uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE
REDES DE NEURONAS ARTIFICIALES
Ricardo Aler

Introducción a Deep Learning

Línea Argumental

- Redes de neuronas con una capa oculta
- ¿Qué hace la capa oculta? Puede tener sentido poner más
- Pero si ponemos muchas capas, aparece el problema del vanishing gradient (la red no aprende)
- Soluciones:
  - Unsupervised pre-training
  - Relu: rectified linear units
  - Dropout: solución frente al sobreaprendizaje por la existencia de muchos parámetros (pesos) en el modelo
  - Convolutional networks (arquitecturas especializadas que limitan el número de pesos)

Alpha Go

- AlphaGo ha sido desarrollado por Google DeepMind. En Marzo de 2016, ganó a Lee Sedol (9-dan, el máximo).
- Combinación de aprendizaje automático y árbol de búsqueda
- Árbol de búsqueda Monte Carlo, guiada por una *value network* y una *policy network*, ambas son *deep neural networks*
- AlphaGo fue entrenado inicialmente para imitar el juego humano (*behavioural cloning* o *imitation learning*), tratando de igualar los movimientos de los jugadores expertos de juegos históricos almacenados (base de datos de alrededor de 30 millones de movimientos).
- Una vez que había alcanzado un cierto grado de habilidad, se autoentrenó jugando partidas contra sí mismo.
- Hardware: 1920 CPUs 280 GPUs

Resultados 2012

- Un ensemble de GPU-MPCNNs consiguió los mejores resultados en ImageNet (Krizhevsky et al., 2012).
- La red de neuronas más grande (10^9 pesos) se entrenó de manera no supervisada / supervisada y se aplicó a ImageNet. Consiguió los mejores resultados hasta ahora en 20000 clases.

Resultados 2013-

- Reconocimiento del habla (TIMIT) 17.7% (record), a pesar de la cantidad de investigación invertida en Hidden Markov Models hasta la fecha
- Optical Character Recognition, redes recurrentes superan a OCRs comerciales
- Reconocimiento de caracteres chinos (ICDAR, 3700 clases)
- Otros. Consultar [Schmidhuber 2014]

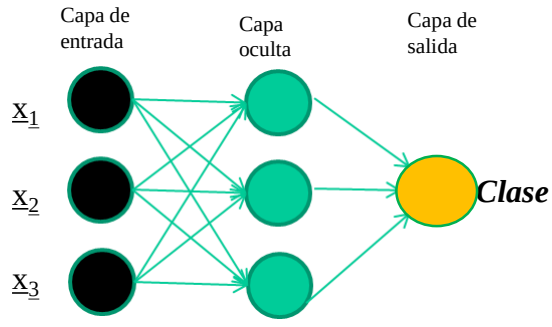
¿Qué es Deep Learning?

- En general, deep learning = deep neural networks. Redes de neuronas con muchas capas ocultas (habitualmente, sólo tienen una).

Redes de neuronas (con una capa oculta)

Datos

Atributos			***Clase***
x_1	x_2	x_3	y
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
…			

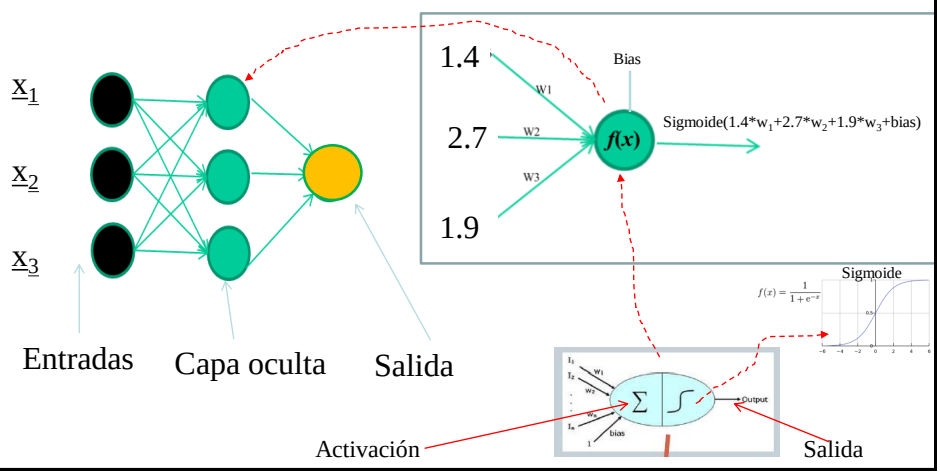


Redes de neuronas

Datos

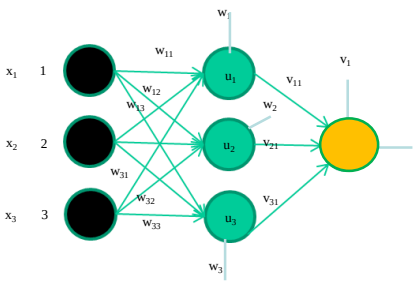
Atributos			***Clase***
x_1	x_2	x_3	
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
…			

- Con una capa oculta

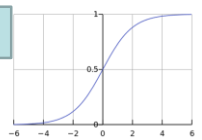


Redes de neuronas

- En último término, se trata de un modelo no lineal, con parámetros ajustables w_{ij}, b_i, v_{ij}, c_i llamados pesos.
- Las RRNN son aproximadores universales:
  - Pueden aproximar cualquier otra función
  - Siempre que haya suficientes neuronas ocultas



S(x)

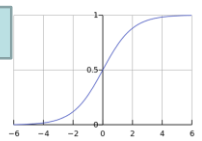


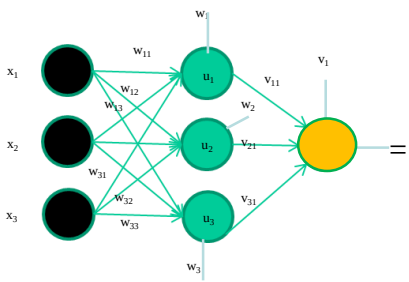
$$y = S(\ S(x_1*w_{11}+x_2*w_{21}+x_3*w_{31}+b_1)*v_{11} + S(x_1*w_{12}+x_2*w_{22}+x_3*w_{32}+b_1)*v_{21} + S(x_1*w_{13}+x_2*w_{23}+x_3*w_{33}+b_1)*v_{31}\ ) + c_1$$

Redes de neuronas

- En último término, se trata de un modelo no lineal, con parámetros ajustables w_{ij}, b_i, v_{ij}, c_i llamados pesos.
- Las RRNN son aproximadores universales:
  - Pueden aproximar cualquier otra función
  - Siempre que haya suficientes neuronas ocultas

S(x)





$$= S\left[S\left[\begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \right] \begin{pmatrix} v_{11} \\ v_{21} \\ v_{31} \end{pmatrix} + \begin{pmatrix} v_1 \end{pmatrix} \right]$$

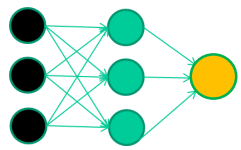
Redes de neuronas. Aprendizaje

- El aprendizaje de la red consiste en encontrar los pesos w_{ij}, w_i, v_{ij}, v_i que transformen correctamente la entrada en la salida
- Típicamente se utiliza el algoritmo de retropropagación del gradiente

$$E = \sum_j \frac{1}{2}(t_j - y_j)^2$$

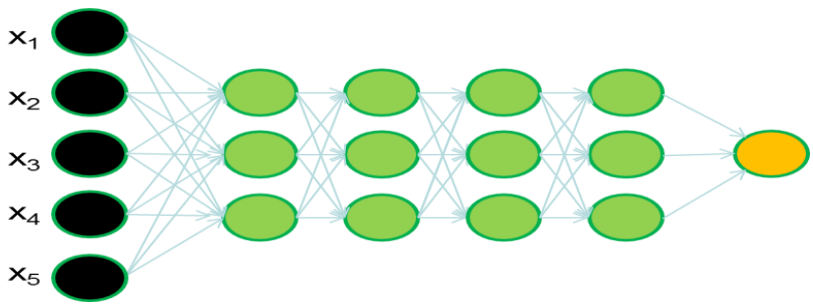
Datos

Atributos			***Clase***
x_1	x_2	x_3	
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
…			



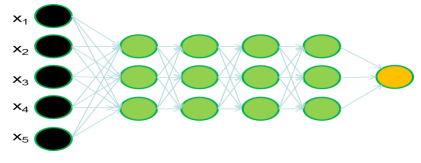
¿Qué es Deep-Learning?

