Name:Karthik Hubli
UID:01689129

Literature Review-1(COMP-5460 Spring 2018)

The paper chosen for the review is 'Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow' [Wongsuphasawat, K, Smilkov. D, Wexler. J, Wilson. J, Mane. D, Fritz. D, Krishnan. D, Viegas. F.B, Wattenberg. M] published in IEEE Transactions on Visualization and Computer Graphics IEEE Trans. Visual. Comput. Graphics Visualization and Computer Graphics, IEEE Transactions on. 24(1):1-12 Jan, 2018. The primary reference of this paper is 'TensorFlow: Large-scale machine learning on heterogeneous systems' [Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng] published in 2015.

Recent years have seen a series of breakthroughs in machine learning, with deep learning bringing dramatic results on standard benchmarks. The authors present the TensorFlow Graph Visualizer, a component of in the TensorFlow machine intelligence platform, to help developers understand and inspect the structure of their TensorFlow models. Given a low-level directed dataflow graph of a model as input, the visualizer produces an interactive visualization that shows the high-level structure of the model, similarly to diagrams that deep learning experts typically draw to explain their models and enables users to explore its nested structure on demand. Since diagrams are critical for their work, machine learning experts desire a tool that can automatically visualize the model structures. Motivated by an apparent need for a visualization tool with potential users to understand the key tasks for such a visualization.

Design goal is to help developers understand and communicate the structures of TensorFlow models, which can be useful in many scenarios. As discussed earlier, researchers often manually create diagrams to explain their models and build mental maps. In model development, developers also need to understand the model beyond the high-level structure. For example, when a developer modifies a part of the code and sees an unexpected result, the reason may lie either in the model itself or in the code that created the model. From conversations with experts about potential usage scenarios and our observation from these hand-drawn diagrams, authors identify a set of main tasks that the visualization should support. Understand an overview of the high-level components of the model and their relationships, similar to schematic diagrams that developers manually draw to explain their model structure.

Design process also included an investigation into the particular properties of dataflow graphs that define TensorFlow models. Authors initially performed rapid prototyping to investigate the reasons that TensorFlow graphs caused problems for standard layouts. While most nodes in TensorFlow graphs have low-degree, most graphs also contain some interconnected high-degree nodes that couple different parts of the graphs. TensorFlow Graph Visualizer. The paper describes the design of TensorFlow Graph Visualizer that aims to help users with the tasks in. For simplicity, authors will use a simple regression model for image classification to illustrate how authors transform an illegible graph into a high-level diagram.

A summary node always has one input edge from a logged operation and one output edge to the summary writer node, which is the global sink node that takes all summary nodes as inputs and write log data to the log file. Authors place embedded nodes on the side of their neighbor nodes to make the overall layout more compact and avoid occlusion with other edges that connect with the node on the top and bottom. Edges are bundled between groups to help simplify the layout and make the clustered flow layout responsive and stable when users expand nodes. If a node's name conflicts with a names-pace the node inside the group node and add parentheses around its name. Bundling Edges to Enable Stable and Responsive Expansion After grouping nodes to build a hierarchy, authors draw group edges, or edges between groups. To do so, authors create group edges only between nodes within the same subgraphs of the group nodes. Within a subgraph, authors create group edges between group nodes and operation nodes as well as group edges between pairs of group nodes. When group nodes are expanded, edges are routed along the hierarchy instead of directly drawing edges between all visible nodes.

Responsiveness With edge routing, authors can compute the layout of each group node's subgraph separately because edges do not cross the boundary of each group node. Moreover, when a group node is expanded or collapsed, authors only need to re-compute the layout for the ancestors of the group node instead of recomputing the whole layout since other nodes are unaffected. Therefore, expanding a node only enlarges the node and its ancestors without causing major changes to the layout. Similarly, the issues by extracting non-critical nodes, but from each subgraph in the clustered graph instead of the raw input graph. When an extracted node or one of its proxies is selected or hovered, authors highlight both the node and all of its proxies to help users locate the node and its neighbors. Since extracting nodes affects the degrees of their neighbors, authors extract high in-degree nodes before high out-degree nodes so core nodes that connect to many auxiliary nodes no longer have high out-degree. To extract high in-degree nodes, calculate the quartiles of in-degrees for a subgraph, ignoring edges of extracted nodes. (To demonstrate our transformations with a simplified example, authors disable this threshold. After extracting high in-degree nodes, authors repeat the same procedure to extract out-degree nodes, but use a conservative threshold to avoid extracting core nodes. Extracting auxiliary nodes declutters clustered graphs and provides a layout that shows core structures of the models.

The paper restricts the search to subgraphs that are defined by single group nodes. The key consists of the number of nodes and edges, and a histogram counting types of operation nodes inside the group node's subgraph. For each cluster, authors first initialize a set of templates to an empty set. For each group node in the cluster, authors compare it with each existing template. For each template, authors compare with a node that belongs to using a subgraph similarity checking method described in the next paragraph. If does not match any existing templates, authors add a new template with as a member to the set . After visiting all nodes in the cluster, authors add all templates that have more than one member to the global template set , which is used for assigning colors. Authors define a node's signature as a tuple of the node's type, which can be an operation type for an operation node or its template unique identifier1 for a group node, the node's in-degree, and the node's out-degree. For the blocking step, authors insert each group node to the cluster hash map based on its key, which is already included in the hierarchy. For comparing group nodes in each cluster, checking similarity for two subgraphs with nodes and edges takes time. These views color nodes using a single-hue color ramp: nodes with higher compute time or memory usage have more saturation.

Since the model is based on a large code base, the user looks at the visualizer along with the code to understand the model structure. Following each of the convolution layers, the model includes a max-pooling operation to downsize the representation size and make the model invariant to low-

level transformations, as well as a local response normalization operation to reduce overfitting. Expanding the conv1 module to see its nested structure, the user observes that the module composes a Conv2D operation with weights and biases variables and forwards the output to a node named conv1. Glancing at the auxiliary nodes, the user sees several groups that connect to all convolution and fully connected layers. After understanding the model structure, the user trains the model. From the overview, the user recognizes that some nodes share the same colors and thus have identical nested structure. The lower part of the model contains the same convolution module, with identical max-pooling layers occasionally interleaving in the middle.

To summarize, deep learning models are becoming increasingly important in many applications. This process emphasizes understanding of both users and data. Authors describe a task analysis for developers of deep learning models and outline the layout challenges presented by model structures. As per the findings of the authors is that developers were willing to change their own code in the interest of improving the visualization, manually adding metadata to their graph in order to clarify the layout.