Trabajo Práctico 1:
Gradiente de Política

Lenguajes permitidos: Python 3.
Librerías permitidas: PyTorch, Gymnasium ≥ 1.0.0, NumPy, tqdm
Entrega: A través del campus virtual — Informe (máx. 2 páginas en PDF) y código fuente (comprimido en un archivo .zip)

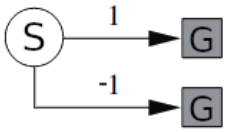
Objetivo

El objetivo de este trabajo práctico es implementar el algoritmo REINFORCE (capítulo 13.3 del libro de Sutton y Barto) y utilizarlo para entrenar agentes en distintos entornos. El trabajo se divide en cuatro partes:

Parte 1 — Entornos de Prueba

Para facilitar el desarrollo y *debugging* del algoritmo, se deben construir los siguientes entornos personalizados. Cada entorno debe implementarse como una subclase de *gymnasium.Env*. Se utilizará un factor de descuento γ=0.99 en todos los casos.

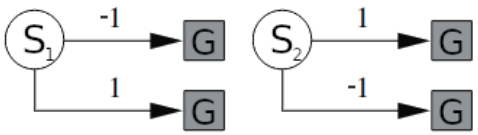
TwoAZeroObsOneStep



Observación: constante 0
Duración: un solo paso
Acciones: dos posibles
Recompensa: +1 o -1, depende únicamente de la acción tomada, hay una acción “buena” y una “mala”

TwoARandomObsOneStep

Observación: puede empezar en S_1 o en S_2 aleatoriamente.

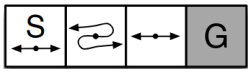


Duración: un solo paso
Acciones: dos posibles
Recompensa: +1 o -1, depende de la acción y de la observación.

LineWorldEasyEnv

Pasillo lineal de 6 casilleros
La observación es la posición actual [1–6]
Recompensa de +1 al alcanzar el objetivo en el extremo derecho; 0 en todos los otros casos
Posición inicial en el extremo izquierdo
El episodio termina al alcanzar el objetivo

LineWorldMirrorEnv



Pasillo de 4 estados: tres intermedios más el objetivo
La observación es la posición actual [1–4]
Las acciones posibles son “izquierda” y “derecha”. En el segundo estado las acciones están invertidas (derecha lleva a la izquierda y viceversa).
Recompensa de -1 por paso
El episodio termina al llegar al objetivo

Parte 2 — Implementación del Algoritmo REINFORCE

Se debe implementar el algoritmo REINFORCE, ver capítulo 13.3 del libro de Sutton y Barto.

Algorithm 1 Vanilla policy gradient algorithm

1: **Input:** Initial policy parameters θ_0.
2: **Parameters:** step size $\alpha > 0$, batch size N
3: Initialize policy parameters $\theta \in R^d$ (eg. to 0)
4: **for** $k = 1, 2, \ldots$ **do**
5: Collect a batch of trajectories by executing the current policy $\pi(a|s, \theta)$
6: **for** each timestep t in each trajectory do **do**
7: Compute the rewards-to-go:

$$R(t) = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$$

8: Estimate the policy gradient $\hat{g}$ as a sum of terms:

$$\hat{g}_k = \frac{1}{N} \sum_{i \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\Big|_{\theta_k} R(t)$$

9: Update the policy using the gradient estimate $\hat{g}_k$:

$$\theta_{k+1} = \theta_k + \alpha \hat{g}_k$$

10: or via another gradient ascent algorithm like Adam.

In step 8, the indices and terms are as follows:

- N: The batch size.
- i: The index for each trajectory within the batch, where $i \in \mathcal{D}_k$. $\mathcal{D}_k$ is the set of trajectories in the current batch.
- t: The timestep index within a specific trajectory, ranging from 0 to T.

Especificaciones:

- La política puede estar representada por una red neuronal con la siguiente arquitectura:

nn.Linear(input, 20),
nn.ELU(),
nn.Linear(20, 20),
nn.ELU(),
nn.Linear(20, output)

Se puede experimentar con distintas configuraciones de capas y diferentes funciones de activación como por ejemplo **nn.Tanh()**

- La política debe ser estocástica: las acciones deben muestrearse de la distribución (por ejemplo, usando Categorical), no seleccionarse con argmax.
- Como lo sugiere el pseudocódigo, **Algorithm 1**, en vez de descenso por gradiente estocástico pueden utilizar un optimizador más eficiente como Adam.
- Para facilitar la convergencia, se recomienda normalizar los retornos descontados dentro del batch antes del paso de retropropagación (media 0 y desvío estándar 1).

Parte 3 — Experimentos y Evaluación

Una vez implementado el algoritmo, se debe entrenar a un agente en los siguientes entornos:

1. TwoAZeroObsOneStep
2. TwoARandomObsOneStep
3. LineWorldEasyEnv
4. LineWorldMirrorEnv
5. CartPole-v1 (provisto por Gymnasium)
6. Acrobot-v1 (provisto por Gymnasium)

Para cada entorno se debe:

- Entrenar hasta que se considere el entorno ***resuelto*** (esto es cuando el reward converge y se está logrando el objetivo).
- Graficar la curva de recompensa promedio por episodio a medida que se entrena, esta debería ser aproximadamente creciente, se recomienda fuertemente, usar *tensorboard*, ver Apéndice.
- Incluir en los gráficos de recompensa líneas horizontales de referencia que representen el retorno medio de:
    - Un agente que actúa al azar.
    - Agentes triviales que toman siempre una de las acciones disponibles (por ejemplo: solo “derecha”, solo “izquierda”, etc.)

