INTRODUZIONE ALL'AI E AL MACHINE LEARNING PER SPECIALISTI DELL'INGEGNERIA – SECONDO INCONTRO

INTELLIGENZA NEL TRATTAMENTO DEI DATI STRUTTURATI E SEMI-STRUTTURATI:

MACHINE LEARNING

AGENDA

- **CONVEGNO ON LINE 1: Martedì 10 Ottobre, ore 15.00 – 17.00**
- Introduzione ai sistema informativi, Introduzione alle applicazioni data-driven: dalle basi di dati ai dati di addestramento per l'AI, Elementi di Data Management: dai modelli relazionali alle basi di conoscenza.

- **CONVEGNO ON LINE 2: Martedì 17 Ottobre, ore 15.00 – 17.00**
- Introduzione all'Intelligenza Artificiale: tra rappresentazione della conoscenza, ragionamento e apprendimento automatico

- **CONVEGNO ON LINE 3: Martedì 31 Ottobre, ore 15.00 – 18.00**
- Intelligenza nel trattamento dei dati strutturati e semi-strutturati: il Machine Learning

- **CONVEGNO ON LINE 4: Martedì 10 Novembre, ore 15.00 – 18.00**
- AI Generativa e Large Scale Language Models

OVERVIEW

CONSIGLIO NAZIONALE DEGLI INGEGNERI

CENTENARIO DELL'ALBO DEGLI INGEGNERI

Con la collaborazione incondizionata della Associazione Italiana di Intelligenza Artificiale

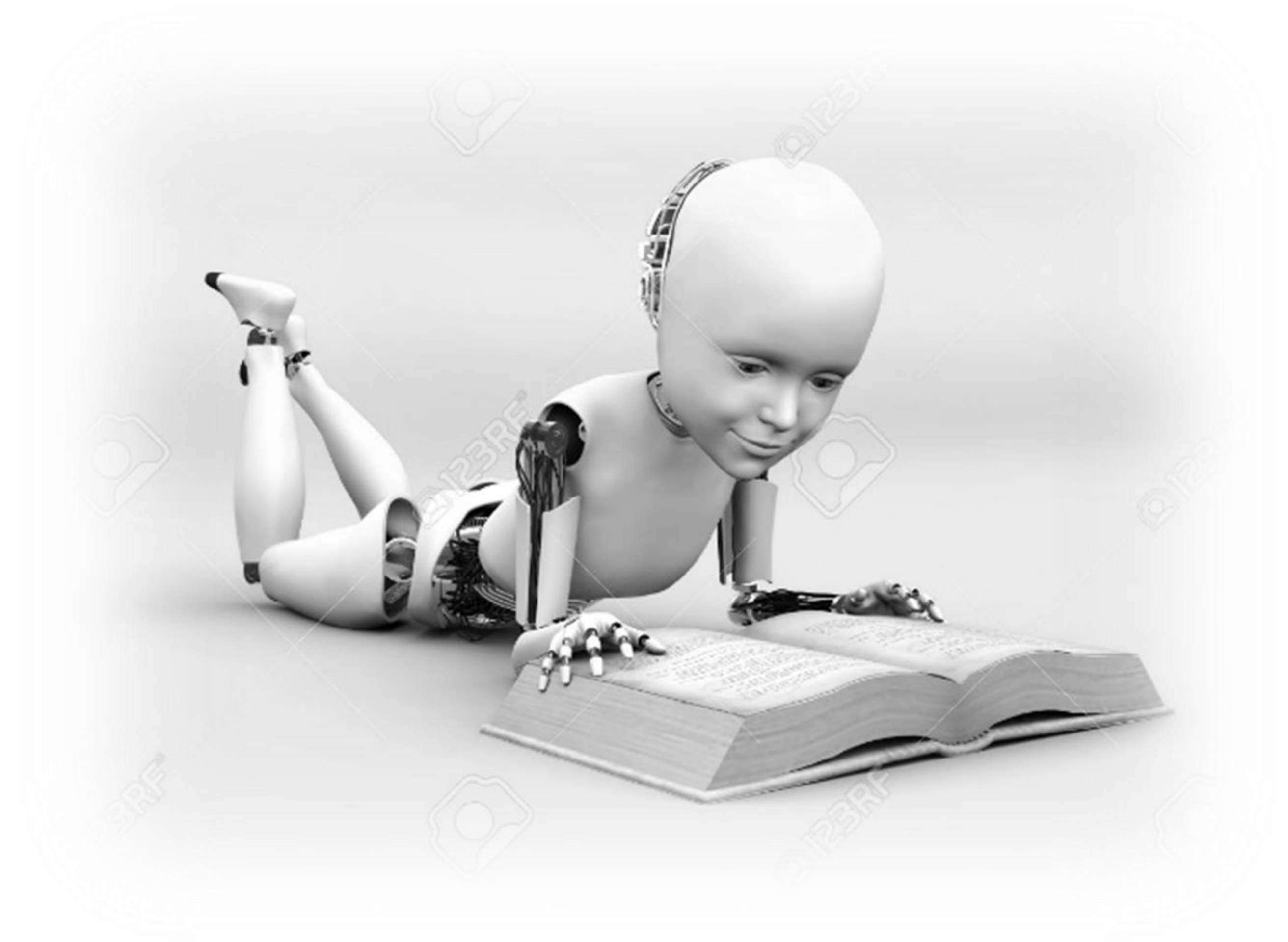
Associazione Italiana per l'Intelligenza Artificiale

- Il Machine Learning: definizioni e obbiettivi
- Statistical Learning Theory
- Le Reti Neurali: dai percettroni ai Transfomers
  - Multilayer Perceptrons
  - Le reti Convoluzionali e le immagini, Reti Ricorrenti, Reti Attenzionali e Autoencoders: i Trasformers
- Applicazioni avanzate ai dati non strutturati
  - ImageNet: Image Processing, Classification
  - Immagini e Testi: Automated Captioning
  - Visual Question Answering
  - Multimodality

MACHINE LEARNING

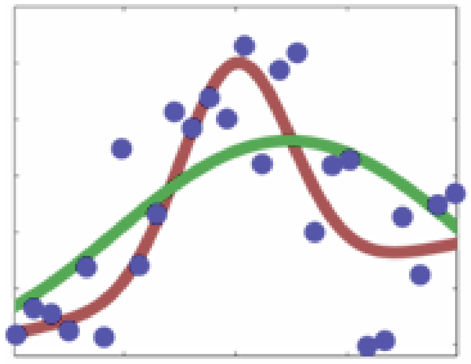
DEFINIZIONI ED OBBIETTIVI

COSA SIGNIFICA APPRENDERE DAI DATI?

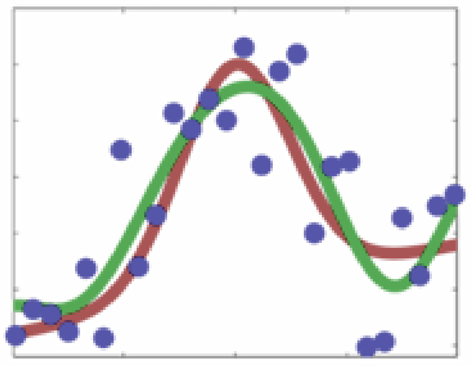


LEARNING MACHINES

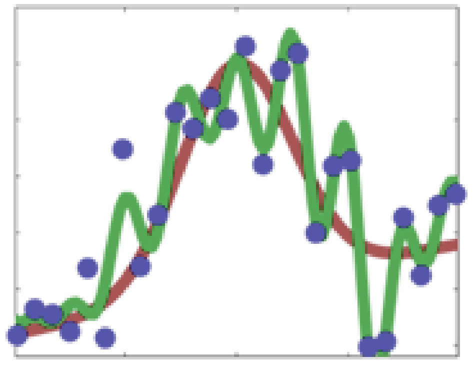
Funzione indotta
& modello troppo semplice

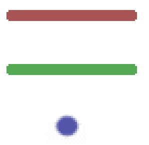


Funzione indotta
& modello adeguato



Funzione indotta
& modello troppo complesso



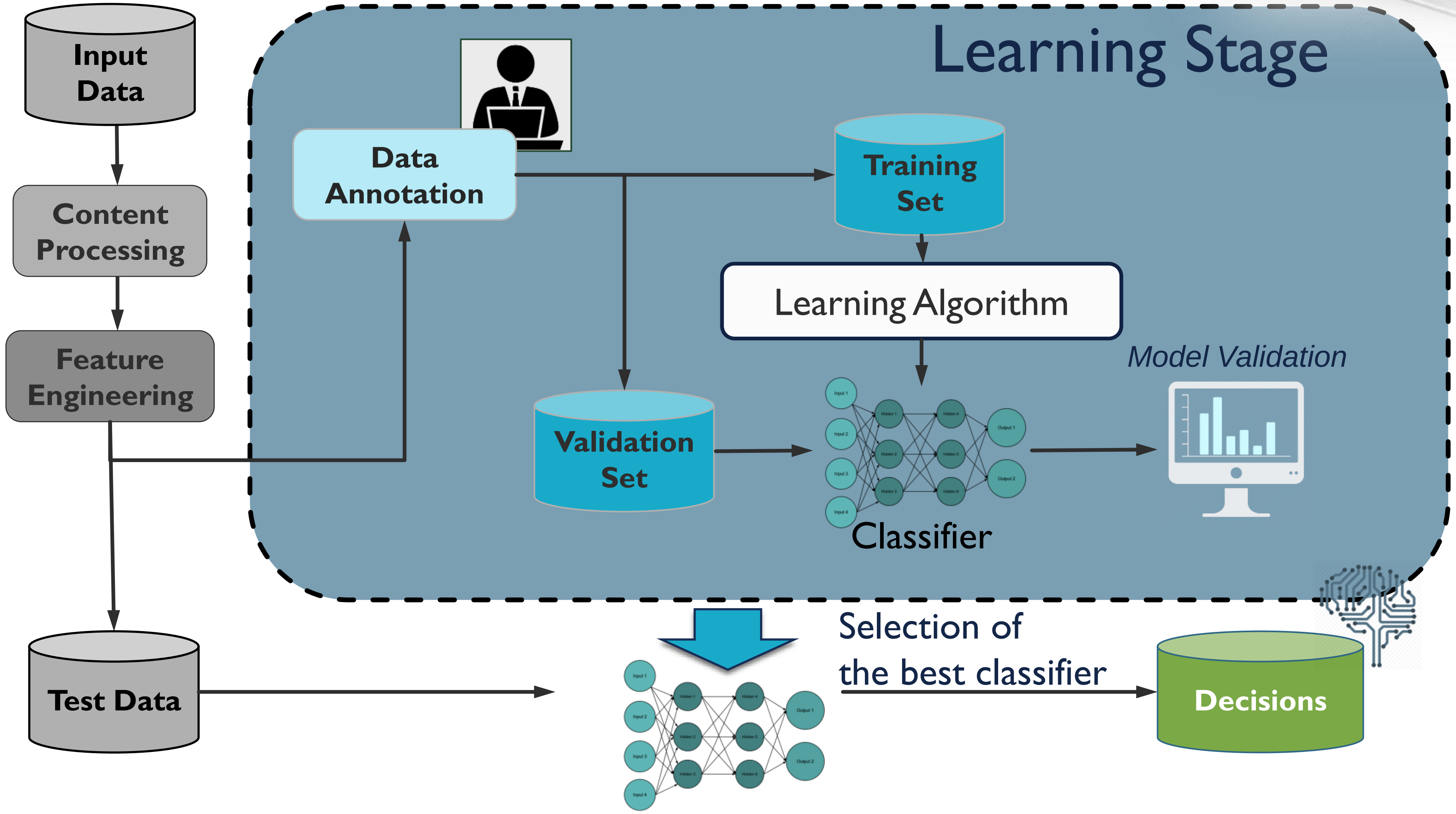


Funzione Target

Funzione Indotta

Istanze di Esempio

MACHINE LEARNING WORKFLOW



MACHINE LEARNING: DEFINITION

- *A computer program is said to* ***learn*** *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E* [Mitchell]
- Problem definition for a learning agent
  - Task T
  - Performance measure P
  - Experience E

DESIGNING A LEARNING SYSTEM

1. Choosing the training experience
   - Examples of best moves, games outcome …
2. Choosing the target function
   - board-move, board-value, …
3. Choosing a representation for the target function
   - linear function with weights (hypothesis space)
4. Choosing a learning algorithm for approximating the target function
   - A method for parameter estimation

INDUCTIVE LEARNING: LEARN A FUNCTION FROM EXAMPLES

- Simplest form: learn a function from examples

f is the target function

An example is a pair $(x, f(x))$

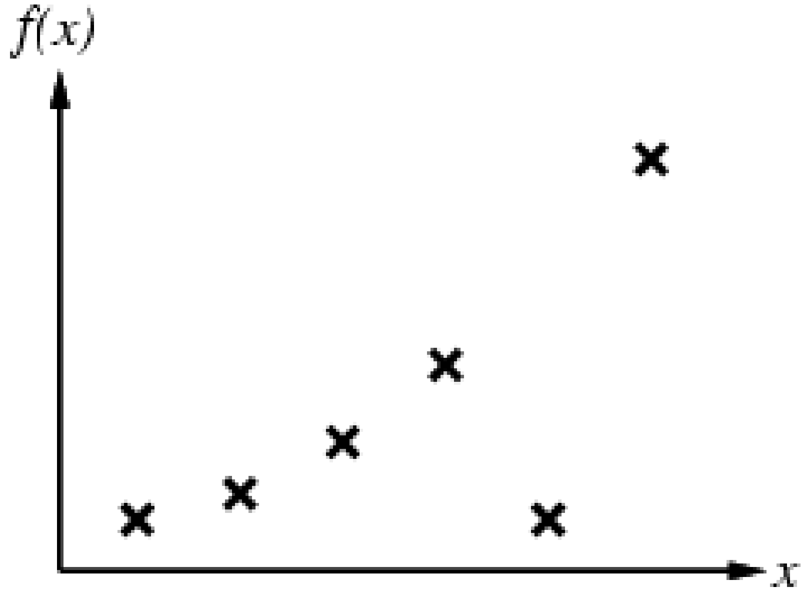
Problem: find a hypothesis h

such that $h \approx f$

given a training set of examples

(This is a highly simplified model of real learning:

- Ignores prior knowledge
- Assumes examples are given)



INDUCTIVE LEARNING METHOD

- Construct/adjust *h* to agree with *f* on training set

(*h* is consistent if it agrees with *f* on all examples)

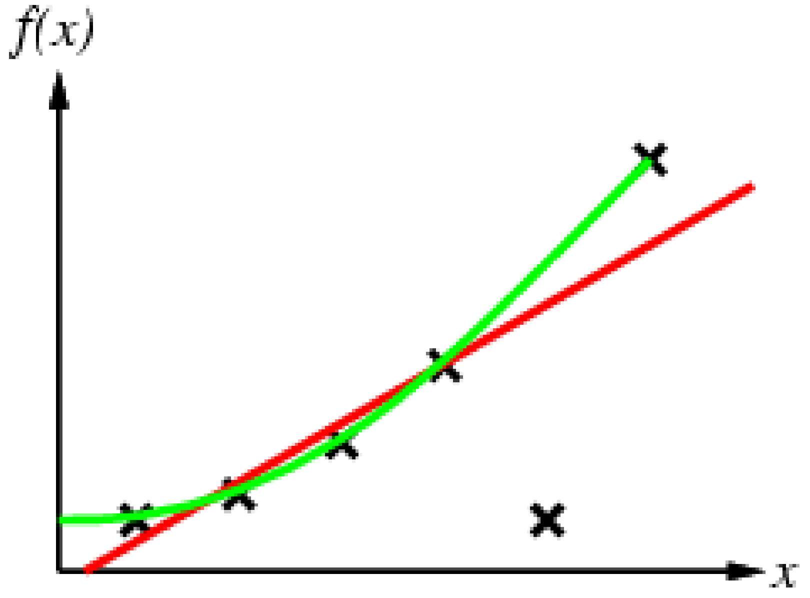
e.g., curve fitting:

INDUCTIVE LEARNING METHOD

- Construct/adjust *h* to agree with *f* on training set

(*h* is consistent if it agrees with *f* on all examples)

e.g., curve fitting:

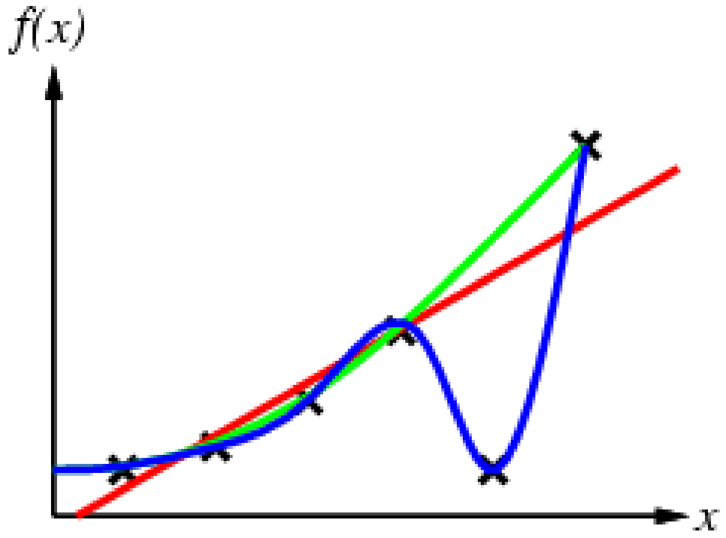


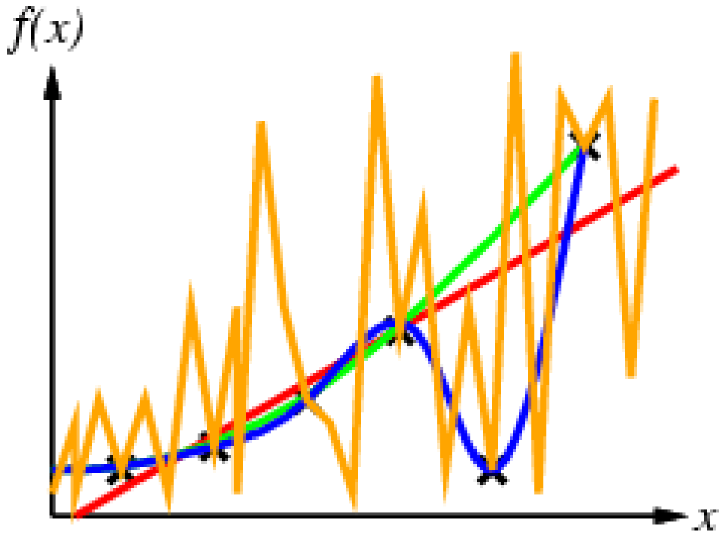
INDUCTIVE LEARNING METHOD

- Construct/adjust *h* to agree with *f* on training set

(*h* is consistent if it agrees with *f* on all examples)

e.g., curve fitting:



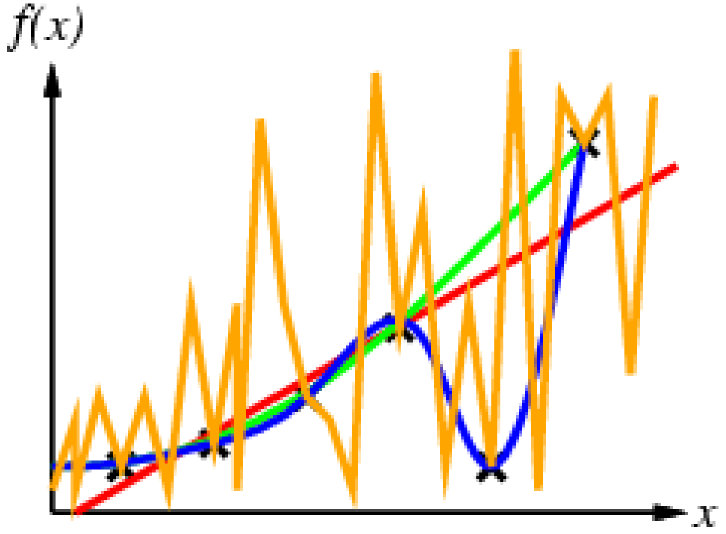


INDUCTIVE LEARNING METHOD

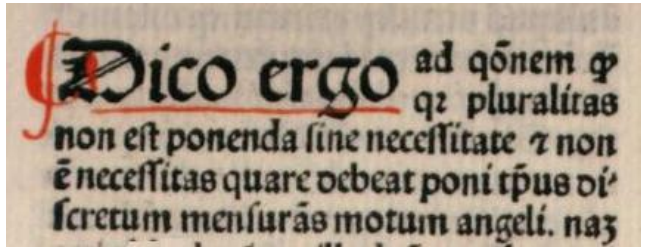
- Construct/adjust *h* to agree with *f* on training set

(*h* is consistent if it agrees with *f* on all examples)

E.g., curve fitting:



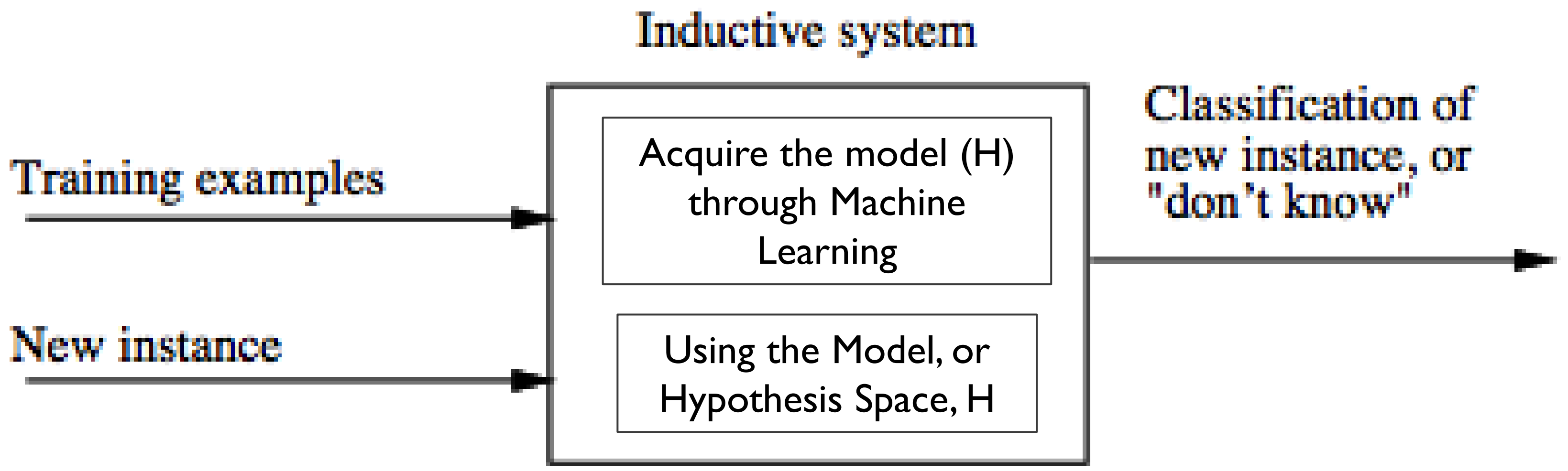
novacula Occami



Dico ergo ad qõnem ꝙ qꝛ pluralitas non eſt ponenda ſine neceſſitate ⁊ non ē neceſſitas quare debeat poni tpus diſcretum menſurãs motum angeli. naꝫ

Ockham’s razor: *prefer the simplest hypothesis consistent with data*

INDUCTIVE SYSTEM


Inductive system
Training examples
New instance
Acquire the model (H) through Machine Learning
Using the Model, or Hypothesis Space, H
Classification of new instance, or "don't know"

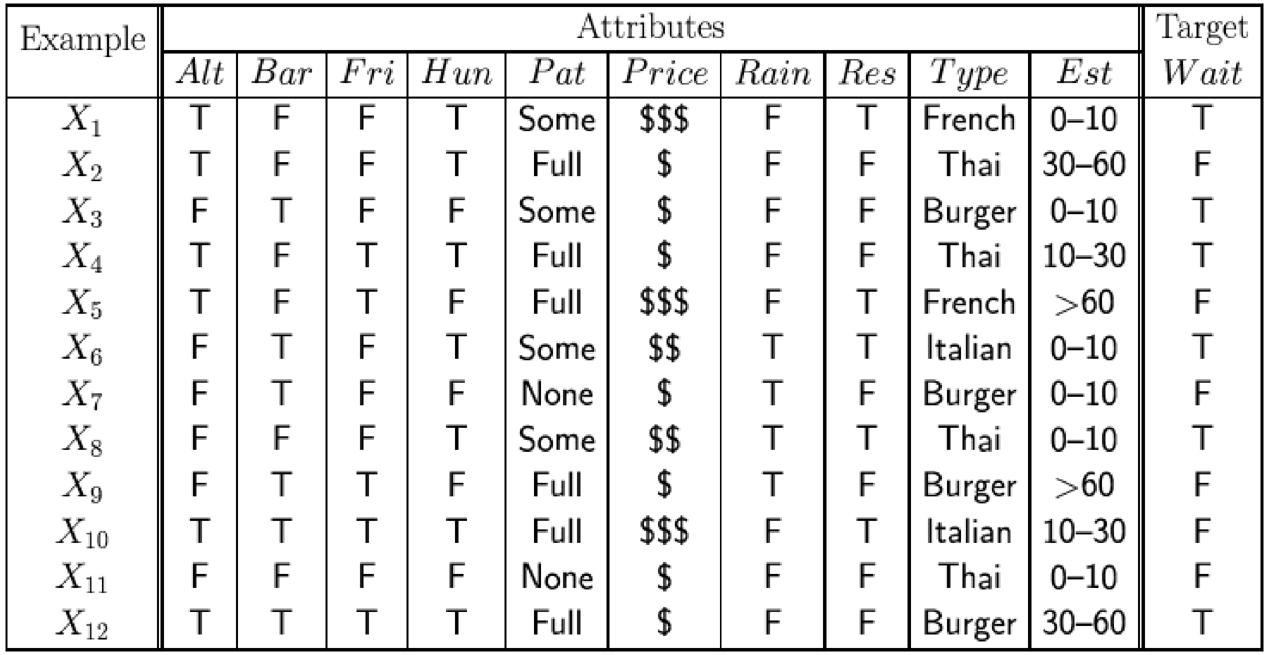
LEARNING DECISION TREES

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

ATTRIBUTE-BASED REPRESENTATIONS

- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

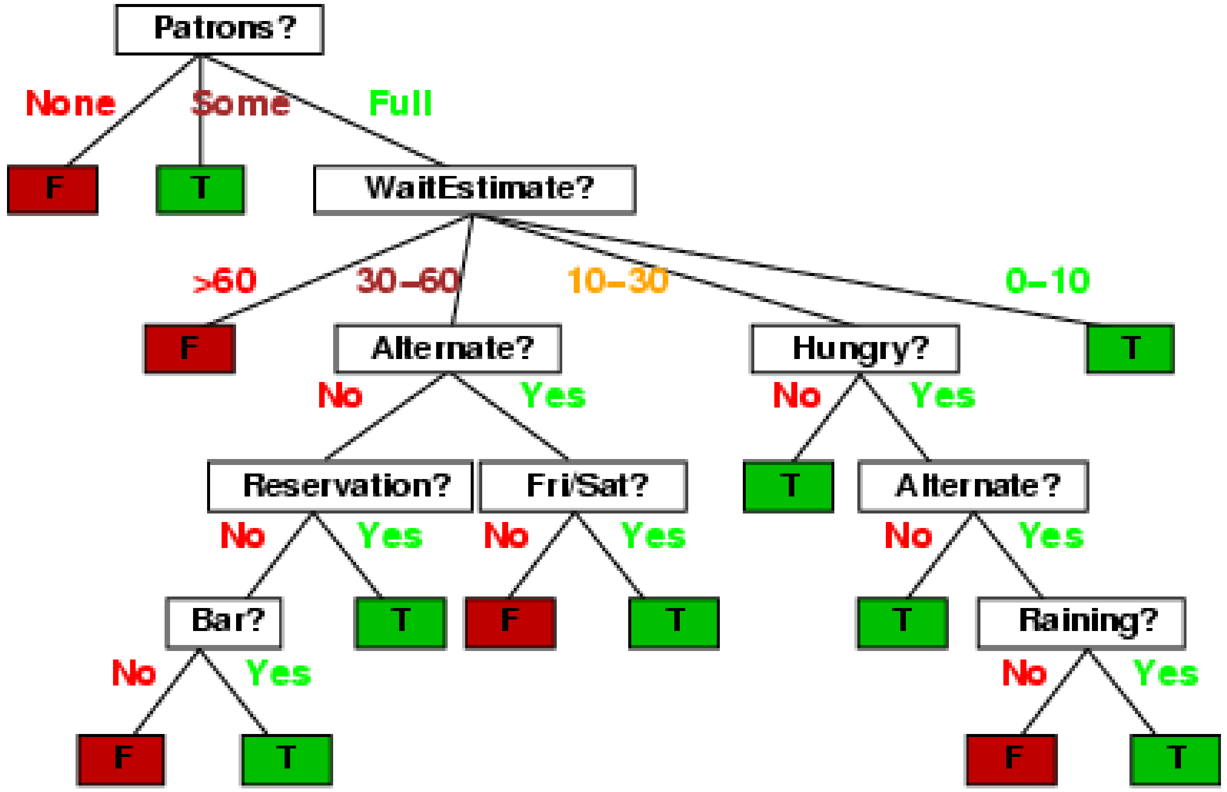


Example	Attributes										Target
	Alt	Bar	Fri	Hun	Pat	$Price$	$Rain$	Res	$Type$	Est	$Wait$
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

- Classification of examples is positive (T) or negative (F)

DECISION TREES

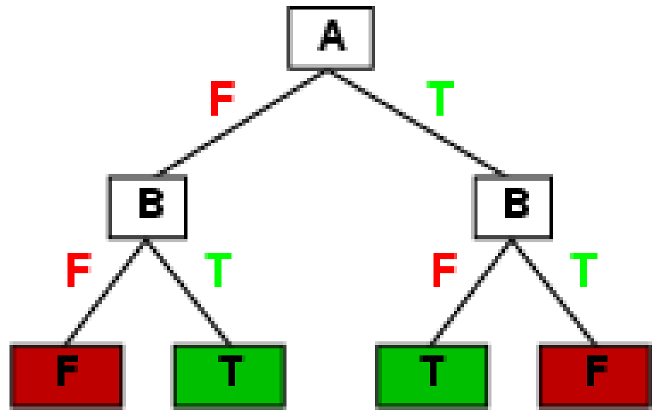
- One possible representation for hypotheses
- E.g., here is the "true" tree for deciding whether to wait:



EXPRESSIVENESS

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples
- Prefer to find more compact decision trees

