UNIVERSIDAD YACHAY TECH

ESCUELA DE CIENCIAS FÍSICAS Y NANOTECNOLOGÍA

The 46th International School for Young Astronomers ISYA 2025: Ecuador

In[•]:=
```
nbook = EvaluationNotebook[];
```

Demo ChatGPT

El material se puede encontrar en:

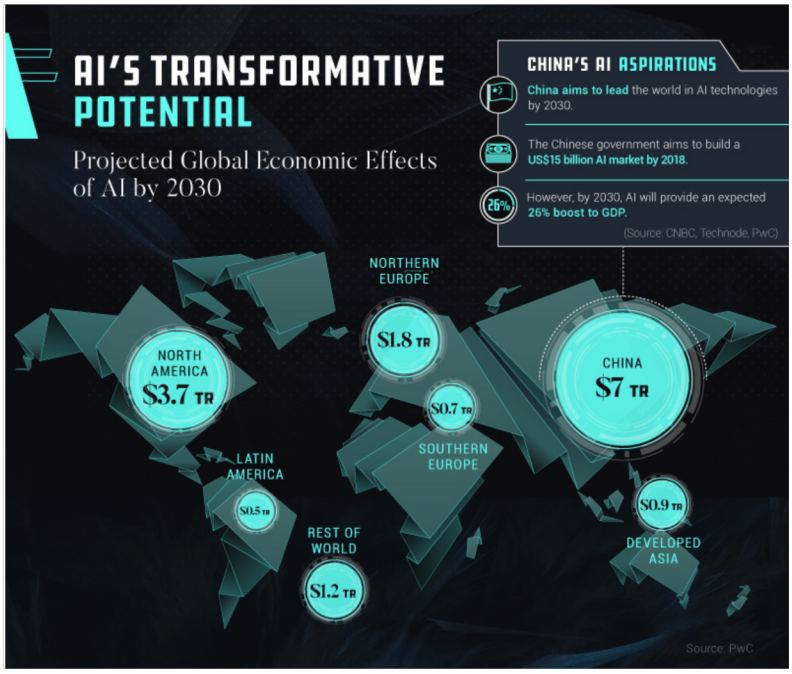
https://github.com/josemramirez/ChatGPT_curso_2025_Mathematica_v1

PDF:
https://drive.google.com/file/d/17oK2jn7bpv_fQYRBtahXYvLb7I2Xk4s4/view?usp=sharing

Out[ ]=

◄ prev – | + prox ►

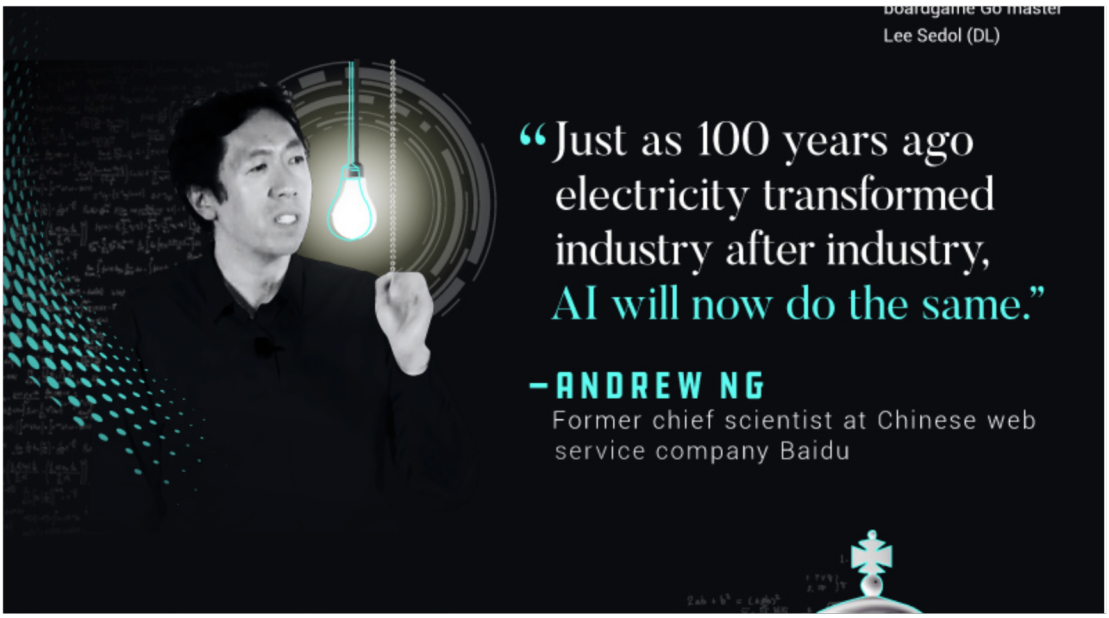
¿Dónde estamos? 101



Crédito: https://www.visualcapitalist.com/economic-impact-artificial-intelligence-ai/

Out[•]=

◄ prev – | + prox ►



x

Crédito: https://www.visualcapitalist.com/economic-impact-artificial-intelligence-ai/

Out[•]=

◄ prev – | + prox ►

Inteligencia Artificial

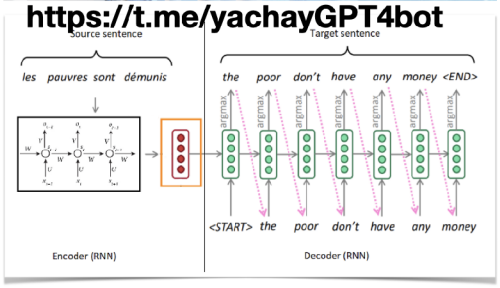
Redes Neuronales,
Natural Language Processing (NLP)
Un curso introductorio 101

Demo de yachayGPT

https://t.me/yachayGPT4bot

José Manuel Ramírez, ISYA Julio 2025

https://t.me/yachayGPT4bot



Out[-]=

◄ prev – | + prox ►

Generative Adversarial Nets (GANs)

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[†], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[‡], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

1 Introduction

The promise of deep learning is to discover rich, hierarchical models [2] that represent probability distributions over the kinds of data encountered in artificial intelligence applications, such as natural images, audio waveforms containing speech, and symbols in natural language corpora. So far, the most striking successes in deep learning have involved discriminative models, usually those that map a high-dimensional, rich sensory input to a class label [14, 22]. These striking successes have primarily been based on the backpropagation and dropout algorithms, using piecewise linear units [19, 9, 10] which have a particularly well-behaved gradient . Deep *generative* models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context. We propose a new generative model estimation procedure that sidesteps these difficulties. [1]

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

4.1 Global Optimality of $p_g = p_{\text{data}}$

We first consider the optimal discriminator D for any given generator G.

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}$$

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity $V(G, D)$

$$\begin{aligned} V(G,D) &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{\boldsymbol{z}} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz \\ &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx \end{aligned} \tag{3}$$

For any $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function $y \to a\log(y) + b\log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $Supp(p_{\text{data}}) \cup Supp(p_g)$, concluding the proof. □

Note that the training objective for D can be interpreted as maximizing the log-likelihood for estimating the conditional probability $P(Y = y|\boldsymbol{x})$, where Y indicates whether $\boldsymbol{x}$ comes from p_{data} (with $y = 1$) or from p_g (with $y = 0$). The minimax game in Eq. 1 can now be reformulated as:

Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

Proof. For $p_g = p_{\text{data}}$, $D^*_G(\boldsymbol{x}) = \frac{1}{2}$, (consider Eq. 2). Hence, by inspecting Eq. 4 at $D^*_G(\boldsymbol{x}) = \frac{1}{2}$, we find $C(G) = \log\frac{1}{2} + \log\frac{1}{2} = -\log 4$. To see that this is the best possible value of $C(G)$, reached only for $p_g = p_{\text{data}}$, observe that

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[-\log 2] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[-\log 2] = -\log 4$$

and that by subtracting this expression from $C(G) = V(D^*_G, G)$, we obtain:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right) \tag{5}$$

where KL is the Kullback–Leibler divergence. We recognize in the previous expression the Jensen–Shannon divergence between the model's distribution and the data generating process:

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{6}$$

Since the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal, we have shown that $C^* = -\log(4)$ is the global minimum of $C(G)$ and that the only solution is $p_g = p_{\text{data}}$, i.e., the generative model perfectly replicating the data generating process. □

4.2 Convergence of Algorithm 1

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D^*_G(\boldsymbol{x}))]$$

then p_g converges to p_{data}

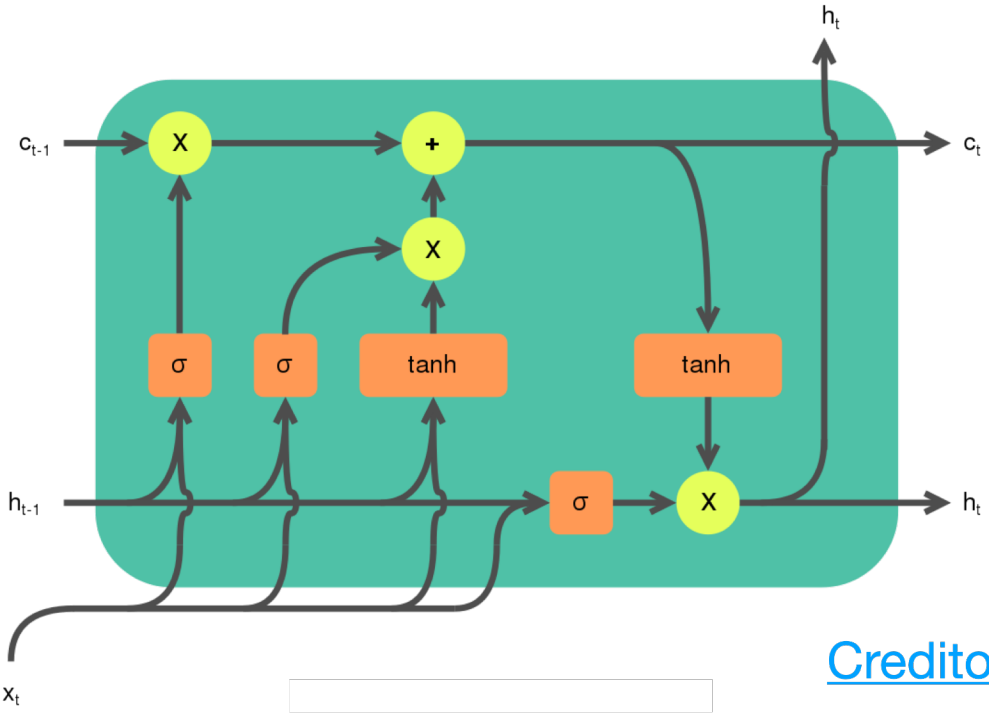
Proof. Consider $V(G, D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g. The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. In other words, if $f(x) = \sup_{\alpha\in\mathcal{A}} f_\alpha(x)$ and $f_\alpha(x)$ is convex in x for every α, then $\partial f_\beta(x) \in \partial f$ if $\beta = \arg\sup_{\alpha\in\mathcal{A}} f_\alpha(x)$. This is equivalent to computing a gradient descent update for p_g at the optimal D given the corresponding G. $\sup_{p_g} U(p_g, D)$ is convex in p_g with a unique global optima as proven in Thm 1, therefore with sufficiently small updates of p_g, p_g converges to p_x, concluding the proof. □

In practice, adversarial nets represent a limited family of p_g distributions via the function $G(\boldsymbol{z}; \theta_g)$, and we optimize θ_g rather than p_g itself. Using a multilayer perceptron to define G introduces multiple critical points in parameter space. However, the excellent performance of multilayer perceptrons in practice suggests that they are a reasonable model to use despite their lack of theoretical guarantees.

ChatGPT ¿Por qué funciona?
Zero2Hero (v1)

by JM Ramirez, Yachay 2025. (GPTZ2H)
basado en los escritos de:
https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/

Es solo agregar una palabra a la vez!

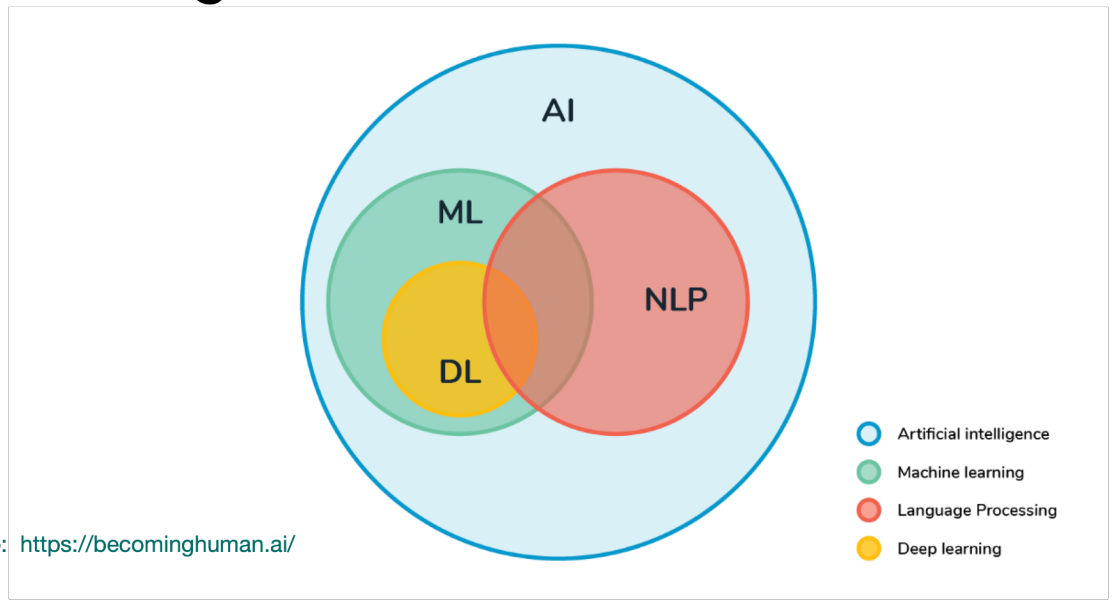


Credito: Wiki

Out[-]=

◄ prev – | + prox ►

¿Dónde estamos?



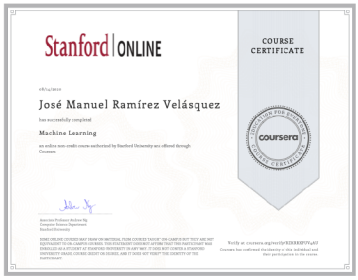
Credito: https://becominghuman.ai/

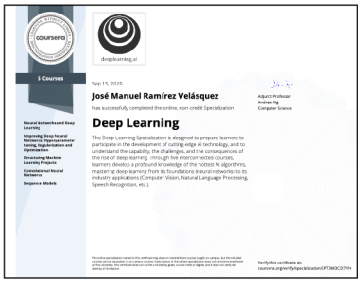
ChatGPT: modelos LLMs

- Machine Learning (Stanford)
- DeepLearning (DeepLearning.AI)
- Bayesian Statistics (HSE - Rusia)
- Alfa Tester (DeepLearning.AI)
- Business Analytics (Cambridge)

