Introduction

paper (https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf)

- supplement (https://www.vanderschaar-lab.com/papers/NIPS2019_TGAN_Supplementary.pdf)
- github (https://github.com/jsyoon0823/TimeGAN)

A GAN learns to produce synthetic ("fake") feature vectors $\hat{\mathbf{x}}$ of length n that are plausible elements $\mathbf{x} \in p_{\text{data}}$.

- The relationship between two features $\hat{\mathbf{x}}, \hat{\mathbf{x}}_{j'}$ of synthetic $\hat{\mathbf{x}}$ should be consistent with the relations between $\mathbf{x}_j, \mathbf{x}_{j'}$ of real $\mathbf{x}$
- **cross sectional**
    - consistency between the pixels of an image

But much data (especially in Finance) *also* has a **time** dimension

- $\mathbf{x}$ is two dimensional
    - $\mathbf{x}_{(t)}$ is a vector of n features, representing the state of the world at time t

The goal of a generative model would be to generate examples that are sequences (of length T) where each element is a vector of length n.

- Each training example is a sequence $\mathbf{x}_{(1)}, \ldots \mathbf{x}_{(T)}$
- Generate synthetic $\hat{\mathbf{x}}_{(1)}, \ldots \hat{\mathbf{x}}_{(T)}$
- with both forms of consistency
    - cross sectional between features at a fixed time t: $\hat{\mathbf{x}}_{(t),j}$ and $\hat{\mathbf{x}}_{(t),j'}$
    - sequential ("time series") between the feature vectors at different time steps: $\hat{\mathbf{x}}_{(t)}$ and $\hat{\mathbf{x}}_{(t+1)}$

Following the notation of the paper

- we use Python-like notation for sequences

$$\mathbf{x}_{(1:T)} = \mathbf{x}_{(1)}, \ldots \mathbf{x}_{(T)}$$

We could try to encode a timeseries relationship as a pseudo cross-sectional relationship

- flatten $(T \times n)$ vector $\mathbf{x}$ into a 1D vector of length $(T * n)$
- the relationship between pairs of vector elements at distance n would be one step of time

This is unlikely to work well for common timeseries relationships: autoregressive

The *Time GAN (TGAN)* is a GAN (Generator/Discriminator pair) with an *extra* component: the *Supervisor*.

The Supervisor is responsible for constraining the Generator to produce sequences with sequential properties

- learns an autoregressive model of $\mathbf{x}_{(1:T)}$
- creates a predicted $\hat{\mathbf{x}}_{(1:T)}$
    - creates a single element $\hat{\mathbf{x}}_{(t)}$ at a time
    - generating $\hat{\mathbf{x}}_{(t)}$ conditional on $\hat{\mathbf{x}}_{(1:t-1)}$
- creates a *Supervised Loss* which is added to the standard Generator Loss

The Supervised Loss enforces the sequence dynamic as a constraint on one step ahead elements of the sequence.

Thus, the Generator is encouraged to produce sequences that

- not only fool the Discriminator
- but also exhibit sequential properties

There is one additional component to the TGAN.

Rather than having the Generator/Discriminator pair work on elements of $\mathbf{x}|(t)$

- they both work on reduced dimensional *encodings* of $\mathbf{x}_{(t)}$ denoted as $\mathbf{h}_{(t)}$
- the reduced dimensional encoding $\mathbf{h}_{(t)}$ is called an *embedding* of $\mathbf{x}_{(t)}$
    - same idea as word embeddings

The embedding is created by the *Embedder*

- The Encoder half
- of an Autoencoder (Encoder/Decoder pair)

This makes for a lot of moving parts (and a lot of notation).

Each component contributes (at least one) one Loss to the total *multi-part* Loss.

