Inteligencia Artificial para profesionales TIC

Optimización y regularización en DL

valgrai.eu

Valencian Graduate School and Research Network of Artificial Intelligence

Índice

1. Introducción

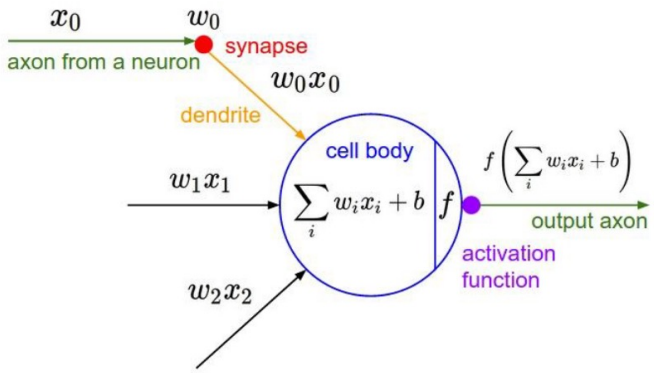
Bueno, entrenar a una NN no es fácil

- Como sabemos, entrenar una NN es un problema de optimización complejo con un alto número de parámetros, donde encontrar una buena solución es complejo
- Las NN dependen de un gran número de **parámetros** e **hiperparámetros**, donde es demasiado fácil cometer errores y encontrar buenas estrategias (una de las razones por las que el aprendizaje profundo requirió muchos años de investigación antes de proporcionar buenos resultados)
- Veremos aquí algunos elementos que nos ayudarán a entrenar mejor un red, con mejores resultados y de una manera más eficiente
- Muchas de las técnicas, aunque con un trasfondo matemático, son muchas veces el resultado de la intuición y los trucos de los investigadores

2. Funciones de activación

No linealidades

- Uno de los elementos más importantes en una NN es la selección de la **función de activación** o no linealidad
- El clásico ha sido el sigmoide y, aunque podemos usarlo en algunos casos particulares (como en la última neurona de la capa de salida en clasificación binaria), tenemos mejores alternativas
- Estas mejores alternativas son una de las razones de los grandes avances en el aprendizaje profundo

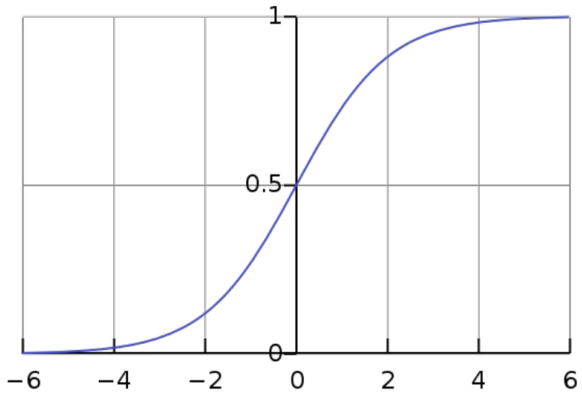


2. Funciones de activación

Sigmoide

La función sigmoide es así:

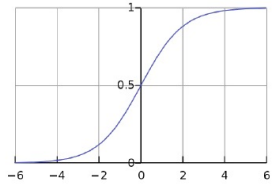
$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$



- Sigmoid es capaz de convertir cualquier número real en un rango entre 0 y 1 (positivos hacia 1, negativos hacia 0)

2. Funciones de activación

Sigmoide



No se usa comúnmente debido a varias razones:

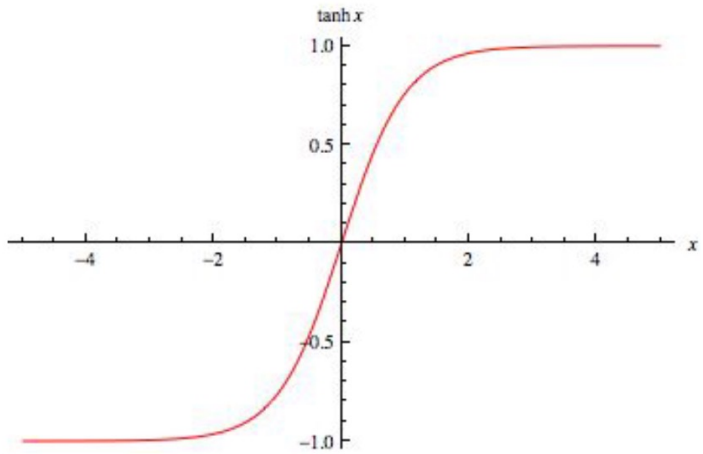
- Las unidades sigmoid saturadas "**matan**" los gradientes
- Una unidad saturada es cuando sus valores se acercan a 0 o 1, donde los cambios en la variable de entrada x casi no cambian su salida y
- Cuando la función está cerca de 0 o 1, como podemos ver en su gráfica, la derivada es casi 0, por lo que este va a ser el efecto en el algoritmo de backpropagation cuando la unidad sigmoide esté trabajando saturada: el gradiente retropropagado se matará (cerca de 0), haciendo que el proceso de aprendizaje a veces sea imposible
- Otro problema con sigmoide, es que la salida **no está centrada en 0**, dándonos valores siempre positivos generando algunas ineficiencias en el entrenamiento
- Además, debemos calcular su derivada (e(x)), lo que introduce otra desventaja en NN con un gran número de neuronas

2. Funciones de activación

Tangente hiperbólica

- Similar a sigmoide pero ahora capaz de producir valores de -1 a 1
- Todavía tiene el efecto de **'eliminación de gradiente'**, pero al menos se centra en 0
- En muchas aplicaciones, mejor opción para sigmoide

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

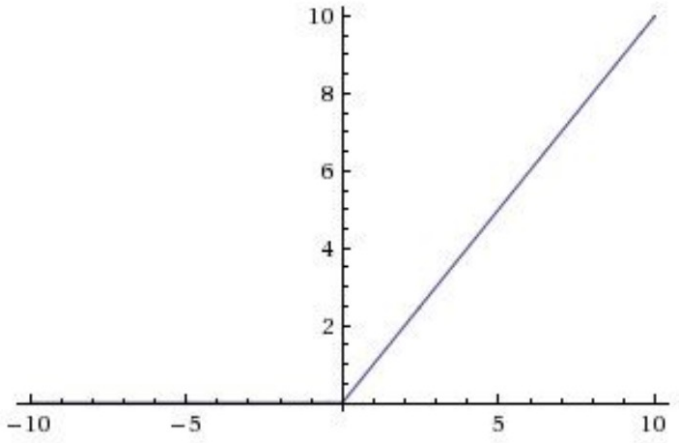


2. Funciones de activación

ReLU (rectified linear unit)

