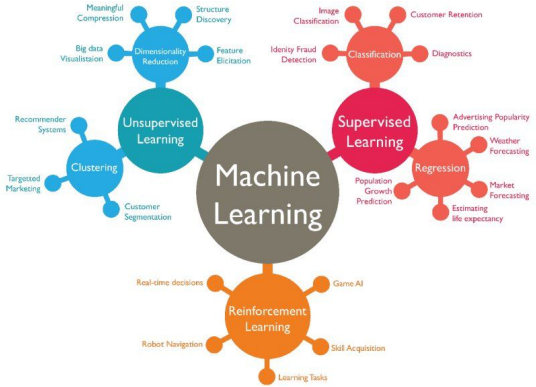
LEAP Summer Bootcamp
Week 2 - Day 2

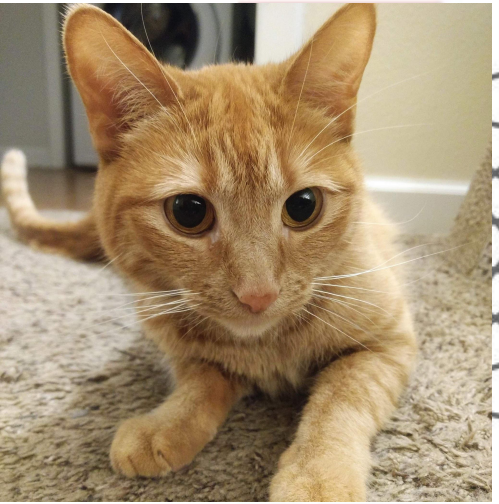
Advanced ANN: CNNs, RNNs, LSTM, GAN…
June 3rd, 2025

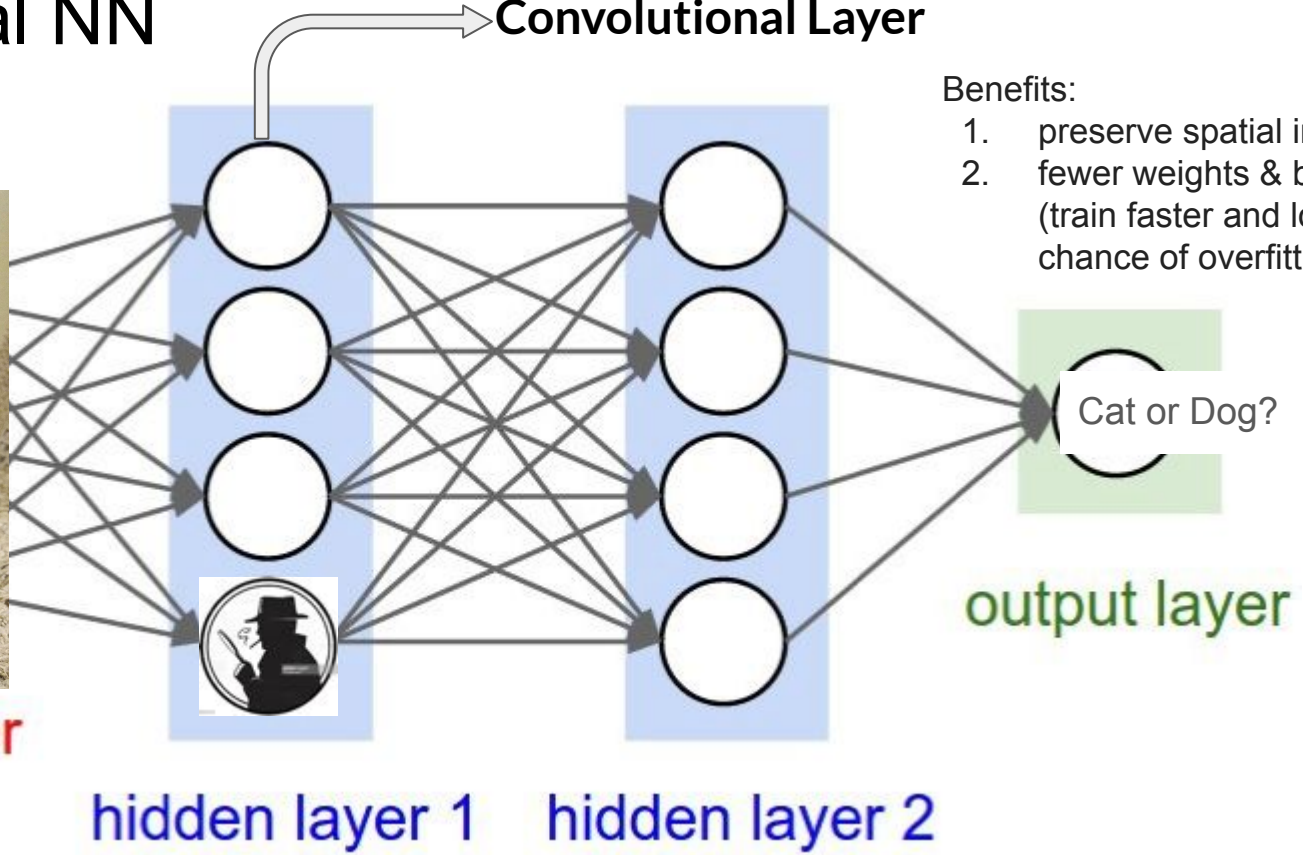
Aya Lahlou
Ph.D. Candidate
al4385@columbia.edu



NSF | LEAP

Convolutional NN

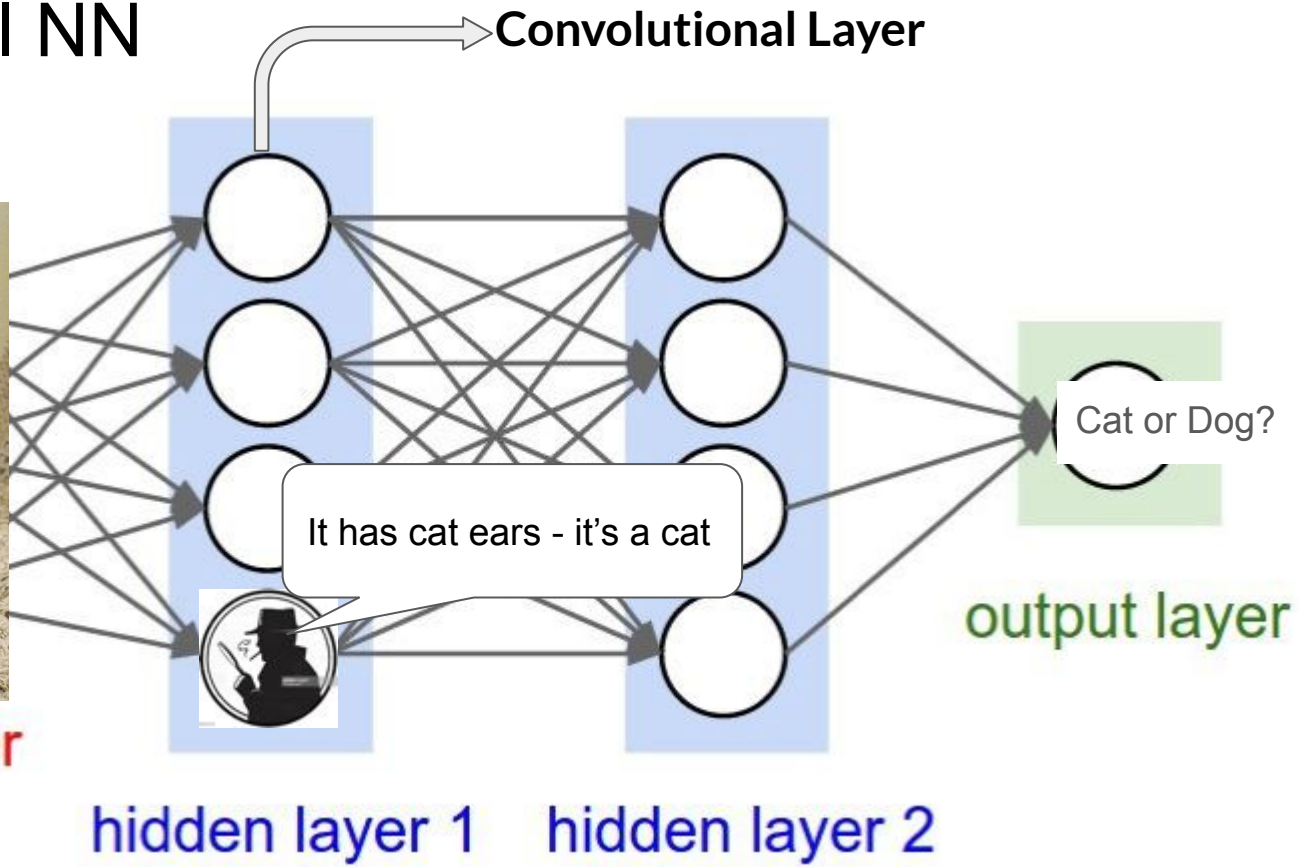




Benefits:

1. preserve spatial info
2. fewer weights & biases (train faster and lower chance of overfitting)

Convolutional NN



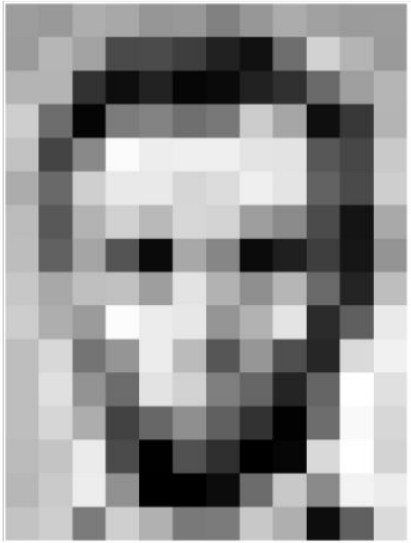
Convolutional NN

Used for **image** detection

- Neurons/Nodes are now 2D matrices i.e. FILTERS
- These FILTERS detect patterns in the input image

Initial Training:
•Edges, Corners, Shapes (circles, squares)
More Training:
•Objects (Ears, Eyes)
Even More Training:
•Complex objects (cats, dogs)

Images are Numbers



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

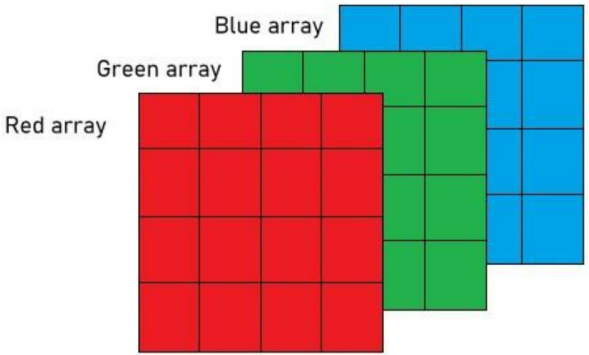
What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Images in Computers

RBG – True Color Image



Arrrays stacked over each other to form a Digital Image.

Image: GeekforGeeks

m x n x 3

- m: row
- n: column
- 3: the strength of each of the three colors

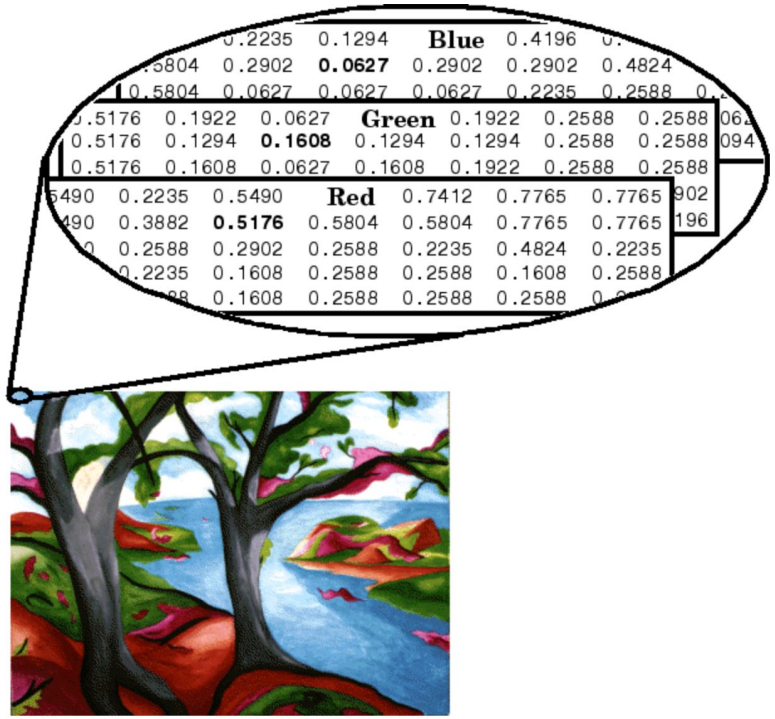
For example, the red, green, and blue color components of the pixel described as located at (10,5) are stored in RGB(10,5,1), RGB(10,5,2), and RGB(10,5,3), respectively.

For the 3rd dimension:

- (0, 0, 0) is black
- (1, 1, 1) is white

- Matlab

Images in Computers

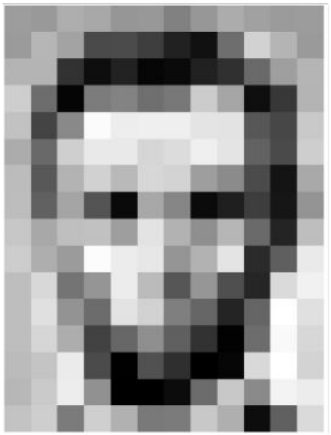


An example from Matlab:

To determine the color of the pixel at (2,3), look at the RGB triplet stored in (2,3,1:3). Suppose (2,3,1) contains the value 0.5176, (2,3,2) contains 0.1608, and (2,3,3) contains 0.0627. The color for the pixel at (2,3) is

0.5176 0.1608 0.0627

Tasks in Computer Vision



Input Image

Pixel Representation

classification

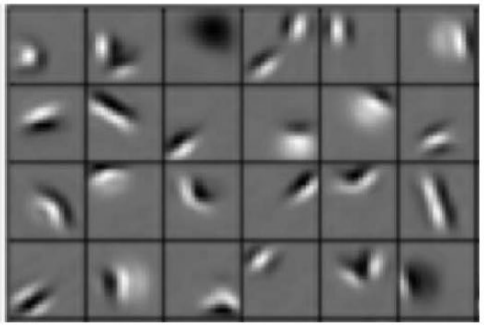
Lincoln	0.8
Washington	0.1
Jefferson	0.05
Obama	0.05

- Regression: output variable takes continuous value
- Classification: output variable takes class label. Can produce probability of belonging to a particular class

Convolutional Neural networks: Images

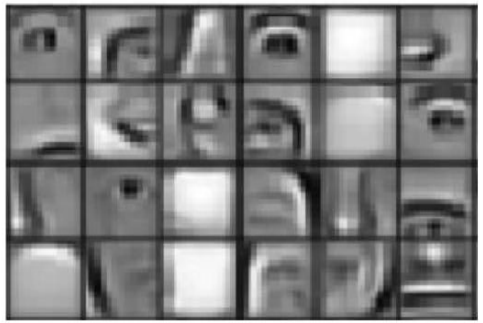
What are Convolutional Neural Networks?

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

High level features

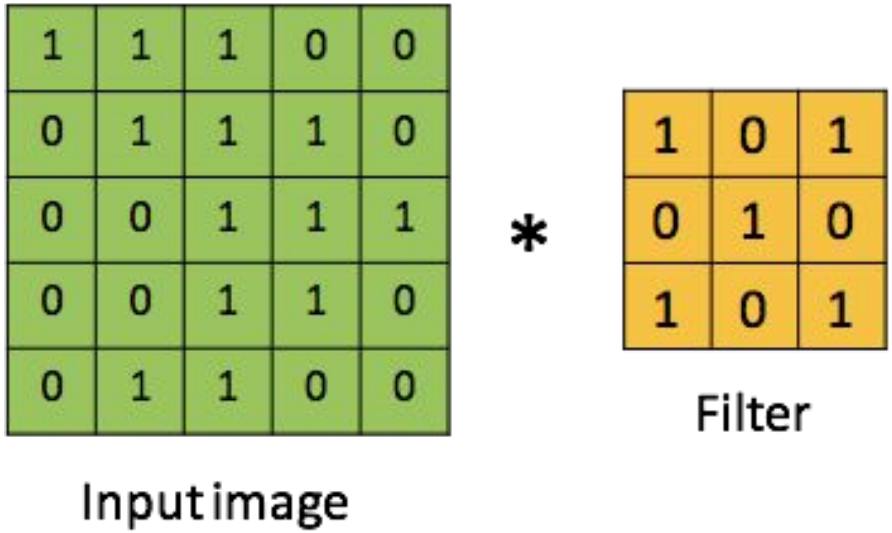
Facial structure

Filters

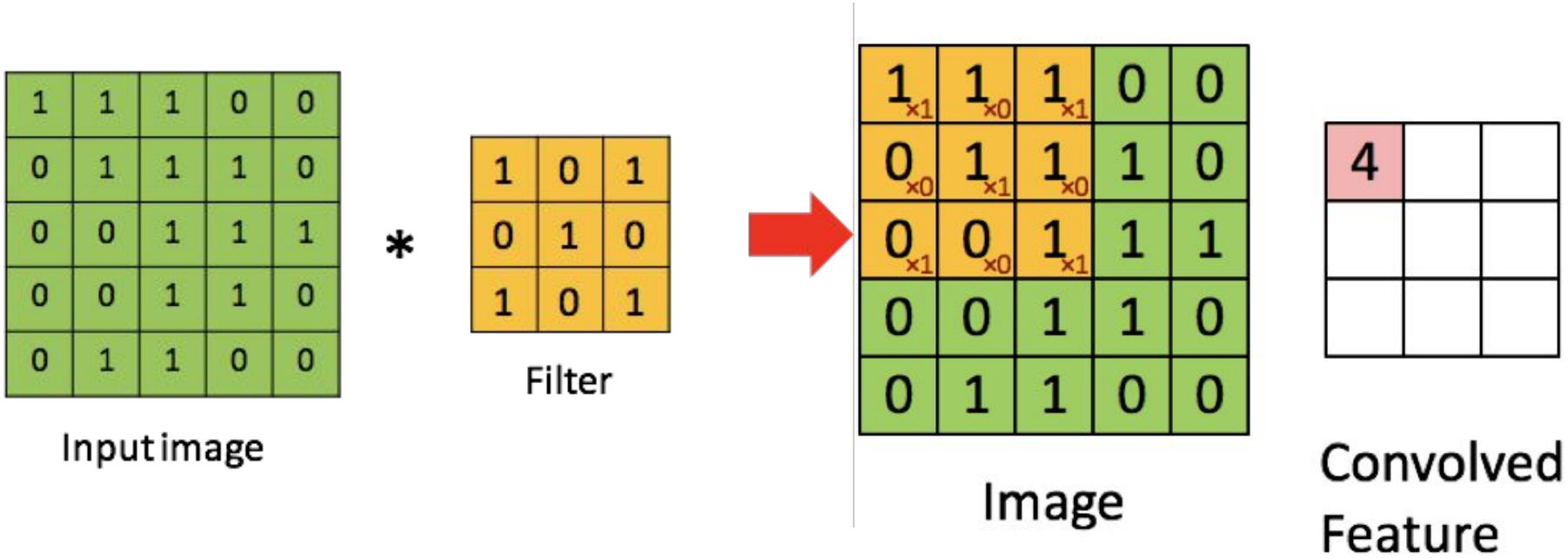
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input image

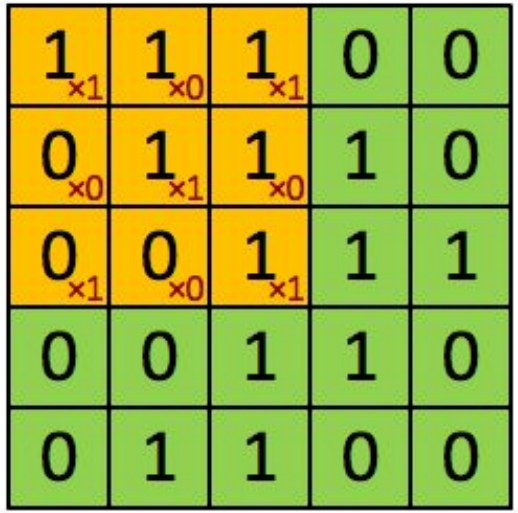
Filters

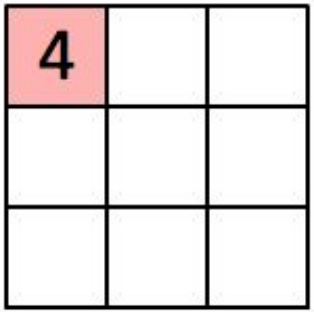

1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
*
1 0 1
0 1 0
1 0 1
Input image
Filter

Convolving


1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
Input image
*
1 0 1
0 1 0
1 0 1
Filter
1x1 1x0 1x1 0 0
0x0 1x1 1x0 1 0
0x1 0x0 1x1 1 1
0 0 1 1 0
0 1 1 0 0
Image
4
Convolved Feature

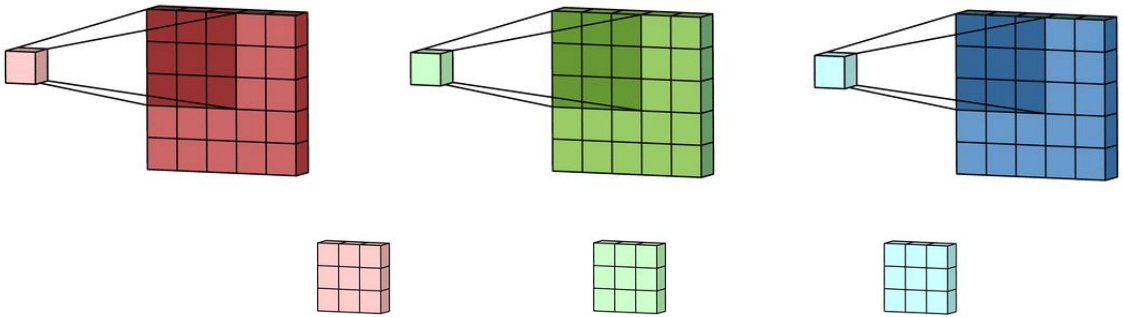
Convolving





CNN layer for a single multi-channel kernel

Perform convolution across channels

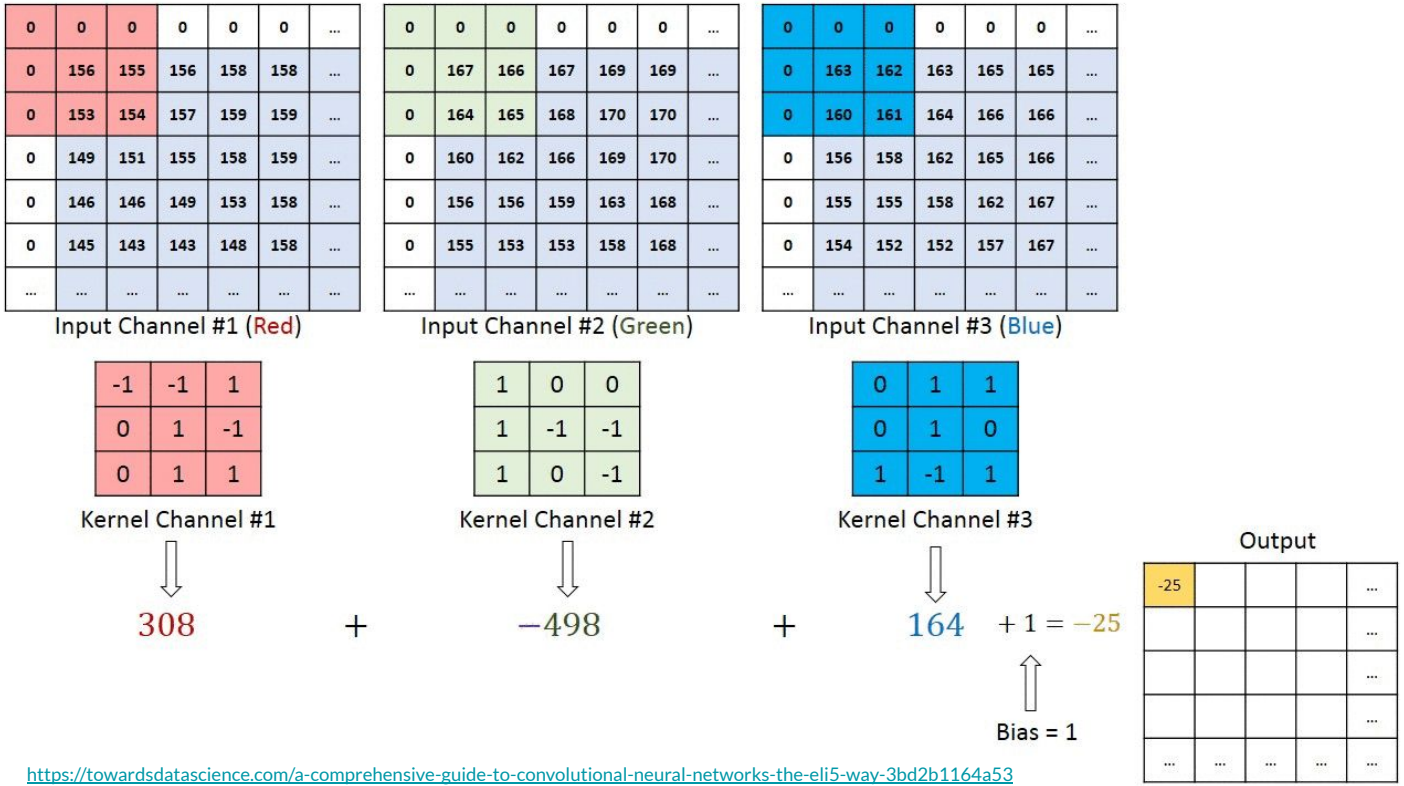


Sum across channels

Add bias term

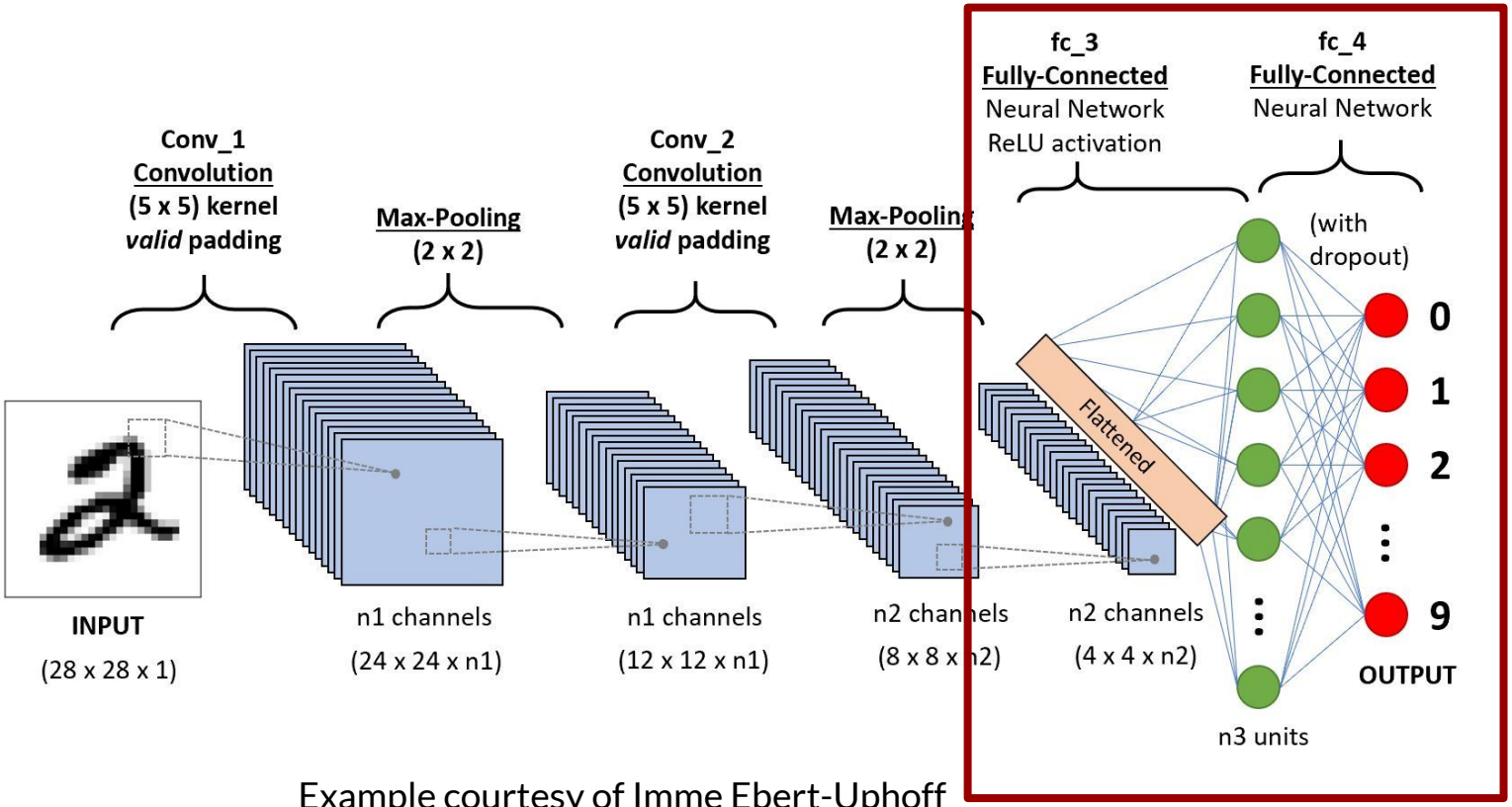
https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Zero padding in action



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

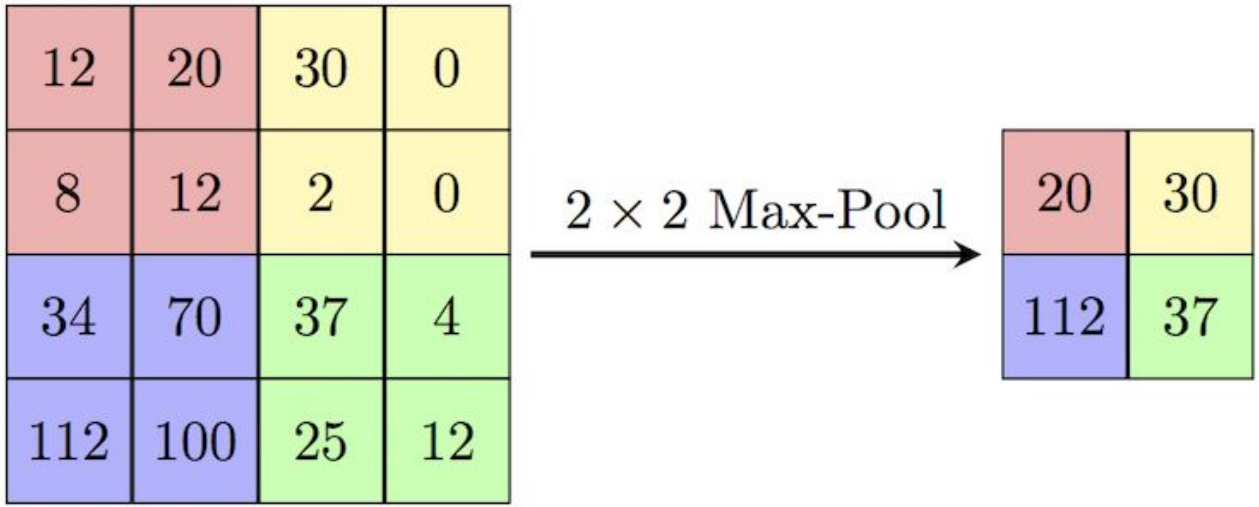
Connecting CNNs to Fully Connected Networks



Example courtesy of Imme Ebert-Uphoff

Pooling Layer

- **Reduce spatial size**
- Noise suppressant
- Max Pooling/Average/Sum Pooling



Seasonal ENSO prediction

Task: Predict Nino3.4 index __ months into the future using maps of the ocean state

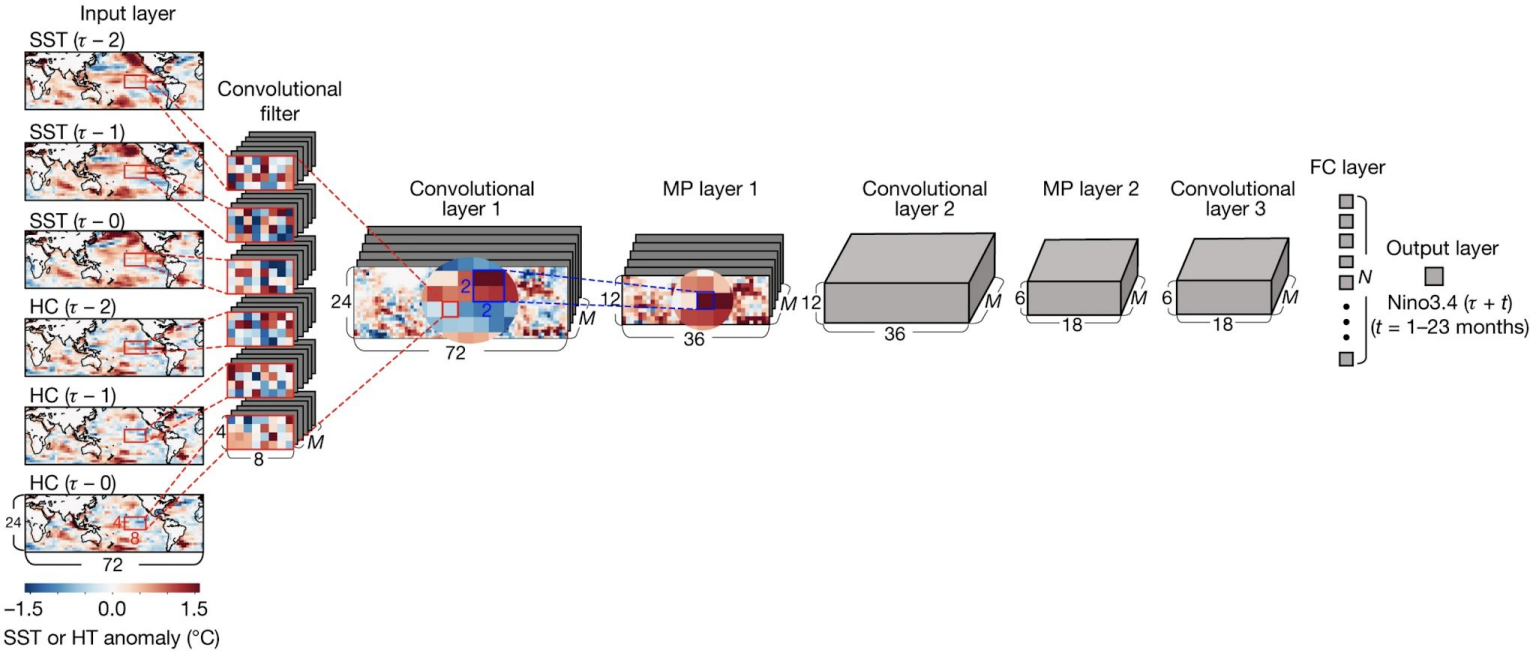


Figure from Ham et al. (2019)

