Spatio-Temporal Deep Neural Networks for Extreme Event Detection

Miguel-Ángel Fernández-Torres

miguel.a.fernandez@uv.es
https://miguelangelft.github.io/

Image and Signal Processing - ISP

IPL Image Processing Laboratory

Vniversitat de València

XAIDA

XAIDA Summer School 2022
21st June 2022

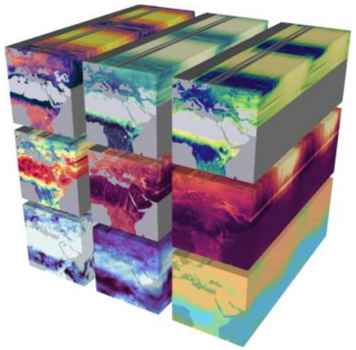
Contents

PyTorch

1. Extreme Event Detection using DNNs

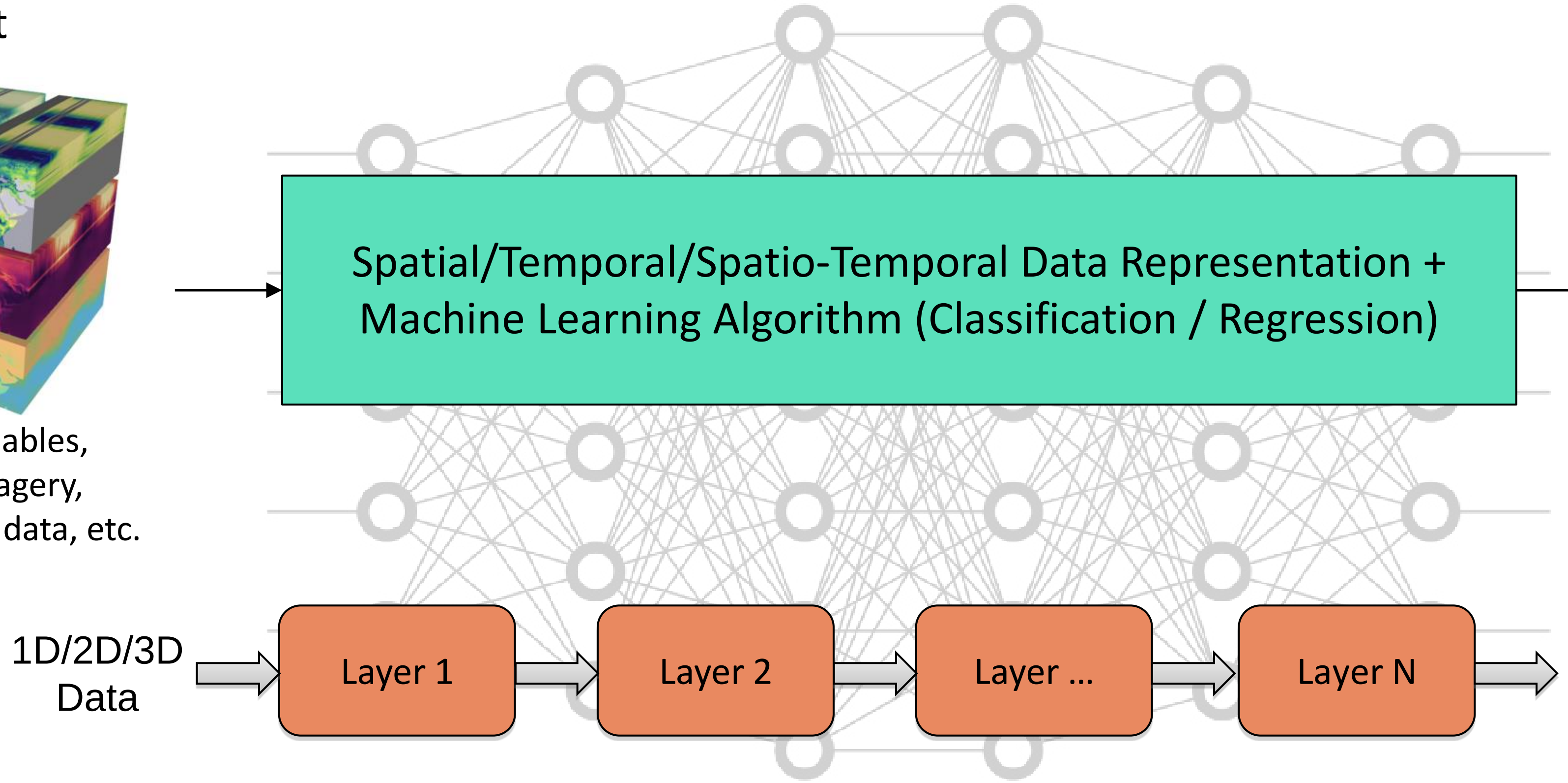
Extreme Event Detection using DNNs

Input

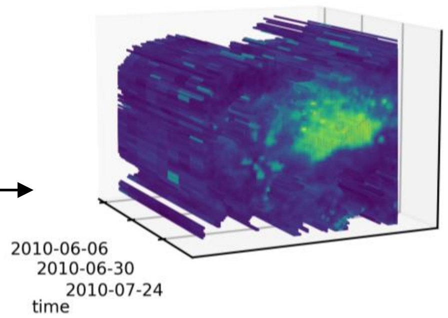


Source: Mahecha et al., 2020

Climate variables,
satellite imagery,
location-aware data, etc.



Output



Extreme event detection

- From classical Machine Learning/Computer Vision to Deep Learning

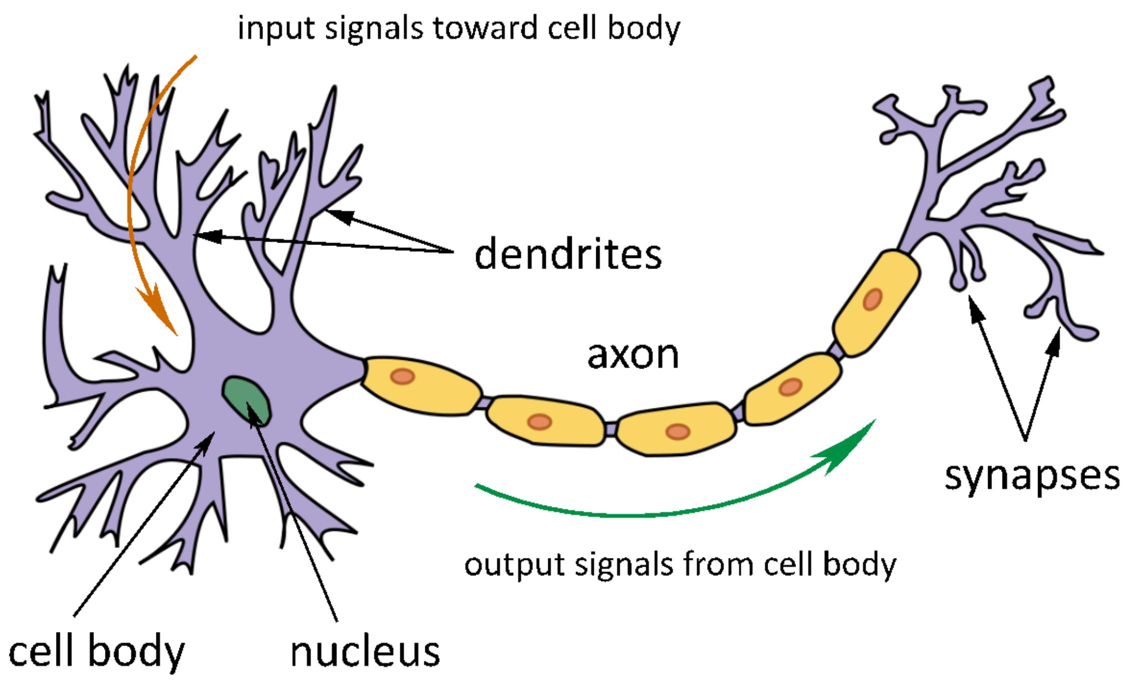
→ From feature engineering (extraction + selection) to feature learning

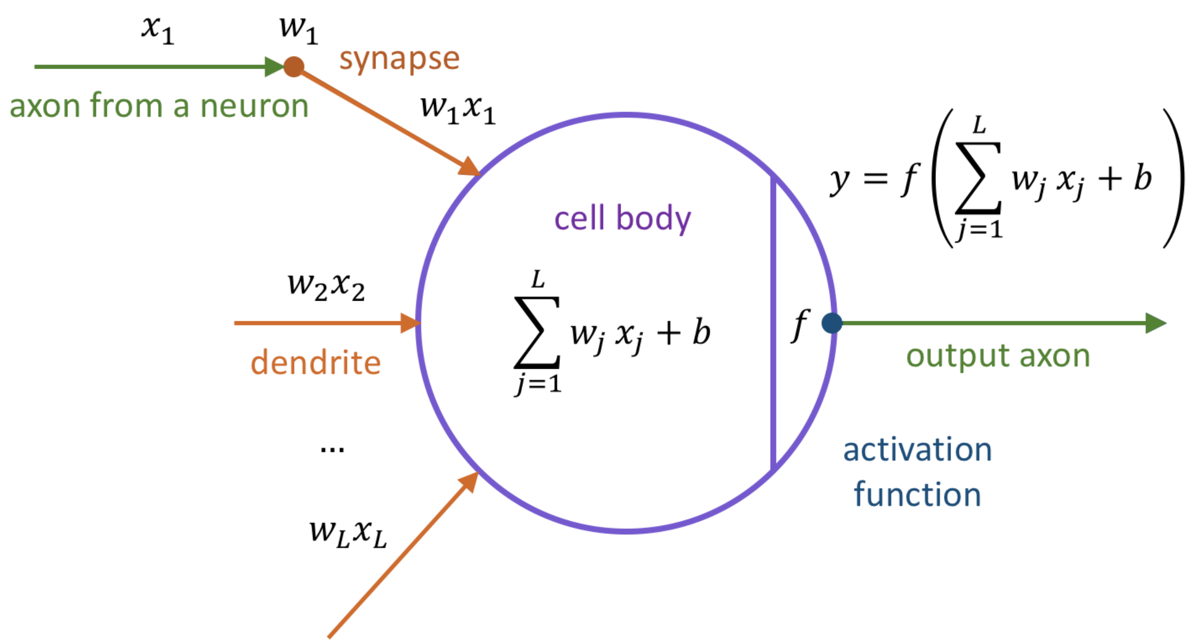
- Spatial/temporal/spatio-temporal hierarchical representations from 1D/2D/3D data to classifier/regressor.
- Each layer extract features from the output of the previous layer.
- End-to-End Learning: Train all layers jointly.

2. Deep Neural Networks (DNNs)

Deep Neural Networks (DNNs)

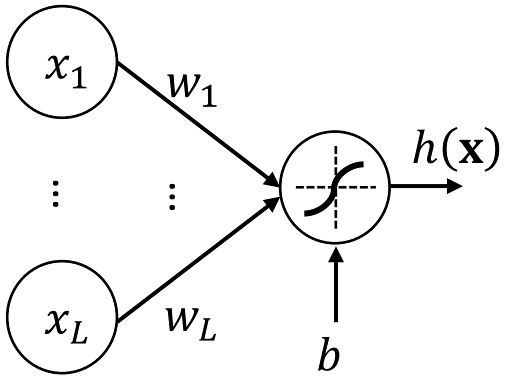
Neuron Model: Biological vs. Computational Neuron





From Linear Regression to Neural Networks

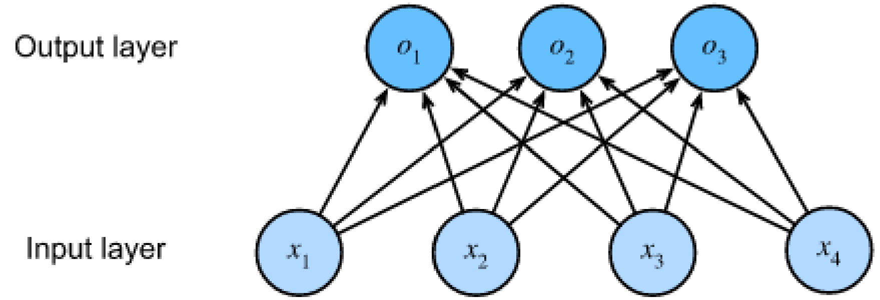
- **Linear regression: Single-layer neural network**



  - Pre-activation: $a(\mathbf{x}) = \sum_i w_i \, x_i + b = \mathbf{w}^T\mathbf{x} + b$
  - Activation: $h(\mathbf{x}) = g\big(a(\mathbf{x})\big) = g(\sum_i w_i x_i + b)$
  - Fully-connected/dense layer: Every input connected to every output

