C4_W2_Lab_1_FirstAutoEncoder.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

RAM Disk Editing

Ungraded Lab: First Autoencoder

In this lab, you will build your first simple autoencoder. This will take in three-dimensional data, encodes it to two dimensions, and decodes it back to 3D.

Imports

[1]
```
import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

Prepare and preview the dataset

You will first create a synthetic dataset to act as input to the autoencoder. You can do that with the function below.

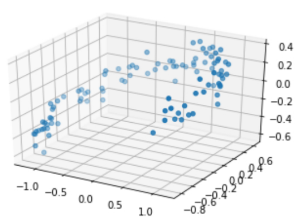
[2]
```
def generate_data(m):
    '''plots m random points on a 3D plane'''

    angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
    data = np.empty((m, 3))
    data[:,0] = np.cos(angles) + np.sin(angles)/2 + 0.1 * np.random.randn(m)/2
    data[:,1] = np.sin(angles) * 0.7 + 0.1 * np.random.randn(m) / 2
    data[:,2] = data[:, 0] * 0.1 + data[:, 1] * 0.3 + 0.1 * np.random.randn(m)

    return data
```

[3]
```
# use the function above to generate data points
X_train = generate_data(100)
X_train = X_train - X_train.mean(axis=0, keepdims=0)

# preview the data
ax = plt.axes(projection='3d')
ax.scatter3D(X_train[:, 0], X_train[:, 1], X_train[:, 2], cmap='Reds');
```



Build the Model

Now you will build the simple encoder-decoder model. Notice the number of neurons in each Dense layer. The model will contract in the encoder then expand in the decoder.

[4]
```
encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])
decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])

autoencoder = keras.models.Sequential([encoder, decoder])
```

Compile the Model

You can then setup the model for training.

[5]
```
autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=0.1))
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/optimizer_v2.py:356: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

Train the Model

You will configure the training to also use the input data as your target output. In our example, that will be `X_train`.

[6]
```
history = autoencoder.fit(X_train, X_train, epochs=200)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.0169
Epoch 172/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 173/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0161
Epoch 174/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 175/200
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 176/200
4/4 [==============================] - 0s 5ms/step - loss: 0.0154
Epoch 177/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0150
Epoch 178/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0148
Epoch 179/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0147
Epoch 180/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0144
Epoch 181/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0141
Epoch 182/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0140
Epoch 183/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0137
Epoch 184/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0135
Epoch 185/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0133
Epoch 186/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0130
Epoch 187/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0128
Epoch 188/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0126
Epoch 189/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0124
Epoch 190/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0122
Epoch 191/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0120
Epoch 192/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0119
Epoch 193/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0116
Epoch 194/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0116
Epoch 195/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0113
Epoch 196/200
4/4 [==============================] - 0s 5ms/step - loss: 0.0112
Epoch 197/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0109
Epoch 198/200
4/4 [==============================] - 0s 4ms/step - loss: 0.0107
Epoch 199/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0106
Epoch 200/200
4/4 [==============================] - 0s 3ms/step - loss: 0.0104
```

Plot the encoder output

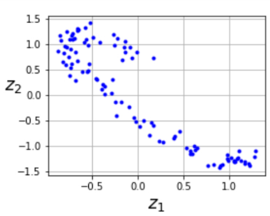
As mentioned, you can use the encoder to compress the input to two dimensions.

[7]
```python
# encode the data
codings = encoder.predict(X_train)

# see a sample input-encoder output pair
print(f'input point: {X_train[0]}')
print(f'encoded point: {codings[0]}')
```

```
input point: [0.98449343 0.07207633 0.12948845]
encoded point: [-0.5676748  1.1031077]
```

[8]
```python
# plot all encoder outputs
fig = plt.figure(figsize=(4,3))
plt.plot(codings[:,0], codings[:, 1], "b.")
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$z_2$", fontsize=18, rotation=0)
plt.grid(True)
plt.show()
```



Plot the Decoder output

The decoder then tries to reconstruct the original input. See the outputs below. You will see that although not perfect, it still follows the general shape of the original input.

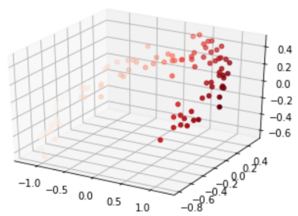
[9]
```python
# decode the encoder output
decodings = decoder.predict(codings)

# see a sample output for a single point
print(f'input point: {X_train[0]}')
print(f'encoded point: {codings[0]}')
print(f'decoded point: {decodings[0]}')
```

```
input point: [0.98449343 0.07207633 0.12948845]
encoded point: [-0.5676748  1.1031077]
decoded point: [0.9554824  0.1652338  0.05854119]
```

```python
# plot the decoder output
```

```python
ax = plt.axes(projection='3d')
ax.scatter3D(decodings[:, 0], decodings[:, 1], decodings[:, 2], c=decodings[:, 0], cmap='Reds');
```



That's it for this simple demonstration of the autoencoder!

✓ 0s completed at 4:38 PM