Introducción a Deep Learning

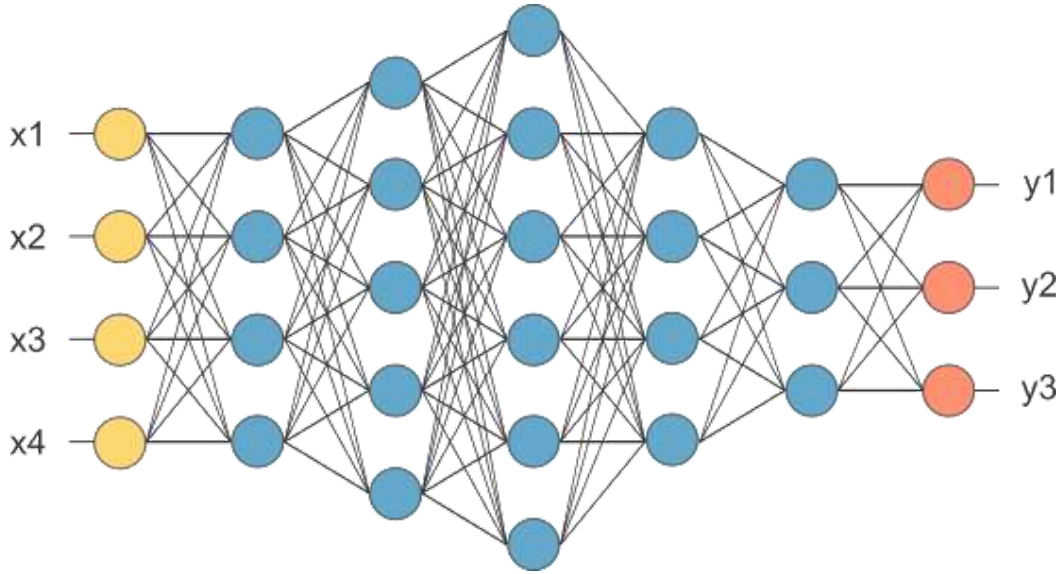
Roberto Muñoz, PhD
Astrónomo y Data Scientist
MetricArts

github.com/rpmunoz

@RobertoKPax

Temario

- Features o representación de características
- ¿Qué es el Deep Learning?
- Principales usos del Deep Learning
- Aprendizaje supervisado
- Entrenamiento



¿Qué queremos que ML haga?

- Dada una imagen, predecir patrones complejos de nivel superior

Reconocimiento

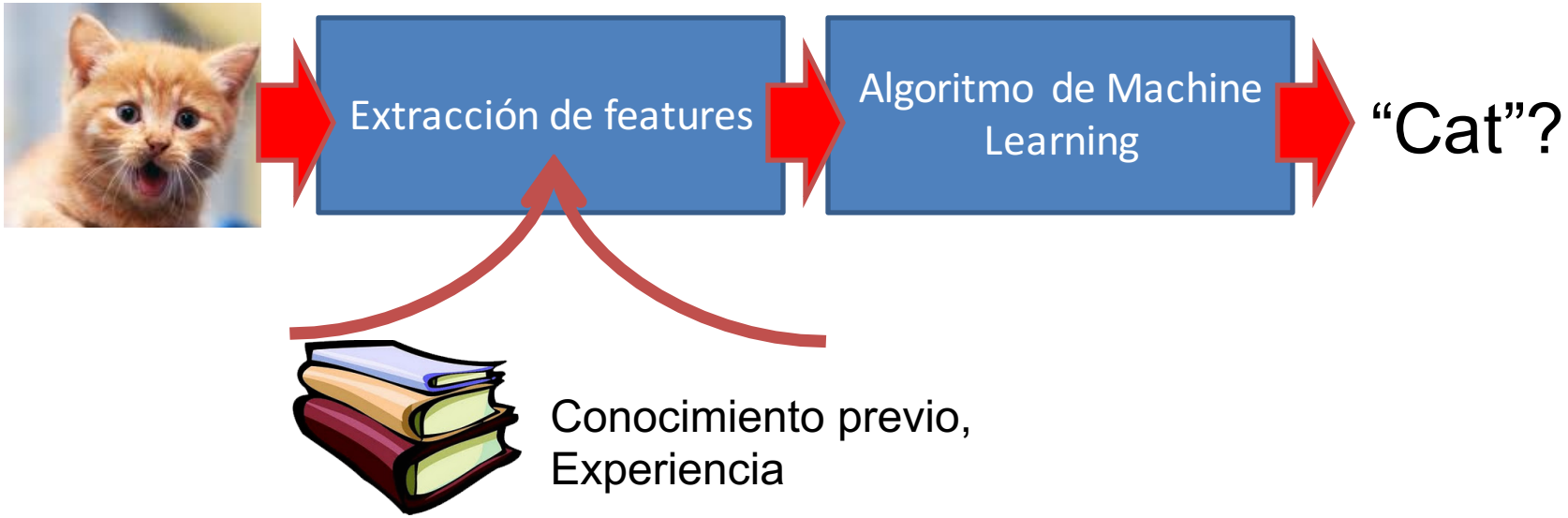
Detección

Segmentación

[Martin et al., 2001]

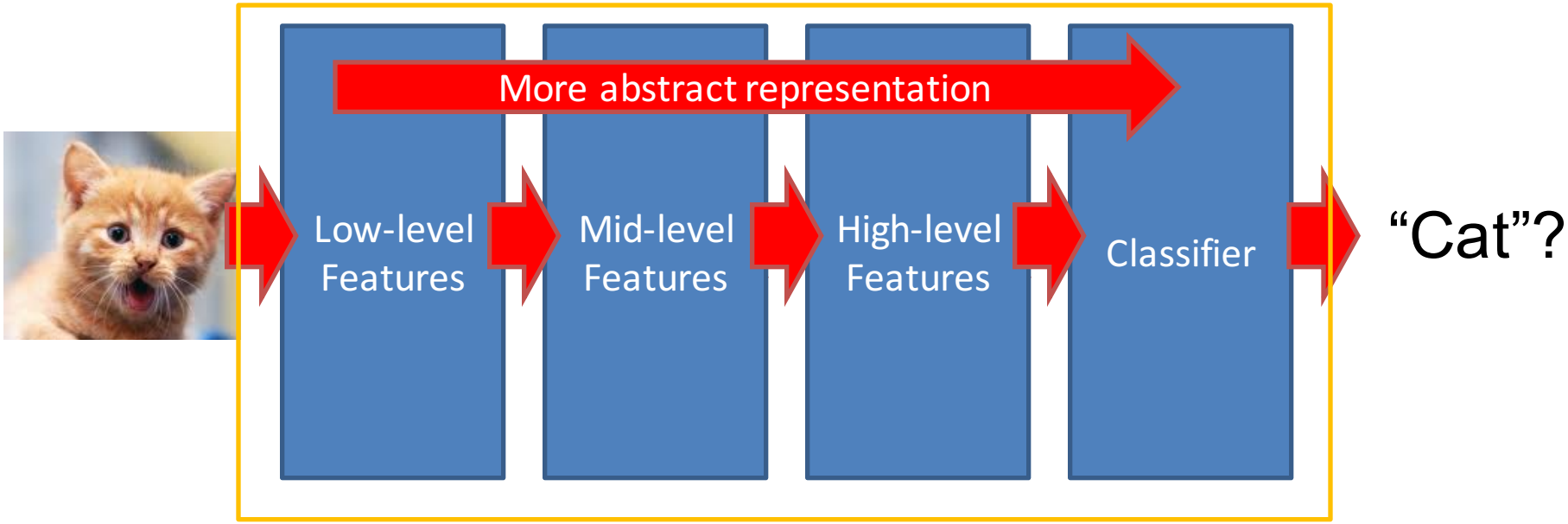
ML con features manuales

- El approach clásico es hacer machine learning con ingeniería de features manual
  - Algoritmo de ML aprende a hacer predicciones a partir de la representación de mayor nivel
  - Se usa conocimiento del dominio y experiencia previa



Deep Learning

- Deep Learning
  - Entrenar múltiples capas de features y abstraciones a partir del dato no estructurado
  - Tratar de descubrir las representaciones que permitan hacer mejores predicciones



Deep Learning

- ¿Porqué queremos usar Deep Learning?
  - Algunas decisiones requieres muchas etapas de procesamiento
    - Casos donde un modelo profundo es compacto pero un modelo superficial es muy grande e ineficiente
  - De manera intuitiva hemos construido de manera manual capas de representación
    - Automaticemos esta parte
  - Algoritmos escalan bien con datos y capacidad cómputo
    - Sumar datos es una manera de obtener buenos resultados en ML

Impacto real

- Los sistemas construidos con DL logran altos rendimientos en muchas tareas en múltiples dominios

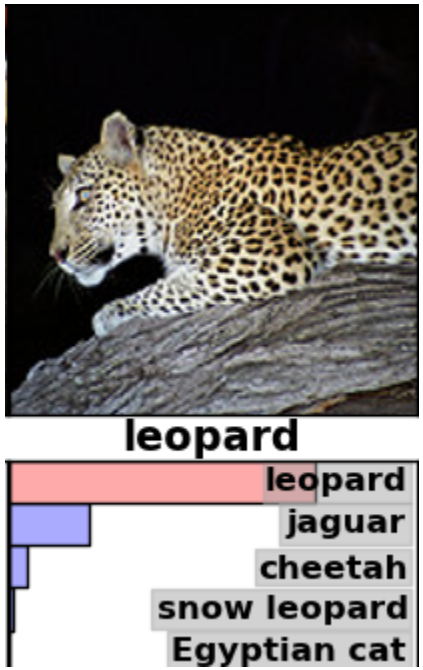
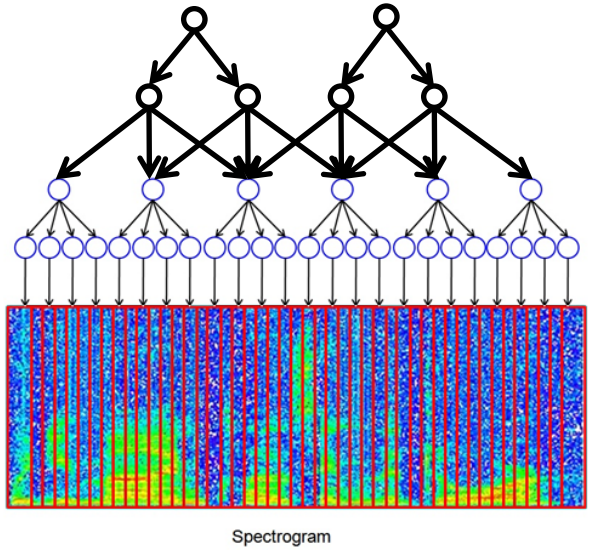
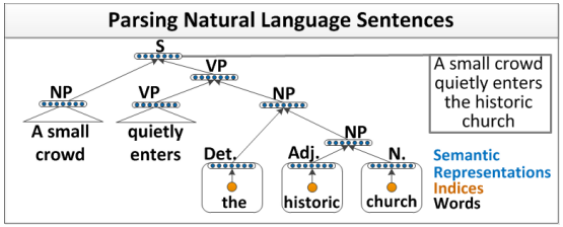


Image recognition
[E.g., Krizhevsky et al., 2012]



[Honglak Lee]

Speech recognition
[E.g., Heigold et al., 2013]



NLP
[E.g., Socher et al., ICML 2011; Collobert & Weston, ICML 2008]