- Graficar la curva de pérdida por episodio de la red de política a medida que se entrena, tenga en cuenta en aprendizaje reforzado las curvas de *loss* no suelen tener la forma canónica que se espera usualmente en ML.

Parte 4 — Implementación del Algoritmo REINFORCE con Baseline.

En esta sección, se extenderá el algoritmo REINFORCE implementado previamente en la Parte 2, incorporando el uso de un **baseline** para reducir la varianza de la estimación del gradiente. El objetivo es mejorar la estabilidad del entrenamiento y acelerar la convergencia.

Motivación

En REINFORCE, la actualización de los parámetros de la política se basa en el gradiente estimado via:

$$\hat{g}_k = \frac{1}{N} \sum_{i \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \Big|_{\theta_k} R(t)$$

Este estimador, aunque no sesgado, tiene una alta varianza, lo que puede dificultar el aprendizaje eficiente. Una manera común de reducir dicha varianza sin introducir sesgo es restar un **baseline** $b(s_t)$ del retorno R_t El nuevo estimador del gradiente se define como:

$$\hat{g}_k = \frac{1}{|N|} \sum_{i \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \Big|_{\theta_k} \cdot A(t)$$

$$A(t) = R(t) - b(s_t)$$

Una elección natural y eficaz para el baseline es utilizar una **función de valor del estado** como estimador de baseline. Esta función puede ser aprendida mediante una red neuronal adicional, entrenada para predecir el retorno esperado a partir de cada estado.

La implementación debe seguir el siguiente pseudo-código:

Algorithm 2 Vanilla Policy Gradient Algorithm with baseline

1: **Input:** initial policy parameters θ_0, initial value function parameters ϕ_0
2: **Parameters:** step size $\alpha > 0$, batch size N
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
5: Compute rewards-to-go $R(t)$.
6: Compute advantage estimates $A(t)$ based on the current value function V_{ϕ_k}.
7: Estimate policy gradient as:

$$\hat{g}_k = \frac{1}{|N|} \sum_{i \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})\big|_{\theta_k} \cdot A(t)$$

8: Compute policy update, either using standard gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

9: or via another gradient ascent algorithm like Adam.
10: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - R(t)\right)^2,$$

11: typically via some gradient descent algorithm like Adam.

Especificaciones:

- Incorporar una **segunda red neuronal más simple** que tome como entrada el estado s_t y devuelva una estimación escalar $V(s_t)$.
- Durante el entrenamiento, calcular la ventaja estimada $A = R(t) - V(s_t)$.
- Ajustar los parámetros de la red de valor minimizando el error cuadrático medio entre la predicción $V(s_t)$ y el retorno real $R(t)$, tal como muestra la línea 9 del pseudocódigo.

Experimentos y Evaluación:

- El agente debe ser capaz de entrenarse exitosamente en los entornos CartPole-v1 y Acrobot-v1.
- Se debe comparar el desempeño del algoritmo con baseline frente a la versión sin baseline implementada en la Parte 2. Para ello, se recomienda utilizar TensorBoard, lo que permitirá superponer las curvas de recompensa y visualizar las diferencias de manera clara.

- En la comparación, debería observarse que el uso de una función de valor como baseline mejora la estabilidad del entrenamiento y acelera la convergencia del algoritmo.

Entrega

La entrega se realiza a través del campus virtual, e incluye:

Código fuente

Todo el código utilizado para definir entornos, entrenar agentes, pesos entrenados de las redes y generar las figuras. Debe ser original y estar comprimido en un archivo .zip.
No se evaluará el estilo ni la legibilidad del código, pero se utilizará para comprobar errores de implementación y verificar la autoría de dicho código.

Informe:

Un archivo PDF de máximo 2 páginas, que debe responder de manera sintética y clara las siguientes preguntas:

- ¿Qué parte del algoritmo REINFORCE ayuda a debuggear cada uno de los entornos de prueba? Ej: LineWorldEasyEnv ayuda a encontrar problemas con el factor de descuento.
- Se incluyen las curvas de recompensa media por episodio para cada uno de los seis entornos:
    a. TwoAZeroObsOneStep
    b. TwoARandomObsOneStep
    c. LineWorldEasyEnv
    d. LineWorldMirrorEnv
    e. CartPole-v1
    f. Acrobot-v1
- Para el entorno que más tardó en converger, explicar por qué se considera que fue más difícil para el algoritmo.
- Escribir una oración que indique qué parte del trabajo fue la más demandante en tiempo y qué se podría haber hecho diferente.

Criterio de aprobación

El trabajo se considera aprobado si se completa hasta la Parte 3.

5 - Apéndice

TensorBoard

TensorBoard es una herramienta de visualización que permite monitorear y analizar el entrenamiento de modelos de aprendizaje automático. Se utiliza para graficar métricas como la función de pérdida, returns, precisión, pesos, histogramas y más.

Se puede instalar con:

pip install tensorboard

Una caso de uso que podría resultarles útil es el siguiente:

```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter(log_dir='runs/experimento1')

for epoch in range(num_epochs):
    # Entrenamiento del modelo...
    writer.add_scalar('Loss/train', train_loss, epoch)
    writer.add_scalar('Loss/val', val_loss, epoch)
    writer.add_scalar('Accuracy/train', train_acc, epoch)
    writer.add_scalar('Accuracy/val', val_acc, epoch)
    writer.add_scalar("Performance/AverageReturn", avg_return, epoch)
    writer.add_scalar("Performance/AverageEpisodeLength", avg_length, epoch)

writer.close()
```

Para iniciar TensorBoard y visualizar los resultados, ejecutar:

tensorboard --logdir=runs

Luego, abrir un navegador y acceder a:

http://localhost:6006

Se debería desplegar algo similar al siguiente panel: