SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 479

# Pronalaženje sustavnih pogrešaka označavanja u podatcima za učenje

Luka Družijanić

Zagreb, srpanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# DIPLOMSKI ZADATAK br. 597

| | |
|---|---|
| Pristupnik: | **Luka Družijanić (0036522126)** |
| Studij: | Računarstvo |
| Profil: | Računarska znanost |
| Mentor: | prof. dr. sc. Siniša Šegvić |

Zadatak: **Pronalaženje sustavnih pogrešaka označavanja u podatcima za učenje**

Opis zadatka:

Klasificiranje slika važan je zadatak računalnog vida s mnogim zanimljivim primjenama. U posljednje vrijeme vrlo zanimljive rezultate postižu duboki modeli zasnovani na konvolucijama i slojevima pažnje. Međutim, veliki kapacitet tih modela omogućava kibernetičke napade zasnovane na trovanju podataka za učenje. U okviru rada, potrebno je odabrati okvir za diferencijabilno programiranje te upoznati biblioteke za rukovanje matricama i slikama. Proučiti i ukratko opisati postojeće arhitekture utemeljene na konvolucijama i pažnji. Odabrati klasifikacijski model te uhodati njegovo učenje na nekom javno dostupnom skupu podataka. Uhodati standardne kibernetičke napade i obrane zasnovane na trovanju podataka. Prikazati i ocijeniti uspješnost obrana te predložiti poboljšanja zasnovana na robusnom učenju i pronalaženju sustavnih pogrešaka označavanja. Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, kao i potrebna objašnjenja te dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 28. lipnja 2024.

*Zahvaljujem se prof.dr.sc. Siniši Šegviću i Ivanu Saboliću na pruženoj pomoći i znanju tijekom pisanja ovog rada. Također se zahvaljujem Lari, roditeljima i prijateljima na podršci tijekom cijelog studija.*

# Sadržaj

# 1. Uvod

Duboki modeli su danas ključni u raznim domenama, od računalnog vida i generiranja slika, do razumijevanja govora i prevođenja teksta. Međutim, kako bismo postigli zadovoljavajuće performanse, duboki modeli zahtijevaju velike količine podataka i parametara. Potrebno je puno procesorske snage i vremena za treniranje takvih modela.

Mnogi odlučuju dijelove učenja modela prepustiti drugima. Primjerice, umjesto da sami sakupljamo podatke, preuzet ćemo javno dostupan skup podataka. Možda učenje možemo započeti od tuđeg, već naučenog modela, te ga samo prilagoditi za naš zadatak. U slučaju da nemamo dovoljno jaku grafičku karticu za učenje našeg modela, odlučit ćemo se za računarstvo u oblaku.

Ovakvi procesi sadrže sigurnosne ranjivosti. Skup podataka koji preuzimamo može imati tisuće slika, a model koji preuzimamo milijune parametara - ne možemo znati što smo točno dobili. Možemo testirati model na zadatku za koji je namijenjen, no moguće je i da model obavlja neki drugi, skriveni zadatak. Takvu funkcionalnost modela nazivamo stražnja vrata (engl. *backdoor*), koja se uobičajeno iskorištava skrivenim uzorkom, *okidačem*, na ulaznom podatku. U većini slučajeva, stražnja vrata vrlo je teško ili nemoguće primijetiti bez zaključivanja na podacima s okidačem. Model sa stražnjim vratima mogao bi, primjerice, u prisutnosti određenog uzorka na slici krivo klasificirati primjer u specifičan razred.

Ugrađivanje stražnjih vrata u model se uobičajeno provodi učenjem na *zatrovanim* podacima. Zatrovani podaci na sebi sadrže okidač, te je njihova oznaka izmijenjena u *ciljni razred*. Kako bi napad ostao sakriven u skupu podataka, te kako bi model i dalje mogao dobro naučiti originalni zadatak, samo mali dio skupa za učenje je otrovan, oko 1% do 10%. Uz to, okidači su vrlo maleni, teško primjetni uzorci.

Postojeće metode obrane se mogu ugrubo podijeliti u tri kategorije:

– Obrana prije učenja - obrana se provodi prije učenja modela, samo na podacima za učenje. Cilj je detektirati otrovane primjere, te ukloniti primjere

iz skupa podataka, ukloniti okidač s primjera, ili popraviti oznaku primjera. Otrovane primjere možemo detektirati promatranjem samonadziranih reprezentacija primjera [17].

- Obrana tijekom učenja - obrana nastoji spriječiti učenje stražnjih vrata iz zatrovanih podataka. Primjerice, ABL [13] iskorištava činjenicu da model puno brže nauči zatrovane primjere, pa ih detektira po vrijednostima funkcije gubitka u ranim epohama učenja.
- Obrana nakon učenja - cilj je detektirati sadrži li dani model stražnja vrata, te potencijalno i ukloniti ili umanjiti stražnja vrata. Neural Cleanse [16] pokušava rekonstruirati okidač analiziranjem izlaza modela za pojedine kandidate okidača. Activation Clustering [4] detektira otrovane primjere promatranjem aktivacija modela.

U 2. poglavlju opisat ćemo nekoliko postojećih metoda napada na duboke modele, te u 3. poglavlju neke postojeće ideje kako se obraniti. U 4. poglavlju detaljno ćemo opisati kako radi obrana temeljena na samonadziranim učenju, te naš prijedlog kako ju poboljšati koristeći smanjenje dimenzionalnosti i grupiranje. Konačno, u 5. poglavlju, validirat ćemo našu metodu, provjeriti kako radi protiv napada, te usporediti ju s ostalim obranama.

# 2. Metode napada

## 2.1. BadNets

Pretpostavka napada je da napadač ima kontrolu barem nad skupom podataka za učenje, $D_{train} = \{(\boldsymbol{x_i}, y_i)\}$, gdje je $\boldsymbol{x_i} \in [0,1]^d$ i $y_i \in \{1, ..., C\}$. Napadač smije izmijeniti skup podataka za učenje na proizvoljan način kako bi umetnuo stražnja vrata. Žrtva će provjeriti točnost naučenog modela na skupu podataka za testiranje, $D_{test}$, stoga napadač mora osigurati da model s ugrađenim stražnjim vratima i dalje postiže dobre rezultate na čistim podacima.

Cilj BadNets [7] napada je ugraditi stražnja vrata koja se aktiviraju jednostavnim okidačem. U prisustvu odabranog okidača, model klasificira svaki primjer u ciljani razred. Kako bi ovo postigao, napadač uzima podskup $D' \in D_{train}$ koji sadrži do $p_a$ (uobičajeno oko 10%) podataka, te iz njega generira zatrovani skup $D'_p$:

$$D'_p = \{(A(\boldsymbol{x_i}, \boldsymbol{m}, \boldsymbol{p}), y_t) \mid (\boldsymbol{x_i}, y_i) \in D'\} \tag{2.1}$$

U prikazanoj jednadžbi, $y_t$ označava ciljni razred u koji se klasificiraju svi podaci sa okidačem, a $A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{p})$ predstavlja funkciju koja na dani $\boldsymbol{x}$ postavlja okidač definiran parametrima $\boldsymbol{m}$ i $\boldsymbol{p}$ gdje je $\boldsymbol{m} \in [0,1]^d$ maska koja određuje poziciju okidača, a $\boldsymbol{p} \in [0,1]^d$ uzorak okidača:

$$A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{p}) = (1 - \boldsymbol{m}) \cdot \boldsymbol{x} + \boldsymbol{m} \cdot \boldsymbol{p} \tag{2.2}$$

Konačno, napadač uči klasifikacijski model $f(\boldsymbol{x}, \boldsymbol{\Theta})$ na otrovanom skupu podataka $(D \setminus D') \cup D'_p$. Slika 2.1a prikazuje primjer BadNets okidača kojeg ćemo koristiti u eksperimentima.

Okidači će uobičajeno biti vrlo maleni uzorci, jer je cilj učiniti ih neprimjetnima. Razmatrat ćemo slučaj kada imamo jedan okidač za jedan ciljni razred, no mogući su slučajevi sa više okidača za isti razred ili sa više ciljnih razreda, svaki sa svojim

okidačima.

**(a)** Primjer BadNets okidača **(b)** Originalna slika **(c)** Slika sa BadNets okidačem u donjem desnom kutu

**Slika 2.1:** Primjer BadNets napada - okidač (a) se dodaje u kut originalne slike (b). Otrovani primjer prikazan je na posljednjoj slici.

## 2.2. WaNet

Umjesto "nalijepljenih" okidača koji direktno mijenjaju piksele na vidljiv način, cilj napada WaNeta je dizajnirati okidače koje će čovjek vrlo teško primijetiti. WaNet to postiže s deformacijskim poljima. Funkcija trovanja je sada:

$$A(\boldsymbol{x}) = \mathcal{W}(\boldsymbol{x}, \mathbf{M}) \tag{2.3}$$

$\mathcal{W}$ je operacija deformiranja (engl. *warping*) koja generira novu sliku uzorkovanjem piksela stare slike s posmaknutih lokacija. $\mathbf{M}$ je deformacijsko polje koje definira koji piksel stare slike se uzorkuje za generiranje piksela nove slike. Dopuštamo i uzorkovanje necjelobrojnih koordinata slike, te u tom slučaju bikubično interpoliramo vrijednost piksela.

Za uspješnost i nevidljivost napada ključno je generirati dobar $\mathbf{M}$. Deformacija bi trebala biti slaba i glatka kako bi bila neprimjetna. Također, deformacija ne bi trebala uzorkovati točke izvan granica slike.

Kako bismo osigurali glatkoću deformacije, prvo nasumično generiramo manje deformacijsko polje $\mathbf{P}$ veličine $k \times k \times 2$, koje ćemo kasnije naduzorkovati na potrebnu veličinu:

$$\mathbf{P} = \psi(rand_{[-1,1]}(k, k, 2)) \times s \tag{2.4}$$

Funkcija $rand_{[-1,1]}(k, k, 2)$ vraća tensor oblika $k \times k \times 2$ sa vrijednostima između $-1$ i 1, $s$ označava jakost deformacijskog polja, a $\psi$ je funkcija koja normalizira snagu deformacije:

$$\psi(\mathbf{A}) = \frac{\mathbf{A}}{\frac{1}{size(\mathbf{A})} \sum_{a_i \in \mathbf{A}} |a_i|} \tag{2.5}$$

Sada bikubičnom interpolacijom naduzorkujemo $\mathbf{P}$ na potrebnu veličinu $h \times w \times 2$, što označavamo sa $\uparrow$, te primjenjujemo funkciju podrezivanja (engl. *clipping*) $\phi$ koja osigurava da nijedna točka uzorkovanja ne pada izvan slike. Konačno deformacijsko polje M dobivamo prema sljedećem izrazu:

$$\mathbf{M} = \phi(\uparrow (\mathbf{P})) \tag{2.6}$$

Kao i u BadNets napadu, ova deformacija se dodaje na $p_a$ primjera iz skupa za učenje, te se njihova oznaka mijenja u ciljnu oznaku $y_t$. WaNet uvodi još jedno poboljšanje u proces napada - osim trovanja $p_a$ primjera dodavanjem okidača i promjenom ciljnog razreda, dodatno se na deformacijsko polje za $p_n$ primjera dodaje i slučajan šum bez promjene ciljnog razreda. Poanta ovog šuma je otežati detekciju napada, budući da će sada model morati naučiti samo točnu deformaciju kao odgovarajući okidač za stražnja vrata. Točnije, za $p_n$ primjera definira se deformacija:

$$\mathbf{M}' = \mathbf{M} + rand_{[-1,1]}(h, w, 2) \tag{2.7}$$

Sljedeća jednadžba prikazuje konačan napad WaNet:

$$(\boldsymbol{x}, y) \mapsto \begin{cases} (\boldsymbol{x}, y) & \text{za } 1 - p_a - p_n \text{ za primjera} \\ (\mathcal{W}(\boldsymbol{x}, \mathbf{M}), y_t) & \text{za } p_a \text{ za primjera} \\ (\mathcal{W}(\boldsymbol{x}, \mathbf{M}'), y) & \text{za } p_n \text{ za primjera} \end{cases} \tag{2.8}$$

WaNet je itekako skriveniji od napada BadNets, no BadNets je praktičniji za primjene u fizičkom svijetu. Primjerice, ako je zadatak modela prepoznavanje prometnih znakova, napadač naučeni okidač može fizički zalijepiti na znakove u prometu, dok se WaNet deformacija može samo računalno unijeti. Učenje napada WaNet je također puno zahtjevnije od BadNets - modeli bi mogli imati slabije performanse na čistim podacima, ili bi mogli zahtijevati više epoha i podataka da dostignu iste performanse.

