

Hello, World of Machine Learning

1. Sebelum memulai

Pada Jupyter Notebook ini, kamu akan mempelajari dasar "Hello, World" pada machine learning, dimana alih-alih kamu memprogram secara eksplisit aturan-aturan pada suatu bahasa pemrograman, seperti C++ atau Java, kamu akan membangun sistem yang dilatih menggunakan data untuk memprediksi aturan-aturan yang menggambarkan keterkaitan antara data.

Bayangkan masalah ini: Kamu membangun sistem fitness tracking yang bisa mengenali aktifitas-aktifitas olahraga. Kamu mungkin memiliki akses ke data kecepatan jalan seseorang dan mencoba untuk memprediksi aktifitas orang tersebut berdasarkan kecepatannya menggunakan kondisi.



```
if speed < 4:
    status = walking
```

Kamu selanjutnya bisa menambahkan kondisi untuk lari:



```
if speed < 4:
    status = WALKING
else:
    status = RUNNING
```

Kamu juga bisa menambahkan kondisi akhir untuk bersepeda:



```
if speed < 4:
    status = WALKING
if speed < 12:
    status = RUNNING
else:
    status = CYCLING
```

Sekarang, coba pertimbangkan apa yang akan terjadi selanjutnya jika kamu mau menambahkan suatu aktifitas

Sekarang, coba perumbangkan apa yang akan terjadi selanjutnya jika kamu mau menambahkan suatu aktivitas baru, misalnya golf. Tentu akan jauh lebih ambigu untuk menentukan aturan untuk aktivitas tersebut.



Selanjutnya gimana?

Sangatlah sulit untuk menulis program yang bisa mengenali aktivitas bermain golf, jadi apa yang harus kamu lakukan? Gunakan machine learning!

Prasyarat

Sebelum mencoba Jupyter Notebook ini, kamu perlu memiliki:

1. Pengetahuan yang solid tentang Python
2. Keterampilan pemrograman dasar

Yang akan kamu pelajari

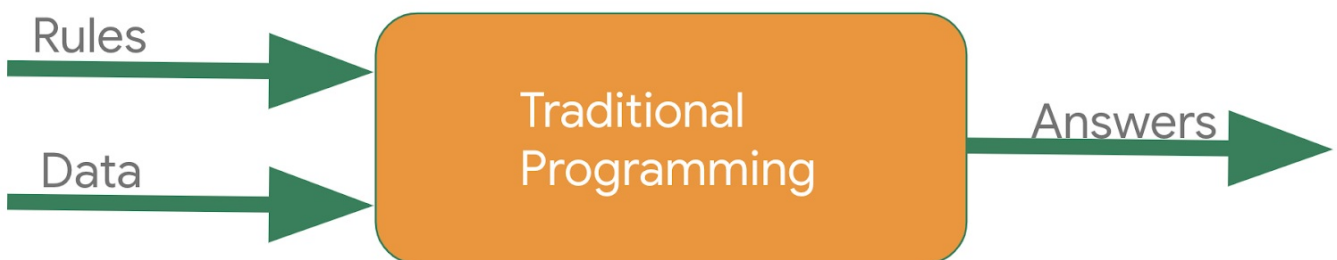
1. Dasar-dasar machine learning

Yang akan kamu buat

1. Model machine learning pertama kamu

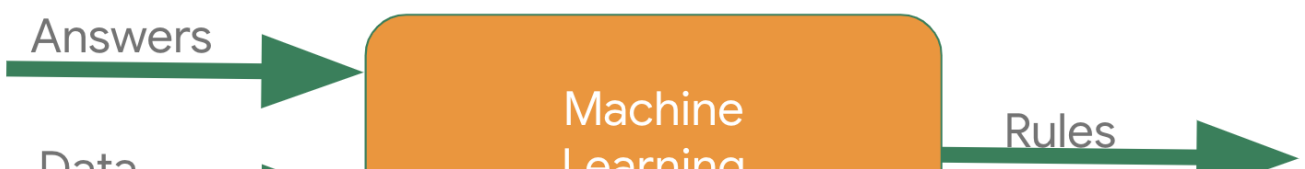
2. Apa itu Machine Learning?

Mari kita lihat cara tradisional membangun suatu aplikasi yang direpresentasikan oleh diagram di bawah:



Kamu mengekspresikan aturan-aturan menggunakan sebuah bahasa pemrograman. Aturan-aturan tersebut bereaksi terhadap data dan program kamu akan memberikan jawaban. Pada kasus deteksi aktivitas olahraga, aturan-aturan (kode yang kamu tulis untuk mendefinisikan tipe-tipe aktivitas) bereaksi terhadap data yang masuk (kecepatan gerak pengguna) untuk menghasilkan jawaban: yaitu output nilai dari fungsi untuk mendeteksi status aktivitas pengguna.

Proses mendeteksi status aktivitas menggunakan ML sebenarnya lumayan mirip, hanya input dan outputnya saja yang berbeda:





Daripada mencoba mendefinisikan aturan-aturan dan mengkespresikannya di dalam sebuah bahasa pemrograman, kamu memberikan jawaban-jawaban (biasanya disebut labels) bersamaan dengan data yang ada, dan selanjutnya mesin akan menyimpulkan aturan-aturan yang menentukan hubungan antara jawaban dan data. Sebagai contohnya, deteksi aktifitas olahraga mungkin akan terlihat seperti ini dalam konteks ML:



```

0101001010100101010
1001010101001011101
0100101010010101001
0101001010100101010
  
```

Label = WALKING

```

1010100101001010101
0101010010010010001
0010011111010101111
1010100100111101011
  
```

Label = RUNNING

```

1001010011111010101
1101010111010101110
1010101111010101011
111110001111010101
  
```

Label = BIKING

```

1111111111010011101
0011111010111110101
0101110101010101110
1010101010100111110
  
```

Label = GOLFING
(Sort of)

Kamu mengumpulkan data dan label yang sangat banyak sehingga bisa dengan efektif bilang, "Kalo jalan tuh gini loh," atau "Kalo lari tuh gini loh." Selanjutnya, dari dataset tersebut komputer bisa menyimpulkan aturan-aturan yang menentukan pola-pola yang menjelaskan aktifitas tertentu.

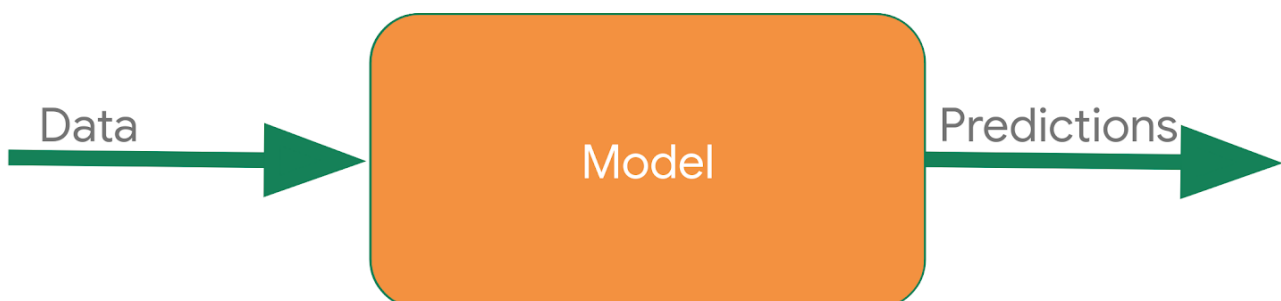
Bukan hanya menjadi metode alternatif dari pemrograman, metode ini juga memberikan kemampuan baru untuk skenario-skenario baru, misalnya menentukan pola-pola kegiatan bermain golf yang tidak mungkin dilakukan dengan pemrograman tradisional.

Dalam pemrograman tradisional, kode kamu terkompilasi menjadi sebuah binary yang biasanya disebut sebagai program. Pada ML, output yang kamu bangun dari data dan labels disebut **model**.

Jadi, jika kita kembali lagi ke diagram ini:



Output dari diagram flow di atas adalah model, dan kita bisa menggunakannya sebagai berikut:



Dimana kamu memberikan data sebagai input dan model menggunakan aturan-aturan yang disimpulkan dari proses pembelajaran mesin untuk menghasilkan prediction, misalnya, "Data ini terlihat seperti orang berjalan" atau "Data ini terlihat seperti orang bersepeda."

3. Membuat ML model pertama kamu

Perhatikan deretan-deretan angka di bawah. Apakah kam bisa melihat hubungan antara mereka?

X	Y
-1	-2
0	1
1	4
2	7
3	10
4	13

Kamu mungkin sadar bahwa nilai X bertambah 1 setiap barisnya dan nilai Y bertambah 3. Kamu mungkin berpikir bahwa Y sama dengan 3X ditambah atau dikurangi suatu angka. Selanjutnya kamu melihat ketika X=0 dan Y=1, kamu akan menyimpulkan bahwa $Y=3X+1$.

Yang baru saja kamu lakukan mirip persis dengan bagaimana kamu melatih ML model untuk melihat pola pada data!

Sekarang, ayoi kita lihat kode untuk melakukannya.

Bagaimana kamu melatih sebuah neural network untuk melakukan task serupa? Dengan menggunakan data! Kita harus memberikan data himpunan X dan Y kepada neural network sehingga ia mampu mengenali hubungan antara himpunan X dan Y.

Import

Mulai dengan meng-import library yang dibutuhkan. Kamu akan menggunakan TensorFlow dan memberi alias `tf` agar lebih mudah digunakan.

Selanjutnya, import `numpy` untuk merepresentasikan data sebagai lists secara mudah dan cepat.

Terakhir, kita akan menggunakan `keras`, sebuah framework untuk membuat neural network sebagai kumpulan layer-layer berurutan

In [1]:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
```

Menentukan dan mengkompilasi jaringan neural

Selanjutnya, kita akan membuat neural network sederhana. Neural networknya hanya memiliki satu layer, layer tersebut hanya memiliki satu neuron, dan input shape nya hanya satu.

In [2]:

```
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Model: "sequential_1"
```

Model: Sequential

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	2

Total params: 2 (8.00 B)

Trainable params: 2 (8.00 B)

Non-trainable params: 0 (0.00 B)

Selanjutnya, kita akan menulis kode untuk mengkompilasi neural network kita. Untuk melakukannya, kamu perlu membuat dua fungsi-- fungsi `loss` dan `optimizer`.

Pada contoh kali ini, kamu telah mengetahui bahwa hubungan antara angka-angka di atas adalah $Y=3X+1$.

Namun, ketika komputer mencoba untuk mempelajari hal ini, komputer akan mencoba membuat tebakan, bisa jadi tebakan pertamanya adalah $Y=10X+10$. Fungsi `loss` digunakan untuk mengukur jarak antara hasil perhitungan menggunakan fungsi tebakan dengan jawaban sesungguhnya, apakah bagus atau buruk.

Selanjutnya, model akan menggunakan fungsi `optimizer` untuk membuat tebakan selanjutnya. Berdasarkan hasil dari fungsi `loss`, fungsi `optimizer` akan mencoba meminimalisir nilai loss. Pada titik ini, komputer mungkin akan menebak menggunakan $Y=5X+5$. Walaupun tebakannya masih jelek, tapi komputer sudah mendekati ke jawaban yang benar (karena nilai loss nya mengecil).

Nah, model mengulangi hal di atas terus menerus sampai batas `epochs`, dimana akan kamu lihat sebentar lagi.

Pertama-tama, kita akan menggunakan fungsi `mean_squared_error` untuk fungsi loss dan stochastic gradient descent (`sgd`) untuk fungsi optimizer. Kamu belum perlu tahu rumus matematika dibalik layar fungsi-fungsi tersebut, tetapi kamu bisa melihat kalau mereka ampuh!

Seiring berjalannya waktu, kamu akan belajar berbagai macam fungsi-fungsi yang bisa ditentukan untuk `loss` dan `optimizer` di skenario-skenario berbeda.

In [3]:

```
model.compile(optimizer="sgd", loss="mean_squared_error")
```

Berikan data

Selanjutnya, kita akan memberikan data. Pada kasus kali ini, kita akan menggunakan enam angka X dan Y dari sebelumnya.

Kita akan menggunakan NumPy untuk membuat array:

In [4]:

```
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-2.0, 1.0, 4.0, 7.0, 10.0, 13.0], dtype=float)
```

Sekarang kamu sudah selesai menulis kode yang mendefinisikan sebuah neural network! Langkah selanjutnya adalah melakukan model training agar neural network kamu bisa menyimpulkan pola-pola antara angka-angka di atas dan menggunakannya untuk membuat model.


































4. Train the neural network

Proses training neural network untuk mempelajari hubungan antara nilai-nilai X dan Y dapat dimulai dengan memanggil fungsi `model.fit`. Menggunakan fungsi ini, neural network akan berulang kali melakukan tebakan, mengukur berapa bagus tebakannya (nilai loss), atau menggunakan optimizer untuk membuat tebakan lain. Neural network akan melakukan perulangan (looping) sesuai dengan jumlah `epochs` yang kamu tentukan.






































Ketika kamu menjalankan fungsi `model.fit` kamu akan melihat nilai loss pada setiap epoch.

In [5]:

```
model.fit(xs, ys, epochs=500)
```






































```
Epoch 1/500
1/1  1s 521ms/step - loss: 22.4574
Epoch 2/500
1/1  0s 112ms/step - loss: 17.6850
Epoch 3/500
1/1  0s 50ms/step - loss: 13.9300
Epoch 4/500
1/1  0s 61ms/step - loss: 10.9754
Epoch 5/500
1/1  0s 61ms/step - loss: 8.6506
Epoch 6/500
1/1  0s 65ms/step - loss: 6.8212
Epoch 7/500
1/1  0s 52ms/step - loss: 5.3815
Epoch 8/500
1/1  0s 59ms/step - loss: 4.2486
Epoch 9/500
1/1  0s 55ms/step - loss: 3.3570
Epoch 10/500
1/1  0s 63ms/step - loss: 2.6552
Epoch 11/500
1/1  0s 54ms/step - loss: 2.1028
Epoch 12/500
1/1  0s 60ms/step - loss: 1.6679
Epoch 13/500
1/1  0s 58ms/step - loss: 1.3255
Epoch 14/500
1/1  0s 56ms/step - loss: 1.0558
Epoch 15/500
1/1  0s 62ms/step - loss: 0.8433
Epoch 16/500
1/1  0s 46ms/step - loss: 0.6759
Epoch 17/500
1/1  0s 60ms/step - loss: 0.5440
Epoch 18/500
1/1  0s 56ms/step - loss: 0.4399
Epoch 19/500
1/1  0s 67ms/step - loss: 0.3578
Epoch 20/500
1/1  0s 133ms/step - loss: 0.2929
Epoch 21/500
1/1  0s 66ms/step - loss: 0.2417
Epoch 22/500
1/1  0s 60ms/step - loss: 0.2011
Epoch 23/500
1/1  0s 59ms/step - loss: 0.1690
Epoch 24/500
1/1  0s 60ms/step - loss: 0.1435
Epoch 25/500
1/1  0s 61ms/step - loss: 0.1232
Epoch 26/500
1/1  0s 51ms/step - loss: 0.1070
Epoch 27/500
1/1  0s 75ms/step - loss: 0.0941
Epoch 28/500
1/1  0s 55ms/step - loss: 0.0837
Epoch 29/500
1/1  0s 58ms/step - loss: 0.0754
Epoch 30/500
1/1  0s 82ms/step - loss: 0.0686
Epoch 31/500
1/1  0s 54ms/step - loss: 0.0631
Epoch 32/500
1/1  0s 56ms/step - loss: 0.0586
Epoch 33/500
1/1  0s 53ms/step - loss: 0.0548
```

```
1/1 0s 59ms/step - loss: 0.0516
Epoch 34/500
1/1 0s 57ms/step - loss: 0.0517
Epoch 35/500
1/1 0s 51ms/step - loss: 0.0491
Epoch 36/500
1/1 0s 62ms/step - loss: 0.0468
Epoch 37/500
1/1 0s 57ms/step - loss: 0.0449
Epoch 38/500
1/1 0s 60ms/step - loss: 0.0432
Epoch 39/500
1/1 0s 138ms/step - loss: 0.0417
Epoch 40/500
1/1 0s 62ms/step - loss: 0.0404
Epoch 41/500
1/1 0s 57ms/step - loss: 0.0392
Epoch 42/500
1/1 0s 50ms/step - loss: 0.0381
Epoch 43/500
1/1 0s 59ms/step - loss: 0.0370
Epoch 44/500
1/1 0s 53ms/step - loss: 0.0361
Epoch 45/500
1/1 0s 57ms/step - loss: 0.0352
Epoch 46/500
1/1 0s 62ms/step - loss: 0.0344
Epoch 47/500
1/1 0s 37ms/step - loss: 0.0336
Epoch 48/500
1/1 0s 59ms/step - loss: 0.0328
Epoch 49/500
1/1 0s 59ms/step - loss: 0.0321
Epoch 50/500
1/1 0s 39ms/step - loss: 0.0314
Epoch 51/500
1/1 0s 37ms/step - loss: 0.0307
Epoch 52/500
1/1 0s 37ms/step - loss: 0.0301
Epoch 53/500
1/1 0s 58ms/step - loss: 0.0294
Epoch 54/500
1/1 0s 57ms/step - loss: 0.0288
Epoch 55/500
1/1 0s 58ms/step - loss: 0.0282
Epoch 56/500
1/1 0s 39ms/step - loss: 0.0276
Epoch 57/500
1/1 0s 43ms/step - loss: 0.0270
Epoch 58/500
1/1 0s 40ms/step - loss: 0.0265
Epoch 59/500
1/1 0s 59ms/step - loss: 0.0259
Epoch 60/500
1/1 0s 38ms/step - loss: 0.0254
Epoch 61/500
1/1 0s 53ms/step - loss: 0.0249
Epoch 62/500
1/1 0s 65ms/step - loss: 0.0243
Epoch 63/500
1/1 0s 55ms/step - loss: 0.0238
Epoch 64/500
1/1 0s 62ms/step - loss: 0.0234
Epoch 65/500
1/1 0s 38ms/step - loss: 0.0229
Epoch 66/500
1/1 0s 41ms/step - loss: 0.0224
Epoch 67/500
1/1 0s 55ms/step - loss: 0.0219
Epoch 68/500
1/1 0s 39ms/step - loss: 0.0215
Epoch 69/500
1/1 0s 33ms/step - loss: 0.0210
```

```
1/1  0s 39ms/step - loss: 0.0210
Epoch 70/500
1/1  0s 39ms/step - loss: 0.0206
Epoch 71/500
1/1  0s 58ms/step - loss: 0.0202
Epoch 72/500
1/1  0s 55ms/step - loss: 0.0198
Epoch 73/500
1/1  0s 40ms/step - loss: 0.0194
Epoch 74/500
1/1  0s 29ms/step - loss: 0.0190
Epoch 75/500
1/1  0s 32ms/step - loss: 0.0186
Epoch 76/500
1/1  0s 29ms/step - loss: 0.0182
Epoch 77/500
1/1  0s 37ms/step - loss: 0.0178
Epoch 78/500
1/1  0s 56ms/step - loss: 0.0175
Epoch 79/500
1/1  0s 58ms/step - loss: 0.0171
Epoch 80/500
1/1  0s 32ms/step - loss: 0.0168
Epoch 81/500
1/1  0s 32ms/step - loss: 0.0164
Epoch 82/500
1/1  0s 63ms/step - loss: 0.0161
Epoch 83/500
1/1  0s 33ms/step - loss: 0.0157
Epoch 84/500
1/1  0s 43ms/step - loss: 0.0154
Epoch 85/500
1/1  0s 64ms/step - loss: 0.0151
Epoch 86/500
1/1  0s 36ms/step - loss: 0.0148
Epoch 87/500
1/1  0s 59ms/step - loss: 0.0145
Epoch 88/500
1/1  0s 35ms/step - loss: 0.0142
Epoch 89/500
1/1  0s 61ms/step - loss: 0.0139
Epoch 90/500
1/1  0s 54ms/step - loss: 0.0136
Epoch 91/500
1/1  0s 40ms/step - loss: 0.0133
Epoch 92/500
1/1  0s 41ms/step - loss: 0.0131
Epoch 93/500
1/1  0s 41ms/step - loss: 0.0128
Epoch 94/500
1/1  0s 57ms/step - loss: 0.0125
Epoch 95/500
1/1  0s 31ms/step - loss: 0.0123
Epoch 96/500
1/1  0s 34ms/step - loss: 0.0120
Epoch 97/500
1/1  0s 33ms/step - loss: 0.0118
Epoch 98/500
1/1  0s 58ms/step - loss: 0.0115
Epoch 99/500
1/1  0s 53ms/step - loss: 0.0113
Epoch 100/500
1/1  0s 56ms/step - loss: 0.0111
Epoch 101/500
1/1  0s 56ms/step - loss: 0.0108
Epoch 102/500
1/1  0s 56ms/step - loss: 0.0106
Epoch 103/500
1/1  0s 39ms/step - loss: 0.0104
Epoch 104/500
1/1  0s 58ms/step - loss: 0.0102
Epoch 105/500
1/1  0s 50ms/step - loss: 0.0100
```







































1/1	0s	39ms/step	loss: 0.0100
Epoch 106/500			
1/1	0s	58ms/step	loss: 0.0098
Epoch 107/500			
1/1	0s	39ms/step	loss: 0.0096
Epoch 108/500			
1/1	0s	55ms/step	loss: 0.0094
Epoch 109/500			
1/1	0s	60ms/step	loss: 0.0092
Epoch 110/500			
1/1	0s	40ms/step	loss: 0.0090
Epoch 111/500			
1/1	0s	44ms/step	loss: 0.0088
Epoch 112/500			
1/1	0s	55ms/step	loss: 0.0086
Epoch 113/500			
1/1	0s	39ms/step	loss: 0.0084
Epoch 114/500			
1/1	0s	58ms/step	loss: 0.0083
Epoch 115/500			
1/1	0s	35ms/step	loss: 0.0081
Epoch 116/500			
1/1	0s	40ms/step	loss: 0.0079
Epoch 117/500			
1/1	0s	55ms/step	loss: 0.0078
Epoch 118/500			
1/1	0s	37ms/step	loss: 0.0076
Epoch 119/500			
1/1	0s	63ms/step	loss: 0.0075
Epoch 120/500			
1/1	0s	60ms/step	loss: 0.0073
Epoch 121/500			
1/1	0s	60ms/step	loss: 0.0072
Epoch 122/500			
1/1	0s	45ms/step	loss: 0.0070
Epoch 123/500			
1/1	0s	49ms/step	loss: 0.0069
Epoch 124/500			
1/1	0s	52ms/step	loss: 0.0067
Epoch 125/500			
1/1	0s	43ms/step	loss: 0.0066
Epoch 126/500			
1/1	0s	40ms/step	loss: 0.0064
Epoch 127/500			
1/1	0s	59ms/step	loss: 0.0063
Epoch 128/500			
1/1	0s	35ms/step	loss: 0.0062
Epoch 129/500			
1/1	0s	57ms/step	loss: 0.0061
Epoch 130/500			
1/1	0s	41ms/step	loss: 0.0059
Epoch 131/500			
1/1	0s	33ms/step	loss: 0.0058
Epoch 132/500			
1/1	0s	58ms/step	loss: 0.0057
Epoch 133/500			
1/1	0s	35ms/step	loss: 0.0056
Epoch 134/500			
1/1	0s	61ms/step	loss: 0.0055
Epoch 135/500			
1/1	0s	54ms/step	loss: 0.0053
Epoch 136/500			
1/1	0s	40ms/step	loss: 0.0052
Epoch 137/500			
1/1	0s	63ms/step	loss: 0.0051
Epoch 138/500			
1/1	0s	51ms/step	loss: 0.0050
Epoch 139/500			
1/1	0s	41ms/step	loss: 0.0049
Epoch 140/500			
1/1	0s	56ms/step	loss: 0.0048
Epoch 141/500			
1/1	0s	56ms/step	loss: 0.0047

