Deep Learning Specialization

by DeepLearning.AI

Course #2: Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization

# 

# 

# 

[Course Site](https://www.coursera.org/specializations/deep-learning)

Made By: [Matias Borghi](https://mattborghi.github.io/)

# 

**Table of Contents**

[**Summary**](#_heading=h.88c3te42lqc7) **3**

[**Week 1 Practical aspects of Deep Learning**](#_heading=h.m8hwaj5tii7u) **4**

[Setting up your Machine Learning Application](#_heading=h.th8ek2ws30pt) 4

[Train/dev/tests sets](#_heading=h.gu0x39uxe13) 4

[Mismatched train/test distribution](#_heading=h.y4vpx5bnl3xl) 4

[Bias/Variance](#_heading=h.kv5puwfhuk6y) 5

[Regularización de una red neuronal](#_heading=h.9wq79az5vd37) 7

[Regularización](#_heading=h.9natwj1h8g6g) 7

[Aplicado a Regresión Logística](#_heading=h.8ottnxrw938b) 7

[¿Qué forma toma la regularización para una red neuronal?](#_heading=h.5kzveroa3fya) 8

[¿Por qué la regularización reduce overfitting?](#_heading=h.5n8nermq4wg7) 8

[Dropout regularización](#_heading=h.mc49kxee8zdo) 9

[Otros métodos de regularización](#_heading=h.8lqplvnnd470) 10

[Data Augmentation](#_heading=h.9tr07it69s9p) 10

[Configurando la optimización de problemas](#_heading=h.nxq4j0txfq7s) 11

[Aproximación numérica de gradientes](#_heading=h.flv0w7kduvgu) 12

[Gradient Checking: técnica para verificar la implementación de backpropagation](#_heading=h.1xywhc2l7w2r) 13

[Notas en gradient checking](#_heading=h.y744xiry85ps) 14

[**Week 2: Algoritmos de optimización**](#_heading=h.y4eselzc214a) **15**

[Optimization algorithms](#_heading=h.ojnje9bk243x) 15

[Mini-batch gradiente descendiente](#_heading=h.kka29ne7aqxc) 15

[Función de coste en función del número de iteraciones](#_heading=h.18o8f38qatc1) 16

[¿Qué tamaño se elige entonces?](#_heading=h.pes5zih953eg) 18

[Exponentially weighted averages](#_heading=h.de2lvf7en4sy) 18

[Implementación computacional](#_heading=h.kgln87c438vu) 19

[Corrección del sesgo (bias)](#_heading=h.75dpm3el75iq) 19

[Gradiente descendente con momentum](#_heading=h.hg5k6013trfe) 20

[Otro algoritmo: RMSProp (RootMeanSquareProp)](#_heading=h.j1d3vz8kuxu4) 21

[Algoritmo de optimización de Adam (Adaptive moment estimation)](#_heading=h.k14o54y9sl8w) 21

[Decaimiento del ritmo de aprendizaje (α)](#_heading=h.34wsv8dghcjf) 22

[Definición de epoch y decaimiento del ritmo de aprendizaje](#_heading=h.xpbsbm3wbnvu) 23

[Óptimos locales en redes neuronales](#_heading=h.8xb89d2c55qo) 24

[**Week 3: Hyperparameter tuning, Batch Normalization and Programming Frameworks**](#_heading=h.40eolzfaymm9) **26**

[Hyperparameter tuning](#_heading=h.tllo502g4q3f) 26

[Tuning process](#_heading=h.74uj1nk1z9d3) 26

[Normalización Batch](#_heading=h.s15j094r9lg7) 28

[Normalizando activaciones en una red](#_heading=h.x1yt5mlfq0n0) 28

[Implementar Batch Norm](#_heading=h.1zewyuxis818) 28

[Agregando Batch Norm a una red neuronal](#_heading=h.u4r02vtvare4) 29

[Trabajando con mini-batch](#_heading=h.e49z6a9jc81) 30

[Implementando grandiente descendiente](#_heading=h.t80b27afdtxt) 30

[Resumiendo Batch norm](#_heading=h.h5zbpa6f4nyu) 32

[Multi-class classification](#_heading=h.k6c2qgw0wood) 32

[Regresión Softmax](#_heading=h.3wc12vrsnf0p) 32

[Entrenar un clasificador Softmax](#_heading=h.7lcuta899ag5) 34

[Introduction to programming frameworks](#_heading=h.b8qya1257khl) 34

[Deep Learning Frameworks](#_heading=h.pdzuykxtwf0h) 34

[Tensor Flow](#_heading=h.tgmcgwvmork9) 34

# 

# 

# Summary

**Week 1 Practical aspects of Deep Learning**

**Learning Objectives**

* **Give examples of how different types of initializations can lead to different results**
* **Examine the importance of initialization in complex neural networks**
* **Explain the difference between train/dev/test sets**
* **Diagnose the bias and variance issues in your model**
* **Assess the right time and place for using regularization methods such as dropout or L2 regularization**
* **Explain Vanishing and Exploding gradients and how to deal with them**
* **Use gradient checking to verify the accuracy of your backpropagation implementation**

**Week 2 Optimization algorithms**

**Learning Objectives**

* **Apply optimization methods such as (Stochastic) Gradient Descent, Momentum, RMSProp and Adam**
* **Use random minibatches to accelerate convergence and improve optimization**
* **Describe the benefits of learning rate decay and apply it to your optimization**

