

Taller de Aprendizaje Automático

Implementación de redes neuronales

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

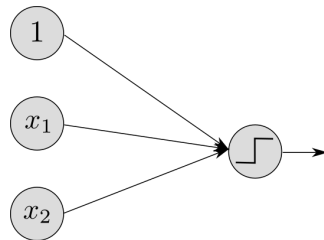
- ① Perceptrón
- ② Perceptrón multicapa
- ③ Propagación hacia atrás
- ④ Funciones de activación
- ⑤ Aproximación versus generalización
- ⑥ Regularización y validación
- ⑦ Implementación en Keras y TensorFlow

Perceptrón

$$h(\mathbf{x}) = \text{sign} \left(\left(\sum_{i=1}^d w_i x_i \right) + w_0 \right)$$

$$h(\mathbf{x}) = \text{sign} \left(\sum_{i=0}^d w_i x_i \right)$$

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$$

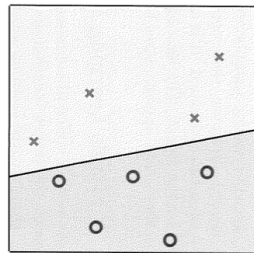


Algoritmo de aprendizaje:

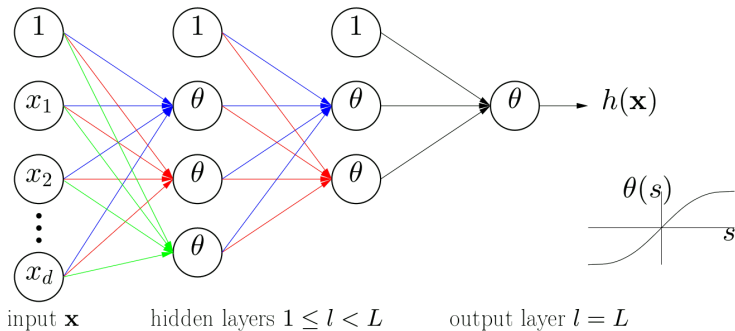
$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$$

$$\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n$$

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$



Perceptrón multicapa



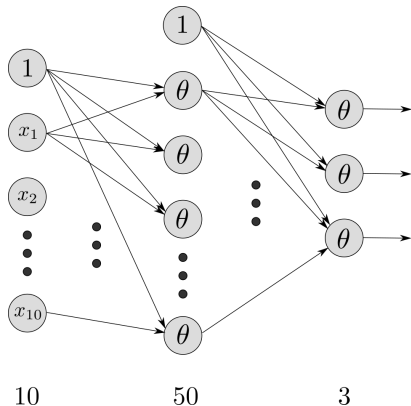
$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta \left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)} \right)$$

$$x_1^{(0)} \dots x_{d^{(0)}}^{(0)} \rightarrow \dots \rightarrow x_1^L = h(\mathbf{x})$$

$$w_{ij}^{(l)} = \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

Ejercicio

Considere un perceptrón multicapa con 10 entradas, seguido de una capa oculta con 50 neuronas y una capa de salida con 3 neuronas. Sea m el tamaño del lote.



- 1 ¿Cuál es la dimensión de la matriz $\mathbf{X}$?
- 2 ¿Cuál es la dimensión de $\mathbf{W}_h$ y $\mathbf{b}_h$?
- 3 ¿Cuál es la dimensión de $\mathbf{W}_o$ y $\mathbf{b}_o$?
- 4 ¿Cuál es la dimensión de la matriz $\mathbf{Y}$?
- 5 Escriba la ecuación para calcular la salida.

$$\mathbf{X} : (m \times 10), \mathbf{W}_h : (10 \times 50), \mathbf{b}_h : (50)$$

$$\mathbf{W}_o : (50 \times 3), \mathbf{b}_o : (3), \mathbf{Y} : (m \times 3)$$

$$\mathbf{Y}^* = \theta(\theta(\mathbf{X}\mathbf{W}_h + \mathbf{b}_h)\mathbf{W}_o + \mathbf{b}_o)$$

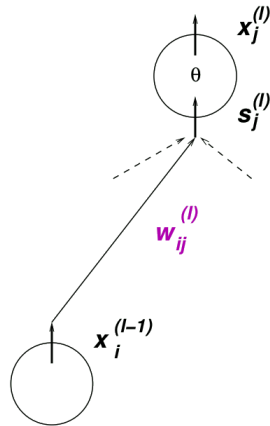
Propagación hacia atrás

Backpropagation*: gradiente descendente eficiente

$$\nabla e(\mathbf{w}) : \frac{\partial e(\mathbf{w})}{\partial w_{ij}^{(l)}} \quad \forall i, j, l \quad \frac{\partial e(\mathbf{w})}{\partial w_{ij}^{(l)}} = \frac{\partial e(\mathbf{w})}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$$

$$\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = x_i^{(l-1)} \quad \frac{\partial e(\mathbf{w})}{\partial s_j^{(l)}} = \delta_j^{(l)}$$

