UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

Soft Computing Research Group

Deep learning

Lluís A. Belanche
Computer Science Department
Universitat Politècnica de Catalunya
belanche@cs.upc.edu

Things you need to know to work in Deep Learning
Part I

Version of February 16, 2025

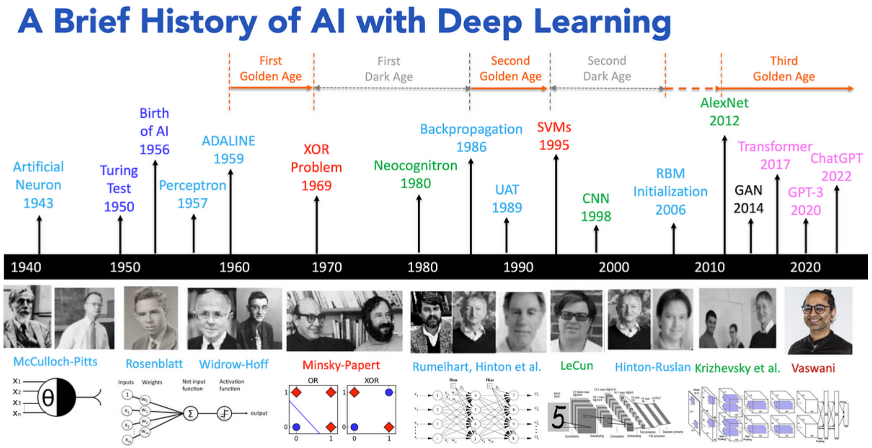
DEEP LEARNING

"The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work - that is correctly to describe phenomena from a reasonably wide area. Furthermore, it must satisfy certain esthetic criteria - that is, in relation to how much it describes, it must be rather simple."

— John von Neumann.

DEEP LEARNING

A timeline of neural networks



A timeline of neural networks
– from `https://medium.com`

The Rosetta stone

Machine Learning	Statistics
model	model
dataset	sample
parameter/weight	parameter/coefficient
train	fit
learn	infer/estimate
regression	regression
classification	discrimination/recognition
clustering	clustering
inputs/features/variables	independent or explanatory variables predictors, regressors
outputs/targets	dependent or response variables
instances/examples	individuals/observations
error/loss function	fit criterion/deviance
training/empirical error	resubstitution/in-sample error
test error	predictive/out-sample error
true/generalization error	risk

Careful with other words! attribute means variable in AI, ...

DEEP LEARNING

Things you NEED to know but are NOT going to be explained (again)

Standard Machine Learning concepts in general:

- All concepts in the previous Table
- Overfitting,
- Bias-variance,
- Regularization,
- Feature selection,
- Curse of dimensionality,
- Basic NN models: MLPs, RBFs, SVMs (?), CNNs (basic),
- Computing gradients: backpropagation and its extensions,
- ...

1. Why do networks with one hidden layer (two layers of weights) “do not work”, if in theory they are universal approximators?
2. Why do models with many hidden layers “work” (in practice) now and didn’t 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions “resistant” to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

DEEP LEARNING
Motivation for DNNs

1. Why do networks with one hidden layer (two layers of weights) "do not work", if in theory they are universal approximators?
2. Why do models with many hidden layers "work" (in practice) now and didn't 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions "resistant" to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

1. Why do networks with one hidden layer (two layers of weights) “do not work”, if in theory they are universal approximators?
2. Why do models with many hidden layers “work” (in practice) now and didn’t 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions “resistant” to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

1. Why do networks with one hidden layer (two layers of weights) “do not work”, if in theory they are universal approximators?
2. Why do models with many hidden layers “work” (in practice) now and didn't 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions “resistant” to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

1. Why do networks with one hidden layer (two layers of weights) "do not work", if in theory they are universal approximators?
2. Why do models with many hidden layers "work" (in practice) now and didn't 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions "resistant" to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

1. Why do networks with one hidden layer (two layers of weights) "do not work", if in theory they are universal approximators?
2. Why do models with many hidden layers "work" (in practice) now and didn't 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
   1. Access to large amounts of data
   2. Much more powerful and parallel hardware (GPUs)
   3. Software with automatic differentiation (Tensorflow, Pytorch,...)
   4. Activation functions "resistant" to the vanishing gradient problem: ReLU and its variants, ...
   5. Better initializations for the weights (RBMs, heuristic rules,...)
   6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

1. Why do networks with one hidden layer (two layers of weights) "do not work", if in theory they are universal approximators?
2. Why do models with many hidden layers "work" (in practice) now and didn't 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions "resistant" to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