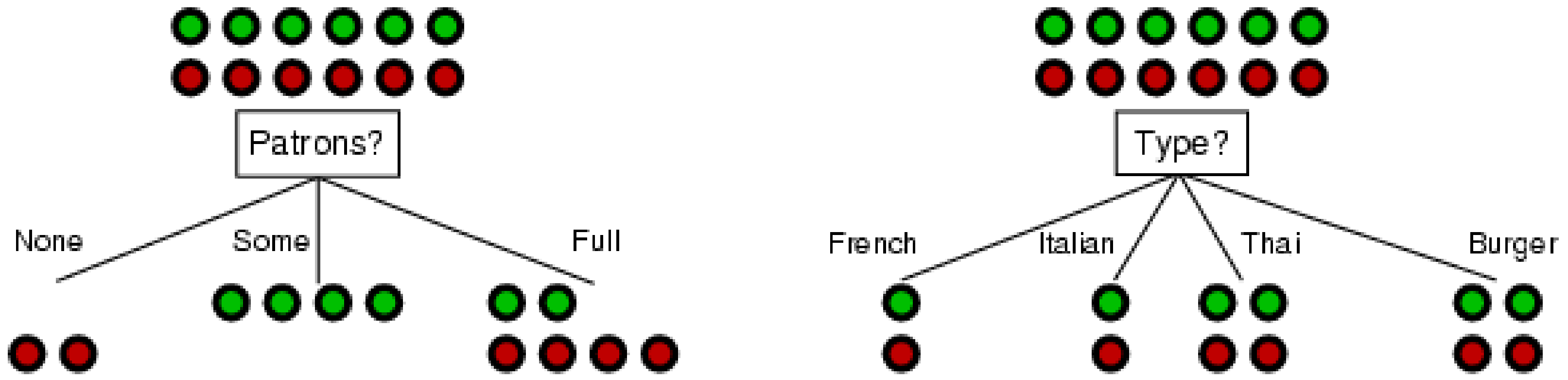
DECISION TREE LEARNING

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL($examples$, $attributes$, $default$) returns a decision tree
    if $examples$ is empty then return $default$
    else if all $examples$ have the same classification then return the classification
    else if $attributes$ is empty then return MODE($examples$)
    else
        $best \leftarrow$ CHOOSE-ATTRIBUTE($attributes$, $examples$)
        $tree \leftarrow$ a new decision tree with root test $best$
        for each value $v_i$ of $best$ do
            $examples_i \leftarrow$ {elements of $examples$ with $best = v_i$}
            $subtree \leftarrow$ DTL($examples_i$, $attributes - best$, MODE($examples$))
            add a branch to $tree$ with label $v_i$ and subtree $subtree$
        return $tree$
```

CHOOSING AN ATTRIBUTE

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



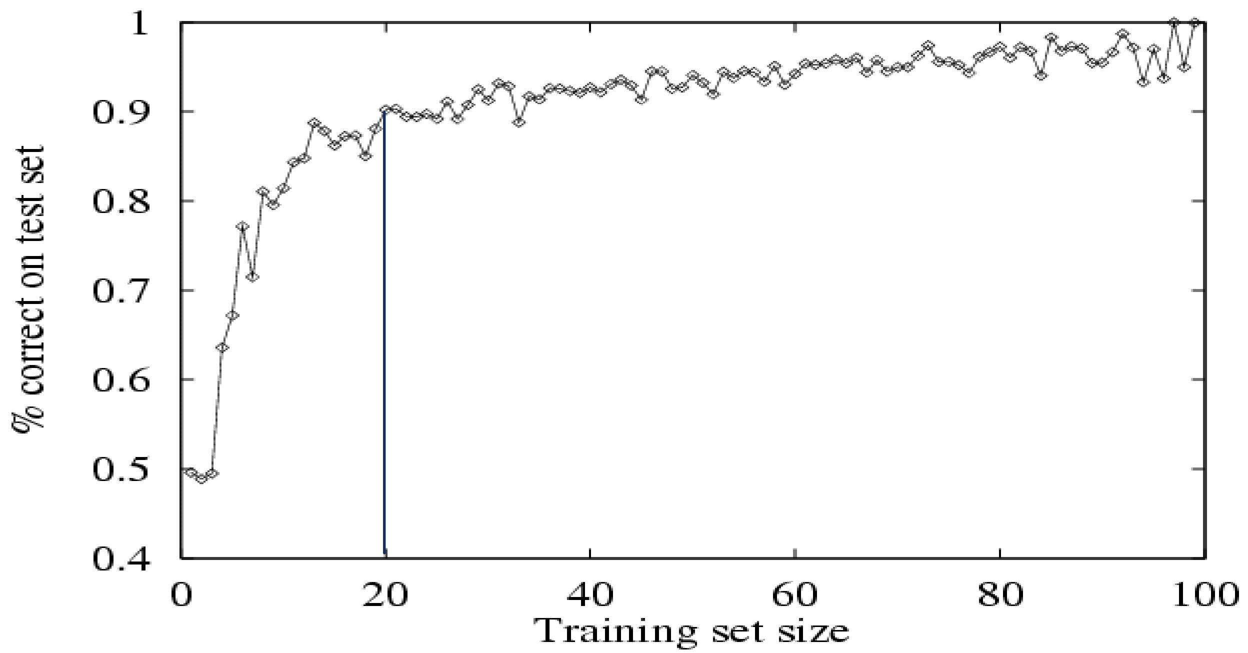
- *Patrons?* is a better choice

PERFORMANCE MEASUREMENT

- How do we know that $h \approx f$?
  1. Use theorems of computational/statistical learning theory
  2. Try h on a new test set of examples

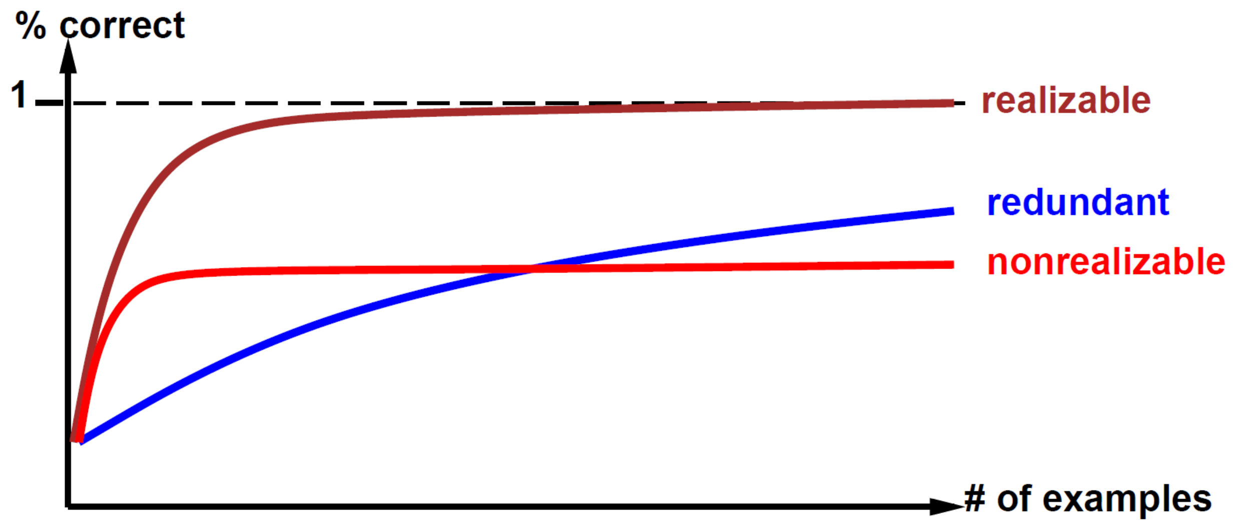
     (use same distribution over example space as training set)

Learning curve = % correct on test set as a function of training set size

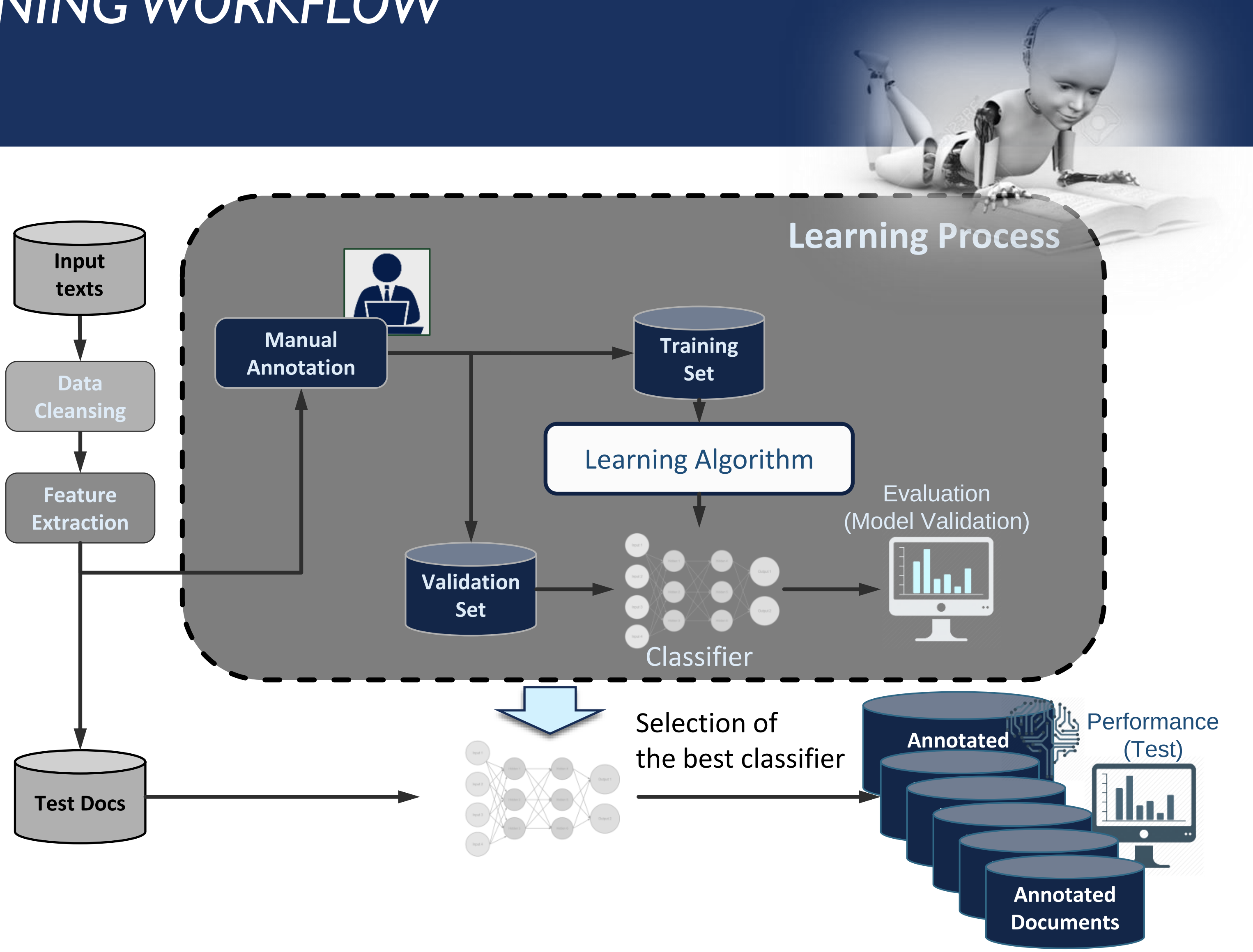


PERFORMANCE MEASUREMENTS (2)

- **Learnability** depends on
  - **realizable** kind of performances vs.
  - … **non-realizable** ones
  - **Non-realizability** depends on
    - Missing attributes
    - Limitation on the hypothesis space (e.g. non expressive functions)
  - **Redundant expressiveness** is related to cases where a a largenumber of irrelevant attributes are used

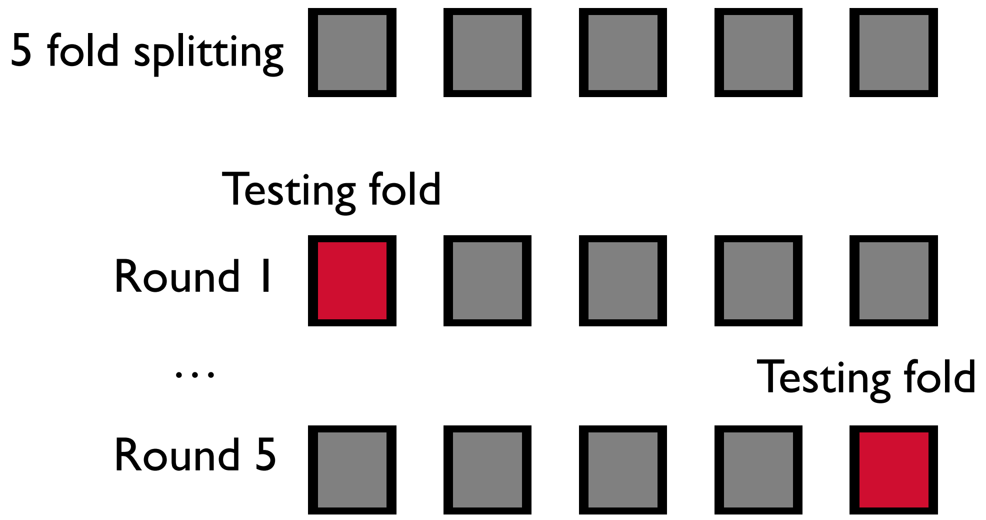


MACHINE LEARNING WORKFLOW



N-FOLD CROSS VALIDATION

- Data is split into *n* subsets of equal size
- Each subset in turn is used for testing and the remainders *n-1* for training
- The metrics estimated in each round are averaged



MACHINE LEARNING TASKS

- **SUPERVISED LEARNING DA ESEMPI**
  - **CLASSIFICATION**
    - Approcci dicriminativi
    - Approcci generative
    - Outlier and deviation detection
  - **REGRESSION**
  - **Dependency modeling**
    - Discovery di Associazioni/Relazioni, Sommari, Inferenza/Causalità
  - **SEQUENCE CLASSIFICATION**
    - Temporal learning
    - Trend analysis and change/anomaly detection

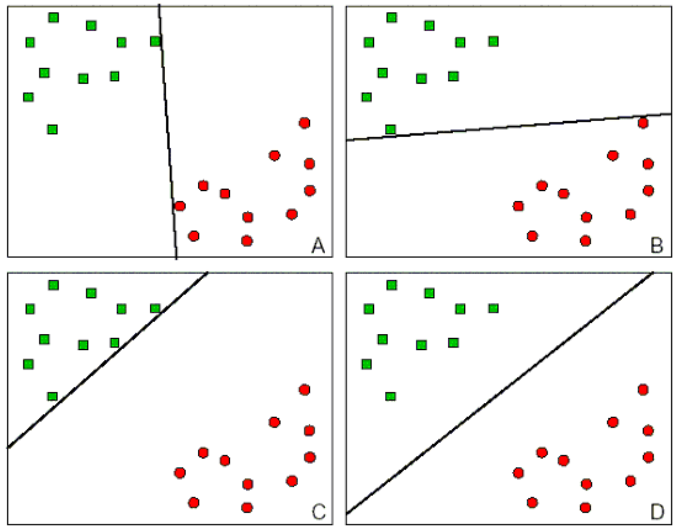
- **UNSUPERVISED LEARNING**
  - Clustering
  - Embedding ottimo: Enconding/Decoding
    - Representation Learning for Images
    - PreTraining as optimal encoding

- **REINFORCEMENT LEARNING**
  - Penalty/Reward function from the Environment
  - Autonomous Systems
  - Hard for complex problems

METODI DI ML: SELEZIONE DEI MODELLI

- Approcci discriminativi
  - Lineari

$$h(x) = sign(\,\mathbf{W} \cdot \mathbf{x} + b)$$

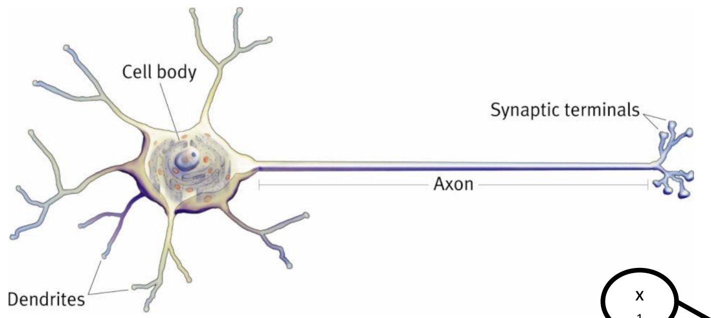


- Approcci probabilistici
  - Stima delle probabilità $p(\mathcal{C}_k|\mathbf{x})$ attraverso un training set
  - Modello generativo ed uso della inversione Bayesiana

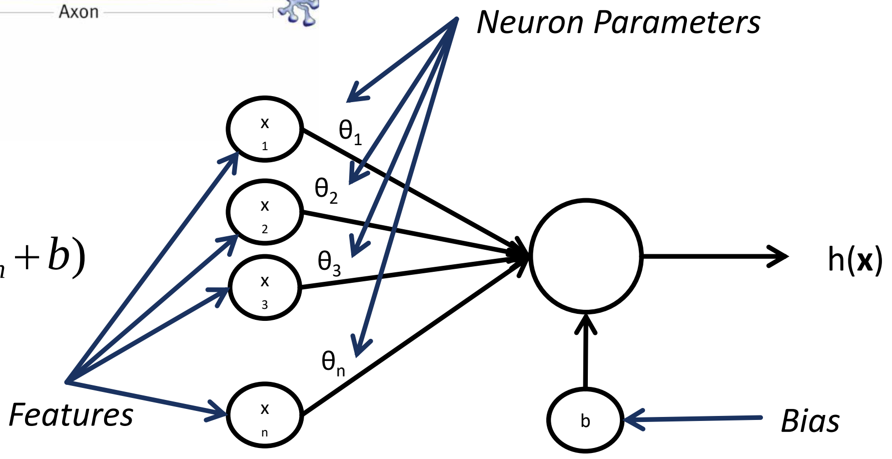
$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}.$$

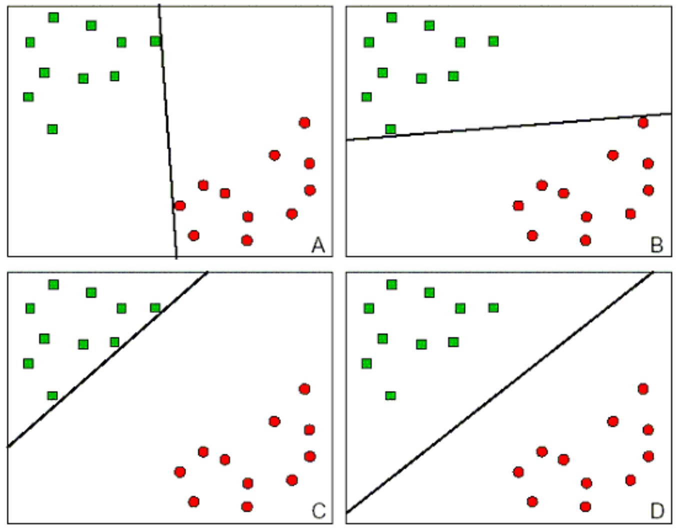
PERCEPTRON (ROSENBLATT, 1958)

- Linear Classifier mimicking a neuron

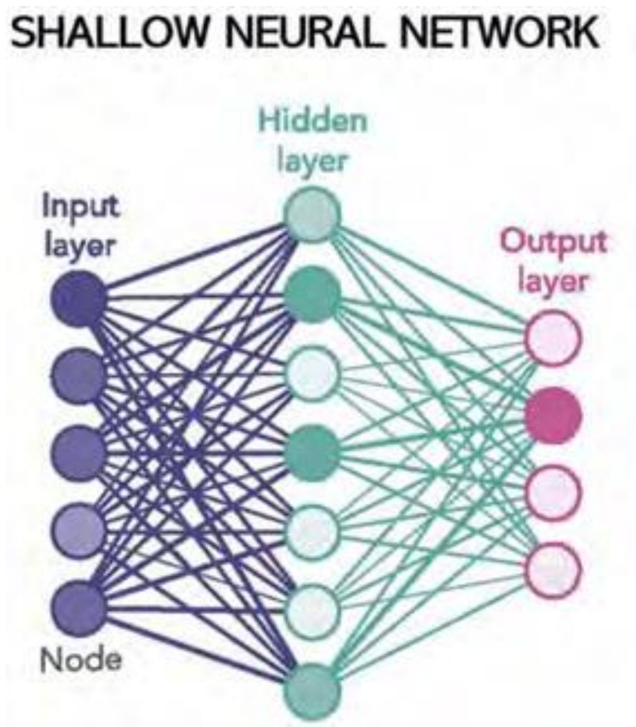


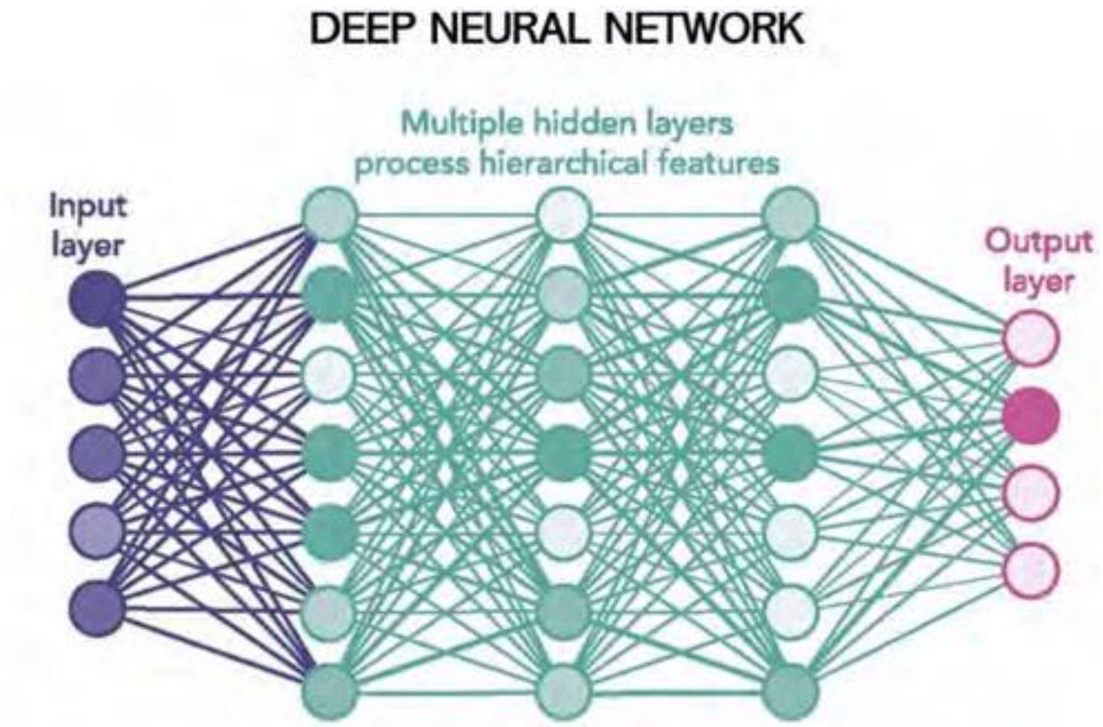
$$h(\vec{x}) = g(\sum_n \theta_n x_n + b)$$



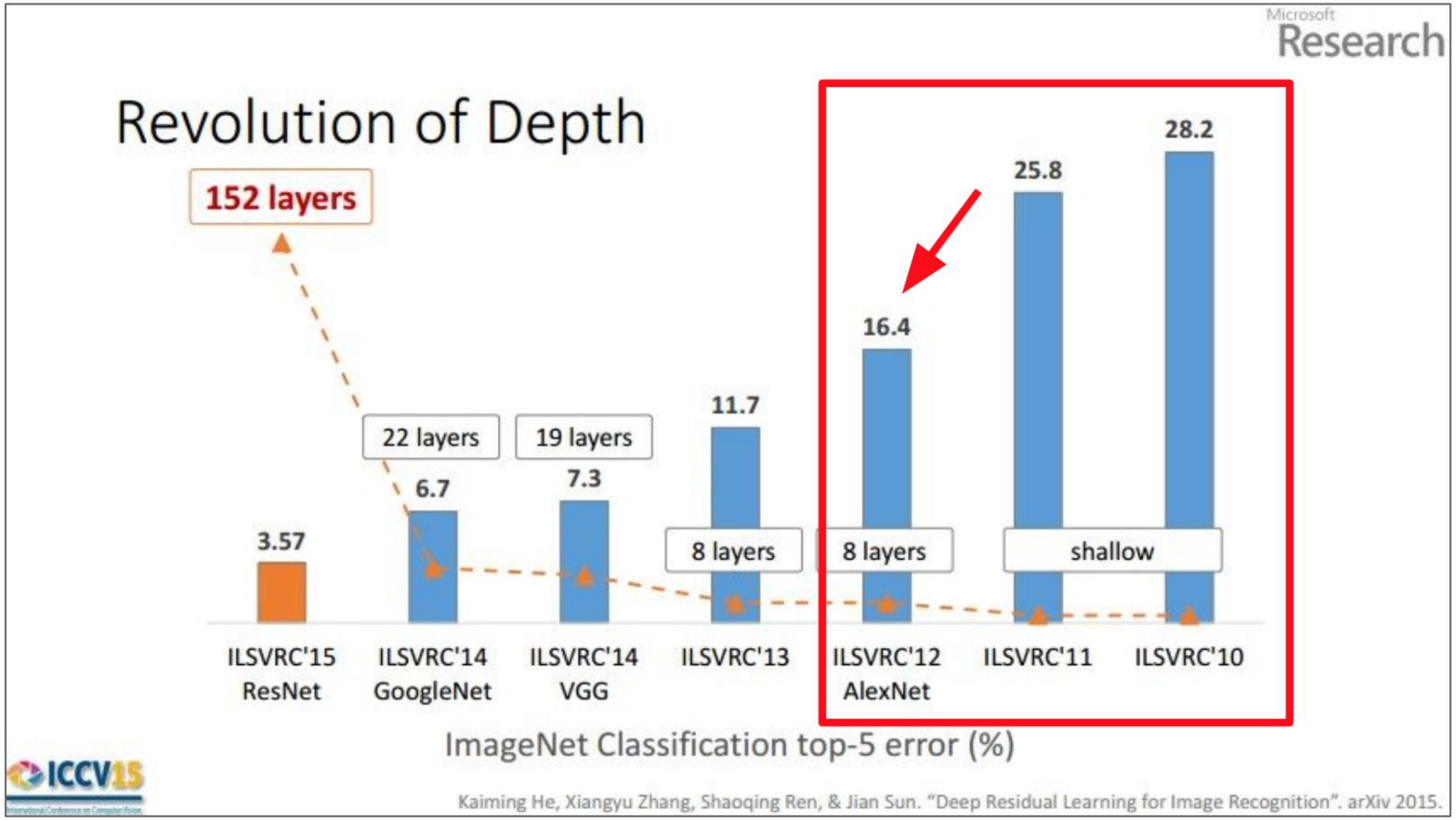


ADDING LAYERS …



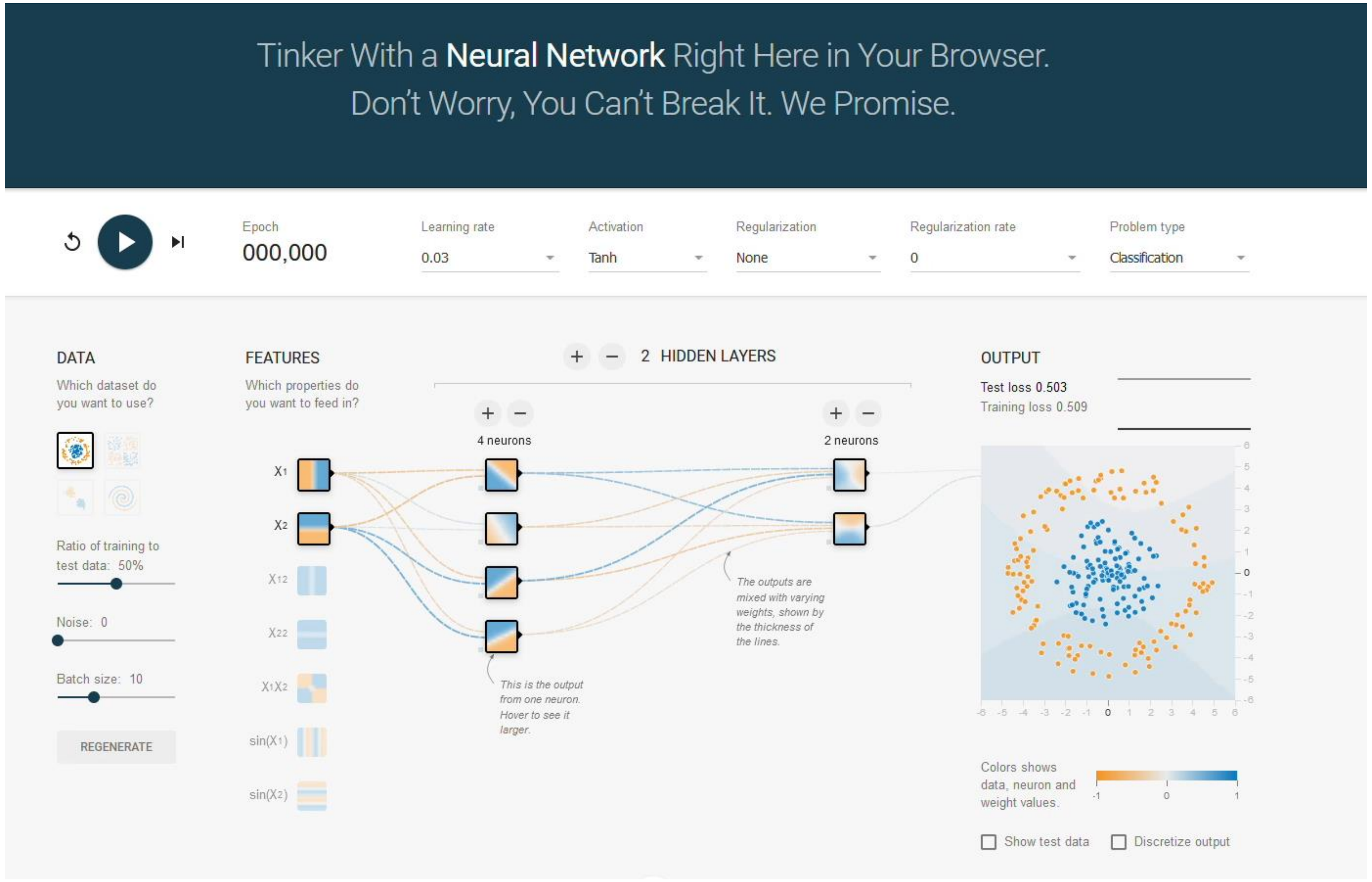


RETI NEURALI PROFONDE



A SIMPLE DEMO ON TENSORFLOW

- Look at: https://playground.tensorflow.org/



Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Epoch 000,000
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification

DATA
Which dataset do you want to use?

Ratio of training to test data: 50%
Noise: 0
Batch size: 10

FEATURES
Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons
2 neurons

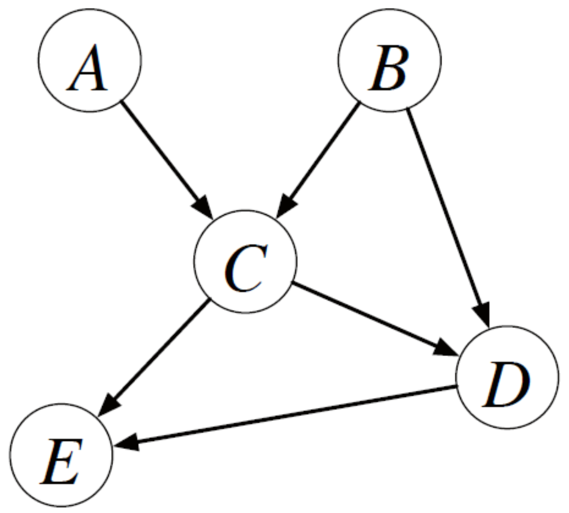
This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

OUTPUT
Test loss 0.503
Training loss 0.509

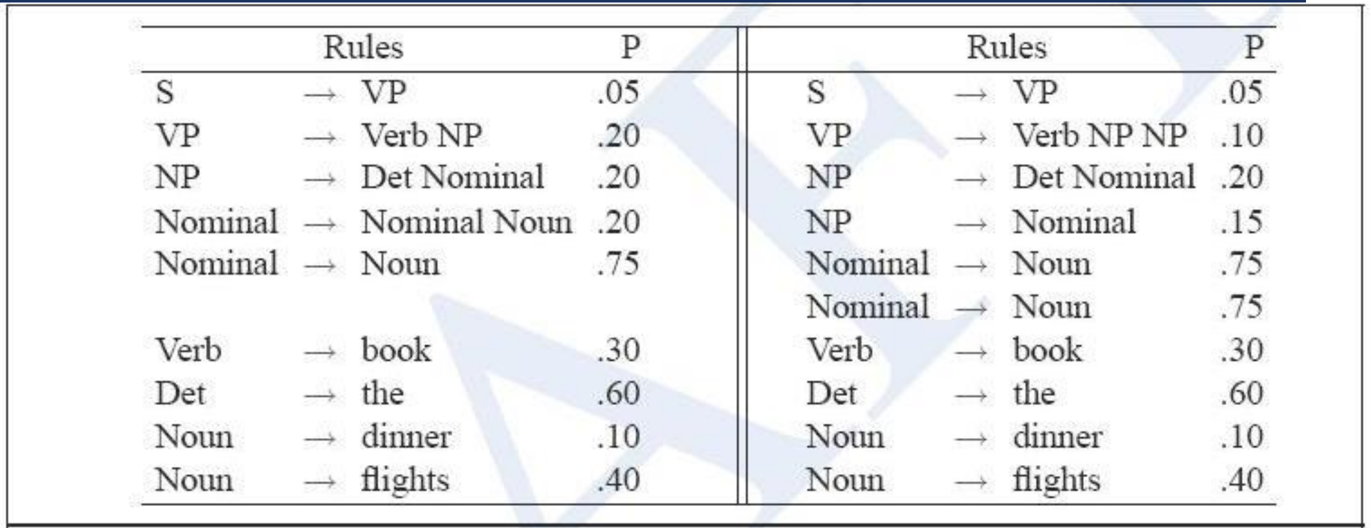
Colors shows data, neuron and weight values.

ALTERNATIVE: MODELLI (BAYESIANI) GRAFICI



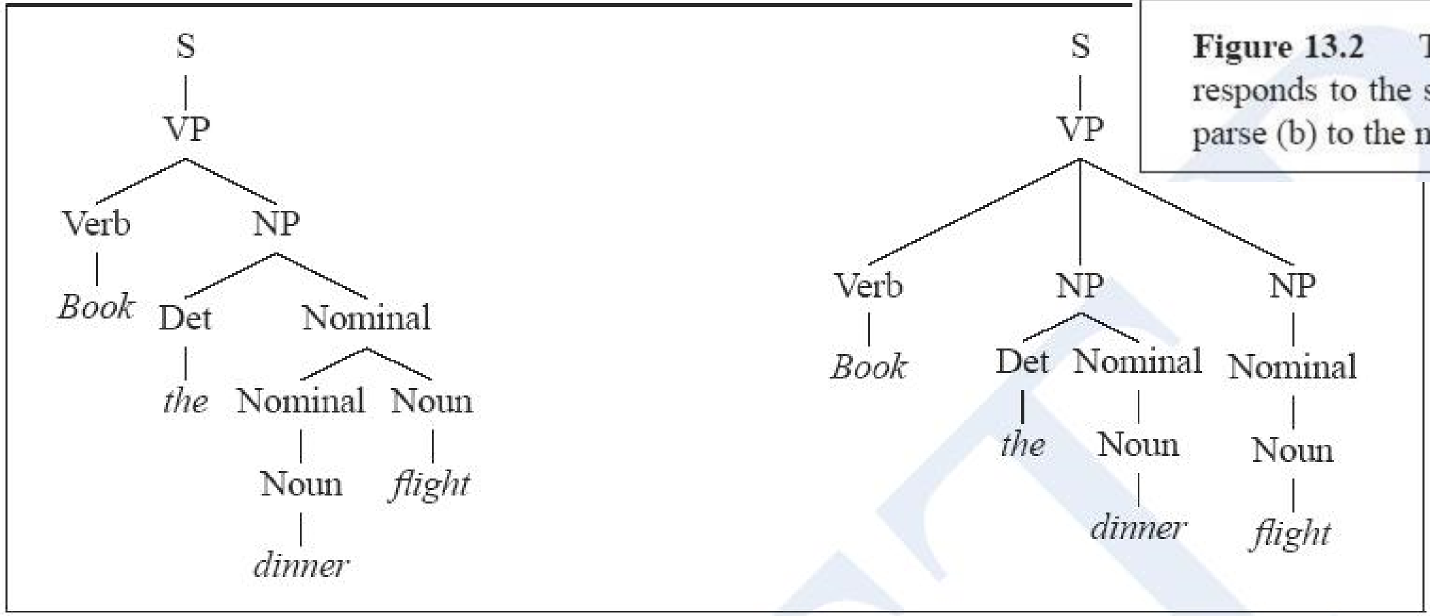
$$p(A,B,C,D,E) = p(A)p(B)p(C|A,B)p(D|B,C)p(E|C,D)$$

GRAMMATICHE PROBABILISTICHE: TRA SINTASSI & STATISTICA

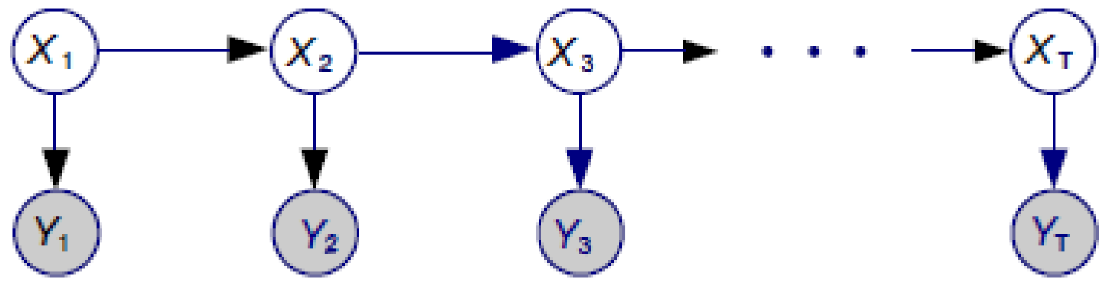


Rules			P	Rules			P
S	→	VP	.05	S	→	VP	.05
VP	→	Verb NP	.20	VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20	NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20	NP	→	Nominal	.15
Nominal	→	Noun	.75	Nominal	→	Noun	.75
				Nominal	→	Noun	.75
Verb	→	book	.30	Verb	→	book	.30
Det	→	the	.60	Det	→	the	.60
Noun	→	dinner	.10	Noun	→	dinner	.10
Noun	→	flights	.40	Noun	→	flights	.40

Figure 13.2 Two parse trees for an ambiguous sentence, The transitive parse (a) corresponds to the sensible meaning "Book flights that serve dinner", while the ditransitive parse (b) to the nonsensical meaning "Book flights on behalf of 'the dinner'?".



APPRENDIMENTO SU SEQUENZE: *HIDDEN MARKOV MODELS*



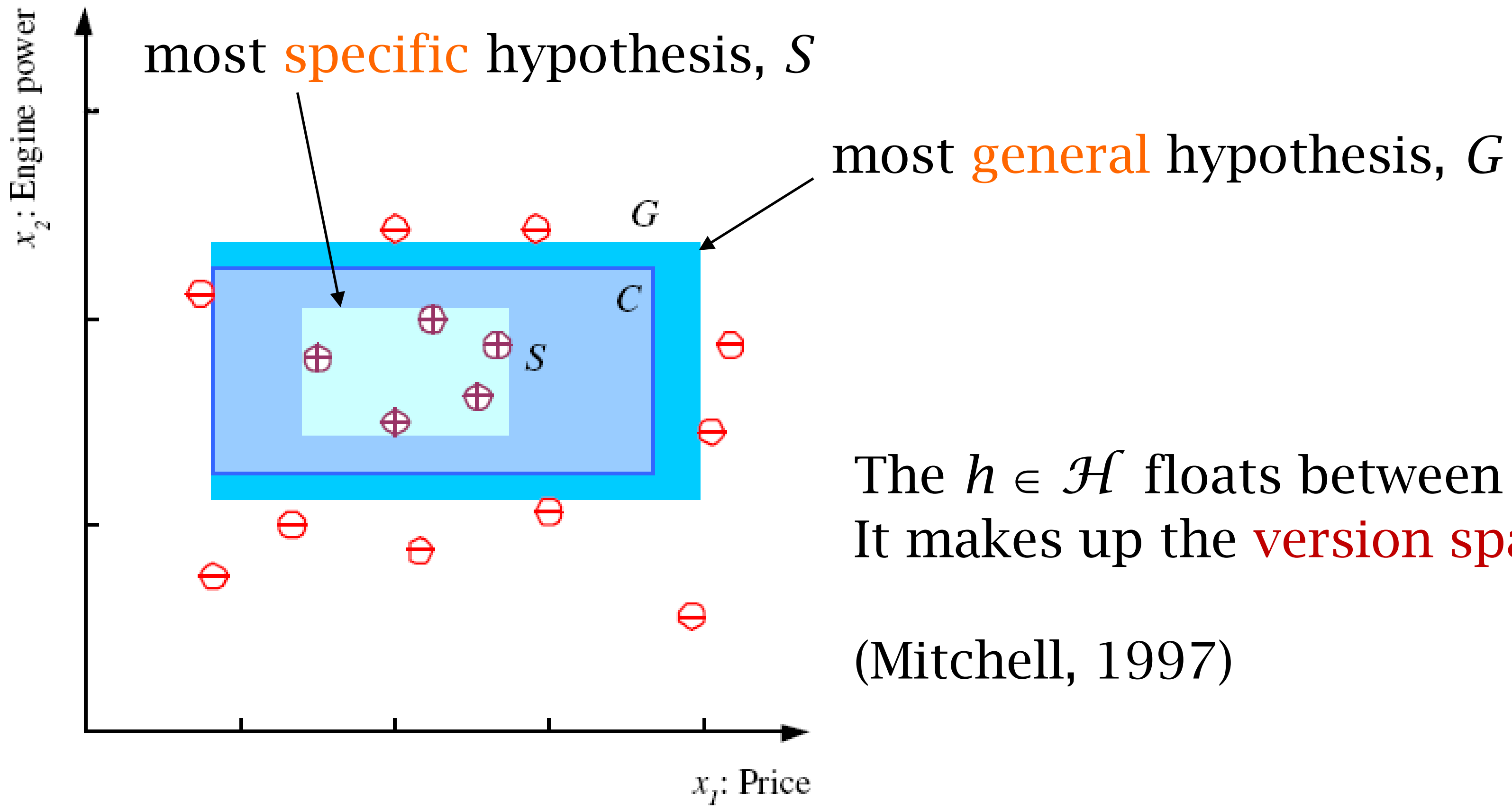
$$p(X_{1,\dots,T}, Y_{1,\dots,T}) = p(X_1)p(Y_1|X_1)\prod_{t=2}^{T}[p(X_t|X_{t-1})p(Y_t|X_t)]$$

- Stati (*X*) = Categorie/Concetti/Proprietà
- Osservazioni (*Y*): simboli di un certo linguaggio
- Emissioni vs. Transizioni
- Applicazioni:
  - Speech Recognition (Simboli: fonemi, Stati: punti di segmentazione)
  - Part-of-Speech (POS) tagging (Simboli: parole, Stati: categorie gramaticali)

STATISTICAL LEARNING THEORY

DALLA PAC LEARNABILITY AI PERCETTRONI

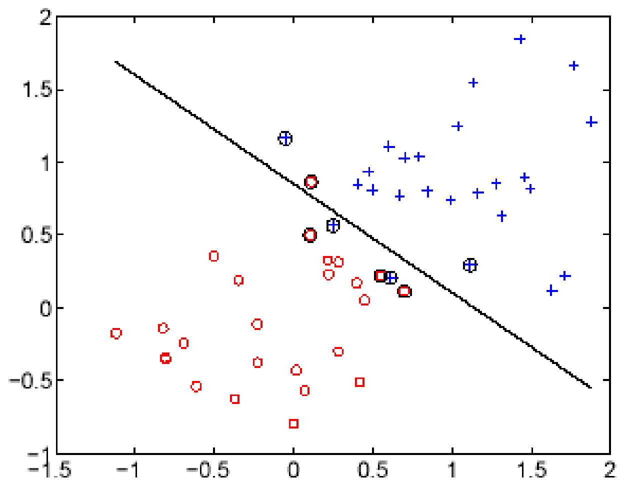
(VECTOR) SPACES, FUNCTIONS AND LEARNING

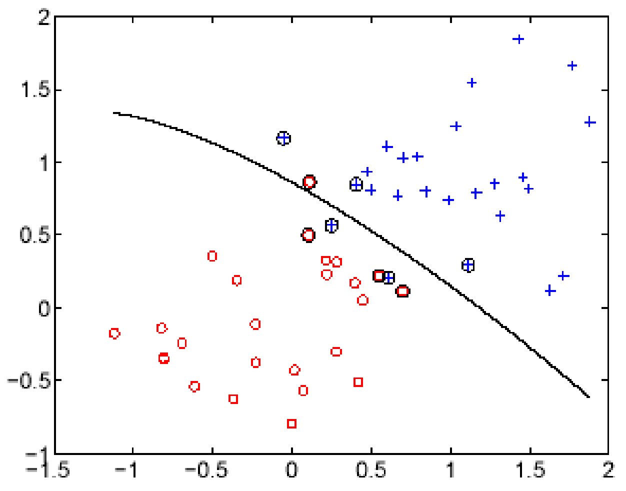


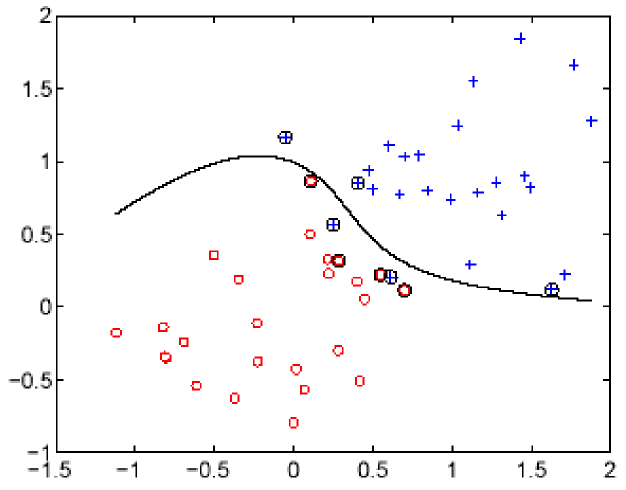
The $h \in \mathcal{H}$ floats between S and G to be consistent
It makes up the version space

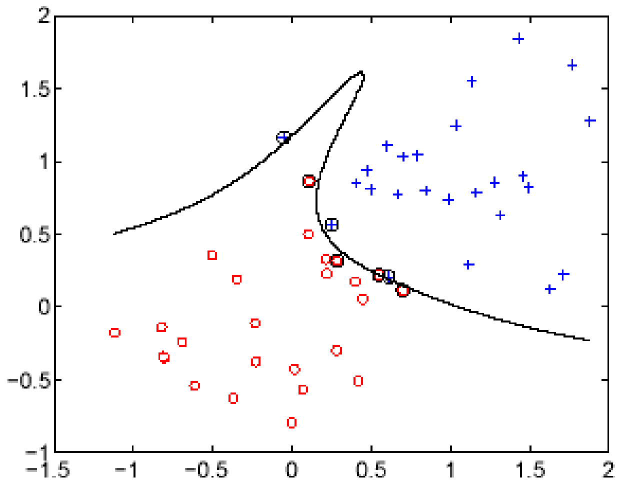
(Mitchell, 1997)

Structural risk minimization: example


linear


2^{nd} order polynomial


4^{th} order polynomial


8^{th} order polynomial

$$y = f^*(\vec{x})$$

$$f^*(\vec{x}) \approx h(\vec{x}) = g(\vec{x};\overrightarrow{\theta})$$

$$such\,that\ \forall \vec{x}_l \in \mathsf{L} \quad h(\vec{x}_l) \approx y_l$$

PROBABLY APPROXİMATELY CORRECT (PAC) LEARNİNG

- How many training examples are needed so that the tightest rectangle S which will constitute our hypothesis, will probably be approximately correct?
  - We want to be *confident* (*above a level*) that $1\text{-}\delta$
  - … the *error probability is bounded* by some value ε

