Comparison of Single- and Multi-Objective Optimization Quality for Evolutionary Equation Discovery

Mikhail Maslyaev Alexander Hvatov

ITMO University, St. Petersburg, Russia

maslyaitis@gmail.com, alex_hvatov@itmo.ru

GECCO '23 Companion, July 15–19, 2023
DOI: 10.1145/3583133.3590601
Fuente: arXiv:2306.17038v1

RESUMEN

El descubrimiento evolutivo de ecuaciones diferenciales ha demostrado ser una herramienta útil para obtener ecuaciones con menos suposiciones a priori que los enfoques convencionales, como la regresión simbólica dispersa sobre una biblioteca completa de términos posibles. El campo del descubrimiento de ecuaciones abarca dos direcciones independientes. La primera es puramente matemática e involucra la diferenciación, el objeto de optimización y su relación con los espacios funcionales, entre otros aspectos. La segunda se enfoca exclusivamente en la formulación del problema de optimización. Ambos temas son de interés para mejorar la capacidad del algoritmo de manejar datos experimentales de una manera más cercana a la inteligencia artificial, sin necesidad de un preprocesamiento significativo ni conocimiento previo de su naturaleza.

En este artículo, consideramos la prevalencia de la optimización mono-objetivo, que solo tiene en cuenta la discrepancia entre los términos seleccionados en la ecuación, frente a la optimización multiobjetivo, que adicionalmente considera la complejidad de la ecuación obtenida. El enfoque de comparación propuesto se muestra en ejemplos de modelos clásicos: la ecuación de Burgers, la ecuación de ondas y la ecuación de Korteweg-de Vries.

PALABRAS CLAVE: Regresión simbólica, modelado de sistemas dinámicos, aprendizaje interpretable, ecuaciones diferenciales, regresión dispersa.

1. INTRODUCCIÓN

El desarrollo reciente de la inteligencia artificial ha otorgado gran importancia a los problemas de aprendizaje automático interpretable. En muchos casos, los usuarios valoran los modelos no solo por su capacidad de predecir el estado del sistema estudiado, sino también por su capacidad de proporcionar información sobre su funcionamiento. En el caso del modelado de procesos físicos, los modelos más adecuados suelen tener la forma

de ecuaciones diferenciales parciales. Por ello, muchos estudios recientes han buscado desarrollar el concepto de descubrimiento de ecuaciones diferenciales basado en datos.

En este artículo, el descubrimiento basado en datos implica obtener una ecuación diferencial a partir de un conjunto de mediciones empíricas que describen la dinámica de una variable dependiente en un determinado dominio. Además, los modelos basados en ecuaciones pueden incorporarse en flujos de trabajo de aprendizaje automático automatizado, que pueden incluir submodelos arbitrarios, según el enfoque discutido en el artículo [**?**].

Los primeros avances en el descubrimiento de ecuaciones diferenciales se lograron con algoritmos de regresión simbólica, como en [**?**]. Este algoritmo emplea programación genética para detectar el grafo que representa la ecuación diferencial. Uno de los grupos de técnicas más simples pero prácticas para la construcción de ecuaciones se basa en la regresión lineal dispersa (operador de reducción y selección absoluta mínima), introducida en los trabajos [**?**, **?**, **?**] y otros proyectos similares. Este enfoque tiene una flexibilidad limitada, con restricciones de aplicabilidad en casos de ecuaciones con coeficientes de baja magnitud y datos ruidosos. Este problema se aborda empleando inferencia bayesiana, como en [**?**], para estimar los coeficientes de la ecuación, tal como se menciona en el trabajo [**?**]. Para considerar la incertidumbre en el modelo resultante, la biblioteca de términos de aproximación puede estar sesgada estadísticamente [**?**]. Las redes neuronales informadas por la física (PINN, por sus siglas en inglés) constituyen la siguiente clase de herramientas para el descubrimiento de ecuaciones basado en datos, representando la dinámica del proceso mediante redes neuronales artificiales. La investigación principal en este tema se encuentra en el trabajo [**?**], mientras que avances recientes han incorporado tipos más complejos de redes neuronales en las PINN [**?**, **?**].

En estudios recientes [**?**, **?**], los algoritmos evolutivos han demostrado ser una herramienta bastante flexible para el descubrimiento de ecuaciones diferenciales, requiriendo solo unas pocas suposiciones sobre las propiedades del proceso. El problema se plantea como la minimización del error en la representación del proceso. La implementación de la optimización evolutiva multiobjetivo, introducida por primera vez para sistemas de ecuaciones diferenciales en [**?**], parece ser un enfoque viable para mejorar la calidad de la búsqueda de ecuaciones, operando con menos suposiciones iniciales y proporcionando una mayor diversidad entre los candidatos procesados. Criterios adicionales pueden representar otras propiedades valiosas de los modelos construidos, como la concisión.

Este estudio compara el rendimiento de la optimización mono-objetivo y multiobjetivo. En particular, se plantea la hipótesis de que la optimización multiobjetivo crea y preserva la diversidad en la población y, por lo tanto, puede alcanzar mejores valores en la función de aptitud que el enfoque mono-objetivo. La comparación teórica muestra que los algoritmos multiobjetivo permiten escapar de mínimos locales siempre que el número de objetivos sea razonablemente pequeño [**?**]. Para aplicaciones de descubrimiento de ecuaciones, los paisajes de funciones tienen una estructura más compleja, por lo que una mayor diversidad en la población puede beneficiar la calidad del resultado.

2. DESCRIPCIÓN DEL ALGORITMO

La identificación de ecuaciones diferenciales basada en datos opera en problemas de selección de un modelo para la dinámica de la variable

$$u = u(t, x)$$

en un dominio espaciotemporal $(0, T) \times \Omega$, que está descrito implícitamente por una ecuación diferencial (Ec. 1) con condiciones iniciales y de contorno correspondientes. Se puede asumir que el orden de la ecuación desconocida puede ser arbitrario, pero relativamente bajo (generalmente de segundo o tercer orden).

$$\mathcal{F}\Big(t, x, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}\Big) = 0 \tag{1}$$

Tanto los enfoques de optimización mono-objetivo como multiobjetivo tienen el mismo núcleo de representación "tipo grafo" de una ecuación diferencial (codificación) y operadores evolutivos similares, que se describirán más adelante.

2.1. Representación de la ecuación diferencial

Para representar la ecuación diferencial candidata, se emplea una estructura de grafo computacional. Se utiliza una estructura de grafo de tres capas fijas para evitar estructuras inviables, relacionadas con la construcción de grafos sin restricciones y problemas de sobreajuste presentes en la regresión simbólica. Los nodos del nivel más bajo contienen tokens, mientras que los nodos intermedios y la raíz realizan operaciones de multiplicación y suma.

Las ecuaciones basadas en datos toman la forma de una combinación lineal de términos producto, representados por la multiplicación de derivadas, otras funciones y un coeficiente de valor real (Ec. 2).

$$\mathcal{F}'\Big(t, x, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}\Big) = \sum_i c_i \prod_j f_{ij} = 0 \tag{2}$$

Aquí, los factores f_{ij} se seleccionan de un conjunto de funciones elementales definidas por el usuario, denominadas tokens. El problema de búsqueda de ecuaciones se transforma en la detección de un conjunto óptimo de tokens para representar la dinámica de $u(t, x)$ y en la formación de la ecuación mediante la evaluación de los coeficientes $c = (c_1, \ldots, c_n)$.

Durante la búsqueda de ecuaciones, se opera con tensores de valores de tokens, evaluados en mallas $u_W = u(t_W, x_W)$ en el dominio procesado $(0, T) \times \Omega$.

La promoción de la escasez en la ecuación se logra filtrando términos nominales con bajo poder predictivo y se implementa mediante regresión LASSO. Para cada individuo, un término (sin pérdida de generalidad, se puede asumir que es el n-ésimo término) se marca como el "lado derecho de la ecuación" con el propósito de filtrar términos y calcular coeficientes. Los términos

$$\theta_i = \prod_j f_{ij}$$

se emparejan con coeficientes de valor real obtenidos del subproblema de optimización de la Ec. 3. Finalmente, los coeficientes de la ecuación se detectan mediante regresión lineal.

$$c' = \arg\min_c \left(\left| \sum_{i, i \neq n} c'_i \prod_j f_{ij} - \prod_j f_{nj} \right|^2 + \lambda \|c'\|_1 \right) \tag{3}$$

En la inicialización del algoritmo, los grafos de ecuaciones se construyen aleatoriamente para cada individuo a partir de los conjuntos de tokens definidos por el usuario, con varias suposiciones sobre las estructuras de las "ecuaciones plausibles".

2.2. Mecánica de los operadores evolutivos implementados

Para dirigir la búsqueda de ecuaciones óptimas, se han implementado operadores evolutivos estándar de mutación y cruce. Aunque la mecánica de la optimización mono-objetivo y multiobjetivo difiere en el algoritmo, funcionan de manera similar en la etapa de aplicación de operadores que modifican la estructura de la ecuación. Con la codificación tipo grafo de las ecuaciones candidatas, los operadores pueden representarse como cambios introducidos en sus subgrafos.

Las propiedades del algoritmo para explorar estructuras se proporcionan mediante operadores de mutación, que operan mediante intercambios aleatorios de tokens y términos. El número de términos a cambiar no tiene límites estrictos. Para tokens con parámetros $(p_1, \ldots, p_n) \in \mathbb{R}^{n-1}$, como una representación paramétrica de una variable dependiente externa desconocida, los parámetros también se optimizan: la mutación se realiza con un incremento aleatorio gaussiano.

Para combinar elementos estructurales de ecuaciones mejores, se implementa el operador de cruce. Las interacciones entre ecuaciones progenitoras se llevan a cabo a nivel de términos. Los conjuntos de pares de términos de las ecuaciones progenitoras se dividen en tres grupos:

1. **Términos idénticos** (presentes en ambas ecuaciones)
2. **Términos similares** (presentes en ambas ecuaciones, pero con parámetros diferentes o tokens ligeramente distintos)
3. **Términos únicos** (presentes solo en una ecuación)

El cruce ocurre en los dos últimos grupos. Para el segundo grupo, se manifiesta como un intercambio de parámetros entre los progenitores: los nuevos parámetros se seleccionan en un intervalo entre los valores de los progenitores.

El cruce entre términos únicos funciona como un intercambio completo entre ellos. La construcción de pares de intercambio entre estos tokens es completamente aleatoria.

2.3. Optimización de la métrica de calidad de la ecuación

La selección de la función de optimización distingue múltiples enfoques en la búsqueda de ecuaciones diferenciales. Un problema de optimización más trivial puede expresarse como la Ec. 4, donde se asume que el operador de la ecuación $\mathcal{F}'(u) = 0$ es cero, como en la Ec. 2.

$$J\Big(\mathcal{F}'(u)\Big) = \Big\|\mathcal{F}'(u)\Big\|^2 = \left|\sum_i c_i \prod_j f_{ij}\right|^2 \to \min_{c_i, f_{ij}} \tag{4}$$

Una función de optimización más compleja es la norma de la discrepancia entre los valores de entrada de la variable modelada y la solución propuesta por la ecuación diferencial del algoritmo, estimada en la misma malla (Ec. 5).

$$J_{data}\Big(\mathcal{F}'(u)\Big) = \|u - \hat{u}\|^2 \to \min_{c_i, f_{ij}} \tag{5}$$

Aquí, $\hat{u}$ representa los valores estimados por el modelo. El método de solución automática para la evaluación de la calidad de las ecuaciones candidatas (Ec. 6) se introduce en [?].

$$\mathcal{F}'(u) = 0 : \sum_i c_i \prod_j f_{ij} = 0 \tag{6}$$

Mientras que ambas métricas de calidad en las Ecs. (4) y (5) proporcionan una buena convergencia bajo condiciones ideales, la presencia de datos ruidosos puede hacer que la discrepancia del operador diferencial (Ec. 4) sea una métrica poco fiable. El algoritmo de solución automática tiene un alto costo computacional, ya que entrena una red neuronal para satisfacer la ecuación discretizada y los operadores de contorno.

Para el método de optimización mono-objetivo en este estudio, se emplea un algoritmo evolutivo simple que minimiza una de las funciones objetivo mencionadas. Dado que los experimentos se realizan con datos sintéticos sin ruido, se adopta el enfoque basado en la discrepancia.

2.4. Aplicación de la optimización multiobjetivo

Además de la representación del proceso, la concisión también es valiosa para mejorar la interpretabilidad del modelo. Por lo tanto, se introduce naturalmente una métrica para esta propiedad en la Ec. 7, donde se cuenta el número total de tokens (n_i para cada término i).

$$J_{simp}\Big(\mathcal{F}'(u)\Big) = \sum_i n_i \cdot \mathbf{1}_{\{c_i \neq 0\}} \tag{7}$$

El algoritmo **MOEA/DD** ha demostrado ser efectivo para el descubrimiento de ecuaciones diferenciales basadas en datos.

3. ESTUDIO EXPERIMENTAL

Esta sección del artículo está dedicada al estudio de las propiedades del marco de descubrimiento de ecuaciones. Como principal objeto de interés, designamos la diferencia entre las ecuaciones derivadas mediante optimización de un solo objetivo y optimización multiobjetivo. La validación se realizó en conjuntos de datos sintéticos, donde la variable dependiente modelada se obtiene resolviendo una ecuación ya conocida y estudiada.

Las pruebas se llevaron a cabo en tres casos:

- **Ecuación de ondas**
- **Ecuación de Burgers**
- **Ecuación de Korteweg-de Vries (KdV)**

Estos casos fueron seleccionados debido a las propiedades únicas de cada ecuación. Los algoritmos fueron probados con los siguientes parámetros:

- **Optimización de un solo objetivo:** 64 iteraciones evolutivas.
- **Optimización multiobjetivo:** 8 iteraciones con una población de 8 ecuaciones candidatas.

Esto resultó en un **consumo de recursos aproximadamente similar** entre ambos enfoques. Cada configuración se probó en **10 ejecuciones independientes** para garantizar validez estadística.

El principal indicador de calidad de las ecuaciones en nuestro estudio es el **análisis estadístico** de la media de la función objetivo

$$\mu = \mathbb{E}\Big[J\big(\mathcal{F}'\big)\Big]$$

y la varianza

$$\sigma^2 = \mathrm{Var}\Big(J\big(\mathcal{F}'\big)\Big)$$

entre diferentes ejecuciones.

Estudio de caso: ecuación de ondas

La primera ecuación analizada fue la **ecuación de ondas** (Ec. 8), con las condiciones de contorno e iniciales necesarias. La ecuación se resolvió utilizando Wolfram Mathematica en el dominio $(x, t) \in [0, 1] \times [0, 1]$ en una **malla de tamaño 101 $\times$ 101**.

$$\frac{\partial^2 u}{\partial t^2} = 0{,}04 \frac{\partial^2 u}{\partial x^2} \tag{8}$$

Se utilizó diferenciación numérica para estimar las derivadas. Debido a la estructura relativamente simple de la ecuación de ondas, tanto los enfoques de optimización de un solo objetivo como los de multiobjetivo **convergieron exitosamente** a la estructura correcta. Sin embargo, este caso sirve como referencia para observar el comportamiento del algoritmo en condiciones ideales.

Estudio de caso: ecuación de Burgers

La segunda ecuación analizada fue la **ecuación de Burgers** (Ec. 9). Esta ecuación tiene una **estructura no lineal**, lo que la convierte en un caso de prueba más complejo. El problema se resolvió sin viscosidad, omitiendo el término $\nu \frac{\partial^2 u}{\partial x^2}$.

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0 \tag{9}$$

Al igual que en el caso de la ecuación de ondas, la ecuación se resolvió con Wolfram Mathematica. Sin embargo, dado que la función no es constante en un pequeño dominio, **las derivadas se calcularon analíticamente** en lugar de numéricamente.

Un desafío importante con esta ecuación es la **presencia de óptimos locales alternativos**, como la relación:

$$u_x\, u_t = u_{tt}$$

Esta estructura alternativa representa un **mínimo local** en términos de la función de optimización, lo que dificulta que el algoritmo escape y encuentre la ecuación correcta.

Estudio de caso: ecuación de Korteweg-de Vries (KdV)

La prueba final se realizó en la **ecuación de Korteweg-de Vries (KdV)**, mostrada en la Ec. 10. Este caso fue elegido debido a su **naturaleza no homogénea**, lo que hace que la búsqueda de la ecuación sea más difícil.

$$\frac{\partial u}{\partial t} + 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = \cos(t)\sin(t) \tag{10}$$

La presencia de términos de forzamiento externo ($\cos t \sin t$) en términos separados de la ecuación **complica aún más el proceso de descubrimiento**, ya que el algoritmo debe diferenciar correctamente entre dinámica interna y externa.

Resultados y análisis estadístico

La Tabla 1 resume el **error medio** (μ) y la **varianza** (σ^2) de los resultados de optimización para cada ecuación bajo ambos enfoques.

Cuadro 1: Resultados del descubrimiento de ecuaciones

Métrica	**Método**	**Ondas**	**Burgers**	**KdV**
μ	**Un solo objetivo**	5.72	2246.38	0.162
μ	**Multiobjetivo**	2.03	1.515	16.128
σ^2	**Un solo objetivo**	18.57	$4{,}41 \times 10^{7}$	$8{,}9 \times 10^{-3}$
σ^2	**Multiobjetivo**	0	20.66	$\approx 10^{-13}$

Los resultados indican que la **optimización multiobjetivo mejora significativamente el rendimiento**, especialmente en los casos de **ecuaciones complejas** (Burgers y KdV). El enfoque multiobjetivo resulta en **una menor varianza**, lo que significa que encuentra soluciones mejores de manera más consistente, evitando mínimos locales deficientes.

La Figura 1 muestra una comparación gráfica de los valores finales de la **función objetivo** para los métodos de optimización de un solo objetivo y multiobjetivo en las tres ecuaciones de prueba.

Figura 1: Calidad de las ecuaciones descubiertas

- **(a)** Ecuación de ondas.
- **(b)** Ecuación de Burgers.
- **(c)** Ecuación de Korteweg-de Vries.

Cada subgráfico compara la distribución de valores finales de la función objetivo entre **optimización de un solo objetivo (izquierda) y multiobjetivo (derecha)**.

Observaciones clave

1. **La optimización multiobjetivo mejora la convergencia**
   - Para **todas las ecuaciones**, el método multiobjetivo **reduce el error** en las ecuaciones descubiertas.
   - La mejora es particularmente notable en la **ecuación de Burgers**, donde la optimización de un solo objetivo tiene dificultades debido a la presencia de **mínimos locales**.
2. **La optimización multiobjetivo reduce la varianza**
   - La **varianza** (σ^2) de la calidad de las ecuaciones es significativamente menor en las ejecuciones multiobjetivo, lo que indica **mayor consistencia**.
   - Para la ecuación de KdV, **la varianza es prácticamente cero**, lo que significa que el enfoque multiobjetivo encuentra de manera confiable la **forma correcta de la ecuación**.
3. **La optimización de un solo objetivo no es confiable para ecuaciones complejas**
   - El **método de un solo objetivo converge con frecuencia a estructuras incorrectas** en casos no triviales (Burgers y KdV).
   - Esto se debe a **sobreajuste** y a la tendencia de quedar atrapado en **mínimos locales**.

Resumen

Este experimento demuestra que la **optimización multiobjetivo proporciona mejores resultados** en el descubrimiento de ecuaciones. Al equilibrar **precisión y complejidad**, los métodos multiobjetivo mejoran la convergencia, evitan mínimos locales y aumentan la robustez. Estas propiedades los convierten en un **enfoque prometedor para la modelización de ecuaciones diferenciales en aplicaciones del mundo real**.

4. CONCLUSIÓN

Este artículo examina las perspectivas de utilizar la **optimización multiobjetivo** para el descubrimiento de **ecuaciones diferenciales parciales** basado en datos. Aunque inicialmente fue introducida para abordar problemas de derivación de sistemas de ecuaciones diferenciales parciales, el **enfoque multiobjetivo** mejora la calidad general del algoritmo. La mejor convergencia, proporcionada por **una mayor diversidad en los candidatos individuales**, hace que el proceso sea más confiable en casos de ecuaciones con estructuras complejas, como se demostró en los ejemplos de las **ecuaciones de Burgers y Korteweg-de Vries**.

Estudios previos han indicado que el algoritmo es **confiable**, logrando converger a la ecuación correcta. Sin embargo, esta investigación propone un método para **mejorar la velocidad** con la que se identifican las estructuras correctas. Esta propiedad es especialmente valiosa para **aplicaciones del mundo real**, donde la incorporación de **conjuntos de datos grandes y completos** mejora la **resistencia al ruido** del enfoque.

El trabajo futuro se centrará en **introducir técnicas para la incorporación de conocimiento experto** en el proceso de búsqueda. Este concepto puede ayudar a generar **candidatos preferibles** o excluir **candidatos inviables** incluso antes de los costosos procedimientos de **cálculo de coeficientes** y **evaluación de aptitud**.

CODIGO DEL PROGRAMA

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
import matplotlib.pyplot as plt

# Generación de datos sintéticos (simulando 10,000 ejemplos)
# Supongamos que cada ejemplo tiene 20 características y queremos predecir una venta
num_examples = 10000
num_features = 20

np.random.seed(42)
X = np.random.randn(num_examples, num_features)
# Creamos una variable target con alguna relación lineal y algo de ruido
true_weights = np.random.randn(num_features, 1)
y = X.dot(true_weights) + np.random.randn(num_examples, 1) * 0.5

# Definición de la arquitectura de la red neuronal
def build_model():
    model = Sequential([
        Dense(64, activation='relu', input_shape=(num_features,)),
        Dense(64, activation='relu'),
        Dense(1)  # Capa de salida para regresión
    ])
    return model

# Función para entrenar el modelo con un optimizador dado
def train_model(optimizer, epochs=50, batch_size=32):
    model = build_model()
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    history = model.fit(X, y, epochs=epochs, batch_size=batch_size,
                        validation_split=0.2, verbose=0)
    return history

# Lista de optimizadores a comparar
optimizers = {
    'SGD': SGD(learning_rate=0.01),
    'Adam': Adam(learning_rate=0.001),
    'RMSProp': RMSprop(learning_rate=0.001)
```

```
}

histories = {}

# Entrenar la misma arquitectura con cada optimizador
for opt_name, opt in optimizers.items():
    print(f'Entrenando con {opt_name}...')
    histories[opt_name] = train_model(opt)

# Comparación de la velocidad de convergencia y error final
plt.figure(figsize=(10, 6))
for opt_name, history in histories.items():
    plt.plot(history.history['val_loss'], label=opt_name)
plt.title('Comparación de validación MSE entre optimizadores')
plt.xlabel('Época')
plt.ylabel('Error cuadrático medio (MSE)')
plt.legend()
plt.grid(True)
plt.show()

# Imprimir error final de cada optimizador
for opt_name, history in histories.items():
    final_loss = history.history['val_loss'][-1]
    print(f'{opt_name} - Error final en validación: {final_loss:.4f}')
```

EXPLICACION DEL CODIGO CON RELACION AL ARTICULO

A continuación se presenta una explicación en profundidad del código, relacionándolo conceptualmente con los temas expuestos en el texto sobre el descubrimiento de ecuaciones diferenciales y la optimización (tanto mono-objetivo como multiobjetivo). Aunque el código y el artículo abordan problemáticas y dominios diferentes, existen puntos de convergencia en cuanto a la formulación y el proceso de optimización que vale la pena destacar.

1. Optimización en el Código y en el Descubrimiento de Ecuaciones

En el Código:

- **Objetivo Único (Mono-objetivo):** El código entrena una red neuronal para un problema de regresión (predicción de una venta) minimizando una única función de pérdida: el *Error Cuadrático Medio (MSE)*.

  ```
  model.compile(optimizer=optimizer, loss='mean_squared_error')
  ```

  Aquí se busca ajustar los parámetros (pesos y sesgos) de la red para que la salida predicha se aproxime lo más posible a los valores reales y.

- **Comparación de Estrategias de Optimización:** Se implementa un proceso en el que se entrena la misma arquitectura utilizando tres algoritmos de optimización diferentes (SGD, Adam y RMSProp) para comparar su convergencia y desempeño en la minimización de la función de pérdida.

```
optimizers = {
    'SGD': SGD(learning_rate=0.01),
    'Adam': Adam(learning_rate=0.001),
    'RMSProp': RMSprop(learning_rate=0.001)
}
```

  Esta comparación es similar a analizar cómo diferentes métodos de optimización afectan la convergencia y la calidad final del modelo, ya que cada optimizador tiene sus propias estrategias para actualizar los parámetros y evitar problemas como quedar atrapado en mínimos locales.

En el Texto:

- **Optimización en el Descubrimiento de Ecuaciones Diferenciales:** El artículo se centra en la identificación de ecuaciones diferenciales a partir de datos experimentales, lo cual implica definir y optimizar una función de aptitud que mide la discrepancia entre la ecuación candidata y los datos observados.

  - **Optimización Mono-objetivo:** Se minimiza una función de error que, en condiciones ideales y con datos sintéticos sin ruido, puede ser la discrepancia entre la salida del modelo (o el operador diferencial) y cero:

$$J\big(\mathcal{F}'(u)\big) = \left| \sum_i c_i \prod_j f_{ij} \right|^2 \to \min_{c_i, f_{ij}}$$

  - **Optimización Multiobjetivo:** Se introduce un segundo objetivo relacionado con la *complejidad* (o concisión) de la ecuación, penalizando aquellos modelos que utilizan demasiados términos o tokens. Esto ayuda a evitar soluciones sobreajustadas y favorece la interpretabilidad:

$$J_{simp}\big(\mathcal{F}'(u)\big) = \sum_i n_i \cdot \mathbf{1}_{\{c_i \neq 0\}}$$

    De esta forma, la búsqueda no solo se centra en minimizar el error, sino también en obtener modelos más simples y robustos.

2. Puntos en Común: Función de Pérdida y la Búsqueda de Soluciones Óptimas

Función de Pérdida:

- *En el Código:* La función de pérdida es el MSE, que cuantifica la discrepancia entre las predicciones de la red y los valores reales. Este valor se utiliza para guiar la actualización de los parámetros durante el entrenamiento.
- *En el Descubrimiento de Ecuaciones:* La función objetivo puede ser similar a la minimización de la discrepancia de la ecuación diferencial propuesta con respecto a los datos medidos (o, en un caso ideal, la evaluación de $\mathcal{F}'(u)$ en la malla de datos). En ambos casos, el proceso de optimización se basa en ajustar parámetros para reducir el error:
  - En el código, se ajustan pesos de una red neuronal.
  - En el artículo, se ajustan coeficientes y la estructura (selección de tokens) de la ecuación diferencial.

Evitar Mínimos Locales y Lograr Convergencia:

- *Optimización en Redes Neuronales:* La comparación entre optimizadores en el código (SGD, Adam, RMSProp) ilustra cómo distintos métodos pueden tener diferentes capacidades para escapar de mínimos locales y converger de forma más rápida o estable.
- *Optimización en el Descubrimiento de Ecuaciones:* El texto destaca que la optimización multiobjetivo, al equilibrar error y complejidad, puede favorecer la diversidad en la población de soluciones y ayudar a escapar de mínimos locales, un problema que afecta especialmente a ecuaciones complejas como la de Burgers o la de Korteweg-de Vries.

3. Analogías Metodológicas y Aplicaciones Prácticas

Modelo Basado en Datos y Validación:

- *En el Código:* Se generan datos sintéticos para entrenar y validar la red. La división en conjuntos de entrenamiento y validación (80 % - 20 %) permite evaluar la capacidad de generalización del modelo, lo que es crucial para evitar el sobreajuste.
- *En el Descubrimiento de Ecuaciones:* Aunque el proceso puede implicar técnicas de regresión dispersa o métodos evolutivos, también se trabaja con datos (ya sean experimentales o simulados) para ajustar y validar la ecuación diferencial candidata. La validación puede involucrar comparar la salida del modelo (por ejemplo, la solución de la ecuación) contra datos medidos, similar a lo que se hace en el entrenamiento de una red neuronal.

Estrategia de Comparación y Selección del Mejor Método:

- *En el Código:* Se compara el desempeño final (error de validación) de cada optimizador para determinar cuál logra una mejor convergencia y menor error. Esta comparación es esencial para seleccionar el algoritmo más adecuado para el problema en cuestión.

- *En el Descubrimiento de Ecuaciones:* Se comparan dos enfoques: el de optimización mono-objetivo y el multiobjetivo. Los experimentos muestran que, en particular para ecuaciones complejas, el método multiobjetivo ofrece soluciones con menor error y menor varianza, lo que significa mayor consistencia y robustez en la solución.

Implementación y Herramientas Computacionales:

- *Uso de Frameworks Modernos:* El código utiliza TensorFlow y Keras, herramientas muy utilizadas en inteligencia artificial y en aplicaciones de modelado basado en datos, incluyendo las *redes neuronales informadas por la física (PINN)*, que se mencionan en el texto. Estas herramientas permiten un desarrollo rápido y flexible de modelos complejos y son parte del arsenal moderno para resolver problemas de optimización.

4. Síntesis y Reflexión Final

- **Enfoque Mono-objetivo vs. Multiobjetivo:** El código se centra en un enfoque de optimización mono-objetivo (minimizar el MSE), lo cual es adecuado en situaciones donde se conoce la naturaleza de los datos y se cuenta con datos sintéticos sin ruido. Esto es comparable a los casos "ideales" mencionados en el texto, donde la optimización se basa únicamente en la discrepancia. Por otro lado, el artículo muestra que en escenarios más complejos (como el descubrimiento de ecuaciones diferenciales en presencia de ruido o estructuras no lineales) es ventajoso incorporar objetivos adicionales (como la simplicidad del modelo) para lograr resultados más robustos y interpretables.

- **Relevancia de la Selección del Algoritmo de Optimización:** La comparación de diferentes optimizadores en el código ilustra cómo la elección del método de optimización puede afectar la convergencia y la calidad final del modelo, una lección que es aplicable de forma análoga en el contexto evolutivo para el descubrimiento de ecuaciones. La diferencia entre optimizadores en ambos contextos puede ser la clave para evitar caer en mínimos locales y alcanzar soluciones óptimas que sean tanto precisas como interpretables.

- **Extensibilidad del Enfoque:** Aunque el ejemplo de código es un caso de regresión tradicional, los conceptos de ajustar parámetros para minimizar un error y la importancia de evaluar la calidad del modelo (por ejemplo, mediante la validación) son fundamentales en ambos escenarios. Esto sugiere que técnicas desarrolladas en el campo del aprendizaje automático pueden inspirar y complementar estrategias en el descubrimiento de ecuaciones diferenciales, especialmente cuando se integran enfoques multiobjetivo para equilibrar precisión y complejidad.

Conclusión: El código presentado es un ejemplo de optimización aplicada en redes neuronales mediante la minimización de una función de pérdida (MSE) utilizando diferentes algoritmos de optimización. Este proceso guarda una analogía conceptual con el proceso de descubrimiento de ecuaciones diferenciales basado en datos, donde se busca ajustar parámetros (ya sean pesos en una red o coeficientes en una ecuación) para

minimizar una discrepancia, pudiendo además incorporar criterios adicionales (como la complejidad) para mejorar la interpretabilidad y robustez del modelo. Ambos enfoques resaltan la importancia de la estrategia de optimización y la elección del método adecuado para evitar mínimos locales y lograr soluciones consistentes y significativas en aplicaciones del mundo real.

EXPLICACION DETALLADA DEL CODE

A continuación, se presenta una explicación paso a paso y detallada del código, indicando el objetivo de cada sección y cómo funciona dentro del flujo completo:

1. Importación de librerías

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
import matplotlib.pyplot as plt
```

- `numpy (np)`: para el manejo de arreglos y funciones matemáticas.
- `tensorflow (tf)` y `keras`: para construir y entrenar redes neuronales.
- `Sequential` y `Dense`: para definir la arquitectura del modelo.
- `SGD`, `Adam` y `RMSprop`: distintos algoritmos de optimización.
- `matplotlib.pyplot (plt)`: para graficar la evolución del error de validación.