- En general, deep learning = deep neural networks. Redes de neuronas con muchas capas ocultas ocultas.



¿Qué es Deep-Learning?

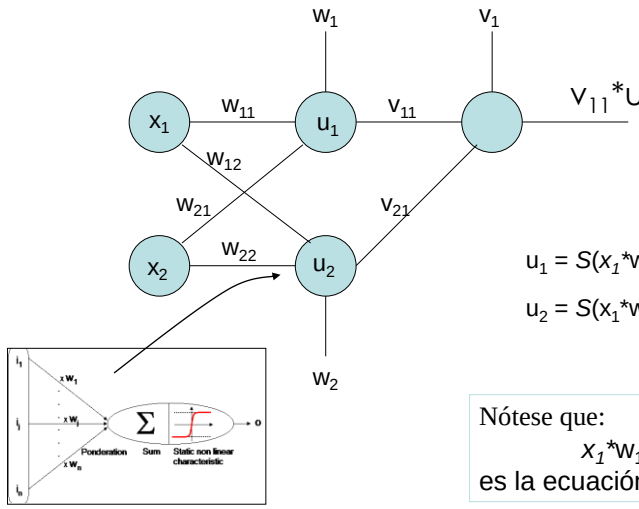
- En las siguientes transparencias, se motivará Deep Learning usando los siguientes argumentos:
  - Las RRNN de una capa clasifican en dos etapas. Puede ser interesante añadir más etapas.
  - Algunos problemas se pueden resolver de manera más sencilla usando más capas.
  - El cerebro parece hacer un procesamiento secuencial de la información.



Clasificación con RRNN de una capa: un proceso en dos etapas

- Las redes de neuronas intentan hacer clasificación transformando el espacio original a otro en el que los datos sean separables linealmente

Ejemplo de red



$V_{11}*U_1 + V_{21}*U_2 >= V_1$

$u_1 = S(x_1*w_{11} + x_2* w_{21} + w_1)$

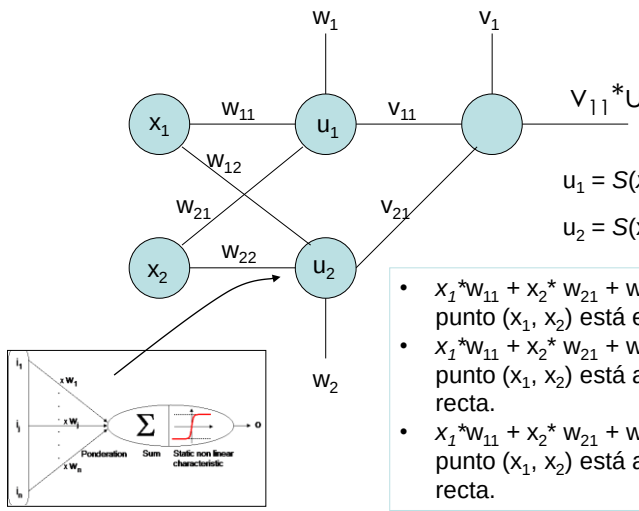
$u_2 = S(x_1*w_{12} + x_2*w_{22} + w_2)$

Nótese que:

$x_1*w_{11} + x_2* w_{21} + w_1=0$

es la ecuación de una recta

Ejemplo de red



$V_{11}*U_1 + V_{21}*U_2 >= V_1$

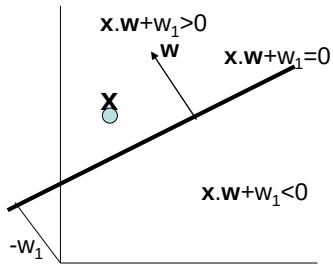
$u_1 = S(x_1*w_{11} + x_2* w_{21} + w_1)$

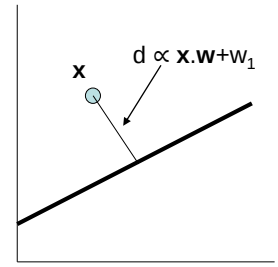
$u_2 = S(x_1*w_{12} + x_2*w_{22} + w_2)$

- $x_1*w_{11} + x_2* w_{21} + w_1 = 0$ implica que el punto (x_1, x_2) está en la recta.
- $x_1*w_{11} + x_2* w_{21} + w_1 > 0$ implica que el punto (x_1, x_2) está a la izquierda de la recta.
- $x_1*w_{11} + x_2* w_{21} + w_1 < 0$ implica que el punto (x_1, x_2) está a la derecha de la recta.

Recordatorio: distancia de un punto a una recta

$$x_1 * w_{11} + x_2 * w_{21} + w_1 = \mathbf{x}.\mathbf{w} + w_1 = 0$$



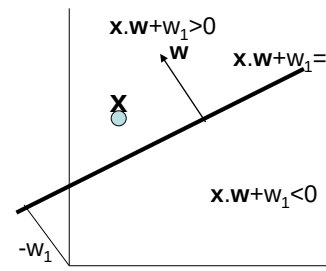


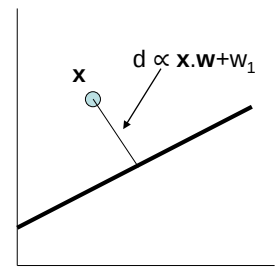
$\mathbf{x}=(x_1, x_2)$, $\mathbf{w}=(w_{11}, w_{21})$

Se puede demostrar que la distancia del punto a la recta es proporcional a $\mathbf{x}.\mathbf{w}+w_1$

Recordatorio: distancia de un punto a una recta

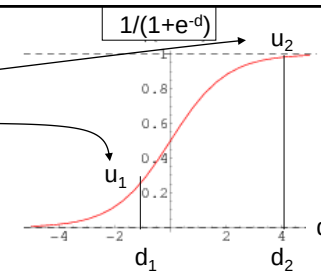
- Sea la ecuación de una recta en 2D:
  - $w_{11}*x_1+w_{21}*x_2+w_1 = 0$
  - $w_{11}*x+w_{21}*y+w_1 = 0$
  - O, expresada mediante el producto escalar del vector $\mathbf{x}=(x_1, x_2)$ y el vector de pesos $\mathbf{w}=(w_{11}, w_{21})$:
  - $\mathbf{x}.\mathbf{w}= -w_1$
- Todos aquellos puntos (x_1,x_2) que están en la misma dirección que **w** cumplen $\mathbf{x}.\mathbf{w}+w_1>0$
- Todos aquellos puntos (x_1,x_2) que están en la dirección contraria que **w** cumplen $\mathbf{x}.\mathbf{w}+w_1<0$
- Todos aquellos puntos (x_1,x_2) que están a la misma distancia de la recta cumplen (para alguna constante k) $\mathbf{x}.\mathbf{w}+w_1=k$
- Se puede demostrar que $\mathbf{x}.\mathbf{w}+w_1 = d*|w| \propto d$
  - O sea, que es proporcional a la distancia d del punto **x** a la recta (no es exactamente la distancia, pero si que es proporcional a ella)
- Si $|w|=1$, entonces $\mathbf{x}.\mathbf{w}+w_1$ si que es exactamente la distancia del punto a la recta

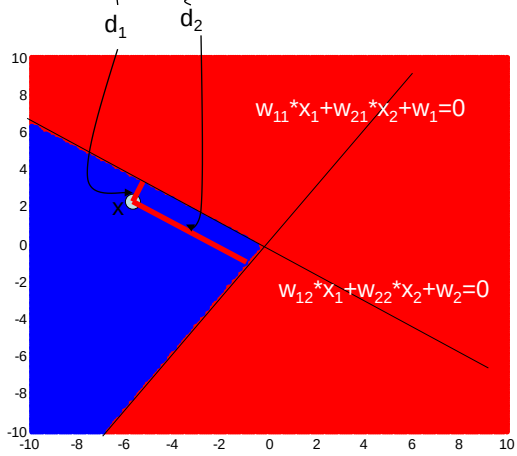




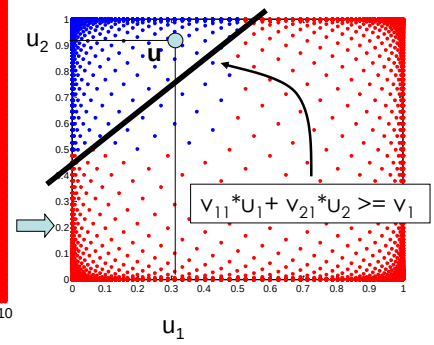
$u_1 = S(d(\mathbf{x}, \text{recta}(w_{11}, w_{21}, w_1)$

