Aprendizagem Aplicada à Segurança

(Mestrado em Cibersegurança-DETI-UA)

LECTURE 6

Introduction do Deep Learning

Petia Georgieva
(petia@ua.pt)

DETI/IEETA – UA

Outline

- **Deep Neural Networks**
- **Convolutional Neural Networks (CNN)**
- **YOLO (You Only Look Once)**
- **Sequence models - Long-Short Term Memory (LSTM)**
- **Time Series Forecasting**

Deep Learning – major applications

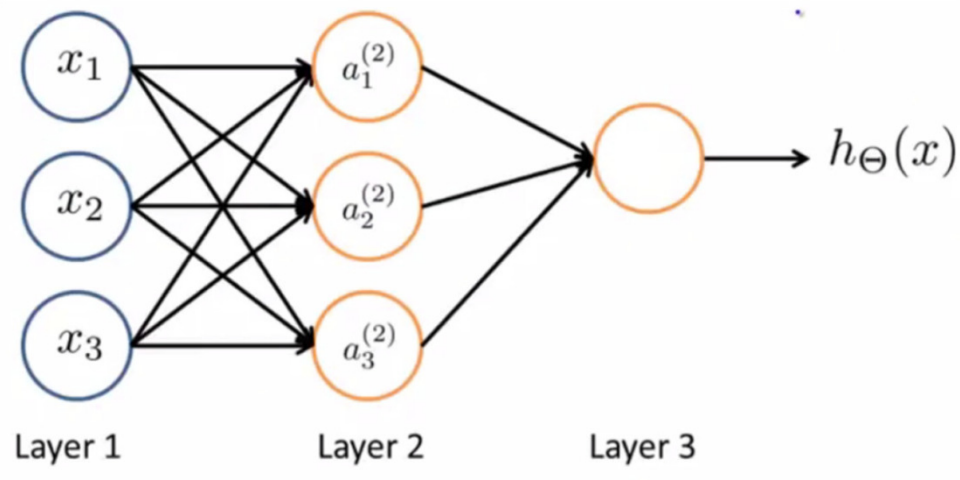
Image, Video, Speech, Text processing (large scale, big data) :

- Computer Vision
- Speech Recognition
- Natural Language Processing (topic classification, sentiment analysis, language translation, etc.)
- Bioinformatics (genomics, drug discovery)
- 5G+ communications
- Time series processing

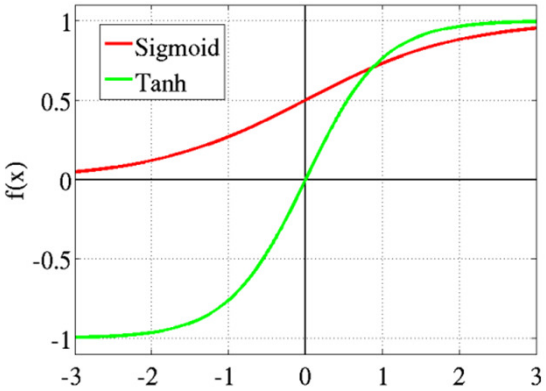
Machine Learning -
Shallow (3 layers) Neural Network

50 x 50 pixel images→ 2500 pixels
$n = 2500$ (7500 if RGB)

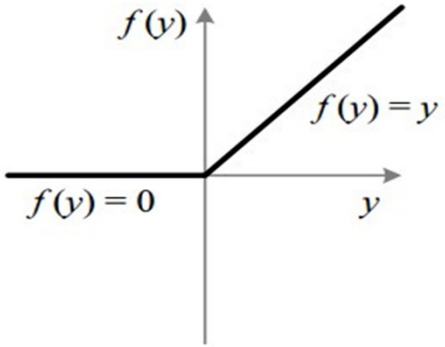
$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$



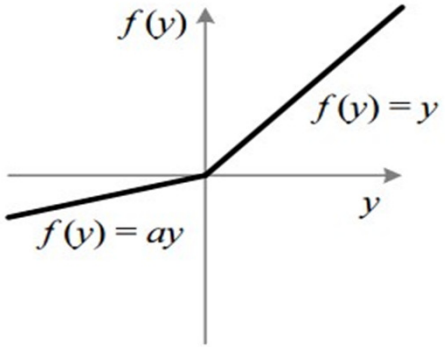
Typical Activation functions



Sigmoid (logistic)/Hyperbolic tangent



ReLU (Rectified Linear Unit)



Leaky ReLU

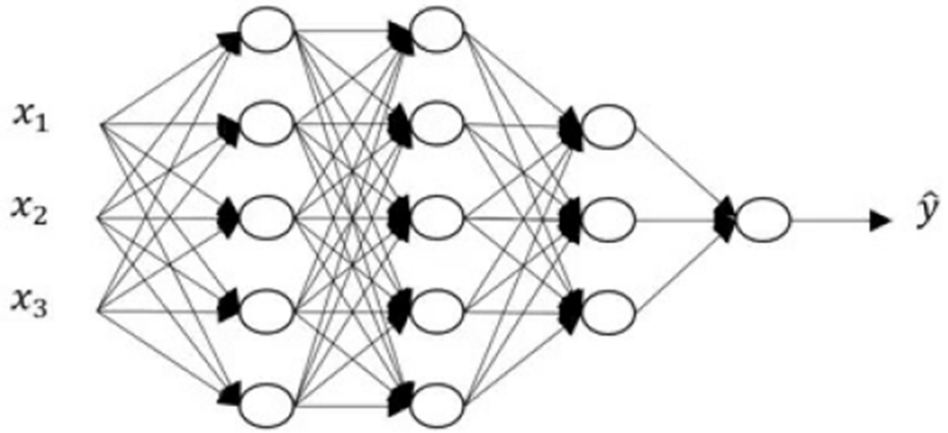
Why deep learning ?

Hardware get smaller.
Sensors get cheaper, widely available IoT devices with high sample-rate.
Data sources: sound, vibration, image, electrical signals, accelerometer, temperature, pressure, LIDAR, etc.

Big Data: Exponential growth of data, (IoT, medical records, biology, engineering, etc.)

How to deals with **unstructured data** (image, voice, text) =>
needs for feature extraction (data mining).

Deep Neural Networks: first extract (automatically) the features, then solve ML tasks (classification, regression, clustering)



Conventional ML vs. Deep Learning

Conventional Machine Learning (ML)

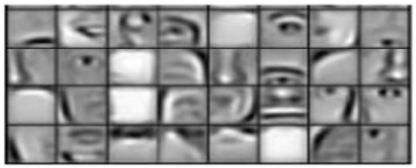
- Needs feature engineering – hard, time-consuming task, requires expert knowledge.
- Don't work well with raw data (e.g. pixels/voxels of images).
- Better than DL if good features are found.

Deep Learning

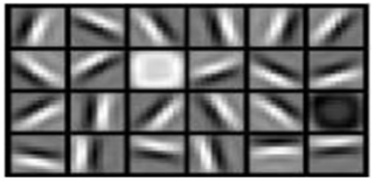
- Representation-learning methods with multiple levels of representation.
- Layers that extract automatically features from raw data (latent structures not seen/difficult to be designed by humans)
- Needs large amounts of labelled training data
- Needs massive computational resources (e.g. GPUs, parallel solvers)

Representation Learning inspired by visual neuroscience

object models

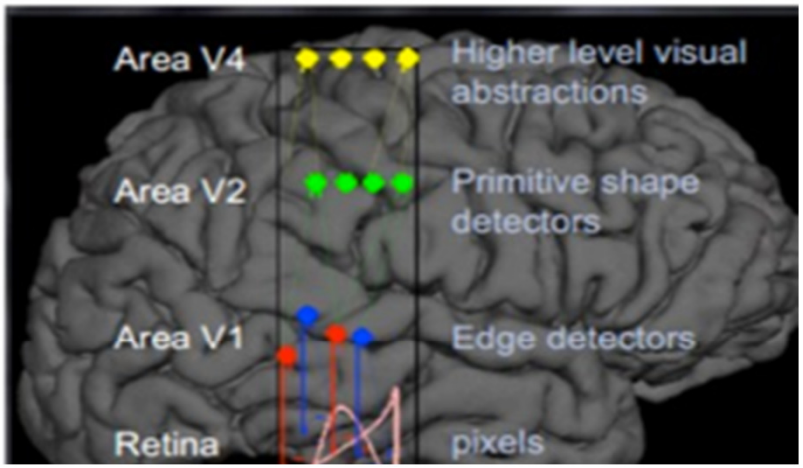


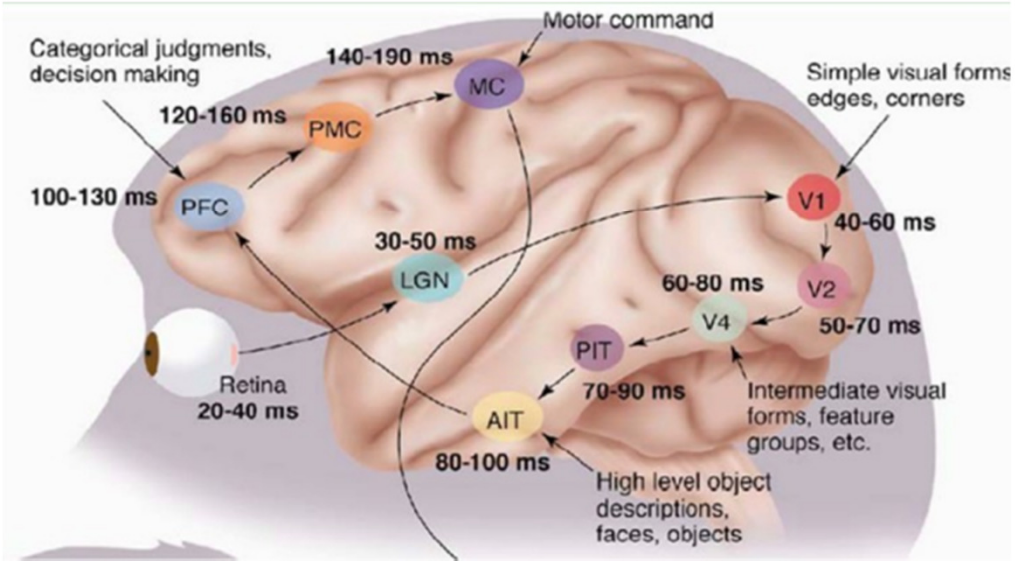
object parts (combination of edges)



