UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

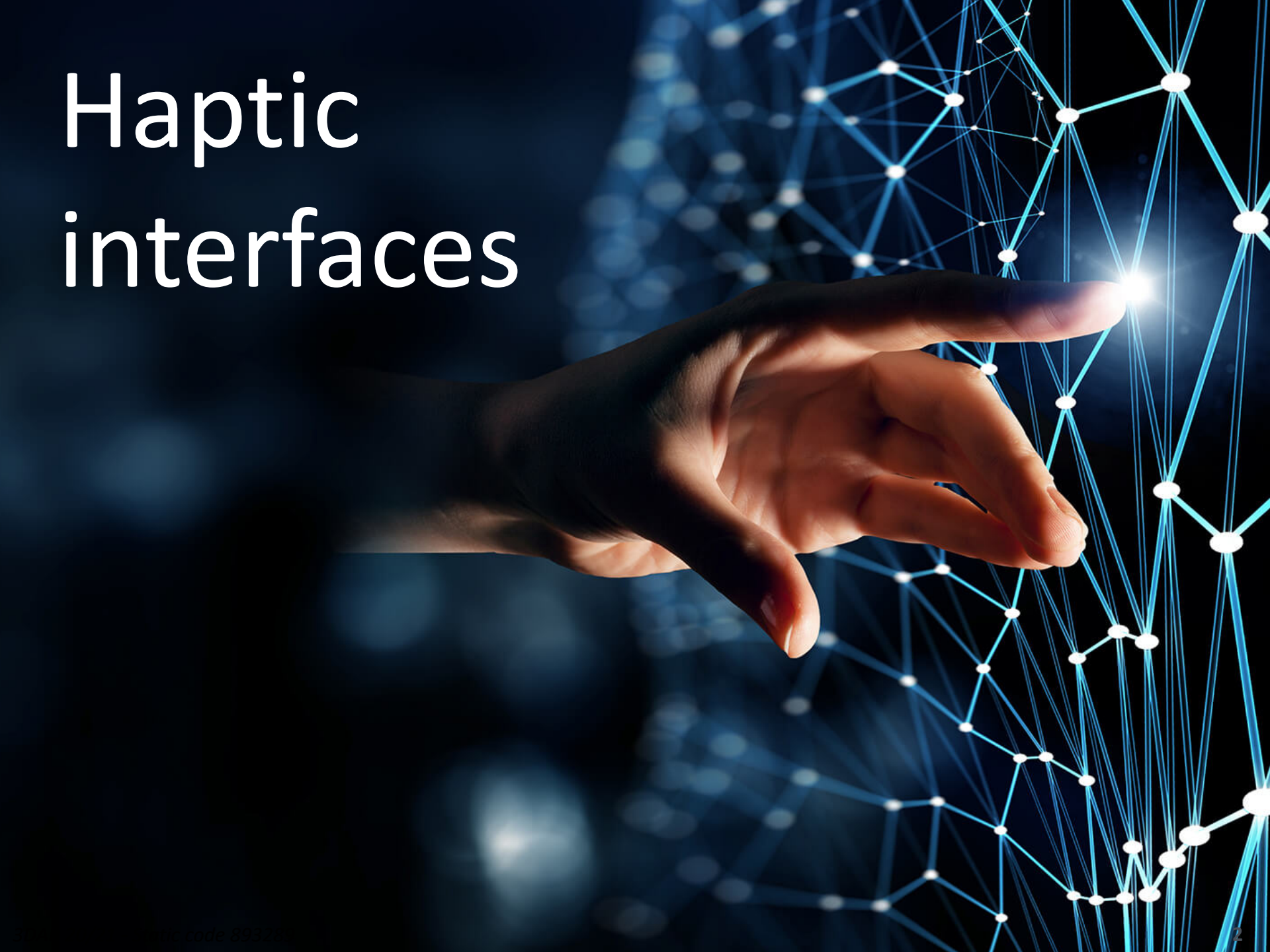
Static code
893289

Simone Milani
Room 216
DEI A

Phone: 049 827 7641
E-mail:
simone.milani@dei.unipd.it

3D AUGMENTED REALITY
A.A. 2020/2021
LECTURE 17 – HAPTIC/DL

Haptic interfaces



OTHER INTERFACES IN AR/VR

So far we have seen,

- Visors (standards screens, stereo VR, stereo see-through, projectors, …)
- 3D scanning camera (to model the space, to acquire movements, …)
- Audio (voice-based) interfaces
- controllers, joysticks (used by hands)

Other interfaces can be possible …

HAPTICS

Haptic technology, also known as kinaesthetic communication or 3D touch, refers to any technology that can create an experience of touch by applying forces, vibrations, or motions to the user

Actuators: create the sense of touch

Sensors: measure motion and poses with inverse process

HAPTIC ACTUATORS

Usually they combine force, vibration, motions, thermal stimuli: from voltage to stimulus

Manipulate the movements of the user (go beyond a mere alert)

Not all the sensors require touch.

- Tactile
- Vibrotactile
- Thermal

- Eccentric Rotating Mass Vibration (ERMV)
  Offcentered motion of a rotating element
- Linear Resonant Actuators (LRA)
  Similar to ERMV but motion is created by an EM field
- Piezoelectric Actuators
  Material reacts to current: expand and contract

} Vibration-based

- TENS
- Hydraulic
- Ultrasound (no touch)

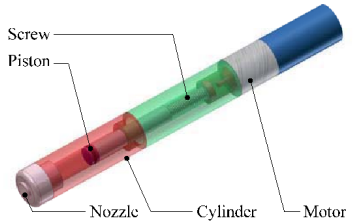
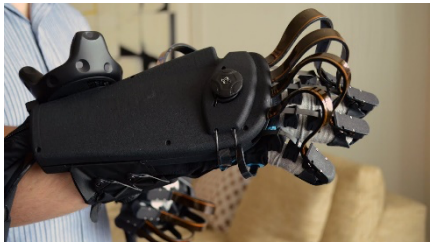


Fig. 4. Compact hydraulic actuator module for the slim artificial muscle.

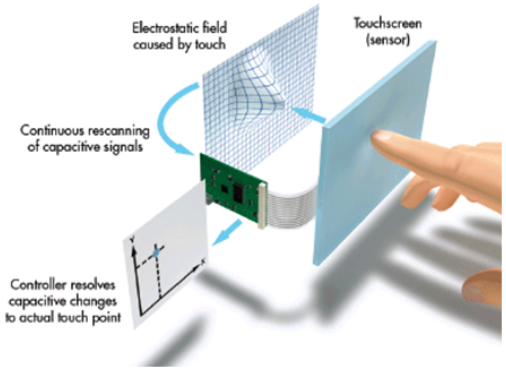


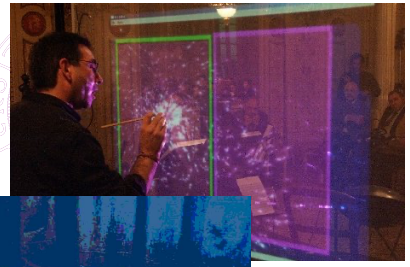
HAPTIC SENSORS

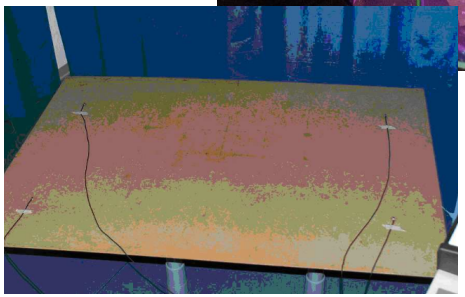
Invert the previous processes: from motion/pressure to voltage=data

The most widely-used haptic sensor: **touch screens**

- Resistive
- Capacitive
- Infrared
- Surface acoustic wave
- Acoustic-based

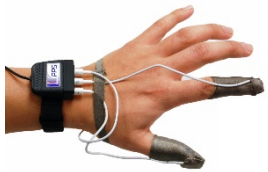






Gloves and wrist-bands

- Often uses a piezoelectric effect
- Relative position of fingers is measured



OTHER HAPTIC SENSORS

- **Electromyography** (EMG)-based
  Myo armband controller
- **Phantom arm based**
  Phantom Omni (touch feedback)
- **Infrared-based**
  Etee VR controlled, similar to leap-motion

HAPTIC INTERFACES

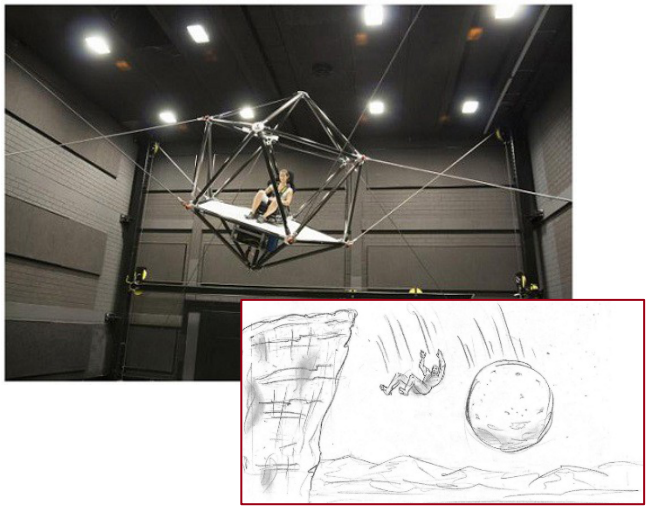
Omni-directional threadmill (illusion of walking)
Omnipad, Infinadeck, Cyberith

- Used also in CAVE
- Crucial if you do not want to limit the motion possibilities of the user
- Natural navigation of the scene
- Used in rehab as well

Haptic rigs (VR env pushes back against you)

- Full-body
- suspension

Axon VR



HAPTIC INTERFACES

Haptic suits

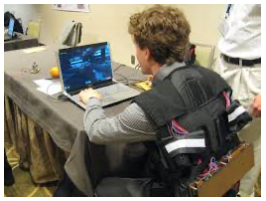
Tesla

- Movements and commands can be captured by sensors
- Suit provide a feedback to the user: sense of touching
- Temperature control

Transcutaneous electrical nerve stimulation (TENS or TNS/EMS)

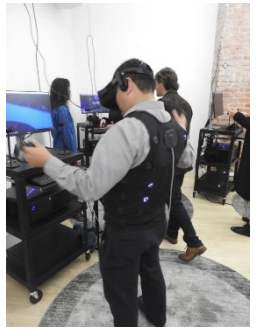
- Introduced to relieve pain
- Electrical stimulation by a couple of electrodes
- Stimulating the motor nerves, muscular contractions can be triggered that can be matched to a haptic event
- Avoid sensitive parts like brain, spinal column

Other products …



Tactile Gaming Vest (2010)

SoundShirt (2016)



NeoSensory exoskin (2018)

Other interfaces

TASTE AND SMELL

Smelling

FeelReal

Smell-integrating sensors exploit previous knowledge on smell synthesis.

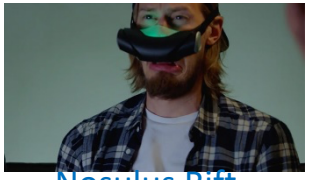
Scentography is the technique of creating and storing odor by artificially recreating a smell using chemical and electronic means.

Previous experience: Smell-O-Vision, AromaRama
(*Scent of Mystery*, Todd Jr., '60; *Behind the Great Wall, Lizzani, '59*)

Cartridges of different smells

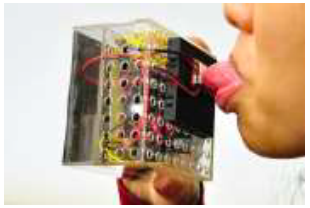
Taste

Experimental actuators that emulate all the fundamental tastes that our tongue can perceive: sour, salty, bitter, sweet + sensations like mint and spicy.



