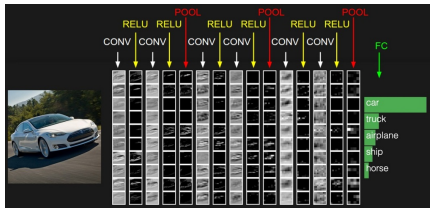
Tutorial

Deep Learning con Tensorflow



Andrés Ortiz García
E.T.S. Ingeniería de Telecomunicación
aortiz@ic.uma.es

Biomedical Signal Processing and Bioinspired Systems

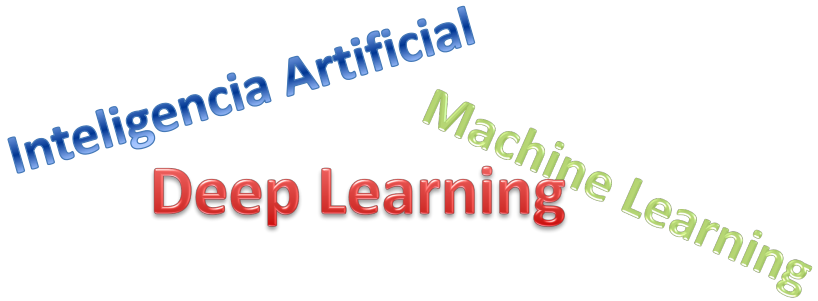
Contenidos

1.Introducción a Deep Learning
- 1.1. Inteligencia Artificial, Machine Learning y Deep Learning
- 1.2. Algunos fundamentos de Machine Learning.
- 1.3. Redes neuronales y Deep Learning.

2.Introducción a TensorFlow
- 2.1. ¿ Qué es TensorFlow ?.
- 2.2. Tensorflow y Deep Learning.
- Práctica 1. Primeros pasos con TensorFlow: Regresión lineal.
- Práctica 2. Perceptrón multicapa

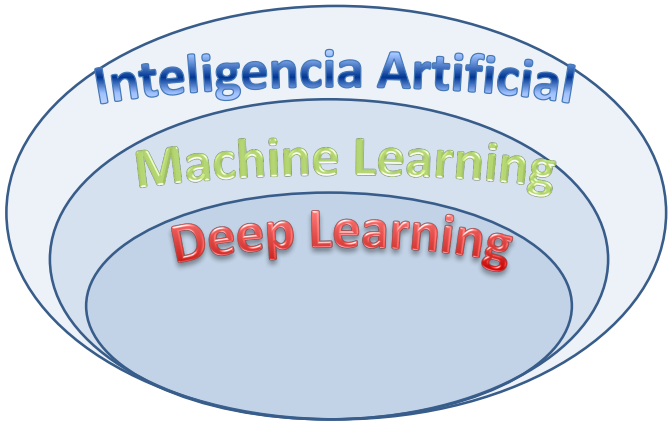
3. Redes Neuronales Convolucionales (CNN)
- 3.1. Introducción a las redes Convolucionales
- Práctica 3. Red Neuronal Convolucional

1.Introducción a Deep Learning



Conceptos parecidos, pero no equivalentes ni intercambiables

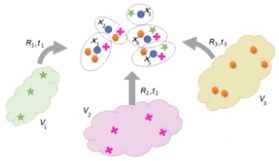
1.Introducción a Deep Learning



1.2. Algunos fundamentos de Machine Learning

Objetivo: modelar datos { Predicción / Generación } Generalización

Machine Learning



Materia prima

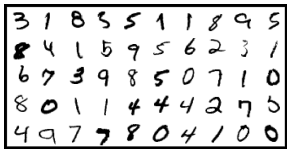
Proceso de aprendizaje

Producto final

Problemas sencillos para los humanos pero muy complejos para un computador

?

?



3 ? 5 ?

Las reglas para diferenciar clases no son triviales

↓

Los modelos no pueden ser sólo estadísticos

↓

Inteligencia Artificial

Deep learning: una idea de los años 40

Cibernética
McCulloch and Pitts, 1943; Hebb, 1949

↓

Perceptrón
Rosenblatt, 1958

↓

Algoritmo back-propagation
Rumelhart, 1958

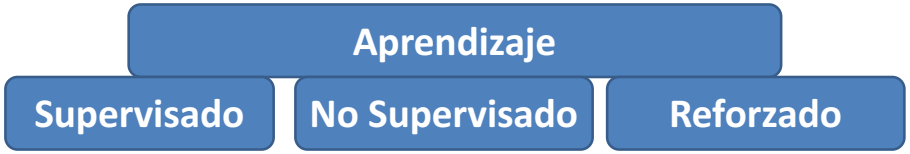
↓

Deep Learning
Hinton et al., 2006; Bengio et al. 2007; Ranzato et al. 2007

- La idea de Deep Learning ha sido rebautizada a lo largo del tiempo , por la influencia de diferentes investigadores.
- En 1995, se abandonaron las técnicas basadas en ANN a favor de las técnicas de aprendizaje estadístico (SVM)
- En la actualidad ha ganado popularidad por los nuevos algoritmos junto con la capacidad de cómputo de los procesadores actuales (uso de clusters y GPUs)

Aprendizaje y capacidad de generalización

Tipos de aprendizaje



Necesaria la solución al problema para reajustar el modelo

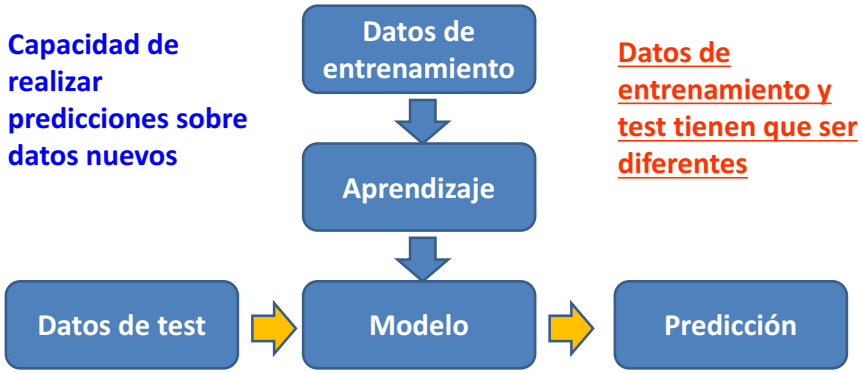
No es necesaria la solución al problema para reajustar el modelo. Se utilizan por ejemplo, medidas de similitud para separar clases

Utiliza algunas soluciones y el grado de bondad de las mismas para determinar "lo buena" que es una solución (recompensa)

Aprendizaje y capacidad de generalización

Generalización. Overfitting, Underfitting.

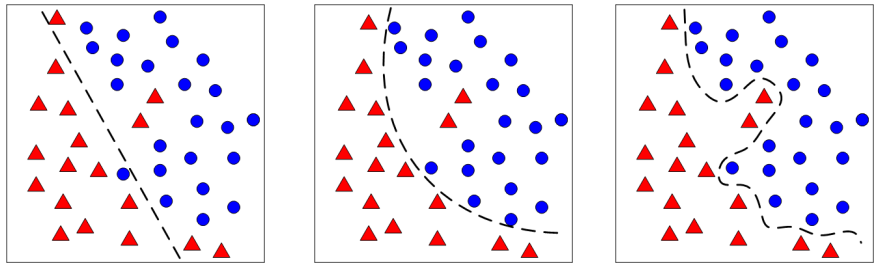
Capacidad de realizar predicciones sobre datos nuevos



Datos de entrenamiento y test tienen que ser diferentes

Aprendizaje y capacidad de generalización

Generalización. Overfitting, Underfitting.



Underfitting **OK** **Overfitting**

Aprendizaje y capacidad de generalización

- Un modelo subentrenado (underfitting) o sobreentrenado (overfitting) generaliza mal.

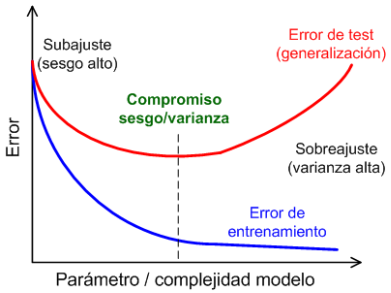
- Solución para el underfitting: normalmente hay que plantear modelos más complejos (ej. pasar de modelos lineales a no lineales, incrementar la dimensión del espacio de características…)
- Solución para el overfitting: incrementar el número de muestras o simplificar el modelo.

Evaluación

Curva de aprendizaje

Curva de aprendizaje

- La validación cruzada permite estimar las prestaciones de un modelo predictivo (error de generalización)

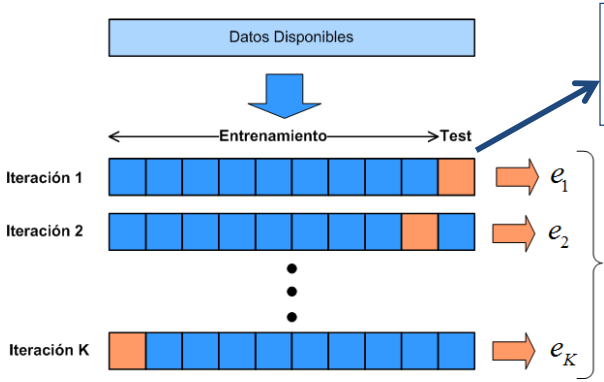


Mejora de la capacidad de generalización

- **Sesgo alto.**
  - aumentar el número de muestras de entrenamiento
- **Varianza alta:**
  - Parada del algoritmo de entrenamiento
  - Regularización

Estimación del error de generalización

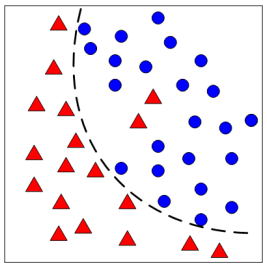
- Validación cruzada



Es conveniente conservar la proporción de etiquetas en los subconjuntos (estratificación)

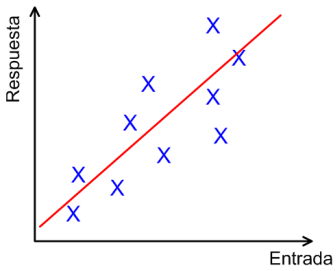
Clasificación y regresión (Ap. Supervisado)

- Clasificación: La predicción es categórica (pertenencia a una clase). Los datos de entrenamiento tienen la forma {muestra, clase} , clase ∈ {▲, ●}



Clasificación y regresión (Ap. Supervisado)

- Regresión: La predicción es continua. Los datos de entrenamiento tienen la forma {muestra, respuesta}, respuesta ∈ °



Evaluación de la capacidad predictiva

Matriz de Confusión y medidas de prestaciones (2 clases)

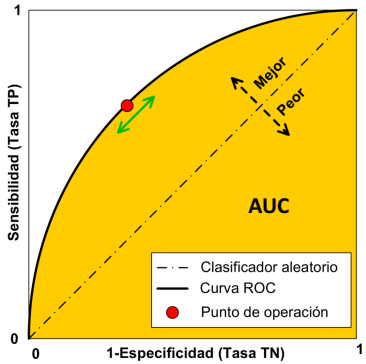
Clase Estimada \ Clase Real	POSITIVO	NEGATIVO
POSITIVO	Verdadero Positivo (TP)	Falso Positivo (FP) **Error Tipo I**
NEGATIVO	Falso Negativo (FN) **Error Tipo II**	Verdadero Negativo (TN)

$$Precisión = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Especificidad = \frac{TN}{TN + FP} \qquad Sensibilidad = \frac{TP}{TP + FN}$$

Evaluación de la capacidad predictiva

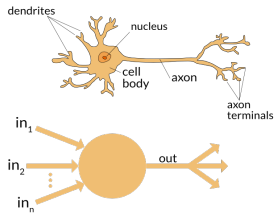
Receiver Operating Characteristic (Curva ROC)

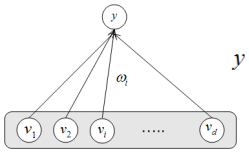


- Mide el solapamiento entre las distribuciones de un predictor continuo para los positivos y negativos
- **Área bajo la curva (AUC):** probabilidad de que una muestra positiva aleatoria obtenga un valor más extremo del predictor que una muestra negativa aleatoria
- El punto de operación es el compromiso entre especificidad y sensibilidad (mejor predicción de los verdaderos positivos y los verdaderos negativos simultáneamente)

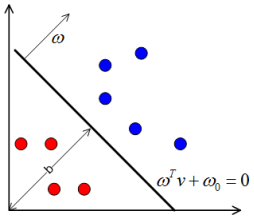
Redes neuronales artificiales

Modelo neurona (McCulloch-Pitts,1943) **Perceptrón simple (Rossenblat, 1957)**





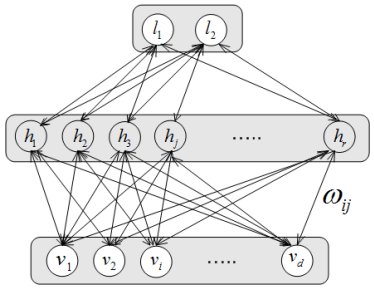
$$y = \omega_0 + \sum_{i=1}^{n} \omega_i v_i$$