**Week 3 Hyperparameter tuning, Batch Normalization and Programming Frameworks**

**Learning Objectives**

* **Master the process of hyperparameter tuning**

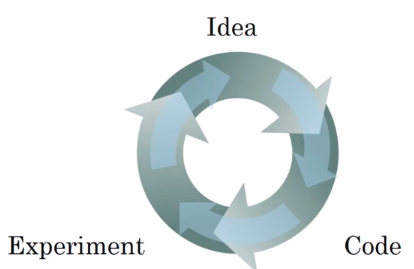
# 

# 

# Week 1 Practical aspects of Deep Learning

## Setting up your Machine Learning Application

### 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.

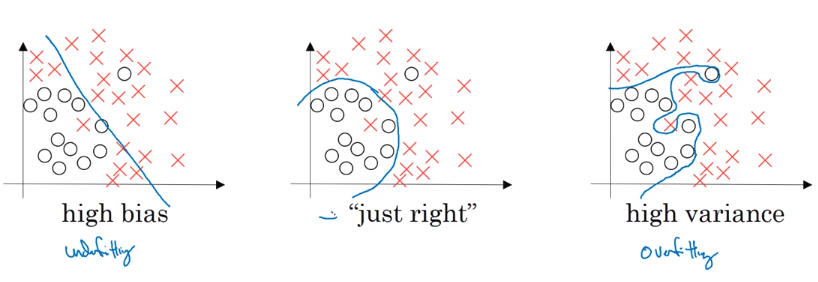
### Mismatched train/test distribution

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



### Bias/Variance

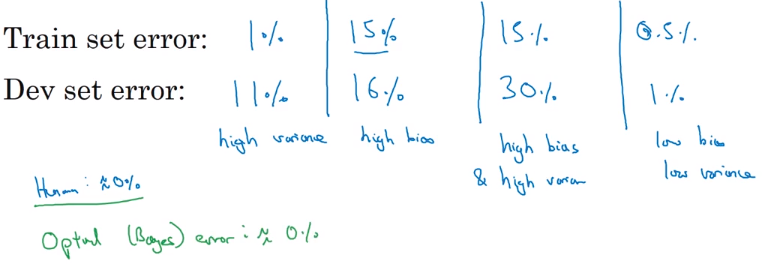
Ejemplo donde se tienen dos features.



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

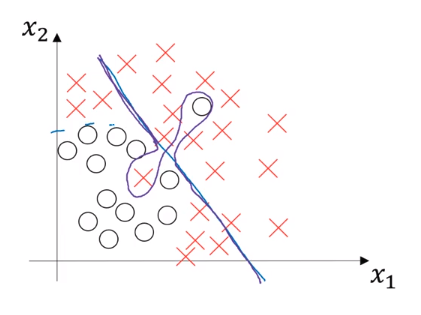
Como ejemplo de clasificación de gatitos, una manera de determinar si se comete algún sesgo o varianza es mirando los errores en el train y dev set.



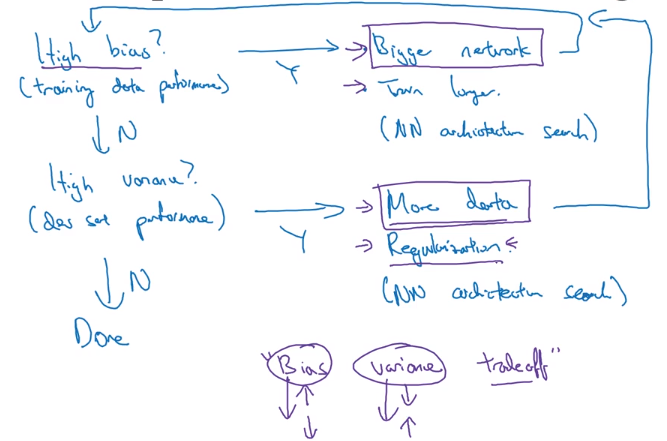


Estas medidas suponen tanto que, como se dijo antes, las distribuciones de los datos provienen de la misma distribución de probabilidad, como que las imágenes no son borrosas para que se considere el error humano como 0%. La falla del primer caso se verá más adelante. En el caso de tener imágenes borrosas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

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



Receta básica para eliminar sesgo y varianza en Machine Learning



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

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

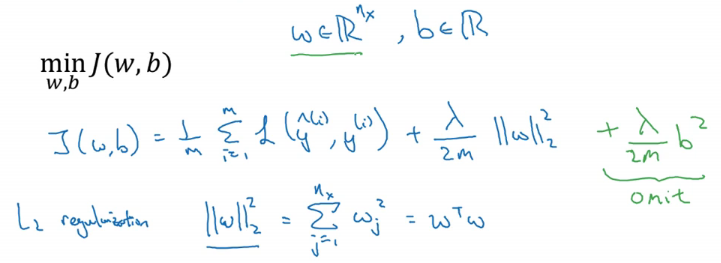
Cuando ambas son negativas se podría afirmar que se terminó.

## Regularización de una red neuronal

### Regularización

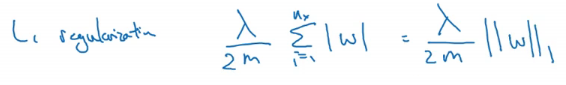
Este método ayuda a prevenir el overfitting y errores de la red neuronal.

#### Aplicado a Regresión Logística

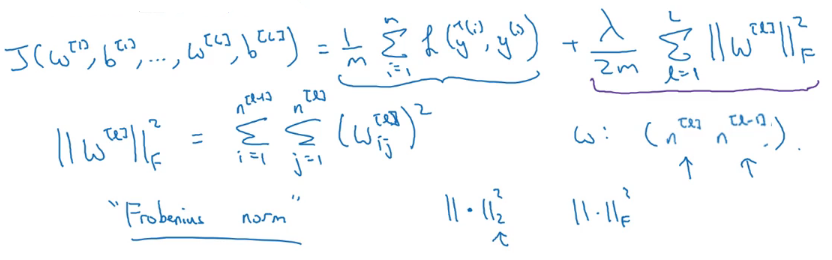


Esto se llama regularización L2, donde lambda es el parámetro de regularización. El último término se omite dado que w suele poseer muchos más parámetros que b (nx >> 1).

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

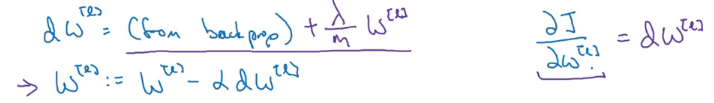


### ¿Qué forma toma la regularización para una red neuronal?

****

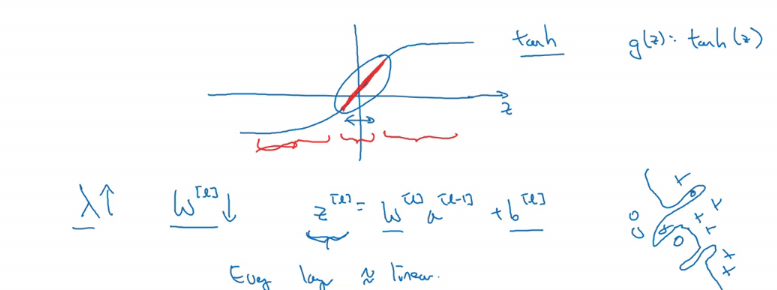
Podemos ver que la expresión es similar pero ahora se toma una norma de Frobenius sobre los parámetros w.

Por otra parte, la backpropagation es de la forma

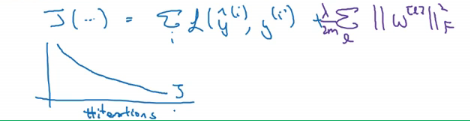


### ¿Por qué la regularización reduce overfitting?

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

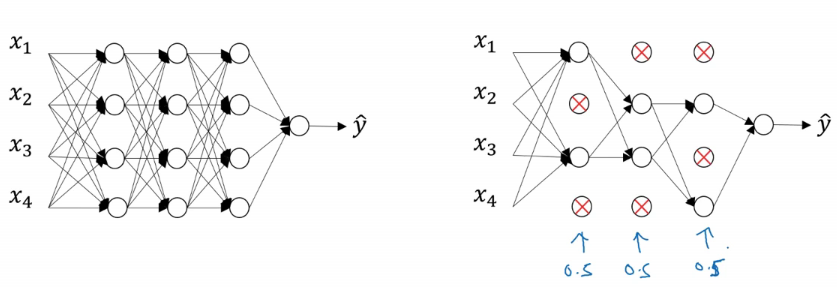


Por otra parte, para probar el correcto funcionamiento de la implementación se puede probar graficando la función de coste en función del número de iteraciones y ver que esta es una función monotónicamente decreciente.

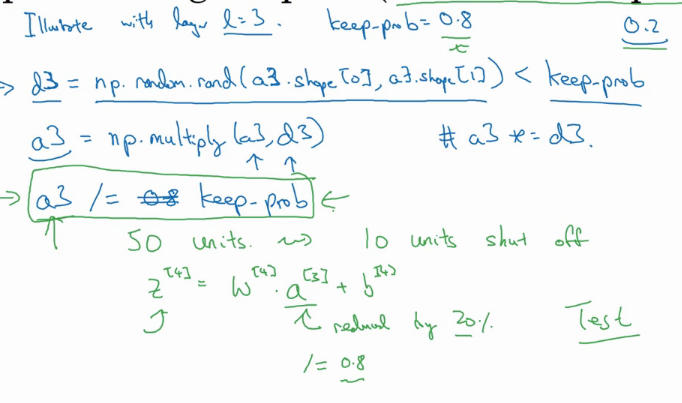


### **Dropout regularización**

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



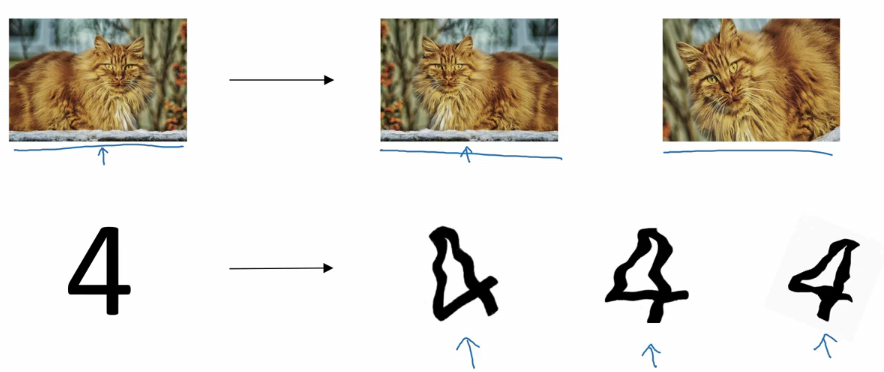
Pero, ¿Cómo se implementa esta regularización? Una técnica se conoce como inverted dropout.



