TÉCNICO LISBOA

Intelligent Systems

Susana M. Vieira

Universidade de Lisboa, Instituto Superior Técnico

IS4, Center of Intelligent Systems, IDMEC, LAETA, Portugal
{susana.vieira}@tecnico.ulisboa.pt

TÉCNICO LISBOA

Deep Learning

SI7 – Deep Learning

Reading:

- Ian Goodfellow. ***Deep Learning***. MIT Press, 2016.
- François Chollet. ***Deep Learning With Python***. 2nd Edition, 2017.

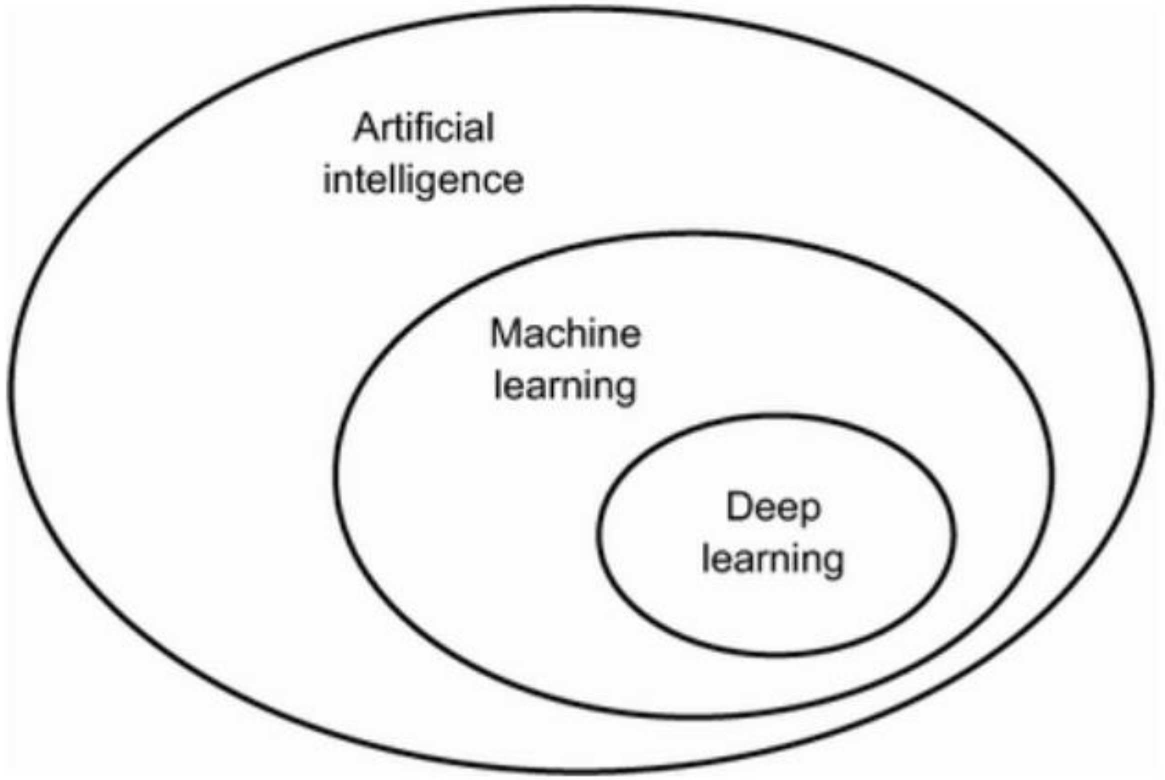
DEEP LEARNING

Outline

- ❑Introduction to Deep Learning
  - what is Deep Learning
  - why is it useful
- ❑Machine Learning basics
- ❑Main components/hyper-parameters:
  - activation functions
  - optimizers, cost functions and training
  - regularization methods
  - tuning
  - classification vs. regression tasks
- ❑DNN basic architectures:
  - Encoder/decoder
  - convolutional
  - recurrent
- ❑Application example: Relation Extraction

What is deep learning?

- AI is: *“the effort to automate intellectual tasks normally performed by humans.”*



What is deep learning?

- 1830s-1840s **Charles Babbage** invented the ***Analytical Engine***
- In 1843, **Ada Lovelace** remarked on the invention of the Analytical Engine:

"The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform. . . . Its province is to assist us in making available what we're already acquainted with."

What is deep learning?

- AI pioneer Alan Turing quoted as "Lady Lovelace's objection" in his landmark 1950 paper "Computing Machinery and Intelligence,"[1] which introduced the *Turing test.*
- Turing was of the opinion that computers could in principle be made to emulate all aspects of human intelligence.

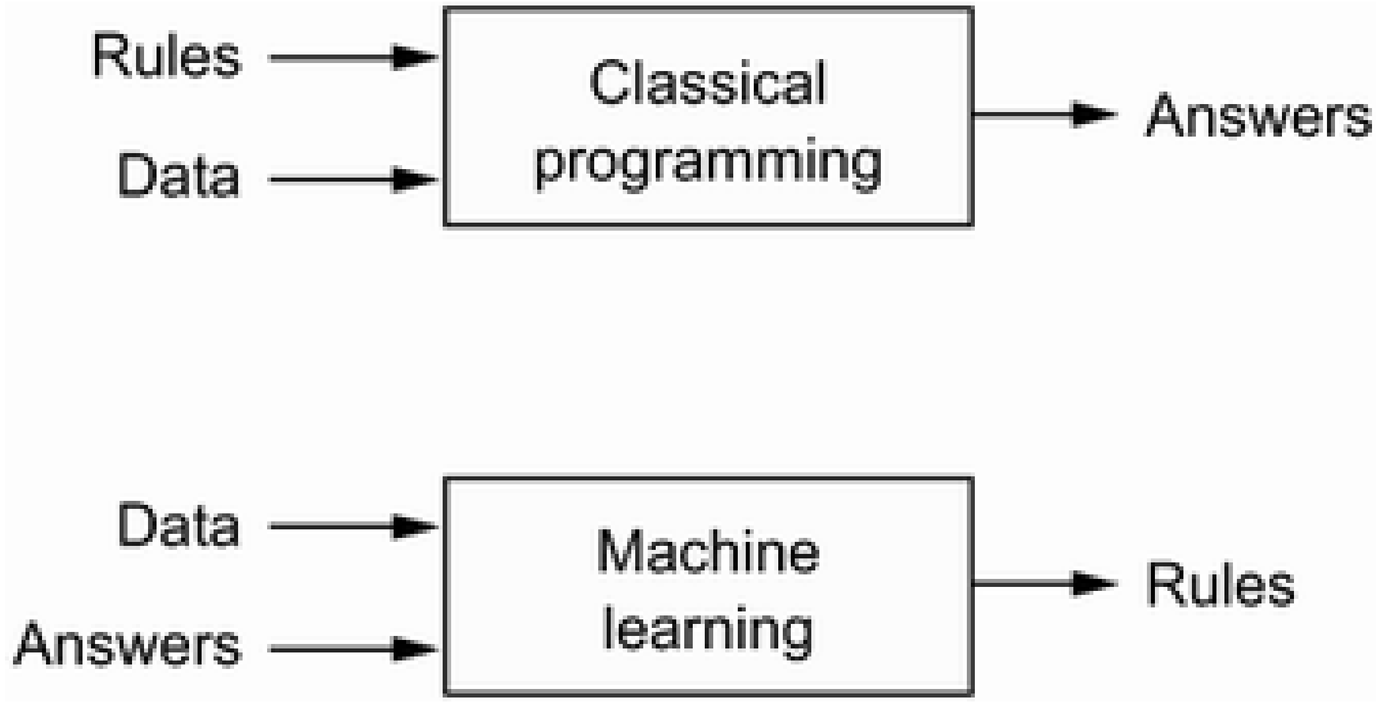
[1] A.M. Turing, "Computing Machinery and Intelligence," *Mind* 59, no. 236 (1950): 433–460.

What is deep learning?

- The usual way to make a computer do useful work is to have a human programmer write down ***rules***—a **computer program**
- Machine learning turns this around: the machine looks at the input data and the corresponding answers, and figures out what the rules should be

What is deep learning?

- A machine learning system is trained rather than explicitly programmed.



What is deep learning?

- Machine learning, and especially deep learning, exhibits little mathematical theory— and is fundamentally an **engineering discipline**.
- Unlike theoretical physics or mathematics, machine learning is a very hands-on field driven by empirical findings and deeply reliant on advances in software and hardware.

What is deep learning?

- To do machine learning, and deep learning we need:
  - ***Input data points***—For instance, if the task is speech recognition, these data points could be sound files of people speaking.
  - ***Examples of the expected output***—In a speech-recognition task, these could be human-generated transcripts of sound files.
  - ***A way to measure whether the algorithm is doing a good job***—determine the distance between the algorithm's current output and expected output. The measurement is used as a feedback signal to adjust the way the algorithm works. This adjustment step is what we call **learning**

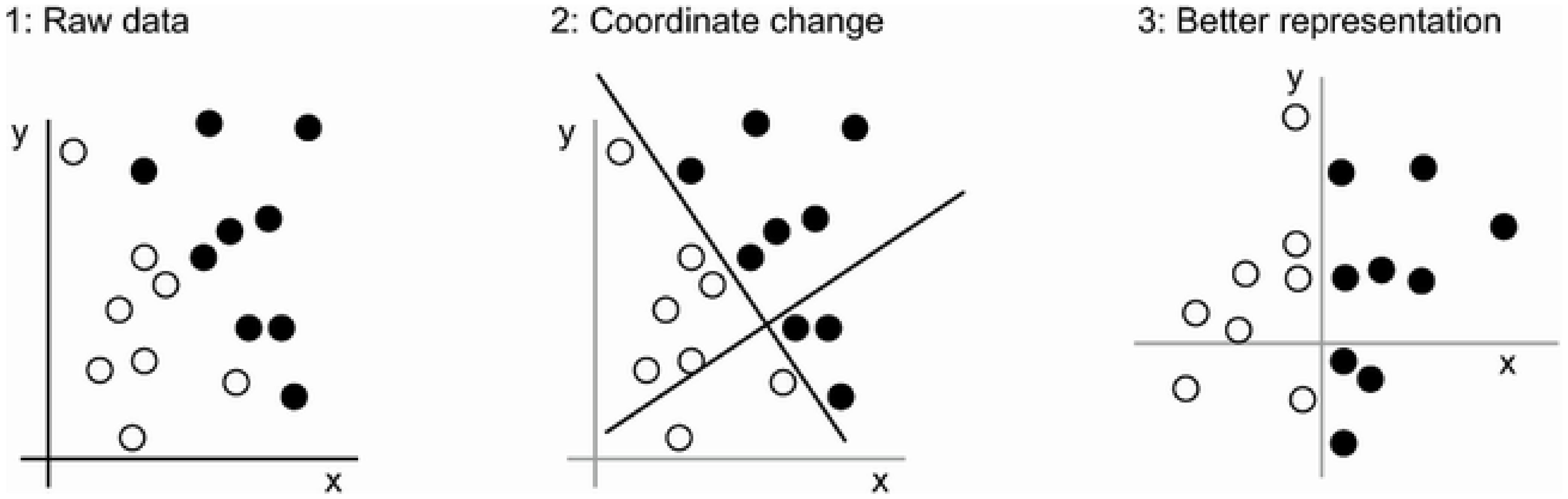
What is deep learning?

- The central problem in machine learning and deep learning is to ***meaningfully transform data***: in other words, to learn useful ***representations*** of the input data at hand—representations that get us closer to the expected output.
- Machine learning models are all about finding appropriate representations for their input data—transformations of the data that make it more amenable to the task at hand.

TÉCNICO LISBOA

What is deep learning?

- Machine learning is finding appropriate ***representations*** for their input data—transformations of the data.



What is deep learning?

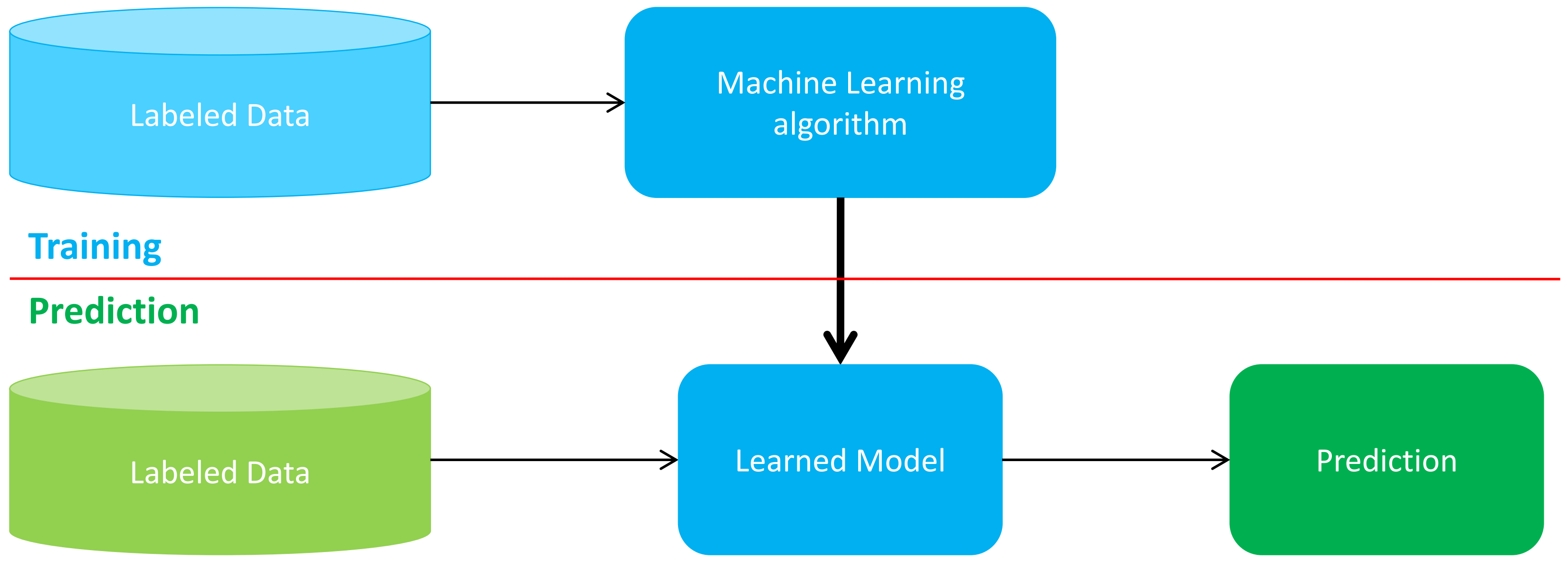
- This is fine for such an extremely simple problem!
- Could you do the same if the task were to classify images of handwritten digits?
- Could you write down explicit, computer-executable image transformations that would illuminate the difference between a 6 and an 8, between a 1 and a 7, across all kinds of different handwriting?

What is deep learning?

- Hardly!
- It will be much easier to automate the process!
- ***Learning***, in the context of machine learning, describes an automatic search process for data transformations that produce useful representations of some data, guided by some feedback signal

Machine Learning Basics

- Machine learning/Computational Intelligence is a field of computer science that gives computers the ability to **learn without being explicitly programmed**

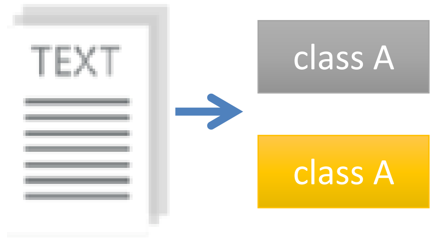


- Methods that can learn from and make predictions on data

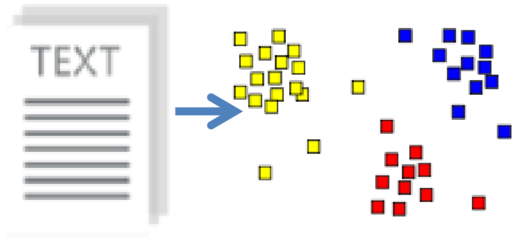
Types of Learning

- **Supervised**: Learning with a **labeled training** set
  - Ex.: email *classification* with labeled emails
- **Unsupervised**: Discover **patterns** in **unlabeled** data
  - Ex.: *cluster* similar documents based on text
- **Reinforcement**: learn to **act** based on **feedback/reward**
  - Ex.: learn to play Go, reward: *win or lose*

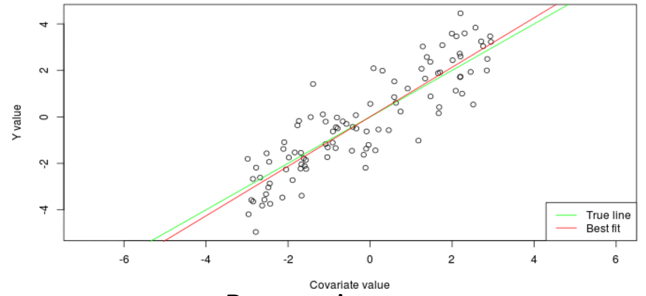
Other appls: Anomaly Detection, Sequence labeling, etc…



Classification

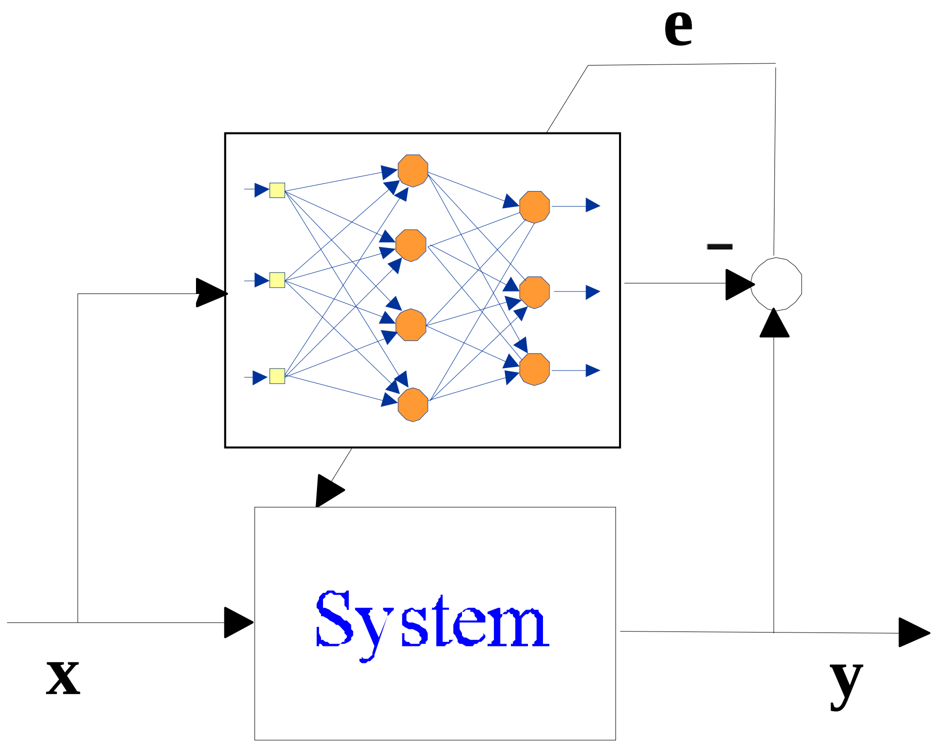


Clustering



Regression

Supervised learning

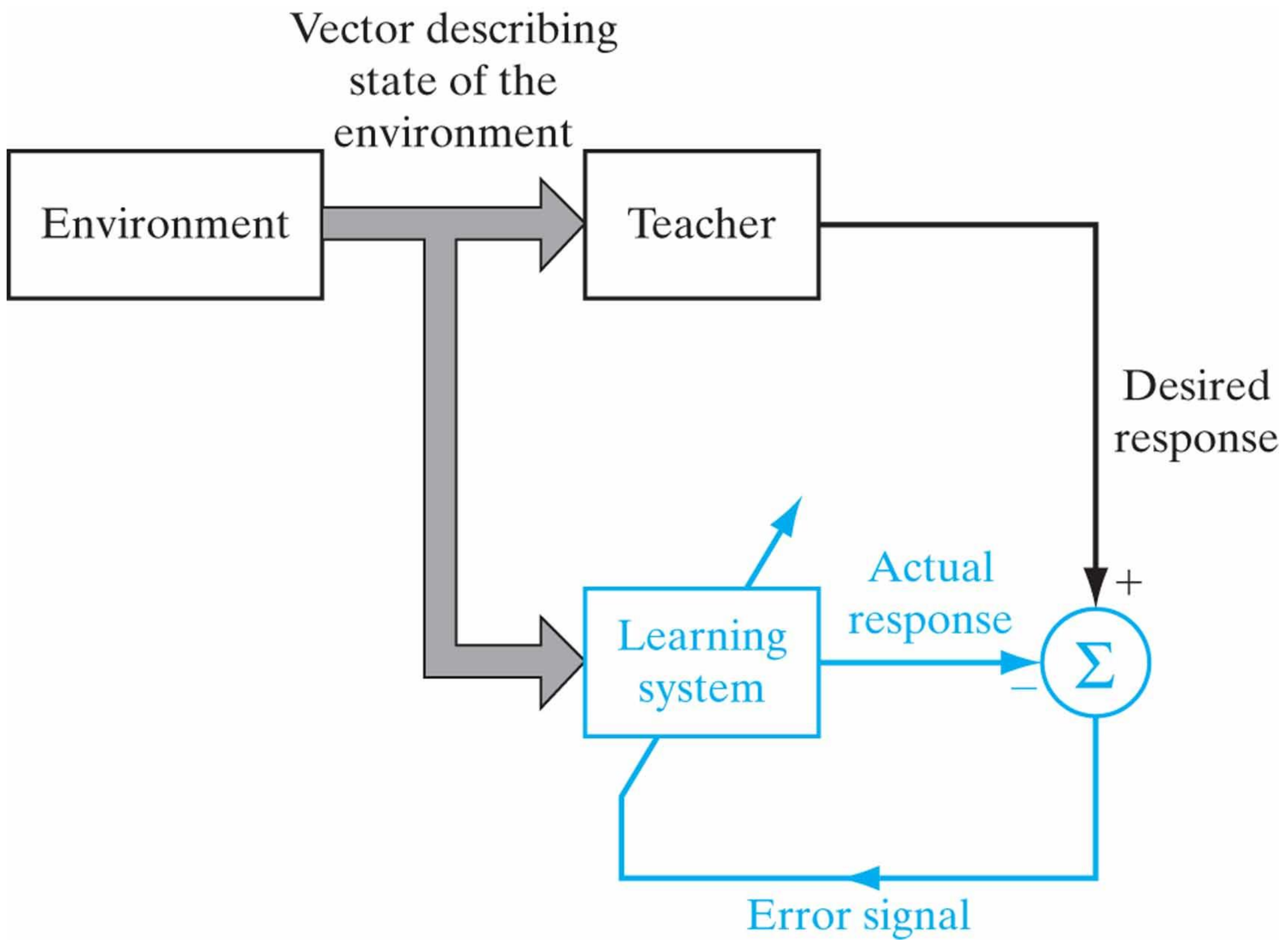


- Training data:

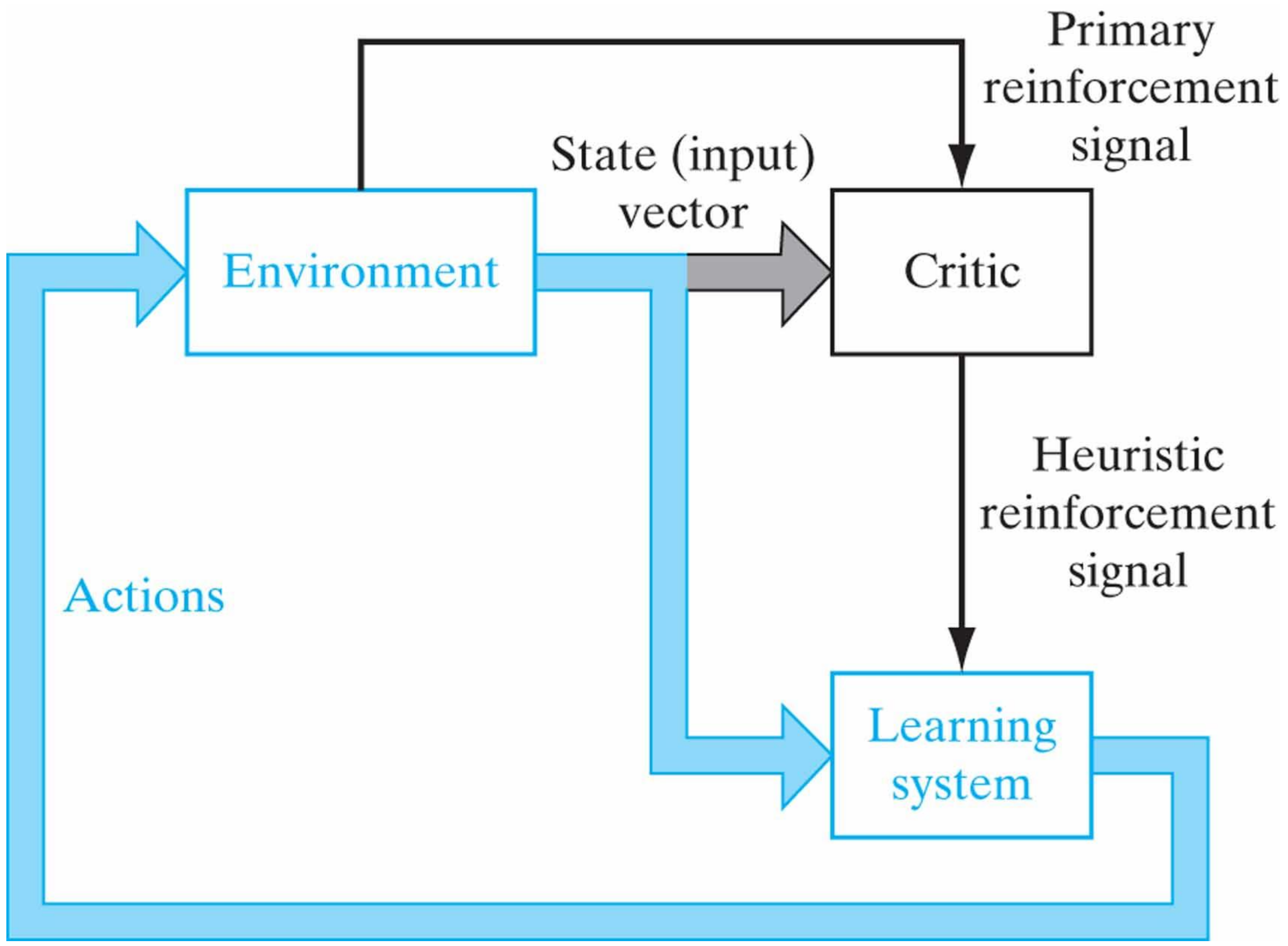
$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T & \mathbf{x}_2^T & \cdots & \mathbf{x}_N^T \end{bmatrix}^T$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T & \mathbf{y}_2^T & \cdots & \mathbf{y}_N^T \end{bmatrix}^T$$

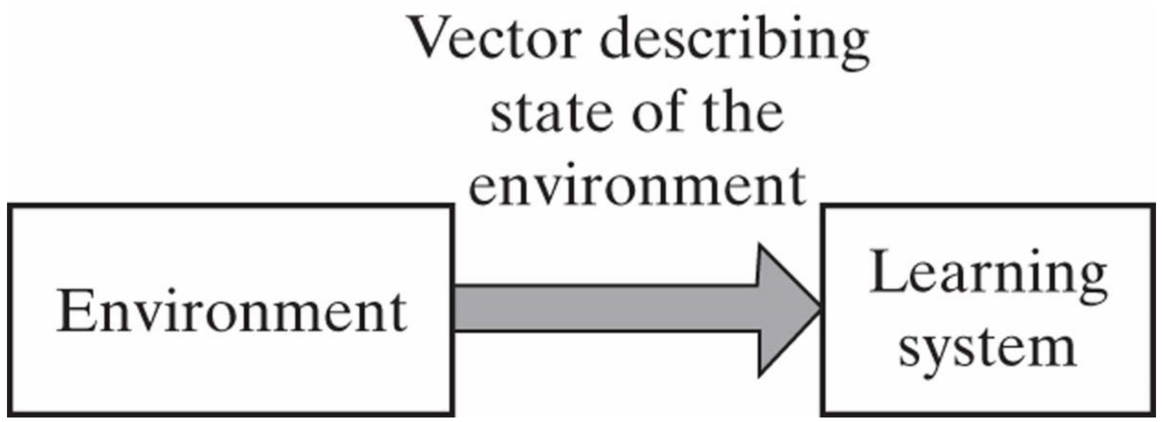
Supervised Learning



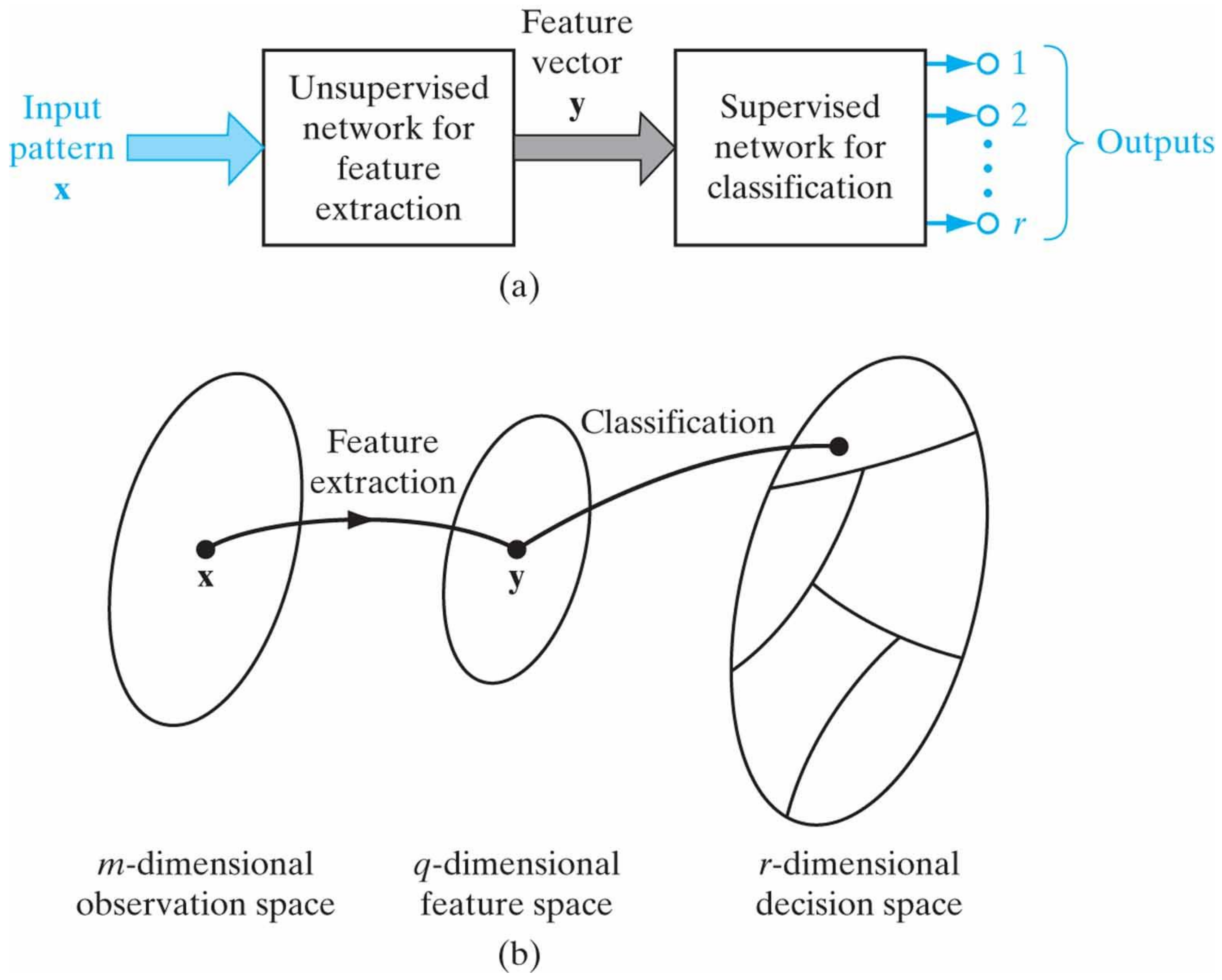
Reinforcement Learning



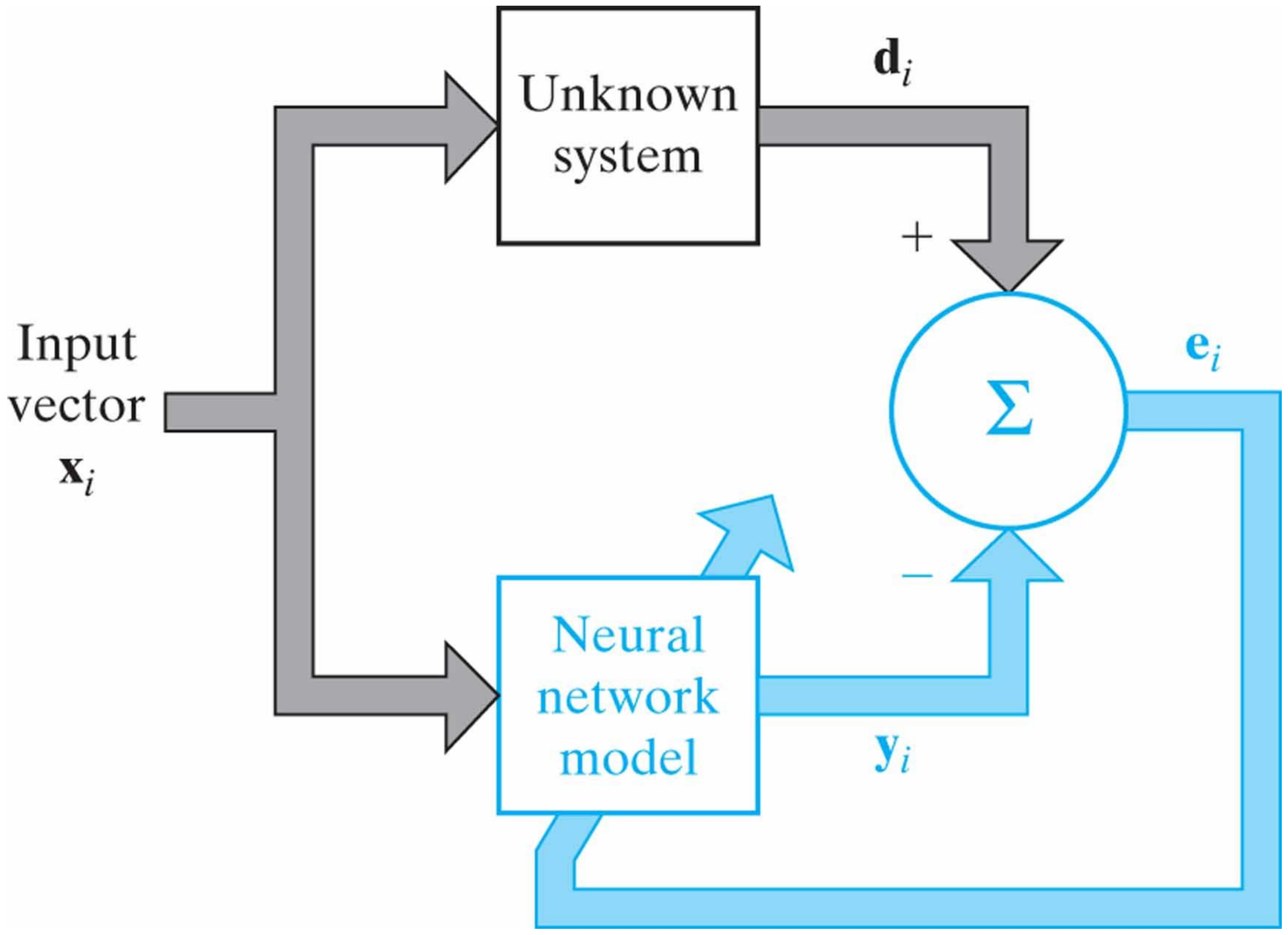
Unsupervised Learning



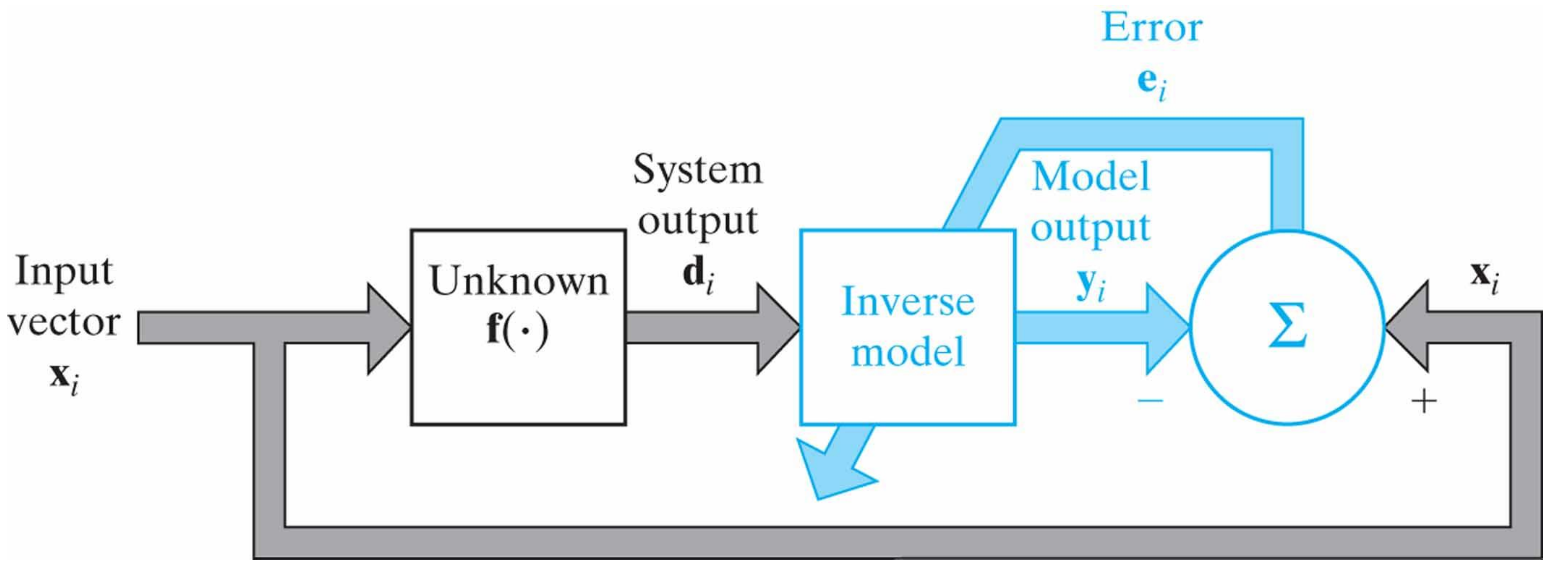
Pattern Recognition


Input pattern x
Unsupervised network for feature extraction
Feature vector y
Supervised network for classification
1
2
r
Outputs
(a)
Feature extraction
Classification
x
y
m-dimensional observation space
q-dimensional feature space
r-dimensional decision space
(b)