$u_2 = S(d(\mathbf{x}, \text{recta}(w_{12}, w_{22}, w_2)$

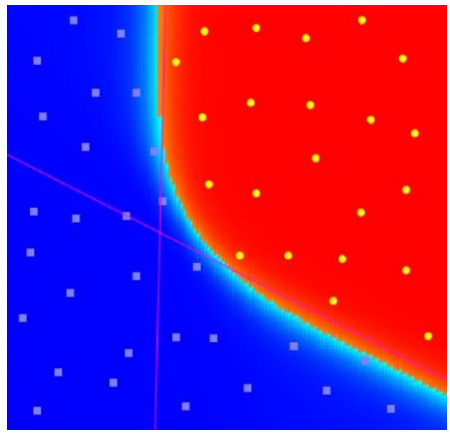




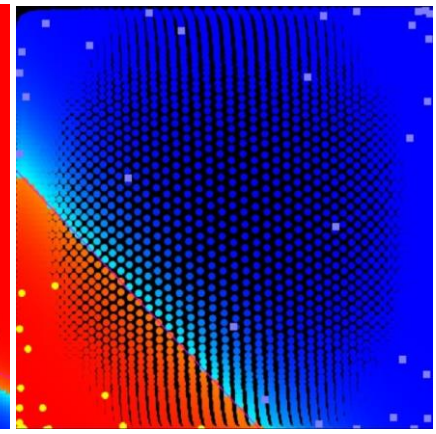
En el espacio (x_1,x_2) no es posible la separación lineal.



En (u_1,u_2), lineal es (casi) posible.



Espacio original (x1,x2)



Espacio transformado por las neuronas ocultas (u1,u2)

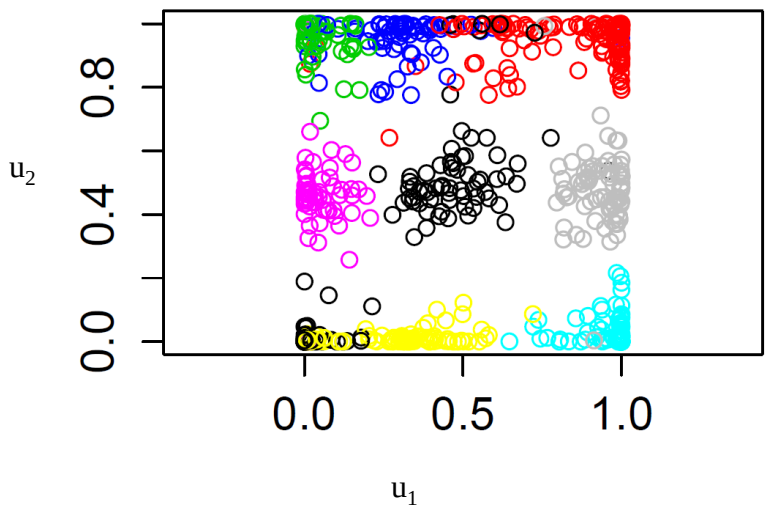
SEMEION dataset

- 1593 dígitos escritos por 80 personas
- Matrices de 16x16 pixeles en blanco y negro
- Dígitos de 0 a 9
- Supongamos una red de neuronas con 16*16=256 entradas,y dos neuronas ocultas.

Figure 1: written in normal way

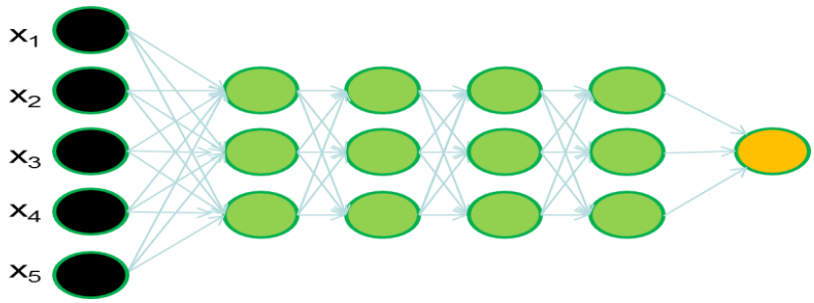
Figure 2: written in fast way

- Representación interna de SEMEION con las dos neuronas de la capa oculta. Cada color son los datos de un dígito distinto (0, 1, 2, …, 9)



¿Y si añadimos más etapas?

Si la clasificación es un proceso en dos etapas, puede tener sentido añadir más etapas para hacer un procesamiento más elaborado.

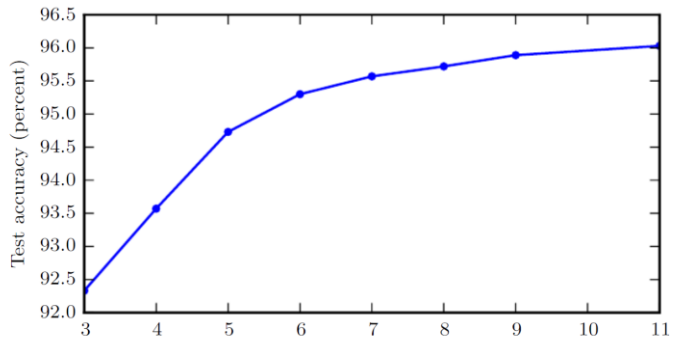


¿Por qué son interesantes múltiples capas?

- Hay un teorema que demuestra que cualquier problema se puede aproximar con una sola capa oculta, siempre que haya suficientes neuronas en esa capa
- ¿Por qué introducir más capas (deep learning)?
- Idea teórica: algunos problemas se expresan de manera más sencilla con múltiples capas (con una sóla capa, harían falta muchísimas neuronas en la capa oculta)
- Idea empírica: parte del procesamiento del cerebro funciona de manera secuencial, de lo más sencillo a lo más abstracto

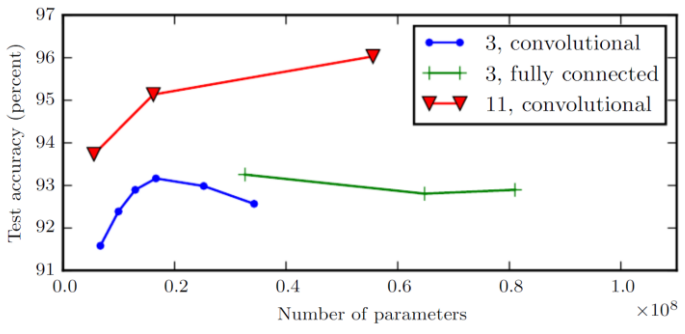
Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2014). Multi-digit number recognition from Street View imagery using deep convolutional neural networks. In *International Conference on Learning Representations*

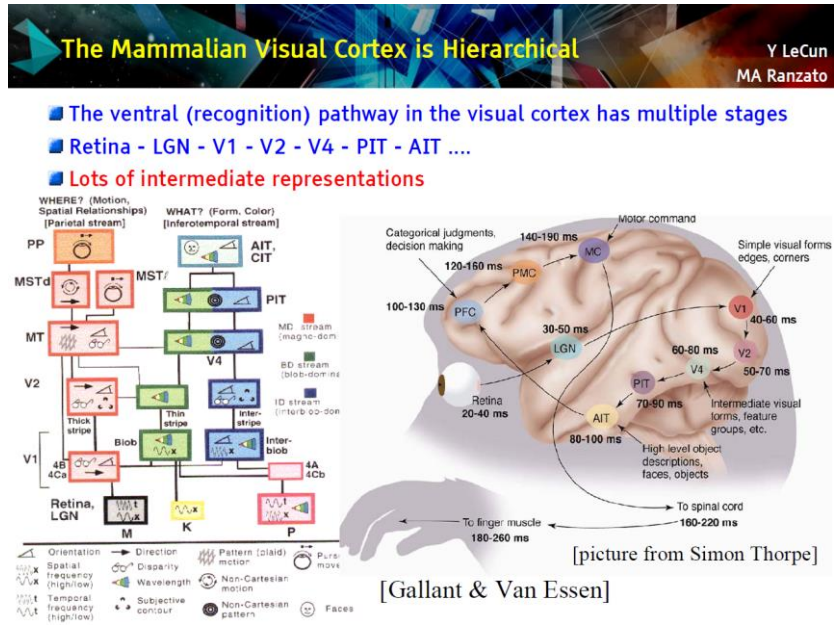
- El número de capas mejora la capacidad de generalización



Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2014). Multi-digit number recognition from Street View imagery using deep convolutional neural networks. In *International Conference on Learning Representations*

- Tener más parámetros es bueno si la profundidad de la red es mayor (11 vs. 3)





[Gallant & Van Essen]

Jerarquía de características

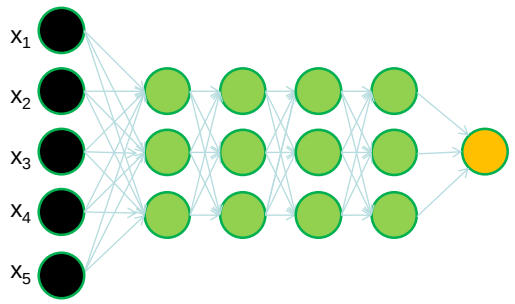
- Reconocimiento de imágenes:
  - Píxel => arista => parte => objeto
- Reconocimiento de textos:
  - Carácter => palabra => grupo => oración => historia
- Reconocimiento del habla:
  - Muestra => banda de frecuencia => sonido => fonema => palabra

El problema del vanishing gradient

$w(t+1) = w(t) - \alpha\ \mathbf{dE/dw}$; $\Delta w = -\ \alpha\ \mathbf{dE/dw}$

$\Delta w = -\ \alpha\ dE/dv_{11} = \alpha * (t_j - y_j) * S'$(activación de s_1) * entrada de s_1

- El algoritmo de backpropagation funciona bien con una sola capa oculta, pero no con muchas.



MNIST

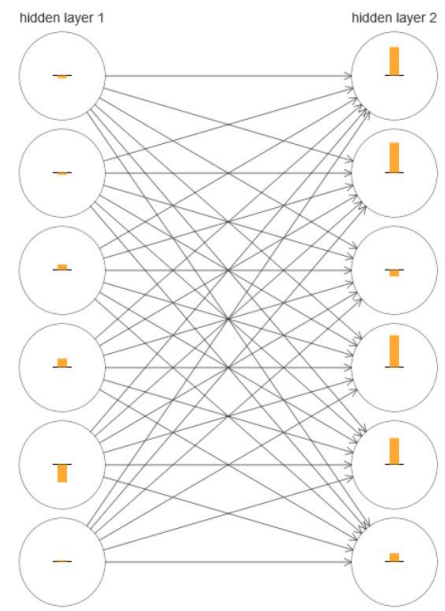
- Reconocimiento de caracteres. Versión complicada de SEMEION (28X28 pixeles)

Planteamiento

- Supongamos una red con 784 neuronas (28x28 pixels) de entrada y 10 neuronas de salida (dígitos '0', '1', '2', …, '9‘)
- 1 capa oculta con 30 neuronas: 96.48%
- 2 capas ocultas con 30 neuronas: 96.90%
- 3 capas: 96.57%
- 4 capas: 96.53

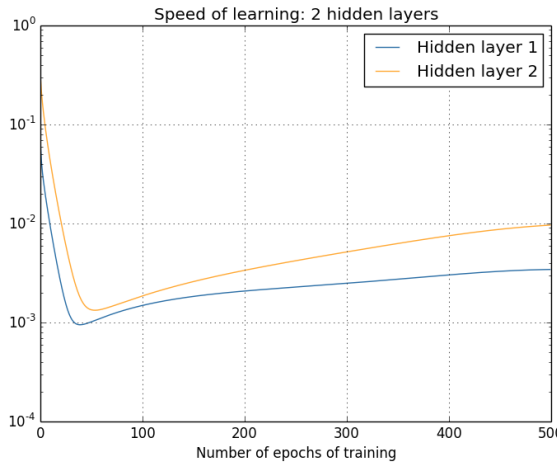
- 6 neuronas de la red con 2 capas ocultas
- La barra naranja representa el gradiente, que es la velocidad con la que cambian los pesos (velocidad de aprendizaje)

- Velocidades de aprendizaje en las capas:
  - $\|\delta_1\|=0.07$
  - $\|\delta_2\|=0.31$
- La primera capa aprende mucho más lento que la segunda
- Con 4 capas: 0.003, 0.017, 0.070, 0.285



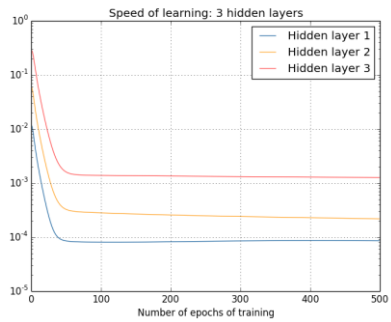
http://neuralnetworksanddeeplearning.com/chap5.html

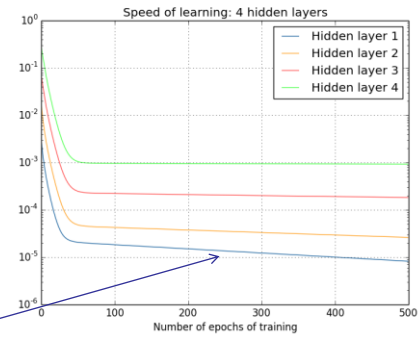
Cómo cambia la velocidad de aprendizaje con el número de iteraciones



http://neuralnetworksanddeeplearning.com/chap5.html

Cómo cambia la velocidad de aprendizaje con el número de iteraciones





- La primera capa aprende 100 veces más lento que la cuarta. No se está aprendiendo bien, y hay que tener en cuenta que de la salida de la primera capa depende todo lo demás.

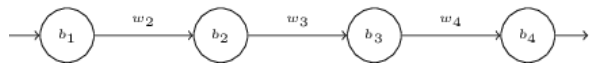
http://neuralnetworksanddeeplearning.com/chap5.html

Caso sencillo de la regla de la cadena

Recordar: $\Delta w_4 = - \alpha\, dE/dw_4 = - \alpha * (t_j - y_j) * S'$(activación de b_4) * salida de b_3

Regla de la cadena:

$dE/dw_2 \propto S'(\ldots) * S'(\ldots) * S'(\ldots) * \ldots$



- La derivada de la sigmoide es $S'(x) = S(x)*(1-S(x))$
- Su máximo es 0.25, en el resto de los casos es menor
- Multiplicar un número menor que uno repetidas veces resulta en valores muy pequeños

Entrenamiento de Deep Networks

- Difficultades
  - Las primeras capas de las redes no se entrenan bien
    - Vanishing Gradient: entrenamiento muy lento
    - Las últimas capas (cercanas a la salida) aprenden rápido, pero se basan en la salida de las primeras capas, que prácticamente no pueden ser entrenadas (o muy lentamente)
    - Hay que conseguir que las primeras capas hagan algo útil
  - ¿Tal vez se puedan inicializar los pesos de las capas de una manera no supervisada?

36

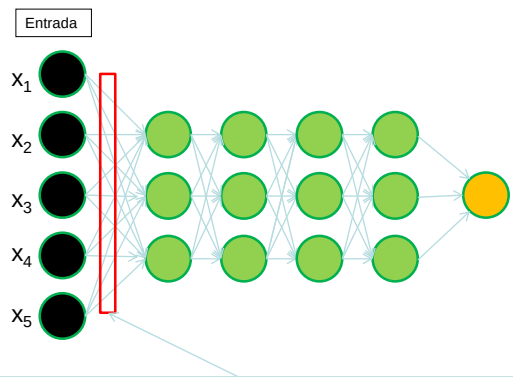
Entrenamiento por capas

(unsupervised pre-training)

1. Entrena la primera capa de manera no supervisada (datos sin etiquetas / clases)
   - En la primera capa no se conoce aún la salida
   - Podemos utilizar datos sin etiqueta (si disponemos de ellos)
2. "Congela" la primera capa y comienza a aprender los pesos de la segunda de la misma manera no supervisada
3. Repetir para tantos niveles como queramos
   - En cierto sentido, estas capas están "reformulando" el problema
4. La última capa, se entrena de manera supervisada (ahí ya se conoce la salida)
5. "Descongela" todos los pesos y haz un ajuste fino de los pesos con backpropagation.

