2 Razonamiento computacional bajo incertidumbre en Redes Neuronales

2.1 Introducción a la incertidumbre

Definiciones

Incertidumbre: falta de seguridad, de confianza o de certeza sobre algo, especialmente cuando crea inquietud.
Fuente: Oxford Languages

En el mundo real, la incertidumbre es un concepto básico en el ámbito de la predicción. Por ejemplo, ante un problema cotidiano como *"Esta nublado, ¿debería llevarme un paragüas?"* o si *" ¿si juego con mi jugador estrella en este partido lo ganaré?"* no se tiene certeza del resultado o solución al mismo.
Para dar respuesta a diferentes problemas computacionales, en el razonamiento computacional se reflejaría una falta de conocimiento veraz y preciso sobre los datos, o una gran variabilidad de los mismos. Además, se puede producir una falta de completitud de la información, presencia de ruido o errores en las mediciones. También afectaría una excesiva complejidad en los sistemas que se estén modelando.
Esta falta de certeza absoluta o de conocimiento preciso (certidumbre) produce una falta de capacidad para predecir con eficacia y precisión los datos o resultados futuros.

Es por esto que toma gran relevancia la valoración de la incertidumbre en el razonamiento computacional, teniendo en cuenta los siguientes aspectos:

- Toma de decisión: frente a decir por seleccionar una solución u otra, hay que hacer una evaluación de la incertidumbre, teniendo en cuenta los riesgos y las consecuencias que puedan derivarse de la acción escogida. En esta decisión entra en juego la ambigüedad e imprecisión que se puede dar en la interpretación de los datos,

donde algunos casos implican diversas interpretaciones posibles que son difícilmente cuantificables (incertidumbre epistémica).
- Evaluación de la confiabilidad del resultado: tras la ejecución de un algoritmo o modelo, la valoración del resultado (si es correcto o no) se realizar a través de la evaluación de la incertidumbre. De esta forma, es posible cuantificar cómo de confiable es cada resultado.
- La variabilidad y el ruido: los datos reales están siempre ligados a la variabilidad y el ruido, siendo complejo el modelado de problemas del mundo real. Los datos a menudo están ligados a errores en la medición, fluctuaciones naturales o influencias externas imprevisibles. Es por ello que se complica la distinción de los efectos reales y los aleatorios.
- Razonamiento lógico: en la mayoría de situaciones, el resultado no puede reducirse binariamente (verdadero o falso), sino que hay grados de veracidad o pertenencia. En este contexto, la incertidumbre se relaciona con el razonamiento lógico y la lógica difusa, para dar respuesta a situaciones sin certeza absoluta.
- Adaptabilidad y robustez: la robustez frente a errores de las aplicaciones o sistemas es un punto muy relevante, que se ve afectado cuando los datos o condiciones cambian. Por lo tanto, la adaptabilidad y la robustez son factores que entran en conflicto. Aquellos sistemas que tratan la incertidumbre se comportan mejor que los que no lo hacen.

Teniendo en cuenta todo esto, se tiene que evaluar la incertidumbre en cada problema, dotando al sistema con la capacidad para manejar la incertidumbre.
En el ejemplo del paragüas, frente a la toma de decisión de si llevarlo o no, se debe tener en cuenta factores como al probabilidad de precipitaciones, los datos meteorológicos, si está muy nublado o no, y la experiencia pasada en estos casos. Sin embargo, hay que tener en cuenta el contexto: la decisión cambiará si el objetivo es ir a un evento cerrado o al aire libre. En el segundo caso, aún con una probabilidad baja de precipitaciones, quizás sería lógico llevarlo.
Respecto a las decisiones, si se decide llevarlo y no se usa, habrá sido molesto cargar con él todo el día. Y si no se lleva y llueve, si se está al aire libre te mojarás. Este contexto añade una incertidumbre adicional que debe ser valorada, por ejemplo, mediante decisiones basadas en el riesgo y la tolerancia al mismo.
Y en el ejemplo del jugador estrella, habrá que analizar el contexto, como si el jugador viene o no de una lesión, de si el partido es decisivo en el campeonato, etc. La toma de decisión afectará al resultado final del partido.

2.2 Fundamentos de las redes neuronales y el aprendizaje con incertidumbre

En este apartado, se realiza un repaso de los conceptos básicos de las redes neuronales, como los nodos, capas, funciones de activación y entrenamiento mediante retropopagación, entre otros.

En la IA, para la tarea de modelar o simular el comportamiento del cerebro humano se hace uso de las redes neuronales artificiales (RNA). Están formadas por un conjunto de nodos o neuronas que se interconectan entre sí. El rol de cada neurona es el cálculo de valores de entrada para producir una salida. Además, estas redes permiten el aprendizaje a partir de datos.
Los conceptos básicos de estas redes se describen brevemente a continuación:

Neuronas (nodos).

Un nodo o neurona en una red neuronal es la unidad básica de procesamiento. Cada nodo tiene una o más entradas y una función de activación que se aplica a la suma ponderada de las entradas. Además, cada entrada está asociada con un peso que determina la importancia de esa entrada en el cálculo del nodo.

