Wasserstein Generative Adversarial Networks for Online Test Generation for Cyber Physical Systems

Jarkko Peltomäki

Information Technology
Åbo Akademi University

9.5.2022

Joint work with F. Spencer and I. Porres

Context

- Aim: online test suite generation using generative adversarial networks (GANs).

Context

- Aim: online test suite generation using generative adversarial networks (GANs).
- We assume that we have a system under test (SUT) $\mathcal{S}$ which is a deterministic (black-box) mapping $\mathcal{I} \to \mathbb{R}^m$ where $\mathcal{I} \subseteq \mathbb{R}^n$.

Context

- Aim: online test suite generation using generative adversarial networks (GANs).
- We assume that we have a system under test (SUT) $\mathcal{S}$ which is a deterministic (black-box) mapping $\mathcal{I} \to \mathbb{R}^m$ where $\mathcal{I} \subseteq \mathbb{R}^n$.
- We assume that our requirements for the SUT are given as a fitness function $f \colon \mathcal{I} \to [0, 1]$ such that an test $t \in \mathcal{I}$ falsifies the requirements if and only if $f(t) = 1$ (high-fitness $\leftrightarrow$ challenging test).

Context

- Aim: online test suite generation using generative adversarial networks (GANs).
- We assume that we have a system under test (SUT) $\mathcal{S}$ which is a deterministic (black-box) mapping $\mathcal{I} \to \mathbb{R}^m$ where $\mathcal{I} \subseteq \mathbb{R}^n$.
- We assume that our requirements for the SUT are given as a fitness function $f : \mathcal{I} \to [0, 1]$ such that an test $t \in \mathcal{I}$ falsifies the requirements if and only if $f(t) = 1$ (high-fitness $\leftrightarrow$ challenging test).
- We assume that executions are expensive, so we want to avoid calls $\mathcal{S}(t)$.

Generative Adversarial Networks

- Let $\mathcal{P}$ be a probability distribution on the input space $\mathcal{I}$.

Generative Adversarial Networks

- Let $\mathcal{P}$ be a probability distribution on the input space $\mathcal{I}$.
- A *generator* G is a mapping $G\colon \mathcal{I}' \subseteq \mathbb{R}^d \to \mathcal{I}$ such that G transforms the, say, uniform distribution on $\mathcal{I}'$ to $\mathcal{P}$ on $\mathcal{I}$.

Generative Adversarial Networks

- Let $\mathcal{P}$ be a probability distribution on the input space $\mathcal{I}$.
- A *generator* G is a mapping $G\colon \mathcal{I}' \subseteq \mathbb{R}^d \to \mathcal{I}$ such that G transforms the, say, uniform distribution on $\mathcal{I}'$ to $\mathcal{P}$ on $\mathcal{I}$. That is, sampling uniformly on $\mathcal{I}'$ produces samples on $\mathcal{I}$ according to $\mathcal{P}$ via the map G.

Generative Adversarial Networks

- Let $\mathcal{P}$ be a probability distribution on the input space $\mathcal{I}$.
- A *generator* G is a mapping $G\colon \mathcal{I}' \subseteq \mathbb{R}^d \to \mathcal{I}$ such that G transforms the, say, uniform distribution on $\mathcal{I}'$ to $\mathcal{P}$ on $\mathcal{I}$. That is, sampling uniformly on $\mathcal{I}'$ produces samples on $\mathcal{I}$ according to $\mathcal{P}$ via the map G.
- A Wasserstein generative adversarial network (WGAN) is a way to find such a G based on a large data sample from $\mathcal{P}$.

WGANs and Testing

- For validation, it would be desirable to have a WGAN trained on the uniform distribution on

$$\{t \in \mathcal{I} : f(t) > 1 - \varepsilon\}$$

  for a small ε (this is the set of challenging tests).
- Sampling from such a WGAN yields a good test suite.

Problem

- Problem: We do not assume to have a large data sample for training a WGAN, so how do we train a WGAN?
- Solution: online training of a WGAN (our proposal).

Broad Ideas

- Obtain a random sample T of tests.

Broad Ideas

- Obtain a random sample T of tests.
- Train a WGAN G on tests of T with “high-fitness”.

Broad Ideas

- Obtain a random sample T of tests.
- Train a WGAN G on tests of T with "high-fitness".
- Use G to sample a "good" new test t.

