DATA MINING 2
Time Series - Classification

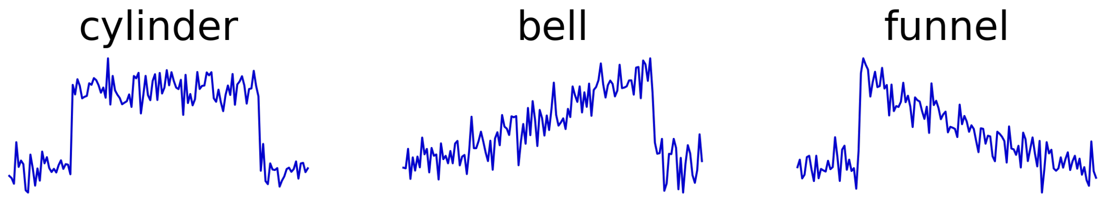
Riccardo Guidotti

a.a. 2019/2020

UNIVERSITÀ DI PISA

Time Series Classification

- Main difference between classification and forecasting: forecasting is about predicting a future state/value, classification is about predicting the current label/class.
- Applications:
  - Automated detection of heart diseases
  - Discovery of presence in a room from temperature, humidity, light
  - Identification of the activity performed from smart devices (walking, sitting, laying)
  - Identification of stock market anomalies in pricing, sales volumes, stocks
  - Warning of Natural Disasters (flooding, hurricane, snowstorm),
- Techniques:
  - Motif Discovery
  - Machine Learning Classifiers
  - Deep Neural Networks



Problem Fromulation

- Given a set X of n time series, $X = \{x_1, x_2, \ldots, x_n\}$, each time series has m ordered values $x_i = < x_{t1}, x_{t2}, \ldots, x_{tm} >$ and a class value c_i.
- The objective is to find a function f that maps from the space of possible time series to the space of possible class values.
- Generally, it is assumed that all the TS have the same length m.

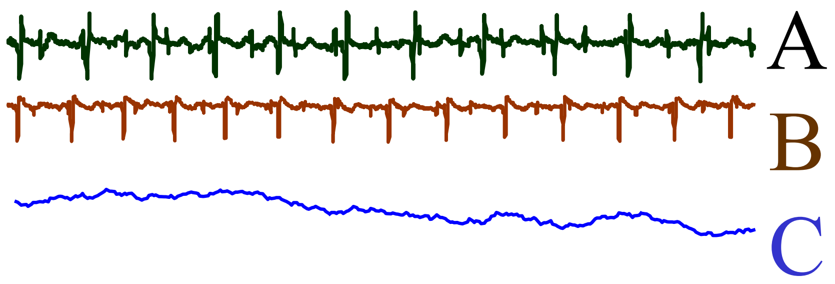
Time Series Classification and Similarities

- To some extent, TS classification rely on a measure of similarity between data.
- What makes time series classification an interesting area of investigation is that similarity between series is often embedded within the autocorrelation structure of the data.
- General approaches to measuring similarity between time series:
  - similarity in time (i.e. correlation-based)
  - similarity in change (autocorrelation-based)
  - ***similarity in shape*** (shape-based)
  - ***similarity in structure*** (features-based)
  - ***similarity in representation*** (NN-based)

Structural-based Classification

Structural-based Classification

- The basic idea is to:
  1. Extract *global* features from the time series,
  2. Create a feature vector, and
  3. Use it to as input for machine learning classifiers
- Example of features:
  - mean, variance, skewness, kurtosis,
  - 1st derivative mean, 1st derivative variance, …
  - parameters of regression, forecasting, Markov model

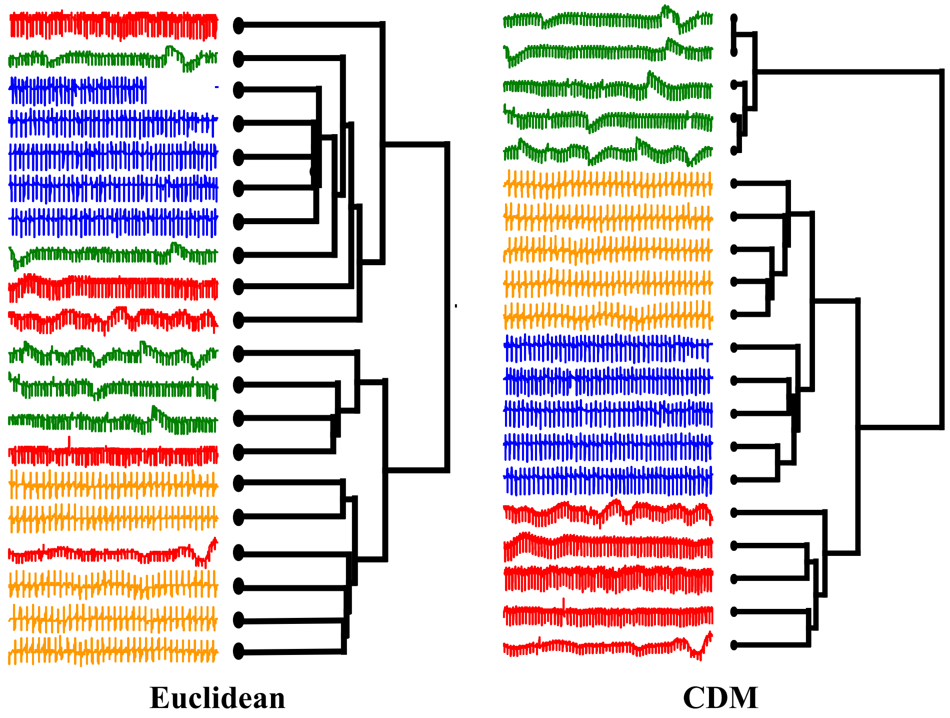


Feature\Time Series	A	B	C
Max Value	11	12	19
Mean	5.3	6.4	4.8
Min Value	3	2	5
Autocorrelation	0.2	0.3	0.5
…	…	…	…

Shape-based Classification

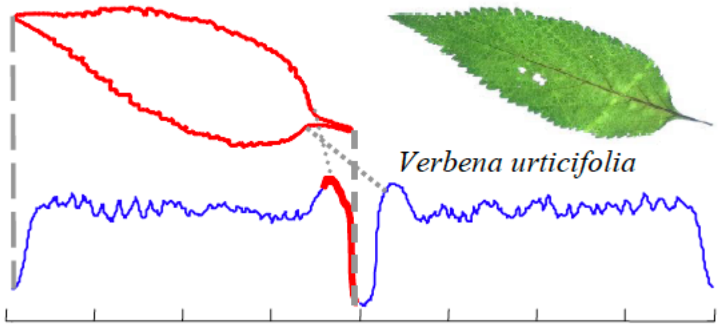
Shape-based Classification

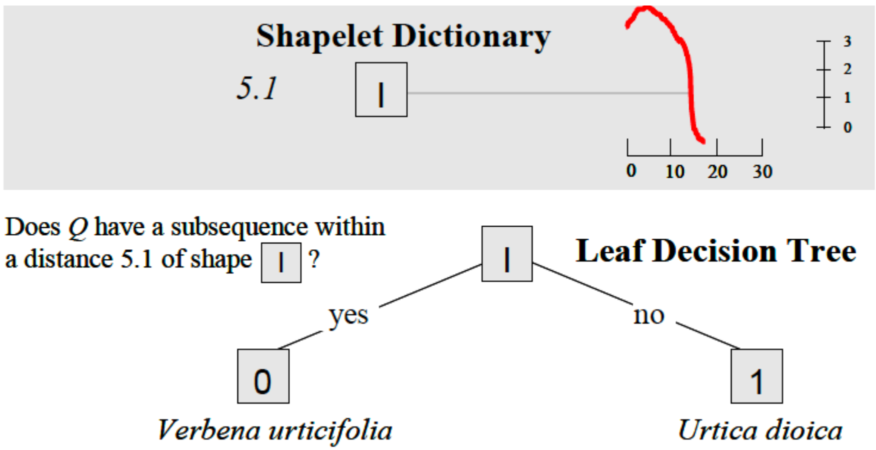
- Calculate the distance between TS using an appropriate distance function:
  - Euclidean/Manhattan
  - Dynamic Time Warping
  - Compression Based Dissimilarity
- Use an instance-based classifier (k-NN) to make the classification.



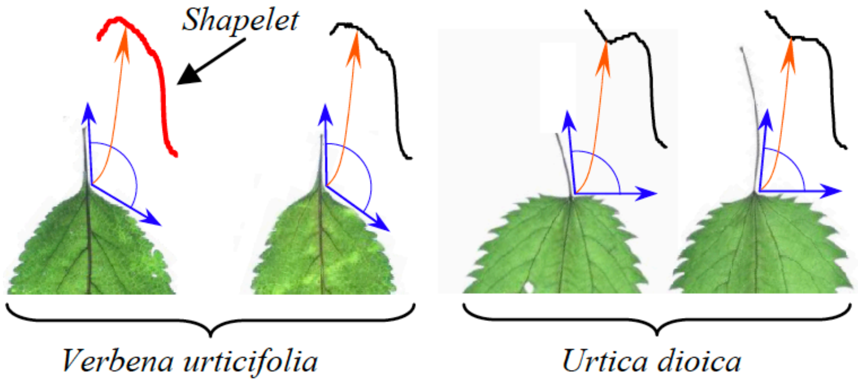
Shape-based Classification

1. Represent a TS as a vector of distances with representative subsequences, namely shapelets.
2. Use it to as input for machine learning classifiers.



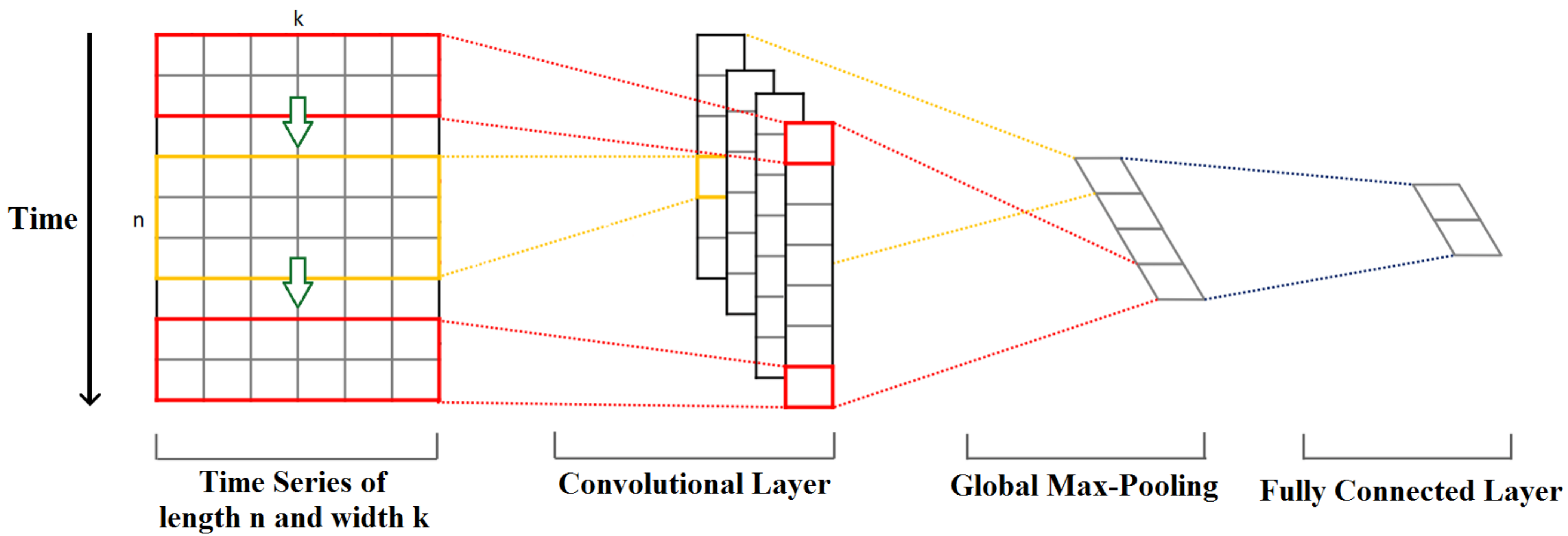


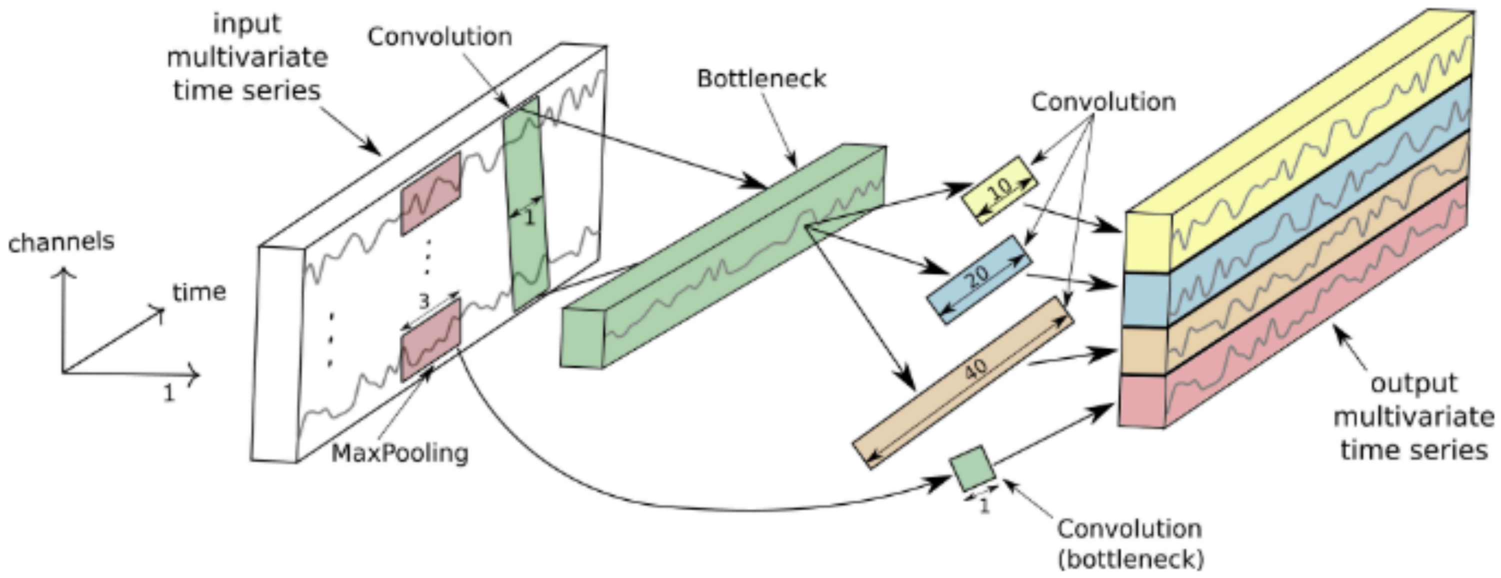
3.2	8.7
1.4	7.9
6.7	4.2
9.2	3.4



Time Series Classification with DNN

Time Series Classification with DNN





Convolutional Neural Network

Slides edited from Stanford

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

Convolutional Neural Network

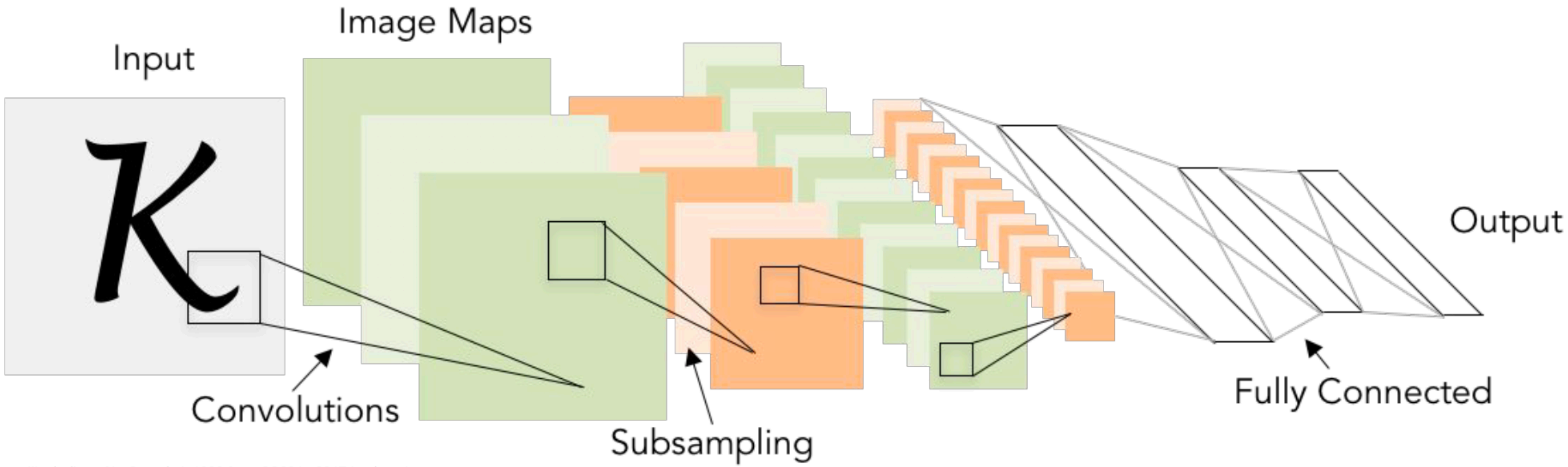
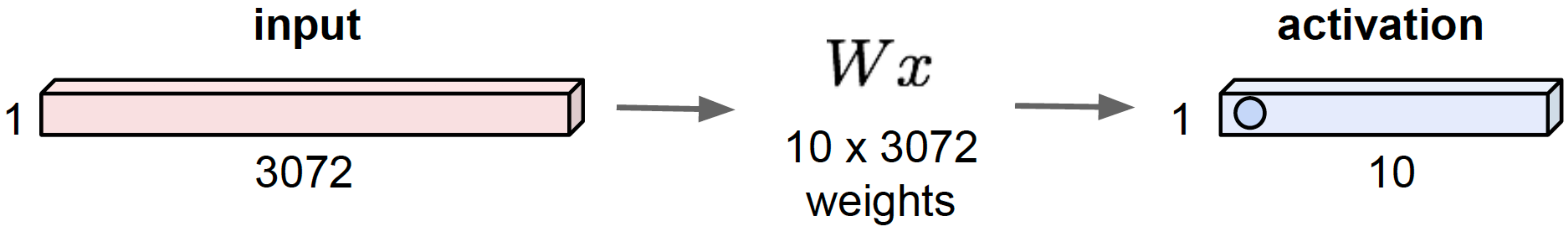


