



deeplearning.ai

Introduction to Deep Learning

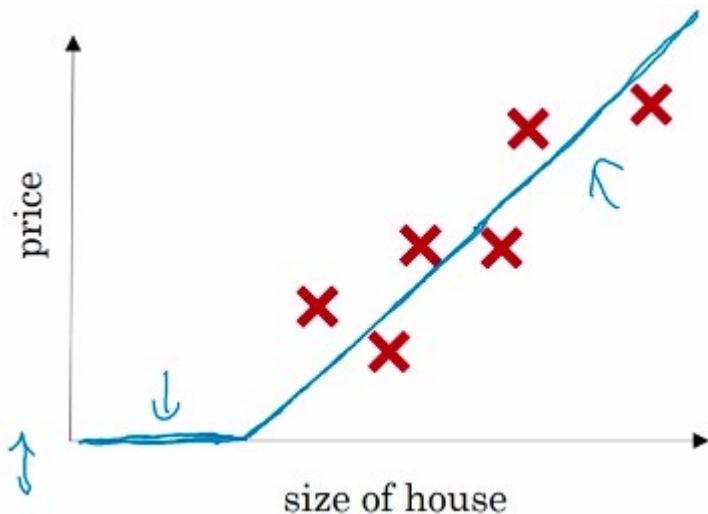
Created by [Borghi, Matias](#)
Based on [Andrew Ng's Deep Learning Course](#).

¿Qué se verá en la Especialización?

1. Redes Neuronales y Deep Learning, los fundamentos. ¿Cómo entrenarlos con datos? Construir una red neuronal para distinguir gatos.
2. ¿Cómo hacer que la red neuronal funcione correctamente?: tuneado de hiper parámetros, regularización y optimización.
3. ¿Cómo estructurar un proyecto de Machine Learning? Por ejemplo, la manera de dividir los datos para testeo, cross validacion y entrenamiento ha cambiado en la era del Deep Learning.
4. Hablar acerca de CNN (Redes Neuronales Convolucionales). Generalmente aplicado a imágenes.
5. NLP (Natural Language Processing): Construir modelos secuenciales. Usando RNN (Recurrent Neural Networks) o LSTM (Long Short Term Memory).

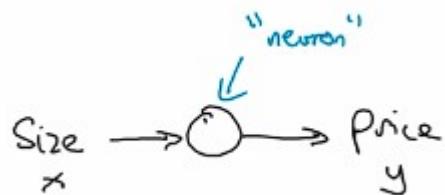
¿Qué es una red neuronal?

Predicción aplicado al precio de las casas.



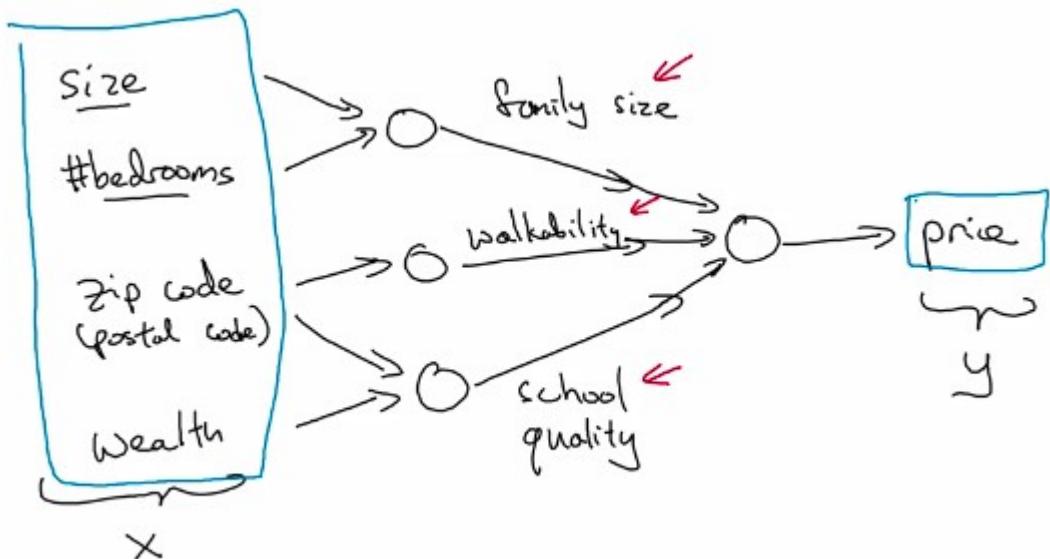
Los datos parecen ajustarse con un ajuste lineal. Dado que los precios no pueden ser negativos podemos agregar una sección donde el ajuste sería cero.

A esta función que predice los precios se la puede pensar como una red neuronal simple.



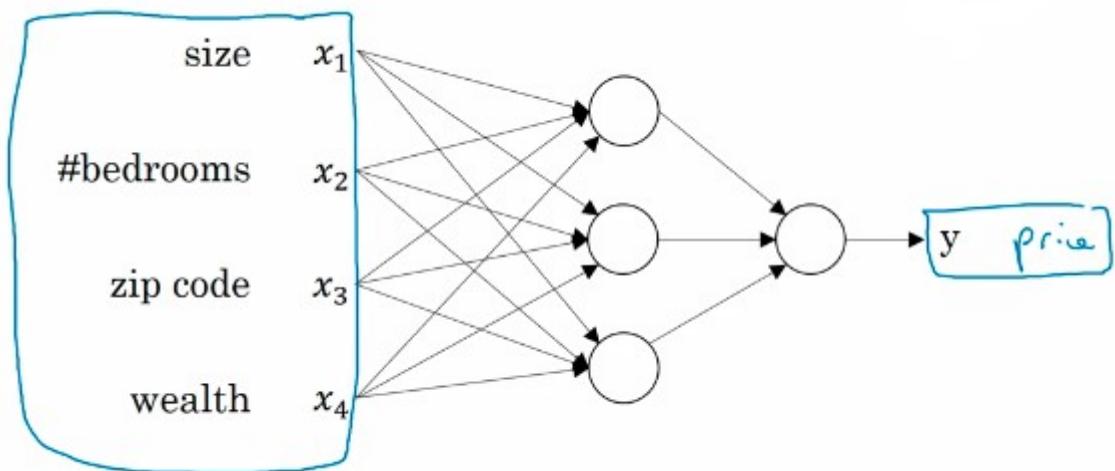
Este tipo de funciones que es cero y luego lineal es llamada RELU (Rectified Linear Unit). Una red neuronal más compleja se podría armar uniendo "neuronas" simples como la recién creada.

En lugar de crear una red neuronal que predice el precio de una casa en base al tamaño, también lo hace en función de otras características (features) como el número de habitaciones. También el tamaño de la familia (4 integrantes, 5 integrantes etc.).



Lo bueno es que, en un modelo de redes neuronales, no necesitamos conocer las características intermedias, sino que basta con determinar x e y .

En la realidad la red neuronal se implementa de la siguiente manera:



donde las unidades ocultas se conectan con todas las unidades precedentes.

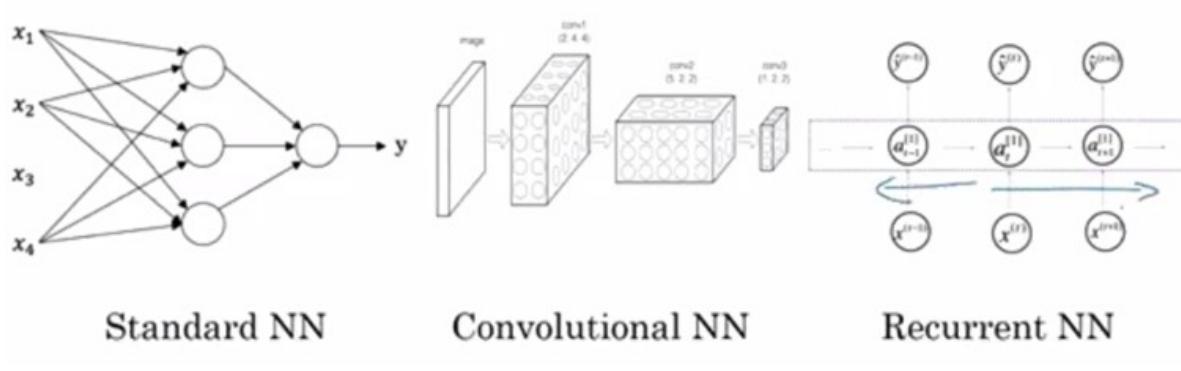
Aprendizaje Supervisado con Redes Neuronales

Ejemplos de Aprendizaje supervisado se muestran en la tabla siguiente:

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

donde cada aplicación en particular requiere una red neuronal específica ya sea CNN, RNN, híbridas, etc.

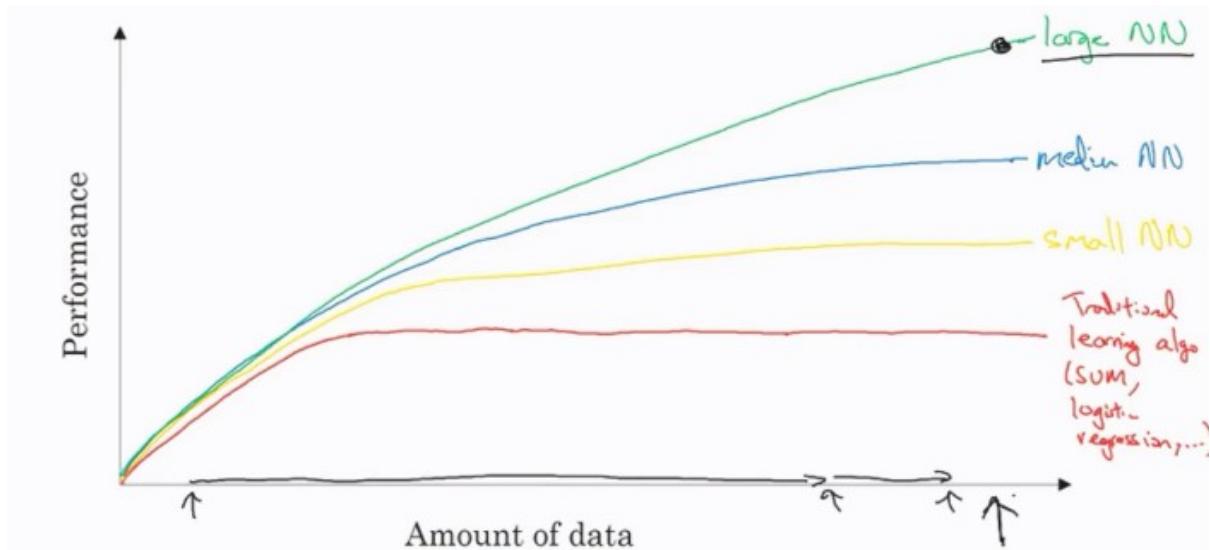
Un diagrama de estas redes neuronales podría ser el siguiente



Aplicaciones de Machine Learning sobre datos estructurados o no estructurados

Structured Data		Unstructured Data	
Size	#bedrooms	...	Price (1000\$)
2104	3		400
1600	3		330
2400	3		369
⋮	⋮		⋮
3000	4		540
User Age		Ad Id	Click
41		93242	1
80		93287	0
18		87312	1
⋮		⋮	⋮
27		71244	1

¿Por qué está despegando Deep Learning?



La cantidad de información disponible ha aumentado considerablemente. Los algoritmos tradicionales de aprendizaje no mejoran su rendimiento dada la mejora de datos, cosa que las redes neuronales si lo hacen.

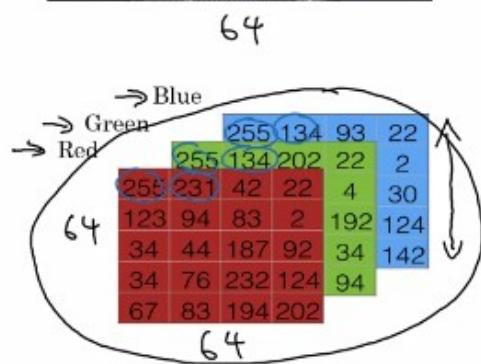
Para obtener un rendimiento alto habría que tener una red neuronal grande, con muchas unidades ocultas y mucha información.

Fundamentos de la programacion de redes neuronales

Clasificacion binaria

Un ejemplo sería determinar si, dada una foto, la misma pertenece a un gato o no.

Una foto color se puede representar como tres matrices RGB de, por ejemplo, 64x64. El vector de entrada (input feature) X tendrá como longitud $64 \times 64 \times 3 = 12288$. La idea del algoritmo de clasificación binaria es dado este vector Y determinar si es gato (1) o no (0).



$$X = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ \vdots \\ 255 \\ 134 \\ \vdots \\ \vdots \\ 67 \\ 83 \\ 194 \\ 202 \end{bmatrix}$$

$$64 \times 64 \times 3 = 12288$$

$$n = n_x = 12288$$

$$X \rightarrow y$$

Notación

Ejemplo de entrenamiento representado por el par (x, y) . En total se tendrán m , conocido como el conjunto de ejemplos de entrenamiento (training set). La matriz X consiste en poner en columnas los distintos ejemplos de entrenamiento. Con las salidas Y se procede de la misma manera.

$$(x, y) \quad x \in \mathbb{R}^{n_x}, y \in \{0, 1\}$$

$$m \text{ training examples: } \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$$M = M_{\text{train}}$$

$$M_{\text{test}} = \# \text{test examples.}$$

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | & | \end{bmatrix}_{n_x \times m}$$

$$X \in \mathbb{R}^{n_x \times m} \quad X.\text{shape} = (n_x, m)$$

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]$$

$$Y \in \mathbb{R}^{1 \times m}$$

$$Y.\text{shape} = (1, m)$$

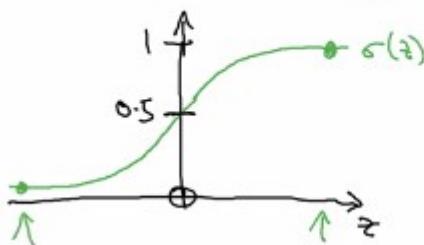
Regresión logística

La idea es determinar w y b . Como z puede tomar cualquier valor real, se le aplica la función sigmoide para que tome valores entre 0 y 1, correspondientes a una probabilidad.

Given x , want $\hat{y} = \frac{P(y=1|x)}{0 \leq \hat{y} \leq 1}$
 $x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$.

Output $\hat{y} = \sigma(w^T x + b)$



Casos extremos

La notación del curso de Machine Learning de Andrew Ng no se utilizará.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+BigNum} \approx 0$$

$$x_0 = 1, \quad x \in \mathbb{R}^{n_x+1}$$

$$\hat{y} = \sigma(\theta^T x)$$

$$\theta = \left[\begin{array}{c} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{array} \right] \right\} b \leftarrow$$

$$\left\{ \begin{array}{c} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{array} \right\} w \leftarrow$$

Función de coste de la regresión logística

Para entrenar los parámetros w y b se necesita una función de coste.

La función de perdida (Loss function) está asociada a un ejemplo únicamente. Puede definirse como el error cuadrado, pero no es convexa. Por eso se define de otra manera.

La función de coste se aplica sobre todo al conjunto de entrenamiento, minimizando la función para cada uno de estos puntos. Los valores w y b que minimizan la función serán los de interés.

A continuación se verá aplicada a una regresión logística como una red neuronal muy pequeña.

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}} \quad z^{(i)} = w^T x^{(i)} + b$$

Given $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$. $x^{(i)}$ $y^{(i)}$ $z^{(i)}$ i -th example.

Loss (error) function: $L(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2$

$$L(\hat{y}, y) = -(\hat{y} \log \hat{y} + (1-\hat{y}) \log (1-\hat{y})) \leftarrow$$

If $y=1$: $L(\hat{y}, y) = -\log \hat{y} \leftarrow$ Want $\log \hat{y}$ large, want \hat{y} large.

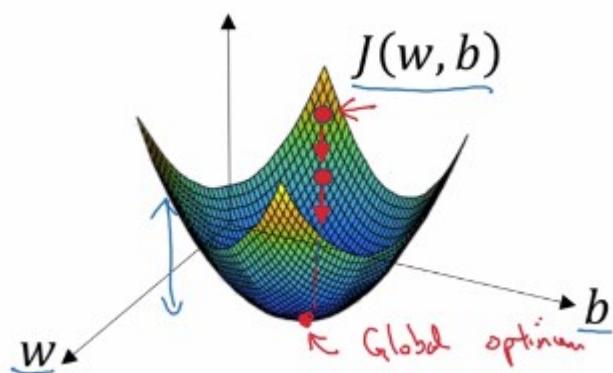
If $y=0$: $L(\hat{y}, y) = -\log (1-\hat{y}) \leftarrow$ Want $\log 1-\hat{y}$ large ... want \hat{y} small

Cost function: $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})]$

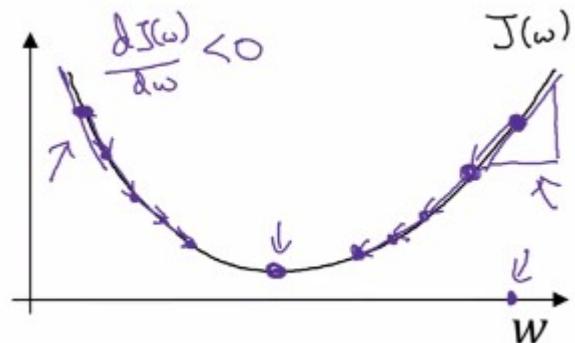


Gradiente descendente

¿Cómo se minimiza $J(w, b)$?



Pensando el problema para un b fijo,



el algoritmo consiste en repetir la ecuación hasta que w o b no varíen drásticamente.

Repeat {

$$w := w - \alpha \frac{\partial J(w)}{\partial w}$$

}

$$w := w - \alpha \frac{\partial J(w)}{\partial w}$$

learning rate

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b}$$

Grafico computacional

Simplemente para tener una idea de back y forward propagation, veamos con un ejemplo como una función de varias variables puede ser evaluada hacia adelante "forward" mientras que para derivarla habría que hacerlo hacia atrás ("backward").

En un ejemplo la función J depende de tres variables a , b y c . Las flechas azules representan la propagación hacia adelante para evaluar la función (cálculo de w y b), mientras que las rojas representan las derivadas (gradiente descendente).

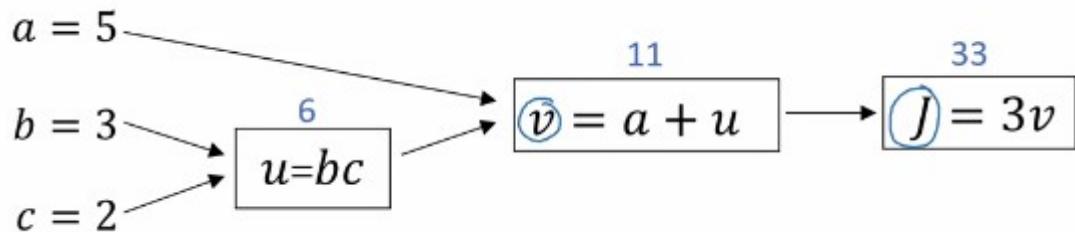
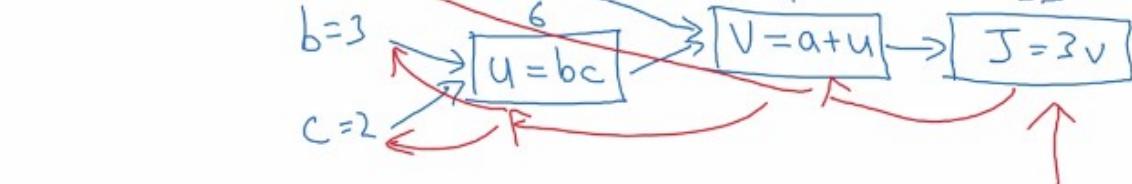
$$J(a, b, c) = 3(\underbrace{a + bc}_u) = 3(5 + 3 \cdot 2) = 33$$

$\underbrace{}_v$

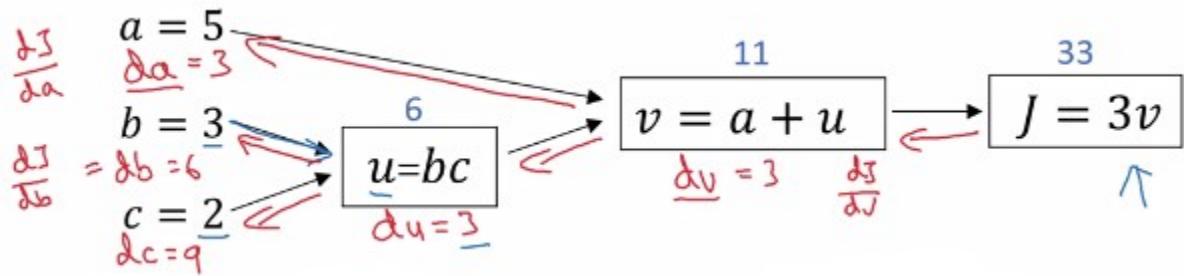
$$u = bc$$

$$v = a + u$$

$$J = 3v$$



Derivadas en un gráfico computacional



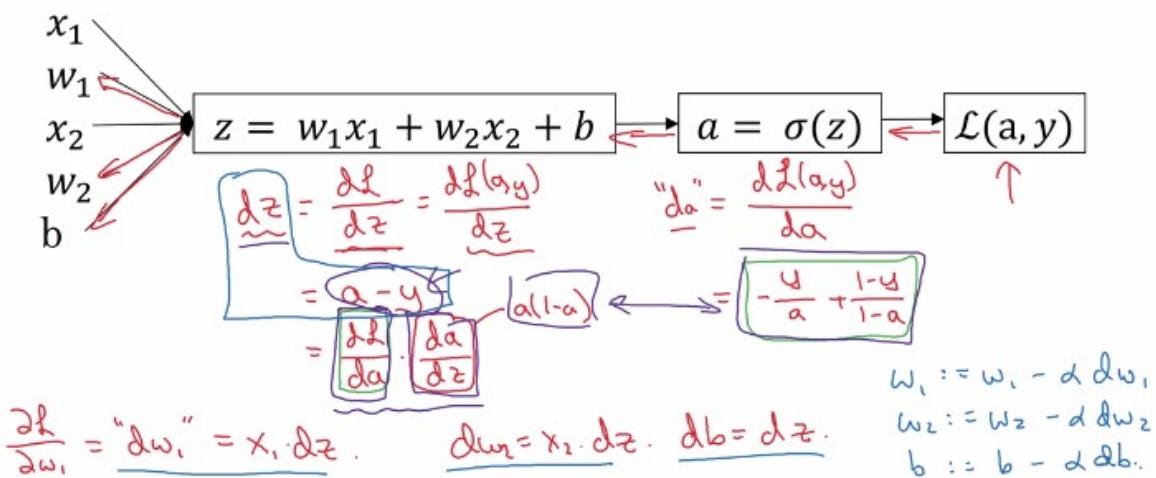
Gradiente descendente en regresión logística

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

Considerando dos entradas x_1, x_2 , para calcular w_1, w_2 y b primero habría que calcular dz y luego multiplicar por su correspondiente entrada.



Gradiente descendente en m ejemplos

En lugar de hacer lo anterior sobre la función de perdida, debemos hacerlo sobre la función de coste.

Pero las derivadas toman una forma simple como el promedio sobre todos los ejemplos.

$$\underline{J(w, b)} = \frac{1}{m} \sum_{i=1}^m l(a^{(i)}, y^{(i)})$$

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^\top x^{(i)} + b)$$

$$\underline{\frac{\partial}{\partial w_1} J(w, b)} = \frac{1}{m} \sum_{i=1}^m \underbrace{\frac{\partial}{\partial w_1} l(a^{(i)}, y^{(i)})}_{d w_1^{(i)}} - (x^{(i)}, y^{(i)})$$

El algoritmo se puede escribir de la siguiente manera

$$J = 0; \underline{\Delta w_1} = 0; \underline{\Delta w_2} = 0; \underline{\Delta b} = 0$$

For $i = 1$ to m

$$z^{(i)} = w^\top x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$$

$$\underline{\Delta z^{(i)}} = a^{(i)} - y^{(i)}$$

$$\begin{aligned} \underline{\Delta w_1} &+= x_1^{(i)} \underline{\Delta z^{(i)}} \\ \underline{\Delta w_2} &+= x_2^{(i)} \underline{\Delta z^{(i)}} \\ \underline{\Delta b} &+= \underline{\Delta z^{(i)}} \end{aligned} \quad \left. \begin{array}{l} n=2 \\ \vdots \end{array} \right.$$

$$J /= m \leftarrow$$

$$\underline{\Delta w_1} /= m; \underline{\Delta w_2} /= m; \underline{\Delta b} /= m. \leftarrow$$

$$\Delta w_1 = \frac{\partial J}{\partial w_1}$$

$$\begin{aligned} w_1 &:= w_1 - \alpha \underline{\Delta w_1} \\ w_2 &:= w_2 - \alpha \underline{\Delta w_2} \\ b &:= b - \alpha \underline{\Delta b} \end{aligned}$$

representando un único paso. Esto se repetirá hasta que tanto w como b converjan a un valor constante.

En este algoritmo tenemos dos for loops. Para que sea más eficiente se debería vectorizar.

Vectorización

$$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad w \in \mathbb{R}^{n_x}$$

$$x \in \mathbb{R}^{n_x}$$

Non-vectorized:

```
z = 0  
for i in range(n - x):  
    z += w[i] * x[i]  
  
z += b
```

Vectorized

$$z = \underbrace{\text{np.dot}(w, x)}_{w^T x} + b$$

Un ejemplo hecho por Andrew Ng demuestra la diferencia entre vectorizar o no el algoritmo.

250286.989866

Vectorized version: 1.5027523040771484ms

250286.989866

For loop: 474.29513931274414ms

Implementación vectorial del gradiente descendente en regresión logística

Todavía se necesitaría un for-loop sobre el número de iteraciones que se quieren para minimizar la función de coste.

$$\begin{aligned} z &= w^T X + b \\ &= np.dot(w.T, X) + b \\ A &= \sigma(z) \end{aligned}$$

$$\delta z = A - Y$$

$$\delta w = \frac{1}{m} X \delta z^T$$

$$\delta b = \frac{1}{m} np.sum(\delta z)$$

$$w := w - \alpha \delta w$$

$$b := b - \alpha \delta b$$

Broadcasting en Python

Es una técnica de Python para hacer funcionar el código más rápido.

A modo de ejemplo se quiere pasar la matriz a una matriz de porcentajes. Para ello primero conviene sumar sobre cada columna los valores y dividir ese valor total sobre cada elemento de la matriz (y multiplicar por 100).

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

cal = A.sum(axis = 0)

percentage = 100*A/(cal.reshape(1, 4))

dónde axis=0 indica sumar las por columna.

En porcentaje se dividió A que tiene dimensión 3x4 por cal que tiene dimensión 1x4.

Python/ Numpy vectors

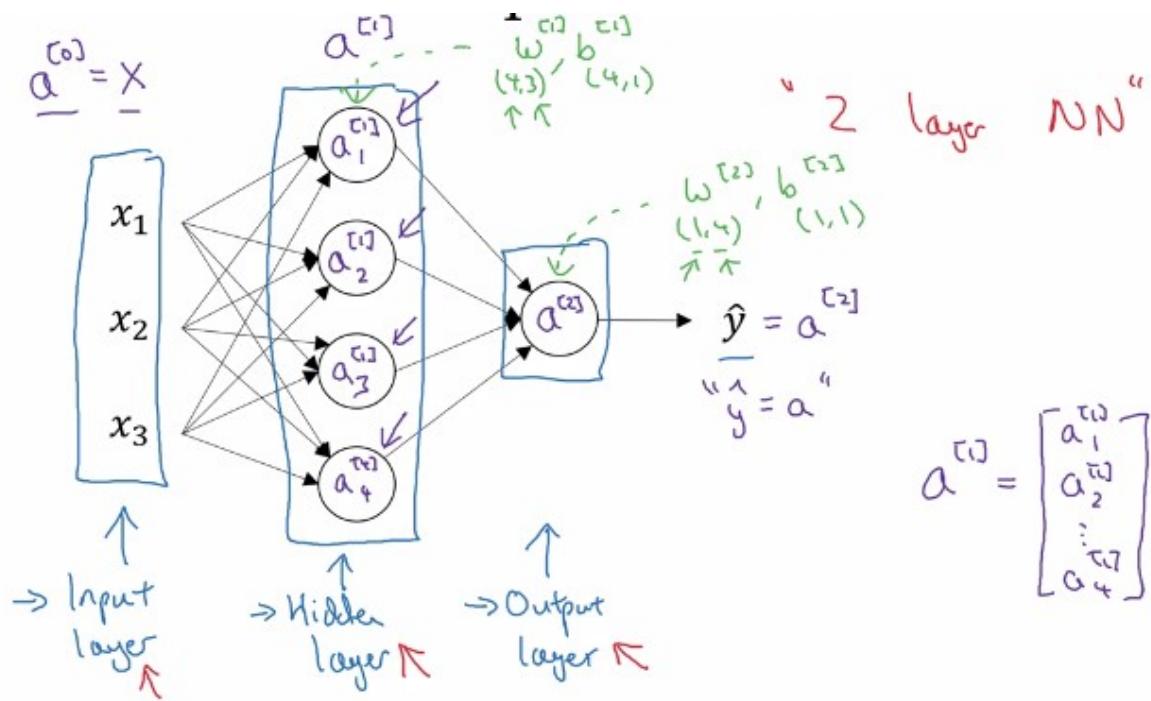
a = np.random.randn(5)
 a.shape = (5,) } "rank 1 array" Don't use

No son recomendables los arrays de rank 1 porque, por ejemplo, al usar np.dot() el comportamiento no es el esperado.

a = np.random.randn(5, 1) → a.shape = (5, 1) column vector ✓
 a = np.random.randn(1, 5) → a.shape = (1, 5) row vector. ✓

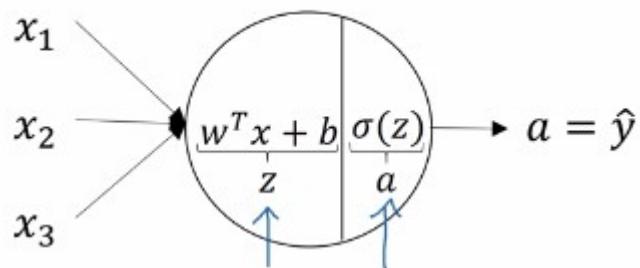
Representación de Redes Neuronales

Una red neuronal con una capa oculta.



¿Cómo se computan las salidas?

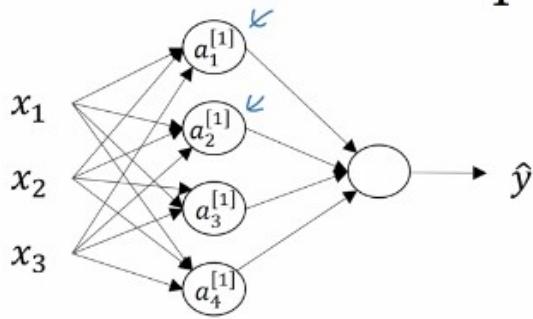
Un nodo en una red neuronal consiste en dos pasos de una regresión logística.



$$z = w^T x + b$$

$$a = \sigma(z)$$

Una red neuronal hace esto mismo, pero más veces. Esto es



$$\begin{aligned}
 z_1^{[1]} &= \underbrace{w_1^{[1]T} x + b_1^{[1]}}_{\text{layer } l}, \quad a_1^{[1]} = \sigma(z_1^{[1]}) \\
 z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]}) \\
 z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]}) \\
 z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})
 \end{aligned}$$

donde la notación viene dada por

$a_i^{[l]}$ \leftarrow layer
 $a_i^{[l]}$ \leftarrow node in layer.

Veamos cómo hacer para vectorizar estas ecuaciones

$$\begin{aligned}
 z^{[1]} &= \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}
 \end{aligned}$$

Given input x :

$$\rightarrow z^{[1]} = W^{[1]} \underset{(4,1)}{\underset{(4,4)}{\underset{(4,1)}{\underset{(4,1)}{a^{[1]}}}}} + b^{[1]}$$

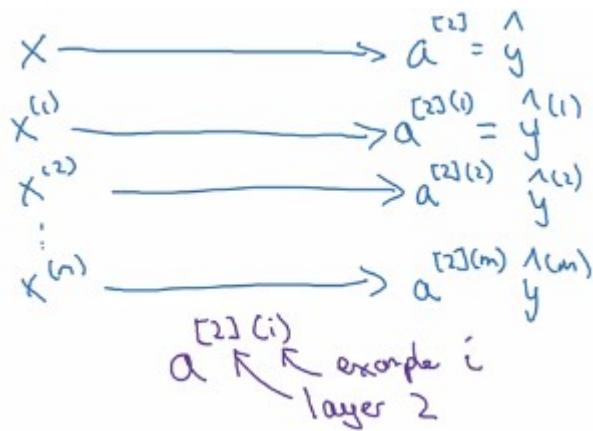
$$\rightarrow a^{[1]} = \sigma(z^{[1]})$$

$$\rightarrow z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$\rightarrow a^{[2]} = \sigma(z^{[2]})$$

Veamos ahora como predecir la salida de una red neuronal pero dadas varias entradas, no solo un ejemplo.

Introduciendo un poco de notación



Loopando sobre todos los ejemplos

for $i = 1$ to m :

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

Lo que queremos hacer ahora es vectorizarlo. Escribiendo las entradas en columnas,

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix}$$

\leftarrow

La vectorización
 resulta (Forward
 propagation)

trabajamos

hidden units.

$$\underline{A^{[1]}} = \begin{bmatrix} | & | & | \\ a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

Explicación de la implementación vectorizada

$$z^{[1]w} = \underbrace{w^{[1]} x^{[1]}}_{\text{Redes}} + b^{[1]}, \quad z^{[2]w} = \underbrace{w^{[2]} x^{[2]}}_{\text{Redes}} + b^{[2]}, \quad z^{[3]w} = \underbrace{w^{[3]} x^{[3]}}_{\text{Redes}} + b^{[3]}$$

$$w^{[1]} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}, \quad w^{[2]} x^{[1]} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}, \quad w^{[3]} x^{[2]} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

$$w^{[1]} \begin{bmatrix} 1 & x^{[1]} & x^{[1]} x^{[1]} \dots \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} z^{[1]w} & z^{[2]w} & z^{[3]w} \dots \\ \vdots & \vdots & \vdots \end{bmatrix} = \vec{z}^{[1]}$$

La función sigmoide no es la mejor elección en redes neuronales. Estudiemos otras funciones de activación más útiles.

Funciones de activación

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

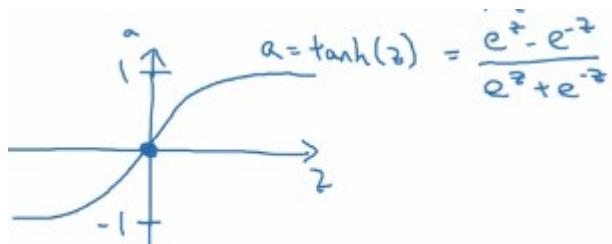
Reemplazamos la función sigmoide por una función de activación general $g()$.

$$a^{[1]} = \sigma(z^{[1]}) g^{[1]}(z^{[1]})$$

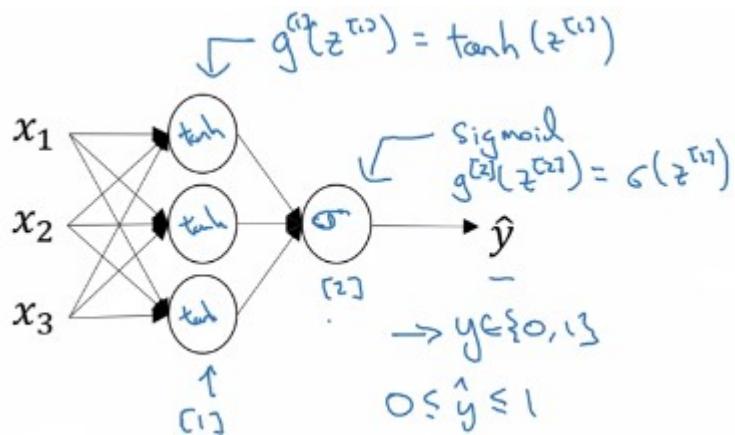
En general la tangente hiperbólica funcionara mejor.

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]}) g^{[2]}(z^{[2]})$$

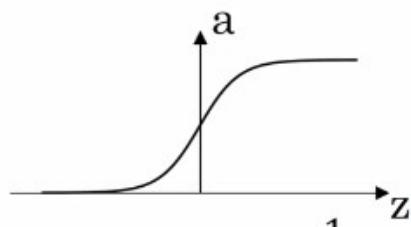


Pero se pueden utilizar distintas funciones de activación en distintas capas. Es conveniente utilizar en la salida la función sigmoide para clasificación binaria, ya que entrega valores entre 0 y 1. Pero en las capas ocultas conviene utilizar la tangente hiperbólica.

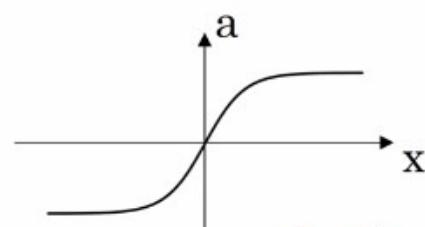


Ventajas y desventajas de las funciones de activación

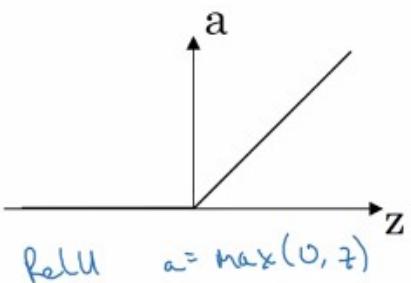
En resumen,



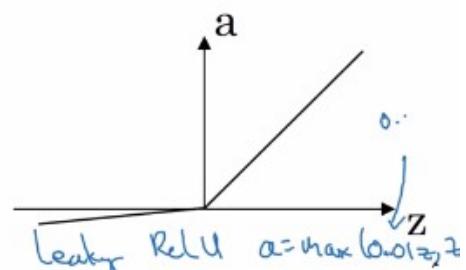
$$\text{sigmoid: } a = \frac{1}{1 + e^{-z}}$$



$$\tanh: a = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



$$\text{ReLU: } a = \max(0, z)$$



Derivadas de las funciones de activación

Para backpropagation necesitamos conocer las derivadas de las funciones de activación.

- Función de activación sigmoide

$$\begin{aligned}
 \frac{\partial}{\partial z} g(z) &= \text{slope of } g(z) \text{ at } z \\
 &= \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}} \right) \\
 &= g(z) (1 - g(z))
 \end{aligned}$$

- Función de activación tangente hiperbólica

$$\begin{aligned}
 g'(z) &= \frac{\partial}{\partial z} g(z) = \text{slope of } g(z) \text{ at } z \\
 &= 1 - (\tanh(z))^2
 \end{aligned}$$

- RELU

$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \\ \cancel{0} & \text{if } z = 0 \end{cases}$$

- Leaky RELU

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Gradiente descendente para redes neuronales

Consideremos primero una capa oculta nomás.