Curso rápido

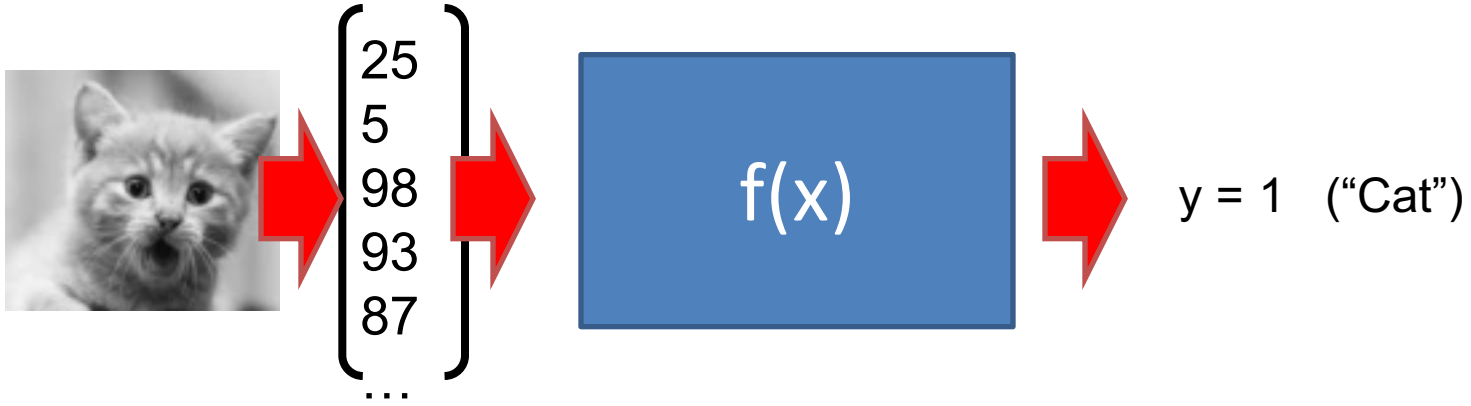
REPASO DE MACHINE LEARNING

Supervised Learning

- Dado un conjunto de datos etiquetados:

$$\mathcal{X} = \{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$$

- En este caso: $x^{(i)}$ = vector de intensidades de pixel
  $y^{(i)}$ = ID de la clase



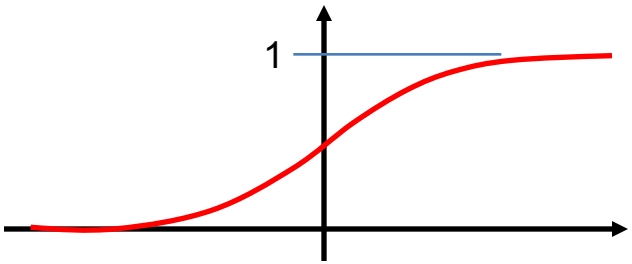
- Objetivo: Encontrar f(x) para predecir y a partir de x en del dataset de entrenamiento
  - Esperanza: Predictor aprendido funciona en el data test

Logistic Regression

- Algoritmo de clasificación binaria
  - Comenzar con función del tipo:

$$f(x;\theta) \equiv \sigma(\theta^\top x) = \frac{1}{1+\exp(-\theta^\top x)}$$

  - Interpretación: f(x) es la probabilidad que y = 1.
    - Sigmoid "nonlinearity" comprime función lineal a [0,1].



  - Encontrar el valor θ de que minimiza la función de costo:

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

$\mathbb{P}(y^{(i)}=1|x^{(i)})$

$\mathbb{P}(y^{(i)}=0|x^{(i)})$

Optimización

- Cómo determinamos θ para minimizar $\mathcal{L}(\theta)$?
- Un algoritmo: Gradient descent
  - Calcular gradiente:

$$\nabla_\theta \mathcal{L}(\theta) = \sum_i^m x^{(i)} \cdot (y^{(i)} - f(x^{(i)}; \theta))$$

  - Seguir dirección del gradiente:

$$\theta := \theta - \eta \nabla_\theta \mathcal{L}(\theta)$$

- Stochastic Gradient Descent (SGD): Dar pasos usando usando el gradiente a partir de un pequeño batch de datos.
  - Escalamiento a datasets grandes [Bottou & LeCun, 2005]

¿Es suficiente?

- Costo es convexo → Podemos siempre encontrar un mínimo
- Funciona para problemas simples:
  - Clasificar digitos como 0 o 1 usando intensidad de pixels.
  - Ciertos pixels contienen más información --- e.g., pixel central

- Falla para problemas levemente más complejos
  - ¿Es esto una taza de café?

¿Porqué CV es difícil?

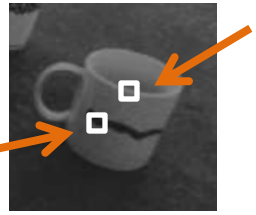
"Taza de café"

Intensidad de los pixels

177	153	118	91	85	100	124	145
151	124	93	77	86	115	148	168
115	93	78	83	108	145	177	191
88	79	84	104	136	168	190	197
82	85	103	127	152	170	180	182
91	101	120	138	150	157	159	159
103	114	127	136	140	140	140	141
111	119	126	130	130	129	128	130

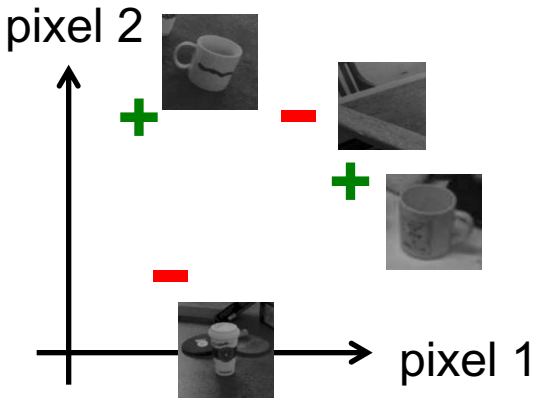
La intensidad es una representación pobre

¿Porqué CV es difícil?



[72 160]

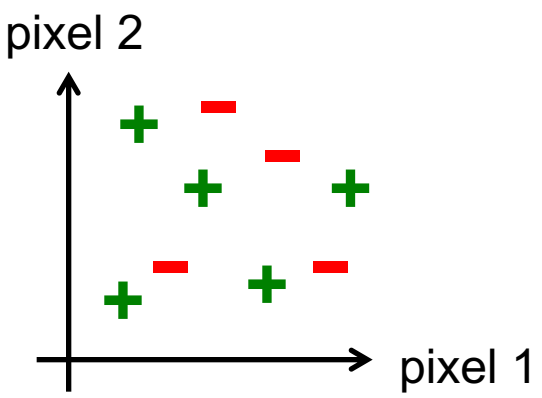
Intensidad de pixel



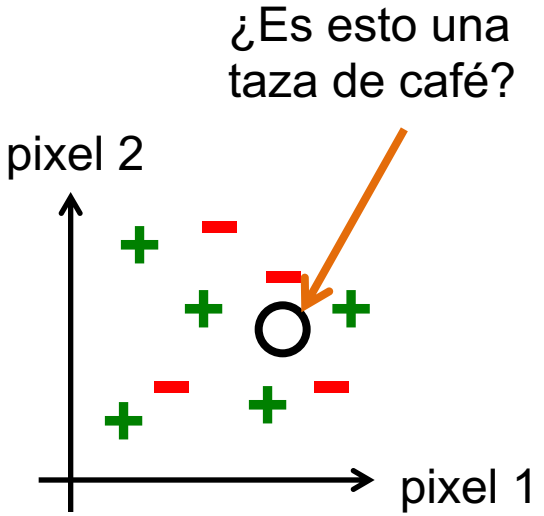
\+ Taza de café

– No es taza de café

¿Porqué CV es difícil?



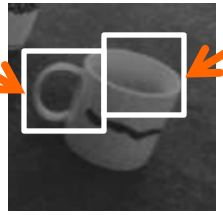
Algoritmo de aprendizaje

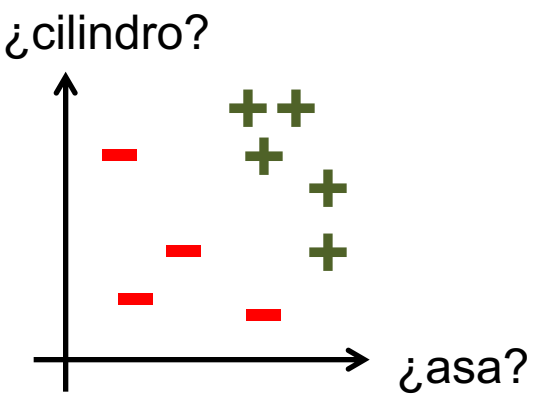


+ Taza de café

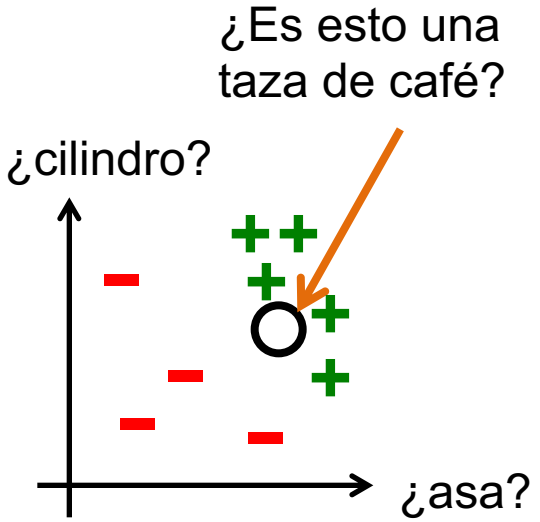
− No es taza de café

Features





Algoritmo de aprendizaje



\+ Taza de café

– No es una taza de café

Features

- Los features son usualmente transformaciones hard-wired que forman parte del sistema.
  - Formalmente, una función que mapea el dato crudo a una representación de mayor nivel.

$$\Phi(x) : \Re^n \rightarrow \Re^K$$

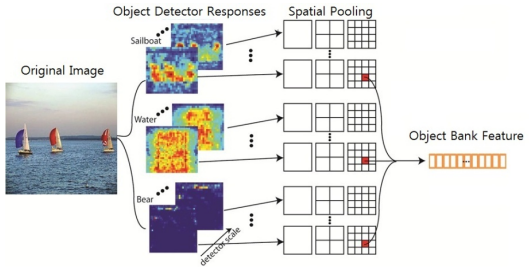
  - Completamente estática --- Use $\varphi(\mathrm{x})$ en vez de x y aplique la regresión logística vista anteriormente.

¿Cómo podemos calcular features útiles?

Features

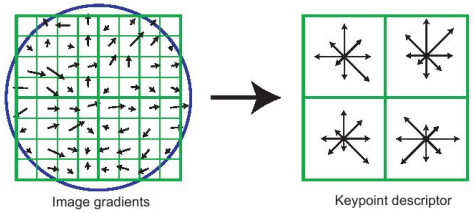
- Una gran inversión destinada a construir representaciones de features específicas para ciertas aplicaciones
  - Encontrar patrones de nivel superior de tal manera que la decisión final se puede hacer fácilmente con los algoritmos de ML

Object Bank [Li et al., 2010]

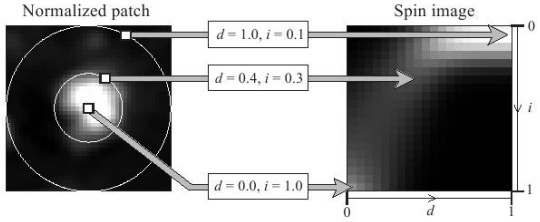


Super-pixels
[Gould et al., 2008; Ren & Malik, 2003]

SIFT [Lowe, 1999]



Spin Images [Johnson & Hebert, 1999]



Redes neuronales

DEEP LEARNING
SUPERVISADO

Idea básica

- Vimos el aprendizaje supervisado cuando los features ϕ(x) son estáticos.
  - Extendamos el análisis al caso en que los features son dados por funciones ajustables y que tienen sus propios parámetros

$$\mathbb{P}(y = 1|x) = f(x; \theta, W) = \sigma(\theta^\top \sigma(Wx))$$

La función más externa es la regresión logísitica.

Inputs son features - un feature por cada fila de W:

$$\begin{bmatrix} \sigma(w_1 x) \\ \sigma(w_2 x) \\ \dots \\ \sigma(w_K x) \end{bmatrix}$$

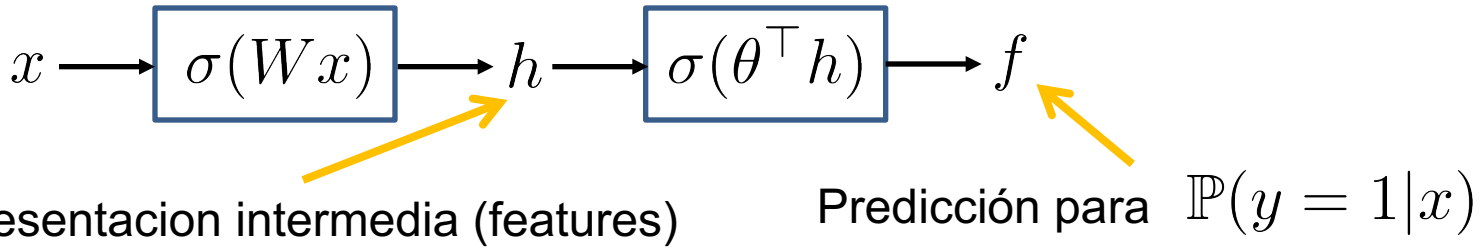
Basic idea

- Aprendizaje supervisado para clasificación de dos clases, minimizar:

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

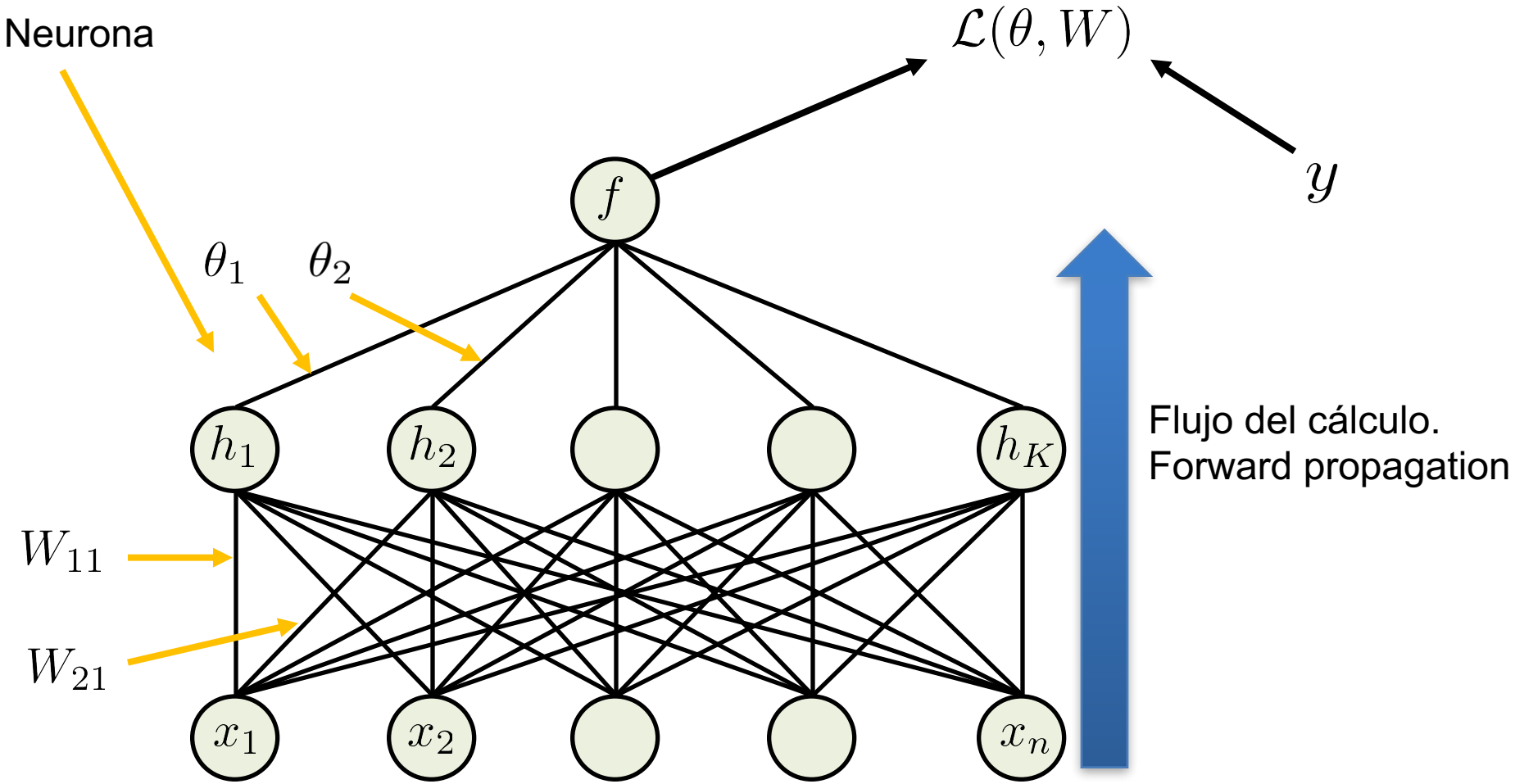
- Similar a regresión logísitica, pero ahora f(x) tiene múltiples etapas (capas, módulos):

$$f(x; \theta, W) = \sigma(\theta^\top \sigma(Wx))$$



Red neuronal artificial (ANN)