- **Sólo separa clases linealmente separables**
- **No permite aprender representaciones abstractas (*representative learning*)**

Redes neuronales artificiales

Perceptron multicapa



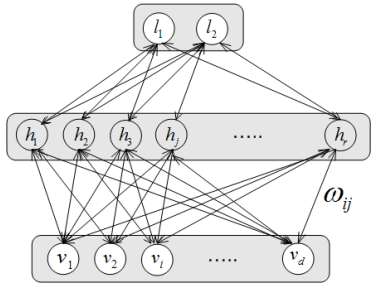
- Separa clases no linealmente separables
- La interconexión entre neuronas permite reconocer patrones complejos
- Algoritmo de aprendizaje *backpropagation (regla delta)*

$$\Delta\omega_{jk}^{(n)} = \frac{-\eta \partial E(\omega_{jk}^{(n)})}{\partial \omega_{jk}^{(n)}}$$

$$\omega_{jk}^{(n+1)} \leftarrow \omega_{jk}^{(n)} + \Delta\omega_{jk}^{(n)}$$

Redes neuronales artificiales

Perceptron multicapa



- Separa clases no linealmente separables
- La interconexión entre neuronas permite reconocer patrones complejos
- Algoritmo de aprendizaje *backpropagation (regla delta)*

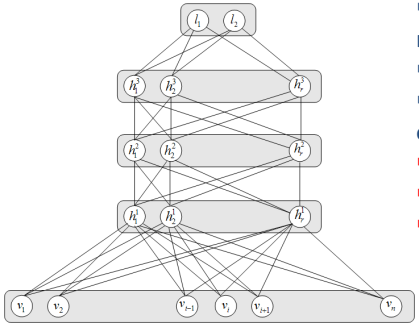
$$\Delta\omega_{jk}^{(n)} = \frac{-\eta\partial E(\omega_{jk}^{(n)})}{\partial\omega_{jk}^{(n)}}$$

Lo hace TensorFlow de forma eficiente!

$$\omega_{jk}^{(n+1)} \leftarrow \omega_{jk}^{(n)} + \Delta\omega_{jk}^{(n)}$$

Redes neuronales artificiales

Arquitecturas profundas



- Es posible modelar problemas complejos
- Permite obtener características en diferentes niveles de abstracción
- El model es una caja negra
- Abandonado a favor de clasificadores estadísticos (ej. SVM) debido a:
  - Difícil de entrenar con *backpropagation*
  - Propenso a *Overfitting* → Generalización
  - **Computacionalmente muy costoso**

2. Introducción a Tensorflow

¿ Qué es tensorflow ?

- Librería para cálculo numérico y optimización, diseñada para ser eficiente con problemas complejos.
- Trabaja con tensores (arrays multidimensionales).
- Usa de forma transparente múltiples CPUs y GPUs
- Altamente escalable: múltiples CPUs, múltiples GPUs y múltiples nodos.
- **Proporciona el soporte necesario para construir arquitecturas neuronales profundas** y para programar algoritmos de machine learning en general.

2. Introducción a Tensorflow

¿ Qué es tensorflow ?

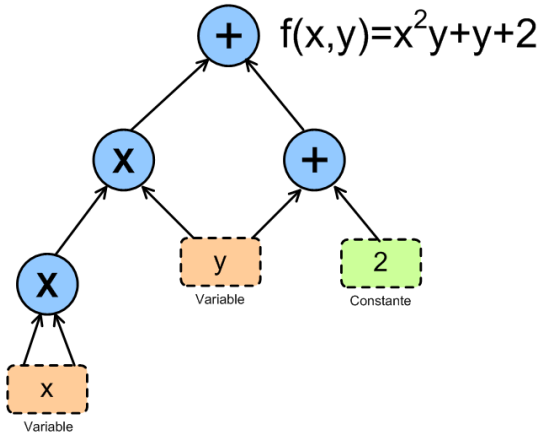
- APIs para C++ y Python
- **El API de Python cuenta con mejor soporte** y existen múltiples webs, foros, etc. con mucha información.
- APIs de alto nivel sobre TF: Keras, Pretty Tensor.
- Versiones para Windows, Linux, macOs, iOS y Android!

https://www.tensorflow.org/

Modelo de programación. Grafos de flujo de datos

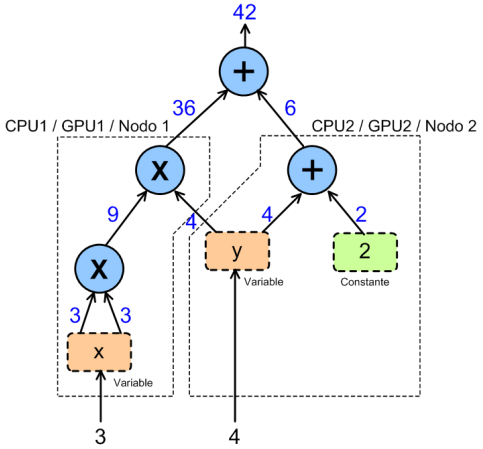
- Un modelo de cálculo en TensorFlow se define como un *grafo*, donde:
  - Nodos: representan una instancia de una operación matemática
  - Enlaces (edges) :
    - a) Tensor sobre el cual se ejecuta una operación.
    - b) Relación de dependencia entre dos nodos (ej. la operación del nodo A no puede realizarse hasta que no termine la operación del nodo B)

Ejemplo: implementación de $f(x,y)=x^2y+y+2$ *



*Geron, A.:"Hands-on machine learning with Scikit-Learn and Tensorflow". O'Reilly Media, 2017

Paralelización f(3,4)=42



*Geron, A.:"Hands-on machine learning with Scikit-Learn and Tensorflow". O'Reilly Media, 2017

Modelo de programación. Sesiones

- Un grafo describe las operaciones a realizar, pero no realiza los cálculos.
- Una sesión permite ejecutar un grafo o parte de un grafo: reserva los recursos necesarios (en una o varias CPUs, GPUs o nodos) y ejecuta los cálculos definidos en el grafo, almacenando cuando sea necesario, resultados intermedios.
  - **Ej.** Hasta que no se llama al método **session.run**, no se ejecutan los cálculos definidos en el grafo.