1. Why do networks with one hidden layer (two layers of weights) "do not work", if in theory they are universal approximators?
2. Why do models with many hidden layers "work" (in practice) now and didn't 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions "resistant" to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

1. Why do networks with one hidden layer (two layers of weights) “do not work”, if in theory they are universal approximators?
2. Why do models with many hidden layers “work” (in practice) now and didn't 30 years ago?
3. Ideas that have helped the explosion of Deep Learning:
    1. Access to large amounts of data
    2. Much more powerful and parallel hardware (GPUs)
    3. Software with automatic differentiation (Tensorflow, Pytorch,...)
    4. Activation functions “resistant” to the vanishing gradient problem: ReLU and its variants, ...
    5. Better initializations for the weights (RBMs, heuristic rules,...)
    6. Alternatives/variants to SGD: Adam, RMSProp, AdaGrad, ...

DEEP LEARNING

Why do networks with one hidden layer (two layers of weights) “do not work”, if in theory they are universal approximators?

In theory, a one-hidden-layer MLP can approximate any continuous function on a compact domain with enough hidden units.

Inefficiencies in Learning
: it may require an exponentially large number of neurons, making training slow and tuning cumbersome

Difficulties in Optimization
: The gradients can become unstable, making it difficult for standard optimization algorithms (e.g., SGD) to converge

Feature Hierarchy
: Many real-world functions are hierarchical in nature (e.g., vision, speech, and language tasks), but a single hidden layer lacks this hierarchical structure, making it harder to learn

DEEP LEARNING

Why do networks with one hidden layer (two layers of weights) “do not work”, if in theory they are universal approximators?

In theory, a one-hidden-layer MLP can approximate any continuous function on a compact domain with enough hidden units.

Inefficiencies in Learning
: it may require an exponentially large number of neurons, making training slow and tuning cumbersome

Difficulties in Optimization
: The gradients can become unstable, making it difficult for standard optimization algorithms (e.g., SGD) to converge

Feature Hierarchy
: Many real-world functions are hierarchical in nature (e.g., vision, speech, and language tasks), but a single hidden layer lacks this hierarchical structure, making it harder to learn

Why do networks with one hidden layer (two layers of weights) “do not work”, if in theory they are universal approximators?

In theory, a one-hidden-layer MLP can approximate any continuous function on a compact domain with enough hidden units.

Inefficiencies in Learning it may require an exponentially large number of neurons, making training slow and tuning cumbersome

Difficulties in Optimization The gradients can become unstable, making it difficult for standard optimization algorithms (e.g., SGD) to converge

Feature Hierarchy Many real-world functions are hierarchical in nature (e.g., vision, speech, and language tasks), but a single hidden layer lacks this hierarchical structure, making it harder to learn

Hardware limitations, poor optimization techniques, (relatively) small datasets, and lack of effective architectures.

Advances in Hardware :

- **30 years ago**: Training was limited to CPUs, making deep networks infeasible due to slow computation.
- **Today**: Graphics Processing Units (GPUs) and specialized hardware (e.g., Tensor Processing Units, TPUs) enable massively parallel computations, drastically reducing training time.

DEEP LEARNING

Why do DEEP models with many hidden layers "work" (in practice) now and didn't 30 years ago?

Hardware limitations, poor optimization techniques, (relatively) small datasets, and lack of effective architectures.

Better Optimization Techniques :

- Traditional stochastic gradient descent (SGD) struggled with deep networks due to the vanishing/exploding gradient problem.
- Modern optimizers such as Adam, RMSprop, and momentum-based methods stabilize training and accelerate convergence.
- Batch Normalization (2015) helps mitigate internal covariate shift, improving training stability.

Hardware limitations, poor optimization techniques, (relatively) small datasets, and lack of effective architectures.

Solving the Vanishing Gradient Problem :

- **30 years ago**: Activation functions like sigmoid and tanh caused vanishing gradients, making training very deep networks infeasible.
- **Today**:
  - ReLU (Rectified Linear Unit) activation avoids vanishing gradients.
  - Improved weight initialization techniques (e.g., Xavier, He initialization) prevent signal degradation.
  - Residual Networks (ResNets, 2015) use skip connections to allow gradients to propagate through deep architectures.

DEEP LEARNING

Why do DEEP models with many hidden layers “work” (in practice) now and didn’t 30 years ago?

Hardware limitations, poor optimization techniques, (relatively) small datasets, and lack of effective architectures.

Availability of Large-Scale Training Data :

- **30 years ago**: Datasets were small, limiting the ability of deep networks to generalize.
- **Today**: Large-scale datasets such as ImageNet, Common Crawl, and YouTube-8M enable deep networks to learn complex patterns effectively.
- Data augmentation and self-supervised learning further enhance training data without requiring explicit labels.

