Policy Gradients utilizando CUDA aplicado al juego Breakout de ATARI

Amaury Gutiérrez Acosta, Ixchel G. Meza Chávez

May 2, 2017

Introducción

Uso de algoritmos de optimización numéricos en el contexto de aplicaciones de machine learning.

Hay dos tipos de problemas de optimización que surgen en machine learning: - problemas de optimización convexa -derivados de el uso de regresión logística o support vector machines - problemas no convexos y altamente no lineales -derivados del uso de redes neuronales DNN.

En el contexto de machine learning a gran escala se ha tenido gran interés la propuesta de Robbins y Monro de algoritmos estocásticos, como el método de gradiente estocástico SG.

En DNN se describe una función de predicción h cuyo valor es calculado aplicando transformaciones sucesivas a un vector de entrada x. Estas transformaciones son capas o layers. Por ejemplo una capa canónica conectada completamente realiza un cálculo donde x es el vector de entrada, la matriz W y el vector b contienen los parámetros de la capa y s es una función de activación no lineal component-wise.

$$x^{(j)} = s(W_j x^{(j-1)} + b_j)$$

La función sigmoidea $s(x) = \frac{1}{(1+exp(-x))}$ y la función hinge $s(x) = max\{0, x\}$ conocida como ReLU son funciones populares para utilizarse como función de activación s.

El problema de optimización en este contexto involucra un training set $(x_1, y_1)...(x_n, y_n)$ y la elección de una función de pérdida l de tal forma que

$$min \frac{1}{n} \sum_{i=1}^{n} l(h(x_i; w), y_i)$$

Como ya se mencionó dicho problema es no convexo altamente no lineal, sin embargo se han calculado soluciones aproximadas por medio de métodos basados en el gradiente, pues el gradiente del objetivo en la ecuación anterior con respecto al vector parámetro w se puede calcular por medio de la regla de la cadena usando diferenciación algorítmica. Ésta técnica de diferenciación es conocida como *back propagation*.

Los problemas de optimización en machine learning surgen a través de la definición de las funciones de pérdida y predicción que aparecen en medidas de riesgo empírico y esperado que intentamos minimizar.

Métodos de optimización

Los problemas de optimización en machine learning surgen a través de la definición de las funciones de predicción y de pérdida que aparecen en medidas de riesgo esperado y empírico que intentamos minimizar.

Asumimos que la función de predicción h es de forma fija y está parametrizada por el vector real $w \in \mathbb{R}^d$ sobre el cual se realiza la optimización. Formalmente para una funcion dada $h(.,.) : \mathbb{R}^{d_x} \times \mathbb{R}^d \to \mathbb{R}^{d_y}$ consideramos la familia de funciones de predicción

$$H := \{h(.; w) : w \in \mathbb{R}^d\}$$

de tal forma que intentamos encontrar la función de predicción en esta familia que minimiza las pérdidas debido a predicciones inexactas. Para esto asumimos una función de pérdida $l : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$ donde dado

el par de entrada-salida (x, y) produce la pérdida $l(h(x; w), y)$ cuando $h(x; w)$ y y son salidas predecidas y verdaderas respectivamente.

En un escenario ideal, el vector parámetro w se escoge para minimizar la pérdida esperada para cualquier par entrada-salida. Es decir, que asumimos que las pérdidas son medidas con respecto a una distribución de probabilidad P(x,y) lo cual representa la verdadera relación entre entrada y salida. Es decir que asumimos que el espacio entrada-salida $\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ es creado con $P : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to [0, 1]$ y la función objetivo que queremos minimizar es

$$R(w) = \int_{\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}} l(h(x; w), y) dP(x, y) = \mathbb{E}[l(h(x; w), y)]$$

Decimos que $R : \mathbb{R}^d \to \mathbb{R}$ produce el riesgo esperado (pérdida esperada) dado un vector parámetro w con respecto a la distribución de probabilidad P.

Si no se conoce P no se puede intentar minimizar la ecuación anterior, por lo que en la práctica se busca la solución a un problema que involucra un estimado del riesgo esperado R.

Como en aprendizaje supervisado se tiene acceso a un set de muestras entrada-salida de $n \in \mathbb{N}$, el cual es $\{(x_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ con el cual se puede definir la función de riesgo empírico $R_n : \mathbb{R}^d \to \mathbb{R}$ por medio de

$$R_n(w) = \frac{1}{n} \sum_{i=1}^{n} l(h(x_i; w), y_i)$$

Por lo general la minimización de R_n es considerado como el problema práctico de optimización de interés.

Método estocástico de gradiente

En el contexto de minimizar el riesgo empírico R_n con $w_1 \in \mathbb{R}^d$ el método estocástico de gradiente SG se define como

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f_{ik}(w_k)$$

donde para todo $k \in \mathbb{N} := \{1, 2, ...\}$, el índice i_k, correspondiente a la seed $\xi[i_k]$ como el par muestra (x_{i_k}, y_{i_k}), es escogido aleatoriamente de $\{1, ..., n\}$ y α_k es un paso positivo. Cada iteración de este método involucra sólo el cálculo del gradiente $\nabla f_{ik}(w_{ik})$ correspondiente a una muestra. A diferencia de un algoritmo de optimización determinística, en éste método la secuencia de iteración no está determinada únicamente por la función R_n, el punto de inicio w_1 y el paso de la secuencia $\{\alpha_k\}$, sino que $\{\alpha_k\}$ es un proceso estocástico cuyo comportamiento está determinado por la secuencia aleatoria $\{i_k\}$. Aunque cada dirección $-\nabla f_{i_k}(w_k)$ pueda no ser descendiente de w_k al producir una derivada direccional negativa para R_n de w_k, si es una dirección descendiente *en espera*, entonces la secuencia $\{w_k\}$ puede ser guiada hacia un mínimo de R_n.

Método batch de gradiente

El método más simple es el algoritmo *steepest descent* o *gradiente batch* definido con la iteración

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla R_n(w_k) = w_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(w_k)$$

donde el cálculo del paso $-\alpha_k \nabla R_n(w_k)$ es más costoso que el del método estocástico $-\alpha_k \nabla f_{i_k}(w_k)$, sin embargo se podría esperar un mejor paso cuando se consideran todas las muestras en la iteración.

Al utilizar métodos tipo batch se tienen a disposición una serie de técnicas de optimización no lineal que han sido desarrolladas a lo largo de varias décadas, incluyendo los métodos full gradient, gradiente acelerado, gradiente conjugado, quasi-Newton y Newton inexacto. Además gracias a la estructura de suma de R_n el método batch puede ser paralelizado pues la mayoría de los cálculos son para evaluar R_n y ∇R_n.

Método de gradiente estocástico

En general la función objetivo es el riesgo esperado o empírico

$$F : \mathbb{R}^d \to \mathbb{R}, R(w) = \mathbb{E}[f(w;\xi)] or R_n(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

Para ambos casos aplica el análisis, la única diferencia es la forma en que se escogen los estimados del gradiente estocástico en el método.

El algoritmo es el siguiente:

1. Escoger una iteración inicialw_1
2. Para $k = 1, 2, ...$ hacer
    3. Generar una realización de una variable aleatoria ξ_k
    4. Calcular un vector estocástico $g(w_k, \xi_k)$
    5. Escoger un tamaño de paso $\alpha_k > 0$
    6. Establecer la nueva iteración como $w_{k+1} \leftarrow w_k - \alpha_k g(w_k, \xi_k)$
3. fin de bucle

El algoritmo establece la existencia de tres herramientas computacionales:

1. Un mecanismo para generar una realización de una variable aleatoria ξ_k, donde $\{\xi_k\}$ representa una secuencia de variables aleatorias mutuamente independientes
2. Dada una iteración $w_k \in \mathbb{R}^d$ y la realización de ξ_k, existe un mecanismo para calcular un vector estocástico $g(w_k, \xi_k) \in \mathbb{R}^d$
3. Dado un número de iteración $k \in \mathbb{N}$, existe un mecanismo para calcular un paso escalar $\alpha_k > 0$

Se deben considerar varios aspectos del algoritmo general: primero, el valor de la variable aleatoria ξ_k se considera sólo como una semilla para generar una dirección estocástica, por lo que escogerla puede verse como la elección de una muestra de entrenamiento como en el método simple de SG o puede representar un set de muestras como en el método mini-batch SG. Segundo, $g(w_k, \xi_k)$ puede representar un gradiente estocástico como por ejemplo un estimador imparcial de $\nabla F(w_k)$ como en el método clásico de Robbins y Monro, o puede representar una dirección estocástica Newton o quasi-Newton. Por lo que $g(w_k, \xi_k)$ puede ser

$$\nabla f(w_k; \xi_k)$$

o

$$\frac{1}{n_k}\sum_{i=1}^{n_k} \nabla f(w_k; \xi_{k,i})$$

o

$$H_k \frac{1}{n_k}\sum_{i=1}^{n_k} \nabla f(w_k; \xi_{k,i})$$

Tercero, el algoritmo de SG permite varias opciones de la secuencia de tamaños de pasos $\{\alpha_k\}$, puede ser fija o reducir paulatinamente. Por último, el algoritmo de SG cubre técnicas de aprendizaje activo donde la iteración w_k influye en la selección de la muestra.

Complejidad computacional

Cuando se va a ejecutar un algoritmo para aprendizaje para grandes cantidades de datos la complejidad computacional juega un papel importante y es un factor limitante. Se necesitan algoritmos de aprendizaje que escalen de forma aproximadamente lineal con el volumen total de datos así como con su tiempo de cómputo.

