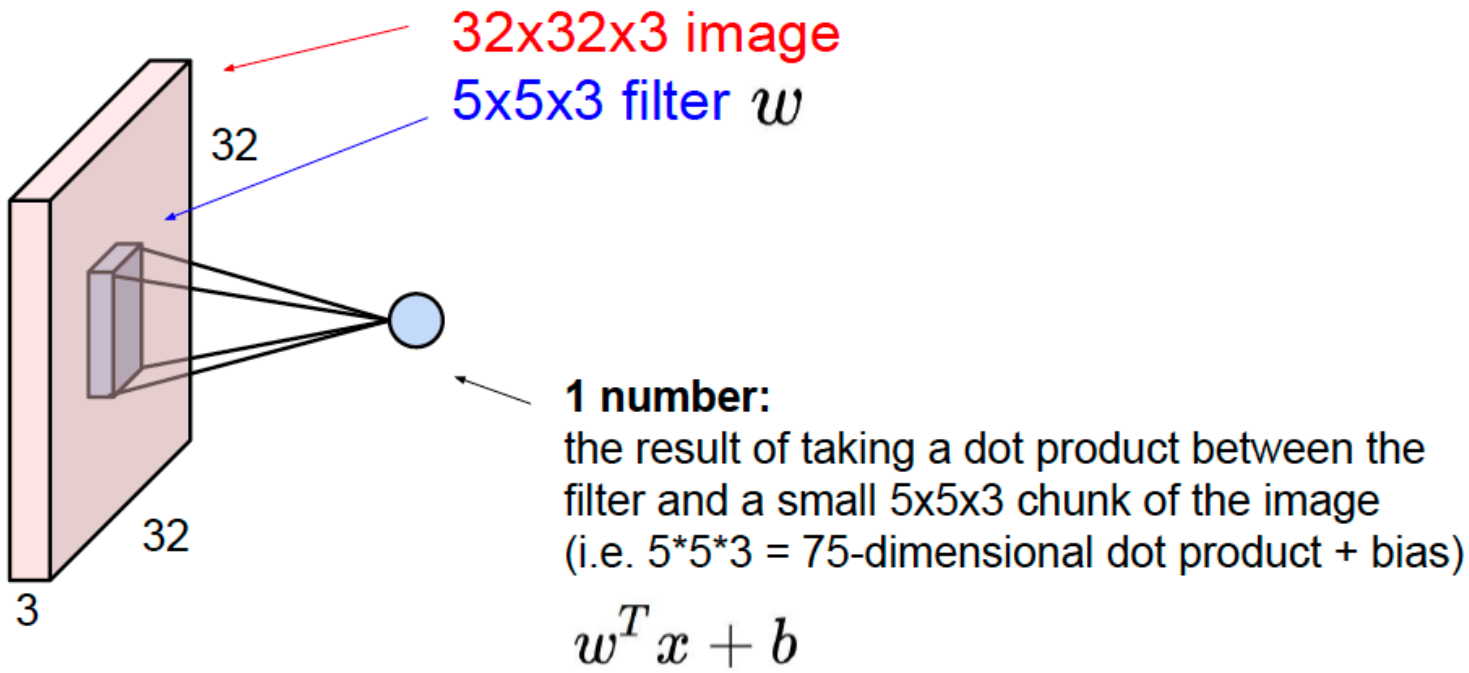
Convolutional Neural Network

Slides edited from Stanford

http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture09.pdf

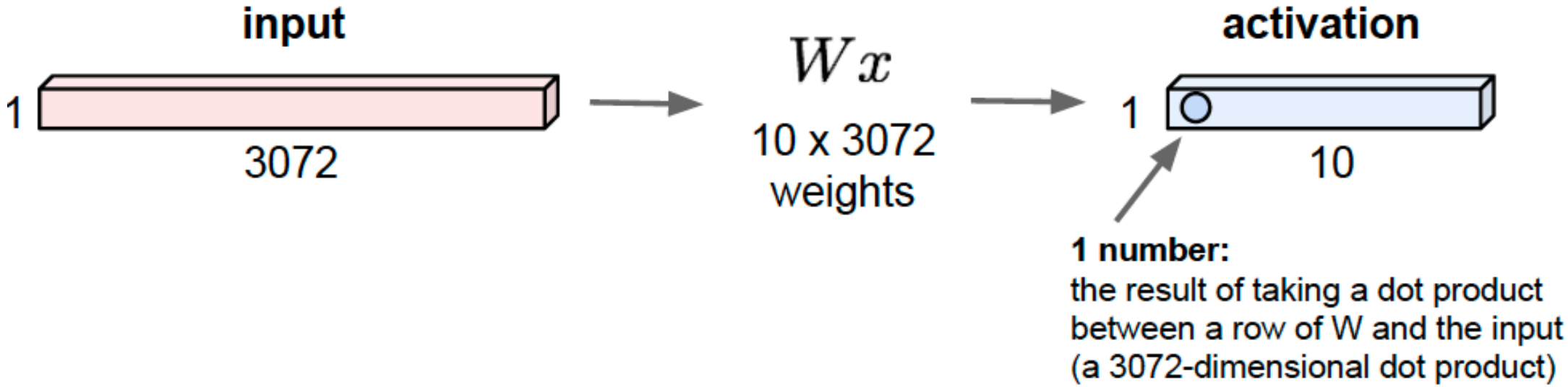
Convolutional Neural Networks

- Instead of having only “fully connected” layers adopt “convolutional layers”
- Are typically applied for the classification of images and time series



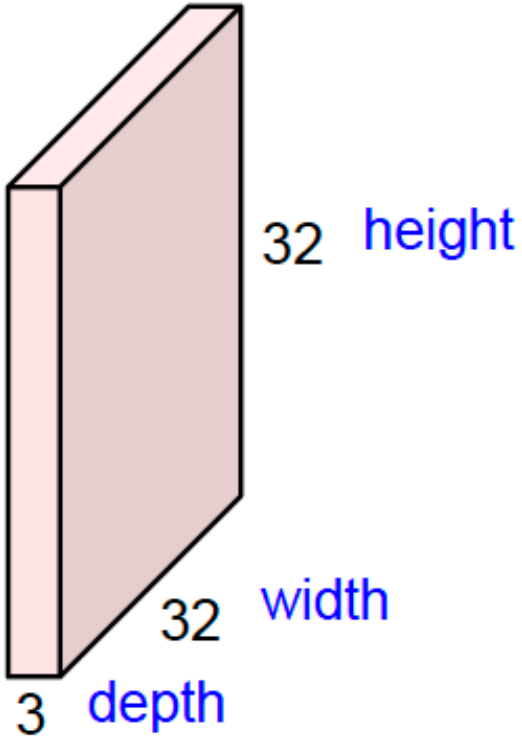
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Convolution Layer

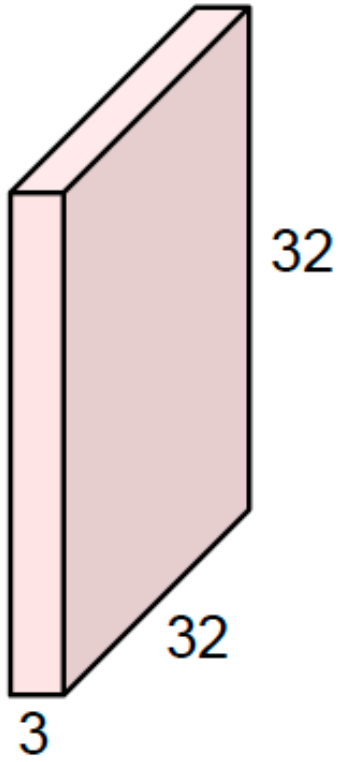
32x32x3 image -> preserve spatial structure



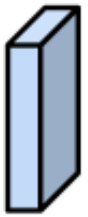
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

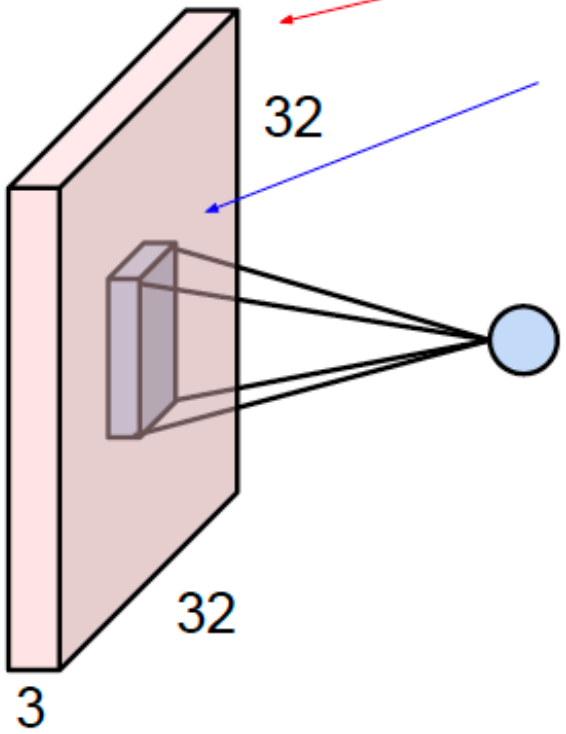


5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Convolution Layer



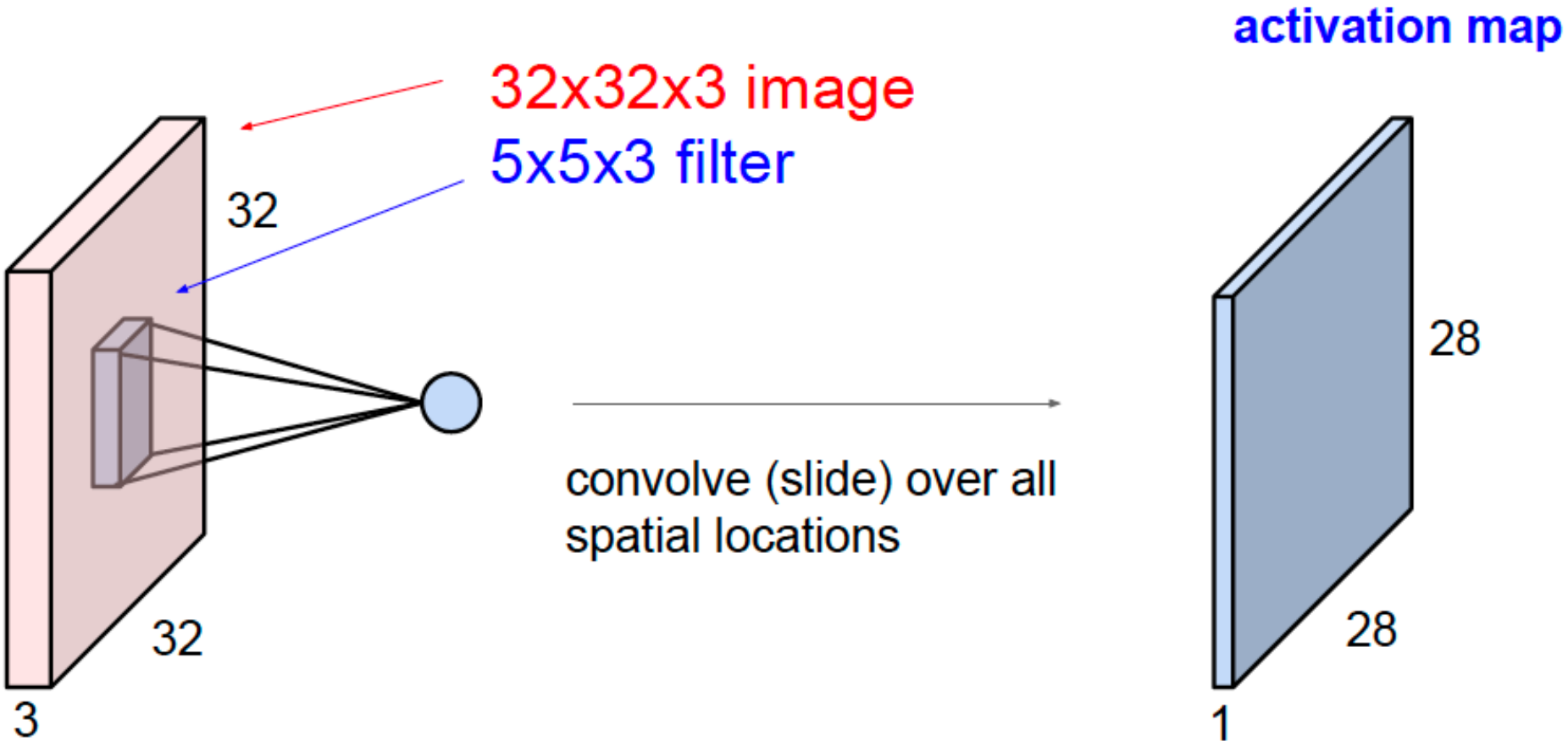
32x32x3 image

5x5x3 filter w

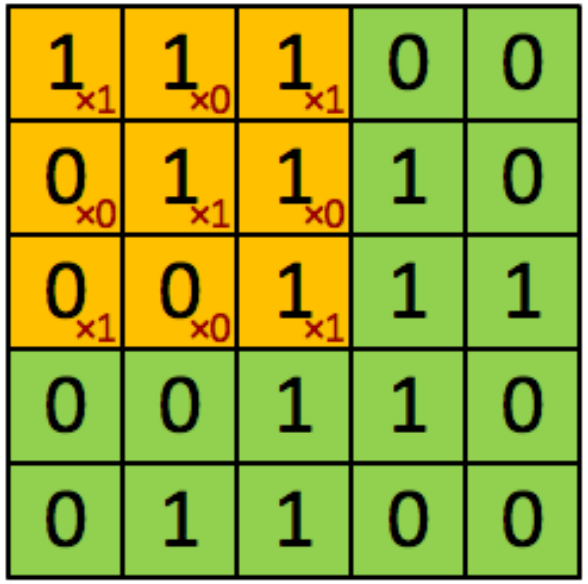
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

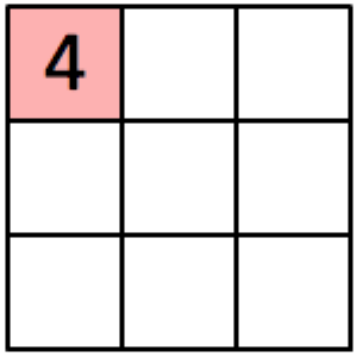
$$w^T x + b$$

Convolution Layer



Convolution Layer

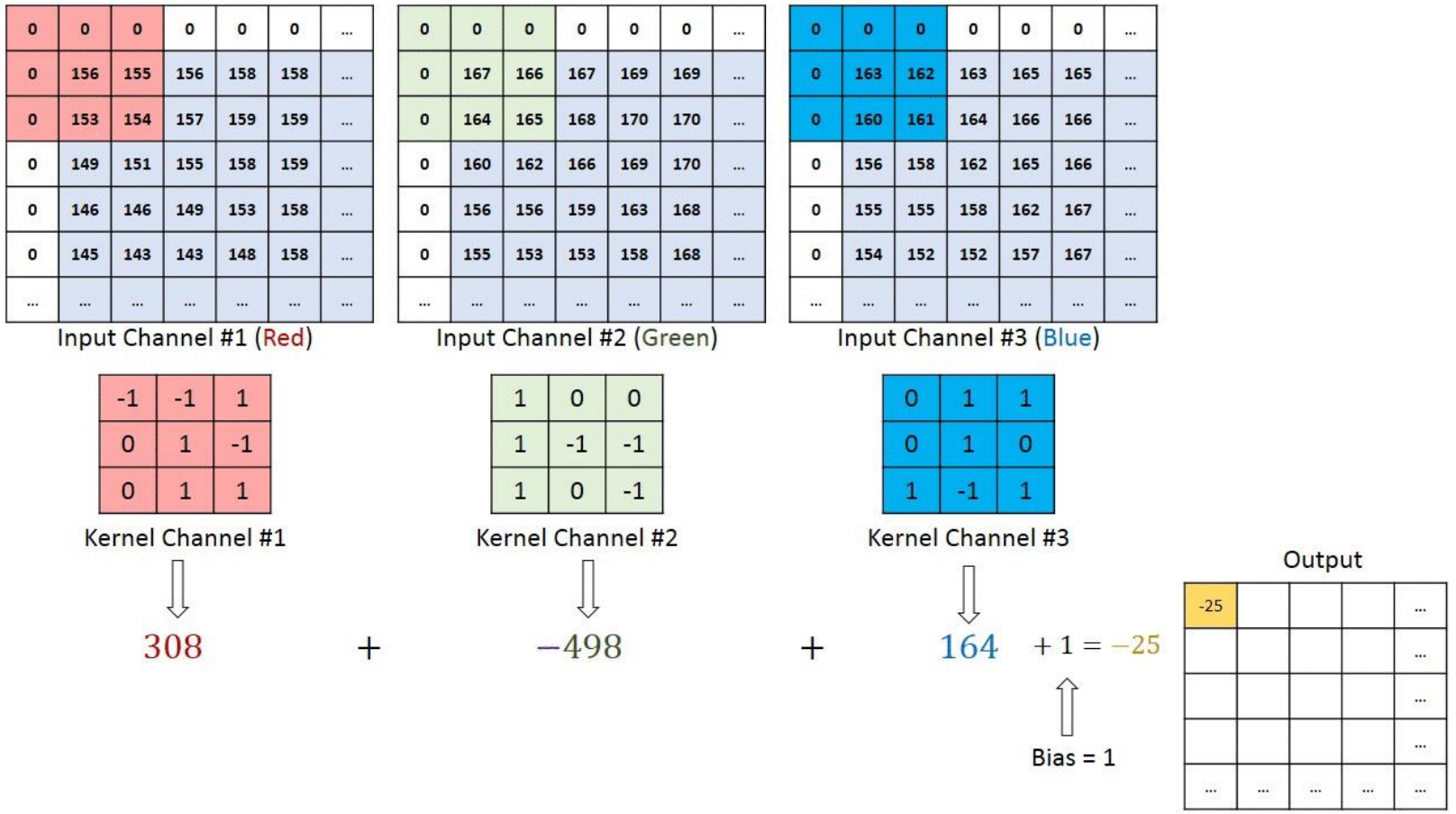




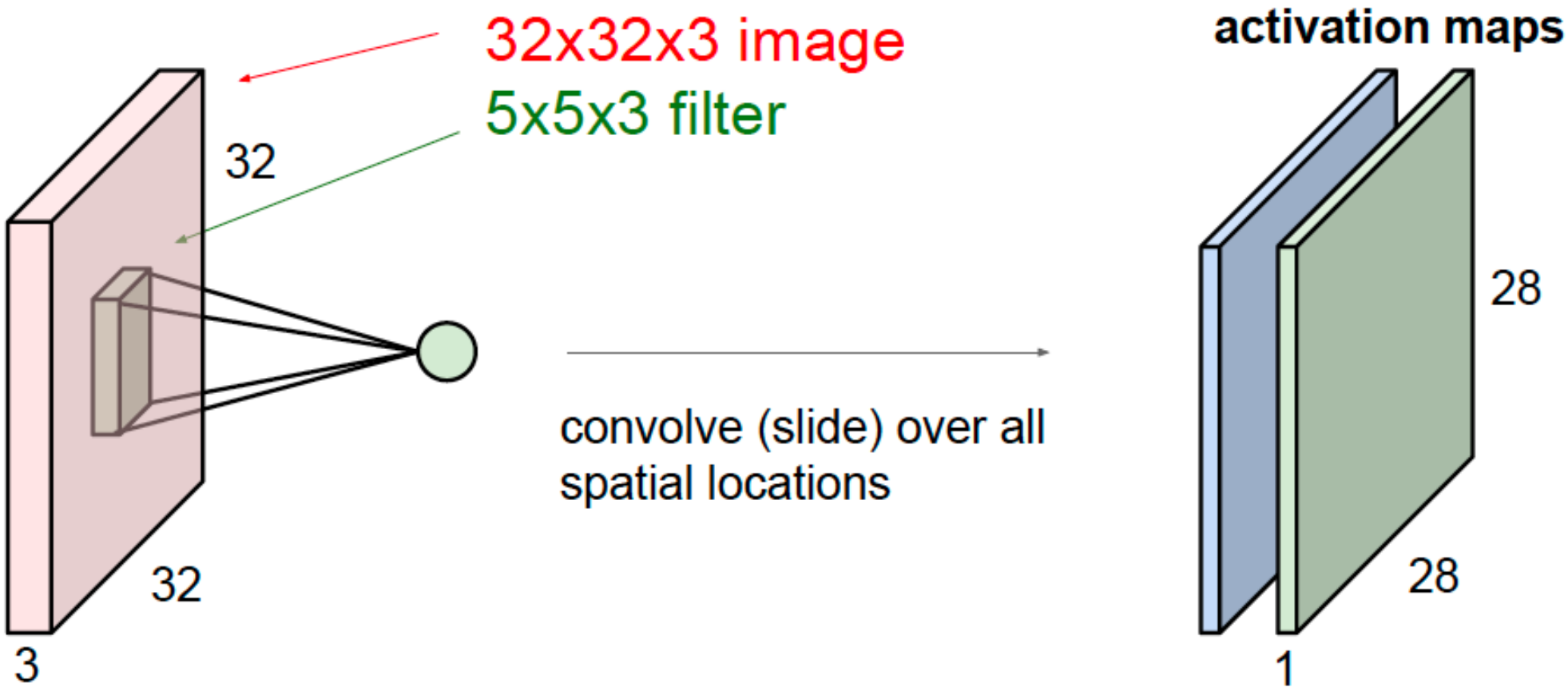
1	0	1
0	1	0
1	0	1

Convolution Kernel

Convolution Layer

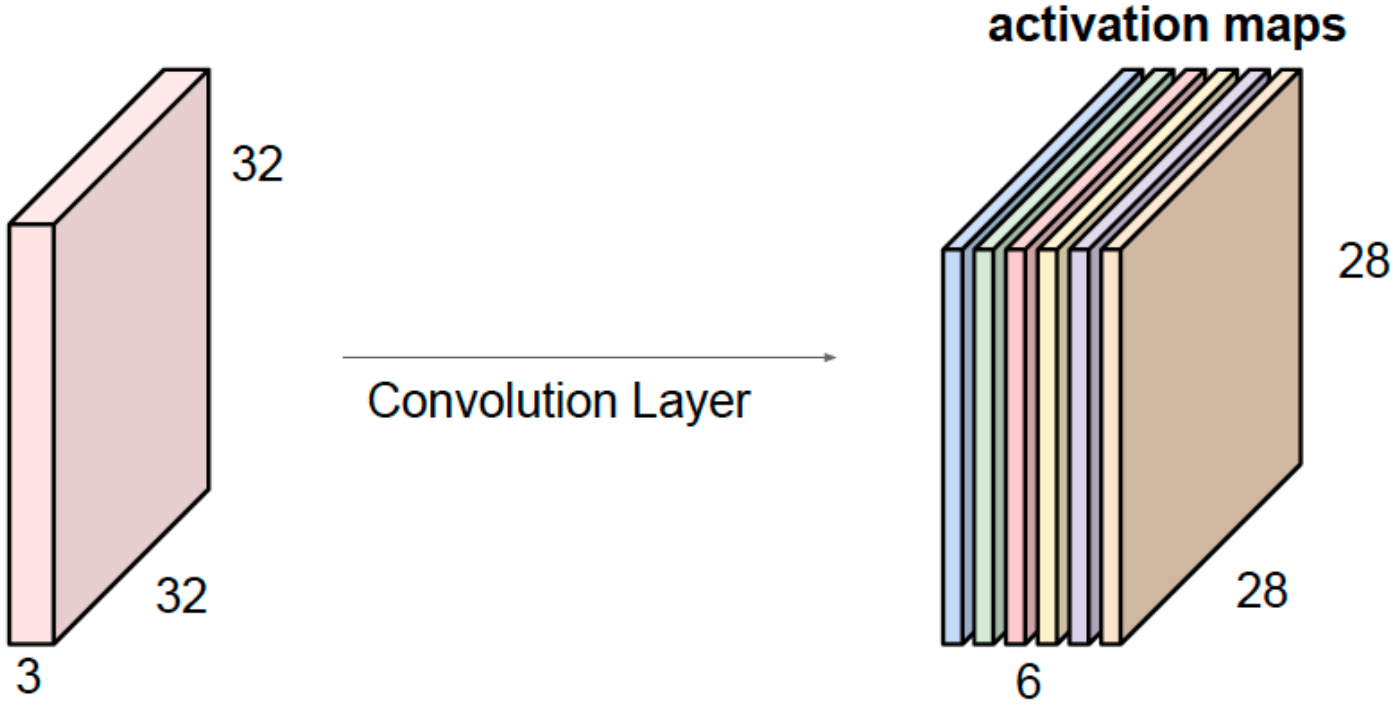


Convolution Layer



Convolution Layer

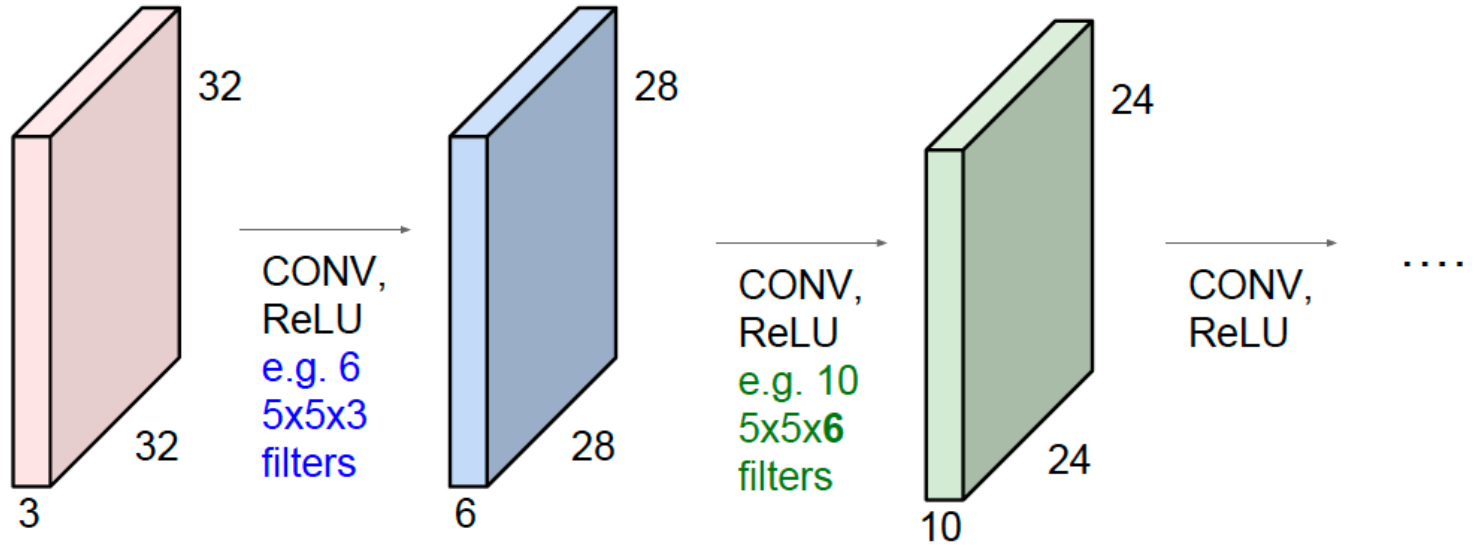
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolutional Neural Network

- CNN is a sequence of Conv Layers, interspersed with activation functions.
- CNN shrinks volumes spatially.
- E.g. 32x32 input convolved repeatedly with 5x5 filters! (32 -> 28 -> 24 ...).
- Shrinking too fast is not good, does not work well.