DEEP LEARNING

Why do DEEP models with many hidden layers “work” (in practice) now and didn’t 30 years ago?

Hardware limitations, poor optimization techniques, (relatively) small datasets, and lack of effective architectures.

Regularization Techniques :

- Dropout (2014) prevents co-adaptation of neurons.
- Careful L_2 regularization or weight decay.
- Data augmentation to artificially increase dataset size.

DEEP LEARNING

Why do DEEP models with many hidden layers “work” (in practice) now and didn’t 30 years ago?

Hardware limitations, poor optimization techniques, (relatively) small datasets, and lack of effective architectures.

Architectural Innovations :

- **30 years ago**: Most networks were simple fully connected architectures, which do not scale well.
- **Today**:
  - Convolutional Neural Networks (CNNs) (1990s–2012) enable efficient learning in vision tasks.
  - Transformers (2017) revolutionized NLP by replacing recurrent networks (RNNs, LSTMs).
  - Residual Networks (ResNets) and Dense Networks (DenseNets) allow very deep networks to be trained effectively.

DEEP LEARNING: Knowledge pearls

Pearl #1

vs.

(example due to Jason Weston)

- Suppose we have a dataset D_{100} of 50 images of elephant faces and 50 of tiger faces, which we digitize into 100×100 pixel RGB images, so we have $\boldsymbol{x} \in \{0, \ldots, 255\}^d$ where $d = 3 \cdot 10^4$
- Given a *new* image, we want to answer the question: is it an elephant or a tiger? [we assume it is one or the other]

Define a **classifier** as a function $f : \mathbb{R}^d \to \{-1, +1\}$

Key fact: Take a data sample $D_n = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^n, y^n)\}$; for any f there exists f^* s.t.

1. f and f^* coincide in all the n images in D_n
2. f and f^* differ in at least one of all possible images (not in D_n)

Moral: ML is about learning **general structure** from data (the data regularities that will appear in **any** data sample) and nothing else.

DEEP LEARNING: Knowledge pearls

Pearl #2

Complete the series! 2, 4, 6, 8, ...

Answer 1: 132 (model 1: $f(n) = n^4 - 10n^3 + 35n^2 - 48n + 24$)

Answer 2: 10 (model 2: $f(n) = 2n$)

How can we rule out the more complex one? (and many others)

1. Supply more "training" data: 2, 4, 6, 8, 10, 12, 14, ...
2. Regularize: add a penalty to higher-order terms
3. Reduce the hypothesis space (e.g. restrict to quadratic models)

DEEP LEARNING: Knowledge pearls

Pearl #2

Complete the series! 2, 4, 6, 8, ...
Answer 1: 132 (model 1: $f(n) = n^4 - 10n^3 + 35n^2 - 48n + 24$)
Answer 2: 10 (model 2: $f(n) = 2n$)
How can we rule out the more complex one? (and many others)

1. Supply more "training" data: 2, 4, 6, 8, 10, 12, 14, ...
2. Regularize: add a penalty to higher-order terms
3. Reduce the hypothesis space (e.g. restrict to quadratic models)

Complete the series! 2, 4, 6, 8, ...
Answer 1: 132 (model 1: $f(n) = n^4 - 10n^3 + 35n^2 - 48n + 24$)
Answer 2: 10 (model 2: $f(n) = 2n$)
How can we rule out the more complex one? (and many others)

1. Supply more "training" data: 2, 4, 6, 8, 10, 12, 14, ...
2. Regularize: add a penalty to higher-order terms
3. Reduce the hypothesis space (e.g. restrict to quadratic models)

DEEP LEARNING: Knowledge pearls
Pearl #2

Complete the series! 2, 4, 6, 8, ...
Answer 1: 132 (model 1: $f(n) = n^4 - 10n^3 + 35n^2 - 48n + 24$)
Answer 2: 10 (model 2: $f(n) = 2n$)
How can we rule out the more complex one? (and many others)

1. Supply more "training" data: 2, 4, 6, 8, 10, 12, 14, ...
2. Regularize: add a penalty to higher-order terms
3. Reduce the hypothesis space (e.g. restrict to quadratic models)

Complete the series! 2, 4, 6, 8, ...
Answer 1: 132 (model 1: $f(n) = n^4 - 10n^3 + 35n^2 - 48n + 24$)
Answer 2: 10 (model 2: $f(n) = 2n$)
How can we rule out the more complex one? (and many others)

