In [2]:
```
import torch
import matplotlib.pyplot as plt
```

Aprendizaje

> Autor: Jorge García González (Universidad de Málaga)

> Última Actualización: 15/09/2025

> Asignatura: Programación para la Inteligencia Artificial

¿Qué entendemos por *aprender* en el contexto de las Redes Neuronales Artificiales? Ajustar los parámetros de un modelo.

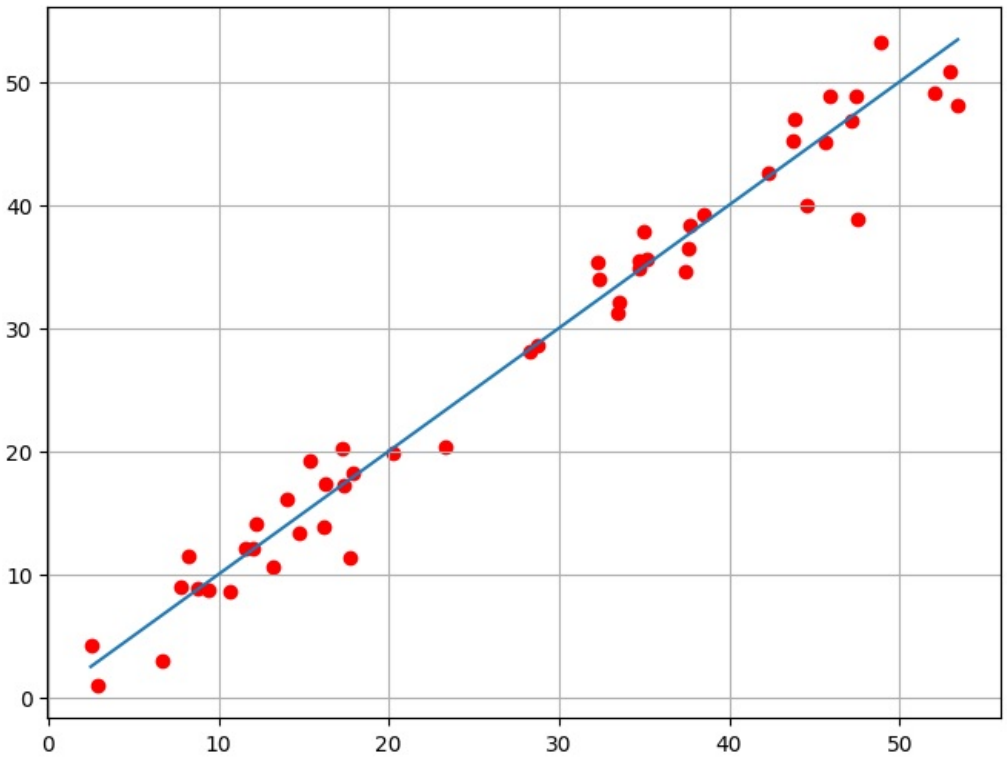
Antes de empezar con las redes, vamos a observar el fenómeno del ajuste de parámetros en un modelo mucho más sencillo. Vamos a hacer un modelo que ajuste una recta a partir de unos datos.

Recordemos que la ecuación de una recta en 2 dimensiones es y=m*x+b. Por tanto una recta solo tiene dos parámetros que ajustar, la m (inclinación) y la b (valor donde se corta el eje y).

Lo primero que necesitamos es una función que, dados los parámetros de una recta, nos devuelva un conjunto de puntos que formen parte de esa recta. Para hacerlo un poquito más realista vamos a añadirle ruido gaussiano a esos datos. Así los puntos no encajarán perfectamente en la recta, pero deberían estar cerca.

Los datos en problemas reales rara vez son perfectos y nuestros modelos son solo aproximaciones.

In [3]:
```
def get_data_from_a_line(m, b, n, sigma=2, range=50):
  """
  Función que devuelve datos a partir de una recta. Recordemos que la ecuación de la recta está definida por y=m
  Args:
    m: inclinación de la recta.
    b: ordenada en el origen de la recta.
    n: número de datos que queremos obtener.
    sigma: desviación típica para el ruido que añadiremos a los datos (2 por defecto).
    range: extremo del rango [0,rango] en el eje x en el que queremos situar los datos (50 por defecto).
  """
  x = range*torch.rand(n)                                          # Generamos los datos en el eje x.
  y = m*x+b                                                        # Obtemeos la posición correspondiente e
  noise_x = torch.normal(torch.zeros_like(y), sigma*torch.ones_like(y)) # Generamos un vector de ruido gaussianc
  noise_y = torch.normal(torch.zeros_like(y), sigma*torch.ones_like(y)) # Generamos un vector de ruido gaussianc
  x_with_noise = x + noise_x
  y_with_noise = y + noise_y

  data = torch.cat((x_with_noise[:,None], y_with_noise[:,None]), dim=1)
  return data

def plot_line_with_data(m,b,data):
  """
  Procedimiento para dibujar una recta y un conjunto de puntos.
  Args:
    m: inclinación de la recta.
    b: ordenada en el origen.
    data: tensor de puntos con tamaño Nx2.
  """

  x = data[:,0]
  y = data[:,1]
  plt.figure(figsize=(8,6))
  plt.scatter(x, y, marker='o', c='r')
  plt.plot([x.min(),x.max()], [m*x.min()+b,m*x.max()+b])
  plt.grid(True)
  plt.show()

data = get_data_from_a_line(1,0,50)
plot_line_with_data(1,0,data)
```



Una vez podemos generar un conjunto de datos "del que aprender", vamos a crear nuestro modelo de recta. Este modelo se trata solo de aplicar la ecuación de la recta a unas entradas dados unos parámetros de la misma. Nótese que si podemos asumir que nuestro modelo a aprender tiene esta estructura es porque nosotros (diseñadores del sistema) *sabemos* que es una recta. Este modelo será incapaz de aprender de manera precisa datos que no provengan de una recta. Es un modelo muy limitado, pero que se ajusta a nuestras necesidades actuales.

In [4]:
```
def line_model(m, b, inputs):
  """
  Función para modelar una recta.
  Args:
    m: inclinación de la recta.
    b: ordenada en el origen de la recta.
    inputs: vector con los valores de entrada para ser calculados por el modelo.
  """
  return m*inputs+b
```

Tenemos datos y un modelo que ajustar. ¿Ahora qué? ¿Cómo lo ajustamos?

