Deep Learning in Finance

GANs II

Damien Challet
damien.challet@centralesupelec.fr

12th November 2024

Lecture plan

1. More advanced GANs
2. Methodological issues and improvements

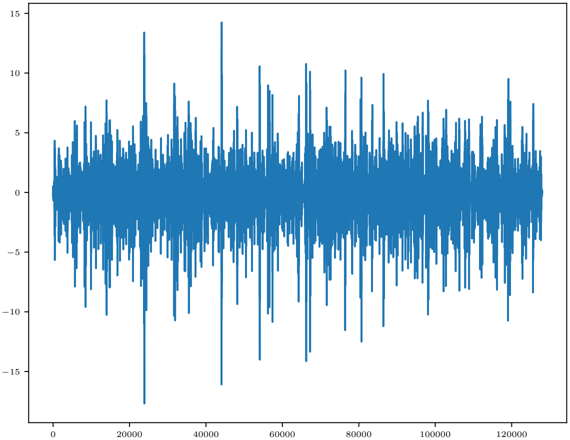
Naive GANs as market simulators

- Input noise $\eta \in \mathbb{R}^{N_F = d \times T}$ with $\eta_i \sim \mathcal{N}(0, 1)$
- Output price returns $r_t \in \mathbb{R}^T$
- Stylized facts?
  - heavy tailed returns — hard/impossible
  - long memory of volatility — hard/impossible
- better than i.i.d. r_t?

discriminator says yes

Results: r_t vs t, Gaussian features

GAN: MLP, $T = 128$, $N_F = 10$, ReLU



Theorems

[link] and [link]

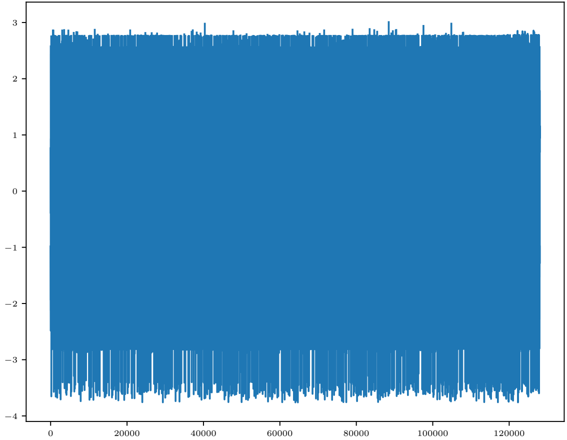
1. CNNs cannot transform Gaussian noise into heavy-tailed noise
2. MLPs cannot transform Gaussian noise into heavy-tailed noise

Solutions:

1. Use another NN architecture
2. Use another type of noise
3. Transform data

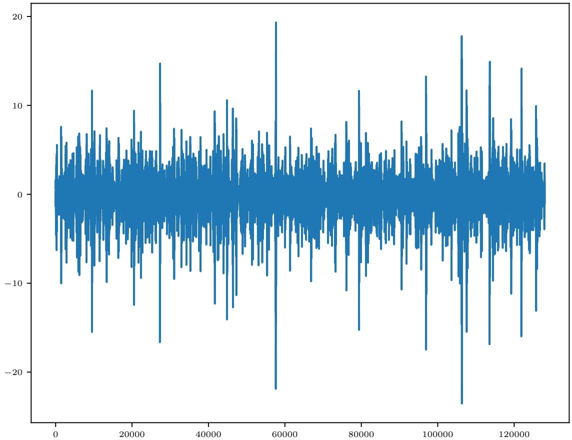
Results: r_t vs t, rescaled, Gaussian features

GAN: MLP, $T = 128$, $N_F = 10$, ReLU, individually rescaled



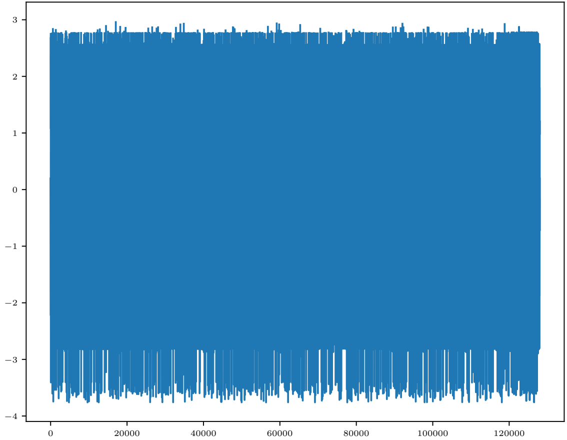
Results: r_t vs t, Student-t features

GAN: MLP, $T = 128$, $N_F = 10$, ReLU



Results: r_t vs t, rescaled, Student-t features

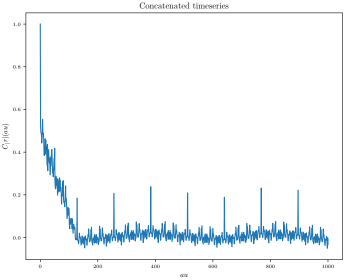
GAN: MLP, $T = 128$, $N_F = 10$, ReLU, individually rescaled



What did the generator learn?

- To modulate the volatility from features
  see stochastic volatility models

$$r_t^{(b)} = \sigma^{(b)} \epsilon_t^{(b)}, \text{ for a given } \eta^{(b)}$$



- $P(\sigma_{sample})$ is heavy tailed
- ACF of σ: memory $\equiv$ length of samples T

What did the discriminator learn?

Experiment

- increase scale of returns
- plot average $\hat{p}_{true}$ vs scale

:

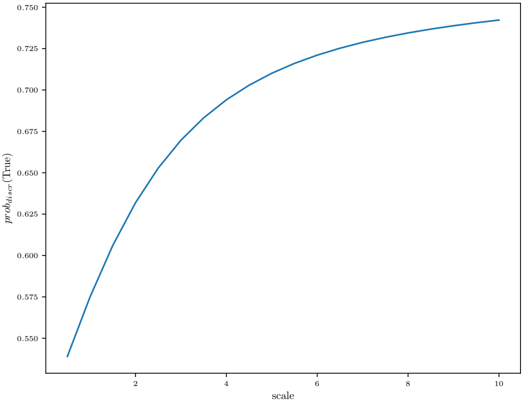
```
X_gauss=np.resize(np.random.normal(size=T*ntries),(ntries,T))
scales=np.linspace(0.5,10,20)
y_avg=[]
for myscale in scales:
    X_gauss_scaled=X_gauss*myscale
    y_gauss_scaled=discriminator.predict(X_gauss_scaled).flatten()

    y_gauss_scaled_avg=np.mean(y_gauss_scaled)
    y_avg.append(y_gauss_scaled_avg)

plt.plot(scales,np.array(y_avg))
```

What did the discriminator learn?

SCALE!



If $P(\sigma_{sample}) \simeq P(\sigma_{real})$, GAN converges (weaker conditions apply)

Return timeseries generation from models

1. Long memory: fractional Brownian motion
   $\rightarrow$ Wiener process+power-law kernel

2. Heavy tails:
   2.1 external events $\rightarrow$ heavy tails features
   2.2 herding: internal state of agents/market

$$\sigma_t = F(past) \rightarrow \text{long memory} + \text{heavy tails}$$

- Better NN architecture
  - CNN with longer memory: Temporal CN (next slide)
  - stateful neural networks: GRU, LSTMs (soon)

GANs with TCNs

Wiese et al. (2019) [preprint]

- $T = 100$

- TCN: temporal convolutional networks $\equiv$ Wavenets (Oord et al. 2016 [preprint])
  stacked CNN with dilation ($\rightarrow O(\log N)$ coefficients)

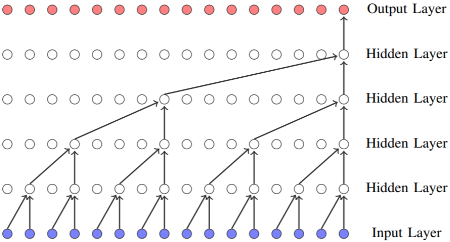


Figure 6: Vanilla TCN with 4 hidden layers, kernel size $K = 2$ and dilation factor $D = 2$ (cf. van den Oord et al. (2016)).

[keras-tcn]

GANs a scenario generators

Mittigating overfitting [link]

Improving robustness of backtesting [link]

- Only touch ONCE a given set of data when backtesting
- Tweaking a strategy and testing it again $\equiv$ data snooping
- Solutions
    1. Use statistical tools to account for multiple hypothesis testing
    2. Use another dataset

Multiple Hypothesis Testing

- For every strategy i, $H_0^{(i)}: \; E(\text{per}f_i) = 0$
- Control the fraction of false positive detection rate.
- Control the number of wrongly selected strategies.
- **Problem**: false negative rate can be very high.

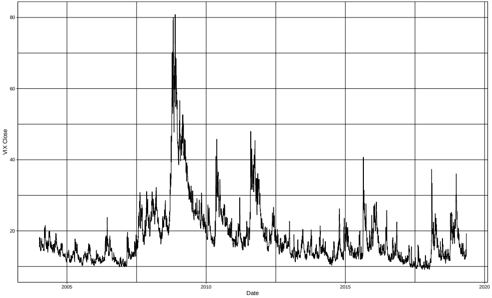
Use another dataset

Naively:

- Aim: use a strategy on US equities
- Keep 20% of equities out of sample
- Calibrate on 80%
- **Problem**: large dependence between equities
- **Problem:** large dependence between countries

Robust backtest with a single asset
VIX prediction problem

Aim: predict when VIX>15



Usual procedure

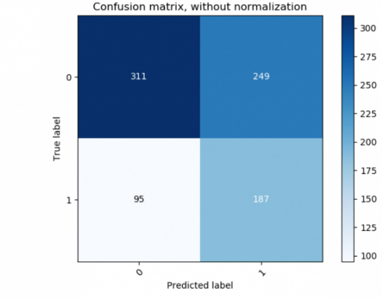
1. Sliding window
2. Tweak moving average of $\log$ VIX or stochastic volatility model

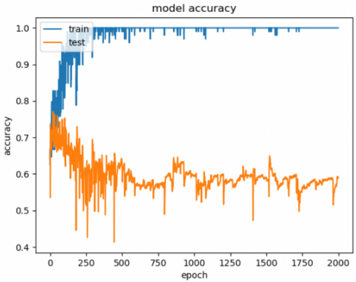
Prediction problem with Generative Model

1. Generate many VIX-like timeseries from training data set
2. Train a predictor of peaks on synthetic data
3. Test on real data

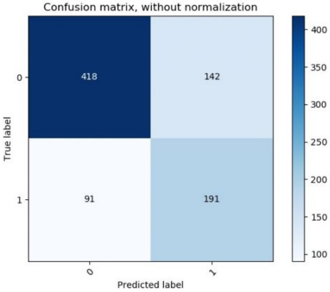
VIX>15 prediction without data augmentation

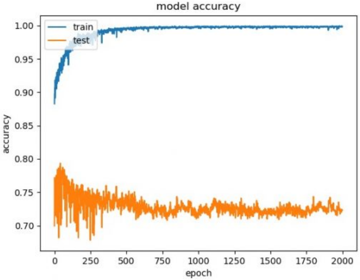
From De Meer (2019) [link]





VIX>15 prediction with data augmentation





1. **Open questions:**
   1.1 Why does it improve testing performance?
   1.2 In what respect is it not overfitting of some kind?

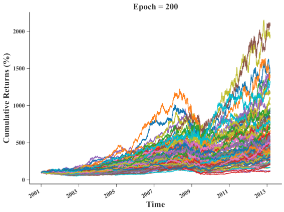
   [code]

conditional GANs

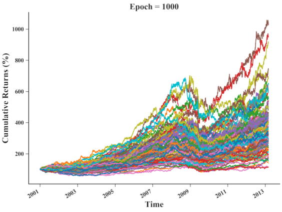
- Generate objects conditional on signal $s \in \mathbb{R}^S$
- Generator: input (η, s)
  - Discriminator: input $(G(\eta|s), s) \rightarrow P_{true}(y|s)$
- Example: scenario [link][link]
  - $s_t = (r_t, r_{t-1}, \cdots, r_{t-S})$
  - Generator: outputs $r_{t+1}, \cdots, r_{t+k}$
  - Discriminator: realistic or not

conGAN: example

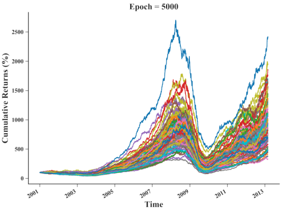
Koshiyama *et al.* (2020) [link]



(d)



(e)



(f)

What to do with scenarios generators

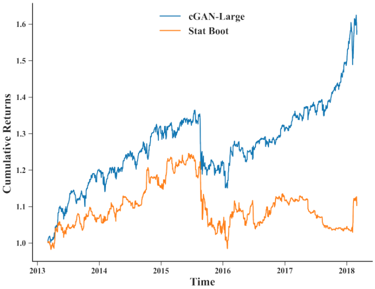
Strategy $\equiv$ model M_θ; performance on data r: $\Pi[M_\theta(r)]$

1. Robust strategy parameter selection
   - Best strategy on a single time series $\equiv$ noise fitting
   - Best strategy on many lookalike time series
     $\arg\max_\theta E(\Pi[M_\theta(r)])$
2. Aggregation of many overfitted strategies
   - Best strategy for each scenario
   - Overall strategy $=$ average strategy
   - Combination of weak learners $\rightarrow$ strong learner (see trees)

Aggregations of weak strategies

Koshiyama *et al.* (2020) [link]

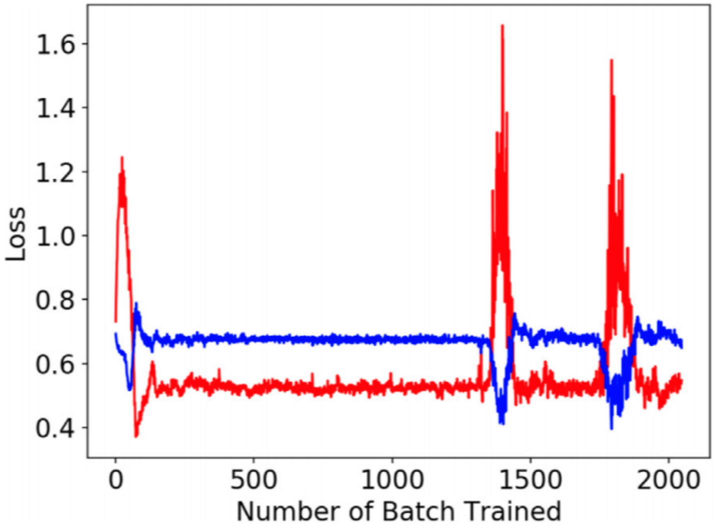
(g)



(h)

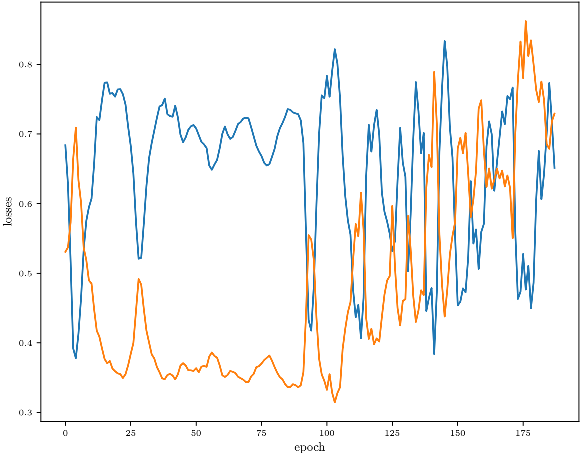
Methodological problems: convergence

Takayashi *et al.* (2019) [paper]



Convergence problems

MLP `use_bias=False` for both G and D.



How to improve convergence

Convergence problems $\equiv$ game problems $\equiv$ design problems

- Adapt G and D architecture
  e.g. MLP + `use_bias=False` for both G and D
- Noisy labelling (Salimans *et al.* 2016 [paper]):
  randomize a fraction of correct / false labels
- Better loss: compare full distributions of $\hat{p}_i$
  WassersteinGAN [link]

How to (almost) solve convergence problems

Relativistic GANs (Jolicoeur-Martineau 2018 [preprint]):

'[...] the discriminator estimates the probability that the given real data is more realistic than fake data, on average'

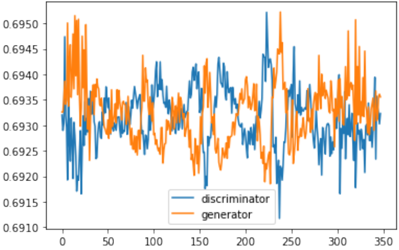
$$L_D^{RaSGAN} = -\mathbb{E}_{x_r \sim \mathbb{P}} \left[\log \left(\bar{D}(x_r) \right) \right] - \mathbb{E}_{x_f \sim \mathbb{Q}} \left[\log \left(1 - \bar{D}(x_f) \right) \right],$$

where

$$\bar{D}(x) = \begin{cases} \text{sigmoid}(C(x) - \mathbb{E}_{x_f \sim \mathbb{Q}} C(x_f)) & \text{if } x \text{ is real} \\ \text{sigmoid}(C(x) - \mathbb{E}_{x_r \sim \mathbb{P}} C(x_r)) & \text{if } x \text{ is fake.} \end{cases}$$

[keras-relativistic-gan]

Example: relGANs for timeseries



Does convergence imply correct stylized facts?

Better generators: conditional GANs

Generator's input:

- feature vector (world, past returns, . . .)
- noise vector

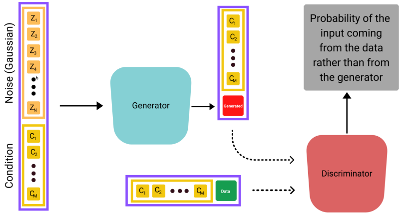


Figure 2.: An illustration of a conditional GAN pipeline.

[source]

- condGAN [preprint]
- ForGan [paper]
- Fin-GAN [preprint]

Better generators: loss

Ask what you want in generator loss

Fin-GAN: generator loss:

- Standard Generator loss
- $+$ losses over prediction horizon
  - SR out-of-sample
  - hit ratio, ...

Better generators: timeseries representation

Pdf

- $r_t \to [0,1]$: CDF: EV-GAN (Allouch *et al.* 2022 [link])
  generate heavy tails consistent with data

Representation of whole timeseries

- Path signatures
  - Signature-GAN [preprint]
  - Sig-Wasserstein-GAN [preprint]

- Wavelet scattering spectra [link]

Perturbations to representations $\to$ new scenarios

Better discriminator: what can a discriminator learn?

Without help, the discriminator easily learns

- moments (average, scale, etc)

Possibly

- Hurst exponent
- $\sum_\tau C_{|r|}(\tau)$
- Wavelets (CNNs)

Not

- quantiles, distribution tail exponent (see EV-GAN, Tail-GAN)

GAN for financial timeseries: a weird methodology

1. Train a GAN in an agnostic way
2. Convergence: happy
3. Look at timeseries.
4. Noise $\rightarrow$ test a set of stylized facts

Unconsistent aims and methodology

Example: heavy tails treatment in GAN literature

[review 2021]

[Lemzi et al. 2021]

