• Deep Learning

• Fundamentos de la IA

DuocUC®

ESCUELA DE
INFORMÁTICA Y
TELECOMUNICACIONES

Regularización

- Regularización
- Inicialización de parámetros
- Batch Normalization

DuocUC®

Regularización – L2 Regularization

La Regularización permite reducir la diferencia entre el error de entrenamiento y el error de generalización.
En rigor, es "obligar" a la red a preferir ciertas funciones dentro del conjunto de funciones que una red tiene dada su capacidad.

El objetivo al entrenar una red es hacer que el error de entrenamiento sea lo más bajo posible y que la diferencia entre este error de entrenamiento y erro de test sea los más pequeño posible.

Ejemplo: "regularización por penalización de tamaño de parámetros" : obligar a que los parámetros sean muy chicos interviniendo la función de error agregando una penalización al final

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \text{error}(\hat{y}^{(i)} - y^{(i)})$$

$$\mathcal{L}' = \frac{1}{n} \left(\sum_{i=1}^{n} \text{error}(\hat{y}^{(i)} - y^{(i)}) + \beta * \text{penalización}(\Theta) \right)$$

Theta representa a todos los parámetros de la red : $\Theta = (W, b, \ldots, U, c)$

Esta nueva función de error depende de los errores que está cometiendo la red con los ejemplos más una penalización sobre los parámetros.
Esto no cambia la capacidad de la red, sino que penaliza los parámetros grandes.

Una de las funciones "penalty" es "L2 Regularization" (En Keras = weight decay): En cada paso del descenso del gradiente achica lo parámetros aplicando penalización logarítmica en base 0,1, 0,01, 0,001, etc.

https://towardsdatascience.com/weight-decay-l2-regularization-90a9e17713cd
https://towardsdatascience.com/batch-normalization-and-dropout-in-neural-networks-explained-with-pytorch-47d7a8459bcd
https://www.deeplearningbook.org/

Regularización – Data Augmentation

Una de las cosas más caras en Deep learning es conseguir más datos etiquetados.

Siempre traer más datos es lo mejor, pero muy caro.

Data augmentation es la técnica de que a partir de los mismos datos se generan más datos.

Ejemplo, si tengo una foto, se podría rotar o cortar en algún lugar (“random crop”).

No existen librerías para realizar estas tareas, básicamente, hay que usar el ingenio para generarlos.

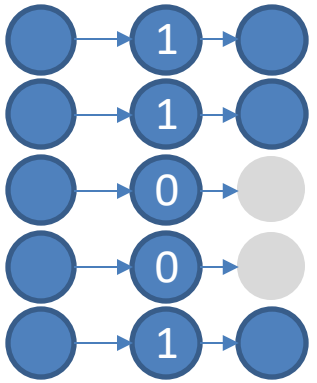
data augmentation es la técnica de a partir de los mismos datos es generar más datos

https://towardsdatascience.com/weight-decay-l2-regularization-90a9e17713cd
https://towardsdatascience.com/batch-normalization-and-dropout-in-neural-networks-explained-with-pytorch-47d7a8459bcd