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

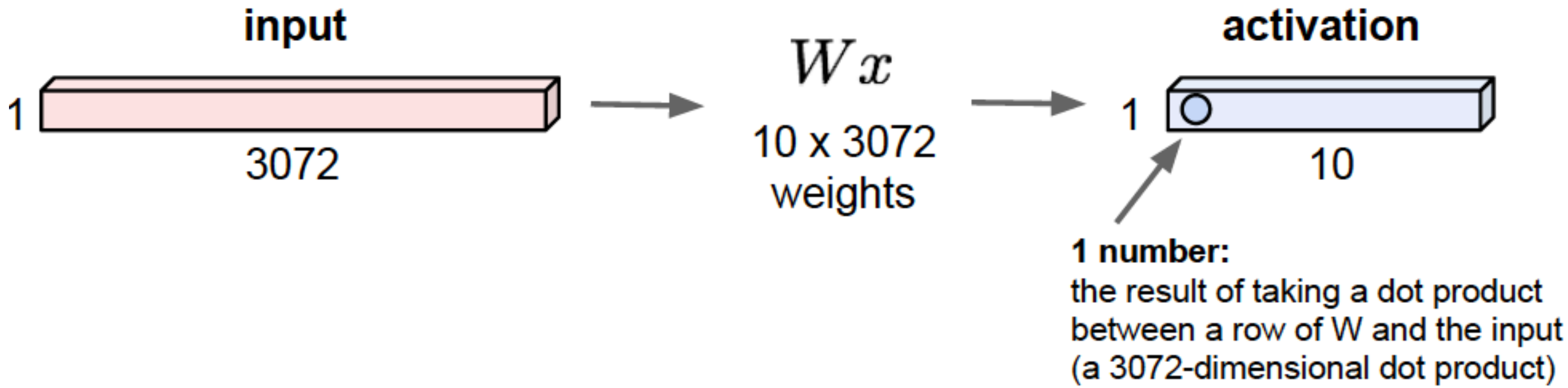
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



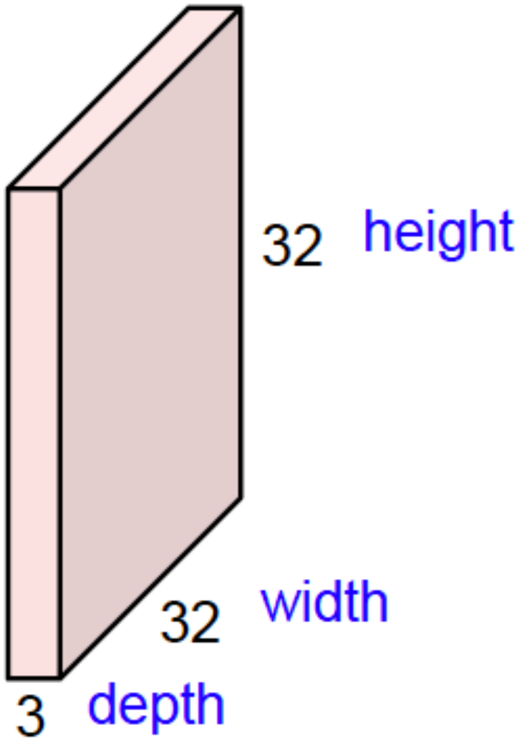
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Convolution Layer

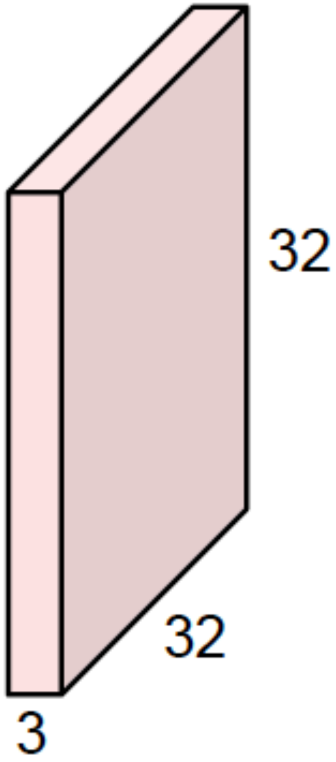
32x32x3 image -> preserve spatial structure



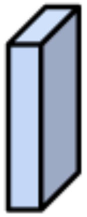
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

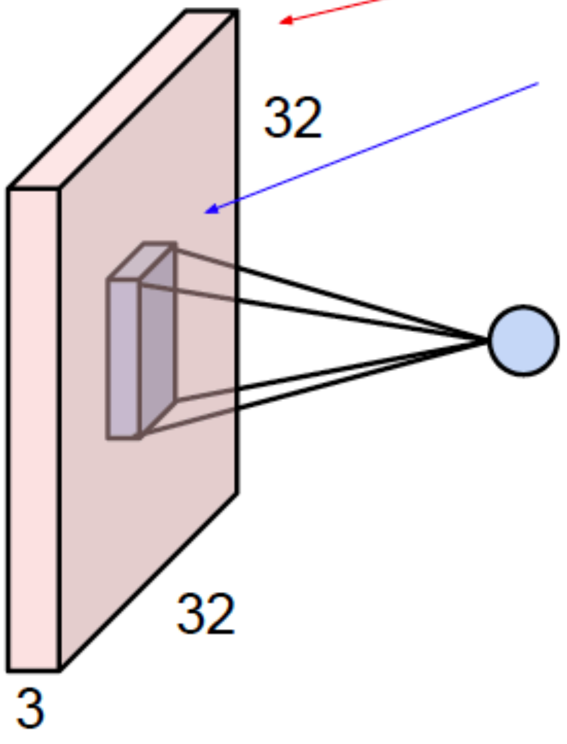


5x5x3 filter



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

Convolution Layer



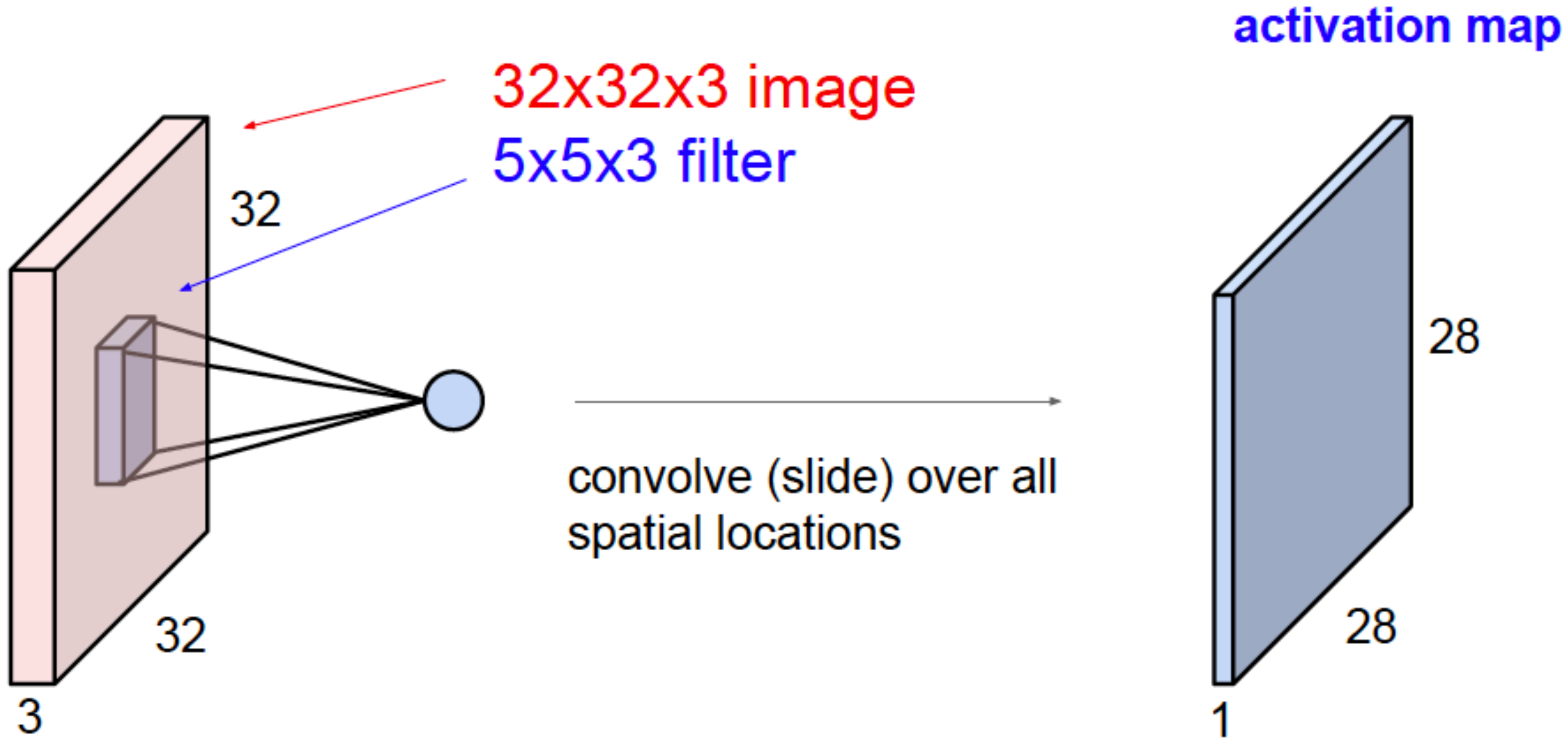
32x32x3 image

5x5x3 filter w

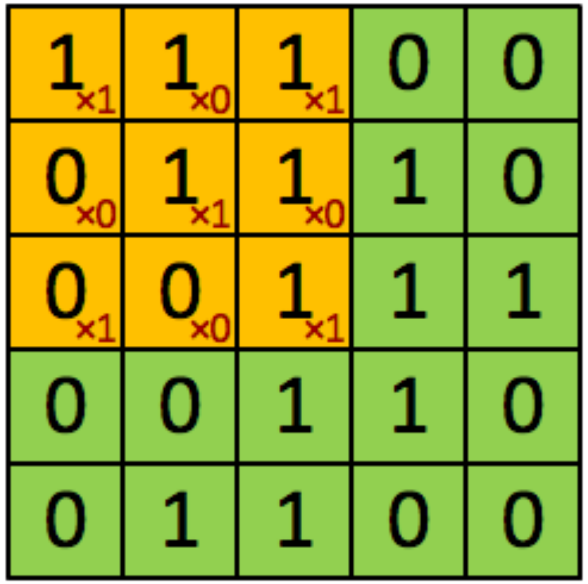
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

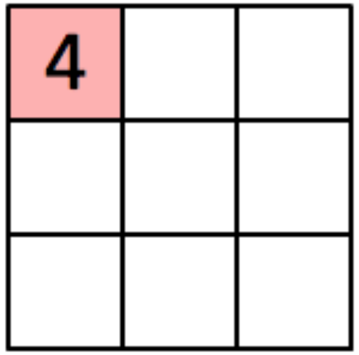
Convolution Layer



Convolution Layer



Image

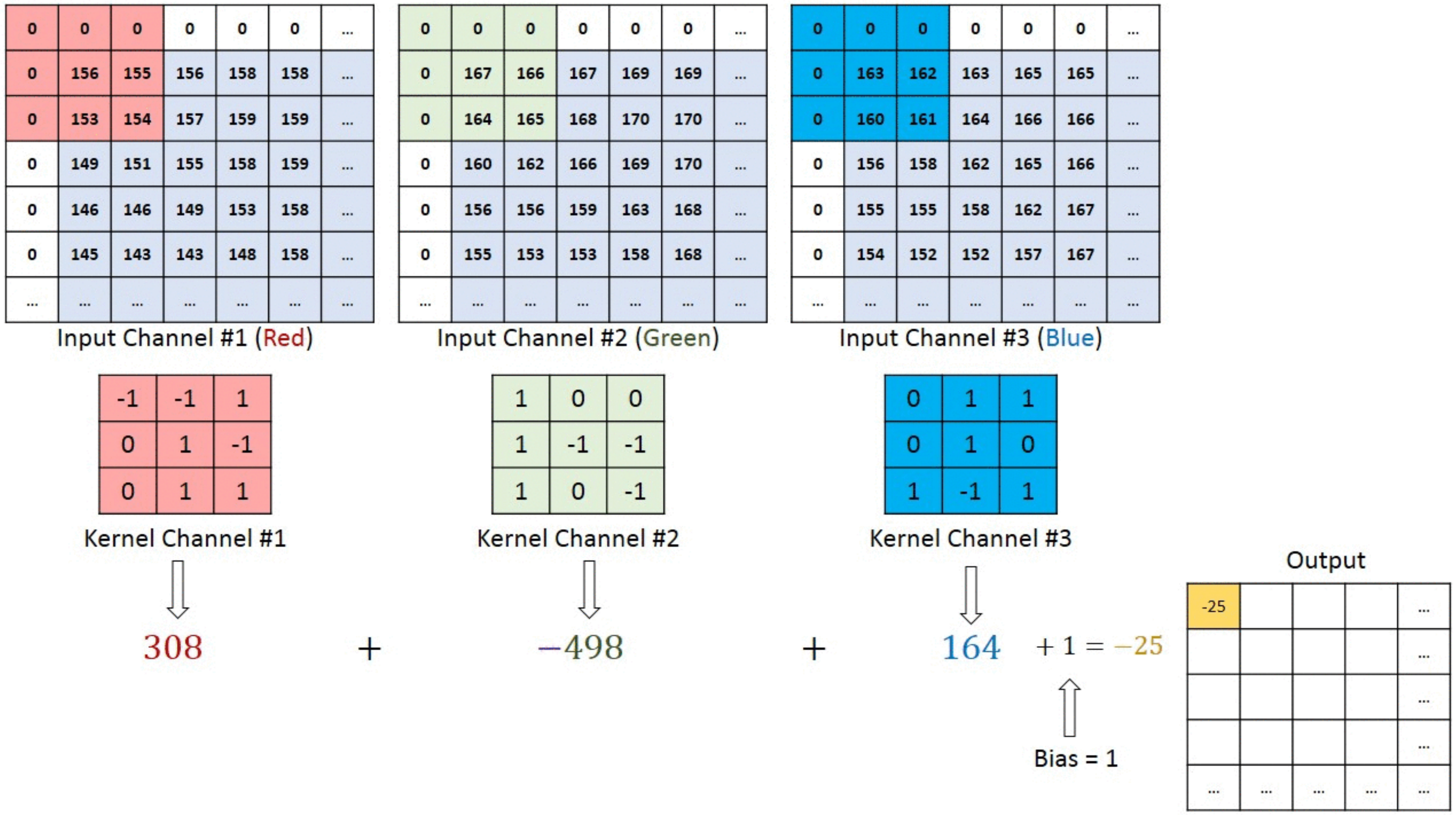


Convolved Feature

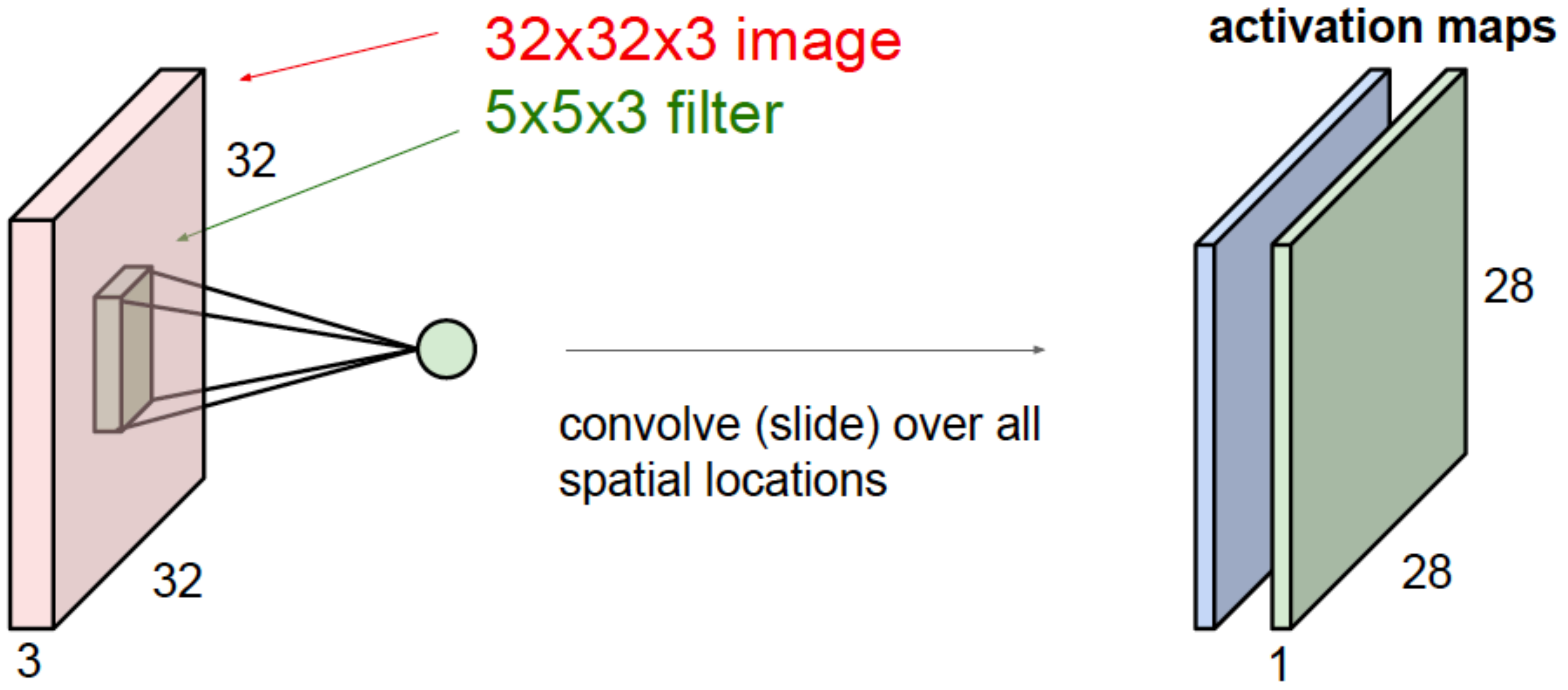
1	0	1
0	1	0
1	0	1

Convolution Kernel

Convolution Layer

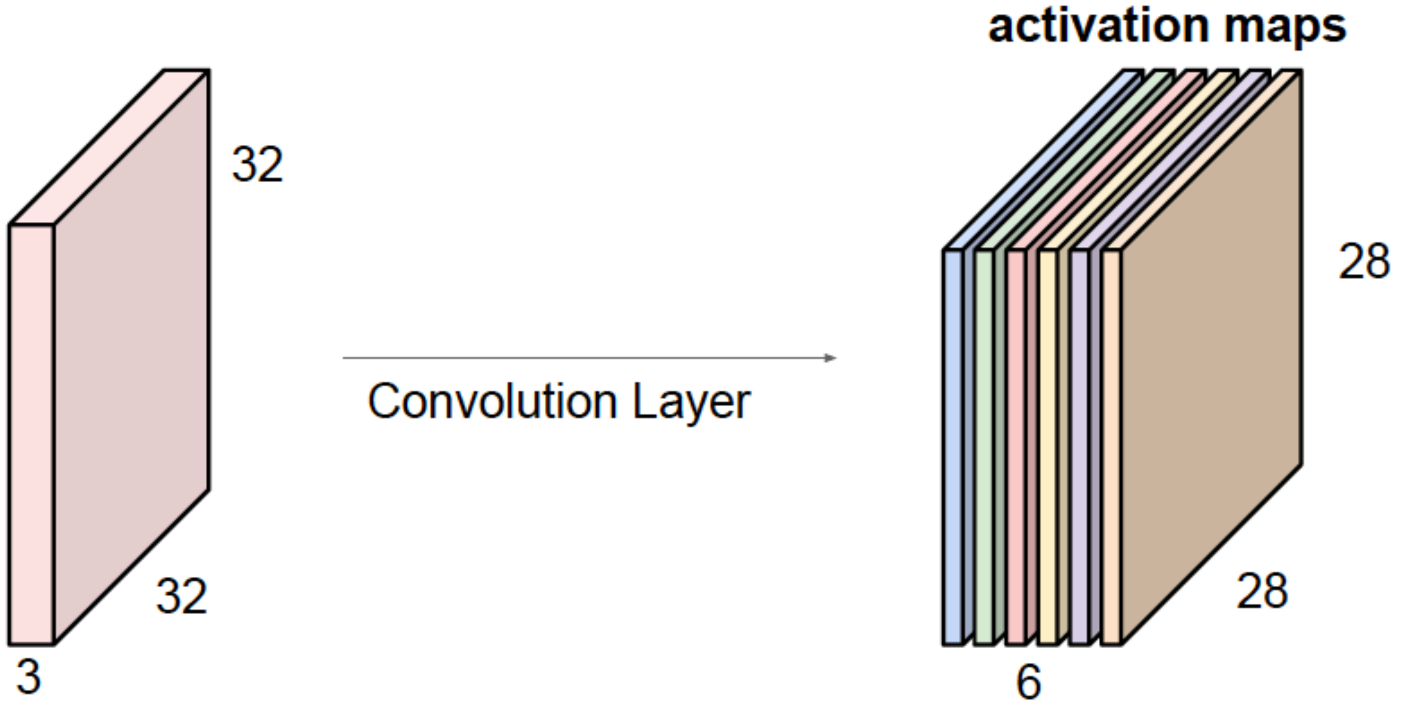


Convolution Layer



Convolution Layer

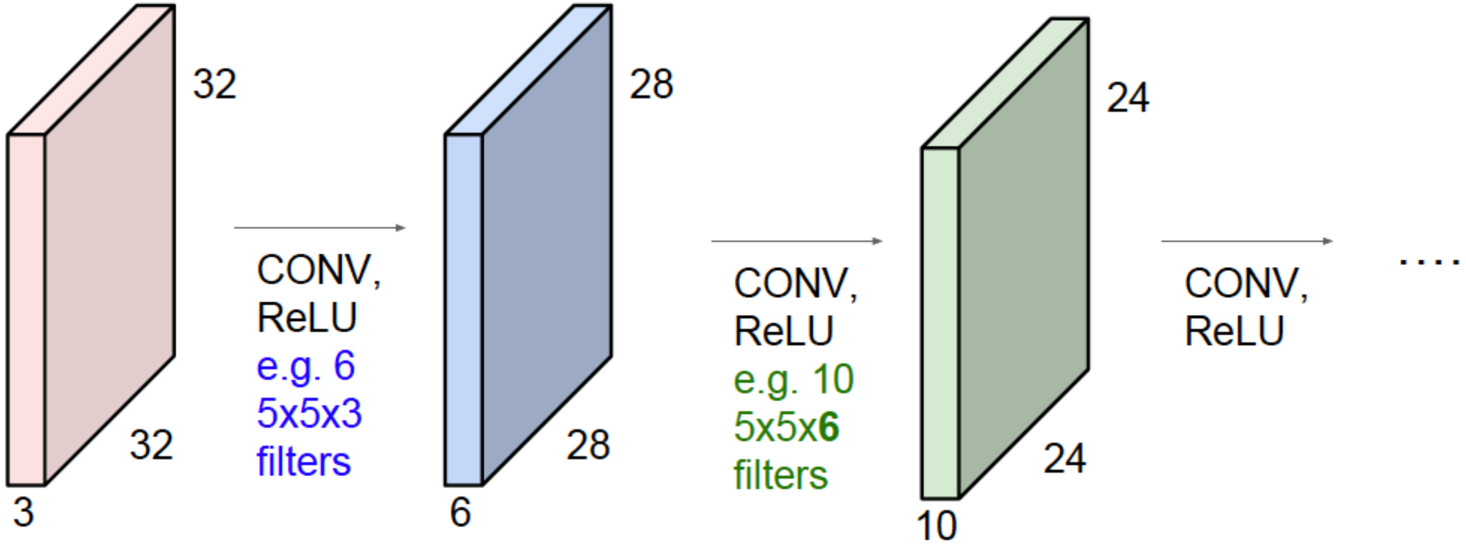
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



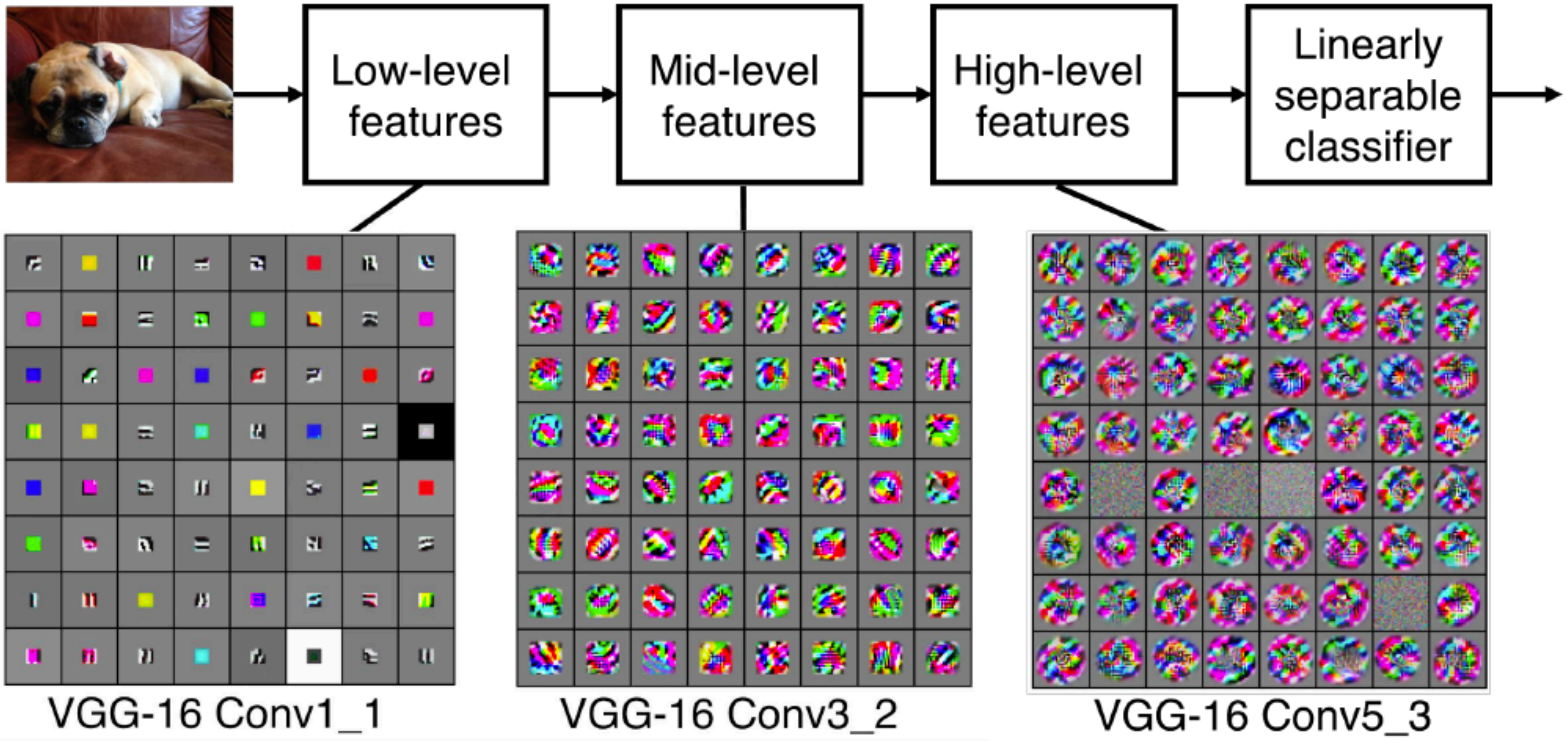
We stack these up to get a "new image" of size 28x28x6!

Convolutional Neural Network

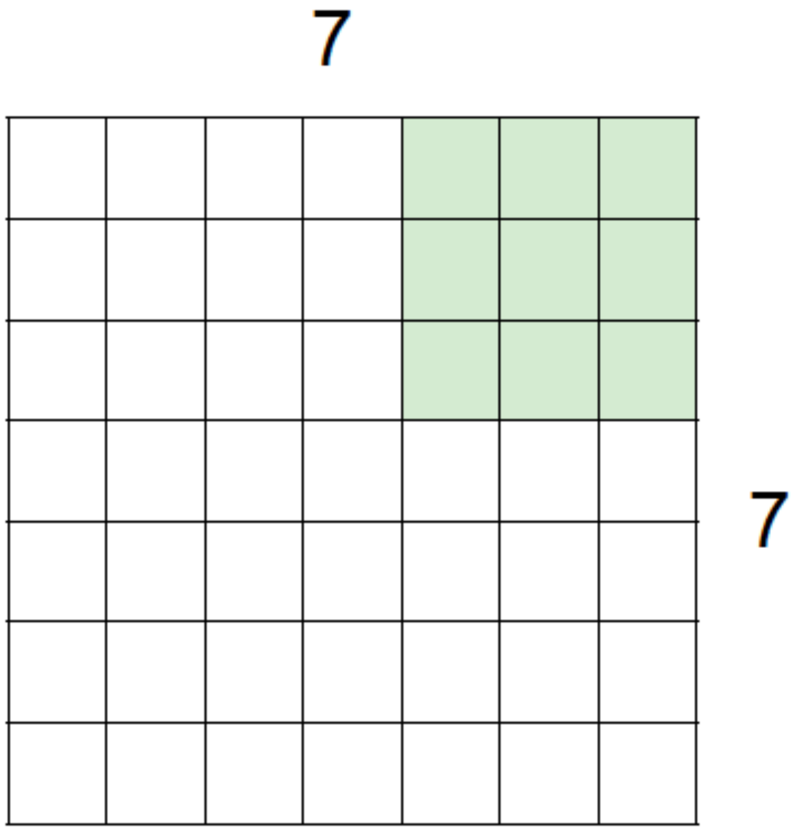
- CNN is a sequence of Conv Layers, interspersed with activation functions.
- CNN shrinks volumes spatially.
- E.g. 32x32 input convolved repeatedly with 5x5 filters! (32 -> 28 -> 24 ...).
- Shrinking too fast is not good, doesn't work well.



CNN for Image Classification



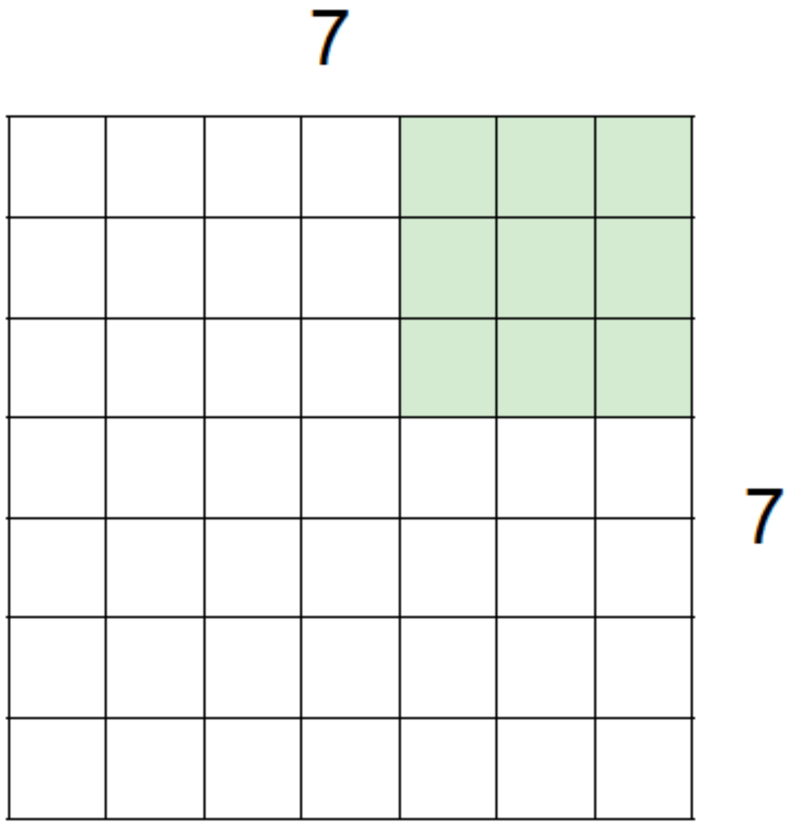
Stride



7x7 input (spatially)
assume 3x3 filter

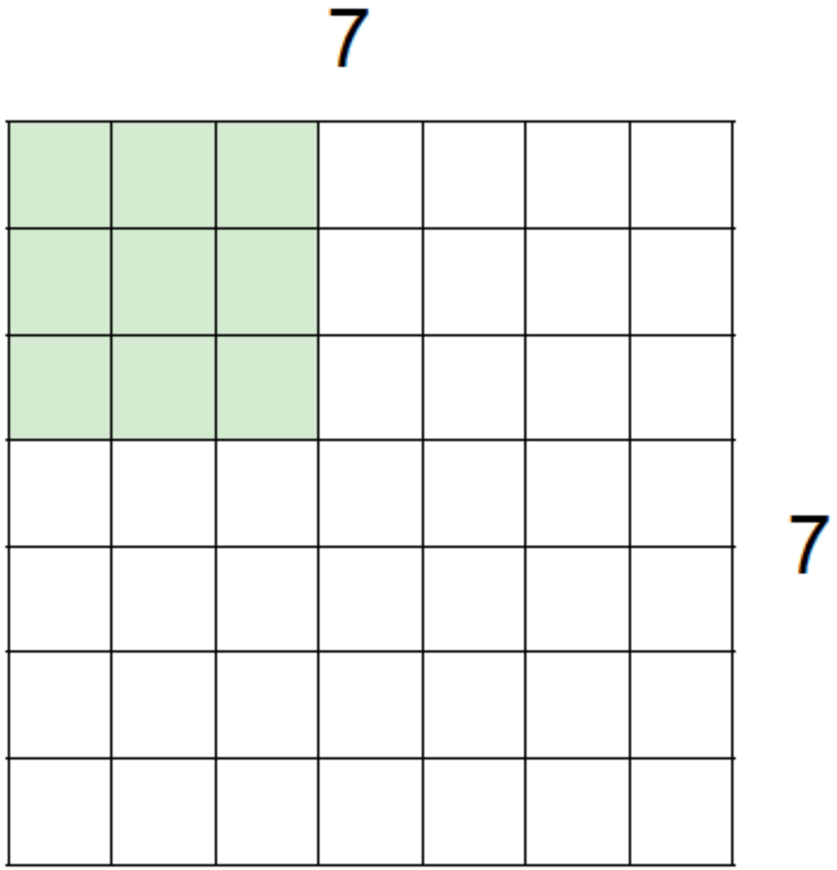
=> 5x5 output

Stride



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

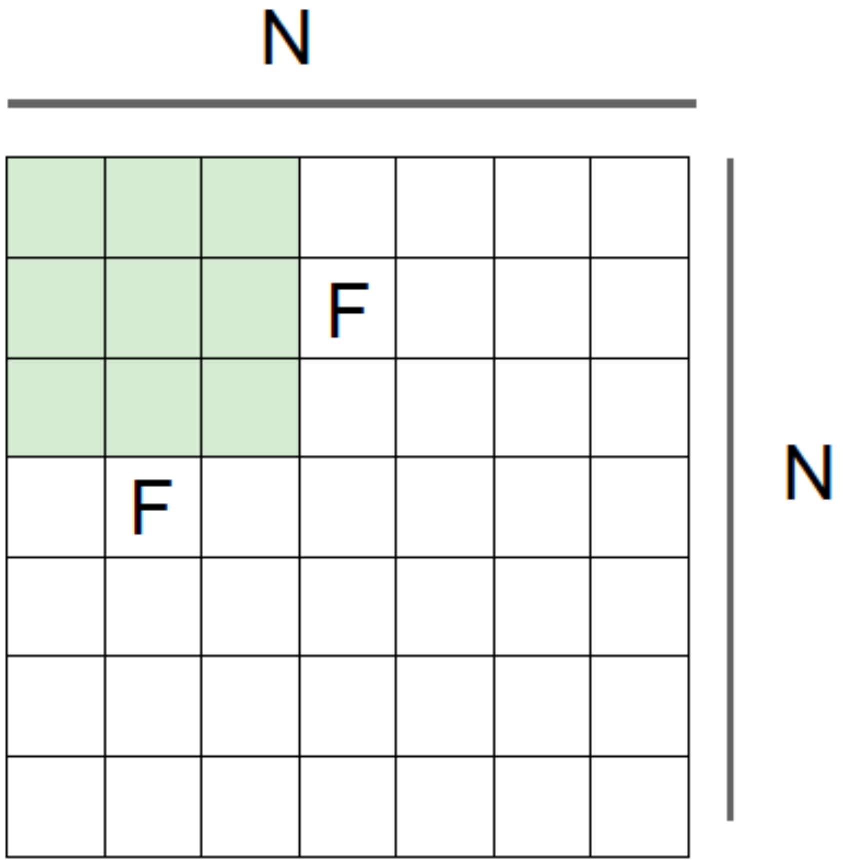
Stride



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

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

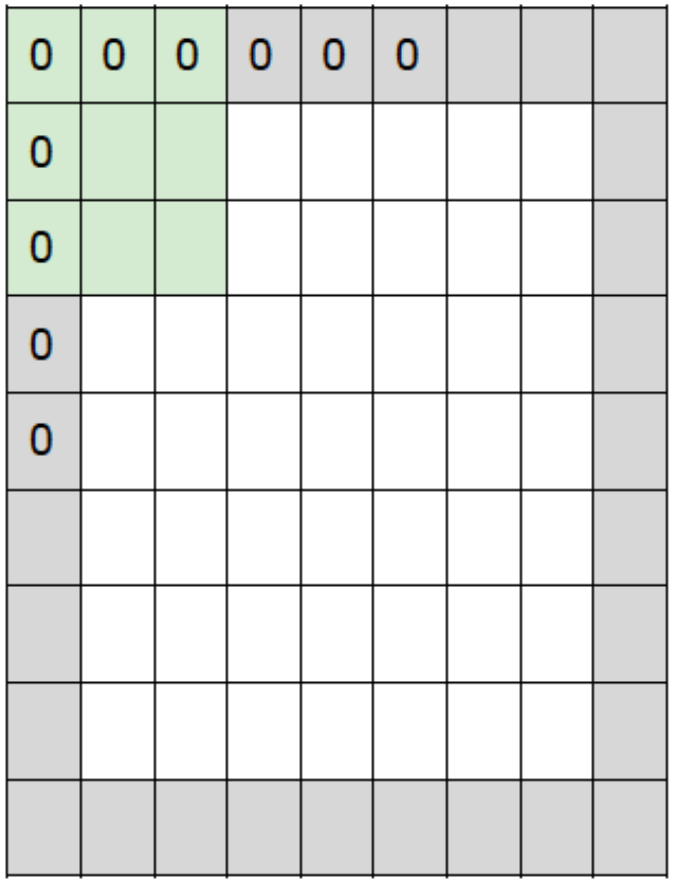
Stride



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)

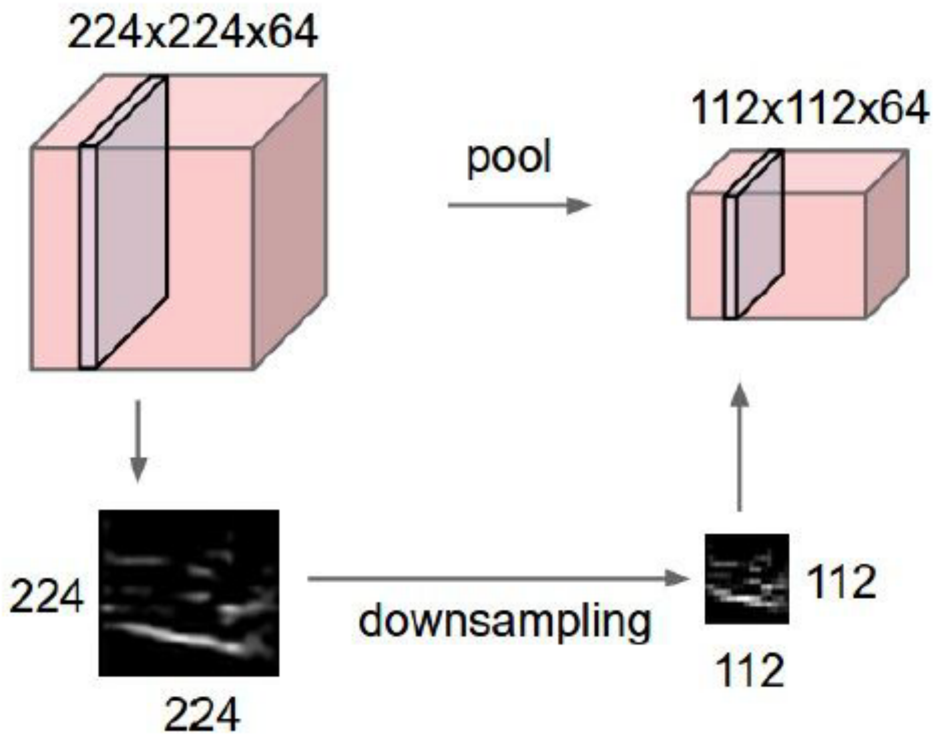
- F = 3 => zero pad with 1 pixel
- F = 5 => zero pad with 2 pixel
- F = 7 => zero pad with 3 pixel

Summary

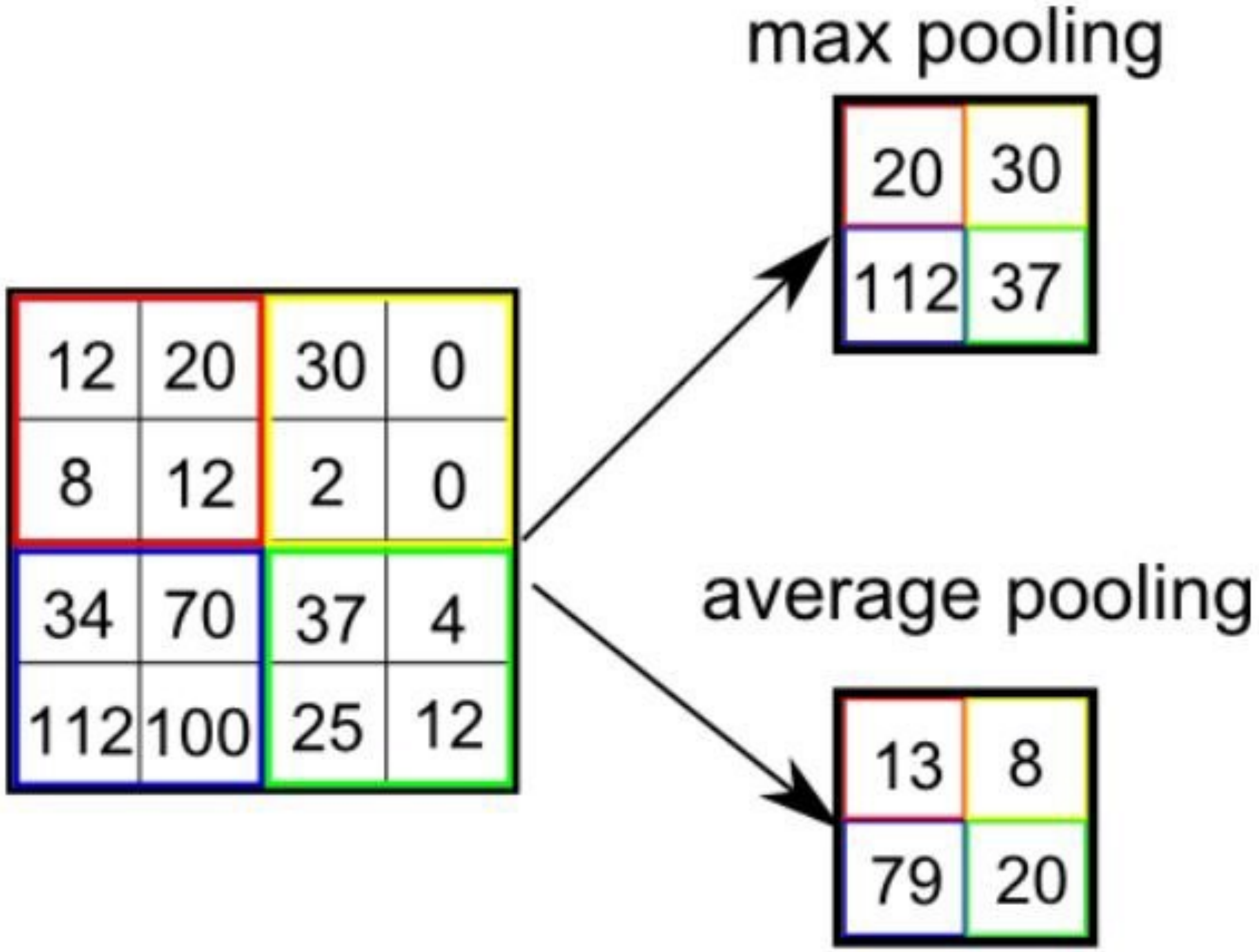
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Pooling Layer

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



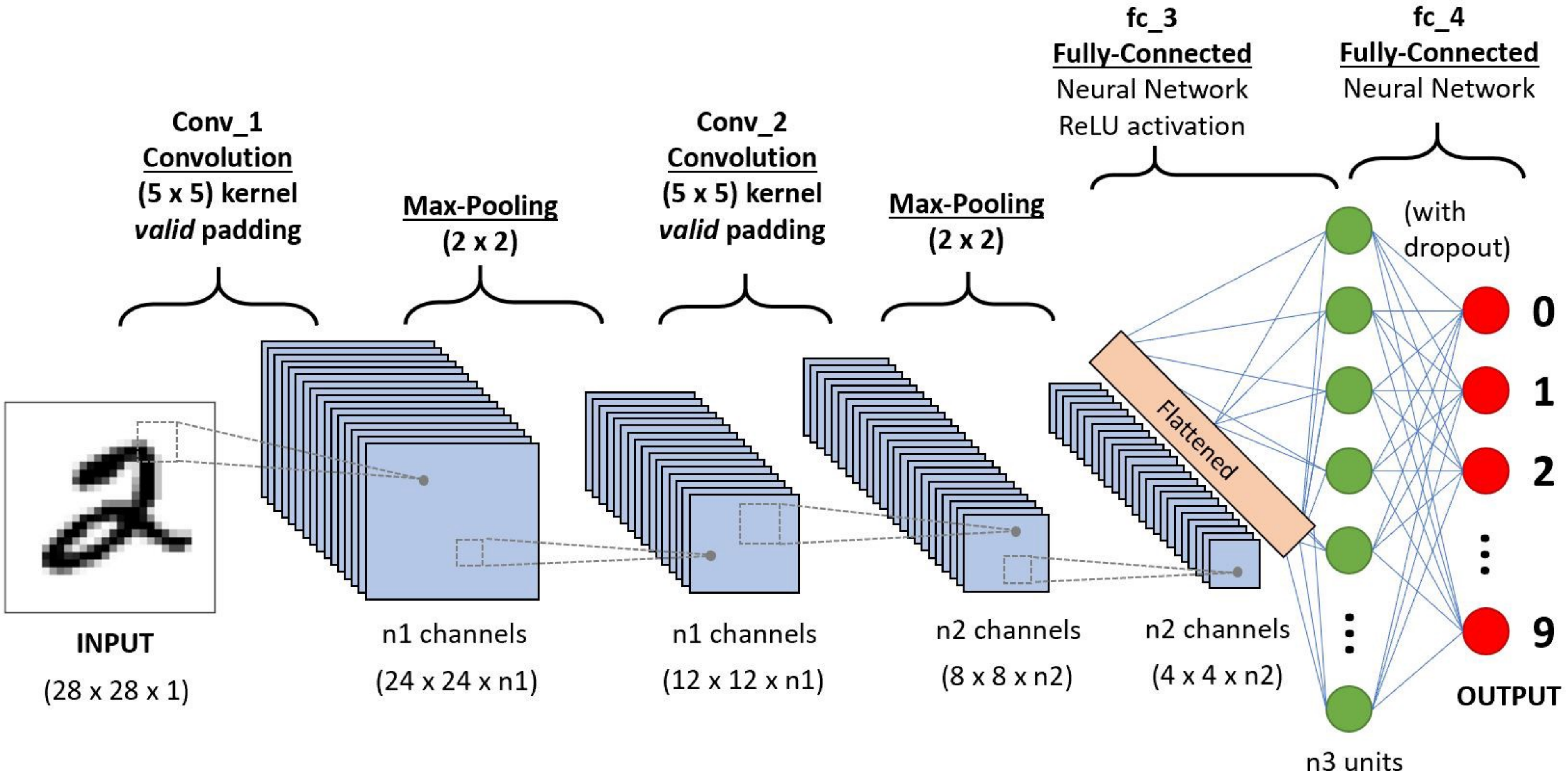
MaxPooling and AvgPoling



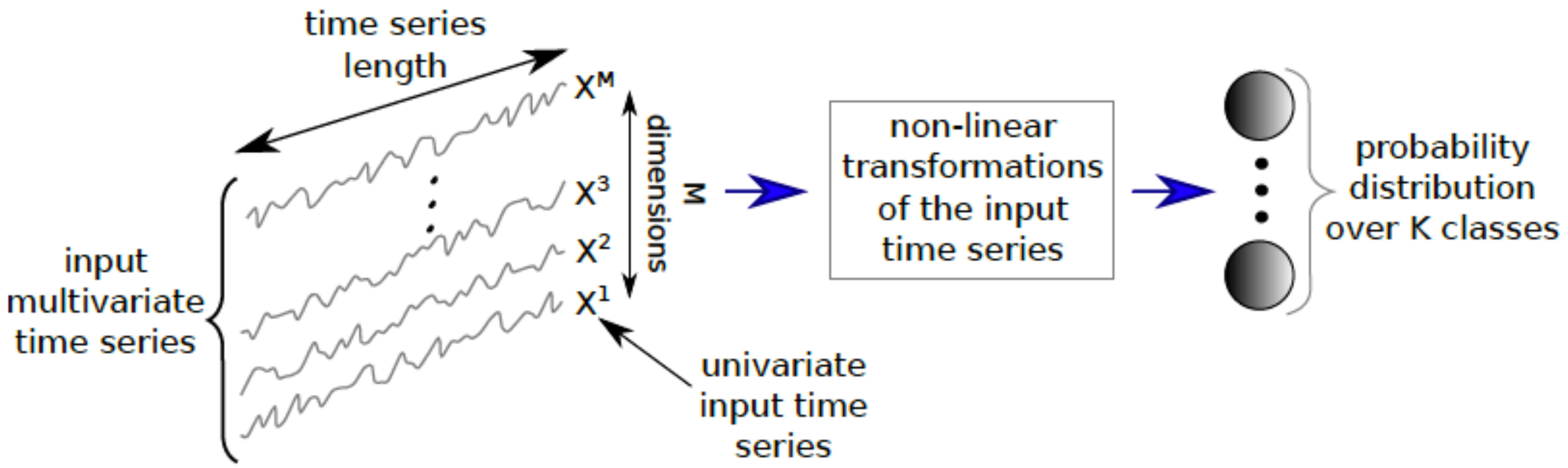
Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Example of CNN

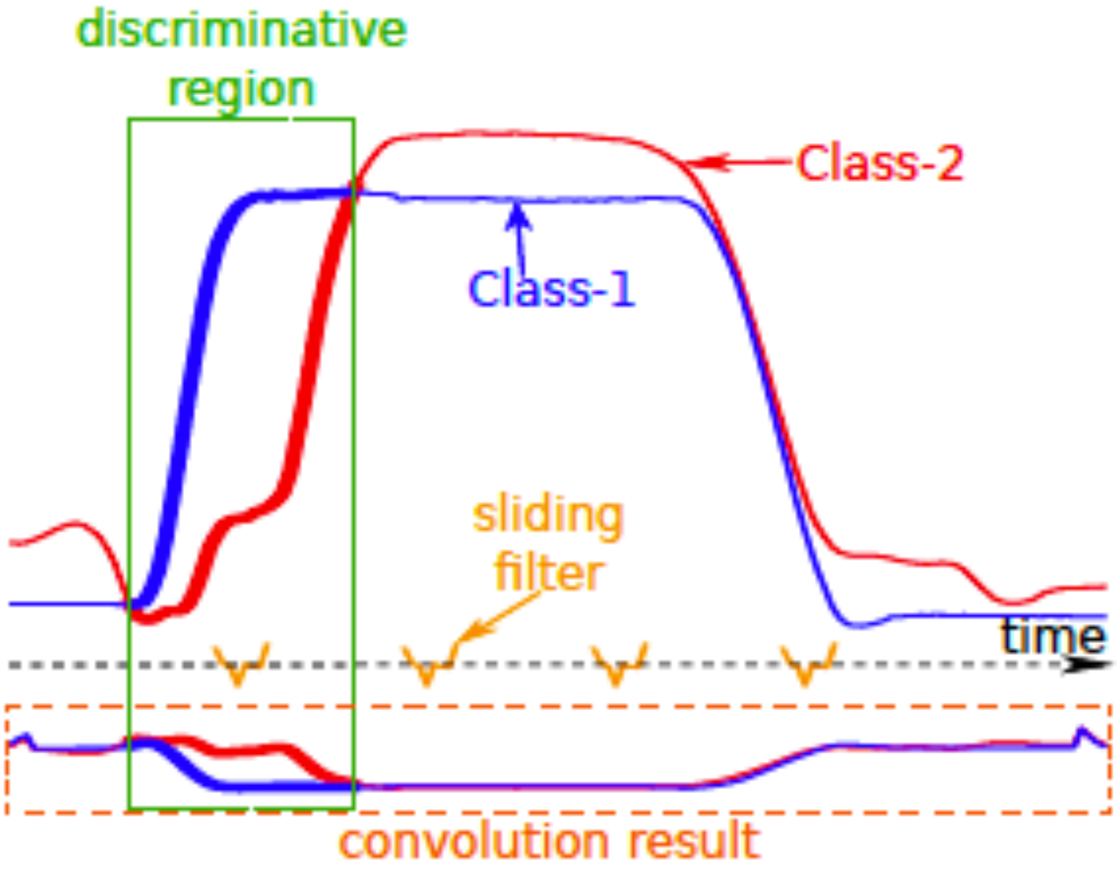


CNN for Time Series Classification

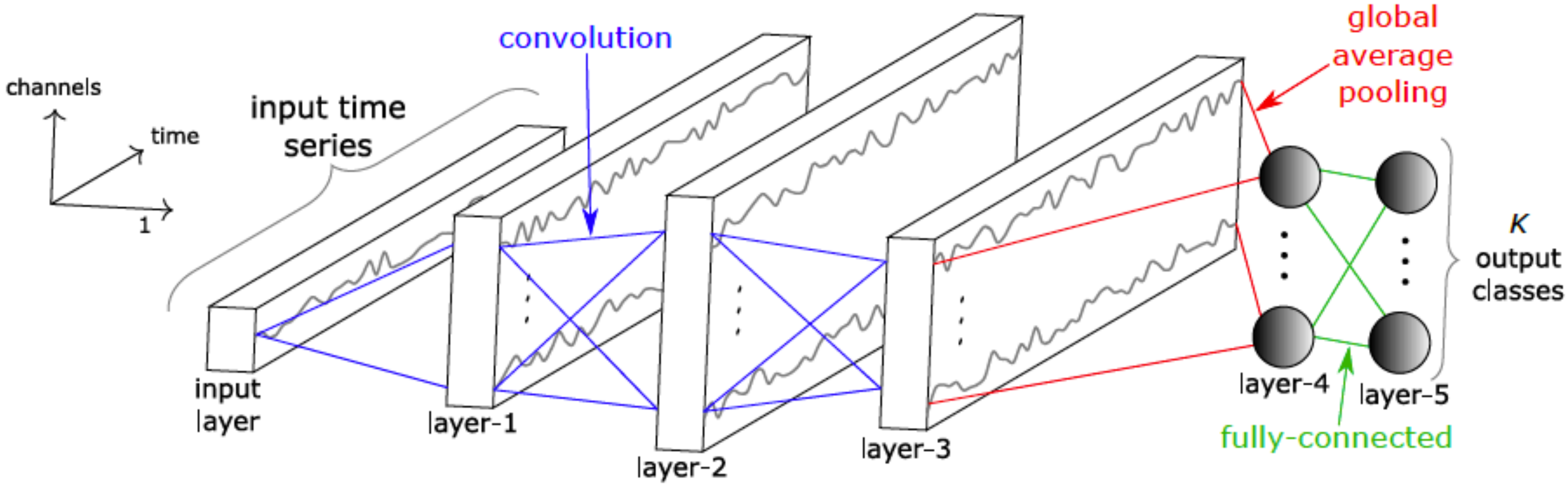


CNN for Time Series Classification

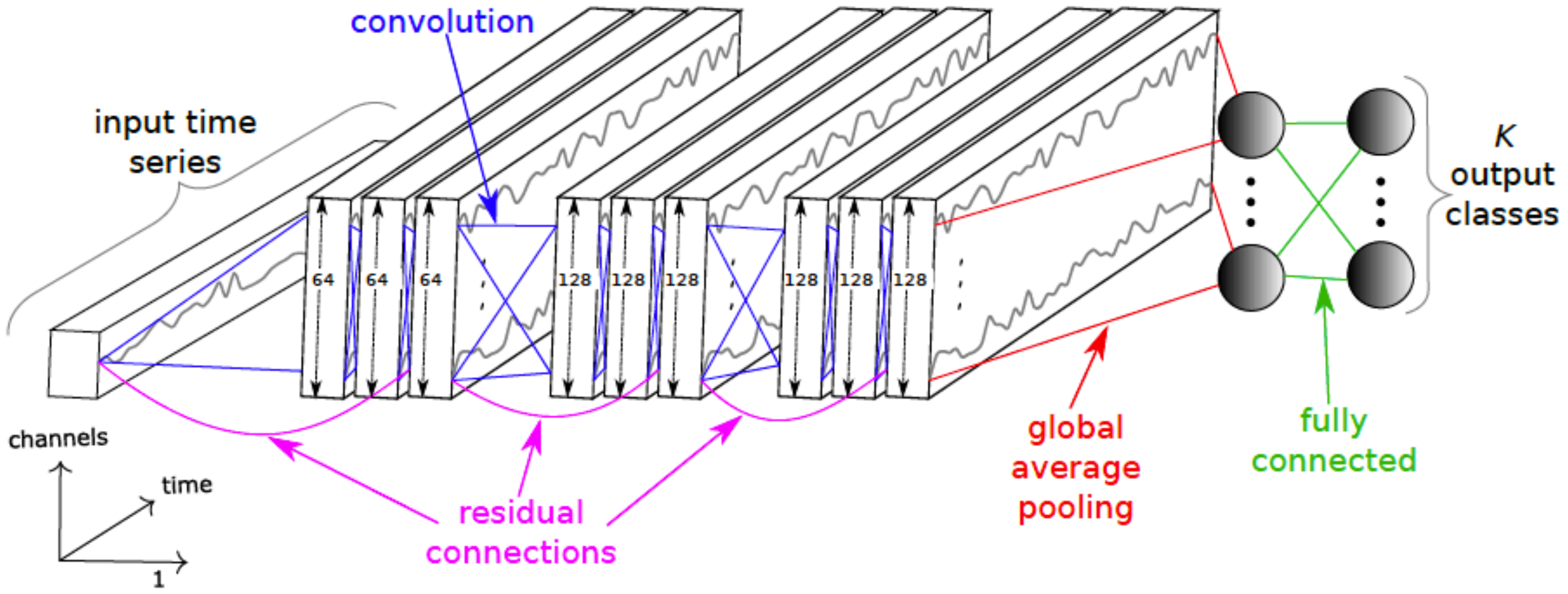
- Result of a applying a learned discriminative convolution.



CNN for Time Series Classification



Residual Nerual Network (ResNN/ResNet)



The main characteristic of ResNets is the shortcut residual connection between consecutive CONV layers. The difference with the usual CNN is that a linear shortcut is added to link the output of a residual block to its input thus enabling the flow of the gradient directly through these connections, which makes training a DNN much easier by reducing the vanishing gradient effect.

CNN Summary

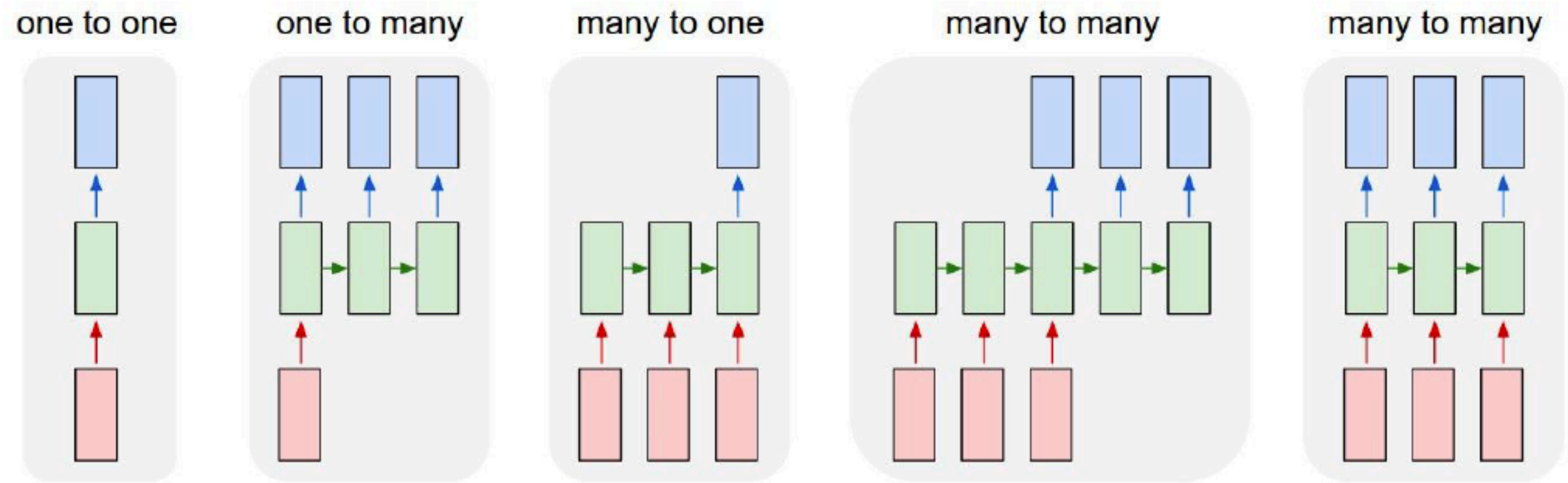
- ConvNets stack Convolutional, Pooling, Fully Connected Layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically CNN looked like
  - [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K, SOFTMAX
  - where N is usually up to ~5, M is large, 0 <= K <= 2.
- Recent advances such as ResNet/GoogLeNet have challenged this paradigm

Recurrent Neural Network

Slides edited from Stanford

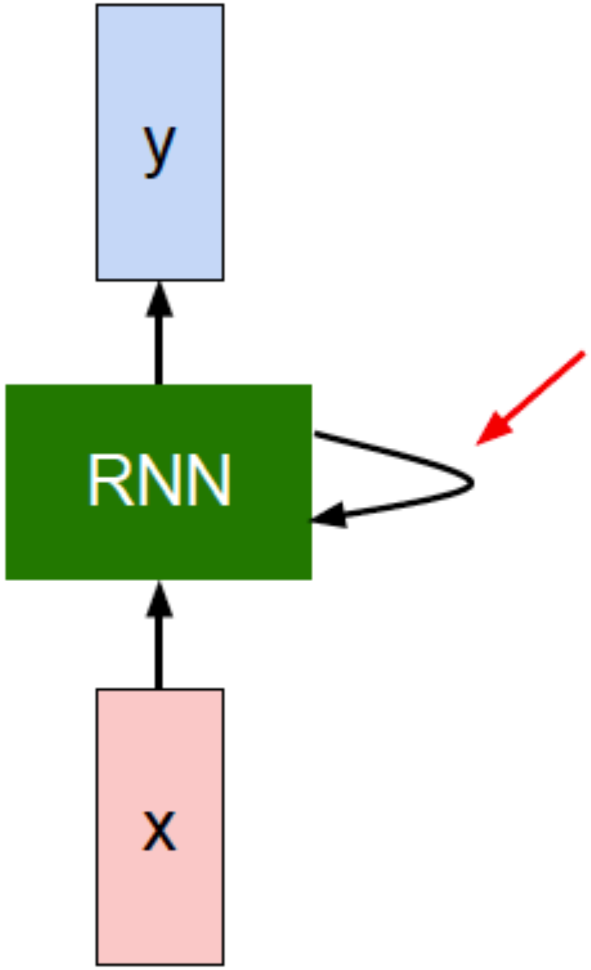
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

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:

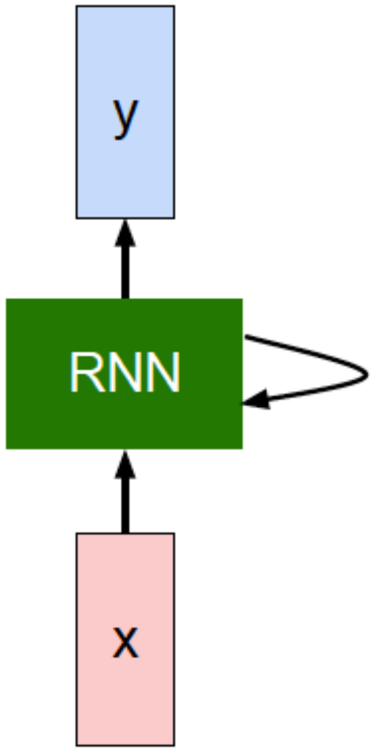
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at
some time step

some function
with parameters W



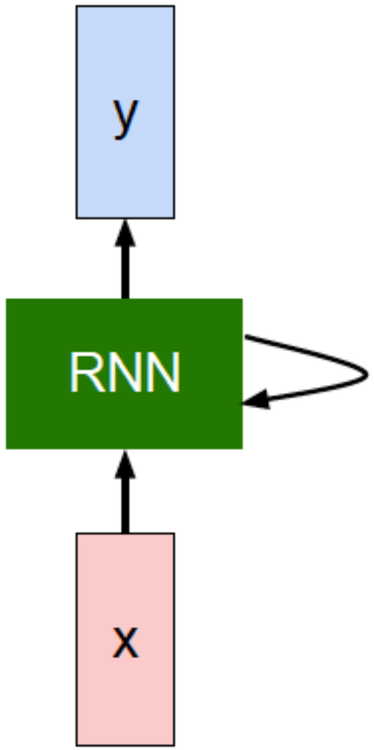
(Simple) Recurrent Neural Network

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

$$\downarrow$$

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

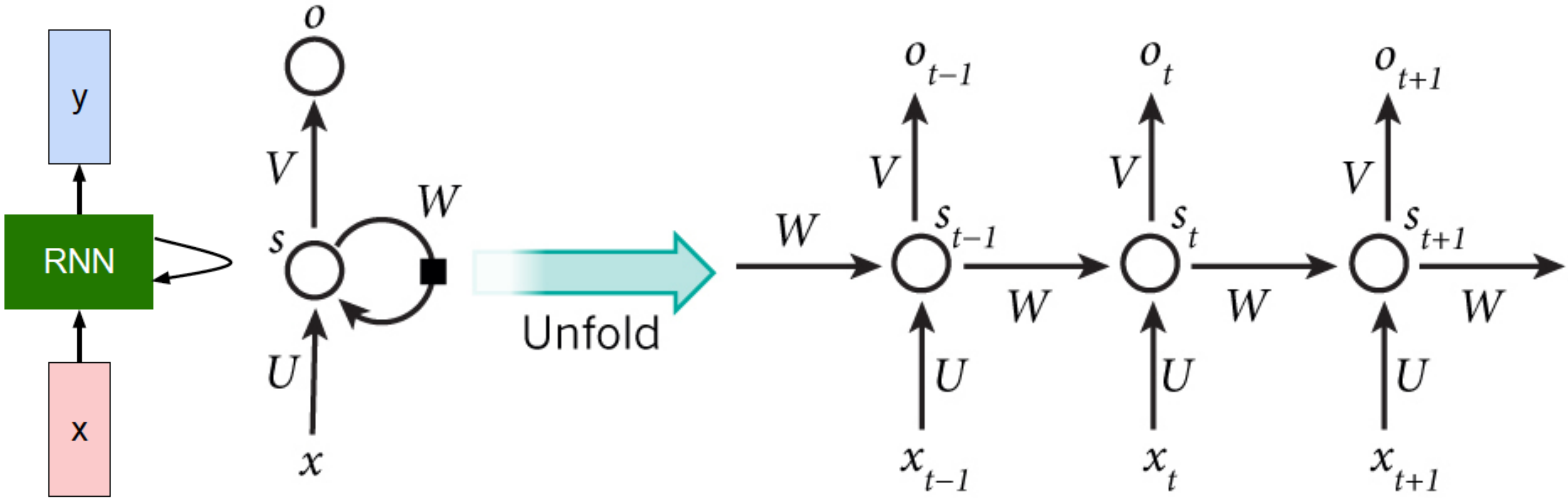
$$y_t = W_{hy}h_t$$



RNN Idea

- The idea behind RNNs is to make use of sequential information.
- In a traditional NN we assume that all inputs (and outputs) are independent of each other.
- But for sequence dependent task this is a bad idea: if you want to predict the next word in a sentence you better know which words came before it.
- RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations.
- Another way to think about RNNs is that they have a “memory” which captures information about what has been calculated so far.
- In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps

Unfolded RNN



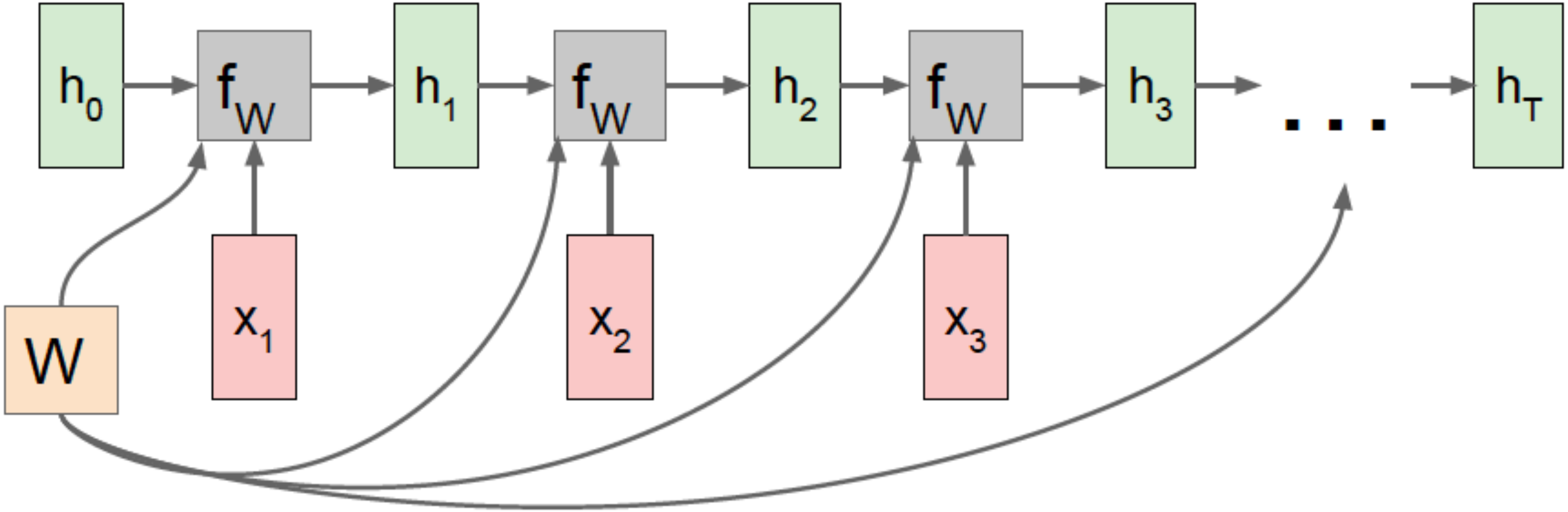
Unfolded RNN

- x_t is the input at time t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time t. It is the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(U x_t + W s_{t-1})$.
- The function f is usually *tanh* or *ReLU*. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at time t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = softmax(Vs_t)$.

Unfolded RNN

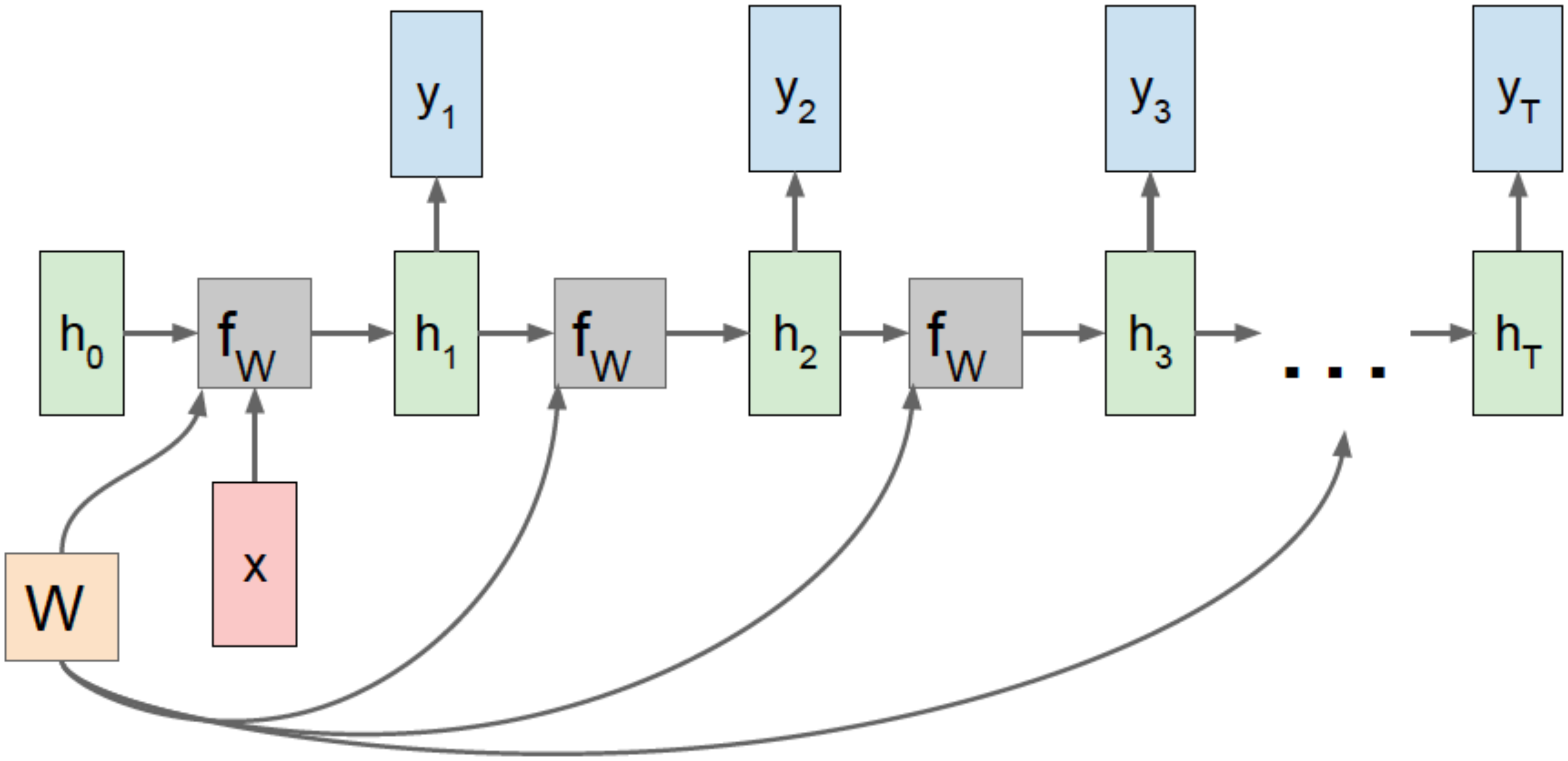
- The hidden state s_t is the memory of the network. s_t captures information about what happened in all the previous time steps.
- The output at step o_t is calculated solely based on the memory at time t.
- s_t typically can not capture information from too many time steps ago.
- Unlike a DNN, which uses different parameters at each layer, a RNN shares the same parameters (U, V, W) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs.
- The previous diagram has outputs at each time step, but depending on the task this may not be necessary.