Nosulus Rift

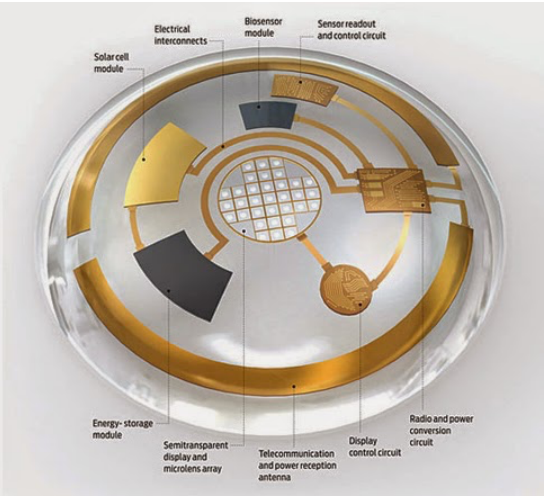
electrical stimuli (the different taste depends on amplitude and frequency of the electrical wave going through the tongue), while **sweet, mint and spicy can be emulated using thermal stimuli** (for instance, a cold metal can simulate mint).



University of Singapore

EXOTIC VISORS

NOVARTIS
Google



AR contact lens

Smart Hardhats

Windows (e.g., WayRay, hyperloop train, buildings)

more to come ….

MACHINE LEARNING IN AR

Supervised classification or learning: the classifier is a trained assuming that external teaching data are available (i.e., there is a set of training data where at each sample corresponds the correct classification label)

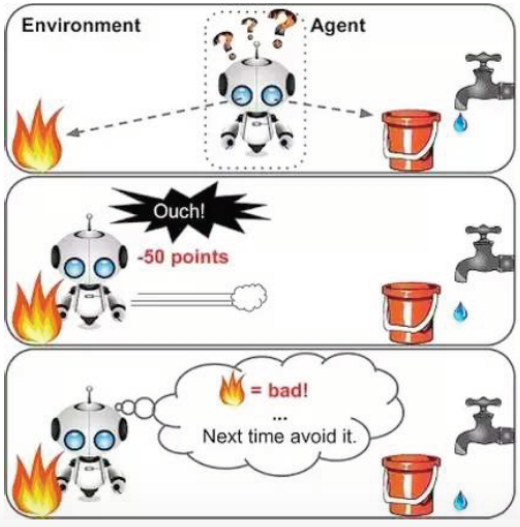
- Very powerful
- Requires extensive dataset and manual labelling of the samples to generate the training data

Unsupervised classification or learning: the classifier learns the underlying regularities and structure of the data without any prior information about their classes.

- Model building process
- Less accurate than supervised

Reinforcement learning: the classifier learns the underlying regularities and structure of the data by actively interacting with the environment.

- Extremely useful
- Computationally demanding



APPLICATIONS

- Basic 3D CV tasks
  (Feature computation, orientation, localization, 3D mapping, SLAM, depth estimation, image stitching)
- Object classification and identification (3D/2D)
- Gesture/speech recognition (interfaces)
  (voice commands recognition, hand pose identification)
- Data augmentation and enhancement (Image conversion)
- Optimizing data (coding/compression)
- Enhancing user experience

EXAMPLE: OBJECT DETECTION

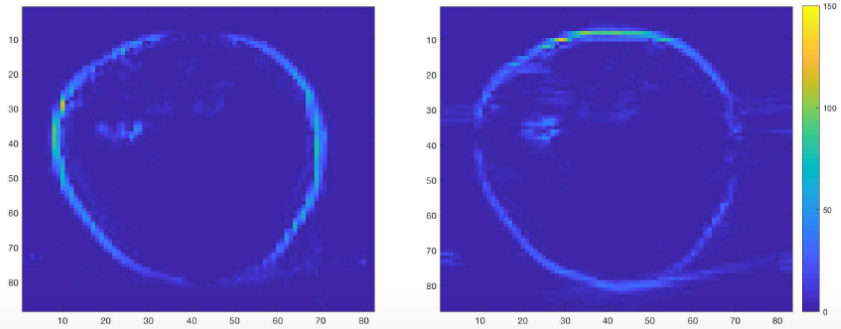
Possible features: vertical and horizontal gradients

(gradients ≈ object edges)

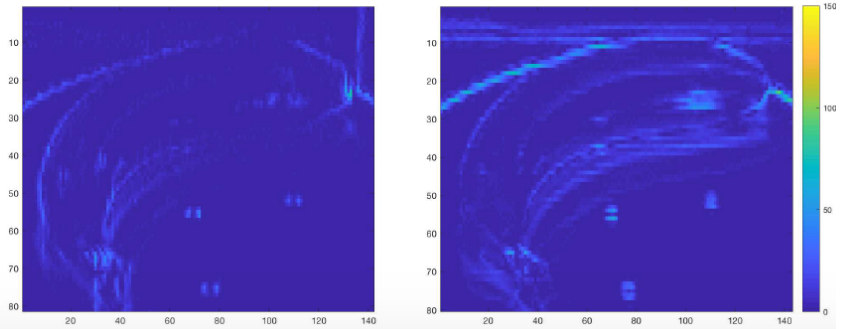
$$G_x(x,y) = I(x+1,y) - I(x,y) \quad \text{horizontal}$$

$$G_y(x,y) = I(x,y+1) - I(x,y) \quad \text{vertical}$$

$G_x(x,y)$ $G_y(x,y)$



$G_x(x,y)$ $G_y(x,y)$



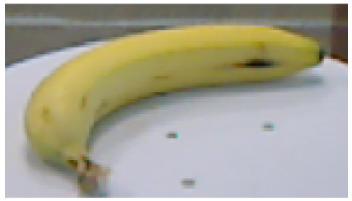
LINEAR CLASSIFIER

Let's try to classify object (belonging to a finite set of classes) from their images!

Apple

Edge are equally oriented along all the directions

Averages x_1 and x_2 are similar.



Banana

Edge are oriented along a main direction

Averages x_1 and x_2 are not similar.

Possible features: vertical and horizontal gradients

(gradients ≈ object edges)

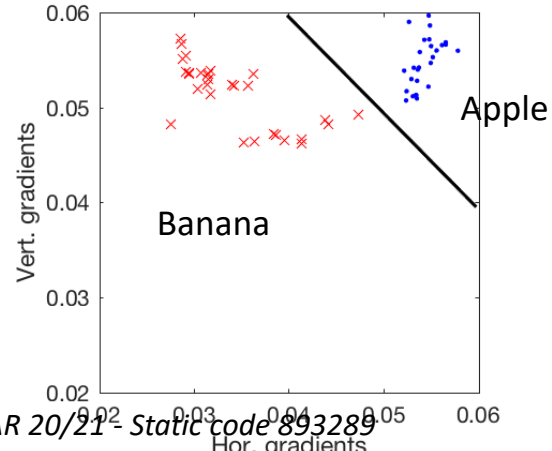
$$G_x(x,y) = I(x+1,y) - I(x,y) \quad \text{horizontal}$$

$$G_y(x,y) = I(x,y+1) - I(x,y) \quad \text{vertical}$$

Adopted features

$$x_1 = E\left[\left|G_x(x,y)\right|\right]_{x,y}$$

$$x_2 = E\left[\left|G_y(x,y)\right|\right]_{x,y}$$



Assuming that the line equation is $w_1 x_1 + w_2 x_2 = b$, the problem to be solved is the

$$\min \|\mathbf{w}\|^2$$

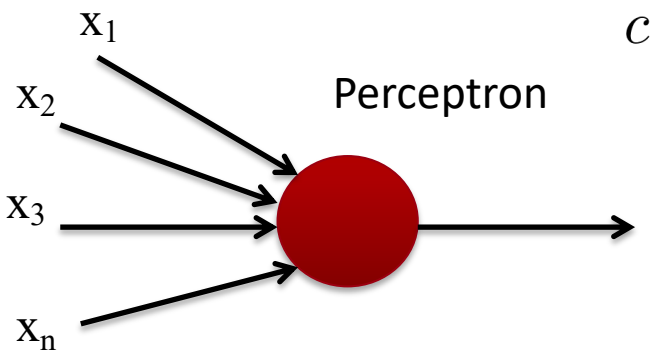
s.t. $\quad y_i\left(\mathbf{w}\cdot\mathbf{x}_i - b\right) \geq 1 \qquad \forall i$

TOWARDS DEEP LEARNING: PERCEPTRON

An R^n to binary maps

N input features ➡ binary output decision/classification

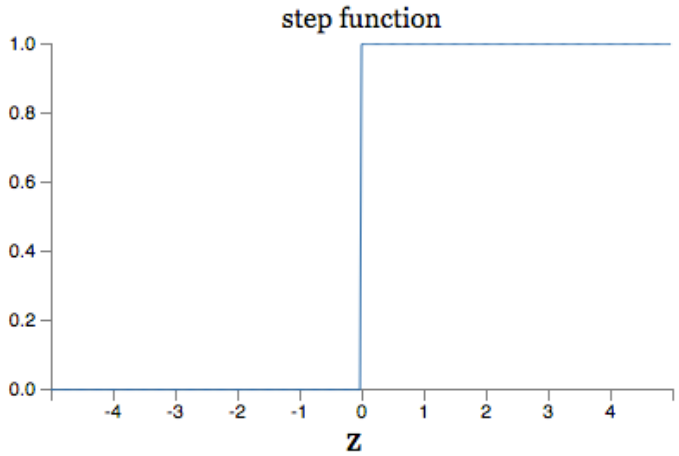
Perceptron



$$c = \begin{cases} 0 & \text{if } \sum_i w_i x_i - b = \mathbf{w}^T \mathbf{x} - b \leq 0 \\ 1 & \text{if } \sum_i w_i x_i - b = \mathbf{w}^T \mathbf{x} - b > 0 \end{cases}$$

It can be decomposed into a linear operator z + non-linear decision function f(): activation function

$$z = \sum_i w_i x_i$$

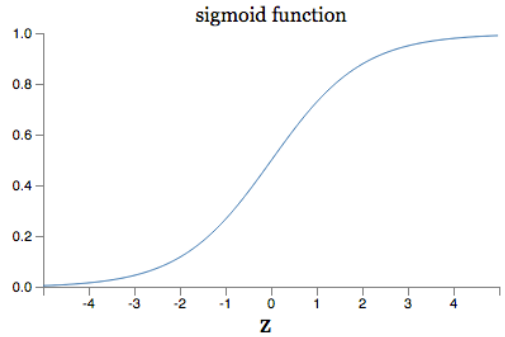


When z is higher than a threshold b, the perceptron/neuron fires

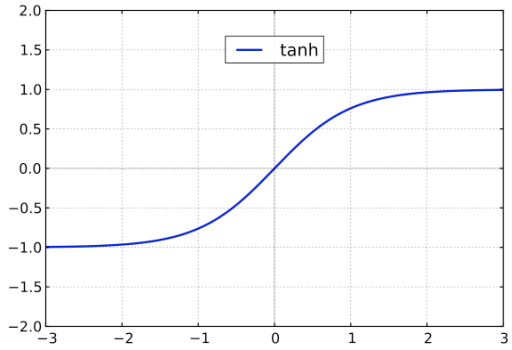
ACTIVATION FUNCTIONS

Sometimes a "real" output is required (or simply we would like to make the neuron fire with a different sensitivity).

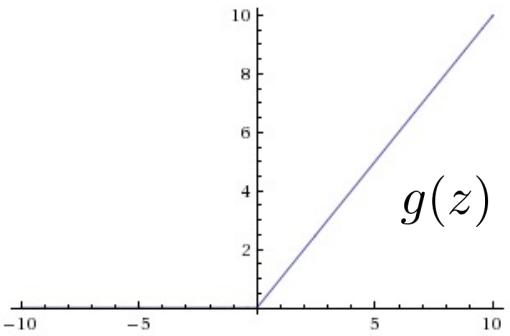
Different activation functions have been introduced.



