Adversarial Attacks on Probabilistic Autoregressive Forecasting Models

Raphaël Dang-Nhu [1] **Gagandeep Singh** [1] **Pavol Bielik** [1] **Martin Vechev** [1]

Abstract

We develop an effective generation of *adversarial attacks* on neural models that output a sequence of probability distributions rather than a sequence of single values. This setting includes the recently proposed deep probabilistic autoregressive forecasting models that estimate the probability distribution of a time series given its past and achieve state-of-the-art results in a diverse set of application domains. The key technical challenge we address is effectively differentiating through the Monte-Carlo estimation of statistics of the joint distribution of the output sequence. Additionally, we extend prior work on probabilistic forecasting to the Bayesian setting which allows conditioning on future observations, instead of only on past observations. We demonstrate that our approach can successfully generate attacks with small input perturbations in two challenging tasks where robust decision making is crucial – stock market trading and prediction of electricity consumption.

1. Introduction

Deep probabilistic autoregressive models have been recently integrated into the Amazon SageMaker toolkit and successfully applied to various kinds of sequential data such as handwriting (Graves, 2013), speech and music (Oord et al., 2016a), images (Oord et al., 2016c;b) and time series from a number of different domains (Salinas et al., 2019). At a high level, given a sequence of input values (i.e., pen-tip locations, raw audio signals or stock market prices), the goal is to train a generative model that outputs an accurate sequence of next values, conditioned on all the previous values. The main benefit of such probabilistic models is that they model the joint distribution of output values, rather than predicting only a single best realization (i.e., the most likely value at each step). Predicting a density rather than just a single best value has several advantages – it naturally fits the inherently stochastic nature of many processes, allows to assess the uncertainty and the associated risk, and has been shown to produce better overall prediction accuracy when used in forecasting tasks (Salinas et al., 2019).

This Work We develop an efficient approach for generating *adversarial attacks* on deep probabilistic autoregressive models. The issue of adversarial robustness and attacks (Szegedy et al., 2013; Goodfellow et al., 2014), i.e., generating small input perturbations that lead to mispredictions, is an important problem with large body of recent work. Yet, to our best knowledge this is the first work that explores adversarial attacks in a new challenging setting where the neural network output is a sequence of probability distributions. The difficulty in generating adversarial attacks in this setting is that it requires computing the gradient of an expectation that it is too complex to be analytically integrated (Schittenkopf et al., 2000; Salinas et al., 2019) and is approximated using Monte-Carlo methods.

Differentiating through Monte-Carlo estimation of the Expectation We address the key technical challenge of efficiently differentiating through the Monte-Carlo estimation, a necessary part of generating white-box gradient based adversarial attacks, by using two techniques to approximate the gradient of the expectation. The first approach is the score-function estimator (Glynn, 1990; Kleijnen & Rubinstein, 1996) obtained by inverting the gradient and the expectation's integral. The second technique differentiates individual samples using a random variate *reparametrizatrion* and originates from the variational inference literature (Salimans et al., 2013; Kingma & Welling, 2013; Rezende et al., 2014).

Main Contributions We present the first approach for generating adversarial attacks on deep probabilistic autoregressive models by applying two techniques that differentiate through Monte-Carlo estimation of an expectation. We show that the reparametrization estimator is efficient at generating adversarial attacks and outperforms the score-function estimator by evaluating on two domains that benefit from stochastic sequential reasoning – stock market trading and electricity consumption. We make our code, datasets and scripts to reproduce our experiments available online[1].

[1]Department of Computer Science, ETH Zürich, Switzerland. Correspondence to: Raphaël Dang-Nhu <dangnhur@student.ethz.ch>.

Preprint.

[1]https://github.com/eth-sri/probabilistic-forecasts-attacks

2. Probabilistic Forecasting Models

In this section, we formally describe the probabilistic autoregressive model used in prior works and throughout this paper. Most notably, the model described in Section 2.1 is based on the recent work of DeepAR (Salinas et al., 2019), which is now an inherent part of the Amazon SageMaker toolkit. Further, in Section 2.2 we describe an extension of this model to the Bayesian setting proposed in our work.

2.1. Sequence Modeling: Preliminaries

Given a sequential process $\boldsymbol{p} = (p_t)_{1 \leq t_0 \leq T}$ and an index t_0, we consider the task of modeling the distribution of predicted (future) values $\boldsymbol{p}_{t_0:T} = (p_{t_0}, \dots, p_T)$ given the observed (past) values $\boldsymbol{p}_{1:t_0-1} = (p_1, \dots, p_{t_0-1})$. Using the chain rule, the joint distribution of predicted values conditioned on observed values $\Pr[\boldsymbol{p}_{t_0:T} | \boldsymbol{p}_{1:t_0-1}]$ can be written as a product of conditional distributions:

$$\Pr[\boldsymbol{p}_{t_0:T} | \boldsymbol{p}_{1:t_0-1}] = \prod_{i=t_0}^{T} \Pr[p_i | \boldsymbol{p}_{1:i-1}] \tag{1}$$

In deep autoregressive models, a neural network is used to approximate the conditional distribution $\Pr[p_i | \boldsymbol{p}_{1:i-1}]$ by a parametric distribution $q_\theta[p_i | \boldsymbol{p}_{1:i-1}]$ specified by learnable parameters θ. This yields a joint model:

$$q_\theta[\boldsymbol{p}_{t_0:T} | \boldsymbol{p}_{1:t_0-1}] = \prod_{i=t_0}^{T} q_\theta[p_i | \boldsymbol{p}_{1:i-1}] \tag{2}$$

This decomposed form for the joint distribution is general and independent of the particular neural architecture chosen for q_θ. In principle, any type of sequential model can be used including well-known architectures such as LSTMs (Hochreiter & Schmidhuber, 1997), Temporal Convolutional Networks (Bai et al., 2018) or Transformers (Vaswani et al., 2017). Next, we describe a LSTM based instantiation of probabilistic autoregressive models.

Probabilistic Autoregressive Models Let h be a function implemented by a LSTM network. Given h, we compute the hidden state $\boldsymbol{h}_i = h(\boldsymbol{h}_{i-1}, p_{i-1}, \theta)$ for each time-step, conditioned on the previous hidden state $\boldsymbol{h}_{i-1}$, previous input value p_{i-1} and the network parameters θ. Then, the hidden state $\boldsymbol{h}_i$ is used to generate a set of parameters $\psi(\boldsymbol{h}_i)$ that specify a distribution with density $\ell_{\psi(\boldsymbol{h}_i)}(p_i)$, giving the following form for the conditional distribution:

$$q_\theta[\boldsymbol{p}_{t_0:T} | \boldsymbol{p}_{1:t_0-1}] = \prod_{i=t_0}^{T} \ell_{\psi(\boldsymbol{h}_i)}(p_i) \tag{3}$$

The main difference here compared to the non-probabilistic models is that the network predicts parameters of a distribution rather than a single value. Commonly used distributions in prior works are Gaussian distribution for real-valued data or the negative binomial distribution for count data (Salinas et al., 2019). When using Gaussian distribution, $\psi(\boldsymbol{h}_i)$ has two components $\psi(\boldsymbol{h}_i) = (\mu(\boldsymbol{h}_i), \sigma(\boldsymbol{h}_i))$, that correspond to the mean and the standard deviation, respectively. The density is defined as:

$$\ell_{\psi(\boldsymbol{h})}(p) = \frac{1}{\sigma(\boldsymbol{h})\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{p - \mu(\boldsymbol{h})}{\sigma(\boldsymbol{h})}\right)^2\right] \tag{4}$$

Note, that the choice of a Gaussian distribution corresponds to the assumption that each value is normally distributed conditioned on past values – a hypothesis which has to be assessed per application domain. In what follows, to make a clear distinction between the network inputs and outputs, we use $\boldsymbol{x} = (x_1, \dots, x_{t_0-1}) := \boldsymbol{p}_{1:t_0-1}$ to denote the inputs (i.e., observed values) and $\boldsymbol{y} = (y_1, \dots, y_{T+1-t_0}) := \boldsymbol{p}_{t_0:T}$ to denote the outputs (i.e., the predicted values).

Inference Performing inference for probabilistic autoregressive models corresponds to characterizing the joint distribution of the output sequence $\boldsymbol{y}$. This includes estimating n-steps ahead both, the mean and the standard deviation of the value y_n, via the first and second moments $\mathbb{E}_{q_\theta[\boldsymbol{y}|\boldsymbol{x}]}[y_n]$ and $\mathbb{E}_{q_\theta[\boldsymbol{y}|\boldsymbol{x}]}[y_n^2]$. More generally, given the space of output sequences $\mathcal{Y}$ and any statistic $\chi : \mathcal{Y} \to \mathbb{R}$, we consider the task of estimating the expectation $\mathbb{E}_{q_\theta[\boldsymbol{y}|\boldsymbol{x}]}[\chi(\boldsymbol{y})]$. The main challenge here is the complexity of the underlying integral on the distribution $q_\theta[\boldsymbol{y}|\boldsymbol{x}]$, which is in general not analytically solvable.