Parametros: $\omega^{[0]}, b^{[0]}, \omega^{[1]}, b^{[1]}$
 $(n^{[0]}, n^{[1]}) \quad (n^{[1]}, 1) \quad (n^{[1]}, n^{[1]}) \quad (n^{[1]}, 1)$

$$n_x = n^{[0]}, n^{[1]}, \underline{n^{[1]} = 1}$$

Previamente habíamos visto como calcular las predicciones y_{hat} , ahora las derivadas de la función de coste.

Cost function: $J(\underline{w}^{(1)}, b^{(1)}, \underline{w}^{(2)}, b^{(2)}) = \frac{1}{m} \sum_{i=1}^m l(\hat{y}_i, y_i)$

Gradient Descent:

→ Repeat {

Compute predict ($\hat{y}^{(i)}$, $i=1 \dots m$)

$$\frac{\partial J}{\partial w^{(1)}} = \frac{\partial J}{\partial w^{(1)}}, \quad \frac{\partial J}{\partial b^{(1)}} = \frac{\partial J}{\partial b^{(1)}}, \dots$$

$$w^{(1)} := w^{(1)} - \alpha \frac{\partial J}{\partial w^{(1)}}$$

$$b^{(1)} := b^{(1)} - \alpha \frac{\partial J}{\partial b^{(1)}}$$

Fórmulas para calcular derivadas (Backward propagation)

Back propagation:

$$\frac{\partial z^{(2)}}{\partial} = A^{(2)} - Y \leftarrow$$

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]$$

$$\frac{\partial w^{(2)}}{\partial} = \frac{1}{m} \frac{\partial z^{(2)}}{\partial} A^{(1)T}$$

$$(n^{(2)}) \leftarrow$$

$$\frac{\partial b^{(2)}}{\partial} = \frac{1}{m} \text{np.sum}(\frac{\partial z^{(2)}}{\partial}, \text{axis}=1, \text{keepdims=True})$$

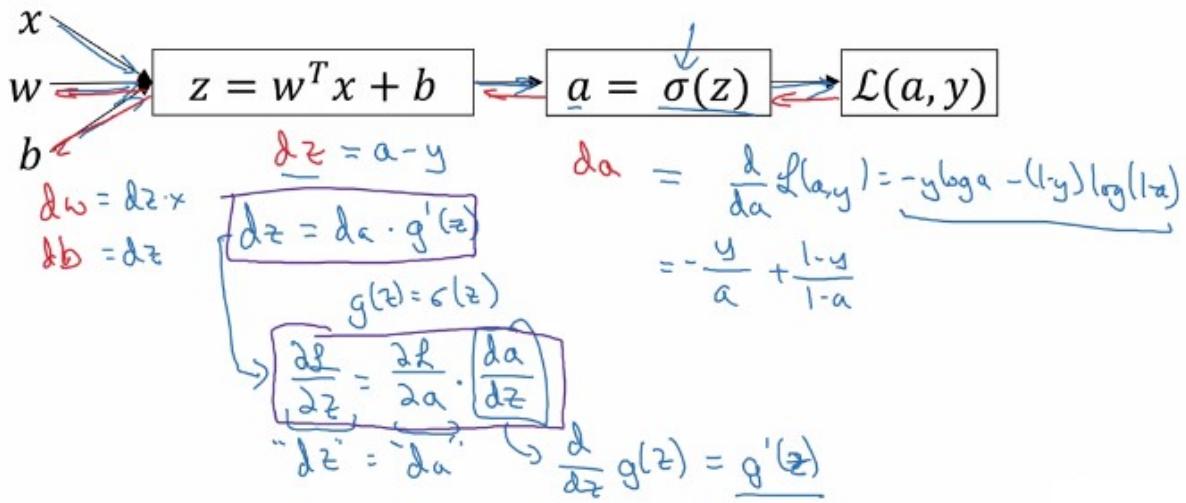
$$\frac{\partial z^{(1)}}{\partial} = \underbrace{w^{(2)T} \frac{\partial z^{(2)}}{\partial}}_{(n^{(2)}, m)} \times \underbrace{g^{(1)'}(\bar{z}^{(1)})}_{\text{element-wise product}} \quad (n^{(1)}, m)$$

$$\frac{\partial w^{(1)}}{\partial} = \frac{1}{n} \frac{\partial z^{(1)}}{\partial} X^T$$

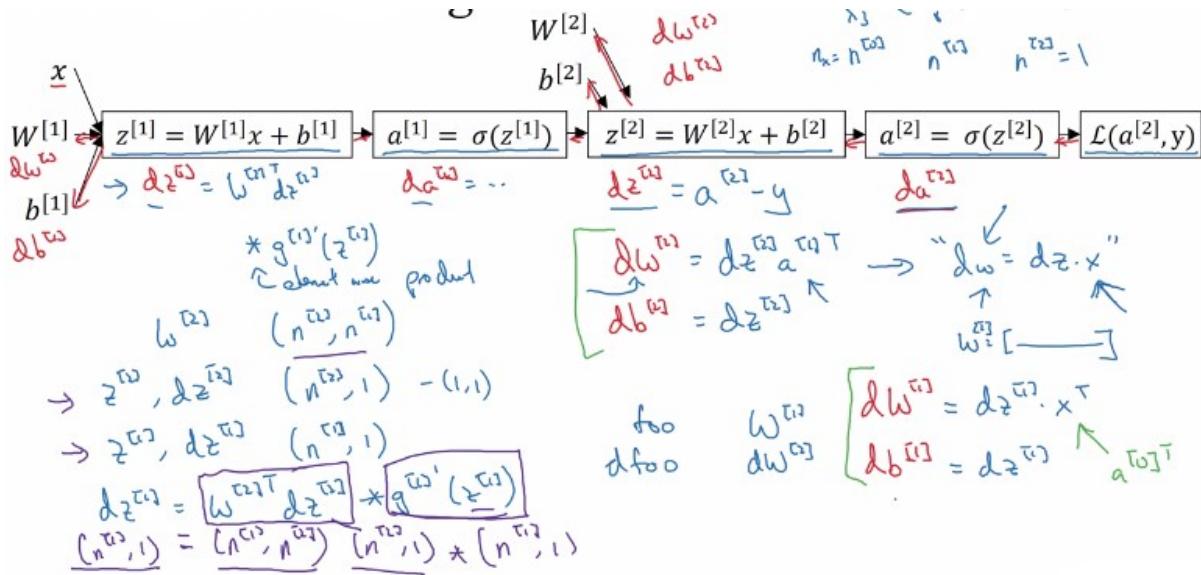
$$\frac{\partial b^{(1)}}{\partial} = \frac{1}{n} \text{np.sum}(\frac{\partial z^{(1)}}{\partial}, \text{axis}=1, \text{keepdims=True})$$

Intuición de la backward propagation

- Regresión logística



- Gradiente en red neuronal



De manera más prolífica, las 6 ecuaciones de interés para el backpropagation

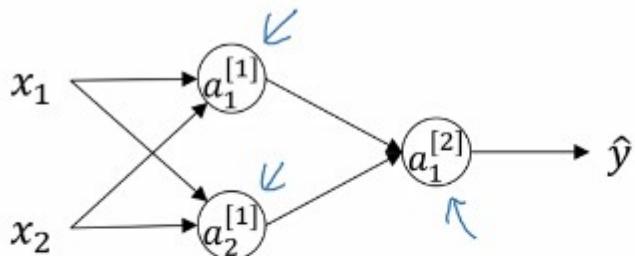
$dZ^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$	$J(\cdot) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}_i, y_i)$
$dW^{[2]} = dZ^{[2]} a^{[1]T}$	$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$	
$db^{[2]} = dZ^{[2]}$	$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims=True)$	
$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(z^{[1]})$	$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'}(z^{[1]})}_{\text{elementwise product}}$	
$dW^{[1]} = dZ^{[1]} x^T$	$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$	
$db^{[1]} = dZ^{[1]}$	$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True)$	

Inicialización parámetros

- En regresión logística se puede inicializar los parámetros con valor cero.
- Pero la red neuronal hay que inicializarla aleatoriamente. Aplicar gradiente descendente a una red inicializada con ceros, no funcionará.

Inicialización aleatoria

El factor 0.01 en w es para que se inicialicen con valores pequeños. Si tomasen valores grandes la función de activación saturaría. Por otra parte, a los parámetros b no les afecta inicializarlos con valores nulos.



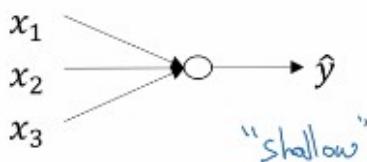
$$\begin{aligned} \rightarrow w^{[1]} &= \text{np. random. random}((2, 2)) * \frac{0.01}{100} ? \\ b^{[1]} &= \text{np. zeros}((2, 1)) \\ w^{[2]} &= \dots \\ b^{[2]} &= 0 \end{aligned}$$



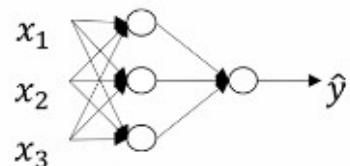
Cuando se entrene alguna red neuronal más profunda sería más razonable usar otro valor que no fuera 0.01, pero se verá más adelante.

Redes neuronales profundas de L-capas

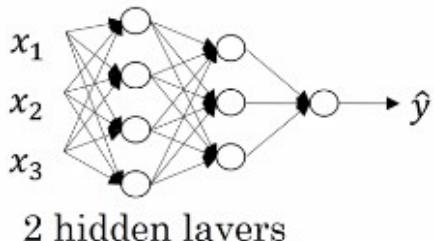
Ejemplos de redes neuronales.



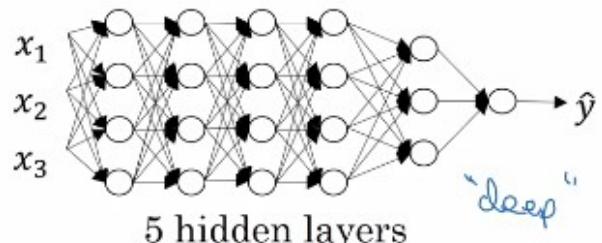
logistic regression



1 hidden layer

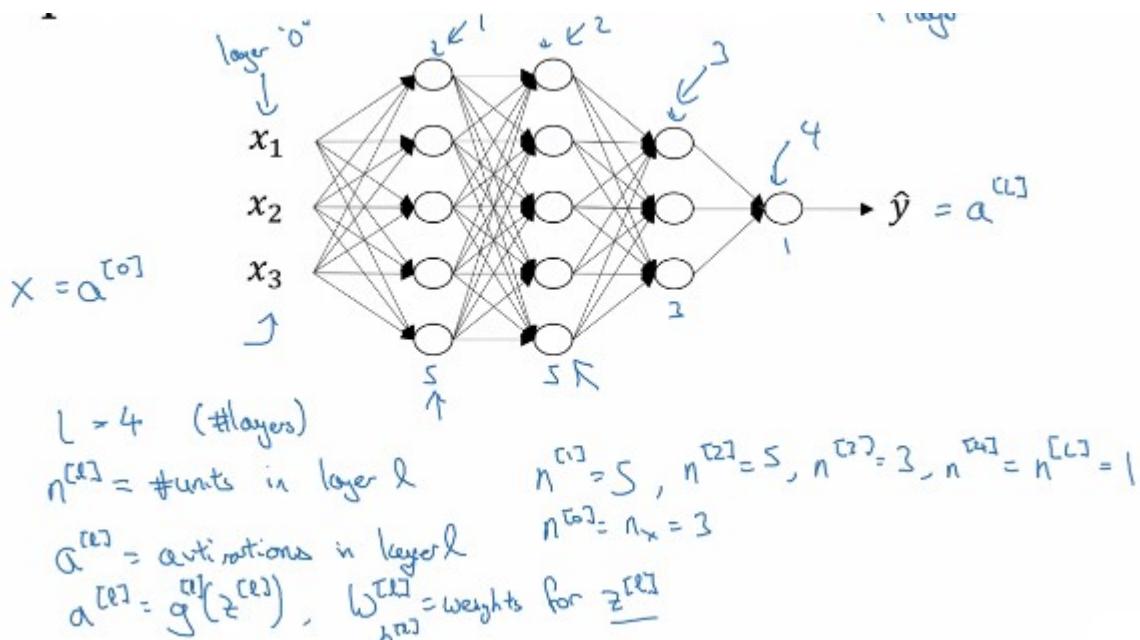


2 hidden layers



5 hidden layers

Veamos la notación a utilizar



Forward and Backward propagation in deep neural networks

- Forward propagation

Input $a^{[l-1]}$ ↗
 Output $a^{[l]}$, cache ($\underline{z}^{[l]}$) ↗

$$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

Vectorial:

$$z^{[l]} = w^{[l]} \cdot A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(z^{[l]})$$

Estas dos ecuaciones se encuentran en un for-loop, el cual no parecería que se puede vectorizar.

- Backward propagation

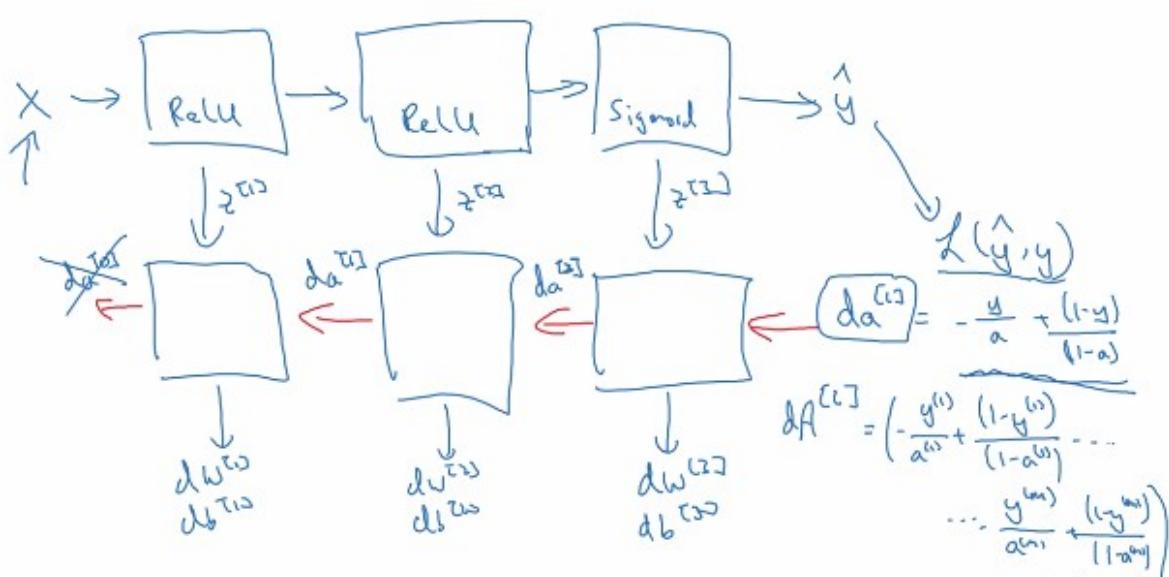
Input $da^{[l]}$

Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

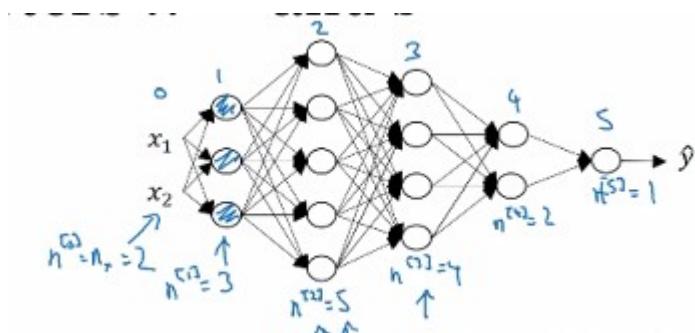
$$\begin{aligned} dz^{[l]} &= da^{[l]} * g'(z^{[l]}) \\ dw^{[l]} &= dz^{[l]} \cdot a^{[l-1]} \\ db^{[l]} &= dz^{[l]} \\ da^{[l-1]} &= w^{[l]} \cdot dz^{[l]} \\ dz^{[l-1]} &= w^{[l-1]} dz^{[l]} * g'(z^{[l]}) \end{aligned}$$

$$\begin{aligned} dz^{[l]} &= dA^{[l]} * g'(z^{[l]}) \\ dw^{[l]} &= \frac{1}{m} dz^{[l]} \cdot A^{[l-1]\top} \\ db^{[l]} &= \frac{1}{m} np \cdot \text{sum}(dz^{[l]}, \text{axis}=1, \text{keepdim=True}) \\ dA^{[l-1]} &= w^{[l]\top} \cdot dz^{[l]} \end{aligned}$$

Resumen



Método de debugging: dimensiones de las matrices



Los parámetros w para un ejemplo:

$$w^{[1]}: (n^{[1]}, n^{[0]})$$

$$w^{[2]}: (5, 3) \quad (n^{[2]}, n^{[1]})$$

De manera general.

$$w^{[l]}: (n^{[l]}, n^{[l-1]})$$

Para b

$$b^{[l]}: (n^{[l]}, 1)$$

Para backpropagation dw y db tienen la misma dimensión que W y b respectivamente.

$$\Delta w^l: (n^{[l]}, n^{[l-1]})$$

$$\Delta b^l: (n^{[l]}, 1)$$

Otras cantidades de interés: z = g(a), y a tienen la misma dimensión,

Considerando los m ejemplos a entrenar

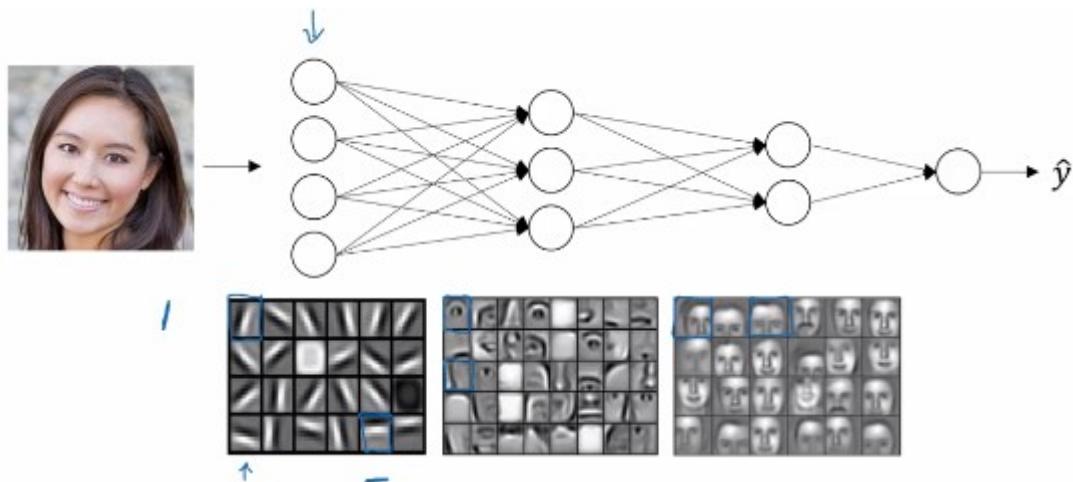
$$\rightarrow \hat{z}^{(l)} = w^{[l]} \cdot X + b^{[l]}$$

$\underbrace{(n^{[l]}, m)}$ $\underbrace{(n^{[l]}, n^{[l-1]})}$ $\underbrace{(n^{[l]}, m)}$ $\underbrace{(n^{[l]}, 1)}$
 $\underbrace{\phantom{(n^{[l]}, m)}}$ $\underbrace{\phantom{(n^{[l]}, n^{[l-1]})}}$ $\underbrace{\phantom{(n^{[l]}, m)}}$ $\underbrace{\phantom{(n^{[l]}, 1)}}$
 $(n^{[l]}, m)$

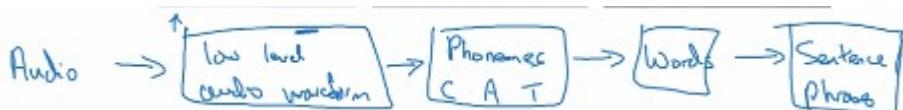
En Python b podría seguir siendo de la misma dimensión, pero por Broadcasting es redimensionada al tamaño correcto.

¿Por que las redes neuronales profundas generalmente funcionan mejor que las que no lo son?

Como ejemplo en la identificación de una cara. Se podría pensar a la primera capa de la red neuronal como la encargada de identificar bordes. A la siguiente para identificar ojos, o narices mediante el encajado de bordes. Mediante el rejunte de ojos, narices la red puede detectar caras.



Para un audio, a bajo nivel la red neuronal podría detectar los cambios de sonido, luego unidades básicas de sonido, como fonemas, luego poder reconocer palabras, para luego reconocer oraciones o frases.



¿Qué son los hiper parámetros?

Dados los parámetros de una red neuronal,

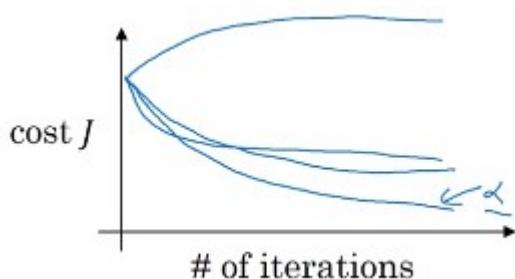
Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]} \dots$

los hiper parámetros son aquellos que controlan y determinan el valor de los parámetros.

Ellos son:

- Learning ratio alfa
- número de iteraciones
- numero de capas ocultas L
- número de unidades ocultas $n^{[1]}, n^{[2]}, \dots$
- elección de funciones de activación
- etc.

Deep Learning es un proceso empírico. El valor de los hiper parámetros se calcula mediante la experimentación.

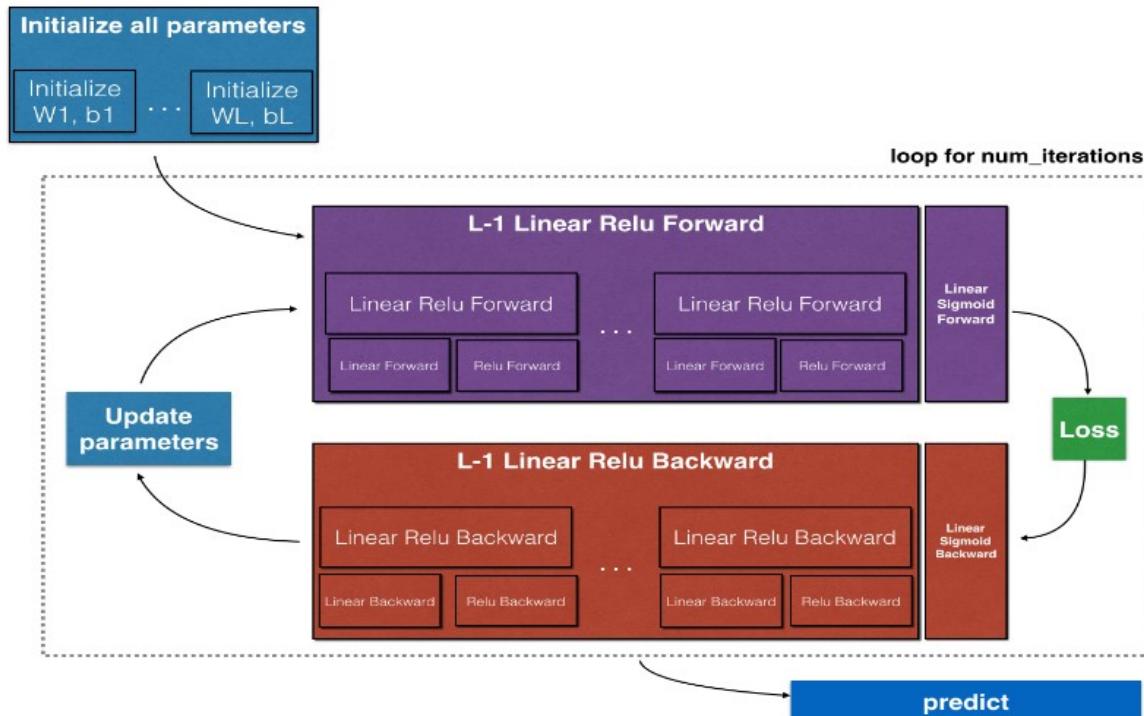


Resumen

Forward y Backward propagation

$$\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= g^{[2]}(Z^{[2]}) \\ &\vdots \\ A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y} \end{aligned}$$

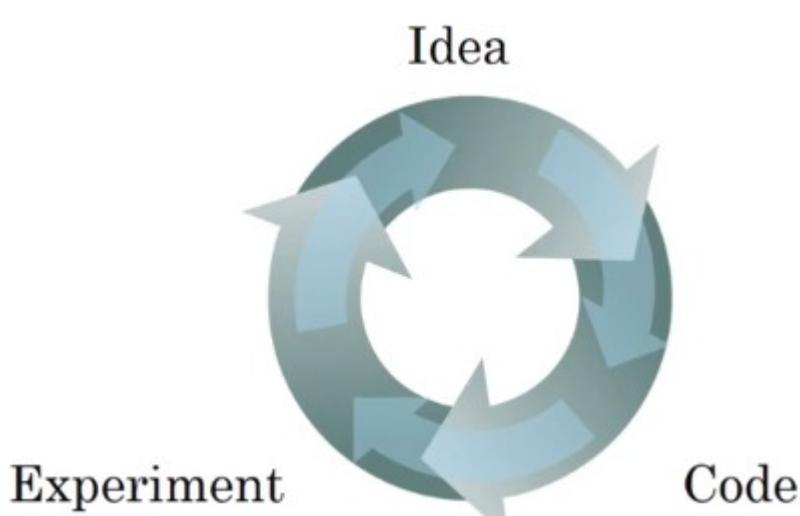
$$\begin{aligned} dZ^{[L]} &= A^{[L]} - Y \\ dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]T} \\ db^{[L]} &= \frac{1}{m} np.\text{sum}(dZ^{[L]}, axis = 1, keepdims = True) \\ dZ^{[L-1]} &= dW^{[L]T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\ &\vdots \\ dZ^{[1]} &= dW^{[L]T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\ dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]T} \\ db^{[1]} &= \frac{1}{m} np.\text{sum}(dZ^{[1]}, axis = 1, keepdims = True) \end{aligned}$$



Curso #2: Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization

Practical Aspects of Deep Learning

Train/dev/tests sets



Generalmente estaba aceptado que los conjuntos de datos se separan de la siguiente forma 70/30% o 60/20/20. Aunque con el Big Data, del orden de 1M de ejemplos, se suelen modificar hasta del orden de 95.5/0.25/0.25 por ejemplo.

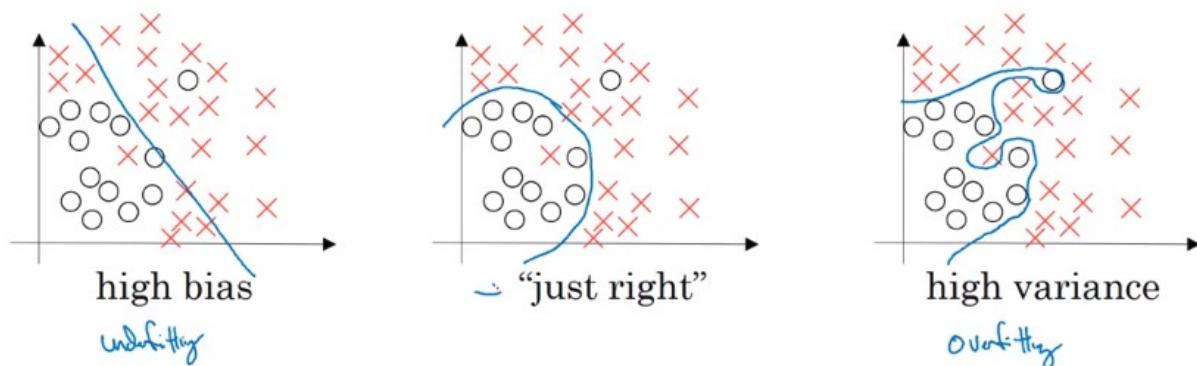
Mismatched train/test distribution

Generalmente se requiere que los conjuntos de datos tengan la misma distribución de probabilidad.

Training set:
Cat pictures from
webpages } ↪
Dev/test sets:
Cat pictures from
users using your app

Bias/Variance

Ejemplo donde se tienen dos features.



En el caso de haber más de una hay diferentes métricas que se pueden utilizar.

Como ejemplo de clasificación de gatitos, una manera de determinar si se comete algún sesgo o varianza es mirando los errores en el train y dev set.

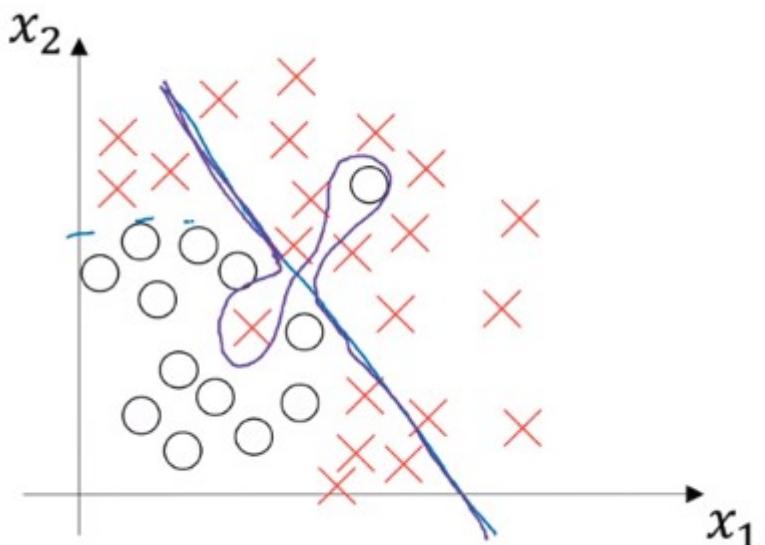
Cat classification



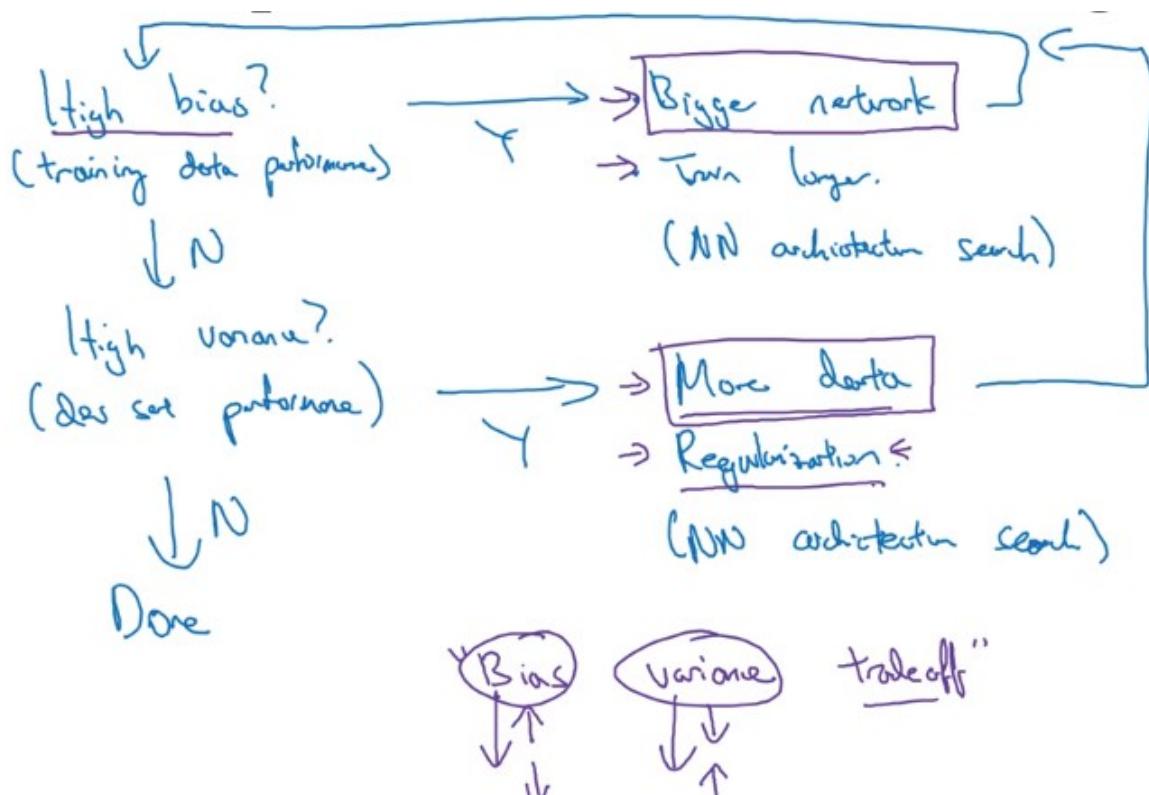
Train set error:	10%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high varan	low bias low variance
Human: ~0%				
Optimal (Bayes) error: ~0%				

Estas medidas suponen tanto que, como se dijo antes, las distribuciones de los datos provienen de la misma distribución de probabilidad, como que las imágenes no son borrosas para que se considere el error humano como 0%. La falla del primer caso se verá más adelante. En el caso de tener imágenes borrosas por ejemplo, puede provocar que por ejemplo el segundo caso de errores de 15% y 16% en el train y dev set error respectivamente, sean considerados relativamente aceptables.

El tercer caso de alto sesgo y varianza parece ser contradictorio. Veamos un ejemplo:



Receta básica para eliminar sesgo y varianza en Machine Learning



Primero podríamos ver si el algoritmo tiene un sesgo alto sobre el training set. En caso afirmativo se podría agrandar la red neuronal con más unidades ocultas, entrenarlo más tiempo, ya sea corriendo gradiente descendente un mayor tiempo o (a veces funciona a veces no) tratar de encontrar una mejor arquitectura para la red neuronal. Luego de probar esto se podría probar nuevamente si el problema de alta sesgo fue solucionado.

Si el problema de alto sesgo fue solucionado se podría probar si hay alta varianza sobre el dev set, probando con más datos o aplicando regularización y también, como antes, probar una nueva arquitectura para la red neuronal.

Cuando ambas son negativas se podría afirmar que se terminó.

Regularización de una red neuronal

Regularización

Este método ayuda a prevenir el overfitting y errores de la red neuronal.

Aplicado a Regresión Logística

$$\min_{w,b} J(w, b)$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2 + \underbrace{\frac{\lambda}{2m} b^2}_{\text{omit}}$$

$$\text{L}_2 \text{ regularization} \quad \|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$$

Esto se llama regularización L₂, donde lambda es el parámetro de regularización. El último término se omite dado que w suele poseer muchos más parámetros que b ($n_x \gg 1$).

A su vez, existe lo que se llama regularización L₁, pero no es utilizado tan comúnmente.

$$\text{L}_1 \text{ regularization} \quad \frac{\lambda}{2m} \sum_{i=1}^{n_x} |w_i| = \frac{\lambda}{2m} \|w\|_1$$

¿Qué forma toma la regularización para una red neuronal?

$$J(w^{(0)}, b^{(0)}, \dots, w^{(L)}, b^{(L)}) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(\hat{y}^{(i)}, y^{(i)})}_{\text{loss function}} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^L \|w^{(l)}\|_F^2}_{\text{regularization term}}$$

$$\|w^{(l)}\|_F^2 = \sum_{i=1}^{n^{(l+1)}} \sum_{j=1}^{n^{(l)}} (w_{ij}^{(l)})^2$$

$$\omega: (n^{(0)} \ n^{(1)} \ \dots)$$

"Frobenius norm" $\|\cdot\|_2^2$ $\|\cdot\|_F^2$

Podemos ver que la expresión es similar pero ahora se toma una norma de Frobenius sobre los parámetros w.

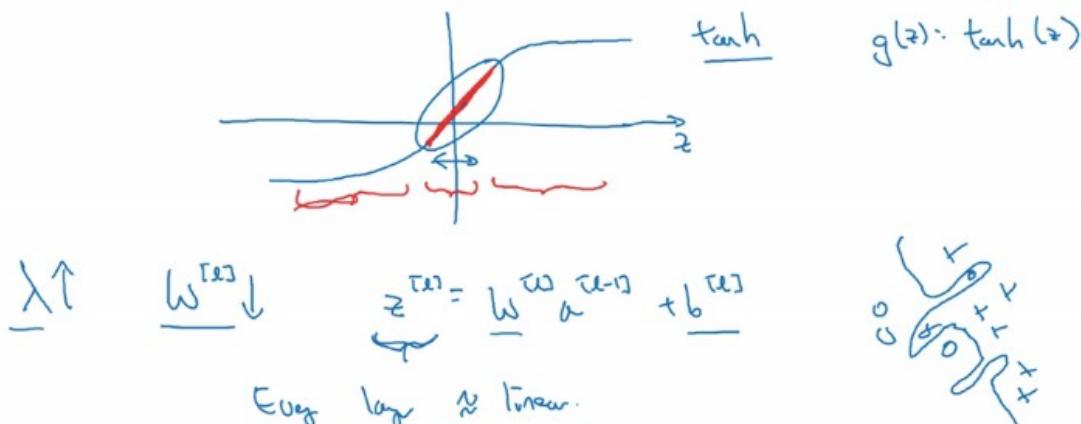
Por otra parte, la backpropagation es de la forma

$$\begin{aligned}\delta w^{(l)} &= (\text{from backprop}) + \frac{\lambda}{m} w^{(l)} \\ \rightarrow w^{(l)} &:= w^{(l)} - \delta \delta w^{(l)}\end{aligned}$$

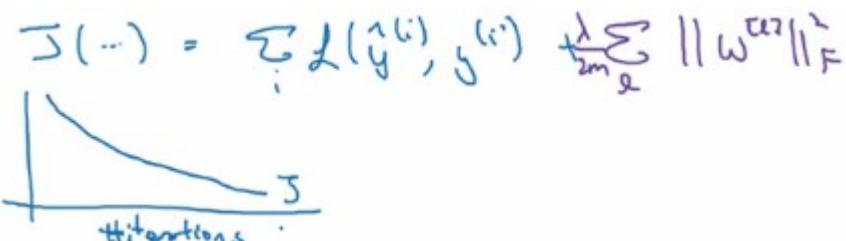
$$\frac{\partial J}{\partial w^{(l)}} = \delta w^{(l)}$$

¿Por qué la regularización reduce overfitting?

Tomando como ejemplo la función de activación $\tanh(z)$, para un valor grande de lambda, w es pequeño y por consiguiente z también. De aquí que \tanh se valúe prácticamente en la región lineal y se prevenga el overfitting. La red neuronal será prácticamente una red lineal.

