

Convolutional Neural Networks

Pablo Augusto Negri

Grupo Procesamiento de Imágenes y Visión por Computadora
Instituto de Ciencias de la Computación (ICC)
Departamento de Computación - Universidad de Buenos Aires (UBA)
CONICET

- 1 Introducción
- 2 Redes Neuronales
- 3 Aprendizaje con Backpropagation
- 4 Redes Neuronales Convolucionales (CNN)

Introducción

Introducción

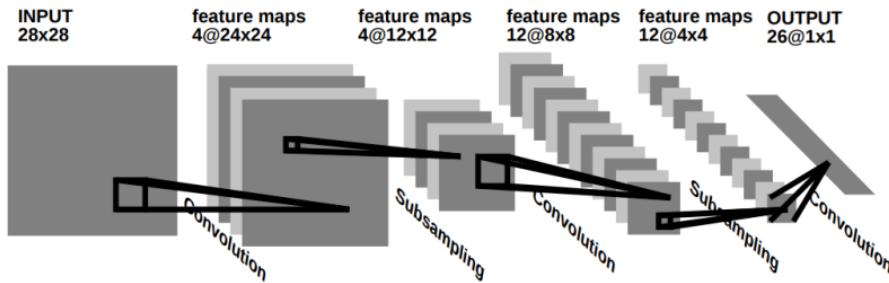
El modelo de Visión por Ordenador se puede dividir en tres niveles:
Procesamiento, Análisis e Interpretación.



Introducción

En 1995 LeCun y Bengio [1] proponen una arquitectura de red neuronal basado en 3 pilares:

- ① Campos receptivos locales,
- ② Pesos compartidos (o pesos replicados),
- ③ Subsampling espacial o temporal.

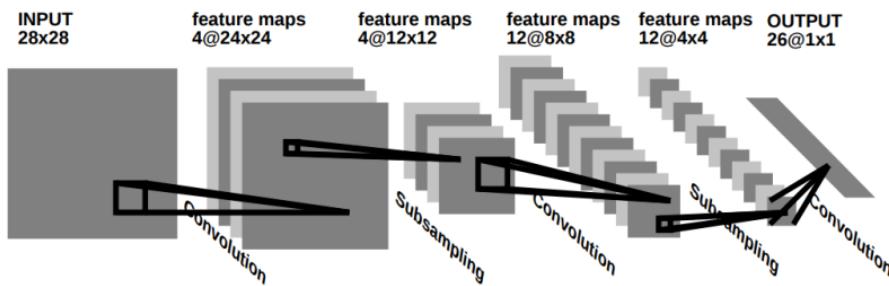


¹Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

Introducción

En 1995 LeCun y Bengio [1] proponen una arquitectura de red neuronal basado en 3 pilares:

- ① Campos receptivos locales,
- ② Pesos compartidos (o pesos replicados),
- ③ Subsampling espacial o temporal.



Esta red posee en su primera capa un cálculo de descriptores, que es entrenado por backpropagation.

¹Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

Introducción

Las Redes Neuronales Convolucionadas integran los dos niveles superiores en un mismo proceso.

Nivel Medio/Alto



Imagen -> Descriptores -> Conocimiento

Nivel Bajo

Imagen -> Imagen

Introducción

Las Redes Neuronales Convolucionadas integran los dos niveles superiores en un mismo proceso.

Nivel Medio/Alto



Imagen -> Descriptores -> Conocimiento

Nivel Bajo

Imagen -> Imagen

Esto significa que la etapa de extracción de descriptores está fuertemente relacionada con el problema a resolver.

Introducción

Las Redes Neuronales Convolucionadas integran los dos niveles superiores en un mismo proceso.

Nivel Medio/Alto



Imagen -> Descriptores -> Conocimiento

Nivel Bajo

Imagen -> Imagen

Esto significa que la etapa de extracción de descriptores está fuertemente relacionada con el problema a resolver.

DEPENDE DE LA BASE DE ENTRENAMIENTO!!!

Introducción

Plan de la clase:

- Breve introducción a las redes neuronales.
- Descripción y práctica para aprendizaje de pesos (backpropagation).
- Implementación de una CNN en reconocimiento de caracteres.

Introducción

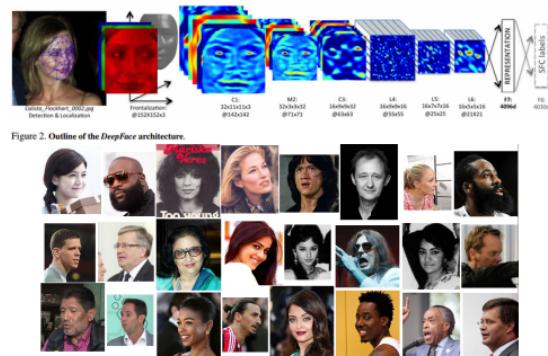
Proyectos de tesis:

- Reconocimiento de Rostro y Gestos.
- Segmentación de Imágenes de esféricas.
- Reconocimiento de gestos con cámara de eventos.

Introducción

Proyectos de tesis:

- **Reconocimiento de Rostro y Gestos**
- Segmentación de Imágenes de esféricas.
- Reconocimiento de gestos con cámara de eventos.



Introducción

Proyectos de tesis:

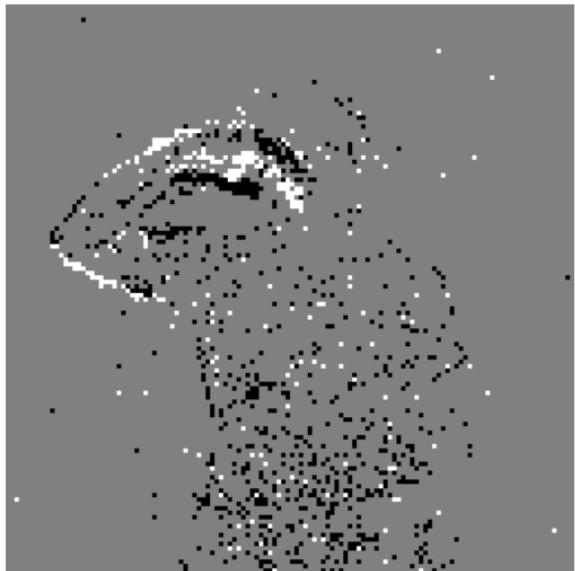
- Reconocimiento de Rostro.
- **Segmentación de Imágenes de esféricas.**
- Reconocimiento de gestos con cámara de eventos.



Introducción

Proyectos de tesis:

- Reconocimiento de Rostro.
- Segmentación de Imágenes de esféricas.
- **Reconocimiento de gestos con cámara de eventos.**

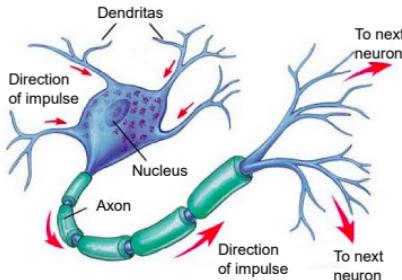


Redes Neuronales

Modelo de neurona

El Modelo Neuronal Clásico está compuesto por:

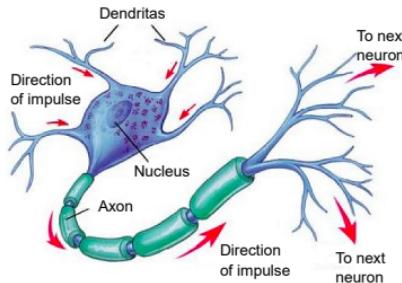
- Dendrites que son las terminales de entrada al núcleo,
- Axones sus salidas.
- Las interfaces entre las dendritas y los axones son las sinápsis.



Modelo de neurona

El Modelo Neuronal Clásico funciona de la siguiente manera:

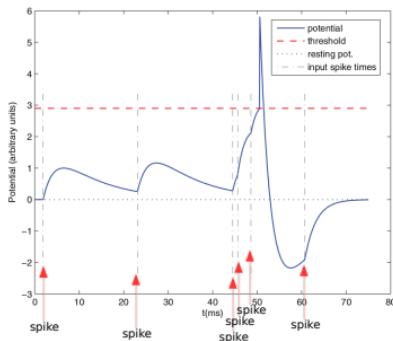
- La dinámica de la neurona está controlada por señales eléctricas de entrada.
- Estas señales de corta duración se denominan *spikes*.
- Los spikes llegan al núcleo a través de las dendritas y se acumulan en su membrana.
- Cuando el potencial de la membrana alcanza un umbral, la neurona (pre-sináptica) dispara una señal eléctrica a la próxima neurona (post-sináptica) vía los axones.



Modelo de neurona

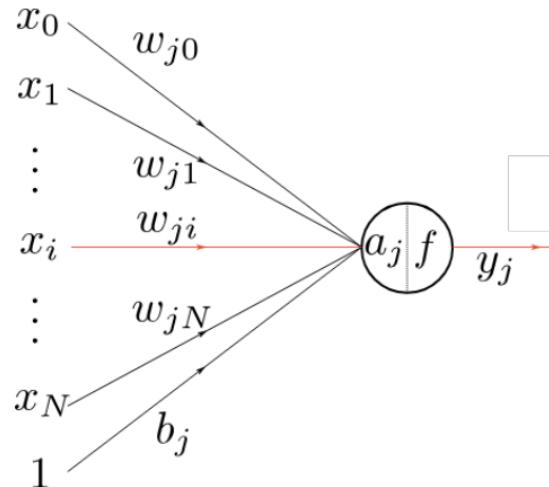
El Modelo Neuronal Clásico funciona de la siguiente manera:

- La dinámica de la neurona está controlada por señales eléctricas de entrada.
- Estas señales de corta duración se denominan *spikes*.
- Los spikes llegan al núcleo a través de las dendritas y se acumulan en su membrana.
- Cuando el potencial de la membrana alcanza un umbral, la neurona (pre-sináptica) dispara una señal eléctrica a la próxima neurona (post-sináptica) vía los axones.



Modelo de neurona

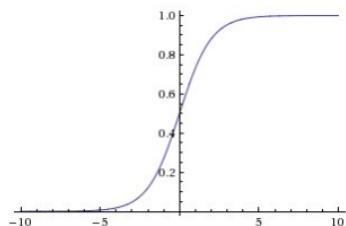
El Modelo Neuronal matemático (perceptrón) de una neurona j



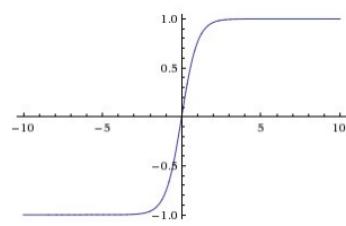
- El set de entradas $\{x_0, \dots, x_N\}$ se conecta a la neurona vía
- las sinapsis con sus los pesos $\{\omega_{j0}, \dots, \omega_{jN}\}$ y el bias b_j
- acumulando la exitación en $a_j = b_j + \sum_i \omega_{ji} x_i$
- que es aplicada a la función de activación f para obtener su salida $y_j = f(a_j)$

Modelo de neurona

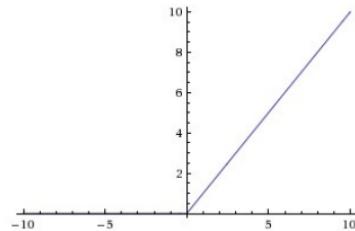
Las funciones de activación f mas utilizadas hoy en día son:



sigmoide



tanh

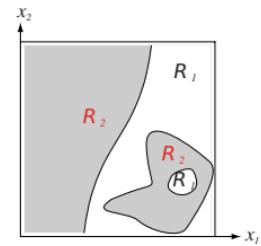
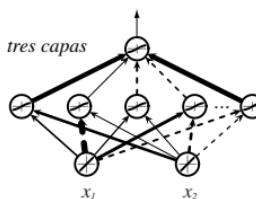
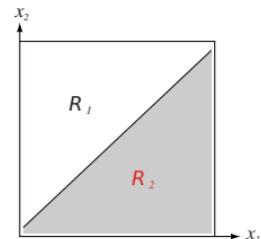
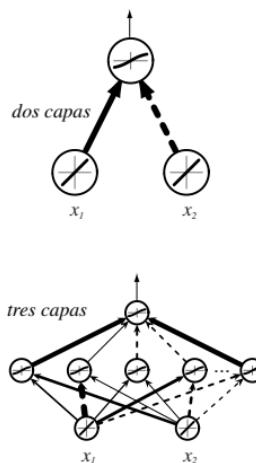


relu

Cada una tiene sus pros y contras, pero lo importante es que estas funciones deben ser *derivables*, para luego ser utilizadas en el aprendizaje.

Red de Multiples Capas

- La arquitectura de base del perceptrón posee dos capas y permite resolver problemas LINEALMENTE SEPARABLES.
- Para problemas LINEALMENTE NO SEPARABLES, la solución es agregar capas a la red, incorporando no linealidades en la resolución.



Red de Multiples Capas

- Las redes multicapa nolineales, compuestas por unidades de entrada, capas escondidas, y unidades de salida, poseen un enorme poder computacional o expressive power.
- En clasificación, con c unidades de salida, una por cada clase, la señal discriminante k de cada una de esas clases es :

$$g_k(\mathbf{x}) = z_k = f \left(\sum_{j=1}^{n_H} \omega_{kj} f \left(\sum_{i=1}^d \omega_{ji} x_i + b_j \right) + b_k \right)$$

- En teoría, con la ecuación anterior, y dado un suficiente número de neuronas escondidas n_H , se podría representar cualquier función continua de entrada.

Red de Multiples Capas

- Las redes multicapa nolineales, compuestas por unidades de entrada, capas escondidas, y unidades de salida, poseen un enorme poder computacional o expressive power.
- En clasificación, con c unidades de salida, una por cada clase, la señal discriminante k de cada una de esas clases es :

$$g_k(\mathbf{x}) = z_k = f \left(\sum_{j=1}^{n_H} \omega_{kj} f \left(\sum_{i=1}^d \omega_{ji} x_i + b_j \right) + b_k \right)$$

- En teoría, con la ecuación anterior, y dado un suficiente número de neuronas escondidas n_H , se podría representar cualquier función continua de entrada.
- El problema es que no sabemos de antemano cual es la arquitectura óptima para obtener esto.

Red de Multiples Capas

- Las redes multicapa nolineales, compuestas por unidades de entrada, capas escondidas, y unidades de salida, poseen un enorme poder computacional o expressive power.
- En clasificación, con c unidades de salida, una por cada clase, la señal discriminante k de cada una de esas clases es :

$$g_k(\mathbf{x}) = z_k = f \left(\sum_{j=1}^{n_H} \omega_{kj} f \left(\sum_{i=1}^d \omega_{ji} x_i + b_j \right) + b_k \right)$$

- En teoría, con la ecuación anterior, y dado un suficiente número de neuronas escondidas n_H , se podría representar cualquier función continua de entrada.
- El problema es que no sabemos de antemano cual es la arquitectura óptima para obtener esto.
- Y TENEMOS QUE ENTRENAR LA RED!!!

Red de Multiples Capas

- Las redes multicapa nolineales, compuestas por unidades de entrada, capas escondidas, y unidades de salida, poseen un enorme poder computacional o expressive power.
- En clasificación, con c unidades de salida, una por cada clase, la señal discriminante k de cada una de esas clases es :

$$g_k(\mathbf{x}) = z_k = f \left(\sum_{j=1}^{n_H} \omega_{kj} f \left(\sum_{i=1}^d \omega_{ji} x_i + b_j \right) + b_k \right)$$

- En teoría, con la ecuación anterior, y dado un suficiente número de neuronas escondidas n_H , se podría representar cualquier función continua de entrada.
- El problema es que no sabemos de antemano cual es la arquitectura óptima para obtener esto.
- Y TENEMOS QUE ENTRENAR LA RED!!!
- COMO ES POSIBLE ENTRENAR LOS PESOS ENTRE LA ENTRADA Y LAS CAPAS ESCONDIDAS???

Aprendizaje con Backpropagation

Backpropagation

- Backpropagation es uno de los métodos más simples e instructivos para el entrenamiento supervisado de redes neuronales multicapas.

Backpropagation

- Función de costo (o error en el entrenamiento)

$$J(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^c (t_k - z_k)^2 = \frac{1}{2} \|\mathbf{t} - \mathbf{z}\|^2 \quad (1)$$

- La regla de aprendizaje sigue el descenso del gradiente:

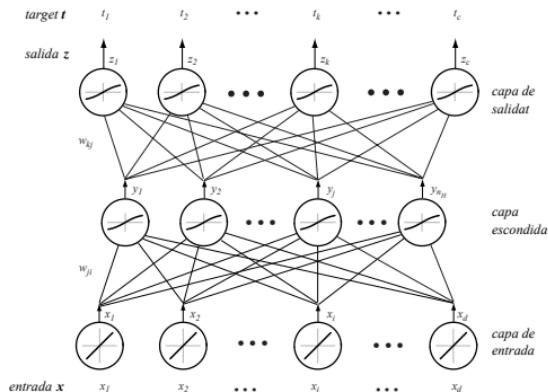
$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

$$\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}} \quad (2)$$

η es la variable de aprendizaje.

- La ley de actualización de los pesos en la iteración m es simplemente:

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta \mathbf{w}(m)$$



Backpropagation

Para resolver la ecuación 2:

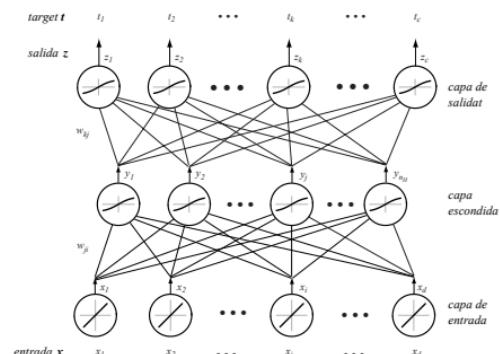
- Considerando los pesos de la capa escondida a la salida w_{kj} , usamos la regla de la cadena:

$$\frac{\partial J}{\partial w_{pq}} = \frac{\partial J}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}} = -\delta_k \frac{\partial a_k}{\partial w_{kj}} \quad (3)$$

donde δ_k es la sensibilidad de la neurona de salida k que muestra como cambia el error total a partir de su activación.

- Aplicando la regla de la cadena, derivando eq. 1 y suponiendo que f' existe, obtenemos la expresión de δ_k :

$$\delta_k = -\frac{\partial J}{\partial a_k} = \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial a_k} = (t_k - z_k) f'(a_k)$$



Backpropagation

- La segunda derivada de eq. 3 se deduce de:

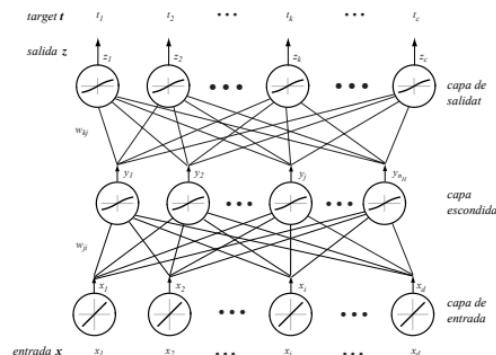
$$a_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj}$$

donde el bias b_k se integra al vector de pesos. Luego:

$$\frac{\partial a_k}{\partial w_{kj}} = y_j$$

- Agrupando, obtenemos el gradiente de los pesos de la capa escondida a la salida:

$$\Delta w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(a_k) y_j \quad (4)$$



Backpropagation

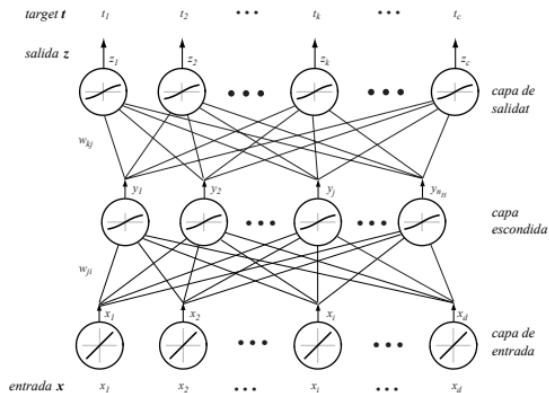
El cálculo del gradiente de las sinapses entre la entrada a la capa escondida es más sutil.

- De la ecuación 2:

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \quad (5)$$

- El primer término involucra todos los pesos w_{kj}

$$\begin{aligned}
 \frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j} \left[\frac{1}{2} \sum_{k=1}^c (t_k - z_k)^2 \right] \\
 &= - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial y_j} \\
 &= - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial a_k} \frac{\partial a_k}{\partial y_j} \\
 &= - \sum_{k=1}^c (t_k - z_k) f'(a_k) w_{kj}
 \end{aligned}$$



Backpropagation

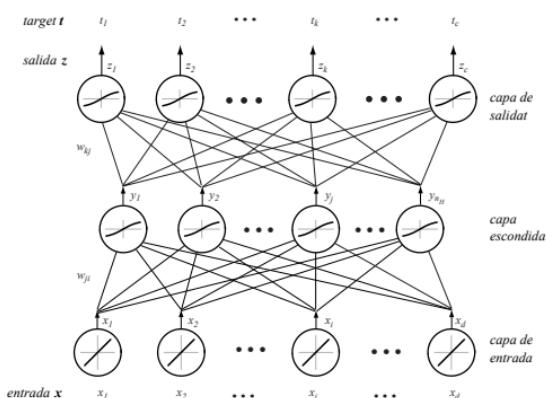
El cálculo del gradiente de las sinapses entre la entrada a la capa escondida es más sutil.

- La sensibilidad de una neurona escondida es:

$$\delta_j = f'(a_j) \sum_{k=1}^c w_{kj} \delta_k \quad (6)$$

- La eq. 6 muestra como el error de una neurona escondida es la suma de las sensibilidades individuales a la neurona de salida ponderada por w_{kj} y multiplicados por $f'(a_k)$.
 - La regla de aprendizaje es:

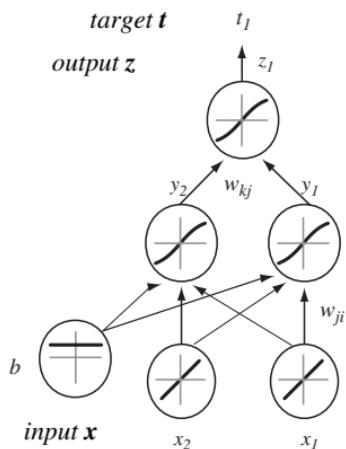
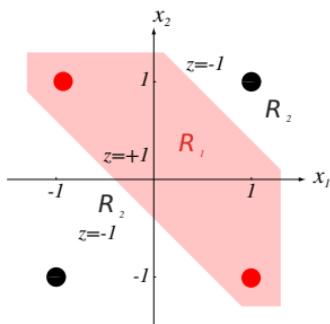
$$\Delta w_{ji} = \eta x_i \delta_j = \eta \left[\sum_{k=1}^c w_{kj} \delta_k \right] f'(a_j) x_i \quad (7)$$



EJERCICIO 1

Se va a resolver el problema del XOR utilizando una red con una capa intermedia de 2 neuronas, y una neurona de salida. Las funciones de activación son de tipo **tanh**. Vamos a simplificar las cuentas fijando el valor de $b = 1$. Se considera la siguiente tabla de verdad:

x_1	x_2	z
-1	-1	-1
-1	1	1
1	1	-1
1	-1	1



Redes Neuronales Convolucionales (CNN)

Introducción

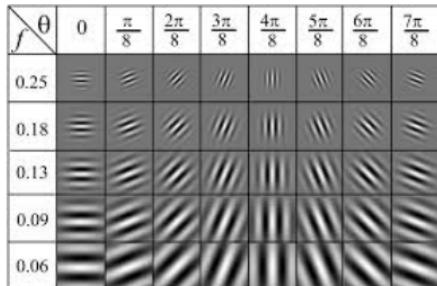
Volvamos a tomar el modelo de Visión por Ordenador dividido en tres niveles:
Procesamiento, Análisis e Interpretación.



Introducción

El nivel Medio o capa de Análisis:

- Aplica algoritmos de segmentación para extracción de características locales de la imagen.
- Un tipo de segmentación básico es el filtrado con diferentes máscaras: Sobel, Prewitt, Gabor.
- Estos filtrados permiten cálculos de gradiente local.



Gabor

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Sobel

Introducción

Para realizar el filtrado, usamos la operación de CONVOLUCIÓN de tipo matricial.
La Convolución entre una matriz de entrada \mathbf{i} y un filtro \mathbf{w} se define como:

$$c[j, k] = \mathbf{w} * \mathbf{i} = \sum_{q=0}^a \sum_{p=0}^b w[q, p] i[j - q, k - p]$$

Introducción

Para realizar el filtrado, usamos la operación de CONVOLUCIÓN de tipo matricial.
La Convolución entre una matriz de entrada \mathbf{i} y un filtro \mathbf{w} se define como:

$$c[j, k] = \mathbf{w} * \mathbf{i} = \sum_{q=0}^a \sum_{p=0}^b w[q, p] i[j - q, k - p]$$

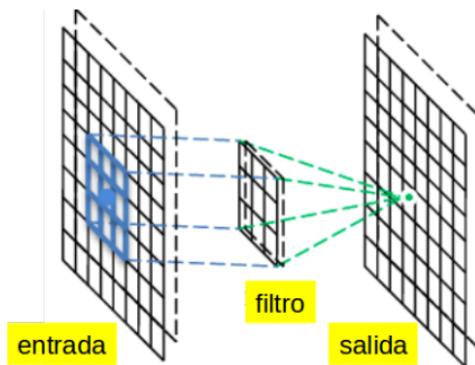
y teniendo mucho cuidado tal que p y q tengan valores para que los indices de $w[q, p]$ y $i[j - q, k - p]$ sean válidos.

Introducción

Para realizar el filtrado, usamos la operación de CONVOLUCIÓN de tipo matricial.
La Convolución entre una matriz de entrada \mathbf{i} y un filtro \mathbf{w} se define como:

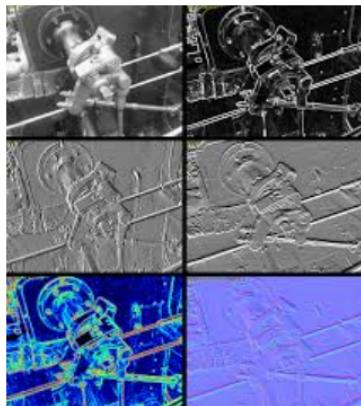
$$c[j, k] = \mathbf{w} * \mathbf{i} = \sum_{q=0}^a \sum_{p=0}^b w[q, p] i[j - q, k - p]$$

y teniendo mucho cuidado tal que p y q tengan valores para que los índices de $w[q, p]$ y $i[j - q, k - p]$ sean válidos.

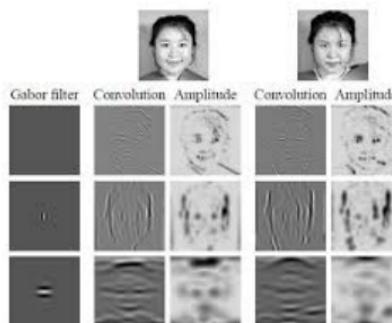


Introducción

El resultado del filtrado va a depender de los valores de la matriz w .



Aplicacion de Sobel

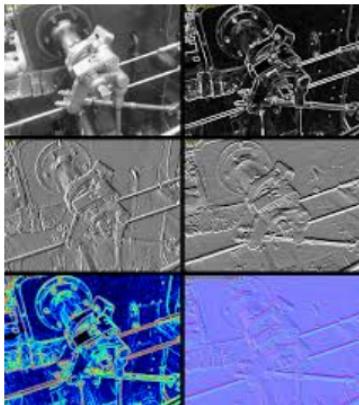


Gabor Wavelets
distintas escalas

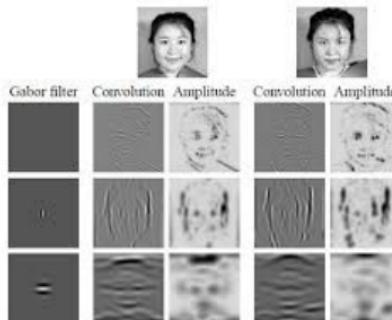
se conoce entonces de antemano la salida esperada del filtrado.

Introducción

El resultado del filtrado va a depender de los valores de la matriz w .



Aplicacion de Sobel

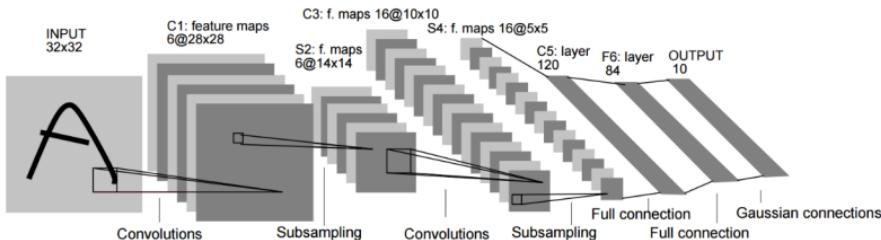


Gabor Wavelets
distintas escalas

se conoce entonces de antemano la salida esperada del filtrado.
Los $w(p, q)$ de los filtros se eligieron expresamente!!!

