Google

Optimizing over trained neural networks

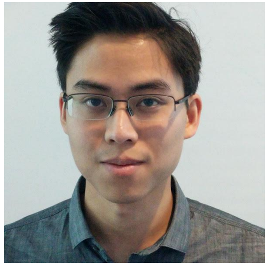
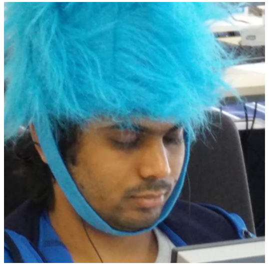
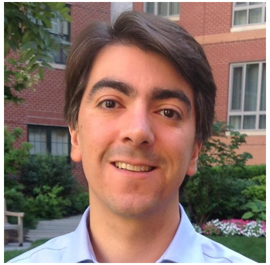
Ross Anderson

Google Research

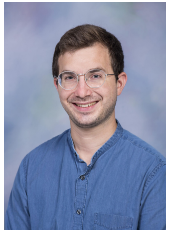
Operations Research team

MIT ORC IAP Seminar 2020

A large collaboration

Google OR Team
C. Tjandraatmadja
K. Patel
J.P. Vielma
A. Delarue
O. Sykora



X-Google OR Team
J. Huchette
W. Ma
Y. Ng



Google Research
C. Boutilier
Y. Chow
M. Ryu

Optimizing over a trained neural network

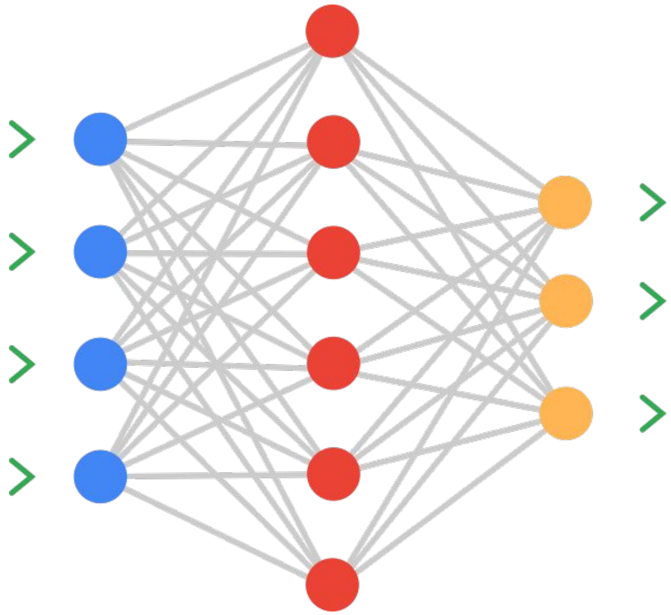
Given a trained neural network NN(x),

$$\begin{aligned} \text{maximize} \quad & \mathrm{NN}(x) \\ \text{such that} \quad & x \in \Omega \end{aligned}$$

Domain may be complex,
e.g. a mixed-integer set

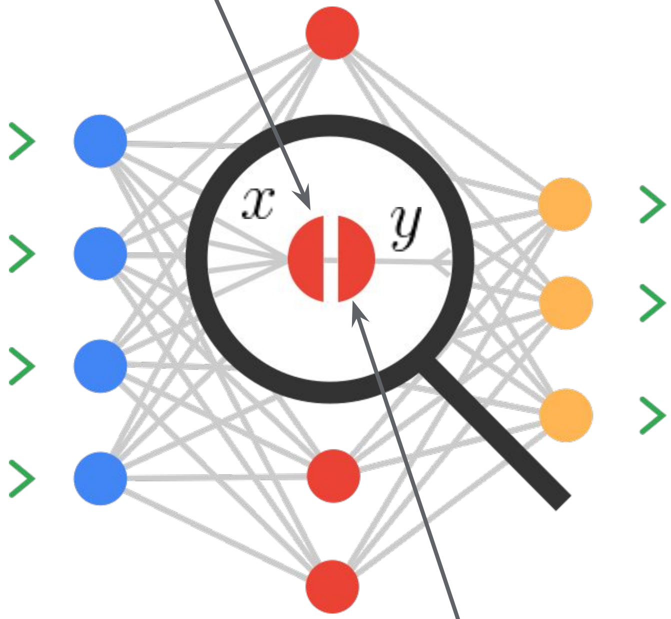
In applications NN(x) is typically in the objective, but it is not much different to have them in constraints

Feedforward neural networks



Feedforward neural networks

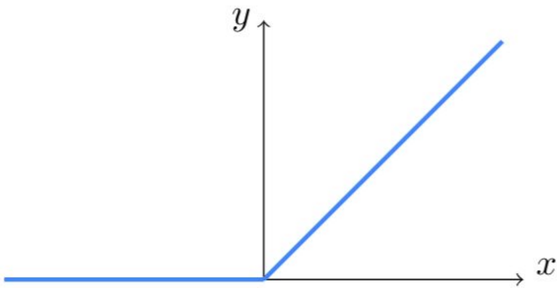
(typically)
affine preactivation function



activation function

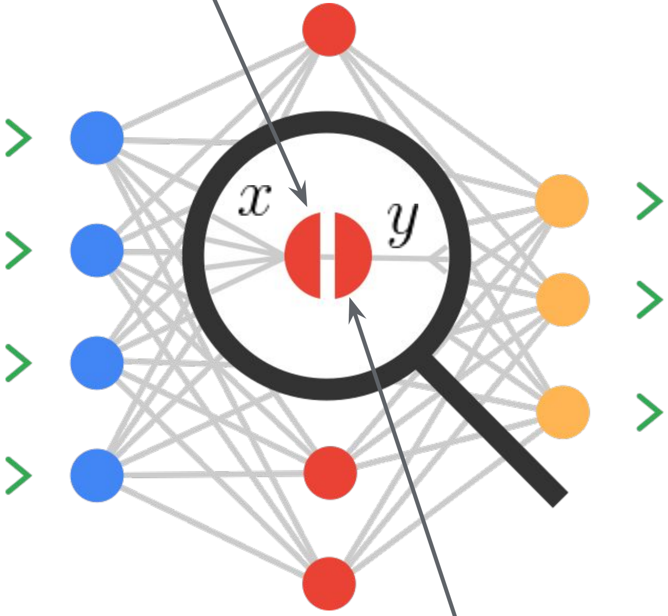
Example of an activation function:
Rectified Linear Unit (ReLU)

$$y = \max(0, x)$$



Feedforward neural networks

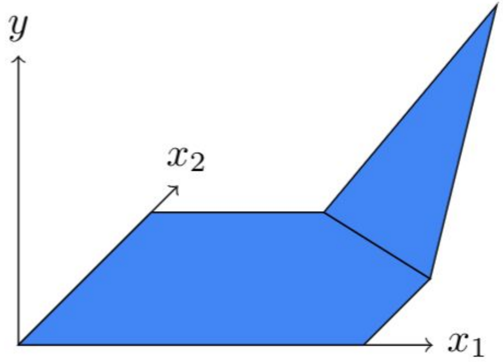
(typically)
affine preactivation function



activation function

Rectified Linear Unit (ReLU)
+ preactivation function

$$y = \max(0, w^\top x + b)$$



Outline

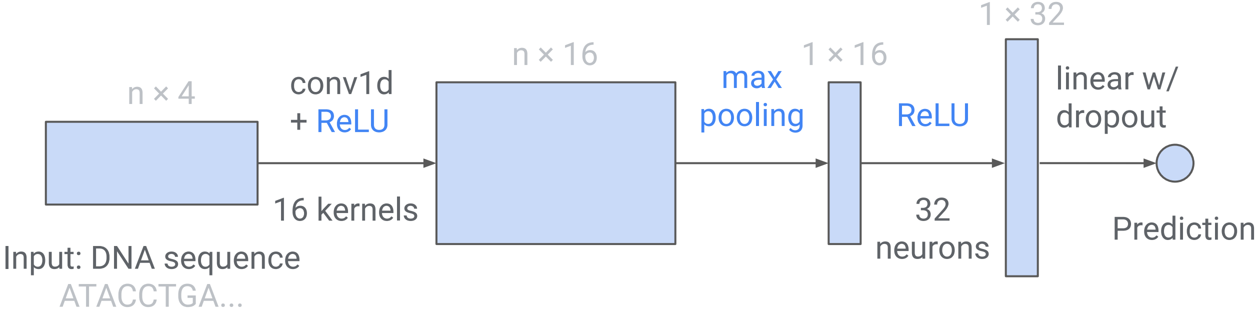
- Applications of optimizing over trained neural networks
  - “Predict and optimize”
  - Neural network verification
  - Large action space ADP
  - Bayesian optimization
- Solution Methods
- Computational Results
- Conclusion

“Predict and optimize” for DNA-protein binding

Predict the binding strength of a DNA sequence with a given transcription factor protein

Example of an architecture from Zeng et al., 2016
“Convolutional neural network architectures for predicting DNA–protein binding”

n = length of input



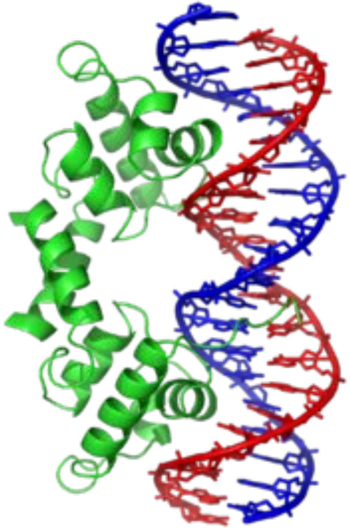


Image credit: Zephyris at English Wikipedia

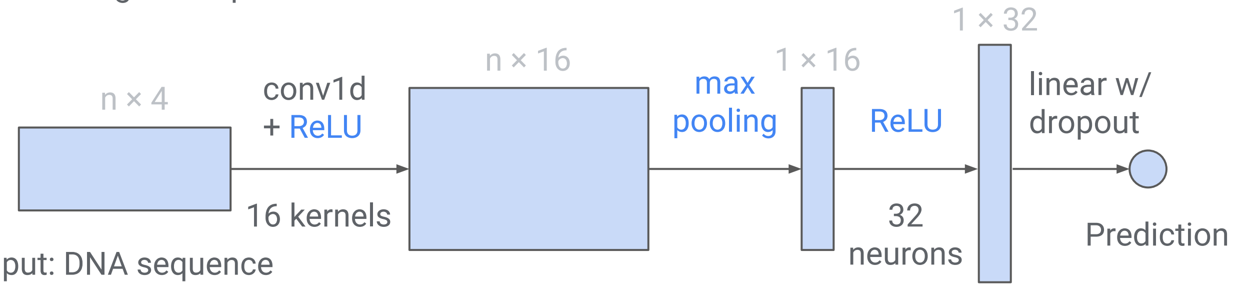
“Predict and optimize” for DNA-protein binding

Design a DNA sequence that maximizes

~~Predict~~ the binding strength ~~of a DNA sequence~~ with a given transcription factor protein

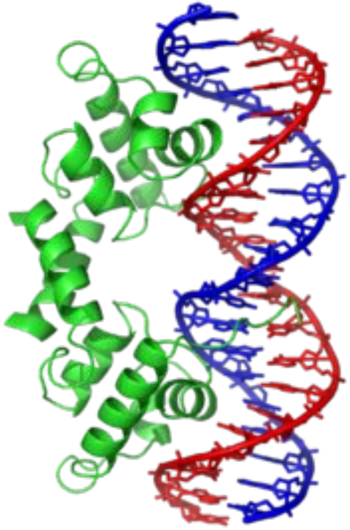
Example of an architecture from Zeng et al., 2016
“Convolutional neural network architectures for predicting DNA–protein binding”

n = length of input



Input: DNA sequence
ATACCTGA...

Decision variables



“Predict and optimize” for DNA-protein binding, does it work?

- How to measure success? Ground truth $f(x)$ available in **TfBind8**.
- $f(\text{random } x) < f(\arg\max_x \mathrm{NN}(x)) \ll \max_{x \in \text{training data}} f(x)$, why?
    - Inaccurate model?
    - Good interpolation, bad extrapolation?
    - Optimizer’s curse ($\mathbb{E}[\mathrm{NN}(x^*) - f(x^*)] < 0$ even if unbiased)
- Multiple guesses? Stay near the data? Instead, see Bayesian optimization

More “predict and optimize” from the literature

Empirical decision model learning: Thermal-aware workload dispatching

(Lombardi, Milano, Bartolini, 2017, http://emlopt.github.io)

Surrogate modeling: Oil production

(Grimstad, Andersson, 2019)

Microgrid Islanding with Frequency Constraints: Electric grid design

(Zhang, Chen, Liu, Hong, Qiu 2020)

Outline

- Applications of optimizing over trained neural networks
  - "Predict and optimize"
  - Neural network verification
  - Large action space ADP
  - Bayesian optimization
- Solution Methods
- Computational Results
- Conclusion

Neural network verification and adversarial examples

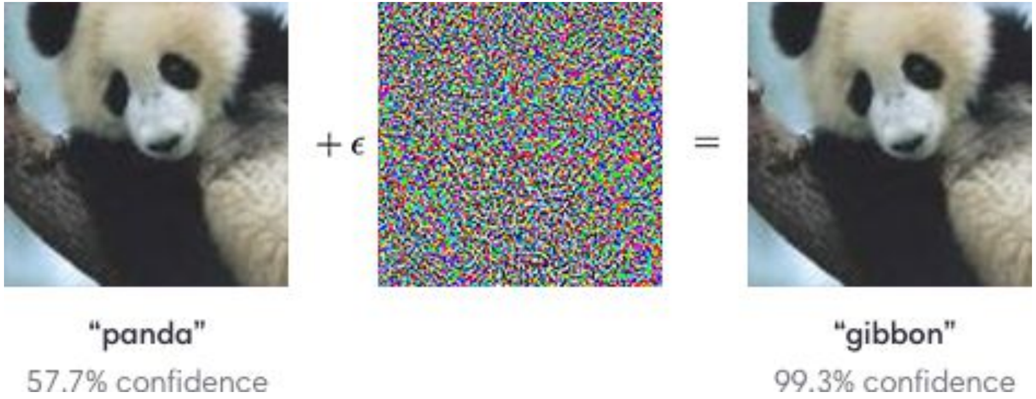


Image from:
Goodfellow et al., *"Explaining and Harnessing Adversarial Examples"*, 2015

Interest is high (≥ 2998 citations)

$$\begin{aligned} \max \quad & \mathbb{P}[x \text{ is a gibbon}] \\ \text{s.t.} \quad & ||x - (\text{reference panda})||_\infty \le \epsilon \end{aligned}$$

Neural network verification and adversarial examples

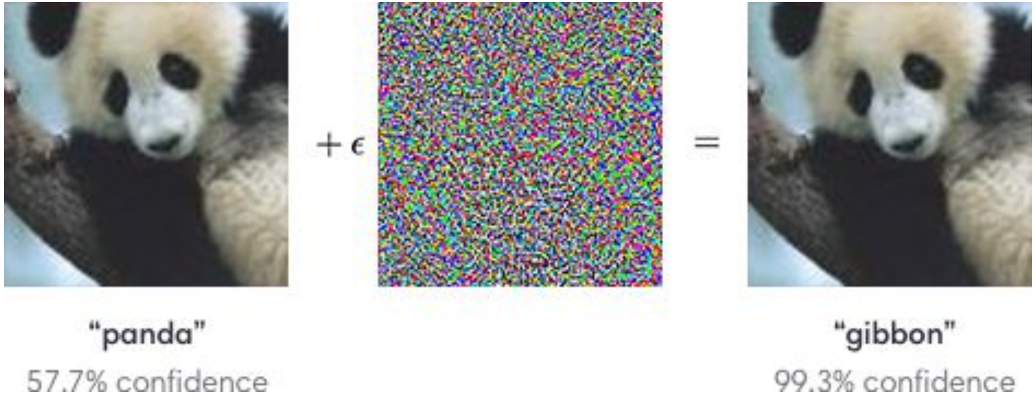


Image from:
Goodfellow et al., *"Explaining and Harnessing Adversarial Examples"*, 2015

Interest is high (≥ 2998 citations)

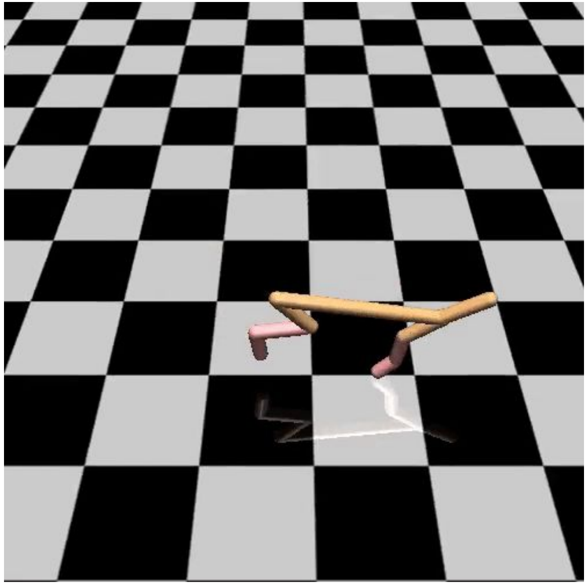
Dual bounds provide guarantees for robustness

MIP-based verifiers: Cheng et al. 2017, Lomuscio and Maganti 2017, Dutta et al. 2018, Fischetti and Jo 2018, Tjeng et al. 2019
Also many SMT-based verifiers (e.g. Katz et al. 2017, Ehlers 2017)

Outline

- Applications of optimizing over trained neural networks
  - “Predict and optimize”
  - Neural network verification
  - Large action space ADP
  - Bayesian optimization
- Solution Methods
- Computational Results
- Conclusion

Reinforcement learning



Half-Cheetah model
in MuJoCo

State x: Cheetah joints position, velocity, etc.

Action a: Joint movements

Immediate reward R(x,a): Distance moved

State transition f: Defined by joint physics

Goal: $$\max_a \sum_t \underbrace{\gamma^t}_{\text{discount factor}} R(x_t, a_t)$$

where $x_{t+1} = f(x_t, a_t)$

(Deep) Q-learning

Find optimal Q* for Bellman equation:

$$\underbrace{Q(x_t, a_t)}_{\text{Q-function: cumulative reward}} = \underbrace{R(x_t, a_t)}_{\text{immediate reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(x_{t+1}, a_{t+1})}_{\text{cumulative reward for next action}}$$

(Deep) Q-learning

Find optimal Q* for Bellman equation:

$$Q(x_t, a_t) = R(x_t, a_t) + \gamma \max_{a_{t+1}} Q(x_{t+1}, a_{t+1})$$

$$Q(x, a) \approx \mathrm{NN}(x, a)$$

Maximize Q-function (with e.g. MIP)

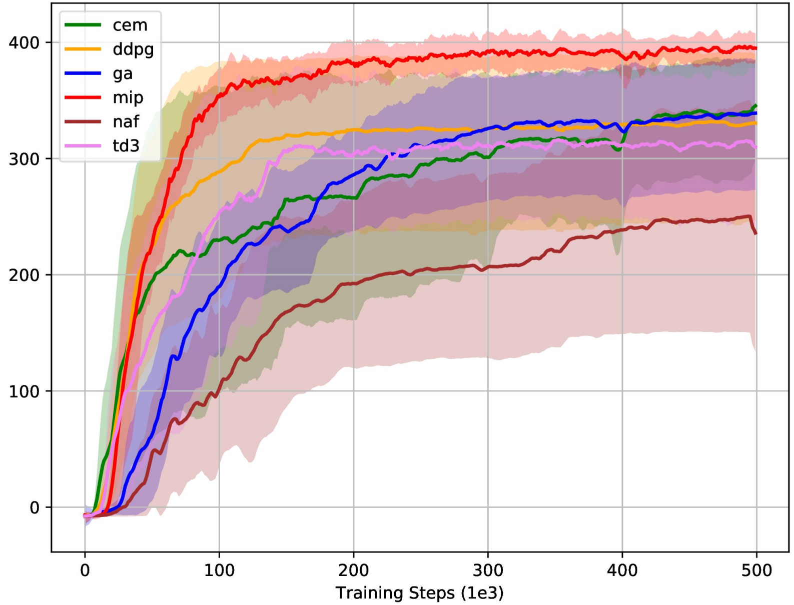
Run policy $\xrightarrow{\text{Reward data}}$ Update Q-function

Update Q-function $\xrightarrow{\text{NN weights}}$ Run policy

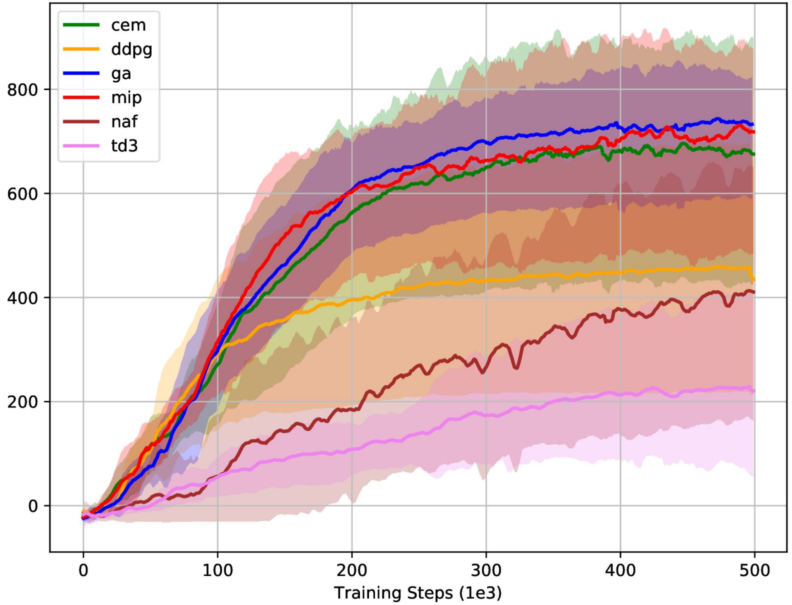
Update neural network

Continuous Action Q-learning

Half-Cheetah



Action range [-0.25, 0.25]



Action range [-0.5, 0.5]

Ryu, Chow, A., Tjandraatmadja, Boutilier, *"CAQL: Continuous Action Q-Learning"*
https://arxiv.org/abs/1909.12397

Challenges in (Large Action Space) RL

- High variability in results
- High sensitivity to (many) hyperparameters
- Large amounts of computation needed
    - One MIP per time step
    - Many steps to learn (RL is not sample efficient)
    - Many replications (for variability)
    - Many experiments (for hyperparameters)
- Computations must be either reliable or fault tolerant

Outline

- Applications of optimizing over trained neural networks
  - “Predict and optimize”
  - Neural network verification
  - Large action space ADP
  - Bayesian optimization
- Solution Methods
- Computational Results
- Conclusion

Bayesian optimization: maximize black-box f

Use a probabilistic model:

- $f(x; w)$ in a parametric family
- Assume prior $\mathbb{P}(w)$
- Bayes' rule gives posterior after data D:

$$\mathbb{P}(w|D) = \frac{\mathbb{P}(D|w)\mathbb{P}(w)}{\mathbb{P}(D)}$$

Explore promising points under $\mathbb{P}(w|D)$:

- Expected Improvement: $\max_x \mathbb{E}_w[(f(x) - y^*)^+]$
- Upper Confidence Bound: $\max_x \mu_w(x) + \alpha\sigma_w(x)$
- **Thompson sampling**: sample $\bar{w}$, $\max_x f(x; \bar{w})$

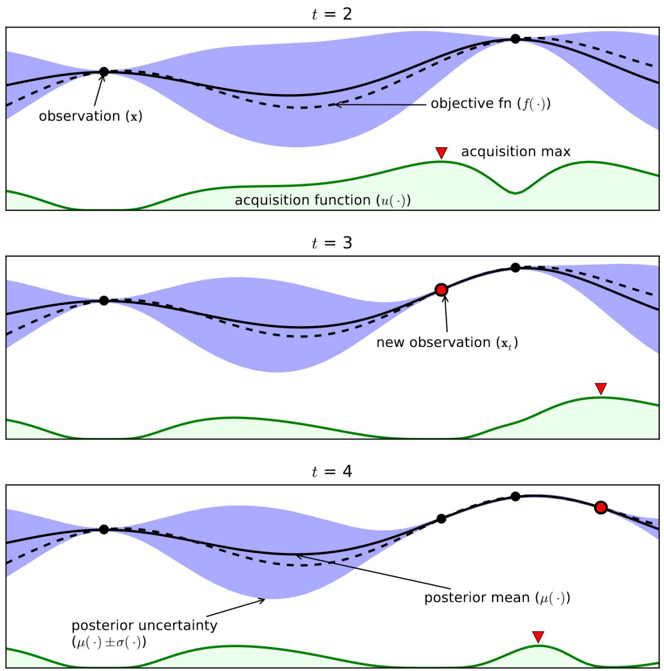


Figure: towardsdatascience.com

Bayesian optimization with neural networks

Model: $f(x) = \mathrm{NN}(x; w)$

Posterior sampling: ~~Hamilton Monte Carlo~~ ensembles are good enough

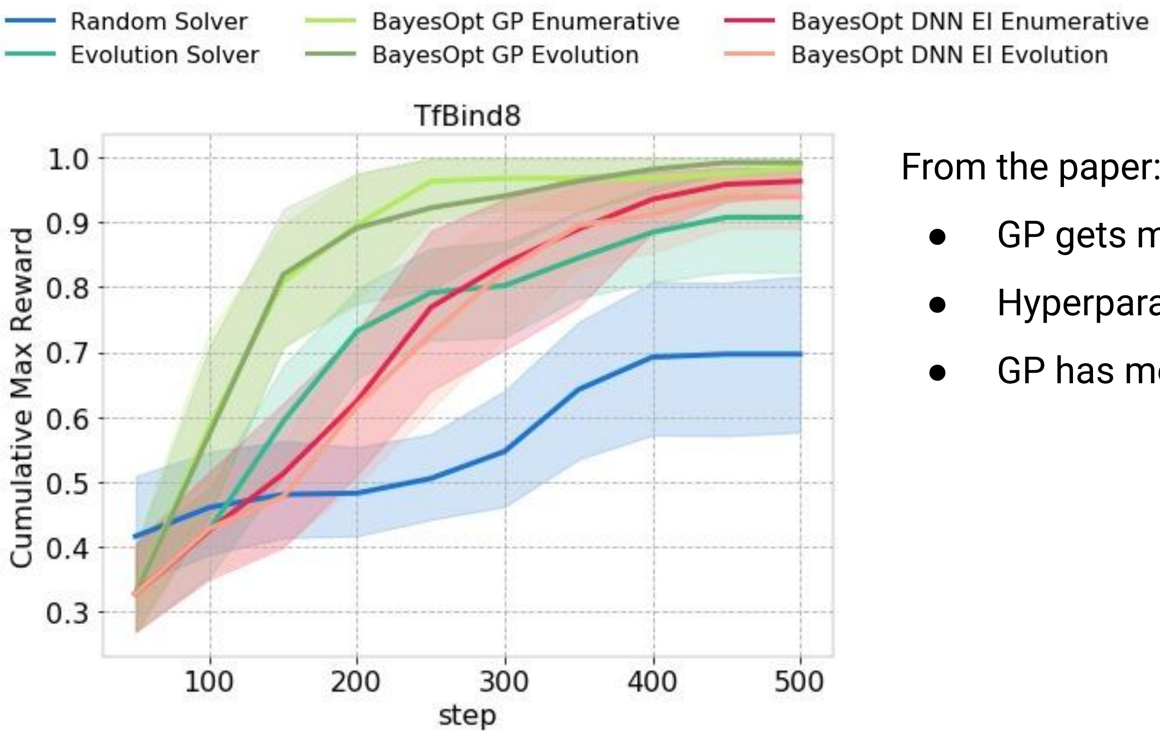
Thompson sampling:

- pick ensemble member for $\bar{w}$
- Solve $x_n = \arg\max_x \mathrm{NN}(x; \bar{w})$ (**The subject of this talk**)

Why neural networks (instead of GPs)?

- Discrete/combinatorial input
- High dimensional input
- Use special architecture to exploit problem structure
- Sampling computation
- Optimization computation
- Less hyper parameter sensitivity

DNA-Protein binding revisited



From the paper:

- GP gets more reward
- Hyperparameters are hindsight optimal
- GP has more hyperparameter sensitivity

"Biological Sequence Design using Batched Bayesian Optimization" (Belanger et al. 2019)

Neural Architecture Search and NAS-Bench-101

Black box optimization problem: minimize out of sample loss of an image classifier

Domain, DAGs with:

- ≤ 7 nodes
- ≤ 9 edges
- 3 node types (convolutions)
- All nodes connected to source and sink

NAS-Bench-101: results for all ~500k graphs

"NAS-Bench-101: Towards Reproducible Neural Architecture Search" (Ying et al. 2019)

A MIP Model for the NAS-bench-101 domain

Variables:

- Edges: $x_{ij} \in \{0,1\} \qquad \forall i < j$
- Nodes: $y_{it} \in \{0,1\} \qquad \forall 1 < i < 7, \forall t \in \{\mathrm{convA}, \mathrm{convB}, \mathrm{convC}\}$

Constraints:

$$\sum_{i=1}^{7} \sum_{j=i+1}^{7} x_{ij} \leq 9$$

(At most 9 edges)

$$\sum_{t} y_{it} = 1 \quad \forall 1 < i < 7$$

(Each node picks a convolution)

$$\sum_{i<j} x_{ij} \geq 1 \qquad \forall j > 1$$

(At most 9 edges)

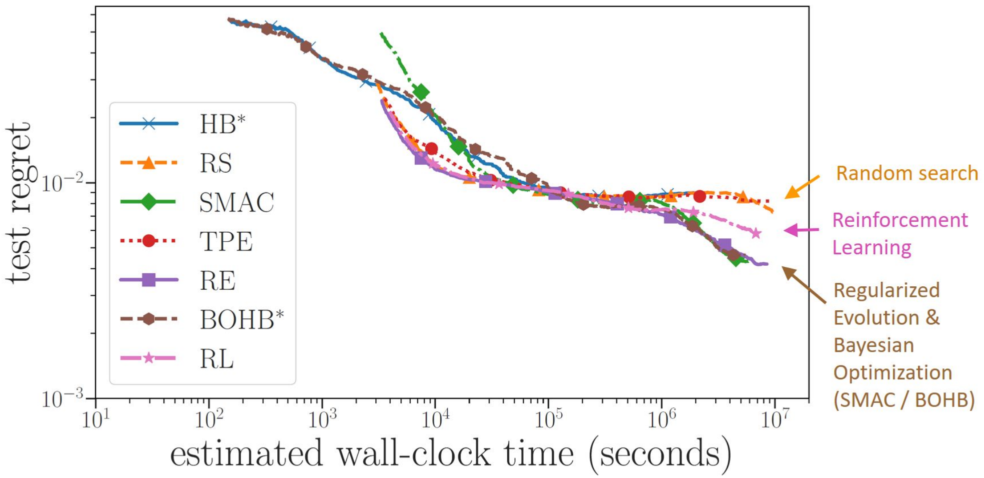
$$\sum_{j>i} x_{ij} \geq 1 \qquad \forall i < 7$$

(At most 9 edges)

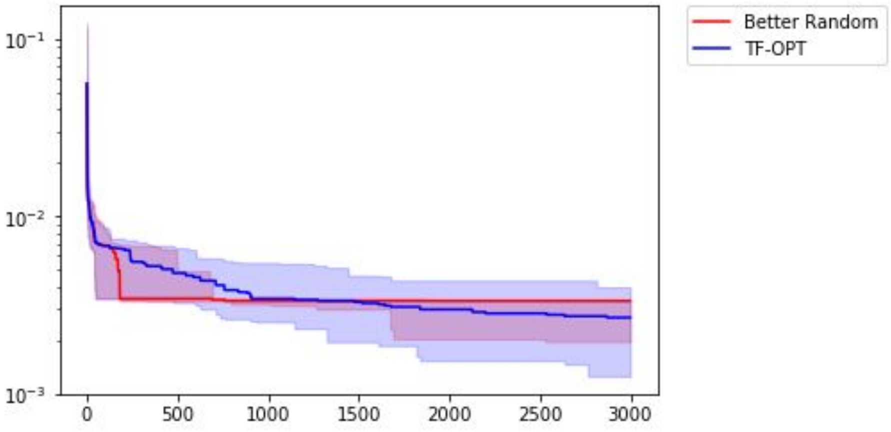
(+Some extra work for null operations)

VERY PRELIMINARY results on NAS-bench-101

From the paper (~3k evaluations)



Batched bayesian optimization with neural networks



Outline

- Applications of optimizing over trained neural networks
  - “Predict and optimize”
  - Neural network verification
  - Large action space ADP
  - Bayesian optimization
- Solution Methods
- Computational Results
- Conclusion

How to optimize over trained neural networks?

tf.opt

MIP solvers

Presolve

Heuristics

Cutting planes

and more...

+

Custom presolve

Custom heuristics

Custom cutting planes

and more in the future...

Modeling ReLU $y = \max\{0, wx + b\}$ with big-M

$$y \geq \sum_i w_i x_i + b$$

$$y \geq 0$$

Natural bounds

Need large constants:

$$M^+ \geq \max_x wx + b$$

$$M^- \leq \min_x wx + b$$

$$z \in \{0, 1\}$$

Indicator $wx + b > 0$

$$y \leq M^+ z$$

$$z = 0 \Rightarrow y = 0$$

$$y \leq \sum_i w_i x_i + b - M^-(1 - z)$$

$$z = 1 \Rightarrow y \leq wx + b$$

Presolve: Bounds tightening

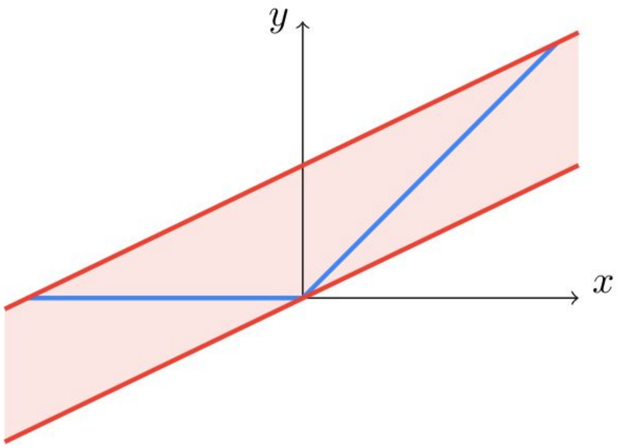
The MIP formulation is heavily based on tight bounds for the inputs of each neuron, due to the big-Ms

Generating bounds is equivalent to finding dual bounds for a smaller version of the same problem

Presolve: Bounds tightening

Solve a simple relaxation with a single pass:

one upper-bounding inequality and one lower-bounding inequality per neuron

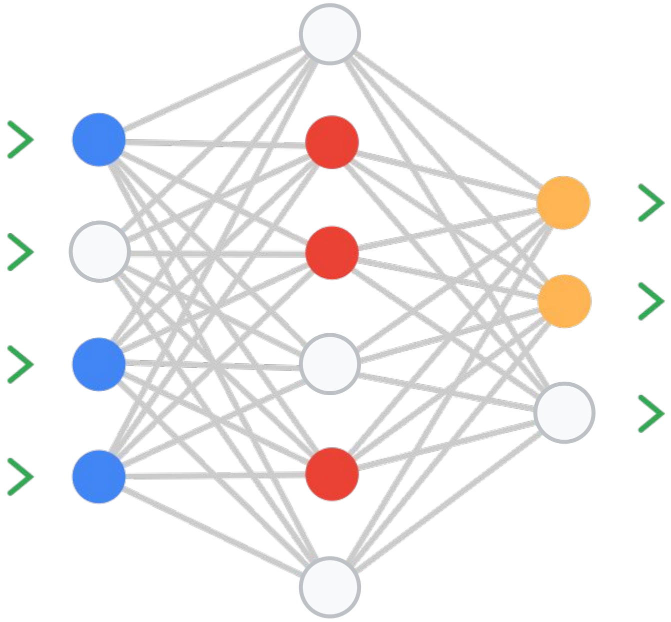


Variants appeared in Wong et al. 2017; Weng et al., 2018; Singh et al., 2018

Heuristic: Active index local search

Local search on the space of activation patterns (set of neurons that are active or not)

When the activation pattern is fixed, the problem becomes an LP



Cuts for a stronger ReLU formulation

Add to big-M formulation the exponential inequality family:

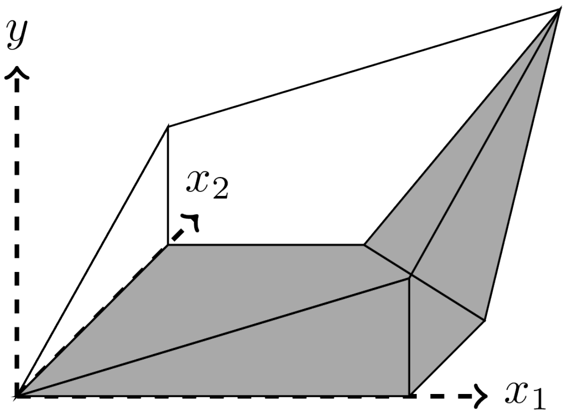
$$y \leq \sum_{i \in I} w_i x_i - M^-(I)(1 - z) + (b + M^+(I))z \qquad \forall I \subset \operatorname{supp}(w)$$

$$M^+(I) = \max_x \sum_{i \notin I} w_i x_i$$

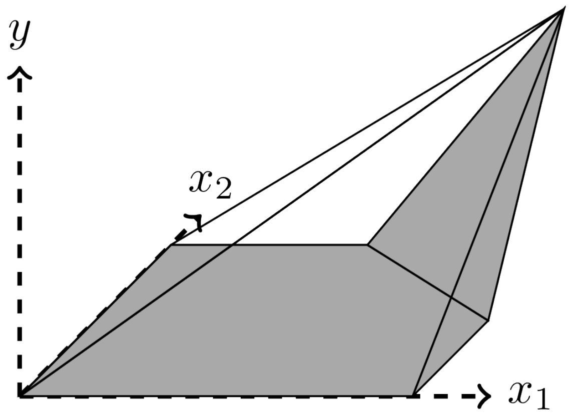
$$M^-(I) = \min_x \sum_{i \in I} w_i x_i$$

(Constants)

Without cuts:



With cuts:
Optimal!*



LP relaxation

MIP feasible

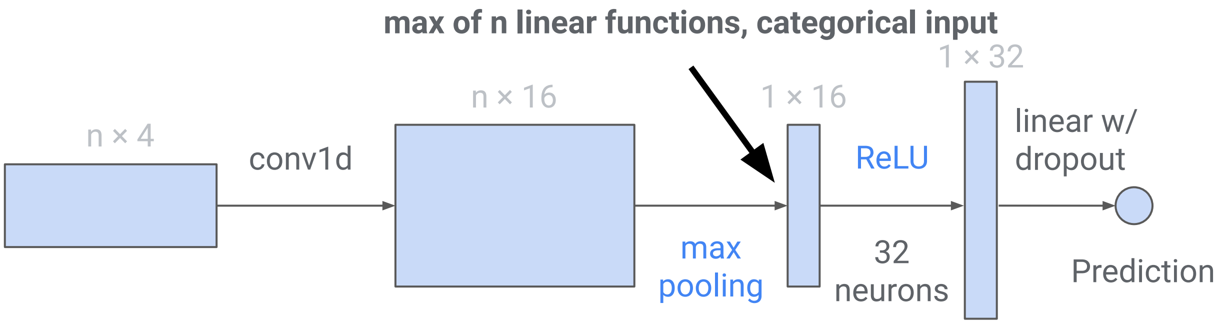
“Strong mixed-integer programming formulations for trained neural networks”

A., Huchette, Ma, Tjandraatmadja, Vielma, 2019 [To appear in Mathematical Programming]

More cutting planes

- **Nonlinearities:** max of n linear functions, clipped ReLU
- **Domains:** box constrained, categorical, Cartesian products
- **Optimality guarantees:** sharpness, hereditary sharpness, idealness

A neural network for DNA-protein binding that needs advanced cuts



Outline

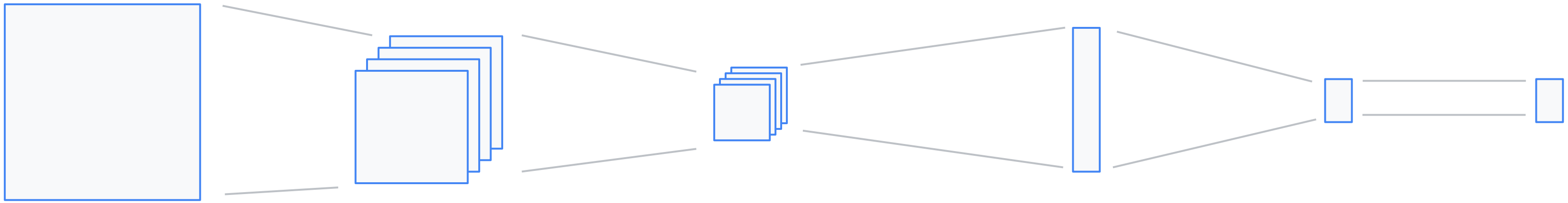
- Applications of optimizing over trained neural networks
  - "Predict and optimize"
  - Neural network verification
  - Large action space ADP
  - Bayesian optimization
- Solution Methods
- Computational Results
- Conclusion

Experiments on adversarial examples

40 images for network trained on MNIST dataset with 98.4% accuracy

with L1 regularization with weight 10^{-4}

Classifies as 7 | Perturbation | Classifies as 0



Input
28x28 (784)

→ Conv2D + ReLU
4x4 kernel with 4 filters and a stride of 2
676 ReLUs

→ Conv2D + Linear
4x4 kernel with 4 filters and a stride of 2

→ Dense ReLU
32 ReLUs

→ Linear
10 logits

→ Softmax
"7"

Experimental results

	Solving time (s)		Time limit hit
All features	205.95	-	42.0%
Without bounds tightening	455.08	2.21x slower	30.5%
Without heuristic	238.97	1.16x slower	44.5%
Without cutting planes	1059.93	5.14x slower	67.7%
Basic formulation	1081.89	5.25x slower	56.0%

Shifted geometric means (shift of 10) of 40 images × 5 solver seeds, with time limit of 30 minutes

Performance sensitivity

Solving time is

- Highly sensitive to network weights: sparsity helps a lot
- Highly variable across instances (std. dev. of 866s) and across seeds
- More sensitive to the quality of bounds the more layers it has

tf.opt

We are developing tf.opt, a software package to optimize over trained neural networks

We plan to open source tf.opt to support research in this area and help bring more Operations Research techniques to Machine Learning

Outline

- Applications of optimizing over trained neural networks
  - "Predict and optimize"
  - Neural network verification
  - Large action space ADP
  - Bayesian optimization
- Solution Methods
- Computational Results
- Conclusion

Conclusions on optimizing over trained neural networks

- Use directly (predict an optimize) or as subroutine in sequential decision making (ADP, Bayesian optimization)
- Combines with MIP for powerful modeling
- Solving is challenging, good LP is key

Interested... Google is (always) hiring!

- Internships (late for this year)
- Post-docs
- Full time
- OR is everywhere, e.g. Paris, Cambridge, NYC, and Bay Area

Email me (rander@google.com) or find me after the talk!