Урок 8 GAN

Введение в нейронные сети

Урок 8. GAN

1. Обучите нейронную сеть любой архитектуры, которой не было на курсе, либо нейронную сеть разобранной архитектуры, но на том датасете, которого не было на уроках. Сделайте анализ того, что вам помогло в улучшения работы нейронной сети
2. Сделайте краткий обзор научной работы, посвящённой алгоритму нейронных сетей, не рассматриваемому ранее на курсе. Проведите анализ: чем отличается выбранная архитектура от других? В чём плюсы и минусы данной архитектуры? Какие могут возникнуть трудности при её применении на практике?

1. Продолжая тему GANов, начнем с выбора набора данных для работы. Один из популярных наборов данных для GANов - это набор данных MNIST, который состоит из черно-белых изображений рукописных цифр размером 28x28 пикселей. Мы можем загрузить этот набор данных, используя следующий код:

```
2. import tensorflow as tf
3. from tensorflow.keras.datasets import mnist
4. from tensorflow.keras.layers import Dense, Flatten, Reshape
5. from tensorflow.keras.layers import LeakyReLU
6. from tensorflow.keras.models import Sequential
7. from tensorflow.keras.optimizers import Adam
8. import numpy as np
9. import matplotlib.pyplot as plt
10.
11.# Загрузка набора данных MNIST
12.(x_train, _), (_, _) = mnist.load_data()
13.
14.# Нормализация изображений
15.x_train = x_train / 127.5 - 1.
16.x_train = np.expand_dims(x_train, axis=3)
17.
18.# Установка размерности вектора шума
19.noise_dim = 100
20.
21.# Модель генератора
22.generator = Sequential()
23.generator.add(Dense(128 * 7 * 7, input_dim=noise_dim))
```

```
generator.add(LeakyReLU(alpha=0.2))
generator.add(Reshape((7, 7, 128)))
generator.add(tf.keras.layers.Conv2DTranspose(128, (4,4), strides=(2,2),
    padding='same'))
generator.add(LeakyReLU(alpha=0.2))
generator.add(tf.keras.layers.Conv2DTranspose(128, (4,4), strides=(2,2),
    padding='same'))
generator.add(LeakyReLU(alpha=0.2))
generator.add(tf.keras.layers.Conv2D(1, (7,7), activation='tanh',
    padding='same'))

# Модель дискриминатора
discriminator = Sequential()
discriminator.add(tf.keras.layers.Conv2D(64, (3,3), strides=(2,2),
    padding='same', input_shape=(28,28,1)))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(tf.keras.layers.Conv2D(128, (3,3), strides=(2,2),
    padding='same'))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(tf.keras.layers.Flatten())
discriminator.add(Dense(1, activation='sigmoid'))

# Комбинированная модель
gan = Sequential([generator, discriminator])

# Установка оптимизатора и компиляция моделей
adam = Adam(lr=0.0002, beta_1=0.5)
discriminator.compile(loss='binary_crossentropy', optimizer=adam,
    metrics=['accuracy'])
discriminator.trainable = False
gan.compile(loss='binary_crossentropy', optimizer=adam)

# Обучение GANa
epochs = 100
batch_size = 128
steps_per_epoch = x_train.shape[0] // batch_size

for epoch in range(epochs):
    for step in range(steps_per_epoch):
        # Обучение дискриминатора
```

```
        real_images = x_train[np.random.randint(0, x_train.shape[0], size=batch_size)]
        noise = np.random.normal(0, 1, size=(batch_size, noise_dim))
        fake_images = generator.predict(noise)
        x = np.concatenate((real_images, fake_images))
        y = np.zeros(2 * batch_size)
        y[:batch_size] = 1
        discriminator.trainable = True
        d_loss = discriminator.train_on_batch(x, y)

        # Обучение генератора
        noise = np.random.normal(0, 1, size=(batch_size, noise_dim))
        y = np.ones(batch_size)
        discriminator.trainable = False
        g_loss = gan.train_on_batch(noise, y)

    # Вывод потерь
    print(f"Epoch {epoch+1}, Discriminator Loss: {d_loss[0]}, Generator Loss: {g_loss}")

    # Сохранение сгенерированных изображений каждые 10 эпох
    if epoch % 10 == 0:
        noise = np.random.normal(0, 1, size=(25, noise_dim))
        generated_images = generator.predict(noise)
        fig, axs = plt.subplots(5, 5)
        count = 0
        for i in range(5):
            for j in range(5):
                axs[i,j].imshow(generated_images[count, :, :, 0], cmap='gray')
                axs[i,j].axis('off')
                count += 1
        plt.show()
```

В этом коде мы сначала загружаем и нормализуем набор данных MNIST. Затем мы определяем модели генератора и дискриминатора и объединяем их в GAN. Мы устанавливаем оптимизатор и компилируем модели, а затем обучаем GAN на заданное количество эпох. Во время обучения мы чередуем обучение дискриминатора и генератора, используя реальные и сгенерированные изображения. Наконец, мы сохраняем и отображаем сгенерированные изображения каждые 10 эпох. Обратите внимание, что качество

сгенерированных изображений будет улучшаться по мере продолжения обучения GANa.
В целом, GANы - это мощный инструмент для генерации реалистичных синтетических данных, и их можно применять в широком спектре приложений, таких как генерация изображений и текстов. Однако обучение GANов может быть сложным, так как они требуют тщательного подбора гиперпараметров и могут быть чувствительны к выбору набора данных и архитектуры модели. Кроме того, режимное затухание и нестабильность во время обучения могут быть распространенными проблемами при работе с GANами.

89.

90. Обзор научной работы: "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"

EfficientNet - это архитектура сверточных нейронных сетей, которая была представлена в 2019 году и получила много внимания в сообществе исследователей машинного обучения. Она была разработана для решения проблемы масштабирования моделей, которая состоит в том, что увеличение размера сети часто приводит к увеличению количества параметров и вычислительной сложности, что затрудняет их использование на ресурсно ограниченных устройствах.
Архитектура EfficientNet была создана путем комбинации трех методов масштабирования: изменения глубины, ширины и разрешения изображения. Она использует сеть свертки с множеством блоков, в которых используется свертка 3x3, а также блоки с различными размерами ядер. Все блоки соединены вместе, чтобы сформировать эффективную архитектуру, которая обеспечивает высокую точность при меньшем количестве параметров.
Одним из главных достоинств EfficientNet является ее эффективность, то есть высокая точность при использовании меньшего числа параметров и меньшего количества вычислительных ресурсов. Это делает ее привлекательной для использования на устройствах с ограниченными ресурсами, таких как мобильные устройства.
Однако, при применении EfficientNet могут возникнуть трудности с настройкой гиперпараметров из-за большого количества блоков и слоев. Также, данная архитектура может потребовать большого количества данных для обучения и дополнительного времени на обучение и настройку.
В целом, EfficientNet является эффективной архитектурой нейронных сетей, которая может быть применена в различных задачах

компьютерного зрения и может быть особенно полезна для решения проблемы масштабирования моделей.