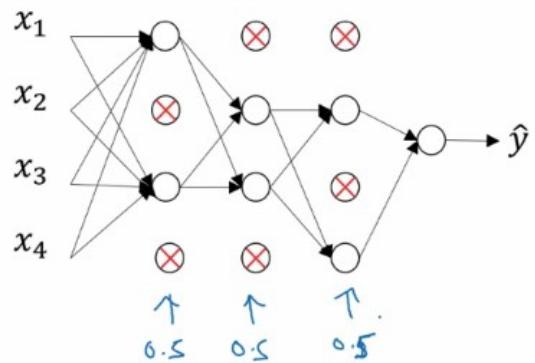
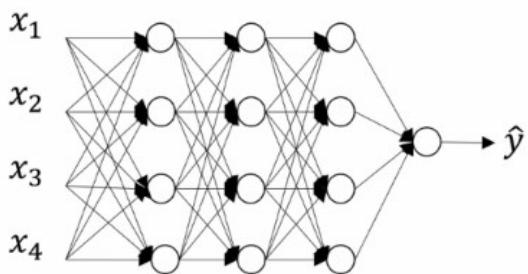


Por otra parte, para probar el correcto funcionamiento de la implementación se puede probar graficando la función de coste en función del número de iteraciones y ver que esta es una función monótonicamente decreciente.

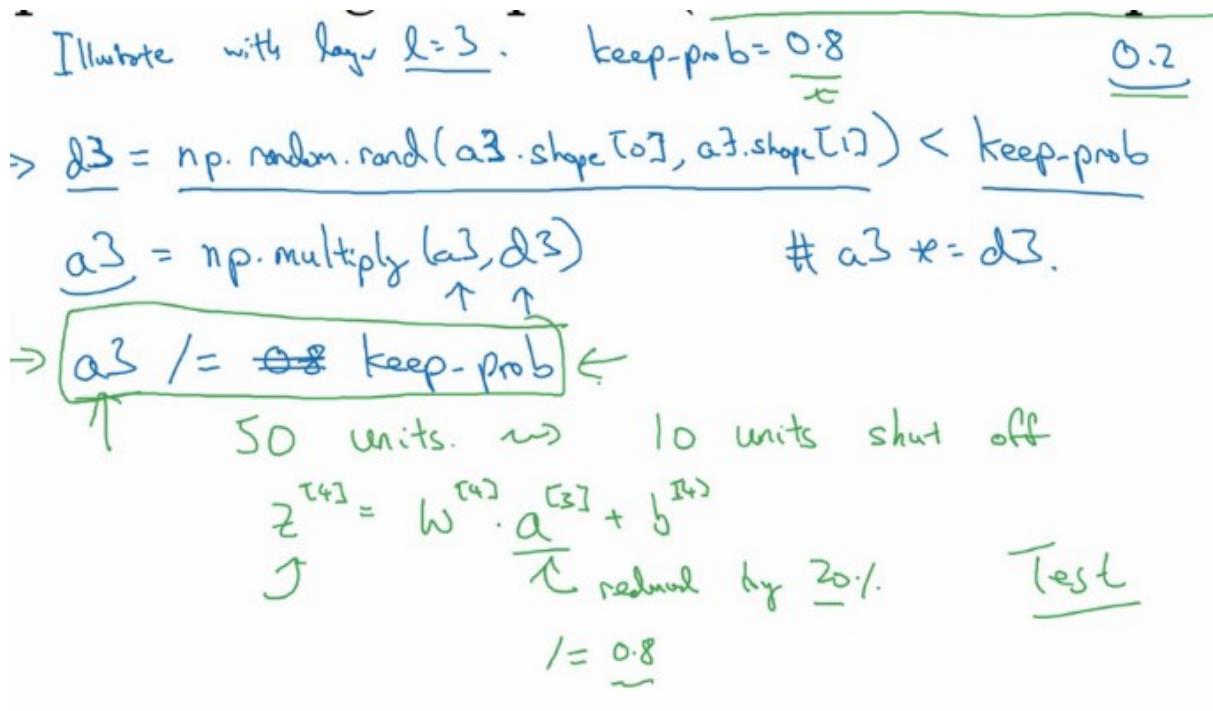


Dropout regularización

Esta es una nueva técnica de regularización. La misma consiste en eliminar nodos con una dada probabilidad. El nuevo cálculo se realiza sin los nodos eliminados y se previene la regularización.



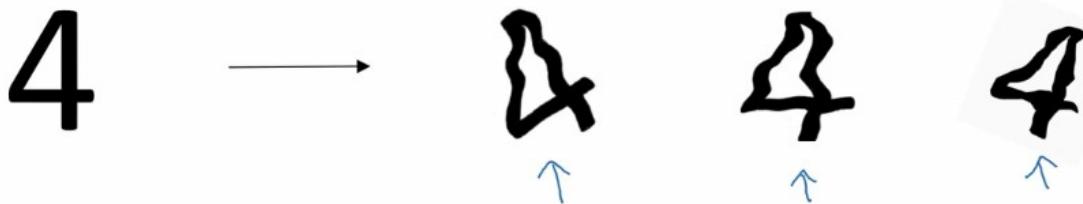
Pero, ¿Cómo se implementa esta regularización? Una técnica se conoce como inverted dropout.



Otros métodos de regularización

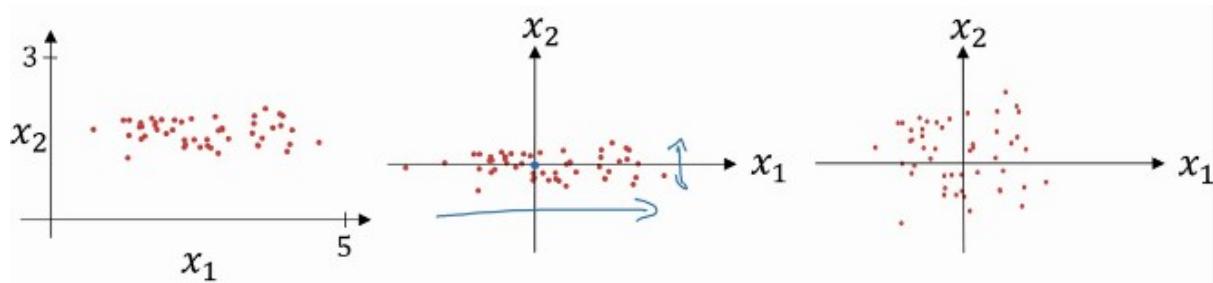
Data Augmentation

En el caso de estar overfitteando. Se pueden agregar más ejemplos. Pero puede ser muy caro. Para ello se pueden espejar horizontalmente algunas fotos o modificarlas sutilmente, como se muestra en la figura inferior.

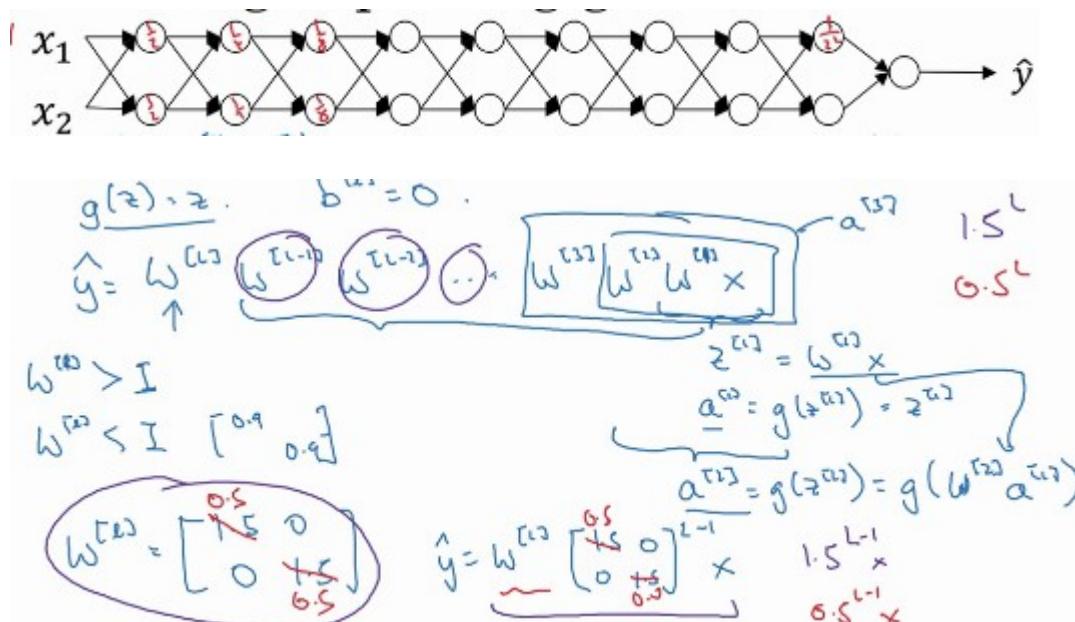


Configurando la optimización de problemas

- Normalización de resultados: evitar w grandes o pequeños. Escalar el training y test set mediante de la misma manera.



- Vanishing/ exploding gradientes: cuando se entrenan redes neuronales muy profundas (muchos hidden layers), las activaciones pueden ser, o muy chicas, o muy grandes.



Suponiendo que $b=0$ y que la función de activación es lineal, se puede demostrar que w puede ser o muy grande o muy pequeño. Esto se puede resolver inicializando los pesos.

Comencemos con un ejemplo para una única neurona, luego se verá el caso de una red neuronal más compleja.

Dado que z es la suma de términos de la forma w^*x , si n aumenta se quiere que w disminuya. ¿De qué manera? Una buena opción es que la varianza de los w vaya como $1/n$. Para el caso de las funciones RELU, $2/n$ funciona mejor. De esta manera se llega a una expresión para w que no es muy grande o muy pequeña si los features aumentan considerablemente.

$$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n \quad \cancel{\text{X}}$$

$\underbrace{\text{Large } n}_{\text{Large } z} \rightarrow \underbrace{\text{Smaller } w_i}_{\text{Small } w}$

$$\text{Var}(w_i) = \frac{1}{n} \frac{2}{n}$$

$$w_{\text{ReLU}}^{(l)} = \text{np.random.rand}(n) * \sqrt{\frac{2}{n^{(l-1)}}}$$

$g_{\text{ReLU}}^{(l)}(z) = \text{ReLU}(z)$

Otras variantes para distintas funciones de activación como tanh: Inicialización de Xavier.

Other variants:

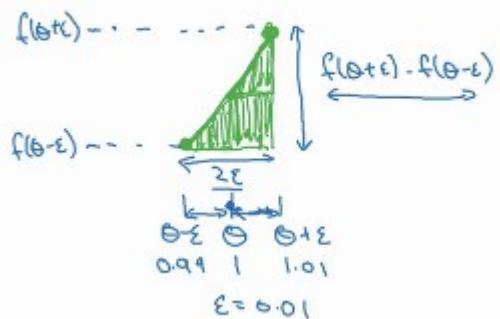
Tanh

Xavier initialization

$$\frac{2}{n^{(L-1)} + n}$$

Aproximación numérica de gradientes

Calculando derivadas a ambos lados.



A diferencia de la derivada a un lado, esta posee un error más pequeño, pero se tarda más en calcular su valor.

$$f'(\theta) = \lim_{\epsilon \rightarrow 0} \frac{f(\theta + \epsilon) - f(\theta - \epsilon)}{2\epsilon} \quad \text{error: } O(\epsilon^2)$$

$\epsilon = 0.01$	$\frac{f(\theta + \epsilon) - f(\theta)}{\epsilon}$	error: $O(\epsilon)$
$\epsilon = 0.0001$	$\frac{f(\theta + \epsilon) - f(\theta)}{\epsilon}$	0.01

Gradient Checking: técnica para verificar la implementación de backpropagation

Primero conviene realizar dos pasos previos y quedarnos con un vector $d\theta$. Y preguntarse, ¿es este vector el gradiente de $J(\theta)$?

Take $\underline{W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}}$ and reshape into a big vector $\underline{\theta}$.

concatenate

$$J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = J(\underline{\theta})$$

Take $\underline{dW^{[1]}, db^{[1]}, \dots, dW^{[L]}, db^{[L]}}$ and reshape into a big vector $\underline{d\theta}$.

concatenate

Is $\underline{d\theta}$ the gradient of $J(\underline{\theta})$?

$$\begin{aligned} \text{for each } i: \\ \rightarrow \underline{d\theta_{\text{approx}}[i]} &= \frac{J(\theta_1, \theta_2, \dots, \overset{\downarrow}{\theta_i + \epsilon}, \dots) - J(\theta_1, \theta_2, \dots, \overset{\downarrow}{\theta_i - \epsilon}, \dots)}{2\epsilon} \\ &\approx \underline{d\theta[i]} = \frac{\partial J}{\partial \theta_i} \quad | \quad d\theta_{\text{approx}} \approx d\theta \end{aligned}$$

¿Cómo hacemos para chequear que estos dos vectores sean los mismos? Se puede calcular la distancia euclídea entre ellos.

Check

$$\rightarrow \frac{\|d\theta_{\text{approx}} - d\theta\|_2}{\|d\theta_{\text{approx}}\|_2 + \|d\theta\|_2}$$

$$\varepsilon = 10^{-7}$$

$\approx 10^{-7}$	- great!
10^{-5}	
10^{-3}	- worry.

Si épsilon es del orden de 10^{-7} y la distancia euclídea la implementación probablemente es la correcta. Pero si la distancia es más grande en 2 órdenes de magnitud por lo menos, entonces conviene mirar si se cometió algún error (idealmente en las componentes individuales, ej. si el error está en db o en dw).

Notas en gradient checking

- No usar en training - solo para debug.
- si el algoritmo falla, mirar las componentes para identificar el bug.
- recordar regularización.
- no funciona con dropout: para implementar ambas se podría "apagar" dropout (`keep_prob=1.0`) chequear gradient checking y luego prenderlo.
- Correr bajo inicialización aleatoria; luego tal vez después del entrenamiento: suele suceder que no hay problema con $w, b \sim 0$, pero falla para errores más grandes.

Algoritmos de optimización

Dado que Deep Learning funciona mejor con Big Data, esto enlentece el computo de problemas. En esta sección se estudiarán diferentes técnicas para acelerar este proceso mediante algoritmos de optimización.

Mini-batch gradiente descendiente

Supongamos que se tiene la siguiente configuración. Para configurar gradiente descendiente se debería procesar todos los features y realizar un paso del gradiente. Luego procesar nuevamente para otro paso y así sucesivamente. En el caso de que $m = 5M$, esto se vuelve muy lento.

$$X = [x^{(1)} \ x^{(2)} \ x^{(3)} \ \dots \ \dots \ x^{(m)}]_{(n, m)}$$

$$Y = [y^{(1)} \ y^{(2)} \ y^{(3)} \ \dots \ \dots \ y^{(m)}]_{(1, m)}$$

Una solución es usar mini-batch, el cual consiste en separar el training set en por ejemplo dos trainings set.

Para el ejemplo anterior, se pueden separar conjuntos de sets de 1000 elementos. En total se tendrán 5000 subconjuntos, introducidos con la notación entre corchetes.

$$X = \underbrace{[x^{(1)} \ x^{(2)} \ x^{(3)} \ \dots \ x^{(1000)}]}_{\{n_x, m\}} \left| \underbrace{x^{(1001)} \ \dots \ x^{(2000)}}_{X^{\{2\}} \{n_x, 1000\}} \right. \dots \left| \underbrace{\dots \ x^{(m)}}_{X^{\{5,000\}} \{n_x, 1000\}} \right.$$

$$Y = \underbrace{[y^{(1)} \ y^{(2)} \ y^{(3)} \ \dots \ y^{(1000)}]}_{\{1, m\}} \left| \underbrace{y^{(1001)} \ \dots \ y^{(2000)}}_{Y^{\{2\}} \{1, 1000\}} \right. \dots \left| \underbrace{\dots \ y^{(n)}}_{Y^{\{5,000\}} \{1, 1000\}} \right.$$

What if $m = 5,000,000$?

5,000 mini-batches of 1,000 each

Mini-batch t : $\underline{X^{\{t\}}, Y^{\{t\}}}$

De esta manera la implementación vectorial del algoritmo mini-batch es la siguiente

for $t = 1, \dots, 5000 \{$

Forward prop on $X^{\{t\}}$.

$$Z^{(t)} = W^{(t)} X^{\{t\}} + b^{(t)}$$

$$A^{(t)} = g^{(t)}(Z^{(t)})$$

$$A^{(t)} = g^{(t)}(Z^{(t)})$$

Vectorized implementation
(1000 examples)

$$\text{Compute cost } J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{1000} f(y^{(i)}, A^{(t)}) + \frac{\lambda}{2 \cdot 1000} \sum_{j} \|W^{(t)}\|_F^2.$$

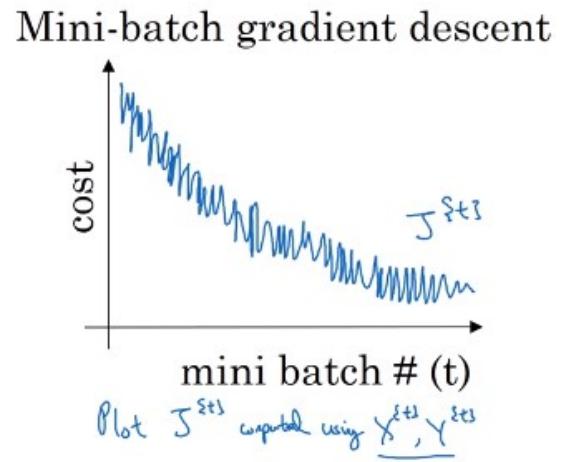
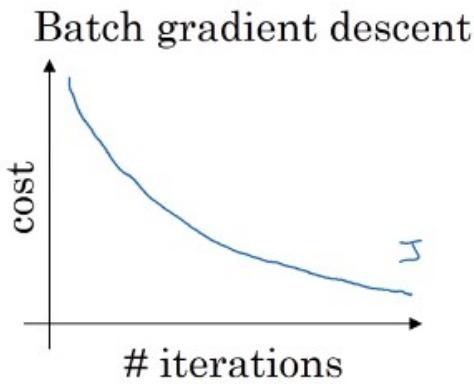
Backprop to compute gradients w.r.t $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

$$W^{(t)} = W^{(t)} - \alpha \delta W^{(t)}, \quad b^{(t)} = b^{(t)} - \alpha \delta b^{(t)}$$

}

A el loop exterior habría que agregarle otro loop del número de iteraciones.

Función de coste en función del número de iteraciones



Uno podría preguntarse, ¿qué tamaño del mini-batch elegir y cómo?

- Si el tamaño del mini-batch = m: batch es el de gradiente descendiente.
- si tamaño mini-batch = 1: gradiente descendiente estocástico. Cada ejemplo es un mini-batch.
- En la práctica el tamaño del mini-batch va a estar entre 1 y m.

Stochastic
gradient
descent

{
Use sparingly
for vectorization

In-between
(mini-batch size
not too big/small)

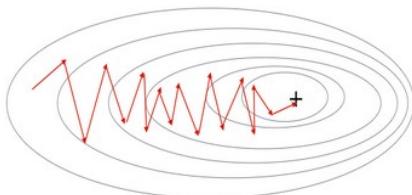
Faster learning.
• Vectorization.
(w/1000)
• Make passes without
processing entire training set.

Batch
gradient descent
(mini-batch size = m)

↓
Too long
per iteration

Andrew Ng

Stochastic Gradient Descent



Gradient Descent

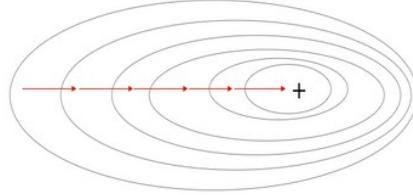
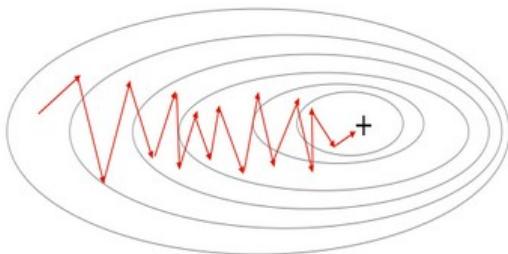


Figure 1 : SGD vs GD

"+" denotes a minimum of the cost. SGD leads to many oscillations to reach convergence. But each step is a lot faster to compute for SGD than for GD, as it uses only one training example (vs. the whole batch for GD).

Stochastic Gradient Descent



Mini-Batch Gradient Descent

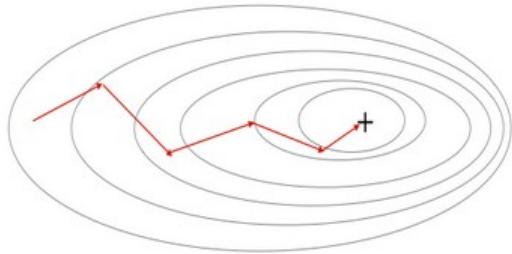


Figure 2 : SGD vs Mini-Batch GD

"+" denotes a minimum of the cost. Using mini-batches in your optimization algorithm often leads to faster optimization.

¿Qué tamaño se elige entonces?

En la práctica este es otro hiper parámetro el cual se debe encontrar.

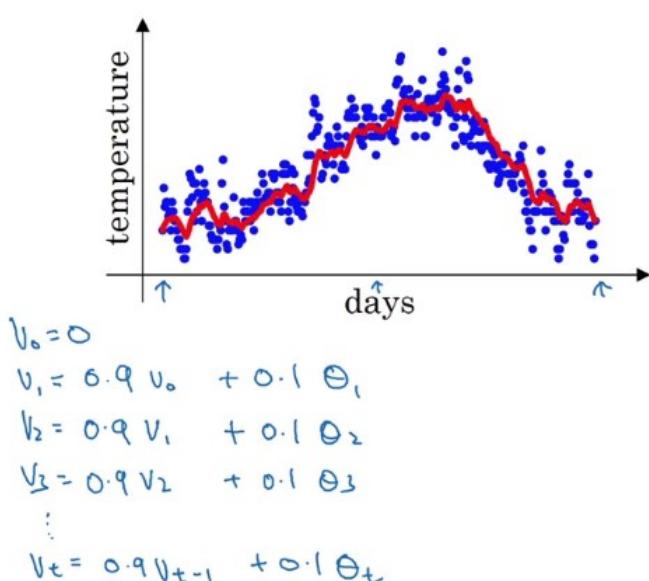
Si el training set es pequeño ($m \leq 2000$) conviene utilizar batch gradiente descendiente. Tamaños usuales para el mini-batch suelen ser potencias de 2: $64(2^6)$, $128(2^7)$, $256(2^8)$, $512(2^9)$. Conviene chequear que el tamaño del mini-batch puede almacenarse en la memoria del CPU/GPU.

Exponentially weighted averages

Para introducir algoritmos de optimización más avanzados, primero hay que explicar el concepto de promedios pesados exponenciales.

Para ello comencemos con un ejemplo de las temperaturas anuales de Londres.

$$\begin{aligned}\theta_1 &= 40^{\circ}\text{F} & 4^{\circ}\text{C} & \leftarrow \\ \theta_2 &= 49^{\circ}\text{F} & 9^{\circ}\text{C} \\ \theta_3 &= 45^{\circ}\text{F} & \vdots \\ &\vdots \\ \theta_{180} &= 60^{\circ}\text{F} & 15^{\circ}\text{C} \\ \theta_{181} &= 56^{\circ}\text{F} & \vdots\end{aligned}$$



El promedio ponderado se calcula para el primer día $V_0 = 0$. Para el segundo, se supone que hay un peso del 90% respecto del valor del día anterior V_0 mas una proporción restante del 10% a determinar por θ_1 .

$$V_t = \underline{\beta} \underline{V_{t-1}} + \underline{(1-\beta) \theta_t}$$

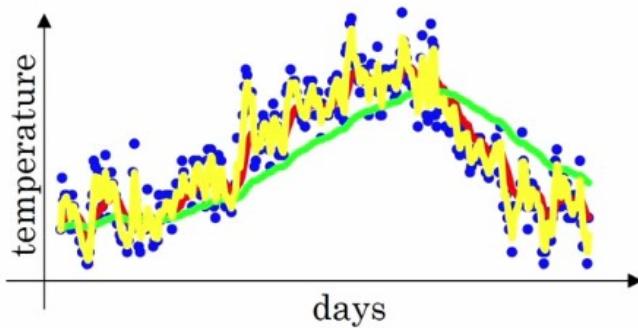
$\beta = 0.9$: ≈ 10 days' temper.

$\beta = 0.98$: ≈ 50 days

$\beta = 0.5$: ≈ 2 days

V_t es aproximadamente
después de
 $\rightarrow \approx \frac{1}{1-\beta}$ days'
temperatura.

$$\frac{1}{1-0.98} = 50$$



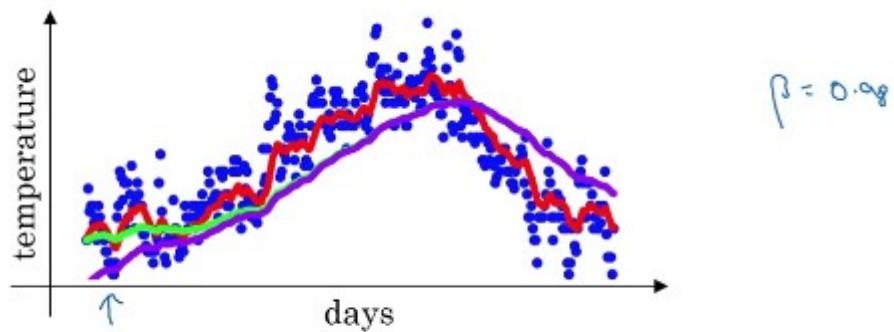
De manera general la ecuación que relaciona la temperatura para un día particular se expresa en la figura superior. Donde los parámetros a determinar son β y θ . Se puede notar que para β más grandes el valor de la temperatura del día anterior es muy importante y los cambios por nuevos puntos no son tan bruscos.

La línea roja equivale a $\beta = 0.9$. La verde a $\beta = 0.89$ y $\beta = 0.5$ a la línea amarilla.

Implementación computacional

$\rightarrow V_0 = 0$
Repetir {
 Get next θ_t
 $V_t := \beta V_{t-1} + (1-\beta) \theta_t$ ←
}

Corrección del sesgo (bias)



$$\rightarrow v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

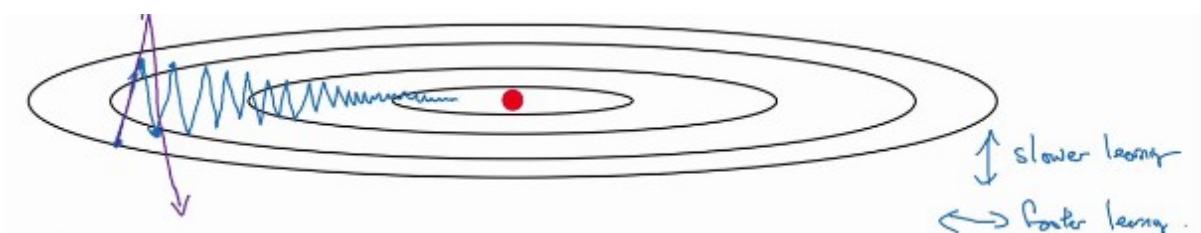
En el caso de $\beta = 0.98$, se debería obtener la línea verde, pero usando la ecuación superior se obtiene la púrpura. La corrección del sesgo permite calcular el promedio de manera más precisa. Simplemente consiste en ver que a tiempos pequeños hay un problema debido a que $V_0 = 0$, generalmente no es una buena implementación. Mientras que a tiempos

mayores esto se soluciona. De esta manera se aplica una corrección $\frac{v_t}{1 - \beta^t}$. La misma actúa cuando los tiempos son pequeños (ej., $1 - (0.98)^2 = 0.0396$), mientras que es prácticamente nula cuando t es mayor (ej., $1 - (0.98)^{100} = 1 - 0 = 1$).

Utilicemos ahora lo que se vio previamente como corrección del sesgo para generar mejores algoritmos de optimización.

Gradiente descendente con momentum

Generalmente funciona más rápido que el algoritmo de gradiente descendente estándar. En una oración, la idea básica es calcular el promedio pesado exponencial de los gradientes para luego usar los gradientes para actualizar los pesos.



On iteration t :

Compute dW, db on the current mini-batch

$$v_{dW} = \beta v_{dW} + (1 - \beta) dW$$

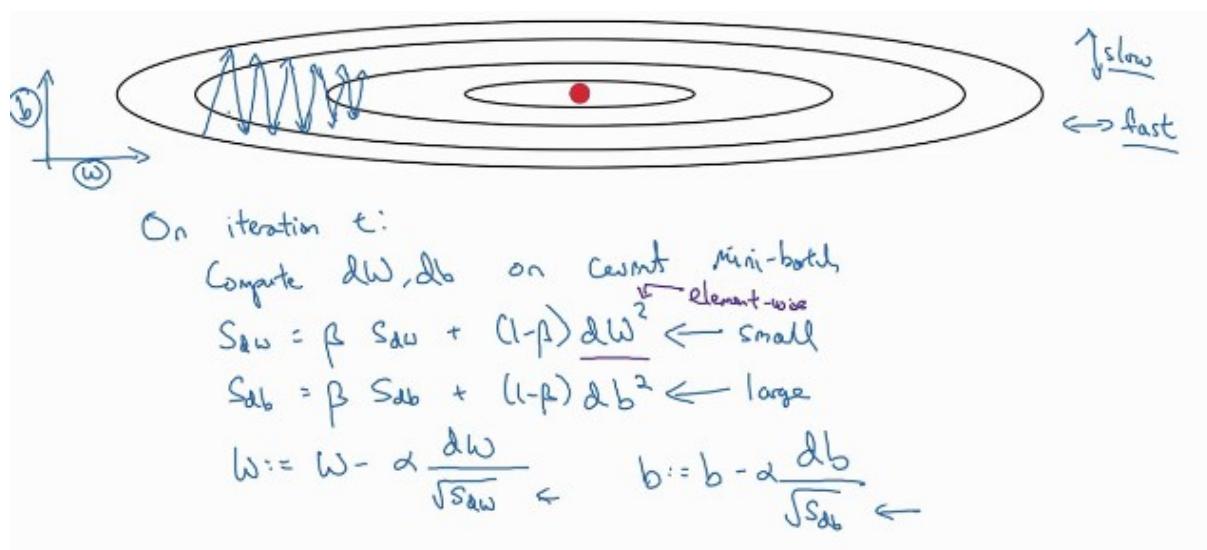
$$v_{db} = \beta v_{db} + (1 - \beta) db$$

$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

Hyperparameters: α, β $\beta = 0.9$

Básicamente este algoritmo lo que hace es suavizar el movimiento lateral que es el que se quiere que converja más rápidamente. El algoritmo toma un camino más directo.

Otro algoritmo: RMSProp (RootMeanSquareProp)



Ahora combinaremos los algoritmos de RMSProp con el Gradiente descendente momentum.

Algoritmo de optimización de Adam (Adaptive moment estimation)

Este algoritmo a diferencia de los anteriores, funciona correctamente en un amplio rango de arquitecturas.

$$V_{dw} = 0, S_{dw} = 0. \quad V_{db} = 0, S_{db} = 0$$

On iteration t :

Compute $\delta w, \delta b$ using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) \delta w, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) \delta b \leftarrow \text{"moment"} \beta_1$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) \delta w^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) \delta b^2 \leftarrow \text{"RMSprop"} \beta_2$$

$$V_{dw}^{\text{corrected}} = V_{dw} / (1 - \beta_1^t), \quad V_{db}^{\text{corrected}} = V_{db} / (1 - \beta_1^t)$$

$$S_{dw}^{\text{corrected}} = S_{dw} / (1 - \beta_2^t), \quad S_{db}^{\text{corrected}} = S_{db} / (1 - \beta_2^t)$$

$$w := w - \alpha \frac{V_{dw}^{\text{corrected}}}{\sqrt{S_{dw}^{\text{corrected}}} + \epsilon} \quad b := b - \alpha \frac{V_{db}^{\text{corrected}}}{\sqrt{S_{db}^{\text{corrected}}} + \epsilon}$$

Elección de hiper parámetros: generalmente con la optimización de Adam se definen valores para β y ϵ y se trata de estimar el mejor valor de α entre un rango de valores.

α : needs to be tune

$$\beta_1: 0.9 \quad (\delta w)$$

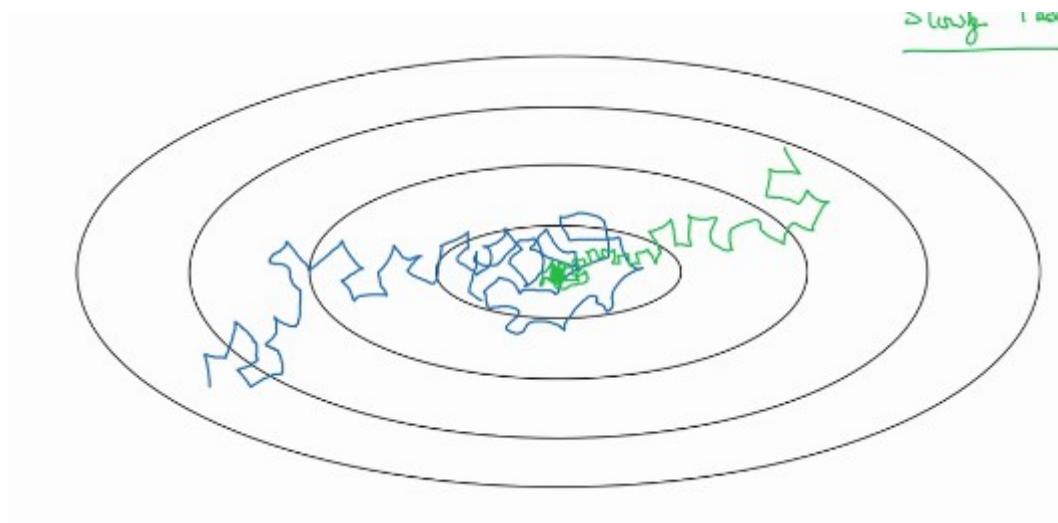
$$\beta_2: 0.999 \quad (\delta w^2)$$

$$\epsilon: 10^{-8}$$

Decaimiento del ritmo de aprendizaje (α)

La idea detrás de este método es que, dado un valor de α fijo el algoritmo podría quedarse dando saltos alrededor del punto de equilibrio no tan cerca de él (línea azul).

El decaimiento propuesto es del tipo exponencial, inicialmente siendo grande dado que se está lejos de la solución optima, y a medida que se va acercando α va siendo cada vez más pequeño (línea roja).



Definición de epoch y decaimiento del ritmo de aprendizaje

Learning rate decay

1 epoch = 1 pass through data.

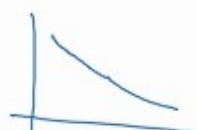
$$\alpha = \frac{1}{1 + \underbrace{\text{decay-rate} * \text{epoch-num}}_{\downarrow}} \alpha_0$$

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
.	.



$$\alpha_0 = 0.2$$

$$\text{decay-rate} = 1$$



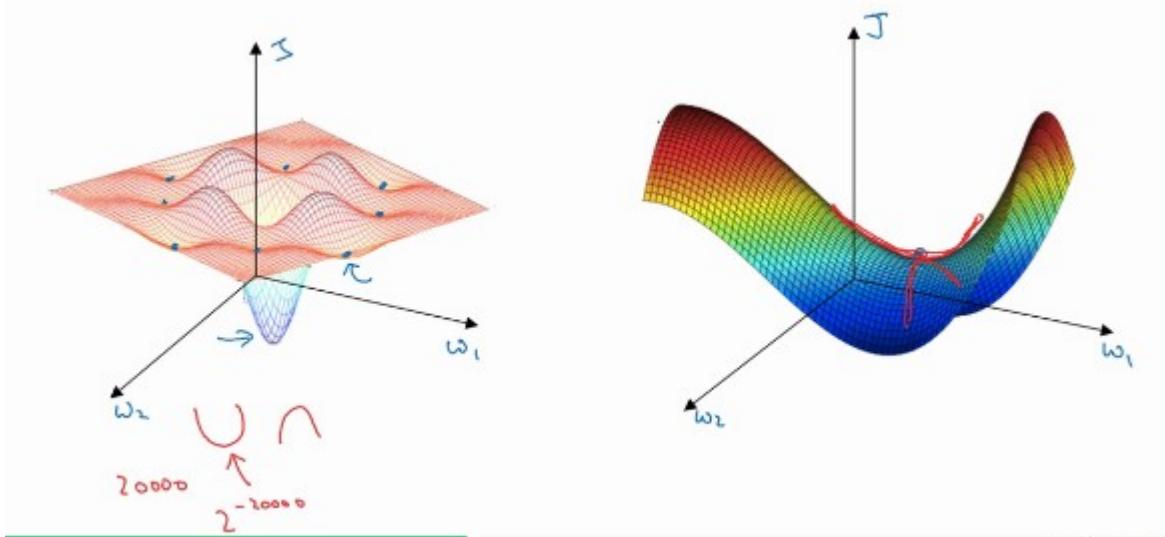
La definición del decaimiento es arbitraria, se podría haber elegido alguna otra como el decaimiento tipo escalera discreto.

formula $\left\{ \begin{array}{l} d = 0.95^{\text{epoch_num}} \cdot d_0 \quad - \text{exponentially decay.} \\ d = \frac{k}{\sqrt{\text{epoch_num}}} \cdot d_0 \quad \text{or } \frac{k}{\sqrt{t}} \cdot d_0 \end{array} \right.$

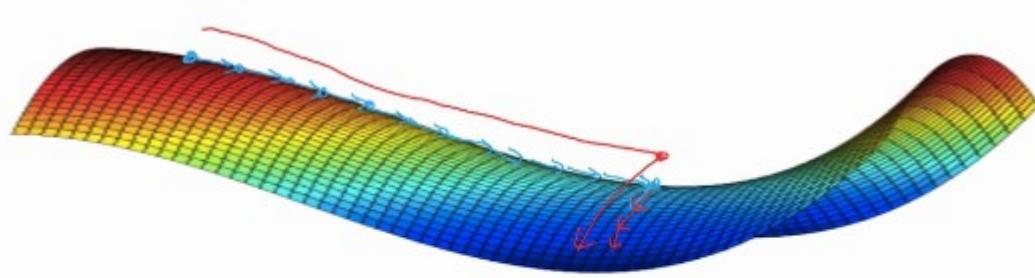
diente de escalera

Óptimos locales en redes neuronales

Generalmente en la estimación de mínimos globales en redes neuronales, no tenemos mínimos locales sino puntos silla (saddle points).



Resulta que los problemas de las redes neuronales no son los mínimos locales sino los plateaus donde el gradiente se vuelve pequeño durante un amplio rango de parámetros.



