Machine Learning for Graphs and Sequential Data Exercise Sheet 07

Robustness of Machine Learning Models

Exercises marked with a $(*)$ will be discussed in the in-person exercise session.

Problem 1: (*) Suppose we have a trained binary logistic regression classifier with weight vector $\boldsymbol{w} \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$. Given a sample $\boldsymbol{x} \in \mathbb{R}^d$ we want to construct an adversarial example via gradient descent on the binary cross entropy loss:

$$\mathcal{L}(\boldsymbol{x}, y) = -y \log(\sigma(z)) - (1-y)\log(1-\sigma(z)),$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic sigmoid function, $z = \boldsymbol{w}^T\boldsymbol{x} + b$, and $y \in \{0,1\}$ is the class label of the sample at hand.

a) Derive the gradient $\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}, y)$. How do you interpret the result?

Hint: You may use the relation $1 - \sigma(z) = \sigma(-z)$.

$$\begin{aligned}
\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}, y) &= \frac{-y}{\sigma(z)}\frac{\partial\sigma(z)}{\partial z}\nabla_{\boldsymbol{x}} z - \frac{1-y}{\sigma(-z)}\frac{\partial\sigma(-z)}{\partial z}\nabla_{\boldsymbol{x}} z \\
&= \frac{-y}{\sigma(z)}\sigma(z)\sigma(-z)\boldsymbol{w} + \frac{1-y}{\sigma(-z)}\sigma(-z)\sigma(z)\boldsymbol{w} \\
&= -y\sigma(-z)\boldsymbol{w} + (1-y)\sigma(z)\boldsymbol{w}
\end{aligned}$$

The gradient is orthogonal to the decision boundary and points in the direction of the wrong class, depending on y.

b) Provide a closed-form expression for the worst-case perturbed instance $\tilde{\boldsymbol{x}}^*$ (measured by the loss $\mathcal{L}$) for the perturbation set $\mathcal{P}(x) = \{\tilde{\boldsymbol{x}} : \|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|_2 \leq \epsilon\}$, i.e.

$$\tilde{\boldsymbol{x}}^* = \underset{\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_2 \leq \epsilon}{\arg\max}\ \mathcal{L}(\tilde{\boldsymbol{x}}, y)$$

Since the loss is convex w.r.t. the data, taking a gradient step of magnitude ϵ towards the wrong class will result in the maximum increase in loss:

$$\tilde{\boldsymbol{x}}^* = \boldsymbol{x} - \epsilon\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \text{ if } y = 1$$
$$\tilde{\boldsymbol{x}}^* = \boldsymbol{x} + \epsilon\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \text{ if } y = 0$$

c) What is the smallest value of ϵ for which the sample $\boldsymbol{x}$ is misclassified (assuming it was correctly classified before)?

For the sample to change classification we need to have $\sigma(z) = 0.5 \Leftrightarrow \boldsymbol{w}^T\tilde{\boldsymbol{x}} + b = 0$. Plugging in the perturbation we get for $y = 1$:

$$\begin{aligned}\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\epsilon\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} + b &= 0\\ \boldsymbol{w}^T\boldsymbol{x} - \epsilon\|\boldsymbol{w}\|_2 + b &= 0\\ \frac{1}{\|\boldsymbol{w}\|_2}(\boldsymbol{w}^T\boldsymbol{x} + b) &= \epsilon\end{aligned}$$

Thus, for a misclassification we need $\epsilon > \frac{1}{\|\boldsymbol{w}\|_2}(\boldsymbol{w}^T\boldsymbol{x} + b)$.

