Ajuste y paralelización de una red neuronal

Alex, Ferreira Iglesias

Diego, Caeiro Costoya

Héctor, Aldao Amoedo

Computación concurrente paralela y distribuida
Grupo 1
{alex.ferreira.iglesias,diego.caeiro,hector.aldao}@rai.usc.es

15 de abril de 2024

Resumen

Optimizamos las operaciones que realizaba una red neuronal simple de predicción mediante paralelización por procesos. Hicimos comparativas con la versión sin paralelización a través de pruebas con diferentes conjuntos de datos y analizamos tanto los resultados de las predicciones como los tiempos de procesamiento para diferentes cantidades de procesadores.

Palabras clave: Palabra clave 1, palabra clave 2,...

I. Introducción

El trabajo propuesto en clase consistía en entender cómo usar la red neuronal ELM para después optimizarla a través de paralelización por procesos (sección II). Para ello importamos los conjuntos de datos de Sckit [1] e hicimos pruebas con varios de ellos para ver los resultados de la clasificación(sección III). También comparamos los tiempos entre las versiones secuencial y paralela de la red neuronal para diferentes número de cores (sección IV).

II. Paralelizaciones aplicadas al ELM

Hemos desarrollado una versión alternativa de ELM llamada ELMMP en la que implementamos técnicas de paralelización multiproceso para reducir el tiempo necesario para realizar la clasificación.

Basándonos en la ley de Amdahl, lo primero que hemos hecho ha sido comprobar que partes del código son las que más tiempo tardaban, para centrarnos en paralelizar estas mismas. Para ello, hemos realizado un análisis de tiempos en los cálculos empleados en las funciones de entrenamiento (train) y predicción (predict):

```python
def train(self, X_train, y_train):
    # Calculate the output of the hidden layer

    t_train_hidden_output = time.time()
    hidden_output = self.sigmoid_matrix_multiply(X_train, self.weights_input_hidden, self.bias_hidden)
    t_train_hidden_output = time.time() - t_train_hidden_output

    # Calculate the weights of the output layer using the pseudo-inverse
    t_train_weights_hidden_output = time.time()
    self.weights_hidden_output = self.calculate_pseudo_inverse(hidden_output, y_train)
    t_train_weights_hidden_output = time.time() - t_train_weights_hidden_output

    print(f'Tiempo para train_hidden_output >> {t_train_hidden_output}')
    print(f'Tiempo para train_wieghts_hidden_output >> {t_train_weights_hidden_output}')

def predict(self, X_test):

    # Calculate the output of the hidden layer for the test data

    t_predic_hidden_output = time.time()
    hidden_output = self.sigmoid_matrix_multiply(X_test, self.weights_input_hidden, self.bias_hidden)
    t_predic_hidden_output = time.time() - t_predic_hidden_output

    # Calculate the output of the output layer
    t_predic_output = time.time()
    output = self.matrix_multiply(hidden_output, self.weights_hidden_output)
    t_predic_output = time.time() - t_predic_output

    # Convert the output to a list of lists of sparse matrices
    # Flatten the list comprehension and convert it to a NumPy array
    t_predict_y_pred_array = time.time()
    y_pred_array = np.array([sparse_matrix.toarray().flatten() for row in output for sparse_matrix in row])
    t_predict_y_pred_array = time.time() - t_predict_y_pred_array

    # Reshape the flattened array to have one row per prediction
    predicted_classes = np.argmax(y_pred_array, axis=1)

    print(f'Tiempo para predict_hidden_output >> {t_predic_hidden_output}')
    print(f'Tiempo para predic_output >> {t_predic_output}')
    print(f'Tiempo para predict_y_pred_array >> {t_predict_y_pred_array}')
```

Figura 1: *Descripción de la imagen.*

Cuadro 1: *Valoración de especificaciones técnicas*

TR	INT	ESC	DIS	FIA	TF	MAN
5	10	9	7	7	8	3
COM	**SEG**	**COS**	**TAM**	**CON**	**Treal**	**IS**
1	6	5	10	10	2	6

III. Análisis del ajuste de parámetros del modelo

IV. Análisis del rendimiento computacional

Por último comparamos el tiempo que tardan en obtener los resultados las versiones secuencial y paralela, probando para diferentes números de núcleos

V. Resultados

Se presentan resultados en tablas o en figuras de modo que todas las tablas y figuras tengan un formato similar. Si son datos que nosotros no hemos obtenido, sino que son sacados de libros, artículos o webs debemos indicar la fuente de que fueron obtenidos. Se presentan los resultados.

En los resultados debemos describir cómo hemos realizado los experimentos para que puedan ser reproducidos por terceros. Hay que comentar los resultados antes de pasar a la siguiente sección.

A. Ejemplo subsección

Esta sección puede estar dividida en varias subsecciones.

VI. Conclusiones

El apartado de conclusiones explica qué problema ha sido tratado en el documento, qué fue lo que se hizo y cómo se trabajó. Será una especie de recordatorio de todo el documento.

Se deberán ir desglosando las principales observaciones, conclusiones que podemos extraer y los problemas que se han ido encontrando al ir realizando nuestro trabajo.

Además, este apartado es el adecuado para exponer cuales podrían ser posibles trabajos futuros que completen lo explicado en este trabajo.

Referencias

[1] Sckit learn Página web [online] última visita 10 de abril de 2024. Computer, vol. 38, no. 5, pp. 48–56, 2005.

[2] Standard Performance Evaluation Corporation, http://www.specbench.org, [online] última visita 20 de marzo de 2017.