```
1/1 Epoch 142/500 0s 35ms/step - loss: 0.0046
1/1 Epoch 143/500 0s 62ms/step - loss: 0.0045
1/1 Epoch 144/500 0s 52ms/step - loss: 0.0044
1/1 Epoch 145/500 0s 49ms/step - loss: 0.0043
1/1 Epoch 146/500 0s 46ms/step - loss: 0.0043
1/1 Epoch 147/500 0s 51ms/step - loss: 0.0042
1/1 Epoch 148/500 0s 30ms/step - loss: 0.0041
1/1 Epoch 149/500 0s 57ms/step - loss: 0.0040
1/1 Epoch 150/500 0s 56ms/step - loss: 0.0039
1/1 Epoch 151/500 0s 32ms/step - loss: 0.0038
1/1 Epoch 152/500 0s 57ms/step - loss: 0.0038
1/1 Epoch 153/500 0s 58ms/step - loss: 0.0037
1/1 Epoch 154/500 0s 59ms/step - loss: 0.0036
1/1 Epoch 155/500 0s 40ms/step - loss: 0.0035
1/1 Epoch 156/500 0s 46ms/step - loss: 0.0035
1/1 Epoch 157/500 0s 58ms/step - loss: 0.0034
1/1 Epoch 158/500 0s 38ms/step - loss: 0.0033
1/1 Epoch 159/500 0s 29ms/step - loss: 0.0033
1/1 Epoch 160/500 0s 56ms/step - loss: 0.0032
1/1 Epoch 161/500 0s 38ms/step - loss: 0.0031
1/1 Epoch 162/500 0s 29ms/step - loss: 0.0031
1/1 Epoch 163/500 0s 56ms/step - loss: 0.0030
1/1 Epoch 164/500 0s 26ms/step - loss: 0.0029
1/1 Epoch 165/500 0s 30ms/step - loss: 0.0029
1/1 Epoch 166/500 0s 32ms/step - loss: 0.0028
1/1 Epoch 167/500 0s 33ms/step - loss: 0.0028
1/1 Epoch 168/500 0s 56ms/step - loss: 0.0027
1/1 Epoch 169/500 0s 29ms/step - loss: 0.0026
1/1 Epoch 170/500 0s 29ms/step - loss: 0.0026
1/1 Epoch 171/500 0s 28ms/step - loss: 0.0025
1/1 Epoch 172/500 0s 32ms/step - loss: 0.0025
1/1 Epoch 173/500 0s 28ms/step - loss: 0.0024
1/1 Epoch 174/500 0s 35ms/step - loss: 0.0024
1/1 Epoch 175/500 0s 58ms/step - loss: 0.0023
1/1 Epoch 176/500 0s 33ms/step - loss: 0.0023
1/1 Epoch 177/500 0s 33ms/step - loss: 0.0022
```






































```
1/1  0s 33ms/step - loss: 0.0022
Epoch 178/500
1/1  0s 32ms/step - loss: 0.0022
Epoch 179/500
1/1  0s 54ms/step - loss: 0.0021
Epoch 180/500
1/1  0s 59ms/step - loss: 0.0021
Epoch 181/500
1/1  0s 33ms/step - loss: 0.0021
Epoch 182/500
1/1  0s 34ms/step - loss: 0.0020
Epoch 183/500
1/1  0s 27ms/step - loss: 0.0020
Epoch 184/500
1/1  0s 29ms/step - loss: 0.0019
Epoch 185/500
1/1  0s 31ms/step - loss: 0.0019
Epoch 186/500
1/1  0s 30ms/step - loss: 0.0019
Epoch 187/500
1/1  0s 32ms/step - loss: 0.0018
Epoch 188/500
1/1  0s 38ms/step - loss: 0.0018
Epoch 189/500
1/1  0s 57ms/step - loss: 0.0017
Epoch 190/500
1/1  0s 29ms/step - loss: 0.0017
Epoch 191/500
1/1  0s 31ms/step - loss: 0.0017
Epoch 192/500
1/1  0s 30ms/step - loss: 0.0016
Epoch 193/500
1/1  0s 58ms/step - loss: 0.0016
Epoch 194/500
1/1  0s 57ms/step - loss: 0.0016
Epoch 195/500
1/1  0s 28ms/step - loss: 0.0015
Epoch 196/500
1/1  0s 57ms/step - loss: 0.0015
Epoch 197/500
1/1  0s 28ms/step - loss: 0.0015
Epoch 198/500
1/1  0s 28ms/step - loss: 0.0014
Epoch 199/500
1/1  0s 32ms/step - loss: 0.0014
Epoch 200/500
1/1  0s 29ms/step - loss: 0.0014
Epoch 201/500
1/1  0s 59ms/step - loss: 0.0014
Epoch 202/500
1/1  0s 29ms/step - loss: 0.0013
Epoch 203/500
1/1  0s 58ms/step - loss: 0.0013
Epoch 204/500
1/1  0s 29ms/step - loss: 0.0013
Epoch 205/500
1/1  0s 55ms/step - loss: 0.0013
Epoch 206/500
1/1  0s 30ms/step - loss: 0.0012
Epoch 207/500
1/1  0s 58ms/step - loss: 0.0012
Epoch 208/500
1/1  0s 31ms/step - loss: 0.0012
Epoch 209/500
1/1  0s 58ms/step - loss: 0.0012
Epoch 210/500
1/1  0s 61ms/step - loss: 0.0011
Epoch 211/500
1/1  0s 38ms/step - loss: 0.0011
Epoch 212/500
1/1  0s 55ms/step - loss: 0.0011
Epoch 213/500
1/1  0s 30ms/step - loss: 0.0011
```

```
1/1 Epoch 214/500 0s 30ms/step - loss: 9.0011e-04
1/1 Epoch 215/500 0s 28ms/step - loss: 0.0010
1/1 Epoch 216/500 0s 31ms/step - loss: 0.0010
1/1 Epoch 217/500 0s 28ms/step - loss: 9.9590e-04
1/1 Epoch 218/500 0s 59ms/step - loss: 9.7544e-04
1/1 Epoch 219/500 0s 56ms/step - loss: 9.5540e-04
1/1 Epoch 220/500 0s 30ms/step - loss: 9.3578e-04
1/1 Epoch 221/500 0s 59ms/step - loss: 9.1655e-04
1/1 Epoch 222/500 0s 30ms/step - loss: 8.9772e-04
1/1 Epoch 223/500 0s 58ms/step - loss: 8.7929e-04
1/1 Epoch 224/500 0s 28ms/step - loss: 8.6123e-04
1/1 Epoch 225/500 0s 29ms/step - loss: 8.4354e-04
1/1 Epoch 226/500 0s 57ms/step - loss: 8.2621e-04
1/1 Epoch 227/500 0s 27ms/step - loss: 8.0924e-04
1/1 Epoch 228/500 0s 31ms/step - loss: 7.9261e-04
1/1 Epoch 229/500 0s 57ms/step - loss: 7.7634e-04
1/1 Epoch 230/500 0s 30ms/step - loss: 7.6039e-04
1/1 Epoch 231/500 0s 60ms/step - loss: 7.4477e-04
1/1 Epoch 232/500 0s 30ms/step - loss: 7.2947e-04
1/1 Epoch 233/500 0s 31ms/step - loss: 7.1449e-04
1/1 Epoch 234/500 0s 38ms/step - loss: 6.9981e-04
1/1 Epoch 235/500 0s 51ms/step - loss: 6.8544e-04
1/1 Epoch 236/500 0s 34ms/step - loss: 6.7136e-04
1/1 Epoch 237/500 0s 30ms/step - loss: 6.5757e-04
1/1 Epoch 238/500 0s 57ms/step - loss: 6.4406e-04
1/1 Epoch 239/500 0s 55ms/step - loss: 6.3083e-04
1/1 Epoch 240/500 0s 58ms/step - loss: 6.1787e-04
1/1 Epoch 241/500 0s 28ms/step - loss: 6.0518e-04
1/1 Epoch 242/500 0s 58ms/step - loss: 5.9275e-04
1/1 Epoch 243/500 0s 30ms/step - loss: 5.8057e-04
1/1 Epoch 244/500 0s 58ms/step - loss: 5.6865e-04
1/1 Epoch 245/500 0s 30ms/step - loss: 5.5697e-04
1/1 Epoch 246/500 0s 56ms/step - loss: 5.4553e-04
1/1 Epoch 247/500 0s 44ms/step - loss: 5.3432e-04
1/1 Epoch 248/500 0s 58ms/step - loss: 5.2335e-04
1/1 Epoch 249/500 0s 61ms/step - loss: 5.1260e-04
1/1 Epoch 250/500 0s 50ms/step - loss: 5.0207e-04
```

```
1/1 Epoch 250/500 0s 142ms/step - loss: 4.9176e-04
1/1 Epoch 251/500 0s 60ms/step - loss: 4.8166e-04
1/1 Epoch 252/500 0s 57ms/step - loss: 4.7176e-04
1/1 Epoch 253/500 0s 39ms/step - loss: 4.6208e-04
1/1 Epoch 254/500 0s 51ms/step - loss: 4.5258e-04
1/1 Epoch 255/500 0s 59ms/step - loss: 4.4328e-04
1/1 Epoch 256/500 0s 41ms/step - loss: 4.3418e-04
1/1 Epoch 257/500 0s 57ms/step - loss: 4.2526e-04
1/1 Epoch 258/500 0s 44ms/step - loss: 4.1652e-04
1/1 Epoch 259/500 0s 57ms/step - loss: 4.0797e-04
1/1 Epoch 260/500 0s 59ms/step - loss: 3.9959e-04
1/1 Epoch 261/500 0s 59ms/step - loss: 3.9138e-04
1/1 Epoch 262/500 0s 45ms/step - loss: 3.8334e-04
1/1 Epoch 263/500 0s 52ms/step - loss: 3.7547e-04
1/1 Epoch 264/500 0s 58ms/step - loss: 3.6776e-04
1/1 Epoch 265/500 0s 60ms/step - loss: 3.6020e-04
1/1 Epoch 266/500 0s 59ms/step - loss: 3.5280e-04
1/1 Epoch 267/500 0s 52ms/step - loss: 3.4556e-04
1/1 Epoch 268/500 0s 47ms/step - loss: 3.3846e-04
1/1 Epoch 269/500 0s 79ms/step - loss: 3.3151e-04
1/1 Epoch 270/500 0s 52ms/step - loss: 3.2470e-04
1/1 Epoch 271/500 0s 61ms/step - loss: 3.1803e-04
1/1 Epoch 272/500 0s 52ms/step - loss: 3.1150e-04
1/1 Epoch 273/500 0s 60ms/step - loss: 3.0510e-04
1/1 Epoch 274/500 0s 62ms/step - loss: 2.9883e-04
1/1 Epoch 275/500 0s 42ms/step - loss: 2.9269e-04
1/1 Epoch 276/500 0s 57ms/step - loss: 2.8668e-04
1/1 Epoch 277/500 0s 49ms/step - loss: 2.8079e-04
1/1 Epoch 278/500 0s 41ms/step - loss: 2.7503e-04
1/1 Epoch 279/500 0s 60ms/step - loss: 2.6938e-04
1/1 Epoch 280/500 0s 57ms/step - loss: 2.6384e-04
1/1 Epoch 281/500 0s 57ms/step - loss: 2.5842e-04
1/1 Epoch 282/500 0s 44ms/step - loss: 2.5311e-04
1/1 Epoch 283/500 0s 54ms/step - loss: 2.4791e-04
1/1 Epoch 284/500 0s 137ms/step - loss: 2.4282e-04
1/1 Epoch 285/500 0s 71ms/step - loss: 2.3784e-04
```

