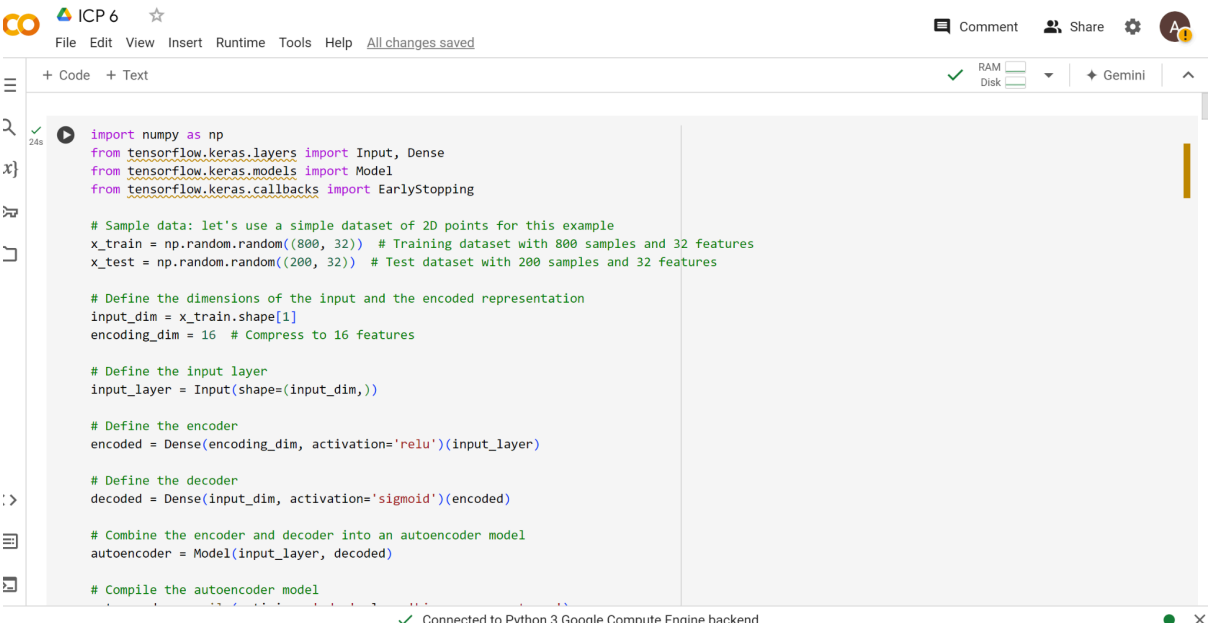
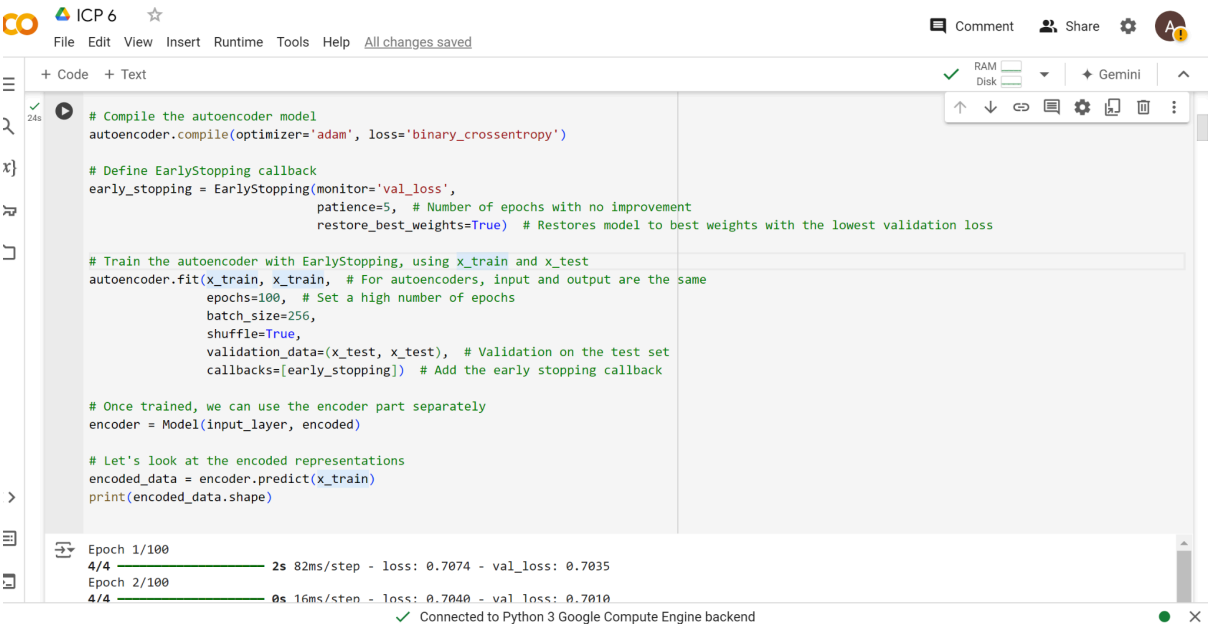
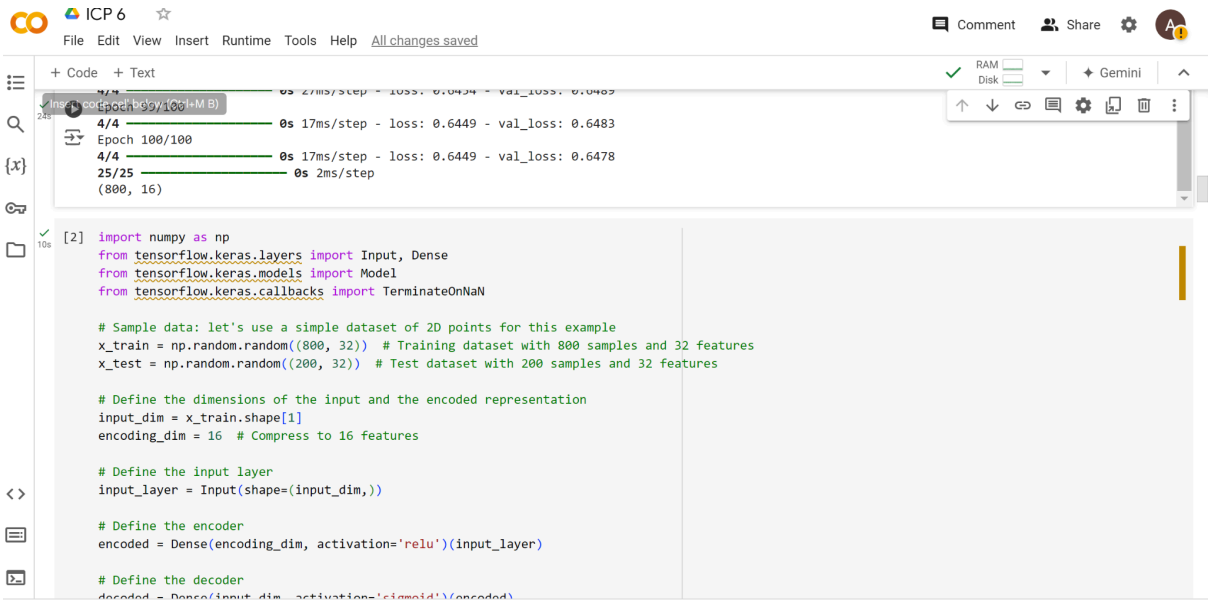
