Entrenamiento de Redes Neuronales - Parte 1

Gonzalo Uribarri • Gabriel B. Mindlin

Sistemas dinámicos e inteligencia artificial aplicados al modelado de datos

@gonzauri

Objetivos de la práctica de hoy

- Familiarizarse con Keras.
- Implementar un primer modelo de clasificación con el dataset Fashion-MNIST.
- Explorar distintas arquitecturas de redes Feedforward (secuenciales).

Estructura de la Práctica

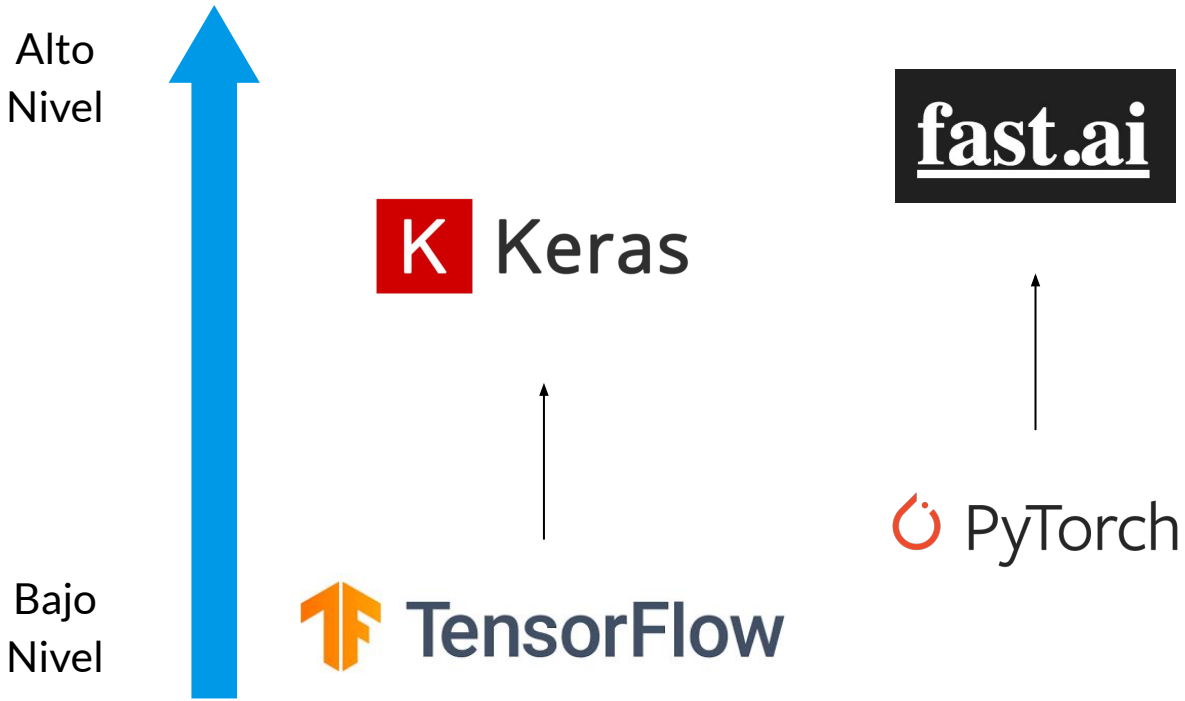
Presnetación
(20min) Redes neuronales en Python. Función de Costo.

Hands-on
(80 min) Actividad en grupos sobre Notebook.

