1. simple vector addition:

```
import tensorflow as tf
# Define two vectors
vector1 = tf.constant([1, 2, 3], dtype=tf.float32)
vector2 = tf.constant([4, 5, 6], dtype=tf.float32)
# Perform vector addition
result = tf.add(vector1, vector2)
```

2. Implement a regression model in Keras.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Generate some example data
np.random.seed(0)
x_train = np.random.rand(100, 1)
y_train = 2 * x_train + 1 + 0.1 * np.random.randn(100, 1)

# Define the model
model = Sequential([
   Dense(1, input_dim=1)
])

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

# Train the model
model.fit(x_train, y_train, epochs=100, batch_size=10)

# Make some predictions
```

```python
x_test = np.array([[0.5], [1.5]])
predictions = model.predict(x_test)

# Print predictions
print("Predictions:", predictions)
```

3. Implement a perceptron in TensorFlow/Keras Environment.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Generate some example data
np.random.seed(0)
x_train = np.random.rand(100, 2)  # 100 points with 2 features
y_train = (x_train[:, 0] + x_train[:, 1] > 1).astype(int)  # Label 1 if the sum of the features > 1, else 0

# Define the model
model = Sequential([
   Dense(1, input_dim=2, activation='sigmoid')  # Single perceptron unit
])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train, epochs=100, batch_size=10)
# Make some predictions
x_test = np.array([[0.1, 0.9], [0.8, 0.2]])
predictions = model.predict(x_test)

# Print predictions
print("Predictions:", predictions)
```

4. Implement a Feed-Forward Network in TensorFlow/Keras.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np
# Generate some example data
np.random.seed(0)
x_train = np.random.rand(100, 1)  # 100 points with 1 feature
y_train = 2 * x_train + 1 + 0.1 * np.random.randn(100, 1)  # Linear relationship with some noise
# Define the model
model = Sequential([
   Dense(10, input_dim=1, activation='relu'),  # First hidden layer with 10 neurons and ReLU activation
   Dense(10, activation='relu'),             # Second hidden layer with 10 neurons and ReLU activation
   Dense(1)                             # Output layer with 1 neuron (for regression)
])
# Compile the model
model.compile(optimizer='adam', loss='mse')

# Train the model
model.fit(x_train, y_train, epochs=100, batch_size=10)

# Make some predictions
x_test = np.array([[0.5], [1.5]])
predictions = model.predict(x_test)

# Print predictions
print("Predictions:", predictions)
```

5. Implement an Image Classifier using CNN in TensorFlow/Keras.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
```

```python
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Reshape the data to include a channel dimension
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.1)
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
```

6. Improve the Deep learning model by fine tuning hyper parameters

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to_categorical


# Load the MNIST dataset

(x_train, y_train), (x_test, y_test) = mnist.load_data()


# Reshape the data to include a channel dimension

x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0

x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0


# One-hot encode the labels

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)


# Define the model

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),

    MaxPooling2D((2, 2)),

    Dropout(0.25),  # Adding dropout to reduce overfitting

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    Dropout(0.25),  # Adding dropout to reduce overfitting

    Flatten(),

    Dense(128, activation='relu'),

    Dropout(0.5),  # Adding dropout to reduce overfitting

    Dense(10, activation='softmax')

])
```

```python
# Compile the model with a custom learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train, epochs=20, batch_size=32, validation_split=0.1)
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
```

7. . Implement a Transfer Learning concept in Image Classification.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.applications import VGG16
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the pixel values
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Load the VGG16 model pre-trained on ImageNet, excluding the top fully connected layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
```

```python
# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Define the model
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(x_train)

# Train the model
model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=20, validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
```

8.Using a pre trained model on Keras for Transfer Learning

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the pixel values
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Load the MobileNetV2 model pre-trained on ImageNet, excluding the top fully connected layers
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
   layer.trainable = False

# Define the model
model = Sequential([
   base_model,
   GlobalAveragePooling2D(),
```

```
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(x_train)

# Train the model
model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=20, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
```

9. Perform Sentiment Analysis using RNN

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the IMDB dataset
```

```
max_features = 10000  # Only consider the top 10,000 words in the dataset
max_len = 500  # Only consider the first 500 words of each review

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to ensure they are all the same length
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# Define the model
model = Sequential([
    Embedding(max_features, 128, input_length=max_len),
    SimpleRNN(128),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
```

10. . Implement an LSTM based Autoencoder in TensorFlow/Keras.

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, RepeatVector, Dense
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
# Load the IMDB dataset
max_features = 10000  # Limit the number of words in the vocabulary
max_len = 100  # Maximum length of the input sequences
(x_train, _), (x_test, _) = imdb.load_data(num_words=max_features)
# Pad sequences to ensure uniform input size
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
# Define the LSTM Autoencoder model
timesteps = max_len
input_dim = max_features
# Encoder
encoder_inputs = Input(shape=(timesteps, input_dim))
encoded = LSTM(64, activation='relu', return_sequences=False)(encoder_inputs)
latent_dim = 32
encoded_latent = Dense(latent_dim, activation='relu')(encoded)
# Latent Space Representation
latent_inputs = Input(shape=(latent_dim,))
decoded_latent = Dense(64, activation='relu')(latent_inputs)
decoded = RepeatVector(timesteps)(decoded_latent)
decoded = LSTM(input_dim, activation='sigmoid', return_sequences=True)(decoded)
# Define the full Autoencoder model
autoencoder = Model(encoder_inputs, decoded)
# Define the Encoder model
encoder_model = Model(encoder_inputs, encoded_latent)
# Define the Decoder model
decoder_inputs = Input(shape=(latent_dim,))
decoder_outputs = autoencoder.layers[-2](decoder_inputs)
decoder_outputs = autoencoder.layers[-1](decoder_outputs)
decoder_model = Model(decoder_inputs, decoder_outputs)
```

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

# Train the model
autoencoder.fit(x_train, x_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
loss = autoencoder.evaluate(x_test, x_test)
print(f"Test loss: {loss:.4f}")
```

11. . Image generation using GAN

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Reshape, Flatten, LeakyReLU, BatchNormalization, Conv2DTranspose, Conv2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
import numpy as np
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, _), (_, _) = mnist.load_data()
x_train = x_train / 255.0
x_train = np.expand_dims(x_train, axis=-1)

# Define the Generator model
def build_generator():
    model = Sequential()
    model.add(Dense(128 * 7 * 7, input_dim=100))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((7, 7, 128)))
    model.add(BatchNormalization())
```

```
    model.add(Conv2DTranspose(128, kernel_size=3, strides=2, padding='same'))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(64, kernel_size=3, strides=2, padding='same'))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2DTranspose(1, kernel_size=3, activation='sigmoid', padding='same'))

    model.compile(loss='binary_crossentropy', optimizer=Adam())
    return model

# Define the Discriminator model
def build_discriminator():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=3, strides=2, padding='same', input_shape=(28, 28, 1)))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Conv2D(128, kernel_size=3, strides=2, padding='same'))
    model.add(LeakyReLU(alpha=0.2))

    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer=Adam())
    return model

# Define the GAN model
def build_gan(generator, discriminator):
    model = Sequential()
    model.add(generator)
```

```
    model.add(discriminator)
    return model

# Create and compile models
generator = build_generator()
discriminator = build_discriminator()
discriminator.trainable = False

gan = build_gan(generator, discriminator)
gan.compile(loss='binary_crossentropy', optimizer=Adam())

# Training parameters
epochs = 10000
batch_size = 64
half_batch = batch_size // 2

# Training loop
for epoch in range(epochs):
    # Train Discriminator
    idx = np.random.randint(0, x_train.shape[0], half_batch)
    real_imgs = x_train[idx]
    real_labels = np.ones((half_batch, 1))

    noise = np.random.randn(half_batch, 100)
    fake_imgs = generator.predict(noise)
    fake_labels = np.zeros((half_batch, 1))

    d_loss_real = discriminator.train_on_batch(real_imgs, real_labels)
    d_loss_fake = discriminator.train_on_batch(fake_imgs, fake_labels)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
    # Train Generator
    noise = np.random.randn(batch_size, 100)
    valid_labels = np.ones((batch_size, 1))
    g_loss = gan.train_on_batch(noise, valid_labels)

    # Print progress
    if epoch % 1000 == 0:
        print(f"{epoch}/{epochs} [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}] [G loss: {g_loss}]")

        # Save generated images
        if epoch % 1000 == 0:
            generated_images = generator.predict(np.random.randn(25, 100))
            generated_images = 0.5 * generated_images + 0.5  # Rescale images to [0, 1]
            fig, axs = plt.subplots(5, 5)
            cnt = 0
            for i in range(5):
                for j in range(5):
                    axs[i,j].imshow(generated_images[cnt, :, :, 0], cmap='gray')
                    axs[i,j].axis('off')
                    cnt += 1
            plt.show()
```

12. Train a Deep learning model to classify a given image using pre trained model

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

# Define the path to your image
img_path = 'path_to_your_image.jpg'  # Replace with the path to your image
```

```
# Load and preprocess the image
img = image.load_img(img_path, target_size=(224, 224))  # VGG16 expects 224x224 images
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Load the VGG16 model pre-trained on ImageNet
model = VGG16(weights='imagenet')

# Predict the class of the image
predictions = model.predict(img_array)

# Decode the predictions
decoded_predictions = decode_predictions(predictions, top=3)[0]  # Get top 3 predictions

# Print the results
for i, (class_id, class_name, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {class_name} ({score:.2f})")
```

13. Recommendation system from sales data using Deep Learning

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Example dataset
data = {
    'user_id': [1, 1, 2, 2, 3, 3, 4, 4],
```

```
    'item_id': [1, 2, 1, 3, 2, 3, 1, 2],
    'rating': [5, 3, 4, 2, 5, 1, 2, 3]
}

df = pd.DataFrame(data)

# Encode user and item IDs
user_encoder = LabelEncoder()
item_encoder = LabelEncoder()

df['user_id_encoded'] = user_encoder.fit_transform(df['user_id'])
df['item_id_encoded'] = item_encoder.fit_transform(df['item_id'])

# Define model parameters
n_users = len(user_encoder.classes_)
n_items = len(item_encoder.classes_)
embedding_dim = 10

# Build the recommendation model
model = Sequential([
    Embedding(input_dim=n_users, output_dim=embedding_dim, input_length=1,
name='user_embedding'),
    Flatten(),
    Dense(embedding_dim, activation='relu'),
    Dense(n_items, activation='sigmoid')  # Use sigmoid for rating prediction
])

model.compile(optimizer=Adam(), loss='binary_crossentropy')  # Use 'binary_crossentropy' for
simplicity

# Prepare input features and target
X = df[['user_id_encoded', 'item_id_encoded']].values
```

```
y = df['rating'].values

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model.fit([X_train[:, 0], X_train[:, 1]], y_train, epochs=10, batch_size=2, validation_split=0.2)

# Evaluate the model
loss = model.evaluate([X_test[:, 0], X_test[:, 1]], y_test)
print(f"Test loss: {loss:.4f}")
```

14. Implement Object Detection using CNN

```
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image

# Load the pre-trained object detection model from TensorFlow Hub
model_url = "https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1"
detector = hub.load(model_url).signatures['default']

# Load and preprocess the image
def load_image(image_path):
    img = Image.open(image_path)
    img = img.convert('RGB')  # Ensure image is in RGB format
    img_np = np.array(img)
    return img_np

def preprocess_image(image_np):
```

```python
    image_tensor = tf.convert_to_tensor(image_np, dtype=tf.float32)
    image_tensor = tf.image.convert_image_dtype(image_tensor, dtype=tf.uint8)
    image_tensor = tf.expand_dims(image_tensor, 0)  # Add batch dimension
    return image_tensor

# Perform object detection on the input image
def detect_objects(image_tensor):
    result = detector(image_tensor)
    return result

def draw_boxes(image_np, boxes, class_ids, scores):
    fig, ax = plt.subplots(1, figsize=(12, 9))
    ax.imshow(image_np)

    for i in range(len(boxes)):
        box = boxes[i]
        class_id = class_ids[i]
        score = scores[i]

        # Draw bounding box
        ymin, xmin, ymax, xmax = box
        rect = patches.Rectangle((xmin * image_np.shape[1], ymin * image_np.shape[0]),
                                 (xmax - xmin) * image_np.shape[1],
                                 (ymax - ymin) * image_np.shape[0],
                                 linewidth=2, edgecolor='r', facecolor='none')
        ax.add_patch(rect)
        plt.text(xmin * image_np.shape[1], ymin * image_np.shape[0],
                 f'{class_id} ({score:.2f})', color='red', fontsize=12, bbox=dict(facecolor='yellow', alpha=0.5))

    plt.show()
```

```
# Example usage
image_path = 'path_to_your_image.jpg'  # Replace with the path to your image
image_np = load_image(image_path)
image_tensor = preprocess_image(image_np)
result = detect_objects(image_tensor)

# Extract and draw results
boxes = result['detection_boxes'].numpy()[0]
class_ids = result['detection_classes'].numpy()[0]
scores = result['detection_scores'].numpy()[0]

draw_boxes(image_np, boxes, class_ids, scores)
```

15. Implement any simple Reinforcement Algorithm for an NLP problem

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM
from tensorflow.keras.optimizers import Adam
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')

class TextGenerationEnv:
    def __init__(self, vocab_size, max_sequence_length):
        self.vocab_size = vocab_size
        self.max_sequence_length = max_sequence_length
        self.current_step = 0
        self.text_sequence = np.zeros(max_sequence_length, dtype=int)

    def reset(self):
        self.current_step = 0
```

```
        self.text_sequence = np.zeros(self.max_sequence_length, dtype=int)
        return self.text_sequence

    def step(self, action):
        self.text_sequence[self.current_step] = action
        reward = self.compute_reward(self.text_sequence)
        self.current_step += 1
        done = self.current_step >= self.max_sequence_length
        return self.text_sequence, reward, done

    def compute_reward(self, text_sequence):
        # For simplicity, we use the length of the sequence as a reward
        return np.sum(text_sequence)

def build_policy_network(vocab_size, embedding_dim, hidden_units):
    model = Sequential([
        Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=1),
        LSTM(hidden_units, return_sequences=False),
        Dense(vocab_size, activation='softmax')
    ])
    return model

def train_policy_network(env, policy_network, optimizer, episodes=1000):
    for episode in range(episodes):
        state = env.reset()
        done = False
        total_reward = 0
        while not done:
            state = np.expand_dims(state, axis=0)  # Add batch dimension
            action_probs = policy_network.predict(state)
            action = np.argmax(action_probs)
```

```
            next_state, reward, done = env.step(action)
            total_reward += reward

            # Calculate the loss and update the policy network
            with tf.GradientTape() as tape:
                action_probs = policy_network(state, training=True)
                loss = -tf.math.log(action_probs[0, action]) * reward
            grads = tape.gradient(loss, policy_network.trainable_variables)
            optimizer.apply_gradients(zip(grads, policy_network.trainable_variables))

            state = next_state

        print(f'Episode {episode+1}, Total Reward: {total_reward}')

# Parameters
vocab_size = 10  # Example vocabulary size
embedding_dim = 8
hidden_units = 16
max_sequence_length = 10
learning_rate = 0.001

# Initialize environment and policy network
env = TextGenerationEnv(vocab_size, max_sequence_length)
policy_network = build_policy_network(vocab_size, embedding_dim, hidden_units)
optimizer = Adam(learning_rate)

# Train policy network
train_policy_network(env, policy_network, optimizer)
```