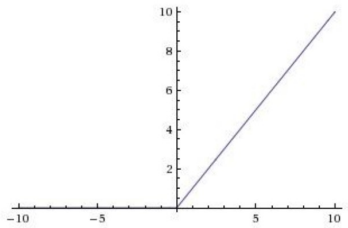
- La salida es 0 para x < 0 y la identidad (lineal) para x > 0
- Es, por muchas razones, **la más utilizada actualmente**

$$f(x) = x^+ = \max(0, x)$$



2. Funciones de activación

ReLU (rectified linear unit)



- **ReLU tiene muchas ventajas:**
- No satura con valores positivos (x > 0), no mata gradientes positivos
- Computacionalmente muy eficiente
- Acelera la convergencia de SGD

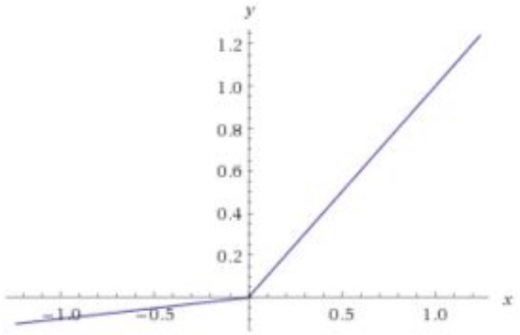
- **Aunque....**
- La salida, como sigmoide, no está centrada en 0
- Las ReLU son frágiles y pueden morir durante el entrenamiento:
  - Un cambio muy grande puede hacer que la unidad nunca vuelva a estar activa, es decir, que nunca produzca valores positivos
  - En estos casos, el gradiente de la neurona es siempre 0, considerando a la neurona como muerta
  - Esto sucede a menudo (ReLUs muertos), siendo interesante a veces controlar es que tenemos muchos es esta situación
  - El uso de pequeñas tasas de aprendizaje (learning rate) puede evitar este problema

2. Funciones de activación

Variantes ReLU

- **Leaky ReLU**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

PReLU

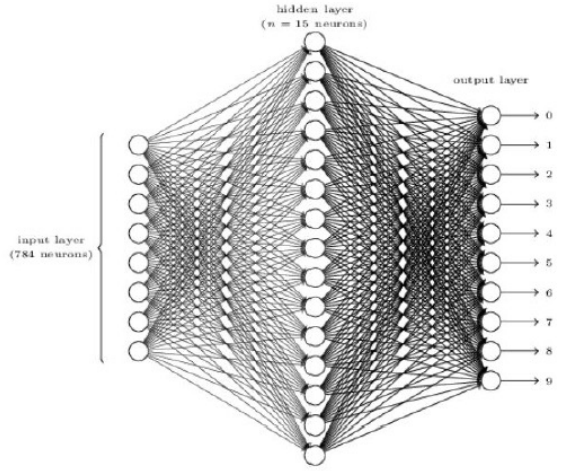
$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{otherwise} \end{cases}$$

ELU

3. Inicialización de parámetros

La asimetría es imprescindible

- La inicialización de **W** y B es fundamental
- **¿Qué sucede si todo se inicializa en 0 o en un valor idéntico?**
- Todas las neuronas producirán el mismo valor, todos los gradientes con respecto a los parámetros serán los mismos.... Por lo tanto, la red nunca aprenderá nada
- Es obligatorio **romper la simetría de los parámetros** (como resultado de esta simetría, diferentes neuronas se especializarán en diferentes características; aprenderán diferentes características como técnica de aprendizaje de representación)



3. Inicialización de parámetros

La asimetría es imprescindible

- Es necesaria la inicialización aleatoria, utilizando desde una distribución normal hasta métodos más específicos (Xavier-Glorot, He), etc.
- Esto es esencial con las ponderaciones pero no crítico para los bias, siendo común inicializarse a 0.

4. Optimización

Más allá de SGD

- Entrenar una NN es un problema de optimización, como sabemos, SGD:

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

- Para un lote de m elementos:

$$w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

4. Optimización

Más allá de SGD

```
for epoch=1 to num_epochs:
    while there are training examples left to be considered in the epoch:
        1. choose a batch of m elements not used in the epoch
        2. calculate gradients of the parameters and apply SGD (update rule)
```

Backpropagation algorithm

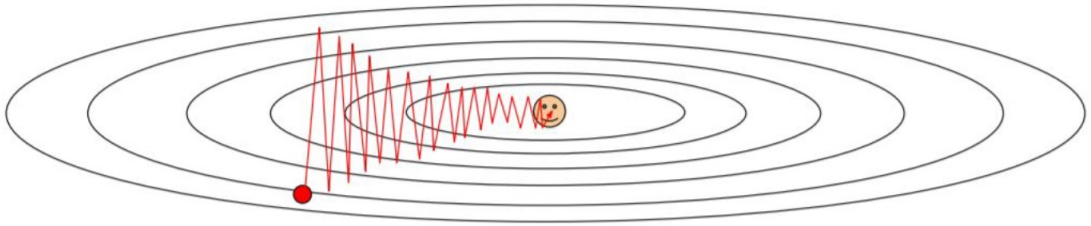
$$w'_k = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b'_l = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

4. Optimización

Más allá de SGD

- Aunque SGD es una buena estrategia, a veces tiene problemas.
- **Zig-zags ineficientes**, cuando la pérdida se reduce muy rápido en una dirección y muy lenta en otras

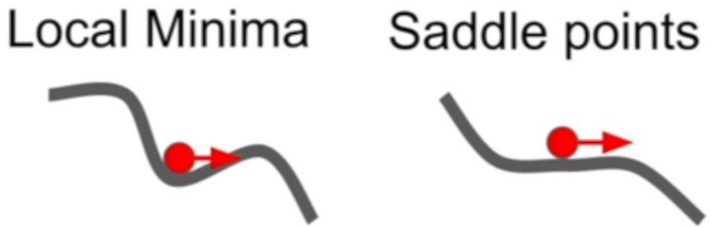


- Mínimos locales y puntos de silla de montar (derivadas iguales a cero), donde tal vez podamos hacer algo mejor

