Deep learning en visión

Robótica

Guillermo Iglesias Hernández y Alberto Díaz Álvarez
Departamento de Sistemas Informáticos - Universidad Politécnica de Madrid

5 de octubre de 2023

Conceptos básicos de redes de neuronas

Sistema matemático capaz de realizar predicciones a partir de datos de entrada

- Propuesta por McCulloch y Pitts en 1943
- Basada en imitar el comportamiento de una neurona biológica
- Toma ciertos estímulos de entrada, los procesa y genera una nueva salida

Neurona biológica

- Estímulos $\rightarrow$ impulsos nerviosos

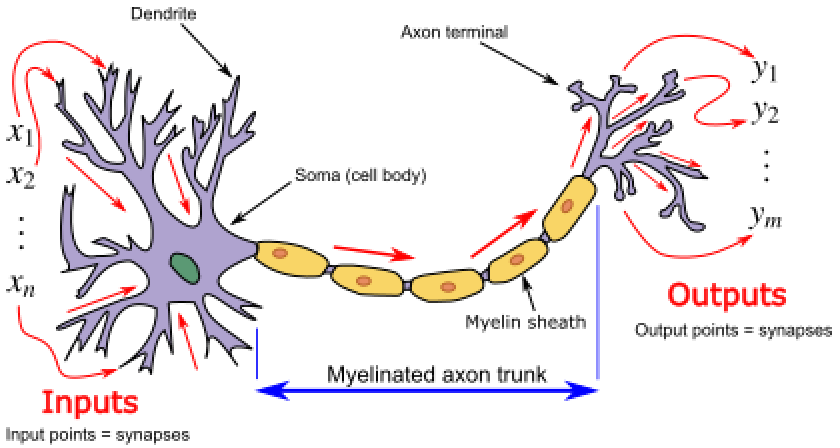
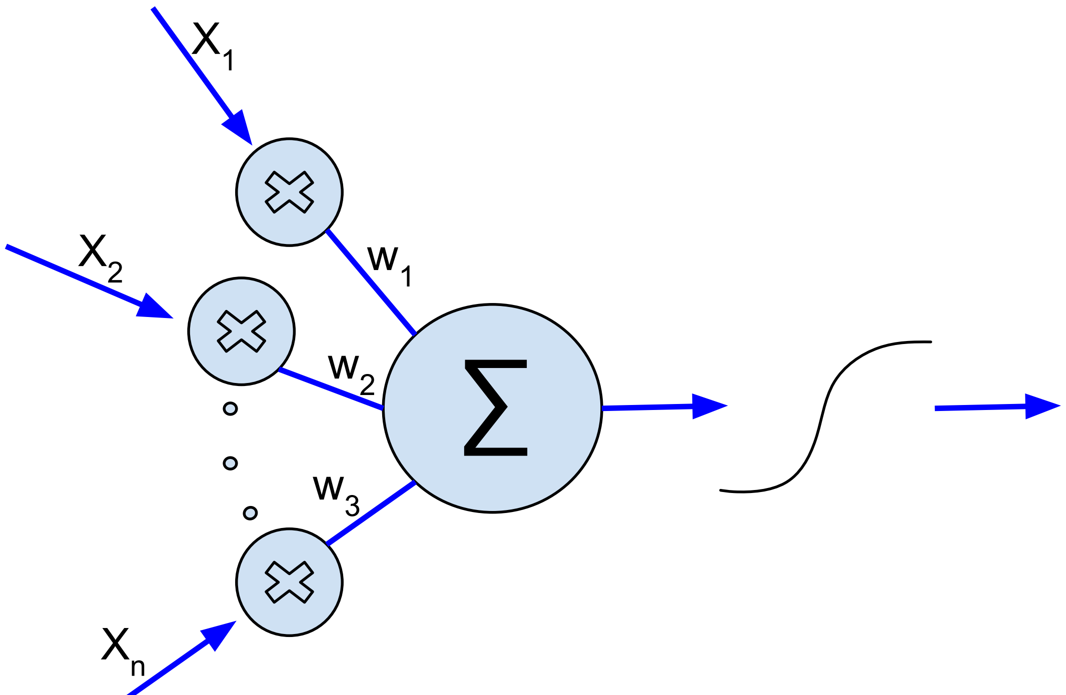


Fig.1 - Neurona biológica. Fuente: Wikipedia.

