

Departamento de Física Médica - Centro atómico Bariloche - IB

Función de costo y optimización

Ariel Hernán Curiale
ariel.curiale@cab.cnea.gov.ar

La mayoría de los slides fueron adaptados de Fei Fei Li, J. Johnson y S. Yeung, cs231n, Stanford 2017.



UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO



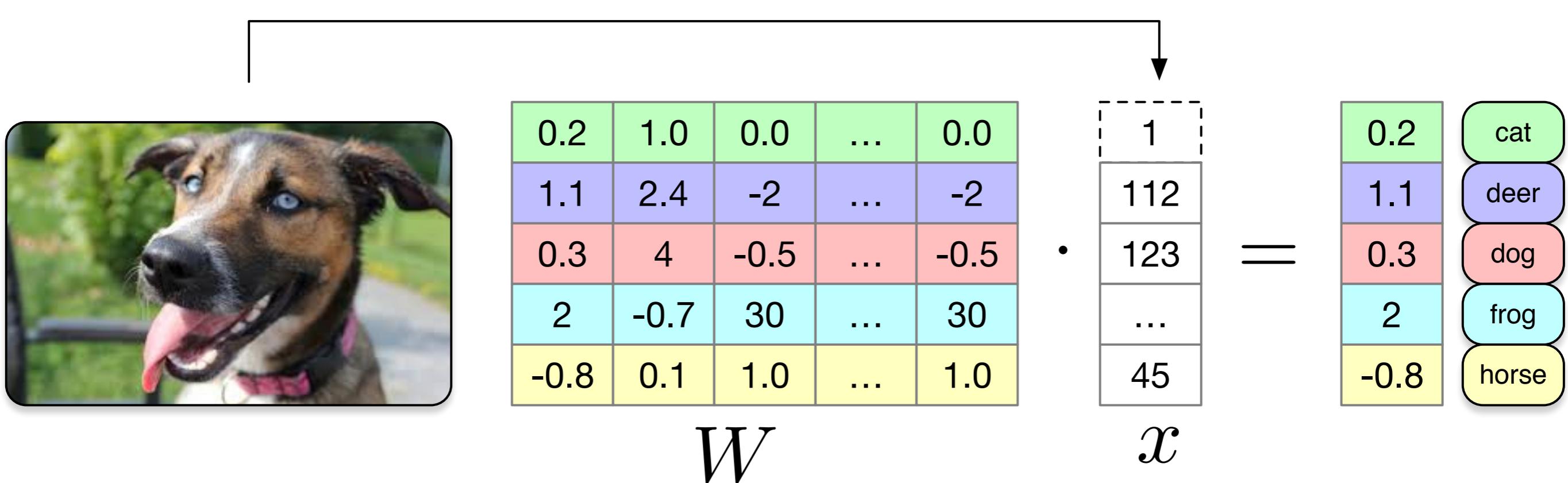
Clasificación de imágenes

- ❖ Nearest Neighbors
- ❖ K-Nearest Neighbors
- ❖ Linear Classifier :
 - ❖ Support Vector Machine
 - ❖ Softmax Classifier

Nos quedaba pendiente

Clasificación lineal

$$f(x, W) = Wx = \hat{W}x + b \quad \xrightarrow{\text{bias}} \quad x_0 = 1$$



¿Cómo calculamos los pesos?

¿Porque no usar cuadrados mínimos?

Loss Function, Cost Function, Objective

Función de costo

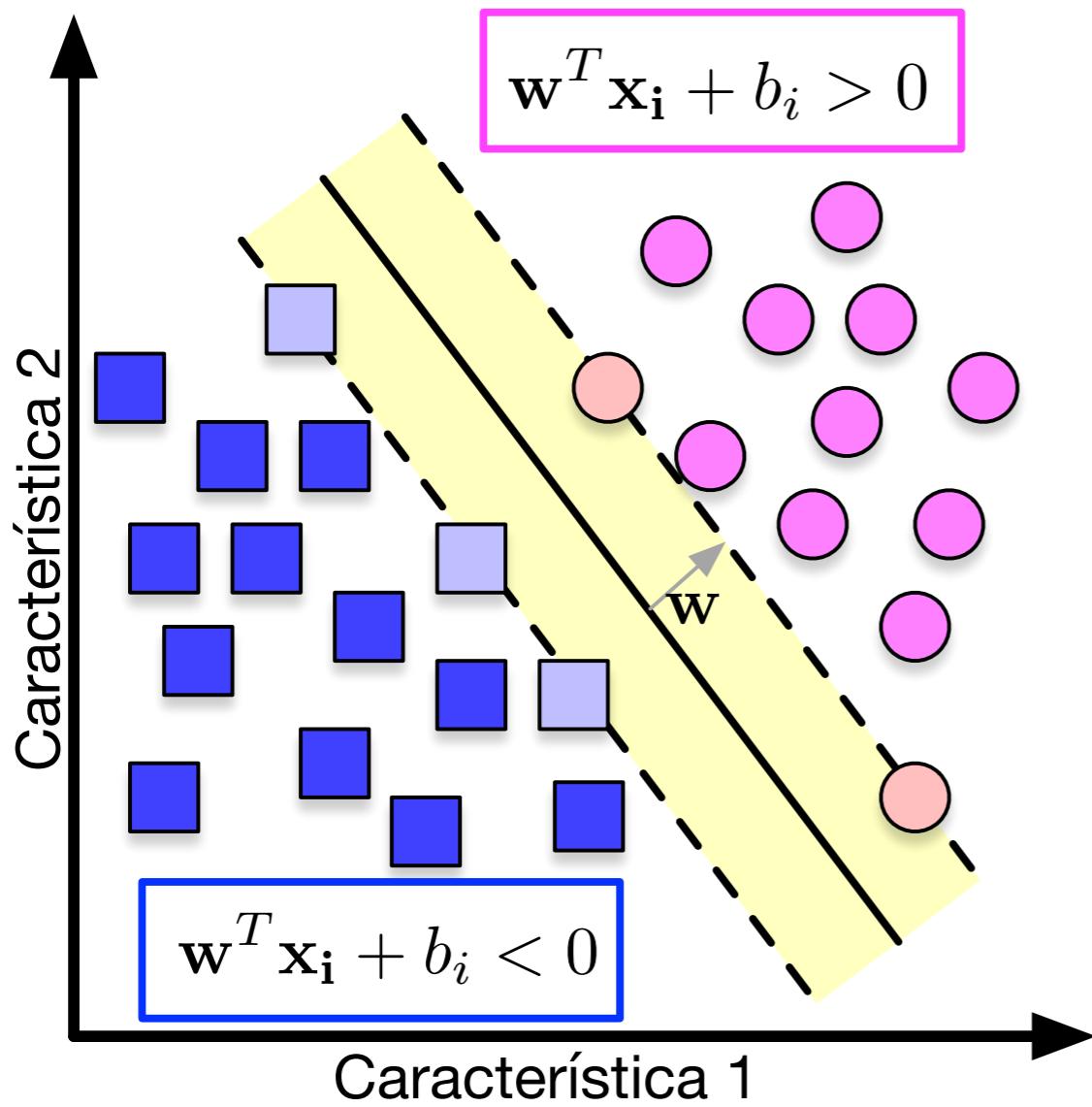
- ❖ Función de costo: loss function, cost function, objective
 - ❖ Nos da una idea de cuán bien estamos resolviendo el problema dado un conjunto de datos $\{(x_i, y_i)\}_{i=1}^N$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

- ❖ Existen diferentes formas de definir esta función de costo y dependen fuertemente del tipo de problema.