Los mejores algoritmos de optimización no son necesariamente los mejores algoritmos de aprendizaje. Incluso se puede decir que ciertos algoritmos se desempeñan bien independientemente de la razón asumida para el error estadístico de estimación.

Backpropagation

Es un algoritmo muy popular en el aprendizaje de redes neuronales, porque es conceptualmente simple, eficiente computacionalmente y porque si funciona normalmente.

Al diseñar y entrenar una red que usa backpropagation se debe escoger arbitrariamente el número de nodos, capas, razones de entrenamiento, sets de entrenamiento y de prueba entre otras. Estas elecciones son críticas y dependen en gran medida del problema y de los datos disponibles.

Un gran número de aproximaciones a machine learning automático se categorizan en la sección de métodos de aprendizaje basados en el gradiente.

La forma más simple de machine learning multicapas con aprendizaje basado en gradiente es un conjunto de modulos, cada uno de los cuales implementa una función $X_n = F_n(W_n, X_{n-1})$ donde X_n es un vector representando la salida del módulo, W_n es el vector de parámetros modificables en el módulo y X_{n-1} es el vector de entrada al módulo. La entrada X_0 al primer módulo es el patrón de entrada Z^P. Si la derivada parcial de E^P con respecto de X_n es conocida, entonces la derivada parcial de E^P con respecto de W_n y X_{n-1} se puede calcular usando la recurrencia backward

$$\frac{\partial E^P}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, X_{n-1})\frac{\partial E^P}{\partial X_n}$$

y

$$\frac{\partial E^P}{\partial X_{n-1}} = \frac{\partial F}{\partial X}(W_n, X_{n-1})\frac{\partial E^P}{\partial X_n}$$

donde $\frac{\partial F}{\partial W}(W_n, X_{n-1})$ es el jacobiano de F respecto de W evaluado en el punto (W_n, X_{n-1}) y $\frac{\partial F}{\partial X}(W_n, X_{n-1})$ es el jacobiano de F respecto de X. Cuando dichas ecuaciones se aplican a los módulos en reversa, es decir desde la capa N a la 1, se pueden calcular todas las derivadas parciales de la función de pérdida con respecto de todos los parámetros. La forma de calcular los gradientes se conoce como backpropagation.

El caso de redes neuronales multicapas es un caso especial del sistema anterior, donde los módulos son capas alternadas de multiplicaciones de matrices (los pesos) y función sigmoidea component-wise (las unidades), es decir

$$Y_n = W_n X_{n-1}$$
$$X_n = F(Y_n)$$

donde W_n es una matriz cuyo número de columnas es la dimensión de X_{n-1} y el número de renglones es la dimensión de X_n. F es una función vector que aplica la función sigmoidea a cada componente de su entrada. Y_n es el vector de sumas pesadas o entradas totales a la capa n. Aplicando la regla de la cadena a la ecuación anterior, las ecuaciones clásicas de backpropagation en su forma matricial son:

$$\frac{\partial E^P}{\partial Y_n} = F'(Y_n)\frac{\partial E^P}{\partial X_n}$$
$$\frac{\partial E^P}{\partial W_n} = X_{n-1}\frac{\partial E^P}{\partial Y_n}$$
$$\frac{\partial E^P}{\partial X_{n-1}} = W_n^T\frac{\partial E^P}{\partial Y_n}$$

La forma más simple de aprendizaje (minimización) es el algoritmo de descenso de gradiente, donde W se ajusta iterativamente de la siguiente manera:

$$W(t) = W(t-1) - \eta\frac{\partial E}{\partial W}$$

En el caso más sencillo η es una constante escalar. En caso más sofisticados η es variable. En otros métodos *eta* es de la forma de una matriz diagonal o es un estimado de la matriz Hessiana inversa de la función de pérdida, como en los métodos de Newton y Quasi-Newton.

En cada iteración de la ecuación anterior se requiere utilizar todo el dataset para calcular el promedio del gradiente verdadero, lo cual se hace utilizando el método batch, a diferencia del método estocástico, donde se escoge un solo ejemplo $\{Z^t, D^t\}$ aleatoriamente del set de entrenamiento en cada iteración t, y se calcula un estimado del gradiente verdadero basado en el error E^t de ese ejemplo y entonces se actualizan los pesos de la siguiente forma

$$W(t+1) = W(t) - \eta \frac{\partial E^t}{\partial W}$$

Debido a que la estimación del gradiente es ruidosa, los pesos podrían no moverse precisamente por el gradiente en cada iteración, lo cual puede tener ventajas, pues es común que redes no lineales tengan varios mínimos locales de diferentes profundidades, por lo que con el ruido los pesos podrían saltar de un mínimo a otro y encontrar uno más profundo. Las ventajas de aprendizaje estocástico son que es mucho más rápido que el parendizaje tipo batch, muchas veces tiene mejores resultados y se puede usar para rastrear cambios.