Exercise 02 - Solution

Neural Networks & Adversarial Examples

Reliable and Interpretable Artificial Intelligence
ETH Zurich

Problem 1. In this introductory exercise we will use a 3 layer toy network N. We denote a layer as $l_{\boldsymbol{A}}(\boldsymbol{x}) := \boldsymbol{A}\boldsymbol{x}$, the ReLU-activation (Rectified Linear Unit) – which applies $\max(\boldsymbol{x}_i, 0)$ elementwise – $\text{ReLU}(\boldsymbol{x}) = \max(\boldsymbol{x}, \boldsymbol{0})$. We write

$$N(\boldsymbol{x}) := (l_{\boldsymbol{C}} \circ \text{ReLU} \circ l_{\boldsymbol{B}} \circ \text{ReLU} \circ l_{\boldsymbol{A}})(\boldsymbol{x})$$

where $\circ$ denotes function composition and $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{10\times 10}$ as well as $\boldsymbol{C} \in \mathbb{R}^{3\times 10}$ are matrices. Further, we let $\text{softmax}(\boldsymbol{x})$ denote the softmax-function for $\boldsymbol{x} \in \mathbb{R}^n$,

$$\text{softmax}(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_{j=1}^{n} e^{\boldsymbol{x}_j}} \tag{1}$$

for $i \in \{1, \dots, n\}$.

Answer the following questions:

1. For the given matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ what is the space of possible inputs to N, i.e. the domain of N?

2. For the given matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ what is the space of possible outputs of N, i.e. the codomain of N? Can the output be seen as a vector of probabilities corresponding to the class probabilities of a categorical distribution?

3. For the given matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ what is the space of possible outputs of $(\text{softmax} \circ N)$, i.e. the codomain of $\text{softmax} \circ N$? Can the output be seen as a vector of probabilities corresponding to the class probabilities of a categorical distribution?

4. If N is a classifier, for how many classes does it work? (Assuming the standard usage of neural networks in classification.)

5. Assume N has been trained and you want to use it to for classification on an input $\boldsymbol{x}$. Do you use $(\text{softmax} \circ N)$ or N? Justify your answer.

6. For a data point $\boldsymbol{x}$ with label $y = 2$ you want to create an adversarial example $\boldsymbol{x}'$ that is close to $\boldsymbol{x}$ ($\|\boldsymbol{x} - \boldsymbol{x}'\|_\infty < \epsilon$) and that classifies to $y' = 1$. Do you use an untargeted or targeted attack?

7. You use the Fast Gradient Sign Method (FGSM) (cf. slides 26-28 of Lecture 2) to compute the perturbation in question 6. Write down the necessary equations to obtain $\boldsymbol{x}'$. Is the computation applied to N or softmax $\circ N$?

8. (**Coding**) In `fgsm.py` you are provided with an implementation for N (for some matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$) in PyTorch[1]. Follow the instructions there (does not assume familiarity with PyTorch) and complete the code.[2]

Solution 1. We consider the function signature of the network step-by-step:

$$
\begin{aligned}
l_{\boldsymbol{A}} &: \mathbb{R}^{10} \to \mathbb{R}^{10} \\
\text{ReLU} \circ l_{\boldsymbol{A}} &: \mathbb{R}^{10} \to (\mathbb{R}^{\geq 0})^{10} \\
l_{\boldsymbol{B}} \circ \text{ReLU} \circ l_{\boldsymbol{A}} &: \mathbb{R}^{10} \to \mathbb{R}^{10} \\
\text{ReLU} \circ l_{\boldsymbol{B}} \circ \text{ReLU} \circ l_{\boldsymbol{A}} &: \mathbb{R}^{10} \to (\mathbb{R}^{\geq 0})^{10} \\
l_{\boldsymbol{C}} \circ \text{ReLU} \circ l_{\boldsymbol{B}} \circ \text{ReLU} \circ l_{\boldsymbol{A}} &: \mathbb{R}^{10} \to \mathbb{R}^{3} \qquad (2)
\end{aligned}
$$

For $\boldsymbol{x} \in \mathbb{R}^n$ the signature of the softmax function is:

$$
\text{softmax}(\boldsymbol{x}) : \mathbb{R}^n \to [0,1]^n \qquad \text{and} \qquad \sum_{i=1}^{n} \text{softmax}(\boldsymbol{x})_i = 1 \qquad (3)
$$

We generally call the inputs to the softmax function logits. So N outputs the class logits and softmax $\circ N$ outputs the class probabilities.