- A concept class C is called ***PAC-learnable*** if there exists a PAC-learning algorithm such that, for any $\varepsilon>0$ and $\delta>0$, there exists a fixed sample size such that, for any concept $c \in C$ and for any probability distribution on X, the learning algorithm produces a probably-approximately-correct hypothesis h
- a (PAC) *probably-approximately-correct hypothesis* h is one that has error at most ε with probability at least $1\text{-}\delta$.

PROBABLY APPROXİMATELY CORRECT (PAC) LEARNİNG

- In PAC learning, given a class *C* and examples drawn from some unknown but fixed distribution $p(x)$, we want to find the number of examples *N*, such that with **probability at least** $1-\delta$, h has **error at most** ε ? (Blumer et al., 1989)

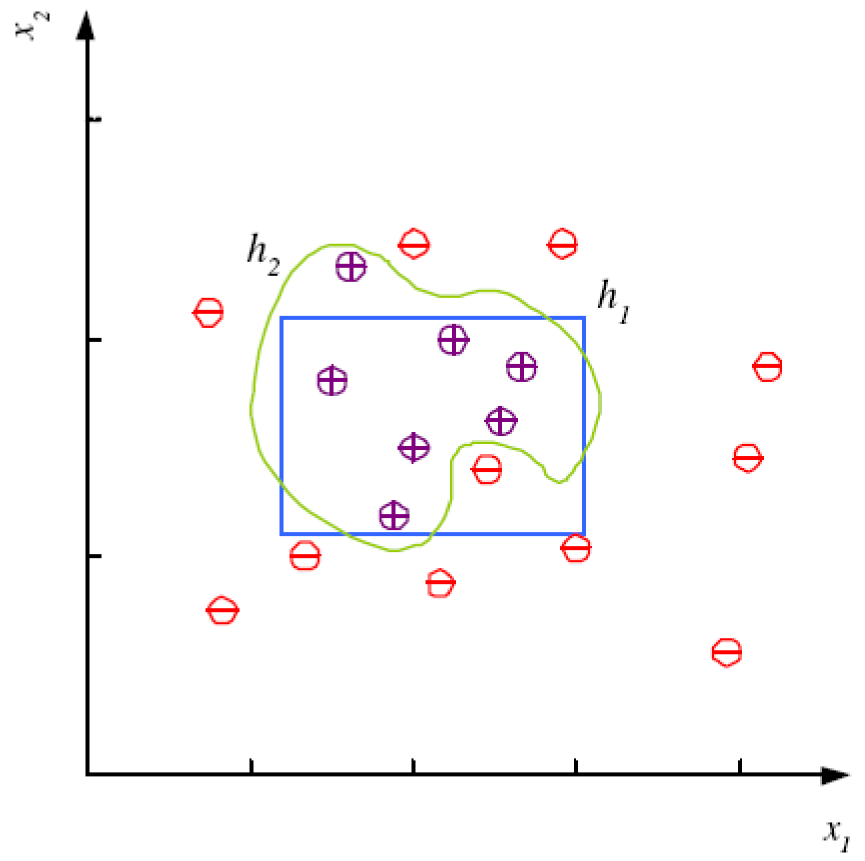
$$P(\,C\Delta h \leq \varepsilon\,) \geq 1\text{-}d$$

where $C\Delta h$ is (the area of) "the region of difference between C and h", and $\delta>0$, $\varepsilon>0$.

MODEL COMPLEXİTY VS. NOISE

Use the simpler one because

- Simpler to use (lower computational complexity)
- Easier to train (lower space complexity)
- Easier to explain (more interpretable)
- Generalizes better (Occam's razor)



MODEL SELECTİON & GENERALİZATİON

- Learning is an ill-posed problem; data is not sufficient to find a unique solution
- The need for inductive bias, assumptions about $\mathcal{H}$
- **Generalization:** How well a model performs on new data
- Different machines have different amounts of "power".

  Tradeoff between:
  - More power: Can model more complex classifiers but might overfit.
  - Less power: Not going to overfit, but restricted in what it can model.
  - **Overfitting**: $\mathcal{H}$ more complex than C or f
  - **Underfitting**: $\mathcal{H}$ less complex than C or f

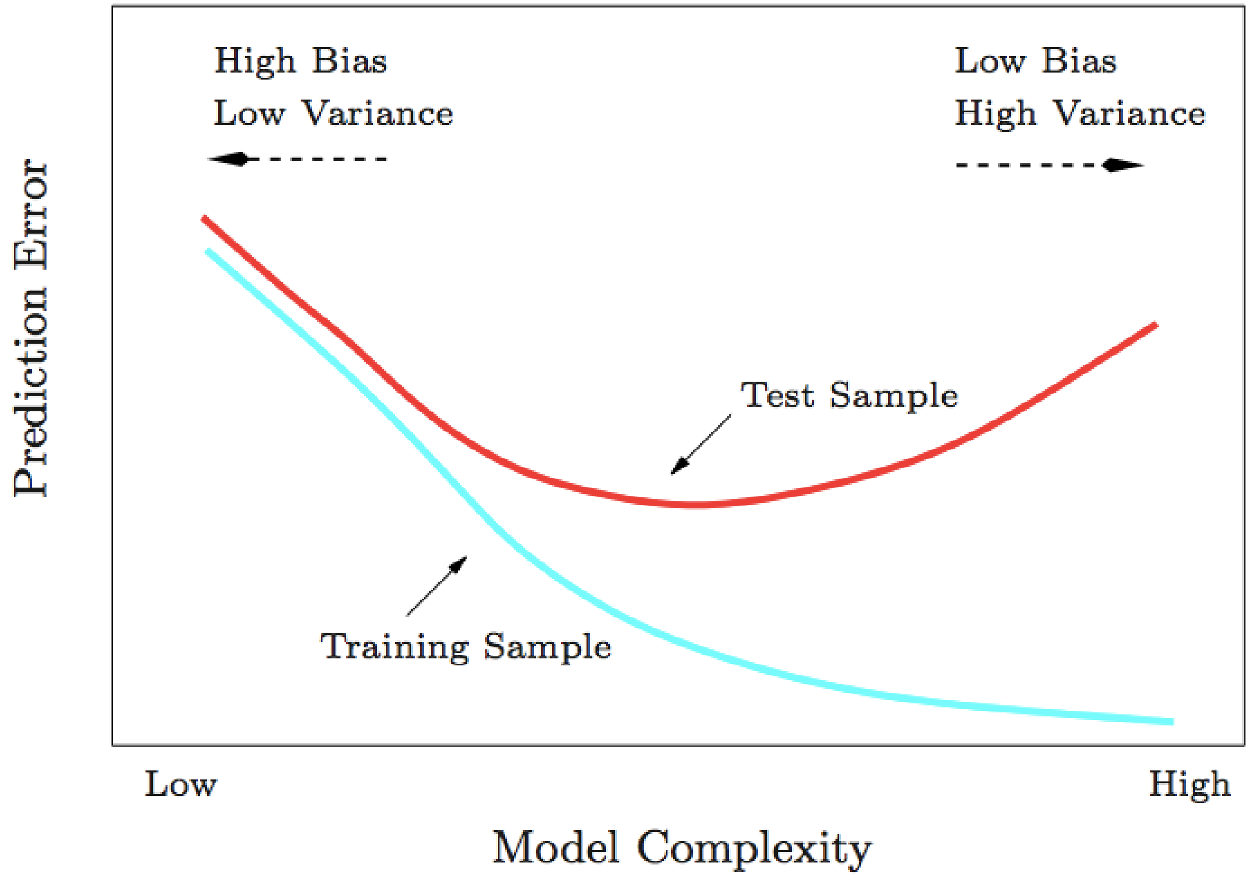
MACHINE LEARNING: IN SEARCH OF GOOD FUNCTIONS

- Model and Learning

$$y = f^*(\vec{x})$$
$$f^*(\vec{x}) \approx h(\vec{x}) = g(\vec{x};\overrightarrow{\theta})$$
$$such\ that\ \ \forall \vec{x}_l \in \mathfrak{L}\ \ h(\vec{x}_l) \approx y_l$$

- Linear models

$$h(\vec{x}) = g(\sum \theta_n x_n + b)$$



TRIPLE TRADE-OFF

- There is a trade-off between three factors (Dietterich, 2003):
  1. Complexity of $\mathcal{H}$, $c(\mathcal{H})$,
  2. Training set size, N,
  3. Generalization error, E, on new data

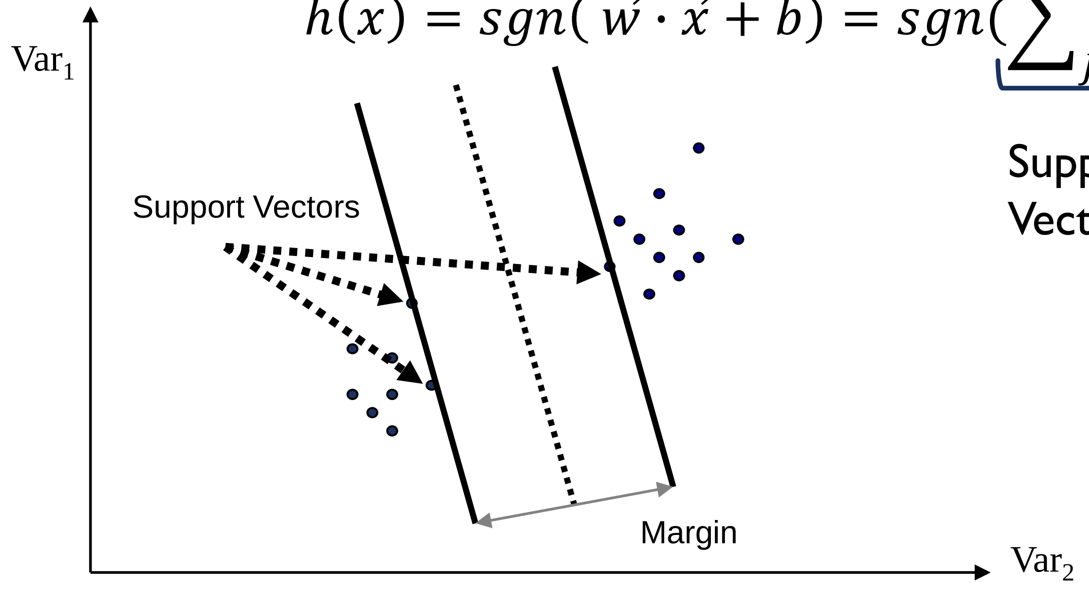
- As $N \uparrow$, $E \downarrow$
- As $c(\mathcal{H}) \uparrow$, first $E \downarrow$ and then $E \uparrow$

SUPPORT VECTOR MACHINES

- Support Vector Machines (SVMs) are a machine learning paradigm based on the statistical learning theory [Vapnik, 1995]
  - No need to remember everything, just the discriminating instances (i.e. the support vectors, SV)
  - The classifier corresponds to the linear combination of SVs

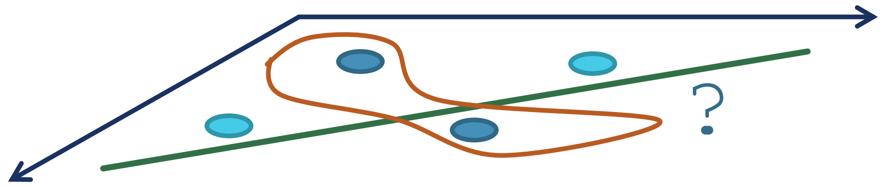
$$h(x) = sgn(\vec{w} \cdot \vec{x} + b) = sgn(\underbrace{\sum_{J=1}^{l} \alpha_j y_j \vec{x_j}}_{\text{Support Vectors}} \boxed{\cdot\, \vec{x}} + b)$$

Only the dot product is required



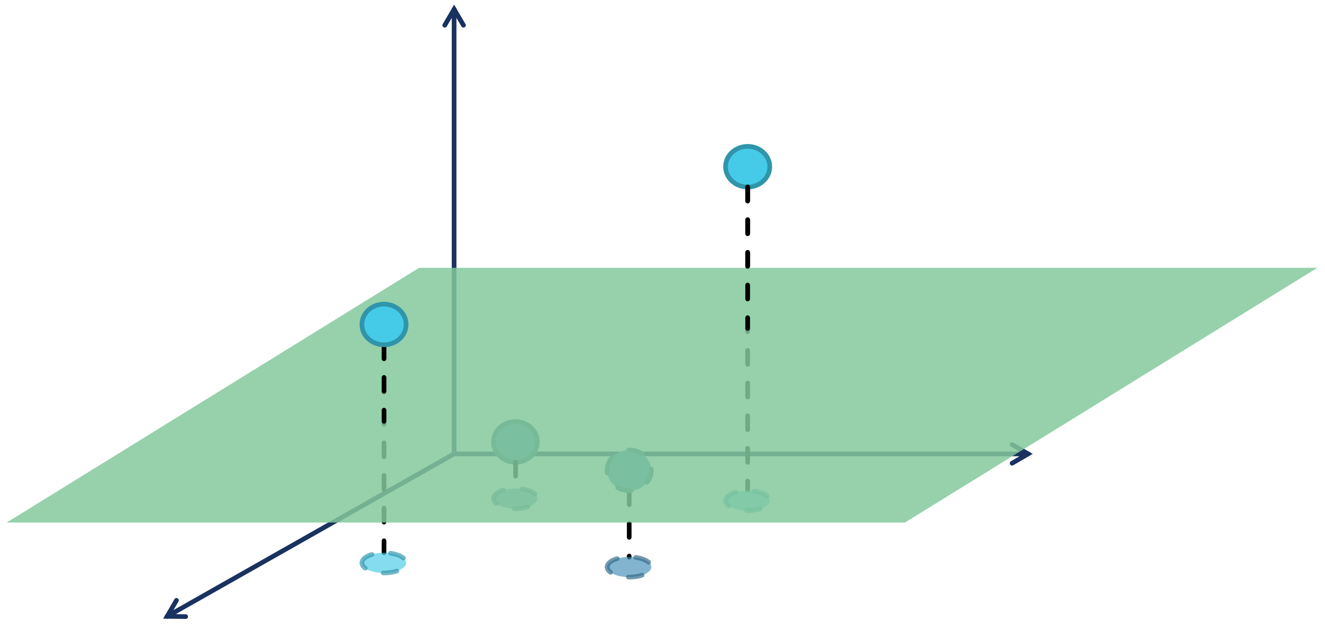
LINEAR CLASSIFIERS AND SEPARABILITY

- In a R^2 space, 3 point can always be separable by a linear classifier
  - but 4 points cannot always be shattered [Vapnik and Chervonenkis(1971)]
- One solution could be a more complex classifier
  - Risk of over-fitting



LINEAR CLASSIFIERS AND SEPARABILITY (2)

- … but things change when projecting instances in a higher dimension feature space through a function ϕ
- IDEA: It is better to have a more complex feature space instead of a more complex function

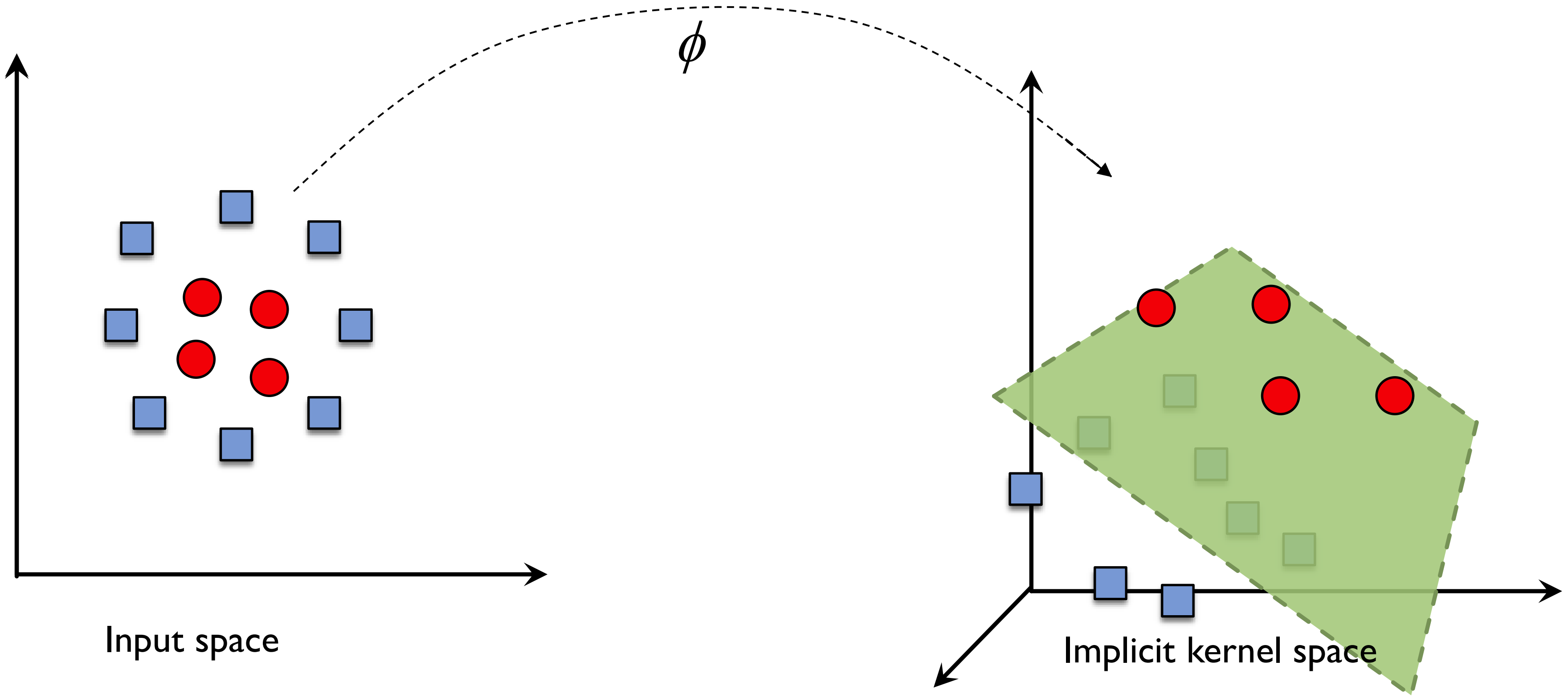


SVM FIRST ADVANTAGE: THE KERNEL TRICK MAKING EXAMPLES LINEARLY SEPARABLE

- Mapping data in a (richer) feature space where linear separability holds

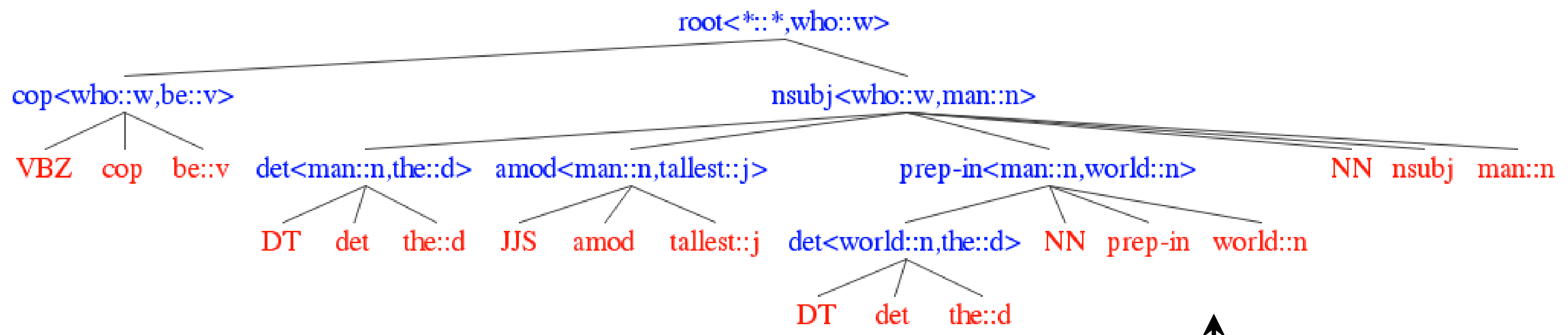
$\vec{x} \rightarrow \Phi(\vec{x})$

(attributes $\longrightarrow$ features)

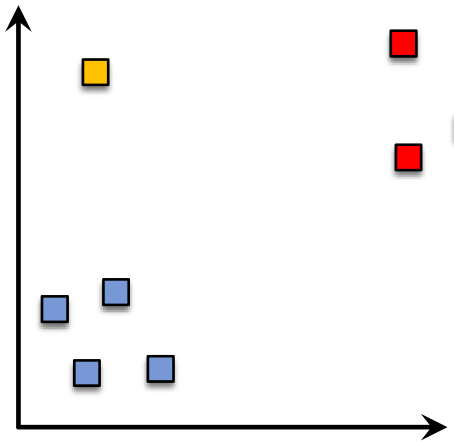


KERNELS AS … EMBEDDING TOOLS: AN NLP EXAMPLE

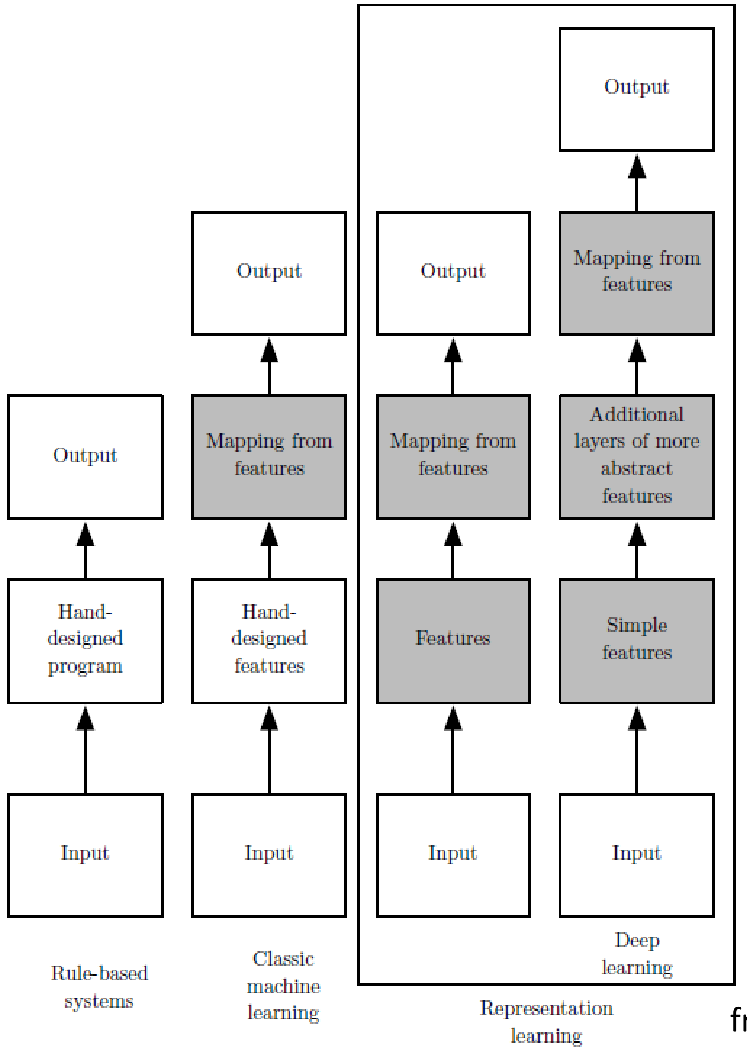
- Semantic Tree Kernels allows generating vectors that reflect syntactic/semantic information of sentences
  - *Who is the tallest man in the world ?*



- What are the most similar trees/vectors/sentences?
  - *Who is the richest woman in the world ?*
  - *Who is the richest person in the world ?*
  - *Who is the fastest swimmer in the world ?*
  - *Who was murdered yesterday by the terrorist group?*
  - ….



DIFFERENT VIEWS ON LEARNING



from Goodfellow et al., Deep Learning MIT book

RETI NEURALI

PERCETTRONI E *MULTILAYER PERCEPTRONS*

NN HISTORY

from (Wang&Raj, 2017):

Wang, Haohan; Raj, Bhiksha,
On the Origin of Deep Learning,

https://arxiv.org/abs/1702.07800 , Feb2017

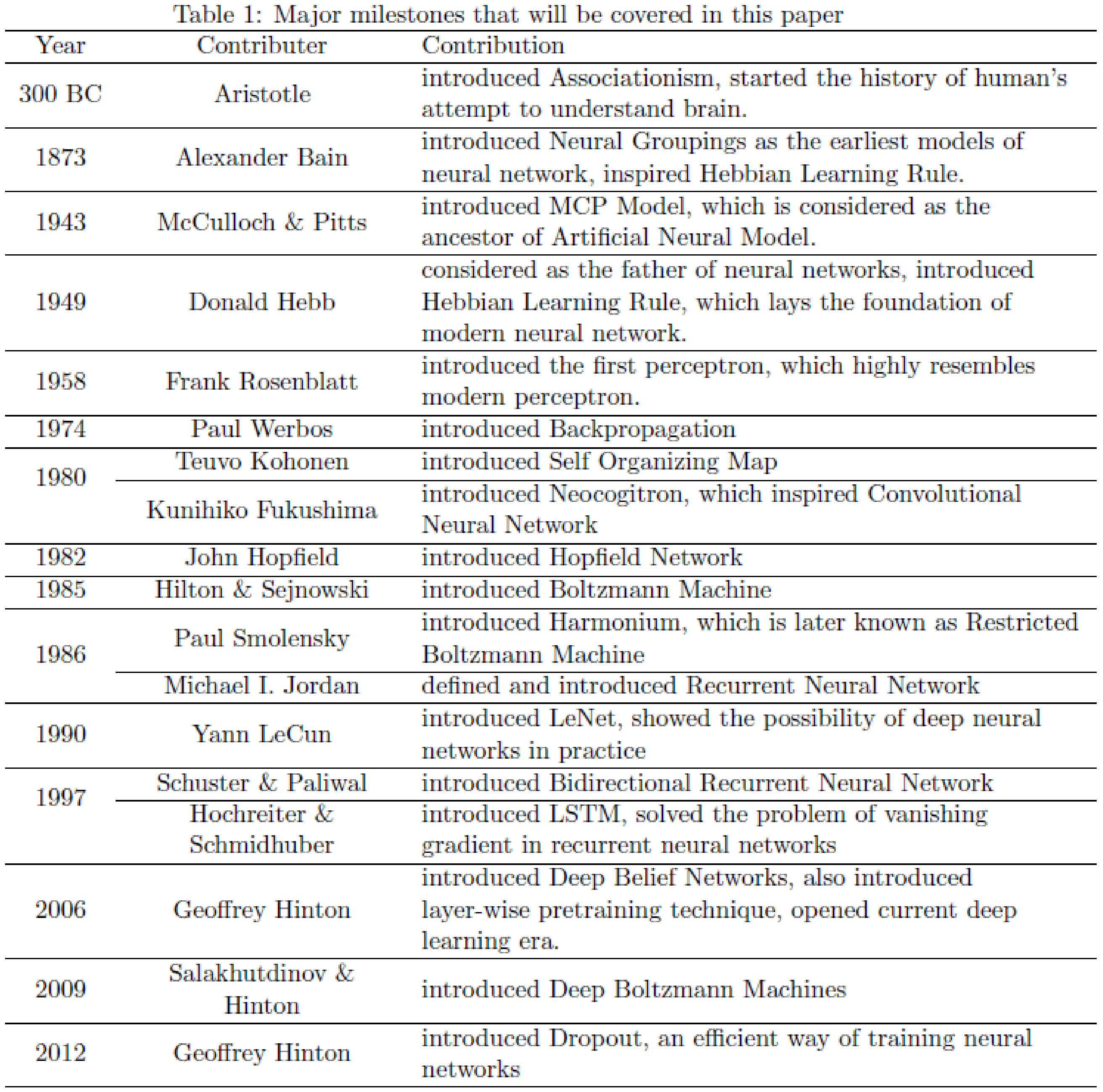


Table 1: Major milestones that will be covered in this paper

Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks

Demystifying neural networks

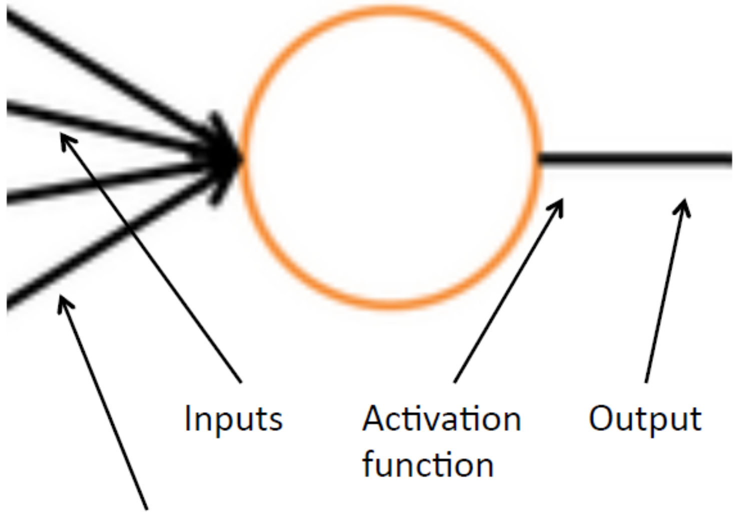
Neural networks come with their own terminological baggage

… just like SVMs

But if you understand how logistic regression or maxent models work

Then **you already understand** the operation of a basic neural network neuron!

A single neuron
A computational unit with *n* (3) inputs
and 1 output
and parameters *W, b*

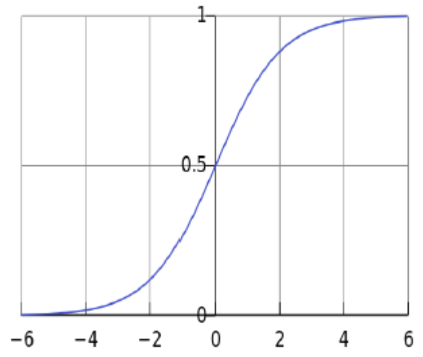


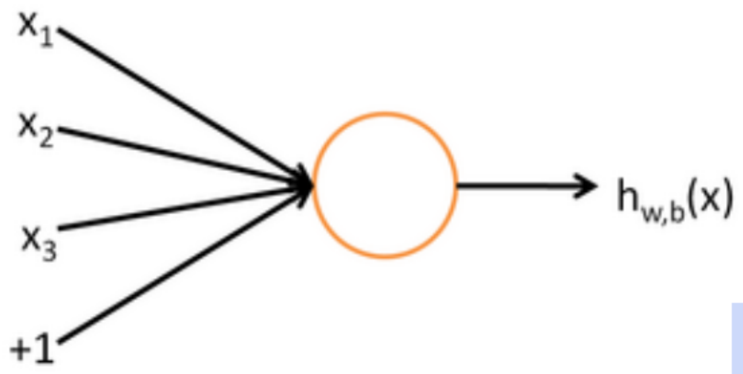
Bias unit corresponds to intercept term

$$h_{w,b}(x) = f(w^{\top}x + b)$$

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

b: We can have an "always on" feature, which gives a class prior, or separate it out, as a bias term



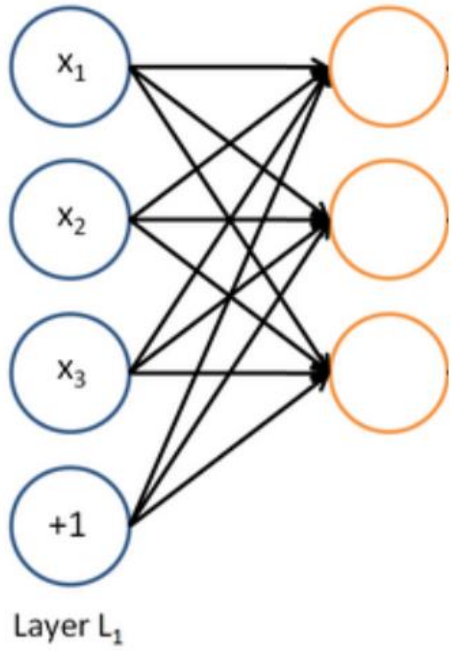


w, *b* are the parameters of this neuron i.e., this logistic regression model

NEURAL NETWORKS: THE IBASIC IDEA

A neural network = running several logistic regressions at the same time

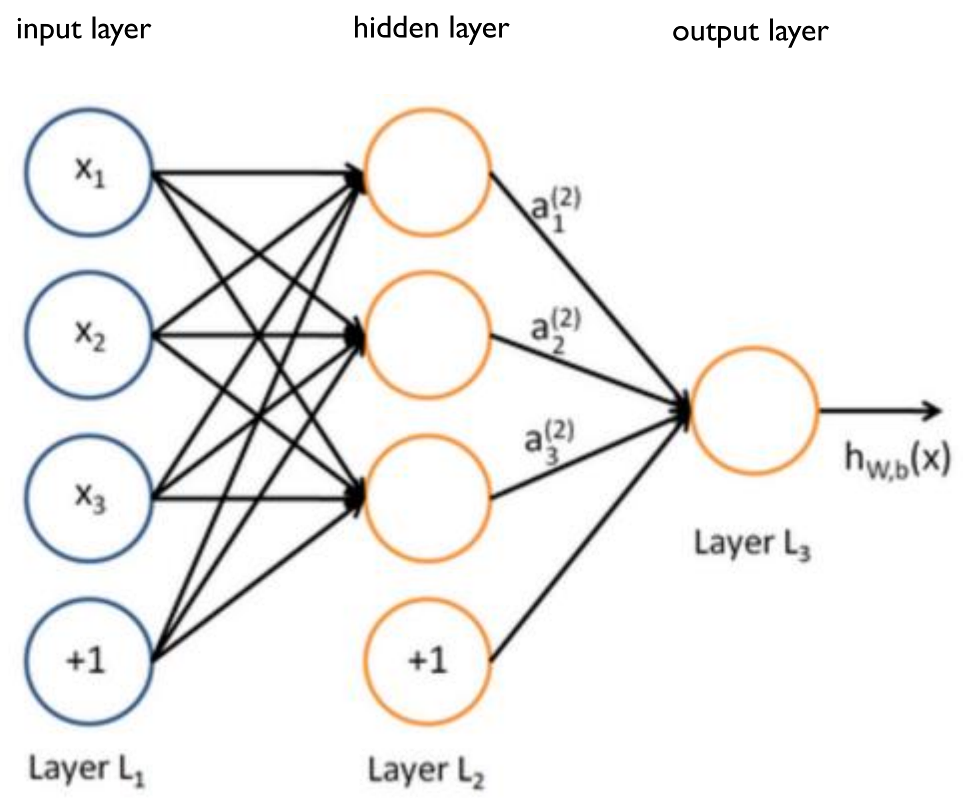
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …



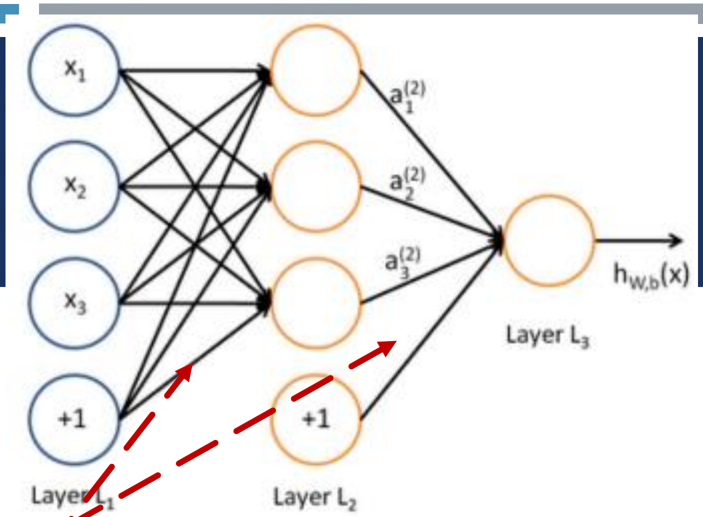
But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

NEURAL NETWORKS

- Each circle represent a **neuron** (or unit)
  - 3 **input**, 3 **hidden** and 1 **output**
- n_l=3 is the number of layers
- s_l denotes the number of units in layer l
- Layers:
  - Layer l is denoted as L_l
  - Layer l and l+1 are connected by a matrix $W^{(l)}$ of parameters
    - $W^{(l)}_{i,j}$ connects neuron j in layer l with neuron i in layer $l+1$
- $b^{(l)}_i$ is the *bias* associated to neuron I in layer l+1



ADDING LAYERS …



- From simple linear laws …

$$h(\vec{x}) = g(\vec{x};\vec{\theta},b) = g(\sum_n \theta_n x_n + b)$$