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

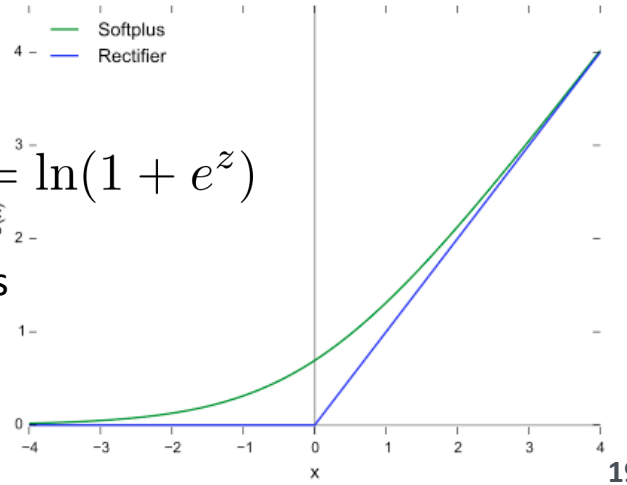


$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



$$g(z) = max\{0, z\}$$

Rectifies Linear Units (ReLU)



$$f(z) = \ln(1+e^{z})$$

Softplus

Modern NN uses ReLU most of the times

SoftMax function

I want an multi-dimensional output. **z** is a vector **z**=[z_i]

e.g. given a set of possible output options I want as output the probability of each option.

$$softmax(\mathbf{z})_j = \frac{\exp(z_i)}{\sum_j \exp z_j}$$

SoftMax	Sigmoid
multi-classification	binary classification
The sum of values will be 1	Values can be real number in [0, 1].
Used in the different layers of neural networks.	Used as activation function while building neural networks.

LOGISTIC REGRESSION

Logistic regression is a probabilistic, linear classifier.

The probability that $\mathbf{x}$ belongs to class i can be expressed as

$$P\left(y=i|\mathbf{x},W,b\right)=softmax(W\mathbf{x}+b)=\frac{e^{W_i\mathbf{x}+b}}{\sum_j e^{W_j\mathbf{x}+b}}$$

s.t. the prediction becomes

$$\hat{y}=\arg\max_i P\left(y=i|\mathbf{x},W,b\right)$$

Ok, but how to find good values for W and b?

We need a loss function!

likelihood

$$\mathcal{L}(W,b,\mathcal{D})=\sum_{i=0}^{N-1}\log P\left(\hat{y}=y_i|\mathbf{x}_i,W,b\right)$$

N: size of traiing set D

Loss or cost function

$$\ell(W,b,\mathcal{D})=-\mathcal{L}(W,b,\mathcal{D})$$

COORDINATES DESCENT (1/2)

$$f(x_1, \ldots, x_n) = f(W, \mathbf{b}) = \max_{W,\mathbf{b}} \ell\left(W, \mathbf{b}, \mathcal{D}\right) = -\sum_{i=0}^{N-1} \log P\left(\hat{y} = y_i | \mathbf{x}_i, W, \mathbf{b}\right)$$

Minimize the function using gradient descend: move in the direction defined by the derivative, ex. 2D case

$$f(x_1, x_2) \longrightarrow \nabla f(x_1, x_2) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)^T$$

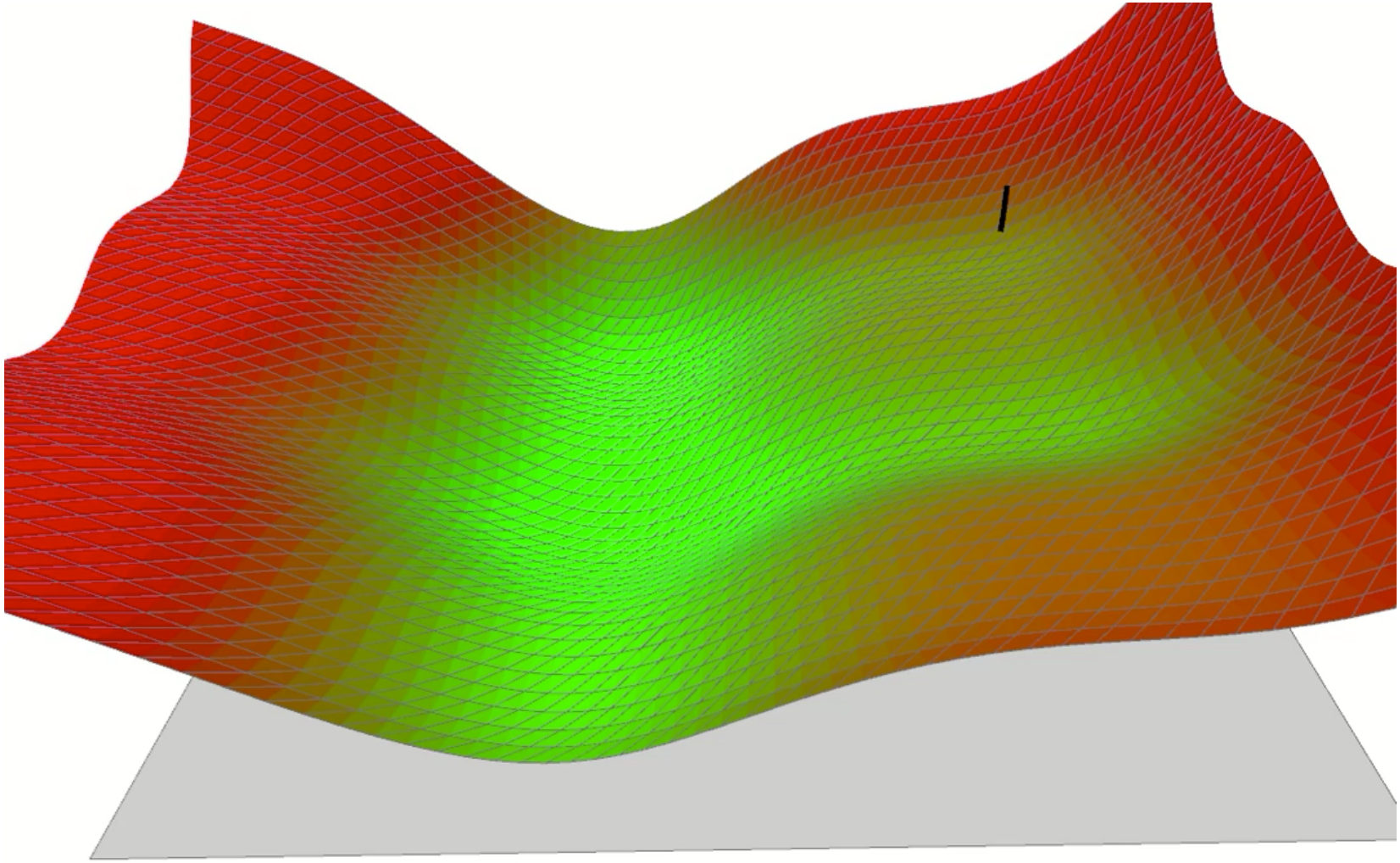
① Start at given location $\mathbf{x}_0 = \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix}$

② Compute the gradient $\nabla f(\mathbf{x}_0)$

③ Move in the direction opposite to gradient, i.e., $\mathbf{x}_1 \leftarrow \mathbf{x}_0 + \alpha \nabla f(\mathbf{x}_0)$

④ Iterate steps 1-3 until it does not move anymore!

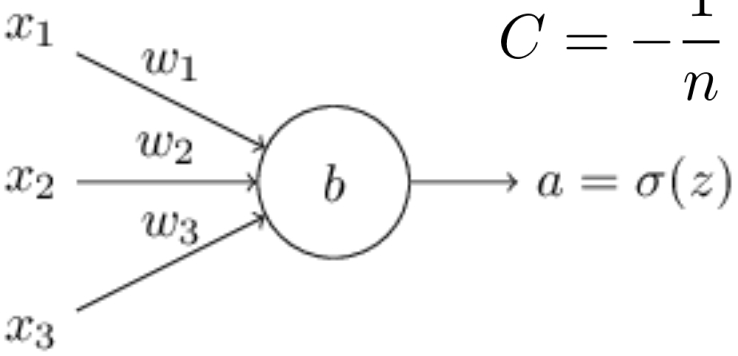
COORDINATES DESCENT (2/2)



CROSS-ENTROPY COST FUNCTION (1/2)

It may happen that learning speed goes down … the learning is too slow.

How can we address the learning slowdown?
quadratic cost -> **cross-entropy**.



$$C = -\frac{1}{n}\sum_i y_i \ln(\mathbf{a}(\mathbf{x_i})) + (1 - y_i)\ln(1 - a(\mathbf{x_i}))$$

$$a(\mathbf{x_i}) = \sigma(\mathbf{w}^T\mathbf{x}_i + b)$$

Partial derivatives then becomes

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\frac{\partial \sigma}{\partial w_j}$$

$$= -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)}\right)\sigma'(z)x_j$$

CROSS-ENTROPY COST FUNCTION (2/2)

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z)-y)$$

$$= \boxed{\frac{1}{n}\sum_x x_j(\sigma(z)-y).}$$

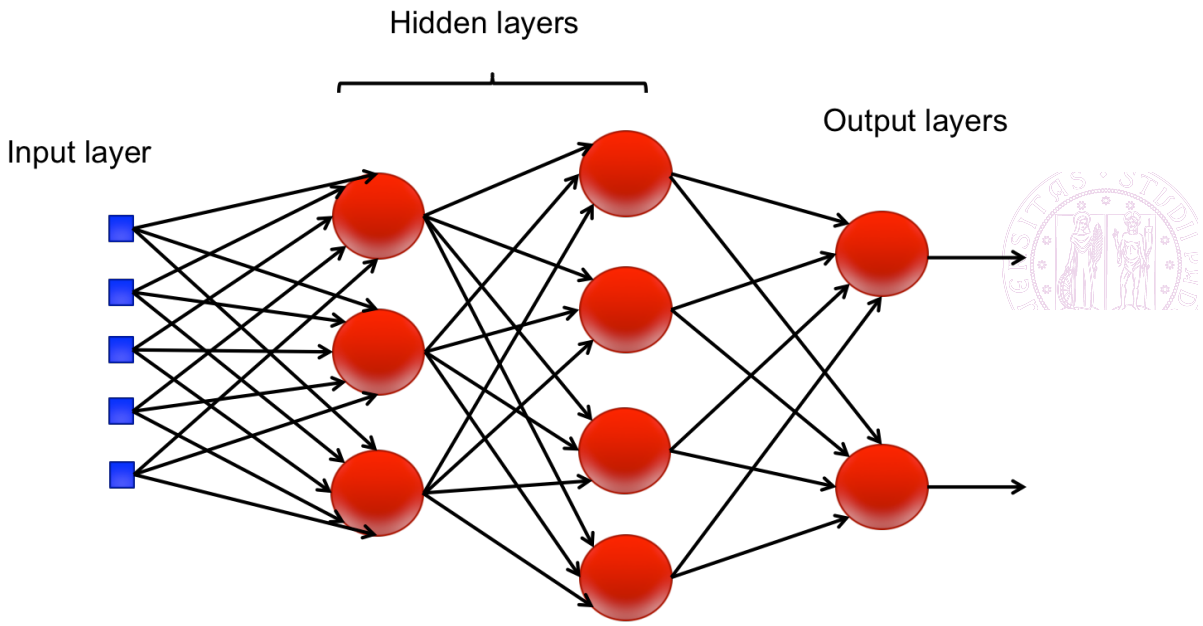
Error defines the learning speed!

It can be generalized to all the neurons in a network!

$$C = -\frac{1}{n}\sum_x \sum_j \left[y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)\right].$$

NEURAL NETWORKS

Combining multiple perceptron models together, it is possible to get a neural network.