Modelo de programación. Sesiones

- **Dos formas de definir y ejecutar una sesión**

1) sess=tf.session**()**
   sess.run(variable, placeholder, etc)
   sess.close()

2) sess=tf.Interactive.Session()
   init.run()
   variable(placeholder).eval()
   sess.close()

Modelo de programación. Variables y Constantes

- **Dos formas de definir variables**

1) **a=tf.Variable(tf.zeros((2,2)),name="var_a")**
   - Puede ser un tensor de cualquier tamaño y tipo.
   - Almacenamiento del estado del grafo. Requiere ser inicializado con un valor inicial.
   - Se definen en TensorFlow (no pueden usarse para cargar datos externos, ej. arrays numpy)
   - **Mantiene su valor entre llamadas a session.run()**
   - **Ejemplo de uso: definición de pesos en una red neuronal**

Modelo de programación. Variables y Constantes

- **Dos formas de definir variables**

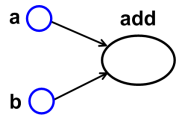
2) **a=tf.Placeholder(tf.float32)**
   - Se utilizan para cargar datos externos en el modelo de computación de TensorFlow (grafo).
   - **Se inicializan cada vez que ejecutamos la sesión.**
   - **Ejemplo de uso: carga de datos de entrenamiento (los datos de entrenamiento no se almacenan en el grafo).**

Modelo de programación. Ejemplo

- **Con Variables:**

```
a=tf.Variable(3,name="a")
b=tf.Variable(4,name="a")
c=tf.add(a,b)
```

Define el grafo y la operación, y **carga** valores concretos



- **Con Placeholders**

```
a=tf.Placeholder(tf.int32)
b=tf.Placeholder(tf.int32)
c=tf.add(a,b)
```

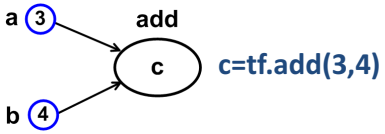
Define el grafo y la operación, **sin cargar** valores concretos

Modelo de programación. Ejemplo

- **Con Variables:**

```
a=tf.Variable(3,name="a")
b=tf.Variable(4,name="a")
c=tf.add(a,b)
```

Define el grafo y la operación, y **carga** valores concretos



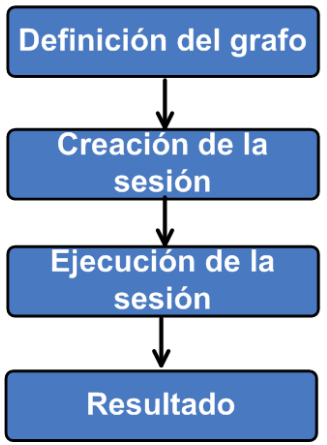
c=tf.add(3,4)

- **Con Placeholders**

```
a=tf.placeholder(tf.int32)
b=tf.placeholder(tf.int32)
c=tf.add(a,b)
```

Define el grafo y la operación, **sin cargar** valores concretos

Modelo de programación. Resumen




```
a=tf.Variable(3,name="a")
b=tf.Variable(4,name="b")
c=a*b
Init=tf.global_variables_initializer()

sess=tf.interactive_session()
sess=tf.Session()
Init.run()

result=c.eval()
result=sess.run(c)

print(result)
```


Ejemplos de equivalencia numpy - Tensorflow

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

https://becominghuman.ai/an-introduction-to-tensorflow-f4f31e3ea1c0

Ejercicio 1. Implementar la operación c=a*b con variables y con placeholders

Ejercicio 1. Modelo de programación. Resumen

Con variables

```
Import tensorflow as tf
a=tf.Variable(2,name="a")
b=tf.Variable(4,name="b")
c=a*b
init=tf.global_variables_
initializer()
sess=tf.Session()
init.run()
result=sess.run(c)
sess=tf.interactive_session()
init.run()
result=c.eval()

print(result)
```

Con placeholders

```
Import tensorflow as tf
a=tf.placeholder(tf.int32)
b=tf.placeholder(tf.int32)
c=a*b
*init=tf.global_variables_initializer()
sess=tf.Session()
*init.run()
result=sess.run(c,feed_dict={a:2,b:4})
sess=tf.interactive_session()
* init.run()
result=c.eval(feed_dict{a:2,b:4})

print(result)

* No es necesario con placeholders!
```


Cálculo de gradientes

- Una de los cálculos más importantes que realiza TensorFlow es el cálculo de derivadas.
- El cálculo de gradientes es esencial en redes neuronales y en Deep Learning!
- Ejemplo: Calculo de la derivada de y con respecto a x

```
Import tensorflow as tf
x=tf.placeholder(tf.float32)
y=2*x*x
grad=tf.gradients(y,x)
sess=tf.Session()
grad_val=session.run(grad,feed_dict={x:1})
print(grad_val)
```


Funciones de costo (o pérdida)

- Son funciones que mapean un valor multidimensional en un valor real relacionado con la bondad de un determinado evento.
- Uso para estimación de parámetros, realimentación en algoritmos de aprendizaje supervisado, wrappers…
- Ejemplo: En regresión lineal,

$$loss(x) = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$$

- Ejemplo (2): Entropía cruzada (clasificación)

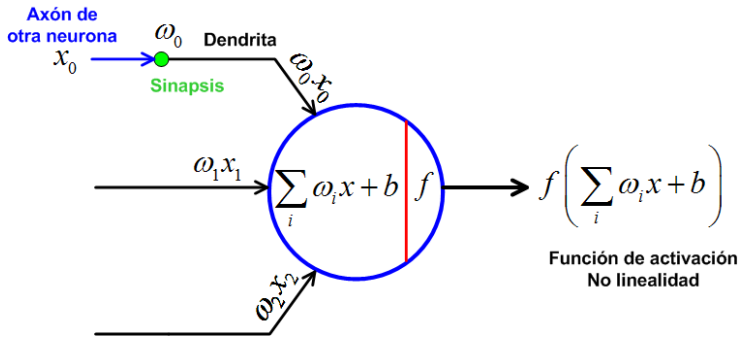
$$H(y,\hat{y}) = -\sum_i y_i \log(\hat{y})$$

Optimizadores

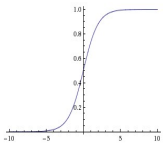
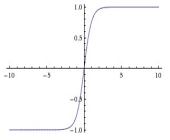
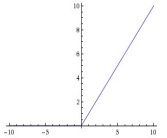
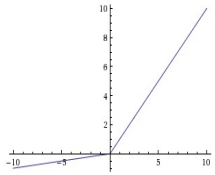
- TensorFlow dispone de un conjunto de optimizadores que pueden utilizarse para minimizar o maximizar una determinada función (ej. la función de costo o pérdida)
- Dos ejemplos representativos:
  1) Gradiente Descendente
  2) AdamOptimizer → Algoritmo *Adam* de Kingma y Ba's

AdamOptimizer presenta ventajas con respecto al algoritmo clásico de gradiente descendente. Por ejemplo, usa un control adaptativo de la tasa de aprendizaje, evitando mínimos locales y acelerando la convergencia.

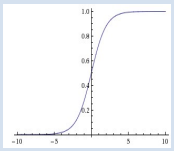
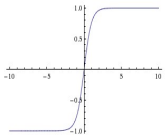
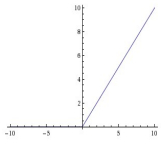
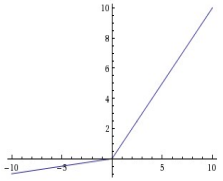
Funciones de activación



Funciones de activación

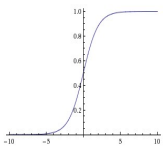
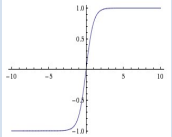
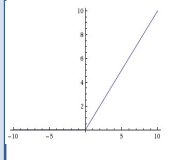
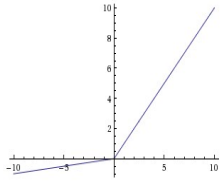
Sigmoide	tanh	ReLU	Leaky ReLU
			
$\frac{1}{1+e^{-x}}$	$tanh(x)$	$max(0,x)$	$max(Cx,x)$

Funciones de activación

Sigmoide	tanh	ReLU	Leaky ReLU
			
$\frac{1}{1+e^{-x}}$	$tanh(x)$	$max(0,x)$	$max(Cx,x)$

- Apropiada para entradas binarias
- Fuerza la activación al intervalo [0,1] → Satura Gradientes
- En tensorflow: tf.nn.sigmoid

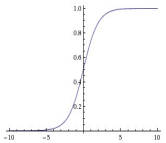
Funciones de activación

Sigmoide	tanh	ReLU	Leaky ReLU
			
$\frac{1}{1+e^{-x}}$	$tanh(x)$	$max(0,x)$	$max(Cx,x)$

- No tan propiada para entradas binarias como la sigmoidal
- Fuerza la activación al intervalo [0,1] → Satura Gradientes!
- En tensorflow: tf.nn.tanh

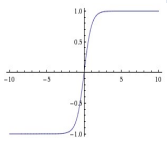
Funciones de activación

Sigmoide



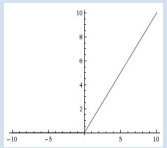
$$\frac{1}{1+e^{-x}}$$

tanh



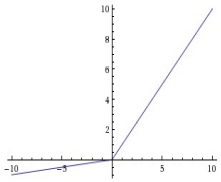
$$tanh(x)$$

ReLU



$$max(0,x)$$

Leaky ReLU

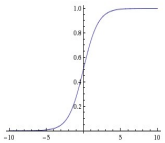


$$max(Cx,x)$$

- Apropiada para entradas cotinuas (ej. Imágenes ☺)
- No se satura en la región x>0
- La red converge más rápido que usando sigmoide o tanh.
- En Tensorflow tf.nn.relu

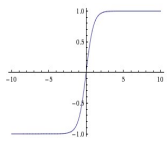
Funciones de activación

Sigmoide



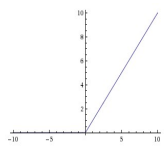
$$\frac{1}{1+e^{-x}}$$

tanh



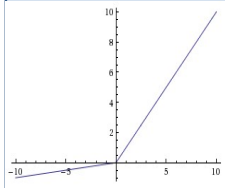
$$tanh(x)$$

ReLU



$$max(0,x)$$

Leaky ReLU

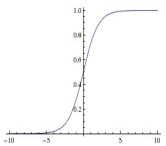


$$max(Cx,x)$$

- Apropiada para entradas cotinuas (ej. Imágenes ☺)
- No se satura
- La red converge más rápido que usando sigmoide o tanh.

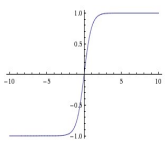
Funciones de activación

Sigmoide



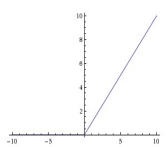
$$\frac{1}{1+e^{-x}}$$

tanh



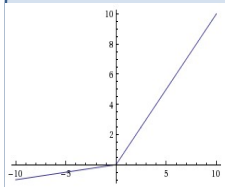
$$tanh(x)$$

ReLU



$$max(0,x)$$

Leaky ReLU



$$max(Cx,x)$$

- En Tensorflow

```
def lrelu(x, alpha):
    return tf.maximum(x, alpha * x)
```

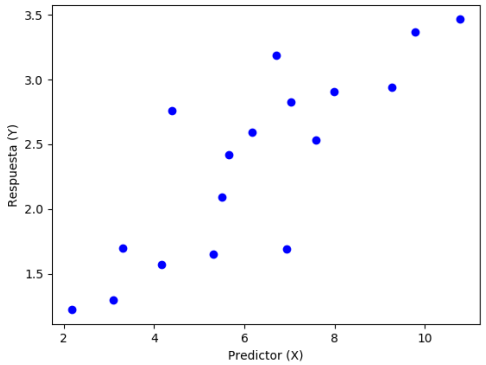

Función *Softmax*

- **La salida de la función *softmax* proporciona una distribución de probabilidad de las categorías: indica la probabilidad de pertenencia a una clase.**

$$P(y=j \mid x) = \frac{e^{x^T w}}{\sum_{l=1}^{K} e^{x^T w_l}}$$

Ejercicio 2: Regresión lineal

- Encontrar la recta que mejor aproxima los datos



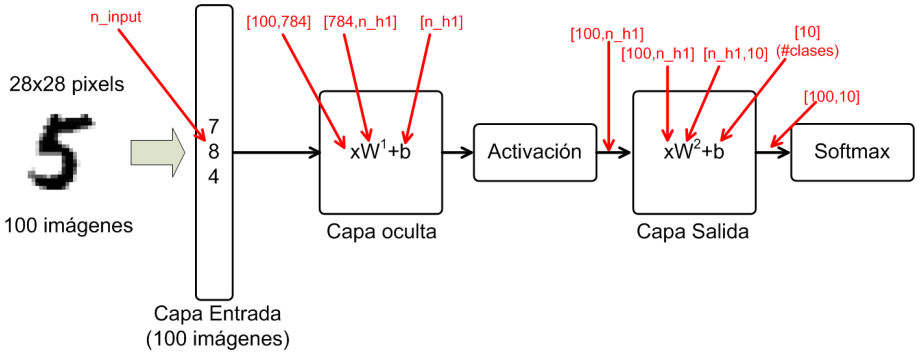
$$Y = wX + b$$

Ejercicio 3: Clasificación de imágenes con MLP

- Base de datos MNIST. Números escritos a mano 0-9
  - Imágenes de 28x28 pixels, etiquetadas

