Autor: **Alejandro Casado**

Universidad Internacional de Valencia

Foro Evaluable

Tabla de Contenidos

1. Ideas & Conceptos

1.1 ¿Qué es un proceso estocástico, qué tipos hay y qué aplicaciones tienen?

Un **proceso estocástico** es una colección de variables aleatorias que representan la evolución de un sistema aleatorio a lo largo del tiempo.

Tipos principales

- **Según el tiempo:**

  Discreto: valores en momentos separados (ej. cadenas de Markov).

  Continuo: valores en cualquier instante (ej. movimiento browniano).

- **Según el espacio de estados**:

  Discreto: número de clientes en una cola.

  Continuo: posición de una partícula.

- **Ejemplos comunes**:

  Cadenas de Markov: el futuro depende solo del presente.

  Proceso de Poisson: cuenta eventos en el tiempo.

  Movimiento browniano: modelo de fluctuaciones continuas.

Simulación y videojuegos

- **IA de personajes no jugables** (NPCs): toma de decisiones probabilística.
- **Simulación de mundos**: generación de entornos realistas (ej. clima, eventos).

  Estas dos aplicaciones se podrían ejemplificar con el videojuego **Minecraft** para así entenderlas mejor. Por ejemplo, los NPC del tipo Zombis: Pueden seguir al jugador, pero su movimiento puede tener pequeños desvíos aleatorios, simulando un comportamiento natural.

1.2. ¿De qué tipo de proceso estocástico son las cadenas de Markov? Describa brevemente estos procesos.

Las **cadenas de Markov** son un tipo de **proceso estocástico en tiempo discreto y espacio de estados discreto**, con una propiedad especial:

Propiedad de Markov: El futuro depende solo del estado actual, no del pasado.

$$P(X_{n+1}|X_n)$$

Ilustración 1. Propiedad de Markov

Sus caracterísitcas principales son:

- **Tiempo discreto**: los pasos se dan en instantes separados
- **Espacio de estados discreto**: conjunto finito o numerable de estados
- **Matriz de transición**: define las probabilidades de pasar de un estado a otro.

1.3. ¿Qué aplicaciones tienen en IA los Procesos de Decisión de Markov (MDP)?

Los **Procesos de Decisión de Markov (MDP)** son una extensión de las cadenas de Markov que incluyen **acciones** y **recompensas**. Se utilizan ampliamente en **inteligencia artificial (IA)** para modelar agentes que deben tomar decisiones secuenciales bajo incertidumbre.

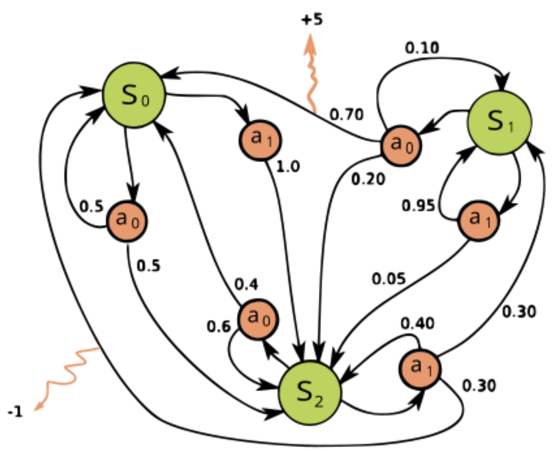

Ilustración 2. Ejemplo MDP

Algunas de sus **aplicaciones principales en IA** son:

- **Aprendizaje por refuerzo:** Por ejemplo, un robot que aprende a jugar al ajedrez y va mejorando básicamente por jugar más partidas
- **Aplicaciones en medicina y sistemas expertos:** Cómo un agente inteligente que te puede atender para tomar turno, medicación o consejos. Todo esto en base a la acumulación de otros 'chats' con otros clientes.

1.4. ¿Qué es una matriz estocástica (o matriz de transición) y qué aplicación tienen en los procesos de Markov?

Una **matriz estocástica** (también llamada **matriz de transición**) es una matriz cuadrada que describe las **probabilidades de pasar de un estado a otro** en una cadena de Markov. Una matriz $P=(p_{ij})$ es **estocástica** si:

- $p_{ij} \geq 0$
- $\sum_j p_{ij} = 1$

A su vez, cada entrada p_{ij} representa la **probabilidad de transición** del estado i al estado j en un solo paso.

