Prevención e identificación del sobreajuste

TÉCNICAS ESTADÍSTICAS PARA EL APRENDIZAJE II

Máster Universitario en Estadística Computacional
y Ciencia de Datos para la Toma de Decisiones

Prevención e identificación del sobreajuste

Introducción

Regularización

- Regularización de la función de coste
- Restricciones sobre los parámetros de la red

Early stopping

Dropout

Introducción

Desde el punto de vista formal, una red neuronal multicapa es un **aproximador universal**. Por este motivo, en principio, debería ser capaz de aprender cualquier cosa que nos pueda interesar.

Sin embargo, en la práctica, no existen criterios formales que nos permitan decidir, de antemano, cuál es la **configuración más adecuada de los hiperparámetros** del algoritmo de entrenamiento que nos garantice un aprendizaje adecuado. De hecho, nada nos garantiza que la red sea capaz de generalizar correctamente más allá del conjunto de entrenamiento.

En este tema veremos **técnicas que nos ayudarán a reducir el error de generalización de nuestros modelos**, intentando prevenir problemas derivados de un proceso de entrenamiento nunca del todo ideal.

Introducción

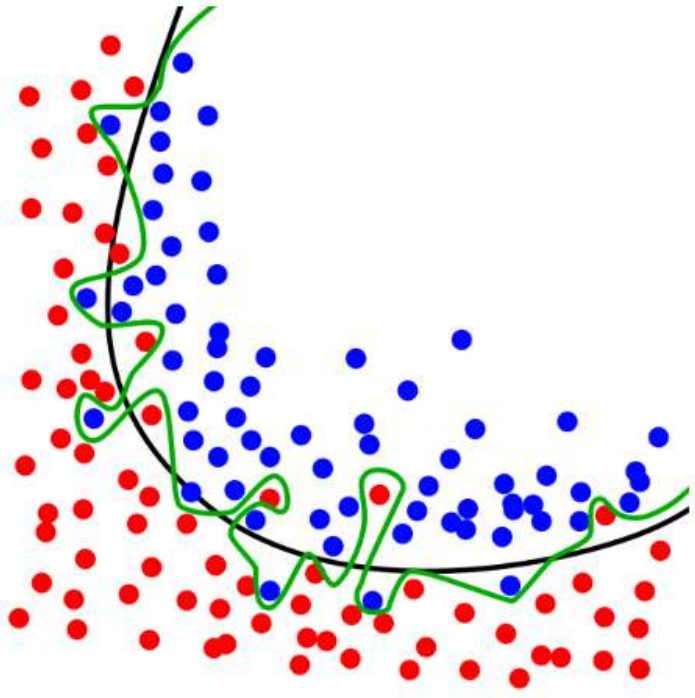
Obviamente, nos gustaría conseguir modelos que funcionen bien en el **conjunto de entrenamiento**, como prueba de que hemos aprendido algo.

Pero también que se comporten adecuadamente en los conjuntos de datos que nos encontraremos una vez puesto el modelo en uso, como prueba de que estamos **generalizando correctamente** (de ahí que la evaluación de los modelos la hagamos siempre con conjuntos de prueba independientes del conjunto de datos de entrenamiento).

El **sobreaprendizaje** aparece cuando nuestro modelo tiene una capacidad mayor que la estrictamente necesaria para generalizar correctamente, motivo por el que es capaz de **aprender detalles de los datos de entrenamiento que no son realmente relevantes** para la tarea para la que lo construimos (por ej. ruido debido a regularidades accidentales en el conjunto particular de datos).

Desgraciadamente, cuando ajustamos los parámetros de una red neuronal a los datos del conjunto de entrenamiento, **no podemos diferenciar las regularidades realmente útiles de las irrelevantes** o de las debidas al muestreo del conjunto de entrenamiento, por lo que **siempre estamos expuestos al riesgo de sobreaprender**.

Cuando sobreaprende, un modelo se ajusta demasiado bien al conjunto de entrenamiento, modelando sus peculiaridades al detalle y su ruido. Como consecuencia, **no generaliza bien**: su rendimiento se degrada cuando lo utilizamos sobre casos distintos a los del conjunto de entrenamiento, que es realmente donde nos interesa que funcione bien.