Neurona artificial

- Estímulos $\rightarrow$ cálculos matemáticos



Neurona artificial

Realiza cálculos matemáticos para transformar ciertos valores numéricos

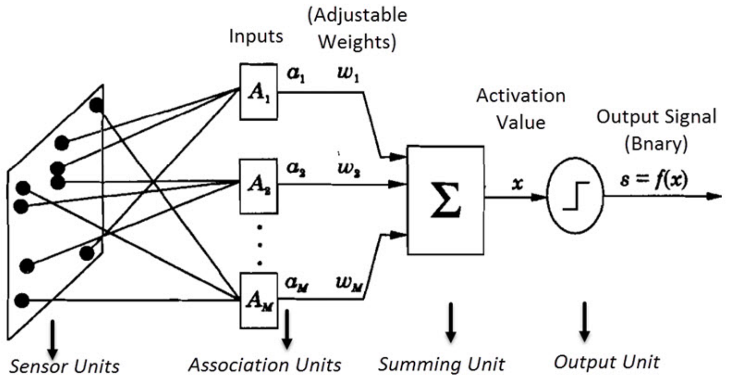


Fig.3 - Neurona artificial. Fuente: ElectronicsHub.

Para ello, existen diversos elementos dentro de una neurona artificial:

- **Entradas** (x_i): Los valores numéricos de entrada
- **Salida** (y): El valor de salida de la neurona
- **Pesos** (w_i): Parámetros capaces de cambiar, suponen el aprendizaje de la neurona
- **Bias** (b): Peso cuya entrada siempre es 1 y que desplaza la función de activación
- **Función de activación**: a: Participa en el cálculo de la salida de la neurona

Entrenamiento

En un esquema supervisado, podemos dividir el entrenamiento de una red neuronal en tres fases:

- **Inferencia**: Calculamos la salida de la red en función de las entradas y los pesos
- **Calculo del error**: Dadas las salida de la red y el resultado que queremos obtener calculamos el error obtenido
- **Ajuste de pesos**: Con dicho error se reajustan los parámetros de la red

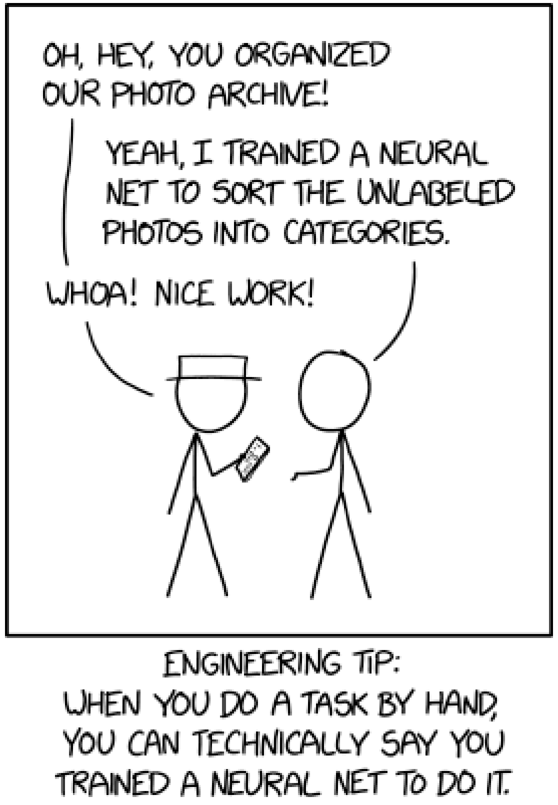


Fig.4 - Neurona artificial.
Fuente: XKCD.

Algoritmo de propagación

La fase de \alert{predicción} de una red neuronal se realiza a través del algoritmo de \alert{propagación}.

Este se encarga de procesar la \alert{entrada} y generar la \alert{salida} correspondiente.

Para ello, la computación de cada \alert{neurona} es la siguiente:
\begin{itemize}
\item Cada entrada x_i es \alert{multiplicada} por el valor de su peso correspondiente w_i.
\item La entrada de \textit{bias} \alert{siempre} es \say{1}, y se multiplica por su peso correspondiente (a veces indicado como w_0).
\item Todas las \alert{entradas} de la neurona se combinan haciendo una \alert{suma} de todas ellas, de tal manera que se realiza una \alert{combinación lineal}.
\item El resultado de la combinación lineal se pasa por una \alert{función no lineal} para generar la \alert{salida} de la neurona.
\end{itemize}

La ecuación que define este \alert{proceso} es la siguiente:
\setcounter{equation}{0}
\begin{equation}
\Large f\left(\sum_{i=0}^{n} W_{i} X_{i}\right)
\end{equation}

\begin{figure}
\centering
\includegraphics[width=0.6\textwidth]{figures/Tema 3/PropagationExample_1.png}
\end{figure}
\end{frame}

\begin{frame}{Algoritmo de propagación}
La ecuación que define este \alert{proceso} es la siguiente:
\setcounter{equation}{0}
\begin{equation}
\Large f\left(\sum_{i=0}^{n} W_{i} X_{i}\right)
\end{equation}

\begin{figure}

Algoritmo de propagación

La ecuación que define este \alert{proceso} es la siguiente:
\setcounter{equation}{0}
\begin{equation}
\Large f\left(\sum_{i=0}^{n} W_{i} X_{i}\right)
\end{equation}

\begin{figure}
\centering
\includegraphics[width=0.6\textwidth]{figures/Tema 3/PropagationExample_3.png}
\end{figure}