**(a)** Originalna slika **(b)** Slika s WaNet deformacijom **(c)** Razlika slike s WaNet deformacijom i originalne slike. Ova razlika će biti drukčija za svaku sliku.

**Slika 2.2**

## 2.3. SIG

Mana BadNets i WaNet napada je što mijenjaju oznake primjera u skupu za učenje. Ovakvi primjeri mogu biti pronađeni ako žrtva krene ručno provjeravati skup podataka, čak i ako su sami okidači potpuno neprimjetni.

Tijekom učenja, napad SIG [3] mijenja samo primjere ciljne klase $y_t$. Okidač je, slično kao i u BadNets napadu, samo uzorak dodan na podatak. Napad možemo prikazati sljedećom funkcijom:

$$(\boldsymbol{x}, y) \mapsto \begin{cases} (\boldsymbol{x}, y) & \text{za } y \neq y_t \\ (\boldsymbol{x}, y) & 1 - p_a \text{ primjera kojima } y = y_t \\ (A(\boldsymbol{x}, \boldsymbol{p}), y_t) & \text{za } p_a \text{ primjera kojima } y = y_t \end{cases} \tag{2.9}$$

U slučaju BadNetsa i WaNeta, promjena razreda je bila veliki signal za model, kojemu je okidač bio jedini način da nauči ispravno mapiranje otrovanih primjera u ciljnu klasu. Međutim, na otrovanim primjerima model u slučaju SIG-a u isto vrijeme vidi i stvarnu sliku ciljnog razreda i okidač, pa uzorak mora biti dovoljno jak da uvjerimo mrežu u okidač. Stoga uzorak $p$ sada "lijepimo" preko cijele slike:

$$A(\boldsymbol{x}, \boldsymbol{p}) = \alpha \cdot \boldsymbol{x} + (1 - \alpha) \cdot \boldsymbol{p} \tag{2.10}$$

Skalar $\alpha$ je hiperparametar koji određuje koliko je okidač $\boldsymbol{p}$ vidljiv - što je $\alpha$ manji, više vidimo sliku i manje okidač. Okidač $\boldsymbol{p}$ definiramo kao horizontalni

sinusoidalni signal veličine kao i sama slika:

$$p_{ij} = \Delta \sin(2\pi j f / m), \tag{2.11}$$

gdje je $p_{ij}$ piksel okidača $\boldsymbol{p}$ u retku $i$ i stupcu $j$, $m$ je broj stupaca (širina) slike, a $\Delta$ i $f$ hiperparametri s kojima definiramo amplitudu i frekvenciju oscilacija.

Jednom kada je napad naučen, model bi tijekom testiranja trebao detektirati okidač čak i kada je on na primjeru drugog razreda.

**(a)** Originalna slika **(b)** Slika sa SIG okidačem

**Slika 2.3**

# 3. Metode obrane

## 3.1. Neural Cleanse

Cilj obrane Neural Cleanse [16] je detekcija razreda na koje se stražnja vrata odnose te rekonstrukcija okidača za dani naučeni model $f(\boldsymbol{x})$.

Glavna pretpostavka obrane je da će okidači biti vrlo maleni uzorci na podacima. Drugim riječima, ako imamo primjer bez okidača koji pripada razredu $y \neq y_t$, onda je potrebna vrlo malena promjena primjera kako bi ga model krivo klasificirao u razred $y_t$. Neural Cleanse pokušava izmjeriti veličinu te promjene za svaki od razreda, te tako detektirati razred na kojeg se odnose stražnja vrata. Ova pretpostavka nije istinita za svaki napad, primjerice WaNet okidači su deformacije po cijeloj slici, pa na takvim napadima Neural Cleanse neće uspjeti.

Za svaki od razreda, Neural Cleanse pokušava pronaći neki okidač $(\boldsymbol{m}, \boldsymbol{p})$ koji će, kada je primijenjen na bilo koji čisti primjer $\boldsymbol{x_i}$, uzrokovati da model $f$ klasificira taj primjer u razred $c$. Takav okidač bi imao vrlo malen ukupni gubitak $\mathcal{L}$:

$$\mathcal{L} = \sum_{\boldsymbol{x_i}} \ell(c, f(A(\boldsymbol{x_i}, \boldsymbol{m}, \boldsymbol{p}))) \tag{3.1}$$

Prikazana jednadžba pretpostavlja da je $\ell$ neka standardna funkcija gubitka, primjerice unakrsna entropija. Gubitak $\mathcal{L}$ će biti minimalan ako je okidač $(\boldsymbol{m}, \boldsymbol{p})$ dovoljan da se svaki čisti primjer $\boldsymbol{x_i}$ klasificira u razred $c$. Primijetimo da ovakav okidač uvijek postoji - trivijalan primjer bi bio "okidač" koji je zapravo neki $\boldsymbol{x_i}$ iz skupa podataka koji model ispravno klasificira u razred $c$. Primjerice, ako tražimo okidače koji će natjerati model da svaki znak klasificira kao "stop", onda je sam znak "stop" validan "okidač".

Očito, takvi veliki "okidači" nas ne zanimaju jer ne podržavaju hipotezu da su podatci zatrovani. Neural Cleanse pokušava pronaći što manje okidače $(\boldsymbol{m}, \boldsymbol{p})$ koji daju što manji gubitak $\mathcal{L}$. Točnije, cilj Neural Cleansea je riješiti sljedeći

optimizacijski problem:

$$\boldsymbol{m}, \boldsymbol{p} = \arg\min_{\boldsymbol{m},\boldsymbol{p}} \left[ \sum_{\boldsymbol{x_i}} \ell(c, f(A(\boldsymbol{x_i}, \boldsymbol{m}, \boldsymbol{p}))) + \lambda \cdot |\boldsymbol{m}|_1 \right] \tag{3.2}$$

Pri tome je $|\boldsymbol{m}|_1$ L1 norma maske, a $\lambda$ je hiperparametar koji određuje koliko jako kažnjavamo velike maske.

Uz pretpostavku da nam je dostupna definicija modela $f$ i sve njegove težine, izraz (3.2) sada možemo riješiti nekim gradijentnim postupkom kao što je Adam [11]. Optimizaciju rješavamo za svaki razred $c$, čime dobivamo rekonstruirane potencijalne okidače za svaki razred. Rekonstruirani okidač za razred na kojeg se stražnja vrata odnose bi trebao imati značajno manju L1 normu maske od ostalih.

Nakon što smo izračunali potencijalne okidače za svaki razred, koristimo neku od metoda za detekciju stršećih vrijednosti nad nizom L1 normi maski. Neural Cleanse koristi srednju apsolutnu devijaciju (engl. *Median Absolute Deviation, MAD*), čime konačno detektiramo razred koji je povezan sa stražnjim vratima.

Jednom kad imamo rekonstruiran približno točan okidač, možemo pokušati i ukloniti stražnja vrata tako da na dio podataka za učenje ipak primijenimo okidač, ali ostavimo ispravnu oznaku podatku. Model bi sada na ovakvom skupu podataka trebao naučiti predviđati ispravne oznake i u prisustvu okidača te tako "zatvoriti" stražnja vrata.

**(a)** BadNets okidač za razred 0

**(b)** Okidač kojeg je NC rekonstruirao za razred 0

**(c)** Okidač kojeg je NC rekonstruirao za razred 1

**Slika 3.1**

## 3.2. Activation Clustering

Iako primjere sa umetnutim okidačem model klasificira u isti razred kao i primjere koji stvarno pripadaju tom razredu, razlog zašto ih model pridjeljuje razredu je drukčiji. Za primjere bez okidača, model uočava značajke koje je naučio da su karakteristične za taj razred. S druge strane, za primjere s okidačem model uočava i značajke izvornog razreda i značajke okidača. Activation Clustering (AC) [4] se zasniva na ideji da bi se ta razlika u zaključivanju trebala vidjeti i u aktivacijama modela.

Pretpostavka Activation Clustering algoritma je da imamo dostupan skup podataka, potencijalno zatrovan, na kojem je model treniran. Modelom klasificiramo sve primjere i sakupimo aktivacije zadnjeg sloja modela. Ideja AC-a je da, za svaki od razreda u koji su primjeri klasificirani, grupiramo aktivacije modela. Ako neki razred nema ugrađena stražnja vrata, tada bi aktivacije trebale biti međusobno slične među svim primjerima. Ako neki razred ima ugrađena stražnja vrata, tada bi aktivacije trebale biti grupirane u dvije različite grupe - jedna grupa aktivacija za otrovane primjere, druga grupa za obične primjere.

Algoritmi grupiranja često loše rade na visoko-dimenzionalnim podacima [1]. Stoga prvo koristimo neki algoritam smanjenja dimenzionalnosti poput metode analize nezavisnih komponenti (engl. *Independent Component Analysis, ICA*). Nadalje koristimo algoritam k-sredina s $k = 2$, kojeg ćemo detaljnije još upisati u poglavlju 4.3. Algoritam k-sredina s $k = 2$ će nam uvijek vratiti dvije grupe. Stoga svakako nakon grupiranja moramo još odrediti je li ijedna grupa otrovana i, ako je - koja.

Prva predložena metoda je da treniramo novi model bez jedne od grupa i onda klasificiramo tu grupu novim modelom. Ako je grupa sadržavala otrovane primjere, novi model će te primjere klasificirati drugačije od staroga. S druge strane, ako grupa nije sadržavala otrovane primjere, novi model će klasificirati primjere skoro jednako kao i stari. Međutim, ova metoda je vrlo skupa budući da moramo trenirati nove modele.

Druga opcija je gledati samo veličine grupa. Ako nema otrovanih podataka za neki razred, očekujemo slične veličine grupa. Ako je model učen sa skupom podataka koji sadrži $p \cdot N$ otrovanih podataka, gdje je $N$ broj svih podataka a $p < 1$, tada će jedna grupa imati otprilike $p \cdot N$ podataka, a druga $(1 - p) \cdot N$ podataka. Uobičajeno je $p$ malen broj, do 10%, stoga ako je jedna grupa puno manja od druge, vjerojatno je riječ o stražnjim vratima. Jednom kada smo

detektirali otrovane primjere, izbacujemo ih iz skupa te ponovno učimo model.

# 4. Čišćenje skupa podataka samonadziranim učenjem

## 4.1. Samonadzirano učenje - SimCLR

Tijekom napada, promjena oznake u ciljni razred nosi veliki dio informacije, dok je sami okidač često samo malena izmjena na primjeru. Stoga, kao dio obrane, možemo koristiti samonadzirane metode, koje tijekom učenja ne uzimaju u obzir oznake primjera. U ovu svrhu, razmotrit ćemo SimCLR [5], okvir za samonadzirano učenje reprezentacija, te ćemo pokušati detektirati otrovane primjere analizirajući prostor reprezentacija.

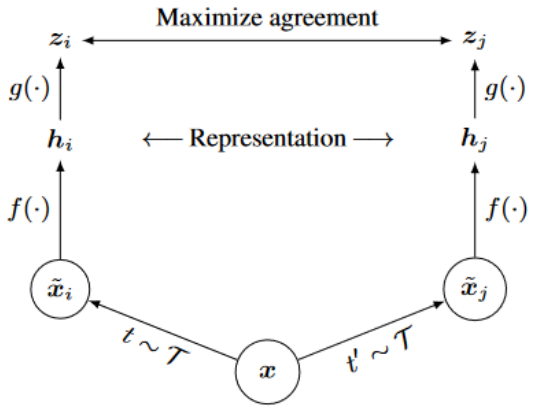
Cilj SimCLR-a je naučiti primjerima dodjeljivati latentne reprezentacije takve da su slični primjeri bliski u prostoru reprezentacije, a različiti primjeri udaljeni. Takve reprezentacije SimCLR uči maksimizirajući sličnost reprezentacija različitih pogleda istog primjera.

Tijekom učenja, za dani primjer $\boldsymbol{x}$ na ulazu, slučajnim deformacijama $t$ i $t'$ generiraju se dva pogleda na taj primjer, $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$ i $\tilde{\boldsymbol{x}}'_{\boldsymbol{i}}$. SimCLR slijedno primjenjuje sljedeće deformacije: slučajno obrezivanje i naduzorkovanje na originalnu veličinu, slučajne distorzije boje, te slučajan Gaussov šum.

Nadalje, okosnica $f(\cdot)$ iz pogleda računa njihove reprezentacije, $\boldsymbol{h}_{\boldsymbol{i}}$ i $\boldsymbol{h}'_{\boldsymbol{i}}$. U eksperimentima ćemo za okosnicu koristiti ResNet-18 [9], od kojeg uzimamo izlaz iz sloja sažimanja prosjekom, prije posljednjeg potpuno povezanog sloja. Ove reprezentacije su onda 512-dimenzionalni vektori.

Na reprezentacije $\boldsymbol{h}_{\boldsymbol{i}}$ i $\boldsymbol{h}'_{\boldsymbol{i}}$ zatim primjenjujemo manju projekcijsku glavu $g(\cdot)$ koja ih preslikava u manje, 128-dimenzionalne reprezentacije $\boldsymbol{z}_{\boldsymbol{i}}$ i $\boldsymbol{z}'_{\boldsymbol{i}}$. Za projekcijsku glavu koristimo potpuno povezanu neuronsku mrežu sa jednim skrivenim slojem veličine 512:

$$\boldsymbol{z}_{\boldsymbol{i}} = g(\boldsymbol{h}_{\boldsymbol{i}}) = W_2 \cdot \sigma(W_1 \cdot \boldsymbol{h}_{\boldsymbol{i}}), \tag{4.1}$$




**Slika 4.1:** Prikaz SimCLR modela. Originalni primjer $\boldsymbol{x}$ se deformira slučajnim deformacijama $t$ i $t'$, na dobivene poglede $\tilde{\boldsymbol{x}}_i$ i $\tilde{\boldsymbol{x}}'_i$ primjenjuje se okosnica $f(\cdot)$. Izlazi iz okosnice, $\boldsymbol{h_i}$ i $\boldsymbol{h}'_i$, su reprezentacije primjere, no pri učenju optimiramo razliku reprezentacija $\boldsymbol{z_i}$ i $\boldsymbol{z}'_i$ dobivene iz jednostavne projekcijske glave $g(\cdot)$. Slika preuzeta iz [5].

gdje je $\sigma$ aktivacijska funkcija ReLU. Pokazuje se da su $\boldsymbol{z_i}$ reprezentacije pogodnije za optimiranje gubitka, međutim za ostale zadatke strojnog učenja bolje su reprezentacije $\boldsymbol{h_i}$ i mi ih dalje koristimo u našoj metodi.

Učenje se odvija u grupama veličine $N$. Za svaki od $N$ primjera, generiramo dvije reprezentacije $\boldsymbol{z_i}$, stoga ukupno imamo $2N$ reprezentacija u grupi. Dvije reprezentacija nastale od istog primjera tretiramo kao jedan pozitivan par, te ostalih $2(N-1)$ primjera u grupi smatramo negativnima. Ako je $\text{sim}(\boldsymbol{u}, \boldsymbol{v})$ kosinusna sličnost primjera:

$$\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u}^\top \boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|}, \tag{4.2}$$

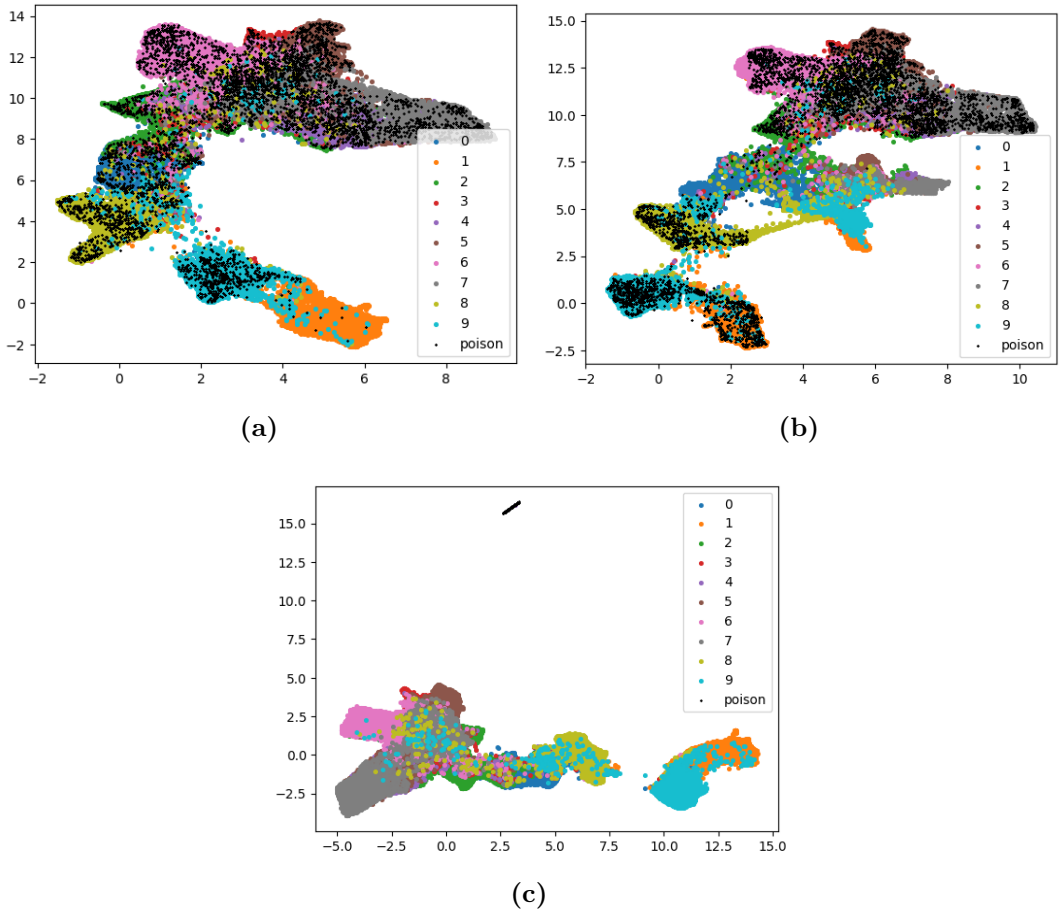
tada definiramo funkciju gubitka za pozitivan par primjera s indeksima $(i, j)$ kao:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z_i}, \boldsymbol{z_j})/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z_i}, \boldsymbol{z_k})/\tau)}, \tag{4.3}$$

gdje je $\mathbb{1}_{[k \neq i]} \in \{0, 1\}$ indikatorska funkcija koja je jednaka 1 samo ako je $k \neq i$, a $\tau$ je hiperparametar temperature. Konačni gubitak se računa kao srednja vrijednost svih pozitivnih parova u grupi, u oba poretka $(i, j)$ i $(j, i)$.

Kako bismo si vizualizirali prostor reprezentacija, koristit ćemo algoritme za smanjenje dimenzionalnosti koji su detaljnije opisani u poglavlju 4.4. Na slikama 4.2 prikazane su SimCLR reprezentacije $\boldsymbol{h_i}$ svih primjera za treniranje iz

skupa CIFAR-10 za pojedine napade. Za BadNets i WaNet, budući da su okidači dizajnirani da budu neprimjetni, reprezentacije otrovanih primjera često završe u grupi primjera originalnog razreda. Ovakve napada nazvat ćemo *nedisruptivnima*. S druge strane, napad SIG je *disruptivan*, njegov okidač je dovoljno jak da u prostoru SimCLR reprezentacija otrovani primjeri završe u posebnoj grupi. Kako bismo detektirali oba tipa napada, morat ćemo koristiti potpuno različite metode.




(a) (b)

(c)

**Slika 4.2:** Prostor SimCLR reprezentacija primjera iz skupa CIFAR-10 vizualiziran UMAP-om za napade BadNets (a), WaNet (b) i SIG (c). Ciljni razredi su 1 za BadNets, 0 za WaNet i 2 za SIG. Otrovani primjeri označeni su manjim, crnim točkama.

U [17], autori koriste algoritam k najbližih susjeda (kNN), detaljnije opisan u poglavlju 4.2. Međutim, s algoritmom kNN uspješno možemo detektirati samo nedisruptivne napade, stoga predlažemo uz kNN koristiti i algoritam k-sredina, opisan u poglavlju 4.3, pomoću kojeg ćemo moći detektirati i disruptivne napade. Pokazuje se da algoritam k-sredina loše radi na visoko-dimenzionalnim podacima, stoga ćemo prvo mapirati SimCLR reprezentacije u dvo-dimenzionalni prostor koristeći neki od algoritama za smanjenje dimenzionalnosti.

## 4.2. Algoritam k najbližih susjeda (kNN)

Jednom kada smo izračunali reprezentacije svih primjera, sljedeći cilj nam je detektirati otrovane primjere u prostoru SimCLR reprezentacija. Za nedisruptivne napade, koji imaju slabe okidače, otrovani primjeri će biti okruženi primjerima njihovog originalnog razreda, no njihova oznaka u skupu podataka će biti oznaka ciljnog razreda, $y_t$. Hipoteza je, dakle, da su otrovani primjeri oni čija se oznaka ne slaže s oznakom njihovih susjeda. Za svaki primjer $\boldsymbol{x_i}$, tj. njegovu reprezentaciju $\boldsymbol{h_i}$, računamo najčešću oznaku, $y_i'$, njegovih $k$ najbližih susjeda u prostoru reprezentacija. Ako predviđena oznaka $y_i'$ nije jednaka oznaci primjera $y_i$ u skupu podataka, tada taj primjer klasificiramo kao otrovani.

Jedini hiperparametar algoritma je $k$, broj susjeda na temelju kojih se određuje oznaka primjera. Algoritam je jako osjetljiv na ovaj hiperparametar - za maleni $k$, algoritam će biti previše lokalan i nestabilan, sklon prenaučenosti, a za veliki $k$ će biti podnaučen.

Kad bi reprezentacije bile savršene, takve da je grupa svakog razreda potpuno odvojena, u ovom koraku ne bismo izgubili niti jedan čisti primjer, bez obzira ima li napada te kakav je on. Međutim, grupe se često pomalo miješaju, te će možda doći do krive klasifikacije primjera na granicama grupa, stoga ćemo ovim korakom ipak izgubiti dio čistih primjera.

## 4.3. Algoritam k-sredina

Ako je napad disruptivan, njegov okidač je toliko jak da utječe i na reprezentacije učene samonadzirano. U prostoru reprezentacija, svi otrovani primjeri završe blizu jedni drugima, u jednoj grupi, daleko od ostalih. Na slici 2.3c, možemo primijetiti da je grupa otrovanih udaljenija od ostalih grupa, daleko više nego što su ostale grupe međusobno udaljene. Stoga, kad bismo mogli konzistentno detektirati sve grupe, odbacivanjem najmanje grupe bismo savršeno odbacili sve otrovane primjere.

U tu svrhu, koristit ćemo algoritam k-sredina. Svaku grupu algoritam k-sredina predstavlja jednim centroidom grupe, te primjer pripada onoj grupi čiji centroid mu je najbliži. Broj grupa, $k$, je hiperparametar, te ga u našem slučaju postavljamo na $k = 11$, jer očekujemo 10 grupa za razrede i 1 otrovanu grupu. Centroide grupa algoritam uči iterativno, alternirajući između dva koraka do konvergencije centroida:

1. Za danih $k$ sredina $\boldsymbol{\mu_j}$, svaki primjer $\boldsymbol{x_i}$ klasificiramo u najbližu grupu. Neka je $b_i^j$ varijabla koja je jednaka 1 ako primjer $\boldsymbol{x_i}$ pripada grupi s centroidom $\boldsymbol{\mu_j}$, a inače 0. Tada:

$$b_i^j = \begin{cases} 1 & \text{ako } j = \text{argmin}_l \|\boldsymbol{x_i} - \boldsymbol{\mu_l}\| \\ 0 & \text{inače} \end{cases} \tag{4.4}$$

2. Ažuriramo vrijednosti centroida na srednju vrijednost svih primjera u grupi:

$$\boldsymbol{\mu_j} = \frac{\sum_{i=1}^N b_i^j \boldsymbol{x_i}}{\sum_{i=1}^N b_i^j} \tag{4.5}$$

Za inicijalizaciju centroida koristimo algoritam *kmeans++* [2]. Prvi centroid odabiremo slučajno. Svaki idući centroid odabiremo tako da je što udaljeniji od ostalih. Točnije, vjerojatnost odabira primjera kao novi centroid proporcionalna je kvadratu udaljenosti tog primjera od njemu najbližeg središta. Ovakva inicijalizacija bi trebala odmah postaviti jedan centroid među otrovane primjere budući da su oni najudaljeniji.

Nedostatak ovog koraka je što svakako odbacujemo dio primjera. U slučaju da napada nema, ili ako je napad nedisruptivan, u najgorem slučaju odbacit ćemo $\frac{1}{k}$ primjera. Ako bismo ovo htjeli izbjeći, moramo ubaciti neki način detekcije postoji li uopće napad prije nego odbacimo primjere. Jedna mogućnost je odbaciti primjere samo ako je grupa *značajno* najudaljenija. Također, možemo primijetiti da će grupa otrovanih biti i najmanja grupa, te možemo odbaciti grupu samo ako je i najudaljenija i najmanja. Međutim, ovime pretpostavljamo da najmanji razred i dalje sadrži više primjera nego što je otrovanih, što ne mora nužno biti istinito.

## 4.4. Smanjenje dimenzionalnosti

Pokazuje se da algoritam k-sredina radi loše na 512-dimenzionalnim SimCLR reprezentacijama. Grupe reprezentacija su vjerojatno vrlo nepravilni visoko-dimenzionalni oblici, dok algoritam k-sredina pretpostavlja da su grupe hipersfere. Stoga, koristit ćemo neki algoritam smanjenja dimenzionalnosti, poput t-SNE ili UMAP, kako bismo pojednostavili visoko-dimenzionalne oblike u jednostavnije dvodimenzionalne oblike.

### 4.4.1. t-SNE

Algoritam t-SNE (*t-Distributed Stochastic Neighbor Embedding*) [15] nadogradnja je na algoritam SNE (*Stochastic Neighbor Embedding*) [10], stoga ćemo opisati prvo SNE.

Algoritam SNE visoko-dimenzionalne točke $\boldsymbol{x_i}$ u prostoru $X$ mapira u nisko-dimenzionalne točke $\boldsymbol{y_i}$ u prostoru $Y$. Euklidske udaljenosti točaka u prostoru $X$ transformiramo u uvjetne vjerojatnosti $p_{j|i}$ koje predstavljaju sličnosti. Sličnost primjera $\boldsymbol{x_j}$ i primjera $\boldsymbol{x_i}$ je uvjetna vjerojatnost, $p_{j|i}$, da $\boldsymbol{x_i}$ izabere $\boldsymbol{x_j}$ kao svog susjeda, ako se susjedi biraju proporcionalno gustoći vjerojatnosti normalne distribucije centrirane nad $\boldsymbol{x_i}$:

$$p_{j|i} = \frac{\exp\left(-\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|\boldsymbol{x_i} - \boldsymbol{x_k}\|^2 / 2\sigma_i^2\right)} \tag{4.6}$$

Svaki primjer ima vlastitu varijancu distribucije, $\sigma_i$, te ćemo metodu za njeno određivanje objasniti kasnije. Uvjetnu vjerojatnost $p_{i|i}$ postavljamo na 0. Slično radimo i za točke u prostoru $Y$:

$$q_{j|i} = \frac{\exp\left(-\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2\right)}{\sum_{k \neq i} \exp\left(-\|\boldsymbol{y_i} - \boldsymbol{y_k}\|^2\right)} \tag{4.7}$$

Opet postavljamo $q_{i|i}$ na 0. Ovdje svakom primjeru dajemo jednaku varijancu. Kada bi mapiranje iz $X$ u $Y$ bilo savršeno, uvjetne vjerojatnosti $p_{j|i}$ i $q_{j|i}$ bi bile jednake. Dakle, SNE pokušava pronaći reprezentacije $\boldsymbol{y_i}$ koje minimiziraju različitost $p_{j|i}$ i $q_{j|i}$. Definiramo funkciju cijene $C$ kao sumu Kullback-Leibler (KL) divergencija po svim primjerima:

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \ln \frac{p_{j|i}}{q_{j|i}}, \tag{4.8}$$

gdje $P_i$ i $Q_i$ predstavljaju distribucije uvjetnih vjerojatnosti $p_{j|i}$ i $q_{j|i}$. Budući da KL divergencija nije simetrična, drukčiji tipovi grešaka u međusobnim udaljenostima su drukčije kažnjene. Konkretno, veći gubitak nanose udaljene točke u $Y$ koje su bliske u $X$, a manji gubitak nanose bliske točke u $Y$ koje su udaljene u $X$. Drugim riječima, SNE više zadržava lokalnu strukturu podataka.

Ostaje nam još odrediti parametar $\sigma_i$ normalne razdiobe nad svakim primjerom $\boldsymbol{x_i}$. Raspored primjera $\boldsymbol{x_i}$ može imati različite gustoće u različitim dijelovima prostora $X$, stoga svaki primjer ima vlastiti $\sigma_i$. Za određeni $\sigma_i$, distribucija $P_i$ ima entropiju:

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}, \tag{4.9}$$

te perpleksiju:

$$\text{Perp}(P_i) = 2^{H(P_i)} \tag{4.10}$$

Perpleksija distribucije je intuitivna mjera "nesigurnosti" - može se pokazati da je perpleksija uniformne razdiobe s $K$ diskretnih stanja jednaka upravo $K$. Dakle, ako bacamo novčić, perpleksija iznosi 2, a ako bacamo kocku, perpleksija iznosi 6.

U kontekstu SNE algoritma, perpleksiju možemo interpretirati kao glatku mjeru efektivnog broja susjeda. Naime, zamislimo da u visoko-dimenzionalnom prostoru imamo primjer $x_1$, 5 primjera $x_2, ..., x_6$ koji su jednako udaljeni od $x_1$, te primjer $x_7$ koji je udaljeniji od $x_1$ od ostalih. Ako postavimo perpleksiju na 5, tada će SNE za $x_1$ postaviti $\sigma_i$ normalne razdiobe koji daje najveću moguću vjerojatnost za $x_2, ..., x_6$, a najmanju za $x_7$. Ako vjerojatnost primjera $x_j$ interpretiramo kao "vjerojatnost da $x_1$ odabere $x_j$ za susjeda", onda $x_1$ ima 5 susjeda sada. Međutim, ako perpleksiju postavimo na 6, tada će odgovarajući $\sigma_i$ biti onaj koji približno jednoliko raspoređuje vjerojatnost na svih 6 primjera, te sada $x_1$ ima 6 susjeda.

SNE binarnom pretragom traži vrijednost $\sigma_i$ koja će rezultirati distribucijom $P_i$ sa određenom perpleksijom, koja se predaje algoritmu kao hiperparametar. Radi konzistentnosti s ostalim algoritmima koji isto imaju hiperparametar za broj susjeda, perpleksiju ćemo u eksperimentima označavati sa $k$.

Funkciju cijene minimiziramo stohastičkim gradijentnim spustom, te njen gradijent iznosi:

$$\frac{\partial C}{\partial \boldsymbol{y_i}} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(\boldsymbol{y_i} - \boldsymbol{y_j})^\top \tag{4.11}$$

Gradijent funkcije cijene možemo interpretirati kao silu opruge između parova primjera u prostoru $Y$. Svaka opruga djeluje silom u smjeru vektora $(\boldsymbol{y_i} - \boldsymbol{y_j})$, te privlači ili odbija primjere, ovisno o tome je li udaljenost u $Y$ premala ili prevelika u usporedbi s udaljenosti u $X$. Iznos sile je proporcionalan i udaljenosti primjeri $(\boldsymbol{y_i} - \boldsymbol{y_j})$, i razlici sličnosti u oba prostora $(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})$.

Ostaje nam još inicijalizirati točke $\boldsymbol{y_i}$. Možemo uzorkovati točke iz normalne distribucije centrirane u ishodištu, ali pokazalo se stabilnijim koristiti reprezentacije koje za skup podataka izračuna algoritam PCA.

Prva nadogradnja na SNE algoritam koji autori rada [15] predlažu je pojednostaviti funkciju cilja koristeći simetrične vjerojatnosti:

$$p_{ij} = p_{ji} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{4.12}$$

$$q_{j|i} = \frac{\exp\left(-\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2\right)}{\sum_{k \neq l} \exp\left(-\|\boldsymbol{y_k} - \boldsymbol{y_l}\|^2\right)} \tag{4.13}$$

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \ln \frac{p_{ij}}{q_{ij}} \tag{4.14}$$

Prednost ovakve formulacije je jednostavniji oblik gradijenta, koji se brže računa:

$$\frac{\partial C}{\partial \boldsymbol{y_i}} = 4 \sum_j (p_{ij} - q_{ij})(\boldsymbol{y_i} - \boldsymbol{y_j})^\top \tag{4.15}$$

Međutim, glavni nedostatak SNE-a je problem gužve (engl. *crowding problem*). Pretpostavimo da SNE-om želimo mapirati točke iz 10-dimenzionalnog prostora $X$ u 2-dimenzionalni prostor $Y$. Ako primjeri u $X$ leže na 2-dimenzionalnoj plohi koja je približno linearna na maloj skali, možemo ih dobro prikazati i u 2-dimenzionalnom prostoru. Međutim, ako primjeri leže na 10-dimenzionalnoj mnogostrukosti (engl. *manifold*), tada ih nije moguće vjerno prikazati. Primjerice, u 10-dimenzionalnom prostoru, moguće je imati 11 primjera koji su svi međusobno jednako udaljeni, dok je u 2-dimenzionalnom prostoru moguće imati najviše 3 takva primjera. Nadalje, volumen sfere u m-dimenzionalnom prostoru oko primjera je $r^m$ - ako su primjeri približno uniformno raspoređeni u prostoru oko nekog primjera $\boldsymbol{x_i}$, površina koju imamo na raspolaganju oko primjera $\boldsymbol{y_i}$ u prostoru $Y$ neće biti ni približno dovoljna da smjestimo sve primjere. Stoga ako želimo precizno modelirati male udaljenosti, većina primjera koji su srednje udaljeni u $X$ će morati biti jako udaljeni u $Y$. U SNE-u, takvi primjeri koje bismo trebali prikazati kao udaljene u $Y$, će zapravo imati srednje veliki $p_{j|i}$ (jer su srednje udaljeni od $\boldsymbol{x_i}$) i maleni $q_{j|i}$ (jer su jako udaljeni u $Y$), te stoga će ih jaka sila vući zajedno i završit će preblizu jedni drugih.

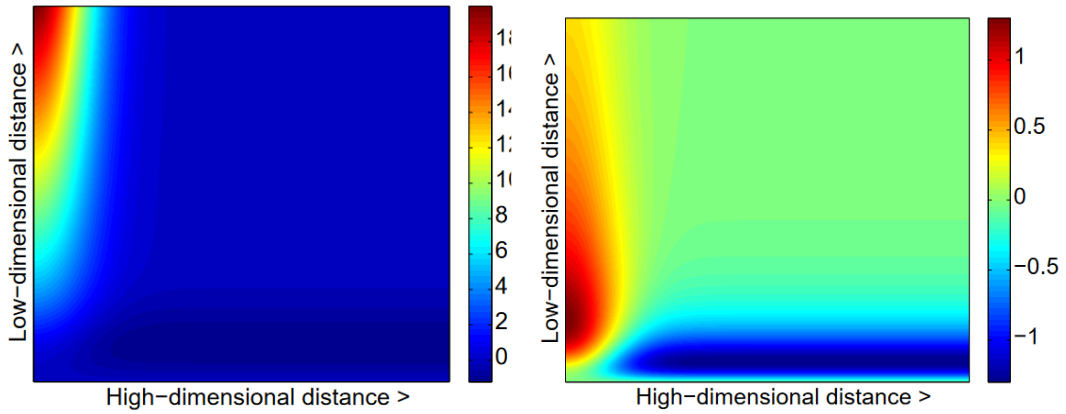
Kao rješenje za ovaj problem, autori predlažu koristiti Studentovu t-distribuciju sa jednim stupnjem slobode kao mjeru sličnosti $Y$:

$$q_{ij} = \frac{(1 + \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2)^{-1}}{\sum_{k \neq l} (1 + \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2)^{-1}} \tag{4.16}$$

Studentova t-distribucija je oblikom slična normalnoj, no ima veću gustoću vjerojatnosti na repovima, što dopušta srednje udaljenim točkama iz $X$ da budu smještene daleko jedna od druge u $Y$. Gradijent funkcije cijene je sada:

$$\frac{\partial C}{\partial \boldsymbol{y_i}} = 4 \sum_j (p_{ij} - q_{ij})(\boldsymbol{y_i} - \boldsymbol{y_j})^\top (1 + \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2)^{-1} \tag{4.17}$$

Slike 4.3 prikazuju iznose gradijenata algoritama SNE i t-SNE kao funkciju udaljenosti dvije točke u visoko-dimenzionalnom (x-os) i nisko-dimenzionalnom (y-os) prostoru. Na slikama se jasno vidi nedostatak SNE-a - privlačna sila je jaka (do 19) za daleke primjere u $Y$ koji su bliski u $X$, ali odbojna sila je jako slaba (do 1) za primjere bliske u $Y$ koji su udaljeni u $X$. Kod gradijenta za t-SNE, vidimo da su privlačne i odbojne sile približno jednake u obje situacije, tj. t-SNE jednako privlači udaljene slične točke i odbija bliske različite točke.