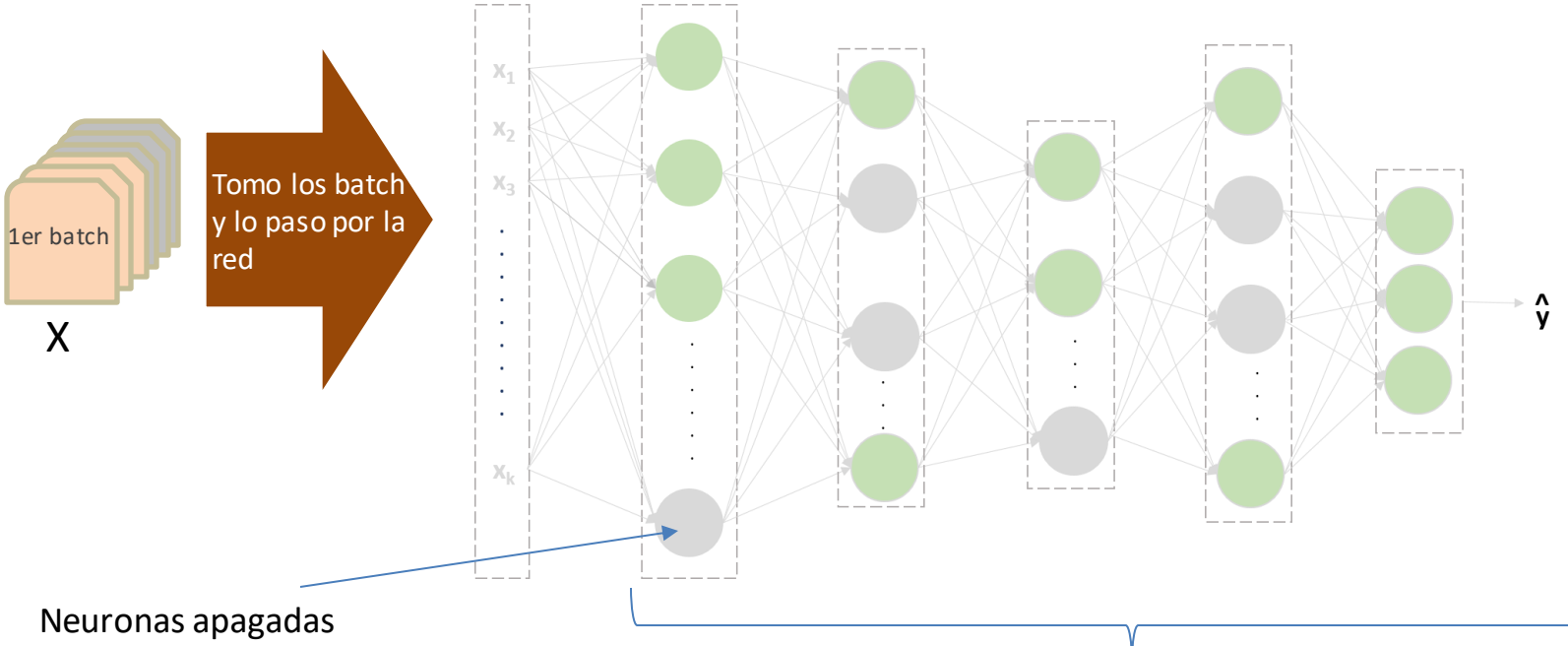
Regularización – DropOut

Al azar se desactivan (apagan) un porcentaje de neuronas en una capa. Según lo publicado en Kaggle.com, partir con un 20% de neuronas apagadas es una buena práctica y se debiera setear este parámetro entre un 20 y 50% SIEMPRE.

El "apagado" aleatorio de las neuronas cambia entre batch y batch que pasa por la red.
Lo que hace la red es agregar una capa y esa nueva capa tiene un 1 o un 0 en cada posición (neurona), al multiplicar por 0, el resultado es 0 obviamente y eso se interpreta como neurona apagada.



Dropout es una manera simple de evitar el overfitting



Implementar Dropout es "apagar" un porcentaje de neuronas (al azar) en las capas de la red antes de pasar un batch

- Antes de pasar el batch, se apaga un % de neuronas al azar en alguna o todas las capas
- Para el siguiente batch, "se encienden" todas las neuronas y se vuelven a apagar otras al azar
- En general, en las últimas capas se tiende a usar % de apagado más bajo (para evitar perder características importantes)

Dropout es SOLO durante el entrenamiento y por ende sobre el dataset de entrenamiento. Para el test y en producción no se usa

Lo que estamos haciendo es hacer más complejo el aprendizaje a la red (la estresamos)

https://towardsdatascience.com/batch-normalization-and-dropout-in-neural-networks-explained-with-pytorch-47d7a8459bcd
White paper written by Geoffrey Hinton and friends (Journal of Machine Learning Research 15 (2014) 1929-1958)
https://keras.io/api/layers/regularization_layers/dropout/
https://www.kaggle.com/pavansanagapati/what-is-dropout-regularization-find-out
https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

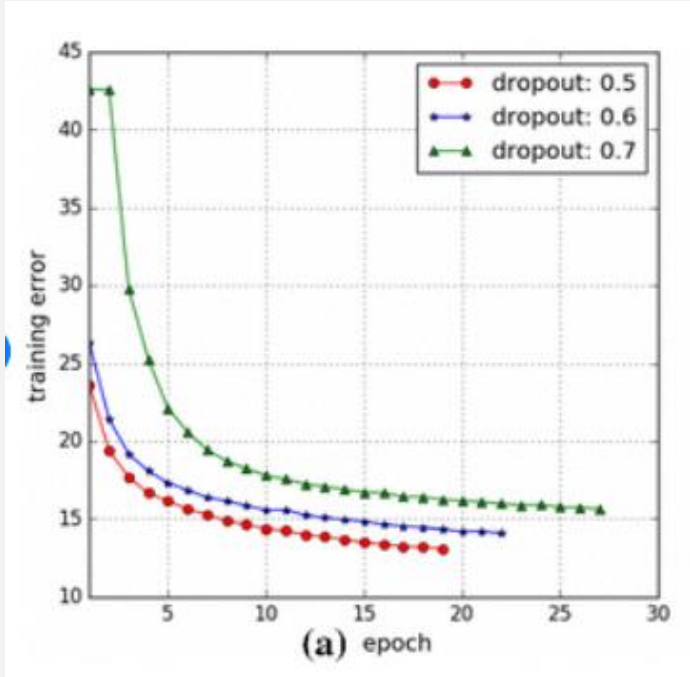
Regularización – DropOut

Durante el entrenamiento se espera que el error de entrenamiento sea mayor

Sin embargo, al usar ejemplos reales (test-producción) la red tiene mejores resultados en la predicción