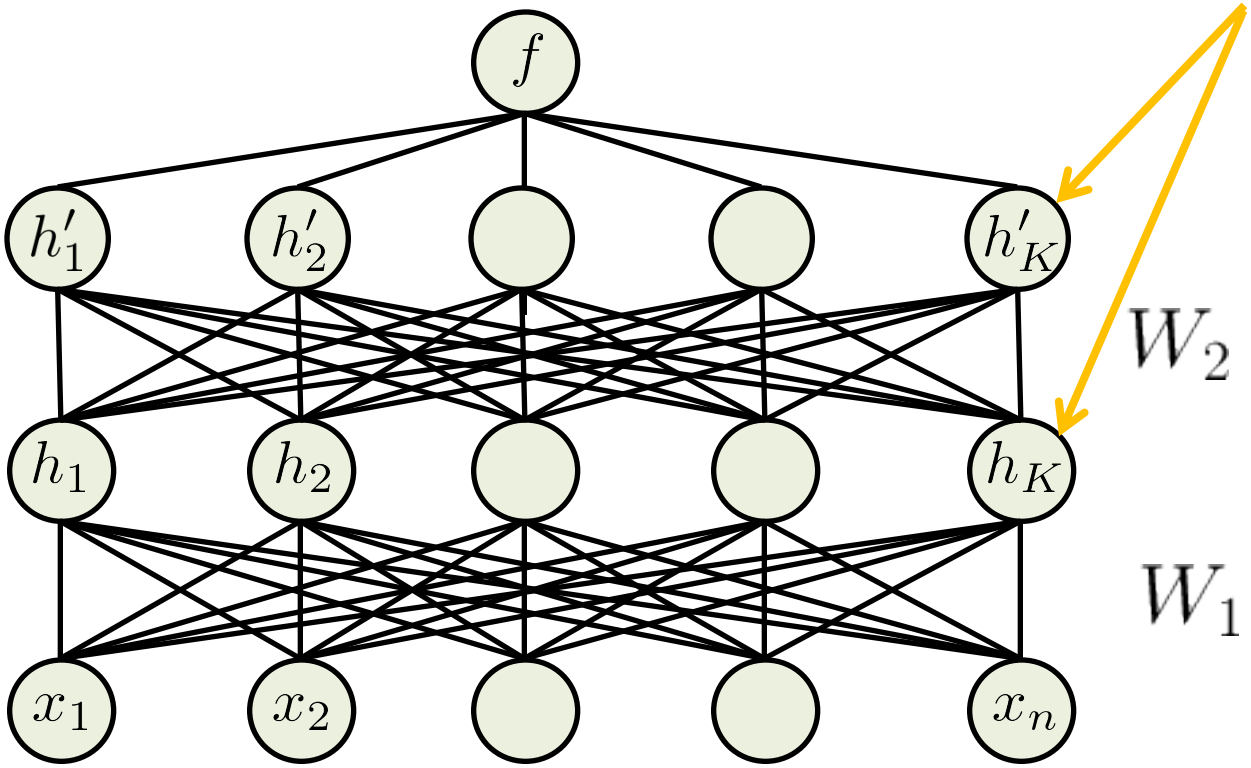
- Este modelo es una red neuronal sigmoidal



Red neuronal artificial

- Se pueden agregar múltiples capas

Deben aprender múltiples etapas de representaciones internas



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

Back-propagation

- Minimizar

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

- Para minimizar $\mathcal{L}(\theta, W)$ necesitamos gradientes:

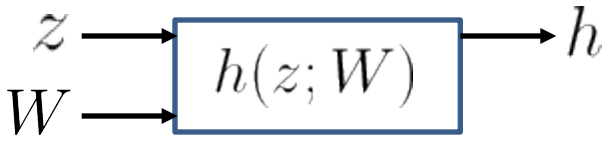
$$\nabla_\theta \mathcal{L}(\theta, W) \text{ and } \nabla_W \mathcal{L}(\theta, W)$$

  – Usamos el algoritmo gradient descent como antes

- Formula para $\nabla_\theta \mathcal{L}(\theta, W)$ puede ser determinada a mano; ¿Pero que hay acerca de W?

Regla de la cadena

- Supongamos que tenemos el siguiente módulo:



- Y conocemos $[\nabla_h \mathcal{L}]_j = \frac{\partial \mathcal{L}(\theta, W)}{\partial h_j}$ y $\frac{\partial h_j}{\partial z_k}$, regla de la cadena da:

Matriz Jacobiana

$$\frac{\partial \mathcal{L}(\theta, W)}{\partial z_k} = \sum_j \frac{\partial \mathcal{L}(\theta, W)}{\partial h_j} \frac{\partial h_j}{\partial z_k} \Rightarrow \nabla_z \mathcal{L} = J_{h,z}(\nabla_h \mathcal{L})$$

Similarmente para W:

$$\frac{\partial \mathcal{L}(\theta, W)}{\partial W_{kl}} = \sum_j \frac{\partial \mathcal{L}(\theta, W)}{\partial h_j} \frac{\partial h_j}{\partial W_{kl}} \Rightarrow \nabla_W \mathcal{L} = J_{h,W}(\nabla_h \mathcal{L})$$