**(a)** **(b)**

**Slika 4.3:** Iznosi gradijenta algoritma SNE (a) i t-SNE (b). Na apscisi se nalazi udaljenost primjera u visoko-dimenzionalnom prostoru, a na ordinati udaljenost primjera u nisko-dimenzionalnom prostoru. Pozitivan iznos gradijenta predstavlja privlačnu, a negativan iznos odbojnu silu primjera $\boldsymbol{y_i}$ i $\boldsymbol{y_j}$. Slike preuzete iz [15].

### 4.4.2. UMAP

Algoritam UMAP (engl. *Uniform Manifold Approximation and Projection*) [14] funkcionira slično kao i t-SNE - na neki način modelira sličnosti primjera u originalnom i rezultirajućem prostoru, te minimizira razliku tih sličnosti. Dok su izbori pojedinih koraka (npr. kako računamo sličnost primjera u $X$) kod algoritma t-SNE potkrijepljeni samo eksperimentima, autori UMAP-a matematički dokazuju da su njihovi izbori koraka optimalni uz sljedeće pretpostavke:

- Postoji mnogostrukost na kojoj su primjeri uniformno distribuirani
- Mnogostrukost je lokalno povezana

– Očuvanje topološke strukture mnogostrukosti je glavni cilj

Kao i u t-SNE, prvi korak je pronaći neku mjeru sličnosti primjera u visokodimenzionalnom prostoru $X$. Umjesto normalnih distribucija, UMAP za svaki primjer $\boldsymbol{x_i}$ gradi težinski graf k-susjeda. Za dani hiperparametar $k$, za svaki $\boldsymbol{x_i}$ računamo skup $k$ najbližih susjeda, $\{\boldsymbol{x_{i_1}}, ..., \boldsymbol{x_{i_k}}\}$. Nadalje, definiramo $\rho_i$ kao najmanju udaljenost susjeda:

$$\rho_i = \min\{\left\|\boldsymbol{x_i} - \boldsymbol{x_{i_j}}\right\| \mid 1 \leq j \leq k, \left\|\boldsymbol{x_i} - \boldsymbol{x_{i_j}}\right\| > 0\} \tag{4.18}$$

Također, binarnom pretragom postavljamo $\sigma_i$ tako da vrijedi:

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, \left\|\boldsymbol{x_i} - \boldsymbol{x_{i_j}}\right\| - \rho_i)}{\sigma_i}\right) = \log_2(k) \tag{4.19}$$

Vrijednost $\log_2(k)$ je odabrana empirijski.

Sada možemo definirati težinski usmjereni graf $G' = (V, E, w')$. Vrhovi $V$ su svi primjeri $\boldsymbol{x_i}$. Skup bridova definiramo kao $E = \{(\boldsymbol{x_i}, \boldsymbol{x_{i_j}}) \mid 1 \leq j \leq k, 1 \leq i \leq N\}$, tj. postoji brid od $\boldsymbol{x_i}$ do $\boldsymbol{x_l}$ ako je $\boldsymbol{x_l}$ u skupu najbližih $k$ susjeda primjera $\boldsymbol{x_i}$. Težine bridova postavljamo na sljedeći način:

$$w'((\boldsymbol{x_i}, \boldsymbol{x_{i_j}})) = \exp\left(\frac{-\max(0, \left\|\boldsymbol{x_i} - \boldsymbol{x_{i_j}}\right\| - \rho_i)}{\sigma_i}\right) \tag{4.20}$$

Ovime smo postavili težinu brida između $\boldsymbol{x_i}$ i njegovog najbližeg susjeda na 1, a težine ostalih susjeda opadaju s udaljenosti. Vrijednost $\sigma_i$ možemo interpretirati isto kao i kod t-SNE-a - njome definiramo zasebnu mjeru udaljenosti za svaki $\boldsymbol{x_i}$ kako bismo mogli mapirati dijelove prostora različite gustoće.

Međutim, opet imamo nesimetričnost - težina brida od $\boldsymbol{x_i}$ do $\boldsymbol{x_j}$ nije jednaka kao i težina brida od $\boldsymbol{x_j}$ do $\boldsymbol{x_i}$. Na graf svakog primjera možemo gledati kao na neizraziti skup, tj. težina brida $w'((\boldsymbol{x_i}, \boldsymbol{x_{i_j}}))$ predstavlja mjeru pripadnosti primjera $\boldsymbol{x_{i_j}}$ neizrazitom skupu susjeda primjera $\boldsymbol{x_i}$. Kako bismo od usmjerenih težinskih bridova stvorili neusmjerene težinske bridove, koristimo t-konormu - neizrazitu ekvivalentu unije skupova. Ako su $w'_1$ i $w'_2$ težine usmjerenih bridova između dva primjera, tada je težina neusmjerenog brida:

$$w = w'_1 + w'_2 - w'_1 \cdot w'_2 \tag{4.21}$$

Drugim riječima, ako je $A$ nesimetrična težinska matrica susjedstva grafa $G'$, tada konstruiramo neusmjereni težinski graf $G$ sa simetričnom težinskom matricom susjedstva:

$$B = A + A^T - A \circ A^T \tag{4.22}$$

gdje je $\circ$ Hadamardov produkt (umnožak po elementima). Ako težine $w_1'$ i $w_2'$ interpretiramo kao vjerojatnost da usmjereni brid postoji, tada je $w$ vjerojatnost da barem jedan od usmjerenih bridova postoji.

Sada kada smo definirali sličnost primjera $\boldsymbol{x_i}$ i $\boldsymbol{x_j}$ kao težinu $w_{ij}$ brida neusmjerenog grafa, trebamo isto učiniti i za reprezentacije $\boldsymbol{y_i}$ i $\boldsymbol{y_j}$ u nisko-dimenzionalnom prostoru. I tu sličnost predstavljamo kao pripadnost neizrazitom skupu susjeda, međutim ovdje trebamo diferencijabilnu funkciju pripadnosti kako bismo mogli optimirati reprezentacije. Odabiremo sljedeću funkciju:

$$v(\boldsymbol{y_i}, \boldsymbol{y_j}) = (1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^{-1} \tag{4.23}$$

Hiperparametre $a$ i $b$ određujemo iz hiperparametra najmanje dopuštene udaljenosti *min_dist*. Što je *min_dist* manji, dopuštamo da reprezentacije završe bliže jedne drugima. Ovaj hiperparametar je estetičke prirode - utjecat će bitnije samo na vizualizaciju reprezentacija (jesu li one prikazane gušće ili raspršenije), no struktura oblika bi trebala ostati jednaka, stoga u eksperimentima ostavljamo ovaj hiperparametar na pretpostavljenoj vrijednosti.

Preostaje još definirati funkciju cijene. Odabiremo unakrsnu entropiju:

$$C = \sum_{i \neq j} \left[ w_{ij} \ln\left(\frac{w_{ij}}{v_{ij}}\right) + (1 - w_{ij}) \ln\left(\frac{1 - w_{ij}}{1 - v_{ij}}\right) \right] \tag{4.24}$$

Jednadžbu 4.24 možemo još raspisati kao:

$$\begin{aligned} C = &\sum_{i \neq j} \left[ w_{ij} \ln(w_{ij}) + (1 - w_{ij}) \ln((1 - w_{ij})) \right] \\ &- \sum_{i \neq j} \left[ w_{ij} \ln(v_{ij}) + (1 - w_{ij}) \ln((1 - v_{ij})) \right] \end{aligned} \tag{4.25}$$

Budući da prva suma u jednadžbi 4.25 ovisi samo o $w_{ij}$ koji su konstantni za vrijeme optimizacije, konačno minimiziramo samo:

$$C = -\sum_{i \neq j} \left[ w_{ij} \ln(v_{ij}) + (1 - w_{ij}) \ln((1 - v_{ij})) \right] \tag{4.26}$$

Gradijent po prvom članu sume nam daje privlačnu silu[1]:

[1] U originalnom radu, izraz za privlačnu silu se blago razlikuje od izraza 4.27 - u nazivniku nema hiperparametara $a$ i $b$. Međutim, to je pogreška, na što su i autori ukazali kasnije (`https://github.com/lmcinnes/umap/issues/460`). Točan izvod nalazi se u dodatku A.

$$\frac{\partial w_{ij} \ln(v_{ij})}{\partial \boldsymbol{y_i}} = \frac{-2ab\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2(b-1)}}{1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}} w_{ij}(\boldsymbol{y_i} - \boldsymbol{y_j})^\top \tag{4.27}$$

Gradijent po drugom članu sume nam daje odbojnu silu:

$$\begin{aligned} &\frac{\partial(1 - w_{ij}) \ln(1 - v_{ij})}{\partial \boldsymbol{y_i}} = \\ &= \frac{2b}{(\epsilon + \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2)(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})} (1 - w_{ij})(\boldsymbol{y_i} - \boldsymbol{y_j})^\top, \end{aligned} \tag{4.28}$$

gdje je $\epsilon = 0.001$ dodan kako bismo izbjegli dijeljenje s nulom. Primjere za koje ćemo primjenjivati privlačne i odbojne sile uzorkujemo slučajno.

Prednosti UMAP-a nad t-SNE su sljedeće:

- UMAP je brži. Računanje $k$ najbližih susjeda UMAP provodi algoritmom Nearest-Neighbor-Descent opisanim u [6], za koji su autori empirijski pokazali da je vremenske kompleksnosti oko $O(N^{1.14})$, dok optimizacija reprezentacija je vremenske kompleksnosti $O(kN)$ - konačna kompleksnost stoga iznosi otprilike $O(N^{1.14} + kN)$. S druge strane, budući da t-SNE mora računati udaljenosti za sve parove primjera, njegova kompleksnost iznosi $O(N^2)$.
- Kvalitativno, vizualizacije izgledaju "bolje", ali i kvantitativno, koristeći kNN klasifikator na reprezentacijama dobivenim s oba algoritma nad raznim skupovima podataka, performanse su bolje s UMAP reprezentacijama.
- UMAP i t-SNE su stohastični, tj. njihovi će rezultati biti drukčiji za svako pokretanje. Međutim, UMAP daje stabilnije reprezentacije.

# 5. Eksperimenti

## 5.1. Mjere uspješnosti

Kako bismo usporedili razne napade i obrane, naučit ćemo model sa primijenjenim napadom i obranom, te promatrati sljedeće metrike nad skupom za testiranje:

- C-Acc (*clean accuracy*) - točnost predikcije čistih primjera u originalne, ispravne razrede
- ASR (*attack success rate*) - točnost predikcije otrovanih primjera u ciljni razred

Cilj napadača je postići visok C-Acc i ASR. Cilj obrane je visok C-Acc i nizak ASR.

