ACÁMICA

¡Bienvenidos/as a Data Science!

Agenda

¿Cómo anduvieron?

Repaso

Hands-On

Break

Explicación: Ampliando el Perceptrón

Hands-On

Cierre

¿Dónde estamos?

¿Cómo anduvieron?

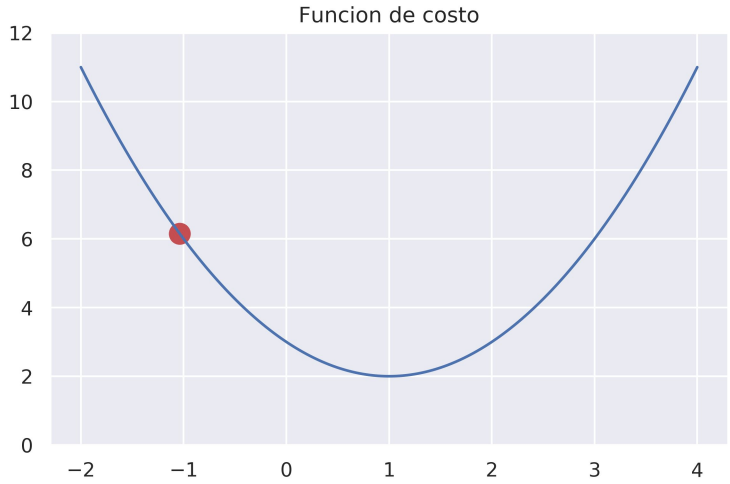
Repaso

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

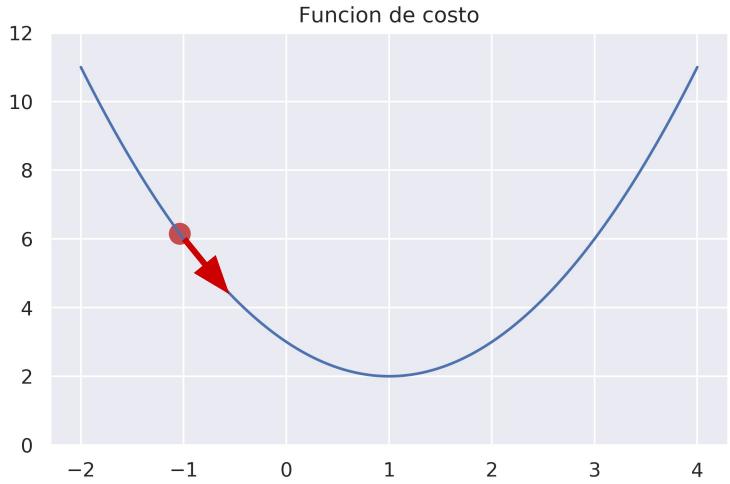


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

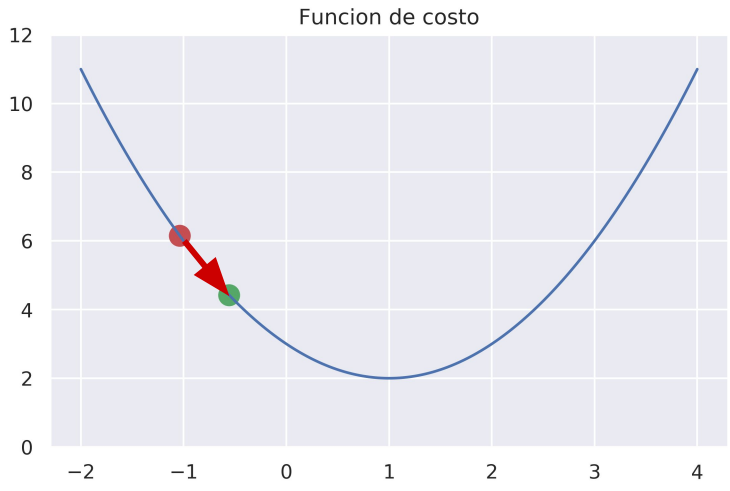


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. Técnicamente, derivamos o calculamos el gradiente.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

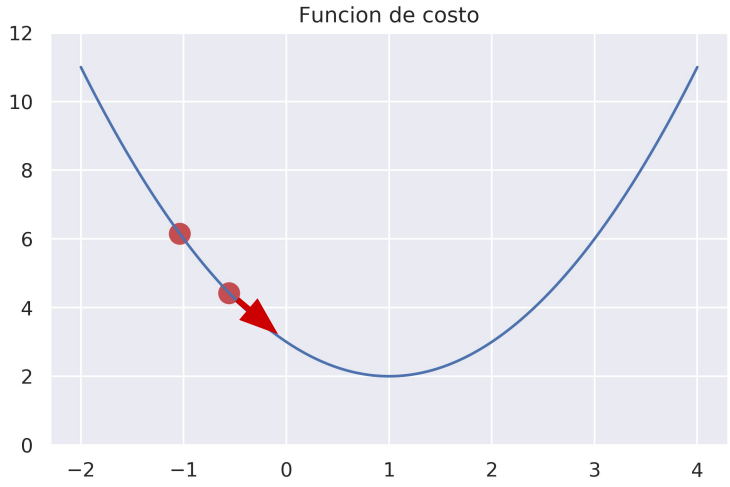


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. Técnicamente, derivamos o calculamos el gradiente.
   b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