El cálculo que puede realizar una neurona viene expresada de forma matemática como:

Remark 2.2.1 Cálculo de una neurona.

$$z = w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n + b,$$
$$a = f(z),$$

donde w son los pesos asociados a las entradas x, b es el sesgo (bias) y f es la función de activación.

Capas.

La organización de las redes neuronales se realiza en capas, existiendo tres tipos principales de capas: de entrada, ocultas y de salida.

- Capa de entrada (*input layer*): es la puerta de entrada a la red neuronal. Las neuronaas que pertenecen a esta capa representan una característica o variable de entrada a la red. Son meros *"transportadores"*, puesto que se limitan a transmitir la información o datos a la capa siguiente, sin realizar ningún cálculo.
- Capa oculta (*hidden layers*): son aquellas capas que se situan entre las capas de entrada y salida. El rol que tienen es el de realizar la transformación no lineal de los datos de entrada hacia la capa de salida. Están compuestas por un número de neuronas que procesan los datos recibidos, generando nuevas representaciones internas ocultas al usuario, que no interactua con ellas. En estas capas es donde se produce el procesamiento mediante las conexiones (pesos y sesgos).
- Capa de salida (*output layer*): se trata de la capa final de la red, y su objetivo principal es dar el resultado final. El número de neuronas de esta capa varía en función del tipo de problema que se esté tratando. Como ejemplo, en un problema de clasificación binaria bastaría con una neurona de salida, que representaría la probabilidad de pertenecer a una clase. Por el contrario, en un

problema de clasificación multiclase sería necesario una neurona por cada clase existente.

Funciones de Activación.

En un sistema de neuronas conectadas entre sí, al igual que en un cerebro humano, se debe decidir si una neurona se activa o no. Computacionalmente, en base al valor de entrada en la red se calcula una suma ponderada con el sesgo, decidiendo el resultado una función llamada función de activación. El resultado se transmite a otras neuronas conectadas ("*passforward*") hasta llegar a la capa de salida. En resumen, el papel más importante que juegan estas funciones es la capacidad para incluir no linealidad en una red neuronal (hay funciones de activación lineales, pero están en desuso).
Existen numerosas funciones de activación, explicando brevemente a continuación las más importantes:

- Función Sigmoide (sigmoid): es muy simple, puesto que transforma los datos en un resultado entre 0 y 1 (muy interesante en el contexto de probabilidades). Se usa especialmente en clasificación binaria.

  Remark 2.2.2 Función sigmoide.
  $$f(x) = \frac{1}{1+e^{-x}},$$

- Función Tangente Hiperbólica o Gaussiana (Tanh): es similar a la función sigmoide, pero incluye rangos negativos (entre -1 y +1). Es especialmente útil en los casos en los que se necesita que la salida tenga valores cercanos a cero.

  Remark 2.2.3 Función Tangente Hiperbólica.
  $$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}},$$

- Función ReLU (Rectified Lineal Unit): es la más usada en la actualidad (en las capas ocultas, no en la de salida). Se trata de una función no lineal donde los valores de salida negativos, la neurona devuelve cero, y devuelve el valor de entrada en el resto de casos.

  Remark 2.2.4 Función ReLU.
  $$f(x) = max(0, x),$$

- Función Softmax: se usa en la capa de salida para problemas de clasificación multiclase. Transforma todas las salidas en probabilidades que suman 1.

  Remark 2.2.5 Función Softmax.
  $$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Hay otras muchas funciones de activación, cada una adaptada a un problema concreto, como las que se enumeran a continuación:

1. Leaky ReLU.
2. Parametric ReLU (PReLU).
3. Exponential Linear Unit (ELU).
4. Softplus.

5. Softsign.
6. Linear.
7. Gaussian.
8. ArcTan (Arcotangente).
9. Sinusoid.
10. Swish.
11. Bent Identity.
12. Maxout.
13. Hardshrink.
14. Hardtanh.
15. LogSigmoid.
16. LogSoftmax.

Entrenamiento de una red neuronal (Retropropagación).

Para obtener un resultado que minimice el error (o función de pérdida), las redes neuronales se entrenan mediante ciertos algoritmos que ajustan los pesos en las conexiones entre neuronas (o nodos). El más usado, el algoritmo de retropropagación o *backpropagation*, se puede resumir en una secuencia de pasos:

1. Inicialización: se asignan pesos aleatorios a las conexiones entre nodos.
2. Alimentación hacia delante (forward propagation): los datos de entrenamiento se transmiten desde la capa de entrada hacia la de salida, pasando por cada neurona, que realiza su tarea de cálculo, y obteniéndose la predicción del resultado.
3. Cálculo del error: se realiza una comparativo del resultado predicho con el real conocido de antemano, calculando mediante una función de pérdida el error cometido (por ejemplo, el error cuadrático MSE).
4. Retropropagación del error: una vez calculado el error, se propaga hacia atrás en la red neuronal, desde la capa de salida hacia la de entrada, donde cada nodo calcula su papel en el error mediante la derivada parcial de la función de pérdida respecto a la salida obtenida.
5. Actualización de pesos: los pesos de las conexiones entre neuronas son actualizados partiendo del gradiente descendente, el cual se encarga de definir la dirección y magnitud necesaria del nuevo peso para reducir y minimizar el error cometido. Además, para manejar la velocidad de convergencia del algoritmo se usa un valor denominado factor de aprendizaje.
6. Repetición del proceso: se vuelve al paso 1, repitiendo el proceso hasta obtener un criterio de parada, que suele ser un umbral de rendimiento concreto o un número de iteraciones máximo.

2.2.1 Ejemplo Práctico

Enunciado: se desea entrenar una red neuronal para clasificar imágenes de naranjas y peras. Para ello, se va a construir una red neuronal sencilla con una capa de entrada, una capa oculta y otra capa de salida con un nodo por cada clase (naranja y pera). Para cada imagen de entrada, se extraerían características importantes para usarlas como entradas para

la red neuronal. En el código de ejemplo estos valores de los pesos serán aleatorios, por sencillez en la explicación. El lenguaje usado será Python.

Paso 1: Inicialización

Se importan las bibliotecas necesarias, se define la función de activación y se inicializan los parámetros de la red neuronal.

```
#Importar libreria numpy
import numpy as np

#Funcion de activacion Sigmoid
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

#Inicializacion de los pesos (w1 y w2) y sesgos (b1 y b2) de la capa oculta y de salida
#Utilizaremos valores aleatorios bajos para los pesos y sesgos iniciales
np.random.seed(42)
input_size = 2
hidden_size = 3
output_size = 1

W1 = np.random.randn(input_size, hidden_size)
b1 = np.random.randn(hidden_size)

W2 = np.random.randn(hidden_size, output_size)
b2 = np.random.randn(output_size)
```

Paso 2: Alimentacion hacia adelante (Forward propagation)

A partir de los datos de entrenamiento de entrada, se pasan primero a la capa oculta, y el resultado hacia la capa de salida. Durante el proceso, se calcula el resultado predicho a partir de la función de activación propuesta en cada neurona.

```
def forward(X):
# Propagacion hacia adelante desde la entrada a la capa oculta
z1 = np.dot(X, W1) + b1
a1 = sigmoid(z1)

# Propagacion hacia adelante desde la capa oculta a la salida
z2 = np.dot(a1, W2) + b2
a2 = sigmoid(z2)

return a2, a1
```

Paso 3: Cálculo del error

Se hace una comparativa entre el resultado predicho obtenido y el valor real conocido de antemano, mediante una función de pérdida.

```
def loss(y_true, y_pred):
    # Funcion de perdida: error cuadratico medio
    return np.mean((y_true - y_pred) ** 2)
```

Paso 4: Retropropagacion del error (backpropagation)

Una vez obtenido el error, éste se propaga hacia atrás por la red neuronal, desde la capa de salida hasta la de entrada. Para ello, cada nodo calcula cuál ha sido su contribución al error mediante el gradiente de la función de pérdida con respecto a su salida (la derivada parcial).

```
    def backward(X, y_true, y_pred, a1):
    m = y_true.shape[0]  # Numero de ejemplos de entrenamiento

    # Gradiente de la funcion de perdida respecto a la salida (derivada
                                    parcial)
    dloss_da2 = 2 * (y_pred - y_true) / m

    # Gradientes de la capa de salida
    da2_dz2 = y_pred * (1 - y_pred)
    dz2_dW2 = a1
    dz2_db2 = 1

    # Gradientes de la capa oculta
    dz2_da1 = W2
    da1_dz1 = a1 * (1 - a1)
    dz1_dW1 = X
    dz1_db1 = 1

    # Gradientes finales
    dloss_dW2 = np.dot(dz2_dW2.T, dloss_da2 * da2_dz2)
    dloss_db2 = np.sum(dloss_da2 * da2_dz2, axis=0)

    dloss_dW1 = np.dot(X.T, np.dot(dloss_da2 * da2_dz2, dz2_da1.T) *
                                    da1_dz1)
    dloss_db1 = np.sum(np.dot(dloss_da2 * da2_dz2, dz2_da1.T) * da1_dz1,
                                    axis=0)

    return dloss_dW2, dloss_db2, dloss_dW1, dloss_db1
```

Paso 5: Entrenar la red neuronal

Se entrena la red neuronal, siguiendo los 4 pasos anteriores. Luego, se actualizan los pesos y segos de las conexiones utilizando el gradiente descendente. Estos pasos se repiten iterativamente hasta que la red neuronal alcance un nivel de rendimiento satisfactorio o se alcance un número máximo de iteraciones (epochs).

```
# Datos de entrenamiento de ejemplo
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_true = np.array([[0], [1], [1], [0]])
# Inicializacion de la tasa de aprendizaje y el numero maximo de las
                                    iteraciones
learning_rate = 0.1
epochs = 10000

for epoch in range(epochs):
    # Propagacion hacia adelante
    y_pred, a1 = forward(X)

    # Calculo de la funcion de perdida
    current_loss = loss(y_true, y_pred)

    # Retropropagacion
    gradients = backward(X, y_true, y_pred, a1)
```

```
17
18      # Actualizacion de los pesos y sesgos
19      W2 -= learning_rate * gradients[0]
20      b2 -= learning_rate * gradients[1]
21      W1 -= learning_rate * gradients[2]
22      b1 -= learning_rate * gradients[3]
23
24      # Mostrar la perdida actual cada 1000 epochs
25      if epoch % 1000 == 0:
26          print(f"Epoch {epoch}: Perdida = {current_loss:.4f}")
```

Una vez configurada la red neuronal con una capa oculta y una de salida, con función de activación soigmoide y función de pérdida de error cuadrático medio, establecemos como criterio de parada 1000 iteraciones. Durante el entrenamiento, habría que alimentar la red con una imagen de una naranja, que se iría propagando entre capas. Finalmente, se compara el resultado con la etiqueta real de la imagen para calcular el error cometido, propagando hacia detrás dicho error y reajustando los pesos. Una vez entrenada la red reuronal, estará preparada para clasificar naranjas respecto a otras frutas.
Se trata de un ejemplo muy simple para entender los conceptos básicos. En la práctica real, habría que ajustar con detalle los hiperparámetros y usar diferentes técnicas para mejorar el rendimiento de la red y evitar sobreajustes (*overfitting*).

2.3 Redes neuronales con inferencia probabilística

Cuando se habla de incertidumbre en los datos y los parámetros del modelo, es imprescindible conocer las herramientas que nos proporciona el modelado probabilístico.

A continuación, a modo de introducción breve, se exponen algunos de los conceptos clave del modelado probabilístico. No obstante, estos se explican con detalle en el Capítulo 3.

Distribuciones de probabilidad.

Una distribución de probabilidad describe la probabilidad de que ocurra un evento concreto, o un valor que se encuentra en un conjunto dado. Matemáticamente hablando, se trata de una función que asigna una probabilidad a cada resultado que se puede dar.

Se podrían clasificar en dos grandes grupos:

- Distribución de probabilidad discreta: trabaja y asigna probabilidades a valores discretos (por ejemplo, la distribución uniforme, de Bernoulli o la multinomial).
- Distribución de probabilidad continua: trabaja con una densidad de probabilidades a valores continuos (por ejemplo, la distribución normal, la exponencial o la gamma).

Teorema de Bayes.

Se trata de un teorema muy sencillo y útil para calcular la probabilidad de que ocurra un evento en base a la ocurrencia confirmada de otro. Por ejemplo, en un juego de cartas, sabiendo que has sacado un as de la baraja, se podría calcular la probabibilidad de que sea negra (picas o tréboles). En este caso sería 50%. Se dice que es la relación entre las probabilidades condicionales inversa y directa de dos eventos.
La formulación matemática del Teorema de Bayes se expresa con la siguiente fórmula:

Remark 2.3.1 Teorema de Bayes.

$$P(A \mid B) = \frac{P(B|A) * P(A)}{P(B)},$$

donde P(A | B) es la probabilidad condicional directa, la probabilidad de que ocurra el evento A si previamente ha ocurrido el evento B, P(B | A) es la probabilidad condicional inversa, y P(A) y P(B) son las probabilidades marginales de A y B, respectivamente.

En el ámbito del modelado probabilístico, se usa para actualizar la distribución de probabilidad de los parámetros del modelo conforme se observan nuevos datos.

Inferencia en redes neuronales bayesianas

Las redes neuronales son modelos que permiten la captura de patrones complejos en los datos, pero no suelen tener en cuenta la incertidumbre en los parámetros del dicho modelo. Este punto puede completarse combinando las redes neuronales con el razonamiento probabilístico.

La herramienta fundamental para el razonamiento probabilístico es la inferencia, que puede definirse de la siguiente manera:

Definición

Inferencia probabilística: proceso del análisis de datos y la estadística para la obtención de conclusiones sobre eventos desconocidos (o no observados) en base a información conocida previamente.

En la práctica, se trata de usar modelos probabilísticos y métodos estadísticos para calcular (o estimar) la distribución de probabilidad de variables de interés. Existen diferentes inferencias (que se detallan en el Capítulo 3), siendo la más usada la inferencia bayesiana, donde se asignan distribuciones de probabilidad a los parámetros del modelo en lugar de valores puntuales.
De esta forma, es posible la captura de la incertidumbre en los parámetros y la obtención de estimaciones probabilísticas en lugar de valores determinísticos.

Esta inferencia se aplica en redes neuronales a través de diversas técnicas, como el muestreo de Monte Carlo, el método de Montecarlo Hamiltoniano o el muestreo de Gibbs, para aproximar la distribución posterior de los

parámetros. Con ellos, se generan muestras de los parámetros del modelo que permiten el cálculo de las estimaciones probabilísticas y la obtención de los intervalos de confianza.

En resumen, la combinación de redes neuronales y el razonamiento probabilístico permite un enfoque más completo que, aparte de capturar patrones en los datos, modela y cuantifica la incertidumbre en los parámetros del modelo.

Aplicaciones de las redes neuronales bayesianas

No son más que una extensión de las redes neuronales tradicionales, incorporando el razonamiento probabilístico y la modelización de la incertidumbre en los parámetros del modelo. Esto las dota para el uso idóneo en problemas con pocos datos o con mucho ruido, permitiendo aplicarse en problemas como la predicción de incertidumbre, la toma de decisiones bajo incertidumbre y el aprendizaje activo.

A continuación se describen algunas aplicaciones de estas redes:

- Visión por computadora: se usan en tareas de reconocimiento de objetos, detección de anomalías y segmentación de imágenes. La capacidad de modelar la incertidumbre permite la evaluación de la confianza de las predicciones y la toma de decisiones de una forma más informadas. Por ejemplo, si se busca encontrar un objeto concreto en una imagen, se podría calcular la estimación de la incertidumbre sobre del mismo y usarla para establecer umbrales de confianza.
- Procesamiento de lenguaje natural: se usan en tareas como el reconocimiento de voz, la traducción automática y la generación de texto. Como ejemplo, se podrían obtener estimaciones más precisas de la probabilidad de encontrar palabra dada una entrada de texto, lo cual permite la mejora de la capacidad de los sistemas para capturar la ambigüedad y la variabilidad que caracteriza al lenguaje natural.
- Robótica: se usan para el procesamiento sensorial, la localización y la planificación de trayectorias. Por ejemplo, se podría calcular incertidumbre en los movimientos del robot, tomando decisiones más seguras y confiables, como podría ser si aplica mayor o menor fuerza en función del material del objeto que va a agarrar. Además, permite una mejora de la robustez debido a la adaptación frente a cambios que puedan darse en tiempo real.
- Medicina: se usan en diagnóstico médico, predicción de enfermedades y análisis de imágenes médicas. Por ejemplo, se podrían estimar con mayor precisión las probabilidades de que un paciente tenga una enfermedad determinada o que un tratamiento concreto sea efectivo.

2.4 Estimación de la incertidumbre

La capacidad de proporcionar estimaciones de incertidumbre en las predicciones es importante en diferentes redes neuronales donde se pretende añadir razonamiento bajo incertidumbre.

Esta estimación es el proceso de cuantificar y proporcionar una medida de la incertidumbre que esté asociada a las predicciones que el modelo genera. Una de las grandes diferentes entre redes neuronales y aquellas con razonamiento bajo incertidumbre es que las primeras dan predicciones determinísticas, las redes neuronales con estimación de incertidumbre incluyen información extra sobre la confiabilidad o falta de confiabilidad de sus predicciones.

Como ya se ha comentado anteriormente, la incertidumbre puede venir debido a diferentes factores, como la ausencia o falta necesarios de datos, la presencia de ruido en los datos de entrada de la red, la variabilidad intrínseca de los patrones subyacentes o, incluso, la complejidad del problema. Por lo tanto, la tarea de estimar y entender esta incertidumbre es vital en muchas aplicaciones para que el usuario pueda tomar decisiones más acertadas y comprender los límites del modelo.

Existen diferentes medidas de incertidumbre que se usan en estas redes para estimar la incertidumbre, siendo interesante destacar las siguientes:

Entropía.

Se trata de una medida de la incertidumbre o desorden en una distribución de probabilidad que permite cuantificar dicha incertidumbre en las predicciones. Se calcula de la siguiente forma:

Remark 2.4.1 Entropía.

$$H = -\sum(p * log(p)),$$

donde p representa las probabilidades de cada clase o valor de salida.

Siguiendo el ejemplo del modelo de red neuronal que clasifica imágenes de naranjas y peras, si se obtiene una distribución de probabilidad de [0.85, 0.15] para una imagen concreta, la estimación de la incertidumbre medianete el cálculo de la entropía sería:

H = - (0.85 * log(0.85) + 0.15 * log(0.15)) = 0.4227

¿Cómo se interpreta este dato? Una entropía alta indica una mayor incertidumbre o ambigüedad en las predicciones, y por lo contrario, una valor más bajo indica una mayor confianza en la predicción. En el caso del ejemplo, se podría decir que una cierta confianza en la predicción.

Varianza.

En estadística, la varianza es una medida de dispersión que muestra como de dispersos están los datos alrededor de la media de los mismos. Un valor alto significa que hay mucha variabilidad en los datos, y por tanto la media no es muy significativa. En el ámbito de las redes neuronales se usa para estimar la incertidumbre en las predicciones, es decir, cuantifica la dispersión de las predicciones alrededor de su valor medio. De esta forma, indica como de confiables o estables son las predicciones del modelo.

El resultado se puede leer de la siguiente forma:

- A mayor valor de la varianza, mayor será la incertidumbre. Si la varianza es alta, significa que las predicciones del modelo varían considerablemente para una misma entrada, indicando una mayor incertidumbre en las predicciones.
- Es un indicador de estabilidad, puesto que un valor bajo indica que las predicciones son consistentes y estables. Por el contrario, un valor alto sugiere que las predicciones pueden ser menos confiables y/o más susceptibles a cambios en los datos de entrada.
- Es un gran complemento a la entropía, puesto que mientras ésta mide la incertidumbre asociada con la distribución de probabilidad de las predicciones, la varianza complementa con información sobre la dispersión de las predicciones individuales.

Un ejemplo de uso de este estimador podría ser en una tarea de regresión, donde se desea predecir el precio de un *smartphone* en función de sus características (memoria, almacenamiento, cámara, etc.). Si se tiene un modelo de red neuronal que genera predicciones probabilísticas, se pueden realizar muchas predicciones para el mismo dispositivo usando diferentes técnicas, como las de muestreo de Monte Carlo. Después, se calcula la varianza de estas predicciones para obtener una medida de la incertidumbre en la estimación del precio.

Si se generan 10 predicciones de precio y se obtuvieron los siguientes valores: [405, 410, 395, 410, 409, 405, 405, 408, 405, 410]. Para calcular la varianza, se realiza la siguiente operación:

Remark 2.4.2 Varianza.

$$\sigma^2 = \frac{1}{N} * \sum_1^N (x - \mu)^2,$$

donde x representa las predicciones, μ es la media de los datos y N el número total de valores.

Tras calcular la media de las predicciones (406.2 euros), la varianza es aproximadamente 18.56 euros. Este resultado indica que las predicciones tienen una dispersión relativamente baja, cercana a la media, que permite considerar como confiables las predicciones en comparación con una varianza que hubiera sido más alta.

Intervalos de confianza.

Una medida para medir la confianza de una predicción consiste en establecer un par de valores que conforman un rango (llamados intervalos de confianza) dentro de los cuales un resultado es correcto o no. Existen diferentes técnicas para obtenerlos, como el muestreo de Monte Carlo.
El proceso a seguir consiste en generar múltiples muestras de los parámetros del modelo y, posteriormente, calcular las predicciones correspondientes para cada muestra.
Un ejemplo sería el cálculo de una predicción con un intervalor de confianza del 95%. Para ello, se entrena el modelo de red neuronal, extrayendo múltiples muestras de los parámetros del modelo con sus predicciones correspondientes. Finalmente, se toma el rango a partir de los umbrales marcados por los percentiles 2.5 y 97.5.

2.5 Razonamiento bajo incertidumbre en Graph Neural Networks (GNNs)

Las Graph Neural Networks o GNNs son una clase de modelos usados en el campo de la Inteligencia Artificial para para procesar datos estructurados y organizados mediante grafos, donde los nodos o vértices representan cualquier tipo de entidad y las aristas o enlaces representan las relaciones entre ellas. En este tipo de redes, se sigue el marco basado en un proceso de envío de mensajes entre nodos (*message passing*) que engloba gran parte de las definiciones espectrales y espaciales (Message Passing Neural Networks - MPNN). La idea de MPNN es simple: en cada capa de la red neuronal se aprenden nuevas características mediante la actualización de los mensajes de los vecinos. Es decir, cada nodo envía un mensaje a sus vecinos conectados, para posteriormente recibir todos los mensajes de sus vecinos y actualizar su representación agregando la información recibida de ellos. La ecuación de actualización quedaría de esta forma:

Remark 2.5.1 Message Passing Neural Networks (MPNN).

$$nodo_i^j = \text{UPDATE}(nodo_i^{j-1}, \text{ADD}(\text{MSG}(nodo_k^{j-1}))),$$

donde i es el nodo a actualizar con los mensajes de todos los nodos k conectados con i, y j es la capa de la red neuronal.

En otras palabras, se actualiza el nodo con los mensajes recibidos de los nodos vecinos/conectados con él. Las tres funciones UPDATE, ADD y MSG se pueden resumir de la siguiente forma:

- UPDATE: es una función que agrega los mensajes recibidos actualizando el estado del nodo destino. Este rol lo pueden jugar las funciones de activacion, la función identidad o incluso otras redes neuronales.
- ADD: se trata de una función sin número prefijado de valores de entrada, ya que no todos los nodos tienen el mismo número de conexiones/vecinos. Por ello, se suele optar por funciones como la suma o la media de los mensajes.
- MSG: se trata de una red neuronal simple, donde se toma la representación de un nodo (también puede ser el peso de la arista entre dos nodos) como entrada y calcula un mensaje de salida.

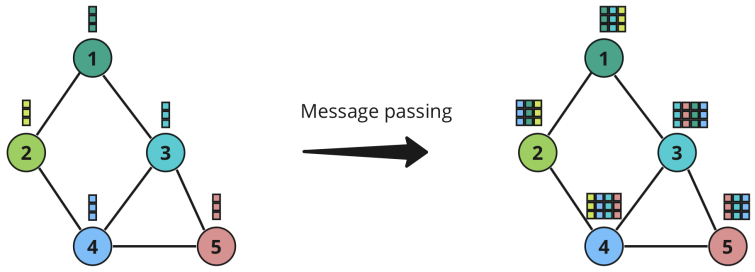

Figure 2.1: Ejemplo de Message passing. Cada nodo es actualizado con la suma los mensajes de los nodos con los que conecta.

Una forma de optimizar recursos y consumo de memoria es definir las funciones para implementarse como una multiplicación de matrices,

actualizando todo al mismo tiempo.

Niveles de representación: hay 3 niveles principales (nivel de nodo, arista y grafo).

1. Nodo: se usa para predecir o clasificar nodos de forma directa, como podría ser la predicción de si un usuario de una red social es hombre o mujer.
2. Arista: se usa para predecir una conexión entre dos nodos, combinando la representación de ambos. Un ejemplo sería predecir la probabilidad de que dos usuarios de una red social tengan la misma edad.
3. Grafo: el más complejo de los tres, puesto que hay que aunar en una representación todo el grafo a partir de las representaciones obtenidas por cada nodo. Un ejemplo sería el cálculo de la probabilidad de que un fármaco sea bueno, modelando donde cada compuesto es modelado como un grafo.

A nivel práctico, estos modelos son especialmente efectivos en una amplia gama de tareas, como pueden ser la clasificación de nodos (node classification), la predicción de enlaces (link prediction) y la recomendación de elementos de diferente índole.

Como toda tarea en los campos de la inteligencia artificial, y más concretamente en el aprendizaje automático, hay que manejar la incertidumbre inherente que hay en los problemas cotidianos, ya que los datos no se suelen encontrar en el formato adecuado, sino que suelen estar incompletos, ruidosos o sujetos a cambios.
Se podría decir que el razonamiento bajo incertidumbre en GNN se refiere a la capacidad de estos modelos para manejar y modelar la incertidumbre en los datos de entrada del modelo, en concreto los pesos de las aristas del grafo inicial y los valores de sus nodos, y en las predicciones que generan tras aplicar el modelo.

Ejemplo.

Enunciado
Se pretende predecir la probabilidad de que un usuario haga clic en un anuncio. Para ello, se representa la red social mediante un grafo que represente las conexiones entre los usuarios y sus características. Mediante una GNN se podría modelar estas relaciones y realizar la predicción.

Se divide el problema en los siguientes pasos:

1. Construcción del grafo: se construye un grafo donde los nodos representan a los distintos usuarios de la red social y las aristas serían las interacciones entre ellos. Cada nodo tiene unas características asociadas, como podrían ser la edad, el género y/o la ubicación. El objetivo es predecir la probabilidad de que un usuario haga clic en un anuncio en función de estas características y las interacciones en el grafo.

2. Aplicación de una GNN: partiendo del grafo construido, se aplica el modelo para aprender las representaciones de los nodos y propagar la información a través de las aristas.
3. Análisis de la incertidumbre: durante la propagación, se realizan diferentes tareas que permiten analizar y capturar la incertidumbre inherente a los datos, como muestreos estocásticos de las variables inciertas (por ejemplo, las características de los usuarios y los pesos de las conexiones).
4. Muestreos adicionales: una vez que se completa la propagación, se realizan muestreos extra para obtener una serie de predicciones de la probabilidad de clic en el anuncio para cada usuario, y se promedian para obtener una estimación de la probabilidad y su incertidumbre asociada.
5. Toma de decisión: en función de la incertidumbre obtenida, se toma una decisión final.

Si se obtiene, por ejemplo, una probabilidad de clic para un usuario del 70% con una desviación estándar del 10% indicaría que el modelo tiene cierta confianza en su predicción, pero con una incertidumbre importante en torno a ese valor.

Por lo tanto, es importante separar qué aporta cada parte: la aplicación de una GNN en un problema concreto permite obtener buenas predicciones, y al mismo tiempo el razonamiento bajo incertidumbre aporta una medida de confianza de esas predicciones para saber si la información resultante puede ser útil o no.

Entrando en más detalle con el razonamiento bajo incertidumbre en GNNs, se analiza la incertidumbre en un problema cualquiera, como podría ser la ausencia de información completa sobre aristas o nodos de un grafo, cómo de variables son los datos de entrada, o la incertidumbre que pueda tener una predicción de un modelo. Para ahondar en esto, existen técnicas para tratarlo, explicando brevemente algunas de ellas, y profundizando en capítulos posteriores.

Propagación de la incertidumbre.

Se trata de propagar la incertidumbre a través de las aristas del grafo. En la práctica, se asignan distribuciones de probabilidad a los pesos de las aristas y se propagan a través de los enlaces del grafo. Para llevar a cabo esta propagación hay diferentes enfoques, como puede ser la lógica difusa, los campos aleatorios condicionales o las redes bayesianas. Por ejemplo, si se modela una red social donde los usuarios (nodos) interaccionan (aristas) entre ellos, se captura la incertidumbre mediante la relación de amistad entre ellos reflejada por una distribución de probabilidad en la arista, propagándola a través de las conexiones de la red.

Otra técnica para tratar la incertidumbre en GNNs sería el muestreo estocástico en el entrenamiento y la inferencia. De esta forma, no se obtendría una sola predicción determinística, sino que se generarían muchas muestras estocásticas que reflejen la variabilidad que tendrían los resultados, mediante el muestreo MonteCarlo o técnicas de *dropout* estocástico, entre otros, para aplicar muestreo aleatorio a las conexiones entre nodos. Un ejemplo sería un modelo clasificador de documentos,

donde no se asignaría una sola etiqueta, sino varias de ellas mediante un muestreo estocástico para medir la distribución de probabilidad sobre las posibles clasificaciones.

Definiciones

Estocástico: sinónimo de aleatorio.
Fuente: Oxford Languages

Otra propiedad de la propagación de la incertidumbre en las GNNs es la robustez obtenida a través de la asignación de distribuciones de probabilidad a los parámetros de un modelo,como podría darse en el aprendizaje bayesiano. Un ejemplo sería si con un modelo se pretende predecir el precio de una propiedad inmobiliaria, donde antes se asignaría un solo valor predictivo a cada inmueble, ahora se asigna una distribución de probabilidad del precio para reflejar su incertidumbre.

Por tanto, hay difernetes técnicas que deben estudiarse con cautela y decidir cual se adapta al problema a tratar, atendiendo a las especificaciones y requisitos del dominio del problema.

2.6 Aplicaciones de las Graph neural networks (GNNs)

Una vez definido qué son las GNNs, faltaría saber en qué problemas se pueden aplicar. En realidad hay un amplio abanico de campos y problemas que son susceptibles de ser representados mediante grafos. En esta sección de describen brevemente algunas de estas aplicaciones, cuyo objetivo es ayudar a expandir la mente y pensar todas las posibilidades que aportan dichas redes.

- Recomendación de contenido: existen multitud de plataformas para recomendaciones, desde las que tienen un enfoque más social (TripAdvisor), pasando por comercio electrónico (Amazon) o servicios de streaming (Netflix). En todas ellas, se usan GNNs para modelar las relaciones entre usuarios, productos y sus interacciones, de cara a generar recomendaciones más personalizadas y precisas. Por ejemplo, cuando Amazon recomienda un producto (nodo), como podría ser un trípode, que supone que puedes llegar a comprar porque otro usuario que compro una cámara igual que el que has comprado, también compro el trípode que te recomiendan. En este caso, las aristas serían las compras conjuntas entre productos (entre el trípode y la cámara habrá una conexión porque un usuario compro ambas).
- Análisis en redes sociales: las redes sociales son muy fáciles de ser representadas mediante grafos. Con las GNNs, se pueden analizar dichas redes para detectar e identificar comunidades, influencers, o incluso predecir comportamientos futuros y rastrear la propagación de información falsa difundida a través de ellas. En estas redes, los nodos son los diferentes usuarios y las aristas son las interacciones entre ellos.

- Predicción de enlaces: uno de los problemas típicos en teoría de grafos es la predicción de nuevos enlaces, que puede ser aplicado con GNNs a predecir que nuevas relaciones de amistad se pueden producir en una red social o incluso temas más técnicos, como la predicción de enlaces entre proteínas en una red biológica o la nueva composición de un fármaco.
- Modelado de materiales: en mecánica de materiales o química computacional, se usan las GNNs para predecir propiedades moleculares, como podrían ser la energía de enlace o la solubilidad, o la relación entre elementos que componen un material concreto, al aprender las relaciones y las estructuras y relaciones de las moléculas o el material compuesto en un grafo.
- Análisis de movilidad urbana: dentro de una ciudad hay múltiples problemas que pueden ser representados mediante grafos, como la movilidad urbana o temas de seguridad vial. En estos problemas, las GNNs pueden modelar redes de transporte y tráfico para predecir el flujo de vehículos, optimización de rutas y mejora de la eficiencia del metro o bus.
- Procesamiento del lenguaje natural: las GNNs pueden extraer información y relaciones complejas entre entidades en oraciones o documentos, para añadir funcionalidad como el autocompletado o el corrector ortográfico, entre otros.
- Segmentación en imágenes: cuando se tratan imágenes, algunas tareas pueden representarse también de esta forma, donde las GNNs se aplican en tareas de segmentación semántica en imágenes para identificar objetos concretos, e incluso establecer relaciones de contexto en la escena.
- Reconstrucción de escenas 3D: pasando a la reconstrucción de una escena, cualquier objeto se puede modelar en 3D mediante grafos. Las GNNs permiten ayudar en esa tarea, reconstruyendo cada objeto a partir de la información recibida y representada en un grafo.
- Aplicaciones de amistades y relaciones: como cualquier red social, ya mencionada antes, las GNNs para sugerir nuevas amistades a los usuarios basándose en sus intereses comunes.

Existen muchas más aplicaciones, que se irán mencionando a lo largo del libro. En la siguiente sección, se propone un ejemplo práctico desde cero.