Budući da naša metoda čisti skup podataka, tj. detektira i uklanja otrovane primjere, tada možemo promatrati i kvalitetu klasifikacije primjera u otrovane i čiste. Tu ćemo promatrati sljedeće:

- Udio otrovanih primjera - omjer otrovanih i čistih primjera u konačnom (očišćenom) skupu
- Udio zadržanih čistih primjera - omjer čistih primjera u konačnom (očišćenom) i originalnom (otrovanom) skupu

## 5.2. Validacija hiperparametara

Eksperimente provodimo nad skupom CIFAR-10 [12], koji sadrži ukupno 60000 slika u boji veličine $32 \times 32$ klasificiranih u 10 razreda. Skup podataka ćemo otrovati napadima:

1. *Badnets1* - BadNets napad sa stopom trovanja $p_a = 1\%$ i okidačem na slici 2.1a u donjem desnom kutu

2. *Badnets10* - BadNets napad sa stopom trovanja $p_a = 10\%$ i okidačem na slici 2.1a u donjem desnom kutu

3. *WaNet* - WaNet napad sa stopom trovanja $p_a = 10\%$, stopom šuma $p_n = 20\%$, veličinom generiranog deformacijskog polja $k = 4$ i jakosti deformacijskog polja $s = 0.5$

4. *SIG* - SIG napad sa stopom trovanja $p_a = 1\%$, $\alpha = 0.2$, $\Delta = 20$ i $f = 6$. Rezultirajući okidačem prikazan je na slici 2.3a

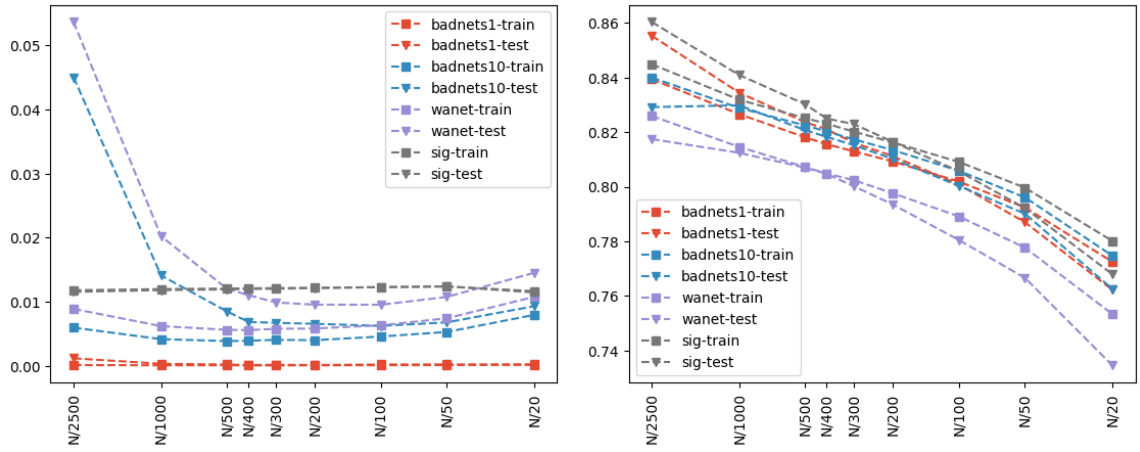
Svi prikazani rezultati su prosjek pokretanja napada (i obrana nad tim napadima) 3 puta, svaki put za drugačiji ciljni razred, $y_t \in \{0, 1, 2\}$. Eksperimente provodimo i na skupu za učenje (50000 primjera) i na skupu za validaciju (10000 primjera).

Prvi korak naše obrane je treniranje modela SimCLR. Model treniramo 250 epoha sa optimizatorom LARS [18] i stopom učenja 0.2. Za okosnicu koristimo model ResNet-18. Naučene reprezentacije, nakon smanjenja dimenzionalnosti algoritmom UMAP, prikazane su na slici 4.2.

Kako bismo bolje procijenili kvalitetu naučenih reprezentacija, na izlaz SimCLR modela dodajemo potpuno povezani sloj (sa 512 ulaznih i 10 izlaznih dimenzija) i aktivacijsku funkciju Softmax. Takav model učimo 10 epoha stohastičkim gradijentnim spustom sa stopom učenja 0.6 i Nesterovim momentom 0.9, te postižemo točnost od 84.08% na skupu za validaciju. Autori rada o SimCLR-u ovakvim postupkom postižu točnost preko 95%, no oni koriste jaču okosnicu, ResNet-50. Također, pokazano je da učenju SimCLR-a pogoduje više epoha i veća veličine grupe.

Sljedeći korak, koji detektira primjere otrovane nedisruptivnim napadima, je klasifikacija primjera kNN-om. Graf 5.1 prikazuje kako broj susjeda, $k$, utječe na udio otrovanih primjera i udio zadržanih čistih primjera za svaki napad. Broj susjeda postavljamo u ovisnosti o ukupnom broju primjera, $N$. Uočavamo da povećanjem $k$, kNN sve lošije klasificira čiste primjere, tj. opada udio zadržanih čistih. Međutim, i preniske i previsoke vrijednosti $k$ povećavaju udio otrovanih primjera, te minimum uvijek postižemo za $k$ od $\frac{N}{500}$ do $\frac{N}{100}$. Budući da povećanjem $k$ zadržavamo sve manje čistih primjera, konačno odabiremo $k = \frac{N}{500}$ kao vrijednost hiperparametra.

Očekivano, kNN nam ne pomaže pri disruptivnim napadima. Pri vizualizaciji SimCLR reprezentacija na slici 4.2c, vidimo da su primjeri otrovani disruptivnim




**(a)** **(b)**

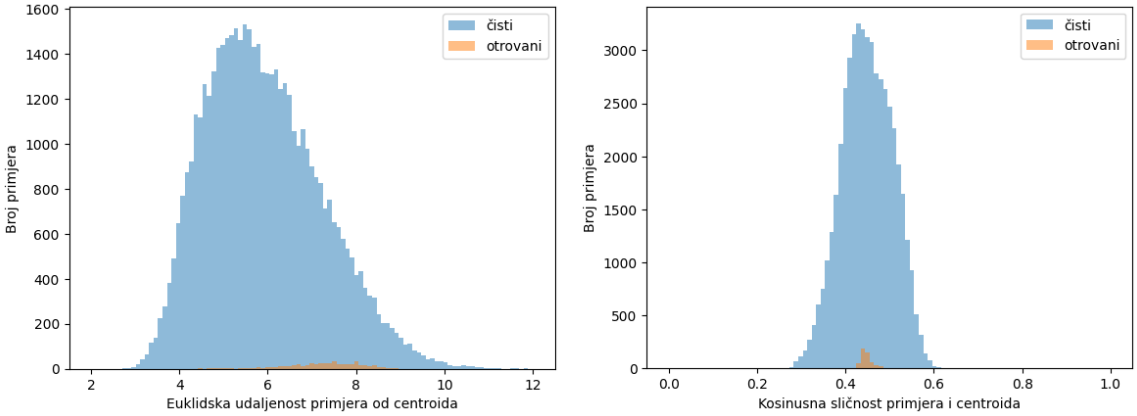
**Slika 5.1:** Udio otrovanih primjera (a) i udio zadržanih čistih primjera (b) nakon primjene algoritma k najbližih susjeda na SimCLR reprezentacijama pojedinih napada za razne vrijednosti hiperparametra k.

SIG napadom smješteni daleko od ostatka skupa, stoga možemo pokušati detektirati otrovane primjere kao najudaljenije. Međutim, slike 5.2 pokazuju da to nije slučaj za originalni, 512-dimenzionalni prostor reprezentacija.

Stoga, osim za vizualizaciju prostora, smanjenje dimenzionalnosti koristimo kao ključan korak detektiranja disruptivnih napada. Razmotrit ćemo algoritme t-SNE i UMAP. Oba algoritma imaju hiperparametar koji označava koliko bi svaki primjer trebao imati susjeda.. Slike 5.3 prikazuju ponašanje algoritama u ovisnosti o broju susjeda, označenog s $k$, definiranog kao udio veličine skupe podataka, $N$. UMAP puno bolje razdvoji grupu otrovanih, te je robustan na promjene $k$. Grupa otrovanih u prostoru t-SNE reprezentacija je na jednakoj udaljenosti kao i dio čistih, te performanse algoritma uvelike ovise o $k$. Bolji izbor je UMAP, te ćemo radi jednostavnosti odabrati $k = \frac{N}{500}$, kao i u kNN koraku.

Gledajući isključivo udaljenost, mogli bismo proglasiti neki postotak najudaljenijih primjera kao otrovane. Međutim, ne možemo znati koliko je stvarno otrovanih primjera, i teško je odlučiti gdje ćemo povući granicu. Osim što su najudaljeniji, otrovani primjeri su i grupirani u jednu grupu, stoga jednostavan način za detektirati ih je algoritam k-sredina, koji bi trebao uvijek smjestiti jedan centroid u grupu otrovanih, koji bi potom trebao biti i najudaljeniji centroid. Izbor hiperparametra $k$ je ovdje očigledan - očekujemo 10 čistih i 1 otrovanu grupu, stoga $k = 11$.

Još jedan potencijalni uvjet koji možemo primijetiti je da će otrovana grupa



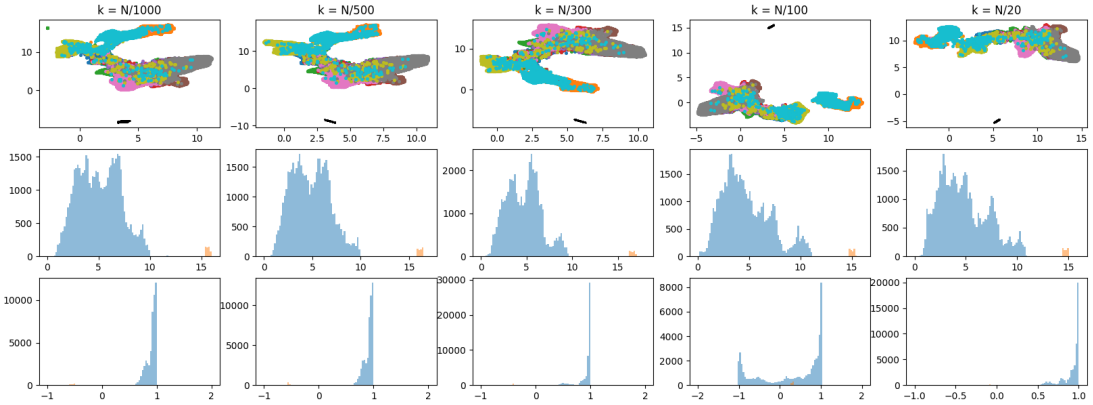

**(a)** **(b)**

**Slika 5.2:** Distribucija udaljenosti (a) i kosinusne sličnosti (b) SimCLR reprezentacija sa centroidom cijelog skupa. Plavom bojom je označena distribucija čistih, a narančastom otrovanih primjera. U prostoru reprezentacija, otrovani primjeri nisu značajno udaljeniji od centroida u usporedbi sa čistima.

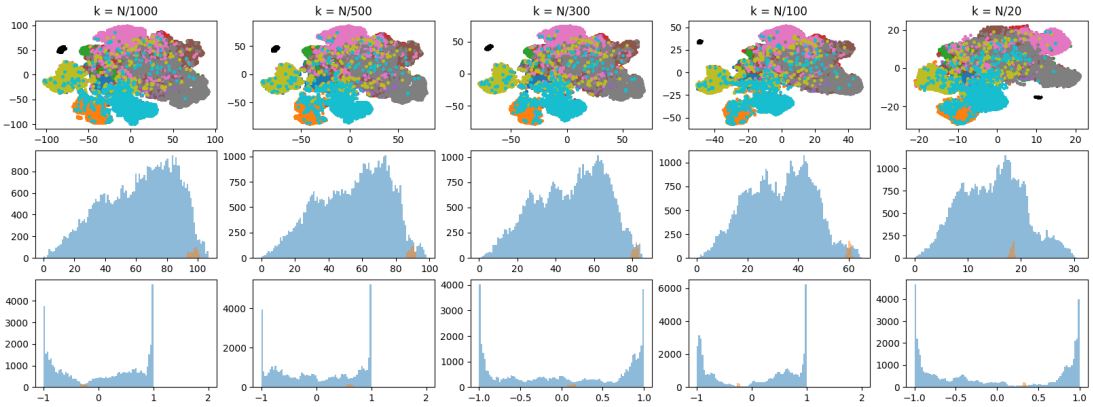
sadržavati najmanje primjera. S dva uvjeta, možemo odlučiti ne odbaciti nijedan primjer ako najudaljeniji primjer nije ujedno i najmanji, dok samo s jednim uvjetom uvijek odbacujemo dio primjera, čak i ako napada nema. Ali, pretpostavka da je disruptivan napad otrovao manje primjera nego što ih je u bilo kojem od razreda je potencijalno netočna, stoga predlažemo ipak koristiti samo uvjet da je otrovana grupa samo najudaljenija.

Tablica 5.1 prikazuje udio otrovanih i udio zadržanih čistih primjera u skupu podataka nakon primjene algoritma k-sredina na UMAP reprezentacijama. Očekivano, k-sredina nam ne pomaže pri nedisruptivnim napadima, te kod njih samo pogoršava situaciju - udio otrovanih se blago i poveća, a izgubimo i do 12% čistih. Najbolje performanse postižemo odbacivanjem grupe koja je i najmanja i najudaljenija. Za disruptivni SIG napad uspijemo ukloniti sve otrovane primjere bez da gubimo čiste.

Kako bismo provjerili konzistentnost algoritma k-sredina za SIG napad, pokrenuli smo algoritam još 1000 puta nad svakim od 6 skupova podataka otrovanih SIG napadom (3 skupa za učenje i 3 skupa za validaciju, za ciljne razrede 0,1,2). Algoritam k-sredina uvijek ispravno odvoji primjere - svi čisti su označeni kao čisti, svi otrovani su označeni kao otrovan. Jedino za jedan od 6 skupova podataka 2 otrovana primjera završe u krivoj grupi, stoga i u tablici 5.1 zadnji redak pokazuje 0.01% otrovanih.




**(a)**




**(b)**

**Slika 5.3:** UMAP (a) i t-SNE (b) za razne vrijednosti hiperparametra $k$ koji predstavlja broj susjeda, definiranog u ovisnosti o veličini skupa podataka, $N$. Prvi red prikazuje prostor 2D reprezentacija. Drugi red prikazuje euklidske udaljenosti, a treći red kosinusne sličnosti 2D reprezentacija sa centroidom cijelog skupa.

| | Udio otrovanih / % | | | Udio zadržanih čistih / % | | |
|---|---|---|---|---|---|---|
| | udaljenost | veličina | oboje | udaljenost | veličina | oboje |
| BadNets1-train | 1.02 | 1.01 | 1.00 | 92.20 | 94.50 | 99.21 |
| BadNets1-test | 1.01 | 1.01 | 1.01 | 91.43 | 93.84 | 98.87 |
| BadNets10-train | 10.17 | 10.09 | 10.00 | 91.97 | 94.57 | 100.00 |
| BadNets10-test | 10.19 | 10.01 | 10.00 | 91.65 | 94.11 | 99.31 |
| WaNet-train | 10.46 | 9.90 | 10.02 | 92.28 | 94.59 | 99.42 |
| WaNet-test | 11.14 | 9.79 | 10.03 | 88.83 | 94.65 | 99.64 |
| SIG-train | **0.00** | **0.00** | **0.00** | **100.00** | **100.00** | **100.00** |
| SIG-test | **0.01** | **0.01** | **0.01** | **100.00** | **100.00** | **100.00** |

**Tablica 5.1:** Udio otrovanih primjera u skupu podataka nakon primjene k-sredina na UMAP reprezentacijama te odbacivanjem grupe koja je najudaljenija, najmanja, ili oboje. Prikazani brojevi prosjek su 30 pokretanja algoritma k-sredina. K-sredina uspije u potpunosti očistiti skup podataka otrovan SIG napadom.

Sada kada imamo metode za detekciju i disruptivnih i nedisruptivnih napada, trebamo ih nekako spojiti. Budući da je veća šteta propustiti otrovani primjeri nego što je odbaciti čisti, odlučujemo se za najstrožu varijantu - odbacujemo primjer ako je ijedna od dviju metoda ga proglasila otrovanim. Primjer onda zadržavamo samo ako su ga obje metode proglasile čistim.

Dakle, naša konačna metoda sastoji se od sljedećih koraka:

1. Učenje SimCLR reprezentacija nad potencijalno otrovanim skupom podataka.

2. Uklanjanje primjera kojima algoritam k najbližih susjeda ($k = \frac{N}{500}$) u prostoru SimCLR reprezentacija da oznaku drukčiju od originalne.

3. Smanjenje dimenzionalnosti na 2 dimenzije algoritmom UMAP ($k = \frac{N}{500}$), te uklanjanje najudaljenije grupe nakon primjene algoritma k-sredina nad UMAP reprezentacijama.

Performanse metode na opisanim napadima prikazane su u tablici 5.2. Za svaki od napada, rezultirajući udio otrovanih primjera bi trebao biti dovoljno malen da se napad suzbije, što ćemo pokazati u poglavlju 5.4. Međutim, u obrani gubimo oko 20-25% čistih primjera.

| | Udio otrovanih / % | Udio zadržanih čistih / % |
|---|---|---|
| Čisti-train | - | 73.67 |
| Čisti-test | - | 73.92 |
| BadNets1-train | 0.01 | 74.34 |
| BadNets1-test | 0.03 | 73.46 |
| BadNets10-train | 0.42 | 73.48 |
| BadNets10-test | 0.91 | 74.25 |
| WaNet-train | 0.62 | 73.84 |
| WaNet-test | 1.29 | 74.80 |
| SIG-train | 0.00 | 82.51 |
| SIG-test | 0.00 | 83.02 |

**Tablica 5.2:** Performanse naše predložene metode.

## 5.3. Reklasifikacija

Nakon što smo postigli relativno niske količine otrovanih primjera u svojim skupovima podataka, postavlja se pitanje možemo li na temelju postojeće klasifikacije u čiste i otrovane postići bolju *reklasifikaciju*. Udio otrovanih je već dovoljno nizak da ASR bude blizu nuli, ali želja nam je podići broj čistih primjera kako bismo poboljšali i C-Acc modela.

Pokušali smo naučiti ResNet-18 [9] model na novim, "očišćenim" skupovima podataka iz tablice 5.2, te ponovno klasificirali sve primjere iz skupa podataka. Budući da je model učio samo iz čistih podataka, neće imati nikakve ugrađena stražnja vrata, te bi i otrovane primjere trebao klasificirati u njihove prave razrede. Hipoteza je da čiste primjere možemo detektirati kao one za koje ovaj model vraća istu oznaku kao što je u skupu podataka, tj. da su otrovani primjeri oni čije se oznake iz modela i iz skupa podataka razlikuju. Uspješnost ovakve *višerazredne* (jer treniramo model koji klasificira primjere u više razreda) reklasifikacije vidimo u tablici 5.3. Ova metoda je uglavnom bezuspješna, no u slučaju BadNets napada s 10% otrovanih primjera blago popravi situaciju, što nam daje neku nadu.

Alternativno, umjesto da učimo višerazredni klasifikator, možemo pokušati naučiti binarni klasifikator kojem je cilj klasificirati primjere u čiste i otrovane. Kao podatke za učenje dajemo mu našu dosadašnju klasifikaciju u čiste i otrovane iz tablice 5.2. Ovi podaci za učenje će biti, naravno, jako šumoviti - postojat će otrovani primjeri u razredu čistih, i čisti primjeri u razredu otrovanih. Naša mreža

| | Udio otrovanih / % | Udio zadržanih čistih / % |
|---|---|---|
| BadNets1-train | 0.02 | 72.41 |
| BadNets1-test | 0.13 | 60.48 |
| BadNets10-train | **0.31** | **76.17** |
| BadNets10-test | 1.52 | 63.37 |
| WaNet-train | 0.82 | 76.91 |
| WaNet-test | 2.22 | 55.48 |
| SIG-train | 0.16 | 80.13 |
| SIG-test | 0.24 | 73.29 |

**Tablica 5.3:** Performanse naše metode nakon višerazredne reklasifikacije.

| | Udio otrovanih / % | Udio zadržanih čistih / % |
|---|---|---|
| BadNets1-train | 0.31 | 63.40 |
| BadNets1-test | 0.83 | 67.23 |
| BadNets10-train | **0.00** | 70.75 |
| BadNets10-test | **0.04** | **77.07** |
| WaNet-train | 2.66 | 51.70 |
| WaNet-test | 5.30 | 53.91 |
| SIG-train | 0.00 | 59.67 |
| SIG-test | 0.00 | 47.06 |

**Tablica 5.4:** Performanse naše metode nakon binarne reklasifikacije.

treba samo naučiti prepoznati okidač, stoga koristimo relativno jednostavan model s dva konvolucijska i 3 potpuno povezana sloja za ovaj zadatak kako mreža ne bi naučila šum u skupu podataka. Uz to, model učimo samo 10 epoha. Rezultati su prikazani u tablici 5.4. Opet, nismo uspješni, osim u slučaju BadNets10.

## 5.4. Usporedba s drugim obranama

Općenito, rezultat neke obrane je novi model koji bi trebao imati što niži ASR i što viši C-Acc. Napade i obrane testirat ćemo na modelu PreAct-ResNet18 [8], treniranom 35 epoha na CIFAR-10 skupu podataka sa SGD optimizatorom i stopom učenja 0.01.

Za svaki napad, stvorena su po 3 skupa za učenje i validaciju, svaki put za drugačiji ciljni razred. Rezultati prikazani u tablici 5.5 su prosjek pokretanja napada i obrane nad ta 3 skupa.

Naša obrana u potpunosti suzbije sve napade, i distruptivne i nedisruptivne. Najotpornijim se pokazao WaNet, koji zadrži 10.31% ASR, no to je i dalje zadovoljavajući rezultat. Međutim, cijena koju plaćamo je gubitak oko 10 postotnih bodova točnosti C-Acc. Uzrok smanjenja C-Acc je to što u obrani odbacujemo oko 25% čistih primjera.

Neural Cleanse radi usporedivo dobro na BadNets napadu, gdje postiže značajno veći C-Acc, ali zato i veći ASR. Pretpostavka Neural Cleansea je da je okidač malen, što napadi WaNet i SIG značajno krše - njihovi okidači su preko cijele slike. Stoga na njima Neural Cleanse uopće ne detektira da postoje ugrađena stražnja vrata. Dodatno, nedostatak Neural Cleansea je što zahtijeva dostupne čiste primjere.

Activation Clustering ne uspijeva ni u jednom slučaju. U originalnom radu [4], eksperimenti su provedeni na jednostavnijim napadima (BadNets sa jednobojnim, većim okidačima), na jednostavnijem skupu podataka (MNIST i LISA prometni znakovi), te sa jednostavnijim modelima (konvolucijski modeli bez preskočnih veza ili BatchNorm sloja), te u takvim uvjetima AC uspijeva. No u malo kompleksnijim uvjetima, AC potpuno zakaže.

Možemo primijetiti i određenu sličnost između naše metode i AC-a. Obje metode svaki primjer predstavljaju kao neku latentnu reprezentaciju - naša metoda kao SimCLR reprezentacije, a AC kao skup aktivacija predzadnjeg sloja nekog dubokog modela. Nadalje, obje metode nastoje detektirati otrovane primjere promatrajući grupacije unutar prostora tih reprezentacija. Međutim, reprezentacije AC-a su učene nadzirano, te tako svaki napad postaje disruptivan u tom prostoru. Glavna mana je što model koji se uči nije uopće namijenjen za učenje reprezentacija. Ideja da su te reprezentacije dovoljno dobre zasniva se na pretpostavci da aktivacije predzadnjeg sloja će se značajno razlikovati za otrovane i neotrovane primjere, no za dovoljno kompleksne modele, skupove podataka i napade, to nije dovoljno. Sve ovo, u neku ruku, validira odabir SimCLR-a za računanje reprezentacija primjera.

Nedostatak naše metode je brzina izvođenja. Dok vrijeme izvođenja NC-a i AC-a traje približno koliko i treniranje samog modela, učenje SimCLR reprezentacija traje red veličine duže budući da zahtijeva daleko više epoha. Uz to, NC i AC zadrže puno veću točnost na čistom modelu, što vidimo u tablici 5.6.

| | BadNets1 | | BadNets10 | | WaNet | | SIG | |
|---|---|---|---|---|---|---|---|---|
| | C-Acc | ASR | C-Acc | ASR | C-Acc | ASR | C-Acc | ASR |
| Bez obrane | 91.03 | 95.28 | 90.48 | 99.97 | 88.22 | 93.44 | 91.23 | 99.87 |
| Naše | 79.33 | 2.24 | 76.94 | 2.6 | 80.89 | 10.31 | 83.38 | 0.21 |
| NC | 86.49 | 11.22 | 86.26 | 20.03 | - | - | - | - |
| AC | 85.36 | 96.78 | 84.68 | 98.38 | 85.11 | 98.84 | 84.37 | 94.41 |

**Tablica 5.5:** C-Acc i ASR modela treniranog nakon primjene pojedinih obrana i napada. Neural Cleanse ima opciju detektirati da stražnja vrata ne postoje u modelu, te onda ni ne uči novi model - stoga su neka polja za NC prazna u tablici. Svi prikazani brojevi su postotci.

| | C-Acc / % |
|---|---|
| Bez obrane | 91.07 |
| Naše | 77.61 |
| NC | - |
| AC | 89.94 |

**Tablica 5.6:** C-Acc modela treniranog na čistom skupu podataka nakon primjene pojedinih obrana. NC je detektirao da nije bilo napada, stoga ni ne uči ponovno model.

# 6. Zaključak

Način na koji učimo duboke modele podložan je napadima koji mogu umetnuti neprimjetna stražnja vrata. Istražili smo metodu koja samonadziranim učenjem nastoji detektirati i izbaciti otrovane primjere. Glavna pretpostavka ove metode je da su deformacije napada na primjeru slabe, te da većinu informacije nosi promjena oznake u napadu. Stoga samonadzirano učenje, koje ne gleda oznake, omogući metodi najbližih susjeda rekonstruirati pravu klasifikaciju primjera, što spusti udio otrovanih primjera sa 10% na ispod 1%. Međutim, postoje napadi koji ne mijenjaju oznaku otrovanih primjera i nanose velike izmjene na primjeru, te se takvi napadi ponašaju potpuno drugačije u prostoru reprezentacije naučenih samonadziranim učenjem. Predložili smo nadogradnju ove metode - osim klasifikacije kNN-om, na reprezentacije primjenjujemo smanjenje dimenzionalnosti i algoritam k sredina, te odbacujemo najudaljeniju grupu. Ovo se pokazuje uspješnim - odbacujemo dovoljno otrovanih primjera. Međutim, zbog nesavršenosti reprezentacija, gubimo i oko 25% čistih primjera, te to šteti točnosti rezultirajućeg modela. Kao budući rad trebalo bi istražiti kako smanjiti gubitak čistih primjera, primjerice upotrebom drugačijeg samonadziranog modela.

# Literatura


[1] Charu C. Aggarwal, Alexander Hinneburg, i Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. U Jan Van den Bussche i Victor Vianu, urednici, *Database Theory — ICDT 2001*, stranice 420–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44503-6.

[2] David Arthur i Sergei Vassilvitskii. K-means++: The advantages of careful seeding. svezak 8, stranice 1027–1035, 01 2007. doi: 10.1145/1283383.1283494.

[3] Mauro Barni, Kassem Kallas, i Benedetta Tondi. A new backdoor attack in cnns by training set corruption without label poisoning. U *2019 IEEE International Conference on Image Processing (ICIP)*, stranice 101–105. IEEE, 2019.

[4] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, i Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, i Geoffrey Hinton. A simple framework for contrastive learning of visual representations. U *International conference on machine learning*, stranice 1597–1607. PMLR, 2020.

[6] Wei Dong, Moses Charikar, i Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. stranice 577–586, 03 2011. doi: 10.1145/1963405.1963487.

[7] Tianyu Gu, Brendan Dolan-Gavitt, i Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Identity mappings in deep residual networks. U *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, stranice 630–645. Springer, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. U *Proceedings of the IEEE conference on computer vision and pattern recognition*, stranice 770–778, 2016.

[10] Geoffrey Hinton i Sam Roweis. Stochastic neighbor embedding. U *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, stranica 857–864, Cambridge, MA, USA, 2002. MIT Press.

[11] Diederik P Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Alex Krizhevsky, Vinod Nair, i Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[13] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, i Xingjun Ma. Anti-backdoor learning: Training clean models on poisoned data. *Advances in Neural Information Processing Systems*, 34:14900–14912, 2021.

[14] Leland McInnes, John Healy, i James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[15] Laurens van der Maaten i Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL `http://jmlr.org/papers/v9/vandermaaten08a.html`.

[16] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, i Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. U *2019 IEEE Symposium on Security and Privacy (SP)*, stranice 707–723, 2019. doi: 10.1109/SP.2019.00031.

[17] Hang Wang, Sahar Karami, Ousmane Dia, Hippolyt Ritter, Ehsan Emamjomeh-Zadeh, Jiahui Chen, Zhen Xiang, David J Miller, i George

Kesidis. Training set cleansing of backdoor poisoning by self-supervised representation learning. U *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, stranice 1–5. IEEE, 2023.

[18] Yang You, Igor Gitman, i Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

**Pronalaženje sustavnih pogrešaka označavanja u podatcima za učenje**

**Sažetak**

Veliki kapaciteti dubokih modela i ogromna količina podataka potrebna za njihovo učenje omogućava napade zasnovane na trovanju podataka za učenje. Otrovani primjeri sadrže neprimjetan uzorak te im je oznaka promijenjena na određeni ciljni razred. Učen na ovakvim podacima, duboki model će se ponašati normalno na običnim primjerima, no, kada vidi umetnuti okidač, krivo će klasificirati primjer u ciljni razred. Pokazano je da pristup baziran na samonadziranom učenju reprezentacija uspijeva očistiti skup podataka od većine napada. Međutim, napadi koji unose jake deformacije na sliku izbjegavaju ovu obranu jer se ponašaju drugačije u prostoru reprezentacija. Predlažemo poboljšanje čišćenja samonadziranim učenjem dodavanjem koraka smanjenja dimenzionalnosti i grupiranja kako bismo suzbili i disruptivne napade.

**Ključne riječi:** duboki modeli, stražnja vrata, samonadzirano učenje

**Finding systematic label errors in training data**

**Abstract**

Large capacities of deep models and the huge amounts of data required for their training enables backdoor attacks based on data poisoning. Poisoned samples contain an unperceptible pattern, and their label is changed into a target class. Trained on such data, a deep model will behave normally on regular samples, but will misclassify samples with the backdoor trigger pattern into the target class. It has been shown that an approach based on self-supervised representation learning can clean the training dataset from these attacks. However, attacks that inject strong perturbations into poisoned samples are not detected by cleansing based on self-supervised representations due to very different behaviours in latent representation space. We suggest an improvement of the self-supervised cleansing method by adding dimensionality reduction and clustering steps to detect disruptive attacks.

**Keywords:** deep models, backdoor, self-supervised learning

# Dodatak A
# Izvod gradijenata funkcije cijene algoritma UMAP

Za funkciju cilja u UMAP-u odabrali smo unakrsnu entropiju, te nam se naš optimizacijski problem pojednostavio na minimizaciju sljedeće funkcije cilja:

$$C = -\sum_{i \neq j} [w_{ij} \ln(v_{ij}) + (1 - w_{ij}) \ln(1 - v_{ij})] \tag{A.1}$$

Iznosi $w_{ij}$ su izračunati jednom za točke u visoko-dimenzionalnom prostoru, te su oni konstantni. Iznosi $v_{ij}$ su funkcije točaka u nisko-dimenzionalnom prostoru, $\boldsymbol{y_i}$ i $\boldsymbol{y_j}$:

$$v_{ij} = v(\boldsymbol{y_i}, \boldsymbol{y_j}) = (1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^{-1} \tag{A.2}$$

Točke $\boldsymbol{y_i}$ i $\boldsymbol{y_j}$ su zapravo parametri koje mijenjamo kako bismo minimizirali funkciju cilja, stoga tražimo $\frac{\partial C}{\partial \boldsymbol{y_i}}$. Međutim, kako bismo pojednostavili izraze, razlozit ćemo ovo na dvije komponente:

$$\frac{\partial C}{\partial \boldsymbol{y_i}} = -\sum_{i \neq j} \left[ \frac{\partial w_{ij} \ln(v_{ij})}{\partial \boldsymbol{y_i}} + \frac{\partial (1 - w_{ij}) \ln(1 - v_{ij})}{\partial \boldsymbol{y_i}} \right] \tag{A.3}$$

Minimizacija prvog člana, $-w_{ij} \ln(v_{ij})$, postiže se povećanjem $v_{ij}$, što približuje točke $\boldsymbol{y_i}$ i $\boldsymbol{y_j}$, stoga gradijent tog člana predstavlja privlačnu silu. S druge strane, minimizaciju drugog člana, $-(1 - w_{ij}) \ln(1 - v_{ij})$, postižemo smanjenjem $v_{ij}$, što udaljuje točke, stoga taj gradijent predstavlja odbojnu silu.

Za izračun oba gradijenta će nam trebat $\frac{\partial v_{ij}}{\partial \boldsymbol{y_i}}$, pa izračunajmo prvo njega:

$$\frac{\partial v_{ij}}{\partial \boldsymbol{y_i}} = \frac{\partial (1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^{-1}}{\partial \boldsymbol{y_i}} =$$
$$= -1 \cdot \frac{1}{(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^2} \cdot \frac{\partial (1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})}{\partial \boldsymbol{y_i}} =$$
$$= -1 \cdot \frac{1}{(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^2} \cdot a \cdot b \cdot \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2(b-1)} \cdot \frac{\partial \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2}{\partial \boldsymbol{y_i}} =$$
$$= -1 \cdot \frac{1}{(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^2} \cdot a \cdot b \cdot \|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2(b-1)} \cdot 2(\boldsymbol{y_i} - \boldsymbol{y_j})^\top =$$
$$= -\frac{2ab\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2(b-1)}}{(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^2} \cdot (\boldsymbol{y_i} - \boldsymbol{y_j})^\top$$

Za privlačnu silu imamo sljedeće:

$$\frac{\partial w_{ij} \ln(v_{ij})}{\partial \boldsymbol{y_i}} = w_{ij} \cdot \frac{1}{v_{ij}} \cdot \frac{\partial v_{ij}}{\partial \boldsymbol{y_i}} =$$
$$= w_{ij} \cdot \cancel{(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})} \cdot -\frac{2ab\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2(b-1)}}{(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})\cancel{^2}} \cdot (\boldsymbol{y_i} - \boldsymbol{y_j})^\top =$$
$$= \frac{-2ab\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2(b-1)}}{1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}} w_{ij} (\boldsymbol{y_i} - \boldsymbol{y_j})^\top$$

Za odbojnu silu imamo sljedeće:

$$\frac{\partial (1 - w_{ij}) \ln(1 - v_{ij})}{\partial \boldsymbol{y_i}} = (1 - w_{ij}) \cdot \frac{1}{1 - v_{ij}} \cdot -1 \cdot \frac{\partial v_{ij}}{\partial \boldsymbol{y_i}}$$

Prije nego li nastavimo dalje, pojednostavimo prvo $\frac{1}{1-v_{ij}}$:

$$\frac{1}{1 - v_{ij}} = \frac{1}{1 - (1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})^{-1}} \cdot \frac{1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}}{1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}} =$$
$$= \frac{1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}}{\cancel{1} + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b} - \cancel{1}} =$$
$$= \frac{1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}}{a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}}$$

Vratimo se na odbojnu silu:

$$\frac{\partial (1 - w_{ij}) \ln(1 - v_{ij})}{\partial \boldsymbol{y_i}} =$$
$$= (1 - w_{ij}) \cdot \frac{\cancel{1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}}}{\cancel{a}\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b}} \cdot -1 \cdot -\frac{2\cancel{a}b\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2(b-1)}}{(1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})\cancel{^2}} \cdot (\boldsymbol{y_i} - \boldsymbol{y_j})^\top =$$
$$= \frac{2b}{\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^2 (1 + a\|\boldsymbol{y_i} - \boldsymbol{y_j}\|^{2b})} (1 - w_{ij}) (\boldsymbol{y_i} - \boldsymbol{y_j})^\top$$

Kako bismo izbjegli dijeljenje s nulom za jako bliske parove točaka $\boldsymbol{y_i}$ i $\boldsymbol{y_j}$, u nazivnik konačnog izraza za odbojnu silu dodajemo i $\epsilon$:

$$\begin{aligned} &\frac{\partial(1-w_{ij})\ln(1-v_{ij})}{\partial \boldsymbol{y_i}} = \\ &= \frac{2b}{(\epsilon + \|\boldsymbol{y_i}-\boldsymbol{y_j}\|^2)(1+a\|\boldsymbol{y_i}-\boldsymbol{y_j}\|^{2b})}(1-w_{ij})(\boldsymbol{y_i}-\boldsymbol{y_j})^\top \end{aligned} \tag{A.4}$$