System Identification

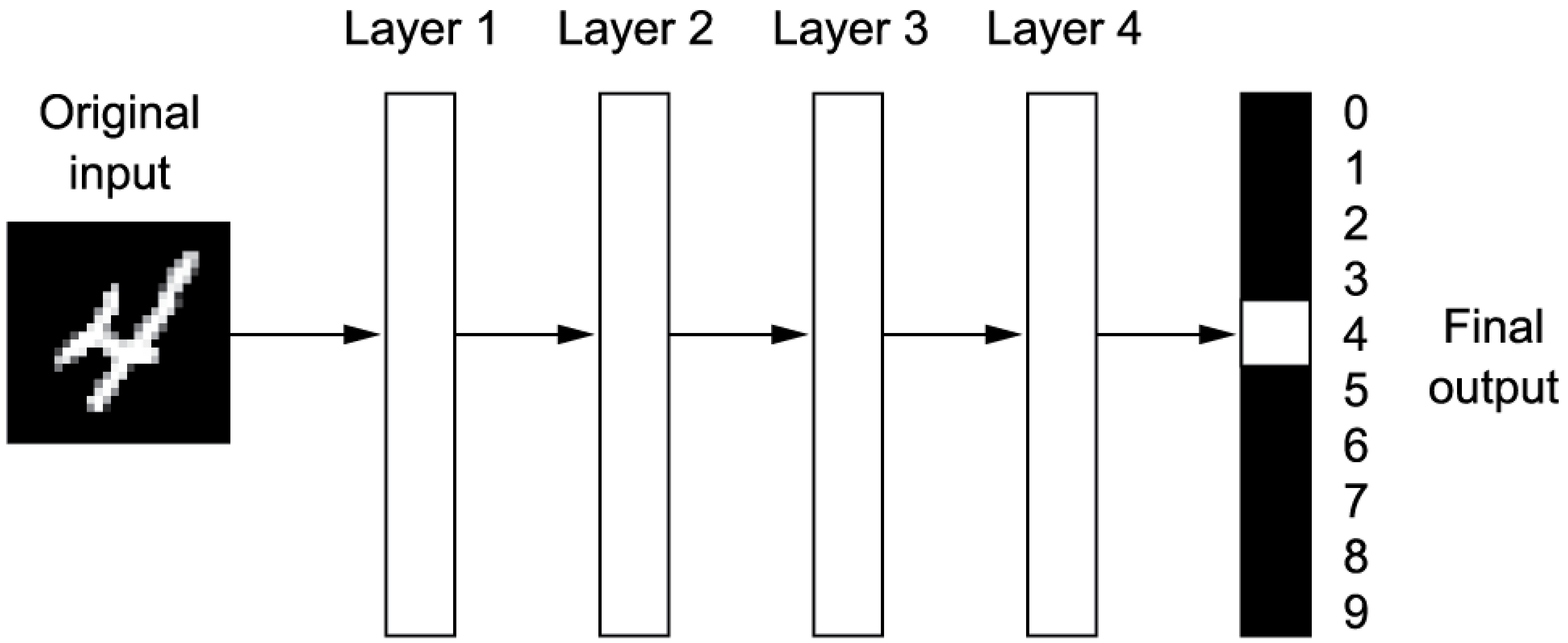


Inverse system modeling



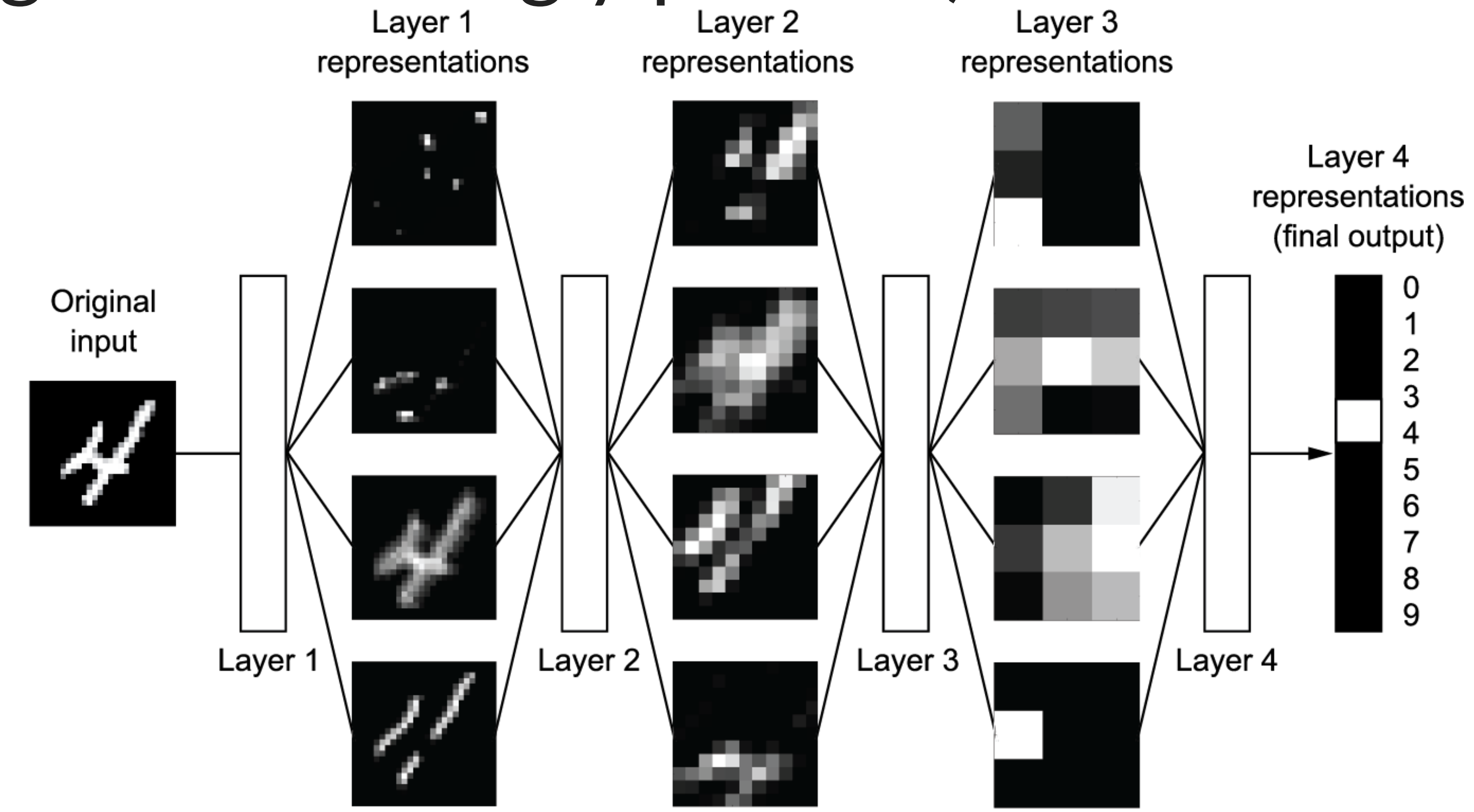
The “deep” in “deep learning”

- A deep neural network for digit classification



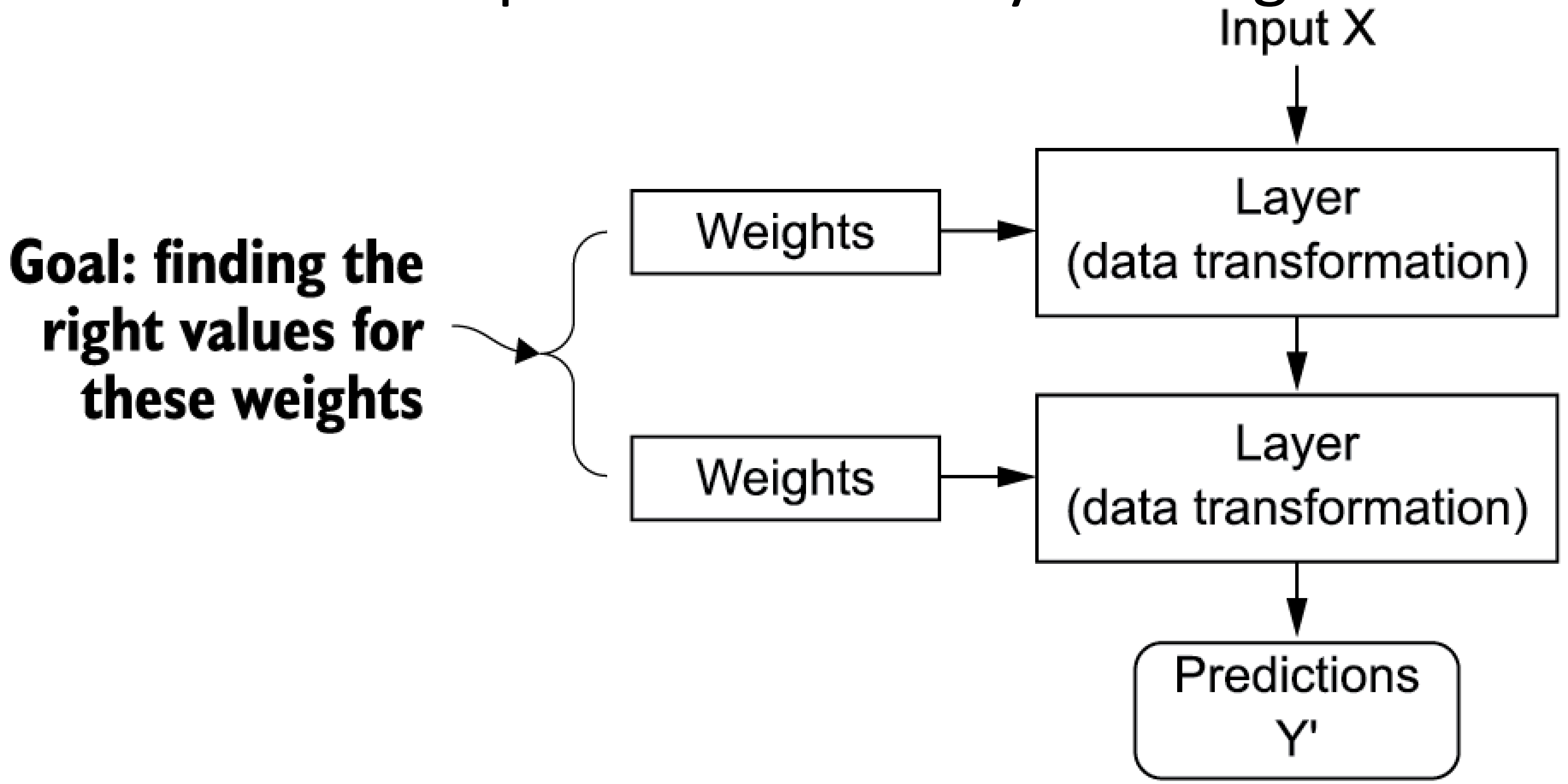
The “deep” in “deep learning”

- Deep network is a multistage **information-distillation** process (gets increasingly purified)



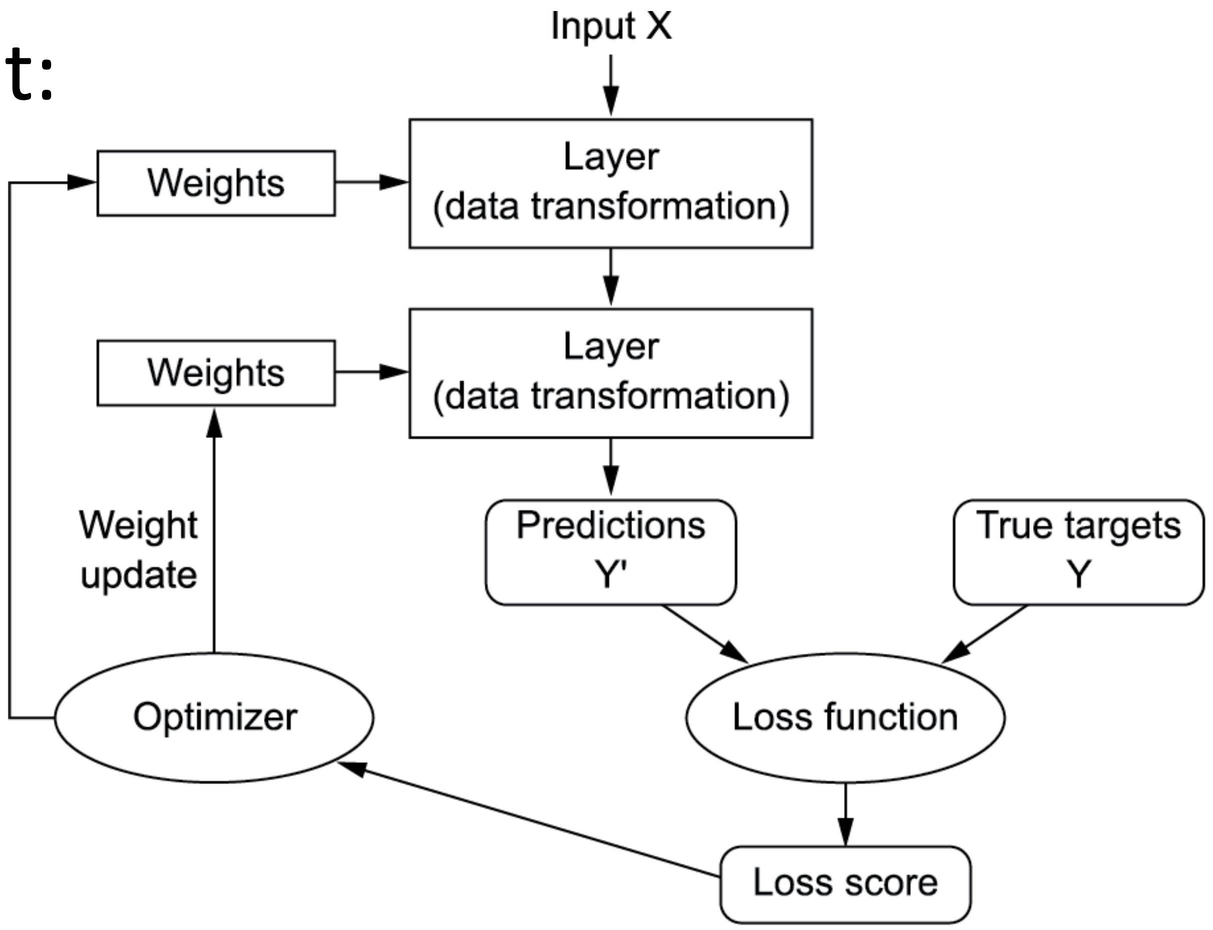
Learning...

- A neural network is parameterized by its weights:


Input X
Goal: finding the right values for these weights
Weights
Weights
Layer (data transformation)
Layer (data transformation)
Predictions Y'

Learning...

- **A loss function** is used to measure the quality of the network output:



What deep learning achieved so far

- Since early 2010s, breakthroughs, all in historically difficult areas of machine learning:
    - Near-human-level image classification
    - Near-human-level speech transcription
    - Near-human-level handwriting transcription
    - Dramatically improved machine translation
    - Dramatically improved text-to-speech conversion
    - Digital assistants such as Google Assistant and Amazon Alexa
    - …

TÉCNICO LISBOA

What deep learning achieved so far

- Since early 2010s, breakthroughs, all in historically difficult areas of machine learning:
  - Near-human-level autonomous driving
  - Improved ad targeting, as used by Google, Baidu, or Bing
  - Improved search results on the web
  - Ability to answer natural language questions
  - Superhuman Go playing
  - And going…

What makes deep learning different

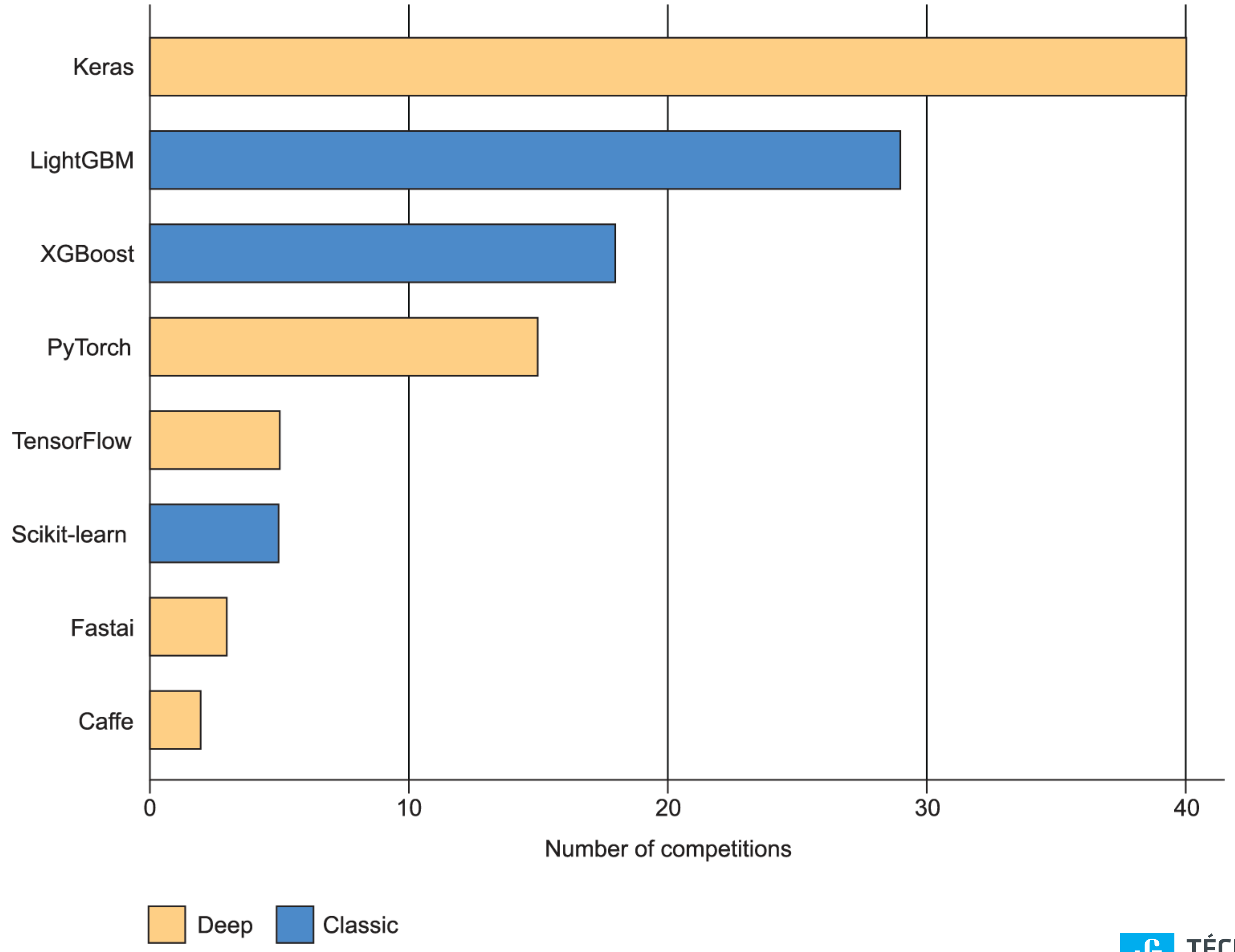
- The primary reason deep learning took off so quickly is that it offered better performance for many problems.
- Deep learning also makes problem-solving much easier, because it completely automates what used to be the most crucial step in a machine learning workflow: ***feature engineering***.

What makes deep learning different

- What is transformative about deep learning is that it allows a model to **learn all layers of representation jointly**, at the same time, rather than in succession (greedily, as it’s called).

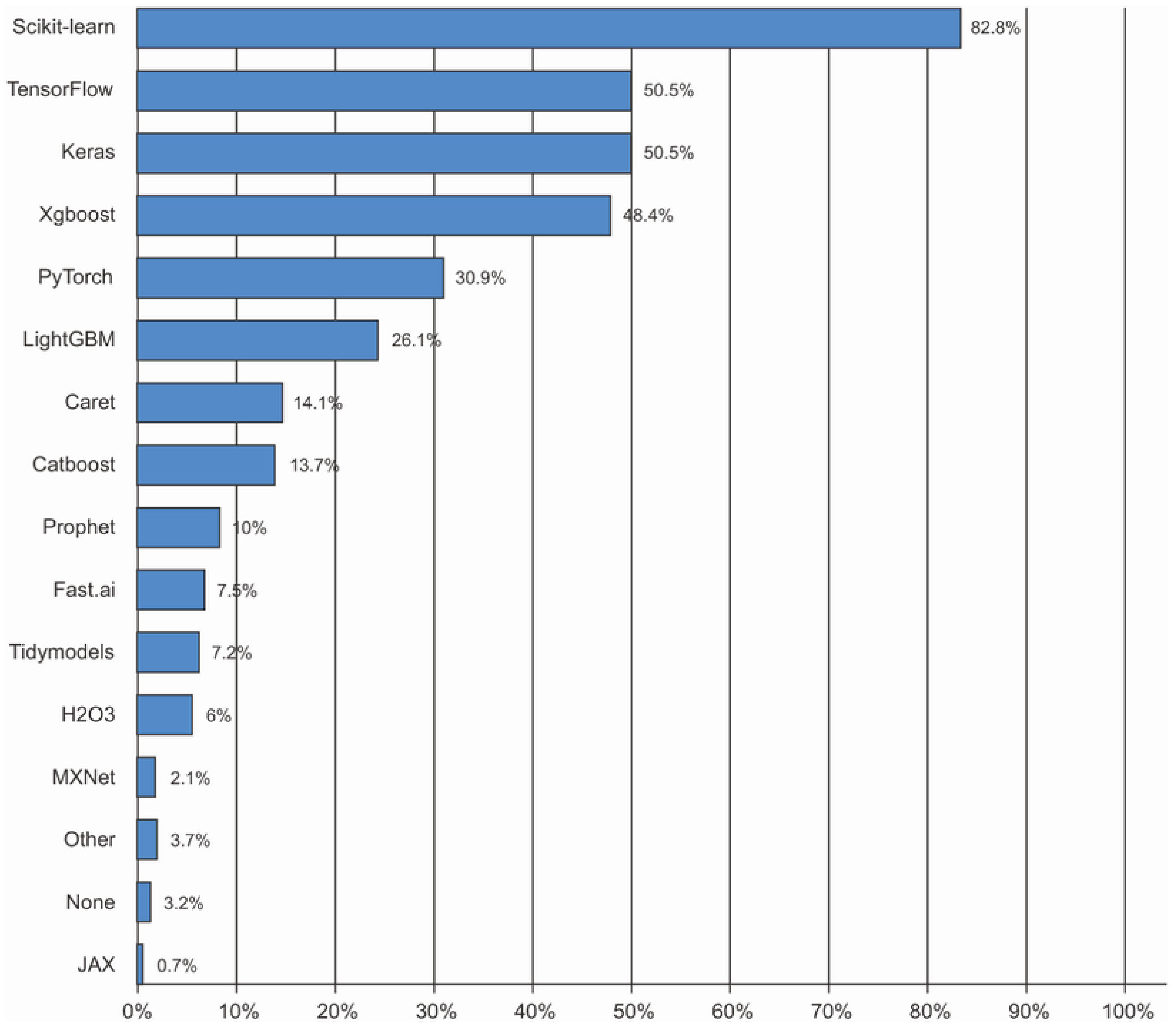
DL Tools

- Top Teams in Kaggle



DL Tools

- Tool usage across the machine learning and data science industry (Source: www.kaggle.com/kaggle-survey-2020)



Why deep learning? Why now?

