Existen dos tipos de entrenamiento para un modelo de red neuronal: online y offline. La diferencia entre estos dos radica principalmente en la manera en que la red es actualizada y la forma como ingresan los datos de entrenamiento. Los datos en un entrenamiento online normalmente llegan 1 a 1, es decir no se tiene el dataset completo pues al principio no se dispone de datos o simplemente son demasiados y un entrenamiento común (batch o mini-batch) llevaría mucho tiempo ejecutándose. Es por esto que se entrena un dato a la vez, y se ejecuta el algoritmo de optimización con cada dato (grandient descent, Adam, Adagrad, etc), además el learning rate es bajo (0.001 aprox, depende mucho del problema) pues la red prácticamente nunca deja de entrenarse (en problemas en tiempo real, como el descrito en la sección anterior). Para el entrenamiento de estas redes se utilizan optimizadores estocásticos, es decir, ejecutan el algoritmo de optimización cada vez que se recibe un nuevo dato. La idea es que la red aprenda de los nuevos sin olvidar los anteriores. Por otro lado el entrenamiento offline (los típicos con batch y mini-batch) realizan un entrenamiento con el dataset completo, y una vez la red converge podría estar lista para ser utilizada.

Está claro que el tipo de entrenamiento que necesitamos es online, porque la red es alimentada con los datos que le brinde el robot cada vez que ejecute una acción y esta tenga un resultado ya sea positivo o negativo. Como la idea es que el robot nunca deje de aprender, se podría decir que se dispone de un dataset infinito.

Lo primero que se realizó fue un entrenamiento con batch (entrenamiento común, utilizando todo el dataset de entrenamiento con n números de iteraciones) para tener claro tiempos de ejecución, error cuadrático medio de las predicciones realizadas en el dataset de prueba, número de iteraciones para alcanzar convergencia de la red en un tiempo corto (aprox 0.01 segundos por iteración) y, muy importante, obtener una arquitectura de red óptima para comenzar a hacer pruebas con el entrenamiento online. Además se probó con distintos learning rates (0.1, 0.01, 0.001, 0.0001) con el optimizador Gradient Descent. Al final de las pruebas, se decidió tener una arquitectura de red con 4 capas ocultas (128, 64, 64, 32 neuronas respectivamente) y learning rate de 0.01. Los datos de entrenamiento fueron 500 y 165 de validación. Se utilizó el pnode0.

Con esta arquitectura se ejecutó un entrenamiento online. Como algoritmo de optimización se utilizó el Stochastic Grandient Descent y los learning rate fueron variando entre 0.1, 0.01 y 0.001, y las iteraciones entre 5, 10, 50 y 100; para escoger el que diera mejor resultado. El error final(de prueba) que nos dio este entrenamiento no nos satisfizo pues, las predicciones no tenían un error cuadrático medio bajo, la desviación estándar era alta (0,2 aprox) y el tiempo que tomaba cada iteración era alto pues el algoritmo de optimización se ejecutaba n veces (n=número de iteraciones) cada vez que veía un dato, es decir si n = 5 el algoritmo se ejecutaba 1250 veces para terminar el entrenamiento. Con este entrenamiento se tenía el problema que las predicciones no eran certeras y costaba mucho evitar el overfitting o underfitting de los pesos, pues al entrenar con solo un dato n veces resulta difícil llegar a una generalización del problema.

Como no era posible obtener un resultado satisfactorio con este método, se decidió hacer uno que combina el batch con el online. Lo que hace es tener una memoria que actúa como una cola de tamaño t la cual comienza vacía y cada vez que llega un dato nuevo, este es almacenado en ella y se entrena la red con los datos que tiene. Como el dataset puede llegar a ser infinito, la memoria tiene un tamaño máximo (como se dijo anteriormente), entonces si llega un dato nuevo y la memoria está llena, se elimina el primero en entrar y se agrega el nuevo dato. Cuando un dato llega, se ejecuta un entrenamiento online con cada dato existente en la memoria, por cada dato se ejecuta una vez el algoritmo de optimización. Este entrenamiento dio muy buenos resultados pero el tiempo de ejecución era un poco alto (0.00 segundos por iteración) por esto se decidió adaptar el mini-batch. El entrenamiento se realiza como un entrenamiento típico con mini-batch, se escoge el tamaño del batch y se divide la memoria. Si el tamaño del batch es de 5 y la memoria tiene 20 datos, entonces se divide la memoria en 4 partes de 5 datos cada una. Una vez se tienen los datos divididos, se calcula el error cuadrático medio de cada mini-batch y se ejecuta el algoritmo de optimización (en este caso descendiente del gradiante). Por ejemplo: Para un número de iteraciones n=10, 20 datos en memoria y mini-batch de tamaño 5; se ejecuta 20 veces el algoritmo de optimización, mientras que sin mini-batch se ejecutaría 100 veces. El tiempo de ejecución fue más rápido (0.000 s por iteración) pero se sacrifica un poco de precisión en las predicciones (0.00 aprox).

Como el algoritmo dio buenos resultados, se procedió a encontrar una arquitectura de red óptima para el problema, se probaron redes con 2, 3, 4 y 5 capas escondidas con M cantidad de neuronas cada una. Cuando se terminaron las pruebas, la red que mejor funcionaba tenía 4 capas escondidas con 128, 64, 64, 32 neuronas respectivamente. Luego de escoger la red, se investigó sobre optimizadores estocásticos que ayudaran a mejorar la solución del problema. Se decidió utilizar Adam porque tiene técnicas que ayudan a olvidar poco la información con la que se ha entrenado anteriormente. Con este nuevo algoritmo se hicieron pruebas nuevamente con un learning rate de 0.1, 0.01 , 0.001 y 0,0001. El mejor resultado lo dio el learning rate 0.0001, además de mejorar el error de prueba, las oscilaciones en la gráfica tanto de entrenamiento como de prueba, eran pequeñas comparadas con las de los otros. Un learning rate más pequeño causaba underfitting.

Con estas pruebas se decidió utilizar para cada pnode, una red neuronal con 4 capas escondidas (128, 64, 64, 32 neuronas respectivamente), el optimizador Adam con un learning rate de 0,0001, tamaño de memoria 100, mini-batch de 5 y 25 iteraciones por cada uno.

Finalizando, se realizaron entrenamientos con los 21 pnodes y los resultados fueron los siguientes:

\*