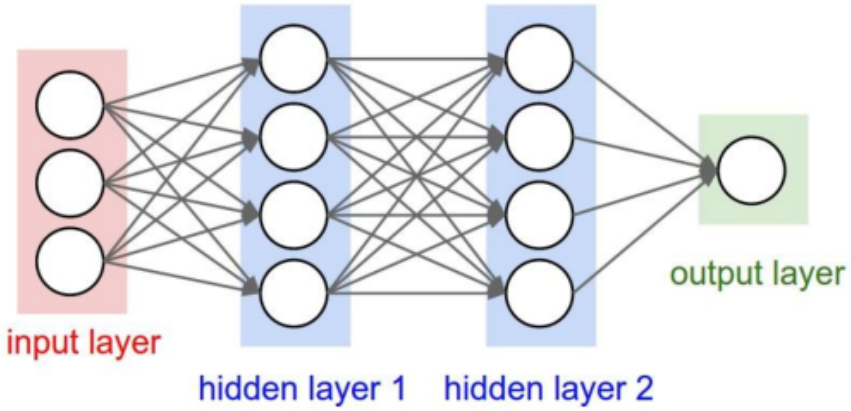
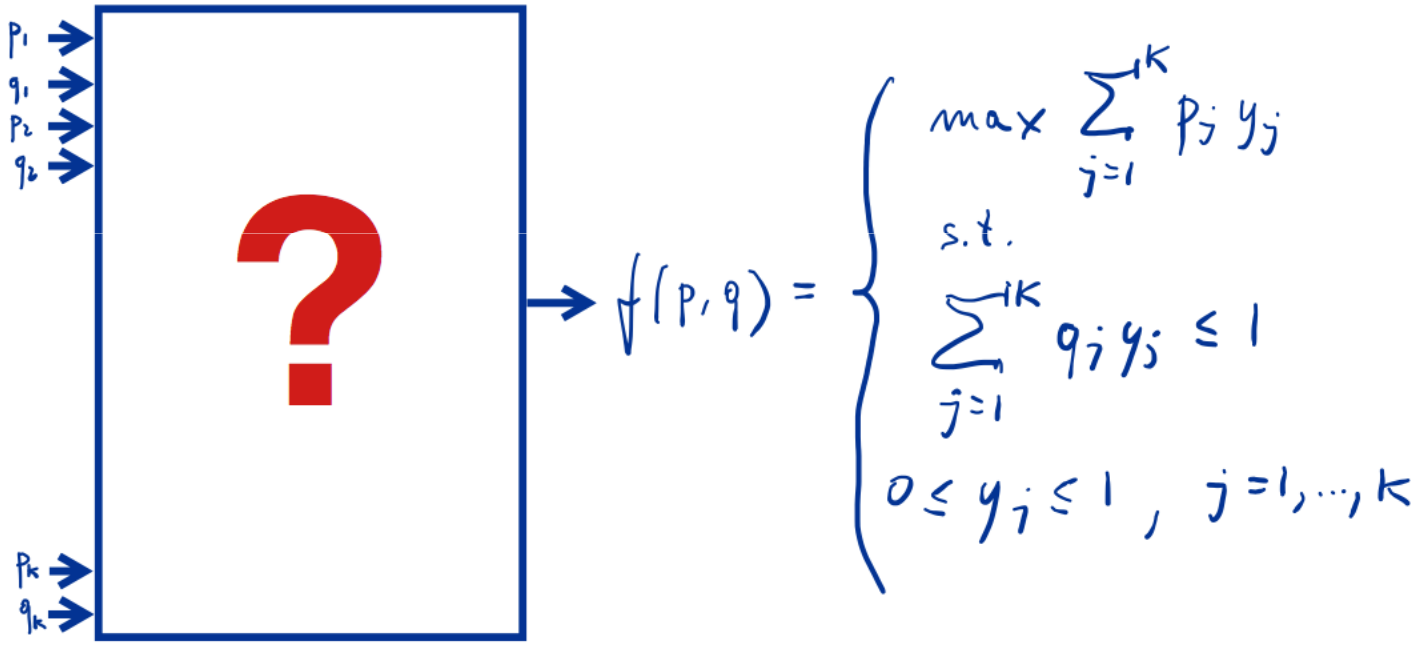
Deep Neural Networks as 0-1 Mixed Integer Linear Programs: A feasibility study

Matteo Fischetti, University of Padova
Jason Jo, Montreal Institute for Learning Algorithms (MILA)

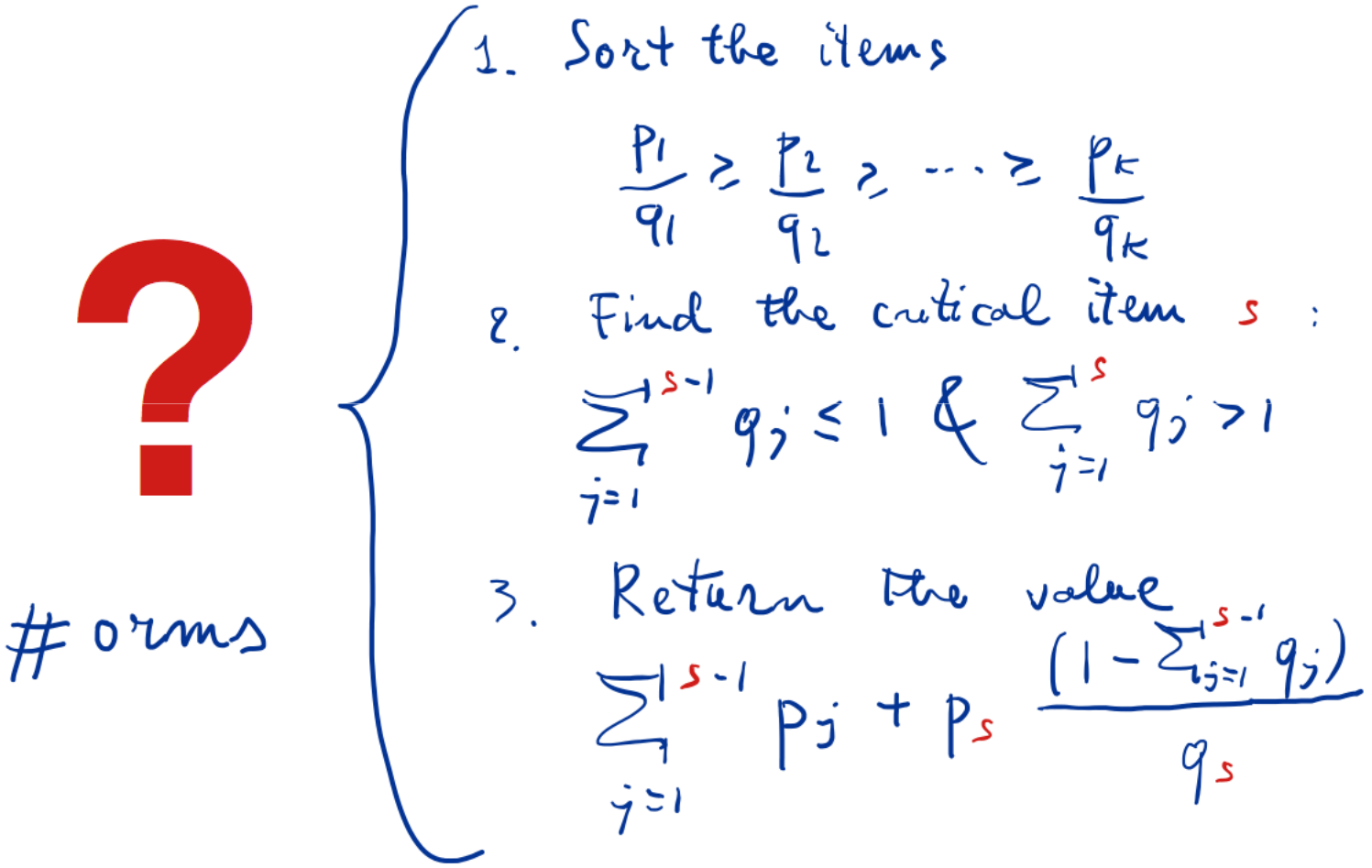


Machine Learning

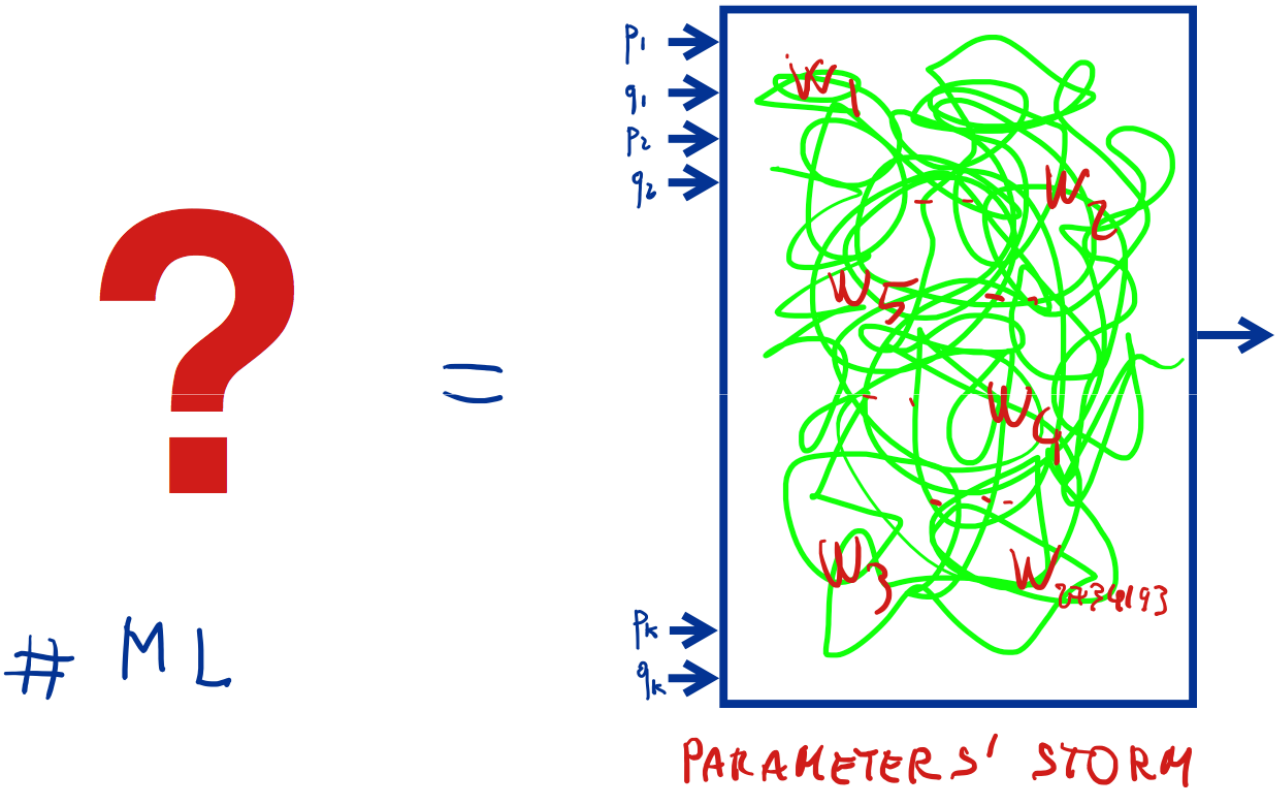
- **Example** (MIPpers only!): Continuous 0-1 Knapack Problem with a fixed n. of items



Implementing the ? in the box

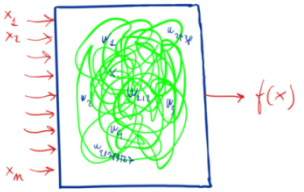


Implementing the ? in the box

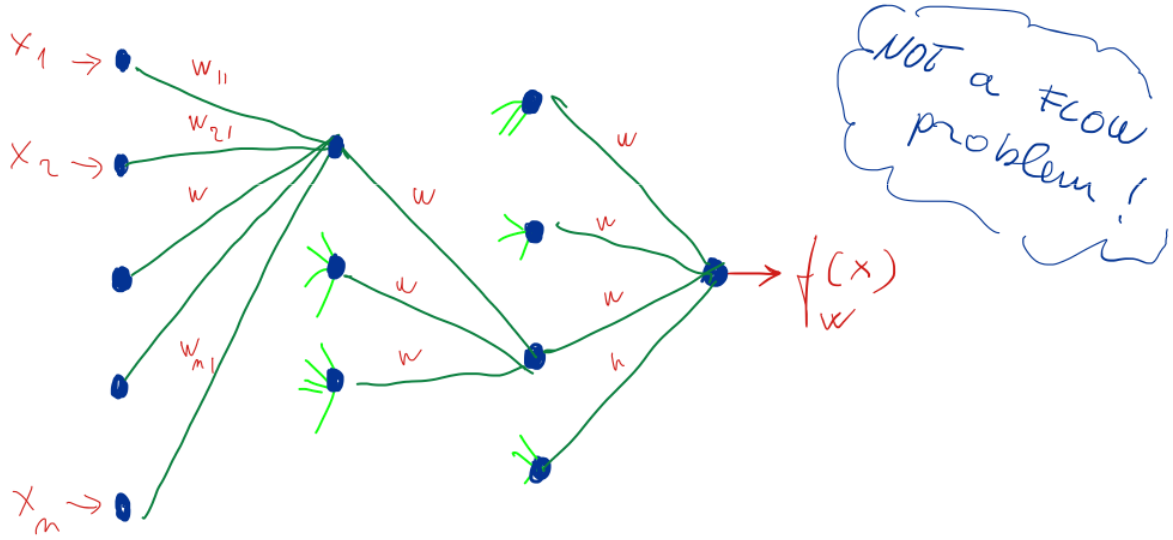


#differentiable_programming (Yann LeCun)

Deep Neural Networks (DNNs)



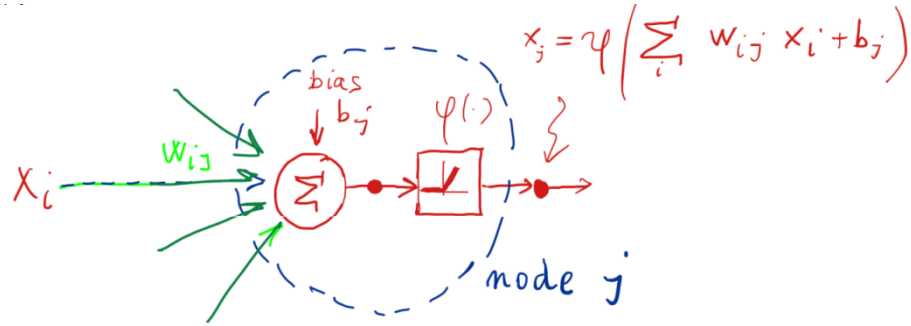
- Parameters *w*'s are organized in a layered feed-forward **network** (DAG = Directed Acyclic Graph)



- Each node (or "**neuron**") makes a **weighted sum** of the outputs of the previous layer → no "flow splitting/conservation" here!

Role of nonlinearities

- We want to be able to play with a huge n. of parameters, but if everything stays **linear** we actually have ***n+1*** parameters only → we need **nonlinearities** somewhere!
- Zooming into neurons we see the nonlinear "**activation functions**"



- Each neuron acts as a **linear SVM**, however …

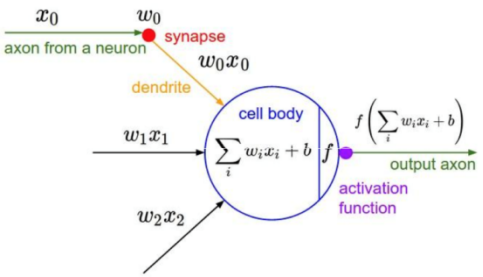
  … its output is not interpreted immediately …

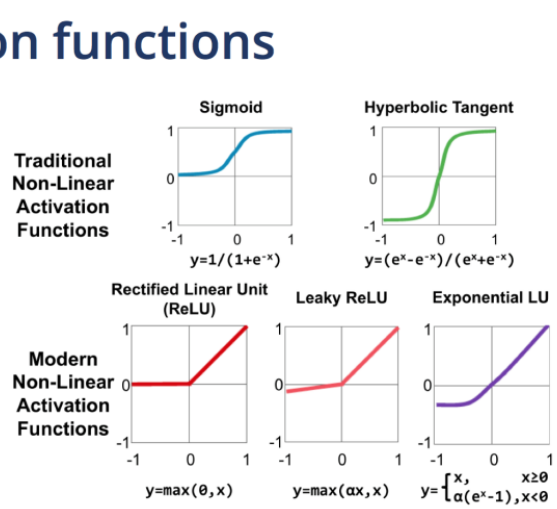
  … but it becomes a **new feature** … **#automatic_feature_detection**

  … to be forwarded to the next layer for further analysis **#SVMcascade**

Modeling a DNN with fixed param.s

- Assume all the parameters (weights/biases) of the DNN are **fixed**
- We want to model the computation that produces the output value(s) as a function of the inputs, using a MINLP #MIPpersToTheBone
- Each hidden node corresponds to a summation followed by a nonlinear activation function

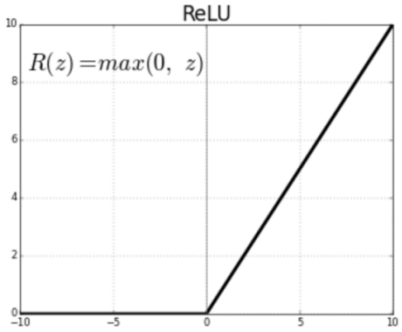




Sze et. al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey." *arXiv (2017)*

Modeling ReLU activations

- Recent work on DNNs almost invariably only use ReLU activations



$$x = ReLU(w^T y + b)$$

- Easily modeled as $w^T y + b = x - s, \quad x \geq 0, \quad s \geq 0$

  - plus the bilinear condition $x\, s \leq 0$
  - or, alternatively, the indicator constraints

$$\left.\begin{aligned} z = 1 \rightarrow x \leq 0 \\ z = 0 \rightarrow s \leq 0 \\ z \in \{0,1\} \end{aligned}\right\}$$

A complete 0-1 MILP

$$\min \sum_{k=0}^{K} \sum_{j=1}^{n_k} c_j^k x_j^k + \sum_{k=1}^{K} \sum_{j=1}^{n_k} \gamma_j^k z_j^k$$

$$\left.\begin{array}{rl} \sum_{i=1}^{n_{k-1}} w_{ij}^{k-1} x_i^{k-1} + b_j^{k-1} = x_j^k - s_j^k & \\ x_j^k, s_j^k \geq 0 & \\ z_j^k \in \{0,1\} & \\ z_j^k = 1 \rightarrow x_j^k \leq 0 & \\ z_j^k = 0 \rightarrow s_j^k \leq 0 & \end{array}\right\} k = 1, \ldots, K, \ j = 1, \ldots, n_k$$

$$lb_j^0 \leq x_j^0 \leq ub_j^0, \qquad j = 1, \ldots, n_0$$

$$\left.\begin{array}{l} lb_j^k \leq x_j^k \leq ub_j^k \\ \overline{lb}_j^k \leq s_j^k \leq \overline{ub}_j^k \end{array}\right\} k = 1, \ldots, K, \ j = 1, \ldots, n_k.$$

Adversarial problem: trick the DNN …

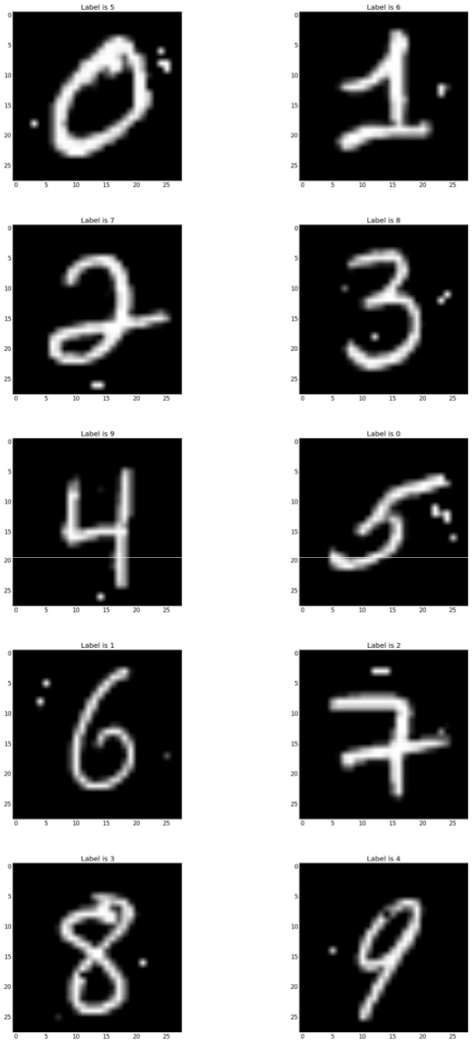


Fig. 2 Adversarial examples computed through our 0-1 MILP model; the reported label is the one having maximum activation according to the DNN (that we imposed to be the true label plus 5, modulo 10). Note that the change of just few well-chosen pixels often suffices to fool the DNN and to produce a wrong classification.

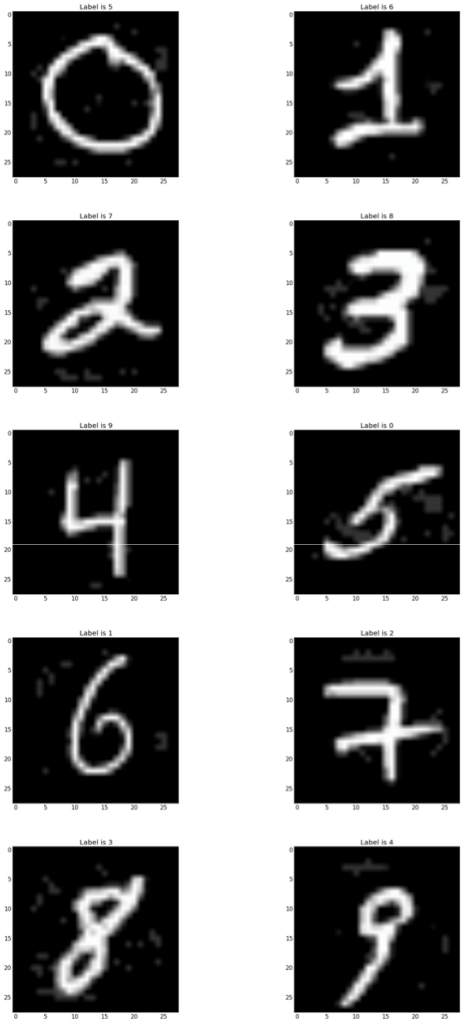


Fig. 3 Adversarial examples computed through our 0-1 MILP model as in Figure 2, but imposing that the no pixel can be changed by more than 0.2 (through the additional conditions $d_j \leq 0.2$ for all j).

… by changing few well-chosen pixels

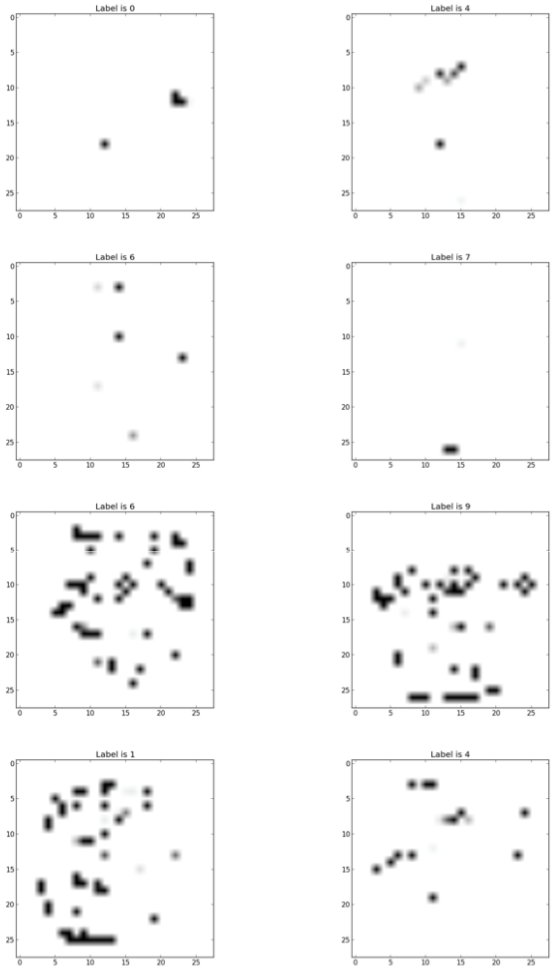
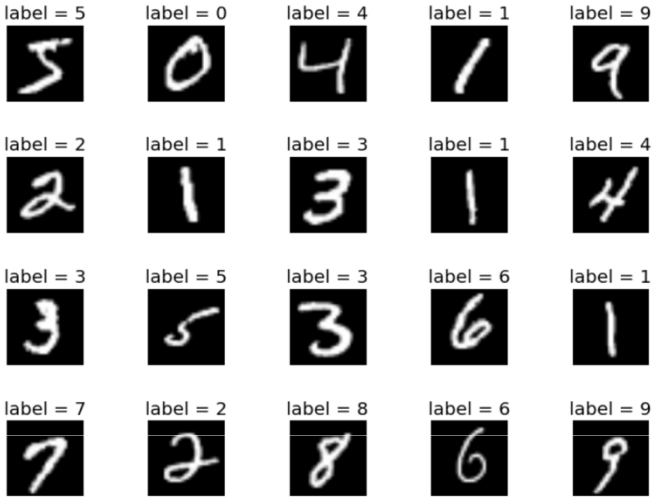


Fig. 4 Pixel changes (absolute value) that suffice to trick the DNN: the four top subfigures correspond to the model where pixels can change arbitrarily, while those on the bottom refer to the case where each pixel cannot change by more than 0.2 (hence more pixels need be changed). To improve readability, the black/white map has been reverted and scaled, i.e., white corresponds to unchanged pixels ($d_j = 0$) while black corresponds to the maximum allowed change ($d_j = 1$ for the four top figures, $d_j = 0.2$ for the four bottom ones).

Experiments on small DNNs

- The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems



- We considered the following (small) DNNs and trained each of them to get a fair accuracy (93-96%) on the test-set
  - `DNN1`: 8+8+8 internal units in 3 hidden layers, as in [13];
  - `DNN2`: 8+8+8+8+8+8 internal units in 6 hidden layers;
  - `DNN3`: 20+10+8+8 internal units in 4 hidden layers;
  - `DNN4`: 20+10+8+8+8 internal units in 5 hidden layers;
  - `DNN5`: 20+20+10+10+10 internal units in 5 hidden layers.

Computational experiments

- **Instances:** 100 MNIST training figures (each with its “true” label 0..9)
- **Goal:** Change some of the 28x28 input pixels (real values in 0-1) to convert the true label *d* into (*d + 5) mod 10* (e.g., “0” → “5”, “6” → “1”)
- **Metric:** L1 norm (sum of the abs. differences original-modified pixels)
- **MILP solver**: IBM ILOG CPLEX 12.7 (as black box)
  - **Basic model:** only obvious bounds on the continuous var.s
  - **Improved model:** apply a MILP-based preprocessing to compute tight lower/upper bounds on all the continuous variables, as in

    P. Belotti, P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gomez, and D. Salvagnin. On handling indicator constraints in mixed integer programming. Computational Optimization and Applications, (65):545–566, 2016.

Differences between the two models

	basic model				improved model			
	%solved	%gap	nodes	time (s)	%solved	%gap	nodes	time (s)
DNN1	100	0.0	1,903	1.0	100	0.0	552	0.6
DNN2	97	0.2	77,878	48.2	100	0.0	11,851	7.5
DNN3	64	11.6	228,632	158.5	100	0.0	20,309	12.1
DNN4	24	38.1	282,694	263.0	98	0.7	68,563	43.9
DNN5	7	71.8	193,725	290.9	67	11.4	76,714	171.1

Table 1 Comparison of the basic and improved models with a time limit of 300 sec.s, clearly showing the importance of bound tightening in the improved model. In this experiment, the preprocessing time needed to optimally compute the tightened bounds is not taken into account.

- `DNN1`: 8+8+8 internal units in 3 hidden layers, as in [13];
- `DNN2`: 8+8+8+8+8+8 internal units in 6 hidden layers;
- `DNN3`: 20+10+8+8 internal units in 4 hidden layers;
- `DNN4`: 20+10+8+8+8 internal units in 5 hidden layers;
- `DNN5`: 20+20+10+10+10 internal units in 5 hidden layers.

Effect of bound-tightening preproc.

	Improved model									
	Exact bounds					Weaker bounds				
	t.pre.	%sol.	%gap	nodes	time (s)	t.pre.	%sol.	%gap	nodes	time (s)
DNN4	1,112.1	98	0.7	68,563	43.9	69.4	98	0.4	80,180	45.5
DNN5	4,913.1	67	11.4	76,714	171.1	72.6	57	16.9	84,328	185.0

Table 2 Performance of the improved model with a time limit of 300 sec.s, with exact vs weaker bounds (the latter being computed with a time limit of 1 sec. for each bound computation). The overall preprocessing time (t.pre.) is greatly reduced in case of weaker bounds, without deteriorating too much the performance of the model. The difference w.r.t. the basic model in Table 1 is still striking.

- `DNN1`: 8+8+8 internal units in 3 hidden layers, as in [13];
- `DNN2`: 8+8+8+8+8+8 internal units in 6 hidden layers;
- `DNN3`: 20+10+8+8 internal units in 4 hidden layers;
- `DNN4`: 20+10+8+8+8 internal units in 5 hidden layers;
- `DNN5`: 20+20+10+10+10 internal units in 5 hidden layers.

Reaching 1% optimality

	Basic model				Improved model (weaker bounds)			
	#timlim	time (s)	nodes	%gap	#timlim	time (s)	nodes	%gap
DNN1	0	1.0	1,920	0.5	0	0.6	531	0.3
DNN2	0	47.0	76,286	0.9	0	7.5	12,110	0.8
DNN3	8	632.8	568,579	2.2	0	11.3	19,663	0.9
DNN4	36	1806.8	1,253,415	10.2	0	50.0	89,380	1.0
DNN5	81	3224.0	1,587,892	43.5	11	851.0	163,135	3.8

Table 3 Performance of the basic and improved model (the latter with the 1-sec. weaker bounds as in Table 2) to get solutions with guaranteed error of 1% or less; each run had a time limit of 3,600 sec.s; the number of time limits, out of 100, is reported in column #timlim.

- `DNN1`: 8+8+8 internal units in 3 hidden layers, as in [13];
- `DNN2`: 8+8+8+8+8+8 internal units in 6 hidden layers;
- `DNN3`: 20+10+8+8 internal units in 4 hidden layers;
- `DNN4`: 20+10+8+8+8 internal units in 5 hidden layers;
- `DNN5`: 20+20+10+10+10 internal units in 5 hidden layers.

Thanks for your attention!

Slides available at http://www.dei.unipd.it/~fisch/papers/slides/

Paper:

M. Fischetti, J. Jo, "Deep Neural Networks as 0-1 Mixed Integer Linear Programs: A Feasibility Study", Constraints 23(3), 296-309, 2018.

.