José M. Ramírez

DeepLearning.ai



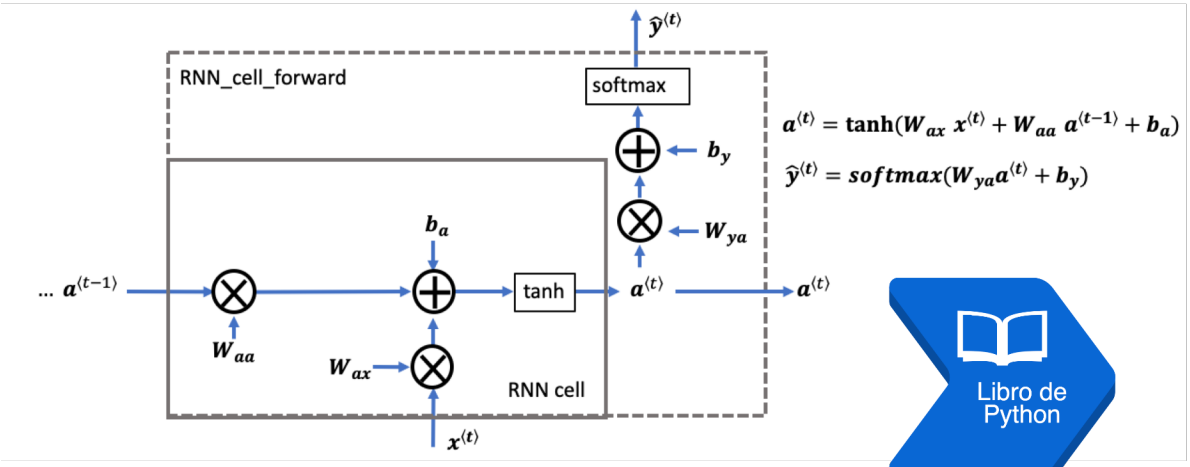


Out[-]=

◂ prev – + prox ▸

Una celda RNN (101)

x



Credito: DL.ai specialization

Libro de Python

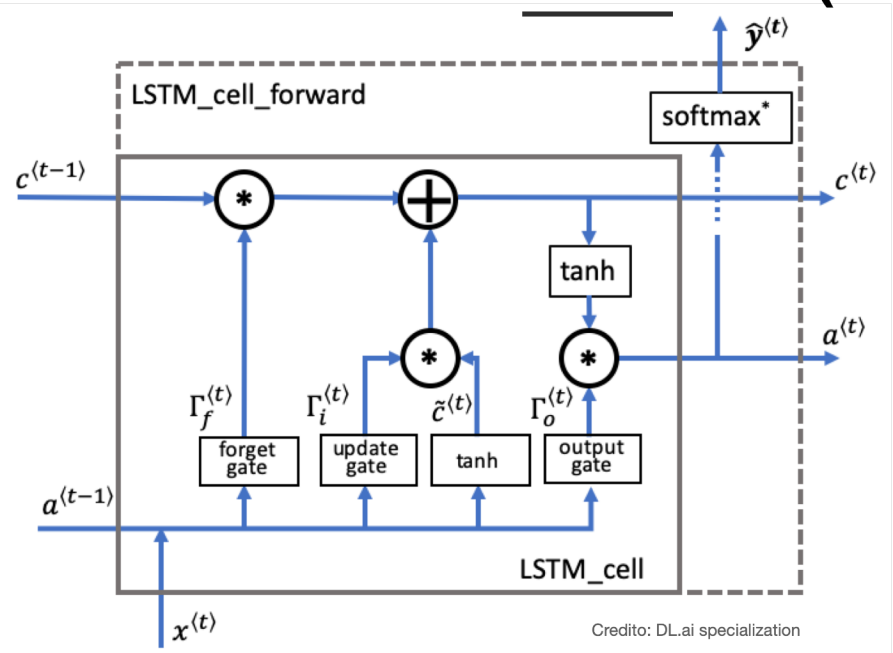
Out[-]=

◄ prev – + prox ►

The best thing about AI is its ability to ____

Una celda LSTM (101)

x



Libro de Python

Out[-]=

◄ prev – + prox ►

NN & A.I (v1.0)

Notas:

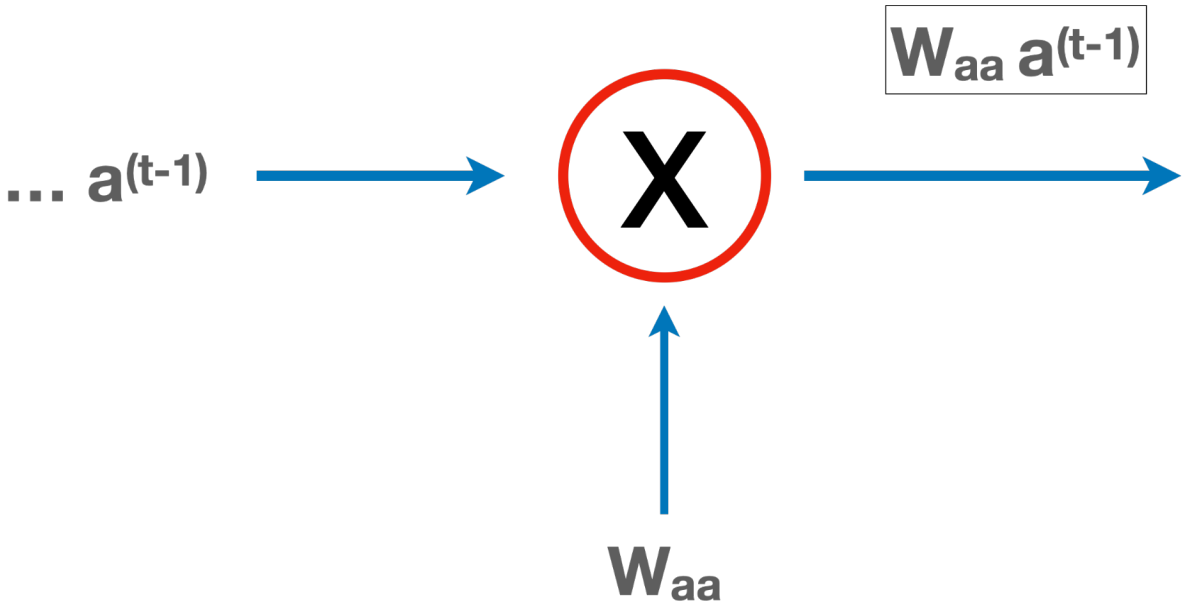
- **Un poco de algebra de vectores.**
- **Un poco de jerga computacional: Arreglos, Loops, Funciones, Argumentos, Variables, Inputs, Outputs.**
- **Láminas en Español, pero Libro de Python mezclado (estilo libre).**
- **Visualización. 2D Plots, 3D Plots.**
- **Un poco de ajustes de polinomios a datos experimentales.**

José Manuel Ramírez, Junio 2025

• • •

Out[ ]=

◄ prev – | + prox ►



Out[•]=

◄ prev – + prox ►

NN & A.I (v1.0)

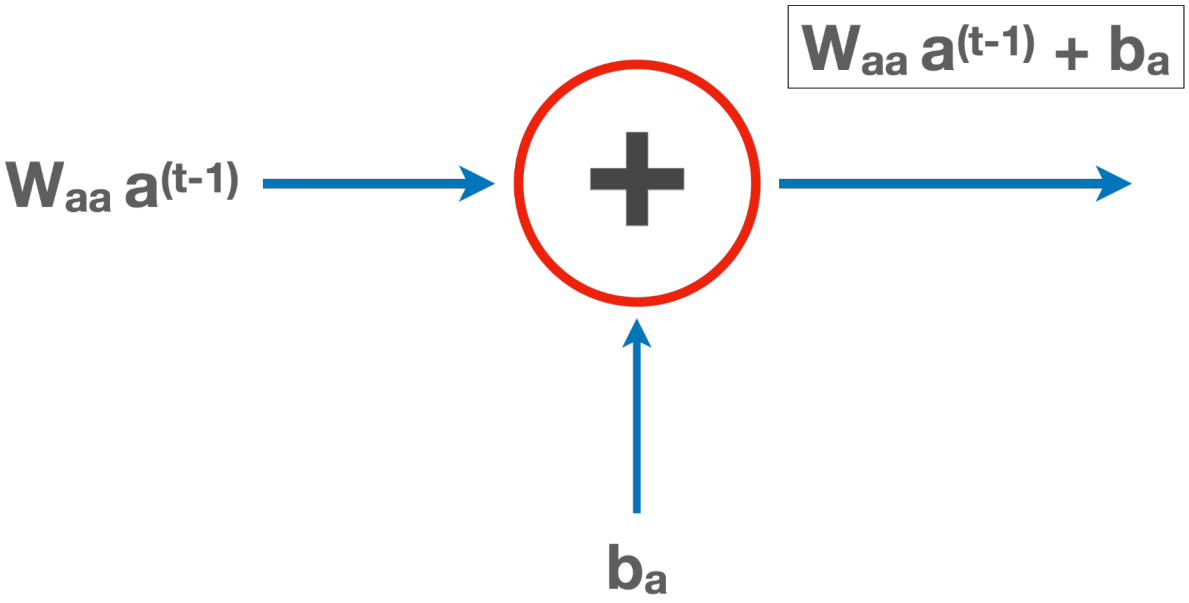
Notas:

- **Un poco de inteligencia artificial: Machine Learning, Deep Learning, NLP=Natural Language Processing.**
- **Input, output, activation functions, Matrices, operaciones con vectors.**
- **Celdas RNN, LSTM, GRU.**
- **“Entrenamiento” de una red neuronal.**

José Manuel Ramírez, Junio 2025

Out[ ]=

◄ prev – + prox ►



Out[-]=

◄ prev – | + prox ►

NN & A.I (v1.0)

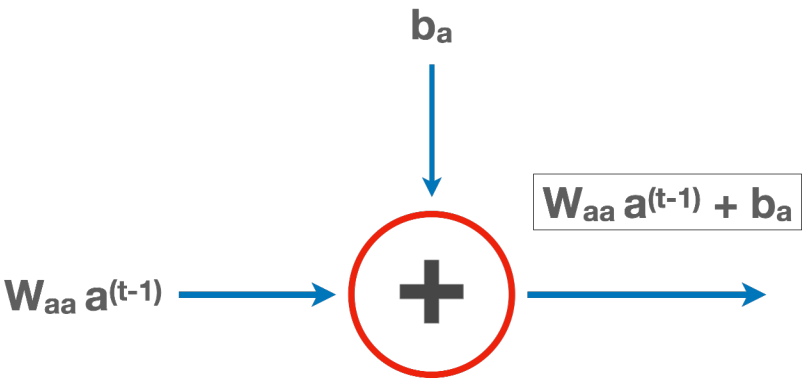
Material:

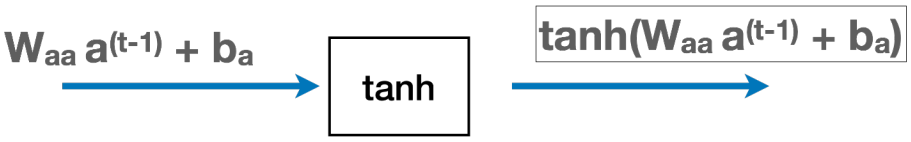
- **Una función tanh(x).**
- **Recurrent Neural Network (RNN).**
- **LSTM cell.**
- **Modelar la salida de un solo LSTM.**
- **Crear Una “red” de LSTM para predecir texto.**
- **Escritura de un código de Python para realizar la tarea!**
- **Comentarios finales.**

José Manuel Ramírez, Junio 2025

Out[ ]=

◄ prev – | + prox ►



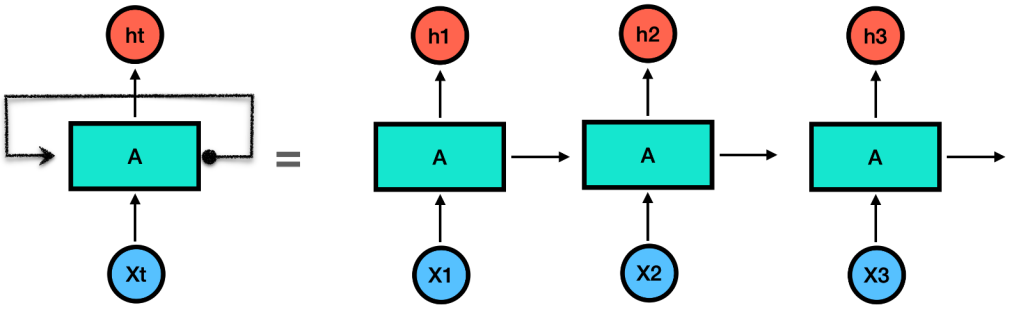


Out[•]=

◄ prev – + prox ►

x

Una red Neuronal 'Recurrente'

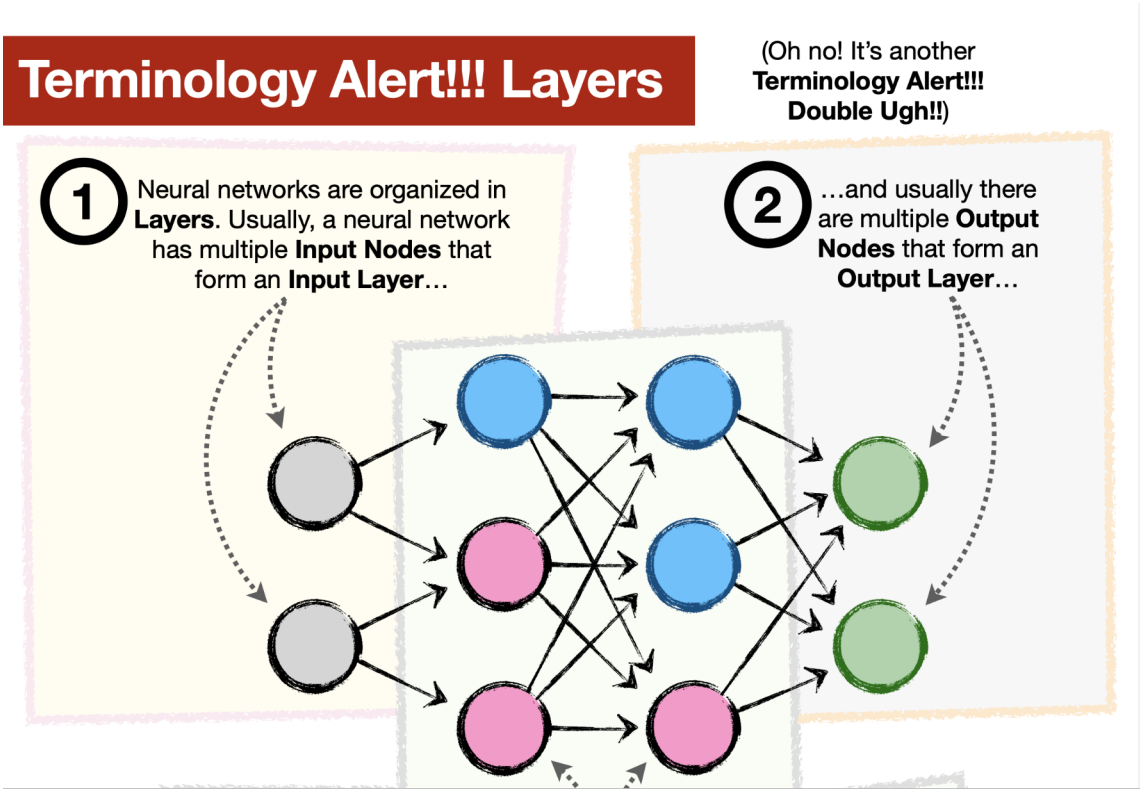


An 'unrolled' Recurrent Neural Network (RNN)

Out[ ]=

◄ prev – | + prox ►

Out[•]=



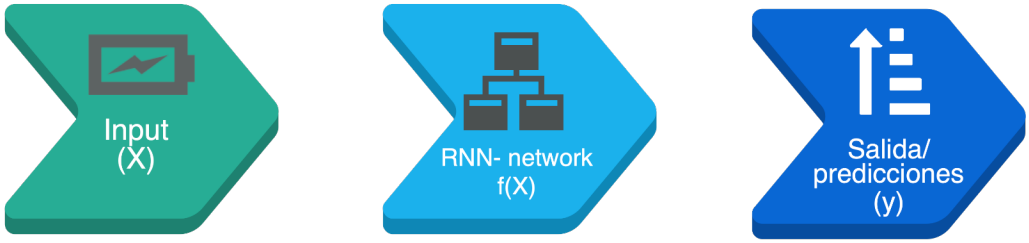
Out[•]=

◄ prev – | + prox ►

The best thing about AI is its ability to ____

Modelo Básico 101

x



Datos a Modelar

Necesitamos elegir los datos de donde queremos sacar nuestro modelo.

Celdas RNN

Caja negra que va a procesar la entrada: una función!

