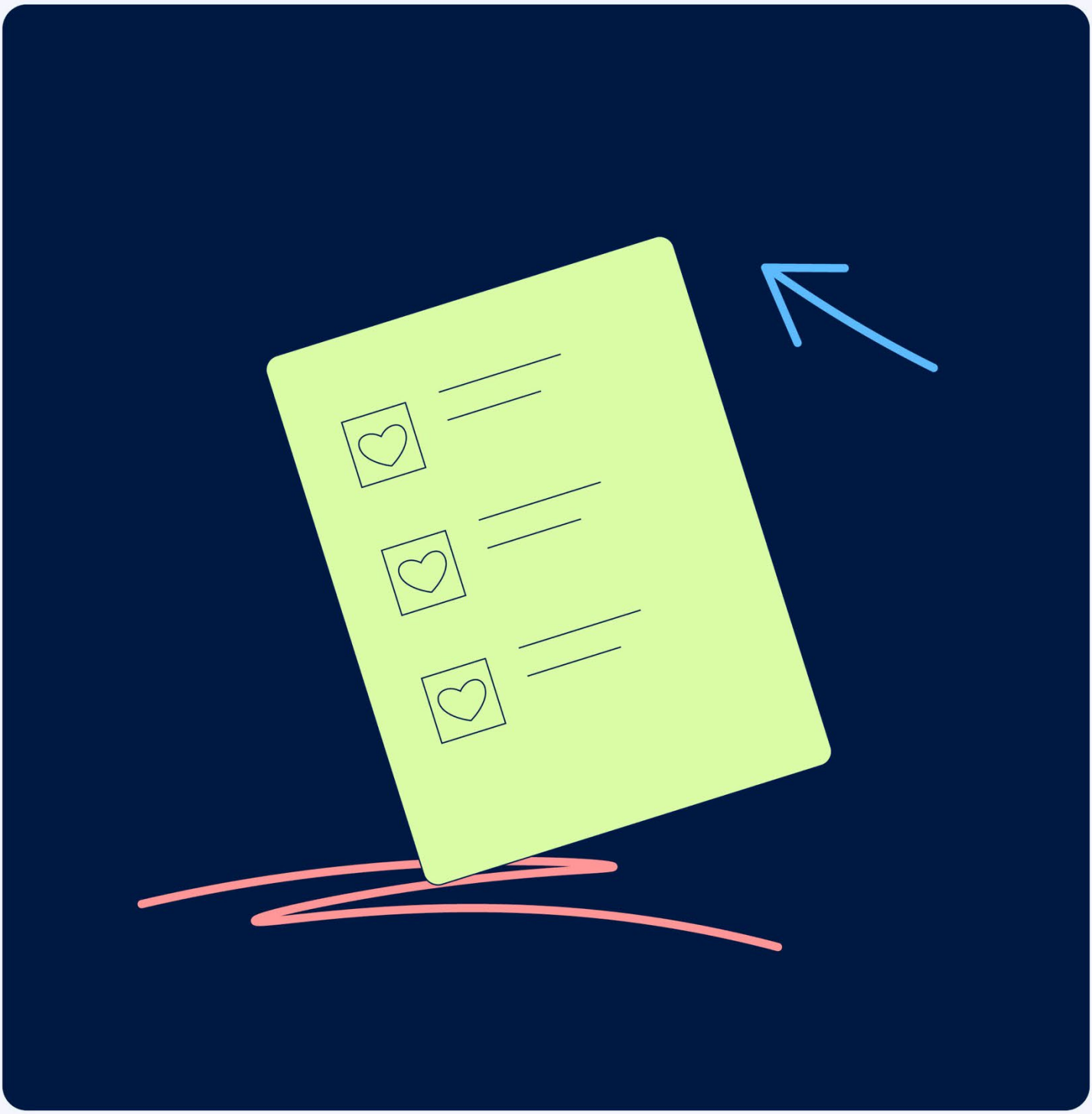


Problema Lab 03

Introducción al Deep Learning

ENTRENAMIENTO Y EVALUACIÓN DE UN MODELO DE DEEP LEARNING

Hemos visto los diferentes pasos del proceso de aprendizaje de un modelo de deep learning, desde la preparación de los datos hasta la evaluación de un modelo entrenado. Ahora vamos a poner en práctica todo este conocimiento.

Usaremos un conjunto de datos para entrenar y evaluar un modelo de deep learning. Aunque hay varios tipos de redes neuronales, para este ejercicio usaremos **un perceptrón multicapa.** En siguientes ejercicios veremos otros tipos de redes neuronales.