1. Supply more "training" data: 2, 4, 6, 8, 10, 12, 14, ...
2. Regularize: add a penalty to higher-order terms
3. Reduce the hypothesis space (e.g. restrict to quadratic models)

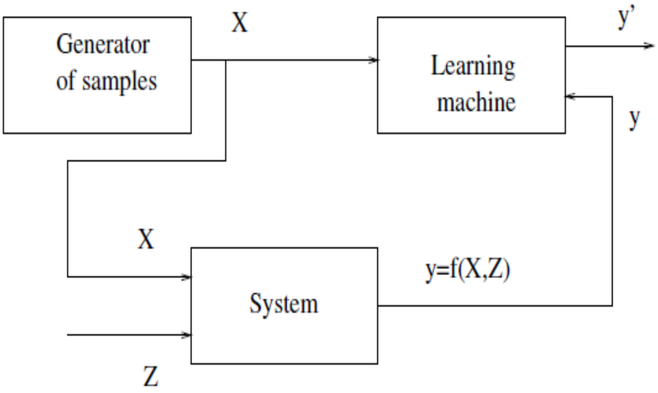
Complete the series! 2, 4, 6, 8, ...
Answer 1: 132 (model 1: $f(n) = n^4 - 10n^3 + 35n^2 - 48n + 24$)
Answer 2: 10 (model 2: $f(n) = 2n$)
How can we rule out the more complex one? (and many others)

1. Supply more "training" data: 2, 4, 6, 8, 10, 12, 14, ...
2. Regularize: add a penalty to higher-order terms
3. Reduce the hypothesis space (e.g. restrict to quadratic models)

Complete the series! 2, 4, 6, 8, ...
Answer 1: 132 (model 1: $f(n) = n^4 - 10n^3 + 35n^2 - 48n + 24$)
Answer 2: 10 (model 2: $f(n) = 2n$)
How can we rule out the more complex one? (and many others)

1. Supply more "training" data: 2, 4, 6, 8, 10, 12, 14, ...
2. Regularize: add a penalty to higher-order terms
3. Reduce the hypothesis space (e.g. restrict to quadratic models)

Pearl #3



X are the measured variables
Z are the non-measured variables
y is the output of true function
y' (or $\hat{y}$) is the output of the model

Let $((\boldsymbol{x}, \boldsymbol{z}), y)$ represent a (complete) input-output relation between data objects:

1. The true relation is $f^* : \mathcal{X} \times \mathcal{Z} \to \mathcal{Y}$, that is $f^*(\boldsymbol{x}, \boldsymbol{z}) = y$.
2. When we measure data about f^*, in the form $D_n = \{(\boldsymbol{x}^i, y^i)\}$ $i = 1, \ldots, n$ we measure the $\boldsymbol{x}$ portion of the input variables $\underline{\text{only}}$.
3. Therefore, the relation between $\boldsymbol{x}$ and y becomes **stochastic**: $\boxed{\text{For every value of } \boldsymbol{x}^i\text{, there is a distribution of values for } y^i.}$

Cats vs. Dogs image Classification

Figure 1: Main features as weel as noise are veryy well by a CNN treated

DEEP LEARNING: Knowledge pearls

Pearl #4

Using exactly the *same* call to R's nnet{**nnet**} three times ...

```
# weights:  19              # weights:  19              # weights:  19
initial  value 143.78535    initial  value 147.09410    initial  value 140.51
iter  10 value 82.67353     iter  10 value 70.01521     iter  10 value 87.626
iter  20 value 7.343014     iter  20 value 34.64103     iter  20 value 33.779
iter  30 value 3.130778     iter  30 value 21.04111     iter  30 value 7.5768
iter  40 value 2.817692     iter  40 value 6.781165     iter  40 value 7.0271
iter  50 value 2.803824     iter  50 value 1.729377     ...... <removed>
iter  60 value 2.781383     iter  60 value 1.051125     iter 100 value 6.4489
iter  70 value 2.743949     iter  70 value 0.725090     iter 110 value 6.4488
iter  80 value 2.310192     iter  80 value 0.318526     iter 120 value 6.4485
iter  90 value 0.148647     iter  90 value 0.136883     final  value 6.446125
iter 100 value 0.031570     iter 100 value 0.116817     converged
iter 110 value 0.018951     iter 110 value 0.111052
...... <removed>            ...... <removed>
iter 890 value 0.000145     iter1970 value 0.001225
iter 900 value 0.000087     iter1980 value 0.001212
final  value 0.000087       iter1990 value 0.001206
converged                   iter2000 value 0.001198
final  value 0.001198       stopped after 2000 iterations
```