Broad Ideas

- Obtain a random sample T of tests.
- Train a WGAN G on tests of T with “high-fitness”.
- Use G to sample a “good” new test t.
- Add t to T and retrain.

Broad Ideas

- Obtain a random sample T of tests.
- Train a WGAN G on tests of T with "high-fitness".
- Use G to sample a "good" new test t.
- Add t to T and retrain.
- Repeat until $|T|$ is large enough (budget exhausted).

Broad Ideas

- Obtain a random sample T of tests.
- Train a WGAN G on tests of T with “high-fitness”.
- Use G to sample a “good” new test t.
- Add t to T and retrain.
- Repeat until $|T|$ is large enough (budget exhausted).

Two issues:

- What does “high-fitness” mean?

Broad Ideas

- Obtain a random sample T of tests.
- Train a WGAN G on tests of T with "high-fitness".
- Use G to sample a "good" new test t.
- Add t to T and retrain.
- Repeat until $|T|$ is large enough (budget exhausted).

Two issues:

- What does "high-fitness" mean?
- How to determine that a candidate test is "good", that is, how to ensure that adding it to T drives G to learn how to sample high-fitness tests?

Solution to Second Issue

- Solution: train an analyzer A (a neural network) for the mapping $t \mapsto f(t)$ using T.

Solution to Second Issue

- Solution: train an analyzer A (a neural network) for the mapping $t \mapsto f(t)$ using T.
- The analyzer can estimate the fitness of a test without executing it on the SUT.

More Complete Algorithm

- Sample N random tests T.
- Repeat while $|T| < \text{budget}$:
  - Train generator G on high-fitness samples of T.

More Complete Algorithm

- Sample N random tests T.
- Repeat while $|T| < \text{budget}$:
  - Train generator G on high-fitness samples of T.
  - Train analyzer A on T.

More Complete Algorithm

- Sample N random tests T.
- Repeat while $|T| <$ budget:
  - Train generator G on high-fitness samples of T.
  - Train analyzer A on T.
  - Sample G for tests and estimate their fitness using A.

More Complete Algorithm

- Sample N random tests T.
- Repeat while $|T| < \text{budget}$:
  - Train generator G on high-fitness samples of T.
  - Train analyzer A on T.
  - Sample G for tests and estimate their fitness using A.
  - Select the test t with best estimated fitness.

More Complete Algorithm

- Sample N random tests T.
- Repeat while $|T| <$ budget:
  - Train generator G on high-fitness samples of T.
  - Train analyzer A on T.
  - Sample G for tests and estimate their fitness using A.
  - Select the test t with best estimated fitness.
  - Execute t on the SUT to learn its true fitness.

More Complete Algorithm

- Sample N random tests T.
- Repeat while $|T| <$ budget:
  - Train generator G on high-fitness samples of T.
  - Train analyzer A on T.
  - Sample G for tests and estimate their fitness using A.
  - Select the test t with best estimated fitness.
  - Execute t on the SUT to learn its true fitness.
  - Add t to T.

More Complete Algorithm

- Sample N random tests T.
- Repeat while $|T| <$ budget:
  - Train generator G on high-fitness samples of T.
  - Train analyzer A on T.
  - Sample G for tests and estimate their fitness using A.
  - Select the test t with best estimated fitness.
  - Execute t on the SUT to learn its true fitness.
  - Add t to T.
- N.B. We execute the best test t on the SUT in order to find more training data for A. Without this the estimates of A can be unreliable.

The Other Issue

- We find a training batch for G by sampling T in a biased way, i.e., high-fitness tests have higher chance of being included in the batch (repetitions possible).

The Other Issue

- We find a training batch for G by sampling T in a biased way, i.e., high-fitness tests have higher chance of being included in the batch (repetitions possible).
- Details in the paper.

Intuition

- The intuition is that over time A gets more accurate and thus more high-fitness tests get included in the training batch of G. Thus G should be able to learn a distribution on high-fitness tests.

Intuition

- The intuition is that over time A gets more accurate and thus more high-fitness tests get included in the training batch of G. Thus G should be able to learn a distribution on high-fitness tests.
- If the validation task is not too difficult, it is expected that the test suite generated will contain falsifying tests.

Experimental Validation

- We have conducted an experiment comparing our approach with a Random search and a genetic algorithm in the context of the SBST 2021 CPS Tool Competition. See the paper for details.
- The results indicate that we can achieve state of the art performance.

Thank You

Thank you for your attention!