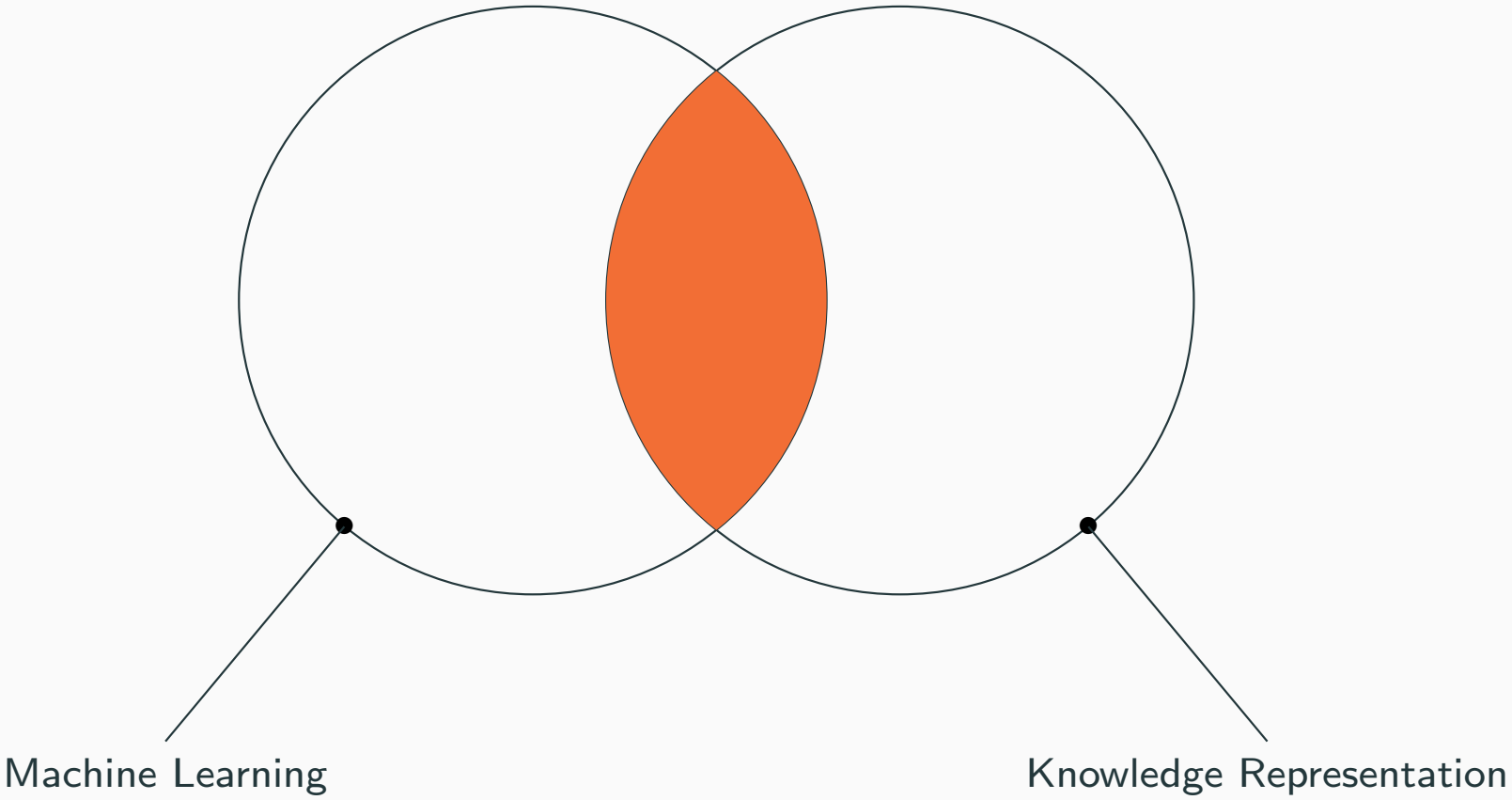
Learning to Reason: From Theory to Practice

Frédéric Koriche (CRIL, CNRS)
koriche@cril.fr

Working Group “Apprentissage & Raisonnement”, Nov 14-15th 2018

Outline

Outline



Learning to Reason (L2R) is a research topic that lies at the intersection of Machine Learning and Knowledge Representation. The key challenge is to develop algorithms capable of performing inference tasks from experience.

When do we need machine learning for reasoning tasks?

When do we need machine learning for reasoning tasks?

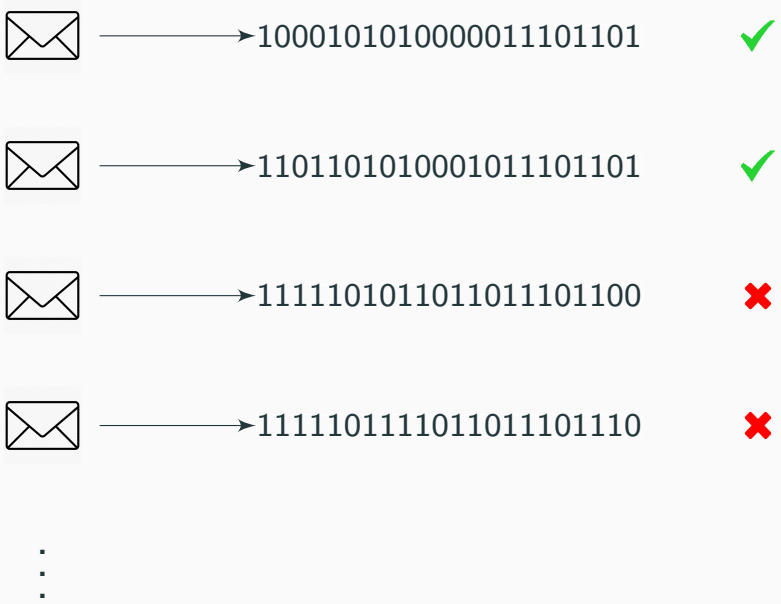
- **Cognitive Barrier:** There are many tasks that humans perform routinely (ex: reasoning about images, videos, texts), yet our knowledge about "how" we do them is not sufficiently elaborate to design well-defined inference models.

When do we need machine learning for reasoning tasks?

- **Cognitive Barrier:** There are many tasks that humans perform routinely (ex: reasoning about images, videos, texts), yet our knowledge about "how" we do them is not sufficiently elaborate to design well-defined inference models.
- **Computational Barrier:** Many, if not most, reasoning tasks are related to NP-hard problems. Learning useful patterns from observed instances can be relevant for solving - or helping a solver in handling - new instances.

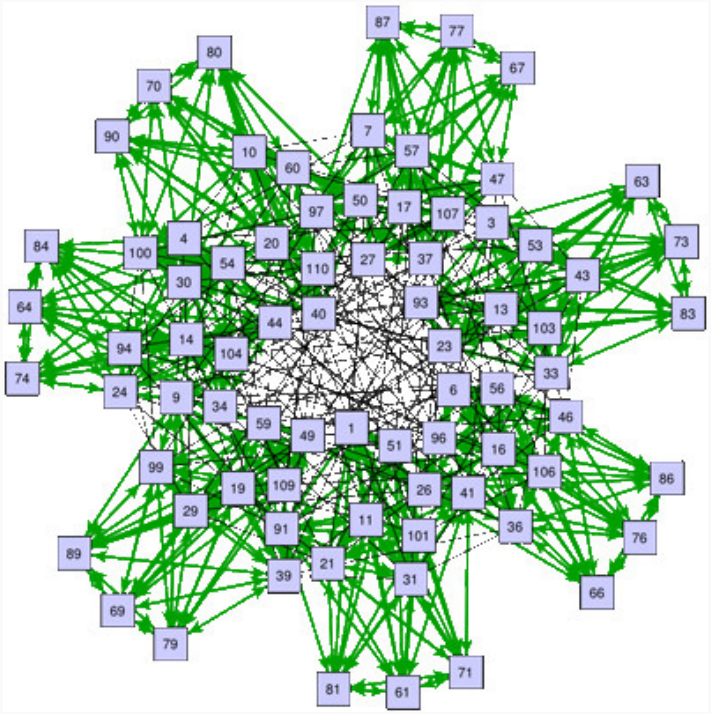
When do we need machine learning for reasoning tasks?

- **Cognitive Barrier:** There are many tasks that humans perform routinely (ex: reasoning about images, videos, texts), yet our knowledge about "how" we do them is not sufficiently elaborate to design well-defined inference models.
- **Computational Barrier:** Many, if not most, reasoning tasks are related to NP-hard problems. Learning useful patterns from observed instances can be relevant for solving - or helping a solver in handling - new instances.
- **Adaptivity:** In many tasks, the distribution of instances is susceptible to change over time (ex: visual scenes). Online learning algorithms are, by nature, adaptative to those changes.



Consider a spam filtering problem: the goal is to predict whether each incoming message, characterized by its feature vector x, is a spam (✓) or not (✖).

This is a standard classification task.



01**1100011***** ✓

*1*011*1100011**110000 ✓

000100*1000011**1*1001 ✗

111111*1100001***1010* ✗

1*00*1*1*00000**110000 ✓

100010*0*00000**101101 ✓

⋮

Suppose we have a (fixed) constraint network representing our knowledge about some application (scheduling, planning, etc.). The goal is to predict whether each incoming *partial assignment* x can be extended (✓) to a *total satisfying assignment*, or not (✗).

This is a much more difficult global consistency task.

$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$ ✖

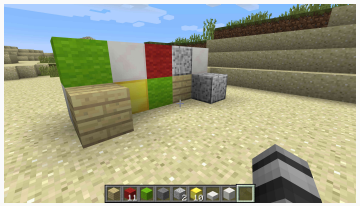
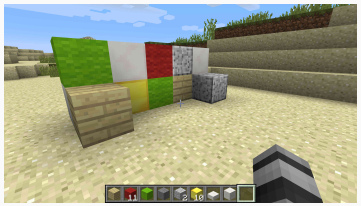
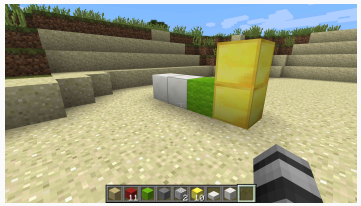
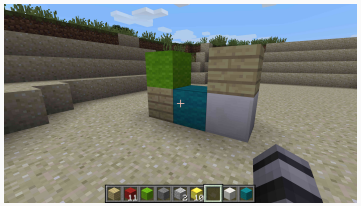
$(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$ ✔

$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_4 \vee \overline{x}_5) \wedge (\overline{x}_1 \vee \overline{x}_4 \vee x_5) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5)$ ✖

$\vdots$

Suppose we have a generator of SAT instances, like **SR**(n), where n is the number of boolean variables. The goal is to predict whether each generated CNF formula is SAT or UNSTAT.

This again a difficult satisfiability task.

. . .

⟨ red,above-of,green ⟩ ⟨ green,below-of,red ⟩ ⟨ green,below-of,red ⟩ ⟨ green,above-of,blue ⟩

✓ ✓ ✗ ✗

Suppose we are observing images (ex: Minecraft scenes), each labeled by a yes/no question involving a relation between objects. The goal is to predict the answer of each incoming visual question.

This is once again a difficult task called Visual Question Answering.

Outline

Machine learning problems are specified by:

- a task we wish to solve, characterized by its instance space $\mathcal{Z}$;
- a hypothesis class (prediction models) $\mathcal{H}$ for solving this task;
- a loss function $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}$ for measuring the performance of a model at solving the task.

Machine learning problems are specified by:

- a task we wish to solve, characterized by its instance space $\mathcal{Z}$;
- a hypothesis class (prediction models) $\mathcal{H}$ for solving this task;
- a loss function $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}$ for measuring the performance of a model at solving the task.

For statistical learning problems, it is further assumed that task instances are i.i.d. according to a fixed, but *hidden*, probability distribution $\mathcal{D}$. The (true) risk of any hypothesis $h \in \mathcal{H}$ is given by

$$L_{\mathcal{D}}(h) = \mathbb{E}_{\boldsymbol{z} \sim \mathcal{D}}[\ell(h, \boldsymbol{z})]$$

Machine learning problems are specified by:

- a task we wish to solve, characterized by its instance space $\mathcal{Z}$;
- a hypothesis class (prediction models) $\mathcal{H}$ for solving this task;
- a loss function $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}$ for measuring the performance of a model at solving the task.

For statistical learning problems, it is further assumed that task instances are i.i.d. according to a fixed, but *hidden*, probability distribution $\mathcal{D}$. The (true) risk of any hypothesis $h \in \mathcal{H}$ is given by

$$L_{\mathcal{D}}(h) = \mathbb{E}_{\boldsymbol{z} \sim \mathcal{D}}[\ell(h, \boldsymbol{z})]$$

The learning algorithm cannot have access to $\mathcal{D}$, but it has access to a *training sample*, that is, a sequence of instances S drawn according to $\mathcal{D}$. So, the main question is

How can we find a model $h \in \mathcal{H}$ with small risk using only S?

Let us start with a common problem: binary classification

Let us start with a common problem: binary classification

What are the instances?

Let us start with a common problem: binary classification

What are the instances? Typically labeled objects of the form $\boldsymbol{z} = (\boldsymbol{x}, y)$, where $\boldsymbol{x} \in \mathbb{R}^n$, and $y \in \{-1, +1\}$ (or $y \in \{0, 1\}$).

What is the hypothesis class?

Let us start with a common problem: binary classification

What are the instances? Typically labeled objects of the form $\boldsymbol{z} = (\boldsymbol{x}, y)$, where $\boldsymbol{x} \in \mathbb{R}^n$, and $y \in \{-1, +1\}$ (or $y \in \{0, 1\}$).

What is the hypothesis class? A basic choice is the class of linear zero-threshold functions

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^\top \boldsymbol{x})$$

So, $\mathcal{H}$ is defined by a (typically closed convex) set $\mathcal{W} \subseteq \mathbb{R}^n$.

What is the loss function?

Let us start with a common problem: binary classification

What are the instances? Typically labeled objects of the form $\boldsymbol{z} = (\boldsymbol{x}, y)$, where $\boldsymbol{x} \in \mathbb{R}^n$, and $y \in \{-1, +1\}$ (or $y \in \{0, 1\}$).

What is the hypothesis class? A basic choice is the class of linear zero-threshold functions

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^\top \boldsymbol{x})$$

So, $\mathcal{H}$ is defined by a (typically closed convex) set $\mathcal{W} \subseteq \mathbb{R}^n$.

What is the loss function? A natural choice is the zero-one loss (i.e. error) function:

$$\ell^{\mathrm{ERR}}(h_{\boldsymbol{w}}, (\boldsymbol{x}, y)) = \begin{cases} 1 & \text{if } h_{\boldsymbol{w}}(\boldsymbol{x}) \neq y \\ 0 & \text{otherwise.} \end{cases}$$

But the separation problem is here NP-hard! So, an alternative choice is a convex loss function, such as the hinge loss:

$$\ell^{\mathrm{HINGE}}(h_{\boldsymbol{w}}, (\boldsymbol{x}, y)) = \max\{0, 1 - y(\boldsymbol{w}^\top \boldsymbol{x})\}$$

Let us turn to the global consistency task, which is to determine whether a partial assignment can be extended, or not, to a total satisfying assignment for a predefined constraint network.

Let us turn to the global consistency task, which is to determine whether a partial assignment can be extended, or not, to a total satisfying assignment for a predefined constraint network.

What are the instances?

Let us turn to the global consistency task, which is to determine whether a partial assignment can be extended, or not, to a total satisfying assignment for a predefined constraint network.

What are the instances? For the sake of simplicity, we can assume that all variables are boolean. So, $\boldsymbol{z} = (\boldsymbol{x}, y)$ where $\boldsymbol{x} \in \{0, *, 1\}^n$, and $y \in \{-1, 1\}$.

What is the hypothesis class?

Let us turn to the global consistency task, which is to determine whether a partial assignment can be extended, or not, to a total satisfying assignment for a predefined constraint network.

What are the instances? For the sake of simplicity, we can assume that all variables are boolean. So, $\boldsymbol{z} = (\boldsymbol{x}, y)$ where $\boldsymbol{x} \in \{0, *, 1\}^n$, and $y \in \{-1, 1\}$.

What is the hypothesis class? The problem is generally not linearly separable. So, we can start with polynomial zero-threshold functions of degree k:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}))$$

where $\boldsymbol{w} \in \mathcal{W} \subseteq \mathbb{R}^d$, and $\boldsymbol{\phi} : \{0, *, 1\}^n \to \{0, 1\}^d$ is a *feature expansion operator*. Namely, $\boldsymbol{\phi}$ can be described by a set $\{f_1, \cdots, f_d\}$ of monomials with degree at most k, and $\phi_i(\boldsymbol{x}) = 1$ iff $\boldsymbol{x}$ can be extended to a total assignment satisfying f_i.

What is the loss function?

Let us turn to the global consistency task, which is to determine whether a partial assignment can be extended, or not, to a total satisfying assignment for a predefined constraint network.

What are the instances? For the sake of simplicity, we can assume that all variables are boolean. So, $\boldsymbol{z} = (\boldsymbol{x}, y)$ where $\boldsymbol{x} \in \{0, *, 1\}^n$, and $y \in \{-1, 1\}$.

What is the hypothesis class? The problem is generally not linearly separable. So, we can start with polynomial zero-threshold functions of degree k:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}))$$

where $\boldsymbol{w} \in \mathcal{W} \subseteq \mathbb{R}^d$, and $\boldsymbol{\phi} : \{0, *, 1\}^n \to \{0, 1\}^d$ is a *feature expansion operator*. Namely, $\boldsymbol{\phi}$ can be described by a set $\{f_1, \cdots, f_d\}$ of monomials with degree at most k, and $\phi_i(\boldsymbol{x}) = 1$ iff $\boldsymbol{x}$ can be extended to a total assignment satisfying f_i.

What is the loss function? Again, a natural choice is the zero-one loss. But the separation problem is also NP-hard here. So, convex loss functions might be more appropriate.

Now, let us go for the satisfiability task, which is to determine whether a random CNF instance is SAT, or UNSAT.

Now, let us go for the satisfiability task, which is to determine whether a random CNF instance is SAT, or UNSAT.

What are the instances?

Now, let us go for the satisfiability task, which is to determine whether a random CNF instance is SAT, or UNSAT.

What are the instances? For k-CNF formulas over n variables, $\boldsymbol{z} = (\boldsymbol{X}, y)$ where $\boldsymbol{X} \in \{0,1\}^{2n \times m}$ is the adjacency matrix of the incidence graph of the formula, where $m = O(n^k)$, and $y \in \{0,1\}$.

What is the hypothesis class?

Now, let us go for the satisfiability task, which is to determine whether a random CNF instance is SAT, or UNSAT.

What are the instances? For k-CNF formulas over n variables, $\boldsymbol{z} = (\boldsymbol{X}, y)$ where $\boldsymbol{X} \in \{0,1\}^{2n \times m}$ is the adjacency matrix of the incidence graph of the formula, where $m = O(n^k)$, and $y \in \{0,1\}$.

What is the hypothesis class? Obvisouly, the problem is not linearly separable. For matrix inputs, we can think about the classes of deep Markov nets, or deep neural networks.

What is the loss function?

Now, let us go for the satisfiability task, which is to determine whether a random CNF instance is SAT, or UNSAT.

What are the instances? For k-CNF formulas over n variables, $\boldsymbol{z} = (\boldsymbol{X}, y)$ where $\boldsymbol{X} \in \{0,1\}^{2n \times m}$ is the adjacency matrix of the incidence graph of the formula, where $m = O(n^k)$, and $y \in \{0,1\}$.

What is the hypothesis class? Obvisouly, the problem is not linearly separable. For matrix inputs, we can think about the classes of deep Markov nets, or deep neural networks.

What is the loss function? For deep learning models, common loss functions for binary labels are the convex and differentiable logistic loss, and sigmoid cross-entropy loss.

A definition of Learnability:

- $\mathcal{S}$ is the set of all finite training sets, i.e. $\mathcal{S} = \bigcup_{m \in \mathbb{N}} \mathcal{Z}^m$.
- A learning algorithm is a map $A : \mathcal{S} \to \mathcal{H}$ that takes as input a training set S and returns as output a prediction model $A(S)$.
- A learning rate is a monotone decreasing map $\epsilon : \mathbb{N} \to \mathbb{R}$ that converges to 0 as m tends to infinite.

A definition of Learnability:

- $\mathcal{S}$ is the set of all finite training sets, i.e. $\mathcal{S} = \bigcup_{m \in \mathbb{N}} \mathcal{Z}^m$.
- A learning algorithm is a map $A : \mathcal{S} \to \mathcal{H}$ that takes as input a training set S and returns as output a prediction model $A(S)$.
- A learning rate is a monotone decreasing map $\epsilon : \mathbb{N} \to \mathbb{R}$ that converges to 0 as m tends to infinite.

Let $\mathcal{Z}$ be an instance space, $\mathcal{H}$ be a hypothesis class, and $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}$ be a loss function. Then, $\mathcal{H}$ is learnable with respect to ℓ if there is a learning algorithm A and a rate ϵ such that, for any distribution $\mathcal{D}$ over $\mathcal{Z}$ and any integer $m \in \mathbb{N}$:

$$\mathbb{E}_{S \in \mathcal{Z}^m}[L_{\mathcal{D}}(A(S))] - \inf_{h \in \mathcal{H}} L_{\mathcal{D}}(h) \leq \epsilon(m) \tag{1}$$

Any algorithm A satisfying (1) is called statistically efficient (aka statistically consistent).

This definition is quite standard in statistical computational learning. Notably it generalizes the concept of (agnostic) PAC learnability.

But is it relevant for reasoning tasks?

This definition is quite standard in statistical computational learning. Notably it generalizes the concept of (agnostic) PAC learnability.

But is it relevant for reasoning tasks?

Bias-Variance Dilemma. The main inequality:

$$\mathbb{E}_{S \in \mathcal{Z}^m}[L_{\mathcal{D}}(A(S))] - \inf_{h \in \mathcal{H}} L_{\mathcal{D}}(h) \leq \epsilon(m)$$

indicates that the risk of our predictive model $A(S)$ must converge to the risk of the best possible hypothesis in $\mathcal{H}$. So,

- if $\mathcal{H}$ is small (simple models), the best hypothesis may perform poorly (large bias);
- if $\mathcal{H}$ is large (complex models), the convergence may be slow (large "variance").

Thus, choosing the right hypothesis class is critical for any task.

Accuracy-Complexity Dilemma. Recall that:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{z \sim \mathcal{D}}[\ell(h, z)]$$

is measured with respect to a loss function ℓ which, for computational reasons, is often a convex surrogate of the true objective function (e.g. zero-one loss).

Thus, choosing the right loss function is also critical for reasoning tasks.

Outline

Consider the global consistency task, in which we are given a constraint network, and the training set consists of partial assignments (the queries), labeled by ✓ or ✖ depending on whether they are globally consistent, or not.

For each partial assignment $\boldsymbol{x} \in \{0, *, 1\}^n$, we use $\textsc{ext}(\boldsymbol{x})$ to denote the set of all its total extensions in $\{0, 1\}^n$.

A Sampling Method.

1. Let T be the set of *total* assignments *labeled by* 1 in S.
2. For each incoming instance $\boldsymbol{x}$, predict ✓ if $\textsc{ext}(\boldsymbol{x}) \cap T \neq \varnothing$, and ✖ otherwise.

Now, consider the following assumption:

> Let W be the set of satisfying assignments of the constraint network, and $\mathcal{D}$ be a (hidden) probability distribution over $\{0, *, 1\}^n$. A partial assignment $\boldsymbol{x}$ is called γ-fair if either $\boldsymbol{x}$ is *not* globally consistent, or $\mathbb{P}_{\mathcal{D}}[W \cap \textsc{ext}(\boldsymbol{x}) \neq \varnothing] > \gamma$.

Theorem (Khardon & Roth, 1997)

Under the assumption that any partial assignment $\boldsymbol{x}$ drawn by $\mathcal{D}$ is γ-fair, the sampling algorithm is consistent.

Theorem (Khardon & Roth, 1997)

Under the assumption that any partial assignment $\boldsymbol{x}$ drawn by $\mathcal{D}$ is γ-fair, the sampling algorithm is consistent.

Proof

1. Clearly, if $\boldsymbol{x}$ is not GC, the algorithm will answer ✖.

Theorem (Khardon & Roth, 1997)

Under the assumption that any partial assignment $\boldsymbol{x}$ drawn by $\mathcal{D}$ is γ-fair, the sampling algorithm is consistent.

Proof

1. Clearly, if $\boldsymbol{x}$ is not GC, the algorithm will answer ✖.
2. The algorithm makes a mistake only if $\boldsymbol{x}$ is GC, but a total assignment in $W \cap \mathrm{EXT}(\boldsymbol{x})$ is not sampled.
3. However, by γ-fairness such an assignment is drawn with probability greater than γ.
4. So, after m trials, the expected error on this instance is bounded by:
$$\mathbb{E}_{S \in \mathcal{Z}^m}[\ell^{\mathrm{ERR}}(T; (\boldsymbol{x}, 1))] \leq (1-\gamma)^m$$
5. Now, let $\mathcal{X} = \{0, *, 1\}^n$. By extending the previous inequality to all instances in $\mathcal{X}$,
$$\mathbb{E}_{S \in \mathcal{Z}^m}[L_{\mathcal{D}}(T)] \leq |\mathcal{X}|(1-\gamma)^m$$
6. Therefore, by taking more than
$$m = \frac{1}{\gamma}\left(\ln|\mathcal{X}| + \ln\frac{1}{\delta}\right)$$
samples, and using the fact that $\ln(1-\gamma) \leq -\gamma$, we get that
$$\mathbb{E}_{S \in \mathcal{Z}^m}[L_{\mathcal{D}}(T)] \leq \delta$$
Finally, since $|\mathcal{X}| = 3^n$, the algorithm is statistically consistent with sample complexity:
$$m = \frac{1}{\gamma}\left(n\ln 3 + \ln\frac{1}{\delta}\right)$$

We might have cheated a little with the notion of γ-fairness :-)

We might have cheated a little with the notion of γ-fairness :-)

Let us try a convex optimization method, using the class of polynomial functions. Let $\{f_1, \cdots, f_d\}$ be the set of all monomials of degree at most k over n variables, where

$$d = \sum_{j=1}^{k} \binom{j}{n}$$

Stochastic Gradient Descent. Initialize θ_1 to $\mathbf{0}$.

For $t = 1$ to T

1. Let $\boldsymbol{w}_t = \frac{1}{\lambda t}\theta_t$
2. Choose the tth sample $(\boldsymbol{x}_t, y_t)$ uniformly over the training set S
3. if $y_t(\boldsymbol{w}_t^\top \phi(\boldsymbol{x}_t) < 1)$ set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + y_t\phi(\boldsymbol{x}_t)$
4. otherwise set $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t$

Output $\boldsymbol{w} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}_t$

Theorem (Shalev-Shwartz & Ben David, 2014)

Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{0, 1\}$, where $\|\phi(\boldsymbol{x})\| \leq \rho\}$. Consider running the SGD algorithm on a training set $S \sim \mathcal{D}^m$, and let $\boldsymbol{w}$ be the solution returned by SGD. Then for every $\boldsymbol{u}$ such that $\|\boldsymbol{u}\| \leq B$, we have

$$\mathbb{E}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{w})] \leq L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) + 2\rho B\sqrt{2/m}$$

Theorem (Shalev-Shwartz & Ben David, 2014)

Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{0, 1\}$, where $\|\phi(\boldsymbol{x})\| \leq \rho\}$. Consider running the SGD algorithm on a training set $S \sim \mathcal{D}^m$, and let $\boldsymbol{w}$ be the solution returned by SGD. Then for every $\boldsymbol{u}$ such that $\|\boldsymbol{u}\| \leq B$, we have

$$\mathbb{E}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{w})] \leq L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) + 2\rho B\sqrt{2/m}$$

So, for the global consistency task, if the given constraint network is representable by a polynomial function of degree k, in which each weight in $\boldsymbol{u}$ statisfies $u_i \in [-c, +c]$, we have:

$$\mathbb{E}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{w})] \leq L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) + 2dc\sqrt{2/m}$$

Theorem (Shalev-Shwartz & Ben David, 2014)

Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{0, 1\}$, where $\|\phi(\boldsymbol{x})\| \leq \rho\}$. Consider running the SGD algorithm on a training set $S \sim \mathcal{D}^m$, and let $\boldsymbol{w}$ be the solution returned by SGD. Then for every $\boldsymbol{u}$ such that $\|\boldsymbol{u}\| \leq B$, we have

$$\mathbb{E}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{w})] \leq L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) + 2\rho B\sqrt{2/m}$$

So, for the global consistency task, if the given constraint network is representable by a polynomial function of degree k, in which each weight in $\boldsymbol{u}$ statisfies $u_i \in [-c, +c]$, we have:

$$\mathbb{E}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{w})] \leq L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) + 2dc\sqrt{2/m}$$

Are we cheating here?

Theorem (Shalev-Shwartz & Ben David, 2014)

Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{0, 1\}$, where $\|\phi(\boldsymbol{x})\| \leq \rho\}$. Consider running the SGD algorithm on a training set $S \sim \mathcal{D}^m$, and let $\boldsymbol{w}$ be the solution returned by SGD. Then for every $\boldsymbol{u}$ such that $\|\boldsymbol{u}\| \leq B$, we have

$$\mathbb{E}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{w})] \leq L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) + 2\rho B\sqrt{2/m}$$

So, for the global consistency task, if the given constraint network is representable by a polynomial function of degree k, in which each weight in $\boldsymbol{u}$ statisfies $u_i \in [-c, +c]$, we have:

$$\mathbb{E}_{S \sim \mathcal{D}^m}[L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{w})] \leq L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) + 2dc\sqrt{2/m}$$

Are we cheating here? Maybe so, because:

$$L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u}) \geq L_{\mathcal{D}}^{\text{ERR}}(\boldsymbol{u}) = 0$$

In fact, we don't know how to bound $L_{\mathcal{D}}^{\text{HINGE}}(\boldsymbol{u})$ without additional assumptions!

The convex optimization approach its extensions are yet interesting from a computational viewpoint.

The convex optimization approach its extensions are yet interesting from a computational viewpoint.

Kernel Approaches. The above SGD algorithm is the core of Pegasos, a well-known soft-SVM technique which is kernelizable. Instead of computing $\boldsymbol{w}_t$ at each iteration, we may use the kernel trick by replacing the large vectors $\boldsymbol{w}_t$ with a kernel $K(\boldsymbol{x}, \boldsymbol{x}')$.

Open question: which kernel would be appropriate for global consistency tasks?

The convex optimization approach its extensions are yet interesting from a computational viewpoint.

Kernel Approaches. The above SGD algorithm is the core of Pegasos, a well-known soft-SVM technique which is kernelizable. Instead of computing $\boldsymbol{w}_t$ at each iteration, we may use the kernel trick by replacing the large vectors $\boldsymbol{w}_t$ with a kernel $K(\boldsymbol{x}, \boldsymbol{x}')$.

Open question: which kernel would be appropriate for global consistency tasks?

Deep Approaches. Polynomial functions can be efficiently represented by sum-product networks, a deep architecture. Livni, Shalev-Shwartz and Shamir (2014) have proposed such an architecture for learning polynomial functions, in which the last layer uses SGD.

Open question: can we efficiently learn sum-product networks for global consistency tasks?

Outline

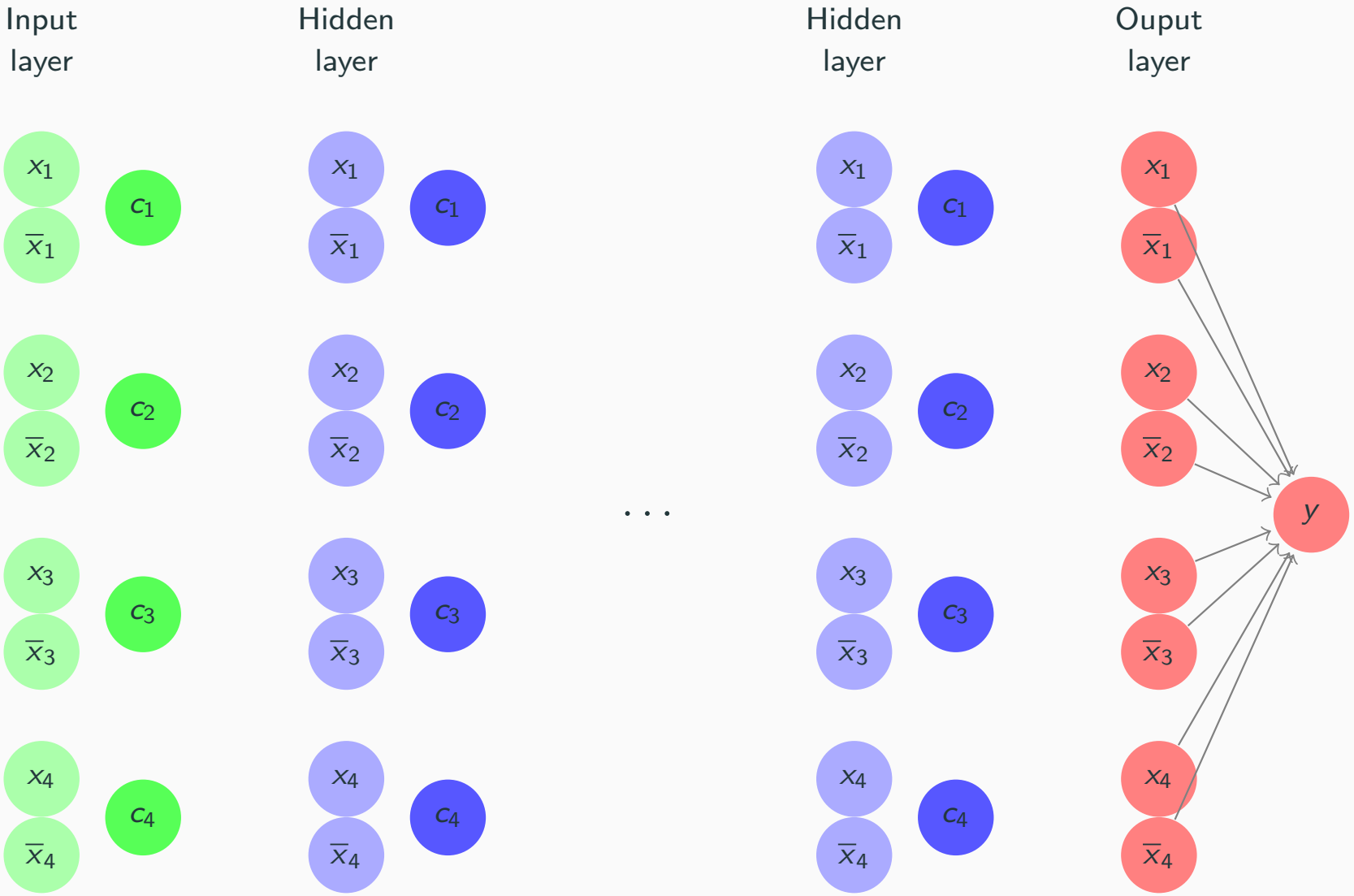
$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$ ✖

$(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_4 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$ ✓

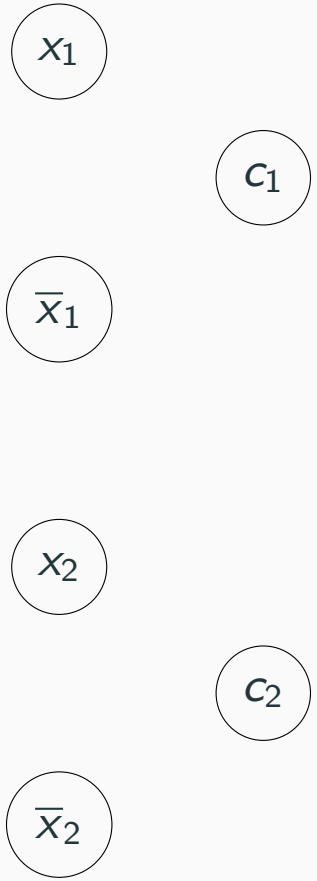
$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_4 \vee \overline{x}_5) \wedge (\overline{x}_1 \vee \overline{x}_4 \vee x_5) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5)$ ✖

⋮

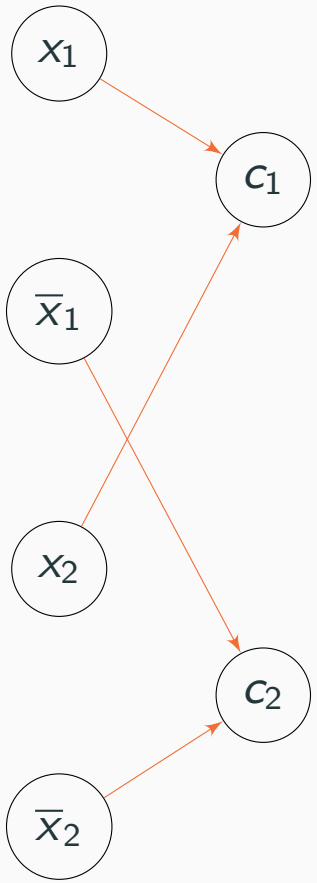
Recall that in the satisfiability task, we have a generator of SAT instances, and the goal is to predict whether each generated CNF formula is SAT or UNSTAT.



The NeuroSAT architecture (Selsam et. al, 2018) is a feedforward neural net in which the input layer and hidden layers are defined from literal embeddings and clause embeddings. The final layer use a vote among literals to determine the output y.

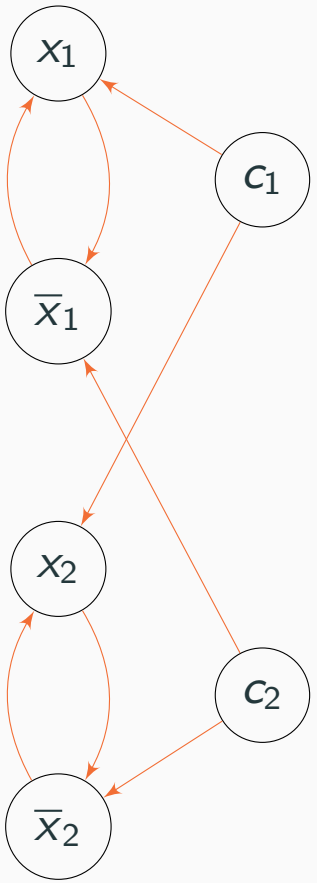


For each layer t, we have an embedding for every literal (l-node) and every clause (c-node). An iteration $t \to t + 1$ consists of two stages:



For each layer t, we have an embedding for every literal (l-node) and every clause (c-node). An iteration $t \to t+1$ consists of two stages:

- first, each c-node receives messages from its connected l-nodes, and updates its embedding accordingly;



For each layer t, we have an embedding for every literal (l-node) and every clause (c-node). An iteration $t \to t+1$ consists of two stages:

- first, each c-node receives messages from its connected l-nodes, and updates its embedding accordingly;
- next, each l-node receives message from its connected c-nodes, as well as its complement, and updates its embedding accordingly.