- The two key ideas of deep learning for computer vision—convolutional neural networks and backpropagation—were already well understood by **1990**.
- The Long Short-Term Memory (LSTM) algorithm, which is fundamental to deep learning for timeseries, was developed in **1997** and has barely changed since.
- So why did **deep learning only take off after 2012**? What changed in these two decades?

TÉCNICO LISBOA

Why deep learning? Why now?

- In general, three technical forces are driving advances in machine learning:
  - Hardware
  - Datasets and benchmarks
  - Algorithmic advances

Hardware

- in 2007, NVIDIA launched CUDA (https://developer.nvidia.com/about-cuda), a programming interface for its line of GPUs.
- A small number of GPUs started replacing massive clusters of CPUs in various highly parallelizable applications, beginning with physics modeling.
- DNN, are many small matrix multiplications, are also highly parallelizable, and around 2011 some researchers began to write CUDA implementations of neural nets—Dan Ciresan[1] and Alex Krizhevsky[2] were among the first.

[1]"Flexible, High Performance Convolutional Neural Networks for Image Classification," Proceedings of the 22nd International Joint Conference on Artificial Intelligence (2011)

[2]"ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems 25 (2012)

Data

- If there’s one dataset that has been a catalyst for the rise of deep learning, it’s the **ImageNet dataset**, consisting of 1.4 million images that have been hand annotated with 1,000 image categories (one category per image). But what makes ImageNet special isn’t just its large size, but also the yearly competition associated with it[1]

[1]The ImageNet Large Scale Visual Recognition Challenge (ILSVRC), www.image-net.org/challenges/LSVRC

Algorithms

- Until the late 2000s, we were missing a reliable way to train very deep neural networks
- The key issue was that of ***gradient propagation*** through deep stacks of layers. The feedback signal used to train neural networks **would fade away** as the **number of layers increased**.

Algorithms

- Vanishing Gradient solutions:
  - Better **activation functions** for neural layers
  - Better **weight-initialization schemes**, starting with layer-wise pretraining, which was then quickly abandoned
  - Better **optimization schemes**, such as RMSProp and Adam
  - Batch normalization,
  - Residual connections, and depthwise separable convolutions.

Extremely large models

- This extreme scalability is one of the defining characteristics of modern deep learning.
- Large-scale model architectures, which feature tens of layers and tens of millions of parameters, have brought about critical advances both in **computer vision** (ResNet, Inception, or Xception) and **natural language processing** (large Transformer-based architectures such as BERT, GPT-3, or XLNet).

DL Properties

- Deep learning has several properties that justify its status as an AI revolution, and it’s here to stay:
  - **Simplicity**—Deep learning removes the need for feature engineering, replacing complex, brittle, engineering-heavy pipelines with simple, end-to-end trainable models that are typically built using only five or six different tensor operations.

DL Properties

- Deep learning has several properties that justify its status as an AI revolution, and it’s here to stay:
  - **Scalability**—Deep learning is highly amenable to parallelization on GPUs or TPUs, so it can take full advantage of Moore’s law. In addition, deep learning models are trained by iterating over small batches of data, allowing them to be trained on datasets of arbitrary size. (The only bottleneck is the amount of parallel computational power available, which, thanks to Moore’s law, is a fast-moving barrier.)

DL Properties

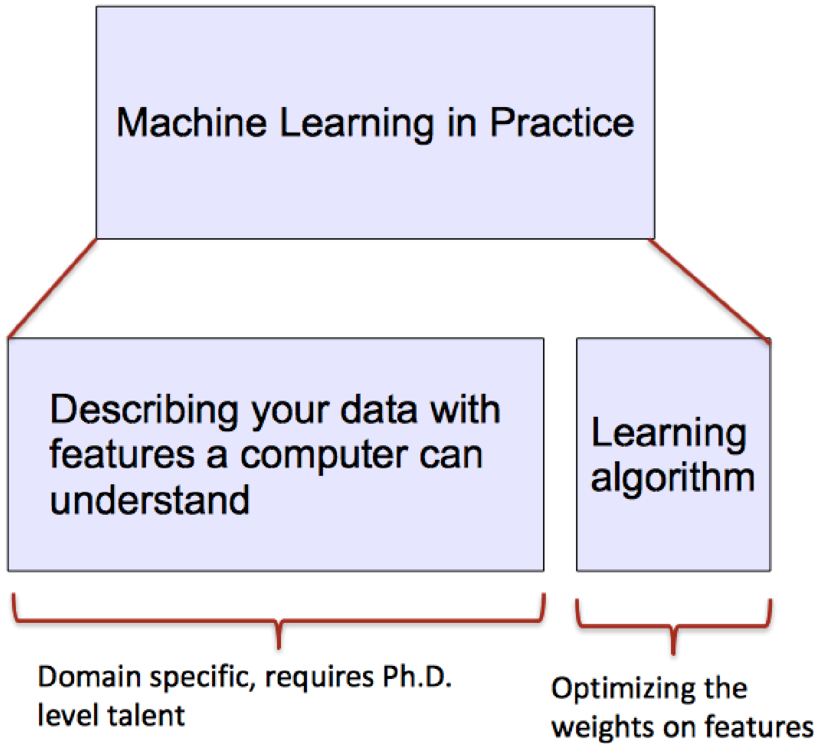
- Deep learning has several properties that justify its status as an AI revolution, and it’s here to stay:
  - **Versatility and reusability**—Unlike many prior machine learning approaches, deep learning models can be trained on additional data **without restarting from scratch**, making them viable for continuous online learning—an important property for very large production models.

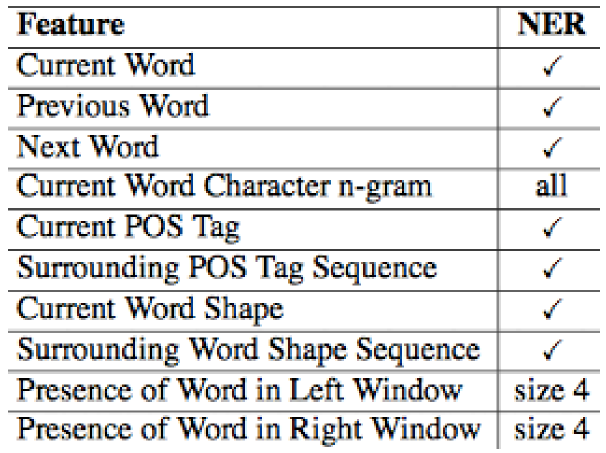
DL Properties

- Deep learning has several properties that justify its status as an AI revolution, and it’s here to stay:
  - **Versatility and reusability**—Furthermore, trained deep learning models are repurposable and thus reusable: for instance, it’s possible to take a deep learning model trained for image classification and drop it into a video-processing pipeline. This allows us to reinvest previous work into increasingly complex and powerful models. ***This also makes deep learning applicable to fairly small datasets.***

ML vs. Deep Learning

- Machine learning methods work well because of **human-designed representations** and **input features**
- ML becomes just **optimizing weights** to obtain the best final prediction

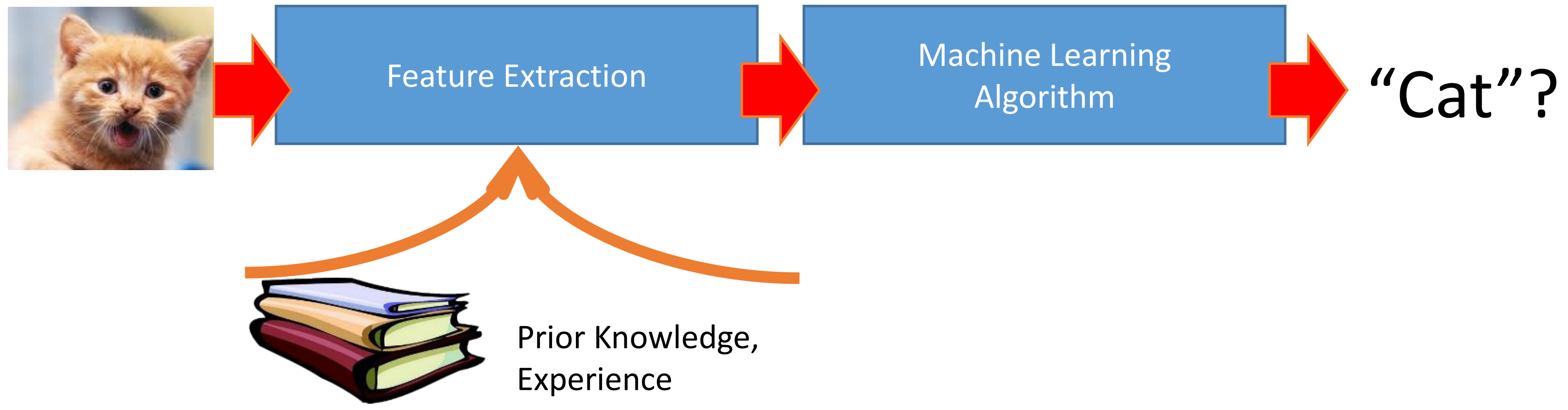

Machine Learning in Practice
Describing your data with features a computer can understand
Learning algorithm
Domain specific, requires Ph.D. level talent
Optimizing the weights on features



Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

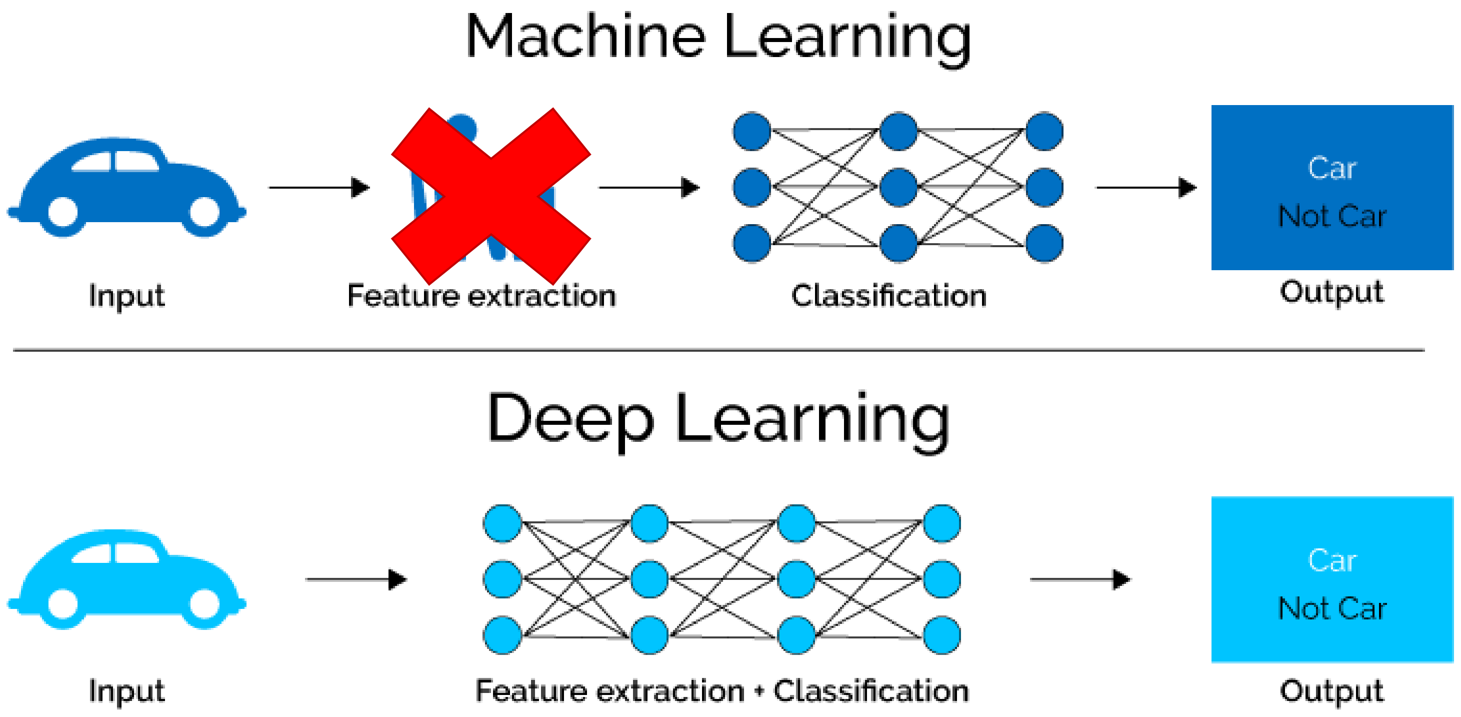
ML vs. Deep Learning

- Machine learning often uses common pipeline with hand-designed feature extraction.
  - Final ML algorithm learns to make decisions starting from the higher-level representation.
  - Sometimes layers of increasingly high-level abstractions.
    - Constructed using prior knowledge about problem domain.



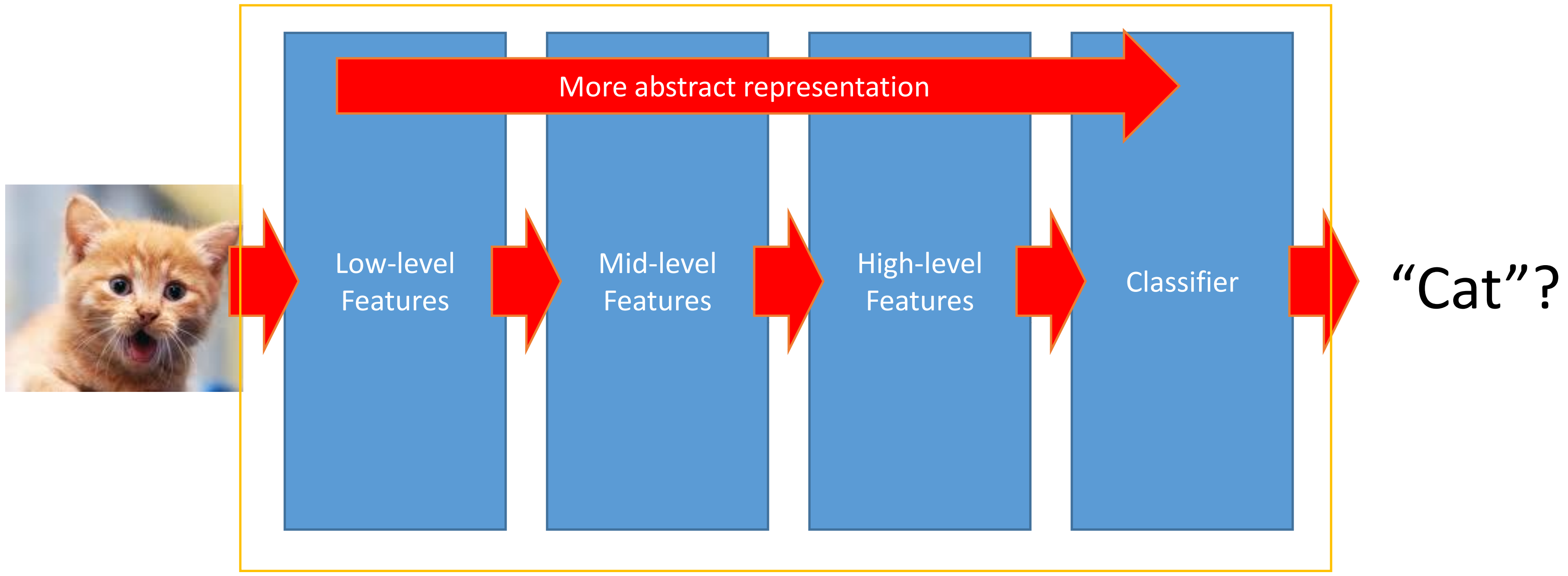
ML vs. Deep Learning

- Learn (multiple levels of) **representations** of data by using a **hierarchy of multiple layers**
- Use **tons of information about the system**, it begins to understand it and respond in useful ways.



Deep Learning

- Deep Learning
  - Train *multiple layers* of features/abstractions from data.
  - Try to discover *representation* that makes decisions easy.

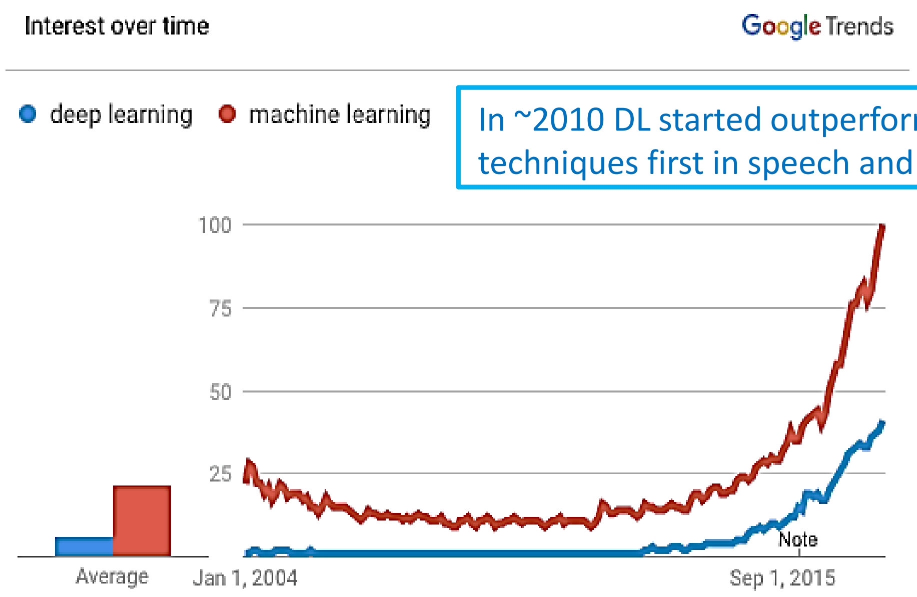


Deep Learning: train layers of features so that classifier works well.

Deep Learning

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, **universal***, learnable framework for representing world.
- Can learn both **unsupervised**

and **supervised**

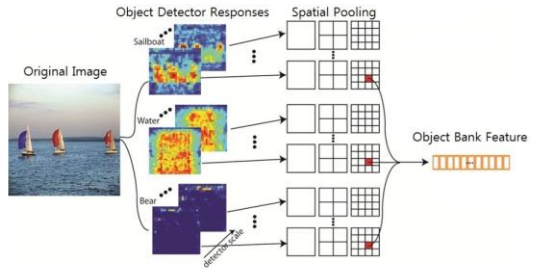


In ~2010 DL started outperforming other ML techniques first in speech and vision, then NLP

Features

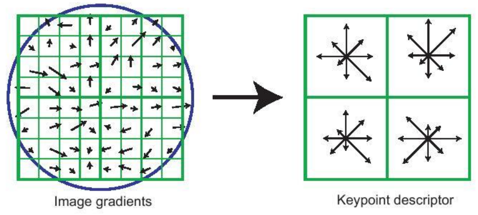
- Huge investment devoted to building application-specific feature representations.
  - Find higher-level patterns so that final decision is easy to learn with ML algorithm.

