Fundamentos de las Redes Neuronales

Bloques de construcción
Multilayer Perceptron (MLP)

Clase 7

Dra. Wendy Aguilar

Modelos Generativos Profundos

UN ENFOQUE DESDE LA CREATIVIDAD COMPUTACIONAL

Bloques de construcción de Redes Neuronales

Bloques matemáticos fundamentales

1. **Neuronas artificiales**
   - Cada neurona aplica una transformación lineal seguida de una no linealidad:

$$h = f(Wx + b)$$

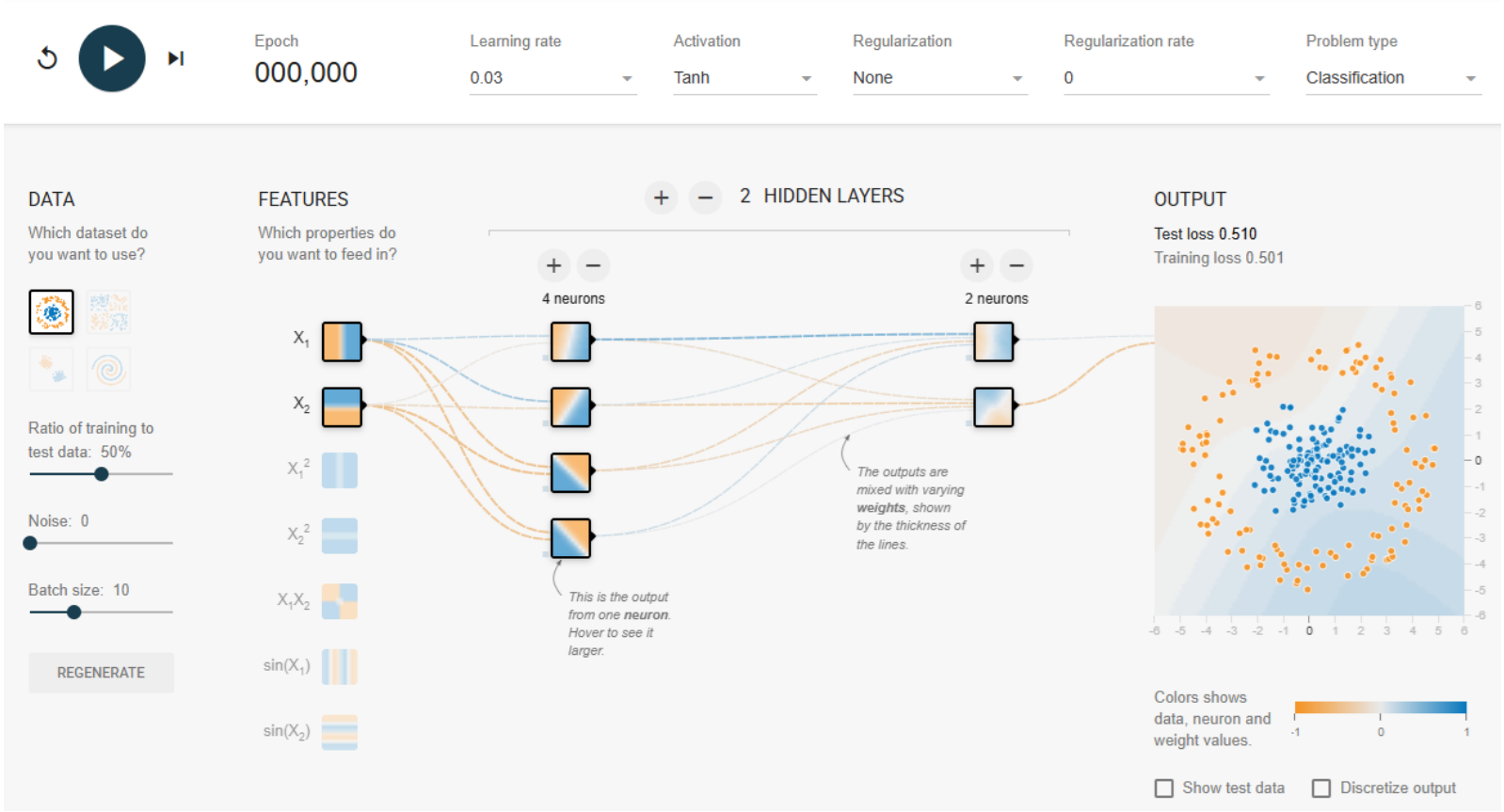
2. **Funciones de activación**
   - Introducen la **no linealidad** necesaria para que la red modele funciones complejas.
   - Ejemplos: sigmoide, tanh, ReLU, Leaky ReLU, GELU.
3. **Funciones de pérdida (loss)**
   - Definen el objetivo a optimizar (ej. MSE en regresión, cross-entropy en clasificación).
4. **Algoritmos de optimización**
   - Usan los gradientes para actualizar los pesos.
   - Ejemplos: SGD, Momentum, RMSProp, Adam.
5. **Backpropagation**
   - El mecanismo que calcula los gradientes de la pérdida respecto a cada peso.

Bloques de construcción de Redes Neuronales

Bloques arquitectónicos básicos

1. **Capas densas (fully connected)**
   - El "bloque universal" de MLPs (Perceptrones Multicapa).
2. **Capas convolucionales (CNNs)**
   - Extraen patrones espaciales, útiles en imágenes.
3. **Capas recurrentes (RNN, LSTM, GRU)**
   - Modelan dependencias temporales o secuenciales.
4. **Normalización**
   - Ej. BatchNorm, LayerNorm → estabilizan los gradientes.
5. **Dropout y regularización**
   - Bloques que ayudan a reducir el sobreajuste.
6. **Atención (Attention blocks)**
   - Bloques que permiten ponderar dinámicamente la importancia de cada entrada (clave en Transformers).

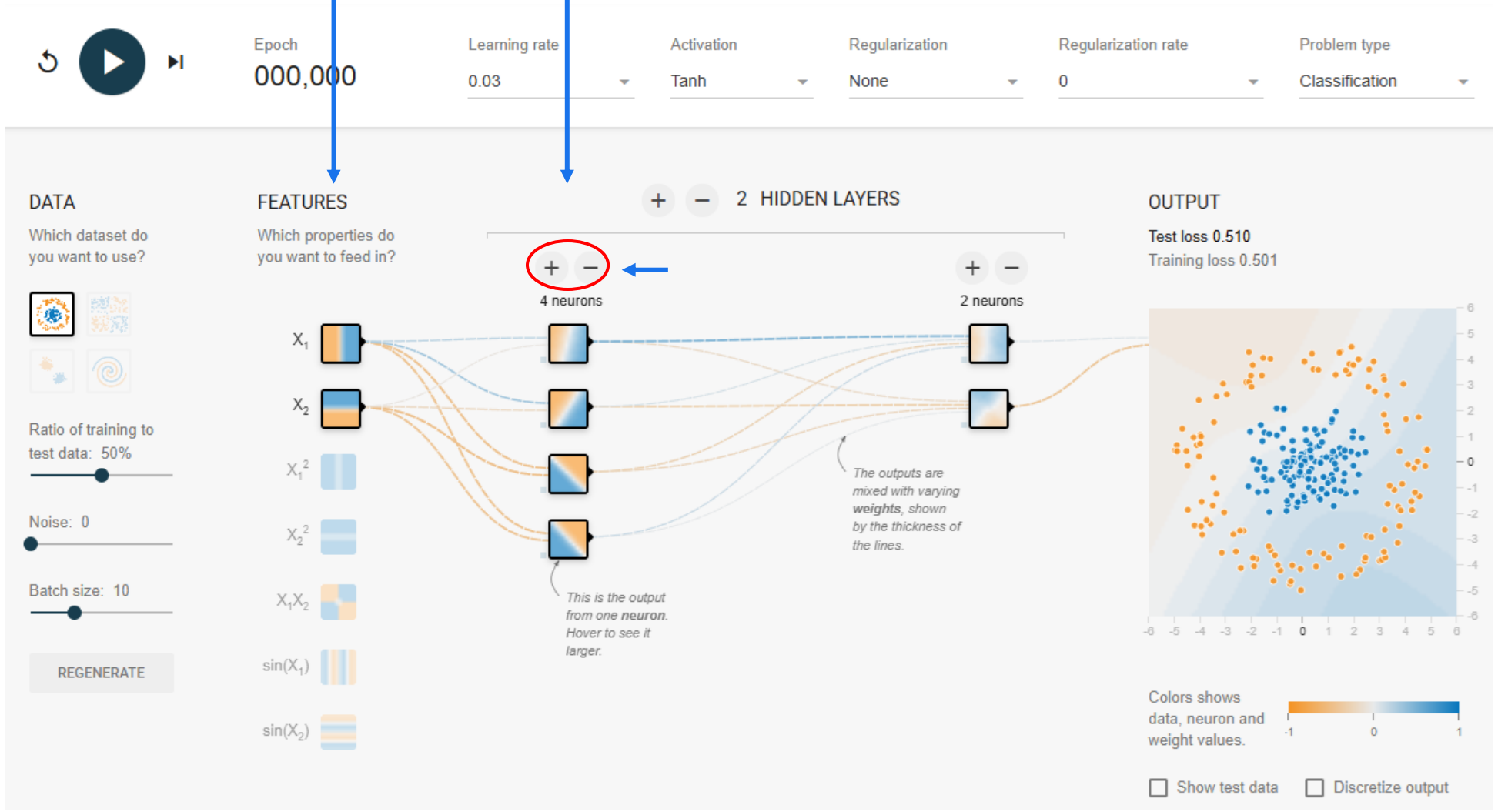
https://playground.tensorflow.org/

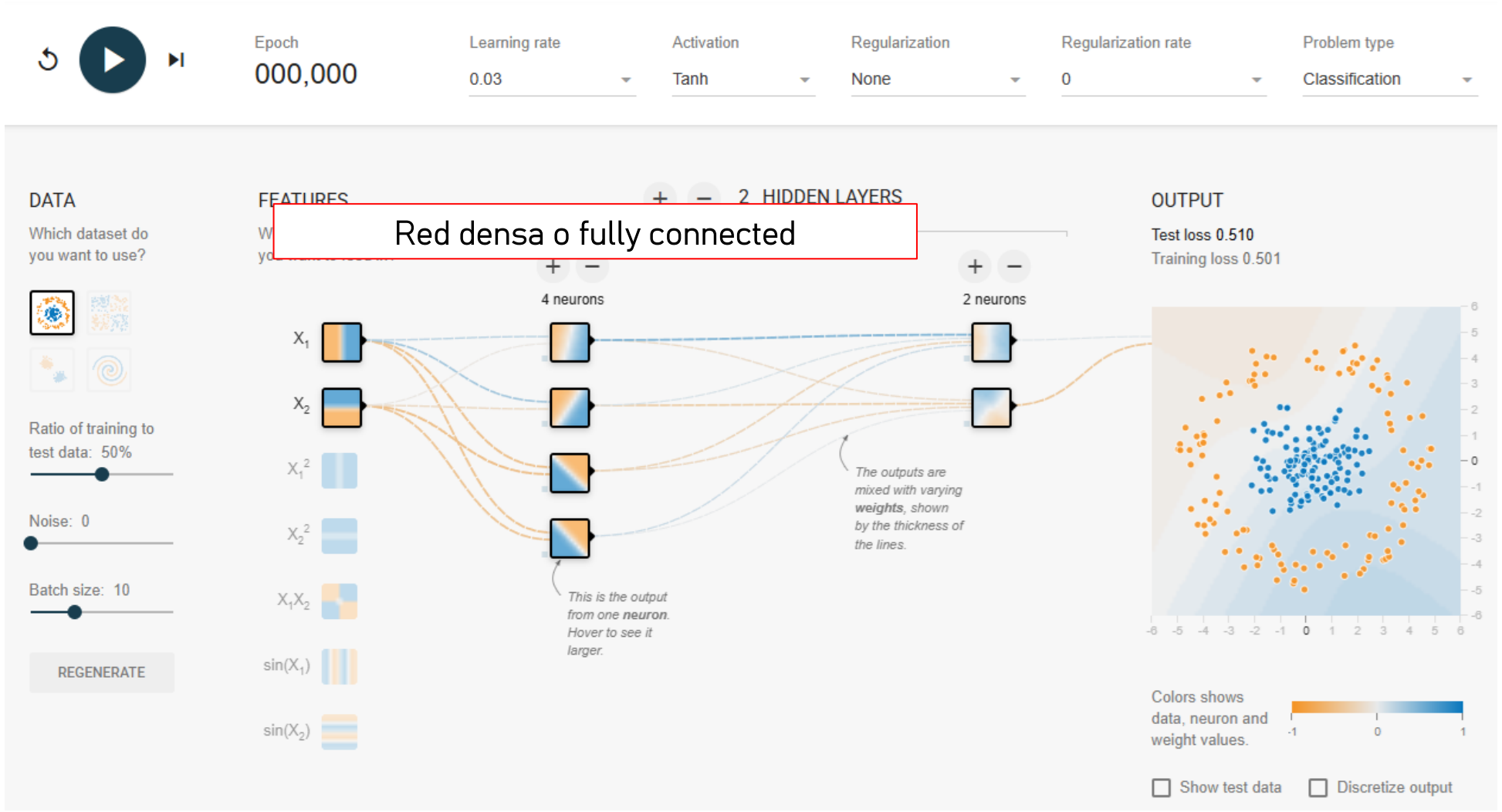


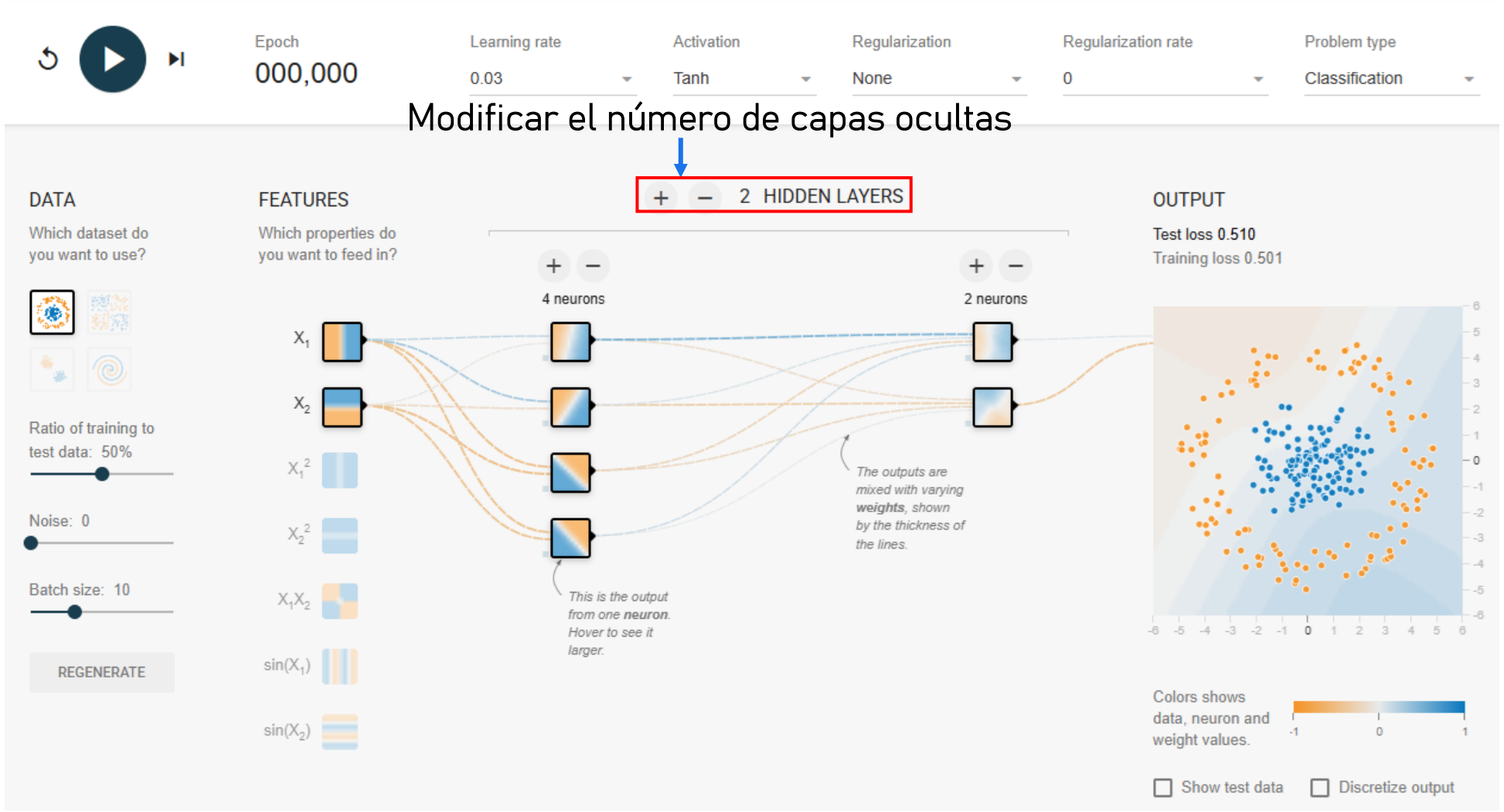
- Podemos ver visualmente cómo podemos construir y entrenar una red neuronal
- Cambiar el número de capas, neuronas, funciones de activación, etc.

