US 20190354870A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2019/0354870 A1**

Odena (43) **Pub. Date: Nov. 21, 2019**

(54) **SYSTEMS AND METHODS FOR DEBUGGING NEURAL NETWORKS WITH COVERAGE GUIDED FUZZING**

(71) Applicant: **Google LLC**, Mountain View, CA (US)

(72) Inventor: **Augustus Quadrozzi Odena**, San Francisco, CA (US)

(21) Appl. No.: **16/415,693**

(22) Filed: **May 17, 2019**

Related U.S. Application Data

(60) Provisional application No. 62/673,751, filed on May 18, 2018.

Publication Classification

(51) **Int. Cl.**

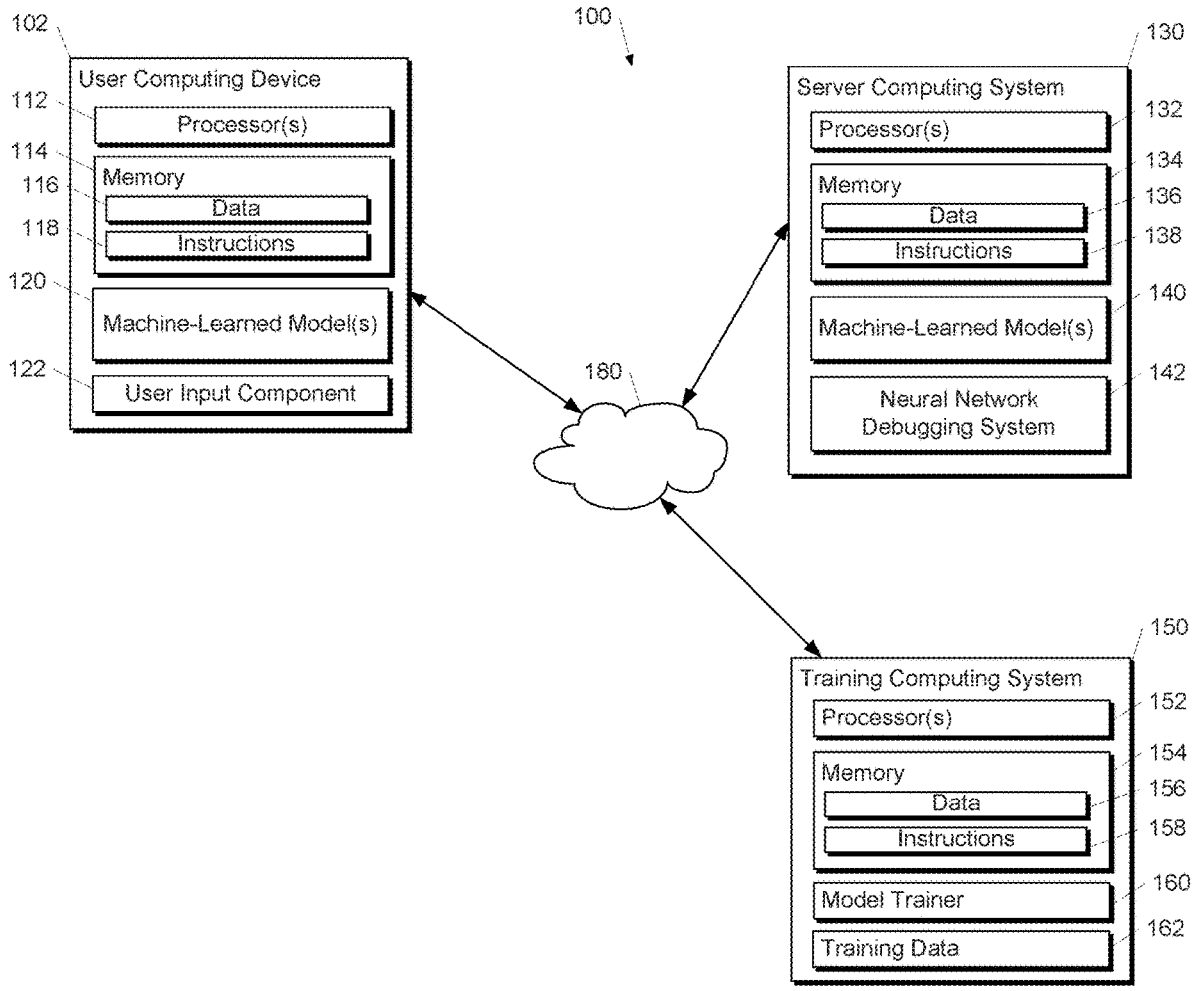
G06N 3/10	(2006.01)
G06N 3/08	(2006.01)
G06F 11/36	(2006.01)

(52) **U.S. Cl.**

CPC ***G06N 3/10*** (2013.01); ***G06F 11/366*** (2013.01); ***G06N 3/08*** (2013.01)

(57) **ABSTRACT**

The present disclosure provides systems and methods for debugging neural networks. In one example, a computer-implemented method is provided, which includes obtaining, by one or more computing devices, one or more inputs from an input corpus. The method further includes mutating, by the one or more computing devices, the one or more inputs and providing the one or more mutated inputs to a neural network; obtaining, by the one or more computing devices as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays; determining, by the one or more computing devices based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage; and upon determining that the one or more mutated inputs provide new coverage, adding the one or more mutated inputs to the input corpus.



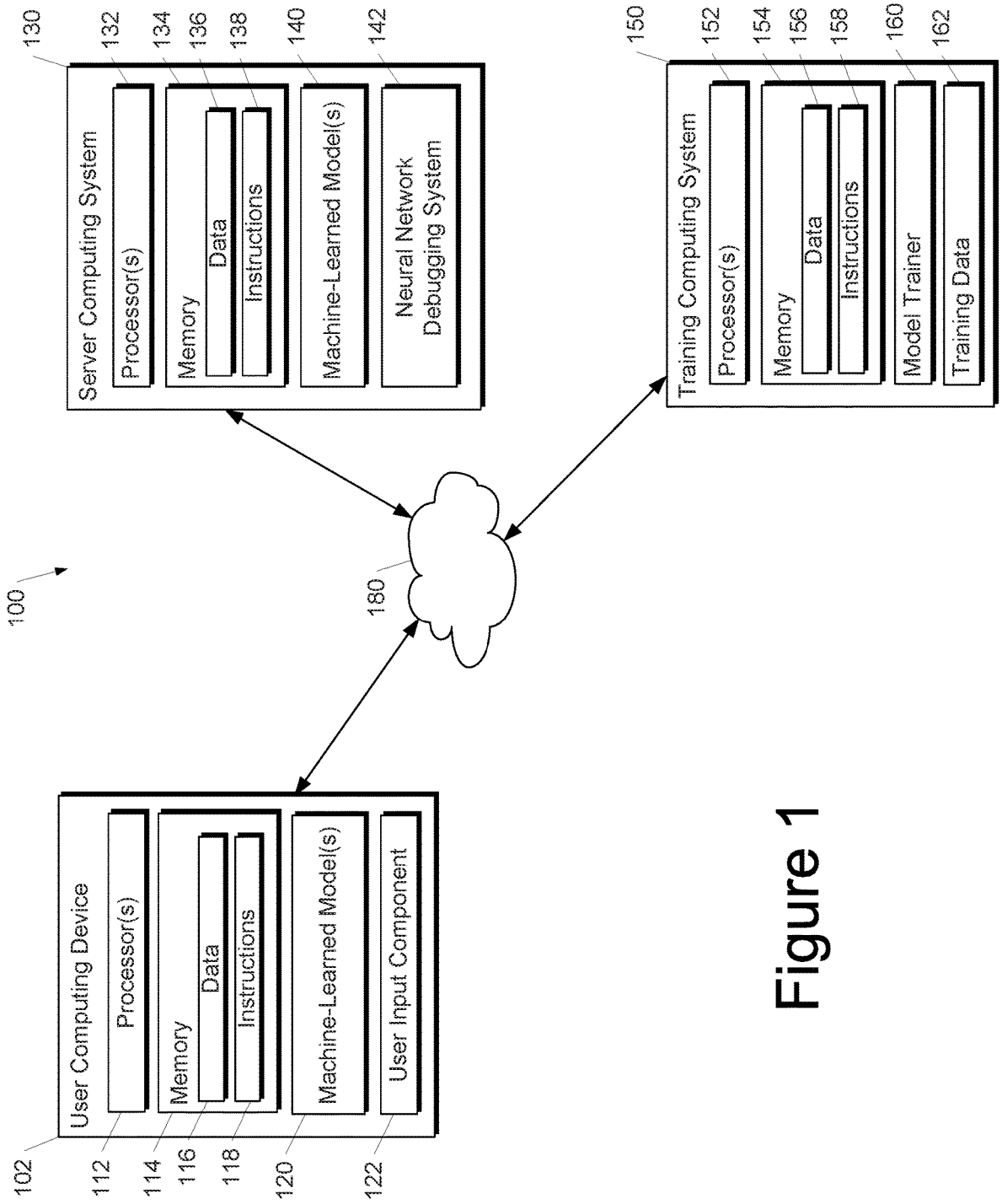

100
180
User Computing Device
Processor(s)
Memory
Data
Instructions
Machine-Learned Model(s)
User Input Component
102
112
114
116
118
120
122
Server Computing System
Processor(s)
Memory
Data
Instructions
Machine-Learned Model(s)
Neural Network Debugging System
130
132
134
136
138
140
142
Training Computing System
Processor(s)
Memory
Data
Instructions
Model Trainer
Training Data
150
152
154
156
158
160
162

Figure 1

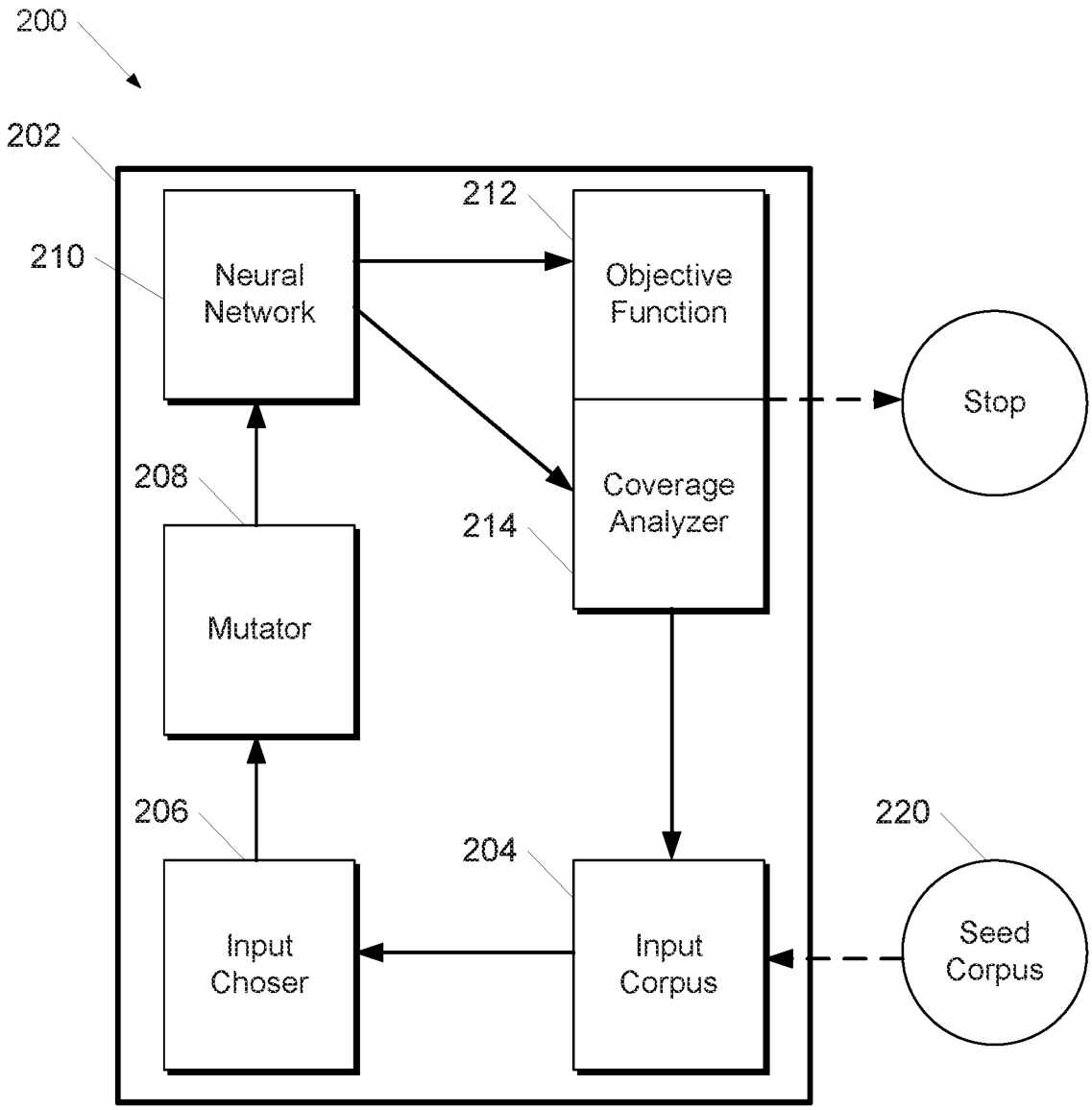

200
202
210
Neural Network
212
Objective Function
Stop
208
Mutator
214
Coverage Analyzer
206
Input Choser
204
Input Corpus
220
Seed Corpus

Figure 2

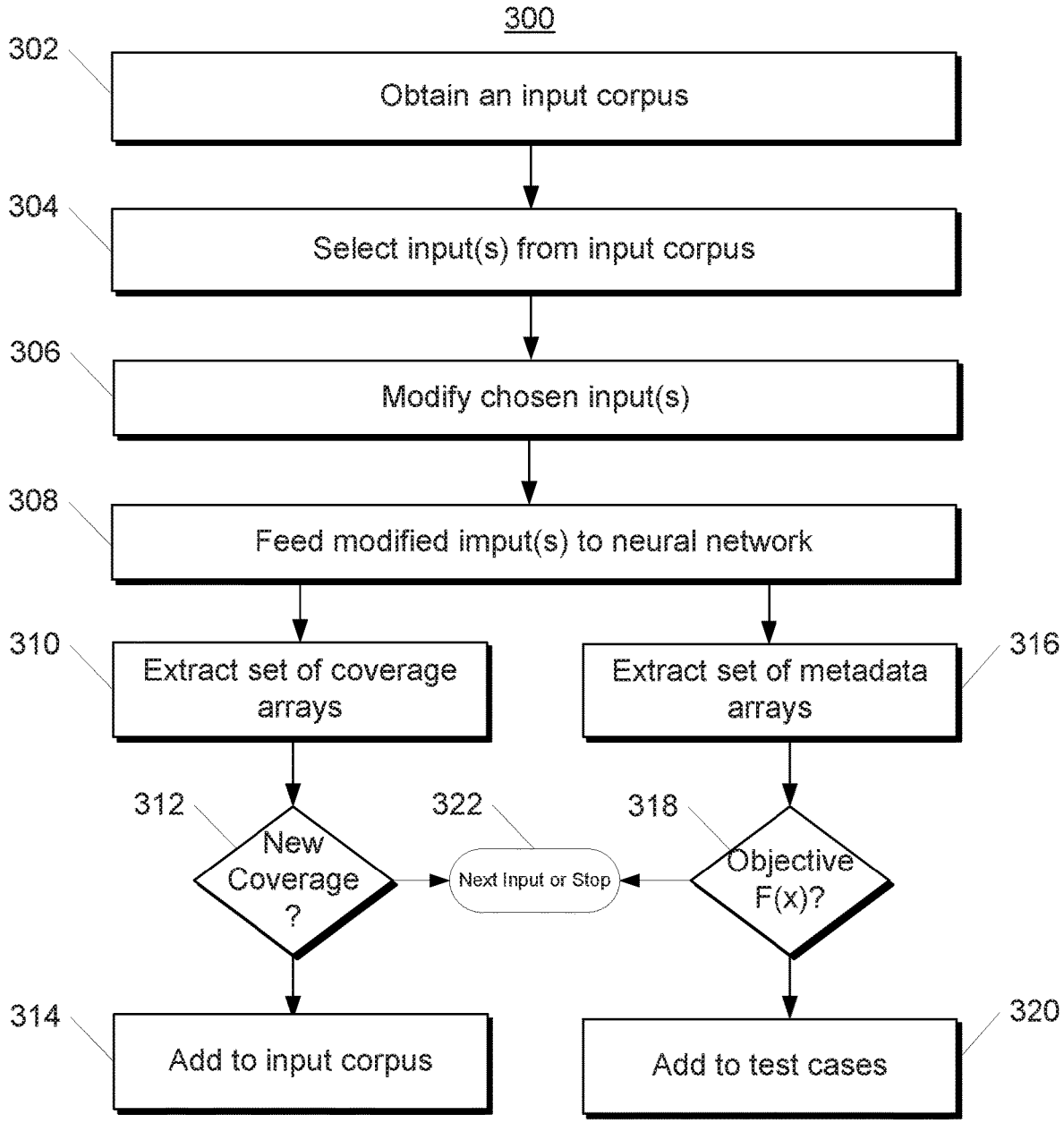

300
302
Obtain an input corpus
304
Select input(s) from input corpus
306
Modify chosen input(s)
308
Feed modified imput(s) to neural network
310
Extract set of coverage arrays
316
Extract set of metadata arrays
312
New Coverage ?
322
Next Input or Stop
318
Objective F(x)?
314
Add to input corpus
320
Add to test cases

Figure 3

SYSTEMS AND METHODS FOR DEBUGGING NEURAL NETWORKS WITH COVERAGE GUIDED FUZZING

[0001] The present application is based on and claims the benefit of U.S. Provisional Application No. 62/673,751 having a filing date of May 18, 2018, which is incorporated by reference herein in its entirety for all purposes.

FIELD

[0002] The present disclosure relates generally to machine learned models. More particularly, the present disclosure relates to finding undesirable behavior in neural networks.

BACKGROUND

[0003] The use of machine learning models, such as neural networks, is becoming more important in solving a variety of tasks that have traditionally been difficult for a computing system. However, machine learning models are generally difficult to interpret and debug. As machine learning models like neural networks become more prevalent, it becomes more desirable to test neural networks to discover bugs and/or other undesirable behavior before a neural network is implemented in the "real world."

SUMMARY

[0004] Aspects and advantages of embodiments of the present disclosure will be set forth in part in the following description, or can be learned from the description, or can be learned through practice of the embodiments.

[0005] One example aspect of the present disclosure is directed to debugging a neural network. The method can include obtaining, by one or more computing devices, one or more inputs from an input corpus. The method can further include mutating, by the one or more computing devices, the one or more inputs. The method can further include providing, by the one or more computing devices, the one or more mutated inputs to a neural network. The method can further include obtaining, by the one or more computing devices as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays that describe whether one or more neurons of the neural network were activated during processing of the one or more mutated inputs by the neural network. The method can further include determining, by the one or more computing devices based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage. The method can further include upon determining that the one or more mutated inputs provide new coverage, adding, by the one or more computing devices, the one or more mutated inputs to the input corpus.

[0006] In some implementations, the method can further include obtaining, by the one or more computing devices as a result of the neural network processing the one or more mutated inputs, a set of metadata arrays that describe metadata associated with execution of the neural network to process the one or more mutated inputs; determining, by the one or more computing devices based at least in part on the set of metadata arrays, whether an objective function is satisfied; and upon determining that the objective function is satisfied, adding, by the one or more computing devices, the one or more mutated inputs to a list of test cases.

[0007] Another example aspect of the present disclosure is directed to a computing device. The computing device can include one or more processors and one or more non-transitory computer-readable media that store instructions that, when executed by the one or more processors, cause the computing device to perform operations. The instructions, when executed, can cause the computing device to obtain one or more inputs from an input corpus. The instructions, when executed, can further cause the computing device to mutate the one or more inputs. The instructions, when executed, can further cause the computing device to provide the one or more mutated inputs to a neural network. The instructions, when executed, can further cause the computing device to obtain as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays that describe whether one or more neurons of the neural network were activated during processing of the one or more mutated inputs by the neural network The instructions, when executed, can further cause the computing device to determine based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage. The instructions, when executed, can further cause the computing device to, upon determining that the one or more mutated inputs provide new coverage, add the one or more mutated inputs to the input corpus.

[0008] In some implementations, the computing device can further include instructions, that when executed, cause the computing device to obtain as a result of the neural network processing the one or more mutated inputs, a set of metadata arrays that describe metadata associated with execution of the neural network to process the one or more mutated inputs; determine based at least in part on the set of metadata arrays, whether an objective function is satisfied; and upon determining that the objective function is satisfied, add the one or more mutated inputs to a list of test cases.

[0009] Another example aspect of the present disclosure is directed to one or more non-transitory computer-readable media that store instructions that, when executed by one or more processors of a computing system, cause the computing system to perform operations. The operations include obtaining one or more inputs from an input corpus. The operations further include mutating the one or more inputs. The operations further include providing the one or more mutated inputs to a neural network. The operations further include obtaining, as a result of the neural network processing the one or more mutated inputs, a set of metadata arrays that describe metadata associated with execution of the neural network to process the one or more mutated inputs. The operations further include determining, based at least in part on the set of metadata arrays, whether an objective function is satisfied. The operations further include upon determining that the objective function is satisfied, add the one or more mutated inputs to a list of test cases.

[0010] In some implementations, the one or more non-transitory computer-readable media can store instructions that, when executed by one or more processors of a computing system, cause the computing system to obtain, as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays that describe whether one or more neurons of the neural network were activated during processing of the one or more mutated inputs by the neural network; determine, based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage; and upon determining

that the one or more mutated inputs provide new coverage, add the one or more mutated inputs to the input corpus

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] Detailed discussion of embodiments directed to one of ordinary skill in the art is set forth in the specification, which makes reference to the appended figures, in which:

[0012] FIG. **1** depicts a block diagram of an example computing system that can be used with machine learning models according to example embodiments of the present disclosure.

[0013] FIG. **2** depicts a block diagram of an example coverage guided fuzzing system according to example embodiments of the present disclosure.

[0014] FIG. **3** depicts a flow chart diagram of example operations to perform neural network debugging according to example embodiments of the present disclosure.

[0015] Reference numerals that are repeated across plural figures are intended to identify the same features in various implementations.

DETAILED DESCRIPTION

[0016] Overview

[0017] Generally, the present disclosure is directed to machine-learned models, such as neural networks. In particular, the systems and methods of the present disclosure can provide for testing neural networks to find bugs and/or other undesirable behavior, for example, before a neural network is deployed. According to an aspect of the present disclosure, coverage guided fuzzing can be applied to neural networks to allow for debugging of the neural networks. For example, in some implementations, coverage guided fuzzing can be applied to neural networks to provide for finding numerical errors in trained neural networks, generating disagreements between neural networks and quantized versions of those networks, surfacing undesirable behavior in models (e.g., character level language models, etc.), and/or the like.

[0018] Machine learning models can be difficult to debug or interpret for a variety of reasons, ranging from the conceptual difficulty of specifying what the user wishes to know about the model in formal terms to statistical and computational difficulties in obtaining answers to formally specified questions. Neural networks can be particularly difficult to debug because even relatively straightforward formal questions about them can be computationally expensive to answer and because software implementations of neural networks can deviate significantly from theoretical models.

[0019] In general, coverage guided fuzzing provides for maintaining an input corpus comprising inputs to a program under consideration. Random changes are made to those inputs according to some mutation procedure, and the mutated inputs are added to an input corpus when they exercise new coverage (e.g., causing code to execute in a different way than previously seen, etc.).

[0020] The systems and methods of the present disclosure provide for inputting random mutations of inputs to a neural network where the mutations are guided by a coverage metric toward the goal of satisfying user-specified constraints. According to an aspect of the present disclosure, coverage can be measured by analyzing the activation vectors of the neural network coverage graph. For example, in some implementations, new coverage can be determined based on whether the neural network has resulted in a state that the neural network has not reached previously, such that the new coverage helps to provide incremental progress in debugging.

[0021] As one example, fast approximate nearest neighbor algorithms can be used to determine if two sets of neural network 'activations' are meaningfully different from each other. This provides a coverage metric producing useful results for neural networks, even when the underlying implementation of the neural network does not make use of many data-dependent branches. For example, in some implementations, the activations (or some subset of them) associated with each input can be stored and checked to determine whether coverage has increased on a given input by using an approximate nearest neighbors algorithm to see whether there are any other sets of activations within a pre-specified distance.

[0022] According to an aspect of the present disclosure, in some implementations, coverage guided fuzzing of a neural network can start with a seed corpus containing at least one set of inputs for the computation graph. The inputs can be restricted to those inputs that are in some sense valid neural network inputs. For example, if the inputs are images, the inputs can be restricted to those inputs that have the correct size and shape, and that lie in the same interval as the input pixels of the dataset under consideration. As another example, if the inputs are sequences of characters, inputs can be restricted to characters that are in the vocabulary extracted from the training set.

[0023] Given this seed corpus, until instructed to stop or some other stopping criterion in met, the neural network debugging system can choose elements from the input corpus according to some heuristic (e.g., uniform random selection, some defined probability heuristic, etc.). Given this input, the neural network debugging system can perform some sort of modification to that input. For example, the modification can be as simple as just flipping the sign of an input pixel in an image. Additionally or alternatively, it can also be restricted to follow some kind of constraint on the total modification made to a corpus element over time. The mutated inputs can then be fed to the neural network. In some implementations, two things can be extracted from the neural network: a set of coverage arrays, from which the actual coverage can be computed, and a set of metadata arrays, from which the result of the objective function can be computed. For example, the coverage arrays can describe which neurons of the neural network were activated during processing of the input, and therefore may be referred to as or used to generate an "activation vector." As another example, the metadata array can describe a behavior, output, result, prediction, outcome, timings, statistics, run times, memory consumption, processor usage, and/or other metadata associated with execution of the neural network to process the input. Once the coverage and/or objective is computed, the mutated input can be added to the corpus if it exercises new coverage, and/or it can be added to a list of test cases if it causes the objective function the be satisfied.

[0024] For example, an objective function can be used to assess whether some particular state (e.g., an erroneous state, etc.) has been reached. The objective function can be applied to the metadata arrays and inputs that caused the objective to be satisfied can be flagged. The neural network debugging system can determine whether the coverage

provided by the mutated input is new coverage (e.g., whether the neural network has reached a state that it has not reached previously, etc.) based on the coverage arrays. For example, in some implementations, when a new activation vector is received, its nearest neighbor can be determined (e.g., through performance of an approximate nearest neighbors algorithm) and checked for how far away the nearest neighbor is (e.g., in Euclidean distance). The input can be added to the corpus if the distance is greater than some defined amount.

[0025] In some implementations, the input mutations can be performed as a batch and the batch of inputs can be fed to the computation graph. The coverage and objective function can then be checked on a batch of output arrays.

[0026] Reference now will be made in detail to embodiments, one or more examples of which are illustrated in the drawings. Each example is provided by way of explanation of the embodiments, not limitation of the present disclosure. In fact, it will be apparent to those skilled in the art that various modifications and variations can be made to the embodiments without departing from the scope or spirit of the present disclosure. For instance, features illustrated or described as part of one embodiment can be used with another embodiment to yield a still further embodiment. Thus, it is intended that aspects of the present disclosure cover such modifications and variations.

Example Devices and Systems

[0027] FIG. **1** depicts a block diagram of an example computing system **100** that provides for the use of machine learning according to example embodiments of the present disclosure. The system **100** includes a user computing device **102**, a server computing system **130**, and a training computing system **150** that are communicatively coupled over a network **180**.

[0028] The user computing device **102** can be any type of computing device, such as, for example, a personal computing device (e.g., laptop or desktop), a mobile computing device (e.g., smartphone or tablet), a gaming console or controller, a wearable computing device, an embedded computing device, or any other type of computing device.

[0029] The user computing device **102** includes one or more processors **112** and a memory **114**. The one or more processors **112** can be any suitable processing device (e.g., a processor core, a microprocessor, an ASIC, a FPGA, a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. The memory **114** can include one or more non-transitory computer-readable storage mediums, such as RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. The memory **114** can store data **116** and instructions **118** which are executed by the processor **112** to cause the user computing device **102** to perform operations.

[0030] In some implementations, the user computing device **102** can store or include one or more machine-learned models **120**. For example, the machine-learned models **120** can be or can otherwise include various machine-learned models such as neural networks (e.g., deep neural networks) or other types of machine-learned models, including non-linear models and/or linear models. Neural networks can include feed-forward neural networks, recurrent neural networks (e.g., long short-term memory recurrent neural networks), convolutional neural networks or other forms of neural networks.

[0031] In some implementations, the one or more machine-learned models **120** can be received from the server computing system **130** over network **180**, stored in the user computing device memory **114**, and then used or otherwise implemented by the one or more processors **112**. In some implementations, the user computing device **102** can implement multiple parallel instances of a single machine-learned model **120**.

[0032] Additionally or alternatively, one or more machine-learned models **140** can be included in or otherwise stored and implemented by the server computing system **130** that communicates with the user computing device **102** according to a client-server relationship. For example, the machine-learned models **140** can be implemented by the server computing system **140** as a portion of a cloud based service. Thus, one or more models **120** can be stored and implemented at the user computing device **102** and/or one or more models **140** can be stored and implemented at the server computing system **130**.

[0033] The user computing device **102** can also include one or more user input component **122** that receives user input. For example, the user input component **122** can be a touch-sensitive component (e.g., a touch-sensitive display screen or a touch pad) that is sensitive to the touch of a user input object (e.g., a finger or a stylus). The touch-sensitive component can serve to implement a virtual keyboard. Other example user input components include a microphone, a traditional keyboard, or other means by which a user can provide user input.

[0034] The server computing system **130** includes one or more processors **132** and a memory **134**. The one or more processors **132** can be any suitable processing device (e.g., a processor core, a microprocessor, an ASIC, a FPGA, a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. The memory **134** can include one or more non-transitory computer-readable storage mediums, such as RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. The memory **134** can store data **136** and instructions **138** which are executed by the processor **132** to cause the server computing system **130** to perform operations.

[0035] In some implementations, the server computing system **130** includes or is otherwise implemented by one or more server computing devices. In instances in which the server computing system **130** includes plural server computing devices, such server computing devices can operate according to sequential computing architectures, parallel computing architectures, or some combination thereof.

[0036] As described above, the server computing system **130** can store or otherwise include one or more machine-learned models **140**. For example, the models **140** can be or can otherwise include various machine-learned models. Example machine-learned models include neural networks or other multi-layer non-linear models. Example neural networks include feed forward neural networks, deep neural networks, recurrent neural networks, and convolutional neural networks.

[0037] In some implementations, the server computing system **130** can further include a neural network debugging system **142**, such as described herein with regard to FIG. **2**. For example, the neural network debugging system **142** can

provide for performing coverage guided fuzzing using a corpus of inputs, for instance, to provide for testing neural networks, such as to discover errors which may occur for rare inputs.

[0038] The user computing device **102** and/or the server computing system **130** can train the models **120** and/or **140** via interaction with the training computing system **150** that is communicatively coupled over the network **180**. The training computing system **150** can be separate from the server computing system **130** or can be a portion of the server computing system **130**.

[0039] The training computing system **150** includes one or more processors **152** and a memory **154**. The one or more processors **152** can be any suitable processing device (e.g., a processor core, a microprocessor, an ASIC, a FPGA, a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. The memory **154** can include one or more non-transitory computer-readable storage mediums, such as RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. The memory **154** can store data **156** and instructions **158** which are executed by the processor **152** to cause the training computing system **150** to perform operations. In some implementations, the training computing system **150** includes or is otherwise implemented by one or more server computing devices.

[0040] The training computing system **150** can include a model trainer **160** that trains the machine-learned models **120** and/or **140** stored at the user computing device **102** and/or the server computing system **130** using various training or learning techniques, such as, for example, backwards propagation of errors. In some implementations, performing backwards propagation of errors can include performing truncated backpropagation through time. The model trainer **160** can perform a number of generalization techniques (e.g., weight decays, dropouts, etc.) to improve the generalization capability of the models being trained. In particular, the model trainer **160** can train the machine-learned models **120** and/or **140** based on a set of training data **162**.

[0041] In some implementations, if the user has provided consent, the training examples can be provided by the user computing device **102**. Thus, in such implementations, the model **120** provided to the user computing device **102** can be trained by the training computing system **150** on user-specific data received from the user computing device **102**. In some instances, this process can be referred to as personalizing the model.

[0042] The model trainer **160** includes computer logic utilized to provide desired functionality. The model trainer **160** can be implemented in hardware, firmware, and/or software controlling a general purpose processor. For example, in some implementations, the model trainer **160** includes program files stored on a storage device, loaded into a memory and executed by one or more processors. In other implementations, the model trainer **160** includes one or more sets of computer-executable instructions that are stored in a tangible computer-readable storage medium such as RAM hard disk or optical or magnetic media.

[0043] The network **180** can be any type of communications network, such as a local area network (e.g., intranet), wide area network (e.g., Internet), or some combination thereof and can include any number of wired or wireless links. In general, communication over the network **180** can be carried via any type of wired and/or wireless connection, using a wide variety of communication protocols (e.g., TCP/IP, HTTP, SMTP, FTP), encodings or formats (e.g., HTML, XML), and/or protection schemes (e.g., VPN, secure HTTP, SSL).

[0044] FIG. **1** illustrates one example computing system that can be used to implement the present disclosure. Other computing systems can be used as well. For example, in some implementations, the user computing device **102** can include the model trainer **160** and the training dataset **162**. In such implementations, the models **120** can be both trained and used locally at the user computing device **102**. In some of such implementations, the user computing device **102** can implement the model trainer **160** to personalize the models **120** based on user-specific data.

Example Debugging System Arrangement

[0045] FIG. **2** depicts a block diagram of an example neural network debugging system **200** using coverage guided fuzzing according to example embodiments of the present disclosure. In some implementations, the neural network debugging system **200** can provide for performing coverage guided fuzzing using a corpus of inputs, for example, to provide for testing neural networks, such as to discover errors which may occur for rare inputs. The neural network debugging system **200** can allow for guiding mutations to corpus inputs by a coverage metric to work toward a goal of satisfying user-specified constraints (e.g., random changes are made to inputs according to some mutation procedure and the mutated inputs are added to an input corpus when they exercise new coverage). As an example, coverage can be measured by analyzing the activation vectors of the neural network coverage graph. For instance, in some implementations, new coverage can be determined based on whether the neural network has resulted in a state that the neural network has not reached previously, such that the new coverage helps to provide incremental progress in debugging of the neural network model. For example, in some implementations, coverage guided fuzzing can be applied to neural networks to provide for finding numerical errors in trained neural networks, generating disagreements between neural networks and quantized versions of those networks, surfacing undesirable behavior in models, and/or the like.

[0046] As illustrated in FIG. **2**, a neural network debugging system **200** can include a coverage guided fuzzer **202** and a seed corpus **220** (e.g., containing at least one set of inputs for the computation graph) which can provide for an initial set of inputs to a coverage guided fuzzer **202** to test a neural network.

[0047] The coverage guided fuzzer **202** can obtain (e.g., select) a set of inputs from the seed corpus **220** to provide an input corpus **204**, which may comprise all or some subset of inputs included in the seed corpus **220**. In some implementations, the inputs can be restricted to some type of valid neural network inputs (e.g., images having a correct size and shape, characters that are in a vocabulary extracted from a training set, etc.). In some implementations, the seed corpus **220** can be supplied by a user and/or can be selected from a set of available seed corpuses. The inputs can be textual inputs, image inputs, audio data inputs, sensor data inputs, and/or various other types of inputs.

[0048] The coverage guided fuzzer **202** can include an input chooser **206** that can select input(s) from the input

corpus **204** to use during a particular iteration of the coverage guided fuzzing. For example, in some implementations, the input chooser **206** can select inputs using uniform random selection. For example, in some implementations, the input chooser **206** can be biased towards selecting inputs that were more recently added to the input corpus **204**. As one example, the input chooser **206** can select inputs using a heuristic such as

$$p(c_k, t) = \frac{e^{t_k - t}}{\sum e^{t_k - t}},$$

wherein $p(c_k, t)$ gives a probability of choosing an input corpus element c_k at time t where t_k is the time when element c_k was added to the input corpus. The intuition behind this is that recently sampled inputs are more likely to yield useful new coverage when mutated, but that this advantage decays as time progresses, and thus inputs can be selected as a function of their age.

[0049] The input chooser **206** can provide the selected input(s) to a mutator **208**. The mutator **208** can apply modifications (e.g., mutations) to the selected input(s) before the inputs are provided to the neural network. For example, in some implementations, the mutator **208** can add white noise of a user-configurable variance to input(s) (e.g., image inputs, etc.). As another example, in some implementations, the mutator **208** can add white noise of a user-configurable variance to the one or more inputs (e.g., image inputs, etc.), wherein a difference between the mutated input and an original input from which the mutated input is descended is constrained to have a user-configurable L_∞ norm. This type of constrained mutation can be useful to find inputs that satisfy some objective function, but are still plausibly of the same "class" as the original input that was used as a seed. In some implementations, the image can be clipped after mutation so that it lies in the same range as the inputs used to train the neural network being debugged.

[0050] As another example, in some implementations, such as with text string inputs, one of a set of operations can be uniformly performed at random, including operations such as deleting a character at a random location, adding a character at a random location, substituting a random character at a random location, and/or the like.

[0051] The input chooser **206** includes computer logic utilized to provide desired functionality. The input chooser **206** can be implemented in hardware, firmware, and/or software controlling a general purpose processor. For example, in some implementations, the input chooser **206** includes program files stored on a storage device, loaded into a memory and executed by one or more processors. In other implementations, the input chooser **206** includes one or more sets of computer-executable instructions that are stored in a tangible computer-readable storage medium such as RAM hard disk or optical or magnetic media.

[0052] The mutator **208** can then provide the mutated input(s) to the neural network **210**. The neural network **210** can provide outputs which can include a set of coverage arrays, for which coverage can be computed, and a set of metadata arrays, from which a result of an objective function can be computed. For example, when the mutated inputs are fed into a computation graph, both coverage arrays and metadata arrays are returned as output.

[0053] The mutator **208** includes computer logic utilized to provide desired functionality. The mutator **208** can be implemented in hardware, firmware, and/or software controlling a general purpose processor. For example, in some implementations, the mutator **208** includes program files stored on a storage device, loaded into a memory and executed by one or more processors. In other implementations, the mutator **208** includes one or more sets of computer-executable instructions that are stored in a tangible computer-readable storage medium such as RAM hard disk or optical or magnetic media.

[0054] The objective function **212** can assess whether the neural network has reached some particular state, for example, a state which may be regarded as erroneous, based on the metadata array(s). An erroneous state may include an incorrect prediction, an execution time greater than a maximum execution time, a processor usage greater than a maximum processor usage, a failure of the neural network to execute, and/or other the existence of other errors or undesirable behavior or performance. In some implementations, the objective function **212** can be specified by a user and/or selected from a set of available objective functions. Generally, the objective function **212** used to assess whether the neural network has reached some particular state can be separate and distinct from some other objective or loss function used to train the neural network. If the objective function **212** is satisfied, the mutated input(s) provided to the neural network can be flagged, such as being added to a list of test cases (e.g., for future debugging, etc.). As an example, when the mutated inputs are fed into a computation graph and metadata arrays are returned as output, the objective function can be applied to the metadata arrays and any mutated inputs that caused the objective function to be satisfied can be flagged.

[0055] The coverage analyzer **214** can determine whether the coverage provided by the mutated input(s) is new coverage (e.g., whether the neural network has reached a state that it has not reached previously, etc.) based on the coverage array(s). For example, in some implementations, coverage analyzer **214** can determine whether new coverage is provided based on whether an activation vector is approximately close to a previous activation vector. If the coverage analyzer **214** determines that the mutated input(s) provide new coverage, the mutated input(s) can be added to the input corpus **204**, for example, to be used as input(s) in future iterations of debugging and/or the like. For example, an approximate nearest neighbor can be computed for a new activation vector and checked to determine how far away the nearest neighbor is in Euclidean distance from the activation vector. The input can be added to the corpus if the distance is greater than some defined amount (e.g., which can be a user-configurable hyperparameter, an adaptive hyperparameter to adapts over time, and/or a dynamic hyperparameter that changes over time, for example, according to a predetermined schedule). In some implementations, the coverage guided fuzzer **202** can continue to select, mutate, and analyze inputs included in the input corpus **204** until instructed to stop and/or some other stopping criterion in met.

[0056] The coverage analyzer **214** includes computer logic utilized to provide desired functionality. The coverage analyzer **214** can be implemented in hardware, firmware, and/or software controlling a general purpose processor. For example, in some implementations, the coverage analyzer

214 includes program files stored on a storage device, loaded into a memory and executed by one or more processors. In other implementations, the coverage analyzer **214** includes one or more sets of computer-executable instructions that are stored in a tangible computer-readable storage medium such as RAM hard disk or optical or magnetic media.

[0057] The coverage arrays and/or associated activation vectors may describe whether some or all of the neurons of the neural network were activated during processing of an input. As one example, a coverage array and/or associated activation vector may be limited to describing only whether the logits of the neural network and/or neurons of a layer of the network prior to the logits were activated.

[0058] In some implementations, the system **200** can be applied (e.g., in parallel) to two or more different (but potentially related) models to identify disagreements between the models. For example, the two or more different models can be two or more different versions of a base model such as a base model and a quantized version of the base model. To identify disagreements, the same input (e.g., a mutated input) can be provided to the two or more different models and the two or more different outputs of the two or more different models can be analyzed (e.g., according to the objective function **212** and/or the coverage analyzer **214**) to detect disagreements or otherwise measure a divergence in the outputs.

Example Methods

[0059] FIG. **3** depicts a flow chart diagram of example operations to perform neural network debugging according to example embodiments of the present disclosure. Although FIG. **3** depicts steps performed in a particular order for purposes of illustration and discussion, the methods of the present disclosure are not limited to the particularly illustrated order or arrangement. The various steps of the method **300** can be omitted, rearranged, combined, and/or adapted in various ways without deviating from the scope of the present disclosure.

[0060] At **302**, a computing system can obtain an input corpus, for example from a seed corpus comprising one or more sets of inputs. For example, a seed corpus can contain at least one set of inputs for the computation graph. The inputs can be restricted to those inputs that are in some sense valid neural network inputs. For example, if the inputs are images, the inputs can be restricted to those inputs that have the correct size and shape, and that lie in the same interval as the input pixels of the dataset under consideration. As another example, if the inputs are sequences of characters, inputs can be restricted to characters that are in the vocabulary extracted from the training set.

[0061] At **304**, the computing system can select one or more inputs from the input corpus for use in debugging a neural network. For example, the computing system can select one or more inputs from the input corpus based on uniform random selection, based on one or more heuristics (e.g.,

$$p(c_k, t) = \frac{e^{t_k - t}}{\sum e^{t_k - t}}$$

giving a probability of choosing an input corpus element c_k at time t where t_k is the time when element c_k was added to the input corpus, etc.), and/or the like.

[0062] At **306**, the computing system can modify the selected input(s) prior to input to the neural network by performing some type of mutation on the selected input(s). For example, in some implementations, the computing system can perform a simple modification of the input such as flipping a sign of an input. As another example, in some implementations, computing system can restrict the modifications to follow a constraint on the total modification made to a corpus element over time.

[0063] At **308**, the computing system feed the modified input(s) to the neural network that is to be debugged.

[0064] At **310**, the computing system can obtain, as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays (e.g., that describe whether one or more neurons of the neural network were activated during processing of the one or more mutated inputs by the neural network) which can be used to compute the actual coverage exercised by the modified input(s).

[0065] At **312**, the computing system the computing system can determine whether the mutated input(s) provide new coverage at least in part on the coverage array(s). For example, the computing system can determine that new coverage is provided is the neural network results in a state that it has not been in before. If the mutated input(s) provide new coverage, operation continues to **314**. If the mutated input(s) do not provide new coverage, operations continue to **322**, where a next input can be analyzed. For example, in some implementations, when a new activation vector is received, its nearest neighbor can be determined and checked for how far away the nearest neighbor is in Euclidean distance. The input can be added to the corpus if the distance is greater than some defined amount.

[0066] At **314**, the computing system can add the mutated input(s) to the input corpus.

[0067] At **316**, the computing system, as a result of the neural network processing the one or more mutated inputs, a set of metadata arrays (e.g., that describe metadata associated with execution of the neural network to process the one or more mutated inputs) for use in computing the objective function.

[0068] At **318**, the computing system can determine whether the objective function is satisfied based at least in part on the metadata array(s). For example, the objective function can assess whether the neural network has reached a particular state, such as state that is regarded as erroneous. For example, the objective function can be applied to the metadata arrays and inputs that cause the objective to be satisfied can be flagged. If the objective function is satisfied, operation continues to **320**. If the objective function is not satisfied, operation continues to **322**.

[0069] At **320**, the computing system can add the mutated input to a list of test cases.

Additional Disclosure

[0070] The technology discussed herein makes reference to servers, databases, software applications, and other computer-based systems, as well as actions taken and information sent to and from such systems. The inherent flexibility of computer-based systems allows for a great variety of possible configurations, combinations, and divisions of tasks and functionality between and among components. For

instance, processes discussed herein can be implemented using a single device or component or multiple devices or components working in combination. Databases and applications can be implemented on a single system or distributed across multiple systems. Distributed components can operate sequentially or in parallel.

[0071] While the present subject matter has been described in detail with respect to various specific example embodiments thereof, each example is provided by way of explanation, not limitation of the disclosure. Those skilled in the art, upon attaining an understanding of the foregoing, can readily produce alterations to, variations of, and equivalents to such embodiments. Accordingly, the subject disclosure does not preclude inclusion of such modifications, variations and/or additions to the present subject matter as would be readily apparent to one of ordinary skill in the art. For instance, features illustrated or described as part of one embodiment can be used with another embodiment to yield a still further embodiment. Thus, it is intended that the present disclosure cover such alterations, variations, and equivalents.

What is claimed is:

1. A computer-implemented method for debugging a neural network, the method comprising:

obtaining, by one or more computing devices, one or more inputs from an input corpus;

mutating, by the one or more computing devices, the one or more inputs;

providing, by the one or more computing devices, the one or more mutated inputs to a neural network;

obtaining, by the one or more computing devices as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays that describe whether one or more neurons of the neural network were activated during processing of the one or more mutated inputs by the neural network;

determining, by the one or more computing devices based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage; and

upon determining that the one or more mutated inputs provide new coverage, adding, by the one or more computing devices, the one or more mutated inputs to the input corpus.

2. The method of claim **1**, further comprising:

obtaining, by the one or more computing devices as a result of the neural network processing the one or more mutated inputs, a set of metadata arrays that describe metadata associated with execution of the neural network to process the one or more mutated inputs;

determining, by the one or more computing devices based at least in part on the set of metadata arrays, whether an objective function is satisfied; and

upon determining that the objective function is satisfied, adding, by the one or more computing devices, the one or more mutated inputs to a list of test cases.

3. The method of claim **1**, wherein determining, by the one or more computing devices based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage comprises:

generating, by the one or more computing devices, an activation vector based at least in part on the set of coverage arrays;

performing, by the one or more computing devices, an approximate nearest neighbors algorithm to identify a previous activation vector;

determining, by the one or more computing devices, a distance between the activation vector and the previous activation vector identified by the approximate nearest neighbors algorithm; and

comparing, by the one or more computing devices, the distance to a threshold distance;

wherein the one or more mutated inputs provide new coverage when the distance is greater than the threshold distance.

4. The method of claim **1**, wherein obtaining one or more inputs from the input corpus comprises using uniform random selection to select the one or more inputs from the input corpus.

5. The method of claim **1**, wherein obtaining one or more inputs from the input corpus comprises selecting the one or more inputs from the input corpus using a heuristic of

$$p(c_k, t) = \frac{e^{t_k - t}}{\sum e^{t_k - t}},$$

wherein $p(c_k, t)$ gives a probability of choosing input corpus element c_k at time t where t_k is the time when element c_k was added to the input corpus.

6. The method of claim **1**, wherein mutating the one or more inputs comprises adding white noise of a user-configurable variance to the one or more inputs.

7. The method of claim **1**, wherein mutating the one or more inputs comprises adding white noise of a user-configurable variance to the one or more inputs, wherein a difference between the mutated input and an original input from which the mutated input is descended is constrained to have a user-configurable L_∞ norm.

8. The method of claim **1**, wherein determining whether the one or more mutated inputs provide new coverage comprises determining whether the neural network has reached a new state that it has not previously reached.

9. The method of claim **8**, wherein determining whether the neural network has reached a new state that it has not previously reached comprises determining whether an activation vector is approximately close to a previous activation vector.

10. The method of claim **2**, wherein determining whether the objective function is satisfied comprises determining whether the neural network has reached a desired state.

11. The method of claim **10**, wherein the desired state is an erroneous state for the neural network.

12. A computing device comprising:

one or more processors; and

one or more non-transitory computer-readable media that store instructions that, when executed by the one or more processors, cause the computing device to:

obtain one or more inputs from an input corpus;

mutate the one or more inputs;

provide the one or more mutated inputs to a neural network;

obtain as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays that describe whether one or more neurons of the

neural network were activated during processing of the one or more mutated inputs by the neural network;

determine based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage; and

upon determining that the one or more mutated inputs provide new coverage, add the one or more mutated inputs to the input corpus.

13. The computing device of claim **12**, further comprising instructions, that when executed, cause the computing device to:

obtain as a result of the neural network processing the one or more mutated inputs, a set of metadata arrays that describe metadata associated with execution of the neural network to process the one or more mutated inputs;

determine based at least in part on the set of metadata arrays, whether an objective function is satisfied; and

upon determining that the objective function is satisfied, add the one or more mutated inputs to a list of test cases.

14. The computing device of claim **12**, further comprising instructions, that when executed, cause the computing device to:

obtain the input corpus from a seed corpus, the seed corpus containing at least one set of inputs.

15. The computing device of claim **12**, wherein obtaining one or more inputs from the input corpus comprises:

using uniform random selection to select the one or more inputs from the input corpus; or

selecting the one or more inputs from the input corpus using a heuristic of

$$p(c_k, t) = \frac{e^{t_k - t}}{\sum e^{t_k - t}},$$

wherein $p(c_k, t)$ gives a probability of choosing input corpus element c_k at time t where t_k is the time when element c_k was added to the input corpus.

16. The computing device of claim **12**, wherein mutating the one or more inputs comprises:

adding white noise of a user-configurable variance to the one or more inputs;

adding white noise of a user-configurable variance to the one or more inputs, wherein a difference between the mutated input and an original input from which the mutated input is descended is constrained to have a user-configurable L_{inf} norm; or

performing one of a set of operations uniformly at random.

17. The computing device of claim **12**, wherein determining whether the one or more mutated inputs provide new coverage comprises determining whether the neural network has reached a new state that it has not previously reached; and

wherein determining whether the neural network has reached a new state that it has not previously reached comprises determining whether an activation vector is approximately close to a previous activation vector.

18. The computing device of claim **13**, wherein determining whether the objective function is satisfied comprises determining whether the neural network has reached a desired state, wherein the desired state is an erroneous state for the neural network.

19. One or more non-transitory computer-readable media that store instructions that, when executed by one or more processors of a computing system, cause the computing system to perform operations, the operations comprising:

obtaining one or more inputs from an input corpus;

mutating the one or more inputs;

providing the one or more mutated inputs to a neural network;

obtaining, as a result of the neural network processing the one or more mutated inputs, a set of metadata arrays that describe metadata associated with execution of the neural network to process the one or more mutated inputs;

determining, based at least in part on the set of metadata arrays, whether an objective function is satisfied; and

upon determining that the objective function is satisfied, adding the one or more mutated inputs to a list of test cases.

20. The one or more non-transitory computer-readable media of claim **19**, wherein the operations further comprise:

obtaining, as a result of the neural network processing the one or more mutated inputs, a set of coverage arrays that describe whether one or more neurons of the neural network were activated during processing of the one or more mutated inputs by the neural network;

determining, based at least in part on the set of coverage arrays, whether the one or more mutated inputs provide new coverage; and

upon determining that the one or more mutated inputs provide new coverage, adding the one or more mutated inputs to the input corpus.

* * * * *