



deeplearning.ai

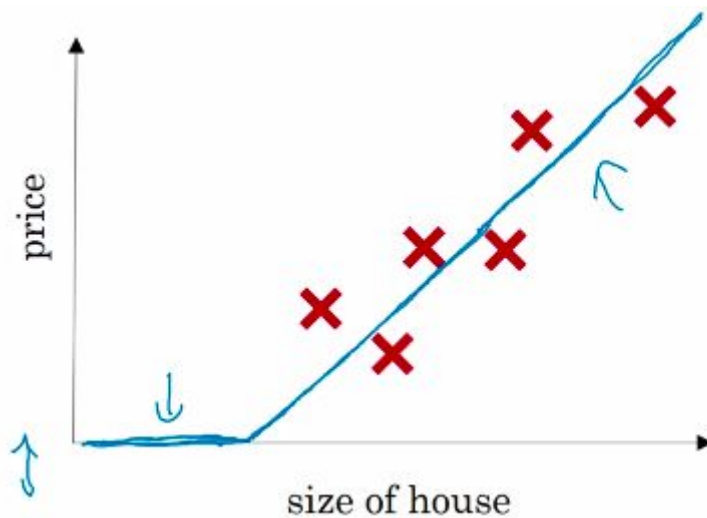
Introduction to Deep Learning

Qué se verá en la Especialización

1. Redes Neuronales y Deep Learning, los fundamentos.
Como entrenarlos con datos. Construir una red neuronal para distinguir gatos.
2. Como hacer que la red neuronal funcione correctamente: tuneado de hyperparametro, regularizacion y optimizacion.
3. Como 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 aplicando a imagenes.
5. NLP (Natural Language Processing): Construir modelos secuenciales. Usando RNN (Recurrent Neural Networks) o LSTM (Long Short Term Memory).

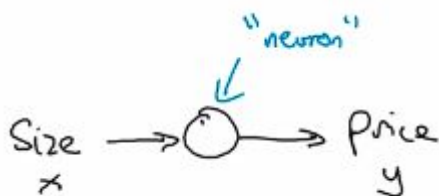
Que es una red neuronal

Prediccion Precio de casa



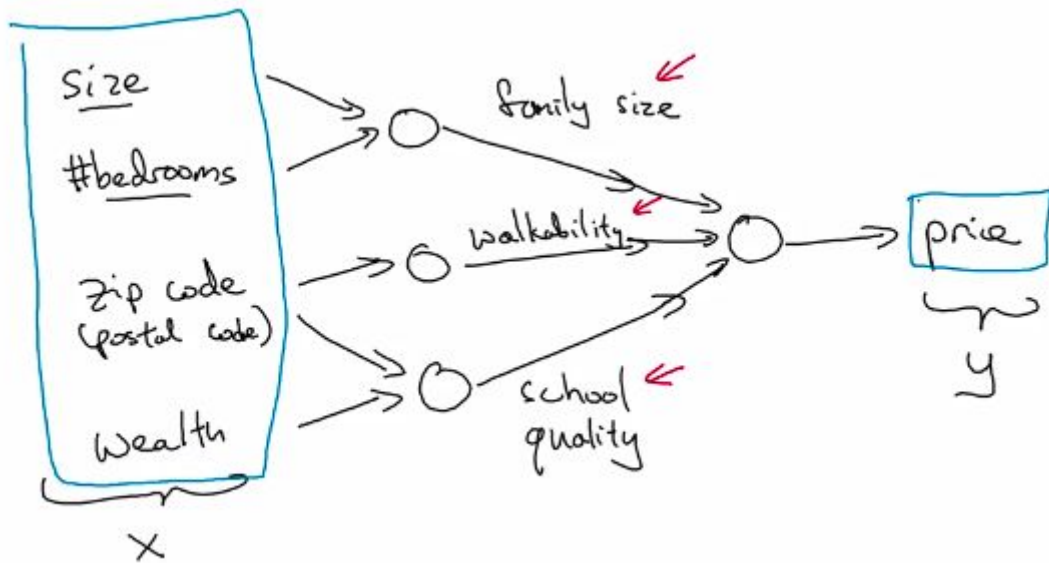
Los datos parecen ajustarse con un ajuste lineal. Dado que los precios no pueden ser negativos podemos agregar una seccion donde el ajuste seria cero.

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



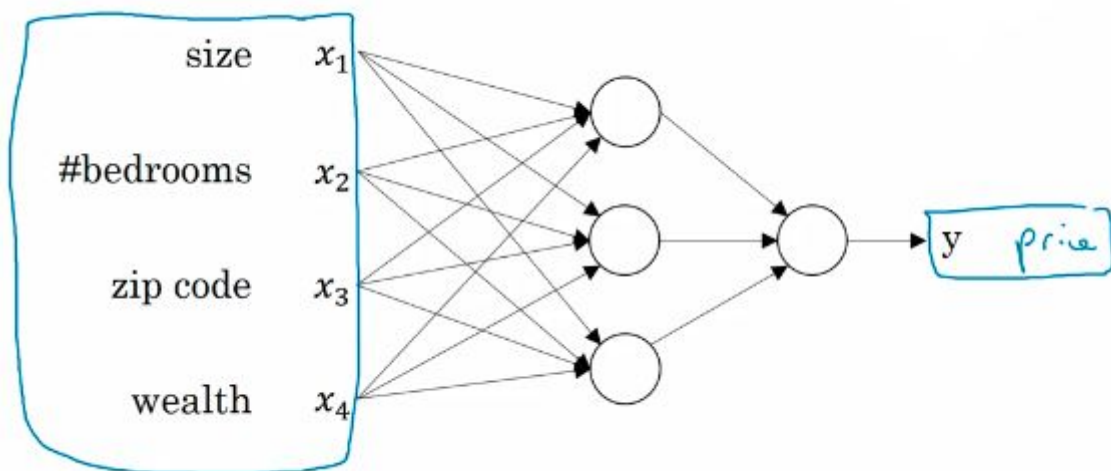
Este tipo de funciones que es cero y luego lineal es llamada RELU (Rectified Linear Unit). Una red neuronal mas compleja se podria armar uniendo "neuronas" simples como la recién creada.

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



Lo bueno es que en un modelo de redes neuronales, no necesitamos conocer los intermedios, 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.

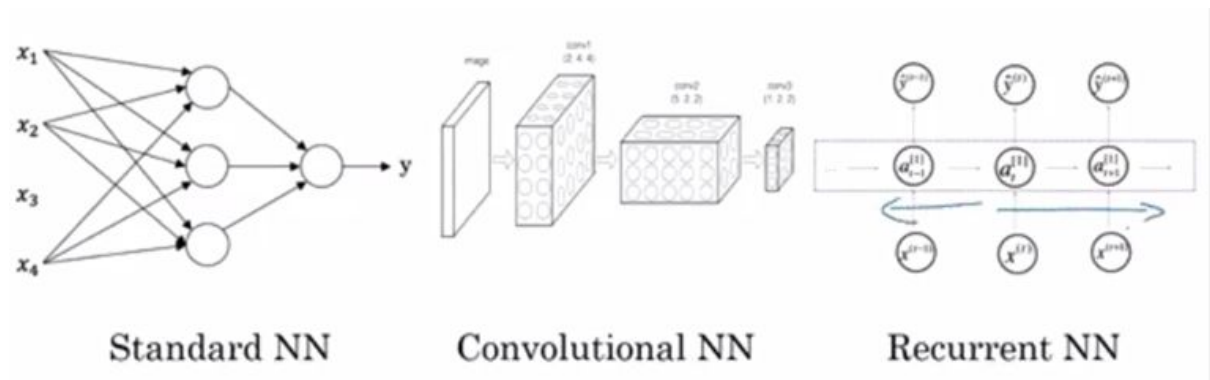
Aprendizaje Supervisado con Redes Neuronales

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



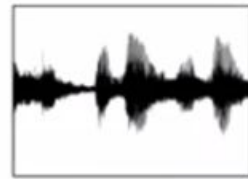
Aplicaciones de Machine Learning sobre datos estructurados o no estructurados

Structured Data

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

Unstructured Data



Audio



Image

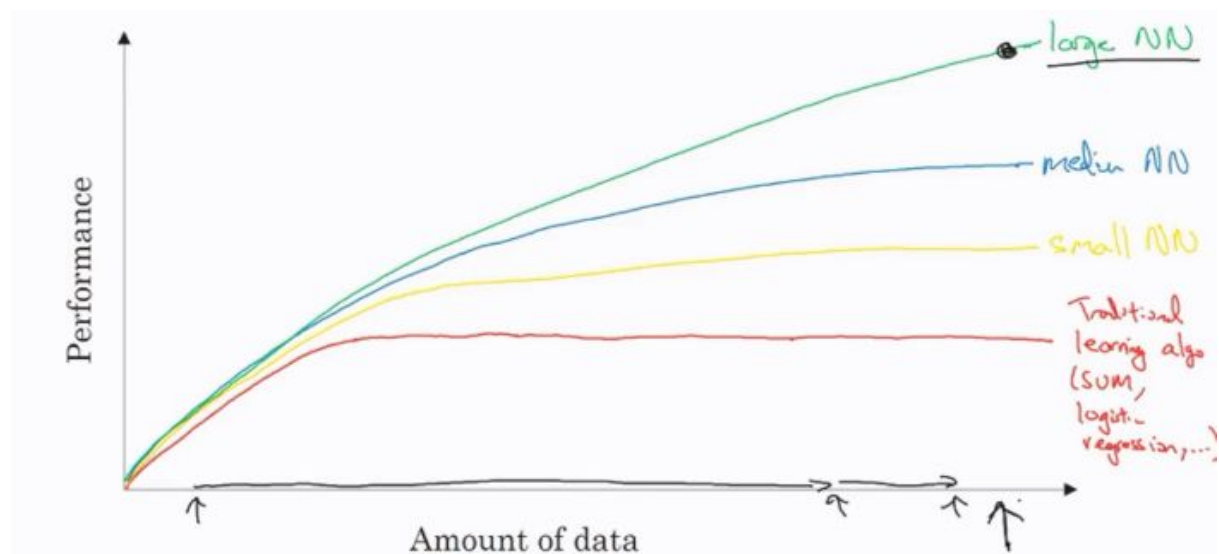
Four scores and seven
years ago...

Text

Por que esta despegando Deep Learning??

La cantidad de informacion 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 habria que tener una red neuronal grande, con muchas unidades ocultas y mucha informacion.

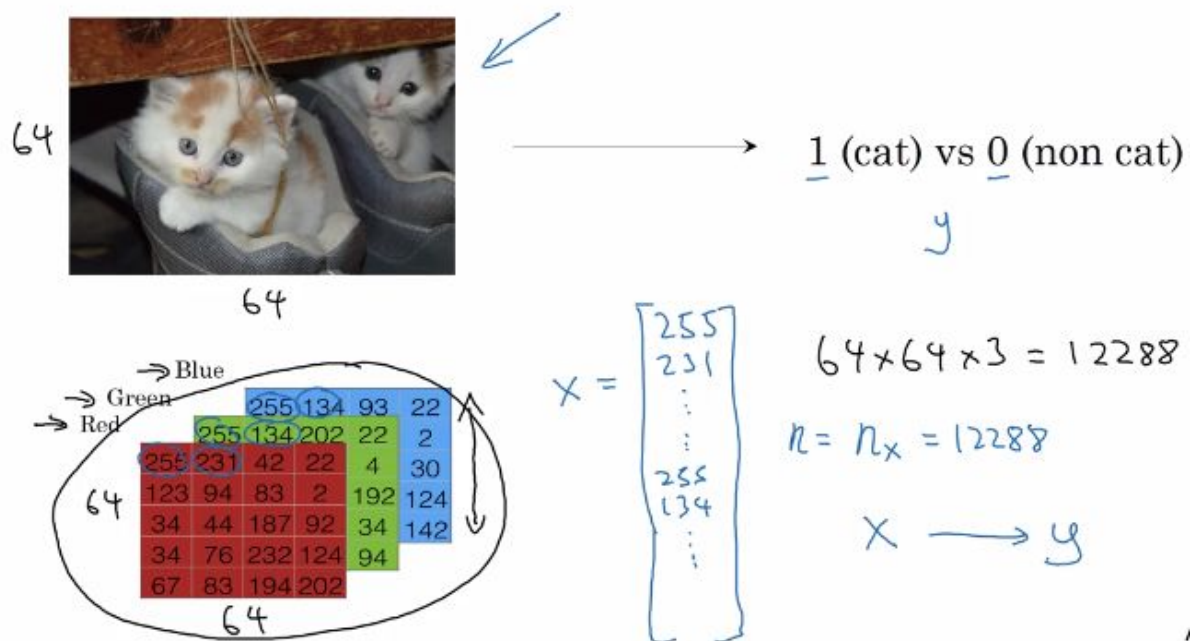


Fundamentos de la programación de redes neuronales

Clasificación 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 determinar Y , si es gato (1) o no (0).



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

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

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

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

\xleftarrow{m}

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

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

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

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

Regresion logistica

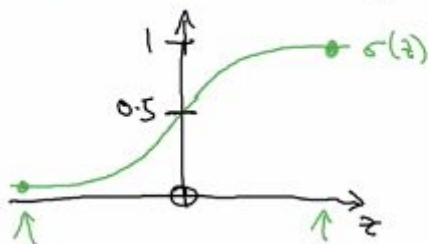
La idea es determinar w y b . como z puede tomar cualquier valor real, se le aplica la funcion sigmoide para que tome valores entre 0 y 1, correspondientes a una probabilidad.

Given x , want $\hat{y} = \frac{P(y=1|x)}{0 \leq \hat{y} \leq 1}$

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

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

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



Casos extremos

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

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

If z large negative number

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

La notacion del curso de Machine Learning de Andrew Ng no se utilizara.

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

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

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \leftarrow \\ \\ \\ \\ \} w \leftarrow \end{matrix}$$

Funcion de coste de la regresion logistica

para entrenar los paramatros w y b se necesita una funcion de coste.

La funcion de perdida (Loss) esta asociada a un ejemplo unicamente. Puede definirse como el error cuadrado pero no es convexa. Por eso se define de otra manera.

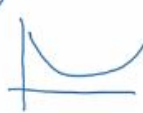
La funcion de coste se aplica sobre todo el conjunto de entrenamiento, minimizando la funcion para cada uno de estos puntos. Los valores w y b que minimizan la funcion seran los de interes.

A continuacion se vera a la regresion logistica como una red neuronal muy pequeña.

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

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

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

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


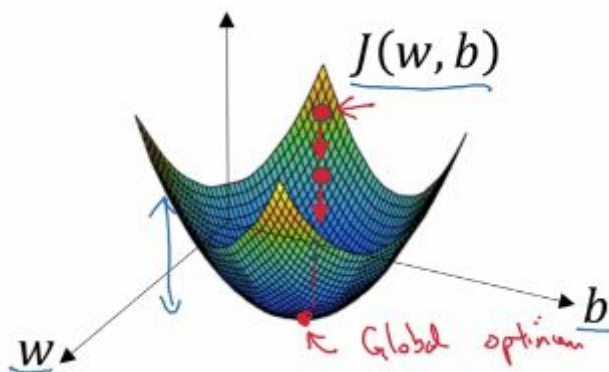
If $y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y} \leftarrow$ want $\log \hat{y}$ large, want \hat{y} large.

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

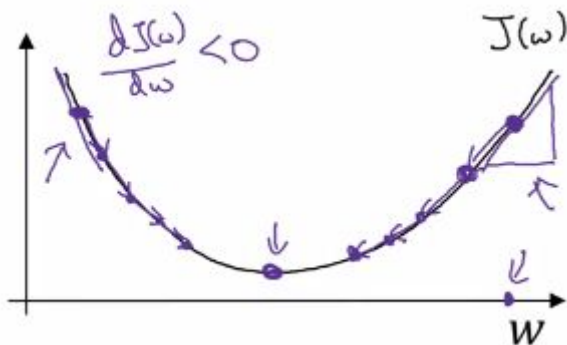
Cost function: $J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$

Gradiente descendente

como se minimiza $J(w, b)$?



Pensando el problema para un b fijo,



el algoritmo consiste en repetir la ecuacion hasta que w o b no varien drasticamente.

Repeat {

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

}

learning rate

$w := w - \alpha dw$

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

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

Grafico computacional

simplemente para tener una idea de back y forward propagation, veamos con un ejemplo como una funcion de varias variables puede ser evaluada hacia adelante "forward" mientras que para derivarla habria que hacerlo hacia atras "backward".

En un ejemplo la funcion J depende de tres variables a, b y c . Las flechas azules representan la propagacion hacia adelante para evaluar la funcion (calculo de w y b), mientras que las rojas representan las derivadas (gradient descent).

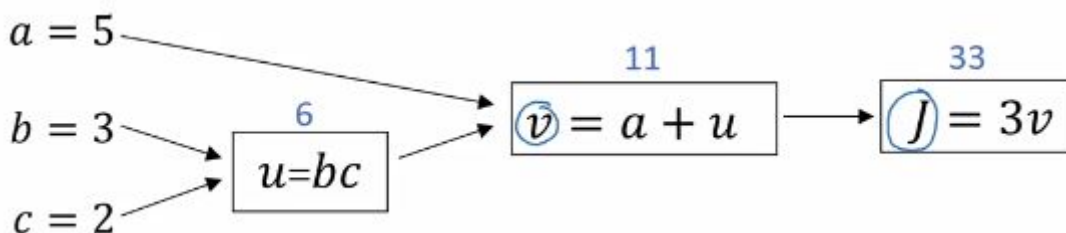
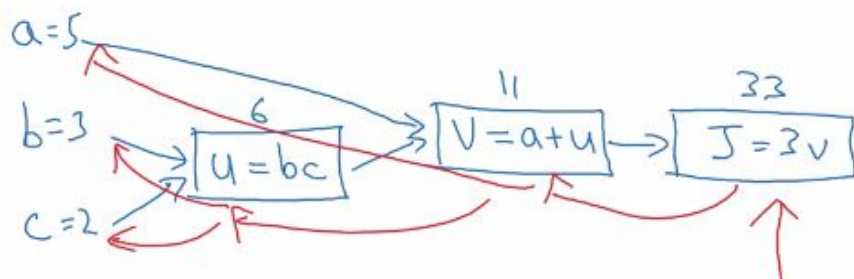
$$J(a,b,c) = 3(a + \underbrace{bc}_u) = 3(5 + 3 \times 2) = 33$$

$\underbrace{\quad}_J$

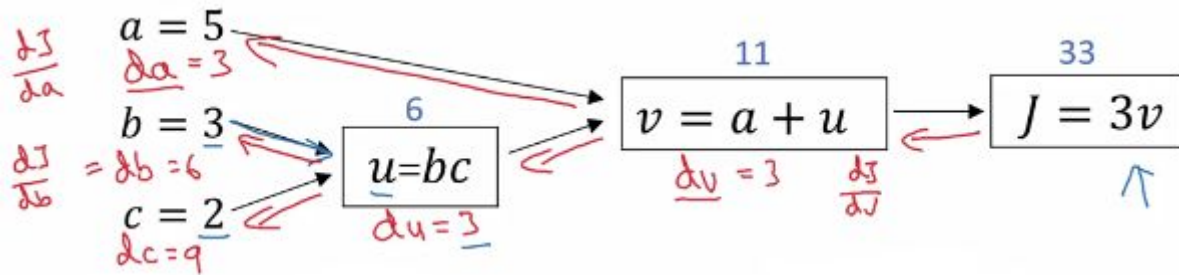
$$u = bc$$

$$v = a + u$$

$$J = 3v$$



Derivadas en un grafico computacional



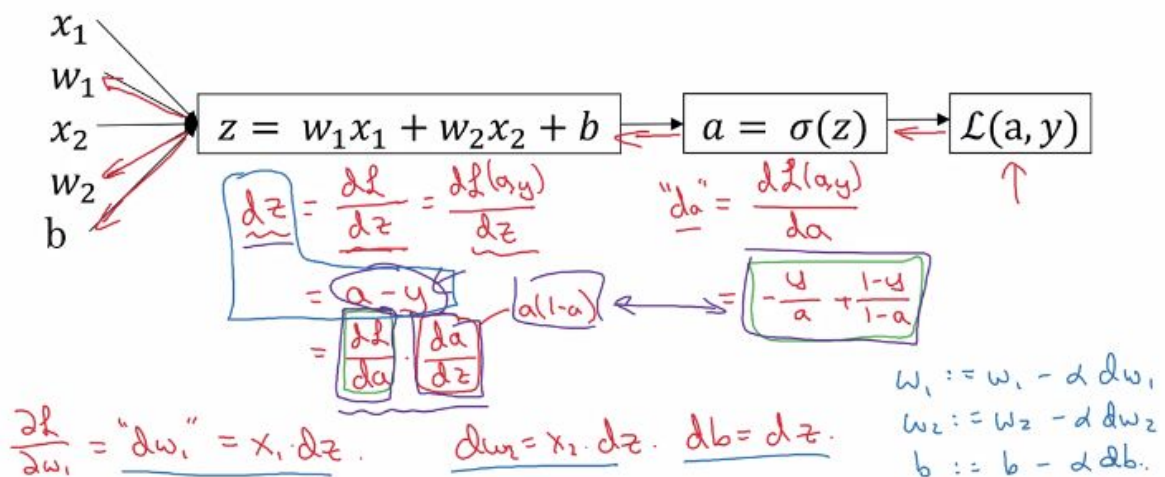
Gradiente descendente en regresion logistica

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

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

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

Considerando dos inputs, x_1, x_2 , para calcular w_1, w_2 y b primero habria que calcular dz y luego multiplicar por su correspondiente input.



Gradiente descendente en m ejemplos

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

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

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

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

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

El algoritmo se puede escribir de la siguiente manera

$J=0; \underline{dw_1}=0; \underline{dw_2}=0; \underline{db}=0$ <p><u>For</u> $i=1$ <u>to</u> m</p> $z^{(i)} = w^T x^{(i)} + b$ $a^{(i)} = \sigma(z^{(i)})$ $J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log (1-a^{(i)})]$ $\underline{dz^{(i)}} = a^{(i)} - y^{(i)}$ <div style="display: flex; align-items: center; justify-content: center;"> <div style="text-align: right; margin-right: 10px;"> \uparrow $\underline{dw_1}$ $\underline{dw_2}$ </div> <div style="text-align: center;"> $dw_1 += x_1^{(i)} dz^{(i)}$ $dw_2 += x_2^{(i)} dz^{(i)}$ $db += dz^{(i)}$ </div> <div style="text-align: left; margin-left: 10px;"> \downarrow $n=2$ </div> </div> $J /= m$ <div style="display: flex; align-items: center; justify-content: center;"> <div style="text-align: center; margin-right: 10px;"> \uparrow $dw_1 /= m$ </div> <div style="text-align: center; margin-right: 10px;"> \uparrow $dw_2 /= m$ </div> <div style="text-align: center;"> \uparrow $db /= m$ </div> ← </div>	$dw_1 = \frac{\partial J}{\partial w_1}$ $w_1 := w_1 - \alpha \underline{dw_1}$ $w_2 := w_2 - \alpha \underline{dw_2}$ $b := b - \alpha \underline{db}$
---	--

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

En este algoritmo tenemos dos for loops. Para que sea mas eficiente se deberia vectorizar.

Vectorizacion

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

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
```

Vectorized logistic regresion's gradient descent

Todavia se necesitaria un for-loop sobre el numero de iteraciones que se quieren para minimizar la funcion de coste.

$$\begin{aligned} z &= w^T X + b \\ &= \text{np.dot}(w.T, X) + b \\ A &= \sigma(z) \\ dz &= A - Y \\ dw &= \frac{1}{m} X dz^T \\ db &= \frac{1}{m} \text{np.sum}(dz) \\ w &:= w - \alpha dw \\ b &:= b - \alpha db \end{aligned}$$

Broadcasting en Python

Es una tecnica de Python para hacer el codigo funciionar mas rapido.

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))
```

donde axis=0 indica sumar las por columna.

En percentage se dividio A que tiene dimension 3x4 por cal que tiene dimension 1x4.

Python/ Numpy vectors

```
a = np.random.randn(5)
a.shape = (5,)
"rank 1 array"
```

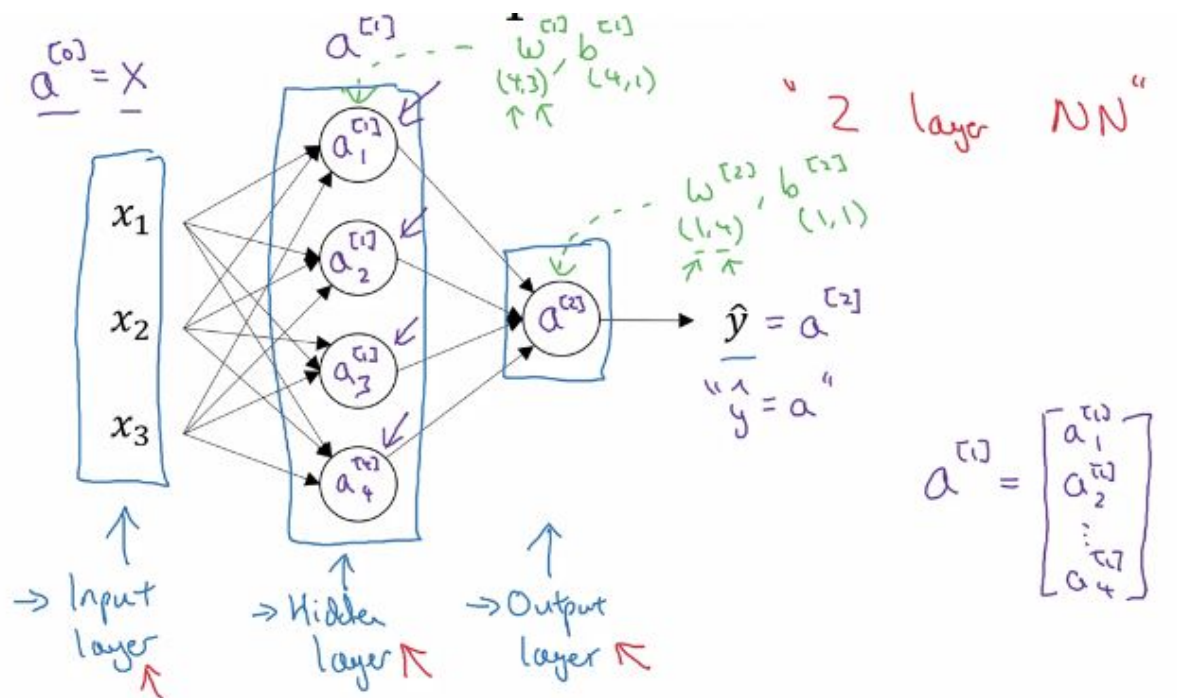
} Don't use

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 ✓
```

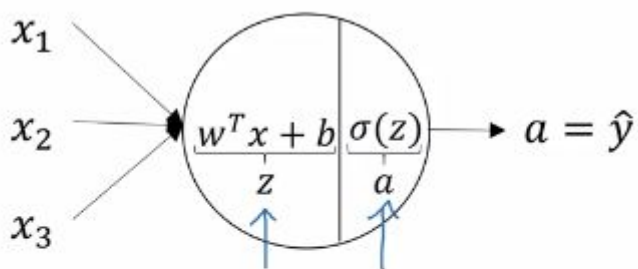
Representacion de Redes Neuronales

Una red neuronal con una capa oculta.



Como se computan las salidas?

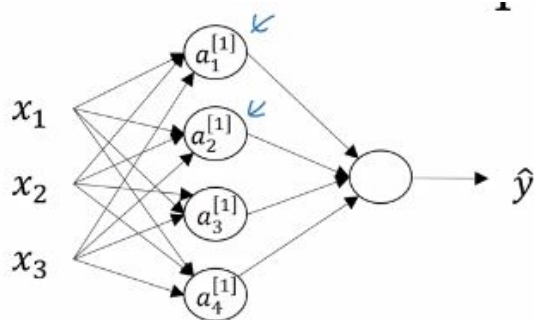
Un nodo en una red neuronal consiste en dos pasos de una regresion logistica.



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

$$a = \sigma(z)$$

Una red neuronal hace esto mismo pero mas veces. Esto es



$$z_1^{[1]} = \underline{w_1^{[1]T}} x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, a_4^{[1]} = \sigma(z_4^{[1]})$$

donde la notacion viene dada por

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

Veamos como hacer para vectorizar estas ecuaciones

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

Given input x :

$$\rightarrow z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} a^{[0]}_{(3,1)} + b^{[1]}_{(4,1)}$$

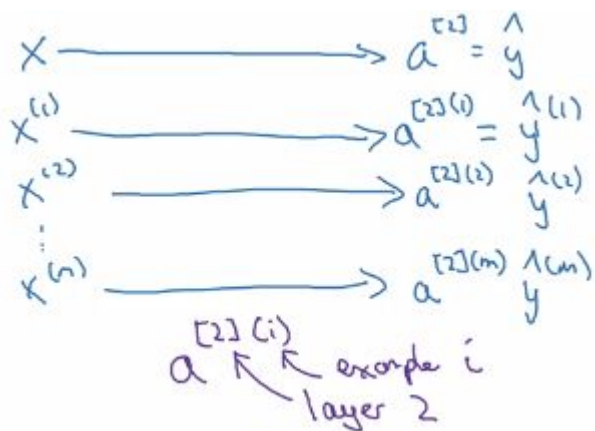
$$\rightarrow a^{[1]}_{(4,1)} = \sigma(z^{[1]}_{(4,1)})$$

$$\rightarrow z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$$

$$\rightarrow a^{[2]}_{(1,1)} = \sigma(z^{[2]}_{(1,1)})$$

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

Introduciendo un poco de notación



Loopeando 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

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

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

← training samples

↑ hidden units.

La vectorizacion resulta (Forward propagation)

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

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

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

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

Explicacion de la implementacion vectorizada

$$z^{1} = w^{[1]} x^{(1)} + b^{[1]}, \quad z^{[1](2)} = w^{[1]} x^{(2)} + b^{[1]}, \quad z^{[1](3)} = w^{[1]} x^{(3)} + b^{[1]}$$

$$W^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \end{bmatrix} \quad W^{[1]} x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} \quad W^{[1]} x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} \quad W^{[1]} x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$W^{[1]} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} | & | & | \\ z^{1} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = Z^{[1]}$$

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

La funcion sigmoidea no es la mejor eleccion en redes neuronales. Estudiemos otras funciones de activacion mas utiles.

Funciones de activacion

Reemplacemos la funcion sigmoidea por una funcion de activacion general $g()$.

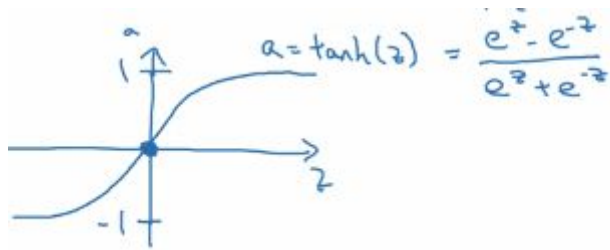
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

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

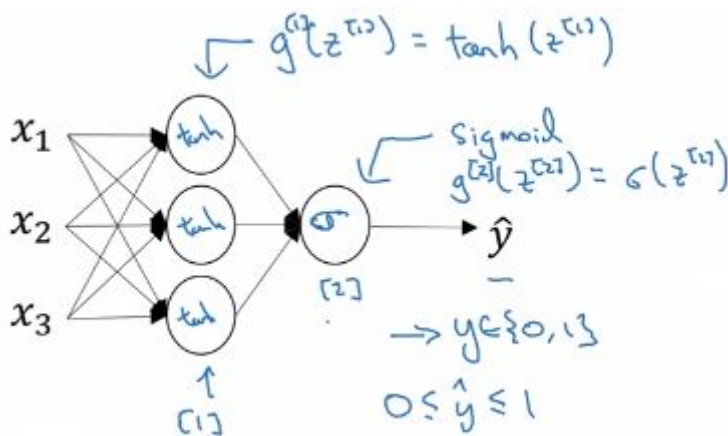
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

En general la tangente hiperbolica funcionara mejor.

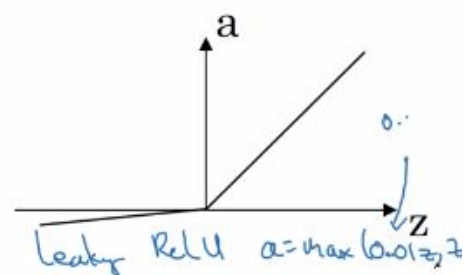
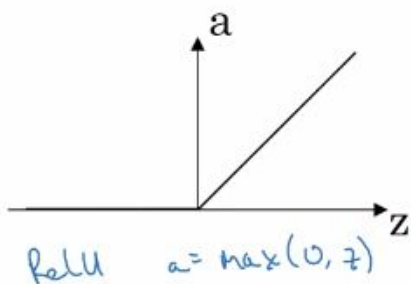
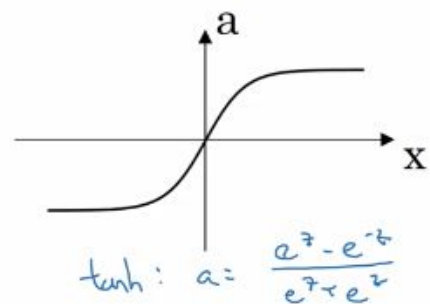
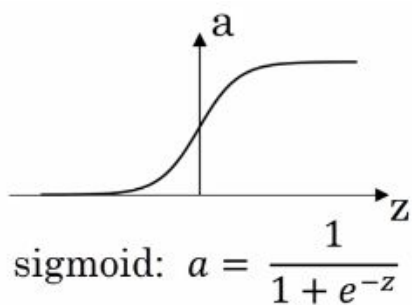


Pero se pueden utilizar distintas funciones de activacion en distintas capas. Es conveniente utilizar en la salida la funcion sigmoidea para clasificacion binaria, ya que entrega valores entre 0 y 1. Pero en las capas ocultas conviene utilizar la tangente hiperbolica.



Ventajas y desventajas de las funciones de activacion

En resumen,



Derivadas de las funciones de activacion

Para el backpropagation necesitamos conocer las derivadas de las funciones de activacion.

- Funcion de activacion sigmoidea

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

- Funcion de activacion tangente hiperbolica

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

- RELU

$$\begin{aligned}g(z) &= \max(0, z) \\ g'(z) &= \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \\ &\quad \text{--- ~~undefined if } z = 0~~ \end{aligned}$$

- Leaky RELU

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

Gradiente descendente para redes neuronales

Consideremos primero una capa oculta nomas.

$$\text{Parameters: } \begin{matrix} W^{[1]} & b^{[1]} & W^{[2]} & b^{[2]} \\ (n^{[1]}, n^{[2]}) & (n^{[2]}, 1) & (n^{[2]}, n^{[3]}) & (n^{[3]}, 1) \end{matrix}$$

$$n_x = n^{[0]}, \quad n^{[1]}, \quad \underline{n^{[2]} = 1}$$

Previamente hemos visto como calcular las predicciones y hat, ahora las derivadas de la funcion de coste.

$$\text{Cost function: } J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}_i, y_i) \\ \uparrow a^{[2]}$$

Gradient descent:

$$\begin{aligned} \rightarrow \text{Repeat } \{ \\ & \text{Compute preds } (\hat{y}^{(i)}, i=1, \dots, m) \\ & \frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial W^{[1]}}, \quad \frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial b^{[1]}}, \dots \\ & W^{[1]} := W^{[1]} - \alpha \frac{\partial J}{\partial W^{[1]}} \\ & b^{[1]} := b^{[1]} - \alpha \frac{\partial J}{\partial b^{[1]}} \\ \} \end{aligned}$$

Formulas para calcular derivadas (Backward propagation)

Back propagation:

$$dz^{[2]} = A^{[2]} - Y \leftarrow$$

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

$$dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$$

$$(n^{[2]}, 1) \leftarrow$$

$$(n^{[2]}, 1) \leftarrow$$

$$db^{[2]} = \frac{1}{m} \text{np.sum}(dz^{[2]}, \text{axis}=1, \text{keepdims}=\text{True})$$

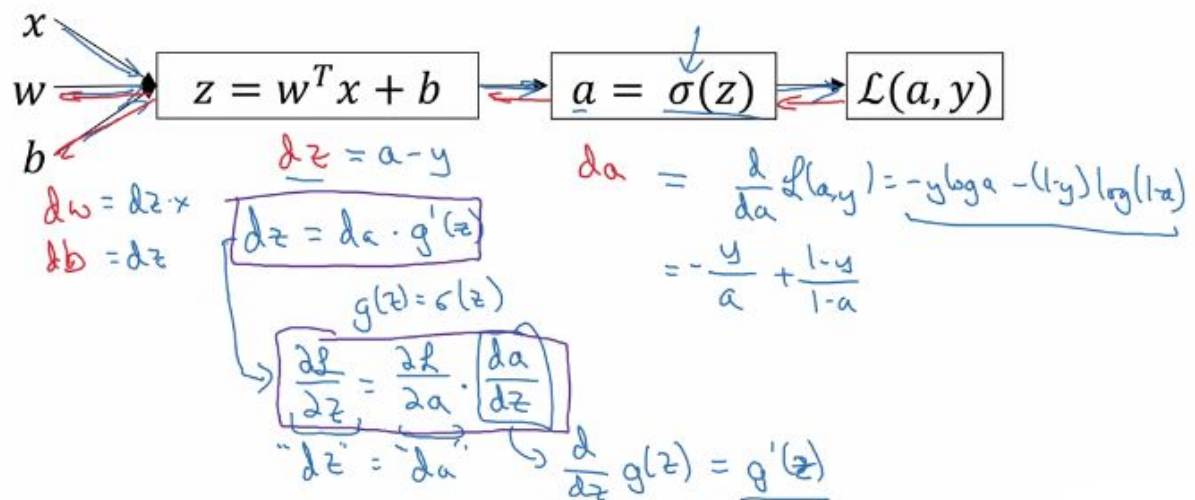
$$dz^{[1]} = \underbrace{W^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} \times \underbrace{g^{[2]'}(z^{[1]})}_{\text{element-wise product}} \quad (n^{[1]}, m)$$

$$dW^{[1]} = \frac{1}{m} dz^{[1]} X^T$$

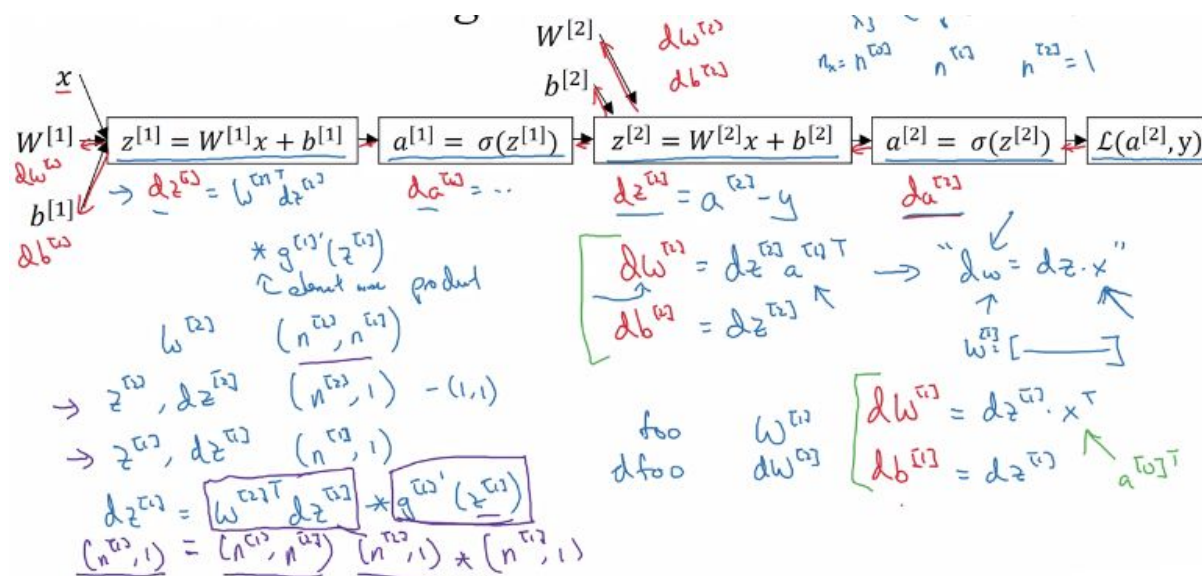
$$db^{[1]} = \frac{1}{m} \text{np.sum}(dz^{[1]}, \text{axis}=1, \text{keepdims}=\text{True})$$

Intuición de la backward propagación

- Regresión logística



- Gradiente en red neuronal



De manera mas prolija, las 6 ecuaciones de interes para el backpropagation

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

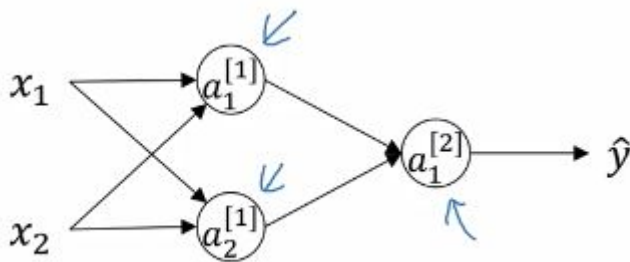
Inicializaacion parametros

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

Inicializacion aleatoria

El factor 0.01 en w es para que se inicialicen con valores pequeños. Si tomasen valores grandes la funcion de activacion saturaria.

Por otra parte a los parametros b no les afecta inicializarlos con valores nulos.



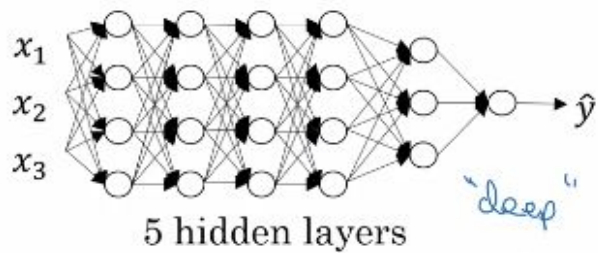
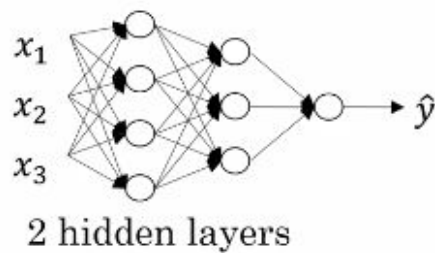
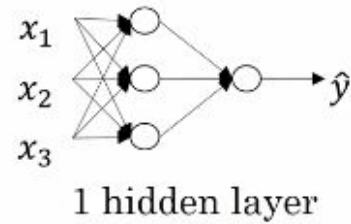
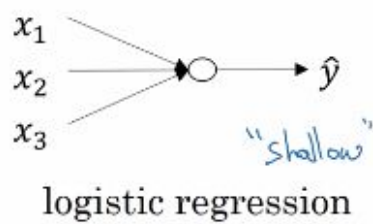
$$\begin{aligned} \rightarrow w^{[1]} &= \text{np.random.randn}(2,2) * \frac{0.01}{100?} \\ b^{[1]} &= \text{np.zeros}(2,1) \\ w^{[2]} &= \dots \\ b^{[2]} &= 0 \end{aligned}$$



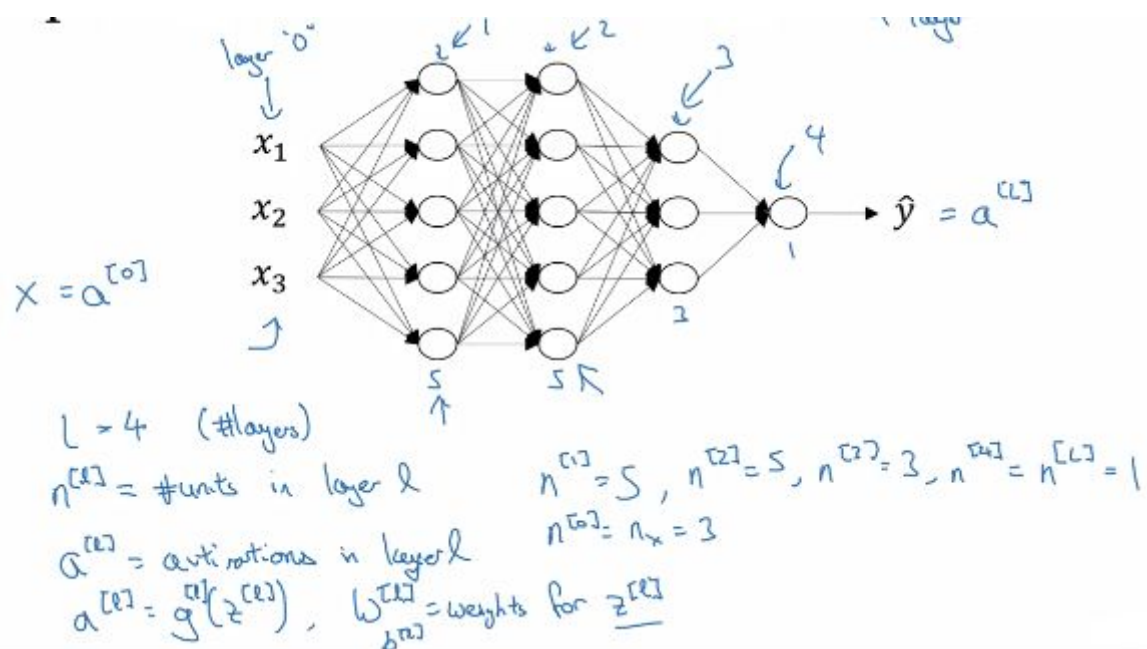
Cuando se entrene algun red neuronal mas profunda seria mas razonable usar otro valor que no fuera 0.01, pero se vera mas adelante.

Redes neuronales profundas de L-capas

Ejemplos de redes neuronales.



Veamos la notacion a utilizar



Forward and Backward propagation in deep neural networks

- Forward propagation

Input $a^{[l-1]} \leftarrow$

Output $a^{[l]}$, cache $(z^{[l]})$

$$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

Vectorized:

$$z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(z^{[l]})$$

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

- Background propagation

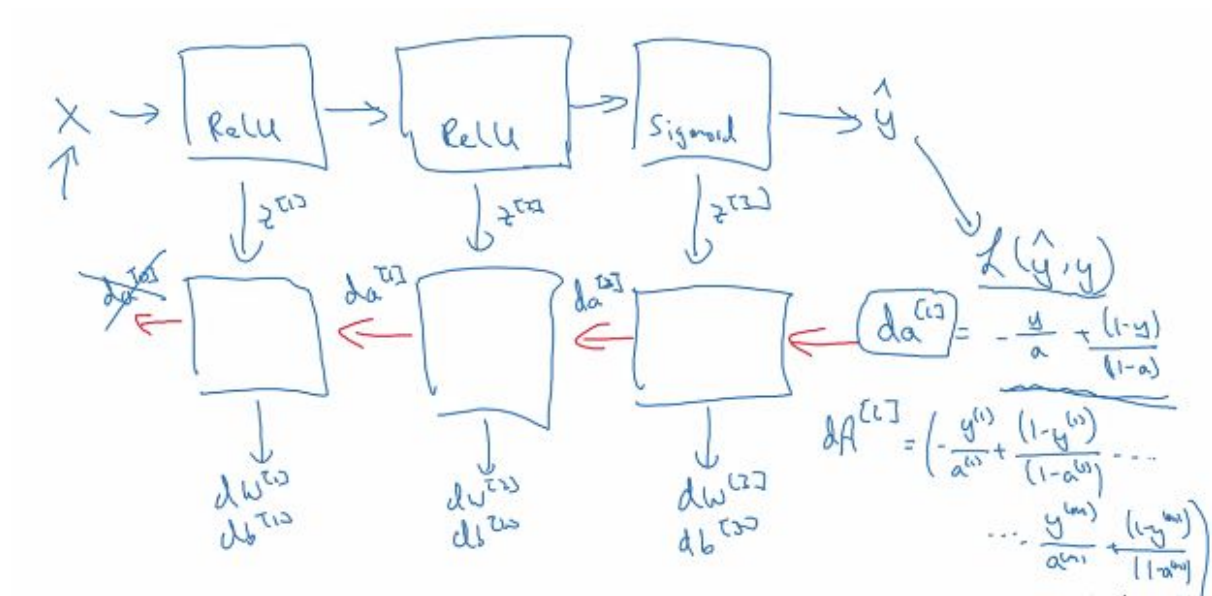
Input $da^{[l]}$

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

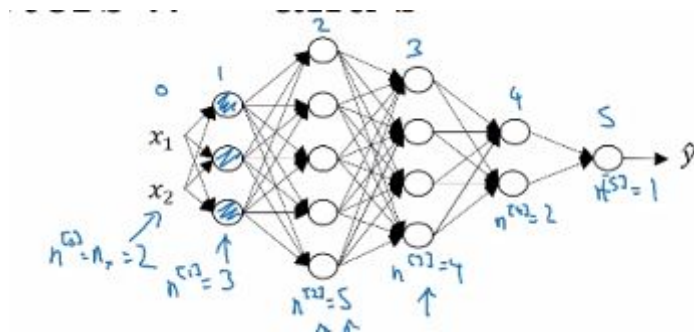
$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$
$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$
$$dz^{[l-1]} = W^{[l+1]T} \cdot dz^{[l]} * g^{[l+1]'}(z^{[l-1]})$$

$$dz^{[l]} = dA^{[l]} * g^{[l]'}(z^{[l]})$$
$$dW^{[l]} = \frac{1}{n} dz^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{n} \text{np.sum}(dz^{[l]}, \text{axis}=1, \text{keepdims}=True)$$
$$dA^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

Resumen



Metodo de debugueo: dimensiones de las matrices



Los parametros w para un ejemplo:

$$W^{(1)}: (n^{(2)}, n^{(1)})$$

$$W^{(2)}: (5, 3) \quad (n^{(3)}, n^{(2)})$$

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 dimension que W y b respectivamente.

$$dW^{(L)}: (n^{(L)}, n^{(L-1)})$$

$$db^{(L)}: (n^{(L)}, 1)$$

Otras cantidades de interes: $z = g(a)$, y a tienen la misma dimension,

Considerando los m ejemplos a entrenar

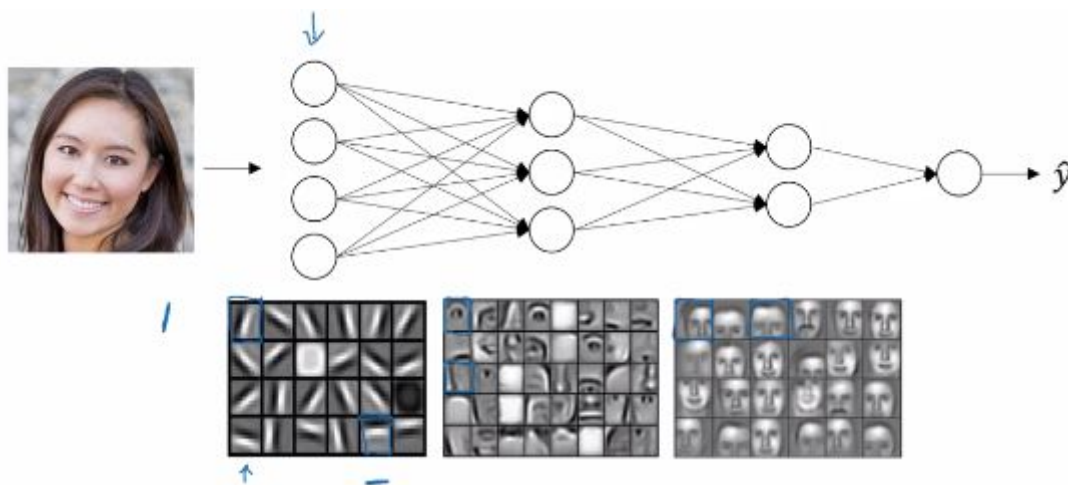
$$\vec{z}^{(1)} = W^{(1)} \cdot X + b^{(1)}$$

$(n^{(1)}, m)$ $(n^{(1)}, n)$ $(n^{(1)}, m)$ $(n^{(1)}, 1)$
 $(n^{(1)}, m)$

En python b podria seguir siendo de la misma dimension, 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 identificacion de una cara. Se podria 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 podria detectar los cambios de sonido, luego unidades basicas de sonido, como fonemas, luego poder reconocer palabras, para luego reconocer oraciones o frases.



¿Qué son los hiperparámetros?

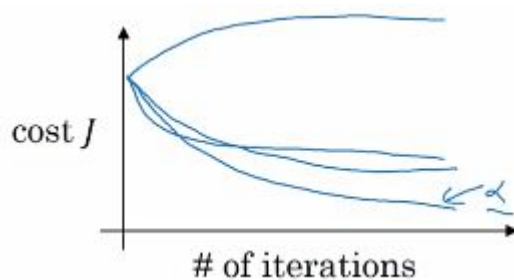
Dados los parámetros de una red neuronal,

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

los hiperparámetros son aquellos que controlan y determinar el valor de los parámetros.
Ellos son:

- Learning rate α
- num de iteraciones
- numero de capas ocultas L
- numero de unidades ocultas $n^{[1]}, n^{[2]}, \dots$
- eleccion de funciones de activacion
- etc

Deep Learning is un proceso empirico. El valor de los hiperparametros se calcula mediante la experimentacion.

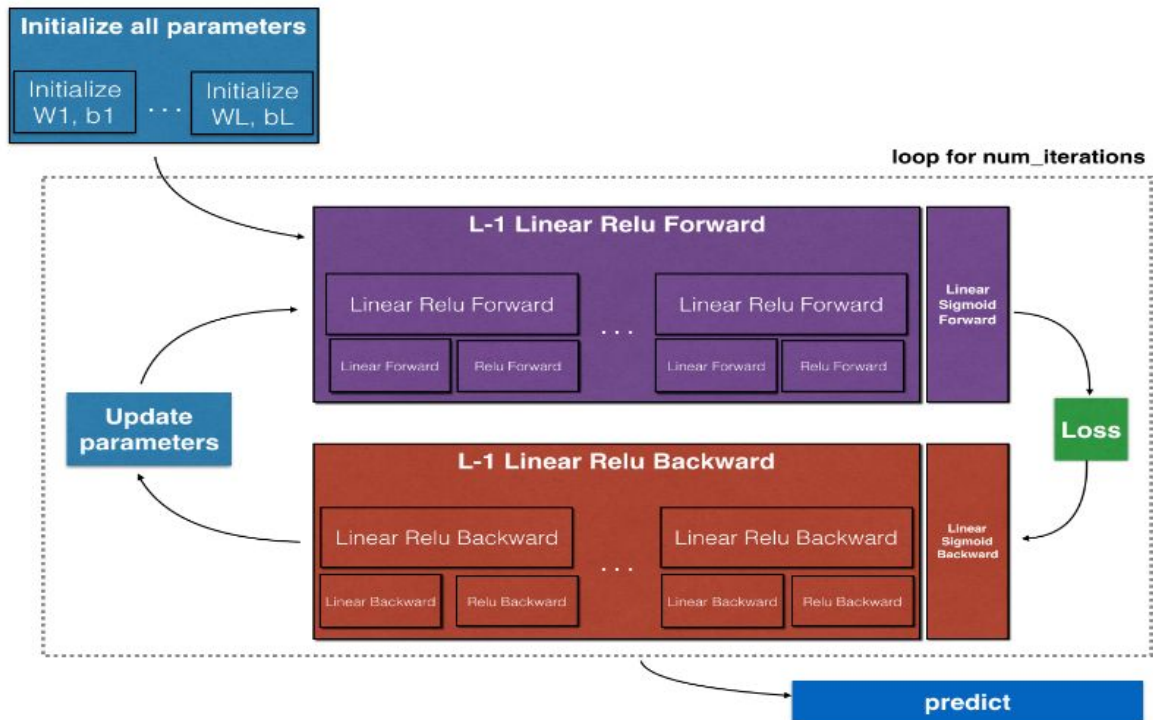


Resumen

Forward y Backward propagation

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

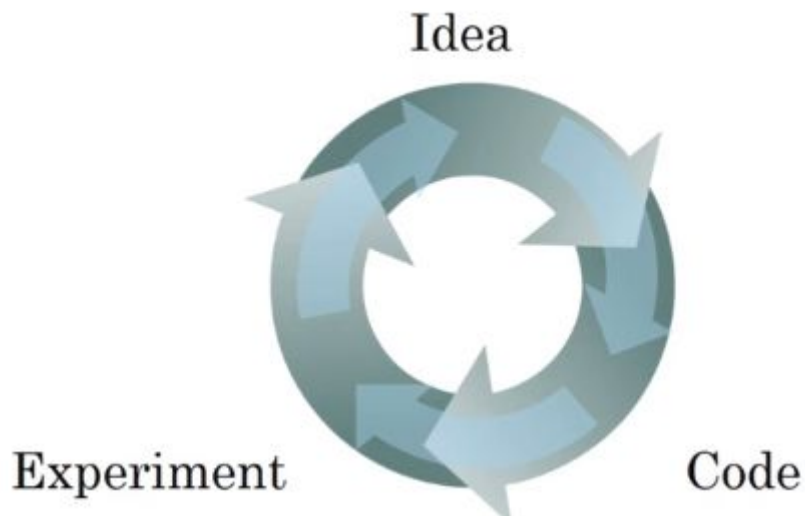