### Otros métodos de regularización

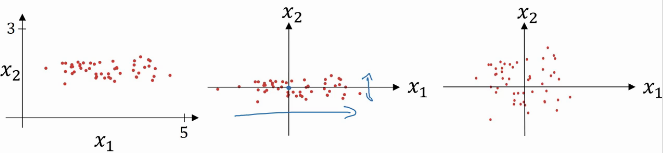
#### Data Augmentation

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



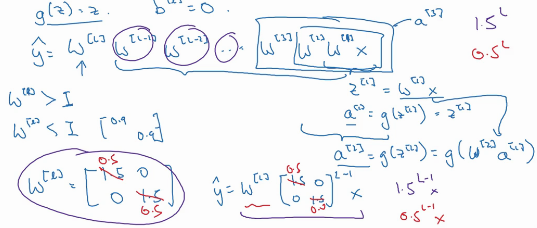
## Configurando la optimización de problemas

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



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

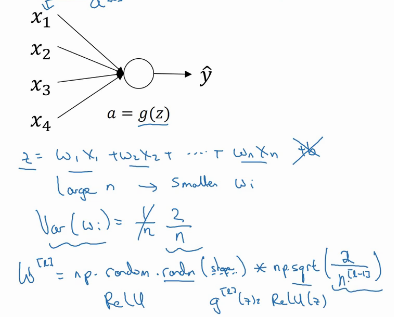




Suponiendo que b=0 y que la función de activación es lineal, se puede demostrar que w puede ser o muy grande o muy pequeño. Esto se puede resolver inicializando los pesos.

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

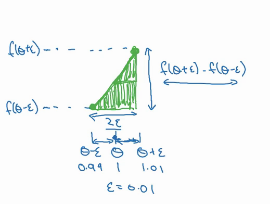
Dado que z es la suma de términos de la forma w\*x, si n aumenta se quiere que w disminuya. ¿De qué manera? Una buena opción es que la varianza de los w vaya como 1/n. Para el caso de las funciones RELU, 2/n funciona mejor. De esta manera se llega a una expresión para w que no es muy grande o muy pequeña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

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



### Aproximación numérica de gradientes

Calculando derivadas a ambos lados.



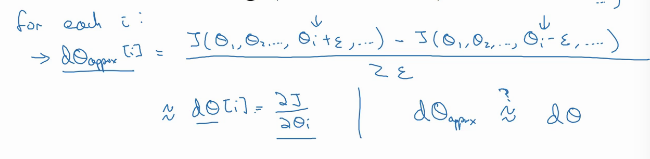
A diferencia de la derivada a un lado, esta posee un error más pequeño, pero se tarda más en calcular su valor.



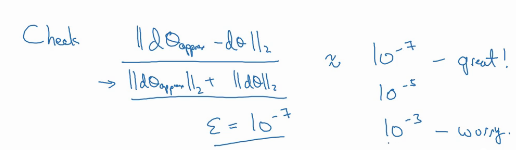
### Gradient Checking: técnica para verificar la implementación de backpropagation

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





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



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

#### Notas en gradient checking

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

# 

# Week 2: Algoritmos de optimización

## Optimization algorithms

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

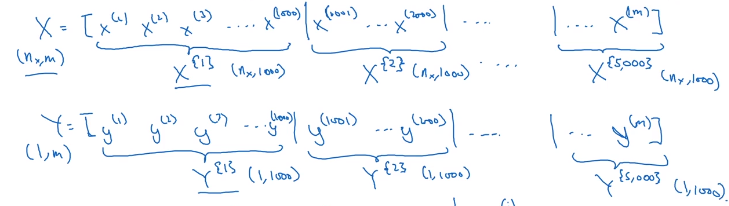
### Mini-batch gradiente descendiente

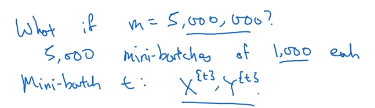
Supongamos que se tiene la siguiente configuración. Para configurar gradiente descendiente se debería procesar todos los features y realizar un paso del gradiente. Luego procesar nuevamente para otro paso y así sucesivamente. En el caso de que m = 5M, esto se vuelve muy lento.



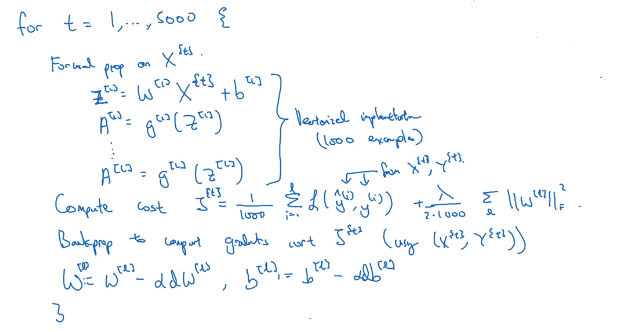
Una solución es usar mini-batch, el cual consiste en separar el training set en por ejemplo dos trainings set.