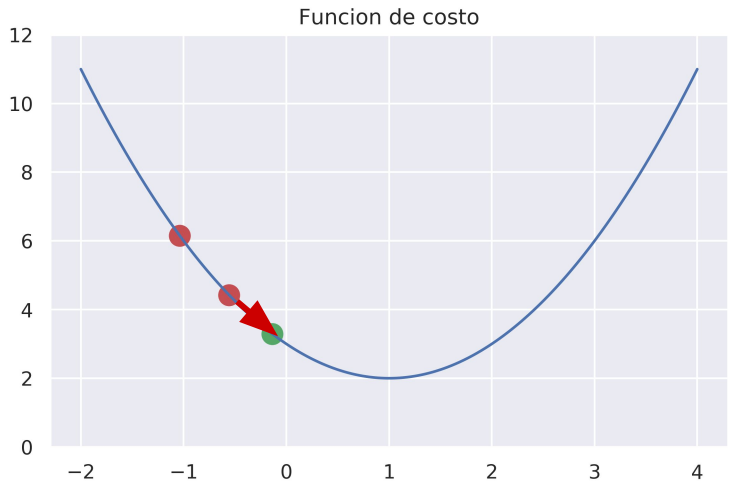


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. **Técnicamente**, derivamos o calculamos el gradiente.
   b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

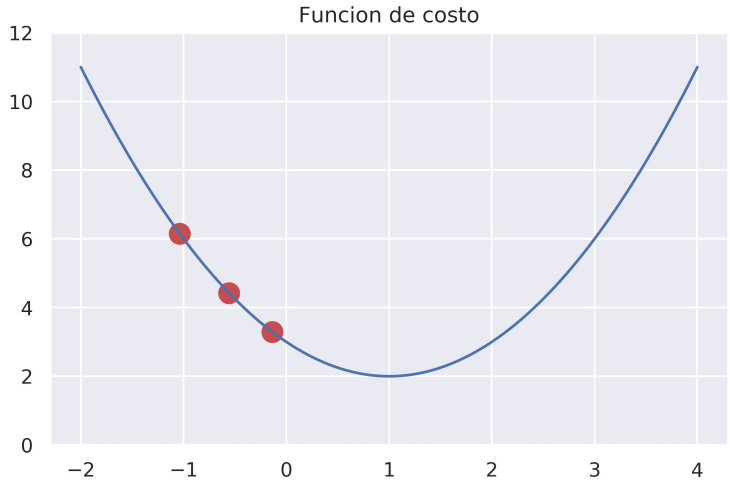


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. **Técnicamente,** derivamos o calculamos el gradiente.
   b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

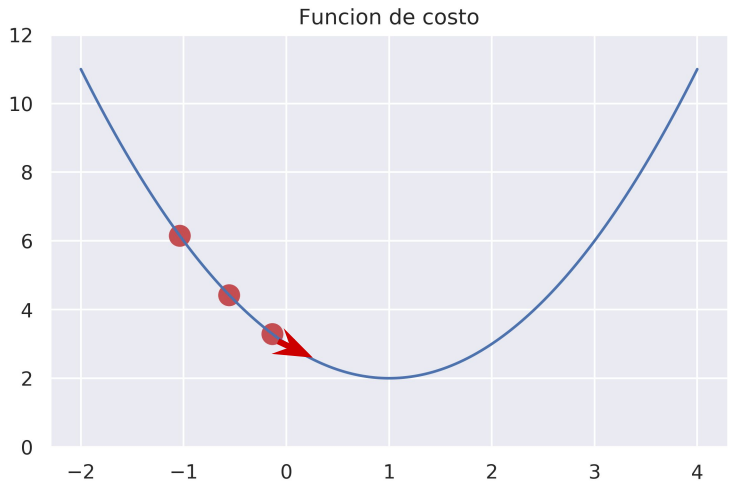


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. **Técnicamente,** derivamos o calculamos el gradiente.
   b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

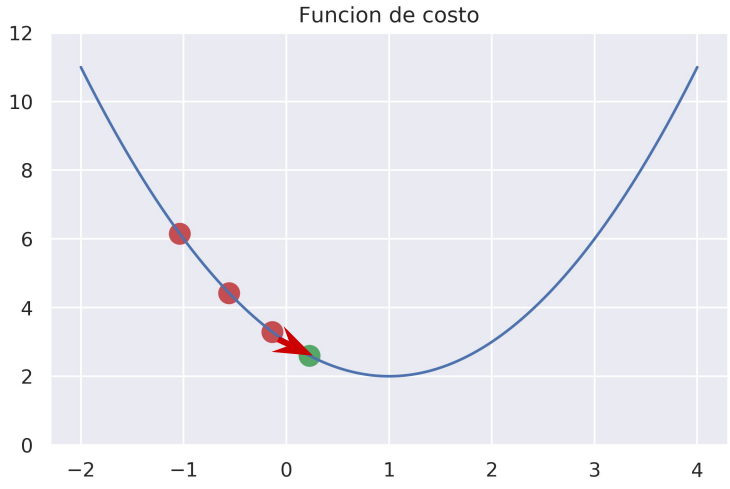


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. **Técnicamente,** derivamos o calculamos el gradiente.
   b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:

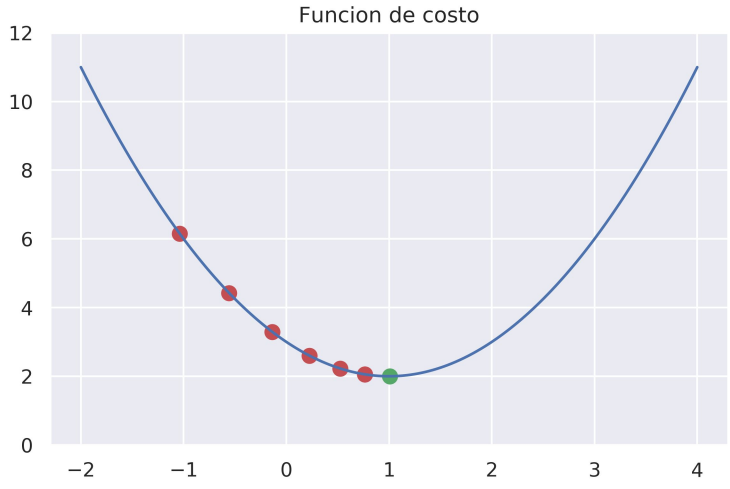


Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. **Técnicamente,** derivamos o calculamos el gradiente.
   b. Actualizamos los valores de los parámetros.

Descenso por gradiente

Queremos explorar el mínimo, pero no hicimos una exploración exhaustiva de la función de costo:



Pasos

1. Calculamos el costo para ciertos valores al azar de los parámetros.
2. Repetimos hasta converger
   a. Nos fijamos la dirección de decrecimiento en ese punto. **Técnicamente,** derivamos o calculamos el gradiente.
   b. Actualizamos los valores de los parámetros.

Descenso por gradiente • Resumen

1. Necesitamos una función de costo/pérdida. La función de costo depende del problema (clasificación, regresión, etc).
2. La función de costo es una función de los parámetros de la red (bueno, también de los datos que tengo, pero ignoremos eso por ahora).
3. Los mejores parámetros de la red son aquellos que minimicen la función de costo.
4. Cómo explorar todo ese espacio de parámetros exhaustivamente (simil *grid search*) es imposible, necesitamos una técnica que lo haga eficientemente. Esa técnica es Descenso por Gradiente.

Descenso por gradiente • Resumen

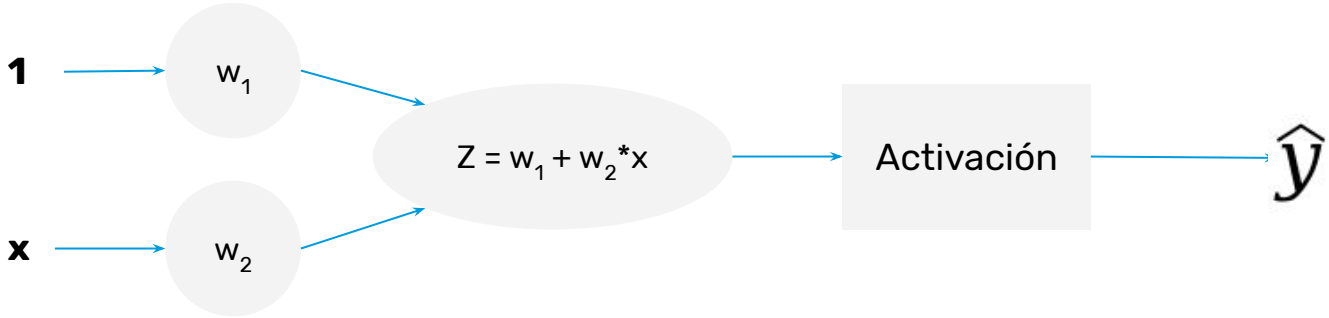
1. Necesitamos una función de costo/pérdida. La función de costo depende del problema (clasificación, regresión, etc).
2. La función de costo es una función de los parámetros de la red (bueno, también de los datos que tengo, pero ignoremos eso por ahora).
3. Los mejores parámetros de la red son aquellos que minimicen la función de costo.
4. Como explorar todo ese espacio de parámetros exhaustivamente (simil *grid search*) es imposible, necesitamos una técnica que lo haga eficientemente. Esa técnica es Descenso por Gradiente.

> Mucha de la jerga en redes neuronales refieren a técnicas para optimizar esta búsqueda

Perceptrón

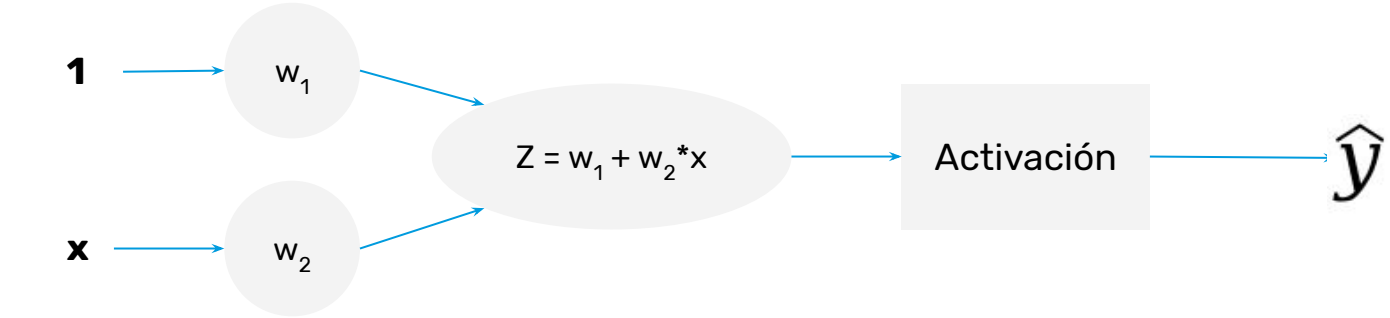
Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



