

# Modelado y simulación del Robot Mitsubishi RV-2JA controlado mediante señales electromiográficas

Busso, Francisco Ignacio  
Gautero, Francisco  
Gregoret, Guillermo

Universidad Tecnológica Nacional  
Facultad Regional Santa Fe

30 de septiembre de 2021

# Valores de Referencia

Para generar valores de cada señal EMG se optó por una distribución uniforme con variación del 5 % sobre las señales de referencia.

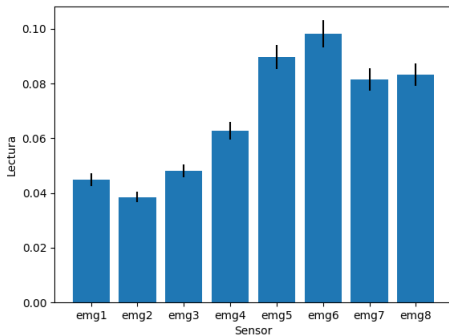


Figura: Valores de Referencia para cada sensor

# Etiquetado del Set de Datos

Se define un valor umbral para determinar si el valor de la lectura de un sensor pertenece a los valores admitidos.

```
def in_range(value, index):  
    x = [0.0449, 0.0385, 0.0481, 0.0628, 0.0897, 0.0982, 0.0814, 0.0833]  
    return x[index] * 0.95 <= value <= x[index] * 1.05
```

Si las lecturas de todos los sensores se encuentran dentro del rango admitido, la entrada se etiqueta como 1; si no, se etiqueta como 0.

```
def tag(row):  
    return 1.00 if all(in_range(row[i], i) for i in range(0, row.shape[0])) else 0.00
```

# Obtención del Set de Entrenamiento

Se lee un set de datos, se lo etiqueta y se lo separa entre un set de entrenamiento y un set de pruebas.

```
>>> import pandas as pd
>>> import numpy as np
>>> from functions import normalize
>>> from sklearn.model_selection import train_test_split
>>> features = pd.read_csv('training-set.csv').to_numpy()
>>> labels = np.array(list(map(lambda x: normalize(x), features)))
>>> X_train, X_test, Y_train, Y_test = train_test_split(features,
                                                         labels,
                                                         test_size=0.50)
```

# Implementación con scikit-learn

La clase `MLPClassifier` implementa un algoritmo de entrenamiento de perceptrón multicapa (*MLP*) utilizando propagación hacia atrás (*Backpropagation*).

```
>>> from sklearn.neural_network import MLPClassifier
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                        hidden_layer_sizes=(20, 10),
                        random_state=1)
```

# El Algoritmo MLP

El algoritmo toma como entrada dos arreglos:  $X$ , de dimensiones  $(n - \text{muestras}, n - \text{características})$ , que representa las muestras de entrenamiento como vectores de punto flotante; e  $Y$  de tamaño  $(n - \text{muestras})$ , que almacena los valores de las etiquetas de clase para las muestras de entrenamiento.

```
>>> clf.fit(X_train , Y_train)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(20, 10),
              random_state=1,
              solver='lbfgs')
```

# Predicciones

Luego del ajuste (entrenamiento), el modelo puede predecir las etiquetas para nuevas muestras

```
>>> clf.predict(Y_test)
array([1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0])
```

# Similitud Coseno

Es una medida de la similitud existente entre dos vectores en un espacio multidimensional (más precisamente un espacio *pre-hilbertiano*) con el que se evalúa el valor del coseno del ángulo comprendido entre ellos.

Proporciona un valor igual a 1 si el ángulo comprendido es cero. Cualquier ángulo existente entre los vectores, el coseno arrojaría un valor inferior a uno.

*La similitud coseno no debe ser considerada como una métrica debido a que no cumple la desigualdad triangular.*



# Complejidad de la Solución

Suponiendo  $n$  muestras de entrenamiento,  $m$  características,  $k$  capas ocultas cada una conteniendo  $h$  neuronas y, para simplificar,  $o$  neuronas en la capa de salida. La complejidad temporal de la propagación hacia atrás es  $O(n * m * h^k * o * i)$ , donde  $i$  es el número de iteraciones.

Dado que el método de Backpropagation tiene alta complejidad temporal, es recomendable comenzar con un número bajo de neuronas en la capa oculta para el entrenamiento.

# Escalabilidad de la Solución

Esta implementación no está pensada para aplicaciones de gran escala. En particular, scikit-learn no ofrece soporte para GPU.