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



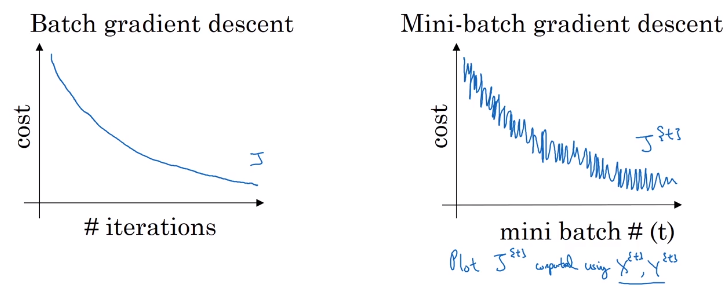


De esta manera la implementación vectorial del algoritmo mini-batch es la siguiente



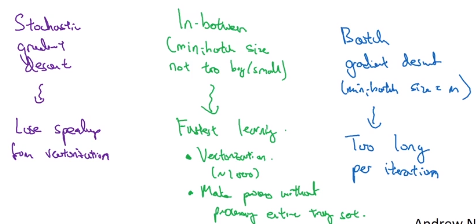
A el loop exterior habría que agregarle otro loop del número de iteraciones.

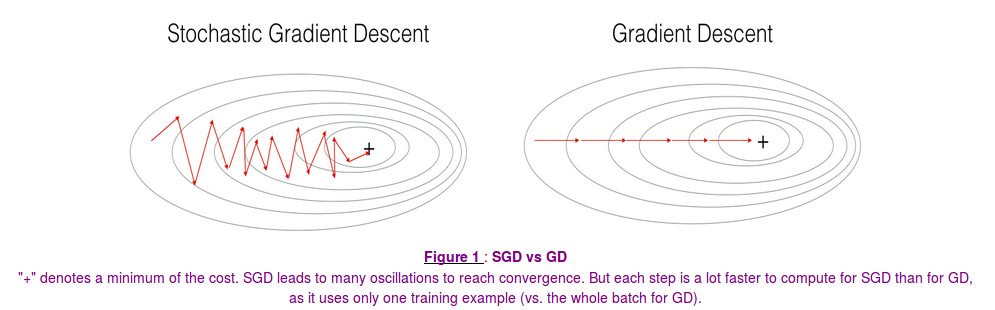
### Función de coste en función del número de iteraciones

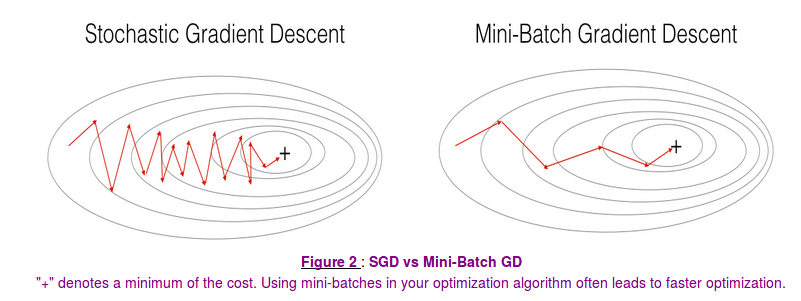


Uno podría preguntarse, ¿qué tamaño del mini-batch elegir y cómo?

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







### ¿Qué tamaño se elige entonces?

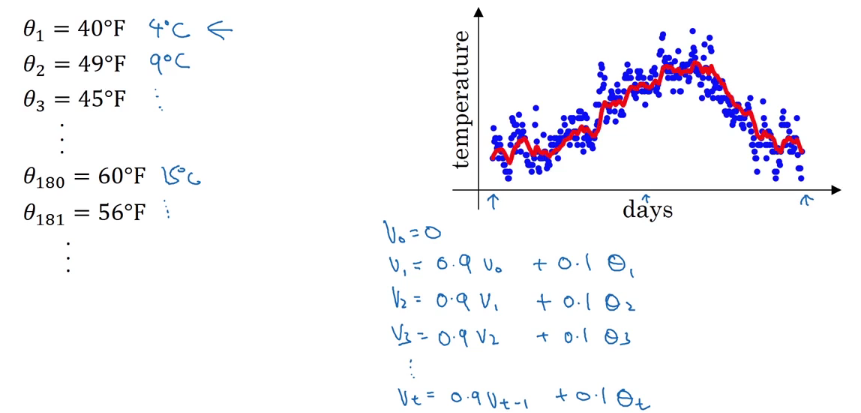
En la práctica este es otro hiper parámetro el cual se debe encontrar.

Si el training set es pequeño (m<=2000) conviene utilizar batch gradiente descendiente. Tamaños usuales para el mini-batch suelen ser potencias de 2: 64(26), 128(27), 256(28), 512(29). Conviene chequear que el tamaño del mini-batch puede almacenarse en la memoria del CPU/GPU.

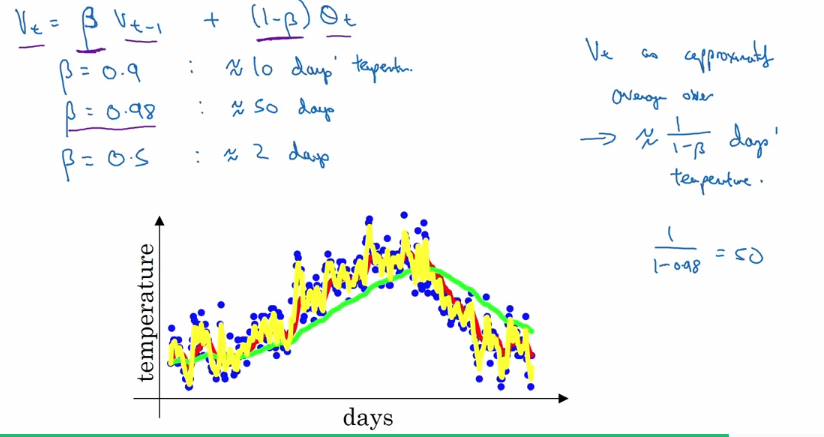
### Exponentially weighted averages

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

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



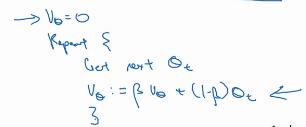
El promedio ponderado se calcula para el primer día V0 = 0. Para el segundo, se supone que hay un peso del 90% respecto del valor del día anterior V0 mas una proporción restante del 10% a determinar por θ1.