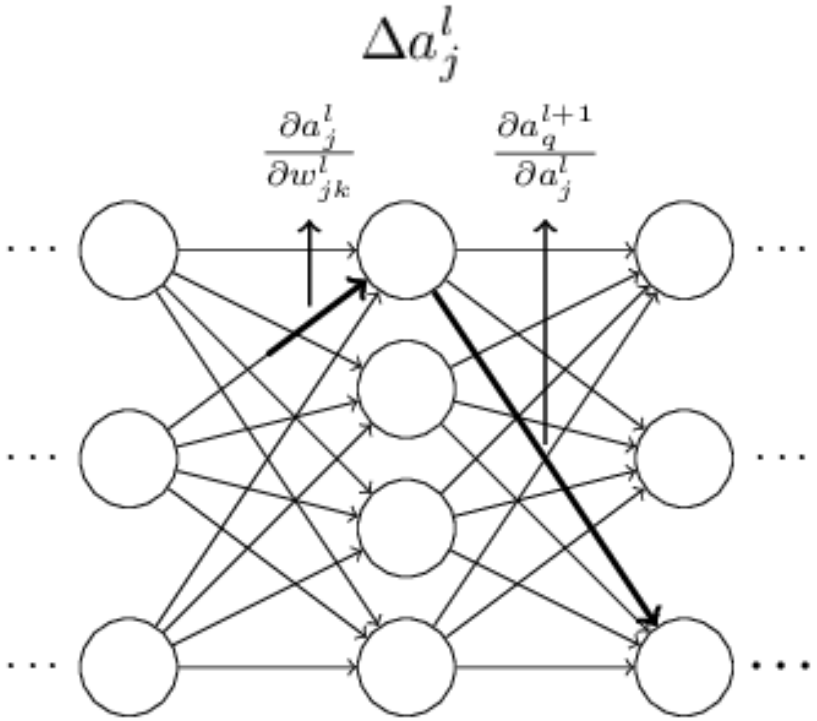


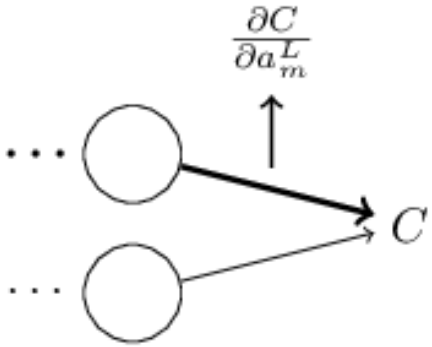
ERROR PROPAGATION

Error propagation depends on derivatives!

$$a_m^l = f(z_m^l)$$

activation





How to optimize weights parameters?

$$C(\overline{\mathbf{W}}, \overline{\mathbf{b}}) \triangleq \frac{1}{M} \sum_k \|\mathbf{y}(\mathbf{x}_k) - \mathbf{c}_k\|^2$$

Cost function to be minimized: difference between real and estimated classes. $\mathbf{y}(\mathbf{x}_k), \mathbf{c}_k$

Minimize C using gradient descend!

We need to compute derivatives!

$$a_m^l = f(z_m^l)$$

activation

$$\mathbf{z}^{l+1} = (W^l)^T \mathbf{a}^l(\mathbf{z}^l) + \mathbf{b}^l,$$

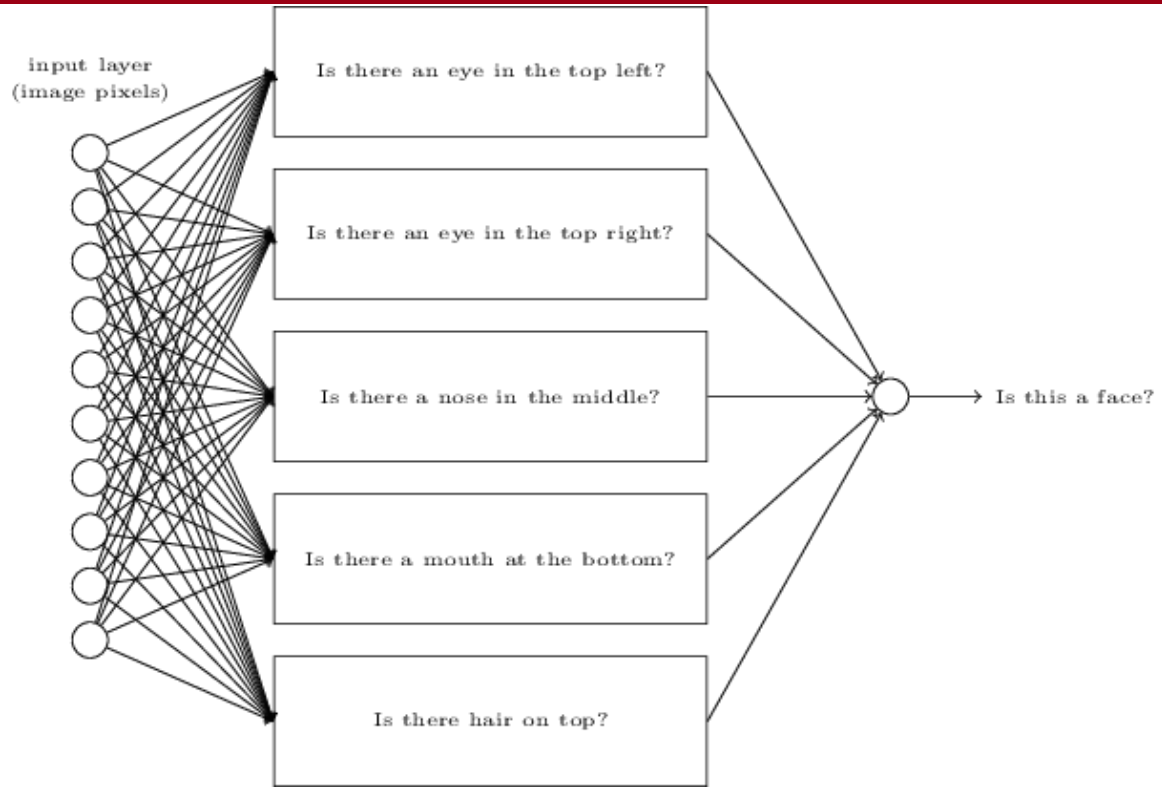
Setting $\delta_m^l = \frac{\partial C}{\partial z_m^l}$ the last layer allows to write (for quadratic distance)

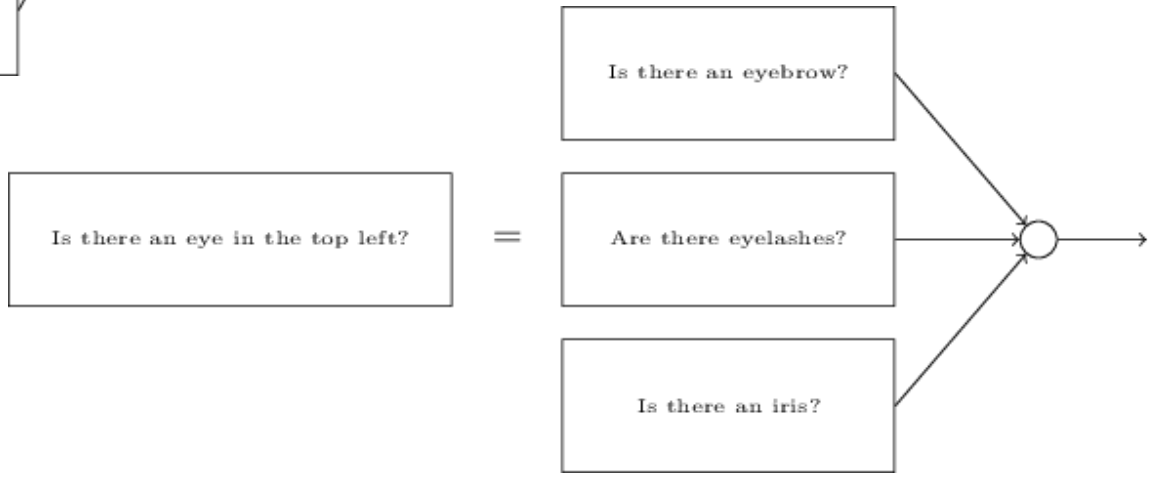
$$\delta_m^L = \frac{\partial C}{\partial a_m^L} a'(z_m^L) = \sum_k (\mathbf{c}_k - a(\mathbf{z}_m^L)) \odot \mathbf{a}'(\mathbf{z}^L)$$

Then it is possible to back-propagate the derivative to

$$\frac{\partial C}{\partial w_{m,i}^l} = \delta_m^l a_m^{l-1} \qquad \frac{\partial C}{\partial b_m^l} = \delta_m^l$$

DEEP NEURAL NETWORKS (DNN)

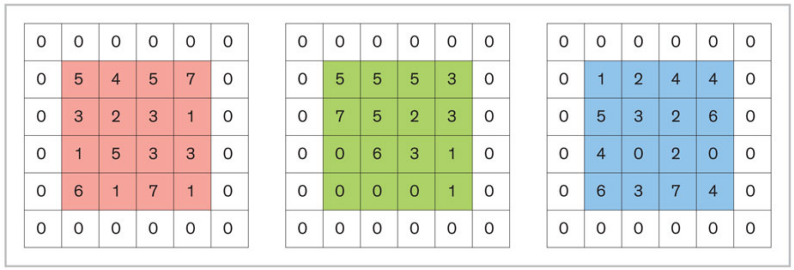




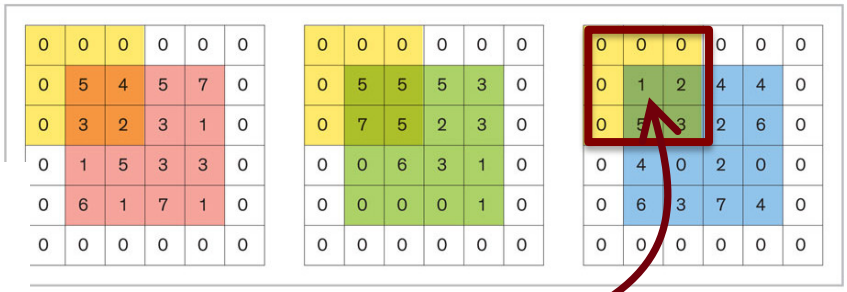
CONVOLUTIONAL LAYERS (1/2)

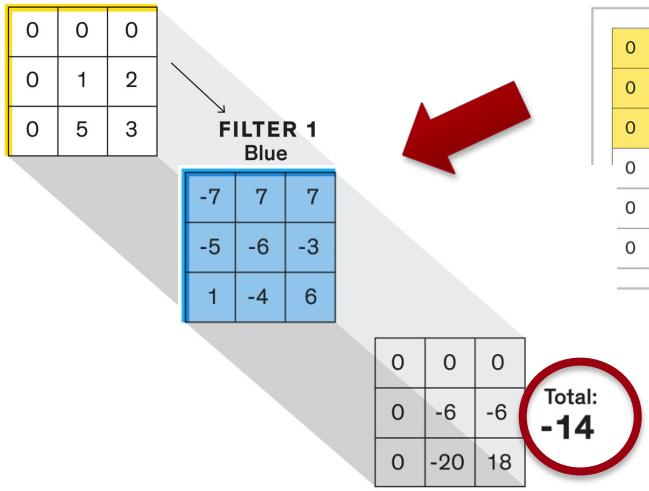
It is possible to extend the linear functional for z using a linear kernel: filtering operations!