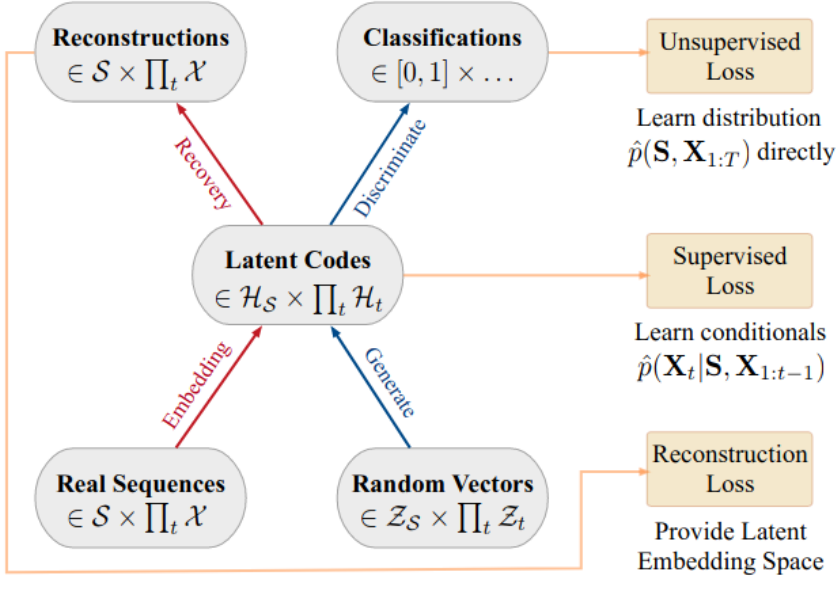
- Generator Loss
- Discriminator Loss
- Supervisor Loss
- Autoencoder Reconstruction Loss

It also leads to options for how to train the components.

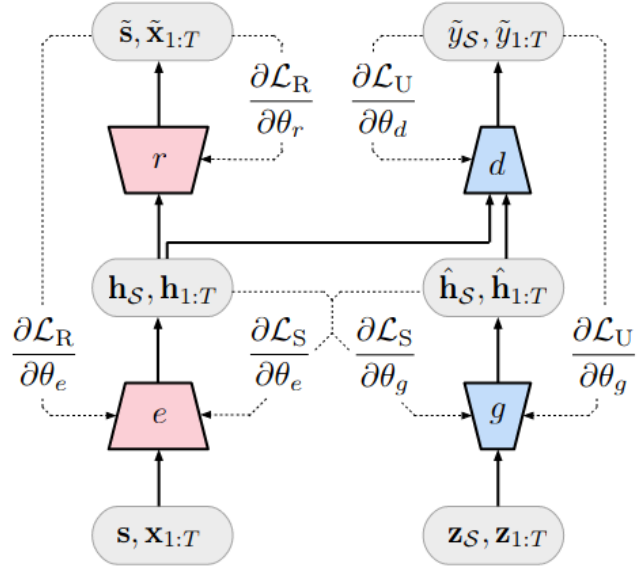
Is the Autoencoder trained

- independently
- or jointly with the GAN + Supervisor components ?

Training scheme (from paper)]



(a) Block Diagram



(b) Training Scheme

Details

Notation

text	Description
$\mathbf{s}$	static feature
$\mathbf{x}_{(1:T)}$	Temporal feature
$\tilde{\mathbf{x}}_{(1:T)}$	reconstructed (by Autoencoder) $\mathbf{x}$
	$\tilde{\mathbf{x}}_{(1:T)} = \texttt{autoencoder}(\mathbf{x}_{(1:T)})$
$\hat{\mathbf{x}}_{(1:T)}$	fake data
	$\hat{\mathbf{x}}_{(1:T)} = \texttt{decoder}(\hat{\mathbf{h}})$
$\mathbf{s}, \mathbf{x}_{(1:T)}$	real data -- s: non-sequence input, $x_{1:t}$ sequence input
$\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{(1:T)}$	reconstructed real data (autoencoder output)
$\mathbf{z}_s, \mathbf{z}_{(1:T)}$	random vector (generator input)
$\mathbf{h}_s, \mathbf{h}_{(1:T)}$	real data in latent space (encoder output)
	$\mathbf{h}_{(1:T)} = \texttt{embedder}(\mathbf{x}_{(1:T)})$
$\hat{\mathbf{e}}_s, \hat{\mathbf{e}}_{(1:T)}$	fake data (generator output) in latent space (encoder output)
	$\hat{\mathbf{e}}_{(1:T)} = \texttt{generator}(\mathbf{z})$
$\hat{\mathbf{h}}_s, \hat{\mathbf{h}}_{(1:T)}$	fake data (generator + supervisor output) in latent space (encoder output)
	$\hat{\mathbf{h}}(1:T) = \texttt{supervisor}(\hat{\mathbf{e}})$
$\hat{\hat{\mathbf{h}}}_s, \hat{\hat{\mathbf{h}}}_{(1:T)}$	fake data (real + supervisor output) in latent space (encoder output)
	$\hat{\hat{\mathbf{h}}}_{(1:T)} = \texttt{supervisor}(\mathbf{h})$
$\mathbf{y}$	Boolean: Real/Fake Discriminator or real data
$\hat{\mathbf{y}}$	Boolean: Real/Fake Discriminator on fake data (generator + supervisor output)
$\hat{\hat{\mathbf{y}}}$	Boolean: Real/Fake Discriminator on fake data (real + supervisor output)