Entradas: Features + Bias

Pesos: Lo que buscamos en el entrenamiento

Unión sumadora: combina los features con los pesos

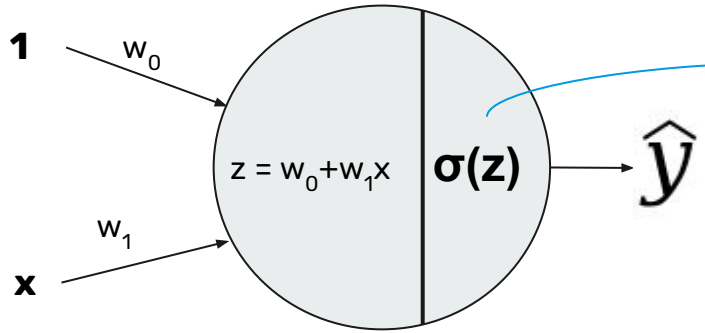
Activación: si no está, ¡es una función lineal!

Salida: entre 0 y 1, probabilidad del evento

Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1

Otra representación



Activación:

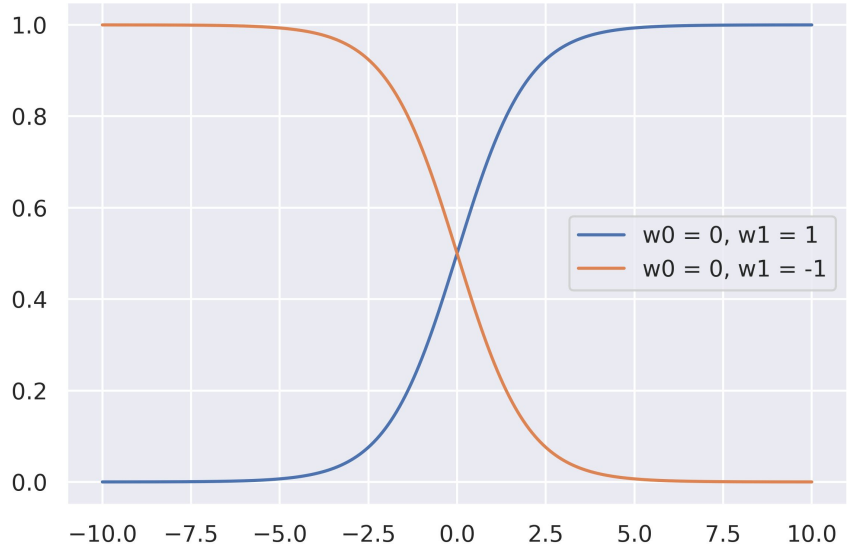
- Sin la activación, es una función lineal
- Necesitamos introducir algo que *sature* la entrada en 0 o en 1 dependiendo del resultado de la unión sumadora

Función Logística / Sigmoide

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

$$z = w_0 + w_1 x$$

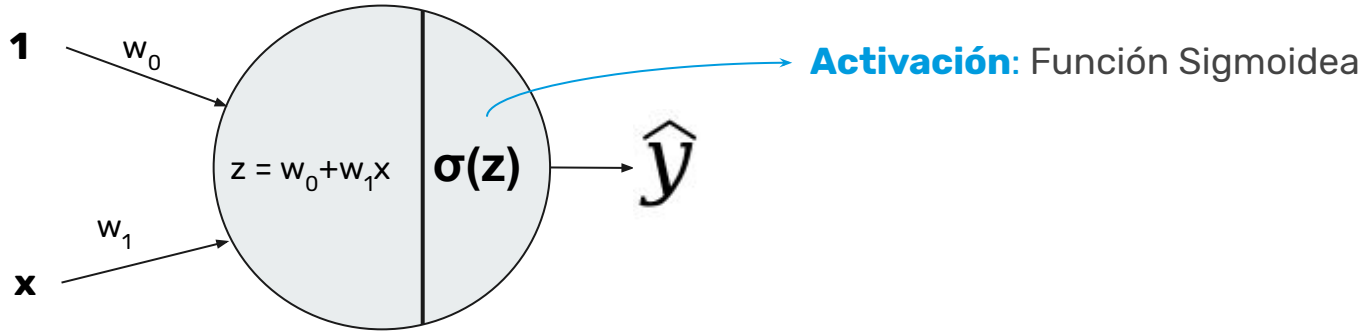
$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$



Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1

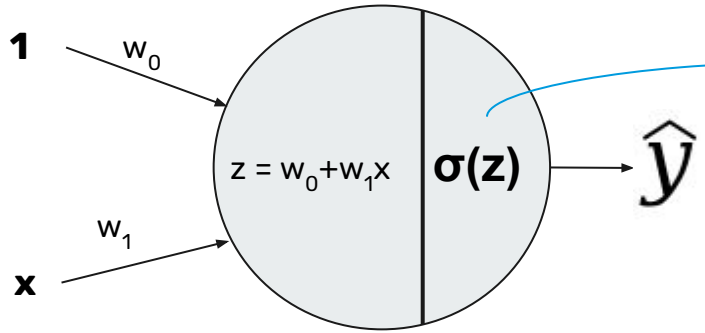
Otra representación



Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1

Otra representación



Activación: Función Sigmoidea

¿Qué falta?

¡Falta encontrar los pesos w_0 y w_1 apropiados para nuestros datos!

Para eso necesitamos una **función de costo**

Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$ **Pérdida para una instancia**

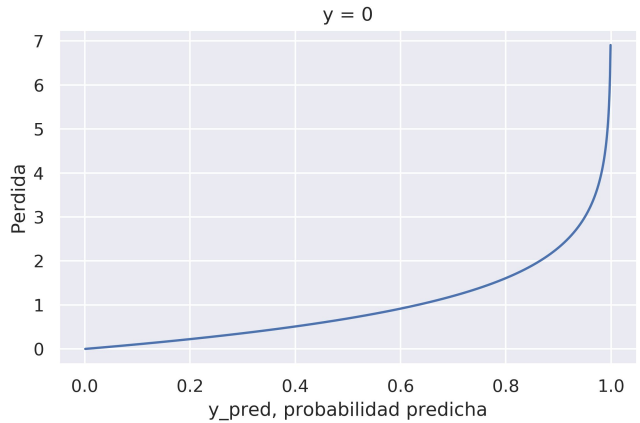
Entropía cruzada (cross-entropy)

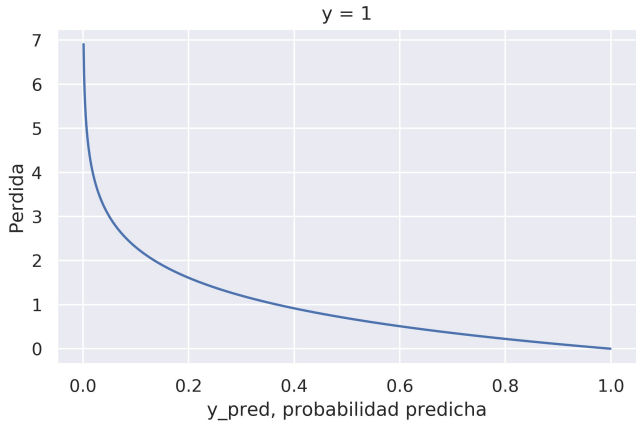
Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$

Pérdida para una instancia





Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$ **Pérdida para una instancia**

Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$

Pérdida para una instancia

$$J(\overline{W}) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

Costo para todas las instancias

Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$

Pérdida para una instancia

$$J(\overline{W}) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

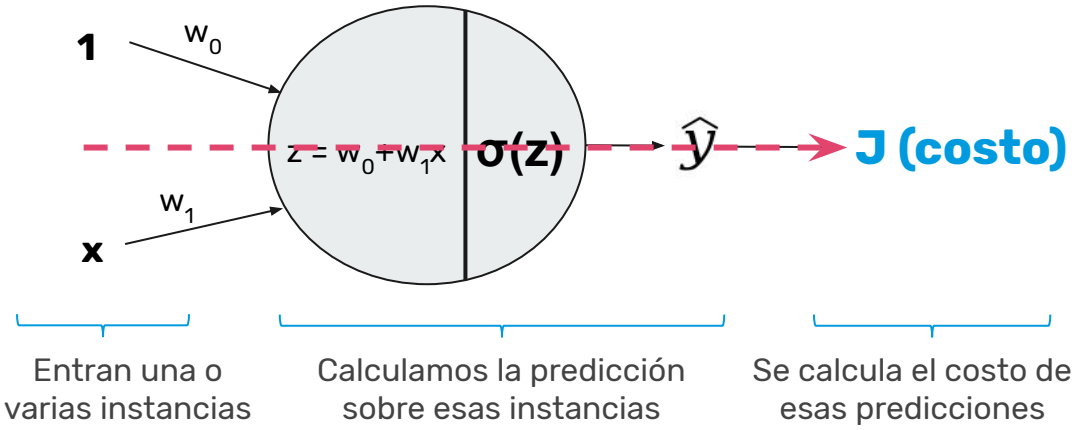
Costo para todas las instancias

$$J(w_0,\ w_1) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

Costo para todas las instancias, caso 1D

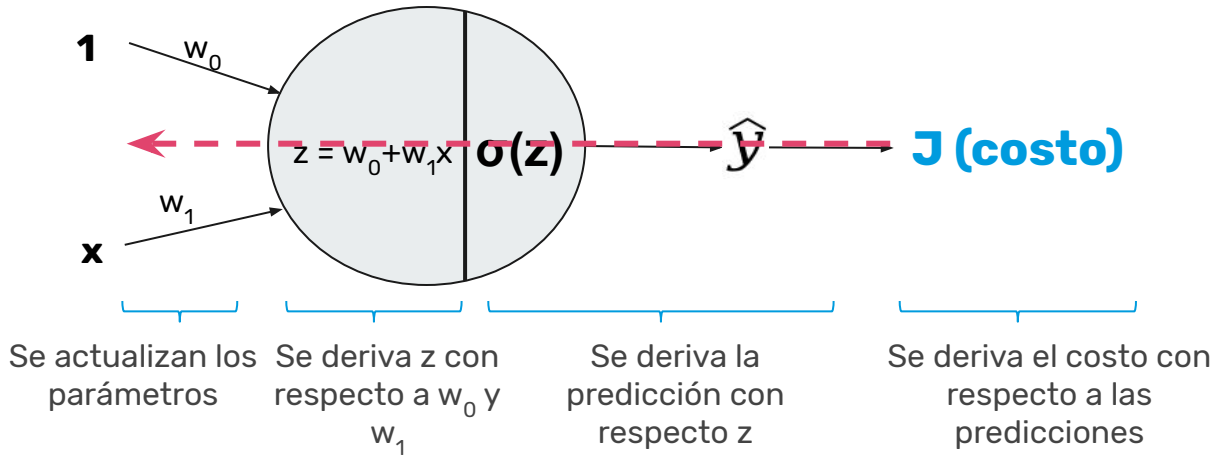
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation.**



1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1.

Calcular las derivadas y actualizar los parámetros "hacia atrás" se conoce como **Backpropagation.**



Hands-on training

Hands-on training

DS_Encuentro_31_Perceptron_Multicapa.ipynb

Parte 1

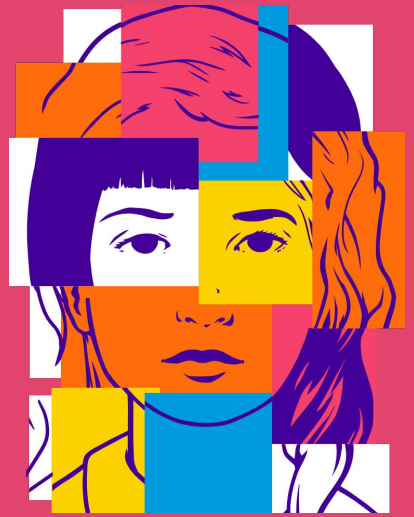
¡BREAK!

Ph. Credit: Drew Coffmann

Ampliando el Perceptrón

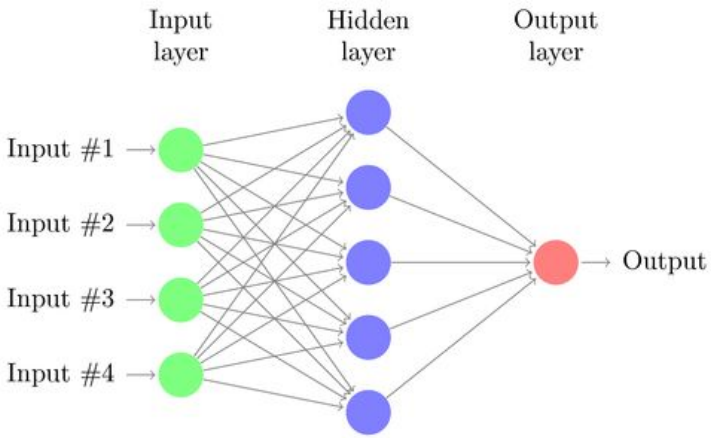
Problema con el Perceptrón:
solo encuentra fronteras lineales.

Ampliando el perceptrón



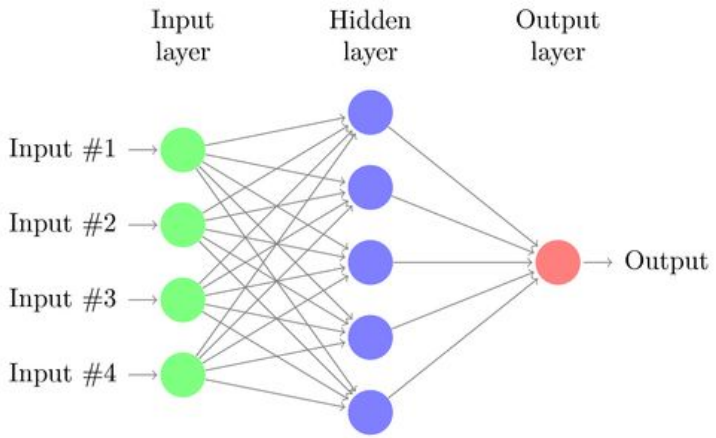
Ampliando el Perceptrón

Solución: Perceptrón Multicapa



- Cada neurona tiene sus propios pesos/parámetros. En aplicaciones comunes suelen ser desde miles a millones de parámetros para toda la red.
- **Deep Learning es** encontrar esos pesos de manera eficiente, bajo la condición de realizar correctamente una tarea objetivo.

Ampliando el Perceptrón



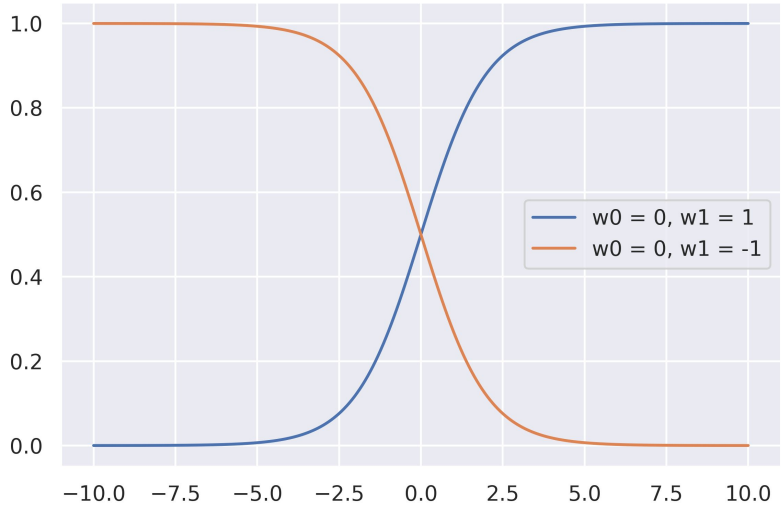
Sigue valiendo:

- Forward Propagation
- Backpropagation
- Descenso por gradiente
- Función de costo