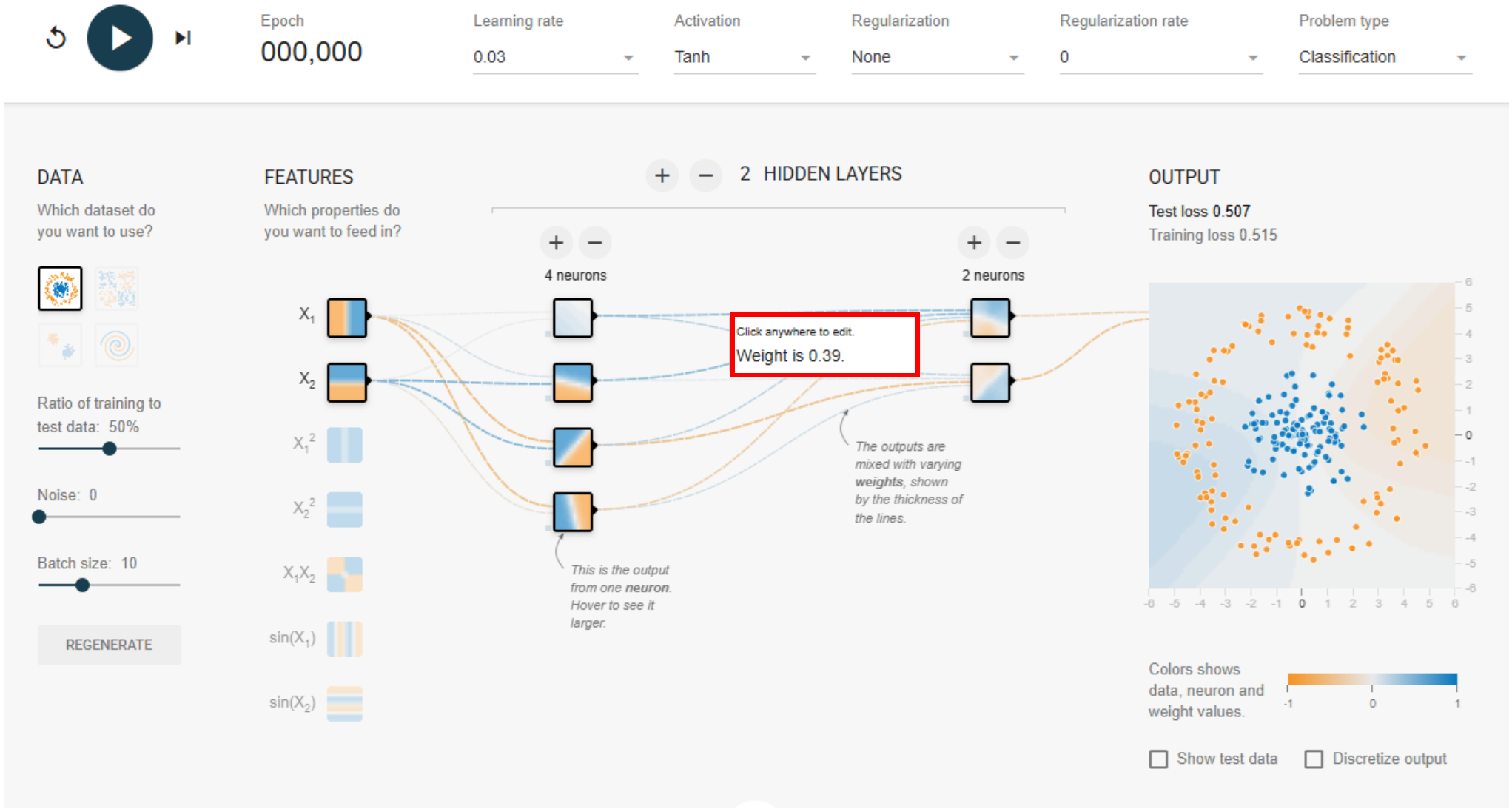
Entradas

Una capa oculta

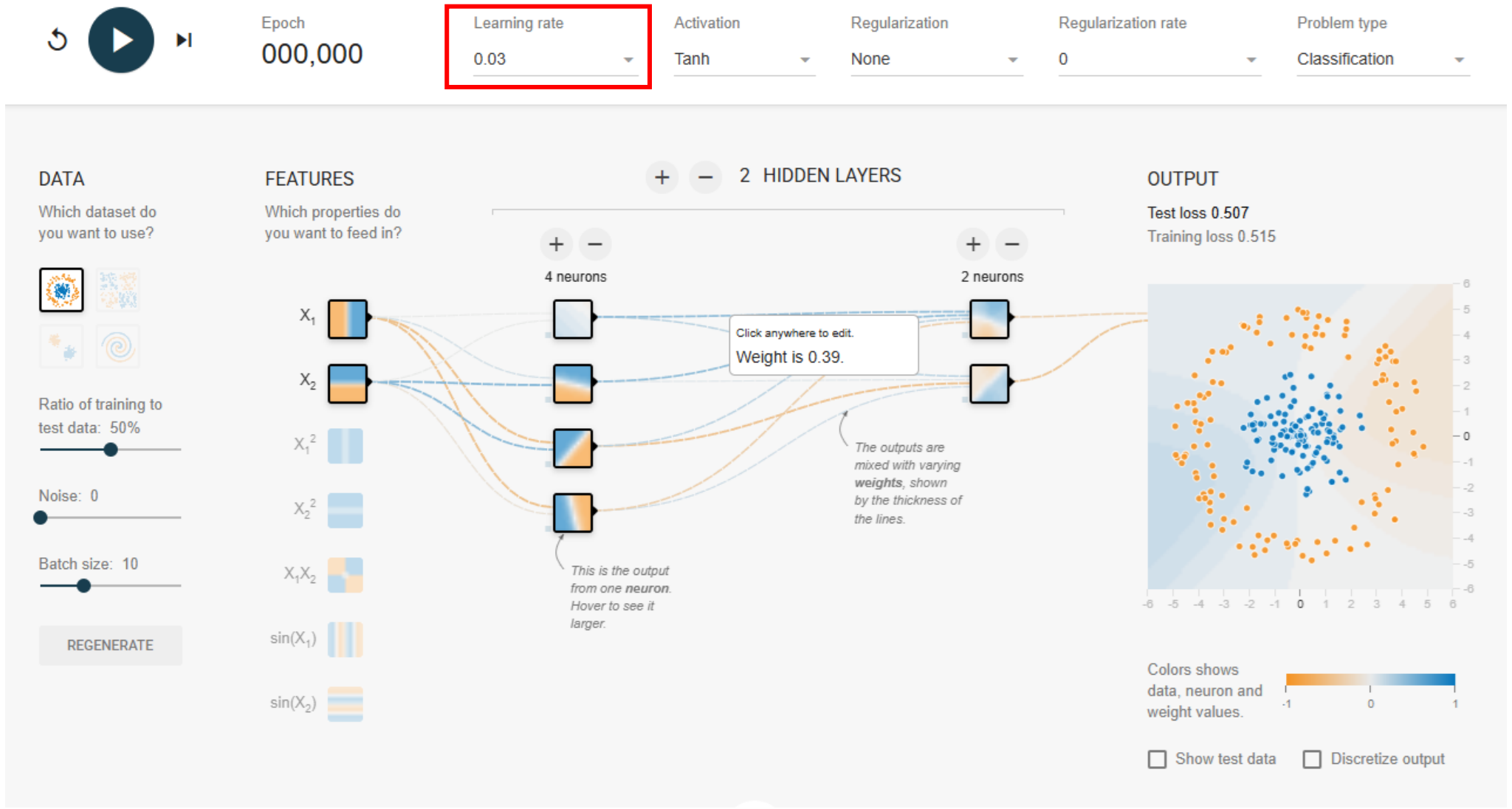



Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
2 HIDDEN LAYERS
Red densa o fully connected
4 neurons
2 neurons
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.510
Training loss 0.501
Colors shows data, neuron and weight values.
Show test data
Discretize output


Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
Modificar el número de capas ocultas
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
4 neurons
2 neurons
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
The outputs are mixed with varying weights, shown by the thickness of the lines.
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.510
Training loss 0.501
Colors shows data, neuron and weight values.
Show test data
Discretize output



- Se inicializan con valores aleatorios.


Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
2 HIDDEN LAYERS
4 neurons
2 neurons
Click anywhere to edit.
Weight is 0.39.
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.507
Training loss 0.515
Colors shows data, neuron and weight values.
Show test data
Discretize output

Epoch
000,000

Learning rate
0.03

Activation
Tanh
ReLU
Tanh
Sigmoid
Linear

Regularization
None

Regularization rate
0

Problem type
Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$

+ N LAYERS

4 neurons

2 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

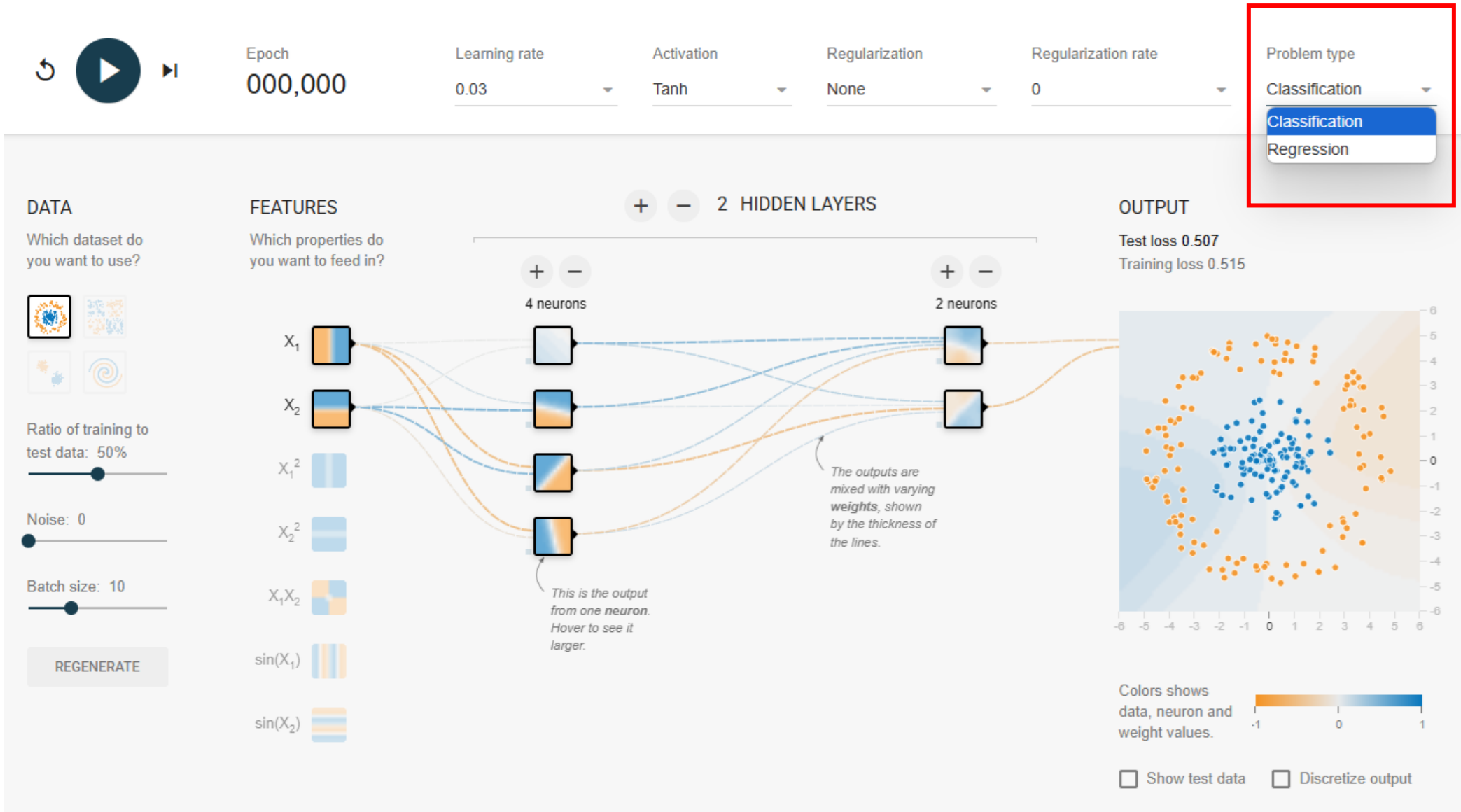
Test loss 0.507
Training loss 0.515

Colors shows data, neuron and weight values.

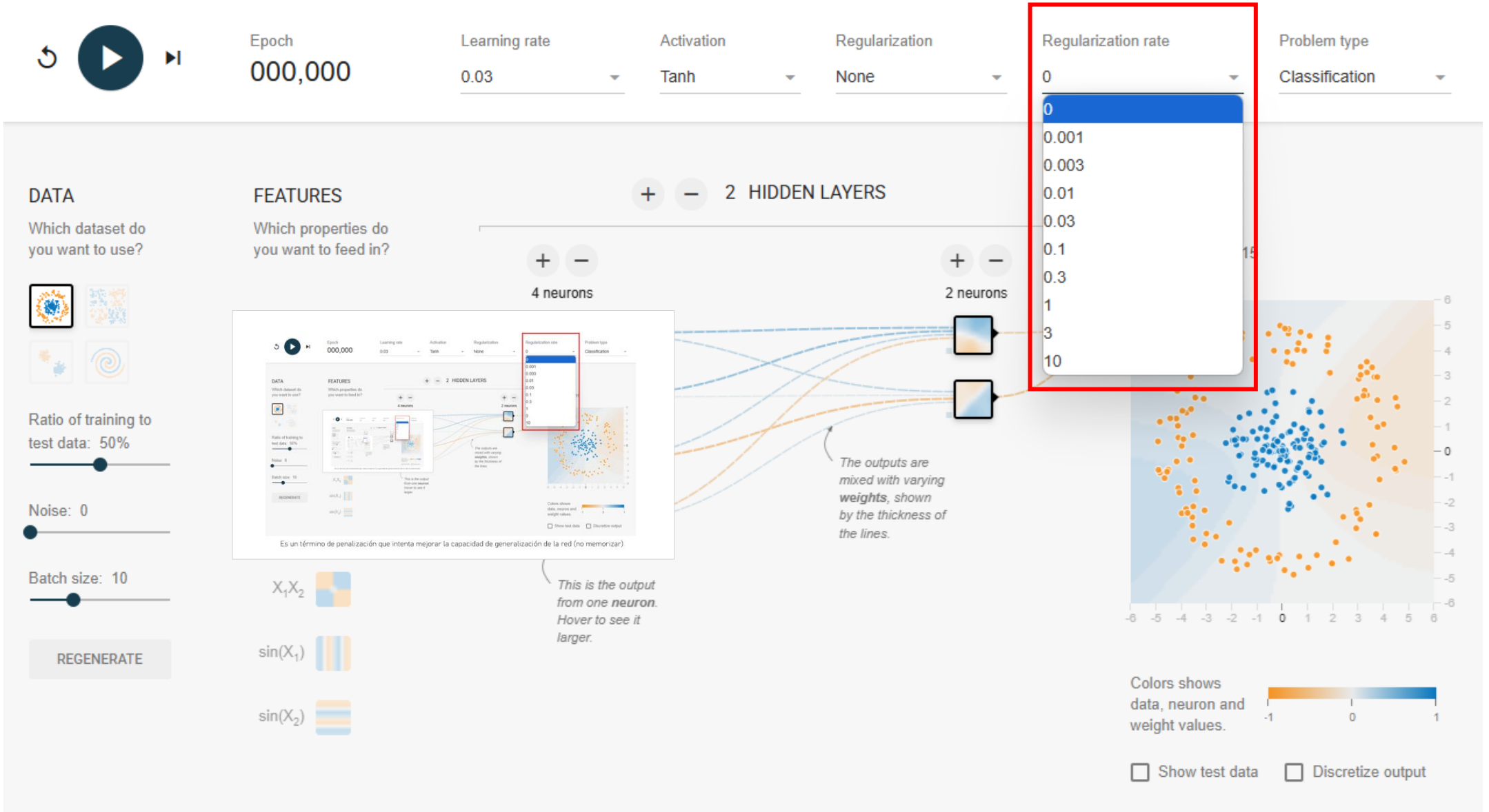
-1 0 1

Show test data

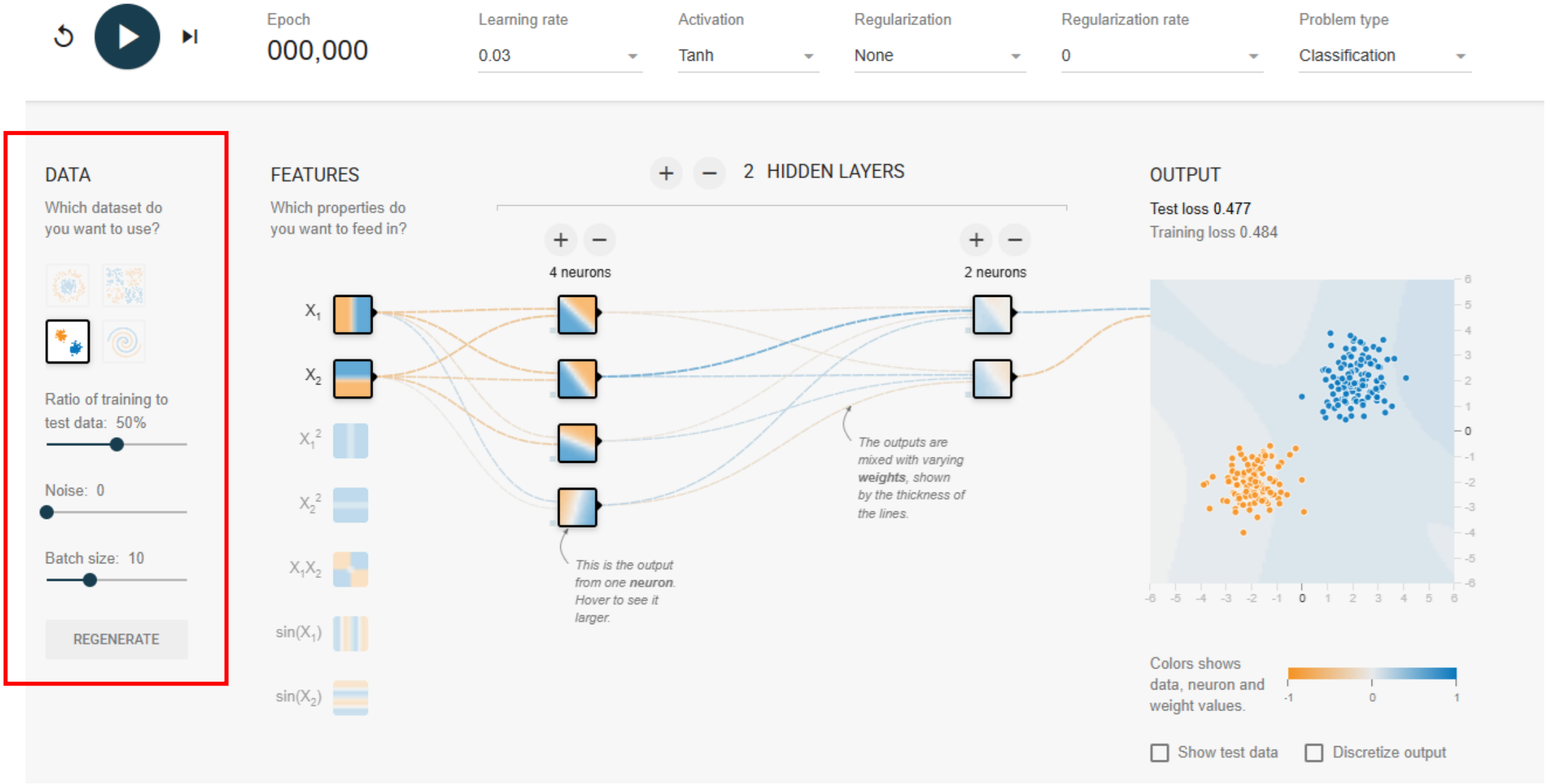
Discretize output



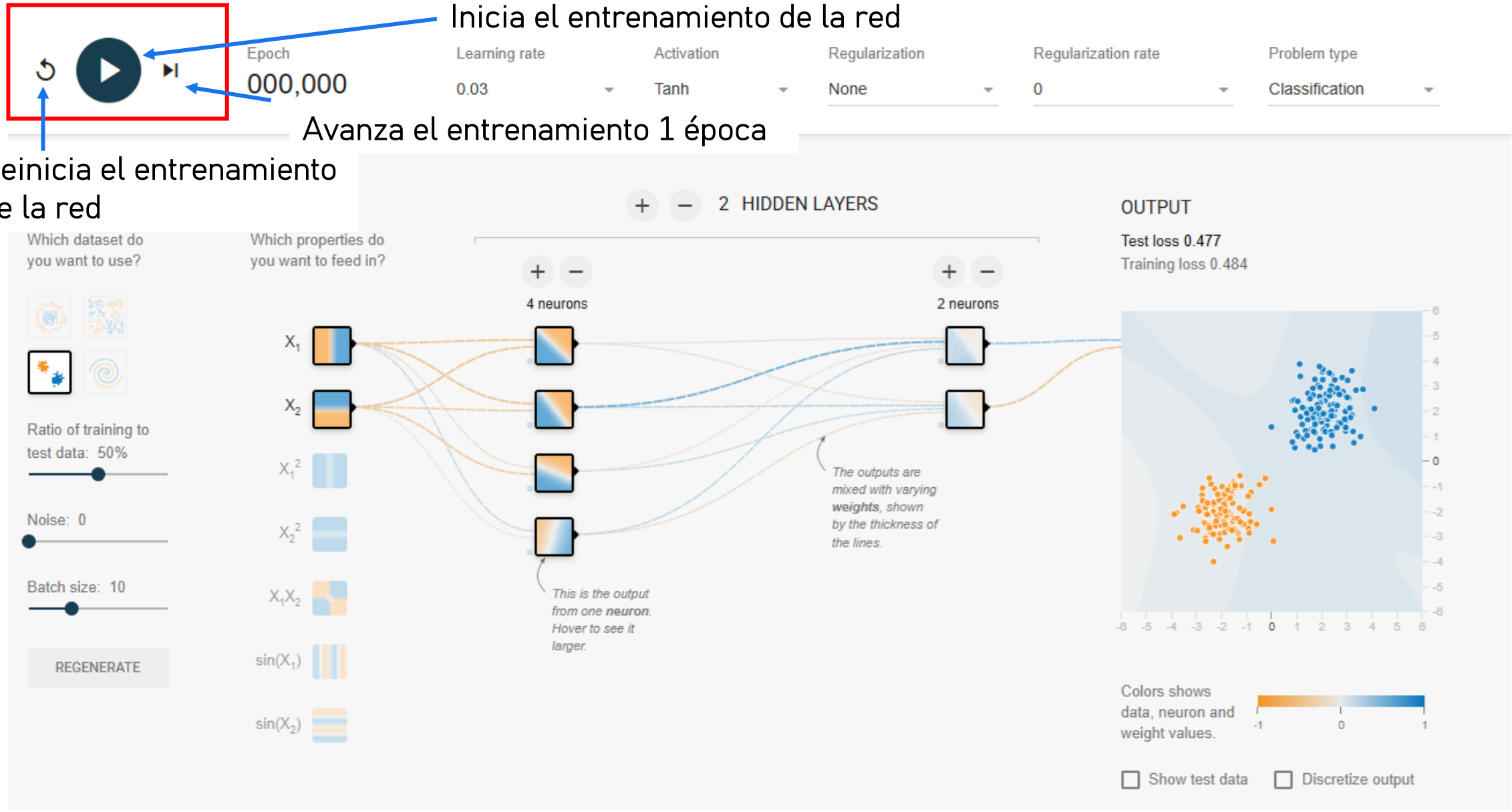
- Dejémoslo en clasificación



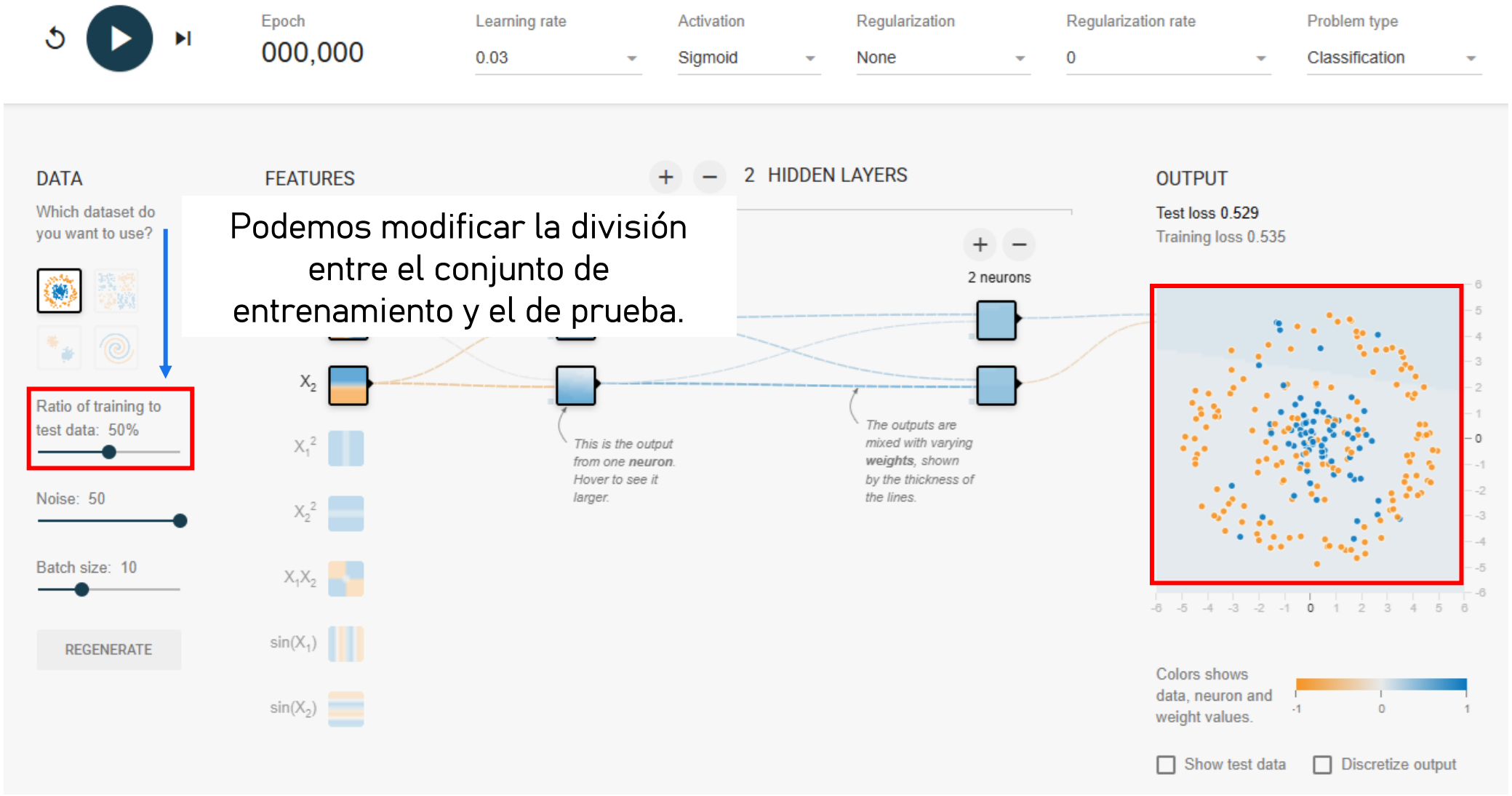
Es un término de penalización que intenta mejorar la capacidad de generalización de la red (no memorizar)



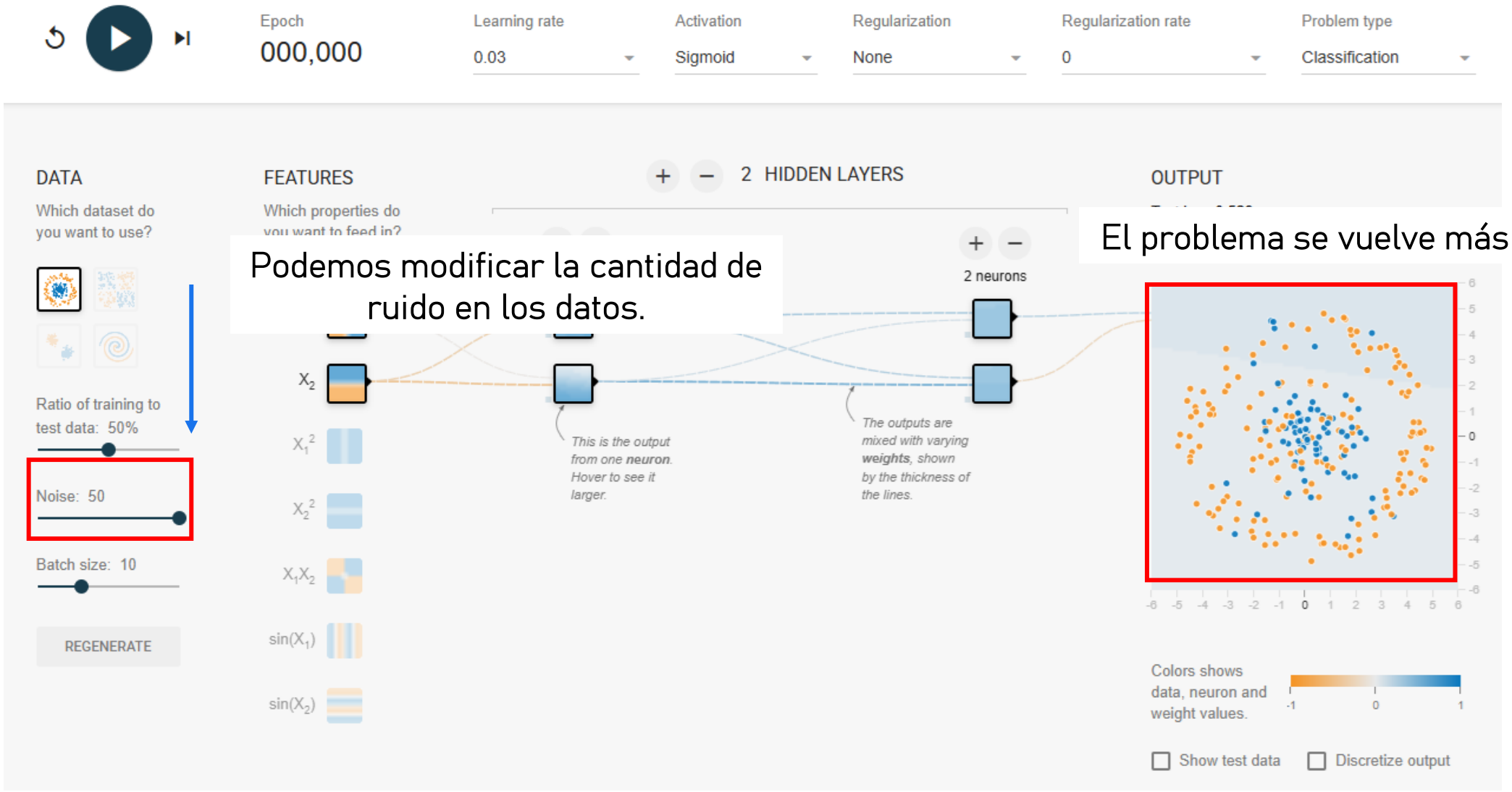
Seleccionamos este dataset que es el más sencillo.



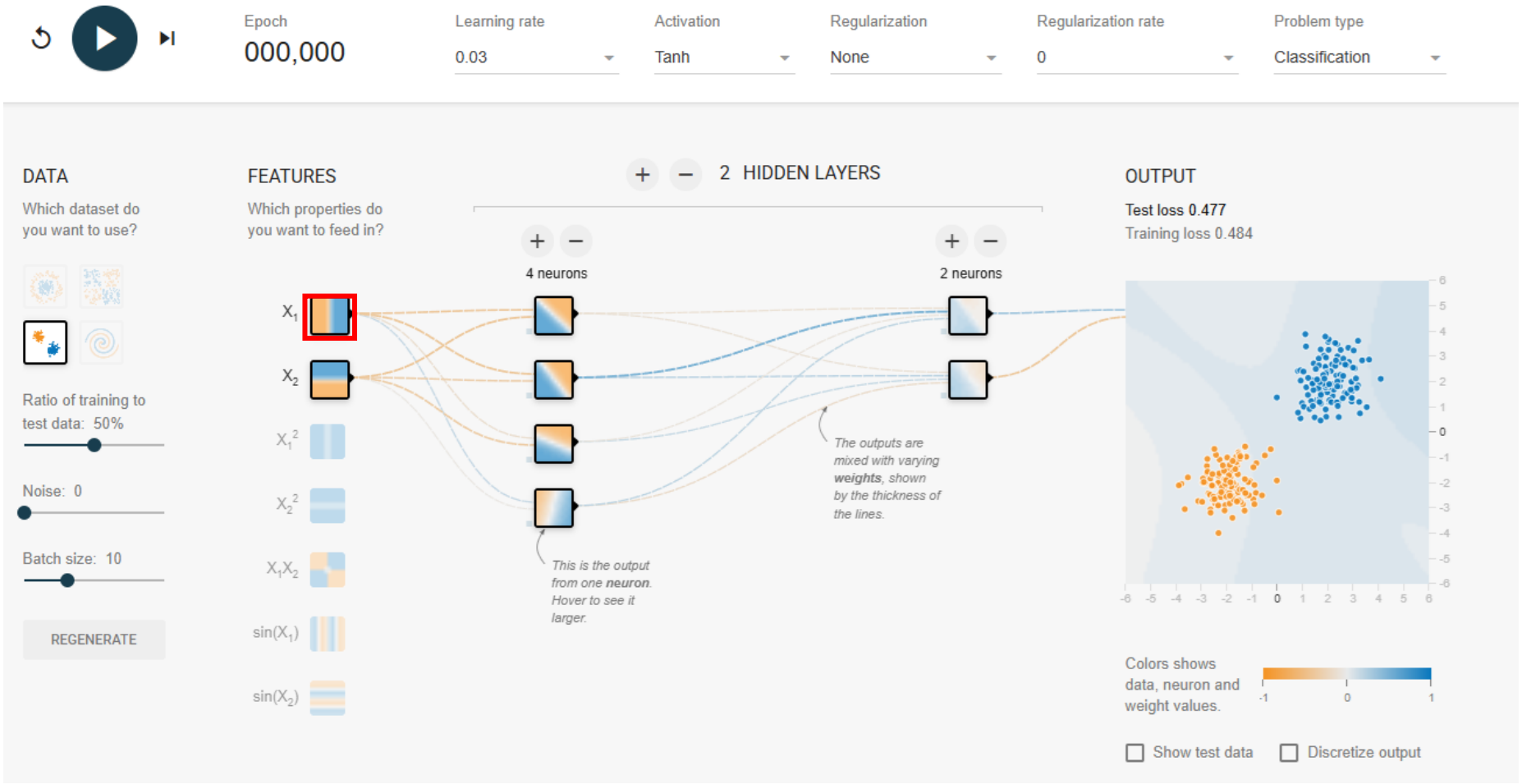
- Alimenta la red con los datos de entrenamiento.
- Actualiza los pesos hasta que minimice el error.



Podemos modificar la división entre el conjunto de entrenamiento y el de prueba.

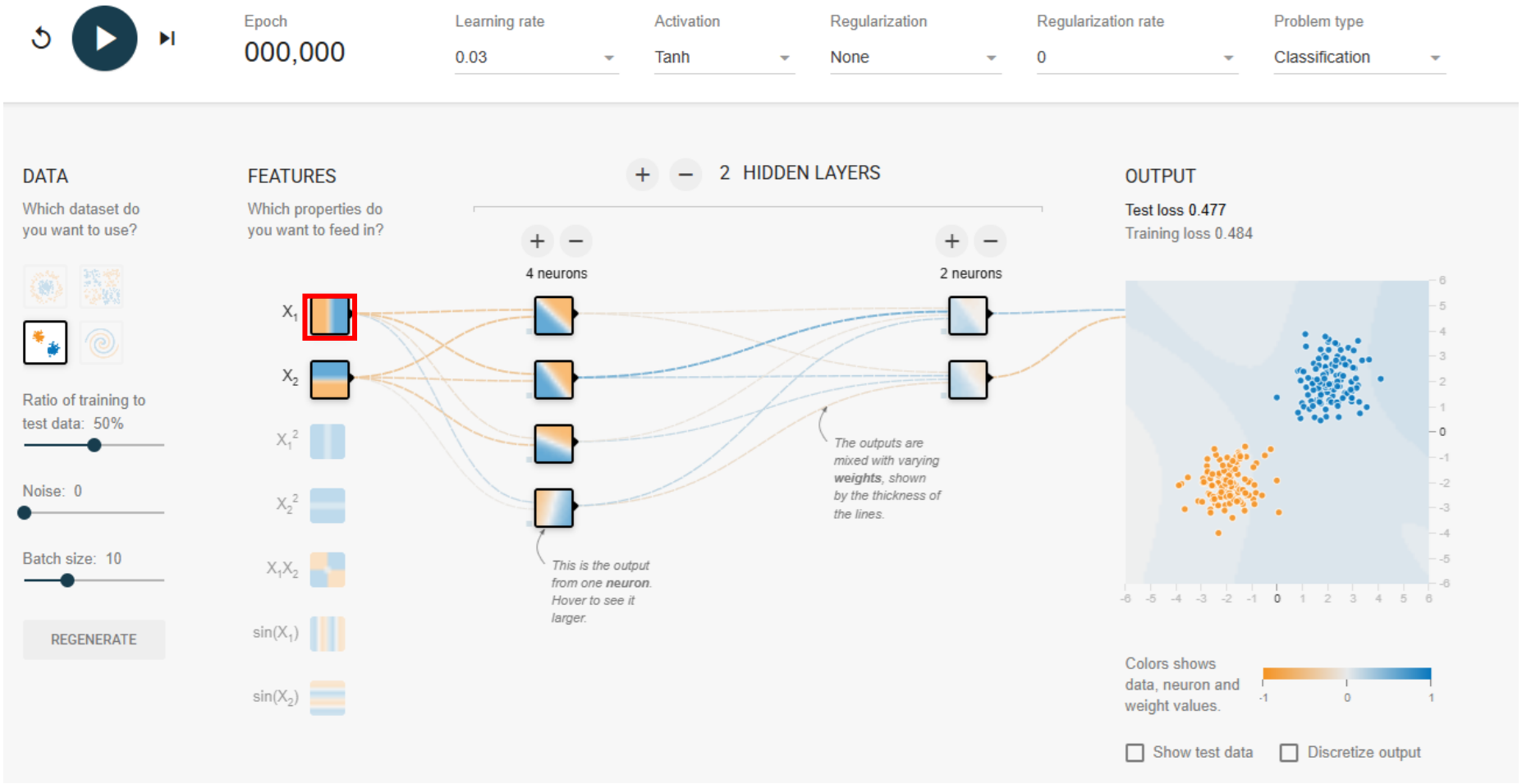

Epoch
000,000
Learning rate
0.03
Activation
Sigmoid
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 50
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
2 neurons
OUTPUT
Podemos modificar la cantidad de ruido en los datos.
El problema se vuelve más difícil.
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
Colors shows data, neuron and weight values.
Show test data
Discretize output

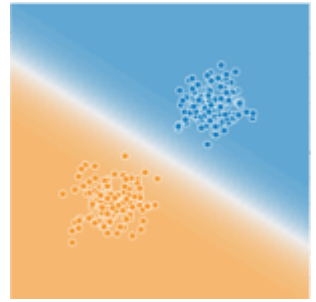

Epoch
000,000
Learning rate
0.03
Activation
Sigmoid
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
2 neurons
2 neurons
OUTPUT
Test loss 0.529
Training loss 0.535
Ratio of training to test data: 50%
Noise: 50
Batch size: 10
REGENERATE
Podemos modificar el tamaño del batch (el número de ejemplos que se le muestran a la red para calcular el error)
The outputs are mixed with varying weights, shown by the thickness of the lines.
Colors shows data, neuron and weight values.
Show test data
Discretize output



¿Qué son estas gráficas?

- **Mapa de activación** que muestra cómo responde esa neurona a todas las posibles entradas en el espacio de características que se está visualizando.
- El Playground siempre grafica el **plano de entrada (X1 vs X2)**, aunque estés en capas ocultas.
- Los colores indican la salida de la neurona para cada punto del plano:
  - Azul/naranja → predicción hacia una clase u otra.
  - Intensidad → cuán fuerte es la activación de esa neurona.





¿Cómo las interpreto?

Neuronas de la primera capa

- Sus gráficas suelen mostrar líneas rectas (hiperplanos).
- Cada neurona actúa como un "filtro" que divide el espacio de entrada con una frontera lineal.