Algoritmo de propagación

La ecuación que define este \alert{proceso} es la siguiente:
\setcounter{equation}{0}
\begin{equation}
\Large f\left(\sum_{i=0}^{n} W_{i} X_{i}\right)
\end{equation}

\begin{figure}
\centering
\includegraphics[width=0.6\textwidth]{figures/Tema 3/PropagationExample_4.png}
\end{figure}

Algoritmo de propagación

La ecuación que define este \alert{proceso} es la siguiente:
\setcounter{equation}{0}
\begin{equation}
\Large f\left(\sum_{i=0}^{n} W_{i} X_{i}\right)
\end{equation}

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{figures/Tema 3/PropagationExample_5.png}
\end{figure}

Estructura de capas

Una red de neuronas \say{estándar} se organiza por \alert{capas}, las cuales se componen por varias \alert{neuronas}.

Cada \alert{capa de neuronas} se conecta con la siguiente y recibe \alert{datos} de la anterior. De esta manera se produce el \alert{flujo de datos} a lo largo de la red.

```
\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{figures/Tema 3/LayerStructure.png}
\end{figure}
```

¿Por qué introducir más capas?

Está matemáticamente \alert{demostrado} que sin función de activación las redes de neuronas sólo son capaces de resolver problemas \alert{linealmente separables}.

Esto es fácilmente demostrable, ya la computación de cada neurona corresponde con la ecuación de \alert{una recta}, y su combinación también.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{figures/Tema 1/Separabilidad_Lineal.png}
\end{figure}

Por otra parte, K. Hornik, M. Stinchcombe, y H. White demostraron el 1985 que con \alert{una única capa oculta} las redes neuronales artificiales se convierten en aproximadores universales \cite{hornik1989multilayer}

Funciones de activación

Las \alert{funciones de activación} de cada neurona pueden variar, entre las más populares se encuentran:

```
\begin{figure}
\centering
\includegraphics[width=\textwidth]{figures/Tema 3/Activations.png}
\caption{\cite{Activations}}
\end{figure}
```

Algoritmo de retropropagación

El algoritmo de \alert{retropropagación} o \alert{backpropagation} es el encargado de \alert{adaptar} la red de neuronas a su cometido específico.

Se basa en actualizar los \alert{pesos} de la red dependiendo del \alert{error} que esta haya tenido a la hora de predecir una \alert{salida} en concreto.

Con backpropagation obtendremos los \alert{gradientes (derivadas)} de la función de pérdida
para cada \alert{peso} de cada \alert{capa oculta}.

\begin{equation}
\bigtriangleup w_i = w_i - \alpha \cdot (Error)
\end{equation}

donde α es el \alert{learning rate}, que define la \alert{magnitud} con la que la red realiza la \alert{actualización} de sus pesos.

Funciones de pérdida

Existen múltiples métodos para calcular la \alert{distancia} de la \alert{predicción $\hat{y}$} con respecto de la \alert{salida deseada y}. Es decir, múltiples funciones de pérdida que nos permiten calcular el error.

\begin{equation}
\textbf{MAE} = \frac{1}{n} \sum{\left | y - \hat{y} \right |}
\end{equation}

\begin{equation}
\textbf{MSE} = \frac{1}{n}\sum{(y - \hat{y})^2}
\end{equation}

\begin{equation}
\textbf{Cross-Entropy} = - \sum{y \cdot \log \hat{y}}
\end{equation}

Entrenamiento de redes neuronales

Al realizar un entrenamiento con \alert{modelos de aprendizaje} se realiza una división del \alert{conjunto de datos} con el que se entrena. Este proceso ayuda a comprobar la \alert{fiabilidad} de la red.

\begin{figure}
\centering
\includegraphics[width=0.6\textwidth]{figures/Tema 3/DatasetDivision.png}
\end{figure}

Bias-variance tradeoff

Existen dos \alert{métricas} de alto nivel que evalúan el rendimiento de una red neuronal:
\begin{itemize}
\item \alert{Bias}: Es el error del modelo ante el conjunto de datos de \alert{entrenamiento}.
\item \alert{Variance}: Es el error del modelo ante el conjunto de datos de \alert{testeo} respecto los de entrenamiento.
\end{itemize}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figures/Tema 3/BiasVariance_1.jpg}
\caption{\cite{BiasVariance_1}}
\end{figure}