NeuroSAT is parameterized by:

- Two vectors $\boldsymbol{L}_{\text{INIT}}$ and $\boldsymbol{C}_{\text{INIT}}$
- Three multilayer percetrons $\boldsymbol{L}_{\text{MSG}}$, $\boldsymbol{C}_{\text{MSG}}$ and $\boldsymbol{L}_{\text{VOTE}}$
- Two layer-norm long-short term memory networks $\boldsymbol{L}_{\text{LSTM}}$ and $\boldsymbol{C}_{\text{LSTM}}$

Each layer t involves four matrices:

- $\boldsymbol{L}^{(t)} \in \mathbb{R}^{2n \times d}$ is the matrix storing each literal embedding.
- $\boldsymbol{C}^{(t)} \in \mathbb{R}^{m \times d}$ is the matrix storing each clause embedding.
- $\boldsymbol{L}^{(t)_h}$ and $\boldsymbol{C}_h^{(t)}$ are the corresponding hidden matrices initialized to zero.

The layers are updated as follows:

- $[\boldsymbol{C}^{(t+1)}, \boldsymbol{C}_h^{(t+1)}] = \boldsymbol{C}_{\text{LSTM}}[\boldsymbol{C}_h^{(t)}, \boldsymbol{M}^\top \boldsymbol{L}_{\text{MSG}}(\boldsymbol{L}^{(t)})]$
- $[\boldsymbol{L}^{(t+1)}, \boldsymbol{L}_h^{(t+1)}] = \boldsymbol{L}_{\text{LSTM}}[\boldsymbol{L}_h^{(t)}, F(\boldsymbol{L}^{(t)}), \boldsymbol{M}\boldsymbol{C}_{\text{MSG}}(\boldsymbol{C}^{(t)})]$

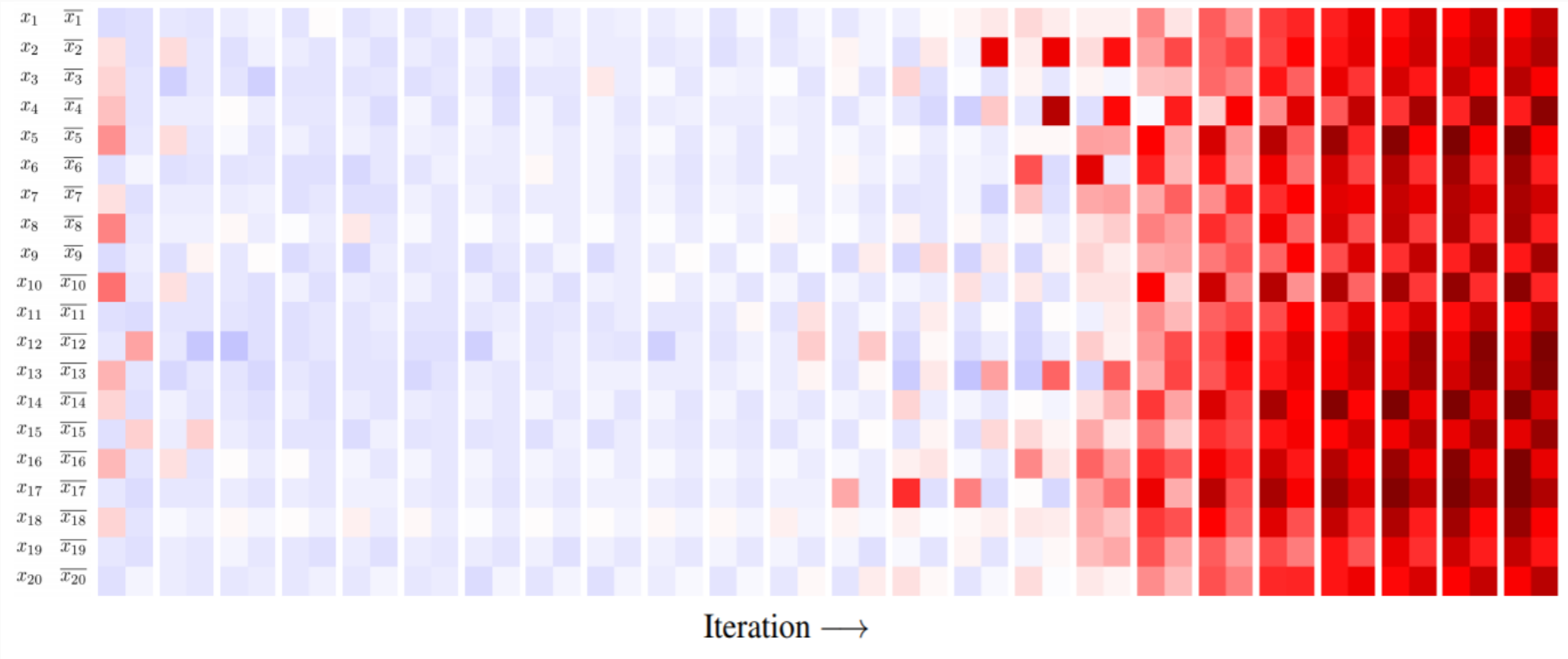
where

- $\boldsymbol{M}$ is the adjacency matrix, i.e. $M(i,j) = 1$ iff $l_i \in c_j$
- $F(\boldsymbol{L})$ swaps each row of $\boldsymbol{L}$ with the row of its complement.

For the output layer,

$$y = \text{mean}(\text{vote}(\boldsymbol{L}^{(T)}))$$

where $\text{vote}(\boldsymbol{L}^{(T)}) \in \mathbb{R}^n$ is the vote for the value of each variable, and y is the average of each vote.



When making a prediction about a new problem, NeuroSAT

- predicts UNSAT with low confidence (light blue) until it finds a satisfying assignment, at which point it guesses SAT with very high confidence (red).
- or keeps predicting UNSAT with low confidence (light blue) until T. It never becomes high confident for UNSAT.

For now, NeuroSAT makes 15 % mistakes in predicting the satisfiability of SR(20) instances. We are far below modern SAT solvers, but this is just the beginning of neuro-sat techniques.