Data Visualization Milestone #2
A high-dimensional inspection tool for deep neural network safety

Kyle Matoba and Arnaud Pannatier

May 7, 2023

1 The core visualization

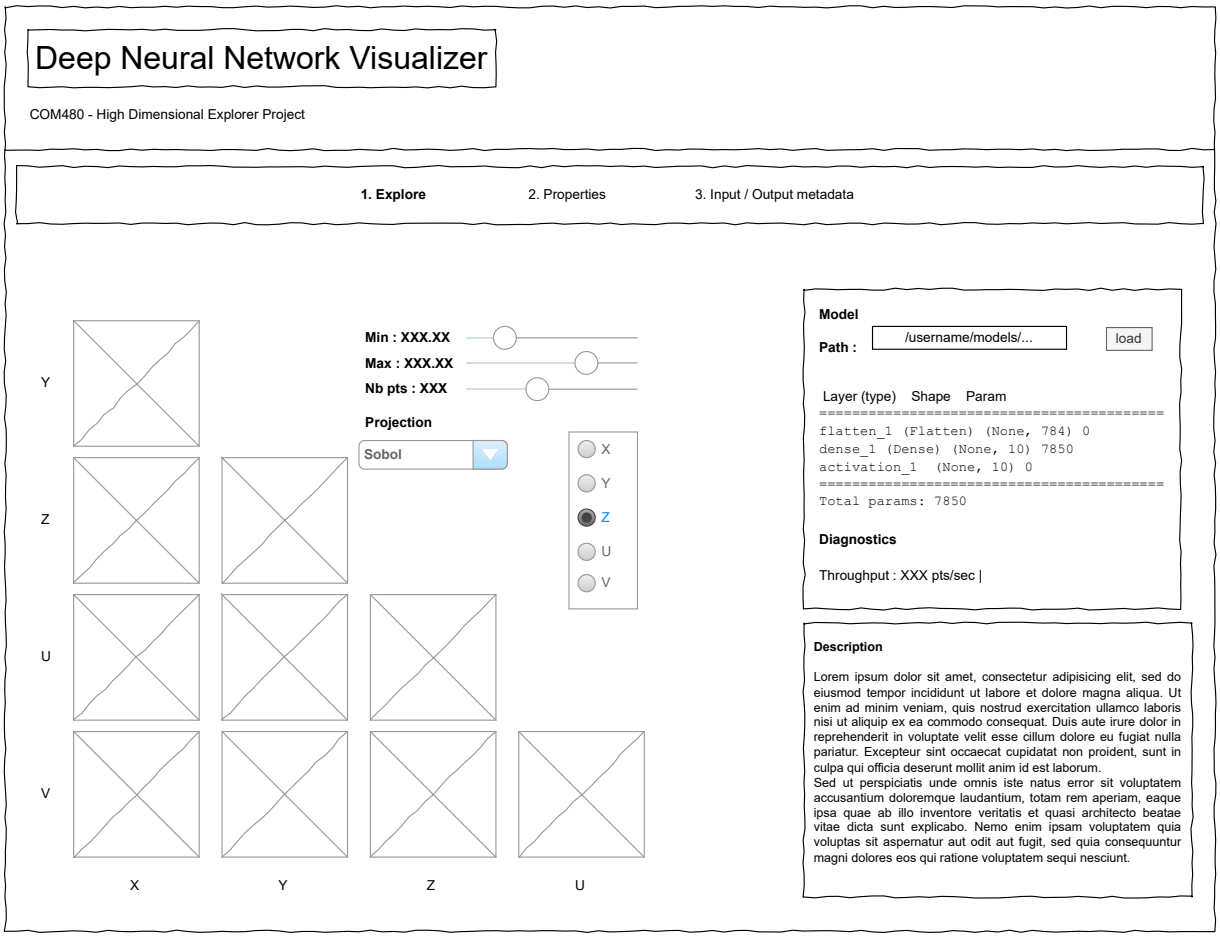

Figure 1: Wireframe design of the first tab of the visualization tool depicting all 2D projections of 5-dimensional spaces as a grid. Users can select a dimension and range of points, and a predefined scheme selects a random number of points that are passed through a deep neural network to generate real-time plots. The right panel provides the user with the option to select a model checkpoint and view relevant information.