Object Bank [Li et al., 2010]

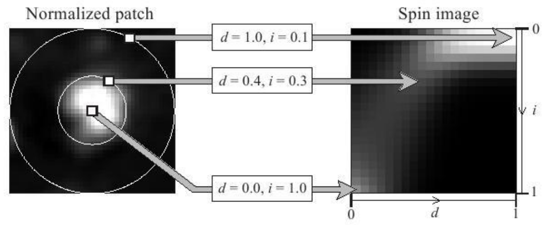


Super-pixels
[Gould et al., 2008; Ren & Malik, 2003]

SIFT [Lowe, 1999]

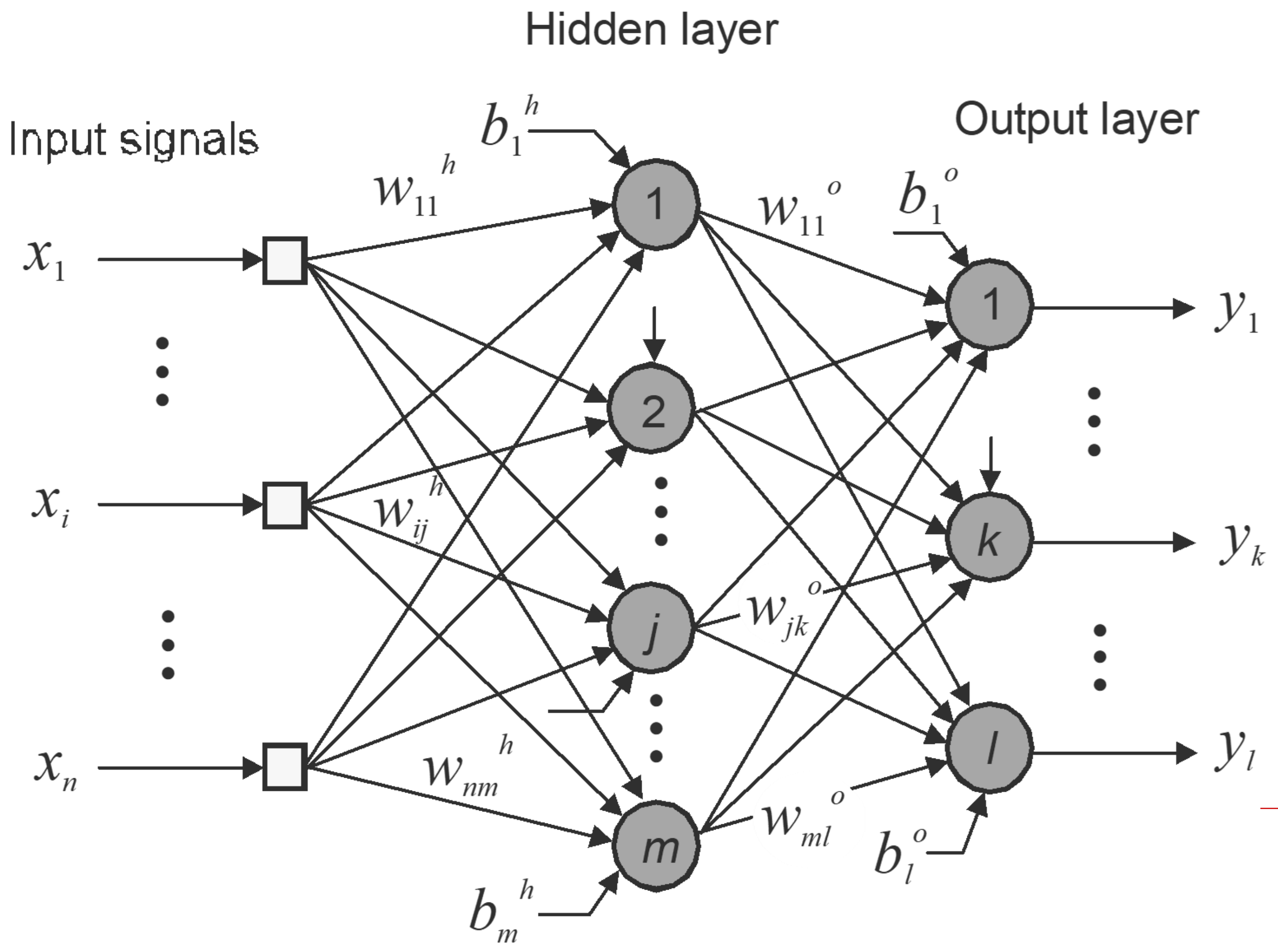


Spin Images [Johnson & Hebert, 1999]



SUPERVISED DEEP LEARNING

Neural Network Recall



$$h_j = \sigma\left(\sum_{i=1}^{n} w_{ij}^h x_i + b_j^h\right) = \sigma\left(\sum_{i=0}^{n} w_{ij}^h x_i\right)$$
$$= \tanh\left(\sum_{i=0}^{n} w_{ij}^h x_i\right)$$

Weights

$$y_k = \sigma\left(\sum_{j=1}^{m} w_{jk}^o h_j + b_j^o\right) = \sigma\left(\sum_{j=0}^{m} w_{jk}^o h_k\right)$$
$$= \sum_{j=0}^{m} w_{jk}^o h_j$$

Activation functions

How do we train?

4 + 3 = 7 neurons (not counting inputs)
[3 x 4] + [4 x 3] = 24 weights
4 + 3 = 7 biases

31 learnable **parameters**

TÉCNICO LISBOA

Error backpropagation algorithm

- Initialize all weights to small random numbers

Repeat:

1. Input training example and compute network outputs.
2. Adjust output weights using gradients:

$$w^o_{jk}(p+1) = w^o_{jk}(p) + \alpha\, h_j e_k$$

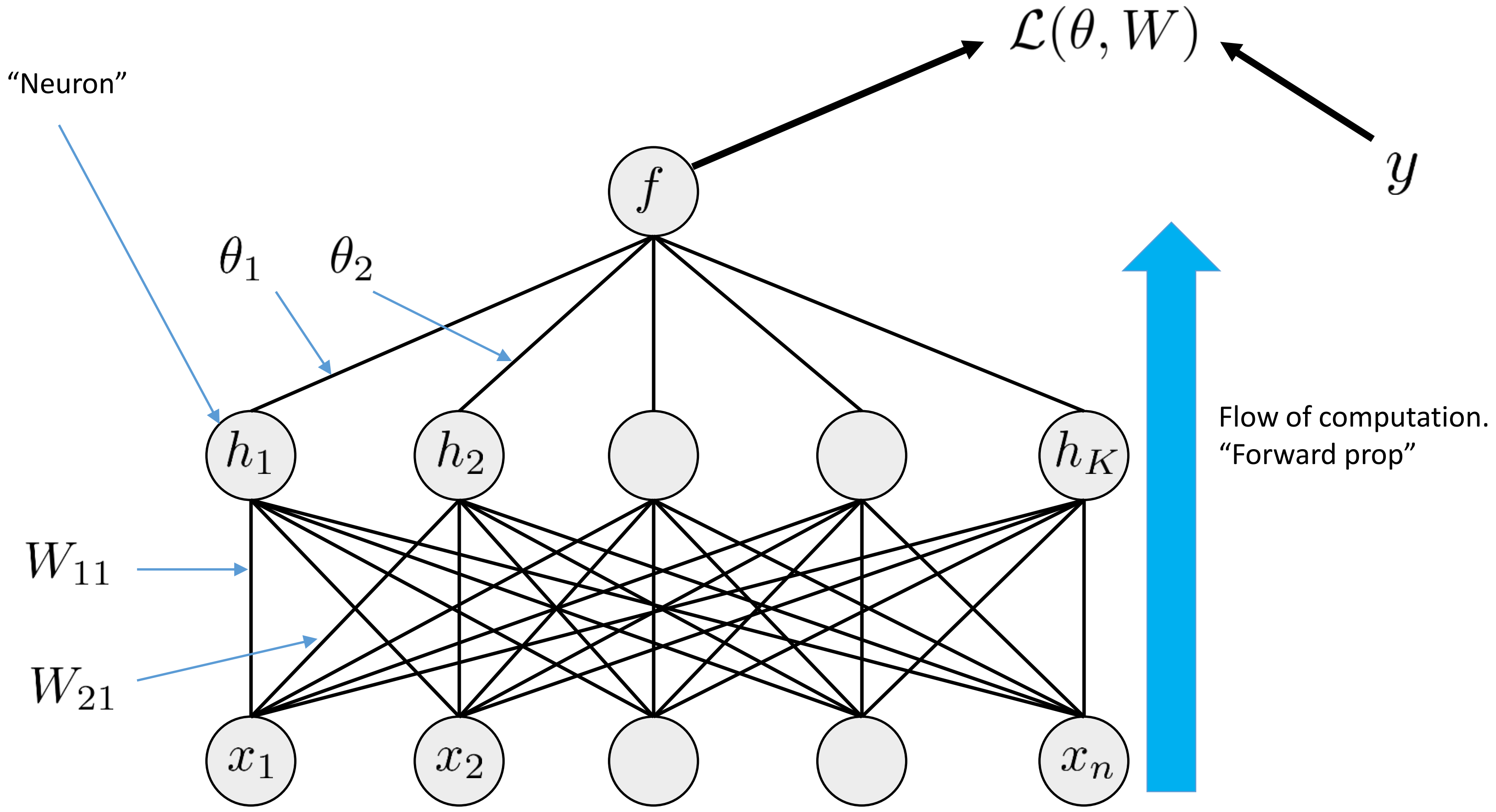
3. Adjust hidden-layer weights:

$$w^h_{ij}(p+1) = w^h_{ij}(p) + \alpha x_i\, \sigma'_j(h_j) \sum_{k=1}^{l} (-e_k w^o_{jk})$$

Until satisfied or fixed number of epochs p

Neural network

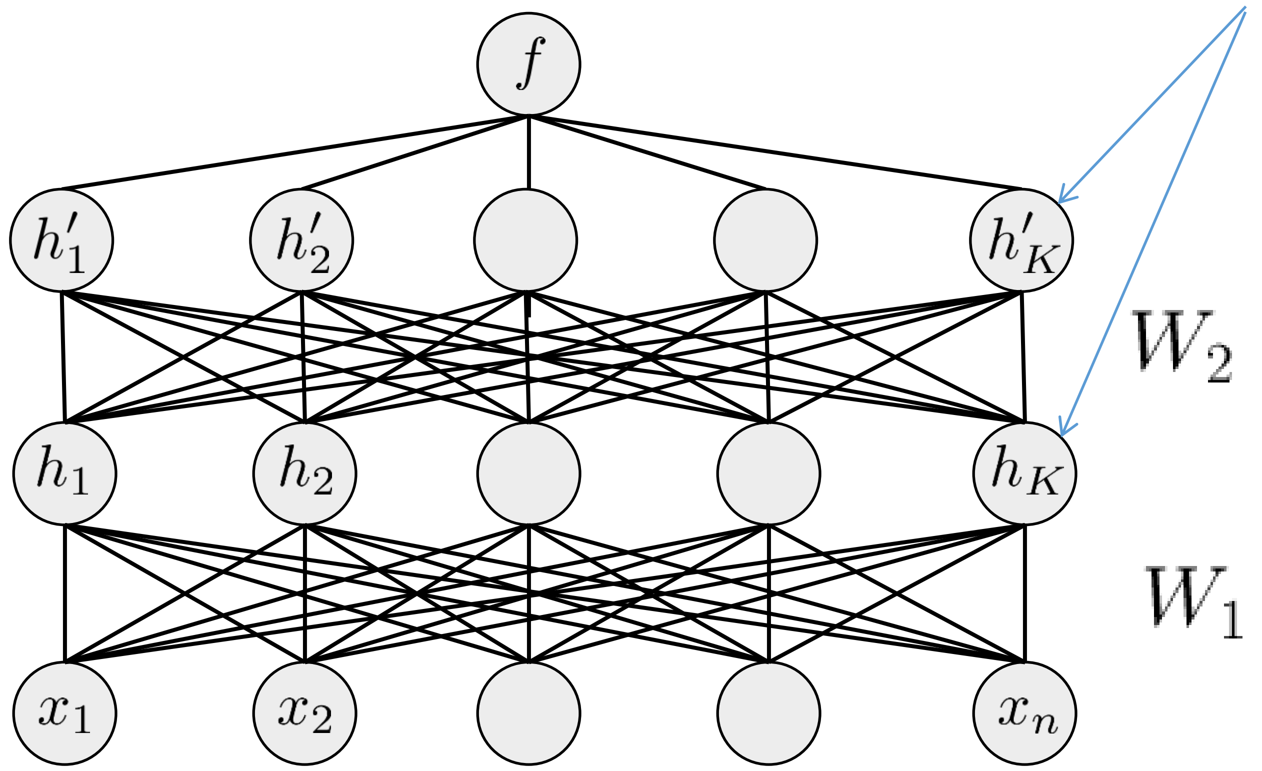
- This model is a sigmoid “neural network”:



Neural network

- Can stack up several layers:

Must learn multiple stages of internal "representation".



$$x \rightarrow \boxed{\sigma(W_1 x)} \rightarrow h \rightarrow \boxed{\sigma(W_2 h)} \rightarrow h' \rightarrow \boxed{\sigma(\theta^\top h')} \rightarrow f$$

Backpropagation

- Minimize: $$\mathcal{L}(\theta, W) = -\sum_{i}^{m} 1\{y^{(i)} = 1\} \log(f(x^{(i)}; \theta, W)) + 1\{y^{(i)} = 0\} \log(1 - f(x^{(i)}; \theta, W))$$

- To minimize $\mathcal{L}(\theta, W)$ we need gradients: $\nabla_\theta \mathcal{L}(\theta, W)$ and $\nabla_W \mathcal{L}(\theta, W)$
  - Then use gradient descent algorithm as before.

- Formula for $\nabla_\theta \mathcal{L}(\theta, W)$ can be found by hand (same as before); but what about W?

TÉCNICO LISBOA

Training Procedure

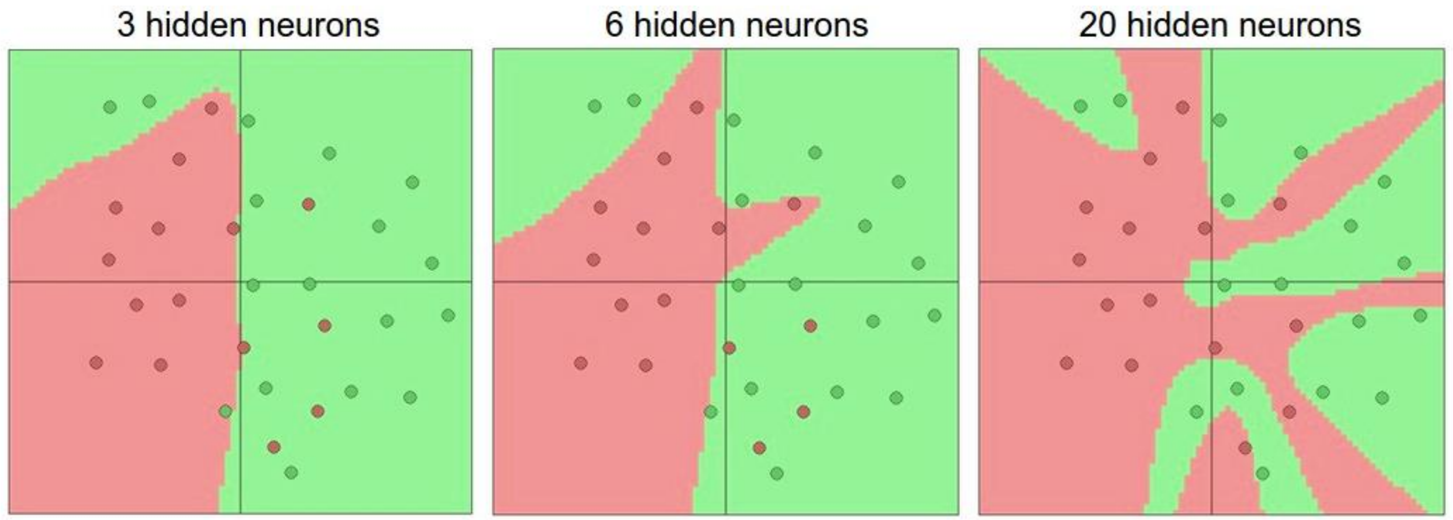
- Historically, this has not worked so easily.
  - Non-convex: Local minima; convergence criteria.
  - Optimization becomes difficult with many stages.
    - “Vanishing gradient problem”
  - Hard to diagnose and debug malfunctions.

- Many things turn out to matter:
  - Choice of **nonlinearities**.
  - Initialization of parameters.
  - Optimizer parameters: step size, schedule.

Nonlinearities

- Non-linearities needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function

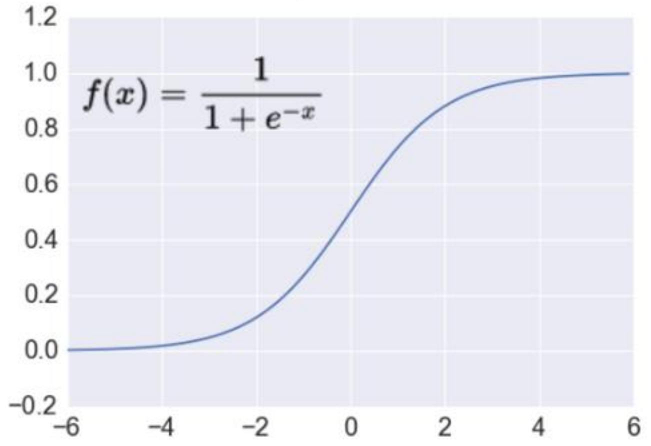
$$W_1 W_2 x = Wx$$



http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

- More layers and neurons can approximate more complex functions

Activation: Sigmoid

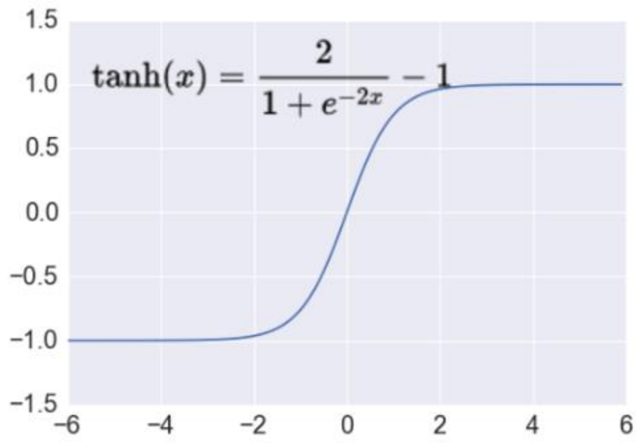


Takes a real-valued number and “squashes” it into range between 0 and 1.

$$R^n \rightarrow [0,1]$$

+ Nice interpretation as the **firing rate** of a neuron
  - 0 = not firing at all and 1 = fully firing
- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn
  - when the neuron’s activation are 0 or 1 (saturate)
    - gradient at these regions almost zero
    - almost no signal will flow to its weights
    - if initial weights are too large then most neurons would saturate

Activation: Tanh



Takes a real-valued number and "squashes" it into range between -1 and 1.

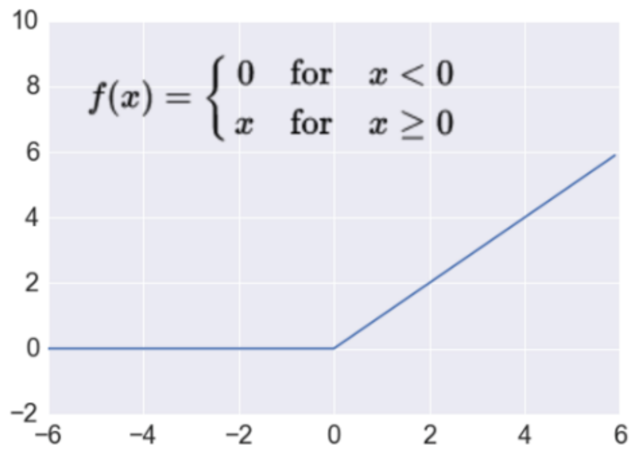
$$R^n \rightarrow [-1,1]$$

- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation: ReLU

Takes a real-valued number and thresholds it at zero $f(x) = \max(0, x)$

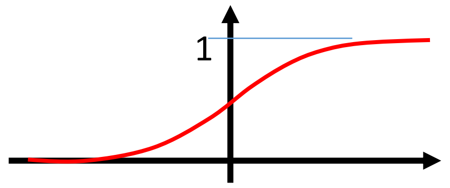
$$R^n \to R^n_+$$



- Most Deep Networks use ReLU nowadays
  - ➢ Trains much **faster** [Nair & Hinton, 2010]
    - ➢ accelerates the convergence of SGD
    - ➢ due to linear, non-saturating form
  - ➢ Less expensive operations
    - ➢ compared to sigmoid/tanh (exponentials etc.)
    - ➢ implemented by simply thresholding a matrix at zero
  - ➢ More **expressive**
  - ➢ Prevents the **gradient vanishing problem**

Initialization

- Usually small random values.
  - Try to choose so that typical input to a neuron avoids saturating / non-differentiable areas.