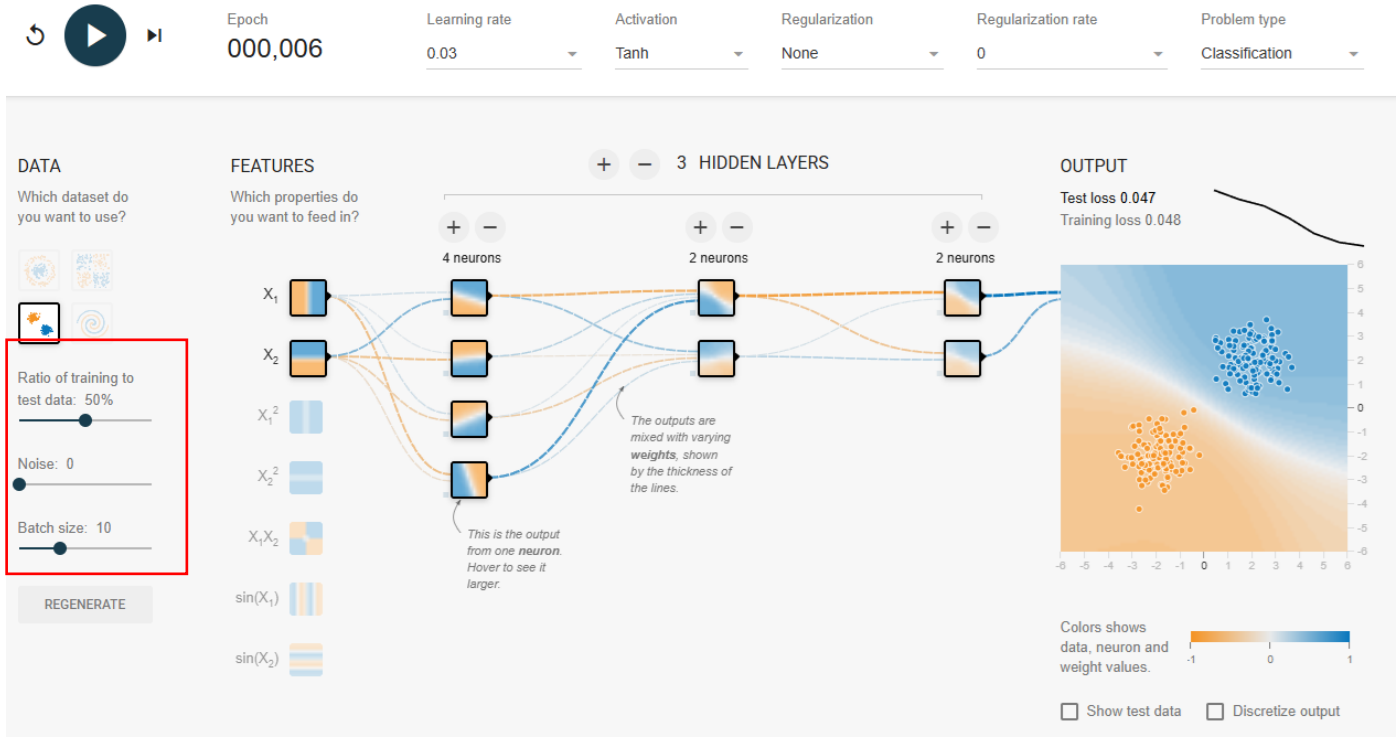
Neuronas en capas más profundas

- Sus gráficas muestran regiones más complejas, porque combinan activaciones de capas anteriores.
- Ahí es donde emergen patrones no lineales: curvas, regiones irregulares, etc.

Frontera de decisión final del modelo entrenado

- Cada punto del plano está coloreado según la **predicción de la red**:
- Por ejemplo, azul para clase 0 y naranja para clase 1.
- La intensidad del color refleja la **confianza** del modelo en su predicción.

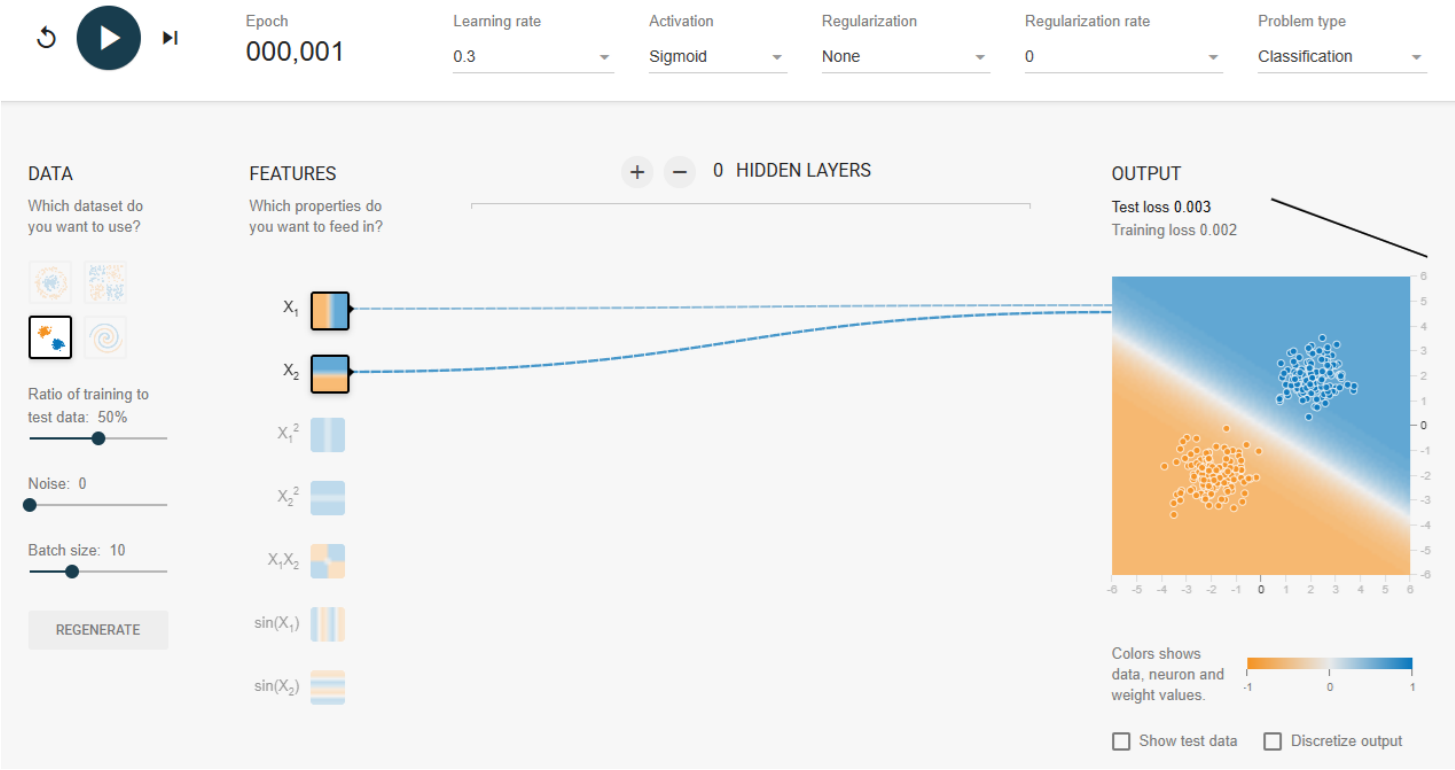
1 Reto: Encontrar la configuración de la red con el mínimo número de capas ocultas y neuronas, que clasifique correctamente (**error <=0.05**) en el menor número de épocas (puedes cambiar la tasa de aprendizaje y función de activación).



# Capas ocultas	# Neuronas	# Épocas	Activación	Tasa A.
3	4, 2, 2	6	Tanh	0.03

1 Reto: Encontrar la configuración de la red con el mínimo número de capas ocultas y neuronas, que clasifique correctamente (**error <=0.05**) en el menor número de épocas (puedes cambiar la tasa de aprendizaje y función de activación).

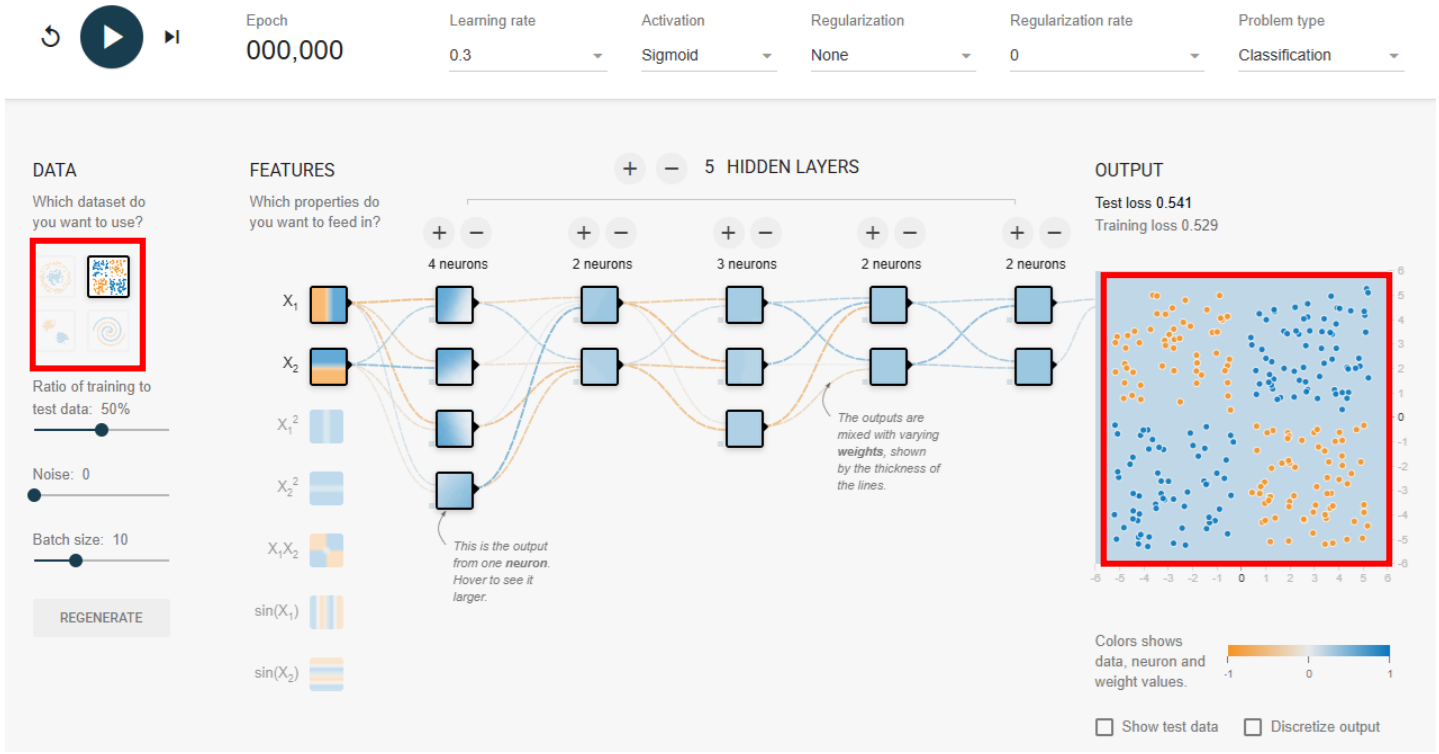
Posible solución



# Capas ocultas	# Neuronas	# Épocas	Activación	Tasa A.
0	0	6	Sigmoide	0.3

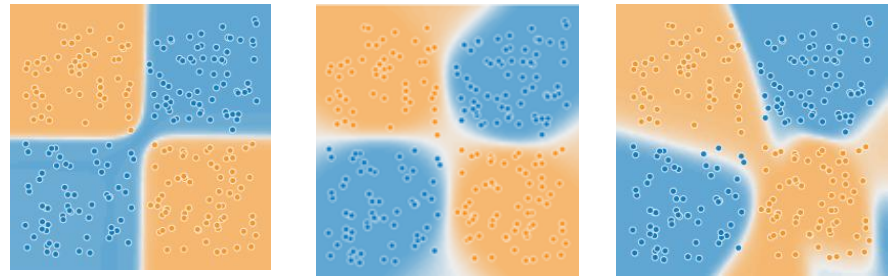
Un problema un poco más difícil

2 **Reto:** Encontrar la configuración de la red con el mínimo número de capas ocultas y neuronas, que clasifique correctamente (**error <=0.05**) en el menor número de épocas (puedes cambiar la tasa de aprendizaje y función de activación).



Capas ocultas # Neuronas # Épocas Activación Tasa A.

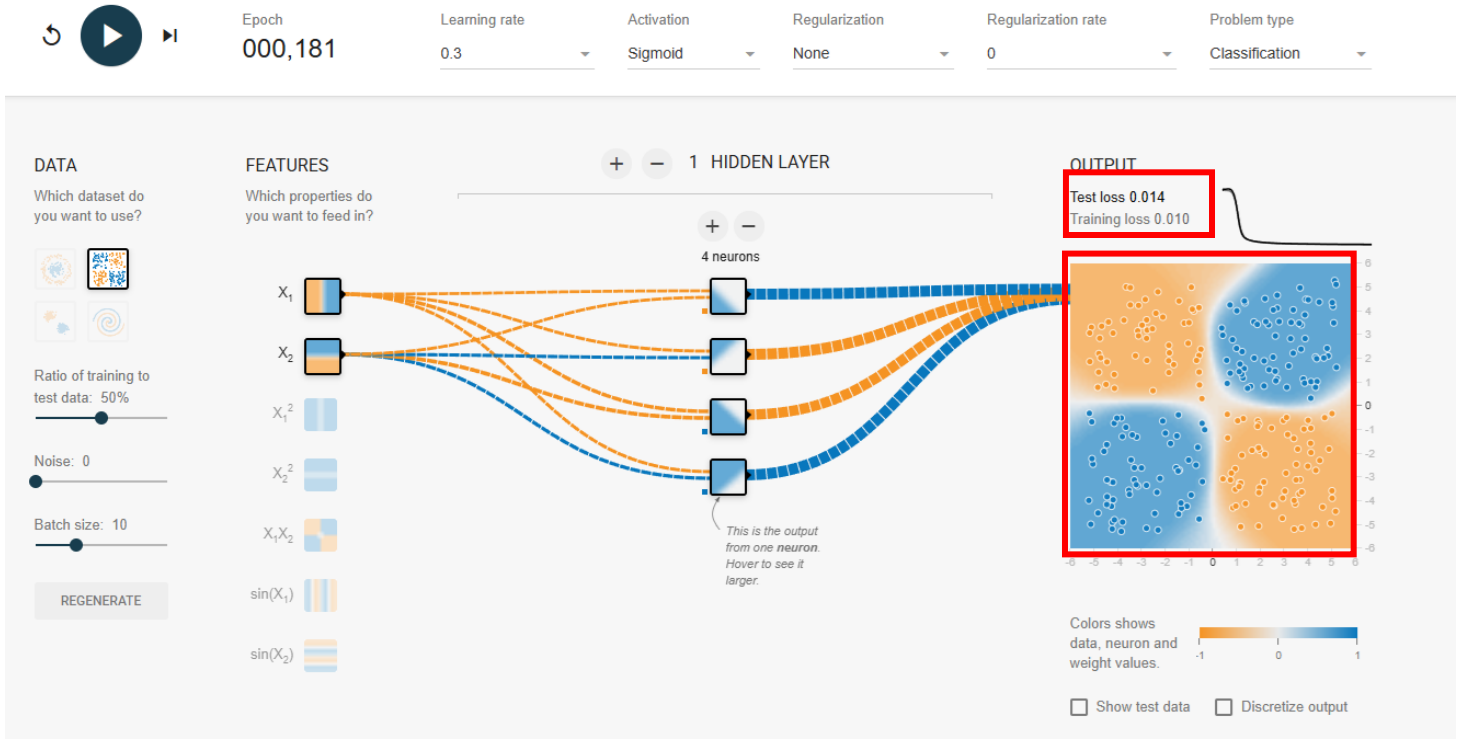
Posibles resultados



Un problema un poco más difícil

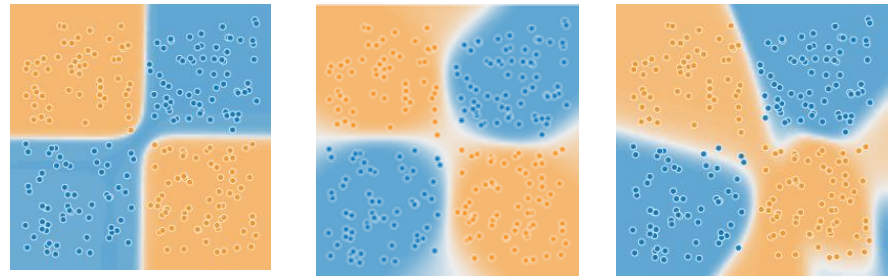
2 Reto: Encontrar la configuración de la red con el mínimo número de capas ocultas y neuronas, que clasifique correctamente (**error <=0.05**) en el menor número de épocas (puedes cambiar la tasa de aprendizaje y función de activación).

Posible solución



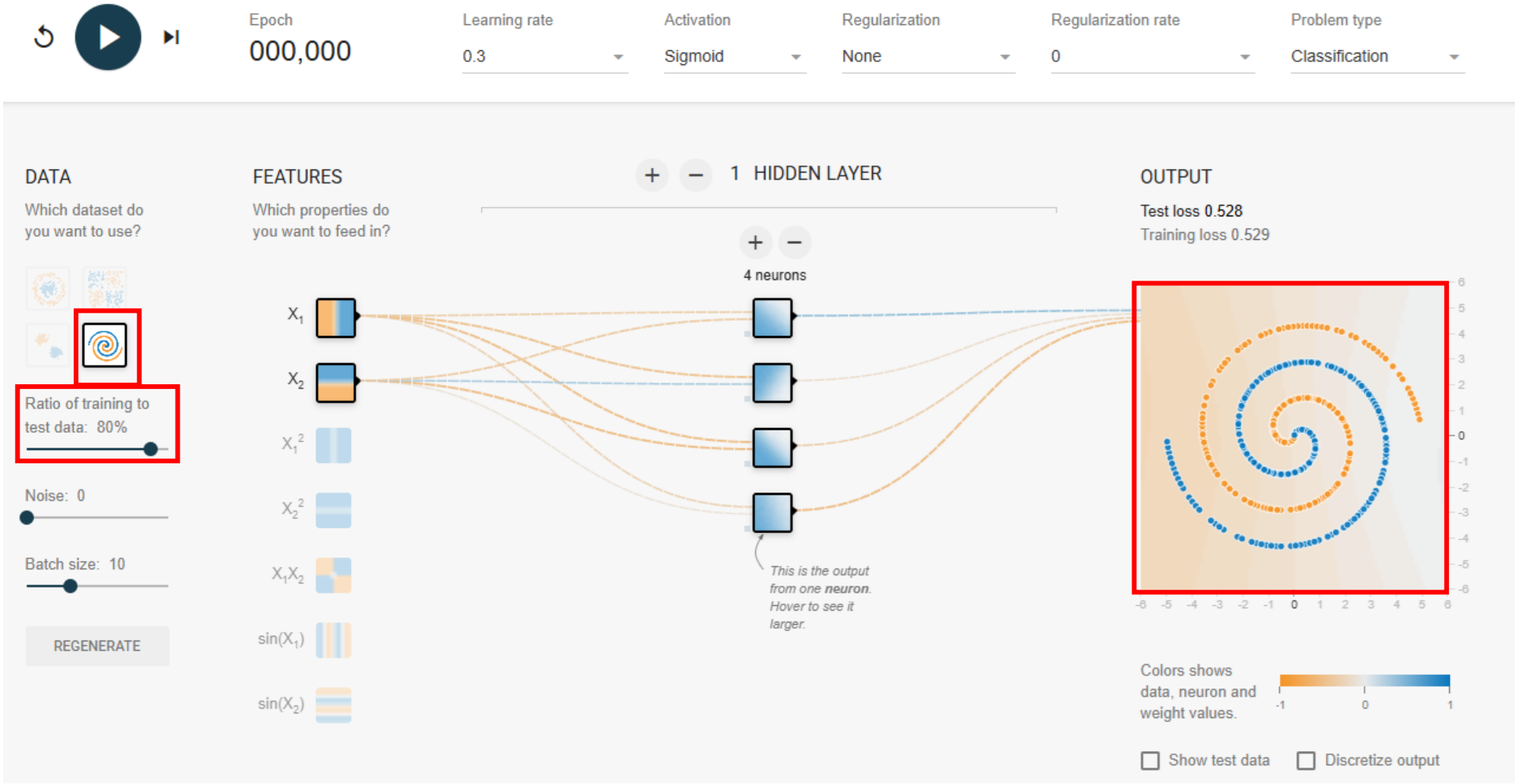
# Capas ocultas	# Neuronas	# Épocas	Activación	Tasa A.
1	4	181	Sigmoide	0.3

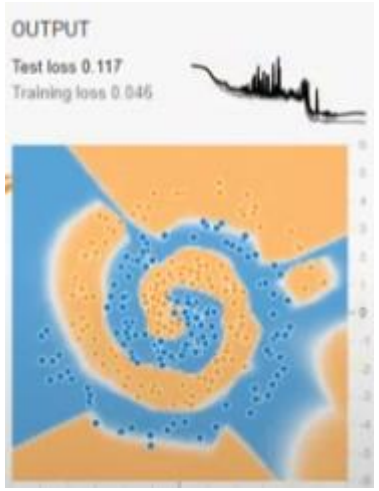
Posibles resultados



Un problema más desafiante

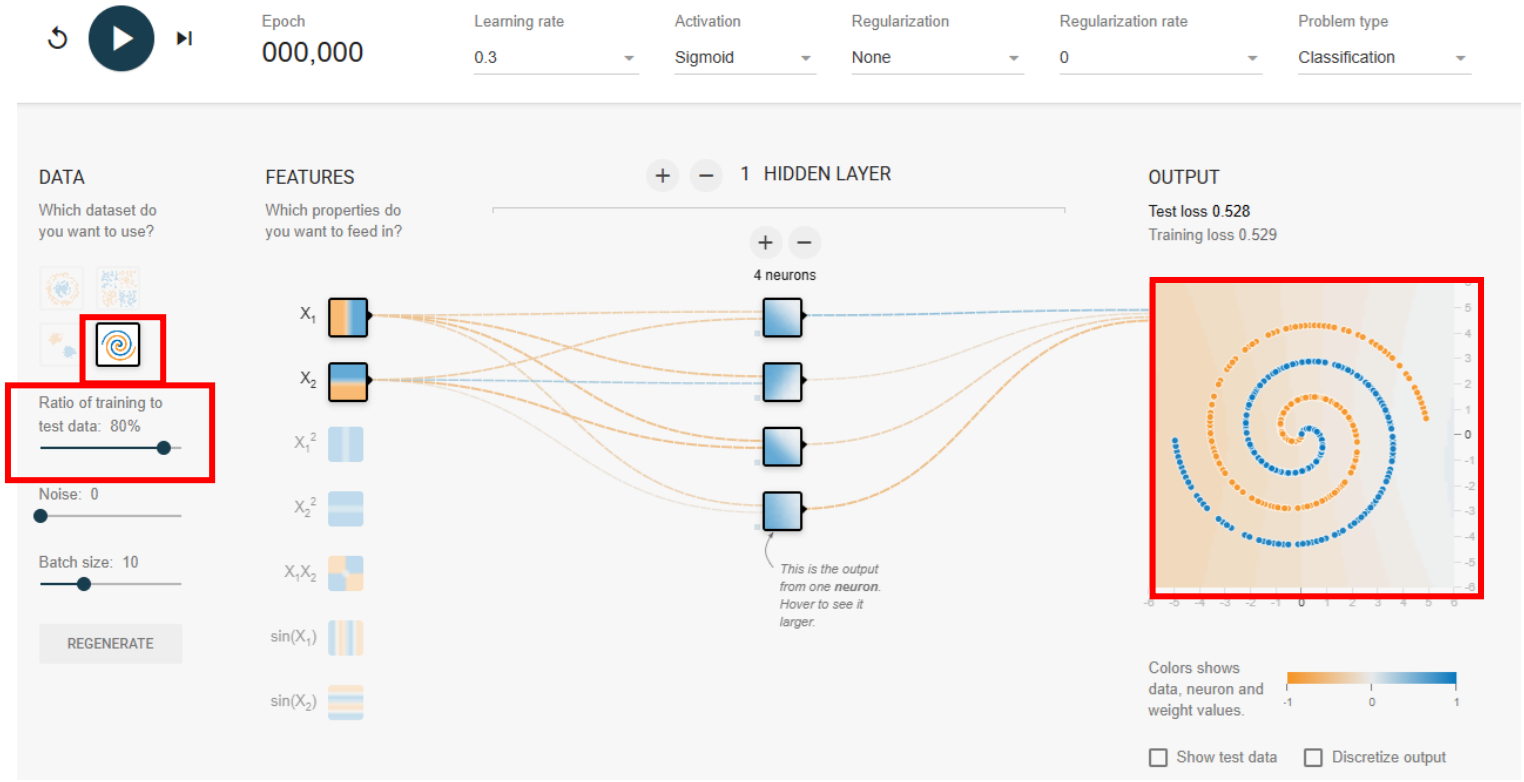
Reto: Encontrar la configuración de la red que clasifique correctamente.

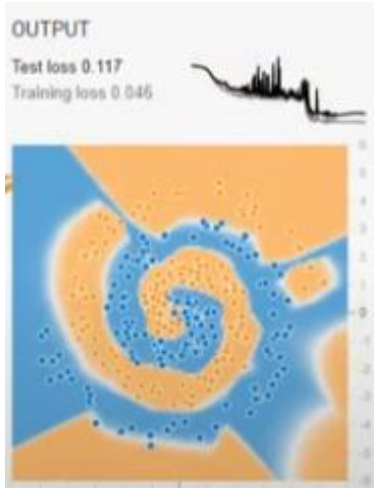




Un problema más desafiante

3 Reto: Encontrar la configuración de la red que clasifique correctamente.





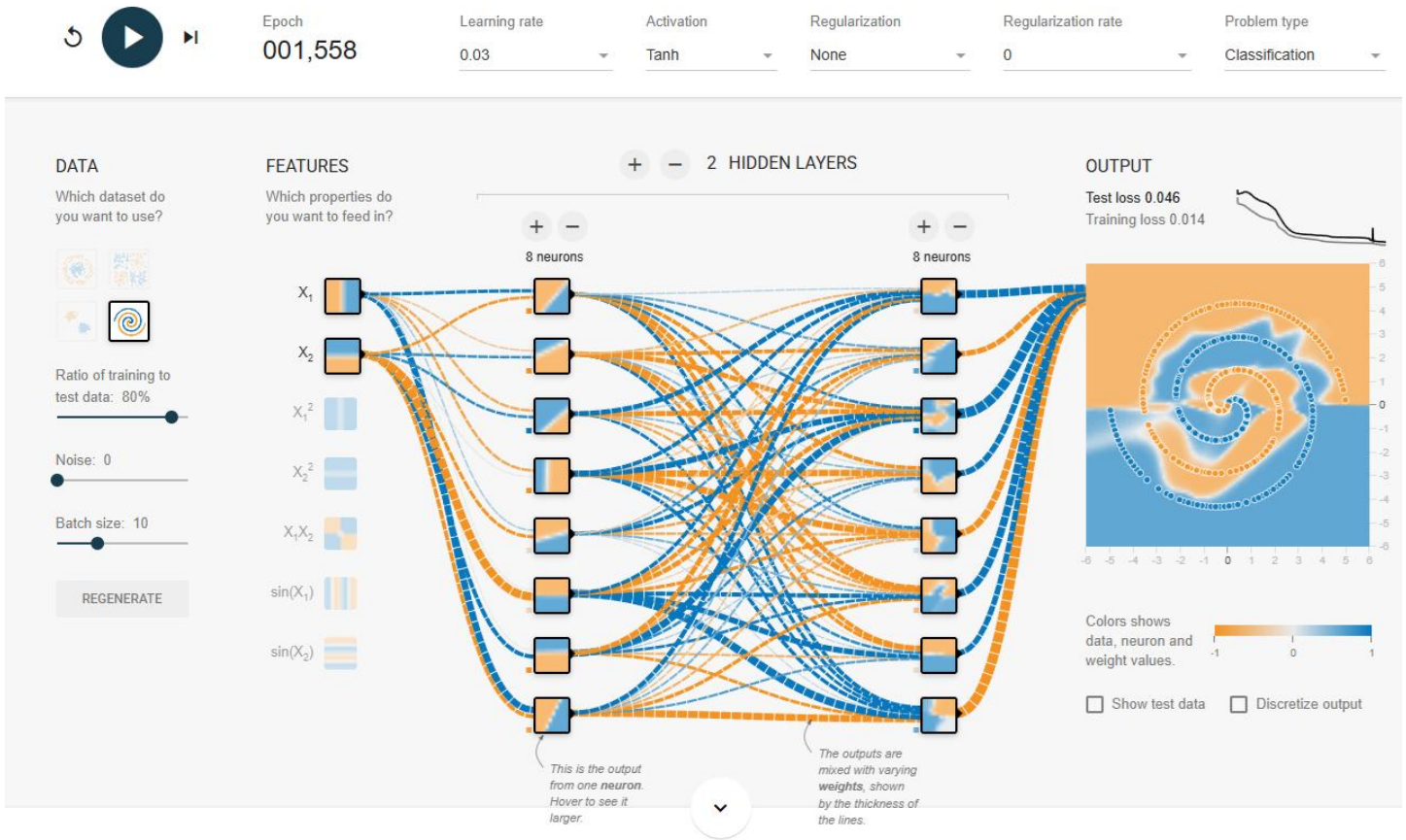
Capas ocultas # Neuronas # Épocas Activación Tasa A.

1. Elige la arquitectura del modelo agregando o eliminando capas ocultas, neuronas, cambiando la función de activación y tasa de aprendizaje.

Un problema más desafiante

3 **Reto:** Encontrar la configuración de la red que clasifique correctamente.

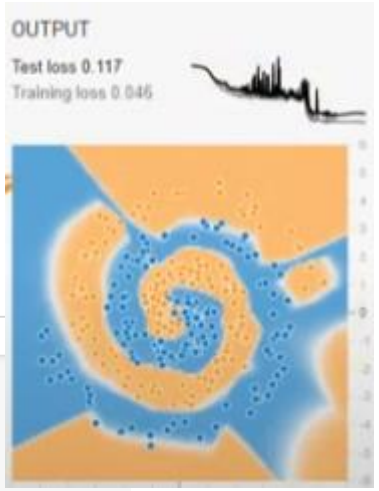
Posible solución



Capas ocultas # Neuronas # Épocas Activación Tasa A.

Un problema más desafiante

Reto: Encontrar la configuración de la red que clasifique correctamente.



Epoch 000,000 | Learning rate 0.03 | Activation Sigmoid | Regularization None | Regularization rate 0 | Problem type Classification

DATA
Which dataset do you want to use?

Ratio of training to test data: 10%

Noise: 0

Batch size: 10

REGENERATE

FEATURES
Which properties do you want to feed in?

X_1, X_2, X_1^2, X_2^2, X_1X_2, $\sin(X_1)$, $\sin(X_2)$

2 HIDDEN LAYERS

2 neurons | 2 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT
Test loss 0.501
Training loss 0.499

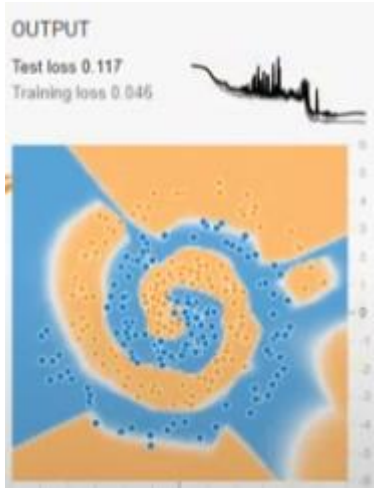
Colors shows data, neuron and weight values. -1 0 1

Show test data | Discretize output

2. Realiza ingeniería de características para mejorar el desempeño del modelo.

Un problema más desafiante

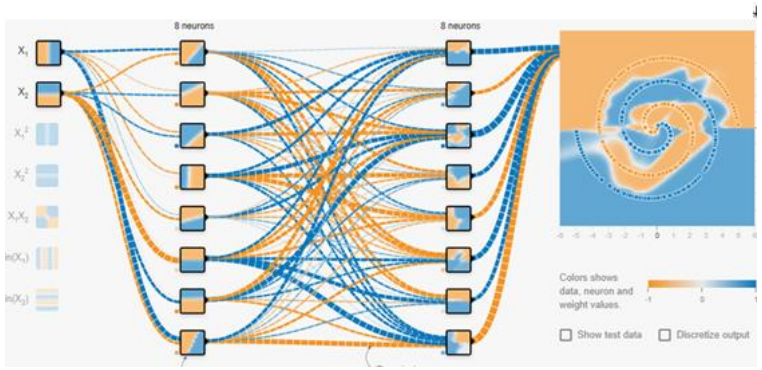
Reto: Encontrar la configuración de la red que clasifique correctamente.



Posible solución

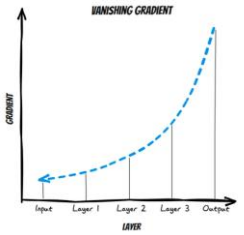
Redes Neuronales en los 80s y 90s

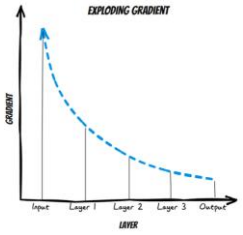
- En los años **80-90**, las redes neuronales artificiales comunes tenían **una o dos capas ocultas**.



- A esto se le llamaba simplemente *redes neuronales* o *modelos conexionistas*.

- Eran **difíciles de entrenar** con más capas porque:
  - aparecía el **desvanecimiento/explosión del gradiente**,
  - la **potencia de cómputo** era limitada,
  - y no había grandes **datasets**.





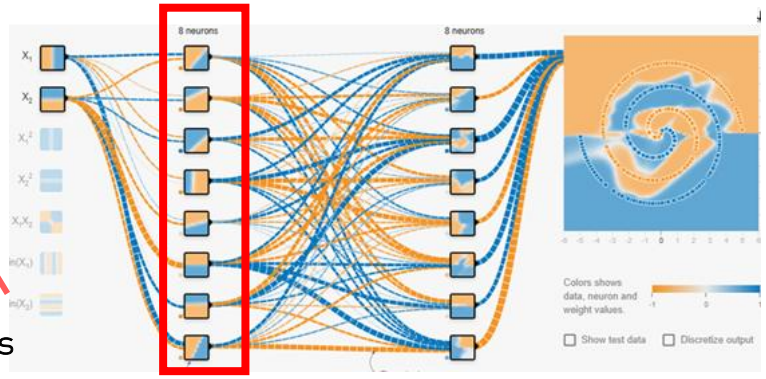
CPUs lentas y sin GPUs accesibles para entrenamiento

En los 90s, **"grande" significaba miles de ejemplos**.

Redes Neuronales en los 80s y 90s

- La investigación se centraba en **SVMs**, **Random Forest**, **Boosting**, que funcionaban muy bien con datasets pequeños y características de los datos diseñados a mano (**feature engineering**).

Datos crudos



- Como las redes neuronales de la época tenían muy pocas capas ocultas (1 o 2).
  - Eso significaba que **no podían aprender representaciones jerárquicas complejas directamente de los datos crudos**.

- Por lo tanto, lo que se hacía era:

1. Un **experto en el dominio** diseñaba las *features* relevantes.
2. Esas *features* ya procesadas eran las que se usaban como **input a la red neuronal** (o a SVMs, regresión logística, etc.).

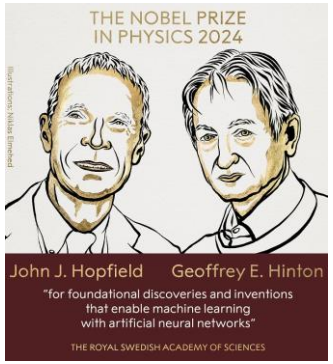
En visión:
- Extracción manual de bordes, histogramas de gradientes (HOG), SIFT, SURF.

En NLP:
- bolsas de palabras (*bag of words*), *n-gramas*.

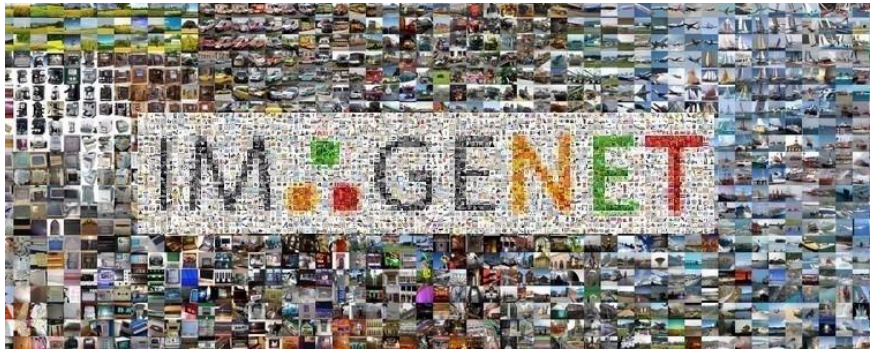
En señales:
- estadísticas (media, varianza), transformadas de Fourier, wavelets.

- Los modelos (SVMs, Random Forests, regresión logística) **dependían totalmente de que los humanos diseñaran buenas representaciones de los datos**.

Redes Neuronales en los 2000s



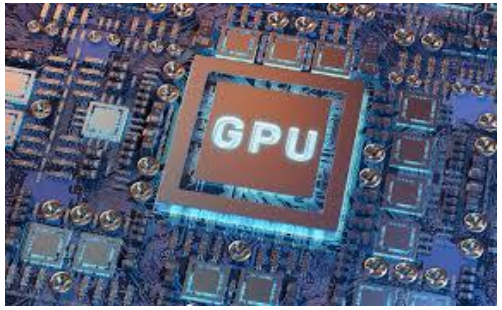
- Nuevos **datasets grandes** empezaron a aparecer:

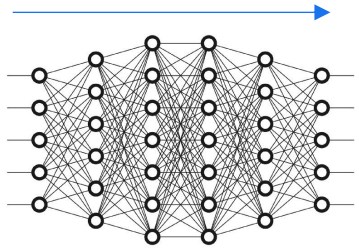


(2009) → Más de 14 millones de imágenes

Más de 20,000 categorías (globos, fresas)

- **GPUs programables** empezaron a usarse fuera de gráficos (CUDA apareció en 2006).





- **Métodos de entrenamiento mejorados**

Geoffrey Hinton, Reducing the dimensionality of data with neural networks, Science, 2006, introdujeron una idea brillante:

1. Entrar la primera capa con los datos crudos usando Restricted Boltzmann Machines o autoencoders.
2. Fijas sus pesos y calculas las representaciones ocultas.
3. Entrenas la segunda capa sobre esas presentaciones.
4. Repites el proceso capa por capa.
5. Al final, conectas todas las capas, y haces un fine-tuning supervisado con backpropagation clásico.

Se logro:

• Cada capa aprendía una **buena inicialización** antes de entrenar la siguiente.
• Se evitaba el problema del gradiente que se "perdía" al inicio de la red.
• Permitió, por primera vez, entrenar **redes profundas de verdad** (5–10 capas) en datasets como MNIST y ver mejoras notables.
• **Resolvió el mayor obstáculo del entrenamiento profundo**: la inicialización de los pesos (se quedaban atrapados en mínimos locales).

Redes Neuronales en los 2010s

- Aunque hoy casi no usamos RBMs (Restricted Boltzmann Machines), esa técnica fue **el puente que permitió pasar de redes superficiales a redes profundas modernas**.

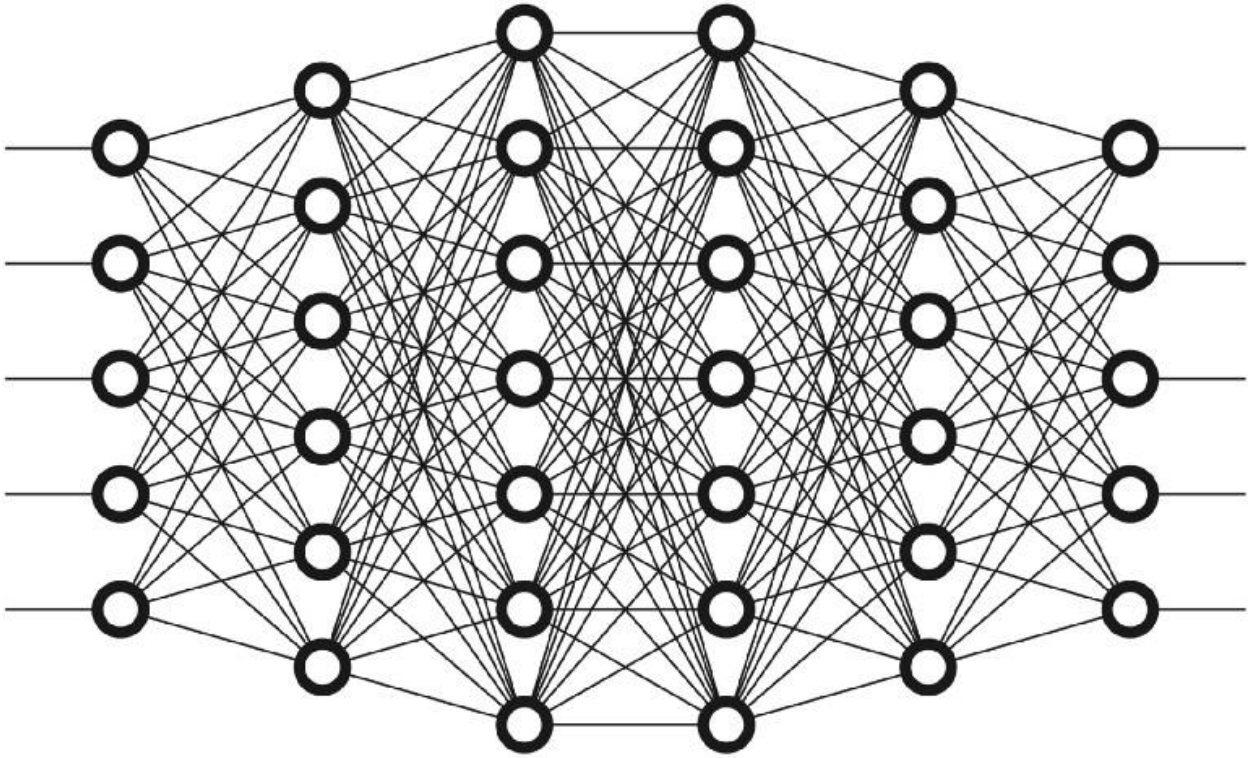
- Posteriormente, con **ReLU (2010), dropout (2012), batch norm (2015)** y GPUs más rápidas, el pretraining dejó de ser necesario para visión y NLP.

