

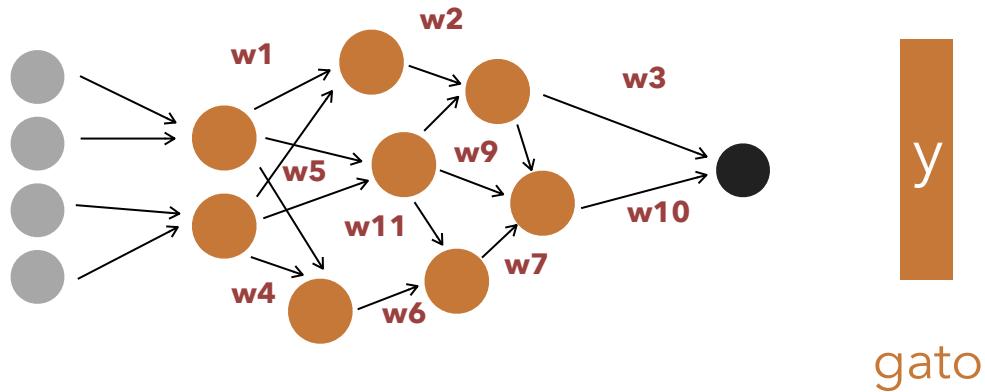
Entrenamiento Básico de Redes Neuronales

Objetivo



¿Qué Significa Entrenar una Red?

- Una **red neuronal** nos permite implementar una función $f(x)$ usando un grafo de computación. Modificando los **pesos de conexión y umbrales de excitación** de las unidades en el grafo, podemos cambiar la función que computa la red.



- **Entrenar la red** significa **determinar valores para esos parámetros desde ejemplos**, es decir, datos que permitan inferir el comportamiento deseado.

Modelo Supervisado

- Antes de diseñar un método que permita entrenar la red, es útil precisar cuál es nuestro objetivo y qué exactamente asumimos disponible.
- En el modelo supervisado, asumimos que podemos disponer de un **conjunto de n ejemplos** $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ **de tipo input-output** (x, y) donde x representa un posible dato de entrada al sistema e y representa la respuesta que el sistema debiese calcular cuando se presenta x .



Modelo Supervisado

- Por ejemplo, en el problema de predecir el precio de una propiedad en la ciudad de Santiago, los ejemplos corresponderían a pares (x, y) donde x describe una determinada propiedad de la ciudad e y corresponde a precio verdadero de esa propiedad (groundtruth).

	$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \end{pmatrix}$	3800
	$\begin{pmatrix} 5 \\ 1 \\ 0 \\ 3 \end{pmatrix}$	4500
	$\begin{pmatrix} 3 \\ 1 \\ 1 \\ 4 \end{pmatrix}$	7200
	$\begin{pmatrix} 1 \\ 3 \\ 0 \\ 2 \end{pmatrix}$	3500
	$\begin{pmatrix} 6 \\ 3 \\ 0 \\ 3 \end{pmatrix}$	4300
	$\begin{pmatrix} 5 \\ 0.5 \\ 0 \\ 4 \end{pmatrix}$	6500

Ejemplos "Etiquetados"

$$x \in \mathbb{R}^d$$
$$y \in \mathbb{R}^+$$

Notación

- Tómese un momento para familiarizarse con la notación:

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^n = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$$

↓

índice correspondiente a un caso o ejemplo



$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \end{pmatrix} \quad 3800$$



$$\begin{pmatrix} 5 \\ 1 \\ 0 \\ 3 \end{pmatrix} \quad 4500$$



$$\begin{pmatrix} 3 \\ 1 \\ 1 \\ 4 \end{pmatrix} \quad 7200$$



$$\begin{pmatrix} 1 \\ 3 \\ 0 \\ 2 \end{pmatrix} \quad 3500$$



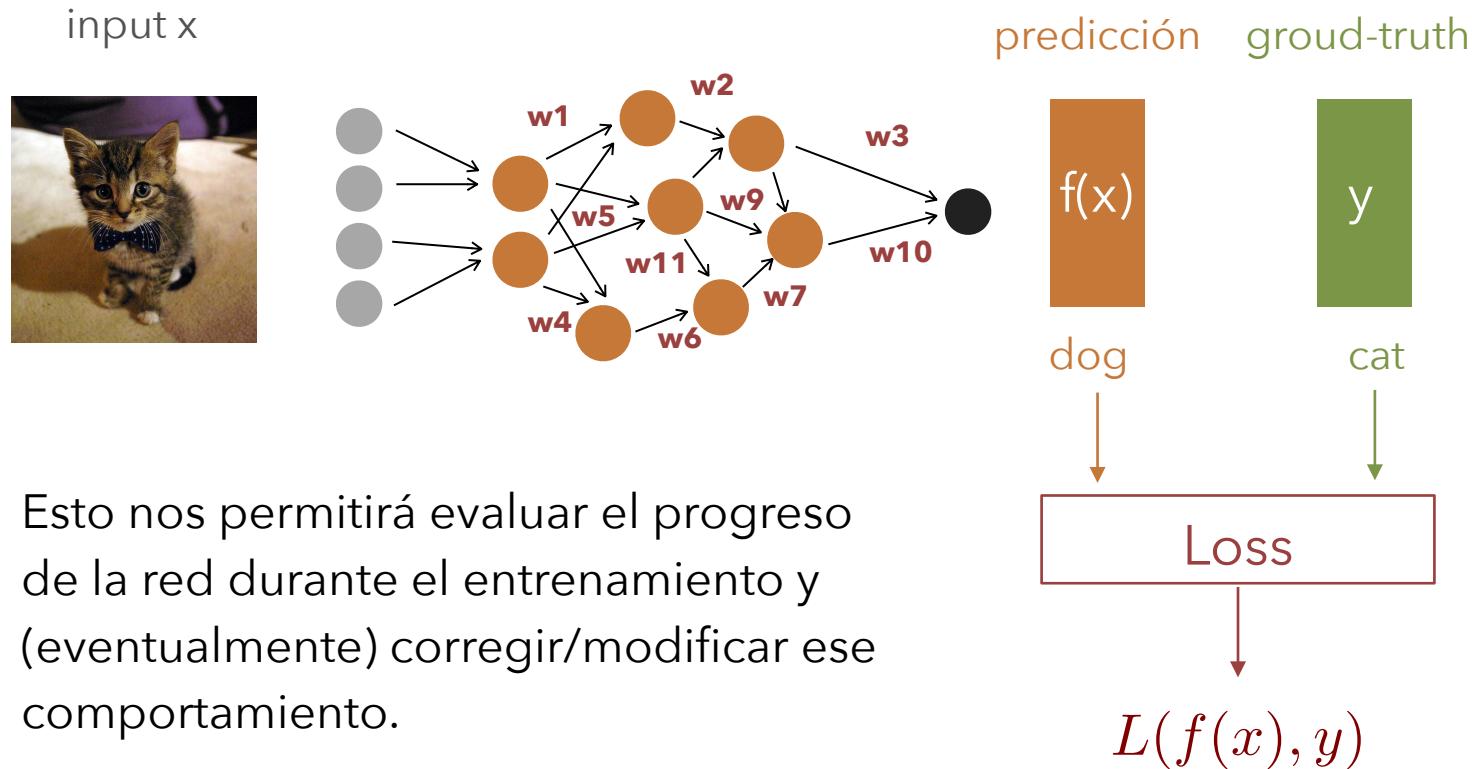
$$\begin{pmatrix} 6 \\ 3 \\ 0 \\ 3 \end{pmatrix} \quad 4300$$



$$\begin{pmatrix} 5 \\ 0.5 \\ 0 \\ 4 \end{pmatrix} \quad 6500$$

Loss (Función de Costo)

- Supongamos que para un ejemplo (x, y) la red produce $f(x) \neq y$. Una función de costo (loss function) es una función $L : Y \times Y \rightarrow \mathbb{R}_0^+$ que nos permitirá cuantificar la magnitud del error cometido por la red: $L(f(x), y)$.

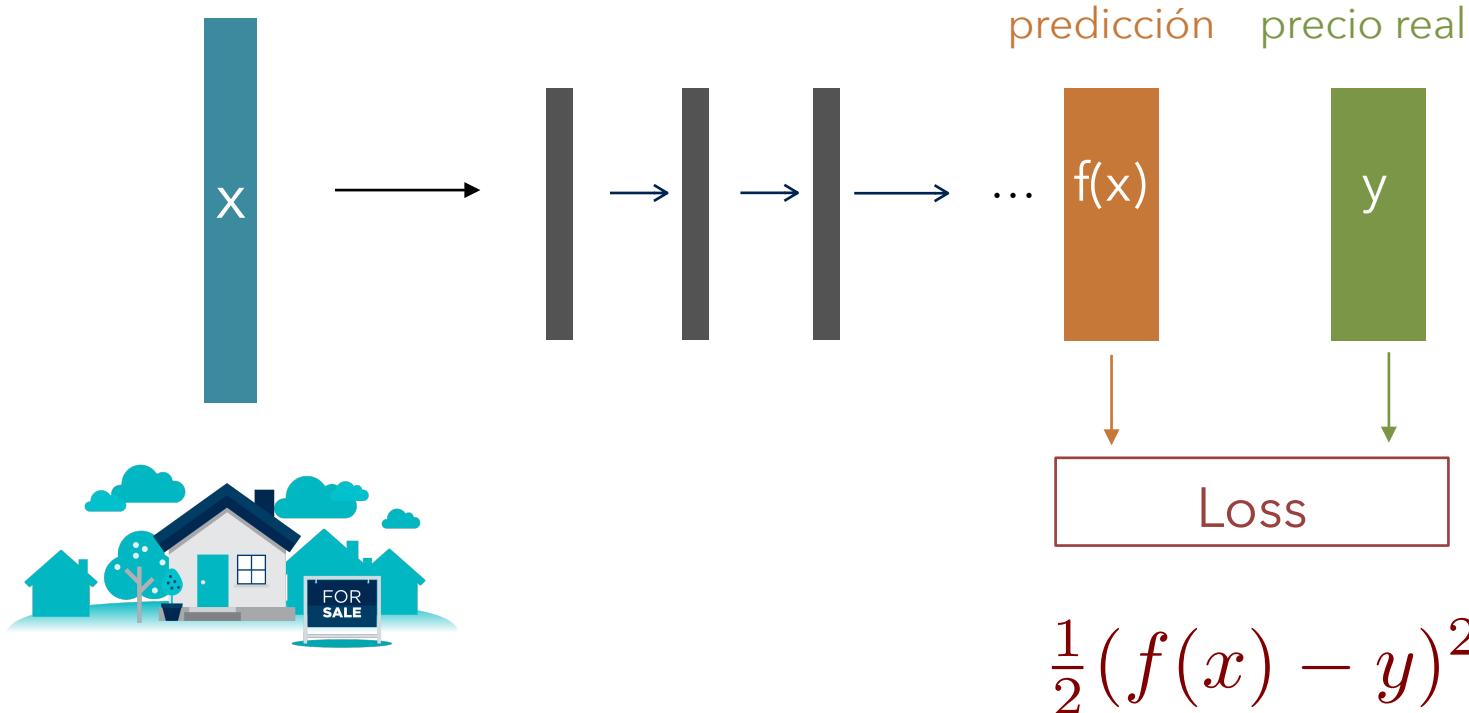


- Esto nos permitirá evaluar el progreso de la red durante el entrenamiento y (eventualmente) corregir/modificar ese comportamiento.

