International College of Economics and Finance

Applied seminar «Quantitative analysis»

Moscow, 2025

INTRODUCTION TO DIPLOMA RESEARCH GENERATIVE ADVERSARIAL NETWORKS

Solovev Georgii

Эта картинка сделана при помощи Deep Learning модели (DALL-E)

Что такое Deep Learning ?

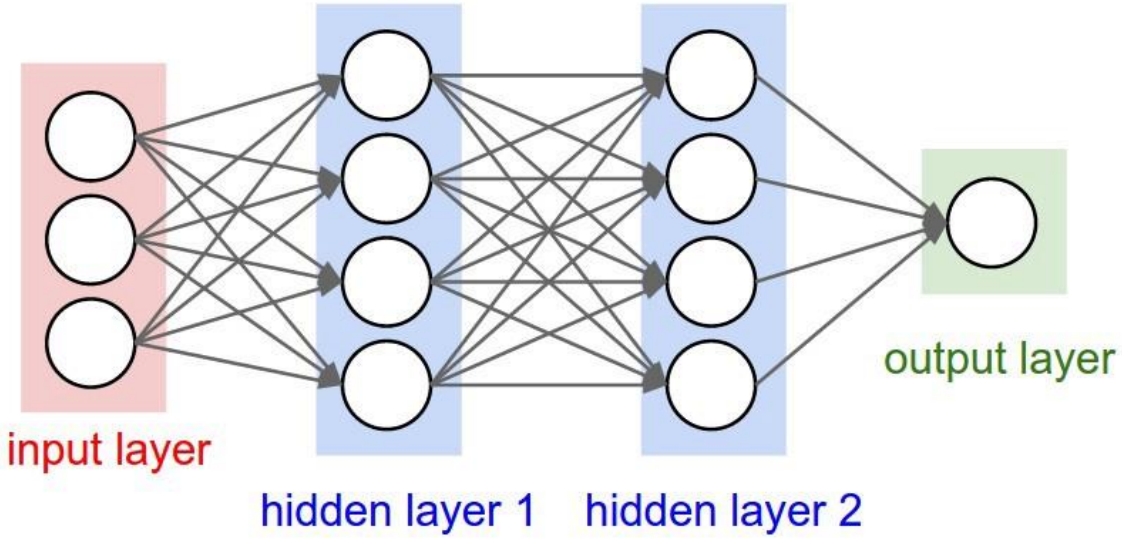
- **Deep Learning -** это метод искусственного интеллекта, в котором **нейросети с большим числом слоев** учатся находить сложные закономерности в данных

- **Принцип Работы:**

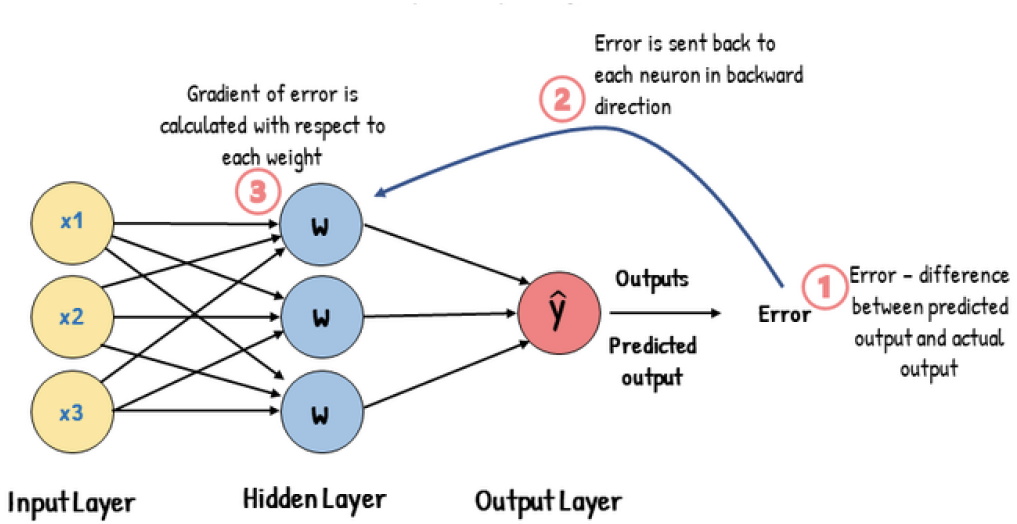
❑ Нейросеть получает данные
❑ Каждый слой нейросети обрабатывает информацию
❑ Выходной слой дает результат
❑ Ошибки анализируются и корректируются

- **Примеры применения**

❑ Face ID – Convolutional Neural Network
❑ Google Translate – трансформерные модели (T5, mBART)
❑ Автопилот Tesla (CNN, ViT, LSTM)
❑ Generative Pre-Trained Transformer + Reinforcement Learning with Human Feedback



Forward Propagation



Backward Propagation

К чему все это ?