- to feedforward structures. It can be made dependent on a sequence of functions $g^{(l)}$ and $g^{(2)}$, …, $g^{(k)}$ that give rise to a structured hypothesis:

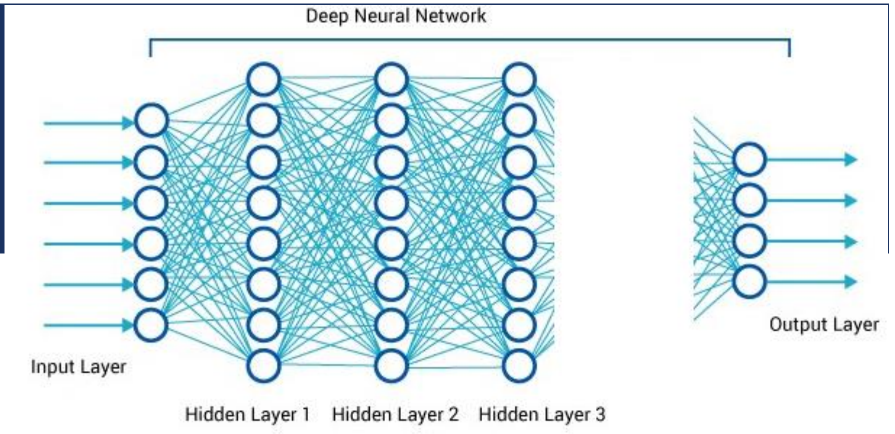
$$h(\vec{x}) = g^{(2)}(g^{(1)}(\vec{x};\vec{\theta}^{(1)},b^{(1)});\vec{\theta}^{(2)},b^{(2)}) =$$

$$= W^{(2)} g^{(2)}(g^{(1)}(W^{(1)}\cdot\vec{x}+b^{(1)})) + b^{(2)}$$

- Hidden layers

$$h^{(1)}(\vec{x}) = g^{(1)}(W^{(1)}\vec{x}+b^{(1)})$$

ADDING LAYERS …



- From simple linear laws …

$$h(\vec{x}) = g(\vec{x};\vec{\theta},b) = g(\sum_n \theta_n x_n + b)$$

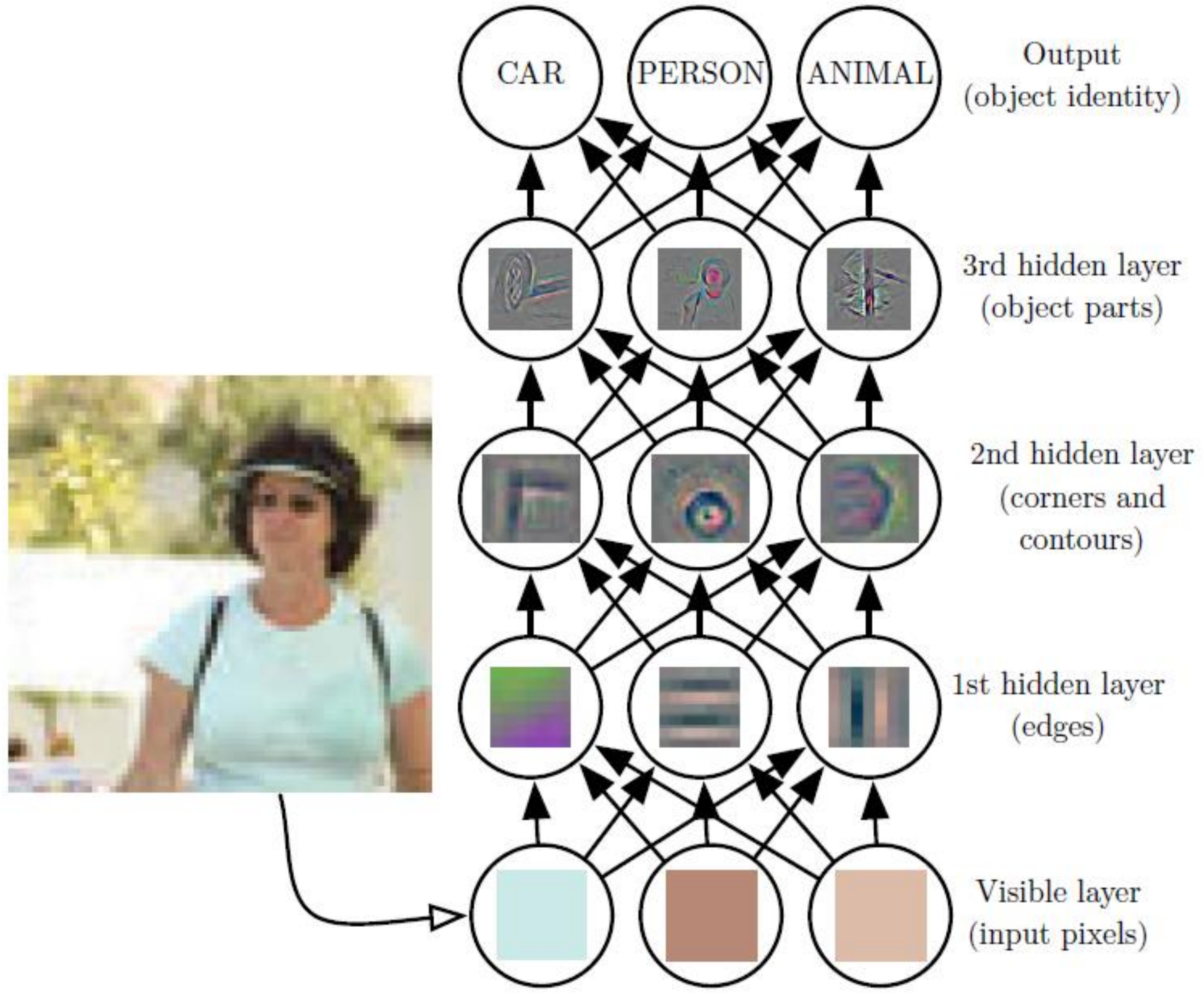
- to feedforward structures. They depend on a sequence of functions $g^{(1)}$, $g^{(2)}$, …, $g^{(k)}$ that give rise to structured hypothesis

$$\begin{aligned} h(\vec{x}) &= g^{(k)}(g^{(k-1)}(\ldots g^{(1)}(\vec{x};\vec{\theta}^{(1)},b^{(1)})\ldots);\vec{\theta}^{(k-1)},b^{(k-1)});\vec{\theta}^{(k)},b^{(k)}) \\ &= g^{(k)}(W^{(k)}g^{(k-1)}(W^{(k-1)}\ldots g^{(1)}(W^{(1)}\cdot\vec{x}+b^{(1)})\ldots)+b^{(k-1)})+b^{(k-1)})) \end{aligned}$$

- Hidden layers

$$h^{(j)}(x) = g^{(j)}(W^{(j)}g^{(j-1)}(\vec{x};\vec{\theta}^{(j-1)},b^{(j-1)})+b^{(j)} \qquad for\ j = 2,\ldots,k-1$$

REPRESENTATION AND LEARNING: THE ROLE OF DEPTH


CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

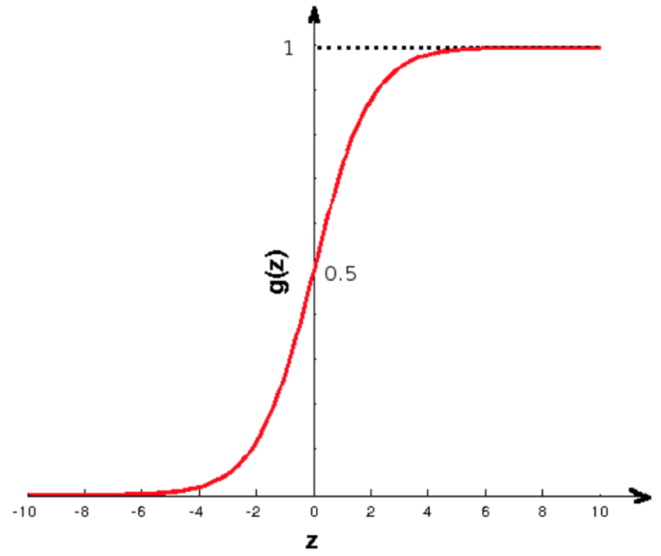
Zeiler and Fergus (2014)

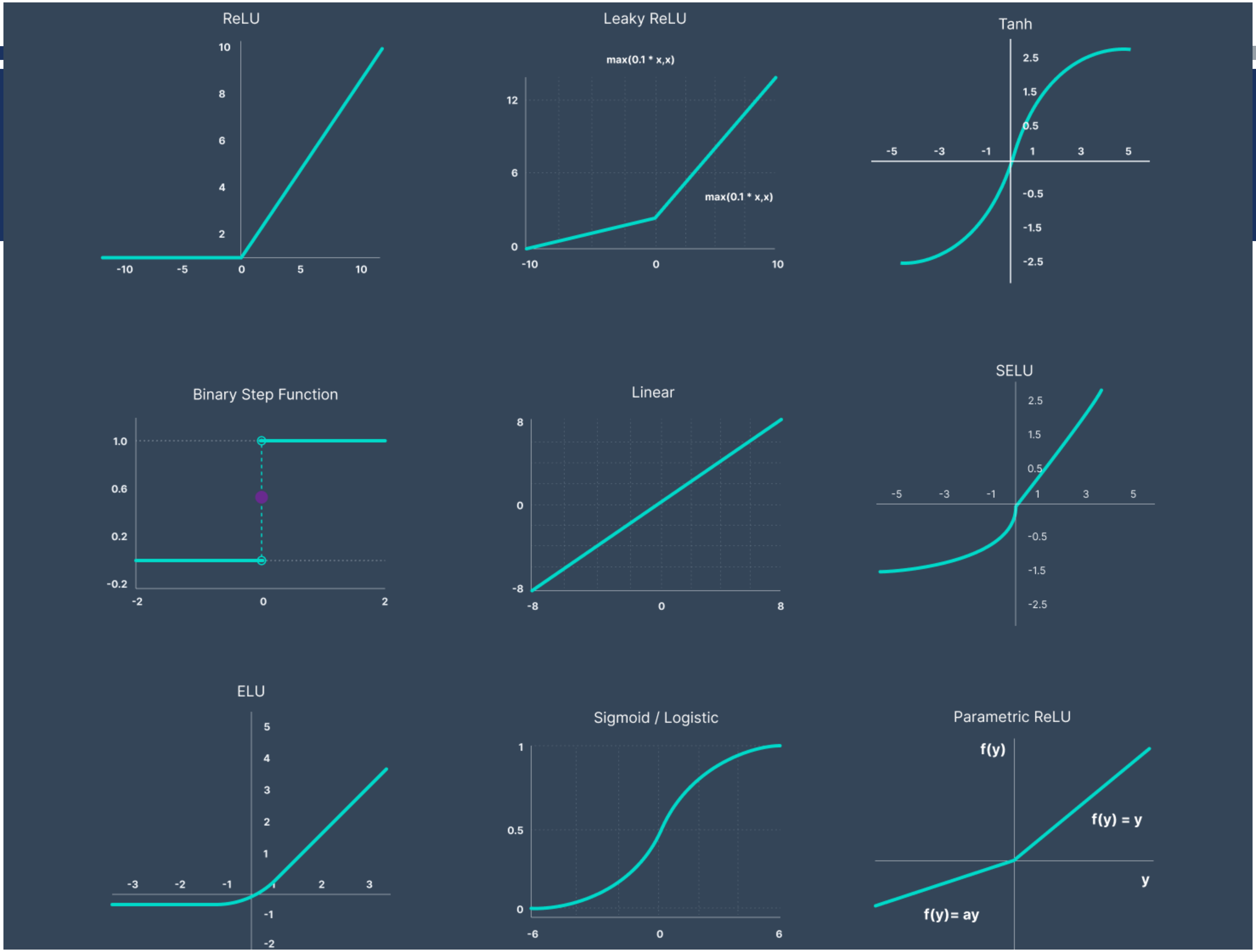
PERCEPTRON AND NON-LINEAR ACTIVATION FUNCTIONS

- We can adopt the *sigmoid* function instead of the *sgn*()
  - to bound the final values between 0 and 1
  - can be interpreted as probabilities of belonging to a class
  - belonging threshold is ">0.5"
- It remains a linear classifier

$$h(\vec{x}) = g(\sum_n \theta_n x_n + b)$$

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




ReLU
Leaky ReLU
max(0.1 * x,x)
Tanh
Binary Step Function
Linear
SELU
ELU
Sigmoid / Logistic
Parametric ReLU
f(y)
f(y) = y
y
f(y)= ay

TRAINING MLPS: BACK-PROPAGATION

- How are parameters of the network, i.e. *W, w* and *c, b* defined?
- This is the role of the training algorithm for which:

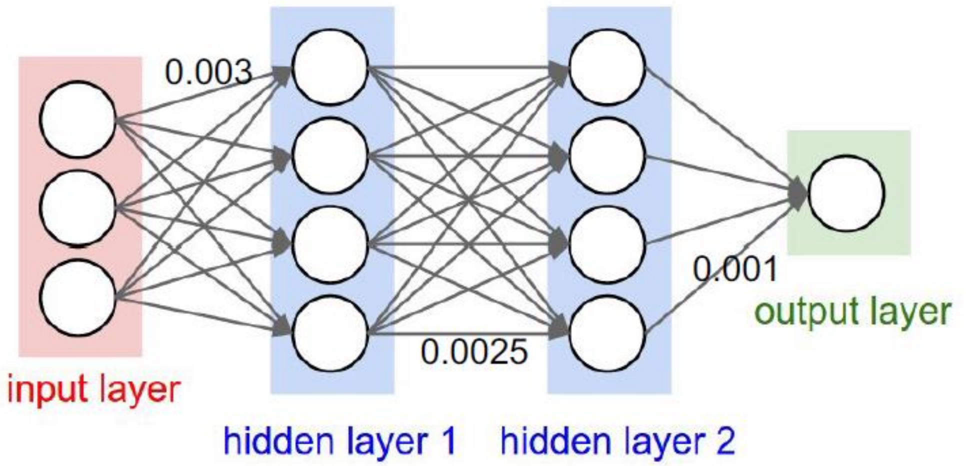
$$\begin{aligned} h(\vec{x}) &= g^{(k)}(g^{(k-1)}(\dots.g^{(1)}(\vec{x};\vec{\theta}^{(1)},b^{(1)})\dots.);\vec{\theta}^{(k-1)},b^{(k-1)});\vec{\theta}^{(k)},b^{(k)}) \\ &= g^{(k)}(W^{(k)}g^{(k-1)}(W^{(k-1)}\dots g^{(1)}(W^{(1)}\cdot\vec{x}+b^{(1)})\dots)+b^{(k-1)})+b^{(k-1)})) \end{aligned}$$

  is an accurate approximation of f*

- The learning process in MLPs is based on two notions:
  - The **optimization local to individual neurons**
  - The **adjustments to the overall network by propagation backwards from the output** (where the error manifests) **through all the hidden layers**.

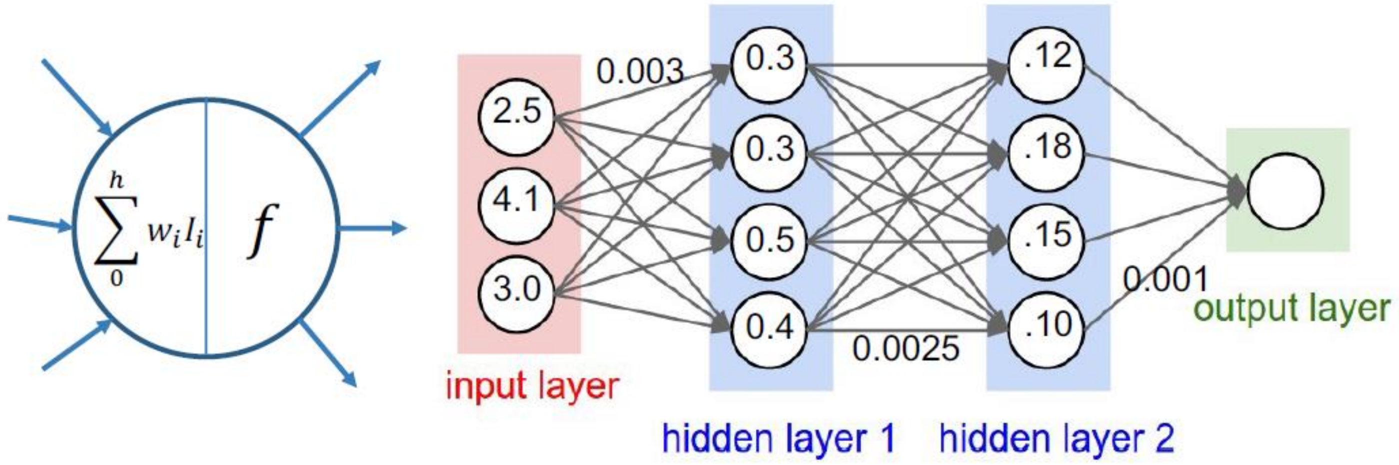
…

2.5
4.1
3.0



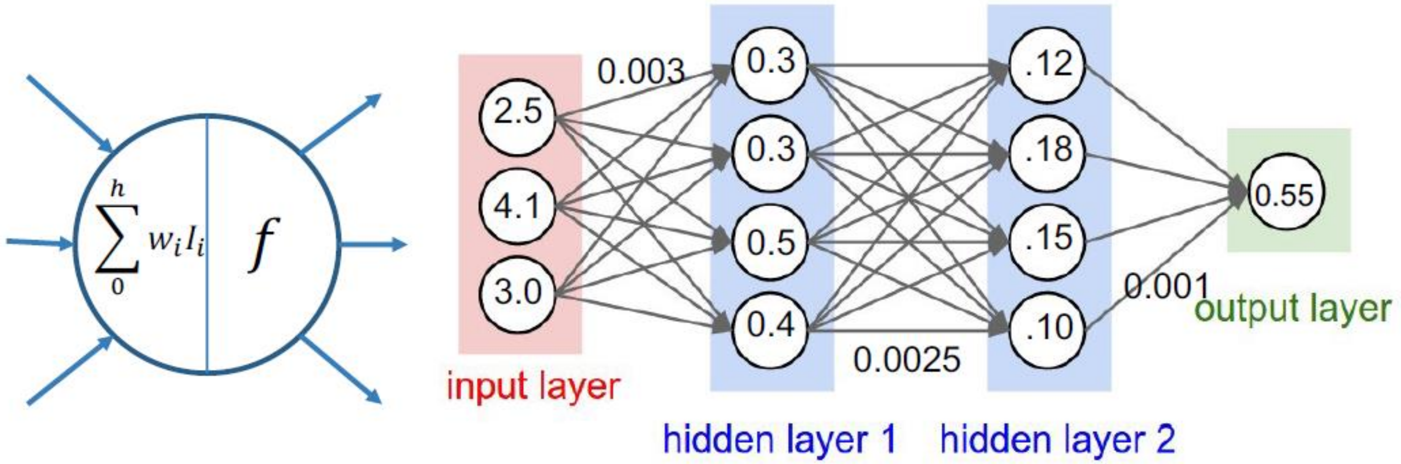
Target: 1.0

Weight Initialization and Loading the Data



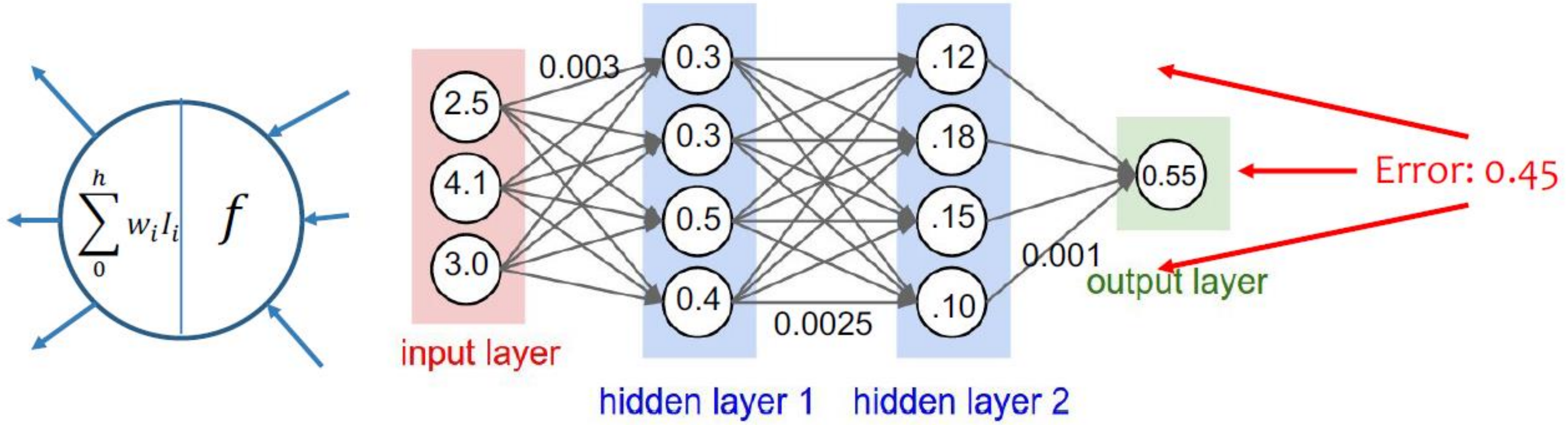
Target: 1.0

Going forward (Regularization, Dropout, Batch Normalization, …)



Target: 1.0

$$Loss\ Function: (T - O) \Rightarrow Loss: 0.45$$



Updating the weights using Backpropagation $\frac{\partial E}{\partial w_j^i}$

HOW TO INDUCE THE HYPOTHESIS *H* FROM EXAMPLES

- Learn the parameters θ and b
- To find these we look at the past data (i.e. training data) optimizing an objective function
- Objective function: the error we make on the training data
  - the sum of differences between the decision function h and the label y
  - also called Loss Function or Cost Function

$$J(\theta,b) = \sum_{i=1}^{m} (h(x^{(i)};\theta,b) - y^{(i)})^2$$

A GENERAL TRAINING PROCEDURE: STOCHASTIC GRADIENT DESCENT

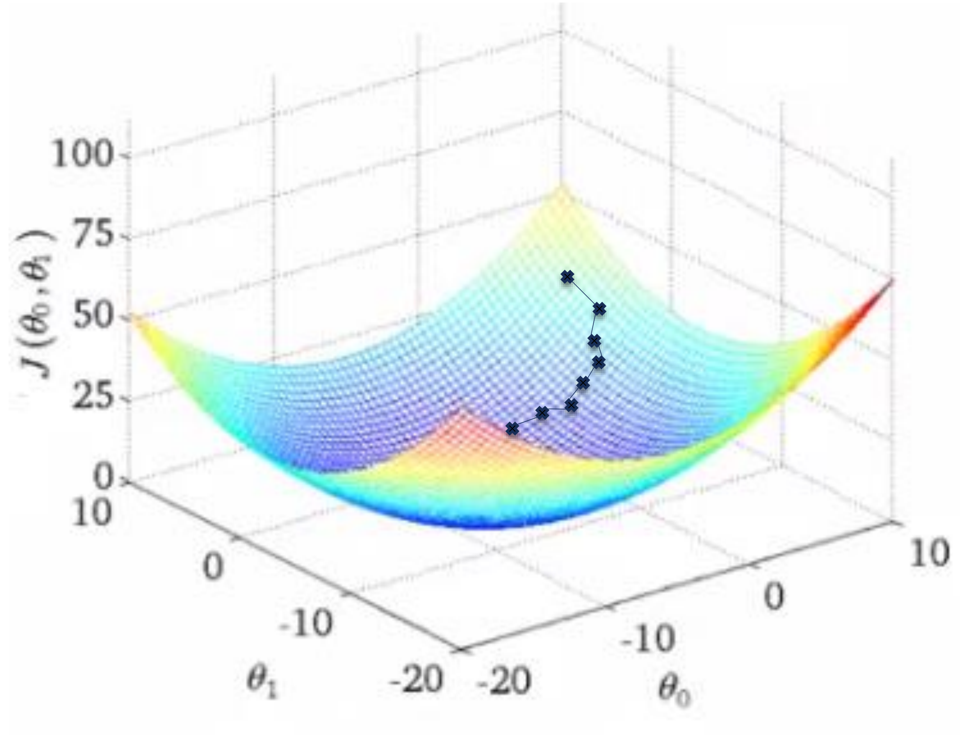
- Optimizing J means minimizing it
  - it measures the errors we make on the training data.
- We can iterate over examples and update the parameters in the direction of smaller costs
  - we aim at finding the minimum of that function
- Concretely,

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$
$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$
$$b = b - \alpha \Delta b$$

- α is a meta-parameter, the learning rate
- Δ are the partial derivatives of the cost function wrt each parameter

WHY SGD?

- Weights are updated using the partial derivatives
- Derivative pushes down the cost following the steepest descent path on the error curve

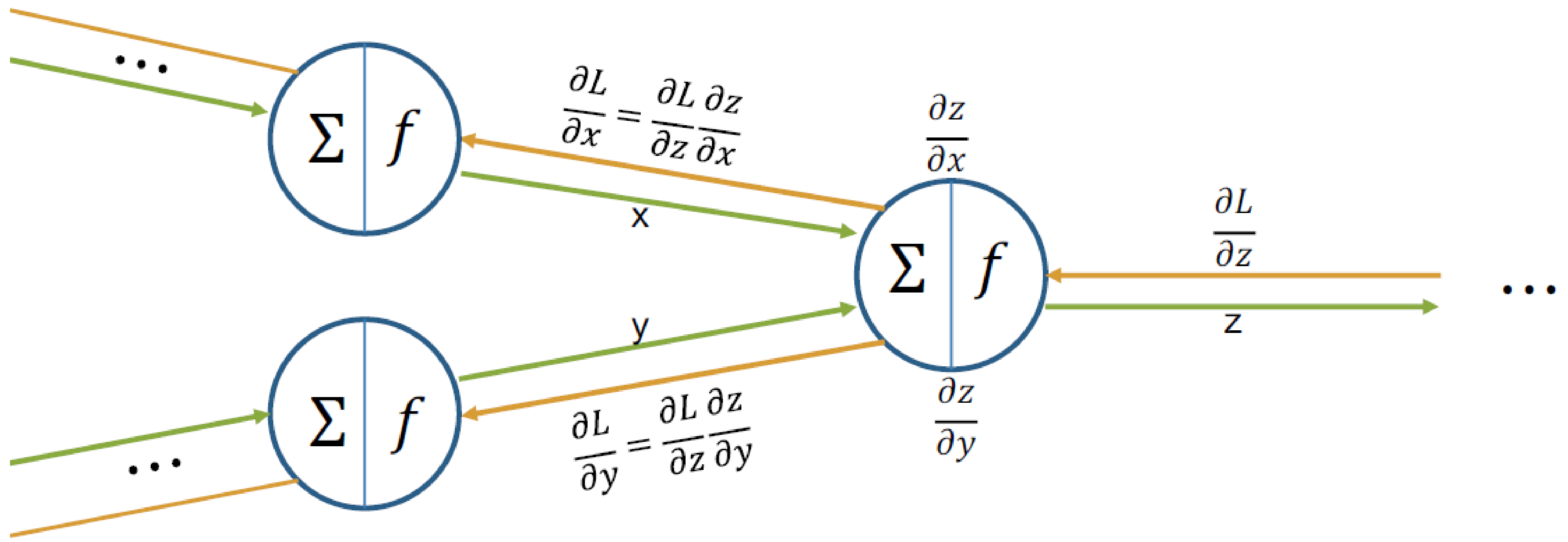


SGD PROCEDURE

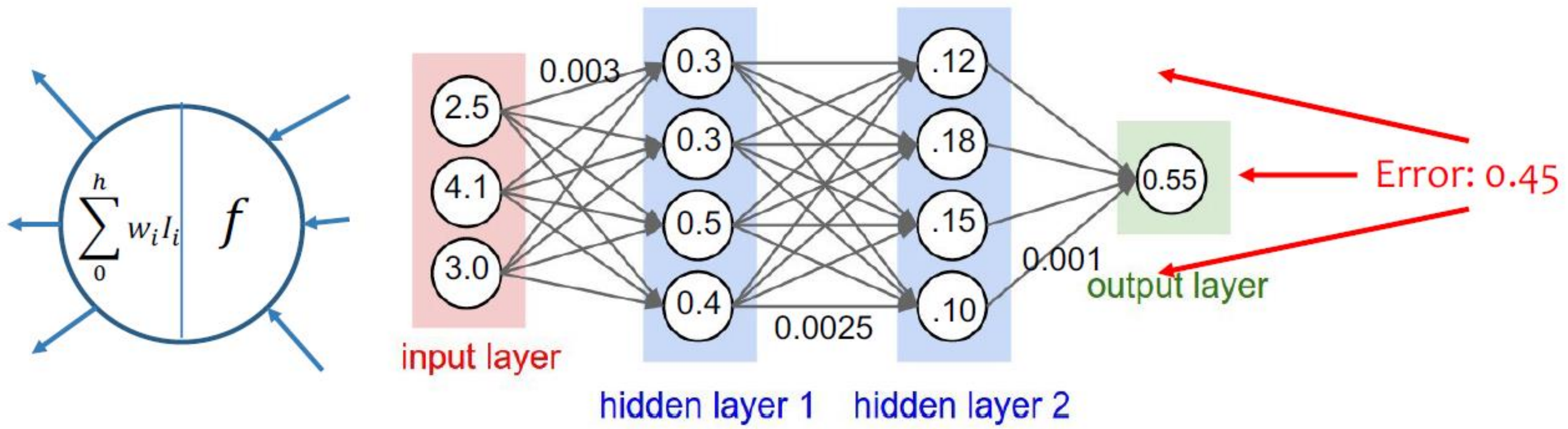
- Choose an initial random values for θ and b
- Choose a learning rate
- Repeat until stop criterion is met:
  - Pick a random training example x(i)
  - Update the parameters with

$$\theta_1 = \theta_1 - \alpha\Delta\theta_1$$
$$\theta_2 = \theta_2 - \alpha\Delta\theta_2$$
$$b = b - \alpha\Delta b$$

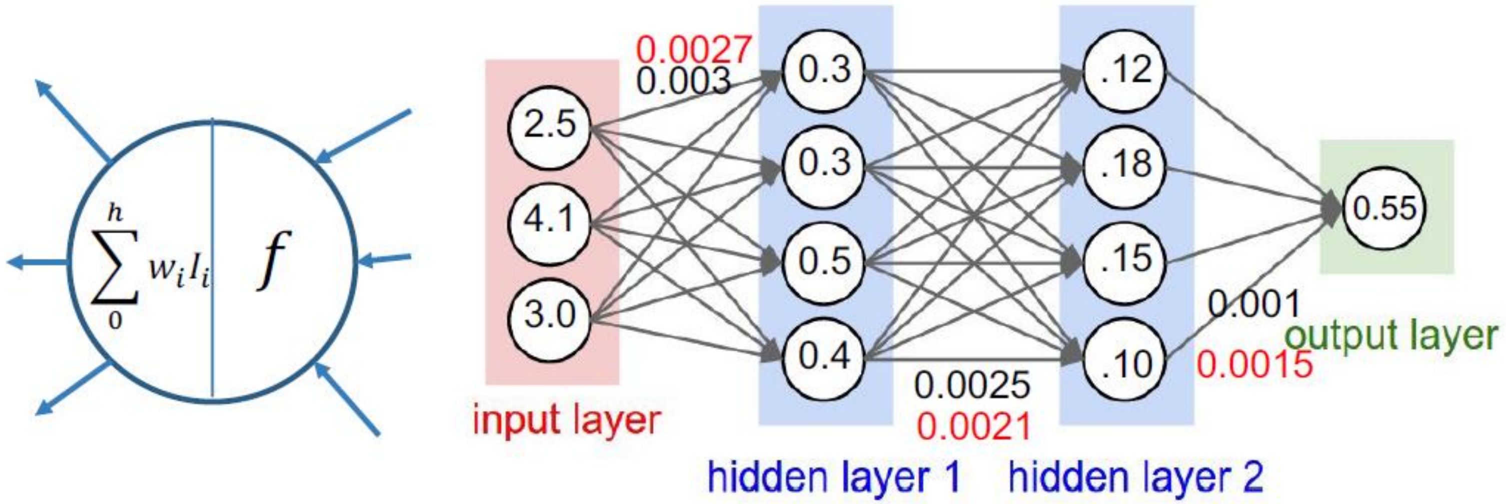
- We can stop
  - when the parameters do not change or,
  - the number of iteration exceeds a certain upper bound


$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial z}$
x
y
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
Σ f
...

Backpropagation

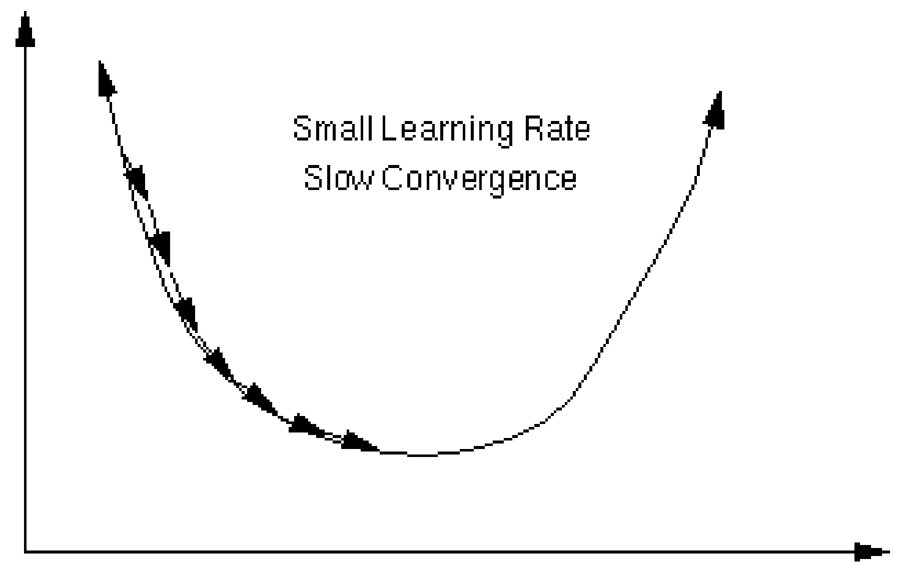


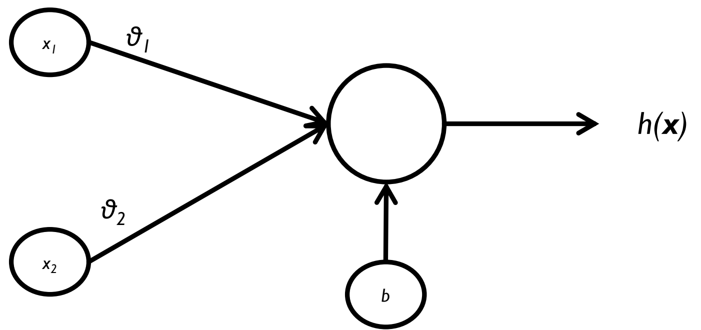
Updating the weights using Backpropagation $\frac{\partial E}{\partial w_j^i}$



By Choosing the Optimizer, new weights will be computed $w_{new} = w - \eta \frac{\partial E}{\partial w}$

LEARNING RATE: LOW VALUES

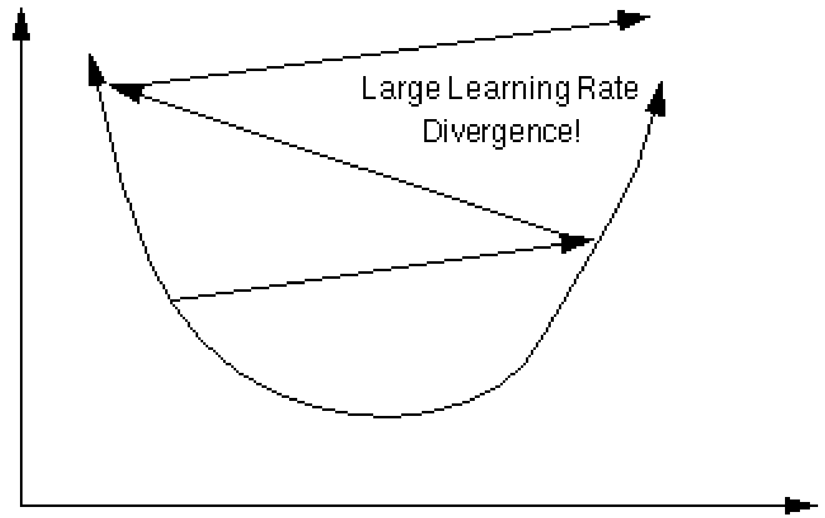


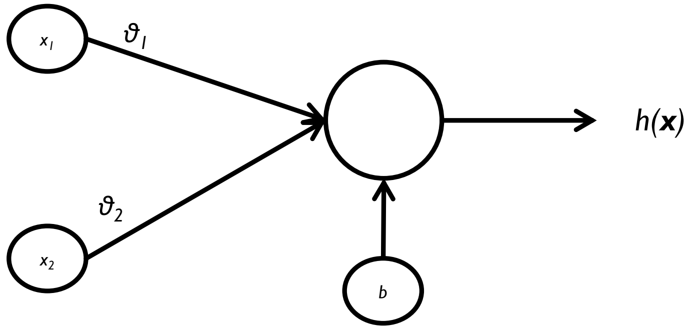


- make the algorithm converge slowly
- it is a conservative and safer choice
- However, it implies very long training

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$
$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$
$$b = b - \alpha \Delta b$$

LEARNING RATE: HIGH VALUES





- make the algorithm converge slowly
- it is a conservative and safer choice
- However, it implies very long training

$$\theta_1 = \theta_1 - \alpha\Delta\theta_1$$
$$\theta_2 = \theta_2 - \alpha\Delta\theta_2$$
$$b = b - \alpha\Delta b$$

HOW TO TRAIN A NN?

- We can **re-use the gradient descent algorithm**
  - define a cost function
  - compute the partial derivatives *wrt* to all the parameters
- As the network models function composition
  - we are going to exploit the chain rule (again)
- Setup:
  - we have a training set of m examples
  - $\{(x^{(1)},y^{(1)}), \ldots, (x^{(m)},y^{(m)})\}$
  - x are the inputs and y are the labels

$$h(z(x))$$

$$\frac{\vartheta h}{\vartheta x} = \frac{\vartheta h}{\vartheta z}\frac{\vartheta z}{\vartheta x}$$

COST FUNCTION OF A NN

- Given a single training example *(x,y)* the cost is

$$J(W,b;x,y) = \frac{1}{2}|h_{W,b}(x) - y|^2$$

