**ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНЫЙ ПОДХОД**

Вспомним прошлые занятия, когда мы говорили об атаках на нейронные сети. Как мы это выполняли? Придумали критерий, который оптимизировали за счет изменения входа в нейронную сеть, но не ее параметров. Что если необходимые изменения входа довольно сложные, нам не хватит простых способов их изменения. А давайте заставим другую нейронную сеть делать такие изменения! Получим две связанные нейронные сети, одна пытается распознать входы, а другая пытается обмануть первую. Но что мы все о плохом, об атаках, давайте используем такую же идею для созидания.

Сделаем нейронную сеть, которая пытается различить (дискриминировать) правдоподобные и неправдоподобные входы, например, изображения лиц людей, одни картинки похожи на лица, другие нет. Сделаем другую сеть, которая могла бы создавать (генерировать) разные выходы (пусть изображения) и будем обучать ее обмануть первую сеть, т.е. сгенерировать такие выходы, которые первая сеть примет за правдоподобные.

Тогда вторая сеть научится создавать искусственные правдоподобные выходы, что можно с пользой использовать.

Такую идею называют генеративно-состязательным подходом, а совокупность сетей, которые используются – генеративно-состязательными сетями. Почему «состязательные»?

Потому что обе сети пытаются превзойти друг друга, одна пытается отличить действительные, не сгенерированные входы, а вторая пытается сгенерировать выходы, которые первая приняла бы за действительные, сети состязаются между собой.

На рисунке 1 показана общая схема работы генеративно-состязатальной сети (generative-adversarial network, GAN).

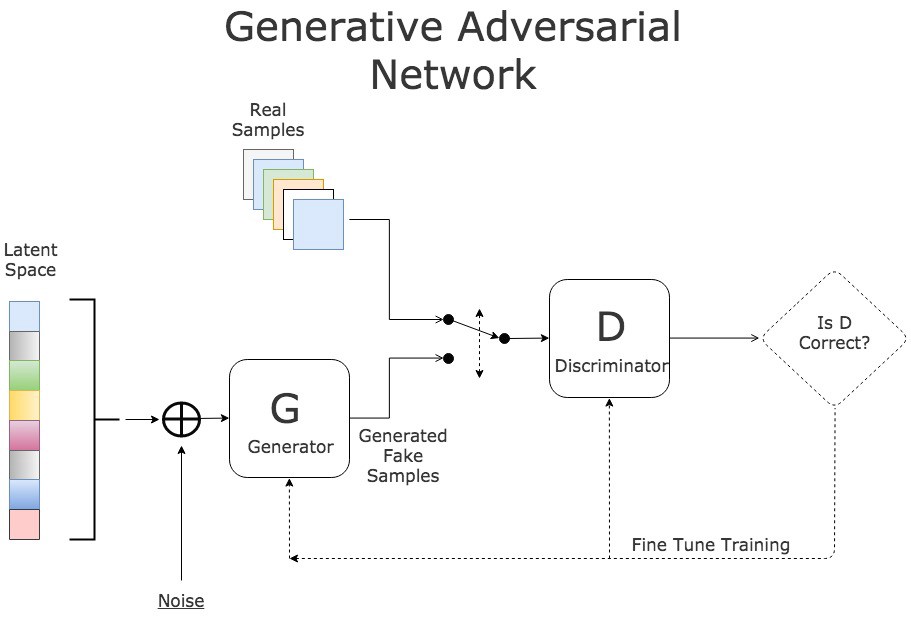
Есть сеть-дискриминатор D, которая принимает некоторый входной образ и обучается тому, чтобы отличить действительные образы, заданные примерами из обучающего набора данных, от сгенерированных. Дискриминатор здесь, по сути, двуклассовый классификатор, различает только действительные\сгенерированные образы. На вход дискриминатора могут поступать образы из обучающего набора или сгенерированные образы. С обучением дискриминатора все более-менее ясно, для каждого входа мы знаем, он взят из обучающего набора или сгенерирован, поэтому можем посчитать ошибку работы дискриминатора и обучать его, например, градиентными методами.