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow

Hyperparameter tuning, Batch Normalization and Programming Frameworks

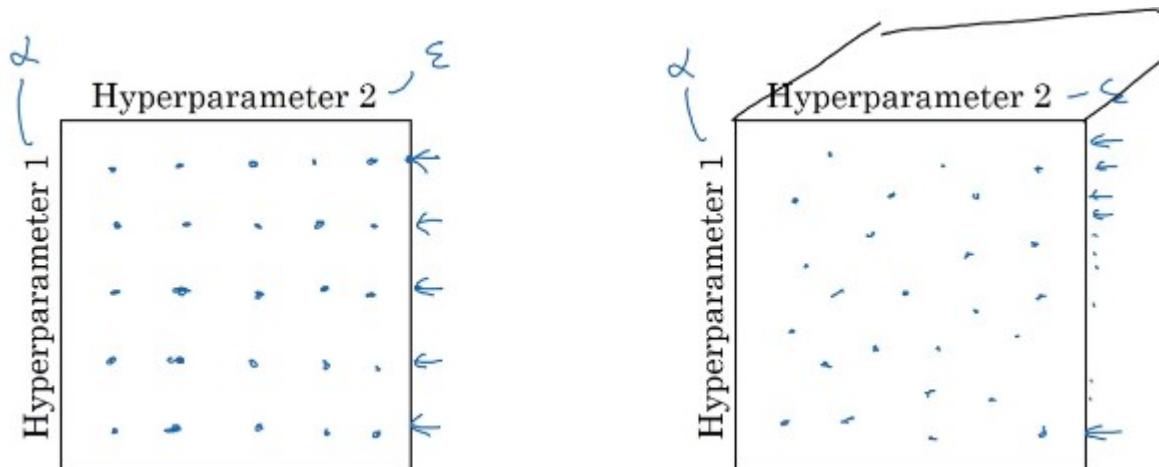
Tuning process

¿Cómo elegimos adecuadamente al conjunto de hiper parámetros: $\alpha, \beta, (\beta_1, \beta_2, \epsilon)$ para Adam, número de capas, número de unidades ocultas, decaimiento de la tasa de aprendizaje, tamaño del mini-batch?

No todos son igual de importantes. α siendo el más importante. β , el número de unidades ocultas o el tamaño del mini-batch las siguientes:

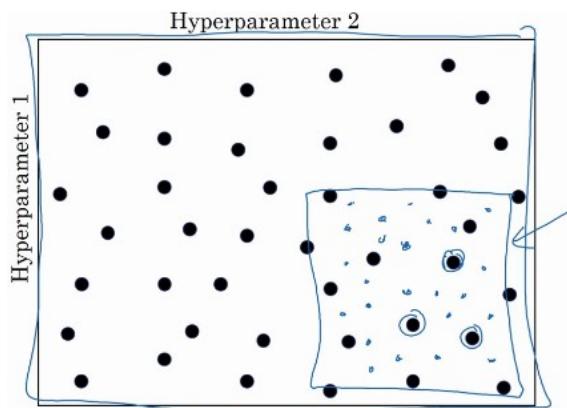
- 1) Elegir valores aleatorios de hiper parámetros: no usar una grilla!!

Cuando el número de hiper parámetros es pequeño la grilla puede funcionar bien, sino no.



- 2) Coarse to fine

Enfocarse más en una región particular del espacio de parámetros.



Usar la escala adecuada para elegir hiperparámetros

Por ejemplo, para alfa entre 10^{-4} y 1 conviene elegir en escala logarítmica.

$$\alpha = 0.0001, \dots, 1$$

$\frac{0.0001}{10^{-4}}$ 0.001 0.01 0.1 1 10^0

$r = -4 * \text{np.random.rand}() \leftarrow r \in [-4, 0]$

$\alpha = 10^r \leftarrow 10^{-4} \dots 10^0$

Detailed description: This block shows a logarithmic scale for the parameter alpha. It starts at 0.0001 and ends at 1 . The intermediate values 0.001 , 0.01 , 0.1 , and 10^0 are also shown. Below this scale, a random number r is generated from the range $[-4, 0]$. This r is then used to calculate $\alpha = 10^r$, resulting in a value between 10^{-4} and 10^0 .

Hiperparámetros para promedios pesados exponencialmente

Por ejemplo, β yendo desde 0.9 a 0.999. Conviene hacer algo similar a lo visto anteriormente.

$$\beta = 0.9 \dots 0.999$$

\downarrow

10 \downarrow
 1000

$$1-\beta = 0.1 \dots 0.001$$

$$\begin{aligned} & \overbrace{\quad \quad \quad \quad \quad}^{\text{1000}} \\ & 0.9 & 0.999 \\ & \overbrace{\quad \quad \quad \quad}^{6.9} \quad \overbrace{\quad \quad \quad}^{6.99} \quad \overbrace{\quad \quad \quad}^{6.999} \\ & 0.1 & 0.01 & 0.001 \\ & \overbrace{\quad \quad \quad}^{10^{-1}} & \overbrace{\quad \quad \quad}^{10^{-2}} & \overbrace{\quad \quad \quad}^{10^{-3}} \\ & r \in [-3, -1] \\ & 1-\beta = 10^r \\ & \beta = 1 - 10^r \end{aligned}$$

Normalización Batch

Este algoritmo hace la búsqueda de hiper parámetros mucho más simple, haciendo la red neuronal más robusta. La elección de hiper parámetros es hecha en un rango mucho más amplio provocando que se pueda entrenar más fácilmente redes neuronales más profundas.

Normalizando activaciones en una red

Ya se ha visto previamente como realizar una normalización de los features de entrada y el efecto que provoca un cambio en la regresión. Pero, ¿cómo varia un modelo más profundo?

¿Se podrán normalizar las capas ocultas de manera que w y b se calculen más rápidamente?

Implementar Batch Norm

Básicamente lo que hace este algoritmo es normalizar los z en capas ocultas a un valor tal que z_i moño sea igual a z_i .

Given some intermediate values in NN $\underbrace{z^{(1)}, \dots, z^{(n)}}_{[z]^{(i)}}$

$\mu = \frac{1}{m} \sum_i z^{(i)}$
 $\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$
 $z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$
 $\hat{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$

If $\gamma = \sqrt{\sigma^2 + \epsilon} \leftarrow$

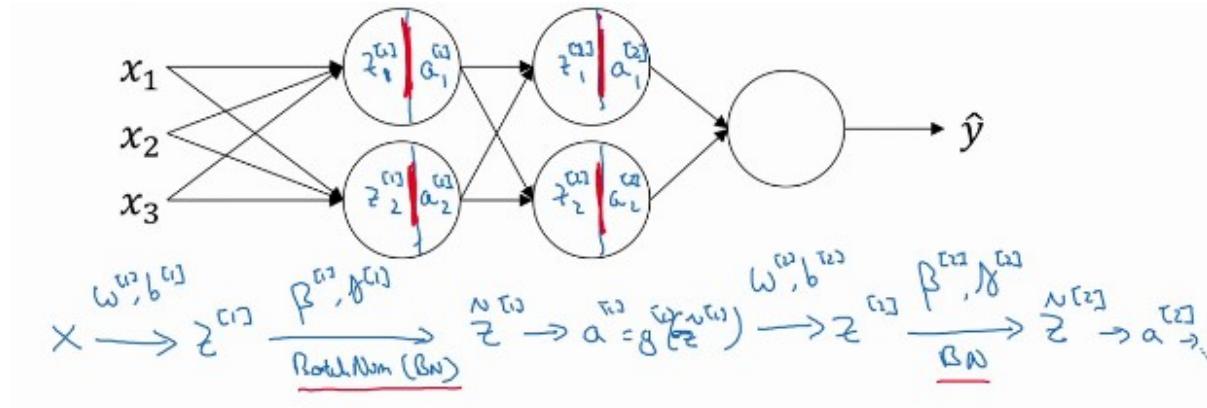
$\beta = \mu \leftarrow$

then $\hat{z}^{(i)} = z^{(i)}$

learnable parameters of model.

Agregando Batch Norm a una red neuronal

Primer se comienza calculando $z^{[1]}$ dadas $w^{[1]}$ y $b^{[1]}$, luego se normaliza usando Batch Norm (BN) dados los parámetros $\beta^{[1]}$ y $\gamma^{[1]}$. Obtenida $z_{\tilde{1}}$ se calcula $a^{[1]}$ evaluando $z_{\tilde{1}}$ con la función de activación. Esto se realiza para todas las capas ocultas.



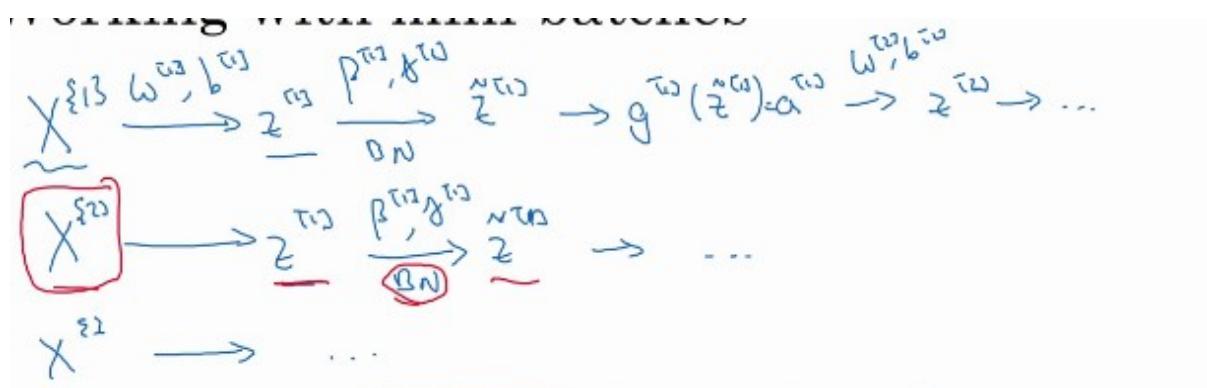
Los parámetros a ajustar serán el doble.

Parámetros: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \dots, w^{[L]}, b^{[L]}$,
 $\rightarrow \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[L]}, \gamma^{[L]}$

En TensorFlow esto se implementaría con **`tf.nn.batch_normalization`**.

Trabajando con mini-batch

El proceso es similar al anterior pero trabajando por separado con cada mini-batch.



Por otra parte, se puede decir que en realidad el parámetro b es innecesario calcularlo.

Parámetros: $w^{[L]}, \gamma^{[L]}, \beta^{[L]}, \hat{z}^{[L]}$.

$\hat{z}^{[L]}$
l_i

$$\begin{aligned}\hat{z}^{[L]} &= w^{[L]} a^{[L-1]} + b^{[L]} \\ z^{[L]} &= w^{[L]} a^{[L-1]} \\ \hat{z}_{\text{norm}}^{[L]} &= \hat{\gamma}^{[L]} \hat{z}^{[L]} + \beta^{[L]}\end{aligned}$$

Andrew N

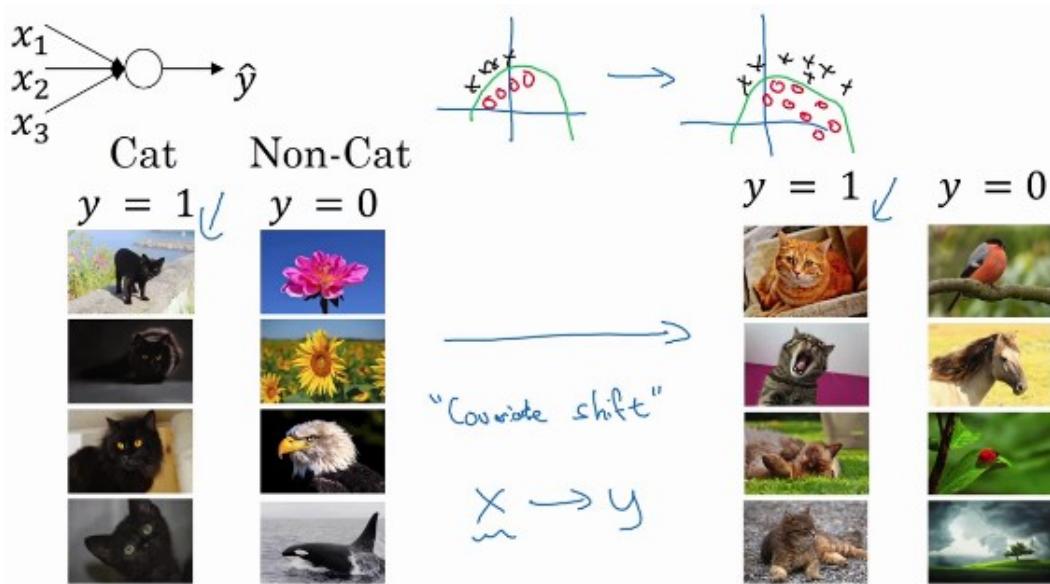
Implementando gradiente descendiente

Para el backpropagation podría utilizarse con momentum, RMSProp o Adam.

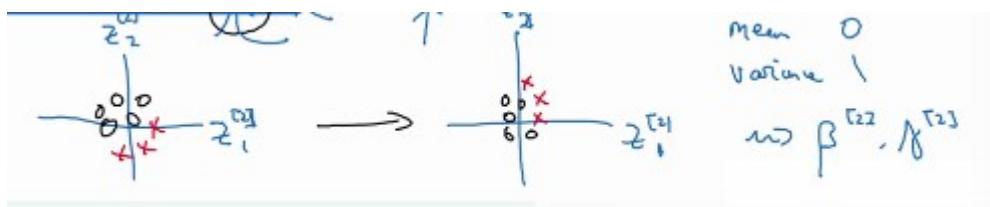
for $t = 1 \dots \text{numMiniBatches}$
 Compute forward pass on $X^{[t]}$.
 In each hidden layer, use BN to replace $\hat{z}^{[L]}$ with $\hat{z}_{\text{norm}}^{[L]}$.
 Use backprop to compute $d\hat{z}^{[L]}, d\hat{\gamma}^{[L]}, d\hat{\beta}^{[L]}, d\hat{w}^{[L]}$
 Update parámetros $\left. \begin{array}{l} w^{[L]} := w^{[L]} - \alpha d\hat{w}^{[L]} \\ \beta^{[L]} := \beta^{[L]} - \alpha d\hat{\beta}^{[L]} \\ \gamma^{[L]} := \dots \end{array} \right\} \leftarrow$
 Works w/ momentum, RMSprop, Adam.

Veamos porque es que realmente Batch Normalization funciona y hace que el aprendizaje sea más rápido.

Para ello consideremos que sucede en el caso del cambio de input features. Por ejemplo, gatos negros y luego gatos que no sean negros. Esto puede provocar que la red no funcione correctamente.



Básicamente Batch Norm lo que hace es tratar que los parámetros no varíen demasiado ante el cambio de valores de entrada. Los parámetros β y γ deberían fluctuar muy poco.



Batch norm aplicado a mini-batch actúa como regularización:

- cada mini batch es escalaeado por el valor medio/varianza en ese mini batch.
- esto agrega algo de ruido a los valores z de ese mini batch. Similarmente a dropout agrega ruido a cada activación de cada capa oculta.
- Esto tiene un efecto de regularización.

Dado que es pequeño no es un efecto notorio de regularización, para ello conviene implementar dropout.

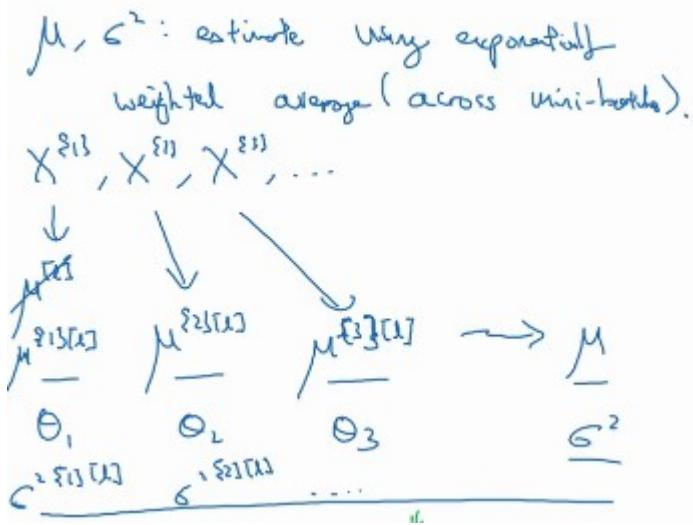
No es recomendable utilizarlo ya que no es la idea del mismo.

Resumiendo Batch norm

$$\begin{aligned}\mu &= \frac{1}{m} \sum_i z^{(i)} \\ \sigma^2 &= \frac{1}{m} \sum_i (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \\ \tilde{z}^{(i)} &= \gamma z_{\text{norm}}^{(i)} + \beta\end{aligned}$$

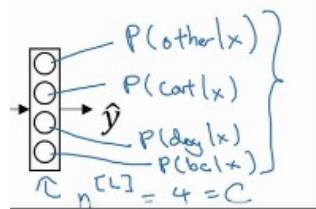
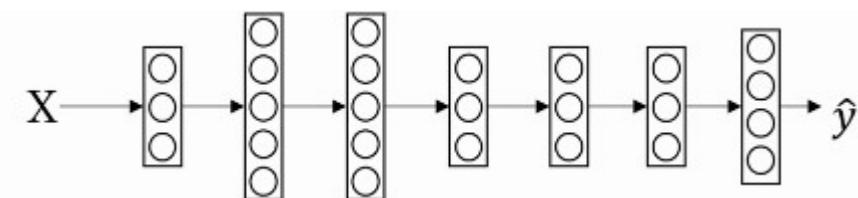
Dado que para calcular z tilde se necesita μ y σ y, para que estos sean calculados con precisión deben ser calculados con un número considerable de ejemplos. Para hacer esto hace lo siguiente: se calcula μ para el conjunto de mini-batches del training set y de ese conjunto se calcula un estimador. Esto mismo se hace para σ .

Luego se implementa el promedio pesado exponencial para utilizar en el test set, para hacer la normalización batch.



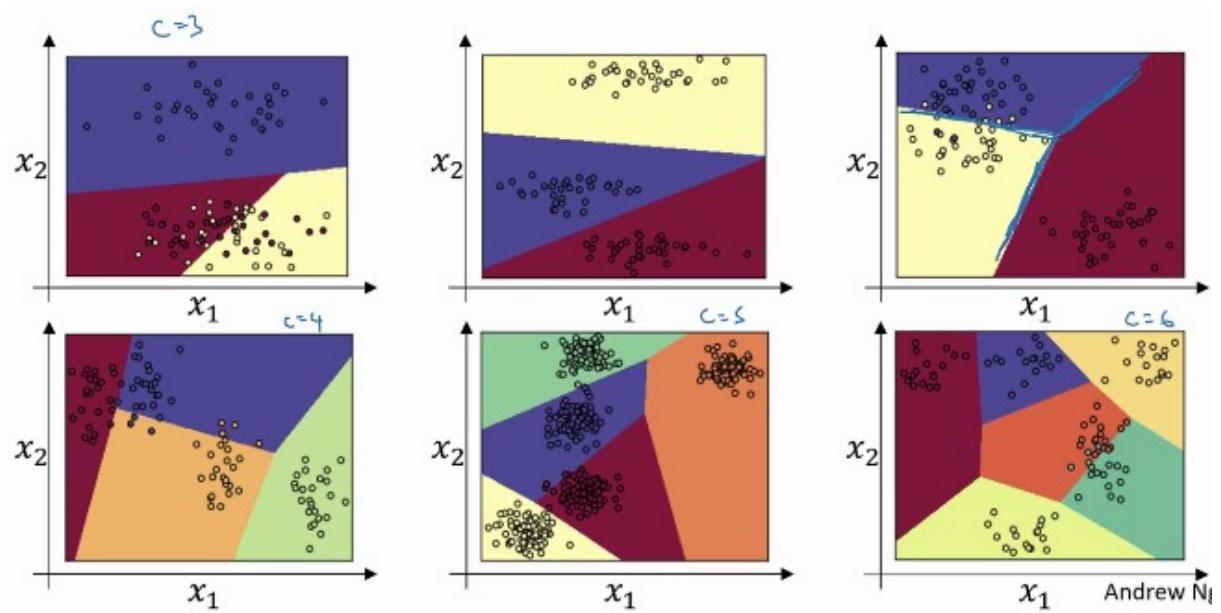
Regresión Softmax

Si en lugar de requerir una clasificación binaria se requiere una clasificación multi conviene usar softmax. Por ejemplo, si se quiere clasificar perros, gatos, pollitos y los que no lo son.



Veamos algunos ejemplos sin capas ocultas, veamos que los contornos son rectos.

$$x_1 \quad x_2 \rightarrow \boxed{\begin{matrix} 0 \\ 0 \\ 0 \end{matrix}} \rightarrow \hat{y}$$



Para tener contornos más complejos se deberían agregar capas ocultas.

Entrenar un clasificador Softmax

Si el número de salidas en softmax = 2, se obtiene una regresión logística.

Deep Learning Frameworks

Ejemplos de estos frameworks:

Caffe/Caffe2, CNTK, DL4J, Keras, Lasagne, mxnet, PaddlePaddle, TensorFlow, Theano, Torch.

Elección:

- Facilidad de programación (desarrollo e implementación).
- Velocidad para correr.
- Que sea verdaderamente open source (con una gran comunidad que lo mejore).

Tensor Flow

Ejemplo de motivación: minimizar la función de coste. De manera más general, se tendrá una función $J(w,b)$.

$$J(\omega) = \frac{\omega^2 - 10\omega + 25}{(\omega - 5)^2}$$

$$\omega = 5$$

```

1 import numpy as np
2 import tensorflow as tf
3
4 w = tf.Variable(0, dtype=tf.float32) # Initialize to zero
5 cost = tf.add(tf.add(w**2, tf.multiply(-10., w)), 25) # Define cost
6 # cost = w**2 - 10*w + 25 also works. Some basic ops are overloaded.
7 train = tf.train.GradientDescentOptimizer(0.01).minimize(cost) # Define training method
8
9 init = tf.global_variables_initializer()
10 session = tf.Session()
11 session.run(init)
12 print(session.run(w)) -> 0.0
13
14 # Como no se corrió nada todavía se obtiene cero.
15
16 session.run(train) # correr un paso de gradient descent -> 0.1
17
18 for i in range(1000):
19     session.run(train)
20 print(session.run(w)) # se obtiene el valor deseado 4.999 cercano a 5.

```

```

import numpy as np
import tensorflow as tf

coefficients = np.array([[1], [-20], [25]])

w = tf.Variable([0], dtype=tf.float32)
x = tf.placeholder(tf.float32, [3,1])
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]    # (w-5)**2
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
init = tf.global_variables_initializer()

session = tf.Session()                         with tf.Session() as session:
session.run(init)                            session.run(init)
print(session.run(w))                        print(session.run(w))

for i in range(1000):
    session.run(train, feed_dict={x:coefficients})
print(session.run(w))

```

Estructurando proyectos de Machine Learning

¿Por qué estrategia de Machine Learning?

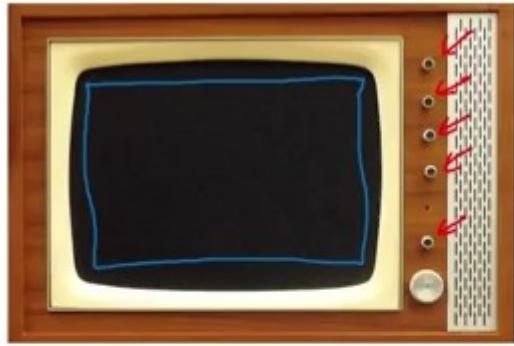
Ideas:

- Collect more data
 - Collect more diverse training set
 - Train algorithm longer with gradient descent
 - Try Adam instead of gradient descent
 - Try bigger network
 - Try smaller network
 - Try dropout
 - Add L_2 regularization
 - Network architecture
 - Activation functions
 - # hidden units
 - ...
- Andrew Ng

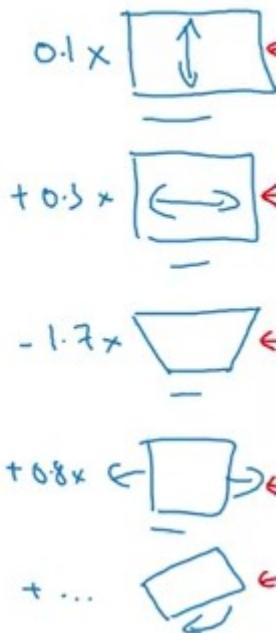
Para no perder tiempo con esto, veremos una cuestión de ideas a continuación.

Ortogonalización

¿Qué hiper parámetro tunear para obtener un efecto deseado? Veamos un ejemplo con una televisión: Las televisiones viejas tenían unas manijas que permitían modificar por separado la altura de la posición de la pantalla, el alto, el ancho, el trapezoide, la rotación, etc. En el caso de que en lugar de esas manijas hubiera una sola que manejase 0.1 por el alto + 0.3 por el ancho, etc. la misma sería muy difícil de calibrar.



Orthogonalization



→ *Steering*
→ *Acceleration*
→ *Braking*

De manera similar se puede considerar un auto. El mismo posee un control para la dirección, el volante, y otros dos para la aceleración y el freno.

En lugar de estos controles se podrían definir dos como los que se muestran debajo. Pero, al ser una combinación del

ángulo y de la velocidad el manejo del auto seguramente será dificultosa. Es por ello que idealmente uno querría tener un mecanismo que separe ambas contribuciones.

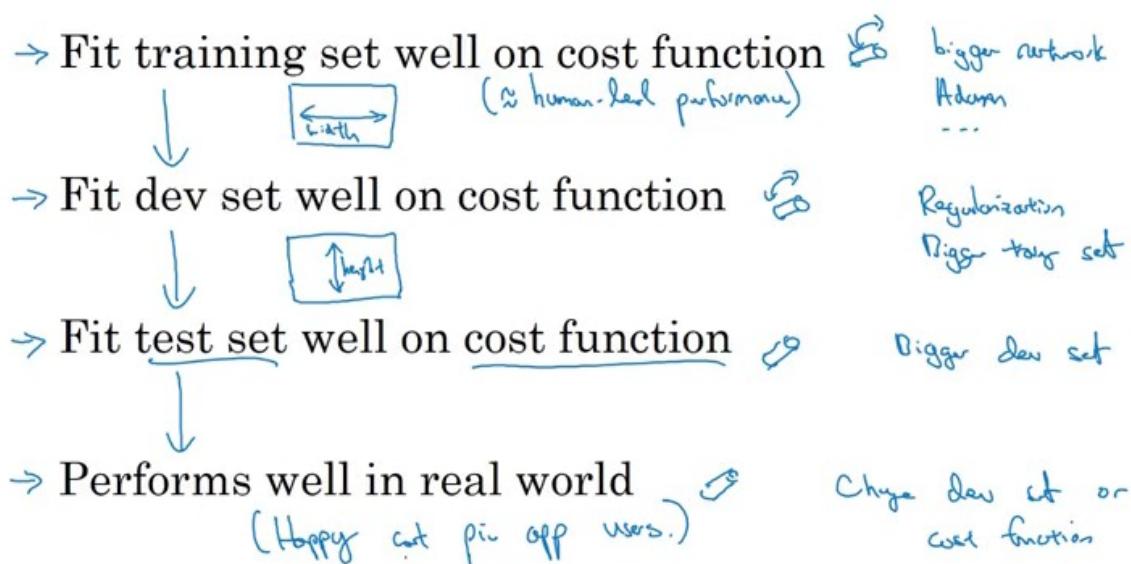
$$\begin{aligned} & \underline{0.3 \times \text{angle} - 0.8 \times \text{speed}} \\ & > 2 \times \text{angle} + 0.9 \times \text{speed} \end{aligned}$$

A.

¿Cómo se extraña esto en proyectos de Machine Learning?

En ML hay cuatro pasos fundamentales:

1. Ajustar el conjunto de entrenamiento con la función de coste: Si esto no se cumple se puede intentar agrandar la red neuronal o probar el Método de Adam. Esto es análogo a ajustar una perilla de la televisión, ya que hacer una de estas modificaciones mejorara más que nada el ajuste del conjunto de entrenamiento sobre la función de coste.
2. Ajustar el conjunto de desarrollo con la función de coste: Si el conjunto de desarrollo no es ajustado correctamente por la función de coste conviene implementar regularización o probar un conjunto de entrenamiento más grande.
3. Ajustar el conjunto de testeo con la función de coste: si el conjunto de entrenamiento no se ajusta correctamente a la función de coste conviene agrandar el conjunto de desarrollo.
4. Buen rendimiento en el mundo real: en este caso quizás convenga cambiar el conjunto de desarrollo o la función de coste.



Métrica de evaluación de un número único

Usualmente se suelen utilizar dos números para evaluar el rendimiento de un dado algoritmo de ML: precisión y recall.

→ Of examples recognized as cat,
what % actually are cats?
→ what % of actual cats
are correctly recognized

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

Precisión: de los ejemplos reconocidos como gatos, que porcentaje son gatos realmente.
 Recall: que porcentaje de gatos son correctamente identificados.

En este ejemplo el clasificador es mejor para el indicador Recall, mientras que el clasificador B lo hace para la precisión. El hecho de usar dos indicadores hace difícil determinar con cuál de los dos conviene seguir estudiando.

Es por ello que se recomienda utilizar una única métrica de evaluación: en la literatura de Machine Learning se suele utilizar el score F1 que combina ambas métricas.

Classifier	Precision	Recall	F1 Score
A	95%	90%	92.4%
B	98%	85%	91.0%

El mismo se define como el promedio harmónico entre la precisión y el recall.

$F_1 \text{ score} = \text{"Average" of } P \text{ and } R.$
 $\left(\frac{2}{\frac{1}{P} + \frac{1}{R}} \cdot \text{"Harmonic mean"} \right)$

Supongamos ahora que se tiene la siguiente tabla donde acorde a la región se registran el error de cada clasificador. Es ideal calcular el promedio teniendo en cuenta que es un estimador aceptable y a partir de él decidir cuál de todos es el mejor. En este caso se toma al C.

Algorithm	US	China	India	Other	Average
A	3%	7%	5%	9%	6%
B	5%	6%	5%	10%	6.5%
C	2%	3%	4%	5%	3.5%
D	5%	8%	7%	2%	5.25%
E	4%	5%	2%	4%	3.75%
F	7%	11%	8%	12%	9.5%

Hay veces en las que conviene tener registro de otros parámetros para determinar con qué clasificador quedarse. Consideremos por ejemplo tres clasificadores donde se registra la exactitud y el tiempo de cómputo para cada uno de ellos.

Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

Se puede definir una función lineal que nos ayude a determinar con cuál de los dos quedarnos. Pero no parece la mejor manera.

$$\text{Cost} = \underline{\text{accuracy}} - 0.5 \times \underline{\text{running Time}}$$

Uno podría querer maximizar la exactitud sujeto a la condición de que el tiempo de computo sea menor a 100ms.

Distribución de los conjuntos de entrenamiento/desarrollo/testeo

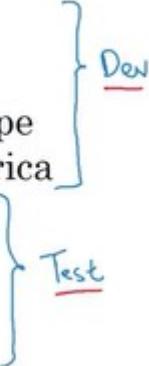


dev set
+
Metric

Clasificación de gatos con el conjunto de desarrollo/ testeo

Suponiendo que se obtiene un conjunto de datos de varias regiones. Elegir los conjuntos de acuerdo a las regiones de las mismas es una mala idea ya que las mismas van a tener distintas distribuciones de probabilidad.

Regions:

- US
 - UK
 - Other Europe
 - South America
 - India
 - China
 - Other Asia
 - Australia
- 

Tamaño del conjunto de desarrollo y testeo

Recién se dijo que ambos deben provenir de la misma distribución, pero, ¿cómo debe ser el tamaño de los mismos?

En la era del Big Data se utiliza alrededor del 98% de datos para entrenamiento.

Set your test set to be big enough to give high confidence
in the overall performance of your system.

¿Cuándo cambiar el conjunto de testeo/desarrollo?

Consideremos por ejemplo dos algoritmos A y B, en el que el A posee una métrica con menor error, pero muestra contenido inadecuado siendo la preferida para el usuario.

En estos casos es recomendable cambiar la métrica por la que se muestra debajo añadiendo pesos de acuerdo al contenido no deseado que aparece.

Cat dataset examples

Metric + Dev : Prefer A
You/uses : Prefer B.

Metric: classification error

Algorithm A: 3% error → pornographic

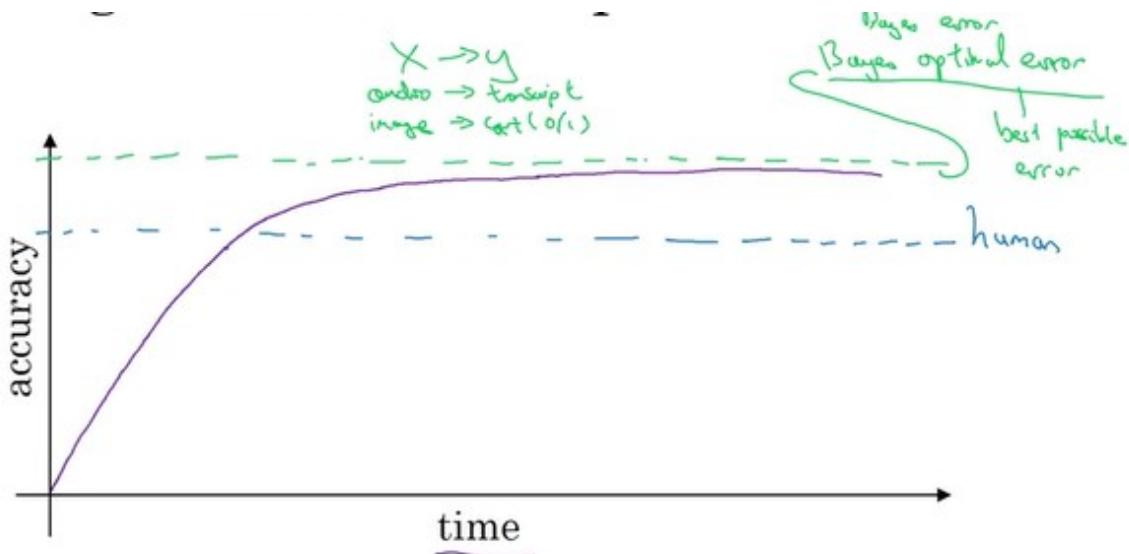
✓ Algorithm B: 5% error

$$\text{Error} = \frac{1}{\sum_{i=1}^{m_{\text{dev}}} w^{(i)}} \sum_{i=1}^{m_{\text{dev}}} \left[\frac{y_{\text{pred}}^{(i)} + y^{(i)}}{\text{predicted value (0/1)}} \right]$$

$\rightarrow w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \rightarrow \text{non-porn} \\ 10 & \text{if } x^{(i)} \rightarrow \text{porn} \end{cases}$

Performance a nivel humano

El error óptimo de Bayes es el máximo valor que podría tomar una función que mapea X en Y. Por ejemplo, dado un audio, este podría ser transcripto con una precisión máxima debido al ruido, o una imagen borrosa no sería identificable como un gato.



Existen técnicas que se pueden utilizar para llegar al nivel humano pero que no mejoran al pasar este nivel, como por ejemplo

- Obtener datos etiquetados por humanos
- obtener información desde un análisis manual del error. ¿Por qué una persona hizo algo bien?

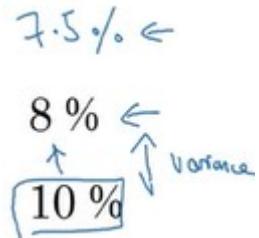
- Mejor análisis del bias/varianza.

Volviendo a la clasificación de gatos. Consideremos el siguiente ejemplo donde el conjunto de datos de entrenamiento posee un error muy grande respecto al humano.

Humans	<u>1%</u>
Training error	<u>8%</u>
Dev error	<u>10%</u>

En este sentido conviene reducir el bias. Entonces uno quiere hacer cosas como probar redes neuronales más grandes o correr conjunto de entrenamiento más tiempo.

Ahora supongamos que los humanos tienen un error del 7.5% debido a problemas de resolución, por ejemplo. Uno querría ahora centrarse en disminuir la varianza.



A la diferencia de error entre el conjunto de entrenamiento y el rendimiento humano se lo llama **Bias evitable (Avoidable Bias)**. Mientras que a la diferencia de error entre el conjunto de entrenamiento y el conjunto de desarrollo se lo llama varianza. En el segundo ejemplo el primero es de 0.5% y el segundo del 2%. Dado un indicador de la conveniencia en disminuir la varianza.

Entendimiento del rendimiento a nivel humano

Error humano como proxy del error de Bayes

Dado el siguiente ejemplo, ¿a qué consideramos como error humano?

Medical image classification example:



Suppose:

- (a) Typical human 3 % error
- (b) Typical doctor 1 % error
- (c) Experienced doctor 0.7 % error
- (d) Team of experienced doctors .. 0.5 % error

Tal vez si se quiere demostrar que el algoritmo supera el rendimiento de un humano lo conveniente es usar (b) como error humano. Pero si el objetivo es utilizarlo como proxy para el error de Bayes entonces conviene definirlo como (d).

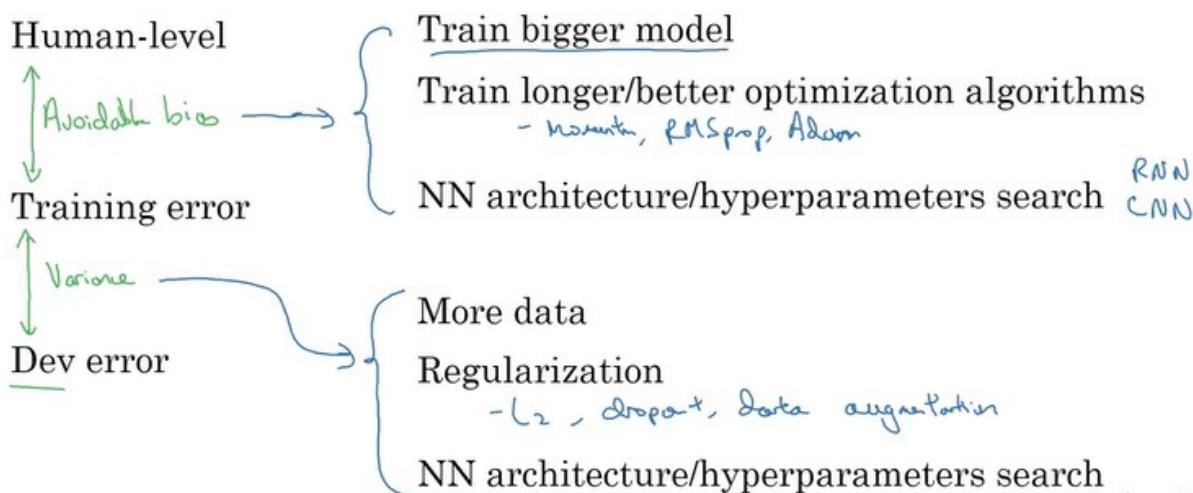
Problemas donde ML sobrepasa el rendimiento humano

Notemos que estos no son natural perception tasks como visión o NLP.