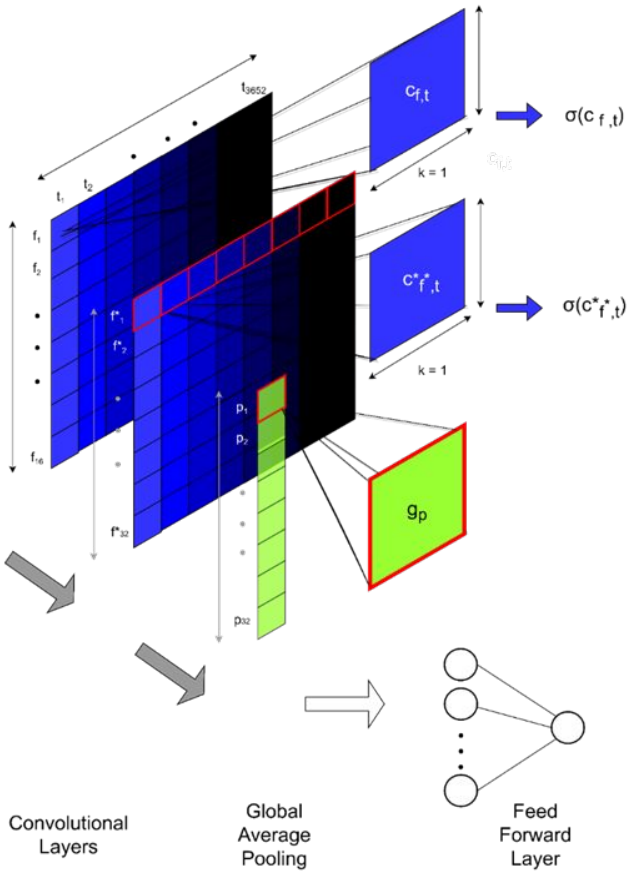
Advantages of CNNs

- Easily retains the spatial relationships between adjacent pixels
  - CNNs are very common for machine learning tasks that use images as inputs
- Fewer weights and biases which speeds up training and reduces overfitting
  - The network only has to learn a small set of kernels and then apply them to *every part of the image* (rather than different weights/kernels for each part)

Idea

Use spatial information in images to reduce the dimensionality of the problem

An example of a convolutional neural network for time-series



The CNN is also comprised of an input, hidden and output layer. The input layer and the output product are the same as the FFN; hence, the main difference from the FFN is the composition of the hidden layers. Particularly, the CNN has two convolutional layers, a global average pooling layer and a FF layer. For the first convolutional layer, there are 16 kernels (or filters), f, of size one (Each kernel is applied independently to each predictor at each time step). Each filter produces an output, c, by generating a unique bias and weight at each time step, t, and for each predictor. Therefore, in the CNN, the convolutional layers involve the weighted sum of predictors over time for each filter; this differs from the FNN, as in the FNN, the predictions are based on the weighted sum of predictors without considering the time dimension. **Fig. 7** illustrates the hidden layers of the CNN. The overview of computations are depicted here:

$$\square_{\square,\square} = \square_{\square,\square,\square,\square}\square_{\square,\square} + \square_{\square,\square,\square,\square}\square_{\square,\square} + \square_{\square,\square,\square,\square}\square_{\square,\square} + \square_{\square,\square,\square,\square}\square_{\square,\square} + \square_{\square,\square,\square,\square}\square_{\square,\square} + \square_{\square,\square} \ldots (15)$$

The output of the first convolutional layer enters the second convolutional layer, c^*, via the ReLU activation function and computes the following:

$$\square_{\square,\square,\square} = \square\square\square_{\square,\square,\square}\square \ldots (16)$$

The second convolutional layer has 32 kernels, f^*. For each filter output, c^*, at a time step, the calculation is shown:

$$\square^{\square}_{\square^*,\square,\square} = \sum_{\square=1}^{16} \square_{\square^*,\square,\square,\square,\square}\square_{\square,\square,\square} + \square_{\square^*,\square,\square} \ldots (17)$$

The output of each kernel in the second convolutional also enters the ReLU function to gain enhanced pattern recognition:

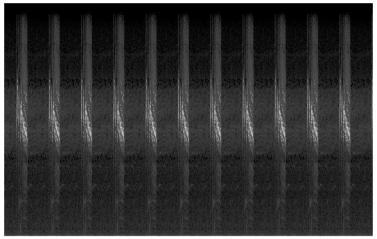
$$\square^{\square}_{\square^*,\square,\square} = \square\square\square^{\square}_{\square^*,\square,\square}\square \ldots (18)$$

After exiting the second convolutional process, there is input into the global average pooling layer. For each of the 32 channels, p, an average, g, is taken across all time steps (3,652 days):

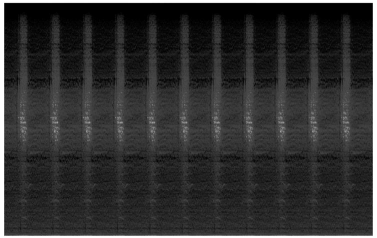
$$\square_{\square,\square} = \frac{1}{3652}\sum_{\square=1}^{3652} \square^{\square}_{\square^*,\square,\square} \quad \square\square\square\square\square^{\square} = \square \ldots (19)$$

Now, the pooling output units are then processed by a feed forward layer of 32 neurons, producing predicted SF counts. The model was run with a learning rate of 0.001, batch size of 32 and 100 epochs.

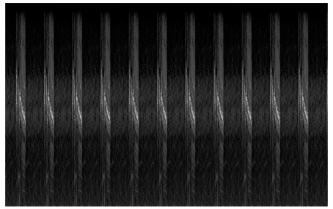
Convolutional Neural Network Audio



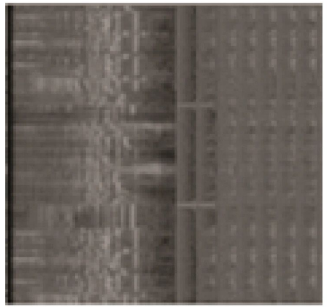
(a) Spectrogram.



(b) Harmonic.



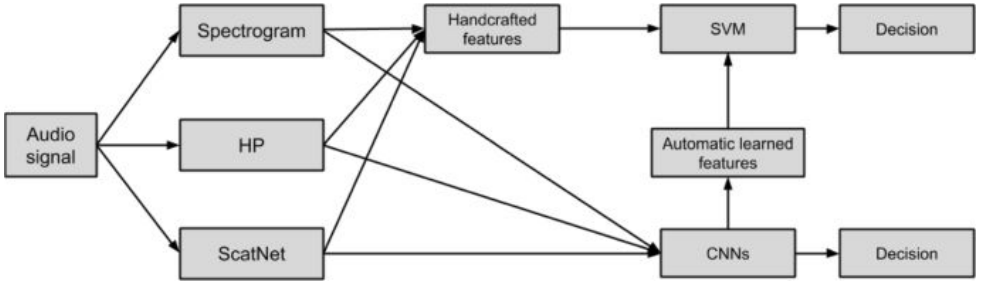
(c) Percussion.



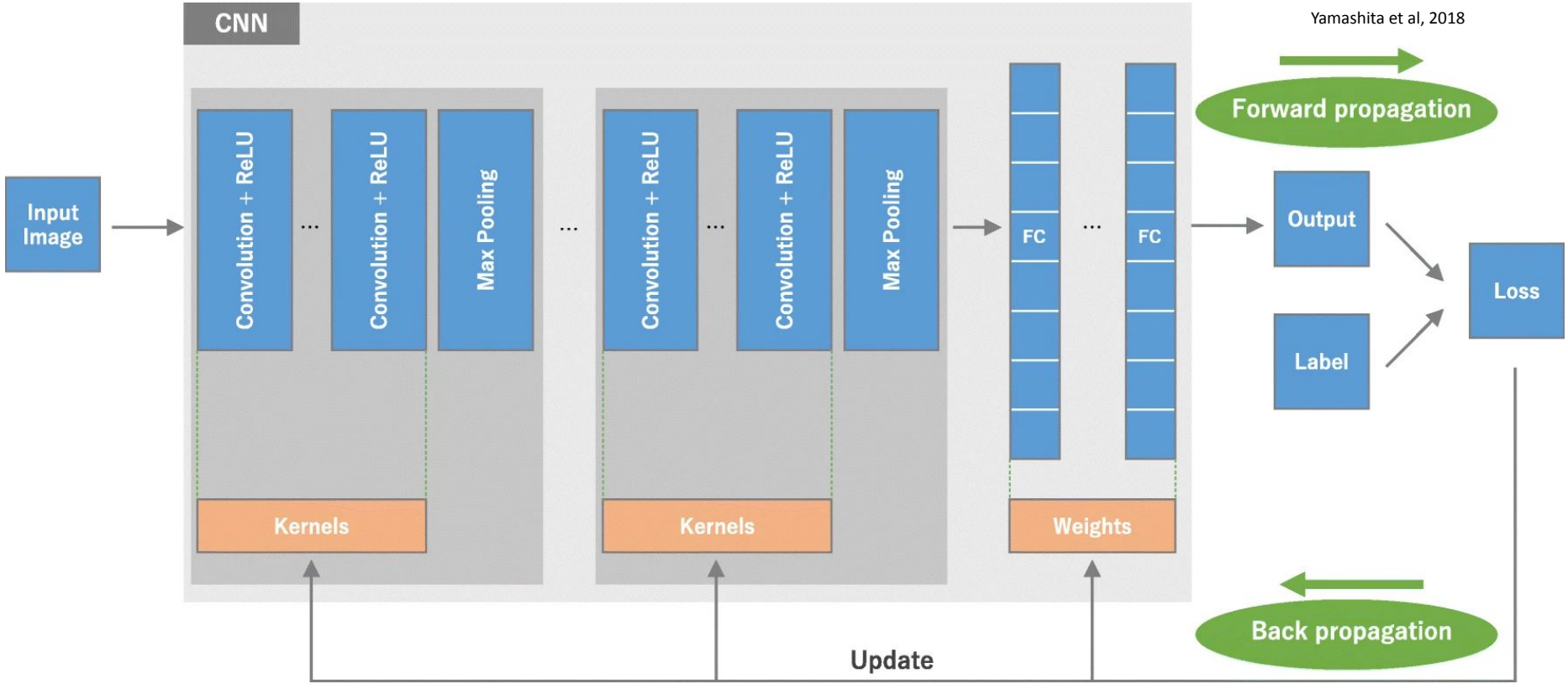
(d) Scattergram.

Convert the signals to images.

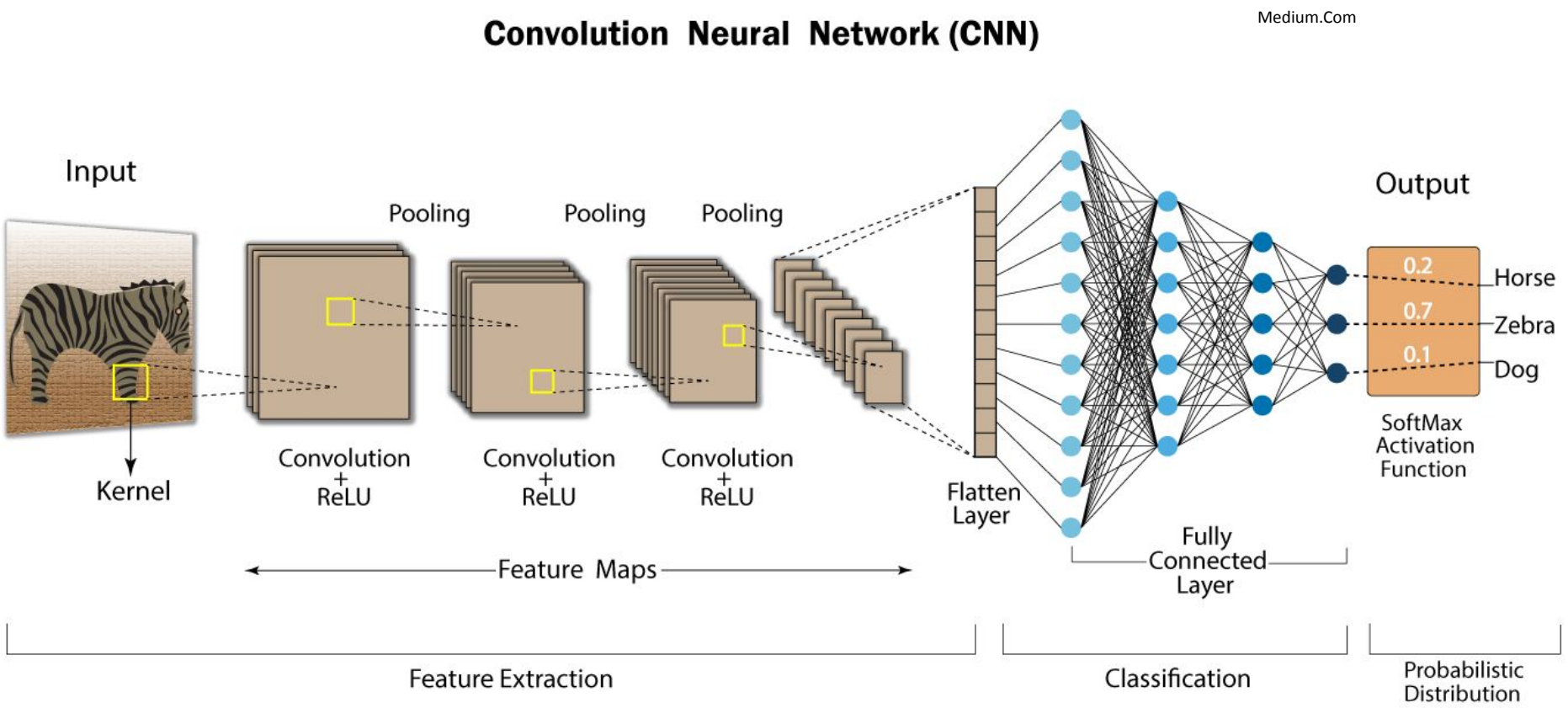
A study by Nanni et al 2020…



The CNN


CNN
Yamashita et al, 2018
Input Image
Convolution + ReLU
Convolution + ReLU
Max Pooling
Convolution + ReLU
Convolution + ReLU
Max Pooling
FC
FC
Kernels
Kernels
Weights
Forward propagation
Output
Label
Loss
Back propagation
Update