Convolutional Neural Network

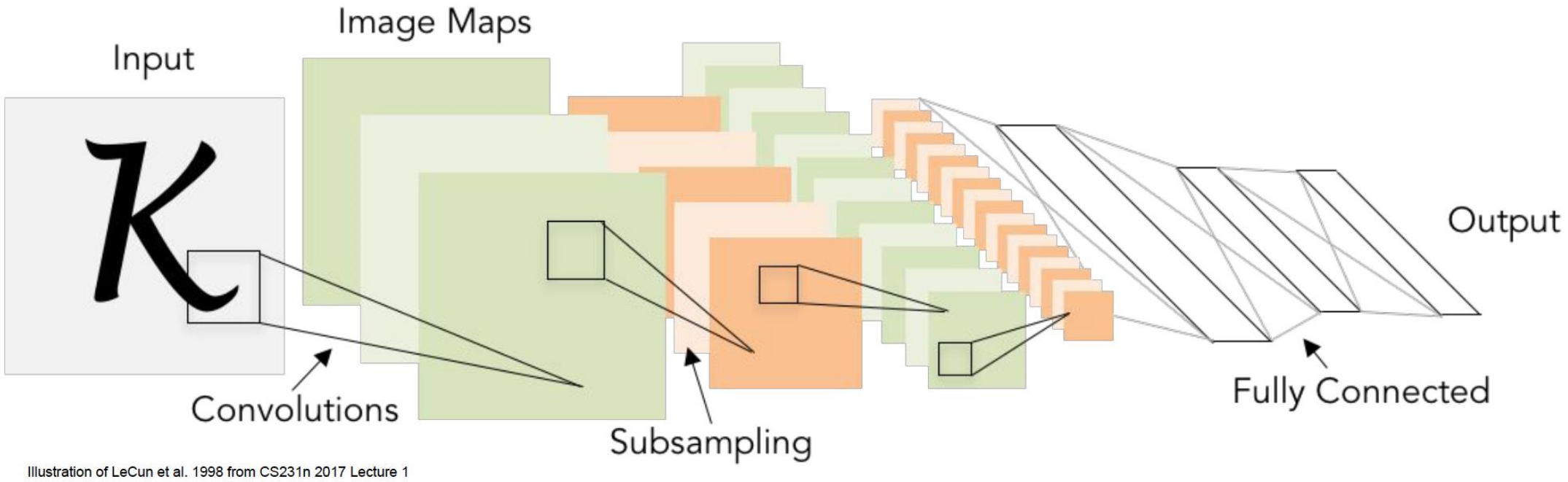
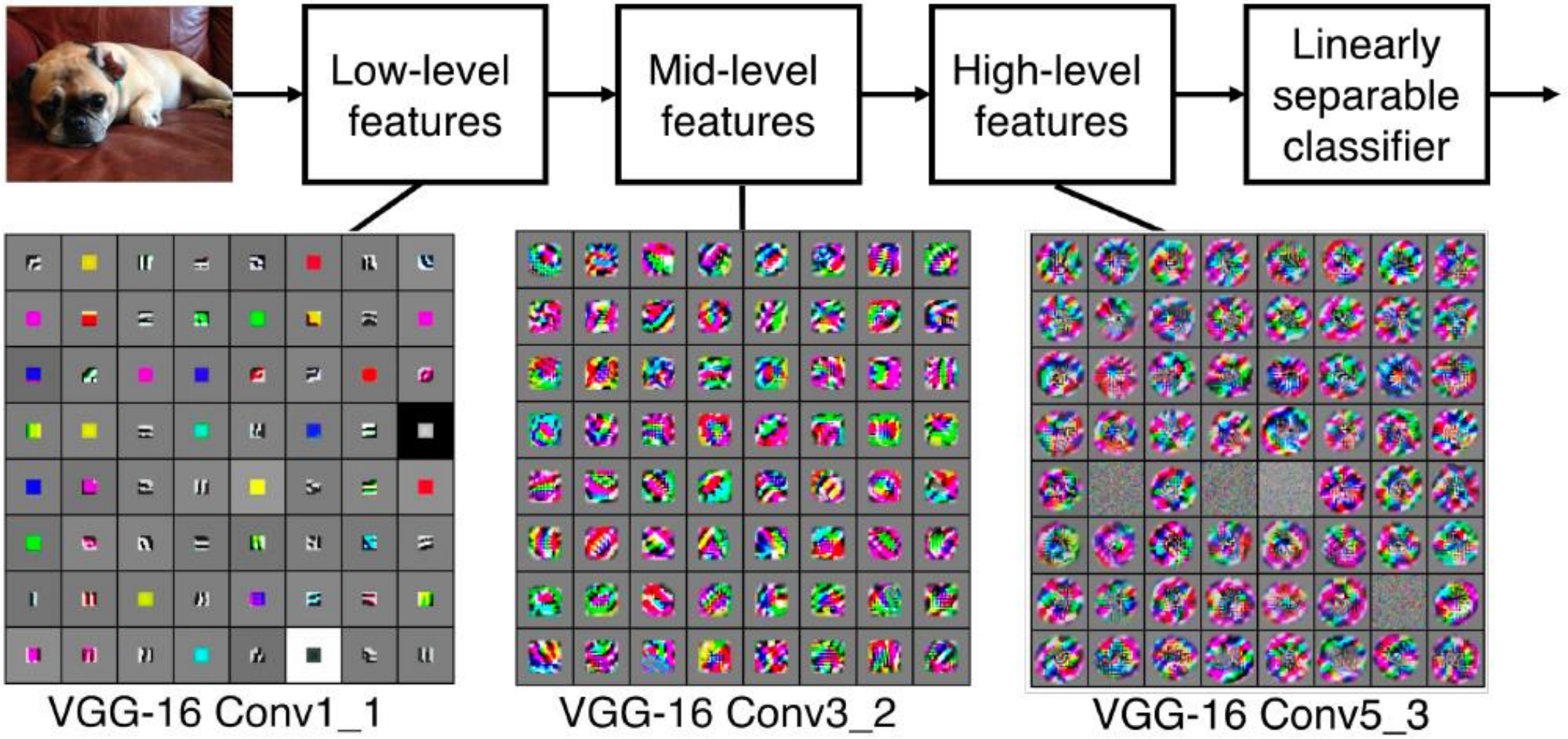
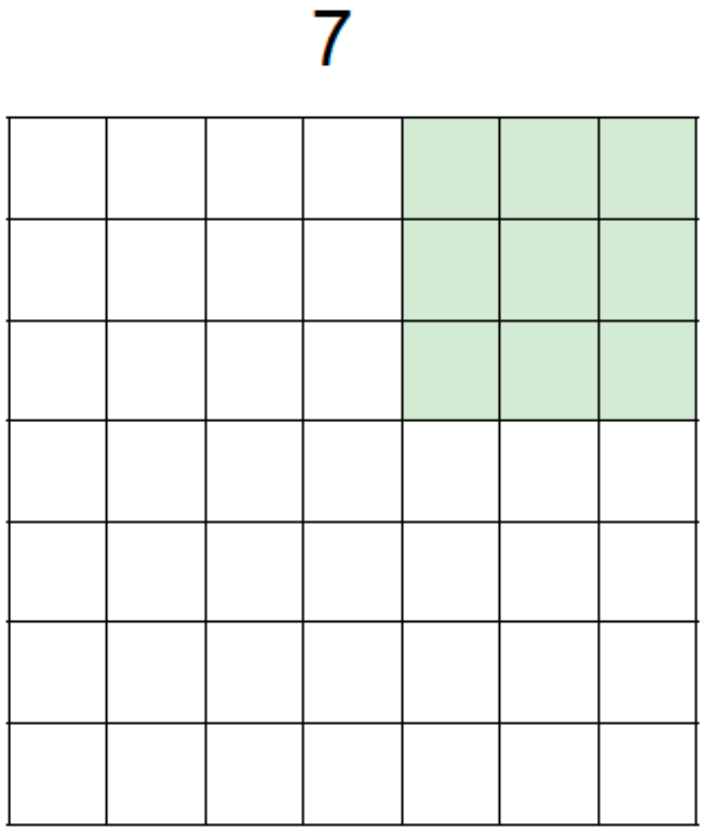


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

CNN for Image Classification



Stride

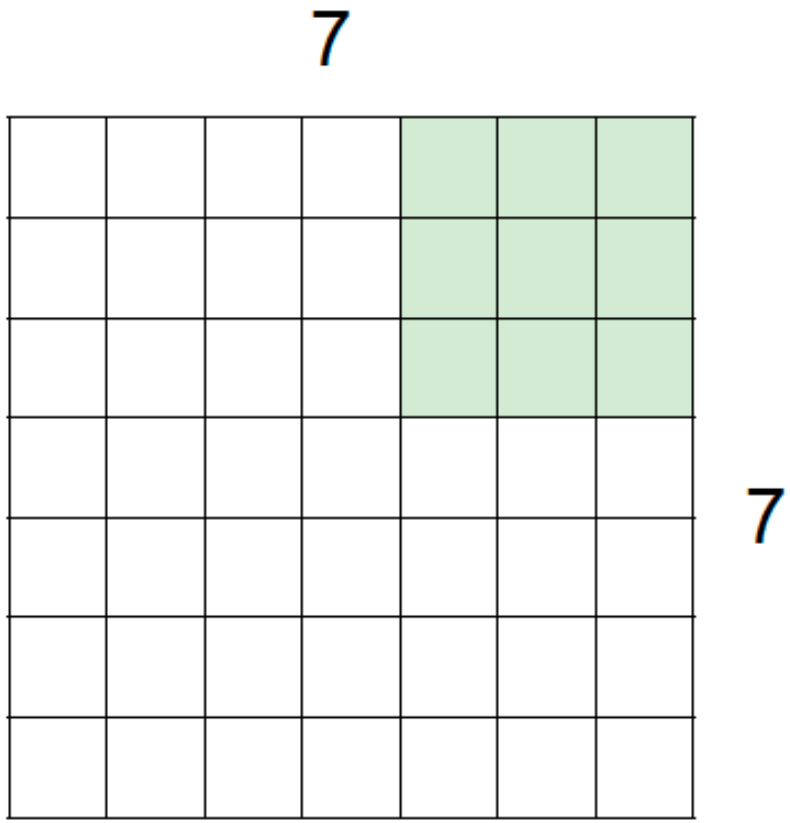


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

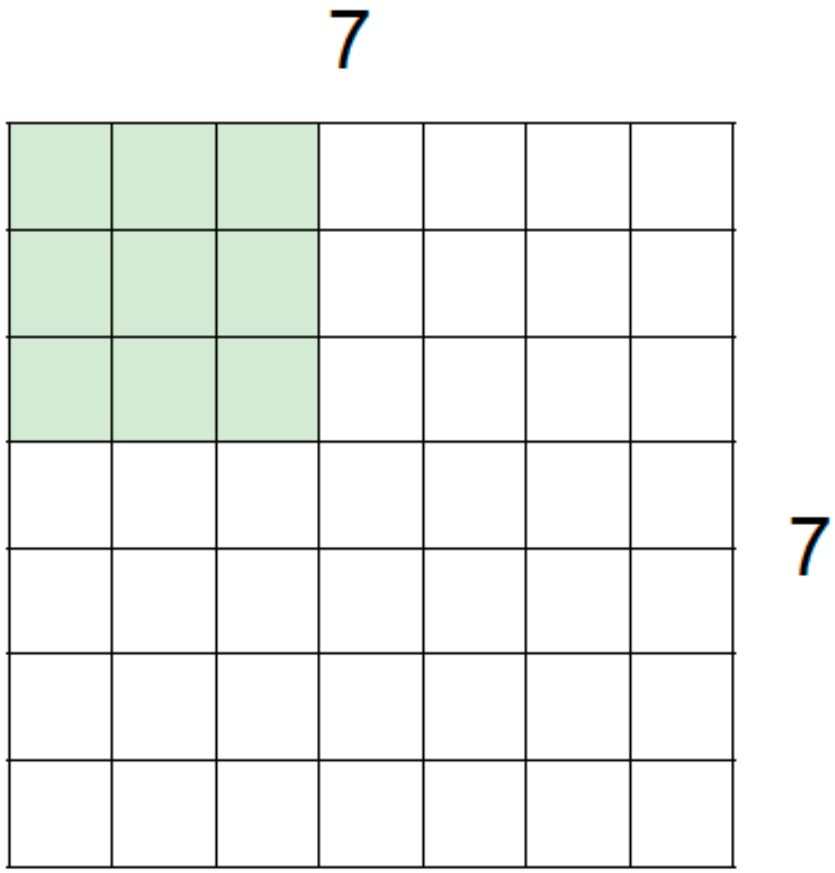
Stride: how much the kernel is moved along the input

Stride



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

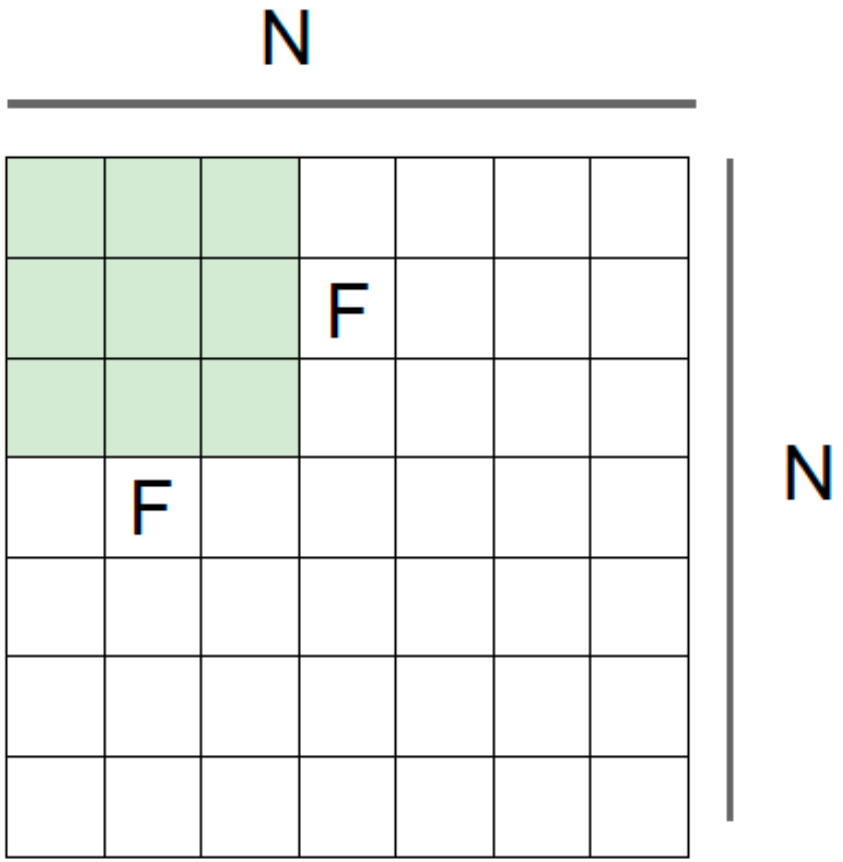
Stride



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

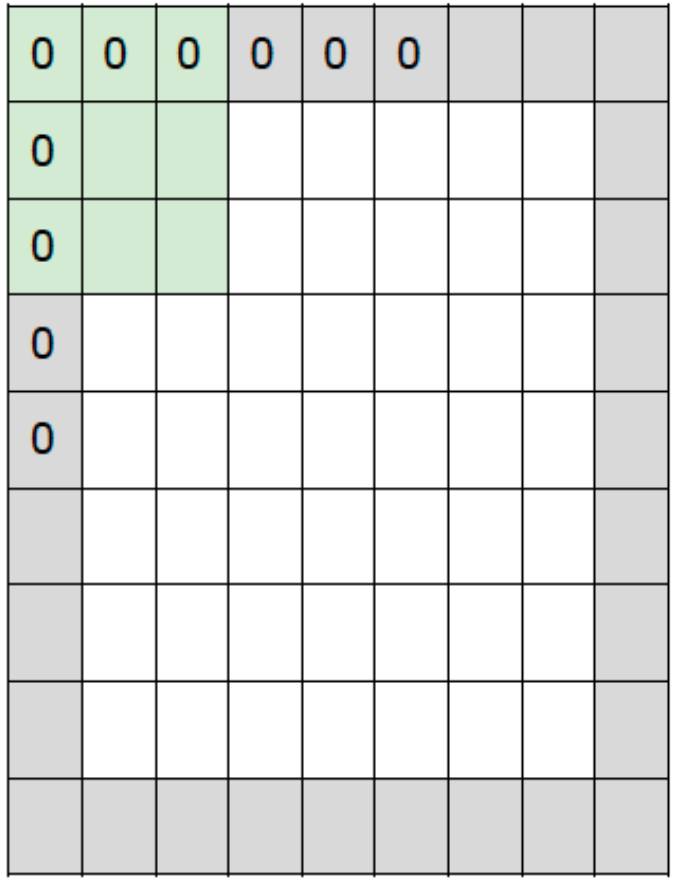
Stride



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Padding



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

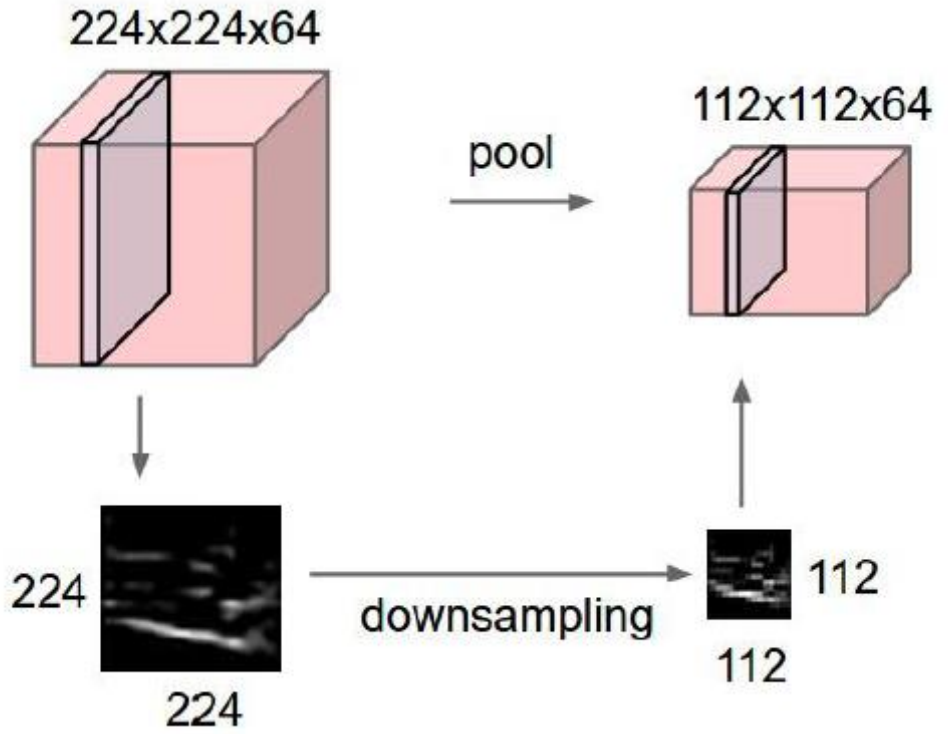
7x7 output!