Bias-variance tradeoff

```
\centering
\includegraphics[width=0.7\textwidth]{figures/Tema 3/BiasVariance.png}
\caption{\cite{BiasVariance}}
\end{figure}
```

```
\Large \alert{Alto bias}
\end{column}

\begin{column}{0.49\textwidth}
\Large \alert{Alto variance}
\end{column}
\end{columns}

\begin{columns}[c]
\begin{column}{0.49\textwidth}
\begin{itemize}
\item Underfitting.
\item Sobre-simplificación del problema.
\item Valores de pérdida demasiado altos.
\item Falla al capturar la tendencia de los datos.
\end{itemize}
\end{column}

\begin{column}{0.49\textwidth}
\begin{itemize}
\item Overfitting.
```

Esquema general de entrenamiento de redes neuronales

```
\begin{figure}
\centering
\includegraphics[width=0.9\textwidth]{figures/Tema 3/NNTrainingScheme.png}
\end{figure}
```

Problemas del gradiente

Los problemas \alert{derivados del gradiente} son comunes a todas las redes neuronales. Estos están \alert{directamente influenciados} por el número de capas de la red.

Se diferencian dos tipos:
\begin{itemize}
\item Gradient explosion.
\item Gradient vanishing.
\end{itemize}

Al realizarse la \alert{retropropagación} los \alert{valores de pérdida} pasan de unas capas a otras. En este algoritmo las derivadas de cada neurona pueden llegar a \alert{descontrolarse}.

\begin{equation}
W_{x}^{\prime}=W_{x}-\mathrm{\alpha}\left(\frac{\partial \text {Loss}}{\partial W_{x}}\right)
\end{equation}

Problemas del gradiente: Gradient explosion

\begin{figure}
\centering
\includegraphics[width=0.6\textwidth]{figures/Tema 3/GradientExplosion.png}
\caption{\cite{GradienExplosion}}
\end{figure}

El \alert{gradient explosion}, también conocido como \alert{exploding gradients} sucede cuando la actualización de pesos toma valores \alert{muy elevados}.

Se identifica con valores de pérdidas de \alert{NaN o muy exageradas}

Problemas del gradiente: Gradient Vanishing

\begin{figure}
\centering
\includegraphics[width=0.6\textwidth]{figures/Tema 3/GradientVanishing.png}
\caption{\cite{GradienVanishing}}
\end{figure}

Cuando sucede \alert{gradient vanishing}, también llamado \alert{vanishing gradients}, la actualización de pesos se hace \alert{nula} por tener valores \alert{muy pequeños}.

Se identifica cuando la pérdida es \alert{constante en el tiempo}

Origen de los problemas derivados del gradiente

La principal \alert{causa} de estos problemas es usar \alert{funciones de activación} cuya derivada \alert{satura a 0}.

\begin{figure}
\centering
\includegraphics[width=0.6\textwidth]{figures/Tema 3/GradientCause.png}
\caption{\cite{GradienExplosion}}
\end{figure}

Sucede principalmente con las funciones \alert{tanh} y \alert{sigmoid}, por lo tanto se \alert{recomienda} el uso de ReLU para capas ocultas en una red.

Playground}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figures/Tema 3/Playground.png}
\caption{\href{https://playground.tensorflow.org/}{Tensorflow playground}}
\end{figure}

Perceptrón multicapa para procesar imágenes

¿Cómo procesar imágenes?

La idea más \alert{básica} para procesar imágenes con redes neuronales es transformar la \alert{matriz numérica} de datos a un \alert{vector unidimensional}.

\begin{figure}
\centering
\includegraphics[width=0.7\textwidth]{figures/Tema 3/NNVideo.jpg}
\caption{\href{https://www.youtube.com/watch?v=aircAruvnKk&t=218s}{Vídeo youtube}}
\end{figure}

Capa de reshape

La capa de \alert{keras} llamada \say{\alert{reshape}} se encarga de realizar esa transformación de \alert{matriz} a \alert{vector}.

Sin embargo el principal \alert{inconveniente} al tratar las imágenes de esta manera es la \alert{pérdida total} de información espacial de la imagen

Implementando un perceptrón multicapa

El siguiente notebook contiene un ejemplo de clasificador de imágenes usando un perceptrón multicapa como red neuronal

Ejercicio: 2.2. Clasificación de dígitos con un perceptrón multicapa.ipynb[1]

[1] https://githubtocolab.com/etsisi/Robotica/blob/main/Notebooks/2.2. Clasificación de dígitos con un perceptrón multicapa.ipynb

\end{figure}
\end{frame}

\begin{frame}{Reducción dimensional en redes convolucionales}
Para formar el \say{\alert{embudo}} de la red se utilizan distintos mecanismos para \alert{reducir las dimensiones} de la información de la red. En concreto los dos mecanismos predominantes son: