

Introdução a Machine Learning e Aprendizagem Estatística

26º SINAPE

Taiane Schaedler Prass e Guilherme Pumi

Departamento de Estatística e PPGEst - UFRGS

Elementos de machine learning e aprendizado estatístico

Tópicos

- O que é machine learning.
- O que é big data.
- Tipos de dados: estruturados versus não estruturados.
- Cultura Inferencial versus preditiva.
- Maldição da dimensionalidade.

O que é Machine Learning?

Origem do termo *Machine Learning*

- Em 1959 Arthur L. Samuel, um cientista da computação da IBM, pioneiro no campo de inteligência artificial, escreveu o primeiro programa de computador para jogar damas.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, 3(3), 210-229.

- No artigo dois procedimentos de aprendizado de máquina foram investigados em detalhes usando o jogo de damas.

Segundo Samuel (1959),

- suficiente trabalho foi realizado para comprovar o fato de que
*um computador pode ser programado
de forma que aprenda a jogar um jogo de damas
melhor do que pode ser jogado pela pessoa que escreveu o programa*

Segundo Samuel (1959),

- o computador pode aprender a fazer isso
 - em um período de tempo notavelmente curto (8 ou 10 horas de tempo de jogo da máquina)
 - quando são fornecidas apenas
 - (a) *as regras do jogo,*
 - (b) *um senso de direção e*
 - (c) *uma lista redundante e incompleta de parâmetros que se acredita terem alguma relação com o jogo, mas cujos sinais corretos e pesos relativos são desconhecidos e não especificados*

Samuel (1959) - Introdução

- *The studies reported here have been concerned with the programming of a digital computer to behave in a way which, if done by human beings or animals, would be described as involving the process of learning. . . .*

there is obviously a very large amount of work, now done by people, which is quite trivial in its demands on the intellect but does, nevertheless, involve some learning.

Samuel (1959) - Introdução

- *We have at our command computers with adequate data-handling ability and with sufficient computational speed to make use of machine-learning techniques,*

but our knowledge of the basic principles of these techniques is still rudimentary.

Lacking such knowledge, it is necessary to specify methods of problem solution in minute and exact detail, a time-consuming and costly procedure.

Samuel (1959) - Introdução

- *Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort.*



Essa frase tem sido citada na literatura para expressar a ideia que o campo de machine learning está preocupado com a questão de como construir programas de computador que melhorem automaticamente com a experiência.

◆ AI Overview

Machine Learning (aprendizado de máquina) é um subcampo da Inteligência Artificial (IA) que dá aos computadores a capacidade de aprender sem serem explicitamente programados.  Data Science Academy +1

Definições Clássicas

As duas definições fundamentais que formam a base da disciplina são:

- **Arthur Samuel (1959):** "O campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados."
- **Tom Mitchell (1997):** "Diz-se que um programa de computador aprende com a **experiência E** em relação a uma certa classe de **tarefas T** e [medida de] **desempenho P**, se o seu desempenho em T, conforme medido por P, melhora com a experiência E."  YouTube · Bóson Treina... +1

Por décadas, essa frase alimentou um *mito*: máquinas que aprenderiam e evoluiriam *sozinhas*, sem intervenção humana.

Mas o que de fato acontece é outra coisa:

- Não se programam *regras específicas* para cada tarefa que a máquina deve resolver;
- Programa-se o *mecanismo de aprendizagem* em si – o algoritmo, a função de perda, o critério de otimização usado para ajustar o modelo aos dados.

Mesmo hoje, com sistemas cada vez mais autônomos, a máquina não aprendeu a aprender sozinha – ela foi **programada para aprender a aprender.**

Machine Learning versus Statistical Learning

Ingredientes básicos:

algoritmos/métodos de computador automatizados +
modelos estatísticos para “aprender” (descobrir) padrões ocultos a
partir de dados.

Fórmula básica:

Pergunta empírica (o que queremos aprender) +
dados (a partir do que queremos aprender) +
modelo (estatístico) + função de perda +
algoritmos de otimização =
Métodos de Machine Learning.

Mais do que simples modelos de previsão...

- Tradicionalmente, os métodos de machine learning tem sido empregados, principalmente, para **previsão**.

Recentemente, observa-se um crescente interesse na sua aplicação para a **inferência causal**. Em particular, na área de econometria.

- **agrupamento** e **análise textual** são áreas que têm ganhado bastante destaque no âmbito do machine learning, especialmente com os LLM's (GPT's - Generative Pre-trained Transformers).

Ambas essas áreas têm despertado grande interesse devido à sua capacidade de **extrair informações valiosas** e **padrões significativos** a partir de conjuntos de **dados complexos**.

Quando os métodos de machine learning são **fundamentados em bases estatísticas sólidas**, são denominados **métodos de aprendizado estatístico** (*Statistical Learning*).

Em resumo:

Machine learning

- Objetivo principal: previsão, classificação, reconhecimento de padrões, análise de clusters, etc.
- Não se dá muita atenção à inferência ou análise causal, pelo menos do ponto de vista da ciência da computação.
- A interpretação não é necessariamente um ingrediente-chave.

Statistical learning

- Dedicar mais atenção à inferência, interpretação, à fundamentação matemática e à análise causal.

O que é Big Data?

*The sexy job in the next ten years will be statisticians.
Because now we really do have essentially free and ubiquitous data.
So the complimentary factor is the ability to understand that data and
extract value from it.*

*Hal Varian, Chief Economist, Google
January, 2009*

- Com o aparecimento da internet, a quantidade de dados disponíveis aumentou abruptamente: da “era” do terabyte para o petabyte, e a desde 2015, a **era do zettabyte**.
- Atualmente usuários de internet geram **quintilhões de bytes diariamente**, sem considerar dados gerados por experimentos de pesquisa, satélites, organizações de pesquisa de opinião, etc.

O termo big data

- foi utilizado pela primeira vez pela NASA na década de 1990 para descrever os **grandes conjuntos de dados e os limites computacionais** da época

Cox, M. and Ellsworth, D. (1997).

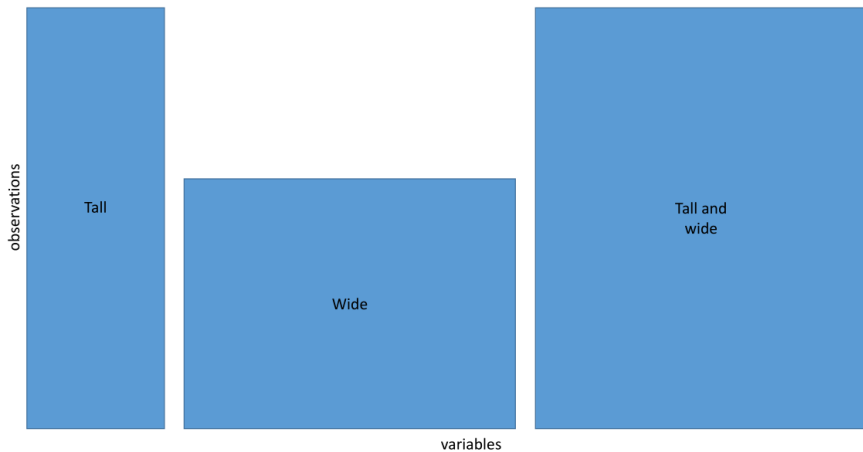
Application-controlled demand paging for out-of-core visualization.

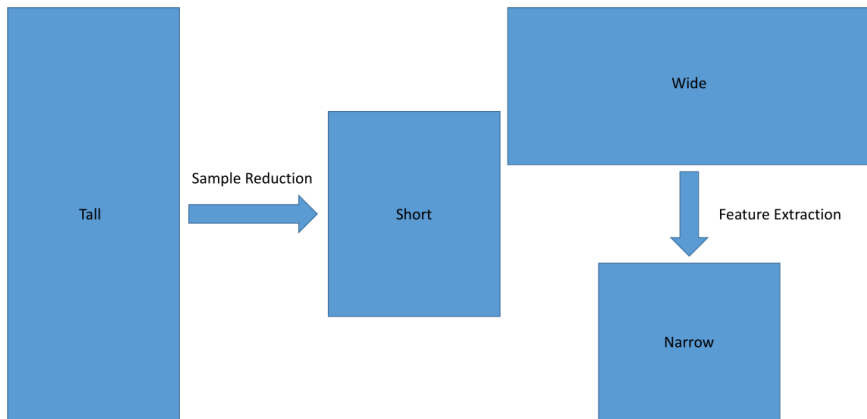
<https://ntrs.nasa.gov/api/citations/20020046803/downloads/20020046803.pdf>

- vem sendo empregado como uma definição para grandes volumes de dados

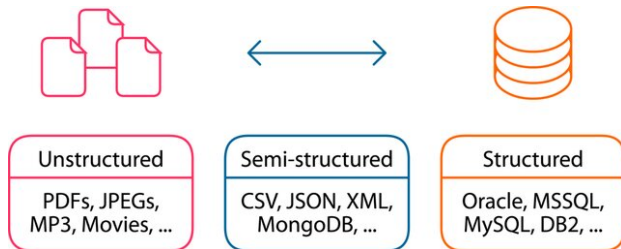
*Big data is a term that describes the **large volume of data** - both **structured** and **unstructured** - that inundates a business on a day-to-day basis. But it's not the amount of data that's important. It's what organizations do with the data that matters. Big data can be analyzed for insights that lead to better decisions and strategic business moves.*

<https://www.sas.com/en-us/insights/big-data/what-is-big-data.html>





Dados estruturados vs não-estruturados



<https://doi.org/10.1049/wss2.12051>

Essas três categorias diferem em termos de **organização**, **formatos** e **facilidade de processamento**.

De forma resumida podemos dizer que, enquanto os dados **estruturados** são altamente organizados e adequados para análises tradicionais de banco de dados, os dados **semi-estruturados** e **não-estruturados** são mais complexos de serem tratados e geralmente requerem abordagens mais flexíveis e técnicas avançadas para extrair *insights* valiosos.

Modelagem Estatística: as duas culturas

There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown.

Breiman, L. (2001). Statistical Modeling: The Two Cultures. Statistical Science, Vol. 16, No. 3, 199-231

Breiman's Black Boxes:

Pense nos dados como sendo gerados por uma caixa preta na qual um vetor de variáveis de entrada $\mathbf{X}$ (variáveis independentes) entra de um lado e, do outro lado, a variável de resposta Y sai. Dentro da caixa preta, a natureza funciona para associar as variáveis preditoras com as variáveis de resposta



Existem dois objetivos na análise dos dados:

- **Previsão.** Ser capaz de prever quais serão as respostas para futuras variáveis de entrada;
- **Informação.** Extrair alguma informação sobre como a natureza está associando as variáveis de resposta às variáveis de entrada.

Existem duas abordagens para atingir esses objetivos

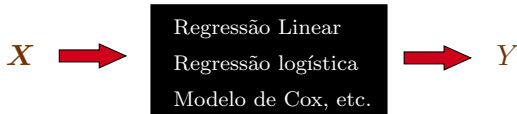
- Cultura de modelagem de dados (*data modeling culture*)
- Cultura de modelagem algorítmica (*algorithmic modeling culture*)

Cultura de modelagem de dados

- A análise começa com a suposição de um modelo estocástico para o interior da caixa preta.

Uma abordagem usual é assumir que os dados são amostras independentes gerados através de uma estrutura do tipo $Y = f(\mathbf{X}, \varepsilon, \boldsymbol{\theta})$, onde ε denota o erro aleatório e $\boldsymbol{\theta}$ é um vetor de parâmetros.

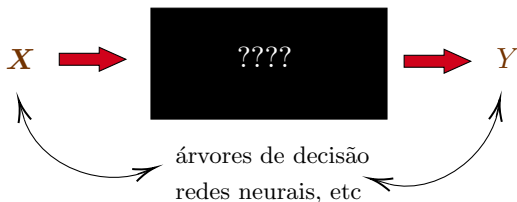
- Os valores dos parâmetros são estimados a partir dos dados e o modelo é então utilizado para obter informações e/ou fazer previsões.



- Validação do modelo: regra do “Sim ou não”, usando testes de goodness-of-fit e análise residual.

Cultura de modelagem algorítmica

- A análise considera o interior da caixa como complexo e desconhecido.
- O objetivo é encontrar uma função $f(\mathbf{X})$ (um algoritmo) para prever a resposta Y a partir de $\mathbf{X}$.



- Validação do modelo: medidas do poder preditivo.

Pra que estimar f ? As duas culturas.

Objetivo inferencial

- Quais preditores são importantes?
- Qual a relação entre a resposta e cada um dos preditores?
- Qual o efeito que mudar o valor de um dos preditores tem na variável resposta?

Cultura de modelagem de dados

- 👉 Objetivo principal é a inferência
- 👉 O modelo (probabilístico) é considerado correto - a incerteza é associada exclusivamente à estimação (dos parâmetros) da f
- 👉 Interpretação (dos parâmetros) é importante.

Objetivo preditivo

Baseado nos dados, gostaríamos de prever $\hat{y} \approx \hat{f}(x_0)$ para algum x_0 não observado de interesse.

Cultura de modelagem algorítmica

- ➡ Objetivo é a previsão
- ➡ O modelo é apenas utilizado para motivar bons algoritmos de previsão
- ➡ Tipicamente nenhum modelo probabilístico é assumido.

Como estimar f ? Os dois Extremos

*Between these two extremes lies the intermediate tranche, where most of the action in data mining and machine learning is. The intermediate tranche is where the finite-dimensional methods confront the **Curse of Dimensionality** on their way to achieving good approximations to the nonparametric setting.*

Clarke et al. (2009). Principles and theory for DMML, p5.

Hipótese geral: $Y = f(\mathbf{X}) + \varepsilon$

Abordagem paramétrica:

- $f = f_{\theta}$ é uma função (estrutura) conhecida. Objetivo é estimar θ
- ε segue uma determinada distribuição de probabilidade

Exemplo clássico: o modelo de regressão linear múltipla, onde
 $Y = \mathbf{X}'\boldsymbol{\beta} + \varepsilon$, com $\varepsilon \sim N(0, \sigma^2)$ independentes.

Hipótese geral: $Y = f(X) + \varepsilon$

Abordagem não-paramétrica:

- f é uma função suficientemente suave. Objetivo é estimar f .
- Pouca ênfase é dada à ε : a grande fonte de variação no modelo está na incerteza associada à f .

Isso é razoável porque, fora grandes desvios de ε 's independentes, simétricos e unimodais, a fonte dominante de incerteza advém da estimativa de f .

- A suavidade de f é crucial para o bom comportamento de estimadores não-paramétricos.
 - ☞ métodos baseados em kernels assumem que f possui uma determinada regularidade nas suas derivadas.
 - ☞ Splines assumem que o comportamento local de f pode ser aproximada por polinômios.

Fontes de incerteza

All models are wrong, but some models are useful
George Box (Box and Draper, 1987, pg 424)

- No contexto paramétrico, aumentar n diminui a variância dos estimadores.

Quando n cresce chegamos a um ponto em que mais dados pouco afetarão as estimativas pois a variância do estimador será minúscula.

- Se o modelo estiver errado (e todos estão), ao invés de continuar refinando o modelo, pode ser vantajoso usar uma parte da amostra para selecionar um modelo diferente.

☞ Isto pode diminuir o efeito do modelo estar incorreto (vício de modelo). Isto é, uma vez que aceitamos incerteza no modelo, podemos manter o vício constante e diminuir a incerteza devido ao modelo!

- De forma geral, existem diversas fontes de variabilidade que devem ser equilibradas: erro aleatório, incerteza de parâmetros e viés, incerteza ou especificação inadequada do modelo, incerteza na classe do modelo, erro de generalização, **incerteza do modelo**.
- No contexto paramétrico não há incerteza devida ao modelo, apenas devida aos **parâmetros**.
- No contexto não-paramétrico, a incerteza é exclusiva do **modelo**.

Não há parâmetros e a classe do modelo é tão grande que certamente contém o modelo verdadeiro
- Machine learning está entre esses dois extremos.

Maldição da Dimensionalidade

- Bellman utilizou o termo **maldição da dimensionalidade** para descrever o problema causado pelo aumento exponencial no volume associado à adição de dimensões extras ao espaço Euclidiano

Bellman, R. E. (1957). Dynamic programming. Princeton, NJ: Princeton University Press.