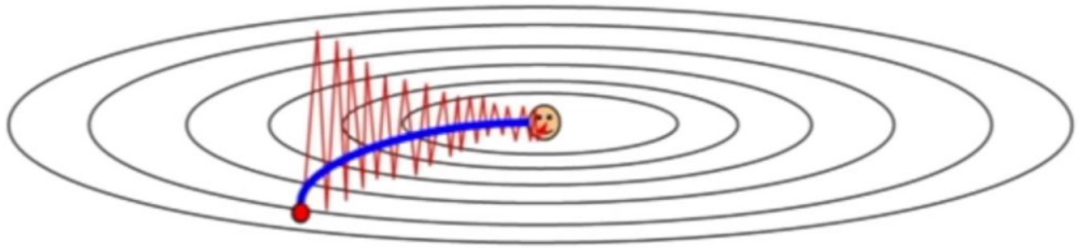
4. Optimización

Momentum

- **SGD con momentum**, **Nesterov Momentum**, y otros enfoques similares, añaden algún **tipo de velocidad.**
- La idea es que el proceso de descenso vaya consiguiendo velocidad en las direcciones correctas hacia el mínimo en todas las actualizaciones, acelerando la convergencia
- La velocidad acumulada nos ayuda a pasar por los puntos de gradiente 0



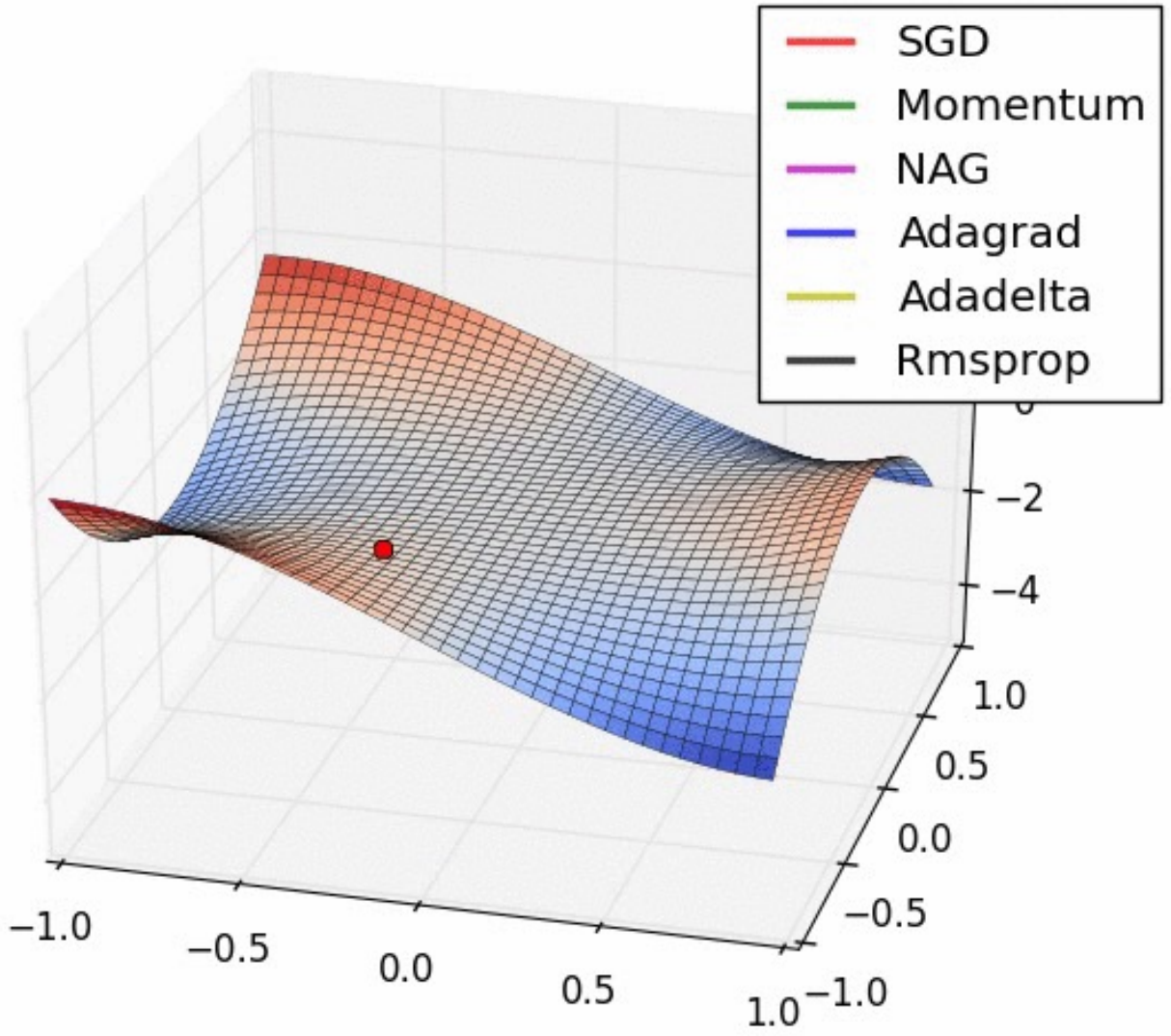
- Y minimiza los problemas en zig-zag:



4. Optimización

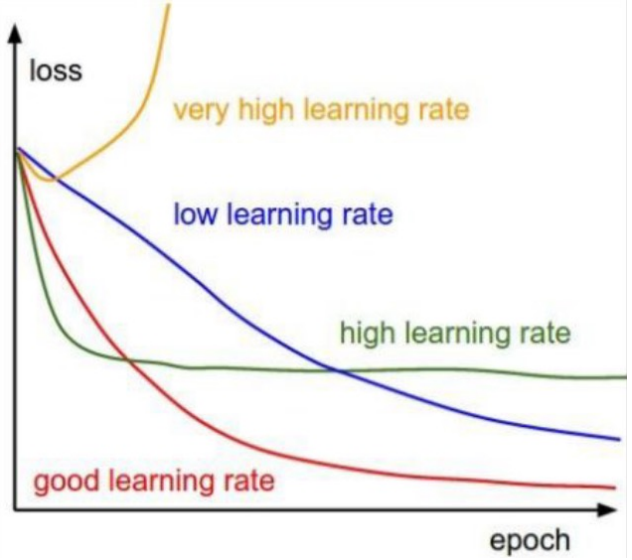
Tasa de aprendizaje adaptativo

- Los **métodos de tasa de aprendizaje adaptativo** tratan de modificar la **tasa de aprendizaje para cada parámetro**, con la idea de que los parámetros que se cambian con más frecuencia, lo hacen con pequeños cambios, mientras que aquellos con menos actualizaciones podrían tener una mayor tasa de aprendizaje
- **Adagrad**: divide la tasa de aprendizaje de cada parámetro por una suma acumulando el cuadrado de los gradientes; De esta manera, los parámetros con mayores valores de gradiente obtienen menores tasas de aprendizaje.
- Otras alternativas, disponibles en Keras, por ejemplo, son **RMSProp** y **Adam**; que también adaptan los ritmos de aprendizaje a los diferentes parámetros
- Solo la experimentación nos puede dar si el uso de SGD, con o sin Momentum, o los métodos de tasa de aprendizaje adaptativo proporcionan más eficiencia y mejores resultados