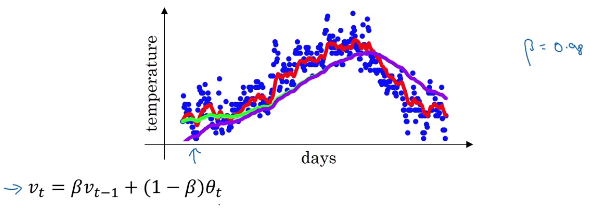
De manera general la ecuación que relaciona la temperatura para un día particular se expresa en la figura superior. Donde los parámetros a determinar son β y θ. Se puede notar que para β más grandes el valor de la temperatura del día anterior es muy importante y los cambios por nuevos puntos no son tan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

La línea roja equivale a β = 0.9. La verde a β = 0.89 y β = 0.5 a la línea amarilla.

### Implementación computacional



### Corrección del sesgo (bias)

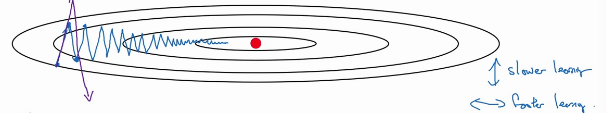


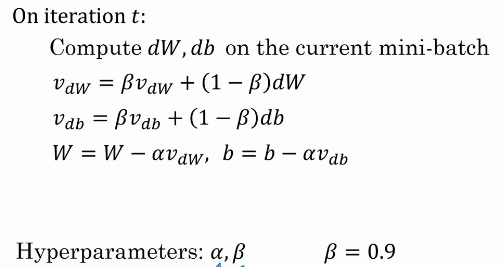
En el caso de β = 0.98, se debería obtener la línea verde, pero usando la ecuación superior se obtiene la púrpura. La corrección del sesgo permite calcular el promedio de manera más precisa. Simplemente consiste en ver que a tiempos pequeños hay un problema debido a que V0 = 0, generalmente no es una buena implementación. Mientras que a tiempos mayores esto se soluciona. De esta manera se aplica una corrección . La misma actúa cuando los tiempos son pequeños (ej., 1-(0.98)2 = 0.0396), mientras que es prácticamente nula cuando t es mayor (ej., 1-(0.98)100 = 1 - 0 = 1).

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

### Gradiente descendente con momentum

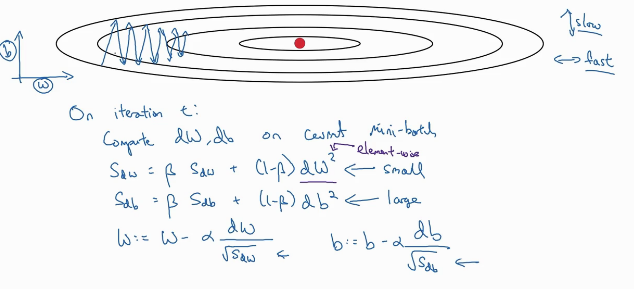
Generalmente funciona más rápido que el algoritmo de gradiente descendente estándar. En una oración, la idea básica es calcular el promedio pesado exponencial de los gradiente para luego usar los gradientes para actualizar los pesos.





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

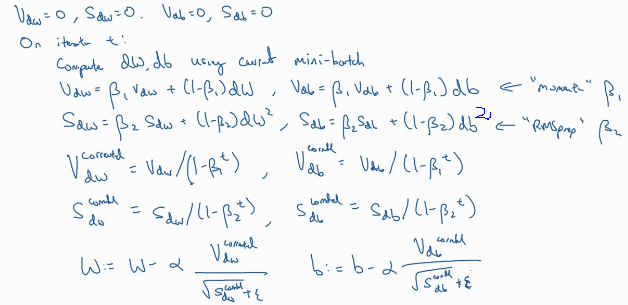
### Otro algoritmo: RMSProp (RootMeanSquareProp)



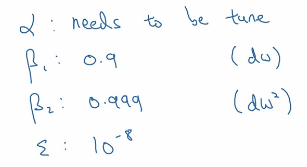
Ahora combinaremos los algoritmos de RMSProp con el Gradiente descendente momentum.

### Algoritmo de optimización de Adam (Adaptive moment estimation)

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



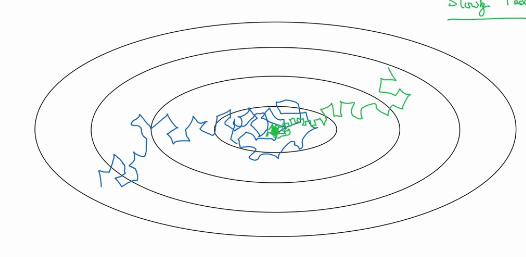
Elección de hiper parámetros: generalmente con la optimización de Adam se definen valores para β y ε y se trata de estimar el mejor valor de α entre un rango de valores.