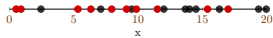
- No contexto de machine learning **maldição da dimensionalidade** refer-se a um conjunto de problemas que surgem ao lidar com dados de alta dimensionalidade.

Em resumo, quando estamos em alta dimensão:

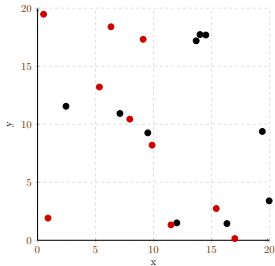
- Todos os conjuntos de dados são **esparsos**;
- A **quantidade de modelos** possíveis cresce extremamente rápido em p ;
- Todos os conjuntos de dados apresentam **multicolinearidade**;
- **Parcimônia** é difícil de se medir e obter.

Esparsidade

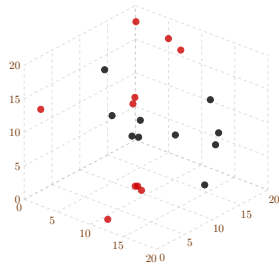
a) 1D - 4 regions



b) 2D - 16 regions



c) 3D - 64 regions



Esparsidade

- Regressão não-paramétrica utiliza os dados para ajustar a estrutura local de f de maneira tipicamente super flexível.

Se nas vizinhanças de um ponto x existem poucas observações, fica difícil decidir como f deve se comportar localmente.

- O problema é que quanto maior p , a quantidade de dados locais tende a zero.

Essencialmente, em dimensões altas não haverão pontos suficientes para estimar f no seu domínio.

Modelos lineares lidam com a esparsidade simplesmente impondo que todos os valores da amostra afetem a estimação de f em todas as vizinhanças!

Muitos modelos

- O número de modelos possíveis para um conjunto de dados cresce exponencialmente com p !

Exemplo: polinômios de grau menor ou igual a 2 em dimensão p . Para $p = 1$ temos 7 possíveis modelos:

$$\begin{array}{lll} \mathbb{E}(Y) = \beta_0, & \mathbb{E}(Y) = \beta_0 + \beta_1 X_1, & \mathbb{E}(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 \\ \mathbb{E}(Y) = \beta_1 X_1, & \mathbb{E}(Y) = \beta_1 X_1 + \beta_2 X_1^2, & \mathbb{E}(Y) = \beta_2 X_1^2. \end{array}$$

Para $p = 2$ são 63. O problema piora se incluirmos modelos com mais termos, como potências mais altas. A situação é ainda mais complicada se o modelo for muito complexo.

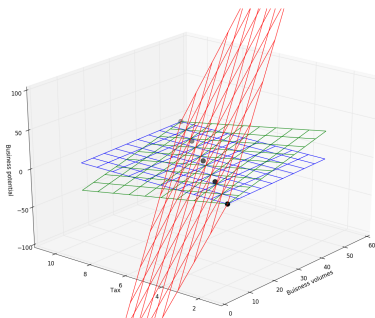
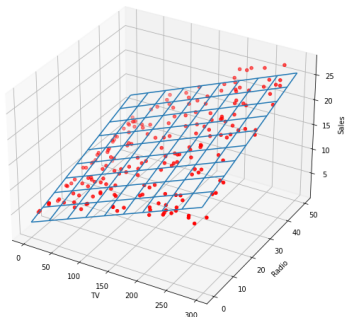
Muitos modelos

- O número de modelos possíveis para um conjunto de dados cresce exponencialmente com p !
- p é moderadamente grande
 - (a) testes de significância em regressão requerem um número grande de testes individuais e a taxa de erro global para o conjunto de inferências será alta
 - (b) ou os testes terão que ser superconservativos ao ponto de serem quase-inúteis).

A impossibilidade prática de identificar corretamente o melhor modelo, ou mesmo um bom modelo, é uma das principais motivações por trás dos métodos ensemble.

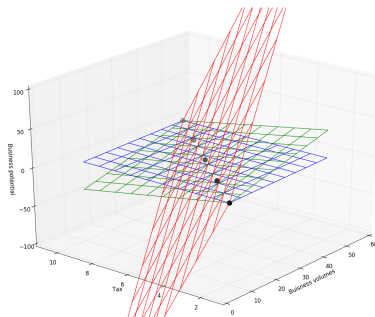
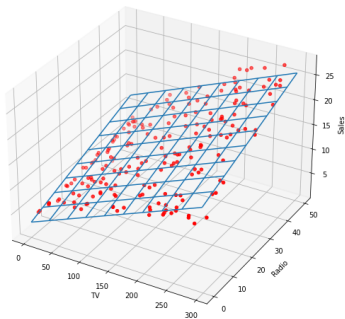
Multicolinearidade

- Na regressão linear, **multicolinearidade** ocorre quando duas ou mais das variáveis explicativas estão **altamente correlacionadas**. Geometricamente, isso significa que todas as **observações** estão próximas a um subespaço afim.



https://duchesnay.github.io/pystatsml/machine_learning/linear_regression.html

Multicolinearidade



https://duchesnay.github.io/pystatsml/machine_learning/linear_regression.html

- Planos/superfícies ajustadas via MQO são bastante estáveis nas vizinhanças dos pontos observados e previsões para vetores similares de variáveis explicativas terão baixas variâncias.

Pode-se girar o plano ajustado em torno da projeção afim sem prejudicar muito o ajuste.

- Em p dimensões, multicolinearidade ocorrerá quando as covariáveis se concentrarem muito próximas a um dos possíveis subespaços afins.
- Quando p cresce, a chance de que as covariáveis estejam contidas num mesmo subespaço afim - simplesmente por coincidência - cresce exponencialmente.
- A regressão não paramétrica ajusta curvas suaves aos dados. A concurvidade é o equivalente não paramétrico da multicolinearidade:

Em analogia com a multicolinearidade, se as variáveis explicativas tendem a se concentrar ao longo de uma curva suave que pertence à família usada para o ajuste, então a previsão e o ajuste serão bons perto da curva projetada, mas ruins em outras regiões. Essa situação é chamada de *concurvidade* (*concurvity*).

- Em p dimensões, multicolinearidade ocorrerá quando as covariáveis se concentrarem muito próximas a um dos possíveis subespaços afins.
- Quando p cresce, a chance de que as covariáveis estejam contidas num mesmo subespaço afim - simplesmente por coincidência - cresce exponencialmente.
- A regressão não paramétrica ajusta curvas suaves aos dados. A concurvidade é o equivalente não paramétrico da multicolinearidade:

Em altas dimensões, quase todo conjunto de variáveis explicativas apresenta multicolinearidade!

Parcimônia

Uma estratégia para lidar com a maldição é o **princípio da parcimônia**.

- Parcimônia é o princípio de preferir explicações **simples** e **concisas** em vez de explicações mais complexas e detalhadas, quando ambas são igualmente eficazes na explicação de um fenômeno.
- Em “estatiquês”: é a preferência que damos a modelos mais simples desde que, ao considerar modelos mais complexos, o ganho no ajuste é marginal.

Pragmaticamente falando, em altas dimensões:

- Se o modelo é de fato complexo, simplesmente pode não ser possível fazer previsões acuradas!
- Se o modelo for simples, então favorecer a escolha de modelos simples aumenta a probabilidade de obtenção de um bom ajuste e boa acurácia preditiva.
- A estratégia é selecionar apenas aquelas variáveis ou função dessas variáveis cujo acréscimo ao modelo é realmente justificado.

Devido à complexidade dos modelos tipicamente empregados em machine learning, parcimônia é um dos itens mais desejados em um modelo!

Tópicos

Algoritmos de Aprendizagem

Métodos de Reamostragem

Ensemble

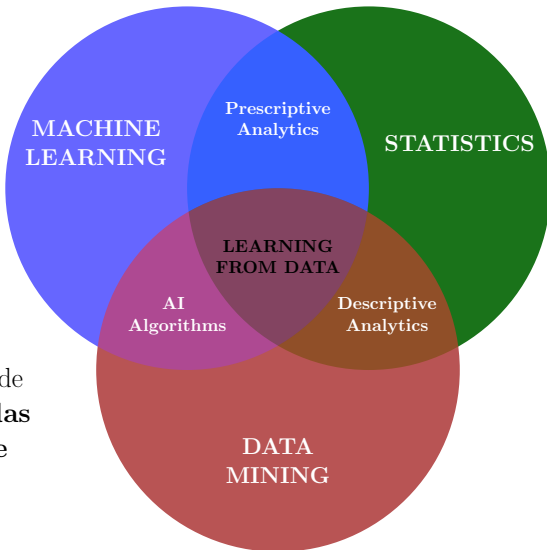
Machine Learning versus Data Mining

Machine learning:

concentra-se na previsão, com base nas propriedades **conhecidas** aprendidas a partir dos dados de treinamento.

Data mining:

concentra-se na descoberta de propriedades **desconhecidas** nos dados. Foca na **análise exploratória** dos dados.



Statistical learning

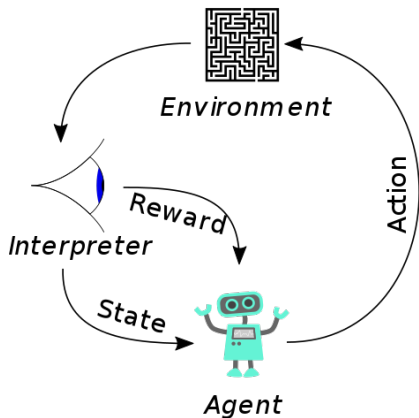
James et al. (2021): *Statistical learning* refere-se a um vasto conjunto de ferramentas para a compreensão dos dados. Estas as ferramentas podem ser classificadas como supervisionadas ou não supervisionadas.

- O *aprendizado supervisionado* envolve a construção de um *modelo estatístico* para prever/estimar uma **saída** com base em uma ou mais **entradas**.
- Com o *aprendizado não supervisionado*, há entradas, mas não há saída de supervisão; entretanto, podemos aprender relacionamentos e estruturas a partir desses dados.

James et al. (2021). *An Introduction to Statistical Learning*

Apredizado por Reforço (Reinforced Learning)

- Um agente interage com seu ambiente em etapas a tempo discreto.
- A cada tempo t , o agente recebe uma observação s_t do ambiente e uma recompensa r_t .
- Em seguida, ele escolhe uma ação a_t do conjunto de todas as ações disponíveis, que é então enviada ao ambiente.
- O ambiente passa para um novo estado s_{t+1} e a recompensa r_{t+1} associada à transição (s_t, a_t, s_{t+1}) é determinada.
- O objetivo de um agente é coletar o máximo de recompensa possível.



Modelo Probabilístico: Processo de decisão Markoviano.

Exemplo Prático:



Primeiros passos:

- 1 Entender o problema que estamos analisando.

Qual a pergunta que precisa ser respondida?

- 2 Pensar sobre o tipo de dados a serem coletados.

Que variáveis potencialmente respondem ou ajudam a responder essa pergunta levantada?

- 3 Identificar os **algoritmos de aprendizagem** que podemos utilizar.

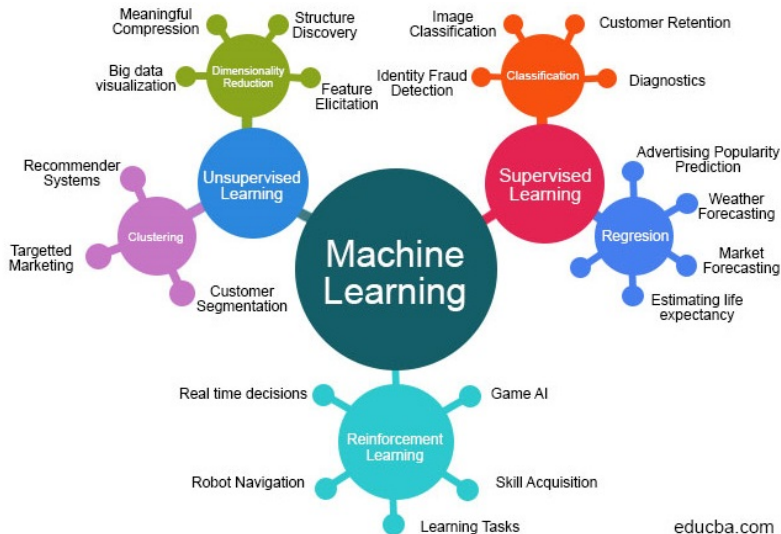
Que tipo de problema temos em mãos?

Previsão, classificação, identificação de padrões?

É necessário reduzir a dimensão dos dados?

Qual a estrutura de dependência dos dados?

Algoritmos de Aprendizagem



No free lunch theorem

Dada a grande variedade de modelos na literatura uma pergunta que surge naturalmente é *“qual é o melhor?”*.

no free lunch theorem: infelizmente, não existe um único melhor modelo que funcione de maneira ideal para todos os tipos de problemas.

D. Wolpert. The lack of a priori distinctions between learning algorithms.
In: Neural Computation 8.7 (1996)

Um conjunto de pressupostos (também chamado de viés indutivo) que funciona bem em um domínio pode funcionar mal em outro.

All models are wrong, but some models are useful
George Box (Box and Draper, 1987, pg 424)

No free lunch theorem

Dada a grande variedade de modelos na literatura uma pergunta que surge naturalmente é *“qual é o melhor?”*.

no free lunch theorem: infelizmente, não existe um único melhor modelo que funcione de maneira ideal para todos os tipos de problemas.

D. Wolpert. The lack of a priori distinctions between learning algorithms.
In: Neural Computation 8.7 (1996)

Um conjunto de pressupostos (também chamado de viés indutivo) que funciona bem em um domínio pode funcionar mal em outro.

All models are wrong, but some models are useful
George Box (Box and Draper, 1987, pg 424)

Algoritmos de Aprendizado Não Supervisionado

- Não existem saídas (output/label), apenas **entradas** (inputs/features).
- O objetivo é **encontrar padrões** “interessantes” nos dados.
 - análise de clusters
 - missing value imputation
 - análise de textos
 - redução de dimensionalidade
- **Medidas de similaridade**. O foco é o padrão ou a proximidade?
 - **Proximidade:** Distância euclidiana
 - **Padrão:** Correlação

Exemplo: Clustering

Cybis, G. B., Valk, M. and Lopes, S. R. C. (2018). Clustering and classification problems in genetics through U -statistics. Journal of Statistical Computation and Simulation, 88(10), 1882–1902.

- Método de *clustering* baseado em U -statistics. (2 grupos)
- Teste estatístico para avaliar a homogeneidade de grupos.
- Algoritmo baseado em dissimilaridades intra e intergrupos.
- Teste para verificar a significância da classificação de uma amostra em um de dois grupos.
- **Aplicações práticas:** três conjuntos de dados genéticos: (a) Diversidade genética humana global; (b) Expressão gênica em tumores de mama; (c) Sorotipos do vírus da dengue.
- **Cenário de aplicação:** Alta dimensão e baixo tamanho amostral.

Exemplo: Clustering

Valk, M. and Cybis, G. B. (2021). *U-Statistical Inference for Hierarchical Clustering*. *Journal of Computational and Graphical Statistics*, 30(1), 133–143.

- Clustering hierárquico baseado em *U-statistics* que avalia a significância estatística dos agrupamentos.
- Especificamente adaptado para cenários de alta dimensão e baixo tamanho amostral.
- Permite a criação de grupos de tamanho 1.
- **Aplicação prática:** reconhecimento de imagens.

Exemplo: Clustering

Bello, D. Z., Valk, M. and Cybis, G. B. (2024). Towards U-statistics clustering inference for multiple groups. Journal of Statistical Computation and Simulation, 94(1), 204–222.

- Método de *clustering* baseado em *U-statistics*. (3 grupos)
- Avaliação da significância estatística dos agrupamentos.
- Permite a criação de grupos de tamanho 1.
- **Aplicação prática:** reconhecimento de imagens.

Exemplo: Clustering

Cybis, G. B., Valk, M. and Lopes, S. R. C. (2018). Clustering and classification problems in genetics through U -statistics. Journal of Statistical Computation and Simulation, 88(10), 1882–1902.

Valk, M. and Cybis, G. B. (2021). U -Statistical Inference for Hierarchical Clustering. Journal of Computational and Graphical Statistics, 30(1), 133–143.

Bello, D. Z., Valk, M. and Cybis, G. B. (2024). Towards U -statistics clustering inference for multiple groups. Journal of Statistical Computation and Simulation, 94(1), 204–222.

Cybis, G. B. and Valk, M. (2021). uclust: Clustering and Classification Inference with U -Statistics. R package version 1.0.0.

<https://CRAN.R-project.org/package=uclust>

Exemplo: Clustering

Cybis, G. B., Valk, M. and Lopes, S. R. C. (2018). Clustering and classification problems in genetics through U -statistics. Journal of Statistical Computation and Simulation, 88(10), 1882–1902.

Valk, M. and Cybis, G. B. (2021). U -Statistical Inference for Hierarchical Clustering. Journal of Computational and Graphical Statistics, 30(1), 133–143.

Bello, D. Z., Valk, M. and Cybis, G. B. (2024). Towards U -statistics clustering inference for multiple groups. Journal of Statistical Computation and Simulation, 94(1), 204–222.

Cybis, G. B. and Valk, M. (2021). uclust: Clustering and Classification Inference with U -Statistics. R package version 1.0.0.

<https://CRAN.R-project.org/package=uclust>

Medida de similaridade: foco em **proximidade**

Alternativa para séries temporais: substituir $\mathbf{X} = (X_1, \dots, X_n)$ (vetor da séries temporal) por $\mathbf{X} = (\rho_1, \dots, \rho_h)$ (autocovariâncias) ou $\mathbf{X} = (I_{\lambda_1}, \dots, I_{\lambda_j})$ (periodograma)

Algoritmos de Aprendizado Supervisionado

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 .

Tom Mitchell. Machine Learning. McGraw Hill, 1997.

- $\mathbf{X} \in \mathcal{X} \subset \mathbb{R}^p$ (inputs) e $Y \in \mathcal{Y} \subset \mathbb{R}$ (output)
 $inputs = \text{features/covariáveis/preditores/variáveis independentes}$
 $outputs = \text{labels/target/variável dependente}$
- Objetivo: aprender (estimar) $f : \mathcal{X} \rightarrow \mathcal{Y}$, uma função que retorna um valor $Y \in \mathcal{Y}$, dado um valor $\mathbf{X} \in \mathcal{X}$. (*Tarefa T*)
- Dados de **treinamento**: $\mathcal{D}_{train} = \{\mathbf{X}_i, Y_i\}_{i=1}^n$ (*Experiência E*)
Dados de **teste**: $\mathcal{D}_{test} = \{\mathbf{X}_i, Y_i\}_{i=n+1}^m$ (*teste vs validação*)
- A *medida de desempenho* P depende do tipo de saída que estamos prevendo.

Modelagem tradicional versus ML: modelo AR(2)

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \varepsilon_t, \quad \varepsilon_t \sim RB(0, \sigma^2)$$

- Modelagem tradicional: $\{Y_t\}_{t=1}^N$.
 $\{Y_t\}_{t=1}^n$: utilizado para ajustar o modelo.
 $\{Y_t\}_{t=n+1}^N$: utilizado para avaliar previsões fora da amostra.
- Modelos de machine learning: $\{\mathbf{Y}_t, \mathbf{X}_t\}_{t=3}^M$,
 $\mathbf{Y}_t = (Y_t, \dots, Y_{t+h})$ e $\mathbf{X}_t = (Y_{t-1}, Y_{t-2})$.
 $T \subset \{1, \dots, M\}$: subconjunto de índices de tamanho $n < N$.
 $\{\mathbf{Y}_t, \mathbf{X}_t\}_{t \in T}$: conjunto de treino.
 $\{\mathbf{Y}_t, \mathbf{X}_t\}_{t \in T^c}$: conjunto de teste.

Perspectiva probabilística

- Mecanismo gerador dos dados (treinamento e teste):

$$\begin{aligned}\mathbf{X} &\sim F_{\mathbf{X}}, \quad F_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} \leq \mathbf{x}) \\ Y|\mathbf{X} &\sim F_{Y|\mathbf{X}}, \quad F_{Y|\mathbf{X}}(y|\mathbf{x}) = P(Y \leq y|\mathbf{X} = \mathbf{x})\end{aligned}$$

- $\{\mathbf{X}_i, Y_i\}_{i=1}^n$ é uma amostra i.i.d. obtida de $F_{\mathbf{X},Y}$

$$(\mathbf{X}, Y) \sim F_{\mathbf{X},Y}, \quad F_{\mathbf{X},Y}(\mathbf{x}, y) = P(\mathbf{X} \leq \mathbf{x}, Y \leq y)$$

onde $F_{\mathbf{X},Y}(\mathbf{x}, y)$ satisfaz

$$F_{\mathbf{X},Y}(\mathbf{x}, y) = \int_{A_{\mathbf{x}}} F_{Y|\mathbf{X}}(y|z) dF_{\mathbf{X}}(z),$$

$\mathbf{x} = (x_1, \dots, x_p)' \in \mathbb{R}^p, y \in \mathbb{R}$ e

$$A_{\mathbf{x}} = \{\mathbf{z} \in \mathbb{R}^p : z_i \leq x_i, i = 1, \dots, p\}$$

- **Função perda:** uma função $L : \mathcal{Y} \times \mathbb{R} \rightarrow [0, \infty)$.

O valor $L(y, f(\mathbf{x}))$ mede o erro que cometemos ao usar $f(\mathbf{x})$ para estimar/prever y .

👉 **Regressão:** Perdas L_1 (perda absoluta) e L_2 (perda quadrática)

$$L_1(y, f(\mathbf{x})) = |y - f(\mathbf{x})| \quad \text{e} \quad L_2(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$$

👉 **Classificação:** Função perda 0-1

$$L(y, f) = \begin{cases} 0, & y = f(\mathbf{x}) \\ 1, & y \neq f(\mathbf{x}) \end{cases}$$

Objetivo: queremos que a perda $L(Y_i, f(\mathbf{X}_i))$ seja pequena para muitos pares $(\mathbf{X}_i, Y_i)$ na amostra de teste.

- **Risco** (ou perda esperada) associado à f :

$$R_{L,F}(f) := \mathbb{E}(L(Y, f(\mathbf{X}))) = \int_{\mathcal{X} \times \mathcal{Y}} L(y, f(\mathbf{x})) dF_{\mathbf{X},Y}(\mathbf{x}, y).$$

Objetivo: encontrar a função $\hat{f}$, dentre todas as funções de uma dada classe $\mathfrak{F}$, para a qual o risco $R_{L,F}(f)$ é mínimo, isto é,

$$\hat{f} = \operatorname{argmin}_{f \in \mathfrak{F}} \{R_{L,F}(f)\}$$

- **Risco empírico**

$$R_{\text{emp}}(f) := \frac{1}{n} \sum_{i=1}^n L(Y_i, f(\mathbf{X}_i)) \xrightarrow[n \rightarrow \infty]{P} R_{L,F}(f). \quad (\text{LLN})$$

Real objetivo: minimizar a perda para os pontos de teste

$$\hat{f} = \operatorname{argmin}_{f \in \mathfrak{F}} \left\{ \frac{1}{m-n} \sum_{i=n+1}^m L(Y_i, f(\mathbf{X}_i)) \right\}$$

O problema é que geralmente não conhecemos $\{Y_i\}_{i=n+1}^m$ de antemão, mas

$$\frac{1}{m-n} \sum_{i=n+1}^m L(Y_i, f(\mathbf{X}_i)) \xrightarrow[m \rightarrow \infty]{P} R_{L,F}(f).$$

E é por isso que minimizamos o risco empírico!

Erro de treinamento e de teste

- $f_{\mathcal{D}_{train}} : \mathcal{X} \rightarrow \mathcal{Y}$ um modelo preditivo obtido a partir de um conjunto de treinamento $\mathcal{D}_{train} = \{(\mathbf{X}_i, Y_i)\}_{i=1}^n$.
- $L : \mathcal{Y} \times \mathbb{R} \rightarrow [0, \infty)$ a função perda.
- **erro de teste** (*erro de generalização*): o erro de previsão para uma amostra de teste independente

$$E_{L,F|\mathcal{D}_{train}}(f_{\mathcal{D}_{train}}) := \mathbb{E}(L(Y, f_{\mathcal{D}_{train}}(\mathbf{X})) \mid \mathcal{D}_{train}).$$

Erro de treinamento e de teste

- $f_{\mathcal{D}_{train}} : \mathcal{X} \rightarrow \mathcal{Y}$ um modelo preditivo obtido a partir de um conjunto de treinamento $\mathcal{D}_{train} = \{(\mathbf{X}_i, Y_i)\}_{i=1}^n$.
- $L : \mathcal{Y} \times \mathbb{R} \rightarrow [0, \infty)$ a função perda.
- **erro de teste** (*erro de generalização*): o erro de previsão para uma amostra de teste independente

$$E_{L,F|\mathcal{D}_{train}}(f_{\mathcal{D}_{train}}) := \mathbb{E}(L(Y, f_{\mathcal{D}_{train}}(\mathbf{X})) \mid \mathcal{D}_{train}).$$

Nesse contexto, o conjunto de treinamento $\mathcal{D}_{train}$ é fixo, e o erro de teste se refere-se a esse conjunto de treinamento específico.

$E_{L,F|\mathcal{D}_{train}}$ representa a média da função perda, condicionada à escolha do modelo feita com o conjunto de treinamento $\mathcal{D}_{train}$.

Erro de treinamento e de teste

- $f_{\mathcal{D}_{train}} : \mathcal{X} \rightarrow \mathcal{Y}$ um modelo preditivo obtido a partir de um conjunto de treinamento $\mathcal{D}_{train} = \{(\mathbf{X}_i, Y_i)\}_{i=1}^n$.
- $L : \mathcal{Y} \times \mathbb{R} \rightarrow [0, \infty)$ a função perda.
- **erro de teste** (*erro de generalização*): o erro de previsão para uma amostra de teste independente

$$E_{L,F|\mathcal{D}_{train}}(f_{\mathcal{D}_{train}}) := \mathbb{E}(L(Y, f_{\mathcal{D}_{train}}(\mathbf{X})) \mid \mathcal{D}_{train}).$$

- O **erro de teste esperado** (ou *erro de predição esperado*) é dado por

$$E_{L,F} := \mathbb{E}(L(Y, f_{\mathcal{D}_{train}}(\mathbf{X}))) = \mathbb{E}(E_{L,F|\mathcal{D}_{train}}(f_{\mathcal{D}_{train}}))$$

- O **erro de treinamento** é a perda média, calculada sobre os pontos do conjunto de treinamento $\mathcal{D}_{train}$, isto é,

$$E_{\mathcal{D}_{train}} = \frac{1}{n} \sum_{i=1}^n L(Y_i, f_{\mathcal{D}_{train}}(\mathbf{X}_i)).$$

$E_{\mathcal{D}_{train}}$ é uma visão otimista do erro que modelo tem para fazer predição.

$E_{\mathcal{D}_{train}}$ é uma medida de pouco interesse pois o foco é o poder de generalização do modelo.

Pergunta que surge naturalmente: como estimar o erro de teste?

- O **erro de treinamento** é a perda média, calculada sobre os pontos do conjunto de treinamento $\mathcal{D}_{train}$, isto é,

$$E_{\mathcal{D}_{train}} = \frac{1}{n} \sum_{i=1}^n L(Y_i, f_{\mathcal{D}_{train}}(\mathbf{X}_i)).$$

$E_{\mathcal{D}_{train}}$ é uma visão otimista do erro que modelo tem para fazer predição.

$E_{\mathcal{D}_{train}}$ é uma medida de pouco interesse pois o foco é o poder de generalização do modelo.

Pergunta que surge naturalmente: como estimar o erro de teste?

Data splitting

Data Splitting

Conjunto de treino:

é usado para treinar o modelo.

Principal finalidade: obter uma estimativa $f_{\mathcal{D}_{train}}$ para a função f que associa $\mathbf{X}$ a Y .

Conjunto de teste:

é usado para avaliar o desempenho do modelo em dados que não foram usados durante o treinamento.

Principal finalidade: Permite verificar se o modelo final selecionado fornece boas generalizações

**Conjunto de validação:**

é usado para estimar o erro de predição dos modelos.

Principal finalidade: auxiliar na seleção e ajuste de hiperparâmetros do modelo, bem como na avaliação do desempenho dos modelos com o objetivo de selecionar o melhor dentre os possíveis candidatos.

Estimando o erro de teste:

- $\mathcal{D}_{train} = \{(\mathbf{X}_i, Y_i)\}_{i=1}^n$ e $\mathcal{D}_{test} = \{(\mathbf{X}_i, Y_i)\}_{i=n+1}^m$ (independentes) conjuntos de treinamento e teste, respectivamente.
- $\mathcal{D}_{train} = \{(\mathbf{X}_i, Y_i)\}_{i=1}^n$ para ajustar o modelo
- $\mathcal{D}_{test} = \{(\mathbf{X}_i, Y_i)\}_{i=n+1}^m$ para obter uma estimativa para o erro de teste

$$\hat{E}_{L,F|\mathcal{D}_{train}}(f_{\mathcal{D}_{train}}) = \frac{1}{m-n} \sum_{i=n+1}^m L(Y_i, f_{\mathcal{D}_{train}}(\mathbf{X}_i)).$$

Intuitivamente, essa é uma boa estimativa para o erro de teste uma vez que estamos considerando um conjunto independente para avaliar o poder preditivo do modelo.

Métodos de Reamostragem

Técnicas utilizadas para extrair repetidamente amostras de um conjunto conjunto de treino.

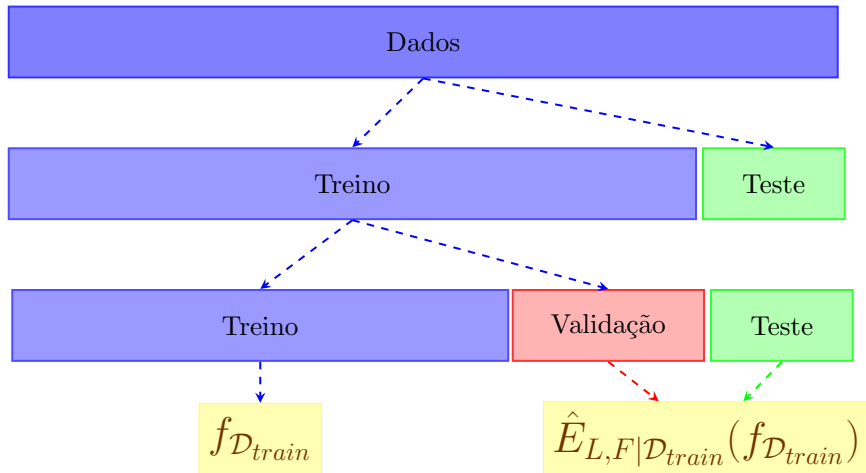
Objetivo: estimar propriedades dos dados ou avaliar o desempenho de modelos estatísticos.

Exemplos: validação cruzada (*cross-validation*) e *bootstrap*

- são amplamente utilizadas na avaliação e seleção de modelos
- têm objetivos diferentes
 - (a) validação cruzada é usada principalmente para avaliar o **desempenho preditivo** de um modelo,
 - (b) bootstrap é mais útil para **estimar a incerteza** associada a estimadores e métricas.

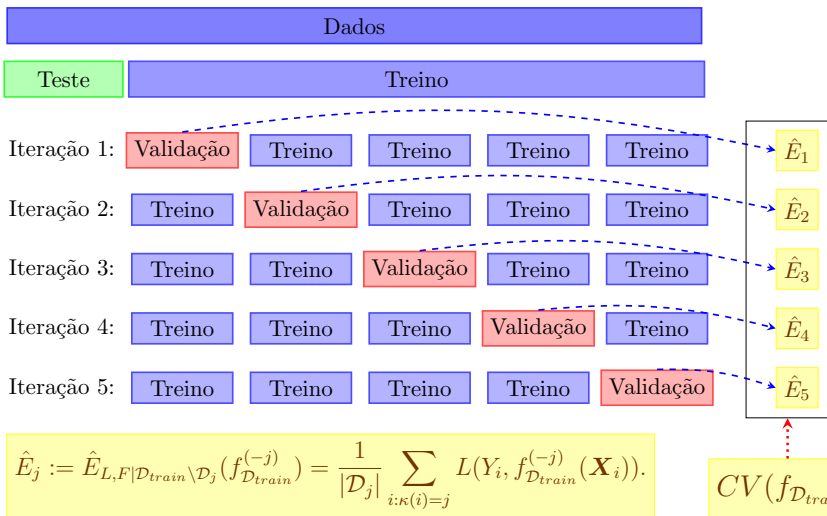
Validação cruzada

Método *holdout*

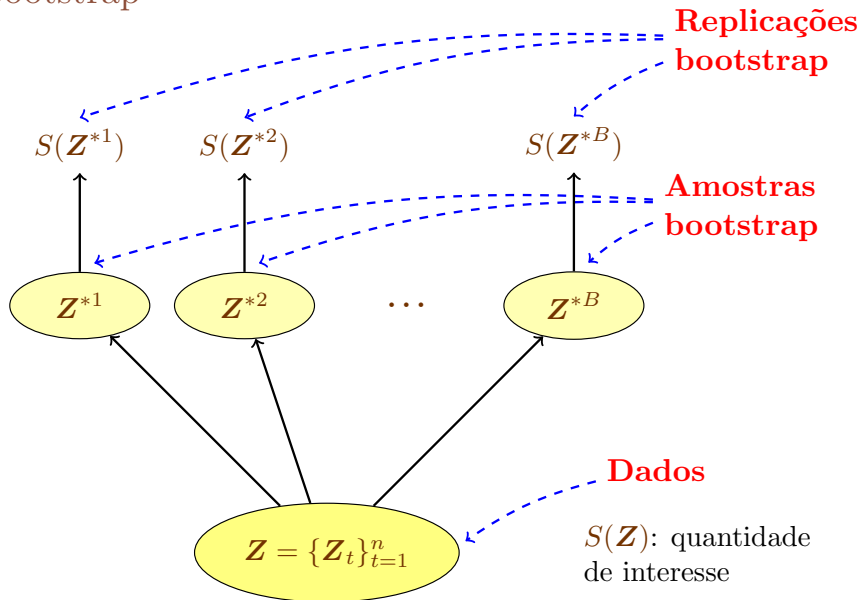


Método k -fold.

Caso particular: *leave-one-out* (k = tamanho da amostra).
 Similares : *leave-p-out* (todas as partições possíveis) e validação cruzada de Monte Carlo (seleciona amostras aleatórias).



Bootstrap



- Melhorar previsões via **bagging** (*bootstrap aggregating*)

$$\hat{f}_{ave}(\mathbf{X}) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(\mathbf{X}).$$

- Analisar o comportamento de um parâmetro (exemplo, variância)

$$\hat{\sigma}_{S(\mathbf{Z})}^2 = \frac{1}{B-1} \sum_{b=1}^B (S(\mathbf{Z}^{*b}) - \hat{\mu}_{S(\mathbf{Z})})^2, \quad \hat{\mu}_{S(\mathbf{Z})} = \frac{1}{B} \sum_{b=1}^B S(\mathbf{Z}^{*b}).$$

- Estimar o erro de teste esperado

$$\hat{E}_{L,F}^{boot} = \frac{1}{B} \sum_{b=1}^B \hat{E}_{L,F|\mathbf{Z}^{*b}}^{boot}(\hat{f}^{*b}) = \frac{1}{B} \frac{1}{n} \sum_{b=1}^B \sum_{i=1}^n L(Y_i, \hat{f}^{*b}(\mathbf{X}_i))$$

Ponto negativo: intersecção entre amostra de treino e teste.

Métodos de Ensemble

- Procedimentos que combinam saídas de múltiplos algoritmos de aprendizado $\{f_t\}_{t \in T}$, onde $f_t : \mathcal{X} \rightarrow \mathcal{Y}$
- **Premissa:** ao combinar adequadamente as previsões individuais, é possível obter previsões globalmente superiores, em média, em comparação com cada algoritmo atuando de forma independente.
- **Ampla aplicabilidade:** Os preditores a serem combinados podem ter como saída valores reais, rótulos de classe, probabilidades posteriores, classificações, agrupamentos ou qualquer outra quantidade.

Métodos de Ensemble

- Procedimentos que combinam saídas de múltiplos algoritmos de aprendizado $\{f_t\}_{t \in T}$, onde $f_t : \mathcal{X} \rightarrow \mathcal{Y}$
- **Premissa:** ao combinar adequadamente as previsões individuais, é possível obter previsões globalmente superiores, em média, em comparação com cada algoritmo atuando de forma independente.
- **Ampla aplicabilidade:**
Combinador linear: para problemas de regressão

$$f(\mathbf{X}) = \sum_{t=1}^T w_t f_t(\mathbf{X}).$$

Métodos de Ensemble

- Procedimentos que combinam saídas de múltiplos algoritmos de aprendizado $\{f_t\}_{t \in T}$, onde $f_t : \mathcal{X} \rightarrow \mathcal{Y}$
- **Premissa:** ao combinar adequadamente as previsões individuais, é possível obter previsões globalmente superiores, em média, em comparação com cada algoritmo atuando de forma independente.
- **Ampla aplicabilidade:**
Combinador de votos: para problemas de classificação.
Para um problema de duas classes

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{t=1}^T w_t f_t(\mathbf{X})\right), \quad f_t(\mathbf{X}) \in \{-1, 1\}.$$

Métodos de Ensemble

- Procedimentos que combinam saídas de múltiplos algoritmos de aprendizado $\{f_t\}_{t \in T}$, onde $f_t : \mathcal{X} \rightarrow \mathcal{Y}$
- **Premissa:** ao combinar adequadamente as previsões individuais, é possível obter previsões globalmente superiores, em média, em comparação com cada algoritmo atuando de forma independente.
- **Ampla aplicabilidade:**
Combinador produto: combinação de probabilidades

$$f(\mathbf{X}) = \frac{1}{\tau} \prod_{t=1}^T (f_t(\mathbf{X}))^{w_t}$$

Métodos de Ensemble

- Procedimentos que combinam saídas de múltiplos algoritmos de aprendizado $\{f_t\}_{t \in T}$, onde $f_t : \mathcal{X} \rightarrow \mathcal{Y}$
- **Premissa:** ao combinar adequadamente as previsões individuais, é possível obter previsões globalmente superiores, em média, em comparação com cada algoritmo atuando de forma independente.
- **Ampla aplicabilidade:** Os preditores a serem combinados podem ter como saída valores reais, rótulos de classe, probabilidades posteriores, classificações, agrupamentos ou qualquer outra quantidade.

Prós, Contrás e Trade-offs

Desvantagens (Contrás)

- Maior custo computacional: requer mais poder de processamento
- Consumo elevado de recursos: CPU, memória e tempo
- Complexidade de implementação: é necessário treinar e avaliar múltiplos modelos
- Mais requisitos de armazenamento

Prós, Contras e Trade-offs

Vantagens (Prós)

- Compensação de algoritmos fracos: combina modelos simples para obter resultados robustos.
- Robustez através da diversidade
- Melhor eficiência marginal: Recursos adicionais aplicados em ensemble trazem **maior ganho** de precisão versus aplicar recursos extras em um único modelo.
- Precisão superior: Resultados globalmente melhores em média.

Desvantagens

- Maior custo computacional
- Consumo elevado de recursos
- Complexidade de implementação
- Mais requisitos de armazenamento

Vantagens

- Compensação de algoritmos fracos
- Robustez através da diversidade
- Melhor eficiência marginal
- Precisão superior

Ensemble	Modelo Único
Múltiplos modelos simples	Modelo complexo único
Paralelizável	Sequencial
Melhor ROI computacional	Rendimentos decrescentes
Robustez via diversidade	Risco de overfitting
+ Armazenamento	- overhead de armazenamento

Algoritmos

- **Característica desejável:** diversidade (leava à previsões mais robustas e precisas).
- Algoritmos podem promover diversidade de maneira **implícita** ou **explícita**.
- Exemplos: *bagging*, *random subspace*, *rotation forests* e *boosting*.

Algoritmos

- **Característica desejável:** diversidade (leava à previsões mais robustas e precisas).
- Algoritmos podem promover diversidade de maneira **implícita** ou **explícita**.

Maneria implícita

- Introdução de aleatoriedade no processo de treinamento.
- Fornecem diferentes subconjuntos aleatórios dos dados de treinamento para cada modelo base.
- Exemplos: *bagging*, *random subspace* ou *rotation forests*.
- Exemplos: *bagging*, *random subspace*, *rotation forests* e *boosting*.

Algoritmos

- **Característica desejável:** diversidade (leava à previsões mais robustas e precisas).
- Algoritmos podem promover diversidade de maneira **implícita** ou **explícita**.

Maneira explícita

- medidas são tomadas para garantir que os modelos base sejam diferentes entre si.
- Exemplo: *boosting*

Adaboost, Gradient boosting e XGBoost (projetado para velocidade e escala computacional)

- Exemplos: *bagging, random subspace, rotation forests e boosting*.

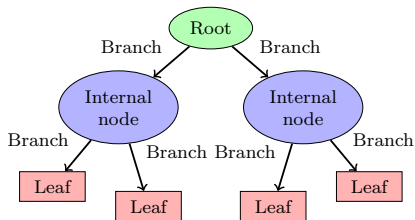
Exemplo: LASSO + árvores de decisão

Gandini, A.L.D., Ziegelmann, F.A. Combining LASSO-type Methods with a Smooth Transition Random Forest. Annals of Data Science 12, 899–928 (2025).

- Método híbrido para a estimação de modelos de regressão: combina métodos do tipo LASSO e florestas aleatórias.
- A abordagem envolve duas etapas: (1) ajuste de uma regressão linear penalizada usando métodos do tipo LASSO. (2) treinamento de uma floresta aleatória utilizando STR-trees (*Smooth Transition Regression Trees*) sobre os resíduos da primeira etapa, usando as covariáveis originais.
- Aplicação à conjuntos de dados reais de diversas áreas, variando quanto ao tamanho de amostra e número de covariáveis.

Árvores de Decisão – Introdução

- **Intuition:** Hierarchical, tree-structured model for decision making. Mimics human decision process with sequential questions.



- **How it works:**
Recursively partitions the features space.
Each internal node is a decision rule on one feature.
Each leaf node represents a prediction.

- **Key characteristics:**
Non-parametric. Interpretable.
Can handle both numerical and categorical data.

CART (Breiman et al., 1984)

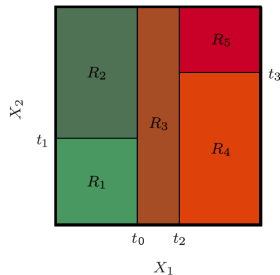
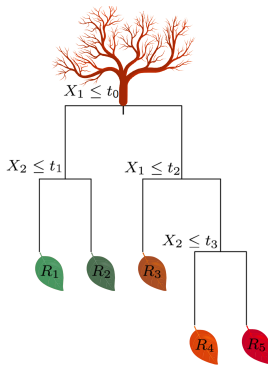
Breiman, L., Friedman, J., Stone, C.J., Olshen, R. (1984). Classification and Regression Trees. Chapman and Hall/CRC.

Recursive splits to minimize prediction error.

Limitation:

Piecewise-constant predictions.

$$f(\mathbf{X}) = \sum_{m=1}^M \gamma_m I(\mathbf{X} \in \mathcal{R}_m)$$



Probabilistic methods main idea

Assuming that $\mathbf{X} \in \mathbb{R}^p$ and there exist only the root node a single binary partition, the prediction function is given by

$$f(\mathbf{X}) = \gamma_1 I(X_j \leq t) + \gamma_2 [1 - I(X_j \leq t)]$$

for some $j \in \{1, \dots, p\}$ and $t \in \mathbb{R}$.

Probabilistic methods main idea

Assuming that $\mathbf{X} \in \mathbb{R}^p$ and there exist only the root node a single binary partition, the prediction function is given by

$$f(\mathbf{X}) = \gamma_1 G(\mathbf{X}; j, t, \alpha) + \gamma_2 [1 - G(\mathbf{X}; j, t, \alpha)].$$

for some $j \in \{1, \dots, p\}$ and $t \in \mathbb{R}$ and $\alpha \in \mathbb{R}$.

STR Trees (da Rosa et. al, 2008)

da Rosa, J.C, Veiga, A., Medeiros, M. (2008). *Tree-structured smooth transition regression models. Computational Statistics & Data Analysis* 52, 2469-2488.

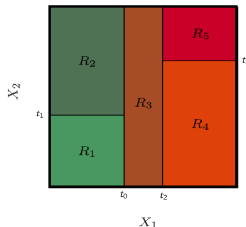
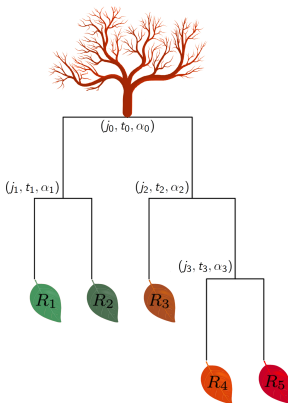
Combines **CART** + **Smooth Transition Regression (STR)**.

Uses **logistic function** for smooth splits.

Benefits:

Enables **standard inferential methods**.

Lagrange Multiplier test for split significance.



$$G(\mathbf{X}; j, t, \alpha) = \frac{1}{1 + e^{-\alpha(X_j - t)}}$$

Soft Trees (İrsoy et al., 2012)

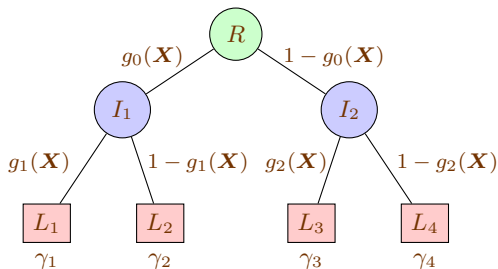
İrsoy, O., Yıldız, O.T., Alpaydın, E. (2012). *Soft decision trees*, in: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 1819-1822.

Soft decisions via sigmoid functions

Advantages:

Continuous response
(smoother predictions)

Enables **oblique splits**
(linear combinations of
covariates)



$$g_N(\mathbf{X}) = \frac{1}{1 + e^{-(\boldsymbol{\omega}'_N \mathbf{X} + \omega_{N0})}}$$

Predictions are given by

$$f(\mathbf{X}) = \sum_{m=1}^M \gamma_m \prod_{\substack{N \in \mathcal{N}_m, \\ N \notin \mathcal{L}}} p_N(\mathbf{X}) = \sum_{m=1}^M \gamma_m P(\mathcal{R}_m | \mathbf{X}),$$

where:

- $\mathcal{L}$ is the set of all leafs,
- $\mathcal{N}_m$ is the set of all nodes in the path from the root to the m th leaf,
- γ_m is the prediction value corresponding to the m th leaf,

Predictions are given by

$$f(\mathbf{X}) = \sum_{m=1}^M \gamma_m \prod_{\substack{N \in \mathcal{N}_m, \\ N \notin \mathcal{L}}} p_N(\mathbf{X}) = \sum_{m=1}^M \gamma_m P(\mathcal{R}_m | \mathbf{X}),$$

where:

- $\mathcal{L}$ is the set of all leafs,
- $\mathcal{N}_m$ is the set of all nodes in the path from the root to the m th leaf,
- γ_m is the prediction value corresponding to the m th leaf,
- **STR Trees**

$$p_N(\mathbf{X}) = \begin{cases} g_N(\omega'_N \mathbf{X} + \omega_{N0}), & \text{if branching left at } N, \\ 1 - g_N(\omega'_m \mathbf{X} + \omega_{N0}), & \text{if branching right at } N, \end{cases}$$

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

Predictions are given by

$$f(\mathbf{X}) = \sum_{m=1}^M \gamma_m \prod_{\substack{N \in \mathcal{N}_m, \\ N \notin \mathcal{L}}} p_N(\mathbf{X}) = \sum_{m=1}^M \gamma_m P(\mathcal{R}_m | \mathbf{X}),$$

where:

- $\mathcal{L}$ is the set of all leafs,
- $\mathcal{N}_m$ is the set of all nodes in the path from the root to the m th leaf,
- γ_m is the prediction value corresponding to the m th leaf,
- **Soft Trees**

$$p_N(\mathbf{X}) = \begin{cases} G_N(\mathbf{X}, j_N, t_N, \alpha_N), & \text{if branching left at } N, \\ 1 - G_N(\mathbf{X}, j_N, t_N, \alpha_N), & \text{if branching right at } N, \end{cases}$$

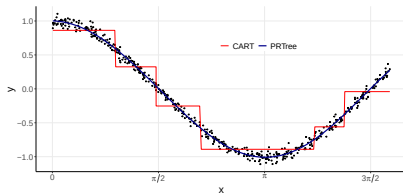
$$G_N(\mathbf{X}) = \frac{1}{1 + e^{-\alpha_N(X_{j_N} - t_N)}}.$$

Probabilistic Regression Trees (Alkhoury et al., 2020)

Alkhoury, S., Devijver, E., Clausel, M., Tami, M., Gaussier, E., Oppenheim, G. (2020). Smooth and consistent probabilistic regression trees, in: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (Eds.), *Advances in Neural Information Processing Systems*, Curran Associates, Inc.. pp. 11345–11355. URL: <https://dl.acm.org/doi/pdf/10.5555/3495724.3496676>.

Probabilistic generalization of decision trees

- Adapt to the smoothness of the prediction function while preserving interpretability.
- Observations are associated to all regions through a probability distribution.



Key theoretical property

The PRTree learned from a training set of size n , denoted $f_{\text{PR}}^{(n)}$, is **consistent**, that is

$$\lim_{n \rightarrow \infty} \mathbb{E} \left(\left| f_{\text{PR}}^{(n)}(\mathbf{X}) - \mathbb{E}(Y|\mathbf{X}) \right|^2 \right) = 0.$$

PRTree replace the indicator function with a smooth function Ψ

$$f_{PR}(\mathbf{X}; \Theta) = \sum_{m=1}^M \gamma_m \Psi(\mathbf{X}; \mathcal{R}_m, \boldsymbol{\sigma}),$$

where $\Theta = (\{\mathcal{R}_m\}_{m=1}^M, \boldsymbol{\gamma}, \boldsymbol{\sigma})$ is the set of model parameters,

$$\Psi(\mathbf{X}; \mathcal{R}_m, \boldsymbol{\sigma}) = \frac{1}{\prod_{i=1}^p \sigma_i} \int_{\mathcal{R}_m} \phi\left(\frac{v_1 - X_1}{\sigma_1}, \dots, \frac{v_p - X_p}{\sigma_p}\right) d\mathbf{v},$$

where $\phi : \mathbb{R}^p \rightarrow \mathbb{R}$ is a *sufficiently regular* probability density function.

Parameter Estimation

- Training set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i)\}_{1 \leq i \leq n}$, with $\mathbf{X}_i \in \mathbb{R}^p$ and $Y_i \in \mathbb{R}$.
- Estimation procedure: based on empirical risk minimization principle with a quadratic loss:

$$\hat{\Theta} = \underset{\Theta \in \Xi}{\operatorname{argmin}} \left\{ \sum_{i=1}^n \left(Y_i - \sum_{m=1}^M \gamma_m P_{im} \right)^2 \right\}, \quad \text{with } P_{im} := \Psi(\mathbf{X}_i; R_m, \boldsymbol{\sigma}),$$

where

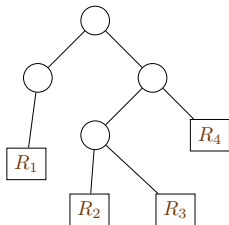
$$\Xi = \left\{ \left(\{R_m\}_{m=1}^M, \boldsymbol{\gamma}, \boldsymbol{\sigma} \right) : M \in \mathbb{N} \setminus \{0\}, R_m \subseteq \mathbb{R}^p, \boldsymbol{\gamma} \in \mathbb{R}^M, \boldsymbol{\sigma} \in \mathbb{R}_+^p \right\}.$$

- The $n \times M$ matrix P encodes the relations between each training example $\mathbf{X}_i$ and each region R_m . Also, $0 \leq P_{im} \leq 1$ and $\sum_{m=1}^M P_{im} = 1$.

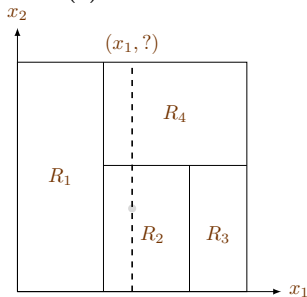
Adapting PRTree to Missing Covariates

(Prass et al., 2026): three strategies (`fill_type` in the PRTree package) – all reduce to the original PRTree when nothing is missing.

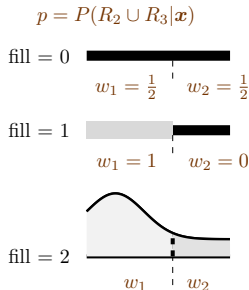
(a) Tree structure



(b) Partition



(c) Redistribution of p



Related Products



“DFA and DCCA
estimation in the
presence of
missing data”
(Alisson’s master
thesis)



“Handling
Missing Data in
Probabilistic
Regression Trees”
(theory)



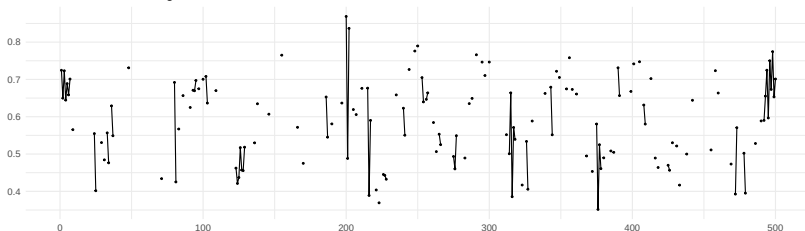
“PRTree: An R
Package for
Probabilistic
Regression Trees
with Built-in Missing
Data Handling”
(software)



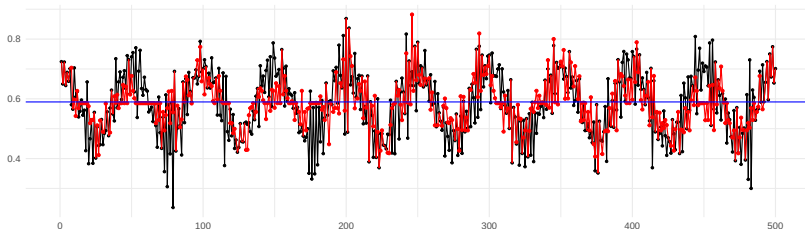
PRTree package
CRAN.R-project.org

Illustrative Example – Capturing Seasonality

Time series with missing values



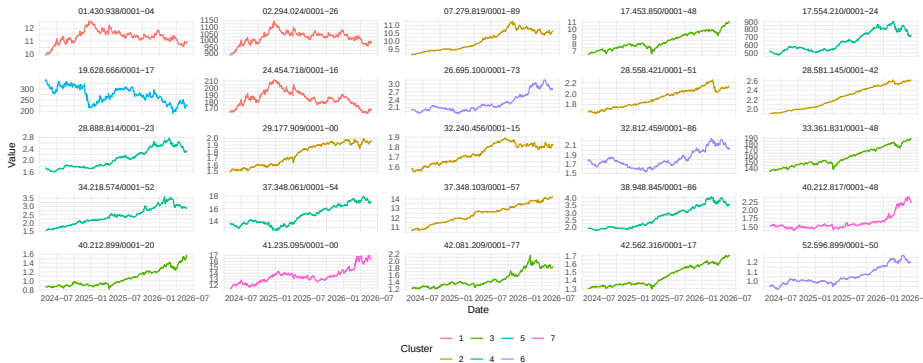
Complete vs imputed time series



BARMA(1,1), logit link, $X_t = \cos(2\pi t/50)$, $\alpha = 0.5$, $\beta = 0.5$, $\phi = -0.4$, $\theta = -0.6$, $\nu = 120$; 70% missing, $(h_1, h_2) = (5, 5)$ neighboring lags/leads as predictors – **black**: complete series, **red**: PRTree-imputed.

Real Data – Clustering time series using exogenous variables.

Caroline Oliveira Alonso's master's dissertation (in progress)



Neural Networks and Deep Learning in Time Series

Main categorization of NNs for time series forecasting:

- 1 Recurrent Neural Networks;
- 2 Convolutional Neural Networks;
- 3 Graph Neural Networks;
- 4 Transformers.

Neural Networks and Deep Learning in Time Series

Main categorization of NNs for time series forecasting:

- 1 Recurrent Neural Networks;
- 2 Convolutional Neural Networks;
- 3 Graph Neural Networks;
- 4 Transformers.

Neural Networks and Deep Learning in Time Series

Main categorization of NNs for time series forecasting:

- 1 Recurrent Neural Networks;
- 2 Convolutional Neural Networks;
- 3 Graph Neural Networks;
- 4 Transformers.

Neural Networks and Deep Learning in Time Series

Main categorization of NNs for time series forecasting:

- 1 Recurrent Neural Networks;
- 2 Convolutional Neural Networks;
- 3 Graph Neural Networks;
- 4 Transformers.

Neural Networks and Deep Learning in Time Series

Main categorization of NNs for time series forecasting:

- 1 Recurrent Neural Networks;
- 2 Convolutional Neural Networks;
- 3 Graph Neural Networks;
- 4 Transformers.

Neural Networks and Deep Learning in Time Series

Main categorization of NNs for time series forecasting:

- 1 Recurrent Neural Networks;
- 2 Convolutional Neural Networks;
- 3 Graph Neural Networks;
- 4 Transformers.

Neural Networks and Deep Learning in Time Series

Main categorization of NNs for time series forecasting:

- 1 Recurrent Neural Networks;
- 2 Convolutional Neural Networks;
- 3 Graph Neural Networks;
- 4 Transformers.

NEURAL NETWORKS

Neural Networks (NN)

A class of models whose name and structure are inspired by the functioning of the human brain, more specifically, by biological neurons.

Core Applications

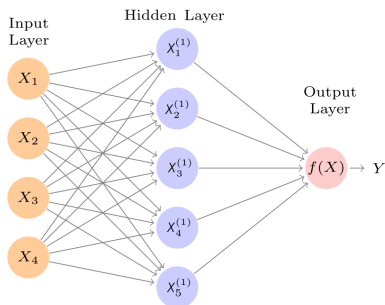
- Pattern recognition
- Classification tasks
- Regression analysis
- Data processing
- Prediction models

Advanced Capabilities

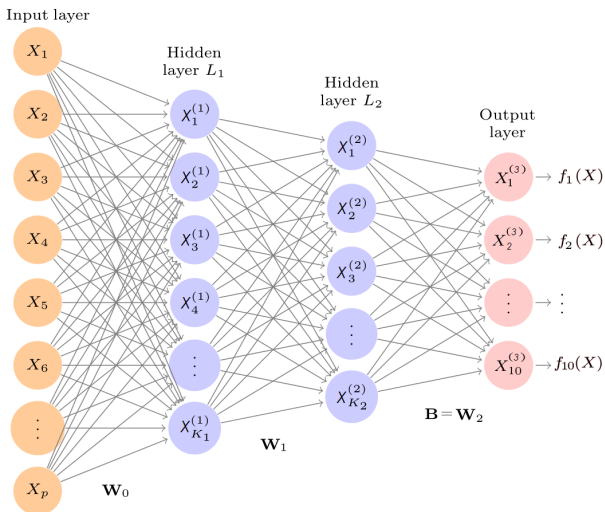
- Image recognition
- Speech processing
- Natural language
- Anomaly detection
- Time series analysis

Neural networks form the **core of deep learning algorithms**, serving as the fundamental building blocks for modern artificial intelligence systems.

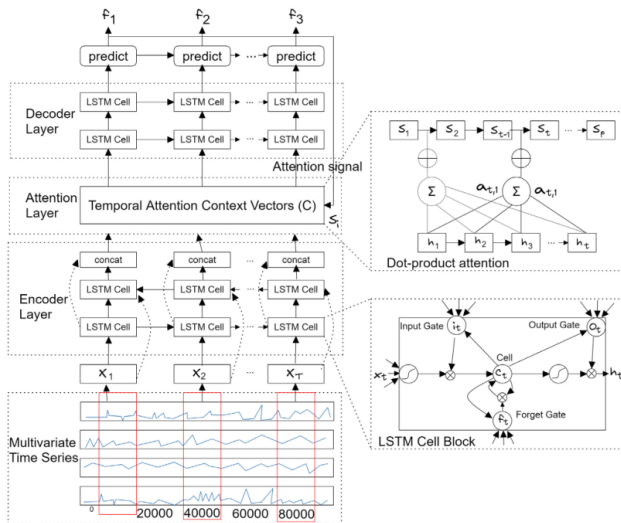
- In the context of supervised learning, a neural network receives a p -dimensional input vector $\mathbf{X} = (X_1, \dots, X_p)'$ and constructs a (typically non-linear) function $f(\mathbf{X})$ to predict the response Y .
- What distinguishes neural networks from other machine learning methods is the specific model structure:



Single Layer Neural Network



Dual Layer Neural Network



A complex deep learning architecture (model) for time series.
The MTSMFF architecture of Du et al. (2020)

- A neural network is composed of a collection of connected units or nodes
- Neurons are grouped into *layers*.
- Signals travel from the *input layer* (the first layer) to the *output layer* (the last layer),
- Between the input and output layers, the signal may pass through *hidden layers* (the intermediate layers) multiple times.
- Networks with many hidden layers are called deep neural networks.
- The connections between neurons of different layers are called *edges*.

The output of a neuron located in layer $l + 1$ is obtained by

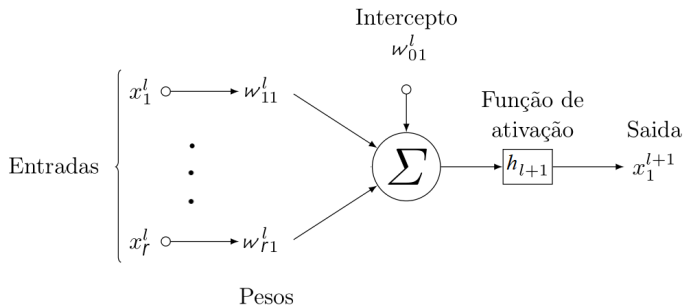
❶ **Linear combination of inputs:**

$$Z_j^{(l+1)} = \omega_{0j}^{(l)} + \sum_{k=1}^{p_l} \omega_{kj}^{(l)} X_k^{(l)}$$

where:

- p_l is the output length from layer l ,
- $X_i^{(l)}$ are the outputs (activations) from the previous layer l ,
- $w_{kj}^{(l)}$ are the weights connecting neuron k in layer l to neuron j in layer $l + 1$.
- $\omega_{0j}^{(l)}$ is the bias term of neuron j in layer $l + 1$.

❷ **Activation function:** $X_j^{(l+1)} = h_{l+1}(Z_j^{(l+1)})$ where $h_{l+1} : \mathbb{R} \rightarrow \mathbb{R}$ is the activation function (e.g., sigmoid, ReLU, **tanh**).



Typical neuron workflow

Example: single layer NN considering a quantitative response, using 4 predictors, X_1, X_2, X_3, X_4 , entering at the input layer. We consider $p_1 = 5$ hidden units.

$$\begin{aligned}
 Y = f(X_1, X_2, X_3, X_4) &= h_2 \left(\underbrace{\omega_{01}^{(1)} + \sum_{k=1}^5 \omega_{k1}^{(1)} X_k^{(1)}}_{X_1^{(2)}} \right) \\
 &= h_2 \left(\omega_{01}^{(1)} + \sum_{k=1}^5 \omega_{k1}^{(1)} \underbrace{h_1 \left(\omega_{0k}^{(0)} + \sum_{j=1}^4 \omega_{jk}^{(0)} X_j \right)}_{X_k^{(1)}} \right).
 \end{aligned}$$

h_1 and h_2 denote activation functions.

Backpropagation

- We need to determine the best possible weights in each layer of the neural network.
- The most commonly used method to achieve that is the so-called backpropagation, which is a gradient descent method.
- Let $\mathcal{W} = \{w_{kj}^{(m)}; 0 \leq k \leq p_m, 1 \leq j \leq p_{m+1}; 0 \leq m \leq M\}$ be the set of all weights to be determined.
- The idea is to obtain the set of weights that minimize some loss function, say $R(\mathcal{W})$, between the observed values, say Y_{ij} , and the value predicted, $f_j(\mathbf{X}_i)$, using the NN.

- Commonly used loss:
$$R(\mathcal{W}) = \sum_{t=1}^n \sum_{k=1}^K (Y_{tk} - f_k(\mathbf{X}_t))^2.$$

Backpropagation

The backpropagation method consists of updating the parameters from step r to step $r + 1$ using the following equations

$$\omega_{kj}^{(m)}(r+1) = \omega_{kj}^{(m)}(r) - \gamma_r \frac{\partial R(\mathcal{W})}{\partial \omega_{kj}^{(m)}(r)},$$

where $\gamma_r > 0$ is a hyperparameter that defines the learning rate.

- It can be shown that

$$\frac{\partial R(\mathcal{W})}{\partial \omega_{kj}^{(m)}(r)} = X_k^{(m-1)} \frac{\partial R(\mathcal{W})}{\partial Z_k^{(m)}}.$$

- Hence, to update the weight in the m -th layer, we need the output from the $(m - 1)$ th layer ($X_k^{(m-1)}$).
- This equation gives name to the method.

Important things to take into account

Tuning neural networks is an art which requires experience!

In general,

- the model is over-parameterized;
- the optimization problem is non-convex (depending on the choice of activation function) and unstable;
- optimization is extremely sensitive to scale;
- in most cases, initialization matters.
- results may vary wildly depending on the architecture.

Initialization

- Weights should not be initialized with zeroes, although the bias can be initialized with 0's without problem.
- Starting values for the weights: values close to 0, but not zero, are preferable;
- Large weights can cause activations and gradients to explode, leading to divergence or NaNs;
- Using random small values are a common choice.

Learning rate

- The learning rate is one of the most important hyperparameters in training neural networks.
- Balances speed of convergence vs. stability of training.
- γ_r too large: updates overshoot the minimum of the loss function, which may remain unstable causing the training to diverge or oscillate without converging.
- γ_r too small: training converges very slowly, increasing computation cost, and may get stuck in local minima or saddle points.
- Common starting points: 10^{-4} to 10^{-2} depending on architecture and optimizer.

Learning rate

- Learning rate schedules: Gradually reduce γ_r as training progresses (e.g., exponential decay, step decay).
- Adaptive learning rates: some optimization algorithms automatically adjust the learning rate for each parameter based on past gradients.
- Example: Adam (adaptive momentum estimator, stochastic gradient descent based) smooths updates by averaging past gradients and scales learning rates inversely with the square root of past gradient magnitudes.
- Cyclical learning rates: Vary the rate between lower and upper bounds to escape saddle points.

Local minima

- Backpropagation uses gradient descent to find (local) function minima;
- At the same time, the loss function is non-convex and may possess multiple local minima;
- Solution: testing different initial points, choosing the one with the smallest loss; Bagging.

Vanishing Gradient

- When training deep networks with backpropagation, the gradient of the loss is propagated backwards through many layers.
- Due to the chain rule, the update rule at layer m depends on the product of many terms.
- If the derivatives are less than 1 in magnitude (as with sigmoid or tanh) $\implies$ gradient shrinks exponentially with depth.
- Consequences: convergence of shallow layers are very slow.
- Network fails to capture long-range dependencies.
- Training becomes unstable and requires careful tuning

Vanishing Gradient

- Use ReLU or variates (leaky ReLU) - they avoid the problem.
- Careful initialization.
- Architectural improvements: LSTM/GRU in RNN uses gating mechanism to preserve the gradient over long sequences.
- Using adaptive optimizers (adjust the step size).
- Normalization Techniques (batch and layer).

Overfitting

- Since NNs have many parameters, the global minimum of the loss function usually causes overfitting.
- Overfitting in neural networks occurs when the model fits the training data too well, including noise, and performs poorly on new, unseen data.
- Mitigation: the implementation of an *early-stop rule* - the model training is stopped before the solution gets too close to the global minima.
- Alternatively, a penalization (ridge/LASSO-style) can be applied to the loss.

Scale

- The scale of $X_1, \dots, X_n$ affects the weights in the first hidden layer, which greatly influences the final result.
- Hence, standardization is generally recommended.
- Z-score standardization are the usual: subtract mean, divide by standard deviation.
- Min-max scaling can also be used: scale features to $[0, 1]$.

Pros:

- Faster convergence: gradient descent converges more quickly when inputs have similar scales;
- Prevents gradient issues: avoids exploding or vanishing gradients caused by very large or very small inputs;

Number of hidden layers

- It depends on the problem.
- It is not always true that the more layers, the better! This is a common misconception.
- The choice of the number of hidden layers is a crucial architectural decision that balances model capacity with practical constraints.
- The trend towards deep learning exists for good reasons, but models need to be carefully crafted.

Why more layers?

- Deep networks can learn a hierarchy of features.
 - Lower layers $\Rightarrow$ simple patterns: sharp spikes or dips, short term trends, noise filtering.
 - Middle layers $\Rightarrow$ combine lower ones to form more complex structures. They learn to recognize these longer, composite shapes: daily diurnal pattern (valley at night, peak in the morning, and plateau during the day), weekly pattern.
 - Higher layers combine those to recognize complex events, dependencies, and context. Ex. Holiday Effect (Monday + Christmas $\neq$ normal Monday); Causal relationships, long range dependence and multivariate interactions (using diverse inputs);
- This compositional structure is incredibly powerful for complex tasks like image and speech recognition (and time series analysis).

Hierarchical Feature Learning:

Layer Level - Receptive Field - What it Learns - Example (Energy Consumption)

Lower

Small (Hours) - Simple, local patterns, point anomalies - A sudden 5-minute spike, a sharp drop;

Middle

Medium (Days) - Composite shapes, repeating regimes - The daily rise-and-fall pattern, a heatwave period;

Higher

Large (Weeks+) - Complex events, long-range dependencies, context - “Christmas morning,” “heatwave ending in a storm”.

- Parameter Efficiency: counterintuitively, a deeper network can often achieve the same function with fewer total parameters than a very wide, shallow network.
- This can lead to better generalization and less computational cost.
- Theoretical expressivity: A classic result (the Universal Approximation Theorem) states that a network with just one hidden layer can approximate any continuous function.
- However, it might require an exponentially large number of neurons in that single layer.
- Depth often provides a more efficient way to represent complex functions.

The pitfalls of more layers

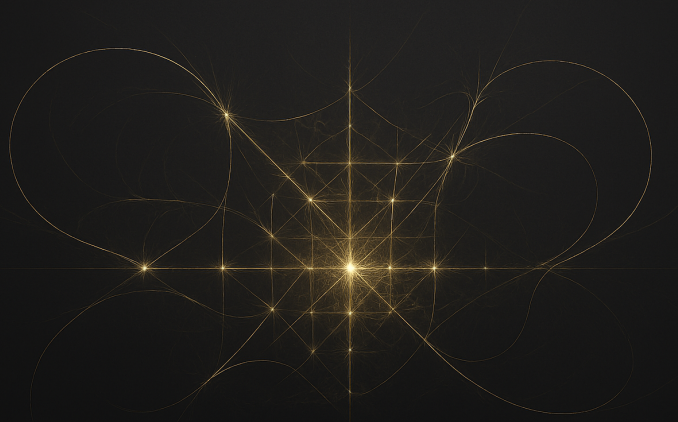
- Simply stacking layers without consideration leads to diminishing returns and eventually worse performance.
- Vanishing/Exploding Gradients.
- Overfitting.
- Increased Computational Cost: more layers = more parameters $\implies$ Longer training times + Longer inference times (making predictions) + Higher memory and energy requirements.
- Degradation Problem: the model may become harder to optimize.

How to Proceed?

- Start simple: model with 1 or 2 hidden layers as baseline. Use a reasonable number of neurons per layer (e.g., 64, 128).
- Gradually increase complexity: if the baseline model is underfitting, slowly add more layers and/or more neurons.
- Validate rigorously: the key metric is the validation/test error, not the training error.
- Look for signs: **underfitting** (high training error) $\Rightarrow$ increase model capacity (add layers/neurons).
overfitting (low training, high validation error): $\Rightarrow$ add regularization or use a simpler model (remove layers).
- For standard problems use proven architectures and potentially fine-tune them on your data. The depth has already been optimized by researchers.

Interpretability

- NN's are excellent under the predictive paradigm.
- The complex and non-linear nature of NN's make it difficult to understand what's under the hood of a given method.
- If interpretability is important to your application, then NN's may (typically) not be the answer.

An abstract, golden geometric pattern on a dark background. The pattern consists of a central square grid of points, with lines connecting them to form a complex, symmetrical design. The lines extend outwards, creating a series of overlapping circles and arcs that resemble a stylized, glowing network or a complex geometric structure. The overall effect is one of intricate, mathematical beauty.

Recurrent Neural Networks

Recurrent Neural Networks (RNN)

- RNN is a class of NN designed to recognize patterns in sequences of data, such as time series, speech, text and video;
- NNs are fixed length, requiring padding or truncation to handle variable-length sequences, which is inefficient.
- RNN can handle variable-length sequences by processing one element at a time while maintaining a hidden state that acts like a “memory”. This makes them ideal for tasks where context and order matter.
- With recurrent computation, the number of RNN model parameters does not grow as the number of time steps increases.

Vanilla RNNs process one element at a time *in loop*, different from the fixed structure of NNs.

- 1 The first element in the input, $\mathbf{x}_1$ and its initial hidden states $\mathbf{h}_0$ are used to update the hidden state through

$$\mathbf{h}_1 = a_h(W\mathbf{x}_1 + U\mathbf{h}_0 + b), \quad \mathbf{h}_1 = (h_{11}, \dots, h_{1p_1})',$$

where W and U are weight matrices and b is a bias term, a_h is an activation function.

- 2 The first output is created just like in NN through a linear combination + activation:

$$\mathbf{y}_1 = a_y \left(w_{01} + \sum_{j=1}^{p_1} w_{j1} h_{1j} \right),$$

where a_y denotes the activation function applied.

- 3 The second element of the sequence $\mathbf{x}_2$ is processed using $\mathbf{h}_1$ to update the hidden state via

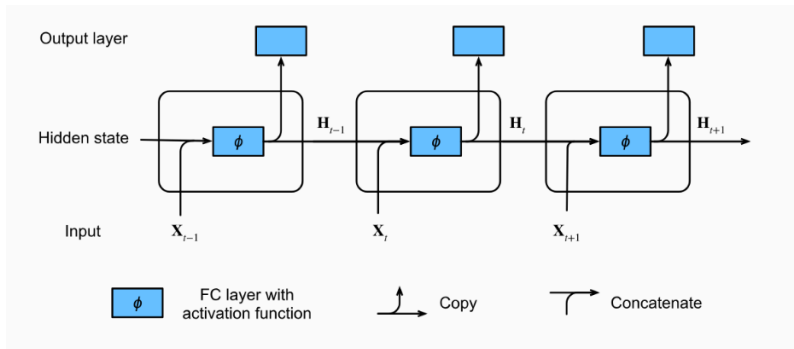
$$\mathbf{h}_2 = a_1(W\mathbf{x}_2 + U\mathbf{h}_1 + b), \quad \mathbf{h}_2 = (h_{21}, \dots, h_{2p_2})',$$

with the **same** W , U and b .

- 4 The second output:

$$\mathbf{y}_2 = a_y \left(w_{02} + \sum_{j=1}^{p_2} w_{j2} h_{2j} \right).$$

- 5 This process repeats for every element in the sequence (called the *RNN unfold*).



RNN with a hidden state.

<http://www.d2l.ai/>

Vanilla RNN: strengths

- RNNs introduce recurrent connections that allow information to persist over time;
- RNNs can model sequences where the order and context matter.
- Parameter sharing across time reduces the number of parameters compared to NNs.
- RNNs can process sequences of arbitrary length by unfolding over time.
- RNNs can capture temporal dependencies (NNs cannot handle this without manually engineering lagged inputs).

Vanilla RNN: the drawbacks

- For very long sequences, the traditional RNN architecture is computationally expensive;
- It struggles to model long-range dependence.
- But the biggest problem is that training RNNs is notoriously difficult due to the vanishing/exploding gradient problem.
- The core problem in a vanilla RNN is the repeated application of the same non-linear function, which compresses values and forces gradients to shrink exponentially with each time step.

LONG SHORT-TERM MEMORY

$$l_t = o(W_U + U_A h_{t-1} + b_j)$$

$$f_E = o(W_{E'} + U_j h_{t-1} + b_j)$$

$$f_L = o(W_{LT} + U_j h_{t-1} + b_j)$$

$$o_t = o(W_{Ov} + U_O h_{t-1} + b_j)$$

Long Short-Term Memory (LSTM)

- The LSTM architecture was designed to handle the vanishing/exploding gradient problem.
- To achieve that, LSTMs store more information outside of the traditional neural network flow in structures called gated cells.
- To make things work in an LSTM an output gate O_t is used to read entries of the cell, an input gate I_t reads data into the cell and a forget gate F_t resets the content of the cell.
- These gates are computed in similar fashion to traditional RNNs, but with more variables.
- There is also a memory cell which together with the other gates controls how memory content fluxes along the layers.

Long Short-Term Memory (LSTM)

- The output, input, forget gates and candidate memory are updated via

$$\mathbf{O}_t = a(\mathbf{X}_t W_{x0} + \mathbf{h}_{t-1} W_{h0} + \mathbf{b}_0);$$

$$\mathbf{I}_t = a(\mathbf{X}_t W_{xI} + \mathbf{h}_{t-1} W_{hI} + \mathbf{b}_I);$$

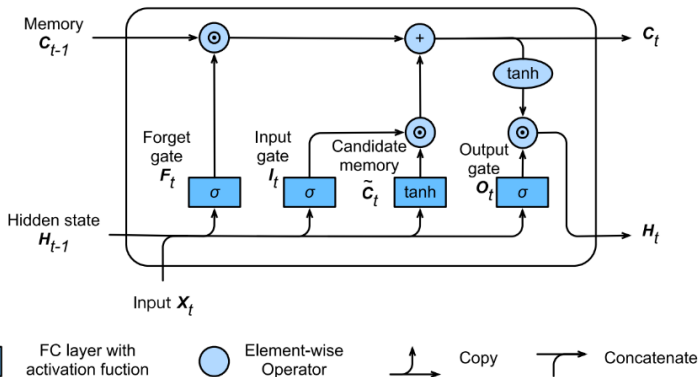
$$\mathbf{F}_t = a(\mathbf{X}_t W_{xF} + \mathbf{h}_{t-1} W_{hF} + \mathbf{b}_F);$$

$$\check{\mathbf{C}}_t = \tanh(\mathbf{X}_t W_{xC} + \mathbf{h}_{t-1} W_{hC} + \mathbf{b}_C),$$

where a is the sigmoid, $W_{..}$ are weight matrices and $\mathbf{b}_{..}$ are biases.

- To plug some things together, $\mathbf{C}_{t-1}$ controls how much of the old memory content we want to preserve, and then the next hidden state is computed, which is done in two steps

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \check{\mathbf{C}}_t, \quad \mathbf{h}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t).$$



Hidden state update diagram in LSTM.

<http://www.d2l.ai/>

LSTM gates solve the vanishing gradient problem with two very clever key mechanisms:

- The Additive Cell State (C_t): unlike multiplication in an RNN, distributes gradients evenly. A gradient arriving at C_t can pass backwards to C_{t-1} essentially unchanged through the forget gate, preventing exponential decay.
- Gated Linear Flow: the cell state's update is linear (simple addition and multiplication). The gates themselves use sigmoids, but the critical path for long-term memory (the cell state) avoids being squashed by a tanh at every single step.
- The output gate protects the cell state from being immediately compressed on its way to the hidden state.

LSTM: Pros and Cons

Pros:

- ➊ (Elegantly) solves the vanishing gradient problem;
- ➋ Superior handling of long-term dependencies compared to traditional RNNs;
- ➌ Modeling complex, sequential patterns, especial temporal dynamics;
- ➍ A versatile architecture with wide applicability.