Loss Clásica para Regresión

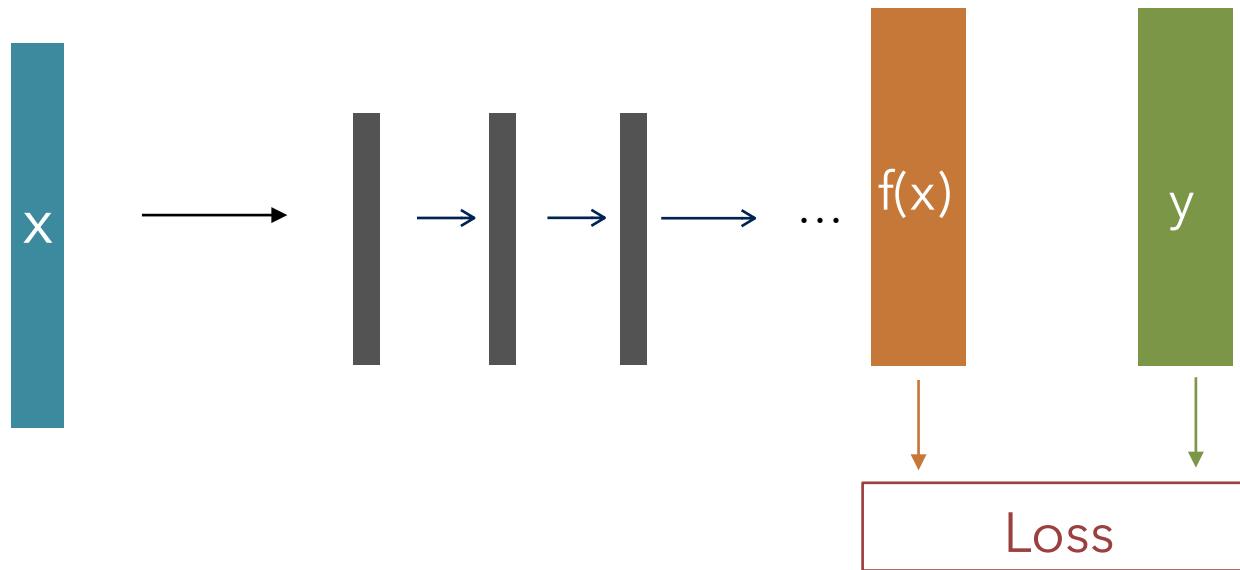
- Por ejemplo, en regresión uni-variada (1 salida real) es común utilizar el error cuadrático (quadratic loss). Justificaremos esta elección más adelante.

house descriptor



Loss Clásica para Regresión

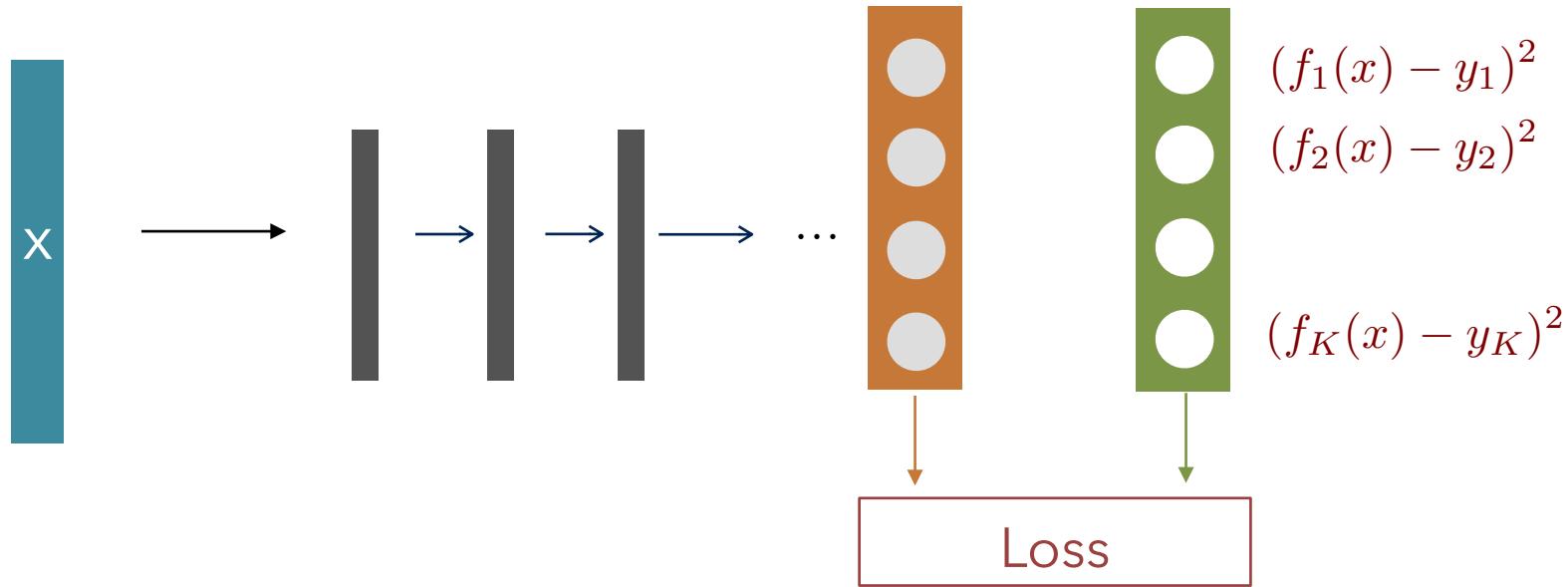
- Por ejemplo, en regresión multi-variada (K salidas real) es común utilizar el error cuadrático (quadratic loss). Justificaremos esta elección más adelante.



$$\frac{1}{2} \|f(x) - y\|^2 = \frac{1}{2} \sum_k (f_k(x) - y_k)^2$$

Loss Clásica para Regresión

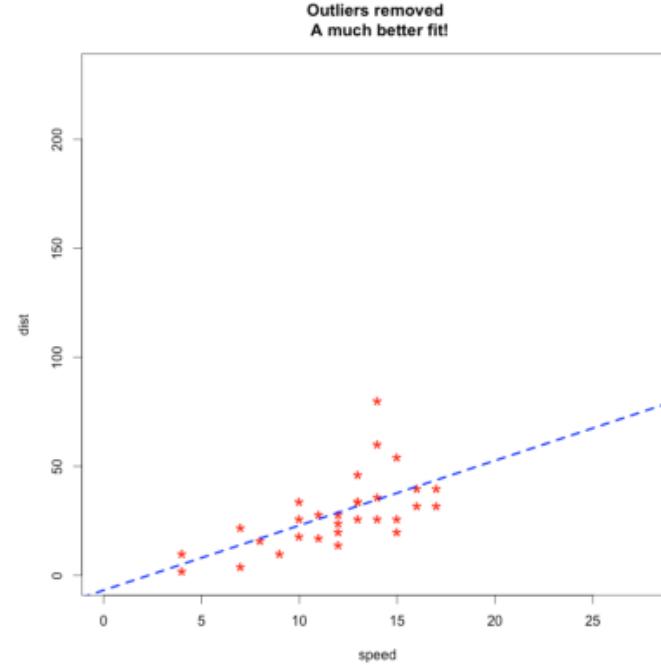
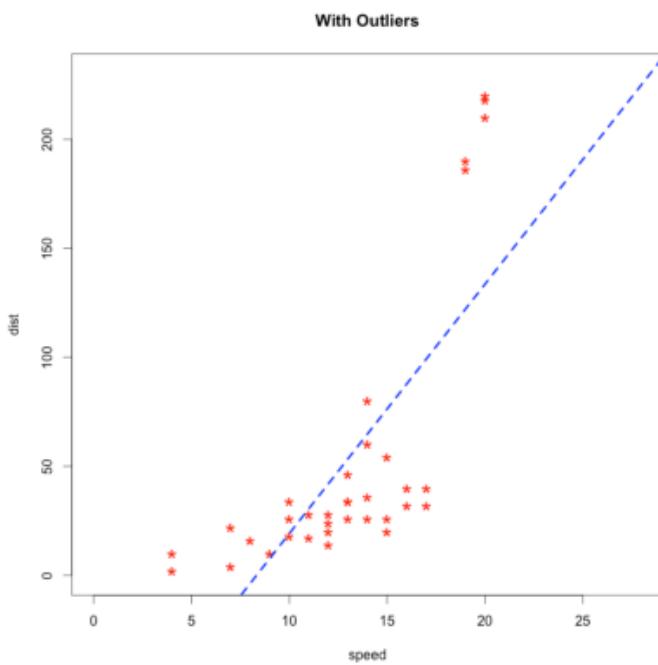
- En este último caso, cada neurona de salida contribuye aditivamente al error total:



$$\frac{1}{2} \|f(x) - y\|^2 = \frac{1}{2} \sum_k (f_k(x) - y_k)^2$$

Casos Especiales (Robustez)

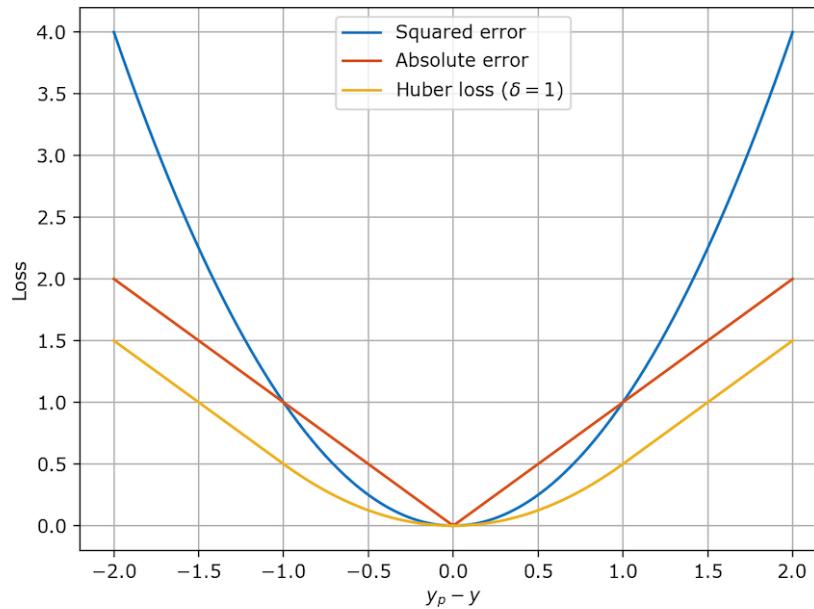
- Es bien sabido que la función de error cuadrático es poco robusta, en el sentido permite que observaciones atípicas (valor de y insusualmente grande, insusualmente pequeño, o que se aleja significativamente de la tendencia general) terminen determinando el entrenamiento del modelo. Las redes neuronales no están exentas de este problema.