Losses

name	Definition	Description
embedding_loss	$\mathrm{MSE}(\mathbf{x}_{(1:T)}, \tilde{\mathbf{x}}_{(1:T)})$	Autoencoder reconstruction error
e_loss	$\mathrm{embedding\_loss}^{0.5}$ $+ \mathrm{generator\_loss}_{\mathrm{supervised}}$	Embedder Loss
		= Autoencoder reconstruction error + one step ahead prediction error

Examples, real and synthetic

To be fully general, the paper *also* allows examples to have 2 parts

- static (non-sequence)
- sequence

Thus

- a real example is a pair $\mathbf{s}, \mathbf{x}_{(1:T)}$

As we are primarily concerned with the sequence properties of $\mathbf{x}_{(1:T)}$, we may omit $\mathbf{s}$ to simplify the presentation.

We will assume that each element $\mathbf{x}_{(t)}$ of (real and fake) sequences $\mathbf{x}_{(1:T)}$ is a vector of length n

- for example, the time t characteristics of each of n tickers

We don't need to assume that $\mathbf{s}$ is also a vector of length n

- e.g., no need to assume a unique characteristic per ticker

There are *two kinds* of synthetic sequences (both sequences of embeddings rather than raw inputs)

- $\hat{\mathbf{h}}_{(1:T)}$: produced by the Supervisor, given input from the Generator

$$\hat{\mathbf{h}}_{(1:T)} = \text{supervisor}(\text{generator}(\mathbf{z}))$$

- $\hat{\hat{\mathbf{h}}}_{(1:T)}$: produced by the Supervisor, given (the embedding of) **real** data $\mathbf{x}$

$$\begin{aligned} \mathbf{h}_{(1:T)} &= \text{embedder}(\mathbf{x}_{(1:T)}) \\ \hat{\hat{\mathbf{h}}}_{(1:T)} &= \text{supervisor}(\mathbf{h}_{(1:T)}) \end{aligned}$$

Loss functions: High level view of objectives

The *Time GAN (TGAN)* tries to achieve both cross sectional and sequence objectives by a multi-part Loss function.

The first objective (enforced by a loss function) is the familiar GAN objective

- $p_{\text{model}} \approx p_{\text{data}}$ for some definition of equality of distributions
- Also called: the *Unsupervised Loss*

The second objective is sequence related

- *Conditional* (one step ahead) distributions of Fake and Real are equal
- $p_{\text{model}}\left(\mathbf{x}_{(t)} \middle| \mathbf{x}_{(1:t-1)}\right) \approx p_{\text{data}}\left(\mathbf{x}_{(t)} \middle| \mathbf{x}_{(1:t-1)}\right)$
- Also called: the *Supervised Loss*

We will use the KL Divergence as our measure of "dissimilarity" of two distributions

- Just like in the plain GAN:
    - under the assumption that the optimal Discriminator can be found, the KL Divergence turns into the JSD.

As we shall see

- there are multiple steps to training the TGAN
    - independent training of the Embedder
    - independent training of the Supervisor
    - joint training of all components

Thus, there will be a separate Loss function for each training step.

The Losses will be multi-part: consisting of sums of terms for sub-losses.