Cons:

- ➊ Tendency to overfit. Their high capacity and complexity make them prone to overfitting, especially on smaller datasets;
- ➋ LSTM are computationally expensive to train (relatively). Each LSTM cell has significantly more parameters (4 times more than a simple RNN cell);

LSTMs are useful for handling problems with long-term dependencies in sequential data, where you have a moderate amount of data and computational budget is not the primary constraint.

The background of the slide features a light gray circuit board pattern with various lines and small dots, set against a white background. At the top, there is a horizontal bar with an orange-to-white gradient.

GATED RECURRENT UNIT

Gated Recurrent Unit (GRU)

- GRU is a simplified version of the LSTM memory cell, offering comparable performance with a smaller computational cost.
- The three LSTM gates (input, forget and output) are replaced by only two, called the reset and the update gate.
- Intuitively:
 - The reset gate controls how much of the previous state we might still want to remember. It helps capturing short-term dependencies in sequences.
 - The update gate controls how much of the new state is just a copy of the old one. It helps capturing long-term dependencies in sequences.

- The reset and update gates and the candidate memory are updated via

$$\begin{aligned}\mathbf{R}_t &= a(\mathbf{X}_t W_{xR} + \mathbf{h}_{t-1} W_{hR} + \mathbf{b}_R); \\ \mathbf{Z}_t &= a(\mathbf{X}_t W_{xZ} + \mathbf{h}_{t-1} W_{hZ} + \mathbf{b}_Z).\end{aligned}$$

where a is the sigmoid, $W_{..}$ are weight matrices and $\mathbf{b}_{..}$ are biases.

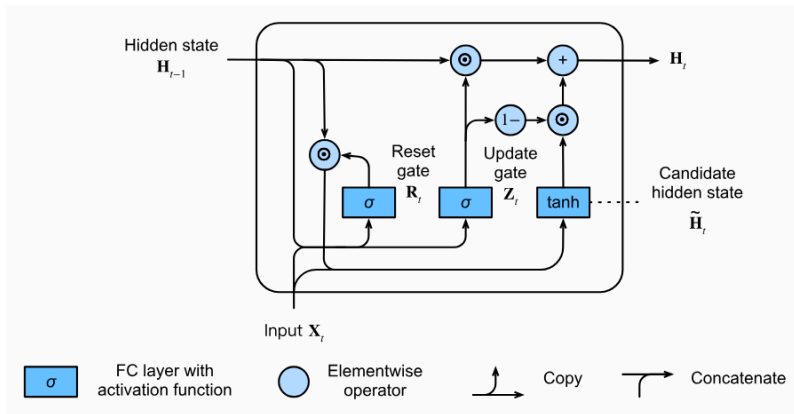
- Next, a candidate hidden state $\check{\mathbf{h}}_t$ is obtained via

$$\check{\mathbf{h}}_t = \tanh(\mathbf{X}_t W_{xh} + (\mathbf{R}_t \odot \mathbf{h}_{t-1}) W_{hh} + \mathbf{b}_h),$$

- Observe that any entry in the reset gate $\mathbf{R}_t$ close to 1 will imply that the entry of $\mathbf{R}_t \odot \mathbf{h}_{t-1}$ to be close to the respective $\mathbf{h}_{t-1}$ entry and we recover the vanilla RNN.
- Similarly, if an entry of $\mathbf{R}_t$ is close to 0, the entry of candidate hidden state will be same as an MLP with $\mathbf{X}_t$ as input, resetting the respective hidden state entry to the default.
- Finally, we need to incorporate the effect of the update gate:

$$\mathbf{h}_t = \mathbf{Z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{Z}_t) \odot \check{\mathbf{h}}_t,$$

which is simply an entry-wise convex combination between the previous state $\mathbf{h}_{t-1}$ and the candidate $\check{\mathbf{h}}_t$ with weights $\mathbf{Z}_t$.



Hidden state update diagram in GRU.

<http://www.d2l.ai/>

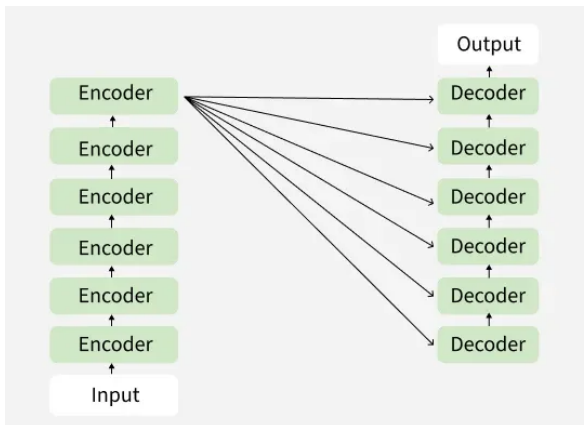
Encoder-Decoder

- Encoder-decoder model is a type of NN mainly used for tasks where both the input and output are sequences, possibly of different lengths and position.
- Examples: neural machine translation, text summarization, image captioning and speech recognition.
- Mapping a token in the input to one in the output is often indirect.
- For example, input: “once upon a time” output “il était une fois” in French and “era uma vez” in Portuguese.

Encoder-Decoder

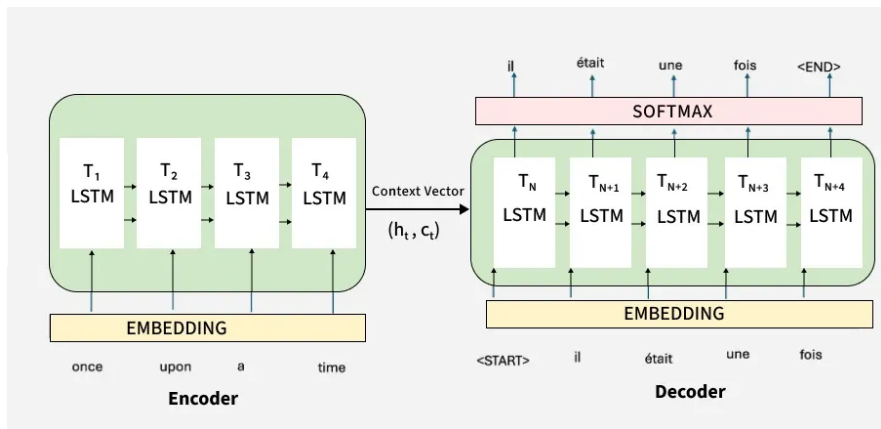
- **Encoder:** takes the input data and creates a single, fixed-size summary of the entire input (the context vector or latent space).
- The context vector is a task-specific summary vector of fixed length optimized to help the decoder generate the target sequence.
- It may capture information useful for translation, say, but might not encode semantics completely.
- **Decoder:** takes the context vector and produces/predicts the output one step at a time.

Encoder-Decoder model

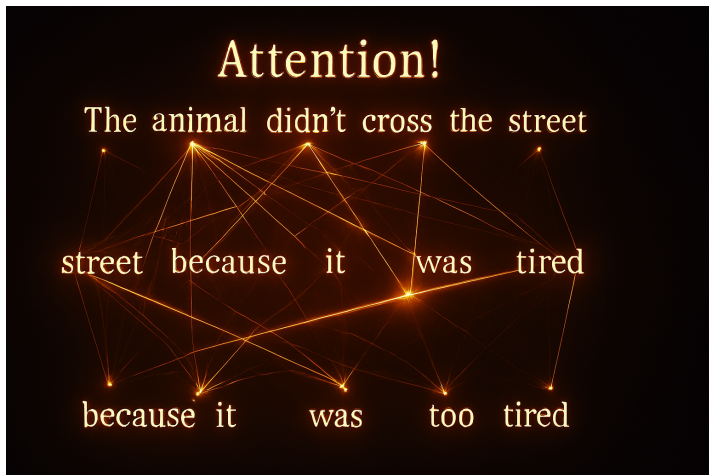


<https://www.geeksforgeeks.org/nlp/encoder-decoder-models/>

Encoder-Decoder LSTM Architecture



<https://www.geeksforgeeks.org/nlp/encoder-decoder-models/>



The rise of Attention! - copilot generated image

The Rise of Attention!

- One drawback of encoder-decoder layers is that all information had to be squeezed into a single bottleneck vector, which can be hard for long sequences.
- An attention layer is a mechanism that allows a model to focus on the most relevant parts of the input when making a prediction, instead of treating all inputs equally.
- Instead of passing just one vector, the decoder can attend to different parts of the encoder's outputs at each decoding step.
- This gave rise to encoder-decoder with attention, which became the dominant approach.

- In its essence, attention is a dynamic re-weighting of inputs/outputs based on their relevance to the query.
- Given a sequence of input vectors (e.g., words in a sentence, time steps in a series), the attention layer:
 - 1 Projects the inputs into three spaces (matrices):
Query (Q) – what we're looking for.
Key (K) – what each input offers (d_k columns).
Value (V) – the element's actual content or information.
 - 2 Computes attention scores:

$$\text{Score}(Q, K) = \frac{QK'}{\sqrt{d_k}}$$

- 3 Normalizes scores with a softmax, turning them into weights

$$\text{Attention}(Q, K, V) = \text{Softmax}(\text{Score}(Q, K))V.$$

Example: You walk into a library looking for material on *neural networks*.

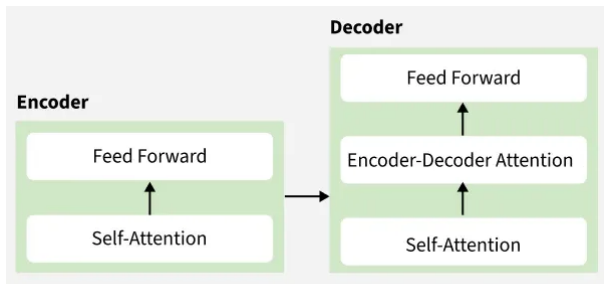
- Values (**V**): The actual content of each book. Say V_i = text of book i .
- Keys (**K**): The label/topic of each book, used to compare with the query.
Say K_i = “culinary”, “neural networks”, “statistics”, etc.
- Query (**Q**): What you are looking for, built from your intent.
Here, Q = “neural networks”.

Attention mechanism: compares your query with each book’s label, assigns high weight to the book on neural networks and low weights to the others, and returns a weighted synthesis of their contents – a *context vector*.

Important distinction:

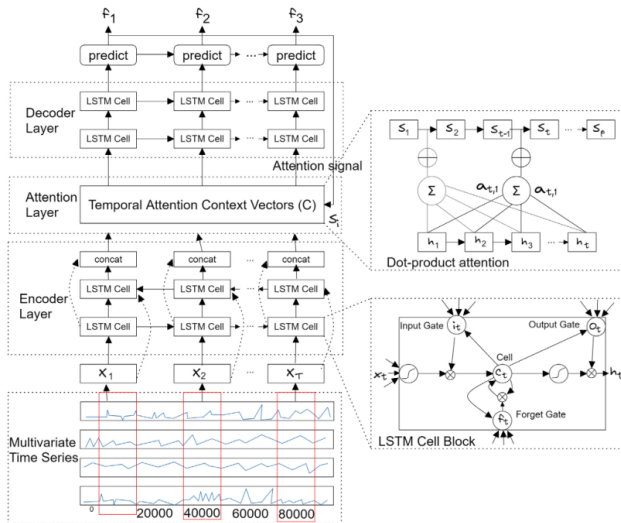
- Regular attention mechanism: Q comes from the decoder, while K and V come from the encoder. Used to align input and output sequences: translate “The cat” $\rightarrow$ attend to encoder outputs.
- Self-Attention mechanism: Q , K and V comes from the same sequence. Capture relationships within a single sequence. Forecast tomorrow $\rightarrow$ attend to previous timesteps.
- In the context of time-series, self-attention is usually the one applied.

Encoder-Decoder with self-attention



<https://www.geeksforgeeks.org/nlp/encoder-decoder-models/>

The Encoder reads the input sequence and transforms it into a rich, context-aware representation; the Decoder uses that representation to produce the output sequence.



A complex deep learning architecture (model) for time series
 The MTSMFF architecture of Du et al. (2020)

❶ Long-range dependencies

- RNNs struggle to capture relationships between distant timesteps.
- Attention connects every input to every output directly.

❷ Encoder-decoder bottleneck

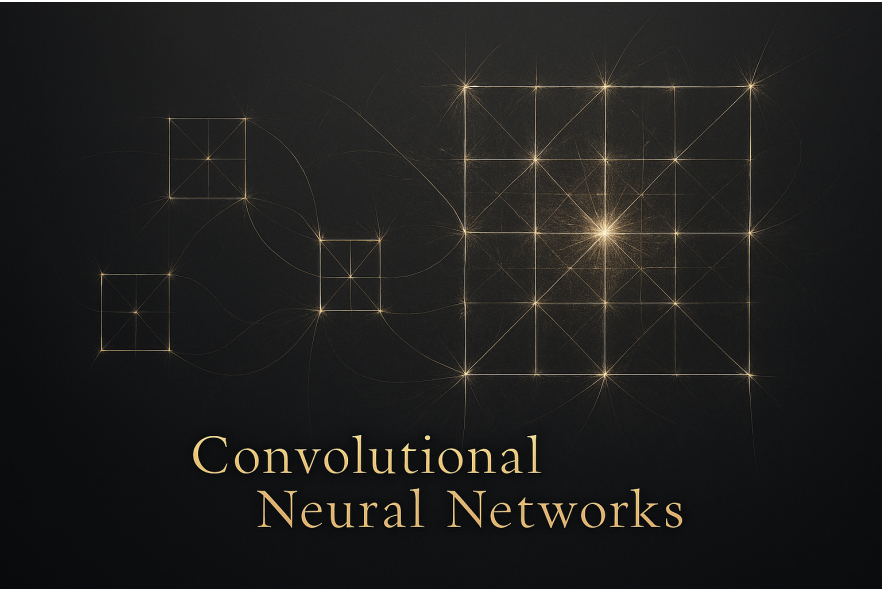
- Classic encoder-decoder compresses all info into a single vector.
- Attention allows dynamic access to all encoder outputs.

❸ Parallelization & efficiency

- RNNs process sequences step by step $\rightarrow$ slow.
- Self-attention is parallelizable across the sequence.

❹ Adaptive importance weighting

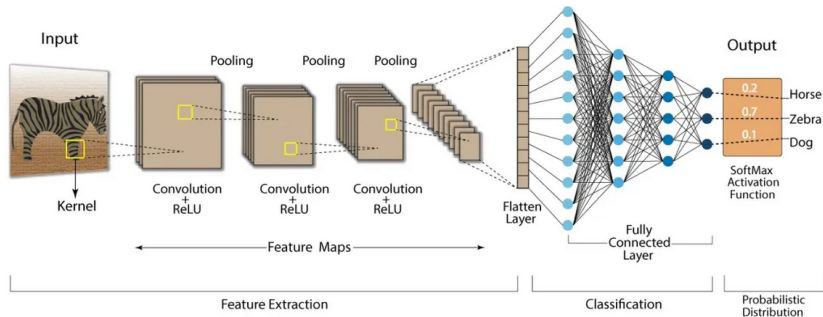
- Fixed-size windows treat all inputs equally.
- Attention assigns dynamic weights depending on relevance.



Convolutional Neural Networks

- The idea of CNNs was developed much earlier than RNNs: LeCun (1989) developed the LeNet;
- But their usage in the context of time series started after 2017, especially with the Temporal Convolutional Network (TCN) of Bai et al. (2018);
- Nowadays it incorporates attention layers, dropouts, blocks containing RNNs among other technologies;
- One feature of CNN is that it can conduct different processes in parallel.