Casos Especiales (Robustez)

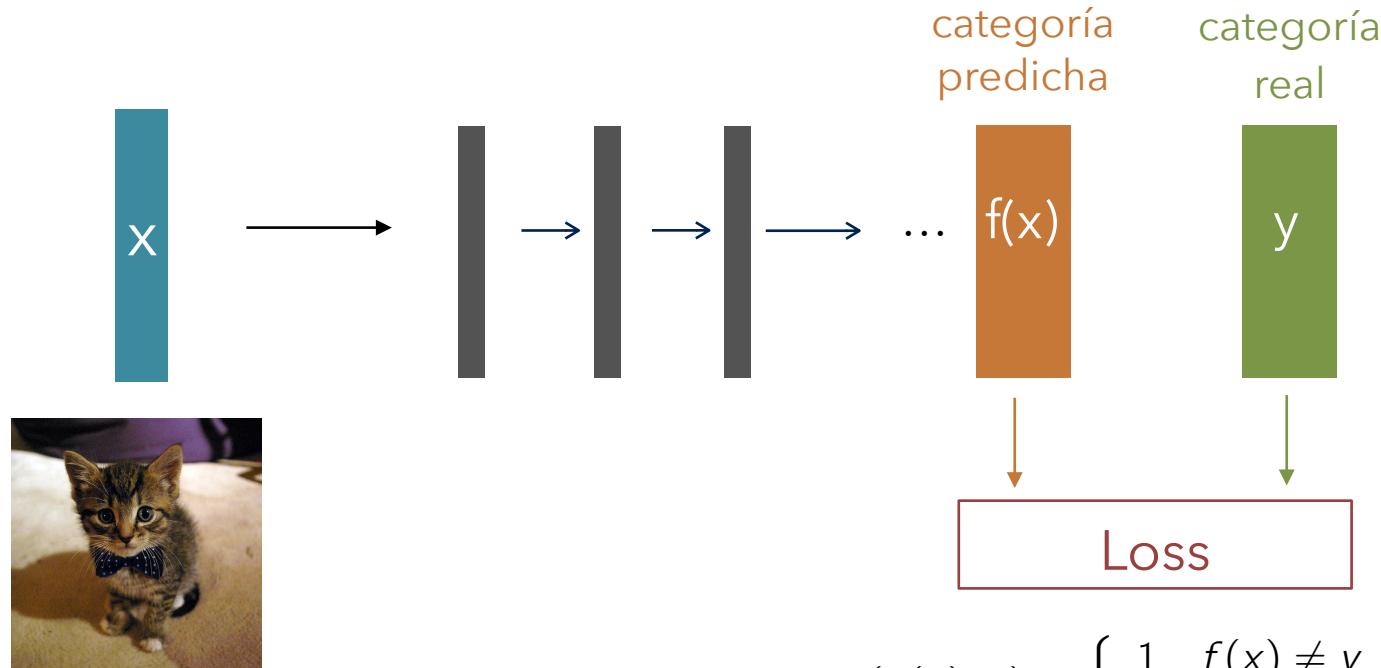
- En esos casos, podría ser conveniente considerar otras funciones de costo (como por ejemplo la función de Huber)

$$L(f(x), y) = \begin{cases} \frac{1}{2}(f(x) - y)^2 & |f(x) - y| \leq \delta \\ |f(x) - y| & |f(x) - y| > \delta \end{cases}$$



Losses para Clasificación

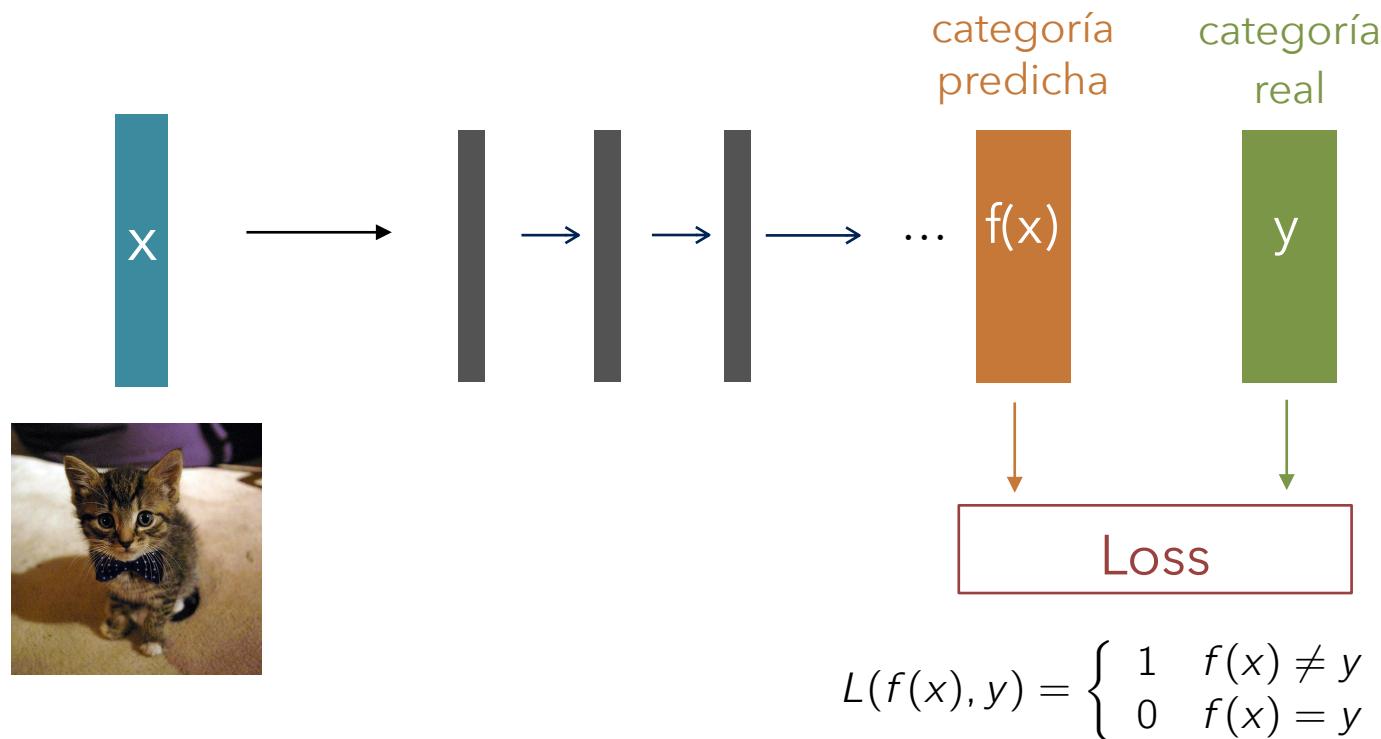
- En clasificación, las respuestas que se desea que la red aprenda son categorías. ¿Cómo medimos el error en este caso?
- Una alternativa es no contabilizar un error ($L=0$) si la predicción y el valor deseado coinciden y en otro caso contabilizar un error unitario ($L=1$).



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

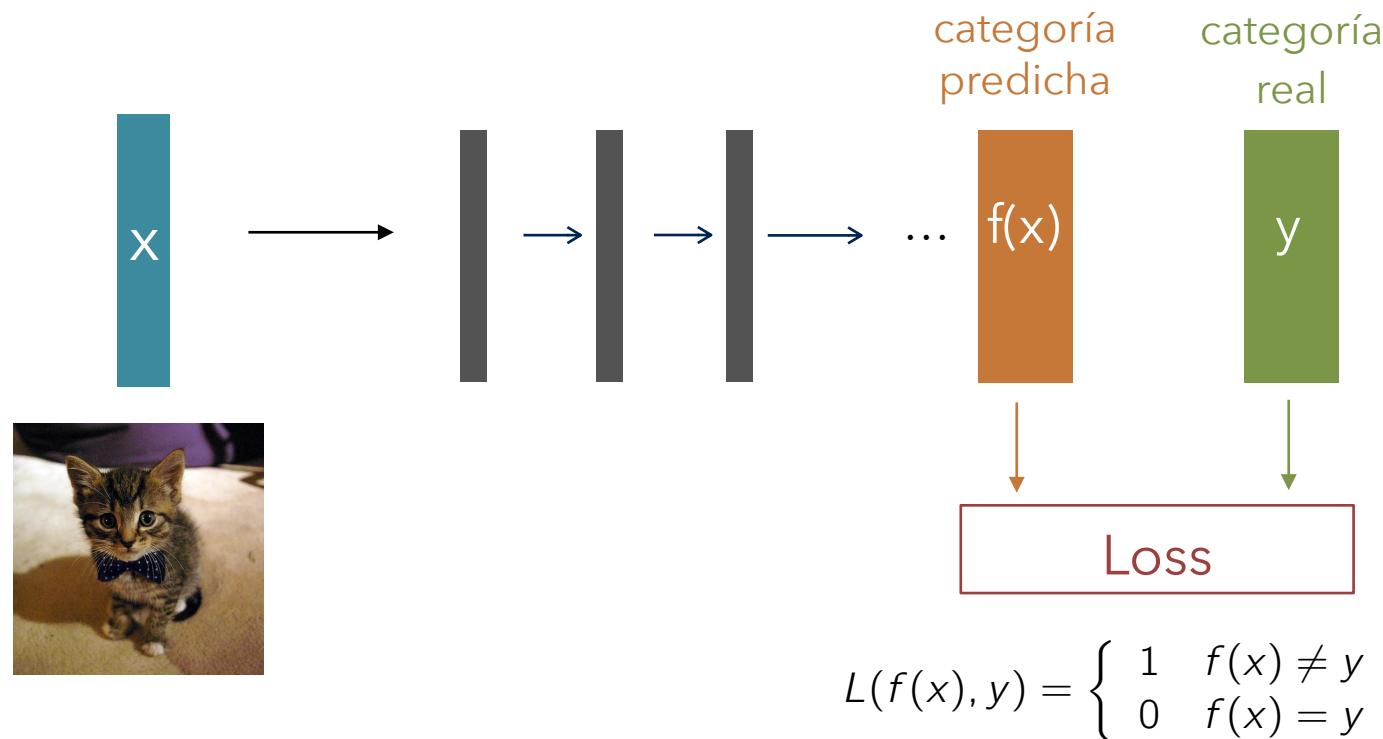
Losses para Clasificación

- Esta función (denominada error de clasificación binario, 0-1 o simplemente error de clasificación) puede ser útil para **evaluar** el resultado del entrenamiento, pero no es una buena alternativa para **entrenar** a la red.



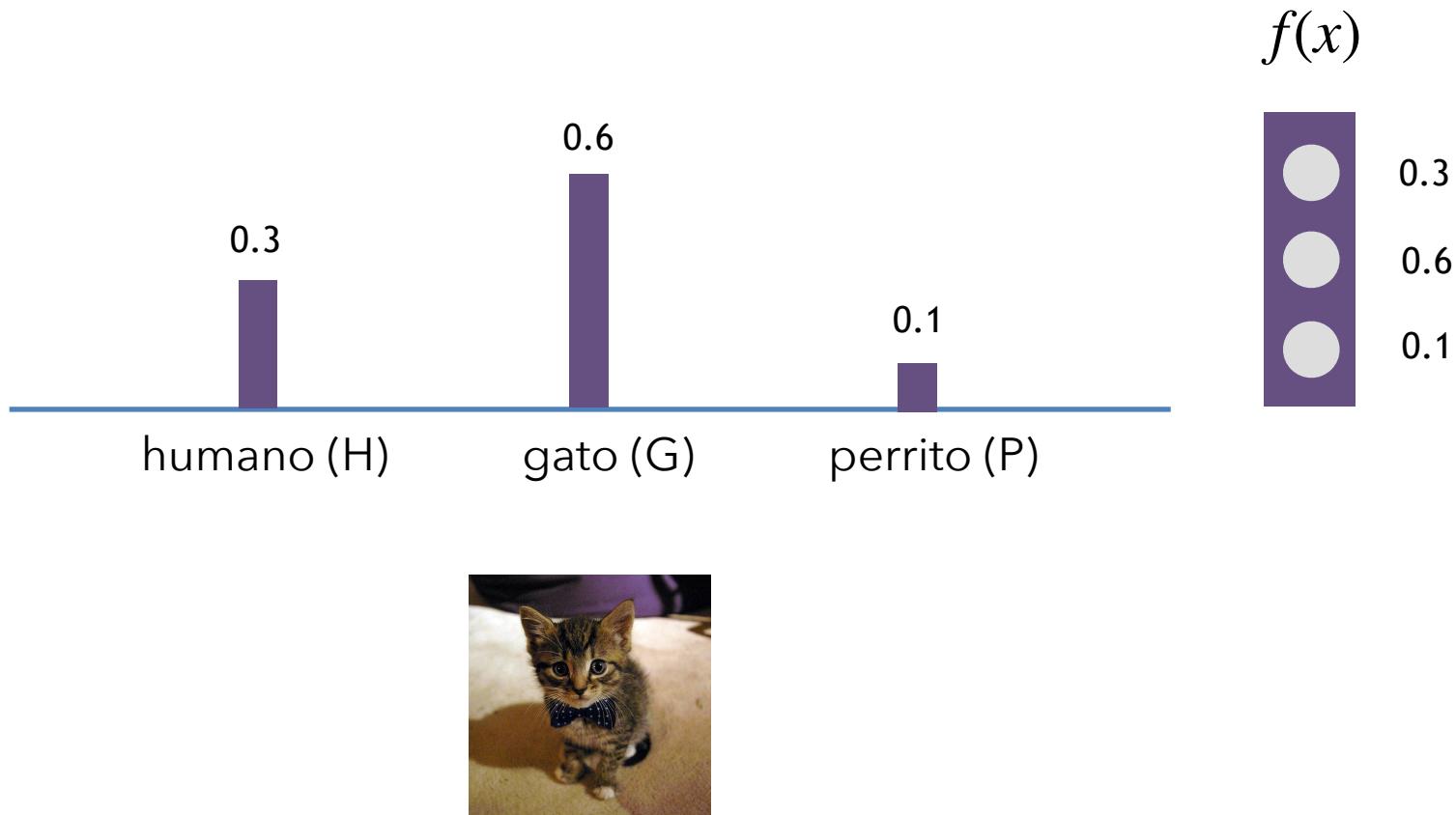
Losses para Clasificación

- Esta función (denominada error de clasificación binario, 0-1 o simplemente error de clasificación) puede ser útil para **evaluar** el resultado del entrenamiento, pero no es una buena alternativa para **entrenar** a la red.



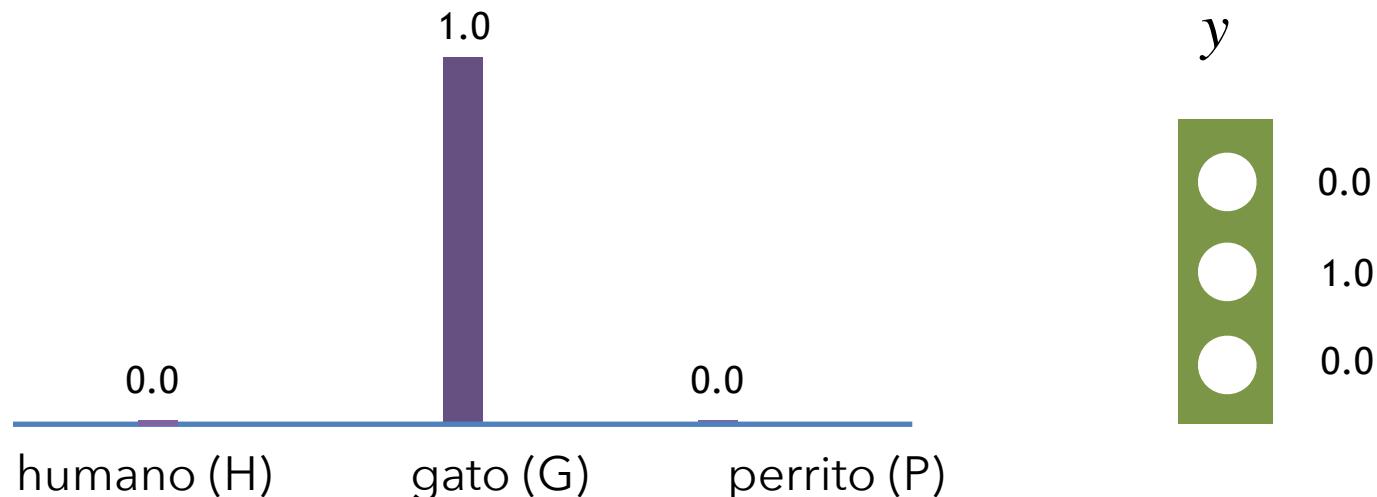
Losses para Clasificación

- Una mejor alternativa es recordar que la red se puede entrenar para predecir una distribución de probabilidad sobre las categorías posibles



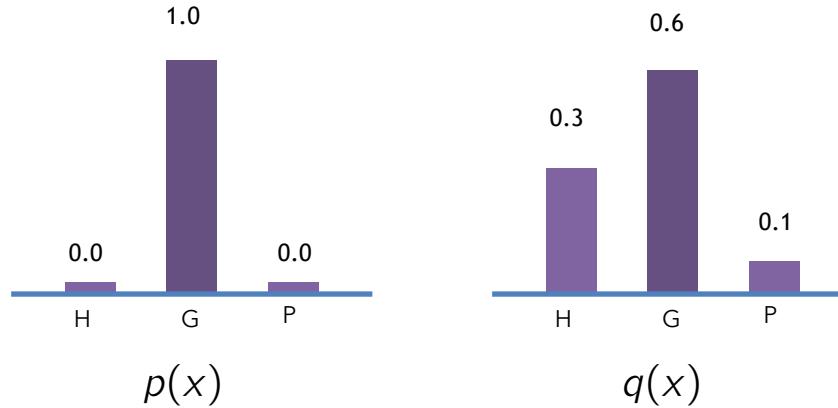
Losses para Clasificación

- En este caso, codificamos las etiquetas de entrenamiento (y) de manera compatible, usando lo que se denomina **one-hot-encoding**.



Divergencias

- Una métrica para comparar dos distribuciones (en nuestro caso predicha y deseada) es lo que en estadística se conoce como una divergencia. Una de las más utilizadas es la **divergencia de Kullback-Leibler** (también llamada *Entropía Relativa*)



$$D_{\text{KL}}(p||q) = \mathbb{E}[p \log(p/q)] = \sum_k p_k \log(p_k/q_k)$$

Divergencia KL y Entropía de Shannon

- Su definición se puede motivar recordando que el modo óptimo (*) de codificar o comprimir un conjunto de símbolos es usando códigos inversamente proporcionales a la probabilidad de cada símbolo

$$x \longrightarrow \boxed{01010101} \quad 1/p(x)$$

de manera que el valor esperado de la longitud de los códigos por transmitir o almacenar es la **Entropía de Shannon**:

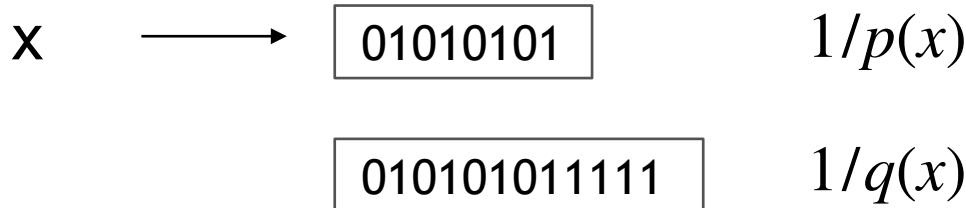
$$H(x) = \sum_x p(x) \log(1/p(x)) = - \sum_x p(x) \log(p(x))$$

(*) Shannon's Source Coding Theorem.

