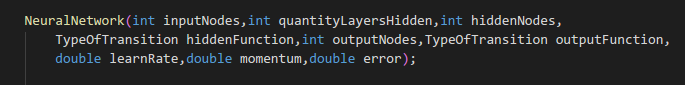
La biblioteca thinker brinda la posibilidad de crear una red neuronal según las preferencias del usuario.

En los siguientes capítulos se explicarán las clases y métodos con los que el usuario interactuara directamente para crear una red neuronal.

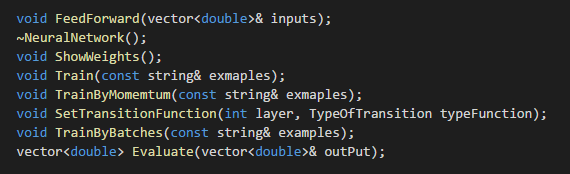
**NEURALNETWORK**

La clase NeuralNetwork nos permite crear una red neuronal en base a nuestros requerimientos.



El constructor recibe los siguientes argumentos:

* **inputNodes:** Cantidad de nodos en la capa de entrada (Depende de la cantidad de variables relevantes en el contexto del problema para alimentar a la red neuronal)
* **quantityLayersHidden:** Cantidad de capas ocultas (No existe un criterio concreto para definir el número de capas ocultas, se define por prueba y error)
* **hiddenNodes:** Cantidad de nodos en la capa oculta (No existe un criterio concreto para definir el número de capas ocultas, se define por prueba y error)
* **HiddenFunction:** Es la función de transición a usar en la capa oculta
* **outputNodes:** Cantidad de nodos en la capa de salida (Depende de la cantidad de posibles resultados en el contexto del problema que puede generar la red neuronal)
* **OutputFunction:** Es la función de transición a usar en la capa de salida
* **learnRate:** Parámetro de aprendizaje (controla el tamaño del cambio de los pesos en cada iteración. No existe un criterio concreto para definir el número de capas ocultas, se define por prueba y error)
* **momentum:** factor momento (acelera la convergencia de los pesos, se elige entre 0.1 y 0.8)
* **error:** Margen de error entre los datos esperados y los obtenidos

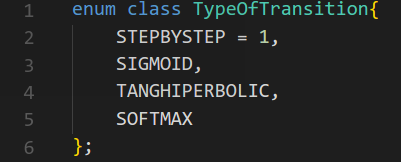


* **Train:** Este método se encarga de entrenar a la red neuronal.
  + **Exmaples**: Es el nombre del archivo json que contiene los datos de entrenamiento
* **ShowWeights:** Este método imprime en consola los pesos en cada capa y sesgos en los perceptrones correspondientes.
* **Evaluate:** Este método genera un resultado en el contexto del problema, en base a los datos pasados como parámetros.
  + **Output:** Es el vector con el resultado de la evaluación

**TYPEOFTRANSITION**

Todas estas funciones de activación están relacionadas matemáticamente.

Si bien hay algunas excepciones, en general, la **función** **hiperbólica** tangente es la mejor opción para la activación de la capa oculta. Para la activación de la capa de salida, si la red neuronal está resolviendo un problema de clasificación, donde la variable dependiente a predecir puede tomar uno de tres o más valores, la **función** **softmax** es la más adecuada. Si la red neuronal está resolviendo un problema de clasificación donde la variable a predecir puede tomar exactamente uno de dos valores, la **función** **sigmoidal** es la más conveniente.



* STEPBYSTEP
* SIGMOID
* TANGHIPERBOLIC
* SOFTMAX

**Ejemplo**

A continuación, se explica un tipo de problema típico que puede ser resuelto usando redes neuronales.

**Problema:** En base a los datos de la longitud y ancho del sépalo y pétalo de una flor de iris. Determinar si pertenece a la especie virginica, setosa o versicolor.

Las variables usadas para hacer la predicción pueden ser llamadas **variables independientes, atributos, características o valores-x**. La variable a predecir puede ser llamada **variable dependiente o valor-y.**

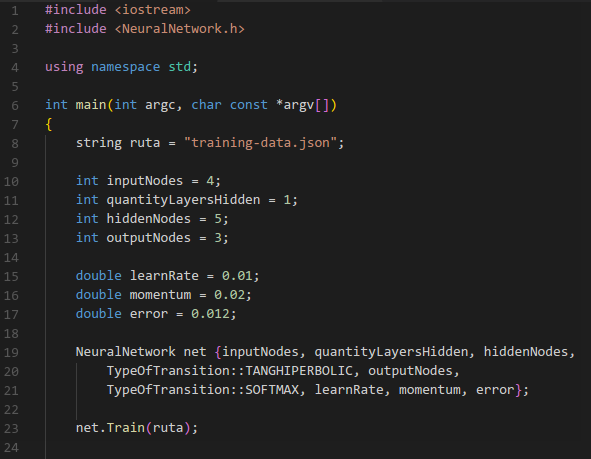
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Largo del sépalo | Ancho del sépalo | Largo del pétalo | Ancho del pétalo | Especie |
| 5.1 | 3.5 | 1.4 | 0.2 | I. Setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | I. Setosa |
| 7.0 | 3.2 | 4.7 | 1.4 | I. Versicolor |
| 6.3 | 3.3 | 6.0 | 2.5 | I. Virginica |

El tipo de problema a resolver es llamado **problema de clasificación** porque el valor de “y” puede tomar uno de tres valores posibles: setosa, versicolor o virginica.

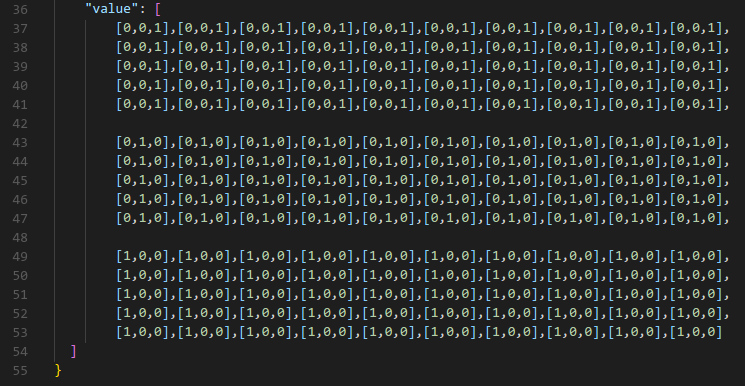
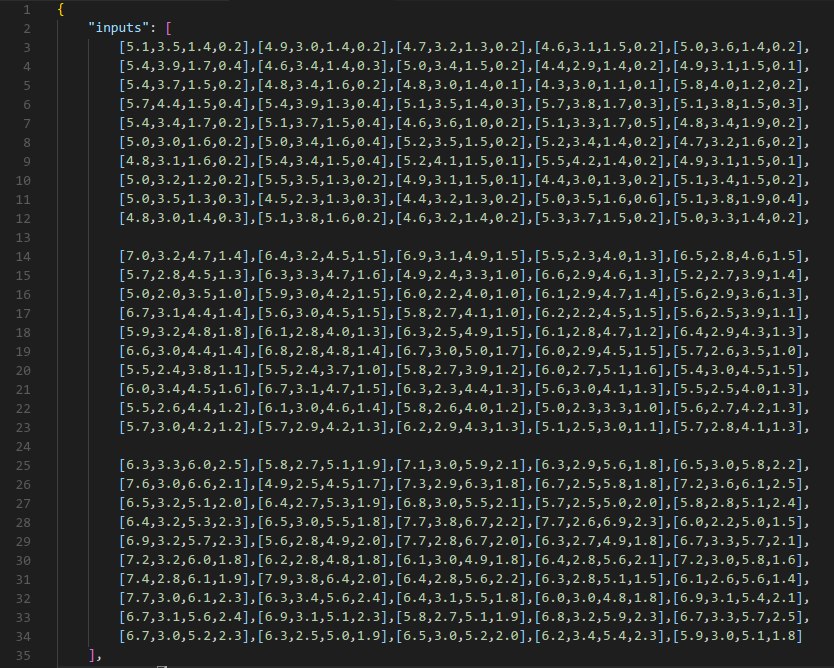
Las redes neuronales requieren datos con **valores-y** conocidos, llamados **datos de entrenamiento**. En la figura 1-a solo hay cuatro datos de entrenamiento. En un escenario de la vida real, es probable que los datos de entrenamiento sean cientos o miles.

Para hacer uso de la biblioteca thinker debemos incluir el archivo **NeuralNetwork.h**

Primero se debe instanciar un objeto de la clase NeuralNetwork.



Para entrenarla se debe llamar al metodo **Train** y pasarle como parametro el nombre del archivo json que contiene los datos de entrenamiento.



* **Inputs:** Contiene un arreglo de arreglos donde cada arreglo contiene los datos de las variables independientes (longitud y ancho de los pistilos y pétalos).
* **Value:** Contiene un arreglo de arreglos, donde cada arreglo contiene el resultado esperado:
  + [0, 0, 1] para setosa
  + [0, 1, 0] para versicolor
  + [0, 0, 1] para virginica

Una vez que la red ya ha sido entrenada, podemos llamar al metodo **Evaluate** para determinar la especie a la que pertenece los datos de una flor pasados como parametros.