- Online advertising
- Product recommendations
- Logistics (predicting transit time)
- Loan approvals

Mejorar el rendimiento del modelo

Resumiendo lo visto.



Análisis de errores

Este es el proceso de examinar manualmente errores para dar una idea de cómo continuar para lograr un rendimiento cercano al humano.

Consideremos el ejemplo de categorización de perros, mirando el conjunto de datos de desarrollo obteniendo una métrica de rendimiento del 90% un error del 10%.

Uno se preguntaría si habría que mejorar la clasificación de los perros. Una solución es agarrar 100 ejemplos del dev set y contar cuantos son perros.

En el caso de que lo fuera el 5%, el error disminuiría un 0.5% solamente. Mientras que si el 50% de los errores proviene de perros el error bajaría al 5%!



Should you try to make your cat classifier do better on dogs? ↩

Error analysis:

- Get ~100 mislabeled dev set examples.
- Count up how many are dogs.

"ceiling"

5% 10%
5/100 95%

50% 10%
50/100 5%

Evaluar múltiples ideas en paralelo

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats ↪
- Fix great cats (lions, panthers, etc..) being misrecognized ↪
- Improve performance on blurry images

Una idea para esto sería crear una tabla con las imágenes que se miraran como filas y las clasificaciones como columnas y se completa de acuerdo a lo que corresponda. Por último, se determina que porcentaje posee mayor error.

Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
:	:	:	:		
% of total	8%	43%	61%	12%	

Claramente muchos errores en este ejemplo provienen de imágenes borrosas o de los “grandes gatos”.

¿Qué hacemos cuando tenemos ejemplos en el conjunto de desarrollo que están mal etiquetados?

En el caso de haber ejemplos mal etiquetados en los ejemplos de entrenamiento no tiene mucho sentido preocuparse porque generalmente los algoritmos de DL suelen ser bastante robustos en los mismos.

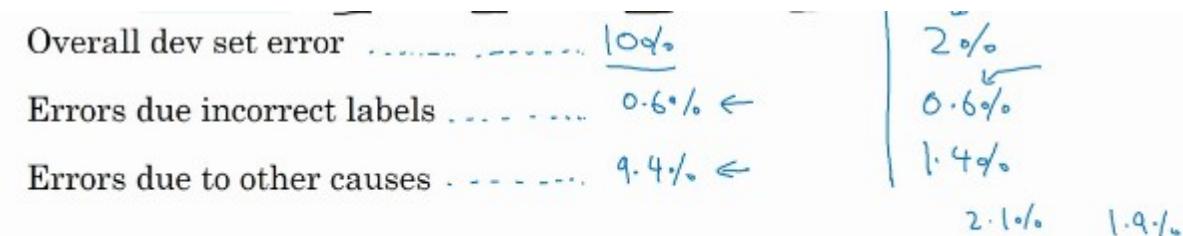


Pero, en el caso de haber en el dev/test sets se recomienda hacer un análisis de errores agregando una columna extra contando el número de ejemplos mal etiquetados.

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

En este ejemplo encontramos que la corrección debido al mal etiquetado no disminuiría el error significativamente. Solo sería necesario hacerlo cuando realmente valga la pena.

A veces conviene tener en cuenta otros errores asociados en el proyecto para compararlos entre si y determinar si es necesario corregir etiquetas.



Correcting incorrect dev/test set examples

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution
- Consider examining examples your algorithm got right as well as ones it got wrong.
- Train and dev/test data may now come from slightly different distributions.

Primero construye el sistema rápidamente, luego itera.

- Set up dev/test set and metric
- Build initial system quickly
- Use Bias/Variance analysis & Error analysis to prioritize next steps.

Entrenamiento y testeo sobre distribuciones diferentes

Supongamos que una aplicación del celular pretende distinguir gatos.

En este caso se tienen dos fuentes: una proveniente de páginas web y otra de los usuarios vía la aplicación, siendo esta última menos profesional con más defectos de distorsión, enfoque, etc.

Data from webpages



Data from mobile app

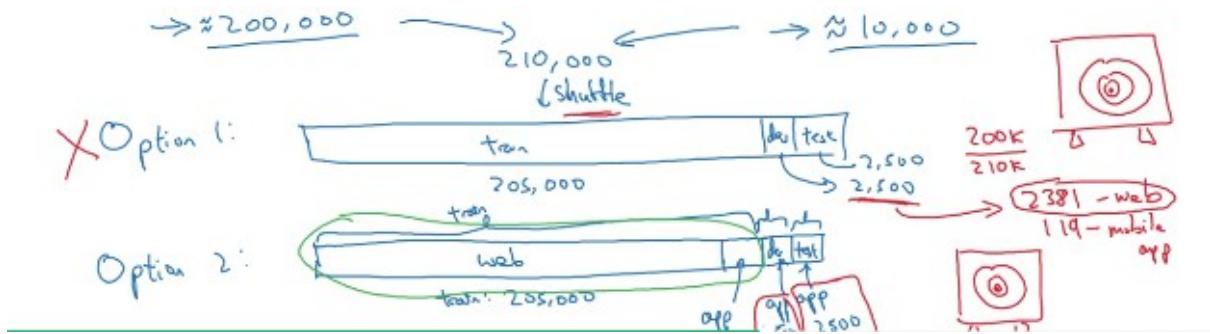


Supongamos que de la web se tienen 200mil imágenes de gatos aproximadamente y que se tiene un número relativamente bajo de usuarios, teniendo un dataset de 10mil imágenes de gatos vía aplicación móvil. Uno se encuentra en un dilema porque el sistema debería funcionar mejor sobre la distribución con menos datos.

Para resolver esto hay varias opciones.

Opción 1: juntar los datos y distribuirlos de manera aleatoria. La ventaja es que la distribución será uniforme, pero posee una desventaja de que prácticamente no hay datos provenientes de la aplicación para el dev/Test set. Esta opción se recomienda no utilizar.

Opción 2: Formar el training set con las imágenes de la web completas más 5mil de la aplicación. El dev/test siendo completamente de la aplicación. Como desventaja el training set tiene una distribución distinta.



¿Es recomendable usar toda la información para la red neuronal? No.

Ejemplo clasificador de gatos.

Assume humans get $\approx 0\%$ error.

Training error 1%

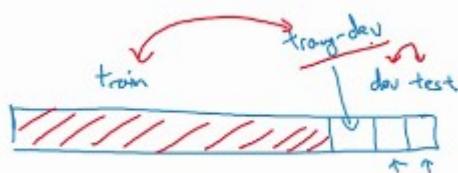
Dev error 10%

Si las distribuciones del training/dev set son las mismas, podemos concluir que habría que disminuir la varianza. Pero si provienen de distintas distribuciones esta suposición no sigue siendo válida. Tal vez no hay un error de varianza y simplemente el dev set posee imágenes más complicadas para clasificar.

Cuando se pasó del error de entrenamiento al error de desarrollo dos cuestiones entraron en juego:

1. El algoritmo vio datos del conjunto de entrenamiento pero no de desarrollo.
2. La distribución de datos es diferente.

Para entender esto se suele definir el **Training-dev set**: tiene la misma distribución que el conjunto de entrenamiento pero no se usa para entrenar la red neuronal.



Solo se entrena el database de entrenamiento. Pongamos algunos ejemplos:

Training error	1%	1%
\rightarrow Training-dev error	9%	1%
\rightarrow Dev error	10%	1.5%
	Variance	Data mismatch

En la columna de la izquierda se puede ver que como el training y training-dev sets tienen la misma distribución el aumento del error es un problema de high variance. Mientras que en la columna de la derecha el error considerable entre el training-dev y el dev es un problema de haber entrenado la red con diferentes distribuciones, a esto se le llama data mismatch.

Otros dos ejemplos

Human error	0%	↓ Available bias	↓ Available bias
Training error	10%		
Training-dev error	11.1	↑ Variance	
Dev error	12.4		↑ Data mismatch

En la primera columna se tiene un ejemplo de high bias, mientras que en segundo tanto high bias como data mismatch.

¿Qué hacer cuando se tiene un problema de data mismatch?

- Realizar un análisis de errores manual para tratar de entender la diferencia entre training y dev/Test sets.
- Hacer que los datos de entrenamiento sean más similares; o recolectar más información similar a los dev/test sets.

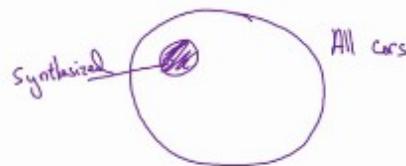
Datos sintetizados artificialmente

El problema con esto es que se puede estar sobre ajustando los datos sintetizados artificialmente.

Car recognition:



N²⁰ cars



Andrew Ng

Transfer learning

Técnica en la que se utiliza un conocimiento aprendido en una tarea para aplicarla a otra diferente.

Supongamos que se entrenó una red neuronal en reconocimiento de imágenes. Si se quiere transferir esto para diagnósticos radiológicos, se puede eliminar la última capa e inicializar pesos w^l , b^l aleatorios para la última capa.

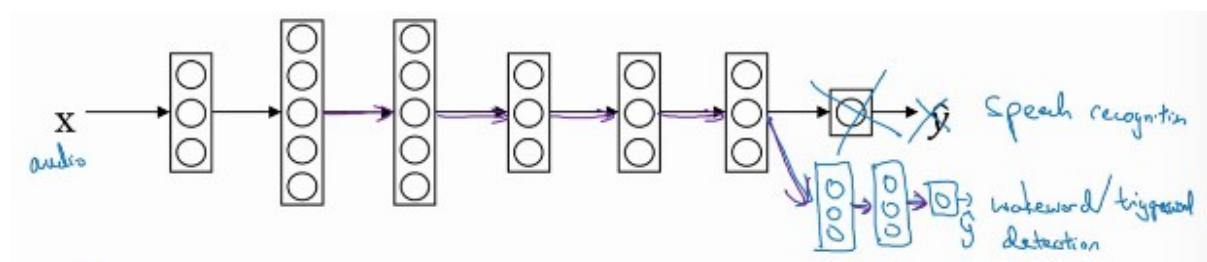
Luego se procederá a entrenar la red neuronal para esta tarea. Si el dataset es lo suficientemente grande se podría entrenar la red completa, sino la última capa únicamente.

Definiciones:

Pre-training: entrenar la red para un proceso previo.

Fine-tuning: entrenar la red para el nuevo proceso.

También se podrían agregar varias capas para el proceso de transferencia.



¿Cuándo tiene sentido este proceso? Cuando se quiere ir desde un aprendizaje con muchos datos a otro con una cantidad mucho menor.

Ej., en reconocimiento de imagen se pueden tener 1M de ejemplos, pero para radiología únicamente 100. En reconocimiento de habla 10mil horas de datos, mientras que en detección de trigger 1 hora.

A su vez, tiene sentido cuando ambos sistemas tienen el mismo tipo de entrada, x .

Si lo opuesto fuera cierto, no sería conveniente aplicar aprendizaje por transferencia.

Multi-task learning

Esto es como una versión generalizada de transfer learning, aprendiendo de muchas tareas en simultáneo.

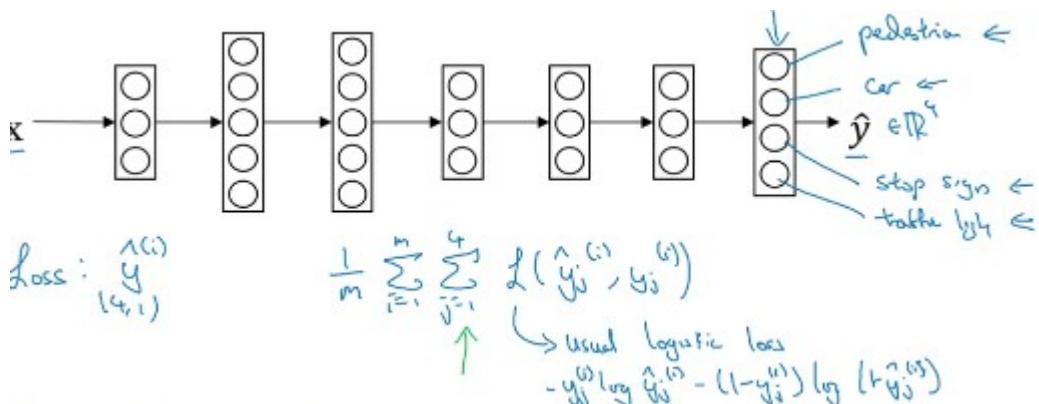
Ejemplo de conducción automática.



Mirando a los outputs como un conjunto.

$$Y = \begin{bmatrix} y_1^{(1)} & y_1^{(2)} & y_1^{(3)} & \dots & y_1^{(n)} \\ \vdots & \vdots & \vdots & & \vdots \\ y_{4,m}^{(1)} & y_{4,m}^{(2)} & y_{4,m}^{(3)} & \dots & y_{4,m}^{(n)} \end{bmatrix}$$

Para predecir estos valores de Y.



A diferencia de softmax, una imagen puede tener varias salidas. Por ejemplo, puede tener un humano y un auto.

Multitask learning es mucho más eficiente que entrenar cuatro redes neuronales para distinguir los cuatro elementos por separado.

Resumen: ¿Cuándo tiene sentido usar multitask learning?

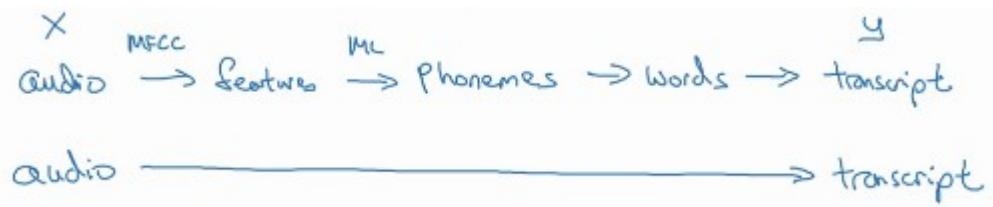
- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually: Amount of data you have for each task is quite similar.

A \downarrow B	$1,000,000$ $1,000$
----------------------------	------------------------
- Can train a big enough neural network to do well on all the tasks.

End-to-end deep learning

Este método toma muchas etapas de un proceso de aprendizaje y los une en una única etapa.

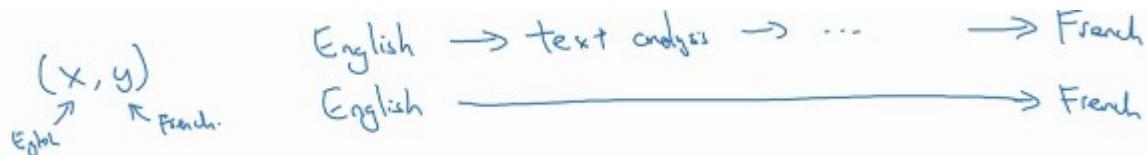
Ejemplo con reconocimiento de audio



En lugar de desarrollar piezas separadas end-to-end deep learning traduce el audio a la transcripción en un único paso.

Para que esto funcione correctamente se necesitan muchos datos.

Ejemplo 2, traducciones



Ejemplo 3, estimar edad de chicos a partir de imágenes por rayos x.

Este último método no funciona del todo bien debido a la poca información de la que se dispone.

Estimating child's age: $\text{Image} \rightarrow \text{bones} \rightarrow \text{age}$



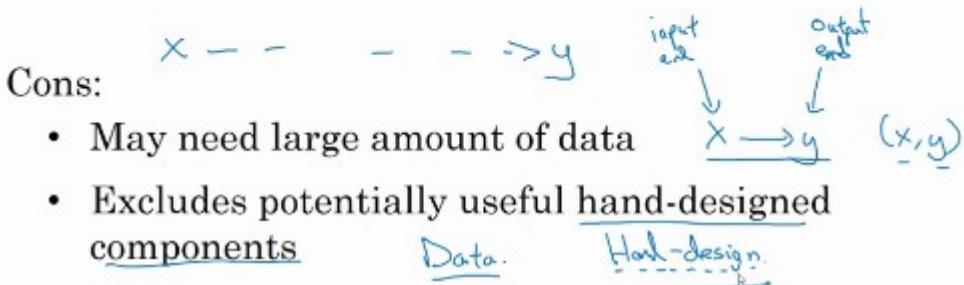
$\text{Image} \longrightarrow \text{age}$

Andre

Ventajas y desventajas del end-to-end deep learning

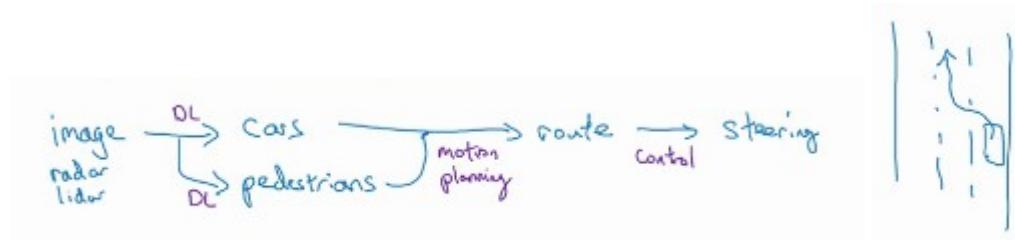
Pros:

- Let the data speak $x \rightarrow y$
- Less hand-designing of components needed



¿Cómo se construye un auto que se maneja solo?

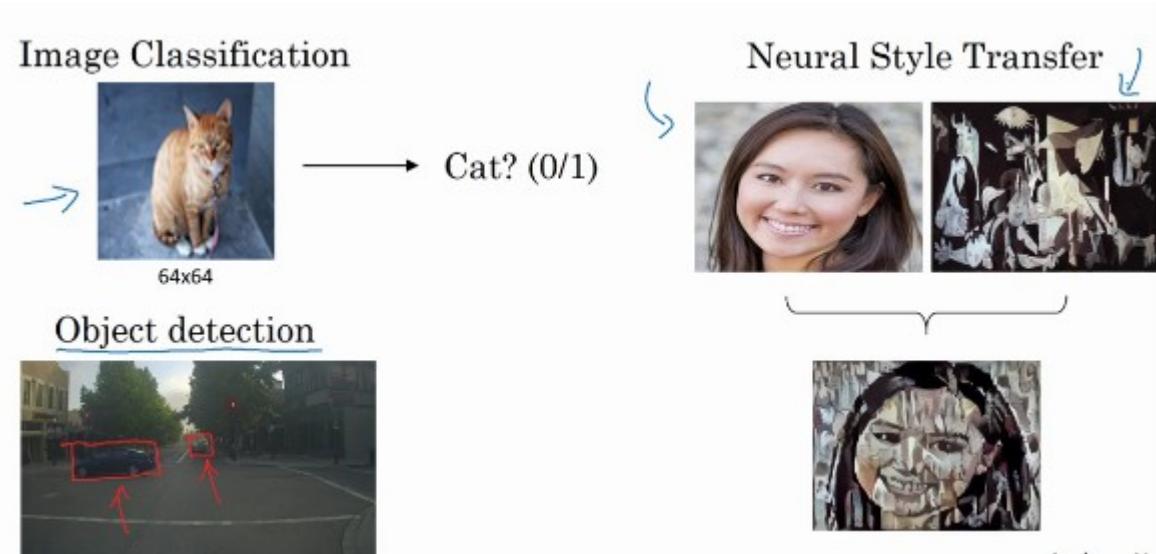
Esto no es un enfoque end-to-end



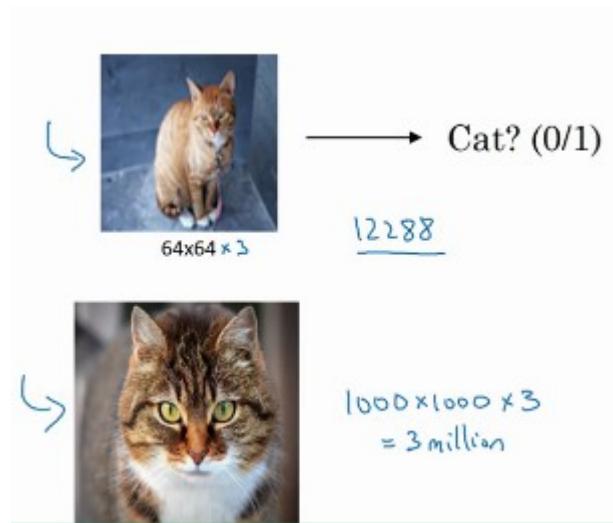
Convolutional Neural Networks (CNN)

Computer Vision

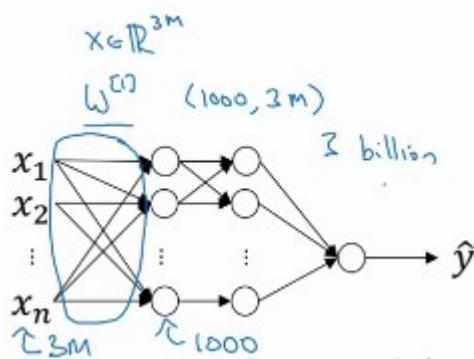
Problemas en computer vision



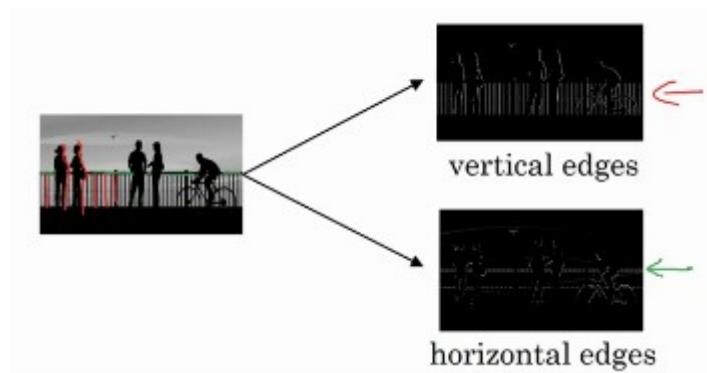
Deep Learning en imágenes grandes



Una red neuronal estándar para una imagen de alta resolución tendría alrededor de 3 mil millones (billions, en inglés) de parámetros. Es por ello que se implementara un nuevo estilo de red neuronal, la red neuronal convolucionada.



Ejemplo de detección de bordes



Detección de bordes verticales

Dado a modo de ejemplo el input de $6 \times 6 \times 1$ se puede convolucionar multiplicando por el filtro indicado en la imagen. ¿Como se aplica la convolución? Se elige el cuadrado de 3×3 y se multiplica elemento por elemento por el filtro sumando todos sus elementos, obteniendo el primer elemento como resultado. Para obtener el segundo elemento se mueve la matriz cuadrada celeste un paso a la derecha y así sucesivamente.

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 3 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

3	0	1	-1	2	7	4
1	5	8	9	3	1	
2	7	2	5	1	3	
0	1	3	1	7	8	
4	2	1	6	2	8	
2	4	5	2	3	9	

6×6

"convolution"

$*$

1	0	-1	
1	0	-1	
1	0	-1	

3×3

filter

$=$

-5			

4×4

El resultado de la convolución es entonces

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4x4

Esto en Python se implementa con `conv_forward`. En TensorFlow `tf.nn.conv2d`. En Keras `conv2d`, etc.

¿Porque esto realiza una detección de bordes verticales? Veamos un ejemplo más simple primero.

The diagram shows a convolution operation. On the left, a 6x6 input image is shown with a green border. The first three columns are labeled '10' and the last three are labeled '0'. A red border highlights the last three columns. Below the input is a 3x3 kernel with values 1, 0, -1 repeated three times. An equals sign follows, and to its right is a 3x3 output image. The first row of the output has values 0, 30, 30, 0. The second row has 0, 30, 30, 0. The third row has 0, 30, 30, 0. The fourth column of the output is highlighted with a red border. Below the input image is a small diagram showing a vertical edge being processed by the kernel. The output image is signed by Andrew N.

El borde blanco indica una presencia de borde sobre el resultado. Parece grueso el borde vertical por el hecho de que la imagen es pequeña.

Más sobre detección de bordes

¿Que sucede si la imagen inicial esta invertida, es decir, si es oscura en la izquierda y brillante en la derecha?

The diagram shows a convolution operation. On the left, an input image is shown with a black border. The first three columns are labeled '0' and the last three are labeled '10'. A blue arrow points to a small diagram of a vertical edge. Below the input is a 3x3 kernel with values 1, 0, -1 repeated three times. An equals sign follows, and to its right is an output image. The output image has a black border. The first row of the output has values 0, -30, -30, 0. The second row has 0, -30, -30, 0. The third row has 0, -30, -30, 0. The fourth column of the output is highlighted with a black border. Below the input image is a small diagram showing a vertical edge being processed by the kernel. The output image is signed by Andrew N.

En este caso se ve que el filtro diferencia los bordes oscuros -> claros de claros->oscuros.
Si no se está interesado se podría tomar valor absoluto del resultado y listo.

Detección de ejes horizontales

Como es de esperarse el filtro para la detección de bordes horizontales viene dado por

1	1	1
0	0	0
-1	-1	-1

Horizontal

Veamos un ejemplo

$$\begin{matrix}
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10
 \end{matrix} * \begin{matrix}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{matrix} = \begin{matrix}
 0 & 0 & 0 & 0 \\
 30 & 10 & -10 & -30 \\
 30 & 10 & -10 & -30 \\
 0 & 0 & 0 & 0
 \end{matrix}$$

Otros ejemplos de filtros para bordes verticales

1	0	-1
1	0	-1
1	0	-1

$$\rightarrow \begin{matrix}
 1 & 0 & -1 \\
 2 & 0 & -2 \\
 1 & 0 & -1
 \end{matrix} \quad \text{Sobel filter}$$

$$\begin{matrix}
 3 & 0 & -3 \\
 10 & 0 & -10 \\
 3 & 0 & -3
 \end{matrix} \quad \text{Scharr filter}$$

Generalizando, los filtros se pueden tomar como parámetros w_i . Detectar bordes para grados que no sean ni horizontales ni verticales.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Padding

Una modificación a la técnica de convolución que se vio previamente.

$\begin{matrix} * & \begin{matrix} \boxed{\quad} & \boxed{\quad} & \boxed{\quad} \\ \boxed{\quad} & \boxed{\quad} & \boxed{\quad} \\ \boxed{\quad} & \boxed{\quad} & \boxed{\quad} \end{matrix} & = \\ \begin{matrix} \frac{6 \times L}{n \times n} \\ n-f+1 \times n-f+1 \\ 6-3+1=4 \end{matrix} & \begin{matrix} 3 \times 3 \\ f \times f \end{matrix} & \begin{matrix} 4 \times 4 \\ \underline{\underline{4}} \end{matrix} \end{matrix}$

Los principales beneficios del padding son los siguientes:

- Permite utilizar una capa convolucional sin achicar necesariamente el alto y ancho de los volúmenes. Esto es importante para construir redes neuronales profundas, dado que, de otro modo, la altura/ancho se achicaría a medida que uno se mueve hacia capas más profundas. Un caso importante especial es la convolución “same” en el cual el ancho/alto se preserva exactamente luego de una capa.
- Ayuda a preservar la información del borde de la imagen. Sin padding, muy pocos valores en la próxima capa se verían afectados por píxeles del borde de la imagen.

Desventajas de la convolución:

- Aplicar continuamente las técnicas de convolución hace que la imagen original se reduzca.
- Los bordes son utilizados una única vez.

Para evitar esto se agrega una borde exterior llamado padding variando el tamaño de la salida.

Considerando el ejemplo anterior, la imagen inicial pasa a ser de 8x8, mientras que la salida de 6x6.

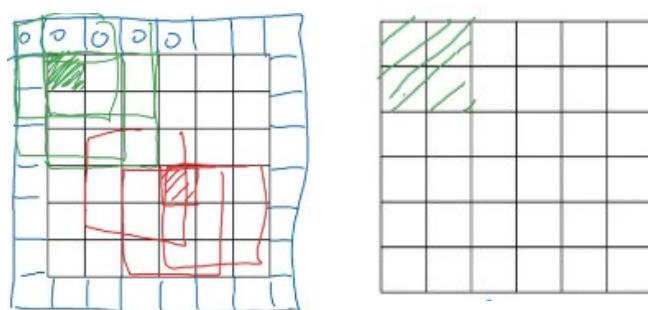


Figure 1. (izquierda) Input

Figure 2. (derecha) Output

Considerando a 'p' (o padding) como el número de capas agregadas, en este caso, $p = 1$. Mientras que ahora se satisface la ecuación:

$$n+2p-f+l \times n+2p-f+l \\ 6+2-3+1 \times \underline{\quad} = 6 \times 6$$

¿Cuántas capas conviene agregar? Convoluciones del tipo Valid y Same:

- Valid: no hay padding.
 - Same: padding tal que el tamaño de salida sea el mismo que el tamaño de entrada.

Esto es

$$n+2p-f+1 \leq n+2p-f+1$$

$$n+2p-f+1 = n \Rightarrow p = \frac{f-1}{2}$$

El tamaño de los filtros generalmente es impar.

Strided Convolutions

Empecemos con un ejemplo

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3 ³	4 ⁴	8 ⁴	3	8	9	7
7 ¹	8 ⁰	3 ²	6	6	3	4
4 ⁻¹	2 ⁰	1 ³	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

$\xrightarrow{7 \times 7}$

*

3	4	4
1	0	2
-1	0	3

3×3

stride = 2

=

91	100	83
69		

Esto quiere decir que dado un stride (paso) = 2, para calcular los siguientes elementos el bloque celeste se mueve de a 2 pasos en lugar de 1.

Summary of convolutions

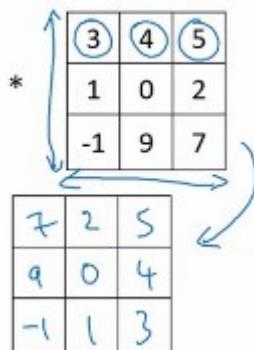
$n \times n$ image $f \times f$ filter

padding p stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad \underbrace{\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor}$$

$$\lfloor z \rfloor = \text{floor}(z)$$

Cross-correlación vs convolución

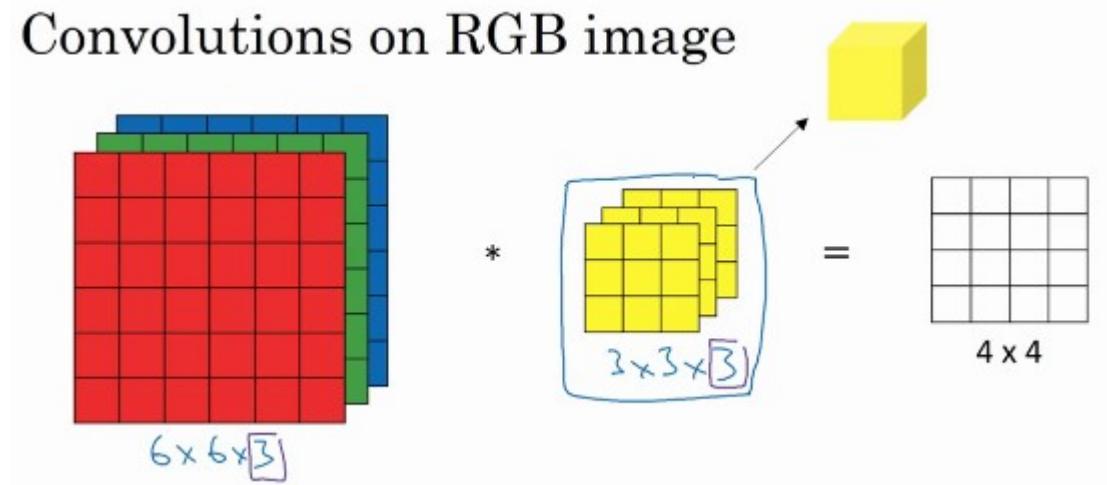


Resulta que en la literatura de Machine Learning se suele decir que la operación recientemente explicada y realizada es llamada convolución, pero realmente es una cross-correlación. En los libros de matemática una convolución viene dada por una operación previa que viene de espejar el filtro de la manera mostrada en la figura de la izquierda, para luego aplicar el operador $*$.

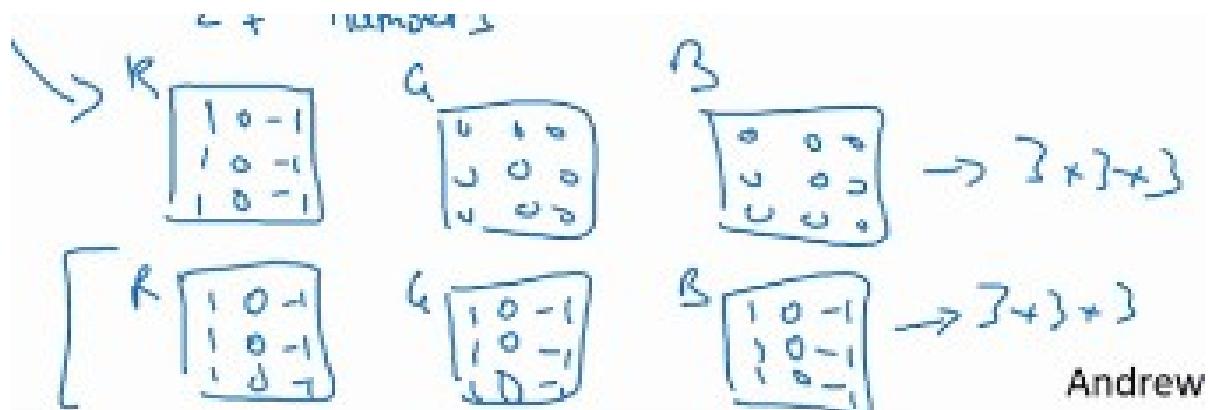
Convolución sobre volúmenes (o imágenes RGB)

El canal correspondiente al tercer número en el input y filtro deben coincidir.

Convolutions on RGB image



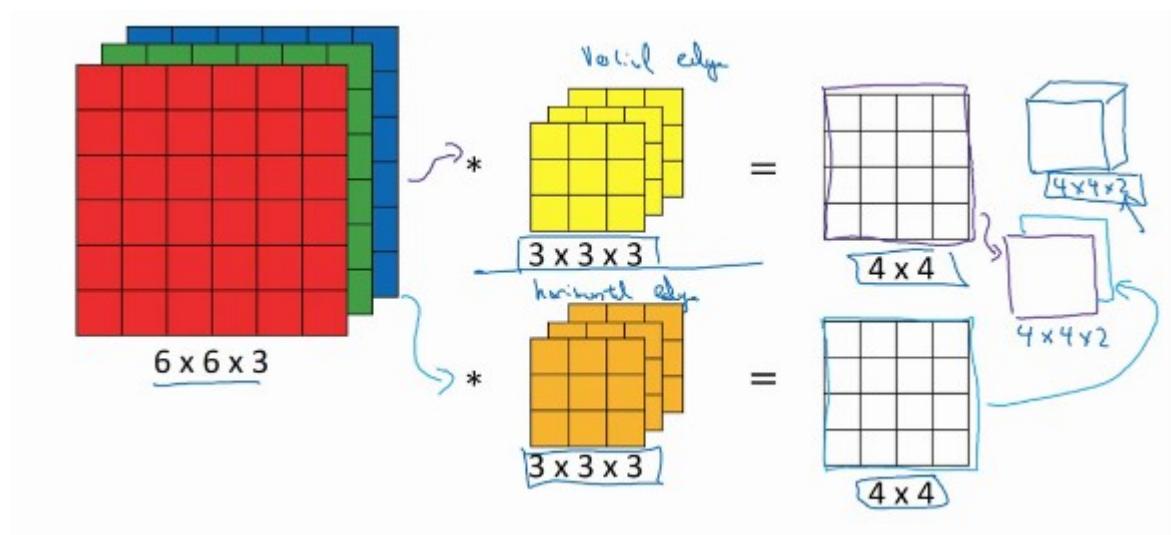
La elección del filtro puede ser la siguiente



donde la primera busca bordes únicamente para el color rojo, mientras que la segunda para cualquier color.

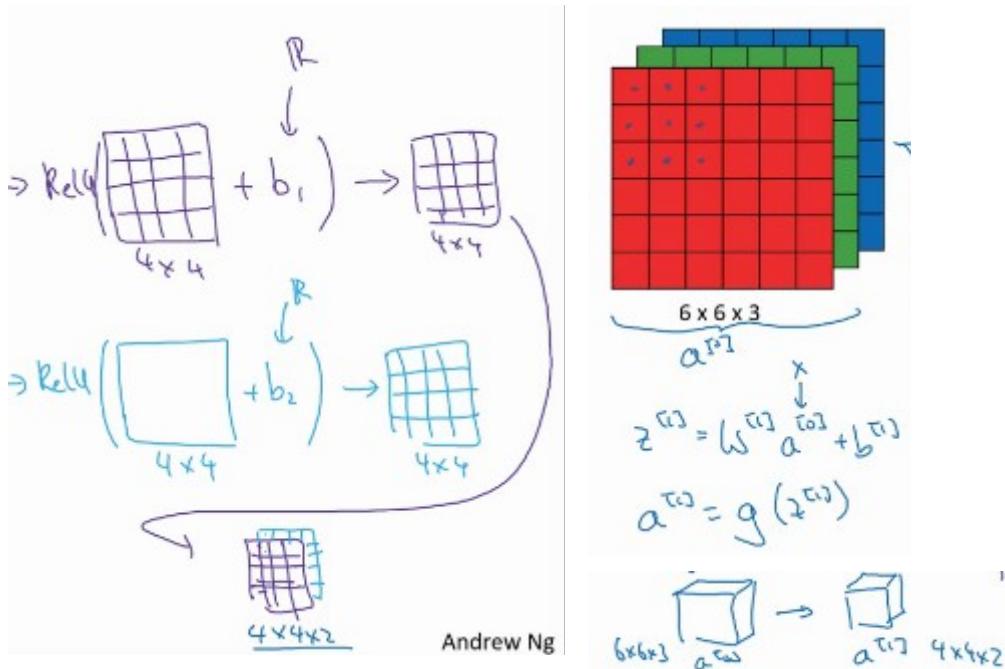
Múltiples filtros

También se pueden aplicar múltiples filtros por separado. Por ejemplo, para bordes verticales y otro para horizontales y juntar los resultados de ambos como muestra la figura.