Perceptrón MultiCapa

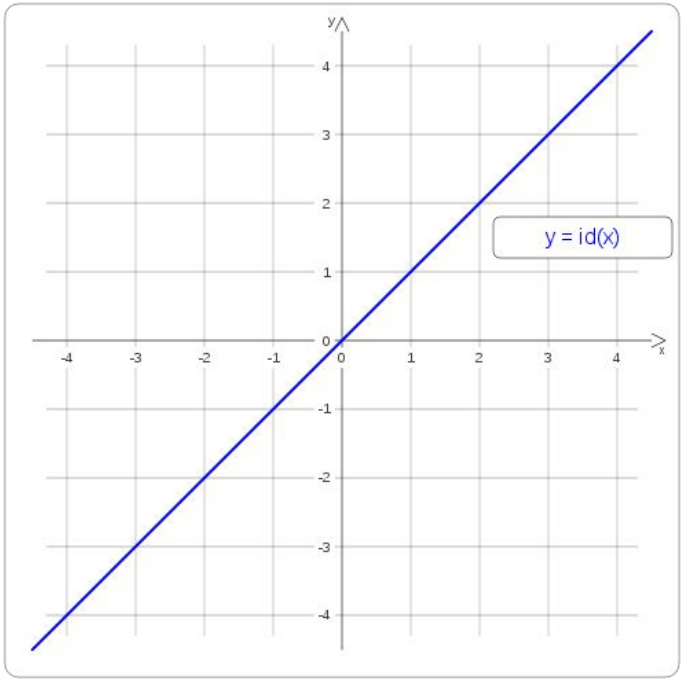
Playground

Perceptrón Multicapa • Funciones de activación



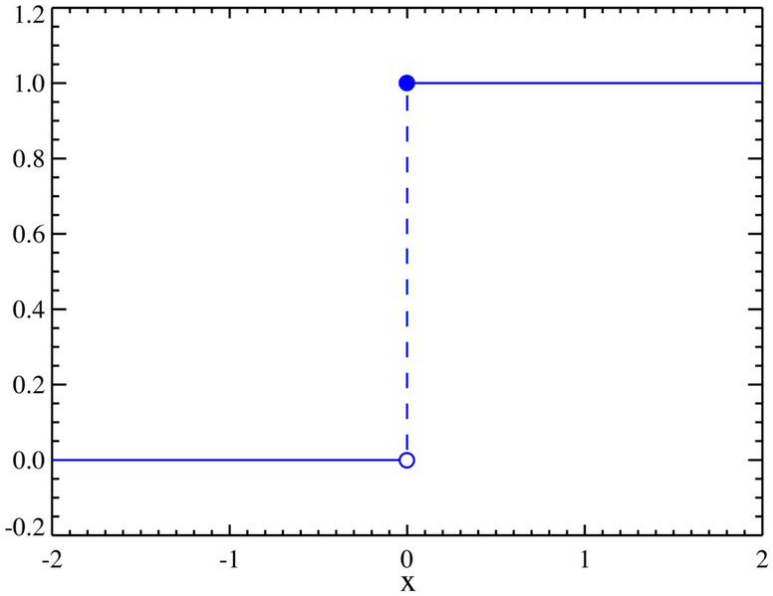
1. **Sigmoide/logística**
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



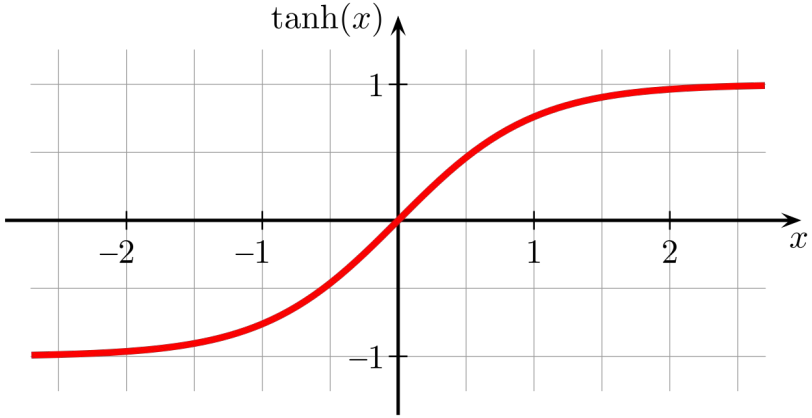
1. Sigmoide/logística
2. **Identidad: f(x) = x**
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



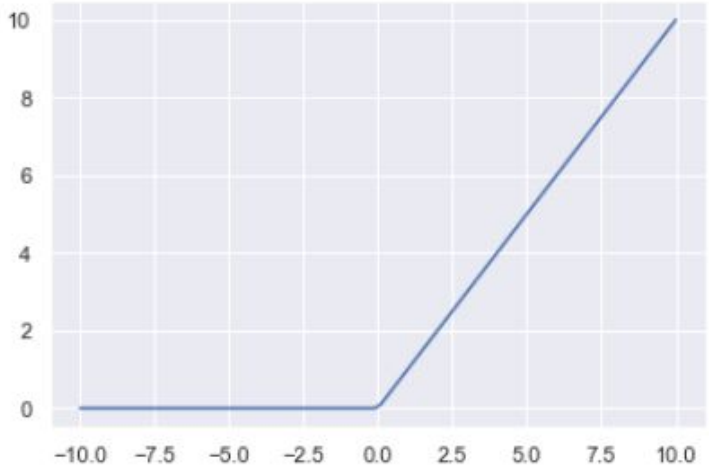
1. Sigmoide/logística
2. Identidad: f(x) = x
3. **Escalón: f(x)=0 si x<0, 1 si x≥0**
4. Tangente Hiperbólica: f(x)=tanh(x)
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



1. Sigmoide/logística
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. **Tangente Hiperbólica: f(x)=tanh(x)**
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



1. Sigmoide/logística
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. **ReLU: f(x)=0 si x<0, x si x≥0**

Perceptrón Multicapa • Funciones de activación

Clasificación:
lo más común es encontrar ReLU en las capas interiores y Sigmoide en la salida

1. **Sigmoide/logística**
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. **ReLU: f(x)=0 si x<0, x si x≥0**

Perceptrón Multicapa

Multiclase. La cantidad de neuronas en la capa de salida tiene que ser igual a la cantidad de clases buscadas.