Генерацией образов занимается другая сеть – генератор G. Эта сеть должна быть способна выдавать выходы, которые соответствуют действительным образам. Чтобы генератор мог выдавать разные выходы, у него есть некоторый вход, который определяет, что именно будет выдавать генератор. Его назвали латентными переменными, они могут иметь физический смысл, а могут и не иметь[[1]](#footnote-1), для случайности ко входу добавляют небольшой шум. Генератор для некоторого входа создает соответствующий образ на выходе. Это может быть абсолютная абракадабра, ведь генератор ничего не знает о действительных образах. Его нужно обучать. Но вот тут возникает проблема, а как, точней чему, обучать генератор? У нас нет примеров хороших или плохих выходов генератора, мы их-то и хотим получить.

Для обучения генератора мы будем использовать дискриминатор! Дискриминатор знает, выход генератора похож или не похож на действительный образ, ведь именно этому он и учился. Мы можем рассчитать ошибку генератора используя дискриминатор, а раз так, то можем и оптимизировать ее, изменяя параметры генератора, но не дискриминатора.

*Можно так представить процесс: генератор это глупый школьник который не знает правильного ответа на вопрос сколько будет два плюс два, а дискриминатор – злой учитель. Школьник говорит «семь», а учитель бьет его указкой и говорит «меньше», школьник говорит «пять», его опять бьют и говорят «меньше», тут школьник говорит «два», а дискриминатор-учитель его лупит со всей дури и орет «больше», продолжаем экзекуцию пока не найдем правильный ответ. И так для всей таблицы умножения.[[2]](#footnote-2)*

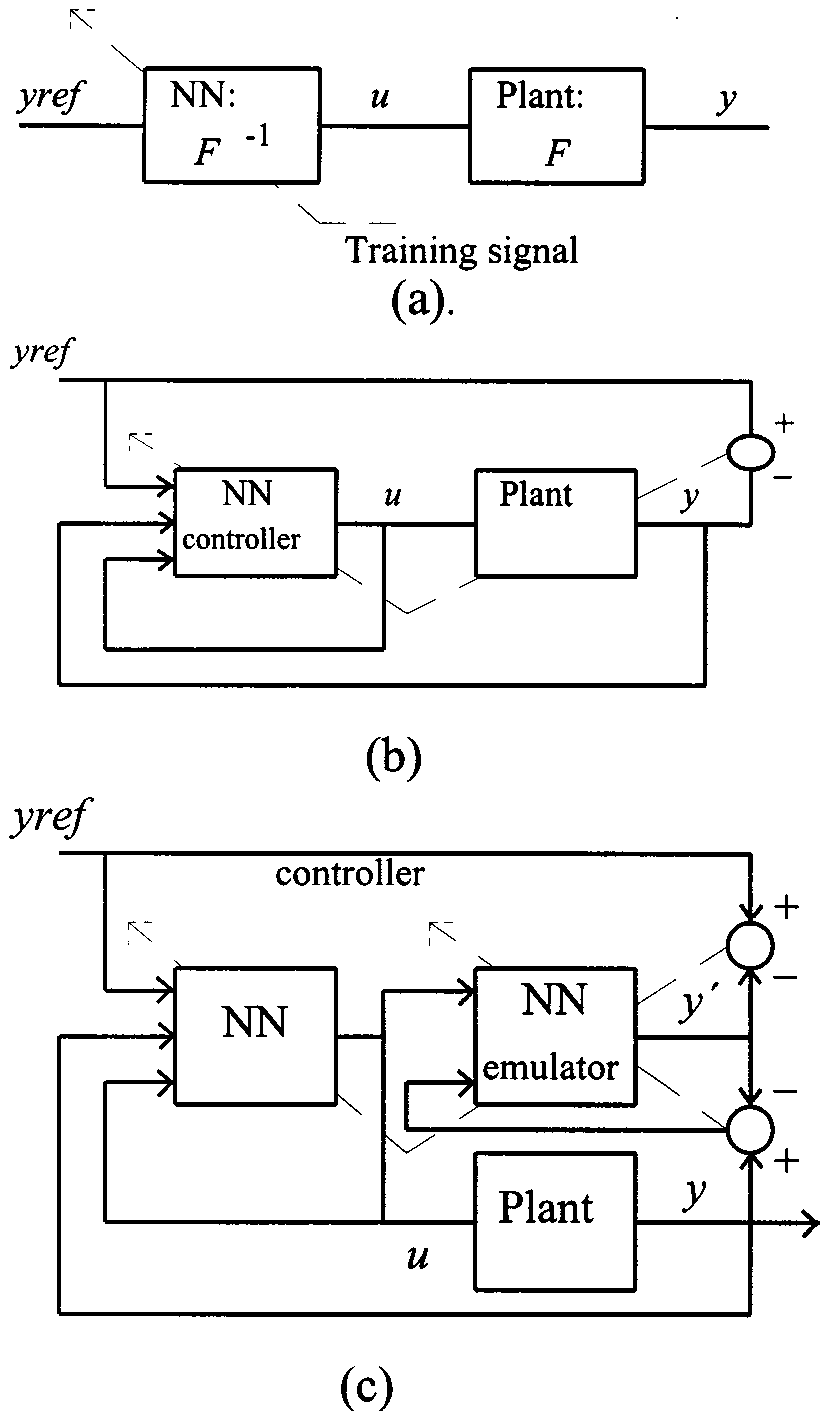
Поскольку дискриминатор это нейронная сеть, то мы можем провести градиент через него в обратную сторону и посчитать градиенты для слоев генератора, значит можем учить и генератор градиентными методами.