Support Vector Machine

- ❖ Idea general de Support Vector Machine: encontrar un hiperplano de margen máximo



Solución: $\mathbf{w}^T \mathbf{x}_i + b_i = 0$

Función de clasificación:

$$f(\mathbf{x}) = \text{sig}(\mathbf{w}^T \mathbf{x} + b_i = 0)$$

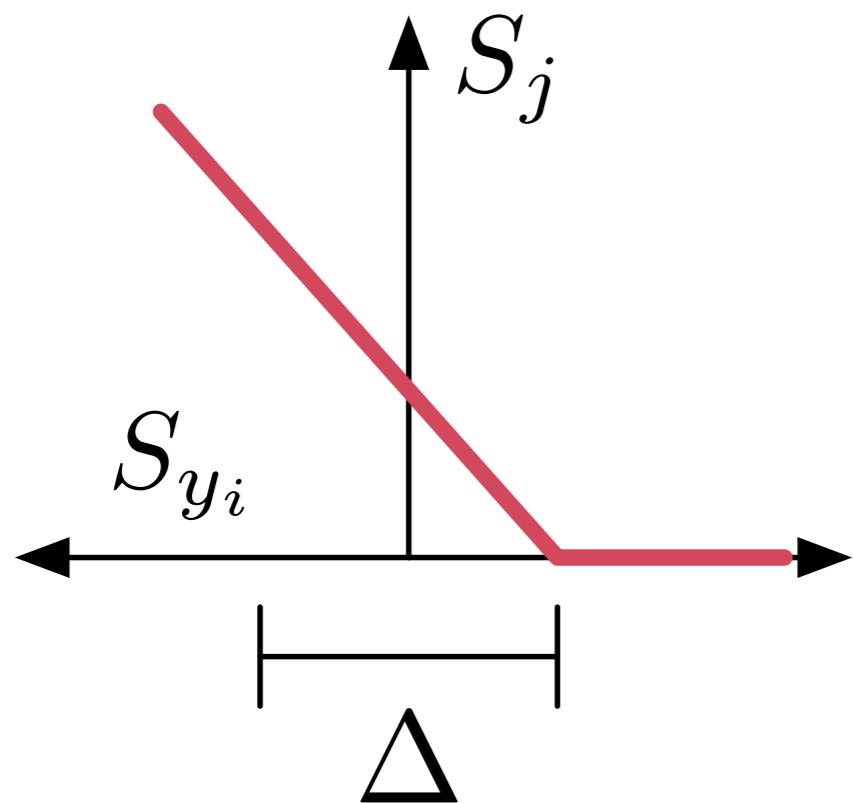
- ❖ SVM es un clasificador lineal que optimiza la función de costo bisagra o **Hinge loss** con una regularización L2

Recordar: Aclarar problemas linealmente separables con kernels al final de todo

Hinge loss: Multi Class SVM Loss

- ❖ Hinge loss: busca que la clasificación correcta de la imagen tenga un valor mayor al de la incorrecta considerando un pequeño margen.

valores obtenido para la
clasificación verdadera



$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

donde $S = f(x, W)$

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9		

$$\begin{aligned} \Delta &= 1 \\ L &= \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta) \\ &= \max(0, 5.1 - 3.2 + 1) + \\ &\quad \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 \end{aligned}$$

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9

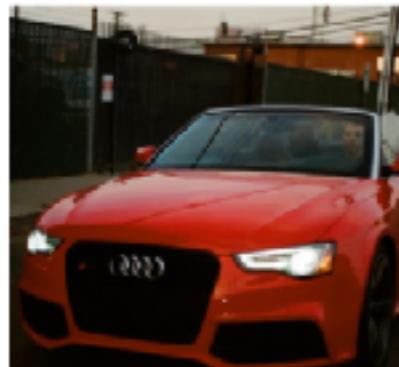
$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

$$\text{loss} = 5.27$$

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9
			loss = 5.27

¿Usamos 1 o 100?
¿De que depende?

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9
			loss = 5.27

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

¿Qué pasa con el valor de loss si cambian un poco los resultados para el ej. del auto?

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9
			loss = 5.27

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

¿Qué valores mínimos y máximos puede tomar?

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

Si delta=1 y W es pequeño,
entonces s ~ 0.

Siendo esto así, ¿qué valor
tiene loss ?

$$\text{loss} = 5.27$$

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9
			loss = 5.27

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

¿Qué pasa si la suma es sobre todas las clases?

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9
			loss = 5.27

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

¿Qué pasa si usamos el promedio en lugar de la suma?

Support Vector Machine

- ❖ Supongamos tenemos sólo 3 clases y 3 ejemplos



Img. source: Stanford CS231n 2017

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
L:	2.9	0.0	12.9

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$

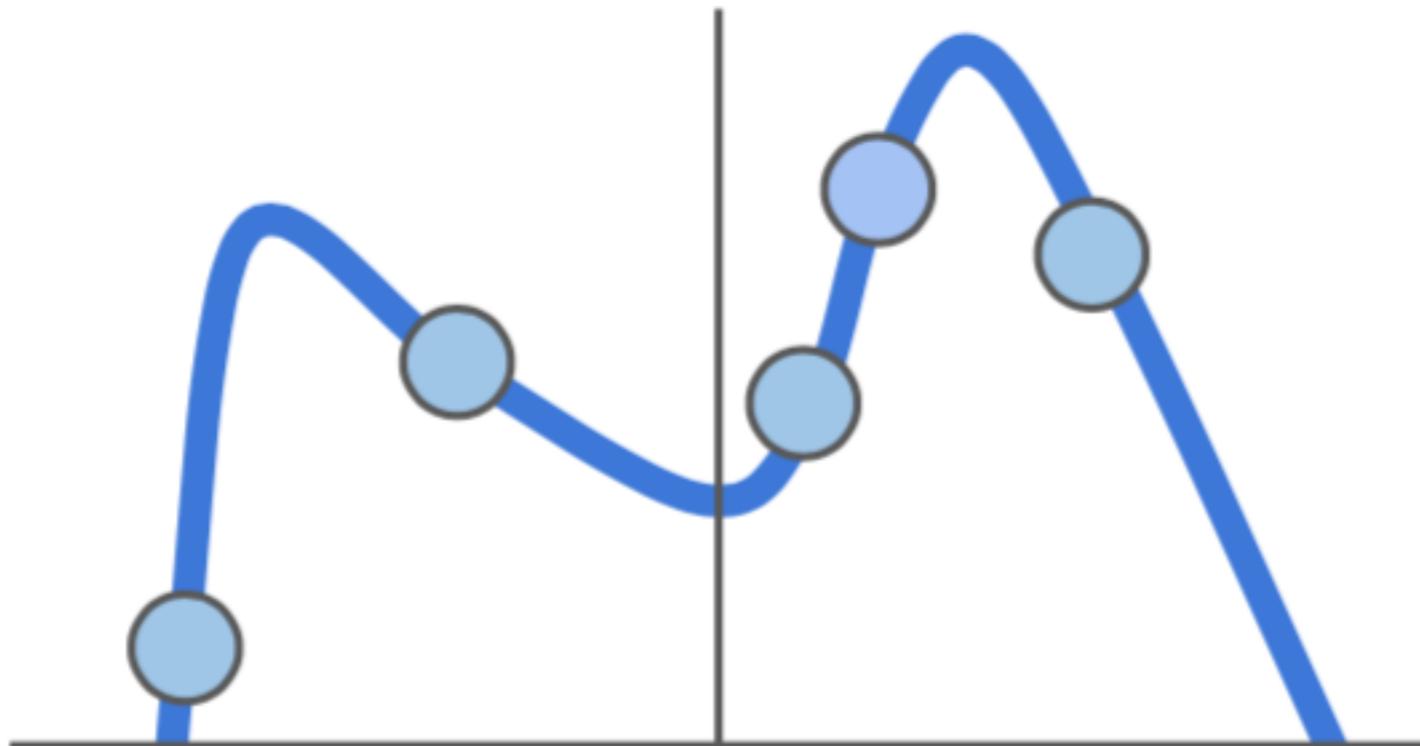
¿Qué pasa si usamos esta métrica ?