Neural networks



CNN

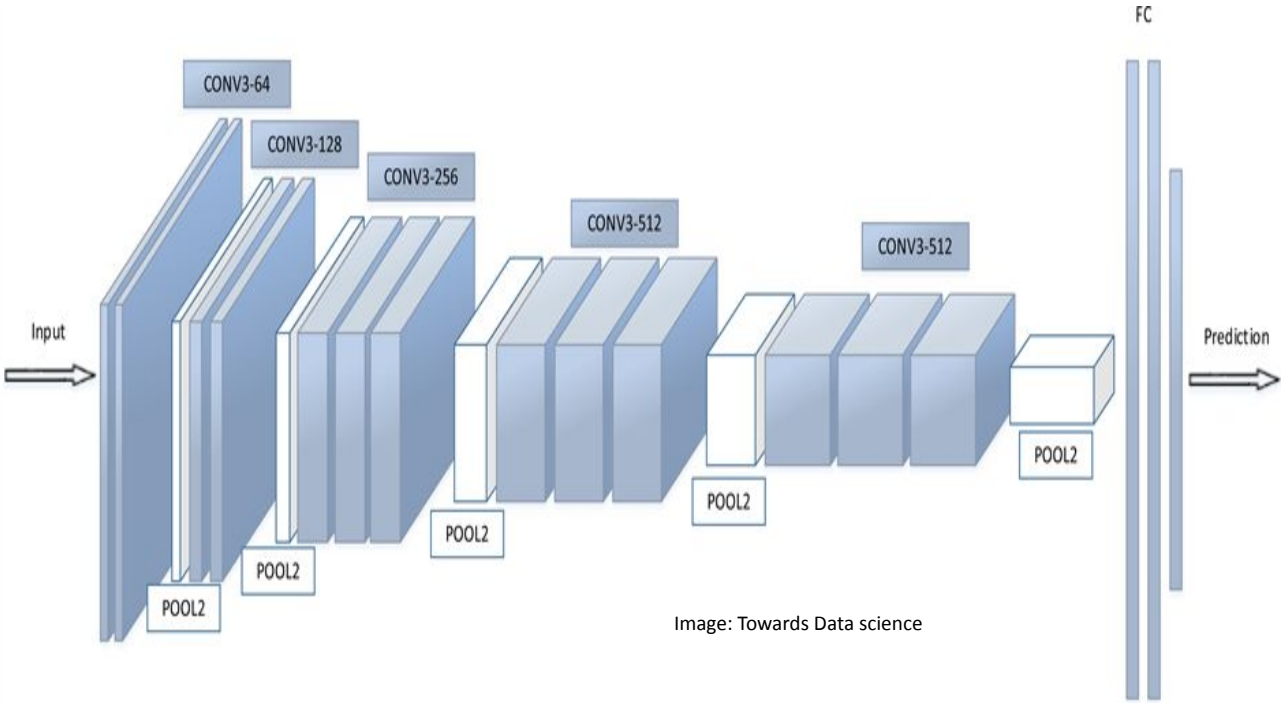


Image: Towards Data science

1. Convolutional Layer (CONV)

2. Pooling Layer (POOL)

3. Fully Connected Layer (FC)

CNN LAYERS

Convolutional Layer

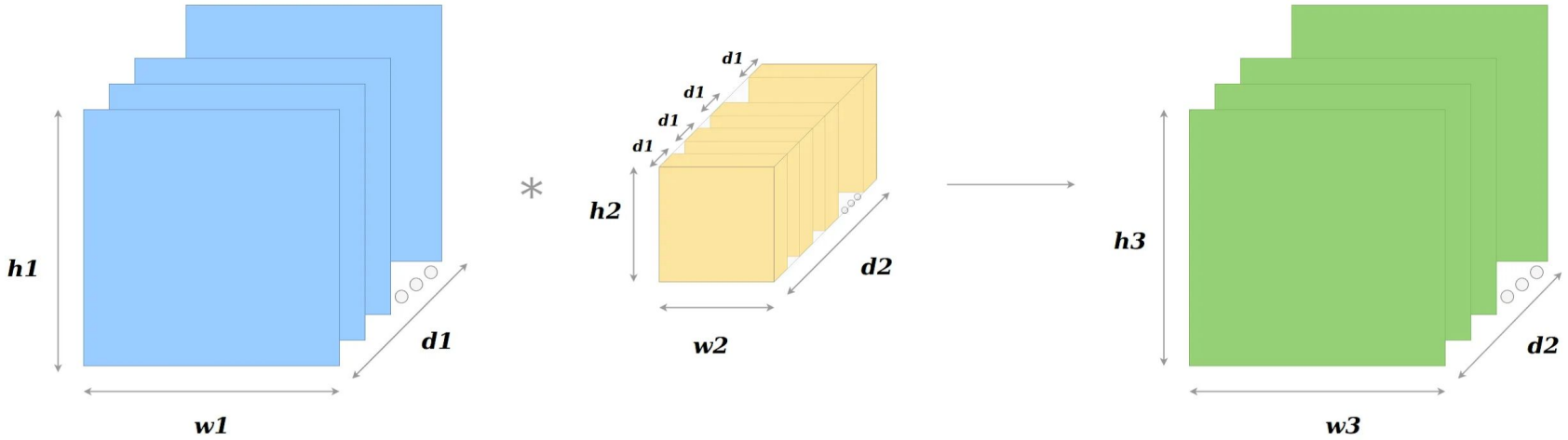


Image: Towards Data science

CNN LAYERS

Pooling Layers

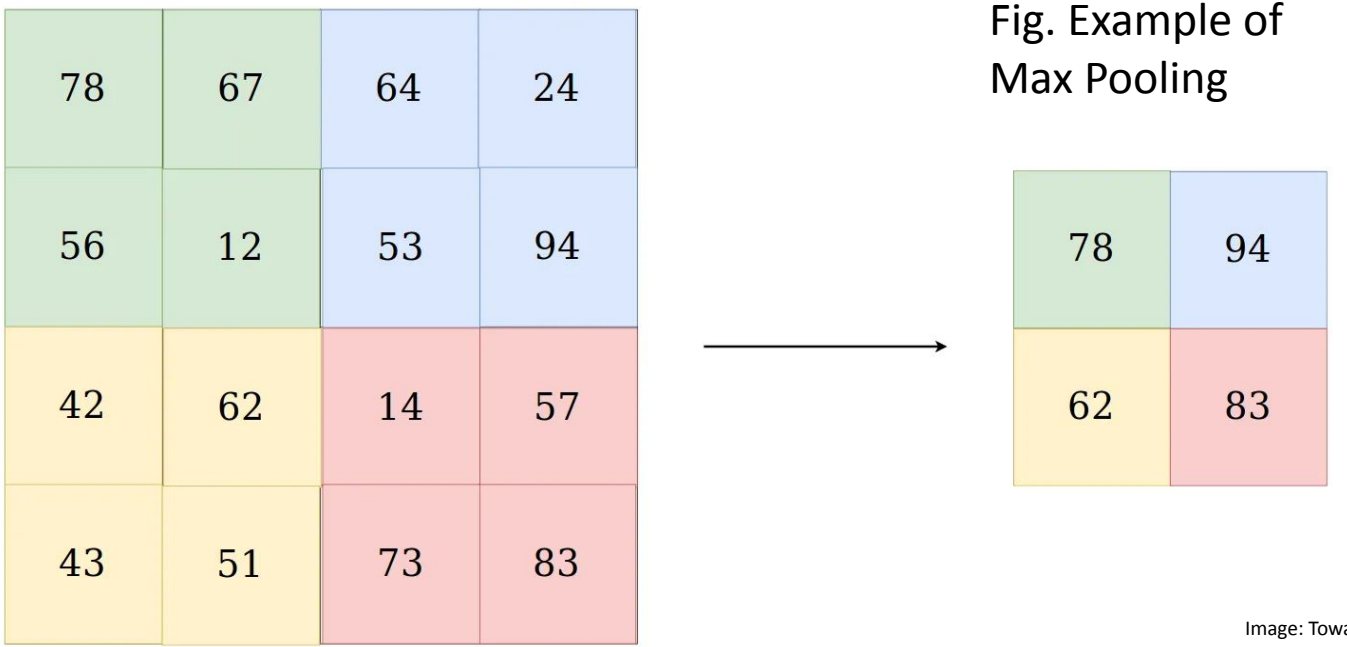
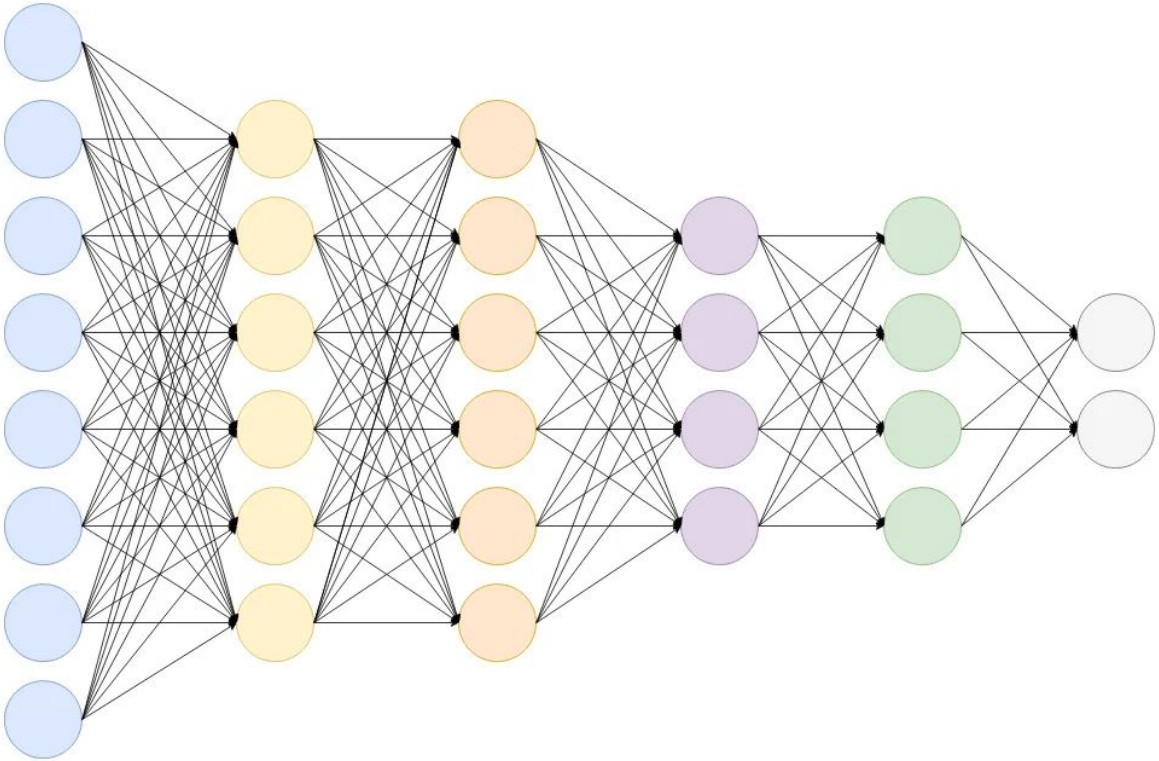


Fig. Example of Max Pooling

Image: Towards Data science

CNN LAYERS

Fully Connected Layers



Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

Overview of Hyperparameters

Learning rate

Learning rate controls how much to update the weight in the optimization algorithm.

Number of epochs

Number of epochs is the number of times the entire training set pass through the neural network. Increase the number of epochs (within a reasonable time limit) until we see a small gap between the test error and the training error.

Batch size

Mini-batch is usually preferable. A range of 16 to 128 is a good choice to test with. We should note that CNN is sensitive to batch size.

Number of Kernels & Kernel Size

Kernels are often referred to as “filters”. A typical starting point for number of kernels is 32. For, kernel size, you may begin with smaller numbers (typically 2 or 4).

Overview of Hyperparameters

Activation function

Usually, ReLu works well. Other alternatives are sigmoid, tanh.

Number of hidden layers and units

It is usually good to add more layers until the test error no longer improves. The trade off is that it is computationally expensive to train the network.

Checking out GoogLeNet

GoogLeNet is a popular CNN for Google Classification

I have an interesting example provided my Matlab.

You can click this link to check it out:

https://www.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-classify-new-images.html

U-Net Architecture

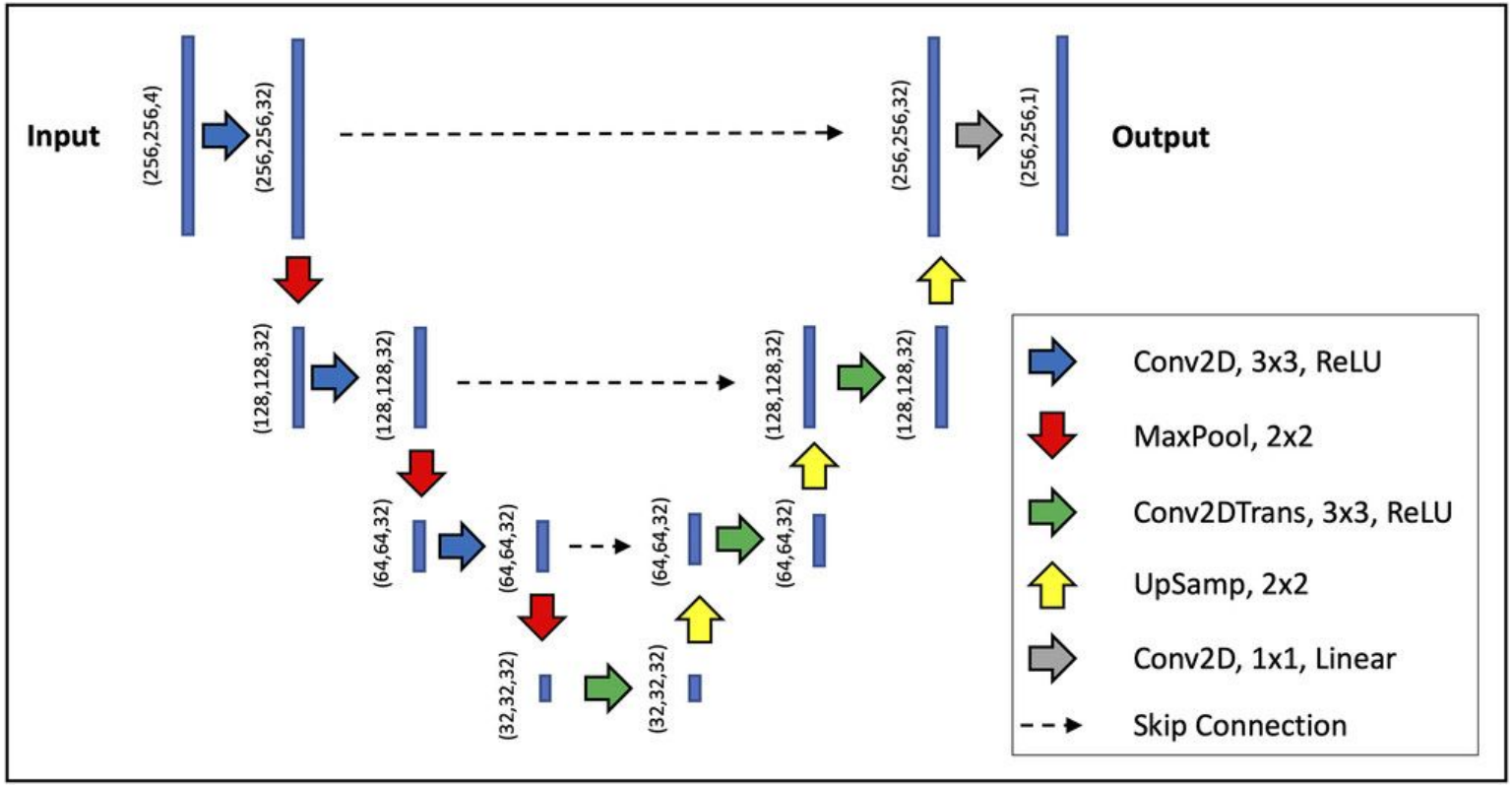
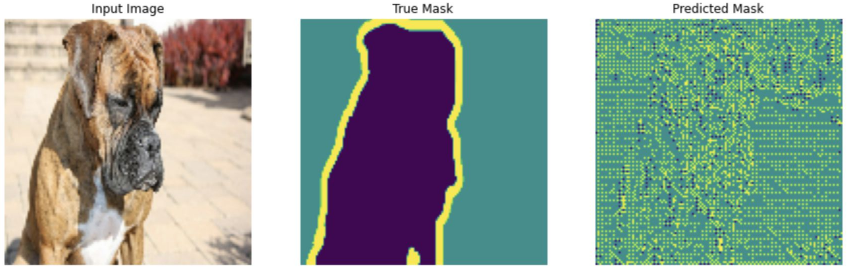
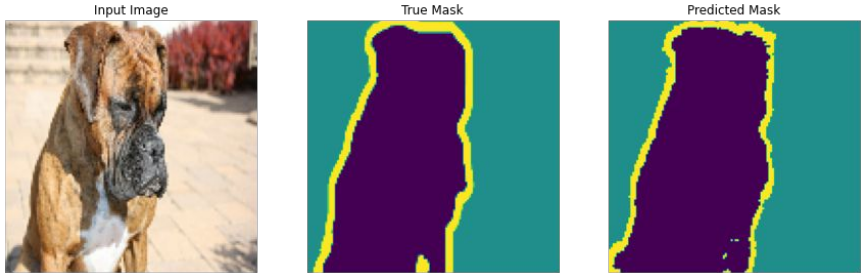


Image Segmentation with a CNN

Before training…



After training…



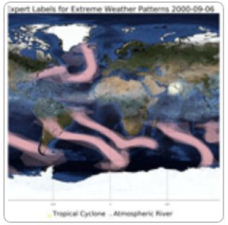
- Class 1: Pixel belonging to the pet.
- Class 2: Pixel bordering the pet.
- Class 3: None of the above/a surrounding pixel.

→ **Lab 2**

https://www.tensorflow.org/tutorials/images/segmentation

ClimateNet - using experts to label

ClimateNet: an expert-labeled open dataset and deep learning architecture for enabling high-precision analyses of extreme weather



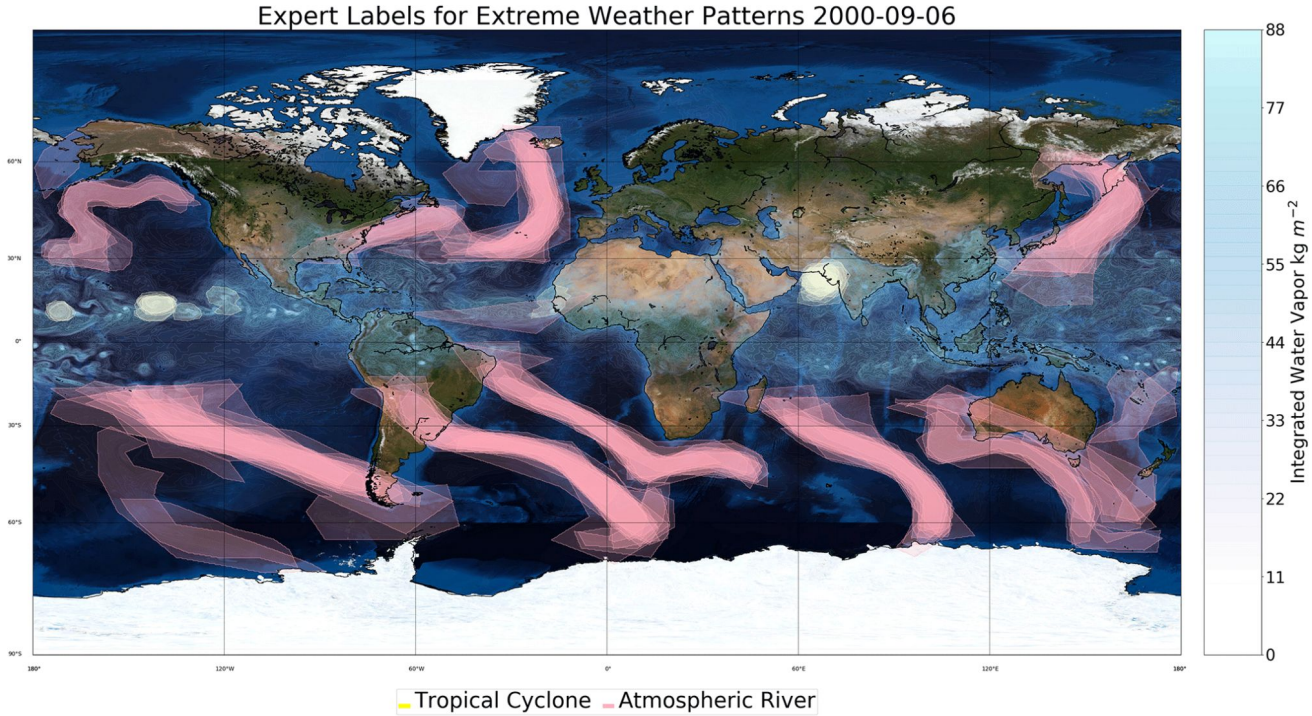
Prabhat[1,2,★], Karthik Kashinath[1,★], Mayur Mudigonda[10,★], Sol Kim[2], Lukas Kapp-Schwoerer[3], Andre Graubner[3], Ege Karaismailoglu[3], Leo von Kleist[3], Thorsten Kurth[4], Annette Greiner[1], Ankur Mahesh[2,1], Kevin Yang[2], Colby Lewis[2], Jiayi Chen[2], Andrew Lou[2], Sathyavat Chandran[5], Ben Toms[6], Will Chapman[7], Katherine Dagon[8], Christine A. Shields[8], Travis O'Brien[9,1], Michael Wehner[1], and William Collins[1,2]

https://gmd.copernicus.org/articles/14/107/2021/
https://portal.nersc.gov/project/ClimateNet/

ClimateNet: an expert-labeled open dataset and deep learning architecture for enabling high-precision analyses of extreme weather

Prabhat[1,2*], Karthik Kashinath[1*], Mayur Mudigonda[3,4*], Sol Kim[5], Lukas Kapp-Schwoerer[6], Andre Graubner[6], Ege Karaismailoglu[6], Leo von Kleist[7], Thorsten Kurth[8], Annette Greiner[1], Ankur Mahesh[1,2], Kevin Yang[2], Colby Lewis[9], Jiayi Chen[2], Andrew Lou[2], Sathyavat Chandran[2], Ben Toms[9], Will Chapman[1], Katherine Dagon[10], Christine A. Shields[9], Travis O'Brien[11,1], Michael Wehner[12], and William Collins[13,1]

ClimateNet - using experts to label

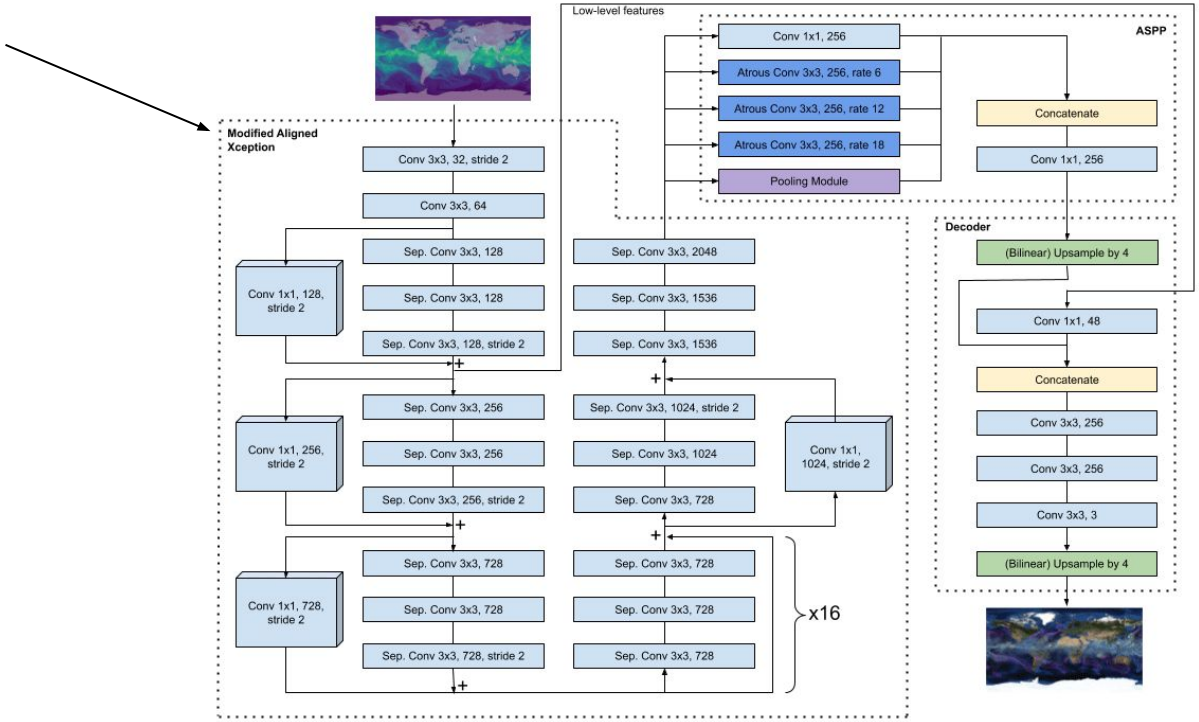


https://gmd.copernicus.org/articles/14/107/2021/

ClimateNet: an expert-labeled open dataset and deep learning architecture for enabling high-precision analyses of extreme weather

Prabhat[1,2,*], Karthik Kashinath[1,*], Mayur Mudigonda[10,*], Sol Kim[2], Lukas Kapp-Schwoerer[3], Andre Graubner[3], Ege Karaismailoglu[3], Leo von Kleist[3], Thorsten Kurth[4], Annette Greiner[1], Ankur Mahesh[1,1], Kevin Yang[2], Colby Lewis[2], Jiaqi Chen[2], Andrew Lou[2], Sathyavat Chandran[5], Ben Toms[6], Will Chapman[7], Katherine Dagon[8], Christine A. Shields[9], Travis O'Brien[10,1], Michael Wehner[2], and William Collins[10,1]

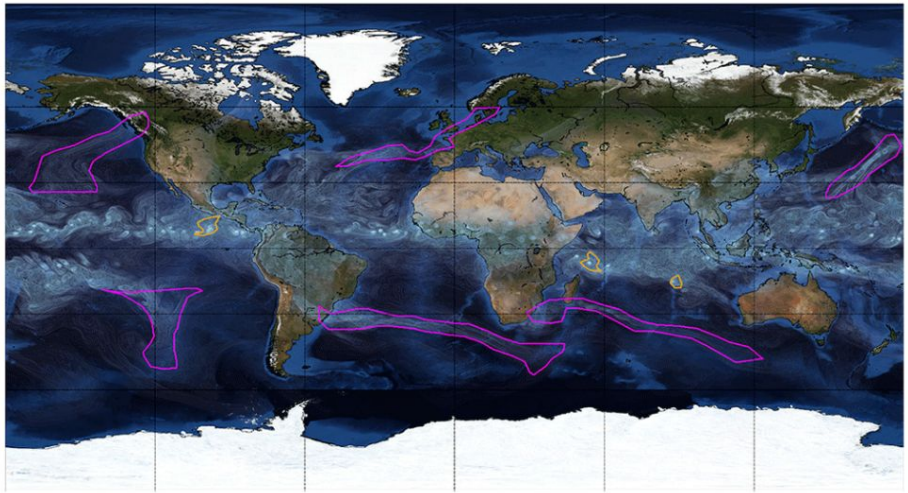
Image Segmentation w/ ClimateNet



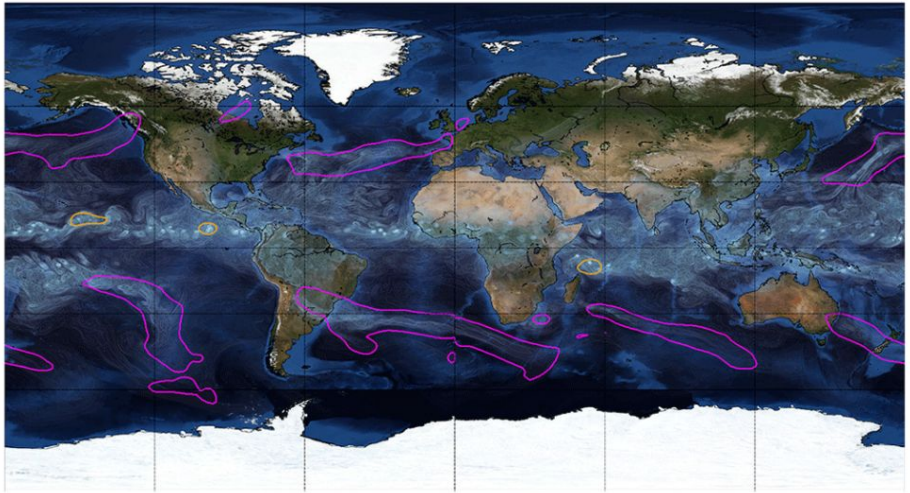
ClimateNet: an expert-labeled open dataset and deep learning architecture for enabling high-precision analyses of extreme weather

Prabhat[1,2*], Karthik Kashinath[1*], Mayur Mudigonda[3a*], Sol Kim[2], Lukas Kapp-Schwoerer[4], Andre Graubner[5], Ege Karaismailoglu[5], Leo von Kleist[5], Thorsten Kurth[2], Annette Greiner[2], Ankur Mahesh[2,1], Kevin Yang[2], Colby Lewis[2], Jiaqi Chen[2], Andrew Lou[2], Sathyavat Chandran[2], Ben Toms[6], Will Chapman[7], Katherine Dagon[8], Christine A. Shields[9], Travis O'Brien[10,11], Michael Wehner[12], and William Collins[13,2]

Image Segmentation w/ ClimateNet



Truth (Labeled)



Predicted by Trained CNN

Trained on ClimateNet

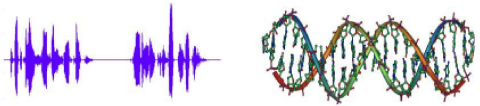
https://gmd.copernicus.org/articles/14/107/2021/

Idea

Use temporal covariations/context to refine prediction at some time t

Recurrent Neural networks

Sequence learning is the study of machine learning algorithms designed for sequential data



Sequential data: data points come in order and successive points may be **dependent**, e.g.,

- Letters in a word
- Words in a sentence/document
- Phonemes in a spoken word utterance
- Time series
- Frames in a video, etc.

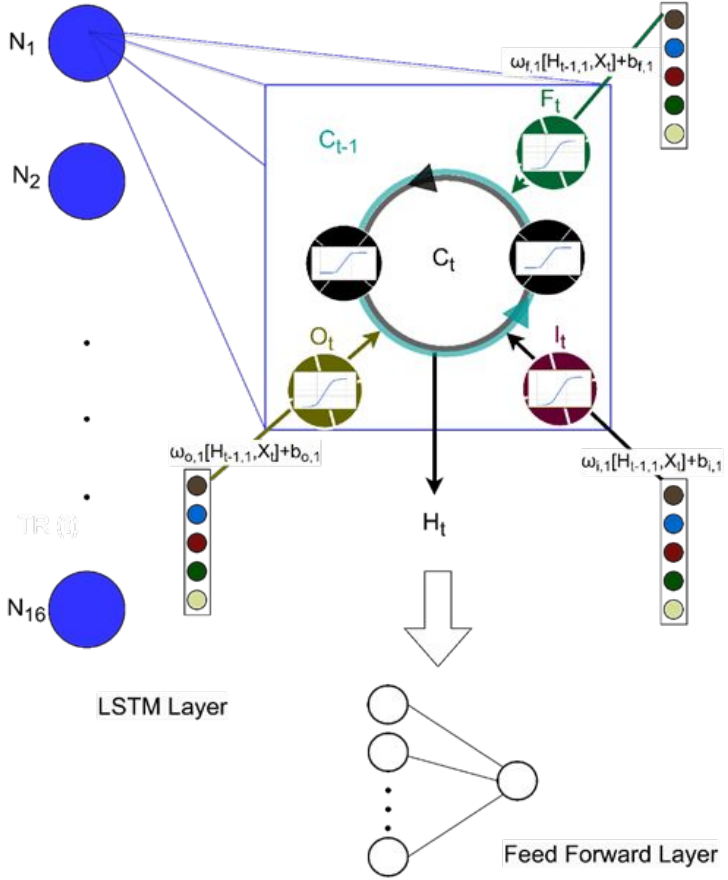
So far, we assume that data points in a dataset are i.i.d (independent and identically distributed)

Does not hold in many applications

Recurrent Neural networks

- Plain CNNs are not typically good at length-varying input and output.
- Difficult to define input and output
  - Remember that
    - Input image is a 3D tensor (width, length, color channels)
    - Output is a fixed number of classes.
  - Sequence could be:
    - "I know that you know that I know "
    - Or "I don't know"
- Input and output are strongly correlated within the sequence.
- (Still, people figured out ways to use CNN on temporal learning)
  - i.e, CNN for time-series data we discussed earlier

An example of a RNN



The RNN architecture, built upon a network of neurons, is similar to the FFN structure. However, there is a difference within the hidden layer composition, where, as opposed to possessing only one **FF layer**, the RNN includes a preceding Long Short-Term Memory (**LSTM**) layer (**Fig 6**). First, a concatenated input vector of the predictors, **X** at each time step enters each neuron. X is then concatenated with the hidden state vector, **H**, at the previous time step. Then, via gradient descent optimization, unique (per neuron) input weight and bias are calculated. The input gate, **i**, controls the extent of input information entering the cell state (Schmidt, 2019; Tsantekidis et al., 2022), and the computations for each neuron, **n**, of the 16 neurons of the RNN layer are as follows:

$$\text{[illegible]} \quad \text{... (7)}$$

After the initial computation, the input enters the sigmoid activation, $\sigma^{[illegible]}$ function:

$$\text{[illegible]} \quad \text{... (8)}$$

Now, the forget gate, **f**, also receives the input vector of predictors; yet, it's function is to filter out irrelevant information from the previous cell state (Schmidt, 2019) .The calculations for f are as follows:

$$\text{[illegible]} \quad \text{... (9)}$$

Then, the sigmoid function is applied:

$$\text{[illegible]} \quad \text{... (10)}$$

With the productions of the input gate and the forget gate, the cell state, **C**, is computed. In the cell state computations, the **tanh** activation function, $\tanh$ is applied:

$$\text{[illegible]} \quad \text{... (11)}$$

Additionally, the input vector of predictors also passes through the output gate. The role of the output gate is to control the flow from the cell state to the hidden state and ultimately producing the output of the LSTM neuron (Chung et al., 2014). The output gate calculations are shown here:

$$\text{[illegible]} \quad \text{... (12)}$$

$$\text{[illegible]} \quad \text{... (13)}$$

Finally, with the product of the output gate, the hidden state, the information that passes to the next layer (the FF layer) is computed. The figuration is as follows:

$$\text{[illegible]} \quad \text{... (14)}$$

The process continues through the FF layer to predict SF counts. Here, the FF has 16 neurons. The RNN model undergoes 1000 epochs, and it has a batch size of 32, a learning rate of 0.007, and a sequence length of 6 days.

Time series analysis

1. **Autoregressive models**

- Predict the next term in a linear sequence from a **fixed number of previous terms** using delay taps.

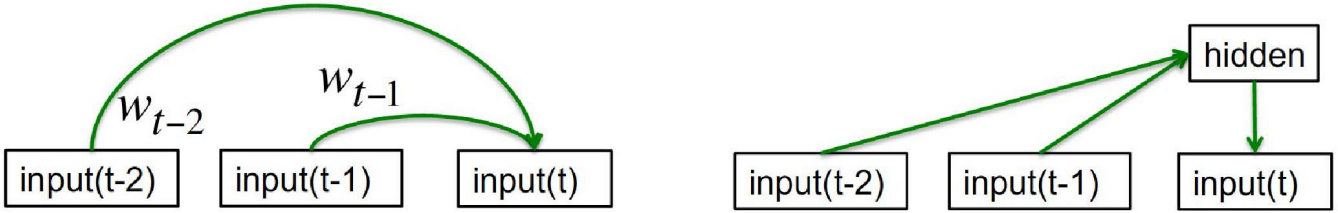
The autoregressive model specifies that the output variable depends linearly on its own previous values

AR(p) $Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \ldots + \beta_p Y_{t-p} + e_t$

Sequence labeling

2. **Feed-forward neural nets**

- These **generalize autoregressive models** by using one or more layers of non-linear hidden units



Memoryless models: limited word-memory window; hidden state cannot be used efficiently.

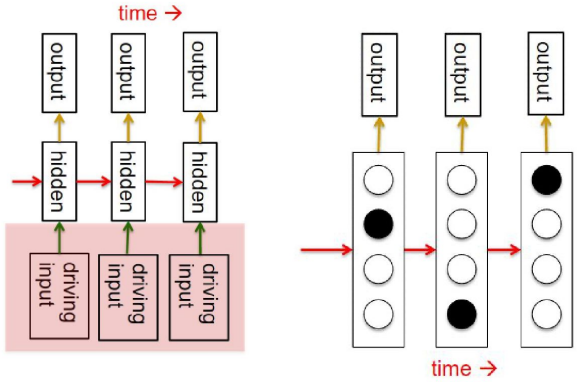
Time series analysis

3**. Linear Dynamical Systems**

- Generative models. They have a hidden state that cannot be observed directly.

4. **Hidden Markov Models**

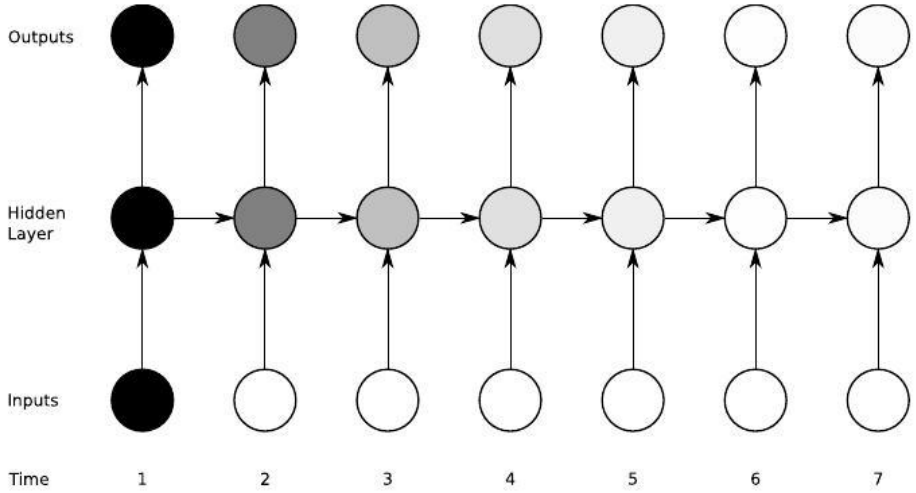
- Have a **discrete one-of-N hidden state**. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.



Memoryful models,
time-cost to infer the hidden state distribution.

Recurrent Neural Networks

- Update the hidden state in a deterministic nonlinear way.
- In a simple speaking case, we send the chosen word back to the network as input.

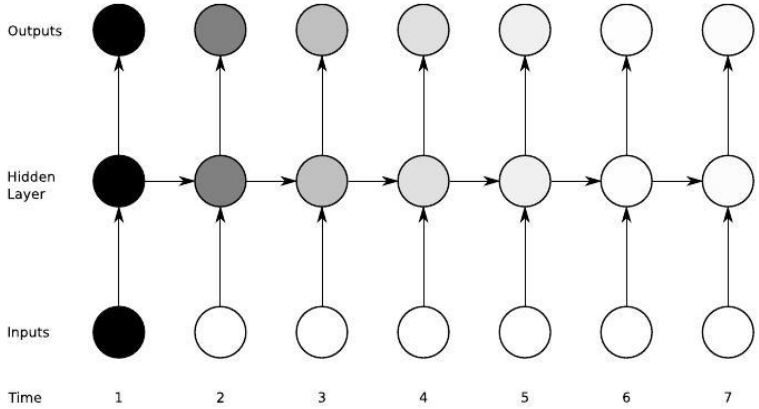


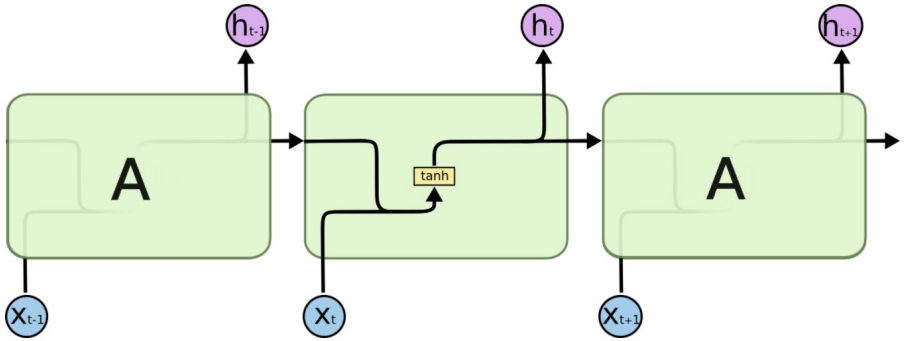
Vanilla forward path RNN

Recurrent Neural Networks

The forward pass of a vanilla RNN

1. The same as that of a perceptron with a single hidden layer
2. Except that activations arrive at the (single) hidden layer from both the current external input and the hidden layer activations one step back in time.



