Reliable and Interpretable AI - HS2018

github.com/manuelbre

Bayes' Rule $p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x|\theta)p(\theta)}{\int_{\tilde{\theta}} p(x|\tilde{\theta})p(\tilde{\theta})d\tilde{\theta}}$

Gaussian $p(X|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp[-\frac{(X-\mu)^2}{2\sigma^2}]$

$p(x|\mu,\Sigma) = \frac{1}{\sqrt{2\pi}^d}\frac{1}{\sqrt{|\Sigma|}}\exp[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)]$

$\ln(p(x|\mu,\Sigma)) = -\frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2}(y-\mu)^T\Sigma(y-\mu)$

Expected value $E[X] = \int_X xp(x)dx = \sum_{x\in X} xp(x)$

$E[aX] = aE[X]; \quad E[XY] \overset{\text{indep.}}{=} E[X]E[Y]$

$E[X+Y] = E[X]+E[Y]$

Variance $Var[X] = \int_x (x-\mu)^2 p(x)dx$

$Var[X] = E[(X-E[X])^2] = E[X^2]-E[X]^2$

Norm

$\|\mathbf{x}\|_p := \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$

$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2+\cdots+x_n^2}$

$\|\boldsymbol{x}\|_1 := |x_1|+\cdots+|x_n|.$

$\|\mathbf{x}\|_\infty := \max_i |x_i|$

Efficient projections only exists for L_1,L_2 and L_∞ norms.

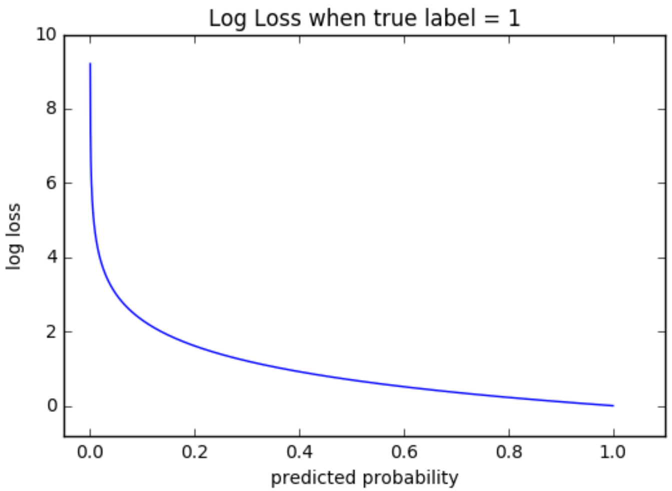
1 Deep Learning

Loss

Cross-entropy $\text{loss}(x, \text{ true label }) = -\log\left(\frac{\exp(x[truelabel])}{\sum_j \exp(x[j])}\right)$

$= -\log\left(softmax(x[\text{truelabel}])\right)$

$= -x[\text{truelabel}] + \log\left(\sum_j \exp(x[j])\right)$, where x is the *logit* output of the NN.



NegativeLogLikelihoodLoss $\text{NLLLoss}(logs, \text{true label}) = -logs[\text{ true label }]$, where for logs the following should be used to make it equal to the *Cross-entropy loss*:

$logs = log(softmax(x))$

2 Adverserial Examples

Targeted FGSM (Fast Gradient Sign Method)

Intuition: Goal is to perturbe the image such that the NN missclassifies the image to target label $\boldsymbol{t}$. Therefore **reduce** the loss of the **target** label.

0. Target label $\boldsymbol{t}$, true label $\boldsymbol{s}$, generally $t \neq s$
1. Compute perturbation: $\eta = \epsilon \cdot \text{sign}\left(\nabla_x \text{loss}_t(x)\right)$
   $\nabla_x \text{loss}_t = \left(\frac{\partial \text{loss}_t}{\partial x_1}, \ldots, \frac{\partial \text{loss}_t}{\partial x_n}\right)$
   ,where loss_t is the entry of the cross entropy loss vector for the target label and x is the image vector.
2. Perturb the input: $x' = x - \eta$
3. Check if: $f(x') = t$, where $f(x)$ is classification result of the NN for x.