Notation

We use BCE as a function that computes *Binary Cross Entropy*

Embedder (Autoencoder)

Rather than working on "raw" examples $\mathbf{s}, \mathbf{x}_{(1:T)}$ the model creates an *embedding* of the example

- lower dimensional representation
- that preserves "semantics"
    - same idea as embedding words, that is, changing the representation of a word
        - from a categorical encoded by a *sparse*, very long OHE vector
        - to a shorter, *dense* vector

The embedding will be created by an Autoencoder

- Given input example
- Pass it through an Encoder: a bottle-neck that reduces the dimensions
- Have the Decoder try to reconstruct the input, given the reduced dimension representation

So the Embedder is implemented by the Encoder half of an Autoencoder.

The Encoder

- takes input $\mathbf{s}, \mathbf{x}_{(1:T)}$
- outputs reduced dimension embedding/latent $\mathbf{h}_s, \mathbf{h}_{(1:T)}$

The Decoder

- takes input $\mathbf{h}_s, \mathbf{h}_{(1:T)}$
- outputs $\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{(1:T)}$ The Encoder/Decoder pair is trained with the objective

$$\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{(1:T)} \approx \mathbf{s}, \mathbf{x}_{(1:T)}$$

That is: the Decoder attempts to create the best reconstruction of the Encoder input, given the embedding.

Both the Encoder and Decoder

- must be autoregressive
    - generate embedding of future element, conditional on the past elements
- must obey *causal ordering* of the sequence
    - can't look at any of $\mathbf{x}_{(t+1:)}$ when embedding $\mathbf{x}_{(t)}$

Why use embeddings rather than raw data ?

One obvious reason is space considerations

- Having the GAN work with lower dimensional embeddings rather than high dimensional raw input

The less obvious (perhaps) reason comes from our experience with sequence data such as time series of returns

- PCA analysis shows that highly reduced dimensions preserve much of the variation
- Factor models propose that a small number of common factors are responsible for the variation of the cross section of stock returns

Autoencoder training (independent pass)

There are two obvious choices for training the Autoencoder

- Independent of the GAN Generator/Discriminator + Supervisor
    - That is: just "learn" how to compress sequences without worrying about where the sequences come from
- Jointly with the GAN + Supervisor

The authors chose

- an initial training of the Autoencoder independently
- followed by a joint training with the GAN + Supervisor

For the independent training of the Autoencoder we use the standard Reconstruction Loss of an Autoencoder:

$$\mathcal{L}_R \quad = \quad \text{embedding\_loss} \quad = \quad \text{MSE}(\mathbf{x}_{(1:T)}, \tilde{\mathbf{x}}_{(1:T)}) \quad \text{Reconstruction Loss}$$

This is the only Loss Term that is optimized in the independent training.

We defer the Autoencoder loss term that arises from joint training until after we introduce the Supervisor.

Supervisor

The Supervisor is a Recurrent Network that helps in learning an autoregressive model of $\mathbf{x}_{(1:T)}$.

However: it works on the embeddings of data rather than the raw data

- creates a predicted $\mathbf{h}_{(1:T)}$
    - creates a single element $\mathbf{h}_{(t)}$ at a time
    - generating $\mathbf{h}_{(t)}$ conditional on $\mathbf{h}_{(1:t-1)}$
- rather than creating a predicted $\mathbf{x}_{(1:T)}$
    - if desired, can map the predicted $\mathbf{h}_{(1:T)}$ to predicted $\mathbf{x}_{(1:T)}$ using the Decoder half of the Autoencoder

$$\hat{\mathbf{x}}_{(1:T)} = \text{decoder}(\hat{\mathbf{h}})$$

There are two sources of embeddings that are fed to the Supervisor.

The Supervisor takes the given sequence of embeddings into the autoregressive sequence.

The two inputs to the Supervisor, and their outputs are

- Embeddings of real examples $\mathbf{x}$

$$\begin{aligned} \mathbf{h}_{(1:T)} &= \text{embedder}(\mathbf{x}_{(1:T)}) \\ \hat{\hat{\mathbf{h}}}_{(1:T)} &= \text{supervisor}(\mathbf{h}) \end{aligned}$$

- Embeddings created by the Generator (i.e., fake examples, in latent space)

$$\begin{aligned} \hat{\mathbf{e}}_{(1:T)} &= \text{generator}(\mathbf{z}) \\ \hat{\mathbf{h}}(1:T) &= \text{supervisor}(\hat{\mathbf{e}}) \end{aligned}$$

Supervisor training (independent pass)

There is a *Supervised Loss* associated with the Supervisor.