Divergencia KL y Entropía de Shannon

- La divergencia KL es simplemente la pérdida de eficiencia que se produce al usar $q(x)$ en vez de $p(x)$ para codificar o comprimir x

$$\begin{aligned} D_{\text{KL}}(p(x)||q(x)) &= \sum_x p(x) \log(p(x)/q(x)) \\ &= - \sum_x p(x) \log(q(x)) + \sum_x p(x) \log(p(x)) \\ &= - \sum_x p(x) \log(q(x)) - H(x) \end{aligned}$$

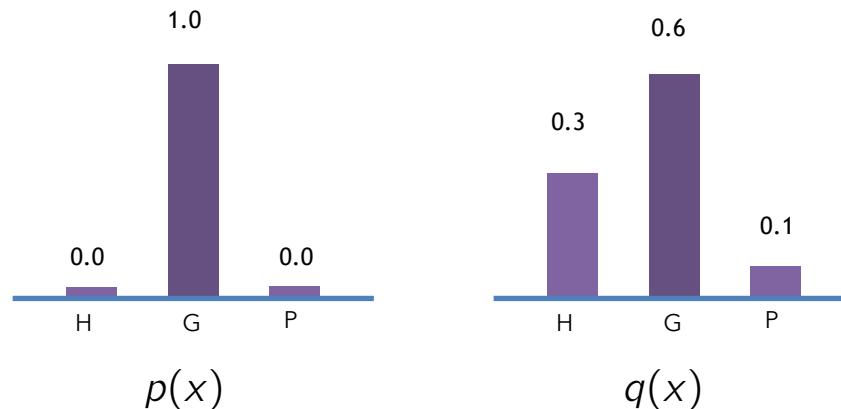


Divergencia KL

- Propiedades:

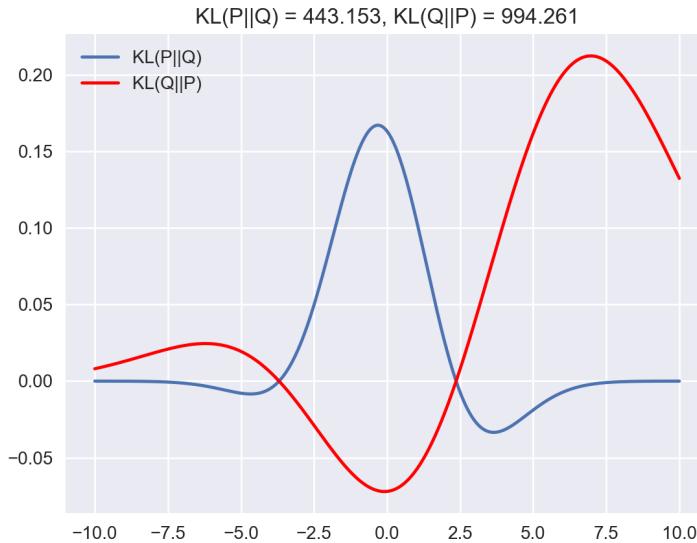
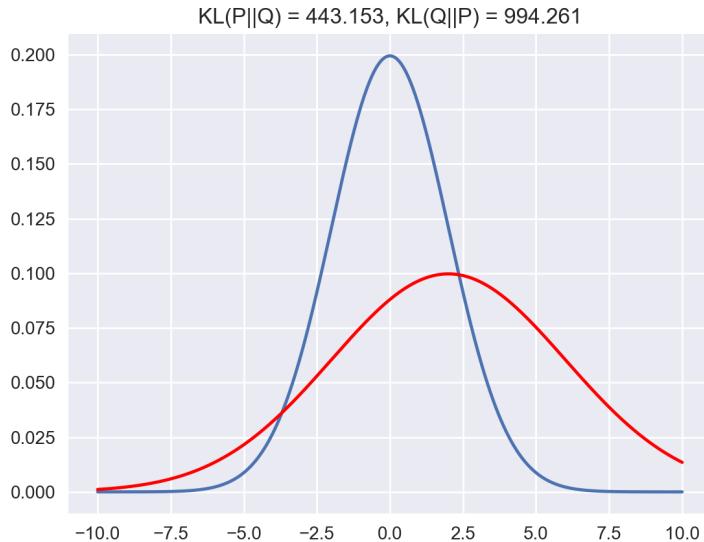
1. $D_{\text{KL}}(p||q) \geq 0$
2. $D_{\text{KL}}(p||q) = 0 \Leftrightarrow p = q$ (p-casi seguramente)
3. $D_{\text{KL}}(p_1p_2||q) = D_{\text{KL}}(p_1||q) + D_{\text{KL}}(p_2||q)$
4. Sean: $p = \alpha p_1 + (1 - \alpha)p_2, q = \alpha q_1 + (1 - \alpha)q_2$, entonces

$$D_{\text{KL}}(p||q) \leq \alpha D_{\text{KL}}(p_1||q_1) + (1 - \alpha)D_{\text{KL}}(p_2||q_2) \quad \forall \alpha \in [0, 1]$$



Divergencia KL

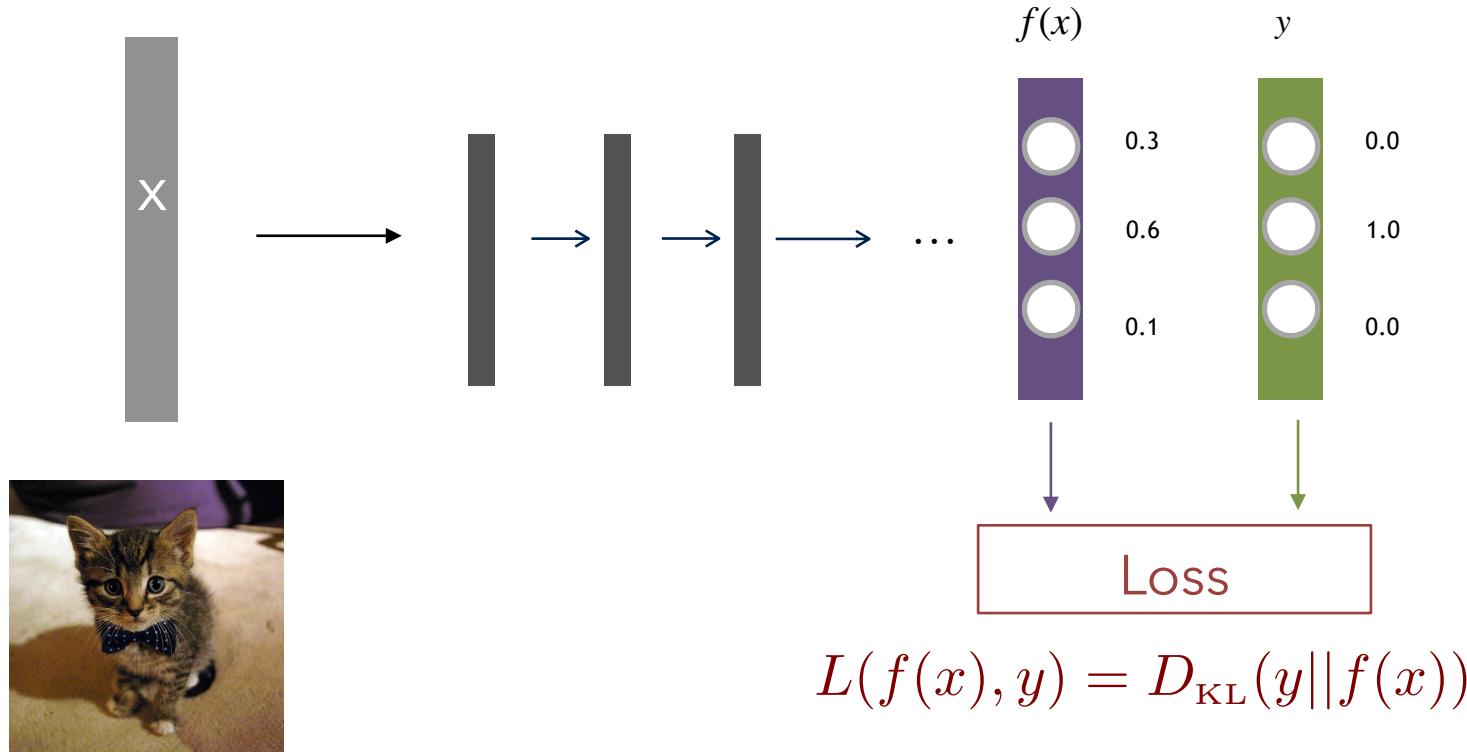
- Notemos que esta divergencia no es simétrica: $D_{\text{KL}}(p||q) \neq D_{\text{KL}}(q||p)$



- Una de las distribuciones se toma como el objetivo (p) y la otra como la aproximación (q). El valor de $q(x)$ cuando $p(x) = 0$ ese irrelevante.

Cross Entropy Loss

- En nuestro caso, el objetivo es el one-hot-encoding de la etiqueta real y la aproximación es la distribución que sale de la red, de modo que podemos usar la divergencia de $f(x)$ respecto a y para cuantificar el error.



Cross Entropy Loss

- Cuando el objetivo es entrenar la red, la función de costo anterior se puede simplificar

$$\begin{aligned} D_{\text{KL}}(y||f(x)) &= \sum_k y_k \log(y_k/f_k(x)) \\ &= \sum_k y_k \log(y_k) - \sum_k y_k \log(f_k(x)) \\ &= \boxed{\sum_k y_k \log(y_k)} + \boxed{\text{CE}(f(x), y)} \end{aligned}$$

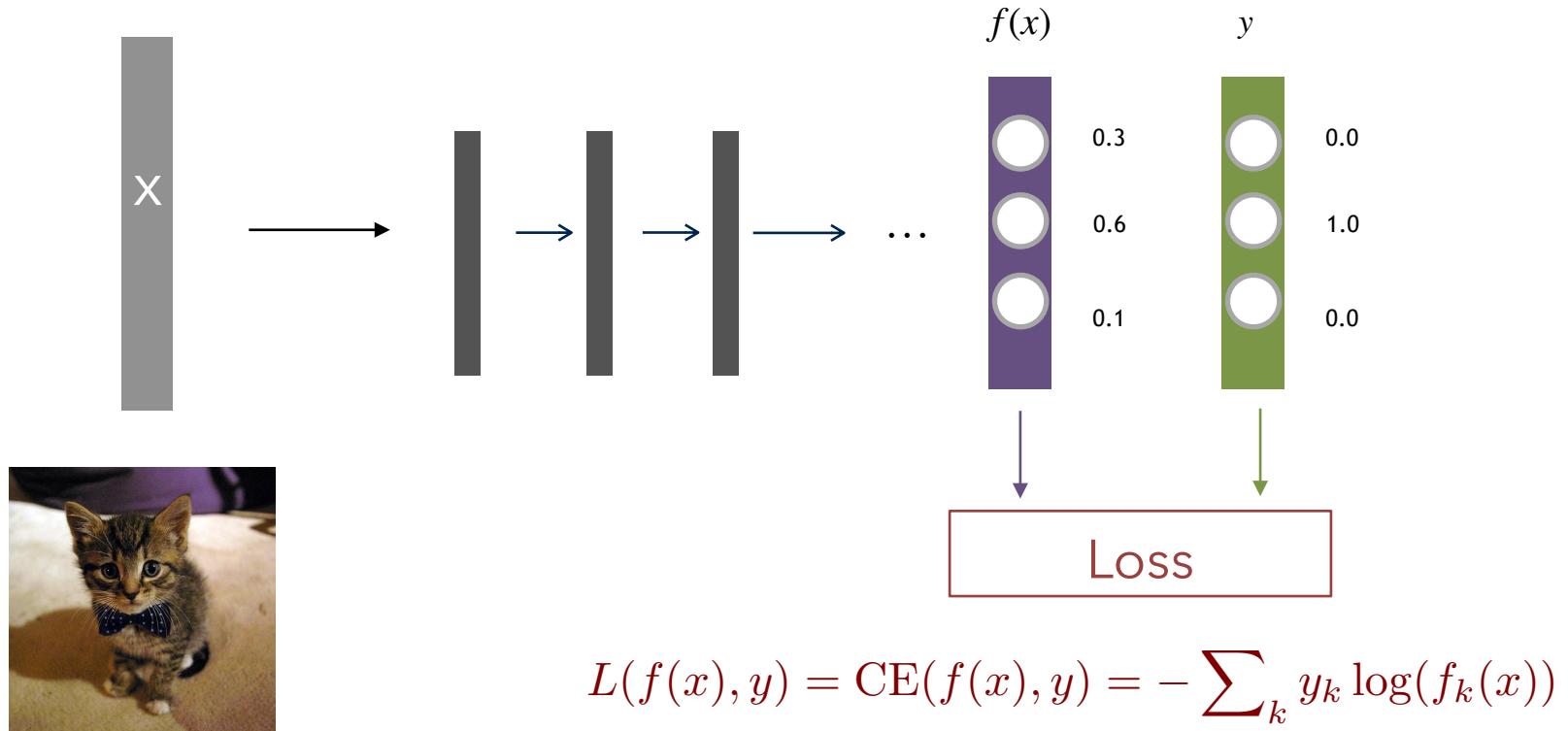
Independiente
de la red

**Cross Entropy
Loss**



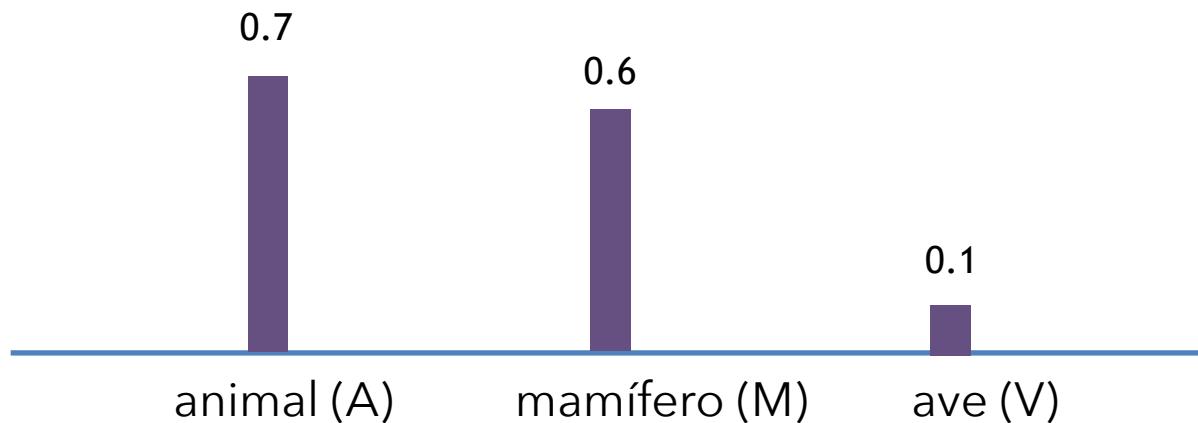
Loss Estándar para Clasificación

- La entropía cruzada es la función de costo más utilizada para entrenar redes neuronales en problemas de clasificación con clases mutuamente excluyentes.



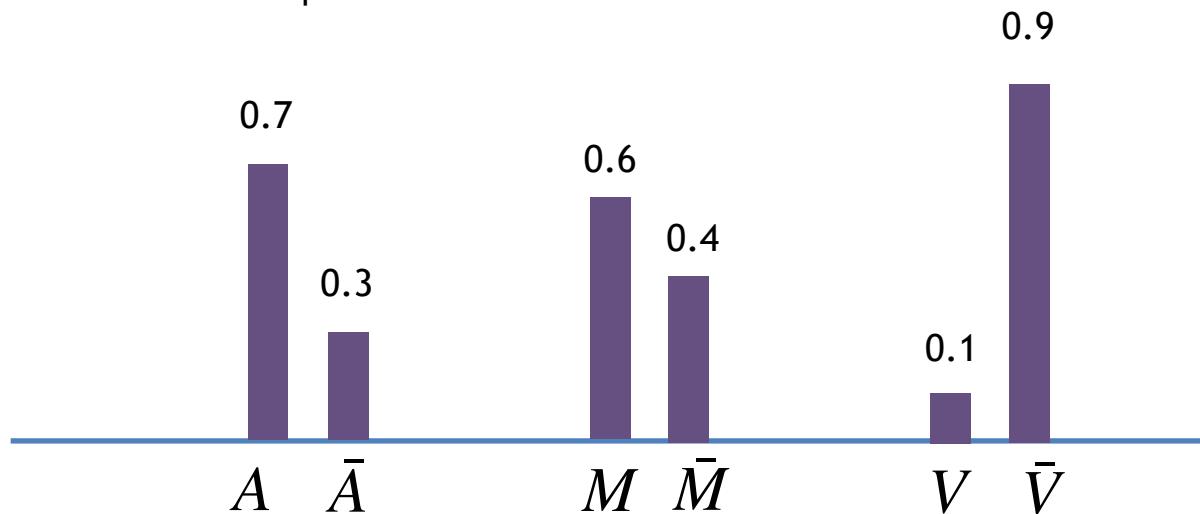
Casos Especiales

- Recodemos que si las clases no son mutuamente excluyentes, la red no predice una distribución sobre todo el conjunto de clases sino que una distribución sobre cada clase independientemente



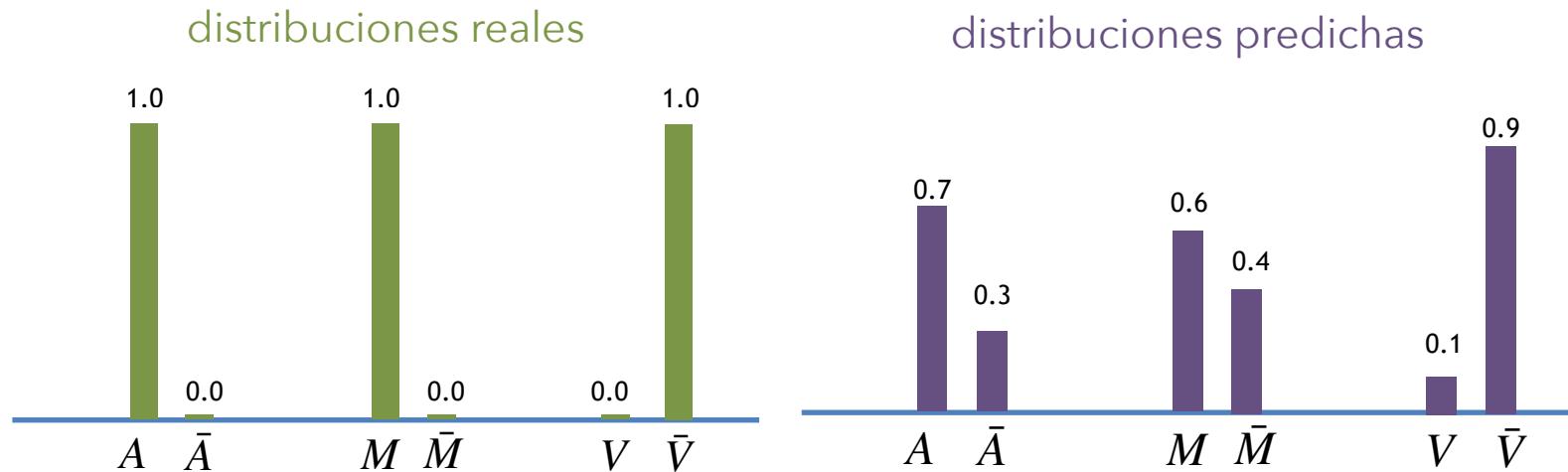
Casos Especiales

- Si clases no son mutuamente excluyentes, la red no predice una distribución sobre todo el conjunto de clases sino que una distribución sobre cada clase independientemente



Casos Especiales

- En este caso, el uso correcto de la divergencia consiste en penalizar la “distancia” entre cada distribución independientemente y luego agregar el resultado de algún modo (e.g. sumando)

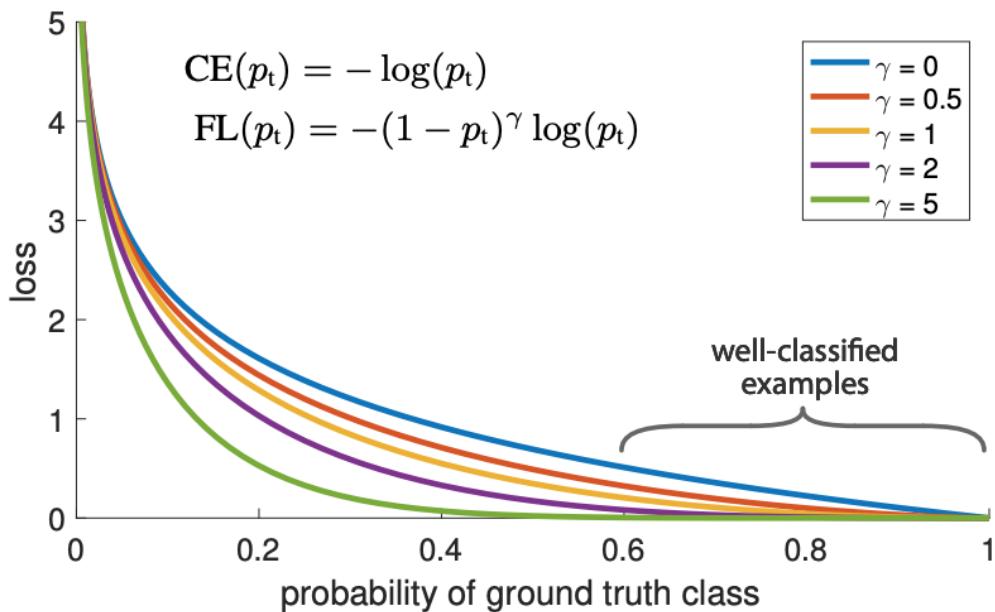


$$\begin{aligned} & \sum_k \text{BCE}(f_k(x), y_k) \\ &= - \sum_k y_k \log(f_k(x)) - \sum_k (1 - y_k) \log(1 - f_k(x)) \end{aligned}$$

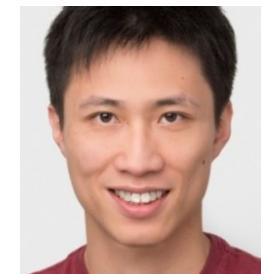
Denominaremos esta loss **binary cross entropy**.

Casos Especiales

- En casos en que las clases difieren mucho en términos de complejidad o representatividad en el conjunto de ejemplos, puede ocurrir que la red no aprenda las clases más complicadas. En estos casos, sería aconsejable optar una loss que castigue más el error en esos casos difíciles para el modelo. Una opción popular en estos días es la **focal loss**.



Lin, Tsung-Yi, et al. "Focal loss for dense object detection." *Proceedings of the IEEE international conference on computer vision*. 2017.



Objetivo de la Red

- Una vez especificada una loss, ¿cuál es el objetivo de la red?
 - Predecir bien (Error 0) todos los ejemplos de entrenamiento?
 - Predecir bien (Error 0) la mayoría de los ejemplos de entrenamiento?
 - Predecir razonablemente bien ($\text{Error} < \epsilon$) todos los ejemplos de entrenamiento?
 - Predecir bien (Error 0) cualquier caso posible?
- Especificar una loss no es suficiente para definir un objetivo.

Objetivo de la Red

- Es claro que lo que nos interesa en verdad no es el error de la red sobre los ejemplos de entrenamiento sino el error sobre los casos que ésta tendrá que decidir en el futuro: casos muy probablemente nos vistos en el conjunto de entrenamiento.
- **¿Cómo formalizamos esta intuición?**



\neq



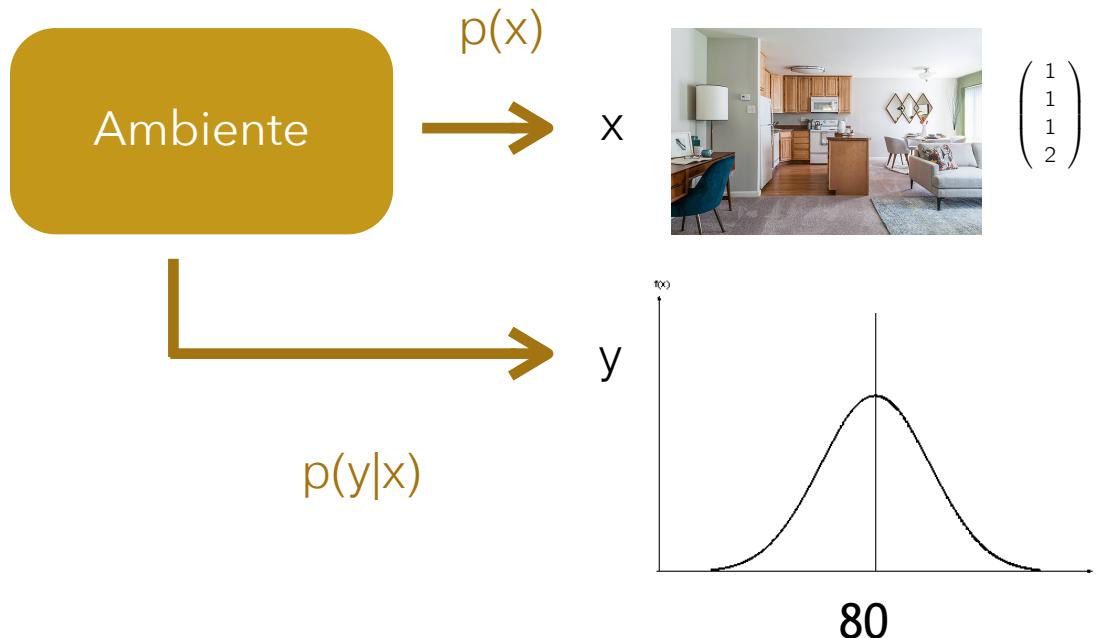
$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \end{pmatrix} \neq$$



$$\begin{pmatrix} 6 \\ 3 \\ 0 \\ 3 \end{pmatrix}$$

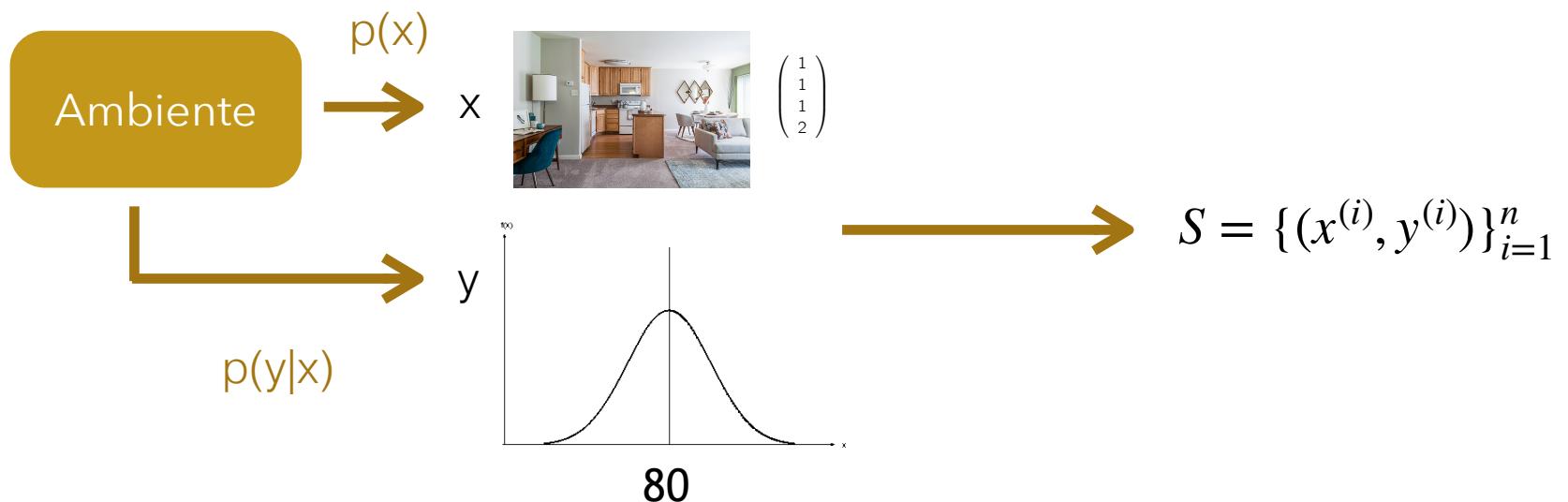
Modelo de Ambiente

- Podemos asumir que el ambiente genera datos de entrada x de acuerdo a una cierta distribución de probabilidad desconocida $P(x)$.
- Por otro lado, las etiquetas y aparecen junto a x de acuerdo a cierta distribución de probabilidad (condicional) $P(y|x)$. El caso determinista (a x le corresponde siempre un único y) es claramente un caso particular.



Error de Predicción

- Si x ocurre con probabilidad (o densidad de probabilidad) $P(x)$ y, dado x , la etiqueta y ocurre con probabilidad (o densidad) $P(y|x)$, entonces el par (x, y) ocurre con probabilidad (o densidad) $P(x, y) = P(y|x)P(x)$.
- Podemos ver el **conjunto de ejemplos como una muestra (IID)** de esta distribución de probabilidad desconocida $P(x, y)$.



Error de Predicción

- Si x ocurre con probabilidad (o densidad de probabilidad) $P(x)$ y, dado x , la etiqueta y ocurre con probabilidad (o densidad) $P(y|x)$, entonces el par (x, y) ocurre con probabilidad (o densidad) $P(x, y) = P(y|x)P(x)$.
- Podemos formalizar el objetivo de la red como el de **minimizar el valor esperado del error** (*):

$$\begin{aligned}\mathbb{E}[L(f(x), y)] &= \sum_{x,y} L(f(x), y)P(x, y) \\ &\stackrel{a.s.}{=} \lim_{n \rightarrow \infty} \frac{1}{n} \sum_i L(f(x^{(i)}), y^{(i)})\end{aligned}$$

- Este objetivo, se denomina el **riesgo** o el **error de predicción** del modelo.

(*) En el caso continuo $\mathbb{E}[L(f(x), y)] = \int L(f(x), y)dP(x, y)$

Error de Entrenamiento

- Naturalmente, en la práctica no conocemos $P(x, y) = P(y|x)P(x)$.
- Esto nos obliga a sustituir el objetivo anterior por un objetivo subrogado que podamos medir en el conjunto (finito) de ejemplos de entrenamiento $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$.
- Lo más común es aproximar el valor esperado usando un promedio sobre el conjunto de ejemplos. Este objetivo se denomina **error de entrenamiento**:

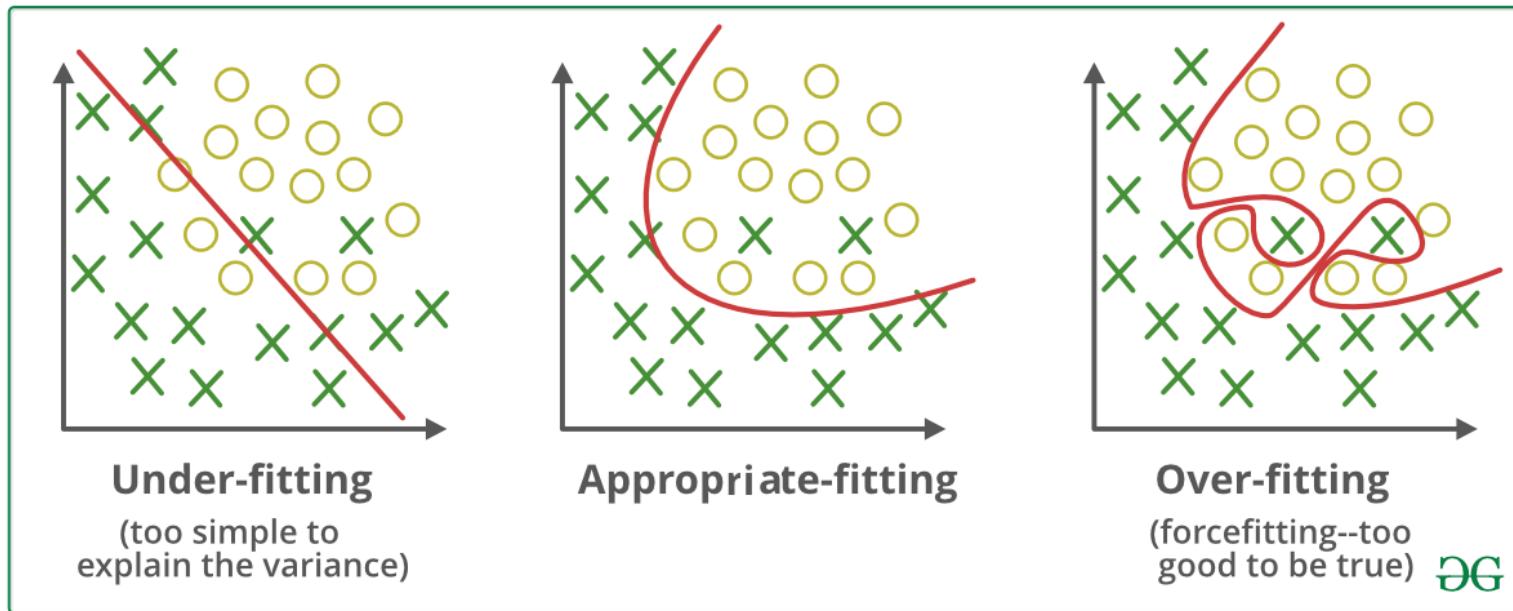
$$\mathbb{E}[L(f(x), y)] \approx \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)})$$

- **Funciona esta aproximación/sustitución?**



Overfitting

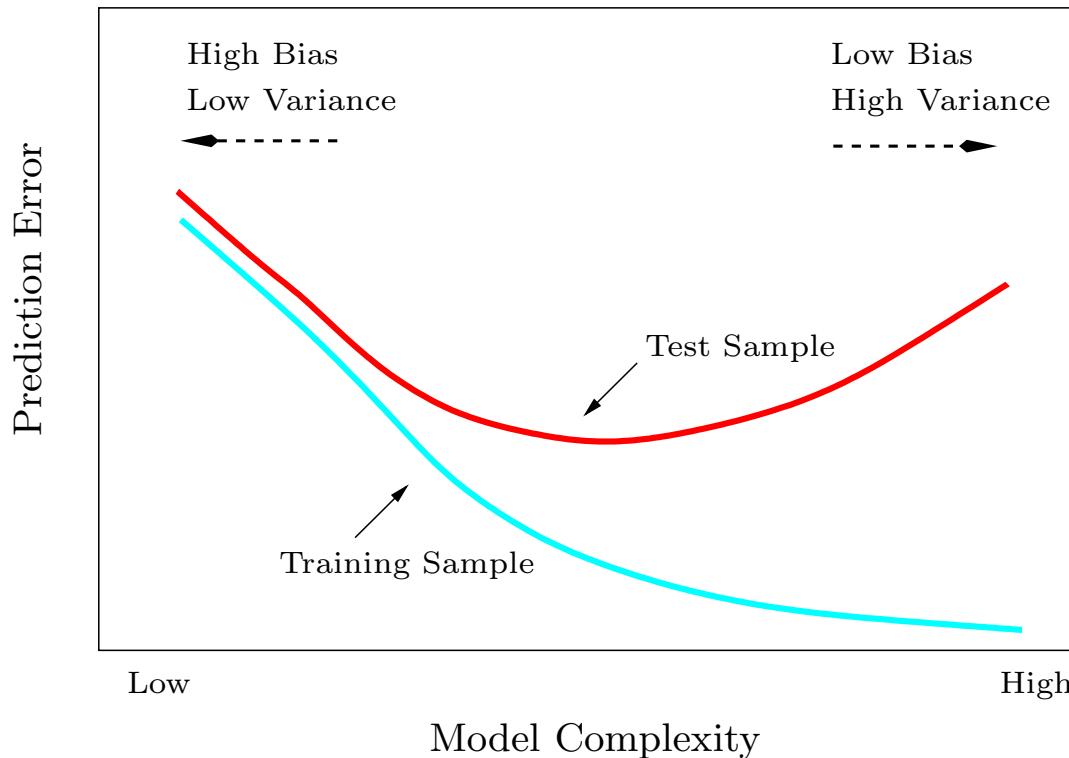
- Funciona esta aproximación/sustitución? No Necesariamente.



- En la práctica depende del número de ejemplos (n) y de la capacidad (expresividad/complejidad) del espacio de funciones candidatas (espacio de hipótesis)

Overfitting

- En la práctica depende del número de ejemplos (n) y de la capacidad (expresividad/complejidad) del espacio de funciones candidatas.



Overfitting

Teorema (Vapnik, 1997)

Sea c la complejidad del espacio de hipótesis (medida mediante la denominada dimensión VC). Entonces, con probabilidad $1 - \eta$

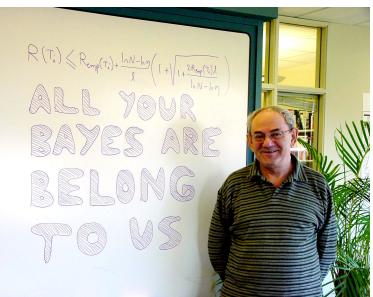
$$\mathbb{E}[L(f(x), y)] < \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)}) + \sqrt{\frac{c \log(2n/c + 1) - \log(\eta/4)}{n}}$$

Error "de pruebas" (riesgo)

Error de entrenamiento

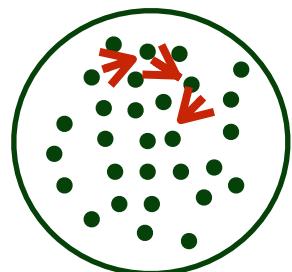
Complejidad

W. Vapnik and A. Chervonenkis (1981) *Necessary and sufficient conditions for the uniform convergence of means to their expectations*. Theory of Probability & Its Applications.



Overfitting

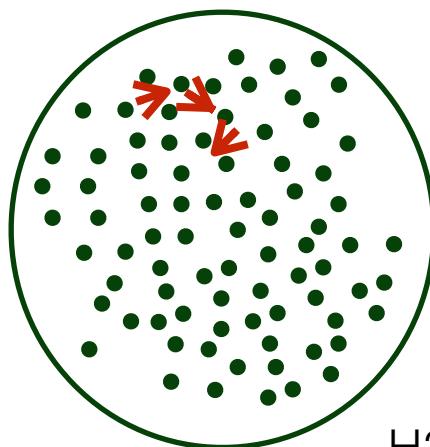
- En la práctica depende del número de ejemplos (n) y de la capacidad (expresividad/complejidad) del espacio de funciones candidatas.



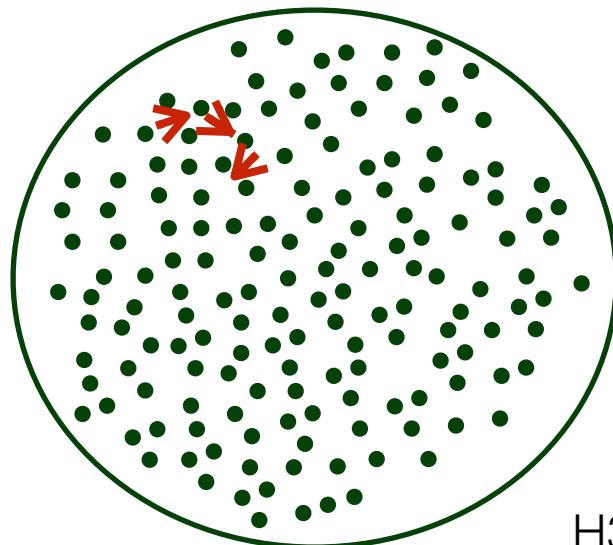
H1



number of examples: n



H2



H3

Regularización

- Estudiaremos más adelante técnicas para prevenir el overfitting, sobre todo en redes profundas (muy expresivas).
- Por ahora, adoptaremos como objetivo de entrenamiento de la red el error de entrenamiento (podemos pensar que tenemos muchos/suficientes ejemplos).

$$\hat{\mathbb{E}}[L(f(x), y)] = \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)})$$

- No debemos olvidar sin embargo, que nuestro objetivo es el error de predicción

$$\begin{aligned}\mathbb{E}[L(f(x), y)] &= \sum_{x,y} L(f(x), y) P(x, y) \\ &\stackrel{a.s.}{=} \lim_{n \rightarrow \infty} \frac{1}{n} \sum_i L(f(x^{(i)}), y^{(i)})\end{aligned}$$



Ejercicios

- Demuestre que si utilizamos la función de costo cuadrática $L(f(x), y) = (f(x) - y)^2$, la solución del problema de aprendizaje es exactamente la función de regresión $r(x) = E(y | x)$.
- Demuestre que si utilizamos la función de costo binaria $L(f(x), y) = I(f(x) \neq y)$ en un problema de clasificación binario (e.g. $y \in \{0,1\}$) la solución del problema de aprendizaje es exactamente el clasificador de Bayes $f(x) = \arg \max_y p(y | x)$.
- Demuestre que la divergencia KL es convexa y use esto para demostrar que la función de costo cross-entropy también lo es.



Entonces

- Entrenar una red neuronal consiste en determinar valores para los pesos y bases del grafo de manera de **minimizar el error de predicción**

$$\mathbb{E}[L(f(x), y)] = \sum_{x,y} L(f(x), y)P(x, y)$$

donde $L : Y \times Y \rightarrow \mathbb{R}_0^+$ es una **función de costo o error** que mide la magnitud $L(f(x), y)$ de una diferencia entre el valor predicho por la red $f(x)$ y la respuesta y que observamos en el ambiente para ese input. Como en la práctica no conocemos $P(x, y)$, es común tomar un conjunto de ejemplos y entrenar a red para minimizar el **error de entrenamiento**

$$\hat{\mathbb{E}}[L(f(x), y)] = \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)})$$

- La elección de L depende del problema. En regresión es común usar el **error cuadrático** y en clasificación la **cross-entropy loss** (binaria o multicategoría).