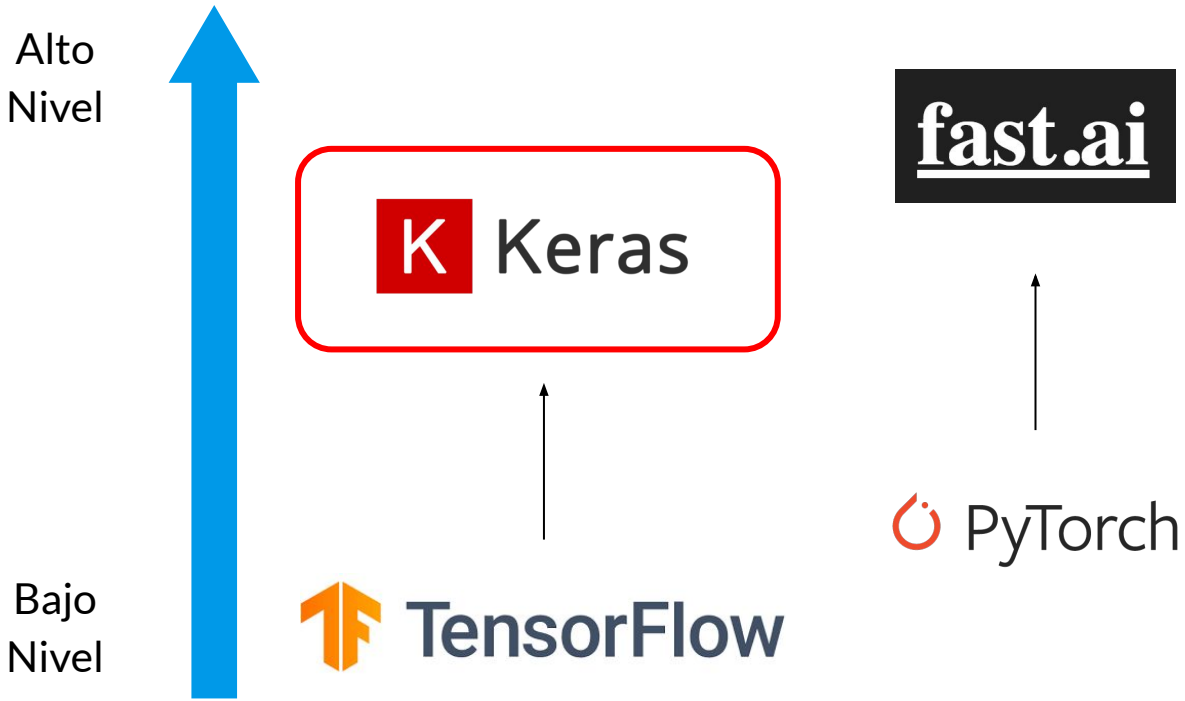
Puesta en común y Conclusiones
(15 minutos) Comentarios sobre el trabajo realizado.

Redes Neuronales en Python

Librerías para redes



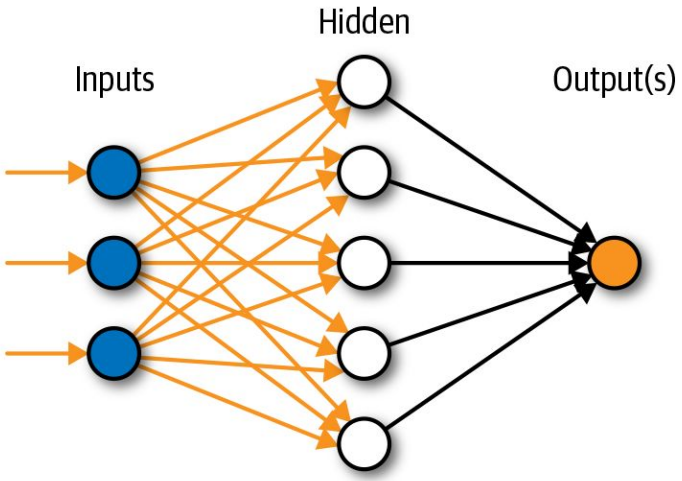
Librerías para redes



Librerías para redes:

Keras

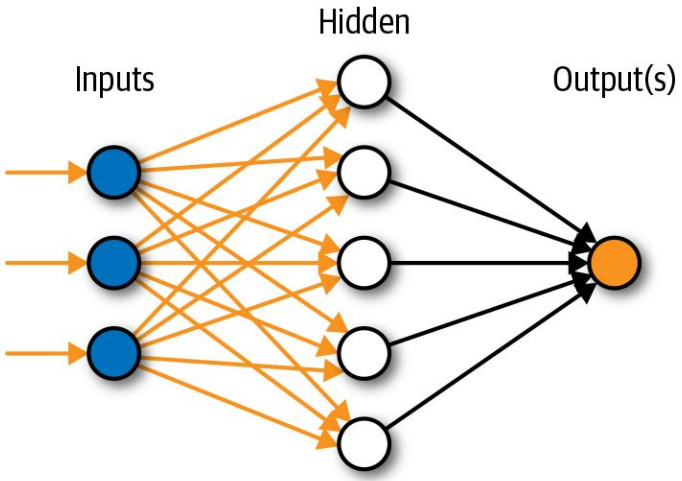
Pasos para entrenar una red neuronal como esta:



Librerías para redes:

Keras

Pasos para entrenar una red neuronal como esta:



1) Crear el Objeto

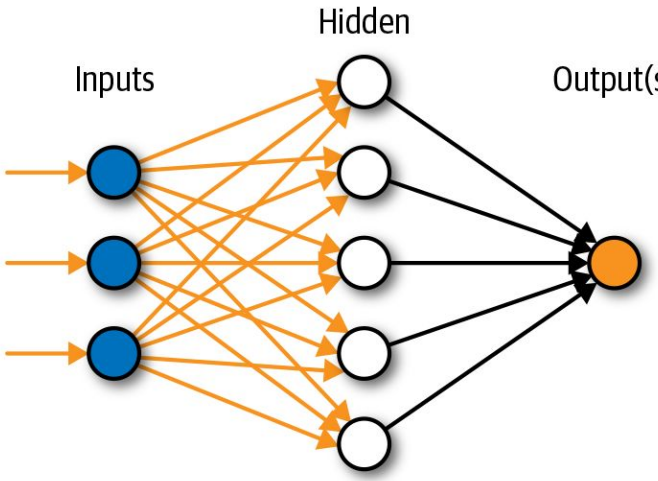
2) Definir la arquitectura (agregar capas)

3) Compilar

4) Entrenar

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

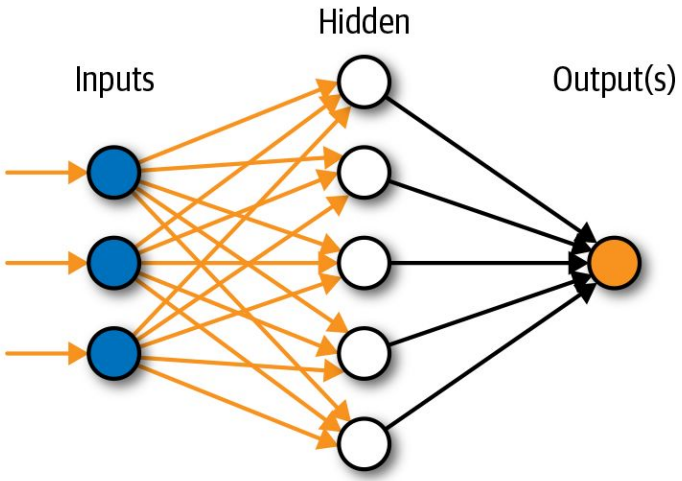
2) **Definir la arquitectura (agregar capas)**

3) **Compilar**

4) **Entrenar**

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

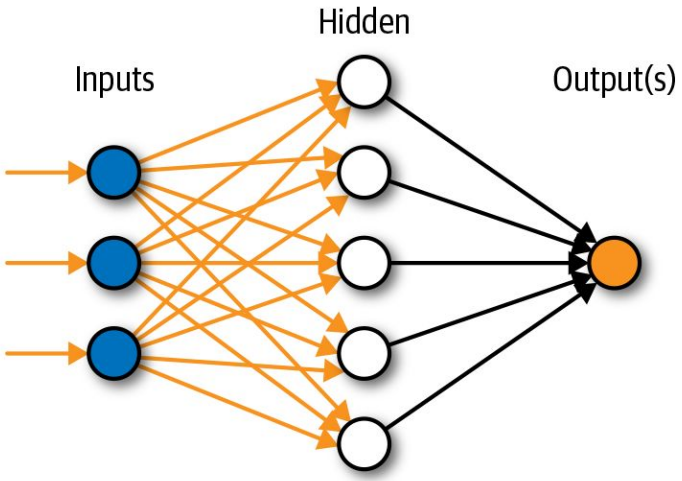
2)
```
model.add(Dense(5), activation = 'sigmoid')
model.add(Dense(1),activation = linear)
```

3) **Compilar**

4) **Entrenar**

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

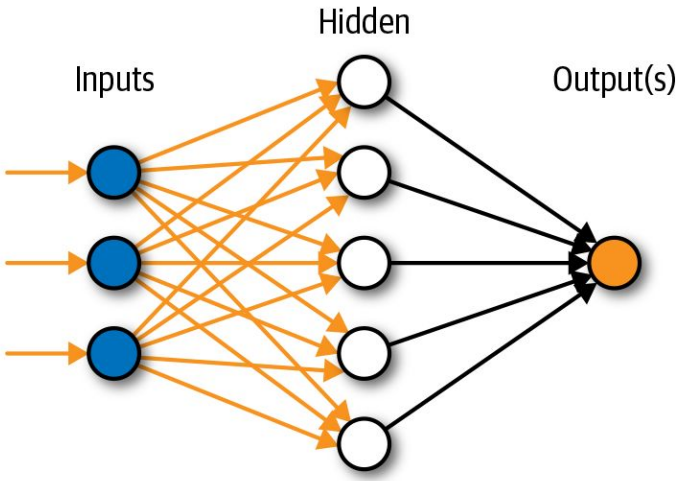
2)
```
model.add(Dense(5), activation = 'sigmoid')
model.add(Dense(1),activation = linear)
```

3)
```
model.compile(loss = 'mse', optimizer='adam')
```

4) Entrenar

Librerías para redes: Keras

Pasos para entrenar una red neuronal como esta:



1)
```
from keras.models import Sequential
model = Sequential()
```

2)
```
model.add(Dense(5), activation = 'sigmoid')
model.add(Dense(1),activation = linear)
```

3)
```
model.compile(loss = 'mse', optimizer='adam')
```

4)
```
model.fit(X,y,batch_size=32, epochs=100)
```

Función de Costo (Loss Function)

Función de Costo: Definición

Es aquello que buscamos optimizar:

- Costo de una instancia:

$$J_i(\hat{y}_i, y_i) = J_i(f_\Theta(x_i), y_i)$$

Función de Costo: Definición

Es aquello que buscamos optimizar.

- Costo de una instancia:

$$J_i(\hat{y}_i, y_i) = J_i(f_\Theta(x_i), y_i)$$

Predicción del modelo

Valor Real

Depende de los parámetros (pesos de la red)

Función de Costo: Definición

Es aquello que buscamos optimizar:

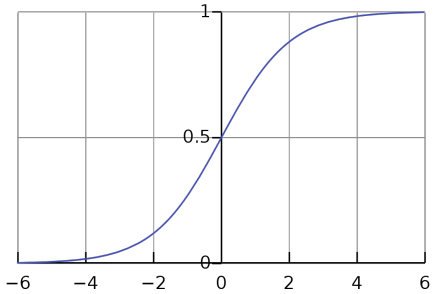
- Costo de una instancia:

$$J_i(\hat{y}_i, y_i) = J_i(f_\Theta(x_i), y_i)$$

- Problema a resolver (todas las instancias):

$$\arg\min_\Theta \; J(\Theta) = \frac{1}{N} \sum_i J_i(f_\Theta(x_i), y_i)$$

Función de Costo: Definición



Noten que los valores que pueda tomar van a estar determinados por la **ACTIVACIÓN de la última capa de neuronas.**

$$\arg\min_{\Theta} \; J(\Theta) = \frac{1}{N} \sum_i J_i\left(f_\Theta(x_i), y_i\right)$$

Función de Costo: Selección

¿Cómo elijo la función de costo J? Hay muchas disponibles.

Función de Costo: Selección

¿Cómo elijo la función de costo J? Hay muchas disponibles.

Clasificación

- Binary Cross Entropy
- Categorical Cross-entropy
- Poisson Loss
- Custom

Regresión

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- MAPE
- Custom

https://neptune.ai/blog/keras-loss-functions https://keras.io/api/losses/

Función de Costo: Selección

¿Cómo elijo la función de costo J? Hay muchas disponibles.

Vamos a ver 3 escenarios:

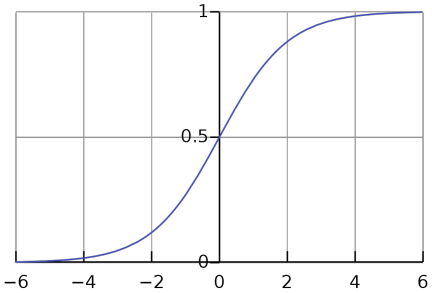
- **Escenario 1:** Clasificación Binaria
- **Escenario 2:** Clasificación Multiclase
- **Escenario 3:** Regresión

Escenario 1: Clasificación Binaria

Escenario 1: Clasificación Binaria

- **Activacion:** Sigmoide

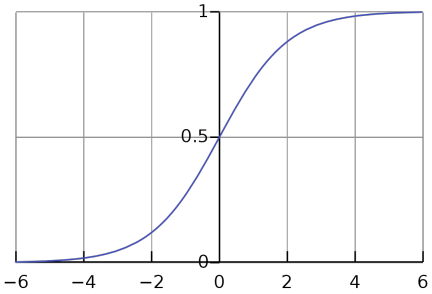
$$h(t) = \frac{1}{1+e^{-t}}$$



Escenario 1: Clasificación Binaria

- **Activacion:** Sigmoide

$$h(t) = \frac{1}{1+e^{-t}}$$



- **Función de costo:** Binary Cross-Entropy

$$J(\Theta) = \frac{1}{N}\sum_i -y_i \cdot log\left(f_\Theta(x_i)\right) - (1-y_i) \cdot log\left(1 - f_\Theta(x_i)\right)$$

Escenario 1: Clasificación Binaria

- **Activacion:** Sigmoide

```
model.add(layers.Dense(1, activation='sigmoid'))
```

- **Función de costo:** Binary Cross-Entropy

```
model.compile(optimizer="Adam",
              loss=tf.keras.losses.BinaryCrossentropy())
```

Escenario 2: Clasificación Multiclase

Escenario 2: Clasificación Multiclase

- **Activacion:** SoftMax

$$h(t)_j = \frac{e^{t_j}}{\sum_{k=1}^{K} e^{t_j}}$$

Generalización de la función logística (Normalizada)

Escenario 2: Clasificación Multiclase

- **Activacion:** SoftMax

$$h(t)_j = \frac{e^{t_j}}{\sum_{k=1}^{K} e^{t_j}}$$

Generalización de la función logística (Normalizada)

- **Función de costo:** Categorical Cross-Entropy

$$J(\Theta) = -\frac{1}{N} \sum_i \sum_k y_i \cdot log\left(f_\Theta(x_i)\right)$$

Generalización de la binaria (k clases)

Escenario 2: Clasificación Multiclase

- **Activacion:** SoftMax

```
model.add(layers.Dense(num_clases, activation='softmax'))
```

- **Función de costo:** Categorical Cross-Entropy

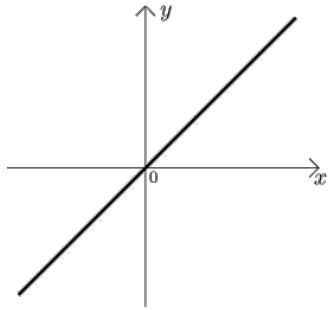
```
model.compile(optimizer="Adam",
              loss=tf.keras.losses.categorical_crossentropy())
```

Escenario 3: Regresión

Escenario 3: Regresión

- **Activacion:** Lineal

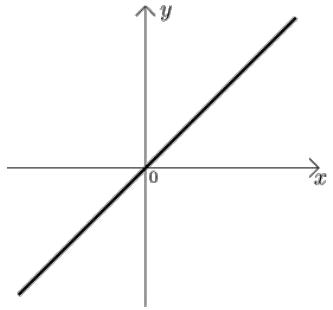
$$h(t) = t$$



Escenario 3: Regresión

- **Activacion:** Lineal

$$h(t) = t$$



- **Función de costo:** Mean Squared Error

$$J(\Theta) = \frac{1}{N} \sum_i (f_\Theta(x_i) - y_i)^2$$

Escenario 3: Regresión

- **Activacion:** Lineal

```
model.add(layers.Dense(1, activation='linear'))
```

- **Función de costo:** Mean Squared Error

```
model.compile(optimizer="Adam", loss='mse')
```

Métricas

Métricas

Son medidas de la performance de la red que queremos monitorear. (NO entrenamos para disminuir estas cantidades)

Métricas

Son medidas de la performance de la red que queremos monitorear. (NO entrenamos para disminuir estas cantidades)

Clasificación

- Accuracy
- F-Score (Precision and Recall)
- ROC Curve and AUC
- Custom

Regresión

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- MAPE
- Custom

Métricas

Son medidas de la performance de la red que queremos monitorear. (NO entrenamos para disminuir estas cantidades)

```
model.compile(loss='categorical_crossentropy',
              metrics=['categorical_accuracy'], optimizer='adam')
```

Si agregamos una métrica al compilar el modelo, se monitorea durante el entrenamiento.

Trabajo en el Notebook

Link

https://drive.google.com/file/d/1EIjJPm7Polq329zXvMrsgf1MJiLb4PFn/view?usp=sharing

Breakout Rooms

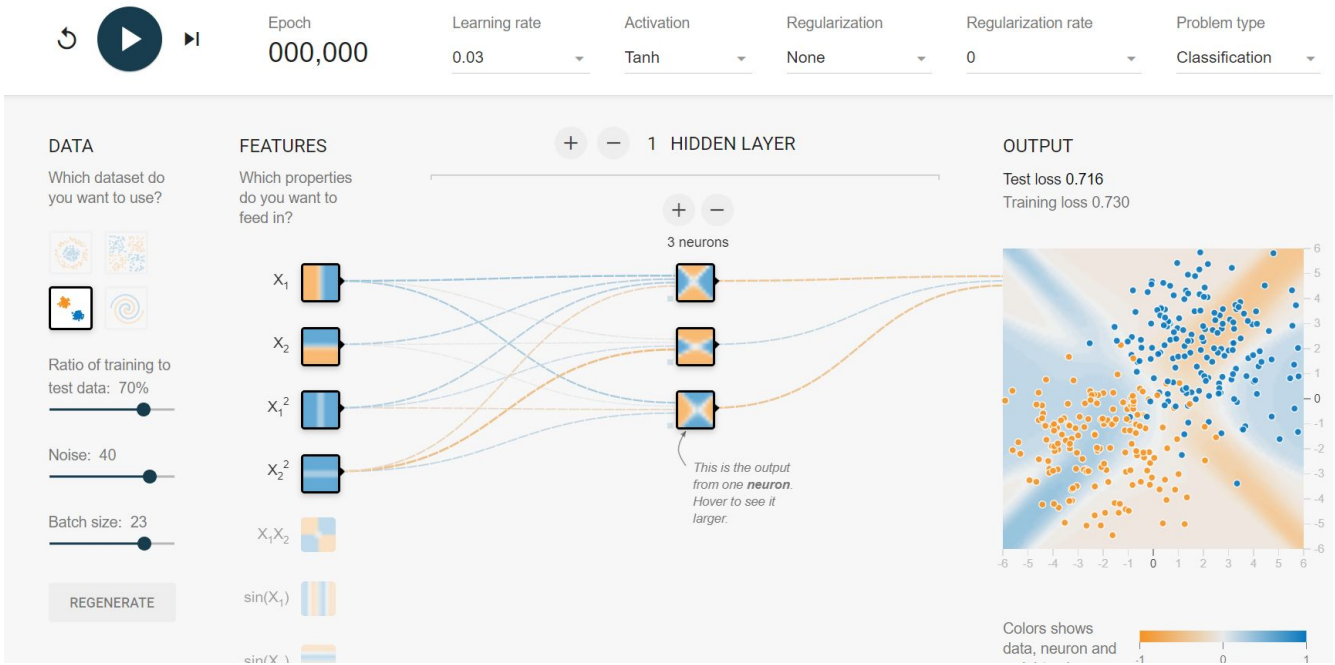
Trabajo en forma grupal, uno comparte pantalla. Pueden llamarme a la sala en cualquier momento.

Puesta en común y Conclusiones

(15 minutos) Comentarios sobre el trabajo realizado.

Comentario sobre Arquitecturas

Explorar arquitecturas



https://playground.tensorflow.org/

Aproximadores Universales

Podemos construir una función arbitraria!

Cybenko (1989)

Approximation by **superpositions** of a **sigmoidal function**
G Cybenko - Mathematics of control, signals and systems, 1989 - Springer
In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are …
☆ 99 Citado por 13458 Artículos relacionados Las 18 versiones

Hornik (1991)

Approximation capabilities of multilayer feedforward networks
K Hornik - Neural networks, 1991 - Elsevier
We show that standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators with respect to L p (μ) performance criteria, for arbitrary finite input environment measures μ, provided only that sufficiently many hidden units are available. If the activation function is continuous, bounded and nonconstant, then continuous mappings can be learned uniformly over compact input sets. We also give very general conditions ensuring that networks with …
☆ 99 Citado por 4868 Artículos relacionados Las 13 versiones