Отчет

Волков Константин Денисович

May 13, 2024

Тема задания: Создание глубокой сверточной состязательной генеративной сети для генерации изображений рукописных цифр **Выполнил**: Волков К. Д., группа 21.Б04-пу

1 Постановка задачи

Реализовать нейронную сеть архитектуры DCGAN (Deep convectional generative adversarial networks) для задачи генерации изображений рукописных цифр. Датасет для обучения - MNIST. Предложенная архитектуры создана на основе этой статьи.

2 Описание данных

MNIST - это набор данных из 60 000 квадратных изображений размером 28×28 пикселей, содержащих написанные от руки однозначные цифры от 0 до 9. Изображения представлены в чернобелом формате. Предобработка данных не проводилась

3 Архитектура DCGAN

DCGAN - Класс генеративных состязательных сетей, в которых генератор и дискриминатор являются глубокими сверточными сетями. Обучение происходит без учителя, методом обратного распространения ошибки. Авторы статьи дают следующие рекомендации по архитектуре для стабильных глубоких сверточных сетей GAN

- Используйте сверточные слои (дискриминатор) и сверточные слои с дробным шагом (генератор).
- Используйте Batch Normalization после каждого слоя свертки, кроме последнего, как в генераторе, так и в дискриминаторе. Это позволяет стандартизировать выход, чтобы он имел нулевое среднее значение и единичную дисперсию.
- Удалите полностью подключенные скрытые слои для более глубоких архитектур.
- Используйте активацию ReLU в генераторе для всех слоев, кроме выходных, которые используют Tanh.
- Используйте активацию LeakyReLU в дискриминаторе для всех слоев.

Опишем созданную на основе этих рекомендауций реализацию:

1. Генератор. Вход - случайным шум размерности $[1 \times 100]$, первый слой - полносвязнный, он проецирует наш шум на пространство размерности $[7 \times 7 \times 256]$, после этого идут два сверточных слоя с дробым шагом, параметры слоев взяты из статьи -
   $filters = 128(64)$
   $kernalsize = 5 \times 5$
   $strides = 2 \times 2$
   padding отсуствует
   Функции активации - ReLU (перед активацией проводится Batch Normalization).
   Размерность данных после прохождения этих слоев - $[28 \times 28 \times 64]$, поэтому последний слой свертки, уже с целым шагом, имеет параметры -

$filters = 1$
$kernalsize = 5 \times 5$
$strides = 1 \times 1$
padding отсутствует
Функция активации - tanh
Обучение происходит по следующей схеме: функция потери - BCE, чтобы генератор учился обманывать дискриминатор, мы настраиваем его весы с целью максимизировать вероятность того, что дискриминатор неправильно классифицирует поддельные изображения как реальные, т.е. максимизируем $BCE(D(G(noise))$
Полное описание генератора

Layer (type)	Output Shape	Param #
dense_13 (Dense)	(None, 12544)	1266944
batch_normalization_27 (BatchNormalization)	(None, 12544)	50176
re_lu_15 (ReLU)	(None, 12544)	0
reshape_7 (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose_8 (Conv2DTranspose)	(None, 14, 14, 128)	819328
batch_normalization_28 (BatchNormalization)	(None, 14, 14, 128)	512
re_lu_16 (ReLU)	(None, 14, 14, 128)	0
conv2d_transpose_9 (Conv2DTranspose)	(None, 28, 28, 64)	204864
batch_normalization_29 (BatchNormalization)	(None, 28, 28, 64)	256
re_lu_17 (ReLU)	(None, 28, 28, 64)	0
conv2d_16 (Conv2D)	(None, 28, 28, 1)	1601

2. Дискриминатор. Вход - вектор $[28 \times 28 \times 1]$, первые два слоя - свертка с параметрами -

$filters = 128(64)$
$kernalsize = 5 \times 5$
$strides = 2 \times 2$
padding отсутствует

Функции активации - $LeakyReLU(alpha = 0.3)$(перед активацией проводится Batch Normalization)
После этого идет выравнивание, и полносвязный слой с размером выхода - 1, функция активации - сигмоида. Схема обучения: функция потери - BCE, по своей сути это задача классификации, на каждом шагу обучения мы генерируем случайные изображения размерности $batchsize$ и помечаем их меткой 0, настоящие изображения из батча - 1 , задача дискриминатора, максимизировать вероятность правильного определения класса изображения.
Полное описание генератора

Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 14, 14, 64)	1664
batch_normalization_30 (BatchNormalization)	(None, 14, 14, 64)	256
leaky_re_lu_12 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_18 (Conv2D)	(None, 7, 7, 128)	204928
batch_normalization_31 (BatchNormalization)	(None, 7, 7, 128)	512
leaky_re_lu_13 (LeakyReLU)	(None, 7, 7, 128)	0
flatten_6 (Flatten)	(None, 6272)	0
dense_14 (Dense)	(None, 1)	6273

В качестве оптимизаторов авторы статьи предлагают использовать $Adam(learningrate = 0.0003, beta_1 = 0.5)$.Все веса инициализируются на основе нормального распределения с нулевым центром и стандартным отклонением 0.02, $batchsize = 32$. Код написан на Python, с использованием TensorFlow, Kedas, MatPlotLib.

4 Модификации архитектуры, относительно предложенной в статье

Так как предложенная в статье архитектуры создавалась под куда больший класс задач, я уменьшил число слоев в генераторе в два раза, а параметры для дискриминатора подбирались эксперементальным путем, исходя из

двух соображений, первое - число слоев свертки должно быть таким же как в генераторе (во всяком случае так делают в найденных мною реализациях),второе - дискриминатор должен быть "проще", чем генератор. Также я поменял предложенное значение параметра *alpha* в LeakyReLU с 0.2 на 0.3 и *learningrate* с 0.0002 на 0.0003, эксперементальным путем выяснил, что с такими значениями сеть обучается быстрее.