Introducción

Las **redes neuronales**, como ya sabemos, disponen de montones de parámetros, los pesos que determinan las interacciones de unas neuronas con otras. Como consecuencia, son **modelos especialmente propensos a ser víctimas del sobreaprendizaje**.

Por suerte para nosotros, los investigadores del tema son perfectamente conscientes del problema y, a lo largo de los años, han propuesto **multitud de técnicas para combatir el sobreaprendizaje**.

A menudo, son técnicas específicamente diseñadas para su uso en redes neuronales, no aplicables a otros modelos de aprendizaje automático (por ejemplo, una de las más populares, denominada ***dropout***).

Prevención e identificación del sobreajuste

Introducción

Regularización

- Regularización de la función de coste
- Restricciones sobre los parámetros de la red

Early stopping

Dropout

Regularización

En aprendizaje automático, la **regularización** es cualquier modificación que realizamos sobre un algoritmo de aprendizaje con la intención de reducir su error de generalización. Es decir, el **objetivo** de la regularización es **prevenir el sobreaprendizaje**.

No hay una técnica de regularización que sea siempre preferible a las demás. Tendremos que **elegir, para cada situación, una forma de regularización** que se adapte bien al problema particular que deseemos resolver.

Afortunadamente, no tendremos que partir de cero, sino que existe un conjunto bastante amplio de **técnicas genéricas de regularización** que resultan aplicables en una amplia gama de problemas. Estas técnicas lo que suelen hacer es **ajustar la capacidad del modelo.**

Regularización

Controlar la capacidad del modelo no es simplemente descubrir de qué tamaño debe ser la red neuronal, de cuántas capas debe constar, cuántos parámetros hemos de ajustar o con qué hiperparámetros hemos de entrenarla.

Para **encontrar el mejor modelo**, en el sentido de minimizar su error de generalización, debemos entrenar un modelo que sea lo **suficientemente grande para evitar el *underfitting* y que haya sido debidamente regularizado**.

¿De qué **estrategias generales** disponemos si deseamos ajustar la capacidad efectiva de una red neuronal mediante técnicas de regularización?

Esencialmente, de las dos siguientes:

Regularización

1) Añadir términos extra en la función de coste o pérdida que se utiliza como objetivo en el proceso de optimización de los parámetros de la red. Esta estrategia puede verse como una forma de imponer restricciones débiles [*soft constraints*] sobre los valores de los parámetros de la red neuronal, ya que nos permite **establecer, de forma indirecta, limitaciones sobre las configuraciones posibles de la red**.

2) Añadir restricciones adicionales al modelo de red neuronal, en forma de restricciones **sobre los valores de sus parámetros**. Por ejemplo, podemos obligar a que distintos pesos compartan siempre el mismo valor, es decir, se reduce el número de parámetros de la red haciendo que distintas neuronas compartan los mismos pesos (por ejemplo, redes convolutivas).

Regularización de la función de coste

La forma más común de regularizar el entrenamiento de una red neuronal artificial consiste en **añadir términos adicionales a la función de coste** o pérdida que intentamos optimizar ajustando sus parámetros.

Hasta ahora, esa función de coste era, básicamente, una **función de error** que evaluábamos sobre el conjunto de entrenamiento con la intención de conseguir que la red aprendiese.

Al introducir un término de regularización, la **función objetivo regularizada** pasa a ser de la forma:

$$\boldsymbol{L = L_E + \lambda L_R}$$

donde L_E es nuestra función de error, L_R es el término de regularización y λ es un parámetro que nos sirve para darle más o menos peso al término de regularización con respecto al término L_E asociado al error.

Regularización de la función de coste

Al incluir un término adicional en la función de coste, también **añadimos un hiperparámetro más al algoritmo de entrenamiento**.

Como sucede con otros hiperparámetros involucrados en el entrenamiento de la red, **establecer un valor adecuado para el parámetro de regularización λ** es esencial para controlar de forma efectiva la capacidad de la red, previniendo el sobreaprendizaje y facilitando su capacidad de generalización.

