Learning To Simulate

Nataniel Ruiz\textsuperscript{1}, Samuel Schulter\textsuperscript{2}, Manmohan Chandraker\textsuperscript{2}

\textsuperscript{1}Boston University \textsuperscript{2}NEC-Labs

\begin{itemize}
\item Sampling data randomly from a simulator can be very beneficial when data is scarce or annotation is costly \cite{1,2}.
\item Previous work simulates large quantities of random scenes for tasks such as semantic segmentation or object detection in traffic scenes \cite{3,4}.
\item Our objective is to learn to simulate better data, which, when trained on yields a model with improved performance.
\item We propose a reinforcement learning-based method for automatically adjusting the parameters of any (non-differentiable) simulator.
\end{itemize}

\begin{itemize}
\item We want to solve the following bi-level optimization problem.
\begin{equation}
\psi^* = \arg \min_{\psi} \sum_{(x,y) \in D_{val}} L(y, h_\theta(x; \theta^*(\psi)))
\end{equation}
\begin{equation}
\text{s.t. } \theta^*(\psi) = \arg \min_{\theta} \sum_{(x,y) \in D_{\psi}(x,y)} L(y, h_\theta(x; \theta)),
\end{equation}
\end{itemize}

\begin{itemize}
\item We resort to reinforcement learning to solve this problem since the simulator is non-differentiable in the general case, among other reasons.
\item We use the vanilla policy gradient method to optimize $\psi$.
\end{itemize}

Our Simulator

Synthetic images generated by our parameterized simulator. We simulate a straight portion of road with houses and five different types of cars with variable weather and length of road.

Our simulator is a heavily modified version of the CARLA \cite{1} plugin in the Unreal Engine 4 development suite.

Motivation

- Sampling data randomly from a simulator can be very beneficial when data is scarce or annotation is costly \cite{1,2}.
- Previous work simulates large quantities of random scenes for tasks such as semantic segmentation or object detection in traffic scenes \cite{3,4}.

Approach

- Our objective is to learn to simulate better data, which, when trained on yields a model with improved performance.
- We propose a reinforcement learning-based method for automatically adjusting the parameters of any (non-differentiable) simulator.

Method

- We want to solve the following bi-level optimization problem.
  \begin{equation}
  \psi^* = \arg \min_{\psi} \sum_{(x,y) \in D_{val}} L(y, h_\theta(x; \theta^*(\psi)))
  \end{equation}
  \begin{equation}
  \text{s.t. } \theta^*(\psi) = \arg \min_{\theta} \sum_{(x,y) \in D_{\psi}(x,y)} L(y, h_\theta(x; \theta)),
  \end{equation}

  $\psi$ are the simulator parameters, $h_\theta$ is the model parametrized by $\theta$, $L$ is the loss, $D_{val}$ is the validation set and $D_{\psi}(x,y)$ describes a dataset generated by the simulator distribution $\psi(x,y; \psi)$.

- We resort to reinforcement learning to solve this problem since the simulator is non-differentiable in the general case, among other reasons.
- We use the vanilla policy gradient method to optimize $\psi$.

\begin{itemize}
\item We work on two computer vision tasks using our traffic scenes simulator: the car counting task and semantic segmentation.
\item For the car counting task we train a convolutional neural network to count all instances individually for five different types of cars in an image.
\item We observe that we learn how to simulate datasets which achieve lower error than the mean error obtained using the validation set parameters, independent of the simulation parameter initialization.
\item For semantic segmentation, our method outperforms random policy parameters on real data (both on the KITTI validation set and on the KITTI test set). Moreover it outperforms the validation parameters on a simulated dataset.
\end{itemize}

References

\begin{itemize}
\end{itemize}

Figure 1: A high-level overview of our “learning to simulate” approach.

Table 1: Mean value of Car IoU on the KITTI test set for models $h_\theta$ trained from synthetic data generated by random or learned parameters.

Figure 2: Top row: The decision boundaries (shaded areas) of a non-linear SVM trained on data generated by $g(x, y; \psi)$ for three different iterations of our policy $\pi$. The data points overlaid are the test set. Bottom row: Decision boundary when trained on data sampled from $g(x, y; \psi_{val})$ (left) and on the converged parameters $\psi^*$ (middle). Data sampled from $g(x, y; \psi^*)$ (right).