Redes neuronales

Formadas por capas de neuronas:

- Capa de entrada
- Capas ocultas
- Capa de salida

Tipos de capas

- **Fully connected**

  Una red formada únicamente por capas fully connected se denomina, en general, "red fully connected".
- **Convolucionales**

  Una red que tiene capas convolucionales se denomina, en general, "red convolucional" (aunque incluyan alguna capa fully connected, especialmente a la salida de la red).
- **Recurrentes**

  Una red que tiene capas recurrentes se denomina, en general, "red recurrente".

Existen arquitecturas complejas que combinan varias de estas capas.

¿Cómo se entrenan las redes neuronales?

- Pase hacia delante (forward propagation o pase forward).
- Cálculo de la pérdida a la salida de la red.
- Pase hacia atrás (backpropagation o pase backward), utilizando el algoritmo de optimización del descenso de gradiente (o sus variantes más evolucionadas, como Adam).

¿Qué problemas suelen presentar las redes neuronales?

- Son modelos complejos que tienden a **sobreajustarse**.
- Las funciones de coste presentan más de un mínimo y es habitual que, durante el proceso de optimización con descenso de gradiente, el entrenamiento finalice en algún **mínimo local**, en lugar de llegar al mínimo global (óptimo) de la función.
- **Vanishing gradient** o gradiente que desaparece. Al hacer el pase hacia atrás, los gradientes calculados van siendo cada vez más pequeños. Como la regla de actualización de los parámetros de la red tiene en cuenta estos gradientes, si son demasiado pequeños puede que los parámetros prácticamente dejen de actualizarse (la red deja de aprender).
- **Internal covariate shift.** Durante el entrenamiento de una red neuronal los pesos de las unidades ocultas cambian en cada época. Esto provoca que las distribuciones de las activaciones de cada capa cambien.

¿Cómo se evitan los problemas de las redes neuronales?

- Aplicando **early stopping** para detectar el punto ideal en el que frenar el entrenamiento, que será aquel en que el modelo responda muy bien ante los datos de entrenamiento y comience a responder peor en los datos de validación.
- Aplicando **regularización**, lo que puede realizarse de varias maneras:
  - Añadiendo un término de penalización a la función de coste: Lasso, Ridge, Elastic-Net...
  - Utilizando la técnica de dropout (muy popular en redes convolucionales).
  - Haciendo uso de otras técnicas de regularización, como Multitask Learning, Data augmentation, Parameter sharing, Sparsity, Ensembles...
- Por ejemplo, utilizando **versiones más potentes del algoritmo de descenso de gradiente** (Momentum, Nesterov, Adam, Adagrad, Adadelta, Adam, AdaMax, Nadam...). Algunas de ellas (como Adam, que es muy popular) van modificando automáticamente el learning rate durante el entrenamiento, lo que permite afinar más en la búsqueda de un buen punto mínimo de la función de coste.
- Inicializando los pesos de forma apropiada. El método más apropiado es el conocido como ***inicialización de Xavier*** (Glorot).
- **Normalizando** siempre los datos de entrada (media 0 y varianza 1), pero esto sólo soluciona el problema en la primera capa oculta. Para el resto, se aplica **batch normalization**, que consiste en normalizar las entradas de cada capa (las activaciones de la capa anterior) para cada batch de datos durante el entrenamiento.

Formas de abordar un problema

- Desarrollando un modelo **desde cero**
- Aplicando **transfer learning**
- Utilizando un **modelo ya entrenado** directamente para hacer predicciones.

Librerías más populares

- **NumPy**: para cálculos y operaciones con vectores y matrices.
- **Pandas**: para carga de datos de archivos y preprocesamiento.
- **Scikit Learn**: fundamentalmente, para modelos de Machine Learning (árboles de decisión, SVM, XG-Boost, K nearest neighbors...)
- **Keras**: para Deep Learning y preprocesamiento de datos más complejos (imágenes).
- **PyTorch**: para Deep Learning. Muy eficiente y flexible.
- **Tensorflow**: para Deep Learning a muy bajo nivel.
- **OpenCV**: para procesamiento de imágenes y vídeos.

Modelos populares

VGG16, Inception, ResNet, MobileNet...

Modelos muy potentes que habitualmente han sido entrenados en datasets enormes. La arquitectura y los pesos suelen estar disponibles de forma pública y gratuita para su reutilización.

miro