- For the whole training set *J* is the mean of the errors plus a regularization term (**weight decay**)

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} J(W,b;\boldsymbol{x}^{(i)},y^{(i)}) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}|h_{W,b}(\boldsymbol{x}^{(i)}) - y^{(i)}|^2\right) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

- λ controls the importance of the two terms (it has a similar role to the C parameter in SVM)

… *DIGRESSION*: ON REGULARIZATION

- "*any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.*"
- In practical deep learning scenarios: the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately
- Many regularization approaches are based on *limiting the capacity of models*, such as neural networks, linear regression, or logistic regression, *by adding a parameter norm penalty* $\Omega(\vartheta)$ to the objective function J
- **Regularization methods**:
  - WEIGHT DECAY (*ridge regression*)
  - … CONSTRAINED OPTIMIZATION
  - DATA AUGMENTATION
  - EARLY STOPPING

SOME CONSIDERATIONS

- Randomly initialize the parameters of the network
  - for symmetry breaking
- Remember that the function g is a non-linear activation function
  - if g is the sigmoid

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

$$g'(z) = (1-g(z))g(z)$$

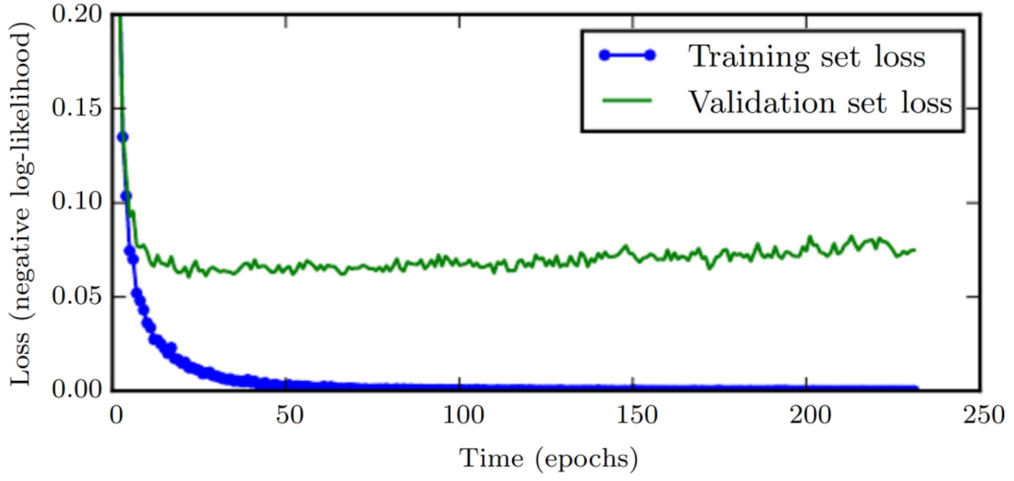
- Activations values can be cached from the forward propagation step!

$$g'(z_i^{(l)}) = (1 - g(z_i^{(l)}))g(z_i^{(l)}) = (1 - h_i^{(l)})h_i^{(l)}$$

- If you must perform multi-classification
  - there will be an output unit for each of the labels

SOME CONSIDERATIONS (2)

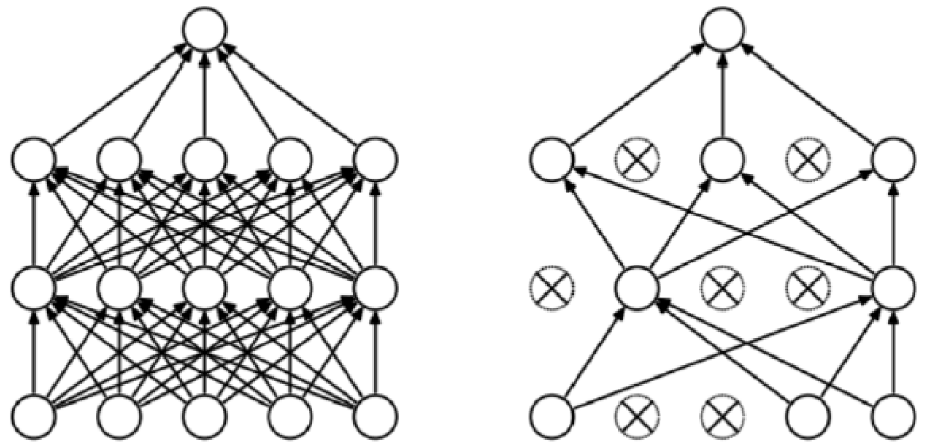
- How to stop and select the best model?
  - Waiting the iteration in which the cost function doesn't change significantly
    - Risk of overfitting
- **Early stopping**
  - Provide hints as to how many iterations can be run before overfitting
  - Split the original training set into a new training set and a validation set
  - Train only on the training set and evaluate the error on the validation set
  - Stop training as soon as the error is higher than it was the last time
  - Use the weights the network had in that previous step
- **Dropout**
  - another form of regularization to avoid overfitting data
  - during training (**only**) randomly "turn off" some of the neurons of a layer
  - it prevents co-adaptation of units between layers



DROPOUT (SVRIVASTAVA ET AL., 2014)

- Dropout can be interpreted as a way of regularizing a neural network by adding noise to its hidden units.
- It speeds-up the learning algorithm through model averaging
- It helps in reducing the risk of greedily promote simplistic solutions
- It can be applied to individual steps or in averaging mode

Randomly setting a fraction rate of input units to 0 at each update during training time.


(a) Standard Neural Net (b) After applying dropout.

DROUPOUT: EFFECTS

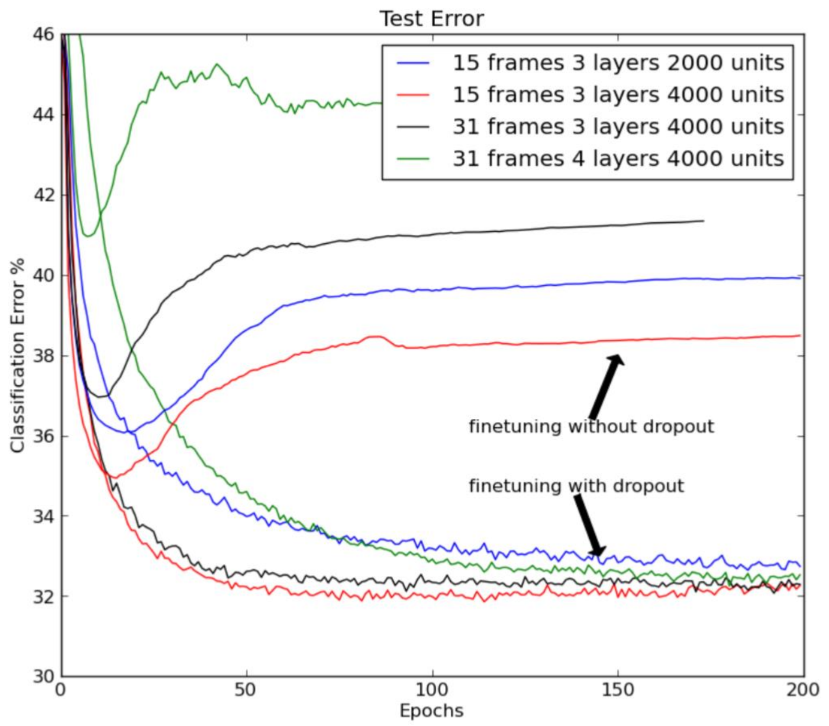
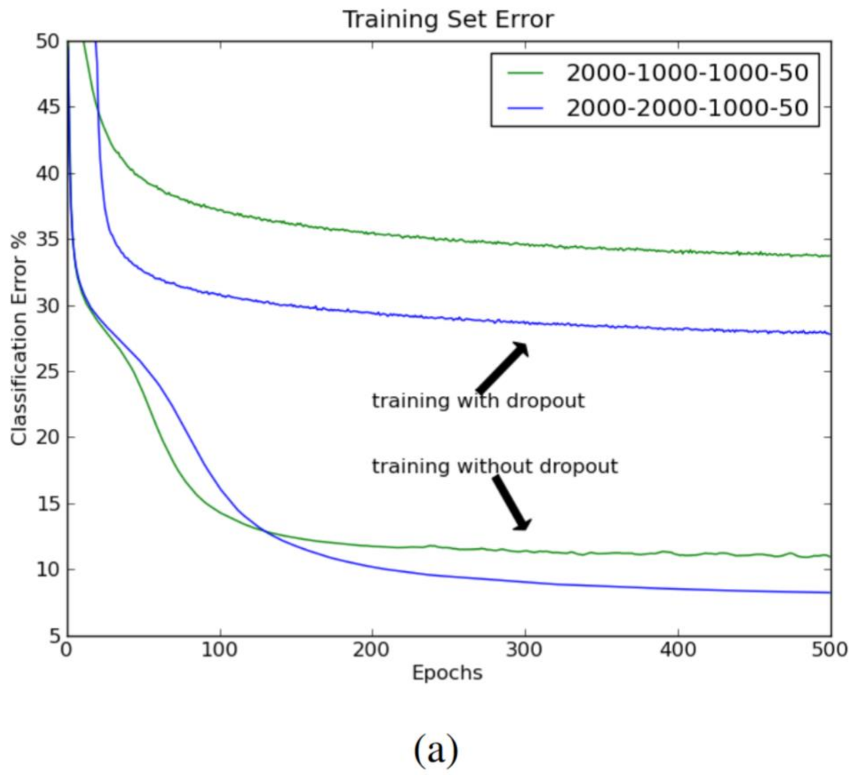


Fig. 2: The frame *classification* error rate on the core test set of the TIMIT benchmark. Comparison of standard and dropout finetuning for different network architectures. Dropout of 50% of the hidden units and 20% of the input units improves classification.

DROPOUT: EFFECTS (2)



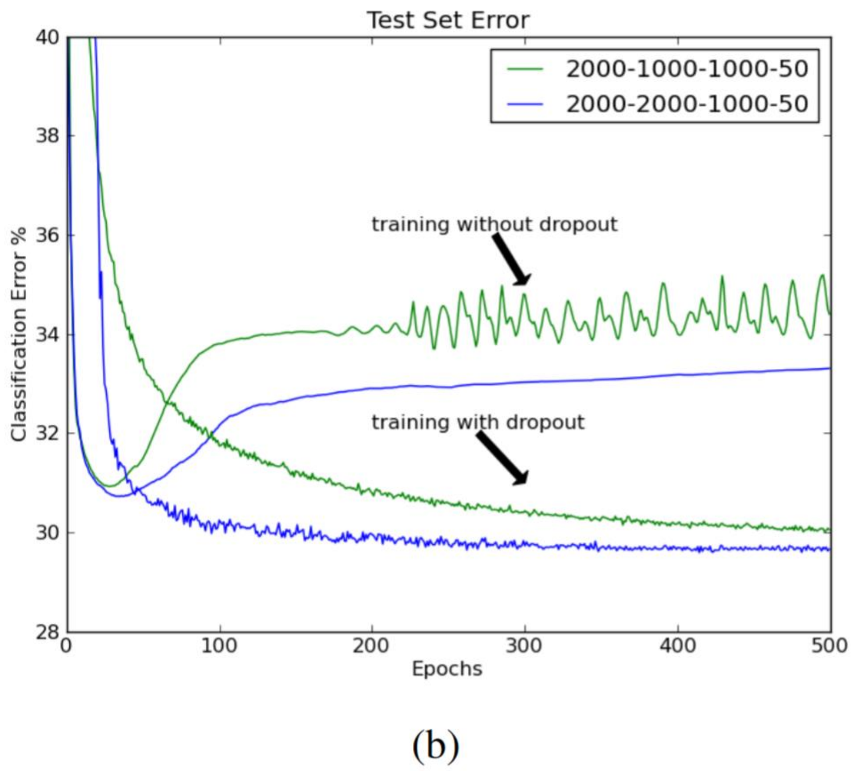


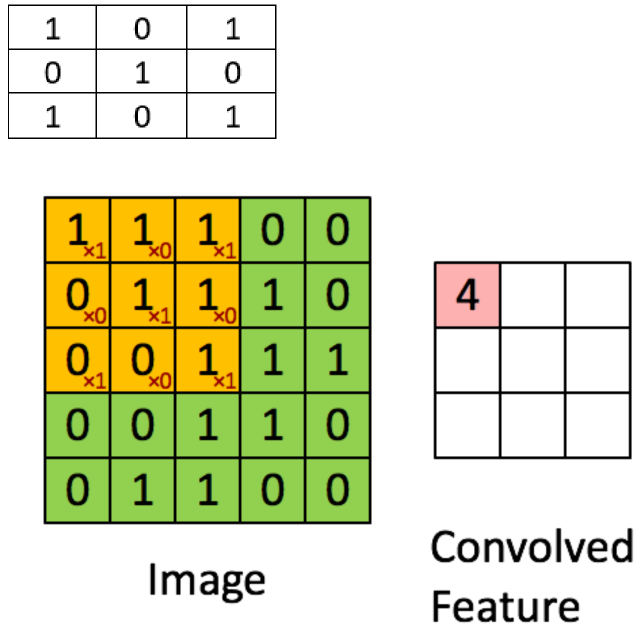
Fig. 7: Classification error rate on the (a) training and (b) validation sets of the Reuters dataset as learning progresses. The training error is computed using the stochastic nets.

RETI NEURALI

LE RETI CONVOLUZIONALI E LE IMMAGINI

CONVOLUTIONAL NEURAL NETWORKS (LE CUN, 1998)

- Mainly used for images related tasks
  - image classification
  - face detection
  - etc…
- **Learn feature representations**
  - by ***convolving*** over the input
  - with a ***filter***, that slides over the input image
- **Compositionality** (local)
  - Each filter composes a local patch of lower-level features into a higher-level representation
- **Location Invariance**
  - the detection of specific patterns is independent of where it occurs



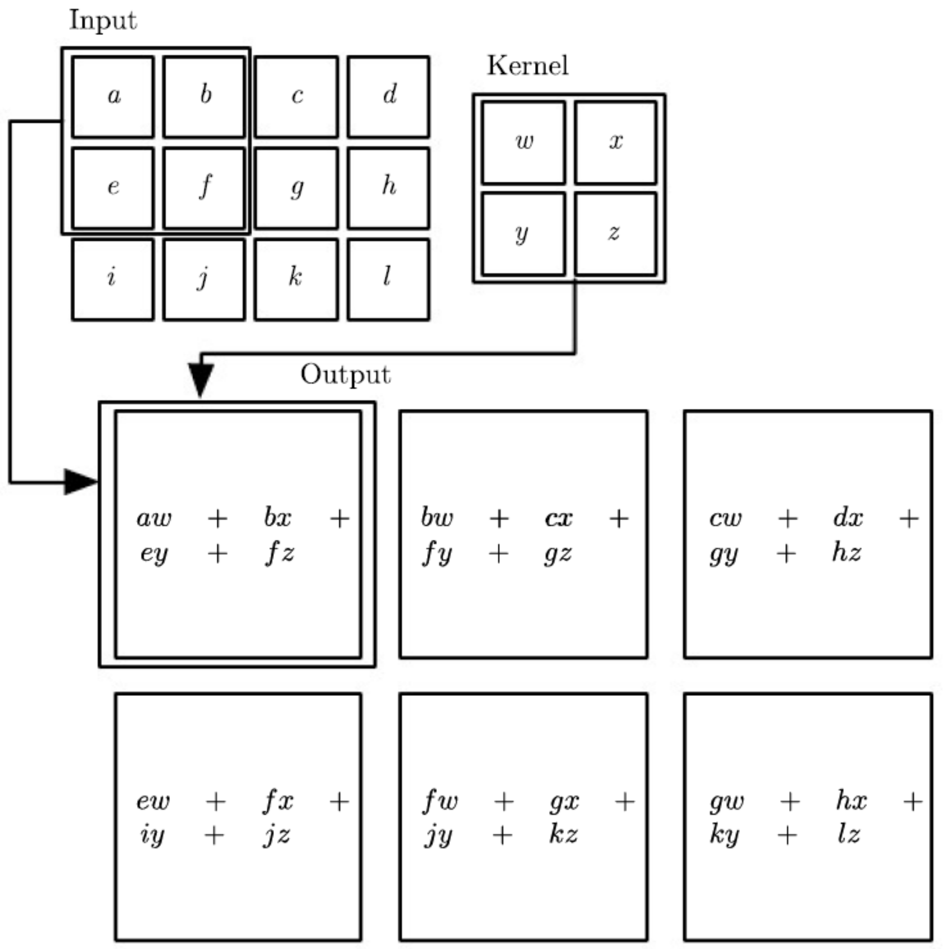
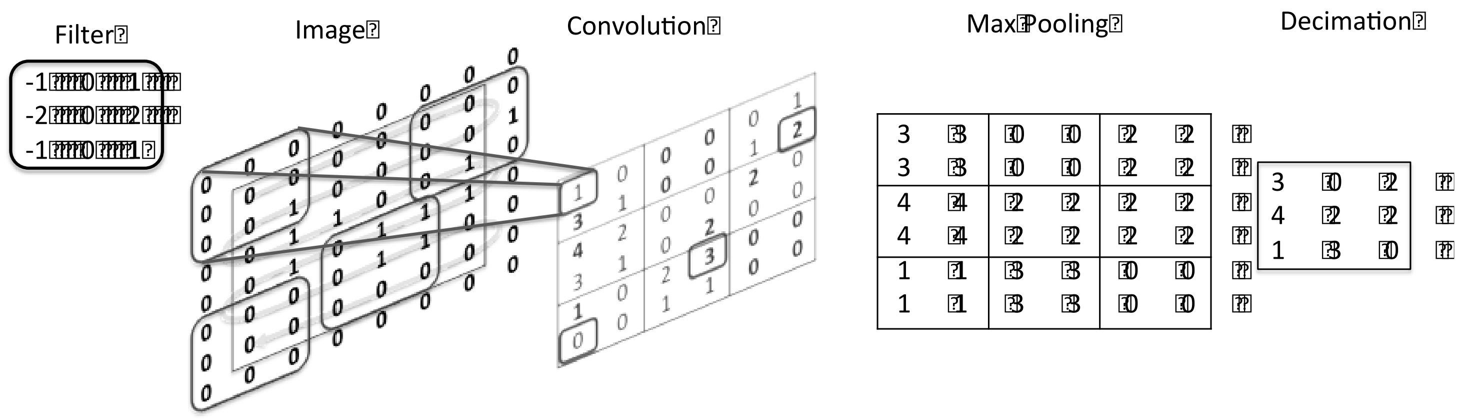


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

A FUTHER EXAMPLE OF: CONVOLUTION WITH POOLING, AND DECIMATION OPERATIONS



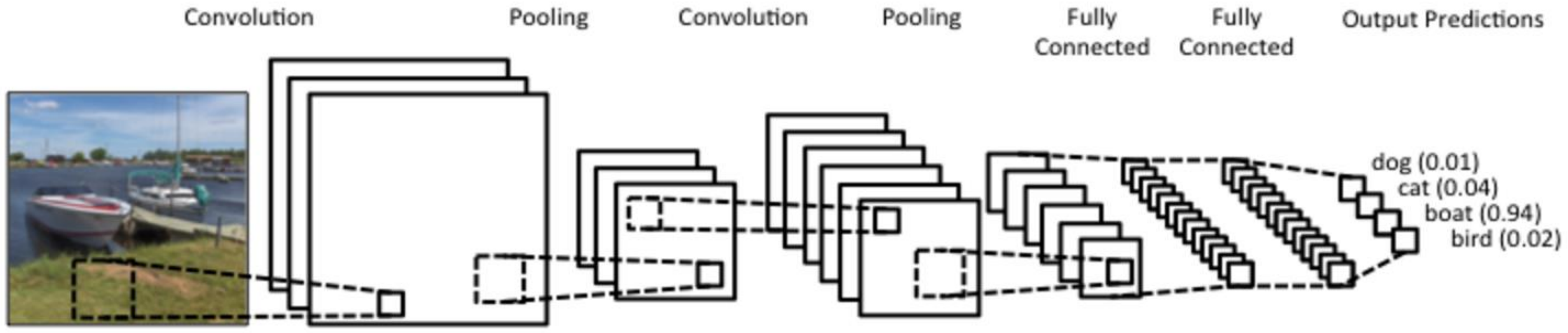
3	3	0	0	2	2
3	3	0	0	2	2
4	4	2	2	2	2
4	4	2	2	2	2
1	1	3	3	0	0
1	1	3	3	0	0

3	0	2
4	2	2
1	3	0

- An image is convolved with a filter; curved rectangular regions in the first large matrix depict a random set of image locations
- Maximum values within small 2×2 regions are indicated in bold in the central matrix
- The results are pooled, using max-pooling then decimated by a factor of two, to yield the final matrix

CONVOLUTIONAL NEURAL NETWORKS

- CNNs automatically learn the parameters of the filters
  - a filter is a matrix of parameters
  - the key aspect is that a filter is adopted for the whole image
- Convolution can be applied in **multiple** layers
  - a layer l+1 is computed by convolving over output produced in layer l
  - Pooling is an operation often adopted for taking the most informative features that are learned after a convolution step

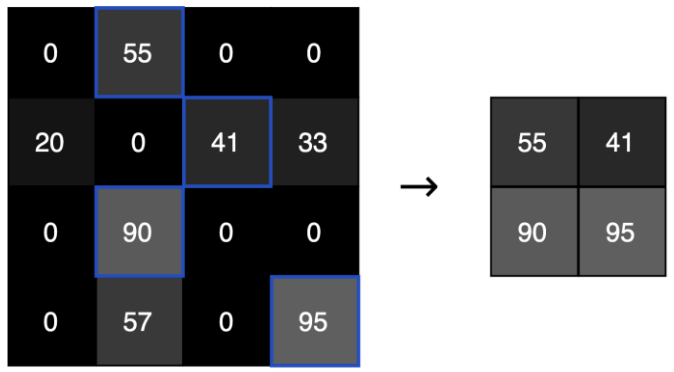


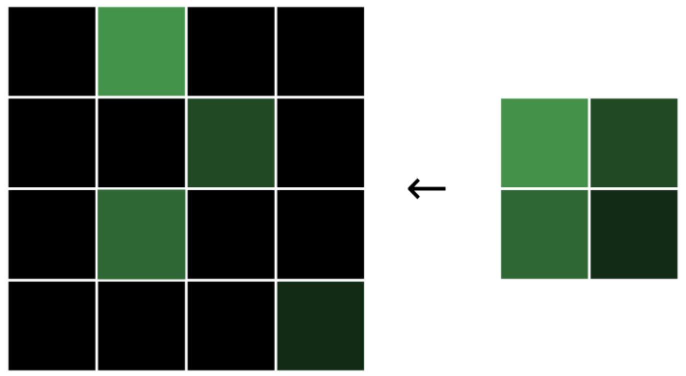
POOLING AND SUBSAMPLING LAYERS

- What are the consequences of backpropagating gradients through max or average pooling layers?
- **Max pooling**: the units that are responsible for the maximum within each zone j, k —the "winning units"— are the only to get the *backpropagated gradient*
- **Average pooling**: the averaging is simply a special type of convolution with a fixed kernel that computes the (possibly weighted) average of pixels in a zone
  - the required gradients are therefore like std conv. layers
- The subsampling step either samples every n^{th} output, or avoids needless computation by only evaluating every n^{th} pooling computation

TRAINING IN CNN: BACKPROPAGATION AND MAX POOLING

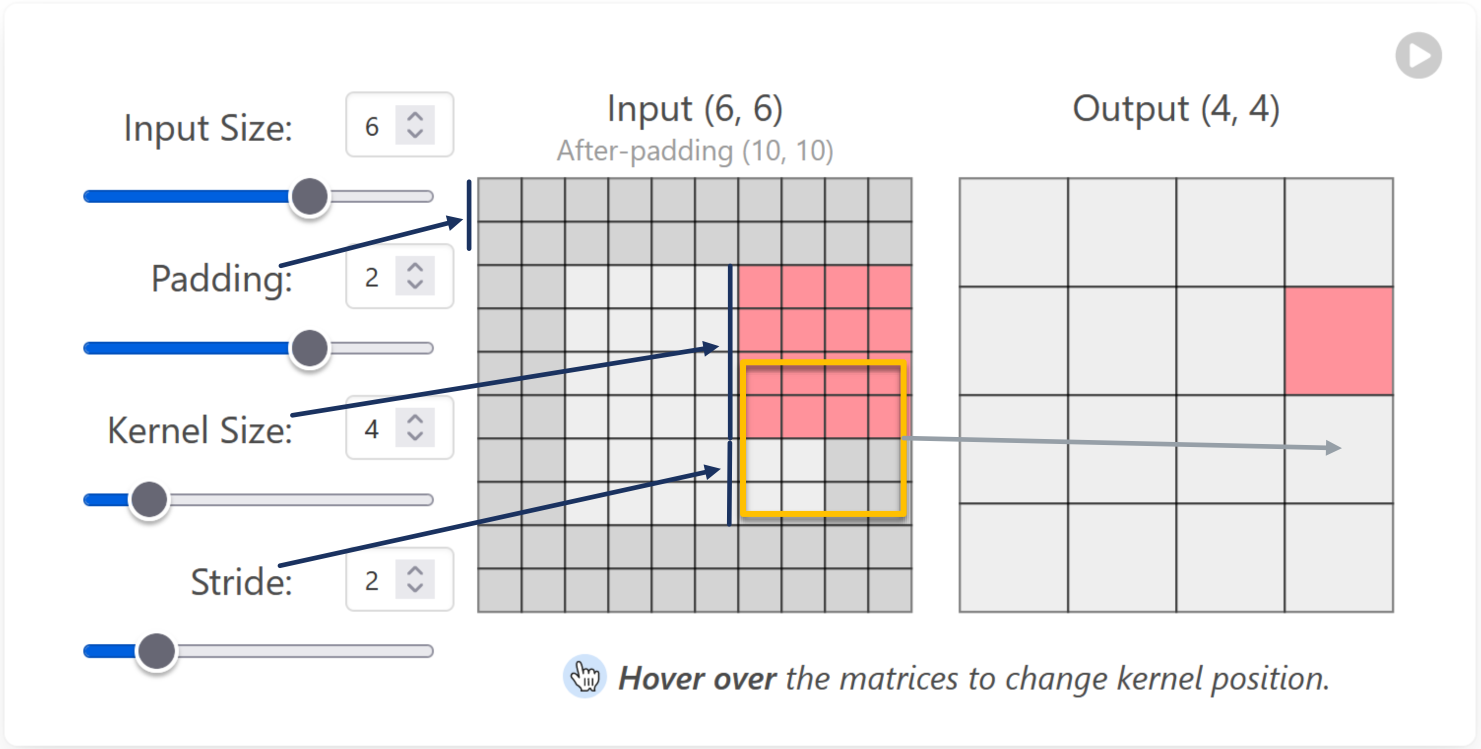
- A Max Pooling layer can’t be trained because it doesn’t actually have any weights
  - It still supports a method for it to calculate gradients





- How is ∂L / ∂inputs ?
  - An input pixel that isn’t the max value in its 2x2 block have *zero* marginal effect on the loss, as any slightly change of its value wouldn’t change the output at all!
    - ∂L / ∂inputs = 0 for any non-max pixels.
  - On the other hand, an input pixel that *is* the max value would have its value passed through to the output, so
    ∂output / ∂input = 1,   meaning   ∂L / ∂input = ∂L / ∂output.