```
1/1  0s 71ms/step - loss: 2.3701e-04
Epoch 286/500
1/1  0s 53ms/step - loss: 2.3295e-04
Epoch 287/500
1/1  0s 63ms/step - loss: 2.2817e-04
Epoch 288/500
1/1  0s 45ms/step - loss: 2.2348e-04
Epoch 289/500
1/1  0s 59ms/step - loss: 2.1889e-04
Epoch 290/500
1/1  0s 59ms/step - loss: 2.1439e-04
Epoch 291/500
1/1  0s 43ms/step - loss: 2.0999e-04
Epoch 292/500
1/1  0s 52ms/step - loss: 2.0568e-04
Epoch 293/500
1/1  0s 53ms/step - loss: 2.0145e-04
Epoch 294/500
1/1  0s 67ms/step - loss: 1.9731e-04
Epoch 295/500
1/1  0s 57ms/step - loss: 1.9326e-04
Epoch 296/500
1/1  0s 42ms/step - loss: 1.8929e-04
Epoch 297/500
1/1  0s 47ms/step - loss: 1.8540e-04
Epoch 298/500
1/1  0s 31ms/step - loss: 1.8160e-04
Epoch 299/500
1/1  0s 59ms/step - loss: 1.7786e-04
Epoch 300/500
1/1  0s 30ms/step - loss: 1.7421e-04
Epoch 301/500
1/1  0s 32ms/step - loss: 1.7063e-04
Epoch 302/500
1/1  0s 59ms/step - loss: 1.6713e-04
Epoch 303/500
1/1  0s 58ms/step - loss: 1.6369e-04
Epoch 304/500
1/1  0s 44ms/step - loss: 1.6033e-04
Epoch 305/500
1/1  0s 32ms/step - loss: 1.5704e-04
Epoch 306/500
1/1  0s 58ms/step - loss: 1.5382e-04
Epoch 307/500
1/1  0s 53ms/step - loss: 1.5066e-04
Epoch 308/500
1/1  0s 33ms/step - loss: 1.4756e-04
Epoch 309/500
1/1  0s 59ms/step - loss: 1.4453e-04
Epoch 310/500
1/1  0s 30ms/step - loss: 1.4156e-04
Epoch 311/500
1/1  0s 31ms/step - loss: 1.3865e-04
Epoch 312/500
1/1  0s 61ms/step - loss: 1.3581e-04
Epoch 313/500
1/1  0s 30ms/step - loss: 1.3301e-04
Epoch 314/500
1/1  0s 59ms/step - loss: 1.3029e-04
Epoch 315/500
1/1  0s 58ms/step - loss: 1.2761e-04
Epoch 316/500
1/1  0s 28ms/step - loss: 1.2499e-04
Epoch 317/500
1/1  0s 32ms/step - loss: 1.2242e-04
Epoch 318/500
1/1  0s 58ms/step - loss: 1.1991e-04
Epoch 319/500
1/1  0s 59ms/step - loss: 1.1744e-04
Epoch 320/500
1/1  0s 32ms/step - loss: 1.1503e-04
Epoch 321/500
1/1  0s 57ms/step - loss: 1.1267e-04
```

```
1/1 0s 57ms/step - loss: 1.1207e-04
Epoch 322/500
1/1 0s 35ms/step - loss: 1.1036e-04
Epoch 323/500
1/1 0s 58ms/step - loss: 1.0809e-04
Epoch 324/500
1/1 0s 59ms/step - loss: 1.0587e-04
Epoch 325/500
1/1 0s 34ms/step - loss: 1.0369e-04
Epoch 326/500
1/1 0s 36ms/step - loss: 1.0156e-04
Epoch 327/500
1/1 0s 32ms/step - loss: 9.9476e-05
Epoch 328/500
1/1 0s 58ms/step - loss: 9.7432e-05
Epoch 329/500
1/1 0s 36ms/step - loss: 9.5431e-05
Epoch 330/500
1/1 0s 58ms/step - loss: 9.3471e-05
Epoch 331/500
1/1 0s 55ms/step - loss: 9.1551e-05
Epoch 332/500
1/1 0s 56ms/step - loss: 8.9670e-05
Epoch 333/500
1/1 0s 32ms/step - loss: 8.7828e-05
Epoch 334/500
1/1 0s 33ms/step - loss: 8.6026e-05
Epoch 335/500
1/1 0s 55ms/step - loss: 8.4258e-05
Epoch 336/500
1/1 0s 30ms/step - loss: 8.2526e-05
Epoch 337/500
1/1 0s 60ms/step - loss: 8.0831e-05
Epoch 338/500
1/1 0s 56ms/step - loss: 7.9171e-05
Epoch 339/500
1/1 0s 57ms/step - loss: 7.7542e-05
Epoch 340/500
1/1 0s 34ms/step - loss: 7.5951e-05
Epoch 341/500
1/1 0s 34ms/step - loss: 7.4390e-05
Epoch 342/500
1/1 0s 34ms/step - loss: 7.2860e-05
Epoch 343/500
1/1 0s 33ms/step - loss: 7.1365e-05
Epoch 344/500
1/1 0s 56ms/step - loss: 6.9899e-05
Epoch 345/500
1/1 0s 43ms/step - loss: 6.8464e-05
Epoch 346/500
1/1 0s 57ms/step - loss: 6.7057e-05
Epoch 347/500
1/1 0s 56ms/step - loss: 6.5679e-05
Epoch 348/500
1/1 0s 57ms/step - loss: 6.4330e-05
Epoch 349/500
1/1 0s 32ms/step - loss: 6.3009e-05
Epoch 350/500
1/1 0s 31ms/step - loss: 6.1715e-05
Epoch 351/500
1/1 0s 59ms/step - loss: 6.0447e-05
Epoch 352/500
1/1 0s 56ms/step - loss: 5.9205e-05
Epoch 353/500
1/1 0s 69ms/step - loss: 5.7990e-05
Epoch 354/500
1/1 0s 41ms/step - loss: 5.6800e-05
Epoch 355/500
1/1 0s 54ms/step - loss: 5.5633e-05
Epoch 356/500
1/1 0s 54ms/step - loss: 5.4490e-05
Epoch 357/500
1/1 0s 57ms/step - loss: 5.3370e-05
```