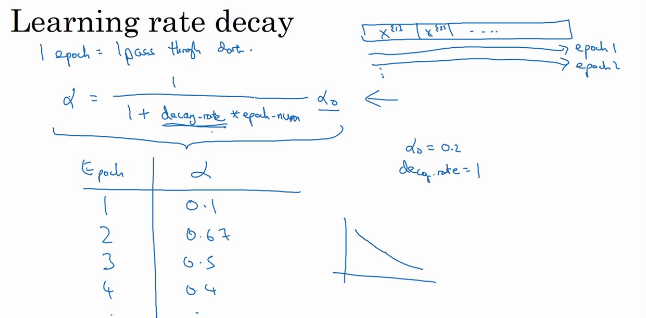
### Decaimiento del ritmo de aprendizaje (α)

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

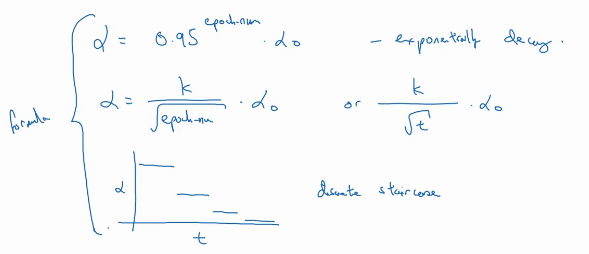
El decaimiento propuesto es del tipo exponencial, inicialmente siendo grande dado que se está lejos de la solución óptima, y a medida que se va acercando α va siendo cada vez más pequeño (línea roja).



### Definición de epoch y decaimiento del ritmo de aprendizaje

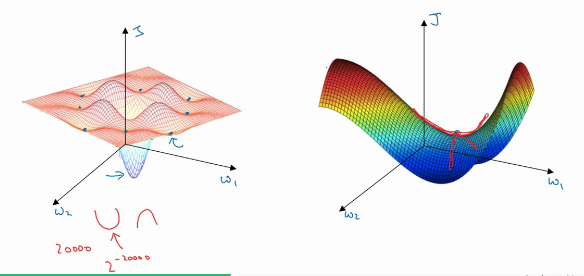


La definición del decaimiento es arbitraria, se podría haber elegido alguna otra como el decaimiento tipo escalera discreto.

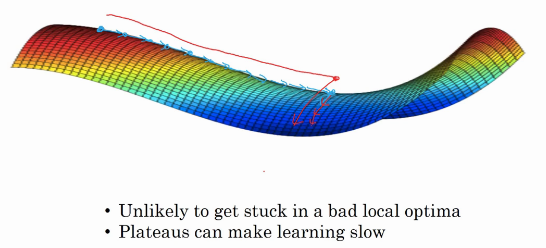


### Óptimos locales en redes neuronales

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



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



# 

# Week 3: Hyperparameter tuning, Batch Normalization and Programming Frameworks

## Hyperparameter tuning

### Tuning process

¿Cómo elegimos adecuadamente al conjunto de hiper parámetros: α, β, (β1, β2, ε) para Adam, número de capas, número de unidades ocultas, decaimiento de la tasa de aprendizaje, tamaño del mini-batch?

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

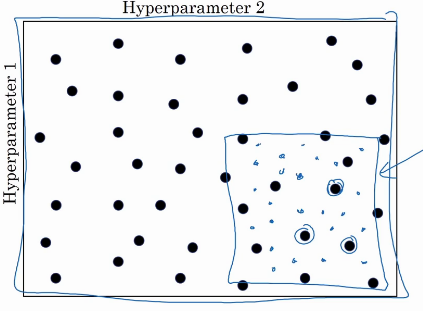
1. Elegir valores aleatorios de hiper parámetros: no usar una grilla!!

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



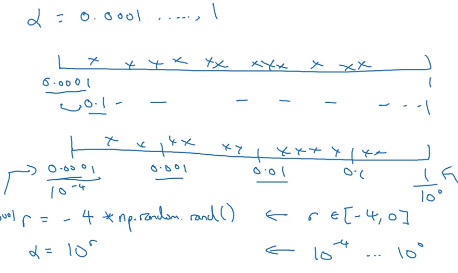
2) Coarse to fine

Enfocarse más en una región particular del espacio de parámetros.



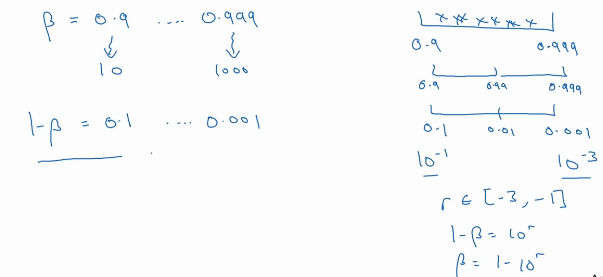
Usar la escala adecuada para elegir hiper parámetros

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



Hiper parámetros para promedios pesados exponencialmente

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



## Normalización Batch

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

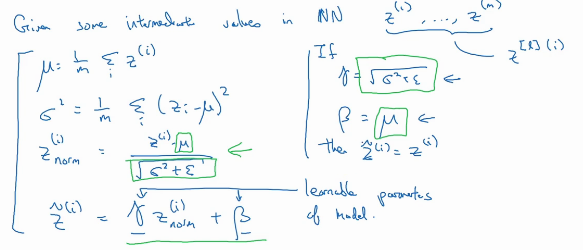
### Normalizando activaciones en una red

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

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

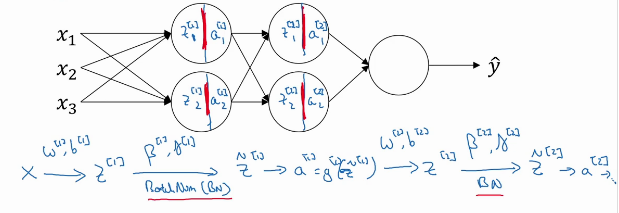
### Implementar Batch Norm

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

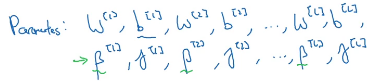


### Agregando Batch Norm a una red neuronal

Primer se comienza calculando z[1] dadas w[1] y b[1], luego se normaliza usando Batch Norm (BN) dados los parámetros β[1] y γ[1]. Obtenida z[1]\_tilde se calcula a[1] evaluando z\_tilde con la función de activación. Esto se realiza para todas las capas ocultas.



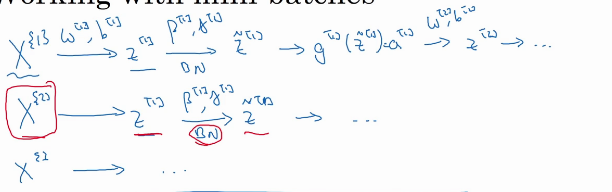
Los parámetros a ajustar serán el doble.



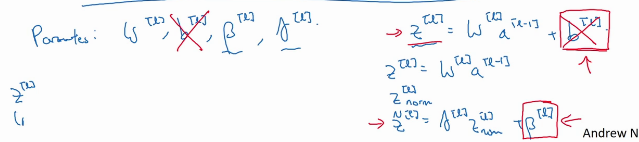
En TensorFlow esto se implementaría con **tf.nn.batch\_normalization.**

### Trabajando con mini-batch

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

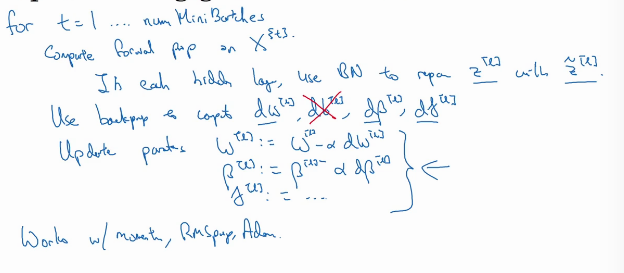


Por otra parte, se puede decir que en realidad el parámetro b es innecesario calcularlo.



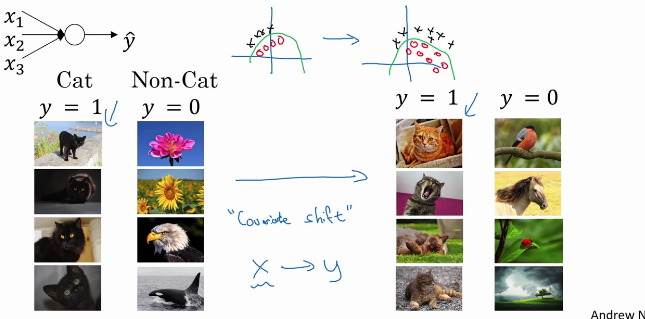
### Implementando grandiente descendiente

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



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

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



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



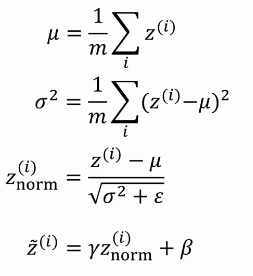
Batch norm aplicado a mini-batch actúa como regularización:

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

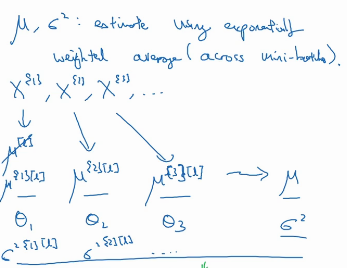
Dado que es pequeño no es un efecto notorio de regularización, para ello conviene implementar dropout.

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

### Resumiendo Batch norm

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

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

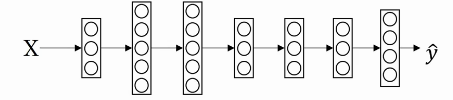
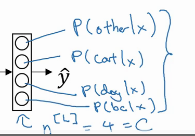


## Multi-class classification

### Regresión Softmax

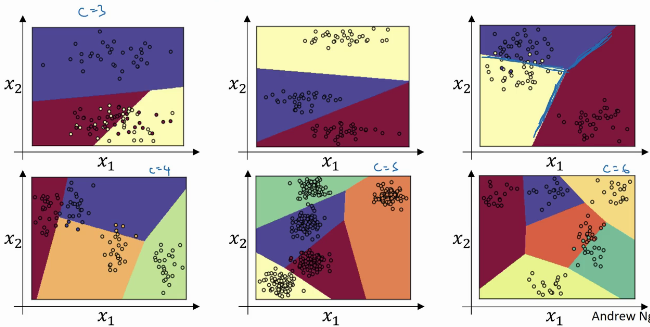
Si en lugar de requerir una clasificación binaria se requiere una clasificación multi conviene usar softmax. Por ejemplo, si se quiere clasificar perros, gatos, pollitos y los que no lo son.



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





Para tener contornos más complejos se deberían agregar capas ocultas.

### Entrenar un clasificador Softmax

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

## Introduction to programming frameworks

### Deep Learning Frameworks

Ejemplos de estos frameworks:

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

Elección:

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

### Tensor Flow

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