Input image: R, G, B, components (padded with 0 on borders)



Filters with a finite support
(every yellow square outputs one value)



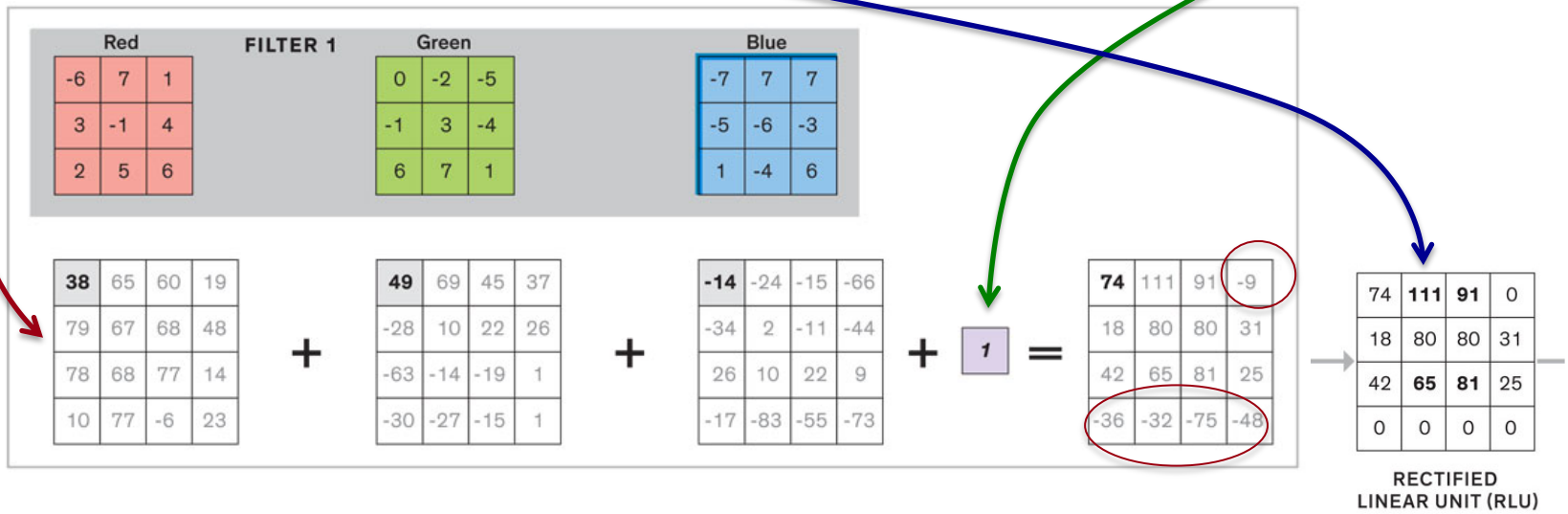


e.g. using the blue filter we have

The operations are repeating sliding the support window (filtering)

CONVOLUTIONAL LAYERS (2/2)

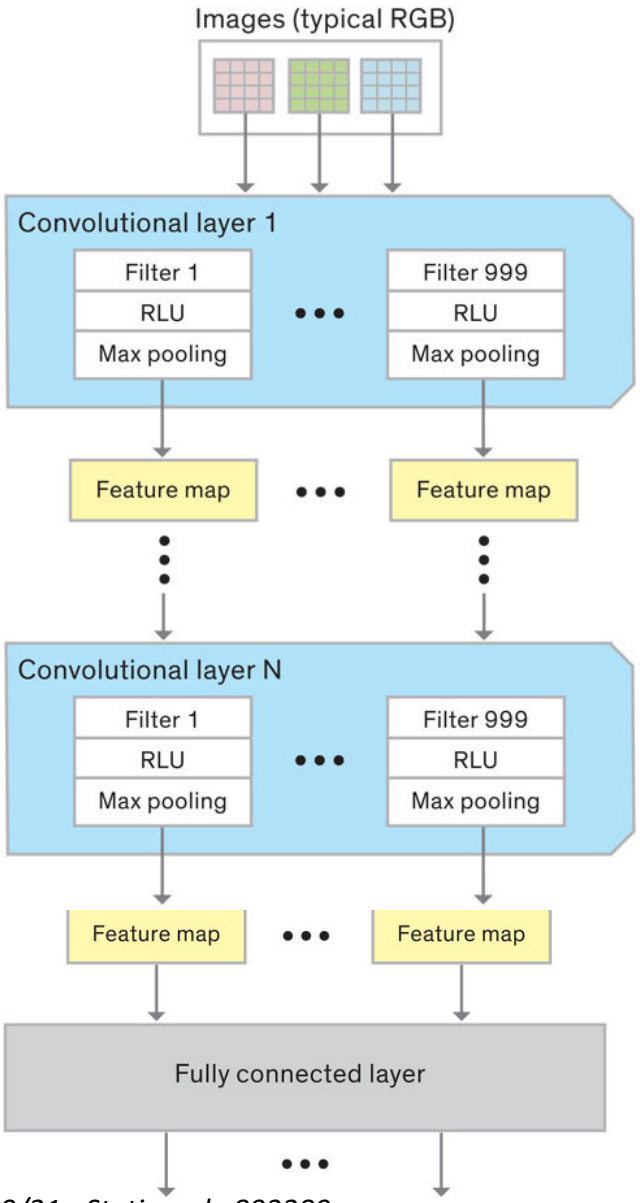
We obtain filtered components, which are then shifted by a constant and modified by the activation function (ReLU)



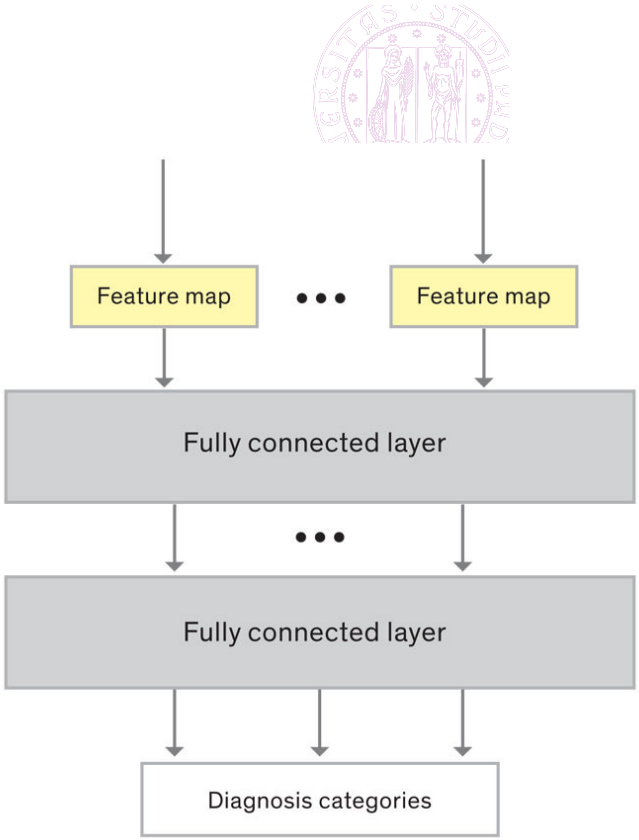
Then, max-pooling operation sample the input image into a smaller one, which is passed to the next layer.

Several filters are used for each color component.

CONVOLUTIONAL NEURAL NETWORK

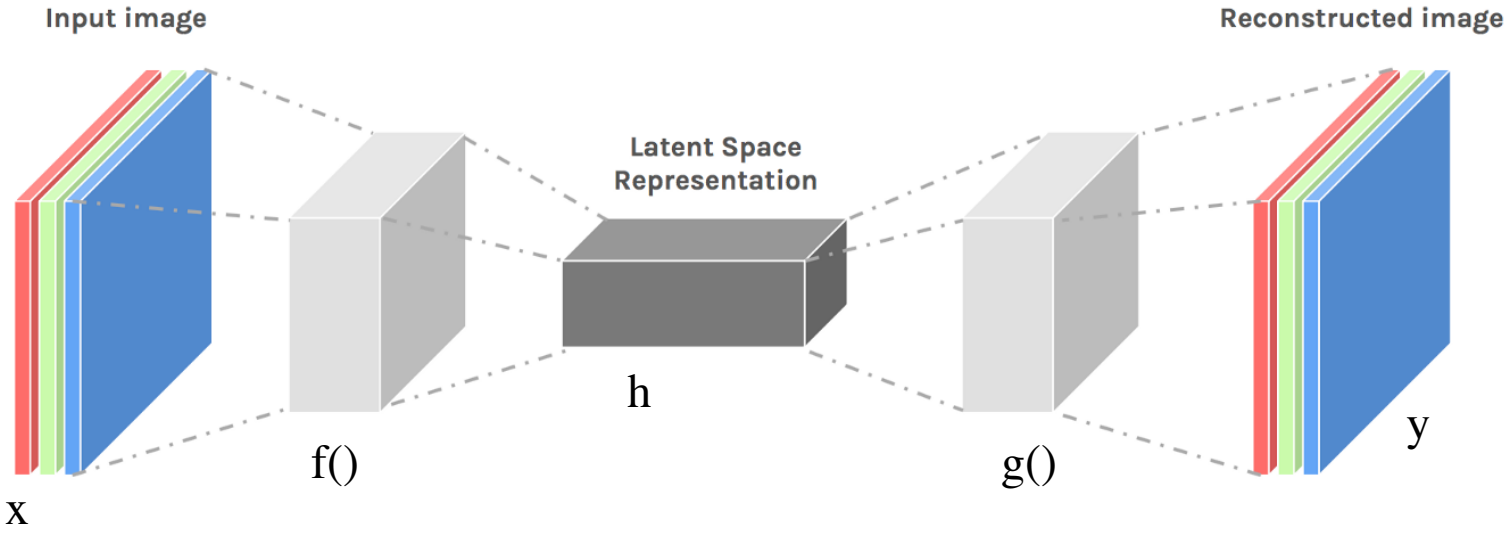


Several layers can be combined (with different input sizes), terminated by a fully connected layer (1 or more)



AUTOENCODER

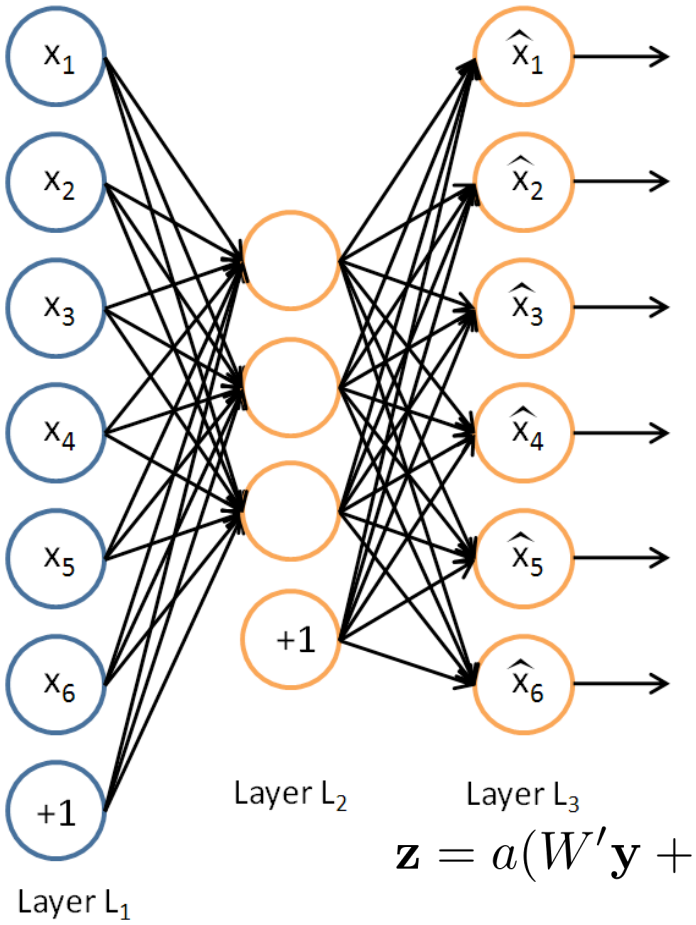
- NN trained to copy its input in the output $g(f(x))=x$
- Created in order to represent the input data by a set of features h (with a lower dimension)