  - Occasionally inspect units for saturation / blowup.
  - Larger values may give faster convergence, but worse models!
- Initialization schemes for particular units:
  - tanh units: Unif[-r, r]; sigmoid: Unif[-4r, 4r].

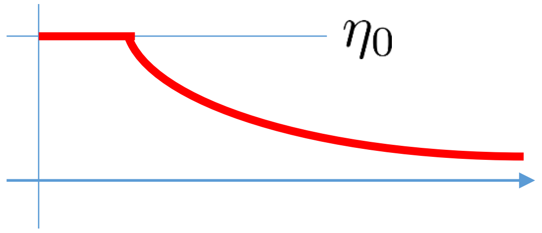
$$r = \sqrt{6/(\text{fan-in} + \text{fan-out})}$$

See [Glorot et al., AISTATS 2010]

Optimization: Step sizes

- Choose SGD step size carefully.
  - Up to factor ~2 can make a difference.
- Strategies:
  - Brute-force: try many; pick one with best result.
  - Choose so that typical "update" to a weight is roughly 1/1000 times weight magnitude. [Look at histograms.]
    - Smaller if fan-in to neurons is large.
  - Racing: pick size with best error on validation data after T steps.
    - Not always accurate if T is too small.
- Step size schedule:
  - Simple 1/t schedule:

$$\eta_t = \frac{\eta_0 \tau}{\max\{\tau, t\}}$$



  - Or: fixed step size. But if little progress is made on objective after T steps, cut step size in half.

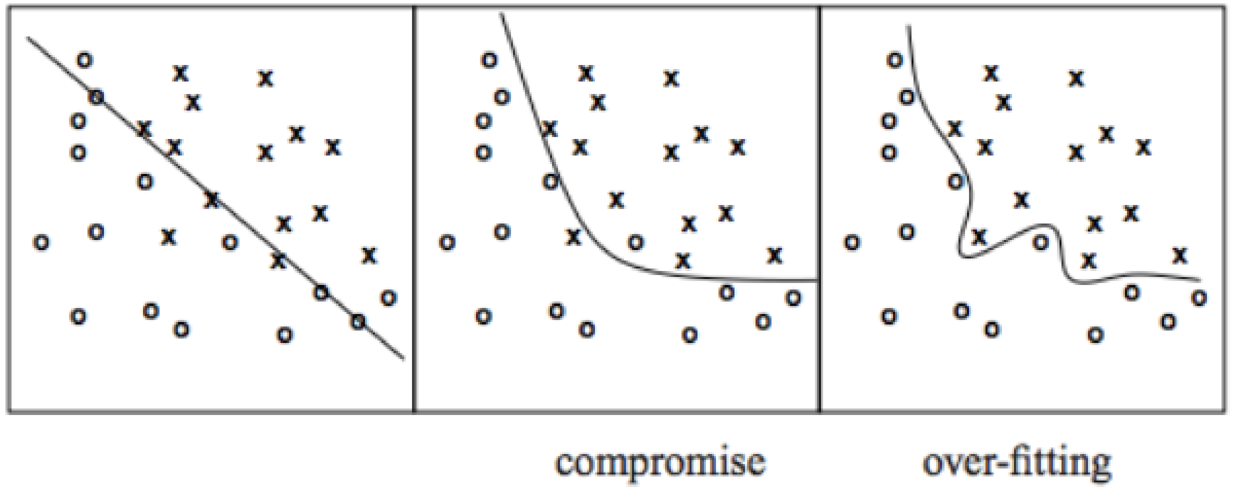
Bengio, 2012: "Practical Recommendations for Gradient-Based Training of Deep Architectures"
Hinton, 2010: "A Practical Guide to Training Restricted Boltzmann Machines"

Other factors

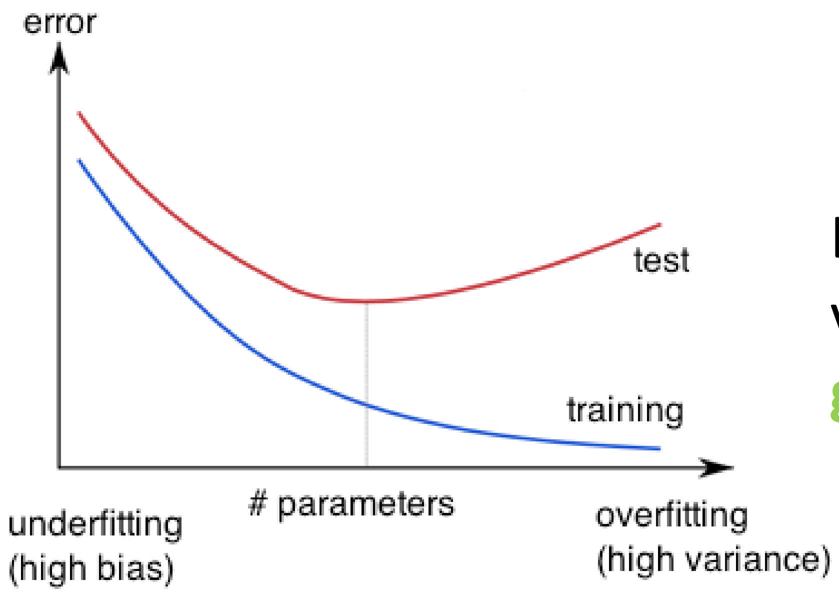
- “Weight decay” penalty can help.
  - Add small penalty for squared weight magnitude.

- For modest datasets, LBFGS or second-order methods are easier than SGD.
  - See, e.g.: Martens & Sutskever, ICML 2011.
  - Can crudely extend to mini-batch case if batches are large. [Le et al., ICML 2011]

Overfitting

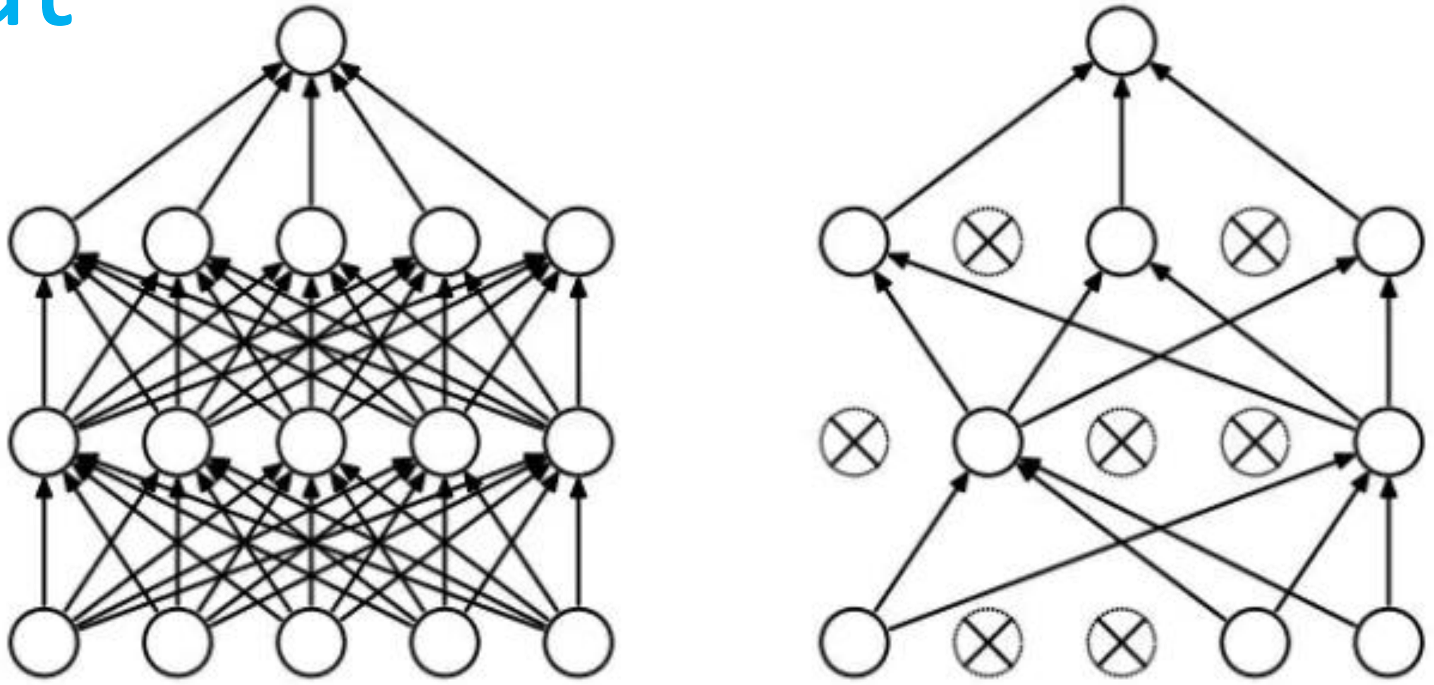


http://wiki.bethanycrane.com/overfitting-of-data



Learned hypothesis may **fit** the training data very well, even outliers (**noise**) but fail to **generalize** to new examples (test data)

Dropout



Dropout

- Randomly drop units (along with their connections) during training
- Each unit retained with fixed probability *p*, independent of other units
- **Hyper-parameter** *p* to be chosen (tuned)

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research (2014)

Regularization

L2 = weight decay

- Regularization term that penalizes big weights, added to the objective

$$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$

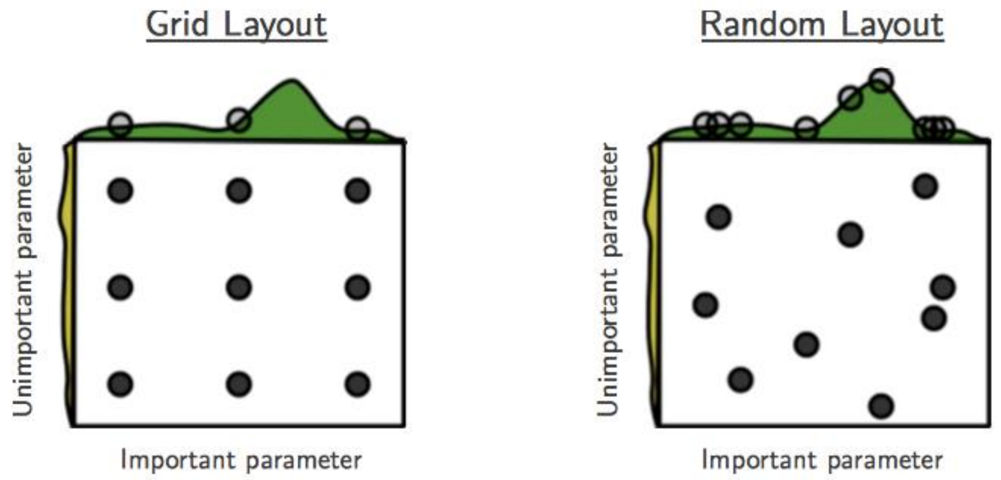
- Weight decay value determines how dominant regularization is during gradient computation
- Big weight decay coefficient → big penalty for big weights

Regularization

Early-stopping

- Use validation error to decide when to stop training
- Stop when monitored quantity has not improved after *n* subsequent epochs
- *n* is called patience

Tuning hyper-parameters



$g(x) \approx g(x) + h(y)$

g(x) shown in **green**
h(y) is shown in **yellow**

Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research, Feb (2012)

Grid and **random** search of 9 trials for optimizing function

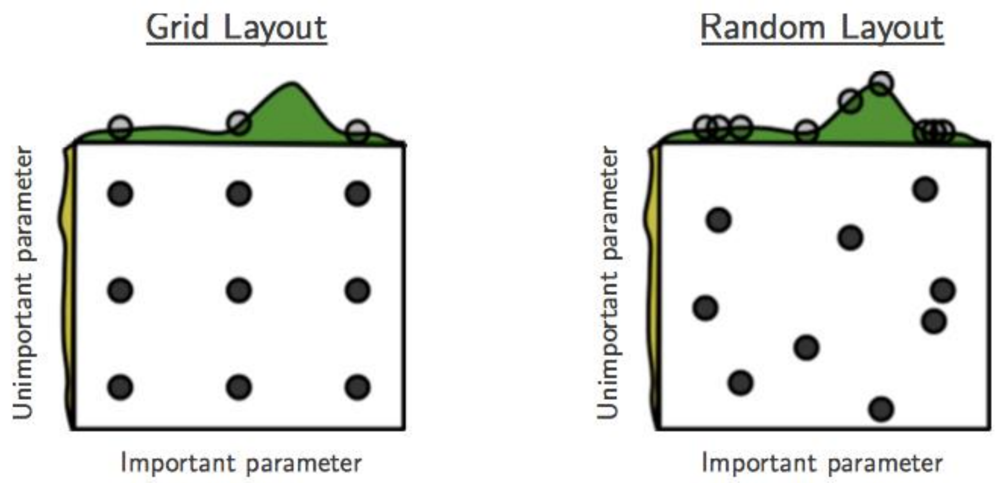
$$g(x) \approx g(x) + h(y)$$

With **grid search**, nine trials only test g(x) in three distinct places.

With **random search**, all nine trials explore distinct values of g(x)

- Both try configurations randomly and **blindly**
- Next trial **is independent** to all the trials done before

Tuning hyper-parameters



$g(x) \approx g(x) + h(y)$

g(x) shown in green
h(y) is shown in yellow

Bergstra, James, and Yoshua Bengio. "[Random search for hyper-parameter optimization.]" Journal of Machine Learning Research, Feb (2012)

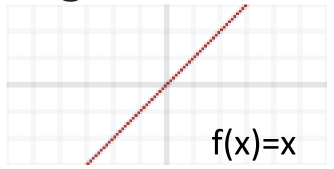
Metaheuristics/Bayesian optimization for hyper-parameter tuning:

- Make smarter choice for the next trial, minimize the number of trials
- Collect the performance at several configurations
- Make inference and decide what configuration to try next

Loss functions and output

	Classification	Regression
Training examples	R^n x {class_1, ..., class_n} (one-hot encoding)	R^n x R^m
Output Layer	Soft-max [map R^n to a probability distribution] $P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$	Linear (Identity) or Sigmoid
Cost (loss) function	Cross-entropy	Mean Squared Error

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)} \log \hat{y}_k^{(i)} + \left(1 - y_k^{(i)}\right) \log\left(1 - \hat{y}_k^{(i)}\right)\right]$$



$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

Mean Absolute Error

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)} - \hat{y}^{(i)}\right|$$

Feature representation

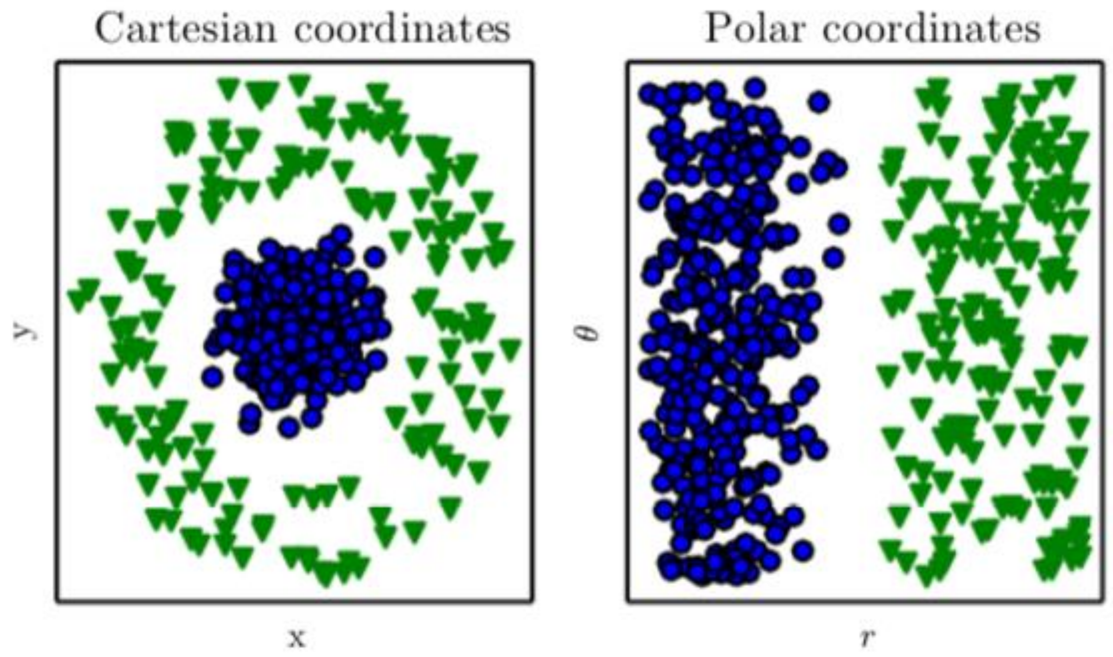


Figure 1.1: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line. Figure produced in collaboration with David Warde-Farley.

Encoder Decoder model

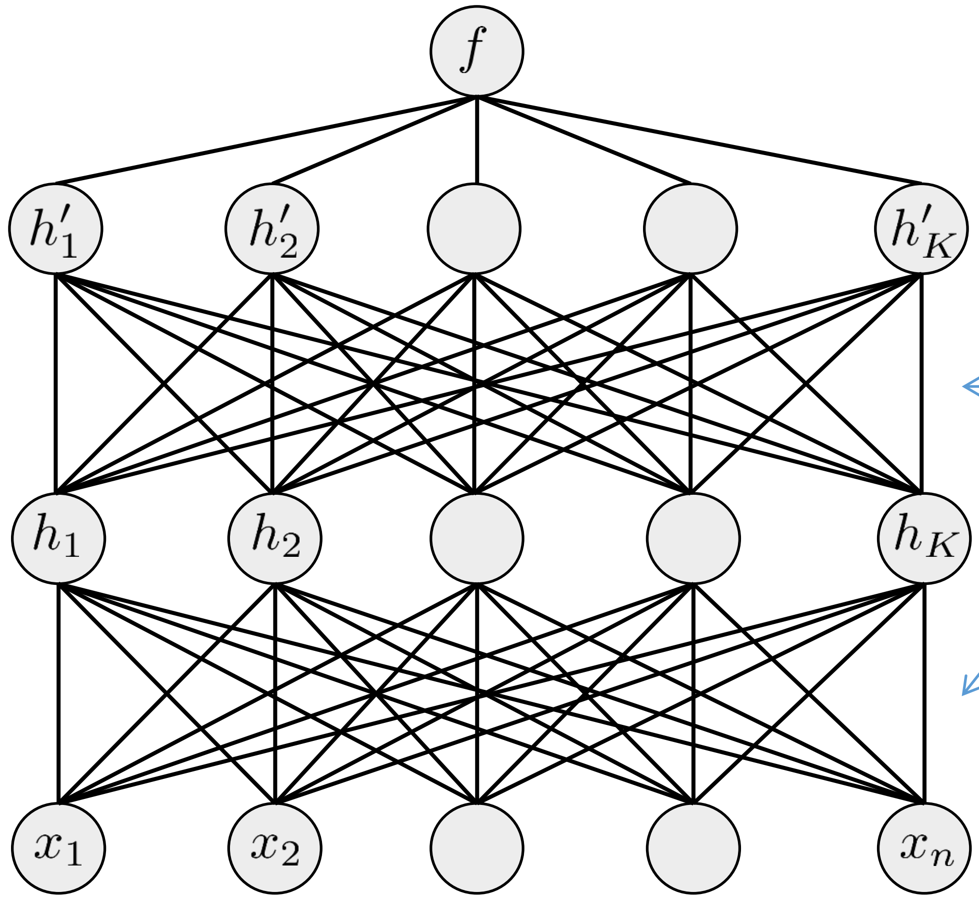
- **Encoder network**:
  - Map raw inputs to feature representations

- **Decoder network:**
  - Take this feature representation as input
  - process it to make its decision
  - produce an output

UNSUPERVISED DEEP LEARNING

Representation Learning

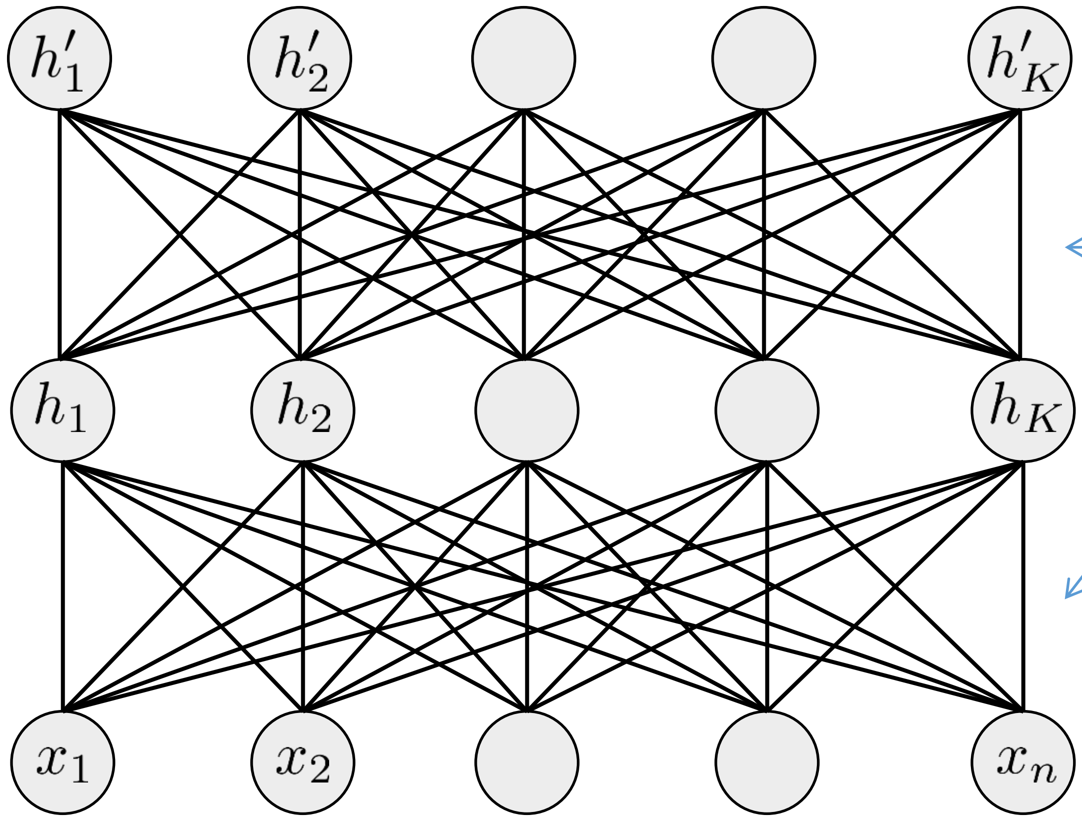
- In supervised learning, train “features” to accomplish top-level objective.