➢ Dado el gradiente con respecto al output, podemos construir un nuevo módulo que calcule el gradiente con respecto al input

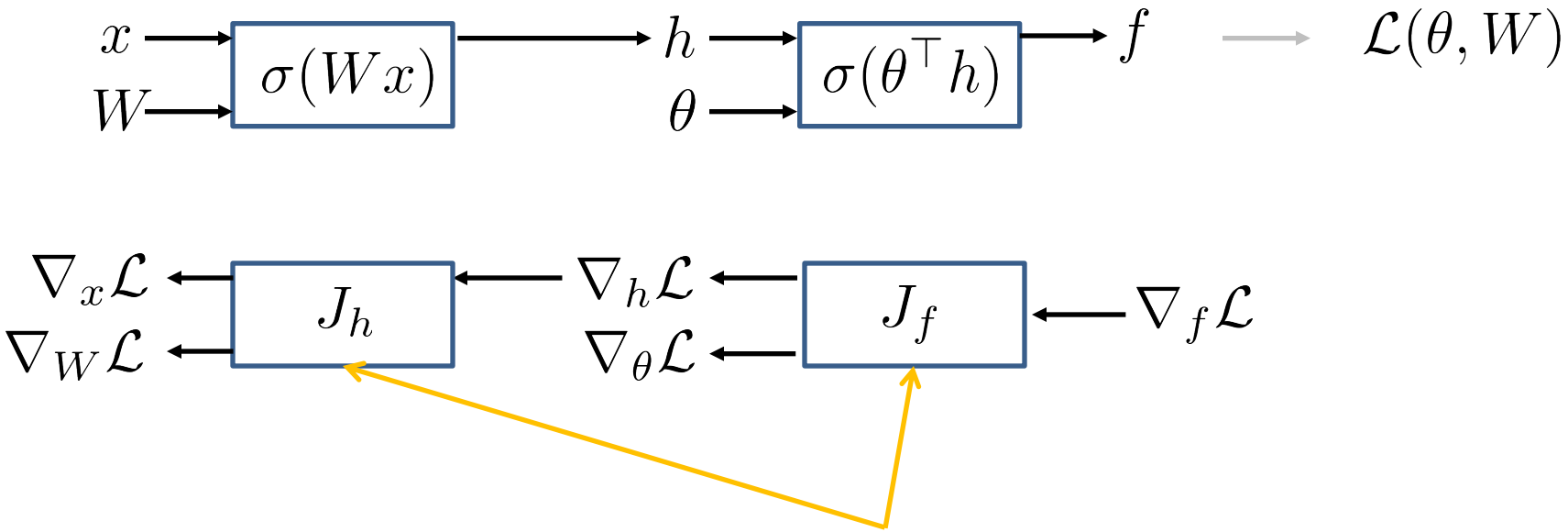
Regla de la cadena

- Fácil de construir un toolkit de reglas para calcular los gradientes $\delta \equiv \nabla_h \mathcal{L}$
  - Diferenciación automatizada! E.g., Theano, TensorFlow, Pytorch

Función $h(z)$	Gradiente c.r. input $\nabla_z \mathcal{L}$	Gradiente c.r. parámetros $\nabla_W \mathcal{L}$
$h = Wz$	$W^\top \delta$	$\delta \quad z^\top$
$h = \sigma(z)$	$\delta \odot \sigma(z) \odot (1 - \sigma(z))$	
$h = \sqrt{Wz^2}$	$(W^\top \frac{\delta}{h}) \odot z$	$\frac{\delta}{2h}(z^2)^\top$
$h = \max_j \{z_j\}$	$1\{z_j = h\}\delta$	

Back-propagation

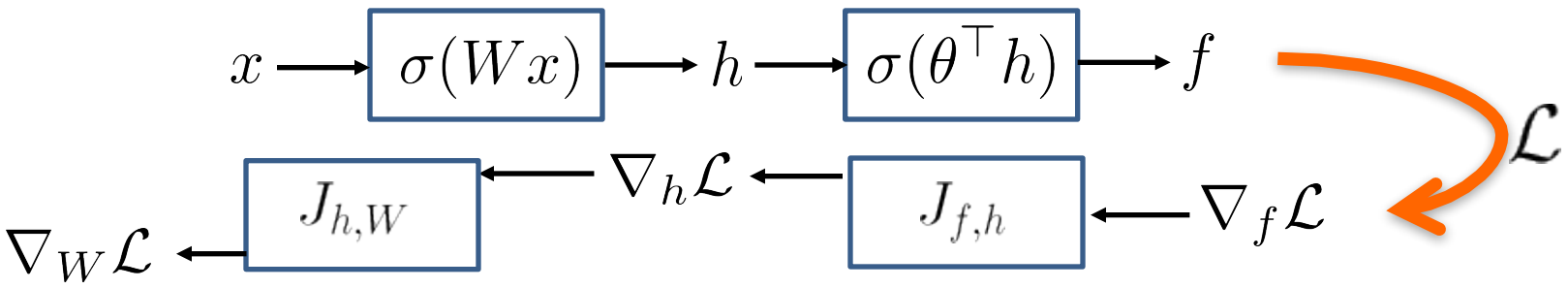
- Podemos nuevamente aplicar la regla de la cadena para obtener gradientes para los valores intermedios y parámetros



Módulos “Backward” para cada etapa

Ejemplo

- Dado $\nabla_f \mathcal{L}$ calcule $\nabla_W \mathcal{L}$:



Usando algunos valores de nuestra tabla:

$$f = \sigma(\theta^T h)$$

$$f'(z) = f(z)\big(1 - f(z)\big)$$

$$\nabla_h \mathcal{L} = \theta[f(1-f)(\nabla_f \mathcal{L})]$$

$$\nabla_W \mathcal{L} = [h \odot (1-h) \odot (\nabla_h \mathcal{L})]x^\top$$

Procedimiento para entrenar

- Recolectar dataset de entrenamiento
  - Para SGD: Random shuffle después de cada época!