Minimize the cost function $\mathcal{L}(\mathbf{x}, g(f(\mathbf{x})))$

In case a sparsity constraint is added $\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h})$

AUTOENCODER SKETCHED



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

$$\mathbf{z} = a(W'\mathbf{y} + \mathbf{b}')$$

$$W' = W^T$$

Minimize the distortion or cost function

$$L(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|^2$$

or

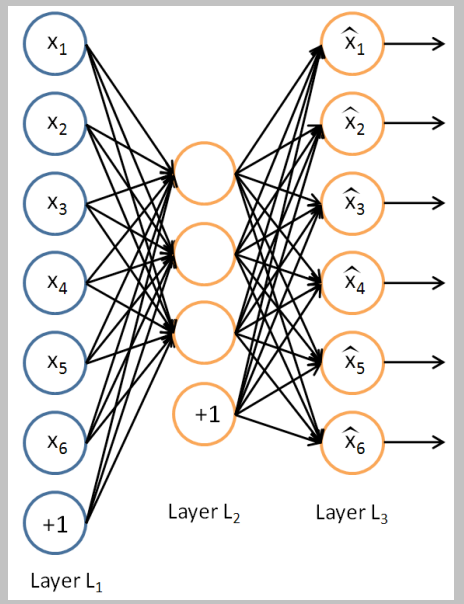
$$L(\mathbf{x}, \mathbf{z}) = -\sum_{k=1}^{d} x_k \log z_k + (1 - x_k) \log(1 - z_k)$$

Hidden layer define a set of features for the input data

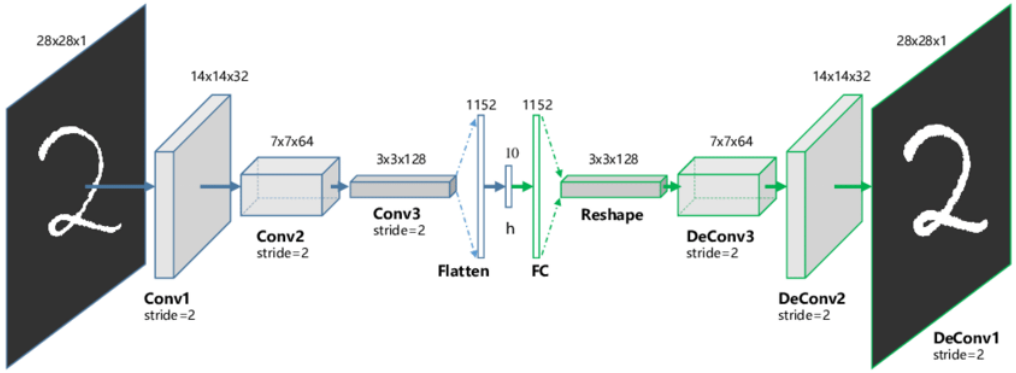
Can be used to denoise input-data: exploits the regularity of statistics.

THE SIMPLEST POSSIBLE KERAS AUTOENCODER

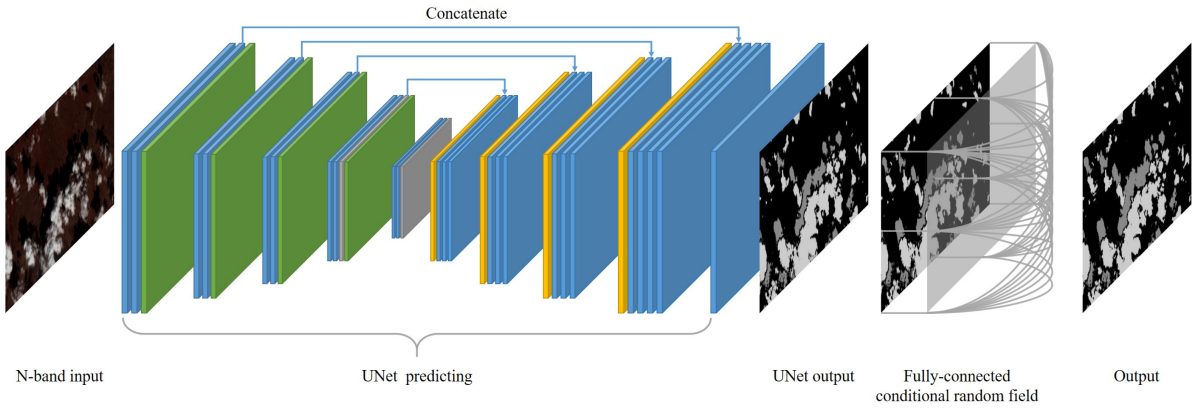
```
#import declarations
import keras
from keras import layers

# This is the size of our encoded representations (in floats)
encoding_dim = 32     # 32 floats

# This is our input vector
input_vet = keras.Input(shape=(64,))

# "encoded" is the  coded representation: hidden features
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction
decoded = layers.Dense(64, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_vet, decoded)
#Create the autoencoder
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

#Train the autoencoder
autoencoder.fit(x_train, x_train, epochs=50, batch_size=256, shuffle=True, validation_data=(x_test, x_test))

#Reconstruct the vector
decoded_vets = autoencoder.predict(x_test)
```



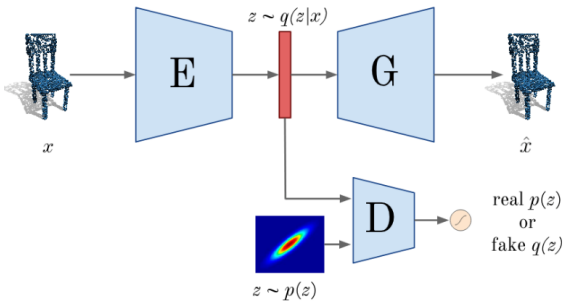
MORE AUTOENCODERS

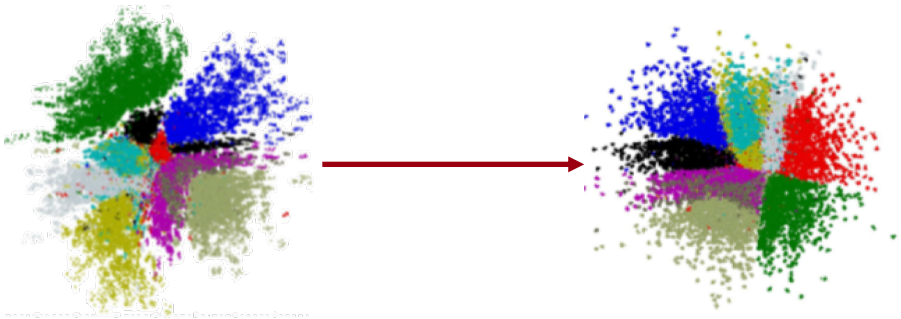


Convolutional



U-Net



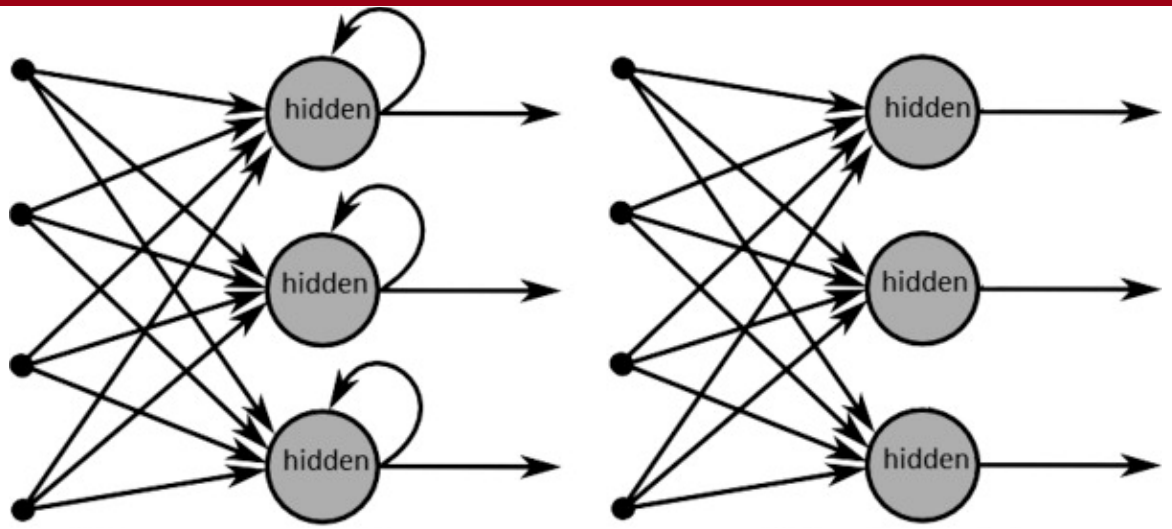


w/o adversarial

w/ adversarial

Adversarial
Autoencoder

RECURRENT NEURAL NETWORKS (RNN) -1/2



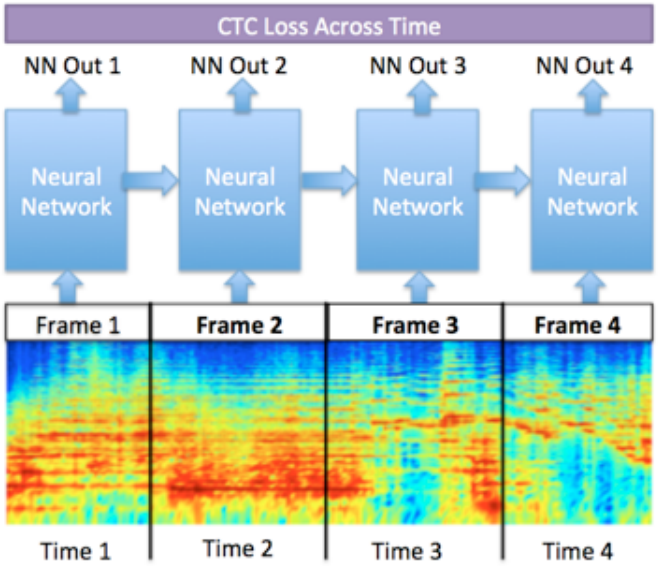
Recurrent

Feed-Forward Network

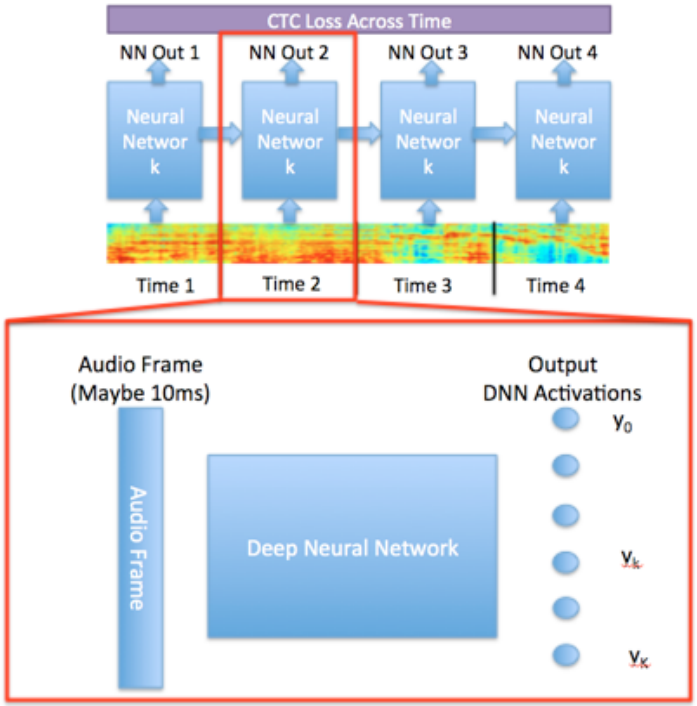
The input of a neuron depends on its output (at different time instants): cyclical connections

Example: speech recognition

Input: arrays of variables (samples) distributed over time.
Output: detected digit



RECURRENT NEURAL NETWORKS (RNN) -2/2



$$\mathbf{y}(t) = \sigma\left(W^L \mathbf{z}^L(t) + \mathbf{b}^L\right)$$

Elman's Network

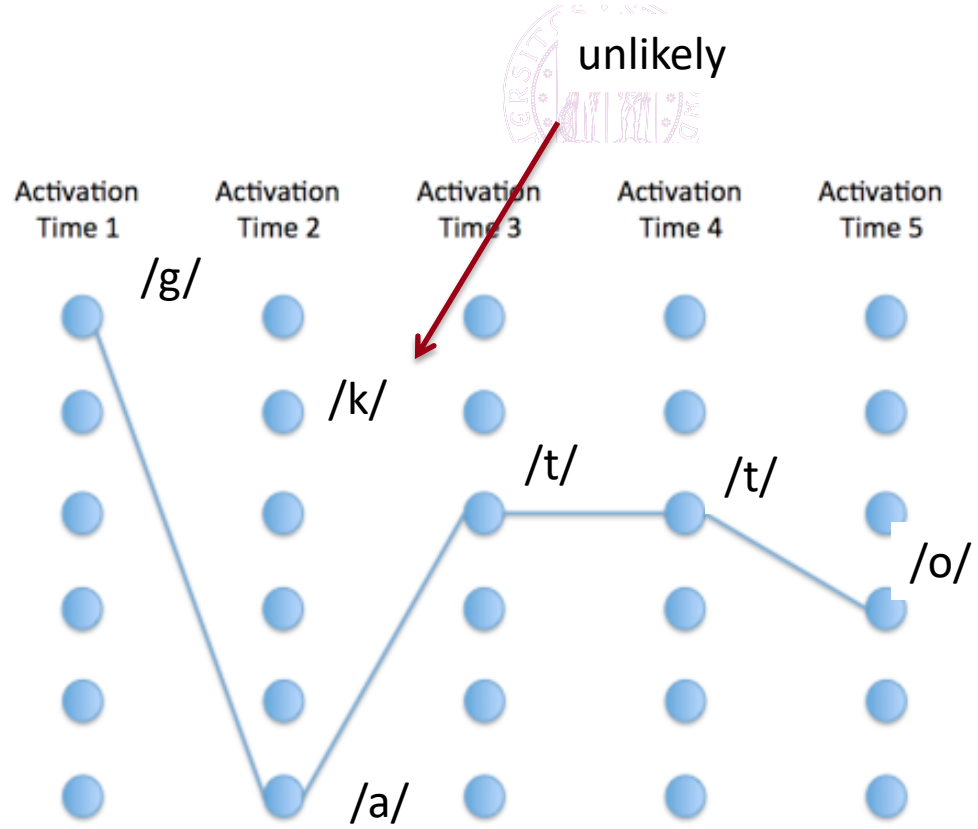
$$\mathbf{z}_k^l(t+1) = \sigma\left(W_k^l \mathbf{z}_k^{l-1}(t+1) + U_k^l \mathbf{z}_k^l(t) + \mathbf{b}_k^l\right)$$

Jordan's Network

$$\mathbf{z}_k^l(t+1) = \sigma\left(W_k^l \mathbf{z}_k^{l-1}(t+1) + U_k^l \mathbf{y}(t) + \mathbf{b}_k^l\right)$$

Feedback from previous instants conditions the output for the next instants.

There is a memory process in the detection.



GENERATIVE ADVERSARIAL NETWORKS (GANS)

$$J_D(\theta_G, \theta_D) = -J_D(\theta_G, \theta_D)$$

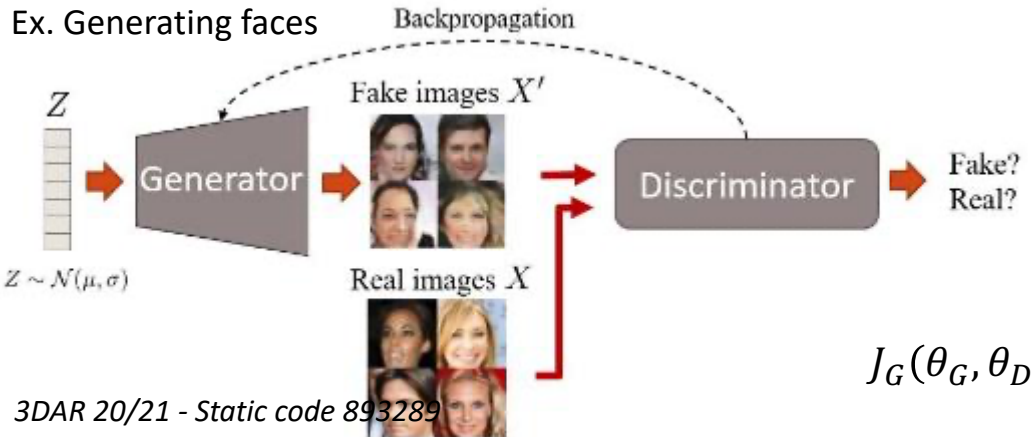
Real $D(\mathbf{x}) = 1$
or
Fake $D(\mathbf{x}) = 0$

DNN to find D(.)

discriminator

$\mathbf{x}$

Ex. Generating faces



$D(G(\mathbf{z}))$

discriminator

DNN to find D(.)

tries to have D(G(z))=0

$G(\mathbf{z})$ generator

DNN to find G(.)

aims at having D(G

$\mathbf{z}$

noise

$$J_G(\theta_G, \theta_D) = -\frac{1}{2}E[\log(D(x))] - \frac{1}{2}E[\log(1 - D(G(z)))]$$

GENERATIVE ADVERSARIAL NETWORKS (GANS)

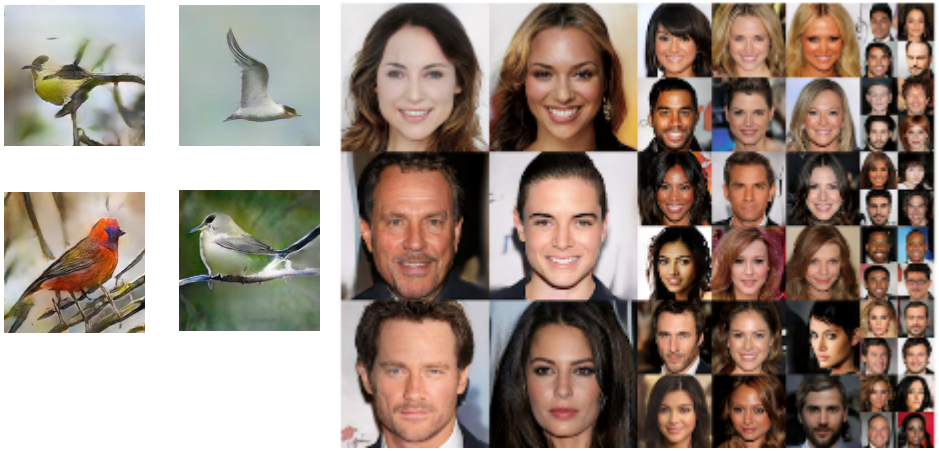
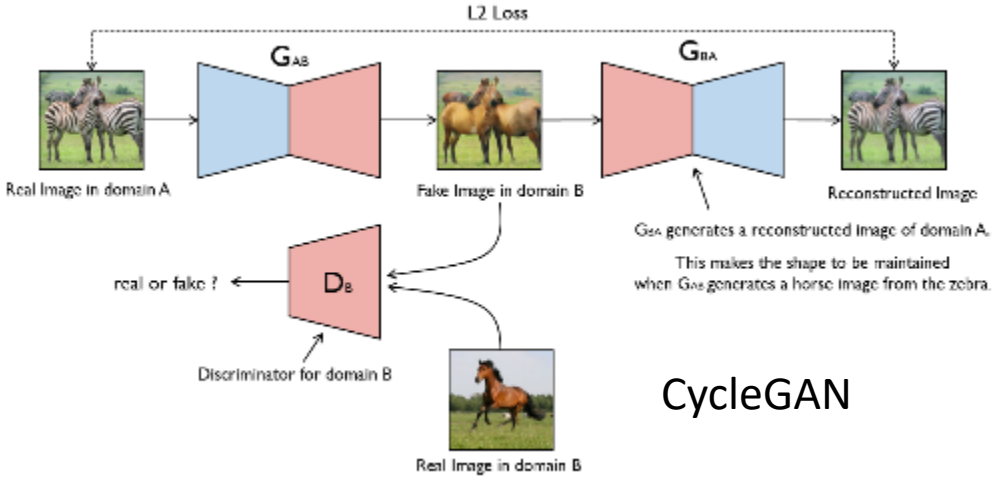
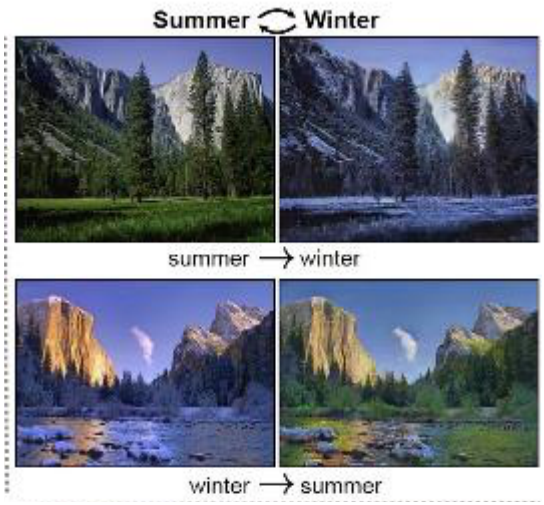


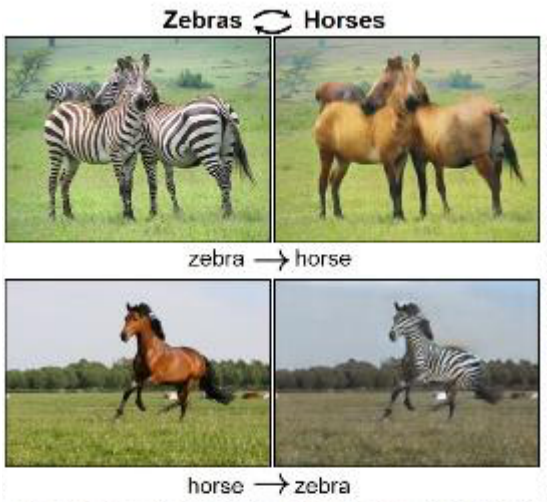
IMAGE-TO-IMAGE CONVERSION

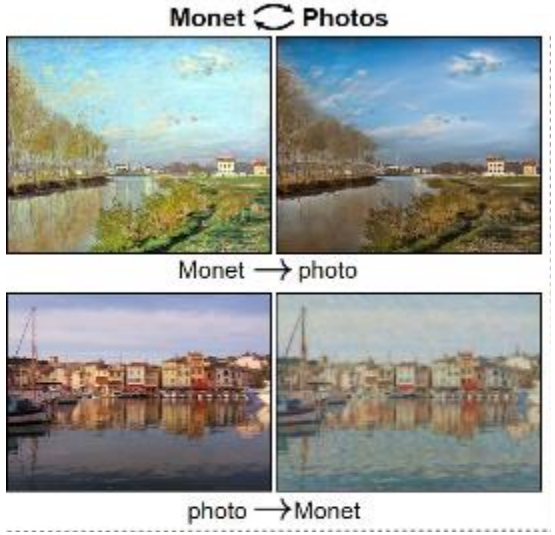
GANs can be used to map an image or face over another