It is a measure of the quality of the embedding as input to the Supervisor

- Given the embedding, can the Supervisor construct a loss-less synthetic
  - Note: this synthetic is derived from a **real** example $\mathbf{x}$, **independent** of the Generator

$$
\begin{aligned}
\mathbf{h}_{(1:T)} &= \text{embedder}(\mathbf{x}_{(1:T)}) \\
\hat{\hat{\mathbf{h}}}_{(1:T)} &= \text{supervisor}(\mathbf{h}) \\
\mathcal{L}_S &= \text{generator\_loss}_{\text{supervised}} = \text{MSE}(\mathbf{h}_{(1:)}, \hat{\hat{\mathbf{h}}}_{(1:T)})
\end{aligned}
$$

As you can see, $\mathcal{L}_S$ depends on the behavior of the Encoder side (`embedder`) of the Autoencoder.

But just as we had an initial independent training of the Autoencoder

- we have an initial independent training of the Superviser

During joint training, we will use Loss functions that cause the weights of the `embedder` and `supervisor` to update together.

Discriminator

There are three sources of examples fed to the Discriminator, each resulting in a different judgment of Real/Fake.

Recall that the Generator, Discriminator and Supervisor all take embeddings (as opposed to raw examples) as input.

The three inputs to the Discriminator, their outputs (judgments), and associated losses are

- Embeddings of Real examples

$$\begin{aligned}\mathbf{h}_{(1:T)} &= \text{embedder}(\mathbf{x}_{(1:T)}) \\ \mathbf{y} &= \text{discriminator}(\mathbf{h}) \\ \text{D\_loss\_real} &= \text{BCE}(1's, \mathbf{y})\end{aligned}$$

- Embeddings of Fake examples (created by Generator only)

$$\begin{aligned}\hat{\mathbf{e}}_{(1:T)} &= \text{generator}(\mathbf{z}) \\ \hat{\mathbf{y}} &= \mathbf{y}_{\text{fake\_e}} = \text{discriminator}(\hat{\mathbf{e}}) \\ \text{D\_loss\_fake\_e} &= \text{BCE}(0's, \mathbf{y}_{\text{fake\_e}}) \\ \text{generator\_loss}_{\text{unsupervised\_e}} &= \text{BCE}(1's, \mathbf{y}_{\text{fake\_e}})\end{aligned}$$

- Embeddings of Fake examples (created by Generator + Supervisor)

$$
\begin{aligned}
\hat{\mathbf{e}}_{(1:T)} &= \text{generator}(\mathbf{z}) \\
\hat{\mathbf{h}}(1:T) &= \text{supervisor}(\hat{\mathbf{e}}) \\
\dot{\mathbf{y}} &= \mathbf{y}_{\text{fake}} = \text{discriminator}(\hat{\mathbf{h}}) \\
\text{D\_loss\_fake} &= \text{BCE}(0's, \mathbf{y}_{\text{fake}}) \\
\text{generator\_loss}_{\text{unsupervised}} &= \text{BCE}(1's, \mathbf{y}_{\text{fake}})
\end{aligned}
$$

The Discriminator Loss $\mathcal{L}_D$ is the (weighted, but we ignore the weights) sum

$$\mathcal{L}_D = \mathrm{D\_loss\_real} + \mathrm{D\_loss\_unsupervised} + \mathrm{D\_loss\_fake}$$

Generator

Remark and warning

There are a number of variables in the code that begin with `generator_loss`.

I find the naming somewhat confusing, but I will preserve it in this presentation after explaining.

The Generator tries to fool the Discriminator in two ways.

- via direct output of the Generator
- by the output of the Supervisor
    - creates an autoregressive model of the direct output of the Generator

Recall that the Generator, Discriminator and Supervisor all take embeddings (as opposed to raw examples) as input.