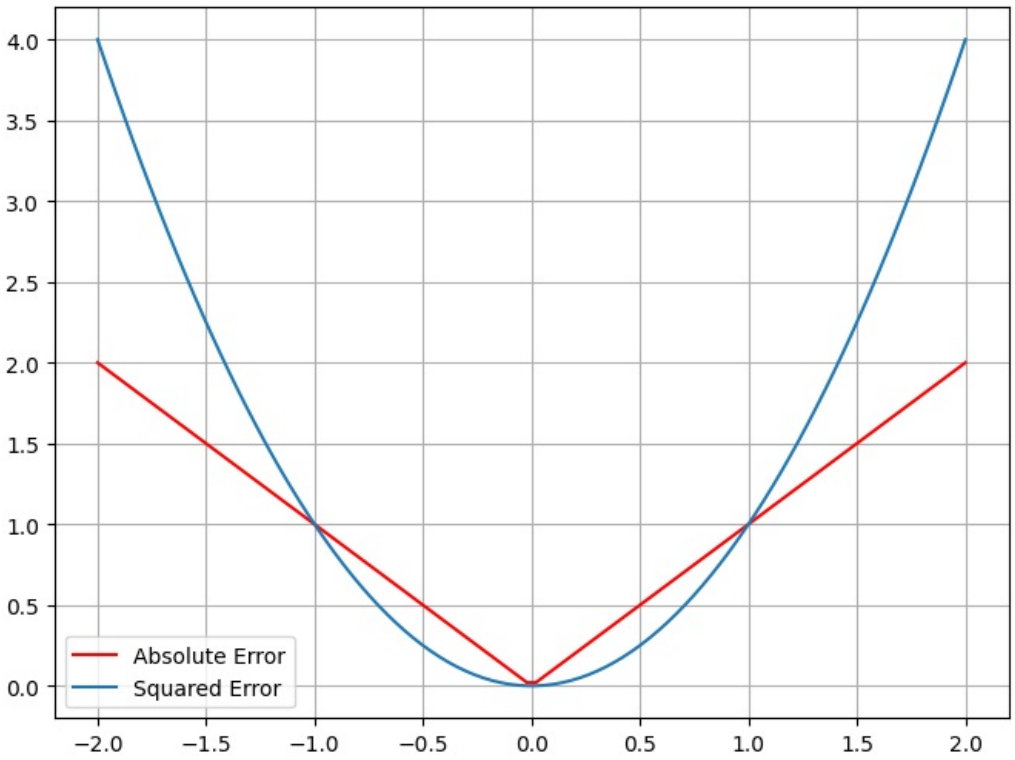
Por simplificar, vamos a a suponer que el valor de b (la ordenada en el origen) es siempre cero, así que solo tendríamos que ajustar la inclinación de una recta que pase por el (0,0) a unos datos obtenidos. No sabemos qué recta originaron los datos (bueno, sí que lo sabemos, pero vamos a fingir que no), así que nuestra estrategia va a ser ir probando y ver si la recta que vamos obteniendo se ajusta mejor o peor.

Para saber si se ajusta mejor o peor tenemos que poder cuantificar el *error* que cometemos en cada una de nuestras pruebas. Para eso vamos a definir un par de funciones de error (también llamadas funciones de pérdida o funciones de coste) muy habituales: el error absoluto y el error cuadrático.

In [5]:
```
def AE_loss(y_pred, y_true):
  return torch.abs(y_pred - y_true)

def SE_loss(y_pred, y_true):
  return (y_pred-y_true)**2
```

Puede ser llamativo el nombre que hemos decidido darle a los argumentos de estas funciones. y_pred hace referencia a la predicción que hace el modelo, mientras que y_true hace referencia a un valor que tomamos como verdadero. Estas funciones tienen este aspecto:

In [6]:
```
x = torch.linspace(-2,2,100)
ae = torch.abs(x)
se = x**2
plt.figure(figsize=(8,6))
plt.plot(x, ae, label="Absolute Error", c="r")
plt.plot(x, se, label="Squared Error")
plt.legend()
plt.grid(True)
plt.show()
```



Aunque su comportamiento general es parecido, hay algunas diferencias interesantes. La más obvia es que en valores de x ∈ (-1,1), el error cuadrático es menor que el error absoluto. Para el resto de los valores es igual (para {-1,1}) o mayor, y de hecho crece más rápido que el error absoluto. Comparemoslas en dos puntos distintos. Para x=0.5, el error absoluto señalará que hay un error de 0.5, mientras que el error cuadrático indicará que el error es de 0.25. Para x=2, el error absoluto indicará un error de 2, mientras que el error cuadrático indicará un error de 4. para x=3, el error absoluto indicará 3, el cuadrático indicará 9. ¡Es importante tener en cuenta el rango de valores en el que nos vamos a mover!

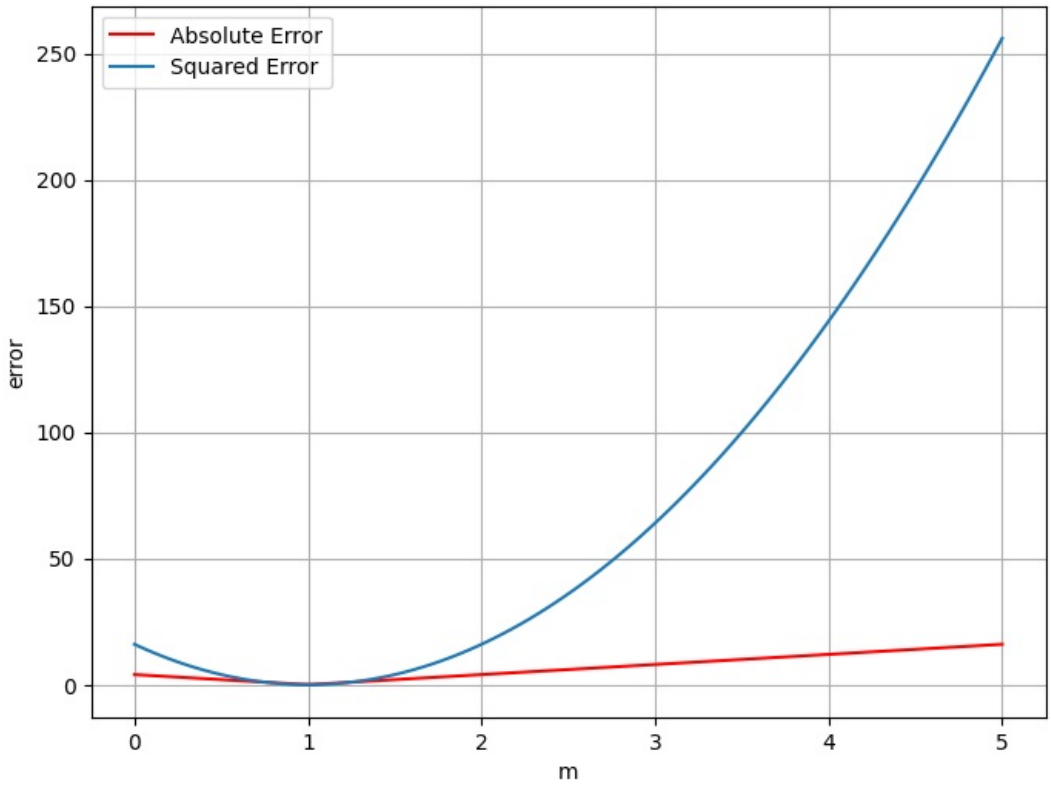
Hay una diferencia relevante más: el error cuadrático es derivable en todos los puntos, mientras que el error absoluto no es derivable en el 0. Por ahora no vamos a prestarle atención a eso, pero en el futuro volveremos a ese detalle.

Ahora que tenemos un par de maneras de cuantificar el *error* de nuestro problema. Vamos a ver cómo van encajando las piezas.

Si seguimos nuestra estrategia de *prueba y error*, lo que haríamos es tomar un punto cualquiera (x_n,y_n) de los datos de los que queremos aprender y lo vamos a *fijar*. Nuestros datos nos dicen que para x=x_n, el modelo tendría que tener y_n como resultado si estuviera bien ajustado. Ajustar el modelo en este caso, como hemos mencionado antes, consiste en ajustar el parámetro m (porque por simplificar el ejemplo asumimos b=0). Así que nuestras funciones de error lo que van a medir es la diferencia entre el resultado que consideramos cierto (y_n) y el resultado que obtenemos (x_n*m+b).

Vamos a ver qué aspecto tiene esa función de error según cómo modificamos m para un punto elegido arbitrariamente.

In [7]:
```
x_true, y_true = (4,4)
ae_list = []
se_list = []
b=0

m= torch.linspace(0,5,100)
y_pred = line_model(m,b,x_true)
ae_error = AE_loss(y_pred, y_true)
se_error = SE_loss(y_pred, y_true)

plt.figure(figsize=(8,6))
plt.plot(m, ae_error, label="Absolute Error", c="r")
plt.plot(m, se_error, label="Squared Error")
plt.legend()
plt.grid(True)
plt.xlabel("m")
plt.ylabel("error")
plt.show()
```



Como era de esperar, alterar la inclinación de nuestro modelo de recta tiene como consecuencia un cambio en el error. Podemos ver que el error se minimiza en m=1. Esto tiene sentido, ya que el punto (4,4) forma parte de la recta y=x. Si no hubieramos asumido b=0, habría infinitas soluciones que minimizan un punto dado.

Con puntos que pertenezcan a rectas distintas obtendríamos curvas distintas.

Ahora que hemos visto cómo encaja la función de error con el modelo y los datos, vamos a intentar definir un algoritmo de optimización iterativo. Una opción sería probar valores aleatorios para m y cada vez que encontremos uno que tenga un error más bajo, actualizar m. Es una opción totalmente aleatoria que para un caso sencillo como este podría parecer una opción aceptable, pero vamos a ir un paso más allá. Vamos a aprovechar la función de error para saber en qué dirección actualizar m.

Dado un punto (x_n,y_n) de los datos de entrenamiento y un valor del parámetro $m=m_s$, vamos a estudiar qué ocurre si incrementamos o disminuimos ligeramente el parámetro m. Para eso vamos a definir un valor Δ y vamos a estudiar qué recta se ajusta mejor al punto dado, la recta con parámetro $m=m_s+\Delta$ o la recta con parámetro $m=m_s-\Delta$.

La diferencia entre uno y otro la vamos a llamar ratio de cambio del error y vamos a actualizar el parámetro m con ese por un ratio de aprendizaje.

Usaremos el error absoluto para evitar retios de cambio del error muy grandes.

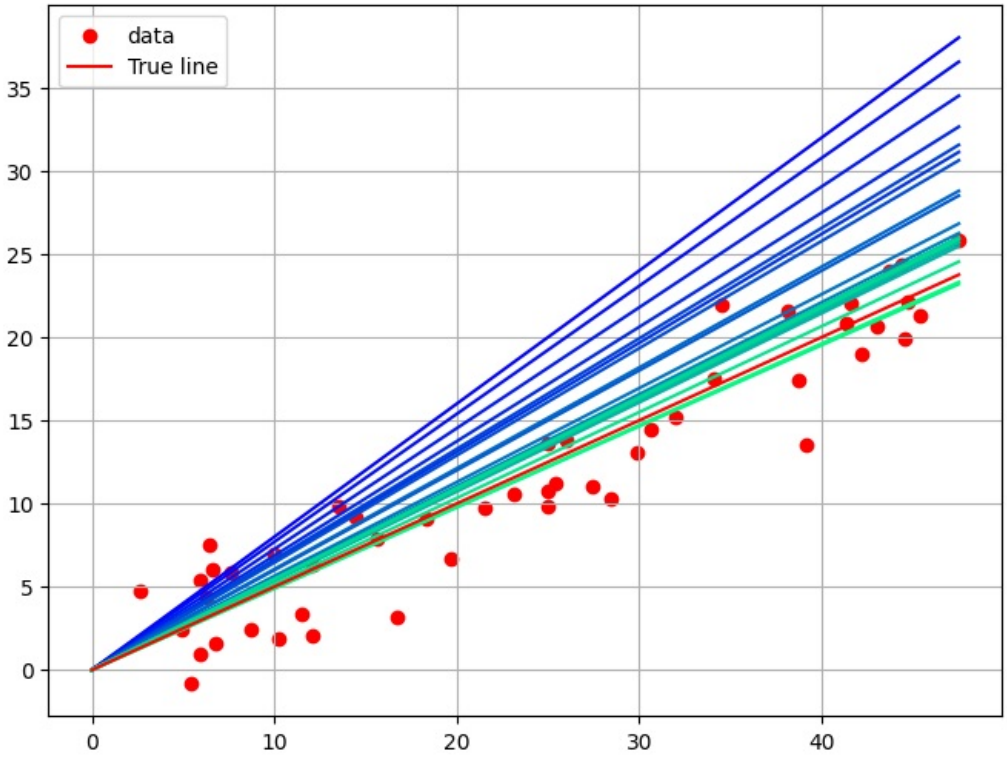
In [8]:
```
learning_rate = 1e-3
delta = 0.1
steps = 20
loss_fn = AE_loss

true_m, true_b = [0.5,0]
data = get_data_from_a_line(true_m, true_b, 50)
m = 0.8                                          # Inicializamos la m a 0.8 con propósitos meramente didácticos.
b = 0
line_evolution = [m]

for step in range(steps):
  random_index_to_select_data = torch.randint(low=0, high=data.size()[0], size=(1,))[0]
  random_point_from_data = data[random_index_to_select_data]
  x_true = random_point_from_data[0]
  y_true = random_point_from_data[1]

  loss_rate_of_change_m = (loss_fn(line_model(m-delta,b,x_true), y_true) - loss_fn(line_model(m+delta,b,x_true),
  print(f"--------- step {step}")
  print(f"m actual: {m}")
  print(f"Dato escogido aleatoriamente (x_true,y_true): {x_true,y_true}")
  print(f"Valor de y predicho: {line_model(m,b,x_true)}")
  print(f"Valor de y predicho 'bajando la m': {line_model(m-delta,b,x_true)}, error: {loss_fn(line_model(m-delta
  print(f"Valor de y predicho 'aumentando la m': {line_model(m+delta,b,x_true)}, error: {loss_fn(line_model(m+de
  print(f"Ratio de cambio en la pérdida de m: {loss_rate_of_change_m}, multiplicado por el ratio de aprendizaje:
  m = m + learning_rate*loss_rate_of_change_m
  line_evolution.append(m)
```

```
print(line_evolution)
plt.figure(figsize=(8,6))
colors = plt.cm.winter(torch.linspace(0,1,steps+1))
for index, m in enumerate(line_evolution):
  plt.plot([0,data.max()], [0,m*data.max()], c=colors[index])
plt.scatter(data[:,0], data[:,1], marker='o', c='r', label="data")
plt.plot([0,data.max()], [0,true_m*data.max()], c='r', label="True line")
plt.legend()
plt.grid(True)
plt.show()
```

```
--------- step 0
m actual: 0.8
Dato escogido aleatoriamente (x_true,y_true): (tensor(30.6232), tensor(14.4707))
Valor de y predicho: 24.498546600341797
Valor de y predicho 'bajando la m': 21.436227798461914, error: 6.965547561645508
Valor de y predicho 'aumentando la m': 27.560863494873047, error: 13.09018325805664
Ratio de cambio en la pérdida de m: -30.623178482055664, multiplicado por el ratio de aprendizaje: -0.0306231807
9173565
--------- step 1
m actual: 0.769376814365387
Dato escogido aleatoriamente (x_true,y_true): (tensor(43.0127), tensor(20.6768))
Valor de y predicho: 33.09295654296875
Valor de y predicho 'bajando la m': 28.79168701171875, error: 8.1148662567138 67
Valor de y predicho 'aumentando la m': 37.39422607421875, error: 16.717405319213867
Ratio de cambio en la pérdida de m: -43.0126953125, multiplicado por el ratio de aprendizaje: -0.043012697249650
955
--------- step 2
m actual: 0.7263641357421875
Dato escogido aleatoriamente (x_true,y_true): (tensor(39.1849), tensor(13.5660))
Valor de y predicho: 28.462520599365234
Valor de y predicho 'bajando la m': 24.544029235839844, error: 10.977991104125977
Valor de y predicho 'aumentando la m': 32.38101577758789, error: 18.814977645874023
Ratio de cambio en la pérdida de m: -39.184932708740234, multiplicado por el ratio de aprendizaje: -0.0391849353
9094925
--------- step 3
m actual: 0.6871792078018188
Dato escogido aleatoriamente (x_true,y_true): (tensor(31.9664), tensor(15.2168))
Valor de y predicho: 21.96661949157715
Valor de y predicho 'bajando la m': 18.769983291625977, error: 3.5531702041625977
Valor de y predicho 'aumentando la m': 25.16325569152832, error: 9.946442604064941
Ratio de cambio en la pérdida de m: -31.96636199951172, multiplicado por el ratio de aprendizaje: -0.03196636214
852333
--------- step 4
m actual: 0.6552128195762634
Dato escogido aleatoriamente (x_true,y_true): (tensor(13.5681), tensor(9.7940))
Valor de y predicho: 8.88999080657959
Valor de y predicho 'bajando la m': 7.5331807136535645, error: 2.260861873626709
Valor de y predicho 'aumentando la m': 10.246800422668457, error: 0.4527578353881836
Ratio de cambio en la pérdida de m: 9.040519714355469, multiplicado por el ratio de aprendizaje: 0.0090405205264
68754
--------- step 5
m actual: 0.6642533540725708
Dato escogido aleatoriamente (x_true,y_true): (tensor(19.6427), tensor(6.6395))
Valor de y predicho: 13.04770278930664
Valor de y predicho 'bajando la m': 11.083436012268066, error: 4.443950653076172
Valor de y predicho 'aumentando la m': 15.011969566345215, error: 8.37248420715332
Ratio de cambio en la pérdida de m: -19.642667770385742, multiplicado por el ratio de aprendizaje: -0.0196426678
44891548
--------- step 6
m actual: 0.6446107029914856
Dato escogido aleatoriamente (x_true,y_true): (tensor(44.5825), tensor(19.8671))
Valor de y predicho: 28.73836898803711
Valor de y predicho 'bajando la m': 24.28011703491211, error: 4.412996292114258
Valor de y predicho 'aumentando la m': 33.19662094116211, error: 13.329500198364258
Ratio de cambio en la pérdida de m: -44.58251953125, multiplicado por el ratio de aprendizaje: -0.04458252340555
191
--------- step 7
m actual: 0.6000281572341919
Dato escogido aleatoriamente (x_true,y_true): (tensor(5.9533), tensor(5.3649))
Valor de y predicho: 3.572150468826294
Valor de y predicho 'bajando la m': 2.9768197536468506, error: 2.3881027698516846
Valor de y predicho 'aumentando la m': 4.167480945587158, error: 1.197441577911377
Ratio de cambio en la pérdida de m: 5.953305721282959, multiplicado por el ratio de aprendizaje: 0.0059533058665
69281
--------- step 8
m actual: 0.6059814691543579
Dato escogido aleatoriamente (x_true,y_true): (tensor(41.3869), tensor(20.8815))
Valor de y predicho: 25.079679489135742
Valor de y predicho 'bajando la m': 20.940990447998047, error: 0.059459686279296875
Valor de y predicho 'aumentando la m': 29.218368530273438, error: 8.336837768554688
Ratio de cambio en la pérdida de m: -41.38689041137695, multiplicado por el ratio de aprendizaje: -0.04138689115
643501
--------- step 9
```

```
m actual: 0.564594566822052
Dato escogido aleatoriamente (x_true,y_true): (tensor(12.0565), tensor(2.0787))
Valor de y predicho: 6.807029724121094
Valor de y predicho 'bajando la m': 5.601380825042725, error: 3.5226950645446777
Valor de y predicho 'aumentando la m': 8.012679100036621, error: 5.933993339538574
Ratio de cambio en la pérdida de m: -12.056490898132324, multiplicado por el ratio de aprendizaje: -0.0120564913
3771658
--------- step 10
m actual: 0.5525380969047546
Dato escogido aleatoriamente (x_true,y_true): (tensor(26.0389), tensor(13.7948))
Valor de y predicho: 14.387463569641113
Valor de y predicho 'bajando la m': 11.783576965332031, error: 2.011202812194824
Valor de y predicho 'aumentando la m': 16.991350173950195, error: 3.19657039642334
Ratio de cambio en la pérdida de m: -5.926837921142578, multiplicado por el ratio de aprendizaje: -0.00592683814
4659996
--------- step 11
m actual: 0.5466112494468689
Dato escogido aleatoriamente (x_true,y_true): (tensor(24.9908), tensor(13.6071))
Valor de y predicho: 13.660249710083008
Valor de y predicho 'bajando la m': 11.16117000579834, error: 2.445969581604004
Valor de y predicho 'aumentando la m': 16.159330368041992, error: 2.5521907806396484
Ratio de cambio en la pérdida de m: -0.5311059951782227, multiplicado por el ratio de aprendizaje: -0.0005311060
231178999
--------- step 12
m actual: 0.5460801720619202
Dato escogido aleatoriamente (x_true,y_true): (tensor(15.6467), tensor(7.9297))
Valor de y predicho: 8.544330596923828
Valor de y predicho 'bajando la m': 6.9796648025512695, error: 0.9500641822814941
Valor de y predicho 'aumentando la m': 10.108997344970703, error: 2.1792683601379395
Ratio de cambio en la pérdida de m: -6.146020889282227, multiplicado por el ratio de aprendizaje: -0.00614602118
73054504
--------- step 13
m actual: 0.5399341583251953
Dato escogido aleatoriamente (x_true,y_true): (tensor(41.5998), tensor(22.0618))
Valor de y predicho: 22.46116065979004
Valor de y predicho 'bajando la m': 18.301179885864258, error: 3.7606449127197266
Valor de y predicho 'aumentando la m': 26.621143341064453, error: 4.559318542480469
Ratio de cambio en la pérdida de m: -3.993368148803711, multiplicado por el ratio de aprendizaje: -0.00399336824
1935968
--------- step 14
m actual: 0.5359407663345337
Dato escogido aleatoriamente (x_true,y_true): (tensor(43.6831), tensor(23.9597))
Valor de y predicho: 23.411537170410156
Valor de y predicho 'bajando la m': 19.043230056762695, error: 4.9164581298828125
Valor de y predicho 'aumentando la m': 27.779844284057617, error: 3.8201560974121094
Ratio de cambio en la pérdida de m: 5.481510162353516, multiplicado por el ratio de aprendizaje: 0.0054815104231
238365
--------- step 15
m actual: 0.5414222478866577
Dato escogido aleatoriamente (x_true,y_true): (tensor(5.9533), tensor(5.3649))
Valor de y predicho: 3.2232515811920166
Valor de y predicho 'bajando la m': 2.6279211044311523, error: 2.737001419067383
Valor de y predicho 'aumentando la m': 3.81858229637146, error: 1.5463402271270752
Ratio de cambio en la pérdida de m: 5.953305721282959, multiplicado por el ratio de aprendizaje: 0.0059533058665
69281
--------- step 16
m actual: 0.5473755598068237
Dato escogido aleatoriamente (x_true,y_true): (tensor(4.9472), tensor(2.4668))
Valor de y predicho: 2.707984447479248
Valor de y predicho 'bajando la m': 2.2132630348205566, error: 0.2534923553466797
Valor de y predicho 'aumentando la m': 3.2027058601379395, error: 0.7359504699707031
Ratio de cambio en la pérdida de m: -2.412290573120117, multiplicado por el ratio de aprendizaje: -0.00241229077
80110836
--------- step 17
m actual: 0.5449632406234741
Dato escogido aleatoriamente (x_true,y_true): (tensor(28.4499), tensor(10.2899))
Valor de y predicho: 15.504166603088379
Valor de y predicho 'bajando la m': 12.659173011779785, error: 2.3692378997802734
Valor de y predicho 'aumentando la m': 18.349159240722656, error: 8.059224128723145
Ratio de cambio en la pérdida de m: -28.44993019104004, multiplicado por el ratio de aprendizaje: -0.02844993211
3289833
--------- step 18
m actual: 0.5165132880210876
Dato escogido aleatoriamente (x_true,y_true): (tensor(28.4499), tensor(10.2899))
Valor de y predicho: 14.694766998291016
Valor de y predicho 'bajando la m': 11.849774360656738, error: 1.5598392486572266
Valor de y predicho 'aumentando la m': 17.53976058959961, error: 7.249825477600098
Ratio de cambio en la pérdida de m: -28.44993019104004, multiplicado por el ratio de aprendizaje: -0.02844993211
3289833
--------- step 19
m actual: 0.48806336522102356
Dato escogido aleatoriamente (x_true,y_true): (tensor(2.6297), tensor(4.7310))
Valor de y predicho: 1.2834376096725464
```

```
Valor de y predicho 'bajando la m': 1.0204722881317139, error: 3.7105319499969482
Valor de y predicho 'aumentando la m': 1.546402931213379, error: 3.184601306915283
Ratio de cambio en la pérdida de m: 2.629653215408325, multiplicado por el ratio de aprendizaje: 0.0026296533178
538084
[0.8, tensor(0.7694), tensor(0.7264), tensor(0.6872), tensor(0.6552), tensor(0.6643), tensor(0.6446), tensor(0.6
000), tensor(0.6060), tensor(0.5646), tensor(0.5525), tensor(0.5466), tensor(0.5461), tensor(0.5399), tensor(0.5
359), tensor(0.5414), tensor(0.5474), tensor(0.5450), tensor(0.5165), tensor(0.4881), tensor(0.4907)]
```



En la gráfica podemos ver cómo la evolución de la recta (las azules son los primeros pasos y las verdes los últimos) se acerca a la recta que usamos para generar los datos (rojo). Si analizamos la evolución numérica podemos ver que es algo irregular. A veces los ratios de cambio están en el orden de varias decenas y otras veces están en el orden de las décimas. A veces son positivos y otras veces negativos. Esto puede ocurrir, aunque es indeseable que un entrenamiento sea tan *inestable*. En este caso uno de los principales sospechosos de que ocurra es que los datos tienen ruido. Muy pocos de los datos que tenemos se ajustan bien a la recta con la que fueron generados. Como cogemos un dato de referencia cada vez, este dato puede quedar más o menos lejos de la recta que representa nuestro modelo en cada paso. ¡Puede incluso hacer que nuestro modelo se ajuste en la dirección equivocada!

Por suerte, eso tiene fácil solución. No cojamos un único dato. Vamos a usarlos todos en cada paso de entrenamiento. Para eso primero hay que hacerle unos pequeños ajustes a nuestras funciones de error.

In [9]:
```python
def MAE_loss(y_pred, y_true):
  """
  Función para calcular la media del error absoluto.
  """
  absolute_error = torch.abs(y_pred-y_true)
  return absolute_error.mean()

def MSE_loss(y_pred, y_true):
  """
  Función para calcular la media del error cuadrático.
  """
  squared_error = (y_pred-y_true)**2
  return squared_error.mean()
```

Una vez nuestras funciones de error obtienen valores medios, podemos ajustar el proceso de entrenamiento del modelo para que trabaje con todos los datos siempre.

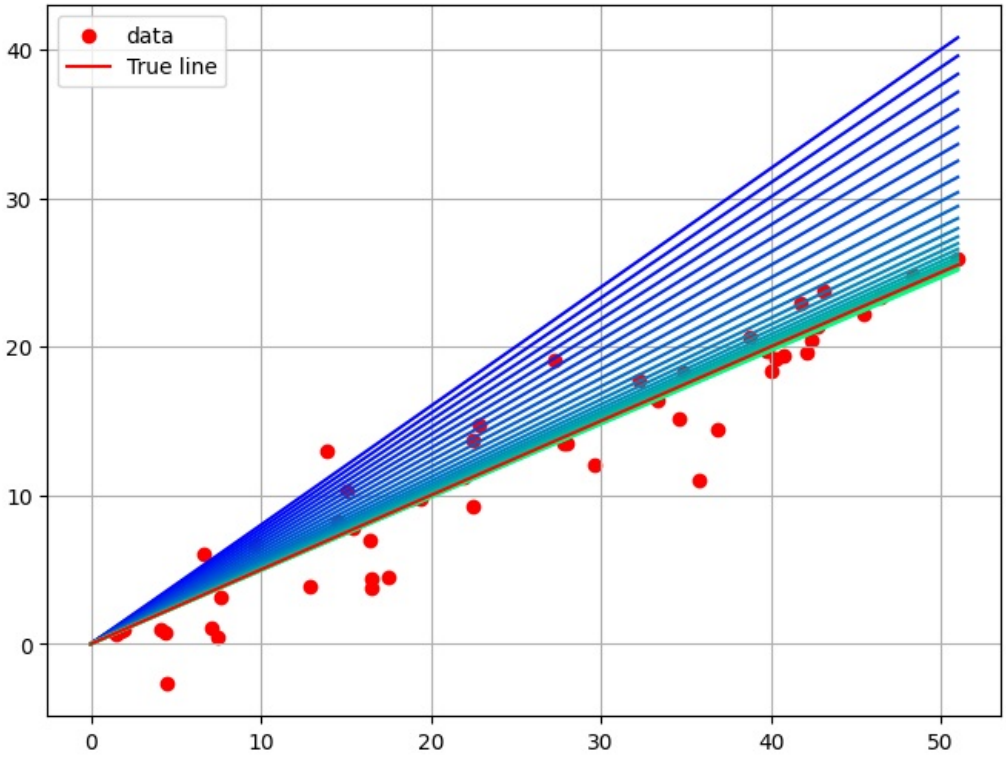
In [18]:
```python
learning_rate = 1e-3
delta = 0.1
steps = 25
loss_fn = MAE_loss

true_m, true_b = [0.5,0]
data = get_data_from_a_line(true_m, true_b, 50)
m = 0.8                                          # Inicializamos la m a 0.8 con propósitos meramente didácticos.
b = 0
line_evolution = [m]

for step in range(steps):
  x_true = data[:,0]
  y_true = data[:,1]
```

```
  loss_rate_of_change_m = (loss_fn(line_model(m-delta,b,x_true), y_true) - loss_fn(line_model(m+delta,b,x_true),
  print(f"--------- step {step}")
  print(f"m actual: {m}")
  print(f"Valor medio de y: {y_true.mean()}")
  print(f"Valor medio de y predicho: {line_model(m,b,x_true).mean()}")
  print(f"Valor medio de y predicho 'bajando la m': {line_model(m-delta,b,x_true).mean()}, error: {loss_fn(line_
  print(f"Valor medio de y predicho 'aumentando la m': {line_model(m+delta,b,x_true).mean()}, error: {loss_fn(li
  print(f"Ratio de cambio en la pérdida de m: {loss_rate_of_change_m}, multiplicado por el ratio de aprendizaje:
  m = m + learning_rate*loss_rate_of_change_m
  line_evolution.append(m)

plt.figure(figsize=(8,6))
colors = plt.cm.winter(torch.linspace(0,1,steps+1))
for index, m in enumerate(line_evolution):
  plt.plot([0,data.max()], [0,m*data.max()], c=colors[index])
plt.scatter(data[:,0], data[:,1], marker='o', c='r', label="data")
plt.plot([0,data.max()], [0,true_m*data.max()], c='r', label="True line")
plt.legend()
plt.grid(True)
plt.show()
```

```
--------- step 0
m actual: 0.8
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 19.94540786743164
Valor medio de y predicho 'bajando la m': 17.452232360839844, error: 5.587164402008057
Valor medio de y predicho 'aumentando la m': 22.438583374023438, error: 10.409149169921875
Ratio de cambio en la pérdida de m: -24.10992431640625, multiplicado por el ratio de aprendizaje: -0.02410992607
474327
--------- step 1
m actual: 0.7758901119232178
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 19.34430503845215
Valor medio de y predicho 'bajando la m': 16.85112953186035, error: 5.039474964141846
Valor medio de y predicho 'aumentando la m': 21.837482452392578, error: 9.827775001525879
Ratio de cambio en la pérdida de m: -23.941499710083008, multiplicado por el ratio de aprendizaje: -0.0239415001
1241436
--------- step 2
m actual: 0.7519485950469971
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 18.747400283813477
Valor medio de y predicho 'bajando la m': 16.254226684570312, error: 4.511997222900391
Valor medio de y predicho 'aumentando la m': 21.240575790405273, error: 9.250459671020508
Ratio de cambio en la pérdida de m: -23.692312240600586, multiplicado por el ratio de aprendizaje: -0.0236923135
81705093
--------- step 3
m actual: 0.7282562851905823
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 18.156713485717773
Valor medio de y predicho 'bajando la m': 15.663535118103027, error: 4.003662586212158
Valor medio de y predicho 'aumentando la m': 20.649885177612305, error: 8.679153442382812
Ratio de cambio en la pérdida de m: -23.37745475769043, multiplicado por el ratio de aprendizaje: -0.02337745577
096939
--------- step 4
m actual: 0.7048788070678711
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 17.573869705200195
Valor medio de y predicho 'bajando la m': 15.080693244934082, error: 3.515810251235962
Valor medio de y predicho 'aumentando la m': 20.067045211791992, error: 8.115439414978027
Ratio de cambio en la pérdida de m: -22.998146057128906, multiplicado por el ratio de aprendizaje: -0.0229981467
1278
--------- step 5
m actual: 0.6818806529045105
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 17.0004825592041
Valor medio de y predicho 'bajando la m': 14.507308006286621, error: 3.0508008003234863
Valor medio de y predicho 'aumentando la m': 19.493661880493164, error: 7.560871601104736
Ratio de cambio en la pérdida de m: -22.55035400390625, multiplicado por el ratio de aprendizaje: -0.02255035564
303398
--------- step 6
m actual: 0.6593303084373474
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 16.438264846801758
Valor medio de y predicho 'bajando la m': 13.945087432861328, error: 2.5966920852661133
Valor medio de y predicho 'aumentando la m': 18.931440353393555, error: 7.017102241516113
Ratio de cambio en la pérdida de m: -22.10205078125, multiplicado por el ratio de aprendizaje: -0.02210205234587
1925
--------- step 7
m actual: 0.63722825050354
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 15.887221336364746
Valor medio de y predicho 'bajando la m': 13.394043922424316, error: 2.2248544692993164
Valor medio de y predicho 'aumentando la m': 18.380399703979492, error: 6.484143257141113
```

```
Ratio de cambio en la pérdida de m: -21.296443939208984, multiplicado por el ratio de aprendizaje: -0.021296445280313492
--------- step 8
m actual: 0.6159318089485168
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 15.3562650680542
Valor medio de y predicho 'bajando la m': 12.863086700439453, error: 1.9445773363113403
Valor medio de y predicho 'aumentando la m': 17.849441528320312, error: 5.97061014175415
Ratio de cambio en la pérdida de m: -20.130165100097656, multiplicado por el ratio de aprendizaje: -0.020130166678392887
--------- step 9
m actual: 0.5958016514778137
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 14.85438346862793
Valor medio de y predicho 'bajando la m': 12.36120891571045, error: 1.824526309967041
Valor medio de y predicho 'aumentando la m': 17.34756088256836, error: 5.490510940551758
Ratio de cambio en la pérdida de m: -18.329923629760742, multiplicado por el ratio de aprendizaje: -0.018329923977248745
--------- step 10
m actual: 0.5774717330932617
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 14.39738655090332
Valor medio de y predicho 'bajando la m': 11.904211044311523, error: 1.8876217603683472
Valor medio de y predicho 'aumentando la m': 16.890562057495117, error: 5.074321746826172
Ratio de cambio en la pérdida de m: -15.933499336242676, multiplicado por el ratio de aprendizaje: -0.015933500602841377
--------- step 11
m actual: 0.5615382194519043
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 14.000136375427246
Valor medio de y predicho 'bajando la m': 11.50696086883545, error: 2.0608811378479004
Valor medio de y predicho 'aumentando la m': 16.49331283569336, error: 4.723276138305664
Ratio de cambio en la pérdida de m: -13.31197452545166, multiplicado por el ratio de aprendizaje: -0.013311974704265594
--------- step 12
m actual: 0.5482262372970581
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 13.668246269226074
Valor medio de y predicho 'bajando la m': 11.175067901611328, error: 2.241755485534668
Valor medio de y predicho 'aumentando la m': 16.161422729492188, error: 4.429986953735352
Ratio de cambio en la pérdida de m: -10.941157341003418, multiplicado por el ratio de aprendizaje: -0.010941158048808575
--------- step 13
m actual: 0.5372850894927979
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 13.395462989807129
Valor medio de y predicho 'bajando la m': 10.902286529541016, error: 2.39502215385437
Valor medio de y predicho 'aumentando la m': 15.888640403747559, error: 4.194331645965576
Ratio de cambio en la pérdida de m: -8.99654769897461, multiplicado por el ratio de aprendizaje: -0.008996548131108284
--------- step 14
m actual: 0.5282885432243347
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 13.171163558959961
Valor medio de y predicho 'bajando la m': 10.677986145019531, error: 2.5327131748199463
Valor medio de y predicho 'aumentando la m': 15.66434097290039, error: 4.004344940185547
Ratio de cambio en la pérdida de m: -7.358158588409424, multiplicado por el ratio de aprendizaje: -0.007358158938586712
--------- step 15
m actual: 0.5209304094314575
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.987710952758789
Valor medio de y predicho 'bajando la m': 10.494536399841309, error: 2.6467342376708984
Valor medio de y predicho 'aumentando la m': 15.480888366699219, error: 3.848958730697632
Ratio de cambio en la pérdida de m: -6.011122226715088, multiplicado por el ratio de aprendizaje: -0.006011122372001 4095
--------- step 16
m actual: 0.5149192810058594
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.837843894958496
Valor medio de y predicho 'bajando la m': 10.3446683883667, error: 2.7428925037384033
Valor medio de y predicho 'aumentando la m': 15.33102035522461, error: 3.722017526626587
Ratio de cambio en la pérdida de m: -4.895625114440918, multiplicado por el ratio de aprendizaje: -0.004895625170320 3202724
--------- step 17
m actual: 0.5100236535072327
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.71578598022461
Valor medio de y predicho 'bajando la m': 10.222611427307129, error: 2.821928024291992
Valor medio de y predicho 'aumentando la m': 15.208963394165039, error: 3.6198372840881348
Ratio de cambio en la pérdida de m: -3.989546298980713, multiplicado por el ratio de aprendizaje: -0.003989546559751034
--------- step 18
```

```
m actual: 0.5060341358184814
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.616321563720703
Valor medio de y predicho 'bajando la m': 10.123144149780273, error: 2.8898260593414307
Valor medio de y predicho 'aumentando la m': 15.109498023986816, error: 3.5391714572906494
Ratio de cambio en la pérdida de m: -3.2467269897460938, multiplicado por el ratio de aprendizaje: -0.0032467271
66697383
--------- step 19
m actual: 0.5027874112129211
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.535374641418457
Valor medio de y predicho 'bajando la m': 10.04219913482666, error: 2.948152542114258
Valor medio de y predicho 'aumentando la m': 15.028552055358887, error: 3.473524570465088
Ratio de cambio en la pérdida de m: -2.6268601417541504, multiplicado por el ratio de aprendizaje: -0.0026268602
814525366
--------- step 20
m actual: 0.5001605749130249
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.46988296508789
Valor medio de y predicho 'bajando la m': 9.97670841217041, error: 2.9964513778686523
Valor medio de y predicho 'aumentando la m': 14.963059425354004, error: 3.4204113483428955
Ratio de cambio en la pérdida de m: -2.119799852371216, multiplicado por el ratio de aprendizaje: -0.00211979984
305799
--------- step 21
m actual: 0.4980407655239105
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.417032241821289
Valor medio de y predicho 'bajando la m': 9.923856735229492, error: 3.035428047180176
Valor medio de y predicho 'aumentando la m': 14.910208702087402, error: 3.377549409866333
Ratio de cambio en la pérdida de m: -1.7106068134307861, multiplicado por el ratio de aprendizaje: -0.0017106068
55340302
--------- step 22
m actual: 0.4963301718235016
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.374384880065918
Valor medio de y predicho 'bajando la m': 9.881209373474121, error: 3.066880464553833
Valor medio de y predicho 'aumentando la m': 14.867562294006348, error: 3.3429620265960693
Ratio de cambio en la pérdida de m: -1.3804078102111816, multiplicado por el ratio de aprendizaje: -0.0013804078
33494246
--------- step 23
m actual: 0.4949497580528259
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.33996868133545
Valor medio de y predicho 'bajando la m': 9.846793174743652, error: 3.092261552810669
Valor medio de y predicho 'aumentando la m': 14.833144187927246, error: 3.315051794052124
Ratio de cambio en la pérdida de m: -1.1139512062072754, multiplicado por el ratio de aprendizaje: -0.0011139512
062072754
--------- step 24
m actual: 0.49383580684661865
Valor medio de y: 12.053668975830078
Valor medio de y predicho: 12.312195777893066
Valor medio de y predicho 'bajando la m': 9.819019317626953, error: 3.112743616104126
Valor medio de y predicho 'aumentando la m': 14.805373191833496, error: 3.2925281524658203
Ratio de cambio en la pérdida de m: -0.8989226818084717, multiplicado por el ratio de aprendizaje: -0.0008989227
353595197
```



¡Lo que cambia una media! Ahora el ajuste es numéricamente estable, el ratio de cambio va disminuyendo a cada paso de entrenamiento y siempre avanza en la misma dirección. Como los datos se han generado con ruido gaussiano, los errores de unos y los de otros se compensan, así que la media es mucho más fiable para hacer el ajuste. Sin embargo, si nos fijamos, la recta no se está ajustando a m=0.5, sino a un valor ligeramente inferior. Esto seguramente se deba a que los datos se ajustan mejor a esa recta con una inclinación ligeramente inferior. Cuantos más datos generemos a partir de nuestra recta, mejor se ajustarán en general a ella. Un ajuste perfecto requeriría una cantidad infinita de datos.

Este proceso de entrenamiento tal como está tiene diversos problemas:

1. Sirve para un parámetro, pero hacerlo con muchos sería muy ineficiente.
2. El delta es una elección arbitraria cuyo valor va a determinar la velocidad de la convergencia y el margen de error de esta. Pisa un poco la función del ratio de aprendizaje. ¿Necesitamos los dos?
3. ¡No saca partido de la GPU!

En la siguiente sesión vamos a refinarlo :)

Recapitulación:

- Hemos creado un sistema de aprendizaje iterativo muy sencillo para ajustar una recta a unos datos.
- Hemos observado la necesidad de un modelo que ajustar.

- Hemos visto la necesidad de una función de error que debe elegir el diseñador del sistema.
- Hemos visto que la velocidad de convergencia se puede controlar con algunos parámetros que controlan cuánto actualizamos los parámetros.
- Hemos la diferencia entre hacer un paso de entrenamiento solo con un parámetro y hacerlo con la media de todos.