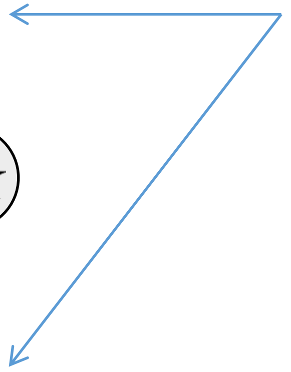
But what if we have too few labels to train all these parameters?

Representation Learning

- Can we train the "representation" without using top-down supervision?



Learn a "good" representation directly?



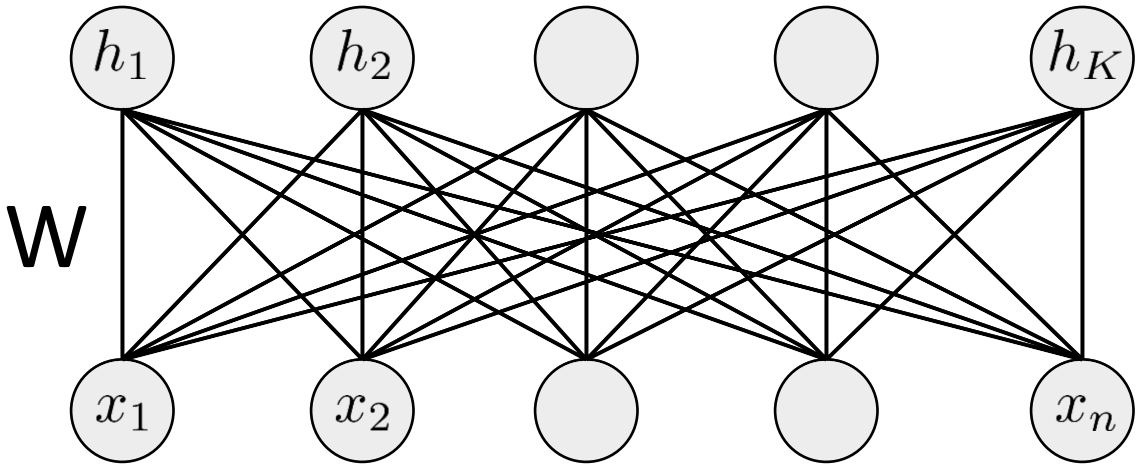
Representation Learning

- What makes a good representation?
  - **Distributed**: roughly, K features represents more than K types of patterns.
    - E.g., K binary features that can vary independently to represent 2^K patterns.
  - **Invariant**: robust to local changes of input; more abstract.
    - E.g., pooled edge features: detect edge at several locations.
  - **Disentangling factors**: put separate concepts (e.g., color, edge orientation) in separate features.

Bengio, Courville, and Vincent (2012)

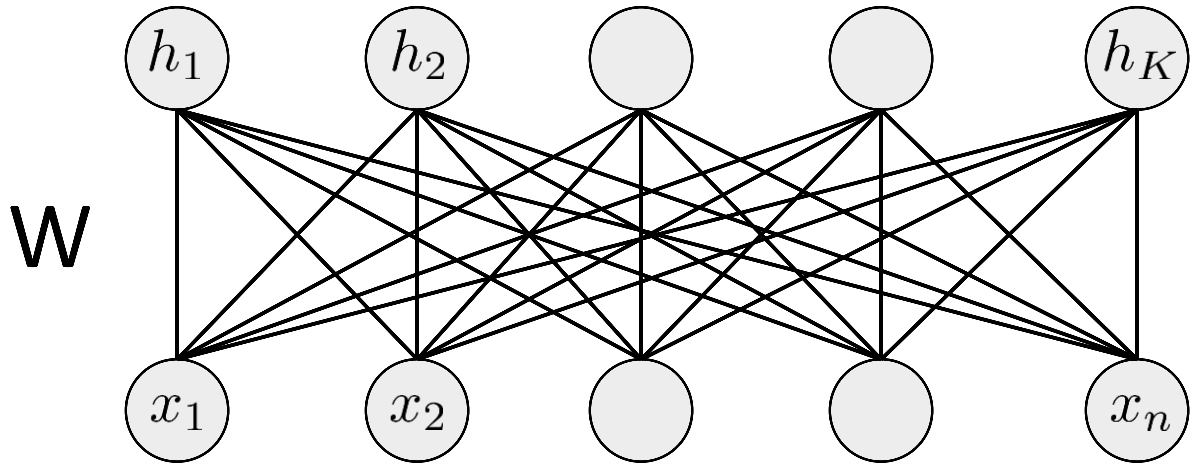
Unsupervised Feature Learning

- Train representations with unlabeled data.
  - Minimize an *unsupervised* training loss.
    - Often based on generic priors about characteristics of good features (e.g., sparsity).
    - Usually train 1 layer of features at a time.
  - Then, e.g., train supervised classifier on top.
    AKA “Self-taught learning” [Raina et al., ICML 2007]



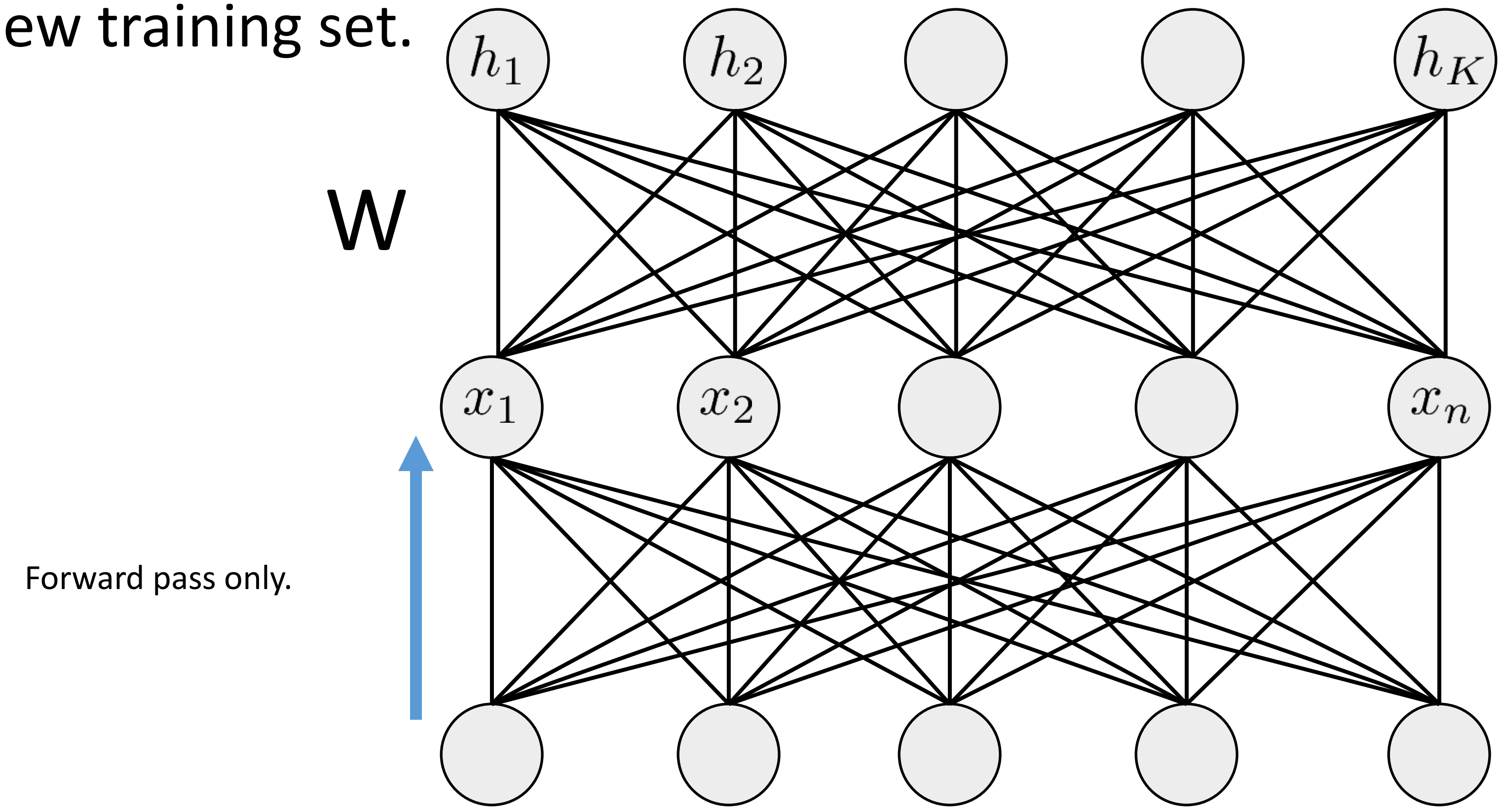
Greedy layer-wise training

- Train representations with unlabeled data.
  - Start by training bottom layer alone.



Greedy layer-wise training

- Train representations with unlabeled data.
  - When complete, train a new layer on top using inputs from below as a new training set.



Unsup Feat Learn Example

- Simple priors for good features:
  - Reconstruction: recreate input from features.

$$\mathcal{L}_{\text{recons}}(W_2, W_1) = \sum_i ||W_2 h(W_1 x^{(i)}) - x^{(i)}||_2^2$$

  - Sparsity: explain the input with as few features as possible.

$$\mathcal{L}_{\text{sparse}}(W_1) = \sum_i ||h(W_1 x^{(i)})||_1$$

TÉCNICO LISBOA

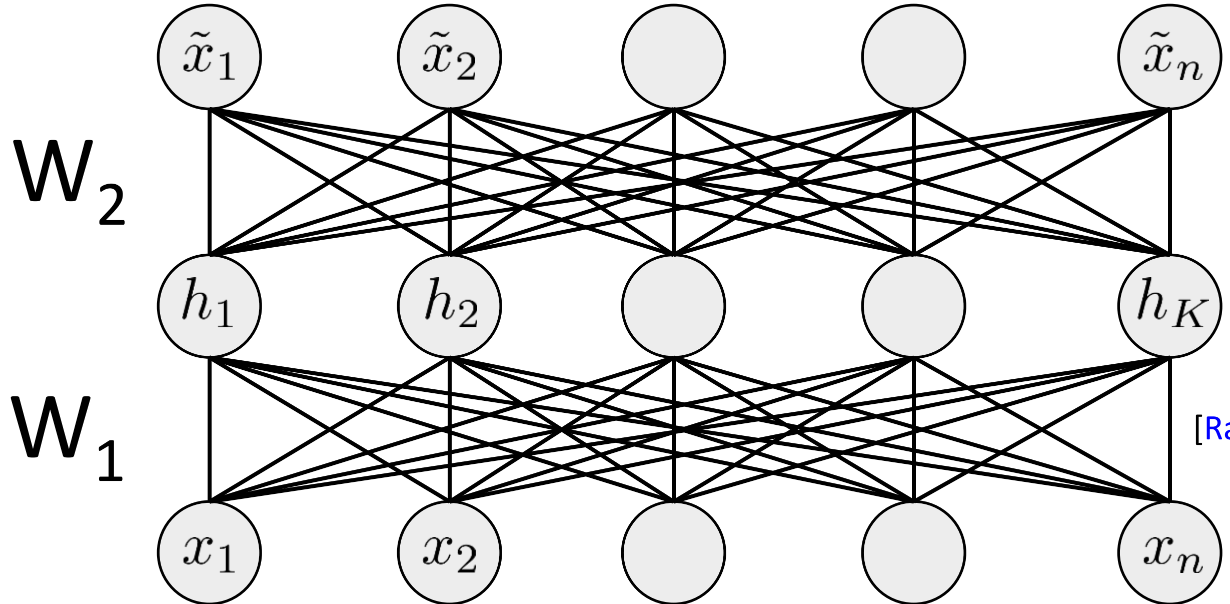
Sparse auto-encoder

- Train two-layer neural network by minimizing:

$$\underset{W_1, W_2}{\text{minimize}} \sum_i ||W_2 h(W_1 x^{(i)}) - x^{(i)}||_2^2 + \lambda ||h(W_1 x^{(i)})||_1$$

$$h(z) = \text{ReLu}(z)$$

- Remove "decoder" and use learned features (h).



[Ranzato et al., NIPS 2006]

Pre-processing

- Unsupervised algorithms more sensitive to pre-processing.
  - Whiten your data. E.g., ZCA whitening:

$$[V, D] := \mathrm{eig}(\mathrm{cov}(X))$$
$$\hat{x}^{(i)} := V(D + I\epsilon)^{-1/2}V^\top(x^{(i)} - \mathrm{mean}(X))$$

  - Contrast normalization often useful.

$$\hat{x} = \frac{x}{\sqrt{Wx^2 + \epsilon}}$$

  - Do these before unsupervised learning at each layer.

[See, e.g., Coates et al., AISTATS 2011; Code at www.stanford.edu/~acoates/]

Group-sparsity

- Simple priors for good features:
  - Group-sparsity:

$$\mathcal{L}_{\text{group-sparse}}(W_1) = \sum_i \sqrt{V[h(W_1 x^{(i)})^2]}$$

  - V chosen to have a "neighborhood" structure. Typically in 2D grid.

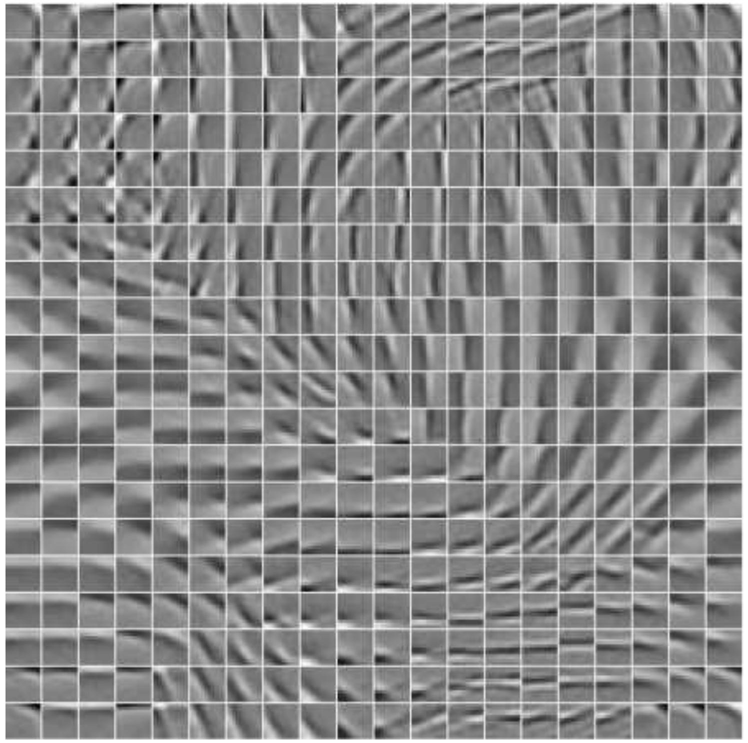
$$V_{ij} = \begin{cases} 1 & \text{if neuron } i \text{ adjacent to } j. \\ 0 & \text{otherwise} \end{cases}$$

Hyvärinen et al., Neural Comp. 2001
Ranzato et al., NIPS 2006
Kavukcuoglu et al., CVPR 2009
Garrigues & Olshausen, NIPS 2010

TÉCNICO LISBOA

What features are learned?

- Applied to image patches:
  - Pool over adjacent neurons to create invariant features.
  - These are *learned* invariances without video.



Predictive Sparse Decomposition
[Kavukcuoglu et al., CVPR 2009]

Works with auto-encoders too.
[See, e.g., Le et al. NIPS 2011]

High-level features?

- Quite difficult to learn 2 or 3 levels of features that perform better than 1 level on supervised tasks.
  - Increasingly abstract features, but unclear how much abstraction to allow or what information to leave out.

Unsupervised Pre-training

- Use as initialization for supervised learning!
  - Features may not be perfect for task, but probably a good starting point.
  - AKA "supervised fine-tuning".

Transfer Learning

- Procedure:
  - Train each layer of features greedily unsupervised.
  - Add supervised classifier on top.
  - Optimize entire network with back-propagation.

Other Unsupervised Criteria

- Neural networks with other unsupervised training criteria.
    - Denoising, in-painting. [Vincent et al., 2008]
    - “Contraction” [Rifai et al., ICML 2011].
    - Temporal coherence [Zou et al., NIPS 2012] [Mobahi et al., ICML 2009]

Summary

- Supervised deep-learning
  - Practical and highly successful in practice. A general-purpose extension to existing ML.
  - Optimization, initialization, architecture matter!

- Unsupervised deep-learning
  - Pre-training often useful in practice.
  - Difficult to train many layers of features without labels.
  - Some evidence that useful high-level patterns are captured by top-level features.