$$\mathcal{X} = \{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$$

- Para cada batch de ejemplos:
  - Calcular gradiente c.r. a todos los parámetros en la red

$$\Delta_\theta := \nabla_\theta \mathcal{L}(\theta, W)$$
$$\Delta_W := \nabla_W \mathcal{L}(\theta, W)$$

  - Hacer pequeños updates a los parámetros

$$\theta := \theta - \eta_\theta \Delta_\theta$$
$$W := W - \eta_W \Delta_W$$

  - Repetir hasta lograr convergencia

Procedimiento para entrenar

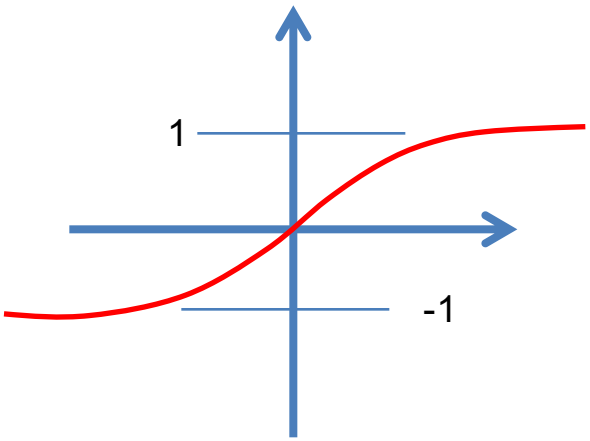
- Históricamente, esto no ha funcionado del todo bien
  - No-convexo: Mínimo local; criterio de convergencia.
  - Optimización es más difícil al agregar más capas.
    - Problema del vanishing gradient
  - Difícil de diagnosticar y hacer debugging

- Importan un par de aspectos:
  - Escoger no linearidades.
  - Inicialización de parámetros.
  - Optimización de parámetros: step size, schedule.

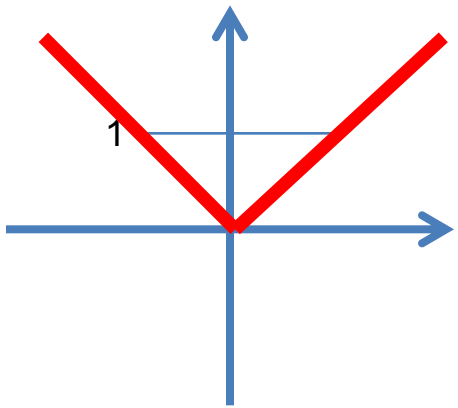
No linearidades

- Escoger las funciones dentro de la red importa.
  - Funciones sigmoidales se vuelven difíciles.
  - Funciones que se usan comúnmente:

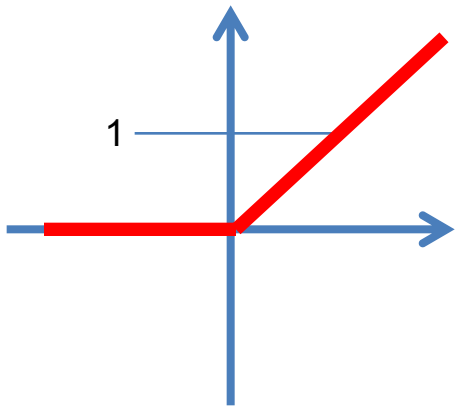
tanh(z)



abs(z)



ReLu(z) = max{0, z}

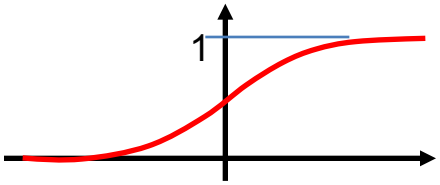


"Rectified Linear Unit"
→ Muy popular

[Nair & Hinton, 2010]

Inicialización

- Usualemente valores aleatorios pequeños.
  - Tratar de escoger valores tales que el input a neurona evite la saturación o llegue a régimen no diferenciable.



  - Ocasionalmente revisar valores que saturan o explotan.
  - Valores grandes convergen rápido, pero modelos son peores!

- Esquemas de inicialización para unidades particulares:
  - tanh units: Unif[-r, r]; sigmoid: Unif[-4r, 4r].

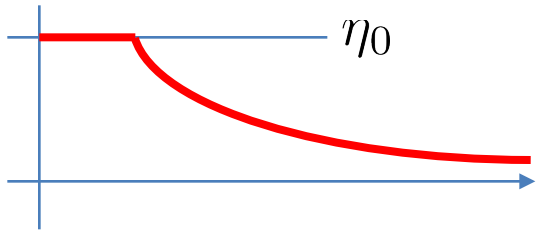
$$r = \sqrt{6/(\text{fan-in} + \text{fan-out})}$$

  - fan-in: Número de valores que entran a capa oculta
    fan-out: Número de valores que salen de capa oculta

Optimización: Step sizes

- Escoja un step size para el SGD de manera cuidadosa.
  - Un factor ~2 puede hacer una diferencia
- Estrategias:
  - Fuerza bruta: Intentar varios; Escoger el que entregó mejores resultados.
  - Escoger de manera que el update típico a los pesos es aproximadamente 1/1000 veces la magnitud del peso
    - Más pequeño si fan-in a neuronas es grande.
  - Carrera: Escoger step size con mejor error en la validación después de T pasos.
    - No es preciso si T es muy pequeño.
- Step size schedule:
  - Simple 1/t schedule:

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



  - O fijar step size. Si hay poco progreso en la función costo después de T pasos, disminuir el step size a la mitad.

Ejemplo en línea

- Jugar con una red neuronal en tu navegador

  http://playground.tensorflow.org