Тут важен баланс, если дискриминатор слабый, с трудом отличает действительный и сгенерированные образы, то генератор быстренько научится его обманывать, и прекратит свое обучение, а правдоподобные сгенерированные образы не получим. Если дискриминатор очень сильный, то генератору, особенно в начале обучения, будет нереально обмануть его, и он не сможет найти нужные параметры, дискриминатор на все скажет, что это сгенерированный фейк. Поэтому генератор и дискриминатор обучаются попеременно, зафиксировали коэффициенты одного, обучаем другой, потом наоборот, фиксируем второй, обучаем первый и т.д. И другая проблема может возникнуть, что, если генератор настолько сильный, что обучился выдавать образы из набора действительных примеров, тогда он бессмысленен, такие образы у нас уже есть, зачем их генерировать. Нельзя обучать сети до нулевой ошибки.

Отвлечемся на секунду…

А ничего такой подход не напоминает? Не видели ли мы раньше что-то похожее? Видели! Вспомните о нейроуправлении, у нас был эмулятор и контроллер, контроллер пытался обучиться так, чтобы эмулятор (а вместе с ним и объект управления) выдавал нужную реакцию - схема прямого нейроуправления с прямым эмулятором - и контроллер обучался через эмулятор, рисунок 2. *Все новое это хорошо забытое старое.*



**Рисунок 2. Схема нейроуправления**

Конечно тут много нюансов, все не так просто, как кажется, но это работает. Сегодня уже много разных примеров использования GAN для разных задач, и генерация изображений и текста, и звука, и чего угодно другого, те же атаки на нейронные сети могут использовать GAN. Есть множество модификаций GAN. Я специально не стал приводить здесь формулы, различные функции ошибки для GAN и другие детали – главное идея. Подробности можно посмотреть в литературе по той сети, которую вы выберете для изучения.

Посмотрим на некоторые примеры и демонстрации.

**GAN Lab**

<https://poloclub.github.io/ganlab/>

Онлайн демонстрация пошаговой работы GAN (здесь персептроны) на двумерных данных.

Попробуйте добиться хорошей генерации, или воспользуйтесь предобученными сетями, чтобы посмотреть результат.

PS. Здесь и во многих других работах изображается что в дискриминатор идет два входа, действительные данные и сгенерированные, это просто способ отображения, на самом деле поступает вход ИЛИ от сгенерированных данных ИЛИ от действительных, но не оба сразу.

**GANPaint**

<http://ganpaint.io/demo/?project=church>

Можно дорисовать некоторые объекты к изображению, которые будут выглядеть фотореалистично (ха-ха, ну почти).

**Несколько примеров от NVIDIA**

<https://www.nvidia.com/en-us/research/ai-playground/>

Изменение объекта.

Рисование почти фотореалистичных объектов GauGAN.

Стирание (Inpainting) объектов на изображении.

Вообще по работе с изображениями уже много чего сделано с помощью GAN, вот только некоторые примеры:

<https://blog.inten.to/welcome-to-the-simulation-dd0d8cb6534d>

<https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/>

В других сферах также применяются GAN.

