Генератор MuseGAN

Подобно всем генеративно-состязательным сетям (GAN), MuseGAN состоит из генератора и критика. Генератор пытается одурачить критика своими музыкальными творениями, а критик старается предотвратить это, пытаясь отличить поддельные хоралы Баха, созданные генератором, от настоящих.

Особенность сети MuseGAN в том, что ее генератор принимает на входе не один вектор шума, а целых четыре, которые соответствуют четырем составляющим — аккордам, стилю, мелодии и дорожке. Управляя каждым из этих входов, можно менять высокоуровневые свойства сгенерированной музыки. Обобщенное представление генератора показано на рис. 11.15.

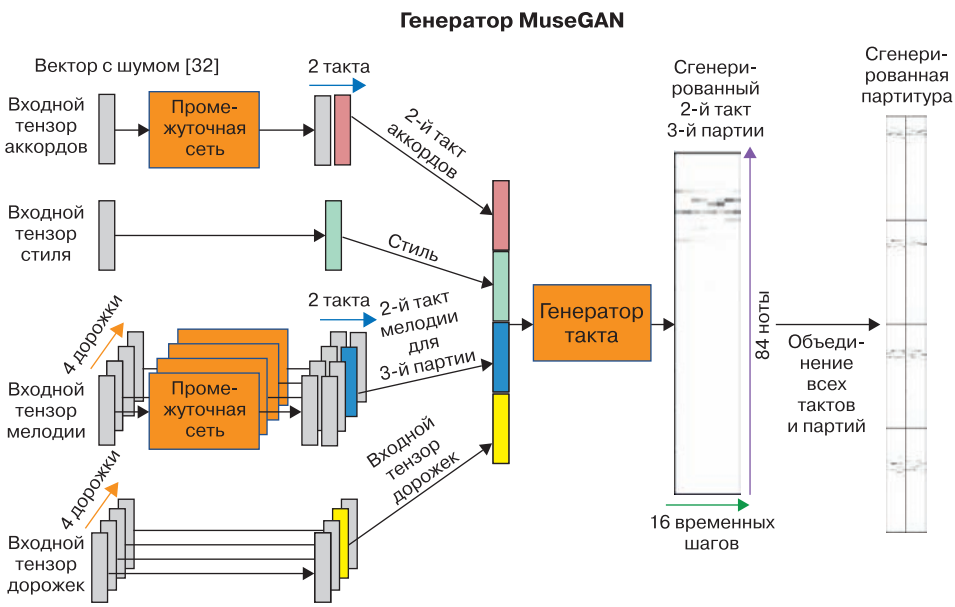

Рис. 11.15. Обобщенная схема генератора MuseGAN

На диаграмме видно, как входные тензоры аккордов и мелодии сначала передаются в *промежуточную сеть*, выводящую тензор, одно из измерений которого равно количеству сгенерированных тактов. Благодаря этому нет необходимости растягивать во времени входные тензоры стилей и дорожек, так как они остаются постоянными на протяжении всего произведения.

Затем, чтобы сгенерировать конкретный такт для конкретной партии, объединяются соответствующие векторы аккордов, стиля, мелодии и дорожек. Они формируют более длинный вектор, передаваемый в генератор тактов, который

в конечном итоге выводит указанный такт для указанной партии. Объединением сгенерированных тактов для всех партий создается партитура, которую критик может сравнить с настоящими.

Сначала посмотрим, как построить промежуточную сеть.

Промежуточная сеть

Задача промежуточной сети (нейронной сети, состоящей из слоев обратной свертки) — преобразовать один входной вектор шума длиной `Z_DIM = 32` в другой вектор шума для каждого такта (также длиной 32). Код создания этой сети показан в примере 11.6.