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

4. Optimización
Momentum

- La tasa de aprendizaje (learning rate) es quizás el hiperparámetro más importante a la hora de entrenar una NN

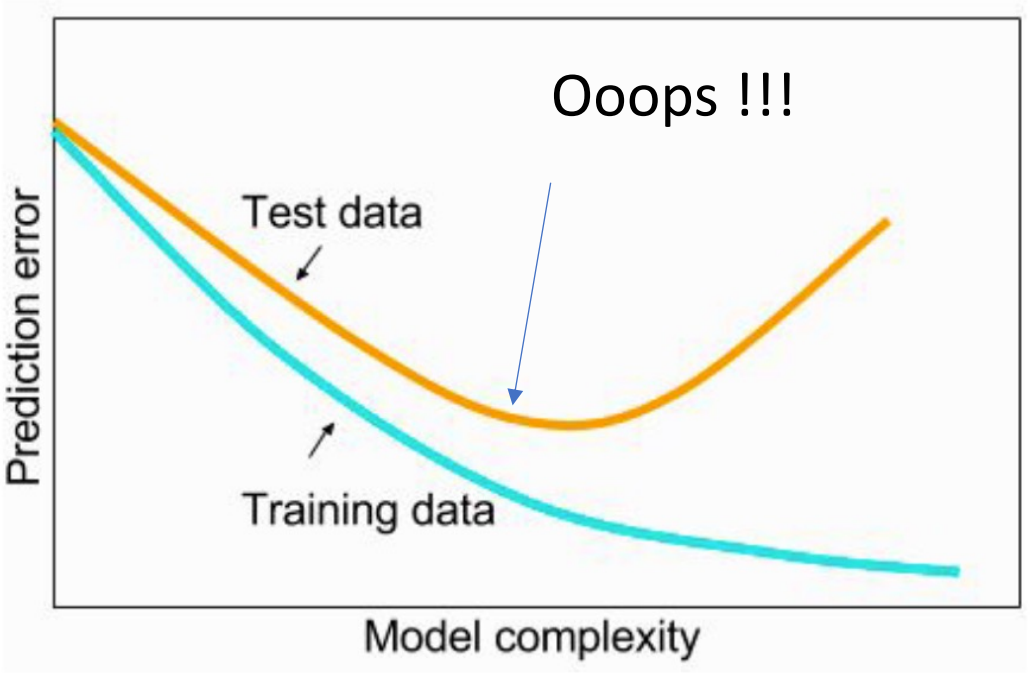


- La técnica de decaimiento de la tasa de aprendizaje consiste en reducirla a medida que evoluciona el proceso de entrenamiento.

5. Regularización

Disminución del error de prueba que mantiene el error de entrenamiento

- Las NN son modelos con un alto poder de representación, siendo capaces de memorizar los datos de entrenamiento y perder su proceso de generalización frente a nuevos datos; Esto es lo que ya hemos definido como sobreajuste
- Veremos diferentes técnicas de regularización para evitar esto (siendo común usar varias)



5. Regularización

Regularización L1 y L2

- La idea es **agregar una penalización a la función de coste**, basada en el valor de los pesos actuales
- En L2, añadimos a la función de coste el cuadrado de los pesos (con una atenuación), permitiendo que los pesos tengan valores absolutos pequeños, por lo que de alguna manera estamos reduciendo el espacio posible de soluciones, siendo el NN menos libre y perdiendo potencia de expresión; evitar que la NN se fije en detalles particulares de los datos de entrenamiento
- En L1, sumamos valores absolutos

- **L2**

Original cost function

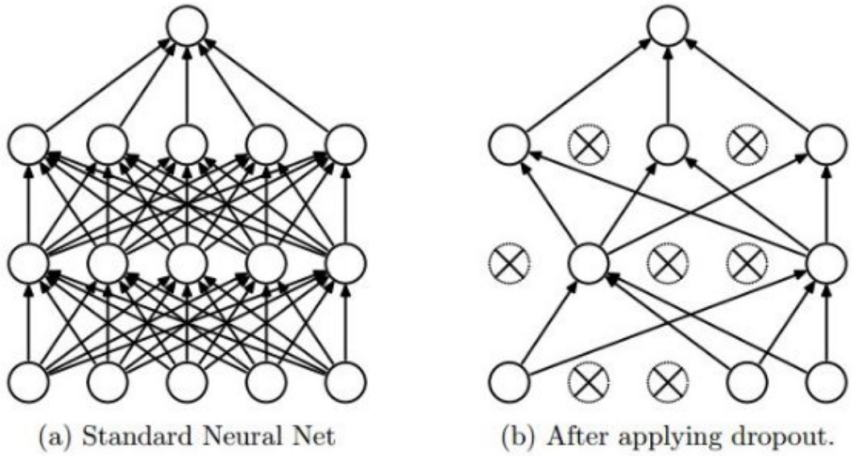
$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

- **L1**

$$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

5. Regularización
Dropout

- La técnica de regularización más utilizada
- Se basa en aplicar una salida 0 a un porcentaje de neuronas de la red en cada lote de forma aleatoria
- Es como desactivar un porcentaje de neuronas mientras se entrena
- Usamos una probabilidad p normalmente de 0.1 a 0.5

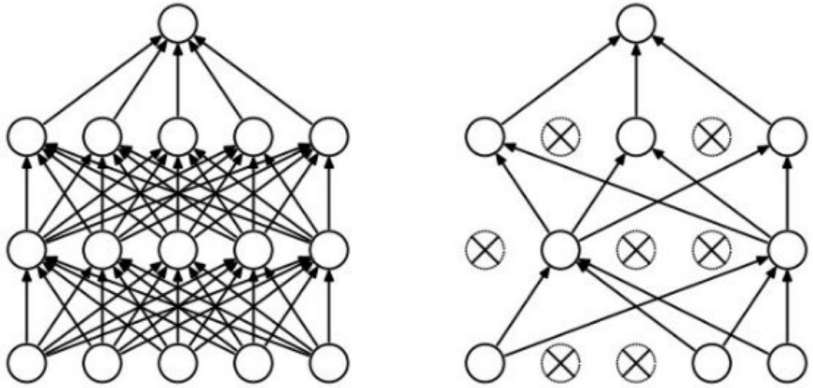


(a) Standard Neural Net

(b) After applying dropout.