Sus aplicación en procesos de Markov son: Determinar **probabilidades futuras**, analizar **comportamientos a largo plazo** (estado estable o distribución invariante) e identificar **clases de estados** (recurrentes, absorbentes, etc.).

1.5. ¿Qué es el vector de estado de un proceso de Markov, y qué representa?

El **vector de estado** de una cadena de Markov es un **vector fila de probabilidades** que indica la **distribución del sistema entre los distintos estados** en un momento dado.

Es decir, sea $\mathbf{S = \{s_1, s_2, s_3 ..., s_n\}}$ el conjunto de estados. El **vector de estado** en el instante **t**, denotado $\boldsymbol{\pi^{(t)}}$ es:

- $\boldsymbol{\pi^{(t)} = (\pi_1^{(t)}, \pi_2^{(t)}, \pi_3^{(t)} ..., \pi_n^{(t)})}$

Esto significa la probabilidad de estar en 1 en el estado de tiempo t, de estar en 2 en el estado de tiempo t y así hasta llegar a la probailidad de estar en el estado n-ésimo en el instante de tiempo t.

Este vector de estados evoluciona multiplicado por la **matriz de transición P:**

- $\boldsymbol{\pi^{(t+1)} = \pi^{(t)} * P}$

Por otro lado, a la pregunta formulada de **qué representa** se podría decir:

- Muestra la **distribución de probabilidades entre los estados** en un momento dado.
- Permite conocer **la probabilidad de estar en cualquier estado en el futuro**.
- Es clave para estudiar el **comportamiento a largo plazo** del sistema

1.6. ¿Qué es el estado estacionario de una cadena de Markov con un número finito de estados? ¿Cómo se calcula? y ¿qué significado tiene?

El **estado estacionario** (o **distribución invariante**) de una cadena de Markov con un número finito de estados es un **vector de probabilidades** π tal que, si la cadena comienza en esa distribución, entonces se mantiene inalterada con el paso del tiempo.

Formalmente, el vector π cumple:

$$\pi P = \pi$$

donde:

- $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ **es un vector fila de probabilidades:** $\pi i \geq 0$ **y** $\sum_{i=1}^{n} \pi_i = 1$
- **P es la matriz de transición de la cadena.**

Una vez conocido el concepto, para calcular el estado estacionario:

1. Se resuelve el sistema de ecuaciones lineales ($\pi P = \pi$)
2. Se añade la condición de normalización $\sum_{i=1}^{n} \pi_i = 1$

Este sistema suele tener una única solución si la cadena es irreducible, aperiódica o de un espacio de estados finito. Es en ese último estado cuando la cadena **converge** al estado estacionario independientemente del estado inicial.

Todo esto significa la **distribución de probabilidades a largo plazo**: indica la proporción de tiempo que se pasa en cada estado en promedio, cuando el número de pasos tiende a infinito. A su vez, es útil para **predecir el comportamiento estable** del sistema modelado por la cadena de Markov.

Para **ejemplificar** todo esto: Imagina una cadena de Markov que modela el clima (soleado, nublado, lluvioso). Si después de un largo período, la probabilidad de que haga sol, nublado o lluvia se mantiene estable, independientemente de si empezaste con un día soleado o lluvioso, entonces **has alcanzado el estado estacionario**.

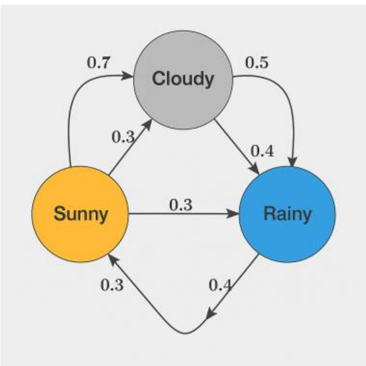

Ilustración 3. Ejemplo estado estacionario

2. Modelo del Clima – Generalidades

Considere un modelo del clima con tres estados posibles:

(1) Soleado, (2) Nublado y (3) Lluvioso.