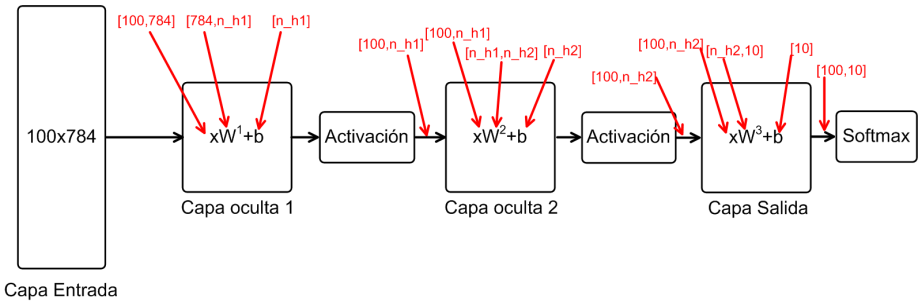
Ejercicio 3: Clasificación de imágenes con MLP

- Perceptrón Multicapa (MLP). Arquitectura (1 capa oculta)



Ejercicio 3: Clasificación de imágenes con MLP

- Perceptrón Multicapa (MLP). Añadir 2 capas ocultas
- Cambiar la función de activación por ReLU



Ejercicio 3: Clasificación de imágenes con MLP

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
```

Importamos librerias

```
# Parametros
learning_rate=0.001
train_iters=20
batch_size=100  # Tamaño del batch: entrenamos de 100 en 100 imágenes
n_input=784 # Numero de neuronas entrada. Las imagenes mnist son de 28x28 = 784 px
n_h1=256 # Numero de neuronas capa oculta 1
n_h2=256 # Numero de neuronas capa oculta 2
n_clases=10 # Numero de neuronas capa salida
```


Ejercicio 3: Definición de variables

```
# Definicion de placeholders para almacenar los datos (imagenes y etiquetas)
x=tf.placeholder(tf.float32,[None,n_input])  # Capa Entrada. None = dinamico
y=tf.placeholder(tf.float32,[None,n_clases]) # Capa Salida

# Definiciones capa oculta 1.
h1=tf.Variable(tf.random_normal([n_input,n_h1])) # Pesos 784x n_h1
b1=tf.Variable(tf.random_normal([n_h1])) # Bias n_h1
#l1=tf.nn.sigmoid(tf.add(tf.matmul(x,h1),b1))  # Activacion sigm capa 1, xW+b
l1=tf.nn.relu(tf.add(tf.matmul(x,h1),b1))  # Activacion ReLU capa 1, xW+b

# Definiciones capa salida
output=tf.Variable(tf.random_normal([n_h1,n_clases]))  # Pesos n_h2 x n_clases
bo=tf.Variable(tf.random_normal([n_clases])) # Bias n_clases
lo=tf.matmul(l1,output)+bo # Activacion

# Definicion funcion de coste.
cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=lo))

# Optimizador
optim=tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```


Ejercicio 3: Definicion de capas del MLP

```
# Definicion de placeholders para almacenar los datos (imagenes y etiquetas)
x=tf.placeholder(tf.float32,[None,n_input])  # Capa Entrada. None = dinamico
y=tf.placeholder(tf.float32,[None,n_clases]) # Capa Salida

# Definiciones capa oculta 1.
h1=tf.Variable(tf.random_normal([n_input,n_h1])) # Pesos 784x n_h1
b1=tf.Variable(tf.random_normal([n_h1])) # Bias n_h1
#l1=tf.nn.sigmoid(tf.add(tf.matmul(x,h1),b1))  # Activacion sigm capa 1, xW+b
l1=tf.nn.relu(tf.add(tf.matmul(x,h1),b1))  # Activacion ReLU capa 1, xW+b

# Definiciones capa salida
output=tf.Variable(tf.random_normal([n_h1,n_clases]))  # Pesos n_h2 x n_clases
bo=tf.Variable(tf.random_normal([n_clases])) # Bias n_clases
lo=tf.matmul(l1,output)+bo # Activacion

# Definicion funcion de coste.
cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=lo))

# Optimizador
optim=tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```


Ejercicio 3: Definición de capas del MLP

```
# Definicion de placeholders para almacenar los datos (imagenes y etiquetas)
x=tf.placeholder(tf.float32,[None,n_input])  # Capa Entrada. None = dinamico
y=tf.placeholder(tf.float32,[None,n_clases]) # Capa Salida

# Definiciones capa oculta 1.
h1=tf.Variable(tf.random_normal([n_input,n_h1])) # Pesos 784x n_h1
b1=tf.Variable(tf.random_normal([n_h1])) # Bias n_h1
#l1=tf.nn.sigmoid(tf.add(tf.matmul(x,h1),b1))  # Activacion sigm capa 1, xW+b
l1=tf.nn.relu(tf.add(tf.matmul(x,h1),b1))  # Activacion ReLU capa 1, xW+b

# Definiciones capa salida
output=tf.Variable(tf.random_normal([n_h1,n_clases]))  # Pesos n_h2 x n_clases
bo=tf.Variable(tf.random_normal([n_clases])) # Bias n_clases
lo=tf.matmul(l1,output)+bo # Activacion

# Definicion funcion de coste.
cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=lo))

# Optimizador
optim=tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Ejercicio 3: Definición función de coste

```
# Definicion de placeholders para almacenar los datos (imagenes y etiquetas)
x=tf.placeholder(tf.float32,[None,n_input])  # Capa Entrada. None = dinamico
y=tf.placeholder(tf.float32,[None,n_clases]) # Capa Salida

# Definiciones capa oculta 1.
h1=tf.Variable(tf.random_normal([n_input,n_h1])) # Pesos 784x n_h1
b1=tf.Variable(tf.random_normal([n_h1])) # Bias n_h1
#l1=tf.nn.sigmoid(tf.add(tf.matmul(x,h1),b1))  # Activacion sigm capa 1, xW+b
l1=tf.nn.relu(tf.add(tf.matmul(x,h1),b1))  # Activacion ReLU capa 1, xW+b

# Definiciones capa salida
output=tf.Variable(tf.random_normal([n_h1,n_clases]))  # Pesos n_h2 x n_clases
bo=tf.Variable(tf.random_normal([n_clases])) # Bias n_clases
lo=tf.matmul(l1,output)+bo # Activacion

# Definicion funcion de coste.
cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=lo))

# Optimizador
optim=tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```


Ejercicio 3: Optimización de la función de coste

```
# Definicion de placeholders para almacenar los datos (imagenes y etiquetas)
x=tf.placeholder(tf.float32,[None,n_input])  # Capa Entrada. None = dinamico
y=tf.placeholder(tf.float32,[None,n_clases]) # Capa Salida

# Definiciones capa oculta 1.
h1=tf.Variable(tf.random_normal([n_input,n_h1])) # Pesos 784x n_h1
b1=tf.Variable(tf.random_normal([n_h1])) # Bias n_h1
#l1=tf.nn.sigmoid(tf.add(tf.matmul(x,h1),b1))  # Activacion sigm capa 1, xW+b
l1=tf.nn.relu(tf.add(tf.matmul(x,h1),b1))  # Activacion ReLU capa 1, xW+b

# Definiciones capa salida
output=tf.Variable(tf.random_normal([n_h1,n_clases]))  # Pesos n_h2 x n_clases
bo=tf.Variable(tf.random_normal([n_clases])) # Bias n_clases
lo=tf.matmul(l1,output)+bo # Activacion

# Definicion funcion de coste.
cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=lo))

# Optimizador
optim=tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```


Ejercicio 3: Definición de funciones auxiliares

```
# Clase con la maxima probabilidad softmax
pred=tf.equal(tf.argmax(lo,1),tf.argmax(y,1))

# Definicion Accuracy
accuracy=tf.reduce_mean(tf.cast(pred,tf.float32))

# Definicion del inicializador
init=tf.global_variables_initializer()
```


Ejercicio 3: Inicializacion de variables

```
# Clase con la maxima probabilidad softmax
pred=tf.equal(tf.argmax(lo,1),tf.argmax(y,1))

# Definicion Accuracy
accuracy=tf.reduce_mean(tf.cast(pred,tf.float32))

# Definicion del inicializador
init=tf.global_variables_initializer()
```

Ejercicio 3: Creación de la sesión

```
with tf.Session() as sess:
    sess.run(init)
    for iter in range(train_iters):
        avg_cost=0
        # Entrenamos de forma incremental usando batches
        total_batch=int(mnist.train.num_examples/batch_size) # Numero de batches
        for i in range(total_batch):
            # Cargamos un batch
            batch_x, batch_y=mnist.train.next_batch(batch_size)
            # Entrenamos con los datos del batch
            sess.run(optim,feed_dict={x:batch_x, y:batch_y})
```


Ejercicio 3: Inicialización de variables

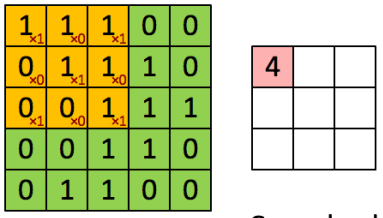
```
with tf.Session() as sess:
    sess.run(init)
    for iter in range(train_iters):
        avg_cost=0
        # Entrenamos de forma incremental usando batches
        total_batch=int(mnist.train.num_examples/batch_size) # Numero de batches
        for i in range(total_batch):
            # Cargamos un batch
            batch_x, batch_y=mnist.train.next_batch(batch_size)
            # Entrenamos con los datos del batch
            sess.run(optim,feed_dict={x:batch_x, y:batch_y})
```


Ejercicio 3: Ejecución

```
with tf.Session() as sess:
    sess.run(init)
    for iter in range(train_iters):
        avg_cost=0
        # Entrenamos de forma incremental usando batches
        total_batch=int(mnist.train.num_examples/batch_size) # Numero de batches
        for i in range(total_batch):
            # Cargamos un batch
            batch_x, batch_y=mnist.train.next_batch(batch_size)
            # Entrenamos con los datos del batch
            sess.run(optim,feed_dict={x:batch_x, y:batch_y})
```


Ejercicio 4: Clasificación de imágenes con CNN

- Convolutional Neural Networks (CNN)
  - Usan el operador de convolución para extraer características en cada capa



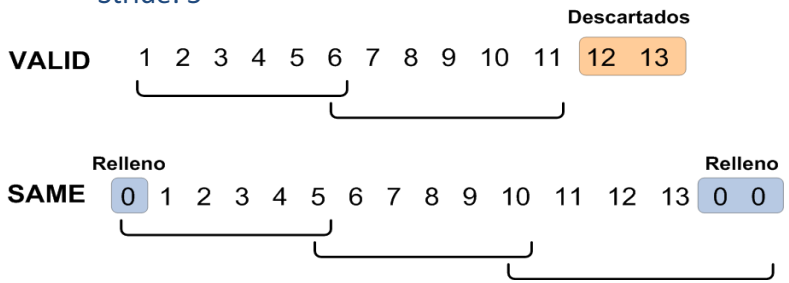
* https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolution.html

Ejercicio 4: Clasificación de imágenes con CNN

- **Convolutional Neural Networks (CNN)**
  - **Stride:** Número de pixels que movemos la ventana de convolución entre convoluciones consecutivas. **Ej. stride=2 con kernels de tamaño 2x2 reduce a la mitad la imagen tras la primera convolucion**
  - **Padding:** relleno de los bordes con ceros para poder tener un número entero de bloques. Dos modos:
    - **VALID:** En lugar de rellenar, descarta el último bloque
    - **SAME:** Rellena con ceros para tener un número entero de bloques

Ejercicio 4: Clasificación de imágenes con CNN

- **Convolutional Neural Networks (CNN)**
  - **Padding. Ejemplo.**
    - Tamaño entrada: 13
    - Tamaño filtro: 6
    - Stride: 5



Ejercicio 4: Clasificación de imágenes con CNN

- **Convolutional Neural Networks (CNN)**

Kernel (batch, dim1, dim2, canales_entrada, canales_salida)

```
tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],padding='SAME', name=name)
```

Tensor entrada (batch, dim1, dim2, canales_entrada)

Dimensión 1 imagen

Dimensión 2 imagen

Canales

Filtros

```
x = tf.placeholder(tf.float32, shape=[None, 28,28,1])
W = weight_variable([5, 5, 1, 32])
```

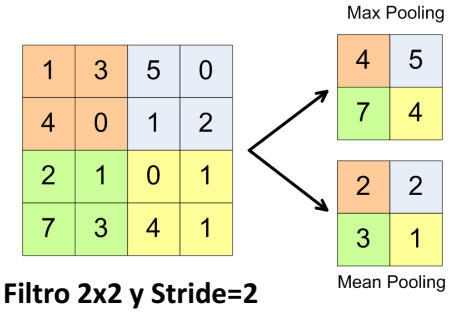
← Filtros 5x5, 1 canal entrada, 32 filtros salida

Ejercicio 4: Clasificación de imágenes con CNN

- **Convolutional Neural Networks (CNN)**
  - **Pooling.** Depués de la capa de activación (ReLU), puede realizarse una operación de pooling.
    - Consiste en realizar un **submuestreo** para reducir el tamaño del resultado de la convolución.
    - Sustituye cada subregion por el máximo (maxpooling) o la media (mean pooling)
    - No es obligatorio, y de hecho, hay una tendencia a evitar las capas de pooling.