GAN – Generative Adversarial Networks (Генеративно Состязательные Сети)

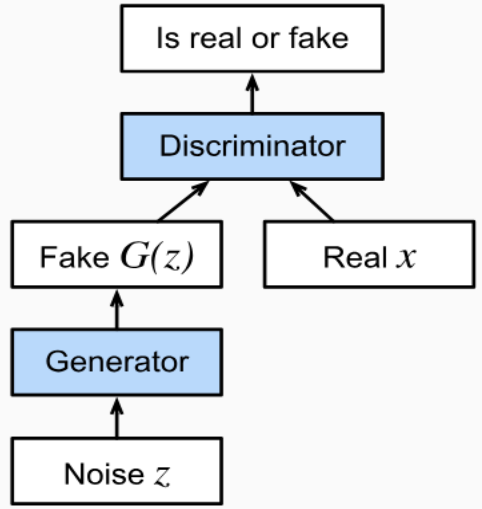
Автор: Иан Гудфеллоу (2014)

Суть: одновременно соревнуются две модели

- **Генератор** - получает случайный шумовой вектор, который он преобразует в синтетические данные, пытаясь воспроизвести реальное распределение
- **Дискриминатор** – оценивает, насколько данные похожи на реальные, обучаясь на разнице между ними

Идея: генератор стремиться “обмануть” дискриминатор, а дискриминатор старается лучше отличать подделки

Обучение заканчивается, когда дискриминатор **уже не может** отличать синтетические данные от настоящих



D2l.AI Fig. 20.1.1

Generator

Discriminator

Generator

Discriminator

I. Пусть **x – случайная величина**, обозначающая сэмпл из нужных нам данных, **z – сэмпл из какого-то распределения**, которое нам легко получить (например, каждая компонента берется из стандартного нормального)

II. Пусть **G – обученная функция**, которая переводит сэмплы из p(z) в сэмплы из p(x) → Процесс генерации происходит в два этапа:

- Случайным образом получаем вектор $\mathbf{z \sim p(z)}$
- Отображаем его в $\hat{x} = \mathbf{G(z)}$: $\hat{x} \sim \mathbf{q(x)}$

III. Пусть существуют генератор с параметрами Θ – $\mathbf{G_\Theta}$
Дискриминатор с параметрами ϕ - $\mathbf{D_\phi}$

IV. Генератор отображает $\mathbf{z \sim N(0,I)}$ в $\hat{\mathbf{x}} \sim \mathbf{q(x)}$, распределение которых приближает реальное распределение данных $\mathbf{p(x)}$
(цель обучения q(x) → p(x))

V. Дискриминатор каждому реальному сэмплу $\mathbf{x}$ и синтетическому $\hat{x}$ ставит в соответствие вероятность $\mathbf{D(x)}$, которая оценивает степень принадлежности $\mathbf{x}$ к реальным данным, т.е. ставит задачу **бинарной классификации** → минимизации **бинарной кросс энтропии**

Максимизируем $\log D_{\phi}(x)$, чтобы заставить дискриминатор давать высокие оценки (вероятности) для реальных данных

$$\left[\min_{\phi} E_{x\sim p(x)}\left[-\log D_{\phi}(x)\right] + E_{x\sim q(x)}\left[-\log\left(1 - D_{\phi}(x)\right)\right]\right]$$

Максимизируем $\log D_{\phi}(x)$, чтобы заставить дискриминатор понижать оценки для сгенерированных данных

Учтем обозначение $\hat{x} = G_{\theta}(z)$, и то, что мы стараемся максимизировать вероятность принадлежности к реальным данным, как ее оценивает дискриминатор →
Задачу которую решает генератор можно записать следующим образом

Дискриминатор возвращает вероятность того, что поданный ему объект является настоящим

$$\left[\theta^{*} = \arg\max_{\theta} E_{z\sim p(z)}\left[D_{\theta}\left(G_{\theta}(z)\right)\right]\right]$$