We thus answer the questions:

1. $\mathbb{R}^{10}$ by eq. (2).

2. $\mathbb{R}^3$ by eq. (2). No. For an arbitrary $\boldsymbol{y} \in \mathbb{R}^3$ normalization is not guaranteed, i.e. there exists $\boldsymbol{y}$ with $\sum_{i=1}^{3} \boldsymbol{y}_i \neq 1$.

3. $[0,1]^3$ by eqs. (2) and (3). Due to the properties of the softmax function (see eq. (3)) any vector $\boldsymbol{y}$ output by a softmax function *can* be considered a probability distribution, and is often treated as such in calculation.

[1] `pytorch.org`

[2] We provide short PyTorch examples along with the rest of the materials in the first exercises, but we strongly recommend that you familiarize yourself with it ahead of the project.

4. 3. As by the answer to 3 we can treat the output of (softmax $\circ N$) function as a probability distribution over the 3 classes and train the network so that it puts the highest probability on the class it believes to be correct.

5. As by the answer to 4, the output of (softmax $\circ N$) can be seen as a probability distribution over the classes. Picking $\text{argmax}_{c\in\{1,\ldots,3\}}\left(\text{softmax}\circ N\right)(\boldsymbol{x})_c$ gives us the most probable class for $\boldsymbol{x}$. Since

$$\text{argmax}_{c\in\{1,\ldots,3\}}\left(\text{softmax}\circ N\right)(\boldsymbol{x})_c = \text{argmax}_{c\in\{1,\ldots,3\}} N(\boldsymbol{x})_c \tag{4}$$

we can also just evaluate N, unless we explicitly also want the class probabilities. To see why eq. (4) holds, let $\boldsymbol{y} = N(\boldsymbol{x})$ be the neural network logits and rewrite eq. (1) as

$$\text{softmax}(\boldsymbol{y})_i = \frac{e^{\boldsymbol{y}_i}}{\sum_{j=1}^{n} e^{\boldsymbol{y}_j}} = \frac{e^{\boldsymbol{y}_i}}{Z}$$

where $Z = \sum_{j=1}^{n} e^{\boldsymbol{y}_j} > 0$ is fixed for a fixed vector $\boldsymbol{y}$. Eq. (4) follows from the fact that $\boldsymbol{y}_i \le \boldsymbol{y}_j \iff \frac{e^{\boldsymbol{y}_i}}{Z} \le \frac{e^{\boldsymbol{y}_j}}{Z}$.

6. As we have the specific target $t = 1$ we use a targeted attack.

7. $\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{\eta} = \boldsymbol{x} - \epsilon \cdot \text{sign}\left(\nabla_{\boldsymbol{x}} \text{loss}_t\left(N(\boldsymbol{x})\right)\right)$.

   The remaining question is what loss to use. The original FGSM paper [2] suggests to use the same loss that is used in training. However, this information is not provided in the question. Since we are considering a classification task, it is reasonable to assume the standard cross-entropy loss:

$$\begin{aligned}\text{loss}_t(\boldsymbol{x}) &= \text{Cross-Entropy}\left(\left(\text{softmax}\circ N\right)(\boldsymbol{x}), t\right) = -\sum_{c=1}^{C}[c=t]\log(\text{softmax}(N(\boldsymbol{x}))_c) \\ &= -N(\boldsymbol{x})_t + \log(Z) = -N(\boldsymbol{x})_t + \underbrace{\log\left(\sum_{c=1}^{C}\exp(N(\boldsymbol{x})_c)\right)}_{(*)} \end{aligned}\tag{5}$$

   where the cross-entropy between a probability vector and a scalar t denotes the cross-entropy between the probability vector and a distribution that puts all mass on the category indicated by t, C indicates the number of classes (which is 3 in this case) and $[\phi]$ is the Iverson Bracket.

   Thus the overall attack becomes:

$$\begin{aligned}
\boldsymbol{x}' &= \boldsymbol{x} - \boldsymbol{\eta} \\
&= \boldsymbol{x} - \epsilon \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{x}}\left(-N(\boldsymbol{x})_t + \log\left(\sum_{c=1}^{C}\exp(N(\boldsymbol{x})_c)\right)\right)\right) \\
&= \boldsymbol{x} + \epsilon \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{x}} N(\boldsymbol{x})_t \underbrace{-\nabla_{\boldsymbol{x}} \log\left(\sum_{c=1}^{C}\exp(N(\boldsymbol{x})_c)\right)}_{(*)}\right) \qquad (6)
\end{aligned}$$

By eq. (6) we see that whether N or (softmax $\circ N$) is used, is mostly a point of semantics. The intuition is that we increase the logits target class in the $\nabla_{\boldsymbol{x}} N(\boldsymbol{x})_t$ term and at the same time decrease the logits of all other terms. Thus, we push the classification more towards class t.

It would be reasonable to say that softmax $\circ N$ yields the version discussed here and the version using N would just drop the $(*)$ terms in eqs. (5) and (6). This version would then only increase the logit for target class, but not decrease all others. For adversarial examples we will only use the "proper" version given by eq. (6).

(Optional) Eq. (6) can be expressed in terms of $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{x}$ by rewriting the gradient computation as follows (the resulting equations are given as a reference to the interested students):

$$\begin{aligned}
&\nabla_{\boldsymbol{x}}\left(-N(\boldsymbol{x})_t + \log\left(\sum_{c=1}^{C}\exp(N(\boldsymbol{x})_c)\right)\right) = \\
&-\nabla_{\boldsymbol{x}} N(\boldsymbol{x})_t + \frac{1}{\sum_{c=1}^{C}\exp(N(\boldsymbol{x})_c)} \cdot \sum_{c=1}^{C}\exp(N(\boldsymbol{x})_c)\nabla_{\boldsymbol{x}} N(\boldsymbol{x})_c \qquad (7)
\end{aligned}$$

The $\nabla_{\boldsymbol{x}} N(\boldsymbol{x})_i$ terms can be subsequently written as

$$\begin{aligned}
\nabla_{\boldsymbol{x}} N(\boldsymbol{x})_i &= \nabla_{\boldsymbol{x}}\left[\boldsymbol{C}\,\mathrm{ReLU}\left(\boldsymbol{B}\,\mathrm{ReLU}\left(\boldsymbol{A}\boldsymbol{x}\right)\right)\right]_i \\
&= \nabla_{\boldsymbol{x}} \boldsymbol{C}_{i,:}\,\mathrm{ReLU}\left(\boldsymbol{B}\,\mathrm{ReLU}\left(\boldsymbol{A}\boldsymbol{x}\right)\right) \\
&= \left(\nabla_{\boldsymbol{x}}\,\mathrm{ReLU}\left(\boldsymbol{B}\,\mathrm{ReLU}\left(\boldsymbol{A}\boldsymbol{x}\right)\right)\right)^T \boldsymbol{C}_{i,:}^T
\end{aligned}$$

where $\boldsymbol{C}_{i,:}$ denotes the i-th row of $\boldsymbol{C}$.

The derivative of a ReLU combined with matrix multiplication is given by

$$\nabla_{\boldsymbol{x}}\,\mathrm{ReLU}\left(\boldsymbol{A}\boldsymbol{x}\right) = R(\boldsymbol{A}\boldsymbol{x})\nabla_{\boldsymbol{x}}\boldsymbol{A}\boldsymbol{x} = R(\boldsymbol{A}\boldsymbol{x})\boldsymbol{A}$$

where

$$R(\boldsymbol{y}) = \operatorname{diag}(h(\boldsymbol{y})), \qquad h(\boldsymbol{y})_i = \begin{cases} 1 & \text{if } \boldsymbol{y}_i > 0 \\ 0 & \text{if } \boldsymbol{y}_i < 0 \end{cases}$$

and $\operatorname{diag}(\boldsymbol{y})$ denotes the diagonal matrix that has $\boldsymbol{y}$ on its diagonal.

By putting all of this together, we obtain

$$\begin{aligned}
\nabla_{\boldsymbol{x}} N(\boldsymbol{x})_i &= \left(\nabla_{\boldsymbol{x}} \operatorname{ReLU}\left(\boldsymbol{B} \operatorname{ReLU}\left(\boldsymbol{A}\boldsymbol{x}\right)\right)\right)^T \boldsymbol{C}_{i,:}^T \\
&= \left(R(\boldsymbol{B} \operatorname{ReLU}(\boldsymbol{A}\boldsymbol{x}))\nabla_{\boldsymbol{x}} \boldsymbol{B} \operatorname{ReLU}\left(\boldsymbol{A}\boldsymbol{x}\right)\right)^T \boldsymbol{C}_{i,:}^T \\
&= \left(R(\boldsymbol{B} \operatorname{ReLU}(\boldsymbol{A}\boldsymbol{x}))\boldsymbol{B}\nabla_{\boldsymbol{x}} \operatorname{ReLU}\left(\boldsymbol{A}\boldsymbol{x}\right)\right)^T \boldsymbol{C}_{i,:}^T \\
&= \left(R(\boldsymbol{B} \operatorname{ReLU}(\boldsymbol{A}\boldsymbol{x}))\boldsymbol{B}R(\boldsymbol{A}\boldsymbol{x})\nabla_{\boldsymbol{x}} \boldsymbol{A}\boldsymbol{x}\right)^T \boldsymbol{C}_{i,:}^T \\
&= \left(R(\boldsymbol{B} \operatorname{ReLU}(\boldsymbol{A}\boldsymbol{x}))\boldsymbol{B}R(\boldsymbol{A}\boldsymbol{x})\boldsymbol{A}\right)^T \boldsymbol{C}_{i,:}^T \\
&= \left(\boldsymbol{C}_{i,:}R(\boldsymbol{B} \operatorname{ReLU}(\boldsymbol{A}\boldsymbol{x}))\boldsymbol{B}R(\boldsymbol{A}\boldsymbol{x})\boldsymbol{A}\right)^T =: F_i(\boldsymbol{x})
\end{aligned}$$

and with this finally write the attack as

$$\boldsymbol{x}' = \boldsymbol{x} + \varepsilon \operatorname{sign}\left(F_t(\boldsymbol{x}) - \frac{1}{\sum_{c=1}^{C} \exp(N(\boldsymbol{x})_c)} \cdot \sum_{c=1}^{C} \exp(N(\boldsymbol{x})_c) F_c(\boldsymbol{x})\right)$$

Luckily, we do not need to calculate all of this for each attack. We can perform a backward-pass through the network back to the input (in a typical training process this is just done back to each *weight*). So we can use a standard neural-network toolkit such as PyTorch in the next question.

8. See `fgsm_solution.py`.

Note: Many of the questions in Task 1 are very open-ended or tricky questions in order to promote longer considerations and discussions on your part and on the part of this master solution. The questions on the exam will be much clearer.

Problem 2. Carlini and Wagner ([1]) (slides 32-43 in Lecture 2) state the following (notation adapted to lecture notation):

We define an objective function obj *such that* $f(\boldsymbol{x}+\boldsymbol{\eta}) = t$ *if and only if* $\operatorname{obj}^t(\boldsymbol{x}+\boldsymbol{\eta}) \leq 0$. *There are many possible choices for* obj*:*

$$\operatorname{obj}_1^t(\boldsymbol{x}') = -\operatorname{loss}_t(\boldsymbol{x}') + 1 \tag{8}$$

$$\vdots$$

N is a neural network, similar to the first problem, and $f(\boldsymbol{x}) = \operatorname{argmax}_k \left((\operatorname{softmax} \circ N)(\boldsymbol{x}) \right)_k$ denotes a neural network making a classification. $N(\boldsymbol{x})$ computes the network logits, and $f(\boldsymbol{x})$ is syntactic sugar for also making the classification. Assume that the network can classify between C classes (0-indexed) and consider $\operatorname{loss}_t(\boldsymbol{x})$ to be the cross-entropy loss on the neural network output $N(\boldsymbol{x}')$, with t as the target label:

$$\operatorname{loss}_t(\boldsymbol{x}) = \text{Cross-Entropy}\left((\operatorname{softmax} \circ N)(\boldsymbol{x}), t \right) = -\sum_{c=0}^{C-1} [c = t] \log(\operatorname{softmax}(N(\boldsymbol{x}))_c)$$

$[\phi]$ (called Iverson Brackets) evaluats to 1 if the predicate ϕ is true and to 0 otherwise.

1. Show that the statement (8) is wrong by giving a counterexample.
2. To correct this oversight you will need to adjust the definition of $\operatorname{obj}^t(\boldsymbol{x}')$. Make these adjustments.
3. What is the additional key constraint in the Carlini-Wagner attacks in comparison to FGSM?
4. Why do the authors introduce the proxy objective function $\operatorname{obj}^t(\boldsymbol{x} + \boldsymbol{\eta})$?

Solution 2.

1. Consider the output $\operatorname{softmax}(N(\boldsymbol{x}')) = \left(\begin{smallmatrix} 0.7 \\ 0.3 \end{smallmatrix}\right)$ and the target label $t = 0$. $\operatorname{obj}_1^{t=0}(\boldsymbol{x}')$ evaluates to 0.64, but $f(\boldsymbol{x}') = \operatorname{argmax}_k N(\boldsymbol{x}')_k = 0 = t$, i.e. we have $\operatorname{obj}_1^{t=0}(\boldsymbol{x}) > 0$ but the property $f(\boldsymbol{x}') = t$ does hold. For $t = 1$ we observe a value of -0.20 for the objective function, but $f(\boldsymbol{x}') = 0 \neq t$.

2. Consider the new objective functions

$$\operatorname{obj}_{1*}^t(\boldsymbol{x}') = \operatorname{loss}_t(\boldsymbol{x}') - 1 = -\log\left((\operatorname{softmax} \circ N)(\boldsymbol{x}')_t \right) - 1$$

   and

$$\operatorname{obj}_{1**}^t(\boldsymbol{x}') = -\frac{1}{\log(C)} \log\left((\operatorname{softmax} \circ N)(\boldsymbol{x}')_t \right) - 1 = -\log_C\left((\operatorname{softmax} \circ N)(\boldsymbol{x}')_t \right) - 1$$

   Figure 1 shows the value of obj_1^t, $\operatorname{obj}_{1*}^t$ and $\operatorname{obj}_{1**}^t$ for a range of inputs. We observe that as soon as $\operatorname{softmax}(N(\boldsymbol{x}'))_t \geq 0.5$ the objective $\operatorname{obj}_{1**}^t$ becomes less than or equal to 0 for $C = 2$. If we perform two-class classification this gives us the exact statement from the paper: $f(\boldsymbol{x} + \boldsymbol{\eta}) = t \iff \operatorname{obj}_{1**}^t(\boldsymbol{x} + \boldsymbol{\eta}) \leq 0$. For classification with n classes we obtain the statement from the lecture: $\operatorname{obj}_{1**}^t(\boldsymbol{x} + \boldsymbol{\eta}) \leq 0 \implies f(\boldsymbol{x} + \boldsymbol{\eta}) = t$.

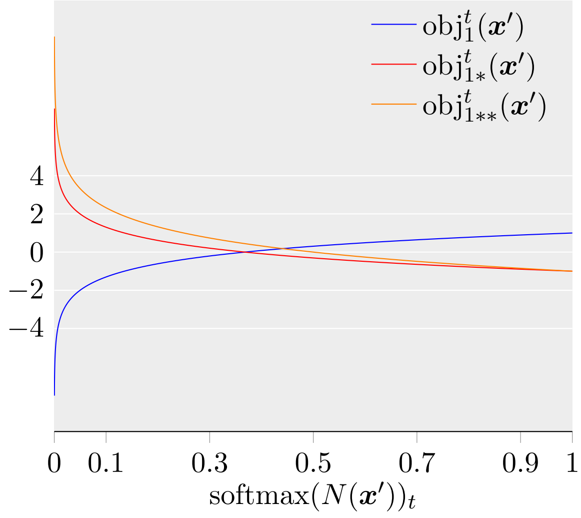

Figure 1: The objective function obj^t_1, obj^t_{1*} and obj^t_{1**} for different values of $\mathrm{softmax}(N(\boldsymbol{x}'))_t$ and $C = 2$.

Note: This is a simple sign mistake, and a small shortcoming in discussing the formulation. [1] is valuable research, and this small mistake does not change its contribution, but it makes for a good introductory exercise.

3. Carlini and Wagner explicitly try to find adversarial examples $\boldsymbol{x}'$ which are close to the original $\boldsymbol{x}$. Therefore, in their formulation of the optimization problem they aim to minimize the distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$, captured by the l_p norm $\|\boldsymbol{x} - \boldsymbol{x}'\|_p = \|\boldsymbol{\eta}\|_p$ (for $p \in \{0, 2, \infty\}$). In contrast, the only guarantee that FGSM gives is that $\boldsymbol{x}' \in [\boldsymbol{x} - \epsilon, \boldsymbol{x} + \epsilon]$ (the addition and subtraction being element-wise). FGSM is designed to be fast, not optimal, in sense that it may not compute minimal perturbation and the adversarial images may end up being too distorted for large values of ϵ.

4. The objective function $\mathrm{obj}^t(\boldsymbol{x}+\boldsymbol{\eta})$ is introduced in order to relax the hard discrete constraint $f(\boldsymbol{x} + \boldsymbol{\eta}) = t$ which is very difficult to integrate into a continuous optimization problem.

Problem 3. In the lecture we also looked at the optimization problem phrases by [1]:

$$\begin{aligned}&\text{find } \boldsymbol{\eta}\\ &\text{minimize } \|\boldsymbol{\eta}\|_p + c \cdot \mathrm{obj}(\boldsymbol{x} + \boldsymbol{\eta})\\ &\text{such that } x + \boldsymbol{\eta} \in [0, 1]^n\end{aligned}$$

Optimizing the norm directly can be problematic, especially in the case of $\|\cdot\|_\infty$. In this task we will investigate this and a surrogate term. To simplify the notation we will assume that $\boldsymbol{x}$ and $\boldsymbol{\eta}$ are n-vectors here (although they are usually matrices representing images). We define $h(\boldsymbol{\eta}) = \|\boldsymbol{\eta}\|_\infty$ and $g(\boldsymbol{\eta}) = \sum_{i=0}^{n} \max(\boldsymbol{\eta}_i - \tau, 0)$ for some constant τ.

1. Calculate $\frac{\partial}{\partial \boldsymbol{\eta}} h(\boldsymbol{\eta})$.
2. Calculate $\frac{\partial}{\partial \boldsymbol{\eta}} g(\boldsymbol{\eta})$.
3. Instantiate the above derivatives for $\boldsymbol{\eta} = (1.00001, 1.0, 1.0, 1.0, 0.001, 0.001)^T$ and for $\tau = 0.9$ and $\tau = 2.0$.
4. What is a problem when minimizing $h(\boldsymbol{\eta})$ with gradient decent?
5. Does $g(\boldsymbol{\eta})$ suffer the same problem?

Solution 3.

$$\frac{\partial}{\partial \boldsymbol{\eta}_i} h(\boldsymbol{\eta}) = \begin{cases} 1 & \text{if } i \in \text{argmax}_k(|\boldsymbol{\eta}_k|) \\ 0 & \text{else} \end{cases}$$

$$\frac{\partial}{\partial \boldsymbol{\eta}_i} g(\boldsymbol{\eta}) = \begin{cases} 1 & \text{if } \boldsymbol{\eta}_i > \tau \\ 0 & \text{else} \end{cases}$$

Note that for $\frac{\partial}{\partial \boldsymbol{\eta}} h(\boldsymbol{\eta})$ there could be multiple maxima. Mathematically there is no defined derivative then, however in automatic differentiation engines such as PyTorch the derivative is 1 for all maxima. For this exercise either definition is fine.

Instantiating the above derivatives for the given $\boldsymbol{\eta}$ we obtain $\frac{\partial}{\partial \boldsymbol{\eta}} h(\boldsymbol{\eta}) = (1.0, 0, 0, 0, 0, 0)$, $\frac{\partial}{\partial \boldsymbol{\eta}} g(\boldsymbol{\eta})_{\tau=0.9} = (1.0, 1.0, 1.0, 1.0, 0, 0)$ and $\frac{\partial}{\partial \boldsymbol{\eta}} g(\boldsymbol{\eta})_{\tau=2.0} = (0, 0, 0, 0, 0, 0)$.

When optimizing $h(\boldsymbol{\eta})$ with gradient decent we are always only optimizing the component of $\boldsymbol{\eta}$ with the largest absolute value. This can yield to gradient decent running many iterations in the best case or – more likely – oscillation between minimizing a few values, as decreasing one component might increase others (due to the obj-term), as discussed in the lecture.

Using $g(\boldsymbol{\eta})$ can mitigate this problem for a well-chosen τ. [1] suggest to lower τ during the optimization procedure in order to produce a solution $\boldsymbol{\eta}$ with small L_∞-norm. Also note that $g(\boldsymbol{\eta})$ only works in the case where we expect the entries of $\boldsymbol{\eta}$ to be positive. To be a closer surrogate to the L_∞-norm we could instead use $g^*(\boldsymbol{\eta}) = \sum_{i=0}^{n} \max(|\boldsymbol{\eta}_i| - \tau, 0)$. However for optimizing in $\boldsymbol{\eta} \in [0, 1]^n$ this does not matter.

References

[1] Nicholas Carlini and David A. Wagner. “Towards Evaluating the Robustness of Neural Networks”. In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 39–57. DOI: `10.1109/SP.2017.49`. URL: `https://arxiv.org/abs/1608.04644`.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: `http://arxiv.org/abs/1412.6572`.