CycleGAN

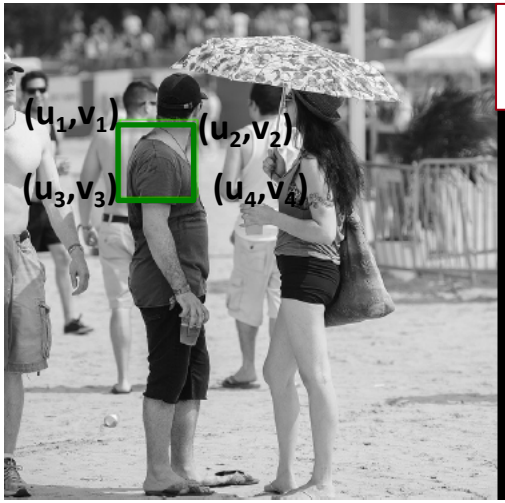


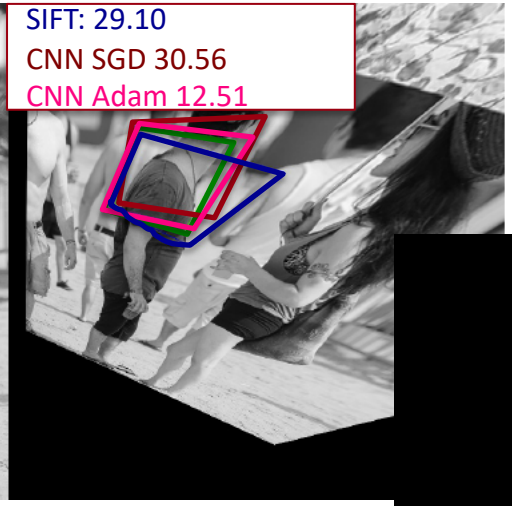


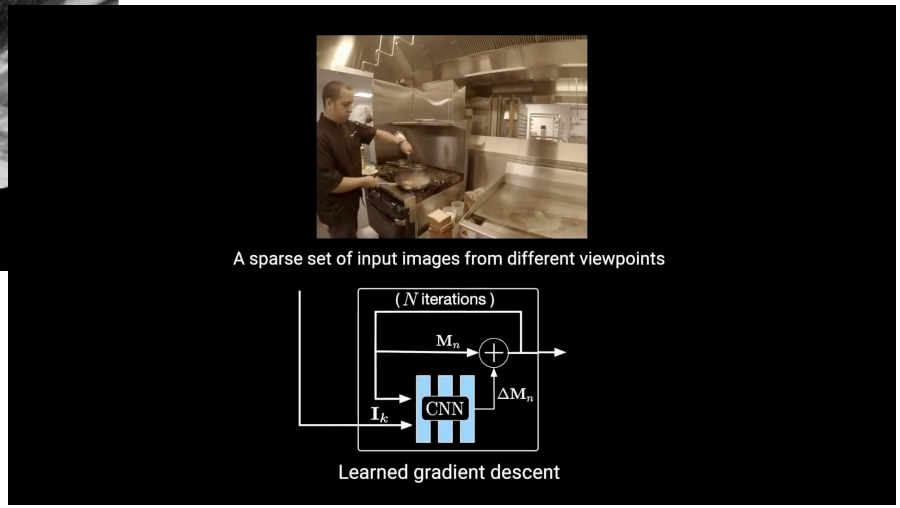


3D COMPUTER VISION TASKS

- Panoramic video and image stitching
- Feature computation, orientation, localization, 3D mapping, SLAM (see SfM lecture)
- Depth estimation
- 3D reconstruction and rendering
- Pose/Gaze estimation
- Scene composition







INTERFACES

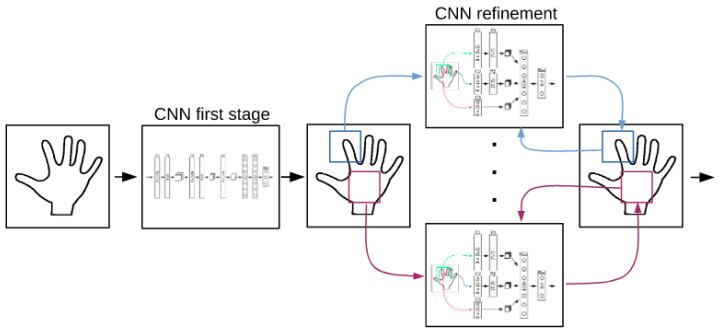
- Hand pose estimation
- Gaze estimation
- Voice processing
- BCI

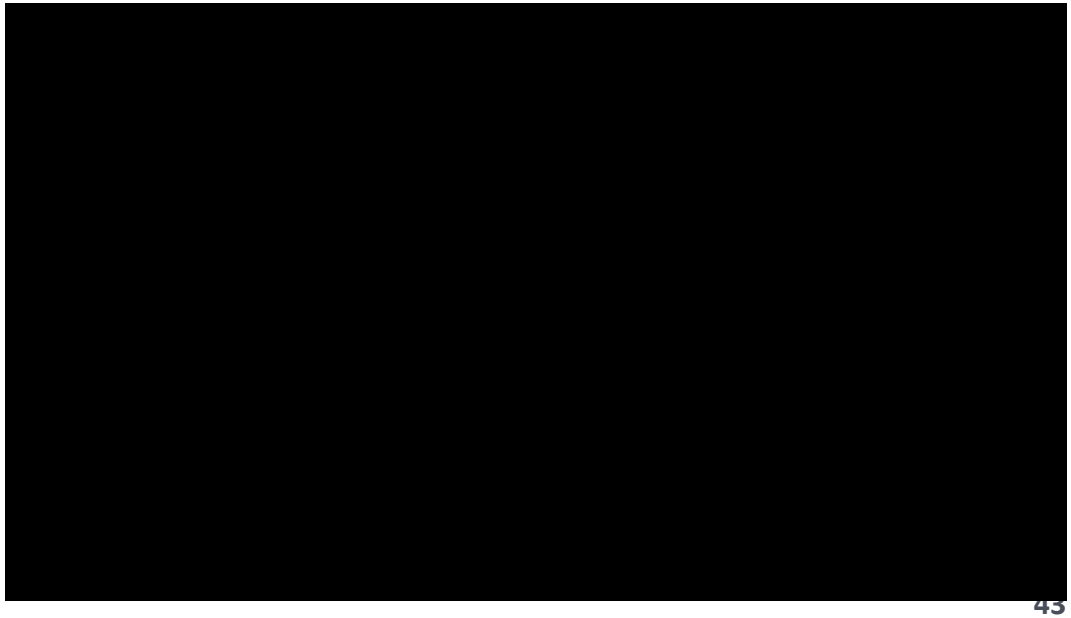
	G1	G10	G11
G1	91.04%	0.83%	8.13%
G10	0%	88.96%	11.04%
G11	0.21%	5.83%	93.86%

G1

G11

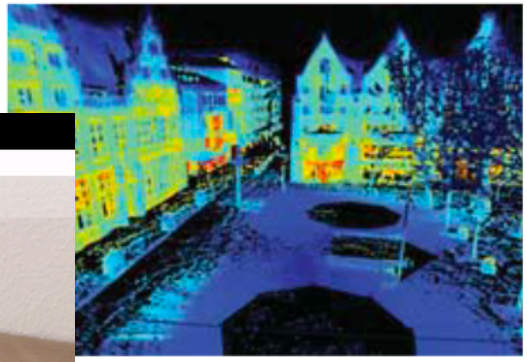
G10

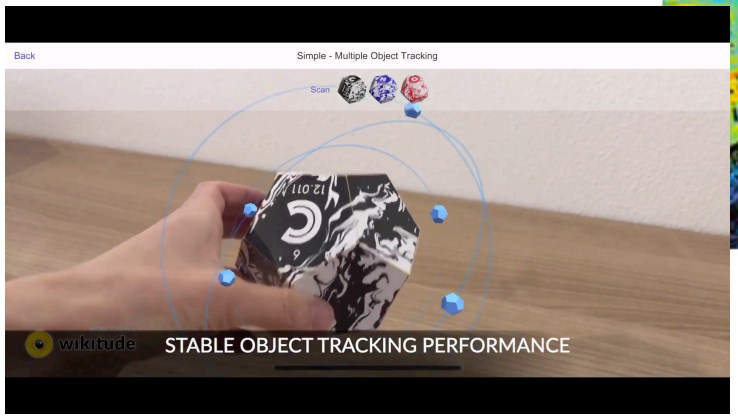


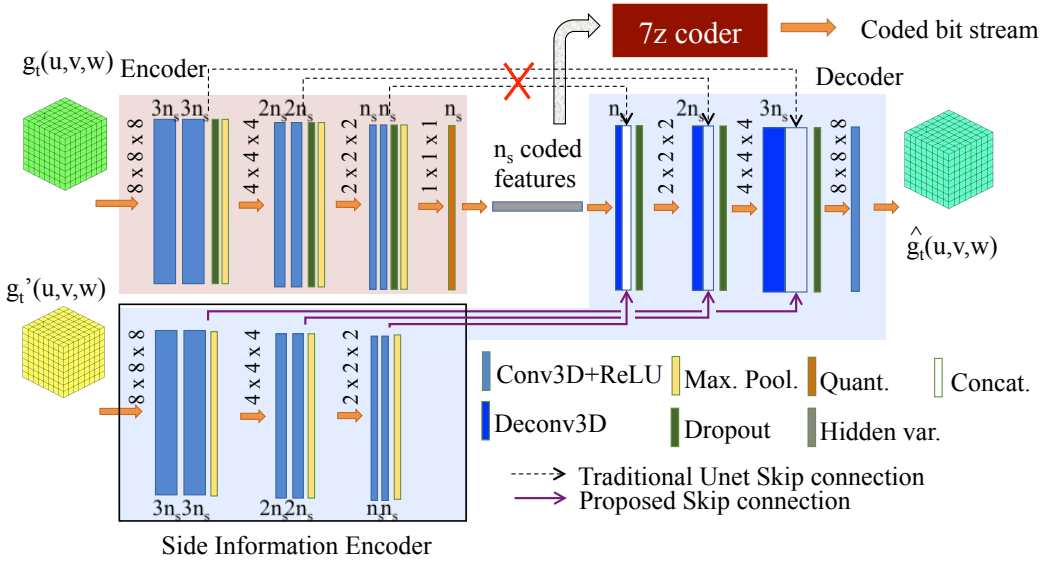


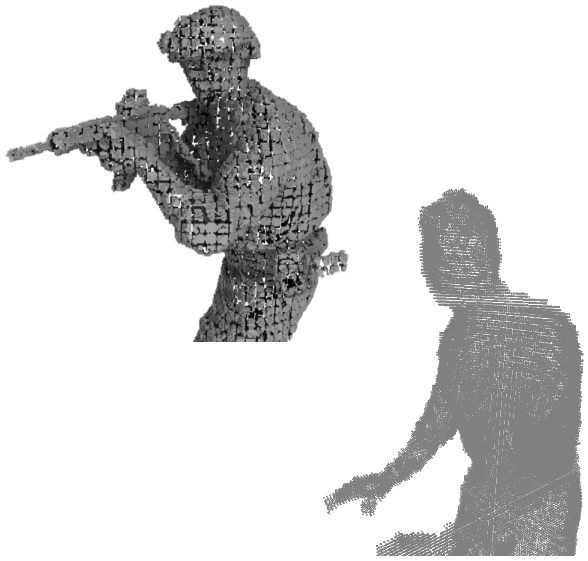
OBJECT DETECTION AND SEGMENTATION

- Object detection and classification
- Semantic segmentation
- Point cloud processing









HUMAN IN THE LOOP

- Model rendering/coding and optimization
- QoE, Cybersickness management
- Image enhancement

  e.g. GAN-based real-time food-to-food translation