Untargeted FGSM (Fast Gradient Sign Method)

0. True label s
1. Compute perturbation: $\eta = \epsilon \cdot \text{sign}\left(\nabla_x \text{loss}_s(x)\right)$
   $\nabla_x \text{loss}_s = \left(\frac{\partial \text{loss}_s}{\partial x_1}, \ldots, \frac{\partial \text{loss}_s}{\partial x_n}\right)$
2. Perturb the input: $x' = x + \eta$
3. Check if: $f(x') \neq s$

PGD (Projected Gradient Descent)

Take $\boldsymbol{k}$ steps of **FGSM** each of size ϵ. After each step project onto $S(x)$. By projecting, we mean that we find the closest point inside the $S(x)$ ball (e.g. L_∞ ball). Here, closest is defined according to some norm (e.g. L_∞). In the region of $S(X)$ we want all point to be classified with the **same** label.

$S(x) = \{x' | \, \|x - x'\|_\infty < \epsilon\}$

Note that the $S(x)$ ball is of size $\boldsymbol{\epsilon}$ which is different than the $\boldsymbol{\epsilon_{step}}$ of the FGSM step and normally it holds that $\epsilon_{step} < \epsilon$.

Projecting on the L_∞ ball is the same as clamping the values: $x'_{projected} = clamp(x', min = x - \epsilon, max = x + \epsilon)$

Note that the resulting $x'_{projected}$ can be inside the L_∞ ball.

Minimization Problem for Defense:

find	θ
minimize	$\rho(\theta)$
where	$\rho(\theta) = \mathbf{E}_{(x,y)\sim D}\left[\max_{x'\in S(x)} L(\theta, x', y)\right]$,
in practice	$\rho(\theta) = \frac{1}{\lvert D_a\rvert}\sum_{(x,y)\in Da} L(\theta, x, y)$

where D_a is dataset of adverserial examples

For which $\rho(\theta)$ is the empirical risk (Loss) and the outer optimization (min) problem (Defense): *Find θ that minimizes the high loss → train robust classifier* (with normal SGD methode $\theta' = \theta - \epsilon_{\text{learning rate}} \cdot \nabla_\theta \rho(\theta)$.)

Further, $\max_{x'\in S(x)} L(\theta, x', y)$ the inner optimization (max) problem (Attack): *Find adverserial x' achieves high loss → adverserial attack.*

Optimization Problem

Objective is to have a **small** perturbation η, such that the image is missclassified. Large perturbation is not wanted:

find	η
minimize	$\|\boldsymbol{\eta}\|_p$
such that	$f(x+\eta) = t$
	$x + \boldsymbol{\eta} \in [0,1]^n$

In general this is a hard problem to optimize with gradient descent. Therefore ease the constrains.

1. Use **objective function**:
   $obj(x+\eta) \leq 0 \quad \Rightarrow \quad f(x+\eta) = t$
   A correct objective function is a function that has $obj(x') \leq 0 \iff p(t) \geq 0.5$

   Sound objective functions for **2-class** NN:
   $obj\,(x') = \text{loss}_t\,(x') - 1$
   $obj\,(x') = \max\left(0, 0.5 - \text{softmax}\,(x')_t\right)$
   For **k-class** NN this is:
   $\text{obj}_{\boldsymbol{k}}(\boldsymbol{x}') = -\log_{\mathbf{k}}\left(\text{softmax}\,(x')_t\right) - 1 = C \cdot \text{loss}_t\,(\boldsymbol{x}') - 1$,
   where $C = \frac{1}{\log_2(\boldsymbol{k})}$ for cross-entropy loss with $\log_2$.

2. Replace norm with **proxy function** because the gradient of e.g. the inf norm is zero for most values except the maximum value.
   Replace $\|\boldsymbol{\eta}\|_\infty$ with $\sum_i \max\left(0, (\boldsymbol{\eta}_i - \tau)\right)$
   If all entries are less then τ then the entire expression is zero. Note: When τ is large, the gradient is similar to the gradient of $\|\eta\|_\infty$. Start with large τ and lower after each iteration.