```
1/1  0s 57ms/step - loss: 5.3370e-05
Epoch 358/500
1/1  0s 65ms/step - loss: 5.2274e-05
Epoch 359/500
1/1  0s 132ms/step - loss: 5.1200e-05
Epoch 360/500
1/1  0s 59ms/step - loss: 5.0150e-05
Epoch 361/500
1/1  0s 64ms/step - loss: 4.9119e-05
Epoch 362/500
1/1  0s 134ms/step - loss: 4.8111e-05
Epoch 363/500
1/1  0s 63ms/step - loss: 4.7122e-05
Epoch 364/500
1/1  0s 52ms/step - loss: 4.6153e-05
Epoch 365/500
1/1  0s 63ms/step - loss: 4.5205e-05
Epoch 366/500
1/1  0s 53ms/step - loss: 4.4277e-05
Epoch 367/500
1/1  0s 53ms/step - loss: 4.3368e-05
Epoch 368/500
1/1  0s 140ms/step - loss: 4.2476e-05
Epoch 369/500
1/1  0s 133ms/step - loss: 4.1604e-05
Epoch 370/500
1/1  0s 139ms/step - loss: 4.0749e-05
Epoch 371/500
1/1  0s 58ms/step - loss: 3.9912e-05
Epoch 372/500
1/1  0s 60ms/step - loss: 3.9092e-05
Epoch 373/500
1/1  0s 63ms/step - loss: 3.8289e-05
Epoch 374/500
1/1  0s 53ms/step - loss: 3.7503e-05
Epoch 375/500
1/1  0s 53ms/step - loss: 3.6733e-05
Epoch 376/500
1/1  0s 45ms/step - loss: 3.5978e-05
Epoch 377/500
1/1  0s 59ms/step - loss: 3.5239e-05
Epoch 378/500
1/1  0s 66ms/step - loss: 3.4516e-05
Epoch 379/500
1/1  0s 58ms/step - loss: 3.3807e-05
Epoch 380/500
1/1  0s 60ms/step - loss: 3.3111e-05
Epoch 381/500
1/1  0s 50ms/step - loss: 3.2432e-05
Epoch 382/500
1/1  0s 57ms/step - loss: 3.1766e-05
Epoch 383/500
1/1  0s 59ms/step - loss: 3.1113e-05
Epoch 384/500
1/1  0s 57ms/step - loss: 3.0474e-05
Epoch 385/500
1/1  0s 60ms/step - loss: 2.9849e-05
Epoch 386/500
1/1  0s 138ms/step - loss: 2.9236e-05
Epoch 387/500
1/1  0s 101ms/step - loss: 2.8635e-05
Epoch 388/500
1/1  0s 128ms/step - loss: 2.8047e-05
Epoch 389/500
1/1  0s 57ms/step - loss: 2.7472e-05
Epoch 390/500
1/1  0s 60ms/step - loss: 2.6907e-05
Epoch 391/500
1/1  0s 54ms/step - loss: 2.6355e-05
Epoch 392/500
1/1  0s 64ms/step - loss: 2.5813e-05
Epoch 393/500
1/1  0s 53ms/step - loss: 2.5283e-05
```



```
1/1 Epoch 394/500 0s 57ms/step - loss: 2.4764e-05
1/1 Epoch 395/500 0s 64ms/step - loss: 2.4255e-05
1/1 Epoch 396/500 0s 50ms/step - loss: 2.3758e-05
1/1 Epoch 397/500 0s 60ms/step - loss: 2.3270e-05
1/1 Epoch 398/500 0s 47ms/step - loss: 2.2792e-05
1/1 Epoch 399/500 0s 58ms/step - loss: 2.2323e-05
1/1 Epoch 400/500 0s 63ms/step - loss: 2.1865e-05
1/1 Epoch 401/500 0s 53ms/step - loss: 2.1416e-05
1/1 Epoch 402/500 0s 63ms/step - loss: 2.0975e-05
1/1 Epoch 403/500 0s 60ms/step - loss: 2.0544e-05
1/1 Epoch 404/500 0s 49ms/step - loss: 2.0122e-05
1/1 Epoch 405/500 0s 51ms/step - loss: 1.9709e-05
1/1 Epoch 406/500 0s 47ms/step - loss: 1.9304e-05
1/1 Epoch 407/500 0s 51ms/step - loss: 1.8907e-05
1/1 Epoch 408/500 0s 56ms/step - loss: 1.8520e-05
1/1 Epoch 409/500 0s 45ms/step - loss: 1.8139e-05
1/1 Epoch 410/500 0s 52ms/step - loss: 1.7766e-05
1/1 Epoch 411/500 0s 64ms/step - loss: 1.7401e-05
1/1 Epoch 412/500 0s 131ms/step - loss: 1.7044e-05
1/1 Epoch 413/500 0s 55ms/step - loss: 1.6694e-05
1/1 Epoch 414/500 0s 59ms/step - loss: 1.6351e-05
1/1 Epoch 415/500 0s 38ms/step - loss: 1.6015e-05
1/1 Epoch 416/500 0s 57ms/step - loss: 1.5686e-05
1/1 Epoch 417/500 0s 60ms/step - loss: 1.5364e-05
1/1 Epoch 418/500 0s 47ms/step - loss: 1.5048e-05
1/1 Epoch 419/500 0s 57ms/step - loss: 1.4739e-05
1/1 Epoch 420/500 0s 62ms/step - loss: 1.4437e-05
1/1 Epoch 421/500 0s 51ms/step - loss: 1.4140e-05
1/1 Epoch 422/500 0s 57ms/step - loss: 1.3849e-05
1/1 Epoch 423/500 0s 57ms/step - loss: 1.3565e-05
1/1 Epoch 424/500 0s 136ms/step - loss: 1.3286e-05
1/1 Epoch 425/500 0s 139ms/step - loss: 1.3013e-05
1/1 Epoch 426/500 0s 51ms/step - loss: 1.2746e-05
1/1 Epoch 427/500 0s 64ms/step - loss: 1.2484e-05
1/1 Epoch 428/500 0s 64ms/step - loss: 1.2228e-05
1/1 Epoch 429/500 0s 60ms/step - loss: 1.1977e-05
```

```
1/1 Epoch 430/500 0s 60ms/step - loss: 1.1377e-05
1/1 Epoch 431/500 0s 70ms/step - loss: 1.1731e-05
1/1 Epoch 432/500 0s 56ms/step - loss: 1.1490e-05
1/1 Epoch 433/500 0s 61ms/step - loss: 1.1253e-05
1/1 Epoch 434/500 0s 52ms/step - loss: 1.1022e-05
1/1 Epoch 435/500 0s 60ms/step - loss: 1.0796e-05
1/1 Epoch 436/500 0s 58ms/step - loss: 1.0574e-05
1/1 Epoch 437/500 0s 44ms/step - loss: 1.0357e-05
1/1 Epoch 438/500 0s 57ms/step - loss: 1.0144e-05
1/1 Epoch 439/500 0s 58ms/step - loss: 9.9358e-06
1/1 Epoch 440/500 0s 70ms/step - loss: 9.7321e-06
1/1 Epoch 441/500 0s 51ms/step - loss: 9.5316e-06
1/1 Epoch 442/500 0s 56ms/step - loss: 9.3358e-06
1/1 Epoch 443/500 0s 61ms/step - loss: 9.1442e-06
1/1 Epoch 444/500 0s 41ms/step - loss: 8.9565e-06
1/1 Epoch 445/500 0s 60ms/step - loss: 8.7723e-06
1/1 Epoch 446/500 0s 47ms/step - loss: 8.5922e-06
1/1 Epoch 447/500 0s 59ms/step - loss: 8.4158e-06
1/1 Epoch 448/500 0s 49ms/step - loss: 8.2423e-06
1/1 Epoch 449/500 0s 60ms/step - loss: 8.0726e-06
1/1 Epoch 450/500 0s 58ms/step - loss: 7.9068e-06
1/1 Epoch 451/500 0s 51ms/step - loss: 7.7444e-06
1/1 Epoch 452/500 0s 142ms/step - loss: 7.5852e-06
1/1 Epoch 453/500 0s 48ms/step - loss: 7.4301e-06
1/1 Epoch 454/500 0s 129ms/step - loss: 7.2773e-06
1/1 Epoch 455/500 0s 68ms/step - loss: 7.1277e-06
1/1 Epoch 456/500 0s 60ms/step - loss: 6.9814e-06
1/1 Epoch 457/500 0s 71ms/step - loss: 6.8385e-06
1/1 Epoch 458/500 0s 58ms/step - loss: 6.6978e-06
1/1 Epoch 459/500 0s 67ms/step - loss: 6.5604e-06
1/1 Epoch 460/500 0s 76ms/step - loss: 6.4258e-06
1/1 Epoch 461/500 0s 119ms/step - loss: 6.2934e-06
1/1 Epoch 462/500 0s 117ms/step - loss: 6.1639e-06
1/1 Epoch 463/500 0s 144ms/step - loss: 6.0377e-06
1/1 Epoch 464/500 0s 51ms/step - loss: 5.9135e-06
1/1 Epoch 465/500 0s 63ms/step - loss: 5.7917e-06
1/1 Epoch 466/500 0s 67ms/step - loss: 5.6733e-06
```

