Tarea 2-TICS579

Deep Learning 2024-2

Profesor: Alfonso Tobar-Arancibia **Ayudante**: María Alejandra Bravo
Fecha de Entrega: 20/10/24 - 23:59 hrs. **Puntos Totales**: 24

INSTRUCCIONES:

- Entregue la tarea en una carpeta comprimida con el siguiente formato: **Tarea_1_AT_MB**. Donde en este caso, **AT** y **MB** corresponden a las iniciales de cada integrante del grupo. Se deben incluir todos los archivos necesarios para ejecutar su Notebook.
- El notebook debe entregarse ejecutado con las celdas **en orden** y **sin errores de ejecución**.
- Recuerde que el código debe ser defendido en una sesión de defensa. **NO COPIE Y PEGUE CÓDIGO** que no entiende.
- No cumplir con las instrucciones implica nota 1.0.

Parte 0: Investigación

- Lea el siguiente **artículo** de Franceso Zapuchini en donde se introduce `nn.Sequential`, `nn.ModuleList` y `nn.ModuleDict`. Entienda su funcionamiento y las recomendaciones dadas por el autor para su uso. Utilizará estas funcionalidades más adelante en la tarea.
- Lea el Quickstart de **Torchmetrics**. Esta librería permite calcular métricas por Batch de manera más eficiente y sin tener que hacer códigos tan complejos. También tendrá que utilizarlo más adelante.

Ojo: **Esta parte no tiene entregable asociado.**

Parte 1: Preparación de los Datos

1. Obtenga el dataset MNIST. Para ello se dispone de un template en el archivo **MNIST.py** que se adjunta en la tarea.
   (a) (1 punto) En el mismo archivo, implemente una separación de Train y Validación utilizando `train_test_split()` de Scikit-Learn. Para ello fije el parámetro `test_size` en **10,000** de modo de obtener **60,000** registros para entrenar y **10,000** para validar.
   (b) (1 punto) Implemente la clase Dataset de Pytorch para convertir los datos en Tensor. La salida de la clase debe ser un Diccionario con claves X para cada fila de un Tensor e y para cada target.
   (c) (1 punto) Aplique la clase Dataset a los datos de entrenamiento y validación según corresponda. Reporte los largos (`len`) del dataset de entrenamiento y validación.

Parte 2: Arquitectura

2. Implemente una Arquitectura de Red Neuronal. Debe implementar **una sóla clase** llamada **MLP** que contenga las siguientes indicaciones en el template entregado en el archivo **model.py**.

   (a) (1 punto) El constructor `__init__()` deberá contener los siguientes parámetros:

   - **hidden_dims**: Una lista con las dimensiones de cada capa.
   - **out_dim**: El número dimensiones de la capa de salida.
   - **bn**: Que su valor por defecto será **None**, esto implica que la red no utilizará BatchNorm. En el caso que este valor sea **"pre"** el BatchNorm se aplica antes de la función de activación. En caso de ser **"post"** el BatchNorm va después de la función de activación. **Hint**: Una buena idea puede ser usar `nn.ModuleDict()`.
   - **activation**: Puede tomar los valores **"relu"** o **"sigmoid"** e indicará qué función de activación utilizar. **Hint**: Una buena idea puede ser usar `nn.ModuleDict()`.
   - **dropout**: Por defecto será **None**, lo que implica que no se aplica Dropout. Si recibe un número entonces corresponderá al Dropout Rate y se debe aplicar el Dropout correspondiente.

   (b) (1 punto) Defina el método `dense_block()` el cual **debe** utilizar `nn.Sequential()` para ir creando cada capa de manera dinámica según los parámetros ingresados. **Hint**: Para llamar este método desde el constructor se debe hacer como `self.dense_block(*)`.

   (c) (1 punto) Finalmente defina la función forward para formular el **forward pass** de la red.

Parte 3: Training Loop

3. Defina la función `train_model()` en el archivo **train.py** adjunto. El loop debe entrenar y validar el modelo utilizando mini-batch y siguiendo las siguientes características.
   (a) (1 punto) `training_params` será un diccionario que contenga los siguientes hiperparámetros de entrenamiento:
      - num_epochs.
      - learning_rate.
      - batch_size.
      - weight_decay.

   (b) (1 punto) Se debe enviar el modelo a la GPU.

   (c) (1 punto) Se debe utilizar como optimizador **Adam** y recibir el **learning_rate** y el **weight_decay**.

   (d) (1 punto) Definir los **dataloader** y recibir el **batch_size** correspondiente.

   (e) (1 punto) Contener el **Training** y **Validation** Loop.

   (f) (1 punto) Reportar **al final de cada Epoch**:
      - El tiempo de cada Epoch.
      - El Train y Validation Loss promedio.
      - El F1-Score asociado. Para ello utilice **Torchmetrics** para calcular métricas acumulativas en cada Epoch.

   El output del entrenamiento debería verse más o menos así:

```
Epoch: 1: Time: 17.85 - Train Loss: 0.6187 - Validation Loss: 0.3342 - Train F1-Score: 0.8513 - Validation F1-Score: 0.9068
Epoch: 2: Time: 18.71 - Train Loss: 0.2956 - Validation Loss: 0.2787 - Train F1-Score: 0.8821 - Validation F1-Score: 0.9116
Epoch: 3: Time: 18.80 - Train Loss: 0.2588 - Validation Loss: 0.2525 - Train F1-Score: 0.8956 - Validation F1-Score: 0.9167
Epoch: 4: Time: 18.67 - Train Loss: 0.2332 - Validation Loss: 0.2267 - Train F1-Score: 0.9039 - Validation F1-Score: 0.9201
Epoch: 5: Time: 18.09 - Train Loss: 0.2153 - Validation Loss: 0.2328 - Train F1-Score: 0.9100 - Validation F1-Score: 0.9220
Epoch: 6: Time: 18.91 - Train Loss: 0.2138 - Validation Loss: 0.2231 - Train F1-Score: 0.9142 - Validation F1-Score: 0.9241
Epoch: 7: Time: 18.57 - Train Loss: 0.1943 - Validation Loss: 0.2039 - Train F1-Score: 0.9179 - Validation F1-Score: 0.9264
Epoch: 8: Time: 18.55 - Train Loss: 0.1794 - Validation Loss: 0.2007 - Train F1-Score: 0.9213 - Validation F1-Score: 0.9277
Epoch: 9: Time: 19.13 - Train Loss: 0.1750 - Validation Loss: 0.1909 - Train F1-Score: 0.9240 - Validation F1-Score: 0.9292
Epoch: 10: Time: 19.15 - Train Loss: 0.1623 - Validation Loss: 0.1905 - Train F1-Score: 0.9266 - Validation F1-Score: 0.9304
```

Figura 1: Log del Proceso de Entrenamiento

   (g) (1 punto) Retornar el modelo, el train loss y el validation loss.

Parte 4: Experimentación

Entregar los resultados al proceso de experimentación en un notebook importando los elementos necesarios de los archivos antes creados. El notebook debe incluir lo siguiente:

(a) (1 punto) Una exploración de datos en la que se sampleen 10 registros del Train y del Validation set. Muestre cada registro sampleado utilizando la función `plot_number()` adjunta en el archivo **utils.py**.

(b) (6 puntos) Entrene un modelo para cada configuración. El entrenamiento **siempre debe partir desde cero** reportando la curva de entrenamiento (use la función `plot_training_curves()` de **utils.py**).

Experimento 1

- Entrene una red con capas ocultas de 128 y 64 dimensiones.
- La red no debe tener ni **Dropout** ni **BatchNorm**.
- La función de activación debe ser **Sigmoide**.
- Entrene en GPU por **10 epochs** utilizando **Batch Size** de 8 y **Learning Rate** igual a la constante de Karpathy.

Experimento 2

- Utilice la misma configuración que el Experimento 1 pero ahora cambie la función de activación por **ReLU**.

Experimento 3

- Replique el Experimento 2 con una capa oculta adicional de 32 dimensiones.
- Además, aplique **BatchNorm** previo a la función de Activación.
- Entrene por **30 epochs en GPU** utilizando un **Batch Size** de 1024.

Experimento 4

- Repita el Experimento 3 pero esta vez utilizando **BatchNorm** después de la función de Activación.

Experimento 5

- Repita el Experimento 4 agregando **Dropout** en cada capa oculta utilizando un Dropout Rate de 0.2.

Experimento 6

- Repita el Experimento 5 y agregue un **Weight Decay** de 0.3.

Parte 5: Discusión

(a) (4 puntos) Comente los resultados obtenidos en el proceso de Experimentación. Comente en términos de:

- Tiempo de Entrenamiento
- Loss Obtenido
- F1-Score Obtenido
- Overfitting
- ¿A qué atribuye el resultado obtenido a cada experimento?