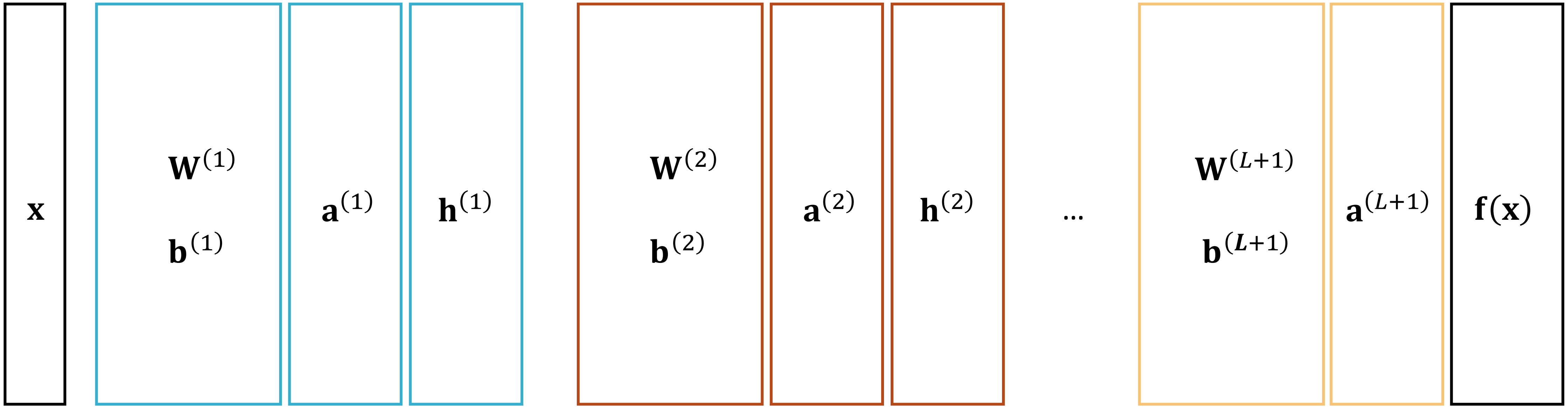
- **Classification/Softmax regression: Single-layer neural network**

$\hat{\boldsymbol{y}} = softmax(\mathbf{Wx} + \mathbf{b}) \quad \boldsymbol{y} \in \mathbb{R}^C, \mathbf{x} \in \mathbb{R}^L, \mathbf{W} \in \mathbb{R}^{C\times L}, \mathbf{b} \in \mathbb{R}^C$



  - L features or descriptors $\mathbf{x}$
  - C classifiers, one per category, evaluated in parallel
  - Objective: Given training set, learn $\boldsymbol{W}$ (weights or parameters) and $\mathbf{b}$ (bias term)
  - Once weights are learned, the set of training data can be discarded
  - Predicted scores for each class $\hat{\boldsymbol{y}}$

Multi-layer Neural Network – Feedforward Propagation



$$\mathbf{a}^{(1)}(\mathbf{x}) = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{a}^{(2)}(\mathbf{x}) = \mathbf{W}^{(2)}\mathbf{h}^{(1)}(\mathbf{x}) + \mathbf{b}^{(2)}$$

$$\mathbf{a}^{(L+1)}(\mathbf{x}) = \mathbf{W}^{(L+1)}\mathbf{h}^{L}(\mathbf{x}) + \mathbf{b}^{(L+1)}$$

$$\mathbf{h}^{(1)}(\mathbf{x}) = \mathbf{g}\left(\mathbf{a}^{(1)}(\mathbf{x})\right)$$

$$\mathbf{h}^{(2)}(\mathbf{x}) = \mathbf{g}\left(\mathbf{a}^{(2)}(\mathbf{x})\right)$$

$$\mathbf{f}(\mathbf{x}) = \mathbf{o}\left(\mathbf{a}^{(L+1)}(\mathbf{x})\right)$$

Why Deep Neural Networks?

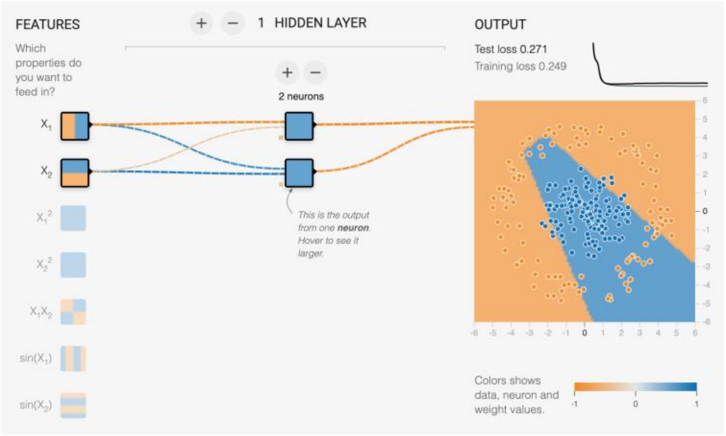
- There are functions you can compute with “deep” NNs that shallower NNs would require exponentially more hidden units
- More layers, less hidden units per layer
- Hierarchical representations: From less to more abstract concepts

`torch.nn.Linear`

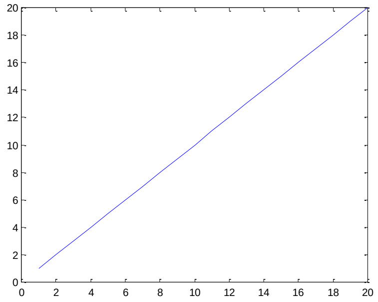
Activation Functions

`nn.Sigmoid | nn.Tanh | nn.ReLU`

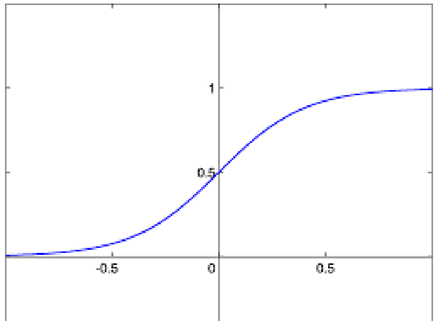
- A Neural Network Playground: https://playground.tensorflow.org/
- A neural network solves linear problems
- Non-linear problems
  - Designing more suitable features (e.g. polynomial features).
  - Building more complex networks, combining several neurons.
  - Activation functions



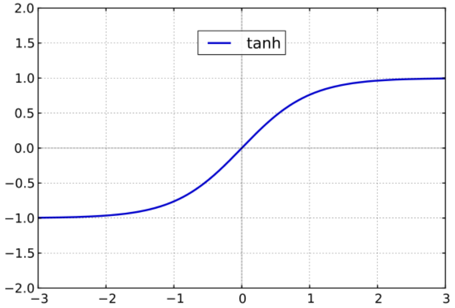
Linear
$g(x) = x$



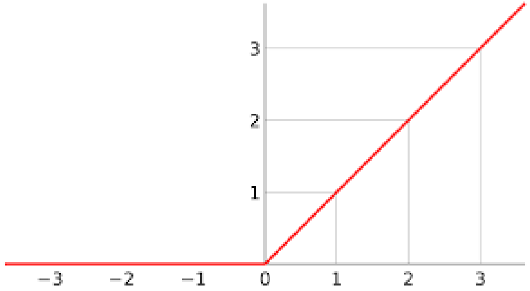
Logistic or Sigmoid
$$g(x) = \frac{1}{1 + e^{-x}}$$



Hyperbolic tangent
$g(x) = \tanh(x)$



RELU
$g(x) = max(0, x)$



Binary classification, Multi-class classification → Softmax

Most hidden layers

“How to Train your (Deep) Neural Network”

Loss Function, cost function, objective l

- Quantitatively determines how accurate the output of the model is.
- Objective: Minimize empirical risk $J(\boldsymbol{\theta})$:

$$\arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i} l(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$$

  - $N \equiv$ Number of training samples
  - $x_i \equiv$ Input *i* features or descriptors
  - $y_i \equiv$ Ground-truth (GT) value, label for input *i*
  - $\theta \equiv$ Network architecture weights

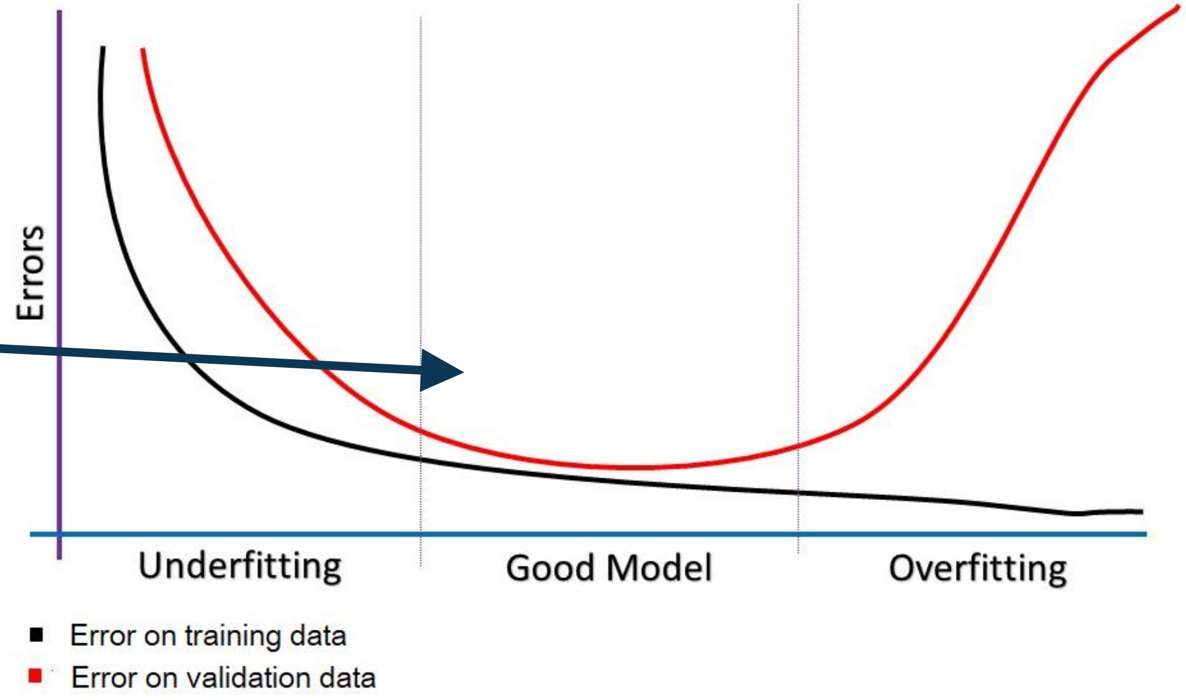
- Some well-known loss functions:
  - Classification: Cross-Entropy Loss, Hinge Loss (Support Vector Machine)
  - Regression: L1 Norm or Mean Absolute Error, L2 or Mean Squared Error

```
torch.nn.CrossEntropyLoss | torch.nn.MultiMarginLoss
torch.nn.L1Loss | torch.nn.MSELoss
```

“How to Train your (Deep) Neural Network”

Regularization Techniques

- L2 regularization: $\Omega(\boldsymbol{\theta}) = \|\mathbf{w}\|_2^2$
- L1 regularization: $\Omega(\boldsymbol{\theta}) = \|\mathbf{w}\|_1$
- Data Augmentation
- Early Stopping
- Dropout



“How to Train your (Deep) Neural Network”

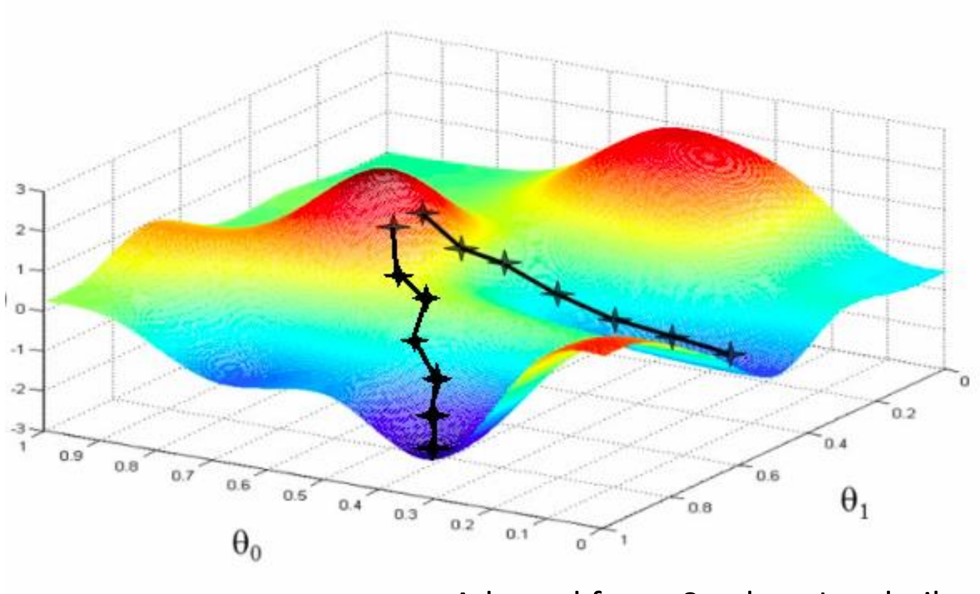
Optimization and Backpropagation

- Optimization: Find network weights q to minimize the error between true and estimated labels of training examples:

$$J(\theta) = \frac{1}{T}\sum_{t} l\left(f\left(\mathrm{x}^{(t)};\theta\right), y^{(t)}\right)$$