5. Regularización
Dropout

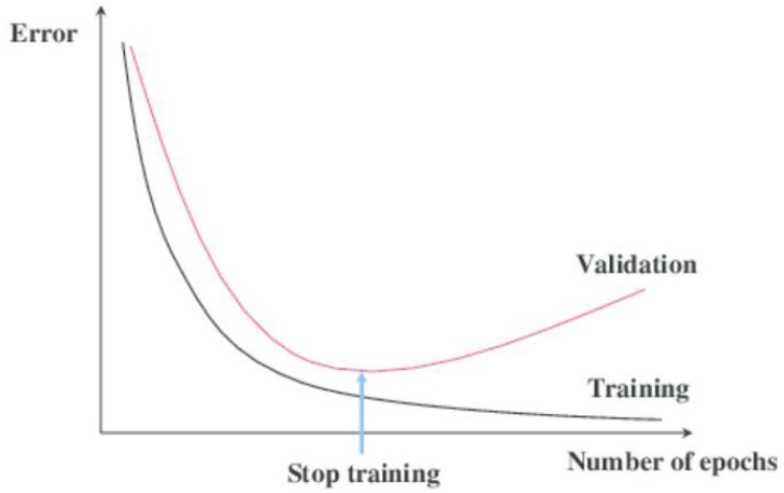
- Los **abandonos evitan la coadaptación de las características**; desactivando las neuronas, la NN tiene que aprender nuevas formas de correlacionar el valor de salida de las neuronas
- Evitamos que la NN memorice los valores de entrenamiento, consiguiendo un fuerte efecto de regularización
- Es como tener un método de ensemble, donde obtenemos una nueva NN por batch
- **Solo se aplica durante el tiempo de entrenamiento**



(a) Standard Neural Net (b) After applying dropout.

5. Regularización
Early stopping

- Muy común en la práctica para evitar el sobreajuste
- **Comparamos métricas entre conjuntos de entrenamiento y validación**, para detener las épocas en las que empezamos a sobreajustar (mejorando las métricas en el entrenamiento pero no en la validación)



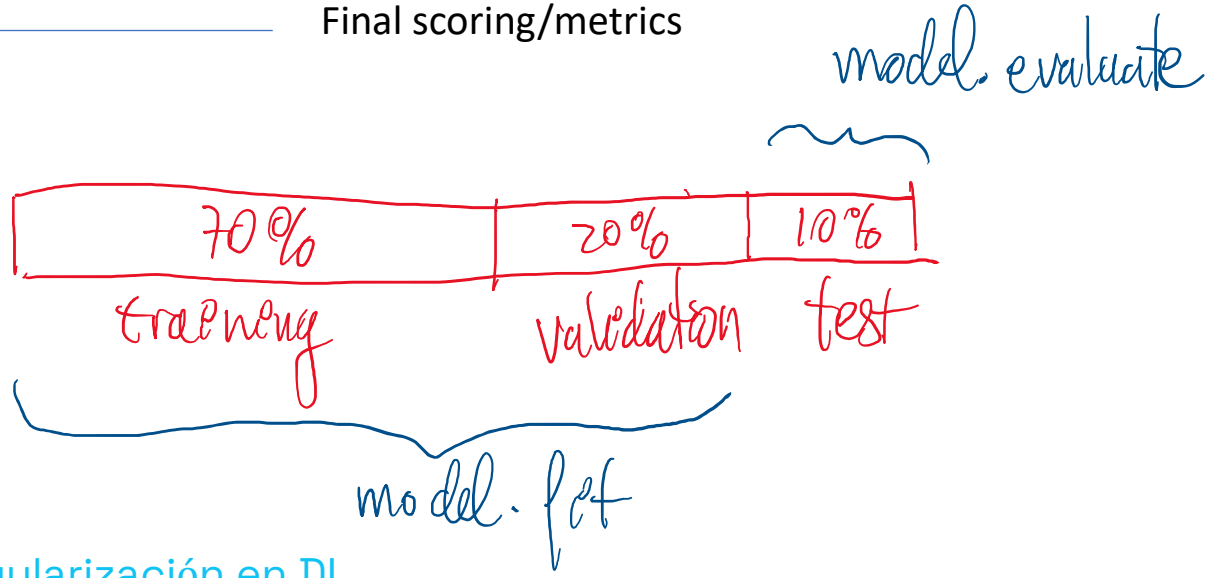
6. Guías de entrenamiento

Train, validation, test

- Entrenar con división en conjuntos de entrenamiento/validación (80 %-20 %, 70 %-30 %)
- Si es posible, evalúa de nuevo el modelo con un conjunto de datos de prueba nuevo y que aún no ha visto
- Una división común es (suponiendo un conjunto de datos de entrada):
- 70% training
- 20% validation
- 10% test