Пример 11.6. Конструирование промежуточной сети

```
def conv_t(x, f, k, s, a, p, bn):
    x = layers.Conv2DTranspose(
                filters = f
                , kernel_size = k
                , padding = p
                , strides = s
                , kernel_initializer = self.weight_init
                )(x)

    if bn:
        x = layers.BatchNormalization(momentum = 0.9)(x)

    x = layers.Activation(a)(x)
    return x

def TemporalNetwork(self):
    input_layer = layers.Input(shape=(Z_DIM,), name='temporal_input') ❶
    x = layers.Reshape([1,1,Z_DIM])(input_layer) ❷
    x = conv_t(
        x, f=1024, k=(2,1), s=(1,1), a= 'relu', p = 'valid', bn = True
    ) ❸
    x = conv_t(
        x, f=Z_DIM, k=(N_BARS - 1,1), s=(1,1), a = 'relu', p = 'valid', bn = True
    )
    output_layer = layers.Reshape([N_BARS, Z_DIM])(x) ❹
    return models.Model(input_layer, output_layer)
```

❶ На входе промежуточная сеть принимает вектор длиной 32 (`Z_DIM`).

❷ Этот вектор преобразуется в тензор с формой 1× 1 с 32 каналами, чтобы к нему можно было применить операцию обратной двумерной свертки.

❸ Слои `Conv2DTranspose` увеличивают размер тензора вдоль одной из осей, чтобы он получил длину, равную `N_BARS`.

❹ С помощью слоя `Reshape` удаляются ставшие ненужными дополнительные измерения.

Мы используем сверточные операции вместо передачи двух независимых векторов тактов с тем, чтобы сеть знала, что такты должны следовать друг за другом. Применение нейронной сети для расширения входного вектора вдоль оси времени позволяет модели узнать, что музыка исполняется тактами и каждый следующий такт не является полностью независимым от предыдущего.

Аккорды, стиль, мелодия и дорожки

Теперь рассмотрим поближе четыре входных вектора, которые передаются в генератор.

Аккорды

Входной тензор аккордов имеет длину 32 (`Z_DIM`). Задача этого вектора — обеспечить развитие музыки во времени, общее для всех дорожек, поэтому мы используем `TemporalNetwork` для преобразования этого единственного вектора в другой скрытый вектор для каждого такта. Обратите внимание: несмотря на свое имя `chords_input`, этот тензор может контролировать все, что касается изменения музыки с началом нового такта, например общий ритмический стиль, без привязки к какой-либо конкретной партии.

Стиль

Входной тензор стиля тоже имеет длину `Z_DIM`. Он передается в генератор тактов без каких-либо изменений, потому что для него не существует понятий партий и тактов. Другими словами, генератор тактов должен использовать этот тензор для согласования тактов и партий.

Мелодия

Входной тензор мелодии — это массив с формой`[N_TRACKS, Z_DIM]`, и для каждой партии модели передается случайный вектор шума длиной`Z_DIM`. Каждый из этих векторов передается через свою копию промежуточной сети, описанной ранее. Обратите внимание на то, что веса в этих копиях разные, поэтому на выходе для каждой партии в каждом такте получается вектор длиной`Z_DIM`. При подобной организации генератор тактов может использовать этот вектор для точной настройки содержимого каждого отдельного такта в каждой партии.

Дорожки

Входной тензор дорожек тоже является массивом с формой `[N_TRACKS, Z_DIM]`, то есть массивом, содержащим вектор со случайным шумом и длиной `Z_DIM` для каждой партии. В отличие от тензора мелодии, он не преобразуется промежуточной сетью, а передается непосредственно в генератор тактов, как и тензор стиля. Но в отличие от тензора стиля, каждой партии соответствует отдельная входная дорожка. Таким образом, эти векторы можно использовать для независимой настройки каждой партии.

Мы можем обобщить назначение каждого компонента генератора MuseGAN, как показано в табл. 11.1.

Таблица 11.1. Компоненты генератора MuseGAN

Параметр	Результат разный для каждого такта?	Результат разный для каждой дорожки?
Стиль	X	X
Дорожки	X	✓
Аккорды	✓	X
Мелодия	✓	✓

Последняя часть генератора MuseGAN — генератор тактов. Давайте посмотрим, как можно использовать его для объединения результатов, выдаваемых в виде векторов аккордов, стиля, мелодии и дорожек.

Генератор тактов

Генератор тактов получает четыре скрытых вектора — аккордов, стиля, мелодии и дорожек. Они объединяются для получения входного вектора длиной `4 * Z_DIM`. Выходные данные имеют вид перфорированной ленты для одного такта и одной дорожки, то есть представлены тензором с формой `[1, N_STEPS_PER_BAR, N_PITCHES, 1]`.

Генератор тактов — это нейронная сеть, которая использует слои обратной свертки для увеличения размерностей, соответствующих времени и нотам. Мы создадим для каждой партии свой генератор тактов, и все генераторы будут иметь свои веса. Код в примере 11.7 показывает, как с помощью Keras построить генератор тактов.

Пример 11.7. Конструирование генератора тактов

```
def BarGenerator(self):

    input_layer = layers.Input(shape=(Z_DIM * 4,), name='bar_generator_input') ❶

    x = layers.Dense(1024)(input_layer) ❷
    x = layers.BatchNormalization(momentum = 0.9)(x)
    x = layers.Activation('relu')(x)
    x = layers.Reshape([2,1,512])(x)

    x = conv_t(x, f=512, k=(2,1), s=(2,1), a= 'relu', p = 'same', bn = True) ❸
    x = conv_t(x, f=256, k=(2,1), s=(2,1), a= 'relu', p = 'same', bn = True)
    x = conv_t(x, f=256, k=(2,1), s=(2,1), a= 'relu', p = 'same', bn = True)
    x = conv_t(x, f=256, k=(1,7), s=(1,7), a= 'relu', p = 'same',bn = True) ❹
    x = conv_t(x, f=1, k=(1,12), s=(1,12), a= 'tanh', p = 'same', bn = False) ❺
```

```
    output_layer = layers.Reshape([1, N_STEPS_PER_BAR, N_PITCHES, 1])(x) ❻

    return models.Model(input_layer, output_layer)
```

❶ На вход генератора тактов подается вектор длиной `4 * Z_DIM`.

❷ Пропустив тензор через слой `Dense`, мы подготавливаем его к операции обратной свертки, изменяя форму.

❸ Сначала тензор расширяется по оси временных шагов...

❹ ...А затем по оси нот.

❺ В последнем слое применяется функция активации tanh, потому что далее для обучения сети используется механизм WGAN-GP, требующий активации tanh.

❻ Добавлением двух измерений размером 1 изменяется форма тензора, чтобы он был готов к объединению с другими тактами и партиями.

Объединяем все вместе

В целом генератор MuseGAN принимает четыре входных тензора (аккорды, стиль, мелодия и дорожки) и преобразует их в партитуру из нескольких тактов и партий. В примере 11.8 показан код, создающий генератор MuseGAN.

Пример 11.8. Конструирование генератора MuseGAN

```
def Generator():
    chords_input = layers.Input(shape=(Z_DIM,), name='chords_input') ❶
    style_input = layers.Input(shape=(Z_DIM,), name='style_input')
    melody_input = layers.Input(shape=(N_TRACKS, Z_DIM), name='melody_input')
    groove_input = layers.Input(shape=(N_TRACKS, Z_DIM), name='groove_input')

    chords_tempNetwork = TemporalNetwork() ❷
    chords_over_time = chords_tempNetwork(chords_input)

    melody_over_time = [None] * N_TRACKS
    melody_tempNetwork = [None] * N_TRACKS
    for track in range(N_TRACKS):
        melody_tempNetwork[track] = TemporalNetwork() ❸
        melody_track = layers.Lambda(lambda x, track = track: x[:,track,:])(
            melody_input
        )
        melody_over_time[track] = melody_tempNetwork[track](melody_track)

    barGen = [None] * N_TRACKS
    for track in range(N_TRACKS):
        barGen[track] = BarGenerator() ❹
```

```
    bars_output = [None] * N_BARS
    c = [None] * N_BARS
    for bar in range(N_BARS): ❺
        track_output = [None] * N_TRACKS

        c[bar] = layers.Lambda(lambda x, bar = bar: x[:,bar,:])(chords_over_time)
        s = style_input

        for track in range(N_TRACKS):

            m = layers.Lambda(lambda x, bar = bar: x[:,bar,:])(
                melody_over_time[track]
            )
            g = layers.Lambda(lambda x, track = track: x[:,track,:])(
                groove_input
            )

            z_input = layers.Concatenate(
                axis = 1, name = 'total_input_bar_{}_track_{}'.format(bar, track)
            )([c[bar],s,m,g])

            track_output[track] = barGen[track](z_input)

        bars_output[bar] = layers.Concatenate(axis = -1)(track_output)

    generator_output = layers.Concatenate(axis = 1, name = 'concat_bars')(
        bars_output
    ) ❻

    return models.Model(
        [chords_input, style_input, melody_input, groove_input], generator_output
    ) ❼

generator = Generator()
```

❶ Определение входных слоев генератора.

❷ Передать входной тензор аккордов через промежуточную сеть.

❸ Передать входной тензор мелодии через промежуточную сеть.

❹ Создать независимую сеть генератора тактов для каждой партии.

❺ Выполнить обход партий и тактов и создать генерируемый такт для каждой комбинации.

❻ Объединить все вместе и сформировать общий выходной тензор.

❼ Модель MuseGAN принимает четыре тензора с шумом и выводит сгенерированную партитуру.

Критик MuseGAN

По сравнению с генератором критик имеет более простую архитектуру, что довольно характерно для сетей GAN. Он пытается отличить партитуры, созданные генератором, от фрагментов настоящих хоралов Баха. Это сверточная нейронная сеть, состоящая в основном из слоев `Conv3D`, которые свертывают партитуру в один выходной прогноз.

Слои Conv3D

До сих пор мы работали только со слоями Conv2D, применяя их к изображениям, имеющим три измерения: ширину, высоту и каналы цвета. Здесь мы должны использовать слои Conv3D. Они подобны слоям Conv2D, но принимают четырехмерные тензоры (N_BARS, N_STEPS_PER_BAR, N_PITCHES, N_TRACKS).

Кроме того, мы не применяем слои пакетной нормализации в критике, потому что для обучения GAN будем задействовать инфраструктуру WGAN-GP, которая запрещает нормализацию.

В примере 11.9 показан код, конструирующий критика с помощью Keras.

Пример 11.9. Конструирование критика MuseGAN

```
def conv(x, f, k, s, p):
    x = layers.Conv3D(filters = f
                , kernel_size = k
                , padding = p
                , strides = s
                , kernel_initializer = initializer
                )(x)
    x = layers.LeakyReLU()(x)
    return x

def Critic():
    critic_input = layers.Input(
        shape=(N_BARS, N_STEPS_PER_BAR, N_PITCHES, N_TRACKS),
        name='critic_input'
    ) ❶

    x = critic_input
    x = conv(x, f=128, k = (2,1,1), s = (1,1,1), p = 'valid') ❷
    x = conv(x, f=128, k = (N_BARS - 1,1,1), s = (1,1,1), p = 'valid')

    x = conv(x, f=128, k = (1,1,12), s = (1,1,12), p = 'same') ❸
    x = conv(x, f=128, k = (1,1,7), s = (1,1,7), p = 'same')
```

```
    x = conv(x, f=128, k = (1,2,1), s = (1,2,1), p = 'same') ❹
    x = conv(x, f=128, k = (1,2,1), s = (1,2,1), p = 'same')
    x = conv(x, f=256, k = (1,4,1), s = (1,2,1), p = 'same')
    x = conv(x, f=512, k = (1,3,1), s = (1,2,1), p = 'same')

    x = layers.Flatten()(x)

    x = layers.Dense(1024, kernel_initializer = initializer)(x)
    x = layers.LeakyReLU()(x)

    critic_output = layers.Dense(
        1, activation=None, kernel_initializer = initializer
    )(x) ❺

    return models.Model(critic_input, critic_output)

critic = Critic()
```

❶ На вход критика подается массив партитуры с формой `[N_BARS, N_STEPS_PER_BAR, N_PITCHES, N_TRACKS]`.

❷ Сначала выполняется свертка тензора вдоль оси тактов. Для работы с четырехмерными тензорами используем слои `Conv3D`.

❸ Далее выполняется свертка тензора вдоль оси нот.

❹ Наконец, выполняется свертка тензора вдоль оси временных шагов.

❺ Результат формируется слоем `Dense` с единственным узлом без функции активации, как того требует механизм WGAN-GP.

Анализ сети MuseGAN

Можете немного поэкспериментировать со своей сетью MuseGAN, генерируя партитуры и затем настраивая некоторые параметры входного шума, чтобы увидеть, как они влияют на результат.

Результатом работы генератора является массив значений в диапазоне [–1, 1] (из-за функции активации tanh в последнем слое). Чтобы преобразовать его в единственную ноту в каждой партии, для каждого временного шага выбирается нота с максимальным значением из всех 84 нот. В оригинальной статье с описанием MuseGAN авторы используют пороговое значение 0, поскольку каждая партия может содержать несколько нот, но в случае с хоралами Баха можно просто взять максимальное значение, чтобы гарантировать ровно одну ноту на каждом шаге в каждой партии.

На рис. 11.16 показана партитура, сгенерированная моделью из векторов со случайным нормально распределенным шумом *(вверху слева)*. Мы можем найти ближайшую (по евклидову расстоянию) партитуру в наборе данных и убедиться в том, что сгенерированная партитура не является копией музыкальной пьесы, существующей в наборе данных, — ближайшая партитура показана чуть ниже, и, как нетрудно заметить, она не похожа на сгенерированную партитуру.

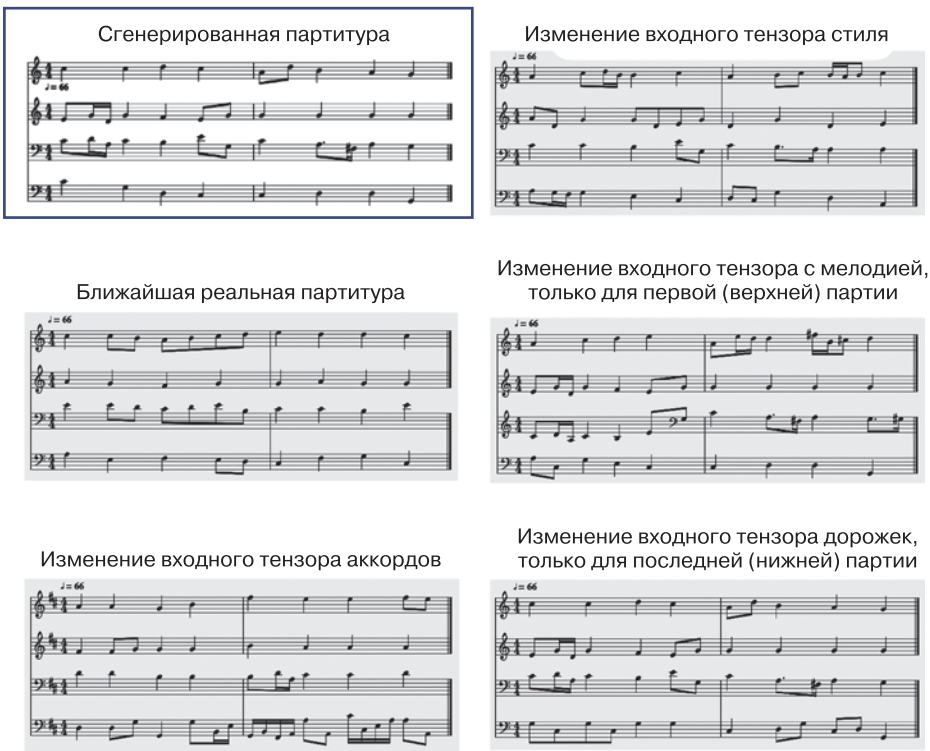

Рис. 11.16. Партитура, сгенерированная сетью MuseGAN, ближайшая к ней реальная партитура из обучающих данных и примеры влияния попыток изменить входной шум на сгенерированную партитуру

Теперь поэкспериментируем с входным шумом, чтобы улучшить сгенерированную партитуру. Для начала изменим вектор шума (см. рис. 11.16, *внизу*). Все партии изменились, как и ожидалось, и теперь такты имеют разные свойства. Во втором такте базовая партия более динамична, а верхняя имеет более высокую тональность, чем в первом такте. Это связано с тем, что скрытые векторы, влияющие на два такта, различны, поскольку входной вектор аккордов прошел через промежуточную сеть.