2. Generación de datos sintéticos

```
# Generación de datos sintéticos (simulando 10,000 ejemplos)
# Supongamos que cada ejemplo tiene 20 características y queremos predecir una venta
num_examples = 10000
num_features = 20

np.random.seed(42)
X = np.random.randn(num_examples, num_features)
# Creamos una variable target con alguna relación lineal y algo de ruido
true_weights = np.random.randn(num_features, 1)
y = X.dot(true_weights) + np.random.randn(num_examples, 1) * 0.5
```

Se crean:

- X: una matriz de dimensiones 10000×20 con datos generados a partir de una distribución normal.
- y: la variable objetivo, definida como una combinación lineal de X con la adición de ruido.

3. Definición de la arquitectura de la red neuronal

```
def build_model():
    model = Sequential([
        Dense(64, activation='relu', input_shape=(num_features,)),
        Dense(64, activation='relu'),
        Dense(1)  # Capa de salida para regresión
    ])
    return model
```

Se define una función que construye una red neuronal secuencial:

- Dos capas ocultas densas con 64 neuronas y activación ReLU.
- Una capa de salida con 1 neurona, adecuada para problemas de regresión.

4. Función para entrenar el modelo con un optimizador dado

```
def train_model(optimizer, epochs=50, batch_size=32):
    model = build_model()
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    history = model.fit(X, y, epochs=epochs, batch_size=batch_size,
                        validation_split=0.2, verbose=0)
    return history
```

Esta función:

- Compila el modelo usando el optimizador especificado y la función de pérdida MSE.
- Entrena el modelo durante 50 épocas con un batch de 32 ejemplos y utiliza el 20 % de los datos para validación.
- Devuelve el objeto `history` que almacena la evolución del error.

5. Lista de optimizadores a comparar

```
optimizers = {
    'SGD': SGD(learning_rate=0.01),
    'Adam': Adam(learning_rate=0.001),
    'RMSProp': RMSprop(learning_rate=0.001)
}
```

Se definen tres optimizadores con sus respectivas tasas de aprendizaje para comparar su desempeño durante el entrenamiento.

6. Entrenamiento de la misma arquitectura con cada optimizador

```
histories = {}

# Entrenar la misma arquitectura con cada optimizador
for opt_name, opt in optimizers.items():
    print(f'Entrenando con {opt_name}...')
    histories[opt_name] = train_model(opt)
```

Se recorre el diccionario de optimizadores y se entrena un modelo para cada uno, almacenando el historial en el diccionario `histories`.

7. Comparación de la velocidad de convergencia y error final

7.1. Graficar la función de pérdida en validación

```
plt.figure(figsize=(10, 6))
for opt_name, history in histories.items():
    plt.plot(history.history['val_loss'], label=opt_name)
plt.title('Comparación de validación MSE entre optimizadores')
plt.xlabel('Época')
plt.ylabel('Error cuadrático medio (MSE)')
plt.legend()
plt.grid(True)
plt.show()
```

Se crea una gráfica que muestra cómo varía el error de validación a lo largo de las épocas para cada optimizador.

7.2. Imprimir el error final de cada optimizador

```
for opt_name, history in histories.items():
    final_loss = history.history['val_loss'][-1]
    print(f'{opt_name} - Error final en validación: {final_loss:.4f}')
```

Se imprime el error final (MSE en validación) obtenido tras la última época, permitiendo una comparación numérica entre los optimizadores.

Resumen conceptual del flujo del código

1. **Generación de datos:** Se crean datos sintéticos X e y para simular un problema de regresión.
2. **Definición del modelo:** Se construye una red neuronal simple con dos capas ocultas y una capa de salida para regresión.
3. **Entrenamiento del modelo:** El modelo se entrena utilizando tres optimizadores diferentes, y se compara el desempeño en función del error de validación.

4. **Visualización y análisis:** Se grafican las curvas de error de validación y se imprime el error final para evaluar la velocidad de convergencia y la precisión de cada método.

Conclusión: El código ejemplifica la generación de datos sintéticos, la construcción y entrenamiento de un modelo de red neuronal con distintos algoritmos de optimización, y la comparación de su rendimiento. Conceptualmente, este proceso guarda analogías con el proceso de descubrimiento de ecuaciones diferenciales, en el que se busca minimizar un error (o función de pérdida) y se compara la efectividad de diferentes estrategias de optimización para obtener modelos robustos y precisos.