Ejercicio 4: Clasificación de imágenes con CNN

- **Convolutional Neural Networks (CNN)**
  - **Pooling.** Tras la capa de activación (ReLU), puede realizarse una operación de **pooling**.



Filtro 2x2 y Stride=2

Ejercicio 4: Clasificación de imágenes con CNN

- **Convolutional Neural Networks (CNN)**
  - **Pooling.** Tras la capa de activación (ReLU), puede realizarse una operación de **pooling**.

```
tf.nn.max_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME', name=name)
```

Tensor entrada (salida de la convolución)

Dimensiones kernel pooling

Saltos pooling 2x2

Ejercicio 4: Clasificación de imágenes con CNN

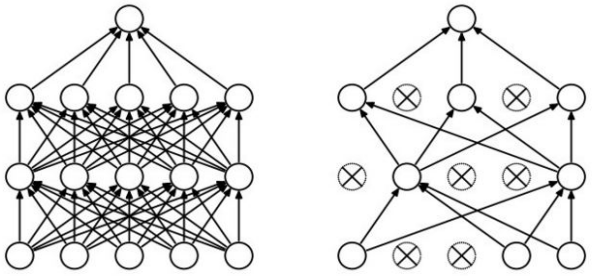
- **Convolutional Neural Networks (CNN)**
  - **Dropout.** Es una forma de regularización para mitigar el overfitting. Consiste en poner a cero un número aleatorio de neuronas con una probabilidad (1-p) en tiempo de entrenamiento.
  - **Fuerza la red** a proporcionar la salida correcta incluso cuando se anulan algunas conexiones.

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

Sugerencia: https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Ejercicio 4: Clasificación de imágenes con CNN

- **Convolutional Neural Networks (CNN)**
  - **Dropout.** Es un método para mitigar el overfitting.



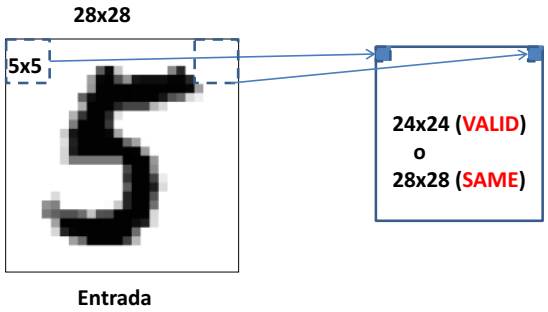
(a) Standard Neural Net (b) After applying dropout.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of machine learning research, 15(1), 1929-1958.

Ejercicio 4: Clasificación de imágenes con CNN

Convolutional Neural Networks (CNN)

- Dataset MNIST – Números escritos a mano

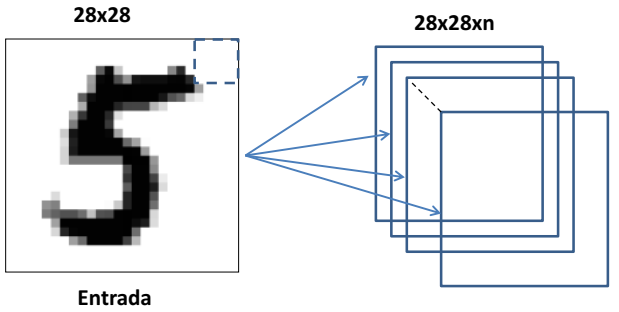


- Tras la primera convolución, la primera capa oculta tendrá *n* mapas de características de dimensión 24x24 (*n*x24x24)

Ejercicio 4: Clasificación de imágenes con CNN

Convolutional Neural Networks (CNN)

- Convolución y mapas de características (kernel 5x5)

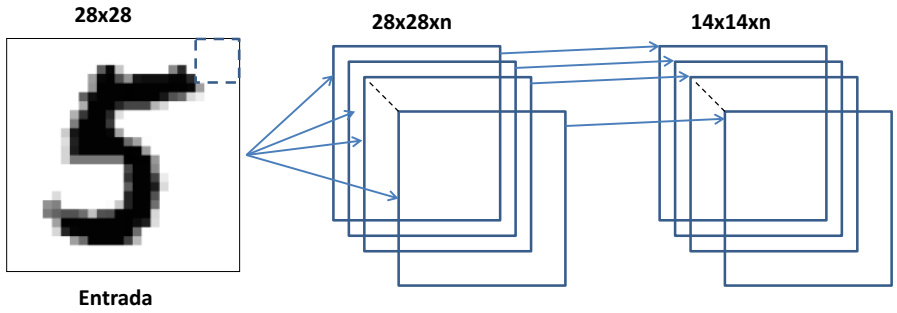


- Tras la primera convolución, la primera capa oculta tendrá *n* mapas de características de dimensión 28x28 (28x28xn) – Usamos Padding "SAME"

Ejercicio 4: Clasificación de imágenes con CNN

Convolutional Neural Networks (CNN)

- Pooling (Kernel 2x2)

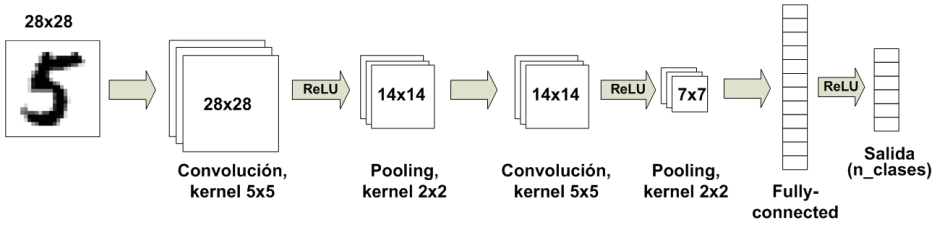


- Tras el pooling 2x2 tendremos n filtros de 14x14

Ejercicio 4: Clasificación de imágenes con CNN

Convolutional Neural Networks (CNN)

- Arquitectura completa



Ejercicio 4: Clasificación de imágenes con CNN

Convolutional Neural Networks (CNN)

- Arquitectura completa