Случайный шум

Генератор преобразует шум в синтетические данные

Эквивалентно, данную задачу можно записать в другом виде, позволяя записать генератор и дискриминатор вместе (+избавимся от лишних минусов)

$$\left[\arg\min_{\theta} \mathrm{E}_{z\sim p(z)} \log\left(1 - \mathrm{D}_{\theta}\left(\mathrm{G}_{\theta}(z)\right)\right) = \arg\max_{\theta} \mathrm{E}_{z\sim p(z)} - \log\left(1 - \mathrm{D}_{\theta}\left(\mathrm{G}_{\theta}(z)\right)\right)\right]$$

$$\left[\min_{\theta}\max_{\phi} \mathrm{E}_{x\sim p(x)} \log \mathrm{D}_{\phi}(x) + \mathrm{E}_{z\sim p(z)} \log\left(1 - \mathrm{D}_{\phi}\left(\mathrm{G}_{\theta}(z)\right)\right)\right]$$

Получается, что на самом деле генератор и дискриминатор пытаются оптимизировать одну функцию: генератор ее минимизирует, а дискриминатор максимизирует

Пусть эта функция (минус бинарная энтропия) обозначается функцией $[\mathcal{L}_{\theta,\phi}]$

- Дискриминатор пытается улучшиться, чтобы лучше отличать реальные данные от синтетических
- Генератор хочет уменьшить ошибку дискриминатора, создавая данные, которые генератор примет за настоящие

$$[\min_{\theta} \max_{\phi} \mathcal{L}_{\theta,\phi}]$$

По параметрам дискриминатора **минимум бинарной кросс-энтропии достигается на** следующей функции **оптимальном дискриминаторе** для фиксированного генератора:

$$[D^*_{\phi}(x) = \frac{p(x)}{p(x) + q(x)}]$$

Учитывая это, и формулу для $[\mathcal{L}_{\theta,\phi}]$, интуицию работы метода обучения GANов со стороны генератора можно сформулировать следующим образом:

1. Мы замеряем, насколько реалистичными являются сгенерированные сэмплы $\hat{x}_1 \ldots \hat{x}_n$ используя для этого оптимальный дискриминатор
2. Мы хотим увеличить отклик дискриминатора на каждом сэмпле, т.е. пытаемся модифицировать каждый предсказанный элемент $\hat{x}_i$ так, чтобы на нем стало выше значение $D^*_\phi(\hat{x}_i)$

Подставив выражение для оптимального дискриминатора в $\mathcal{L}$, мы можем избавиться от внутренней максимизации в исходной задаче и оставить только внешнюю минимизацию по параметрам генератора:

$$[\mathcal{D}_\theta = \mathrm{E}_{x\sim p(x)} \log \mathrm{D}^*_\phi(x) + \mathrm{E}_{x\sim q(x)} \log\left(1 - \mathrm{D}^*_\phi(x)\right)$$

$$[= -\mathbf{log4} + \mathbf{KL}\left(\mathbf{p} \parallel \frac{\mathbf{p}+\boldsymbol{q}}{\mathbf{2}}\right) + \mathbf{KL}\left(\mathbf{q} \parallel \frac{\mathbf{p}+\boldsymbol{q}}{\mathbf{2}}\right)$$
$$= -\boldsymbol{log}\,\mathbf{4} + \mathbf{2} \cdot \mathbf{JSD}(\mathbf{p} \parallel \boldsymbol{q})]$$

Получается, что при оптимальном дискриминаторе генератор, решая внешнюю оптимизационную задачу, уменьшает расстояние между распределениями реальных и синтетических данных, действительно приближая их друг к другу. Следовательно необходимо:

1. Решить внутреннюю задачу максимизации по ϕ, повторяя шаги ниже до сходимости ϕ к оптимальному значению
- Составить мини-батч из сэмплов шума $z_1, \dots z_n$ **из p(z)**
- Составить мини-батч из сэмплов данных $x_1, \dots x_n$ **из p(x)**

- Обновить дискриминатор, сделав шаг вверх по его градиенту

$$\left[\nabla_\phi \frac{1}{n}\sum_{i=1}^{n}\left[log\, D_\phi(x_i) + log\left(1 - D_\phi\left(G_\theta(z_i)\right)\right)\right]\right]$$