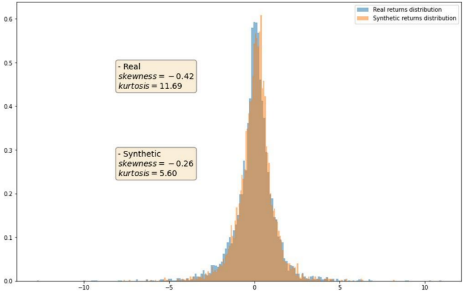
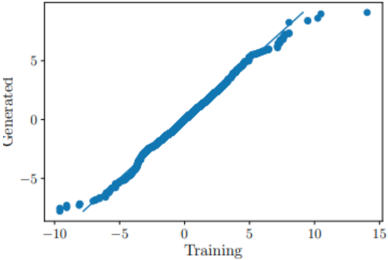


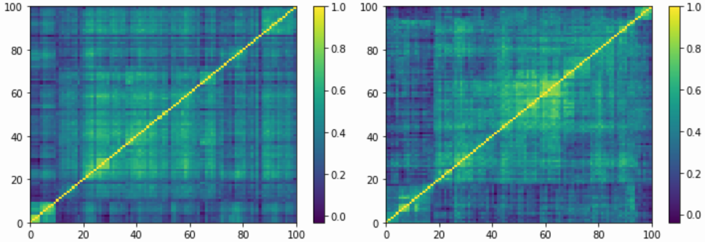
Figure 13: PDF - SAGAN



(b) Student's t-distribution with $\nu = 4$

Example: generating correlation matrices as images

Marti (2019) [preprint][link]



(Left) Empirical correlation matrix estimated on stocks returns; (Right) GAN-generated correlation matrix.

Nice looking, but

- generated matrices are not positive definite
- eigenvalue distributions not realistic

Be helpful, be explicit

- Pre-process data (remember: MLP cannot compute quantiles)
  - Hurst exponent of $|r_t|$
  - $\sum_{\tau=1} |C_r(\tau)|$: absence of linear autocorrelation
  - $\sum_{\tau>0} C_{|r|}(\tau)$: long memory
  - Non-Gaussian test (e.g. N-test [preprint])
  - Heavy-tail test: quantiles ratio
  - ...
- Post-process data: loss
  - Tail-GAN [preprint]: score-inspired loss for VaR/C-VaR

Discriminator with pre-processing abilities

Generalized moments method (GMM) [paper]

- Compute a set of K quantities $\{q_k\}$ from r_t (moments, ACF of moments, quantiles, etc)
- Implicit model calibration:
  - find model parameters θ so that $\{\hat{q}_k|\theta\} \simeq \{q_k^{\text{real data}}\}$
- Here: feed discriminator with $\{q_k|\theta\}$ that you kindly compute, plus possibly raw returns

Pre-processing: maths and stats

- math: $\rightarrow$ tf.math
  e.g. `np.sum()` $\rightarrow$ `tf.math.reduce_sum()`

- stats: import tensorflow_probability as tfp

  `tfp.stats.auto_correlation()`

How to help the discriminator: Lambda layers

- Python: lambda function
- Keras: Lambda layer: apply a function to the input of a layer
- Here: price returns $\rightarrow$ discriminator $\rightarrow$ statistics $\{a, b, c, \cdots\}$
- Discriminator: first layer is a Lambda layer that calls the function computing the statistics
- Technical hint: convert list [a,b,c] to tensor

```
return tf.transpose(tf.convert_to_tensor([a,b,c]))
```

Keras: functional models

- Several inputs, several sub-networks

- Merge inputs, outputs

```
input_merged = Concatenate()([input_conds, input_eta])

model=Dense(dim_conds,use_bias=use_bias,activation="PReLU")(input_merged)
```

- Shortcuts: merge input and model output (inception)

Technical note on model re-use: weights

```
#define model
model.compile(...)
model.fit(...)
model.saveweigths('model_weights.h5')
```

Later, elsewhere

```
#define model
model.compile(...)
model.loadweigths('model_weights.h5')   #instead of .fi
```

Technical note on model-reuse: architecture

- Each model can be described as a tree
- Model architecture saved as json [link]

Save

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weigths('model_weights.h5')
```

Load

```
with open('model.json', 'r') as json_file:
    model_descr = json_file.read()
model = model_from_json(model_descr)
model.load_weights("model_weights.h5")
```