2.1. ¿Pueden la filas de la siguiente matriz, ser las filas de una matriz de transición de un tal proceso de Markov?

$$Q = \begin{pmatrix} 0.2 & 0.5 & 0.4 \\ 0.1 & 0.5 & 0.3 \\ 0.7 & 0.5 & -0.2 \end{pmatrix}$$

Ilustración 4. Matriz ejercicio 2.1

Para que una matriz QQQ sea una **matriz de transición** de una cadena de Markov, debe cumplir dos condiciones fundamentales:

1. **Todos los elementos deben ser mayores o iguales que cero**
2. **La suma de cada fila debe ser igual a 1**

Procedemos entonces al chequeo por filas:

- Primera fila: 0.2 + 0.5 + 0.4 = 1.1 → **No suma 1**
- Segunda fila: 0.1 + 0.5 + 0.3 = 0.90 → **No suma 1**
- Tercera fila: contiene un **valor negativo**: −0.2 → Prohibido en matrices de transición

Dado que ambas condiciones deben cumplirse estrictamente —es decir, que todos los elementos sean no negativos y que cada fila sume exactamente uno— y en este caso **no se cumple ninguna de ellas**, se puede afirmar que **las filas de la matriz mostrada en la ilustración 4 no corresponden a una matriz de transición de un proceso de Markov**.

2.2. ¿Puede alguno de los vectores siguientes, ser el vector de estado actual de un tal proceso de Markov?

$u_0 = (0.3, 0.6, 0.7)$ // $w_0 = (0.2, 0.1, 0.5)$

Siguiendo la misma lógica que en el apartado anterior, al no cumplir la caracterísitca de **tener que ser la suma de los valores igual a 1, no es el vector estado actual de un proceso de Markov**

3. Modelo del Clima – Ejercicio

Considere un modelo del clima con tres estados posibles: (1) Soleado, (2) Nublado y (3) Lluvioso, con matriz de transición **P**:

$$\mathcal{P} = \begin{pmatrix} 0.2 & 0.5 & 0.3 \\ 0.1 & 0.6 & 0.3 \\ 0.5 & 0.3 & 0.2 \end{pmatrix}$$

Ilustración 5. Matriz ejercicio 3

Si el vector de estado actual es $v_0 = (P(S_0), P(N_0), P(L_0)) = (0.75, 0.20, 0.05)$ calcule:

1. **El vector de estado para mañana v = (P(S), P(N), P(L)).**

   Para calcular el vector de estado para mañana $\mathbf{v_1}$, usamos la regla fundamental de evolución de una cadena de Markov:

   $$\mathbf{v_1=v_0 \cdot P}$$

   $$v1 = (0.75, 0.20, 0.05) \cdot \begin{matrix} 0.2 & 0.5 & 0.3 \\ 0.1 & 0.6 & 0.3 \\ 0.5 & 0.3 & 0.2 \end{matrix}$$

   Componente 1 (Soleado):
   $P(S)=0.75 \cdot 0.2+0.20 \cdot 0.1+0.05 \cdot 0.5=0.15+0.02+0.025=0.195$

   Componente 2 (Nublado):
   $P(N)=0.75 \cdot 0.5+0.20 \cdot 0.6+0.05 \cdot 0.3=0.375+0.12+0.015=0.51$

   Componente 3 (Lluvioso):
   $P(L)=0.75 \cdot 0.3+0.20 \cdot 0.3+0.05 \cdot 0.2=0.225+0.06+0.01=0.295$

   $\mathbf{v_1=(P(S), P(N), P(L))=(0.195, 0.51, 0.295)}$

2. **Los vectores de estado para pasado mañana y de hoy a tres días.**

   PASADO MAÑANA
   P(S): $0.195 \cdot 0.2+0.51 \cdot 0.1+0.295 \cdot 0.5=0.039+0.051+0.1475=0.2375$
   P(N): $0.195 \cdot 0.5+0.51 \cdot 0.6+0.295 \cdot 0.3=0.0975+0.306+0.0885=0.492$
   P(L): $0.195 \cdot 0.3+0.51 \cdot 0.3+0.295 \cdot 0.2=0.0585+0.153+0.059=0.2705$

   $\mathbf{v_2 = (0.2375, 0.492, 0.2705)}$

Universidad Internacional de Valencia

TRES DÍAS

Usamos el vector recién calculado **v_2=(0.2375, 0.492, 0.2705)**

P(S): 0.2375·0.2+0.492·0.1+0.2705·0.5=0.0475+0.0492+0.13525=0.23195
P(N): 0.2375·0.5+0.492·0.6+0.2705·0.3=0.11875+0.2952+0.08115=0.4951
P(L): 0.2375·0.3+0.492·0.3+0.2705·0.2=0.07125+0.1476+0.0541=0.27295

v_3=(0.23195, 0.4951, 0.27295)

3. **Calcule el estado estacionario de la matriz de transición P.**

$$\begin{matrix} 0.2 & 0.5 & 0.3 \\ 0.1 & 0.6 & 0.3 \\ 0.5 & 0.3 & 0.2 \end{matrix}$$ → Vamos a calcular el **estado estacionario** π=(π1, π2, π3)

Buscamos tal que **πP=π** y **π1+π2+π3 = 1**

a. Esto da lugar al siguiente sistema de ecuaciones:

$0.2\pi_1+0.1\pi_2+0.5\pi_3 = \pi_1$
$0.5\pi_1+0.6\pi_2+0.3\pi_3 = \pi_2$
$0.3\pi_1+0.3\pi_2+0.2\pi_3 = \pi_3$
$\pi_1+\pi_2+\pi_3 = 1$

b. Restamos π_i de ambos lados

$-0.8\pi_1+0.1\pi_2+0.5\pi_3 = 0$
$0.5\pi_1-0.4\pi_2+0.3\pi_3 = 0$
$0.3\pi_1+0.3\pi_2-0.8\pi_3 = 0$
$\pi_1+\pi_2+\pi_3 = 1$

c. Utilizamos para llegar a la solución las primeras dos filas multiplicadas por 10 (para simplificar)

$-8\pi_1+1\pi_2+5\pi_3 = 0$
$5\pi_1-4\pi_2+3\pi_3 = 0$

i) $\pi_2 = 8\pi_1-5\pi_3$

$-8\pi_1+1\pi_2+5\pi_3 = 0$
$5\pi_1-4(8\pi_1-5\pi_3)+3\pi_3 = 0$

ii) $\pi_1 = 23/27\ \pi_3$

iii) $\pi_2 = 49/27\ \pi_3$

d. Finalmente sustituimos en la ecuación de **normalización**

$\pi_1+\pi_2+\pi_3 = 1$
$23/27\ \pi_3 + 49/27\ \pi_3 + \pi_3 = 1$
$\boldsymbol{\pi_3 = 3/11}$

e. Calculamos el valor de π_1 y π_2

π_1=23/27·3/11= **23/99**
π_2=49/27·3/11= **49/99**
π_3=3/11= **27/99**

Esto acaba resultando en el vector decimal aproximado: **π≈(0.232, 0.495, 0.273)**

4. **¿Cuál debería ser el vector de estado actual para que mañana el vector de estado sea v = (0.75, 0.20, 0.05)?**

Esta pregunta consiste en **invertir el proceso** de evolución de una cadena de Markov. Queremos encontrar el **vector de estado actual** v_0, que al multiplicarlo por la matriz de transición **P** nos da el vector del día siguiente:

v1 = (0.75, 0.20, 0.05)

$$\begin{matrix} 0.2 & 0.5 & 0.3 \\ 0.1 & 0.6 & 0.3 \\ 0.5 & 0.3 & 0.2 \end{matrix}$$

a) Plantear un sistema de ecuaciones

0.2x+0.1y+0.5z=0.75
0.5x+0.6y+0.3z=0.20
x+y+z=1

b) Resolvemos el sistema planteado

z=1+-x-y

0.2x+0.1y+0.5(1−x−y) = 0.75

(-0.3x - 0.4y + 0.5) = 0.75

-0.3x - 0.4y = 0.25

0.5x+0.6y+0.3(1−x−y)=0.20

$0.5x+0.6y+0.3-0.3x-0.3y=0.20$

$(0.2x+0.3y+0.3)=0.20$

$0.2x+0.3y=-0.10$

c) Seguimos con la resolución igualando estas dos ecuaciones

$-0.6x-0.8y+0.2x+0.3y=0.5-0.1$
$(-0.4x-0.5y)=0.4$
$0.4x+0.5y=-0.4$

d) Despejamos x

$x=(-0.4-0.5y)/0.4 \Rightarrow x=-1-1.25y$

e) Sustituimos para conseguir el valor de z

$z=1-x-y=1-(-1-1.25y)-y =$
$1+1+1.25y-y=2+0.25y$

Llegados a este punto observamos que el vector tendría que ser [-3.5, 2, 2.5], aunque se tiene que cumplir la propiedad que todos los miembros del mismo sean mayores que cero.

Es por ello, que se considera que este sistema no tiene una solución realista y no existe un estado valido que produzca (en un paso) **v = [0.75 0.20 0.05]**

5. **Si la entrada ij de la matriz P la denotamos por pij , donde los índices i, j corresponden a los estados i y j respecϴvamente (1 para soleado, 2 para nublado y 3 para lluvioso)**

   a. **¿Qué representa la probabilidad p_{ij} en términos de los estados i y j?**

      La probabilidad p_{ij} prepresenta la probabilidad condicional de que el sistema **pase del estado i al estado j en un solo paso de tiempo** en una cadena de Markov.

   b. **Si S_0, N_0 y L_0 corresponden a los estados actuales, y S, N y L a los estados de mañana, identifique con las entradas de la matriz P las siguientes probabilidades:**

      i) $P(S|S_0)$ = $p_{11} = 0.2$

      ii) $P(N|L_0)$ = $p_{32} = 0.3$

      iii) $P(L|N_0)$ = $p_{23} = 0.3$

4. Tensores - Resumen

4.1 Introducción

A Simple and Efficient Tensor Calculus for Machine Learning" presenta un nuevo enfoque para calcular derivadas de expresiones tensoriales en el contexto del aprendizaje automático. Aquí tienes un resumen con las ideas principales.

4.2 Motivación

Se considera que puede haber varias fuentes de motivación detrás del artículo sobre todo realizando un análisis del estado del arte al respecto y los posibles trabajos previos sobre lo mismo. Algunas son:

- **Calcular derivadas** (gradientes, Jacobianos, Hessianos) de expresiones tensoriales es crucial en machine learning.
- **Los frameworks actuales** (TensorFlow, PyTorch, JAX, autograd) son eficientes para funciones escalares, pero **ineficientes (hasta 1000× más lentos)** con funciones no escalares.
- Un **método anterior** basado en notación de Ricci (Laue et al., 2018) es mucho más eficiente, pero **no es compatible** con estos frameworks.

4.3 Desarrollo

Una de las contribuciones técnicas centrales es la introducción de un operador de multiplicación tensorial genérico denotado como $*(s_1, s_2, s_3)$, donde cada conjunto de índices define cómo se combinan los tensores de entrada y cómo se estructura el tensor de salida. Este operador unifica distintos tipos de productos (interno, externo y elemento a elemento), proporcionando una base coherente para realizar cálculos diferenciales sobre expresiones complejas.

Además, el artículo presenta un marco formal para calcular derivadas en tres modos distintos:

Modo forward (hacia adelante): Propaga derivadas desde las variables de entrada hasta la salida, útil cuando se desean derivadas respecto a pocas variables de entrada.

Modo reverse (hacia atrás): Propaga derivadas desde la salida hacia las entradas, eficiente cuando se calcula el gradiente completo de una función escalar.

Modo cross-country (modo mixto): Combina forward y reverse de forma estratégica para reducir los costos computacionales, especialmente útil para funciones no escalares o derivadas de orden superior.

El **modo cross-country** -el último análizado previamente- no solo permite una mayor flexibilidad en la evaluación de derivadas, sino que también habilita estrategias de optimización. Una de ellas consiste en **reordenar las multiplicaciones tensoriales** según el orden de los tensores (empezando por vectores, luego matrices, etc.), lo que puede reducir significativamente el tiempo de cómputo. Otra mejora clave es la **compresión de derivadas de orden superior**, como los Hessianos, que normalmente son representados como tensores de cuarto orden.

4.4 Conclusiones y resultados

Para validar la eficiencia de su método, los autores lo comparan empíricamente en tres tareas típicas del aprendizaje automático, como son la Regresión logística, la factorización matricial y las redes neuronales con capas totalmente conectadas

Tras la realización de todos estos experimentos en distintos frameworks y sobre distintos campos, los resultados muestran que el enfoque propuesto:

- **Calcula Hessianos mucho más rápido** que los frameworks existentes, superando incluso al método anterior de los autores basado en notación de Ricci.
- **Hace viable el cálculo del Hessiano** en redes neuronales pequeñas, tarea que era computacionalmente prohibitiva con otros métodos.

Estas mejoras se observan tanto en CPU como en GPU, y confirman la robustez y aplicabilidad del algoritmo propuesto. Es por ello, que se concluye con que proveen una **solución práctica y eficiente para derivadas tensoriales, además de ser compatible con los framework más actuales.** Por lo tanto, cumple a la perfección con la intención inicial que hablaba el artículo.