In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

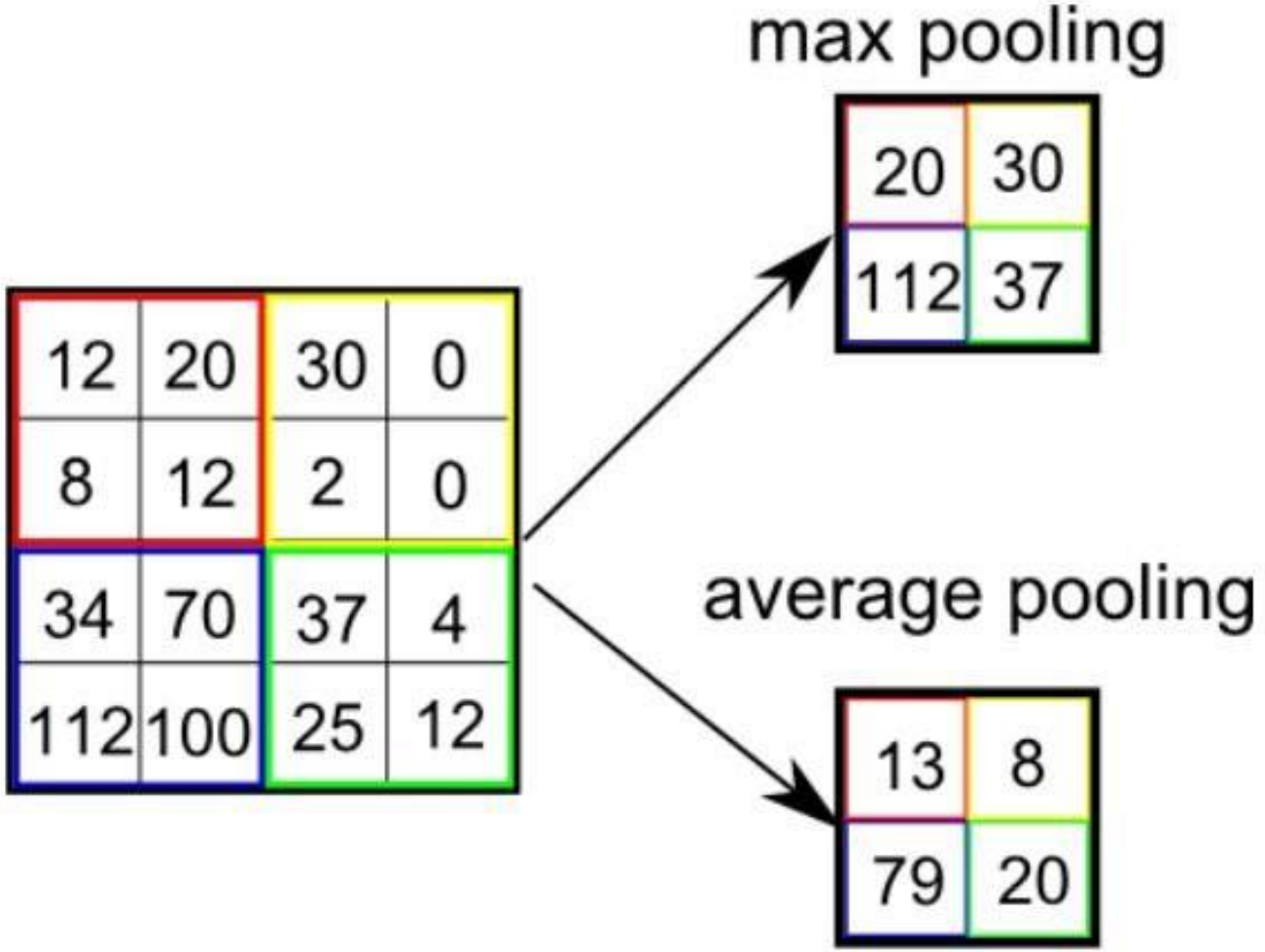
- F = 3 => zero pad with 1 pixel
- F = 5 => zero pad with 2 pixel
- F = 7 => zero pad with 3 pixel

Pooling Layer

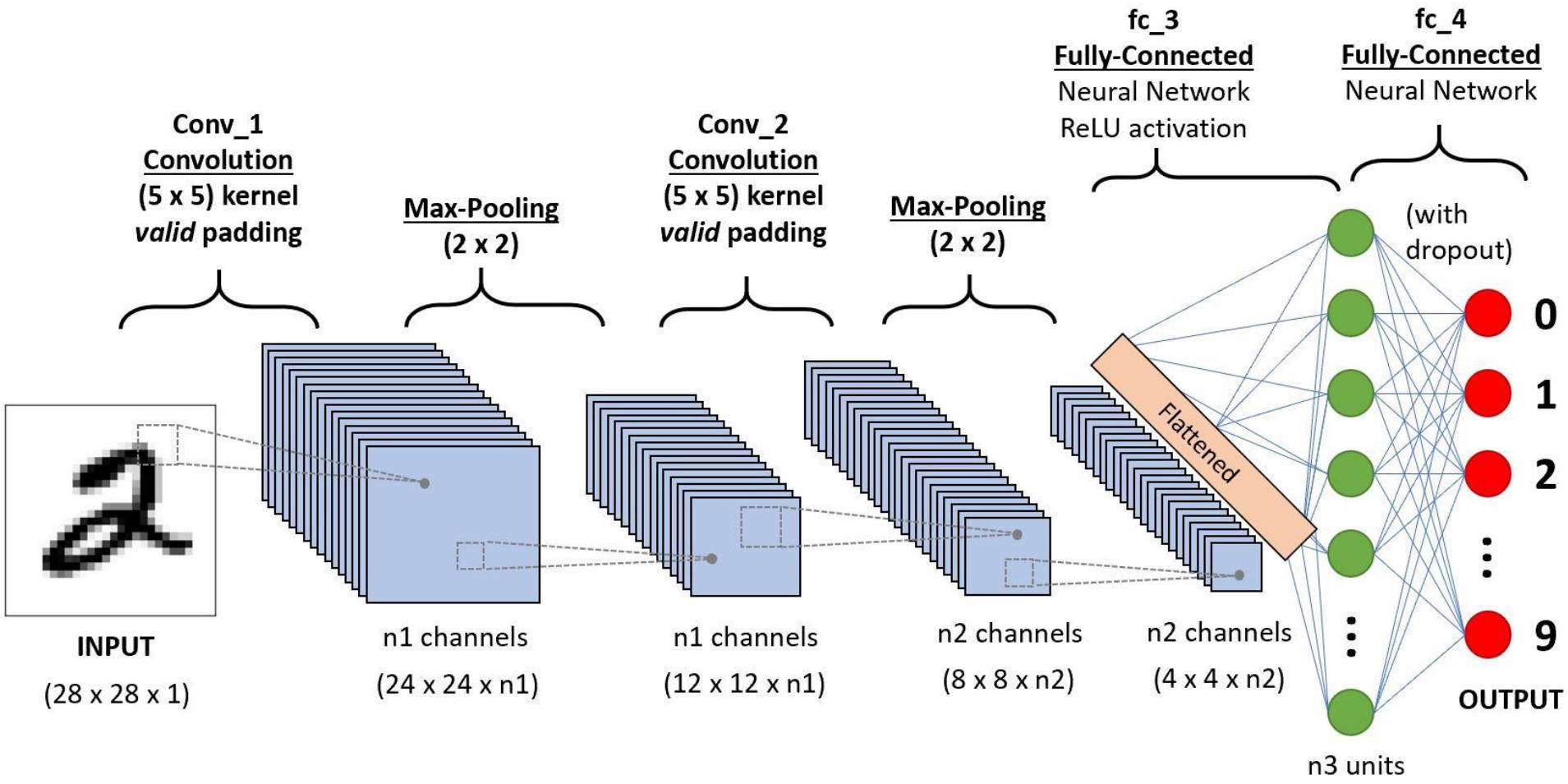
- Makes the representations smaller and more manageable
- Operates over each activation map independently



MaxPooling and AvgPoling



Example of CNN

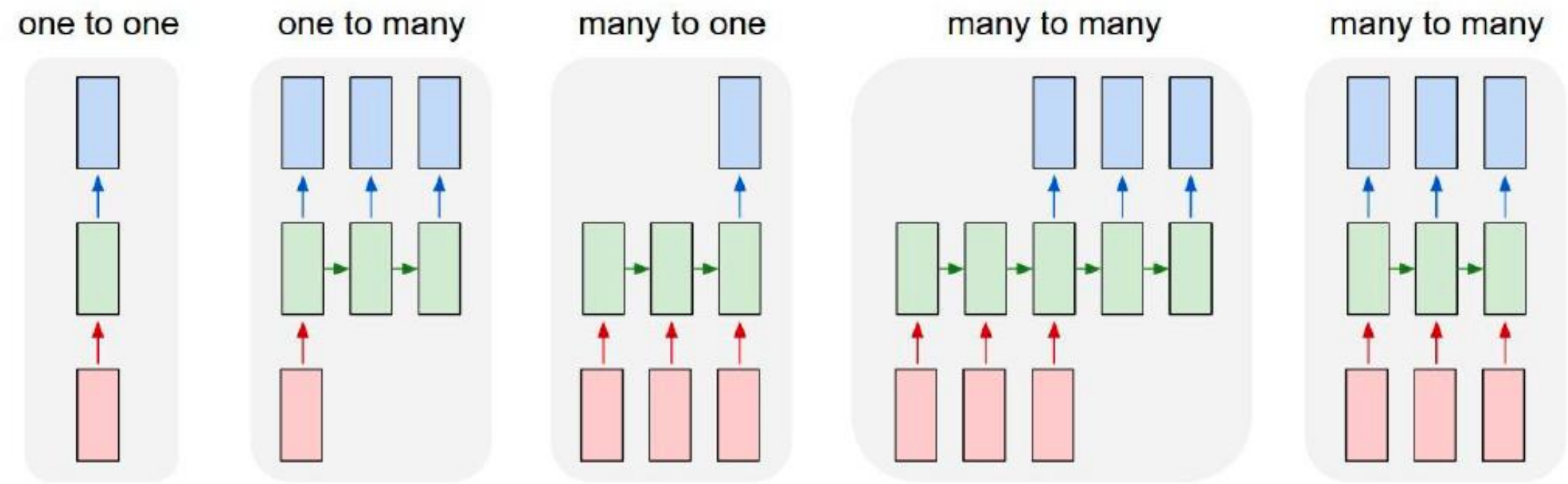


Recurrent Neural Network

Slides edited from Stanford

http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

Types of Recurrent Neural Networks



Vanilla NN

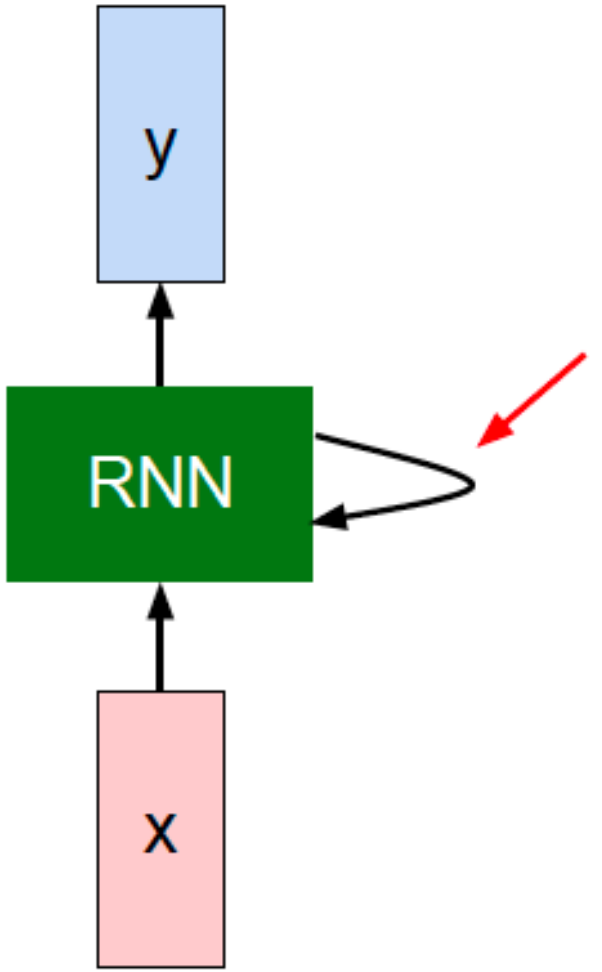
Image --> Sequence of Words
Image Captioning

Sequence of Words --> Sentiment
Sentiment Classification
TS Classification

Sequence of Words --> Sequence of Words
Machine Translation

Video Classification

Recurrent Neural Network - RNN



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Recurrent Neural Network - RNN

- We can process a sequence of vectors ***x*** by applying a ***recurrence formula*** at every time step:

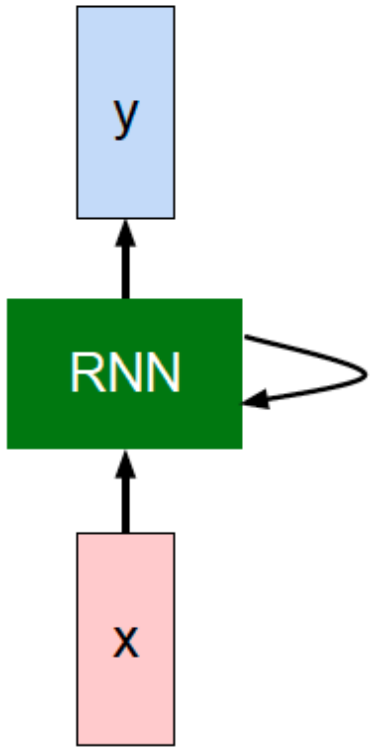
$$h_t = f_W(h_{t-1}, x_t)$$

new state

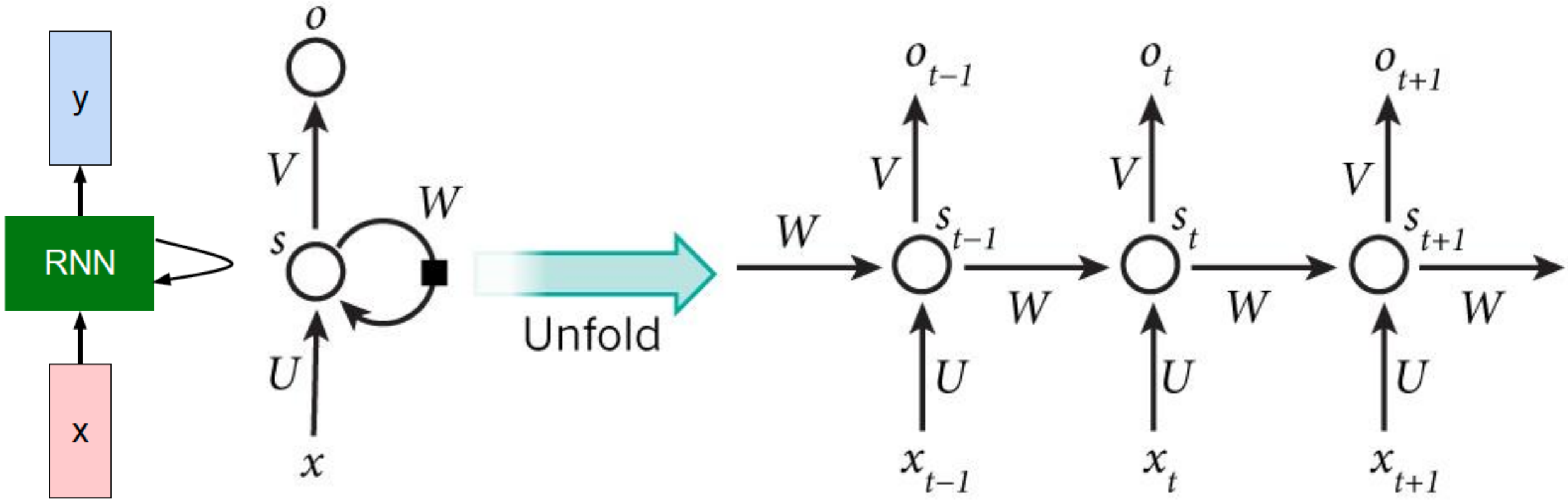
old state

input vector at some time step

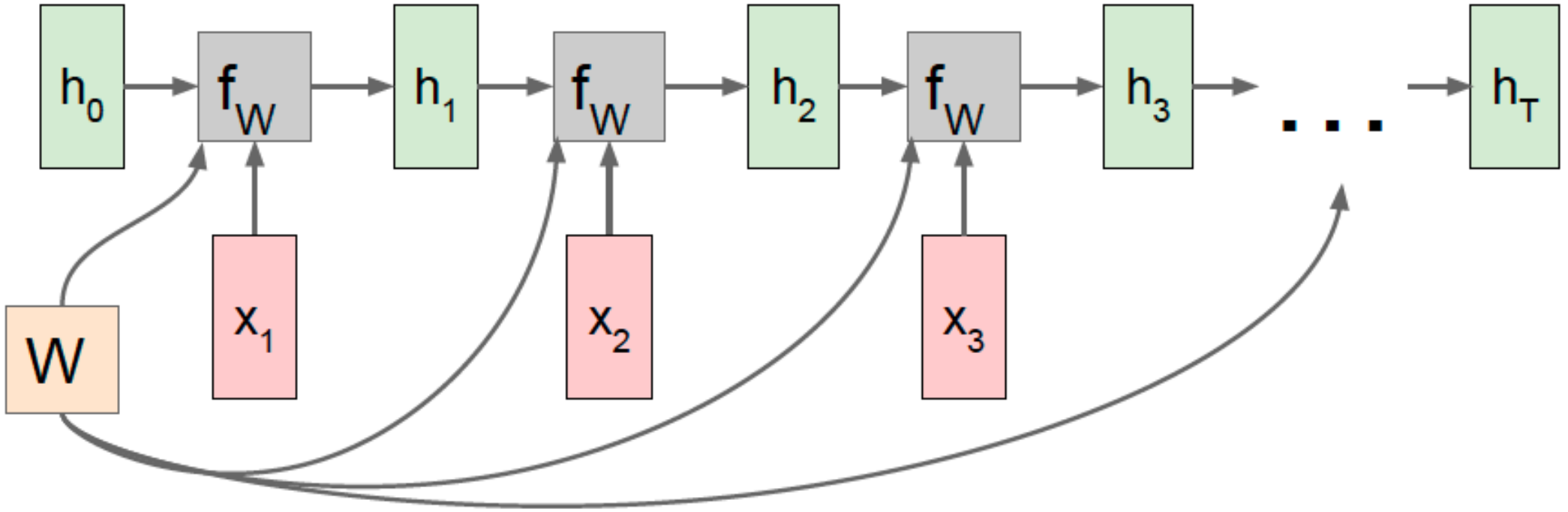
some function with parameters W



Unfolded RNN

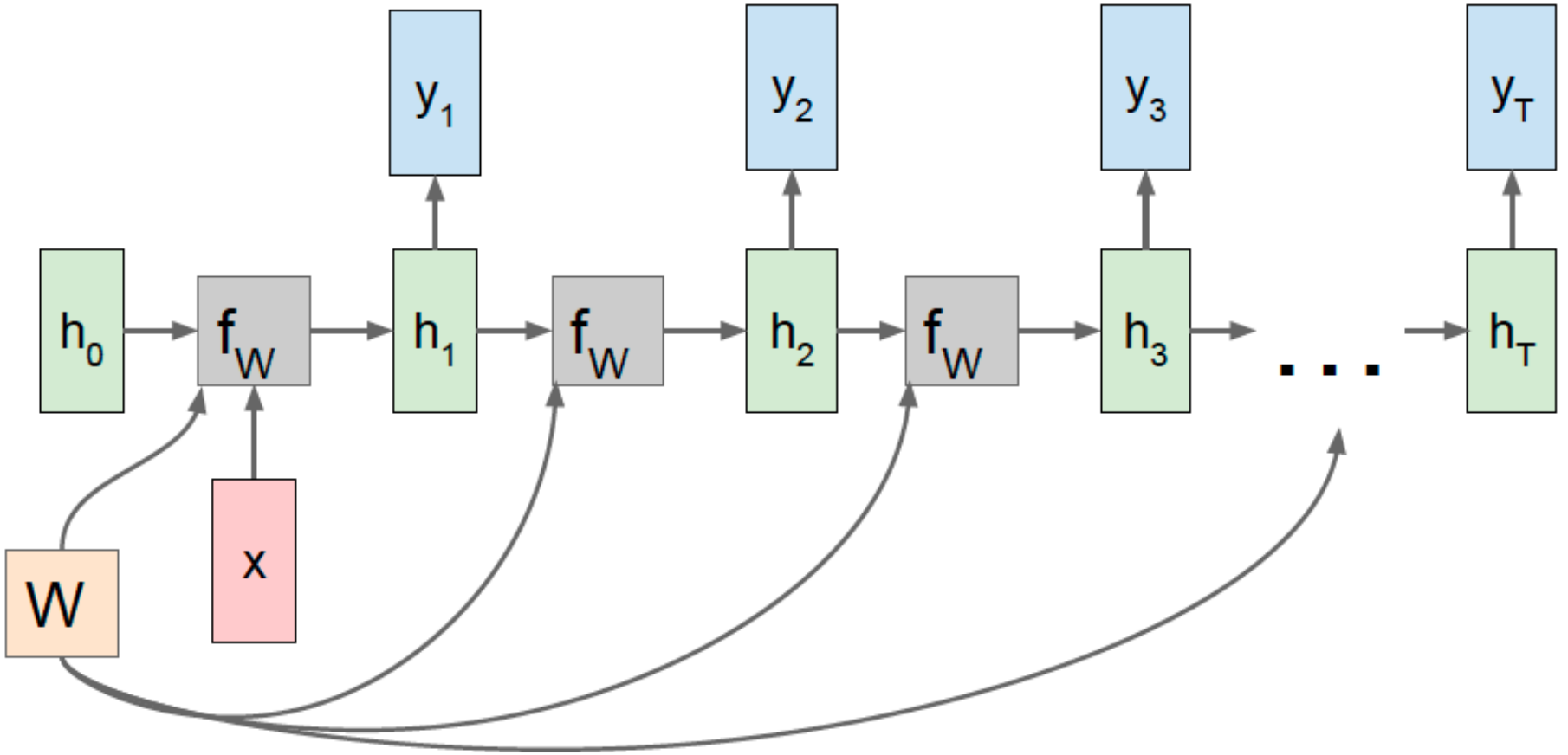


RNN: Computational Graph

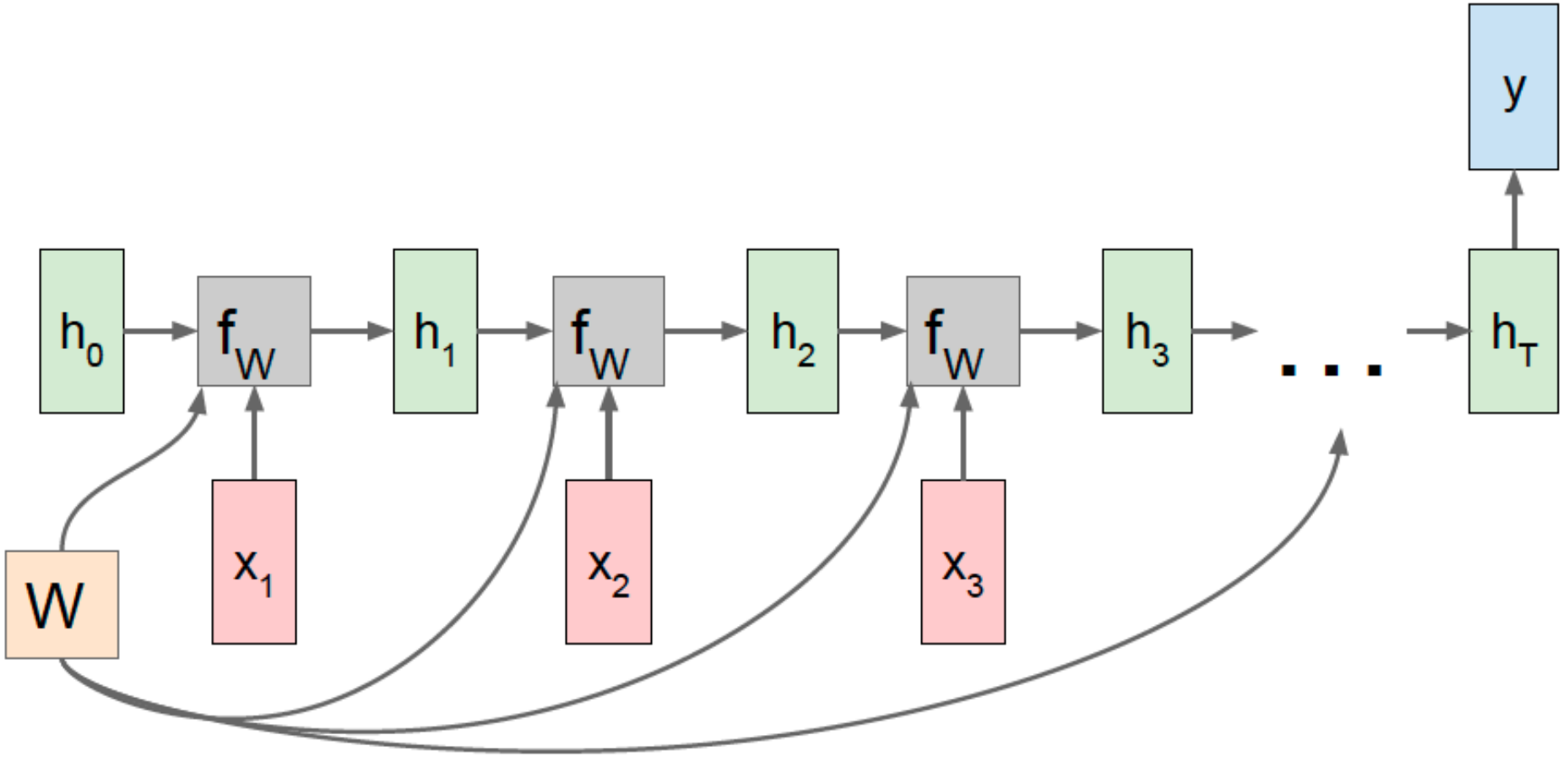


Reminder: Re-use the same weight matrix at every time-step

RNN: Computational Graph: Many to Many



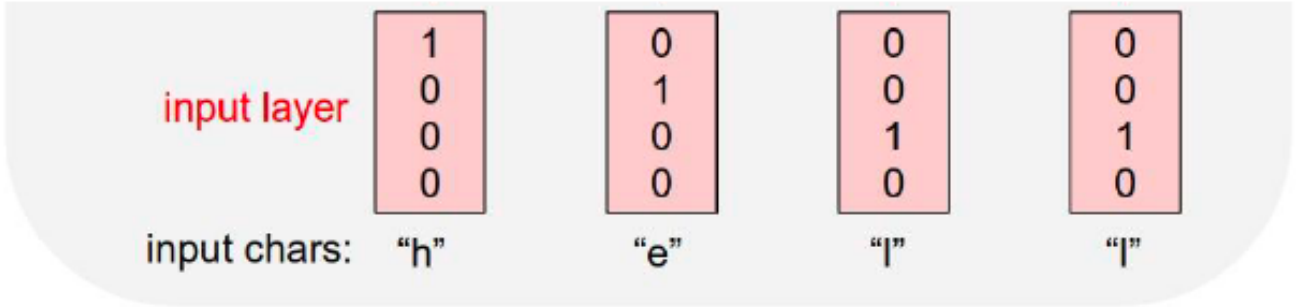
RNN: Computational Graph: Many to One



RNN: Example Training

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



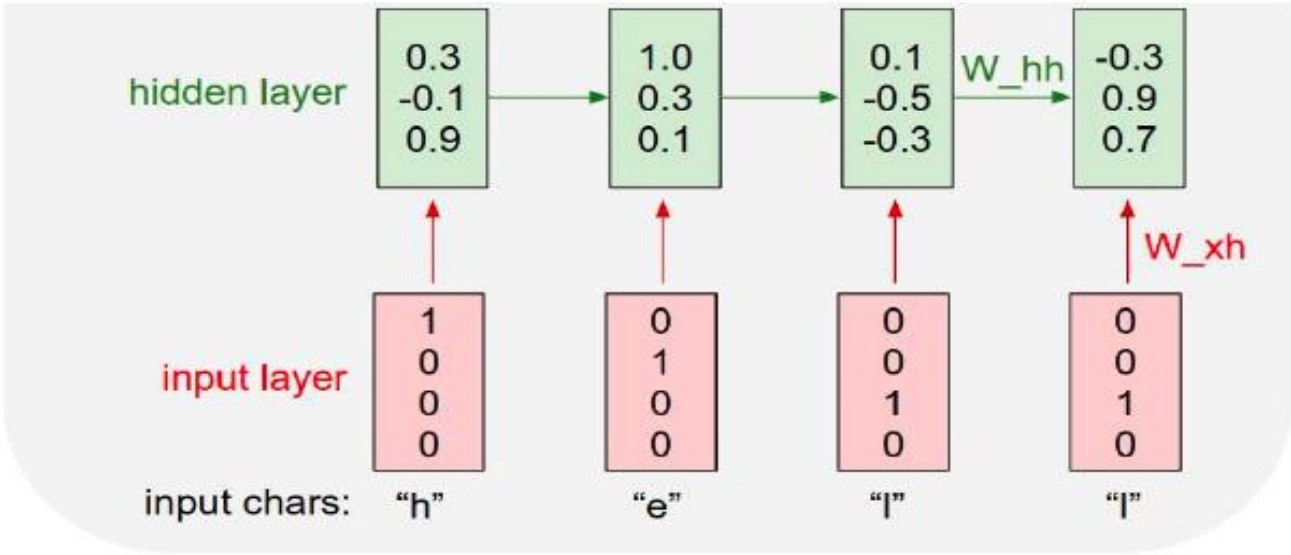
RNN: Example Training

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

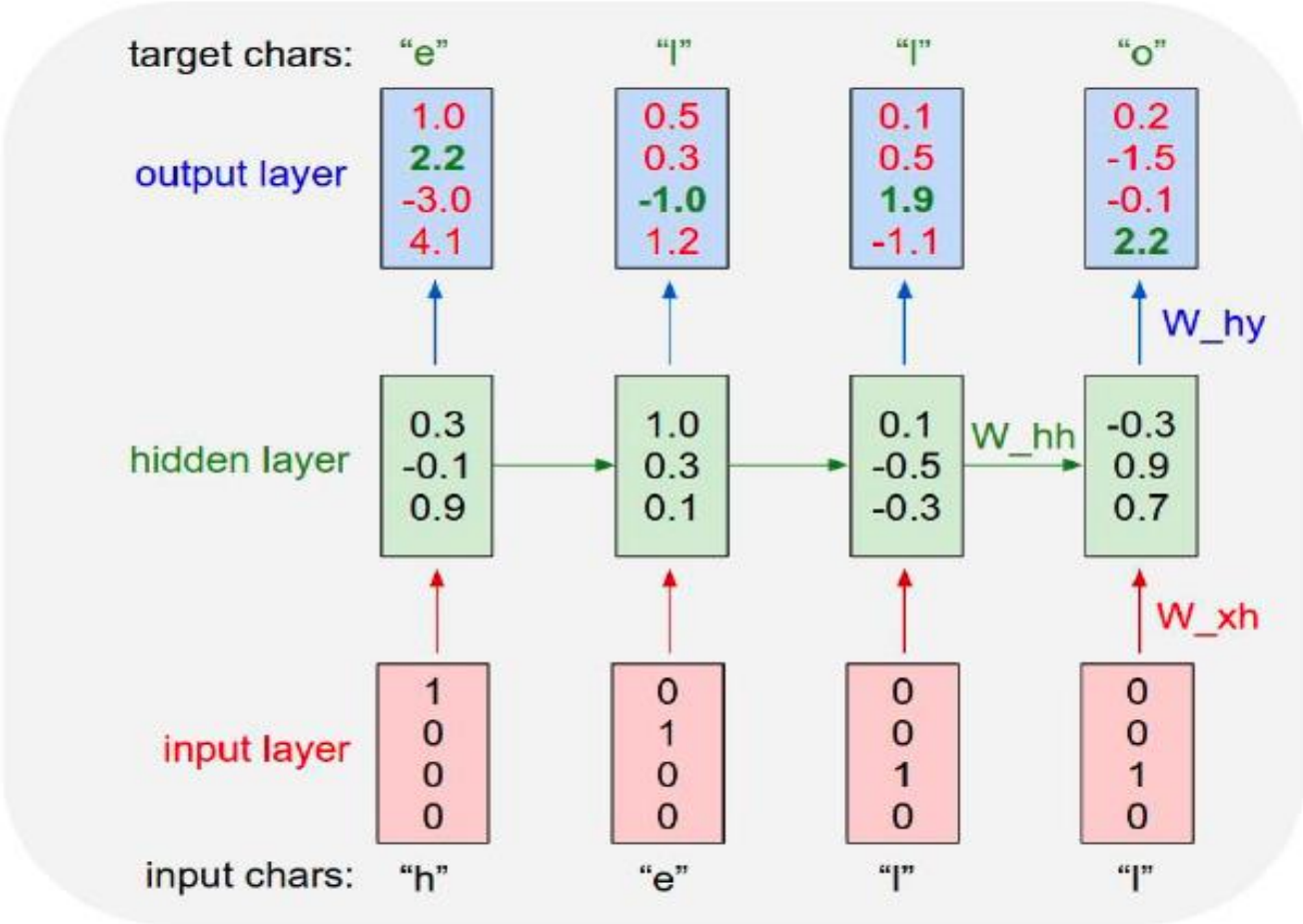


RNN: Example Training

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

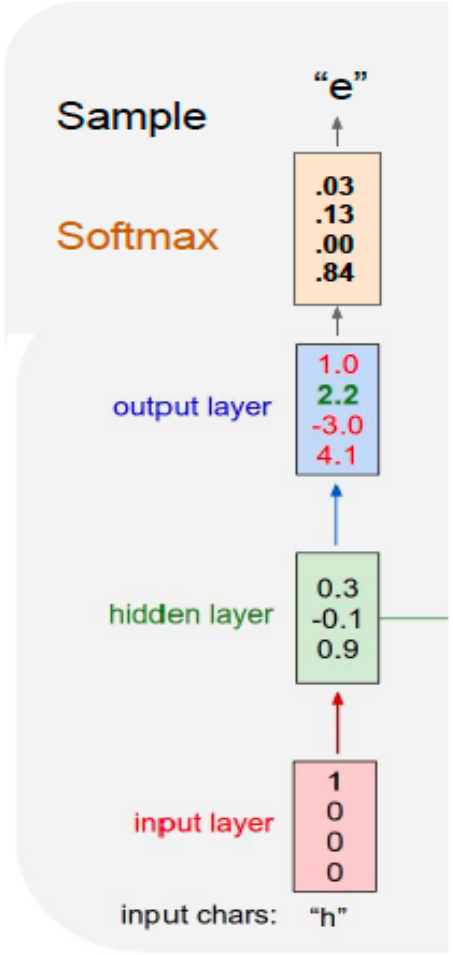


RNN: Example Test

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

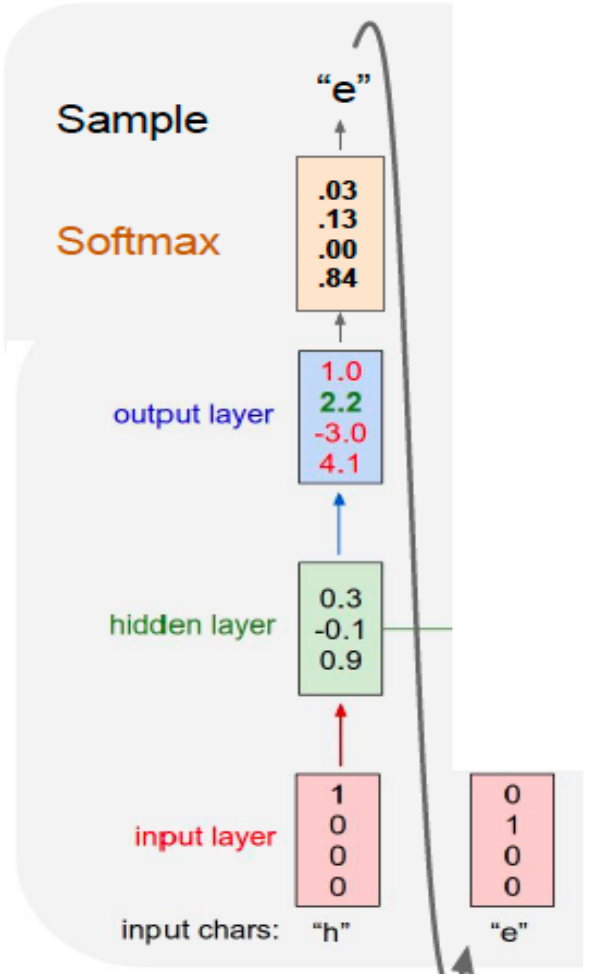


RNN: Example Test

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

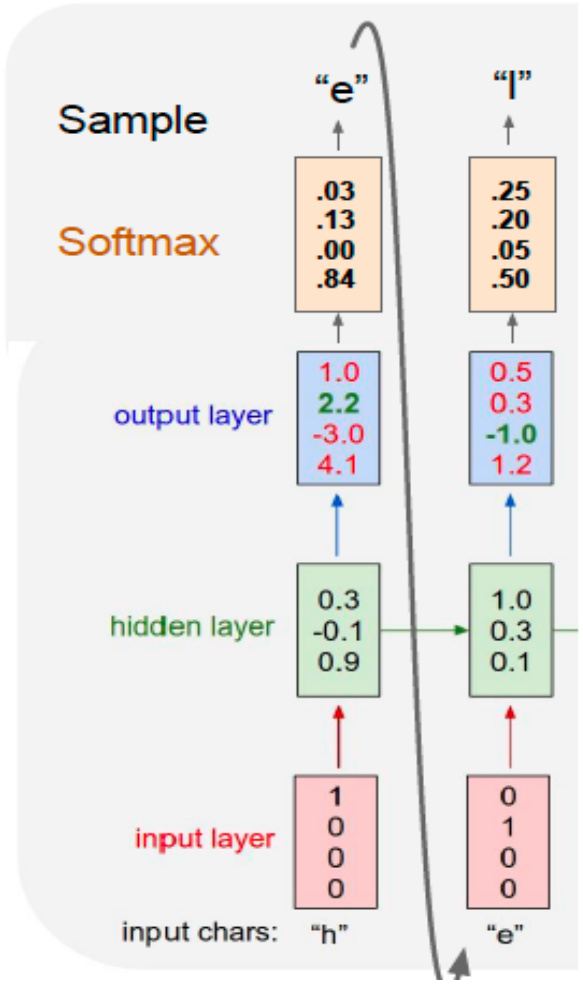


RNN: Example Test

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

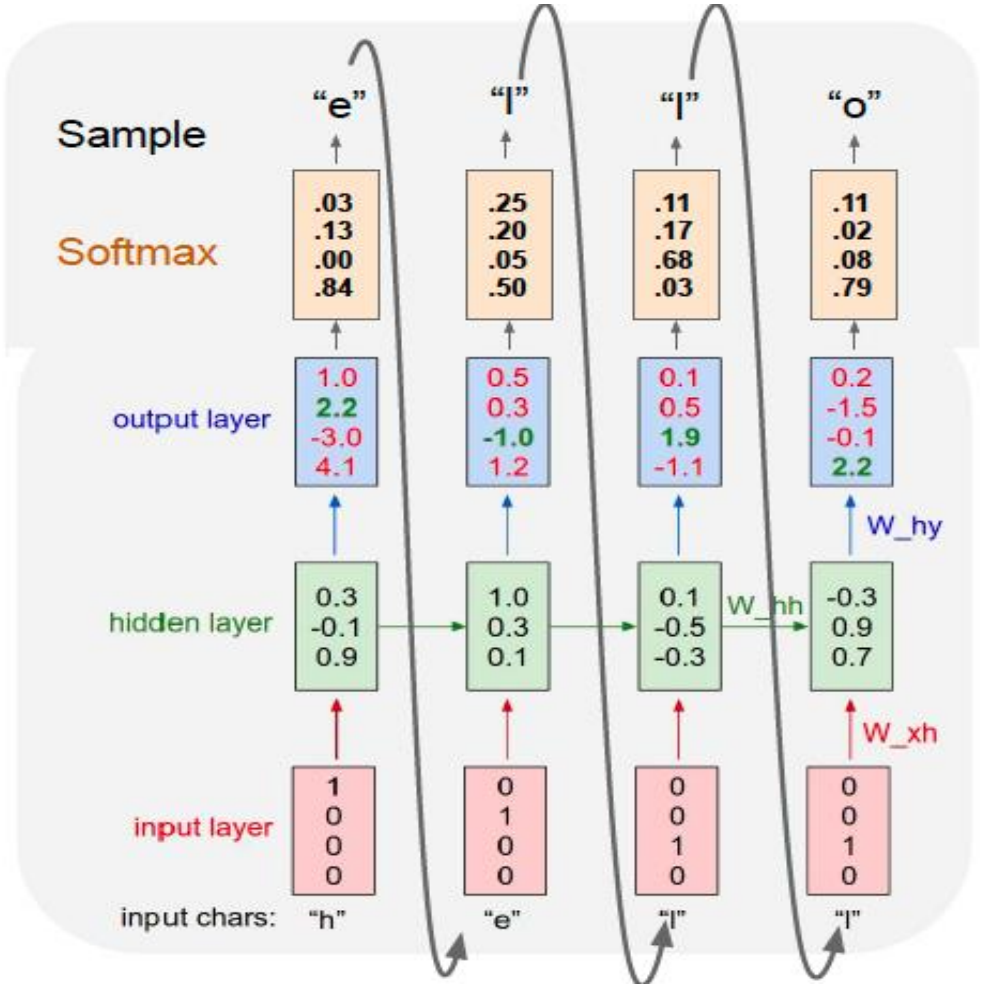


RNN: Example Test

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

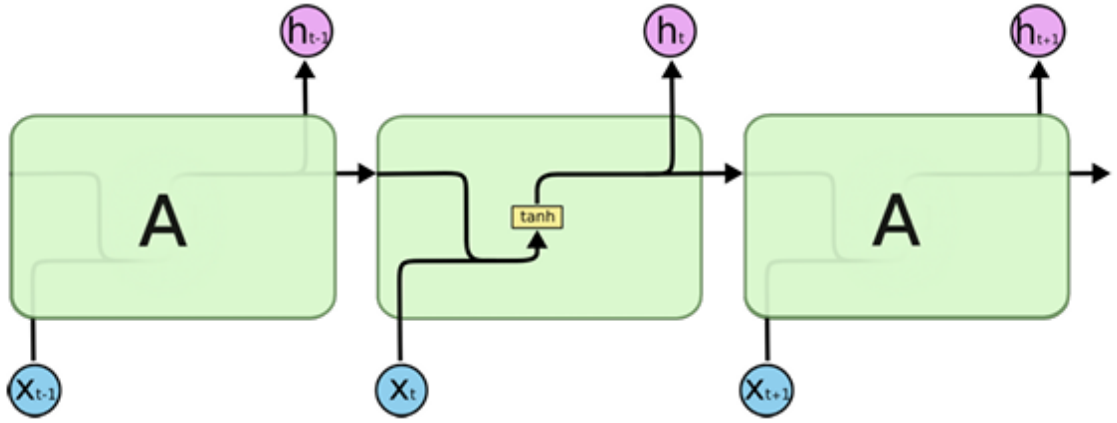


Problems with RNN

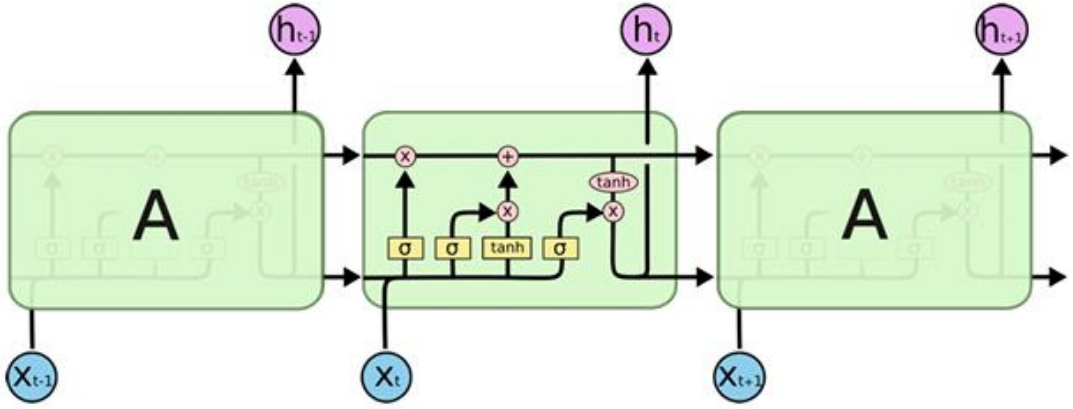
- RNN cannot handle situation where the gap between the relevant information and the point where it is needed is very large.

LSTM - Long Short Term Memory

- LSTM are a special RNN, capable of learning long-term dependencies.
- All RNN have the form of a chain of repeating modules of neural network with a single layer typically with tanh activation.
- LSTM have the same structure but different repeating module.



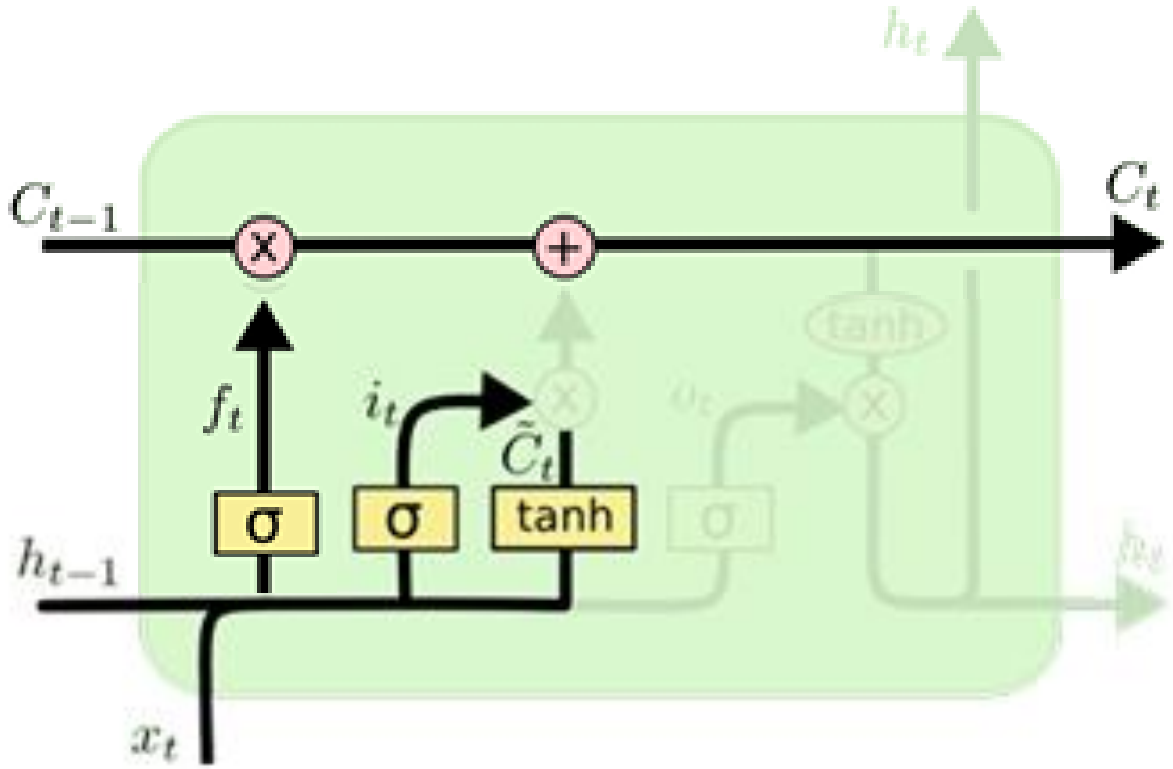
RNN



LSTM

LSTM Characteristics

- The core idea is the **cell state**.
- It can remove or add information to the cell state with gates.
- Gates are composed out of a sigmoid neural net layer and a pointwise multiplication operation to decide what information:
  - to throw away from the cell state
  - to store in the cell state

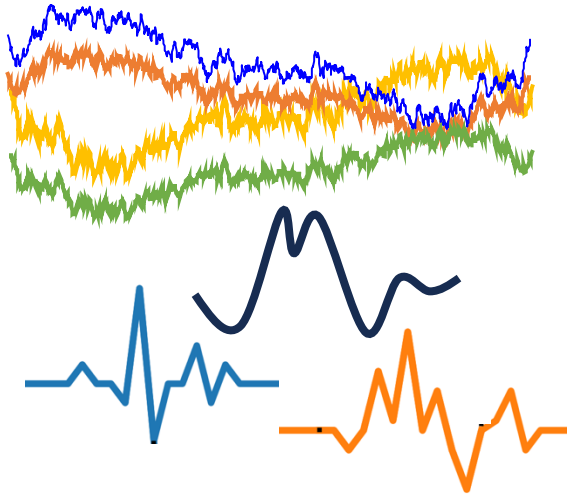


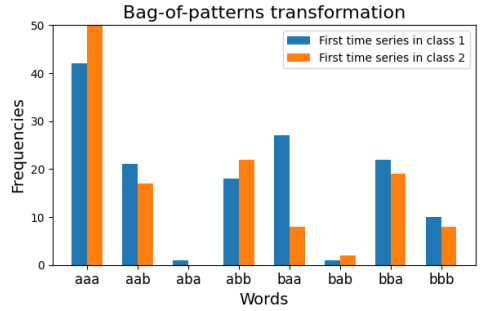
Deep-learning Models for TS

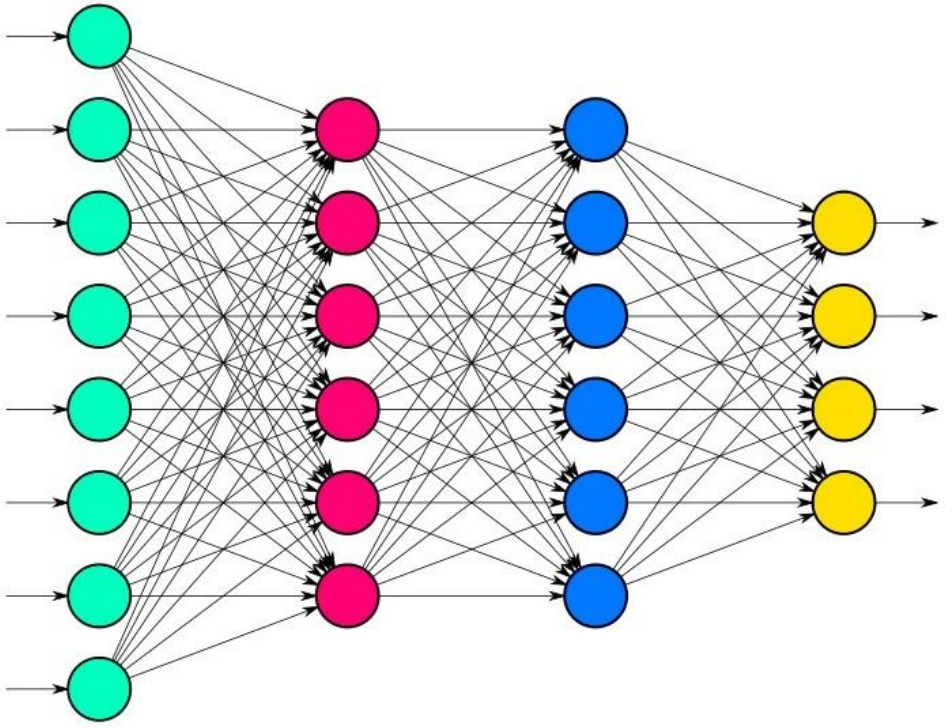
DNN for TS

DNN, i.e., Multilayer Neural Networks can be used as they are on any data representation discussed so far calculated on on any domain:

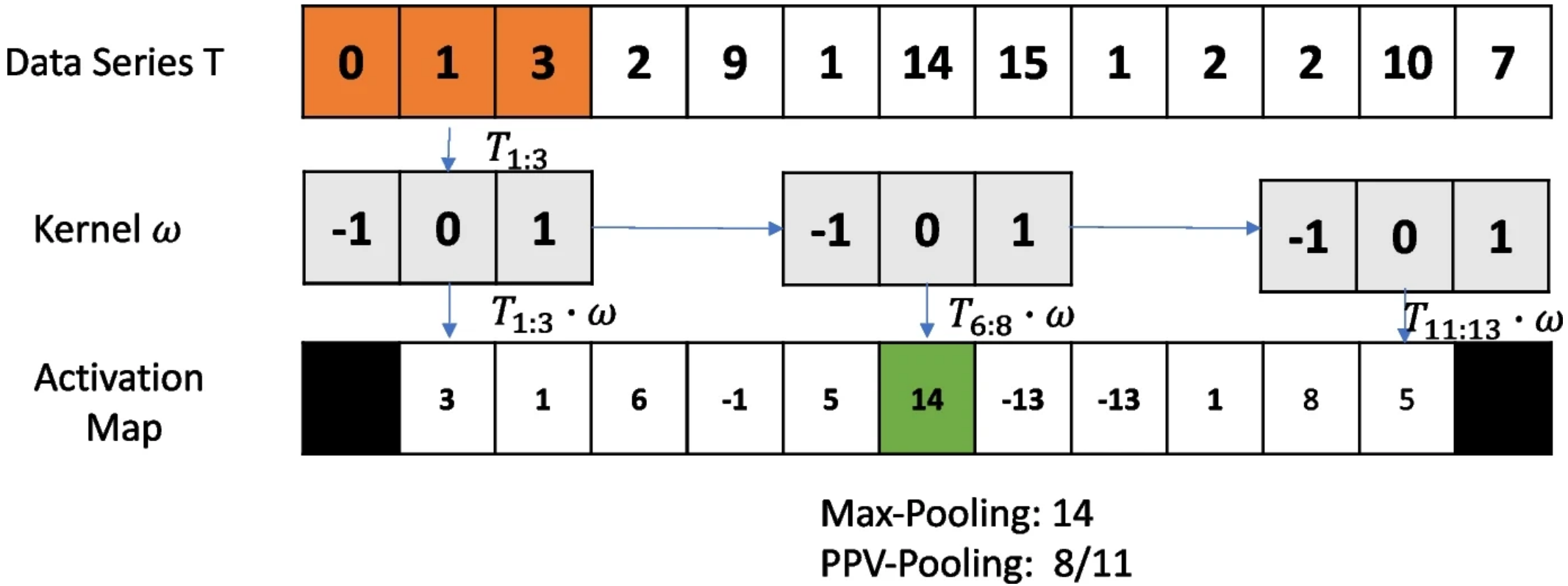
- Raw TS
- Shapelets
- Intervals
- Distances
- Bag of Patterns





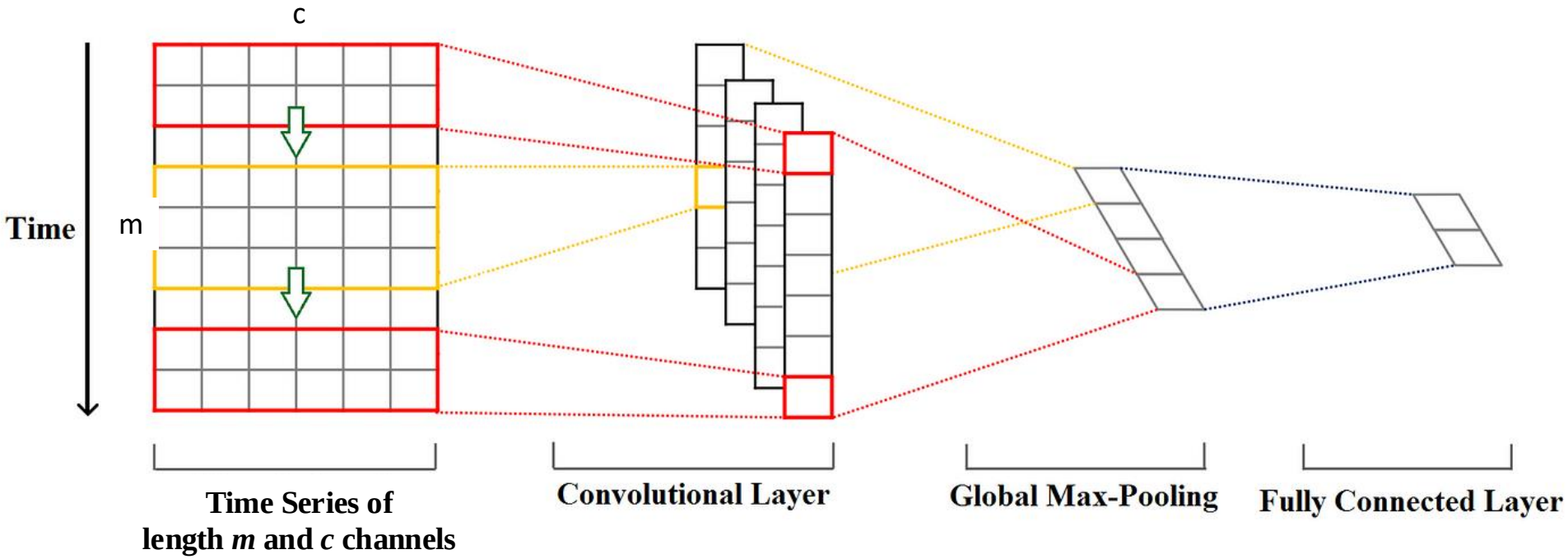


Convolutions for TS



CNN for TS

- Convolution and pooling operations are alternatively used to generate deep features of the raw data.
- Then the features are connected to a multilayer perceptron (MLP) to perform classification.

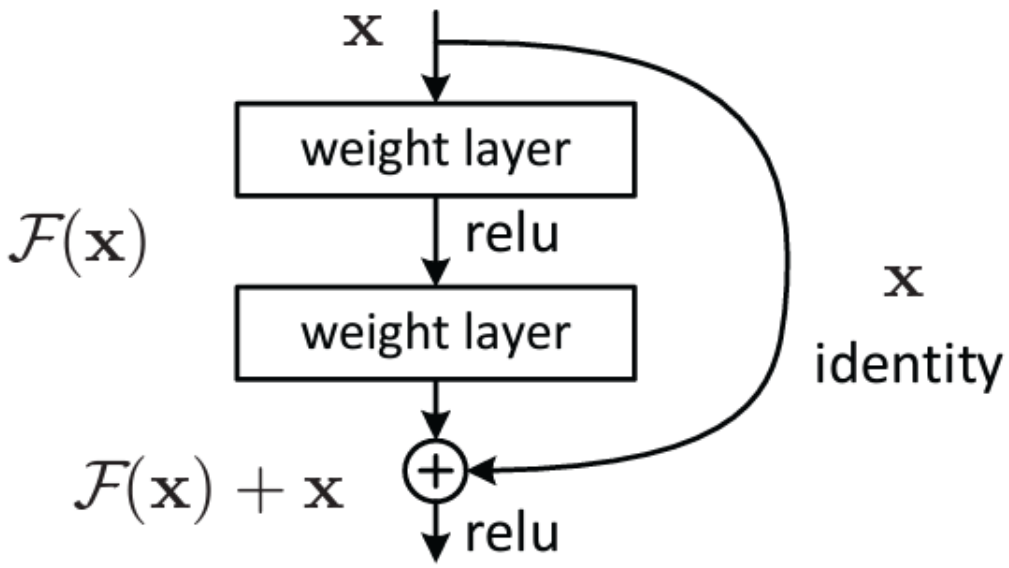


Problems with DNN

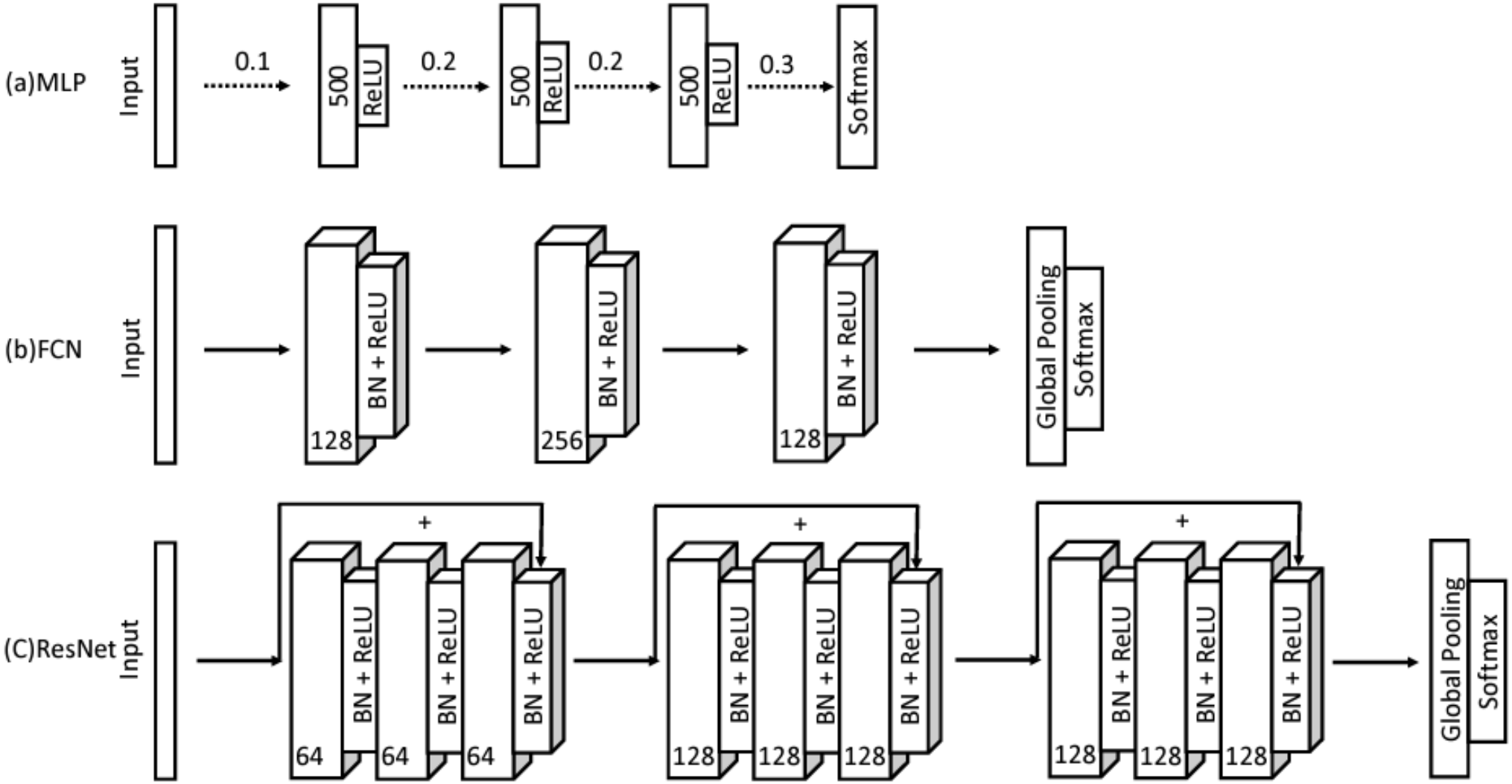
- When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly.
- Such degradation is not caused by overfitting, and adding more layers to a suitably DNN leads to higher training error.

ResNet

- Deep Residual Learning framework.
- Denoting the desired underlying mapping as H(x), we let the stacked nonlinear layers fit another mapping of F(x) = H(x) − x.
- The original mapping is recast into F(x) + x.
- F(x) + x can be realized with a **shortcut connection**, i.e., skipping one or more layers by simply performing identity mapping and adding the outputs of the stacked layers.

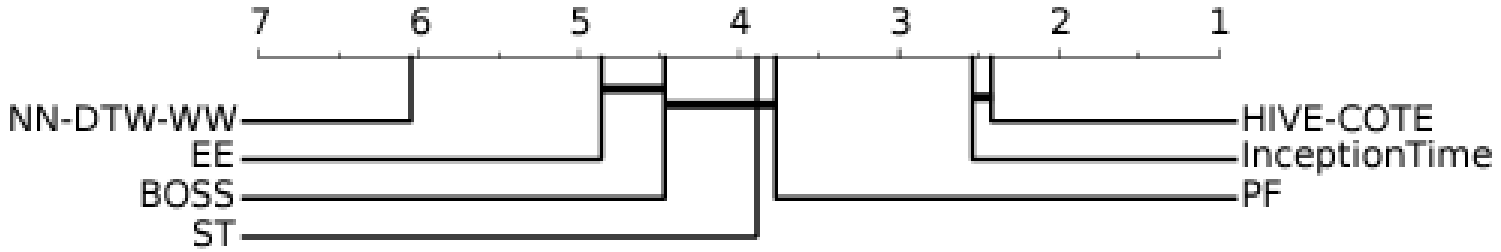


MLP vs CNN vs ResNet

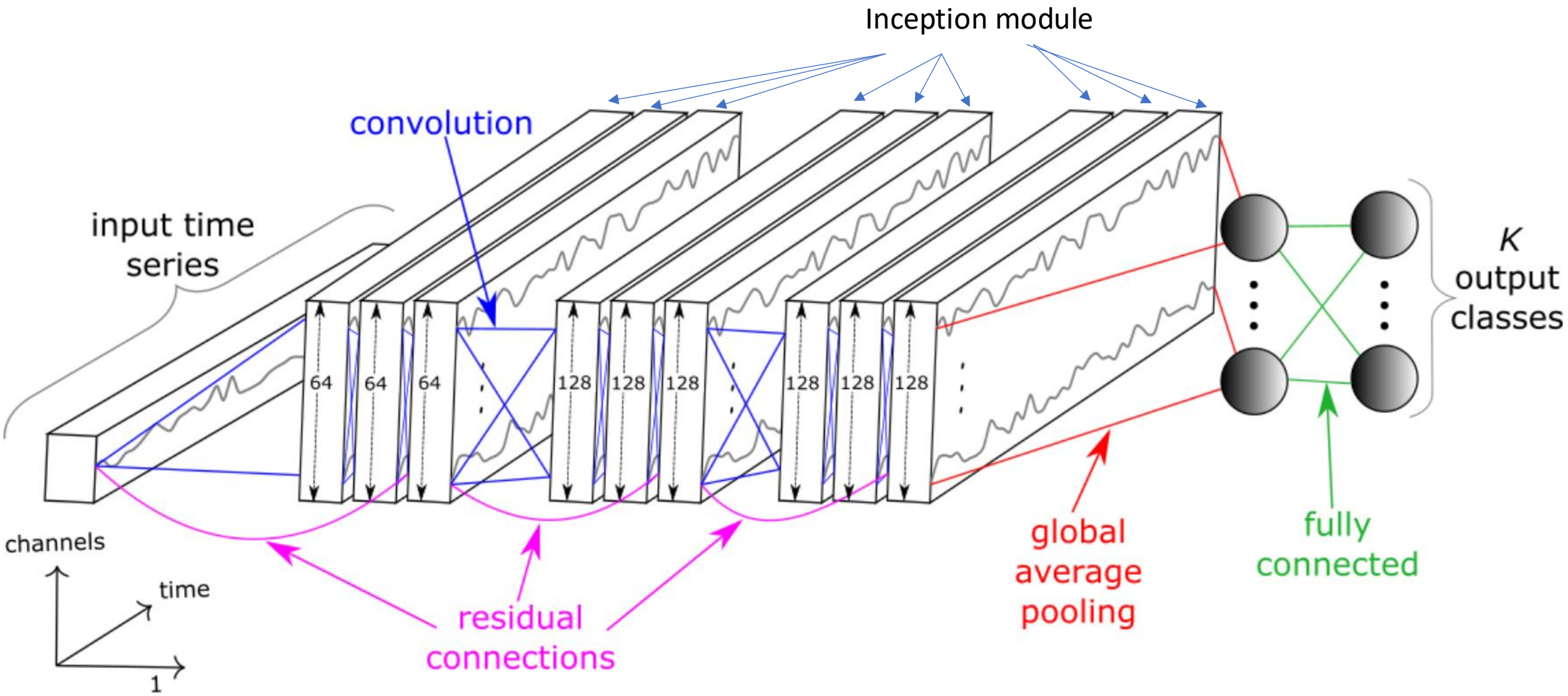


InceptionTime

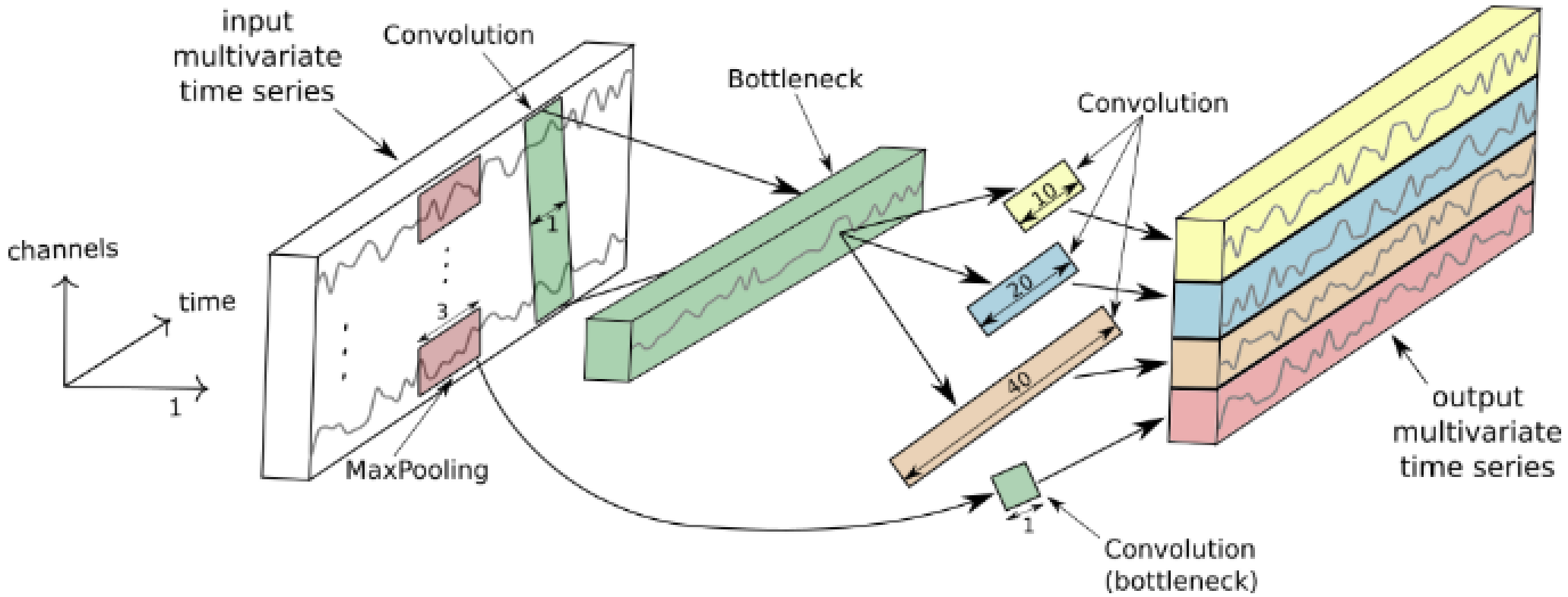
- Ensemble of CNN consisting of five Inception Networks.
- For each Inception Network:
  - 3 Inception Modules (6 blocks by default)
  - Global Averaging Pooling
  - Fully-Connected layer with the softmax activation function.
- Each Inception module consists of convolutions with kernels of several sizes followed by batch normalization and RELU activation function.



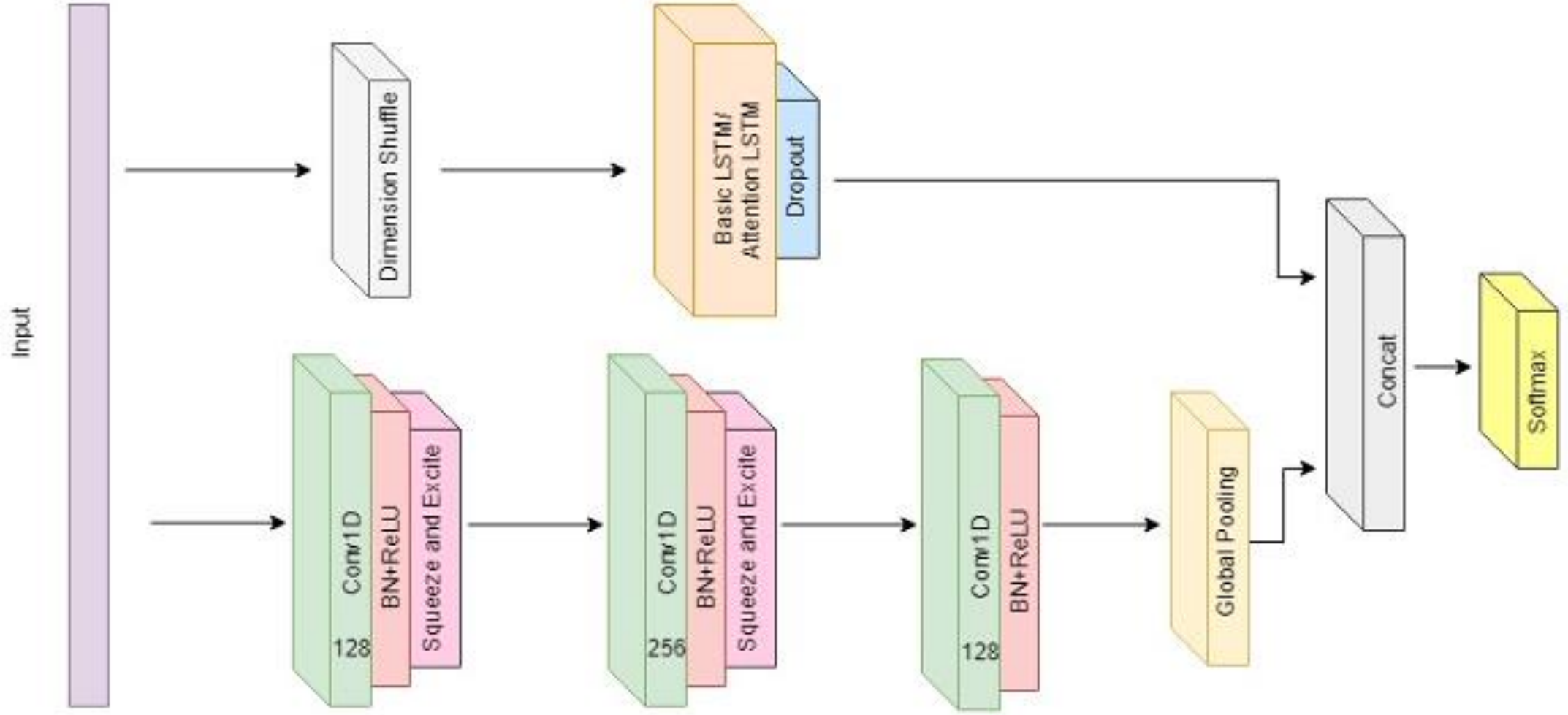
Inception Network



Inception Module



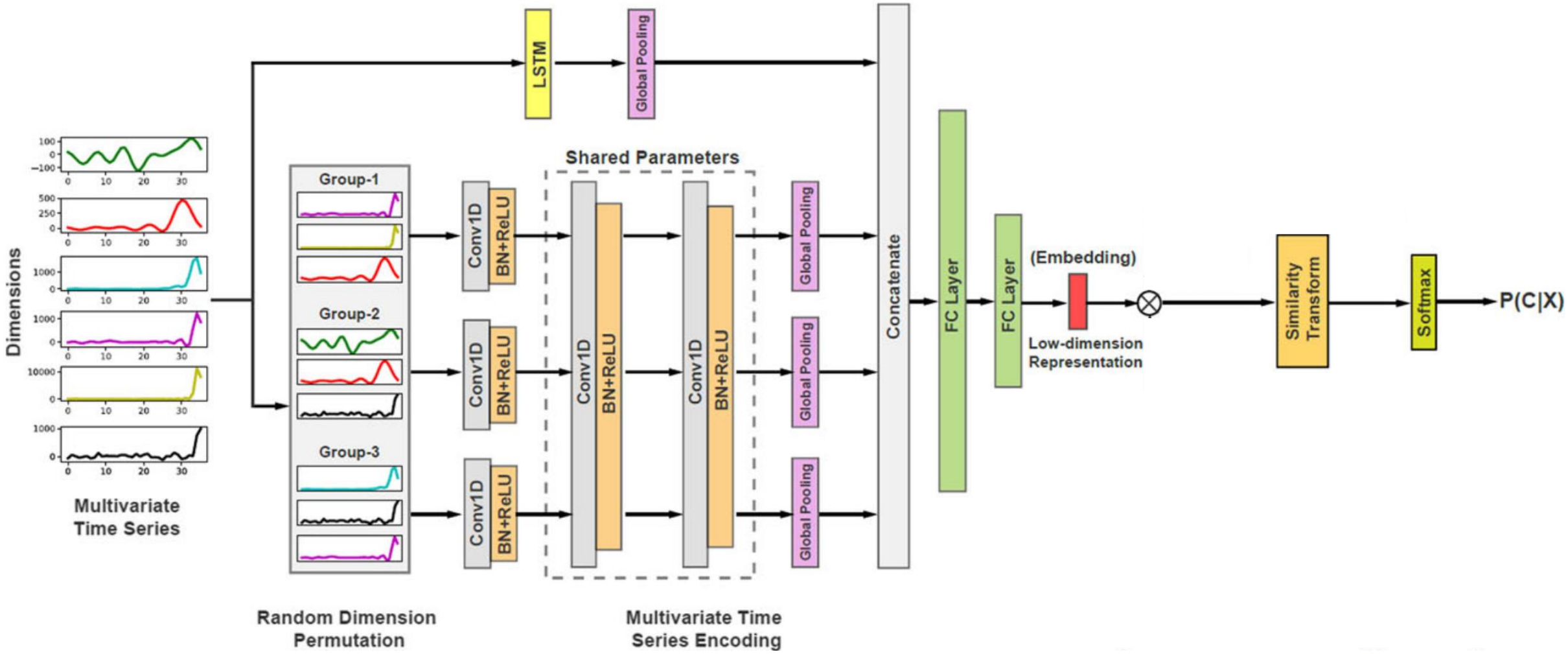
Multivariate LSTM-FCN



TapNet

- Draws on the strengths of both traditional and deep learning approaches:
  - **Deep learning Approaches:** excel at learning low dimensional features without the need for embedded domain knowledge, whereas
  - **Traditional Approaches:** work well on small datasets.
- Three distinct modules:
  - Random Dimension Permutation: produce groups of randomly selected dimensions with the intention of increasing the likelihood of learning how combinations of dimension values effect class value.
  - Multivariate Time Series Encoding:
    - 3 sets of 1d Convolutional layers followed by Batch Normalisation
    - Raw data is also passed through an LSTM and Global Pooling Layer
  - Attentional Prototype Learning: used for unlabelled data

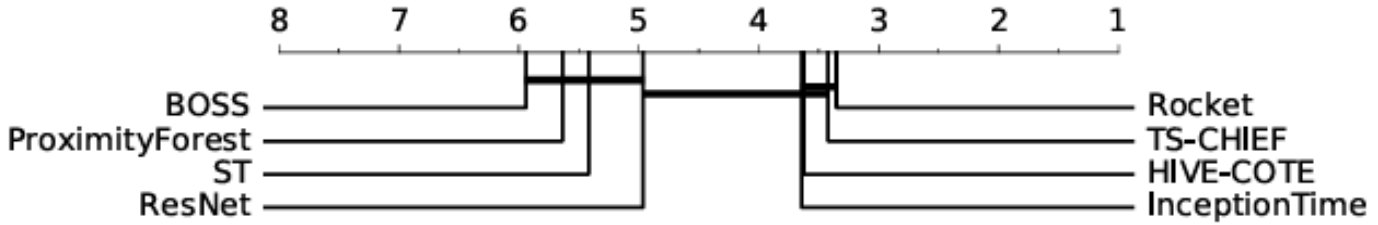
TapNet


LSTM
Global Pooling
Shared Parameters
Group-1
Group-2
Group-3
Conv1D
BN+ReLU
Conv1D
BN+ReLU
Conv1D
BN+ReLU
Global Pooling
Global Pooling
Global Pooling
Concatenate
FC Layer
FC Layer
(Embedding)
Low-dimension Representation
Similarity Transform
Softmax
P(C|X)
Dimensions
Multivariate Time Series
Random Dimension Permutation
Multivariate Time Series Encoding

Kernel-based Models

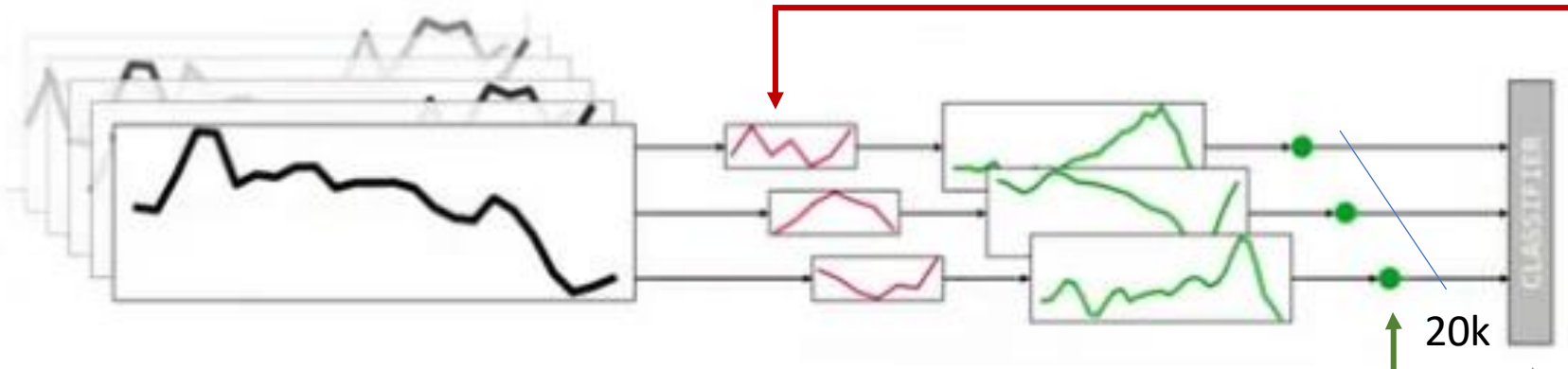
ROCKET - RandOm Convolutional KErnel Transform

- ROCEKT transforms TS using random convolutional kernels.
- Then uses the transformed features to train a linear classifier.
- It is accurate, fast and scalable.
- Much faster than other methos of comparable accuracy.



1 h 40 m (ROCKET) ≪ 6 days (InceptionTime) < 11 days (TS-CHIEF)

ROCKET - Core Aspects

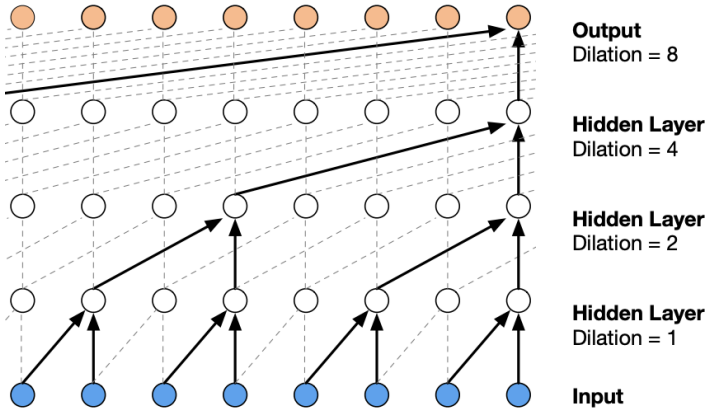


- 10k Random Convolutional Kernels with Dilation
- Two Pooling Strategies:
  - Max Value
  - PPV - Proportion of Positive Values, how much of the input matches the kernel
- Predictive Model:
  - Ridge Regression
  - Logistic Regression + SGD

Produces 20k Features

ROCKET vs. CNN

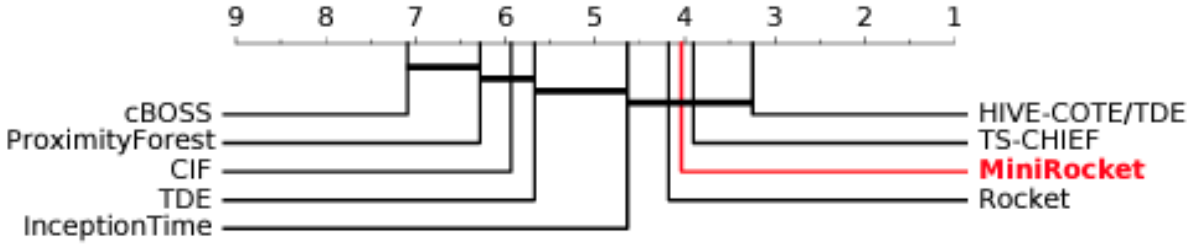
- CNNs use trainable kernels optimized by SGD to find patterns in the input data.
- ROCKET uses a single layer containing a very large number of random kernels.
- ROCKET uses a large variety of kernels: each kernel has random length, dilation, and padding, weights and biases.
- In CNNs kernel dilation increases exponentially with depth.
- ROCKET sample dilation randomly for each kernel.
- CNN uses Global Max Pooling
- ROCKET uses the Max value and the PPV.
- CNN hyperparameters are learnign rates, and network architecture
- ROCKET only hyperparameter is the number of kernels that handles the trade-off between classification accuracy and computation time.



MINIROCKET - MINImally ROCKET

- Like ROCKET, MINIROCKET transforms input TS using convolutional kernels, and uses the transformed features to train a linear classifier.
- MINIROCKET maintains dilation and PPV.
- Unlike ROCKET, MINIROCKET uses a small, fixed set of kernels, dose not use Max value pooling, and is almost entirely deterministic.
- MINIROCKET is up to 75 times faster than ROCKET

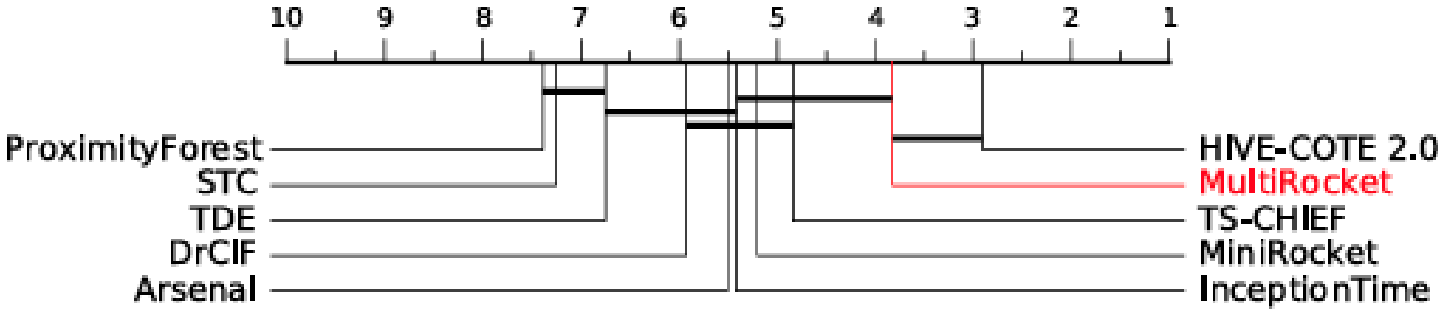
	ROCKET	MINIROCKET
length	$\{7, 9, 11\}$	9
weights	$\mathcal{N}(0, 1)$	$\{-1, 2\}$
bias	$\mathcal{U}(-1, 1)$	from convolution output
dilation	random	fixed (rel. to input length)
padding	random	fixed
features	PPV + max	PPV
num. features	20K	10K



MultiROCKET

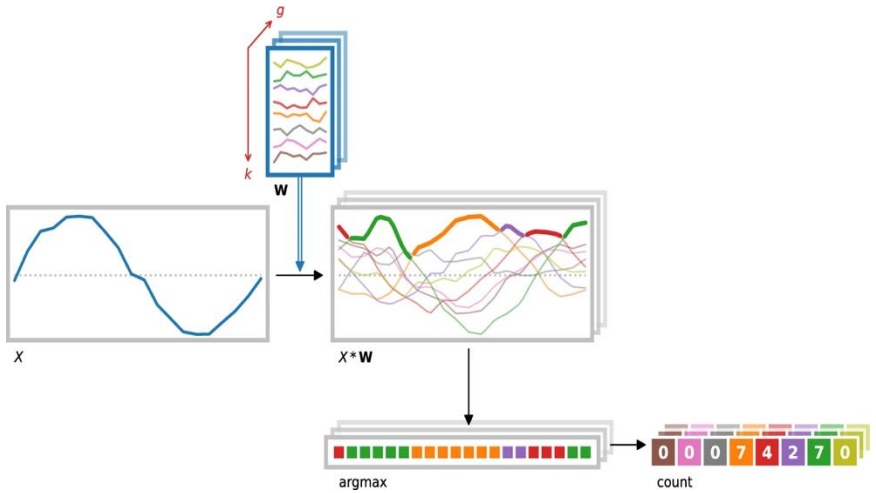
"Multi" does not refer to Multivariate TS

- MultiROCKET transforms a TS into its first order difference.
- Then both the original and the first order difference TS are convolved with the 84 MINIROCKET kernels.
- A different set of dilations and biases is used for each representation because both representations have different and range of
- Besides PPV, MultiROCKET adds 3 additional pooling operators
- By default, MultiROCKET produces approximately 50,000 features per TS.
- The transformed features are used to train a linear classifier.

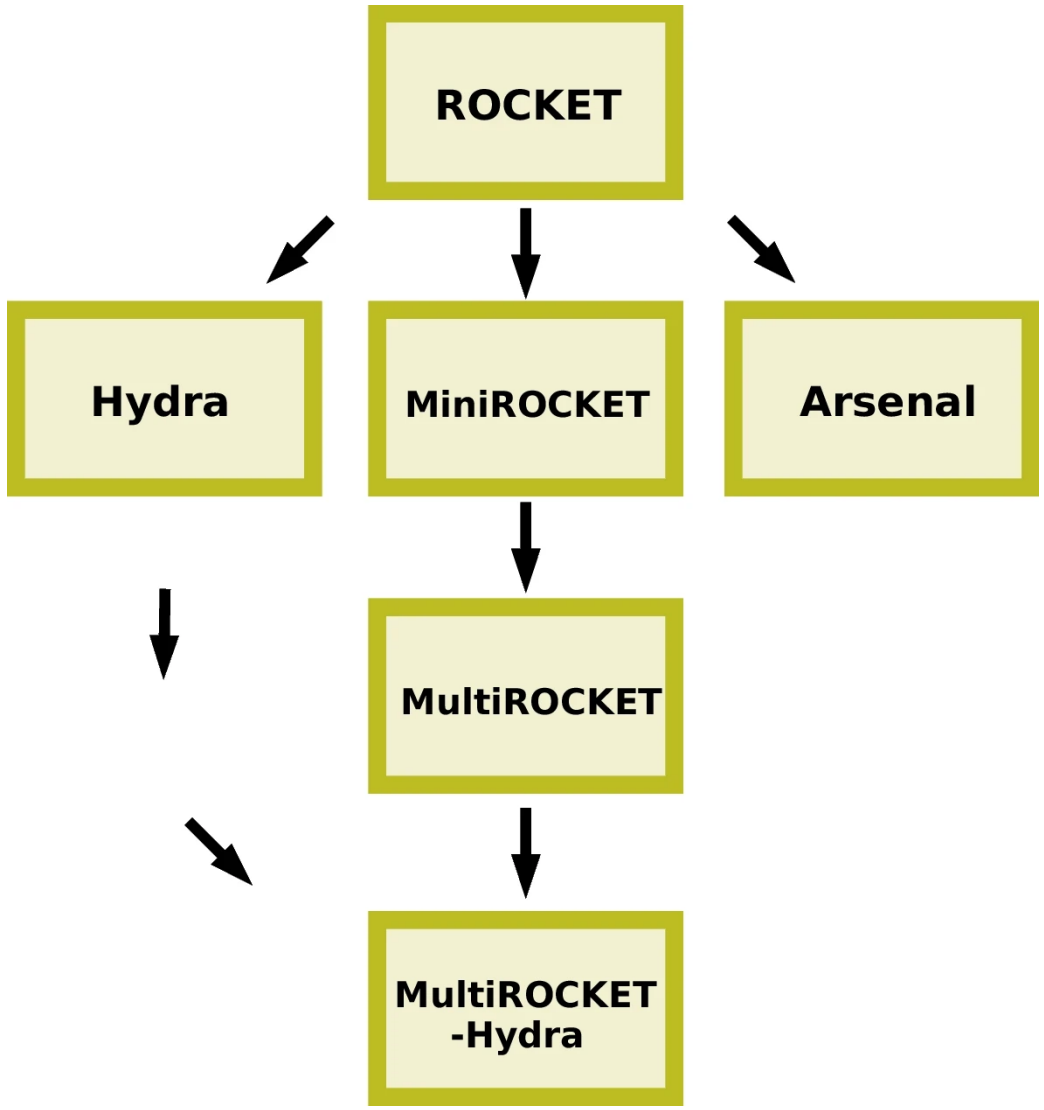


Hydra & MultiROCKET-Hydra

- Hydra: HYbrid Dictionary-ROCKET Architecture combines dictionary-based and convolution-based models.
- It starts with *g* groups of *k* random convolutional kernels each to calculate the activation of time series.
- In each group, is calculated the activation of a kernel with the time series and it is recorded how frequently this kernel is the best match (counts the highest activation).
- This results in a *k*-dimensional count vector for each of the *g* groups, resulting in a total of *g x k* features. Default *g = 64* and *k = 8.*
- Hydra is applied to both the time series and its first-order differences
- MultiROCKET-Hydra concatenates features from MultiROCKET and Hydra.



Overview of Kernel-based Models and Relationships



Hybrid Models

HIVE-COTE - Hierarchical Vote Collective of Transformation-based Ensembles

- Heterogeneous meta ensemble for TSC.
- Five ensembles working on features from four different data transformation:

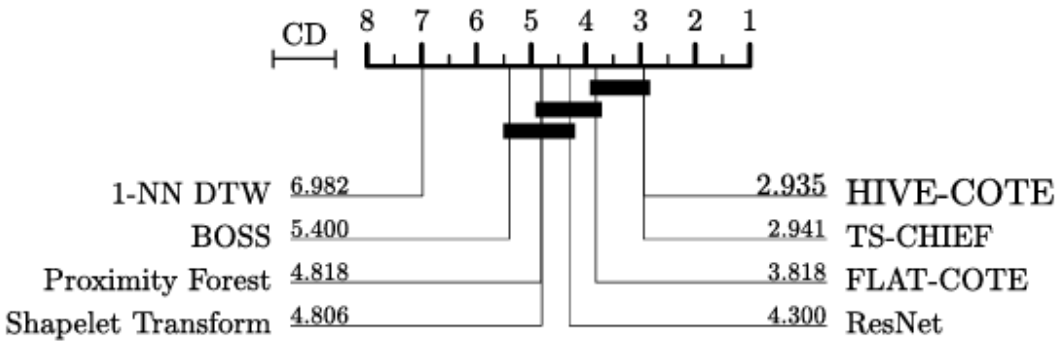
- Elastic Ensemble
- Shapelet Transform Classifier
- Time Series Forest
- Bag of Symbolic-Fourier-Approximation Symbols
- Random Interval Spectral Ensemble

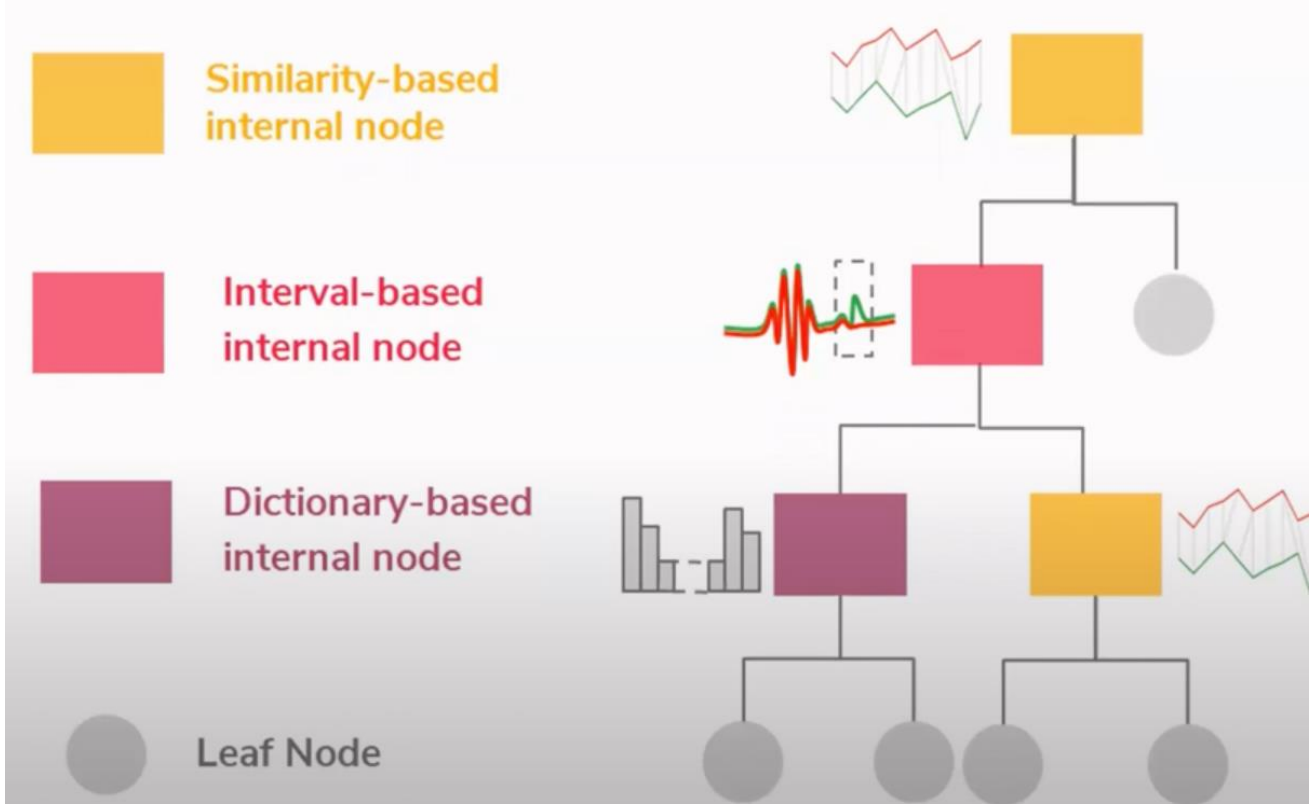
- Raw TS
- Shapelet-Tranformed TS
- Autocorrelation Features
- Power Spectrum Features

- Each ensemble is trained on the train data independently of the others.
- For new data, each ensemble passes an estimate of class probabilities to the control unit, which combines them to form a single prediction.
- It does this by weighting the probabilities of each module by an estimate of its testing accuracy formed from the training data.

TS-CHIEF - Time Series Combination of Heterogeneous and Integrated Embeddings Forest

- Tree-based ensemble for TSC using heterogeneous splits.
- Extends the Proximity Forest with trees considering splits w.r.t. dictionary-based and interval-based features.

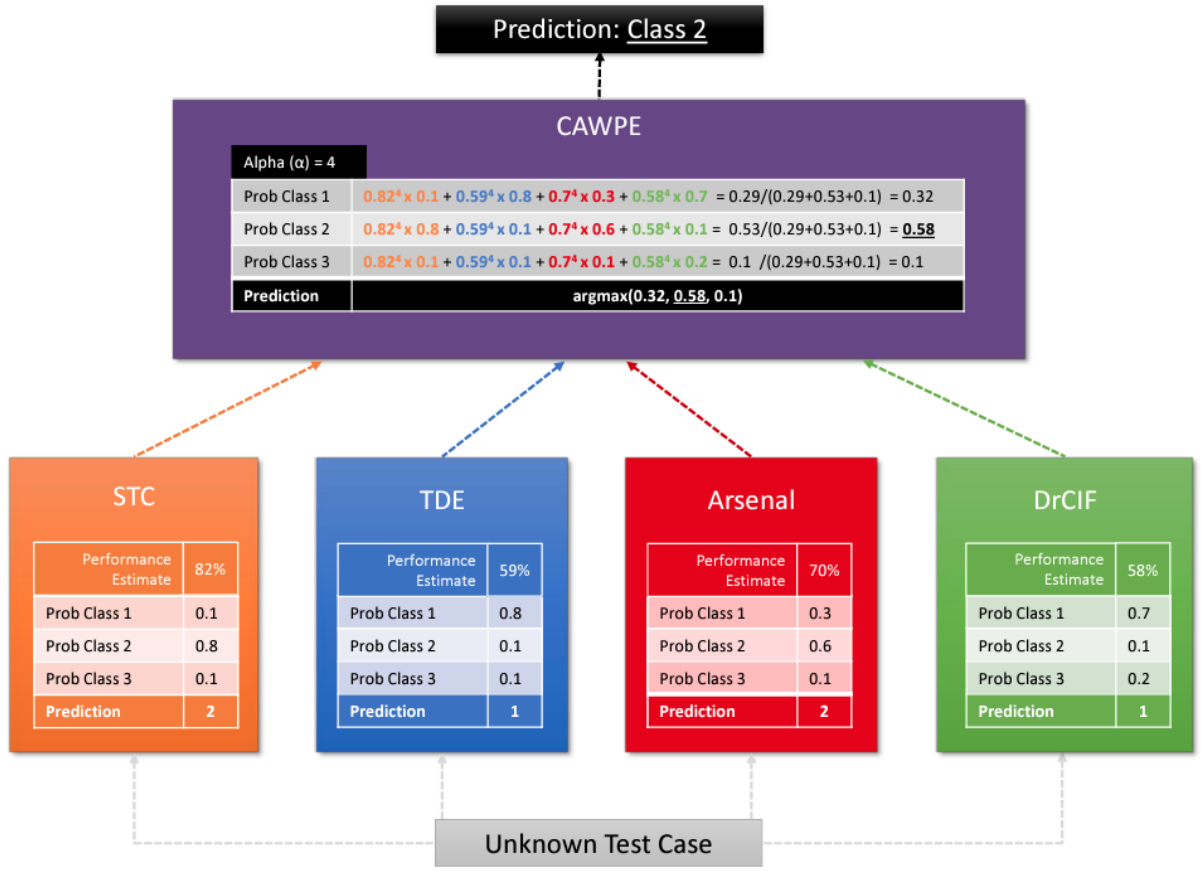


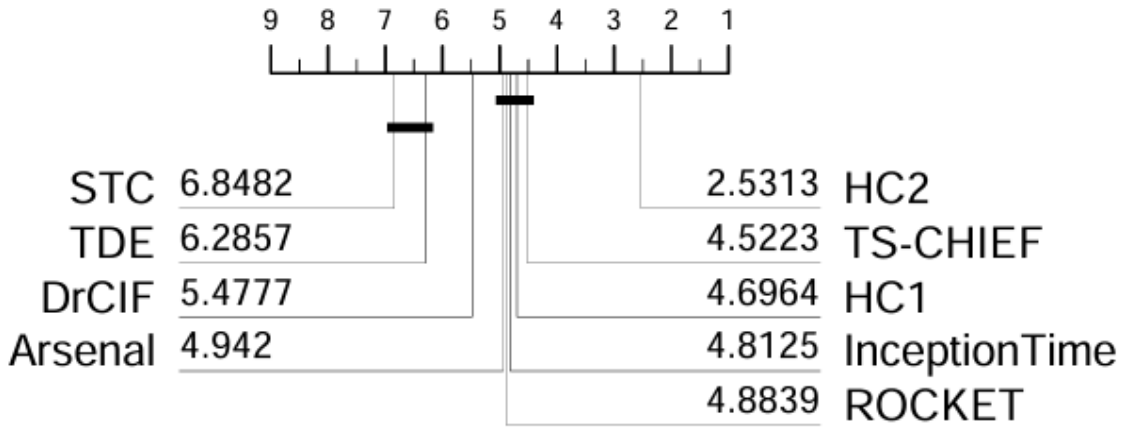


HIVE-COTE 2.0

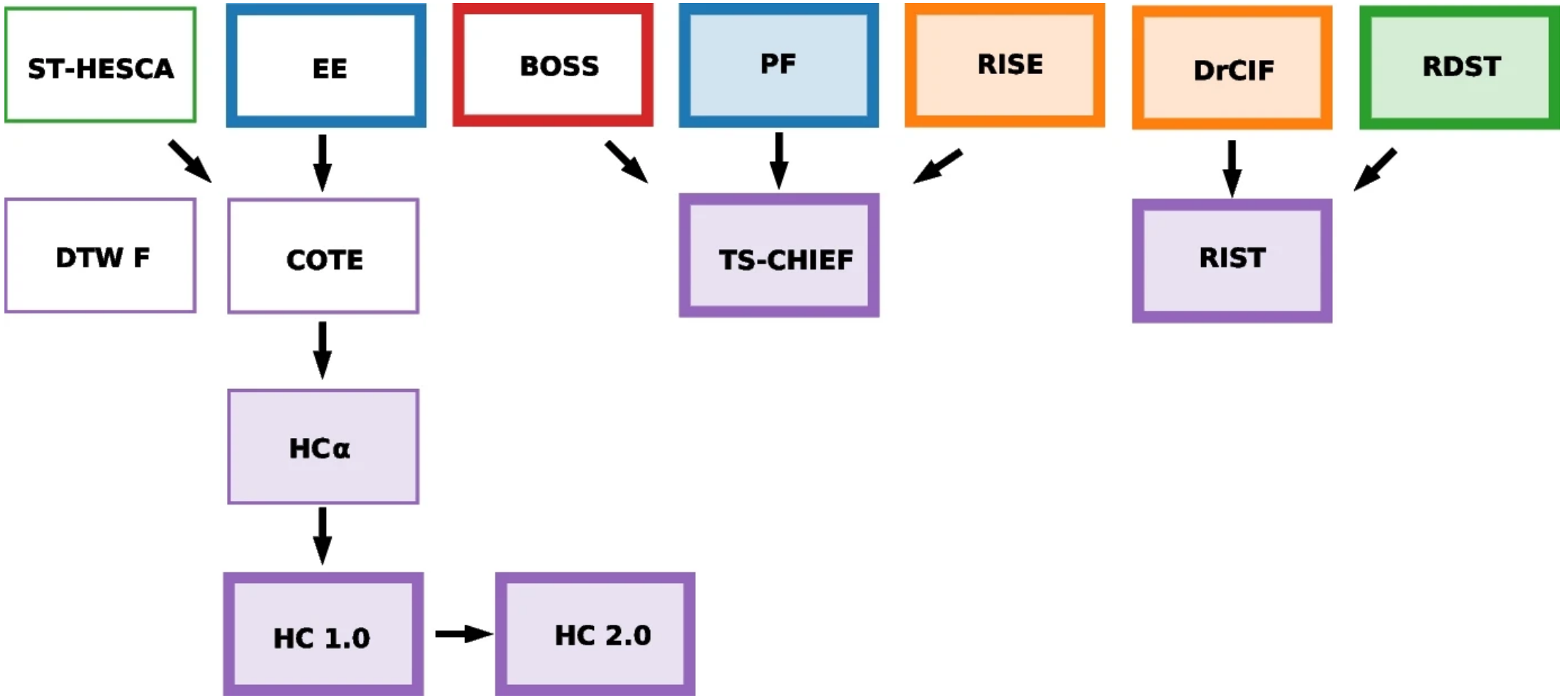
Adopts the following ensembles:

- Shapelet Transform Classifier.
- A convolution-based ensemble of ROCKET named Arsenal.
- The dictionary-based Temporal Dictionary Ensemble, i.e., a fast version of BOSS.
- The interval-based DrCIF.

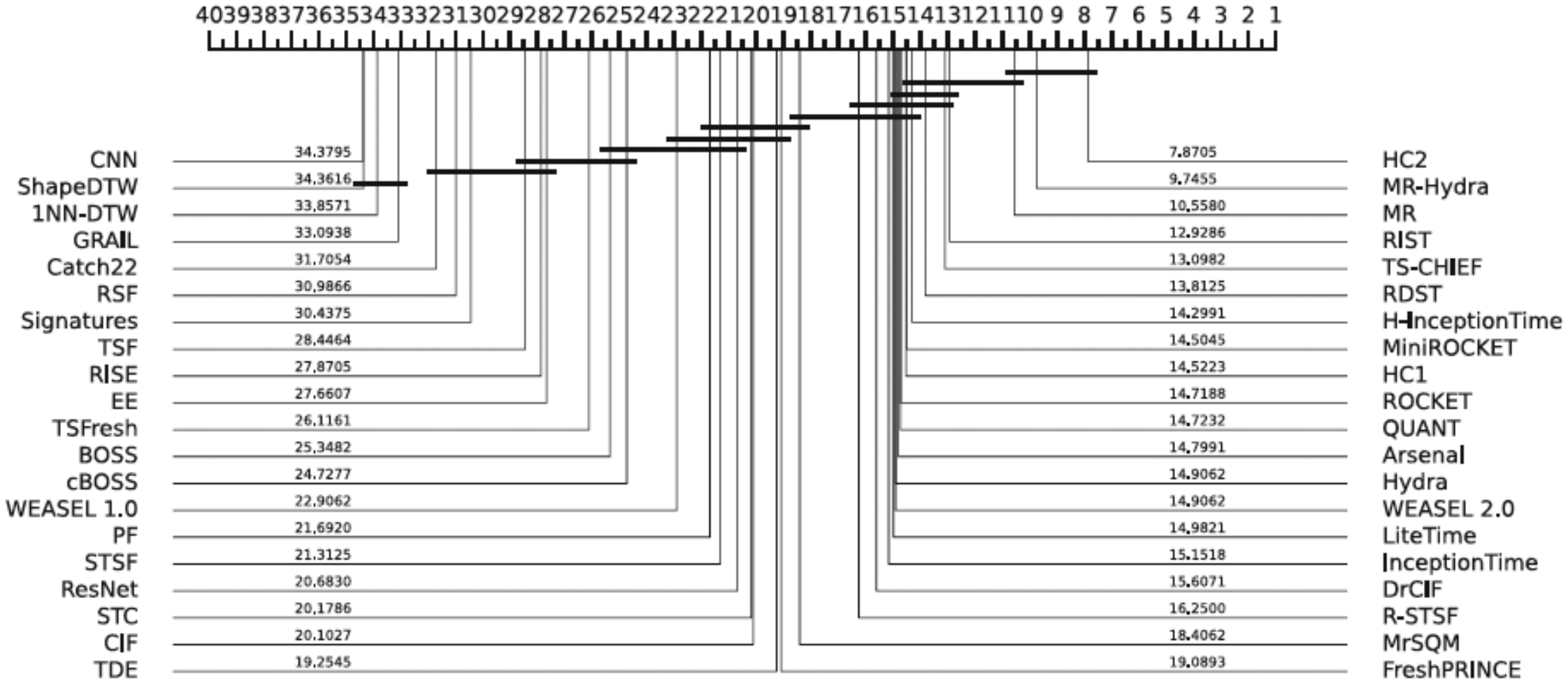




Overview of Hybrid Models and Relationships



TSC Methods Comparison



TSC Methods Comparison

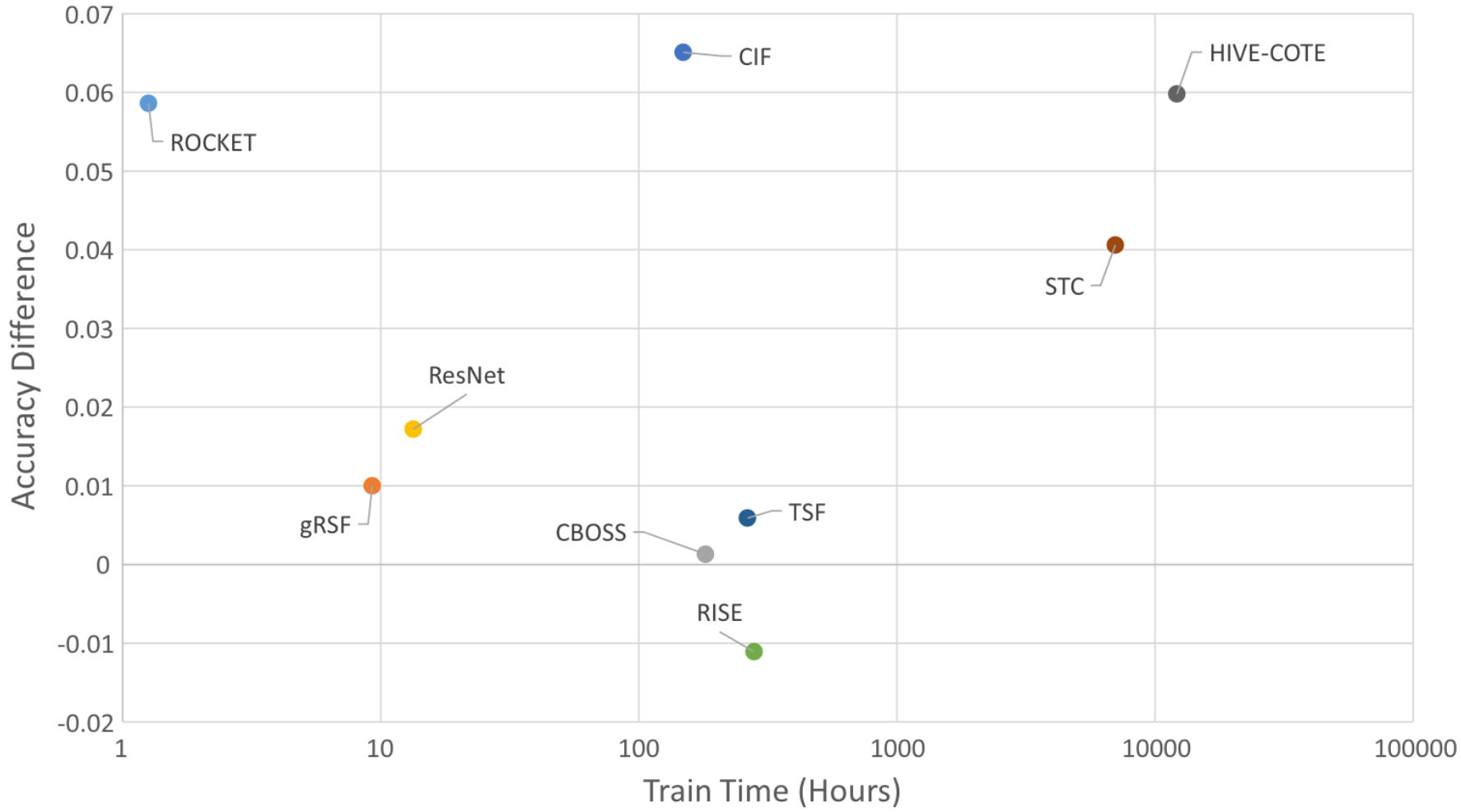


Fig. 10 Average difference in accuracy to DTW_D versus train time for 9 MTSC algorithms

eXplainable Artificial Intelligence

XAI - eXplainable Artificial Intelligence

- Set of processes and methods that allow human users to understand, trust, and manage the outputs of AI models.
- XAI aims to make the inner workings of AI systems more transparent and interpretable, addressing the “black box” nature of many advanced AI models, especially deep learning algorithms.

What is a Black Box Model?

- A ***black box*** is a model, whose internals are either unknown to the observer or they are known but uninterpretable by humans.

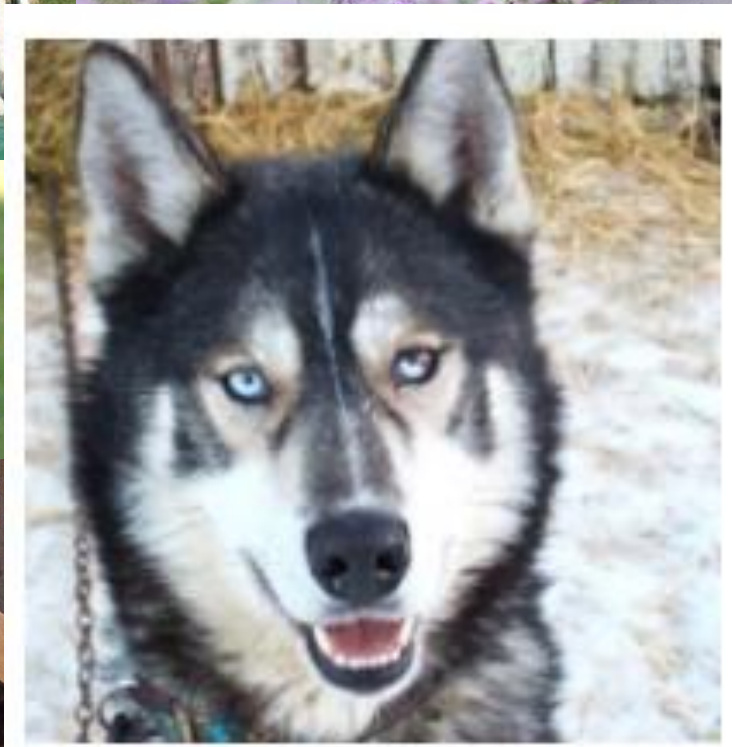
Example:

- DNN
- Ensembles
- ROCKET
- HIVE-COTE

Motivations For Explanation Methods

COMPAS Recidivism

The Wolf and the Husky



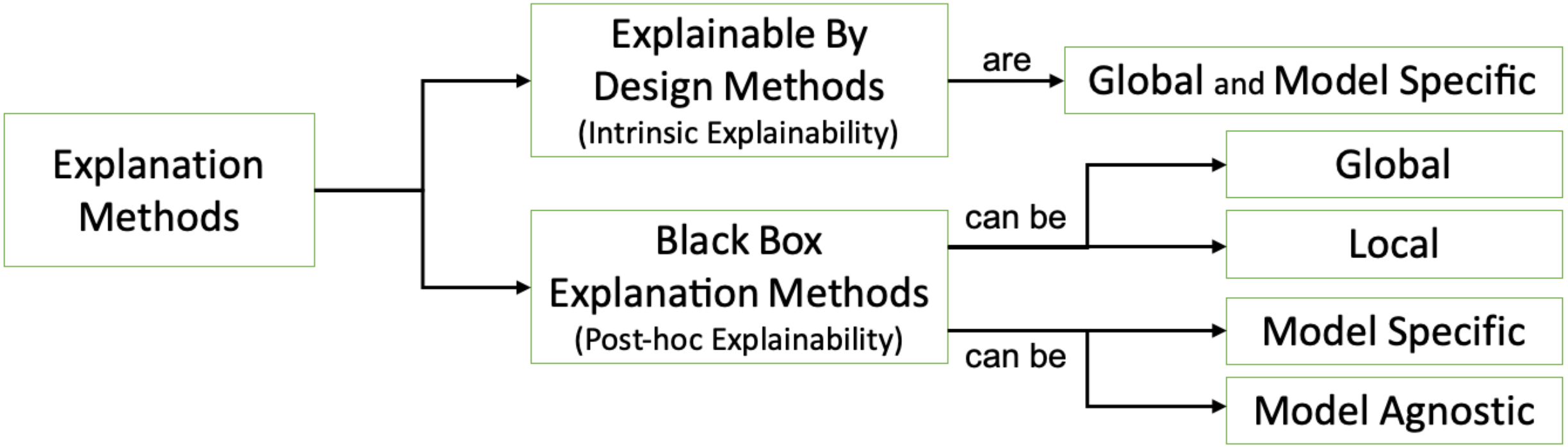
(a) Husky classified as wolf

(b) Explanation

Right of Explanation

Since 25 May 2018, GDPR establishes a right for all individuals to obtain “meaningful explanations of the logic involved” when “automated (algorithmic) individual decision-making”, including profiling, takes place.

XAI Taxonomy of Explanation Methods



Local Post-hoc Explanations Types

- Outlook = Sunny
- Temp = Hot
- Humidity = Normal
- Wind = Weak

- Black Box Prediction:
- Play Tennis = Yes

- Logic-based
  - Rule-based
  - Decision Tree
- Score-based
  - Features Importance
  - Saliency Maps
  - Attributions
- Instance-based
  - Prototypes
  - Counter-exemplars

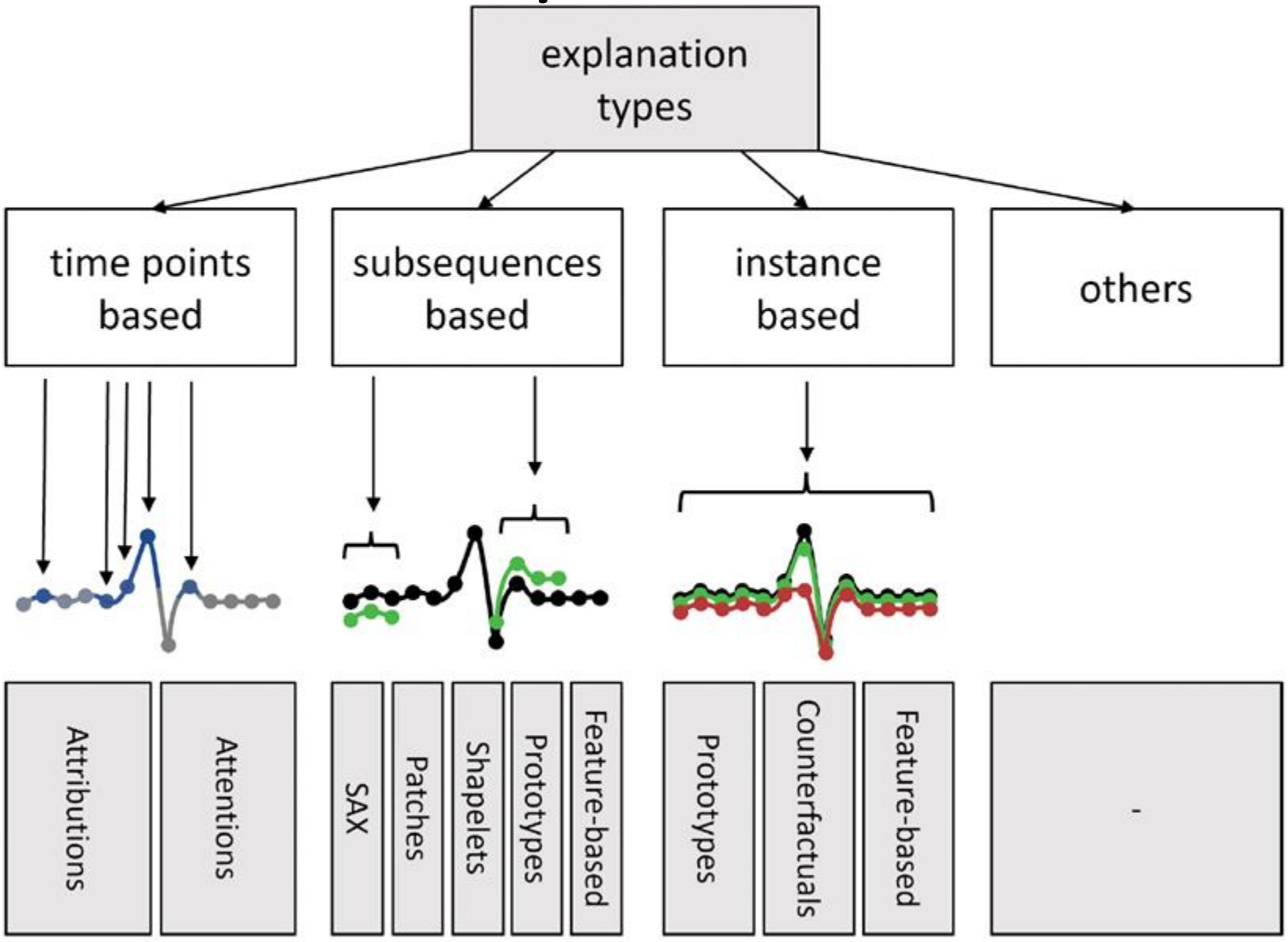
- IF Outlook = Sunny AND Humidity = Normal THEN Play Tennis = Yes

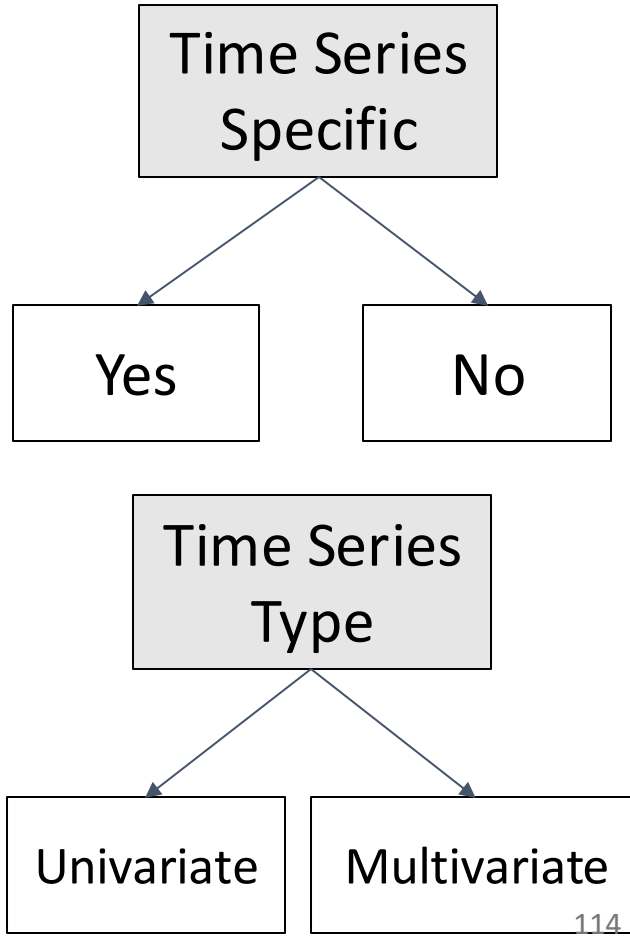
- Outlook: 0.7
- Temp: 0.0
- Humidity: -0.4
- Wind: 0.0

- Outlook = Sunny
- Temp = Hot
- **Humidity = Hight**
- Wind = Weak

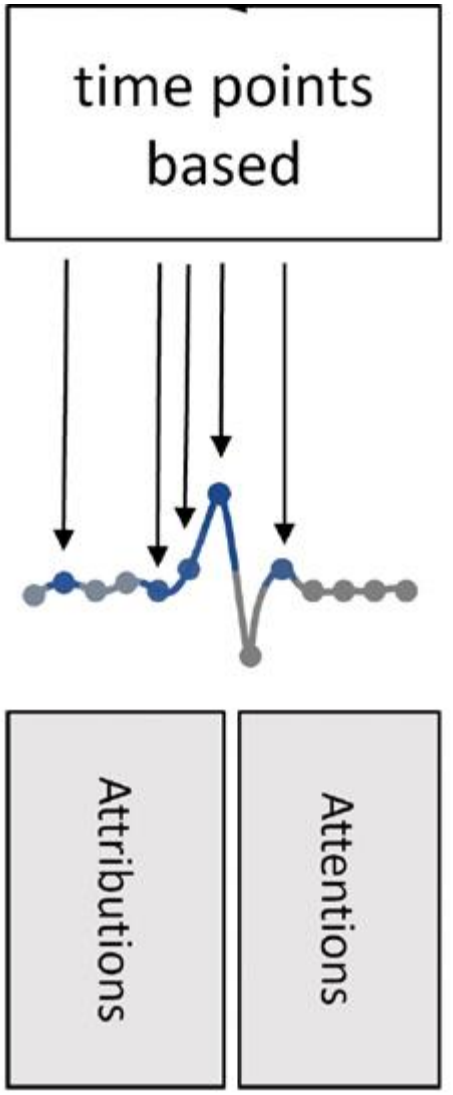
- Black Box Prediction:
- Play Tennis = Yes

XAI-TS Taxonomy



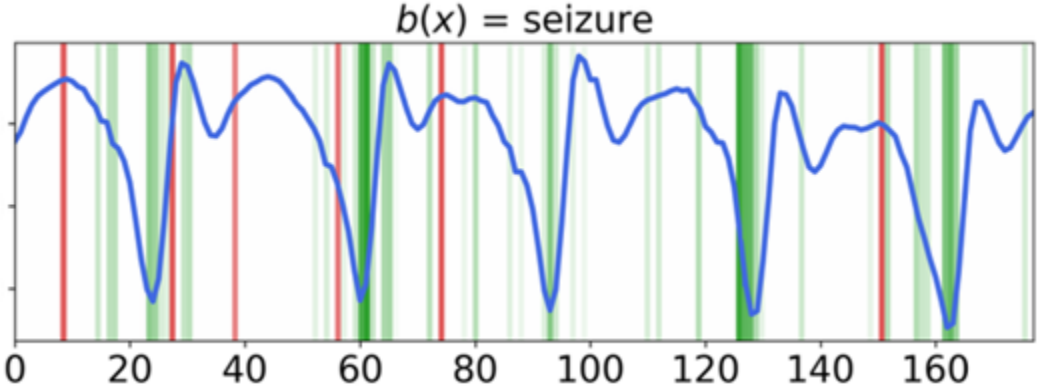


Time Points-based Explanations



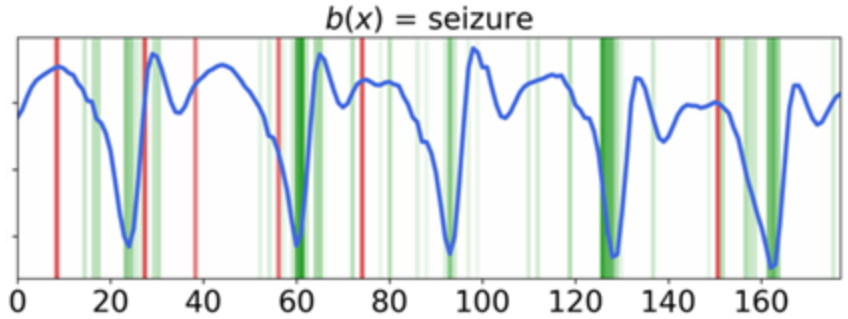
Score-based Explanations

- Score-based explanation methods attribute a local importance to each input variable w.r.t. their contribution towards the predicted.
- The higher is the value in absolute term, the higher is the importance, the closer is to zero the small is the importance for the returned outcome.
- If the score is positive the feature-value has a positive contribution towards the outcome, while if the score is negative the feature-value has a negative contribution.
- Issue: these approaches can require a “default” value to be used as baseline or to simulate the “removal” of a point/subsequence.
- Explanation depends on the value used to replace real values.