edges

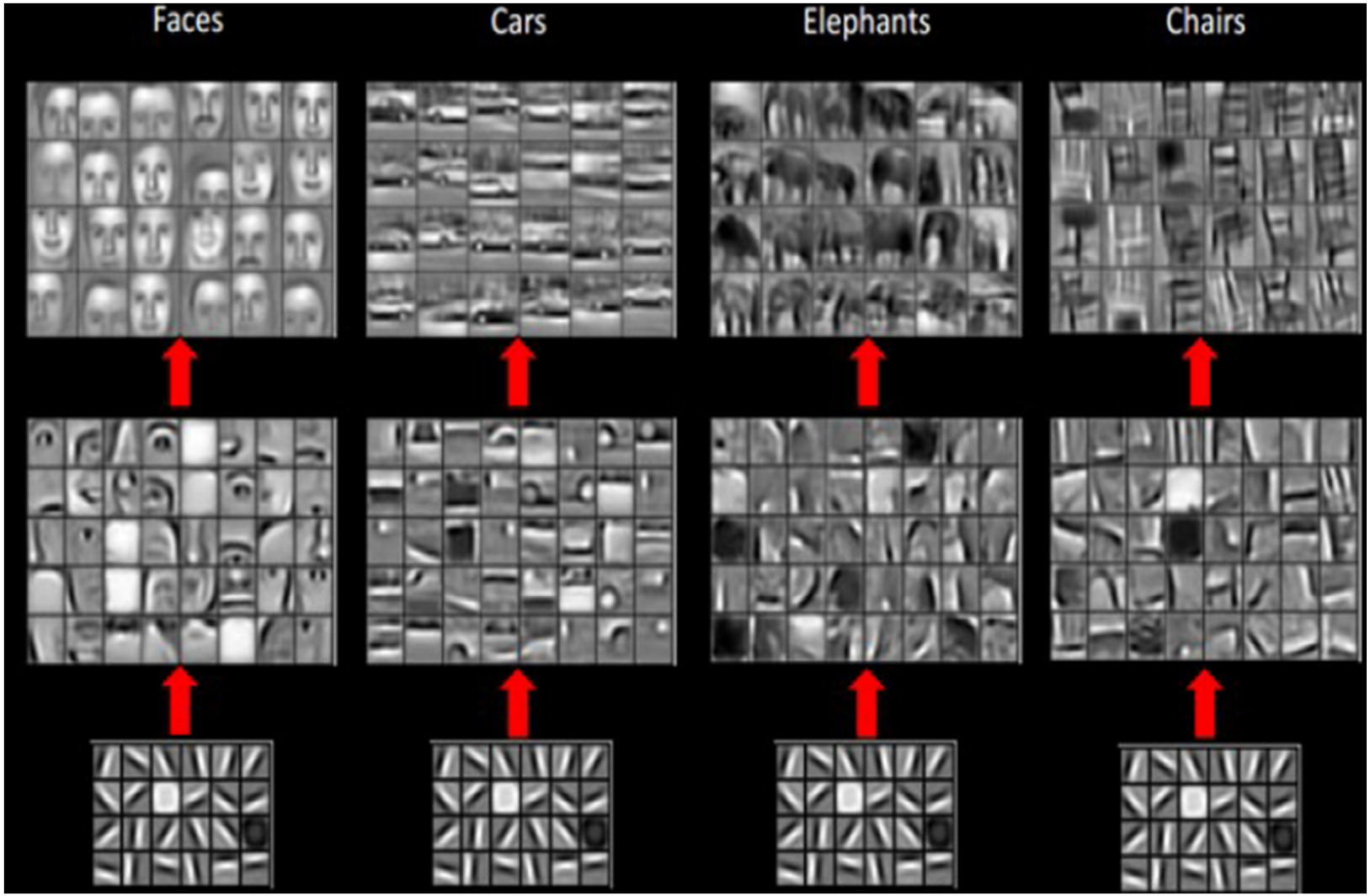
pixels





By Yann LeCun, Yoshua Bengio & Geoffrey Hinton
Nature 521, 436–444, 2015, doi:10.1038/nature14539

Representation Learning



By Yann LeCun, Yoshua Bengio & Geoffrey Hinton
Nature 521, 436–444, 2015, doi:10.1038/nature14539

Convolutional Neural Networks (CNN)

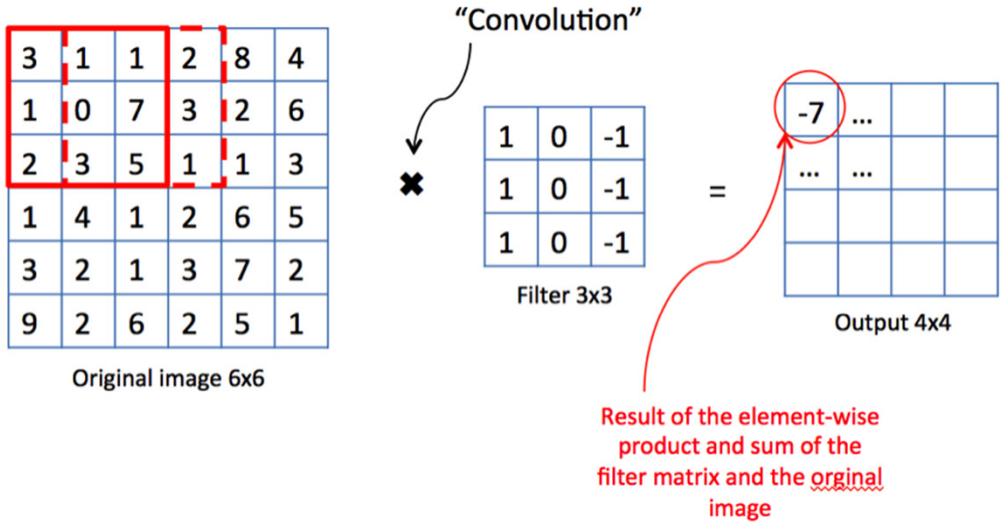
Why Convolution Learning ?

Deep learning on large images

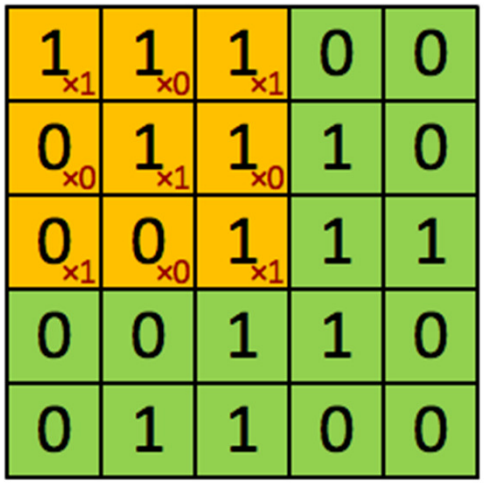
If the image has 1000x1000x3 (RGB) pixels=3 million features (inputs)
If the first hidden layer has 1000 nodes =>
The parameter matrix between the input and the hidden layer has (1000x3million) 3billion parameters.

1st problem: Difficult to get enough data to prevent model overfitting
2nd problem: Computational (memory) requirements to train such networks are not feasible.

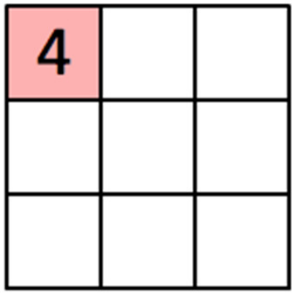
Solution: implement convolution operation



OPERATION CONVOLUTION

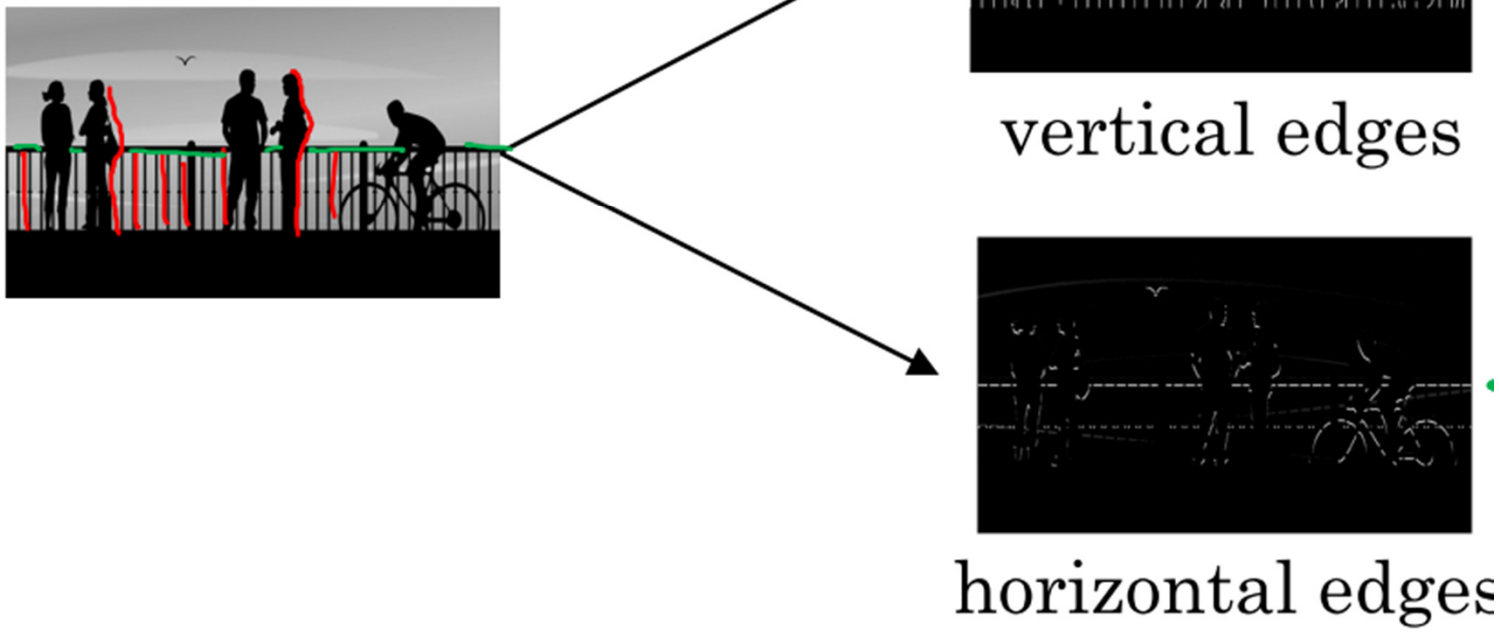


Image



Convolved Feature

Detect horizontal/vertical edges



VERTICAL EDGES DETECTOR

Ilustrative example:

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

*

Detection of bright to dark transition (+30)

Hand-picked convolutional filters(kernels)

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

=> Horizontal edge detector

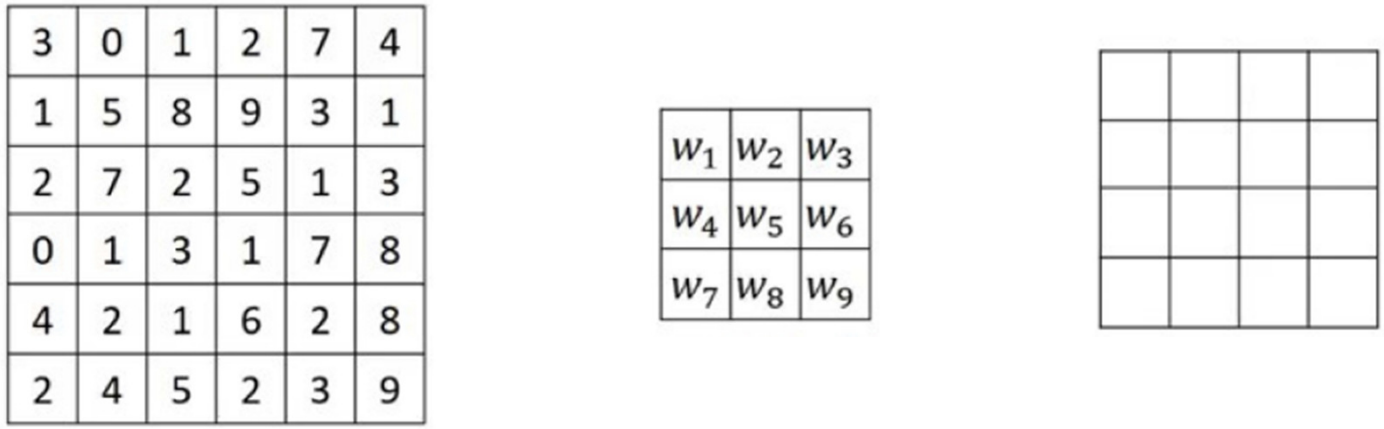
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$\Rightarrow$ **Sobel filter**

$$\begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$$

$\Rightarrow$ **Sharr filter**

CONVOLUTIONAL FILTERS (KERNELS)



Hand-picking the filter values is difficult.
Why not let the computer to learn them ?
Treat the filter numbers as network trainable parameters (weights), and let the computer learn them automatically.
Other than vertical and horizontal edges, such computer-generated filter can learn information from different angles (e.g. 45°, 70°, 73°) .

By convention the conv filter is a square matrix with odd size (typically 3x3; 5x5; 7x7, also 1x1) .
It is nice to have a central pixel and it facilitates the padding.

PADDING

$$\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}$$

Ex. Take 6 × 6 image, apply 3 × 3 conv filter, get 4 × 4 output matrix, because we shift the filter one row down or one column right and therefore we have 4 × 4 possible positions for the 3 × 3 filter to appear in the 6 × 6 input matrix.

In general: given $n \times n$ input matrix and $f \times f$ filter matrix, the convolution operation will compute $(n-f+1) \times (n-f+1)$ output matrix by applying one pixel up/down left/right rule.