Introducción

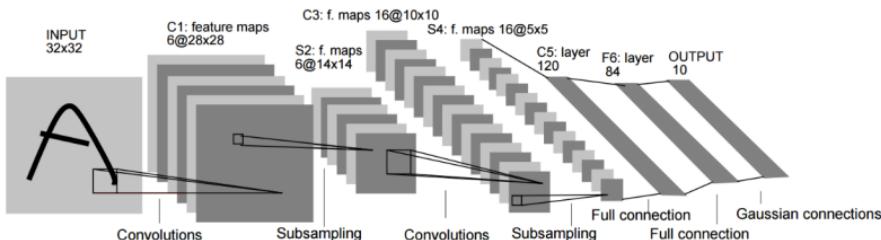
La propuesta de Lecun y Bengio [2] es que estos pesos sean *entrenados* en una arquitectura de red neuronal:



²Y. LeCun, P. Haffner, L. Bottou, Y. Bengio. Object recognition with gradient-based learning. In Shape, Contour and Grouping in Computer Vision, pp. 319-345, 1999.

Introducción

La propuesta de Lecun y Bengio [2] es que estos pesos sean *entrenados* en una arquitectura de red neuronal:



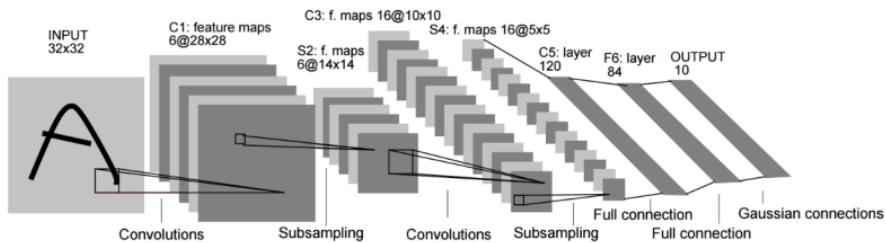
Entonces tenemos una arquitectura con:

- ① Campos receptivos locales,
- ② Pesos compartidos (o pesos replicados),
- ③ Subsampling espacial o temporal,
- ④ Capas de salida Densas.

²Y. LeCun, P. Haffner, L. Bottou, Y. Bengio. Object recognition with gradient-based learning. In Shape, Contour and Grouping in Computer Vision, pp. 319-345, 1999.

Arquitectura de la Red

Retomamos la arquitectura LeNet5, de Y. Lecun.

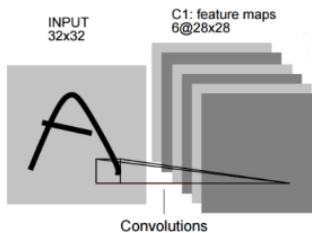


El detalle es el siguiente:

Tipo	Entrada	CONV	POOL	CONV	POOL	FC	FC	OUTPUT
Nombre		C1	S2	C3	S4	F5	F6	
Dimension	32x32	6x28x28	6x14x14	16x10x10	16x5x5	120	84	10

La entrada consiste en una matriz de un solo plano de 32x32 pixels. Esto significa que es una imagen en niveles de gris. La salida es un vector de 10 unidades, donde cada una da la probabilidad de la pertenencia de la imagen de entrada a una de las 10 clases (en el MNIST son dígitos manuscritos).

Arquitectura de la Red : CONV:C1



Esta capa se obtiene de la convolución de 6 filtros de 5x5 pixeles. La función de activación es:

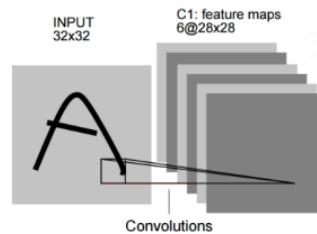
$$A_{1i} = \mathbf{w}_{1i} * \mathbf{I} + b_{1i}, \quad i = 1, \dots, 6$$

\mathbf{w}_{1i} es el *i*ésimo filtro, y b_{1i} es el bias. El tamaño de A_{1i} es de $28 = 32 - 5 + 1$. Los parámetros a ser entrenados en cada capa son 25, más el bias. En total son 156 parámetros en esta capa.

Para el cálculo de la convolución se incorporan dos conceptos:

- *padding*: es la cantidad de pixels con valor 0 que se agregan por fuera de la imagen de entrada ($p = 0$)
- *stride*: es el paso de convolución, o sea, la cantidad de pixeles que se desplaza el filtro entre una convolución y la siguiente ($s = 1$).

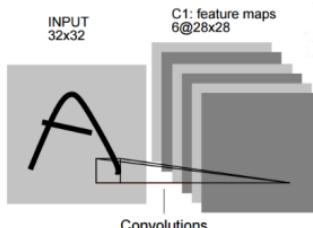
Arquitectura de la Red : CONV:C1



Luego de la convolución se aplica la función *sigmoide*:

$$C_{1i} = \frac{1}{1 + e^{-A_{1i}}}, i = 1, \dots, 6$$

Arquitectura de la Red : CONV:C1



Tips de implementación en MATLAB.

Dado que la definición matemática de la convolución involucra un flipping de la matriz de entrada, invirtiendo sus líneas y columnas, al utilizar la convolución de MATLAB, es preciso hacer un flip de la máscara o filtro W ANTES de la convolución. Si W es una máscara de 3x3 pixels:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \xrightarrow{\text{flip}} \quad \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix}$$

Esto se puede realizar con

```
> W = rot90(W, 2);
```

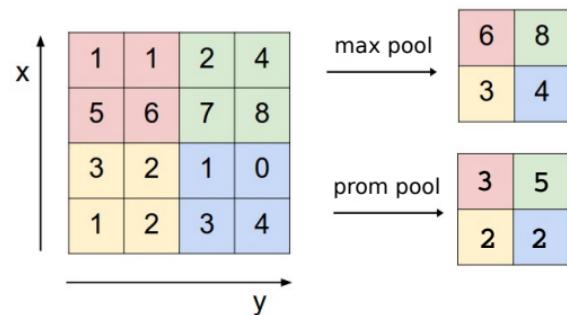
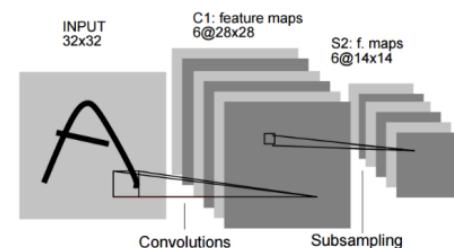
Arquitectura de la Red : POOL:S2

La salida de C1 se propaga a la capa de subsampleo o pooling S2.

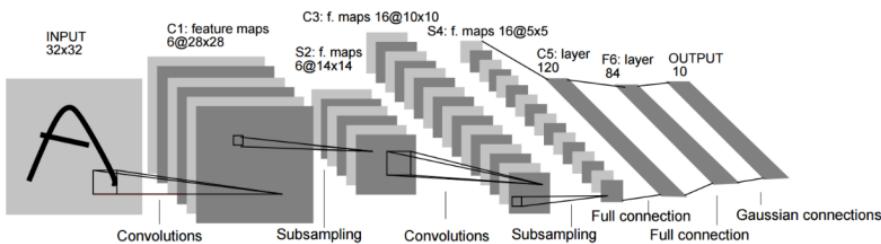
El tamaño de pooling es de 2x2 pixeles sin overlapping (*stride = 2*), reduciendo las matrices de C1 a la mitad. De esta manera, la capa S2 termina con un tamaño de 6x14x14.

Hay dos tipos de pooling: maximo y promedio. En la grafica vemos el resultado de aplicarlos a una matriz de 4x4 usando un *poolDim* = 2 sin overlapping.

El pooling puede resolverse eficientemente con la función **conv2**.

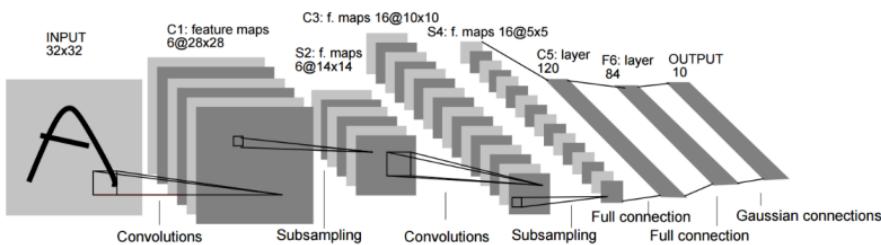


Arquitectura de la Red



- Las capas C3 y S4 son una capa convolucional seguida de un pooling.
- Las capas FC:F5 estan completamente conectadas entre las salidas de S4 (400 neuronas) y las neuronas escondidas de F5 (120). Las funciones de activación son tambien *sigmoide*.
- Entre F5 y F6 es tambien una conexión completa entre las neuronas.
- Finalmente, el valor de cada neurona de salida mide la probabilidad de que la imagen de entrada pertenezca a esta clase.

Arquitectura de la Red



- Esto se calcula mediante la función softmax.

$$P(y^{(i)} = k | z^{(i)}; W^{F6}, b^{F6}) = \frac{e^{W_k^{F6} z + b_k^{F6}}}{\sum_j^K e^{W_j^{F6} z + b_j^{F6}}} \quad (8)$$

donde W_i^{F6} son los pesos conectando la capa F6 con la neurona de salida i , y b_i es el bias correspondiente, z es la salida de la capa FC precedente, y $K = 10$.

Aprendizaje: Backpropagation

Llamamos ahora:

- δ^{l+1} es el termino de sensibilidad (o error) de la capa $(l + 1)$
- la red posee una función de costo $J(W, b; x, y)$.

Si la capa l es de tipo FC, y se conecta con $(l + 1)$, la sensibilidad o error se propaga como: $\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \bullet f'(z^{(l)})$.

$$\begin{aligned}\nabla_{W^{(l)}} J(W, b; x, y) &= \delta^{(l+1)} (a^{(l)})^T, \\ \nabla_{b^{(l)}} J(W, b; x, y) &= \delta^{(l+1)}\end{aligned}$$

donde $a^{(l)}$ es la entrada de la capa l

Aprendizaje: Backpropagation

En el caso que la capa l sea convolucional/pooling, el error se propaga como

$$\delta_k^{(l)} = \text{upsample} \left((W_k^{(l)})^T \delta_k^{(l+1)} \right) \bullet f'(z_k^{(l)}) \quad (9)$$

k es el número de filtro convolucional de esta capa, y $f'(z_k^{(l)})$ es la derivada de la función de activación. La operación `upsample` propaga el error a través de la capa pooling hacia la capa convolucional. El proceso inverso al pooling significa volver al tamaño original (que espera para hacer la convolución). Para el caso del pooling promedio, se puede distribuir el valor del error en todo el parche del pooling.

Los gradientes se calculan como:

$$\begin{aligned}\nabla_{W_k^{(l)}} J(W, b; x, y) &= \sum_{i=1}^m (a_i^{(l)}) * \text{rot90}(\delta_k^{(l+1)}, 2), \\ \nabla_{b_k^{(l)}} J(W, b; x, y) &= \sum_{a,b} (\delta_k^{(l+1)})_{a,b}.\end{aligned}$$

Aprendizaje: Backpropagation

Tips para realizar el upsampling en MATLAB.

La función `kron` de MATLAB puede ser utilizada para el upsampling del error proveniente de la capa siguiente, para poder propagarlo en la capa de pooling de tipo promedio. Esta función calcula el producto de tipo delta Kronecker entre dos matrices. Suponiendo se recibe una matriz delta de 2x2, y se debe hacer el upsampling a una de 4x4, la operación es algo como:

$$\text{kron} \left(\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \right) \xrightarrow{\text{flip}} \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{pmatrix}$$

Luego de que el error ha sido repartido en la nueva matriz, lo que queda dividirlo por el tamaño de la región de pooling y propagarlo a la capa de convolución. La implementación sería:

Esto se puede realizar con

```
> deltaPool = (1/(poolDim*poolDim)) *  
kron(delta, ones(poolDim));
```

Aprendizaje: Backpropagation

La función de salida se calcula mediante la ecuación 8 que implementa el softmax.
Agrupamos los parámetros $\theta = (W, b)$ de manera de simplificar la notación.
La función de costo asociada::

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K 1\{y^{(i)} = k\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)\top} x^{(i)})} \right] \quad (10)$$

donde m es la cantidad de imágenes en el set. $1\{\cdot\}$ es la función que devuelve 1 si lo que está en $\{\cdot\}$ es verdadero o 0 en caso contrario.

Calculando derivadas, se obtiene:

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^m \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right] = -\sum_{i=1}^m \left[x^{(i)} E^{(i)}(\theta; x^{(i)}, y^{(i)}) \right] \quad (11)$$

Aprendizaje: Stochastic Gradient Descend

El SGD es un método de entrenamiento basado en el descenso del gradiente a partir de un set limitado de ejemplos de entrenamiento. De esta manera posee dos ventajas frente a las metodologías de batch training: permite trabajar con bases de entrenamiento muy grandes, y, al mismo tiempo, acelera la convergencia.

Dado que el update de los parámetros de la red se realiza sobre un set pequeño (o incluso un solo ejemplo), podemos escribirlo:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)})$$

η , el ratio de aprendizaje, toma valores pequeños y en general disminuye a lo largo del entrenamiento, a cada epoch.

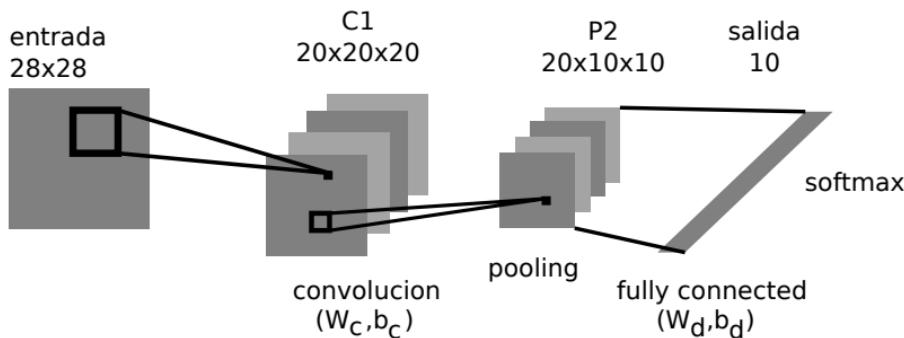
En las cercanías del mínimo, métodos como el SGD tienen a oscilar, retrasando la convergencia. Una metodología para acelerar el proceso es el llamado momentum. El update se define ahora como

$$\begin{aligned} v &= \gamma v + \eta \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)}) \\ \theta &= \theta - v \end{aligned}$$

donde v es la velocidad, del mismo largo que el θ , y γ determina la cantidad de iteraciones del gradiente son incorporadas al update.

EJERCICIO 2

Vamos a implementar un aprendizaje de una red CNN como muestra la figura

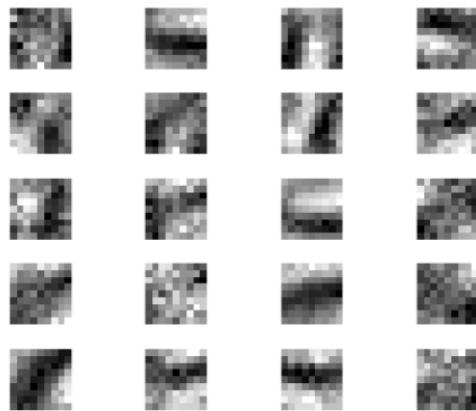


Los datos de entrada de la red son imágenes de dígitos (MNIST o LPR) en niveles de gris con un tamaño 28×28 pixeles. Los parámetros son los pesos y bias de la capa convolucional (W_c, b_c) y los pesos y bias de la capa densa (W_d, b_d). El script que se debe seguir es `cnnEj2.m`. Se debe completar el código en las funciones: `cnnConvolve`, `cnnPool`, `cnnCost`.

EJERCICIO 2

En la función `cnnCost` se calculan los gradientes de los parámetros de la red de la siguiente manera:

- ① Propagar las imágenes de entrada guardando en `probs` las salidas. Usar eq. 8.
- ② Calcular el costo usando eq. 10 y guardarlo en `cost`.
- ③ Calcular el error de salida del softmax usando $E^{(i)}(\theta; x^{(i)}, y^{(i)})$ de eq. 11.
- ④ $\Delta W_d = \frac{1}{m} \sum_i E^{(i)} * a^{(P2)}$, donde $a^{(P2)}$ es la activación de la capa de pooling.
- ⑤ $\Delta b_d = \frac{1}{m} \sum_i E^{(i)}$
- ⑥ Upsample de $\delta^{(P2)} = W_d * E^{(i)}$.
- ⑦ Calcular el $\delta^{(C1)} = \delta^{(P2)} * f'(z)$ (eq. 9).
- ⑧ $\Delta W_c = \frac{1}{m} \sum_i x^{(i)} * \text{rot90}(\delta^{(C1)})$, siendo $x^{(i)}$ la imagen de entrada.
- ⑨ $\Delta b_c = \frac{1}{m} \sum \delta^{(C1)}$



pnegri@dc.uba.ar