Вот пример создания музыки:

<https://magenta.tensorflow.org/gansynth>

и прочие.

**О структуре генератора.**

<https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d>

Пусть, например, мы хотим использовать GAN для генерации изображений.

В генераторе мы получаем на вход некоторый вектор (шум, латентные переменные) и должны выдать на выходе изображение – массив чисел (двух или трехмерный).

Мы умеем делать наоборот, операциями свертки постепенно уменьшать размер исходного изображения и в конце концов вытянуть его в вектор. Теперь же нам нужно из вектора получить изображение, увеличить размеры. На ум приходит выполнение операции, обратной к свертке – развертки (deconvolution). Конечно ее можно придумать, свертка берет некоторые входы, умножает их на веса, получаем один выход. Для нескольких выходов получаем систему уравнений, где неизвестные это входы, а веса и выходы известны. Такую систему можно попробовать решить, но проблема в том, что эта система может не иметь решений или иметь много решений. Выполнить операцию развертки можно, но сложно. А нужно ли? Ведь у нас нет цели сделать так, чтобы получить из развертки тот же результат, что был на входе свертки, нам надо только увеличить размер изображения до нужного.

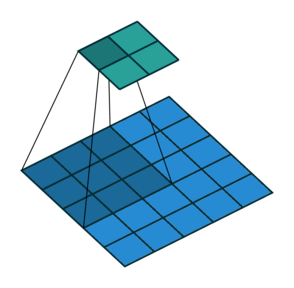
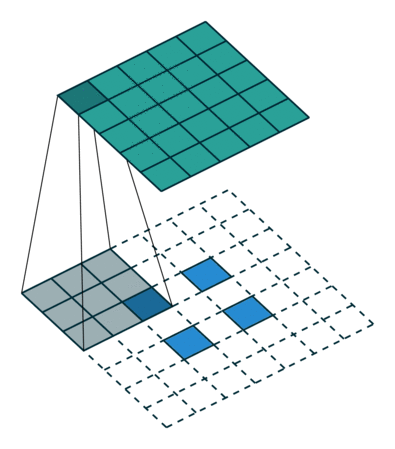
Здесь можно обойтись и без развертки в чистом виде, а воспользоваться, скажем транспонированной сверткой.

На рисунке 3 слева изображен процесс свертки, а справа – транспонированной свертки.

В этом примере

1. вход свертки, матрица 5\*5, пробегается с шагом (stride) 2 ядром размером 3\*3, без набивки (padding) и получается выход размером 2\*2.
2. В транспонированной свертке вход – матрица 2\*2 – распределен (dilation) на больший диапазон, в матрицу 3\*3, к которой добавлена набивка (нули) (шириной 2) и по этой результирующей матрице пробегает ядро свертки размером 3\*3, в результате получается выход размером 5\*5.

Конечно, если мы теперь к выходу транспонированной свертки 2) применим операцию свертки из 1) то не получим того же самого выхода. Но и не надо. Мы выполнили задачу – увеличили размер изображения с 2\*2 до 5\*5. Можно придумать и другие подходы, транспортированная свертка себя хорошо показала на практике.

[](https://miro.medium.com/max/588/1*BMngs93_rm2_BpJFH2mS0Q.gif) [](https://miro.medium.com/max/790/1*Lpn4nag_KRMfGkx1k6bV-g.gif)

**Рисунок 3**. Свертка (слева) и транспонированная свертка (справа) [перейдите по ссылке для анимации].

**Заключение**

От вас требуется понять принцип работы GAN, посмотреть примеры работы их. Понять механизм транспонированной свертки.

Для углубленного изучения выбирайте какую-нибудь конкретную архитектуру GAN, это хорошая тема для домашней работы.

**Литература**

<https://neurohive.io/ru/osnovy-data-science/gan-rukovodstvo-dlja-novichkov/>

1. Kalin, J. Generative Adversarial Networks Cookbook. Packt Publishing, 2018.

2. Foster, D. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media, 2019.

1. Ну, например, мы пытаемся сгенерировать лицо человека, одна из латентных переменных может отвечать, скажем, за цвет глаз. [↑](#footnote-ref-1)
2. Все события вымышленные, имена героев изменены. [↑](#footnote-ref-2)