The two outputs of the Generator, their evaluation by the Discriminator, and associated losses ar

- Embeddings of Fake examples (created by Generator only)

$$\begin{aligned} \hat{\mathbf{e}}_{(1:T)} &= \text{generator}(\mathbf{z}) \\ \hat{\mathbf{y}} &= \mathbf{y}_{\text{fake\_e}} = \text{discriminator}(\hat{\mathbf{e}}) \\ \text{generator\_loss}_{\text{unsupervised\_e}} &= \text{BCE}(1's, \mathbf{y}_{\text{fake\_e}}) \end{aligned}$$

- Embeddings of Fake examples (created by Generator + Supervisor)

generator(z)

There is an *additional* loss term associated with the Generator.

It compares the statistical moments (mean, variance) of the real examples with fake examples

- real examples with first two moments μ, σ
- fake examples with first two moments $\hat{\mu}, \hat{\sigma}$
    - fake examples obtained from the sequence of embeddings $\hat{\mathbf{h}}$ produced by (Generator + Supervisor) using the Decoder side of the Autoencoder

$$\hat{\mathbf{x}}_{(1:T)} = \text{decoder}(\hat{\mathbf{h}}_{(1:T)})$$

The Generator Moments loss is

$$\text{generator\_loss}_{\text{moment}} = (\mu - \hat{\mu}) + (\sigma - \hat{\sigma})$$

The total loss $\mathcal{L}_G$ for the Generator is the sum of the sub-losses

$$\mathcal{L}_G = \text{generator\_loss}_{\text{unsupervised\_e}} + \text{generator\_loss}_{\text{unsupervised}} + \mathcal{L}_S + \text{generator\_loss}_{\text{moment}}$$

Joint training

The Embedder and Supervisor are initially trained (separately) independently of the GAN.

It is during joint training that the GAN (Generator and Discriminator) are trained, in the usual Adversarial Training manner of a GAN.

It may not be obvious but, even though Adversarial Training looks like it is designed to update the Generator and Discriminator weights

- all weights are potentially updated, including the Embedder and Supervisor

There is a bit of subtlety here.

The most obvious purpose of joint training is to implement the GAN adversarial training

- Generator weights Θ_G are updated to better produce examples to fool the Discriminator
- Discriminator weights Θ_D are updated to better distinguish between Real and Fake examples.

How do the weights of the Supervisor and Autoencoder come into play ?

Recall that the Generator creates two kinds of Fake embeddings of examples.

- Embeddings of Fake examples (created by Generator only)

$$
\begin{aligned}
\hat{\mathbf{e}}_{(1:T)} &= \text{generator}(\mathbf{z}) \\
\hat{\mathbf{y}} &= \mathbf{y}_{\text{fake\_e}} = \text{discriminator}(\hat{\mathbf{e}}) \\
\text{generator\_loss}_{\text{unsupervised\_e}} &= \text{BCE}(1's, \mathbf{y}_{\text{fake\_e}})
\end{aligned}
$$

- Embeddings of Fake examples (created by Generator + Supervisor)

$$
\begin{aligned}
\hat{\mathbf{e}}_{(1:T)} &= \text{generator}(\mathbf{z}) \\
\hat{\mathbf{h}}_{(1:T)} &= \text{supervisor}(\hat{\mathbf{e}}) \\
\dot{\mathbf{y}} &= \mathbf{y}_{\text{fake}} = \text{discriminator}(\hat{\mathbf{h}}) \\
\text{generator\_loss}_{\text{unsupervised}} &= \text{BCE}(1's, \mathbf{y}_{\text{fake}})
\end{aligned}
$$

The first

- $\hat{\mathbf{e}}_{(1:T)} = \text{generator}(\mathbf{z})$

is independent of the Supervisor, but the second

- $\hat{\mathbf{h}}_{(1:T)} = \text{supervisor}(\hat{\mathbf{e}})$ is affected by the Supervisor (hence its weights).

Moreover, the second kind of example is fed into the Discriminator.