Analogously for $y = 0$ we obtain $\epsilon > \frac{1}{\|\boldsymbol{w}\|_2}(-\boldsymbol{w}^T\boldsymbol{x} - b)$

d) We would now like to perform adversarial training. Provide a closed-form expression of the worst-case loss

$$\hat{\mathcal{L}}(\boldsymbol{x}, y) = \max_{\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_2 \le \epsilon} \mathcal{L}(\tilde{\boldsymbol{x}}, y)$$

as a function of $\boldsymbol{x}$ and $\boldsymbol{w}$. How do you interpret the results?

$$\begin{aligned}\hat{\mathcal{L}}(\boldsymbol{x}, y) &= \max_{\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_2 \le \epsilon} \mathcal{L}(\tilde{\boldsymbol{x}}, y)\\ &= \mathcal{L}(\tilde{\boldsymbol{x}}^*, y)\\ &= -y\log(\sigma(\boldsymbol{w}^T\boldsymbol{x} - \epsilon\frac{\boldsymbol{w}^T\boldsymbol{w}}{\|\boldsymbol{w}\|_2} + b)) - (1-y)\log(\sigma(-\boldsymbol{w}^T\boldsymbol{x} - \epsilon\frac{\boldsymbol{w}^T\boldsymbol{w}}{\|\boldsymbol{w}\|_2} - b))\\ &= -y\log(\sigma(\boldsymbol{w}^T\boldsymbol{x} - \epsilon\|\boldsymbol{w}\|_2 + b)) - (1-y)\log(\sigma(-\boldsymbol{w}^T\boldsymbol{x} - \epsilon\|\boldsymbol{w}\|_2 - b))\end{aligned}$$

Consider the case $y = 1$ ($y = 0$ follows symmetrically). The input to the sigmoid function is shifted to the left (i.e. negative direction) by $\epsilon\|\boldsymbol{w}\|_2$, reducing the predicted probability of the sample $\boldsymbol{x}$ belonging to class 1. Thus, only if $\boldsymbol{w}^T x + b \ge \epsilon\|\boldsymbol{w}\|_2$ the sample will be classified as belonging to class 1. We can interpret this as trying to enforce that each sample has at least a distance of $\epsilon\|\boldsymbol{w}\|_2$ to the decision boundary. Moreover, this margin is proportional to the norm of the weight vector, so simply increasing the norm of $\boldsymbol{w}$ does not lead to the desired outcome, since we can move $\epsilon\|\boldsymbol{w}\|_2$ units towards the decision boundary for a unit norm change on the sample $\boldsymbol{x}$. Note that, in contrast to support vector machines (SVMs), even when the samples have a margin of at least $\epsilon\|\boldsymbol{w}\|_2$ to the decision boundary, we have non-zero loss and continue training.

Problem 2: (*) In the lecture on exact certification of neural network robustness we have considered $K - 1$ optimization problems (one for each incorrect class) of the form (c.f. slide 42):

$$m_t^* = \min_{\tilde{\boldsymbol{x}}, \boldsymbol{y}^{(l)}, \hat{\boldsymbol{x}}^{(l)}, \boldsymbol{a}^{(l)}} [\hat{\boldsymbol{x}}^{(L)}]_{c^*} - [\hat{\boldsymbol{x}}^{(L)}]_t \quad \text{subject to MILP constraints.}$$

That is, for each class $t \neq c^*$, we optimize for the **worst-case margin** m_t^*, and conclude that the classifier is robust if and only if

$$\min_{t \neq c^*} m_t^* \ge 0.$$

However, we can equivalently solve the following single optimization problem:

$$m^* = \min_{\tilde{\boldsymbol{x}},\boldsymbol{y}^{(l)},\hat{\boldsymbol{x}}^{(l)},\boldsymbol{a}^{(l)}} \left([\hat{\boldsymbol{x}}^{(L)}]_{c^*} - y \right) \quad \text{subject to } y = \max_{t \neq c^*} [\hat{\boldsymbol{x}}^{(L)}]_t \wedge \text{MILP constraints},$$

where we have introduced a new variable y into the objective function.

Express the equality constraint

$$y = \max(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{K-1})$$

using only linear and integer constraints. To simplify notation, here $\boldsymbol{x}_k \in \mathbb{R}$ denotes the logit corresponding to the k-th incorrect class, and $\boldsymbol{l}_k$ and $\boldsymbol{u}_k$ its corresponding lower and upper bound.

Hint: You might want to introduce binary variables to indicate which logit is the maximum.

We first define $u_{max} := \max_k \boldsymbol{u}_k$, i.e. the largest upper bound.

Now we introduce the following constraints:

$$\begin{aligned}
& y \leq \boldsymbol{x}_k + (1 - \boldsymbol{b}_k)(u_{max} - \boldsymbol{l}_k) && \forall 1 \leq k \leq K-1 && (1)\\
& y \geq \boldsymbol{x}_k && \forall 1 \leq k \leq K-1 && (2)\\
& \boldsymbol{b}_k \in \{0, 1\} && \forall 1 \leq k \leq K-1 && (3)\\
& \sum_{k=1}^{K-1} \boldsymbol{b}_k = 1 && && (4)
\end{aligned}$$

The last constraint (4) simply ensures that only one element in $\boldsymbol{b}$ is 1 and all others are zero.

The only valid assignment of $\boldsymbol{b}$ is to have $\boldsymbol{b}_k = 1$ for the (unique) maximum value $\boldsymbol{x}_k = \max(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{K-1})$. To see this, consider the case that $\boldsymbol{b}_k = 1$ but $\boldsymbol{x}_k$ is not the maximum value. Then, (1) resolves to $y \leq \boldsymbol{x}_k$. However, for the maximum value $\boldsymbol{x}_{max} > \boldsymbol{x}_k$ we have from (2) $y \geq \boldsymbol{x}_{max}$, leads to a contradiction.

Consider the case $\boldsymbol{b}_k = 1$ and the corresponding value $\boldsymbol{x}_k$ is indeed the (unique) maximum. (1) and (2) imply that $y = \boldsymbol{x}_k$. The remaining values $\boldsymbol{b}_i$ are zero, and in this case we need to show that (1) and (2) are never binding, regardless of the values $\boldsymbol{x}_i$. (2) is not binding since $\boldsymbol{x}_i$ is not the maximum value. (1) is not binding because we have that $\boldsymbol{x}_i + u_{max} - \boldsymbol{l}_i \geq u_{max} \geq y$.

Problem 3: On slide 15 of the robustness chapter, we have defined an optimization problem for untargeted attacks, i.e. we aim to have the sample $\hat{\boldsymbol{x}}$ classified as **any** class other than the correct one:

$$\min_{\hat{\boldsymbol{x}}} \mathcal{D}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \lambda \cdot L(\hat{\boldsymbol{x}}, y)$$

The loss function is defined as:

$$L(\hat{\boldsymbol{x}}, y) = \left[Z(\hat{\boldsymbol{x}})_y - \max_{i \neq y} Z(\hat{\boldsymbol{x}})_i \right]_+,$$

where $[\boldsymbol{x}]_+$ is shorthand for $\max(\boldsymbol{x}, 0)$ and $Z(\boldsymbol{x})_i = \log f(\boldsymbol{x})_i$ (i.e. log probability of class i. Here, $L(\hat{\boldsymbol{x}}, y)$ is positive if $\hat{\boldsymbol{x}}$ is classified correctly and 0 otherwise.

Provide an alternative loss function to turn this attack into a targeted attack, i.e. we aim to have the sample $\boldsymbol{x}$ classified as a *specific* target class t.

$$L(\hat{\boldsymbol{x}}, t) = \left[\max_{i \neq t} Z(\hat{\boldsymbol{x}})_i - Z(\hat{\boldsymbol{x}})_t\right]_+$$

This loss is positive if $\hat{\boldsymbol{x}}$ is classified as a class that is **not** t, and is zero otherwise.

Problem 4: Recall from slide 41 the MILP constraints expressing the ReLU activation function:

$$\begin{aligned}
&y_i \leq x_i - l_i(1 - a_i), \\
&y_i <= a_i \cdot u_i, \\
&y_i \geq x_i, \\
&y_i \geq 0, \\
&a_i \in \{0, 1\},
\end{aligned}$$

where $u_i, l_i \in \mathbb{R}$ are upper and lower bounds on the value of the ReLU input x_i.

Show that – for an unstable unit (i.e. $u_i > 0 \wedge l_i < 0$) – a continuous relaxation on a leads to the convex relaxation constraints on slide 54. That is, replacing the constraint $a_i \in \{0, 1\}$ with $a_i \in [0, 1]$ yields

$$(u_i - l_i)y_i - u_i x_i \leq -u_i l_i.$$

We can combine the first two constraints on slide 41:

$$\begin{aligned}
&y_i \leq x_i - l_i(1 - a_i) \\
&y_i \leq u_i \cdot a_i
\end{aligned}$$

by expressing them as

$$y_i \leq \min(x_i - l_i(1 - a_i), u_i \cdot a_i).$$

Note that we are free to choose any value for a_i between 0 and 1. We want to choose a_i so that it leads to the loosest-possible constraint on y_i, since this leads to the maximum 'leeway' to optimize the objective function. More formally,

$$y_i \leq \max_{a_i} \min(x_i - l_i(1 - a_i), u_i \cdot a_i)$$

Further note that the two terms in the $\min(\cdot, \cdot)$ are two linear functions in a_i. Since $l_i < 0$, the first term is a function with negative slope in a_i. Since $u_i > 0$, the second term in the $\min(\cdot, \cdot)$ is a function with positive slope in a_i.

Consequently, the function $\min(x_i - l_i(1 - a_i), u_i \cdot a_i)$ is maximal at the intersection of the two linear functions. Solving for a_i we get:

$$\begin{aligned}
x_i - l_i(1 - a_i) &= a_i u_i \\
\Leftrightarrow a_i &= \frac{x_i - l_i}{u_i - l_i}
\end{aligned}$$

Plugging the expression of a_i into one of the original constraints, e.g. $y_i \leq a_i \cdot u_i$ we get:

$$\begin{aligned} y_i &\leq \frac{x_i - l_i}{u_i - l_i} u_i \\ \Leftrightarrow y_i(u_i - l_i) - u_i x_i &\leq -u_i l_i, \end{aligned}$$

and therefore we have recovered the constraint of the convex relaxation.

Problem 5: Convex relaxations of non-linearities are not limited to ReLU. For this exercise, we consider the ReLU6 non-linearity

$$\mathrm{ReLU6}(x) = \min(\max(0, x), 6),$$

which is used in MobileNet models performing low-precision computations on mobile devices.

Given input bounds l and u with $l \leq x \leq u$, provide a set of linear constraints corresponding to the convex hull of $\left\{ \begin{pmatrix} x & \mathrm{ReLU6}(x) \end{pmatrix}^T \mid l \leq x \leq u \right\}$.

Hint: You have to make a case distinction over different ranges of l and u.

To faciliate our discussion, we first rewrite the non-linearity as

$$\mathrm{ReLU6}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x < 6 \\ 1 & \text{if } x \geq 6 \end{cases}. \tag{5}$$

We can distinguish six different cases.

If $u < 0$ (and thus also $l < 0$), the non-linearity is always inactive (first case of Eq. 5). Thus, the convex hull is characterized by

$$y = 0.$$

If $l \geq 0$ and $u \leq 0$, we are always in the second case of Eq. 5 and thus

$$y = x.$$

If $l \geq 6$ (and thus also $u \geq 6$), the non-linearity is always saturated (third case of Eq. 5). In this case, we have

$$y = 6.$$

If $l < 0$ and $0 \leq u < 6$, ReLU6 behaves like an unstable ReLU unit. As discussed in lecture 4, the convex hull is a triangle spanned between the vertices $\begin{pmatrix} l & 0 \end{pmatrix}^T$, $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ and $\begin{pmatrix} u & u \end{pmatrix}^T$. It can thus be characterized by three linear constraints corresponding to its faces:

$$\begin{aligned} y &\geq 0 \\ y &\geq x \\ y &\leq \frac{u}{u - l}(x - l). \end{aligned}$$

If $0 \leq l \leq 6$ and $u > 6$, the convex hull is – similar to the previous case – a triangle spanned between vertices $\begin{pmatrix} l & l \end{pmatrix}^T$, $\begin{pmatrix} 6 & 6 \end{pmatrix}^T$ and $\begin{pmatrix} u & 6 \end{pmatrix}^T$. It can also be characterized by three linear constraints corresponding to its faces:

$$\begin{aligned} y &\leq x \\ y &\leq 6 \\ y &\geq l + \frac{6-l}{u-l}(x-l), \end{aligned}$$

with the last constraint corresponding to the line connecting $\begin{pmatrix} l & l \end{pmatrix}^T$ and $\begin{pmatrix} u & 6 \end{pmatrix}^T$.

If $l < 0$ and $u > 6$, the convex hull has vertices $\begin{pmatrix} l & 0 \end{pmatrix}^T$, $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$, $\begin{pmatrix} 6 & 6 \end{pmatrix}^T$ and $\begin{pmatrix} u & 6 \end{pmatrix}^T$. It can be be characterized by the following four linear constraints corresponding to its faces:

$$\begin{aligned} y &\geq 0 \\ y &\leq 6 \\ y &\leq \frac{6}{6-l}(x-l) \\ y &\geq \frac{6}{u}x, \end{aligned}$$

with the third constraint corresponding to the line connecting $\begin{pmatrix} l & 0 \end{pmatrix}^T$ and $\begin{pmatrix} 6 & 6 \end{pmatrix}^T$ and the fourth constraint corresponding to the line connecting $\begin{pmatrix} 0 & 0 \end{pmatrix}^T$ and $\begin{pmatrix} u & 6 \end{pmatrix}^T$.

Randomized smoothing

Problem 6: (*) In the previous exercise we investigated the adversarial robustness of linear classifiers

$$f(x) = \mathrm{I}[\boldsymbol{w}^T \boldsymbol{x} + b > 0]$$

with weight vector $\boldsymbol{w} \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$, mapping samples from $\mathbb{R}^d$ to binary labels $\{0, 1\}$.

Given such a linear classifier f, we can define the randomly smoothed classifier $g : \mathbb{R}^d \mapsto \{0, 1\}$ with

$$g(\boldsymbol{x}) = \mathrm{argmax}_{c \in \{0,1\}} g_c(\boldsymbol{x})$$

and

$$g_c(\boldsymbol{x}) = \Pr_{\boldsymbol{\epsilon}} \left(f(\boldsymbol{x} + \boldsymbol{\epsilon}) = c \right) = \begin{cases} \Pr_{\boldsymbol{\epsilon}} \left(\boldsymbol{w}^T (\boldsymbol{x} + \boldsymbol{\epsilon}) + b \leq 0 \right) & \text{if c} = 0 \\ \Pr_{\boldsymbol{\epsilon}} \left(\boldsymbol{w}^T (\boldsymbol{x} + \boldsymbol{\epsilon}) + b > 0 \right) & \text{else} \end{cases},$$

where $\boldsymbol{\epsilon} \in \mathbb{R}^d$ is a random variable.

For this exercise, we assume that $\boldsymbol{\epsilon}$ follows an isotropic normal distribution, i.e. $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ with elementwise standard deviation $\sigma \in \mathbb{R}_+$.

As discussed in the lecture, evaluating randomly smoothed classifier is typically not tractable and requires sampling. This is however not the case for our simple linear classifier.

Given input $\boldsymbol{x} \in \mathbb{R}^d$, weights $\boldsymbol{w} \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$, show that $g_0(\boldsymbol{x}) = \Phi_{0,1}\left(-\frac{\boldsymbol{w}^T\boldsymbol{x}}{\sigma||\boldsymbol{w}||_2} - \frac{b}{\sigma||\boldsymbol{w}||_2}\right)$, where $\Phi_{0,1} : \mathbb{R} \mapsto [0,1]$ is the cumulative distribution of the standard normal distribution $\mathcal{N}(0,1)$.

Hint: $\Pr_{\boldsymbol{\epsilon}}\left(\boldsymbol{w}^T(\boldsymbol{x}+\boldsymbol{\epsilon}) + b \leq 0\right)$ can alternatively be written as:

$$\int_{\mathbb{R}^d} \mathrm{I}\left[\boldsymbol{w}^T(\boldsymbol{x}+\boldsymbol{\epsilon}) + b \leq 0\right] \mathcal{N}(\boldsymbol{\epsilon} \mid \mathbf{0}, \sigma^2\mathbf{I})\ d\boldsymbol{\epsilon}.$$

We begin by deriving the expression for $g_0(\boldsymbol{x}) = \Pr_{\boldsymbol{\epsilon}}\left(\boldsymbol{w}^T(\boldsymbol{x}+\boldsymbol{\epsilon}) + b \leq 0\right)$. The smoothed output could be calculated by integrating over all possible values of $\boldsymbol{\epsilon}$:

$$g_0(\boldsymbol{x}) = \int_{\mathbb{R}^d} \mathrm{I}\left[\boldsymbol{w}^T(\boldsymbol{x}+\boldsymbol{\epsilon}) + b \leq 0\right] \mathcal{N}(\boldsymbol{\epsilon} \mid \mathbf{0}, \sigma^2\mathbf{I})\ d\boldsymbol{\epsilon}. \tag{6}$$

We can however simplify our derivations by noticing that the value of the indicator function only depends on the value of the scalar random variable

$$z = \boldsymbol{w}^T(\boldsymbol{x}+\boldsymbol{\epsilon}) + b.$$

Any affine transformation $\boldsymbol{A}\boldsymbol{y} + \boldsymbol{c}$ of a multivariate normal random variable $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is also a multivariate normal random variable following distribution $\mathcal{N}(\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{c}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^T)$. Thus, the scalar random variable Z, which is the result of an affine transformation of $\boldsymbol{\epsilon}$ follows a univariate normal distribution:

$$z \sim \mathcal{N}(\mu, \alpha)$$

with

$$\mu = \boldsymbol{w}^T\mathbf{0} + \boldsymbol{w}^T x + b = \boldsymbol{w}^T x + b$$
$$\alpha = \sqrt{\boldsymbol{w}^T\sigma^2\mathbf{I}\boldsymbol{w}} = \sigma\sqrt{\boldsymbol{w}^T\boldsymbol{w}} = \sigma||\boldsymbol{w}||_2.$$

Note that we take the square root since we parameterize the univariate distribution based on its standard deviation, not its variance.

Rather than integrating over the vector $\boldsymbol{\epsilon}$, we can instead integrate over the scalar z:

$$g_0(\boldsymbol{x}) = \int_{-\infty}^{\infty} [z \leq 0]\, \mathcal{N}(z \mid \mu, \alpha)\ dz = \int_{-\infty}^{0} \mathcal{N}(z \mid \mu, \alpha)\ dz \tag{7}$$

From the right-hand side of Eq 7, it becomes clear that we are simply evaluating the cumulative distribution function $\Phi_{\mu,\alpha}$ of $\mathcal{N}(\mu, \alpha)$, i.e.

$$g_0(\boldsymbol{x}) = \Phi_{\mu,\alpha}(0).$$

The distribution $\mathcal{N}(\mu, \alpha)$ is a standard normal distribution $\mathcal{N}(0,1)$ that is translated by μ and scaled by a factor α. Thus, we can alternatively write

$$g_0(\boldsymbol{x}) = \Phi_{0,1}((0-\mu)/\alpha) = \Phi_{0,1}\left(-\frac{\boldsymbol{w}^T\boldsymbol{x}}{\sigma||\boldsymbol{w}||_2} - \frac{b}{\sigma||\boldsymbol{w}||_2}\right),$$

where the last equality simply follows from the definition of μ and α.

Randomized smoothing for discrete data

For the sake of simplicity, we consider a slightly different setup than in the lecture. In this exercise, we assume no knowledge about $f_\theta(\mathbf{x})$ respectively $g(\mathbf{x})_c$ (usually we would estimate a lower bound of $g(\mathbf{x})_c$ via Monte Carlo sampling, but here we do not).

We use the same sparsity-aware randomization scheme $\phi(\mathbf{x})$ as in the lecture:

$$g(\mathbf{x})_c = \mathcal{P}(f(\phi(\mathbf{x})) = c) = \sum_{\tilde{\mathbf{x}} \text{ s.t. } f(\tilde{\mathbf{x}})=c} \prod_{i=1}^{n^2} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i) \tag{8}$$

with

$$\mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i) = \begin{cases} p_d^{\mathbf{x}_i} p_a^{1-\mathbf{x}_i} & \tilde{\mathbf{x}}_i = 1 - \mathbf{x}_i \\ (1-p_d)^{\mathbf{x}_i}(1-p_a)^{1-\mathbf{x}_i} & \tilde{\mathbf{x}}_i = \mathbf{x}_i \end{cases} \tag{9}$$

and the number of nodes n. For an illustration we refer to Slide 15 "Smoothed Classifier for Discrete Data"

Problem 7: (*) Given an arbitrary graph $\mathbf{x}$, and a perturbed one $\mathbf{x}'$ where $\mathbf{x}'$ differs from $\mathbf{x}$ in exactly one edge. What is the worst-case base classifier $h^*(\mathbf{x})$? In this context, we refer to the worst-case base classifier $h^*(\mathbf{x})$ as the classifier that has the largest drop in classification confidence between $g(\mathbf{x})_c$ and $g(\mathbf{x}')_c$. Or in other words, $h^*(\mathbf{x})$ results in the most instable smooth classifier if we switch a single edge. This motivates the importance of analyzing robustness for graph neural networks (or other models with discrete input data).

The classifier with the *largest drop in classification accuracy between* $g(\mathbf{x})_c$ *and* $g(\mathbf{x}')_c$ can be formalized as a minimization problem $h^*(\mathbf{x}) = \arg\min_{h(\mathbf{x})\in\mathcal{H}} g(\mathbf{x}')_c - g(\mathbf{x})_c$. In the following we consider a random order of edges and hence we may assume w.l.o.g. that all edges are identical but the last edge. Hence, from (8) it follows:

$$
\begin{aligned}
\min_{h(\mathbf{x})\in\mathcal{H}} g(\mathbf{x}')_c - g(\mathbf{x})_c &= \min_{h(\mathbf{x})\in\mathcal{H}} \left(\sum_{\tilde{\mathbf{x}} \text{ s.t. } h(\tilde{\mathbf{x}})=c} \prod_{i=1}^{n^2} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}'_i) \right) - \left(\sum_{\tilde{\mathbf{x}} \text{ s.t. } h(\tilde{\mathbf{x}})=c} \prod_{i=1}^{n^2} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i) \right) \\
&= \min_{h(\mathbf{x})\in\mathcal{H}} \sum_{\substack{\tilde{\mathbf{x}} \text{ s.t.} \\ h(\tilde{\mathbf{x}})=c}} \left[\left(\prod_{i=1}^{n^2-1} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}'_i) \right) \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}'_{n^2}) - \left(\prod_{i=1}^{n^2-1} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i) \right) \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}_{n^2}) \right] \\
&= \min_{h(\mathbf{x})\in\mathcal{H}} \sum_{\tilde{\mathbf{x}} \text{ s.t. } h(\tilde{\mathbf{x}})=c} \left(\prod_{i=1}^{n^2-1} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i) \right) \underbrace{\left(\mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}'_{n^2}) - \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}_{n^2}) \right)}_{\Delta_{\tilde{\mathbf{x}}}}
\end{aligned}
$$

$\Delta_{\tilde{\mathbf{x}}} = \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}'_{n^2}) - \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}_{n^2})$ resolves to two cases (each case occurs 50% of the time): (1) $1-(p_a+p_d)$ and (2) $p_a + p_d - 1$. To minimize $g(\mathbf{x}')_c - g(\mathbf{x})_c$ we now choose $h^*(\mathbf{x})$ to predict c for all cases where $\Delta_{\tilde{\mathbf{x}}} < 0$ (assuming $p_a + p_d \neq 1$). Hence, $\Delta = \Delta_{\tilde{\mathbf{x}}}$ for $\tilde{\mathbf{x}}$ s.t. $h(\tilde{\mathbf{x}}) = c$.

We conclude the worst-case base classifier $h^*(\mathbf{x})$ exactly classifies exactly 50% of the random graphs $\tilde{\mathbf{x}}$ with c (note that in the general case $g(\mathbf{x})_c \neq 1/2$). In the case where one edge is removed from $\mathbf{x}'$ (relatively to $\mathbf{x}$) and $p_a + p_d < 1$, the worst case base classifier $h^*(\mathbf{x})$ predicts c for all graphs where this edge is not missing (e.g. $h^*(\mathbf{x}) = c$ and $h^*(\tilde{\mathbf{x}}) \neq c$).

Problem 8: (*) How many of the possible graphs $\tilde{\mathbf{x}}$ does the worst-case base classifier assign the label c (see Problem 7)? To be more specific, we are looking for a term reflecting the absolute number and not a ratio?

Since we have n^2 edges there are 2^{n^2} possible adjacency matrices (each adjacency matrix represents one graph). Since we predict 50% with class c, we have a total of $2^{n^2}/2 = 2^{n^2-1}$ graphs resulting in c.

This clearly shows that enumerating all possible $\tilde{\mathbf{x}}$ is infeasible also for very small graphs.

Problem 9: What is $g(\mathbf{x}')_c$, $g(\mathbf{x})_c$, and $g(\mathbf{x}')_c - g(\mathbf{x})_c$ for the worst-case base classifier $h^*(\mathbf{x})$ (see Problem 1)? Please derive the equations (given $p_a + p_d < 1$). Subsequently, we would like to know the precise values for $p_a = 0.001$ and $p_d = 0.1$.

Since $p_a + p_d < 1$ we conclude that $\Delta = p_a + p_d - 1$.

$$\begin{aligned}
\min_{h(\mathbf{x})\in\mathcal{H}} g(\mathbf{x}')_c - g(\mathbf{x})_c &= \min_{h(\mathbf{x})\in\mathcal{H}} \sum_{\tilde{\mathbf{x}} \text{ s.t. } h(\tilde{\mathbf{x}})=c} \left(\prod_{i=1}^{n^2-1} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i)\right) \underbrace{\left(\mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}'_{n^2}) - \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}_{n^2})\right)}_{\Delta} \\
&= \min_{h(\mathbf{x})\in\mathcal{H}} \Delta \underbrace{\sum_{\tilde{\mathbf{x}} \text{ s.t. } h(\tilde{\mathbf{x}})=c} \prod_{i=1}^{n^2-1} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i)}_{=1} \\
&= p_a + p_d - 1
\end{aligned}$$

Please note that $\sum_{\tilde{\mathbf{x}} \text{ s.t. } h(\tilde{\mathbf{x}})=c} \prod_{i=1}^{n^2-1} \mathcal{P}(\tilde{\mathbf{x}}_i|\mathbf{x}_i)$ can be understood as a sum over the entire sample space of a product of (n^2-1) Bernoulli random variables (i.e. sum over all possible combinations). Due to the basic laws of probability it must sum up to one.

Using $\Delta = \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}'_{n^2}) - \mathcal{P}(\tilde{\mathbf{x}}_{n^2}|\mathbf{x}_{n^2})$ s.t. $h^*(\tilde{\mathbf{x}}) = c$, we can easily go back and fourth between $g(\mathbf{x}')_c$, $g(\mathbf{x})_c$, and $g(\mathbf{x}')_c - g(\mathbf{x})_c$. Consequently, the the worst-case base classifier, with the given flip probabilities $p_a = 0.001$ and $p_d = 0.1$, has the following probabilities:

- $g(\mathbf{x}')_c = p_a = 0.001$
- $g(\mathbf{x})_c = 1 - p_d = 0.9$
- $g(\mathbf{x}')_c - g(\mathbf{x})_c = p_a + p_d - 1 = -0.899$

Please acknowledge that a smooth classifier might predict the right class c with high probability $g(\mathbf{x})_c = 1 - p_d = 0.9$, but flipping a single edge can result in $g(\mathbf{x}')_c = p_a = 0.001$. Hence, the probability of the smooth classier drops by around 90%.