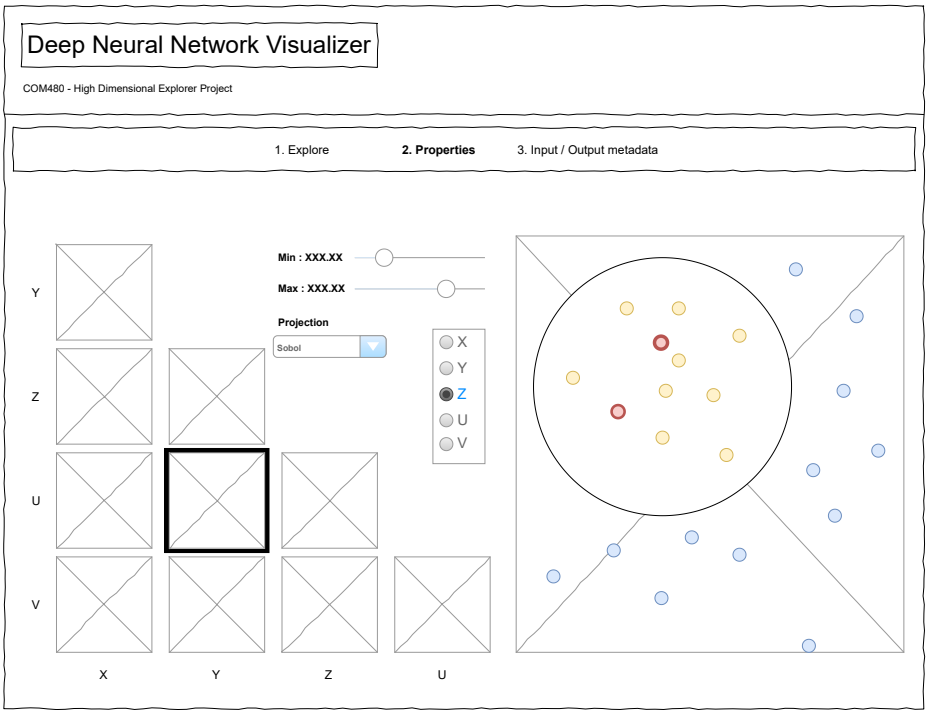

Figure 2: Wireframe design of the second tab of the visualization tool depicting all 2D projections of 5-dimensional spaces as a grid. Users can select a specific section of the diagram where all points should adhere to a given criterion. The tool will highlight any points that do not meet the criteria, enabling users to identify flaws in their models. This feature improves the model's reliability in identifying and correcting anomalies.