```
1/1 0s 57ms/step - loss: 5.5565e-06
Epoch 466/500
1/1 0s 57ms/step - loss: 5.5565e-06
Epoch 467/500
1/1 0s 63ms/step - loss: 5.4425e-06
Epoch 468/500
1/1 0s 56ms/step - loss: 5.3305e-06
Epoch 469/500
1/1 0s 54ms/step - loss: 5.2212e-06
Epoch 470/500
1/1 0s 63ms/step - loss: 5.1135e-06
Epoch 471/500
1/1 0s 55ms/step - loss: 5.0086e-06
Epoch 472/500
1/1 0s 62ms/step - loss: 4.9055e-06
Epoch 473/500
1/1 0s 61ms/step - loss: 4.8049e-06
Epoch 474/500
1/1 0s 66ms/step - loss: 4.7068e-06
Epoch 475/500
1/1 0s 62ms/step - loss: 4.6098e-06
Epoch 476/500
1/1 0s 80ms/step - loss: 4.5152e-06
Epoch 477/500
1/1 0s 68ms/step - loss: 4.4224e-06
Epoch 478/500
1/1 0s 58ms/step - loss: 4.3318e-06
Epoch 479/500
1/1 0s 44ms/step - loss: 4.2425e-06
Epoch 480/500
1/1 0s 59ms/step - loss: 4.1555e-06
Epoch 481/500
1/1 0s 67ms/step - loss: 4.0701e-06
Epoch 482/500
1/1 0s 69ms/step - loss: 3.9864e-06
Epoch 483/500
1/1 0s 77ms/step - loss: 3.9045e-06
Epoch 484/500
1/1 0s 67ms/step - loss: 3.8246e-06
Epoch 485/500
1/1 0s 65ms/step - loss: 3.7457e-06
Epoch 486/500
1/1 0s 132ms/step - loss: 3.6685e-06
Epoch 487/500
1/1 0s 72ms/step - loss: 3.5933e-06
Epoch 488/500
1/1 0s 62ms/step - loss: 3.5198e-06
Epoch 489/500
1/1 0s 63ms/step - loss: 3.4473e-06
Epoch 490/500
1/1 0s 57ms/step - loss: 3.3765e-06
Epoch 491/500
1/1 0s 148ms/step - loss: 3.3071e-06
Epoch 492/500
1/1 0s 133ms/step - loss: 3.2393e-06
Epoch 493/500
1/1 0s 126ms/step - loss: 3.1725e-06
Epoch 494/500
1/1 0s 140ms/step - loss: 3.1073e-06
Epoch 495/500
1/1 0s 141ms/step - loss: 3.0435e-06
Epoch 496/500
1/1 0s 65ms/step - loss: 2.9810e-06
Epoch 497/500
1/1 0s 132ms/step - loss: 2.9197e-06
Epoch 498/500
1/1 0s 72ms/step - loss: 2.8598e-06
Epoch 499/500
1/1 0s 72ms/step - loss: 2.8010e-06
Epoch 500/500
1/1 0s 65ms/step - loss: 2.7436e-06
```

Out[5]:

```
<keras.src.callbacks.history.History at 0x7b1c91153b20>
```

Yeay, proses training selesai!

Sebelum lanjut, ayo kita review lagi proses pembelajaran neural network kita.

Di awal epochs, kamu bisa melihat nilai loss yang begitu besar, tetapi terus mengecil seiring pengulangan selanjutnya. Ketika training selesai, nilai loss sangatlah kecil. Hal ini menunjukkan bahwa model kita memiliki performa yang sangat baik dalam menyimpulkan hubungan antara angka X dan Y.

Kamu mungkin sadar bahwa kamu tidak butuh 500 epochs dan kamu bisa mencoba bereksperimen dengan epochs berbeda. Seperti yang kamu lihat dari contoh di atas, nilai loss nya sudah sangat kecil setelah epochs ke-50!

Menggunakan model

Kamu telah memiliki model yang telah di-training untuk mempelajari hubungan antara X dan Y. Kamu bisa menggunakan fungsi `model.predict` untuk memprediksi nilai Y dari nilai X baru. Misalnya, jika nilai X nya adalah 10, berapakah nilai Y?

Coba kamu tebak sebelum menjalankan kode di bawah:

In [6]:

```
print(model.predict(np.array([10.0])))
```

```
1/1 ————— 0s 111ms/step  
[[31.00483]]
```

Kamu mungkin menebak jawabannya adalah 31, tapi hasil dari model sedikit berbeda. Mengapa begitu?

Neural network berurusan dengan probabilitas, sehingga neural network mengkalkulasikan bahwa terdapat probabilitas yang sangat besar bahwa hubungan antara X dan Y adalah $Y=3X+1$, tapi dia tidak bisa menjawab dengan yakin hanya dengan menggunakan 6 data point. Hasilnya sangat dekat dengan 31, tapi belum tentu 31.

Semakin sering kamu menggunakan neural network, kamu akan semakin sering melihat pola seperti di atas terjadi. Kamu pasti akan selalu berurusan dengan probabilitas, bukan kepastian, dan akan melakukan sedikit coding untuk mengetahui hasil berdasarkan probabilitas, terutama jika berurusan dengan klasifikasi.

Selamat! 🎉

Percaya atau tidak, kamu telah mempelajari sebagian besar konsep ML yang dapat kamu gunakan dalam skenario yang lebih kompleks. Kamu telah mempelajari cara melatih neural network untuk mengetahui hubungan antara dua himpunan angka. Kamu telah membuat himpunan layers (walau dalam tutorial ini hanya satu lapisan) yang berisi neuron (juga dalam kasus ini, hanya satu), yang kemudian kamu kompilasi menggunakan fungsi `loss` dan `optimizer`.

Neural network, fungsi `loss`, dan fungsi `optimizer` dapat digunakan untuk proses menebak hubungan antara angka-angka, mengukur seberapa baik mereka melakukannya, lalu membuat parameter baru untuk tebakan baru. Begitulah cara kerja machine learning secara sederhana.