Therefore, in order to reduce the associated Generator loss

$$\text{generator\_loss}_{\text{unsupervised}} = \text{BCE}(1's, \mathbf{y}_{\text{fake}})$$

or associated Discriminator Loss

$$\text{D\_loss\_fake} = \text{BCE}(0's, \mathbf{y}_{\text{fake}})$$

a gradient may arise that affects the weights of the Supervisor.

Note that the Supervisor Loss does not depend *directly* on Fake examples.

It is defined solely on real examples

$$
\begin{aligned}
\mathbf{h}_{(1:T)} &= \text{embedder}(\mathbf{x}_{(1:T)}) \\
\hat{\hat{\mathbf{h}}}_{(1:T)} &= \text{supervisor}(\mathbf{h}) \\
\mathcal{L}_S &= \text{generator\_loss}_{\text{supervised}} = \text{MSE}(\mathbf{h}_{(1:)}, \hat{\hat{\mathbf{h}}}_{(1:T)})
\end{aligned}
$$

It is not exposed directly to fake examples, only indirectly

- Through the dependence of part of the Generator and Disciminator losses on the Supervisor

It makes sense that $\mathcal{L}_S$ depends only on real examples

- they are from p_{data}, which defines all statistical relationships
- we can't infer any true relationship from Fake data

So Adversarial Training of the GAN can modify $\mathcal{L}_S$ which depends on the Encoder half of the Autoencoder

- see the equations above
    - $\mathcal{L}_S$ depends on $\mathbf{h}_{(1:T)}$
    - $\mathbf{h}_{(1:T)} = \text{embedder}(\mathbf{x}_{(1:T)})$

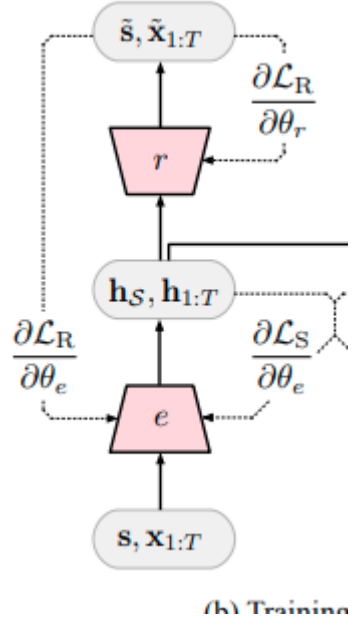
Hence Adversarial Training can also affect the weights of the Autoencoder.

Thus, the Autoencoder learns a better encoding by joint training.

We can see the definition of `e_loss` combines these the two "reconstruction" related terms

$$
\begin{aligned}
\tilde{\mathbf{x}}_{(1:t)} &= \text{autoencoder}(\mathbf{x}_{(1:T)}) && \text{Reconstru} \\
\text{embedding\_loss} &= \text{MSE}(\mathbf{x}_{(1:T)}, \tilde{\mathbf{x}}_{(1:T)}) && \text{Reconstru} \\
&= \text{embedding\_loss}^{0.5} + \text{generator\_loss}_{\text{supervised}}
\end{aligned}
$$

The fact that the Embedder and Supervisor/Generator/Discriminator are tied together in training is also apparent if we zoom in on the diagram of the Training Scheme



Training Scheme: zoom in on Embedder gradient updates

Notice: the gradient

$$\frac{\partial \mathcal{L}_S}{\partial \Theta_e}$$

of the Supervised Loss with respect to the Encoder weights Θ_e flowing back to the Encoder e

Generating synthetic examples

Once trained, we can generate new Fake examples

- by creating a random noise vector $\mathbf{z}$
- feeding $\mathbf{z}$ to the `generator` and `supervisor` to get a synthetic embedding (created by Generator + Supervisor)
- decoding the synthetic embedding (by the Decoder part of the Autoencoder) to create a sequence in the original data space

$$
\begin{aligned}
\hat{\mathbf{e}}_{(1:T)} &= \text{generator}(\mathbf{z}) \\
\hat{\mathbf{h}}_{(1:T)} &= \text{supervisor}(\hat{\mathbf{e}}) \\
\hat{\mathbf{x}}_{(1:T)} &= \text{decoder}(\hat{\mathbf{h}}_{(1:T)})
\end{aligned}
$$

In [ ]:
```
print("Done")
```