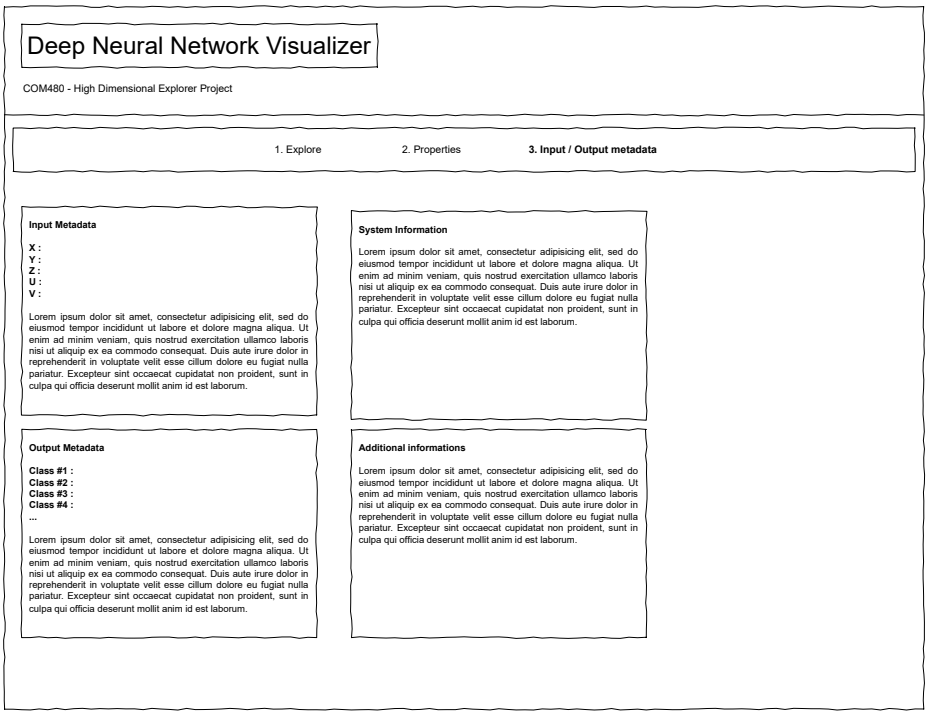

Figure 3: Wireframe design of the third tab of the visualization tool depicting the input and output metadata, as well as additional information about the system and task. Additionally, this tab displays system information, such as the hardware and software configuration, as well as details about the task being performed.

2 Tools and lectures

2.1 Tools

Python-based PyTorch ([3]) is the standard deep learning framework for both research and production. There is no credible JS-based alternative. We use SciKit learn ([4]) for their quasi-Monte Carlo pointsets. On the visualization side, we are drawn to open-source Bokeh ([1]) as a mature, performant, and featureful framework. To quote their site:[1] "Tools and widgets let you and your audience probe "what if" scenarios or drill down into the details of your data.", which is exactly our use case. Bokeh inherently supports deployment to web pages that can hide. To manage our visualization we anticipate using the recently-introduced configuration management system *Hydra* ([5]). This brilliant tool is a yaml-based way to hierarchically specify all aspects of runtime behavior that can be flexibly overridden and composed.

Lastly, our method applies to any deep neural network that predicts which of a discrete set of actions to take. If we have time, it would be interesting to demonstrate our tool on a wider variety of problems. OpenAI Gym ([2]), offers a standardized testbed for training and evaluating deep neural network-based control policies.

2.2 What lectures do I need?

Our data does not have any of the special structure (text, sound, graphs, cartographic, tabular, etc.) covered in some specialized lectures.

The choice of color, covered in lecture 6, is somewhat important, though less so than in applications where fine-grained parsing of continuous-valued data is important. We will ensure that our visualizations have "integrity" as described in Lecture 7.2. Happily, our application, concerned with physical phenomena has none of the "lying with statistics" motivations that pervade the presentation of many domains.

The most related lecture by far seems to be 7.1 on design. A notable aspect of our visualization is that our anticipated end user is evaluating the safety of a deep neural network in solving some control tasks. We can therefore assume that the user is quite knowledgeable and mathematically sophisticated. This means that we can dispense with some purely presentational niceties (e.g. "chartjunk"), but also that we have to be especially aware of ergonomics in the sense that users may use the tool continuously.

3 Atomic steps

3.1 Core visualization

Here are distinct steps that would need to be completed.

- Quasi-Monte Carlo point set creation: build and cache pointsets of needed size and dimensionality.
- Neural network loader: retrieve models from non-volatile storage and prepare for inference.
- Deep neural network inference: evaluate model predictions at all points in the domain.
- Metadata (limits, variable name) on input space and output actions.
- Subplot layout: generate a grid of plots.
- Plot navigator: linked panning, zooming, scrolling, etc. highlight points for further inspection.
- Analysis summary: display in a friendly way a summary of all aspects of the analysis being performed.

[1] `https://bokeh.org/`

3.2 Nice to have

- Screener: input conditions and automatically flag violations.
- Fancy projections: Rather than presenting all $\binom{d}{2}$ projections of a d-dimensional domain, look at ways to reduce dimensionality without substantially losing information.
- Dynamic configuration of plot substance: number of points, type of point set, etc.
- Dynamic configuration of plot presentation: color, markers, etc.
- Hardware acceleration: minimize I/O from deep learning hardware (e.g. graphics processing units) to CPU for visualization.

Works Cited

[1] Bokeh Development Team. *Bokeh: Python library for interactive visualization*. 2018. URL: `https://bokeh.pydata.org/en/latest/`.

[2] Greg Brockman et al. "OpenAI Gym". In: (2016). URL: `http://arxiv.org/abs/1606.01540`.

[3] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS 2017 Workshop Autodiff Program*. 2017.

[4] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: `https://doi.org/10.5555/1953048.2078195`.

[5] Omry Yadan. *Hydra - A framework for elegantly configuring complex applications*. Github. 2019. URL: `https://github.com/facebookresearch/hydra`.