- Backpropagation:
  - Compute the gradient of the loss function w.r.t. parameters, from output to input layers
  - Update weights by gradient descent:

$$\theta \leftarrow \theta - \alpha\frac{\partial J}{\partial \theta} \qquad \alpha \equiv Learning\ rate$$

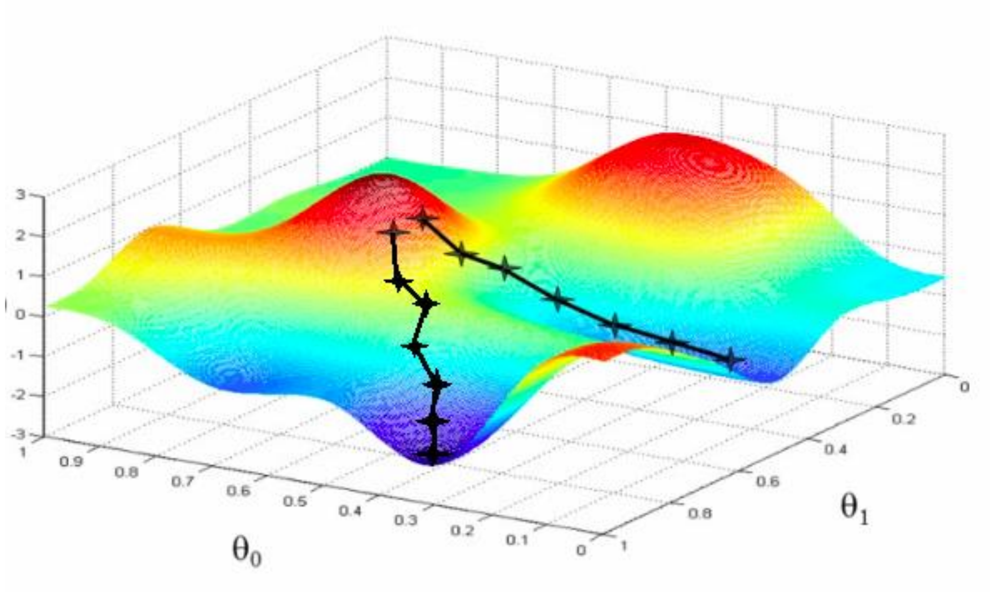


Adapted from: Svetlana Lazebnik

“How to Train your (Deep) Neural Network”

Optimization and Backpropagation

- Learning rate:
  - A learning rate α that is too large can cause the model to converge too quickly to a suboptimal solution…
  - whereas a learning rate that is too small can cause the process to get stuck
- Stochastic Gradient Descent (SGD):
  - Compute the weight update w.r.t. one training example at a time…
  - or a small batch of examples: Mini-batch SGD
  - Cycle through training examples in random order in multiple epochs

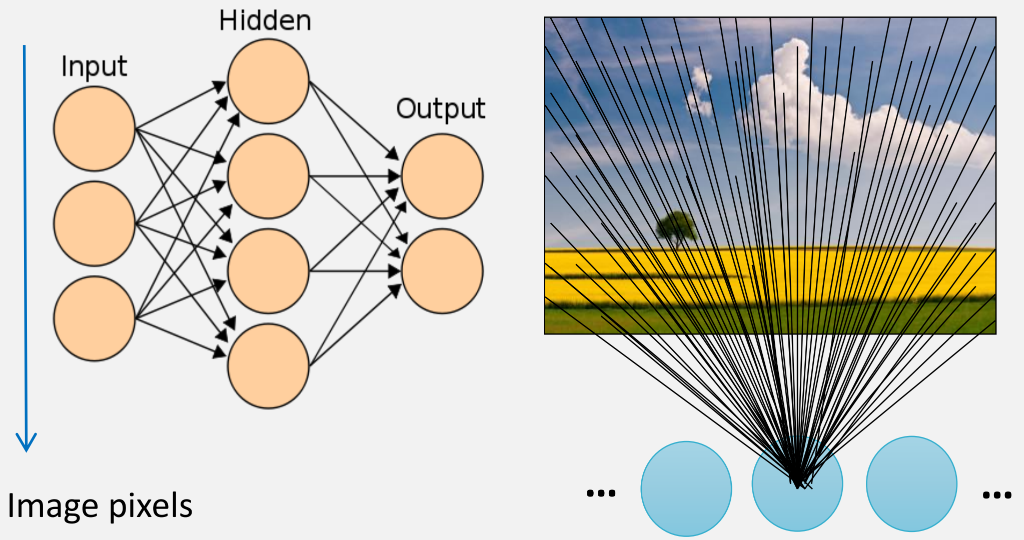


Adapted from: Svetlana Lazebnik

`torch.optim.SGD | torch.optim.RMSprop | torch.optim.Adam | ...`

3. Space: Convolutional Neural Networks (CNNs)

Space: Convolutional Neural Networks (CNNs)



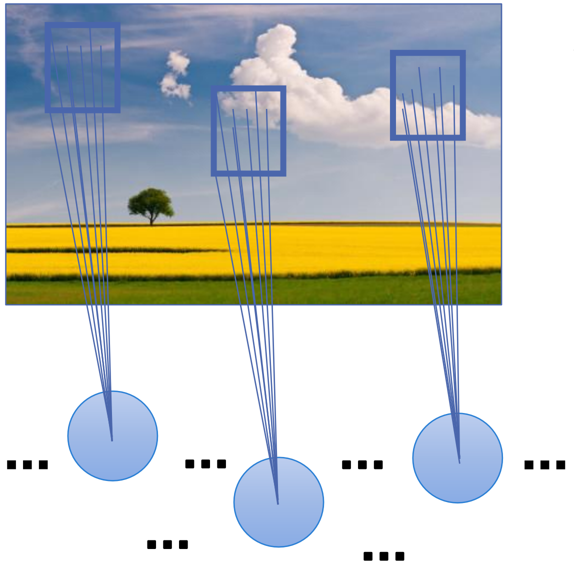
Fully Connected NNs

Example: 100x100px images
- 1 fully connected hidden layer: 10k neurons (1 per pixel)
- 1 fully connected output.

Complexity: ~100M parameters

Example: 100x100px images
- Spatial correlation/filtering is local
- 1 locally connected hidden layer with a filterbank:
  - Spatial filter size 10x10
  - 100-1000 filters
- 1 fully connected output layer

Complexity: ~10k-100k parameters, more efficient!!



- The number of parameters does not depend on the input size! (less prone to overfitting)
- Inspired by Hubel and Wiesel (1962): *Neurons are sensitive to simple patterns of light (oriented edges, color blotches)*

CNN Architecture Design

Input → A×[B×[CONV → ReLU] → POOL?] → C×[FC → ReLU] → FC → Output

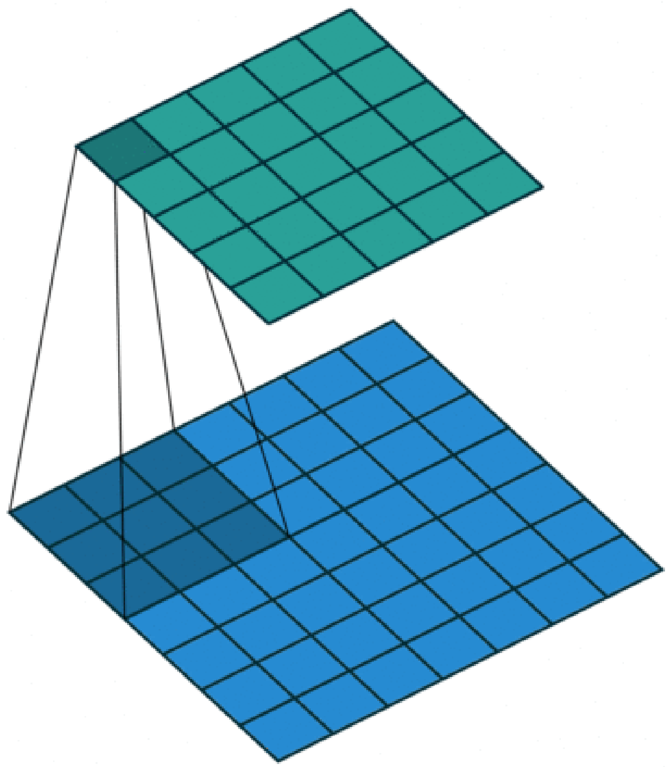
- Input
- Sequence of **A blocks** of **B** CONV layers with ReLU (or other) activations, sometimes followed by a POOL layer.
- Stack of **C** FC layers.
- Last FC layer holds the output predicted values.
- Output

`torch.nn.Conv2d | torch.nn.MaxPool2d | torch.nn.ReLU | torch.nn.Linear`

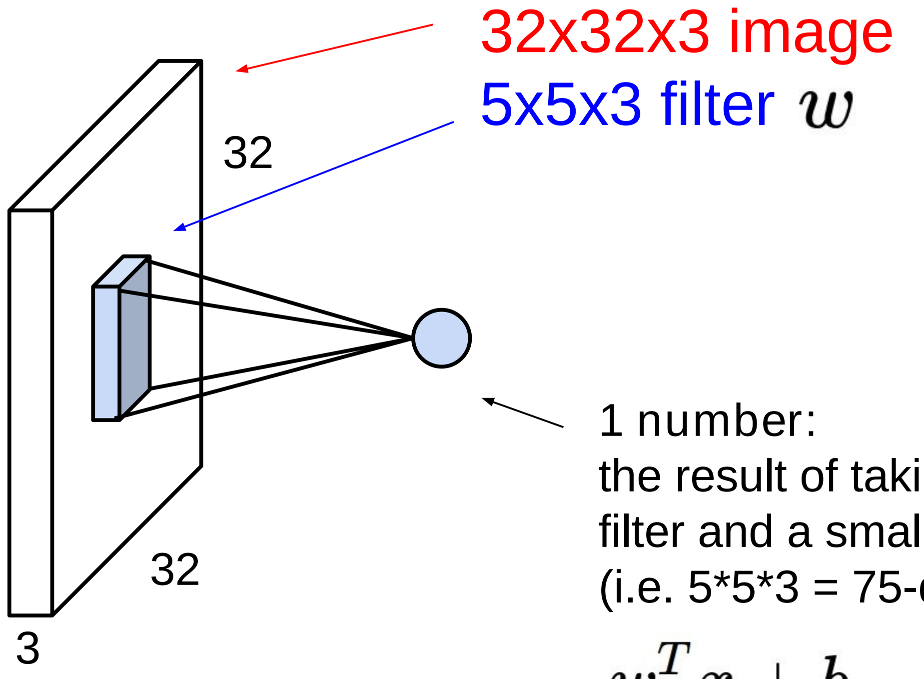
Convolutional Layer (CONV)

`torch.nn.Conv2d`

- Input: 3D volume (width, height and depth)
- Set of filters with learnable weights
  - Spatially slided along width and height of channels or spatial maps stacked on depth
- Filter or kernel size *F* → Receptive field of the neuron
- Output: 3D volume
  - Number of filters *U*
    → Depth or number of output channels
  - Stride *S* or down-sampling factor
    → Spatial size of the output
    - Bigger strides, smaller output volumes
  - Zero-padding to vary the output spatial size



Convolutional Layer (CONV)

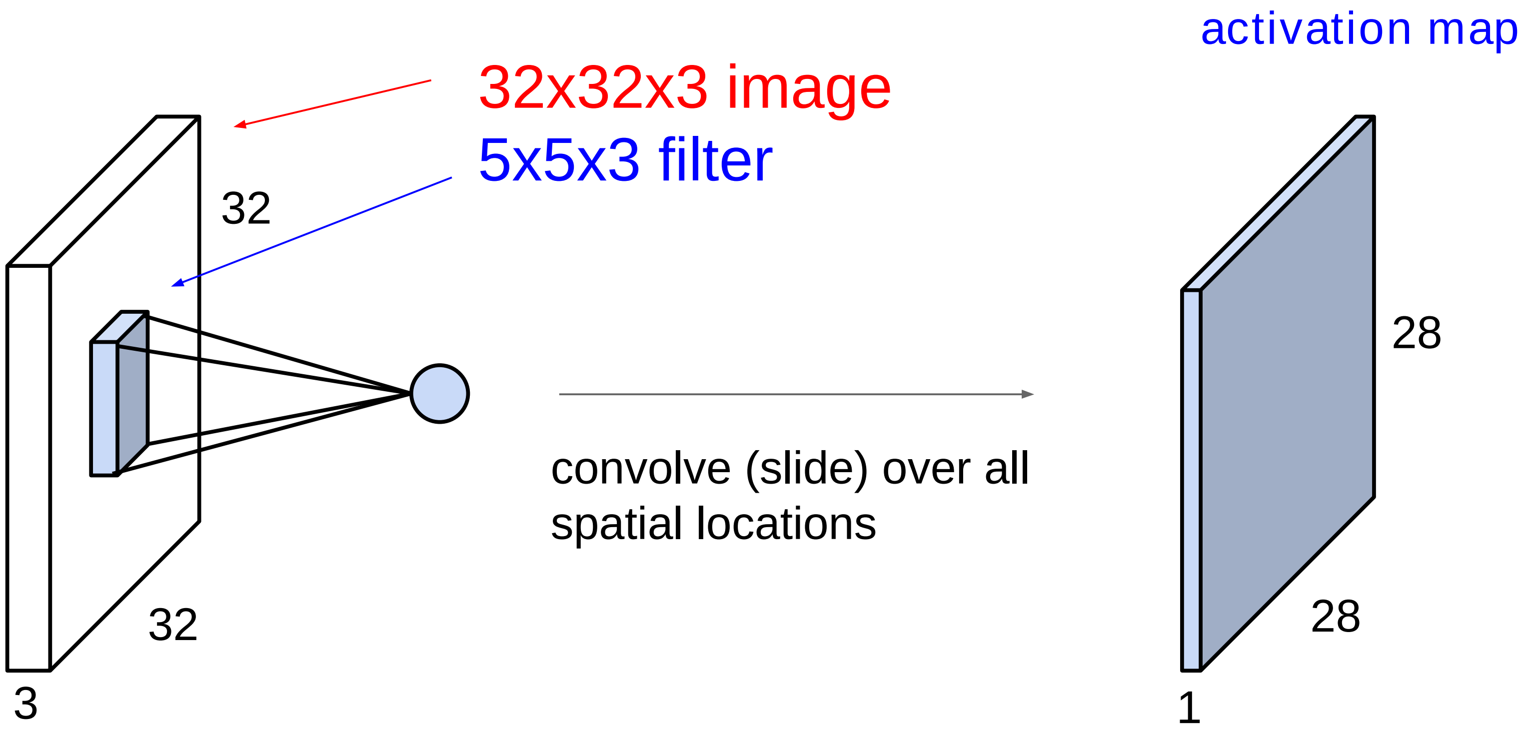


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

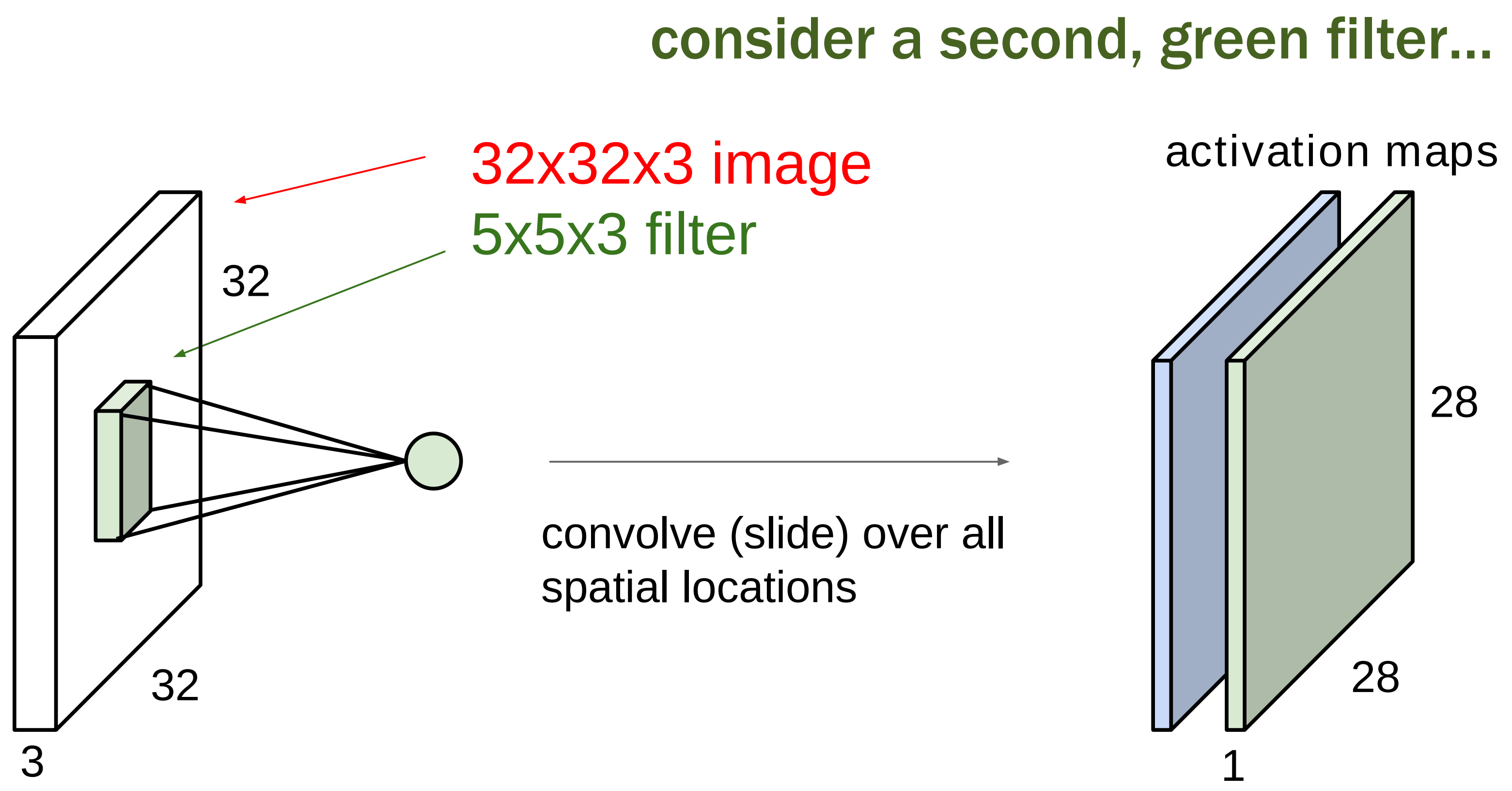
Source: CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University

Convolutional Layer (CONV)



Source: CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University

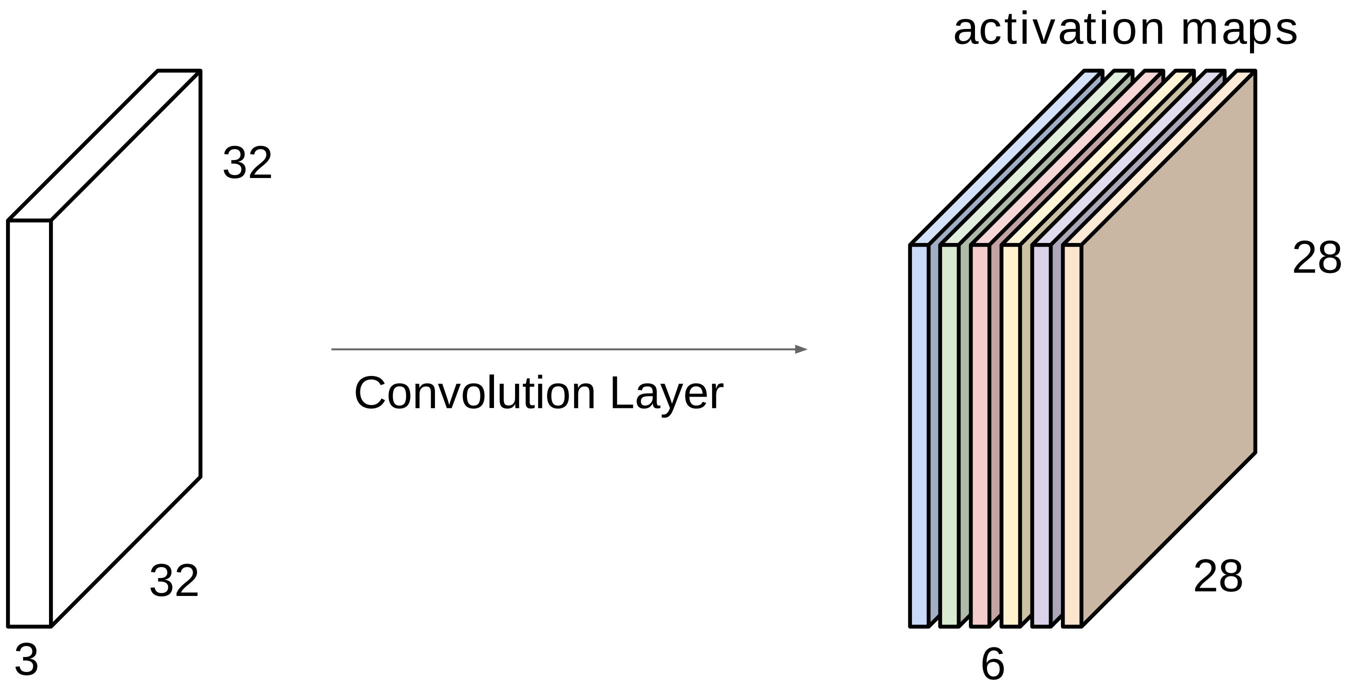
Convolutional Layer (CONV)



Source: CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University

Convolutional Layer (CONV)

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!

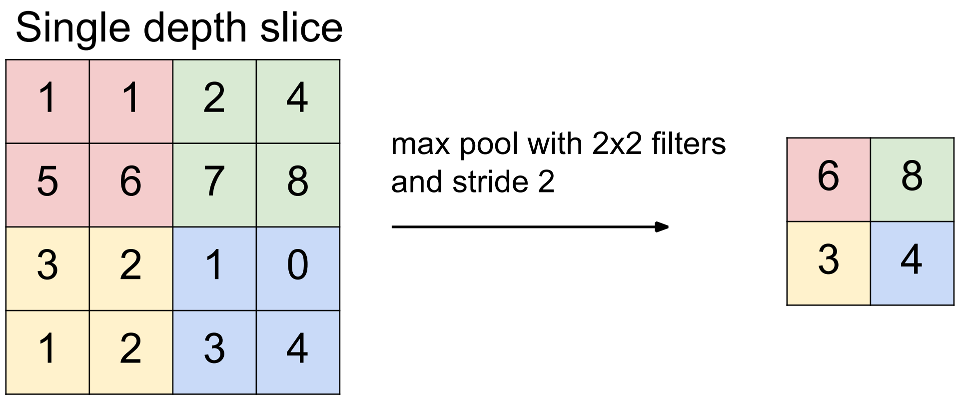
Source: CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University

Pooling Layer (POOL)

```
torch.nn.MaxPool2d
torch.nn.AvgPool2d
```

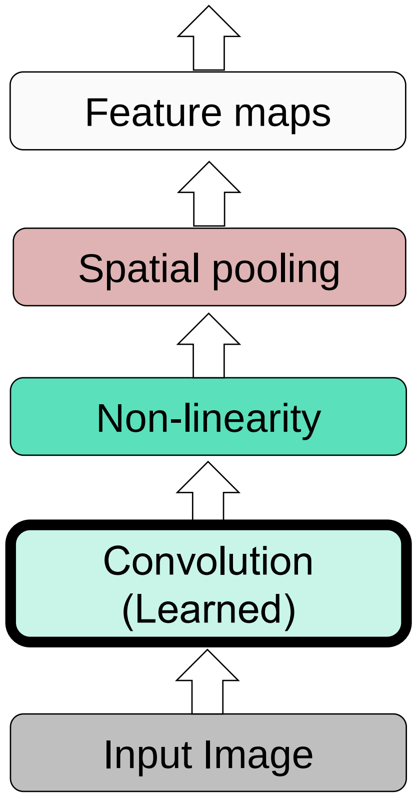
- Pooling: Fusing information from nearly locations
- Types:
  - Max pooling: Stimuli competition, only the strongest survives. In bottom/medium layers
  - Average pooling: Used more in top layers
- Invariance against small translations/shifts
- No parameters!

MAX POOLING

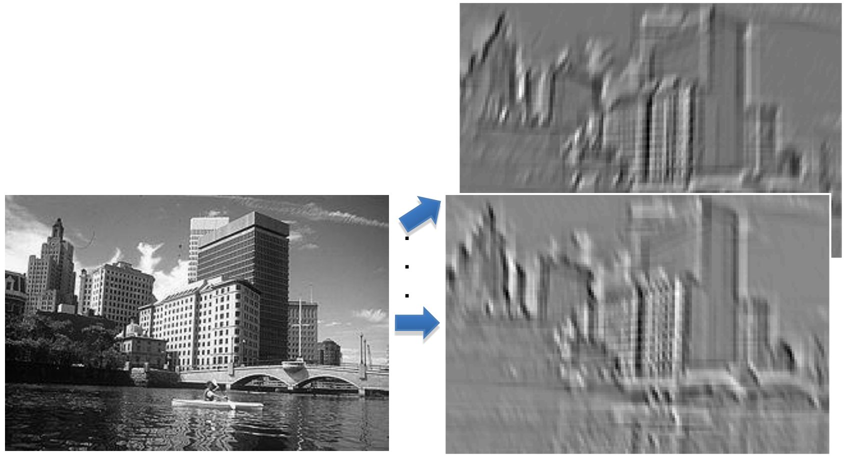


Summary: CNN Pipeline

`torch.nn.Conv2d`



Source: R. Fergus, Y. LeCun

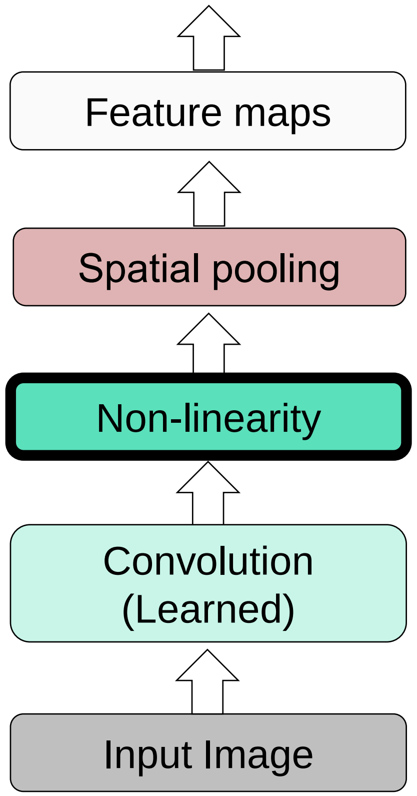


Input

Feature Map

Summary: CNN Pipeline

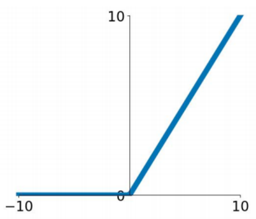
`torch.nn.ReLU`



Source: R. Fergus, Y. LeCun

ReLU

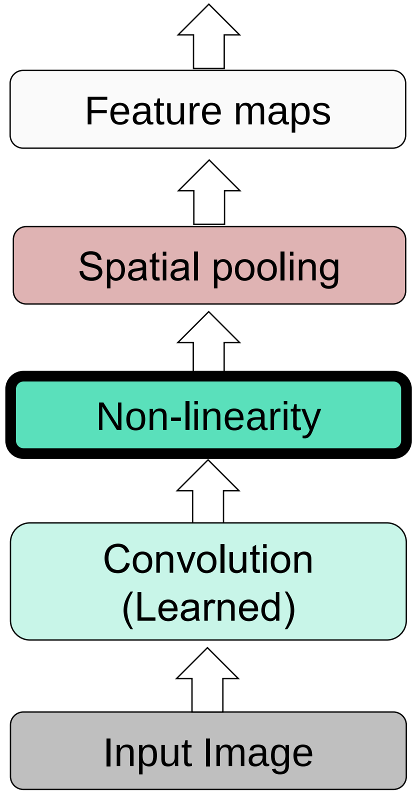
$$\max(0, x)$$



Source: [Stanford 231n]

Summary: CNN Pipeline

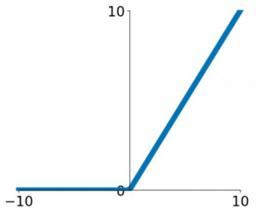
`torch.nn.LeakyReLU`
`torch.ELU`



Source: R. Fergus, Y. LeCun

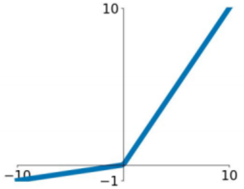
ReLU

$$\max(0, x)$$



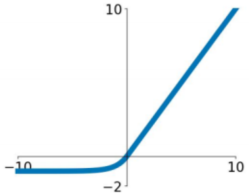
Leaky ReLU

$$\max(0.1x, x)$$



ELU

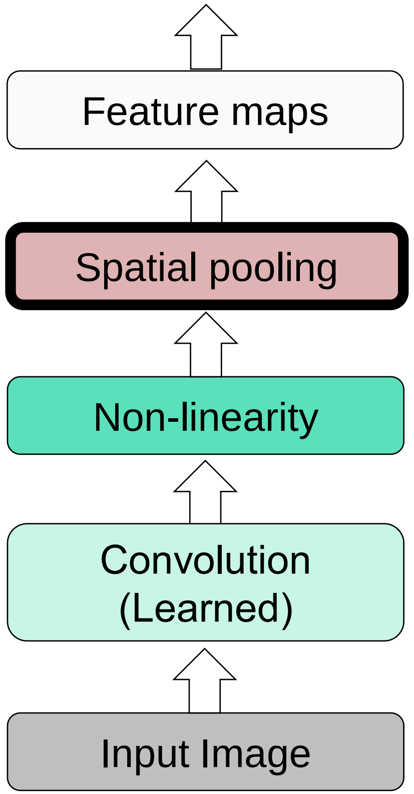
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



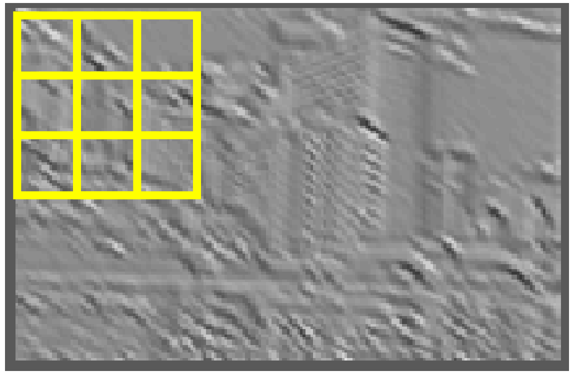
Source: [Stanford 231n]

Summary: CNN Pipeline

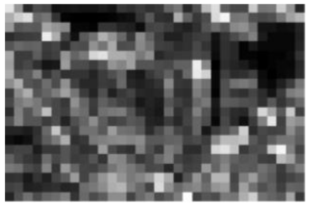
```
torch.nn.MaxPool2d
Torch.nn.AvgPool2d
```



Source: R. Fergus, Y. LeCun



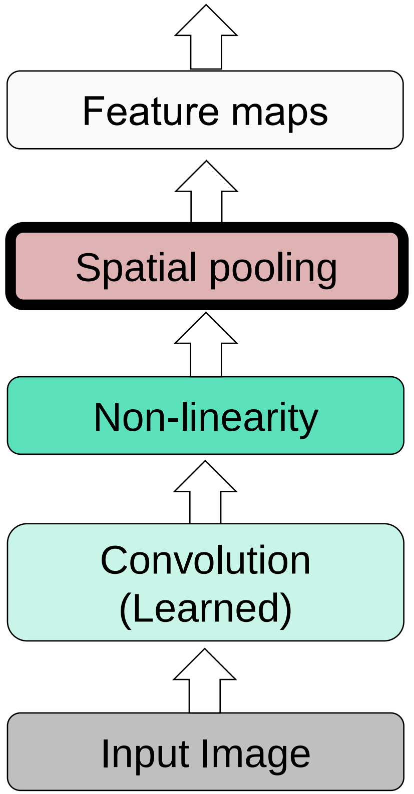
Max
(or Avg)



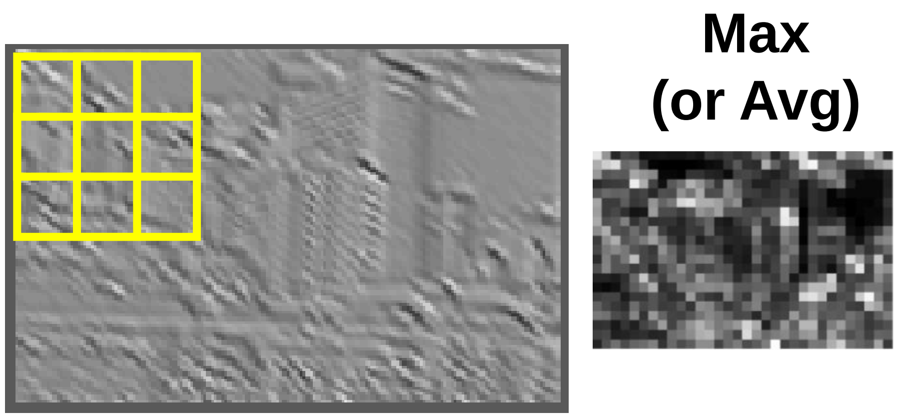
Source: [Stanford 231n]

Summary: CNN Pipeline

```
torch.nn.MaxPool2d
Torch.nn.AvgPool2d
```



Source: R. Fergus, Y. LeCun

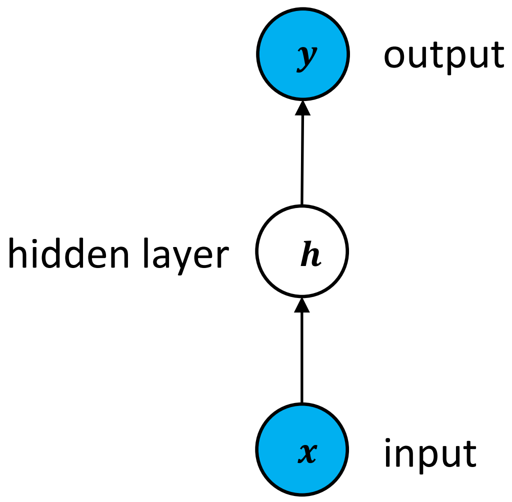


Source: [Stanford 231n]

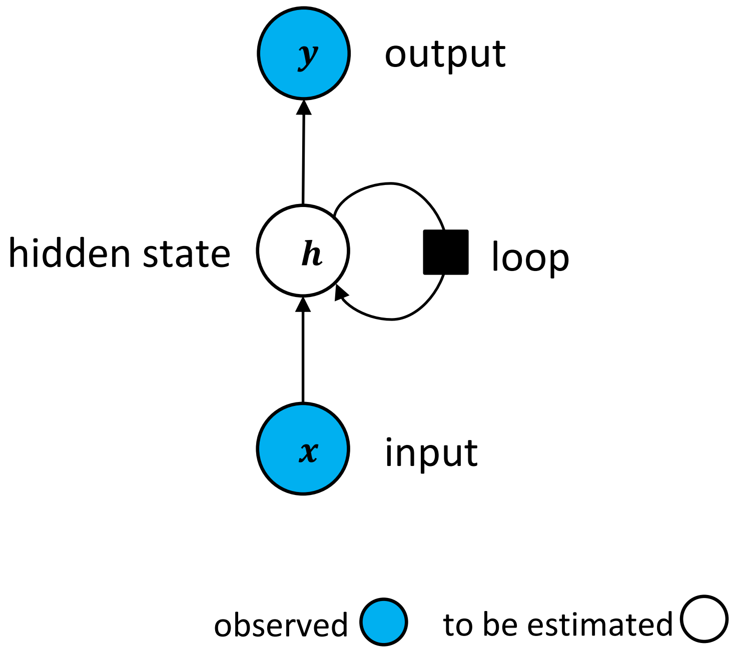
4. Time: Recurrent Neural Networks (RNNs)

Time: Recurrent Neural Networks (RNNs)

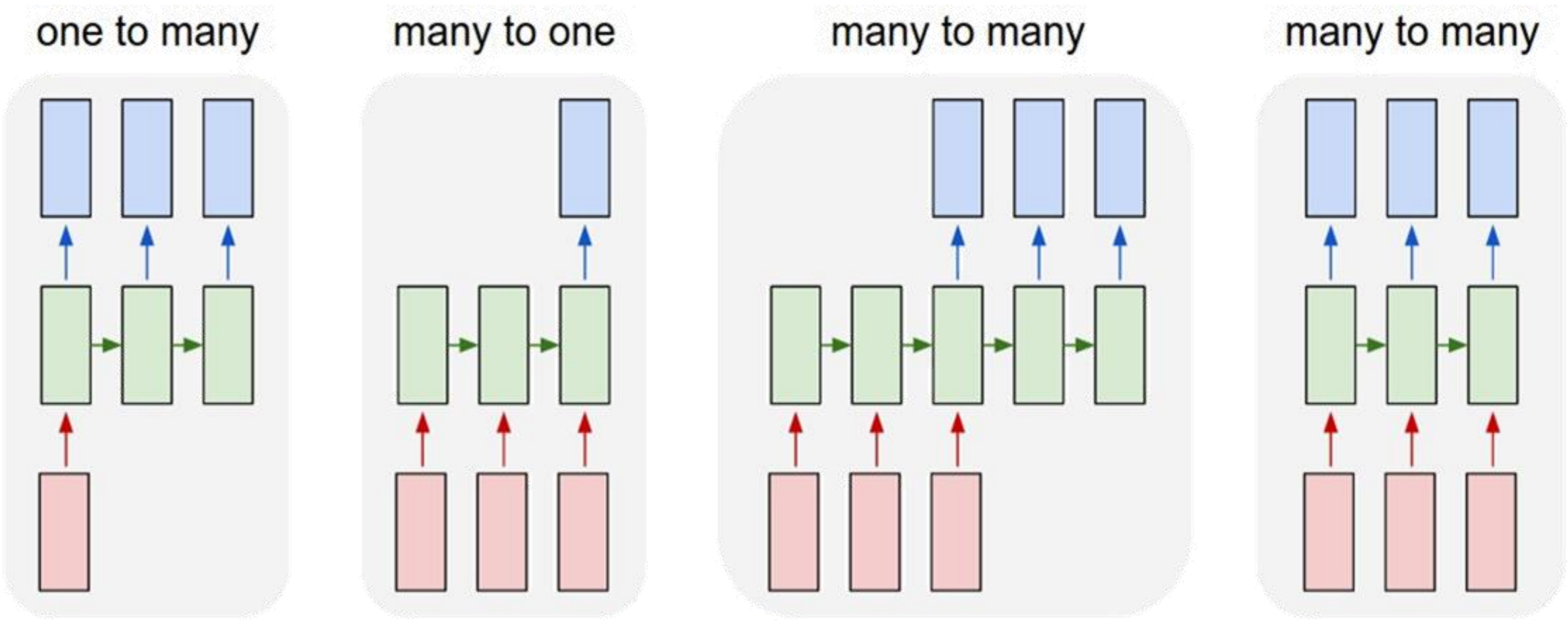
Neural Networks (NNs)



Recurrent Neural Networks (RNNs)



Types of Recurrent Neural Networks



1 input, N outputs	N inputs, 1 output	N inputs, M outputs	N inputs, N outputs

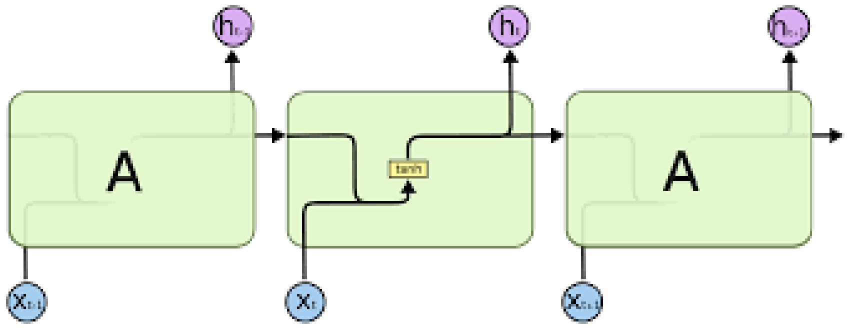
e.g. **Event description**
Image/Feature map
→ Sequence of words

e.g. **Event recognition**
Time/Space-time series
→ Class(es)

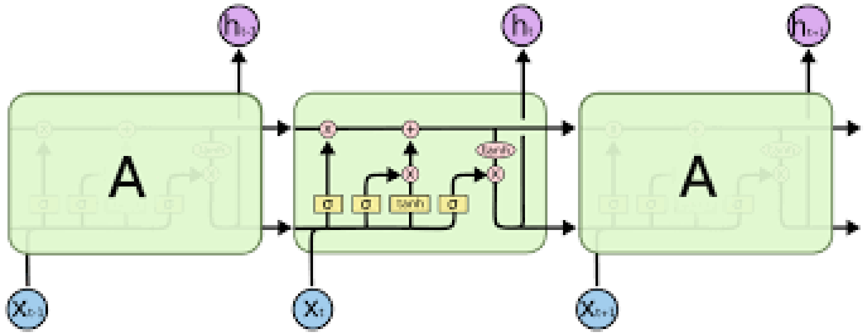
e.g. **Event detection/forecasting**
Time/Space-time series
→ Time/Space-time series for event detection/forecasting

The Problem of Long-Term Dependencies

Recurrent Neural Networks (RNNs)



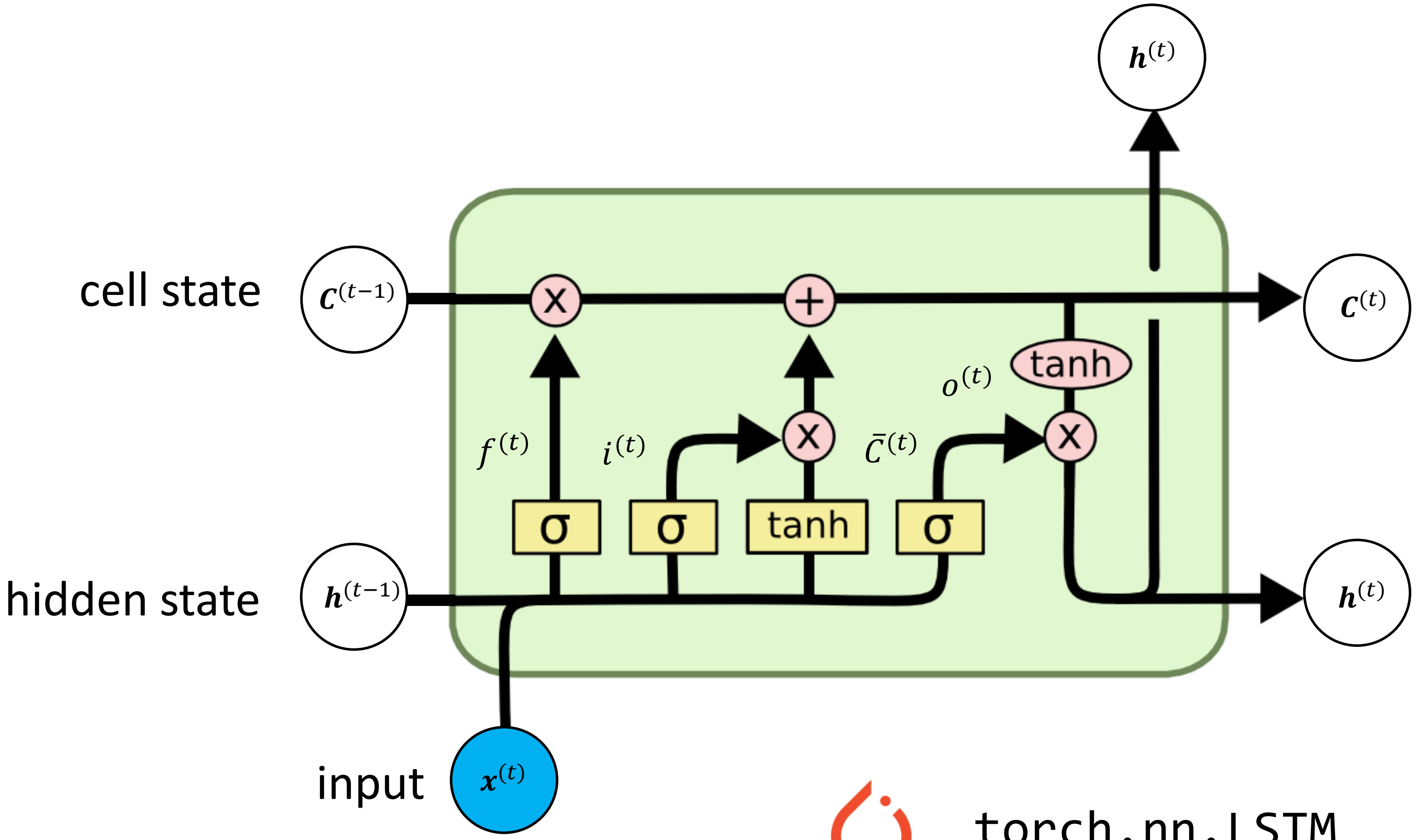
Long Short-Term Memory (LSTM) Networks



Designed to alleviate the problem of vanishing and exploding gradients

Source: Colah’s Blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

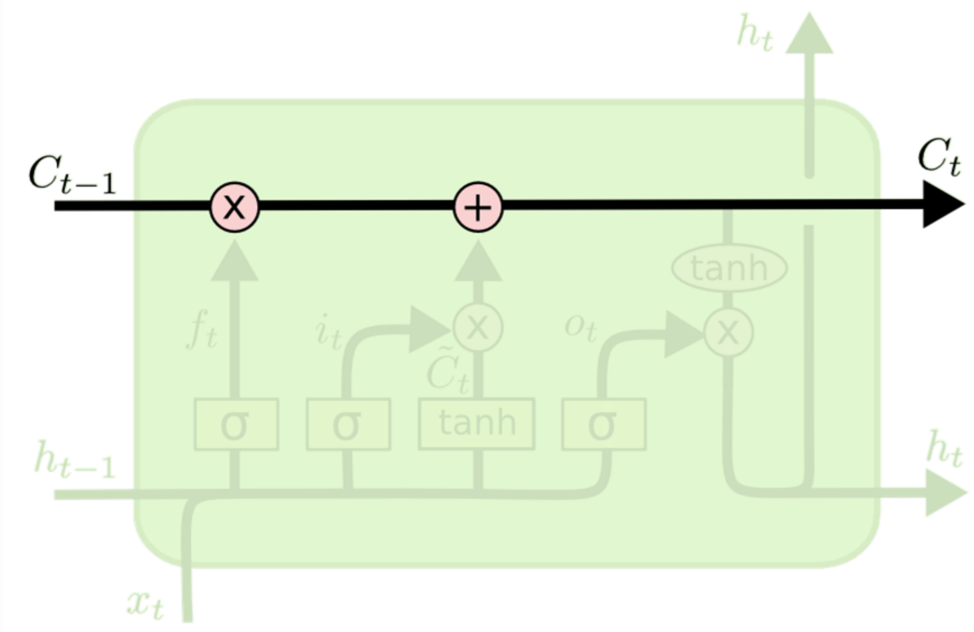
Long Short-Term Memory (LSTM) Networks [Hochreiter et al., 1997]



torch.nn.LSTM

Source: Colah's Blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

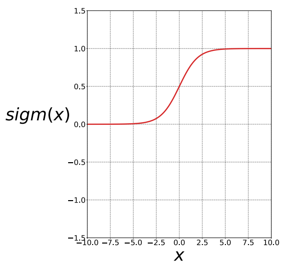
Long Short-Term Memory (LSTM) Networks [Hochreiter et al., 1997]



Source: Colah's Blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

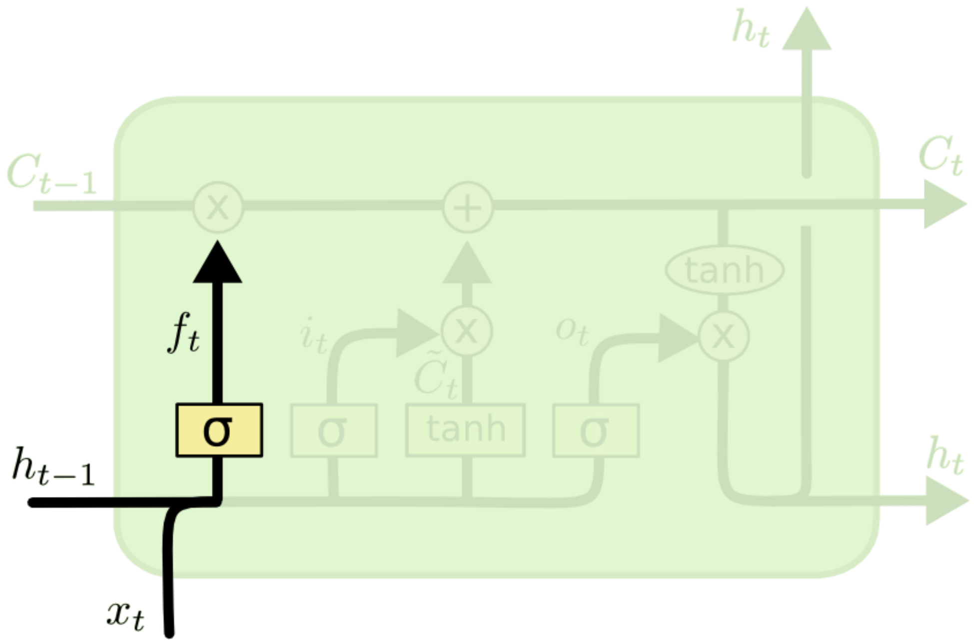
Cell state

- "Conveyor belt"
- Uninterrupted gradient flow
- LSTMs add or remove info to cell state.
- Info is regulated by gates.
- Gate: sigmoid + pointwise multiplication operation



`torch.nn.LSTM`

Long Short-Term Memory (LSTM) Networks [Hochreiter et al., 1997]



Forget gate layer

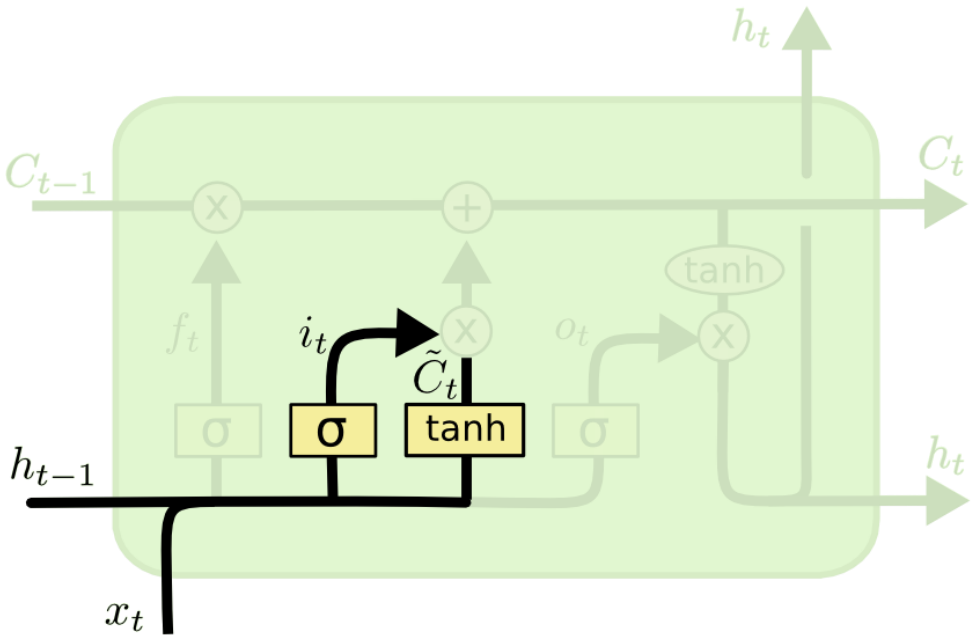
What info to remove?

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

`torch.nn.LSTM`

Source: Colah's Blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM) Networks [Hochreiter et al., 1997]



Input gate layer

What info to add?

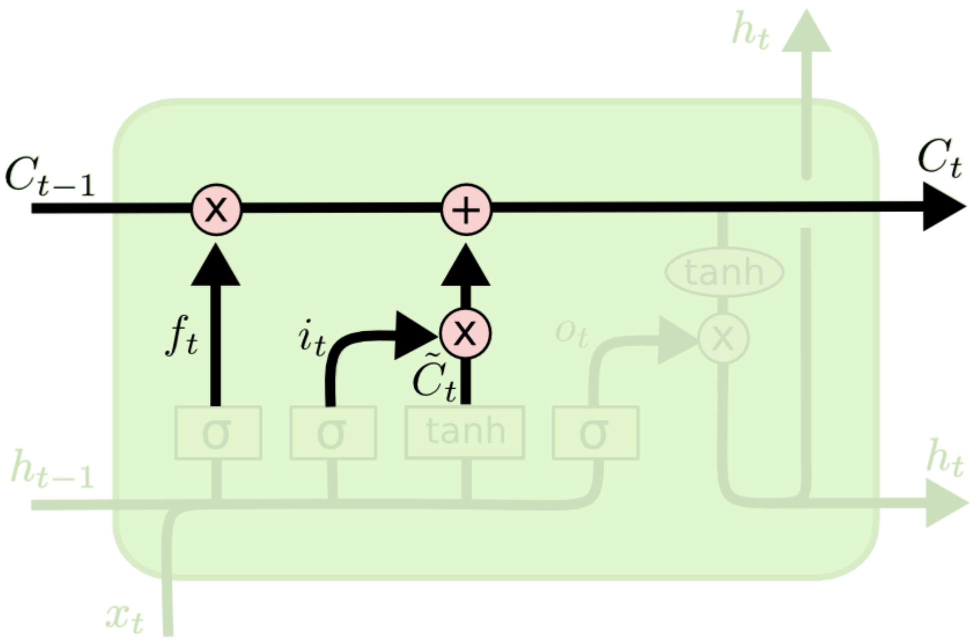
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

`torch.nn.LSTM`

Source: Colah's Blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM) Networks [Hochreiter et al., 1997]



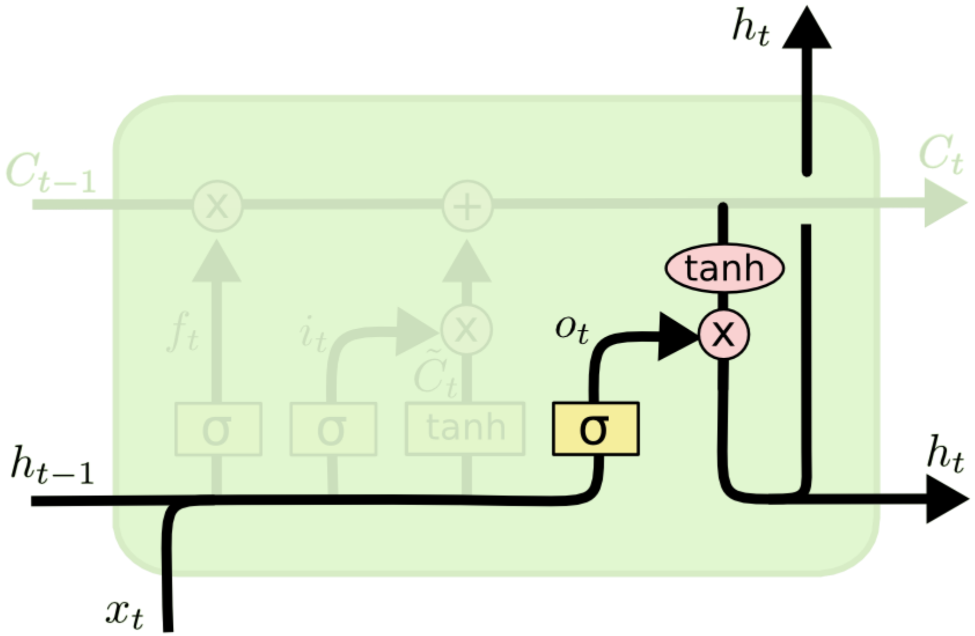
Old cell state update

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

`torch.nn.LSTM`

Source: Colah's Blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM) Networks [Hochreiter et al., 1997]



Output

$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

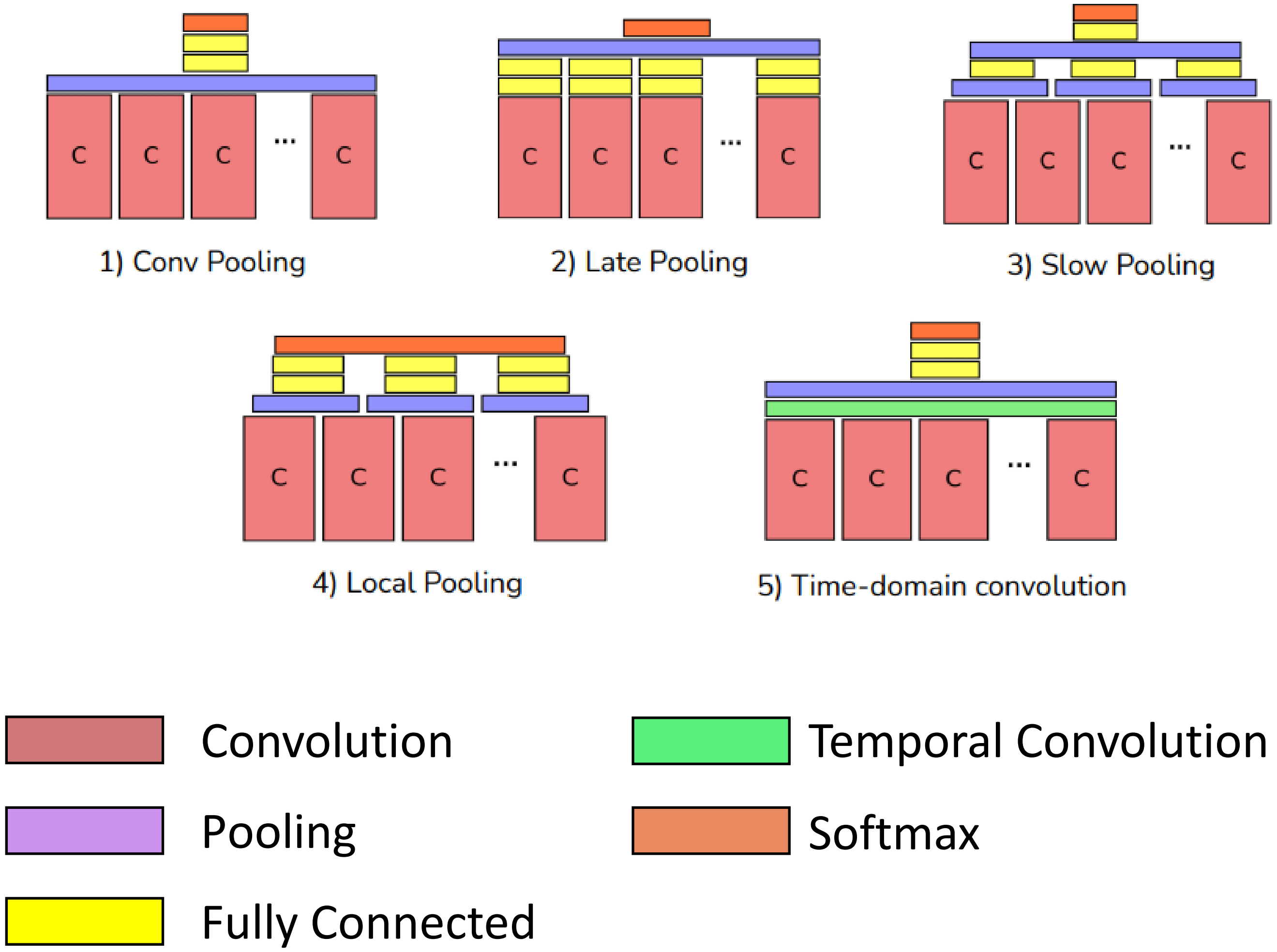
`torch.nn.LSTM`

Source: Colah's Blog, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

5. Spatio-Temporal DNNs

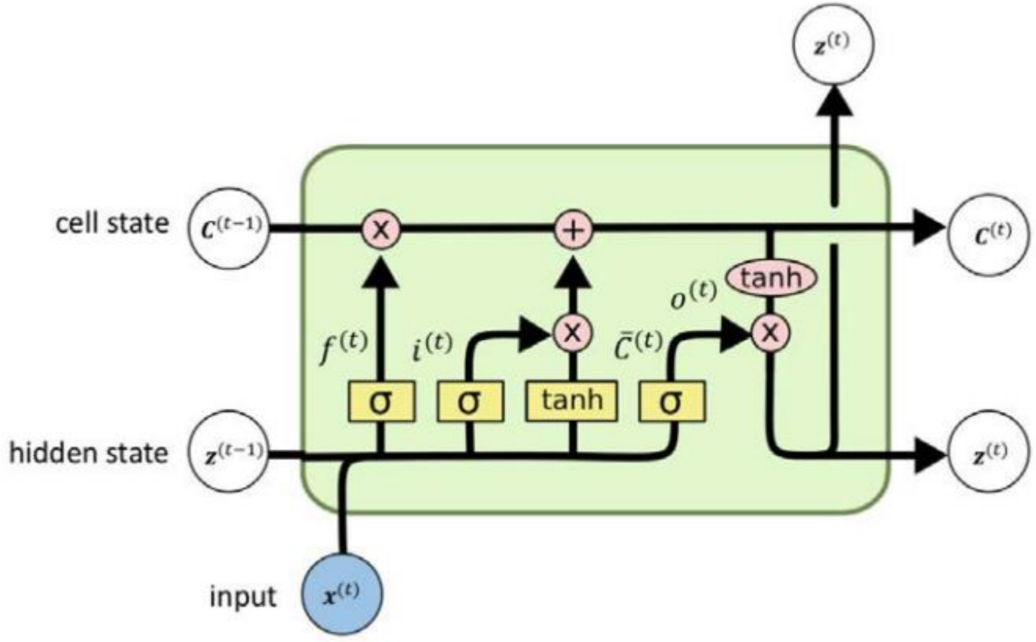
Spatio-Temporal DNNs: CNN+RNN

- Multi-frame features are temporally local (e.g. 10 frames)
- Hypothesis: A global description would be beneficial
- Challenge: Model variable length sequences with a fixed number of parameters
- Design choices:
  - Modality:
    1) RGB
    2) Motion estimation (e.g. optical flow)
    3) RGB + motion estimation
  - Features:
    1) Hand-crafted
    2) Extracted using CNN
  - Temporal aggregation:
    1) Temporal feature pooling
    2) RNN (e.g. LSTM, GRU)

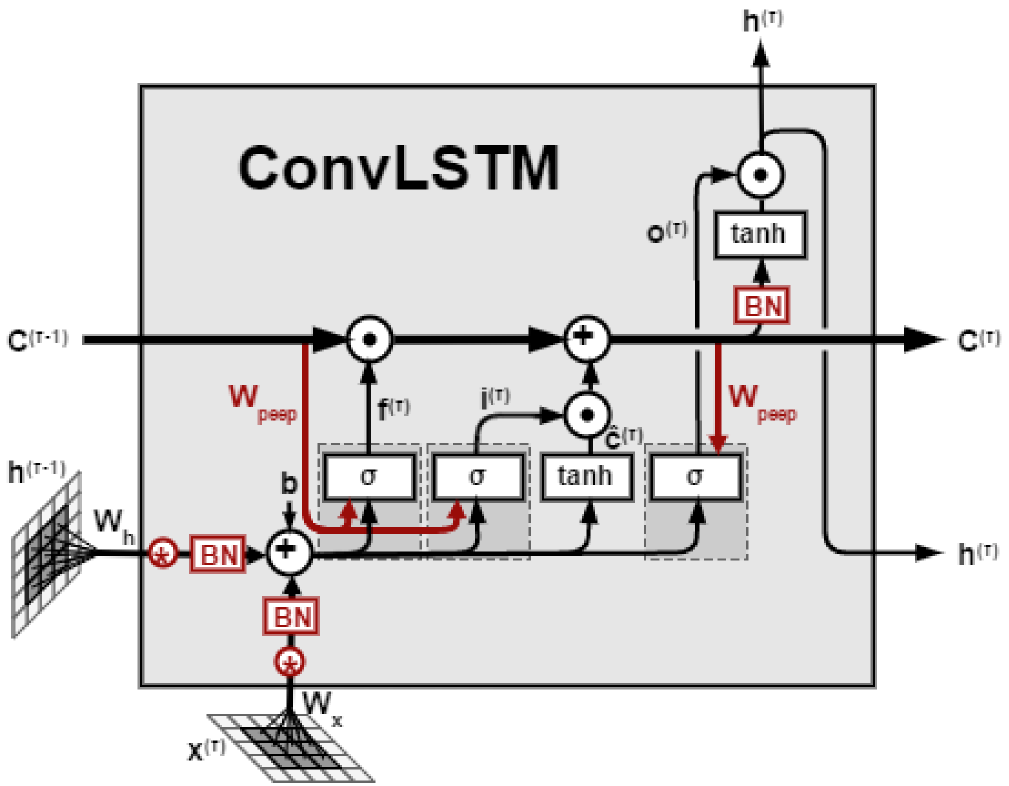


Convolutional LSTM [Shi et al., 2015]

LSTM cell



ConvLSTM cell



Source: https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

Convolutional LSTM [Shi et al., 2015]

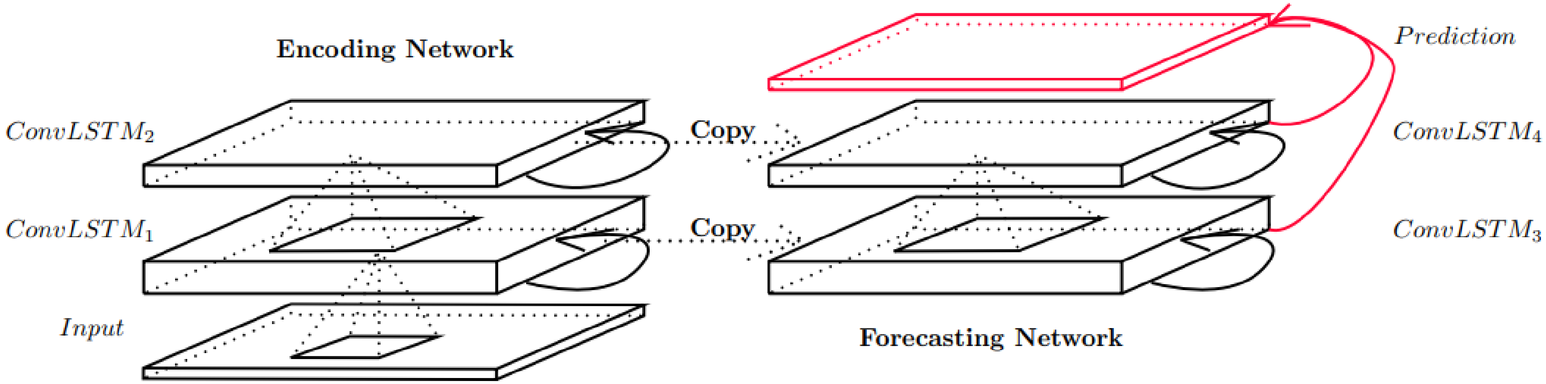


Figure 3: Encoding-forecasting ConvLSTM network for precipitation nowcasting

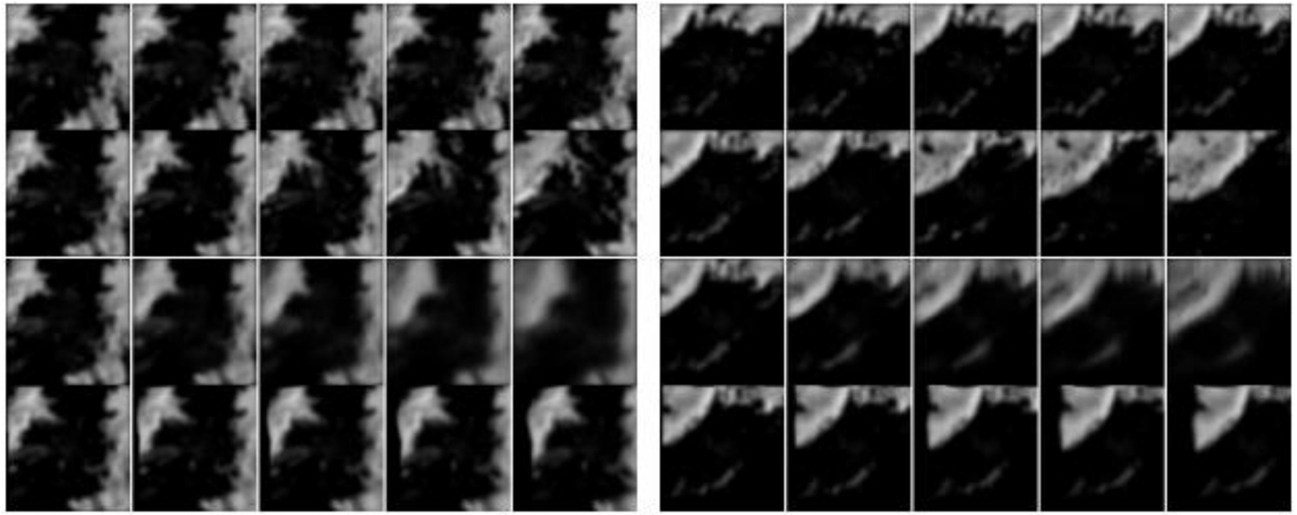
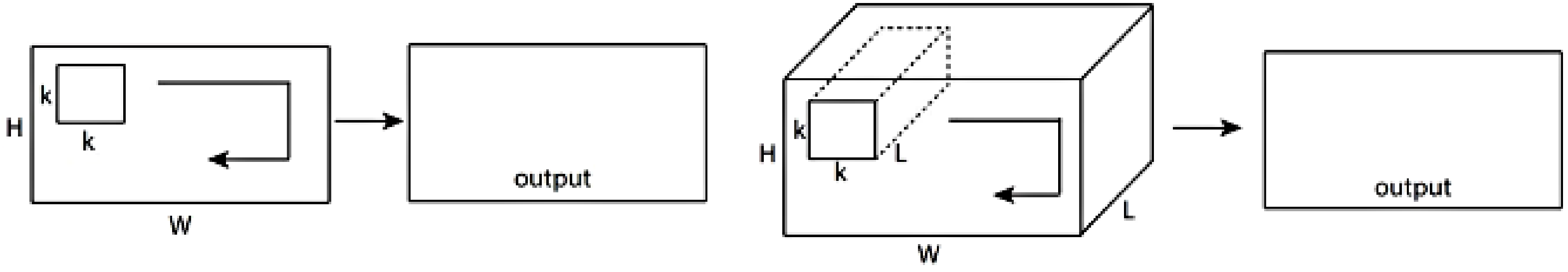


Figure 8: **(Larger Version) Two prediction examples for the precipitation nowcasting problem.** All the predictions and ground truths are sampled with an interval of 3. From top to bottom: input frames; ground truth; prediction by ConvLSTM network; prediction by ROVER2.

3D Convolution: Embed Temporal Dimension to CNN

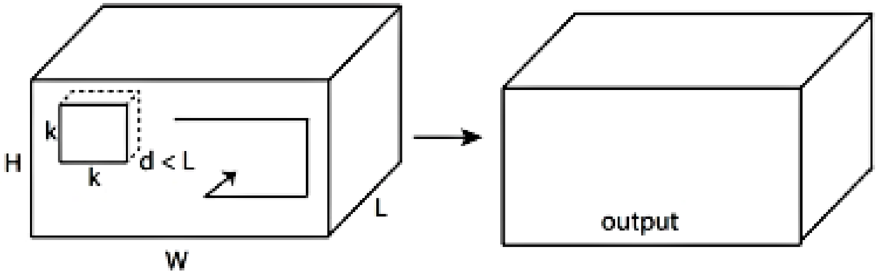
Previous work: 2D convolutions collapse temporal information



(a) 2D convolution

(b) 2D convolution on multiple frames

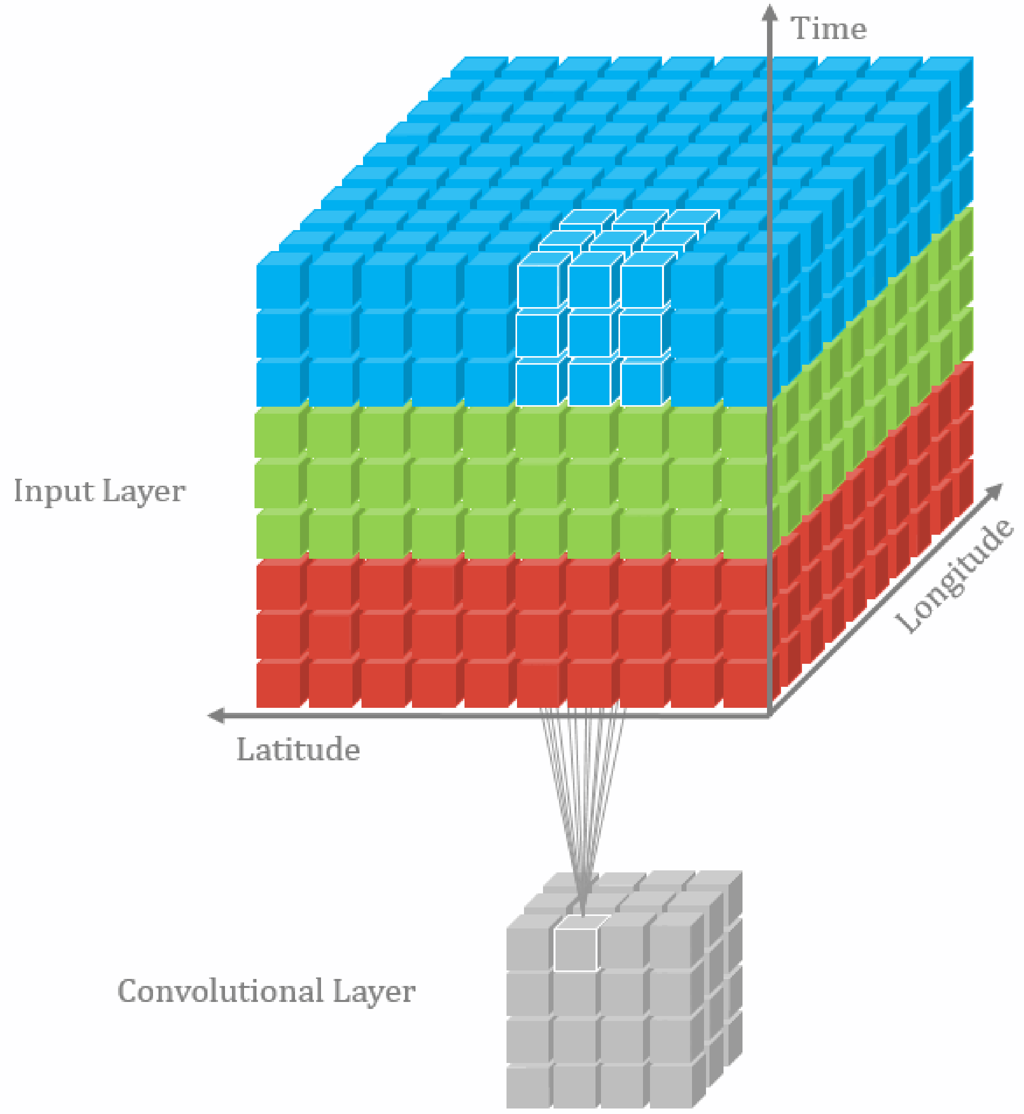
Proposal: 3D convolution → learning features that encode temporal information



(c) 3D convolution

3D Convolution

```
torch.nn.Conv3d
torch.nn.MaxPool3d
torch.nn.AvgPool3d
```



Source: Resuly's Blog, http://resuly.me/

References

- CS231 “Convolutional Neural Networks for Visual Recognition”. Stanford University. http://cs231n.stanford.edu/index.html
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.
- Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. arXiv preprint arXiv:2106.11342.
- Camps-Valls, G., Tuia, D., Zhu, X. X., & Reichstein, M. (Eds.). (2021). Deep learning for the Earth Sciences: A comprehensive approach to remote sensing, climate science and geosciences. John Wiley & Sons.
- Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285.
- “Speech, audio, image, and video processing applications”. Universidad Carlos III de Madrid. https://aplicaciones.uc3m.es/cpa/generaFicha?est=227&asig=15936&idioma=2
- Fernández Torres, M. Á. (2019). Hierarchical representations for spatio-temporal visual attention: modeling and understanding. https://doi. org/10.1109/TCSVT. 2019.2909427.
- Colah’s Blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Andrej Karpathy Blog: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

References

- Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 1725-1732).
- Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. Advances in neural information processing systems, 28.

Spatio-Temporal Deep Neural Networks for Extreme Event Detection

Miguel-Ángel Fernández-Torres

miguel.a.fernandez@uv.es
https://miguelangelft.github.io/

Image and Signal Processing - ISP

IPL IMAGE PROCESSING LABORATORY

VNIVERSITAT DE VALÈNCIA

XAIDA

XAIDA Summer School 2022
21st June 2022