RL_Verif: verification of neural networks for baggage routing

Igor Buzhinsky

igor.buzhinsky@gmail.com

Aalto University
School of Electrical
Engineering

ITMO UNIVERSITY

September 23, 2020

Preface: the best algorithm by D. Mukhutdinov

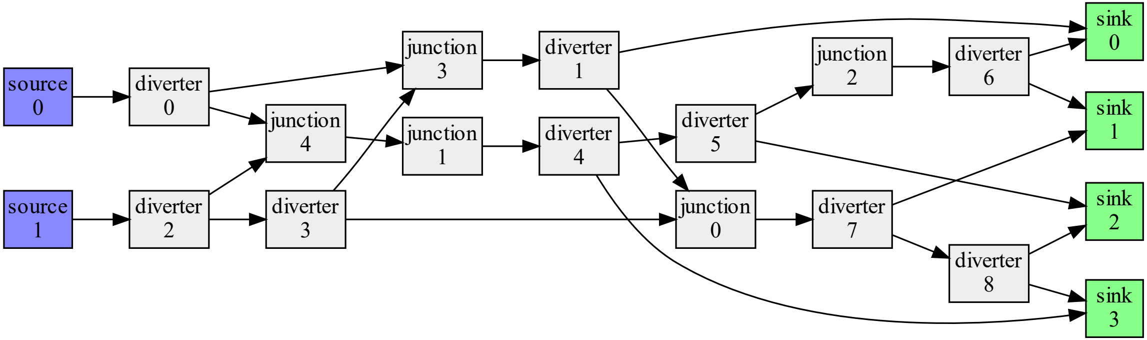
Introduction

- dqnroute (`https://github.com/flyingleafe/dqnroute`): library by D. Mukhutdinov
    - Simulation models: baggage delivery, package delivery in computer networks
    - Reinforcement learning for distributed routing
- Neural network (NN)
    - Input: current node, delivery target, candidate neighbor
    - Each node is passed as an embedding (10 dimensions for a graph with 20 nodes)
    - Output: Q value (expected cumulative future reward) of sending the bag to this neighbor
    - Pretraining + training during operation
- Assumptions of these slides
    - Fixed topology graph (in particular, no conveyor failures)
    - Check delivery from a fixed source to a fixed sinc
    - Frozen network (or a network being changed during one learning step; this will be mentioned separately)

Routing decisions

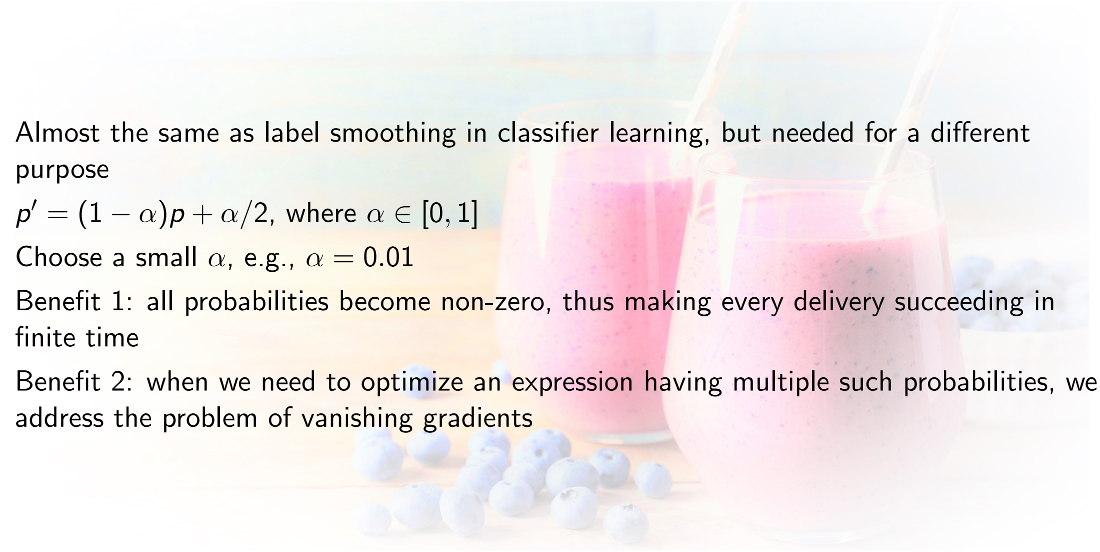
- In baggage handling network, routers have up to 2 successors
- Router holders
    - **Sources** and **junctions** have exactly 1 successor, routing decision is fixed
    - **Sinks** do not have successors and do not perform routing decisions
    - **Diverters** have 2 successors, but due to **reachability shielding** one of the options can be blocked
- We are interested in non-shielded routers in junctions
- Stochastic routing decision
    - Hyperparameter T: the higher the temperature, the higher the entropy of the decision
    - Mukhutdinov: $(p, 1-p) = \text{softmax}((q_1, q_2)/T)$
    - Equivalently, $p = \text{sigmoid}((q_1 - q_2)/T)$

Conveyor graph used by D. Mukhutdinov



- We'd like to test that a bag from a chosen source is delivered to the chosen sink
- But assuming that all probabilities are non-zero, any delivery will always succeed in finite time!

A slight change: probability smoothing



- Almost the same as label smoothing in classifier learning, but needed for a different purpose
- $p' = (1 - \alpha)p + \alpha/2$, where $\alpha \in [0, 1]$
- Choose a small α, e.g., $\alpha = 0.01$
- Benefit 1: all probabilities become non-zero, thus making every delivery succeeding in finite time
- Benefit 2: when we need to optimize an expression having multiple such probabilities, we address the problem of vanishing gradients

NN verification tools

General ideas

- NN is assumed to a feed-forward and represented as a **composition of affine layers and element-wise nonlinearities**
- **Nonlinearities are often restricted to be ReLUs**, sometimes arbitrary non-decreasing functions
- Verification results: counterexamples, reachability regions, adversarial robustness bounds
- Some common ideas
  - Compute bounds on activations of each neuron with interval arithmetic, with explicit or symbolic computation (layer by layer)
  - Refine all reachable multidimensional bodies (layer by layer)
  - Split the current hyperrectangle into two
- Some tools are sound-and-complete (usually slower), some are only sound (usually faster)

Charon (2019)

- Anderson, Greg, et al. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2019
- `https://github.com/gavlegoat/charon`
- Sound-and-complete, supports ReLUs only (but may potentially support other nonlinearities), verifies adversarial robustness of classification
- Approach
  - Represent the current verification domain as a finite union of zonotopes (Minkowski sums of line segments)
  - Try to find a counterexample: adversarial search with projected gradient descent (PGD)
  - Try to prove absence of counterexamples: zonotope analysis with the ELINA library
  - If both options fail, split the region and try recursively
  - Learn the splitting policy and the the domain policy (how many disjuncts to use) with Bayesian optimization

Marabou (2019)

- Katz, Guy, et al. The Marabou framework for verification and analysis of deep neural networks. International Conference on Computer Aided Verification. Springer, Cham, 2019.
- `https://github.com/NeuralNetworkVerification/Marabou`
- Sound-and-complete, supports ReLUs only (but may potentially support other piecewise-linear nonlinearities), verifies reachability of an output space region defined by a number of hyperplanes
- Approach: SMT-based
- Easy to compile, currently used in this project
- More recent work based on this tool: Elboher, Y. Y., Gottschlich, J., & Katz, G. (2020, July). An abstraction-based framework for neural network verification. In International Conference on Computer Aided Verification (pp. 43-65). Springer, Cham.

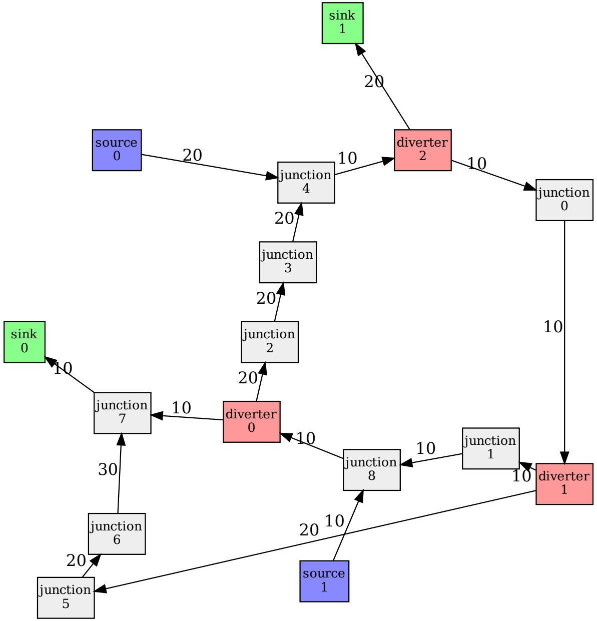
Towards verification

What is the model of the attacker?

- A momentary out-of-distribution adversarial disruption?
    - This is something to be verified for a NN
    - **Currently, only this view on the situation is considered**
- A change in bag input distribution to which the algorithm cannot adapt?
    - This is something to be verified for a learning algorithm
    - May be much more difficult to verify

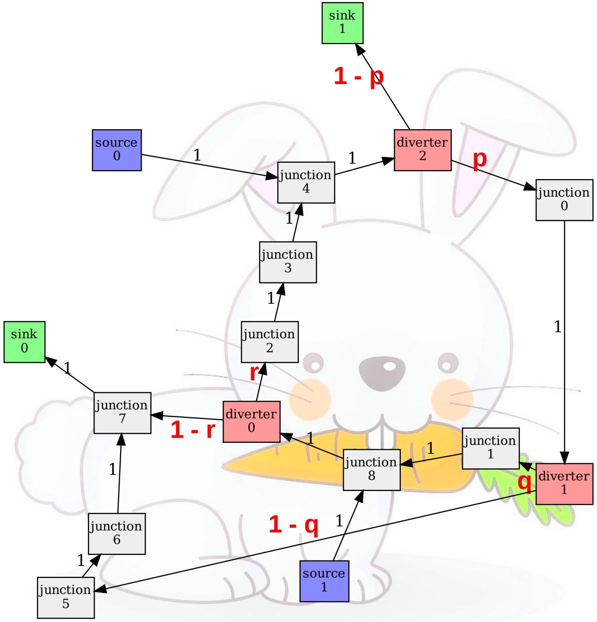
Let's consider a graph with a cycle

- The numbers on edges are conveyor section lengths
- Select a source (e.g., source 0) and a sink (e.g., sink 0)
- Recall that routing decisions are stochastic, and they cannot be made deterministic due to the need to expore during learning
- Verification with deterministic decisions will be unrealistic
- With stochastic decisions, bad probabilities of routing decisions may lead to high expected delivery time



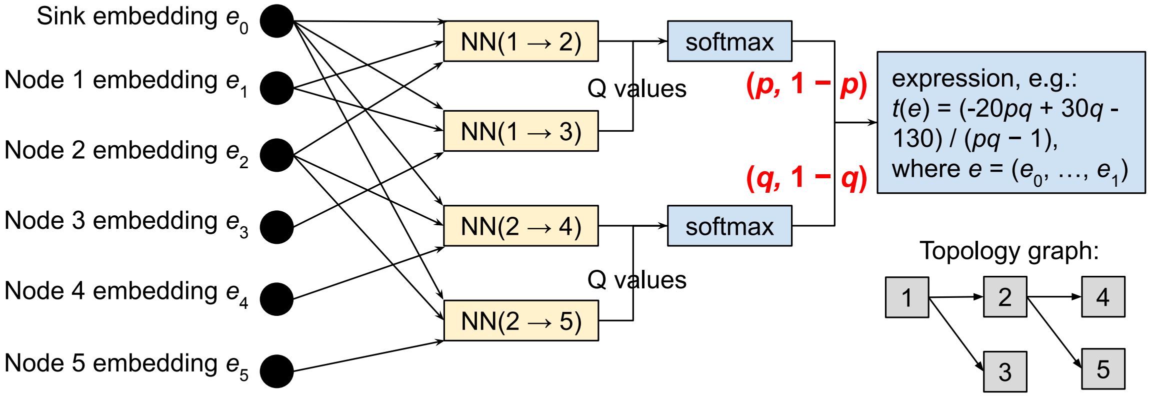
Markovian analysis

- If the NN is fixed, we can treat the topology graph as a Markov chain
- Markovian analysis to determine **expected hitting times** (possibly with weighted edges) of sinks
- Will get a **ratio of two polynomials** over probabilities generated by the NN (in this example, with *p*, *q*, *r*)
- This expression can be obtained symbolically (e.g., with the sympy library), by solving a system of linear equations



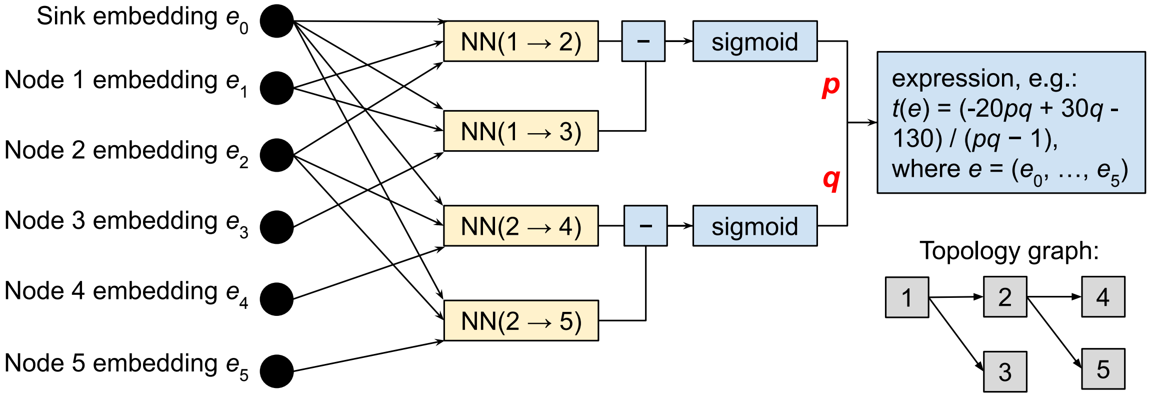
Verification of adversarial robustness w.r.t. input embeddings for a fixed NN

What happens: computation graph of the expected hitting time



- Definition: a **concatenated embedding** e is a concatenation of embeddings of all nodes that are needed to compute expected hitting time t
- Goal: verify that $\forall e \in E \; t(e) < t_0$ for some threshold t_0 for some E
- But softmax is not supported by NN verification tools!

Applying a trick



- For two arguments, softmax can be rewritten as an affine layer (minus) and sigmoid, which is a monotonic element-wise function
- For a particular probability (e.g., p or q), can verify that it is within a given range
- Two network executions can be modeled as a single execution

Verification problem in more detail

- Suppose that e_0 is the concatenated embedding computed based on the current topology graph (Mukhutdinov: Laplacian Eigenmaps embeddings)
- Suppose that $t_\theta(e)$ is the expected hitting time for concatenated embedding e and NN parameters θ
- For a fixed NN with parameters θ, some ϵ, t_0 and $||\cdot||$, verify **adversarial robustness**: $\forall e\,(||e - e_0|| < \epsilon \Rightarrow t_\theta(e) < t_0)$
  - $||\cdot||$ is usually $||\cdot||_2$ (Euclidean) or $||\cdot||_\infty$ (max-norm)

Proposed idea of a solution

- Maintain a hyperrectangle R of probability vectors $\mathbf{p} = (p_1, ..., p_n)$, where n is the number of diverters with nontrivial routing decisions
- Start with $R = [\alpha/2, 1 - \alpha/2]^n$, where α is the smoothing parameter
- Prove or refute $\forall \mathbf{p} \in R \; t \leq t_0$ with CSP/SMT solvers
  - If this is true, we have a proof
- With a verification tool, find out whether R is reachable (for some allowed e) or not
  - For a fixed R, can bypass sigmoid and verify a constraint on logits
  - If R is unreachable, we have a proof
  - If R is reachable and $t(\mathbf{p}) > t_0$, where $\mathbf{p}$ was found by the verification tool, we have a counterexample
- If no conclusion has been made, split R and try recursively
  - In this case, we already have reachable $\mathbf{p}$ for one of the parts
  - The simplest strategy is to split the longest dimension of R in two halves

Currently implemented partial solution

- For a fixed current node, destination node and neighbor: verify that small L_∞ changes in input embeddings do not alter the predicted Q value too much
- For a fixed diverter and destination node: verify that small L_∞ changes in input embeddings do not alter the predicted local routing probability too much
  - Two NN executions (for both neighbors) are modeled as one by constructing weight matrices from blocks

Verification of adversarial robustness w.r.t. a single learning step

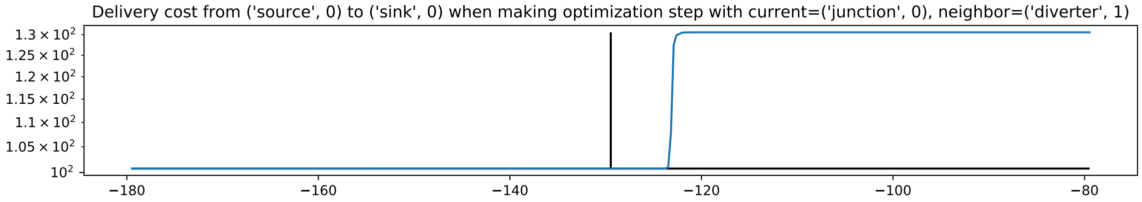
Accounting for a single learning step

- Adversarial robustness w.r.t. network parameters
  - Note: this requires “swapping” NN inputs and parameters
    - An MLP is transformed to a non-MLP
    - This may cause difficulties with verification tools
  - Much more parameters than inputs $\rightarrow$ lower robustness expected
  - But the NN is not changed arbitrarily during training!
- NN training as implemented by D. Mukhutdinov
  - MSE loss: minimize $L = \mathrm{E}(Q_{\mathrm{Predicted}} - Q_{\mathrm{NewlyEstimated}})^2$
  - RMSprop: the update of the parameters on each training step is nonlinear w.r.t. the loss gradient
  - Other adaptive optimizers also behave nonlinearly
  - Out of scope: how the newly estimated Q value is obtained

Adversarial robustness w.r.t. a training step

- On each training step, we have:
    - Current node, delivery target, candidate neighbor
    - Q value discrepancy (signed)
- For each source and sink, if the topology graph has n nodes with routers, then can examine $2n$ **linear restrictions** of the NN
    - A linear restriction of a NN is a one-argument function that executes this network
    - This one argument is the Q value discrepancy
- Work [Sotoudeh M., and Aditya V. Thakur. Computing linear restrictions of neural networks. NeurIPS 2019]
    - This work only considers piecewise-linear (e.g., ReLU) networks
    - But we have nonlinearities due to the learning step and getting probabilities from predicted Q values
- In any case, one-argument function look much more amenable to analysis

Adversarial robustness w.r.t. a training step visualized



- For simplicity, assume that the parameter step $\Delta\theta$ is linear w.r.t. parameter gradient (like in SGD) $\nabla_\theta L$: $\Delta\theta = -\alpha \nabla_\theta L$, where α is the learning rate
- X axis: target Q value (black vertical line corresponds to zero discrepancy $\Delta Q = 0$ with the predicted value)
- Y axis: $t_{\theta+\Delta\theta}(e)$ (expected delivery cost measured in meters) when making optimization step from junction 0 to diverter 1

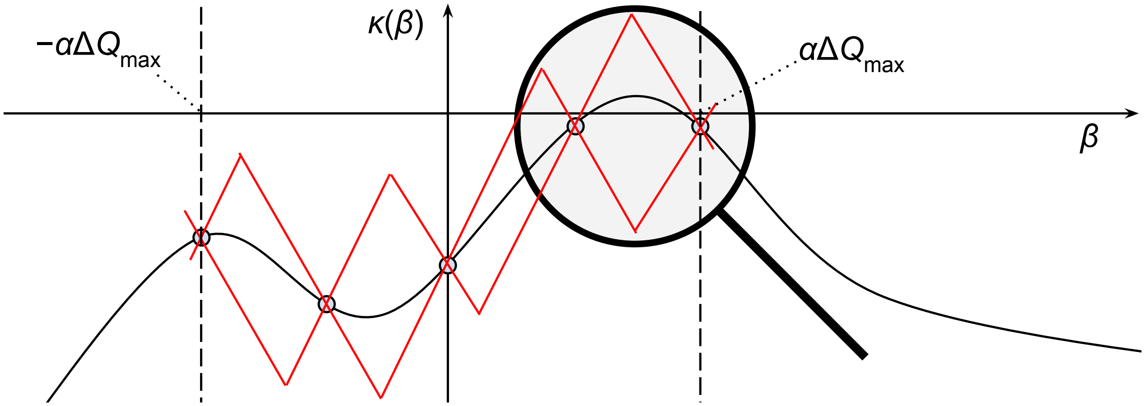
Verification problem that we get

- Problem: for a fixed source, sink and NN parameters θ and maximum allowed Q value discrepancy ΔQ_{max}, verify that $t_{\theta+\Delta\theta}(e) < t_0$ for all possible learning steps
- The number k of possible learning steps does not exceed $2m$, where m is the number of the nodes in the topology graph
- Consider $\tau(\beta) = t_{\theta+\beta\nabla_\theta L(\Delta Q)}(e)$, where:
    - $e = \text{const}$
    - $\beta = -\alpha\Delta Q$
    - $|\Delta Q| \leq \Delta Q_{\text{max}}$
- Now we need to check $\tau(\beta) < t_0$ for all possible β

Proposed solution

- $\tau(\beta) = u(\mathbf{p}(\beta))/v(\mathbf{p}(\beta))$, where u and v are some (symbolically computed) polynomials
- $\tau(\beta)$ is difficult to operate with due to the denominator
- Due to probability smoothing, $v \neq 0$
  - Although, can both u and v be zeros with non-zeros probabilities?
- v is also continuous, and thus sign-constant
- We can compute $\mathrm{sgn}(v(\mathbf{p}))$ at any $\mathbf{p}$ that is possible according to probability smoothing, and ensure that it is 1
- Consider $\kappa(\beta) = u(\mathbf{p}(\Delta Q)) - t_0 v(\mathbf{p}(\Delta Q))$
- Now we need to check that $\kappa(\beta) < 0$
- Can compute $\kappa(\beta)$ on a grid of points and estimate its value in other points by finding a bound on the Lipschitz constant

Using the Lipschitz constant



- Here, more precise investigation of the rightmost interval will make us see the counterexample

Computing the Lipschitz constant

- Lipschitz constant: $K = \sup\{|\kappa'(\beta)| : |\beta| \le \alpha \Delta Q_{\mathsf{max}}\}$
- $\kappa(\beta)$ can be computed symbolically
  - This is an expression with sums, products, ReLUs and sigmoids
- $\kappa'(\beta)$ can be computed symbolically
  - This is an expression with sums, products, ReLUs, sigmoids, and the Heaviside function (derivative of ReLU)
  - Due to the Heaviside function, it is undefined in a finite number of points, but in the expression for $\kappa(\beta)$, it is fine to exclude these points from consideration
- A function with sigmoids is hard to analyze
  - Instead we can find the bounds on the derivatives on all logits inside these sigmoids, and then get an (imprecise) estimate of K
  - For our NN architecture, the derivative of logits turned out to be discontinuous piecewise quadratic functions!

Checking dynamical isometry

Checking dynamical isometry (1)

- An isometry is a distance-preserving function
- Arip Asadulaev's research: a certain form of regularization (Jacobian clamping) sometimes improves training speed and the quality of the learned policy
- Specified in the grant application
- Dynamical isometry
    - For all inputs, all singular values of the Jacobian matrix of the NN are ones
    - Only for most inputs that occur during training?
    - Weaken 1 to $O(1)$? Looks like that the closer to one, the better
- For a fixed network, we can verify that these singular values are within a certain range

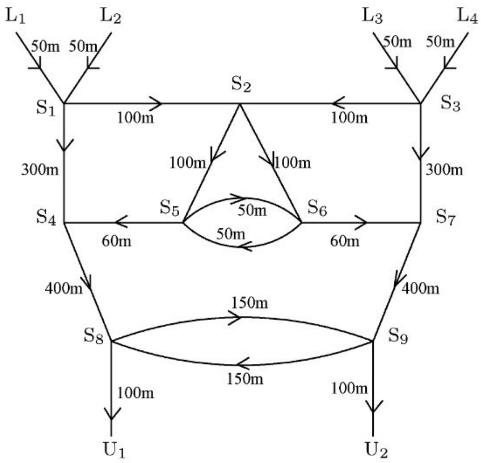
Checking dynamical isometry (2)

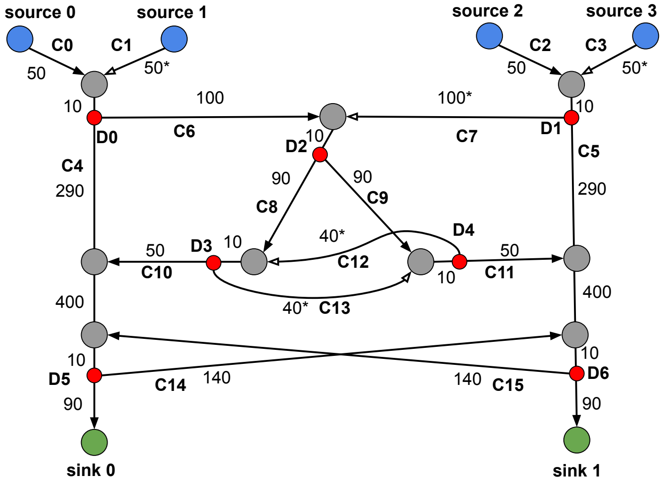
- Singular value of matrix A are the square roots of the non-zero eigenvalues of $A^\top A$
- Compute eigenvalues of an $m \times m$ matrix with $2m$ parameters, where m is the dimension of the NN input
- But this is a rank 1 matrix, and we can obtain the single singular value symbolically!
- For a network $\mathcal{N}$ with scalar output, this is $||\nabla_x \mathcal{N}(x)||_2$
- This will be an expression with affine operations, ReLUs and Heavisides
- Can solve with CSP / SMT solvers
- But it is not meaningful to check for an arbitrary network
  - It was shown [J. Pennington, S. Schoenholz, S. Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. NIPS 2017] that ReLU networks cannot achieve it, but sigmoid networks can
  - Training techniques: orthogonal weight initialization, Jacobian clamping
- To sum up, this looks possible, but we first need to learn networks that have a chance to be dynamically isometric and only then verify them

Misc

More interesting graphs with cycles (1)

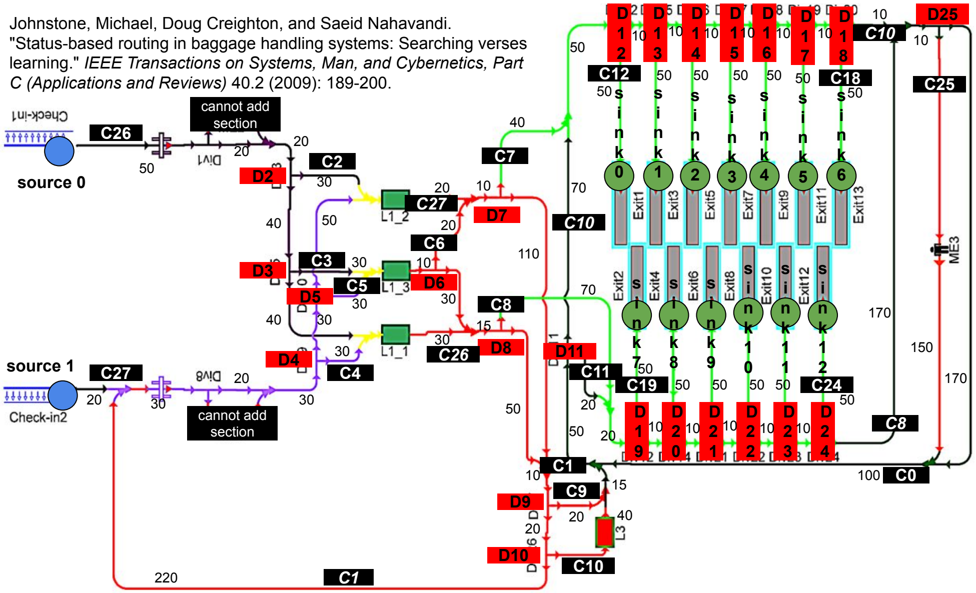
Tarau, Alina N., Bart De Schutter, and Hans Hellendoorn. "Model-based control for route choice in automated baggage handling systems." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.3 (2010): 341-351.





More interesting graphs with cycles (2)

Johnstone, Michael, Doug Creighton, and Saeid Nahavandi. "Status-based routing in baggage handling systems: Searching verses learning." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.2 (2009): 189-200.



Literature

- Mukhutdinov, Dmitry, et al. Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system. Future Generation Computer Systems 94 (2019): 587–600
- Liu, Changliu, et al. Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758 (2019)
- Anderson, Greg, et al. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2019
- Katz, Guy, et al. The Marabou framework for verification and analysis of deep neural networks. International Conference on Computer Aided Verification. Springer, Cham, 2019.
- Sotoudeh, Matthew, and Aditya V. Thakur. Computing linear restrictions of neural networks. Advances in Neural Information Processing Systems. 2019