$$\begin{aligned} dZ^{[L]} &= A^{[L]} - Y \\ dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]T} \\ db^{[L]} &= \frac{1}{m} np.sum(dZ^{[L]}, axis = 1, keepdims = True) \\ dZ^{[L-1]} &= dW^{[L]T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\ &\vdots \\ dZ^{[1]} &= dW^{[L]T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\ dW^{[1]} &= \frac{1}{m}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



Course number 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 separen 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.

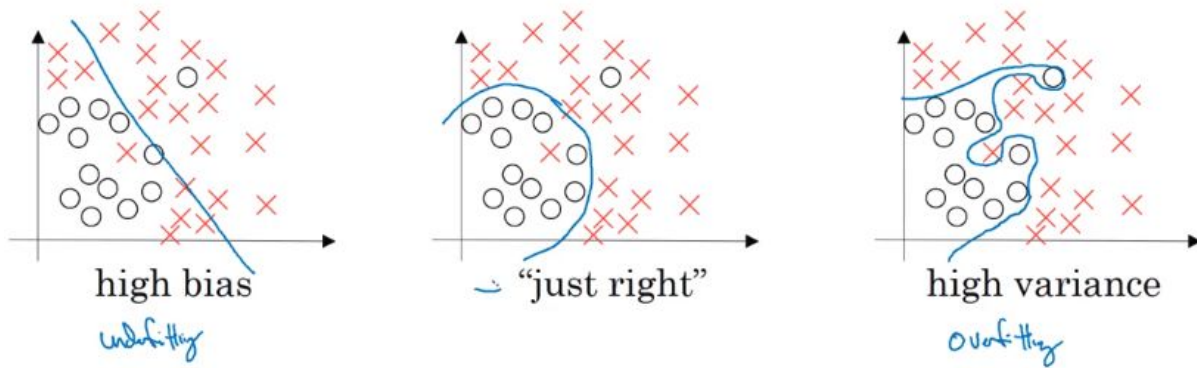
Mismatched train/test distribution

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



Bias/Variance

Ejemplo donde se tienen dos features.



En el caso de haber mas de una hay diferentes metricas que se pueden utilizar.

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

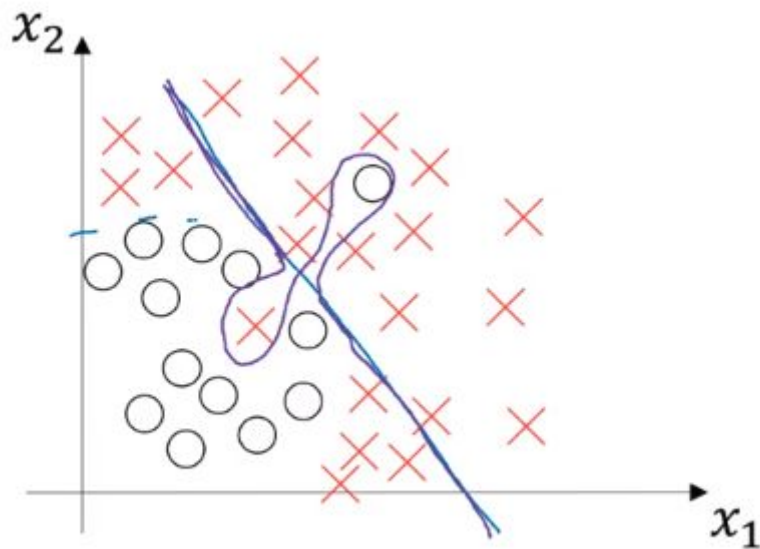
Cat classification



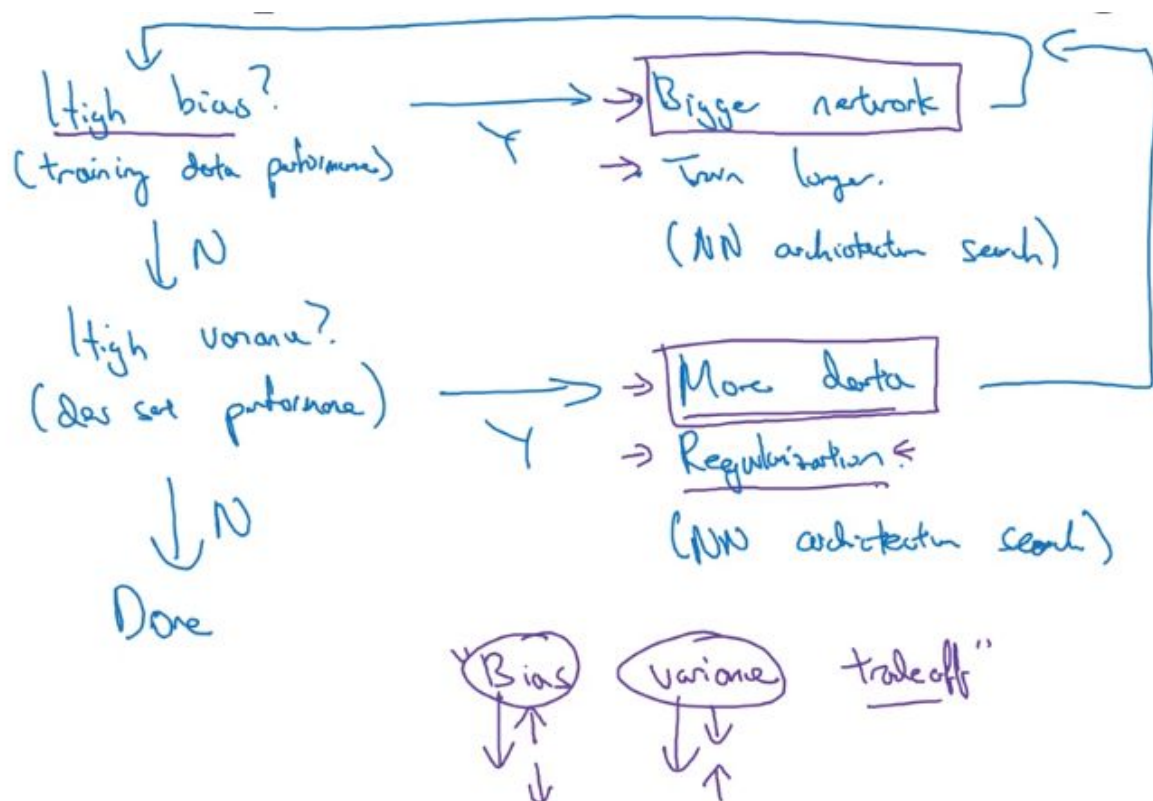
Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias low variance
Human:	0%			
Optimal (Bayes) error:	0%			

Estas medidas suponen tanto que, como se dijo antes, las distribuciones de los datos provienen de la misma distribucion de probabilidad, como que las imagenes no son borrosas para que se considere el error humano como 0%. La falla del primer caso se vera mas adelante. En el caso de tener imagenes borrosas por jemplo, 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 mas unidades ocultas, entrenarlo mas 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 podria probar si hay alta varianza sobre el dev set, probando con mas datos o aplicando regularizacion y tambien, como antes, probar una nueva arquitectura para la red neuronal.

Cuando ambas son negativas se podria afirmar que se termino.

Regularizacion de una red neuronal

Regularizacion

help prevent overfitting and errors of the neural network.

Aplicado a Logistic Regression

$$\begin{aligned} & \min_{w,b} J(w,b) \quad \underline{w \in \mathbb{R}^{n_x}}, b \in \mathbb{R} \\ J(w,b) &= \frac{1}{m} \sum_{i=1}^m \ell(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2 + \underbrace{\frac{\lambda}{2m} b^2}_{\text{omit}} \\ \text{L}_2 \text{ regularization} \quad \underline{\|w\|_2^2} &= \sum_{j=1}^{n_x} w_j^2 = w^T w \end{aligned}$$

Esto se llama regularizacion L_2, donde lambda es el parametro de regularizacion. El ultimo termino se omite dado que w suele poseer muchos mas parametros que b ($n_x \gg 1$).

A su vez, existe lo que se llama regularizacion L_1, pero no es utilizado tan comunmente.

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

Que forma toma la regularizacion para una red neuronal?

$$\begin{aligned} J(w^{[0]}, b^{[0]}, \dots, w^{[L]}, b^{[L]}) &= \frac{1}{m} \sum_{i=1}^m \ell(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|w^{[l]}\|_F^2 \\ \|w^{[l]}\|_F^2 &= \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2 \quad w: \begin{pmatrix} n^{[l-1]} & n^{[l]} \\ \uparrow & \uparrow \end{pmatrix} \\ \text{"Frobenius norm"} & \quad \| \cdot \|_2^2 \quad \| \cdot \|_F^2 \end{aligned}$$

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

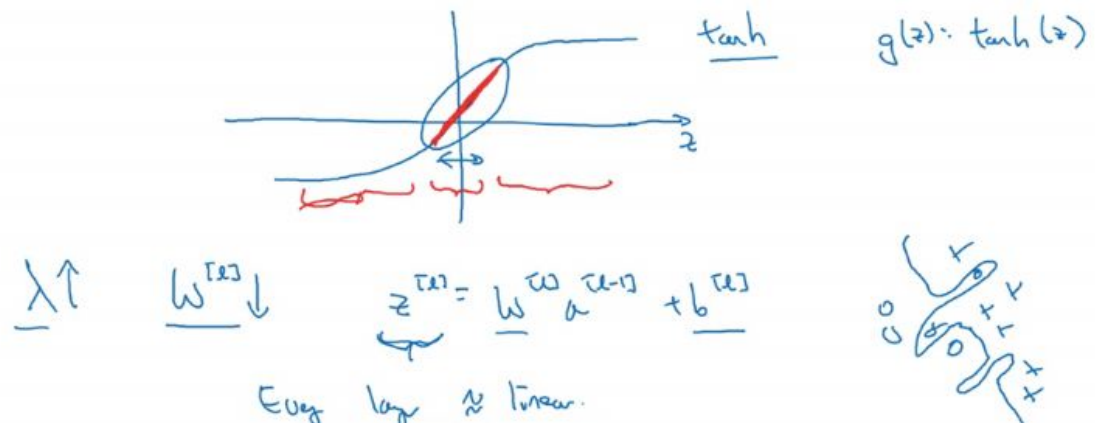
Por otra parte la backpropagation es de la forma

$$dW^{[L]} = \underbrace{(\text{from backprop}) + \frac{\lambda}{n} W^{[L]}}_{\rightarrow W^{[L]} := W^{[L]} - 2dW^{[L]}}$$

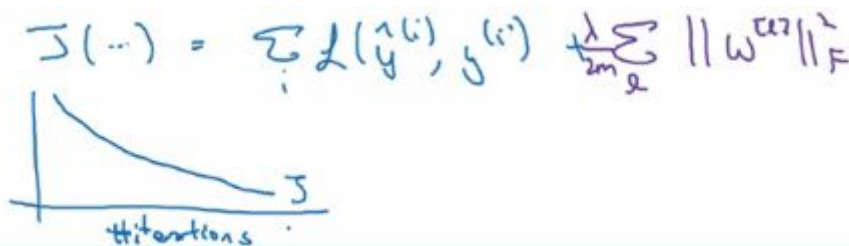
$$\frac{\partial J}{\partial W^{[L]}} = dW^{[L]}$$

¿Por que la regularizacion reduce overfitting?

Tomando como ejemplo la funcion de activacion $\tanh(z)$, para un valor grande de λ , w es pequeño y por consiguiente z tambien. De aqui que \tanh se valore practicamente en la region lineal y se prevenga el overfitting. La red neuronal sera practicamente una red lineal.

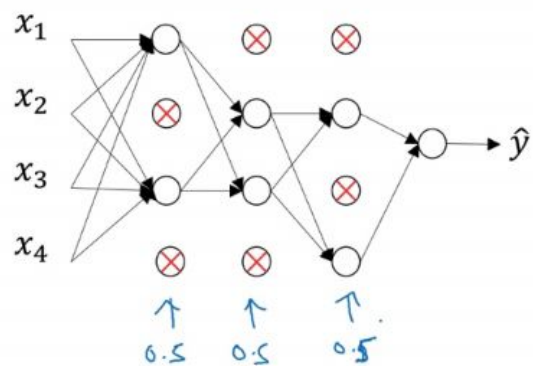
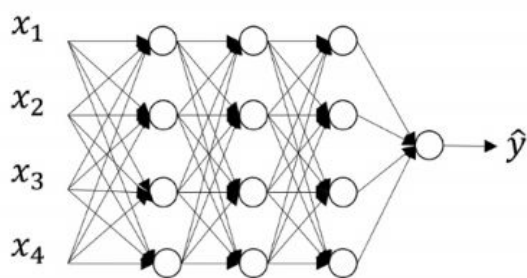


Por otra parte para probar el correcto funcionamiento de la implementacion se puede probar graficando la funcion de coste en funcion del numero de iteraciones y ver que esta es una funcion monotonicamente decreciente.



Dropout regularization

nueva tecnica de regularizacion. La misma consiste en eliminar nodos con una dada probabilidad. El nuevo calculo se realiza sin los nodos eliminados y se previene la regularizacion.



Pero, como se implementa esta regulariacion? Una tecnica se conoce como inverted dropout

Illustrate with layer $l=3$. keep-prob = $\frac{0.8}{x}$ 0.2

$\Rightarrow d3 = \text{np.random.rand}(a3.\text{shape}[0], a3.\text{shape}[1]) < \text{keep-prob}$

$a3 = \text{np.multiply}(a3, d3)$ # $a3 \neq d3$.

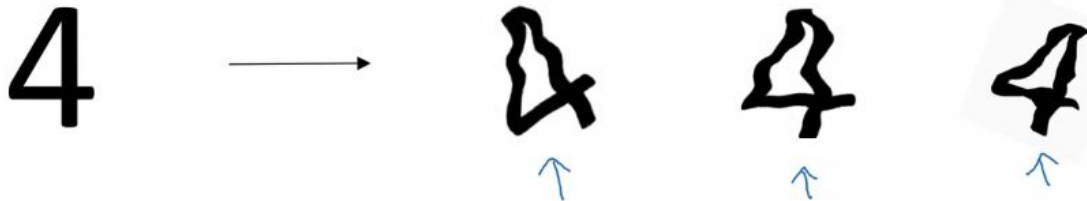
$\Rightarrow a3 /= \frac{0.8}{\text{keep-prob}}$ 50 units. \leadsto 10 units shut off

$$z^{[4]} = w^{[4]} \cdot \underbrace{a^{[3]}}_{\substack{\uparrow \\ \text{reduced by } 20\% \\ \downarrow \\ 1 = 0.8}} + b^{[4]}$$
 Test

Otros metodos de regularizacion

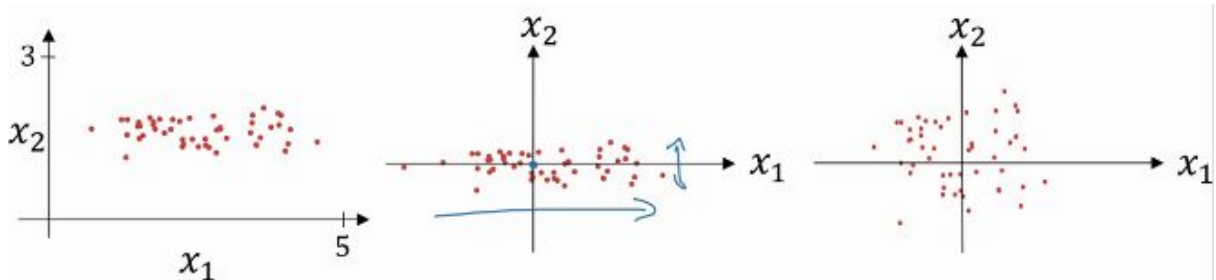
Data augmentation

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

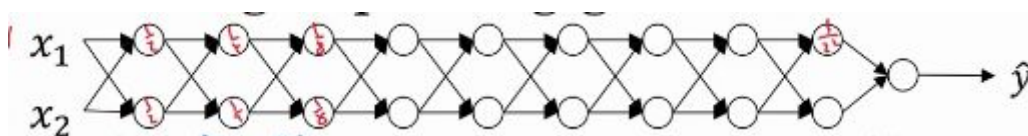


Configurando la optimización de problemas

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



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



$g(z) = z$ $b^{(0)} = 0$
 $\hat{y} = W^{(L)} \cdot \dots \cdot W^{(2)} \cdot W^{(1)} \cdot x$
 $W^{(0)} > I$ $W^{(0)} < I$ $\begin{bmatrix} 0.9 & 0.9 \end{bmatrix}$
 $W^{(1)} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$
 $z^{(1)} = W^{(1)} x$
 $a^{(1)} = g(z^{(1)}) = z^{(1)}$
 $a^{(2)} = g(z^{(2)}) = g(W^{(2)} a^{(1)})$
 $\hat{y} = W^{(L)} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L-1} x$
 1.5^L 0.5^L
 $1.5^{L-1} x$ $0.5^{L-1} x$

Suponiendo que $b=0$ y que la funcion de activacion 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 unica neurona, luego se vera el caso de una red neuronal mas compleja.

Dado que z es la suma de terminos de la forma $w \cdot x$, si n aumenta se quiere que w disminuya. De que manera? Una buena opcion 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 expresion para w que no es muy grande o muy pequeña si los features aumentan considerablemente.

$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n$
 Large $n \rightarrow$ Smaller w_i
 $\text{Var}(w_i) = \frac{2}{n}$
 $w^{\text{RELU}} = \text{np.random.randn}(\text{shape}) * \text{np.sqrt}\left(\frac{2}{n^{(L-1)}}\right)$
 $g^{(L)}(z) = \text{ReLU}(z)$

Otras variantes para distintas funciones de activacion como tanh: Inicializacion de Xavier.

Other variants:

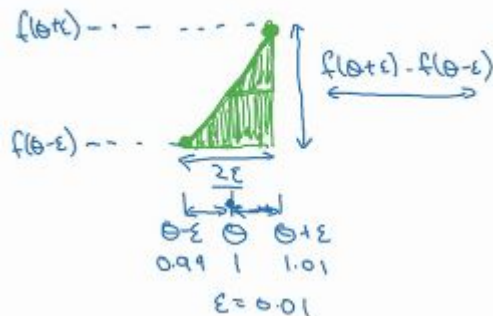
tanh

Xavier initialization

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

Aproximacion numerica de gradientes

Calculando derivadas a ambos lados.



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

$$f'(\theta) = \lim_{\epsilon \rightarrow 0} \frac{f(\theta + \epsilon) - f(\theta - \epsilon)}{2\epsilon} \quad \begin{array}{l} O(\epsilon^3) \\ 0.01 \\ 0.0001 \end{array} \quad \left| \quad \frac{f(\theta + \epsilon) - f(\theta)}{\epsilon} \quad \begin{array}{l} \text{error: } O(\epsilon) \\ 0.01 \end{array}$$

Gradient Checking: tecnica para verificar la implementacion 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 $\boxed{W^{[1]}}, \boxed{b^{[1]}}, \dots, \boxed{W^{[L]}}, \boxed{b^{[L]}}$ and reshape into a big vector $\underline{\theta}$.
concentrate
 $J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = J(\theta)$

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

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

for each i :

$$\rightarrow \underline{d\theta_{approx}}[i] = \frac{J(\theta_1, \theta_2, \dots, \theta_i^{\downarrow} + \epsilon, \dots) - J(\theta_1, \theta_2, \dots, \theta_i^{\downarrow} - \epsilon, \dots)}{2\epsilon}$$

$$\approx \underline{d\theta}[i] = \frac{\partial J}{\partial \theta_i} \quad \left| \quad d\theta_{approx} \stackrel{?}{\approx} d\theta$$

Como hacemos para chequear que estos dos vectores sean los mismos? Se puede calcular la distancia euclidea entre ellos.

Handwritten notes showing the formula for checking vector similarity and a scale reference:

$$\text{Check } \frac{\|d\Theta_{\text{approx}} - d\Theta\|_2}{\|d\Theta_{\text{approx}}\|_2 + \|d\Theta\|_2} \approx \begin{matrix} 10^{-7} & - \text{great!} \\ 10^{-5} & \\ 10^{-3} & - \text{worry.} \end{matrix}$$

Below the formula, it is noted: $\epsilon = 10^{-7}$

Si epsilon es del orden de 10^{-7} y la distancia euclidea la implementacion probablemente es la correcta. Pero si la distancia es mas grande en 2 ordenes de magnitud por lo menos, entonces conviene mirar si se cometio algun error (idealmente en las componentes i individuales, ej si el error esta en dw o en db).

Notas en gradient checking

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

Algoritmos de optimizacion

Dado que Deep Learning funciona mejor con Big Data, esto enlantece el computo de problemas. En esta seccion se estudiaran diferentes tecnicas para acelerar este proceso mediante algoritmos de optimizacion.

Mini-batch gradiente descendiente

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

$$X = \begin{bmatrix} x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(m)} \end{bmatrix}$$

(n_x, m)

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(m)} \end{bmatrix}$$

$(1, m)$

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

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

$$X = \begin{bmatrix} x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(1000)} & | & x^{(1001)} & \dots & x^{(2000)} & | & \dots & | & \dots & x^{(m)} \end{bmatrix}$$

(n_x, m) $X^{\{1\}} (n_x, 1000)$ $X^{\{2\}} (n_x, 1000)$ $X^{\{5,000\}} (n_x, 1000)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(1000)} & | & y^{(1001)} & \dots & y^{(2000)} & | & \dots & | & \dots & y^{(m)} \end{bmatrix}$$

$(1, m)$ $Y^{\{1\}} (1, 1000)$ $Y^{\{2\}} (1, 1000)$ $Y^{\{5,000\}} (1, 1000)$

What if $m = 5,000,000$?
 5,000 mini-batches of 1,000 each
 Mini-batch t : $\underline{X^{t+1}, Y^{t+1}}$

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

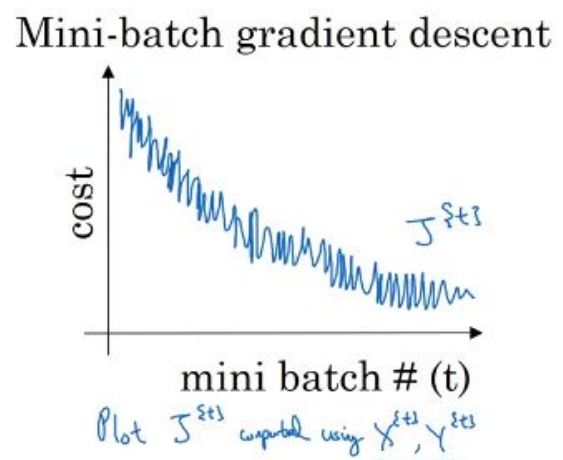
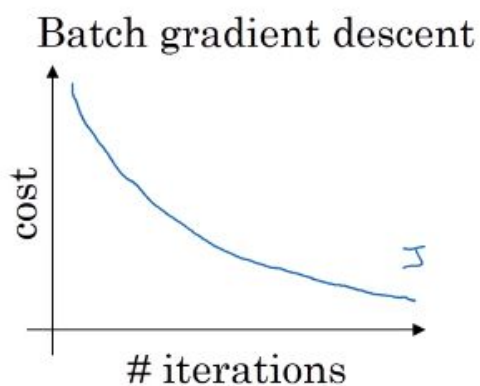
```

for  $t = 1, \dots, 5000$  {
  Forward prop on  $X^{t+1}$ :
   $Z^{(1)} = W^{(1)} X^{t+1} + b^{(1)}$ 
   $A^{(1)} = g^{(1)}(Z^{(1)})$ 
  ...
   $A^{(L)} = g^{(L)}(Z^{(L)})$ 
  Compute cost  $J^{t+1} = \frac{1}{1000} \sum_{i=1}^n \ell(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum \|W^{(l)}\|_F^2$ 
  Backprop to compute gradients w.r.t  $J^{t+1}$  (using  $X^{t+1}, Y^{t+1}$ )
   $W^{(1)} := W^{(1)} - \alpha dW^{(1)}, b^{(1)} := b^{(1)} - \alpha db^{(1)}$ 
}
  
```

} Vectorial implementation (1000 examples)
 ↘ for X^{t+1}, Y^{t+1}

A el loop exterior habria que agregarle otro loop del numero de iteraciones.

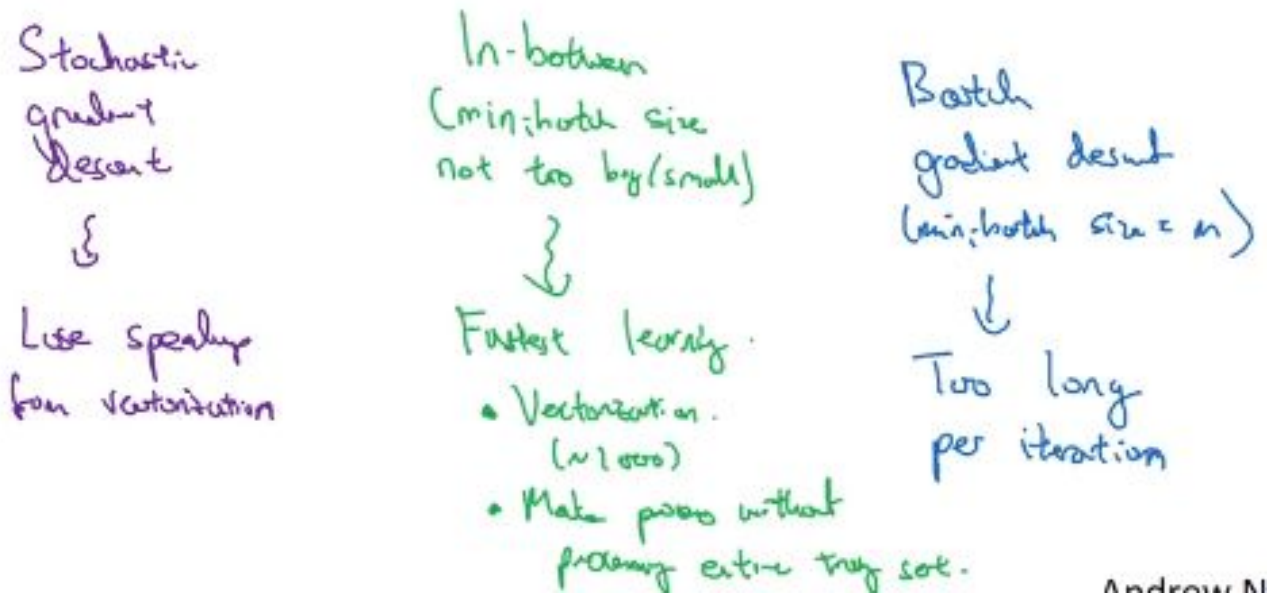
Funcion de coste en funcion del numero de iteraciones



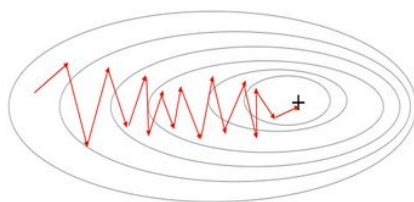
Uno podria preguntarse qué tamaño del mini-batch elegir y cómo.

- Si tamaño mini-batch = m : batch es el de gradient descent.

- si tamaño mini-batch = 1: gradiente descendente estocastico. Cada ejemplo es un mini-batch.
- En la practica el tamaño del mini-batch va a estar entre 1 y m .



Stochastic Gradient Descent



Gradient Descent

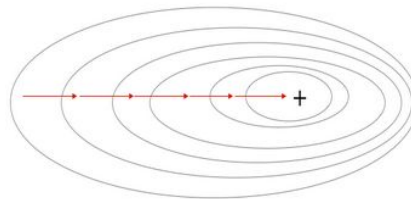
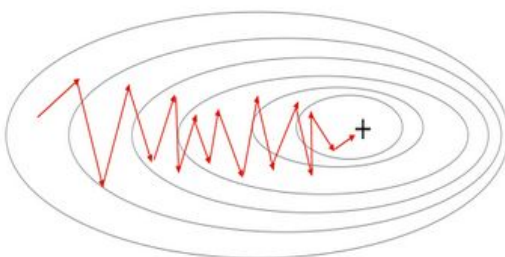


Figure 1: SGD vs GD

"+" 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

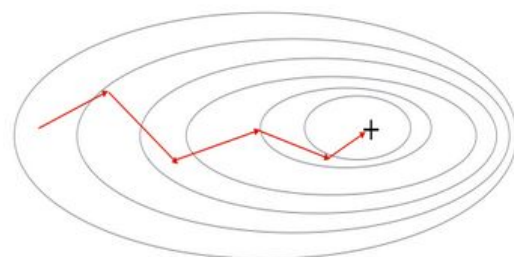


Figure 2: SGD vs Mini-Batch GD

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

Que tamaño se elige entonces?

En practica este es otro hiperparametro 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:

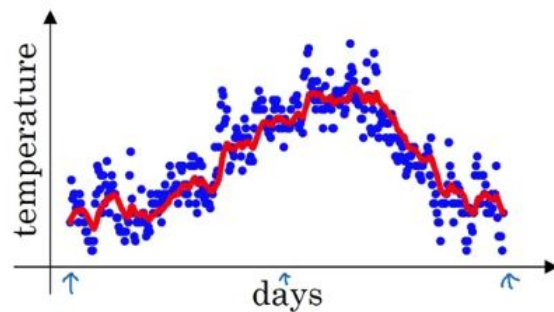
$64(2^{**6}), 128(2^{**7}), 256(2^{**8}), 512(2^{**9})$. Conviene chequear que el tamaño del mini-batch puede almacenarse en la memoria del CPU/GPU.

Exponentially weighted averages

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

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

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



$$\begin{aligned}V_0 &= 0 \\ V_1 &= 0.9 V_0 + 0.1 \theta_1 \\ V_2 &= 0.9 V_1 + 0.1 \theta_2 \\ V_3 &= 0.9 V_2 + 0.1 \theta_3 \\ &\vdots \\ V_t &= 0.9 V_{t-1} + 0.1 \theta_t\end{aligned}$$

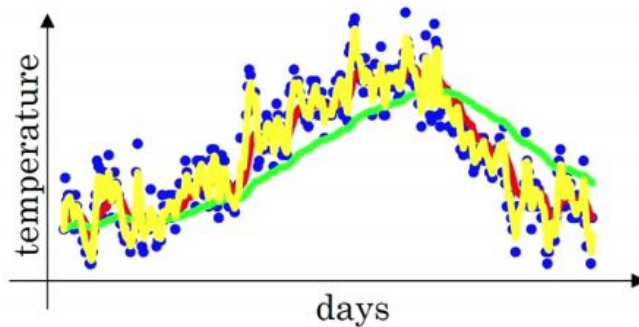
El promedio ponderado se calcula para el primer dia $V_0 = 0$. Para el segundo, se supone que hay un peso del 90% respecto del valor del dia anterior V_0 mas una proporcion restante del 10% a determinar por θ_1 .

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

$\beta = 0.9$: ≈ 10 days' temper.
 $\beta = 0.98$: ≈ 50 days
 $\beta = 0.5$: ≈ 2 days

V_t is approximately
 average over
 $\rightarrow \approx \frac{1}{1-\beta}$ days' temperature.

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



De manera general la ecuacion que relaciona la temperatura para un dia particular se expresa en la figura superior. Donde se tiene un parametros a determinar son beta y theta. Se puede notar que para beta mas grandes el valor de la temperatura del dia anterior es muy importante y los cambios por nuevos puntos no son tan bruscos.

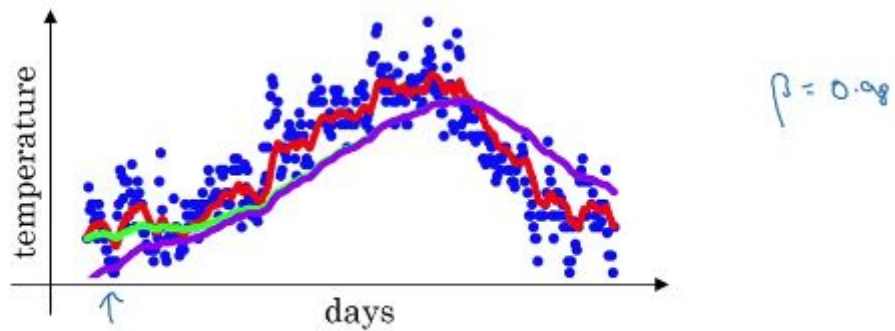
La linea roja equivale a beta = 0.9. La verde a beta = 0.89 y beta = 0.5 a la linea amarilla.

Implementacion computacional

```

→ V0 = 0
Repeat {
  Get next Θt
  Vt := β Vt + (1-β) Θt ←
}
  
```

Correccion 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.

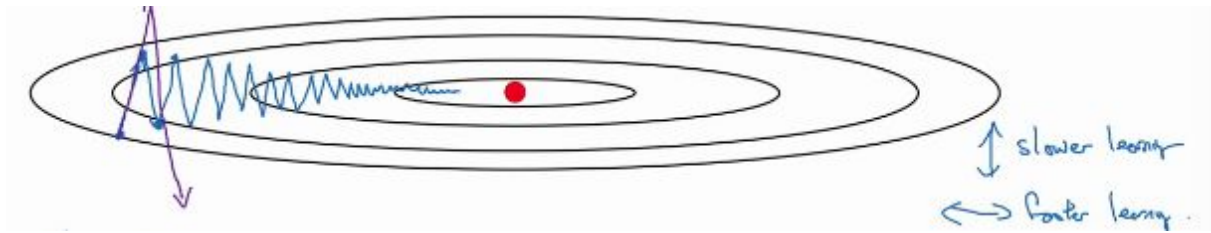
Simplemente consiste en ver que a tiempos pequeños hay un problema debido a que $V_0 = 0$, generalmente no es una buena implementación. Mientras que a tiempos mayores esto se

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 Bias correction 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

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

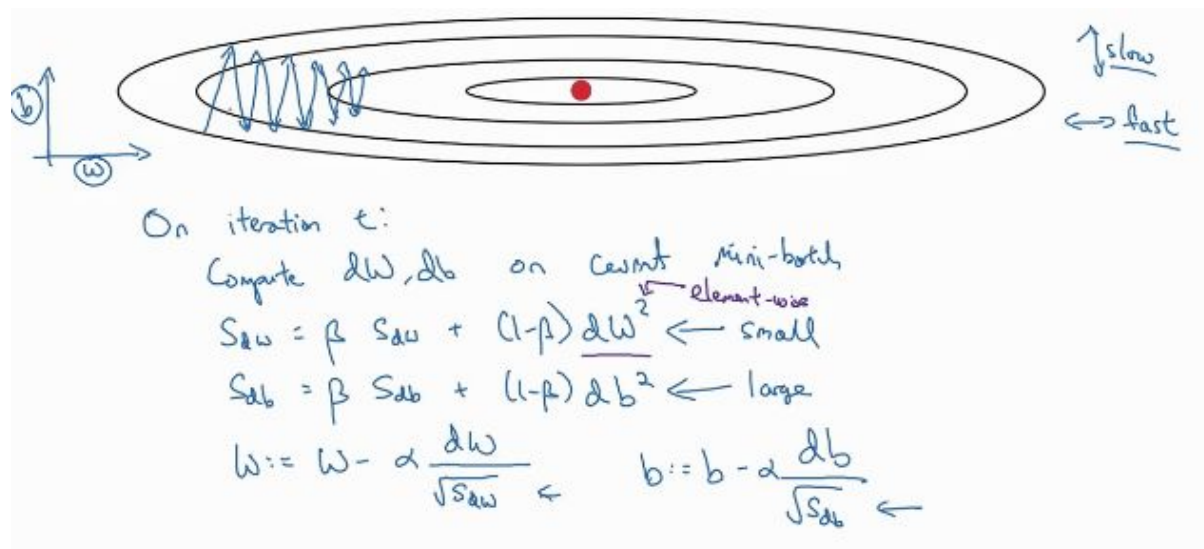
$$v_{db} = \beta v_{db} + (1 - \beta) db$$

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

Hyperparameters: α, β $\beta = 0.9$

Basicamente este algoritmo lo que hace es suavizar el movimiento lateral que es el que se quiere que converja mas rapidamente. El algoritmo toma una camino mas directo.

Otro algoritmo: RMSProp: RootMeanSquareProp



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

algoritmo de optimizacion de Adam

ADAM: Adaptive moment estimation

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

$$V_{dw} = 0, S_{dw} = 0, V_{db} = 0, S_{db} = 0$$

On iteration t :

Compute dw, db using current mini-batch

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

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

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

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

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

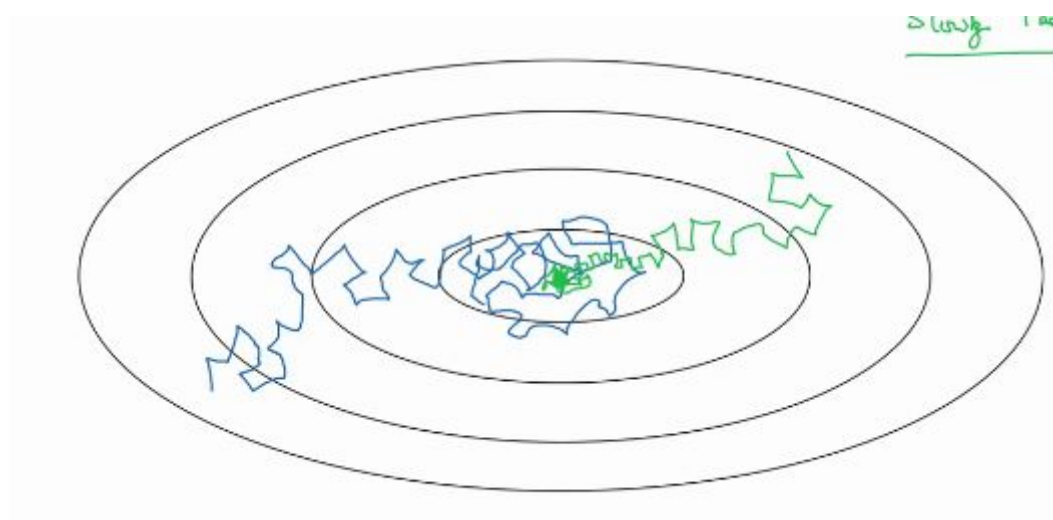
Elección de hiperparámetros: generalmente con la optimización de adam se definen valores para beta y epsilon y se trata de estimar el mejor valor de alpha entre un rango de valores.

α : needs to be tune
 β_1 : 0.9 (dw)
 β_2 : 0.999 (dw²)
 ϵ : 10^{-8}

Decaimiento del ritmo de aprendizaje (alpha)

La idea detras de este metodo es que, dado un valor de alpha fijo el agoritmo podria quedarse dando saltos alrededor del punto de equilibrio no tan cerca de el(linea azul).

El decaimiento propuesto es del tipo exponencial, inicialmente siendo grande dado que se esta lejos de la solucion optima, y a medida que se va acercando alpha va siendo cada vez mas pequeño(linea roja).



Definicion de epoch y decaimiento del ritmo de aprendizaje

Learning rate decay

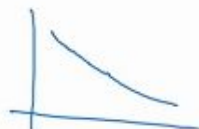
1 epoch = 1 pass through data.

$$\alpha = \frac{1}{1 + \text{decay-rate} \times \text{epoch-num}} \alpha_0$$

epoch	α
1	0.1
2	0.67
3	0.5
4	0.4



$\alpha_0 = 0.2$
decay-rate = 1



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

for example

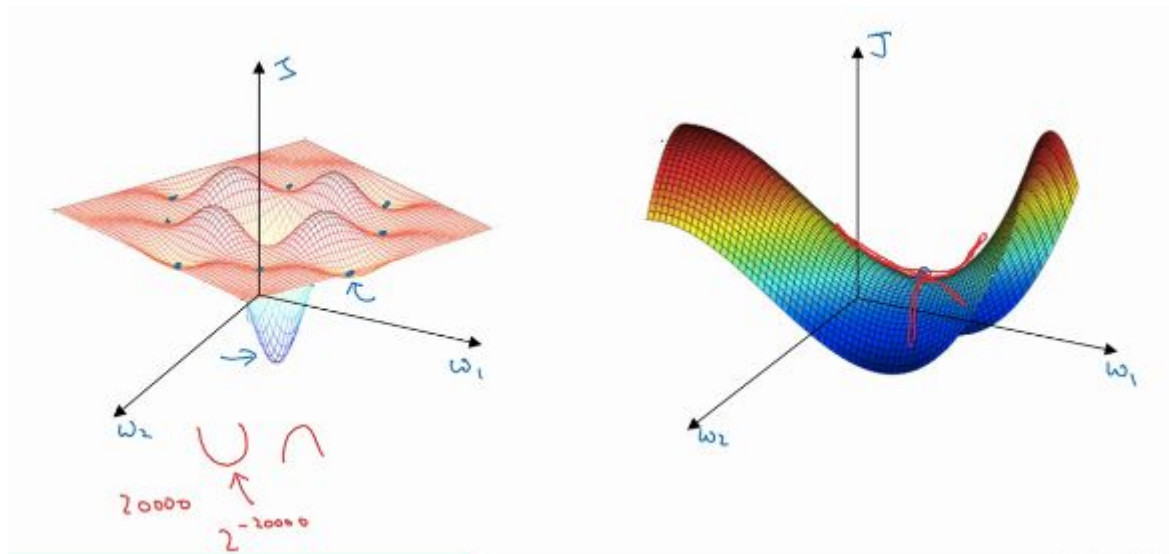
$$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0 \quad - \text{exponentially decay.}$$

$$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \cdot \alpha_0$$

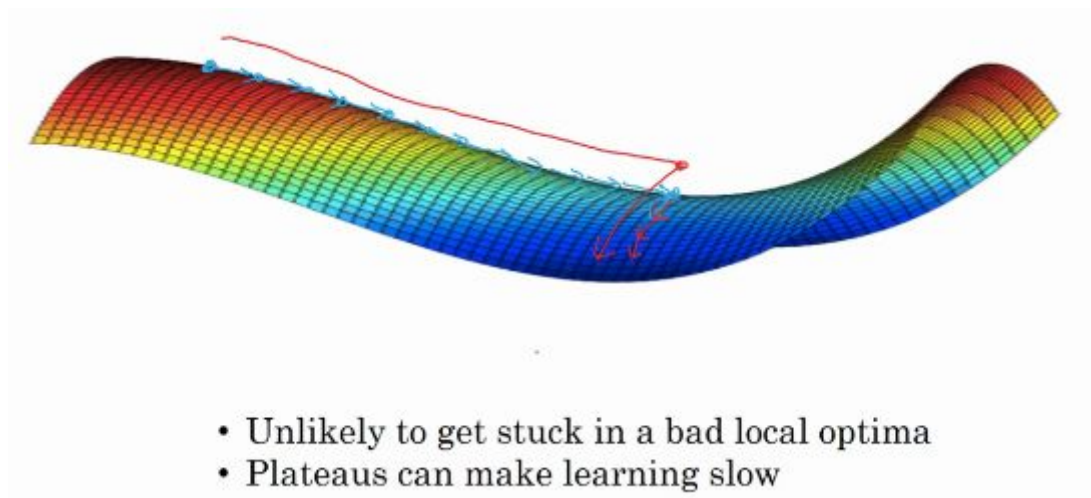
discrete staircase

Local optima in neural networks

Generally in the estimation of global minimum in neural networks, we don't have local minimums but saddle points.



Resulta que el problema de las redes neuronales no son los minimos locales sino los plateaus donde el gradiente se vuelve pequeño durante un amplio rango de parametros.



Hyperparameter tuning, Batch Normalization and Programming Frameworks

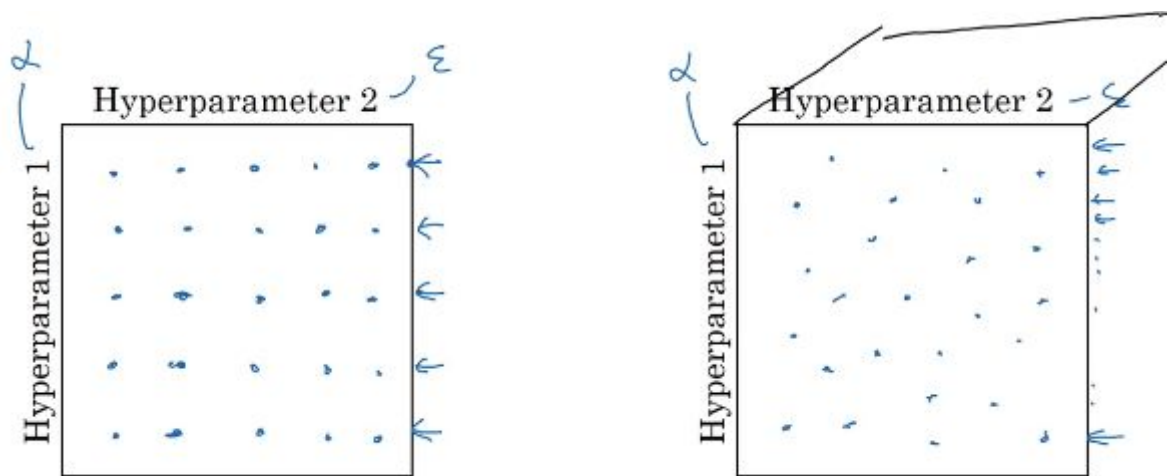
Tuning process

Como elegimos adecuadamente al conjunto de hiperparametros: alfa, beta, (beta1, beta2, epsilon) para adam, num capas, numero de unidades ocultas, decaimiento de la tasa de aprendizaje, tamaño del mini-batch?

No todos son igual de importantes. Alfa siendo el mas importante. beta, el numero de unidades ocultas o el tamaño del minibatch las siguientes.

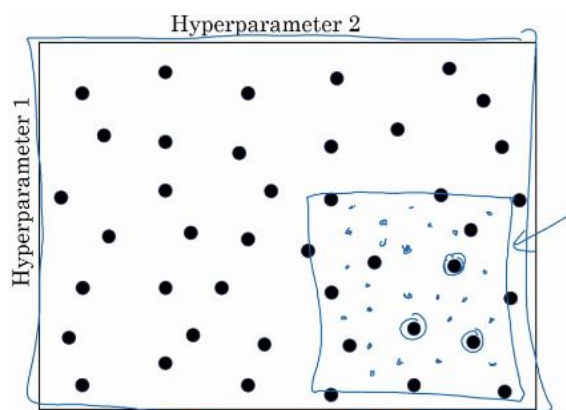
- 1) Elegir valores aleatorios de hiperparametros: no usar una grilla!!

Cuando el numero de hiperparametros es pequeño la grilla puede funcionar bien, sino no.



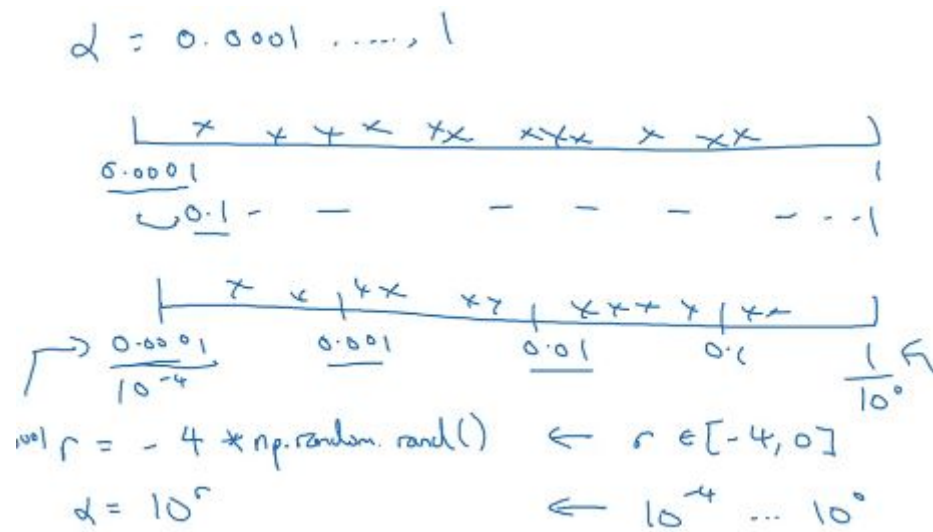
- 2) Coarse to fine

Enfocarse mas en una region particular del espacio de parametros.



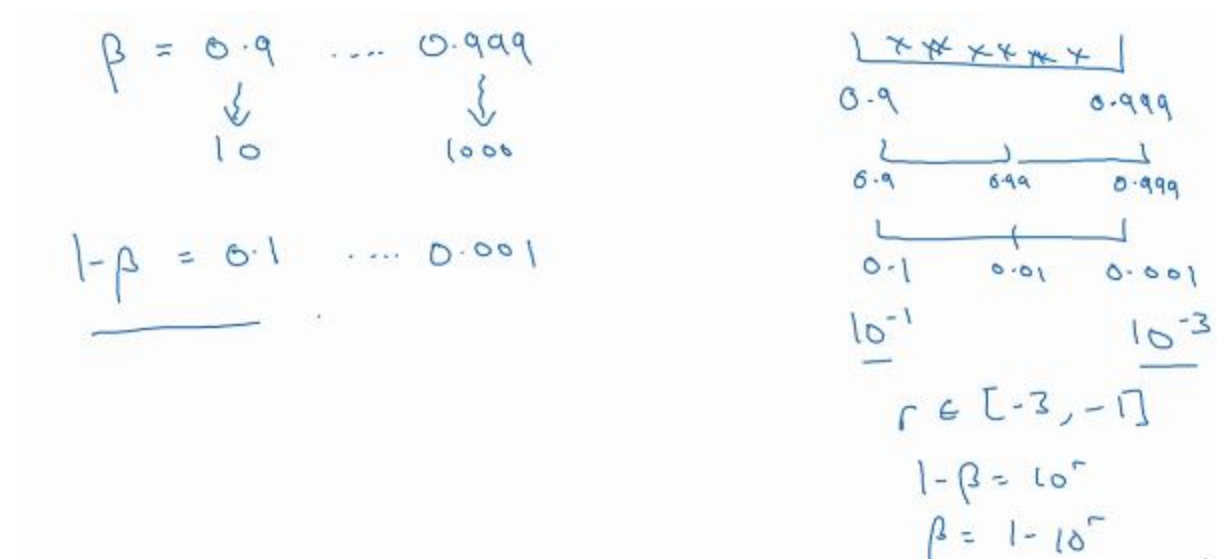
Usar la escala adecuada para elegir hiperparametros

Por ejemplo para alfa entre $10e-4$ y 1 conviene elegir en escala logaritmica.



Hiperparametros para proemdios pesados exponencialmente

por ejemplo beta yendo desde 0.9 a 0.999. Conviene hacer algo similar a lo visto anteriormente.



Normalizacion Batch

Este algoritmo hace la busqueda de hyperparametros mucho mas simple, haciendo la red neuronal mas robusta. La eleccion de hyperparametros es hecha en un rango mucho mas amplio provocando que se pueda entrenar mas facilmente redes neuronales mas profundas.

Normalizando activaciones en una red

Ya se ha visto previamente como realizar una normalizacion de los features de entrada y el efecto que provoca un cambio en la regresion. Pero como varia un modelo mas profundo?

Se podra normalizar las capas ocultas de manera que w, b se calculen mas rapidamente?

Implementar Batch Norm

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

Given some intermediate values in NN $z^{(1)}, \dots, z^{(m)}$

If $\sigma = \sqrt{\frac{1}{m} \sum_i (z_i - \mu)^2}$

$\mu = \frac{1}{m} \sum_i z^{(i)}$

$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$

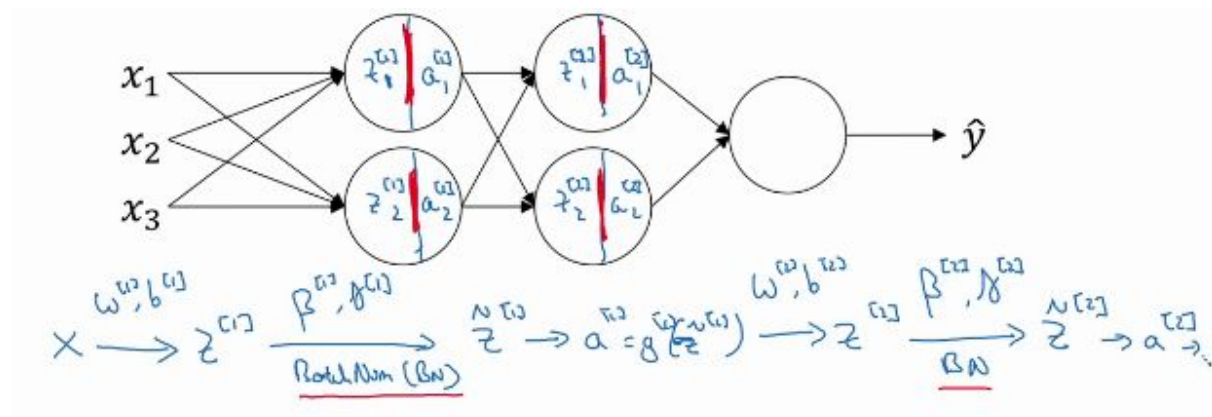
$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$

$\hat{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$

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 $\tilde{z}^{[1]}$ se calcula $a^{[1]}$ evaluando \tilde{z} con la funcion de activacion. Esto se realiza para todas las capas ocultas.



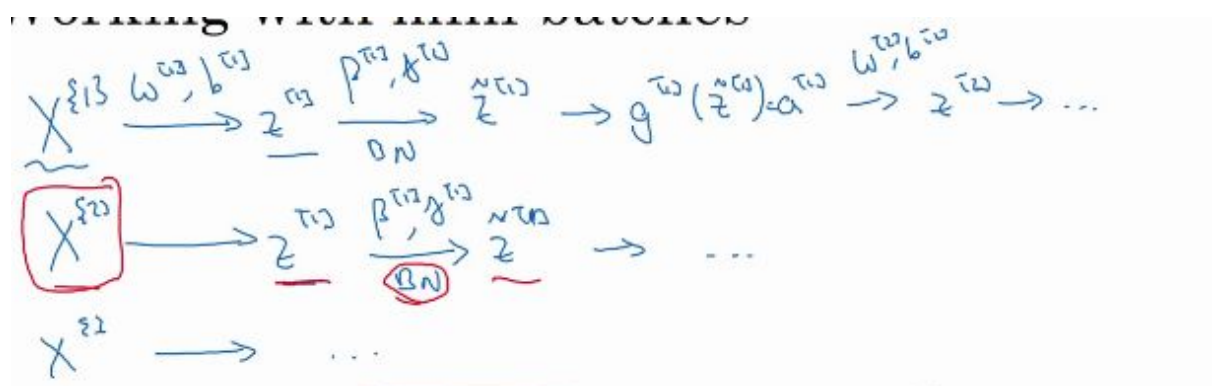
Los parametros a ajustar seran el doble.

Parametros: $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(L)}, b^{(L)}$
 $\rightarrow \beta^{(1)}, \gamma^{(1)}, \beta^{(2)}, \gamma^{(2)}, \dots, \beta^{(L)}, \gamma^{(L)}$

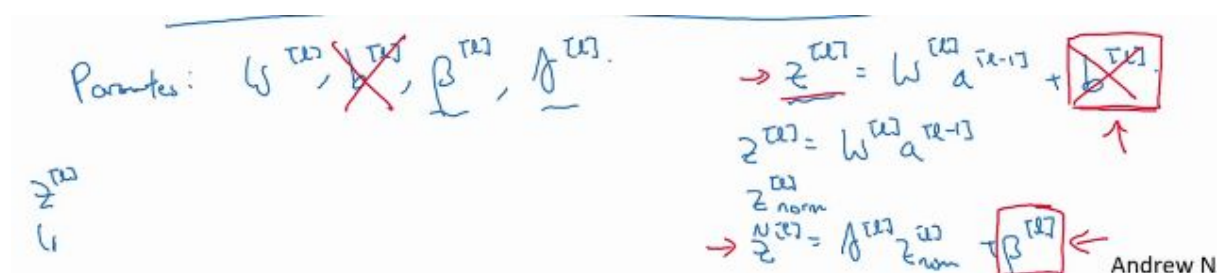
En tensor flow esto se implementaria con **tf.nn.batch_normalization**.

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 parametro b es innecesario calcularlo.



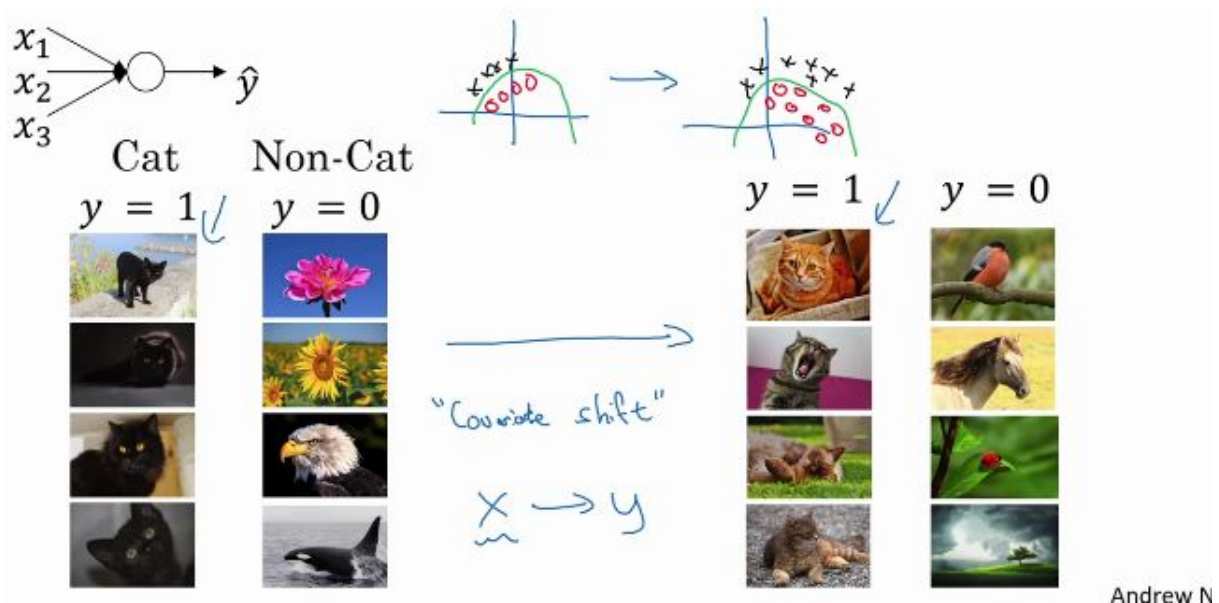
Implementando gradiente descendiente

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

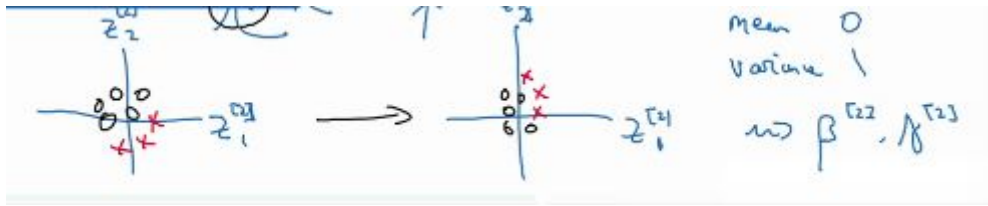
for $t = 1 \dots \text{num Mini Batches}$
Compute forward pass on X^{batch} .
In each hidden layer, use BN to replace \hat{z}^{layer} with \tilde{z}^{layer} .
Use backprop to compute $\frac{dW^{\text{layer}}}{dt}$, $\frac{d\beta^{\text{layer}}}{dt}$, $\frac{d\gamma^{\text{layer}}}{dt}$.
Update params $\left. \begin{aligned} W^{\text{layer}} &:= W^{\text{layer}} - \alpha \frac{dW^{\text{layer}}}{dt} \\ \beta^{\text{layer}} &:= \beta^{\text{layer}} - \alpha \frac{d\beta^{\text{layer}}}{dt} \\ \gamma^{\text{layer}} &:= \gamma^{\text{layer}} - \alpha \frac{d\gamma^{\text{layer}}}{dt} \end{aligned} \right\} \leftarrow$
Works w/ momentum, RMSprop, Adam.

Veamos porque es que realmente Batch Normalization funciona y hace que el aprendizaje sea mas rapido.

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.



Basicamente Batch Norm lo que hace es tratar de que los parametros no varien demasiado ante el cambio de input values. Los parametros beta y gamma deberian fluctuar muy poco.



Batch norm aplicado a mini-batch actua como regularizacion

- 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 actiovacion de cada capa oculta.
- Esto tiene un efecto de regularizacion.

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

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

Resumiendo Batch norm

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

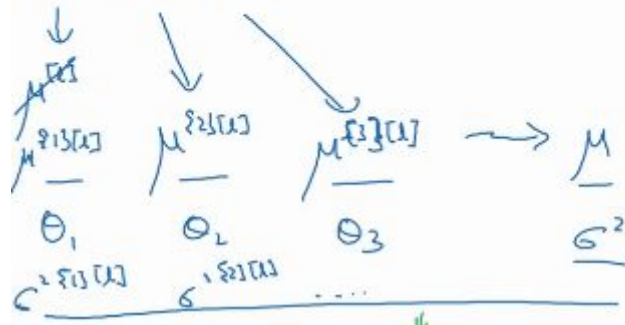
$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

Dado que para calcular z tilde se necesita mu y sigma y, para que estos sean calculados con precision deben ser calculados con un numero considerable de ejemplos. Para hacer esto hace lo siguiente: se calcula mu para el conjunto de minibatches del training set y de ese conjunto se calcula un estimador. Esto mismo se hace para sigma.

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

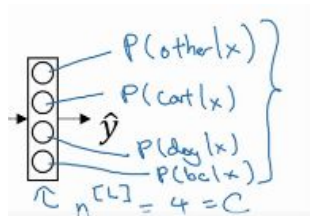
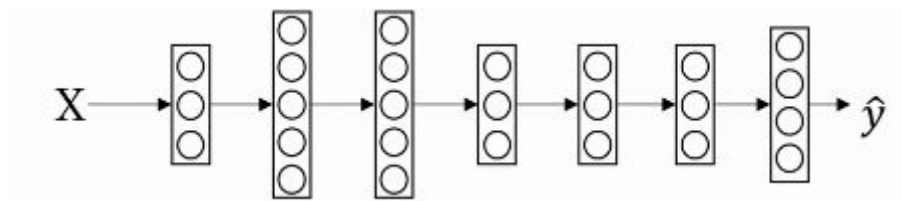
μ, σ^2 : estimate using exponentially
weighted average (across mini-batches).

$X^{(1)}, X^{(2)}, X^{(3)}, \dots$

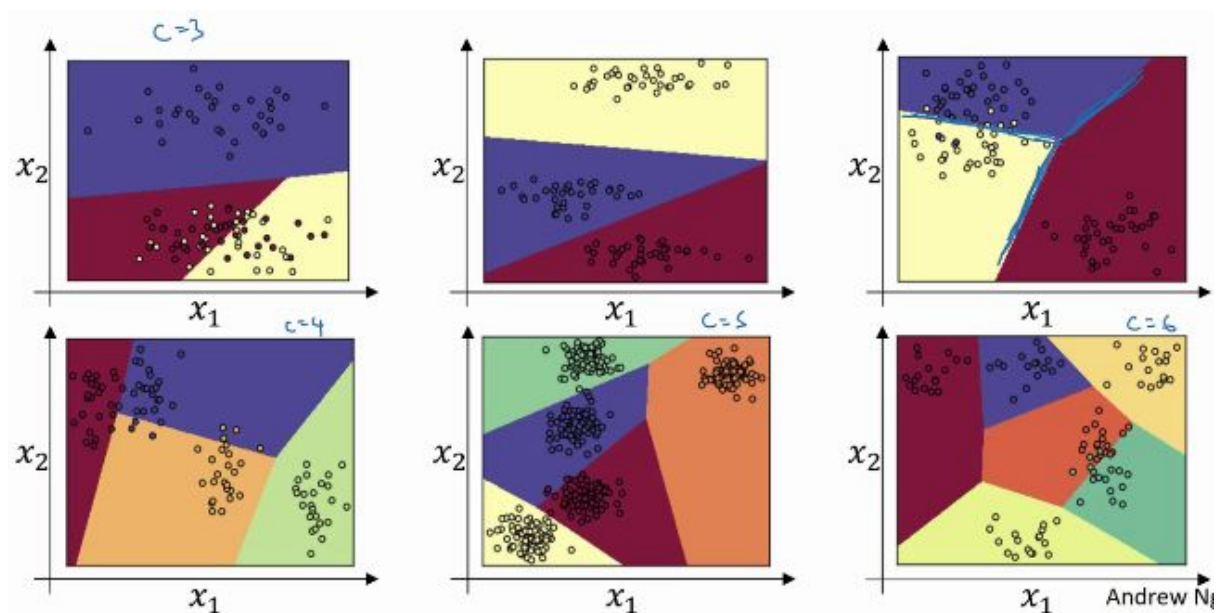


Softmax regression

Si en lugar de requerir una clasificacion binaria se requiere una clasificacion 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.



Para tener boundaries mas complejos se deberian agregar capas ocultas.

Entrenar un clasificador Softmax

Si el numero de salidas en softmax = 2, se obtiene regresion logistica.

Deep Learning Frameworks

Ejemplos de estos frameworks:

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

Eleccion:

Facilidad de programacion (desarrollo e implementacion).

Velocidad para correr.

Que sea verdaderamente open source (con una gran comunidad que lo mejore).

Tensor Flow

Ejemplo de motivacion: minimizar la funcion de coste. De manera mas general se tendra una funcion $J(w,b)$.

Handwritten diagram showing the cost function $J(w) = w^2 - 10w + 25$, which is equivalent to $(w-5)^2$. The minimum is indicated at $w=5$.

```
import numpy as np
```

```
import tensorflow as tf
```

```
w = tf.Variable(0, dtype=tf.float32) # Initialize to zero
```

```
cost = tf.add(tf.add(w**2, tf.multiply(-10., w)), 25) # Define cost
```

```
# cost = w**2 - 10*w + 25 also works. Some basic ops are overloaded.
```

```
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost) # Define training method
```

```
init = tf.global_variables_initializer()
```

```
session = tf.Session()
```

```
session.run(init)
```

```
print( sess.run(w) ) -> 0.0
```

Como no se corrio nada todavia se obtiene cero.

```
session.run(train) # correr un paso de gradient descent -> 0.1
```

```
for i in range(1000):
```

```
session.run(train)
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()
session.run(init)
print(session.run(w))

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

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

Structuring Machine Learning Projects

Por que 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
 - ...

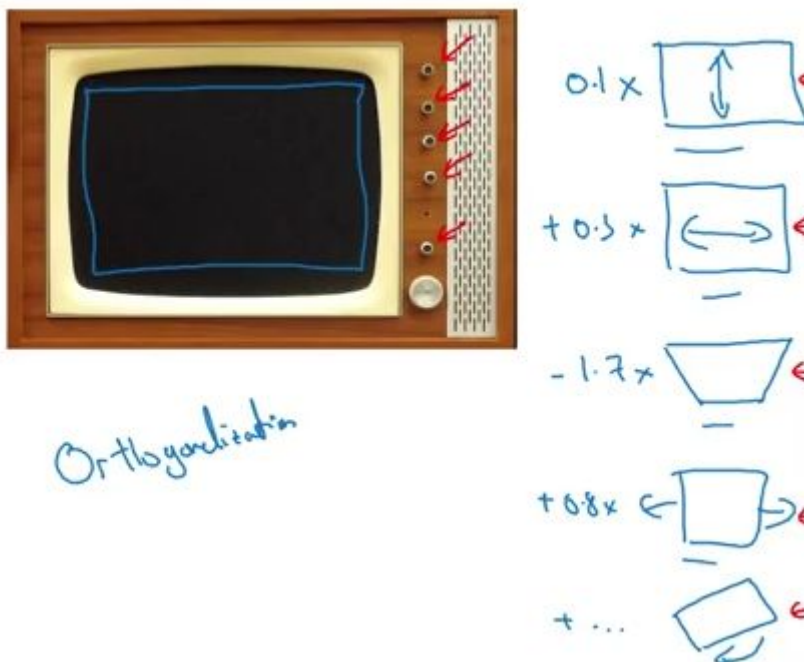
Andrew Ng

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

Ortogonalizacion

Que hiperparametro tunear para obtener un efecto deseado?

Veamos un ejemplo con una television: Las televisiones viejas tenian unas manijas que permitian modificar por separado la altura de la posicion de la pantalla, el alto, el ancho, el trapezoide, la rotacion, 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 seria muy dificil de calibrar.



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



→ Steering]

→ { Acceleration
Braking }

En lugar de estos controles se podrian definir dos como los que se muestran debajo. Pero, al ser una combinacion del angulo y de la velocidad el manejo del auto seguramente sera dificultosa. Es por ello que idealmente uno querria tener un mecanismo que separe ambas contribuciones.

$$\frac{0.3 \times \text{angle} - 0.8 \text{ speed}}{2 \times \text{angle} + 0.9 \text{ speed}}$$

speed ↑

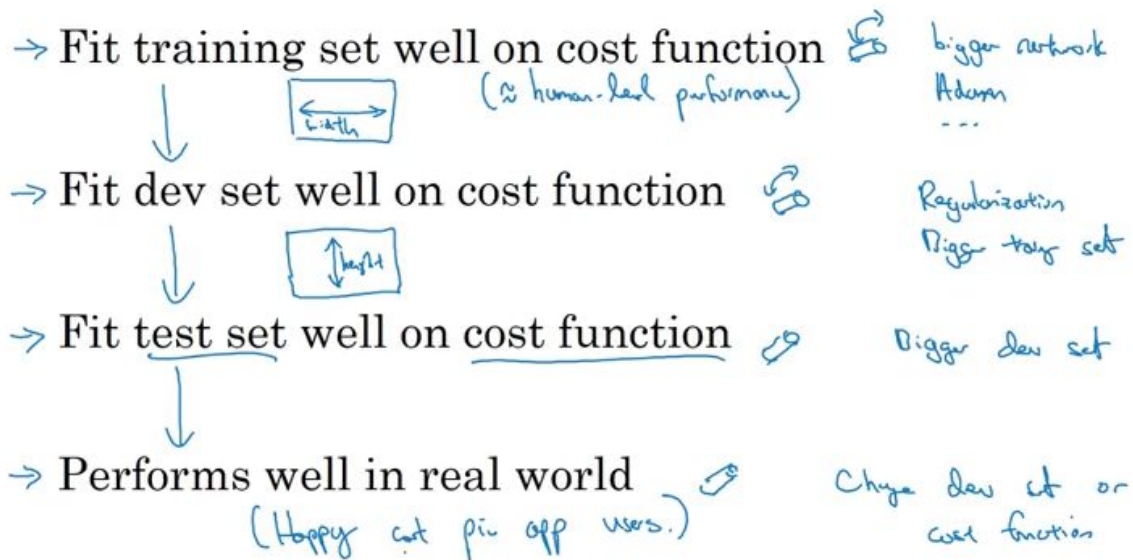
angle →

A.

Como se extrapola esto en proyectos de Machine Learning?

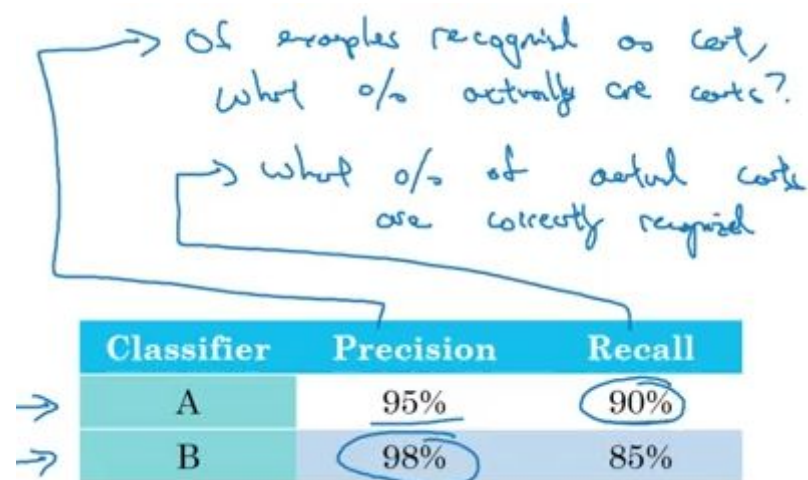
En ML hay cuatro pasos fundamentales.

1. Ajustar el conjunto de entrenamiento con la funcion de coste: Si esto no se cumple se puede intentar agrandar la red neuronal o probar el Metodo de Adam. Esto es analogo a ajustar una perilla de la television, ya que hacer una de estas modificaciones mejorara mas que nada el ajuste del conjunto de entrenamiento sobre la funcion de coste.
2. Ajustar el conjunto de desarrollo con la funcion de coste: Si el conjunto de desarrollo no es ajustado correctamente por la funcion de coste conviene implmentar regularizacion o probar un conjunto de entrenamiento mas grande.
3. Ajustar el conjunto de testeo con la funcion de coste: si el conjunto de entrenamiento no se ajusta correctamente a la funcion de coste conviene agrandar el conjunto de desarrollo.
4. Buen rendimiento en el mundo real: en este caso quizas convenga cambiar el conjunto de desarrollo o la funcion de coste.



Metrica de evaluacion de un numero unico

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



Precision: 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 casificador B lo hace para la precision. El hecho de usar dos indicadores hace dificil determinar con cual de los dos conviene seguir estudiando.

Es por ello que se recomienda utilizar una unica metrica de evaluacion: en la literatura de ML se suele utilizar el score F1 que combina ambas metricas.

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

El mismo se define como el promedio armonico entre la precision y el recall.

$$F_1 \text{ score} = \text{"Average" of P and R.}$$

$$\left(\frac{2}{\frac{1}{P} + \frac{1}{R}} \cdot \text{"Harmonic mean"} \right)$$

Supongamos ahora que se tiene la siguiente tabla donde acorde a la region se registran el error de cada clasificador. Es ideal calcular el promedio teneineod en cuenta que es un estimador aceptable y a partir de el decidir cual 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 parametros para determinar con que clasificador quedarse. Consideremos por ejemplo tres clasificadores donde se registra la exactitud y el tiempo de computo para cada uno de ellos.

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

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

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

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

Distribucion de los conjuntos de entrenamiento/desarrollo/testeo



dev set
+
Metric

Clasificacion 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
- } Dev
- } Test

Tamaño del conjunto de desarrollo y testeo

Recien se dijo que ambos deben provenir de la misma distribucion pero como debe ser el tamaño de los mismos?.

En la era del BigData 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.

Cuando 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/users : Prefer B.

Metric: classification error

Algorithm A: 3% error → pornographic

✓ Algorithm B: 5% error

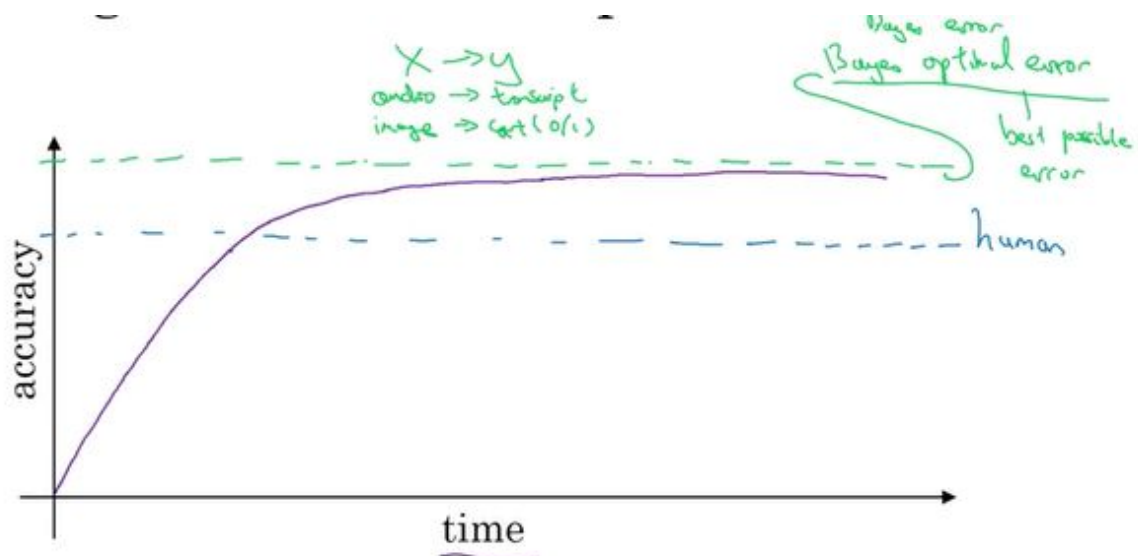
Error: $\frac{1}{\sum w^{(i)}} \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathbb{I}\{y_{pred}^{(i)} \neq y^{(i)}\}$

↗ predicted value (0/1)

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

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 transcrito con una precisión máxima debido al ruido, o una imagen borrosa no sería identificable como un gato.



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

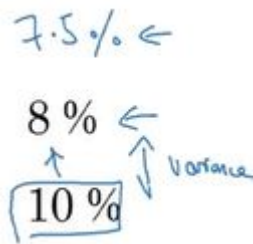
- Obtener datos etiquetados por humanos
- obtener informacion desde un analisis manual del error. Por que una persona hizo algo bien?
- Mejor analisis del bias/varianza.

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

Humans	1%
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 mas grandes o correr conjunto de entrenamiento mas tiempo.

Ahora supongamos que los humanos tienen un error del 7.5% debido a problemas de resolucion por ejemplo. Uno querria 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 que 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).

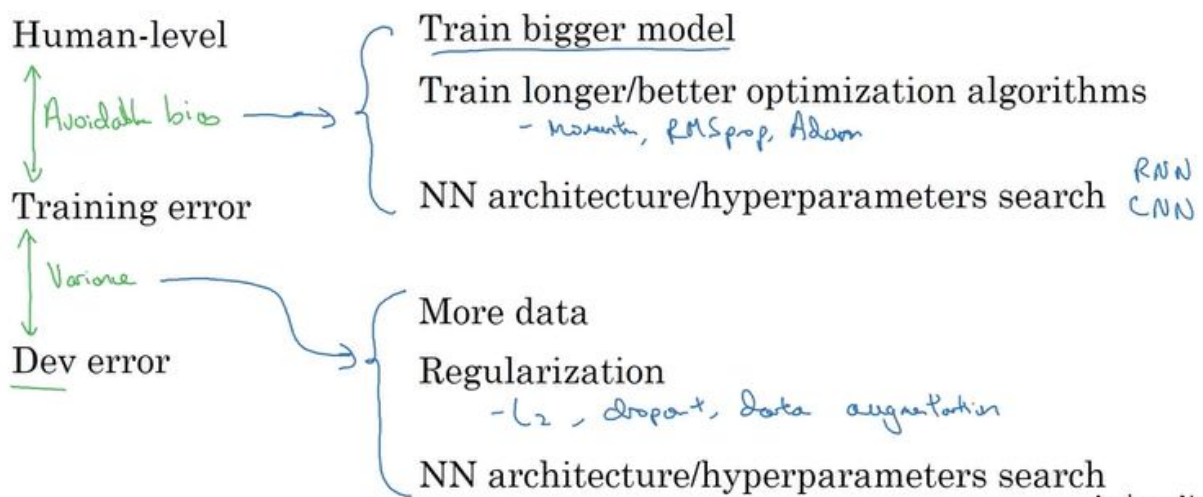
Problemas donde ML sobrepasa el rendimiento humano

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

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

Mejorar el rendimiento del modelo

Resumiendo lo visto.



Error Analysis

Este es el proceso de examinar manualmente errores para dar una idea de como 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%!



90% accuracy
→ 10% error

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 ↓
50% 5%

Evaluar multiples 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 seria crear una tabla con las imagenes que se miraran como filas y las clasificaciones como columnas y se completa de acuerdo a lo que corresponda. Por ultimo 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 imagenes borrosas o de los "grandes gatos".

Que hacemos cuando tenemos ejemplos en el cjo de desarrollo que estan 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 analisis de errores agregando una columna extra contando el numero 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 correccion debido al mal etiquetado no disminuira el error significativamente. Solo seria 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.

Overall dev set error	10%	2%
Errors due incorrect labels	0.6% ←	0.6%
Errors due to other causes	9.4% ←	1.4%
		2.1%
		1.9%

Goal of dev set is to help you select between two classifiers A & B.

Andrew

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 rapidamente, 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 aplicacion del celular pretende distinguir gatos.

En este caso se tienen dos fuentes: una proveniente de paginas web y otra de los usuarios via la aplicacion, siendo esta ultima menos profesional con mas defectos de distorsion enfoque etc.

Data from webpages



Data from mobile app

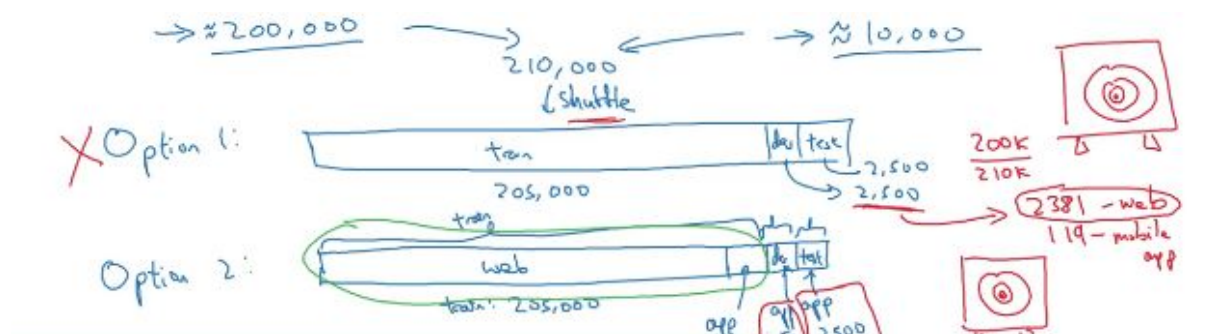


Supongamos que de la web se tienen 200mil imagenes de gatos aproximadamente y que se tiene un numero relativamente bajo de usuarios, teniendo un dataset de 10mil imagenes de gatos via aplicacion movil. Uno se encuentra en un dilema porque el sistema deberia funcionar mejor sobre la distribucion con menos datos.

Para resolver esto hay varias opciones.

Opcion 1: juntar los datos y distribuirlos de manera aleatoria. La ventaja es que la distribucion sera uniforme, pero posee una desventaja de que practicamente no hay datos provenientes de la aplicacion para el dev/Test set. Esta opcion se recomienda no utilizar.

Opcion 2: Formar el training set con las imagenes de la web completas mas 5mil de la aplicacion. El dev/test siendo completamente de la aplicacion. Como desventaja el training set tiene una distribucion distinta.



Es recomendable usar toda la informacion para la red neuronal? No.

Ejemplo clasificador de gatos.

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

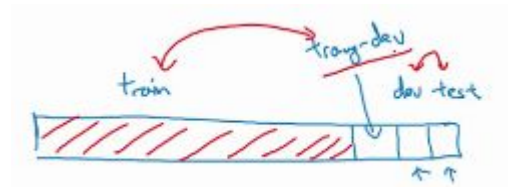
Training error 1%
Dev error 10%

Si las distribuciones del training/dev set son las mismas, podemos concluir que habria que disminuir la varianza. Pero si provienen de distintas distribuciones esta suposicion no sigue siendo valida. Tal vez no hay un error de varianza y simplemente el dev set posee imagenes mas complicadas para clasificar.

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

1. El algoritmo vio datos del cijo de entrenamiento pero no de desarrollo.
2. La distribucion 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%
→ Training-dev error	9%	1.5%
→ Dev error	10%	10%
	Variance	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%	0%
Training error	10%	10%
Training-dev error	11%	11%
Dev error	12%	20%
	bias	bias
	variance	data mismatch

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

Que 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.

Artificial data synthesis

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

Car recognition:



N 20 cars



Synthesized



All 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 entreno 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.

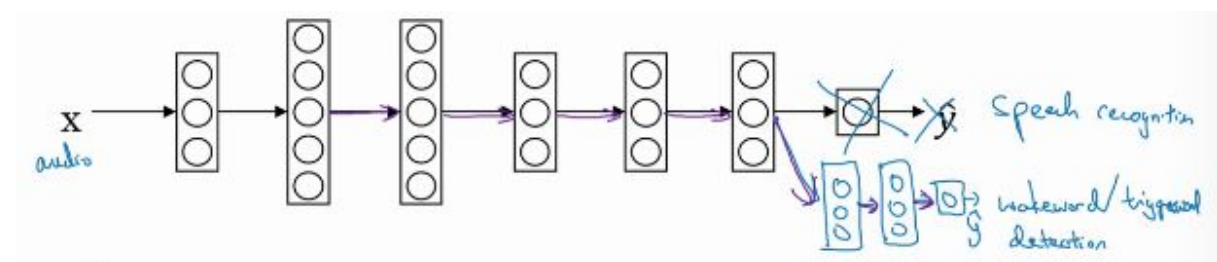
Luego se procederá a entrenar la red neuronal para esta tarea. Si el dataset es lo suficientemente grande se podría entrenar la red completa, sino la última capa únicamente.

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.



Cuando este proceso tiene sentido? 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 unicamente 100. En reconocimiento de habla 10mil horas de datos, mientras que en deteccion de trigereo 1 hora.

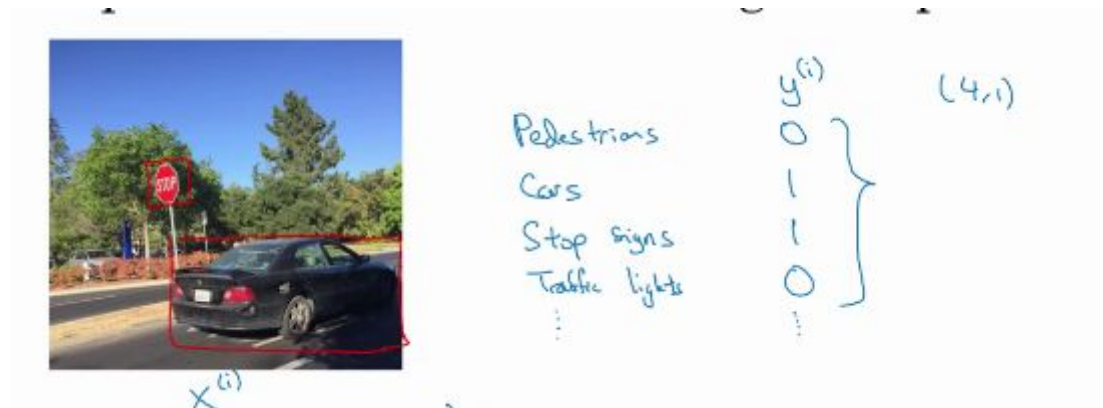
A su vez, tiene sentido cuando ambos sistemas tienen el mismo tipo de entrada, x .

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

Multi-task learning

Esto es como una version generalizada de transfer learning, aprendiendo de muchas tareas en simultaneo.

Ejemplo de conduccion automatica.

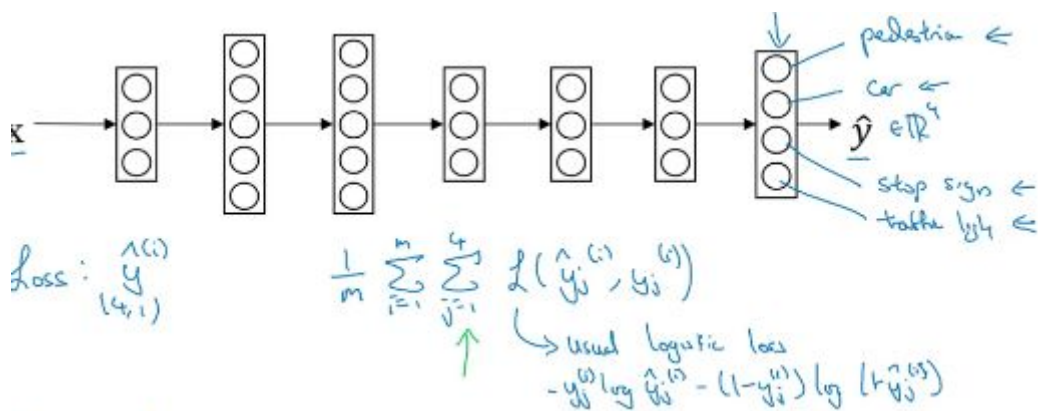


Mirando a los outputs como un conjunto.

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(m)} \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

$(4, m)$

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 mas eficiente que entrenar cuatro redes neuronales para distinguir los cuatro elementos por separado.

Resumen: cuando 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.

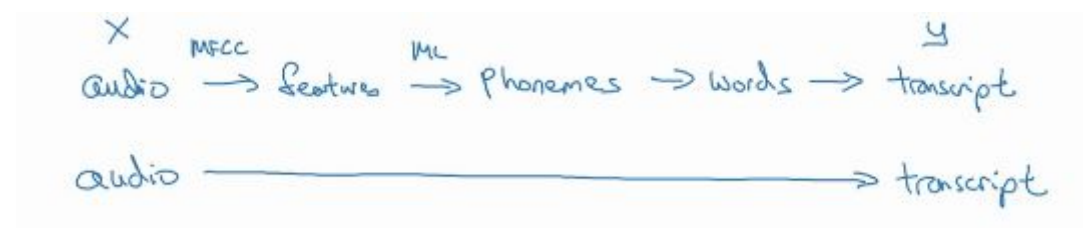


- Can train a big enough neural network to do well on all the tasks.

End-to-end deep learning

Este metodo toma muchas etapas de un proceso de aprendizaje y los une en una unica etapa.

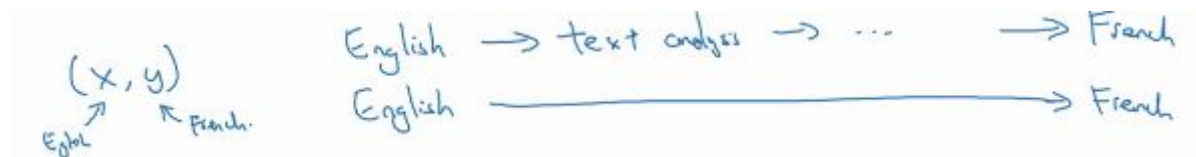
Ejemplo con reconocimiento de audio



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

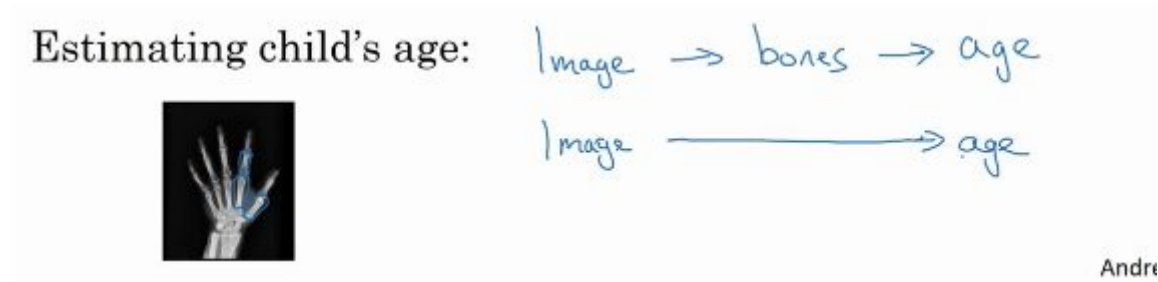
Para que esto funcione correctamente se necesitan muchos datos.

Ejemplo 2, traducciones

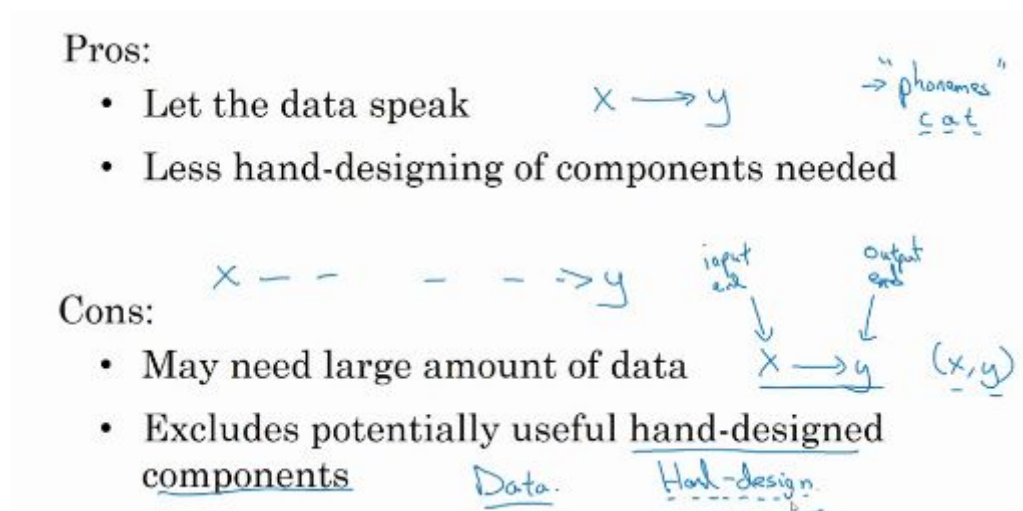


Ejemplo 3, estimar edad de chicos a partir de images por rayos x.

Este ultimo metodo no funciona del todo bien debido a la poca informacion de la que se dispone.

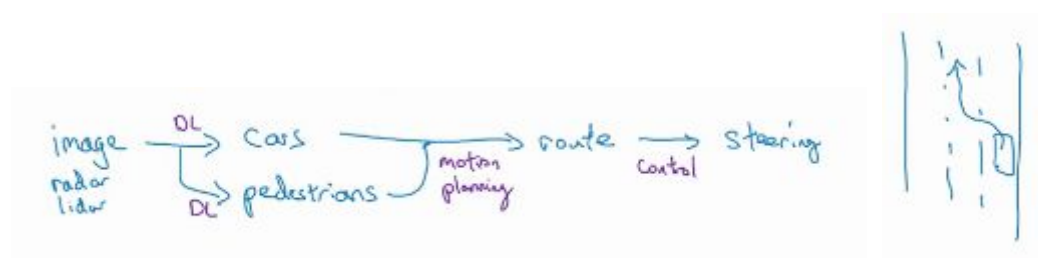


Ventajas y desventajas del end-to-end deep learning



Como se construye un auto que se maneja solo?

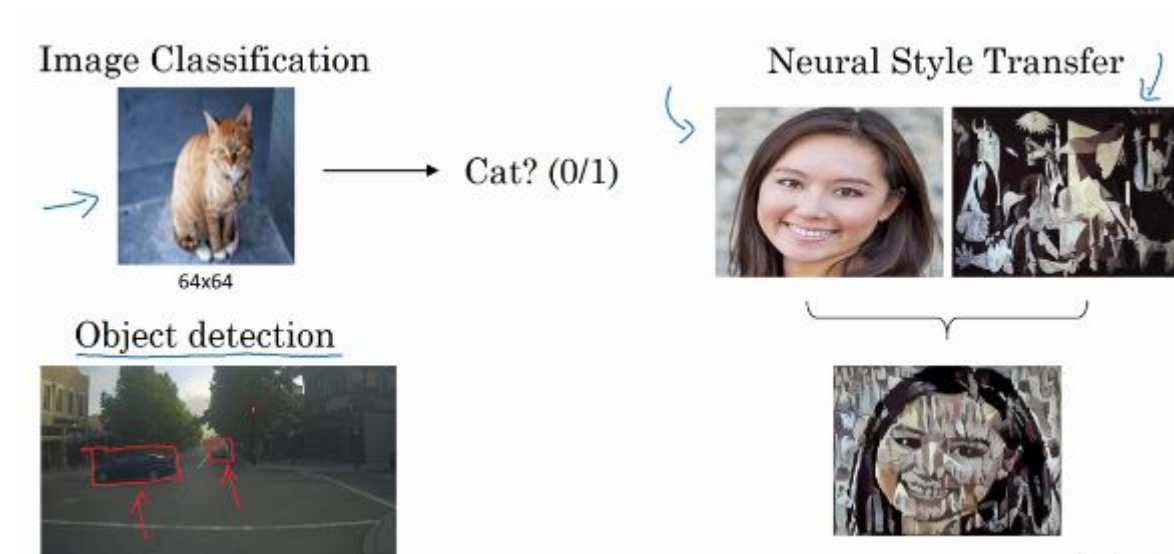
Esto no es un enfoque end-to-end



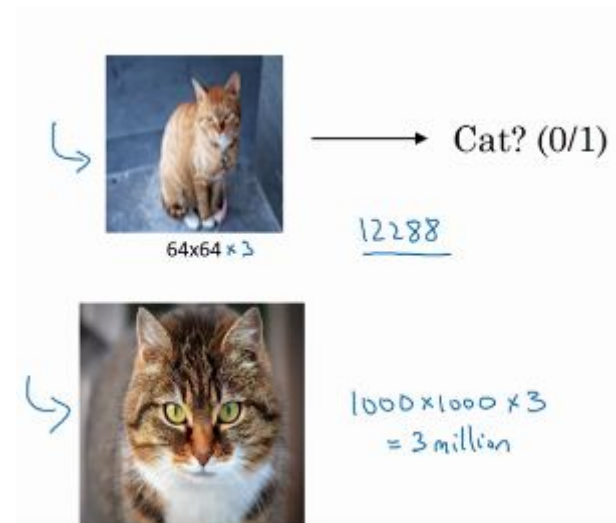
Convolutional Neural Networks

Computer Vision

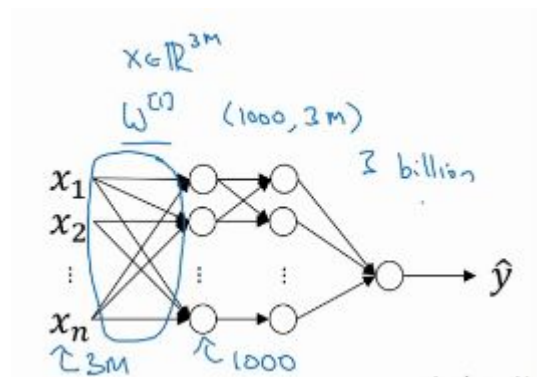
Problemas en computer vision



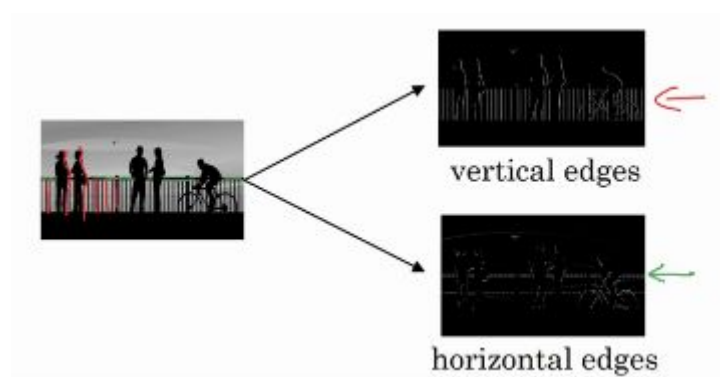
Deep Learning en imagenes grandes



Una red neuronal estandar para una imagen de alta resolucion tendria alrededor de 3 mil millones (billions) de parametros. Es por ello que se implementara un nuevo estilo de red neuronal, la red neuronal convolucionada.

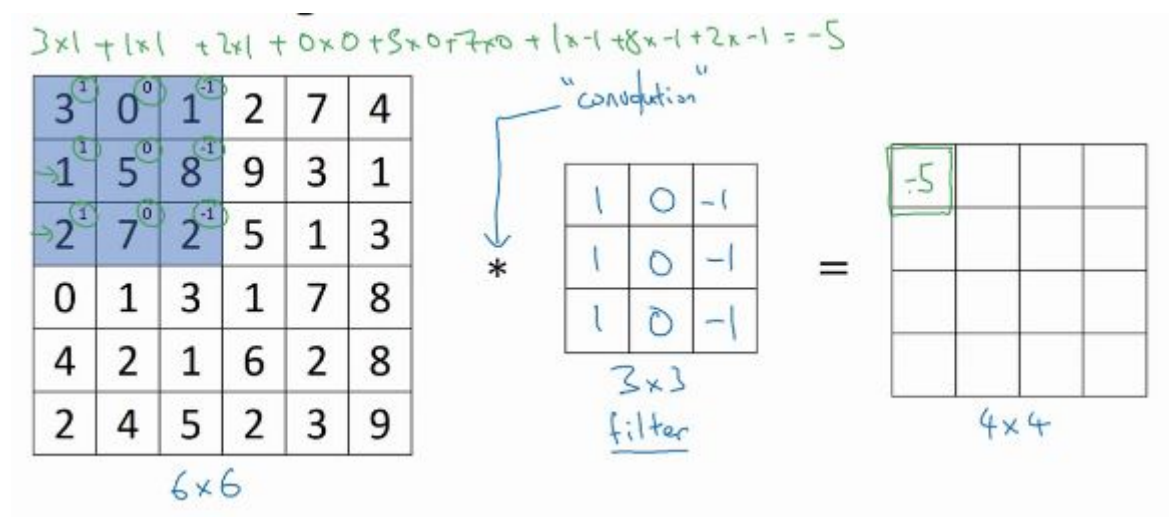


Ejemplo de detección de bordes



Detección 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. ¿Cómo 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.



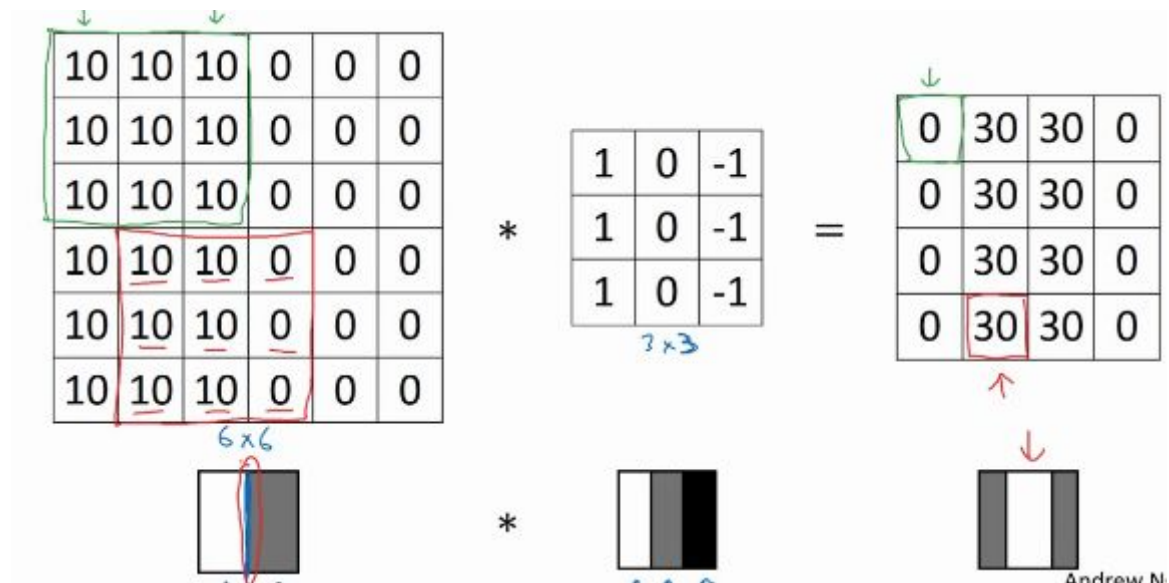
El resultado de la convolucion es entonces

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

4x4

Esto en python se implmenta con `conv_forward`. En tensorflow `tf.nn.conv2d`. En keras `conv2d`, etc.

¿Porque esto realiza una deteccion de bordes verticales? Veamos un ejemplo mas simple primero.



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.

Mas sobre deteccion de bordes

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

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

 \times

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

 $=$

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

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

Deteccion ejes horizontales

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

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

Horizontal

Veamos un ejemplo

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

 \times

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

 $=$

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Otros ejemplos de filtros para bordes verticales

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

 \rightarrow

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

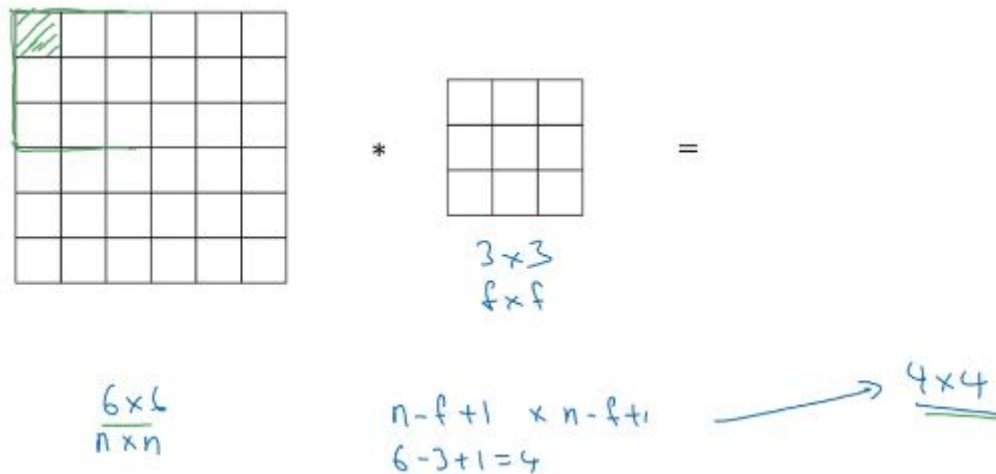
3	0	-3
10	0	-10
3	0	-3

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



Padding

una modificacion a la tecnica de convolucion que se vio previamente.



The main benefits of padding are the following:

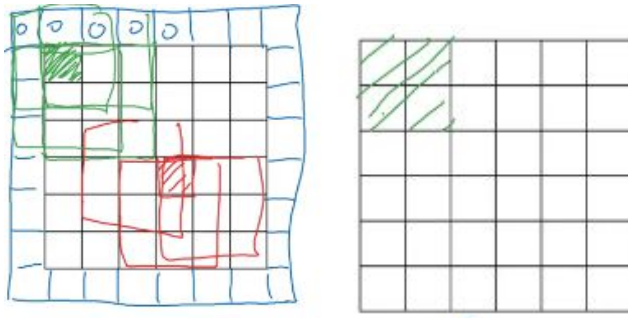
- It allows you to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as you go to deeper layers. An important special case is the "same" convolution, in which the height/width is exactly preserved after one layer.
- It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

Desventajas de la convolucion:

- Aplicar continuamente las tecnicas de convolucion hace que la imagen original se reduzca.
- Los bordes son utilizados una unica 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.



izq.: Input. Der.: Output

Considerando a "p" o padding como el numero de capas agregadas, en este caso, $p = 1$.
Mientras que ahora se satisface la ecuacion:

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

$$6+2-3+1 \times \text{---} = 6 \times 6$$

Cuántas capas agregar? Valid y Same convoluciones

- Valid: no hay padding
- Same: pad tal que el output size sea el mismo que el input size.

Esto es

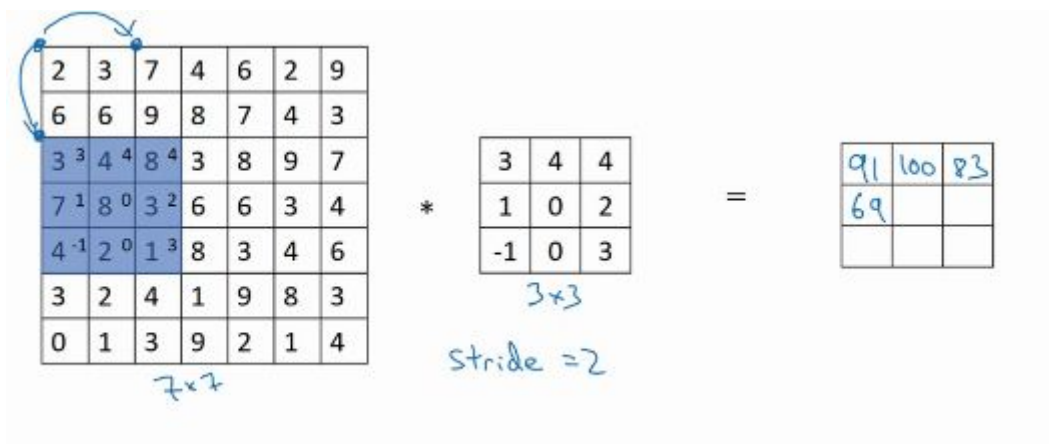
$$n+2p-f+1 \times n+2p-f+1$$

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

El tamaño de los filtros generalmente es impar.

Strided Convolutions

Empecemos con un ejemplo



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

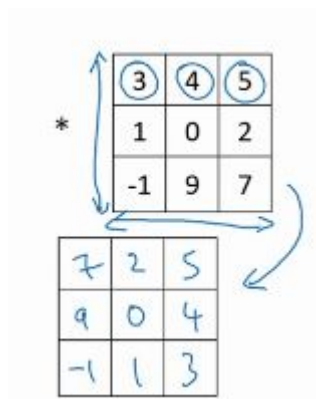
padding p stride s

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

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

Cross-correlation vs convolution

Resulta que en la literatura de ML se suele decir que la operacion recientemente explicada y realizada es llamada convolucion pero realmente es una cross-correlacion. En los libros de matematica una convolucion viene dada por una operacion previa que viene de espejar el filtro de la siguiente manera

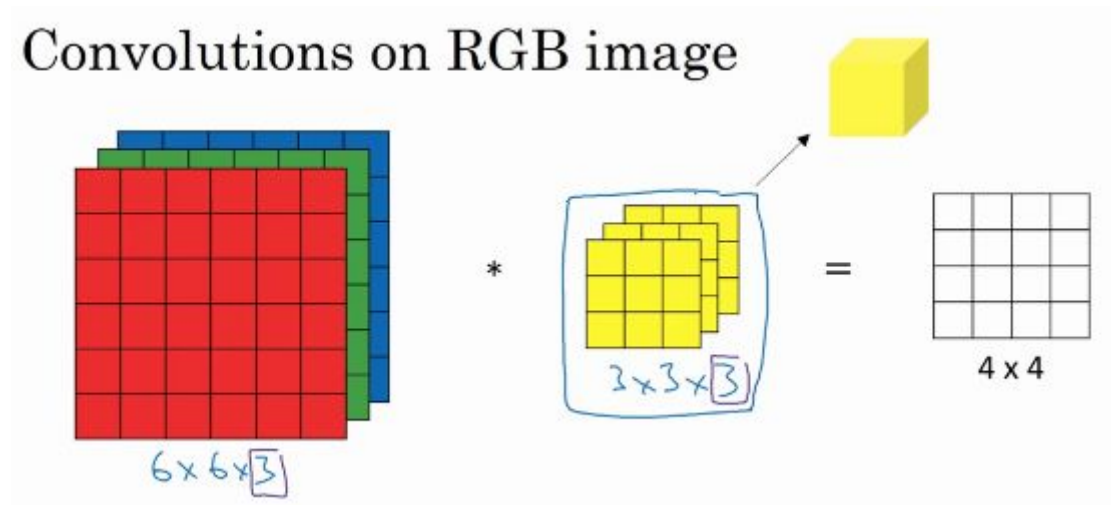


y luego aplicar el operador $*$.

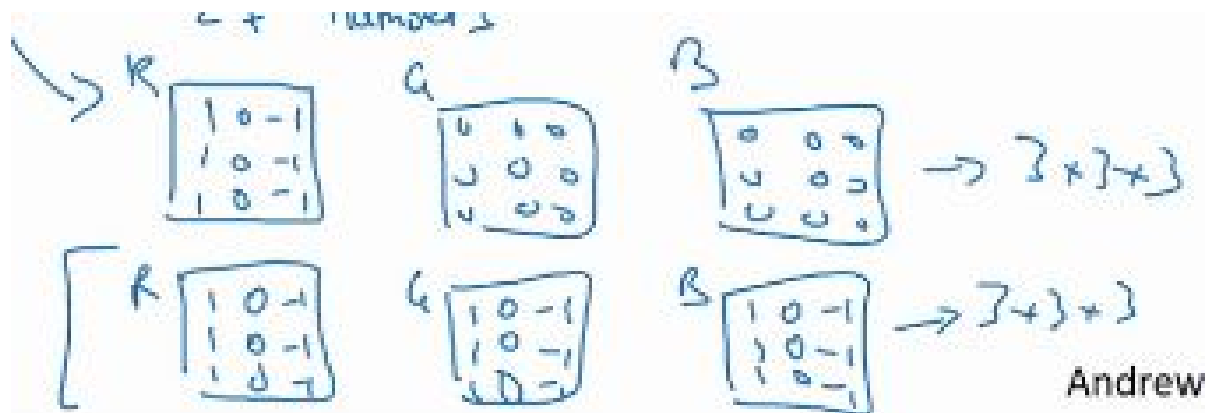
Convolucion sobre volúmenes (o imagenes RGB)

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

Convolutions on RGB image



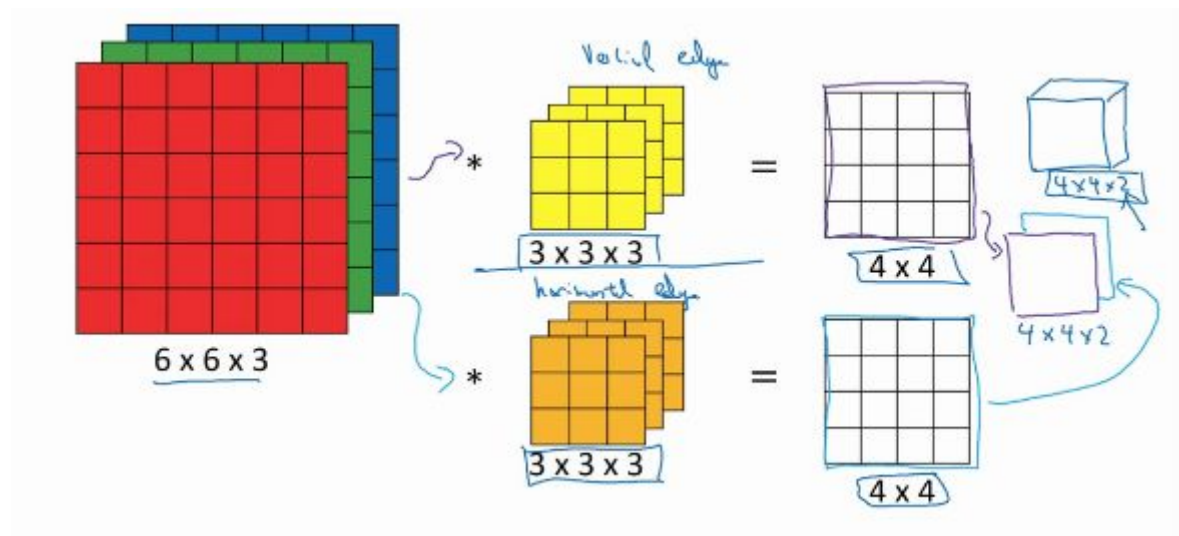
La eleccion del filtro puede ser la siguiente



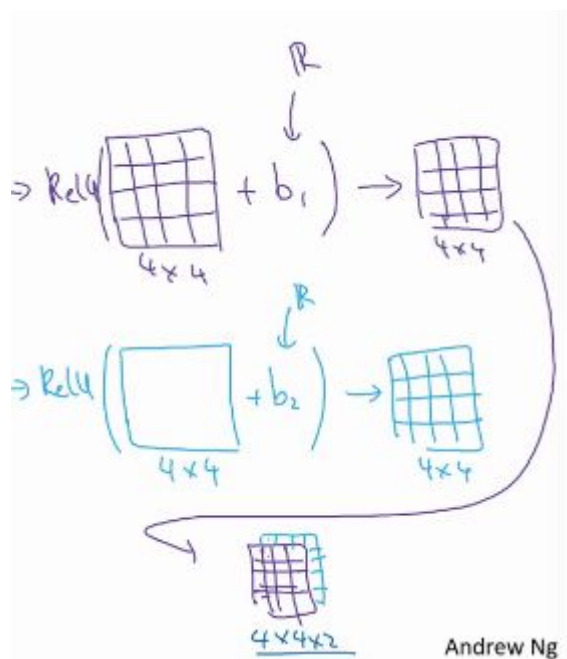
donde la primera busca bordes unicamente para el color rojo, mientras que la segunda para cualquier color.

Multiples filtros

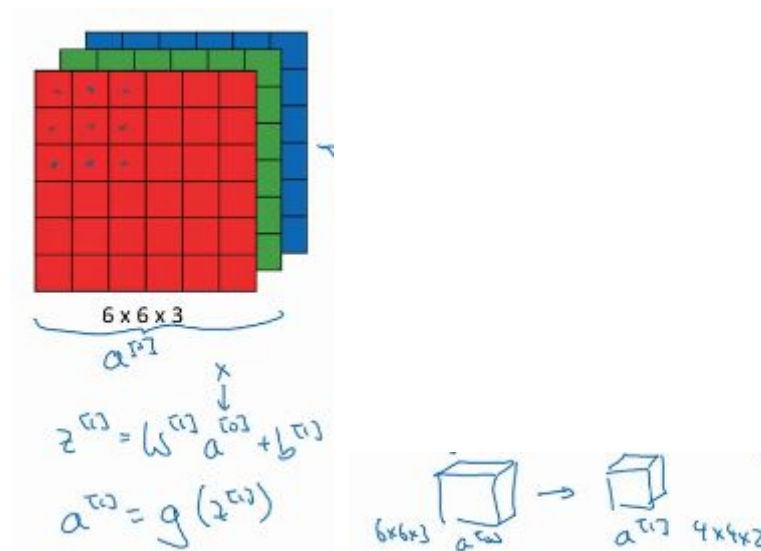
Tambien se pueden aplicar multiples filtros por separado. Por ejemplo para bordes verticales y otro para horizontales y juntar los resultados de ambos como muestra la figura.



Una capa de red convolucional



Claramente, la matriz de 4x4 juega el rol de $w \cdot a$



Resumen de la notacion

If layer l is a convolution layer:

$f^{[l]}$ = filter size
 $p^{[l]}$ = padding
 $s^{[l]}$ = stride
 $n_c^{[l]}$ = number of filters

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$
 Output: $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$

Each filter is: $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$
 Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
 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.

$$n_{HW}^{[l]} = \left\lfloor \frac{n_{HW}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$A^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$

Ejemplo de una red convolucional

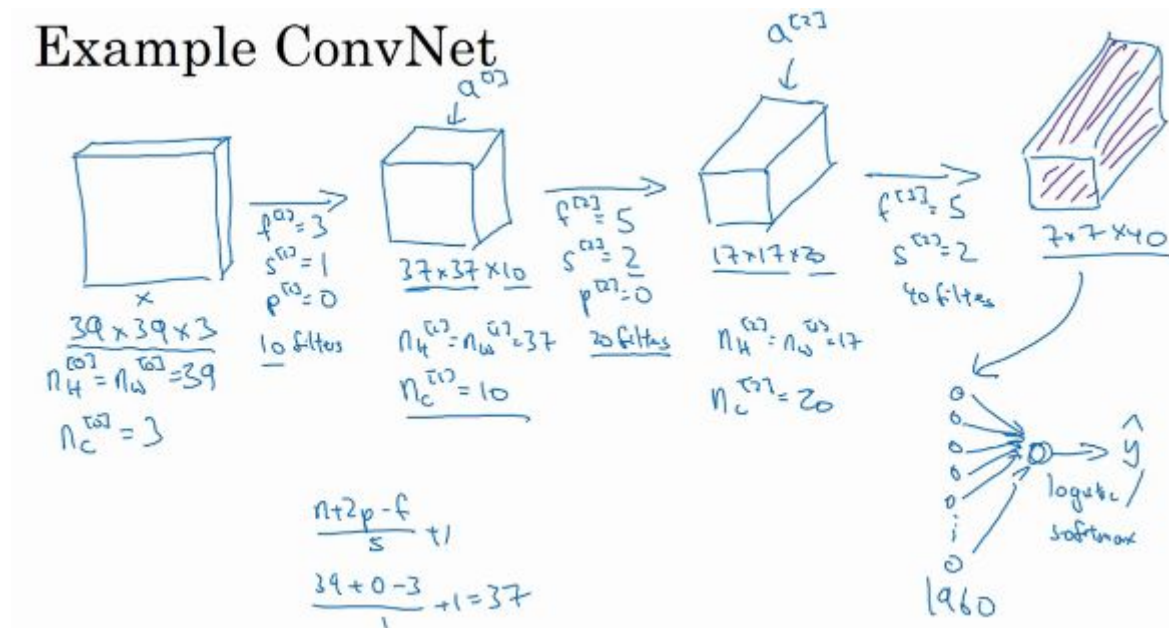
Veamos como aplicar una red convolucional en varias capas de redes neuronales.

Los parametros 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 desafios en CNN es elegir estos hiperparametros.

Example ConvNet



Tipos de capas en redes convolucionales:

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

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

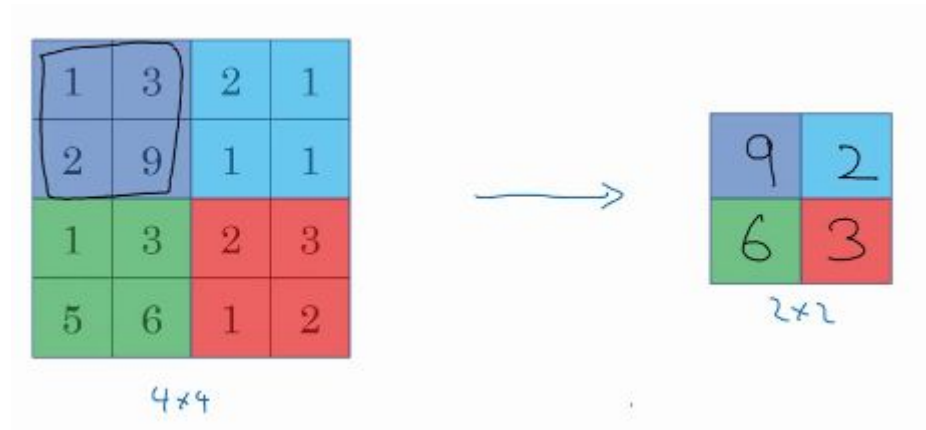
Capas tipo Pooling

Max Pooling

Tomando el maximo de cada cuadrado coloreado.

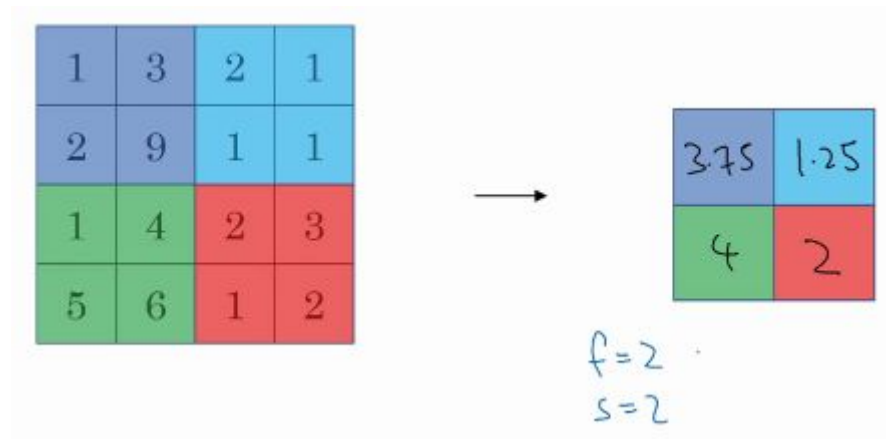
Hiperparametros correspondientes a la tecnica de max pooling.

Hyperparametros:
 $f = 2$
 $s = 2$



Pooling promedio

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



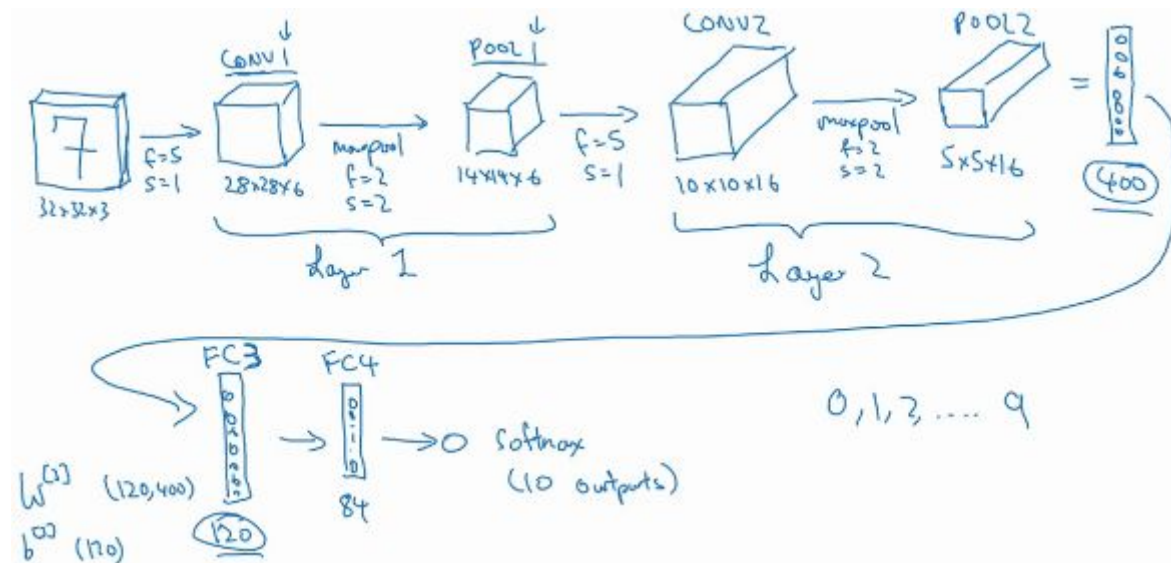
En redes muy profundas suele ser mas comun utilizar esta ultima tecnica antes que el pooling maximo.

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

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

Una convencion es decir que una capa es como la que figura la imagen. Otros definen a los pooling layers como una capa de por si, pero no se suele hacer por el hecho de que no poseen hiperparametros.

La capa fullyconnected 3 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 aqui que los parametros w y b tengan los tamaños que se muestran en la figura.



Características que se notan al realizar ejemplos

$$\begin{matrix} n_H, n_W \downarrow \\ n_C \uparrow \end{matrix}$$

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 hiperparámetros de los pooling layers 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.

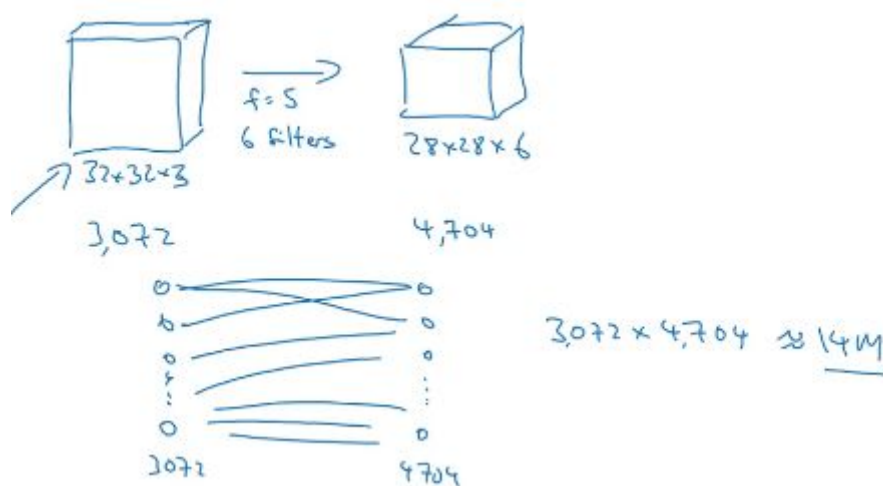
	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{(0)}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 ←
POOL1	(14,14,8)	1,568	0 ←
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 ←
POOL2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Por qué usar capas convolucionales en lugar de únicamente FC?

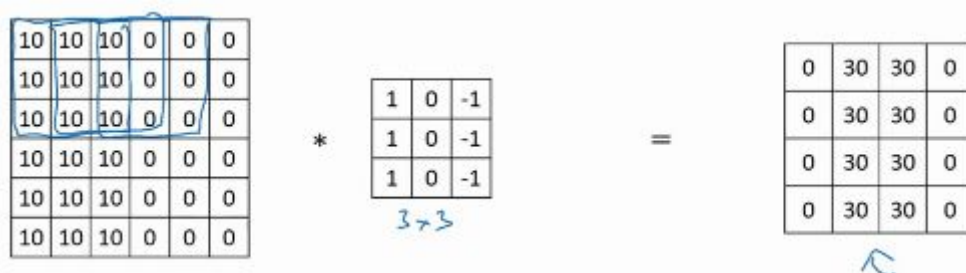
Ventajas por sobre utilizar únicamente las redes totalmente conectadas (FC).

- Parameter sharing
- sparsity of connections

Veamos esto con un ejemplo, como se dijo previamente, conectar estas dos imagenes requeriria entrenar en una capa 14M de parametros. A diferencia en una red convolucional es 156 considerando un filtro de 5 ($5 \times 5 \times 6 = 156$), mucho menor.



Parameter sharing: una característica a detectar sobre que una parte util de una imagen es probablemente util en otra parte del aimagen.

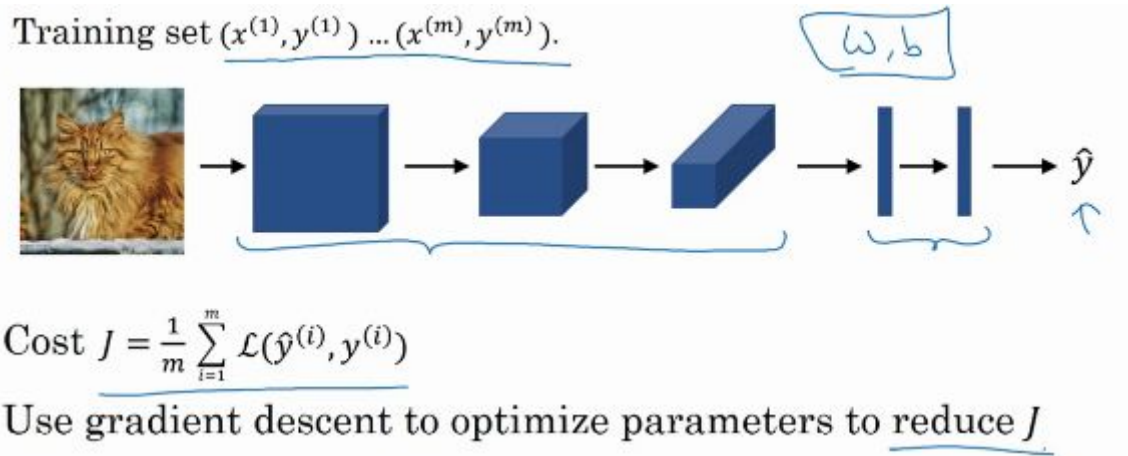


Sparcity of connections: En cada capa, cada valor de salida depende unicamnete de un numero pequeño de entradas. Ej, el valor 0 del extremo izquierdo de la salida depende del cuadro superior izquierdo de 3x3 de la entrada con valor 10s.

Ejemplo entrenando una CNN con fotos de gatos

x: imagenes de felinos

y: etiqueta identificando gatos o no.



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 costo como figura en la ecuación superior y luego se usará gradiente descendiente para optimizar los parámetros y reducir esta función.

Week 2 : Case studies

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 tipo Inception.

Outline

Classic networks:

- LeNet-5 ←
- AlexNet ←
- VGG ←

ResNet (152)

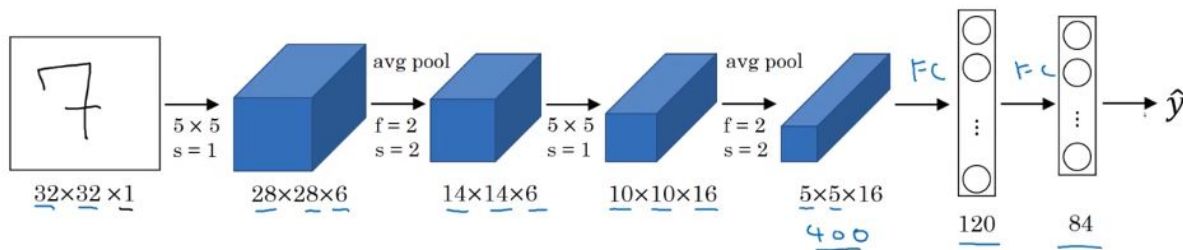
Inception

LeNet-5

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 32x32x1 se procesa mediante 6 filtros de 5x5 con stride = 1 y no padding. Es por ello que la salida es una red de

28x28x6. Luego se aplica a pooling. Avg pooling era utilizado mas en 1998, ahora no tanto. Luego se aplica otra capa de red convolucional y un avg pooling. 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 version moderna usaria softmax al final.



Esta red posee alrededor de 60k parámetros. A día de hoy se encuentran redes con mas 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 mas 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]. 138M de parámetros. Se llama VGG-16 por el número de capas que posee.

Veamos ahora algunas redes más poderosas y mas 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 activacion de una capa y alimentarla en una capa mas 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 funcion de activacion de orden l se pude obtener una de orden $l+2$ pasando por dos capas de la siguiente manera.

Donde primero se aplica una funcion lineal para obtener Z al orden $l+1$. Luego se aplica una funcion no lineal como RELU obteniendo la activacion a al orden $l+1$. De manera análoga se obtiene a al orden $l+2$. A esto se lo llama “main path”.

En redes residuales se hace un cambio, este es tomar a al orden l y sumarlo antes de calcular la función no lineal RELU para obtener a a la $l+2$, como muestra la figura llamado "short cut", skip connection o atajo.

La información de a a la l va más adelante de la red neuronal.

De esta manera la función de activación es calculada como

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 como se comporta una red neuronal plana a medida que el numero de capas aumenta. En teoria el error de entrenamiento deberia decrecer, pero llega un punto en el que aumenta.

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

Por qué funcionan tan bien las ResNets?

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

En la primer figura se considera el caso de dado un input X , una gran red neuronal actua sobre la misma y se obtiene una funcion de activacion a a la l . Si se agrega un bloque residual, se puede ver que la funcion identidad es facil de aprender a pesar de tener mas capas que la red anterior.

Suposicion importante: Tanto z a la $l+2$ como a a la l poseen la misma dimension. Para lograr esto es por ello que se usan same convoluciones. En el caso de que posean diferentes dimensiones, por ejemplo a a la $l+2$ es de dimension 256 y a a la l de dimension 128, se agrega una matriz w que multiplica a esta ultima de dimension 256×128 . Esta puede ser una matriz de parametros o una matriz que implementa zero padding.

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 a la $l+2$ y a a la l . Pero también hay incluidas pooling layers 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 $6 \times 6 \times 1$ multiplicar por un filtro de tamaño $1 \times 1 \times 1$ parece trivial como multiplicar un escalar. Pero si la entrada posee otros valores la cosa cambia. Por ejemplo si es de $6 \times 6 \times 32$, multiplicar por un filtro de $1 \times 1 \times 32$ provoca que se obtenga un resultado de dimension $6 \times 6 \times 32$. 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 la inception network.

Usando una red de 1x1

Cuando 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 Inception networks las cuales veremos a continuación.

Inception network

al momento de elegir una capa CNN se tiene que elegir de qué 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 de inception

Basicamente como se dijo anteriormente, dada una entrada de tamaño $28 \times 28 \times 192$ se aplicara una convolucion de, digamos, 1×1 , el resultado sera de $28 \times 28 \times 64$, pero tal vez querriamos aplicar una de 3×3 con same convolution 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 ultimo si se aplica max-pool 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 numero de filtros es de $28 \times 28 \times 32$ de esta manera el numero de operaciones es 120M -> Operacion muy extensa. Esto se puede reducir en un factor de 10.

Esto se soluciona agregando una red convolucional de 1x1 luego de la entrada antes de aplicar la red de 5x5. El tamaño del input cambia a 28x28x16, siendo este ultimo sobre el que se aplica la red de 5x5, el tamaño de salida sigue siendo el mismo. Esta capa de 1x1 se la 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 $12.4\text{M} \ll 120\text{M}$.

Modulo de inepcion

Haciendo esto mismo para todas las convoluciones se obtiene el siguiente MODULO. Para la capa tipo pooling conviene adjuntarle 32 filtros de tamaño 1x1x32 para que la salida sea de 28x28x32 como queríamos. Concatenando todos estos resultados se obtiene el valor deseado de 28x28x256, habiendo realizado muchas menos operaciones que si se hubieran aplicado las convoluciones de 5x5 o 3x3 directamente a la entrada.

Entonces, qué es una red tipo inception? Es unir estos modulos. Tambien se muestra con un poco de zoom. Puede notarse que el modulo de inepcion que se describio 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 practicos sobre como utilizar redes convolucionales (Conv-Nets)

- Uso de implementaciones en codigo libre

Implementar codigo libre para hacer mas facil la replicacion de resultados.

Tambien se puede obtener codigo 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 parametros de la misma, se pueden obtener los resultados de alguien mas 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 dataset 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 entrenada 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 desacoplar una porción de la red y agregar nuestra propia capa oculta 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 descendiente 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 imagenes otro thread/hilo de la cpu/gpu puede estar generando nuevas imagenes mediante distorsion. El resultado de esto sera un mini-batch del cual se realimentara la red para continuar entrenandola.

Estado de la vision computacional

Datos vs ingenieria dura

Estados de los principales problemas en deep learning en funcion del numero de datos disponibles. Deteccion de objetos posee muchos menos datos que reconocimiento de imagenes debido a que necesita el encuadre de los objetos.

Cuando un problema tiene muchos datos, se suele utilizar menos ingenieria dura, mientras que si se poseen menos datos se utiliza mas (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.

Tips para obtener buenos resultados en benchmarking/ganar competencias

- ensamblado: entrenar 3 a 15 redes neuronales separadamente y promediar su salida. (no sirve promediar sus pesos). 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: Object Detection

Antes de estudiar la deteccion de objetos vamos a estudiar la localizacion de objetos.

Localizacion de objetos

(izq) Como hemos visto en clasificacion de imagenes basta con decir si la imagen pertenece a un auto o no. (centro) Pero en clasificacion con localizacion no solo hay que indicar si hay un auto, sino que hay que encuadrar el auto en cuestion. (der) multiples autos son posibles y hay que identificarlos. No solo autos, sino tambien personas, bicicletas, etc.

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

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

Para localizar un objeto necesitamos mas 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 rectangulo, 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.

La funcion de coste podria definirse a partir de cuadrados minimos como

Tambien se podria haber determinado **log-likelihood** loss para c_1, c_2, c_3 , regression logistica para p_c y SRE para b 's, aunque esta ultima puede funcionar bien en total como se dijo previamente.

Deteccion 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.

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 detecto una cara y luego si limitamos el numero de puntos (cada dos correspondiente a un landmark, x e y) a 64 entonces se tendran 129 filas en total.

Deteccion de objetos

ALgoritmo de deteccion 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

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 metodo es el costo computacional que requiere evaluar, aun utilizando strides grandes. Se podria aumentar el stride para mejorar el costo computacional pero disminuiría la precisión. Afortunadamente esto se puede implemetar 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 5x5. En este caso la salida será de tamaño 1x1x400. A esta ultima se le aplican 400 filtros de 1x1 como muestra la figura.

Cómo implementamos el algoritmo de ventanas deslizantes utilizando CNN? Esto esta 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)

Based on this 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.

La salida de la primera grilla donde no hay objeto se obtendrá un 0 y un Don't care 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, #degrillasx8, 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 #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 rectangulo con el p_c mas grande. Predecir su salida.
- b. Descartar cualquier rectangulo 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 izquierda se puede notar que los puntos medios de los objetos estan practicamente 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.

Todavia 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 automoviles (anchos y bajos). En la practica se suelen elegir entre 5 y 10 tipos de anchor boxes.

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 $2 = \# \text{ anchor boxes}$ (esto suele

ser a elección. Generalmente se eligen 5) y $8=5+\text{\#clases}=\text{\#parametros}$ de objeto localizado. Usualmente el tamaño de la salida suele ser de $19 \times 19 \times 40$ debido al aumento del grillado y el num de anchors a 5. La salida del elemento 1×1 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 3×2 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 rectangulo 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 $100 \times 100 \times 3$ en este caso que pasa por una ConvNet y otorga la salida dicha anteriormente de $3 \times 3 \times 16$.

Predicciones

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

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 suppression para general la predicción final.

Region 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 window 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 sigue siendo lo más lenta. 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.](#)]