При изменении стиля *(вверху справа)* оба такта меняются одинаково. Они не сильно различаются по стилю, но весь фрагмент изменился по сравнению с первоначально сгенерированной партитурой (то есть для настройки всех дорожек и тактов используется один и тот же скрытый вектор).

Мы можем также изменять отдельные партии через входные тензоры мелодии и дорожек. На рис. 11.16 можно видеть эффект изменения входного шума в начальном тензоре мелодии только для верхней музыкальной партии. Она изменилась довольно существенно, а все прочие остались неизменными. Можно заметить и изменение ритма между тактами в верхней партии: второй такт получился более динамичным — он содержит более короткие ноты, чем первый.

Наконец, *внизу справа* на рис. 11.16 показана партитура, сгенерированная после изменения входного тензора дорожек только для нижней партии. И снова изменилась только она, все остальные части неизменны. Более того, в целом такты в нижней партии сохранили сходство между собой, как и следовало ожидать.

Этот пример показывает, как, изменяя входные параметры, можно влиять непосредственно на высокоуровневые признаки сгенерированной музыкальной последовательности. Все происходит почти так же, как в предыдущих главах, где, регулируя скрытые векторы в вариационных автокодировщиках и генеративно-состязательных сетях, мы изменяли внешний вид сгенерированных образов. Один из недостатков модели — необходимость заранее определить количество генерируемых тактов. Чтобы избавиться от него, авторы предложили расширение, реализующее передачу предыдущих тактов на вход, что позволяет модели генерировать длинные партитуры за счет непрерывной передачи самых последних сгенерированных тактов обратно в модель в качестве дополнительных входных данных.

Резюме

Итак, мы рассмотрели два типа моделей для генерирования музыки: трансформер и MuseGAN. Своим дизайном трансформер напоминает сети для генерирования текста (см. главу 9). Музыка и текст имеют много общего, и часто для их генерирования можно использовать одинаковые методы. Мы расширили модель трансформера, добавив в нее входные и выходные потоки нот и их длительностей, и увидели, как модель смогла выявить такие понятия, как октавы и тональности, научившись точно имитировать музыку Баха.

Затем мы посмотрели, как адаптировать процесс кодирования для создания полифонической музыки. Кодирование сеткой создает представление партитуры

в виде перфорированной ленты для механического пианино, позволяя обучать трансформер на одной последовательности кодов, описывающих каждую ноту в каждой партии через дискретные, равноотстоящие друг от друга интервалы времени. Кодирование на основе событий позволяет описать сразу несколько музыкальных партий с помощью единого потока инструкций. Оба метода имеют свои преимущества и недостатки: успех или неудача подхода к созданию музыки на основе модели трансформера во многом зависит от выбора метода кодирования.

Вы также узнали, что для генерирования последовательных данных не обязательно применять последовательный подход — сеть MuseGAN создает полифонические музыкальные партитуры с несколькими партиями, используя сверточные слои и обрабатывая партитуру как своеобразное изображение, в котором партии играют роль каналов изображения. Новизна MuseGAN заключается в том, что четыре входных тензора с шумом (аккорды, стиль, мелодия и дорожки) организованы так, чтобы с их помощью можно было управлять высокоуровневыми свойствами музыки. Да, базовая гармония все еще не так совершенна или разнообразна, как в произведениях Баха, но эта сеть является хорошей попыткой выполнить чрезвычайно сложную задачу и подчеркивает способность генеративно-состязательных сетей решать широкий спектр задач.