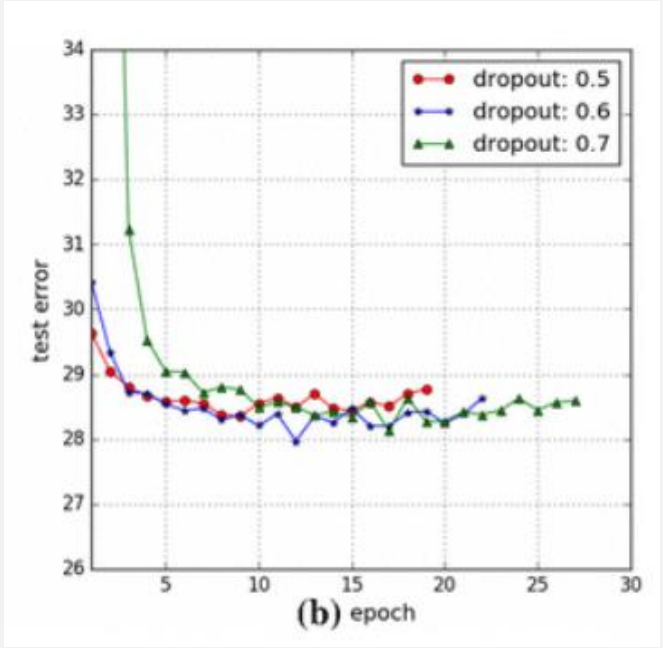
NO EXAHUSTIVO

ILUSTRATIVO



Efecto del Dropout en la Tasa de error durante "**entrenamiento**" con diferentes % de apagado de neuronas

Y su impacto en la Tasa de error de "**test**"



Regularización – DropOut

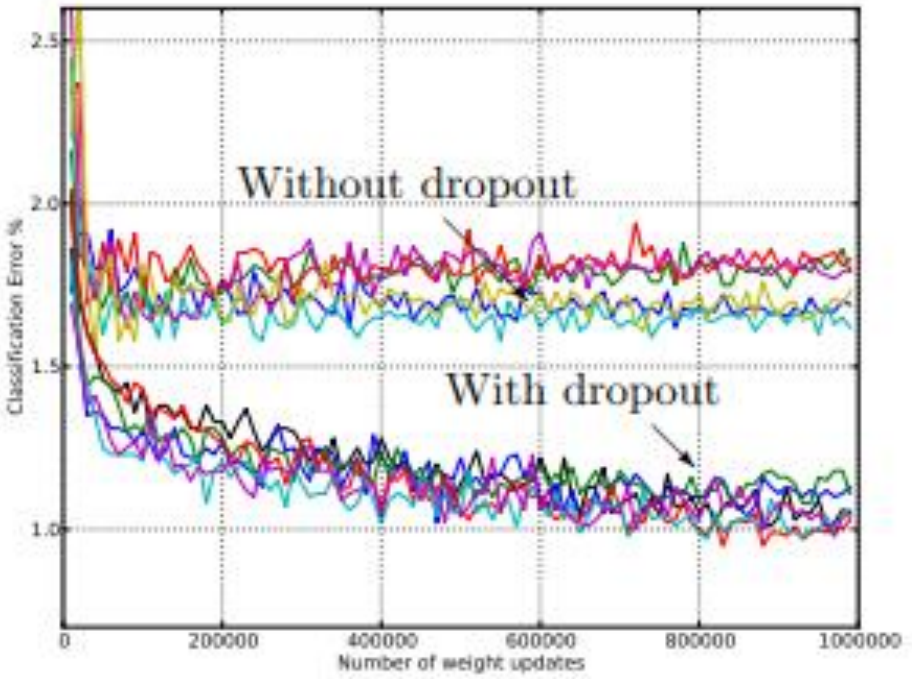
NO EXAHUSTIVO

ILUSTRATIVO

Demostración gráfica de cómo impacta el uso de dropout al error de test de una red neuronal.

Cinco data sets de imágenes (MNIST, SVHN, CIFAR-10, CIFAR-100, ImageNet)

Experimento realizado en el departamento de las ciencias de la computación de la Universidad de Toronto



Inicialización de parámetros

Se entiende que los parámetros (pesos y bias) deben, en general, ser números pequeños y en general rondar en torno al cero.

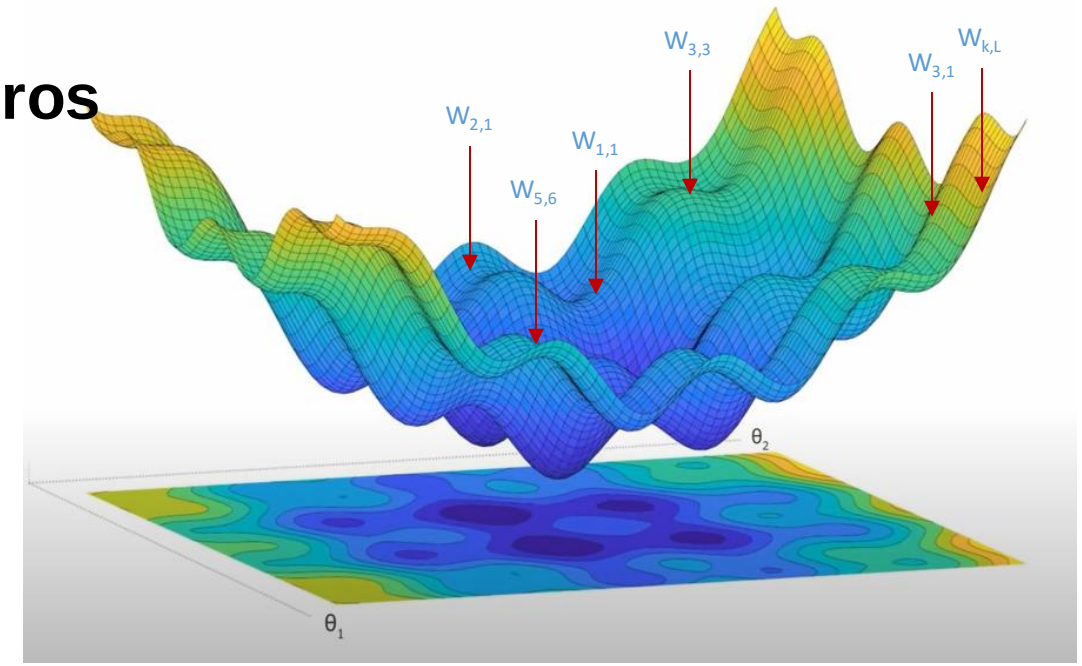
Recordatorio: El descenso del gradiente es el encargado de actualizar los pesos una vez que comienza el entrenamiento.

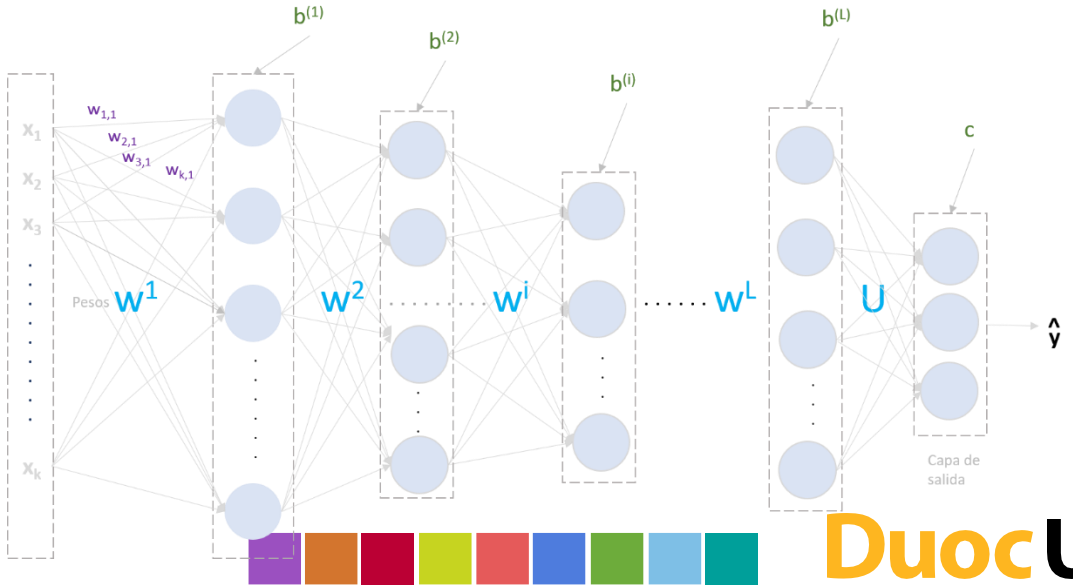
Existe un proceso que, al no ser ejecutado correctamente siguiendo las mejores prácticas, podría generar una arquitectura de red neuronal que finalmente no lograra "aprender" o que demorara muchísimo tiempo en su proceso.

Esta tarea se llama **inicialización de los parámetros:**

- Inicializar o no los bias "b" y "c", o para ser más preciso, inicializarlos todos en cero, no impacta el correcto entrenamiento
- Inicializar los pesos "W", "U" es una tarea crítica que debe abordarse con cuidado. Los pesos no pueden ser inicializados con ceros ya que el descenso del gradiente nunca comenzaría a bajar
- En general, no inicializar correctamente los pesos "W" y "U" podría causar alguno de los siguientes efectos, o mejor dicho defectos:
    - "desvanecimiento del gradiente"
    - "explosión del gradiente"

Inicializar los pesos es indicar al algoritmo del descenso del gradiente el punto de partida para comenzar a descender hasta encontrar todos los parámetros que minimizan la función de error.





https://proceedings.neurips.cc/paper/2018/file/d81f9c1be2e08964bf9f24b15f0e4900-Paper.pdf
https://keras.io/api/layers/initializers/#usage-of-initializers
https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

Inicialización de parámetros – Inicialización Xavier

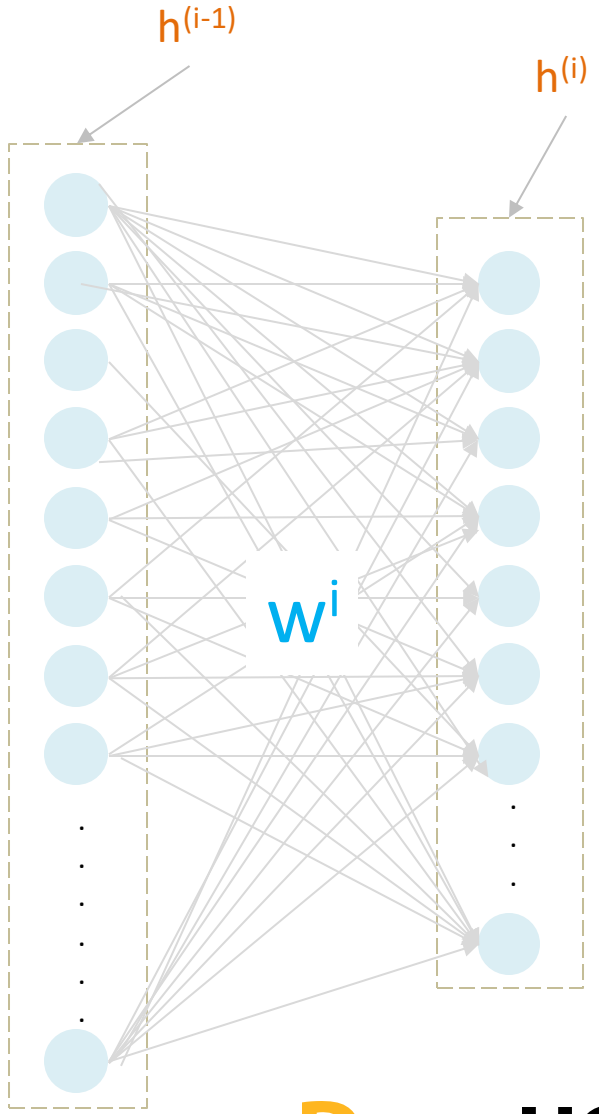
Esta inicialización se utiliza generalmente para sigmoid o tanh como función de activación

Esta técnica, también llamada Glorot Initialization, donde inicializa los bias en cero y los pesos según la siguiente fórmula, en la cual considera la cantidad de neuronas de la capa anterior (i-1) y la capa actual (i)

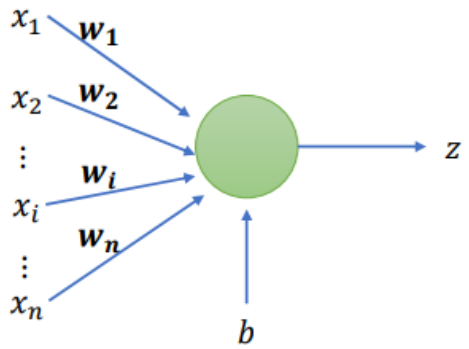
Genera números al azar entre el número de neuronas de la capa anterior y la cantidad de neuronas de la capa actual y los pondera por la raíz cuadrada de uno dividido por el número de neuronas de la capa anterior i menos 1

$$w^i = \text{np.random.randn}(k_i, K_{i-1}) * \sqrt{\frac{1}{K_{i-1}}}$$

Esto asegura que la varianza de todos los pesos es 1



Análisis para una neurona



Asumimos que la entrada tiene *media cero*.
También asumimos que los pesos tienen media cero.
La pregunta es:

Cómo debe ser la varianza de los pesos para preservar la Varianza entre entradas x y salidas z?

$$Var(z) = Var(x)$$

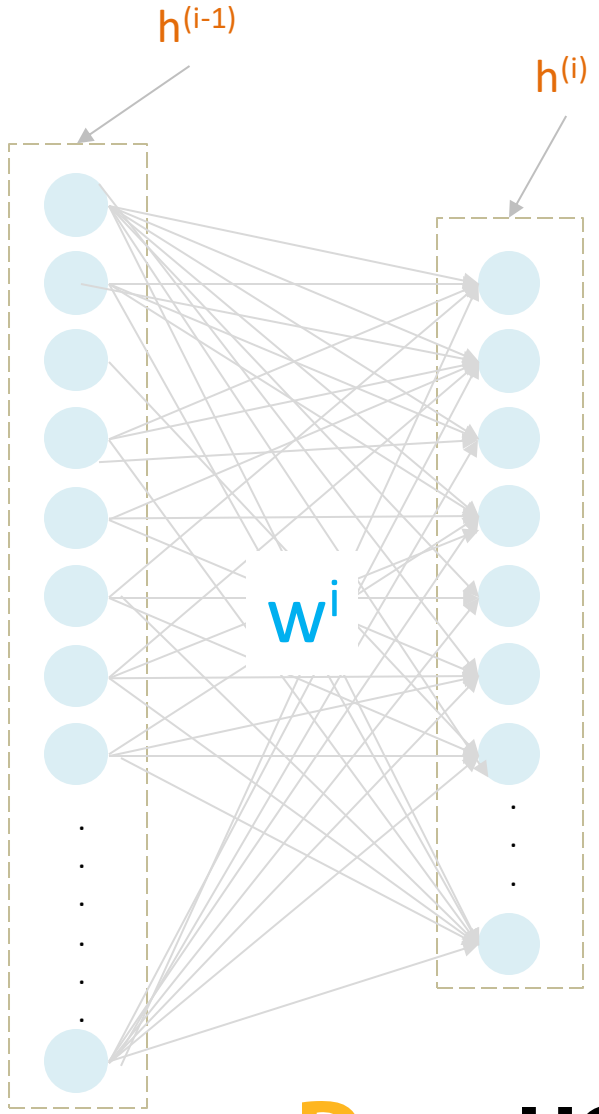
Glorot y Bengio concluyeron que $Var(w) = \frac{1}{n}$. Los pesos deben estar en la distribución $\mathcal{N}(0, {}^{1}/_{\sqrt{n}})$

Inicialización de parámetros – Inicialización He-et-al

Esta inicialización se utiliza generalmente cuando uso ReLu como función de activación

Genera números al azar con distribución normal entre el número de neuronas de la capa anterior y la cantidad de neuronas de la capa actual y los pondera por la raíz cuadrada de dos dividido por el número de neuronas de la capa anterior i menos 1

$$w^i = \text{np.random.randn}(k_i, K_{i-1}) * \sqrt{\frac{2}{K_{i-1}}}$$



Asumimos que los pesos tienen media cero. La pregunta es:

Cómo debe ser la varianza de los pesos para preservar la varianza entre las entradas x y la salida z?

$$Var(z) = Var(x)$$

He et al. concluyó que $Var(w) = \frac{2}{n}$. Los pesos tienen que estar en la distribución $\mathcal{N}(0, \sqrt{2/n})$

Escalando los valores de entrada (X)

Otra buena práctica es inicializar o normalizar los valores de entrada (X) para evitar que alguna dimensión sea interpretada como más o menos importante que la otra.

Podría ocurrir que los datos contengan dimensiones como la edad y el salario de una persona.

Claramente la diferencia de escala, al menos en Chile, sería notoria, si consideramos que una persona adulta promedio puede tener entre 18 y 100 y tener un salario o entrada de $1,000,000

Durante el entrenamiento, teóricamente, podría interpretar que salario es más importante que la edad.

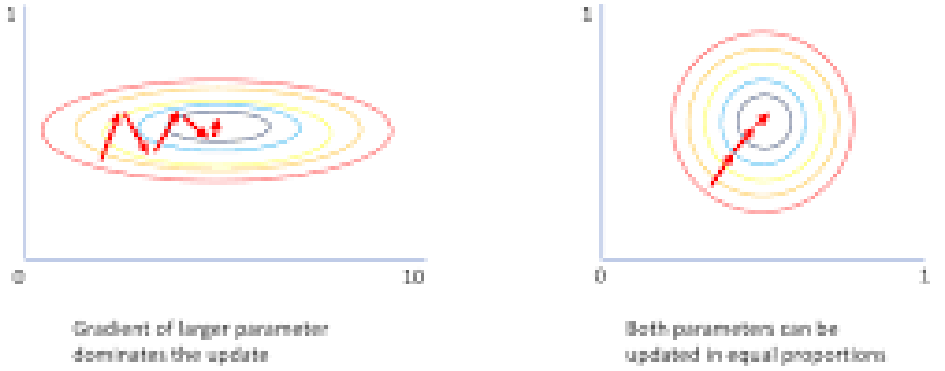
Normalizando las entradas (X), en otras palabras, que todas las dimensiones estén en la misma escala, permite que la función de costo es minimizada de manera más rápida y directa, lo que se traduce en un entrenamiento más efectivo y eficiente

En general, lo que se hace para escalar las entradas es forzar todo a tener una media de 0 y varianza 1.

Otra manera es todo a estar entre 0 y 1, como se hace cuando se trata de imágenes, donde se divide el valor de cada pixel por 255

$$\mathscr{L} = \frac{1}{n} \sum_{i=1}^{n} \text{error}(\hat{y}^{(i)}; y^{(i)})$$

Sin normalización de los inputs la optimización puede demorar en converger al punto mínimo (óptimo)



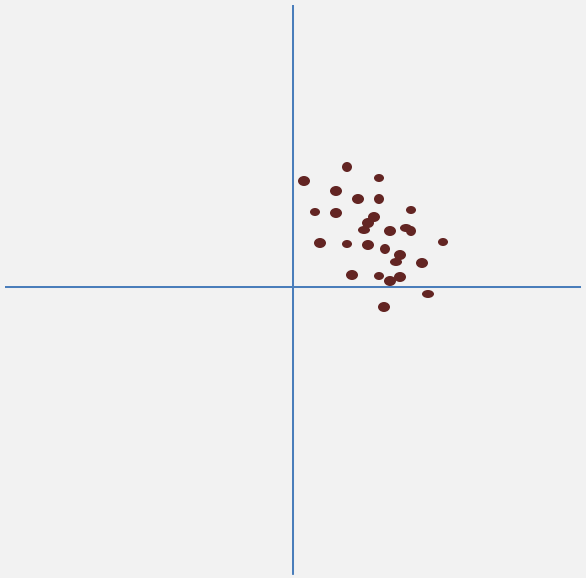
DuocUC®

Escalando los valores de entrada (X)

$$x_{[0,1]} = \frac{x}{max - min}$$

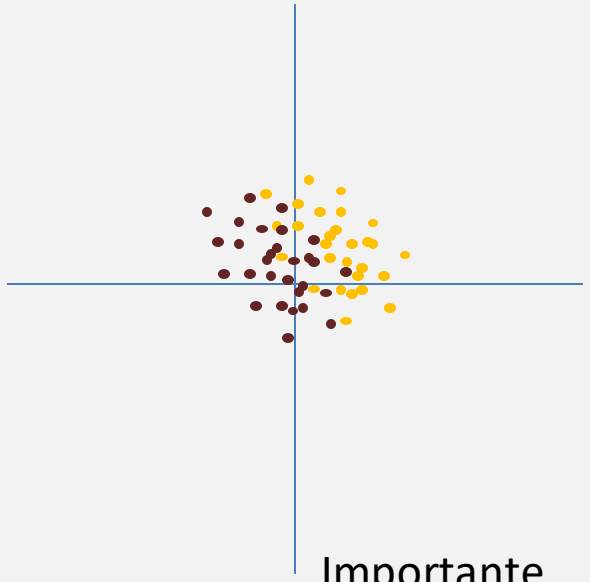
$$z = \frac{x - \mu}{\sigma}$$

X1: Años de antigüedad



X2: Valor de la vivienda

Input escalado (media 0 y varianza 1)



NO EXAHUSTIVO

ILUSTRATIVO

Importante

Escalar las entradas es una tarea “barata” y el beneficio es significativo. Es una tarea de “preprocesamiento” que debería realizar siempre