Perceptrón Multicapa

	Funciones de activación	**Costos (Keras)**
Multiclase	1. Sigmoide/logística 2. Softmax	Categorical_crossentropy

Perceptrón Multicapa

	Funciones de activación	Costos (Keras)
Multiclase	1. Sigmoide/logística **2. Softmax**	Categorical_crossentropy

Generalización de la sigmoide, útil cuando las clases son excluyentes.

Perceptrón Multicapa

	Funciones de activación	**Costos (Keras)**
Multiclase	1. Sigmoide/logística **2. Softmax**	Categorical_crossentropy
Regresión	Identidad	1. mean_squared_error 2. mean_absolute_error 3. Otras

Generalización de la sigmoide, útil cuando las clases son excluyentes.

Perceptrón Multicapa

	Funciones de activación	**Costos (Keras)**
Multiclase	1. Sigmoide/logística 2. Softmax	Categorical_crossentropy
Regresión	Identidad	1. mean_squared_error 2. mean_absolute_error 3. Otras

Regularización

Objetivo: castigar parámetros/pesos muy grandes.

están asociados a overfitting.

Regularización

Objetivo: castigar parámetros/pesos muy grandes.

están asociados a overfitting.

¿Cómo? Tres técnicas muy comunes

- Regularización L2 y L1: agregan un término a la función de costo que castiga los pesos grandes.
- **Dropout:** funciona como una capa que "apaga" neuronas de la capa anterior al azar.

Regularización

¿Cómo?

Dropout: funciona como una capa que “apaga” neuronas de la capa anterior al azar.

Muy utilizado. Al apagar neuronas, obliga a que ninguna se aprenda “de memoria” una muestra, sino que tengan que aprender entre todas.
Otra interpretación: Ensamble

Recursos

Recursos

Optimización de Hiperparámetros

Activaciones
https://keras.io/activations/

Pérdidas
https://keras.io/losses/

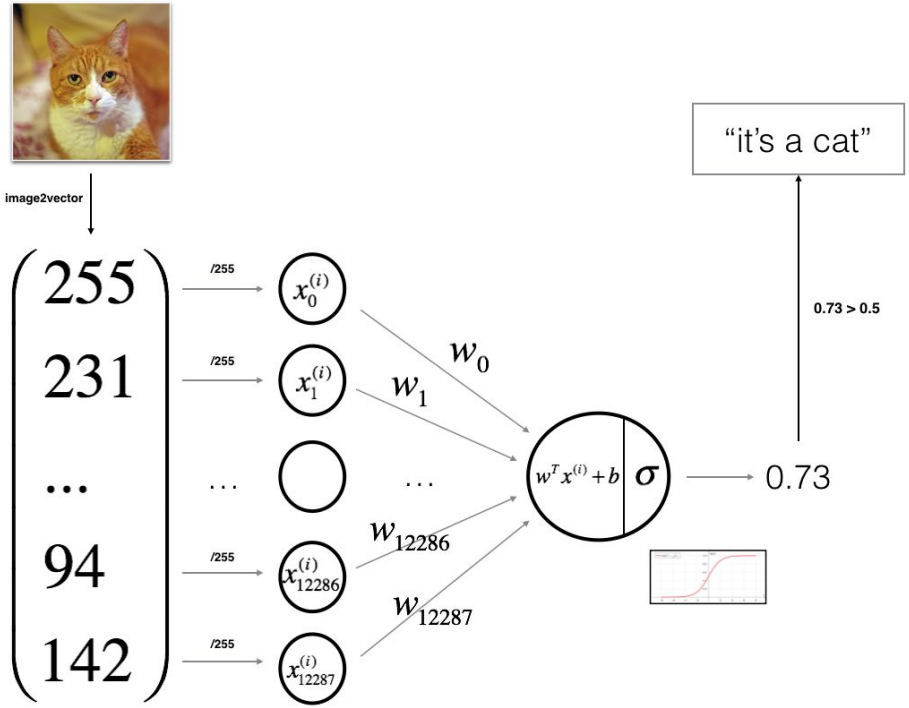
Optimizadores
https://keras.io/optimizers/

Regularizadores
https://keras.io/regularizers/

Hands-on training

Hands-on training

Comentario para el Hands-On



Hands-on training

DS_Encuentro_31_Perceptron_Multicapa.ipynb

Parte 2

Recursos

Recursos

- ¿Pero qué "es" una Red neuronal?| aprendizaje profundo, Parte 1
- Descenso de gradiente, es como las redes neuronales aprenden| Aprendizaje profundo, capítulo 2.
- ¿Qué es la retropropagación y qué hace en realidad? | Aprendizaje profundo, Capítulo 3.

Para la próxima

1. Terminar los notebooks de hoy y atrasados.
2. Ver los videos mencionados en “Recursos”.

ACÁMICA