IMT2112 Semestre 2024-2
Tarea 4

Elwin van 't Wout

October 23, 2024

Introducción

Uno de los desafíos más importantes dentro del área de visión por computadores es la clasificación de imágenes. Un ejemplo interesante es el reconocimiento automático de carácteres como letras y números, lo cual tiene aplicaciones en lectores de patentes de autos o en la digitalización de documentos escritos a mano. En esta tarea, analizaremos una base de datos con números escrito a mano, llamado MNIST. El objetivo específico de esta tarea es detectar si un número es par o impar con el algoritmo de regresión logística.

La regresión logística es un algoritmo tradicional para la clasificación binaria y puede ser interpretado como una red neuronal con una capa de entrada y salida pero sin capas ocultas. Supongamos que $\mathbf{x}^{(i)} \in \mathbb{R}^n$ es el registro i en la base de datos. En nuestro caso, un registro corresponde a una imágen representada por un vector. El objetivo es predicir la etiqueta $y^i \in \{0, 1\}$ para cada registro, lo cual representa un número par o impar acá.

Supongamos que tenemos m registros para entrenar el clasificador. Cada vector de entrada $\mathbf{x}^{(i)}$ tiene dimensión n y el objetivo del entrenamiento es optimizar los parámetros de la red neuronal. En regresión logística, los parámetros son los pesos (*weights*) $\mathbf{w} \in \mathbb{R}^n$ y el *bias* $b \in \mathbb{R}$. La parte llamada *forward propagation* significa calcular

$$\begin{aligned} z^{(i)} &= \mathbf{w} \cdot \mathbf{x}^{(i)} + b, \\ a^{(i)} &= \sigma(z^{(i)}) \end{aligned}$$

para cada registro i, con

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

la función *sigmoid* como función de activación. Dado que $0 < \sigma(z) < 1$ para todo z real, podemos comparar esta salida $a^{(i)}$ con la etiqueta real $y^{(i)}$ del registro de entrenamiento. En específico, la función de pérdida es elegida como

$$\mathcal{L}(a, y) = -\left(y \log(a) + (1 - y) \log(1 - a)\right).$$

Además, la función de costo es el promedio de las pérdidas de cada registro de entrenamiento:

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(a^{(i)}, y^{(i)}\right).$$

La parte llamada *backward propagation* se trata de calcular las derivadas de la función de costo para cada parámetro. Se puede demostrar que

$$\frac{\partial \mathcal{L}}{\partial w_j} = x_j(a - y),$$
$$\frac{\partial \mathcal{L}}{\partial b} = a - y$$

para $j = 1, 2, \ldots, n$ los componentes de los vectores. Ahora, optimizar los parámetros con el método iterativo de descenso por gradiente (*gradient descent*) requiere hacer un *update* de los parámetros como

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial J}{\partial \mathbf{w}}, \quad b \leftarrow b - \alpha \frac{\partial J}{\partial b}$$

con $\alpha \in \mathbb{R}$ el *learning rate*. Se sigue iterando sobre los pasos de *forward* y *backward propagation* hasta llegar a una red neuronal optimizada.

Dado que la base de datos podría ser grande, la implementación del algoritmo debe ser vectorizado para mejorar el rendimiento. Por este motivo, se coloca todas las entradas como columnas de una matriz grande

$$X = \begin{bmatrix} | & | & & | \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \ldots & \mathbf{x}^{(m)} \\ | & | & & | \end{bmatrix}$$

y las etiquetas en un vector

$$\mathbf{y} = \begin{bmatrix} y^{(1)} & y^{(2)} & \ldots & y^{(m)} \end{bmatrix}.$$

Eligiendo parámetros iniciales $\mathbf{w}_0$ y b_0, p.ej. todos iguales a cero, la regresión logística consta de la iteración siguiente:

1. $\mathbf{z}_k = \mathbf{w}^T X + b$,
2. $\mathbf{a}_k = \sigma(\mathbf{z}_k)$,
3. $\Delta \mathbf{z} = \mathbf{a}_k - \mathbf{y}$,
4. $\Delta b = (\mathbf{1} \cdot \Delta \mathbf{z})/m$,
5. $\Delta \mathbf{w} = (X \cdot \Delta \mathbf{z}^T)/m$,
6. $b_{k+1} = b_k - \alpha \Delta b$,
7. $\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \Delta \mathbf{w}$,

para $k = 0, 1, 2, \ldots, N_{\text{iter}}$.

Dados los parámetros $\mathbf{w}$ y b de la red optimizada, se puede hacer una predicción para un registro $\hat{\mathbf{x}}$ como

$$\hat{y} = \text{round}\left(\sigma(\mathbf{w} \cdot \hat{\mathbf{x}} + b)\right)$$

en donde 'round' significa redondear hacia el número entero más cercano.

Tarea

Esta tarea contempla una implementación de regresión logística con PyOpenCL.

1. Corren el código de ejemplo en Canvas. Revisen bien lo que pasa en la carga de la base de datos, su preprocesamiento y la creación de *features* y *labels*.