- Sin embargo, la idea de *pretraining + fine-tuning* **sobrevive en los Transformers** actuales:
  - Se hace un pretraining masivo (auto-supervisado) → luego fine-tuning para la tarea específica.

Aprendizaje Profundo

El aprendizaje profundo es una clase de algoritmos de aprendizaje automático que utiliza **múltiples capas apiladas** de unidades de procesamiento para **aprender representaciones de alto nivel** a partir de datos no estructurados.

ResNet contiene 152 capas



Datos para Aprendizaje Profundo

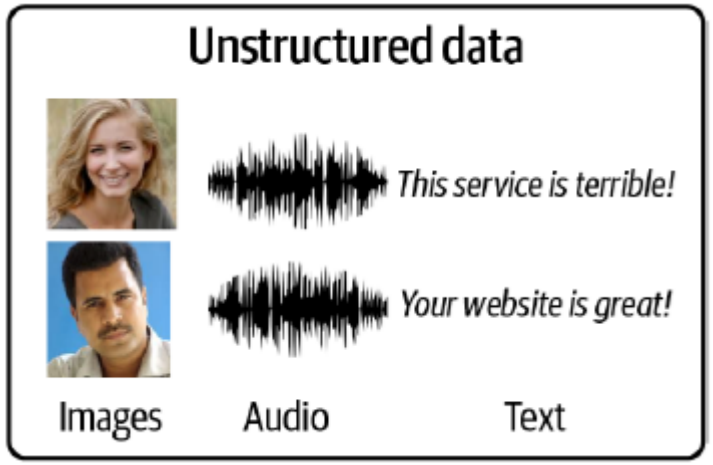
Datos tabulares que están organizados en columnas de características que describen cada observación.

Structured data

ID	Age	Gender	Height (cm)	Location
0001	54	M	186	London
0002	35	F	166	New York
0003	62	F	170	Amsterdam
0004	23	M	164	London
0005	25	M	180	Cairo
0006	29	F	181	Beijing
0007	46	M	172	Chicago

DL también puede ser aplicado a este tipo de datos, pero su poder real es con datos no estructurados.

Se refiere a aquellos datos que no están organizados de manera natural en columnas de características, como las imágenes, audio y texto.



Cada pixel, frecuencia o carácter por sí solo aporta muy poca información.
Hay un alto grado de dependencia espacial y/o temporal.

Hay estructura espacial

Hay estructura temporal

Hay estructura temporal

Video

Hay estructura espacial y temporal

Técnicas "tradicionales" como regresión logística y random forest tienen un desempeño muy pobre con este tipo de datos.

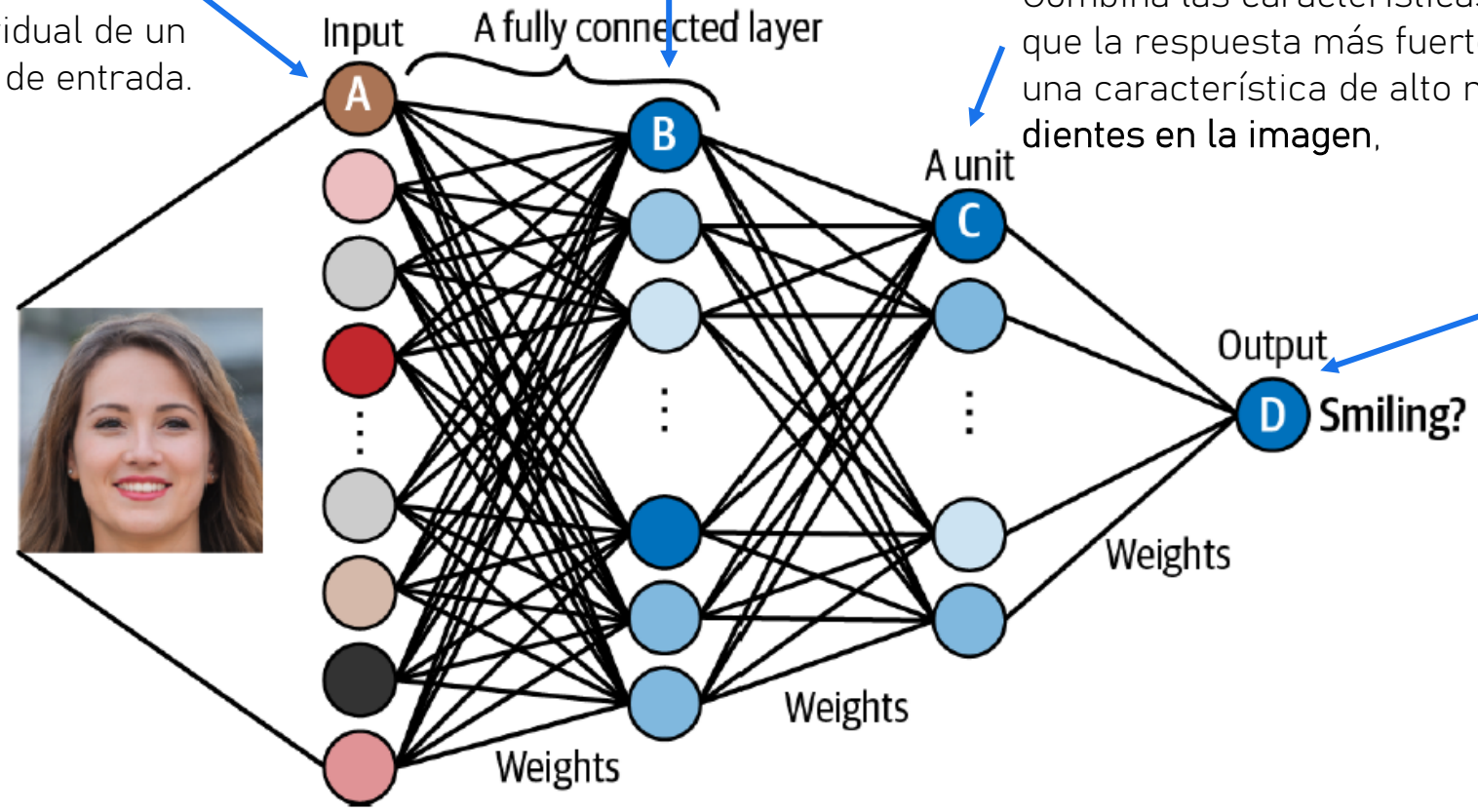
Aprendiendo características de alto nivel

Recibe el valor para un canal individual de un píxel de entrada.

Combina los valores de entrada de tal forma que la respuesta sea más fuerte cuando esté presente una característica de bajo nivel, como la **presencia de bordes**.

Combina las características de bajo nivel de tal forma que la respuesta más fuerte sea cuando esté presente una característica de alto nivel, como que se vean los **dientes en la imagen**,

Combina las características de alto nivel de tal forma que la respuesta más fuerte sea cuando la persona en la imagen original esté sonriendo.



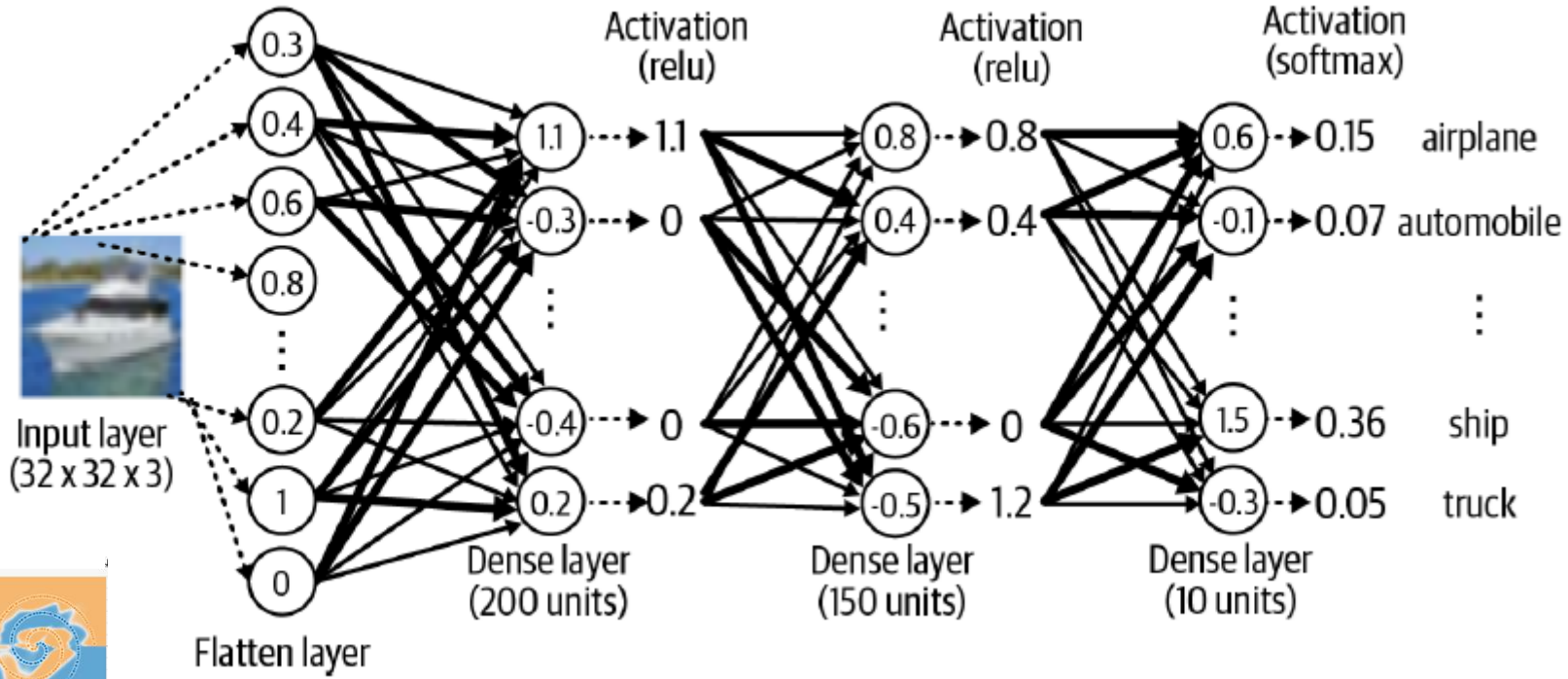
La imagen se transforma en cada capa/ forward pass

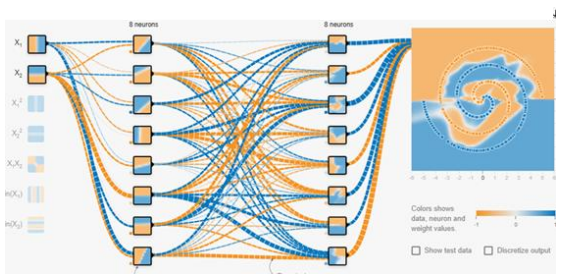
Perceptrón Multicapa usando Keras

Gracias a estos avances, ahora podemos entrenar un perceptrón multicapa con varias capas ocultas:

Datos crudos.

Sin necesidad de hacer ingeniería de características.





Perceptrón Multicapa usando Keras

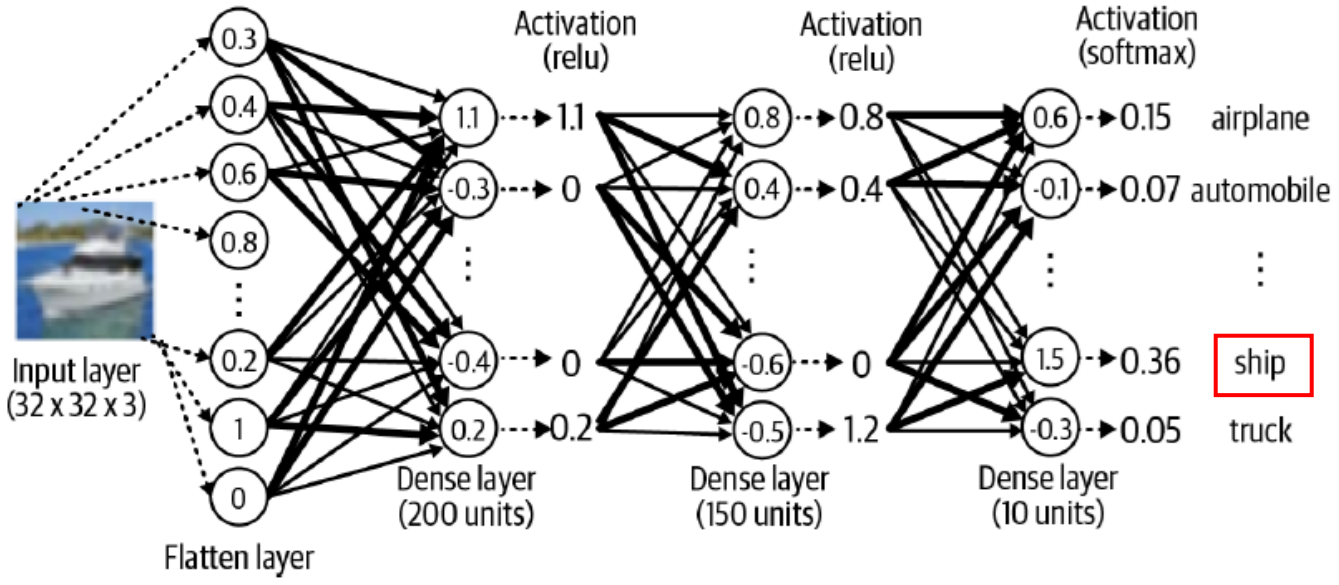
Tarea a resolver: Clasificador de imágenes (discriminativo y no generativo)
Tipo de aprendizaje: Supervisado

10 clases

CIFAR-10 dataset

60,000 32 ×32–pixeles a color

Queremos que aprenda a
mapear entre las imágenes de entrada y las etiquetas de salida.



La representación de la imagen se va transformando

dog

Perceptrón Multicapa usando Keras

1. Preparar los datos
2. Construir el modelo
    - 2.1 Capas
    - 2.2 Funciones de activación
3. Compilar el modelo
    - 3.1 Optimizador
    - 3.2 Función de pérdida
4. Entrenar el modelo
5. Evaluar el modelo

practica_mlp.ipynb


```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ]  import sys
     sys.path.append("/content/drive/My Drive/Colab Notebooks/CursoIAGenerativa")
```

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt

     from tensorflow.keras import layers, models, optimizers, utils, datasets
     from notebooks.utils import display
```

Perceptrón Multicapa usando Keras

1. Preparar los datos

1.1 Descargar el dataset

Etiquetas con la clase a la que pertenece cada imagen.

`(50000, 1)` `(10000, 1)`

```
[ ]  (x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
```

Imágenes

`(50000, 32, 32, 3)`

83.3%

32x32 pixeles

3 canales (RGB)

`(10000, 32, 32, 3)`

16.7%

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

Perceptrón Multicapa usando Keras

1. Preparar los datos

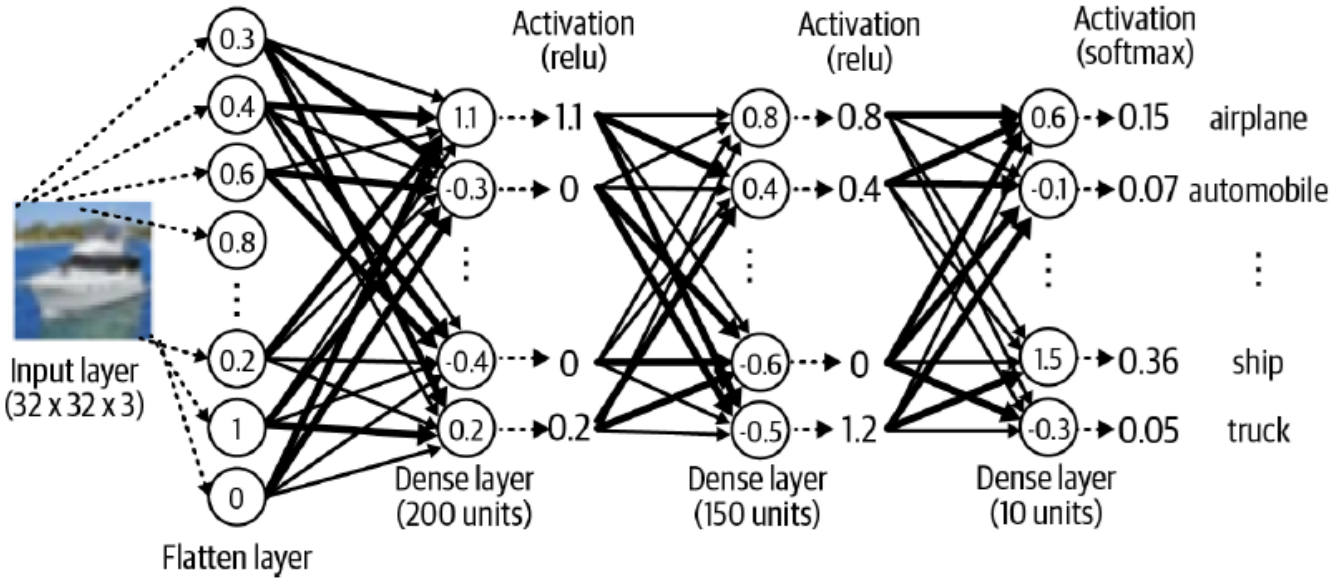
1.2 Escalar los valores de las imágenes

Los datos en `x_train` `x_test` tienen valores en el rango [0,255]

Antes de pasárselas a la red, necesitamos escalar sus valores entre [0,1].

```
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
```

- Facilita el aprendizaje.
- Acelera la convergencia durante el entrenamiento.
- Mejor desempeño del modelo.



Perceptrón Multicapa usando Keras

1. Preparar los datos

1.3 Codificar las etiquetas con one-hot encoding

Los datos en `y_train` `y_test` tienen valores en el rango [0,9]

Como la salida de la red será la probabilidad de que la imagen pertenezca a cada una de las 10 clases, es más adecuado trabajar con un vector binario para cada etiqueta.