CNN layers



<https://developersbreach.com/convolution-neural-network-deep-learning/>

Basic architecture:

- Convolutional layers;
- Pooling layers;
- Flatten layer;
- Fully connected (dense) layers;
- The so-called ReLU activation: $\max\{0, x\}$;
- Dropouts;

Understanding Hyperparameters

The interface displays the following settings and visualizations:

- Input Size:** 5 (with a slider and up/down arrows)
- Padding:** 1 (with a slider and up/down arrows)
- Kernel Size:** 3 (with a slider and up/down arrows)
- Stride:** 2 (with a slider and up/down arrows)

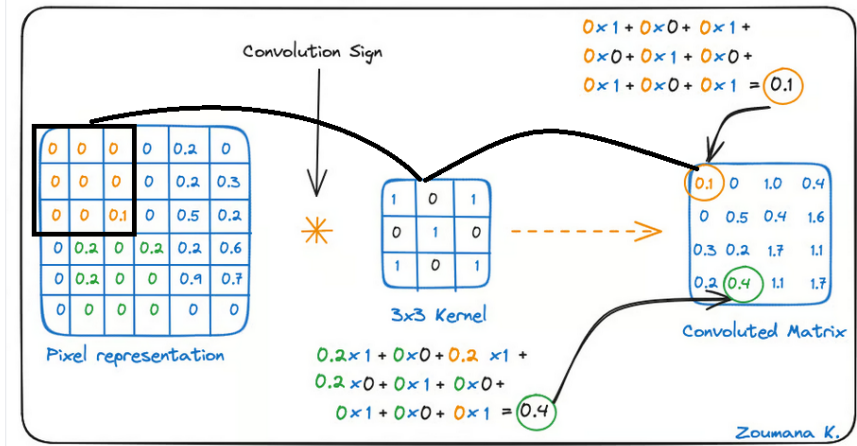
Input (5, 5) After-padding (7, 7)

Output (3, 3)

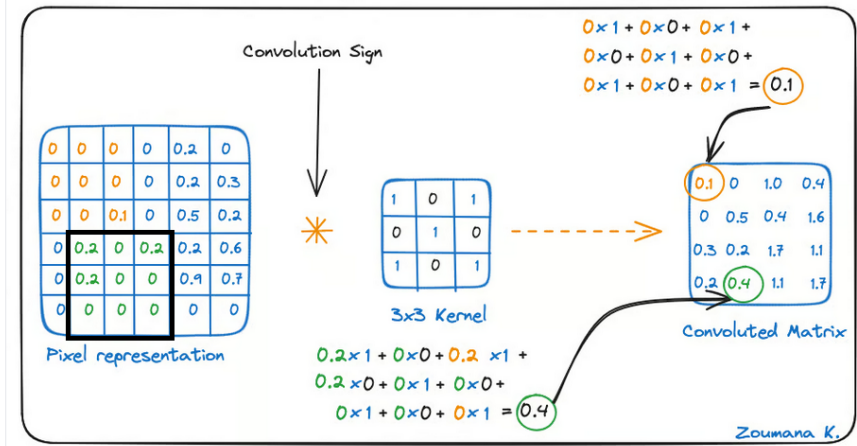
Hover over the matrices to change kernel position.

<https://poloclub.github.io/cnn-explainer/>

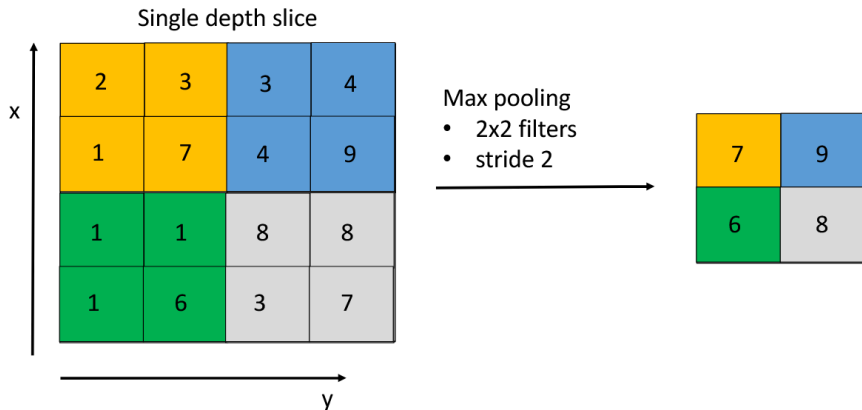
Application of the convolution task using a stride of 1 with 3x3 kernel



Application of the convolution task using a stride of 1 with 3x3 kernel



- The convolutional layers are the foundation of CNN, as they contain the learned kernels (weights), which extract features that distinguish different features of the object.
- Convolutions can also be thought of as regular neural networks with two constraints: Local connectivity and Weight sharing.
- The former comes from the fact that a convolutional filter has much smaller dimensions than the input on which it operates.
- The latter comes from the fact that in such a framework the same filter is applied across the input.



Roffo (2017)

<https://arxiv.org/pdf/1706.05933>

Advantages of CNN

- High Accuracy: They can achieve high accuracy in various image recognition tasks.
- Efficiency: They are efficient, especially when implemented on GPUs.
- Robustness: They are robust to noise and variations in input data.
- Adaptability: It can be adapted to different tasks by modifying their architecture.

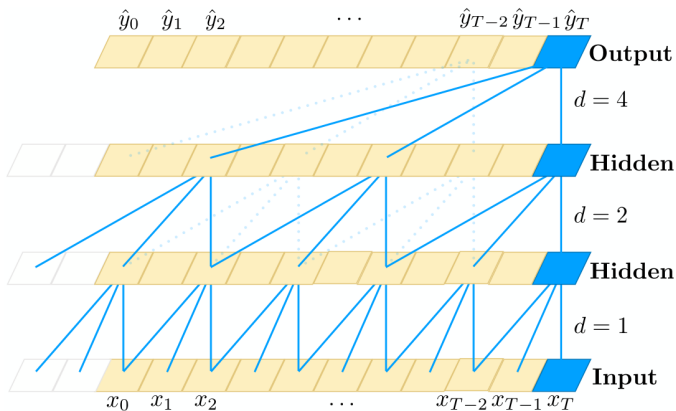
Disadvantages of CNN

- Complexity: It can be complex and difficult to train, especially for large datasets.
- Resource-Intensive: It requires significant computational resources for training and deployment.
- Data Requirements: They need large amounts of labeled data for training.
- Interpretability: They can be difficult to interpret making it challenging to understand their predictions

TCN

Temporal Convolution Networks

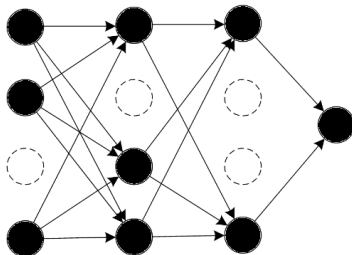
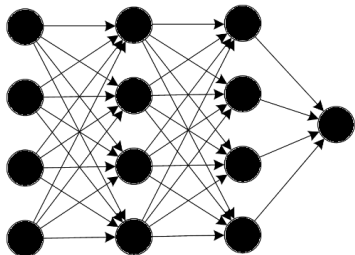
- The convolution layer now slides only in 1 dimension: time and only considering past values (causal layers).
- A size kernel k operates in k consecutive time stamps.
- For time series, the architecture must use causal padding (at time T , padding is used for non-observed values of $t < T$).
- Dilated Convolution layers are employed: the kernel is applied over inputs that are spaced apart.
- Residual Block (to avoid gradient problems).



Causal dilated convolution - Bai et. al (2018)

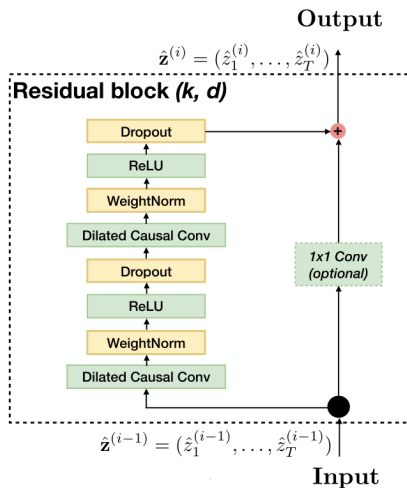
<https://arxiv.org/pdf/1803.01271>

Dropout



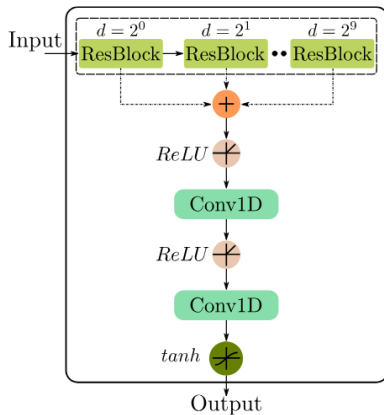
Left: dense layer with all nodes active.

Right: dense layers with some nodes dropped (weight set to 0).



TCN's Residual block - Bai et. al (2018)

<https://arxiv.org/pdf/1803.01271>



TCN block architecture

Guirguis et. al (2020)

<https://arxiv.org/pdf/2003.01609>

TCN's Strengths

- **Parallelism during training.** The biggest strength over RNNs. A TCN can apply convolutions to all time steps in a layer simultaneously allowing for massive parallelization on GPUs.
- **Stable, parallelizable gradients.** TCN's backpropagation path is different and typically shorter (goes through layers, not time steps), leading to more stable training. Furthermore, gradients can be computed in parallel for an entire sequence.
- **Large and flexible receptive field** Through the use of dilated convolutions and residual blocks, a TCN can capture very long-term dependencies without an exponential increase in parameters or loss of resolution (unlike pooling in CNNs).

TCN's Strengths

- **Causality Enforcement.** The model's architecture itself guarantees causality, which you must manually ensure in many other architectures.
- **Good performance on many tasks.** TCNs often outperform canonical RNNs and LSTMs on a variety of sequence modeling tasks, including audio synthesis, time series forecasting, and language modeling, while training much faster.

TCN's Weaknesses

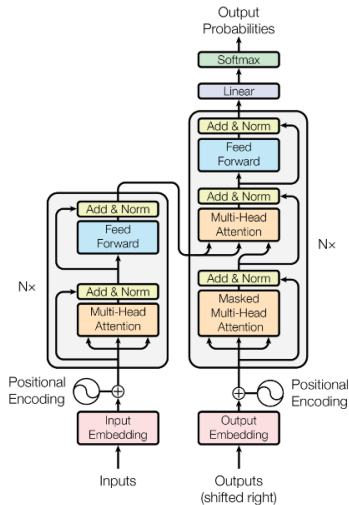
- **Inference (evaluation) latency on long sequences.** While *training* is parallelizable, generating a new prediction for a very long, unseen sequence during inference can be slow. To predict the next time step, the model must process the entire input history up to that point through all layers.
- **Data-intensive memory requirements.** To process a long sequence in parallel, the entire input must be stored in memory, this can lead to high memory consumption during training, which can be a limiting factor.
- **Lack of dynamism.** The receptive field of a TCN is determined at design time by its depth, dilation factors, and kernel sizes. If a task requires a different context window, the model must be redesigned and retrained.

TCN's Weaknesses

- **Potential parameter inefficiency.** For long-range dependence capturing it is necessary to stack several residual blocks, despite the help of the dilated convolutional layers.



- The Transformer architecture, introduced in the groundbreaking 2017 paper “Attention Is All You Need” by Vaswani et al., is a profound shift from previous sequence models.
- It abandoned recurrence and convolution entirely, relying solely on the self-attention mechanism.
- Transformers consistently achieve state-of-the-art results on a vast array of tasks, especially NLP-related.
- Of course the success of transformer-based architecture in NLP-related tasks prompt researchers to use it for time series forecasting.



The Transformer architecture.

<https://arxiv.org/pdf/1706.03762>

Why Are Transformers So Much Better?

- ❶ **Parallelization:** This is the biggest practical advantage over RNNs/LSTMs. Since self-attention processes all words simultaneously, training is dramatically faster on modern hardware (GPUs/TPUs), which are designed for parallel computation.
- ❷ **Long-range dependency modeling:** self-attention connects any two positions in the sequence with a constant number of computational steps, making it effective at capturing context from anywhere in the input, regardless of distance.

Why Are Transformers So Much Better?

- ➊ **Superior performance:** Transformers consistently achieve state-of-the-art results on a vast array of tasks (translation, summarization, question-answering, etc.).
- ➋ **Scalability:** The architecture scales very well with more data and larger models, leading to the era of LLMs like GPT, BERT, and their successors.

What Are Transformers Drawbacks?

- ❶ **Computational Complexity:** Self-attention has a complexity of $O(n^2)$ with respect to sequence length. For very long sequences (e.g., long documents, high-resolution images), this becomes prohibitively expensive in terms of both memory and computation.
- ❷ **Data Hunger:** Transformers typically require massive amounts of training data to reach their full potential, unlike simpler models that can work on smaller datasets.

What Are Transformers Drawbacks?

- ❶ **Black Box Nature:** They are even less interpretable than LSTMs. While attention weights can be visualized, interpreting the model's reasoning across dozens of layers and attention heads remains a significant challenge.
- ❷ **Environmental Cost:** Training massive transformer models (Claude, ChatGPT, Gemini, etc.) consumes immense computational resources, leading to a high carbon footprint.
- ❸ **Hardware Inflation:** The AI boom has driven up prices and demand for consumer hardware — RAM sticks, SSDs, GPUs are becoming more expensive and harder to find, affecting prices of laptops, PCs, and even cell phones.

How Transformers became GPTs?

- **Architectural Simplification:** The original Encoder-Decoder Transformer was stripped down to a Decoder-only stack, removing the parts not needed for generation.
- **Training Revolution:** This new architecture was then trained on a colossal scale using a simple unsupervised objective (next token prediction).
- **Scalability:** The resulting model (GPT) demonstrated that simply scaling up this approach (more data, more parameters) led to emergent abilities and unprecedented performance.
- For instance, GPT-3 has 175 billion parameters, whereas for GPT-4, this number is speculated at ~ 1.8 trillion total parameters, but *only* about 280 bi are activated per token (Mixture of Experts architecture). GPT 5.5 ~ 9.7 trillion pars.

- As Transformer becomes the norm in deep learning, more publications have begun the debate about its effectiveness in time series
- Adaptation of transformers and GPTs to the context of time series analysis is not without challenges.
- Transformers are locally-agnostic, that is, the attention mechanism is insensitive to the local context, which can make the model prone to anomalies in time series forecasting.
- Computational complexity is $O(n^2)$ and memory requirement are extremely high, making it expensive for multivariate long time series.

- Compared to natural language, temporal data exhibits lower information density, necessitating longer sequences to capture temporal patterns.
- Temporal dynamics, rapid evolution, and the presence of both long and short-term effects are difficult to train.
- Time series data is complex in nature and without specially designed attention mechanism or specific modules, Transformers struggle in learning these characteristics.

- Interestingly, the classical time series decomposition into trend + seasonality + noise has been a primary focus in the research of Transformers.
- Working in time domain has proven to be more effective in feature differentiation, aiding in identifying repetitive patterns, periodic trends, and frequency-domain features such as seasonal patterns or periodic oscillations.
- The adoption of patch-based data organization has become prevalent, through segmenting long time series into smaller patches, serving as input tokens for the Transformer.

Recent reviews

Kong, X., Chen, Z., Liu, W., et al. (2025). *Deep learning for time series forecasting: a survey*. International Journal of Machine Learning and Cybernetics, 16, 5079–5112.

W. Li and K.L.E. Law (2024). *Deep Learning Models for Time Series Forecasting: A Review*. IEEE Access, 12, 92306-92327.

Wang, Y., Wu, H., Dong, J., et al. (2024). *Deep Time Series Models: A Comprehensive Survey and Benchmark*. <https://arxiv.org/pdf/2407.13278>.

Casolaro, A., Capone, V., Iannuzzo, G. and Camastra, F. (2023). *Deep Learning for Time Series Forecasting: Advances and Open Problems*. Information, 14(11), 598.

- Bai, S., Kolter, J.Z. and Koltun, V. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. <https://arxiv.org/pdf/1803.01271>.
- Vawamani, A., Shazeer, N., Parmar, N., et al. (2023). *Attention Is All You Need* <https://arxiv.org/pdf/1706.03762>.
- Guirguis, K., Schorn, C., Guntoro, A., et al. (2020) *SELD-TCN: Sound Event Localization & Detection via Temporal Convolutional Networks*. <https://arxiv.org/pdf/2003.01609>.
- Roffo, G. (2017). “Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications” - Ph.D. thesis - Università degli Studi di Verona. <https://arxiv.org/pdf/1706.05933>.
- Du, D., Su, B. and Wei, Z. (2023). “Preformer: Predictive transformer with multiscale segment-wise correlations for long-term time series forecasting”, in Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP), pp. 1–5.