El entrenamiento y evaluación de un modelo requiere varios pasos, pero si sigues las pautas indicadas en el taller dedicado a este tema y en el fastbook correspondiente, vas a poder crear tu dataset y modelo, en PyTorch, y entrenarlo con éxito.

Objetivos de este ejercicio

La idea principal de este ejercicio es que puedas aplicar los pasos del proceso de aprendizaje explicados en el fastbook y el videotutorial a un conjunto de datos. Queremos que realices una predicción, en este caso, sobre la calidad de un vino tinto en función de sus características.

Después de realizar este ejercicio podrás aplicar tus conocimientos para entrenar y evaluar un perceptrón multicapa con otros conjuntos de datos en el futuro.

Descripción de la actividad

Toma nota de las pautas que te compartimos para realizar el ejercicio.

- Tienes que trabajar con el conjunto de datos 'winequality-white.csv', que te hemos compartido en el campus. Este conjunto de datos contiene características de los vinos que vamos a usar para predecir la calidad de ellos. Puedes encontrar más información sobre el conjunto de datos aquí: https://archive.ics.uci.edu/dataset/186/wine+quality.
- Dicho conjunto contempla la variable 'quality', que es la variable que vamos a predecir (*quality* es calidad en inglés). Es decir, vamos a intentar predecir el valor real de la calidad. Y recuerda: el tipo de problema es de regresión.
- Desarrollamos la actividad en Python y PyTorch.

Te compartimos aquí y en el campus algunos ejemplos de código que puedes usar para resolver el problema. En estos ejemplos verás comentarios donde hay que añadir código. Puedes usar el código de los ejemplos o crear tus propias funciones.

- Para realizar la actividad sigue los siguientes pasos:

1. Lee los datos y lleva a cabo un análisis básico para ver los tipos de datos y cuántos datos hay.

2. Separa el conjunto de datos en entrenamiento (65%), validación (20%) y prueba (15%). Define **'quality'** como la variable para predecir.

3. Estandariza las variables que vamos a usar para predecir (no debes estandarizar la variable a predecir 'quality').

4. Crea los datasets en PyTorch (tienes un ejemplo a continuación) para los conjuntos de entrenamiento, validación y prueba.

```
class WineDataset(Dataset):
    def __init__(self, x, y):
        self.x = torch.tensor(x, dtype=torch.float32)
        self.y = torch.tensor(np.array(y).astype(np.float32), dtype=torch.float32)

    def __len__(self):
        # Devolver el número de ejemplos en el conjunto de datos

    def __getitem__(self, idx):
        # Devolver un ejemplo del dataset con la índice 'idx'
```

```
train_dataset = WineDataset(x_train, y_train)
val_dataset = # Crear el dataset para los datos de validación
test_dataset = # Crear el dataset para los datos de prueba
```

5. Crea los DataLoaders en PyTorch (observa el siguiente ejemplo) para los conjuntos de entrenamiento, validación y prueba. Hay que especificar el hiperparámetro batch_size (tamaño del batch).

```
batch_size = ## Especificar este hiperparámetro

train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = # Crear el DataLoader para los datos de validación
test_dataloader = # Crear el DataLoader para los datos de prueba
```

6. Define un modelo para hacer la predicción.

```
class WineQualityModel(nn.Module):
    def __init__(self, input_shape): #input_shape es el número de variables que vamos a usar para la predicción
        super(WineQualityModel, self).__init__()
        # Añadir las capas del modelo

    def forward(self, x):
        # Añadir los pasos a seguir en el forward pass
        return x
```

```
model = WineQualityModel(input_shape=x.shape[1])
```

7. Especifica los hiperparámetros: número de épocas (epochs) y tasa de aprendizaje (learning_rate) para el optimizador

8. Selecciona una función de coste (¡no olvides que este es un problema de regresión!).

9. Selecciona un optimizador y especifica que la tasa de aprendizaje sea lo que has especificado como la tasa de aprendizaje.

```
learning_rate = # Especificar tasa de aprendizaje
epochs = # Especificar número de epocas

optimizer = # Especificar el optimizador con la tasa de aprendizaje con el valor de learning_rate
loss_fn = # Especificar la función de costo
```

10. Crea una función para el entrenamiento. A continuación, te compartimos un ejemplo.

```python
def train(model, train_dataloader, optimizer, loss_fn):
    model.train()
    epoch_loss = 0
    for i_batch, (x_train, y_train) in enumerate(train_dataloader):

        # Pon a cero los gradientes para cada batch

        # Calcula la salida (predicciones) del modelo para los ejemplos del batch

        batch_loss = # Calcula el costo (función de costo con los valores predichos y reales)

        # Calcula los gradientes

        # Ajusta los pesos

        epoch_loss += batch_loss.item()

    loss_train = epoch_loss / i_batch

    return loss_train
```