Una capa de red convolucional

Claramente, la matriz de 4×4 juega el rol de w^*a



Resumen de la notación

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

→ Each filter is: $f^{(l)} \times f^{(l)} \times n_c^{(l)}$

Activations: $a^{(l-1)} \rightarrow n_H^{(l-1)} \times n_W^{(l-1)} \times n_c^{(l-1)}$

Weights: $f^{(l)} \times f^{(l)} \times n_c^{(l-1)} \times n_c^{(l)}$

bias: $n_c^{(l)} - (1, 1, 1, n_c^{(l)})$ ← #filters in layer l .

$$\begin{aligned} \text{Input: } & h_H^{(l-1)} \times n_W^{(l-1)} \times n_c^{(l-1)} \\ \text{Output: } & h_H^{(l)} \times n_W^{(l)} \times n_c^{(l)} \\ n_H^{(l)} = & \left\lfloor \frac{n_H^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} + 1 \right\rfloor \end{aligned}$$

$$A^{(l)} \rightarrow m \times n_H^{(l)} \times n_W^{(l)} \times n_c^{(l)}$$

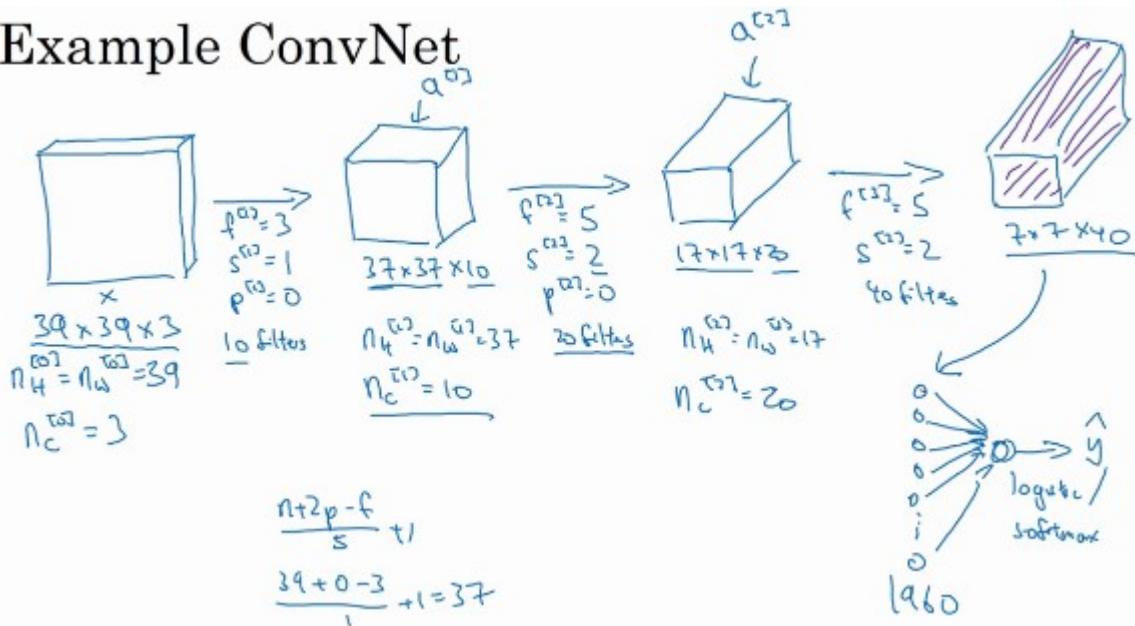
Ejemplo de una red convolucional

Veamos cómo aplicar una red convolucional en varias capas de redes neuronales.

Los parámetros correspondientes al tamaño y numero de los filtros, stride y padding los decidimos nosotros previa a cada capa. Los resultados se obtienen con las ecuaciones mencionadas arriba. Por ejemplo, para la primera capa el tamaño de la imagen viene dado por $(39 + 0 - 3) / 1 + 1 = 37$.

Uno de los mayores desafíos en CNN es elegir estos hiper parámetros.

Example ConvNet



Tipos de capas en redes convolucionales:

- Convolución (CONV)
- Pooling (POOL)
- Completamente conectada (FC)

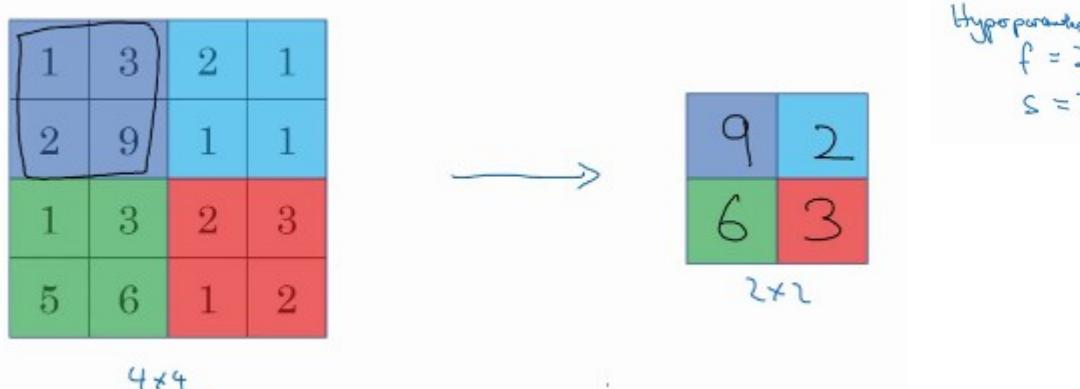
Esto se realiza para mejorar el tiempo de implementación de las CNN principalmente.

Capas tipo Pooling

Max Pooling

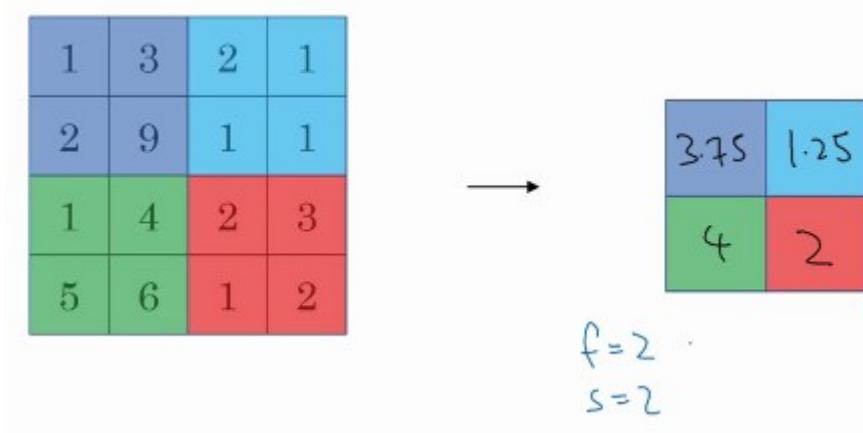
Tomando el máximo de cada cuadrado coloreado.

Hiper parámetros correspondientes a la técnica de max pooling.



Pooling promedio

Al igual que lo anterior se toma el promedio de cada cuadrado coloreado.



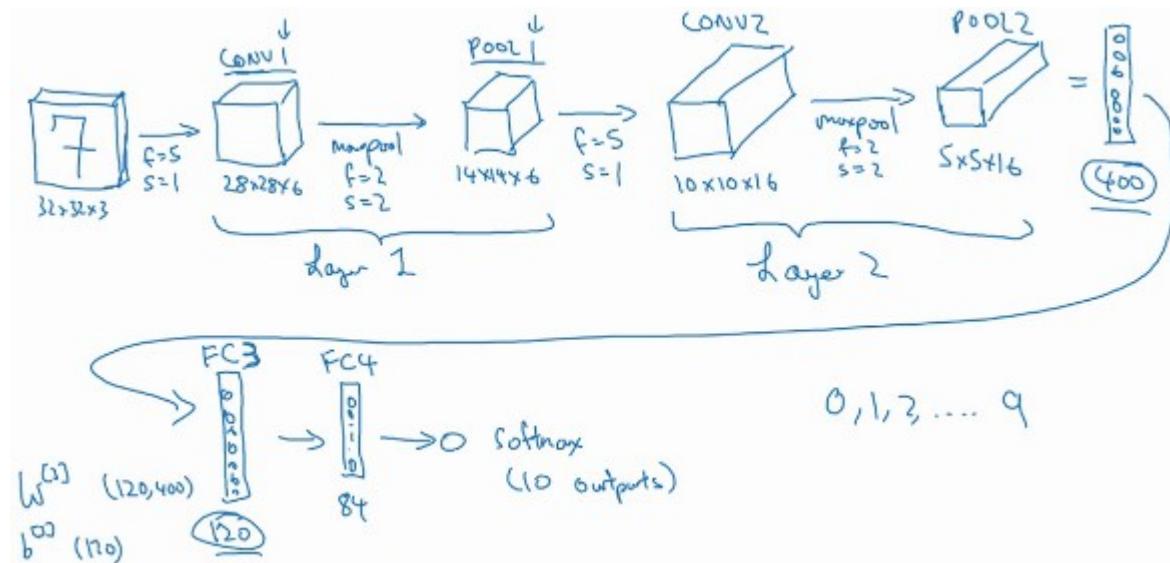
En redes muy profundas suele ser más común utilizar esta última técnica antes que el pooling máximo.

El tema con pooling es que no hay parámetros que aprender en backpropagation.

Ejemplo red neuronal basado en el trabajo de Yann LeCunn: LeNet-5

Una convención es decir que una capa es como la que figura la imagen. Otros definen a las capas tipo pooling como una capa de por si, pero no se suele hacer por el hecho de que no poseen hiper parámetros.

La capa completamente conectada utilizada en la red neuronal es simplemente una capa utilizada como se hizo previamente donde los 400 inputs se conectan completamente con los 120, de aquí que los parámetros w y b tengan los tamaños que se muestran en la figura.



Características que se notan al realizar ejemplos

$n_H, n_W \downarrow$
 $n_c \uparrow$

El tamaño de las imágenes tiende a disminuir al aumentar el número de capas, mientras que el número de filtros tiende a aumentar.

Esquema de la red neuronal aplicada a CNN

CONV - POOL - CONV - POOL - FC - FC - FC - SOFTMAX

Algunos temas importantes que se notan mirando la tabla inferior y el ejemplo previo:

- El número de hiper parámetros de las capas tipo pooling es cero, mientras que de las capas convolucionales es bastante pequeño comparados con las FC.
- El tamaño de las funciones de activación disminuyen a medida que nos movemos a capas más profundas.

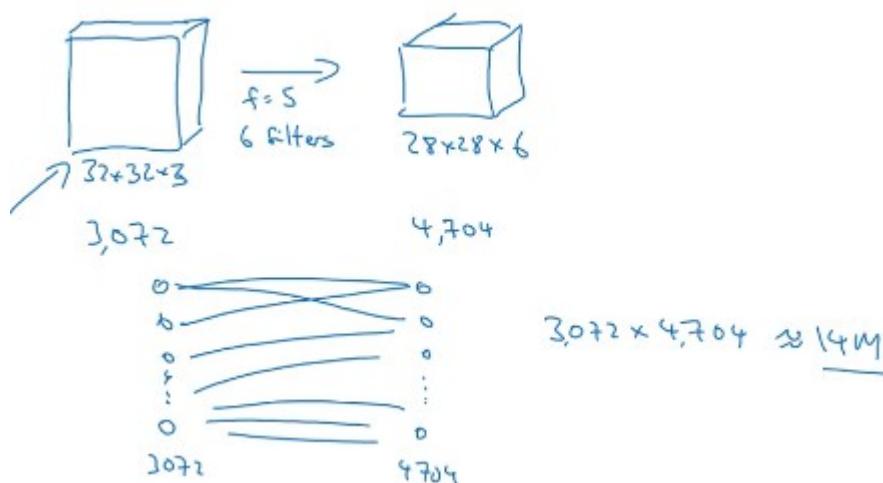
	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	— 3,072 $a^{32 \times 3}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 ←
POOL1	(14,14,8)	1,568	0 ←
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 ←
POOL2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48,001 {
FC4	(84,1)	84	10,081 }
Softmax	(10,1)	10	841

¿Por qué usar capas convolucionales en lugar de únicamente FC?

Ventajas por sobre utilizar únicamente las redes totalmente conectadas (FC).

- Parámetros compartidos
- Sparcity de las conexiones.

Veamos esto con un ejemplo, como se dijo previamente, conectar estas dos imágenes requeriría entrenar en una capa 14M de parámetros. A diferencia en una red convolucional es 156 considerando un filtro de 5x5x6=156, mucho menor.



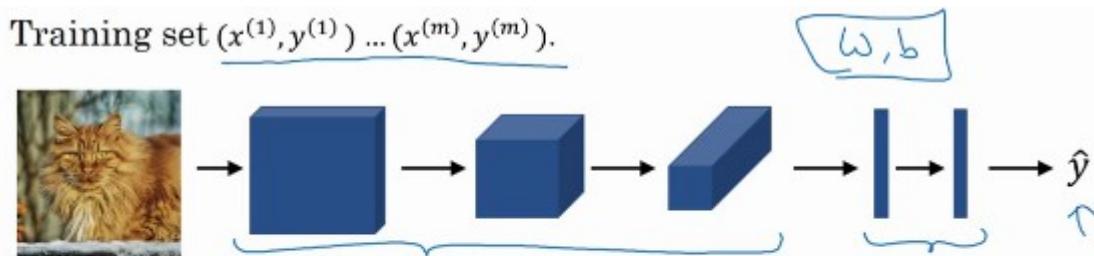
Parámetros compartidos: una característica a detectar sobre que una parte útil de una imagen es probablemente útil en otra parte de la imagen.

Sparcity de las conexiones: En cada capa, cada valor de salida depende únicamente de un número pequeño de entradas. Ej., el valor 0 del extremo izquierdo de la salida depende del cuadro superior izquierdo de 3x3 de la entrada con valor 10s.

Ejemplo entrenando una CNN con fotos de gatos

x: imágenes de felinos

y: etiqueta identificando gatos o no.



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce J

Supongamos que se eligió una estructura para la CNN empezando por una secuencia de CONV-POOL seguida por FC y SOFTMAX al final.

Con esto se calculará una función de coste como figura en la ecuación superior y luego se usara gradiente descendiente para optimizar los parámetros y reducir esta función.

Week 2: Case studies.

Outline

Classic networks:

- LeNet-5 ←
- AlexNet ←
- VGG ←

Estudiar otros ejemplos son útiles para entender muchos conceptos de CNN. Los temas a abordar incluyen CNNs clásicas como LeNet-5, AlexNet y VGG, como así también ResNet una red neuronal muy profunda de 152 capas y las redes del tipo Inception.

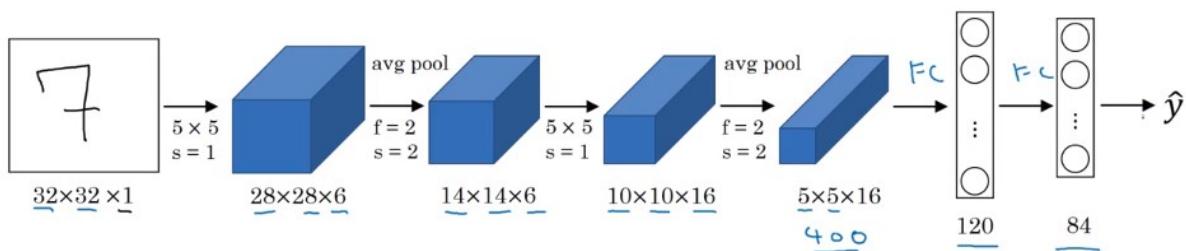
LeNet-5

ResNet (152)

Inception

La arquitectura de esta red se esquematiza en la imagen siguiente. Obtenido del documento: [[Le Cun et al., 1998. Gradient-based learning applied to document recognition](#)] (si se quiere leer mirar principalmente secciones II y III).

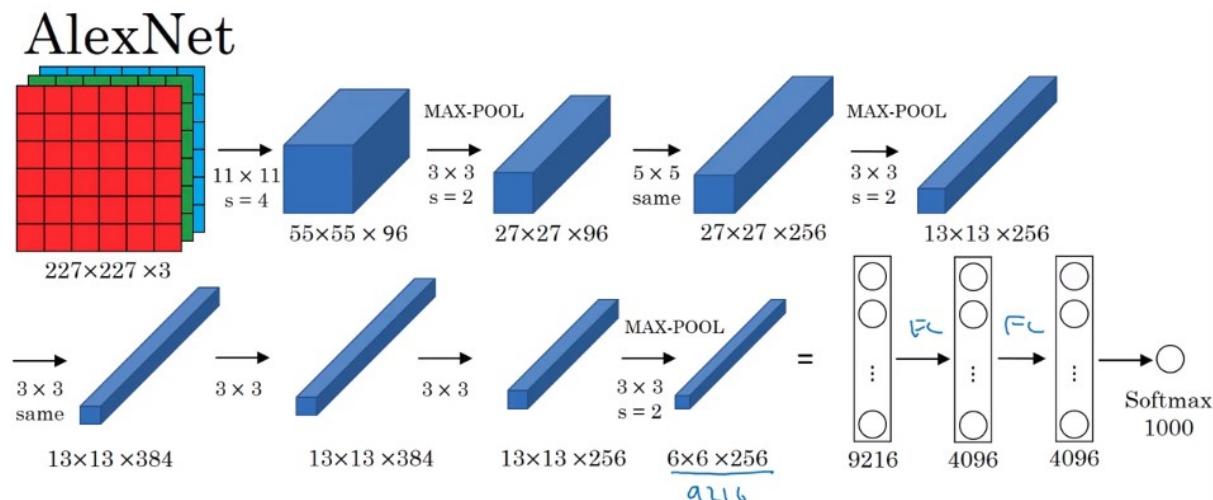
Inicialmente la entrada (imágenes en escala de grises) de tamaño $32 \times 32 \times 1$ se procesa mediante 6 filtros de 5×5 con stride = 1 y no padding. Es por ello que la salida es una red de $28 \times 28 \times 6$. Luego se aplica a pooling. Pooling promedio era utilizado más en 1998, ahora no tanto. Luego se aplica otra capa de red convolucional y un pooling promedio. Luego dos redes completamente conectadas y por ultimo un último nodo que puede tomar 10 valores, uno para cada una de las posibles soluciones: 0,1, 2, ...,9. Una versión moderna usaría Softmax al final.



Esta red posee alrededor de 60k parámetros. A día de hoy se encuentran redes con más de 10M.

AlexNet

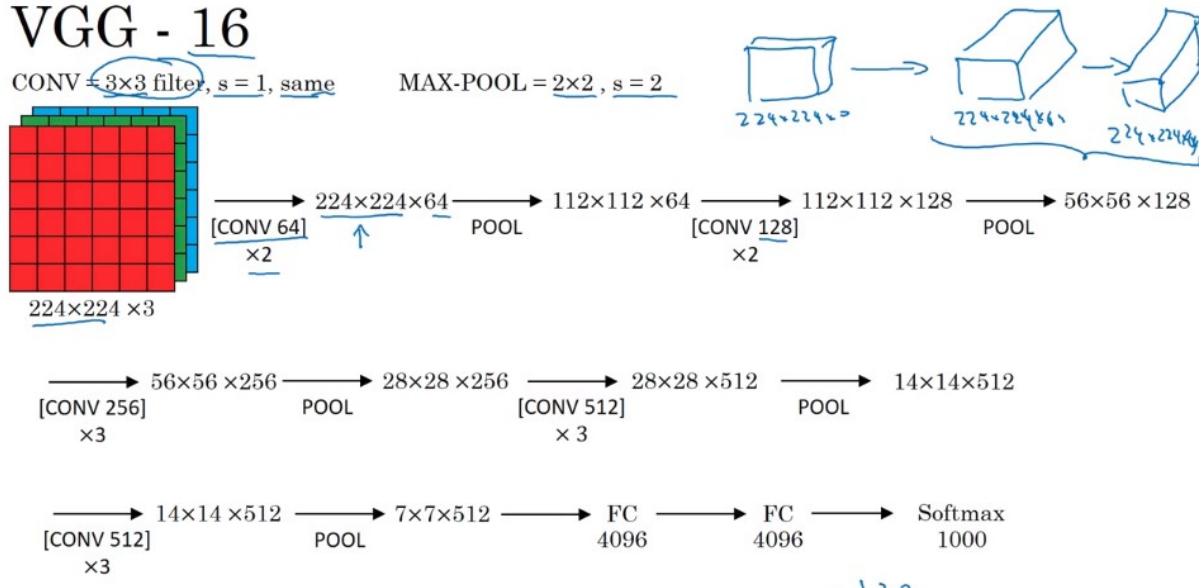
Esta red está basada en el siguiente documento: [[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks](#)]. La misma tiene una arquitectura similar a la anterior, pero es mucho más grande, tiene alrededor de 60M de parámetros.



VGG-16

Esta red está basada en el siguiente documento: [[Simonyan & Zisserman, 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition](#)]. Posee 138M de parámetros.

Se llama VGG-16 por el número de capas que posee, 16.



Veamos ahora algunas redes más poderosas y más importantes.

ResNet

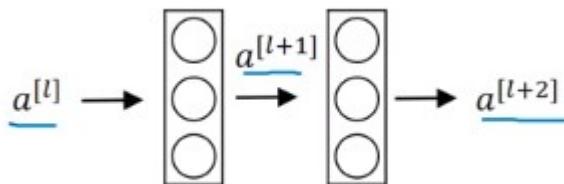
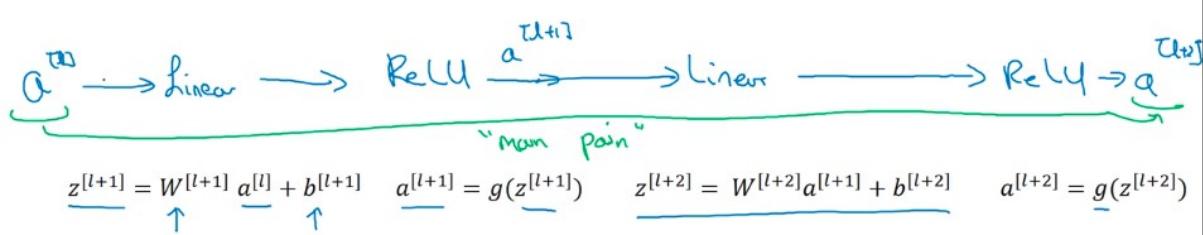
Vanishing and exploding gradients son un problema con las redes neuronales muy profundas. Para entrenar redes con hasta 100 capas se va a explicar un concepto en el que la activación de una capa y alimentarla en una capa más profunda que la primera.

Estas redes están basadas en el paper: [[Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.](#)].

Bloques residuales

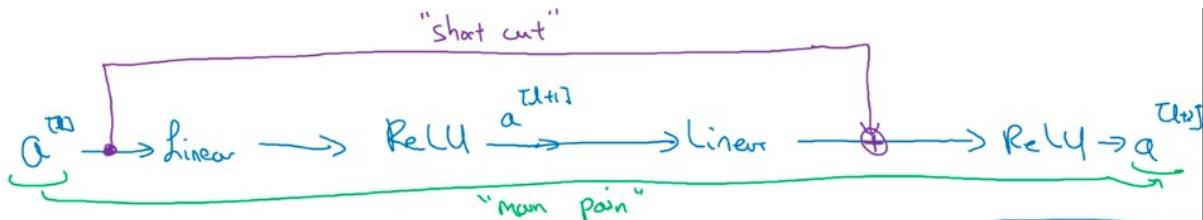
Dado una función de activación de orden l se puede obtener una de orden l+2 pasando por dos capas de la siguiente manera:

Donde primero se aplica una función lineal para obtener $Z^{[l+1]}$. Luego se aplica una función no lineal como ReLU obteniendo la activación $a^{[l+1]}$. De manera análoga se obtiene $a^{[l+2]}$. A esto se lo llama “main path”.



En redes residuales se hace un cambio, esto es, tomar $a^{[l]}$ y sumarlo antes de calcular la función no lineal ReLU para obtener $a^{[l+2]}$, como muestra la figura llamado "short cut" (skip connection o atajo).

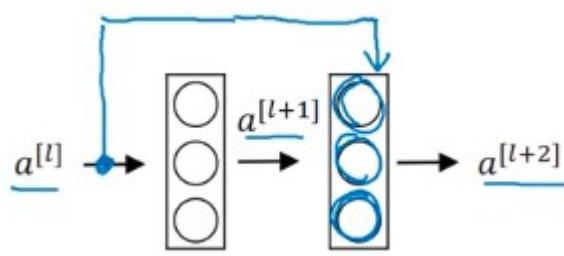
La información de $a^{[l]}$ va más adentro de la red neuronal.



De esta manera la función de activación es calculada como

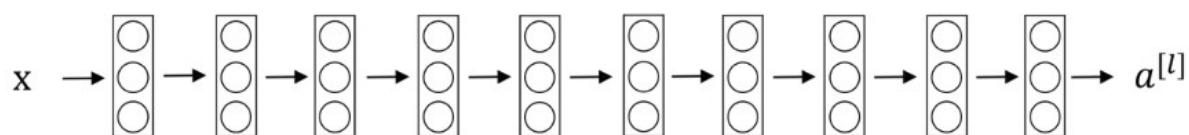
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

El diagrama de esta nueva red es entonces

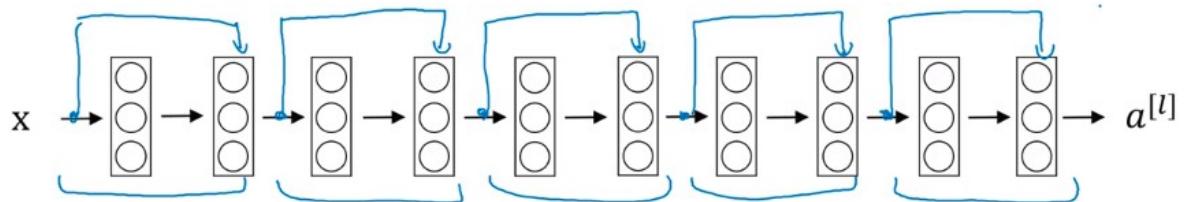


Las Resnets agarran varios de estos bloques residuales para formar una red con varias capas.

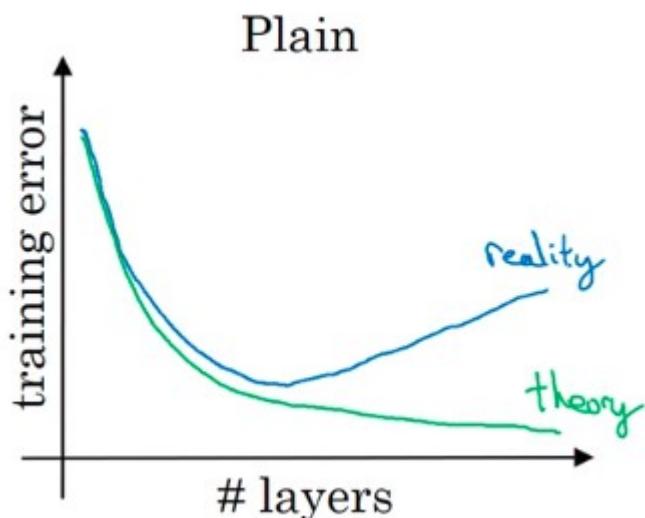
Red neuronal plana (plain network)



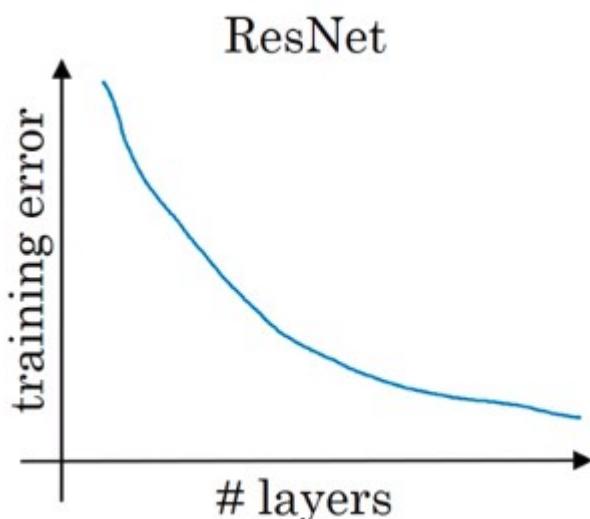
Para convertirla en una red residual agregamos los bloques residuales de la siguiente manera



Veamos ahora cómo se comporta una red neuronal plana a medida que el número de capas aumenta. En teoría el error de entrenamiento debería decrecer, pero llega un punto en el que aumenta.



En cambio, con las Resnets se pueden entrenar redes más profundas dado que el error de entrenamiento disminuye al incluir bloques residuales.

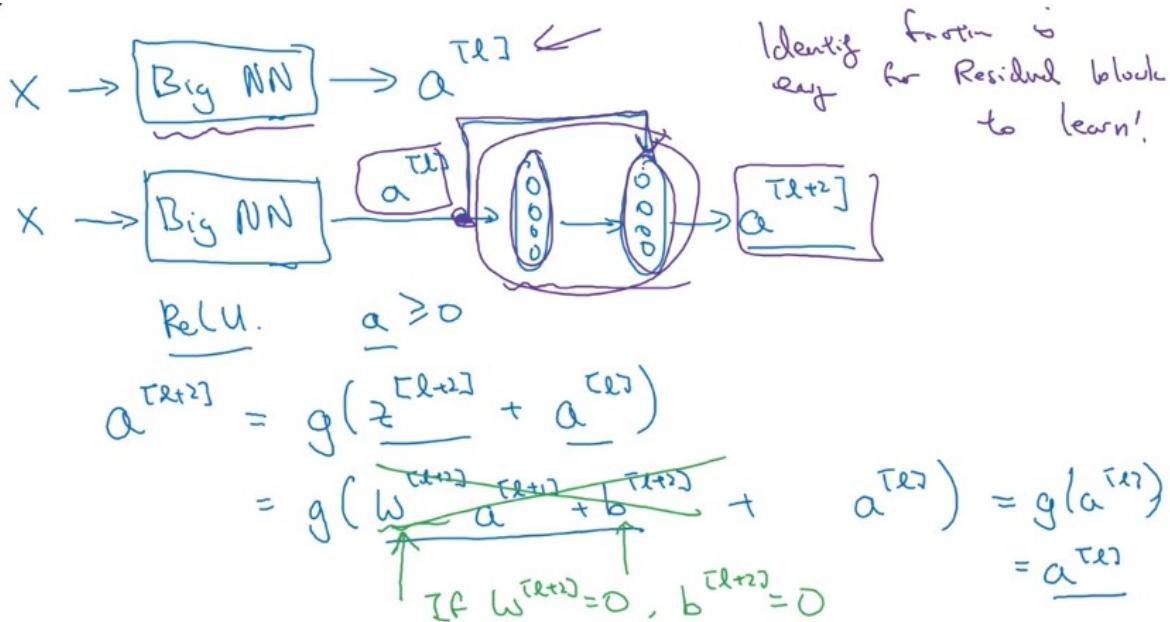


¿Por qué funcionan tan bien las ResNets?

Funcionar bien en sets de entrenamiento es un pre requisito para que funcione correctamente en dev o test sets.

En la primera figura se considera el caso de dado un input X, una gran red neuronal actúa sobre la misma y se obtiene una función de activación $a^{[l]}$. Si se agrega un bloque residual,

se puede ver que la función identidad es fácil de aprender a pesar de tener más capas que la red anterior.



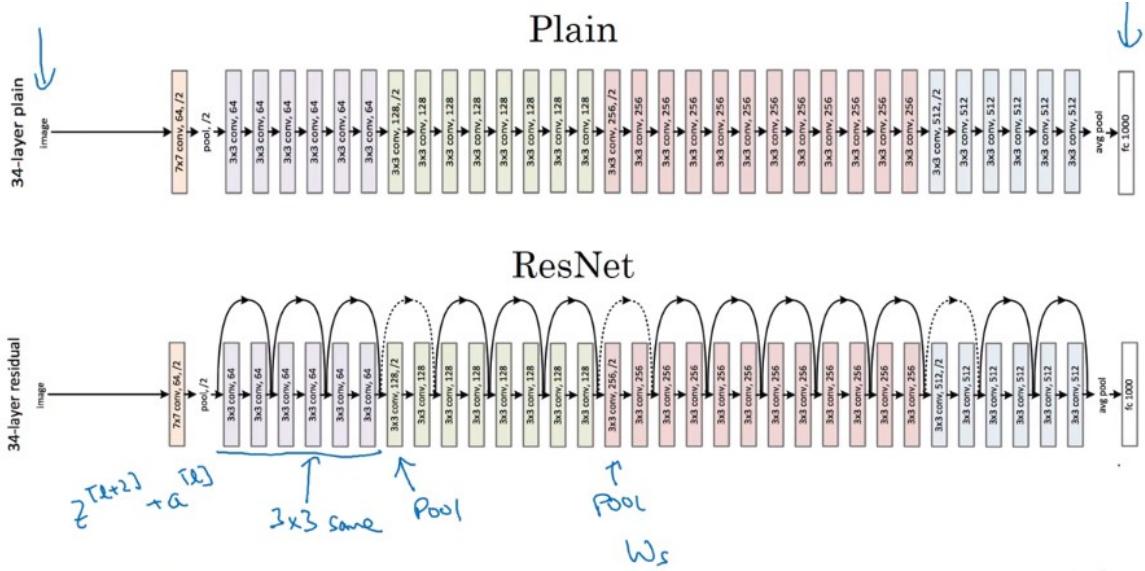
Suposición importante: Tanto $z^{[l+2]}$ como $a^{[l]}$ poseen la misma dimensión. Para lograr esto es por ello que se usan convoluciones del tipo same. En el caso de que posean diferentes dimensiones, por ejemplo, $a^{[l+2]}$ es de dimensión 256 y $a^{[l]}$ de dimensión 128, se agrega una matriz w que multiplica a esta última de dimensión 256x128. Esta puede ser una matriz de parámetros o una matriz que implementa padding igual a cero.

$$a^{[l+2]} = R \cdot w_s \cdot a^{[l]} = \underbrace{\dots}_{128} = \underbrace{\dots}_{256}$$

R 256×128

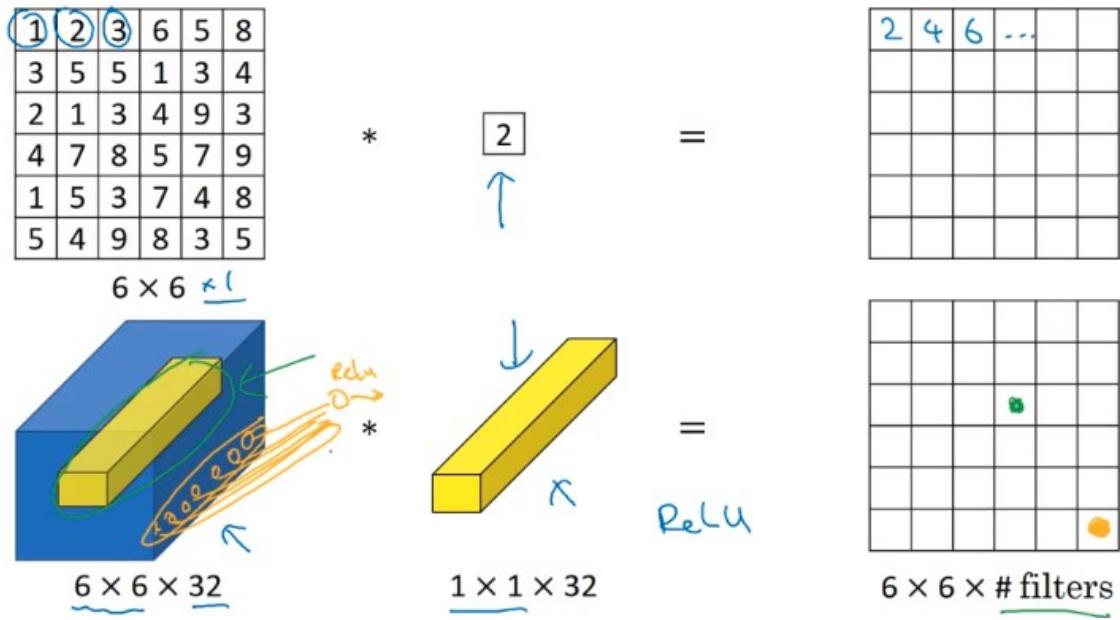
Ejemplo aplicado a una imagen (obtenido del paper referenciado)

En el primer caso se ve una red neuronal plana de 34 capas que utiliza CNNs con same padding. Para que en la ResNet al aplicar la red se mantienen las dimensiones entre $z^{[l+2]}$ y $a^{[l]}$. Pero también hay incluidas capas tipo pooling por lo que ajustes son necesarios mediante la matriz m .



Convoluciones 1x1

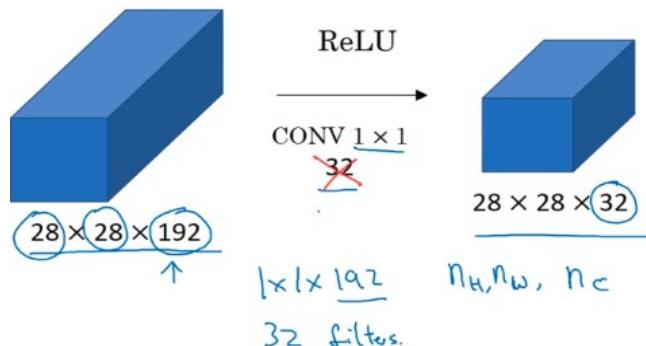
¿Qué es lo que hacen las convoluciones 1x1? Para el caso de una entrada de 6x6x1, multiplicarla por un filtro de tamaño 1x1x1 parece trivial como multiplicar un escalar. Pero si la entrada posee otros valores la cosa cambia. Por ejemplo, si es de 6x6x32, multiplicar por un filtro de 1x1x32 provoca que se obtenga un resultado de dimensión 6x6x32. Esto es conocido también como Network in Network, nombrado en el paper: [[Lin et al., 2013.](#) [Network in network](#)]. Este método ha influenciado otras redes como las del tipo Inception.



Usando una red de 1x1

¿Cuándo son útiles? Nos permite mantener fijo el número de filas y columnas y cambiar únicamente el número de filtros, tanto aumentarlo como disminuirlos. En el caso de la

Imagen se disminuye la red hasta una con 32 filtros a la salida de la red convolucional. Por otra parte, permite construir redes tipo Inception las cuales veremos a continuación.

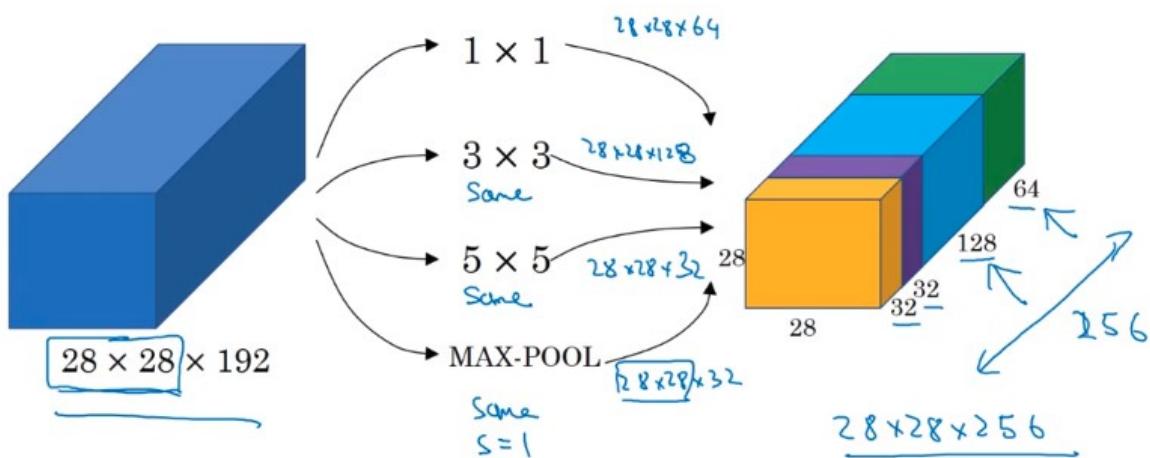


Redes tipo Inception

Al momento de elegir una capa CNN se tiene que elegir de que tamaño se quiere utilizar un filtro o si se usa una capa tipo pooling. Lo que dice la red tipo Inception es por qué no se utilizan todas. Esto hace que la arquitectura sea más complicada pero funciona mucho mejor. Obtenido del paper: [Szegedy, Christian, et al. "Going deeper with convolutions." Cvpr, 2015.]