```
NUM_CLASSES = 10
[ ]  y_train = utils.to_categorical(y_train, NUM_CLASSES)
     y_test = utils.to_categorical(y_test, NUM_CLASSES)
```

One hot encoding:

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
]
```

```
display(x_train[:10])
print(y_train[:10])
```

- Permite tratar cada clase como una categoría independiente y facilita el cálculo de las probabilidades de pertenencia a cada clase durante el entrenamiento y la inferencia

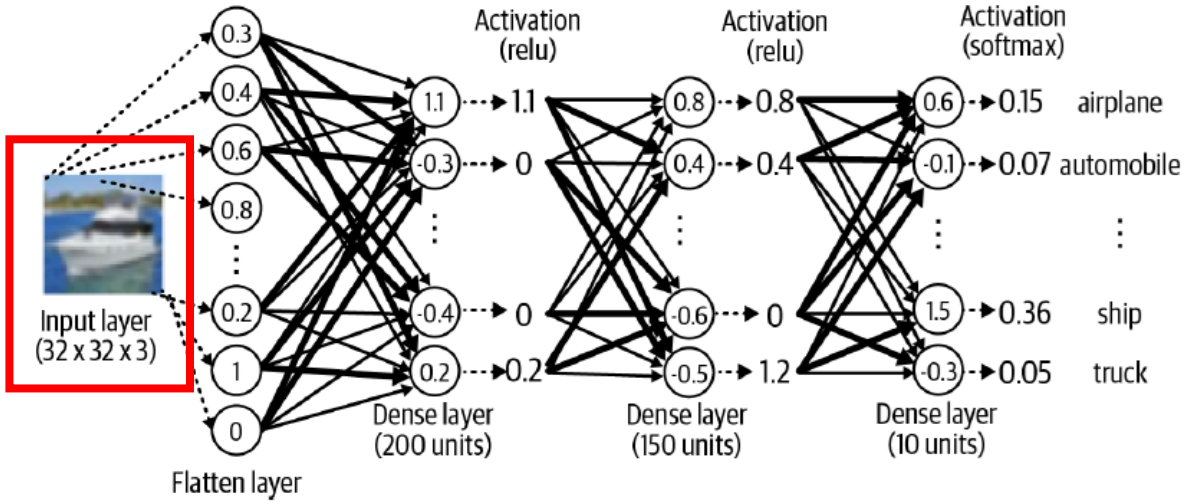
```
display(x_train[:10])
print(y_train[:10])
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación



Le decimos a la red que la forma esperada de cada dato de entrada es un tensor de dimensión 32x32x3.

```
input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)
```

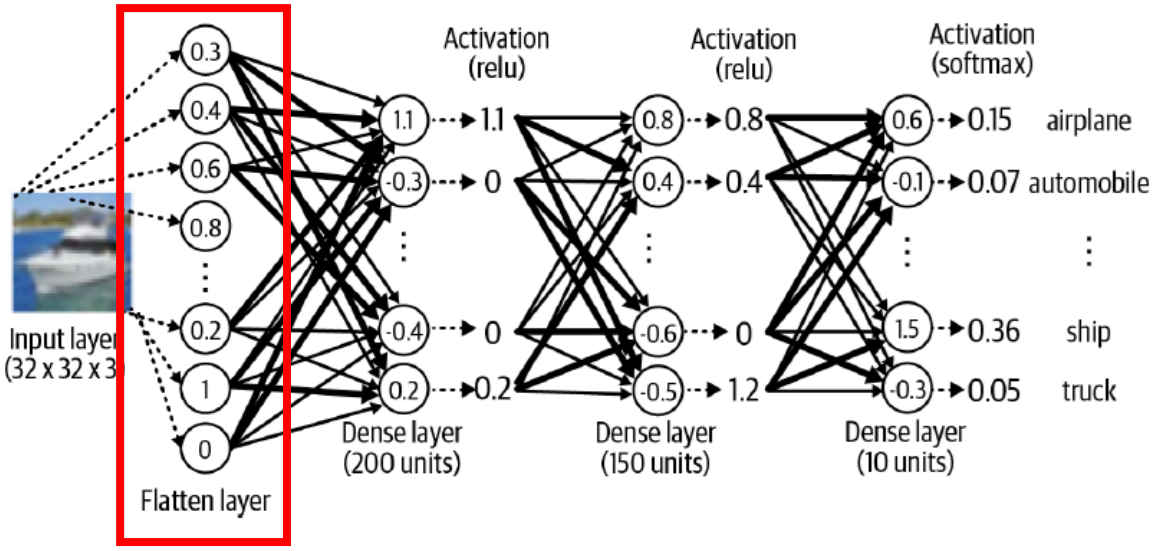
Nota: No especificamos el **tamaño del lote**, ya que podemos pasar cualquier cantidad de imágenes a la capa de entrada sin declararlo explícitamente.

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación



Aplanamos la entrada en un vector usando una capa de tipo *Flatten*.

```
NUM_CLASSES = 10

input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)
```

Necesitamos hacerlo porque la capa siguiente es una capa densa, y ésta requiere como entrada un vector aplanado, y no tensor multidimensional.

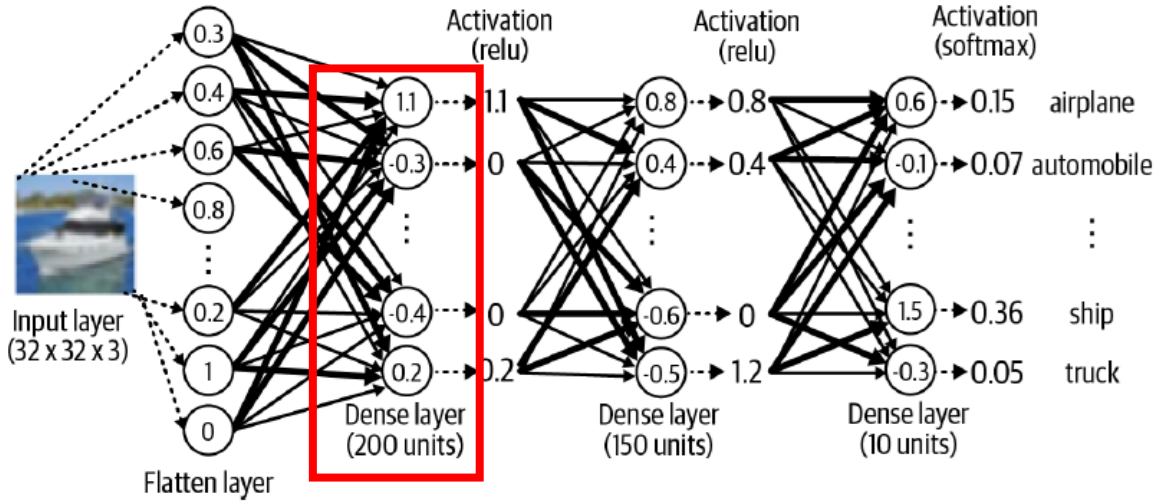
Nota: Hay otro tipo de capas, diferentes a las Dense, que sí requieren que su entrada sea un tensor multidimensional.
Hay que fijarnos en las formas requeridas de las entradas y salidas de cada capa.

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación

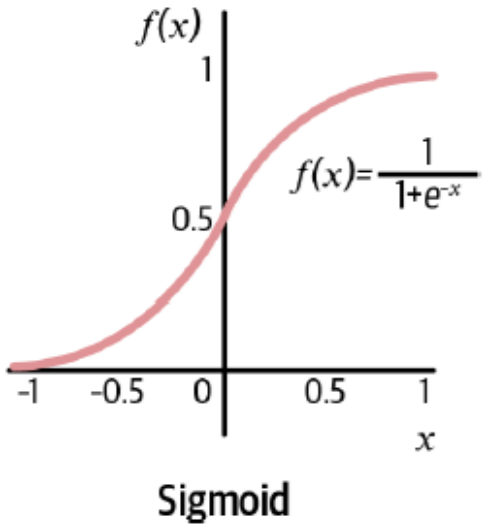


La capa densa es uno de los bloques de construcción de una red neuronal.

Cada neurona/unidad de la capa está conectada a cada neurona/unidad de la capa anterior.

```
NUM_CLASSES = 10

input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)
```

Perceptrón Multicapa usando Keras



Sigmoid

Problema:

Puede causar desvanecimiento del gradiente.

El valor máximo de la derivada ocurre en $x = 0$:

$$f'(0) = 0.25$$

Para $|x|$ grandes (muy positivos o muy negativos):

$$f(x) \to 1 \text{ o } 0 \quad \Longrightarrow \quad f'(x) \to 0$$

Esto significa que **fuera de la región cercana a 0, el gradiente es casi nulo.**

Recomendaciones

No usar sigmoide:

- En capas ocultas en una red profunda por el desvanecimiento del gradiente.

Sí usar sigmoide:

Salidas de clasificación binaria

- La sigmoide transforma cualquier número real en un valor en $[0, 1]$.
- Esto permite interpretarlo como una **probabilidad de pertenecer a la clase positiva**.

Modelos de regresión logística

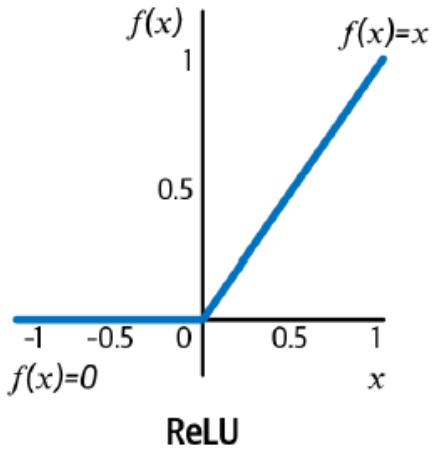
- La regresión logística es básicamente una neurona con activación sigmoide.
- Es el modelo base para clasificación binaria en estadística y ML.

Cuando el rango natural de salida es [0,1]

- Estimar probabilidad de supervivencia,
- Probabilidad de compra,

Puertas en redes recurrentes (LSTM/GRU)

Perceptrón Multicapa usando Keras



ReLU

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x > 0 \\ \text{indefinida} & \text{si } x = 0 \end{cases}$$

Su derivada es 1 (en positivo), lo que evita el desvanecimiento del gradiente que ocurría con sigmoide o tanh.

Ventajas:

Es una de las funciones de activación más confiable en la actualidad para usar entre las capas de una red neuronal profunda, ya que favorecen un entrenamiento estable

- La pérdida debe de ir disminuyendo poco a poco.
- Los gradientes no deben desvanecerse ni explotar.
- Generalización razonable.

Desventaja

Pueden algunas veces morir si siempre dan como salida 0, debido a un sesgo grande hacia un número negativo.

En este caso, el gradiente es 0 y por lo tanto el error ya no se propaga hacia capas anteriores.

Deja de aprender.

Recomendaciones

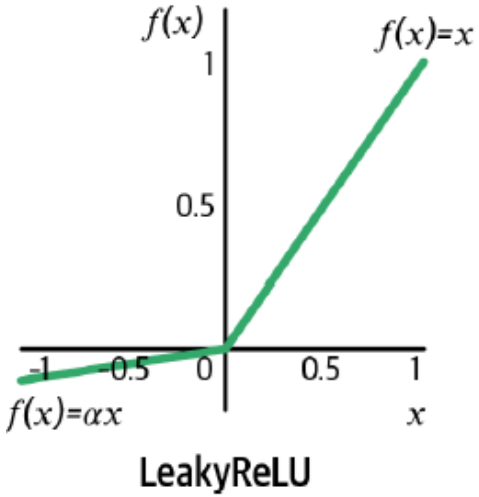
No usar ReLU:

- En la capa de salida de clasificación multiclase
  - Allí se usa softmax
- En la capa de salida de clasificación binaria
  - Se usa sigmoide, no ReLU
- En problemas con pocos datos, ReLU puede ser demasiado agresiva y perder información (mejor usar tanh o sigmoide en esos casos).

Sí usar ReLU:

- Capas ocultas de redes profundas (MLP, CNN, Transformers)
- Problemas con gran cantidad de datos
- Modelos convolucionales y visión por computadora
  - Casi todas las arquitecturas modernas usan ReLU o variantes en sus capas convolucionales/densas.

Perceptrón Multicapa usando Keras



LeakyReLU

Ventajas:

Arreglan el problema de que las neuronas mueran, asegurando que el gradiente siempre sea diferente de 0.

$$f(x) = \begin{cases} \alpha x & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$$

donde α es un valor pequeño (ej. 0.01 o 0.1).

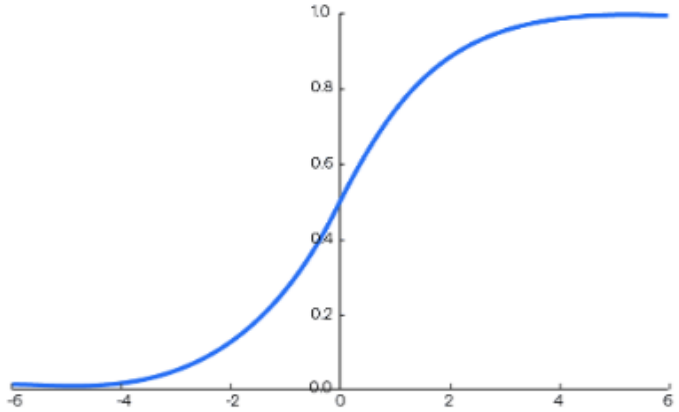
$$f'(x) = \begin{cases} \alpha & \text{si } x < 0 \\ 1 & \text{si } x > 0 \end{cases}$$

Recomendaciones

Sí usar LeakyReLU:

- Cuando observas el problema de *dying ReLU*
- Redes muy profundas sin batch normalization
- Cuando tus datos tienen muchas entradas negativas
  - En dominios donde las activaciones suelen ser negativas (ej. datos centrados alrededor de 0), LeakyReLU evita que se "aplastan" todas en 0.
- Modelos que requieren entrenamiento más estable
  - GANs (Generative Adversarial Networks) son un ejemplo:
    - En el *discriminador*, LeakyReLU es más común que ReLU, porque ayuda a que el modelo no se estanque y capture mejor los detalles.

Perceptrón Multicapa usando Keras



Softmax

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$$

J = número de unidades en la capa.

La función softmax toma un vector de valores reales $z = (z_1, z_2, ., .. z_K)$y los transforma en un **vector de probabilidades**: [0,1]

La suma de todas las salidas = 1

Recomendaciones

No usar Softmax:

- Capas ocultas
  - No se usa softmax en capas intermedias, porque su efecto de "normalizar" puede bloquear la propagación de información útil.

Sí usar Softmax:

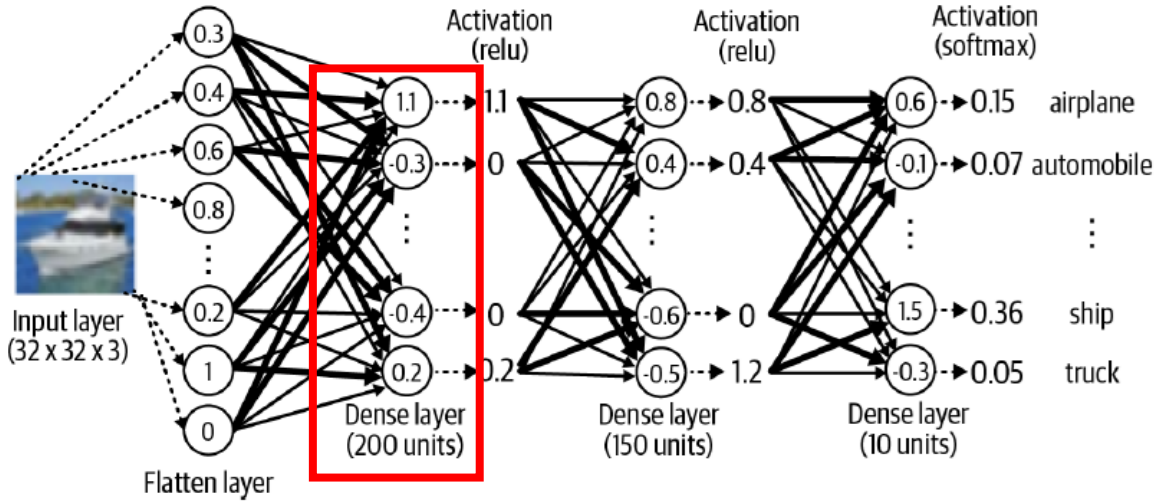
- **Clasificación multiclase** (una sola clase correcta).
- Cuando necesitas interpretar la salida como probabilidades normalizadas
  - Ejemplo: un modelo que recomienda un producto entre N posibles.
  - Softmax da una "confianza relativa" de cada opción.
- En mecanismos de atención (Transformers)
  - El softmax convierte *scores* de atención en pesos normalizados entre 0 y 1.
  - Esto permite que los pesos se interpreten como "cuánta atención asignar" a cada token.

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación



Es en una capa oculta de una red profundas

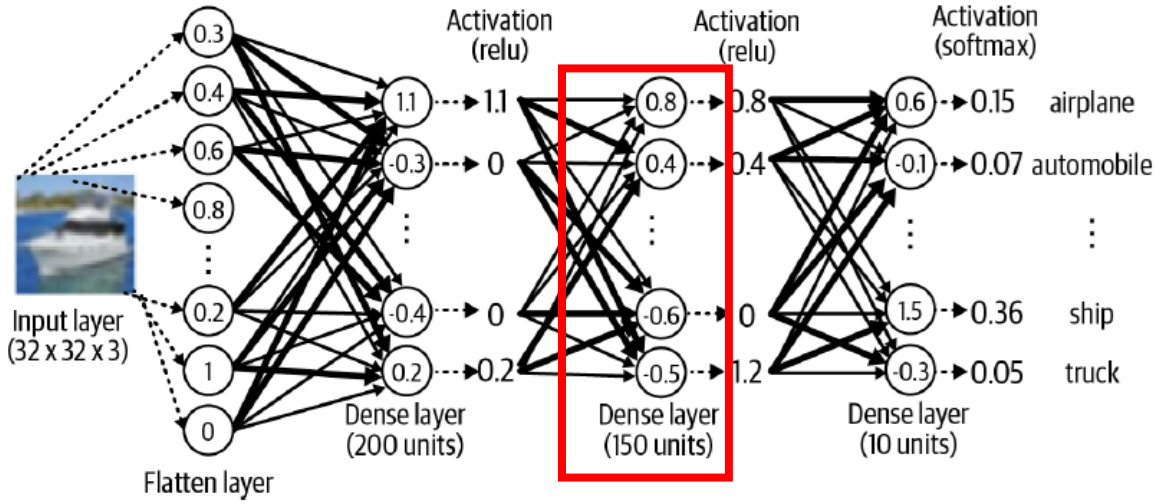
```
NUM_CLASSES = 10

input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación



Es en una capa oculta de una red profundas

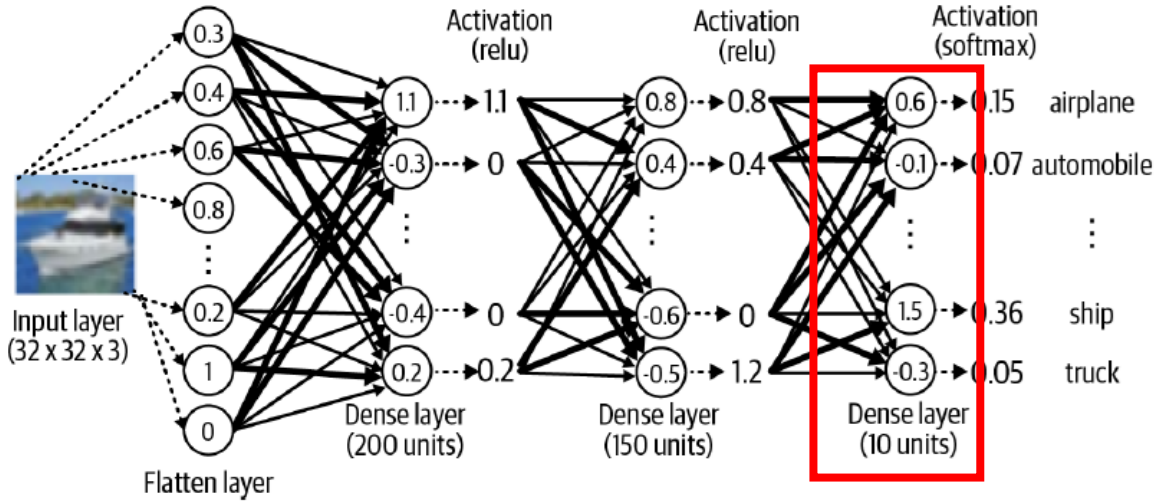
```
NUM_CLASSES = 10

input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación



Clasificación multiclase
(una sola clase correcta).

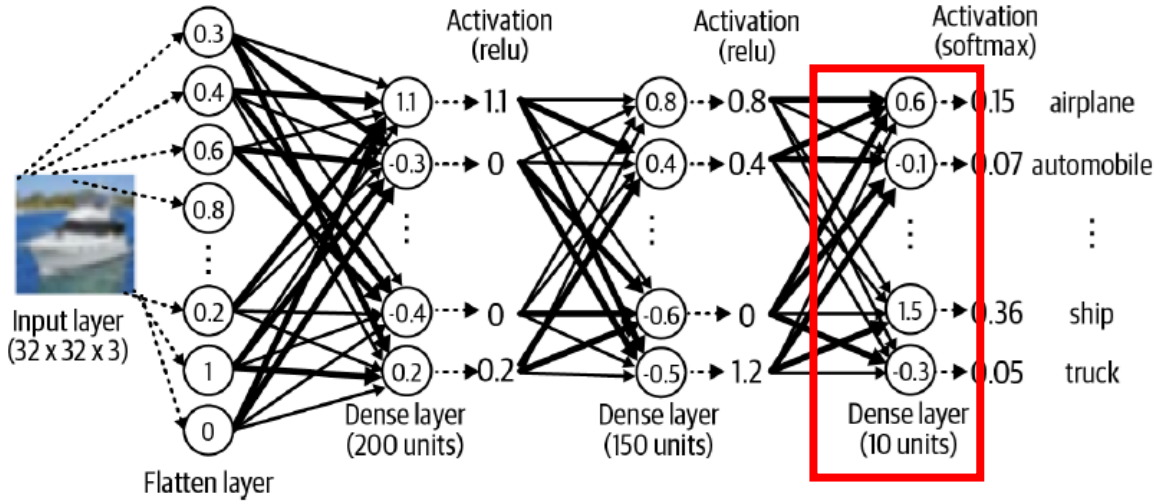
```
NUM_CLASSES = 10