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + 1)^2$$

$$\text{loss} = 5.27$$

Regularización

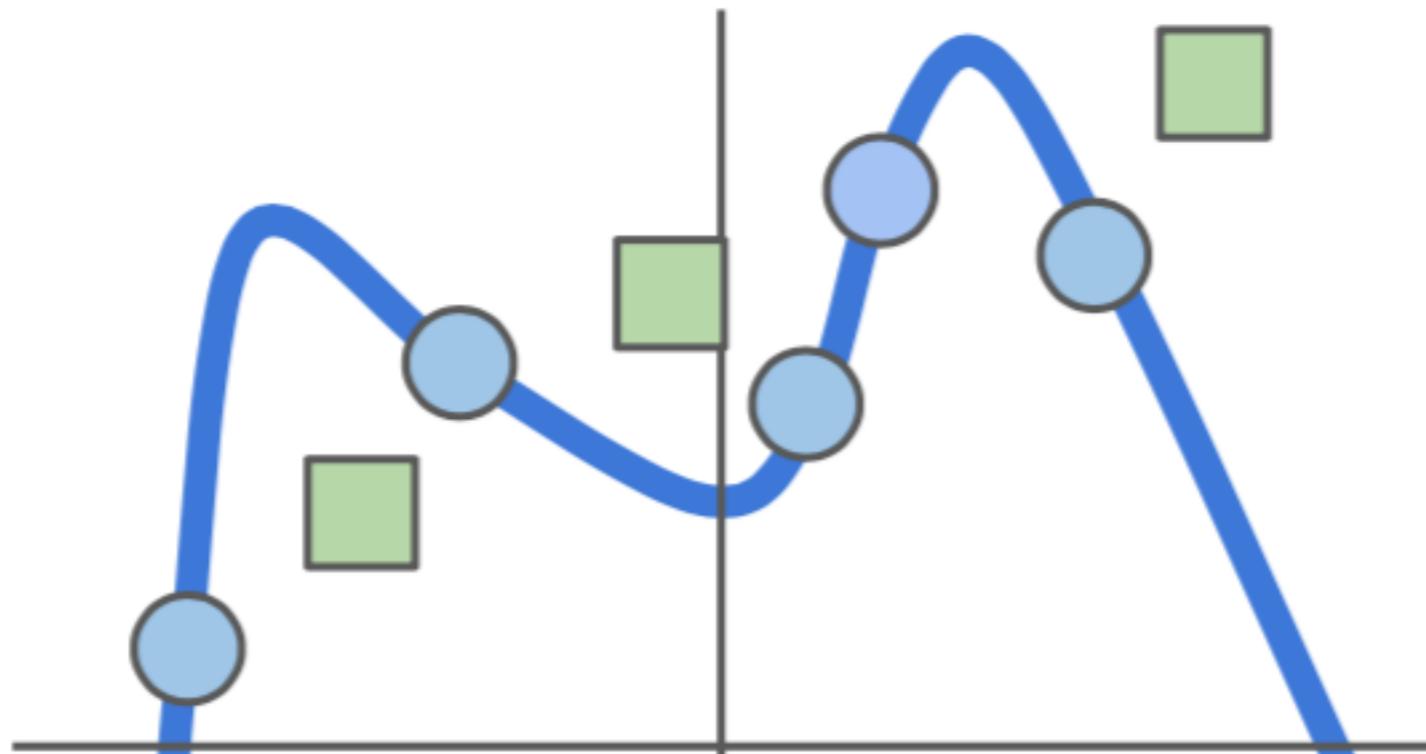
$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$



Img. source: Stanford CS231n 2017

Regularización

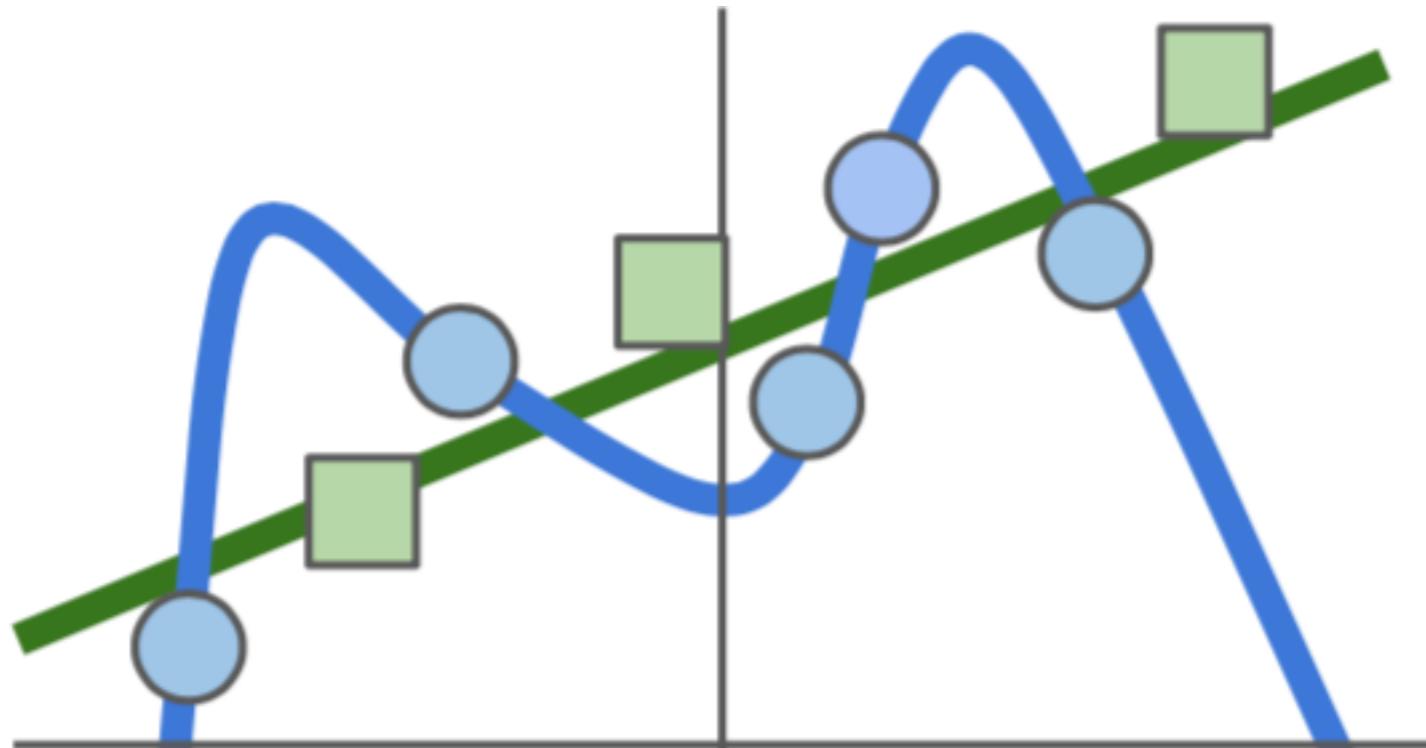
$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$



Img. source: Stanford CS231n 2017

Regularización

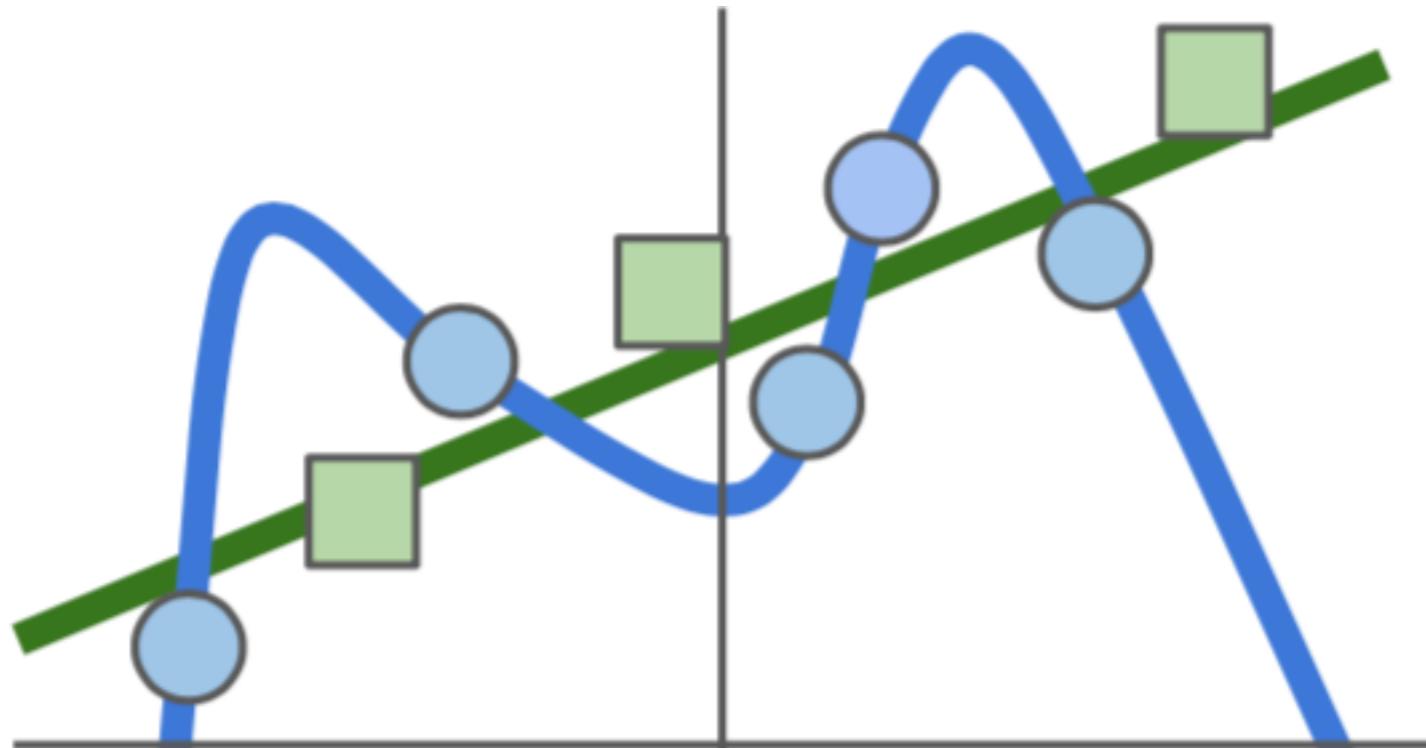
$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i)$$



Img. source: Stanford CS231n 2017

Regularización

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W)$$



Img. source: Stanford CS231n 2017

Regularización

$$\text{loss} = \frac{1}{N} \sum_i \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) + \lambda R(W)$$

- ❖ L2 $R(W) = \sum_{k,l} W_{k,l}^2$ 0.5 * L2
- ❖ L1 $R(W) = \sum_{k,l} |W_{k,l}|$
- ❖ Elastic net (L1 + L2) $R(W) = \sum_{k,l} \beta W_{k,l}^2 + |W_{k,l}|$
- ❖ Max norm regularization
- ❖ Dropout
- ❖ Batch normalization también aplica un efecto de regularización

Clasificación de imágenes

- ❖ Nearest Neighbors
- ❖ K-Nearest Neighbors
- ❖ Linear Classifier :
 - ❖ Support Vector Machine
 - ❖ Softmax Classifier

Nos quedaba pendiente

Softmax Classifier

- ❖ Al igual que SVM, en Softmax la función de activación es lineal

$$f(x, W) = Wx$$

done cambia su función de costo, si lo comparamos con SVM

- ❖ Softmax es la generalización del clasificador conocido como **Binary Logistic Regression**
- ❖ La función de costo Softmax también es conocida como **Categorical Cross Entropy**

Softmax Classifier

- ❖ Interpretando los resultados como log probabilidades no normalizadas para cada clase tenemos

$$P(Y = j | X = x_i) = \frac{e^{f_j(x_i)}}{\sum_k e^{f_k(x_i)}}$$

↓ softmax

$$f_i(z) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- ❖ Ahora buscamos maximizar la log likelihood

$$L = -\log P(Y = j | X = x_i)$$

Softmax Classifier

- ❖ Al igual que SVM sigue siendo un clasificador lineal (\mathbf{Wx}), pero en lugar de utilizar la función de costo de Hinge utilizamos la log likelihood:

$$L = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) = -f_{y_i} + \log \left(\sum_j e^{f_j} \right)$$

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W)$$

Softmax Classifier



$$L = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) = -f_{y_i} + \log \left(\sum_j e^{f_j} \right)$$

unnormalized probabilities

cat
car
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

$$\rightarrow L = -\log(0.13) = 0.89$$

unnormalized log probabilities

probabilities

Softmax Classifier

- ❖ Teoría de la información: La entropía cruzada entre la distribución real p y la distribución estimada q se define

$$H(p, q) = - \sum_x p(x) \log q(x)$$

- ❖ El clasificador softmax minimiza la entropía cruzada entre las probabilidades estimadas (log prob. normalizada) y la distribución real (i.e $p=(0,\dots,1,\dots,0)$)
 - ❖ En otras palabras, esta función de costo busca que la distribución estimada tengan su centro de masa en la clasificación correcta

SVM vs Softmax

- ❖ SVM: Sólo interesa que la correcta gane, sin importar la diferencia entre las otras clases.
- ❖ Softmax: Busca mejorar la relación entre las clases de la clasificación ya que intenta maximizar la probabilidad de la correcta (que sea 1) y minimizar la de la incorrecta a cero.

- ❖ Hasta acá definimos tres formas de clasificar imágenes, KNN, SVM y Softmax.
- ❖ SVM y Softmax son ambos lineales y su principal diferencia radica en la función de costo que utilizan Hinge vs Categorical Cross Entropy
- ❖ También vimos el efecto de la regularización y posibles formas de aplicarla (L1, L2 dropouts, ...)

¿Cómo calculamos los pesos o parámetro de nuestro modelo de clasificación lineal?

Optimización

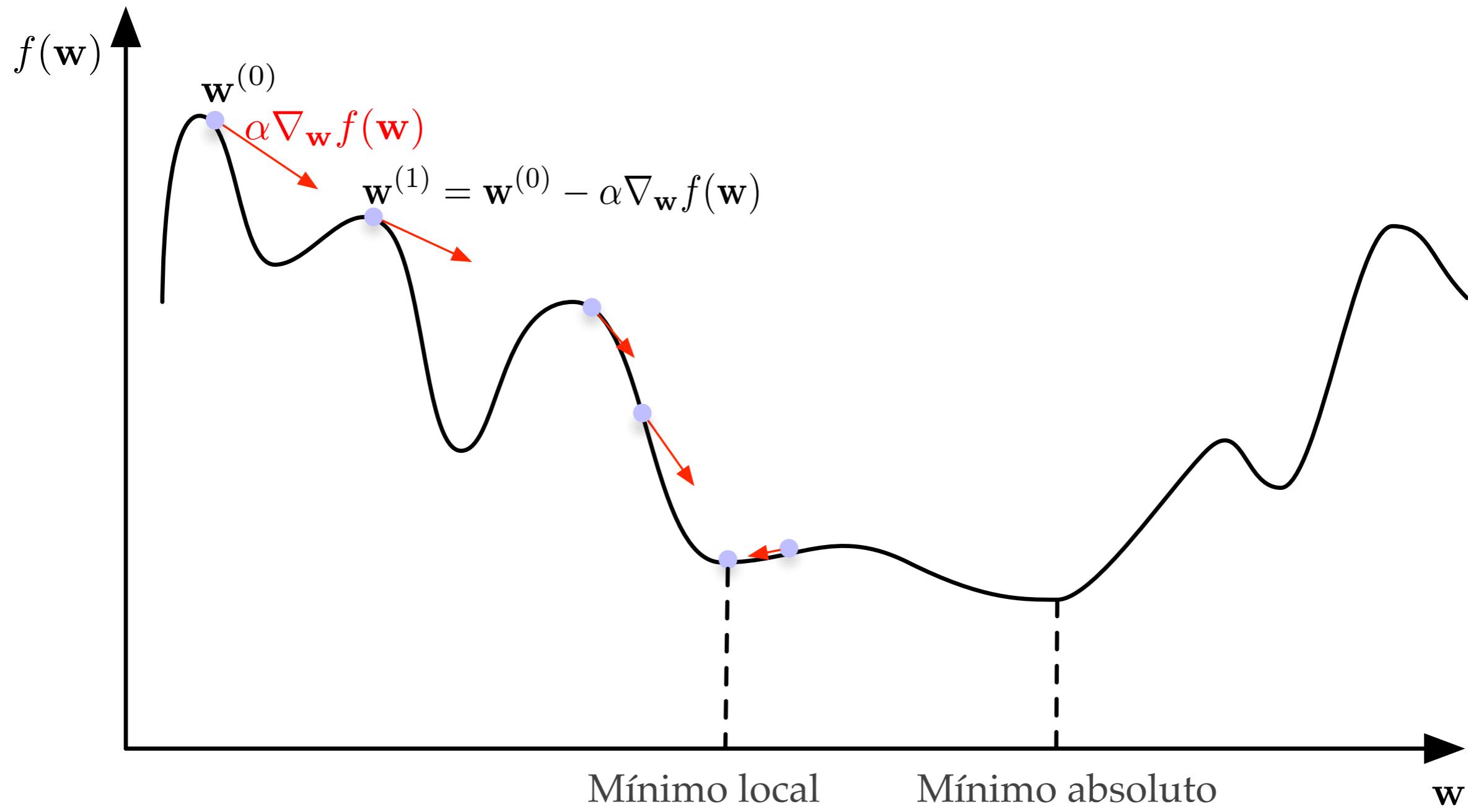
Optimización



Copyright: Michael Werlberger

Optimización

❖ Gradiente descendente



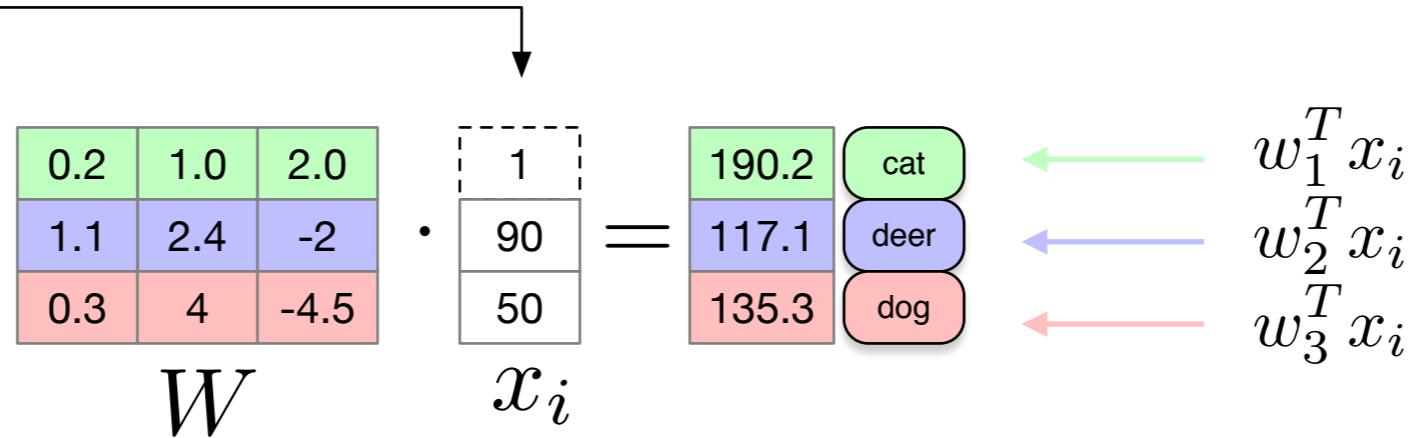
Optimización

- ❖ Buscamos minimizar la función de costo

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W)$$

- ❖ SVM
$$L = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + \Delta)$$

Optimización



$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W)$$

$$L = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

$$\text{loss} = \frac{1}{2} [(\max(0, w_1^T x_1 - w_2^T x_1 + \Delta) + \max(0, w_3^T x_1 - w_2^T x_1 + \Delta)) \leftarrow y_1 = 2]$$

$$+ (\max(0, w_1^T x_2 - w_3^T x_2 + \Delta) + \max(0, w_2^T x_2 - w_3^T x_2 + \Delta)) \leftarrow y_2 = 3$$

$$+ \lambda R(W)$$

Corresponde a los pesos de la clase verdadera, y_1=2

gradiente descendente $W = W - \alpha \nabla_W \text{loss}$

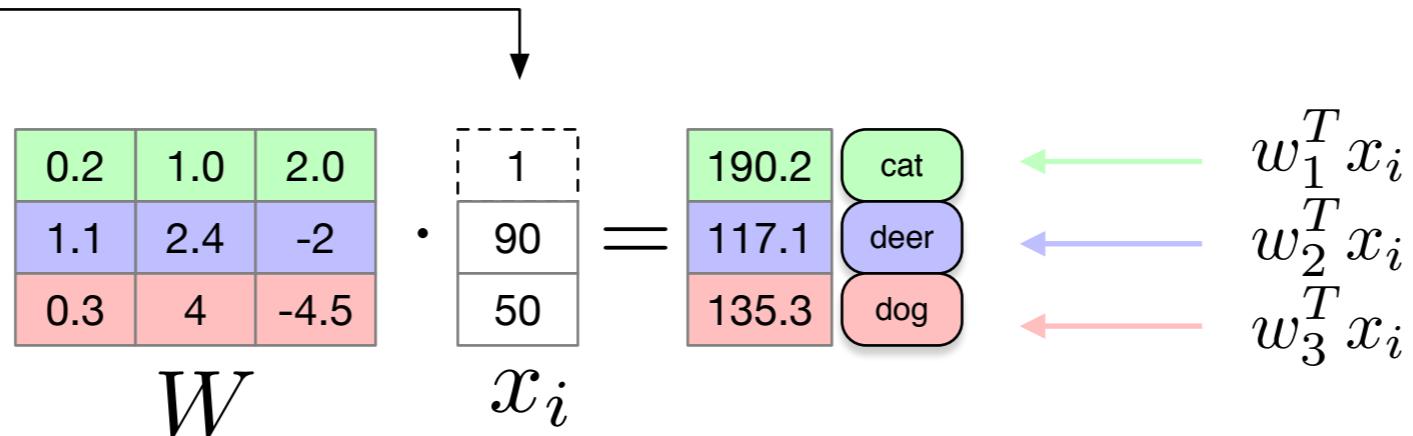
Notar que se escala la imagen “1” de una forma en particular

$$\nabla_W \text{loss} = \frac{1}{2} (\nabla_W L_1 + \nabla_W L_2) + \lambda \nabla_W R(W)$$

Primer componente de la imagen “1”

$$\nabla_W L_1 = \begin{pmatrix} \mathbb{I}(w_1^T x_1 - w_2^T x_1 + \Delta)(x_1)_1 & \cdots & \mathbb{I}(w_1^T x_1 - w_2^T x_1 + \Delta)(x_1)_3 \\ -\mathbb{I}(w_1^T x_1 - w_2^T x_1 + \Delta)(x_1)_1 - \mathbb{I}(w_3^T x_1 - w_2^T x_1 + \Delta)(x_1)_1 & \cdots & \cdots \\ \mathbb{I}(w_3^T x_1 - w_2^T x_1 + \Delta)(x_1)_1 & \cdots & \mathbb{I}(w_3^T x_1 - w_2^T x_1 + \Delta)(x_1)_3 \end{pmatrix}$$

Optimización



$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W)$$

$$L = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

❖ gradiente descendente $W = W - \alpha \nabla_W \text{loss}$

¡¡ Podemos evitar un for con un producto matricial !!

$$\nabla_W \text{loss} = \frac{1}{2} (\nabla_W L_1 + \nabla_W L_2) + \lambda \nabla_W R(W)$$

```

2 # ---- Evitar loops
3 # Vamos a utilizar el producto punto para sumar los ejemplos
4 # escalados como indica el gradiente que esta almacenado en binary
5 dW = x_batch.T.dot(binary)
6 dW /= x_batch.shape[0]
7 # sumamos el gradiente de la regularizacion
8 sW += reg * self.W

```

Optimización

- ❖ Gradiente descendente: $W = W - \alpha \nabla_W \text{loss}$

$$\nabla_W \text{loss} = \frac{1}{N} \sum_{k=1}^N \nabla_W L$$



0.2	1.0	0.0	...	0.0
1.1	2.4	-2	...	-2
0.3	4	-0.5	...	-0.5
2	-0.7	30	...	30
-0.8	0.1	1.0	...	1.0

W

1
112
123
...
45

x_i

$$L = \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta)$$

$$L = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

0.2	cat
1.1	deer
0.3	dog
2	frog
-0.8	horse

$$\frac{\partial L}{\partial w_{ij}} = (x_i)_j$$

$$(\nabla_W L)_{y_i} = - \sum_{j \neq y_i} \mathbb{I}(w_j^T x_i - w_{y_i}^T x_i + \Delta) x_i$$

$$(\nabla_W L)_j = \mathbb{I}(w_j^T x_i - w_{y_i}^T x_i + \Delta) x_i$$

Batch Gradient Descent (BGD)

$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W)$$

- ❖ En lugar de tomar todo el conjunto de datos de entrenamiento lo partimos en mini-batches (32 / 64 / 128)

```
2 # ----- Naive SGD implementation
3 tol = 1e-3
4 delta = np.inf
5 while delta > tol:
6     data_batch = sample_data(training_data, batch_size)
7     old = loss_fun(data_batch, weights)
8     w_grads = evaluate_gradient(loss_fun, data_batch, weights)
9     weights += - step_size * w_grads
10    delta = np.abs(old - loss_fun(data_batch, weights))
```

Stochastic Gradient Descent (SGD)

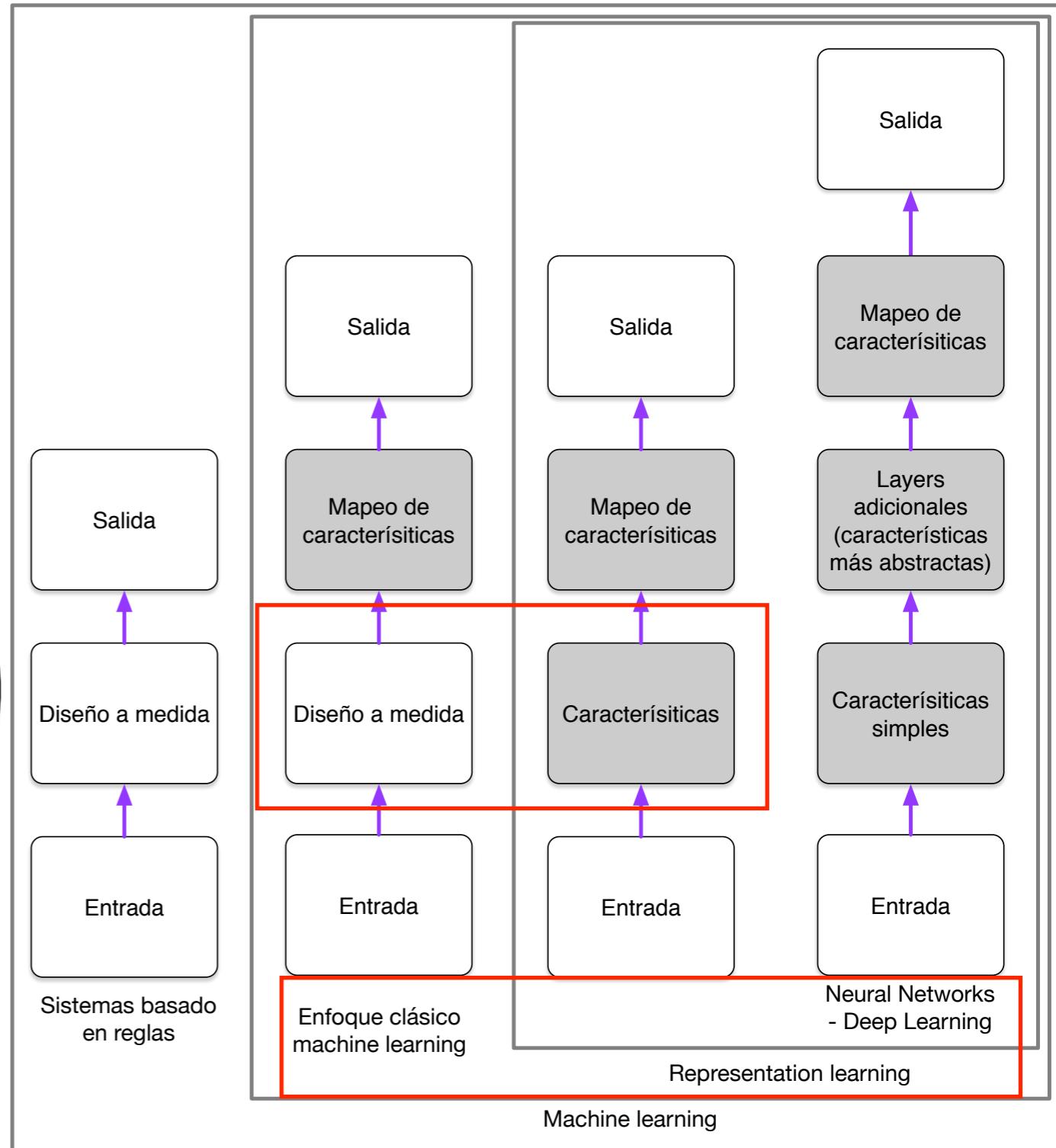
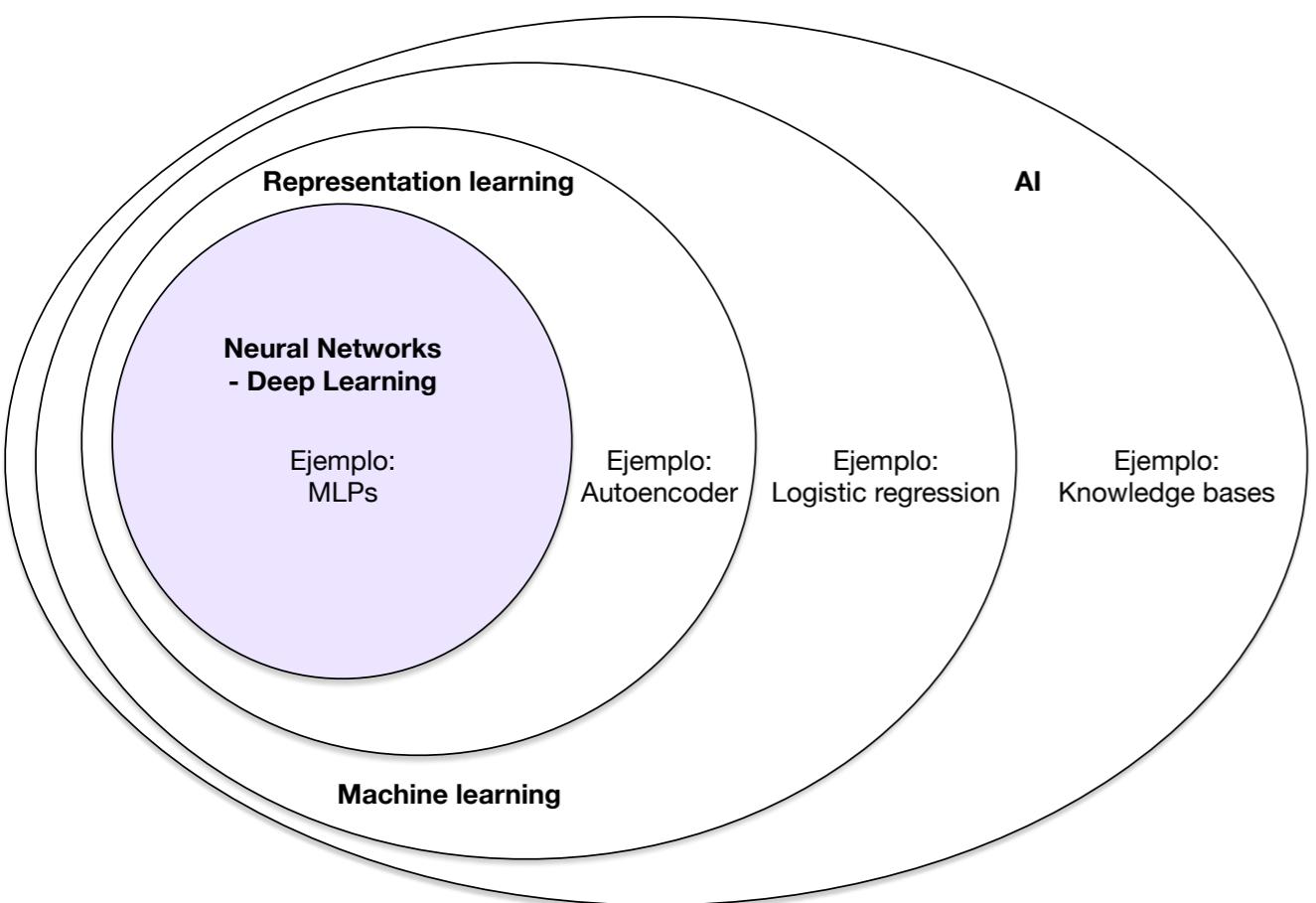
$$\text{loss} = \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W)$$