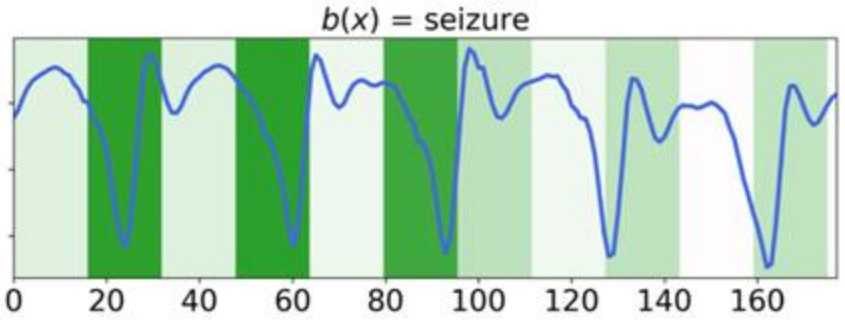


Score-based Explanation Strategies

- **Consider each timestep as a feature:**
  - Works fine for time-independent TS transformation such as DTF, SFA, BOSS, Kernels
  - Assume time independent values if used on raw TS, so minimal misalignments can cause problem, close time stamps can have opposite contribution



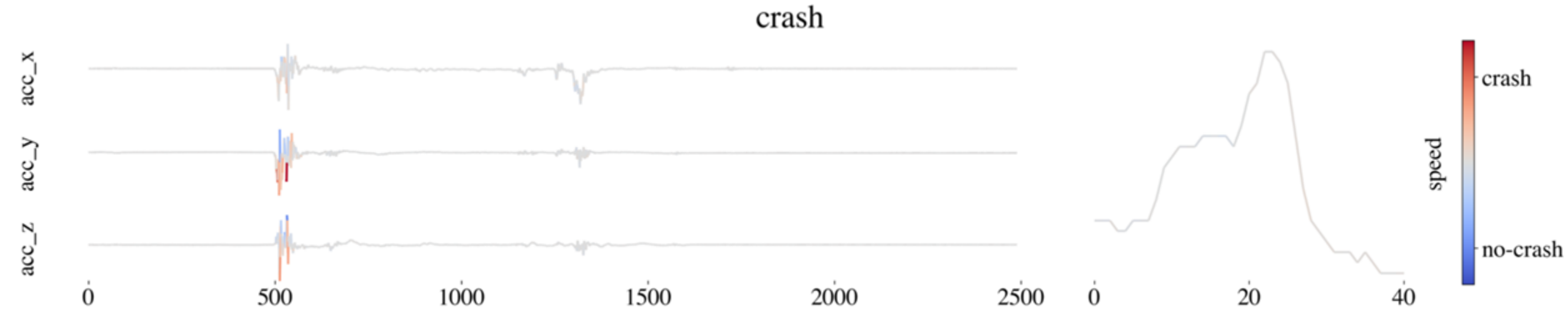
- **Split the TS in subsequences and consider them as features**:
  - Explanation depends on the time-dependent transformation: PAA, SAX, BOP, Shapelets



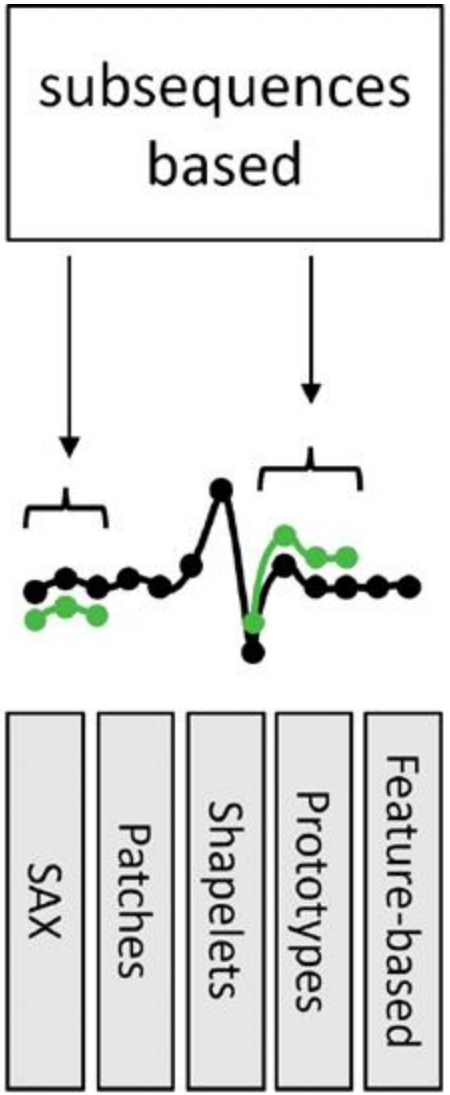
Most Common Score-based TS Explainers

Attribution methods can be model-agnostic or model-specific.

- **SHAP** is available in an inefficient agnostic version, and in multiple efficient model specific version (TreeShap, GradienShap, LinearShap).
- **GradientShap** is a modified version of Integrated Gradients, optimized for neural networks.

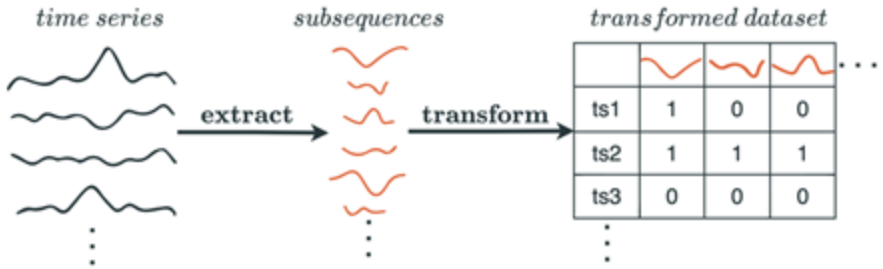


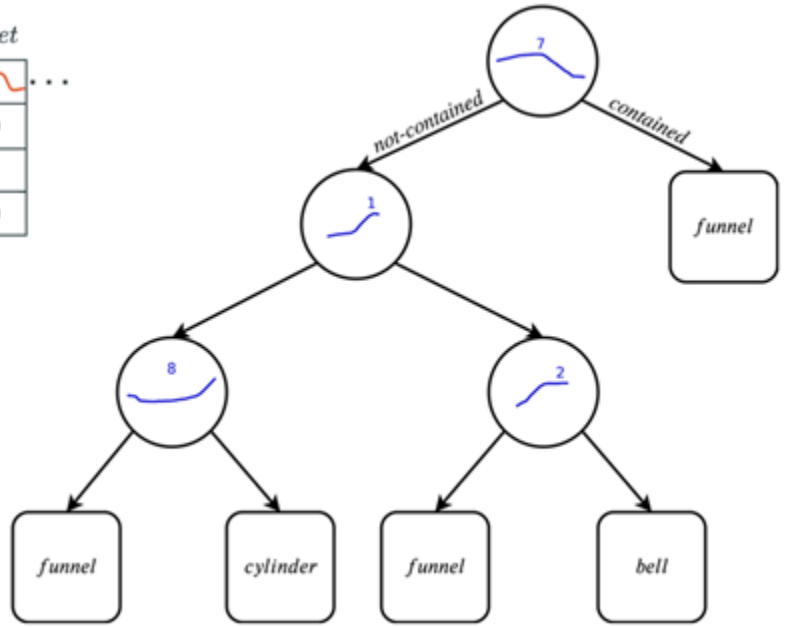
Subsequences-based Explanations

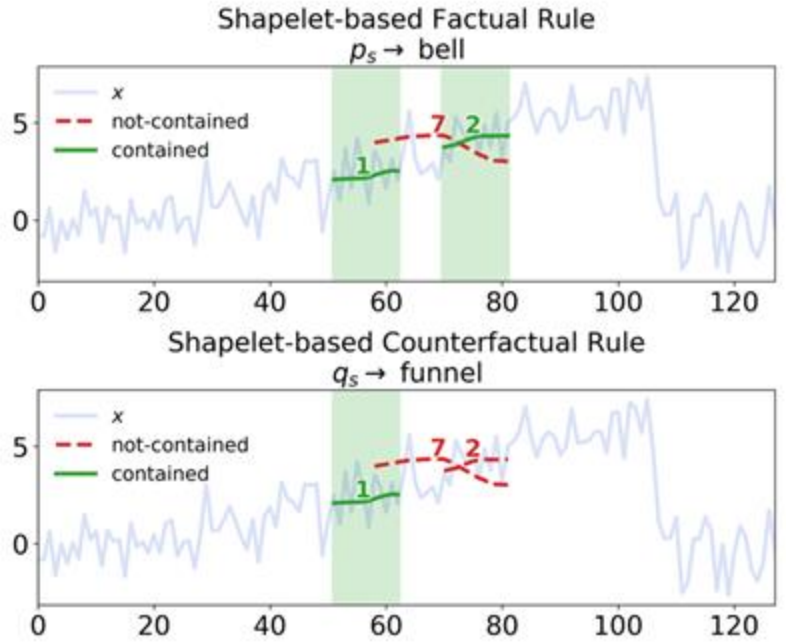


Subsequences-based Explanations

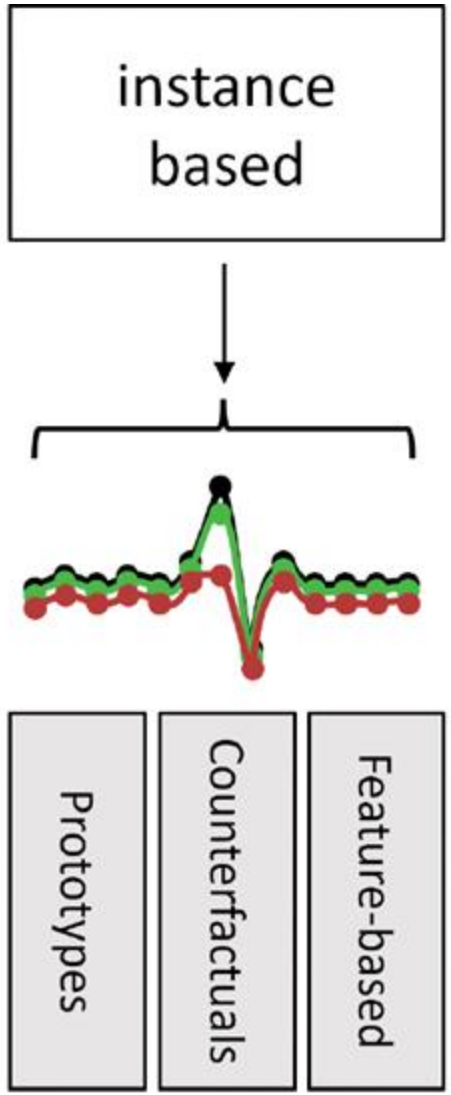
- After having transformed the TS into time-dependent humanly understandable feature describing subsequence that can be referred into the input TS, any interpretable ML approach can be used as it is or as a surrogate to explain another model.





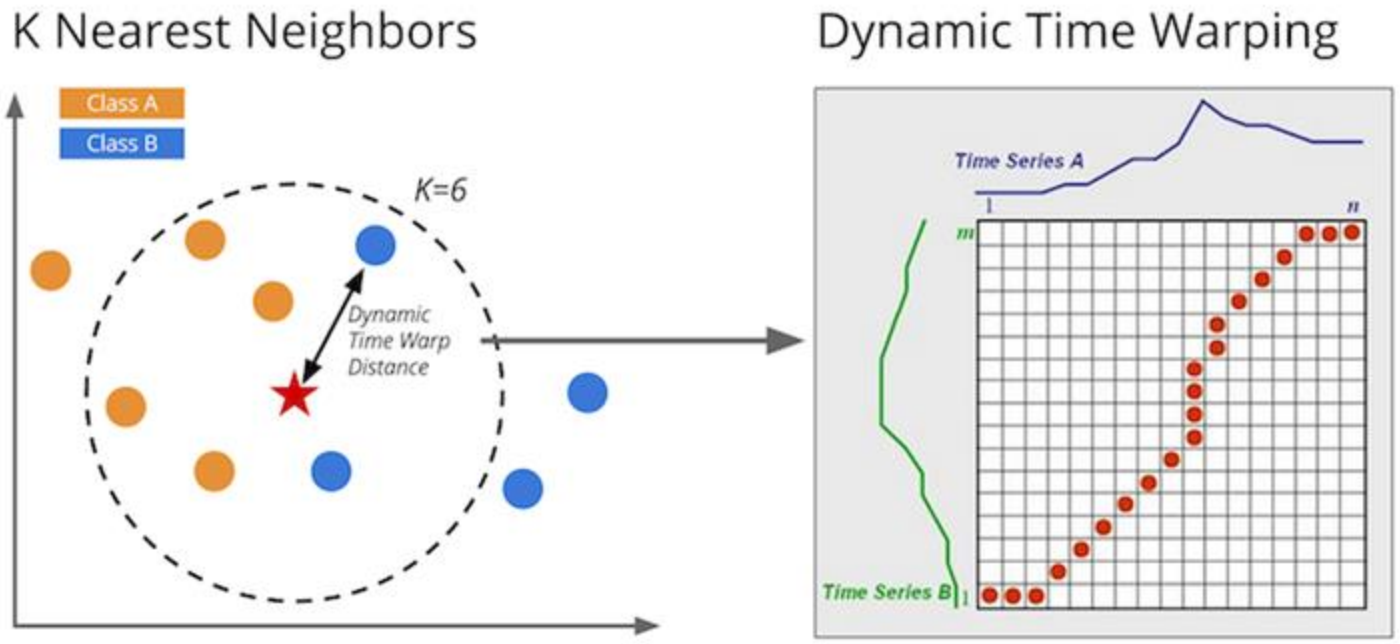


Instance-based Explanations



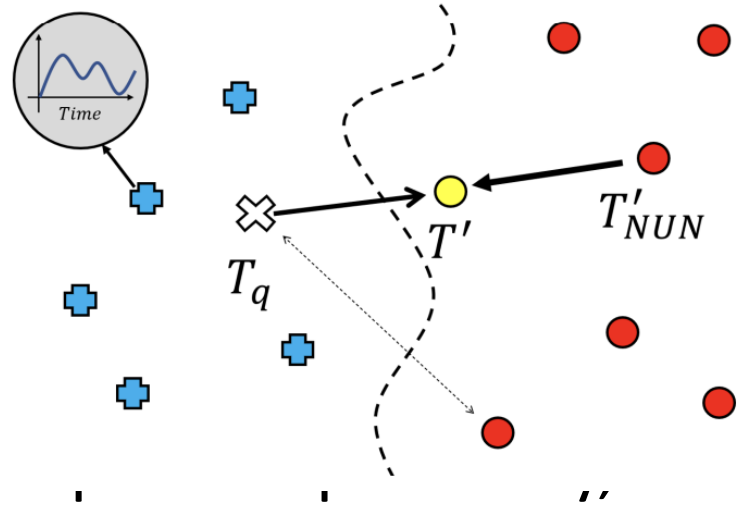
Counterfactuals

- Counterfactual time series show the minimal changes in the input data that lead to a different decision outcome.

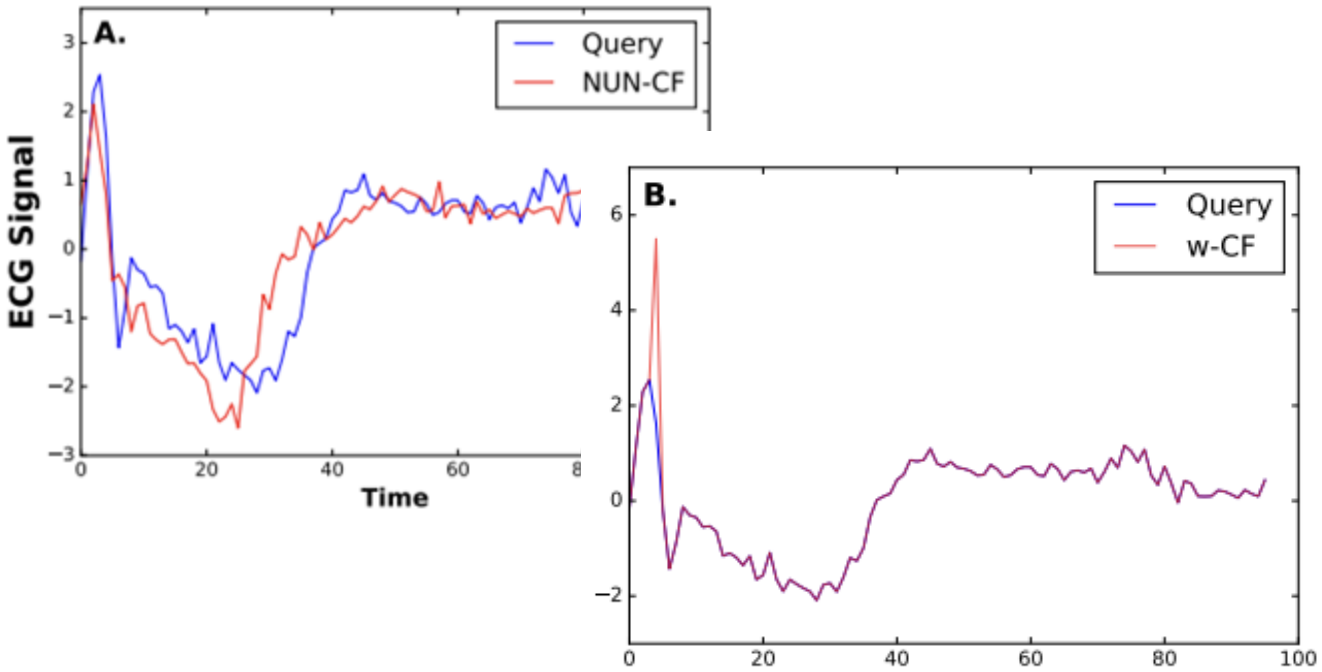


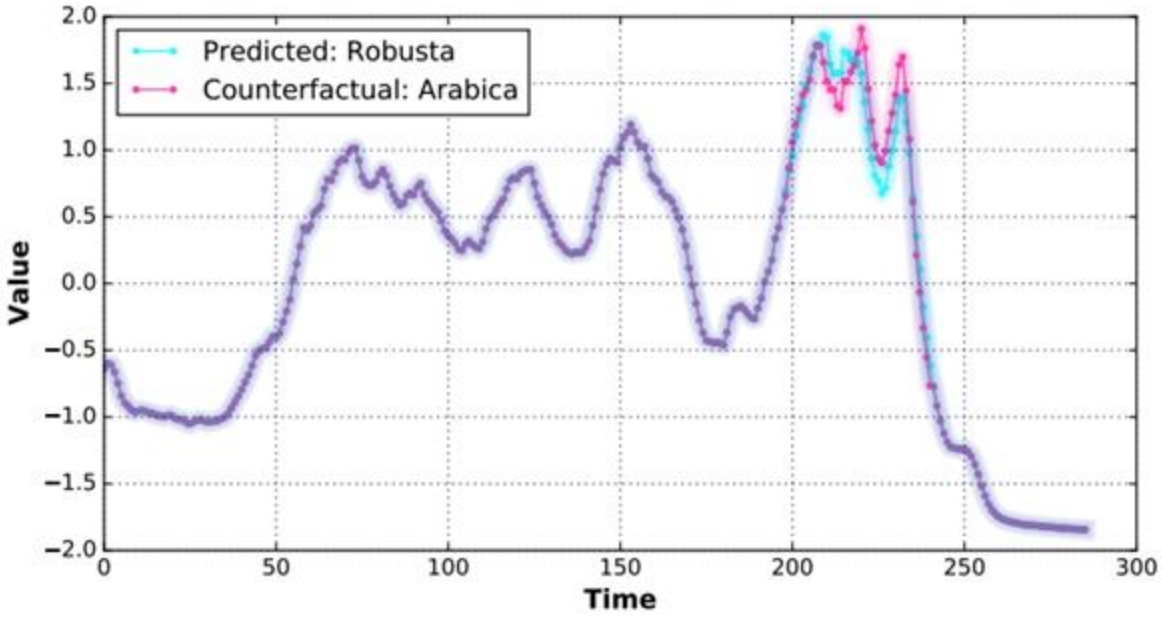
- KNN can be paired with the Euclidean or DTW distance to classify time series.
- To find a counterfactual, it searchers for the closest instance having a different class.

Counterfactuals



- Native Guides builds upon the KNN approach, counterfactuals, following four identified key p sparsity, plausibility, and diversity





References

- Convolutional neural networks for time series classification. Bendong Zhao et al. 2017
- Deep Residual Learning for Image Recognition. Kaiming He et al. 2015
- Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline. Zhiguang Wang et al 2016
- InceptionTime: Finding AlexNet for Time Series Classification Hassan Ismail Fawaz et al. 2019
- Multivariate LSTM-FCNs for Time Series Classification. Karim et al. 2019
- Tapnet: Multivariate time series classification with attentional prototypical network. Zhang et al. 2020
- ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. Angus Dempster et al. 2019
- MINIROCKET: A Very Fast (Almost) Deterministic Transform for Time Series Classification. Angus Dempster et al. 2021
- MultiRocket: Multiple pooling operators and transformations for fast and effective time series classification. Chang Wei Tan et al. 2021
- Time Series Classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles. Jason Lines et al. 2018.
- TS-CHIEF: A Scalable and Accurate Forest Algorithm for Time Series Classification. Ahmed Shifaz et al. 2019
- HIVE-COTE 2.0: a new meta ensemble for time series classification. Matthew Middlehurst et al. 2021
- Bake off redux: a review and experimental evaluation of recent time series classification algorithm. Matthew Middlehurst et al. 2024
- A survey of methods for explaining black box models. Riccardo Guidotti et al. 2018.
- Explainable AI for Time Series Classification: A Review, Taxonomy and Research Directions. Andreas Theissler et al. 2022