input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación



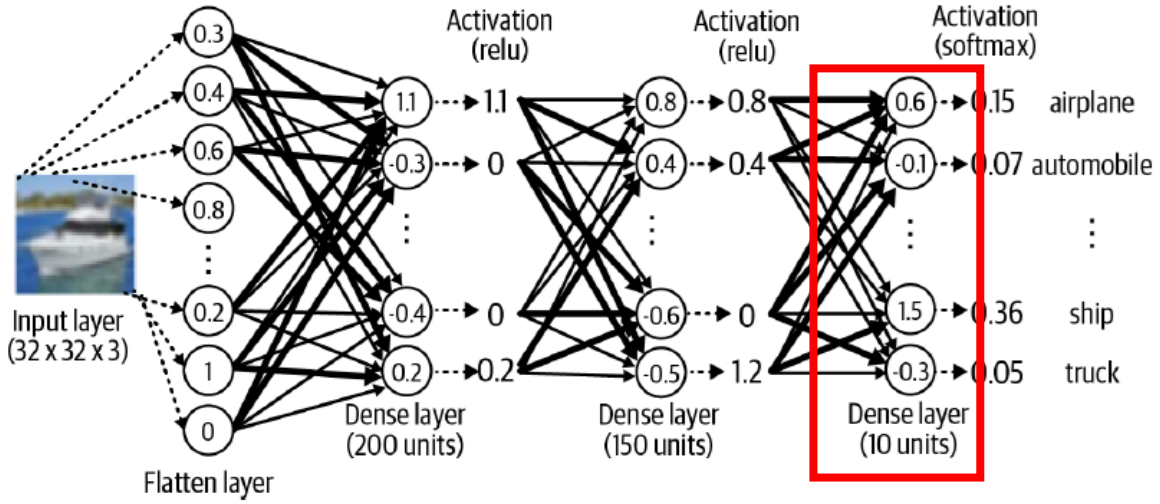

```
NUM_CLASSES = 10

input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación



Indica que aún no sabe el número de observaciones que se le pasarán de entrada.

`x_train` `x_test`

Misma forma de

```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 200)	614,600
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

```
Total params: 646,260 (2.47 MB)
Trainable params: 646,260 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

Misma forma de

`y_train` `y_test`

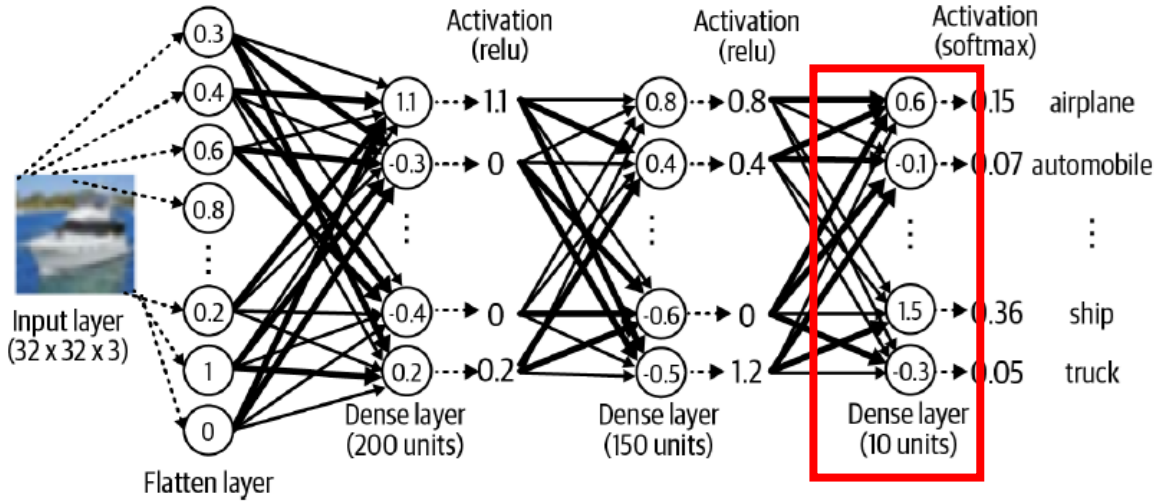
Nota: No necesita saber el número de observaciones, que puede ser 1 o 1000 a la vez, porque las operaciones con los tensores se llevan a cabo con todas las observaciones al mismo tiempo usando álgebra lineal.

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación




```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 200)	614,600
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

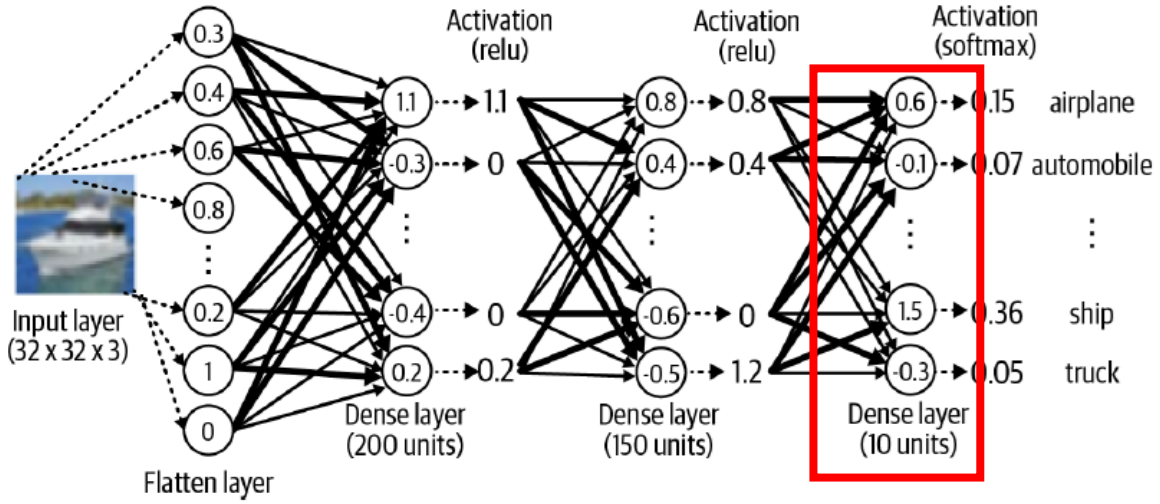
```
Total params: 646,260 (2.47 MB)
Trainable params: 646,260 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación




```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 200)	614,600
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

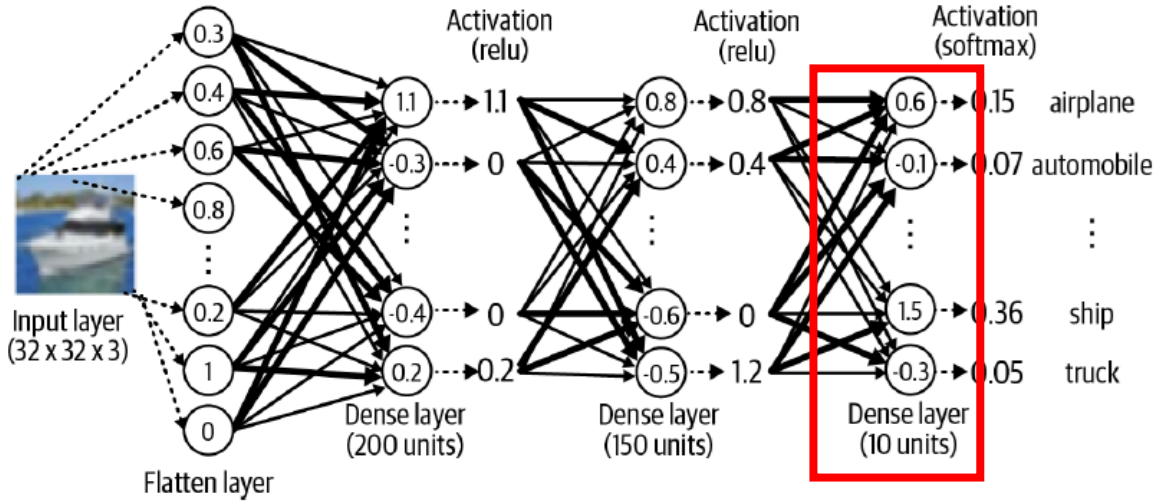
```
Total params: 646,260 (2.47 MB)
Trainable params: 646,260 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación




```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 200)	614,600
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

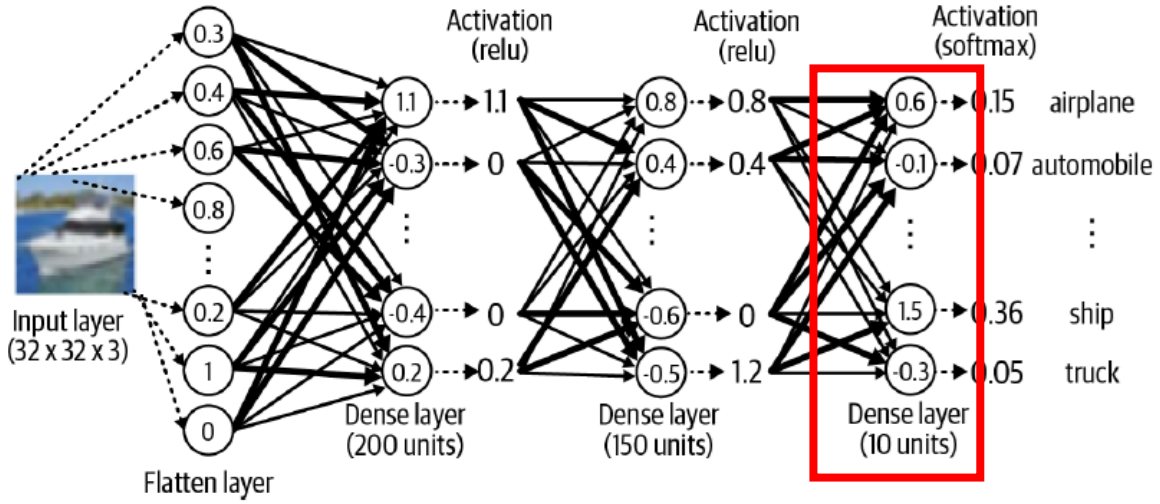
```
Total params: 646,260 (2.47 MB)
Trainable params: 646,260 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación




```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 200)	614,600
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

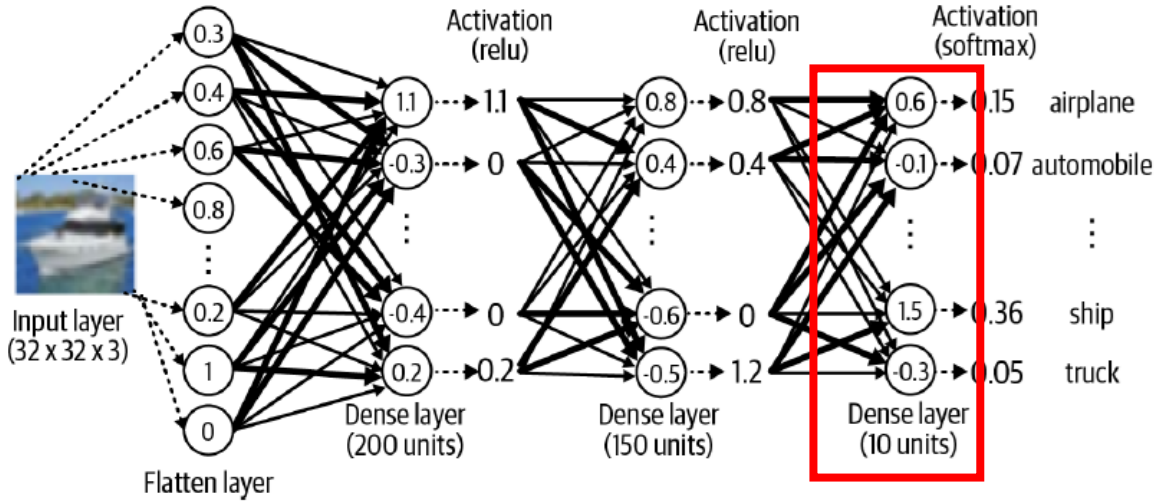
```
Total params: 646,260 (2.47 MB)
Trainable params: 646,260 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación




```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 200)	614,600
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

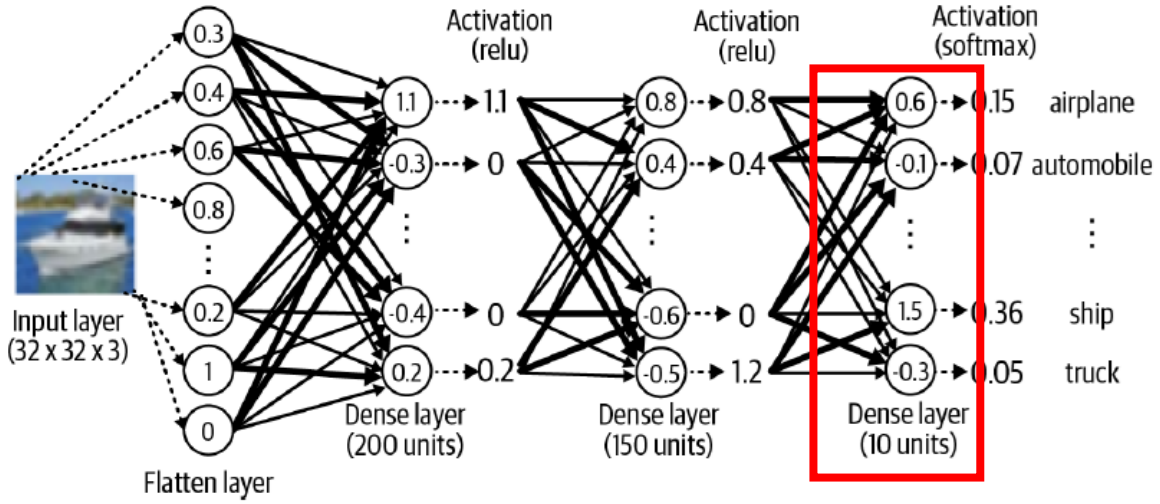
```
Total params: 646,260 (2.47 MB)
Trainable params: 646,260 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

Perceptrón Multicapa usando Keras

2. Construir el modelo

2.1 Capas

2.2 Funciones de activación




```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 200)	614,600
dense_1 (Dense)	(None, 150)	30,150
dense_2 (Dense)	(None, 10)	1,510

```
Total params: 646,260 (2.47 MB)
Trainable params: 646,260 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

Son aquellos que **no se actualizan durante el entrenamiento.**
Por ejemplo:

Capas congeladas (frozen layers)

- Cuando usas **transfer learning** y "congelas" las capas de un modelo preentrenado.
- Esos pesos existen y afectan la salida, pero no se modifican.

Perceptrón Multicapa usando Keras

$$\text{accuracy} = \frac{\text{número de predicciones correctas}}{\text{total de ejemplos}}$$

3. Compilar el modelo

3.1 Optimizador

2.2 Función de pérdida

Configurar el modelo para el entrenamiento.

```
opt = optimizers.Adam(learning_rate=0.0005)
model.compile(
    loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"]
)
```

Error cuadrático medio

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2$$

Si tu red está diseñada para resolver un problema de regresión (la salida es continua).

Binary cross-entropy

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Clasificación binaria
Si estás resolviendo un problema en donde cada observación puede pertenecer solo a una de dos categorías.

Categorical cross-entropy

$$-\sum_{i=1}^{n} y_i \log(p_i)$$

Clasificación multiclase
Si estás resolviendo un problema en donde cada observación pertenece a solo una clase.

Categorical cross-entropy

$$-\sum_{i=1}^{n} y_i \log(p_i)$$

Clasificación multietiqueta
Si estás trabajando en un problema en el que cada observación puede pertenecer a múltiples clases simultáneamente.

Perceptrón Multicapa usando Keras

4. Entrenar el modelo

Las etiquetas correctas de cada imagen codificadas con one-hot encoding

El número de veces que se le mostrará a la red el conjunto de entrenamiento completo.

```
model.fit(x_train, y_train, batch_size=32, epochs=10, shuffle=True)
```

Las imágenes en crudo aplanadas

Determina cuántas imágenes se pasarán a la red en cada paso de entrenamiento.

Los batches se tomarán aleatoriamente sin reemplazo del conjunto de entrenamiento en cada paso de entrenamiento.

Nota: Entre más grande sea el tamaño del batch, más estable será el cálculo del gradiente, pero será más lento cada paso de entrenamiento.

Se recomienda usar un número entre 32 y 256

Perceptrón Multicapa usando Keras

4. Evaluar el modelo

Hasta ahora sabemos que el modelo tiene un accuracy alrededor del 50% en el conjunto de entrenamiento, pero ¿cómo se desempeña con datos que nunca ha visto?

```
model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4879 - loss: 1.4613
[1.47357976436615, 0.4787999987602234]
```

Es aún alrededor del 50%.

Si el modelo estuviera adivinando aleatoriamente, entonces lograría un desempeño aproximado del 10% (porque hay diez clases).

Es un muy buen resultado, dado que estamos usando una red muy básica.

Perceptrón Multicapa usando Keras

4. Evaluar el modelo

Para ver algunas de las predicciones del modelo:

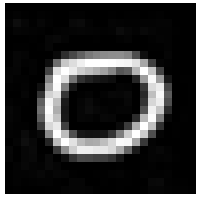
```
n_to_show = 10
indices = np.random.choice(range(len(x_test)), n_to_show)

fig = plt.figure(figsize=(15, 3))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

for i, idx in enumerate(indices):
    img = x_test[idx]
    ax = fig.add_subplot(1, n_to_show, i + 1)
    ax.axis("off")
    ax.text(
        0.5,
        -0.35,
        "pred = " + str(preds_single[idx]),
        fontsize=10,
        ha="center",
        transform=ax.transAxes,
    )
    ax.text(
        0.5,
        -0.7,
        "act = " + str(actual_single[idx]),
        fontsize=10,
        ha="center",
        transform=ax.transAxes,
    )
    ax.imshow(img)
```

Ejercicio de tarea

Programar un perceptrón multicapa que aprenda a clasificar el dataset MNIST

70,000 imágenes de dígitos escritos a mano de 28x28 pixeles.

```
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()
```



0