- ❖ mini-batch = 1
- ❖ Es común que se diga SGD cuando en realidad es BGD

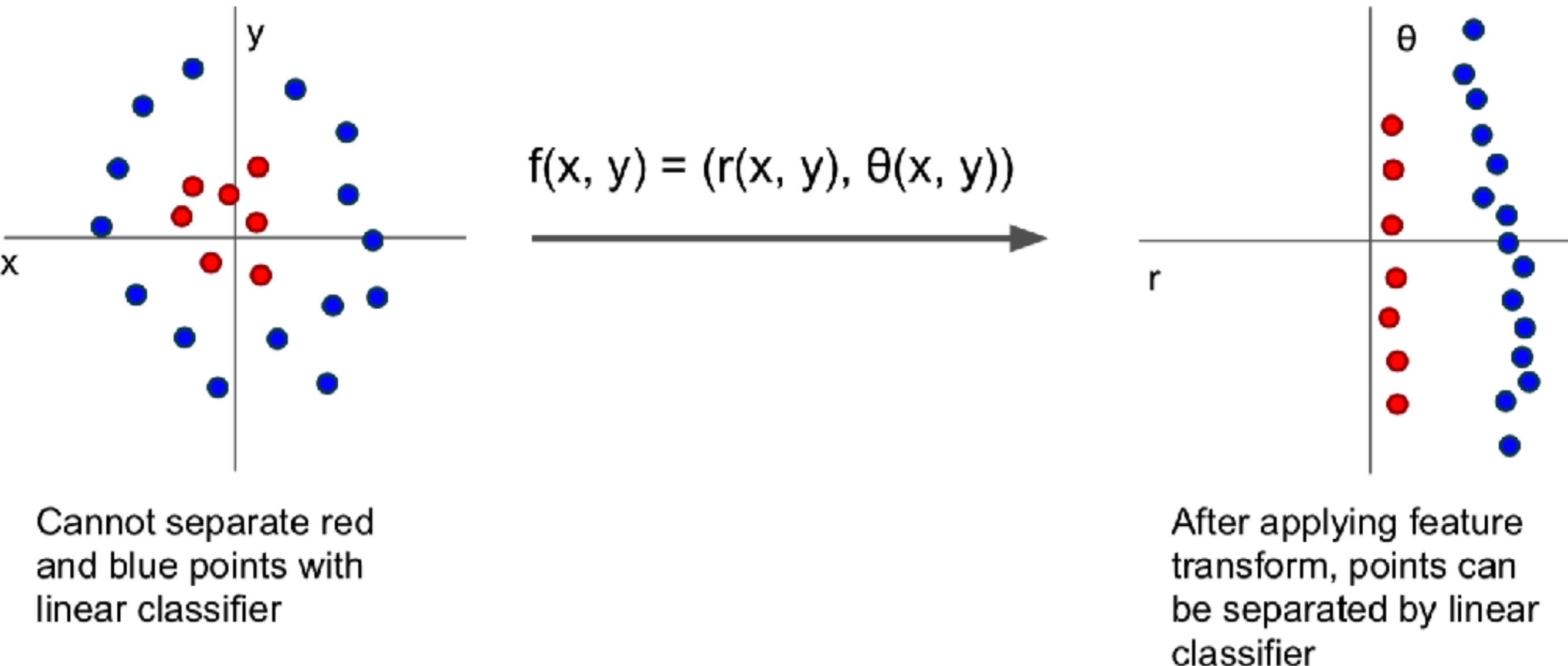
```
2 # ----- Naive SGD implementation
3 tol = 1e-3
4 delta = np.inf
5 while delta > tol:
6     data_batch = sample_data(training_data, batch_size)
7     old = loss_fun(data_batch, weights)
8     w_grads = evaluate_gradient(loss_fun, data_batch, weights)
9     weights += - step_size * w_grads
10    delta = np.abs(old - loss_fun(data_batch, weights))
```