11. Crea una función para la validación.

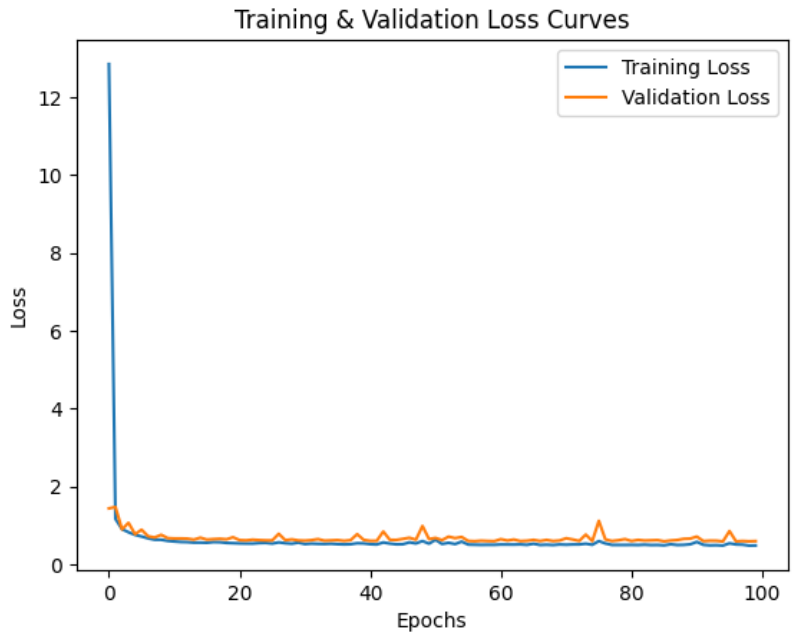
```python
def evaluation(model, val_dataloader, loss_fn):
    model.eval()
    epoch_loss = 0
    with torch.no_grad():

        for i_batch, (x_val, y_val) in enumerate(val_dataloader):

            # Calcula la salida (predicciones) del modelo para los ejemplos del batch

            batch_loss = # Calcula el costo (función de costo con los valores predichos y reales)

            epoch_loss += batch_loss.item()

    loss_val = epoch_loss / i_batch

    return loss_val
```

12. Crea el bucle para el entrenamiento y validación del modelo

```python
def training_evaluation_loop(epochs, model, train_dataloader, val_dataloader, optimizer, loss_fn):
    start = time.time()

    loss_values_train = []
    loss_values_val = []

    for epoch in range(epochs):

        loss_train = # Llama la función para el entrenamiento
        loss_values_train.append(loss_train)

        loss_val = # Llama la función para la evaluación
        loss_values_val.append(loss_val)

        # Imprime cada 10 épocas loss_train y loss_val

    end = time.time()
    total_time = end - start

    print(f'Total training time: {total_time}')

    return loss_values_train, loss_values_val
```

13. Entrena el modelo con el número de épocas que has especificada.

14. Dibuja una curva de aprendizaje como el siguiente ejemplo.



15. Crea una función para las predicciones usando el conjunto de prueba y calcula el error.

```python
def predictions(model, test_dataloader):
    predictions = []
    real_values = []

    model.eval()
    with torch.no_grad():
        for i_batch, (x_test, y_test) in enumerate(test_dataloader):

            outputs = # Calcular la salida (predicciones) del modelo para los ejemplos del batch

            predictions.append(outputs.detach().cpu().numpy())
            real_values.append(y_test.detach().cpu().numpy())

    predictions = np.vstack(predictions)
    real_values = np.hstack(real_values)

    # Calcula la métrica para la regresión para ver el error entre los valores predichos y reales

    # Imprima la métrica

    return predictions, real_values
```

16. Ejecuta la función y responde: ¿cuál es el error?, ¿las predicciones salen más o menos bien?

Crea tu propio repositorio de ejercicios

Realiza el ejercicio en un **Jupyter notebook,** indicando tu nombre y apellido, por si necesitas compartírselo al profesor, y elige el formato *.ipynb* (el propio de los Jupyter notebooks).

Criterios de corrección

En este ejercicio es más importante el proceso que has seguido que el error en las predicciones. Comprueba que los valores que has seleccionado para los hiperparámetros y para la arquitectura del modelo tienen sentido.

Qualentum