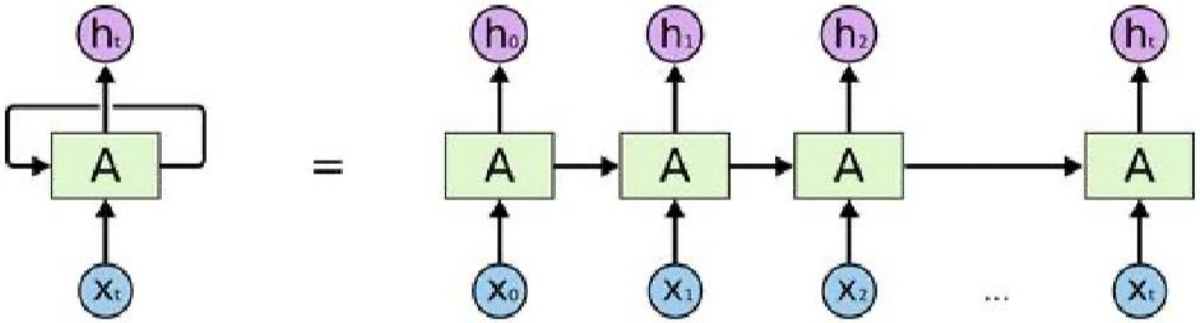


The repeating module in a standard RNN contains a single layer.

Vanilla backward path RNN

Recurrent Neural Networks

- Back-propagation through time
- It's just the standard back-propagation. Many times.

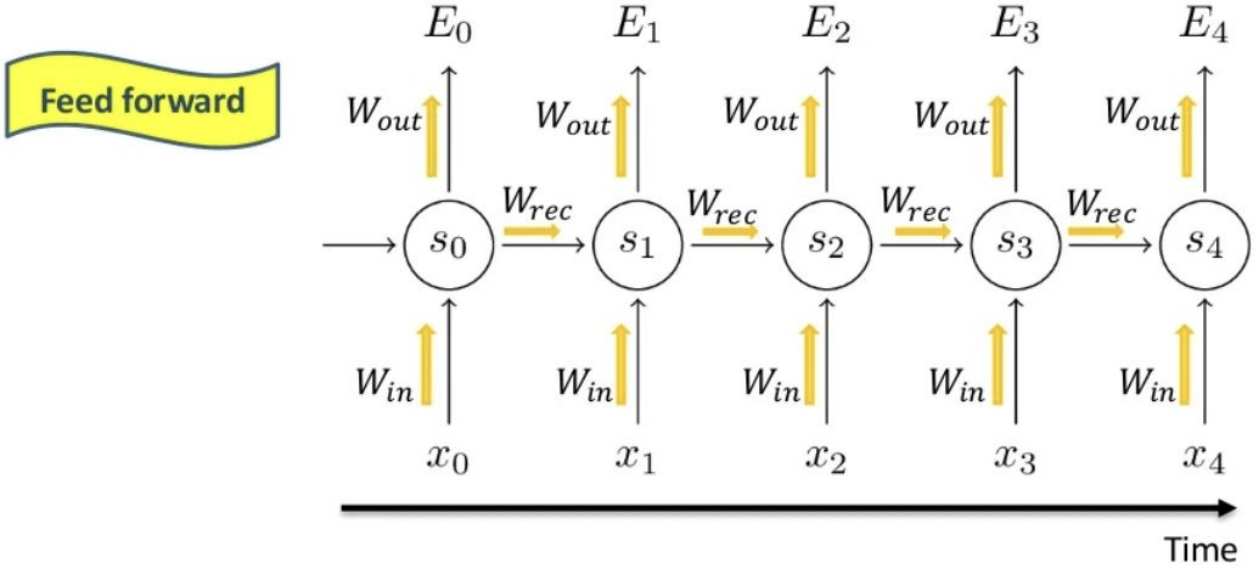


An unrolled recurrent neural network.

Vanilla forward path RNN

Recurrent Neural Networks

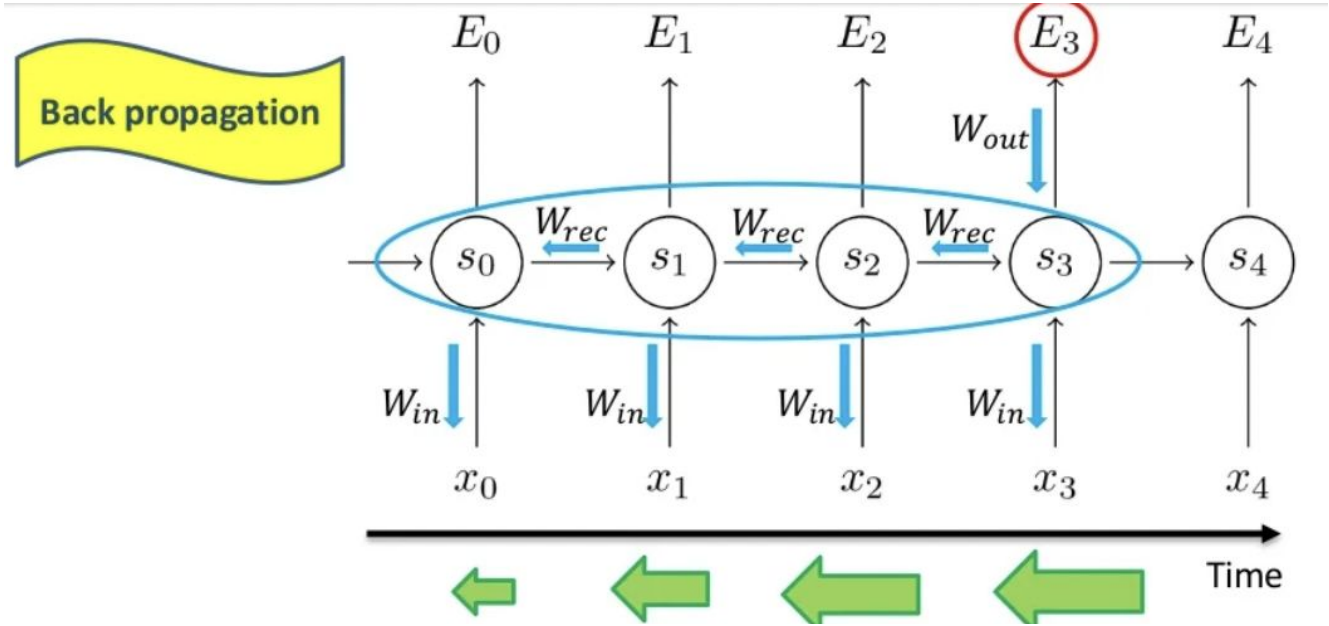
- Like standard back-propagation, RNN consists of a repeated application of the chain rule.



Vanilla forward path RNN

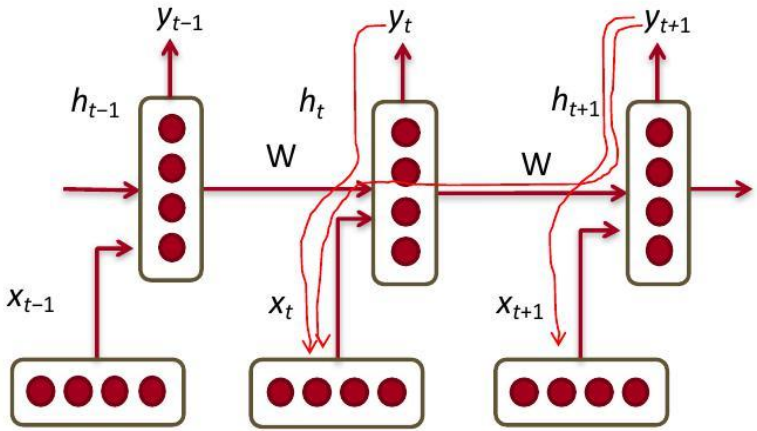
Recurrent Neural Networks

- Like standard back-propagation, RNN consists of a repeated application of the chain rule.
  - **Vanishing gradients**



Problem: Vanishing and exploding gradients

- Multiply the same matrix at each time step during back- prop



Vanilla backward path RNN

- Back-propagation through time
- The complete sequence of delta terms can be calculated by starting at t = T and recursively applying the below functions, decrementing t at each step.
- Note that $\delta_j^{T+1} = 0 \; \forall j$, since no error is received from beyond the end of the sequence.

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right), \qquad a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial a_j^t} \qquad b_h^t = \theta_h(a_h^t)$$

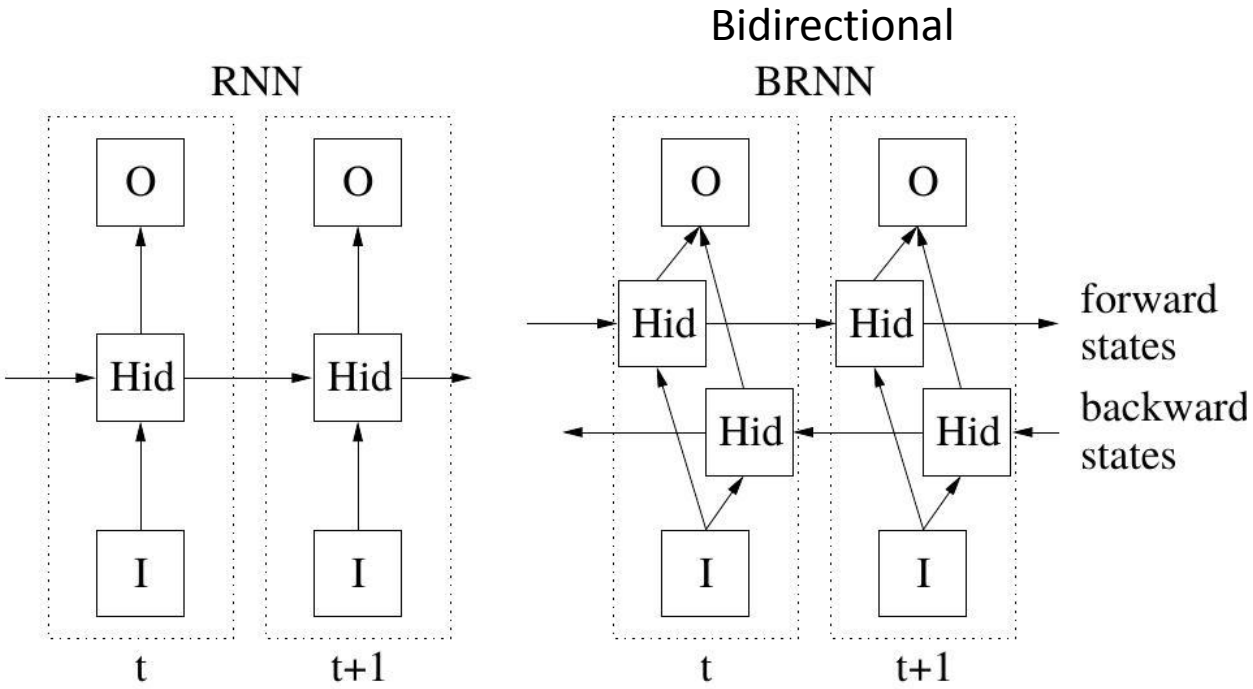
$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

- Finally, bearing in mind that the weights to and from each unit in the hidden layer are the same at every time-step, we sum over the whole sequence to get the derivatives with respect to each of the network weights

$$\frac{\partial O}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial O}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

Vanilla bidirectional path RNN

•For many time series analysis, we might have access to future.



LSTM: long short term memory

- The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially

- The **most effective solution so far is the Long Short Term Memory** (LSTM) architecture (Hochreiter and Schmidhuber, 1997).

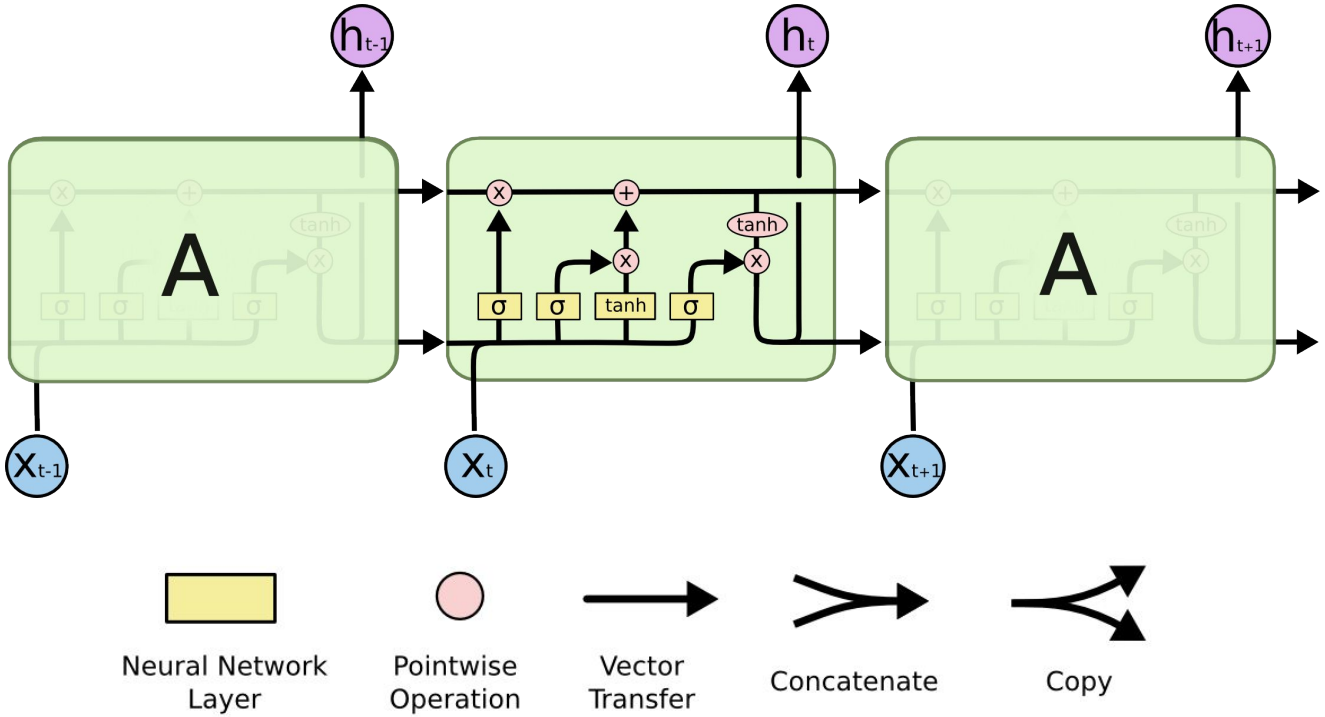
- The LSTM architecture consists of a set of recurrently connected subnets, known as **memory blocks**. These blocks can be thought of as a differentiable version of the memory chips in a digital computer.

  Write, read and reset operations for the cells

- The **input, output and forget gates**.