During training, one can either use scheduled sampling (Bengio et al., 2015), where a single sample from the distribution $\ell_{\psi(\boldsymbol{h}_i)}(\cdot)$ is used, or avoid this issue completely by using teacher forcing (Williams & Zipser, 1989), where the deterministic ground truth value for y_i is fed back into the network in the next time-step. This setting is solvable but only because the prediction is only for a single next step. However, when performing iterated prediction at test time, the value used in the feedback loop is sampled from the predicted distribution of $y_i \sim \ell_{\psi(\boldsymbol{h}_i)}(\cdot)$. Therefore, the next hidden state $\boldsymbol{h}_{i+1}$ depends on the randomness introduced in sampling y_i. This yields an arbitrarily complex form for the joint distribution $q_\theta[\boldsymbol{y}|\boldsymbol{x}]$. To address this issue, prior works perform Monte-Carlo inference (Schittenkopf et al., 2000; Salinas et al., 2019) to approximate the expectation as:

$$\mathbb{E}_{q_\theta[\boldsymbol{y}|\boldsymbol{x}]}[\chi(\boldsymbol{y})] \simeq \frac{1}{L} \sum_{l=1}^{L} \chi(\boldsymbol{y}^l) \tag{5}$$

That is, the Monte-Carlo estimations of the expected value of $\chi(\boldsymbol{y})$ is computed using L generated samples $\boldsymbol{y}^1, \dots, \boldsymbol{y}^L$ for the output sequence.

2.2. Extension to Bayesian Setting

We extend the probabilistic autoregressive models presented in this section to the Bayesian setting, where the output sequence can be conditioned on arbitrary values (i.e., both past and from the future). Formally, we define an observation function $\gamma\colon \mathbb{R}^m \to \{\texttt{true}, \texttt{false}\}$, which takes as input a sequence of values and outputs a boolean denoting whether the observation holds. As an example, using $\gamma(\boldsymbol{y}) = (y_{10} \geq 3)$ denotes that we would like the model to predict values $y_{1:9}$ conditioned both on the inputs $\boldsymbol{x}$ as well as on the observation $y_{10} \geq 3$.

There are several cases for which the Bayesian setting is useful: *(i)* some of the data is missing (e.g., due to sensor failures), *(ii)* to allow encoding prior beliefs about the future evolution of the process, or *(iii)* to evaluate complex domain-specific statistics. The financial domain offer good examples for (iii), in pricing exotic derivatives such as barrier options (Rich, 1994) whose existence depends upon the underlying asset's price breaching a preset barrier level.

To remove clutter, in the remainder of the paper we will use $z = \gamma(\boldsymbol{y})$ to denote the output of the observation function when evaluated on $\boldsymbol{y}$. In this Bayesian setting, the expectation $\mathbb{E}_{q_\theta[\boldsymbol{y}|\boldsymbol{x},z]}[\chi(\boldsymbol{y})]$ can be estimated via Monte-Carlo importance sampling as:

$$\mathbb{E}_{q_\theta[\boldsymbol{y}|\boldsymbol{x},z]}[\chi(\boldsymbol{y})] \simeq \frac{\sum_{l=1}^{L} \chi(\boldsymbol{y}^l) q_\theta[z|\boldsymbol{x}, \boldsymbol{y}^l]}{\sum_{l=1}^{L} q_\theta[z|\boldsymbol{x}, \boldsymbol{y}^l]} \tag{6}$$

This corresponds to generating samples from the prior distribution $q_\theta[\boldsymbol{y}|\boldsymbol{x}]$, and reweighing with Bayes rules. Note, that this formula includes the former by using $z = \texttt{true}$.

3. Adversarial Attacks on Probabilistic Forecasting Models

In this section, we present our approach for generating adversarial attacks on deep probabilistic forecasting models. We start by formally defining the problem statement suitable for this setting and then we describe two practical adversarial attacks that address it.

Adversarial Examples Recall that in the canonical classification setting, adversarial examples are typically found by solving the following optimization problem (Szegedy et al., 2013; Papernot et al., 2016; Carlini & Wagner, 2017):

$$\arg\min_{\boldsymbol{\delta}} ||\boldsymbol{\delta}|| \text{ s.t } f(\boldsymbol{x} + \boldsymbol{\delta}) = t \tag{7}$$

where f is a classifier, $\boldsymbol{x}$ is an input (i.e., an image), t is the desired adversarial output (the target), and $\boldsymbol{\delta}$ is the minimal perturbation (according to a given norm) applied to the input image such that the classifier f predicts the desired output t.

To make the above formulation applicable to probabilistic forecasting models we perform two standard modifications – *(i)* we replace the hard equality constraint with an easier to optimize soft constraint that captures the distance between real values, and *(ii)* we replace the single value output with the expected value of a given statistic of the output joint probability distribution. Applying the first modification leads to the following formulation:

$$\arg\min_{\boldsymbol{\delta}} \phi(f(\boldsymbol{x} + \boldsymbol{\delta}), t) \text{ s.t } ||\boldsymbol{\delta}|| \leq \epsilon \tag{8}$$

That is, we use a soft constraint that minimizes the distance between the target and the predicted value, subject to a given tolerance ϵ on the perturbation norm. Applying the second modification corresponds to replacing $f(\boldsymbol{x} + \boldsymbol{\delta})$ with the expected value $\mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})]$, where z is an observation over outputs as defined in Section 2.2, and $\chi : \mathbb{R}^m \to \mathbb{R}$ is a statistic of the output sequence. Overall, this leads to the following problem statement.

Problem Statement Let $f\colon \mathbb{R}^n \to \mathcal{D}(\mathbb{R}^m)$ be a function (i.e., a probabilistic neural network) that takes as input a sequence of values $\boldsymbol{x} \in \mathbb{R}^n$ and outputs a probability distribution $f(\boldsymbol{x})$ with density $q[\boldsymbol{y}|\boldsymbol{x}]$ that can be sampled from to obtain a concrete output sequence $\boldsymbol{y} \in \mathbb{R}^m$. Given an observation variable z, a statistic $\chi : \mathbb{R}^m \to \mathbb{R}$ of the network output distribution and a target t, the goal of the adversarial attack is to find a perturbation $\boldsymbol{\delta}$ that solves the following optimization problem:

$$\arg\min_{\boldsymbol{\delta}} \phi(\mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})], t) \text{ s.t } ||\boldsymbol{\delta}|| \leq \epsilon \tag{9}$$

3.1. Practical Attack on Probabilistic Networks

The constrained minimization problem defined in Equations 8 and 9 has repeatedly been identified as very difficult to solve in the adversarial attacks literature (Szegedy et al., 2013; Carlini & Wagner, 2017). As a result, we instead follow the approach of Szegedy et. al. 2013 and solve an adjusted optimization problem. Given a real hyper-parameter $c \in \mathbb{R}^+$, we aim at minimizing:

$$\text{obj}(\boldsymbol{\delta}) := ||\boldsymbol{\delta}|| + c \cdot \phi(\mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})], t) \tag{10}$$

via gradient-descent. The attack is run with different values of c, and the final value is chosen to ensure that the hard constraint $||\boldsymbol{\delta}|| \leq \epsilon$ is satisfied.

While optimizing the objective function in Equation 9 is standard (Szegedy et al., 2013; Goodfellow et al., 2014; Kurakin et al., 2017b;a), the crucial aspect in ensuring efficient gradient descent is to obtain a good estimation of the objective function's gradient. In particular, this involves computing:

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})]$$

The difficulty of computing this gradient comes from the fact that the expectation can not be analytically computed, but only approximated via Monte-Carlo methods. Informally, it raises the question of how to efficiently differentiate through the Monte-Carlo estimation. We compare two different ways of performing this differentiation, described next.

3.1.1. Score-function Estimator

The first approach is to express the gradient of the expectation as an expectation over the distribution $q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]$ by inverting the gradient and integral, and estimate the resulting expectation via Monte-Carlo methods. This technique is known under different names in the literature: score-function method (Glynn, 1990), REINFORCE (Williams, 1992), or log-derivative trick. Below we show how this applies to our setting.

Score-function Estimator 3.1. *In the general Bayesian setting where $\boldsymbol{y} \sim q[\cdot|\boldsymbol{x}+\boldsymbol{\delta}, z]$, the score-function gradient estimator of the expected value of $\chi(\boldsymbol{y})$ is:*

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] \simeq \frac{\sum_{l=1}^{L} \chi(\boldsymbol{y}^l) q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l] \nabla_{\boldsymbol{\delta}} \log(q[\boldsymbol{y}^l|\boldsymbol{x}+\boldsymbol{\delta}, z])}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]}$$

where $\boldsymbol{y}^l$ is sampled from the prior distribution $q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}]$, and $q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}]$ denotes the probability that z is true knowing that $\boldsymbol{y}^l$ is generated.

The proof of 3.1 is given in the supplementary material. Note that in the non-Bayesian setting where the observation z is always true, we have $q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l] = 1$ and we obtain a simpler form:

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}]}[\chi(\boldsymbol{y})] \simeq \frac{1}{L} \sum_{l=1}^{L} \chi(\boldsymbol{y}^l) \nabla_{\boldsymbol{\delta}} \log(q[\boldsymbol{y}^l|\boldsymbol{x}+\boldsymbol{\delta}])$$

While this estimator allows for generating adversarial perturbations, we observe that it has two drawbacks – high-variance and high sampling complexity.