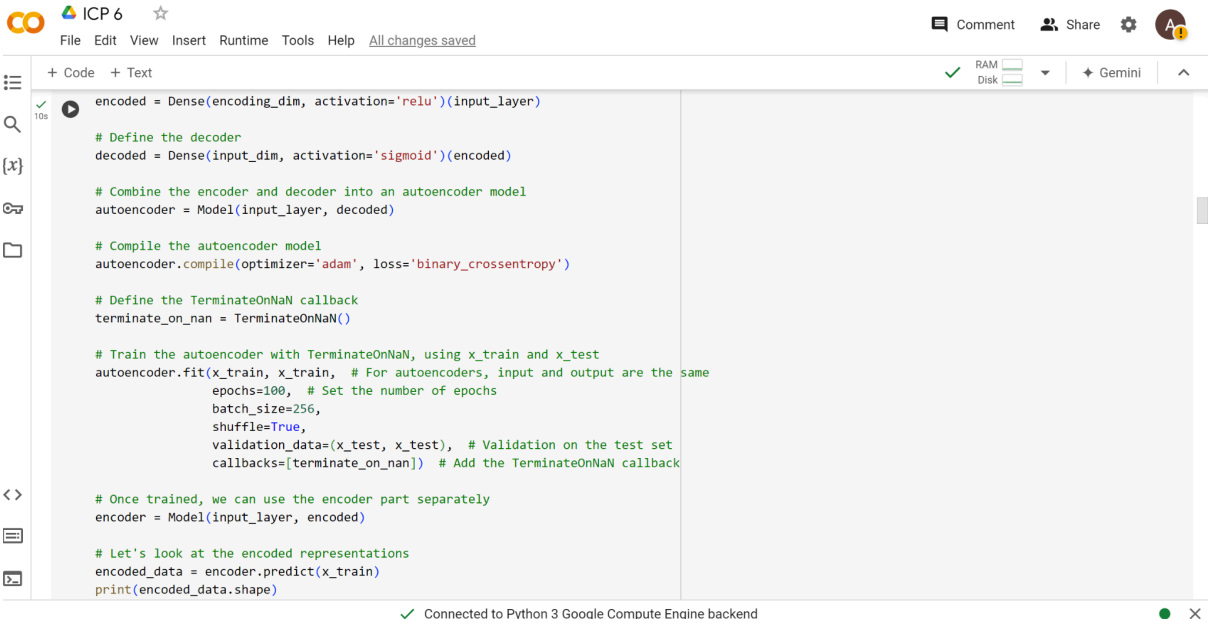
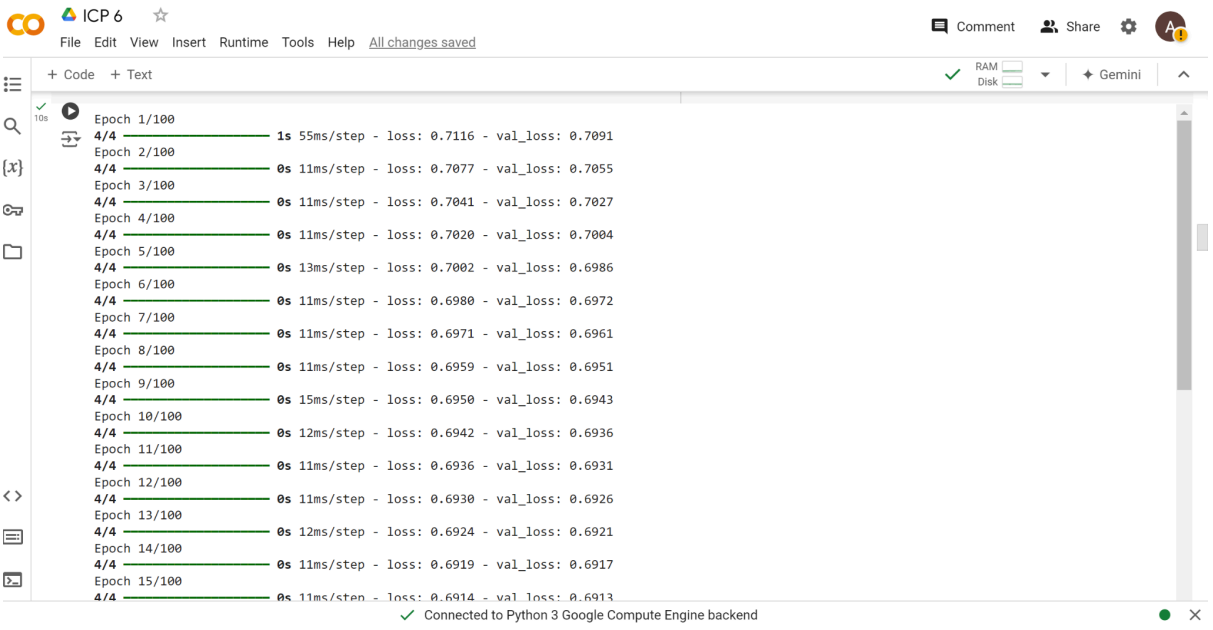
ICP 6

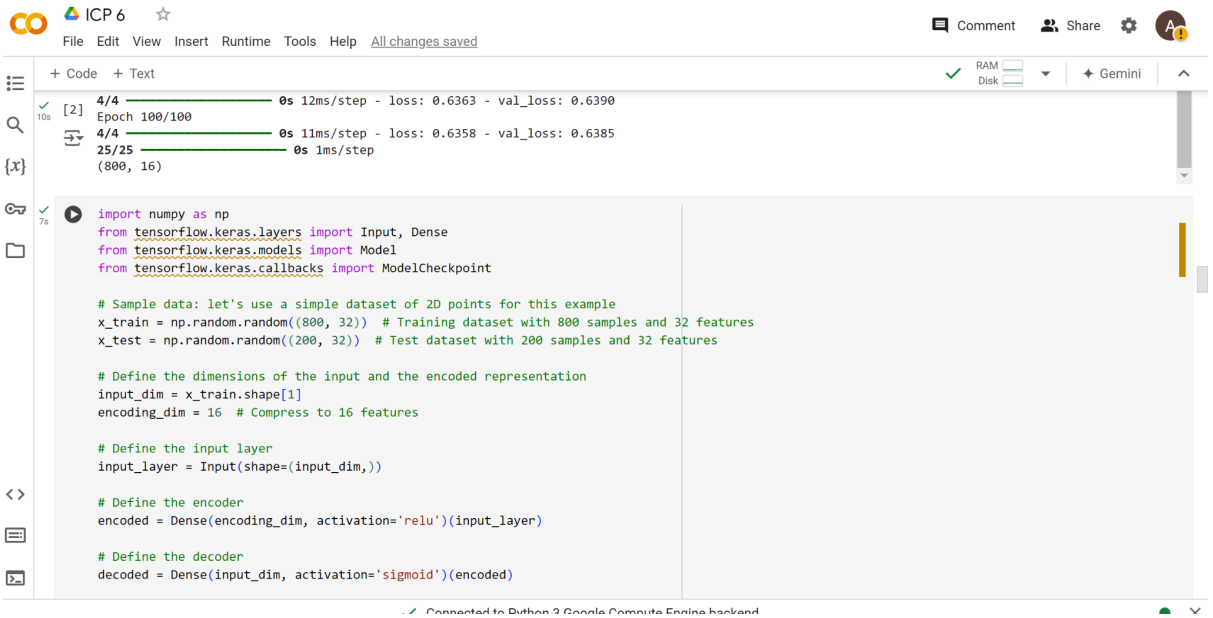

ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
# Sample data: let's use a simple dataset of 2D points for this example
x_train = np.random.random((800, 32)) # Training dataset with 800 samples and 32 features
x_test = np.random.random((200, 32)) # Test dataset with 200 samples and 32 features
# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16 # Compress to 16 features
# Define the input layer
input_layer = Input(shape=(input_dim,))
# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)
# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)
# Compile the autoencoder model
Connected to Python 3 Google Compute Engine backend


ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
# Define EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss',
patience=5, # Number of epochs with no improvement
restore_best_weights=True) # Restores model to best weights with the lowest validation loss
# Train the autoencoder with EarlyStopping, using x_train and x_test
autoencoder.fit(x_train, x_train, # For autoencoders, input and output are the same
epochs=100, # Set a high number of epochs
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test), # Validation on the test set
callbacks=[early_stopping]) # Add the early stopping callback
# Once trained, we can use the encoder part separately
encoder = Model(input_layer, encoded)
# Let's look at the encoded representations
encoded_data = encoder.predict(x_train)
print(encoded_data.shape)
Epoch 1/100
4/4 2s 82ms/step - loss: 0.7074 - val_loss: 0.7035
Epoch 2/100
4/4 0s 16ms/step - loss: 0.7040 - val_loss: 0.7010
Connected to Python 3 Google Compute Engine backend


ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
Epoch 99/100
4/4 0s 17ms/step - loss: 0.6449 - val_loss: 0.6483
Epoch 100/100
4/4 0s 17ms/step - loss: 0.6449 - val_loss: 0.6478
25/25 0s 2ms/step
(800, 16)
[2] import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import TerminateOnNaN
# Sample data: let's use a simple dataset of 2D points for this example
x_train = np.random.random((800, 32)) # Training dataset with 800 samples and 32 features
x_test = np.random.random((200, 32)) # Test dataset with 200 samples and 32 features
# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16 # Compress to 16 features
# Define the input layer
input_layer = Input(shape=(input_dim,))
# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Define the decoder


ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)
# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)
# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
# Define the TerminateOnNaN callback
terminate_on_nan = TerminateOnNaN()
# Train the autoencoder with TerminateOnNaN, using x_train and x_test
autoencoder.fit(x_train, x_train, # For autoencoders, input and output are the same
epochs=100, # Set the number of epochs
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test), # Validation on the test set
callbacks=[terminate_on_nan]) # Add the TerminateOnNaN callback
# Once trained, we can use the encoder part separately
encoder = Model(input_layer, encoded)
# Let's look at the encoded representations
encoded_data = encoder.predict(x_train)
print(encoded_data.shape)
Connected to Python 3 Google Compute Engine backend


ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
Epoch 1/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 55ms/step - loss: 0.7116 - val_loss: 0.7091
Epoch 2/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.7077 - val_loss: 0.7055
Epoch 3/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.7041 - val_loss: 0.7027
Epoch 4/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.7020 - val_loss: 0.7004
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.7002 - val_loss: 0.6986
Epoch 6/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6980 - val_loss: 0.6972
Epoch 7/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6971 - val_loss: 0.6961
Epoch 8/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6959 - val_loss: 0.6951
Epoch 9/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.6950 - val_loss: 0.6943
Epoch 10/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6942 - val_loss: 0.6936
Epoch 11/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6936 - val_loss: 0.6931
Epoch 12/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6930 - val_loss: 0.6926
Epoch 13/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6924 - val_loss: 0.6921
Epoch 14/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6919 - val_loss: 0.6917
Epoch 15/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6914 - val_loss: 0.6913
Connected to Python 3 Google Compute Engine backend


ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
[2]
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6363 - val_loss: 0.6390
Epoch 100/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6358 - val_loss: 0.6385
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
(800, 16)
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
# Sample data: let's use a simple dataset of 2D points for this example
x_train = np.random.random((800, 32)) # Training dataset with 800 samples and 32 features
x_test = np.random.random((200, 32)) # Test dataset with 200 samples and 32 features
# Define the dimensions of the input and the encoded representation
input_dim = x_train.shape[1]
encoding_dim = 16 # Compress to 16 features
# Define the input layer
input_layer = Input(shape=(input_dim,))
# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)

ICP 6

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Gemini

```
decoded = Dense(input_dim, activation='sigmoid')(encoded)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Define the ModelCheckpoint callback
checkpoint = ModelCheckpoint(filepath='autoencoder_best.keras',  # File path to save the model
                             monitor='val_loss',  # Metric to monitor
                             save_best_only=True,  # Save only the best model based on the monitored metric
                             mode='min',  # Minimize the monitored metric (e.g., validation loss)
                             save_weights_only=False,  # Save the entire model (set True to save only weights)
                             verbose=1)  # Print a message when saving the model

# Train the autoencoder with the ModelCheckpoint callback
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=50,  # Number of epochs
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),  # Validation data
                callbacks=[checkpoint])  # Add the ModelCheckpoint callback

# Once trained, we can use the encoder part separately
encoder = Model(input_layer, encoded)

# Let's look at the encoded representations
```

Connected to Python 3 Google Compute Engine backend

ICP 6

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

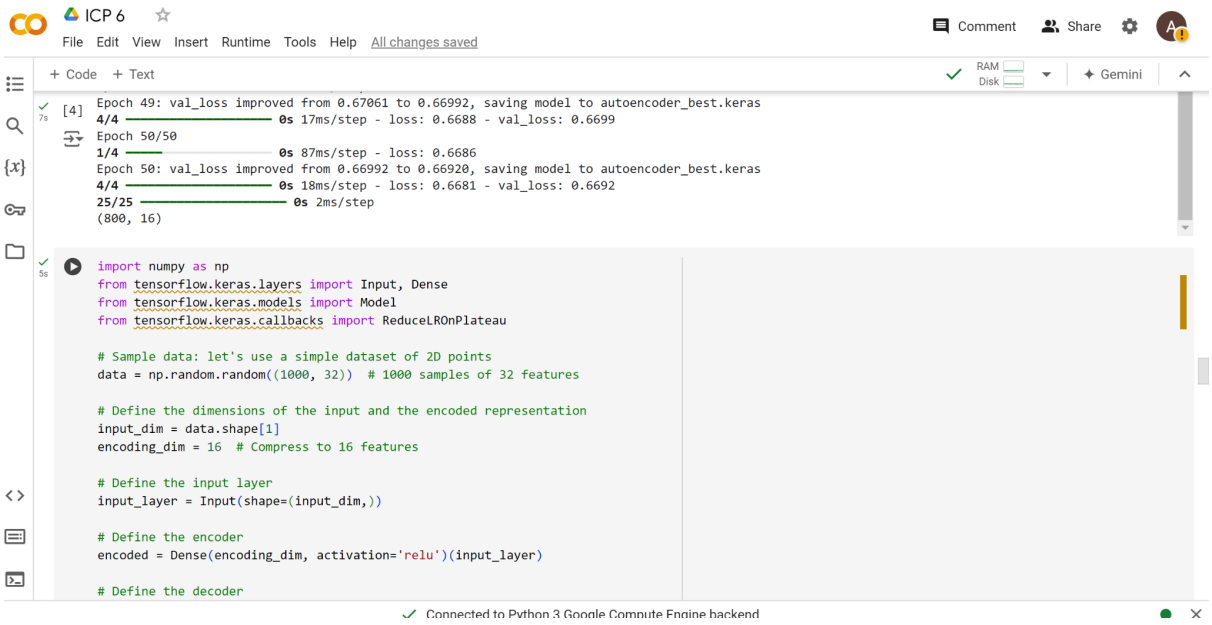
RAM Disk Gemini

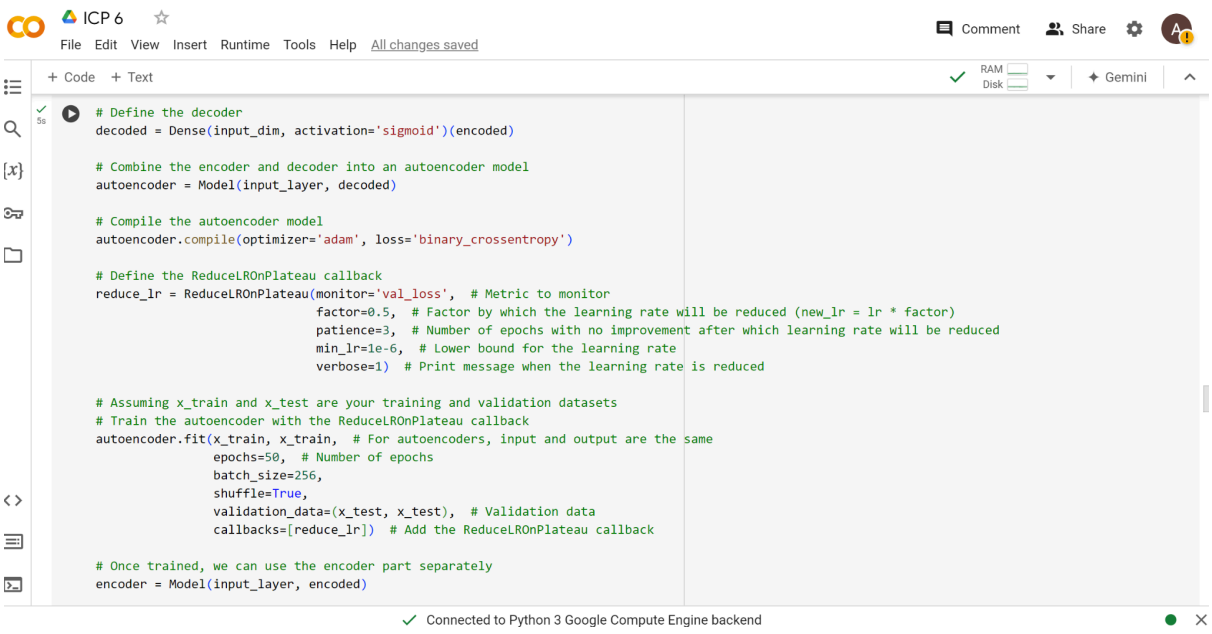
```
# Once trained, we can use the encoder part separately
encoder = Model(input_layer, encoded)

# Let's look at the encoded representations
encoded_data = encoder.predict(x_train)
print(encoded_data.shape)
```

```
Epoch 1/50
1/4 ━━━━━━━━━━━━━━━━━━━━ 2s 855ms/step - loss: 0.7062
Epoch 1: val_loss improved from inf to 0.70645, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - loss: 0.7066 - val_loss: 0.7064
Epoch 2/50
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.7037
Epoch 2: val_loss improved from 0.70645 to 0.70376, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.7039 - val_loss: 0.7038
Epoch 3/50
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - loss: 0.7024
Epoch 3: val_loss improved from 0.70376 to 0.70155, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 0.7019 - val_loss: 0.7015
Epoch 4/50
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.7004
Epoch 4: val_loss improved from 0.70155 to 0.69976, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.6999 - val_loss: 0.6998
Epoch 5/50
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step - loss: 0.6987
Epoch 5: val_loss improved from 0.69976 to 0.69832, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.6984 - val_loss: 0.6983
Epoch 6/50
```

Connected to Python 3 Google Compute Engine backend


ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
[4]
Epoch 49: val_loss improved from 0.67061 to 0.66992, saving model to autoencoder_best.keras
4/4 0s 17ms/step - loss: 0.6688 - val_loss: 0.6699
Epoch 50/50
1/4 0s 87ms/step - loss: 0.6686
Epoch 50: val_loss improved from 0.66992 to 0.66920, saving model to autoencoder_best.keras
4/4 0s 18ms/step - loss: 0.6681 - val_loss: 0.6692
25/25 0s 2ms/step
(800, 16)
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ReduceLROnPlateau
# Sample data: let's use a simple dataset of 2D points
data = np.random.random((1000, 32)) # 1000 samples of 32 features
# Define the dimensions of the input and the encoded representation
input_dim = data.shape[1]
encoding_dim = 16 # Compress to 16 features
# Define the input layer
input_layer = Input(shape=(input_dim,))
# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)
# Define the decoder
Connected to Python 3 Google Compute Engine backend


ICP 6
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)
# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)
# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
# Define the ReduceLROnPlateau callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss', # Metric to monitor
factor=0.5, # Factor by which the learning rate will be reduced (new_lr = lr * factor)
patience=3, # Number of epochs with no improvement after which learning rate will be reduced
min_lr=1e-6, # Lower bound for the learning rate
verbose=1) # Print message when the learning rate is reduced
# Assuming x_train and x_test are your training and validation datasets
# Train the autoencoder with the ReduceLROnPlateau callback
autoencoder.fit(x_train, x_train, # For autoencoders, input and output are the same
epochs=50, # Number of epochs
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test), # Validation data
callbacks=[reduce_lr]) # Add the ReduceLROnPlateau callback
# Once trained, we can use the encoder part separately
encoder = Model(input_layer, encoded)
Connected to Python 3 Google Compute Engine backend

ICP 6

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Gemini

```
# Once trained, we can use the encoder part separately
encoder = Model(input_layer, encoded)

# Let's look at the encoded representations
encoded_data = encoder.predict(data)
print(encoded_data.shape)
```

```
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.6848 - val_loss: 0.6855 - learning_rate: 0.0010
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6841 - val_loss: 0.6850 - learning_rate: 0.0010
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6838 - val_loss: 0.6845 - learning_rate: 0.0010
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6833 - val_loss: 0.6839 - learning_rate: 0.0010
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.6826 - val_loss: 0.6834 - learning_rate: 0.0010
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6823 - val_loss: 0.6828 - learning_rate: 0.0010
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6814 - val_loss: 0.6821 - learning_rate: 0.0010
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6808 - val_loss: 0.6815 - learning_rate: 0.0010
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6802 - val_loss: 0.6809 - learning_rate: 0.0010
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6794 - val_loss: 0.6802 - learning_rate: 0.0010
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.6789 - val_loss: 0.6796 - learning_rate: 0.0010
Epoch 34/50
```

Connected to Python 3 Google Compute Engine backend

ICP 6

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Gemini

```
[5] 4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6684 - val_loss: 0.6692 - learning_rate: 0.0010
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.6674 - val_loss: 0.6686 - learning_rate: 0.0010
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
(1000, 16)
```

```
import numpy as np
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TerminateOnNaN, ReduceLROnPlateau

# Sample data: let's use a simple dataset of 2D points
data = np.random.random((1000, 32))  # 1000 samples of 32 features

# Define the dimensions of the input and the encoded representation
input_dim = data.shape[1]
encoding_dim = 16  # Compress to 16 features

# Define the input layer
input_layer = Input(shape=(input_dim,))

# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)

# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)

# Combine the encoder and decoder into an autoencoder model
```

Connected to Python 3 Google Compute Engine backend

ICP 6

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Gemini

```
# Define the encoder
encoded = Dense(encoding_dim, activation='relu')(input_layer)

# Define the decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)

# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(input_layer, decoded)

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Define the callbacks
# EarlyStopping to stop training if validation loss doesn't improve
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# ModelCheckpoint to save the best model based on validation loss
checkpoint = ModelCheckpoint(filepath='autoencoder_best.keras', monitor='val_loss', save_best_only=True, verbose=1)

# TerminateOnNaN to stop training if the loss becomes NaN
terminate_on_nan = TerminateOnNaN()

# ReduceLROnPlateau to reduce the learning rate when validation loss plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6, verbose=1)

# Assuming x_train and x_test are your training and validation datasets
# Training the autoencoder with all the callbacks
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
```

Connected to Python 3 Google Compute Engine backend

ICP 6

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Gemini

```
# Assuming x_train and x_test are your training and validation datasets
# Training the autoencoder with all the callbacks
autoencoder.fit(x_train, x_train,  # For autoencoders, input and output are the same
                epochs=100,  # Set a higher number of epochs, since callbacks will handle stopping early
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test),  # Validation data
                callbacks=[reduce_lr, early_stopping, checkpoint, terminate_on_nan])  # Using multiple callbacks

# Once trained, we can use the encoder part separately
encoder = Model(input_layer, encoded)

# Let's look at the encoded representations
encoded_data = encoder.predict(data)
print(encoded_data.shape)
```

```
Epoch 1/100
1/4 ━━━━━━━━━━━━━━━━━━━━ 2s 798ms/step - loss: 0.7144
Epoch 1: val_loss improved from inf to 0.71033, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - loss: 0.7139 - val_loss: 0.7103 - learning_rate: 0.0010
Epoch 2/100
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.7093
Epoch 2: val_loss improved from 0.71033 to 0.70587, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.7086 - val_loss: 0.7059 - learning_rate: 0.0010
Epoch 3/100
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - loss: 0.7042
Epoch 3: val_loss improved from 0.70587 to 0.70254, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.7041 - val_loss: 0.7025 - learning_rate: 0.0010
```

Connected to Python 3 Google Compute Engine backend

ICP 6

File Edit View Insert Runtime Tools Help All changes saved

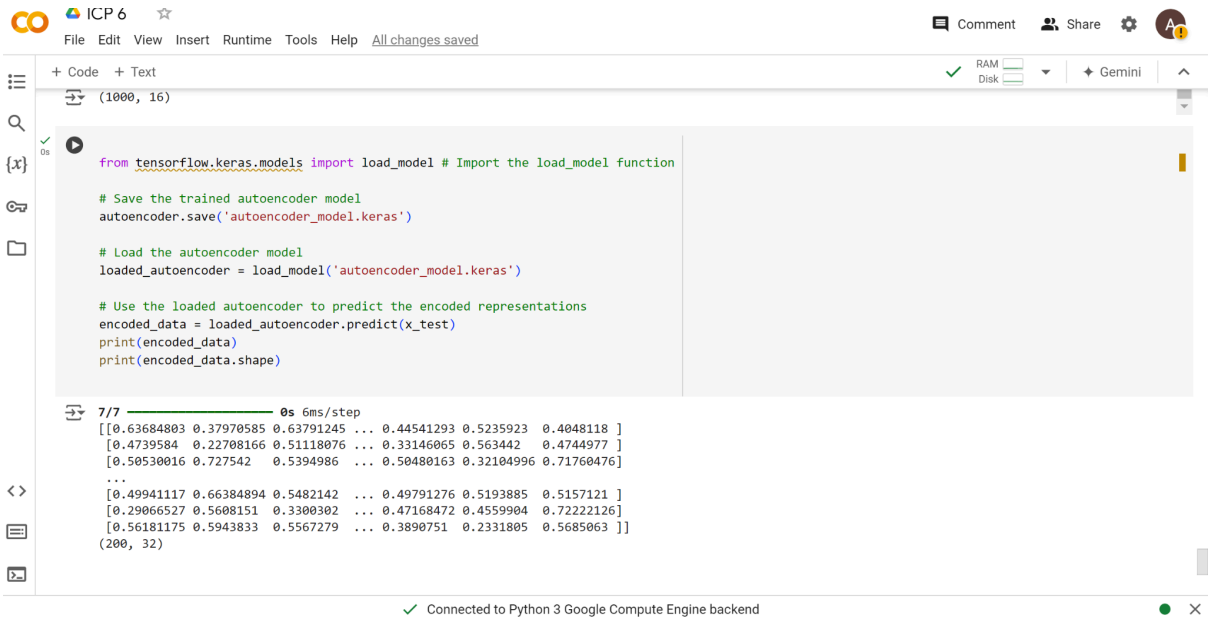
Comment Share

+ Code + Text

RAM Disk Gemini

```
[7] Epoch 99: val_loss improved from 0.64151 to 0.64111, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.6413 - val_loss: 0.6411 - learning_rate: 0.0010
Epoch 100/100
1/4 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.6399
Epoch 100: val_loss improved from 0.64111 to 0.64071, saving model to autoencoder_best.keras
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.6401 - val_loss: 0.6407 - learning_rate: 0.0010
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
(1000, 16)
```

Loading the model and predicting



Youtube video:https://youtu.be/DYkWaphYpko

GitHub link :https://github.com/w8162583/bda.git