3. Clamp perturbed image back to box domain after optimization.

Then the optimization becomes:

find	η
minimize	$\|\boldsymbol{\eta}\|_p + c \cdot obj(x+\boldsymbol{\eta})$
such that	$x + \eta \in [0,1]^n$

Diffing Networks

Given two NN trained to learn same function. Perturb Input x such that $class(f_1(x')) \neq class(f_2(x'))$

Use the following objective function, where $f_i(x')_t$ is the softmax output of of NN i w.r.t.

while $\text{class}\,(f_1(x)) = \text{class}\,(f_2(x))$:
  $obj(x) = f_1(x)_t - f_2(x)_t$ → Use as Loss
  $x = x + \epsilon \cdot \frac{\partial obj(x)}{\partial x}$ → Maximize Loss
return x

3 Logic

Goal: Want to query NN such that some logical constrains are satisfied.
Problem: Formulating this as a constrained problem is hard to solve and times out for large NN.
Solution: Translate logical constrain into loss.

Translations

$\forall$x, if $\mathrm{T}(\phi)(\mathrm{x}) = 0 \Rightarrow \phi(x)$ satisfied,
where $\phi(x)$ is logical formula
Use the following constrains to generate loss function.

Logical Term	Translation	Logic Negation ($\neg$)
$t_1 = t_2$	$\|t_1 - t_2\|$	$t_1 < t_2 \vee t_2 < t_1$
$t_1 \leq t_2$	$\max(0, t_1 - t_2)$	$t_1 > t_2$
$t_1 < t_2$	$\mathrm{T}(t_1 + \epsilon \leq t_2)$	$t_1 \geq t_2$
$t_1 \neq t_2$	$\mathrm{T}(t_1 < t_2 \vee t_2 < t_1)$	$t_1 = t_2$
$\varphi \vee \psi$	$T(\varphi) \cdot T(\psi)$	$\neg\varphi \wedge \neg\psi$
$\varphi \wedge \psi$	$T(\varphi) + T(\psi)$	$\neg\varphi \vee \neg\psi$

Problem: When dealing with **real** values in logical domain and **floats** in loss domain one has to assign ϵ to smallest machine value. Thereafter Translation is only valid in one direction ($\mathrm{T}(\phi)(\mathrm{x}) = 0 \Rightarrow \phi(x)$ satisfied). It no longer holds that when $\phi(x)$ satisfied, that the loss is 0.
E.g. $t_1 < t_2 \Rightarrow T(\phi) = \max(0, t_1 + \epsilon - t_2)$
Use, $t_1 = t_2 - \frac{\epsilon}{2}$. It holds that $t_1 < t_2$ but Translation is not satisfied since $T(\phi) = \max(0, t_2 - \frac{\epsilon}{2} - t_2) = \frac{\epsilon}{2} \neq 0$
Therefore, $\mathrm{T}(\phi)(\mathrm{x}) = 0 \Rightarrow \phi(x)$ satisfied,
but $\phi(x)$satisfied $\not\Rightarrow \mathrm{T}(\phi)(\mathrm{x}) = 0$

Example

Goal: Find an image i which gets classified to 9 where the image i is within some distance of the image deer.

0. Logical Formula:
   $\phi(i) = \bigwedge_{j=1, j\neq 9}^{k} \mathrm{NN(i)[j]} < \mathrm{NN(i)[9]} \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty < 25$
   $\wedge \|\mathrm{i} - \mathrm{deer}\|_\infty > 5$

1. Translation into loss:
   $T(\phi) = \sum_{j=1, j\neq 9}^{k} \max(0, \mathrm{NN(i)}[j] + \epsilon - \mathrm{NN(i)[9]}$
   $+ \max(0, \|\mathrm{i} - \mathrm{deer}\|_\infty + \epsilon - 25)$
   $+ \max(0{,}5 + \epsilon - \|\mathrm{i} - \mathrm{deer}\|_\infty)$

2. Train Network with SGD

Train NN with Logic

0. Goal: Want to enforce property ϕ. Find weights for NN, such that expected value of the property increases:
   find θ
   maximize $\rho(\theta)$
   where $\rho(\theta) = \mathbf{E}_{s-D}[\forall \mathbf{z} \cdot \phi(\mathbf{z}, s, \theta)]$

1. Translate into loss:
   find θ
   minimize $\rho(\theta)$
   where $\rho(\theta) = \mathbf{E}_{s\sim D}\left[\mathrm{T}(\phi)\left(z_{\text{worst}}, s, \theta\right)\right]$
   and $z_{\text{worst}} = \arg\min_Z(\mathrm{T}(\neg\phi)(z, s, \theta))$
   Inner minimization: Find worst violation of property.
   Outer minimization: Find weight such that worst violation is minimized.
   Intuitively, we are trying to get the worst possible violation of the formula and then to find a network that minimizes its effect.

2. Solve inner minimization by splitting loss:
   e.g. $\mathrm{loss}(z, x, \theta) = \max(0, \|x - z\|_\infty - \epsilon)$
   $+ \max(0, NN_\theta(z)[3] - \delta)$

   $\rightarrow$ split loss!

   2.1. Solve with PGD:
   $\min_z \mathrm{loss}(z, x, \theta) = \max(0, NN_\theta(z)[3] - \delta)$
   2.2. Project z back onto the $L_\infty(x, \epsilon)$ ball

4 Certifay AI - Abstract Domains

Sound: Correct Approximation of NN.
Precise: Approximation is superset of NN, but should not approximate too much, otherwise can not verify NN.
Efficient: Efficient to compute

Interval Domain

Input x is in the form of $x = [a, b]$.

Operation Rules

Addition: $x_1 + x_2 = [x_1[0] + x_2[0], x_1[1] + x_2[1]]$
Substract.: $[x_1, x_2] - [y_1, y_2] = [x_1 - y_2, x_2 - y_1]$
Addition Scalar: $x_1 + a = [x_1[0] + a, x_1[1] + a]$
Multipl.: $[x_1, x_2] \cdot [y_1, y_2] = [\min(x_1y_1, x_1y_2, x_2y_1, x_2y_2),$
$\max(x_1y_1, x_1y_2, x_2y_1, x_2y_2)]$
Multipl Scalar: $x_1 \cdot a = [l, u]$,
with $l = min(x[0] \cdot a, x[1] \cdot a), u = max(x[0] \cdot a, x[1] \cdot a)$
Funct.: $f([y_1, y_2]) = [\min\{f(y_1), f(y_2)\},$
$\max\{f(y_1), f(y_2)\}]$
Lower Equal: $\leq([l_1, u_1], [l_2, u_2]) = ([l_1, u_1] \sqcap_i [-\infty, u_2]$
$, [l_1, \infty] \sqcap_i [l_2, u_2])$

Zonotope Abstrac Domain

$\hat{m} = a_0^m + \sum_{i=1}^k a_i^m \epsilon_i$
ϵ_i : noise terms ranging $[-1, 1]$ shared between abstract neurons
a_i^n : real number that controls magnitude of noise
Centered around a_0^m

Operation Rules

Multiplication with scalar

$\left(a_0^n + \sum_{i=1}^k a_i^n \epsilon_i\right) \cdot C = \left(C \cdot a_0^n + \sum_{i=1}^k C \cdot a_i^n \epsilon_i\right), C \in \mathbb{R}$

Multiplication of two variable

$$\left(a_0^n + \sum_{i=1}^k a_i^n \epsilon_i\right) \cdot \left(a_0^m + \sum_{i=1}^k a_i^m \epsilon_i\right) = (a_0^n \cdot a_0^m) +$$
$$\sum_{i=1}^k (a_i^n \cdot a_0^m + a_i^m \cdot a_0^n) \cdot \epsilon_i + \sum_{i=1}^k \sum_{j=1}^k a_i^m \cdot a_j^n * \epsilon_i \cdot \epsilon_j$$

where $\epsilon_i \cdot \epsilon_j$ becomes new variable $\epsilon_{i,j}$ and
$\epsilon_{i,j} \in [-1, 1]$ if $i \neq j$
$\epsilon_{i,j} \in [0, 1]$ if $i = j$

Summation

$$\left(a_0^n + \sum_{i=1}^k a_i^n \epsilon_i\right) + \left(a_0^m + \sum_{i=1}^k a_i^m \epsilon_i\right) = (a_0^n + a_0^m)$$
$$+ \sum_{i=1}^k (a_i^n + a_i^m) \cdot \epsilon_i$$

Join

Operation is not closed. Example of the Operation:

$\psi_1 = \begin{cases} \hat{n} = 3 + \epsilon_1 + 2\epsilon_2 \\ \hat{m} = 0 + \epsilon_1 + \epsilon_2 \end{cases}$

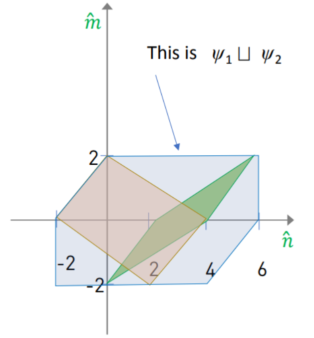
$\sqcup$

$\psi_2 = \begin{cases} \hat{n} = 1 - 2\epsilon_1 + \epsilon_2 \\ \hat{m} = 0 + \epsilon_1 + \epsilon_2 \end{cases}$

$=$

$\hat{n} = 2 + \epsilon_2 + 3\epsilon_u$
$\hat{m} = 0 + \epsilon_1 + \epsilon_2$

New error term is introduced



ReLU

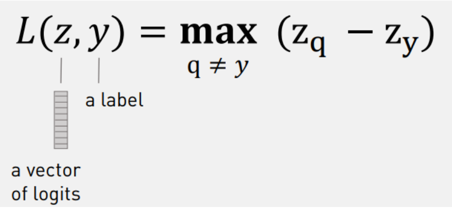
$f_{ReLU}^{\#} = \mathrm{Re}\,LU_2^{\#}(b) \circ \mathrm{Re}\,LU_1^{\#}(a)$ (Affine)
$\mathrm{ReLU}_i^{\#}(x_i)(\psi) = \psi_{\{x_i \geq 0\}} \sqcup \psi_{\{x_i < 0\}}$,
where $\psi_{\{x_i \geq 0\}} = (\psi \sqcap \{x_i \geq 0\})$, for which $\sqcap$ is not defined.
and $\psi_{\{x_i < 0\}} = \begin{cases} [[x_i = 0]](\psi) & \text{if } (\psi \cap \{x_i < 0\}) \neq \perp \\ \perp & \text{otherwise} \end{cases}$

Box

$$z_{box} = \left[\left(a_0^n + \sum_{i=1}^{k} a_i^n \cdot \text{sign}(a_i^n) \cdot (-1) \right), \left(a_0^n + \sum_{i=1}^{k} a_i^n \cdot \text{sign}(a_i^n) \right) \right]$$

4.1 Train provable robust NN

find	θ
minimize	$\rho(\theta)$
where	$\rho(\theta) = \mathbf{E}_{(xy)\sim D} \left[\max_{z \in \gamma(NN^*(\alpha(S(x))))} L(\theta, z, y) \right]$



Set of z can be large. Instead of enumerate the set first transform it. For each z_q use the following: $d_q = z_q - z_y$ and then $u_q = \max(\text{box}(d_q))$ where u_q is the upper bound of the polytope d_q transformed into the box domain. Therefore, $L(z, y) = \max_{q \neq y}(z_q - z_y) = \max_q(u_q)$

5 Visualize CNN

Early layers are Gabor-filter-like filters for edges.
Later layers have more complex, abstract patterns.

Feature Visualization

Template that the NN is looking for.
First layers as Images:
Interpret weights of layers as images. Only works for layers with up to 3 channels.
Visualization by Optimization:

1. Initialize input image with noise.
2. Maximize response of a channel of a certain later. For that define score, i.e. score $(x) = \text{mean}(\text{layer}_n[:, :; , 3])$
3. Use SGD to update input image and add regularization term so that image is constraint to look like an image: $x \leftarrow x + \eta \nabla_x \text{score}(x) + \sum_t \lambda_t R_t(x)$
   Regularization: E.g. penalyze high frequencies

Early layers produce strong line patterns.
While **later** layers show higher level concepts.

Attribution

Location/pixels that are important for NN decision.

Grad-CAM

Highlight region of image that activates layer for some label s.

Attribution I: Grad-CAM for image x and class s

1. Run network forward on x
2. Pick a conv-layer n (usually the last convolutional layer)
3. For each filter $layer_n[0, :, :, k]$ in layer n calculate
   $\alpha_s^k = mean\left(\frac{\partial logit[s]}{\partial layer_n[0,:,:,k]}\right) = \frac{1}{Z}\sum_i\sum_j \frac{\partial logit[s]}{\partial layer_n[0,i,j,k]}$
4. Calculate $L_s = ReLU(\sum_k \alpha_s^k \cdot layer_n[0, :, :, k])$
5. Rescale the map L_s to image-dimensions to obtain saliency map M_s
6. Optionally: Overlay M_s on image x

Where, $L_S = \text{ReLU}\left(\sum_k \underbrace{\alpha_S^k}_{\text{Importance of class s}} \cdot \underbrace{\text{layer}_n[0, :; ; k])}_{\text{Spatial Activations}} \right)$

Meaningful Perturbations

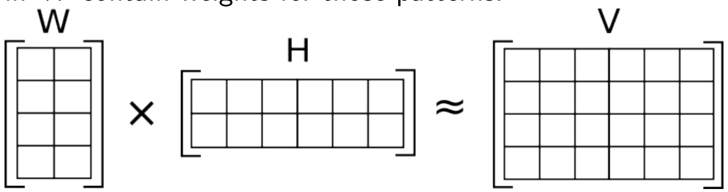
Learn which part of image can be perturbed such that the NN will predict wrong label.

Non Negative Matrix Factorization (NMF)

$\mathbf{WH} = \mathbf{V}$

$\mathbf{W} \in \mathbb{R}^{r \times k}, \quad \mathbf{H} \in \mathbb{R}^{k \times c}, \quad \mathbf{V} \in \mathbb{R}^{r \times c}$

Columns in $\mathbf{H}$ correspond to patterns in $\mathbf{V}$. Whereas rows in $\mathbf{W}$ contain weights for those patterns.



Use NMF on post-ReLU activation of last convolutional layer. Thereafter use feature visualization with help of optimization to visualize the patterns. Can also be used for attribution by thresholding the prototypes and combining them.

6 Probabilistic Programming

$$P(A_i \mid B) = \frac{P(B|A_i)\,P(A_i)}{\sum_j P(B|A_j)\,P(A_j)}.$$

Distributions `x := Distribution`

```
uniform(a,b)
uniformInt(a,b)
gauss(mean,variance)
bernoulli(p)
poisson(mean)
```

Obvervations `observe (x >= 0.5)`

Examples

Run a *psi* program with `psi dice.psi --expectation`
The `--expectation` flag is used to get the probability. Without the flag the exact distribution (PDF) is returned.

```
def main(){ // didItRain
cloudy := flip(0.5);
rain := 0; sprinkler := 0;

if (cloudy){
    rain = flip(0.8);
    sprinkler = flip(0.1);
}else{
    rain = flip(0.2);
    sprinkler = flip(0.5);
}

wetGrass := rain || sprinkler;

observe(wetGrass);
return rain==1; // Probability that it rains,
                // given grass is wet.
}
```

```
def main(){ // Dice
n := 40;
sum := 0;
for i in [0..n){
    dice := uniformInt(1,6);
    sum += dice;
}
average := sum / n;
return average > 4; // Prob. that average of 40..
                    // dice throws is above 4.
}
```

```
def main(){ // Dice 2
n := 20;
sum := 0;
n_6 := 0;
for i in [0..n){
    roll := uniformInt(1,6);
    sum += roll;
    if roll == 6{
        n_6 += 1;
    }
}
average := sum / n;
observe(n_6 >= 10);
eturn average > 4; // Prob. that average of 20...
                   // dice throws is above 4, given
                   // at least 10 rolls are a 6.
}
```

```
def main(){ \\ Program with expectation of Pi.
    x := uniform(0,1);
    y := uniform(0,1);
    within_radius := x*x+y*y <= 1;
    return 4*within_radius; \\Prob. that point ...
                            \\is within radius.
}
```

Differential Privacy

Epsilon - Differential Privacy

$\frac{\Pr[F(x)\in\Phi]}{\Pr[F(x')\in\Phi]} \leq \exp(\varepsilon)$

, where $F(x)$ is the output of the database query including randomization, x' is a database that differs on a single element w.r.t. to x and Φ is the secret.

7 Programming by Examples (PBE)

A new frontier in AI where one learns an interpretable program from user-provided examples.

Requires very few input-output examples. Assumes the given examples are representatives.

PBE problem definition

Given: A domain specific Language (DSL) & set of input-output examples.

Goal: Learn a function over the DSL which is consistent with the provided examples.