El **entrenamiento de la red**, como siempre, se hará calculando el gradiente de la función de coste o pérdida y el resto del proceso de entrenamiento sigue siendo exactamente el mismo. **Sólo cambia la señal que se propaga hacia atrás cuando usamos *backpropagation***. En esa señal, además del error sobre el conjunto de entrenamiento, se incluye información que, idealmente, contribuirá a reducir el error de generalización de la red neuronal una vez entrenada.

Regularización de la función de coste

Los dos **métodos más comunes de regularización de la función de coste** penalizan la aparición de pesos grandes en función de sus valores al cuadrado (regularización L2) o absolutos (regularización L1). La diferencia reside en la forma que toma el término de penalización L_R en la función de pérdida.

Regularización L2: es la forma más habitual de regularización en redes neuronales artificiales, donde se suele conocer con el término de ***weight decay*** (decaimiento de pesos):

$$L = L_E + \frac{1}{2}\lambda \sum w_k^2$$

El factor 1/2 se introduce sólo para que el 2 desaparezca al calcular la derivada y no haya que ir arrastrándolo en todos los cálculos. De esta forma, el gradiente del término de regularización con respecto al vector de pesos w es, simplemente, λw.

Regularización de la función de coste

Regularización L2:

$$L = L_E + \frac{1}{2}\lambda \sum w_k^2$$

¿Por qué se denomina L2?

Porque la norma L2 de un vector w es su norma euclídea $||w||_2$ (raíz cuadrada de la suma de cuadrados de los elementos del vector), la cual se usa, por ejemplo, para medir distancias en un espacio euclídeo.

$$||w||_2^2 = \sum w_k^2$$

Regularización de la función de coste

Al calcular el **gradiente de la función regularizada** para utilizarlo en el gradiente descendente con *backpropagation*, la actualización de los pesos se realiza de la siguiente manera:

$$\Delta w_k = -\eta \frac{\partial L}{\partial w_k} = -\eta \left(\frac{\partial E}{\partial w_k} + \lambda w_k \right)$$

donde vemos cómo el único cambio que se introduce en la regla de aprendizaje es un **término adicional que contrae los pesos en un factor $-\eta\lambda$ en cada paso del entrenamiento de la red**.

Dada la tendencia al sobreaprendizaje de las redes neuronales, este sencillo mecanismo que disminuye la magnitud de los pesos en cada actualización, proporciona una forma de hacer que los pesos tiendan a desaparecer. **Sólo los pesos más relevantes para reducir el error tendrán una magnitud significativamente distinta de cero al final del entrenamiento**.

Regularización de la función de coste

Regularización L1:

$$L = L_E + \lambda \sum |w_k|$$

La regularización L1 se denomina así por utilizar la norma L1 de un vector w (suma de los valores absolutos de los elementos del vector), que es la que se emplea, por ejemplo, para calcular la distancia de Manhattan (la distancia que habría que recorrer, en una ciudad con un plano perfectamente cuadriculado, para llegar de un punto a otro).

$$||w||_1 = \sum |w_k|$$

En Estadística se denomina LASSO [*Least Absolute Shrinkage and Selection Operator*], término que a menudo aparece en minúsculas, como *lasso*.

Regularización de la función de coste

Al utilizar este término de regularización, el gradiente de la función de pérdida hace que nuestra regla de aprendizaje para actualizar los pesos usando el gradiente descendente tenga la forma:

$$\Delta w_k = -\eta \frac{\partial L}{\partial w_k} = -\eta \left(\frac{\partial E}{\partial w_k} + \lambda sign(w_k) \right)$$

donde $sign(w_k)$ es la función signo aplicada al peso w_k.

Cuando el peso es positivo, su valor se reducirá; cuando el peso es negativo, se incrementará y el cambio en el peso debido al término de regularización será de magnitud $\eta\lambda$.

La **contribución del término de regularización** al gradiente ya no depende linealmente de cada peso, como en *weight decay*, sino que es **constante**.

Regularización de la función de coste

Tanto la regularización L2 como la L1 hacen que la magnitud de los pesos se reduzca, penalizando, de forma ligeramente distinta, la aparición de pesos grandes y conservando aquellos parámetros que tienen un impacto notable sobre la función objetivo que pretendemos minimizar.

En la regularización L1, los pesos encogen una cantidad constante.

En la regularización L2, la reducción es proporcional al valor de los pesos.

Por tanto, cuando la magnitud de un peso es elevada, la regularización L2 reduce el peso mucho más rápidamente que la regularización L1. Sin embargo, cuando el peso ya es pequeño, la regularización L1 es más pronunciada que la regularización L2.

Regularización de la función de coste

Al final, la **regularización L1** tiende a concentrar los pesos de la red en un número relativamente pequeño de conexiones importantes mientras que los demás pesos acaban valiendo cero. En otras palabras, la red utiliza sólo un subconjunto de sus parámetros.

En cambio, la **regularización L2** tiende a obtener vectores finales de pesos en los que todos los pesos tienen valores pequeños.

Combinando la regularización L1 con la regularización L2, podemos obtener ventajas tanto de la rápida eliminación de pesos grandes, propia de la regularización L2, como de las soluciones dispersas características de la regularización L1:

$$L = L_E + \lambda_1 \sum |w_k| + \frac{1}{2}\lambda_2 \sum w_k^2$$

Regularización de la función de coste

Aunque las anteriores son las técnicas de regularización de la función de coste más populares y utilizadas en la práctica, existen **muchas otras propuestas de regularización**.

En general, todas promueven que algunos de los pesos de la red tengan valores cercanos a cero. De esta forma, **se regula la capacidad de la red, haciéndola menos propensa al sobreaprendizaje**.

En contadas ocasiones, puede resultar deseable utilizar una **penalización diferente en la regularización de los pesos para las diferentes capas** de una red neuronal multicapa. Esto equivale a añadir nuevos hiperparámetros al algoritmo de entrenamiento de la red, ya que se utilizará un parámetro de regularización λ diferente para cada capa de la misma. No es algo demasiado habitual y rara vez se usa salvo, tal vez, para la capa de salida de la red.

Restricciones sobre los parámetros de la red

Una alternativa a la regularización de la función de coste utilizada para guiar el entrenamiento de la red consiste en **realizar la regularización directamente sobre los valores de los pesos**.

Las formas más habituales de hacerlo consisten en imponer, de forma explícita, restricciones sobre los valores de los pesos, ya sea **limitando su magnitud o vinculando los valores de diferentes pesos de la red** (por ejemplo, compartiendo parámetros entre distintas neuronas).

Si se establece una **cota superior sobre la magnitud del vector de pesos de cada neurona** (su norma L2), se evita el uso de la regularización L2 y se permite el uso de una tasa inicial de aprendizaje más alta, para explorar más rápidamente el espacio de posibles soluciones manteniendo la estabilidad del algoritmo.

Restricciones sobre los parámetros de la red

En la práctica, lo más habitual es establecer una restricción $||\boldsymbol{w}||_2 < \boldsymbol{k}$(también conocida como ***max-norm regularization***), utilizando para *k* un valor que puede ser del orden de 3 ó 4.

Aunque establecer restricciones sobre la norma del vector de parámetros es una forma efectiva de regularizar sus valores, hay otra forma de imponer restricciones sobre los **valores de los pesos de una red: forzar que sean iguales** (conocido como ***weight sharing***).

Al usar los mismos pesos para diferentes neuronas, **se reduce la capacidad efectiva de la red**, con lo que se reduce también la posibilidad de que sobreaprenda. De esta forma, se consigue "regularizar" la red actuando sobre los valores de sus pesos. El *weight sharing* suele utilizarse en redes convolutivas.

Prevención e identificación del sobreajuste

Introducción

Regularización

- Regularización de la función de coste
- Restricciones sobre los parámetros de la red

Early stopping

Dropout

Early stopping

Existe una **técnica de regularización muy efectiva** que se ha convertido en una de las más utilizada en *deep learning*, ya que es una forma sencilla, rápida y económica de regularizar un modelo y prevenir los efectos negativos del sobreaprendizaje.

La técnica se podría denominar "**parada temprana**", aunque todo el mundo la conoce por su denominación en inglés: ***early stopping***. Se puede ver como una forma de "regularización en el tiempo" y los expertos recomiendan que se utilicen de forma casi universal.

Del mismo modo que utilizamos un conjunto de validación para determinar la configuración óptima de los hiperparámetros del algoritmo de entrenamiento de la red, **ahora utilizaremos un conjunto de validación para detectar cuándo la red ha dejado de aprender y comienza a sobreaprender**.

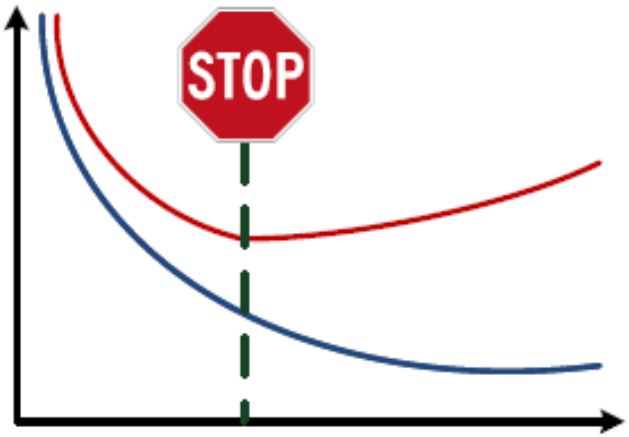
El conjunto de validación lo emplearemos para **evaluar periódicamente el error de la red conforme avanza su entrenamiento**. No utilizaremos el conjunto de entrenamiento porque nunca nos serviría como medida del error de generalización de la red.

En principio, si el algoritmo de entrenamiento de una red funciona adecuadamente, la **medida de error sobre el conjunto de entrenamiento** debería decrecer en cada iteración del algoritmo.

Sin embargo, **sobre el conjunto de validación, el error llega un momento en el que tiende a aumentar**, cuando el modelo empieza a dar muestras de sobreaprendizaje. Cuando detectemos esa situación, deberíamos detener el algoritmo iterativo de aprendizaje y devolver los valores de los parámetros para los que el error sobre el conjunto de validación era mínimo.

Early stopping

El entrenamiento de la red se detiene cuando observamos que la función de coste deja de mejorar sobre el conjunto de validación (en rojo), aunque podamos seguir reduciéndola sobre el conjunto de entrenamiento (en azul).



En *early stopping*, se añade un parámetro que se suele denominar "**paciencia**" que es el número de veces consecutivas tras las que, si no mejora la evaluación de la función de coste sobre el conjunto de validación, decidimos abandonar.

En cuanto a la paciencia que hemos de tener, podemos comenzar, por ejemplo, con 10. Posteriormente, conforme comprendamos mejor cómo se comporta la red particular que estamos entrenando, tal vez podemos usar 20, 50 o más.

Una vez establecido el número más indicado de pasos de entrenamiento (ya sean muestras en aprendizaje online, minilotes o épocas completas), podemos **continuar con el refinamiento de los valores de los parámetros** utilizando únicamente los datos del conjunto de validación.

Y… **¿cuándo paramos?**

Un posible criterio es detener el proceso de ajuste final de los parámetros de la red cuando la función de coste medida sobre el conjunto de validación quede por debajo de la que obtuvimos sobre el conjunto de entrenamiento al emplear *early stopping*.

El *early stopping* se puede **combinar con otras técnicas que tengan un efecto regularizador** sobre la red, como *dropout* que veremos a continuación.

Prevención e identificación del sobreajuste

Introducción

Regularización

- Regularización de la función de coste
- Restricciones sobre los parámetros de la red

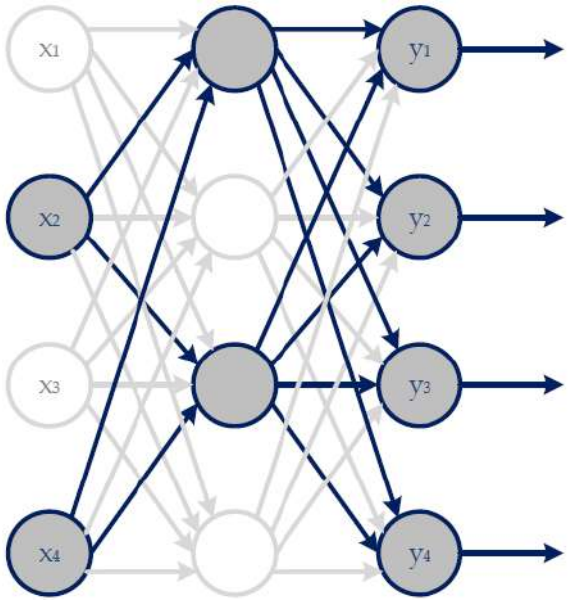
Early stopping

Dropout

Dropout es una forma muy sencilla y efectiva de regularización propuesta en 2014[1]: durante el entrenamiento de la red, **se mantiene una neurona activa con una probabilidad** p, usualmente $p = 0.5$, y su salida se anula o descarta [*drop*] con probabilidad $1-p$.

Esta técnica modifica la estructura de la red:

1) **Se eliminan de forma aleatoria** (y temporal) la mitad de las **neuronas ocultas** de la red (cuando $p = 0.5$), dejando intactas las neuronas de las capas de entrada y de salida (o, como mínimo, las de salida, que siempre resultan necesarias).



[1]Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, y Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html

2) **Se propaga la entrada x a través de la red modificada y se propaga hacia atrás el error**, también a través de la red modificada, como haríamos habitualmente para entrenar una red neuronal.

3) **Se actualizan los pesos de la red**, igual que se hace siempre a partir de la estimación del gradiente del error (obviamente, sólo una fracción de las neuronas se están utilizando actualmente, por lo que sólo se actualizarán los pesos de esas neuronas).

4) **Se vuelve a repetir el proceso, restaurando las neuronas eliminadas** [*dropout neurons*] antes de volver a seleccionar aleatoriamente qué neuronas se anularán a continuación.

La red entrenada de esta manera aprenderá un conjunto de pesos en condiciones diferentes a las habituales: en cada iteración se ha eliminado una fracción de las neuronas ocultas dada por p.

Dropout

La idea de *dropout* parte de la premisa de que, descartando neuronas de la red durante su entrenamiento, **se previene que diferentes neuronas se coadapten demasiado**. De esta forma, se reduce significativamente el sobreaprendizaje de la red.

Las neuronas entrenadas con *dropout* **tenderán a ser útiles por sí mismas**, sin depender de las contribuciones de otras neuronas de su misma capa, en cuya presencia no pueden confiar (recordemos que, en cada iteración del proceso de entrenamiento, se anulan neuronas).

Es una forma de conseguir que el aprendizaje de las neuronas ocultas no sea demasiado dependiente del contexto, sino que sea capaz de extraer características útiles bajo múltiples circunstancias, lo que hace a la **red más robusta**.

Dropout

Dropout puede verse como una forma de **muestrear una red neuronal de una familia enorme de posibles redes neuronales** (todas las subredes con una fracción p de las neuronas ocultas de la red original) y actualizar los pesos de esa red muestreada utilizando los datos de entrenamiento.

Durante su uso, ya no se aplica *dropout*. Ya no se descartan partes de la red, sino que **se utiliza toda la red** para obtener un resultado que puede interpretarse como una **estimación de la predicción media realizada por una colección de modelos** de tamaño exponencial.

Su popularidad ha ocasionado que se propongan diferentes **variaciones** que giran en torno a la misma idea: descartar neuronas, junto con sus conexiones, durante el proceso de entrenamiento de la red neuronal.

En la práctica, es habitual utilizar **regularización L2** (*weight decay*) para entrenar redes neuronales artificiales, utilizando un **parámetro de regularización global λ** previamente ajustado usando un conjunto de validación o validación cruzada.

Esta regularización global **se puede combinar con *dropout*** aplicado sobre todas las capas ocultas de la red (y, opcionalmente, la capa de entrada).

En cuanto al hiperparámetro p, el valor por defecto $\boldsymbol{p = 0.5}$ **suele ser razonable**, aunque también podemos ajustarlo sobre el conjunto de validación si lo consideramos oportuno.

Cualquier red que se entrene con ***early stopping*** puede hacerlo mejor si se utiliza *dropout*, aunque costará más tiempo entrenarla.

Bibliografía

•[1] Fernando Berzal. Redes Neuronales & Deep Learning. Edición independiente.

•[2] François Chollet. Deep learning with Python. Manning Shelter Island.

•[3] Ian Goodfellow, Yoshua Bengio & Aaron Courville. Deep Learning . MIT Press.