Batch Normalization

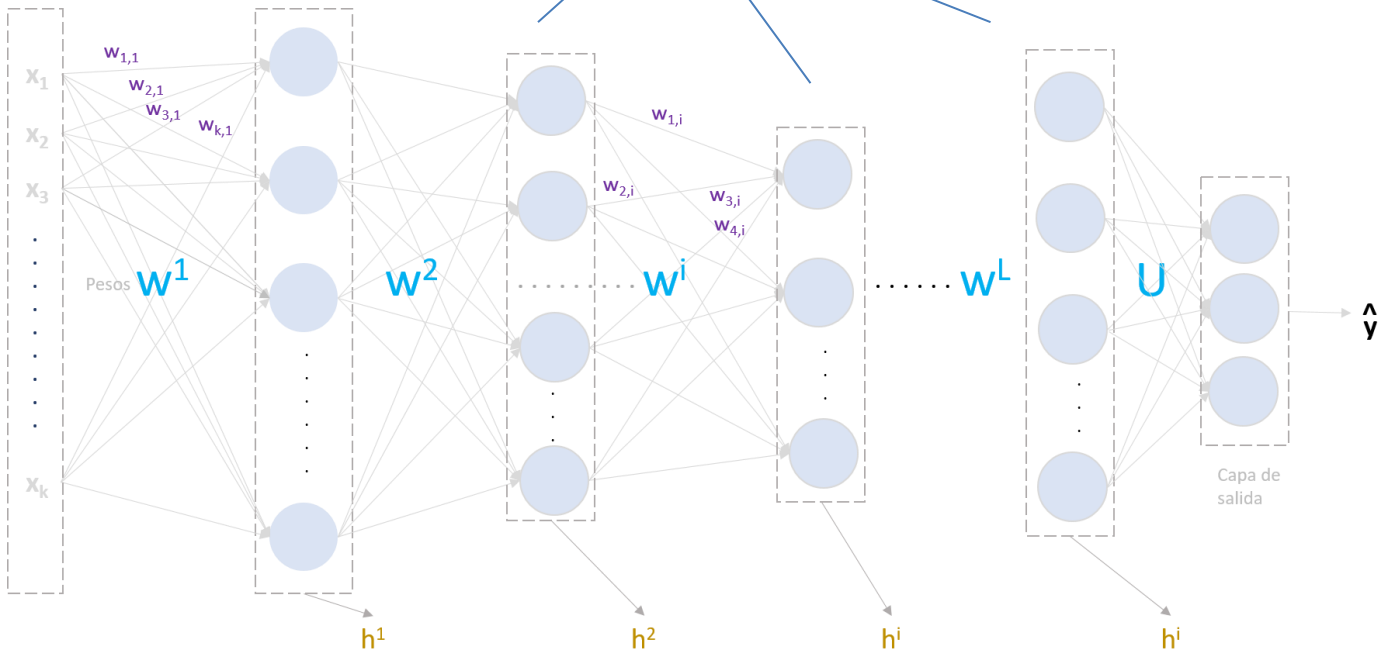
"El entrenamiento de redes neuronales profundas se complica por el hecho de que la distribución de las entradas de cada capa cambia durante el entrenamiento, a medida que cambian los parámetros de las capas anteriores. Esto ralentiza el entrenamiento al requerir tasas de aprendizaje más bajas y una inicialización cuidadosa de los parámetros" (Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.)

Batch normalization consiste en forzar los valores de los pesos de cada capa al rango que necesitamos para que la red siga entrenando de buena manera, esto hace que el tiempo de duración del entrenamiento sea más rápido, también ayuda de cierta manera a regularizar la red y a reducir el error de generalización. Batch normalization tiene un efecto positivo sustancial en la optimización, especialmente en redes convolucionales.

Los valores idealmente se deben inicializar en valores chicos, por ejemplo: media 0 y varianza 1; rangos entre -1 y 1 o rangos entre 0 y 1, etc

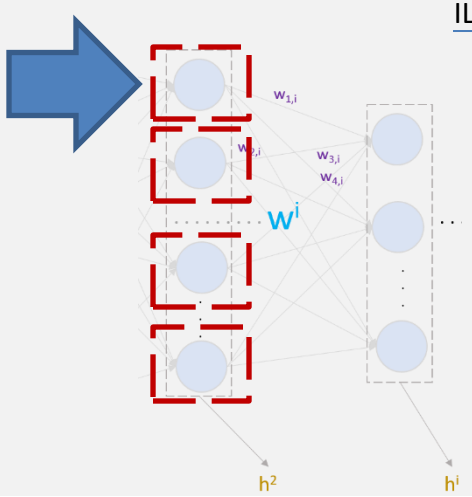
El prefijo "batch" se refiere a que la normalización de una dimensión de los datos, se realiza sobre todos los ejemplos del batch que en ese momento se está evaluando en la capa oculta h sub i

En la medida que entreno la red y profundizo en las diferentes capas, voy perdiendo el control de los valores y escalas que van adquiriendo los valores en las capa ocultas



Todos los ejemplos del batch ingresan a las neuronas de manera paralela

ILUSTRATIVO



Importante

Solo se normaliza en las capas escondidas

Batch Normalization

Batch normalization agrega una nueva capa, después de la capa oculta h supra i

Esta normalización, se realiza de manera estándar:

Se aplica la media y la varianza de la dimensión de todos los ejemplos del batch que está pasando por la red y ese valor lo deja en la neurona respectiva de la capa de normalización

Nótese que ahora, la entrada de la capa h supra i+1 es la capa normalizada h' supra i