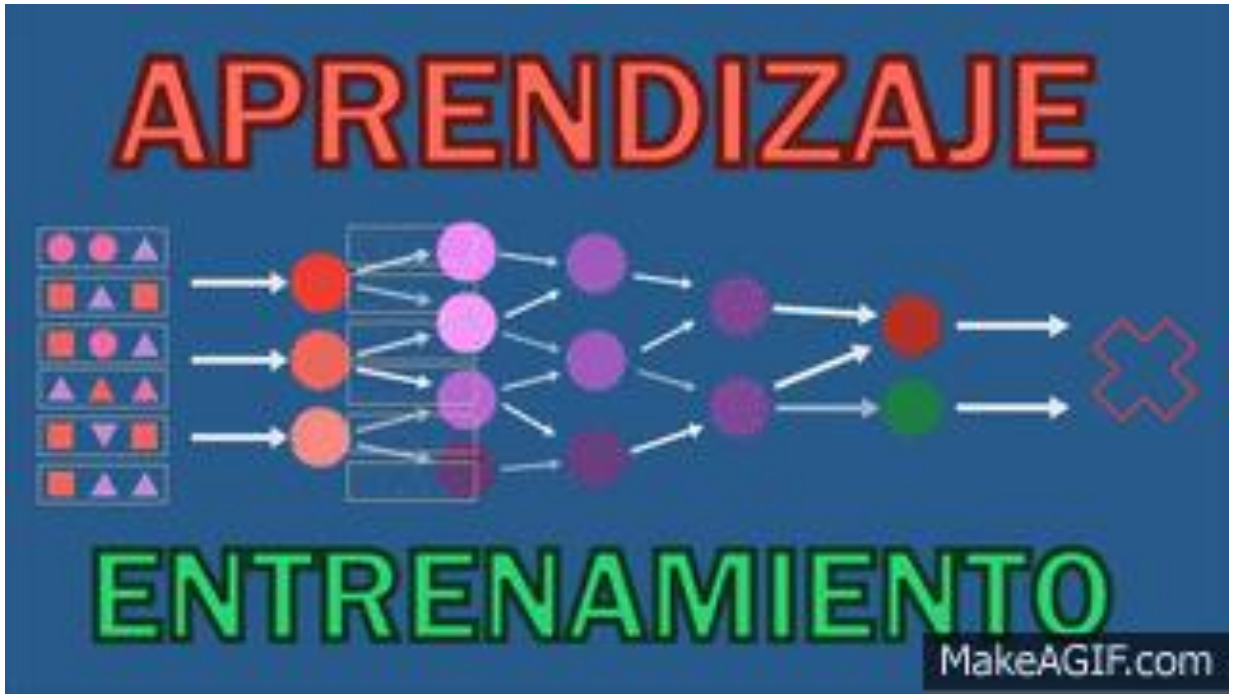
Tips para entrenar mejor las redes neuronales artificiales



¿Qué factores podemos variar en la red?

- Valores aleatorios con los que se inicializan los valores de los pesos de la red
- Número de capas y neuronas por capa (capas ocultas)
- Hiperparámetros de los algoritmos de optimización, como la tasa de aprendizaje
- Funciones de activación utilizadas en las capas ocultas y en la capa de salida
- Función de pérdida utilizada en la capa de salida

¿Qué factores podemos variar el conjunto de entrenamiento?

- Agruparlo en varias baterías, para calcular el vector gradiente con respecto a subconjuntos del conjunto de entrenamiento, en vez de hacerlo con todo el conjunto de entrenamiento
- Normalizar los ejemplos del conjunto de entrenamiento
- Ordenación aleatoria de sus elementos

Inicialización de los valores aleatorios de los pesos

- Nunca se deben inicializar todos los pesos con el mismo valor
- Un método es poner valores entre dos decimales conocidos, por ejemplo, en el rango [1.0,-1.0]
- Otro método es generar valores aleatorios en una distribución normal de media 0 y desviación típica 1
- Otro es generar valores aleatorios en una distribución normal de media 0 y desviación típica 1/n o 2/n, siendo n el número de nodos de salida que tiene una neurona
- Otro especial para ReLU, Xavier Weight Initialization
- Más ejemplos: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/

Cuantas más neuronas y capas, mejor

- Permite que los algoritmos de optimización funcionen mejor en general y que se puedan utilizar funciones de activación más efectivas
- No existe una regla general para establecer el número de capas y el número de neuronas en cada capa, pero se puede ir probando, la idea es que de menor número de capas a mayor, hasta alcanzar buenos resultados
- Algo recomendable es evitar utilizar una neurona por capa, mínimo 2, a menos de que el problema sea muy simple, o tenga 1 entrada.

