Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Aplicación de deep learning al aprendizaje de modelos de mundo en mecanismos cognitivos

Informe final de práctica profesional

Bachillerato en Ingeniería en Computación

Ariel Antonio Rodríguez Jiménez

2014053647

Desarrollado en Grupo Integrado de Ingeniería, Universidade Da Coruña

Coruña, España; junio 2018

Resumen ejecutivo

El Grupo Integrado de Ingeniería ha estado trabajando en el MDB [1] desde hace más de 10 años. Este se compone de 5 grandes bloques que contienen modelos matemáticos cada uno de ellos y se quieren aprender por medio de mecanismos de aprendizaje automático. El proyecto nace porque el método que está utilizando la el grupo para obtener este aprendizaje es costoso y los resultados que han tenido no han sido buenos. Es por esto que se propuso tratar de resolver el problema mediante Deep Learning.

Lo primero que se hizo fue tener un objetivo al cual tratar de llegar (tener un modelo cuyas predicciones sean casi perfectas). Para esto se decidió entrenar una red utilizando Mini-Batch, y los resultados obtenidos fueron excelentes. A partir de ese objetivo, se comenzaron a probar técnicas de aprendizaje online que simularan el mundo real en el cual se encontraría el robot. Los resultados fueron buenos y el problema fue resuelto, como se esperaba.

Contenido

Descripción del problema

Actualmente se está trabajando con modelos de mundo que aprenden en tiempo real a predecir acciones que en un futuro les puede generar una pérdida o una ganancia. Llámese pérdida a las acciones que estarían mal hechas, por ejemplo: tener como objetivo agarrar una bola, pero que la acción decidida por el robot sea simplemente golpearla; una acción que le produzca ganancia sería agarrar la bola (es el objetivo inicial).
Las acciones que se pueden ejecutar por el robot hasta ahora son las siguientes:

* Agarrar objeto
* Pedir favor (Ejemplo: Pedir que acerquen una caja)
* Cambiar objeto de mano
* Poner objeto en una caja
* Arrastrar objeto
* Agarrar con ambas manos
* Tirar
* Poner el objeto en el robot

El objetivo es poder aprender basándose en los resultados que me dieron las acciones que he ejecutado anteriormente. Es decir, saber que si mi objetivo era agarrar la bola y lo que hice fue golpearla, aprender que esa acción para este objetivo, no me puede traer ningún beneficio entonces tiene una probabilidad baja de ejecución, mientras que agarrar la bola tiene una probabilidad alta pues anteriormente me dio un buen resultado.

Cada acción que se quiere predecir va a depender completamente del contexto actual en el que se encuentre el robot. Porque el ambiente es muy poco probable que sea idéntico a los anteriores, es por esto que las predicciones son probabilidades ([0, 1] probabilidad de ejecutar o no una acción).

La idea es que por cada acción (llámese acción a un movimiento que pueda ejecutar el robot como mover una mano) haya un mecanismo de aprendizaje automático que pueda decidir qué tan probable es que sea una buena para lograr un objetivo. Podemos decir que un objetivo puede componerse de una o más

acciones, es por esto que las anteriores dependen de las siguientes, y se debe de aprender de estas, es decir: olvidar lo menos posible.

Dentro de la memoria de largo plazo del MDB[1] existen modelos de aprendizaje automático cuya función es predecir si ejecutar o no cierta acción (como mover alguna parte, o puede ser incluso un grupo de acciones); estos son llamados pnodes. Los pnodes reciben información del contexto actual y predicen la probabilidad de que la acción que les corresponde sea buena o mala para el objetivo propuesto. Ejemplo: Para agarrar una bola que se encuentra al lado derecho del robot, es necesario activar los pnodes que tienen como objetivo predecir si: mover el hombro derecho, mover la mano derecha, agarrar un objeto; van a generar alguna recompensa. Para poder realizar este trabajo, cada pnode recibe la siguiente información:

* ball_in_right_hand: Booleano que indica si el robot tiene la bola agarrada con la mano derecha, en metros.
* ball_dist: Distancia que hay entre el robot y la bola, en metros.
* box_size: Tamaño de la caja, en metros.
* ball_in_left_hand: Booleano que indica si el robot tiene la bola agarrada con la mano izquierda.
* box_ang: Ángulo del robot a la caja, en radianes de -pi a pi.
* ball_ang: Ángulo del robot a la bola, en radianes de -pi a pi.
* box_dist: Distancia que hay entre el robot y la caja, en metros.
*ball_size: Tamaño de la bola, en metros.

Como se puede observar, los objetivos del robot para estos pnodes serían meter la bola en una caja, mover una caja, agarrar una bola, etc. Hasta el momento, se ha trabajado con 21 pnodes que cada uno ejecuta una acción de las mencionadas anteriormente.

Es importante saber que el aprendizaje del robot es en tiempo real, los datos van ingresando 1 por 1 al modelo de aprendizaje automático, y se necesita algún mecanismo que ayude a aprender el modelo matemático del problema.

Para poder retornar buenas acciones, lo que se hace es: hacer una predicción, y posteriormente recibir una respuesta indicando si la acción realizada estuvo buena o mala. De esta manera, entrenar el modelo de predicciones (cada pnode que participó en la acción) para ajustarlo y aprender de la acción realizada.

El tiempo que le toma al robot realizar una acción puede ser bastante largo (1 min, por ejemplo), por ello lo que se hizo fue simular la ejecución de acciones por parte del robot, y almacenar los datos obtenidos para poder entrenar un modelo de aprendizaje automático. De esta manera se obtuvo un dataset por cada pnode para poder entrenar y probar los modelos. (Cada dataset con datos entre 250 y 1000 aproximadamente)

Hasta ahora, se ha estado utilizando un algoritmo evolutivo llamado Neat como el modelo de apredizaje automático (cada pnode). El algoritmo lo que hace es buscar una topología de red neuronal óptima para la predicción de los datos. Lo que se quiere hacer es buscar un método de Deep Learning que mejore el resultado de las predicciones, olvide muy poco para no perder información y que cada vez que entrene le tome poco tiempo; pues es un trabajo que realiza en tiempo real (como se mencionó anteriormente).

Solución implementada

Existen dos tipos de entrenamiento para redes neuronales: online[4] 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 para su entrenamiento. Es por esto que se entrena un dato a la vez, y se ejecuta el algoritmo de optimización con cada dato (Stachastic Grandient Descent, Adam, Adagrad, etc), además el learning rate es bajo (~0.001, 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 y mini-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 (~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 Adam[2] (pues funciona muy bien para entrenamiento estocásticos), los learning rate fueron variando entre 0.1, 0.01 y 0.001; y los pasos de entrenamiento 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 tenían un error cuadrático medio alto (~0.16) y la desviación estándar era alta (~0,31 aprox), pero el tiempo que tomaba cada iteración era bajo (~0.009 por iteración, único punto a favor). 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 modelo.

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 con error cuadrático medio de ~0.02 y desviación estándar de ~0.14 (en el test final); pero el tiempo de ejecución era un poco alto (~0.7 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 ADAM). 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 (~0.17s por iteración), el error final y desviación también disminuyeron (~0.01 y ~0.12 respectivamente).

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 pasos de entrenamiento por cada uno.

Las gráficas de predicción de los pnods 0, 7 y 20; se encuentran en la sección de anexos.

Conclusiones y comentarios

Cumplimiento de objetivos propuestos

Sí se cumplió con los objetivos propuestos pues se pudo encontrar un método de entrenamiento online que lograra aprender en tiempo real. Sus predicciones son muy buenas y además el tiempo de ejecución es bajo.

Si se compara el método resultante con el que se estaba utilizando, hay una mejora positiva en cuanto a rendimiento y eficiencia.

Productos entregados

Se entregó el código con los entrenamientos Batch, Mini-Batch, entrenamiento estocástico, estocástico con memoria y estocástico con memoria y Mini-Batch. Además de clases para guardar datos en un csv. Se codificó un programa que grafica un csv con la librería matplotlib.

Se entregaron bitácoras de varios entrenamientos, incluyendo el entrenamiento final de cada pnode, se realizó una validación cruzada con el pnode0 para comprobar la generalización en el aprendizaje de la red.

Las bitácoras incluyen:

- 3 archivos csv con los siguientes datos:
    - Error de entrenamiento y test por iteración (Formato: iteración, entrenamiento,prueba)
    - Tiempo de cada iteración (Formato: iteración,segundos)
    - Predicciones finales (Formato: registro,predicción,etiqueta)
- 3 gráficas mostrando los siguientes datos:
    - Curvas de entrenamiento y test
    - Tiempos por iteración
    - Gráfica de dispersión de las predicciones
- Un archivo que contiene información de cada entrenamiento:
    - Número de entrenamiento
    - Tamaño de memoria utilizado

- Pasos de entrenamiento
- Valor del learning rate
- Error cuadrático medio de entrenamiento
- Raíz del error cuadrático medio de las pruebas
- Raíz del error cuadrático medio del test final
- Desviación estándar del test final
- Optimizador utilizado
- Número de neuronas por cada capa de la red neuronal
- Número del pnode utilizado para entrenar
- Nombre de las funciones de activación utilizadas por capa
- Booleano que indica que se utilizó Dropout en las capas intermedias
- Tipo de entrenamiento utilizado

El entrenamiento aún está en prueba, los resultados que se han obtenido han sido buenos pero se tiene que seguir comparando con otros datasets y distintos hiperparámetros para ver sus comportamiento ante escenarios distintos.

El método de eliminar el elemento de la memoria cuando está llena, puede ser distinto a una cola, se pueden probar otras técnicas como eliminar datos repetidos.

Se puede crear una función de coste para la red que combine el error cuadrático medio y la desviación estándar. (en caso de los entrenamientos con mini-batch o batch).

Se puede realizar la integración con el MDB, utilizando TensorFlow.

Experiencias adquiridas

Mi trabajo realizado en el GII de la UDC me brindó mucho conocimiento en materia de Deep Learning y principalmente entrenamiento online. El campo aún se encuentra en investigación y cada vez hay más técnicas que ayudan al buen funcionamiento de las redes neuronales profundas.

Aprendí que lidiar con problemas cuya solución tiene que estar optimizándose en tiempo real, conlleva muchos detalles que tomar en cuenta, como tiempo de

ejecución y métodos de aprendizaje distintos a los habituales pues no se disponen de datos previos conocidos.

El proyecto cumplió con sus objetivos y se realizó lo descrito en el anteproyecto.

Fue una experiencia realmente enriquecedora viéndolo desde un ángulo técnico y social.

Anexos

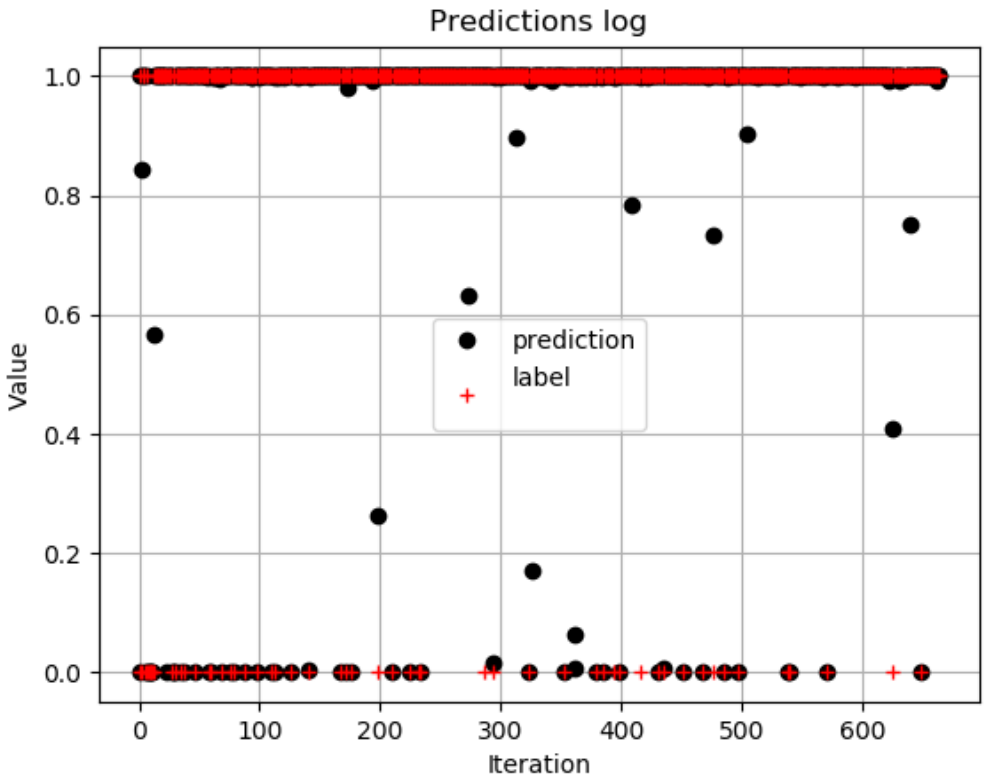

Imagen 1: Gráfica de predicciones del pnode0. Los puntos negros son las predicciones y las cruces rojas son las etiquetas.

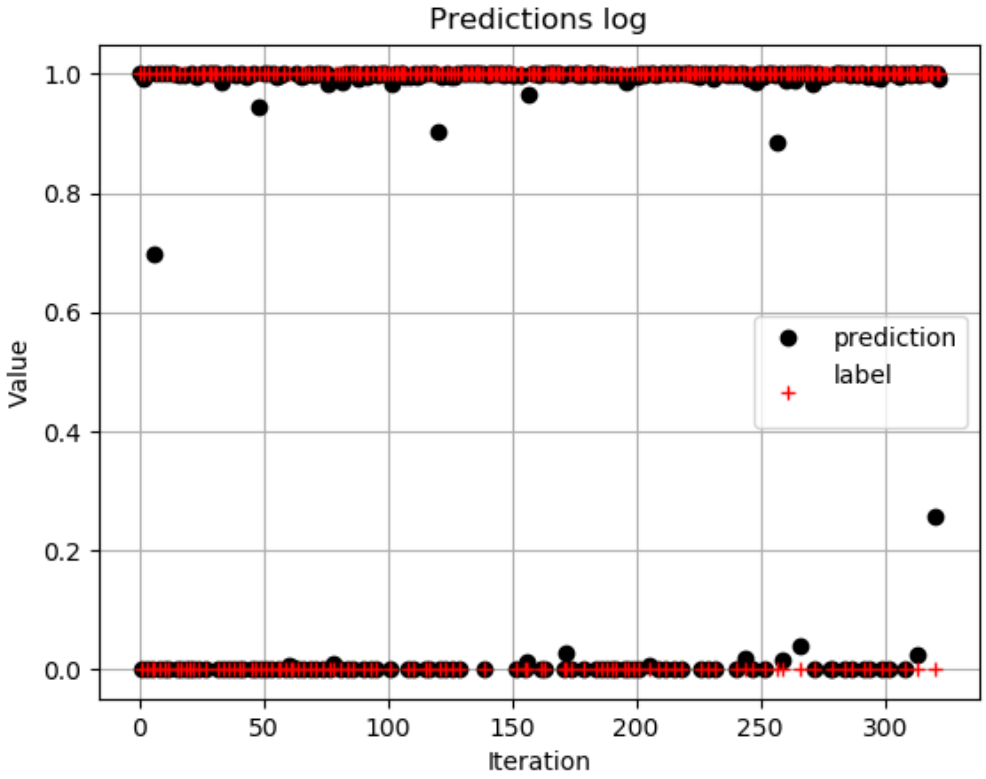

Imagen 2: Gráfica de predicciones del pnode7. Los puntos negros son las predicciones y las cruces rojas son las etiquetas.

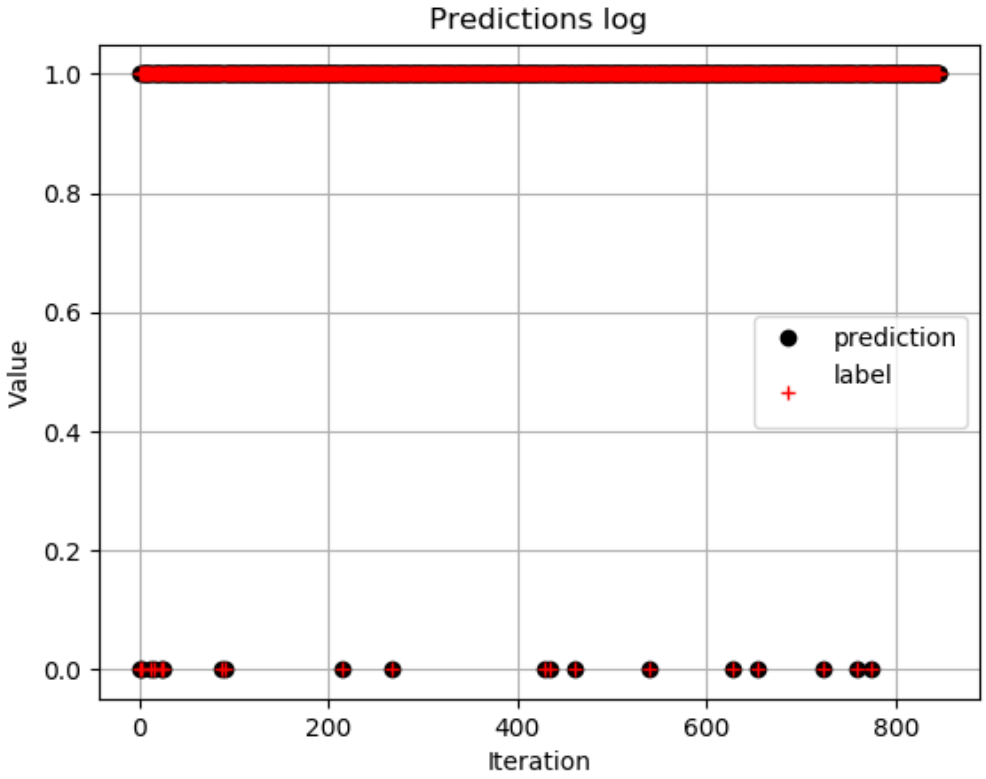

Imagen 3: Gráfica de predicciones del pnode20. Los puntos negros son las predicciones y las cruces rojas son las etiquetas.

Bibliografía

[1] Bellas, F. (2003). MDB: Mecanismo cognitivo darwinista para agentes autónomos.

[2] Kingma, D., y Lei J. (2015). ADAM: A method for stochastic optimization.

[3] Poggio, T., Voinea, S., y Rosasco, L. (2018). Online learning, stability and stochastic gradient descent.

[4] Sahoo, D., Pham, Q., Lu J., y C.H S. (2017). Online Deep Learning: Learning Deep Neural Networks on the Fly.