1st problem: Shrink the matrix size as we continue to further apply convolution. The image will get very small if we have many convolution layers.
2nd problem: Pixels on the corner of the image are used only once while the pixels in the centre of the image are used many times. This is uneven, loose of inf.

Solution: Padding.

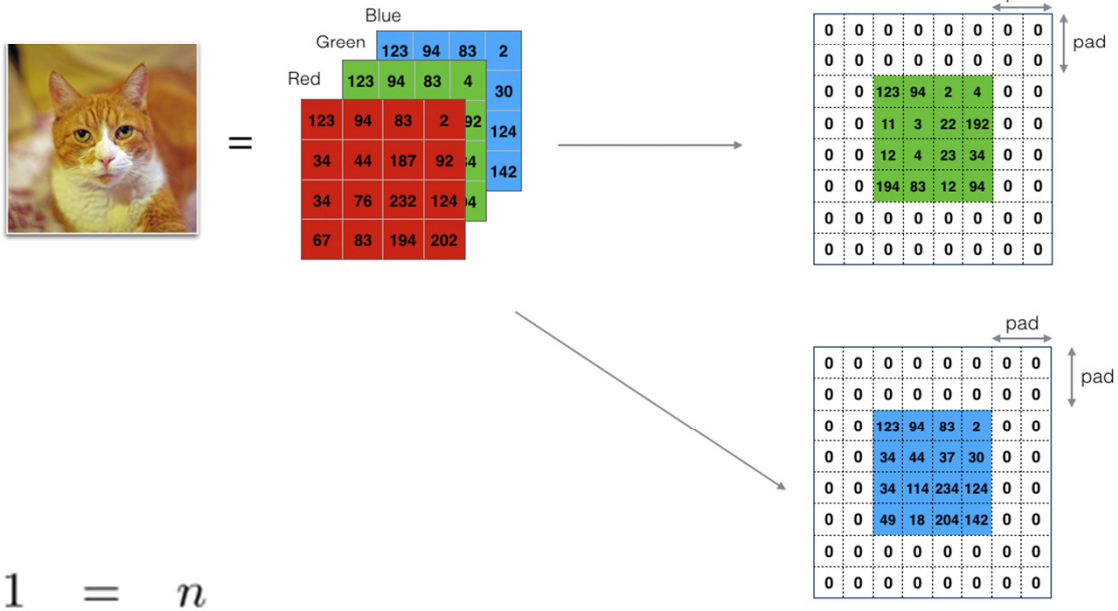
SYMMETRIC PADDING

Add p extra columns and rows at the image borders with 0 values (zero padding). Output matrix size:

$(n+2p-f+1) \times (n+2p-f+1)$

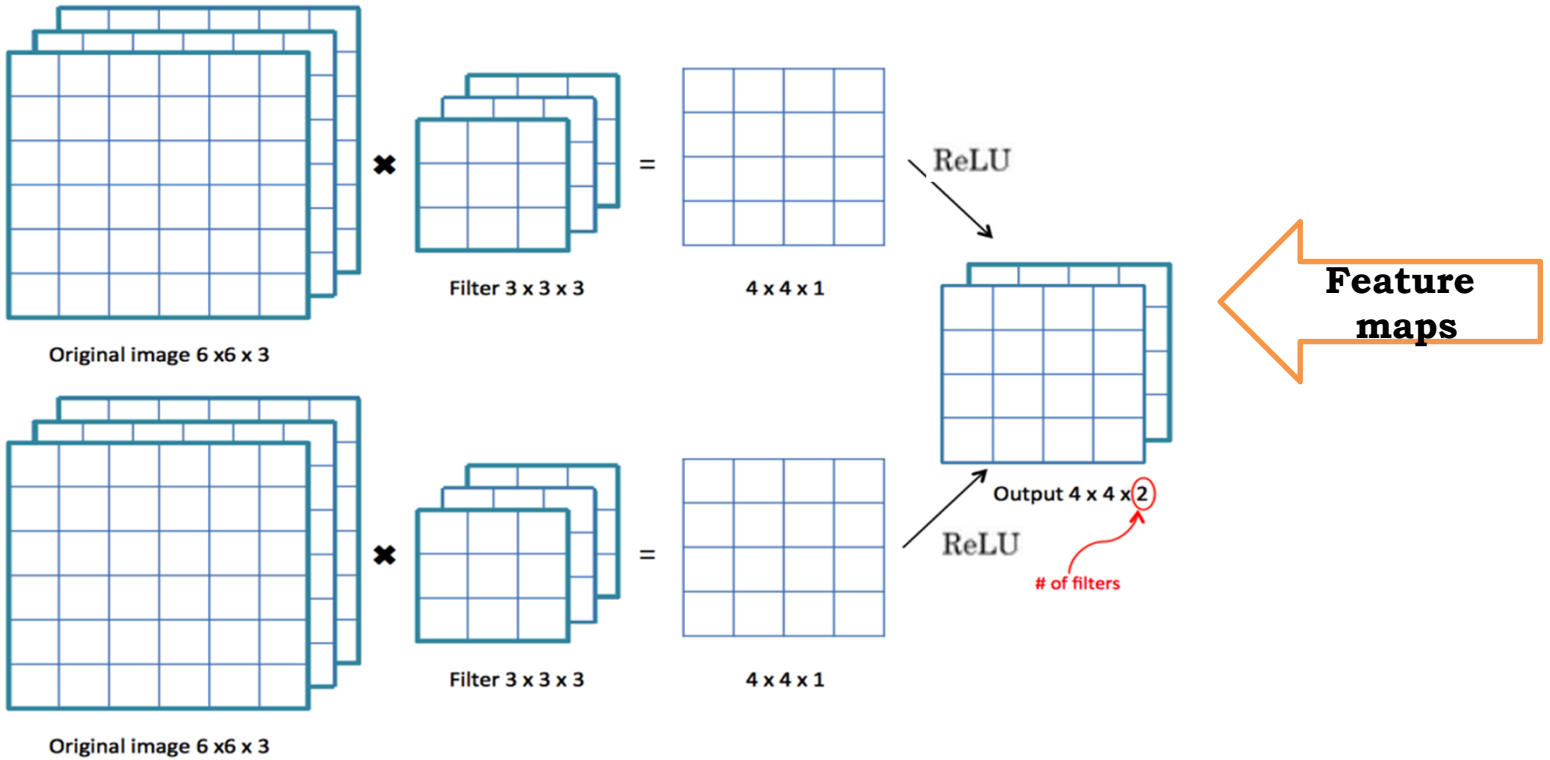
"valid" convolution =>
no padding

"same" covolution =>
Pad so that output size is the same as the input size. Formula for choosing p
(f is usually odd number!):



$$
\begin{aligned}
n + 2p - f + 1 &= n \\
2p - f + 1 &= 0 \\
2p &= f - 1 \\
p &= (f-1)/2
\end{aligned}
$$

ONE CONV LAYER OF CNN



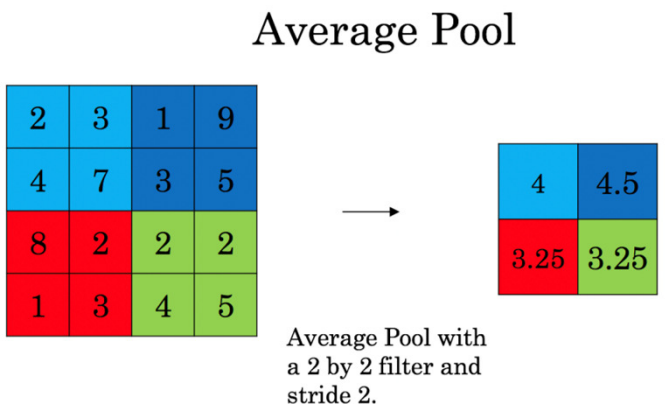
Different 3D filters (kernels) are applied to the 3D input image and the result matrices are stacked to form a 3D output volume.

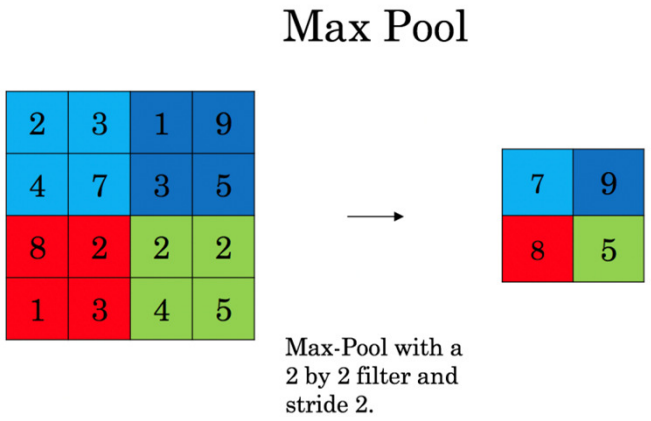
After the convolution operation the result is passed through an activation function (e.g. ReLU, sigmoid, linear, etc.).

The outputs of the conv layers are known as feature maps.

Play conv kiank

POOLING (POOL)





Pooling operation reduces the size of the representation to speed up the computation and make the features more robust.

Ex. Divide the input in regions (e.g. 2 × 2 filter), choose stride (e.g. s=2), each output will be the max (**max pooling**) or the average (**average pooling**) from the corresponding regions.

Some intuition:
Large number means there is some strong feature (edge, eye) detected in this part of the image, which is not present in another part.
Max Pool: whenever this feature is detected it remains preserved in the output.

Softmax Layer

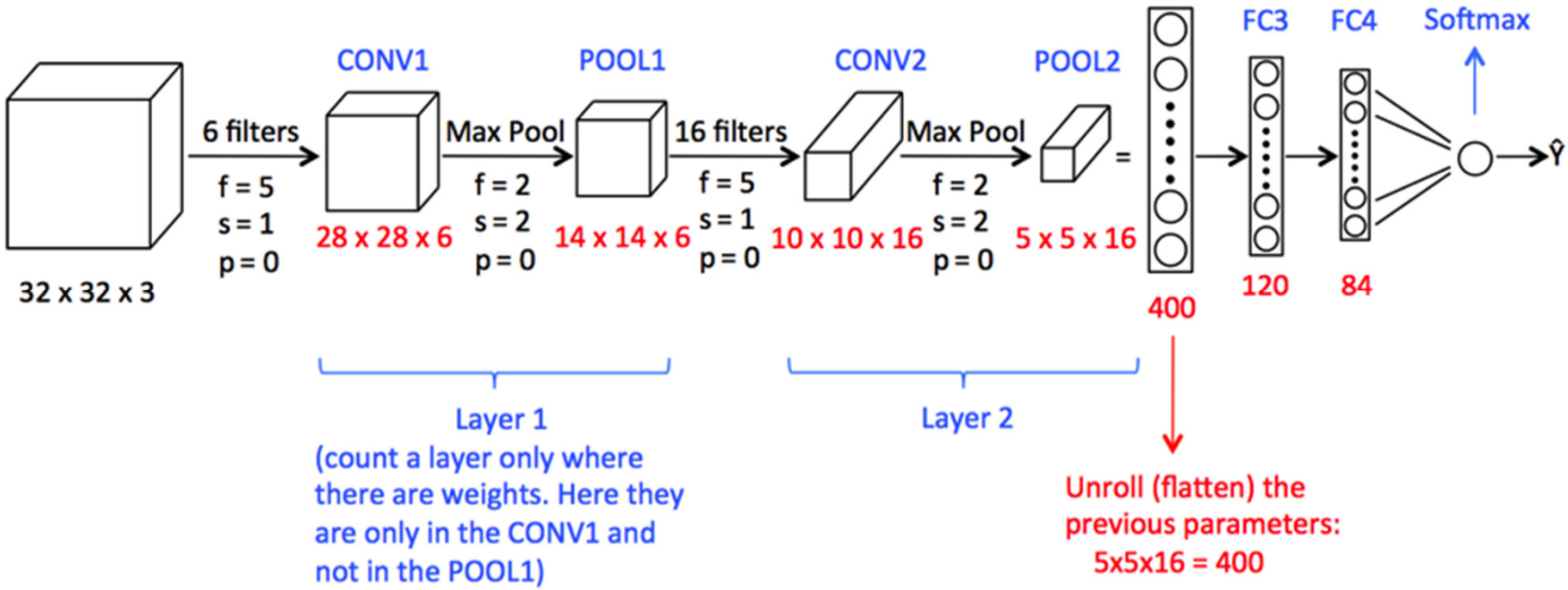
Softmax Layer (SL) estimates the probability that an example $x^{(i)}$ belongs to each of the k classes ((j=1,2,..k).

$$p(y^{(i)} = j|x^{(i)};\theta) = \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}$$

SL outputs *k* dimensional vector with estimated probability for each class k:

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

CNN - LeNet5*

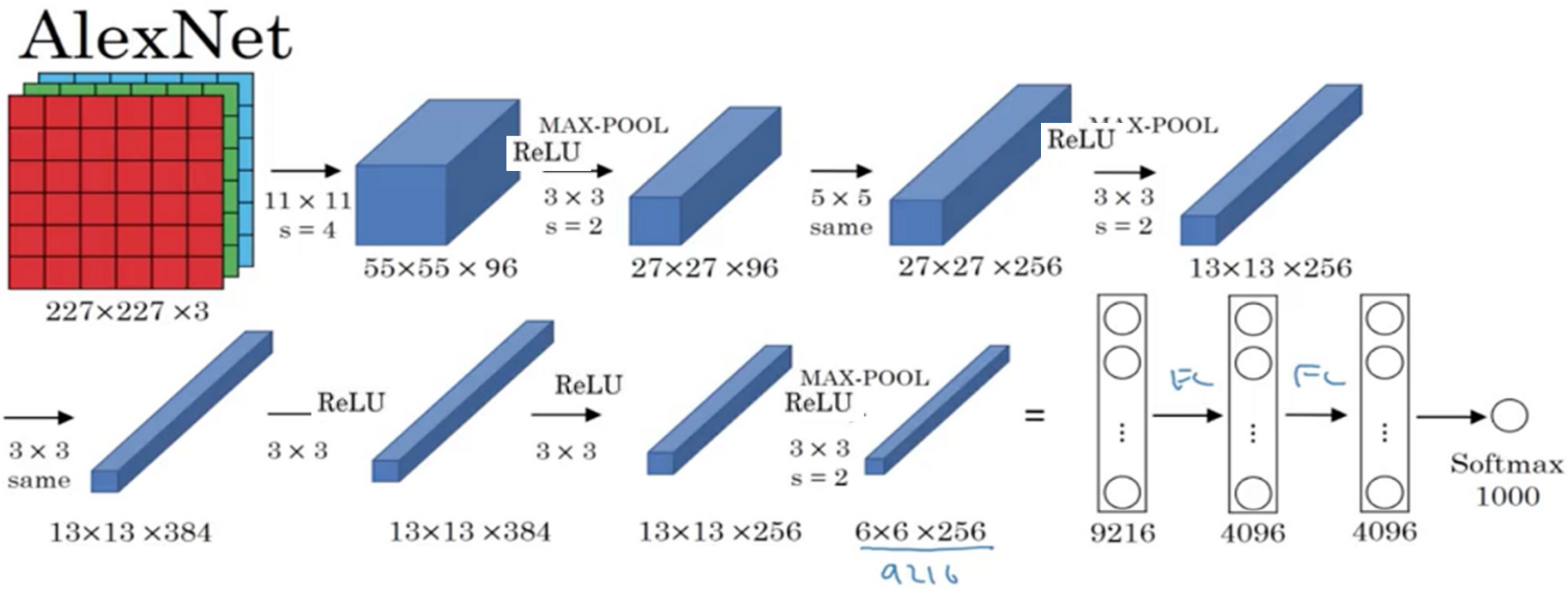


***LeCun et al., 1998, "Gradient-based learning applied to document recognition".** Original LeNet5 applied to handwritten digit recognition (grey scale images). Avg pooling, no padding, softmax classifier; ReLU and sigmoid/tanh neurons in the Fully Connected (FC) layers.

The activation function is always present after the convolution, even if it is not drawn on the CNN diagram.

General trend: CNNs start with large image, then height and width gradually decrease as it goes deeper in the network, where as the number of channels increase.

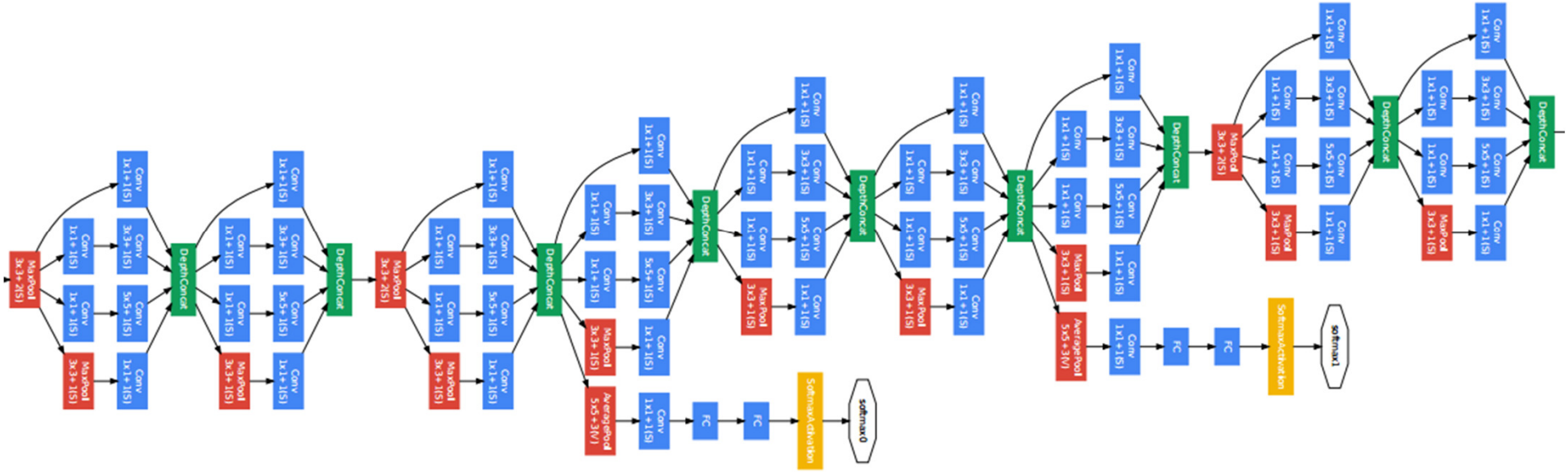
CNN - AlexNet*



*** Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012, ImageNet classification with deep convolutional neural networks.**
5 conv layers, 3 FC layers with softmax output, 60 million parameters in total. AlexNet applied to ImageNet LSVRC-2010 dataset to recognize 1000 different classes.

This paper convinced the Computer Vision (CV) community that DL really works and will have a huge impact not only in CV but also in speech/language processing.

INCEPTION NETWORK (GoogLeNet)



Ref. Szegedy et al 2014, “Going deeper with Convolutions”. Arround 25 mln. Of parameters

You Only Look Once
(YOLO)
CNN Architecture

Image classification/localization/detection

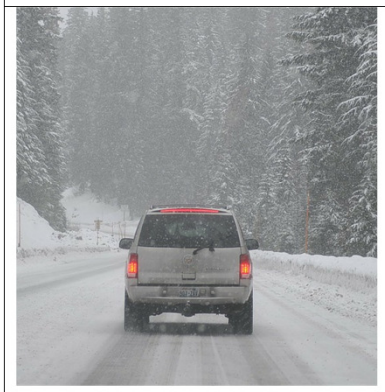
Image classification	Classification & Localization	Detection
	 b_x, b_y, b_h, b_w	
"car"	"car"	multiple objects

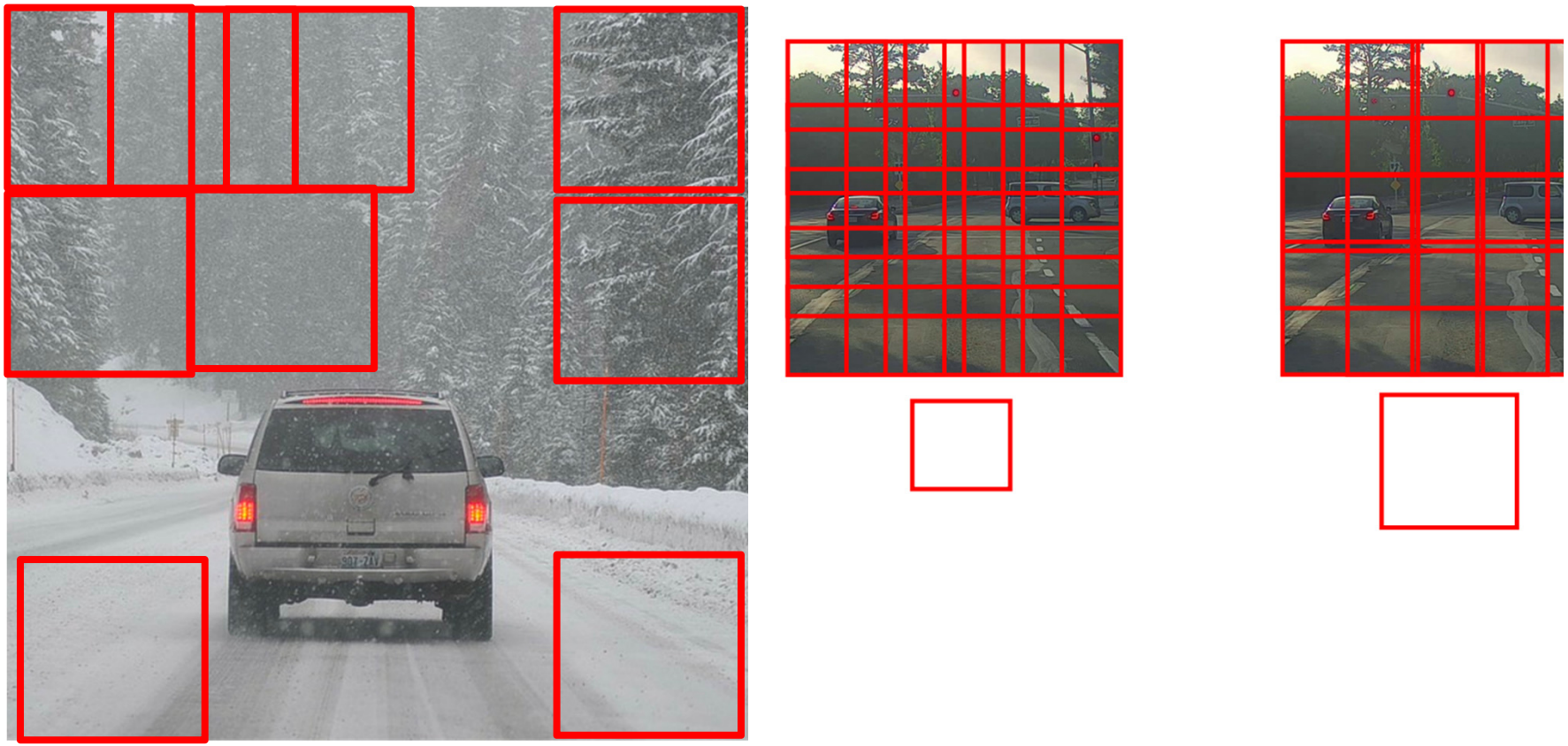
Image classification: input a picture to CNN and the output is a class label (e.g. person, bike, car, background, etc.)

Classification with localization: the algorithm gives not only the class label of the object but also draws a bounding box (the coordinates) of its position in the image.

Standard notation: (0,0) as the upper-left corner and (1,1) to be the lower-right corner.

(b_x , b_y , b_h , b_w) describes the bounding box.