RNN: Computational Graph

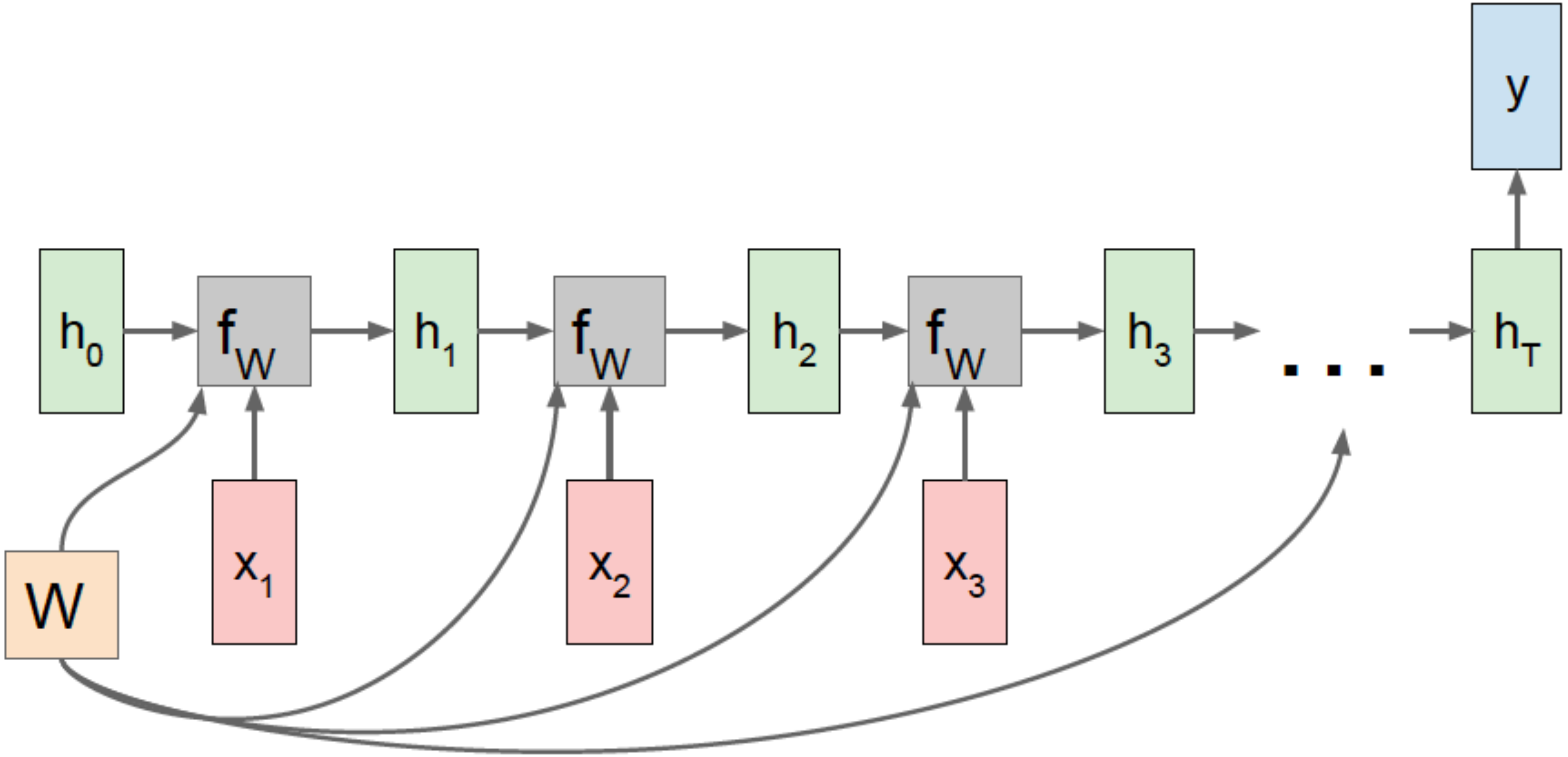


Reminder: Re-use the same weight matrix at every time-step

RNN: Computational Graph: Many to Many



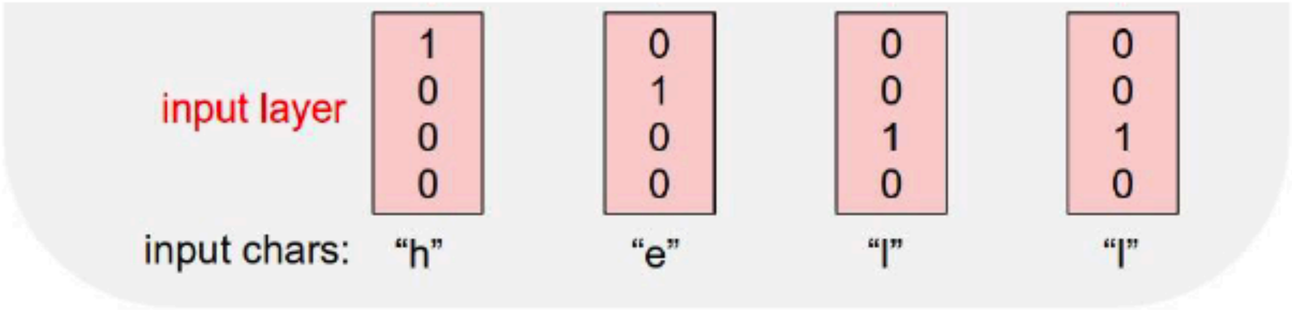
RNN: Computational Graph: Many to One



RNN: Example Training

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



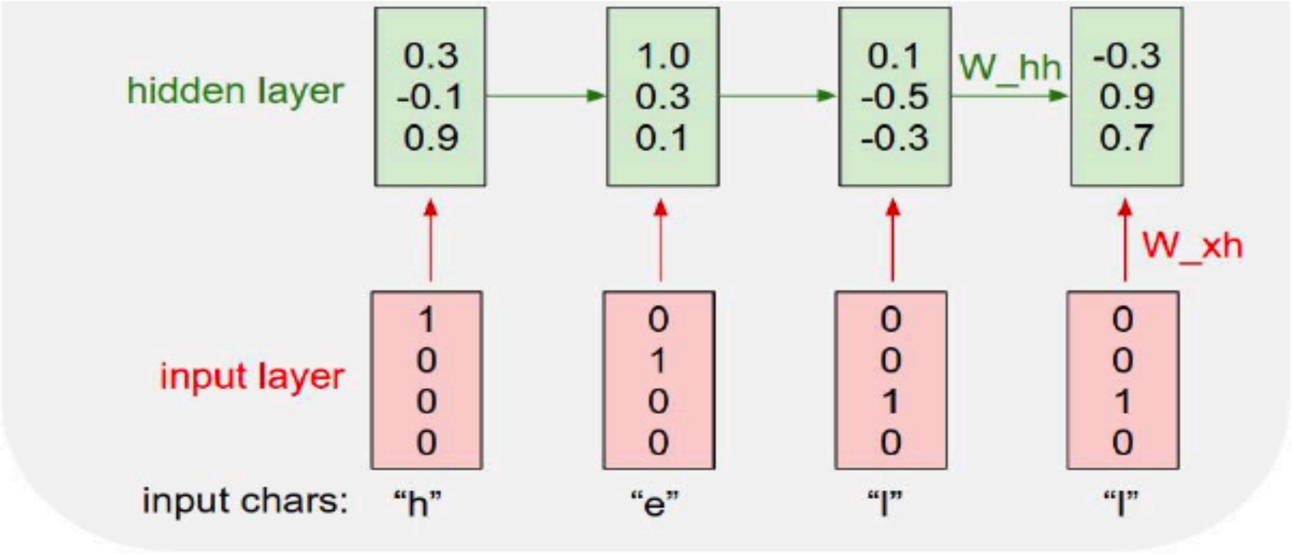
RNN: Example Training

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

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

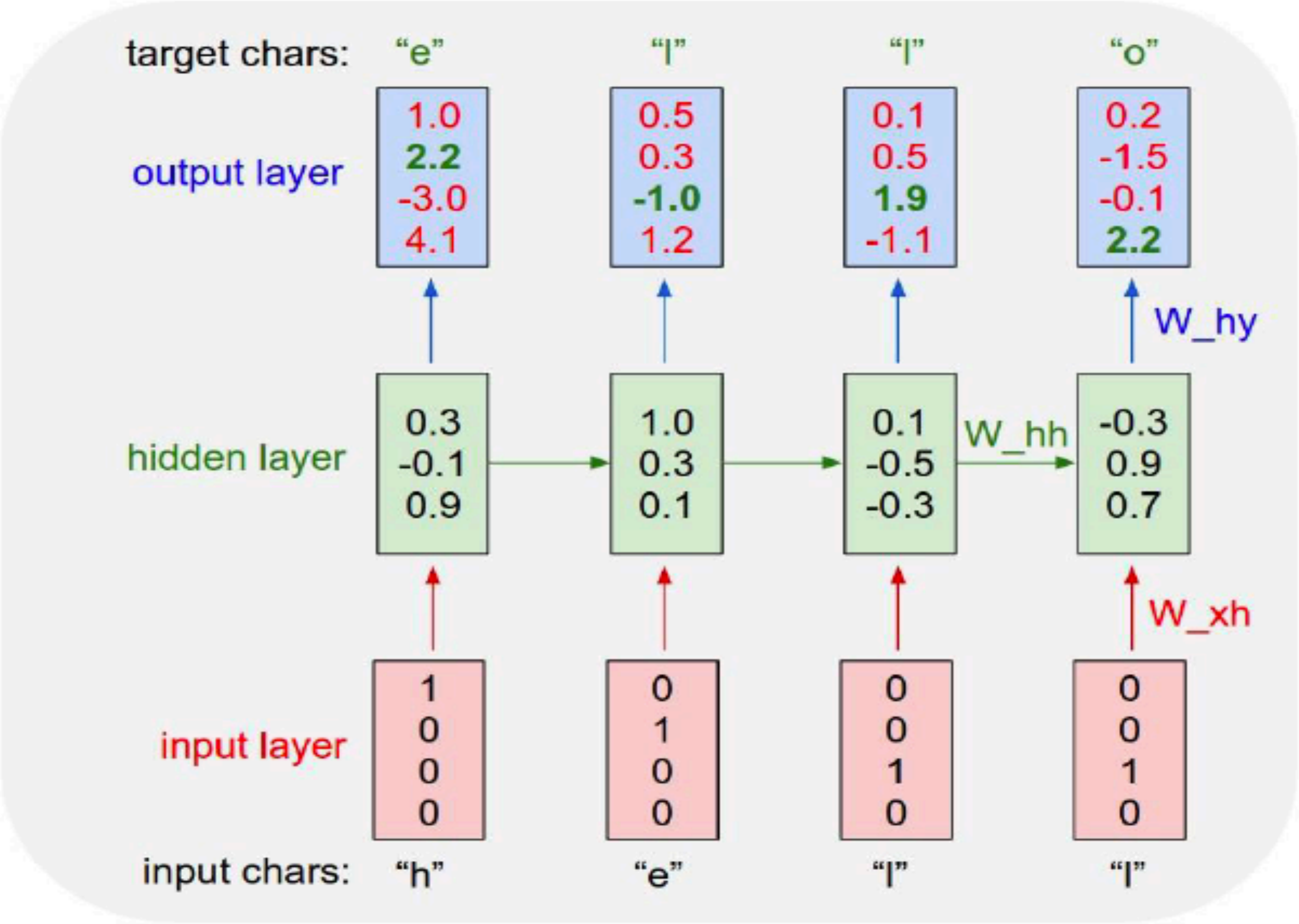


RNN: Example Training

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

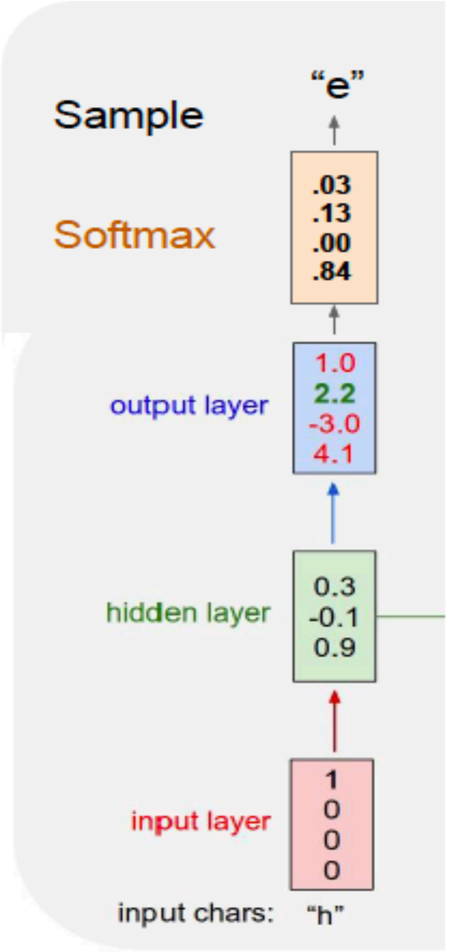


RNN: Example Test

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

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

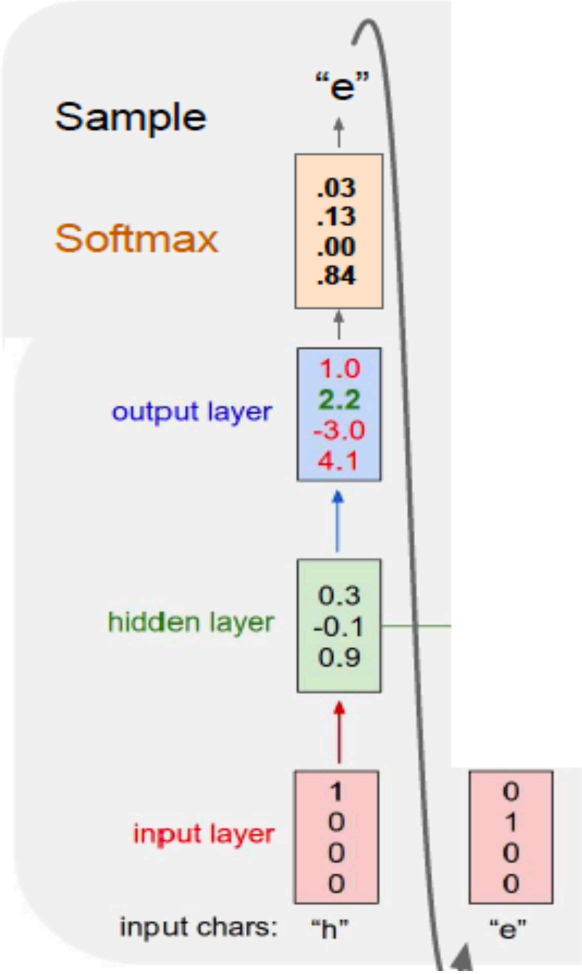


RNN: Example Test

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

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

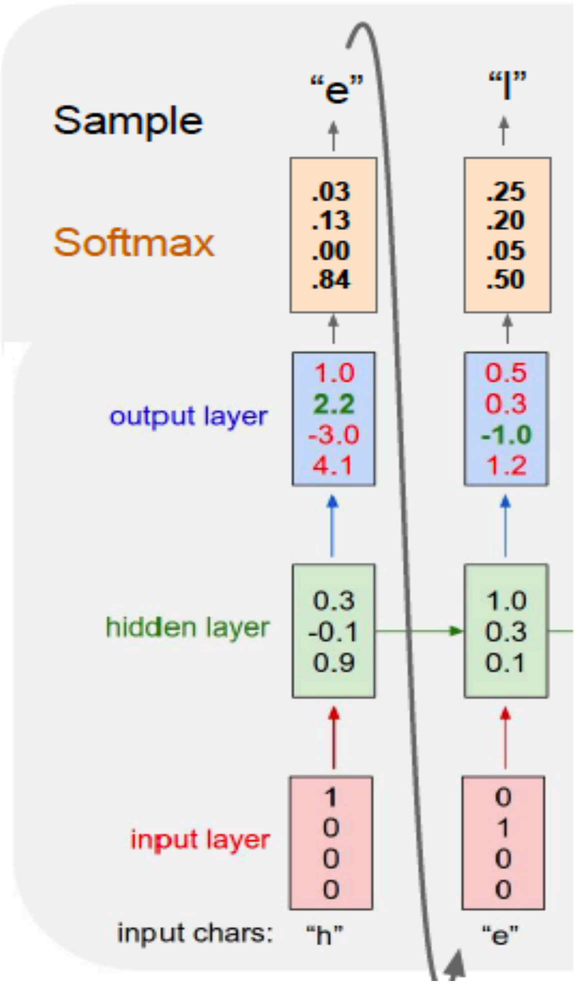


RNN: Example Test

Example:
Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

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

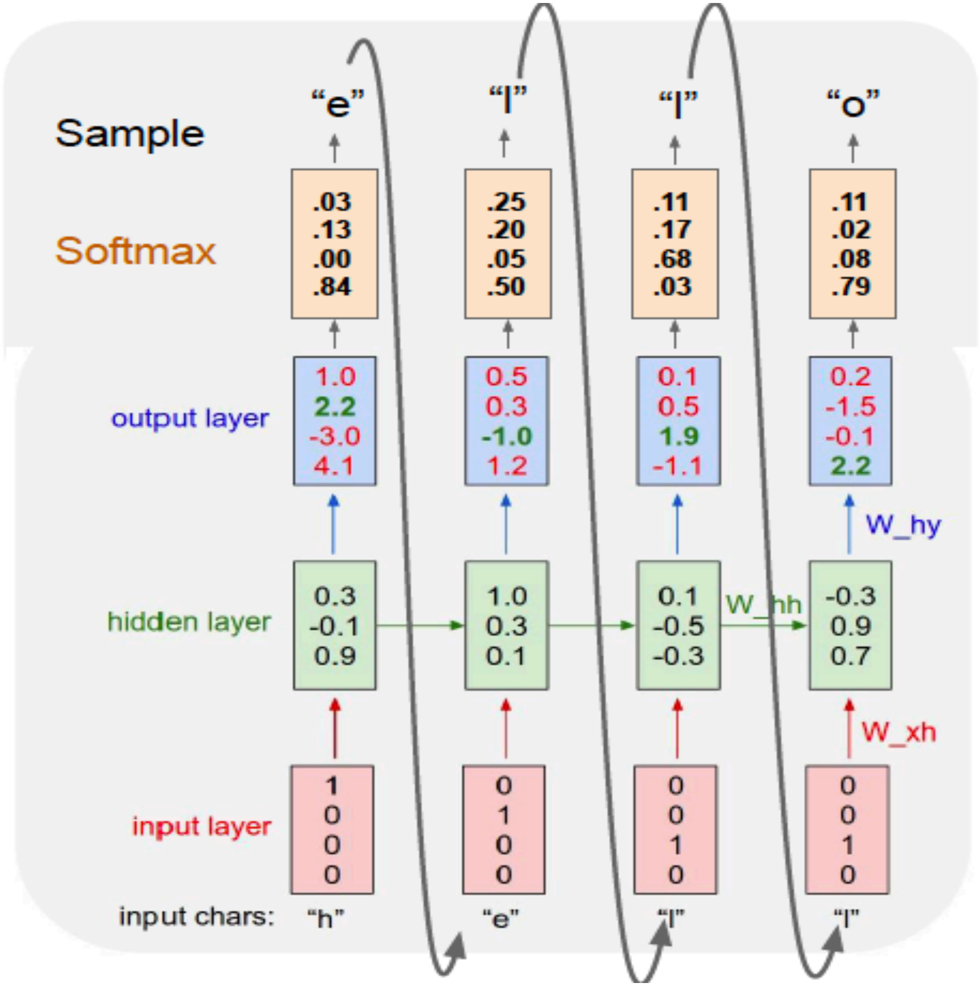


RNN: Example Test

Example:
Character-level Language Model Sampling

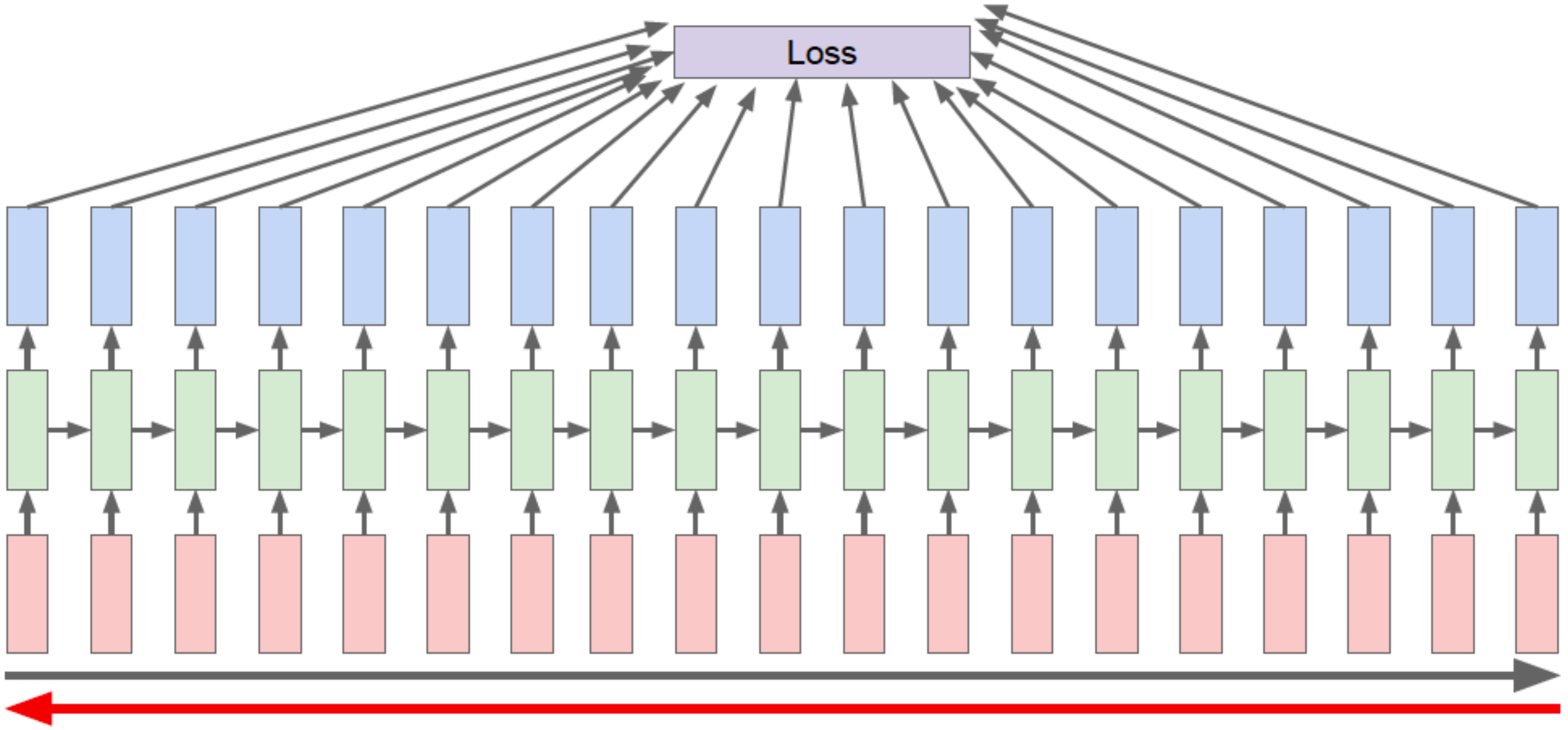
Vocabulary:
[h,e,l,o]