High Variance Score-function estimators typically lead to slow convergence because they suffer from high variance (Ranganath et al., 2014). It is due to the fact that they operate in a black-box way with respect to the gradients of the network f and the statistic χ.

Complexity of the Bayesian Setting The score-function requires computing the gradient of $q[\boldsymbol{y}^l|\boldsymbol{x}+\boldsymbol{\delta}, z]$ with respect to $\boldsymbol{\delta}$. This is always possible in the special case when the observation z is constantly true, however in the general setting this might require another step of sampling, which makes the estimator overly complex.

3.1.2. Reparametrization Estimator

The second estimator is based on the *reparametrization trick*. It reparametrizes the output distribution $\boldsymbol{y}$ in terms of auxiliary random variables whose distribution does not depend on $\boldsymbol{x}$, in order to make individual samples $\boldsymbol{y}^l$ differentiable with respect to $\boldsymbol{\delta}$. The differential $\partial \boldsymbol{y}^l / \partial \boldsymbol{\delta}$ has *a priori* no specific meaning when the distribution from which $\boldsymbol{y}^l$ is sampled depends on $\boldsymbol{\delta}$. However, if $\boldsymbol{y}^l \sim q[\cdot|\boldsymbol{x}+\boldsymbol{\delta}]$ can be reparametrized as $\boldsymbol{y}^l = g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta})$, where $\boldsymbol{\eta}$ is a random variable whose distribution is independent from $\boldsymbol{\delta}$, then it makes sense to define the differential of $\boldsymbol{y}^l$ with respect to $\boldsymbol{\delta}$ as $\partial \boldsymbol{y}^l / \partial \boldsymbol{\delta} = \partial g_{\boldsymbol{x}} / \partial \boldsymbol{\delta}$.

Reparametrization estimators were first proposed as a way of mitigating the variance problems of score-function estimators (Salimans et al., 2013; Kingma & Welling, 2013; Rezende et al., 2014). However, to our best knowledge, they have not been used in a Bayesian setting where the estimator to differentiate uses importance sampling.

Reparametrization Estimator 3.2. *Assume there exists a differentiable transformation $g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta})$ such that the random variable $\boldsymbol{y} \sim q[\cdot|\boldsymbol{x}+\boldsymbol{\delta}]$ can be reparametrized as $\boldsymbol{y} = g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta})$, where $\boldsymbol{\eta}$ is an independent random variable whose marginal distribution $p(\boldsymbol{\eta})$ is independent from $\boldsymbol{\delta}$. Then the importance sampling reparametrization estimator of the expectation's gradient is:*

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] \simeq \nabla_{\boldsymbol{\delta}} \left(\frac{\sum_{l=1}^{L} \chi(g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)) q[z|\boldsymbol{x}+\boldsymbol{\delta}, g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)]}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)]} \right)$$

where for $1 \leq l \leq L$, $\boldsymbol{\eta}^l$ is sampled from the distribution $p(\boldsymbol{\eta})$, and $\boldsymbol{y}^l = g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)$.

A proof of 3.2 is given in the supplementary material.

3.2. Reparametrization of Probabilistic Networks

Here, we discuss the question of reparametrizing probabilistic autoregressive models. The stochasticity of such architectures comes from the iterated sampling of $y_i \sim \ell_{\psi(\boldsymbol{h}_i)}(\cdot)$. Assuming a Gaussian likelihood, the value y_i follows a normal distribution $\mathcal{N}(\mu(\boldsymbol{h}_i)|\sigma(\boldsymbol{h}_i)^2)$. Let $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_{T-t_0})$ be a standard normal random vector, i.e., all of its components are independent and each is a zero-mean unit-variance normally distributed random variable. Iteratively writing:

$$y_i \sim \mu(\boldsymbol{h}_i) + \eta_i \cdot \sigma(\boldsymbol{h}_i)$$

for all i such that $1 \leq i \leq T - t_0$ yields a valid reparametrization. This simple reasoning applies to the particular implementation described in (Salinas et al., 2019) and adapts readily to any kind of likelihood parameterized by location and scale, such as Laplace or logistic distribution.

The case of mixture density likelihoods is more complex as they do not enter this category of "location-scale" distributions, and their inverse cumulative density function does not admit a simple closed form. The problem of how to adapt reparametrization to mixture densities is outside the scope of this paper, and we refer to the relevant literature (Graves, 2016; Figurnov et al., 2018; Jankowiak & Obermeyer, 2018) for more information about this question.

4. Case Study: Stock Market Trading

In this section, we apply the probabilistic autoregressive models and discuss the types of adversarial attacks in the domain of financial decision making.

Output Sequence Statistics While a given machine learning model is typically trained to predict the future stock prices given its past, various statistics of the output sequence are used in downstream algorithmic trading and option pricing tasks. This is the reason why the approach presented so far already assumed presence of such statistics. The different statistics used in our evaluation are shown in Table 2 and include predicting cumulated stock return, pricing derivatives such as European call and put options (Black & Scholes, 1973), as well as an example of a binary statistic that predicts the success probability of limit orders (Handa & Schwartz, 1996). All statistics are defined with respect to the last known price, denoted as x_{-1}.

4.1. Probabilistic Autoregressive Models Performance

Before we show the effectiveness of generative adversarial attacks, we first demonstrate that using probabilistic autoregressive models leads to state-of-the-art results. We use two baselines as the current state-of-the-art for financial predictions: LSTM networks (Fischer & Krauss, 2018), and Temporal Convolutional Networks (TCN) (Borovykh et al., 2017). We provide detailed description of all the training hyper-parameters, the dataset used (S&P 500) and extended version of all the experiments in the supplementary material.

Long-Short Trading Strategies Given a prediction horizon $h \in [\![1, 10]\!]$, we analyze the characteristics of the following portfolio: at time-step t, buy (long) the k stocks for which the model predicts the highest gain, and sell (short) the k stocks with the highest predicted loss. This task is a generalization of the one presented in (Fischer & Krauss, 2018), where only direct prediction ($h = 1$) is considered.

Formally, we consider the cumulative return statistic $\chi_x(\boldsymbol{y}) = y_h/x_{-1} - 1$ of the output sequence, which corresponds to the gain of investing one dollar in the stock at time t, and then selling at time $t + h$. In a non-Bayesian setting, we estimate the expectation $\mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})}[\chi_x(\boldsymbol{y})]$ via Monte-Carlo sampling for each stock, and buy (or sell) the stocks for which the estimate is the highest (or the lowest). Note that this setting also applies to the deterministic baselines, it suffices to consider that $q(\boldsymbol{y}|\boldsymbol{x})$ is a Dirac distribution centered in the deterministic prediction.

Table 1. Financial gain on algorithmic trading tasks for different horizons h and portfolio sizes k (expressed per mille ‰). An extended version is included in the supplementary material.

Params		Non-probabilistic		Probabilistic
h	k	TCN (Borovykh et al., 2017)	LSTM (Fischer & Krauss, 2018)	LSTM This Work
1	10	3.53 (± 0.49)	**4.89 (± 0.39)**	4.37 (± 0.51)
1	30	1.74 (± 0.30)	**2.41 (± 0.24)**	2.35 (± 0.23)
1	100	0.70 (± 0.19)	0.93 (± 0.1)	**0.99 (± 0.12)**
5	10	5.57 (± 1.93)	8.86 (± 1.03)	**9.02 (± 1.52)**
5	30	3.40 (± 1.36)	5.34 (± 0.61)	**5.66 (± 0.87)**
5	100	1.64 (± 0.78)	2.47 (± 0.28)	**2.70 (± 0.48)**
10	10	6.21 (± 3.52)	**9.68 (± 1.58)**	9.55 (± 2.30)
10	30	4.28 (± 2.69)	6.39 (± 0.76)	**6.63 (± 1.70)**
10	100	2.09 (± 1.58)	3.12 (± 0.52)	**3.48 (± 1.03)**

The performance of all models is summarized in Table 1. We can see that the TCN is consistently outperformed by both probabilistic and non-probabilistic LSTM models. For the probabilistic model, we observe that it is generally outperformed by the LSTM for direct prediction ($h = 1$), but it has better performance on iterated prediction ($h > 1$), provided that enough samples are used for Monte-Carlo estimation. We observe that a large number of samples (at least 1000) is required to match the LSTM performance. We provide extended evaluation results in the supplementary material, including the effect of the number of samples.

Quality of the Probabilistic Forecast Table 2 shows evaluation of the forecast quality for each of the statistics described earlier. To compare deterministic and probabilistic forecast, we use as metric the Ranked Probability Skill (RPS) (Weigel et al., 2007) of the prediction. However, because it applies only to predictions with finite output space, we first discretize the output before we apply RPS[2]. Here, lower score means better prediction. We provide extended evaluation results in the supplementary material, including multiple different values for the horizon h, price π and the number of samples for Monte-Carlo estimation.

[2]There exists a continuous version (Gneiting & Raftery, 2007), but it is impractical for our setting because of the memory consumption of computing the score: we favor metrics computable in an on-line fashion with respect to the sampling process.

Table 2. Definition of various output sequence statistics used in our work (left) and performance of various models used to predict them (right). h is the prediction horizon and π the price of the option. The comparison metric is Ranked Probability Skill (Weigel et al., 2007) of the prediction (lower scores correspond to better predictions). An extended version is provided in the supplementary material.

Statistics		Params		Non-probabilistic		Probabilistic
Name	$\chi(\boldsymbol{y})$	h	π	`TCN` (Borovykh et al., 2017)	`LSTM` (Fischer & Krauss, 2018)	`LSTM` This Work
Cum. Return	$y_h/x_{-1} - 1$	10	-	1.548 ($\pm$ 0.029)	1.541 ($\pm$ 0.019)	**1.002** ($\pm$ **0.008**)
European Call	$\max(0, y_h/x_{-1} - \pi)$	10	1	1.122 ($\pm$ 0.002)	1.121 ($\pm$ 0.002)	**0.982** ($\pm$ **0.005**)
European Put	$\max(0, p - y_h/x_{-1})$	10	1	1.302 ($\pm$ 0.003)	1.300 ($\pm$ 0.002)	**0.974** ($\pm$ **0.005**)
Limit Sell	$\mathbb{1}\left[\max(\boldsymbol{y}_{1:h})/x_{-1} \geq \pi\right]$	10	1.05	1.516 ($\pm$ 0.001)	1.514 ($\pm$ 0.002)	**0.940** ($\pm$ **0.006**)
Limit Buy	$\mathbb{1}\left[\min(\boldsymbol{y}_{1:h})/x_{-1} \leq \pi\right]$	10	0.95	1.412 ($\pm$ 0.000)	1.410 ($\pm$ 0.001)	**0.958** ($\pm$ **0.008**)

4.2. Market Manipulations

The possibility of artificially influencing stock prices to make profit has always been a major problem of financial markets (Allen & Gale, 1992; Diaz et al., 2011; Öğüt et al., 2009). In our work, we focus on trade-based manipulation, in which a trader attempts to manipulate the price of a stock only by buying and then selling, without taking any other publicly observable action. The core of such an attack is to anticipate the reactions of other agents to a provoked market event, in order to drive the price up or down. In order to decrease the cost and visibility of the attack, an additional constraint for the manipulating trader is to minimize the amplitude of the perturbation. This creates a natural connection with finding adversarial perturbations over the inputs $\boldsymbol{x}$.

Adversarial Attacks Scenario To measure the perturbation size, we choose a variant of the Euclidean norm specifically tailored to stock price data, defined in Equation 11, where each component is normalized by the corresponding price x_i. This normalization aims at capturing the fact that stock prices are fixed for an arbitrary unit quantity of the underlying asset, and thus should be invariant with respect to multiplication by a scalar. Besides, we add a box constraint to the perturbed prices such that they remain positive. This constraint is enforced using projected gradient descent.

$$||\boldsymbol{\delta}||_{\boldsymbol{x}} = \left(\sum_{i=1}^{|\boldsymbol{x}|} \left(\frac{\delta_i}{x_i}\right)^2\right)^{1/2} \tag{11}$$

5. Experimental Results

In this section, we evaluate the effectiveness of our approach for generating adversarial attack on probabilistic autoregressive models. The two key results of our evaluation are:

- The reparametrization estimator leads to significantly better adversarial examples (i.e., with smaller perturbation norm ϵ) than the score-function estimator.
- The reparametrization estimator successfully generates adversarial attacks for a number of different tasks. For example, using a small perturbation norm $\epsilon = 0.016$[3] the attack is powerful enough to cause financial loss when applied to stock market trading.

Datasets We evaluate on the following two datasets:

S&P 500 dataset, which contains historical prices of S&P 500 constituents from 1990/01 to 2000/12. We consider study periods of four consecutive years. The first three years serve as training data, while the last year is used for out-of-sample testing. The different periods have non-overlapping test years, resulting in eight different periods (for each we train four different models) with test year going from 1993 to 2000. We generate input-output samples by considering sequences of 251 consecutive daily prices for a fixed constituent. The first 241 prices serve as input $\boldsymbol{x}$, while the last 10 are the ground truth output $\boldsymbol{y}$. We ensure that output sequences from the training and test sets do not overlap and reserve $\approx 15\%$ of training samples as a validation set. We use the same preprocessing as in prior work (described in supplementary material) and train our own probabilistic autoregressive model (described in Section 2). The order of magnitude of the cumulated test set size is 10^6.

UCI electricity dataset[4], which contains the electricity consumption of 370 households from 2011 to 2014, downsampled to hourly frequency for the measurements. For this dataset we reuse an existing implementation and already trained models provided by Zhang and Jiang[5]. The model is trained on data from 2011/01 to 2014/08 (included) and we perform the attack on test samples from 2014/09. The

[3] Here, $\epsilon = 0.016$ corresponds to perturbing one price in the sequence by 1.6%, or 10 prices by 0.51%, or 100 prices by 0.16%.

[4] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

[5] https://github.com/zhykoties/TimeSeries

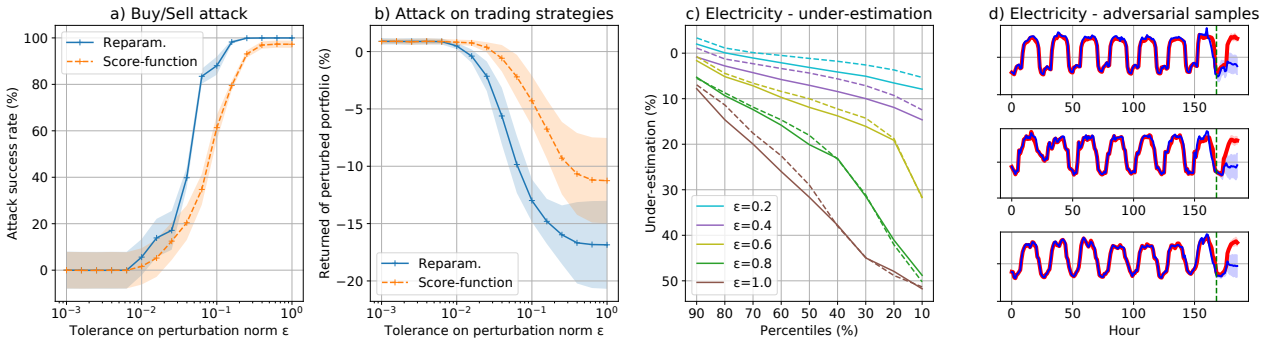

Figure 1. a) Success rate of the classification attack for different perturbation norms. b) Impact of the adversarial attack against trading strategies on financial gain, with portfolio size of $k = 10$. For both a) and b), standard deviation (shaded) is computed across test years c) Under-estimation of electricity consumption for reparametrization (continuous) and score-function (dashed) estimators. For example, the graph shows that with $\epsilon = 1.0$, the attack using reparametrization estimator leads to under-estimation of at least 20% (y-axis) for 70% of samples (x-axis). d) Under-estimation adversarial samples for the electricity dataset, with $\epsilon = 0.9$. Red curve is the original sample, blue curve is the generated adversarial sample. The vertical dashed line separates the input sequence from the network's prediction.

input sequence consists of 168 consecutive measurements, and the network predicts the next 24 values (corresponding to the next day). The total number of test samples is 2590.

Experimental Setup We performed all experiments on a machine running Ubuntu 18.04, with 2.00GHz Intel Xeon E5-2650 CPU and using a single GeForce RTX 2080 Ti GPU. For the S&P500 dataset, each model's training time is under one hour, and running the attack on one model for all test-set elements of a period takes approximately 24 hours. For the electricity dataset, running the attack on a batch of 256 test sequences takes approximately three hours.

5.1. Attacks on Buy/Sell Classification

We start by considering a classification task on the S&P dataset where each sample is classified as *buy*, *sell* or *uncertain*. For each stock, we predict the cumulated return using the statistic $\chi(\boldsymbol{y}) = y_h/x_{-1} - 1$. Then, let τ be a threshold used to decide whether to buy or sell the stock. Concretely, if the 95% confidence interval (assuming Student's t-distribution) of the estimation $\chi(\boldsymbol{y})$ is entirely above τ, the stock is classified as buy. If it is entirely below τ, it is classified as sell. Finally, if τ is inside the confidence interval, the classification is uncertain. We set τ to be the average over all stocks of the ground truth cumulated return, which leads to roughly balanced decisions to buy and sell.

We attack the statistic χ twice. First, we perturb all samples initially classified as buy or uncertain, in order to make it classify as sell. Similarly, we perturb all samples initially classified as sell or uncertain, in order to make it classify as buy. The target of the attack is set as $\tau + \lambda$ for the buy attack and $\tau - \lambda$ for the sell attack. We fix $\lambda = 0.03$ in our experiments. This is aimed at making the 95% confidence interval fit entirely in the buy (resp. sell) zone. Indeed, with 10^4 samples, the interval width order of magnitude is 0.01.

The results without a Bayesian observation z are summarized in Figure 1 a) and show that the reparametrization estimator is significantly better at generating adversarial examples that the score-function estimator. For example, using $\epsilon = 0.1$ the reparametrization estimator attack succeeds in 16% more cases. The reparametrization estimator can also successfully attack the model when considering a Bayesian setting with similar results. We include such experiment that uses a smaller horizon $h = 5$ and an observation $y_{10}/x_{-1} = \gamma$ in the supplementary material.

5.2. Attacks on Long-Short Trading Strategies

Next, we evaluate the impact of attacking the cumulated return statistic $\chi(\boldsymbol{y}) = y_h/x_{-1} - 1$ on the financial gain of the long-short trading strategy described in Section 4.1. We suppose that the attacker is allowed to perturb all inputs at test time without changing the corresponding ground truth outputs, with a maximum tolerance on the perturbation norm. We consider a horizon of $h = 10$ and different portfolio sizes k. Given a ground truth output $\tilde{\boldsymbol{y}}$, the target is set to be $t = \tau - \alpha \cdot (\chi(\tilde{\boldsymbol{y}}) - \tau)$, where τ is the buy/sell threshold defined previously, and $\alpha > 0$ is a scaling factor that rescales the ground truth output to the prediction range of the network. Intuitively, this attack target corresponds to reversing the prediction of the network around its average, in order to swap the top and flop k stocks.

We report the return of the perturbed portfolios in Figure 1 b). We observe that the reparametrization estimation is again significantly better compared to the score-function estimator. Additionally, in both experiments the thresholds for appearance of adversarial perturbation to some of the samples is approximately $\epsilon \approxeq 10^{-2}$.

5.3. Attacks on Electricity Consumption Prediction

We perturb each input sequence twice: in order to make the consumption forecast abnormally high (resp. low). We designate these as over-estimation (resp. under-estimation) attack. The attacked statistic is $\chi(\boldsymbol{y}) = y_h$, with $h = 18$. Given an input $\boldsymbol{x}$, we first approximate the expected value $y^* = \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}]}[\chi(\boldsymbol{y})]$, and set the target to $t = (1 \pm 0.5) \cdot y^*$ to cause over or under-estimation.

We show the attack success for different perturbation tolerances in Figure 1 c). We observe that the reparametrization estimator (continuous line) yields stronger under-estimation of the prediction than the score-function estimator (dashed-line) In Figure 1 d), we give examples of generated adversarial samples for the under-estimation attack. We observe a recurrent pattern in the under-estimation attack, where the perturbed prediction matches closely the original prediction for the first time-steps, but eventually becomes significantly inferior. In the supplementary material, we provide similar figures for the over-estimation attack.

6. Related Work

Probabilistic Autoregressive Forecasting Models Probabilistic autoregressive forecasting models have been used in diverse applications. Schittenkopf et al. (2000) developed the Recurrent Mixture Density Network (RMDN) to predict stock prices volatility iteratively, and were the first to propose using Monte-Carlo methods for iterative prediction. RMDN is based on a vanilla recurrent neural network coupled with Gaussian mixture likelihood. The recent `DeepAR` architecture (Salinas et al., 2019) uses several LSTM layers with Gaussian likelihood, and has been applied to forecasting electricity consumption, car traffic and business sales. Follow-up work (Chen et al., 2019) considers an alternative TCN-based architecture. Our characterization of probabilistic forecasting models encompasses these different architectures, and presents the following novelties *(i)* it generalizes inference to any statistic of the output sequence, and *(ii)* it extends the prior work to a Bayesian inference setting.

Stock-Market Prediction Various methods have been applied for predicting future stock prices including random forests, gradient-boosted trees, or logistic regression. Two notable deterministic neural models applied to this task are TCN (Borovykh et al., 2017) and LSTM (Fischer & Krauss, 2018), which achieved state-of-the-art results. In our work, we trained a probabilistic autoregressive model for the same task and achieved comparable or even better results. Further, there exists a parallel line of work that performs density estimation, but apart from the RMDN (Schittenkopf et al., 2000), most papers restrict to prediction one-step ahead (Ormoneit & Neuneier, 1996), or to less-expressive and solvable dynamics such as the GARCH (Duan, 1995).

Adversarial Attacks A growing body of recent work on adversarial attacks deals with generating small input perturbations causing mis-predictions (see (Wiyatno et al., 2019) for a survey). The objective function defined in Equation 9 is standard in generating adversarial examples (Szegedy et al., 2013; Carlini & Wagner, 2017). However, to the best of our knowledge this is the first time that adversarial attacks are applied to probabilistic autoregressive models. The most related work is the adversarial training of smoothed classifiers (Salman et al., 2019), where the noise applied to the input leads to a stochastic behavior.

Robust Algorithms for Financial Decision Making The adoption of machine learning in financial decision making makes it crucial to develop algorithms robust against small environment variations. Recent work here include robust assessment of loan applications (Ballet et al., 2019), deepfakes on accouting journal entries (Schreyer et al., 2019), robust inverse reinforcement learning on market data (Roa-Vicens et al., 2019). Adversarial attacks against stock-market prediction algorithms was studied by (Feng et al., 2018). Compared to the latter, our work is the first to operate on a probabilistic network for iterative prediction.

Reparametrization The *reparametrization trick* has been applied in several fields under different names: perturbation analysis/pathwise derivatives (Glasserman & Ho, 1991) in stochastic optimization, stochastic backpropagation (Rezende et al., 2014), affine independent variational inference (Challis & Barber, 2012) or correlated sampling in evaluating differential privacy (Bichsel et al., 2018). The actual reparametrization of our model resembles that of the deep generative model of Rezende et al. (2014), but there it is used to perform variational inference.

7. Conclusion

In this work, we explored applying adversarial attacks to a recently proposed probabilistic autoregressive forecasting models. Our work is motivated by the fact that: *(i)* this model has been included in the Amazon SageMaker toolkit and achieved state-of-the-art results on a number of different tasks, and *(ii)* adversarial attacks and robustness are pressing and important issues that affect it.

Concretely, we implemented and evaluated two techniques, reparametrization and score-function estimators, that are used to differentiate trough Monte-Carlo estimation inherent to this model and instantiate existing gradient based adversarial attacks. While we show that both of these techniques can be used to generate adversarial attacks, we evidence that using the reparametrization estimator is crucial for producing adversarial attacks with a small perturbation norm. Further, we extend the prior work to the Bayesian setting which enables using these models with new types of queries.

References

Allen, F. and Gale, D. Stock-price manipulation. *The Review of Financial Studies*, 5(3):503–529, 1992.

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Ballet, V., Renard, X., Aigrain, J., Laugel, T., Frossard, P., and Detyniecki, M. Imperceptible adversarial attacks on tabular data. *arXiv preprint arXiv:1911.03274*, 2019.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 1171–1179. Curran Associates, Inc., 2015.

Bichsel, B., Gehr, T., Drachsler-Cohen, D., Tsankov, P., and Vechev, M. Dp-finder: Finding differential privacy violations by sampling and optimization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 508–524. ACM, 2018.

Bishop, C. M. Mixture density networks. 1994.

Black, F. and Scholes, M. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3): 637–654, 1973.

Borovykh, A., Bohte, S., and Oosterlee, C. W. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57. IEEE, 2017.

Challis, E. and Barber, D. Affine independent variational inference. In *Advances in Neural Information Processing Systems*, pp. 2186–2194, 2012.

Chen, Y., Kang, Y., Chen, Y., and Wang, Z. Probabilistic forecasting with temporal convolutional neural network. *arXiv preprint arXiv:1906.04397*, 2019.

Diaz, D., Theodoulidis, B., and Sampaio, P. Analysis of stock market manipulations using knowledge discovery techniques applied to intraday trade prices. *Expert Systems with Applications*, 38(10):12757–12771, 2011.

Duan, J.-C. The GARCH option pricing model. *Mathematical finance*, 5(1):13–32, 1995.

Feng, F., Chen, H., He, X., Ding, J., Sun, M., and Chua, T.-S. Enhancing stock movement prediction with adversarial training. *arXiv preprint arXiv:1810.09936*, 2018.

Figurnov, M., Mohamed, S., and Mnih, A. Implicit reparameterization gradients. In *Advances in Neural Information Processing Systems*, pp. 441–452, 2018.

Fischer, T. and Krauss, C. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

Glasserman, P. and Ho, Y.-C. *Gradient estimation via perturbation analysis*, volume 116. Springer Science & Business Media, 1991.

Glynn, P. W. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33 (10):75–84, 1990.

Gneiting, T. and Raftery, A. E. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Graves, A. Stochastic backpropagation through mixture density distributions. *arXiv preprint arXiv:1607.05690*, 2016.

Handa, P. and Schwartz, R. A. Limit order trading. *The Journal of Finance*, 51(5):1835–1861, 1996.

Hesterberg, T. C. Estimates and confidence intervals for importance sampling sensitivity analysis. *Mathematical and computer modelling*, 23(8-9):79–85, 1996.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jankowiak, M. and Obermeyer, F. Pathwise derivatives beyond the reparameterization trick. *arXiv preprint arXiv:1806.01851*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kleijnen, J. P. and Rubinstein, R. Y. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, 1996.

Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *ICLR'17 Workshop*, 2017a.

Kurakin, A., Goodfellow, I. J., and Bengio, S. Adversarial machine learning at scale. ICLR17, 2017b.

Öğüt, H., Doğanay, M. M., and Aktaş, R. Detecting stock-price manipulation in an emerging market: The case of turkey. *Expert Systems with Applications*, 36(9):11944–11949, 2009.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

Oord, A. v. d., Kalchbrenner, N., Espeholt, L., kavukcuoglu, k., Vinyals, O., and Graves, A. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems 29*, pp. 4790–4798. 2016b.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML16, pp. 17471756, 2016c.

Ormoneit, D. and Neuneier, R. Experiments in predicting the German stock index DAX with density estimating neural networks. In *IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering (CIFEr)*, pp. 66–71. IEEE, 1996.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 372–387. IEEE, 2016.

Ranganath, R., Gerrish, S., and Blei, D. Black box variational inference. In *Artificial Intelligence and Statistics*, pp. 814–822, 2014.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Rich, D. R. The mathematical foundations of barrier option-pricing theory. *Advances in futures and options research*, 7, 1994.

Roa-Vicens, J., Wang, Y., Mison, V., Gal, Y., and Silva, R. Adversarial recovery of agent rewards from latent spaces of the limit order book. *arXiv preprint arXiv:1912.04242*, 2019.

Salimans, T., Knowles, D. A., et al. Fixed-form variational posterior approximation through stochastic linear regression. *Bayesian Analysis*, 8(4):837–882, 2013.

Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 2019.

Salman, H., Li, J., Razenshteyn, I., Zhang, P., Zhang, H., Bubeck, S., and Yang, G. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, pp. 11289–11300, 2019.

Schittenkopf, C., Dorffner, G., and Dockner, E. J. Forecasting time-dependent conditional densities: a semi non-parametric neural network approach. *Journal of Forecasting*, 19(4):355–374, 2000.

Schreyer, M., Sattarov, T., Reimer, B., and Borth, D. Adversarial learning of deepfakes in accounting. *arXiv preprint arXiv:1910.03810*, 2019.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Weigel, A. P., Liniger, M. A., and Appenzeller, C. The discrete Brier and ranked probability skill scores. *Monthly Weather Review*, 135(1):118–124, 2007.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Wiyatno, R. R., Xu, A., Dia, O., and de Berker, A. Adversarial examples in modern machine learning: A review, 2019.

We provide the following three appendices:

- Appendix A provides proofs of Score-function Estimator 3.1 and Reparametrization Estimator 3.2 defined in section 3.
- Appendix B provides details of our datasets, preprocessings steps, architectures and hyper-parameters used in our experiments.
- Appendix C provides extended version of the experiments presented in sections 4 and 5.

A. Proofs

Score-function Estimator. *In the general Bayesian setting where $\boldsymbol{y} \sim q[\cdot|\boldsymbol{x}+\boldsymbol{\delta}, z]$, the score-function gradient estimator of the expected value of $\chi(\boldsymbol{y})$ is:*

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] \simeq \frac{\sum_{l=1}^{L} \chi(\boldsymbol{y}^l) q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l] \nabla_{\boldsymbol{\delta}} \log(q[\boldsymbol{y}^l|\boldsymbol{x}+\boldsymbol{\delta}, z])}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]}$$

where $\boldsymbol{y}^l$ is sampled from the prior distribution $q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}]$, and $q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}]$ denotes the probability that z is true knowing that $\boldsymbol{y}^l$ is generated.

Proof. The expectation is defined as the following integral over the output space:

$$\mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] = \int_{\boldsymbol{y}} \chi(\boldsymbol{y}) q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z] dy$$

Using Leibniz rule, we obtain

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] = \int_{\boldsymbol{y}} \chi(\boldsymbol{y}) \nabla_{\boldsymbol{\delta}} q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z] dy$$

at every point $\boldsymbol{\delta}$ around which the gradient $\nabla_{\boldsymbol{\delta}} q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]$ is locally continuous (in the model described in this paper, this regularity condition holds everywhere). The resulting integral can be transformed as follows into an expectation over the distribution $q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]$.

$$\begin{aligned}
&\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] \\
&= \int_{\boldsymbol{y}} \chi(\boldsymbol{y}) \cdot \nabla_{\boldsymbol{\delta}} q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z] dy \\
&= \int_{\boldsymbol{y}} \chi(\boldsymbol{y}) q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z] \frac{\nabla_{\boldsymbol{\delta}} q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]}{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]} dy \\
&= \int_{\boldsymbol{y}} \chi(\boldsymbol{y}) q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z] \nabla_{\boldsymbol{\delta}} \log\left(q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]\right) dy \\
&= \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y}) \nabla_{\boldsymbol{\delta}} \log\left(q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]\right)]
\end{aligned}$$

This expectation can be approximated via Monte-Carlo methods. While it is in general not possible to directly sample from $q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}, z]$, what can be done instead is generating samples $\boldsymbol{y}^l$ for $l \in 1 \leq l \leq L$ from the prior $q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}]$, and attribute an importance weight to each of the resulting samples, yielding:

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] \simeq \frac{\sum_{l=1}^{L} \chi(\boldsymbol{y}^l) q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l] \nabla_{\boldsymbol{\delta}} \log(q[\boldsymbol{y}^l|\boldsymbol{x}+\boldsymbol{\delta}, z])}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]}$$

The choice of $q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]$ as the importance weight for $\boldsymbol{y}^l$ results from the application of Bayes rule:

$$q[\boldsymbol{y}^l|\boldsymbol{x}+\boldsymbol{\delta}, z] = \frac{q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]}{q[z|\boldsymbol{x}+\boldsymbol{\delta}]} q[\boldsymbol{y}^l|\boldsymbol{x}+\boldsymbol{\delta}]$$

□

Reparametrization Estimator. *Assume there exists a differentiable transformation $g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta})$ such that the random variable $\boldsymbol{y} \sim q[\cdot|\boldsymbol{x}+\boldsymbol{\delta}]$ can be reparametrized as $\boldsymbol{y} = g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta})$, where $\boldsymbol{\eta}$ is an independent random variable whose marginal distribution $p(\boldsymbol{\eta})$ is independent from $\boldsymbol{\delta}$. Then the importance sampling reparametrization estimator of the expectation's gradient is:*

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] \simeq \nabla_{\boldsymbol{\delta}} \left(\frac{\sum_{l=1}^{L} \chi(g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)) q[z|\boldsymbol{x}+\boldsymbol{\delta}, g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)]}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)]} \right)$$

where for $1 \leq l \leq L$, $\boldsymbol{\eta}^l$ is sampled from the distribution $p(\boldsymbol{\eta})$, and $\boldsymbol{y}^l = g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)$.

Proof. Approximating the expectation $\mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})]$ via Monte-Carlo estimation with importance sampling yields:

$$\nabla_{\boldsymbol{\delta}} \mathbb{E}_{q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta},z]}[\chi(\boldsymbol{y})] \simeq \nabla_{\boldsymbol{\delta}} \left(\frac{\sum_{l=1}^{L} \chi(\boldsymbol{y}^l) q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]} \right)$$

where $\boldsymbol{y}^l$ is sampled from the prior distribution $q[\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\delta}]$. With the assumptions of the theorem, we can rewrite:

$$\left(\frac{\sum_{l=1}^{L} \chi(\boldsymbol{y}^l) q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, \boldsymbol{y}^l]} \right) = \left(\frac{\sum_{l=1}^{L} \chi(g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)) q[z|\boldsymbol{x}+\boldsymbol{\delta}, g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)]}{\sum_{l=1}^{L} q[z|\boldsymbol{x}+\boldsymbol{\delta}, g_{\boldsymbol{x}}(\boldsymbol{\delta}, \boldsymbol{\eta}^l)]} \right)$$

Since the respective effects of the perturbation and of randomness are decoupled in this final expression, it is differentiable with respect to $\boldsymbol{\delta}$, which concludes. □

B. Experimental Details

Here we provide details of all our experiments to support reproducibility. Additionally, we will make all our datasets and source code available online.

B.1. Datasets and Preprocessing

S&P500 The S&P500 dataset is obtained via the *yfinance* API[6]. We focus on data-points between 1990/01 and 2000/12, identified by Fischer and Krauss (2018) as a period of exceptionally high trading returns compared to the following decades. We also follow Fischer and Krauss for preprocessing the data. A sequence of prices $\boldsymbol{p} = (p_1, \dots, p_T)$ is first preprocessed to obtain a sequence of returns $(r_2, \dots, r_T)$, defined as $r_i = \frac{p_i}{p_{i-1}} - 1$. Intuitively r_i is the gain (when positive) or loss obtained by investing one dollar in the stock at time $i-1$, and then selling at time i. Inversely, given a sequence of returns $\boldsymbol{r}$ and an initial price p_1, the corresponding sequence of prices can be obtained as:

$$p_k = p_1 \prod_{i=2}^{k} (1 + r_i)$$

Both transformations are differentiable, which allows to perform the attack in the application space of prices rather than on returns. Besides, returns are normalized to have zero mean and unit variance. Denoting μ and σ for the mean and standard deviation of returns in the training set, the normalized sequence is $(\tilde{r}_2, \dots, \tilde{r}_T)$, where $\tilde{r}_i = (r_i - \mu)/\sigma$. We refer to Fischer and Krauss (2018) for a thorough analysis of the properties of the S&P500 dataset.

Electricity Dataset We use the same preprocessing steps as described in (Salinas et al., 2019). Input sequences are divided by their average value v, and the corresponding prediction sequence is multiplied by v. This guarantees that all inputs are approximately in the same range.

B.2. Neural Architectures: S&P500 Dataset

LSTM The LSTM baseline used on the S&P500 dataset, we follow (Fischer & Krauss, 2018), and use a single LSTM layer with 25 hidden units, followed by a linear output layer. However, we use only one input neuron without activation instead of two neurons with softmax activation.

TCN In (Borovykh et al., 2017), several sets of hyper-parameters for Temporal Convolutional Networks are used depending on the experiment. We decided to use 8 layers and a dilation of 2, in order to match as closely as possible the size of the LSTM receptive field. We selected the other parameters via grid-search, resulting in a kernel size of 2 and 3 channels. We use the TCN implementation provided by the authors of (Bai et al., 2018).

[6] https://github.com/ranaroussi/yfinance

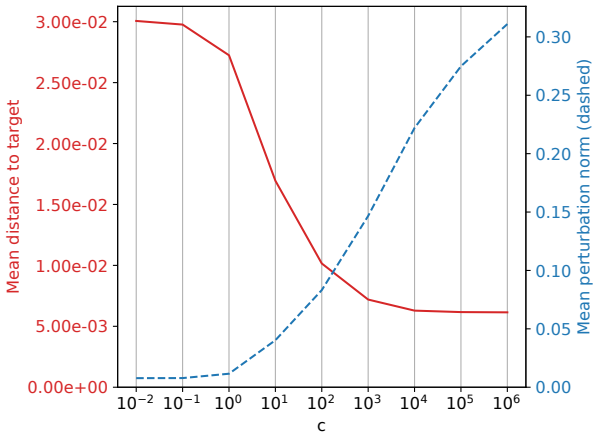

Figure 2. Perturbation norm and distance to target for different values of c when evaluated on the S&P500 Dataset.

Ours For our probabilistic autoregressive model, we chose to use a single LSTM layer with 25 hidden units similar to the LSTM baseline, in order to guarantee the most fair comparison. We only changed the output layers to parametrize a Gaussian distribution. Following (Bishop, 1994), we use a linear layer without activation for the mean, and a linear layer with exponential activation for the scale of the distribution. We also performed experiments with a Gaussian mixture likelihood, but it did improve the performance on our two benchmarks.

Training For both deterministic networks, we minimize mean-squared error on the training set. For our model, we use negative log-likelihood as a loss function. In both cases, we use the `RMSPROP` optimizer (Tieleman & Hinton, 2012) advised by Fischer and Krauss, with default parameters and learning rate of 0.01. We use an early-stopping patience of 20, and a large batch size of 2048 for training. Experiments with different values did not reveal a significant influence of these parameters.

B.3. Neural Architectures: Electricty Dataset

`DeepAR` The `DeepAR` architecture used for the Electricity experiments is based on a three-layer LSTM with 40 hidden units each. The number of samples used for Monte-Carlo estimation of the output is set to 200. The network is trained for 20 epochs with the `Adam` optimizer (Kingma & Ba, 2014), with batch size of 64 and learning rate of 0.001.

B.4. Attack Hyper-Parameters: S&P500 Dataset

For the S&P500 dataset, we optimize the attack objective function with the RMSPROP optimizer using a learning rate of 0.001 and 1000 iterations. These parameters were selected with a simple grid search be-

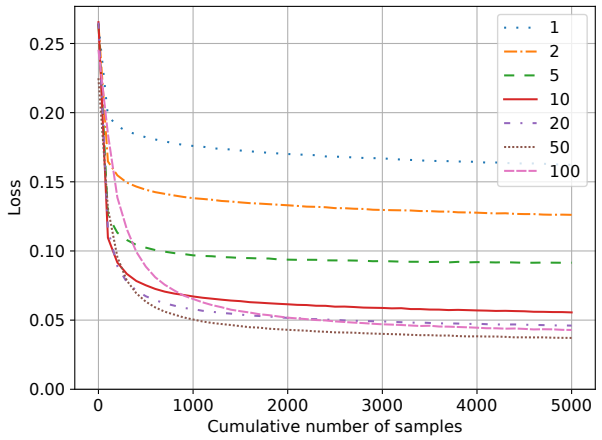

Figure 3. Attack loss for different values of L when evaluated on the S&P500 Dataset. The x-axis is the total number of generated samples rather than the number of perturbation updates, in order to provide a fair comparison in terms of attack computational cost.

cause of the computational cost of running the attack repeatedly. The values used for the coefficient c are $10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3, 10^4, 10^5, 10^6$. We select the value that yields the best adversarial sample under the constraint that the perturbation norm is below the tolerance ϵ. The number of samples used to estimate the gradient is chosen to be $L = 50$.

Buy/Sell Attack We use $\lambda = 0.03$ for the target. For the Bayesian setting, we use $\gamma = y_{10}/x_{-1} = 1.0008$, in order to approximately balance the different classes. The 95% confidence interval is computed assuming Student's t-distribution. In the Bayesian case, the formula for the 95% confidence interval with importance sampling is derived in (Hesterberg, 1996).

Attack on Trading Strategies We use $\alpha = 0.1$ for the target scaling factor.

Influence of c In Figure 2, we examine the influence of tuning the attack objective function on average perturbation norm and distance to the attack target. We observe a trade-off between these two quantities that depends on the coefficient c: higher value for c yields better adversarial samples, at the cost of more input perturbation.

Influence of L We evaluate the effect of the number of samples L used in the reparametrization estimator on the attack loss in Figure 3. In this experiment, the value of c is fixed to 1000. We notice a trade-off in terms of convergence speed vs. final loss, that depends on the number of samples used for estimating the gradient. As a result, we choose to use $L = 50$ in our attacks.

B.5. Attack Hyper-Parameters: Electricity Dataset

We optimize the attack objective function with the ADAM optimizer. We use different optimizers for the two datasets so that the same optimizer is used for training the network and to attack it. We use a learning rate of 0.01 and 1000 iterations. These parameters were also selected via informal search. The values used for the coefficient c are 0.1, 0.2, 0.3, 0.5, 0.7, 1, 2, 3, 5, 7, 10, 20, 30, 50, 70, 100, 200 and 300. The number of samples used to estimate the gradient is chosen to be $L = 50$.

C. Experimental Results

C.1. Trading Strategies

In Figure 4, we provide extended results for the long-short trading benchmark (Section 4.1), with different horizons h and number of samples used for Monte-Carlo estimation of the prediction. We observe that the quality of the probabilistic prediction improves with the number of samples until 10^4 samples. Further increasing the number of samples does not yield significant performance improvements.

C.2. Evaluation of the Probabilistic Forecast

In Table 3, we give detailed results for the comparison of probabilistic forecasts quality with Ranked Probability Skill (a summary of these results is provided in Table 2, Section 4.1). We observe that the forecasting quality of our model improves with the number of samples, and that an order of magnitude of the number of samples needed to obtain the best possible estimation is 10^4. As a comparison, the `DeepAR` implementation on the electricity dataset uses 200 samples. We surmise that this discrepancy is due to the low signal-to-noise ratio of financial data, that makes inference more difficult.

C.3. Bayesian Attack

In Figure 5, we plot the results of the classification attack in the Bayesian setting with observation $y_{10}/x_{-1} = \gamma$, where $\gamma = 1.0008$ (Section 5.1). We only implemented the reparametrization estimator, as the score-function estimator requires the overly complex estimation of $\nabla_{\boldsymbol{\delta}} \log(q[\boldsymbol{y}^l|\boldsymbol{x} + \boldsymbol{\delta}, z])$ for each sample $\boldsymbol{y}^l$. We observe that the attack success rate is very similar to the non-Bayesian setting, demonstrating that the reparametrization estimator adapts readily to the Bayesian setting. The attack success rate is approximately 80% for $\epsilon = 0.1$.

C.4. Electricity Dataset

In Figure 6, we show results of both over-estimation and under-estimation attacks on the electricity dataset, with examples of generated adversarial samples (Section 5.3).

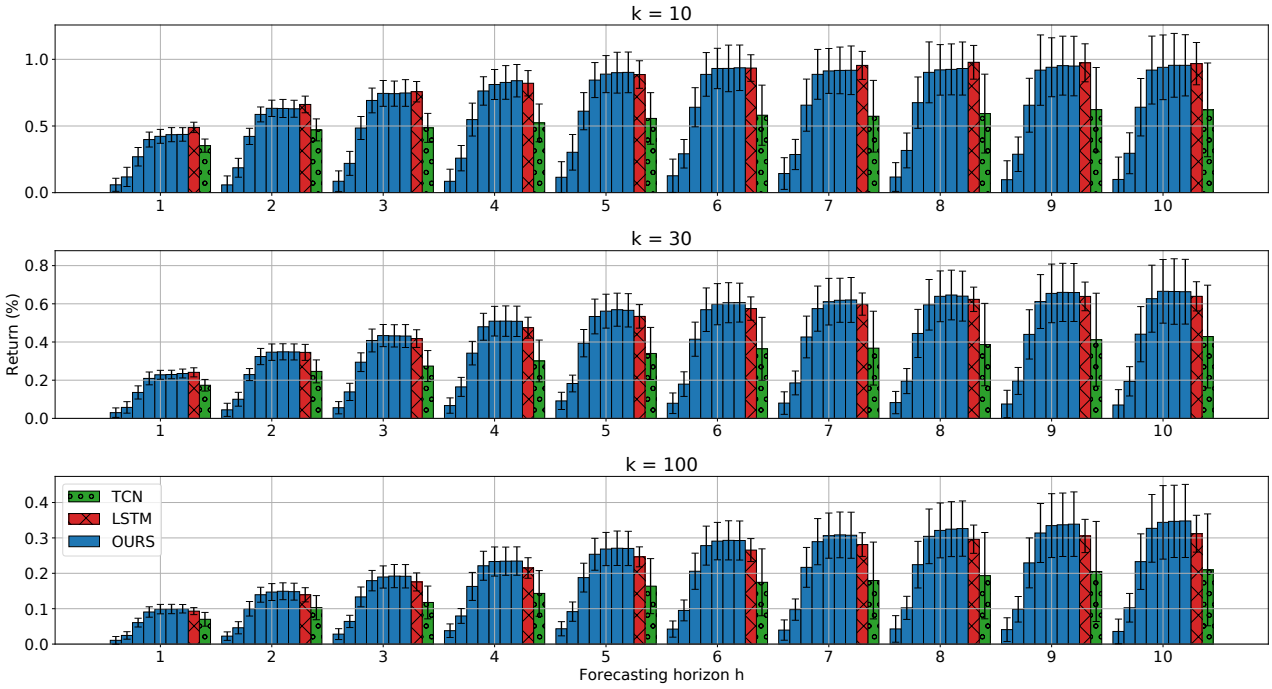

Figure 4. Financial gain on algorithmic trading tasks for different horizons h and portfolio sizes k (in $\%$ of the invested capital). The blue bars correspond to different number of samples for Monte-Carlo estimation of the prediction: from left to right $1, 10, 10^2, 10^3, 10^4, 10^5, 10^6$.

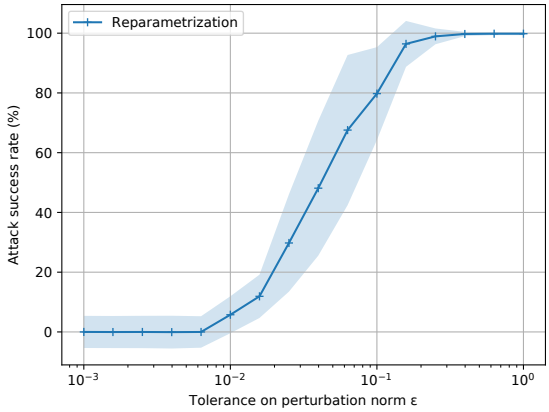

Figure 5. Success rate of the classification attack for different perturbation norms, in a Bayesian setting with observation $y_{10}/x_{-1} = \gamma = 1.0008$, and prediction horizon $h = 5$.

We observe that for equal perturbation tolerance, the over-estimation attack yields mis-predictions of smaller amplitude. For instance, the reparametrization attack with $\epsilon = 0.8$ causes median over-estimation of around 15%, whereas it causes median under-estimation of 20%. We believe that this is due to the particular nature of the dataset rather than asymmetry in the attack. We do not observe such a discrepancy in the financial experiments.

Table 3. Performance of different models on probabilistic forecast of various statistics. The comparison metric is Ranked Probability Skill (Weigel et al., 2007) of the prediction (lower scores correspond to better predictions). The performance of our architecture is given for different number of samples used in the Monte-Carlo estimation (1, 100, and 10000).

Statistics			Non-probabilistic		Probabilistic		
Name	h	π	`TCN` (Borovykh et al., 2017)	`LSTM` (Fischer & Krauss, 2018)	Ours 1 sample	Ours 100 samples	Ours 10000 samples
Cumulated Return	1	-	1.423 ($\pm$ 0.022)	1.424 ($\pm$ 0.016)	2.016 ($\pm$ 0.023)	0.992 ($\pm$ 0.002)	**0.982** ($\pm$ **0.002**)
	5	-	1.468 ($\pm$ 0.01)	1.466 ($\pm$ 0.008)	1.992 ($\pm$ 0.013)	1.0 ($\pm$ 0.005)	**0.99** ($\pm$ **0.004**)
	10	-	1.548 ($\pm$ 0.029)	1.541 ($\pm$ 0.019)	1.995 ($\pm$ 0.011)	1.012 ($\pm$ 0.008)	**1.002** ($\pm$ **0.008**)
European Call Option	10	0.9	1.019 ($\pm$ 0.004)	1.017 ($\pm$ 0.003)	1.961 ($\pm$ 0.22)	0.999 ($\pm$ 0.009)	**0.989** ($\pm$ **0.009**)
	10	1	1.122 ($\pm$ 0.002)	1.121 ($\pm$ 0.002)	1.966 ($\pm$ 0.103)	0.992 ($\pm$ 0.006)	**0.982** ($\pm$ **0.005**)
	10	1.1	1.342 ($\pm$ 0.002)	1.341 ($\pm$ 0.002)	1.987 ($\pm$ 0.03)	1.003 ($\pm$ 0.007)	**0.993** ($\pm$ **0.007**)
European Put Option	10	0.9	1.445 ($\pm$ 0.021)	1.445 ($\pm$ 0.017)	1.984 ($\pm$ 0.015)	1.002 ($\pm$ 0.007)	**0.992** ($\pm$ **0.007**)
	10	1	1.302 ($\pm$ 0.003)	1.3 ($\pm$ 0.002)	1.957 ($\pm$ 0.036)	0.984 ($\pm$ 0.005)	**0.974** ($\pm$ **0.005**)
	10	1.1	1.046 ($\pm$ 0.005)	1.044 ($\pm$ 0.002)	1.856 ($\pm$ 0.094)	0.968 ($\pm$ 0.004)	**0.959** ($\pm$ **0.003**)
Limit Sell	10	1.01	2.822 ($\pm$ 0.501)	3.137 ($\pm$ 0.307)	1.917 ($\pm$ 0.021)	1.013 ($\pm$ 0.008)	**1.004** ($\pm$ **0.008**)
	10	1.05	1.516 ($\pm$ 0.001)	1.514 ($\pm$ 0.002)	1.899 ($\pm$ 0.027)	0.953 ($\pm$ 0.006)	**0.944** ($\pm$ **0.006**)
	10	1.20	1.035 ($\pm$ 0.003)	1.034 ($\pm$ 0.002)	1.792 ($\pm$ 0.12)	0.951 ($\pm$ 0.006)	**0.942** ($\pm$ **0.005**)
Limit Buy	10	0.8	1.02 ($\pm$ 0.002)	1.019 ($\pm$ 0.002)	1.948 ($\pm$ 0.223)	0.982 ($\pm$ 0.015)	**0.972** ($\pm$ **0.013**)
	10	0.95	1.412 ($\pm$ 0.0)	1.41 ($\pm$ 0.001)	1.926 ($\pm$ 0.039)	0.967 ($\pm$ 0.009)	**0.958** ($\pm$ **0.008**)
	10	0.99	3.047 ($\pm$ 0.012)	3.025 ($\pm$ 0.028)	1.963 ($\pm$ 0.024)	1.013 ($\pm$ 0.006)	**1.003** ($\pm$ **0.006**)

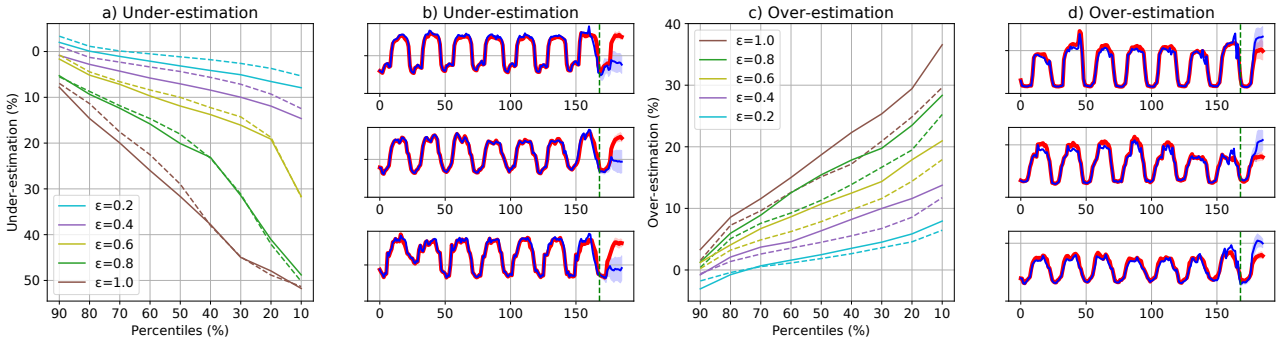

Figure 6. Results for the electricity dataset. a) Under-estimation of electricity consumption. For example, with $\epsilon = 1.0$, the attack using reparametrization estimator leads to under-estimation of at least 20% (y-axis) for 70% of samples (x-axis). b) Under-estimation adversarial samples. c) Over-estimation of electricity consumption. d) Over-estimation adversarial samples. In a) and b), results are given for reparametrization (continuous) and score-function (dashed) estimators. In c) and d), the reparametrization estimator is used, and ϵ is fixed to 0.9. Red curve is the original sample, blue curve is the generated adversarial sample. The vertical dashed line separates the input sequence from the network's prediction.