Tasa de aprendizaje

- Es buena idea empezar con el valor 0,01. En el caso de utilizar un algoritmo de optimización y observar que nos quedamos estancados en un mal o no suficientemente buen mínimo local, es recomendable reducir este número, aunque no mucho, por ejemplo, 0,001
- Si se ven mejoras al reducir esta tasa, se puede buscar un punto donde comience a ser menos efectivo o incluso contraproducente aumentar su valor

Otros hiperparámetros

- Es recomendable, por ejemplo si se utiliza ADAM, utilizar los hiperparámetros que se suelen utilizar, y sólo probar con aquellos que se sospeche que se pueda mejorar el aprendizaje y se puedan manejar con cierta intuinción, por ejemplo, la tasa de aprendizaje

Funciones de coste/pérdida

- En el caso de intentar resolver problemas de regresión (predecir números concretos), es mejor utilizar el error cuadrático medio
- En el caso de intentar clasificar en un solo tipo cada ejemplo, es mejor utilizar la entropía cruzada, si puede ser con softmax como función de activación en la capa de salida
- En problemas de clasificación múltiple, se utiliza algo diferente a la entropía cruzada,

Algoritmos de optimización

- Conviene utilizar, en redes más densas, algoritmos de optimización complejos, como puede ser ADAM
- En cambio, en redes con pocas neuronas, es mejor utilizar el descenso de gradiente normal o el descenso de gradiente estocástico, con una tasa de aprendizaje, ya que los algoritmos sofisticados suelen quedarse atascados en un mal mínimo local sin poder salir del mismo

Funciones de activación

- Conviene utilizar funciones de activación que computen resultados más rápido, como puede ser ReLU o pReLU, en redes más densas y con un buen algoritmo de optimización
- En cambio, en redes con pocas neuronas y sin un algoritmo de optimización sofisticado (no suelen dar buenos resultados en estas pequeñas redes) suele ser mejor utilizar funciones de activación más compactas, como la sigmoide o la tangencial

Resumen redes poco densas

- En este caso, es mejor utilizar el algoritmo de descenso de gradiente o descenso del gradiente estocástico a utilizar algoritmos de optimización como Adam, ya que estos algoritmos no suelen funcionar bien en redes con pocas neuronas
- Aparte, lo ideal es utilizar funciones de activación en las capas ocultas como la sigmoide o la tangencial, ya que dan resultados normalmente más decentes en estos escenarios
- Cuidado, en este escenario es más probable que nos quedemos estancados en un mínimo local bastante malo, aunque no mucho

Resumen redes bastante densas

- En este caso, es mejor utilizar un algoritmo de optimización como puede ser ADAM, ya que suele hacer aprender más rápido a la red
- Aparte, lo ideal es utilizar funciones de activación como pReLU, ya que son mucho más rápidos de computar y dan resultados decentes, incluso a veces permiten aprender mejor en algunos casos
- En este escenario es menos probable quedarse estancado en un mínimo local malo, y normalmente, con un buen algoritmo de optimización, es más probable salir de uno de esos mínimos locales. A veces un problema es que nos quedamos atascados en un mínimo local, aceptable, pero estaría bien mejorarlo

¿Habrá un método mejor?

- Sería buena idea hacer validación cruzada para ver si nuestro modelo aprende bien o no, y para comprobarlo con otros modelos. Esta técnica permite asegurar que haya cierta independencia de los datos con los que se han entrenado los modelos
- Vídeo que lo explica bastante bien: https://www.youtube.com/watch?v=fSytzGwwBVw