Used when training the NN

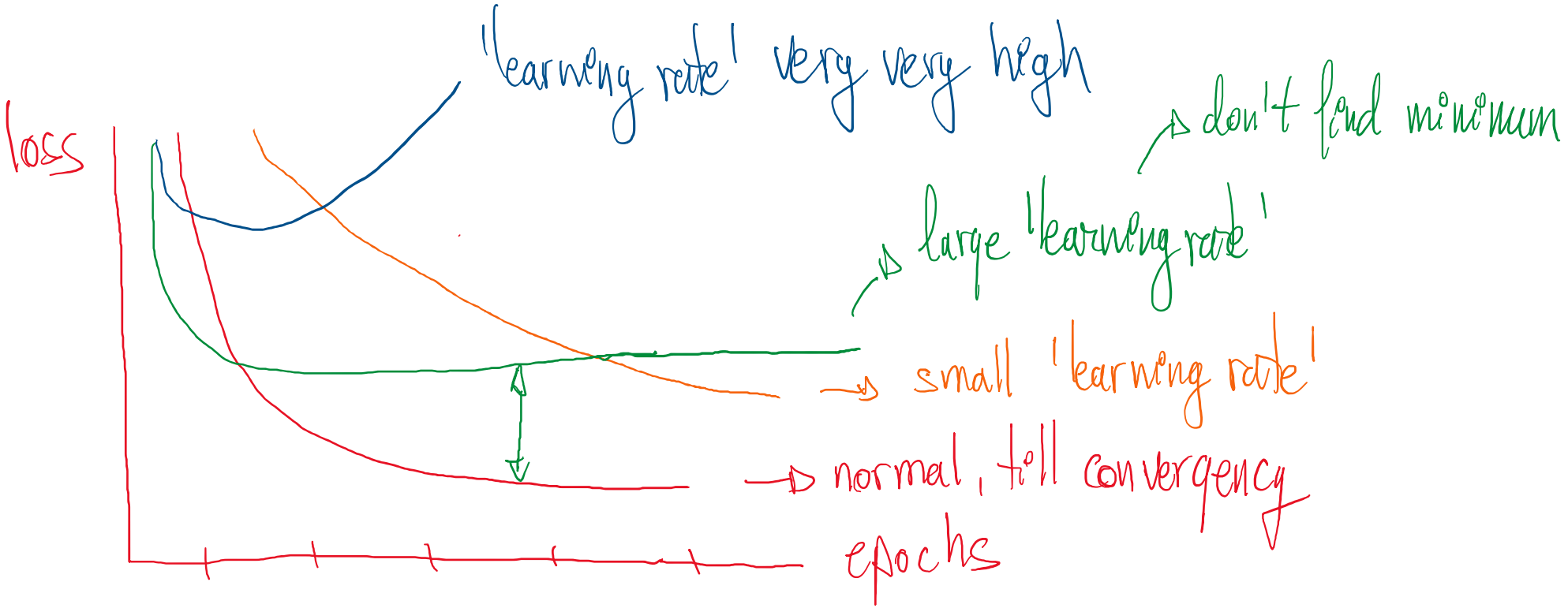
Final scoring/metrics



6. Guías de entrenamiento

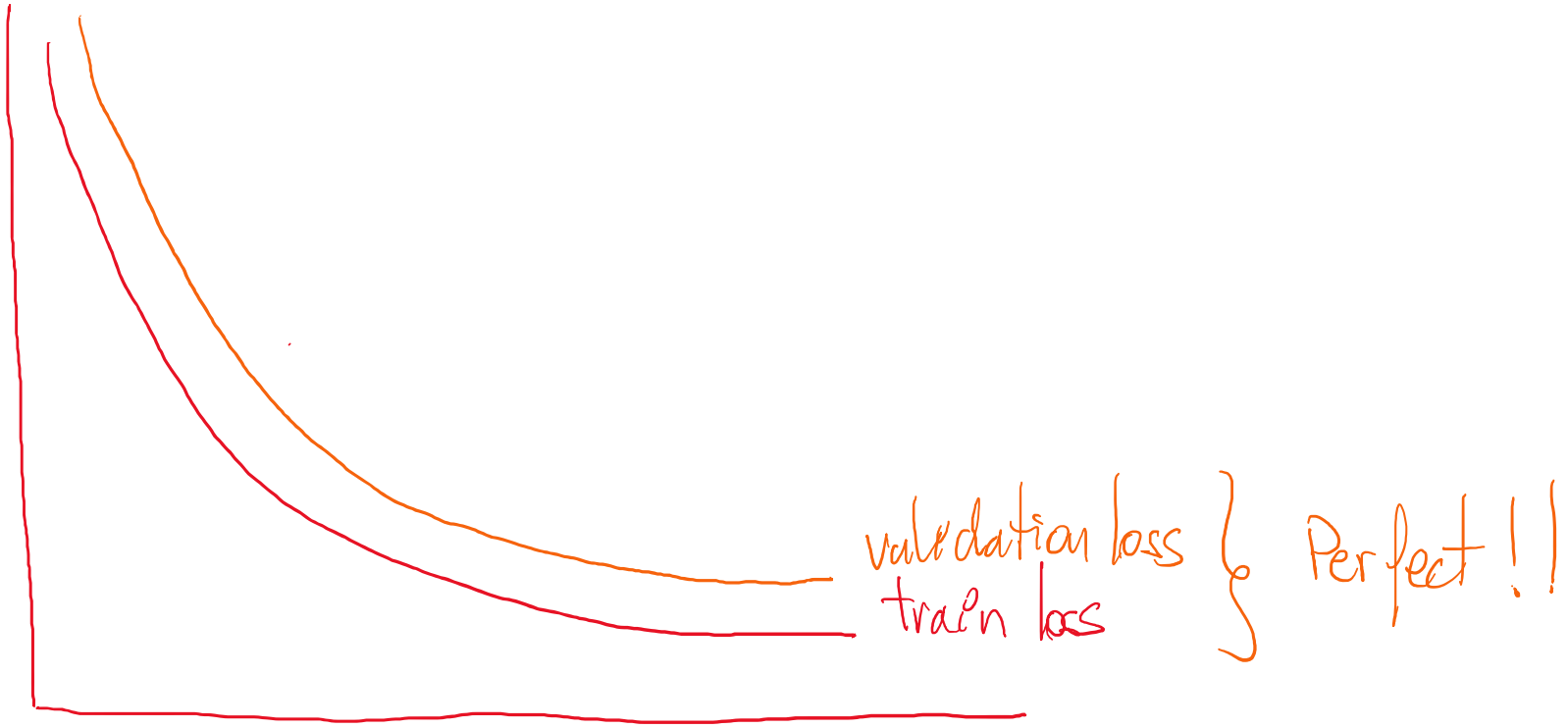
Más información

- La función de pérdida debe disminuir (presta mucha atención a la 'tasa de aprendizaje')



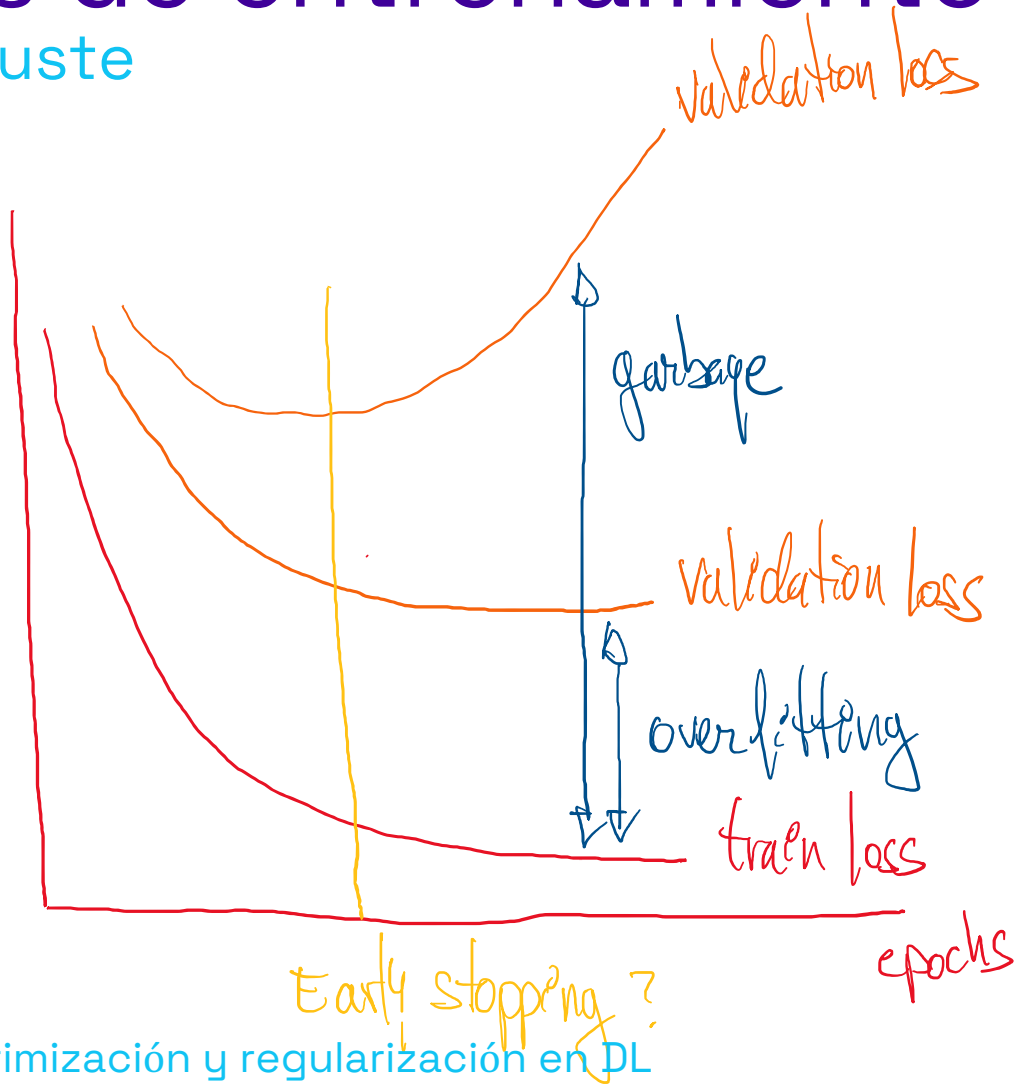
6. Guías de entrenamiento

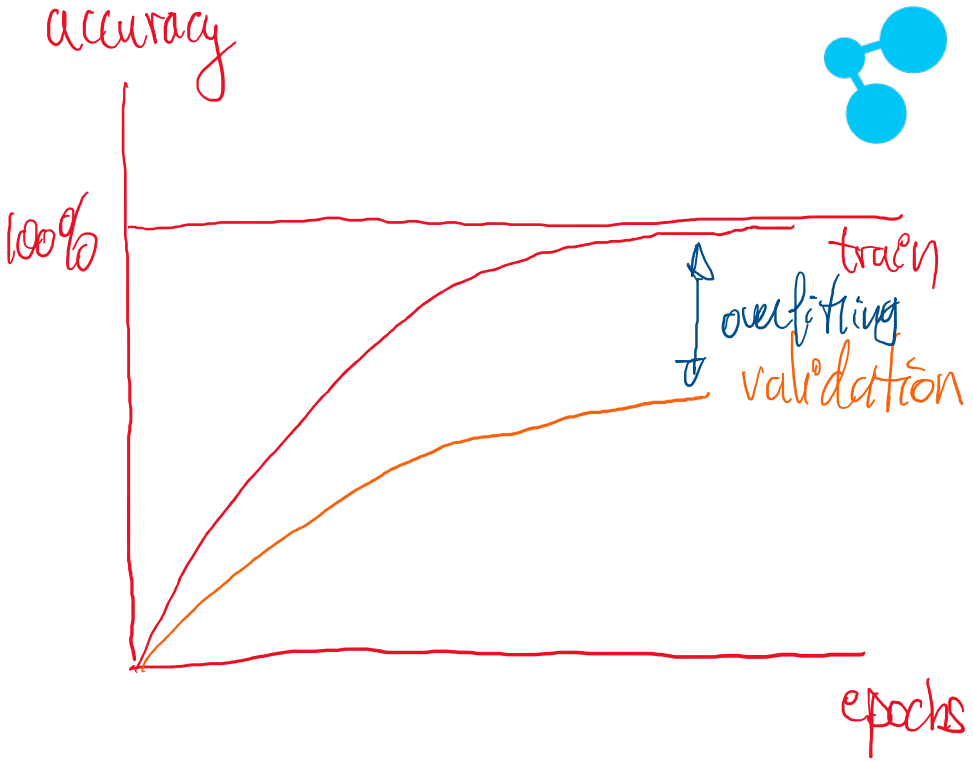
Sobreajuste



6. Guías de entrenamiento

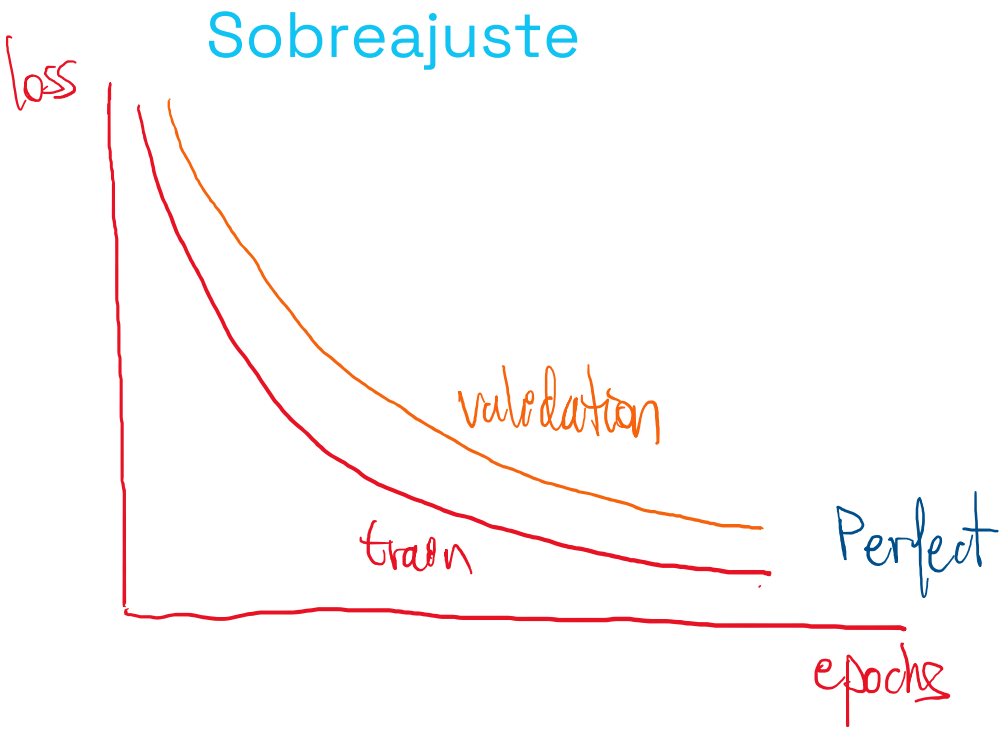
Sobreajuste

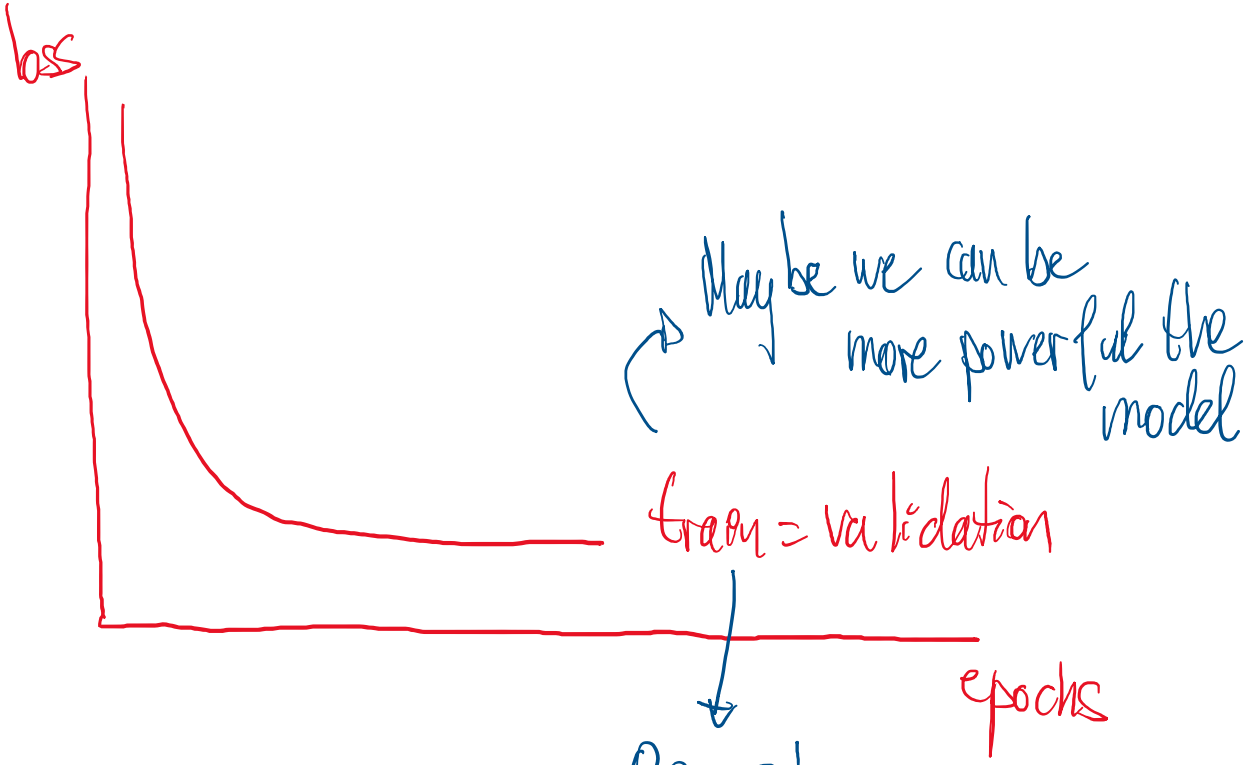




6. Guías de entrenamiento

Sobreajuste





6. Guías de entrenamiento
Sobreajuste

- ¿Sobreajuste?
- Aumentar la regularización (L1, L2, dropout...)
- **Obtener más datos**; cuantos más datos, menos difícil será para la NN memorizar aspectos específicos de los datos de entrenamiento
- Usar un **modelo menos complejo** (eso significa que asumimos que no podemos hacer más con los datos actuales)

6. Guías de entrenamiento

Más consejos

- **Más consejos sobre la formación de NN**
- Es importante tener datos de entrenamiento correctos, sin imprecisiones, valores atípicos o errores
- Es importante normalizar/estandarizar los datos
- Comienza con un modelo simple
- Antes de utilizar técnicas más avanzadas, utiliza un modelo sencillo (sin regularización) y comprueba si el modelo aprende (disminuye la pérdida)
- Después de eso, podemos agregar más elementos para ver si mejoramos las métricas
- Probar el modelo con un pequeño conjunto de muestras (20-100), desactivar la regularización y comprobar si tenemos sobreajuste; Si no es capaz de sobreajustar, será difícil que el modelo sea capaz de aprender con grandes conjuntos de datos

6. Guías de entrenamiento

Más consejos

- **¿Qué optimizador y tasa de aprendizaje funcionan bien?**
- Seleccionar la tasa de aprendizaje y el optimizador es clave para encontrar un modelo que nos dé los mejores resultados y con un entrenamiento eficiente
- Es una buena idea ver qué valores hacen que nuestro modelo converja y cuáles lo hacen divergir
- Una buena idea es probar las primeras tasas de aprendizaje en una escala logarítmica (0,00001, 0,0001, 0,001, 0,01. 1, 10)
- Y luego, ¡ajústalo!

Valencian Graduate School and Research Network of Artificial Intelligence

valgrai.eu

valgrai.eu

Valencian Graduate School and Research Network of Artificial Intelligence