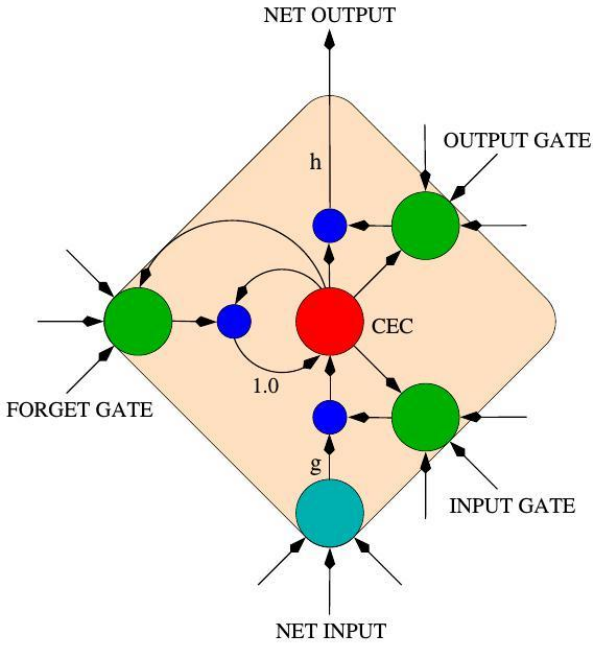
LSTM: long short term memory

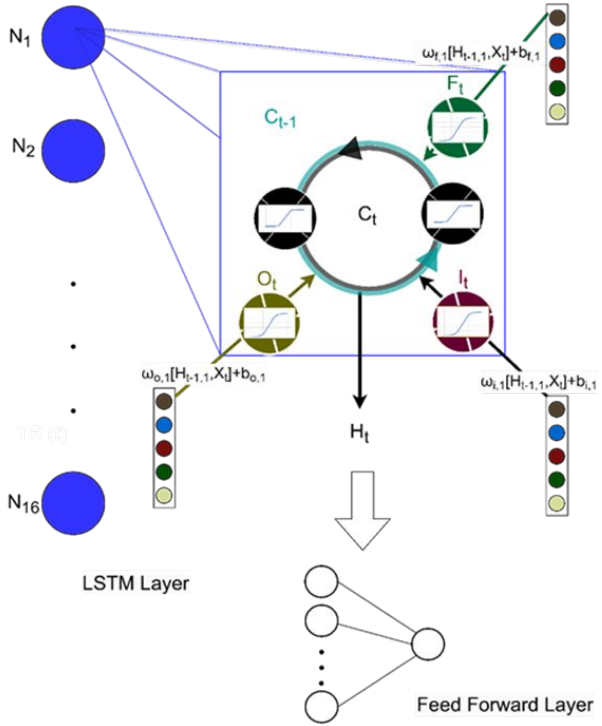
- Structure of an LSTM, contains four interacting layers



LSTM: long short term memory

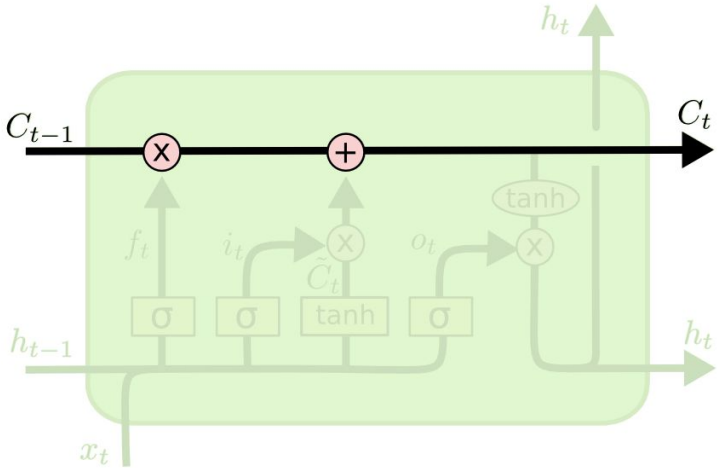
- Sometimes schematized this way

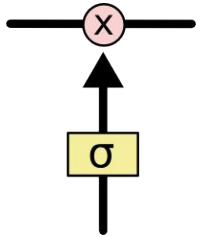




LSTM: long short term memory

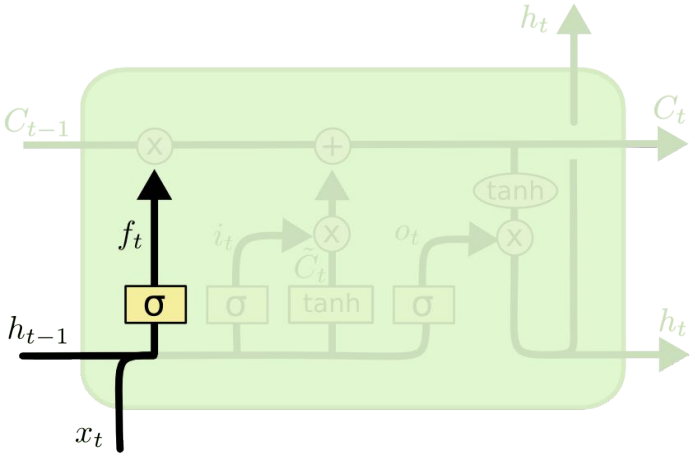
- Two key ideas of LSTM:
  - A backbone to carry state forward and gradients backward.
  - **Gating** (pointwise multiplication) to modulate information flow. Sigmoid makes 0 < gate < 1.





LSTM: long short term memory

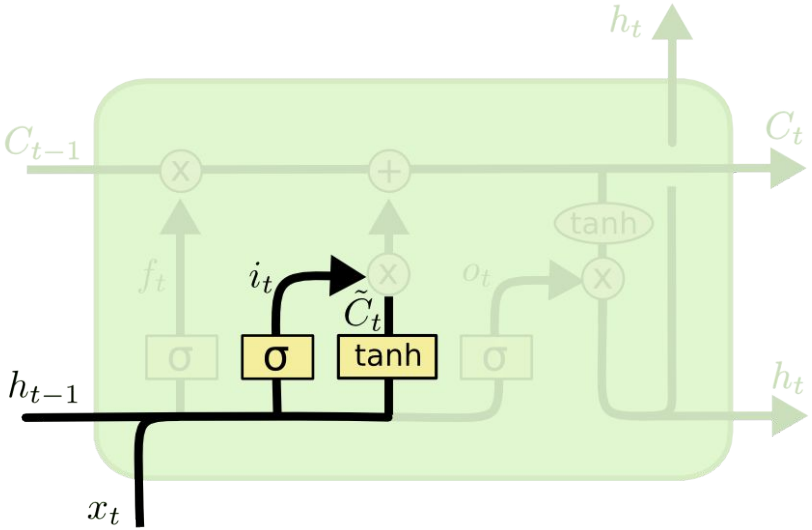
- The *f* gate is '**forgetting**.' Use previous state, *C,* previous output, *h*, and current input, *x*, to determine how much to suppress previous state.
- E.g., *C* might encode the fact that we have a subject and need a verb. Forget that when verb found.



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

LSTM: long short term memory

- Input gate *i* determines which values of *C* to **update**.
- Separate tanh layer produces new state to add to *C.*

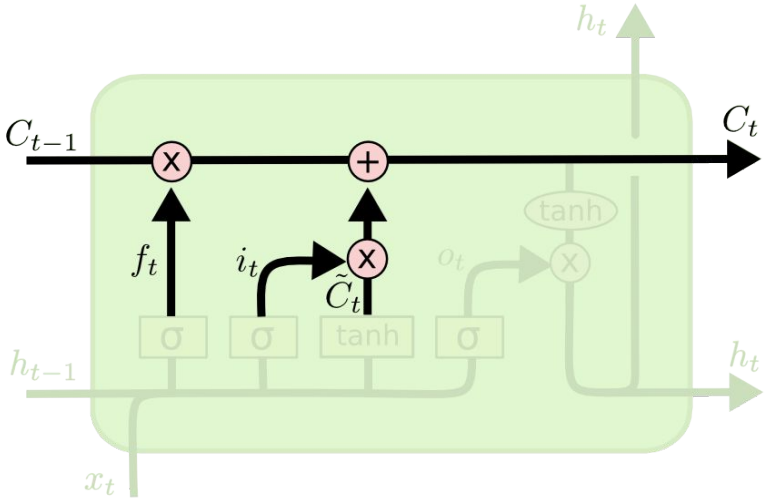


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

LSTM: long short term memory

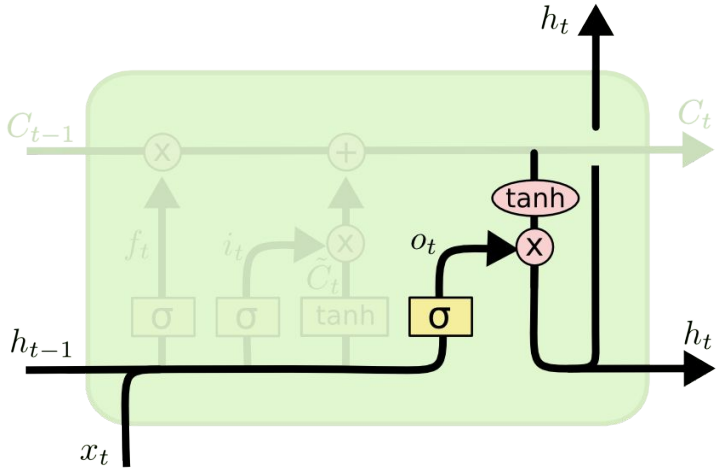
- **Forget gate** does pointwise modulation of *C.*
- **Input gate** modulates the tanh layer – this is added to *C.*



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

LSTM: long short term memory

- *o* is the **output gate**: modulates what part of the state *C* gets passed (via tanh) to current output *h*.
- E.g., could encode whether a noun is singular or plural to prepare for a verb.
- But the real features are learned, not engineered.



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

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

LSTM: long short term memory

- Comparison

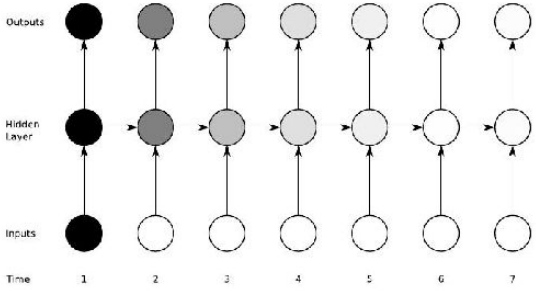


Figure 4.1: **Vanishing gradient problem for RNNs.** The shading of the nodes indicates the sensitivity over time of the network nodes to the input at time one (the darker the shade, the greater the sensitivity). The sensitivity decays exponentially over time as new inputs overwrite the activation of hidden unit and the network 'forgets' the first input.

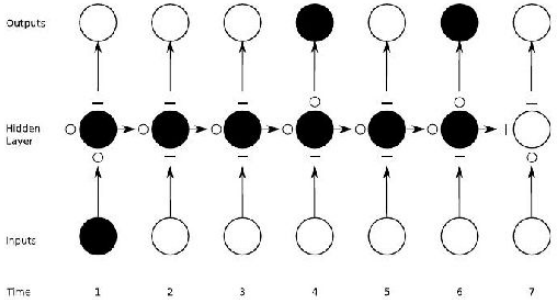
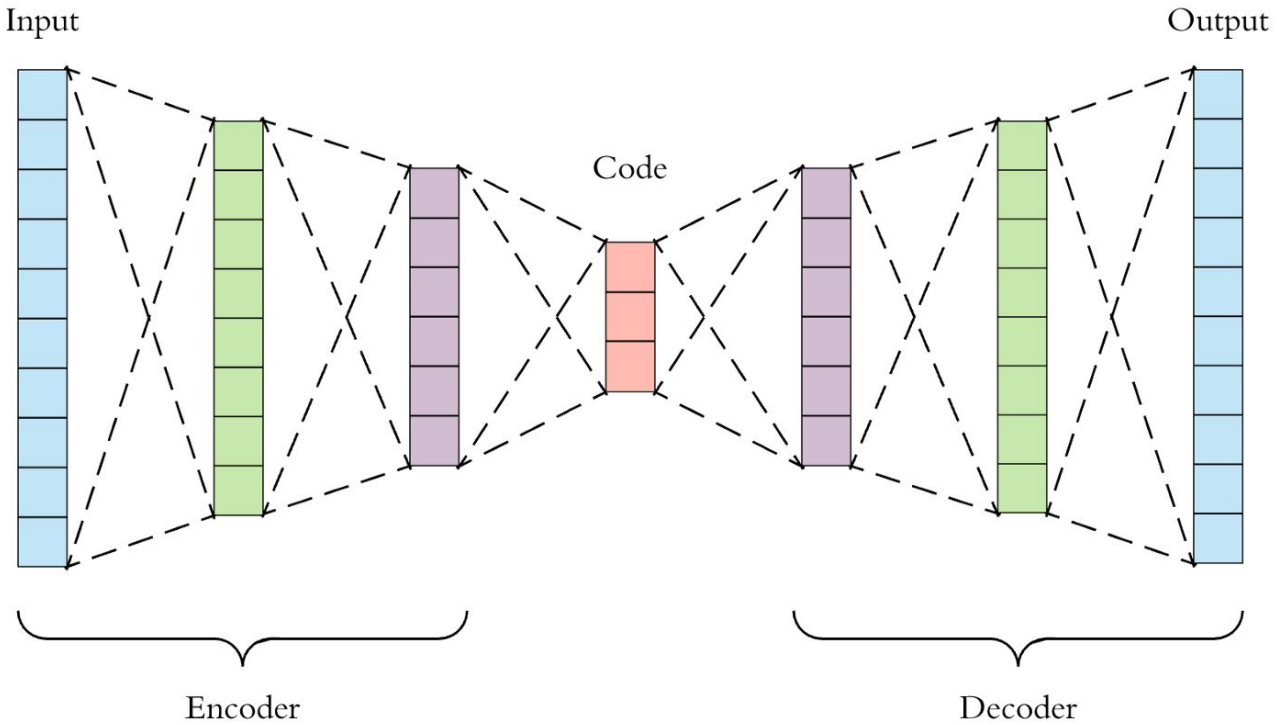


Figure 4.3: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the input unit at time one. The state of the input, forget, and output gate states are displayed below, to the left and above the hidden layer node, which corresponds to a single memory cell. For simplicity, the gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed, and the sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

Autoencoders

Even more advanced techniques

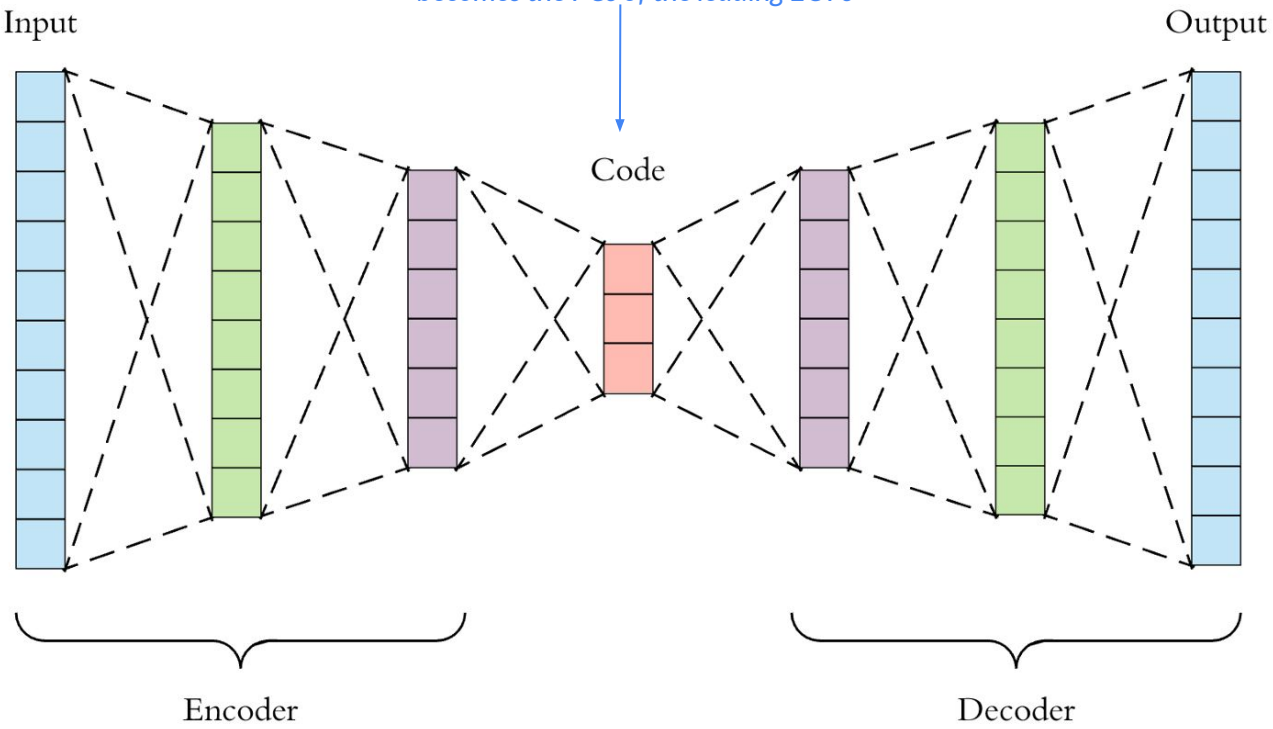


https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Autoencoders

Even more advanced techniques

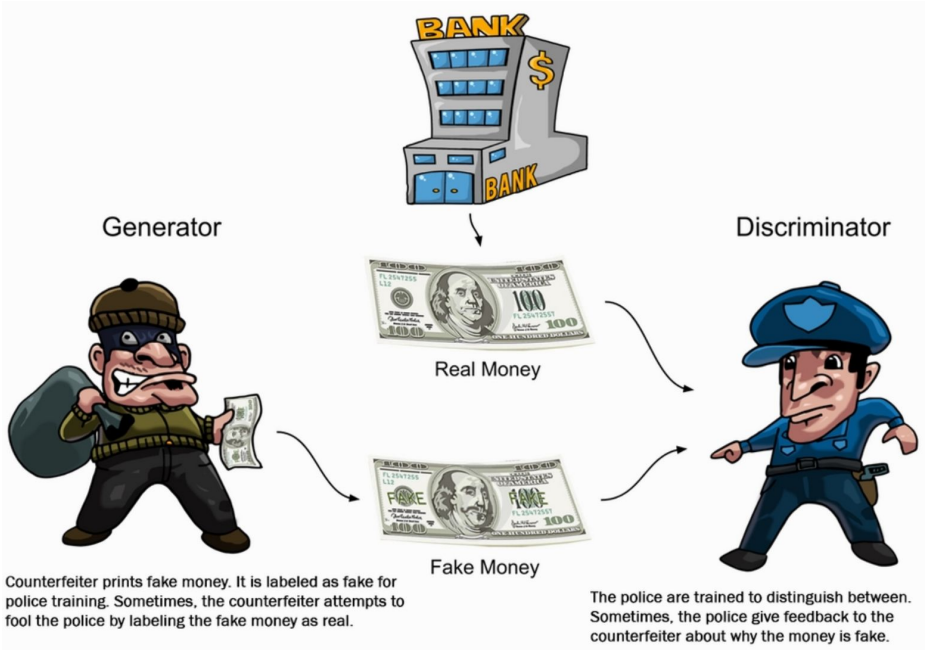
If your activation function is linear, this becomes the PCs of the leading EOFs



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

Generative Adversarial Networks

Even more advanced techniques



- First introduced by Goodfellow et al. (2014)
- Competition between **generator** and **discriminator**

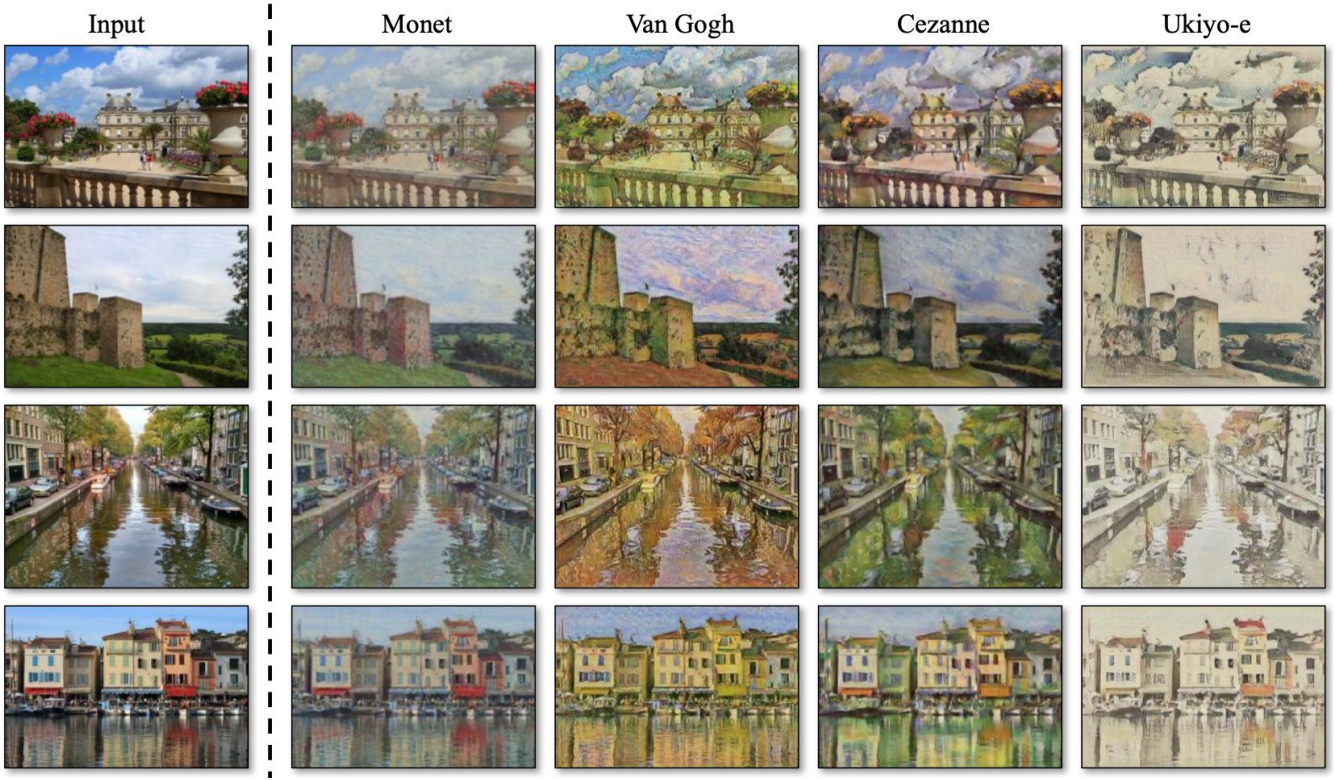
Figure by Atienza, Rowel. Advanced Deep Learning with Keras: Apply deep learning techniques, autoencoders, GANs, variational autoencoders, deep reinforcement learning, policy gradients, and more. Packt Publishing Ltd, 2018.

Visualizing Climate Change & Associated Hazards

https://mila.quebec/en/article/this-climate-does-not-exist-picturing-impacts-of-the-climate-crisis-with-ai-one-address-at-a-time/
https://thisclimatedoesnotexist.com/home

Style GANs generated these faces. They are not real.

https://arxiv.org/abs/1406.2661



Cycle GAN

https://arxiv.org/abs/1703.10593

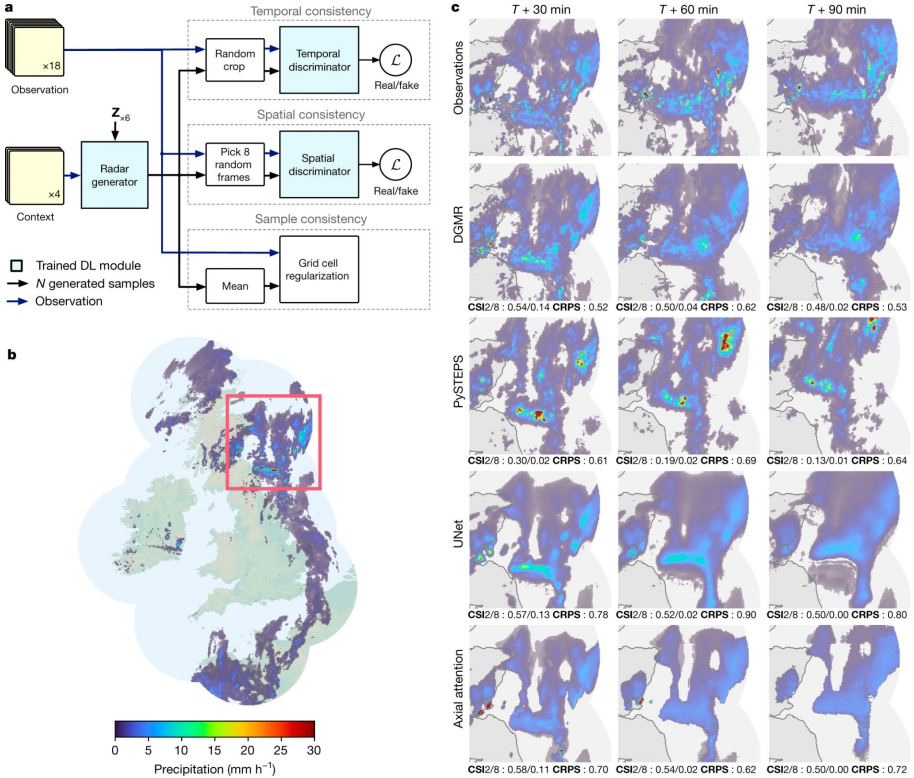
winter Yosemite → summer Yosemite

summer Yosemite → winter Yosemite

Cycle GAN

https://arxiv.org/abs/1703.10593

Fig. 1: Model overview and case study of performance on a challenging precipitation event starting on = 24 June 2019 at 16:15 UK, showing convective cells over eastern Scotland.



DGMR is better able to predict the spatial coverage and convection compared to other methods over a longer time period, while not over-estimating the intensities, and is significantly preferred by meteorologists (93% first choice, $n = 56$, $P < 10^{-4}$). **a**, Schematic of the model architecture showing the generator with spatial latent vectors **Z**. **b**, Geographic context for the predictions. **c**, A single prediction at $T + 30$, $T + 60$ and $T + 90$ min lead time for different models. Critical success index (CSI) at thresholds 2 mm h^{-1} and 8 mm h^{-1} and continuous ranked probability score (CRPS) for an ensemble of four samples shown in the bottom left corner. For axial attention we show the mode prediction. Images are 256 km × 256 km. Maps produced with Cartopy and SRTM elevation data[46].

nature

Explore content | About the journal | Publish with us

nature > articles > article

Article | Open Access | Published: 29 September 2021

Skilful precipitation nowcasting using deep generative models of radar

Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, Rachel Prudden, Amol Mandhane, Aidan Clark, Andrew Brock, Karen Simonyan, Raia Hadsell, Niall Robinson, Ellen Clancy, Alberto Arribas & Shakir Mohamed

Nature **597**, 672–677 (2021) | Cite this article

95k Accesses | **24** Citations | **855** Altmetric | Metrics

GANs for Nowcasting

https://arxiv.org/abs/2005.10374

Performance Measures

*Note- these are only used for **classification** problems!

Confusion Matrix: summary of prediction results

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Assessment Scores:

Precision = TP / (TP + FP); (between 0 and 1)
Accuracy of positive predictions

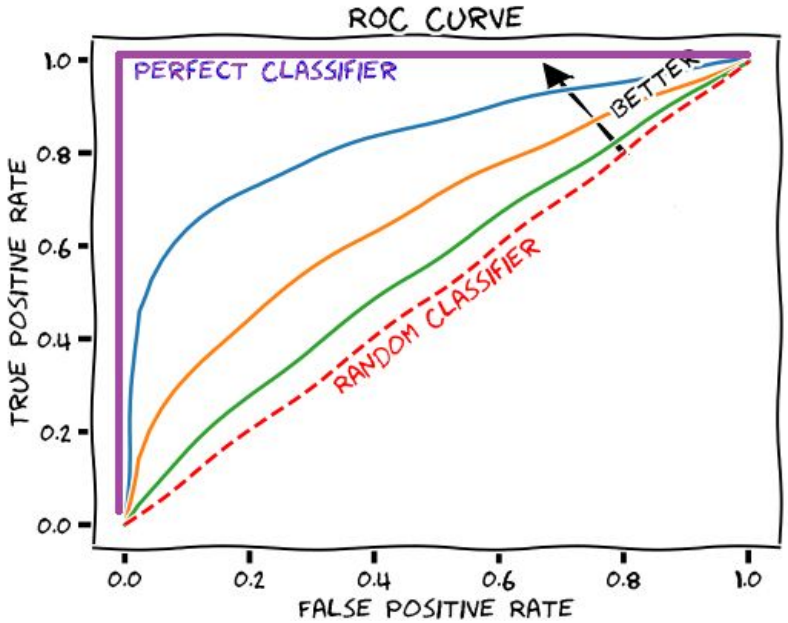
Recall = TP/ (TP + FN); (between 0 and 1)
Ratio of positive instances correctly detected by classifier

F1 score: harmonic mean of precision and recall; (between 0 and 1)

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

Performance Measures

Receiver Operating Characteristic (ROC)

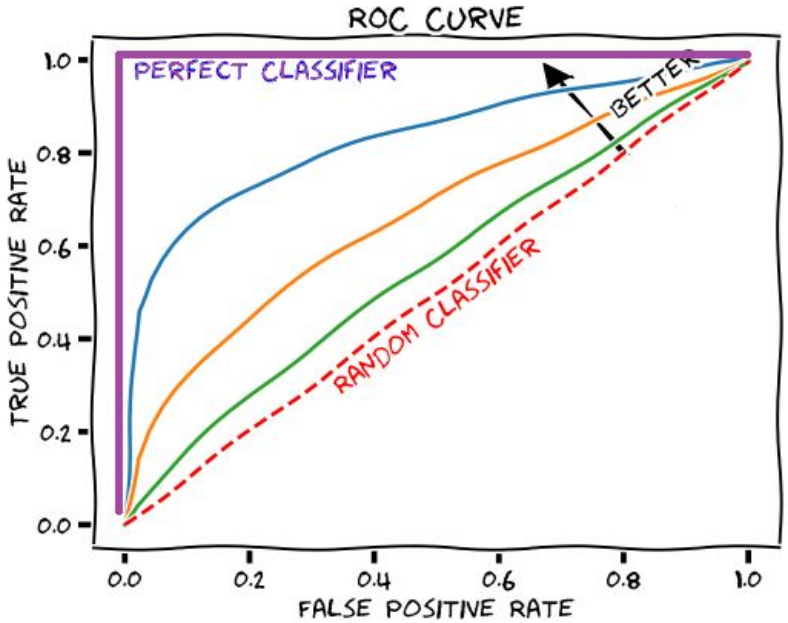


- Plots the false positive rate against the true positive rate for all possible thresholds
- Further from red dotted line, the better the model

- True positive rate = ratio of positive instances correctly detected by classifier (i.e. recall)
- False positive rate = ratio of negative instances that are *incorrectly* classified as positive

Performance Measures

Receiver Operating Characteristic (ROC)



Assessment Scores: measure the ***Area Under the Curve (AUC)***

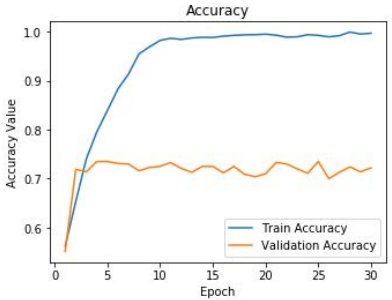
- Ranges from 0 to 1
- Higher AUC → better* model

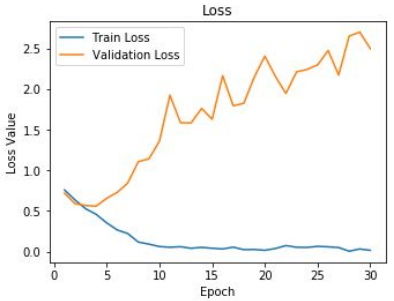
*remember, you (the scientist) decides what better means!

Performance Measures

Loss & Accuracy Assessment: User- Defined Loss Function

Overfit Model: training outperforms validation

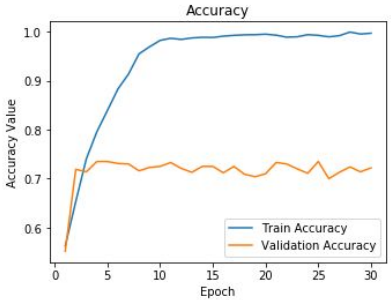


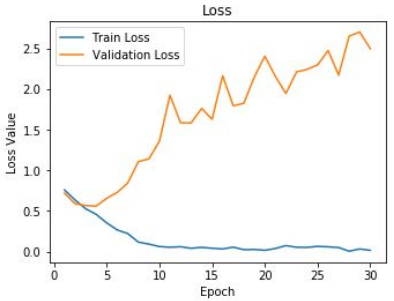


Performance Measures

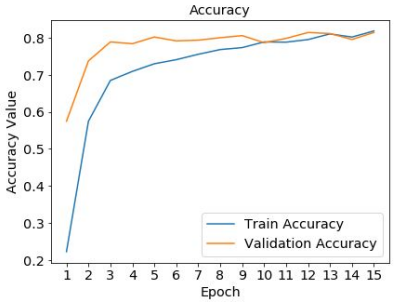
Loss & Accuracy Assessment: User- Defined Loss Function

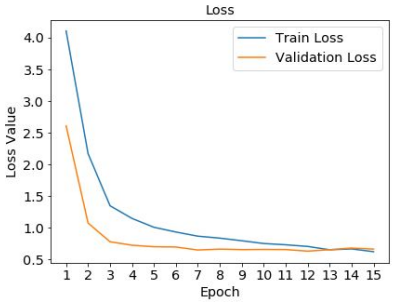
Overfit Model: training outperforms validation





Generalized Model! training matches validation





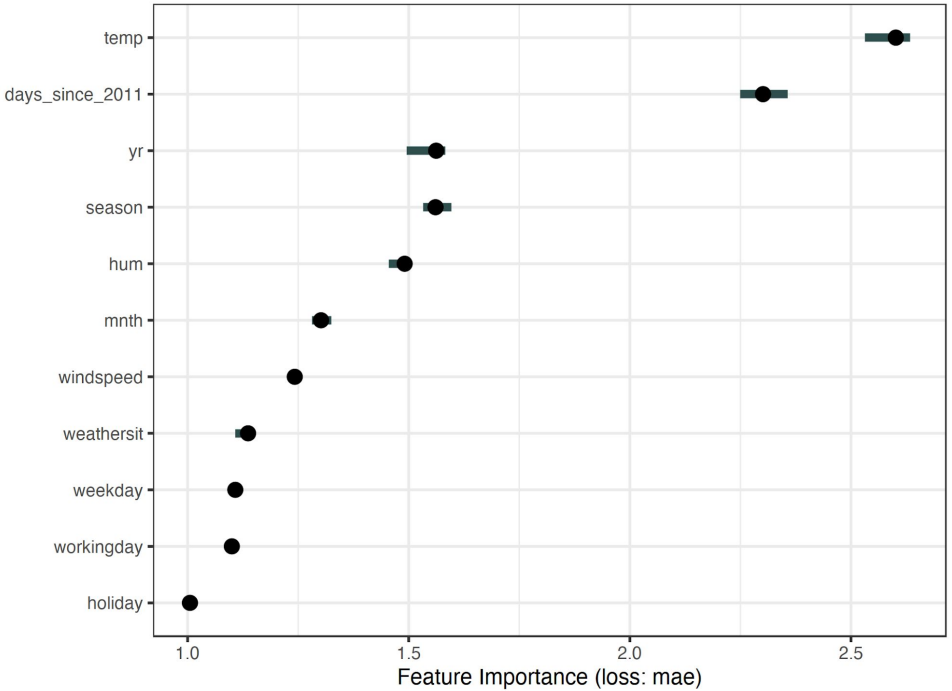
Permutation Importance

A way to detect features that are more important for accurate predictions, *Brieman (2001)*

For each feature, the feature values are shuffled, and new predictions made (with the shuffled feature values included).

If the model error increases after the shuffling, this feature is considered important. If there is no change in the model error, the feature in question adds no value to the model prediction.

Model error in predicting the number of rented bikes, given different weather and calendar related variables.



https://christophm.github.io/interpretable-ml-book/feature-importance.html

Permutation Importance

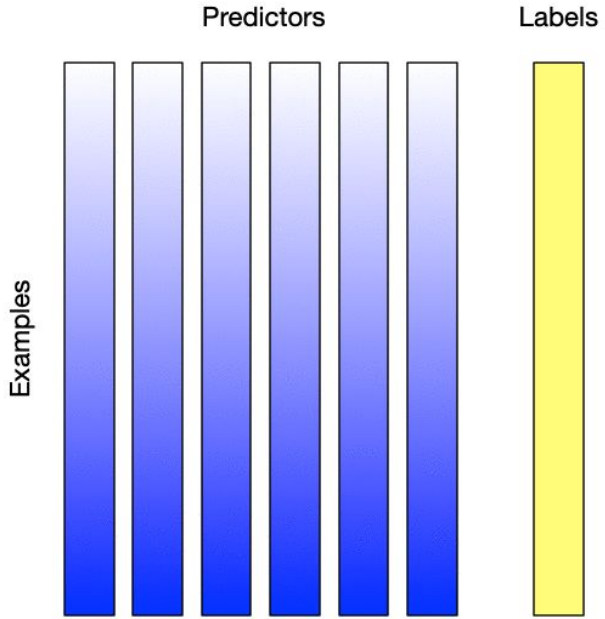
What if x is highly correlated with another predictor? → Multi-pass permutation

Freeze initially permuted predictor that degrades model performance the most, and re-permute other predictors to select second frozen predictor, and continue until all features are permuted and frozen

In comparison to the baseline model skill, can determine which predictors impact skill the most

Rank predictors based on error increase after permutation

McGovern et al. (2019)



Lakshmanan et al. (2015)

Lab