TRAINING A CNN: TERMINOLOGY


Input Size: 6

Padding: 2

Kernel Size: 4

Stride: 2

Input (6, 6)

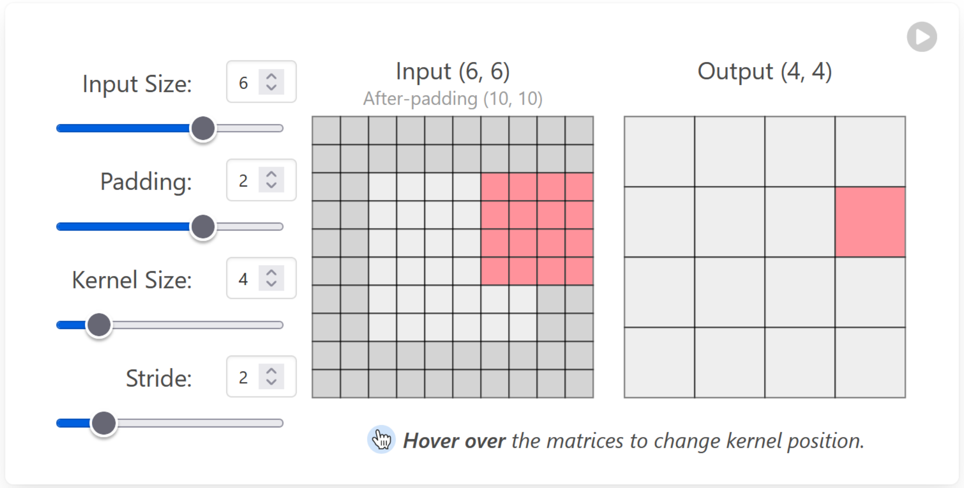
After-padding (10, 10)

Output (4, 4)

***Hover over** the matrices to change kernel position.*

DIMENSIONS

- The dimension of the output of a convolution is the following



Input Size: 6

Padding: 2

Kernel Size: 4

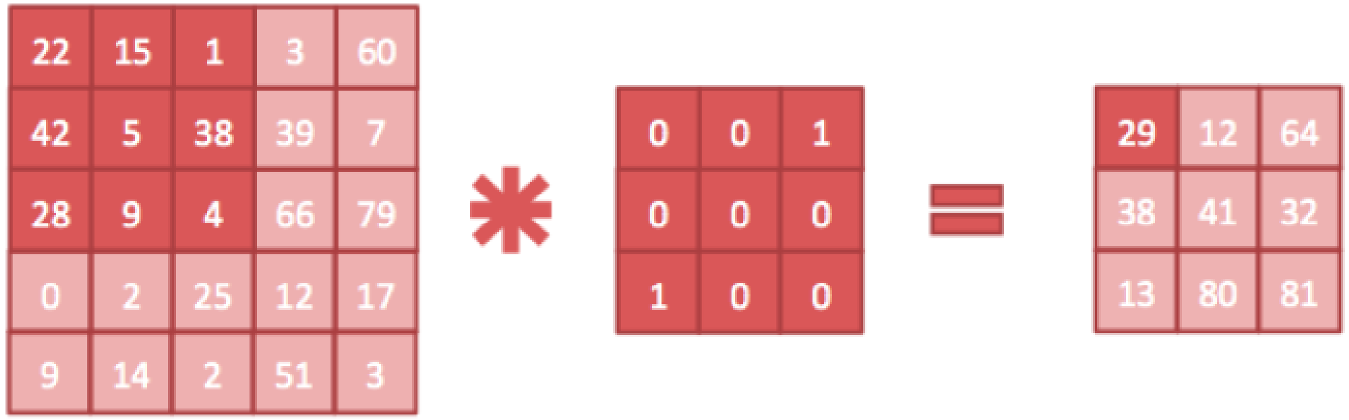
Stride: 2

Hover over the matrices to change kernel position.

$$O = \frac{InputD - KernelD + 2PaddingD}{StrideD} + 1$$

CONVOLUTIONAL NEURAL NETWORKS

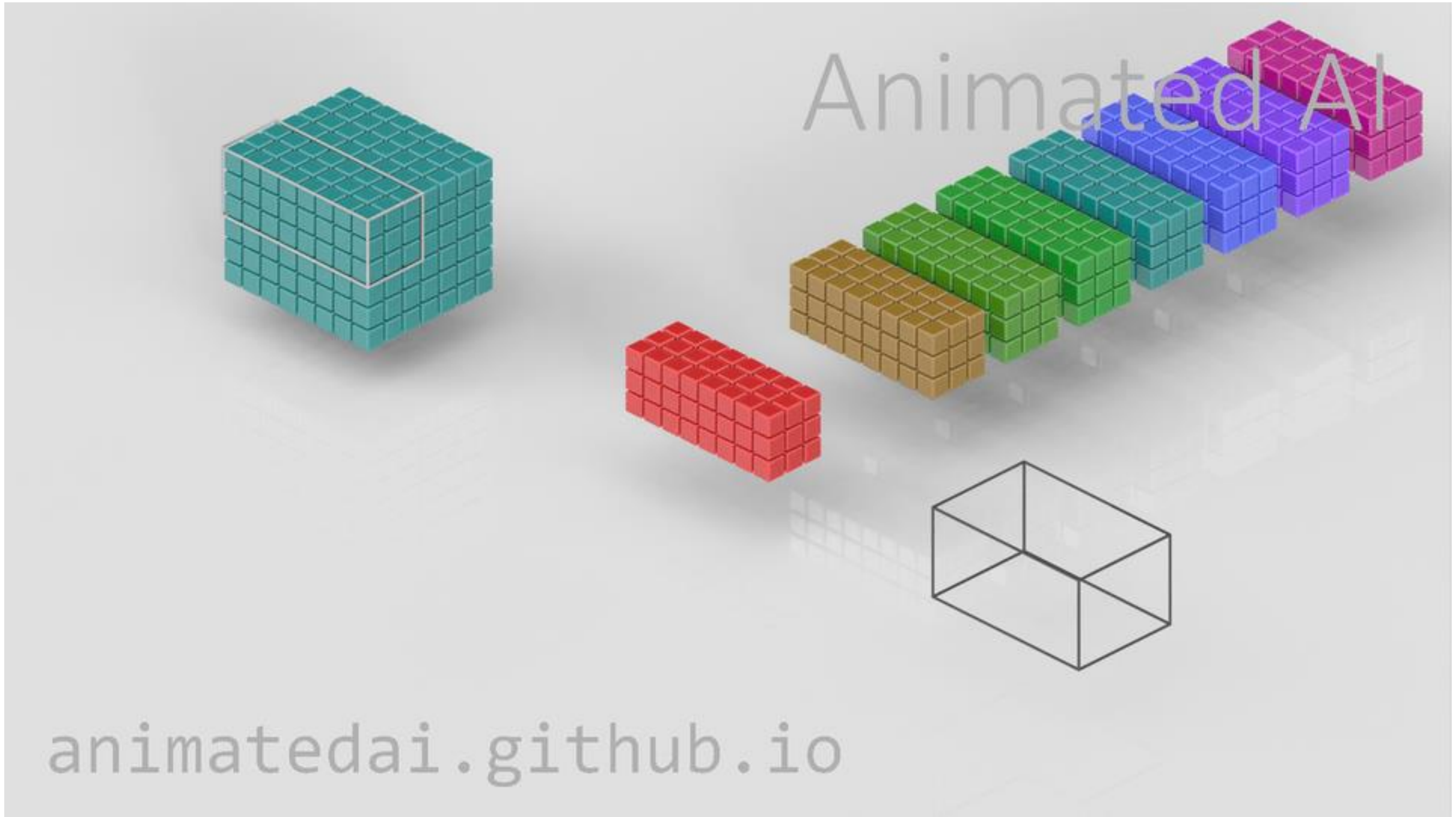
- Convolutional networks (LeCun,1998) are neural networks for processing data with a grid-like topology (e.g. 2D images, time-series data, texts)
- Convolution is a mathematical operation obtained by combining two functions



$$\begin{bmatrix} 22 & 15 & 1 & 3 & 60 \\ 42 & 5 & 38 & 39 & 7 \\ 28 & 9 & 4 & 66 & 79 \\ 0 & 2 & 25 & 12 & 17 \\ 9 & 14 & 2 & 51 & 3 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 29 & 12 & 64 \\ 38 & 41 & 32 \\ 13 & 80 & 81 \end{bmatrix}$$

- In CNNs at least one layer is expressed through a convolution matrix

UNA VISIONE ANIMATA …



THE IMAGENET CHALLENGE

- Crucial in demonstrating the effectiveness of deep CNNs
- Task: recognize object categories in Internet imagery
- The 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) classification task - classify image from Flickr and other search engines into 1 of 1000 possible object categories
- Serves as a standard benchmark for deep learning
- The imagery was hand-labeled based on the presence or absence of an object belonging to these categories. 1.2 million images in the training set with 732-1300 training images available per class
- A random subset of 50,000 images was used as the validation set, and 100,000 images were used for the test set where there are 50 and 100 images per class respectively

Goal

ImageNet

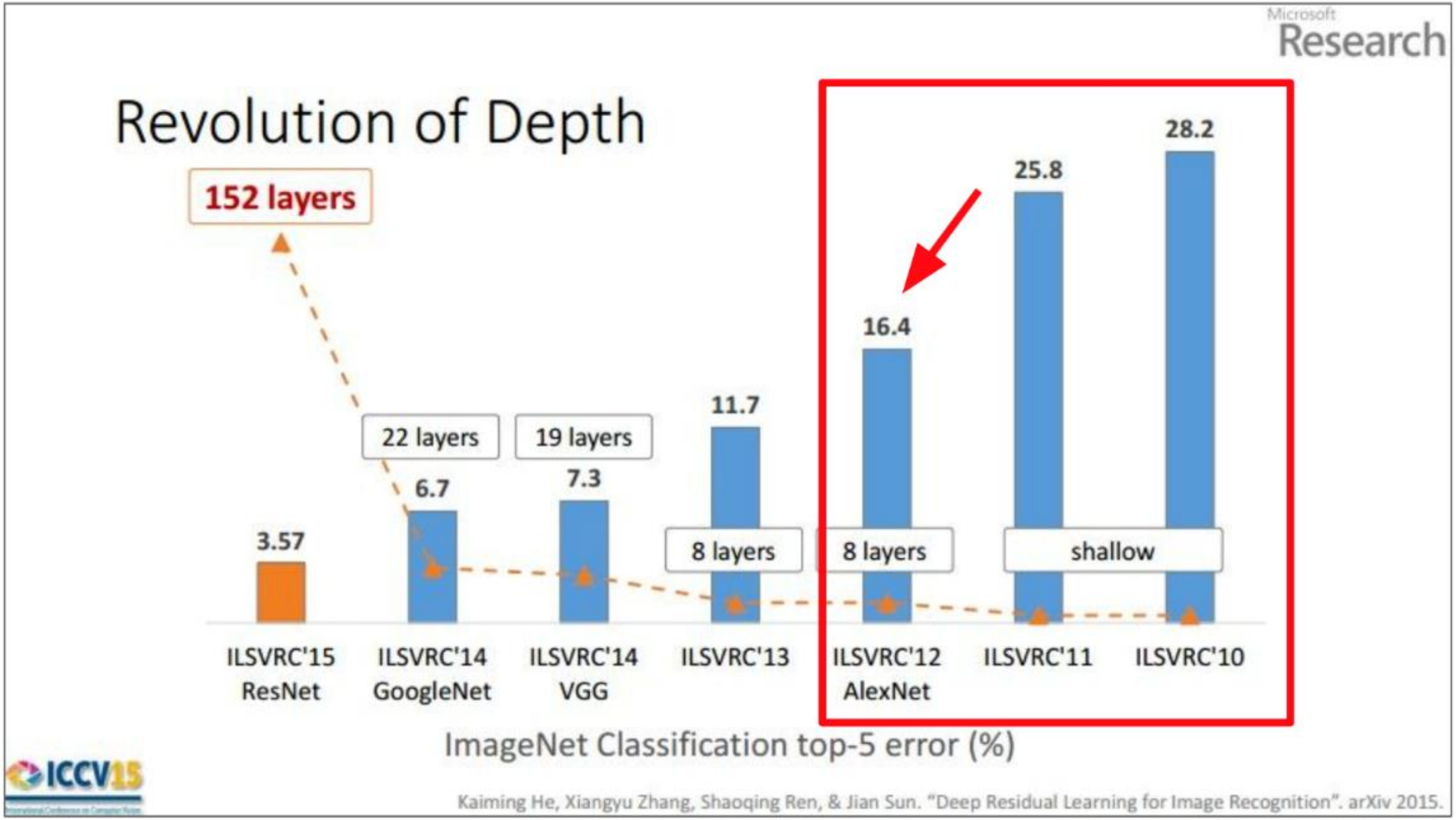
ILSVRC

- Over 1
- Rough
- Collect Turk

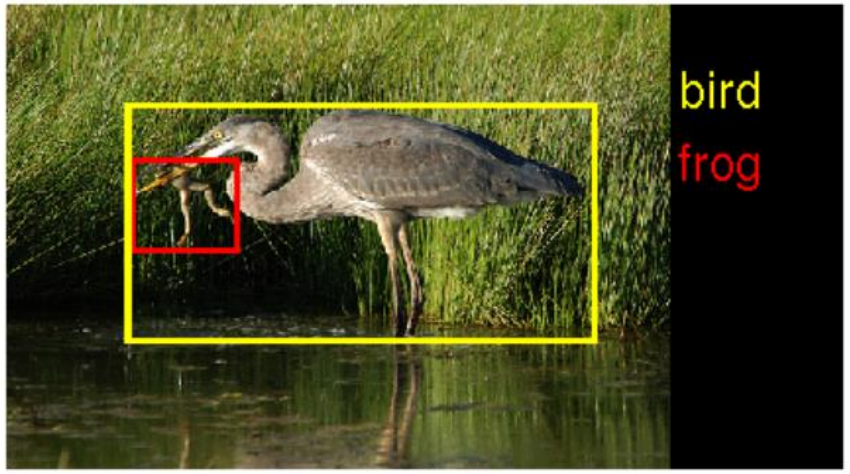
- Annual competition of image classification at large scale
- 1.2M images in 1K categories
- Classification: make 5 guesses about the image label

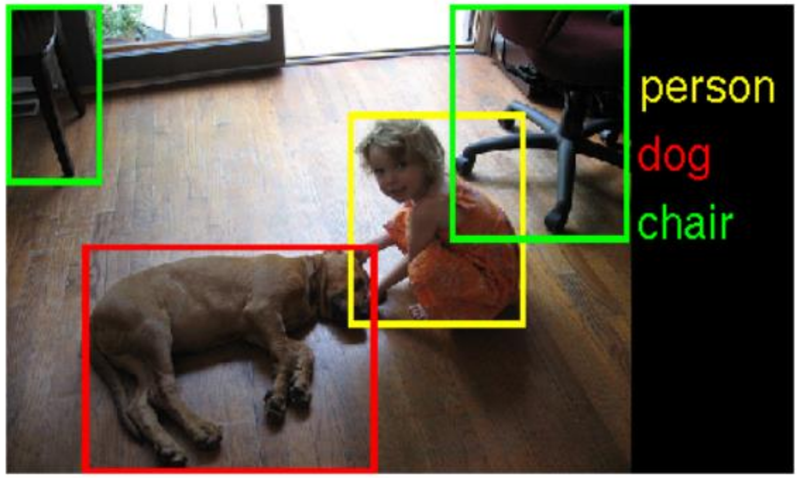
[illegible]

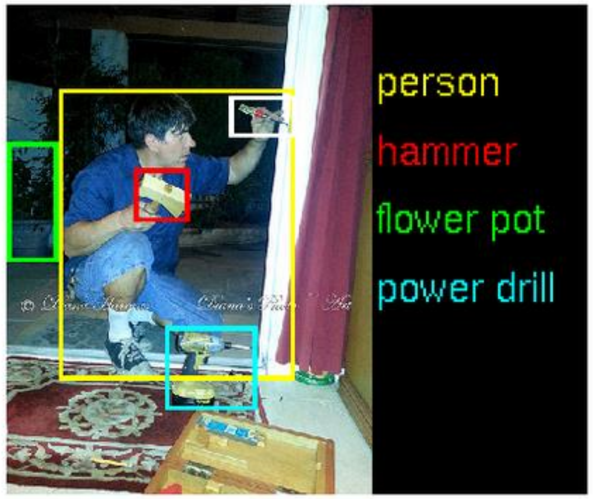
[illegible]

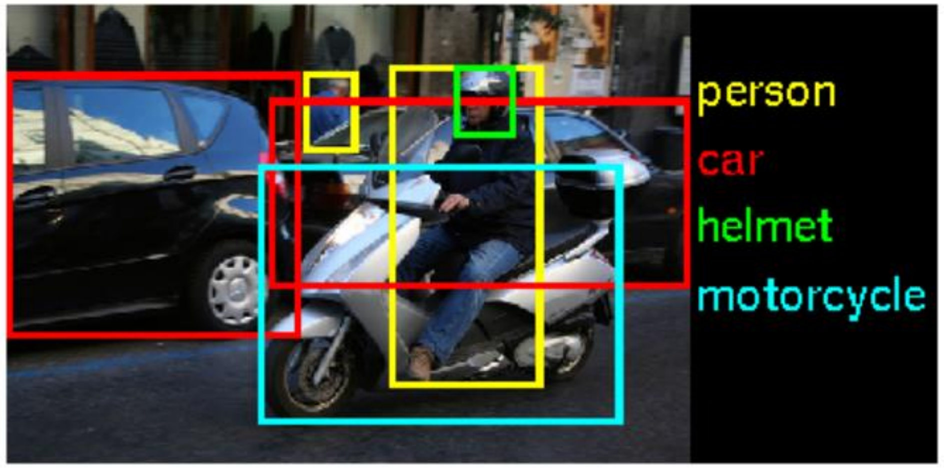

Microsoft Research
Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

ILSVRC2014 EXAMPLES

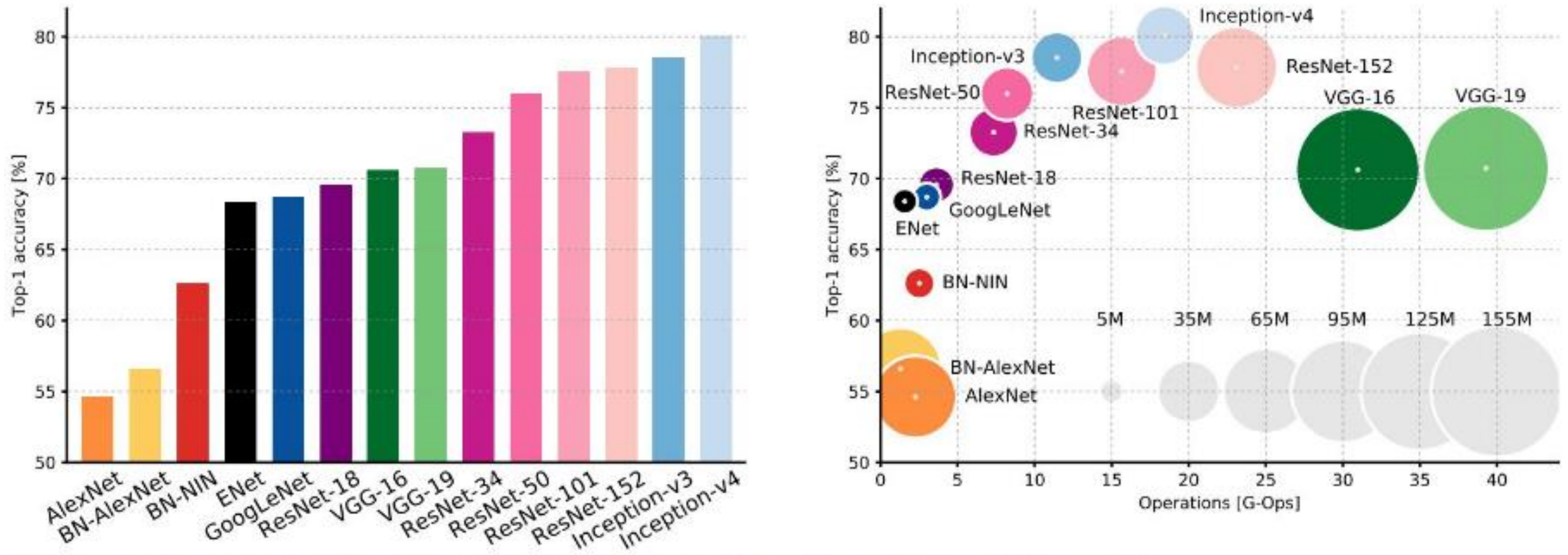






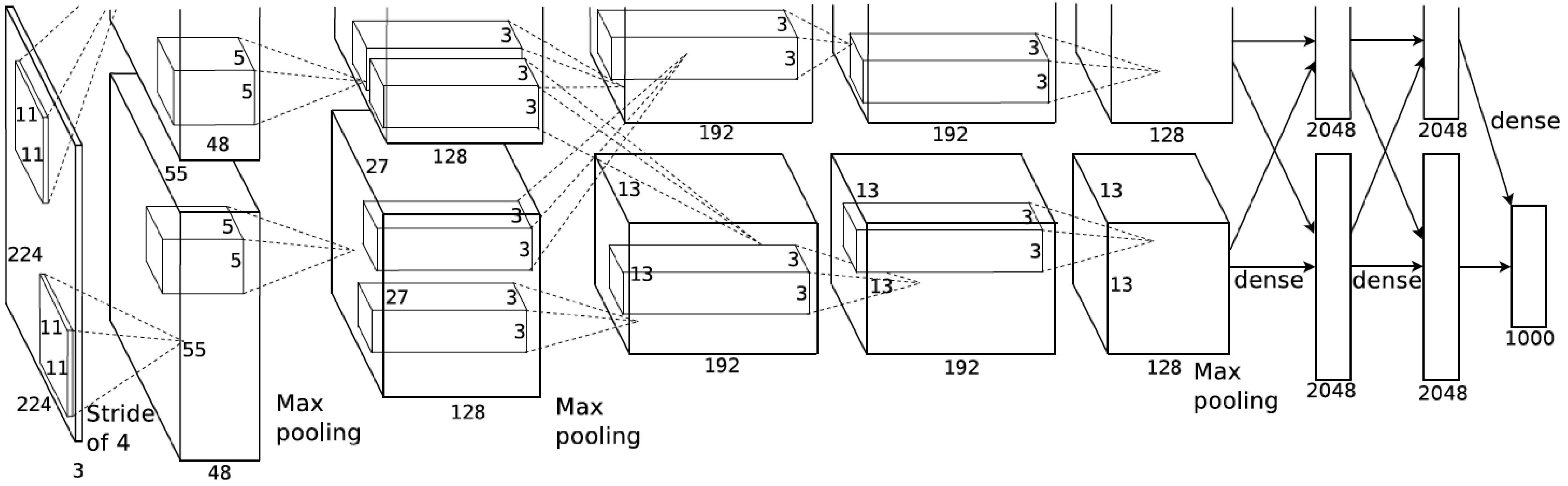


DEEP CONVOLUTIONAL NETWORKS AND THEIR SCALE

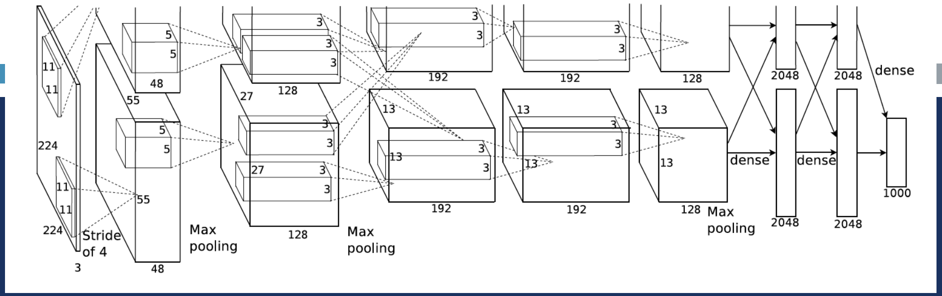


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

AN EXAMPLE: ALEXNET (8 LAYERS)



AlexNet won the 2012 ImageNet competition with a top-5 error rate of 15.3%, compared to the second place top-5 error rate of 26.2%



ALEXNET: OVERVIEW

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

ALEXNET: THE ARCHITECTURE

- It has 8 layers with learnable parameters.
- The input to the Model is RGB images.
- It has 5 convolution layers with a combination of max-pooling layers.
- Then it has 3 fully connected layers.
- The activation function used in all layers is **Relu**, whereas Softmax is used in the output layer is
- It used **two Dropout layers**.
- The total number of parameters in this architecture is **62.3 million**.

WHAT HAS BEEN LEARNT?

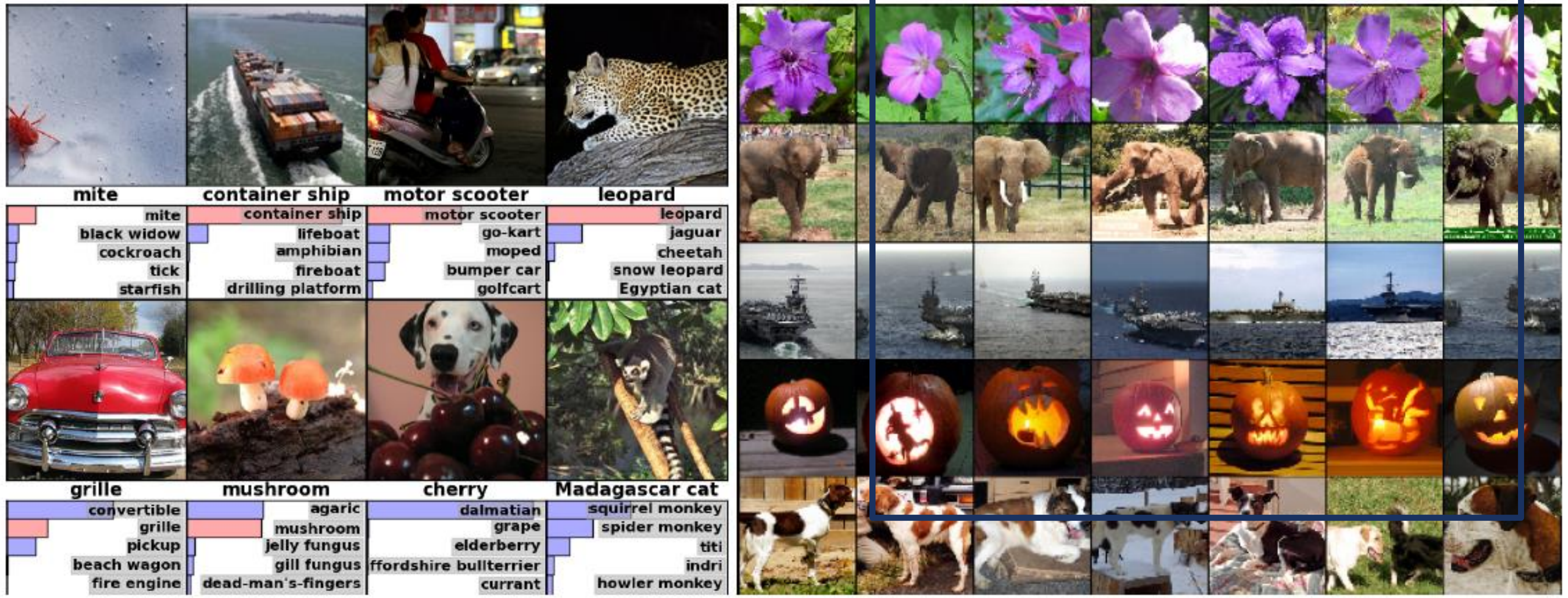


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

CURRENT CNNS: YOLO (BOCHKOVSKIY ET AL.(2020))

Bochkovskiy et al.(2020), Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.~M. , YOLOv4: Optimal Speed and Accuracy of Object Detection, 2020, https://arxiv.org/abs/2004.10934v1.

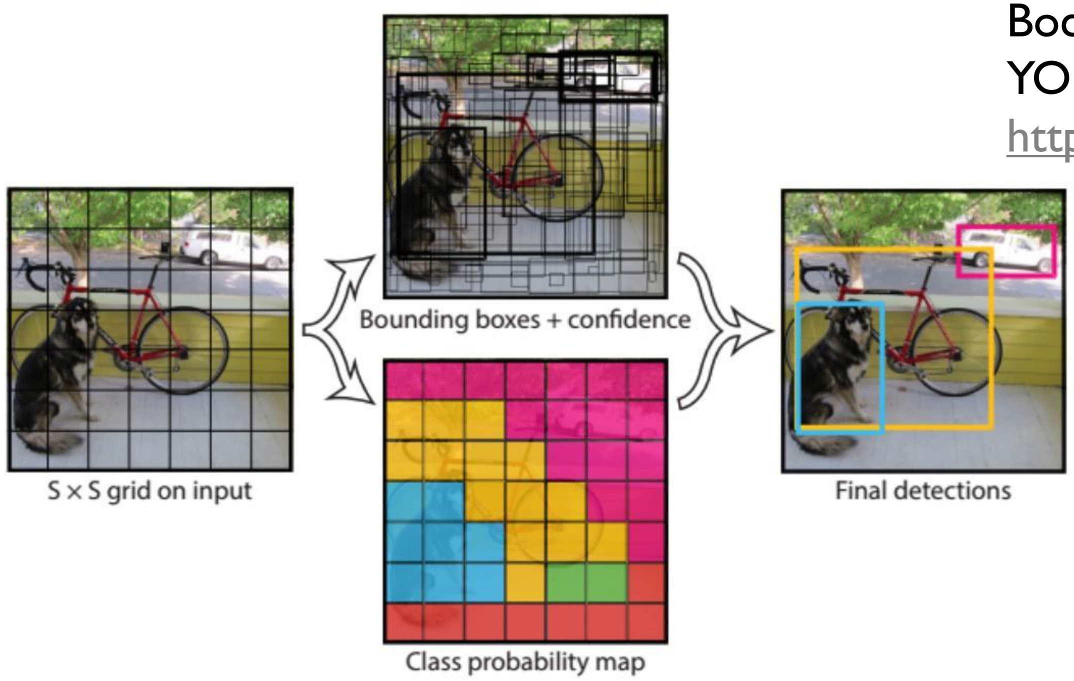
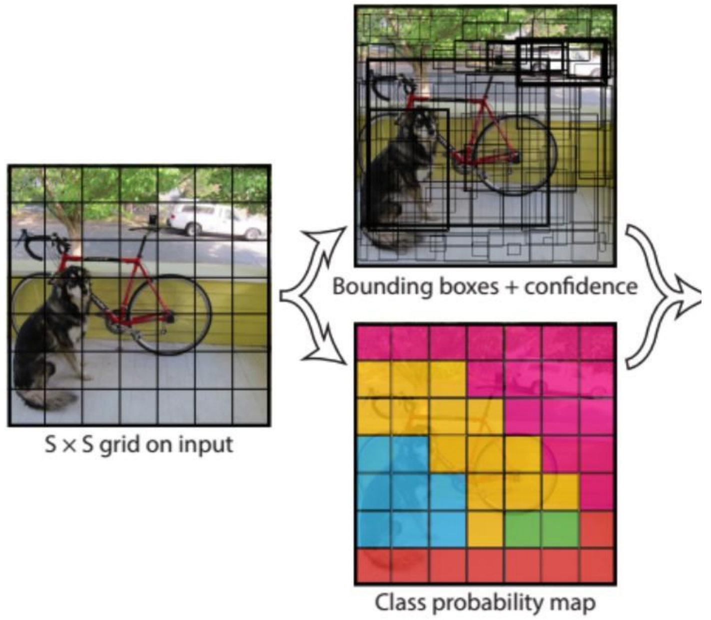


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

RECENT CNNS: YOLO (BOCHKOVSKIY ET AL.(2020))



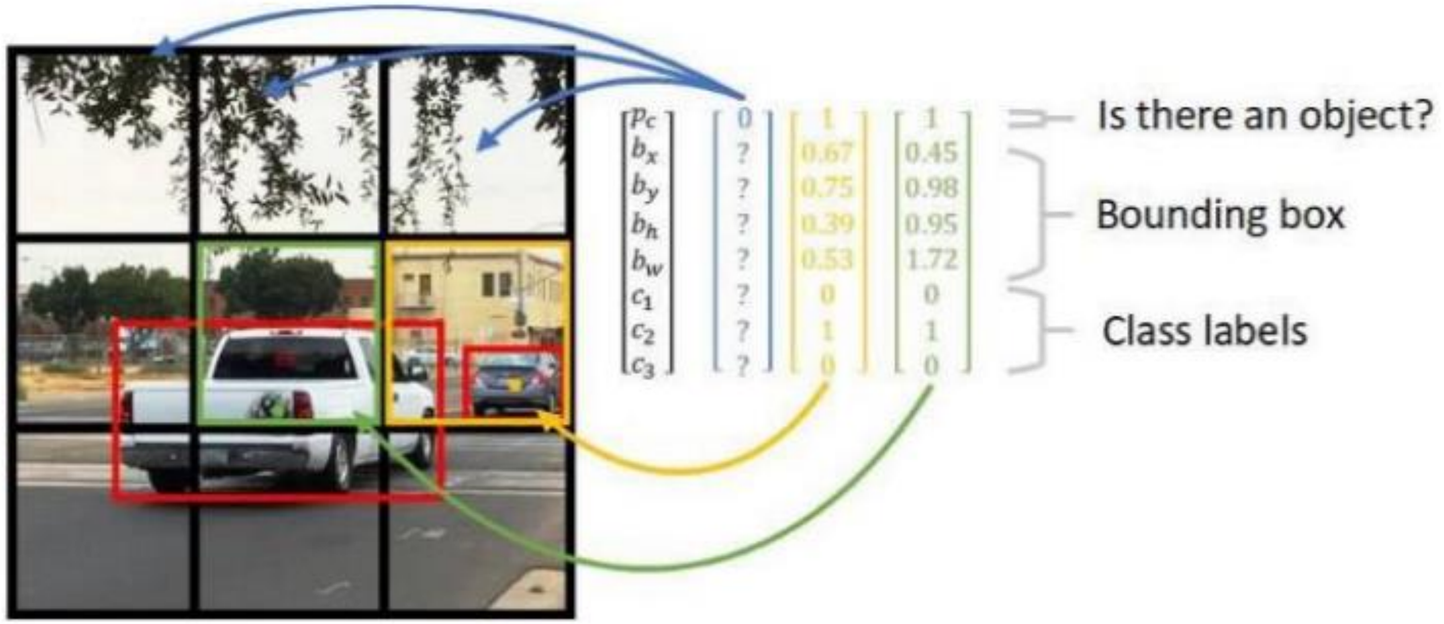
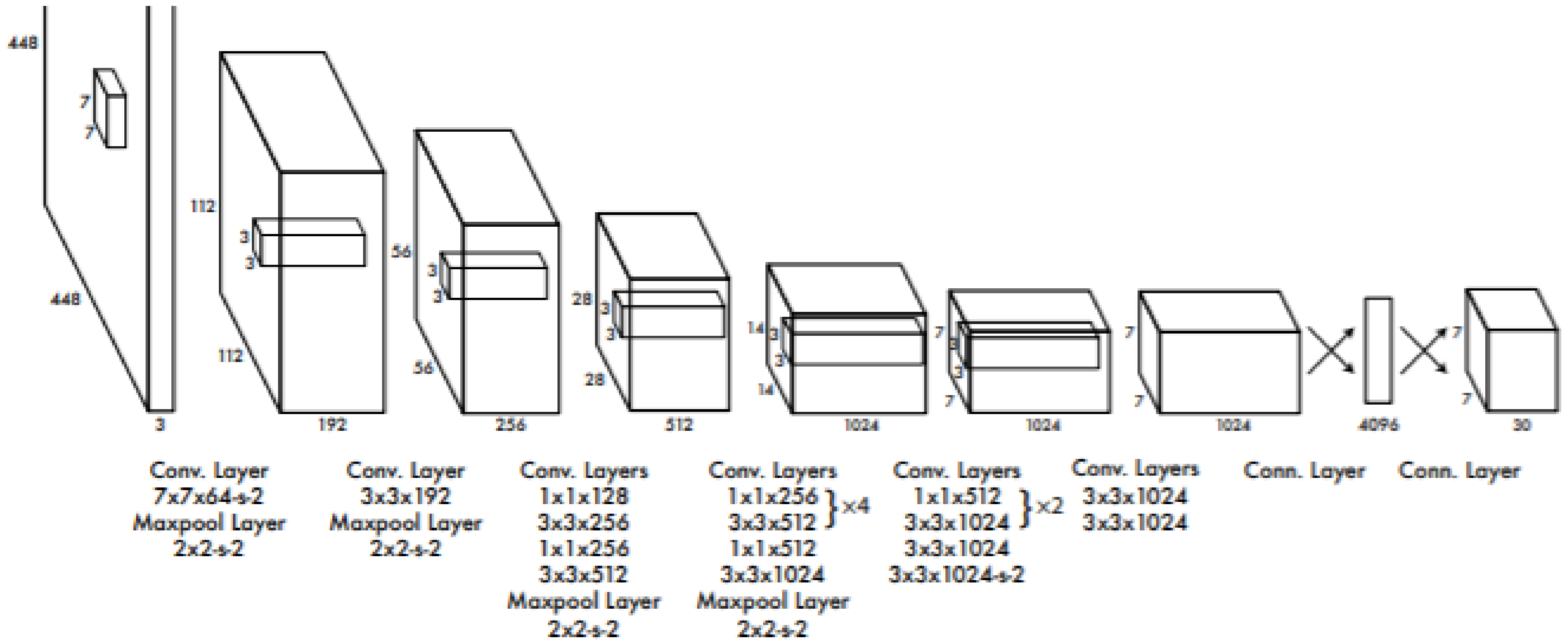


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

YOLO: THE ARCHITECTURE



YOLO: RESULTS

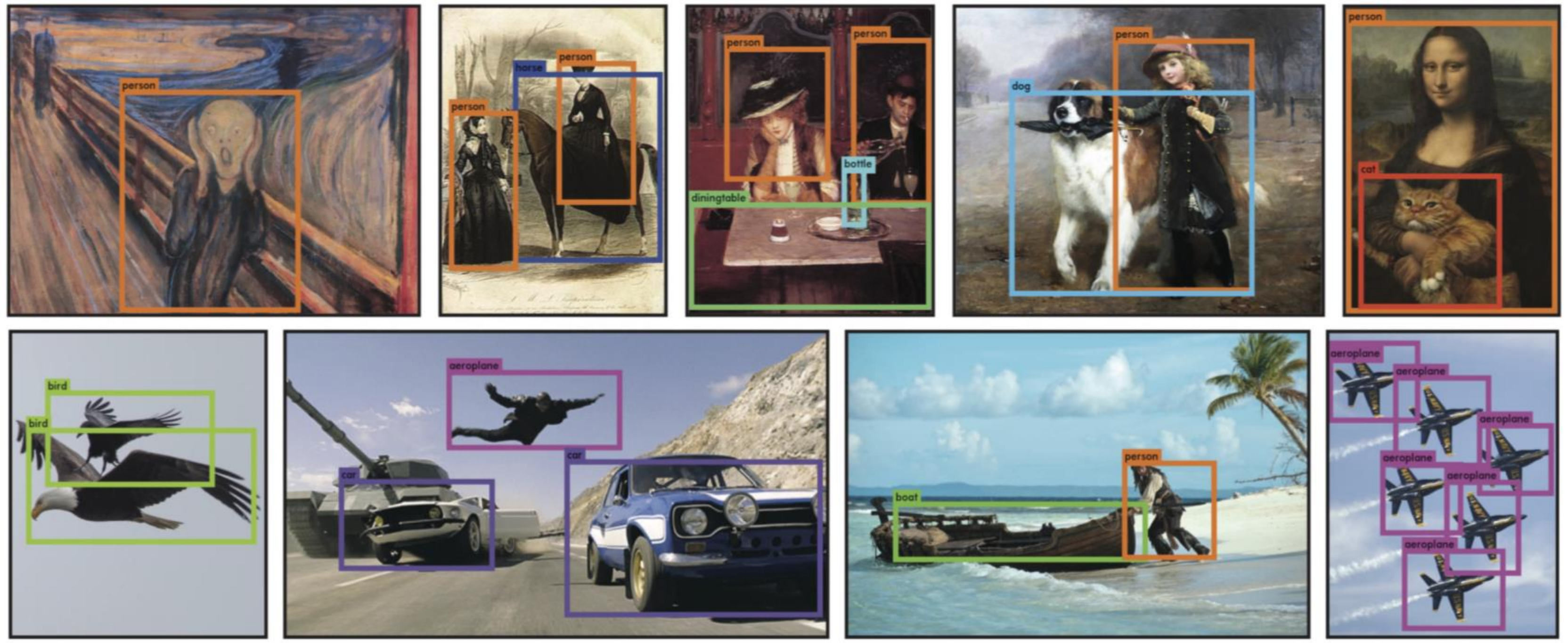


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

RETI NEURALI

LE RETI RICORRENTI

RECURRENT NEURAL NETWORKS

For example, consider the classical form of a dynamical system:

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}), \tag{10.1}$$

where $\boldsymbol{s}^{(t)}$ is called the state of the system.

Equation 10.1 is recurrent because the definition of $\boldsymbol{s}$ at time t refers back to the same definition at time $t-1$.

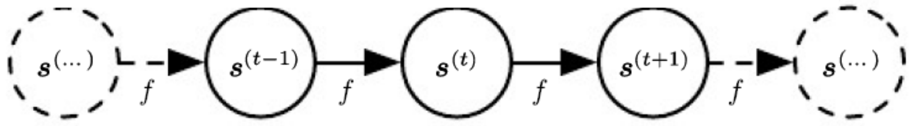
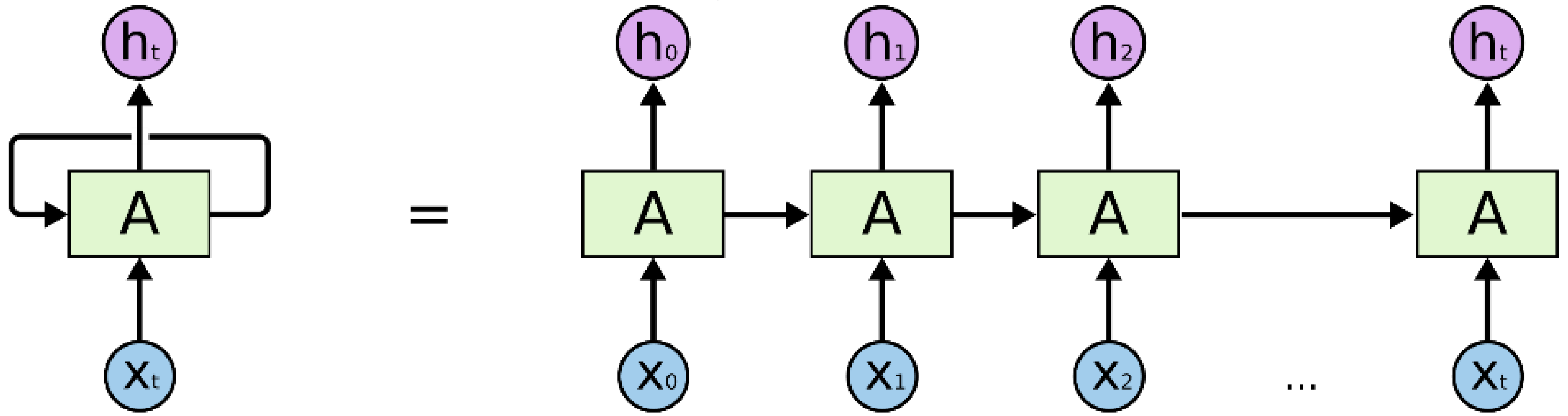


Figure 10.1: The classical dynamical system described by equation 10.1, illustrated as an unfolded computational graph. Each node represents the state at some time t, and the function f maps the state at t to the state at $t+1$. The same parameters (the same value of $\boldsymbol{\theta}$ used to parametrize f) are used for all time steps.



TRAINING A RNN

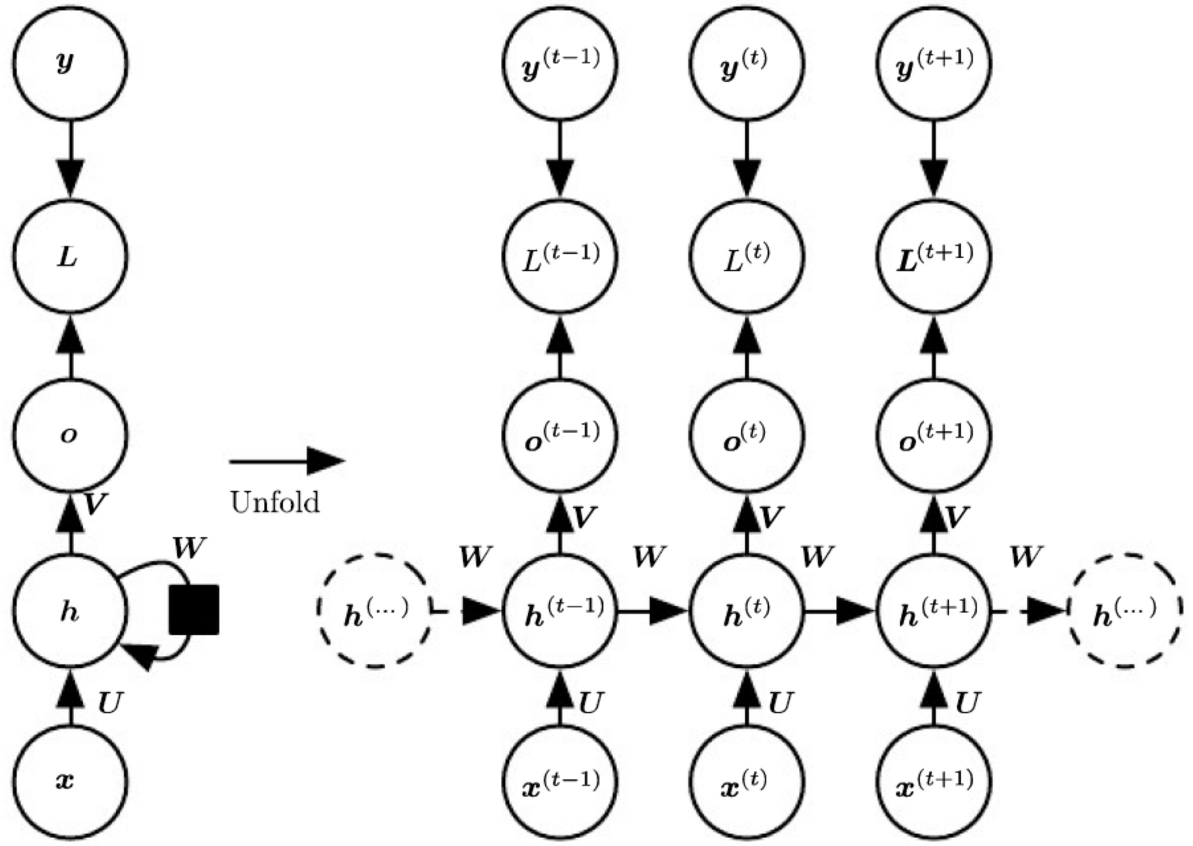
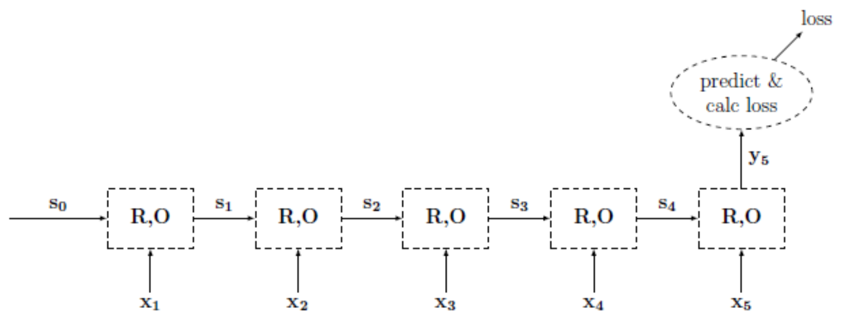


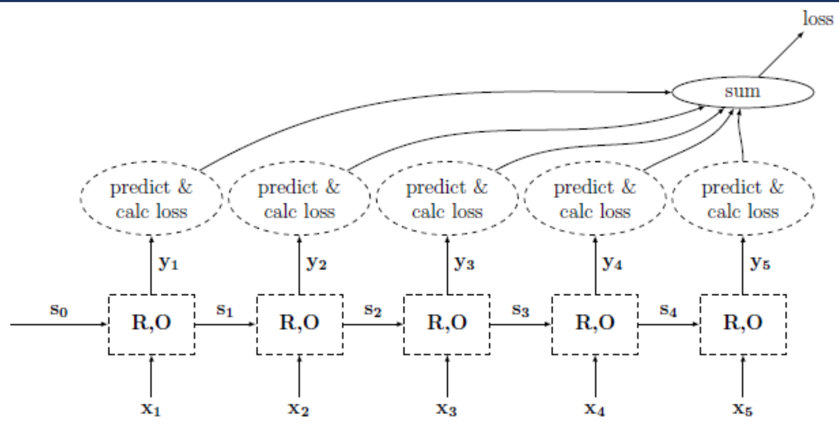
Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation 10.8 defines forward propagation in this model. *(Left)* The RNN and its loss drawn with recurrent connections. *(Right)* The same seen as a time-unfolded computational graph, where each node is now associated with one particular time instance.

TYPES OF RNNS



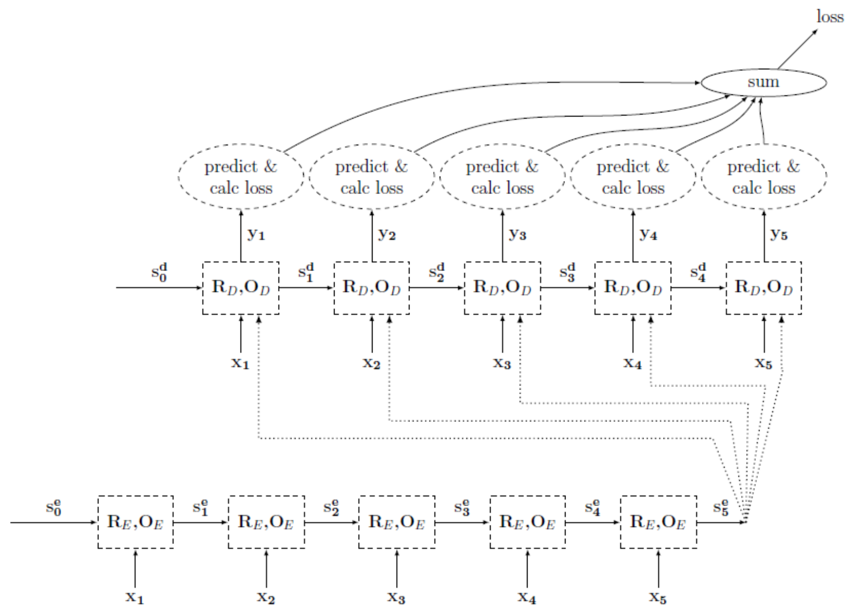
loss
predict & calc loss
y_5
s_0 R,O s_1 R,O s_2 R,O s_3 R,O s_4 R,O
x_1 x_2 x_3 x_4 x_5

Figure 7: Acceptor RNN Training Graph.



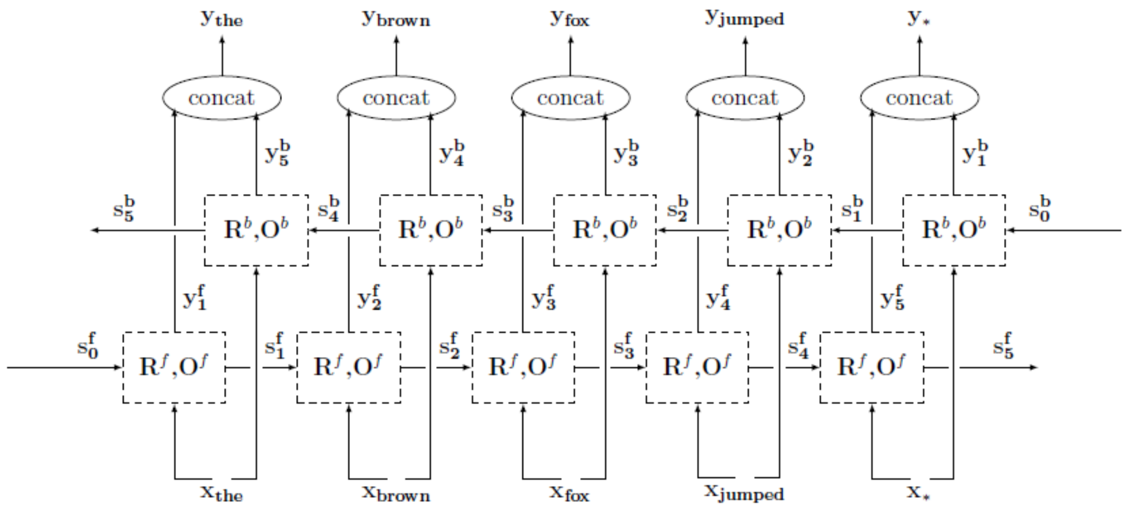
loss
sum
predict & calc loss
y_1 y_2 y_3 y_4 y_5
s_0 R,O s_1 R,O s_2 R,O s_3 R,O s_4 R,O
x_1 x_2 x_3 x_4 x_5

Figure 8: Transducer RNN Training Graph.



loss
sum
predict & calc loss
y_1 y_2 y_3 y_4 y_5
s^d_0 R_D,O_D s^d_1 R_D,O_D s^d_2 R_D,O_D s^d_3 R_D,O_D s^d_4 R_D,O_D
x_1 x_2 x_3 x_4 x_5
s^e_0 R_E,O_E s^e_1 R_E,O_E s^e_2 R_E,O_E s^e_3 R_E,O_E s^e_4 R_E,O_E s^e_5
x_1 x_2 x_3 x_4 x_5

Figure 9: Encoder-Decoder RNN Training Graph.



y_{the} y_{brown} y_{fox} y_{jumped} y_*
concat
y^b_5 y^b_4 y^b_3 y^b_2 y^b_1
s^b_5 R^b,O^b s^b_4 R^b,O^b s^b_3 R^b,O^b s^b_2 R^b,O^b s^b_1 R^b,O^b s^b_0
y^f_1 y^f_2 y^f_3 y^f_4 y^f_5
s^f_0 R^f,O^f s^f_1 R^f,O^f s^f_2 R^f,O^f s^f_3 R^f,O^f s^f_4 R^f,O^f s^f_5
x_{the} x_{brown} x_{fox} x_{jumped} x_*

Figure 11: biRNN over the sentence "the brown fox jumped .".

HTTPS://GITHUB.COM/MICROSOFT/CNTK/WIKI/HANDS-ON-LABS-LANGUAGE-UNDERSTANDING

Task: Slot tagging with an LSTM