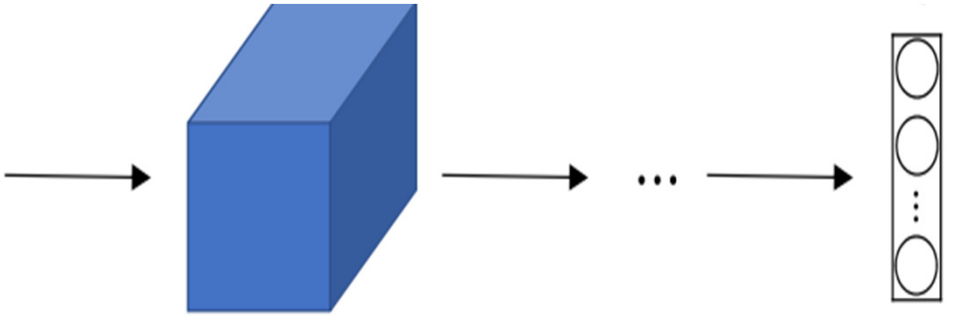
Sliding windows detection algorithm



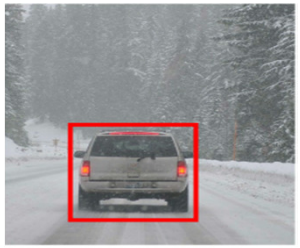
Pick a certain window size, input this sub-picture into already trained CNN to detect objects. Then shift the detection window to the right with one step (stride) and feed the new sub-picture into ConvNet. Go through every region (the stride needs to be small for the detection algorithm to run smoothly). Repeat the same with different sizes of the detection window in order to detect objects with different sizes of the picture.

The Sliding Windows object detection algorithm has **infeasibly high computationally cost.**

Object classification with localization



Classes (e.g.):
1. person
2. Bikes
3. Car
4. Background (no object)



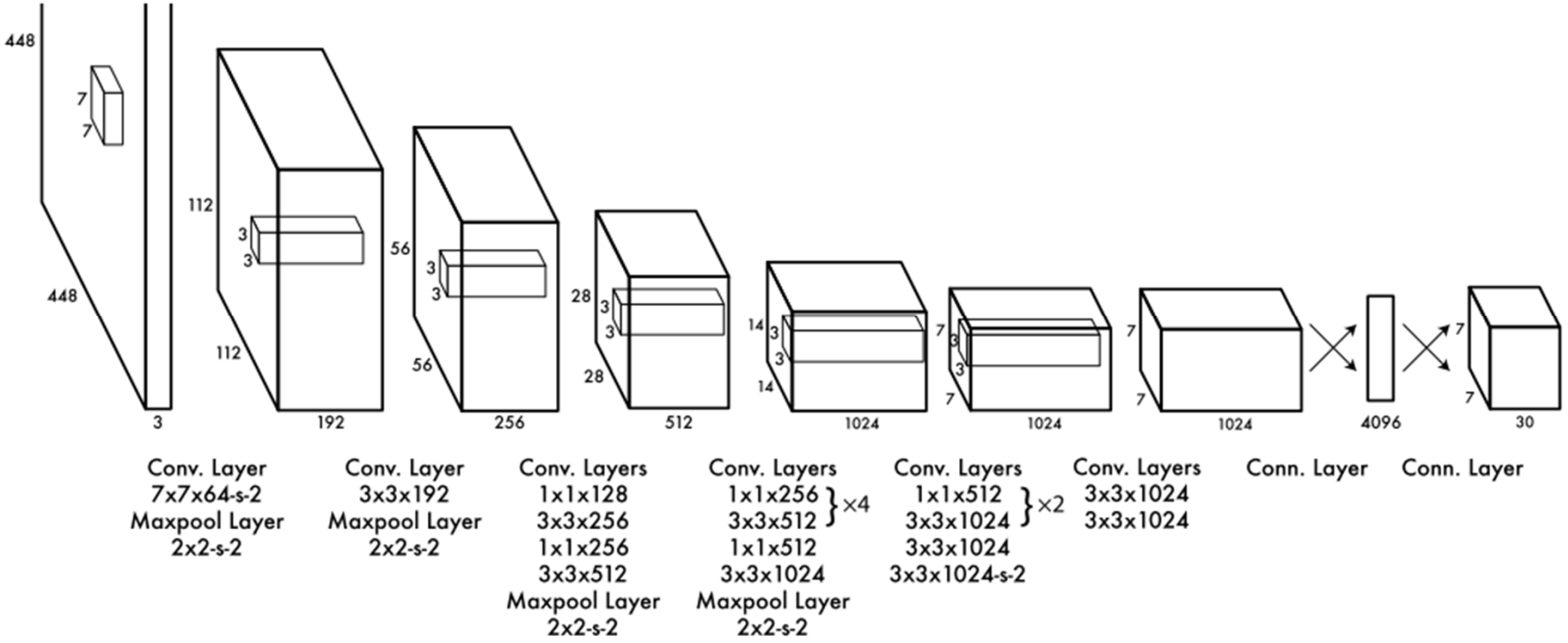
b_x, b_y, b_h, b_w

Output: (p_c , b_x , b_y , b_h , b_w , c= [c_1, c_2….c_{end}])
p_c – is there an object or not (1/0)

<= Image label: [1 , b_x , b_y , b_h , b_w , 0, 0, 1]

<= Image label: [0 , ? , ? , ? , ? . ?, ?; ?]
? – “don’t care”

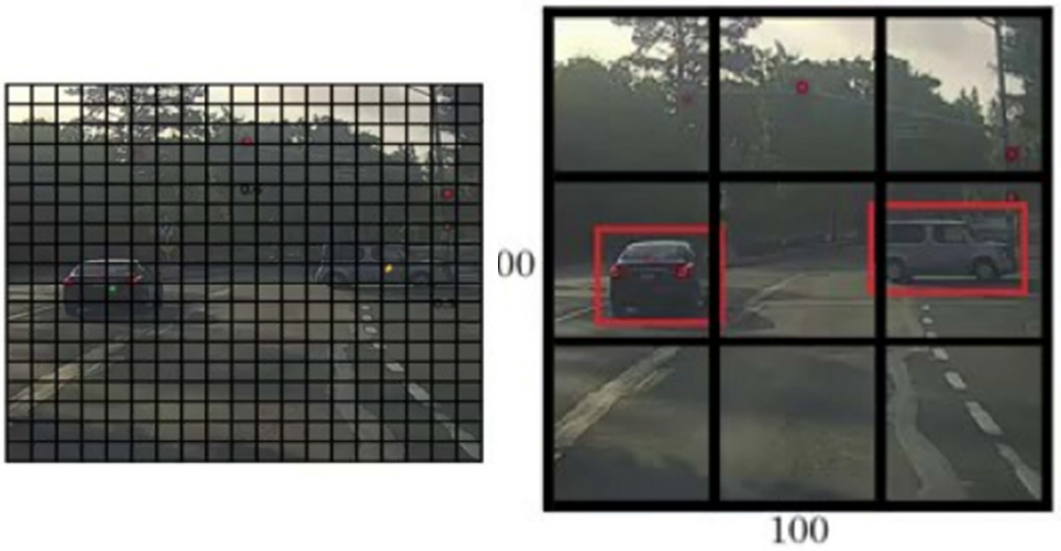
YOLO (You Only Look Once)



YOLO - CNN network for both classification and localising the object using bounding boxes.
24 convolutional layers + 2 fully connected layers.
Conv layers pretrained on ImageNet dataset.

*Redmon et al, 2015, "You Only Look Once: Unified, Real-Time Object Detection" (https://arxiv.org/abs/1506.02640)
Redmon & Farhadi, 2016 (https://arxiv.org/abs/1612.08242).

YOLO (You Only Look Once) algorithm



<= cell label: [0 , ? , ? , ? , ? . ?, ?; ?]
? – "don't care"

<= cell label: [1 , b_x , b_y , b_h , b_w , 0, 0, 1]

<= cell label: [0 , ? , ? , ? , ? . ?, ?; ?]
? – "don't care"

Let we have 100x100 pixels input image. We place a grid on this image.
In the original paper the grid is 19x19, here for illustration is 3x3 grid (9 cells).
Apply object classification and localization to each cell.

How to define the labels used for training:
For each cell, the label is 8 dimensional vector (if we have 3 classes)
y = [p_c, b_x , b_y , b_h , b_w , c1, c2, c3], where

- p_c (0 or 1) specifies if there is or not an object in that cell.
- [b_x , b_y , b_h , b_w] specify the bounding box if there is an object in that cell.
- c1; c2; c3 (0 or 1) - to assign the class (e.g. person, bicycle, car)

YOLO assigns the object only to the cell containing the midpoint of the object.
Since we have 3x3 grid, the total volume of the target output Y is 3x3x8.

Sequence Models - Long-Short Term Memory (LSTM)

Examples of sequence data

	x (input)	y (output)
Speech recognition		==> "The quick brown fox jumped over the lazy dog."
Sentiment classification	"There is nothing to like in this movie."	==> 
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	==> AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	==> Do you want to sing with me?
Video activity recognition		==> Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	==> Yesterday, Harry Potter met Hermione Granger.

x and y are both sequences, or only x is a sequence, or only y is a sequence.

Sequence Model Notation

Name Entity Recognition application is to find people's names, companies names, locations, countries names, currency names, etc. in text.

Given an input sequence of words (X), the sequence model has to automatically tell where are the proper names in this sentence.

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ... $x^{<9>}$

For this exampe the sequence length is 9 words (features) **=> Tx=9**

Target output y is binary vector with same length as the input
(1 if the word is name; 0 if not)

y= [1 1 0 1 1 0 0 0 0]

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ ..... $y^{<t>}$ $y^{<Ty>}$

Ty=9

In this problem every input $x^{<i>}$ has an output $y^{<i>}$ => **length Ty=Tx**
Each sentence (example) can have different sequence length.

NLP - representing individual words

Natural Language Processing (NLP). Create vocabulary or download existing dictionary.
For modern NLP, 10000 words is a small dictionary, 30-50 thousand is more common.
Large internet companies use 1 million words dictionary.
Each word is represented by a binary vector with # elements = dimension of dictionary where only the index of the word in the dictionary =1, all others are 0 (one-hot vector).

word index (in dictionary) **one-hot encoding**

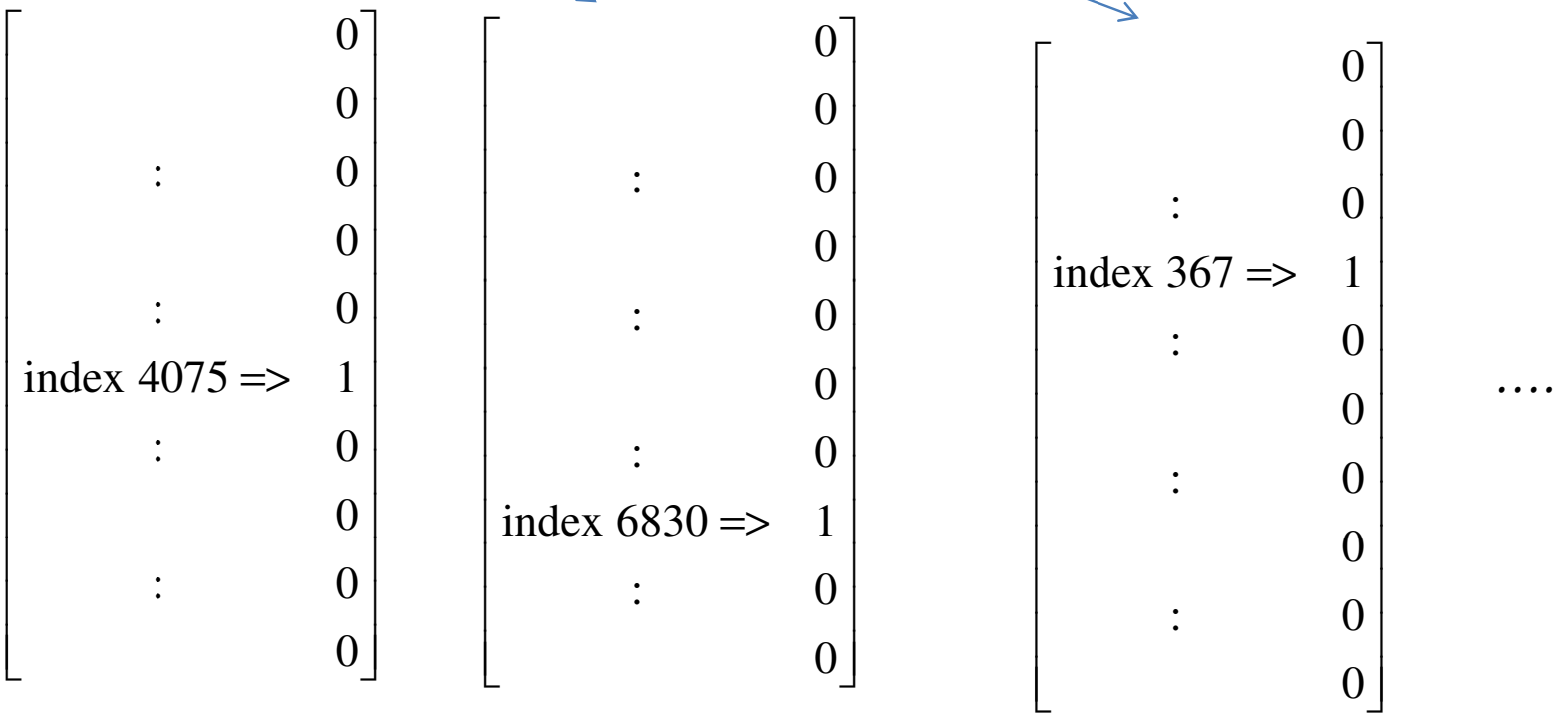
$$\begin{bmatrix} a => & 1 \\ aaron => & 2 \\ \vdots & \\ and => & 367 \\ \vdots & \\ Harry => & 4075 \\ \vdots & \\ Potter => & 6830 \\ \vdots & \\ zulu => & 10000 \end{bmatrix}$$

$$Harry = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

NLP – one hot encoding

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … $x^{<9>}$



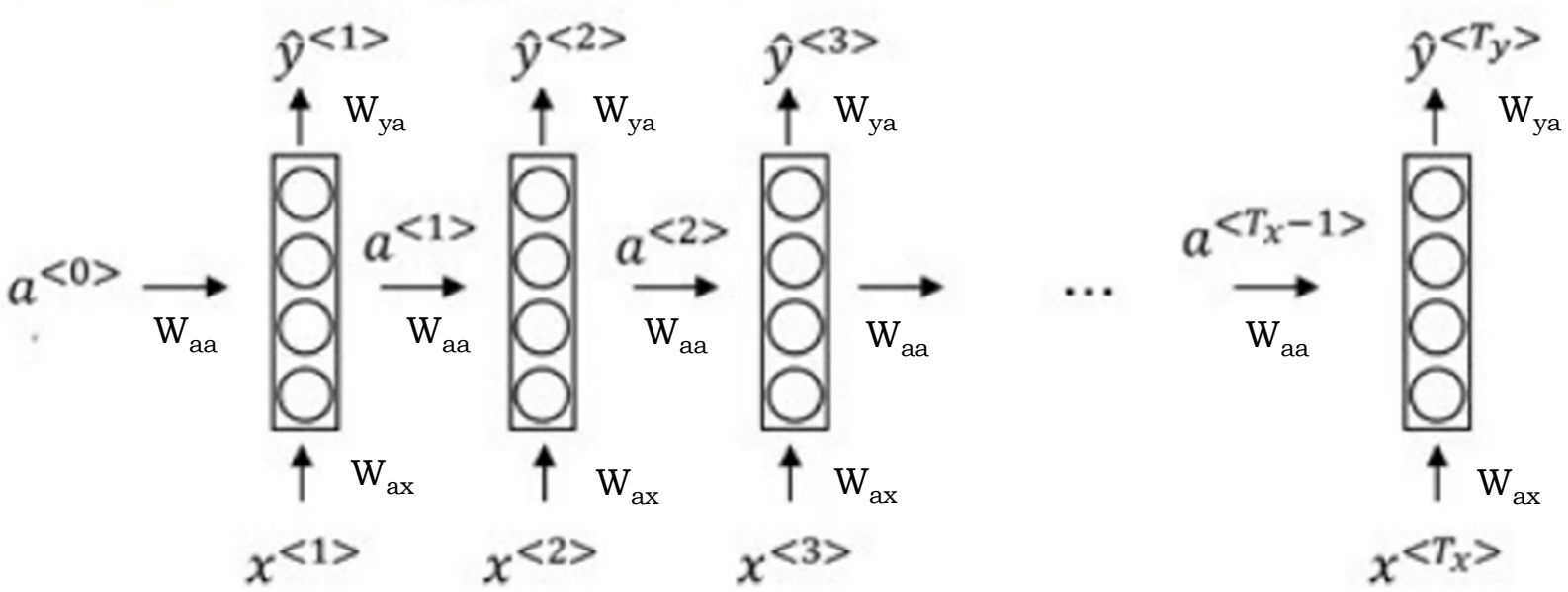
<unk> - notation for words not in the dictionary. It can be added to encode all missing words that may appear in sentences.

The problem is formulated as supervised learning with labelled data (x,y).

Recurrent Neural Networks (RNN)

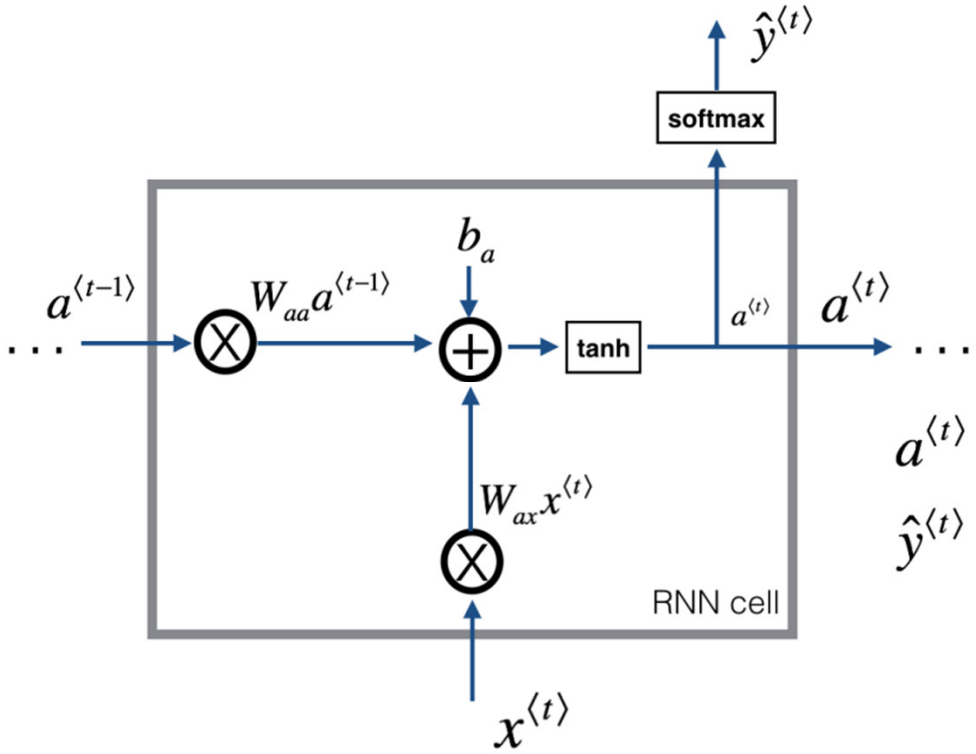
Read the sentence word by word (one time step per word).
Activation produced by 1st word is taken into account when read 2nd word.
Notation: inputs -$x^{<t>}$; outputs - $y^{<t>}$; hidden states - $a^{<t>}$
$a^{<0>}$ - initial state at time step 0, usually vector of zeros.
Previous results are passed as inputs => we get context !

Fig. Flow chart of RNN representation



Basic RNN unit

Takes $x^{<t>}$ (current input) and $a^{\langle t-1 \rangle}$ (activation from previous step) as inputs, and computes $a^{\langle t \rangle}$ (current activation). $a^{\langle t \rangle}$ is then used to predict $y^{\langle t \rangle}$ (current output) and passed forward to the next RNN unit (next time step). W_{aa}, W_{ax}, W_{ya}, b_a, b_y – the same RNN weights used in all time steps.

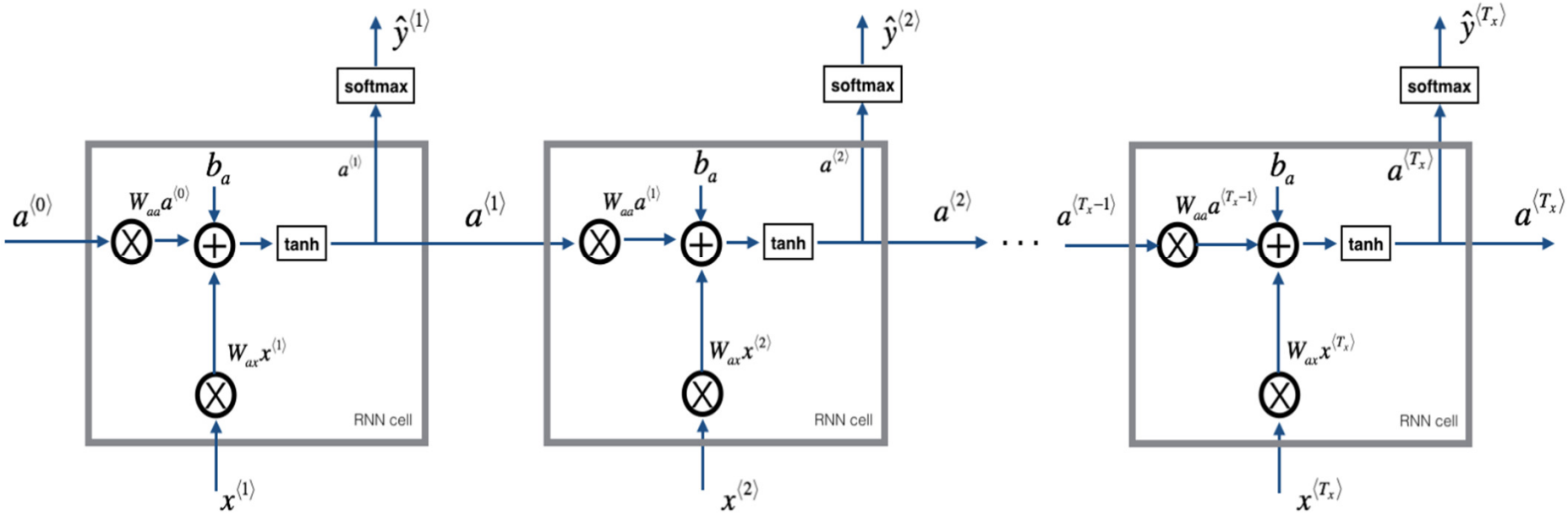


$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

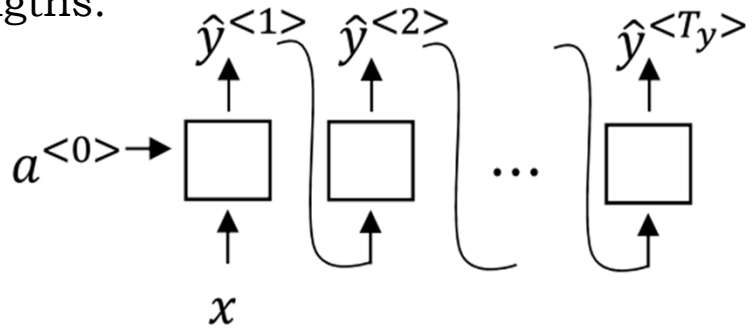
RNN forward pass

RNN is a sequence of basic RNN cells.
If input sequence is x=[$x^{<1>}$, $x^{<2>}$,...$x^{<Tx>}$] => RNN cell is copied T_x times.
RNN outputs y=[$y^{<1>}$, $y^{<2>}$,...$y^{<Tx>}$]

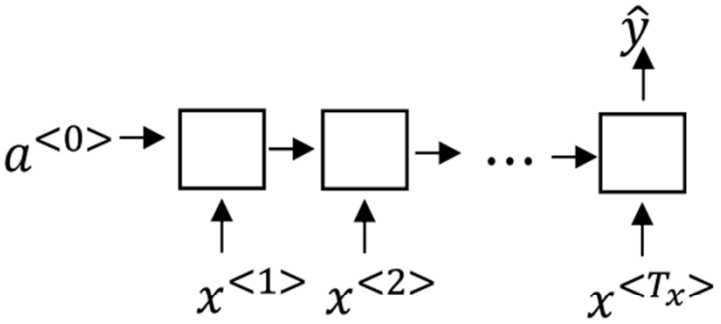


Different Types of RNN

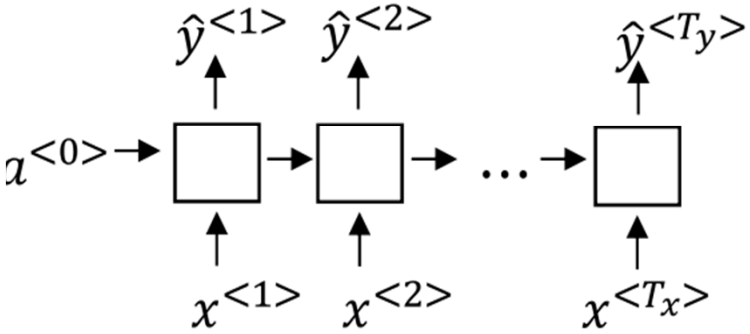
The input X and Y can be of many types and they do not have to be of the same lengths.



One to many (e.g. Music generation)

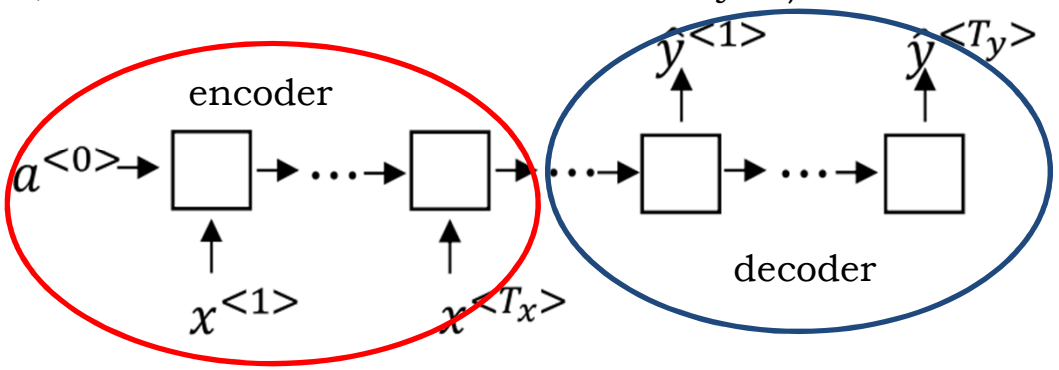


Many to one (e.g. Sentiment analysis)



Many to many

Tx=Ty (e.g. Name entity rec.)

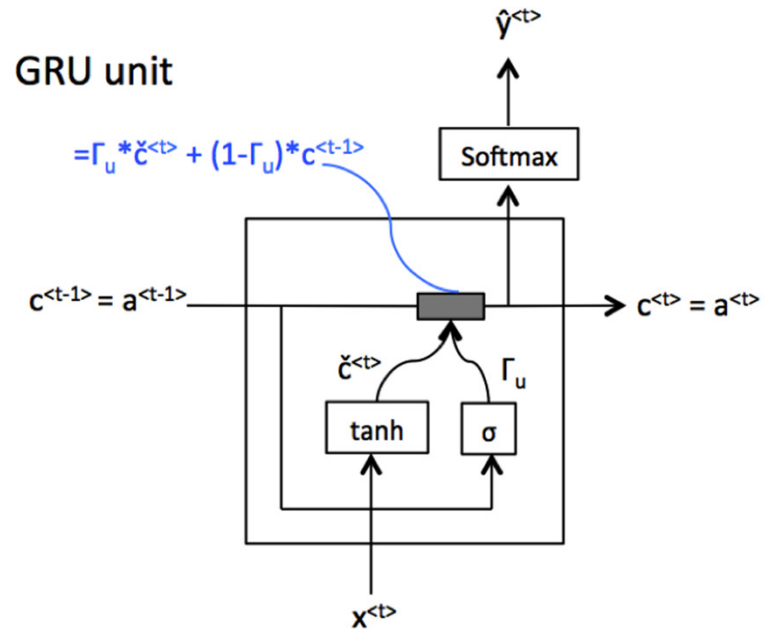


Many to many

Tx different from Ty (e.g. machine translation)

Note: Time series forecasting (many to one, or many to many)

Gated Recurrent Unit (GRU) vs basic RNN unit



c - memory cell

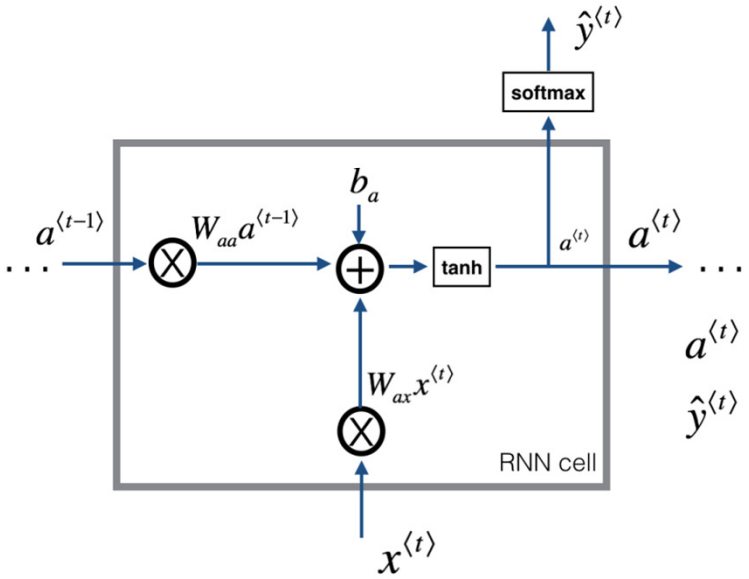
$c^{<t>} = a^{<t>}$ (for LSTM they are different)

GRU outputs activation value = memory cell

$\tilde{c}^{<t>} = \tanh\left(w_c\left[c^{<t-1>}, x^{<t>}\right] + b_c\right)$ – candidate

$\Gamma_u = \sigma\left(w_u\left[c^{<t-1>}, x^{<t>}\right] + b_u\right)$ – update gate

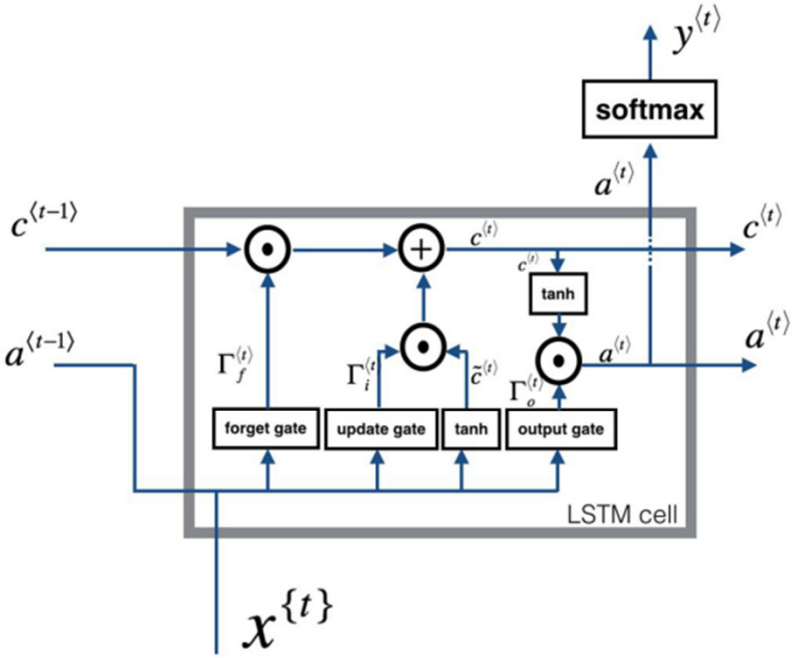
$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$ – update memory cell



$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$

$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$

Long Short-Term Memory (LSTM) unit vs RNN



$$\tilde{c}^{<t>} = \tanh\left(W_c\left[a^{<t-1>}, x^{<t>}\right] + b_c\right) - \text{candidate}$$

$$\Gamma_u = \sigma\left(W_u\left[a^{<t-1>}, x^{<t>}\right] + b_u\right) - \text{update gate}$$

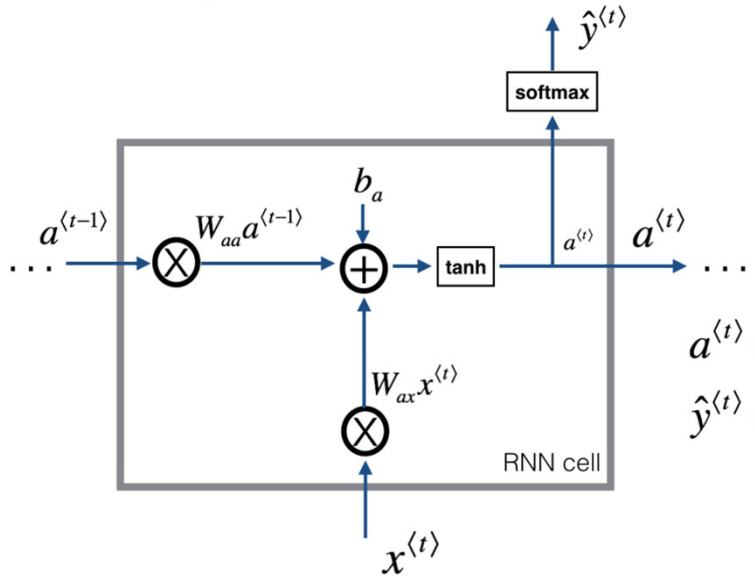
$$\Gamma_f = \sigma\left(W_f\left[a^{<t-1>}, x^{<t>}\right] + b_f\right) - \text{forget gate}$$

$$\Gamma_o = \sigma\left(W_o\left[a^{<t-1>}, x^{<t>}\right] + b_o\right) - \text{output gate}$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} - \text{update memory cell}$$

$$a^{<t>} = \Gamma_o * \tanh\left(c^{<t>}\right) - \text{activation}$$

LSTM outputs both activation value and memory cell



$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

GRU & LSTM

LSTM and GRU are two variations of RNNs to capture better long range dependencies (connections) in sequences.

They mitigate short-term memory using mechanisms called memory cell and gates.

Gates remember (keep saved) bits of inf for many time steps.

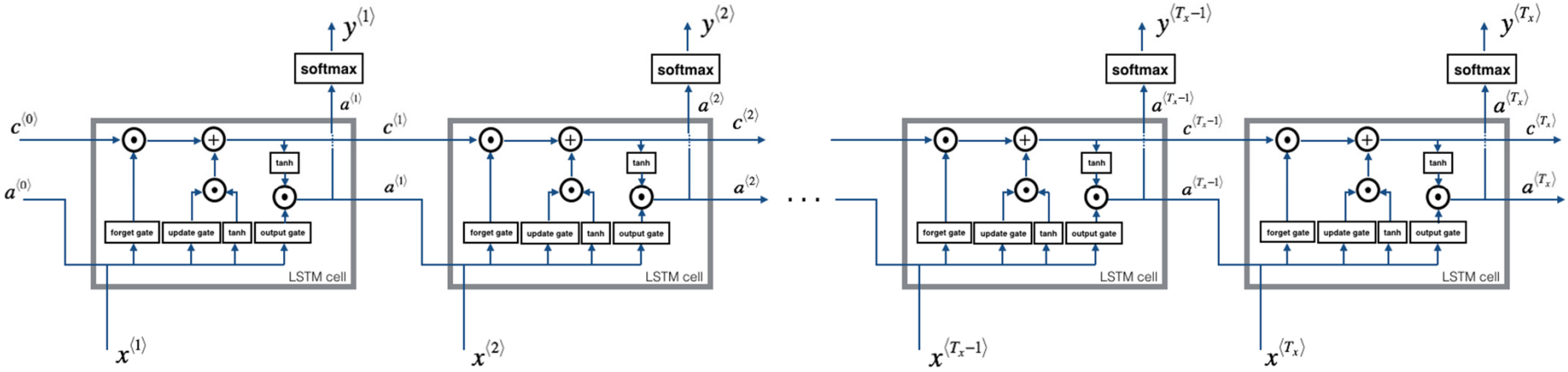
Ex.: Read text, and use LSTM/GRU to keep track of grammatical structures => if the subject is singular or plural. If the subject changes from a singular word to a plural word, forget the previously stored memory value of the singular state.

LSTM's and GRU's are successfully applied in speech recognition, speech synthesis, natural language understanding, time series forecasting, etc.

When to use LSTM or GRU ? - there is not a consensus.
LSTM appeared first (1997), GRU (2014) is a simplification of LSTM.
LSTM is more powerful (3 gates in LSTM, 1 gate in GPU).

** ref. Hochreiter and Schmidhuber 1997, "Long short-term Memory".*

LSTM network

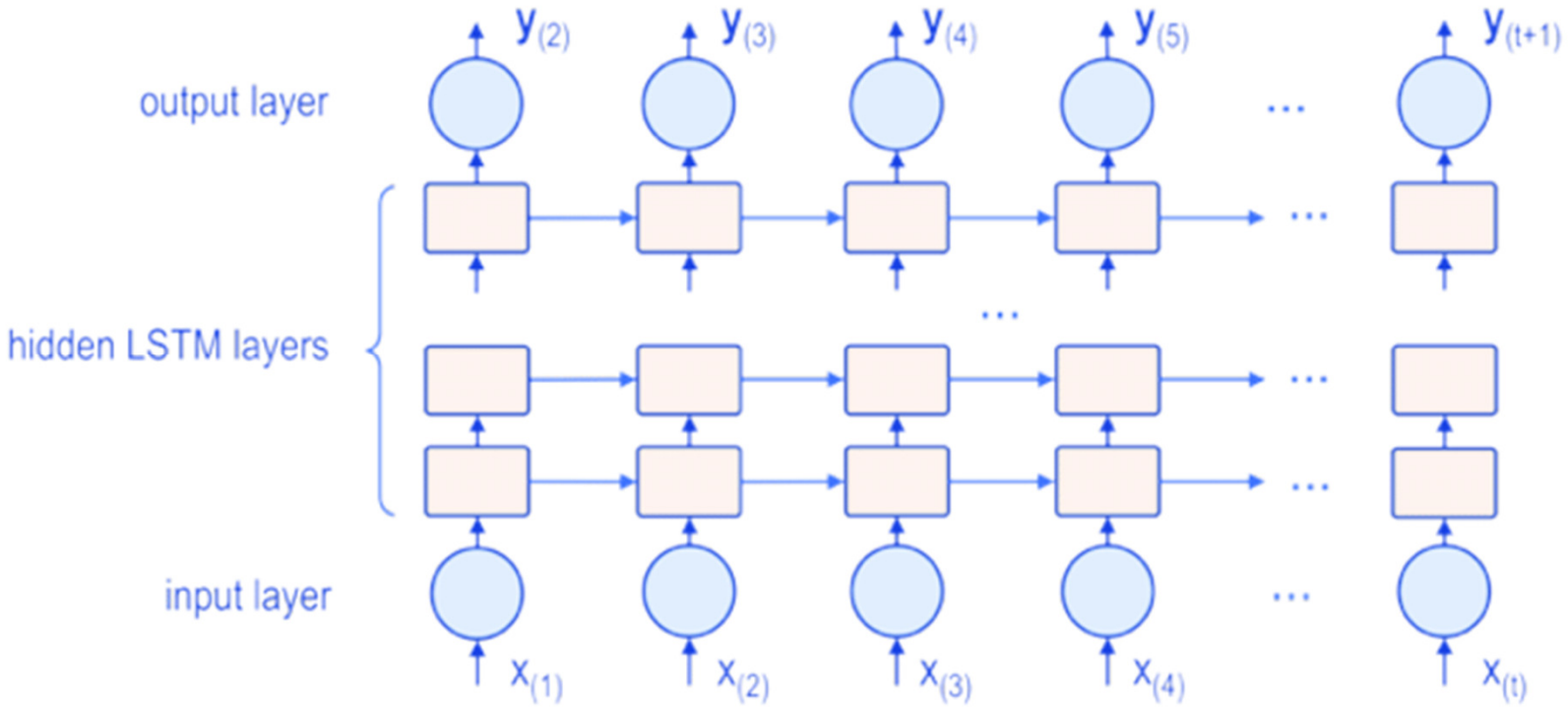


LSTM network is a temporal sequence of LSTM units.
There is a line at the top that shows how LSTM can memorise and pass certain values $c^{<t>}$ through several temporal steps.

Other versions : the gate's values depend also on $c^{<t-1>}$. (peephole connection)

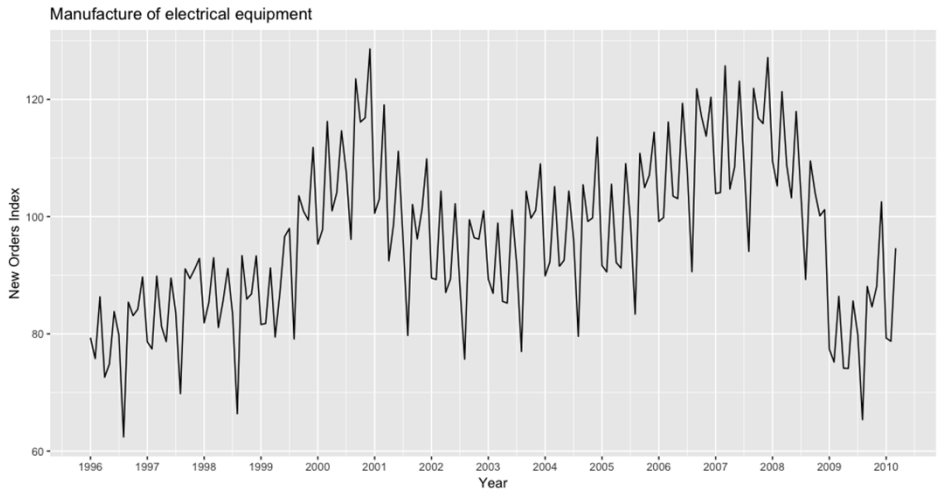
The gates are vectors (e.g. if 100 dimensional hidden memory cell units, the gates will have 100 elements).

Deep LSTM


output layer
hidden LSTM layers
input layer
y(2)
y(3)
y(4)
y(5)
y(t+1)
x(1)
x(2)
x(3)
x(4)
x(t)

Time Series Data

Sequence Models for Time Series Data



Time Series (TS) - collection of data points indexed based on the time they were collected => TS models are naturally sequence models.
Most often, data is recorded at regular time intervals.

TS forecasting (prediction) is a hot topic with many applications in practice:

- Stock prices forecasting / Weather forecasting
- Bitcoin price forecasting
- Biological sensors (ECG, EEG) – predict health problems
- Network traffic prediction

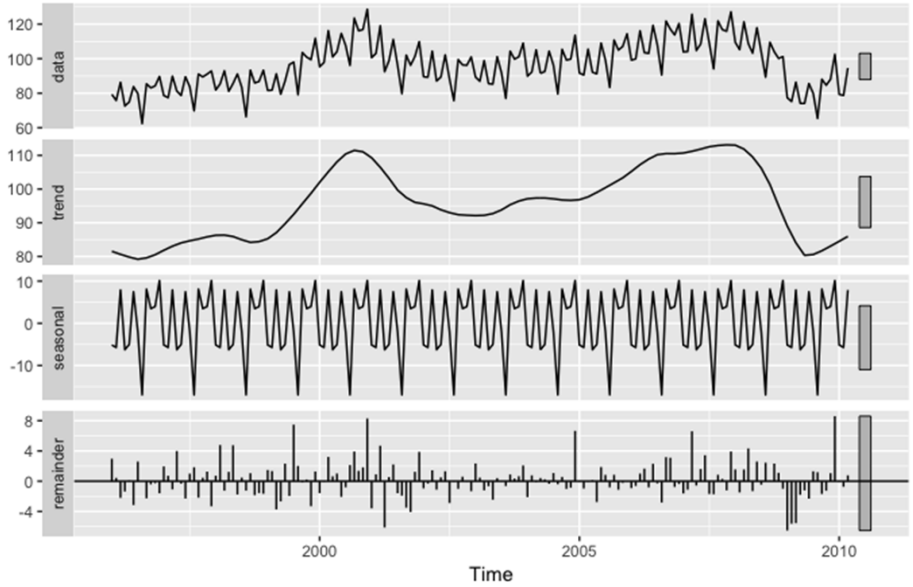
Time Series Decomposition

Y(t) =>

T(t) =>

S(t) =>

R(t) =>



If TS shows some seasonality (e.g. daily, weekly, yearly) it may be useful to decompose the original TS *Y(t)* into sum of 3 components:

Y(t) = S(t) + T(t) + R(t)

T(t) - trend component, estimate T(t) through the mean of the last *n* samples (rolling mean)

S(t) - seasonal component : **S(t) = Y(t)-T(t)**

R(t) - remaining component: **R(t)=Y(t)-T(t)-S(t)**

Trend component in TS

Trend is a long-term increase or decrease in the level of the time series (TS). Systematic change in TS that does not appear to be periodic is known as a trend.

Identify a Trend => Plot TS data to see if a trend is obvious or not.

Remove a Trend => TS with a trend is called non-stationary. Identified trend can be removed (TS de-trending).
If TS does not have a trend or the trend is successfully removed, the dataset is said to be trend stationary.
To get stationary TS, subtract the trend => S(t) = Y(t)-T(t)
To get the trend, subtract the seasonality => T(t)=Y(t)-S(t)
Y(t) – original time series

Before forecasting, it is helpful to remove both the trend and seasonality.

Time Series Forecasting – classical methods

1) Exponential smoothing

The forecasts are equal to a weighted average of past observations and the corresponding weights decrease exponentially as we go back in time.

$\hat{Y}(t+h|t) = \alpha Y(t) + \alpha(1-\alpha)Y(t-1) + \alpha(1-\alpha)^2 Y(t-2) + \ldots$, $0<\alpha<1$ (basic model)

2) ARIMA (popular method for TS forecasting)

AutoRegressive model - linear combination of past values of the TS.
Moving Average model - linear combination of past forecasting errors.

ARIMA (AutoRegressive Integrated Moving Average) models combine the two models.
TS has to be stationary => instead of the original values use differences : **$\hat{Y}(t) = Y(t)-Y(t-T)$**

3) SARIMA model (Seasonal ARIMA) extends the ARIMA by adding a linear combination of seasonal past values and/or forecast errors.

Time Series Forecasting – with LSTM/GRU

Forecasting with classical methods (ARIMA, etc.) => work better with features (e.g. extract trend, seasonality, etc.) than with raw TS data.

Forecasting with Sequence models (LSTM, GRU, etc.) => no need to extract features, work with raw data.

Cut TS into smaller sequences, and use Sequence models to forecast it.

LSTM/GRU inputs: Y(t) , Y(t-1) ….Y(t-p)
LSTM /GRU output (predict the next value): Y(t+1) , Y(t+2) ,… Y(t+k)

Further Reading

- **Ian Goodfellow, and Yoshua Bengio, Deep Learning, MIT Press, 2016**

- **Andrew Ng, Machine Learning Yearning, 2018 (https://www.deeplearning.ai/machine-learning-yearning/)**

- Understanding LSTM Networks – colah's blog : http://colah.github.io/posts/2015-08-Understanding-LSTMs/