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



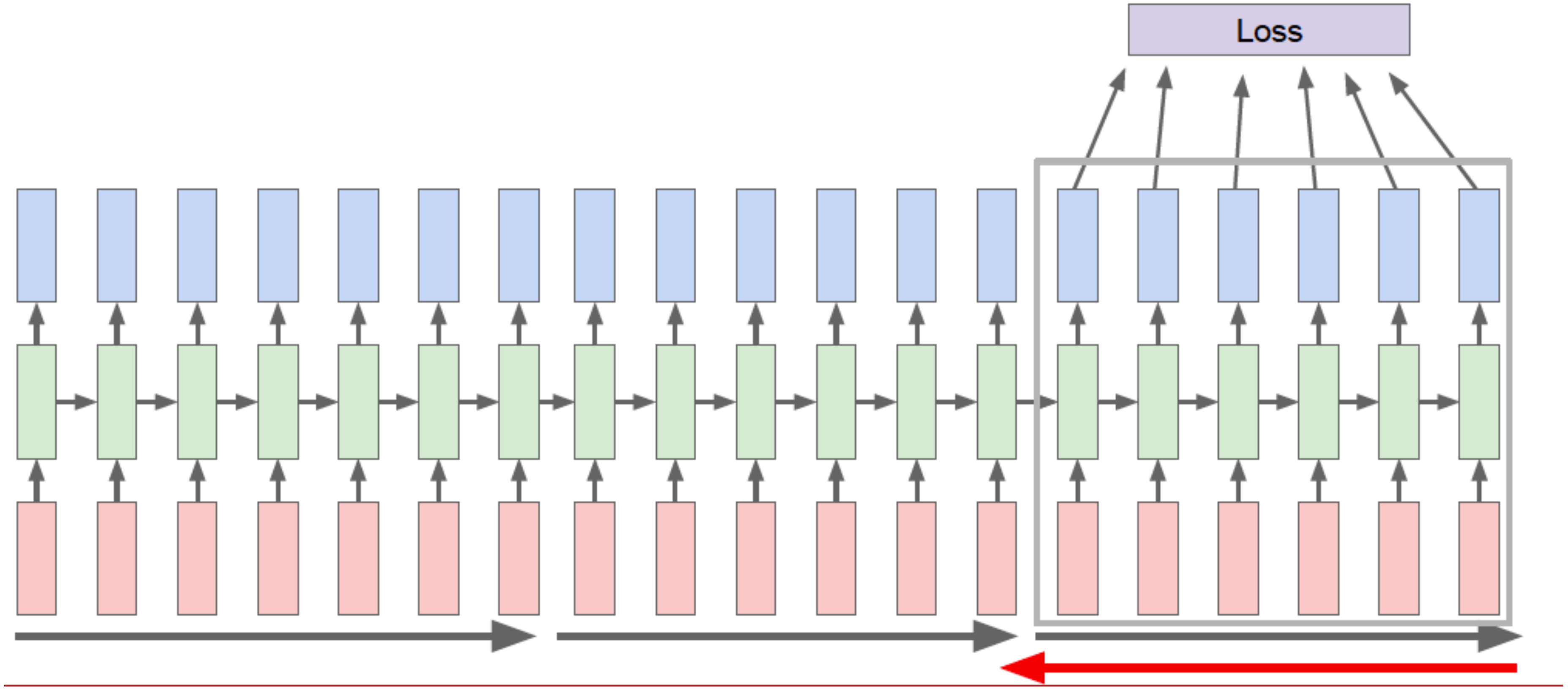
Backpropagation Through Time

- Forward through entire sequence to compute loss
- Then backward through entire sequence to compute gradient



Truncated BPTT

- It is an approximation of full BPTT that is preferred for long sequences since full BPTT's forward/backward cost per parameter update becomes very high over many time steps.
- The downside is that the gradient can only flow back so far due to that truncation, so the network can not learn dependencies that are as long as in full BPTT.

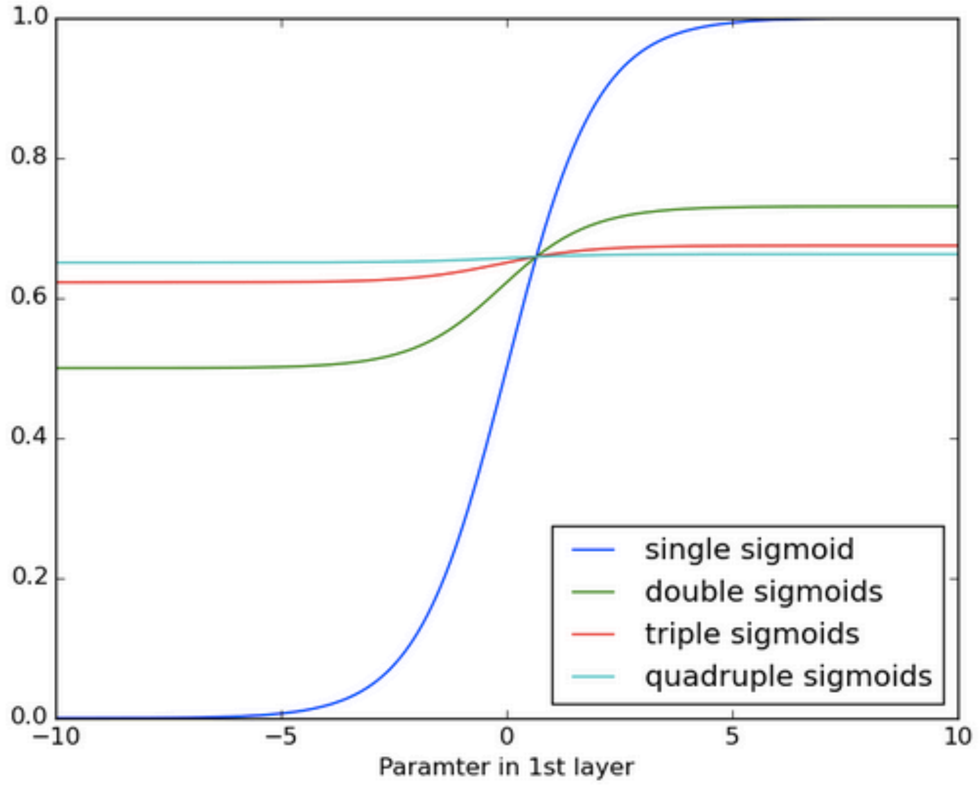


Limitations of RNNs

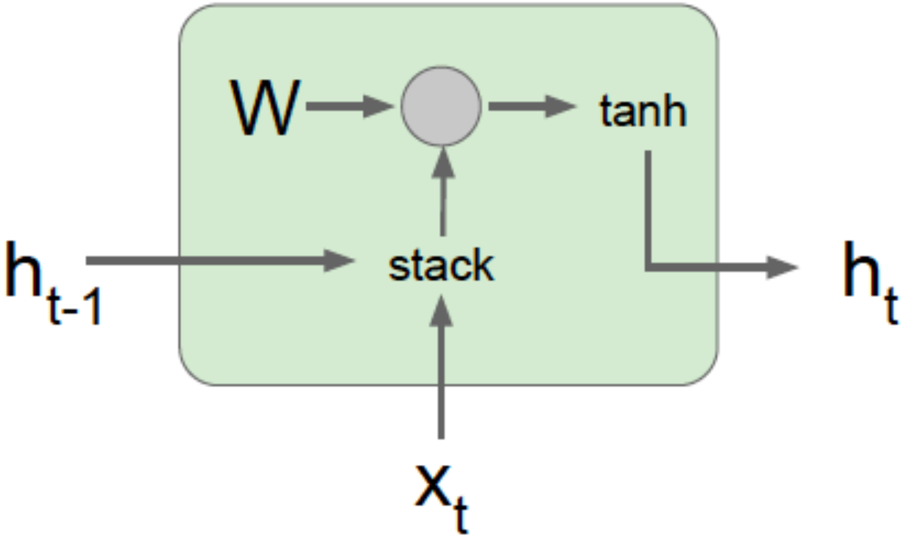
- RNN work fine when we are dealing with *short-term* dependencies.
- However, RNNs fail to understand the context behind an input.
- For instance, something that was said long before, cannot be recalled when making predictions in the present.
- The reason behind this is the problem of *Vanishing Gradient*.
- For a DNN, the weight updating that is applied on a particular layer is a multiple of the learning rate, the error term from the previous layer and the input to that layer. The error for a particular layer is a product of all previous layers’ errors.
- When dealing with functions like *sigmoid/tanh*, the small values of its derivatives (occurring in the error function) gets multiplied multiple times as we move towards the starting layers. As a result of this, the gradient almost vanishes as we move towards the starting layers, and it becomes difficult to train these layers.

Vanishing (and Exploding) Gradients

- The gradient expresses the change in all weights with regard to the change in error.
- If we can not know the gradient, we can not adjust the weights in a direction that will decrease error, and our network ceases to learn.
- Effects of applying a sigmoid function over and over again.



Vanilla RNN Gradient Flow

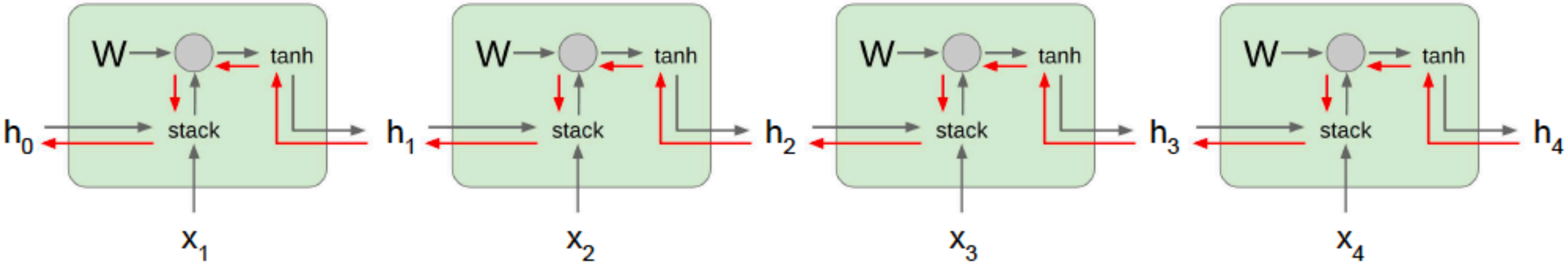


$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{aligned}
$$

Backpropagation from h_t to h_{t-1} multiplies by W

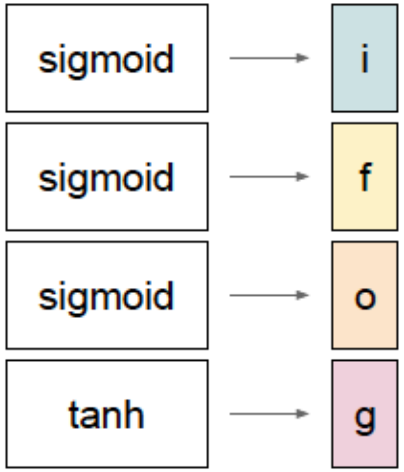
Vanilla RNN Gradient Flow

- Computing gradient of h_0 involves many factors of W (and repeated *tanh*)
- Largest singular value > 1 → **Exploding Gradients**
  - Gradient clipping: Scale Computing gradient gradient if its norm is too big
- Largest singular value < 1 → **Vanishing Gradients**
  - Change RNN architecture



Long Short Term Memory (LSTM)

- LSTM contains in a gated cell information outside the normal flow of the recurrent network.
- Information can be stored in, written to, or read from a cell.



Vanilla RNN

$$\boxed{h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)}$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

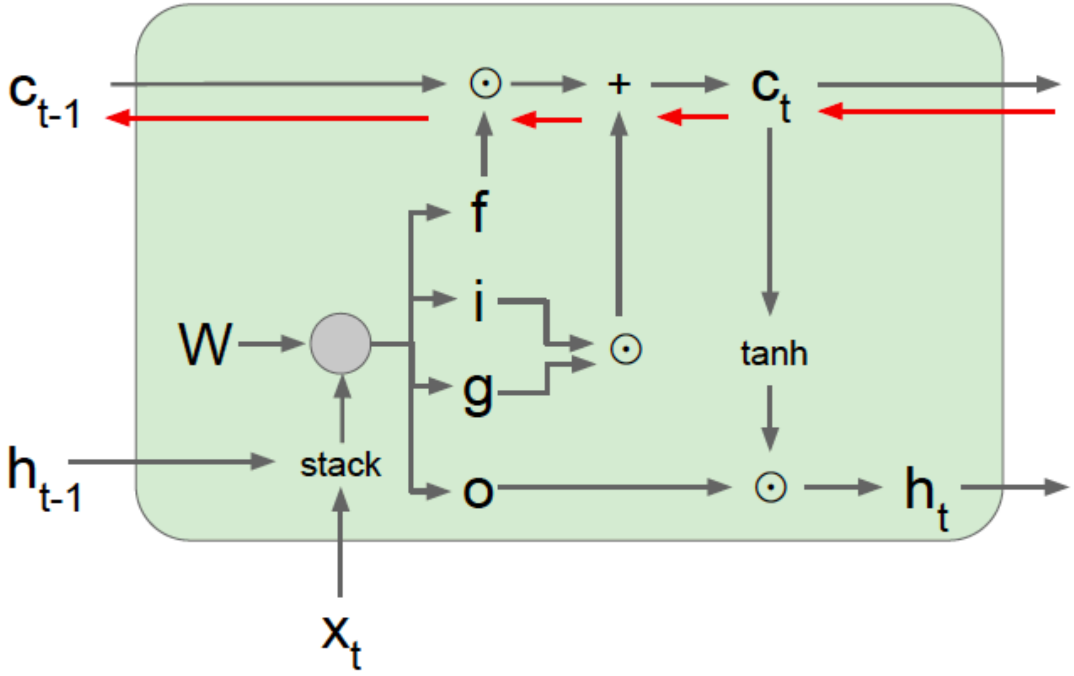
Long Short Term Memory (LSTM)

- The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close.
- These gates are implemented with element-wise multiplication by sigmoids, which are all in the range of 0-1, thus are differentiable and suitable for backpropagation

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

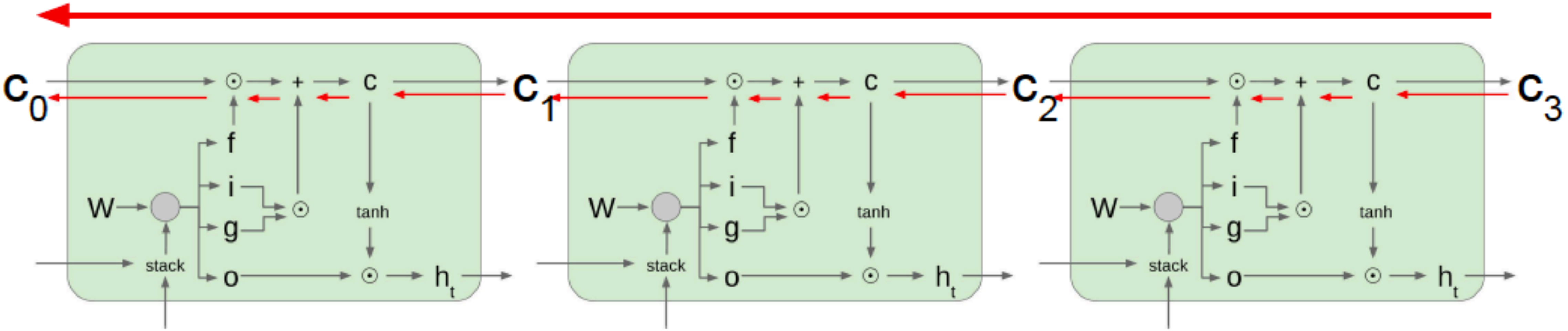
$$h_t = o \odot \tanh(c_t)$$



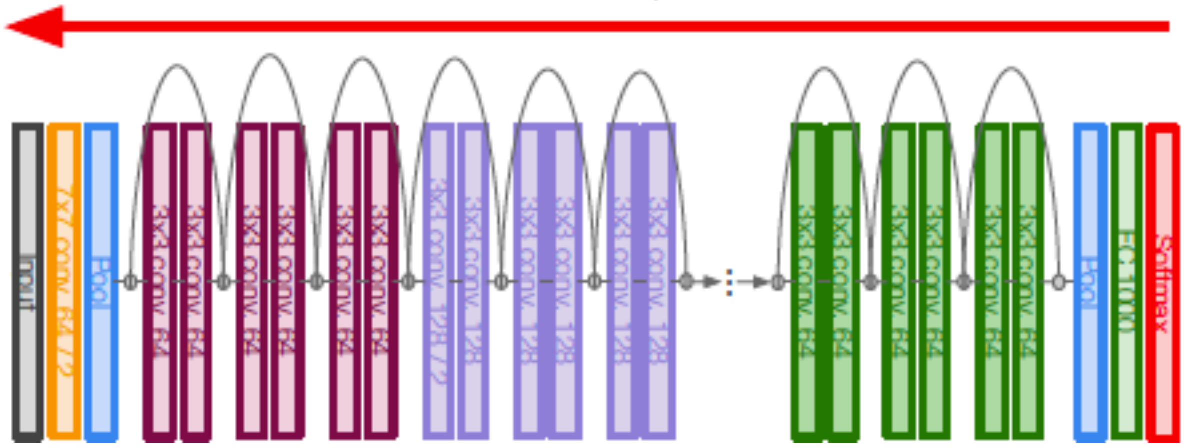
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!



RNN Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences
- Better understanding (both theoretical and empirical) is needed.

References

- Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. Chin-Chia Michael Yeh et al. 1997
- Time Series Shapelets: A New Primitive for Data Mining. Lexiang Ye and Eamonn Keogh. 2016.
- Josif Grabocka, Nicolas Schilling, Martin Wistuba, Lars Schmidt-Thieme (2014): Learning Time-Series Shapelets, in Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2014
- Deep learning for time series classication: a review. Hassan Ismail Fawaz et al. 2019.