Predicciones (y)

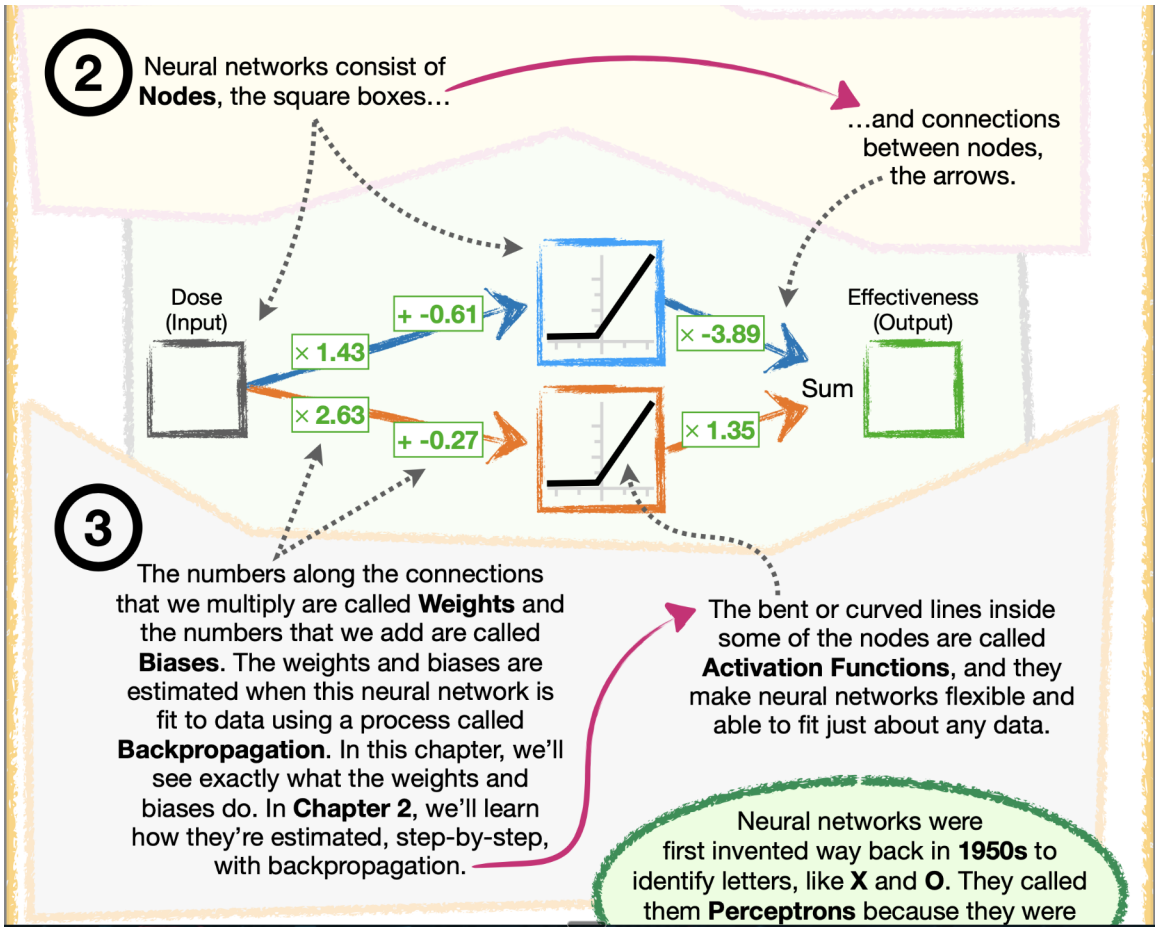
A una caja negra le inyectas una entrada, esta la procesa y sale algo!

Transformers: https://www.jeremyjordan.me/transformer-architecture/

Out[•]=

◄ prev – | + prox ►

Out[•]=



Out[•]=

◂ prev – | + prox ▸

In[•]:=
```
model =
 NetModel[{"GPT2 Transformer Trained on WebText Data", "Task" → "LanguageModeling"}]
```

Out[•]=
```
NetChain[ Input port: string
          Output port: class ]
```
This object cannot be used as input.

In[•]:=
```
model["The best thing about AI is its ability to", {"TopProbabilities", 5}]
```

Out[•]=
```
{ do → 0.0288513,  understand → 0.0307801,  make → 0.0319072,  predict → 0.0349754,
  learn → 0.0445319}
```

In[•]:=
```
Dataset[ReverseSort[Association[%]], ItemDisplayFunction → (PercentForm[#, 2] &)]
```

Out[•]=

Si aplicamos el modelo

In[•]:=
```
NestList[StringJoin[#, model[#, "Decision"]] &,
 "The best thing about AI is its ability to", 7]
```

Out[•]=
```
{The best thing about AI is its ability to,
 The best thing about AI is its ability to learn,
 The best thing about AI is its ability to learn from,
 The best thing about AI is its ability to learn from experience,
 The best thing about AI is its ability to learn from experience.,
 The best thing about AI is its ability to learn from experience. It,
 The best thing about AI is its ability to learn from experience. It's,
 The best thing about AI is its ability to learn from experience. It's not}
```

- Si seguimos con Temperatura Zero:

...

In[•]:=
```
StringReplace[Nest[StringJoin[#, model[#, "Decision"]] &,
  "The best thing about AI is its ability to", 132], "\n" .. → " "]
```

Out[•]=
```
The best thing about AI is its ability to learn from experience. It's not just a
  matter of learning from experience, it's learning from the world around you. The
  AI is a very good example of this. It's a very good example of how to use AI to
  improve your life. It's a very good example of how to use AI to improve your life.
  The AI is a very good example of how to use AI to improve your life. It's a very
  good example of how to use AI to improve your life. The AI is a very good example
  of how to use AI to improve your life. It's a very good example of how to
```

...

NOT GREAT...

Pero con temperature de 0.8

```
In[•]:= NestList[StringJoin[#, model[#, {"RandomSample", "Temperature" → .8}]] &,
         "The best thing about AI is its ability to", 7]
Out[•]= {The best thing about AI is its ability to,
         The best thing about AI is its ability to provide,
         The best thing about AI is its ability to provide a,
         The best thing about AI is its ability to provide a certain,
         The best thing about AI is its ability to provide a certain level,
         The best thing about AI is its ability to provide a certain level of,
         The best thing about AI is its ability to provide a certain level of privacy,
         The best thing about AI is its ability to provide a certain level of privacy when}
```

... Mira como queda

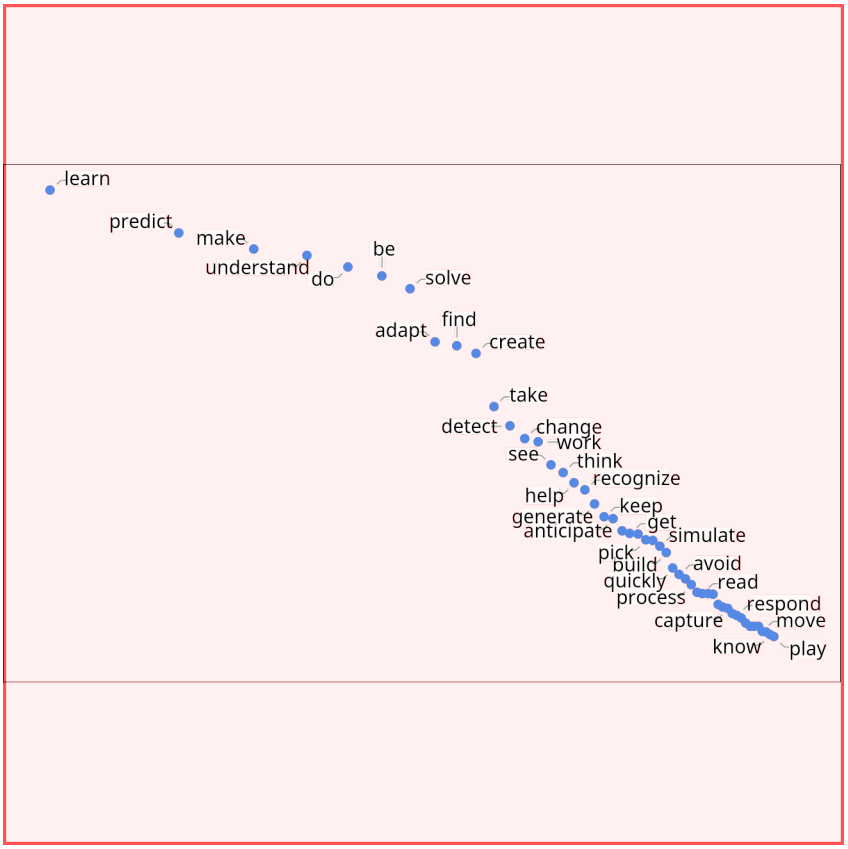
```
In[•]:= Table[Nest[StringJoin[#, model[#, {"RandomSample", "Temperature" → .8}]] &,
           "The best thing about AI is its ability to", 7], 5]
Out[•]= {The best thing about AI is its ability to be able to estate    ly more,
         The best thing about AI is its ability to understand, to tell you which functions,
         The best thing about AI is its ability to apply usually significant cognitive
           procedures to complex,
         The best thing about AI is its ability to learn. That is the only thing,
         The best thing about AI is its ability to solve problems in the natural world."}
```

- n^(-1) “Power law”

...

• • •

In[•]:=
```
ListLogLogPlot[
 Take[ReverseSort[model["The best thing
      about AI is its ability to",
    "Probabilities"]], 50],
 ImageSize → {700, 700},
 FrameStyle →
  Directive[FontFamily → "Times"],
 LabelStyle → Directive[Black, 16],
 Frame → True, PlotStyle → Thick]
```

Out[•]=


Un poco mejor que el caso de temperatura ZERO (GPT-2):

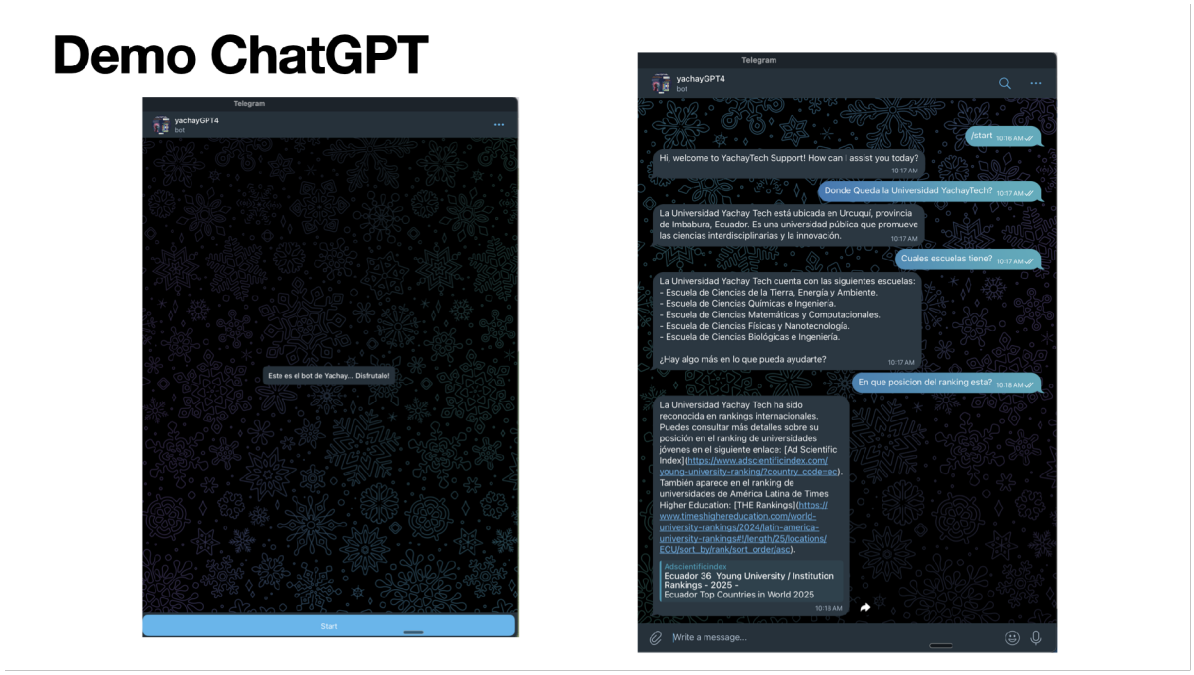
In[•]:=
```
StringReplace[Nest[StringJoin[#, model[#, {"RandomSample", "Temperature" → .8}]] &,
  "The best thing about AI is its ability to", 100], "\n" .. → " "]
```

Out[•]=
```
The best thing about AI is its ability to find its way around and work so much
  harder to understand the realm. The ability to do so is also what allows it to
  know the ten most common problems in your business (or, better yet, beyond it),
  and it has many other skills of its own staff. So you'll need to be extremely
  worrying about your AI to actually learn to interact with it. And in martyrdom,
  it does tend to be a bit cruel: you'll need an AI that doesn't want to
```

- Con GPT-4 (**@Telegram**. Pedirle a la audiencia que pueda, usar el bot)

...

Demo: https://t.me/yachayGPT4bot



¿De donde provienen esas probabilidades?

Un problema mucho mas sencillo!
solo contemos las letras de un artículo en Wikipedia...

```
Articulos con "cats"
```

Contemos en Wikipedia

In[•]:=
```
LetterCounts[WikipediaData["cats"]]
```

Out[•]=
<|e → 4345, a → 3485, t → 3411, i → 2818, s → 2632, n → 2530, o → 2476, r → 2180, h → 1607, l → 1598, c → 1433, d → 1346, m → 1023, u → 932, g → 768, f → 743, p → 664, y → 618, b → 515, w → 475, v → 403, k → 196, T → 121, x → 94, C → 82, A → 76, I → 68, S → 59, F → 43, z → 42, H → 35, E → 33, D → 30, M → 29, B → 29, N → 27, q → 23, P → 21, G → 20, j → 19, L → 17, O → 17, U → 16, R → 15, W → 13, K → 11, J → 7, Z → 5, V → 4, Q → 2, í → 2, ä → 2, á → 2, Y → 1, ė → 1, ž → 1, ď → 1, ö → 1, đ → 1, ī → 1, î → 1|>

Con dogs?

```
In[•]:= LetterCounts[WikipediaData["dogs"]]
```

```
Out[•]= <| e → 3908, a → 2736, o → 2606, i → 2556, t → 2521, s → 2399, n → 2336, r → 1864,
  d → 1581, h → 1460, l → 1355, c → 1082, g → 928, m → 858, u → 782, f → 662,
  p → 636, y → 499, b → 462, w → 409, v → 405, k → 151, T → 90, C → 85, I → 80,
  A → 73, x → 71, S → 65, D → 58, B → 47, P → 40, N → 38, z → 36, W → 27, H → 27,
  q → 25, M → 24, L → 24, E → 23, F → 21, O → 21, U → 20, R → 18, G → 17, j → 17,
  K → 10, J → 7, V → 7, Y → 4, é → 2, X → 1, á → 1, ó → 1, ŵ → 1 |>
```

En el Ingles!

```
In[•]:= English LANGUAGE [ character frequencies ]
```

```
Out[•]= {e → 12.7% , t → 9.06% , a → 8.17% , o → 7.51% , i → 6.97% , n → 6.75% ,
  s → 6.33% , h → 6.09% , r → 5.99% , d → 4.25% , l → 4.03% , c → 2.78% ,
  u → 2.76% , m → 2.41% , w → 2.36% , f → 2.23% , g → 2.02% , y → 1.97% ,
  p → 1.93% , b → 1.49% , v → 0.978% , k → 0.772% , j → 0.153% , x → 0.150% ,
  q → 0.0950% , z → 0.0740% }
```

Un ejemplo, si generamos una secuencia con estas probabilidades:

In[•]:=
```
With[{freq =
    Entity["Language", "English::385w8"][
     EntityProperty[
      "Language", "LetterFrequency"]
    ]}, SeedRandom[53 424]; StringJoin[
   RandomChoice[UnitConvert[Values[freq]] →
    Keys[freq], 150]]]
```

Out[•]=
```
rronoitadatcaeaesaotdoysaroiyiinnbantoioestlhddeocneooewceseciselnodrtrdgriscsatsepes ⋱
    dcniouhoetsedeyhedslernevstothindtbmnaohngotannbthrdthtonsipieldn
```

Podemos hacerlo con palabras

veamos

In[•]:=
```
With[{freq = Entity["Language", "English::385w8"][
    EntityProperty["Language", "LetterFrequency"]
    ]}, SeedRandom[34 232]; StringJoin[
   RandomChoice[Append[UnitConvert[Values[freq]
    ], .18] → Append[Keys[freq], " "], 150]]]
```

Out[•]=
```
sd n oeiaim satnwhoo eer rtr ofiianordrenapwokom del oaas ill e h f
    rellptohltvoettseodtrncilntehtotrkthrslo hdaol n sriaefr hthehtn ld gpod a  h y oi
```

Forzarlas a parecerse al Ingles

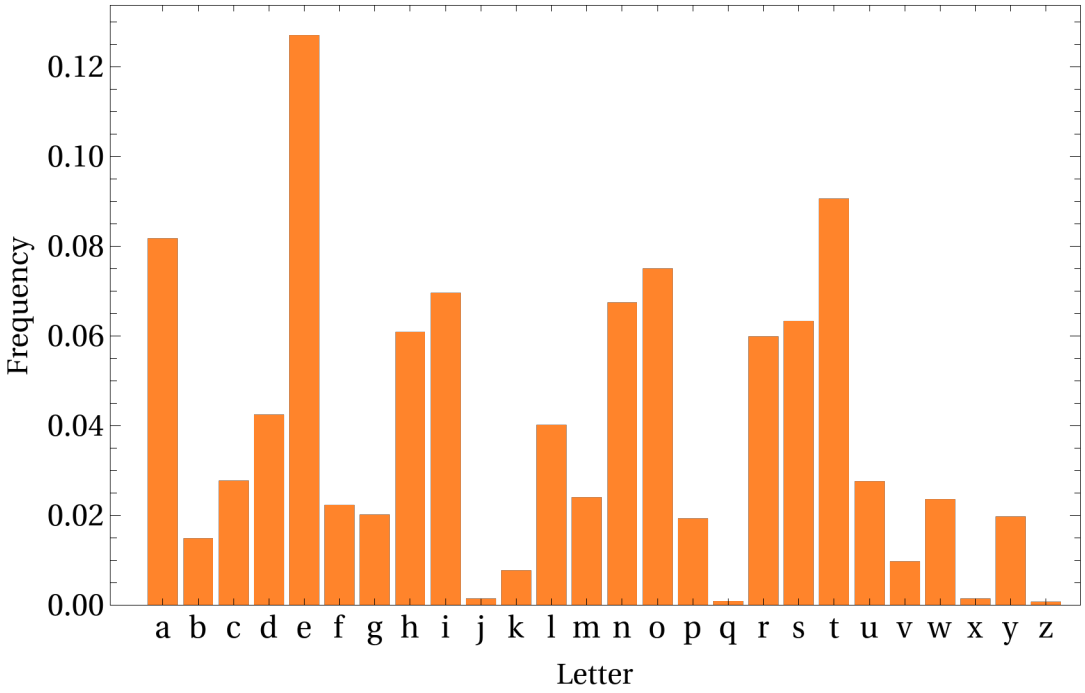
In[•]:=
```
SeedRandom[23 424]; StringJoin[Riffle[Table[
   RandomLettersWord[0], {25}], " "]]
```

Out[•]=
```
ni hilwhuei kjtn isjd erogofnr n rwhwfao rcuw lis fahte uss cpnc nlu oe nusaetat
    llfo oeme rrhrtn xdses ohm oa tne ebedcon oarvthv ist
```

Las Probabilidades

In[•]:=
```
BarChart[Normal[Values[KeySort@Entity["Language", "English::385w8"][
     EntityProperty["Language", "LetterFrequency"]
    ]]], ChartLabels → Alphabet[], Frame → True,
 FrameLabel → {{Style["Frequency", FontSize → 24], None},
   {Style["Letter", FontSize → 24], None}},
 FrameTicksStyle → Directive[FontSize → 26],
 ChartStyle → Lookup[ChatTechColors["Data"], "Orange"],
 ChartBaseStyle → EdgeForm[], ImageSize → Large]
```

Out[•]=



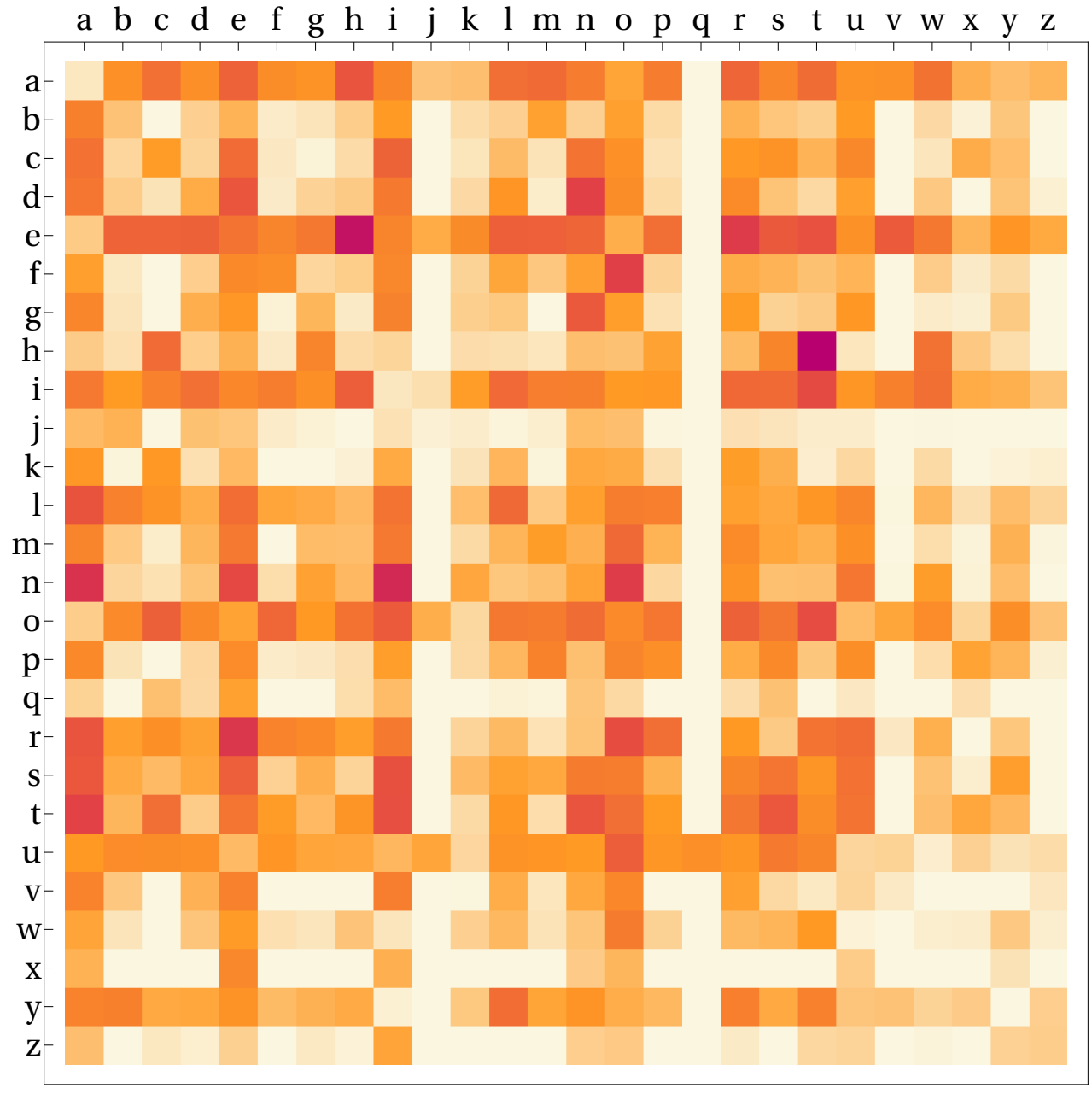
La probabilidad de Pares

...

$$2-\text{grams}\left(\text{nombre de la teoría}\right)$$

In[•]:=
```
MatrixPlot[Transpose@With[{data = ‹…›},
  Outer[Lookup[Lookup[data, Key@{#1}], Key@{#2}, 0] &, Alphabet[], Alphabet[]]]
 FrameTicks → {{#, None}, {None, #}} &[Table[{i, FromLetterNumber[i]}, {i, 26}]],
 FrameTicksStyle → Directive[FontSize → 30],
 ColorFunction → ChatTechColors["Orange"], ImageSize → Full]
```

Out[•]=



Veamos que la q es ZERO excepto para la "u" !!! .. interesante.

Usando 2-grams

In[•]:=
```
SeedRandom[23 424]; StringJoin[Riffle[Table[
  RandomLettersWord[1],
 {25}], " "]]
```

Out[•]=
```
olur weatmai froginda wha rcothe vo os olev cit ato peme trito refla arn edon
  parutast cuchrhedar ts we tas pof ten sath stik tea
```

Y con n-grams:

In[•]:=
```
SeedRandom[234 098 234];
Grid[Table[
{Style[Text@n, Lighter@Gray],
 StringJoin[Riffle[
  Table[RandomLettersWord[n],
  {12}], " "]]}, {n, 0, 5}],
 FrameStyle → LightGray,
 Frame → All, Alignment → Left,
 Spacings → {1, 0.5}]
```

Out[•]=

0	on gxeeetowmt tsifhy ah aufnsoc ior oia itlt bnc tu ih uls
1	ri io os ot timumumoi gymyestit ate bshe abol viowr wotybeat mecho
2	wore hi usinallistin hia ale warou pothe of premetra bect upo pr
3	qual musin was witherins wil por vie surgedygua was suchinguary outheydays theresist
4	stud made yello adenced through theirs from cent intous wherefo proteined screa
5	special average vocab consumer market prepara injury trade consa usually speci utility

Con millones de palabras

In[•]:=
```
SeedRandom[2134]; StringJoin[Riffle[Table[
  WeightedRandomWord[], {30}], " "]]
```

Out[•]=
```
from not emergency tree of is its are and their the rose and default cold statuesque
  had worth director orientation have possibly one the the used be said that little
```

No tiene sentido!!!

Con n-grams, empezando por la palabra cats:

...

In[•]:=
```
Column[{"cat through shipping variety is made the aid emergency can the",
  "cat for the book flip was generally decided to design of",
  "cat at safety to contain the vicinity coupled between electric public",
  "cat throughout in a confirmation procedure and two were difficult music",
  "cat on the theory an already from a representation before a"},
 Spacings → 1, Dividers → {None, Thread[Range[2, 5] → LightGray]}]
```

Out[•]=

cat through shipping variety is made the aid emergency can the
cat for the book flip was generally decided to design of
cat at safety to contain the vicinity coupled between electric public
cat throughout in a confirmation procedure and two were difficult music
cat on the theory an already from a representation before a

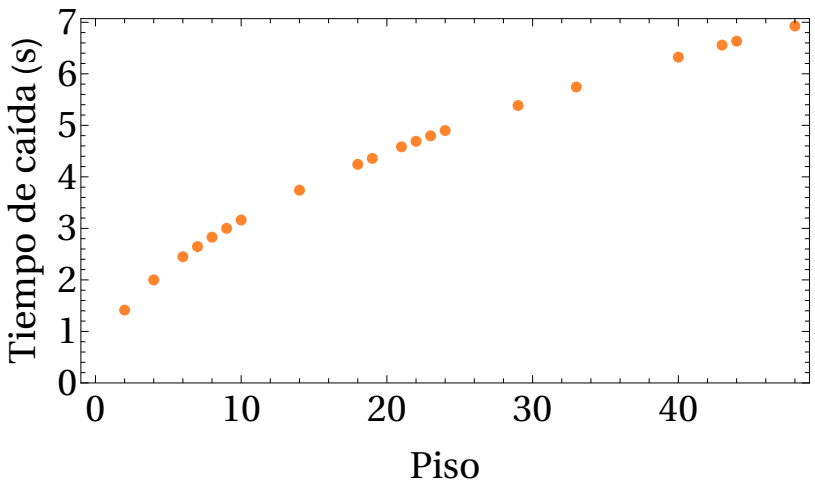
¿Que es un modelo?

Como Galileo, cuanto dura la caída de una bola desde la torre de Pizza?

Tenemos estos datos

In[•]:=
```
SeedRandom[34 535]; ListPlot[{#, Sqrt[#]} & /@ Sort[
RandomSample[Range[50], 20]], Frame → True, PlotRange → {0, Sqrt[50]},
FrameLabel → {{Style["Tiempo de caída (s)", FontSize → 30], None},
{Style["Piso", FontSize → 30], None}},
FrameTicksStyle → Directive[FontSize → 30],
PlotStyle → {ChatTechColors["Data"]["Orange"],
PointSize[.015]}, AspectRatio → 1/2]
```

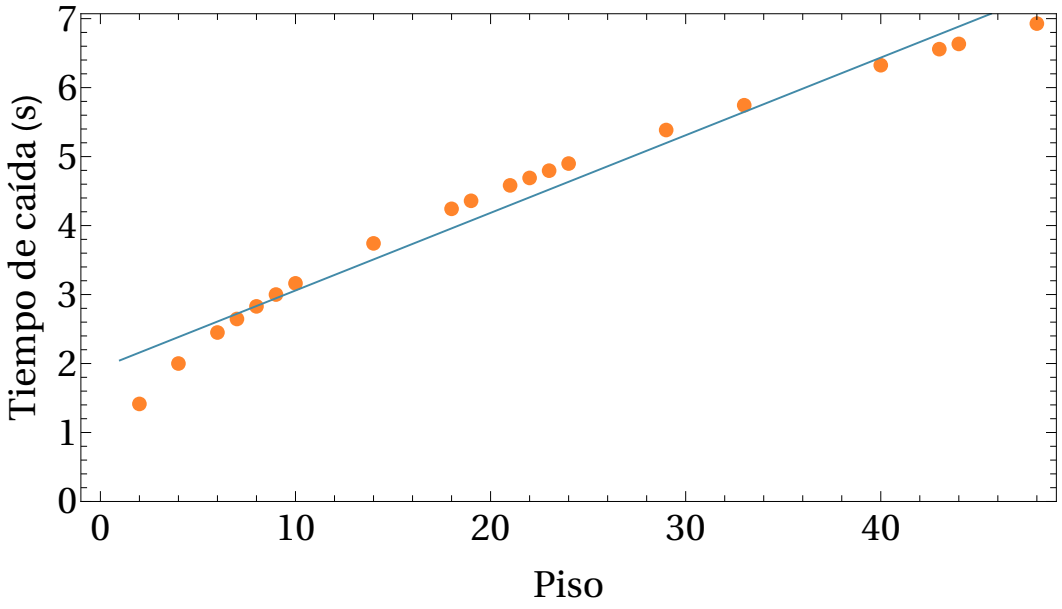
Out[•]=



Simplifiquemos:

```
With[{data = (SeedRandom[34 535];
{#, Sqrt[#]} & /@ Sort[RandomSample[Range[50], 20]])},
Show[ListPlot[data, Frame → True,
FrameLabel → {{Style["Tiempo de caída (s)", FontSize → 30], None},
{Style["Piso", FontSize → 30], None}},
FrameTicksStyle → Directive[FontSize → 30],
PlotRange → {0, Sqrt[50]}, PlotStyle → {ChatTechColors["Data"]["Orange"],
PointSize[.015]}, AspectRatio → 1/2],

Plot[Evaluate[Fit[data, {1, x}, x]], {x, 1, 50},

PlotRange → {0, Sqrt[50]},
PlotStyle → ChatTechColors["Data"]["Teal"]]]]
```

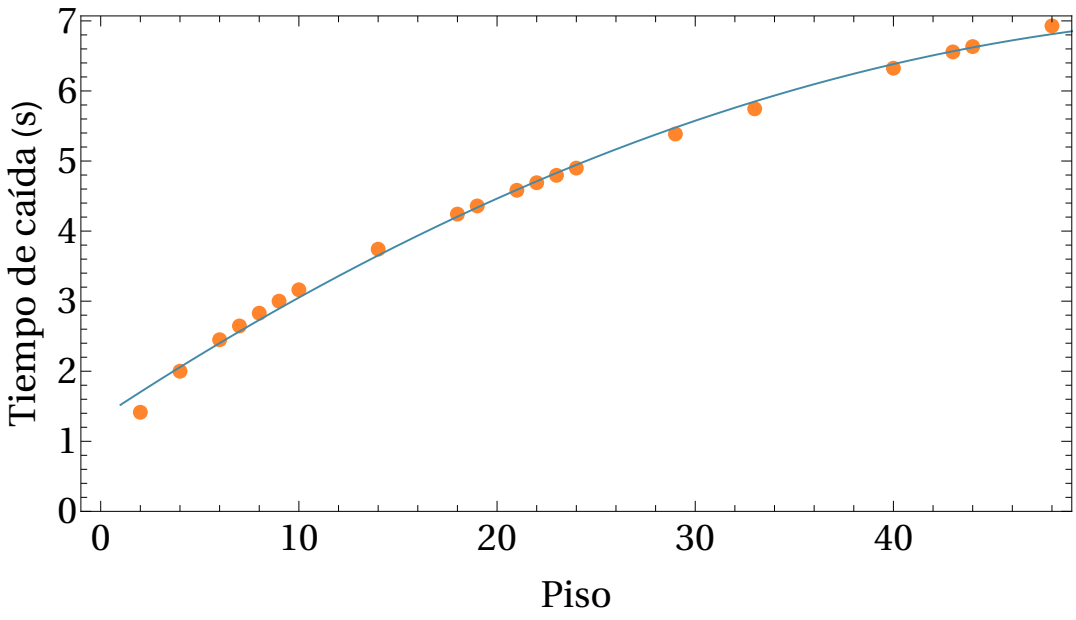
Out[•]=



Ajustamos!!

```
With[{data = (SeedRandom[34 535];
    {#, Sqrt[#]} & /@ Sort[RandomSample[Range[50], 20]])},
 Show[ListPlot[data, Frame → True,
   FrameLabel → {{Style["Tiempo de caída (s)", FontSize → 30], None},
     {Style["Piso", FontSize → 30], None}},
   FrameTicksStyle → Directive[FontSize → 30],
   PlotRange → {0, Sqrt[50]}, PlotStyle → {ChatTechColors["Data"]["Orange"],
     PointSize[.015]}, AspectRatio → 1/2],

  Plot[Evaluate[Fit[data, {1, x, x^2}, x]], {x, 1, 50},

   PlotRange → {0, Sqrt[50]},
   PlotStyle → ChatTechColors["Data"]["Teal"]
  ]]]
```

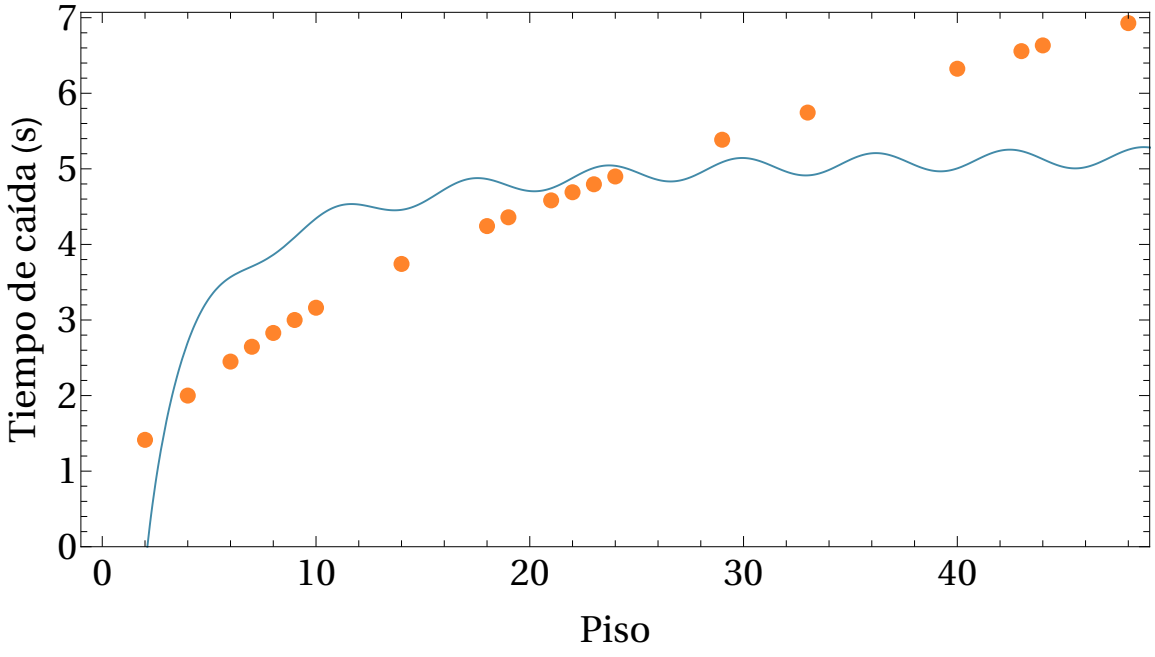
Out[-]=



Un mal modelo

```
With[{data = (SeedRandom[34 535];
    {#, Sqrt[#]} & /@ Sort[RandomSample[Range[50], 20]])},
 Show[ListPlot[data, Frame → True,
   FrameLabel → {{Style["Tiempo de caída (s)", FontSize → 30], None},
     {Style["Piso", FontSize → 30], None}},
   FrameTicksStyle → Directive[FontSize → 30],
   PlotRange → {0, Sqrt[50]}, PlotStyle → {ChatTechColors["Data"]["Orange"],
     PointSize[.015]}, AspectRatio → 1/2],

  Plot[Evaluate[(a + b/x + c Sin[x]) /. FindFit[data, a + b/x + c Sin[x], {a, b, c}, x]],
   {x, 1, 50}, PlotRange → {0, Sqrt[50]},

   PlotStyle → ChatTechColors["Data"]["Teal"]
  ]]]
```

Out[-]=



ChatGPT lo hace con 175 — millardos de Parametros

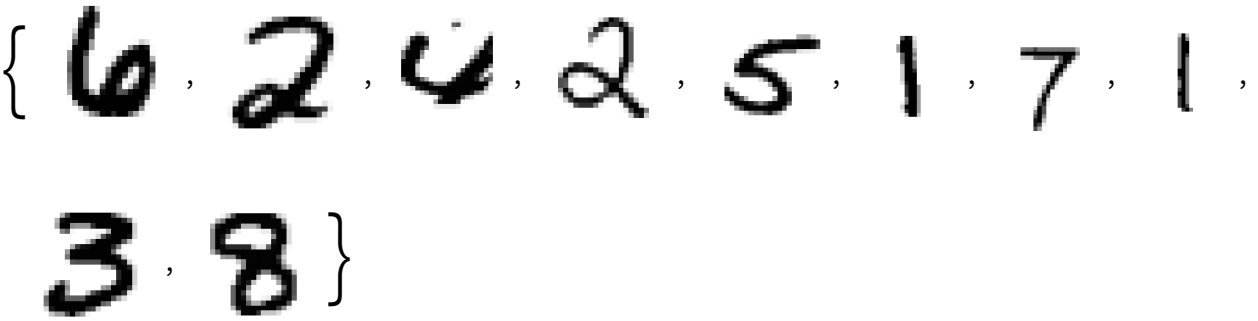
Modelos para tareas muy HUMANAS!

Modelo de como funciona el Cerebro?

Comencemos por una tarea sencilla : "Reconocimiento de imagenes"

In[•]:=
```
First /@ RandomSample[ResourceData["MNIST"], 10]
```

Out[•]=



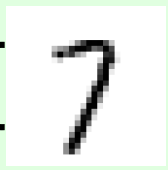
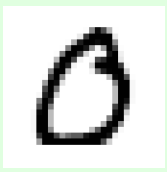
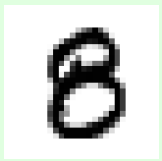
Para cada imagen:

Como humanos pordemos reconocer esto!

¿Podemos construir un modelo que sepa a que numero representa una imagen?

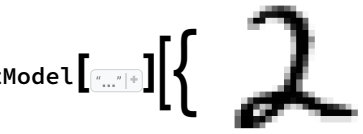
Si!

Tratemos a la función como una caja negra:

In[•]:=
```
Style[
 NetModel["..."][{  ,  ,  ,
   ,  ,  ,  , 1, 1, 1}]
 , 70]
```

Out[•]=
```
{7, 0, 9, 7, 8, 2, 4, 1, 1,
 1}
```

Que ocurre?: Vamos a progresivamente BLUR las imagenes

In[•]:=
```
Style[
  NetModel["..."][{  ,  ,
   }]
  , 70]
```

Out[•]=
```
{2, 2, 2, 1, 1, 1, 1, 1, 1}
```

–– Redes Neuronales ––

1940 - Como funciona el cerebro!

En el cerebro, despues de que lo fotones chocan el ojo:

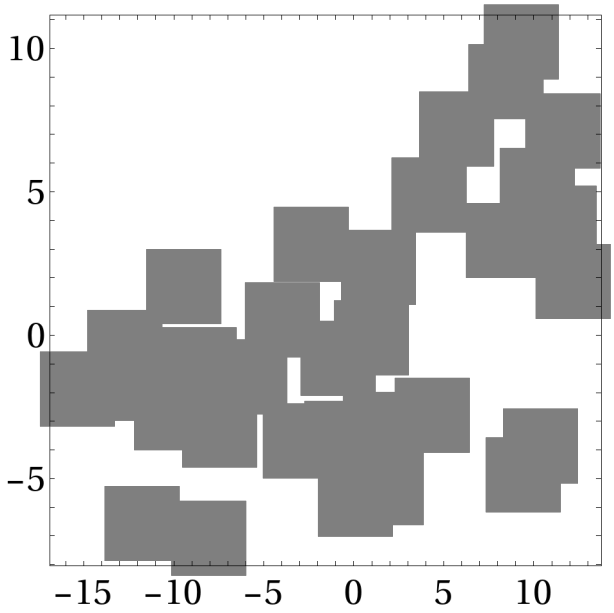
```
In[•]:= NetModel["LeNet Trained on MNIST Data"]
```

```
Out[•]= NetChain[  ]
```

No hay teoria. Solo funcionó en 1998!! -- Noción del atractor: queremos que los 1 esten juntos..

In[•]:=
```
FeatureSpacePlot[Catenate[Cases[ResourceData["MNIST"],
    (x_ → #) :> (x → SetAlphaChannel[x, ColorNegate[x]]), {1}, 16] & /@ {1, 2}],
 RandomSeeding → 5, Frame → True, FrameTicksStyle → Directive[FontSize → 30]]
```

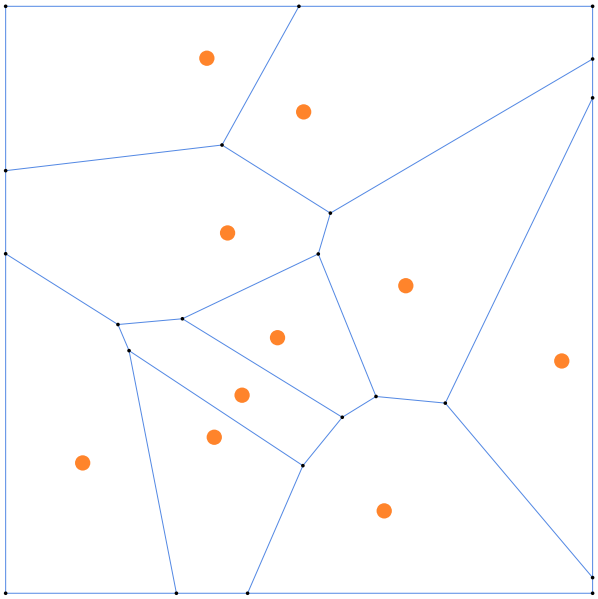
Out[•]=



Imagine que ...

In[•]:=
```
SeedRandom[3 423 424];
With[{pts = RandomReal[{-.9, .9}, {10, 2}]},
 Show[VoronoiMesh[pts, {{-1, 1}, {-1, 1}}],
   PlotTheme → "Lines", BaseStyle → GrayLevel[.7]],
  Graphics[Style[Point[pts], PointSize[.025],
    ChatTechColors["Data"]["Orange"]
   ]]]
```

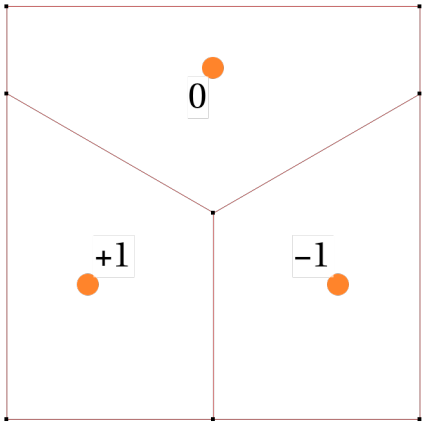
Out[•]=



Como hacemos de una red neuronal realice la tarea de “Reconocer imagenes” ?

In[•]:=
```
SeedRandom[3 423 424];
With[{pts = .7 CirclePoints[3]},
 Show[VoronoiMesh[pts, {{-1, 1}, {-1, 1}}, PlotTheme → "Lines",
   BaseStyle → GrayLevel[.7]], ListPlot[MapIndexed[Labeled[#, Style[
       Switch[First[#2], 1, Style["-1", 30], 2, Style["0", 30], 3, Style["+1", 30]
       ], Black]] &, pts], PlotStyle → Directive[PointSize[.05],
     ChatTechColors["Data"]["Orange"]]]]]
```

Out[•]=

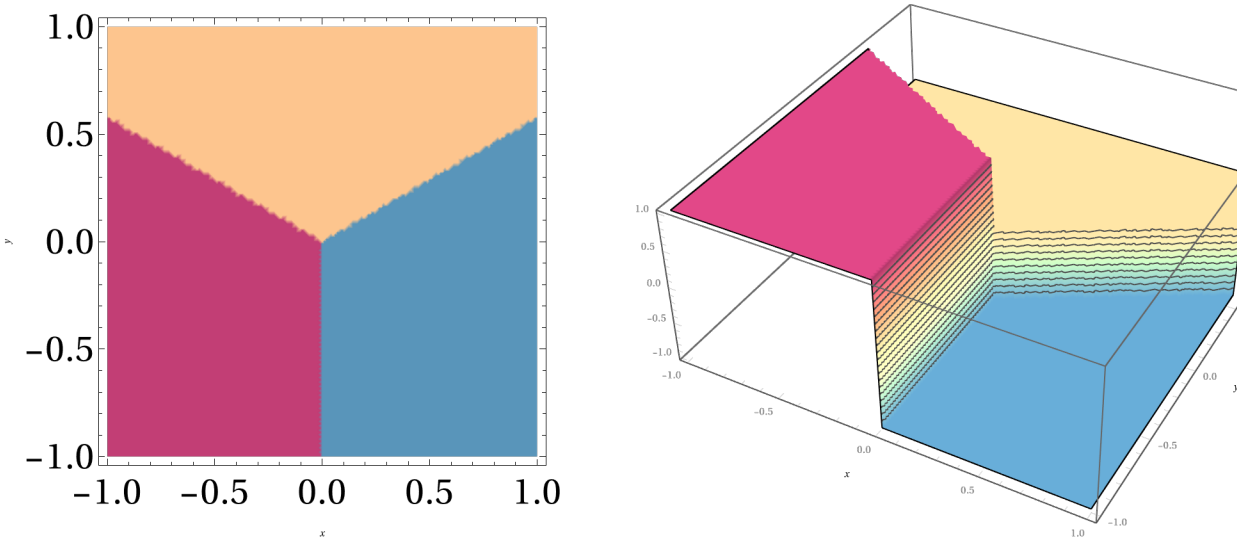


- Dado {x,y} en donde cae?

Una función asi:

In[•]:=
```
With[{nf = Nearest[Thread[.7 CirclePoints[3] → {-1, 0, 1}]]},
 GraphicsRow[{
   DensityPlot[nf[{x, y}], {x, -1, 1}, {y, -1, 1},
    ColorFunction → Function[ ChatTechColors["Normal"][2 # - 1]],
    PlotPoints → 50, FrameTicksStyle → Directive[FontSize → 30], FrameLabel → (Style[#, Italic] & /@ {"x", "y"}), ImageSize → 310],
   Plot3D[nf[{x, y}], {x, -1, 1}, {y, -1, 1}, PlotPoints → 50, MeshFunctions → {# 3 &}, Mesh → 20,
    TicksStyle → Gray,
    ColorFunction → Function[ ChatTechColors["Normal"][2 # 3 - 1]], SphericalRegion → False, AxesLabel → (Style[#, Italic] & /@ {"x", "y"})],
   BarLegend[{Function[ ChatTechColors["Normal"][#]], {-1, 1}}, LegendMarkerSize → 150]
  }, Spacings → {25, 5}]]
```

Out[•]=



Como?

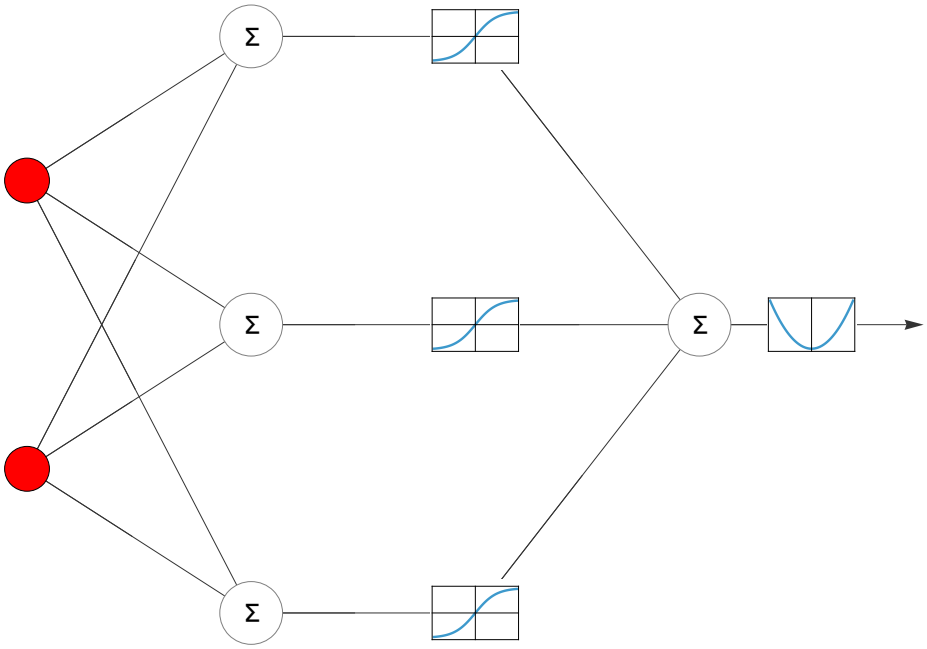
In[•]:=
```
Clear[FormulaNeuralNetworkGraph]
FormulaNeuralNetworkGraph[layerCounts : {_Integer, _Integer, _Integer}] :=
  Block[{gr1, gr2, gr21, gr3, gr4, gr, bc},
   gr1 = IndexGraph[CompleteGraph[Take[layerCounts, 2]]];
   gr2 =
    Graph[Map[(layerCounts[[1]] + #) ⟞ (layerCounts[[1]] + layerCounts[[2]] + #) &,
      Range[layerCounts[[2]]]]];


   gr3 = IndexGraph[CompleteGraph[Take[layerCounts, -2]],
     layerCounts[[1]] + layerCounts[[2]] + 1];
   bc = layerCounts[[1]] + 2 * layerCounts[[2]];


   gr4 = Graph[Map[(bc + #) ⟞ (bc + layerCounts[[3]] + #) &,
      Range[layerCounts[[3]]]], VertexLabels → "Name"];
```

```
    gr = GraphUnion[gr1, gr2, gr3, gr4];


    Graph[gr,
     GraphLayout → {"MultipartiteEmbedding", "VertexPartition" → {layerCounts[[1]],
         layerCounts[[2]], layerCounts[[2]], layerCounts[[3]], layerCounts[[3]]}}]];

Clear[FormulaNeuralNetworkGraphPlot]
Options[FormulaNeuralNetworkGraphPlot] = Options[Graphics];

FormulaNeuralNetworkGraphPlot[
    layerCounts : {_Integer, _Integer, _Integer}, func1_, opts : OptionsPattern[]] :=
  FormulaNeuralNetworkGraphPlot[layerCounts, func1, # &, opts];

FormulaNeuralNetworkGraphPlot[layerCounts : {_Integer, _Integer, _Integer},
    func1_, func2_, opts : OptionsPattern[]] :=
  Block[{plOpts, grFunc1, grFunc2, gr, vNames, vCoords, vNameToCoordsRules, edgeLines},
    plOpts = {PlotTheme → "Default", Axes → True, Ticks → False,
      Frame → True, FrameTicks → False, ImageSize → Small};
    grFunc1 = Plot[func1[x], {x, -2, 2}, Evaluate[plOpts]];
    grFunc2 = Plot[func2[x], {x, -2, 2}, Evaluate[plOpts]];
    gr = FormulaNeuralNetworkGraph[layerCounts];
    vNames = VertexList[gr];
    vCoords = VertexCoordinates /. AbsoluteOptions[gr, VertexCoordinates];
    vNameToCoordsRules = Thread[vNames → vCoords];
    edgeLines = Arrow @ ReplaceAll[List @@@ EdgeList[gr], vNameToCoordsRules];
    Graphics[{Arrowheads[0.02], GrayLevel[0.2], edgeLines, EdgeForm[Black],
      FaceForm[Red], Map[Disk[#, 0.1] &, vCoords[[1 ;; -layerCounts[[-1]] - 1]]],
      Black, Map[{EdgeForm[Gray], FaceForm[White],
        Disk[#, 0.14], Text[Style["∑", 16, Bold], #]} &,
       Join[vCoords[[layerCounts[[1]] + 1 ;; layerCounts[[1]] + layerCounts[[2]]]],
        vCoords[[-2 layerCounts[[-1]] ;; -layerCounts[[-1]] - 1]]], Map[
       {EdgeForm[None], FaceForm[White], Rectangle[# - {0.2, 0.15}, # + {0.2, 0.15}],
        Inset[grFunc1, #1, Center, 0.4]} &, vCoords[[Total[layerCounts[[1 ;; 2]]] + 1
        ;; Total[layerCounts[[1 ;; 2]]] + layerCounts[[2]]]],
      Map[{EdgeForm[None], FaceForm[White], Rectangle[# - {0.2, 0.15},
```

```
#1 + {0.2, 0.15}], Inset[grFunc2, #1, Center, 0.4]} &, MapThread[
 Mean@*List, {vCoords[[-2 layerCounts[[-1]] ;; -layerCounts[[-1]] - 1]],
  vCoords[[-layerCounts[[-1]] ;; -1]]}]}, opts]];
```

In[•]:=
```
FormulaNeuralNetworkGraphPlot[
 {2, 3, 1},
 Tanh, #1^2 &, ImageSize → 800]
```

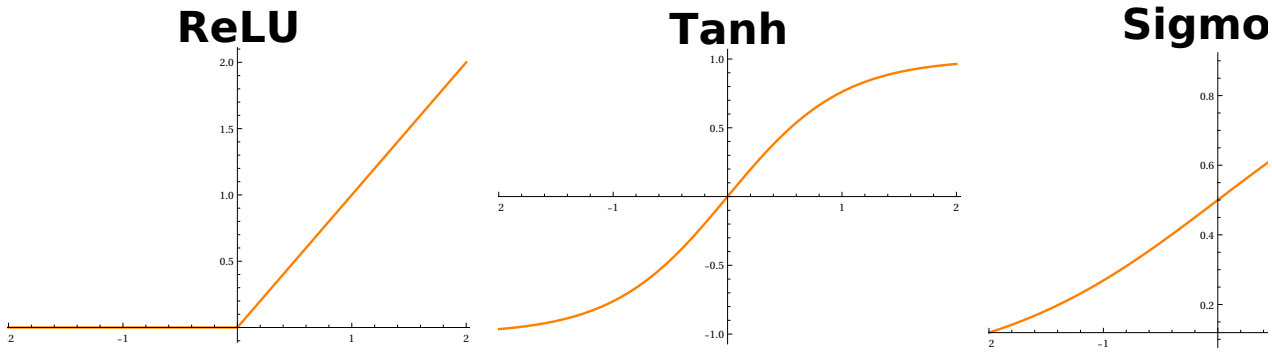
Out[•]=



Funciones de Activación

```
GraphicsRow[
 Labeled[With[{f = ElementwiseLayer[#, "Input" → "Real"]}, Plot[f[x], {x, -2, 2},
       PlotStyle → Directive[Thick, Orange], (*Thicker lines*)
       FrameTicksStyle → Directive[FontSize → 30], ImageSize → 400]],
    Style[#, FontSize → 36, Bold], Top] & /@ {"ReLU", Tanh, "Sigmoid"},
 ImageSize → Large]
```

Out[•]=



Una función matematica cualquiera:

In[•]:=
```
NetToFunction[PointRankData2D["Result431"], {x, y}, {w, b}][[-1, 1]] /. Ramp → f // TraditionalForm
```

Out[•]//TraditionalForm=

$$w_{511}\, f\left(w_{311}\, f\left(b_{11}+x\, w_{111}+y\, w_{112}\right)+w_{312}\, f\left(b_{12}+x\, w_{121}+y\, w_{122}\right)+w_{313}\, f\left(b_{13}+x\, w_{131}+y\, w_{132}\right)+ w_{314}\, f\left(b_{14}+x\, w_{141}+y\, w_{142}\right)+b_{31}\right)+ w_{512}\, f\left(w_{321}\, f\left(b_{11}+x\, w_{111}+y\, w_{112}\right)+w_{322}\, f\left(b_{12}+x\, w_{121}+y\, w_{122}\right)+w_{323}\, f\left(b_{13}+x\, w_{131}+y\, w_{132}\right)+ w_{324}\, f\left(b_{14}+x\, w_{141}+y\, w_{142}\right)+b_{32}\right)+ w_{513}\, f\left(w_{331}\, f\left(b_{11}+x\, w_{111}+y\, w_{112}\right)+w_{332}\, f\left(b_{12}+x\, w_{121}+y\, w_{122}\right)+w_{333}\, f\left(b_{13}+x\, w_{131}+y\, w_{132}\right)+ w_{334}\, f\left(b_{14}+x\, w_{141}+y\, w_{142}\right)+b_{33}\right)+b_{51}$$

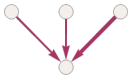
In[•]:=
```
"LayersList"
```

Out[•]=
```
LayersList
```

The neural net of ChatGPT also just corresponds to a mathematical function like this—but effectively with billions of terms.

In[•]:=
```
NetGraphPlot[LinearLayer["Weights" → {{1, 1, 3}}, "Biases" → {1}],
 PlotRange → {{-1.14, 6.8}, Automatic}]
```

Out[•]=



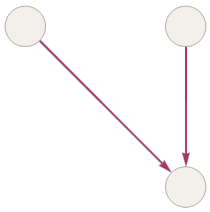
In[•]:=
```
NetChain[{ElementwiseLayer[Ramp], SummationLayer[]}]
```

Out[•]=
```
NetChain[
```

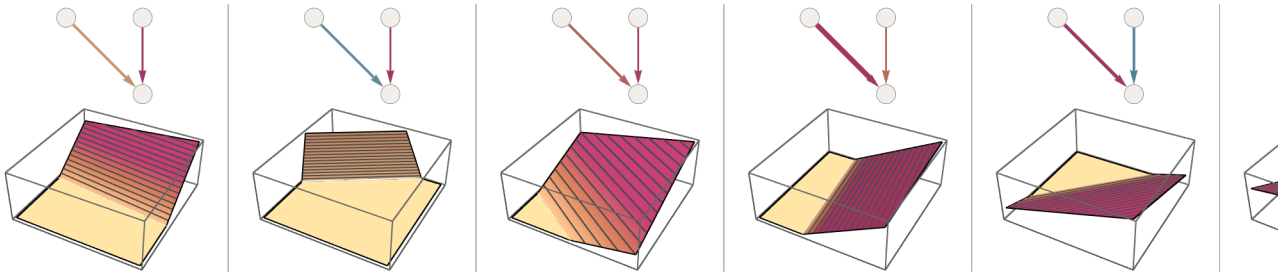
```
]
```

In[•]:=
```
NetGraphPlot[
 LinearLayer["Weights" → {{1, 1}},
  "Biases" → {1}],
 PlotRange → {{-.9, 1.8}, Automatic}]
```

Out[•]=



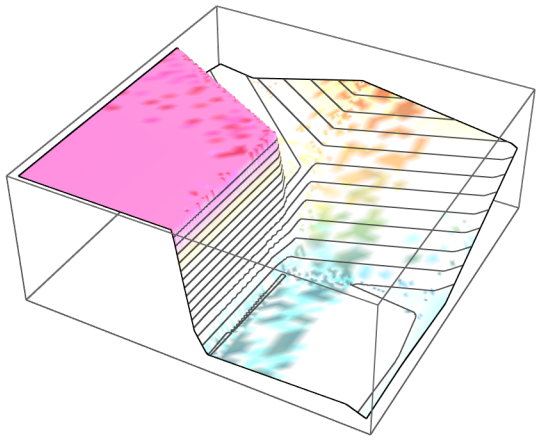
In[•]:=
```
Module[{netelem3D},
 netelem3D[{a_, b_, c_}] :=
  {NetGraphPlot[LinearLayer["Weights" → {{a, b}}, "Biases" → {c}],
    PlotRange → {{-.8, 1.8}, Automatic}],
   Plot3D[Ramp[{x, y}.{a, b} + c], {x, -1, 1}, {y, -1, 1}, PlotPoints → 40,
    ColorFunction → Function[ChatTechColors["Normal"][#3]], Ticks → None,
    MeshFunctions → {#3 &}, ColorFunctionScaling → False, ClippingStyle → None]};
 GraphicsGrid[
  netelem3D[#] & /@ {2 {.1, .8, 0}, 3 {-.3, .5, -.3}, 3 {.2, .3, .3}, 5 {.8, .1, .3},
    4 {.5, -.5, .3}, 2 {.2, -.7, .3}} // Transpose, Spacings → {80, 0},
   ImageSize → 500, Dividers → {{False, {True}, False}}, FrameStyle → LightGray]
 ]
```

Out[•]=


Que hay de redes mas grande? Bueno sale algo asi:

In[•]:=
```
GraphicsRow[{DensityAndContourPlot3D[
  PointRankData2D["Result431"],
 PlotPoints → 40], ImageSize → 500}]
```

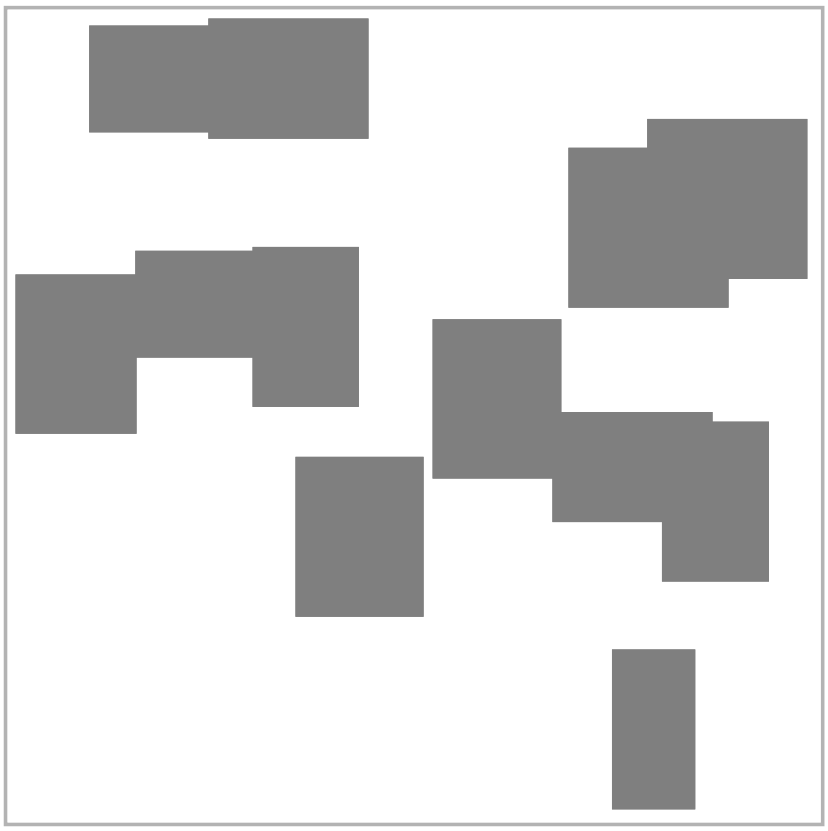
Out[•]=



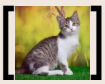
Una tarea mas elaborada.

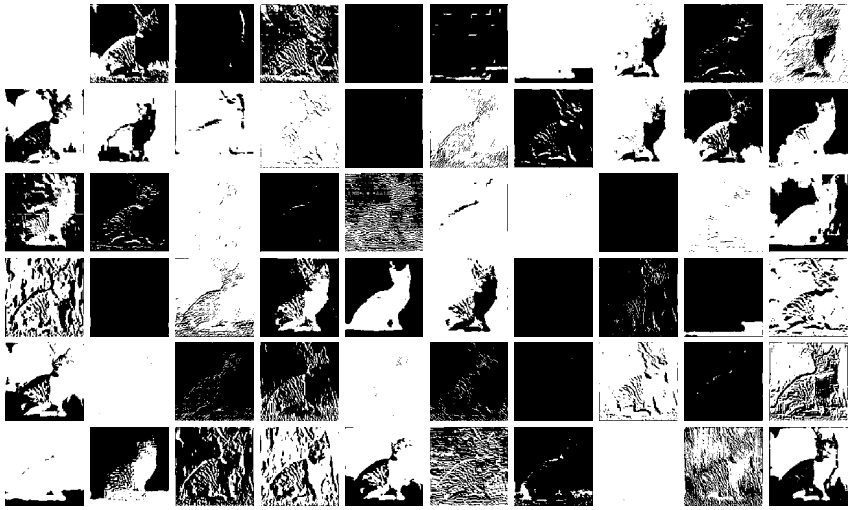
In[•]:=
```
Framed[FeatureSpacePlot[ {…}+ , LabelingSize → 60], FrameStyle → {GrayLevel[0, .3]}]
```

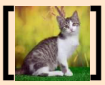
Out[•]=

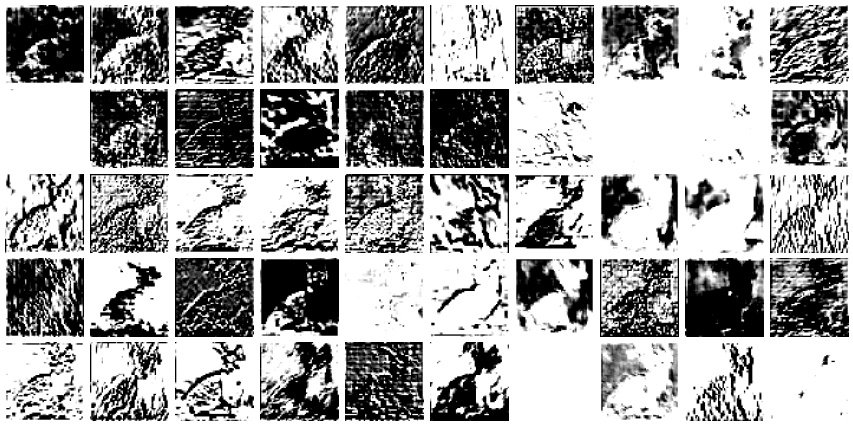


Diferentes capas:

In[•]:=
```
GraphicsGrid[
 Partition[Image /@ Take[NetModel["Wolfram ImageIdentify Net V1"], 1][  ], 10]]
```

Out[•]=


In[•]:=
```
GraphicsGrid[Partition[
  Take[Image /@ Take[NetModel["Wolfram ImageIdentify Net V1"], 10][  ], 50], 10]]
```

Out[•]=


Machine Learning y entrenamiento de redes neuronales:

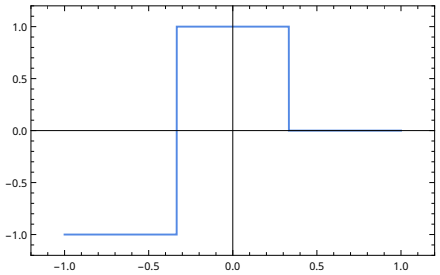
1. Como se entrena una red neuronal?

Lo que queremos es encontrar WEIGHTS! (w)

Vamos a tratar de encontar una red que reproduzca esta función:

In[•]:=
```
Plot[Piecewise[{{-1, x < -1/3},
   {1, -1/3 < x < 1/3}, {0, x > 1/3}}],
 {x, -1, 1}, Exclusions → None,
 Frame → True, PlotRangePadding → .2]
```

Out[•]=



Necesitamos una red que luzca mas o menos asi:

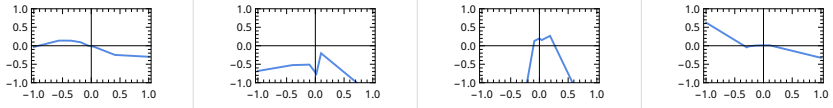
In[•]:=
```
NetGraphPlot[PiecewiseNet[
  {4, 3, 1}],
 EdgeStyle → GrayLevel[.1, 1.],
 "AddArrows" → True,
 ImageSize → 380]
```

Out[•]=

Cuales Weights usamos? :

In[•]:=
```
GraphicsGrid[
 With[{net = NetInitialize[PiecewiseNet[{4, 3, 1}], RandomSeeding → ⌗,
         Method → {"Random", "Biases" → NormalDistribution[0, .2]}]},
      {NetGraphPlot[net, ImageSize → {140}], Plot[net[x], {x, -1, 1}, Frame → True,
        PlotRange → {-1, 1}, ImageSize → {140}]} & /@ {4, 6, 8, 10} // Transpose,
 ImageSize → 700, Alignment → Center, Dividers → {{False, {True}, False}},
 FrameStyle → LightGray, Spacings → {50, -20}]
```

Out[•]=



Vemos que ninguno lo reproduce bien!

Consegimos mejores weights y:

```
(** GraphicsGrid[{CompositeLossPlot[{PiecewiseData1D}],
  ImageSize→600,Dividers→{{False,{True},False}},
  FrameStyle→LightGray,Spacings→{45,Automatic}]**)
```

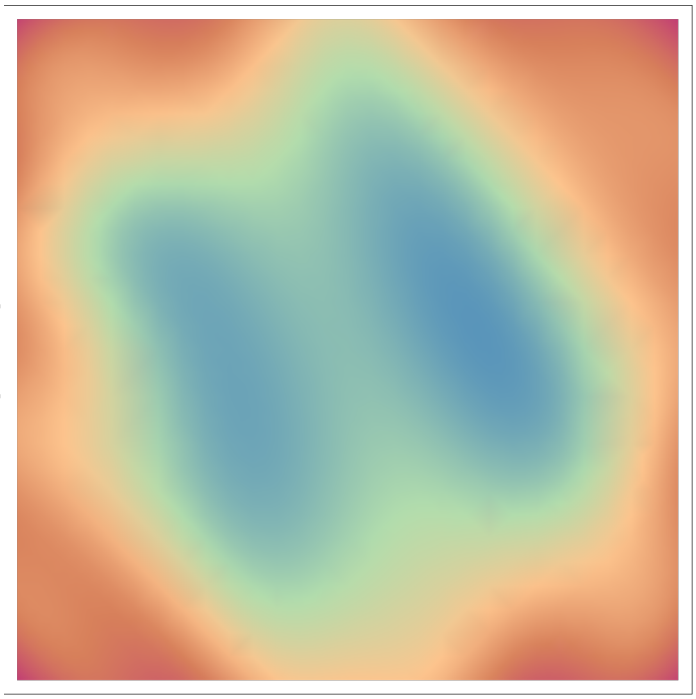
Loss function (L2).

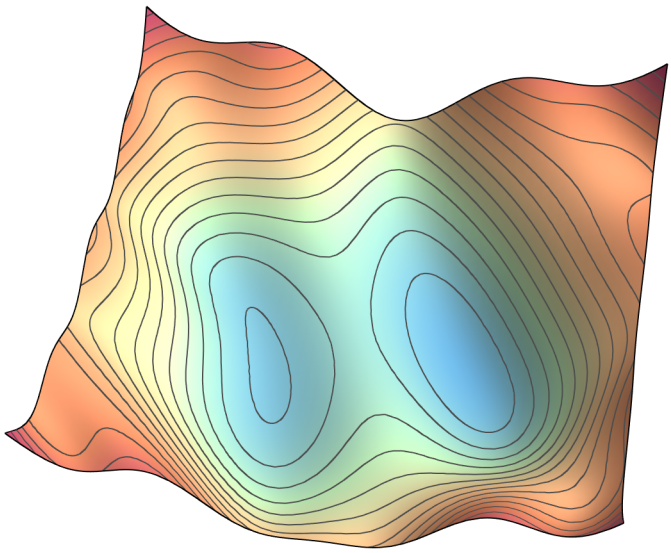
```
GraphicsGrid[{CompositeLossPlot[{PiecewiseData1D}, "ShowLoss" → True], ImageSize → 700,
  Alignment → Center, Dividers → {{False, {True}, False}}, FrameStyle → LightGray, Spacings → {15, -10}}]
```

Como un ejemplo agarremos una función que solo sea función de dos variables w1 y w2. Entonces tenemos una función loss que a su vez es función de w1 y w2 y lucir asi:

```
Module[{bound = Pi, fun},
 fun = x^2 + y^2 + Sin[(x - 1/2) (y - 1/2)] + 2 Sin[2 (x + 1/2) + (y + 1/2)];
 {DensityPlot[fun, {x, -bound, bound}, {y, -bound, bound},
  ColorFunction → Function[
   {CompositeLossPlot}[{PiecewiseData1D}, "ShowLoss" → True], ImageSize → 700,
  Alignment → Center, Dividers → {{False, {True}, False}}, FrameStyle → LightGray, Spacings → {15, -10}]
```

Out[•]=





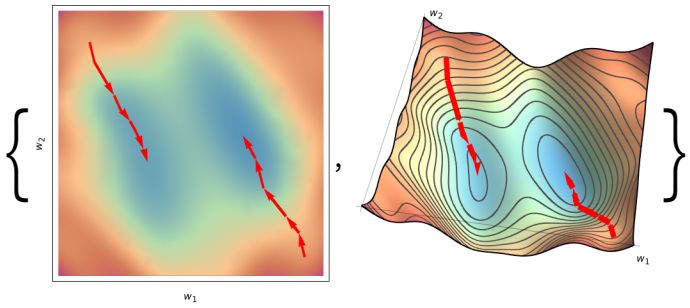
Tecnicas para minimizar:

In[•]:=

```
Module[{bound = Pi, fun, paths, IterateDescent, OverlapSplits},
 OverlapSplits[splits_] := Append[Map[
  Append[First[#], First[Last[#]]] &,
  Partition[splits, 2, 1]], Last[splits]];
```

```
IterateDescent[step_][pt_] := Plus[pt, Times[
    step, ReplaceAll[
     -D[fun, #] & /@ {x, y}, Thread[{x, y} -> pt]]]];
fun = x^2 + y^2 + Sin[(x - 1/2) (y - 1/2)] + 2 Sin[2 (x + 1/2) + (y + 1/2)];
paths = TimeConstrained[FixedPointList[
    IterateDescent[0.12], N@#, 2, Nothing
    ] & /@ {.8 {-3, 3}, .9 {3, -3}};
paths = OverlapSplits[SplitBy[#,
    Floor[EuclideanDistance[#[[1]], #]] &]] & /@ paths;
Show[#, ImageSize -> {Automatic, 250}] & /@
 {Show[DensityPlot[fun, {x, -bound, bound}, {y, -bound, bound},
    FrameLabel -> (Text /@ {"w1", "w2"}),
    ColorFunction -> Function[
     ChatTechColors["Normal"][2 # - 1]], FrameTicks -> None],
    Graphics[{Thick, Red,
      Arrowheads[0.05],
      Map[Arrow[#] &, paths, {2}]}]
   ], Show[Plot3D[fun, {x, -bound, bound}, {y, -bound, bound},
    ColorFunction -> Function[
     ChatTechColors["Normal"][2 #3 - 1]],
    Ticks -> None,
    MeshFunctions -> {#3 &},
    PlotPoints -> 50,
    Axes -> {True, True, False},
    AxesOrigin -> {-Pi, -Pi, 18},
    AxesLabel -> (Text /@ {"w1", "w2"}),
    PerformanceGoal -> "Quality",
    SphericalRegion -> False,
    Boxed -> False,
    ViewPoint -> {0.33, -2.01, 2.70}],
   Graphics3D[{Thick, Red, Arrowheads[.02],
     Map[Arrow[Append[#, (fun + 2) /. Thread[{x, y} -> (#)]
       ] & /@ #] &, paths, {2}]}]
   ]}
]
```

Out[•]=



Podemos usar cálculo:

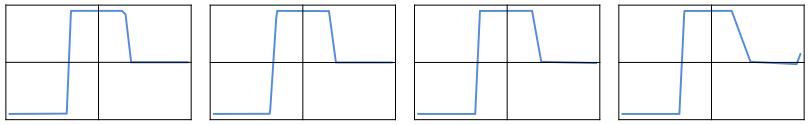
Derivadas para funciones con 175 millardos de párametros!

Diferentes redes dando los mismos resultados:

In[•]=

```
GraphicsRow[Plot[#[x], {x, -1, 1},
    Frame → True,
    FrameTicks → None,
    ImageSize → 100] & /@ PiecewiseData1D["AlternativeTrainings"],
 ImageSize → 700]
```

Out[•]=

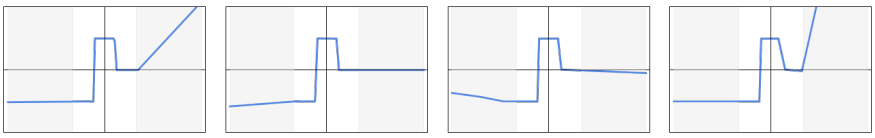


Pero si pedimos valores extrapolados, podemos tener diferentes resultados:

In[•]=

```
GraphicsRow[
 Plot[#[x], {x, -3, 3}, PlotRange → {-2, 2}, Frame → True, FrameTicks → None,
    Prolog → {GrayLevel[.8, .2], Rectangle[{-3, -3}, {-1, 3}],
      Rectangle[{1, -3}, {3, 3}]}, ImageSize → 200] & /@
  PiecewiseData1D["AlternativeTrainings"], ImageSize → 750]
```

Out[•]=



El arte de entrenar una red neuronal -
1) Arquitectura
2) data
3) Se pueden usar redes ya entrenadas y agregar cosas.

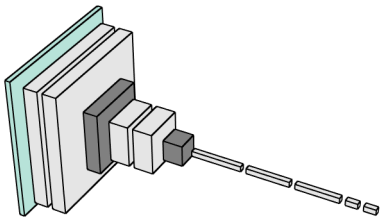
Parece que es mejor end-to-end, dejando descubrir los pasos intermedios etc, Y componentes sencillos

---

Estamos lidiando con arreglo de numeros. Y en el andar podemos rearreglar, cambiar de forma, etc, Por ejemplo en el reconocimiento de digitos empezamos por una imagen 2D y terminamos con output de 1D:

In[•]=
```
NetBlockPlot3D[NetModel["LeNet Trained on MNIST Data"]]
```

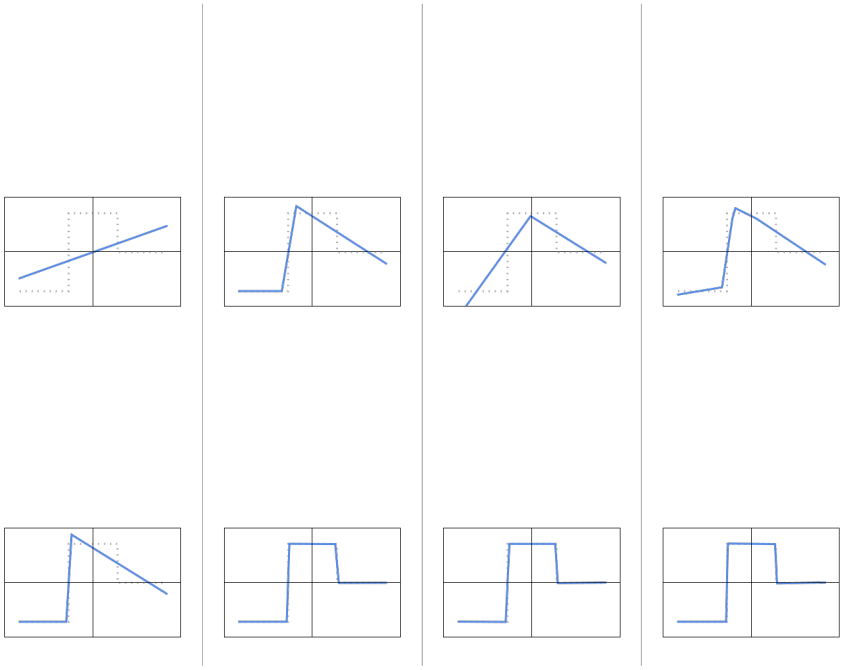
Out[•]=



Tratar de reproducir la función escalon de arriba

In[•]:=
```
GraphicsGrid[Flatten[Transpose /@ Partition[MapThread[
    {NetGraphPlot[#, #2,
       PlotRange → {{-1.5, 1.5}, All}],
      Show[Plot[PiecewiseData1D["TargetFunction"][x], {x, -1, 1}, Frame → True,
        PlotRange → {{-1.2, 1.2}, {-1.4, 1.4}}, FrameTicks → None,
        PlotStyle → Directive[GrayLevel[.3, .5], Dotted], Exclusions → None],
       Plot[#[x], {x, -1, 1}, Frame → True, FrameTicks → None]]} &,
    {PiecewiseData1D["SmallNetsV2"], {{}, {}, {}, {AspectRatio → 1,
       PlotRange → {{-2.5, 1.5}, All}}, {}, {}, {}, {}}}], 4], 1],
 Dividers → {{False, {True}, False}}, FrameStyle →
  LightGray,
 Spacings → {80, -20}, ImageSize →
  700,
 Alignment → {Center, {Bottom, Center}},
 AspectRatio →
  1]
```

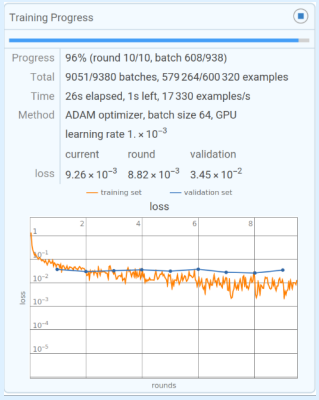
Out[•]=



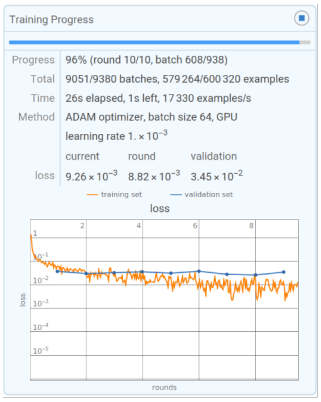
"Transfer Learning" es importante!

Lo importante es ver que la función Loss decrezca!

In[•]:=
```
(* == Loss function == *)
```



Out[•]=



Seguro que una red lo suficientemente grande puede hacer lo que sea!

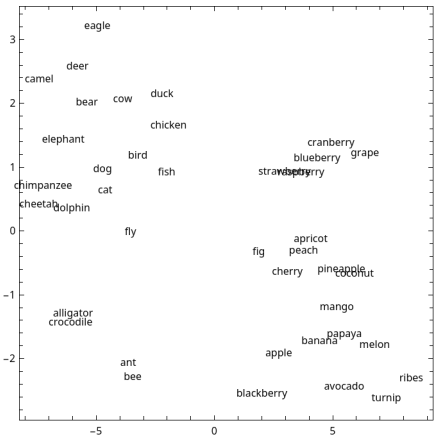
Matematica no trivial es un ejemplo de tarea dificil!

El concepto de Embeddings -

Las redes neuronales como estan concebidas justo ya trabajan con numeros! Y pot lo tanto tenemos que transformar el texto a numero! ... Como un "**meaning space**"

In[•]:=
```
FeatureSpacePlot[
 {"alligator", "ant", "bear", "bee", "bird", "camel", "cat", "cheetah", "chicken",
  "chimpanzee", "cow", "crocodile", "deer", "dog", "dolphin", "duck", "eagle",
  "elephant", "fish", "fly", "apple", "apricot", "avocado", "banana", "blackberry",
  "blueberry", "cherry", "coconut", "cranberry", "grape", "turnip", "mango", "melon",
  "papaya", "peach", "pineapple", "raspberry", "strawberry", "ribes", "fig"},
 FeatureExtractor → NetModel["GloVe 300-Dimensional Word Vectors
  Trained on Wikipedia and Gigaword 5 Data"], Frame → True]
```

Out[•]=



En la tarea de reconocimiento de imagen, tenemos 11 capas:

In[•]:=
```
Information[NetModel["LeNet Trained on MNIST Data"], "SummaryGraphic"]
```

Out[•]=

Por ejemplo le damos un 4:

In[•]:=
```
SoftmaxLayer[][Drop[NetModel["LeNet Trained on MNIST Data"], -1][ ]]
```

Out[•]=
```
{1.42072 × 10^-22, 7.69859 × 10^-14, 1.96532 × 10^-16, 5.55234 × 10^-21, 1., 8.33845 × 10^-14,
 6.89742 × 10^-17, 6.52287 × 10^-19, 6.51468 × 10^-12, 1.9751 × 10^-14}
```

Que pasa si tomamos la salida justo antes del SoftMax?:

Una capa anterior a softmax:

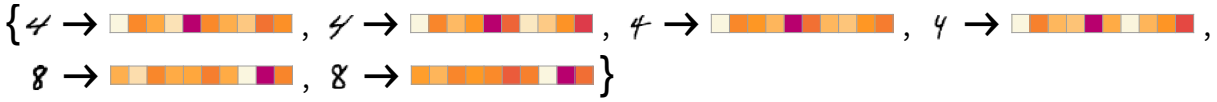
In[•]:=
```
Drop[NetModel["LeNet Trained on MNIST Data"], -1][4]
```

Out[•]=
```
{-26.134, -6.02347, -11.994, -22.4684, 24.1717, -5.94363, -13.0411, -17.7021,
 -1.58528, -7.38389}
```

O sea, un 4 lo podemos represantar como una imagen de una sola fila:

In[•]:=
```
# → ArrayPlot[{Drop[NetModel["LeNet Trained on MNIST Data"], -1][#]}/30,
   ImageSize → 160, Frame → None, Mesh → True,
   ColorFunction → ChatTechColors["OrangeShort"]] & /@ {4, 5, 4, 4, 8, 8}
```

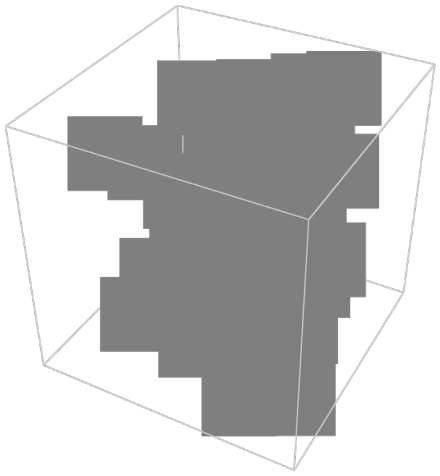
Out[•]=



Tratemos de construir un “espacio de imagen” como el meaning space del embedding para palabras, en 3D

In[•]:=
```
FeatureSpacePlot3D[SeedRandom[123];
 # → SetAlphaChannel[#, ColorNegate[#]] & @@@
  RandomSample[ResourceData["MNIST"], 50],
 FeatureExtractor → NetDrop[NetModel["LeNet Trained on MNIST Data"], -3]]
```

Out[•]=

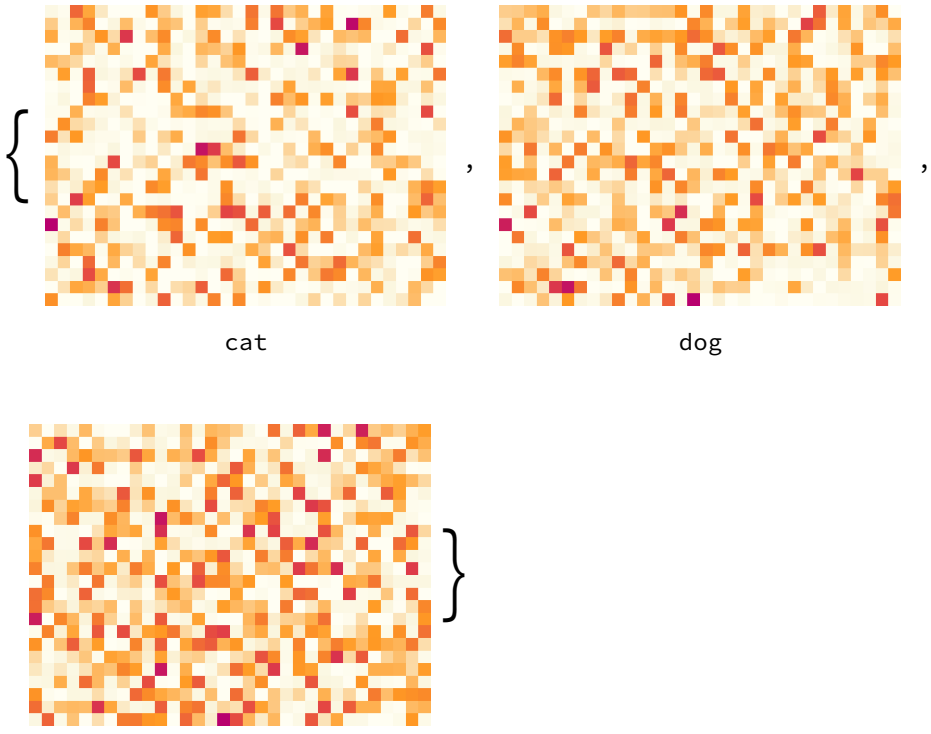


La tarea que nos trajo aquí ! ... “word prediction”. Imagina que tenemos “the ___ cat”.

Como podemos trabajar esa caracterización de las palabras. Por ejemplo:

In[•]:=
```
Labeled[ArrayPlot[Partition[
    First[NetExtract[NetModel["GPT2 Transformer Trained on WebText Data"],
      {"embedding", "embeddingtokens"}][NetExtract[
       NetModel["GPT2 Transformer Trained on WebText Data"], "Input"][[#]]], 32],
   Frame → None, ImageSize → 350, ColorFunction → ChatTechColors["Orange"]],
  #] & /@ {"cat", "dog", "chair"}
```

Out[•]=



Lo bueno es que tenemos una forma de poner a las palabras en una forma AMIGABLE para una red neuroral.

ChatGPT lidia con “tokens”. Unidades de lenguaje.

Adentro de ChatGPT -- Una red GPT-3 con 175 millardos de parametros!

Su elemento de arquitectura mas destacable es una cosa que se llama “Transformer”

Introducen la noción de “Attention” y la idea de prestar mas atención a un sector de la secuencia de token que de las demas partes. Recuerden, agregar el siguiente “token”!

Tres pasos:
1) Agarra el texto y hace el embedding de los tokens.
2) Los trata como lo hace cualquier red neuronal.
3) Genera un arreglo de 50,000 numeros y le asigna probabilidades.

Hay un monton de detalles en la manera que la red esta creada.
(1) El modulo de Embeddings. Aqui GPT-2

```
In[•]:= NetExtract[NetModel["GPT2 Transformer Trained on WebText Data"], "embedding"]
```

```
Out[•]= NetGraph[ Input port:  (range: 1..50257)
                  Output port: (size: n × 768) ]
```

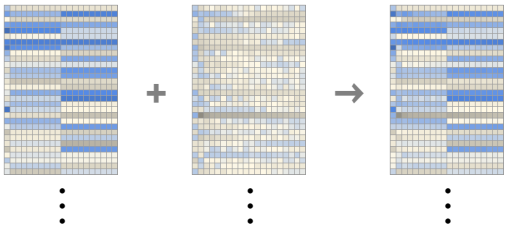
n es el numero de tokens dado como “input”. [x] 768 es la dimensión del arreglo con la que estamos representando cada token.

El **SI** es importante. Es una secuencia de enteros de **POSICIONES para saber donde estaba cada palabra o token. Y de estos enteros sacar otro vector embedding!**

Aqui lo que hace el modulo de Embedding para la secuencia:
“hello my dear how are you hope everything is all right”

In[•]:=
```
Module[
 {matrixPlot},
 matrixPlot[matrix_?MatrixQ] := Labeled[
   MatrixPlot[Take[Transpose@matrix, 30],
    Mesh → True, ImageSize → 100, Frame → None], ":"];
 Row[Riffle[
   matrixPlot /@ Values@NetModel["GPT2 Transformer Trained on WebText Data"][
      "hello hello hello hello hello hello hello hello
        hello hello bye bye bye bye bye bye bye bye bye bye"
      , {NetPort[{"embedding", "embeddingtokens"}],
       NetPort[{"embedding", "embeddingpos"}], NetPort[{"embedding"}]}],
   Style[#, 18, Gray, Bold] & /@ {"+", "→"}], Spacer[5]]
]
```

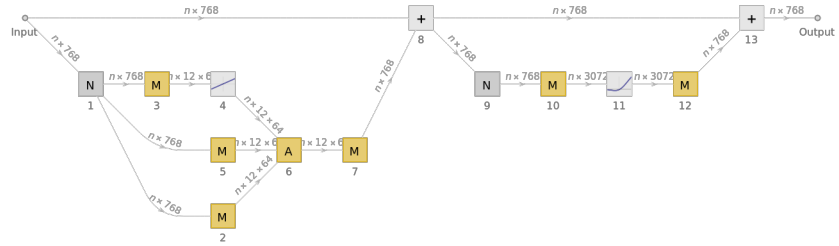
Out[•]=



(2) El bloque de atención (attention)

In[•]:=
```
(* == Un bloque de attention: 12 para GPT−2 y 96 para GPT−3 == *)
Information[NetFlatten@NetExtract[
   NetModel["GPT2 Transformer Trained on WebText Data"],
   {"decoder", 1}], "FullSummaryGraphic"]
```

Out[•]=

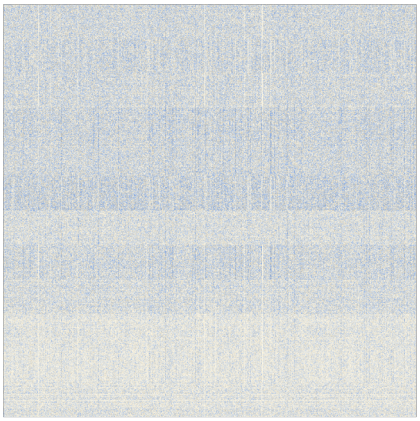


Los 12 attention heads lucen asi:

Despues de procesar: y como (3) lo pasamos por una red complemente conectada!

como una red cualquiera!

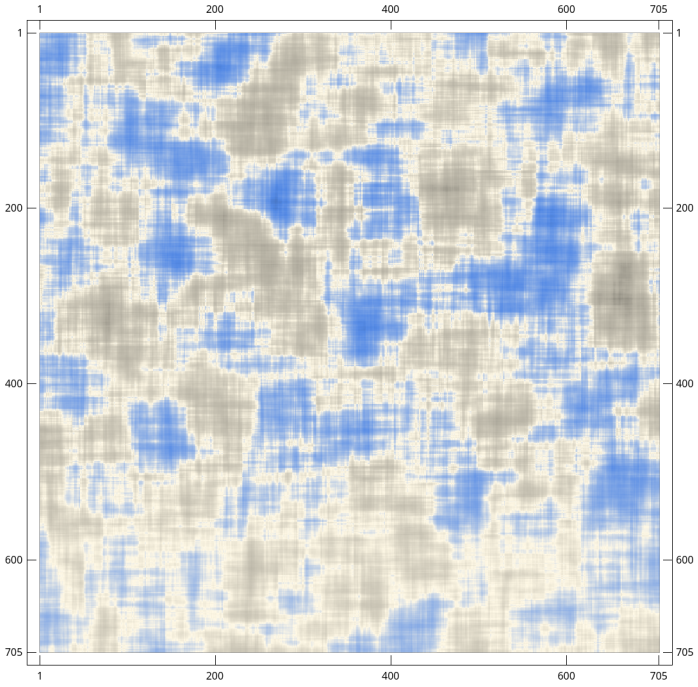
In[•]:=
```
MatrixPlot[Normal@NetExtract[
  NetModel["GPT2 Transformer Trained on WebText Data"],
 {"decoder", 1, "1", "attention", 1(*key*), "Net", "Weights"}],
 Frame → False, FrameTicks → None]
```

Out[•]=


Tomando un average que se mueve 64x64:

In[•]:=
```
MatrixPlot[
 MovingAverage[MovingAverage[#, 64] & /@
   Normal@NetExtract[
     NetModel["GPT2 Transformer Trained
        on WebText Data"],
     {"decoder", 1, "1", "attention", 1
      (*key*), "Net", "Weights"}], 64],
 Frame → True, ImageSize → Large]
```

Out[•]=

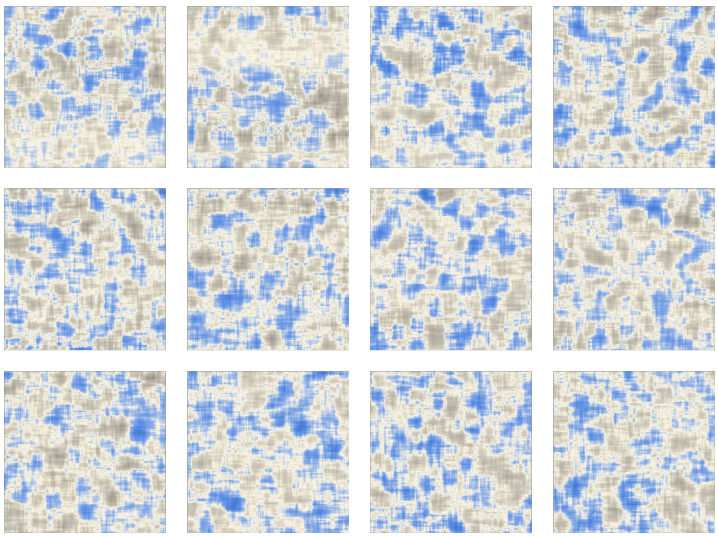


Que determina esa estructura? El lenguaje humano en ultima instancia!

Los averages que se mueven para todos los 12 heads:

In[•]:=
```
GraphicsGrid[Partition[Table[MatrixPlot[
    MovingAverage[MovingAverage[#, 64] & /@ Normal@NetExtract[
        NetModel["GPT2 Transformer Trained on WebText Data"],
        {"decoder", n, "1", "attention", 1(*key*), "Net", "Weights"}], 64],
    Frame → False], {n, 12}], 4]]
```

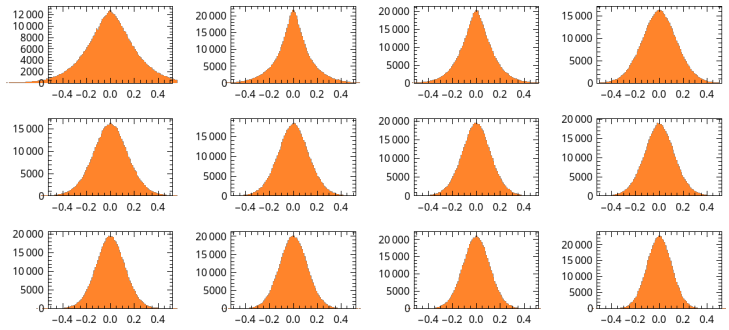
Out[•]=



Aunque luzcan parecidas, sus distribuciones:

In[•]:=
```
GraphicsGrid[Partition[Table[Histogram[
    Flatten@Normal@NetExtract[
      NetModel["GPT2 Transformer Trained on WebText Data"],
      {"decoder", n, "1", "attention", "1", "Net", "Weights"}],
    {.01}, Frame → True, PlotRange → {{-.5, .5}, Automatic},
    ChartStyle → ChatTechColors["Data"]["Orange"]], {n, 12}], 4]]
```

Out[•]=



Que hizo el transformer? Bueno transformó el embedding original en otro embedding!
(último paso) Toma estos embeddings y asigna probabilidades!

No hay looping back, ni repetidos reprocesamientos.. nada .. solo forward.
Bueno támbien lee hacia atras para saber que palabra colocar.