Recordemos

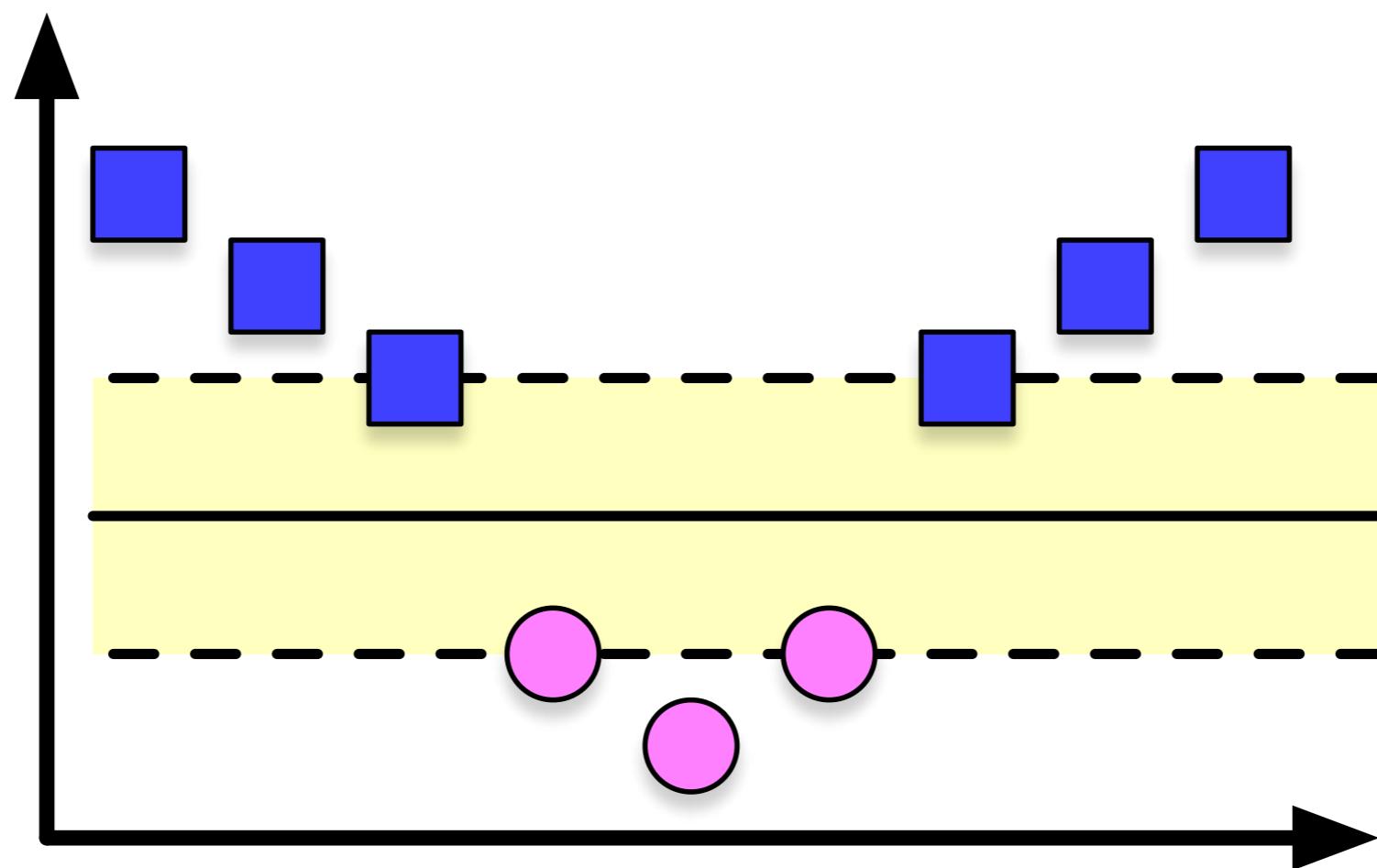
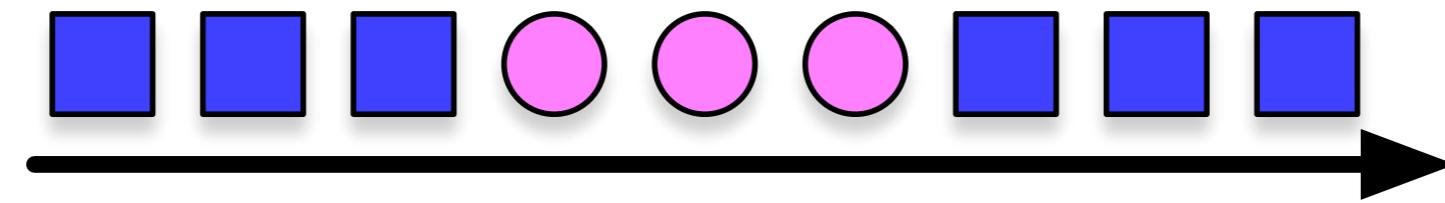
- ❖ Relación entre distintas técnicas de AI



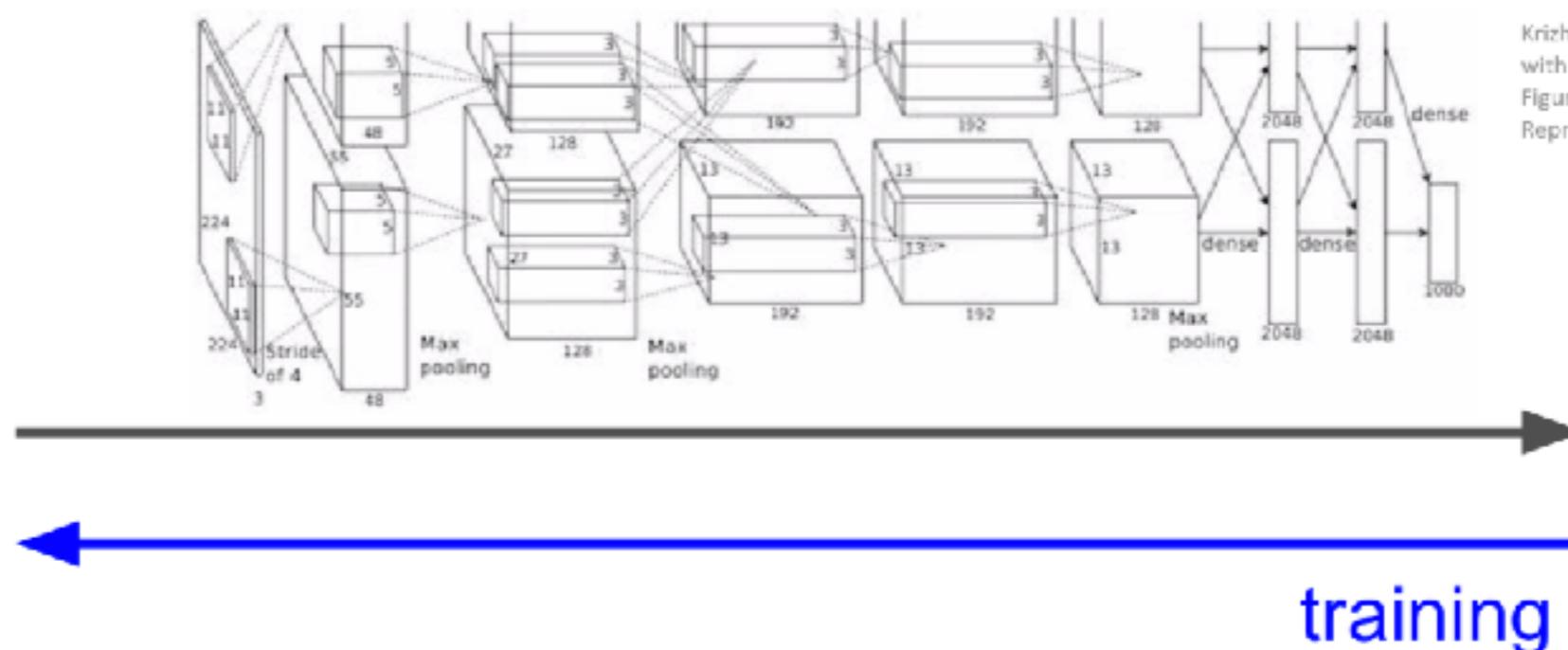
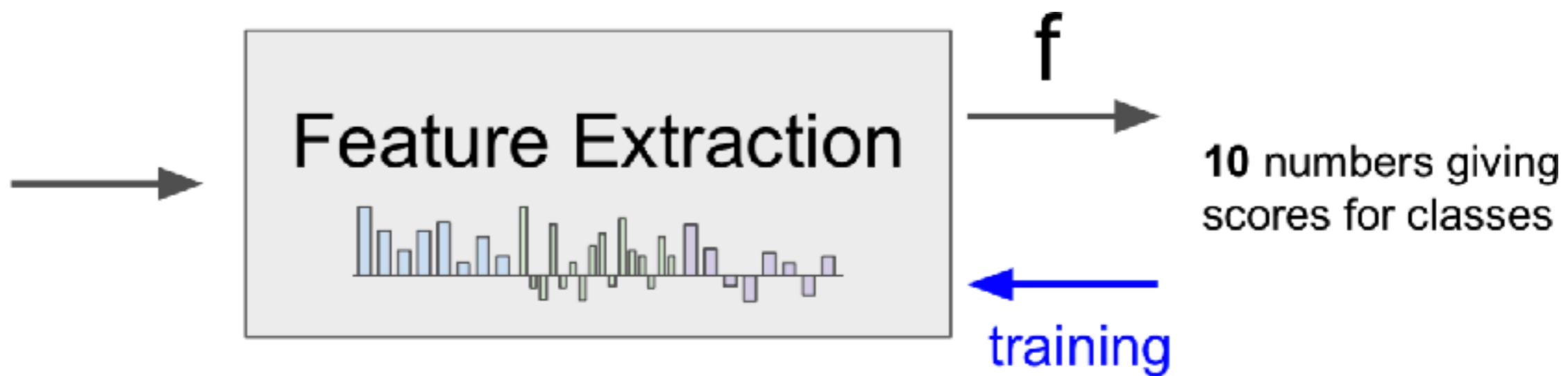
Extracción de características



Extracción de características



Extracción de características



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012.
Figure copyright Krizhevsky, Sutskever, and Hinton, 2012.
Reproduced with permission.