2. Implementen regresión logística en PyOpenCL.

   - Muestren las características de los dispositivos disponibles para PyOpenCL en el sistema del computador.
   - Las 7 instrucciones de regresión logística (ver arriba) deben ser programadas en dos *kernels*, uno para el *forward* y otro para el *backward propagation*. Se puede usar Numpy para el preprocesamiento y postprocesamiento.
   - Eligen el tamaño del grupo de trabajo (*local size*) y calculen el tamaño del dominio (*global size*). Los valores de 32 o 64 son razonables como tamaño de grupo.
   - El código debe funcionar para cualquier tamaño del problema, en especial para cualquiera dimensión, número de registros y tamaño de grupo de trabajo.
   - Expliquen cómo diseñaron la implementación SIMD del algoritmo.

3. Eligen un valor del *learning rate* razonable. Un valor referencial es 0.01.

4. Eligen una cantidad fija de iteraciones, tal que el algoritmo converge a un resultado razonable. Un valor referencial es 1000.

5. Entrenen la regresión logística, es decir, optimizen los parámetros de la red neuronal.

6. Predicen la etiqueta de todos los datos en la base de datos de entrenamiento y testeo con la regresión logística.

   - El cálculo para la predicción debe ser hecho con PyOpenCL en la forma de un *kernel* específico.
   - Expliquen cómo eligieron los tamaños (*local* y *global size*).

7. Calculen el porcentaje de predicciones correctas para la base de entrenamiento y de testeo. Es decir, calculen la precisión de entrenamiento y de testeo.

8. Miden el tiempo de cálculo de regresión logística y discuten las diferencias entre:

   - usar NumPy,
   - usar OpenCL en la CPU,
   - usar OpenCL en la GPU.

Observaciones

No se puede compartir código entre compañeros, tampoco usar código de fuentes externos salvo la página del curso en Canvas. En específico, hay un ejemplo del método de regresión logística y varios ejemplos de PyOpenCL disponible en Canvas que pueden usar para esta tarea.

Para los estudiantes motivados, pueden escribir un número con lapiz y papel, tomar una foto e importarla en Python para predicir su valor con regresión logística. Ojo que deben reducir el tamaño de la foto al mismo tamaño de las imágenes en la base de datos. Por último, la imágen $(1 - \mathbf{x})$ tiene la escala gris invertido. Un humano puede leer este sin problemas. ¿La regresión logística ya entrenada también? (Estas sugerencias son opcionales y no se evaluarán.)

Evaluación

Entreguen todo el código de Python y las respuestas a las preguntas a través de Canvas. Pueden responder las preguntas en el mismo Jupyter notebook o en un informe separado.

Los reglamientos del curso se puede encontrar en Canvas. Se destaca que las tareas deben ser hechas de forma individual.

Sugerencias

Los kernels de OpenCL pueden sobrescribir datos declaradas como entrada de una función.

Los arreglos de PyOpenCL (`pyopencl.array`) tienen la función `get()` para obtener los valores del arreglo y la función `set()` para especificar los valores del arreglo.

Para asegurar que un programa de OpenCL terminó en la GPU, se puede usar el comando `event.wait()`.

Tengan cuidado que el tipo de dato en Python y el kernel son consistente.

NumPy	C
np.int32	int
np.int64	long int
np.float32	float
np.float64	double

En Python, tengan cuidado con dividir números enteros. Si `a` y `b` son *integers*, la división `a/b` entrega un *float*. La división `a//b` entrega un *int*, redondeando hacia abajo. Por lo tanto, `-(-a//b)` es redondear hacia arriba. Además, `a%b` entrega el módulo (*remainder after division*).

Instalación de PyOpenCL

La instalación recomendada es con Anaconda: `conda install pyopencl`. Como alternativa, se puede usar `pip install pyopencl`.

OpenCL driver

La librería PyOpenCL es una interfaz a la librería OpenCL. Por lo tanto, deben instalar y configurar OpenCL también. Anaconda y pip no instalan OpenCL mismo, solo la librería de Python.
Para ver si PyOpenCL está instalado correctamente, corre

```
import pyopencl as cl
cl.get_platforms()
```

En el caso que recibes un error `clGetPlatformIDs failed: PLATFORM_NOT_FOUND_KHR`, este significa que la librería PyOpenCL es instalado correctamente pero que no puede encontrar el driver (ICD) de OpenCL.

Linux

Normalmente, el driver de OpenCL para la GPU se encuentra disponible en la mapa `/etc/OpenCL/vendors`. Deben copiar el archivo `.icd` a la mapa `/FOLDER/TO/ANACONDA/envs/ENVIRONMENT/etc/OpenCL/vendors/` en donde las palabras en mayúsculo depende de tu instalación (corre `which python` para ver la mapa de Anaconda).

PyOpenCL en CPUs

El objetivo de OpenCL es proporcionar una librería que ejecute en cualquier dispositivo. Por lo tanto, también se puede usar OpenCL para hacer cálculos en una CPU. La librería POCL (`conda install pocl`) proporciona funcionalidad para correr OpenCL en CPUs y crea un driver propio de OpenCL.

Google Colab

En Canvas pueden ver un Jupyter notebook que explica la instalación de PyOpenCL en Google Colab y cómo elegir una CPU o GPU como *runtime environment*.