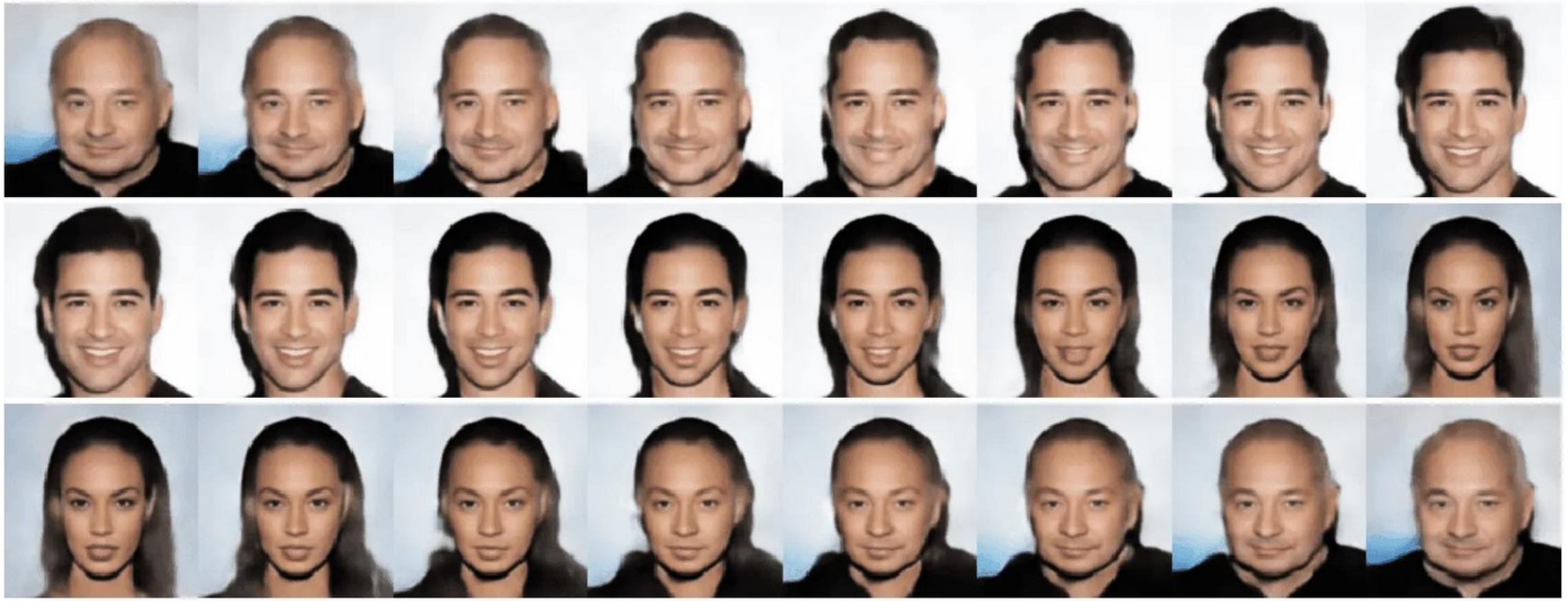
2. Сделать шаг SGD для внешней задачи минимизации по θ
- Составить мини-батч из сэмплов шума $z_1, \dots z_n$ **из p(z)**
- Обновить генератор, сделав шаг вниз по его градиенту

$$\left[\nabla_\theta \frac{1}{n}\sum_{i=1}^{n}\left(\log\left(1-D_{\phi^*}\left(G_\theta(z_i)\right)\right)\right) = \frac{1}{n}\sum_{i=1}^{n}-\frac{\nabla_\theta f\left(G_\theta(z_i)\right)}{1-f\left(G_\theta(z_i)\right)}\right]$$

Помимо этого, на практике используются модификации метода стохастического спуска, например, Adam (adaptive model algorithm)

$$\nu_t = \beta_1 * \nu_{t-1} - (1-\beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1-\beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta\frac{\nu_t}{\sqrt{s_t+\epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$



Все мы сейчас Джуниор из Сопрано…

Но теперь мы переходим к TimeGAN

Хорошая генеративная модель для временных рядов должна сохранять **временную динамику**, то есть генерируемые последовательности должны уважать оригинальные зависимости между переменными во времени. Например, если у нас есть многомерные последовательные данные $x_{\{1:T\}} = (x_1, \dots x_T)$ модель должна правильно оценивать **условное распределение**: $\mathrm{p}\left(\mathrm{x_t}\middle|\mathrm{x_{\{1:t-1\}}}\right)$
то есть зависимость каждого последующего элемента от предыдущих.