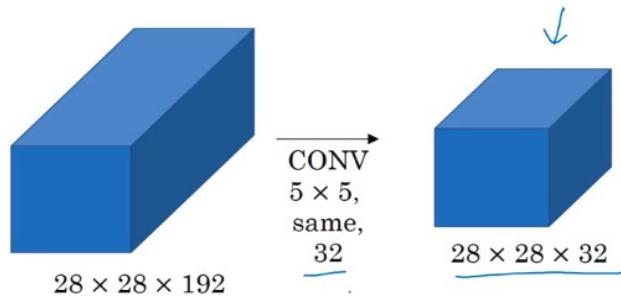
Motivacion para la red tipo Inception

Básicamente como se dijo anteriormente, dada una entrada de tamaño $28 \times 28 \times 192$ se aplicara una convolución de, digamos, 1×1 , el resultado será de $28 \times 28 \times 64$, pero tal vez querríamos aplicar una de 3×3 con convolución tipo same para que el tamaño sea el mismo, el resultado sería $28 \times 28 \times 128$. De la misma manera con una capa de 5×5 . Por último, si se aplica max pooling de tamaño $28 \times 28 \times 32$, para que coincidan los tamaños hay que elegir same padding y stride = 1. El resultado total es un volumen concatenado de $28 \times 28 \times 256$.

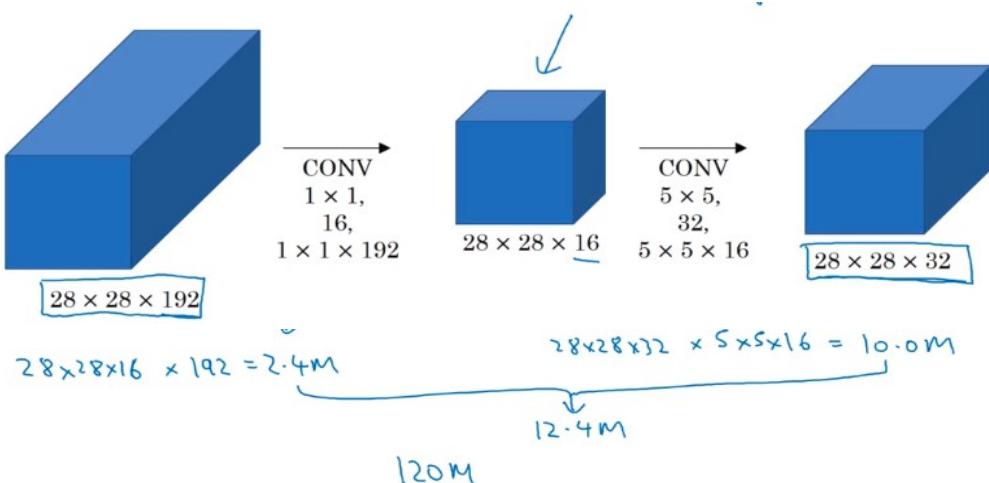


Hay un problema con todo esto con el coste computacional de la red. Veamos, por ejemplo, la de 5×5 .

EL tamaño de los filtros viene dado por $5 \times 5 \times 192$. Mientras que el número de filtros es de $28 \times 28 \times 32$ de esta manera el número de operaciones es 120M -> Operación muy extensa. Esto se puede reducir en un factor de 10.

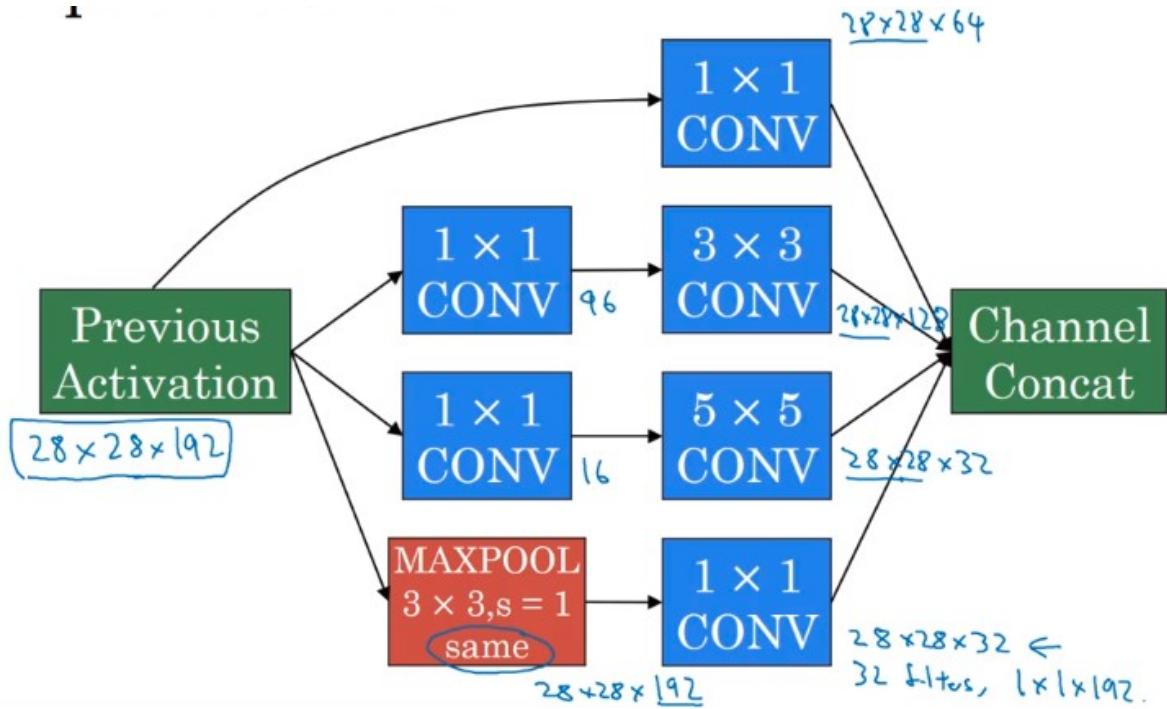


Esto se soluciona agregando una red convolucional de 1×1 luego de la entrada antes de aplicar la red de 5×5 . El tamaño del input cambia a $28 \times 28 \times 16$, siendo este último sobre el que se aplica la red de 5×5 , el tamaño de salida sigue siendo el mismo. Esta capa de 1×1 se llama capa cuello de botella (bottleneck layer). El costo computacional de esta red es mucho menor, la primera capa realiza $28 \times 28 \times 16 \times 192 \sim 2.4 \text{ M}$ de cálculos, mientras que la segunda $28 \times 28 \times 32 \times 5 \times 5 \times 16 \sim 10 \text{ M}$, lo que es igual a $12.4 \text{ M} \ll 120 \text{ M}$.

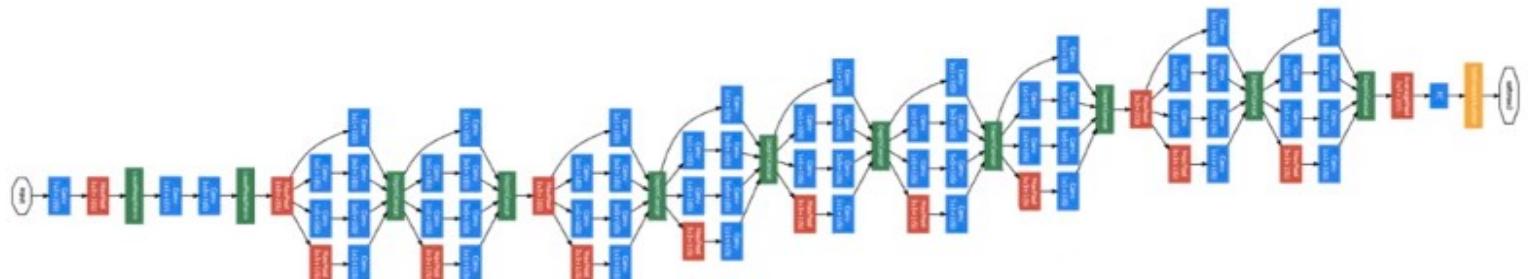


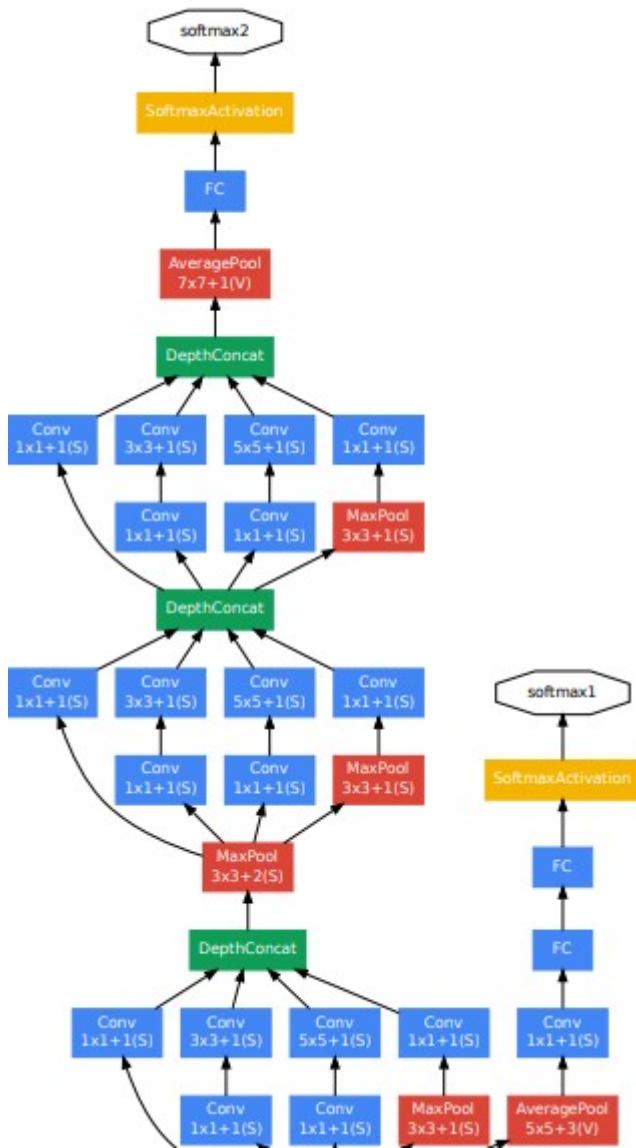
Módulo de la red tipo Inception

Haciendo esto mismo para todas las convoluciones se obtiene el siguiente módulo: para la capa tipo pooling conviene adjuntarle 32 filtros de tamaño $1 \times 1 \times 32$ para que la salida sea de $28 \times 28 \times 32$ como queríamos. Concatenando todos estos resultados se obtiene el valor deseado de $28 \times 28 \times 256$, habiendo realizado muchas menos operaciones que si se hubieran aplicado las convoluciones de 5×5 o 3×3 directamente a la entrada.



Entonces, ¿qué es una red tipo Inception? Es unir estos módulos. También se muestra con un poco de zoom. Puede notarse que el módulo de Inception que se describió anteriormente esta repetido continuamente en la red. Una sutil diferencia con el esquema de la red mostrada con aumento es que tiene adicionalmente adjuntada unas capas para realizar predicciones con dos capas completamente conectadas y un Softmax.





Consejos prácticos sobre cómo utilizar redes convolucionales (Conv-Nets)

- Uso de implementaciones en código libre

Implementar código libre para hacer más fácil la replicación de resultados.

También se puede obtener código libre en GitHub de una arquitectura o tema que nos interese, por ejemplo

```
$ git clone https://github.com/tensorflow/models.git
```

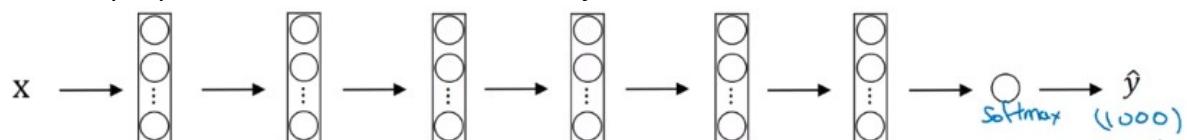
- Transferencia de aprendizaje (Transfer learning)

En lugar de hacer que una red aprenda los parámetros de la misma, se pueden obtener los resultados de alguien más y utilizarlos para realizar nuevas predicciones.

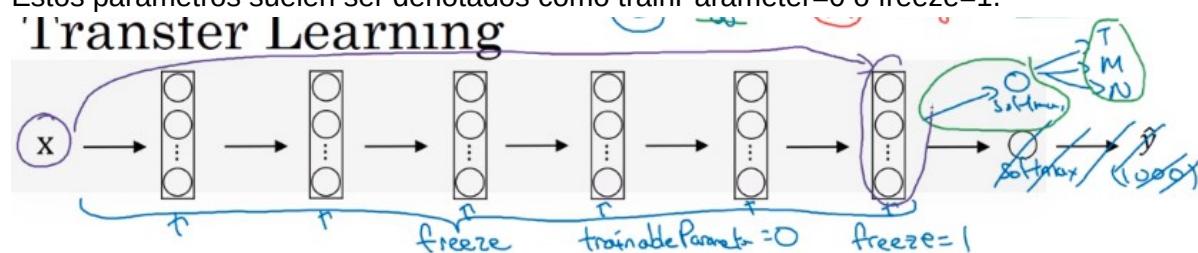
Veamos un ejemplo: si se quiere implementar una red para identificar si un gato es Tiger, Misty o ninguno de los dos (ignorando las fotos en las que aparezcan ambos). Dado que el set de datos seguramente sea pequeño, ¿qué se puede hacer?



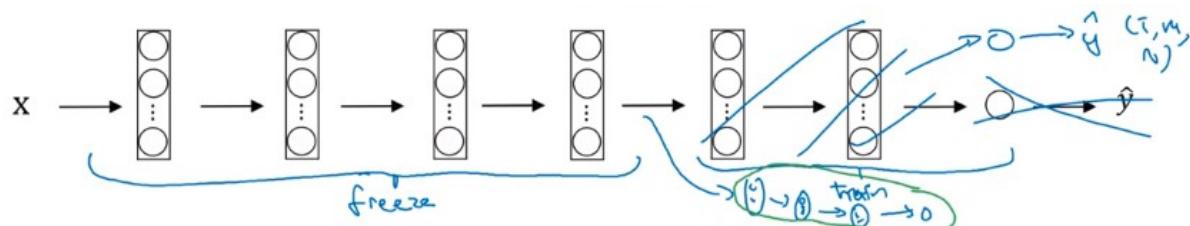
Para ello conviene conseguir una red neuronal open source que ya haya sido entrenada (source code y los pesos), por ejemplo una red que haya sido entrada sobre el ImageNet data set que permite identificar hasta 1000 objetos.



Para hacer esto conviene eliminar la salida del Softmax y agregar las nuestras (en nuestro caso T,M,N). Algunas redes tienen parámetros que permiten que los pesos de las capas anteriores no sean entrenadas sino que mantengan los valores entrenados con ImageNet. Estos parámetros suelen ser denotados como `trainParameter=0` o `freeze=1`.



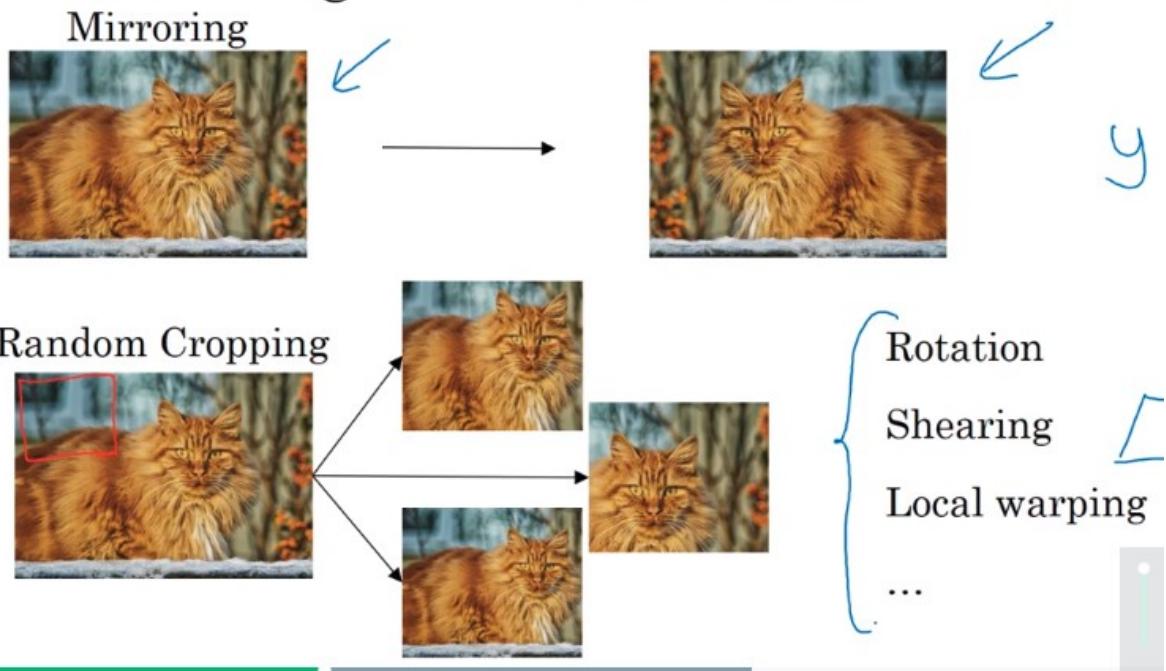
¿Qué sucede si se tiene una cantidad de datos bastante grande que puede entrenar una red de algunas capas? En este caso conviene deshacernos de una porción de la red y agregar nuestras propias capas ocultas y Softmax y entrenarlos con nuestros datos.



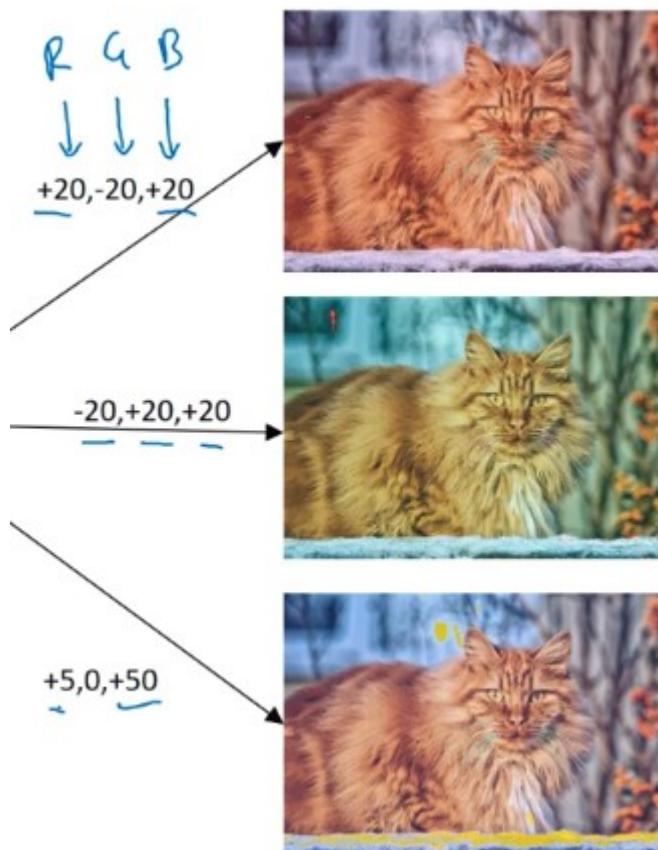
Por último, como caso límite, si se poseen suficientes datos, se puede entrenar la red completa pero, en lugar de inicializar con números aleatorios los parámetros, se realiza con los parámetros descargados previamente y luego realizar gradiente descendente para actualizar estos parámetros.

- Aumento de datos (data augmentation)

Espejado: será una buena técnica si preserva la identificación del objeto.
Cortado aleatorio, rotación, achique, etc.

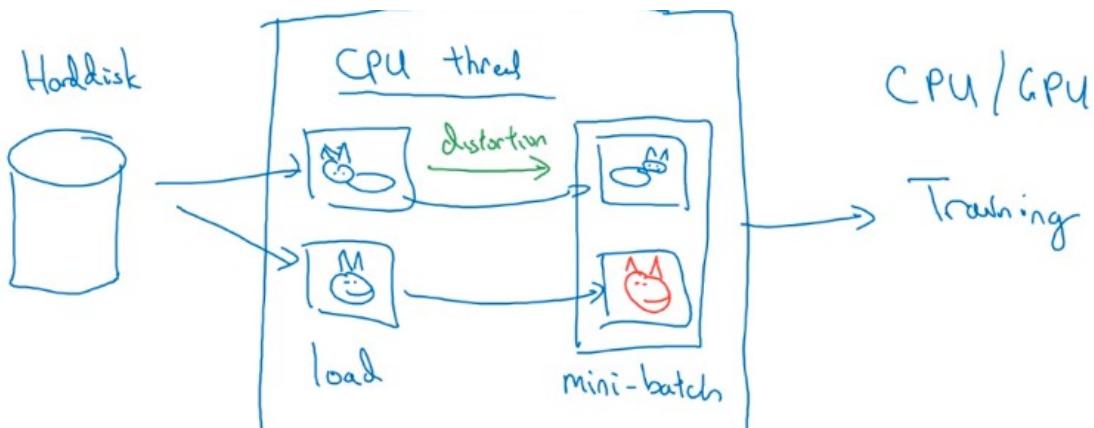


Cambiado de color: distorsionar los colores RGB para tener distintos colores.



Implementar distorsiones durante el entrenamiento

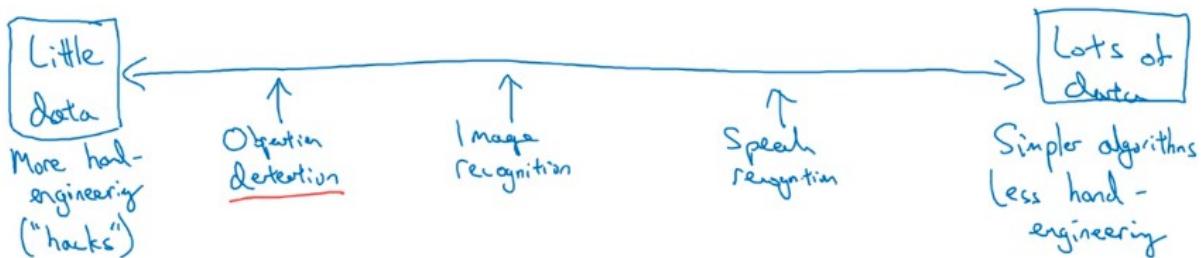
Mientras se entrena un conjunto de imágenes otro thread/hilo de la CPU/GPU puede estar generando nuevas imágenes mediante distorsión. El resultado de esto será un mini-batch del cual se realimentará la red para continuar entrenándola.



Estado de la visión computacional

Datos vs ingeniería dura

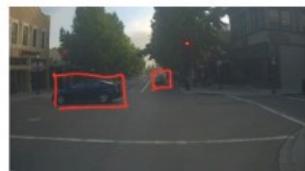
Estados de los principales problemas en Deep Learning en función del número de datos disponibles. Detección de objetos posee muchos menos datos que reconocimiento de imágenes debido a que necesita el encuadre de los objetos.



Cuando un problema tiene muchos datos, se suele utilizar menos ingeniería dura, mientras que si se poseen menos datos se utiliza más (o hacks o trucos, Transfer Learning). De esta manera se concluye que las dos fuentes de conocimiento de una red son: datos etiquetados o hacks que vienen dados por features modificados como los vistos anteriormente, modificaciones a la arquitectura de la red, etc.

Two sources of knowledge

- • Labeled data (x, y)
- • Hand engineered features/network architecture/other components



Consejos para obtener buenos resultados en benchmarking/ganar competiciones

- Ensamblado: entrenar 3 a 15 redes neuronales separadamente y promediar su salida (no sirve promediar sus pesos). Esto entrega resultados 1% o 2% mejor.

Como inconveniente el tiempo adicional que se tarda es proporcional al número de redes adicionales que se entrenan.

- Multi cortado en tiempo de test (multicrop at test time): correr el clasificador en muchas versiones de imágenes de prueba y promediar los resultados.

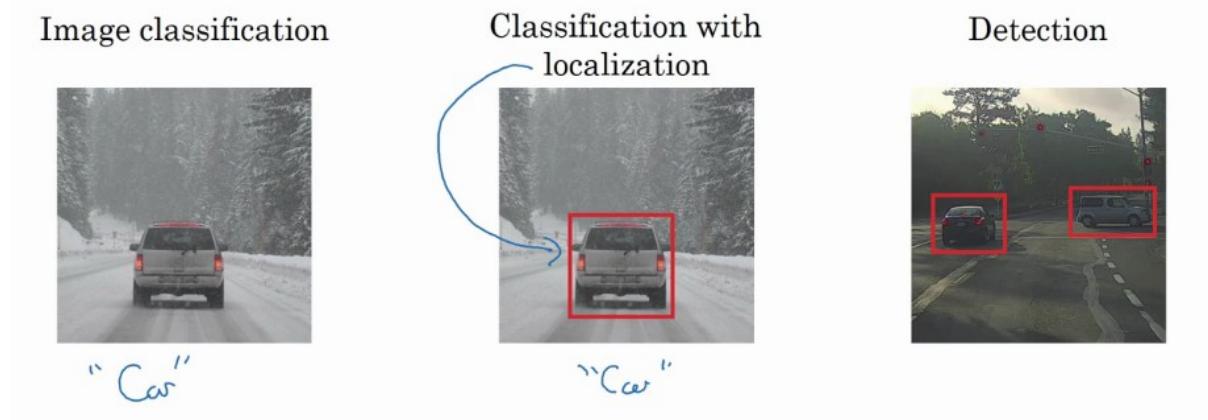


Week 3: Detección de objetos

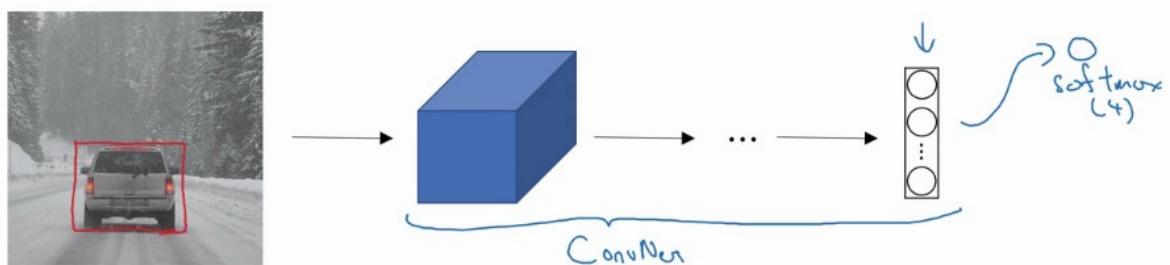
Antes de estudiar la detección de objetos vamos a estudiar la localización de objetos.

Localización de objetos

(izquierda) Como hemos visto en clasificación de imágenes, basta con decir si la imagen pertenece a un auto o no. (centro) Pero en clasificación con localización no solo hay que indicar si hay un auto, sino que hay que encuadrar el auto en cuestión. (derecha) múltiples autos son posibles y hay que identificarlos. No solo autos, sino también personas, bicicletas, etc.



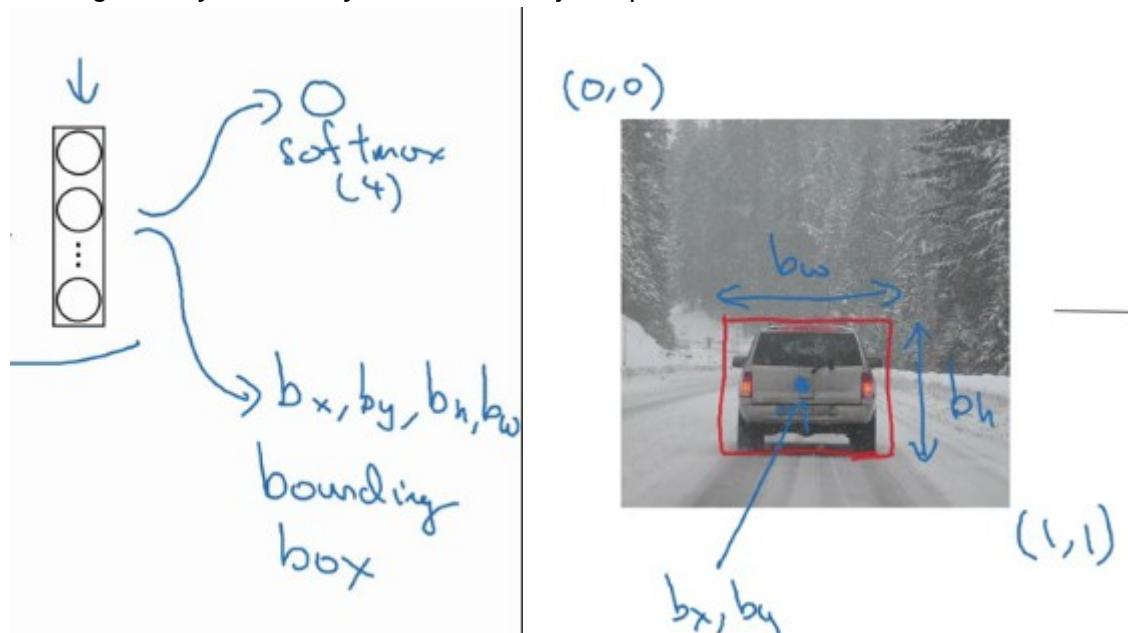
En el caso de que se quiera clasificar una foto, la red neuronal a implementar seria la siguiente:



- 1 - pedestrian ←
- 2 - car ←
- 3 - motorcycle ←
- 4 - background

Una red convolucional con un Softmax a la salida de 4 filas correspondiente a la detección de una persona, un auto, una moto o ninguna de las anteriores.

Para localizar un objeto necesitamos más salidas a la red neuronal. En este caso, 4 valores adicionales como pueden ser: b_x , b_y , b_h y b_w . Donde b_x y b_y corresponden al centro del rectángulo, b_h y b_w el alto y ancho de la caja respectivamente.



¿Como construir la salida 'y' a partir de una imagen?

El primer elemento nos dice si se ha detectado un objeto. Caso negativo, el resto de la salida no nos interesa.

Need to output b_x, b_y, b_h, b_w , class label (1-4)

$x =$

$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ is there obj?

$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \leftarrow \text{"don't care"}$

La función de coste podría definirse a partir de cuadrados mínimos como

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ + \dots + (\hat{y}_8 - y_8)^2 & \text{if } \underline{y_1 = 1} \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

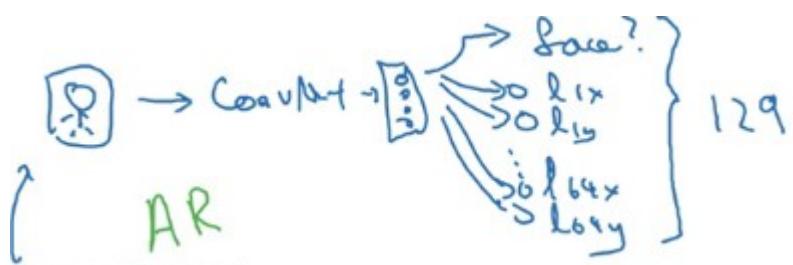
También se podría haber determinado **log-likelihood** loss para c_1, c_2, c_3 , regresión logística para p_c y SRE para b 's, aunque esta última puede funcionar bien en total como se dijo previamente.

Detección de puntos importantes (landmarks)

supongamos que usamos una red neuronal para detectar los puntos de un ojo o cualquier otro punto arbitrario de una cara, por ejemplo.



$l_{1x}, l_{1y},$
 $l_{2x}, l_{2y},$
 $l_{3x}, l_{3y},$
 l_{4x}, l_{4y}
 \vdots
 l_{64x}, l_{64y}

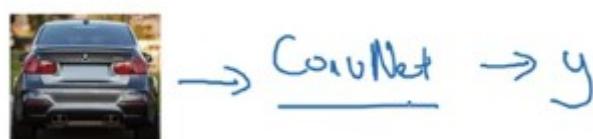


Esto suele ser utilizado en aplicaciones como realidad aumentada para editar fotos en tiempo real, por ejemplo. A la salida uno obtiene por ejemplo una fila que dice si se detectó una cara y luego si limitamos el número de puntos (cada dos correspondiente a un landmark, x e y) a 64 entonces se tendrán 129 filas en total.

Detección de objetos

Algoritmo de detección de autos

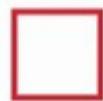
- 1) Creamos una base de datos con autos cortados de manera cercana y otros sin autos
- 2) Luego de crearla se puede entrenar una CNN de la siguiente manera



Algoritmo de las ventanas deslizantes

Training set:

X	y
	1
	1
	1
	0
	0



Ahora dada una imagen que puede contener uno o varios autos, se selecciona una ventana de un dado tamaño como se muestra en la figura y se empieza a mover a lo largo de la imagen. La ConvNet procesa cada uno de estos cuadrados que cubren el total de la imagen y detecta si hay un auto o no en base a lo entrenado anteriormente.

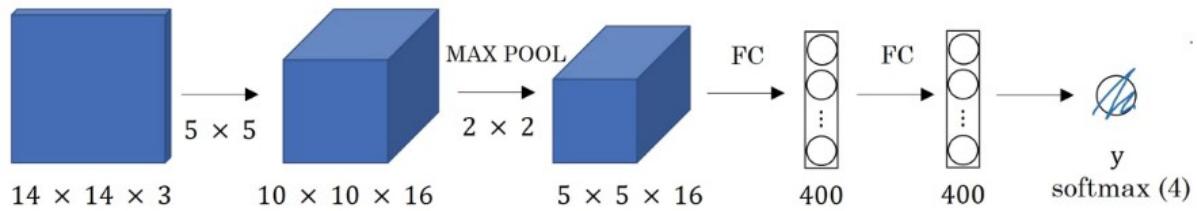


Luego esto se puede repetir para otros tamaños de rectángulos y distintos strides.

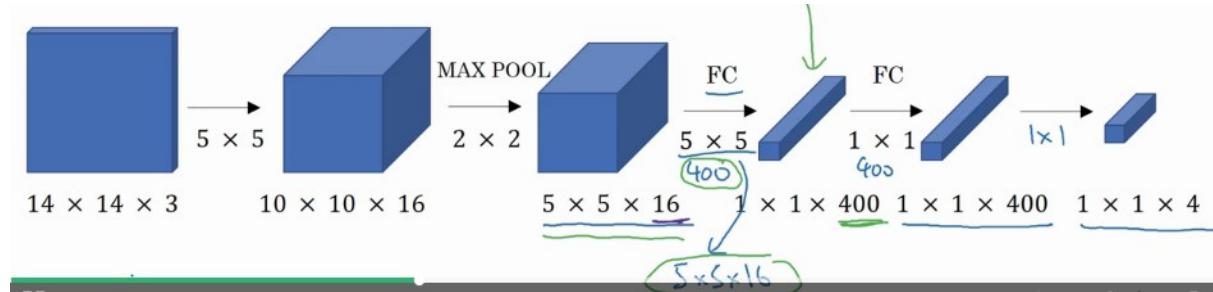
La desventaja de este método es el costo computacional que requiere evaluar, aun utilizando strides grandes. Se podría aumentar el stride para mejorar el costo computacional, pero disminuiría la precisión. Afortunadamente esto se puede implementar de una manera mucho más eficiente.

Implementación convolucional de las ventanas deslizantes

Antes de explicar estas ideas, es necesario introducir como pasar de una capa completamente conectada (FC) a una capa convolucional. Para ello primero dada la siguiente red

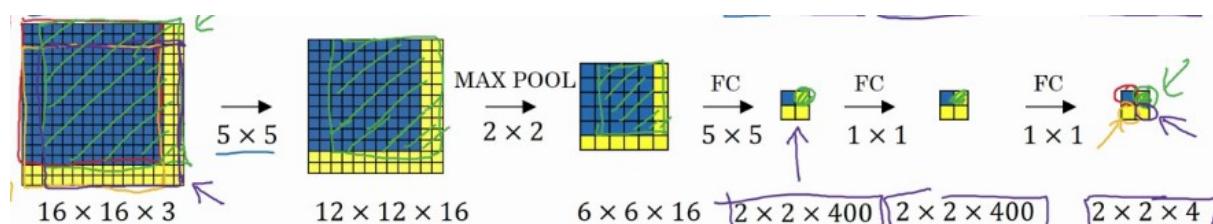


Se pueden reemplazar las capas completamente conectadas utilizando por ejemplo 400 filtros de 5×5 . En este caso la salida será de tamaño $1 \times 1 \times 400$. A esta última se le aplican 400 filtros de 1×1 como muestra la figura.



¿Cómo implementamos el algoritmo de ventanas deslizantes utilizando CNN? Esto está basado en el paper siguiente: [\[Sermanet, Pierre, et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks." arXiv preprint arXiv:1312.6229 \(2013\).\]](#)

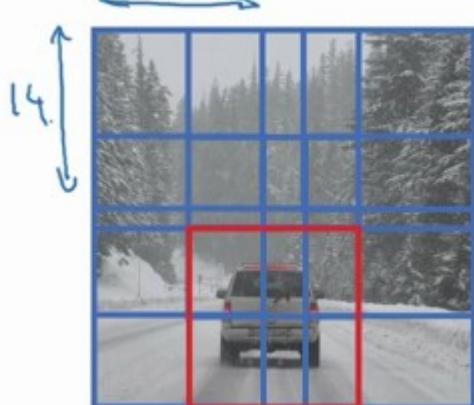
Si, por ejemplo, se tiene una imagen de $16 \times 16 \times 3$ con un stride de 2. El algoritmo se debería correr 4 veces para realizar la detección de objetos. Esto provoca que haya mucho cómputo repetido. Veamos que si, en su lugar, se realiza el cómputo de la siguiente red



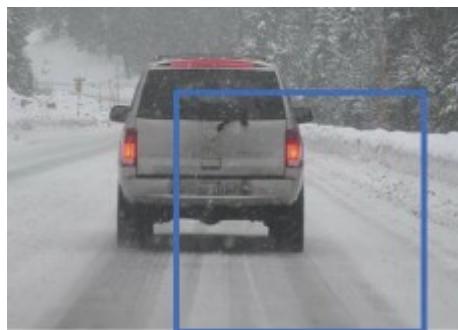
La salida será de, en lugar de $1 \times 1 \times 4$, de $2 \times 2 \times 4$, donde el primer subset de $1 \times 1 \times 4$ (arriba a la izquierda) corresponde a la primera ventana de $14 \times 14 \times 3$, el segundo subset (arriba a la derecha) corresponde a la segunda ventana de $14 \times 14 \times 3$ y así sucesivamente.