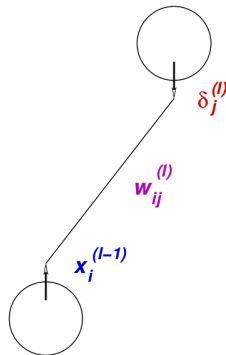
se calcula $\delta_1^{(L)}$ (última capa) y se propaga hacia atrás



* D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986

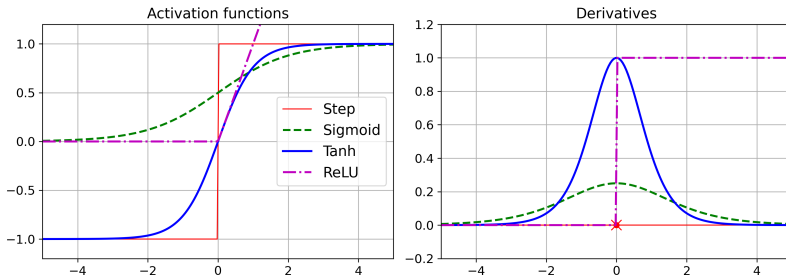
Propagación hacia atrás

- 1: Initialize all weights $w_{ij}^{(l)}$ **at random**
- 2: **for** $t = 0, 1, 2, \dots$ **do**
- 3: Pick $n \in \{1, 2, \dots, N\}$
- 4: *Forward:* Compute all $x_j^{(l)}$
- 5: *Backward:* Compute all $\delta_j^{(l)}$
- 6: Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)} \delta_j^{(l)}$
- 7: Iterate to the next step until it is time to stop
- 8: Return the final weights $w_{ij}^{(l)}$



Funciones de activación

- Sigmoid : $\theta(x) = \frac{1}{1+e^{-s}}$
- Tanh : $\theta(x) = \tanh(x) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$
- ReLU : $\theta(x) = \max(0, x)$



Pregunta

¿Para qué necesitamos funciones de activación? ¿Sería suficiente con funciones lineales?

Aproximación versus generalización

Teorema de aproximación universal

Una red neuronal prealimentada (feed-forward) con una única capa oculta y un número finito de neuronas, puede aproximar cualquier función continua en un espacio compacto de $\mathbb{R}^n$.^{†}*

- el ancho de la red debe ser exponencialmente grande
- aproximación universal para redes profundas de ancho acotado [‡]

Generalización

Las redes neuronales tienen tendencia al **sobreajuste**.

- resultados teóricos sobre error de generalización (d_{VC})
- **validación** y **regularización** para prevenir sobreajuste

^{*}G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989

[†]K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991

[‡]Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in NIPS*, vol. 30, 2017

Regularización y validación

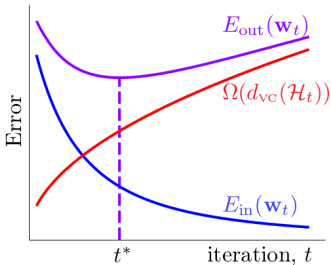
- Ridge (weight decay):

Error con penalización sobre la complejidad del modelo.

$$E_{aug}(\mathbf{w}) = E_{in}(\mathbf{w}) + \frac{\lambda}{N} \sum_{i,j,l} (w_{ij}^{(l)})^2$$

- Early stopping:

Se detiene el entrenamiento antes de sobreajustar, evaluando en validación.



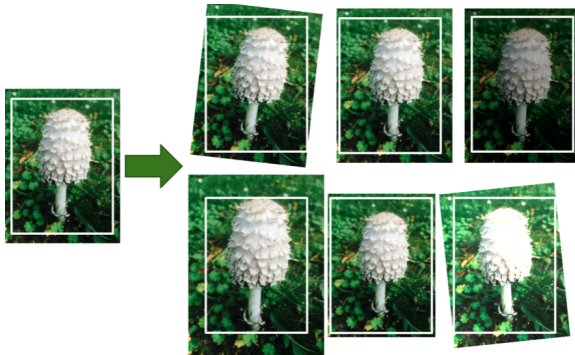
Regularización y validación

- Dropout

Ignorar aleatoriamente algunas neuronas (10 a 50%) en cada paso de entrenamiento. Se obtiene una red menos sensible a cambios en la entrada y menos sobreajustada.

- Data augmentation

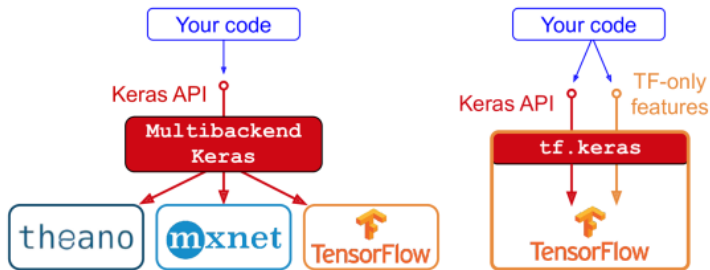
Aumentar artificialmente los datos de entrenamiento mediante transformaciones (realistas).



Keras y TensorFlow

Keras API de alto nivel para aprendizaje profundo

TensorFlow biblioteca para aprendizaje profundo



```
>>> import tensorflow as tf
>>> from tensorflow import keras
```

Clasificación de imágenes

```
fashion_mnist = keras.datasets.fashion_mnist  
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

```
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.  
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]  
X_test = X_test / 255.
```



Definición del modelo

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 300)	235500
dense_1 (Dense)	(None, 100)	30100
dense_2 (Dense)	(None, 10)	1010

Total params: 266,610

softmax

regresión logística multiclase

$$\sigma(\mathbf{s}(\mathbf{x}))_k = \frac{e^{s_k(\mathbf{x})}}{\sum_{j=1}^K e^{s_j(\mathbf{x})}}$$

- convierte K valores reales a que sumen 1
- para clases excluyentes

$$\hat{y} = \underset{k}{\operatorname{argmax}} \sigma(\mathbf{s}(\mathbf{x}))_k$$

Inicialización de pesos

```
hidden1 = model.layers[1]  
weights, biases = hidden1.get_weights()
```

```
>>> weights  
array([[ 0.02448617, -0.00877795, -0.02189048, ..., -0.02766046,  
        0.03859074, -0.06889391],  
       ...,  
       [-0.06022581,  0.01577859, -0.02585464, ..., -0.00527829,  
        0.00272203, -0.06793761]], dtype=float32)
```

```
>>> weights.shape  
(784, 300)
```

```
>>> biases  
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
       ...,  
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
biases.shape  
(300,)
```

Compilación del modelo

```
model.compile(loss="sparse_categorical_crossentropy",  
              optimizer="sgd",  
              metrics=["accuracy"])
```

cross-entropy loss

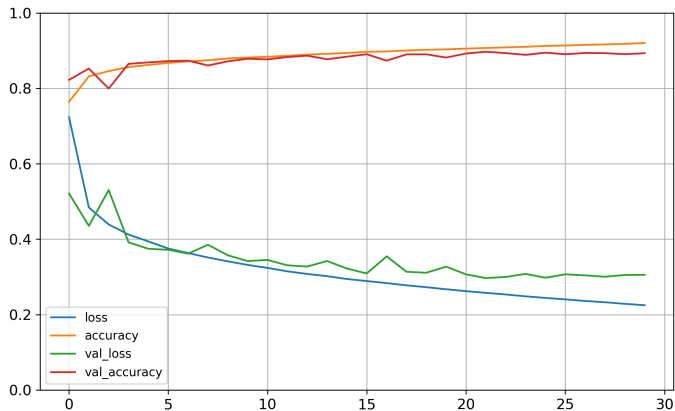
- binary_crossentropy
- categorical_crossentropy
- sparse_categorical_crossentropy

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N [y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)]$$

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^N \sum_{k=1}^K y_n^k \log \hat{y}_n^k$$

Entrenamiento del modelo

```
history = model.fit(X_train, y_train, epochs=30,  
                    validation_data=(X_valid, y_valid))
```



Evaluación del modelo

```
>>> model.evaluate(X_test, y_test)
313/313 [=====] - 1s 2ms/step - loss: 0.3382 - accuracy: 0.8822
[0.3381877839565277, 0.8822000026702881]
```

```
>>> y_pred = np.argmax(model.predict(X_new), axis=-1)
>>> y_pred
array([9, 2, 1])
```

Ankle boot



Pullover



Trouser



Guardando y cargando el modelo

```
model = keras.models.Sequential([...])  
model.compile(...)  
history = model.fit(..)
```

```
model.save("my_keras_model.h5")
```

```
model = keras.models.load_model("my_keras_model.h5")  
model.predict(X_new)
```

Ejercicios

Estudie la implementación de un MLP aplicado al problema de `california_housing`.

Pregunta de salida

¿Cuáles son los hiperparámetros de un perceptrón multicapa? ¿Cómo elegiría sus valores?

Referencias



D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.



G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.



K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.



Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in NIPS*, vol. 30, 2017.



A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*. O'Reilly Media, Inc., 2022.