TimeGAN – это расширение классического GAN для временных рядов

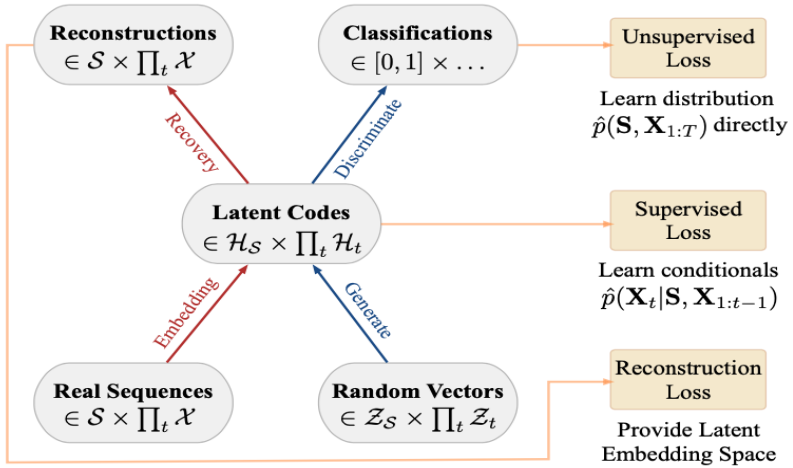
- ❑ Учитывает временные зависимости в данных.
- ❑ Совмещает **контролируемое и неконтролируемое обучение**.
- ❑ Генерирует последовательности с **сохранением структуры оригинальных данных**.

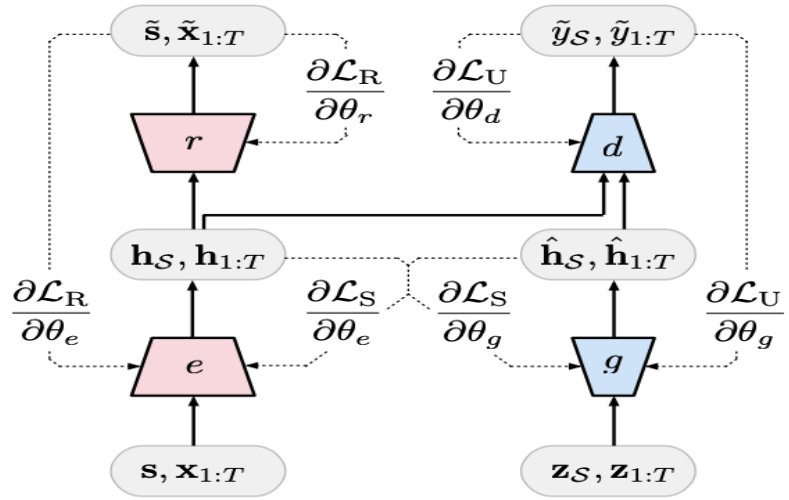
Почему обычные GAN не работают?

- **GAN** обучаются только на моментных срезах данных (iid-распределение).
- **Временные ряды** требуют **учета последовательных зависимостей**.
- **TimeGAN** решает эту проблему с помощью **рекуррентной архитектуры**.

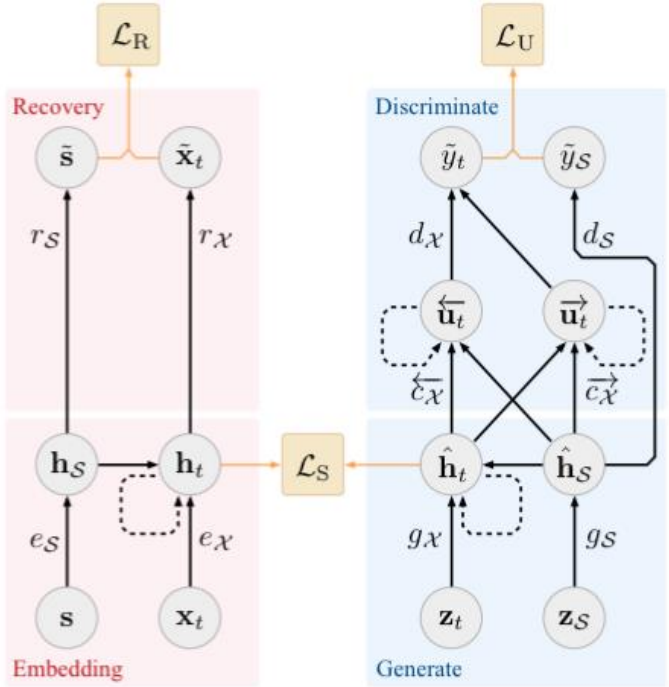
- TimeGAN включает **четыре ключевых компонента**:
- **Embedding Network** – переводит временные ряды в скрытое представление, уменьшая их размерность
- **Generator** – создает новые скрытые представления, которые потом преобразуются в временные ряды
- **Recovery Network** – восстанавливает временные ряды из скрытых представлений
- **Discriminator** – отличает реальные временные ряды от искусственных

Ключевая идея: **автоэнкодерные компоненты (кодирование и восстановление) обучаются совместно с состязательными компонентами (генерация и дискриминация)**, что позволяет модели одновременно кодировать признаки, создавать представления и **учитывать временные зависимости**.





WOOO MATHS ON THE WHITEBOARD WOOO



WOOO CODE ON THE SCREEN WOOO


That's all Folks!