Un ejemplo más grande sería el siguiente, donde la salida se corresponde con un stride de 2.

Aplicando esto al problema del principio, la imagen se toma como entrada a la red y aplicando convolución la red detectará que en el recuadro rojo hay un auto efectivamente.



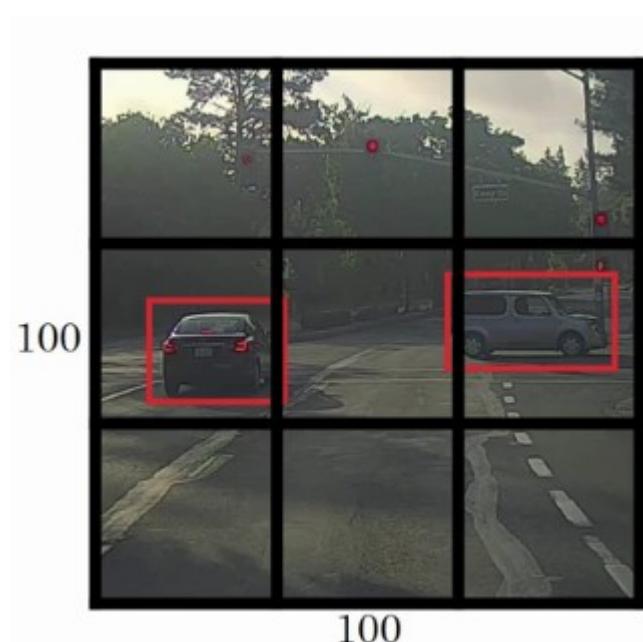
Este algoritmo todavía presenta un problema que está relacionado con la salida de los cuadrados (ver figura inferior). Veamos a continuación como solucionar este problema.



Predicciones de los cuadrados (bounding box)

Algoritmo YOLO (You Only Look Once)

Basado en el siguiente paper: [[Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.](#)]



Consideremos primero una imagen a analizar de 100x100. La misma se dividirá en una grilla que en este caso será de 3x3 pero que usualmente será más fina, digamos, de 19x19.

Cada una de estas grillas se procesará por una NN. La salida de las mismas vendrá dada por los parámetros que vimos a principio de la semana, estos son: p_c , si hubo detección, la posición del centro de la grilla, b_x, b_y , el ancho y alto del rectángulo, b_w, b_h y un conjunto de números que indica el tipo de detección que hubo c_1, c_2 y c_3 .

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_w \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_w \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

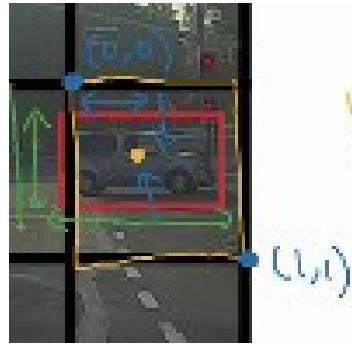
La salida de la primera grilla donde no hay objeto se obtendrá un 0 y un Don't care (nada) para todo lo demás. El algoritmo si detecta un objeto en la grilla, detecta el punto medio correspondiente al objeto que detectó.

La salida total (target output) de la red será de tamaño 3x3x8 o, lo que es lo mismo, #degrillas x 8, donde 8 es el número de parámetros que indican la detección del objeto y el tamaño de la grilla. Para la salida de la grilla del medio a la izquierda se obtendrá una de los vectores de la figura de la izquierda.

El problema de tener varios objetos en una grilla se discutirá más adelante.

Nota: El algoritmo no se corre un #de grillas veces sino que al ser convolucional se hace una vez, donde la salida viene dada por este número y cada uno de ellos equivale a una posición en particular de la grilla. Este algoritmo es lo suficientemente rápido como para funcionar en detección y clasificación de objetos en tiempo real.

¿Cómo se codean los parámetros b_x, b_y, b_w y b_h ?



Podemos notar que una grilla en particular posee extremos entre (0,0) y (1,1). La posición de la mitad del objeto se calcula y deben ser números entre 0 y 1, Por otra parte, el ancho y el alto del objeto no necesariamente tienen que ser menores a uno pueden ser mayores si el objeto es más ancho o alto que la grilla en la que se encuentra.

Ahora vemos algunas características que hacen funcionar el algoritmo un poco mejor.

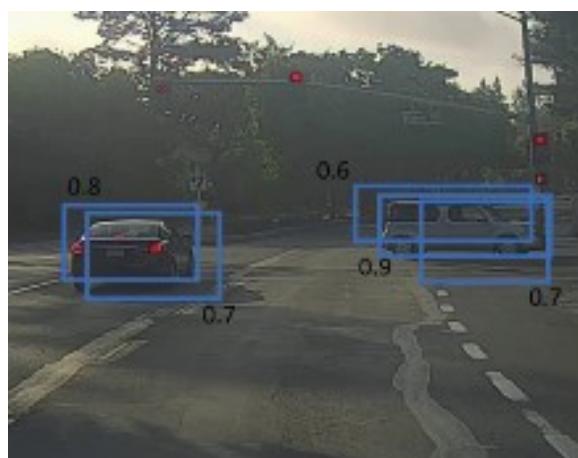
Intersección sobre unión (IoU)



Medida de solapamiento entre dos bounding boxes: Dado el rectángulo obtenido como salida al evaluar la localización de un objeto (azul) y el rectángulo correspondiente como verdadero (rojo), la intersección sobre unión viene dado como el cociente entre el tamaño del rectángulo de la intersección (amarillo) dividido el tamaño del rectángulo de la unión (rojo). En el caso de obtener un IoU mayor o igual a 0.5 (a veces se usa 0.6 también) la predicción de evaluar la localización del objeto se tomará como correcta. Mientras que si es menor a 0.5 será incorrecto.

¿Como evitar que un objeto se detecte una vez en lugar de detectar varias veces un solo objeto?

Podríamos usar Non-max suppression. Un ejemplo donde el algoritmo detecta varios objetos duplicados donde debería haber solo uno, se muestra en la figura de la derecha, cada rectángulo con una probabilidad. Non-max suppression lo que hace es quedarse con



el rectángulo de máxima probabilidad y descartar los que tengan mucho solapamiento entre si.

Implementación:

1. Descartar todos los rectángulos con probabilidad menor a 0.6.
2. Mientras hayan rectángulos restantes
 - a. Elegir un rectángulo con el p_c más grande. Predecir su salida.
 - b. Descartar cualquier rectángulo restante con IoU mayor o igual a 0.5 con respecto a la salida del paso anterior.

Anchor boxes

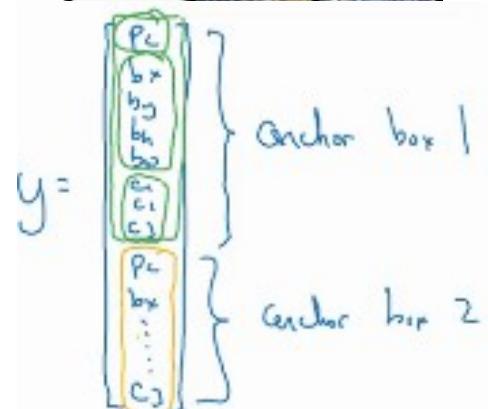
Volvamos al problema donde hay varios objetos por cuadrado de grilla, ¿Cómo hacemos para detectar dos o más objetos en un mismo cuadro? Veamos un ejemplo primero. En la figura de la derecha se puede notar que los puntos medios de los objetos están prácticamente en el mismo lugar.

Para ello la salida se concatena para poder realizar la detección.

Antes: cada objeto de entrenamiento era asignado a cada celda de la grilla que contiene el punto medio del objeto.

Ahora con dos anchor boxes: cada objeto en la imagen de entrenamiento es asignado a la celda de la grilla que contiene el punto medio del objeto y anchor box para la celda de la grilla con IoU más grande.

Todavía siguen habiendo varias cuestiones inconclusas. Pero antes previa detección se espera que se detecte objetos con una forma determinada denotada en la imagen como anchor box 1 y 2. Si hay dos anchor boxes y tres objetos, el algoritmo no funcionará correctamente. Si hay dos objetos con el mismo tipo de anchor box tampoco funcionará correctamente. Cuando se usa un grillado chico el uso de anchor boxes no es tan importante dado que es poco probable que haya detección de varios objetos en una región tan compacta de la figura. Por otra parte, cabe agregar que el tipo anchor box 1 se suele utilizar para personas (altas y angostas) mientras que el tipo 2 suele ser utilizado para automóviles (anchos y bajos). En la práctica se suelen elegir entre 5 y 10 tipos de anchor boxes.



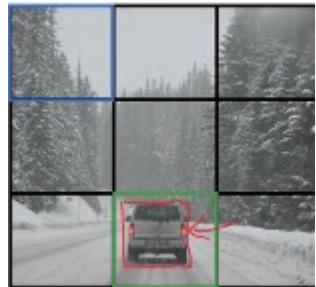
Anchor box 1: Anchor box 2:



Juntando todo lo visto en el algoritmo YOLO

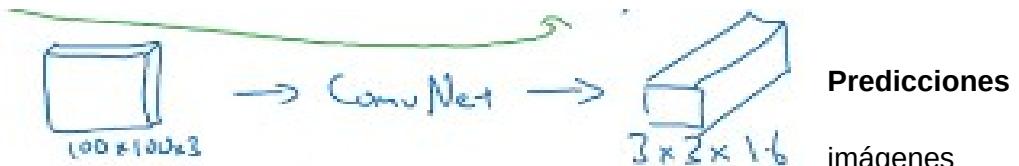
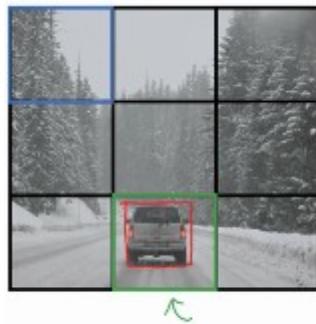
Entrenamiento

Dada la siguiente figura y una grilla de 3x3 donde se quieren detectar 1. personas, 2. autos o 3. motocicletas, la salida poseerá un tamaño dado por $3 \times 3 \times 2 \times 8$, donde 2 = #anchor boxes (esto suele ser a elección. Generalmente se eligen 5) y $8 = 5 + \#clases = \#parametros de$



objeto localizado. Usualmente el tamaño de la salida suele ser de 19x19x40 debido al aumento del grillado y el número de anchors a 5. La salida del elemento 1x1 al no haber detección vendrá dada por un 0 en el elemento p_c y un don't care en los restantes, mientras que en la celda inferior central hay una detección de un 2. automovil por lo cual elemento 3x2 de tamaño 16 tiene sus primeros 8 elementos como un 0 en p_c y don't cares en el resto, los siguientes 8 elementos vienen dados por un 1 en p_c dada la detección positiva, los siguientes cuatro numeros determinan el tamaño y posición del rectángulo localizador y los últimos 3 la clase de objeto detectado, en este caso, 0 1 0. Por último, la entrada viene a ser una imagen de 100x100x3 en este caso que pasa por una ConvNet y otorga la salida dicha anteriormente de 3x3x16.

Nuevas
serán
entradas a la red neuronal y si todo funciona bien predecirá lo que es esperado a la salida de la misma.



Predicciones

imágenes
pasadas como

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Salida del algoritmo non-max suppressed

- Para cada celda de la grilla, se obtienen dos cajas de predicciones.
- Deshacerse de las predicciones con probabilidad baja.
- Para cada clase (persona, auto, moto) usar non-max supression para general la predicción final.

Región de propuestas (R-CNN, Region proposal)

Basado en el paper: [\[Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.\]](#)



Básicamente como la figura muestra hay figuras en las que grandes regiones no importantes pueden ser estudiadas perdiendo gran tiempo de cómputo. En lugar de correr sliding

windows en todas las ventanas se hace en una cierta cantidad. Esto se detecta corriendo un algoritmo llamado algoritmo de segmentación



Como muestra la figura, el algoritmo va detectando figuras de interés y la red se corre únicamente en estos objetos que sean de interés, por ejemplo, del orden de 2000.

R-CNN: Propone regiones. Clasifica regiones propuestas una a la vez. Salida del rectángulo localizador y etiqueta.

Algoritmos más rápidos

Fast R-CNN: Usa la implementación convolucional de ventanas deslizantes para clasificar todas las regiones propuestas. Las regiones para proponer regiones siguen siendo lo más lento. Paper: [[Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.\]](#)

Faster R-CNN: Usa redes convolucionales para proponer regiones. Paper: [[Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.\]](#)

Aplicaciones especiales: Reconocimiento de caras & Neural Style transfer

¿Qué es reconocimiento facial?

En las oficinas de Baidu, China se utiliza reconocimiento facial para dejar entrar a una persona a cierto lugar, pero a su vez usa la técnica de detección de “liveness” para evitar que alguien entre con una cara de la persona impresa en una foto, por ejemplo. Esto es logrado simplemente con aprendizaje supervisado, pero en el curso se prestará más atención al reconocimiento facial.

Verificación facial vs Reconocimiento facial

La verificación es un proceso más simple (1-1). Se necesita una imagen de entrada y un nombre o ID con el objetivo de determinar si la imagen es la de la persona en cuestión. Por otra parte, en reconocimiento se posee una base de datos de K personas (1-K), se requiere una imagen como entrada y como salida se obtiene la ID en el caso de que la imagen coincida con alguna de las K personas (o no reconocida en el caso de no estar en la base de datos). Cabe notar que, si se posee un algoritmo de verificación facial de 99% de accuracy, el reconocimiento para dadas 100 personas, será mucho menor, por lo cual un accuracy del orden de 99.9% es requerida.

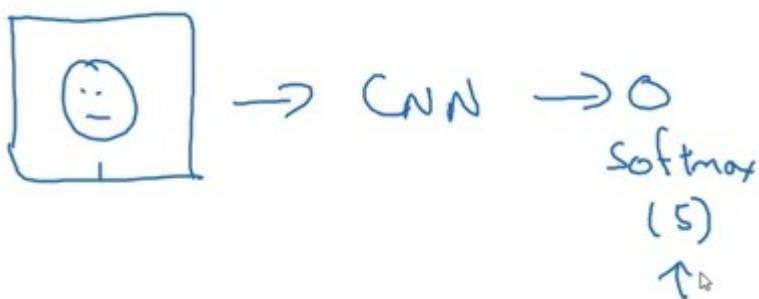
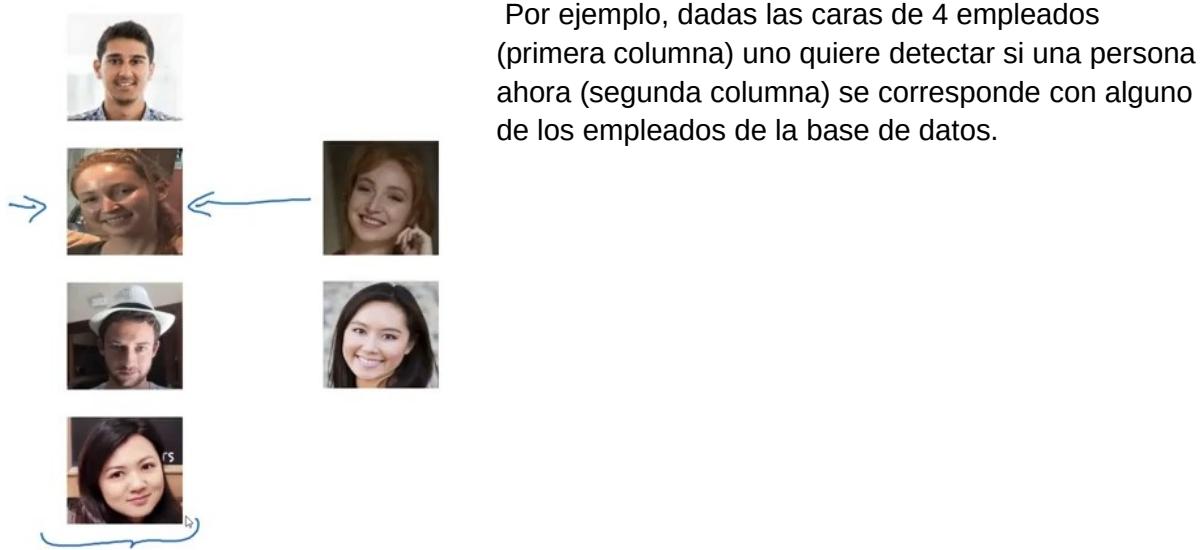
Face verification: "is this the claimed person?". For example, at some airports, you can pass through customs by letting a system scan your passport and then verifying that you (the

person carrying the passport) are the correct person. A mobile phone that unlocks using your face is also using face verification. This is a 1:1 matching problem.

Face Recognition - "who is this person?". For example, the video lecture showed a face recognition video (<https://www.youtube.com/watch?v=wr4rx0Spihs>) of Baidu employees entering the office without needing to otherwise identify themselves. This is a 1:K matching problem.

One Shot Learning

Uno de los desafíos más importantes para el reconocimiento facial es el hecho de poder reconocer una persona con una única imagen de ejemplo.



Una manera de tratar este problema sería agarrar las 4 fotos y entrenar una CNN con una salida Softmax (5) cada una correspondiente a una persona y una para ninguna. Pero dada la baja cantidad de empleados esto no funcionará correctamente. Por otra parte si un nuevo empleado es tomado, la salida deberá ser Softmax(6) lo cual provocaría que se tenga que entrenar la red nuevamente, siendo un proceso sumamente ineficiente.

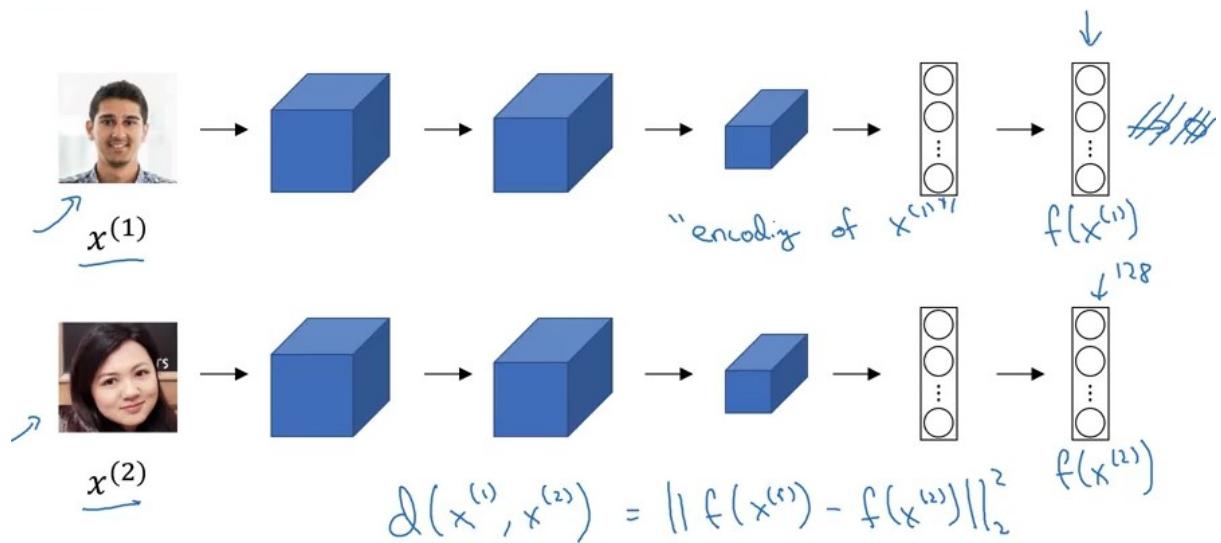
En su lugar, se aprenderá una función de “similitud”. Esta mide la diferencia entre las imágenes y se compara si esta es menor a una threshold. En caso afirmativo se considera que las imágenes pertenecen a la misma persona, mientras que si la diferencia es mayor al threshold se consideran a las personas distintas. Como se muestra en la figura de la izquierda, se espera que, si las personas son las mismas, la diferencia entre ellas sea menor al threshold y que para las restantes personas sea mayor. Por otra parte, si una nueva persona es incorporada a la empresa, no habrá que entrenar una red como sucede con la CNN.



Pero, ¿Cómo se entrena una red neuronal para aprender esta función?

Siamese Network

[Esta red fue introducida en el siguiente trabajo: [Taigman, Yaniv, et al. "Deepface: Closing the gap to human-level performance in face verification." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.](#)]



Básicamente, dada dos imágenes que se quieren comparar, estas se insertan en una red como la de la figura compuesta por ejemplo de filtros y dos capas completamente conectadas (FC) (sin Softmax). Esto hecho por cada imagen se llama Encoding de $x(i)$ donde la salida de la red es $f(x(i))$. La manera de entrenar esta red es aprender los parámetros de manera tal que la diferencia entre estos Encodings sea pequeña dadas imágenes de la misma persona.

Learn parameters so that:

If $x^{(i)}, x^{(j)}$ are the same person, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\|f(x^{(i)}) - f(x^{(j)})\|^2$ is large.

Pero, ¿Cómo se define una función de costo que entrene los parámetros de la red?

Triplet Loss

[Este trabajo fue desarrollado principalmente del paper siguiente: [Schroff, Florian, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.](#)]

Dadas dos personas iguales yo quiero que los Encodings sean pequeños, mientras que son si no son la misma, que sea grande. En la terminología Anchor (A) es la imagen con la que comparar unas imágenes Positiva (P) o negativa (N), estas son la misma persona u otra, respectivamente.



Anchor
A



Positive
P



Anchor
A



Negative
N

Básicamente, queremos que la diferencia entre las imágenes anchor y positive sea menor que la anchor y negative, pero para evitar que la red entrene soluciones triviales (iguales a cero) se agrega un hiper parámetro llamado margen (α).

$$\text{Want: } \frac{\|f(A) - f(P)\|^2}{d(A, P)} \leq \frac{\|f(A) - f(N)\|^2}{d(A, N)}$$

$$\frac{\|f(A) - f(P)\|^2}{\alpha} - \frac{\|f(A) - f(N)\|^2}{\alpha} + \alpha \leq 0 \quad \text{if } f(\text{img}) = \vec{0}$$

Por ejemplo, en el caso de que α fuera cero, si la diferencia entre Anchor y positive fuera 0.5, bastaría con que la diferencia entre anchor y negative sea 0.51 por ejemplo, pero si elegimos $\alpha = 0.2$ haría que la diferencia $d(A, N)$ sea del orden de 0.7.

Definamos la función de perdida como:

$$L(A, P, N) = \max \left(\underbrace{\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2}_{\text{constraint}} + \lambda, 0 \right)$$

Para minimizar esto, la red quiere que el primer término sea lo más negativo posible. La función de coste total sobre un training set será la suma de esta función de coste sobre cada imagen.

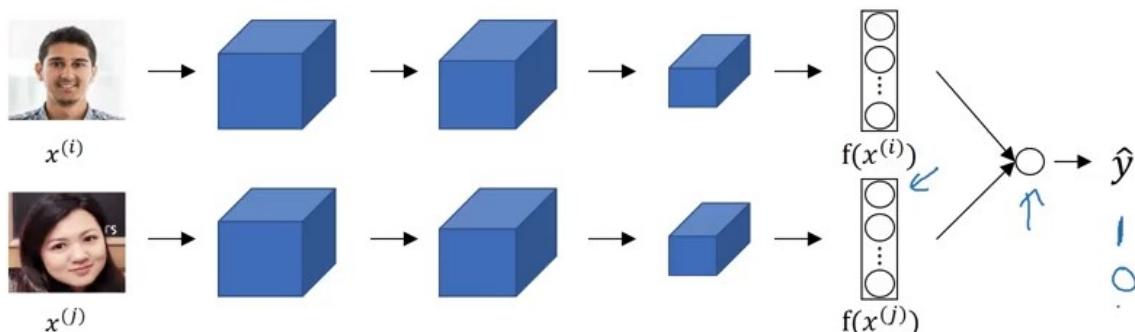
$$J = \sum_{i=1}^m L(A^{(i)}, P^{(i)}, N^{(i)})$$

Training set: 10k pictures of 1k persons

Para hacer esto necesitamos si o si pares de imágenes de la misma persona.

Un inconveniente con esto es que si se eligen A, P y N aleatoriamente entonces el constrain será fácilmente satisfecho, ya que $d(A, N)$ será muy grande. Es por ello que conviene utilizar triplets de tal manera que la red sea "difícil" de entrenar.

Verificación facial y clasificación binaria



El reconocimiento facial puede ser pensado como un problema de clasificación binaria mediante la figura superior.

¿Qué es lo que hace la unidad de regresión logística final? La salida y_{hat} será evaluar una función sigmoide digamos a la diferencia entre vectores de la siguiente manera

$$\hat{y} = \sigma \left(\sum_{k=1}^{128} w_i \underbrace{|f(x^{(i)})_k - f(x^{(j)})_k|}_{\text{difference}} + b \right)$$

Esta unidad puede tener parámetros w y b adicionales los cuales deben ser entrenados para predecir si o no las imágenes son de la misma persona. Otra manera de computar esto sería mediante la diferencia chi cuadrada entre $f(x(i))$ y $f(x(j))$ como

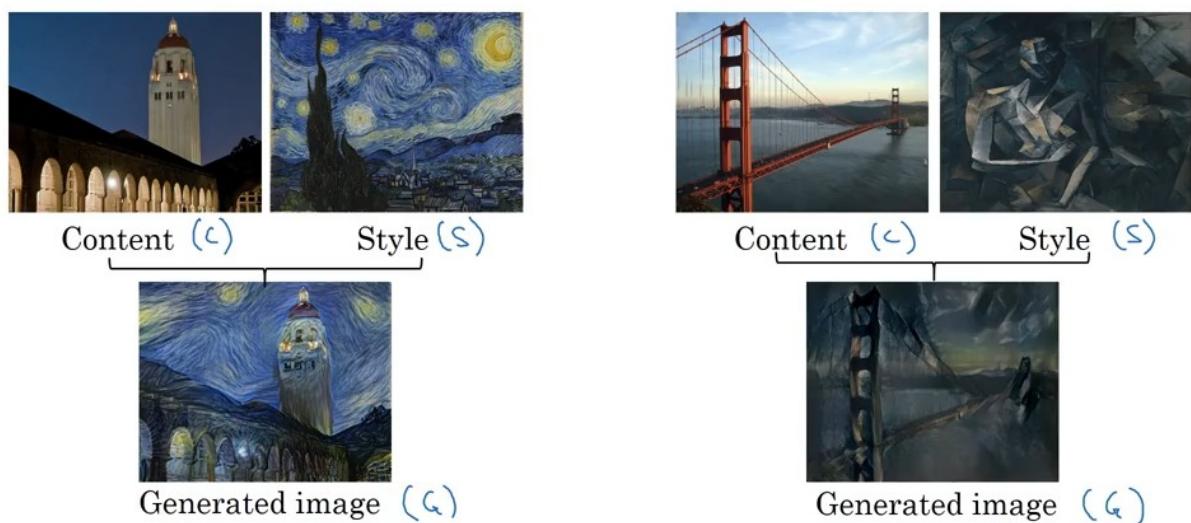
$$\frac{(f(x^{(i)})_k - f(x^{(j)})_k)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$$

La entrada es un par de imágenes. La salida es 1 o 0 dependiendo si las imágenes son similares o no. Tanto la red siamesa superior como inferior poseen parámetros los cuales hay que entrenar.

Una manera de agilizar el entrenamiento consiste en si, un empleado ingresa en lugar de re computar toda la red nuevamente, lo que se puede hacer es pre-computarla así, si alguien nuevo entra, usa esos Encodings para realizar la predicción y_{hat} .

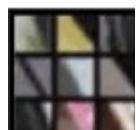
Neural Style Transfer

Aplicación de las CNN más interesantes, genera obras de arte. Dada una imagen a la que llamamos contenido (c) se le va a aplicar un estilo de imagen (s), la salida de este proceso se llamará (g).



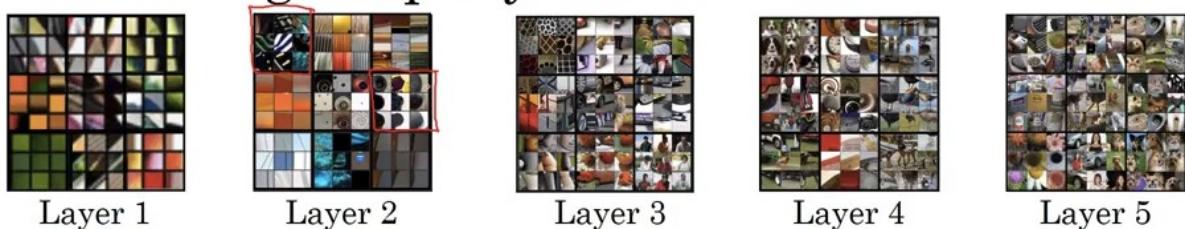
En estos ejemplos, ¿qué es lo que las capas profundas de la red convolucional aprende?

[Esta sección está basada en la investigación realizada por el paper: [Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.](#)]



Tomemos por ejemplo una red tipo AlexNet. Agarremos una unidad oculta de la primera capa y miremos la imagen que maximiza esa activación de la unidad. Podemos ver que buscará líneas inclinadas hacia la izquierda. Si agarramos otras buscará líneas inclinadas hacia la derecha, otras líneas horizontales anaranjadas. Es decir que en la primera capa se distinguen fragmentos simples como bordes (ver imagen inferior).

En las capas más profundas de la red neuronal veremos estructuras más profundas. En resumen, parece que la capa 1 detecta bordes. La capa 2, texturas verticales más complejas, figuras redondas en la parte superior de la imagen, líneas delgadas, etc. En la capa 3, tenemos personas, figuras redondas, texturas, etc. En la capa 4, se detectan patas de animales, agua, perros pero muy similares entre si, etc. Por último, la capa 5 parece detectar cosas más sofisticadas, perros más diferentes entre si, flores o texto.



Función de coste

[La idea principal de Neural Style Transfer es obra del siguiente paper: [Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style." arXiv preprint arXiv:1508.06576 \(2015\).](#)]



Definamos una función de coste que mide cuán bien es una imagen generada. Esta posee dos partes: una que mide qué tan bien la imagen generada representa el contenido de C y que tan bien la imagen generada representa el contenido de S, cada una con su respectivo peso asociado.



$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$



En orden para generar la nueva imagen lo que se hace es lo siguiente

1. Iniciar G aleatoriamente, por ejemplo, G: 100x100x3.
2. Usar gradiente descendente para minimizar J(G).



Es por ello que a medida que aumentan las iteraciones, la imagen generada pasa de ser una imagen completamente ruidosa a algo que adquiere el estilo de S, pero manteniendo el contenido de C.

Función de coste de contenido

Digamos que usamos una capa oculta I para computar el costo. Si es la primera capa por ejemplo la imagen generada aleatoriamente deberá ser similar a C para no haber tanta diferencia. En la práctica, se utiliza un valor intermedio entre las primeras y las ultimas.

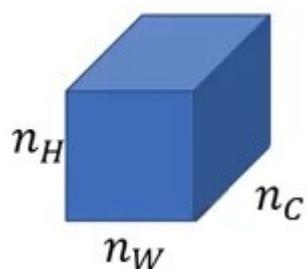
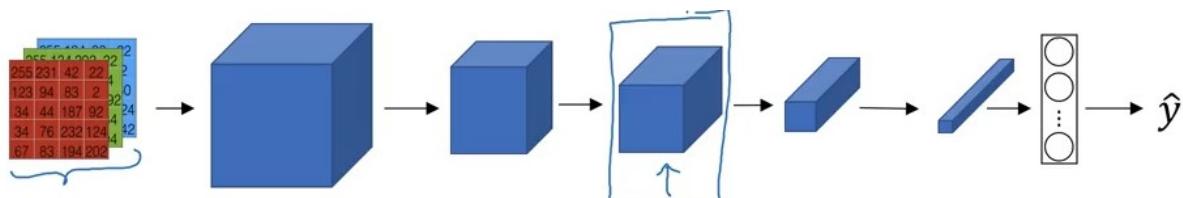
Usa una red entrenada previamente por ejemplo como ConvNet. ¿Cuán similares son C y G en contenido? Sean $a^{[l](C)}$ and $a^{[l](G)}$ las activaciones de la capa l en las imágenes C y G respectivamente. Si ambos números son similares, las imágenes poseen contenidos similares. Entonces se define la función de coste como

$$J_{content}(C, G) = \frac{1}{2} \|a^{[l](C)} - a^{[l](G)}\|^2$$

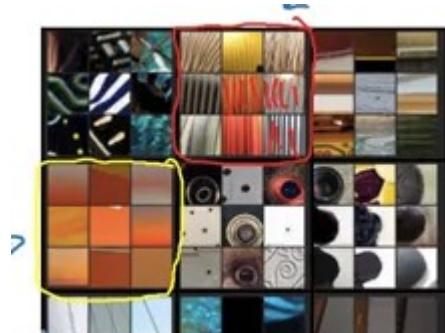
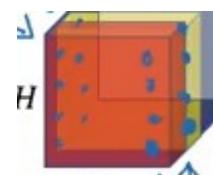
Función de coste de estilo

¿Qué significa el estilo de una imagen?

Dada la siguiente red, considerando la capa l sobre la cual se mide el estilo. Definamos estilo como la correlación entre activaciones a lo largo de los canales.



Dada la capa en cuestión, veamos cuan correlacionadas están las activaciones a lo largo de los diferentes canales (N_c). Veamos por ejemplo los primeros dos canales, para un h y w fijos tenemos definidos un par de puntos correspondientes uno a cada canal. Esto nos define para todo H y W conjuntos de puntos del cual tenemos que medir la correlación.



Viéndolo en base a un ejemplo anterior, recordemos que para la capa 2 teníamos la figura de la izquierda. Si un canal (rojo) se corresponde con las figuras de las líneas verticales y el otro (amarillo) se corresponde con la de las figuras horizontales, entonces una manera de medir correlación seria por ejemplo, si siempre que se obtienen líneas verticales rojas, en la otra capa hay figuras horizontales naranjas.

Dada una imagen, computemos una matriz de estilo (Gram matrix en álgebra lineal). Sea

$a_{i,j,k}^{[l]}$ la activación en el punto (i,j,k) donde $i=1,\dots,n_h$, $j=1,\dots,n_w$ y $k=1,\dots,N_c$. Por otra parte, la matriz de estilos que define la correlación entre dos

$$G_{k,k'}^{[l]} = \sum_{i,j} a_{i,j,k}^{[l]} a_{i,j,k'}^{[l]}$$

$$k, k' = 1, \dots, N_c$$

capas de activación k y k' posee la simbolización definida a la izquierda. Un elemento de esta matriz entonces se calcula de la siguiente manera

$$\overline{G}_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

Cabe aclarar que esto no es una correlación sino una cross-covarianza sin normalizar. Esto se realiza tanto para la imagen S como con G . Para mejorar la notación mejor escribimos

$$\begin{aligned} \overline{G}_{kk'}^{[l](S)} &= \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l](S)} a_{ijk'}^{[l](S)} \\ \overline{G}_{kk'}^{[l](G)} &= \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l](G)} a_{ijk'}^{[l](G)} \end{aligned}$$

De esta manera la función de coste de estilo viene definida según

$$J_{style}^{[l]}(S, G) = \| G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)} \|_F^2$$

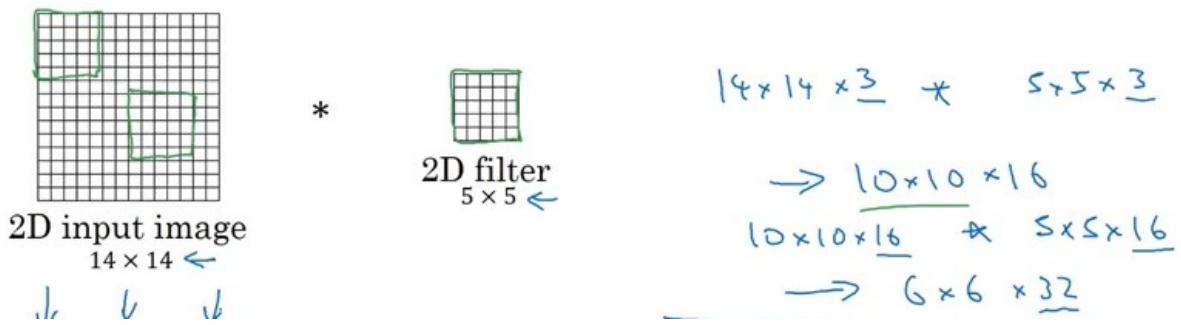
$$J_{style}^{[l]}(S, G) = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_C^{[l]})^2} \sum_k \sum_{k'} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})$$

Se obtienen mejores resultados calculando la función de coste para distintas capas, que tienen en cuenta tanto high-level features como low-level.

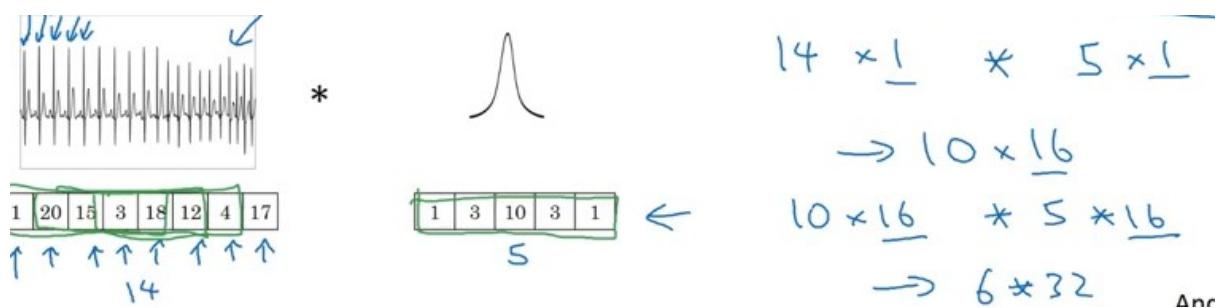
$$J_{style}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S, G)$$

Generalización a 1D y 3D

Hasta ahora hemos usado todo esto teniendo en cuenta imágenes en 2D, pero podemos ver cómo aplicarlo a otros tipos de datos. Recordemos que la convolución en 2D venia dada de la siguiente manera



En 1D se puede aplicar algo similar como sigue



Un ejemplo de dato de este estilo podría ser un electrocardiograma. Para 3D podemos pensar un CT scan, que toma distintos slices del cuerpo. Una convolución a un volumen de 3 dimensiones se puede calcular como sigue