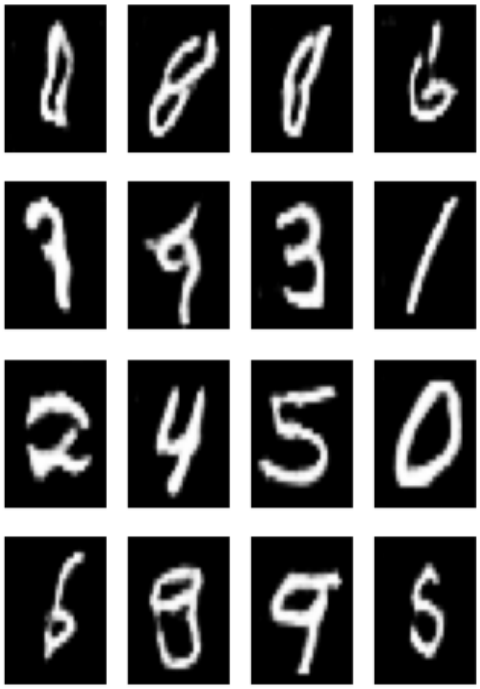
5 Результаты

Результаты обучения после:

1-ой эпохи

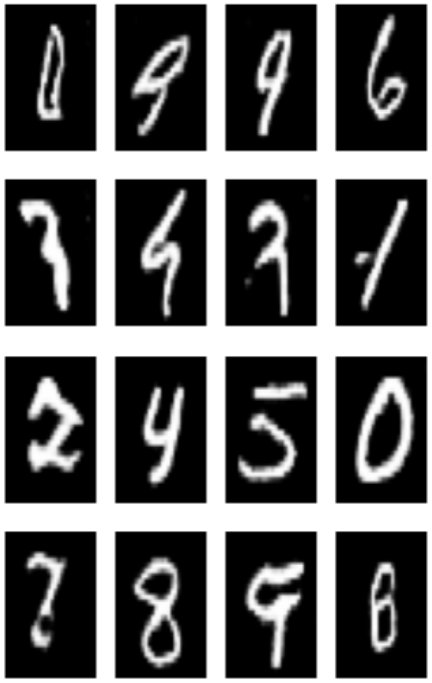
1875/1875 [==============================] - 68s 33ms/step - d_loss: 0.2040 - g_loss: 1.8327

5-ой эпохи



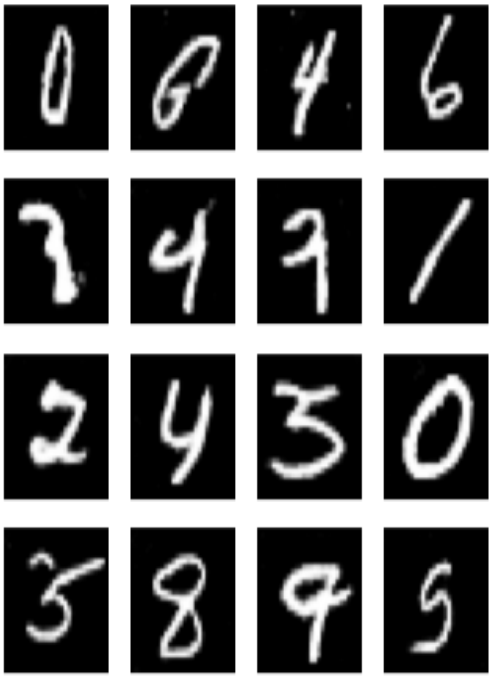
1875/1875 [==============================] - 62s 33ms/step - d_loss: 0.6227 - g_loss: 0.9230

10-ой эпохи



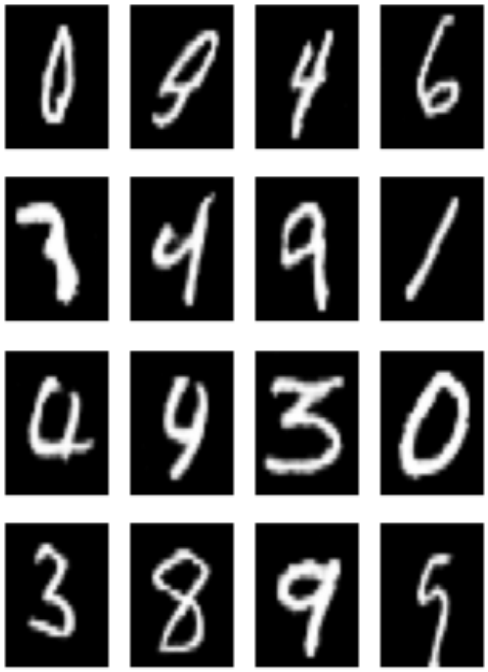
1875/1875 [==============================] - 63s 34ms/step - d_loss: 0.6709 - g_loss: 0.8162

20-ой эпохи



```
1875/1875 [==============================] - 63s 34ms/step - d_loss: 0.6944 - g_loss: 0.7377
```

35-ой эпохи



```
1875/1875 [==============================] - 63s 34ms/step - d_loss: 0.6953 - g_loss: 0.7279
```

Вывод: Предложенная архитектура показывает удобоваримые результаты, но не лишена проблемы нестабильности обучения т.е. увеличение эпох не обязательно приведет к улучшению результаты, так как в попытках улучшить результат, секта в некоторых случаях может ухудшить качество изображния, стараясь заменить один образ на другой. (например 0 на 8)