```
|# show           # O
|# fligh          # O
|# from           # O
|# burba          # B-fromloc.city_name
|# to             # O
|# st.            # B-toloc.city_name
|# louis          # I-toloc.city_name
|# on             # O
|# monda          # B-depart_date.day_name
```

[HTTPS://GITHUB.COM/MICROSOFT/CNTK/WIKI/HANDS-ON-LABS-LANGUAGE-UNDERSTANDING](https://github.com/microsoft/cntk/wiki/hands-on-labs-language-understanding)

Task: Slot tagging with an LSTM

```
19  |x 178:1 |# BOS      |y 128:1 |# O
19  |x 770:1 |# show     |y 128:1 |# O
19  |x 429:1 |# flights  |y 128:1 |# O
19  |x 444:1 |# from     |y 128:1 |# O
19  |x 272:1 |# burbank  |y 48:1  |# B-fromloc.city_name
19  |x 851:1 |# to       |y 128:1 |# O
19  |x 789:1 |# st.      |y 78:1  |# B-toloc.city_name
19  |x 564:1 |# louis    |y 125:1 |# I-toloc.city_name
19  |x 654:1 |# on       |y 128:1 |# O
19  |x 601:1 |# monday   |y 26:1  |# B-depart_date.day_name
19  |x 179:1 |# EOS      |y 128:1 |# O
```

```
    ^
    |
+-------+
| Dense |
+-------+
    ^
    |
+------+
| LSTM |
+------+
    ^
    |
+-------+
| Embed |
+-------+
    ^
    |
```

[HTTPS://GITHUB.COM/MICROSOFT/CNTK/WIKI/HANDS-ON-LABS-LANGUAGE-UNDERSTANDING](https://github.com/microsoft/CNTK/wiki/Hands-On-Labs-Language-Understanding)

Task: Slot tagging with an LSTM

```
19  |x 178:1 |# BOS       |y 128:1 |# O
19  |x 770:1 |# show      |y 128:1 |# O
19  |x 429:1 |# flights   |y 128:1 |# O
19  |x 444:1 |# from      |y 128:1 |# O
19  |x 272:1 |# burbank   |y 48:1  |# B-fromloc.city_name
19  |x 851:1 |# to        |y 128:1 |# O
19  |x 789:1 |# st.       |y 78:1  |# B-toloc.city_name
19  |x 564:1 |# louis     |y 125:1 |# I-toloc.city_name
19  |x 654:1 |# on        |y 128:1 |# O
19  |x 601:1 |# monday    |y 26:1  |# B-depart_date.day_name
19  |x 179:1 |# EOS       |y 128:1 |# O
```

```
y       "O"        "O"        "O"        "O"  "B-fromloc.city_name"
         ^          ^          ^          ^          ^
         |          |          |          |          |
     +-------+  +-------+  +-------+  +-------+  +-------+
     | Dense |  | Dense |  | Dense |  | Dense |  | Dense |  ...
     +-------+  +-------+  +-------+  +-------+  +-------+
         ^          ^          ^          ^          ^
         |          |          |          |          |
     +------+   +------+   +------+   +------+   +------+
0 -->| LSTM |-->| LSTM |-->| LSTM |-->| LSTM |-->| LSTM |-->...
     +------+   +------+   +------+   +------+   +------+
         ^          ^          ^          ^          ^
         |          |          |          |          |
     +-------+  +-------+  +-------+  +-------+  +-------+
     | Embed |  | Embed |  | Embed |  | Embed |  | Embed |  ...
     +-------+  +-------+  +-------+  +-------+  +-------+
         ^          ^          ^          ^          ^
         |          |          |          |          |
x ------>+--------->+--------->+--------->+--------->+------...
        BOS      "show"    "flights"    "from"   "burbank"
```

APPLICAZIONI DELLE RETI NEURALI

IMMAGINI: IMAGE CLASSIFICATION, OBJECT DETECTION, ENCODING, MAP COLOURING

APPLICAZIONI DELLE RETI NEURALI

TESTI E IMMAGINI: AUTOMATIC CAPTIONING

APPLICAZIONI

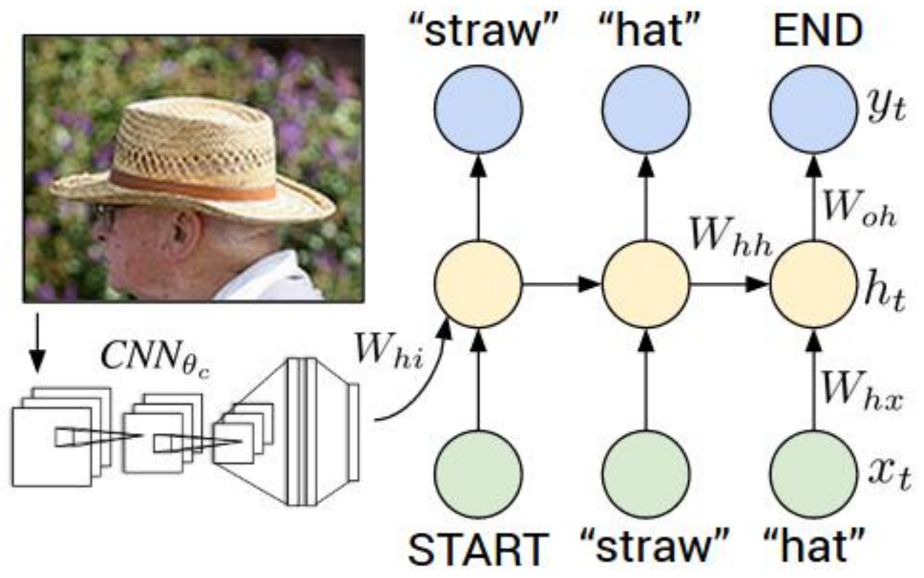
IMAGE RETRIEVAL, VISUAL QUESTION ANSWERING

RETI NEURALI: APPLICAZIONI

ESEMPI ILLUSTRI E USE CASE INDUSTRIALI

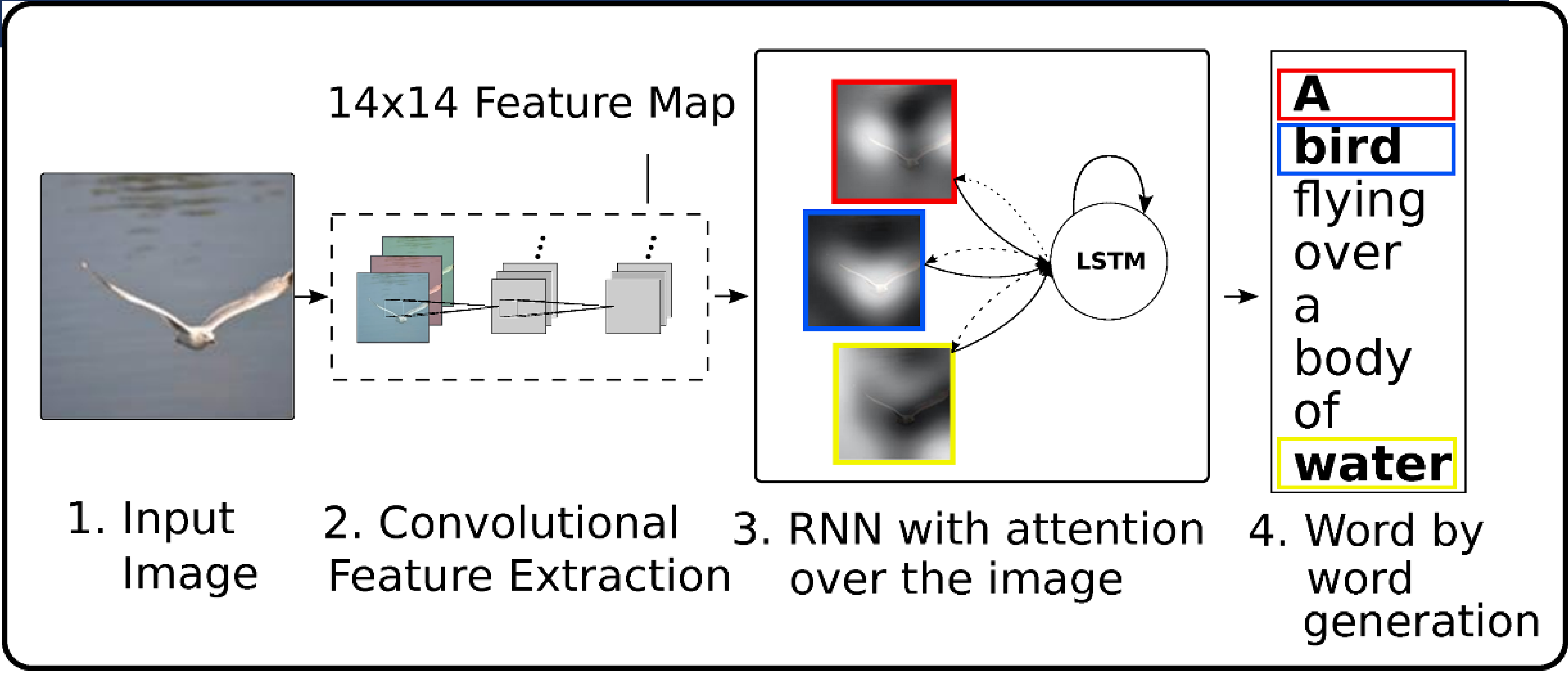
IMAGE CAPTIONING: ADVANCED ARCHITECTURES

- Image to captions
  - Convolutional Neural Network to learn a representation of the image
  - (Bi-directional) Recurrent Neural Network to generate a caption describing the image
    - its input is the representation computed from the CNN
    - its output is a sequence of words, i.e. the caption

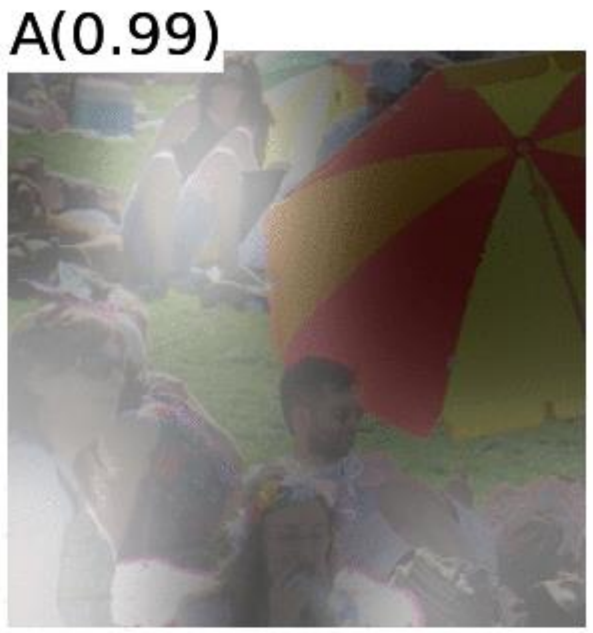


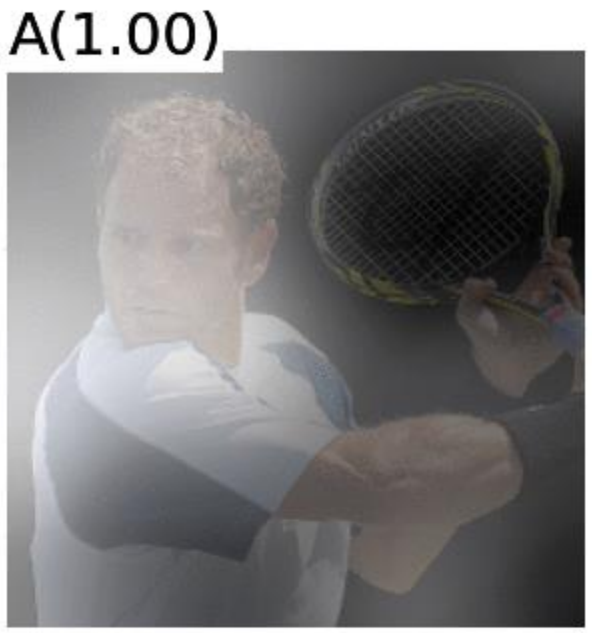
"baseball player is throwing ball in game."

THE ARCHITECTURE

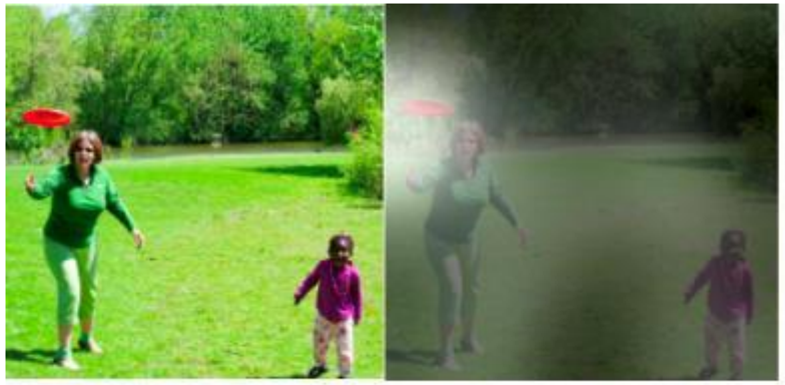

14x14 Feature Map
LSTM
A
bird
flying
over
a
body
of
water
1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

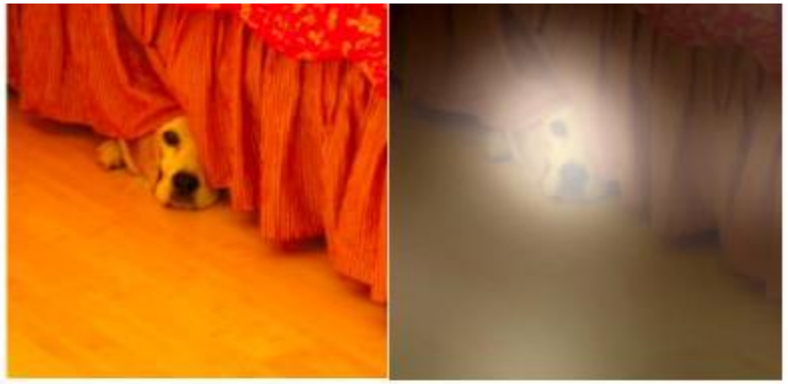
ATTENTION: A BRODGE BETWEEN VISION AND LANGUAGE

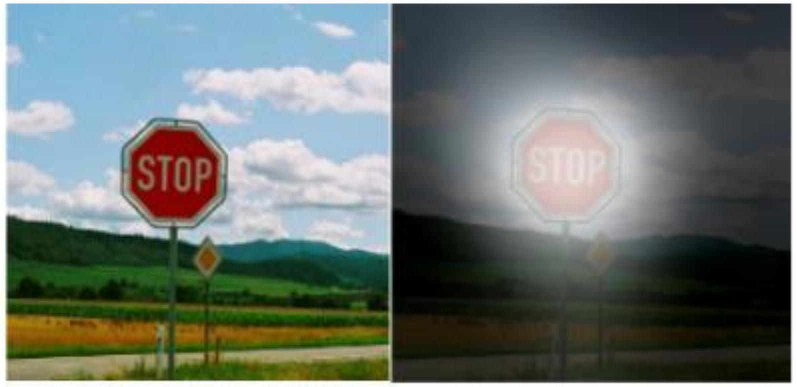




INTEGRATED VISION AND LANGUAGE PROCESSING: IMAGE CAPTIONING AND ATTENTION

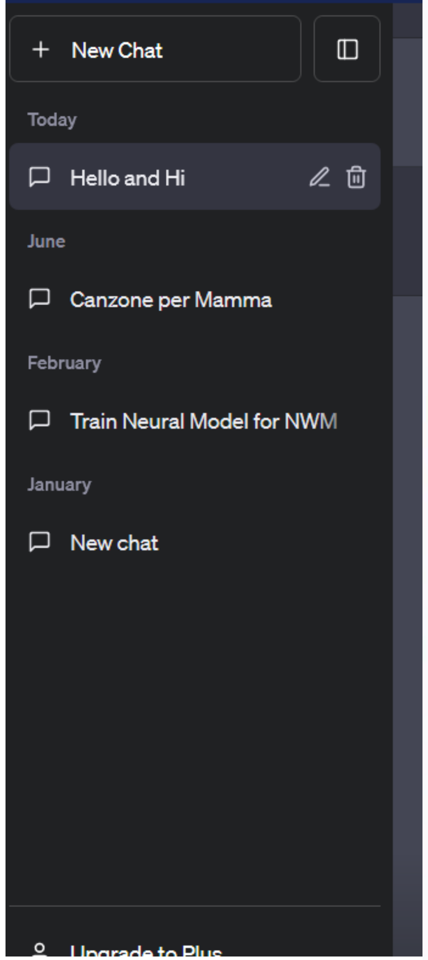

A woman is throwing a frisbee in a park.

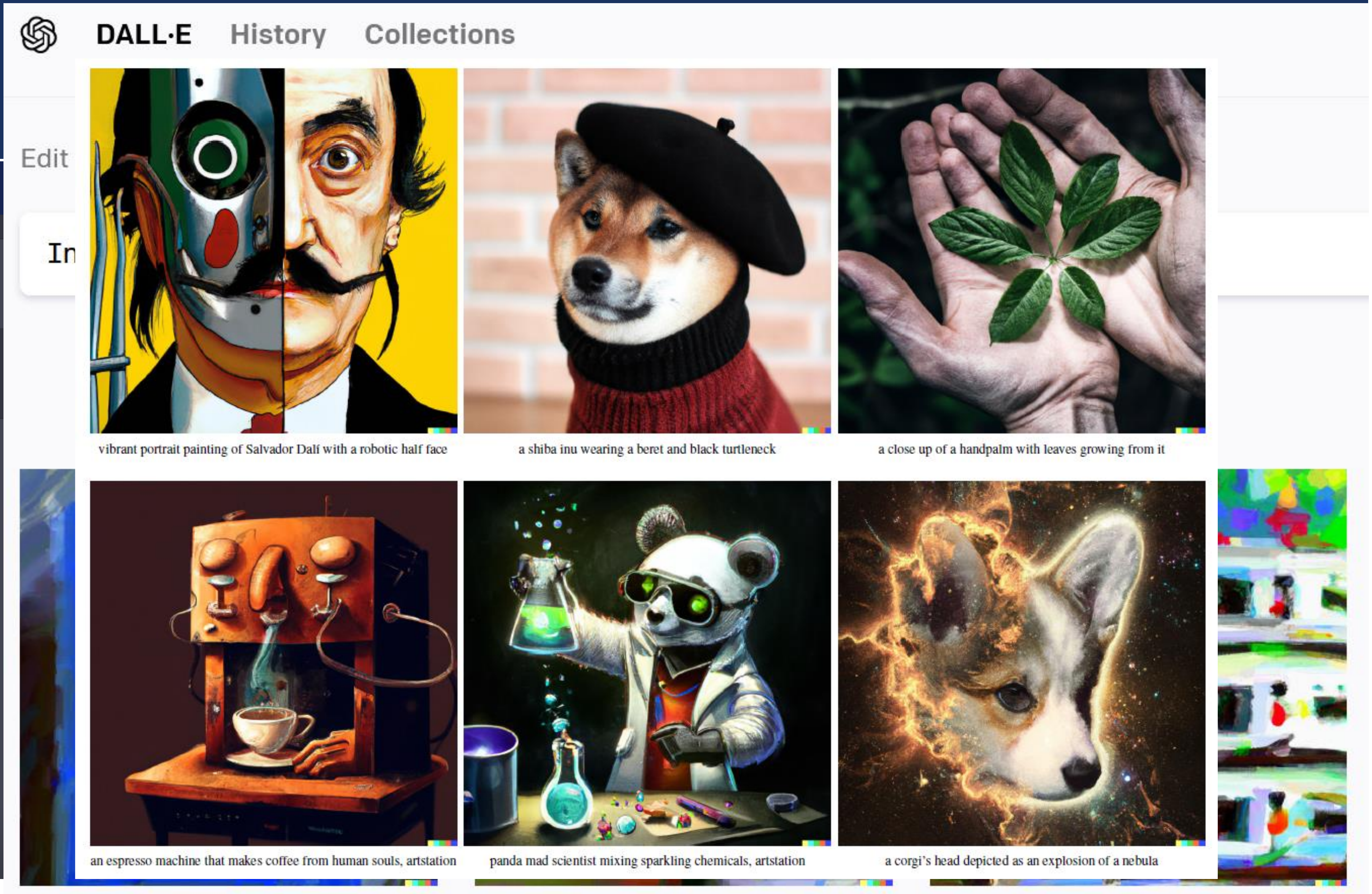

A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.

ESEMPI





vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

a close up of a handpalm with leaves growing from it

an espresso machine that makes coffee from human souls, artstation

panda mad scientist mixing sparkling chemicals, artstation

a corgi's head depicted as an explosion of a nebula

NEURAL ENCODING-DECODING FOR DALL-E

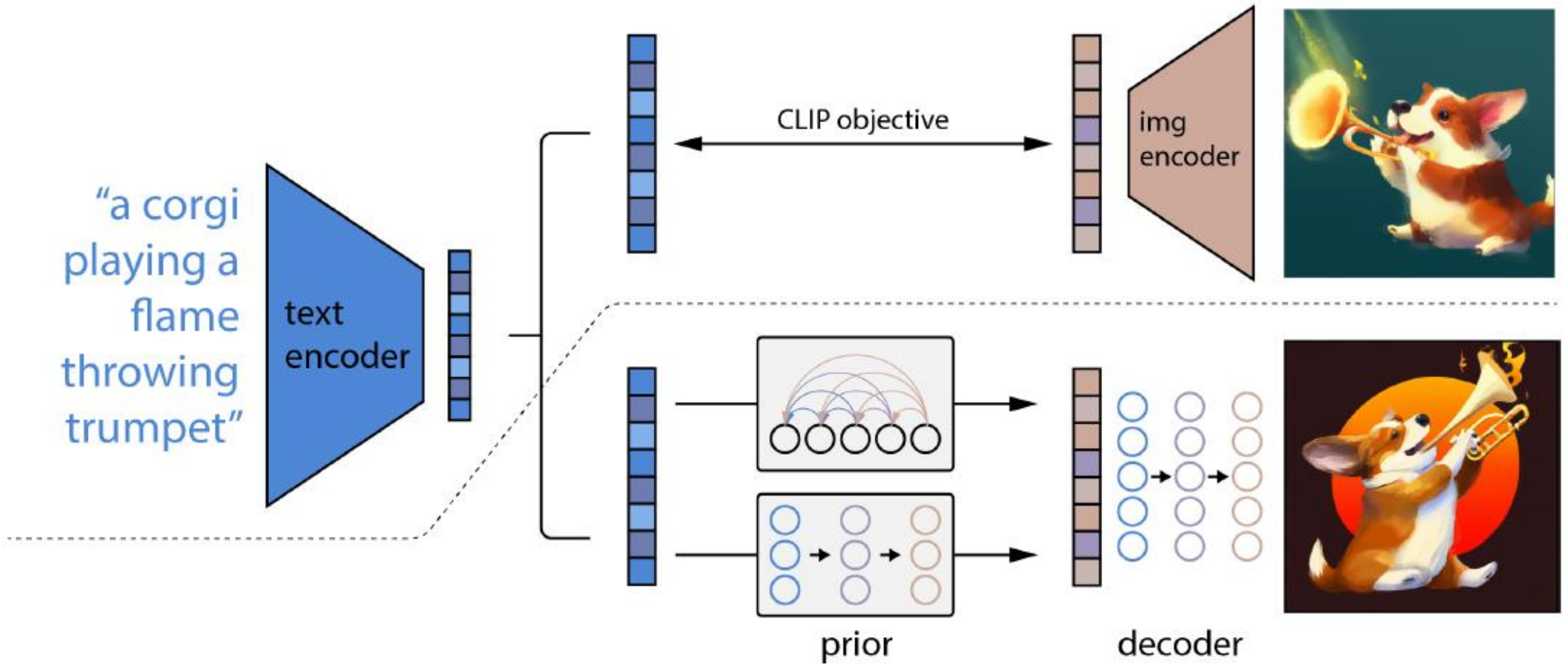
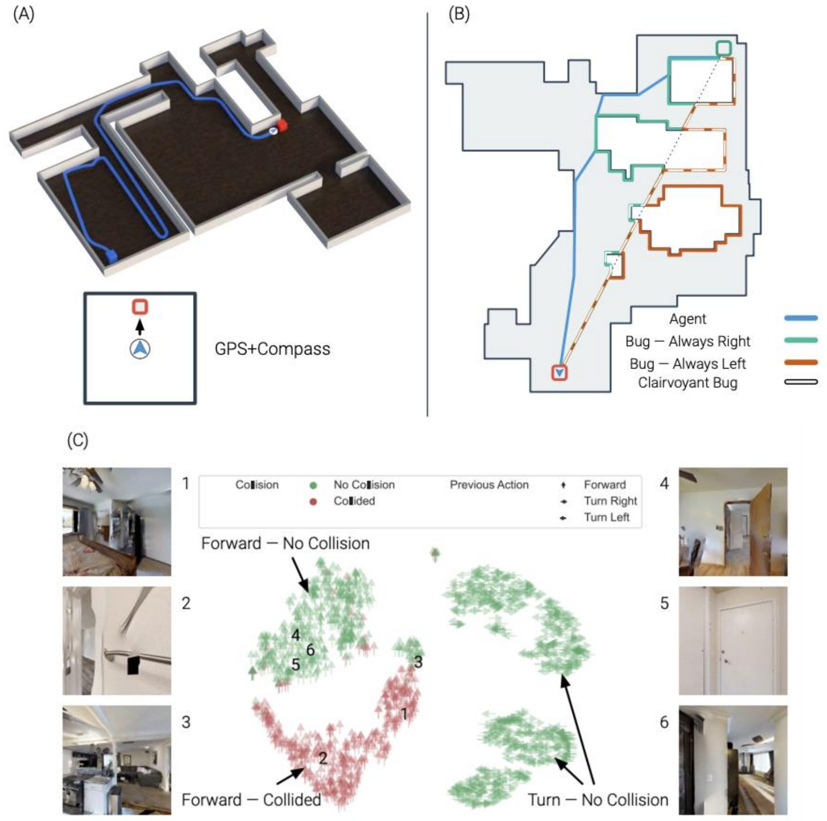


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

The emergence of maps in the memories of blind navigation agents

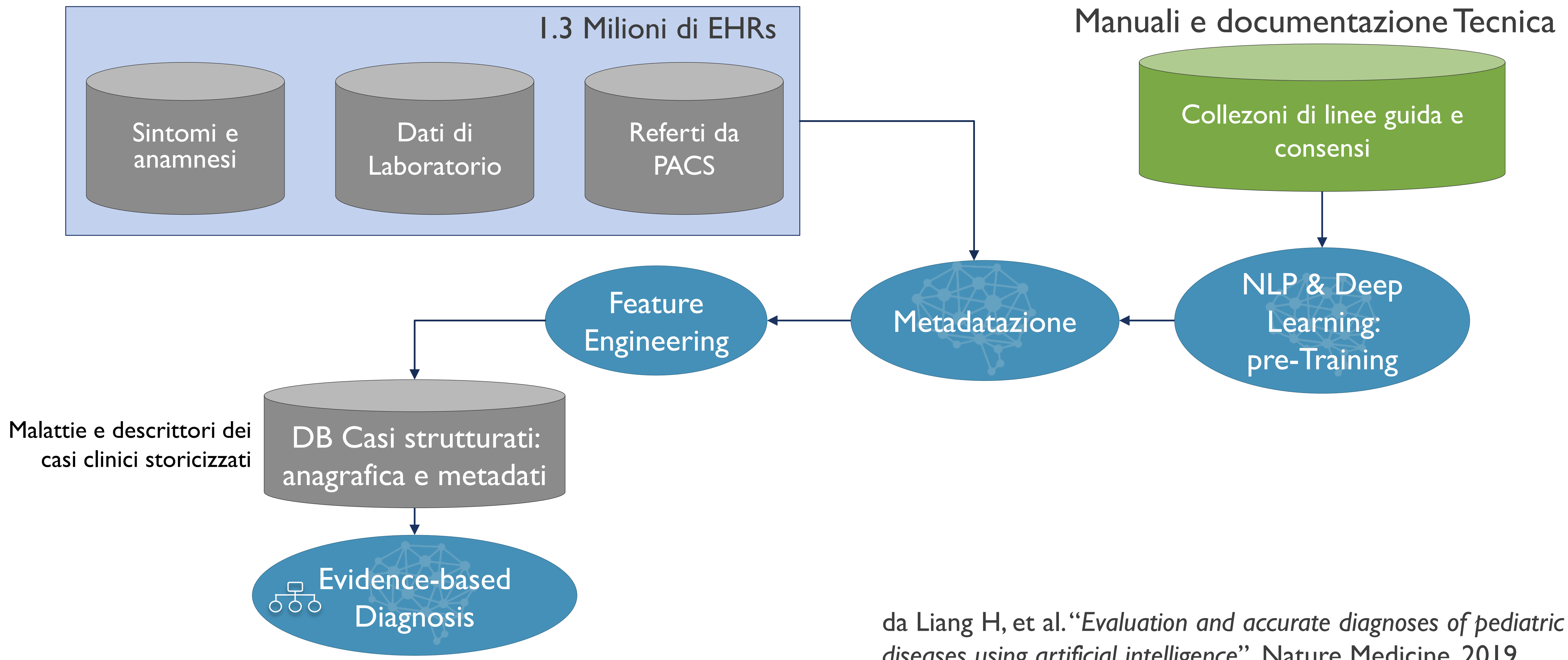
Map-building is an emergent phenomenon in the course of AI agents learning to navigate. It explains why we can feed neural networks images with no explicit maps and can predict navigation policies.

- *The Emergence of Maps in the Memories of Blind Navigation Agents* shows that giving an agent knowledge of only ego-motion (change in agent's location and orientation as it moves) and goal location is sufficient to successfully navigate to the goal. Note that this agent does not have any visual information as input and yet its success rates compared to 'sighted' agents are very similar, only efficiency differs.
- The model doesn't have any inductive bias towards mapping and is trained with on-policy reinforcement learning. The only mechanism that explains this ability is the memory of the LSTM.
- It is possible to reconstruct metric maps and detect collisions solely from the hidden state of this agent.



Georgia Tech Meta Google

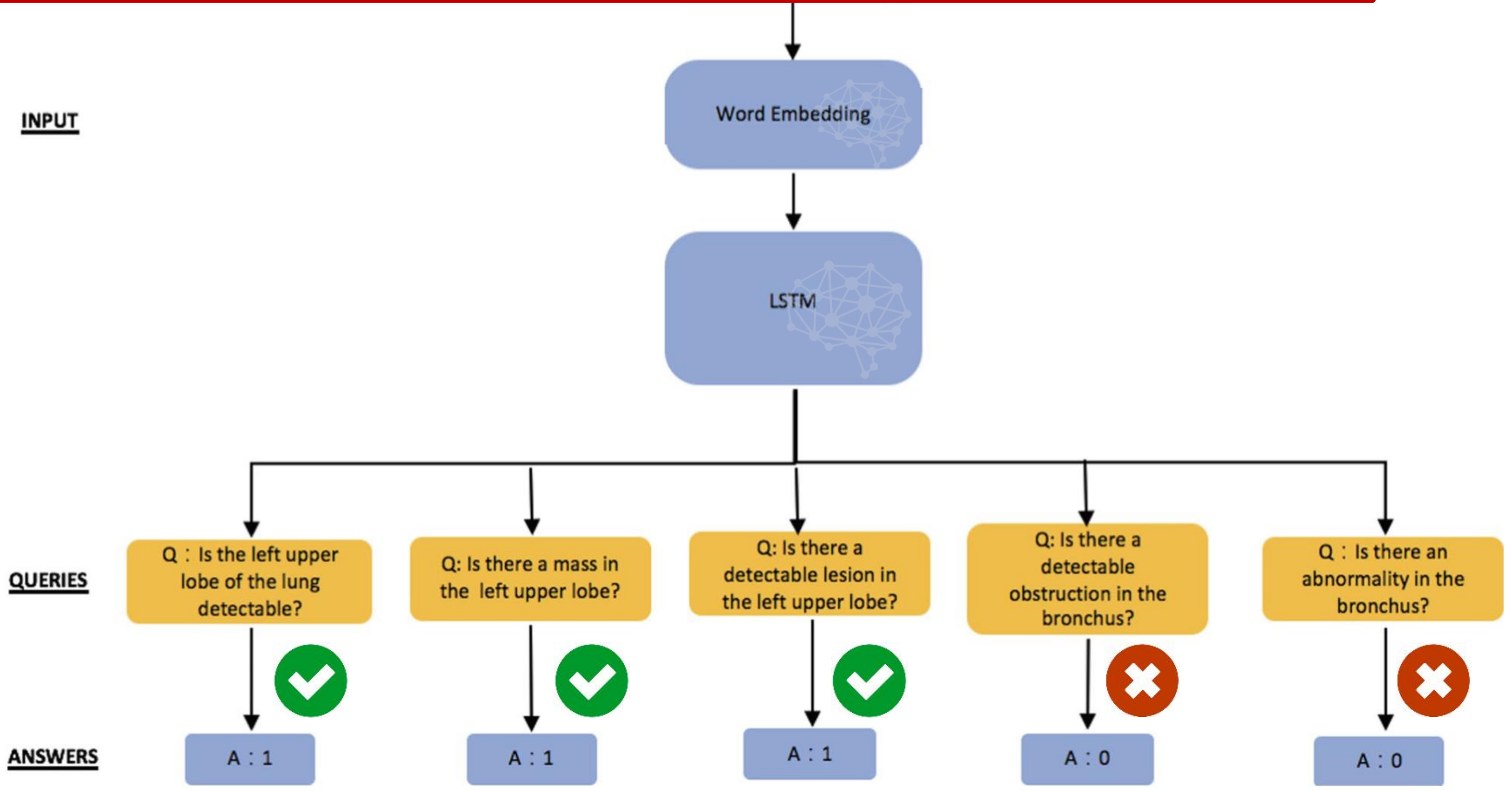
DIAGNOSI MALATTIE PEDIATRICHE: UN WORKFLOW ORIENTATO AL ML


1.3 Milioni di EHRs
Sintomi e anamnesi
Dati di Laboratorio
Referti da PACS
Manuali e documentazione Tecnica
Collezoni di linee guida e consensi
NLP & Deep Learning: pre-Training
Metadatazione
Feature Engineering
Malattie e descrittori dei casi clinici storicizzati
DB Casi strutturati: anagrafica e metadati
Evidence-based Diagnosis

da Liang H, et al. "*Evaluation and accurate diagnoses of pediatric diseases using artificial intelligence*", Nature Medicine, 2019

MEDICAL INFORMATION EXTRACTION

INPUT: "Si osserva una lesione nel lobo superiore sinistro del polmone del paziente . …"


INPUT
Word Embedding
LSTM
QUERIES
Q : Is the left upper lobe of the lung detectable?
Q: Is there a mass in the left upper lobe?
Q: Is there a detectable lesion in the left upper lobe?
Q: Is there a detectable obstruction in the bronchus?
Q : Is there an abnormality in the bronchus?
ANSWERS
A : 1
A : 1
A : 1
A : 0
A : 0

EVIDENCE BASED DIAGNOSIS: RISULTATI (11,926 PAZIENTI)

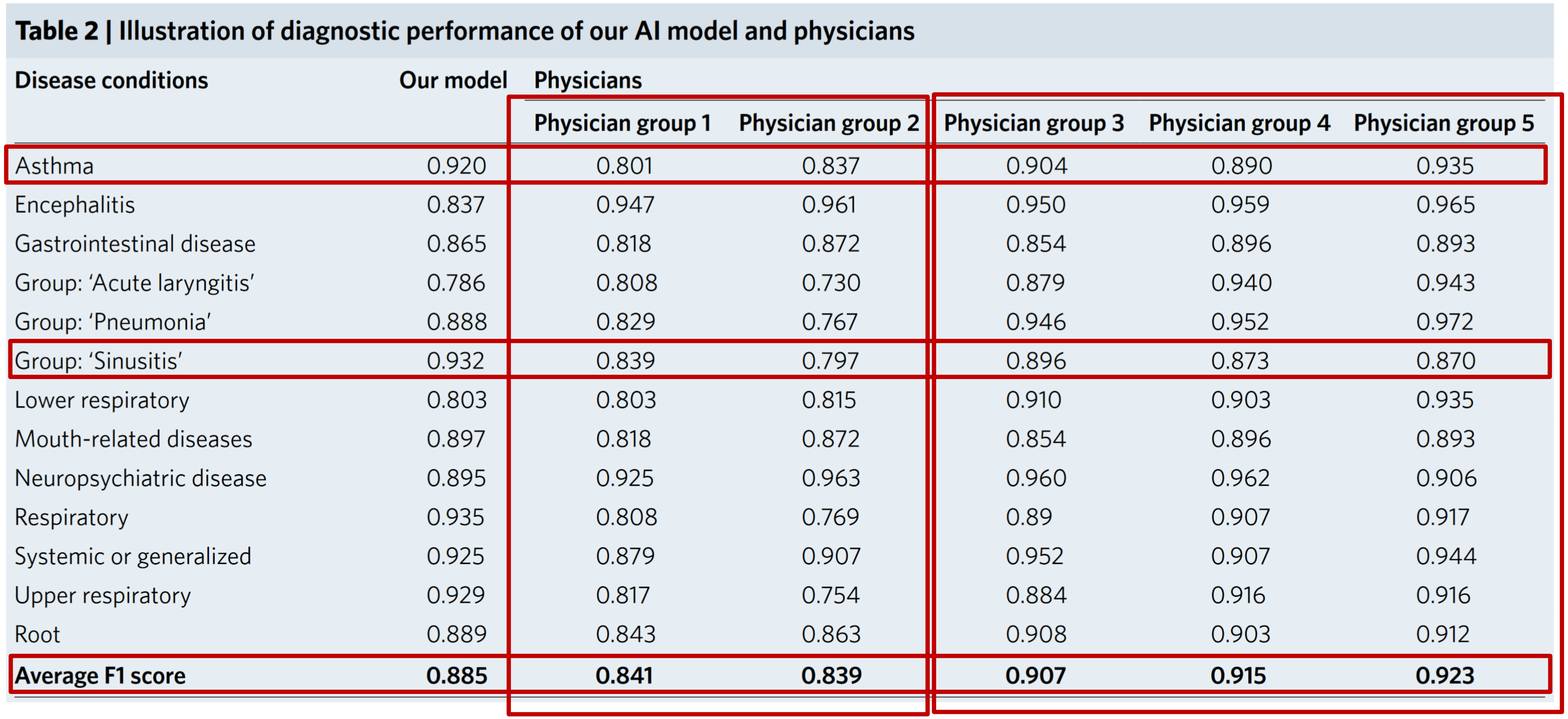
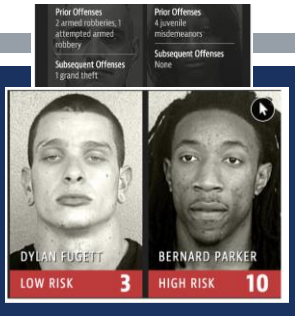


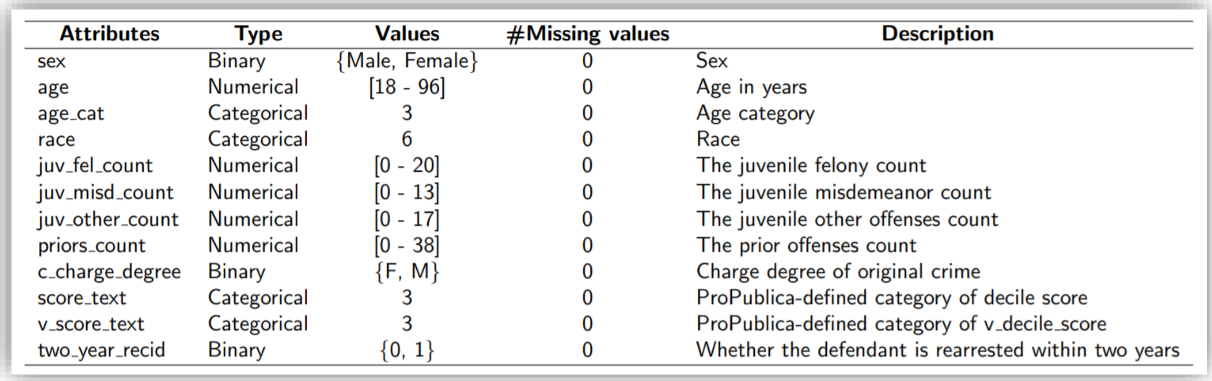
Table 2 | Illustration of diagnostic performance of our AI model and physicians

Disease conditions	**Our model**	**Physicians**				
		Physician group 1	**Physician group 2**	**Physician group 3**	**Physician group 4**	**Physician group 5**
Asthma	0.920	0.801	0.837	0.904	0.890	0.935
Encephalitis	0.837	0.947	0.961	0.950	0.959	0.965
Gastrointestinal disease	0.865	0.818	0.872	0.854	0.896	0.893
Group: 'Acute laryngitis'	0.786	0.808	0.730	0.879	0.940	0.943
Group: 'Pneumonia'	0.888	0.829	0.767	0.946	0.952	0.972
Group: 'Sinusitis'	0.932	0.839	0.797	0.896	0.873	0.870
Lower respiratory	0.803	0.803	0.815	0.910	0.903	0.935
Mouth-related diseases	0.897	0.818	0.872	0.854	0.896	0.893
Neuropsychiatric disease	0.895	0.925	0.963	0.960	0.962	0.906
Respiratory	0.935	0.808	0.769	0.89	0.907	0.917
Systemic or generalized	0.925	0.879	0.907	0.952	0.907	0.944
Upper respiratory	0.929	0.817	0.754	0.884	0.916	0.916
Root	0.889	0.843	0.863	0.908	0.903	0.912
Average F1 score	**0.885**	**0.841**	**0.839**	**0.907**	**0.915**	**0.923**

COMPAS: PROFILING



- COMPAS dataset (*Correctional Offender Management Profiling for Alternative Sanctions*)
  - raccoglie dati nell'ambito della giustizia penale utilizzati per prevedere il rischio di recidiva di un imputato.
  - pubblicato da ProPublica nel 2016 sulla base dei dati raccolti dalla contea di Broward.



Attributes	Type	Values	#Missing values	Description
sex	Binary	{Male, Female}	0	Sex
age	Numerical	[18 - 96]	0	Age in years
age_cat	Categorical	3	0	Age category
race	Categorical	6	0	Race
juv_fel_count	Numerical	[0 - 20]	0	The juvenile felony count
juv_misd_count	Numerical	[0 - 13]	0	The juvenile misdemeanor count
juv_other_count	Numerical	[0 - 17]	0	The juvenile other offenses count
priors_count	Numerical	[0 - 38]	0	The prior offenses count
c_charge_degree	Binary	{F, M}	0	Charge degree of original crime
score_text	Categorical	3	0	ProPublica-defined category of decile score
v_score_text	Categorical	3	0	ProPublica-defined category of v_decile_score
two_year_recid	Binary	{0, 1}	0	Whether the defendant is rearrested within two years

Caratteristiche Contiene 7.214 istanze. Ogni imputato è descritto da 52 attributi (31 categorici, 6 binari, 14 numerici e un attributo nullo)

Task L'obiettivo è **prevedere se un individuo viene nuovamente arrestato entro due anni dal primo arresto**

Possibili rischi
Alcuni gruppi sociali (gli afroamericani) hanno maggiori probabilità di essere erroneamente etichettati come a rischio più elevato rispetto agli altri (i caucasici).
Eticamente ingiusto.
Obbiettivo: ottenere un sistema equo tra gruppi sociali diversi.

https://github.com/propublica/compas-analysis

REFERENCES: EARLY MACHINE LEARNING

- Quillian, M. R. (1968). Semantic memory. In Minsky, M. L., editor, Semantic Information Processing, pages 216-270. MIT Press, Cambridge, Massachusetts.
- Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1:81-106.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, Parallel Distributed Processing, volume 1, chapter 8, pages 318-362. MIT Press, Cambridge, Massachusetts.
- Russell S., Norvig, P., Artificial Intelligence, A modern Approach, the intelligent agent book. Prentice Hall, 2003-2020
- Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, http://www.deeplearningbook.org, 2016.

BIBLIOGRAFIA: AN HISTORICAL OVERVIEW ON NNS

- Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115{133, 1943.
- Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386, 1958.
- Donald Olding Hebb. The organization of behavior: A neuropsychological theory. Psychology Press, 1949.
- John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences, 79(8):2554-2558, 1982.
- David E Rumelhart, Georey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.
- Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, Cambridge.
- Teuvo Kohonen. The self-organizing map. Proceedings of the IEEE, 78(9):1464{1480, 1990.
- David H Ackley, Georey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. Cognitive science, 9(1):147-169, 1985.
- Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaected by shift in position. Biological cybernetics, 36(4): 193-202, 1980.
- Le Cun B. Boser, John S. Denker, D. Henderson, Richard E. Howard, W. Hubbard and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems. Citeseer, 1990.

BIBLIOGRAFIA: AN HISTORICAL OVERVIEW ON NNS (2)

- Michael I Jordan. Serial order: A parallel distributed processing approach. Advances in psychology, 121:471-495, 1986.
- Jerey L Elman. Finding structure in time. Cognitive science, 14(2):179-211, 1990.
- AJ Robinson and Frank Fallside. The utility driven dynamic error propagation network. University of Cambridge Department of Engineering, 1987.
- Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673-2681, 1997.
- Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735-1780, 1997.
- Hugo Larochelle and Georey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In Advances in neural information processing systems, pages 1243-1251, 2010
- Denil, M., Bazzani, L., Larochelle, H., and de Freitas, N. (2012). Learning where to attend with deep architectures for image tracking. Neural Computation, 24 (8), 2151–2184
- Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. Advances in neural information processing systems, 19:153, 2007.

GRAZIE DELL’ATTENZIONE

BASILI@INFO.UNIROMA2.IT