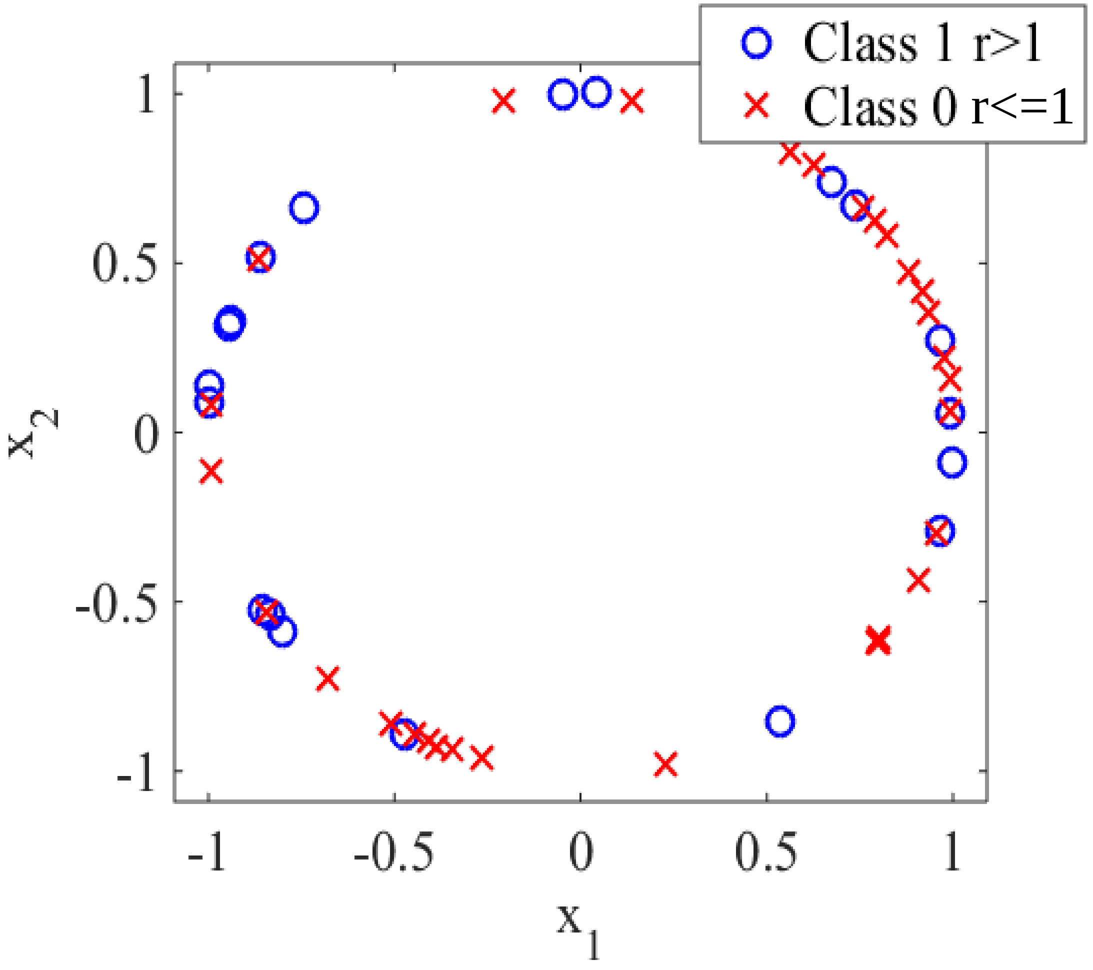
Test example

- Problem definition:

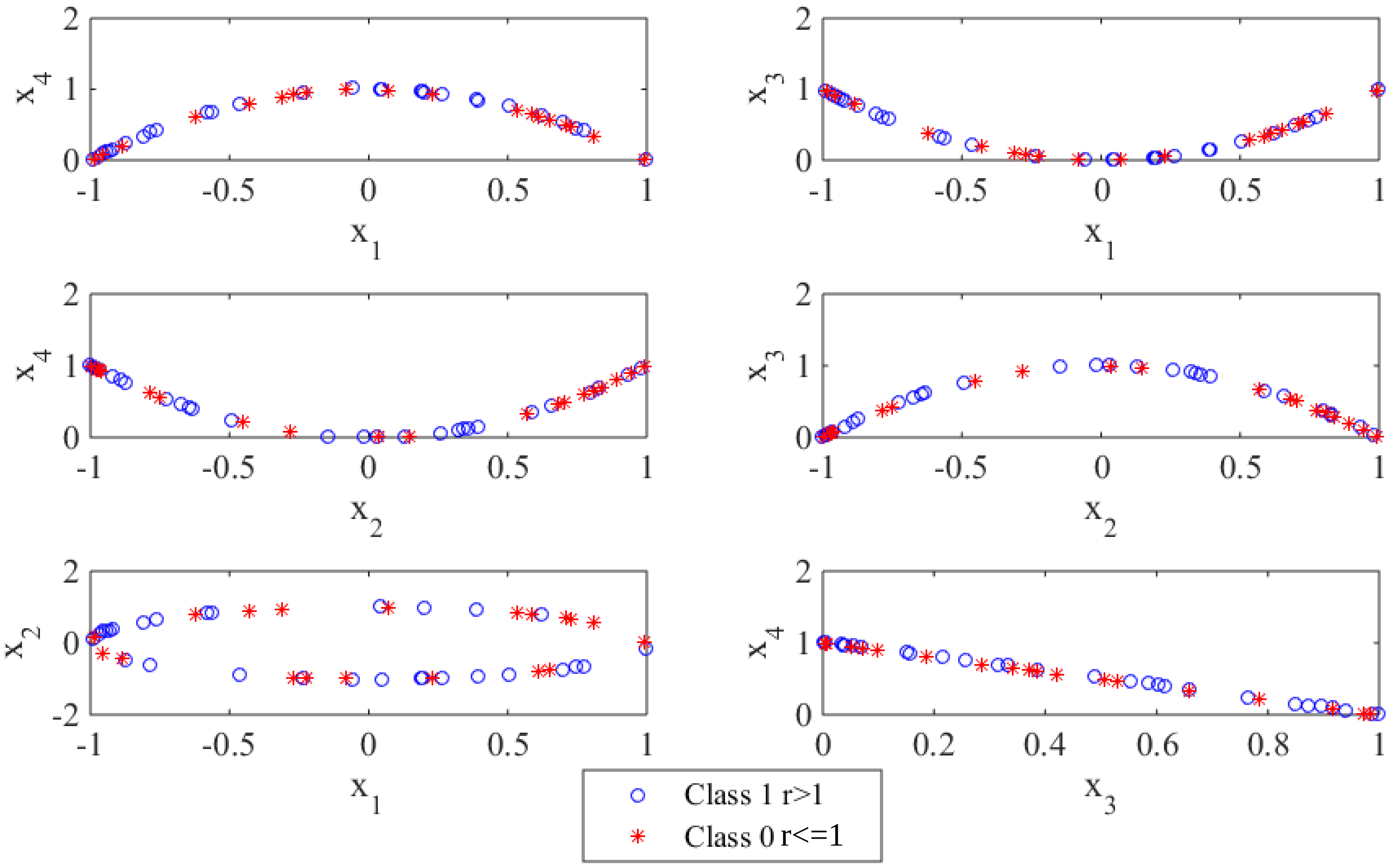
$$x_1 = r\cos(t)$$
$$x_2 = r\sin(t)$$
$$r \in [0.99, 1.01]$$
$$y = r > 1$$

- Features: $F = \begin{bmatrix} x_1 & x_2 & x_1^2 & x_2^2 \end{bmatrix}$

- Output: $y = [0 \quad 1]$



Test example



Test example

- Features: $F = \begin{bmatrix} x_1 & x_2 & x_1^2 & x_2^2 \end{bmatrix}$
- Output: $y = [0 \quad 1]$
- Correlation:

		x_1	x_2	x_3	x_4	
		x_1	x_2	x_1^2	x_2^2	y
x_1	x_1	1.0000	-0.1163	-0.1784	0.1790	-0.1090
x_2	x_2	-0.1163	1.0000	0.2002	-0.2085	-0.1162
x_3	x_1^2	-0.1784	0.2002	1.0000	-0.9995	0.1050
x_4	x_2^2	0.1790	-0.2085	-0.9995	1.0000	-0.0772
	y	-0.1090	-0.1162	0.1050	-0.0772	1.0000

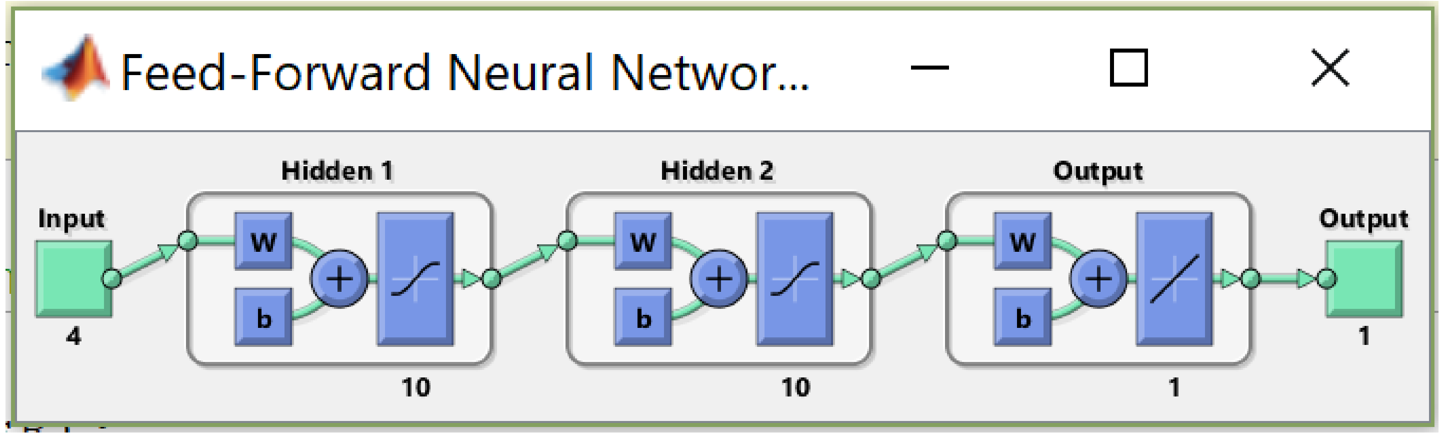
Test example

- All possible combinations of feature subsets:
  - N(1) = {1}, {2}, {3}, {4}
  - N(2) = {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}
  - N(3) = {1,2,3}, {1,2,4}, {1,3,4}, {2,3,4}
  - N(4) = {1,2,3,4}
- Accuraccy for all combinations using fuzzy models:
  - N(1) = [46.1538] [50] [69.2308] [57.6923]
  - N(2) = [53.8462] [50] [53.8462] [50] [50] [96.1538]
  - N(3) = [53.8462] [53.8462] [100] [92.3077]
  - N(4) = [96.1538]

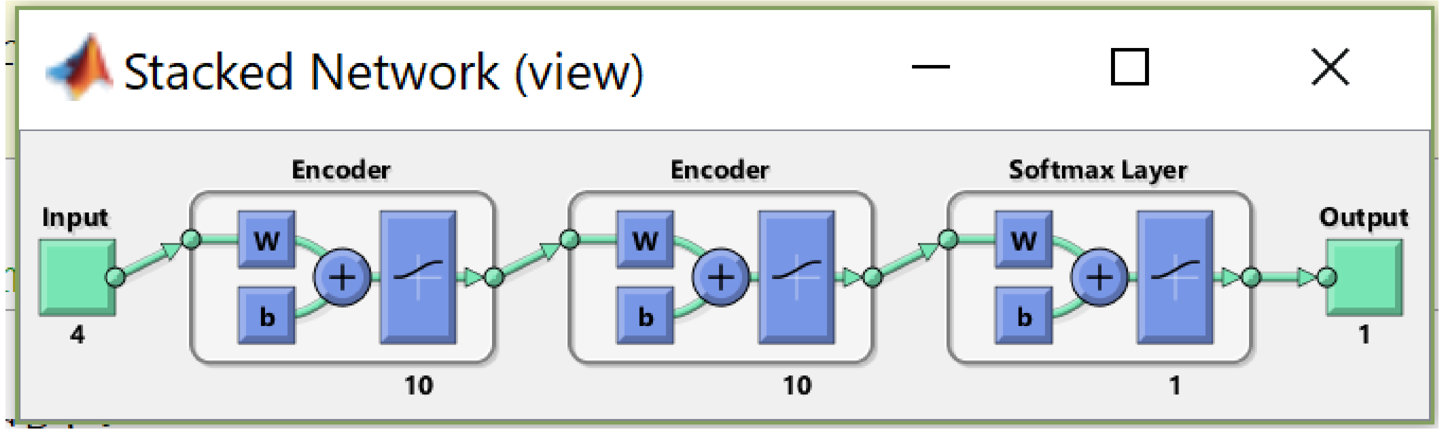
TÉCNICO LISBOA

Test Example

- **Shallow Network:**

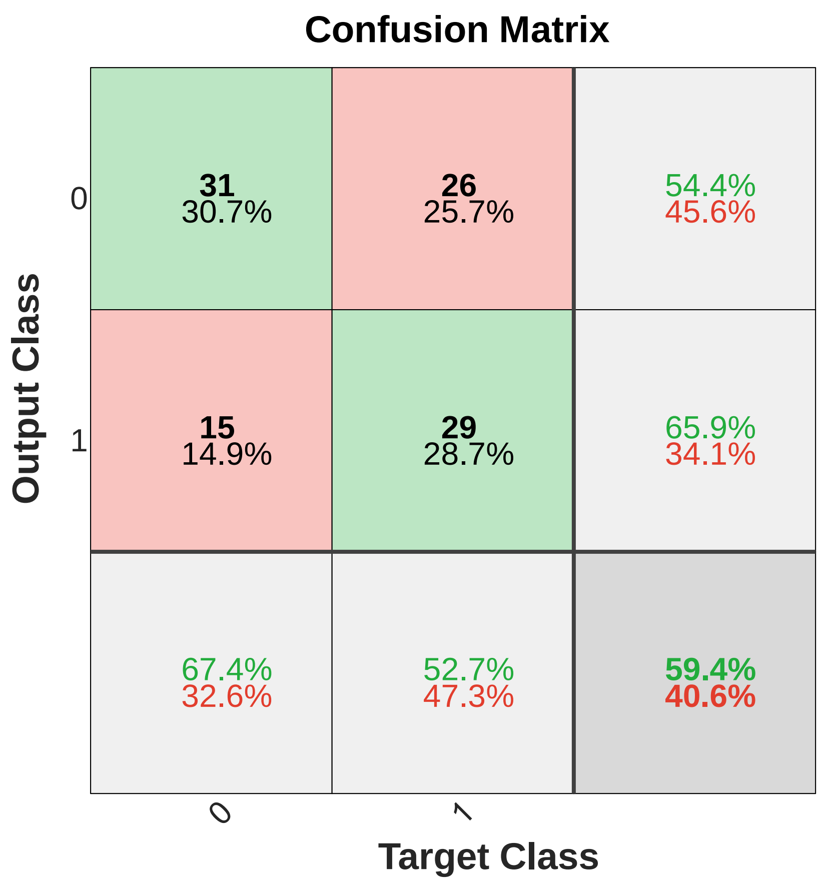


- **Deep Network:**



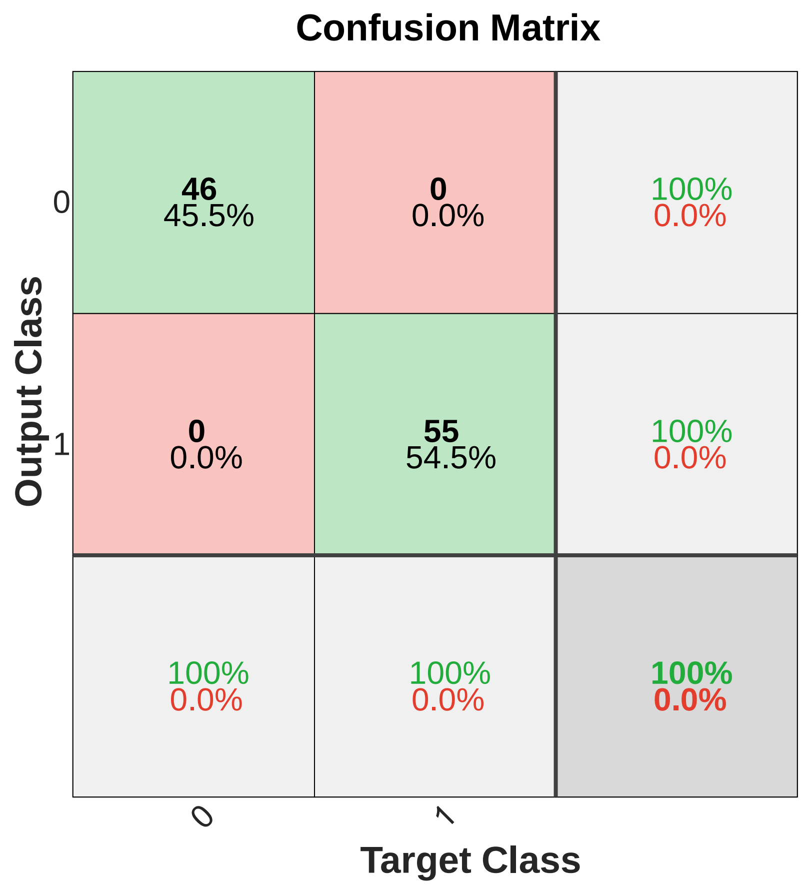
Test Example

Shallow network:


Confusion Matrix

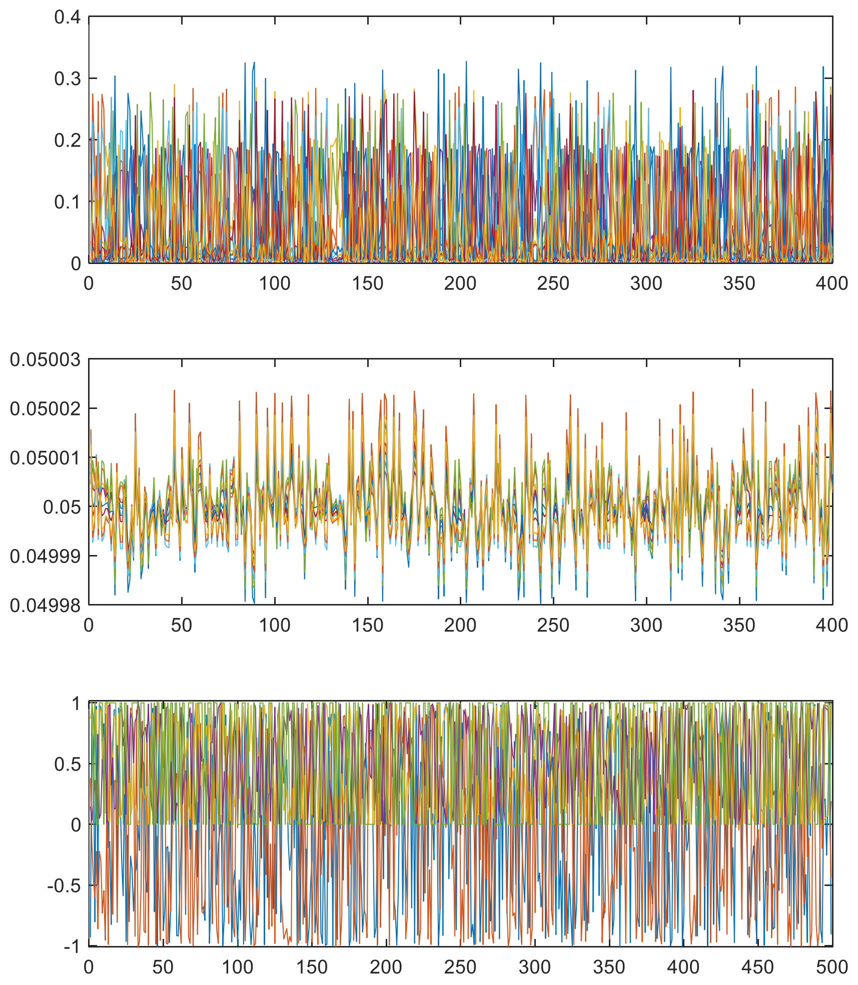
Output Class \ Target Class	0	1	
0	**31** 30.7%	**26** 25.7%	54.4% 45.6%
1	**15** 14.9%	**29** 28.7%	65.9% 34.1%
	67.4% 32.6%	52.7% 47.3%	**59.4%** **40.6%**

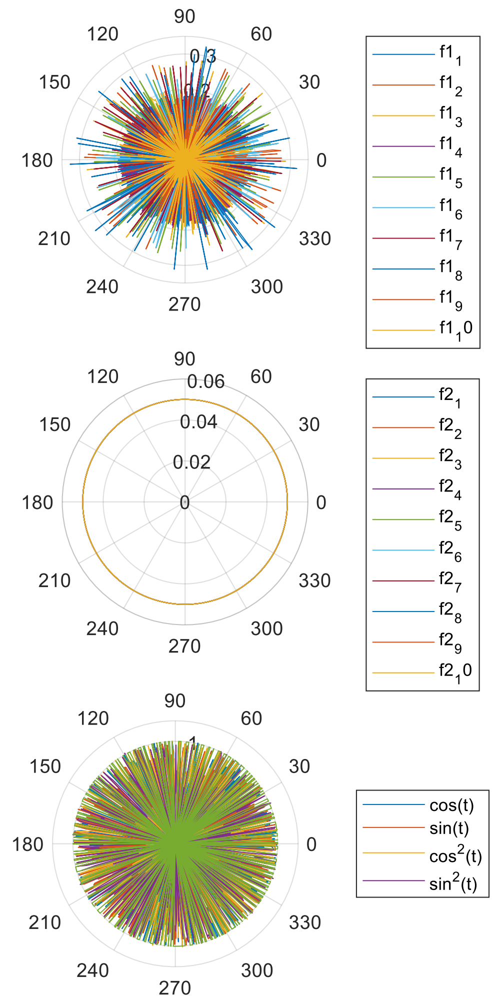
Deep network:


Confusion Matrix

Output Class \ Target Class	0	1	
0	**46** 45.5%	**0** 0.0%	100% 0.0%
1	**0** 0.0%	**55** 54.5%	100% 0.0%
	100% 0.0%	100% 0.0%	**100%** **0.0%**

Test exemple: Extracted Features





References

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

TÉCNICO LISBOA