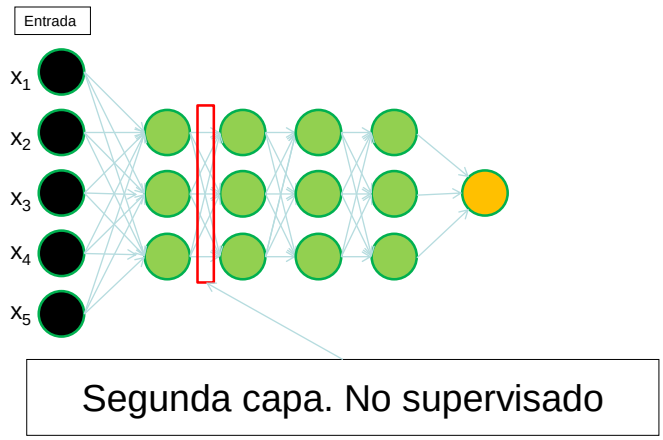
37

Entrenamiento por capas



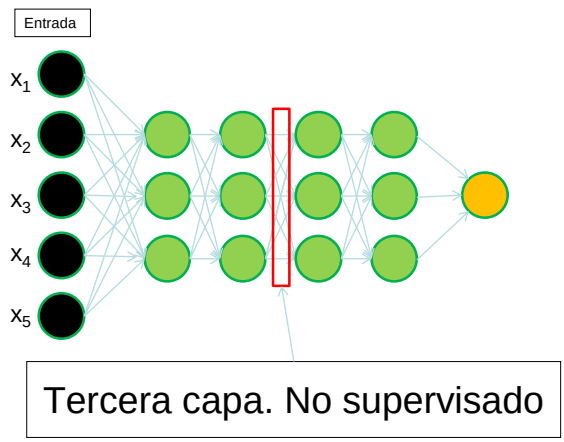
Primera capa. No supervisado

Entrenamiento por capas



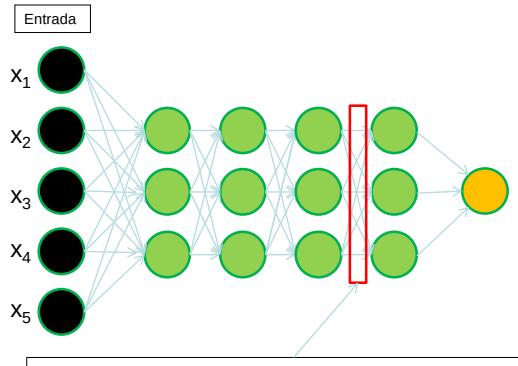
Segunda capa. No supervisado

Entrenamiento por capas



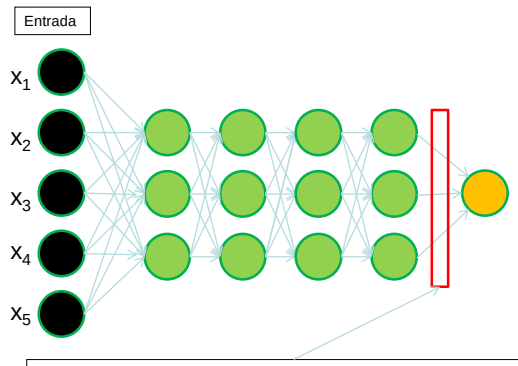
Tercera capa. No supervisado

Entrenamiento por capas



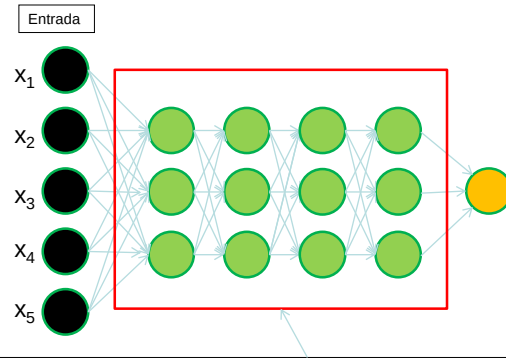
Cuarta capa. No supervisado

Entrenamiento por capas



Quinta capa. **Supervisado**

Entrenamiento por capas



Ajuste fino final completo **Supervisado**

Entrenamiento por capas (unsupervised pre-training)

1. Entrena la primera capa de manera no supervisada (datos sin etiquetas / clases)
   - En la primera capa no se conoce aún la salida
   - Podemos utilizar datos sin etiqueta (si disponemos de ellos)
2. "Congela" la primera capa y comienza a aprender los pesos de la segunda de la misma manera no supervisada
3. Repetir para tantos niveles como queramos
   - En cierto sentido, estas capas están "reformulando" el problema
4. La última capa, se entrena de manera supervisada (ahí ya se conoce la salida)
5. "Descongela" todos los pesos y haz un ajuste fino de los pesos con backpropagation.

Entrenamiento por capas

(unsupervised pre-training)

- A fast learning algorithm for deep belief nets. GE Hinton, S Osindero, YW Teh - Neural computation, 2006
- Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning?. *Journal of Machine Learning Research*, *11*(Feb), 625-660.
- Y Bengio, P Lamblin, D Popovici, H Larochelle. (2007). Greedy layer-wise training of deep networks. Advances in neural information processing systems

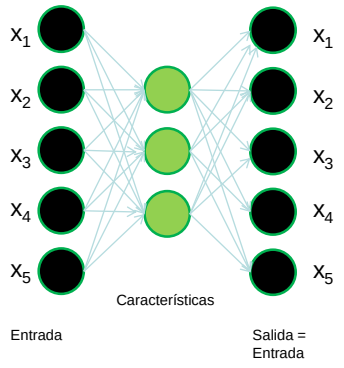
Entrenamiento por capas

- Todas las capas se aprenden a la misma velocidad puesto que en cada etapa, es la única capa siendo entrenada
- Se pueden utilizar datos sin etiquetar
- Cuando se hace el ajuste fino final con backpropagation, se supone que todos los pesos están ya en la cercanía de una buena solución
- ¿Cómo hacer el aprendizaje no supervisado?
  - Stacked Auto-Encoders
  - Deep Belief Networks (Stacked Restricted Boltzman Machines o RBMs)

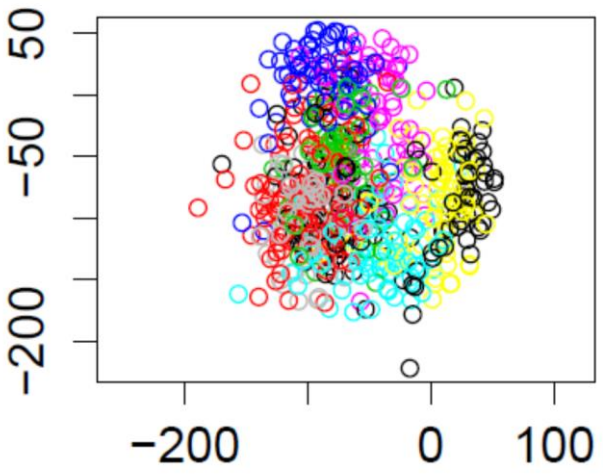
46

Autoencoders

- La salida es la propia entrada
- Intenta encontrar buenos nuevos atributos que permitan recodificar los atributos originales
- Al haber menos neuronas ocultas que neuronas de entrada, le forzamos a que las neuronas ocultas contengan información de más alto nivel
- También se puede hacer con denoising auto-encoders: añaden ruido a la entrada, pero intentan regenerar el dato sin ruido

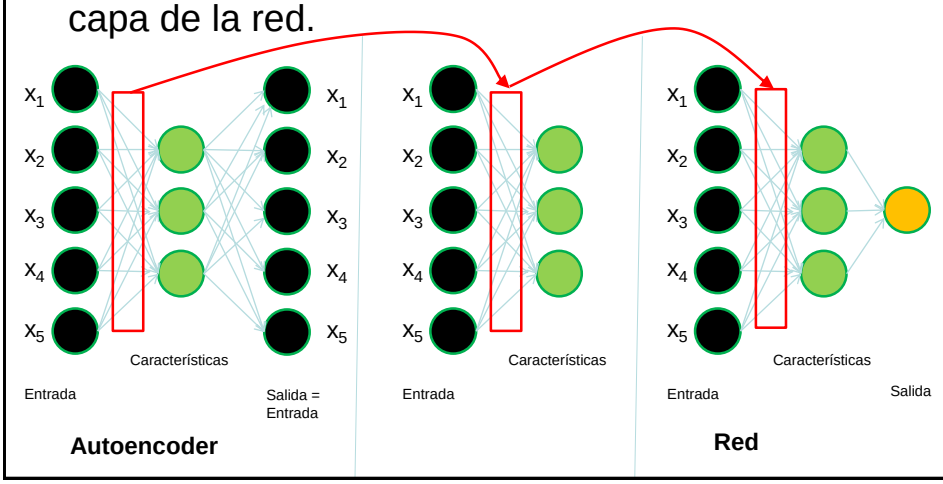


Auto-encoder para SEMEION con dos neuronas ocultas



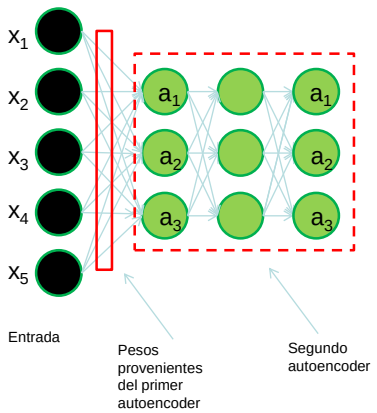
Autoencoders

- Podemos usar los pesos de la capa de entrada del autoencoder para inicializar los pesos de la primera capa de la red.



Stacked Autoencoders

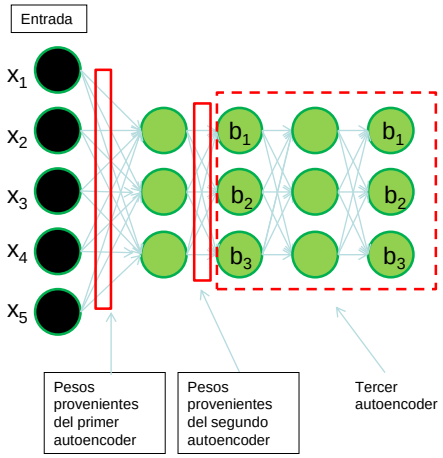
- Bengio (2007, 2006)
- Apila de manera secuencial auto-encoders
- Elimina la capa de decodificación en cada etapa



50

Stacked Autoencoders

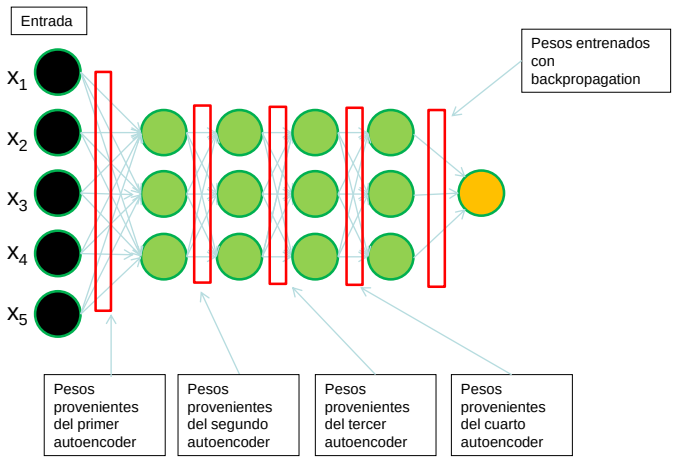
- Bengio (2007, 2006)
- Apila de manera secuencial auto-encoders
- Elimina la capa de decodificación en cada etapa



51

Stacked Autoencoders

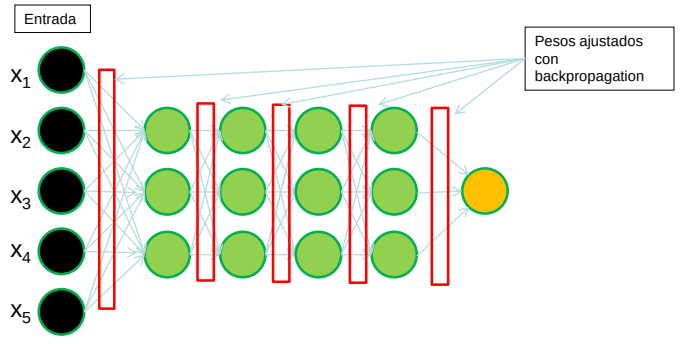
- Bengio (2007, 2006)
- Apila de manera secuencial auto-encoders
- Elimina la capa de decodificación en cada etapa



52

Stacked Autoencoders

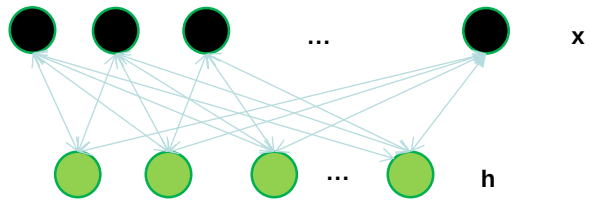
- Bengio (2007, 2006)
- Ajuste final con backpropagation



53

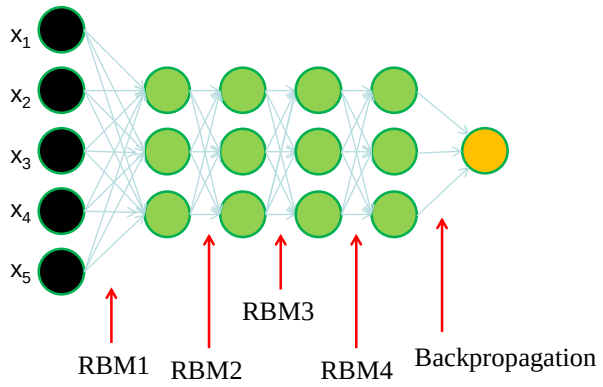
Deep Belief Networks (DBN)

- Geoff Hinton (2006)
- Exactamente la misma idea que stacked autoencoders, pero en lugar de autoencoders, usa RBMs (Restricted Boltzmann Machine)
- Nodos hidden (**h**) / visible (**x**)
  - Pesos simétricos



54

Entrenamiento por capas para Deep Belief Networks (DBN)



Entrenamiento RBM

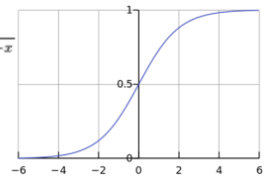
- Estado inicial de **x** es un dato
- El muestreo es un proceso iterativo:
  - $\mathbf{x} \rightarrow \mathbf{h} \rightarrow \mathbf{x} \rightarrow \mathbf{h} \rightarrow \ldots$
  - $P(h_i = 1|\mathbf{x}) = \text{sigmoid}(W_i\mathbf{x} + c_i) = 1/(1+e^{-\text{net}(h_i)})$
  - $P(x_i = 1|\mathbf{h}) = \text{sigmoid}(W'_i\mathbf{h} + b_i) = 1/(1+e^{-\text{net}(x_i)})$
- Se actualizan los pesos para minimizar la Contrastive Divergence (CD-*k*): diferencia entre el dato original y el dato reconstruido después de k iteraciones
- Suele ser suficiente con CD-1

RELU (Rectified linear units)

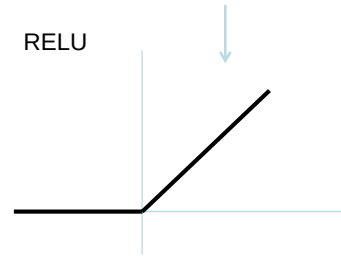
- Se usan en las neuronas ocultas en lugar de la sigmoide: y = max(0, x)

Sigmoide

$$f(x) = \frac{1}{1+e^{-x}}$$



RELU



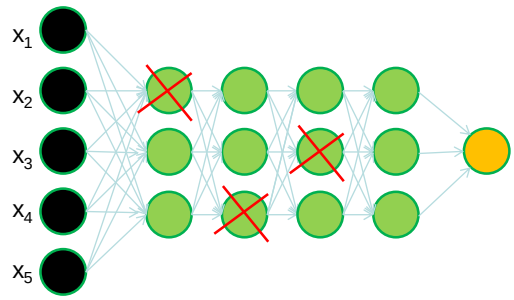
- El gradiente no se desvanece, al menos en la parte derecha (la pendiente de la recta es uno)

Dropout

- Las deep networks tienen muchos parámetros
- Cuanto más parámetros tiene un modelo, más complejo puede ser y más fácil es que sobreajuste a los datos de entrenamiento
- Una manera sencilla de obligar a que el modelo sea sencillo es dropout

Dropout

- Durante el entrenamiento (cada iteración): cada neurona se mantiene con probabilidad p (y se elimina con probabilidad 1-p)
- Durante el test: se usa toda la red, pero los pesos se multiplican por p
- Esto obliga a las neuronas a ser “robustas” puesto que no pueden asumir que siempre van a disponer de todas sus entradas. Tienen que generalizar.



Redes de neuronas convolucionales (CNN). Introducción

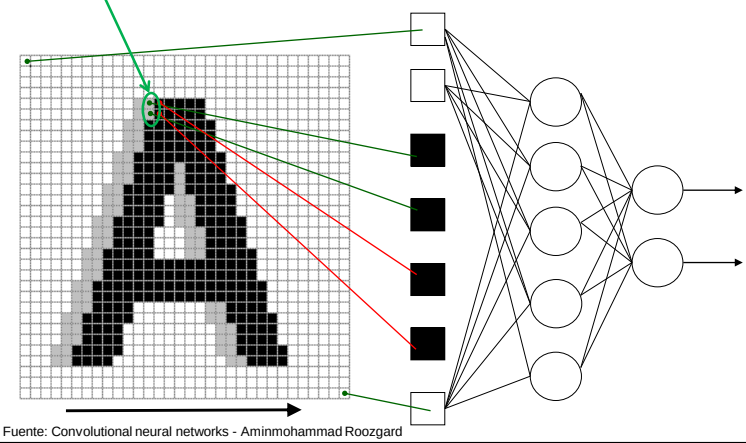
- Especializadas en el reconocimiento de imágenes o caracteres.
- Limitaciones en las RRNN estándar para este problema:
  - El número de pesos se incrementa enormemente.
  - No tienen invarianza translacional.
  - No tienen en cuenta propiedades del dominio de reconocimiento de imágenes.

Redes CNN. Limitaciones de las RRNN estándar

- Con RRNN estándar, el número de pesos se incrementa enormemente.
- Supongamos un problema de clasificación de letras en 16x16 píxeles en blanco y negro con 100 neuronas en la capa oculta. Se requerirían 256*(100+1) pesos en la capa de entrada, y 100*(26+1) en la capa de salida

Limitaciones de las RRNN estándar

- Las RRNN estándar no tienen invarianza translacional.
- En el siguiente gráfico, si la red está preparada para reconocer dos pixeles en la parte superior izquierda de la A, y movemos la A ligeramente hacia la derecha, los píxeles negros se convierten en blancos y la red ya no reconoce el carácter.



Fuente: Convolutional neural networks - Aminmohammad Roozgard

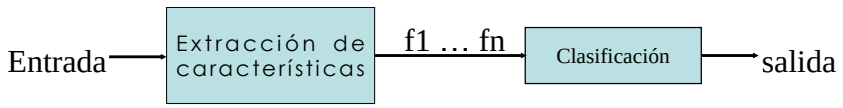
Limitaciones de las RRNN estándar

- No tienen en cuenta propiedades típicas del dominio (reconocimiento de caracteres).
- Por ejemplo, muchas características (segmentos, esquinas, …) son locales: las relaciones entre los píxeles contiguos en la imagen son más altas que entre los píxeles lejanos). Pero una red estándar aprende relaciones entre los píxeles independientemente de su distancia.

63

Limitaciones de las RRNN estándar

- Debido a esto, cuando se usan RNN estándar, no se pone en la entrada la imagen en bruto, sino que se extraen características (con algoritmos programados ad-hoc) y después se usa la red para clasificación, con esas características.

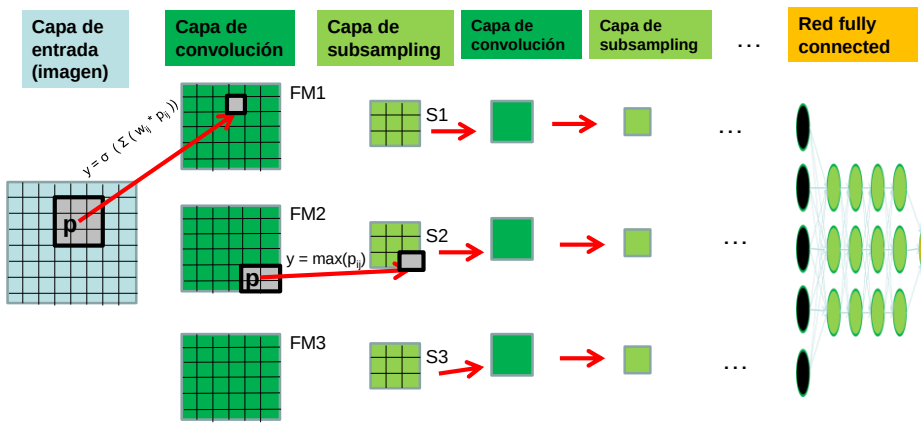


- Con las redes convolucionales se quiere hacer la clasificación a partir de la imagen en bruto (los píxeles de la imagen).

Redes CNN.

- Yann LeCun, Yoshua Bengio, 1995
- Están motivadas por la neurología (neuronas sensitivas a características locales de la imagen)
- Son redes de neuronas multi-capa, pero resuelven el problema del vanishing gradient reduciendo el número de pesos
- Usan backpropagation

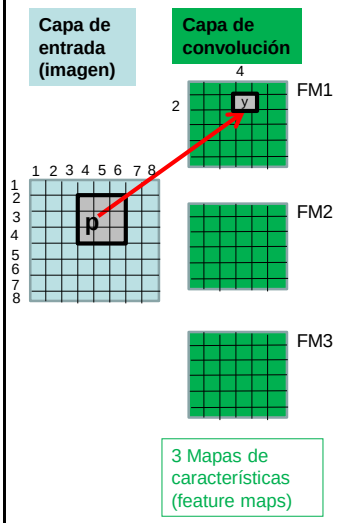
Arquitectura de redes CNN.



3 Mapas de características (feature maps)

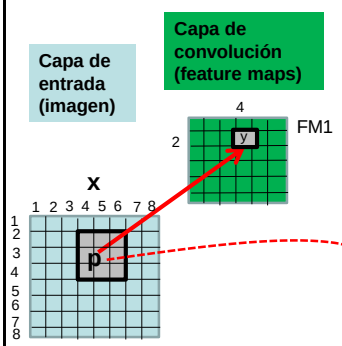
Alterna capas de convolución y de subsampling, terminando en una rede de neuronas estándar.

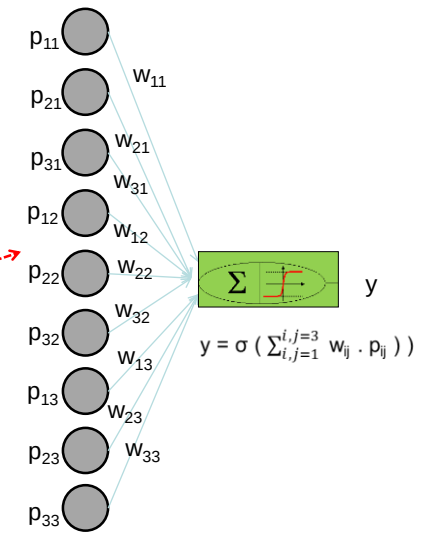
Capa de convolución



- Supongamos una imagen de 8x8 píxeles (como en el gráfico de la izquierda).
- Por ejemplo, para generar y_{24} en el primer feature map de la capa de convolución, un filtro de 3x3 (en gris) toma de la entrada los 9 píxeles cuya esquina superior izquierda es x_{24} y cuya esquina inferior derecha es $x_{(2+3-1)(4+3-1)}$
- Y les aplica la transformación
  - $y = \sigma\left(\sum_{i,j=1}^{i,j=3} w_{ij} \cdot p_{ij} \right)$
  - donde p_{ij} son los nueve píxeles de la región 3x3, w_{ij} son los 9 pesos del filtro y σ es una función de activación (Sigmoide, RELU, …)

Filtro de convolución

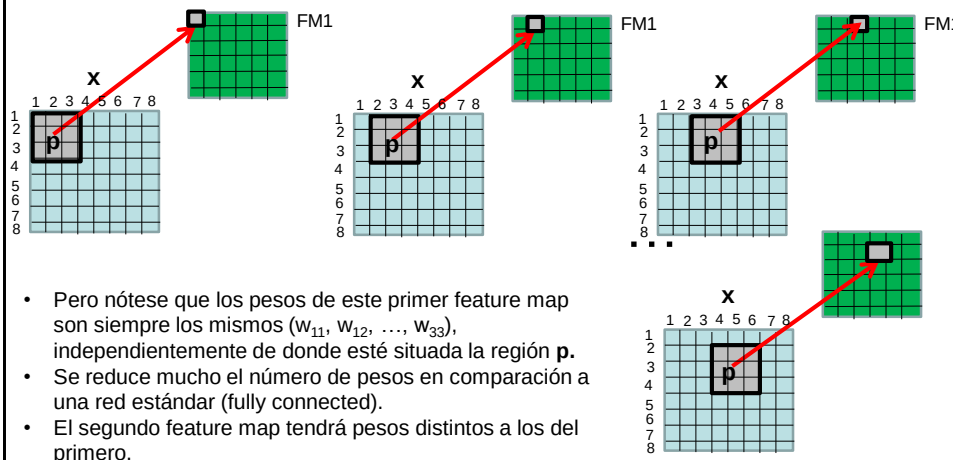




- En el fondo, el filtro es la aplicación de una única neurona a la región de 3x3 (**p**).
- A la región p se la llama campo receptivo.

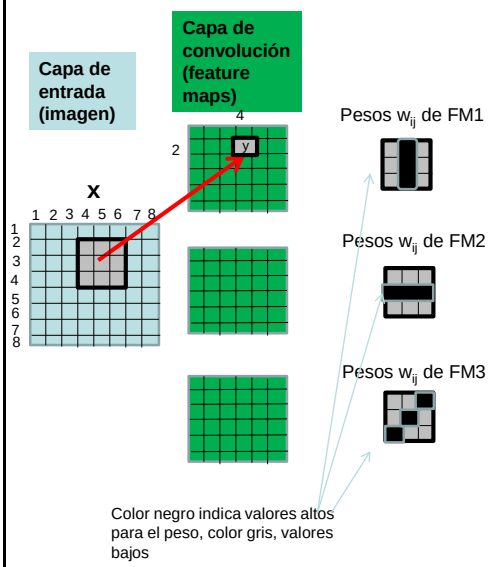
Filtro de convolución

- El filtro se va moviendo por la imagen para generar todos los valores del primer feature map.



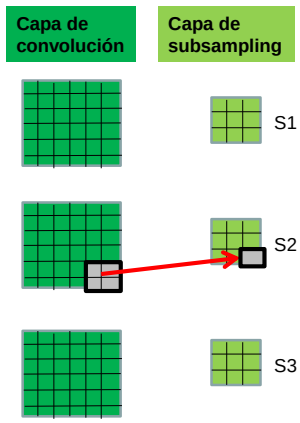
- Pero nótese que los pesos de este primer feature map son siempre los mismos (w_{11}, w_{12}, …, w_{33}), independientemente de donde esté situada la región **p.**
- Se reduce mucho el número de pesos en comparación a una red estándar (fully connected).
- El segundo feature map tendrá pesos distintos a los del primero.

Capa de convolución

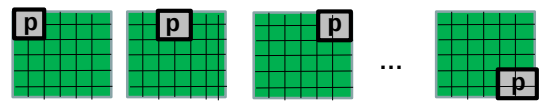


- Cada feature map detecta una característica distinta (por eso cada FM tiene pesos distintos).
- Por ejemplo, FM1 podría detectar segmentos verticales, mientras que FM2 podría detectar horizontales. FM3 tal vez segmentos oblicuos.
- Así, que y_{24} de FM1 esté activo indica que se ha detectado un segmento vertical en la zona x_{24} … x_{46} de la imagen.
- Nótese cada feature map detecta características invariantes a la translación, puesto que se detectan allí donde estén en la imagen.

Capa de subsampling o pooling

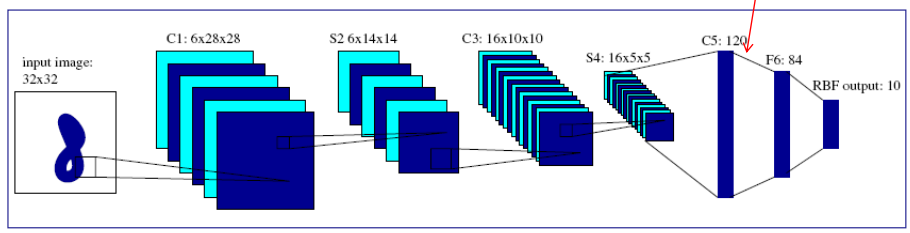


- El objetivo de la capa de subsampling es reducir la resolución espacial.
- Por ejemplo, en la figura de la izquierda, se hace una compresión 4 a 1.
- Realiza la transformación y=max(w **p**) (max-pooling) o y=mean(**p**).
- Detecta característica en regiones más grandes de la imagen. Por ejemplo, max(p_{11}, p_{12}, p_{21}, p_{22}) detecta que la característica existe en algún punto de la región 2x2 del mapa de convolución.
- Los campos receptivos **p** no solapan (a diferencia de la capa de convolución)



Entrenamiento de CNNs

- Se pueden entrenar con back-propagation, debido a que la compartición de pesos para cada feature-map reduce enormemente el número de pesos. El problema del vanishing gradient y sobreaprendizaje es menos grave
- Por ejemplo, un feature-map con un campo receptivo de 5x5 tiene un total de 26 pesos
- La última capa es una capa (o varias) de red de neuronas estándar (fully connected)
- La arquitectura de una CNN se suele diseñar a mano
  - Típicamente, el campo receptivo de los feature-maps es grande en las primeras capas, pero hay pocos feature-maps, y se decrementa en las capas del final, pero con un menor número de feature-maps
- El siguiente gráfico corresponde a la arquitectura LeNet5, con 187000 conexiones pero sólo 14000 pesos diferentes, que se puede encontrar en:
  - http://yann.lecun.com/exdb/lenet/index.html



Conclusiones

- Backpropagation sufre de vanishing gradient cuando se utilizan muchas capas ocultas (se considera “very deep learning” cuando el número de capas es mayor de 10)
- Una solución es la inicialización no supervisada de las capas intermedias, mediante auto-encoders o RBM, y el ajuste fino con backpropagation después.
- Substituir las activaciones sigmoidales por RELUs puede ayudar a combatir el vanishing gradient.
- Otra solución es usar redes de convolución, que reducen enormemente el número de pesos a ajustar (aunque son redes especializadas en imágenes).
- Una última solución es seguir usando backpropagation, pero incrementar enormemente la potencia de cómputo mediante GPUs, para compensar .
- Deep Learning NN obtienen los mejores resultados en problemas de reconocimiento de imágenes y sonido, una vez superado el problema del vanishing gadient.

Bibliografía

- Schmidhuber, J. (2014). Deep Learning in Neural Networks: An Overview.
  - http://arxiv.org/abs/1404.7828
- Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, *2*(1), 1-127.
- Goodfellow, Bengio, Courville. Deep Learning. MIT press. 2016. http://www.deeplearningbook.org/