Generalización de Terrenos con Redes Neuronales Multicapa

Sistemas de Inteligencia Artificial
Instituto Tecnológico de Buenos Aires

Garrigó, Mariano 54393
Raies, Tomás A. 56099
Saqués, M. Alejo 56047

Resumen

Se ha analizado el potencial de una red neuronal multicapa supervisada a los efectos de aprender la forma de un terreno, y, posteriormente, generalizar sobre la misma.

Se ha evaluado una serie de variaciones sobre el algoritmo de *backpropagation*, particularmente *adaptive eta* y *momentum*, como así también combinaciones de ambas técnicas. Asimismo, se han comparado los resultados obtenidos actualizando los pesos de la red tanto de manera incremental como en *batch*.

Por último, se han comparado estas técnicas con el estado del arte en algoritmos de generación aleatoria de terrenos, como el *Diamond-Square Algorithm* (DSA). El objetivo ha sido probar las capacidades de una red neuronal de aprender terrenos con una *apariencia* lo más aleatoria posible, tal como busca lograr DSA.

Palabras clave: Redes multicapa supervisadas, *feature scaling*, *backpropagation*, *adaptive eta*, *momentum*, actualización de pesos incremental / en *batch*, funciones de activación, *Diamond-Square Algorithm*.

1. Descripción del entrenamiento

En esta sección, se discutirán las decisiones tomadas por el Equipo para entrenar redes neuronales. Esto incluye tanto toda la instancia de pre-procesamiento, como así también el entrenamiento propiamente dicho.

1.1. Pre-procesamiento

1.1.1. *Feature scaling*

Dado que las imágenes de las funciones de activación son intervalos en $\mathbb{R}^2$ ($tanh : \mathbb{R} \to [-1, 1]$ y $logistic : \mathbb{R} \to [0, 1]$), si los datos del *set* de entrenamiento no se encuentran dentro de dichos intervalos, cierto ajuste es necesario.

Para ello, se utiliza *feature scaling* para estandarizar muestras en algún intervalo deseado. De todas las variaciones de dicho método, se utiliza el *rescaling* de datos:

$$x_i^{'} = \frac{x_i - min(X)}{max(X) - min(X)}(b - a) + a \qquad (1)$$

Donde X es el espacio de muestras a estandarizar, a y b son las cotas inferior y superior del intervalo de llegada, respectivamente.

1.1.2. Selección de muestras de entrenamiento

Dado que una de las características que se quiere probar es la capacidad de generalización de la red, es razonable que parte de las muestras del terreno se utilicen para verificar que la red, efectivamente, aprendió el problema. Por ende, se ha decidido utilizar el 90 % de las muestras para el entrenamiento, y un 10 % para testeo. Los patrones de cada conjunto se toman de manera no determinística. Se debe notar que la proporción es un dato parametrizable.

1.2. Implementaciones de *backpropagation*

A continuación, se elaborará sobre los diferentes algoritmos utilizados para entrenar las redes neuronales. Las descripciones atacarán fundamentalmente aquellos rasgos distintivos de los mismos, que emanan de decisiones tomadas por el Equipo durante el proceso de desarrollo.

Nota: El error cuadrático medio sobre el conjunto de datos de entrenamiento se calcula, para cada versión del algoritmo, después de ejecutar cada *epoch*.

1.2.1. Algoritmo incremental con *adaptive eta* y *momentum*

Tal como su nombre lo indica, las actualizaciones de pesos en este algoritmo se realizan patrón a patrón. Esto lleva consigo la necesidad de iterar sobre todos los patrones en cada *epoch*, desaprovechándose así las mejoras que realiza *Octave/Matlab* sobre el producto matricial. Esto se verá reflejado en la lentitud de ejecución de cada *epoch* por parte de este algoritmo, relativa a la versión en *batch* que se mencionará a continuación.

En lo que respecta al orden de las muestras de entrenamiento en cada *epoch*, se ha tomado la decisión de presentarlas en un orden no determinístico, dado que se ha observado que, de esta forma, es menos probable que el algoritmo se atasque en mínimos locales.

Por último, se han considerado dos variaciones sobre el proceso de eta adaptativo, las cuales versan sobre si después de k iteraciones en las que el error cuadrático medio descendió consistentemente, dicho contador se restablece tras incrementar el eta o si se prosigue incrementando *epoch* tras *epoch* hasta que haya un incremento en el error. Se ha notado que ambas variaciones presentan comportamientos similares si para la primera versión se establecen valores de k *pequeños* y, para la segunda versión, *grandes*. Para el caso, *pequeños* y *grandes* valores se refieren a cifras en el orden de 10^1 y 10^2 respectivamente. Se ha decidido utilizar la versión que restablece k tras cada incremento.

En lo que respecta a las condiciones de corte, el algoritmo retorna cuando:

1. Se ha llegado al máximo de iteraciones,
2. Se ha llegado al valor del error cuadrático medio deseado,
3. El eta se ha hecho ~ 0.
4. La diferencia entre el error cuadrático medio anterior y el actual es 0.

Nota: El ítem 3. refleja la situación en la que el error ha crecido consistentemente, haciendo que tras sucesivas disminuciones del eta dicho valor se acerque a 0.

1.2.2. Algoritmo en *batch* con *momentum*

En este algoritmo, las actualizaciones de pesos se realizan tras cada *epoch*. A diferencia del caso anterior, esto permite tomar las muestras del set de entrenamiento de manera conjunta, sin tener que iterar por cada patrón. De esta forma, se aprovechan las mejoras que *Octave/Matlab* realizan sobre el producto matricial, haciendo que la ejecución de cada *epoch* sea notoriamente más rápida que en el algoritmo anterior.

En lo que respecta a las condiciones de corte, el algoritmo retorna cuando:

1. Se ha llegado al máximo de iteraciones,
2. Se ha llegado al valor del error cuadrático medio deseado,
3. La diferencia entre el error cuadrático medio anterior y el actual es ~ 0.