Žilinská univerzita v Žiline
Fakulta riadenia a informatiky
Katedra informačných sietí

Reverzibilná extrakcia príznakov

Diplomová práca

Autor práce: Bc. Matúš Mrázik
Vedúci práce: prof. Ing. Martin Klimo, PhD.
Študijný odbor: Informačné systémy

Reg. č.: 815/2018 Žilina, 2. mája 2019

Čestné prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením vedúceho práce prof. Ing. Martina Klima, PhD. a že som uviedol všetku použitú literatúru.

V Žiline, 2. 5. 2019

Matúš Mrázik

Poďakovanie

Ďakujem vedúcemu práce, prof. Ing. Martinovi Klimovi, PhD., za odbornú pomoc a cenné rady pri vypracovávaní tejto práce.

Abstrakt

MRÁZIK, Matúš: *Reverzibilná extrakcia príznakov*. Diplomová práca. Žilinská univerzita v Žiline. Fakulta riadenia a informatiky. Katedra informačných sietí. Vedúci diplomovej práce: prof. Ing. Martin Klimo, PhD. Stupeň odbornej kvalifikácie: Inžinier. Fakulta riadenia a informatiky Žilina, 2019 – 82s.

Práca sa zaoberá porovnaním lineárnych a nelineárnych metód extrakcie príznakov na úlohe rozpoznávania rukou písaných číslic s použitím databázy MNIST, a neskôr skúma systémy detekcie cudzích vzorov, teda vzorov tried, ktoré nie sú obsiahnuté v trénovacej množine. Na začiatku analyzujeme súčasný stav rozpoznávania vzorov. Neskôr popisujeme lineárne metódy extrakcie (PCA, ICA a LDA) a nelineárne metódy extrakcie (neurónové siete, autoenkodéry) použité v práci. V praktickej časti prezentujeme výsledky porovnania metód extrakcie príznakov, pričom pri porovnaní sme použili pre klasifikáciu metódu najbližšieho suseda. Pri lineárnych metódach je porovnaná spoločná extrakcia vzorov všetkých tried a extrakcia vzorov každej triedy zvlášť. Potom porovnávame lineárne metódy s nelineárnymi metódami. Na konci sa zaoberáme problémom detekcie cudzích vzorov, kde sme navrhli dva systémy. Prvý z nich používa klasifikačnú neurónovú sieť, v druhom sú okrem klasifikačnej siete doplnkové autoenkodéry pre každú triedu. Výkonnosť týchto systémov je meraná na databáze Fashion MNIST ako množine cudzích vzorov.

Kľúčové slová: extrakcia príznakov, rozpoznávanie vzorov, detekcia cudzích vzorov, strojové učenie, analýza hlavných komponentov, analýza nezávislých komponentov, lineárna diskriminačná analýza, neurónová sieť, autoenkodér, MNIST.

Abstract

MRÁZIK, Matúš: *Reversible feature extraction*. Master thesis. University of Žilina. Faculty of Management Science and Informatics. Department of Information Networks. Tutor: prof. Ing. Martin Klimo, PhD. Qualification level: Master. Faculty of Management Science and Informatics Žilina, 2019 – 82p.

This thesis deals with comparison of linear and non-linear feature extraction methods on handwritten digit recognition task using the MNIST dataset, and later examines systems for detecting foreign patterns, i.e. patterns from classes not included in the training set. In the first part, we analyse the current state of pattern recognition. Later, we describe the linear feature extraction methods (PCA, ICA and LDA) and non-linear feature extraction methods (neural networks, autoencoders) used in the thesis. In the practical part, results of feature extraction methods comparison are presented, with nearest neighbour classification method used for classification. Linear methods are compared using extraction of patterns of all classes at once and extraction of patterns of each class separately. After that, linear methods are compared to non-linear methods. At the end, we examine the foreign pattern detection problem, where two systems are proposed. The first one uses a single classification neural network, the other one employs supplementary autoencoders for each class. Performance of the systems is measured on the Fashion MNIST database as foreign pattern set.

Keywords: feature extraction, pattern recognition, foreign pattern detection, machine learning, principal component analysis, independent component analysis, linear discriminant analysis, neural network, autoencoder, MNIST.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam skratiek

API	Application Programming Interface (Rozhranie pre programovanie aplikácií)
FT	Fourierova transformácia
ICA	Independent Component Analysis (Analýza nezávislých komponentov)
LDA	Linear Discriminant Analysis (Lineárna diskriminačná analýza)
NLICA	nelineárna ICA
PCA	Principal Component Analysis (Analýza hlavných komponentov)

1 Súčasný stav

V posledných desaťročiach sme zaznamenali obrovský pokrok v oblasti informačných a komunikačných technológií, ktoré sa stali neodmysliteľnou súčasťou našich každodenných životov. Dnes sa využívajú takmer všade. S rozšírením týchto technológií však prichádza aj oveľa väčšie množstvo dát.

Tieto dáta často pozostávajú z veľmi veľkého počtu premenných, teda ide o mnohorozmerné dáta. V oblastiach analýzy dát, ako napríklad rozpoznávanie vzorov či data mining, môže byť veľká rozmernosť dát problém. Ak napríklad rozpoznávame objekty z obrázkov s veľkosťou 80×80 pixelov, spracúvané dáta majú 6400 rozmerov. Ak sú obrázky farebné, tento počet je 3-krát väčší.

Vo väčšine prípadov však nie sú potrebné všetky tieto údaje. Z obrázkov sa napríklad dá pomocou rôznych metód extrahovať len tá časť, kde sa nachádza skúmaný objekt. Napriek tomu však dáta stále môžu mať príliš veľa rozmerov. Preto existujú metódy, ako rozmernosť dát zredukovať.

1.1 Úloha rozpoznávania vzorov

Rozpoznávanie vzorov hrá dôležitú úlohu v oblastiach analýzy dát a data miningu. Už niekoľko desaťročí je predmetom intenzívneho skúmania za účelom jednoduchšieho a efektívnejšieho využitia v praxi. S rýchlym vývinom hardvéru, ktorý je dnes omnoho výkonnejší, sa jeho použitie stáva rozšírenejším. Využíva sa napríklad v oblastiach ako počítačové videnie či rozpoznávanie hlasu.

Stále sú však v tejto oblasti výzvy, ktorým je potrebné sa venovať. Dáta, ktorých objem sa neustále zvyšuje, sú totiž často komplexné, nevyvážené, alebo iným spôsobom zašumené. Preto je pri rozpoznávaní vzorov často nevyhnutné dáta najskôr predspracovať.

Štandardná úloha rozpoznávania vzorov podľa [HP18] spočíva v rozdelení množiny objektov (vzorov)

$$O = \{o_1, o_2, ...\}$$

do C podmnožín objektov patriacich do tej istej triedy

$$O_1, O_2, ..., O_C$$

tak, že platí

$$O = \bigcup_{i=1}^{C} O_i \quad \text{a} \quad (\forall i, j \in \{1, 2, ..., C\}, i \neq j)\ O_i \cap O_j = \emptyset \tag{1.1}$$

To znamená, že každý objekt je priradený do práve jednej z podmnožín $O_1, O_2, ..., O_C$. Proces, ktorého výsledkom je rozdelenie objektov do podmnožín $O_1, O_2, ..., O_C$, sa nazýva klasifikácia. Klasifikácia je definovaná zobrazením

$$\Psi : O \to \Theta \tag{1.2}$$

kde $\Theta = \{O_1, O_2, ..., O_C\}$ je množina uvažovaných tried. Toto zobrazenie sa nazýva klasifikátor. Pre jednoduchosť sa často namiesto množiny tried v zobrazení Ψ používa množina $\Theta = \{1, 2, ..., C\}$ ich číselných označení.

Pri rozpoznávaní vzorov sa obvykle namiesto samotných objektov používa množina premenných, ktoré tieto objekty charakterizujú. Tieto premenné sa nazývajú vlastnosti (features), alebo príznaky. Príznaky sa získavajú extrakciou príznakov, ktorá je definovaná zobrazením z priestoru objektov O do priestoru príznakov X

$$\varphi : O \to X \tag{1.3}$$

nazývaným extraktor príznakov. Na základe tohto sa definuje klasifikátor príznakov ako zobrazenie z priestoru príznakov X do priestoru tried Θ

$$\psi : X \to \Theta \tag{1.4}$$

Klasifikátor Ψ sa dá pomocou dvoch predchádzajúcich zobrazení napísať ako $\Psi = \varphi \circ \psi$. To znamená, že zobrazenie $O \xrightarrow{\Psi} \Theta$ je možné rozdeliť na tieto dve zobrazenia realizované za sebou $O \xrightarrow{\varphi} X \xrightarrow{\psi} \Theta$.

Vo všeobecnosti platí, že klasifikátor Ψ nie je známy, a teda nevieme určiť, do ktorej triedy daný vzor patrí. Pri úlohách učenia s učiteľom (supervised learning), kam

rozpoznávanie vzorov, ktorému sa v tejto práci venujeme, patrí, sa však predpokladá, že klasifikátor Ψ je známy pre nejakú podmnožinu priestoru vzorov, ktorá sa nazýva trénovacia množina (learning set). Trénovacia množina je množina vzorov $L \subset O$, pre ktoré poznáme triedu, do ktorej patria. Cieľom pri úlohách rozpoznávania vzorov je nájsť klasifikátor $\Psi : O \to \Theta$ s využitím známeho zobrazenia $\Psi : L \to \Theta$. Pre nájdenie čo najlepšieho klasifikátora, ktorý dokáže správne určiť triedy všetkých vzorov, je potrebné mať dostatočne reprezentatívnu trénovaciu množinu.

1.2 Štruktúra systému rozpoznávania vzorov

Systém rozpoznávania vzorov zvyčajne pozostáva z troch hlavných častí. Najskôr sa dáta predspracujú tak, aby boli vhodné pre nasledujúce časti. Predspracovanie závisí od druhu dát (zvuk, obraz, ...). V práci sa mu podrobne venovať nebudeme. Ďalším krokom je extrakcia príznakov, kde sa extrahujú príznaky reprezentujúce pôvodné dáta v zmenšenom priestore. V poslednom kroku sa objekty podľa získaných príznakov klasifikujú do jednotlivých tried.

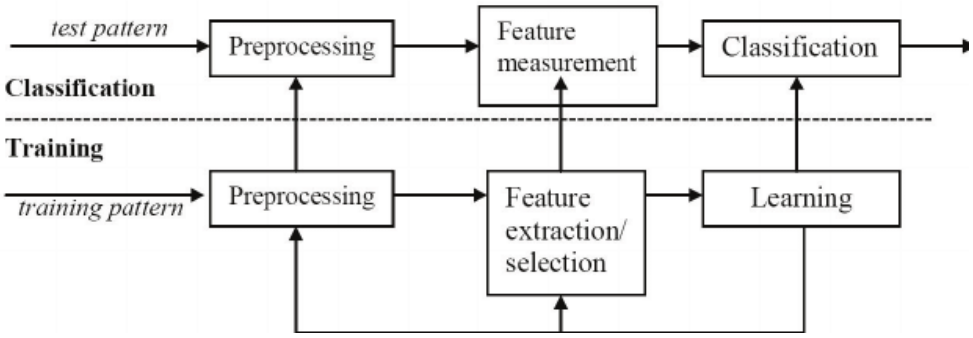
Systém sa učí rozpoznávať vzory vo fáze trénovania. Pri tom sa používa trénovacia množina vzorov, pre ktoré sú známe triedy, do ktorých patria. Trénovacia množina by mala byť čo najlepšia z hľadiska rozmanitosti, to znamená že by mala obsahovať čo najviac rôznych vzorov z celého priestoru vzorov.

Po fáze tréningu sa vo fáze testovania vyhodnocuje správnosť natrénovaného systému, kde sa používa testovacia množina. Tá by mala byť iná ako trénovacia množina, podobne by ale mala obsahovať čo najviac rôznych vzorov. Presnosť systému sa potom vypočíta ako pomer počtu úspešne klasifikovaných testovacích vzorov k celkovému počtu testovacích vzorov.

V nasledujúcich sekciách sú podrobnejšie popísané jednotlivé časti systému rozpoznávania vzorov.

1.2.1 Predspracovanie dát

Hodnoty jednotlivých zložiek zvyčajne majú rôzne rozsahy, v závislosti od toho, čo vyjadrujú. Niektoré zložky majú preto inú váhu ako ostatné. Pri určitých algoritmoch



Obr. 1.1: Štruktúra štandardného systému rozpoznávania vzorov s vyznačenou fázou trénovania (Obrázok prebratý z [NFP12])

spôsobuje takáto nevyváženosť problém, pretože zložky s vyššou váhou môžu "zatieniť" zložky, ktorých váha je nižšia. Preto sa tieto surové dáta musia najskôr predspracovať. Jednotlivé zložky sa škálujú tak, aby sa zjednotil rozsah ich hodnôt. Existujú dva prístupy: normalizácia hodnôt do jednotkového intervalu a štandardizácia, založená na zjednotení stredných hodnôt a štandardnej odchýlky. V oboch prístupoch sa využíva lineárna transformácia, aby sa zachovali charakteristické vlastnosti vzorov [HP18].

Normalizácia

Pri normalizácii sa pôvodné hodnoty lineárne transformujú do intervalu $[0, 1]$ alebo do intervalu $[-1, 1]$. Predpokladajme, že vzory sú popísané N zložkami označenými ako $X_1, X_2, ..., X_N$. Pre každú zložku X_i označíme $x_{i,\min}$ a $x_{i,\max}$ ako najmenšiu a najväčšiu hodnotu tejto zložky zo všetkých vzorov v rámci množiny vzorov. Hodnota $x_{i,j}$ tejto zložky pre vzor o_j sa potom transformuje do intervalu $[0, 1]$ takto:

$$a_{i,j} = \frac{x_{i,j} - x_{i,\min}}{x_{i,\max} - x_{i,\min}} \tag{1.5}$$

Do intervalu $[-1, 1]$ sa hodnota transformuje takto:

$$b_{i,j} = 2 * \frac{x_{i,j} - x_{i,\min}}{x_{i,\max} - x_{i,\min}} - 1 \tag{1.6}$$

Štandardizácia

Pri štandardizácii (označovanej aj ako normalizácia na nulový stred) sa pracuje so strednou hodnotou danej zložky a jej štandardnou odchýlkou. Štandardizácia hodnoty $x_{i,j}$

zložky X_i pre vzor o_j sa realizuje podľa vzťahu:

$$u_{i,j} = \frac{x_{i,j} - \bar{x}_i}{\sigma_i} \tag{1.7}$$

kde $\bar{x}_i$ je stredná hodnota zložky X_i a σ_i je štandardná odchýlka tejto zložky. Ich odhady vypočítame:

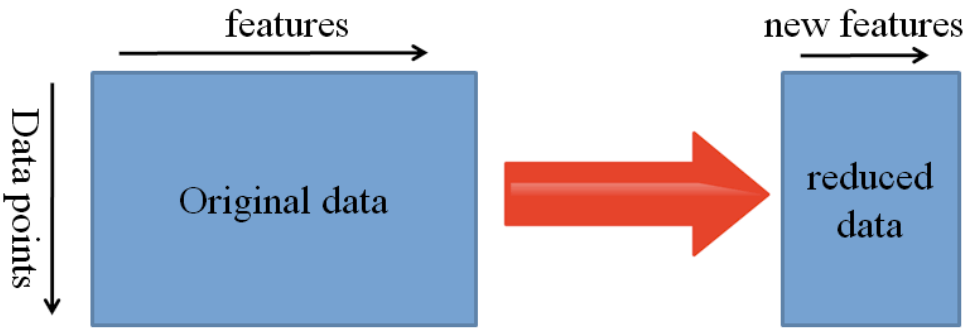
$$\bar{x}_i = \frac{1}{N}\sum_{j=1}^{N} x_{i,j}, \quad \sigma_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(x_{i,j} - \bar{x}_i)^2} \tag{1.8}$$

N je počet vzorov trénovacej množiny.

1.2.2 Extrakcia príznakov

Po predspracovaní dát sa zvyčajne podľa zobrazenia z (1.3) redukuje rozmernosť dát, pretože dáta často pozostávajú z veľmi veľkého počtu premenných. Redukcia rozmernosti (po anglicky dimensionality reduction) je transformácia mnohorozmerných dát na ich významnú reprezentáciu s nižšou rozmernosťou [MPH09]. Táto transformácia sa snaží z pôvodných dát zachovať čo najviac informácie potrebnej pre danú úlohu. Redukcia rozmernosti je dôležitá vo veľa oblastiach, najmä v tých, ktoré trpia tzv. "prekliatím rozmernosti" ("curse of dimensionality") a inými nežiaducimi vlastnosťami vyplývajúcimi z veľkej rozmernosti dát.

Prekliatie rozmernosti je pojem používaný v oblastiach, ktoré pracujú s mnohorozmernými dátami, ako napríklad oblasť strojového učenia. Hovorí o tom, že s rastúcim počtom rozmerov dát nadmieru rastie zložitosť riešeného problému. Vyplýva to z toho, že počet možných konfigurácií množiny premenných rastie exponenciálne s rastúcim počtom premenných [GBC16].



Obr. 1.2: Redukcia rozmernosti

Zdroj: https://towardsdatascience.com/transfer-learning-946518f95666

Prečo redukovať rozmernosť dát

Aplikovanie redukcie rozmernosti na dáta sa snaží riešiť tento problém, ako aj mnohé iné. Tu sú uvedené najdôležitejšie dôvody použitia redukcie rozmernosti:

- **zníženie výpočtovej zložitosti** – zníženie rozmernosti dát môže výrazne urýchliť výpočet. Keďže mnohorozmerné dáta sa používajú v takmer každej oblasti, zníženie počtu rozmerov môže výrazne pomôcť znížiť výpočtový čas, čo je výhodné najmä ak je čas výpočtu dôležitý, napríklad v systémoch pracujúcich v reálnom čase.

- **zníženie pamäťových nárokov** – uchovanie menšieho množstva dát vyžaduje menej miesta na disku.

- **vizualizácia** – pre človeka je nemožné predstaviť si dáta vo viac ako trojrozmernom priestore. Redukcia rozmernosti vhodnou metódou umožňuje vizuálne analyzovať štruktúru dát.

- **odstránenie prebytočných zložiek** – v pôvodných dátach sa často nachádzajú zložky, ktoré sú nadbytočné. Ak je napríklad v dátach viac zložiek, ktoré sú korelované, nie sú potrebné všetky tieto zložky. Redukciou rozmernosti sa vyberú tie zložky, ktoré sú relevantné pre riešenú úlohu.

Delenie metód redukcie rozmernosti

Redukcia rozmernosti sa robí dvomi základnými prístupmi:

- **výber príznakov (feature selection)** – pri výbere príznakov sa z pôvodných dát vyberie podmnožina premenných, ktorá tieto dáta podľa daných kritérií najlepšie reprezentuje.

- **extrakcia príznakov (feature extraction)** – pri extrakcii príznakov sa pôvodné dáta transformujú do menej rozmerného priestoru.

Počas rokov vzniklo veľa metód redukcie rozmernosti. Ich základným delením je delenie podľa toho, ako sa výsledné premenné (príznaky) v novom priestore so zníženým počtom rozmerov získavajú. Takto sa delia na:

- **lineárne** – sem patria napríklad:
  - Principal Component Analysis (Analýza hlavných komponentov, PCA)
  - Linear Discriminant Analysis (Lineárna diskriminačná analýza, LDA)
  - Independent Component Analysis (Analýza nezávislých komponentov, ICA)
- **nelineárne** – medzi nelineárne patria napríklad:
  - nelineárna PCA
  - Isomap
  - neurónové siete, napríklad:
    - autoenkodér
    - samoorganizujúca sa mapa (známa aj ako Kohonenova mapa)

V tejto práci sa kvôli hlavným aplikáciám pri rozpoznávaní vzorov bude hovoriť o redukcii rozmernosti ako o extrakcii príznakov.

Pri rozpoznávaní vzorov, ktoré rozpoznáva aj človek, bolo prirodzené, že systémy extrakcie príznakov sa inšpirovali ľudskými zmyslami. Pri rozpoznávaní zvukov sa použila spektrálna analýza tak, ako mení bazilárna membrána priebeh tlaku vo vnútornom uchu na podráždenie vláskových nervových zakončení, alebo pri rozpoznávaní obrazu sa použili hranové detektory ako v očnej neurónovej sieti. Matematické modely extrakcie príznakov sa opierali o lineárne modely, ktorých návrh je úplne zvládnutý. Zásadný pokrok však prinieslo použitie nelineárnych modelov, najmä hlbokých neurónových sietí. Nedostatok vhodnej teórie pre návrh nelineárnych systémov bol nahradený metódami strojového učenia. Pri dostatočne veľkom objeme trénovacích dát je možné numerickými optimalizačnými metódami identifikovať parametre aj zložitejších nelineárnych systémov.

1.2.3 Klasifikácia

Tu sa vzory reprezentované extrahovanými príznakmi zaraďujú do jednotlivých tried. Výstupom klasifikátora zvyčajne nebýva konkrétna trieda, do ktorej je vzor zaradený.

Namiesto toho klasifikátor určí pre vopred definované akcie ich váhy. Akcia je definovaná pre každú triedu (napríklad vzor patrí alebo nepatrí do danej triedy). Ako váha sa najčastejšie používa miera príslušnosti do triedy.

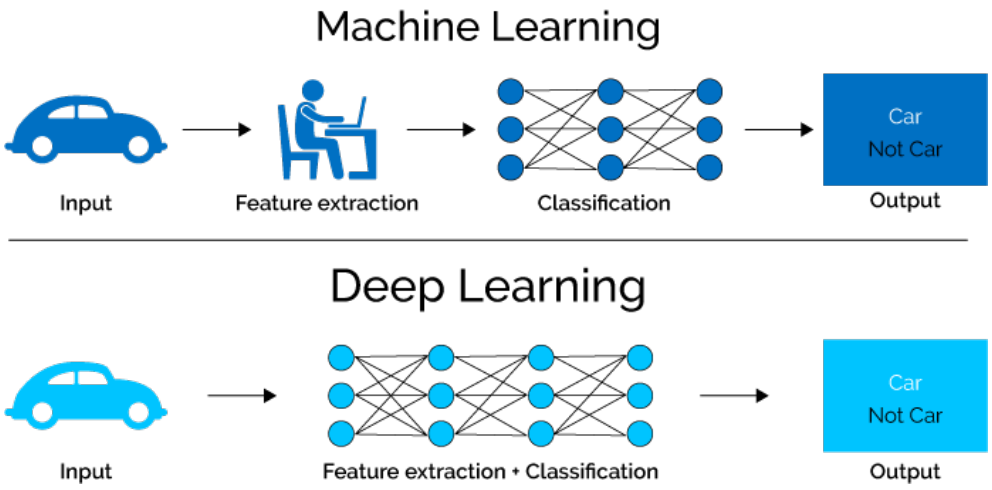
Na základe váh akcií sa v post-processingu robí rozhodnutie o triede, do ktorej bude vzor zaradený. Ak je to potrebné, pred rozhodnutím sa tieto váhy ešte upravia (napríklad ak súčet mier nie je 1). Post-processing nebýva súčasťou klasifikátora, zvyčajne sa ale uvádza ako súčasť klasifikácie.

Časti extrakcie príznakov a klasifikácie bývajú v tradičných algoritmoch strojového učenia samostatné – najskôr sa extrahujú príznaky a potom prebieha klasifikácia. V algoritmoch hlbokého strojového učenia (deep learning) sa extrakcia príznakov vykonáva automaticky ako súčasť algoritmu. Na obrázku 1.3 je zobrazený rozdiel medzi klasickým strojovým učením a hlbokým učením.

V tejto práci uprednostňujeme oddelenú extrakciu príznakov a klasifikáciu, pretože takáto štruktúra je lepšia z pohľadu vysvetliteľnosti výsledkov rozpoznávania.

1.3 Klasifikácia metódou najbližšieho suseda

Pre úlohu klasifikácie v tejto práci použijeme metódu najbližšieho suseda. Klasifikácia metódou najbližšieho suseda je založená na podobnosti vzorov, pričom podobnosť vzorov s m premennými je vyjadrená obrátenou vzdialenostnou funkciou (metrikou)



Obr. 1.3: Rozdiel medzi tradičným strojovým učením a hlbokým učením

Zdroj:

https://ydsedu.com/2017/12/22/deep-learning-neural-network-a-subfield-of-machine-learning/

v m-rozmernom priestore reálnych čísel $\mathbb{R}^m$. To znamená, že čím menšia je vzdialenosť medzi dvoma vzormi, tým sú si podobnejšie.

Existuje veľký počet metrík, z ktorých sú najznámejšími a najpoužívanejšími Euklidova, Manhattanská a Čebyševova metrika. Tieto metriky budú použité aj v tejto práci. Výpočet vzdialeností pomocou týchto metrík je vyjadrený nasledovne:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_m - y_m)^2} \tag{1.9}$$

$$d_M(\mathbf{x}, \mathbf{y}) = |x_1 - y_1| + |x_2 - y_2| + ... + |x_m - y_m| \tag{1.10}$$

$$d_C(\mathbf{x}, \mathbf{y}) = \max(|x_1 - y_1|, |x_2 - y_2|, ..., |x_m - y_m|) \tag{1.11}$$

kde d_E, d_M a d_C označujú postupne Euklidovu, Manhattanskú a Čebyševovu metriku, $\mathbf{x}$ a $\mathbf{y}$ sú body patriace do $\mathbb{R}^m$.

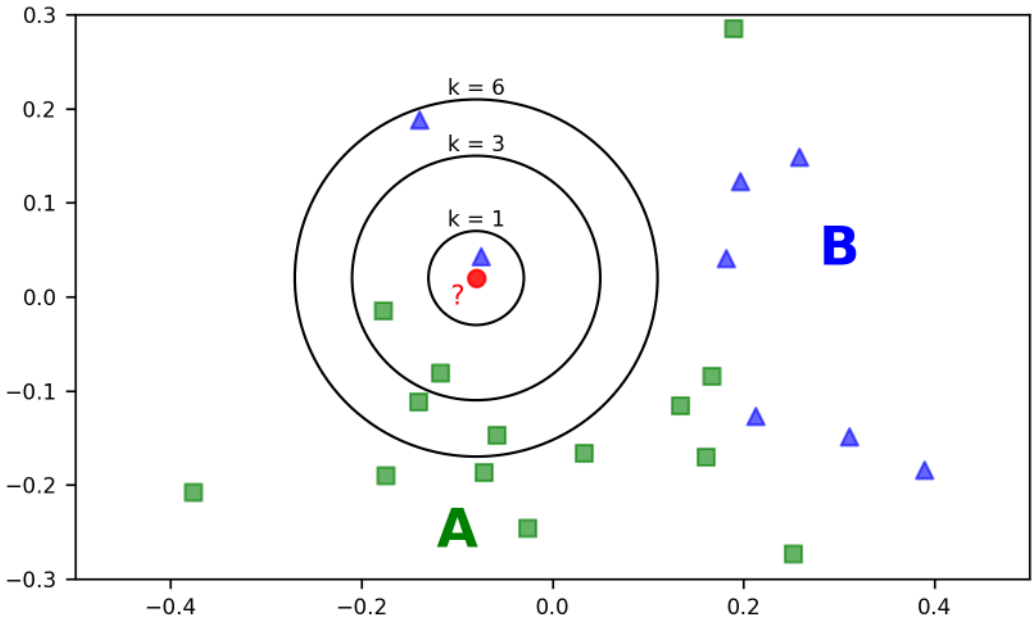
Klasifikácia neznámeho vzoru metódou najbližšieho suseda prebieha tak, že prehľadáme vzory trénovacej množiny a hľadáme vzor, ktorý je najbližšie (z pohľadu vzdialenosti podľa zvolenej metriky) k neznámemu vzoru. Neznámy vzor potom priradíme do triedy, do ktorej patrí nájdený vzor z trénovacej množiny. Toto pravidlo sa dá formálne napísať ako:

$$\Psi(p) = \Psi(r_{\min}) \quad \text{kde} \quad r_{\min} = \arg\min_{r \in L}\{d(p, r)\} = \arg\min_{r \in L}\{d(\mathbf{x}, \mathbf{y})\} \tag{1.12}$$

kde p je neznámy vzor, ktorý chceme klasifikovať, r je vzor z trénovacej množiny L, $\mathbf{x}$ a $\mathbf{y}$ sú vektory premenných reprezentujúce vzory p a r, a $r_{\min}$ je vzor z trénovacej množiny, ktorý je najbližšie k neznámemu vzoru p podľa metriky d.

Doteraz sme hovorili o hľadaní jedného vzoru, ktorý je najbližšie k neznámemu vzoru. Zovšeobecnením tohto prístupu je metóda k najbližších susedov. V takom prípade nájdeme k vzorov trénovacej množiny, ktoré sú najbližšie k neznámemu vzoru, a tento vzor priradíme do tej triedy, do ktorej patrí najviac z k nájdených vzorov.

V prípade, ak je najväčší počet vzorov rovnaký pre viacero tried, je potrebné použiť sekundárne rozhodovacie pravidlo. Najjednoduchším pravidlom býva náhodný výber triedy z tried s maximálnym počtom vzorov. Klasifikácia takýmto spôsobom však nemusí byť správna. Vhodnejšie je napríklad z tried s maximálnym počtom vzorov vybrať tú, do ktorej patrí najbližší vzor k neznámemu vzoru. Ďalšou možnosťou je vybrať triedu, pre ktorú je súčet vzdialeností vzorov od neznámeho vzoru minimálny. V práci budeme pre jednoduchosť používať metódu najbližšieho suseda s $k = 1$.



Obr. 1.4: Metóda najbližšieho suseda pre rôzne k: *pre* $k = 1$ *bude neznámy vzor zaradený do triedy B, pre* $k = 3$ *a* $k = 6$ *bude zaradený do triedy A*

Metóda najbližšieho suseda nevyžaduje trénovanie – to spočíva v uložení trénovacej množiny vzorov. Pri klasifikácii sa potom prehľadáva táto množina.

1.4 Metodika tvorby modelu

Úspešné vytvorenie správne fungujúceho systému rozpoznávania vzorov vyžaduje viac ako len dobré znalosti algoritmov. Je dôležité vedieť zvoliť správny algoritmus pre konkrétnu úlohu a vyhodnocovať výsledky vykonaných experimentov, aby bolo možné systém vylepšovať. Algoritmy majú rôzne parametre, ktoré ovplyvňujú ich fungovanie a výsledok. Výber algoritmu a nastavenie jeho parametrov závisí aj od použitých dát – počtu vzorov trénovacej množiny, počtu zložiek dát, ich hodnôt, atď.

V priebehu rokov boli vyskúšané a zdokumentované rôzne algoritmy pre rôzne typy úloh, takže pri návrhu systému rozpoznávania vzorov, ako aj všeobecne akéhokoľvek systému pracujúceho na princípe strojového učenia, sú známe kombinácie algoritmov a ich nastavení, ktoré fungujú pre danú úlohu. Preto nie je potrebné skúšať kombinácie všetkých algoritmov a ich nastavení.

Goodfellow, Bengio a Courville v [GBC16] uvádzajú jednoduchú metodiku, od zvládnutia ktorej závisí správne použitie algoritmu. Odporúčajú takýto praktický pro-

ces návrhu:

- Stanoviť si ciele - výber chybovej metriky a cieľovej hodnoty pre túto metriku, ktorú chceme dosiahnuť. Tieto ciele a chybové metriky by mali zodpovedať problému, ktorý má systém riešiť.
- Vytvoriť fungujúci systém, vrátane odhadu vhodnej metriky výkonnosti systému.
- Nastaviť systém tak, aby bolo možné detegovať slabé miesta vo výkonnosti. Diagnostikovať komponenty, ktoré fungujú horšie ako sa od nich očakáva, a či je slabší výkon spôsobený poddimenzovaním (underfitting), predimenzovaním (overfitting) alebo chybou v dátach či softvéri.
- Opakovane vykonávať inkrementálne zmeny ako napríklad zber nových dát, úprava parametrov, alebo zmena algoritmu, na základe pozorovaní systému.

2 Ciele práce

Základným cieľom práce pri jej zadaní bolo:

1. implementovať databázy vzorov (MNIST, cestné značky) a určiť entropie vzorov,
2. implementovať metódu najbližšieho suseda (KNN) pre klasifikáciu,
3. implementovať lineárne metódy extrakcie príznakov (FT, PCA, ICA),
4. implementovať nelineárne metódy extrakcie príznakov (autoenkodér, NLICA),
5. porovnať výkonnosť implementovaných metód pre spoločnú extrakciu a extrakciu po triedach.

Jadrom práce je porovnanie základných metód extrakcie príznakov s možnosťou spätnej rekonštrukcie vzorov. Počas riešenia diplomovej práce bol jej cieľ špecifickejšie vymedzený na vývoj metódy pre odhaľovanie vzorov tried, ktoré nie sú obsiahnuté v trénovacej množine vzorov (sample bias).

Podľa pokynov vedúceho diplomovej práce bolo určenie entropie vzorov zo zadania vypustené. Takisto boli vypustené niektoré metódy extrakcie príznakov – Fourierova transformácia (FT) a nelineárna ICA (NLICA). Databáza cestných značiek nebola použitá.

3 Lineárne metódy extrakcie príznakov

V nasledujúcich dvoch kapitolách budú bližšie popísané jednotlivé metódy extrakcie príznakov použité v tejto práci. V tejto kapitole sú popísané lineárne metódy.

3.1 Analýza hlavných komponentov

Analýza hlavných komponentov, po anglicky Principal Component Analysis (PCA), je jednou z najstarších a najznámejších metód redukcie rozmernosti dát. Využíva lineárnu transformáciu dát pozostávajúcich z veľkého počtu premenných, ktoré môžu byť korelované, na novú množinu premenných, nazývaných hlavné komponenty (principal components), ktoré sú nekorelované. V týchto hlavných komponentoch sa snaží zachovať čo najviac rozptylu z pôvodných premenných. Hlavné komponenty sa zoraďujú zostupne podľa ich rozptylu.

Hoci pôvod štatistických techník je často náročné vystopovať, je všeobecne akceptovaným faktom, že ako prvý popísal techniku dnes známu ako PCA anglický matematik Karl Pearson v roku 1901 [Pea01] a neskôr, v roku 1933, americký matematik Harold Hotelling [Hot33].

3.1.1 Hlavné komponenty

Predpokladajme, že máme p premenných $X_1, X_2, X_3, ..., X_p$, a pre každú premennú poznáme n hodnôt. Premenné $X_1, X_2, X_3, ..., X_p$ označme ako stĺpce matice $\mathbf{X}$ s rozmermi $n \times p$. Chceme zredukovať počet premenných na m, kde $m < p$. Pri PCA je prvým krokom nájdenie tzv. hlavných komponentov. Každý hlavný komponent je lineárnou

kombináciou premenných $X_1, X_2, X_3, ..., X_p$.

Ako už bolo spomenuté, hlavné komponenty sú zoradené zostupne podľa rozptylu. To znamená, že prvým hlavným komponentom bude normalizovaná lineárna kombinácia premenných

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p \tag{3.1}$$

ktorá má najväčší rozptyl. Normalizovaná znamená, že platí $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. Prvky $\phi_{11}, \phi_{21}, ..., \phi_{p1}$ sa nazývajú komponentové váhy prvého hlavného komponentu. Komponentové váhy spoločne vytvárajú váhový vektor tohto hlavného komponentu, $\phi_1 = (\phi_{11} \ \ \phi_{21} \ \ ... \ \ \phi_{p1})^T$. Suma druhých mocnín týchto váh sa obmedzuje na 1, pretože inak by rozptyl mohol byť nekonečne veľký.

Pri výpočte prvého hlavného komponentu predpokladáme, že stĺpce matice $\mathbf{X}$ majú strednú hodnotu rovnú 0. Potom hľadáme lineárnu kombináciu

$$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + ... + \phi_{p1}x_{ip} \tag{3.2}$$

ktorá má najväčší rozptyl pre $i = 1, ..., n$, pričom musí platiť $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. Hodnoty x_{ij} sú hodnoty matice $\mathbf{X}$. Pri hľadaní váhového vektora hlavného komponentu teda riešime optimalizačnú úlohu

$$\underset{\phi_{11},\phi_{21},...,\phi_{p1}}{\text{maximalizuj}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1}x_{ij} \right)^2 \right\} \quad \text{za podmienky} \quad \sum_{j=1}^{p} \phi_{j1}^2 = 1 \tag{3.3}$$

Hodnoty $z_{11}, z_{21}, ..., z_{n1}$ sa označujú ako komponentové skóre prvého hlavného komponentu.

Po vypočítaní prvého hlavného komponentu sa druhý hlavný komponent Z_2 počíta ako lineárna kombinácia premenných $X_1, ..., X_p$ s najväčším rozptylom zo všetkých lineárnych kombinácií, ktoré sú nekorelované so Z_1. Komponentové skóre druhého hlavného komponentu vyjadruje vzťah

$$z_{i2} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + ... + \phi_{p2}x_{ip} \tag{3.4}$$

Všeobecne sa dá napísať, že k-tý hlavný komponent Z_k je lineárna kombinácia premenných $X_1, X_2, ..., X_p$

$$Z_k = \phi_{1k}X_1 + \phi_{2k}X_2 + ... + \phi_{pk}X_p \tag{3.5}$$

s najväčším rozptylom zo všetkých lineárnych kombinácií, ktoré sú nekorelované s $Z_1, Z_2, ..., Z_{k-1}$. Jeho komponentové skóre vyjadruje vzťah

$$z_{ik} = \phi_{1k}x_{i1} + \phi_{2k}x_{i2} + ... + \phi_{pk}x_{ip} \tag{3.6}$$

Pri hľadaní hlavného komponentu Z_k je podmienka nekorelácie s $Z_1, ..., Z_{k-1}$ ekvivalentná s podmienkou ortogonálnosti smeru vektora ϕ_k voči smerom vektorov $\phi_1, ..., \phi_{k-1}$ (Karhunen–Loèvova transformácia).

Potom sú váhové vektory $\phi_1, \phi_2, ...$ hlavných komponentov usporiadanou postupnosťou vlastných vektorov kovariančnej matice pre maticu $\mathbf{X}$, a rozptyly komponentov sú jej vlastné hodnoty. Hlavných komponentov existuje najviac $min(n-1, p)$.

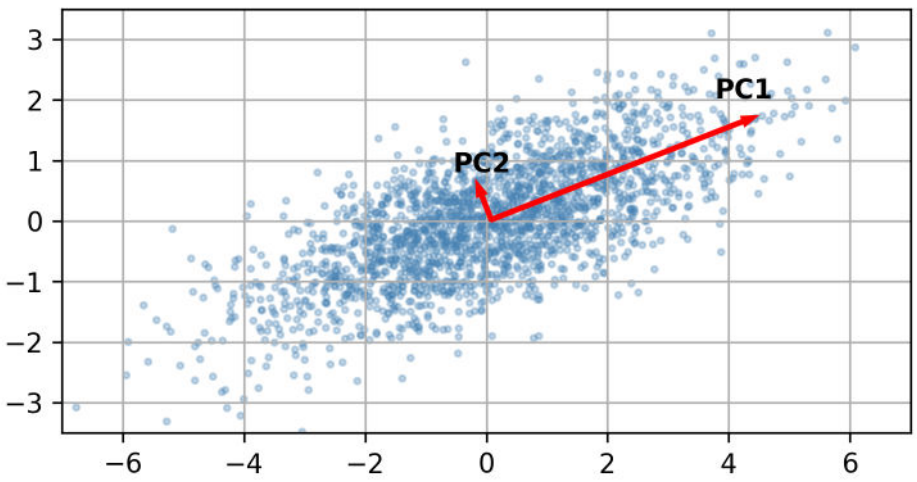
Na obrázku 3.1 sú zobrazené hlavné komponenty pre dve premenné.

3.1.2 Transformácia dát pomocou hlavných komponentov

Po nájdení m hlavných komponentov a ich váhových vektorov $\phi_1, ..., \phi_m$ dostaneme maticu $\mathbf{W} = (\phi_1 \ \ \phi_2 \ \ ... \ \ \phi_m)$ s rozmermi $p \times m$, kde vektor ϕ_i je i-tým stĺpcom matice $\mathbf{W}$. Ak máme maticu dát $\mathbf{X}$ s rozmermi $n \times p$, kde p je počet premenných a n je počet vzorov, pomocou vzťahu

$$\mathbf{X}_T = \mathbf{X}\mathbf{W} \tag{3.7}$$

tieto dáta transformujeme do m-rozmerného priestoru. Matica $\mathbf{X}_T$ je maticou extrahovaných príznakov a má rozmery $n \times m$.



Obr. 3.1: Hlavné komponenty pre dve premenné

3.1.3 Príprava dát pre PCA

Ako už bolo povedané, pri hľadaní hlavných komponentov je potrebné jednotlivé premenné (stĺpce matice $\mathbf{X}$) upraviť tak, aby mali strednú hodnotu rovnú 0. Pre každý stĺpec sa vypočíta stredná hodnota a odčíta sa od všetkých hodnôt v stĺpci. Tieto stredné hodnoty sa odčítavajú aj od dát pri transformácii a, pokiaľ robíme spätnú transformáciu, opäť pripočítavajú k spätne transformovaným dátam.

Pri použití PCA závisí výsledok aj od toho, či boli jednotlivé premenné škálované (každá násobená inou konštantou). V tomto sa PCA líši od iných metód extrakcie, kde škálovanie premenných nemá na výsledok vplyv.

3.2 Analýza nezávislých komponentov

Analýza nezávislých komponentov, v angličtine nazývaná Independent Component Analysis (ICA), je metóda extrakcie príznakov, ktorá, podobne ako PCA, hľadá množinu komponentov z pôvodných dát. ICA však hľadá komponenty, ktoré sú nielen štatisticky nekorelované, ale aj nezávislé.

Techniku ICA, hoci nie pod týmto názvom, predstavili na začiatku 80-tych rokov Jeanny Hérault, Christian Jutten a Bernard Ans [HA84; HJA85]. ICA však nezískala veľkú pozornosť až do polovice 90-tych rokov, keď Anthony J. Bell and Terrence J. Sejnowski predstavili rýchly a efektívny ICA algoritmus na princípe infomax [BS97]. O niekoľko rokov neskôr bol predstavený algoritmus FastICA, ktorý vyvinuli Aapo Hyvärinen a Erkki Oja [HO00].

3.2.1 Hľadanie nezávislých komponentov

Ak máme maticu dát $\mathbf{X}$ s rozmermi $n \times p$, kde p je počet premenných a n je počet pozorovaní týchto premenných, predpokladajme, že boli vygenerované ako lineárna kombinácia nezávislých komponentov

$$\mathbf{X} = \mathbf{SA} \tag{3.8}$$

kde $\mathbf{S}$ je matica nezávislých komponentov s rozmermi $n \times m$, kde m je počet komponentov. Stĺpce matice $\mathbf{S}$ sú jednotlivé komponenty. Maticu $\mathbf{A}$ ani maticu $\mathbf{S}$ nepoznáme,

vieme však, že komponenty majú byť nezávislé.

Nezávislosť komponentov

ICA sa spolieha na nezávislosť komponentov, čo je omnoho silnejšia vlastnosť ako nekorelovanosť. Jedným z dôvodov, prečo je nezávislosť silnejšia vlastnosť ako nekorelovanosť, je, že nezávislosť implikuje nelineárnu nekorelovanosť: Ak dva vektory s_1 a s_2 sú nezávislé, potom ľubovoľné nelineárne transformácie $g(s_1)$ a $h(s_2)$ sú nekorelované (ich kovariancia je rovná 0). Naproti tomu, ak sú dve náhodné premenné nekorelované (ale nie nezávislé), takéto nelineárne transformácie vo všeobecnosti nemajú nulovú kovarianciu.

Od komponentov sa takisto vyžaduje, aby mali negaussovské rozdelenia pravdepodobnosti. Podľa centrálnej limitnej vety totiž suma náhodných premenných s iným ako gaussovým rozdelením je bližšie ku gaussovmu rozdeleniu ako pôvodné premenné. Z toho vyplýva, že akákoľvek lineárna kombinácia $y = \sum_i b_i x_i$ pôvodných premenných, ktorá je zároveň aj lineárnou kombináciou nezávislých komponentov, bude maximálne negaussovská, ak sa bude rovnať niektorému z nezávislých komponentov. Ak by bola súčtom dvoch alebo viacerých nezávislých komponentov, bola by podľa centrálnej limitnej vety bližšie ku gaussovmu rozdeleniu.

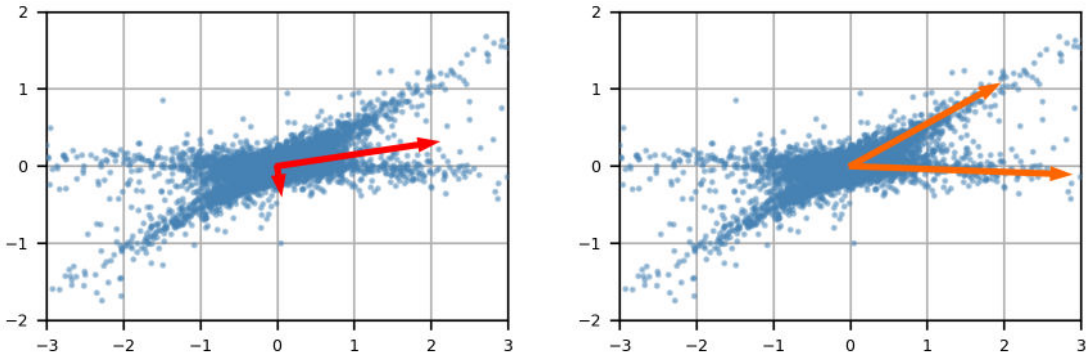
Odhad komponentov

V [HKO01] sú uvedené tieto princípy odhadu ICA na základe požiadaviek na komponenty:

1. **nelineárna nekorelovanosť** - nájdenie matice $\mathbf{W}$ takej, aby pre všetky $i \neq j$ boli komponenty s_i a s_j nekorelované, a aby aj transformované komponenty $g(s_i)$ a $h(s_j)$ boli nekorelované, pričom g a h sú vhodne zvolené nelineárne funkcie.

2. **maximálna negaussovosť** - nájdenie lokálnych maxím negaussovosti lineárnej kombinácie $y = \sum_i b_i x_i$ za podmienky, že rozptyl y je konštantný. Každé lokálne maximum zodpovedá jednému nezávislému komponentu.

Matica $\mathbf{W}$ predstavuje inverznú maticu k matici $\mathbf{A}$ z (3.8). Používa sa pri ex-



(a) Hlavné komponenty PCA (b) Nezávislé komponenty ICA

Obr. 3.2: Rozdiel medzi PCA a ICA

trakcii príznakov z pôvodných dát podobne ako pri PCA:

$$\mathbf{X}_T = \mathbf{X}\mathbf{W} \quad \mathbf{W} = \mathbf{A}^{-1} \tag{3.9}$$

Na spätnú transformáciu sa používa matica $\mathbf{A}$ podľa (3.8).

3.2.2 Príprava dát pre ICA

Nulový stred

Podobne ako pri PCA, aj pri ICA je potrebné, aby mali jednotlivé stĺpce matice $\mathbf{X}$ strednú hodnotu 0. Toto implikuje nulový stred aj pre maticu $\mathbf{S}$. Pri ICA sa však toto predspracovanie používa najmä pre zjednodušenie algoritmu, keďže stredné hodnoty je možné odhadnúť.

Bielenie

Ďalším spôsobom predspracovania dát pre ICA je tzv. bielenie pôvodných premenných. To znamená, že pred aplikovaním algoritmu ICA (po normalizácii na nulový stred) sa pôvodná matica $\mathbf{X}$ lineárne transformuje tak, aby jednotlivé premenné boli nekorelované a aby ich rozptyly boli rovné 1.

Bieliaca transformácia je vždy možná. Populárnou metódou bielenia je použitie dekompozície vlastných hodnôt – po anglicky eigenvalue decomposition (EVD), ktorá je v princípe zhodná s PCA. Pre viac informácií viď [HO00] a [HKO01].

3.2.3 Algoritmus FastICA

V tejto práci budeme používať algoritmus FastICA. Algoritmus vyvinuli Aapo Hyvärinen a Erkki Oja [HO00]. Ide o veľmi populárny ICA algoritmus, ktorého výhoda je nízka výpočtová a pamäťová náročnosť. Dokáže bežať paralelne na viacerých procesoroch. FastICA predpokladá, že dáta sú vycentrované a vybielené.

Extrakcia jedného komponentu

Iteratívny algoritmus hľadá smer, teda jednotkový vektor $\mathbf{w}$, pre ktorý projekcia $\mathbf{w}^T\mathbf{x}$ maximalizuje negaussovosť. Na meranie negaussovosti využíva FastICA nekvadratickú nelineárnu funkciu $f(u)$, jej prvú deriváciu $g(u)$ a jej druhú deriváciu $g'(u)$. Najčastejšie sa používajú funkcie

$$f(u) = \log\cosh(u) \qquad g(u) = \tanh(u) \qquad g'(u) = 1 - \tanh^2(u) \tag{3.10}$$

ktoré sú vhodné pre všeobecné účely, alebo funkcie

$$f(u) = -e^{-u^2/2} \qquad g(u) = ue^{-u^2/2} \qquad g'(u) = (1-u^2)e^{-u^2/2} \tag{3.11}$$

ktoré sú robustnejšie.

Základná postupnosť krokov algoritmu FastICA je nasledovná:

1. Zvoľ počiatočný (náhodný) váhový vektor $\mathbf{w}$.
2. Priraď $\mathbf{w}^+ = E\{\mathbf{x}g(\mathbf{w}^T\mathbf{x})\} - E\{g'(\mathbf{w}^T\mathbf{x})\}\mathbf{w}$.
3. Priraď $\mathbf{w} = \mathbf{w}^+/\|\mathbf{w}^+\|$.
4. Ak algoritmus neskonvergoval, choď na 2.

Algoritmus skonverguje, ak stará a nová hodnota $\mathbf{w}$ majú ten istý smer, čiže ak ich vektorový súčin je takmer rovný 1. Implementácie FastICA umožňujú nastaviť parameter ϵ, ktorý vyjadruje toleranciu rozdielu medzi vektorovým súčinom a 1. Ak je menší ako ϵ, algoritmus končí.

Extrakcia viacerých komponentov

Vyššie popísaný algoritmus odhaduje iba jeden váhový vektor, ktorým sa extrahuje jeden nezávislý komponent. Pre odhad viacerých komponentov je potrebné použiť tento algoritmus s viacerými vektormi $\mathbf{w}_1, ..., \mathbf{w}_n$.

Aby sa zabránilo konvergencii viacerých vektorov k tomu istému maximu, musíme dekorelovať výstupy $\mathbf{w}_1^T\mathbf{x}, ..., \mathbf{w}_n^T\mathbf{x}$ po každej iterácii. Existujú dva spôsoby, ako to dosiahnuť.

Prvý spôsob, ako dosiahnuť dekoreláciu, je deflačná schéma. To znamená, že jednotlivé nezávislé komponenty odhadujeme postupne jeden po druhom. Keď odhadneme k komponentov, alebo k vektorov $\mathbf{w}_1, ..., \mathbf{w}_k$, spustíme algoritmus pre odhad jedného vektora $\mathbf{w}_{k+1}$, po každej iterácii odčítame od $\mathbf{w}_{k+1}$ "projekcie" $\mathbf{w}_{k+1}^T\mathbf{w}_j\mathbf{w}_j, j = 1, ..., k$ predtým odhadnutých k vektorov, a potom normalizujeme $\mathbf{w}_{k+1}$:

1. Priraď $\mathbf{w}_{k+1} = \mathbf{w}_{k+1} - \sum_{j=1}^{k} \mathbf{w}_{k+1}^T\mathbf{w}_j\mathbf{w}_j$
2. Priraď $\mathbf{w}_{k+1} = \mathbf{w}_{k+1}/\sqrt{\mathbf{w}_{k+1}^T\mathbf{w}_{k+1}}$

V niektorých prípadoch je však vhodnejšie použiť symetrickú dekoreláciu, kde žiadny vektor nie je uprednostňovaný pred ostatnými. To sa dá dosiahnuť napríklad iteratívnym algoritmom

1. Priraď $\mathbf{W} = \mathbf{W}/\sqrt{\|\mathbf{W}\mathbf{W}^T\|}$
2. Priraď $\mathbf{W} = \frac{3}{2}\mathbf{W} - \frac{1}{2}\mathbf{W}\mathbf{W}^T\mathbf{W}$
3. Ak algoritmus neskonvergoval, opakuj 2.

kde $\mathbf{W}$ je matica $(\mathbf{w}_1, ..., \mathbf{w}_n)^T$ vektorov.

3.3 Lineárna diskriminačná analýza

Lineárna diskriminačná analýza, po anglicky Linear Discriminant Analysis (LDA), hľadá lineárnu kombináciu premenných (tzv. diskriminantorov), ktorá najlepšie charakterizuje alebo rozlišuje triedy objektov. Pri učení potrebuje poznať triedy objektov trénovacej množiny, ide teda o algoritmus učenia s učiteľom (supervised learning), na rozdiel od PCA a ICA.

LDA je zovšeobecnením metódy Fisher's Linear Discriminant, navrhnutej britským štatistikom a genetikom Ronaldom Fisherom v roku 1936 [Fis36]. Úzko súvisí s PCA, pretože PCA aj LDA hľadajú lineárnu kombináciu premenných, ktoré najlepšie popisujú dáta. LDA sa však pokúša modelovať rozdiely medzi triedami objektov, zatiaľ čo PCA rozdiely medzi triedami objektov neberie do úvahy.

3.3.1 Ako funguje LDA

Prvým krokom LDA je nájsť dve rozptylové matice - rozptylovú maticu medzi triedami (between class scatter matrix) a rozptylovú maticu v rámci tried (within class scatter matrix). Predpokladajme, že matica dát $\mathbf{X}$ obsahuje n vzorov, pričom každý z týchto vzorov patrí do jednej z c tried. Pre i-tú triedu označíme μ_i ako vektor stredných hodnôt

$$\mu_i = \frac{1}{n_i} \sum_{x_k \in X_i} x_k \tag{3.12}$$

kde n_i je počet vzorov v triede i a X_i je množina vzorov patriacich do triedy i, pričom x_k označuje k-tý vzor tejto triedy. Podobne definujeme vektor stredných hodnôt celej množiny dát:

$$\mu = \frac{1}{n} \sum_{i=1}^{c} n_i \mu_i = \frac{1}{n} \sum_{i=1}^{c} \sum_{x_k \in X_i} x_k \tag{3.13}$$

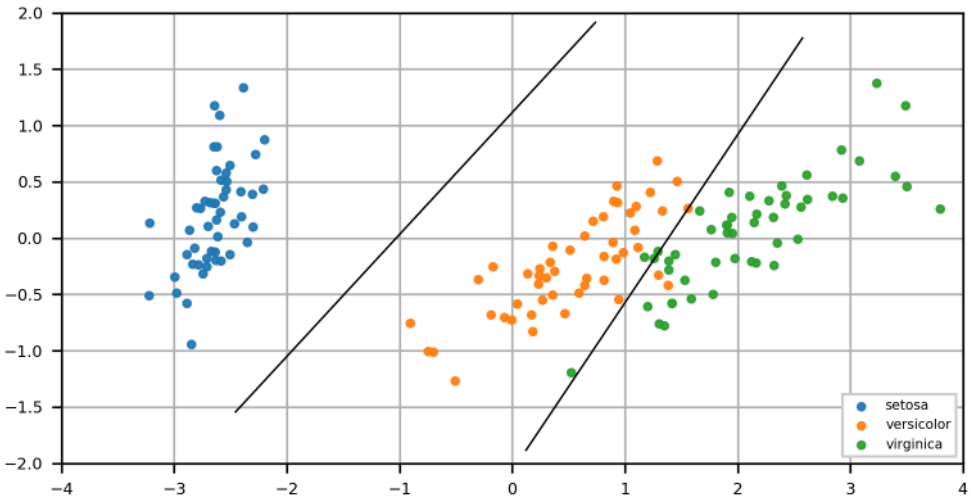
Na základe tohto definujeme rozptylovú maticu medzi triedami S_B a rozptylovú maticu v rámci tried S_W:

$$S_B = \sum_{i=1}^{c} n_i (\mu_i - \mu)(\mu_i - \mu)^T \tag{3.14}$$

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T \tag{3.15}$$

S_W reprezentuje rozptyl premenných okolo strednej hodnoty každej triedy a S_B reprezentuje rozptyl premenných okolo strednej hodnoty všetkých tried.

Cieľom je nájsť transformačnú maticu $\mathbf{W}$, ktorá maximalizuje $\frac{\det |S_B|}{\det |S_W|}$, teda pomer determinantov matíc S_B a S_W. Tento pomer je maximálny, keď stĺpcové vektory transformačnej matice $\mathbf{W}$ sú vlastné vektory matice $S_W^{-1} S_B$, kde S_W^{-1} je inverzná matica k matici S_W. Pre výpočet inverznej matice je potrebné, aby matica S_W bola regulárna. To je možné dosiahnuť použitím PCA pri predspracovaní dát.



Obr. 3.3: LDA pre Iris dataset

3.3.2 Využitie LDA

LDA dokáže extrahovať z množiny dát pozostávajúcej z c tried najviac $c-1$ príznakov. Preto sa používa najmä pri datasetoch s vyšším počtom tried (napríklad pri rozpoznávaní tvárí). Aj vtedy je však potrebné sa uistiť, že premenné sú nekorelované.

Ak dáta pozostávajú z menšieho počtu tried, používa sa LDA spolu s inou metódou extrakcie príznakov (napríklad PCA, ICA alebo iné), ktorou sa najskôr zredukuje rozmernosť dát a následne sa tieto dáta použijú v LDA.

LDA neslúži len ako metóda extrakcie príznakov, ale dá sa použiť aj na klasifikáciu.

4 Nelineárne metódy extrakcie príznakov

V tejto kapitole sú popísané nelineárne metódy extrakcie príznakov. V prvej sekcii budú popísané neurónové siete všeobecne, v druhej budú bližšie popísané autoenkodéry.

4.1 Neurónové siete

Neurónové siete, po anglicky označované ako Neural Networks, alebo Artificial Neural Networks, sú založené na simulácii funkcií biologických nervových systémov. Základnou motiváciou tohto prístupu je, že biologické systémy zvládajú zložité výpočty bez využitia explicitných kvantitatívnych operácií. Organizmy sa dokážu učiť postupne v čase. Predpokladá sa, že táto vlastnosť učenia odráža schopnosť veľkých skupín neurónov učiť sa pomocou vonkajších stimulov a zovšeobecňovať súvisiace inštancie signálu. Tieto vlastnosti biologických nervových systémov ich robia atraktívnymi pre modelovanie výpočtových metód navrhnutých pre spracovanie komplexných dát relatívne nezávisle od zadanej úlohy.

Výhodami biologických nervových systémov sú relatívna rýchlosť, s ktorou pracujú vďaka masívnemu paralelizmu, a ich robustnosť (odolnosť voči chybám) pri poškodení častí systému. Tieto výhody majú efekt aj v neurónových sieťach – poškodenie lokálnej časti neurónovej siete má väčšinou malý dosah na fungovanie siete ako celku, vďaka prirodzenej redundancii výpočtovej architektúry.

Neurónové siete označujú skôr skupinu modelov, ako jednu konkrétnu techniku. Každý typ neurónovej siete je optimalizovaný pre špecifickú množinu podmienok, analogicky so špecifickými funkciami rôznych častí mozgu.

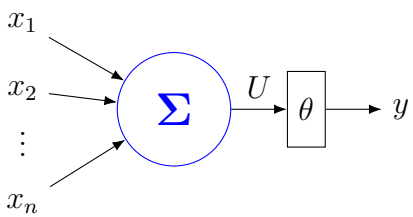
4.1.1 Neuróny

Mozgová kôra pozostáva z veľkého počtu navzájom prepojených buniek, nazývaných neuróny. Neuróny sú základné nervové bunky, ktoré formujú stavebné časti nervového systému. V ľudskom mozgu sa nachádza približne sto miliárd ($\approx 10^{11}$) neurónov.

Neurón prijíma signály od iných neurónov cez dendrity, ktoré slúžia ako prijímače vstupnej informácie, a odosiela výstupný signál cez axón. Axón sa na konci rozdeľuje na viacero vlákien, z ktorých každé končí synapsiou. Tie sa pripájajú k dendritom iných neurónov, prípadne k svalovému tkanivu. Mozog sa "učí" zmenou síl spojení medzi neurónmi alebo vytváraním nových spojení, prípadne ich odstraňovaním. Učenie prebieha postupne s rastúcim množstvom skúseností.

Myšlienka "umelej" neurónovej siete vychádza z modelu výpočtového stroja od McCullocha a Pittsa, ktorí zostrojili zjednodušenú abstrakciu aktivity neurónov v ľudskom mozgu [MP43]. McCullochov-Pittsov neurón pozostáva z viacerých vstupov (dendrity) a jedného výstupu (axón). Vstupy sa označujú $x_1, x_2, ..., x_n$, a každý z nich má hodnotu 0 (neaktívny) alebo 1 (aktívny). Signál pri každom vstupnom spojení závisí od toho, či je daná synapsia excitačná alebo inhibičná. Ak je niektorá synapsia inhibičná a vysiela hodnotu 1, neurón nevysiela signál (výstup je 0). Ak žiadna synapsia nie je inhibičná, vstupy sa sčítajú do celkovej excitácie $U = \sum_{i=1}^{n} x_i$, ktorá sa porovná s hraničnou hodnotou θ. Ak je $U \geq \theta$, výstupná hodnota y je 1 a neurón vysiela signál, inak je $y = 0$ a neurón signál nevysiela.

McCullochov-Pittsov neurón sa po anglicky označuje aj threshold logic unit (TLU), čo v preklade znamená logická jednotka hraničnej hodnoty. Dokáže spočítať len jednoduché logické funkcie s n parametrami, napríklad logický AND a logický OR. Spojením viacerých TLU do veľkých sietí s aspoň 2 vrstvami je možné počítať všetky ostatné logické funkcie.



Obr. 4.1: Schéma McCullochovho-Pittsovho neurónu

4.1.2 Perceptróny

Takýto model sa však ukázal ako nie veľmi presný odhad toho, ako funguje biologický systém. Sieť nemala žiadne nastaviteľné parametre alebo váhy, takže pre vyriešenie rôznych problémov bolo potrebné opakovane meniť štruktúru vstupov a hraničné hodnoty θ.

Tento problém sa snažil v roku 1958 vyriešiť Frank Rosenblatt, keď navrhol systém perceptrónov [Ros58]. Perceptróny sú siete modifikovaných McCullochových-Pittsových neurónov, kde ku každému vstupu x_i je priradená váha spojenia $w_i, i = 1, 2, ..., n$, ktorá nadobúda reálne hodnoty. Vstupy $x_1, x_2, ..., x_n$ môžu byť binárne alebo môžu tiež mať reálne hodnoty. Váhy s kladnými hodnotami ($w_i > 0$) zodpovedajú excitačným synapsiám, negatívne váhy ($w_i < 0$) zodpovedajú inhibičným synapsiám. Absolútna hodnota váhy ukazuje silu spojenia.

Pri perceptrónoch sa hodnota celkovej excitácie U počíta ako vážený súčet $U = \sum_{i=1}^{n} w_i x_i$, a výstupná hodnota y je 1, pokiaľ je $U \geq \theta$, v opačnom prípade je $y = 0$. Hraničná hodnota θ sa dá skonvertovať na 0 pridaním hraničného (bias) elementu $w_0 = -\theta$. Potom stačí hodnotu $U = \sum_{i=0}^{n} w_i x_i$ porovnať s 0, pričom $x_0 = 1$. Ak je $U \geq 0$, potom $y = 1$, inak $y = 0$.

Aktivačná funkcia

Výpočet výstupnej hodnoty môžeme napísať ako funkciu

$$f(u) = \begin{cases} 1 & \text{ak} \quad u \geq 0 \\ 0 & \text{ak} \quad u < 0 \end{cases} \tag{4.1}$$

Funkcia f sa nazýva aktivačná funkcia neurónu. Hodnotu y potom vypočítame ako $y = f(U) = f(\sum_{i=0}^{n} w_i x_i)$.

Každý neurón môže využívať inú aktivačnú funkciu f. Funkcia z (4.1) sa využíva najmä pri úlohách klasifikácie pre výstupné neuróny. V tabuľke 4.1 sú uvedené niektoré aktivačné funkcie.

Aktivačná funkcia	$f(u)$	Obor hodnôt
identická funkcia	u	$\mathbb{R}$
prahová funkcia	$I_{[u \geq 0]}$	$\{0, 1\}$
signum	$sign(u)$	$\{-1, 0, 1\}$
ReLU	$max(u, 0)$	$\mathbb{R}^+$
logistická funkcia (sigmoid)	$(1 + e^{-u})^{-1}$	$(0, 1)$
hyperbolický tangens (sigmoid)	$(e^u - e^{-u})/(e^u + e^{-u})$	$(-1, 1)$

Tabuľka 4.1: Prehľad aktivačných funkcií

4.1.3 Jednovrstvové perceptróny

Jedným zo spôsobov reprezentácie sietí prepojení neurónov je pomocou orientovaného acyklického grafu. Každá orientovaná hrana reprezentuje prepojenie medzi neurónmi, pričom jej orientácia vyjadruje smer toku informácií.

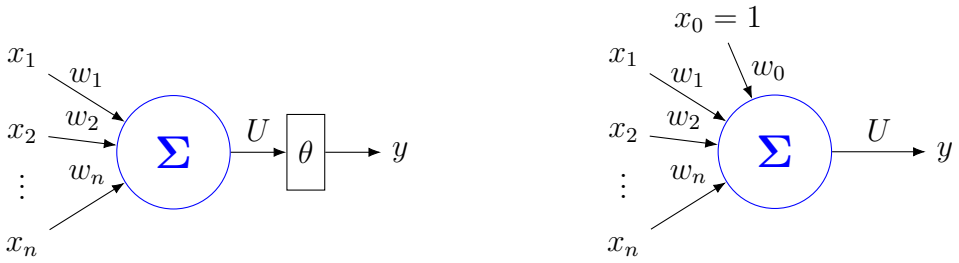
Najjednoduchší typ takéhoto grafu rozdeľuje vrcholy do dvoch skupín: n vstupných vrcholov $x_1, ..., x_n$ a m výstupných vrcholov $y_1, ..., y_m$. Vstupné vrcholy sa nazývajú aj zdrojové vrcholy, alebo vstupné premenné. V týchto vrcholoch sa nevykonávajú žiadne výpočty. Vstupné aj výstupné vrcholy môžu mať spojité hodnoty alebo diskrétne hodnoty (väčšinou binárne).

Hoci sú vrcholy rozdelené do dvoch skupín, takáto sieť sa nazýva jednovrstvová neurónová sieť, pretože iba vo výstupných vrcholoch prebiehajú výpočty. Vstupné vrcholy môžeme chápať ako "nultú" vrstvu, ktorá slúži len ako vstup pre prvú vrstvu.

Každé spojenie $x_i \longrightarrow y_j$ medzi vstupnými a výstupnými vrcholmi nesie váhu spojenia, w_{ij}, ktorá vyjadruje silu spojenia. Váhy môžu byť kladné, záporné alebo nulové – kladné vyjadrujú excitačné signály, záporné vyjadrujú inhibičné signály a nulové vyjadrujú neexistujúce spojenia.

Trénovanie

Rosenblatt ukázal, že perceptróny je možné natrénovať tak, aby vedeli rozpoznávať a klasifikovať objekty [Ros58]. Jeho trénovací algoritmus spočíva v postupnej klasifikácii objektov. Ak je výstup neurónu správny, váhy sa neupravujú. Ak je nesprávny, váhy daného neurónu sa upravia nasledovne:



(a) perceptrón s hraničnou hodnotou θ (b) ekvivalent s hraničným vstupom

Obr. 4.2: Schéma Rosenblattovho jednovrstvového perceptrónu

- ak je na výstupe 0, ale správna hodnota je 1, váhy sa zmenšia,
- ak je na výstupe 1, ale správna hodnota je 0, váhy sa zväčšia.

Obmedzenia perceptrónov

Napriek počiatočným veľkým očakávaniam sa možnosti perceptrónov ukázali ako veľmi obmedzené. Minsky a Papert v [MP69] ukázali, že perceptróny nedokážu riešiť lineárne neseparovateľné problémy. Klasickým príkladom takéhoto problému je funkcia XOR.

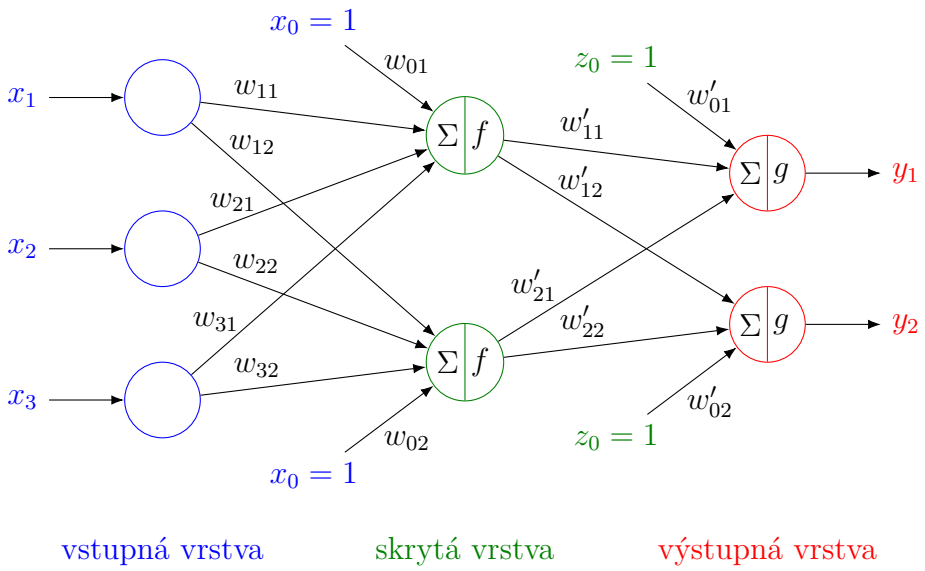
Ich návrhom, ako prekonať tieto obmedzenia, bolo "vrstvenie" perceptrónov a aplikácia nelineárnych transformácií pred skombinovaním transformovaných vážených vstupov. Kvôli nedostatku výpočtového výkonu sa však s týmito návrhmi nepracovalo.

4.1.4 Viacvrstvové perceptróny

Návrhy Minskyho a Paperta sa stali významnými až po príchode prvých superpočítačov v 80-tych rokoch a po objavení algoritmu učenia sa metódou spätného šírenia chýb (backpropagation algorithm).

Viacvrstvová dopredná neurónová sieť (perceptrón) je technika, ktorá mapuje vstupný vektor $\mathbf{X} = (x_1, ..., x_n)^T$ premenných na výstupný vektor $\mathbf{Y} = (y_1, ..., y_m)^T$ premenných. Medzi vstupmi a výstupmi sú tzv. "skryté" premenné usporiadané vo vrstvách. Skryté a výstupné premenné sa nazývajú aj neuróny.

Architektúra viacvrstvových perceptrónov je nasledovná: n vstupných vrcholov $x_1, ..., x_n$, jedna alebo viac vrstiev skrytých vrcholov a m výstupných vrcholov $y_1, ..., y_m$. Vrstvy skrytých vrcholov sa nazývajú skryté vrstvy (hidden layers). Ak má sieť L skrytých vrstiev, hovorí sa, že je to $(L + 1)$-vrstvová sieť. Vstupné vrcholy sa nepočítajú



Obr. 4.3: Príklad viacvrstvového perceptrónu

medzi vrstvy, keďže v týchto vrcholoch nie sú žiadne trénovateľné premenné. Neuróny v každej vrstve majú aktivačnú funkciu, ktorá určuje výstupnú hodnotu neurónu z kombinácie ováhovaných vstupov.

Na obrázku 4.3 je príklad viacvrstvového perceptrónu s jednou skrytou vrstvou s dvoma skrytými vrcholmi, vstupnou vrstvou s $n = 3$ vstupnými vrcholmi, a výstupnou vrstvou s $m = 2$ výstupnými vrcholmi.

Algoritmus učenia sa metódou spätného šírenia chýb

Algoritmus učenia sa metódou spätného šírenia chýb (po anglicky backpropagation algorithm) prvýkrát predstavil Werbos v [Wer74]. Vychádza z gradientovej metódy hľadania extrému funkcie viacerých premenných, v ktorej gradient je daný chybou odhadu hodnoty funkcie. Algoritmus pozostáva z dvoch krokov:

1. dopredné šírenie (forward propagation) vstupného vzoru od vstupnej vrstvy k výstupnej vrstve siete,

2. spätné šírenie (backpropagation) chybového vektora od výstupnej vrstvy k vstupnej vrstve siete.

Po kroku dopredného šírenia sa vypočítajú výstupné hodnoty, z ktorých sa vypočíta hodnota chybovej funkcie $E^q(\mathbf{W})$ matice váh $\mathbf{W}^q$ v kroku q. Chybová funkcia

môže byť ľubovoľná diferencovateľná funkcia. Ak hodnota chybovej funkcie nie je 0, je potrebná úprava váh o $\Delta\mathbf{W}^q$. Táto úprava sa aplikuje na všetky váhy w_{ik} spájajúce neurón k vo vrstve $l-1$ s neurónom i vo vrstve l.

$$\Delta w_{ik}^q = -\eta \frac{\partial E^q}{\partial w_{ik}} \tag{4.2}$$

Hodnota η sa po anglicky nazýva learning rate a vyjadruje rýchlosť učenia. Ak je η príliš veľké, algoritmus sa bude rýchlejšie približovať lokálnemu minimu, avšak ľahšie ho môže minúť. Ak je príliš malé, algoritmu môže oveľa dlhšie trvať dosiahnutie lokálneho minima. Po vypočítaní $\Delta\mathbf{W}^q$ sa váhy aktualizujú podľa

$$\mathbf{W}^{q+1} = \mathbf{W} + \Delta\mathbf{W}^q \tag{4.3}$$

Hlboké siete

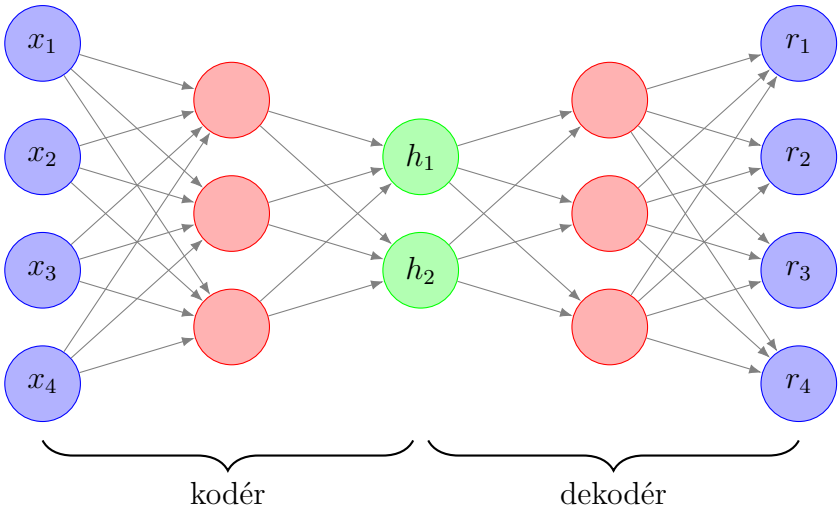
Hlboké siete, alebo hlboké neurónové siete, sú neurónové siete s viac ako jednou skrytou vrstvou. Pri trénovaní týchto sietí metódou spätného šírenia chýb nastáva problém, pretože v kroku spätného šírenia sú najviac ovplyvňované váhy neurónov blízko výstupnej vrstvy a váhy pri vstupnej vrstve sú ovplyvňované minimálne. To spôsobuje, že pridaním skrytých vrstiev sa výsledky nezlepšia, a navyše narastá zložitosť a čas potrebný na natrénovanie. Preto hlboké siete využívajú pri trénovaní metódy hlbokého strojového učenia. Medzi hlboké neurónové siete sa radia napríklad konvolučné siete a rekurentné siete.

4.2 Autoenkodéry

Autoenkodér je typ neurónovej siete, ktorá pozostáva z dvoch častí: kodéra reprezentovaného kódovacou funkciou $\boldsymbol{h} = f(\boldsymbol{x})$, ktorý kóduje vstupné dáta, a dekódera reprezentovaného dekódovacou funkciou $\boldsymbol{r} = g(\boldsymbol{h})$, ktorý rekonštruuje zakódované dáta.

Autoenkodér sa zvyčajne trénuje tak, aby rozdiel medzi $\boldsymbol{x}$ a $\boldsymbol{r}$ bol minimálny, pričom dôležitý je výstup kodéra $\boldsymbol{h}$, ktorý pri vhodne zvolenej architektúre dokáže z dát extrahovať relevantné informácie. Veľkosť $\boldsymbol{h}$ nie je obmedzená a zvyčajne sa určuje podľa typu autoenkodéra a jeho využitia.

Myšlienka autoenkodérov je súčasťou oblasti neurónových sietí už niekoľko desaťročí. Ich počiatky siahajú do 80-tych rokov minulého storočia [LeC87; BK88]. Auto-



Obr. 4.4: Schéma autoenkodéra

enkodéry sa zvyčajne využívali na extrakciu príznakov, ale v poslednej dobe sa dostali do popredia v oblasti generatívneho modelovania vďaka teoretickým prepojeniam medzi autoenkodérmi a modelmi latentných premenných, ako uvádzajú Goodfellow, Bengio a Courville v [GBC16].

Autoenkodéry sú špeciálnym typom dopredných neurónových sietí a pri ich trénovaní sa dajú použiť všetky techniky známe z všeobecných neurónových sietí. Na rozdiel od všeobecných dopredných sietí sa však autoenkodéry dajú trénovať aj pomocou recirkulácie [HM88], učiaceho algoritmu založeného na porovnávaní aktivácií siete pri pôvodnom vstupe s aktiváciami pri rekonštruovanom vstupe. Tento algoritmus však v práci používať nebudeme.

4.2.1 Typy autoenkodérov

Kopírovanie vstupu na výstup sa môže zdať zbytočné, ale výstup dekodéra zvyčajne nie je dôvodom pre použitie autoenkodéra. Pre nás je dôležitý výstup kodéra $\boldsymbol{h}$, pre ktorý chceme, aby extrahoval zo vstupov zaujímavé vlastnosti.

Na základe tohto môžeme autoenkodéry deliť podľa veľkosti $\boldsymbol{h}$. Ak má $\boldsymbol{h}$ menší rozmer ako vstup $\boldsymbol{x}$, nazýva sa takýto autoenkodér v angličtine undercomplete. Naopak, ak má $\boldsymbol{h}$ väčší rozmer ako $\boldsymbol{x}$, hovoríme o overcomplete autoenkodéri.

Každý z týchto typov má iné využitie. Autoenkodéry s menším rozmerom zakódovaných dát sa využívajú najmä pri kompresii dát, alebo pre účely extrakcie príznakov

pre klasifikačné úlohy. Autoenkodéry, ktoré kódujú vstup do viacrozmerného priestoru, sú vhodnejšie pre iné typy úloh. Všeobecne však platí, že výber architektúry závisí od konkrétnej úlohy. Nižšie sú popísané niektoré špecifické typy autoenkodérov.

Autoenkodér pre odstraňovanie šumu

Pri trénovaní autoenkodéru sa zvyčajne snažíme minimalizovať hodnotu stratovej funkcie

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) \tag{4.4}$$

Pri autoenkodéri pre odstraňnovanie šumu (po anglicky denoising autoencoder) sa namiesto tejto funkcie minimalizuje

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))) \tag{4.5}$$

kde $\tilde{\boldsymbol{x}}$ je kópia $\boldsymbol{x}$, ktorá bola poškodená nejakou formou šumu. Takýto autoenkodér sa snaží odstrániť tento šum zo vstupu, namiesto kopírovania vstupu na výstup.

Riedky autoenkodér

Pre riedky autoenkodér (v angličtine sparse autoencoder) je špecifické, že k stratovej funkcii sa pridáva pokuta za riedkosť $\Omega(\boldsymbol{h})$ pre kódovaciu vrstvu $\boldsymbol{h}$:

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}) \tag{4.6}$$

Takýmto spôsobom je možné obmedziť počet aktívnych neurónov vrstvy $\boldsymbol{h}$, ktorá má zvyčajne väčší rozmer ako vstup. Riedky autoenkodér tak môže získať zaujímavejšie výsledky.

4.2.2 Vzťah medzi autoenkodérom a PCA

Ak je autoenkodér lineárny a ako stratová funkcia L sa použije stredná kvadratická chyba (mean squared error), jeho optimálne riešenie silne súvisí s PCA. Váhy takéhoto autoenkodéra s veľkosťou vrstvy $\boldsymbol{h}$ rovnej p, pričom p je menšie ako veľkosť vstupu, transformujú vstupné dáta do toho istého podpriestoru ako prvých p hlavných komponentov PCA. Ak je autoenkodér nelineárny, dokáže sa naučiť silnejšie nelineárne zovšeobecnenie PCA.

5 Implementácia

Pre implementáciu jednotlivých častí systému bol použitý programovací jazyk Python. Python odporuje objektovo orientované, štruktúrované aj funkcionálne programovanie. Je to dynamicky typový jazyk, podporuje veľké množstvo vysokoúrovňových dátových typov a na správu pamäte používa garbage collection. Jeho syntax bola navrhnutá tak, aby bol kód dobre čitateľný. Pre určenie blokov sa v Pythone nepoužívajú zátvorky ani kľúčové slová, ale odsadenie.

Python je vyvíjaný ako open-source. Dá sa jednoducho rozširovať, pričom moduly je možné písať aj v nižších programovacích jazykoch, ako sú C a C++. Výhodou je rozsiahla štandardná knižnica a dostupnosť veľkého počtu ďalších knižníc, takže práca v Pythone je jednoduchá a rýchla.

5.1 Implementácia databázy vzorov MNIST

5.1.1 Knižnica NumPy

Dátové štruktúry v Pythone sú heterogénne, čo znamená, že v jednej dátovej štruktúre sa môžu nachádzať objekty viacerých typov. Preto nie sú vhodné pre použitie v úlohách, kde je potrebné pracovať s viacrozmernými poliami či maticami. Knižnica NumPy pridáva do jazyka Python podporu veľkých, viacrozmerných polí a matíc, ako aj veľké množstvo funkcií pre rýchlu a efektívnu prácu s týmito poliami.

Hlavnou štruktúrou knižnice NumPy je štruktúra `ndarray`, ktorá reprezentuje všeobecné n-rozmerné pole. Toto pole má fixnú veľkosť a je homogénne, čiže obsahuje objekty len jedného typu. Veľká časť NumPy je napísaná v jazyku C, takže umožňuje pracovať s číselnými typmi jazyka C. To zároveň robí operácie nad poliami efektívnejšími.

Knižnica NumPy sa stala veľmi obľúbenou a používa ju veľmi veľké množstvo iných knižníc. Všetky ostatné knižnice použité v tejto práci využívajú NumPy.

5.1.2 Dataset MNIST

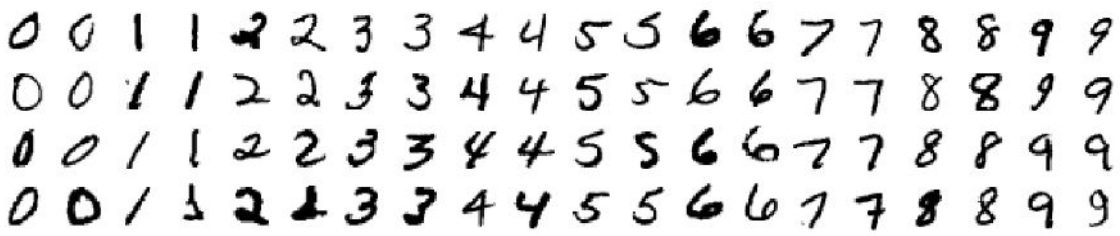
Dataset MNIST je databáza rukou písaných číslic, ktorú uverejnil Yann LeCun v roku 1998 [LCB98]. Obsahuje obrázky arabských číslic 0 až 9. Je to populárny dataset pre štúdium rozpoznávania obrazových vzorov, pretože rukou písané číslice sú ideálnym príkladom vyvážených dát.

Trénovacia množina obsahuje celkovo 60 000 vzorov a testovacia množina obsahuje 10 000 vzorov. Všetky obrázky majú veľkosť 28×28 pixelov a sú v odtieňoch sivej.

Dáta oboch množín (trénovacej a testovacej) boli uložené ako dvojrozmerné NumPy polia, kde jednotlivé obrázky tvoria riadky týchto polí. Obrázky sú uložené ako vektory 784 premenných, pričom každá premenná uchováva hodnotu jedného pixelu. Takéto spracovanie dát umožňuje jednoduché použitie pri extrakcii príznakov a klasifikácii pomocou ďalších knižníc.

Fashion MNIST

Pri detekcii vzorov nepatriacich do trénovacej množiny budeme používať dataset Fashion MNIST [XRV17]. Tento dataset má tak isto ako MNIST 60 000 trénovacích obrázkov a 10 000 testovacích obrázkov, každý z nich v odtieňoch sivej s rozmermi 28×28 pixelov. Na týchto obrázkoch však nie sú rukou písané číslice, ale, ako už napovedá názov, módne doplnky (oblečenie a obuv). Tieto sú rozdelené do 10 tried (tak ako MNIST) a v každej triede sú obrázky iného typu oblečenia alebo obuvi.



Obr. 5.1: Dataset MNIST

Obr. 5.2: Dataset Fashion MNIST

Fashion MNIST bol vytvorený s cieľom nahradiť MNIST, pretože podľa autora je MNIST príliš jednoduchý a príliš často používaný[1].

5.2 Implementácia klasifikátora

5.2.1 Knižnica Scikit-learn

Knižnica Scikit-learn implementuje širokú škálu algoritmov strojového učenia s učiteľom (supervised) a bez učiteľa (unsupervised), ako sú algoritmy pre klasifikáciu, regresiu, extrakciu príznakov a iné. Taktiež implementuje veľký počet pomocných modulov pre predspracovanie dát, výpočet metrík či výpis reportov klasifikácie. Knižnica vznikla v roku 2007 ako projekt v rámci Google Summer of Code, prvá verejná beta verzia bola vydaná v roku 2010.

5.2.2 Implementácia metódy najbližšieho suseda

Klasifikácia metódou najbližšieho suseda je v knižnici scikit-learn implementovaná v module `neighbors`. Tento modul obsahuje viacero tried, pre nás je z nich najdôležitejšia trieda `KNeighborsClassifier`. Trieda má tieto funkcie:

- `fit` – natrénovanie klasifikátora. Má dva parametre: maticu dát $\mathbf{X}$ s rozmermi $n \times p$, kde n je počet vzorov a p je počet premenných reprezentujúcich vzory, a vektor $\mathbf{y}$, ktorý má dĺžku n a obsahuje triedy, do ktorých jednotlivé vzory patria.

- `predict` – predikcia tried pre maticu vzorov zadanú na vstupe. Vracia vektor priradených tried pre každý vzor vo vstupnej matici.

[1] `https://github.com/zalandoresearch/fashion-mnist#why-we-made-fashion-mnist`

- `kneighbors` – nájde k najbližších susedov ku každému vzoru z matice na vstupe a vráti vzdialenosti k nim spoločne s ich pozíciami v trénovacej množine.

Ako už bolo spomenuté, v práci použijeme metódu najbližšieho suseda s $k = 1$, preto nie je potrebné riešiť výber triedy v prípade, ak viacero tried obsahuje rovnaký maximálny počet z k najbližších vzorov.

5.3 Implementácia lineárnych metód extrakcie príznakov

Lineárne metódy extrakcie príznakov použité v tejto práci sú tiež implementované v knižnici Scikit-learn. Metódu PCA implementuje trieda `PCA`. Pre metódu ICA sme použili algoritmus FastICA, ktorý je implementovaný v triede `FastICA` podľa [HO00]. Metódu LDA implementuje trieda `LinearDiscriminantAnalysis`. Všetky triedy majú tieto funkcie:

- `fit` – natrénovanie modelu pre extrakciu príznakov, alebo, v prípade LDA, aj klasifikáciu. Parametrom je matica dát trénovacej množiny. LDA je algoritmus učenia s učiteľom, preto potrebuje aj vektor tried pre jednotlivé vzory.
- `transform` – extrakcia príznakov z dát na vstupe aplikáciou daného algoritmu.
- `inverse_transform` – spätná transformácia extrahovaných príznakov do pôvodného tvaru. LDA túto funkciu nepodporuje.

5.4 Implementácia nelineárnych metód extrakcie príznakov

5.4.1 Knižnica Keras

Pre implementáciu neurónových sietí použijeme knižnicu Keras. Keras je vysokoúrovňové rozhranie pre programovanie aplikácií (API) pre prácu s neurónovými sieťami. Je napísaný v Pythone a dokáže bežať nad frameworkami ako Tensorflow, Microsoft

Cognitive Toolkit (CNTK) alebo Theano. Bol vyvinutý so zameraním na užívateľskú prívetivosť, modularitu a jednoduchú rozširovateľnosť.

Základnou štruktúrou v Kerase je model. Model organizuje jednotlivé vrstvy neurónovej siete. Základným modelom je sekvenčný model, kde sú vrstvy usporiadané v rade za sebou. Pre vytvorenie komplexnejších architektúr poskytuje Keras funkcionálne API. Najdôležitejšie funkcie modelu sú:

- `compile` – pomocou tejto funkcie sa model nakonfiguruje pre proces trénovania. Nastaví sa optimalizátor a stratová funkcia.
- `fit` – natrénovanie modelu podľa zadaných parametrov ako sú trénovacie dáta, počet iterácii (epochs), batch size a iné.
- `predict` – výpočet výstupov podľa dát na vstupe algoritmom dopredného šírenia.

Keras podporuje veľké množstvo vrstiev, aktivačných funkcií a optimalizátorov. Taktiež umožňuje vizualizáciu modelov. Dokumentácia je dostupná na [Cho+15].

6 Výsledky experimentov

V tejto kapitole prezentujeme výsledky vykonaných experimentov. V sekcii 6.1 popisujeme porovnanie metód extrakcie príznakov podľa úspešnosti klasifikácie extrahovaných príznakov metódou najbližšieho suseda. V sekcii 6.2 pridáme klasifikáciu neurónovou sieťou. Sekcia 6.3 sa bude venovať rekonštrukcii vzorov pomocou autoenkodéra. V poslednej sekcii 6.4 popíšeme výsledky experimentov pri detekcii vzorov, ktoré nepatria do priestoru vzorov, z ktorého pochádza trénovacia množina.

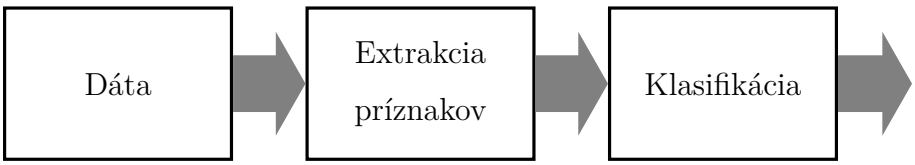
6.1 Extrakcia príznakov

V tejto sekcii ukážeme porovnanie lineárnych metód extrakcie príznakov. Porovnáme metódy PCA a ICA na spoločnej extrakcii a následne na extrakcii po triedach.

6.1.1 Spoločná extrakcia

Pod spoločnou extrakciou príznakov rozumieme využitie jedného spoločného extraktora. Tento extraktor natrénujeme s využitím spoločnej trénovacej množiny vzorov všetkých tried. Následne pomocou neho extrahujeme príznaky trénovacej množiny, s ktorými natrénujeme klasifikátor. Na obrázku 6.1 je zobrazená štruktúra systému použitého pri spoločnej extrakcii.

Základnou metrikou pre porovnanie metód extrakcie príznakov bude úspešnosť



Obr. 6.1: Štruktúra systému spoločnej extrakcie príznakov

klasifikácie vzorov podľa extrahovaných príznakov. Tú budeme počítať na testovacej množine, z ktorej pred klasifikáciou, tak ako pri trénovacej množine, extrahujeme príznaky vzorov pomocou extraktora.

Ako už bolo spomenuté, pre klasifikáciu použijeme metódu najbližšieho suseda. Najčastejšími metrikami, ktoré sa používajú pri metóde najbližšieho suseda, sú Euklidova, Manhattanská a Čebyševova. Tieto metriky boli popísané v sekcii 1.3 na strane 21. Pri klasifikácii použijeme pre porovnanie všetky tri metriky.

Algoritmus PCA je implementovaný v knižnici Scikit-learn v triede `PCA`. Jej konštruktor má takéto parametre:

```
PCA(self, n_components=None, copy=True, whiten=False,
    svd_solver='auto', tol=0.0, random_state=None)
```

Pri experimente s PCA použijeme defaultné hodnoty týchto parametrov okrem parametra `n_components`. Trieda `PCA` automaticky normalizuje dáta na nulový stred, takže nie je potrebná žiadna úprava dát.

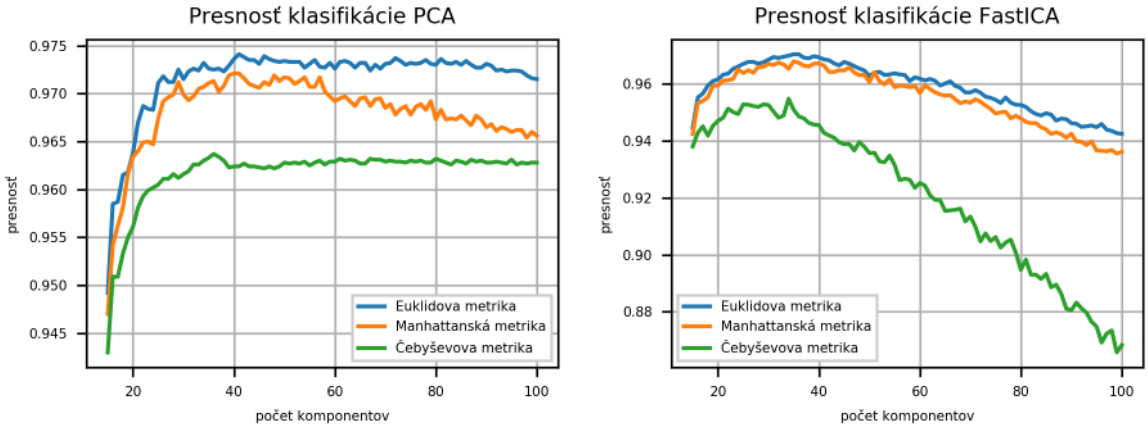
Pre metódu ICA použijeme algoritmus FastICA, ktorý má v knižnici Scikit-learn takéto parametre:

```
FastICA(self, n_components=None, algorithm='parallel', whiten=True,
        fun='logcosh', fun_args=None, max_iter=200, tol=1e-4,
        w_init=None, random_state=None)
```

Tu tiež použijeme defaultné hodnoty parametrov, okrem parametrov `n_components` a `max_iter`. Parameter `max_iter` určuje maximálny počet iterácií algoritmu. Ak algoritmus neskonverguje po vykonaní tohto počtu iterácií, skončí. Trieda `FastICA`, tak ako `PCA`, automaticky normalizuje dáta na nulový stred.

Pre `PCA` aj `FastICA` budeme sledovať úspešnosť klasifikácie v závislosti na počte komponentov (extrahovaných príznakov), ktorý sa nastavuje hodnotou parametra `n_components`. Pri `FastICA` je dôležité nastavenie parametra `max_iter`, pretože ak algoritmus neskonverguje, nájdené komponenty nebudú nezávislé.

Obrázok 6.2 na nasledujúcej strane zobrazuje grafy presnosti klasifikácie pre PCA a FastICA s rôznym počtom extrahovaných komponentov. Výsledky pre počet komponentov menší ako 15 nie sú zobrazené kvôli prehľadnosti.



Obr. 6.2: Porovnanie presnosti klasifikácie pre PCA a FastICA

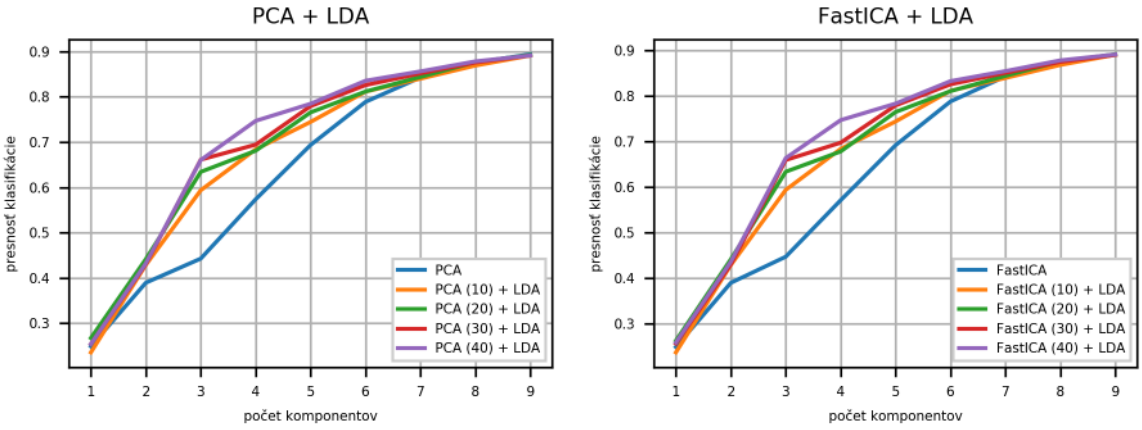
Na grafoch vidíme, že najväčšiu presnosť pri oboch metódach extrakcie dosahuje metóda najbližšieho suseda s použitím Euklidovej metriky. Čebyševova metrika dosahuje najhoršie výsledky, čo je viditeľné najmä pri algoritme FastICA, kde sa so zvyšujúcim sa počtom extrahovaných komponentov zvyšuje rozdiel medzi presnosťou Čebyševovej metriky a zvyšných dvoch metrík.

Presnosť pri FastICA však klesá pre všetky metriky. Najväčšiu presnosť dosahuje pri 35 komponentoch, so zväčšujúcim sa počtom komponentov presnosť klasifikácie klesá. Vysvetľujeme si to tým, že trvanie na podmienke nezávislosti premenných (na rozdiel od nekorelovanosti pri PCA) neprispieva k zvýšeniu presnosti pri rovnakom počte komponentov. PCA dosahuje najväčšiu presnosť klasifikácie pri 40 komponentoch a pri viacerých komponentoch sa príliš nemení.

Pridanie LDA

Zatiaľ sme pri porovnaní lineárnych metód extrakcie príznakov neuvažovali LDA. Je to preto, lebo pomocou LDA je možné extrahovať najviac $c - 1$ komponentov (kde c je počet tried), čo je v prípade datasetu MNIST rovné 9. LDA navyše nie je vhodné použiť v prípade, ak sú premenné korelované, ako pri MNISTe. Pomocou PCA a ICA sa však tento problém dá odstrániť. Môžeme teda porovnať, aký vplyv má na presnosť klasifikácie použitie LDA spoločne s PCA a ICA.

Pri tomto experimente budeme postupovať tak, že najskôr extrahujeme určitý počet komponentov pomocou PCA alebo FastICA, a následne na tieto dáta aplikujeme LDA. Výstupy z LDA použijeme pri klasifikácii metódou najbližšieho suseda s Euklido-



Obr. 6.3: Porovnanie presnosti klasifikácie pri použití LDA

vou metrikou. Presnosť klasifikácie pri takejto extrakcii potom porovnáme s presnosťou klasifikácie pri použití PCA a FastICA.

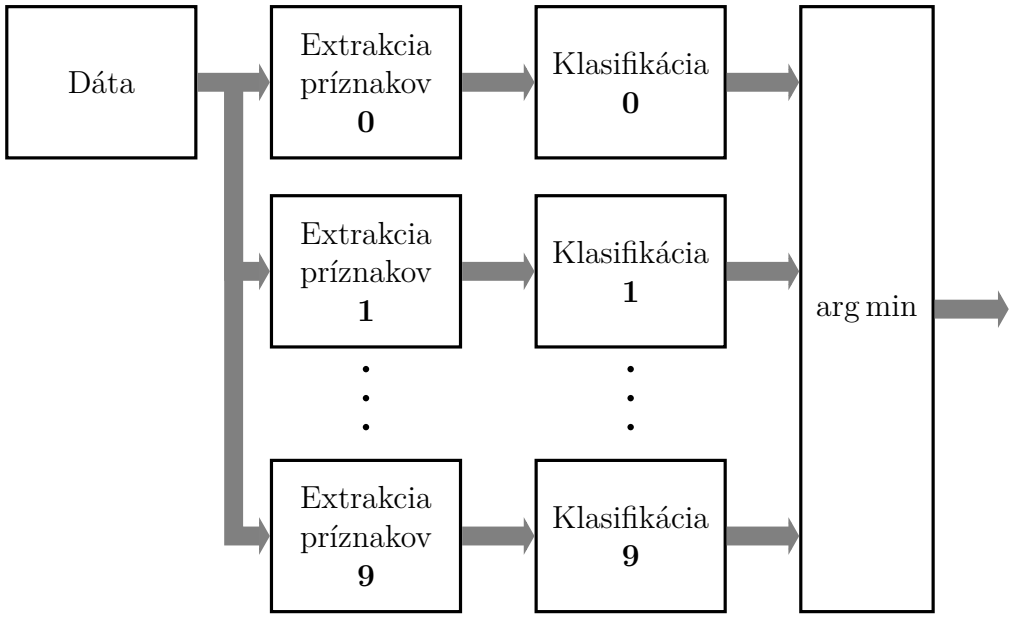
Na obrázku 6.3 môžeme vidieť, že pri použití LDA sa presnosť klasifikácie zväčší v oboch prípadoch. Modrá čiara predstavuje presnosť pri extrakcii n komponentov len s použitím PCA, resp. FastICA. Ostatné čiary predstavujú presnosť pri extrakcii pomocou LDA, pričom predtým sa extrahuje určitý počet komponentov pomocou príslušnej metódy. Tento počet je uvedený v zátvorke.

Grafy na obrázku sú takmer rovnaké, čo je pri malom počte komponentov pochopiteľné. LDA je vhodnejšie použiť pri datasetoch s väčším počtom tried. Týmto sme však ukázali, že kombinácia PCA a LDA, resp. ICA a LDA môže zlepšiť presnosť klasifikácie. Je to spôsobené tým, že pokiaľ PCA a ICA extrahujú príznaky z trénovacej množiny ako celku, LDA pridáva informáciu o rozlišovaní medzi jednotlivými triedami.

6.1.2 Extrakcia po triedach

Ak extrahujeme príznaky spoločne, extraktor sa učí na celej trénovacej množine tak, aby vedel extrahovať najrelevantnejšie informácie o vzoroch. V niektorých triedach však môžu byť určité premenné vhodnejšie pre extrakciu ako v iných triedach. Extrakcia po triedach môže preto lepšie extrahovať dôležité informácie pre klasifikáciu.

Vyskúšajme teda pre MNIST extrahovať príznaky pre každú triedu zvlášť. Navrhovaný systém bude mať takúto štruktúru: pre každú triedu $(0, 1, ..., 9)$ budeme trénovať samostatný extraktor príznakov a samostatný klasifikátor. Extraktory natrénujeme len so vzormi trénovacej množiny z danej triedy. Následne z týchto vzorov

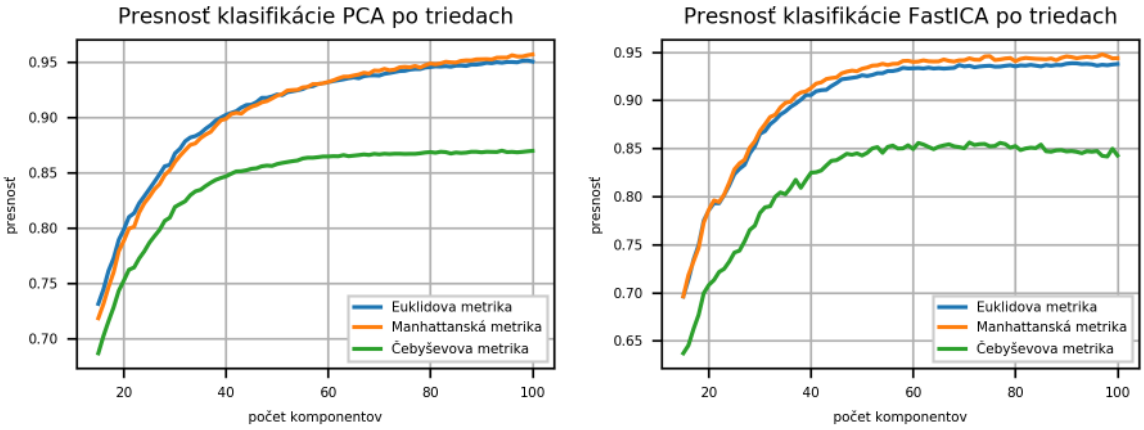


Obr. 6.4: Štruktúra systému extrakcie príznakov po triedach

extrahujeme príznaky a s nimi natrénujeme príslušný klasifikátor.

Pri klasifikácii neznámeho vzoru z neho extrahujeme príznaky extraktorom každej triedy a príslušným klasifikátorom nájdeme najbližšieho suseda. Tento najbližší sused bude najbližším vzorom v rámci danej triedy, pretože každý extraktor a jeho príslušný klasifikátor pracujú len s jednou triedou. Vzor zaradíme do triedy, do ktorej patrí ten z nájdených najbližších susedov, ktorého vzdialenosť od neznámeho vzoru je najmenšia. Štruktúra tohto systému je zobrazená na obrázku 6.4.

Pre jednoduchšie porovnanie budeme všetky extraktory príznakov trénovať na rovnaký počet komponentov. Teoreticky je však možné trénovať extraktor pre každú



Obr. 6.5: Porovnanie presnosti klasifikácie extrakcie po triedach

triedu s iným počtom komponentov, pretože pre dáta rôznych tried môže klasifikácia dosahovať dostatočnú presnosť pri inom počte komponentov.

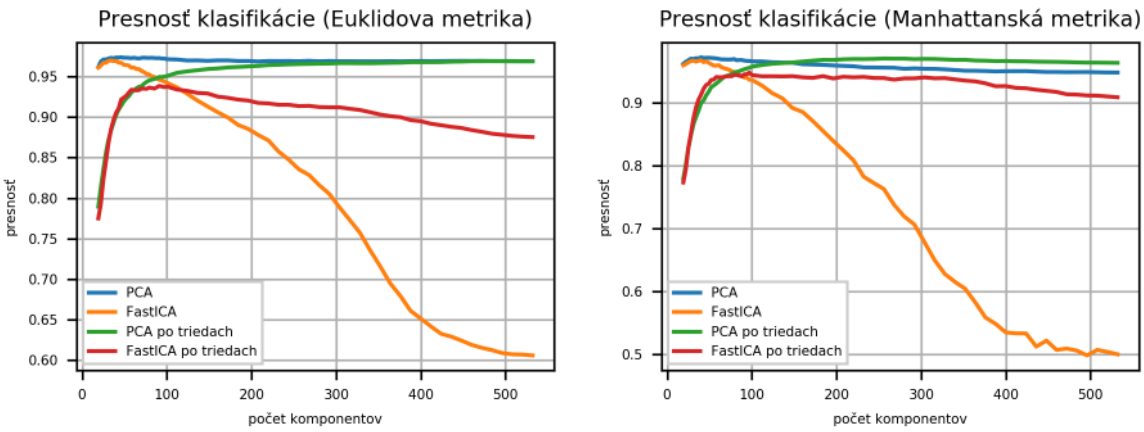
Obrázok 6.5 zobrazuje presnosť klasifikácie po triedach pre PCA a FastICA. Ukazuje sa, že presnosť klasifikácie nie je taká dobrá ako pri spoločnej extrakcii, avšak so zväčšujúcim sa počtom extrahovaných komponentov stále rastie a neustálila sa už pri malom počte komponentov, ako pri spoločnej extrakcii. Najhorší výsledok aj tu dosahuje Čebyševova metrika.

6.1.3 Porovnanie spoločnej extrakcie a extrakcie po triedach

Porovnajme presnosť klasifikácie pre oba prístupy – spoločnú extrakciu príznakov a extrakciu po triedach. V oboch prípadoch sa pre PCA aj pre FastICA ukázala ako najslabšia metrika pre metódu najbližšieho suseda Čebyševova metrika. Zvyšné dve metriky dosahovali lepšiu presnosť.

Doteraz sme porovnávali metódy extrakcie pre $N \leq 100$. Na obrázku 6.6 uvádzame porovnanie spoločnej extrakcie a extrakcie po triedach pre PCA a FastICA, pričom sme zväčšili maximálnu hodnotu N, aby bolo možné vidieť, ako sa metóda najbližšieho suseda správa pri extrakcii väčšieho počtu príznakov. Grafy sú rozdelené podľa metriky použitej v metóde najbližšieho suseda. Presnosť klasifikácie s použitím Čebyševovej metriky neuvádzame.

Na obrázku vidíme, že presnosť spoločnej extrakcie a extrakcie po triedach pre PCA sa so zvyšujúcim sa N približujú (96,9 %) a pri použití Manhattanskej metriky je presnosť pri extrakcii po triedach dokonca vyššia (96,3 % oproti 94,8 % pri spoločnej



Obr. 6.6: Porovnanie presnosti klasifikácie oboch prístupov extrakcie

extrakcii). Pri FastICA je presnosť spoločnej extrakcie najvyššia pri $N = 35$ a so zväčšujúcim sa N neustále klesá, až sa dostáva pod úroveň 60 %. Presnosť pri extrakcii po triedach je výrazne lepšia.

Presnosť klasifikácie pri extrakcii po triedach stúpa pomalšie ako pri spoločnej extrakcii, pretože pri trénovaní extraktorov sa každý extraktor a príslušný klasifikátor trénujú oveľa menším počtom trénovacích vzorov. Algoritmus FastICA však od určitého počtu komponentov dokáže pri extrakcii po triedach dosiahnuť výrazne vyššiu presnosť klasifikácie pri použití oboch metrík.

6.2 Rozpoznávanie neurónovou sieťou

V predchádzajúcej sekcii sme porovnali lineárne metódy extrakcie príznakov na úlohe klasifikácie pomocou metódy najbližšieho suseda. V tejto sekcii sa dostávame od lineárnych systémov k nelineárnym.

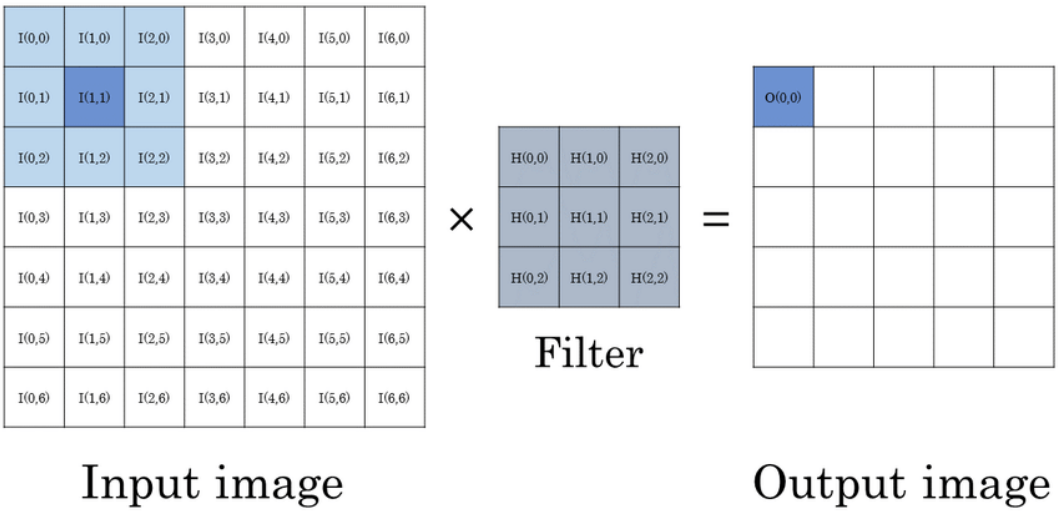
Neurónové siete sa bežne využívajú pre klasifikáciu rôznych typov dát. Ich výhodou je, že vykonávajú extrakciu príznakov spolu s klasifikáciou. My si však tieto časti rozdelíme a ukážeme, ako neurónová sieť s dátami pracuje a ako vyzerajú dáta po extrakcii pomocou neurónovej siete.

Konvolučné neurónové siete

Pri klasifikácii grafických dát, akými je aj dataset MNIST, sa využívajú najmä konvolučné neurónové siete. V týchto sieťach sa striedajú konvolučné vrstvy so vzorkovacími vrstvami. Na konci je jedna alebo viac tzv. full-connect vrstiev, teda vrstiev s úplnými spojeniami.

Konvolučná vrstva (convolution layer) je množina menších filtrov, ktoré sa aplikujú na celú plochu obrázku. Veľkosť a počet týchto filtrov sú hyper-parametrami siete.

Vzorkovacia vrstva (pooling layer) sa používa na zmenšenie veľkosti dát tak, že kombinuje výstupy viacerých neurónov z predchádzajúcej vrstvy do jedného vstupu pre neurón v nasledujúcej vrstve. Kombinujú sa výstupy malých skupín neurónov, zvyčajne 2×2. Najčastejšie sa počíta maximum alebo priemer výstupov (max pooling, resp. average pooling).



Obr. 6.7: Aplikácia filtra s veľkosťou 3×3 na obrázok s veľkosťou 7×7 pixelov (Obrázok prebratý z [Bas+17])

Po sérii konvolučných a vzorkovacích vrstiev nasleduje jedna alebo viac vrstiev s úplnými spojeniami, pričom výstup poslednej z týchto vrstiev sa používa na klasifikáciu obrázkov.

Konvolučné siete ako typ hlbokých sietí dosahujú oveľa lepšie výsledky pri klasifikácii grafických dát ako klasické viacvrstvové perceptróny. Navyše majú oveľa menej trénovateľných váh. Napríklad pri práci s obrázkami s veľkosťou 100×100 má každý neurón v druhej vrstve perceptrónu $100 * 100 = 10\ 000$ váh. Konvolučná vrstva s 5 filtrami s veľkosťou 16×16 má iba $5 * 16 * 16 = 1280$ trénovateľných váh.

Aktivačné funkcie

V klasifikačných neurónových sieťach (nielen tých konvolučných) sa zvyčajne ako aktivačné funkcie používali sigmoid funkcie. Tieto funkcie sa časom nahradili funkciou ReLU (Rectified Linear Unit), pretože sa ukázalo, že pri trénovaní hlbokých sietí často dosahuje lepšie výsledky ako sigmoid, a navyše jej výpočet je jednoduchší.

Vo výstupnej vrstve siete sa zvyčajne ako aktivačná funkcia používa softmax, ktorá výstupy upravuje tak, aby ich hodnoty boli v intervale $(0, 1)$ a aby ich súčet bol rovný 1, takže sa dajú interpretovať ako pravdepodobnosti.

Označenie	Simple	Deep	LeNet1	LeNet4	ConvNet
Vstup	784	784	28×28	28×28	28×28
Skryté vrstvy	FC(N)	FC(300)	C(4, (5×5))	C′(4, (5×5))	C(16, (5×5))
		FC(N)	AvgP(2×2)	AvgP(2×2)	MaxP(2×2)
			C(12, (5×5))	C(16, (5×5))	C(8, (5×5))
			AvgP(2×2)	AvgP(2×2)	MaxP(2×2)
			FC(N)	FC(120)	FC(N)
				FC(N)	
Výstup	FC(10)	FC(10)	FC(10)	FC(10)	FC(10)

Tabuľka 6.1: Architektúra porovnávaných neurónových sietí

6.2.1 Porovnanie viacerých typov sietí

Pre porovnanie budeme uvažovať viacero architektúr neurónových sietí. Dve architektúry budú pozostávať z vrstiev úplne prepojených neurónov. Tieto porovnáme s dvoma konvolučnými architektúrami, založenými na architektúrach LeNet-1 a LeNet-4 predstavenými v [LeC+98]. Okrem týchto dvoch pridáme vlastnú konvolučnú sieť, ktorú sme označili ConvNet.

V tabuľke 6.1 sú zobrazené architektúry porovnávaných sietí. Názvy vrstiev sú skrátené, aby sa vošli do tabuľky. FC(n) označuje full-connect vrstvu (vrstvu s úplnými spojeniami) s n neurónmi, AvgP a MaxP sú vzorkovacie vrstvy (average pooling, resp. max pooling) s veľkosťou vzorky uvedenou v zátvorke. C označuje konvolučnú vrstvu s počtom filtrov a veľkosťou každého filtra uvedenými v zátvorke. C′ je typ konvolučnej vrstvy, ktorej výstup má rovnakú veľkosť ako vstup. Všetky vrstvy okrem poslednej používajú aktivačnú funkciu ReLU. Posledná vrstva používa softmax.

Predposledná vrstva každej siete má N neurónov. Túto vrstvu sme pridali, aby sme ukázali výstup extrakcie N príznakov neurónovou sieťou pred klasifikáciou. Pri experimentoch budeme N meniť a ukážeme, ako od tohto parametra závisí presnosť klasifikácie.

6.2.2 Trénovanie sietí

Všetky spomínané architektúry implementujeme v Pythone s použitím knižnice Keras. Keras umožňuje jednoducho vytvárať modely sietí, pričom modely môžu zdieľať určité vrstvy. To nám umožní oddeliť extrakčnú časť siete, ktorú môžeme použiť samostatne.

Ukážka implementácie modelu siete LeNet4 z tabuľky 6.1:

```
from keras import Model, layers

N = ...  # počet extrahovaných príznakov
input_layer = keras.layers.Input(shape=(28, 28, 1))  # musíme zadať počet kanálov
x = layers.Conv2D(4, (5, 5), activation='relu',
                  padding='same')(input_layer)       # 28 × 28 × 4
x = layers.AveragePooling2D((2, 2))(x)               # 14 × 14 × 4
x = layers.Conv2D(16, (5, 5), activation='relu')(x)  # 10 × 10 × 16
x = layers.AveragePooling2D((2, 2))(x)               # 5 × 5 × 16
x = layers.Flatten()(x)                              # 400
x = layers.Dense(120, activation='relu')(x)          # full-connect vrstva
encoded_layer = layers.Dense(N, activation='relu')(x)
output_layer = keras.layers.Dense(10, activation='softmax')(encoded_layer)

model = Model(input_layer, output_layer)             # model celej siete
encoder = Model(input_layer, encoded_layer)          # extrakčná časť siete
```

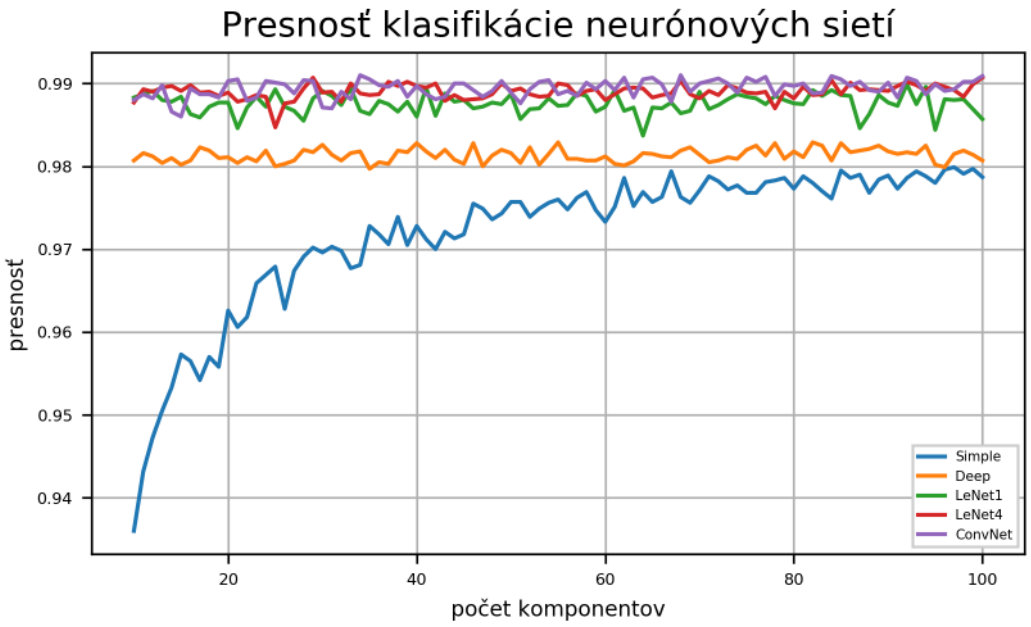
Vytvorené modely je potrebné natrénovať. Trénovať stačí len `model`, pretože `encoder` je iba akýsi pohľad na časť tohto modelu. Pred trénovaním je potrebné model pripraviť na proces trénovania nastavením optimalizátora a stratovej funkcie:

```
model.compile(optimizer='adadelta', loss='binary_crossentropy')
```

Zvolili sme optimalizátor Adadelta, ktorý nevyžaduje nastavenie počiatočnej hodnoty learning rate, a pri mnohých úlohách ukazuje rýchlu konvergenciu. Stratová funkcia binary cross-entropy sa často používa pri klasifikačných algoritmoch. Model po nastavení natrénujeme takto:

```
model.fit(x_train, y_train, epochs=100, batch_size=256,
          validation_data=(x_test, y_test))
```

V tejto ukážke model trénujeme s veľkosťou batch-u 256, čo vyjadruje počet vzorov, po spracovaní ktorého sa má spätne šíriť chyba. Parameter `epochs` vyjadruje počet iterácií cez trénovaciu množinu. Trénovacie dáta sú v `x_train`, `y_train` je matica núl a jednotiek, pričom jednotka na pozícii i, j znamená, že vzor i patrí do triedy j, nula znamená, že do tejto triedy nepatrí. Premenné `x_test` a `y_test` musia mať rovnaký tvar ako `x_train` a `y_train` a slúžia na validáciu modelu počas trénovania. Trénovacie a testovacie dáta musia byť pred použitím v neurónovej sieti normalizované do intervalu $[0, 1]$.



Obr. 6.8: Presnosť klasifikácie neurónových sietí

6.2.3 Výsledky porovnania

Najskôr ukážeme porovnanie presnosti klasifikácie. Na obrázku 6.8 je presnosť klasifikácie porovnávaných architektúr pre všetky N medzi 10 a 100. Sieť Simple dosahuje podľa očakávania najmenšiu presnosť. Presnosť siete Deep je o niečo lepšia, ale nedosahuje presnosť konvolučných sietí, ktorá sa pohybuje okolo hranice 99 %.

Presnosti klasifikácie pre $N < 10$ nie sú v grafe zobrazené kvôli prehľadnosti. Preto ukážeme tieto presnosti pomocou tabuľky. Najväčšiu presnosť dosahuje LeNet4, ktorá prekonala 98 % už pri $N = 3$.

Veľkosť N	Simple	Deep	LeNet1	LeNet4	ConvNet
1	38.72 %	61.84 %	54.64 %	**67.07 %**	57.10 %
2	69.91 %	93.18 %	94.70 %	**97.70 %**	94.10 %
3	82.39 %	95.76 %	97.05 %	**98.25 %**	96.97 %
4	87.00 %	96.76 %	97.85 %	**98.42 %**	97.84 %
5	89.53 %	97.60 %	**98.56 %**	98.48 %	98.32 %
6	91.59 %	97.75 %	98.24 %	**98.65 %**	98.41 %
7	92.10 %	97.77 %	98.61 %	**98.89 %**	98.65 %
8	93.01 %	97.96 %	98.55 %	**98.88 %**	98.75 %
9	93.74 %	97.89 %	98.65 %	**98.94 %**	98.34 %
10	93.60 %	98.07 %	**98.83 %**	98.77 %	98.80 %

Tabuľka 6.2: Presnosť klasifikácie sietí pre $N \leq 10$

Počet trénovateľných váh

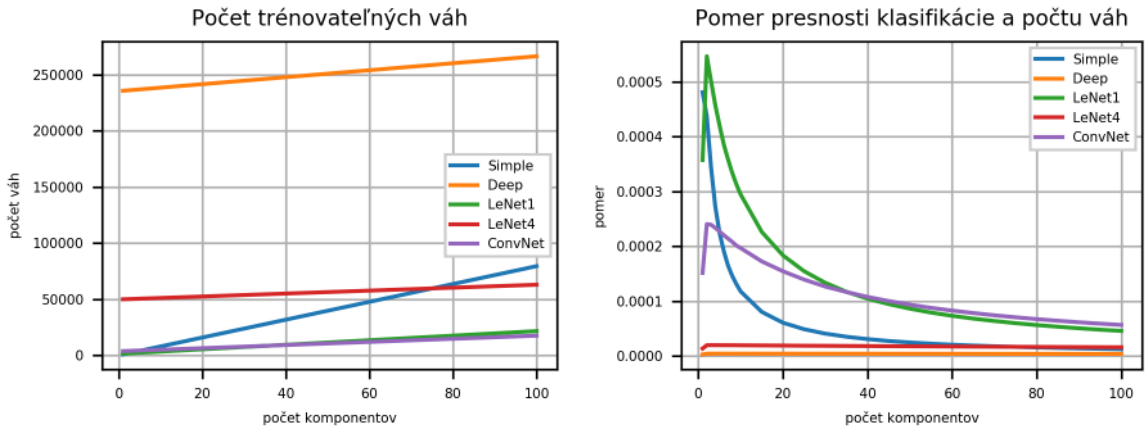
Pre porovnanie výkonnosti sietí je dôležité porovnať počet ich trénovateľných parametrov (váh). Pri popise konvolučných neurónových sietí na začiatku sekcie sme spomenuli, že dokážu dosiahnuť lepší výsledok pri menšom počte váh ako klasické viacvrstvové perceptróny. Počet váh je dôležitý z pohľadu času potrebného na natrénovanie siete, ako aj miesta, ktoré daný model zaberie v pamäti. Ak je čas trénovania veľký a sieť dosahuje slabé výsledky, je takáto sieť neefektívna.

Pretože v našich modeloch sietí má jedna vrstva variabilnú veľkosť, mení sa podľa nej aj celkový počet váh siete. Môžeme preto porovnať pomer presnosti klasifikácie a počtu trénovateľných váh pre rôzne N. Výsledok porovnania je na obrázku 6.9 – obrázok 6.9a zobrazuje celkový počet váh modelov sietí pre rôzne N a obrázok 6.9b zobrazuje pomer presnosti klasifikácie a počtu váh pre dané N.

Vizualizácia dát predposlednej vrstvy

Predposlednú vrstvu s N neurónmi sme pridali najmä preto, aby sme vedeli oddeliť časť extrakcie príznakov a časť klasifikácie. To nám umožňuje pozrieť sa, ako neurónová sieť pracuje s dátami.

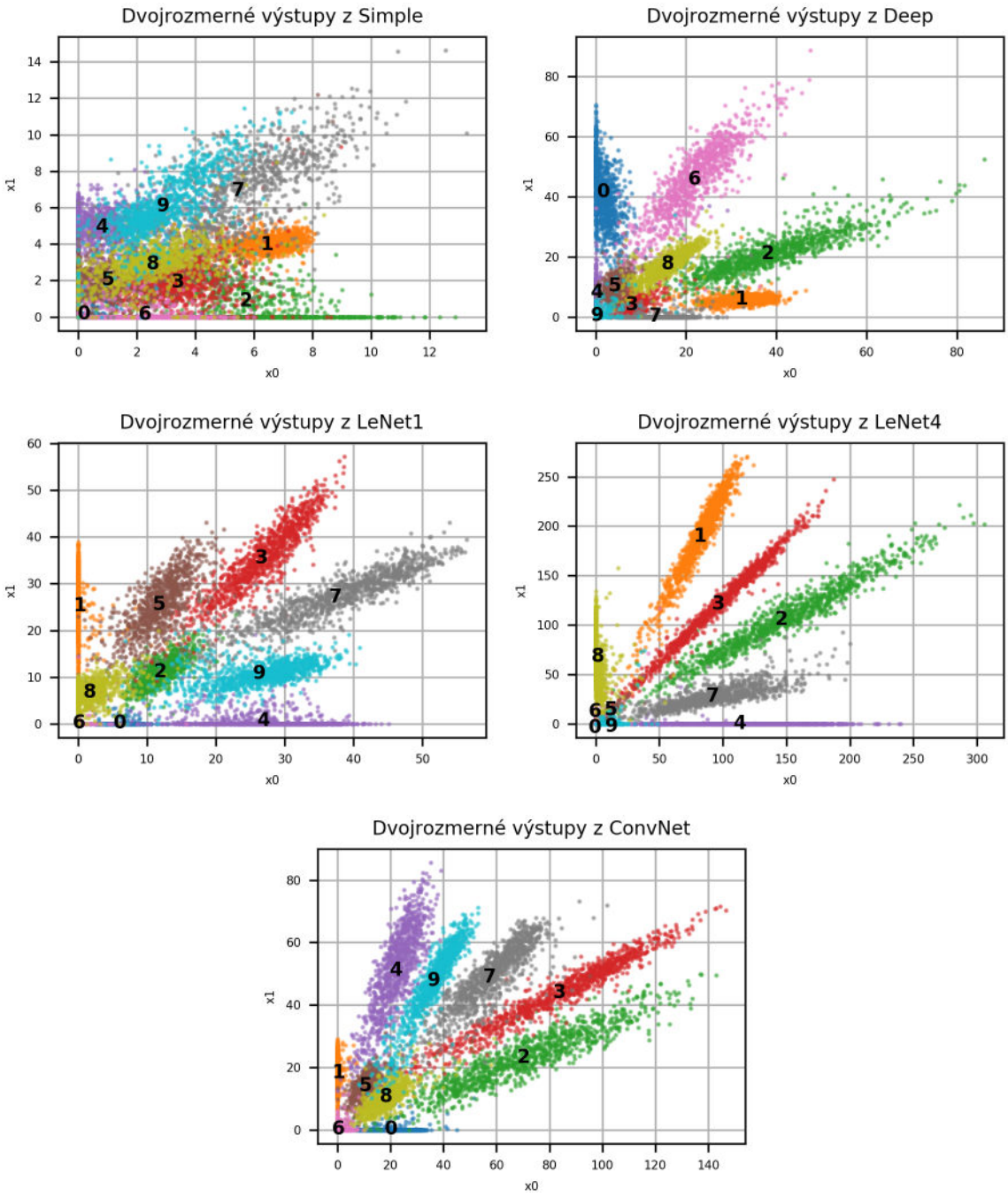
Naše modely sietí dosahujú dobrú presnosť klasifikácie už pri $N = 2$ extrahovaných príznakoch. To znamená, že už pri transformácii dát do dvojrozmerného priestoru



(a) Počet trénovateľných váh porovnávaných sietí pre rôzne N

(b) Pomer presnosti klasifikácie a počtu váh pre rôzne N (väčšie je lepšie)

Obr. 6.9: Počet parametrov (váh) sietí



Obr. 6.10: Dvojrozmerné výstupy predposlednej vrstvy sietí

dokáže sieť dostatočne dobre odlíšiť jednotlivé číslice (triedy). To je výhodné, pretože dvojrozmerné dáta môžeme jednoducho graficky zobraziť a analyzovať. Tieto dáta sú zobrazené na obrázku 6.10.

6.3 Rekonštrukcia vzoru autoenkodérom

Autoenkodér je typ neurónovej siete, ktorý dáta neklasifikuje, ale dekódovacou časťou rekonštruuje späť na pôvodný tvar. Slúži len na extrakciu príznakov, pričom pri trénovaní nepotrebuje poznať triedy pôvodných vzorov. Ide teda o učenie bez učiteľa.

Kódovacia a dekódovacia časť sa trénujú spoločne ako jeden celok. Po natrénovaní sa používa kódovacia časť na extrakciu príznakov a dekódovacia časť na spätnú rekonštrukciu zakódovaných dát. Dekódovacia časť zvyčajne zložením vrstiev zodpovedá kódovacej časti, nemusí to tak ale byť vždy.

6.3.1 Typy autoenkodérov

Pre porovnanie autoenkodérov použijeme v kódovacích častiach rovnaké skryté vrstvy ako pri klasifikačných neurónových sieťach. Dekódovacia časť každého typu autoenkodéra bude zodpovedať kódovacej. Pre ukážku uvedieme implementáciu autoenkodéra založeného na LeNet4 v Pythone.

```
from keras import Model, layers

N = ...  # počet extrahovaných príznakov
input_layer = keras.layers.Input(shape=(28, 28, 1))  # musíme zadať počet kanálov
x = layers.Conv2D(4, (5, 5), activation='relu',
                  padding='same')(input_layer)              # 28 × 28 × 4
x = layers.AveragePooling2D((2, 2))(x)                      # 14 × 14 × 4
x = layers.Conv2D(16, (5, 5), activation='relu')(x)         # 10 × 10 × 16
x = layers.AveragePooling2D((2, 2))(x)                      # 5 × 5 × 16
x = layers.Flatten()(x)                                     # 400
x = layers.Dense(120, activation='relu')(x)                 # full-connect vrstva
encoded_layer = layers.Dense(N, activation='relu')(x)

x = layers.Dense(120, activation='relu')(encoded_layer)
x = layers.Dense(400, activation='relu')(x)
x = layers.Reshape((5, 5, 16))(x)
x = layers.UpSampling2D((2, 2))(x)                          # 10 × 10 × 16
x = layers.Conv2DTranspose(4, (5, 5), activation='relu')(x) # 14 × 14 × 4
x = layers.UpSampling2D((2, 2))(x)                          # 28 × 28 × 4
decoded_layer = layers.Conv2DTranspose(1, (5, 5),
        activation='sigmoid', padding='same')(x)            # 28 × 28 × 1

autoencoder = Model(input_layer, decoded_layer)             # model autoenkodéra
encoder = Model(input_layer, encoded_layer)                 # model kodéra
```

Trénovanie modelu autoenkodéra prebieha tak isto ako trénovanie klasifikačnej neurónovej siete:

```
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train, epochs=100, batch_size=256,
                validation_data=(x_test, x_test))
```

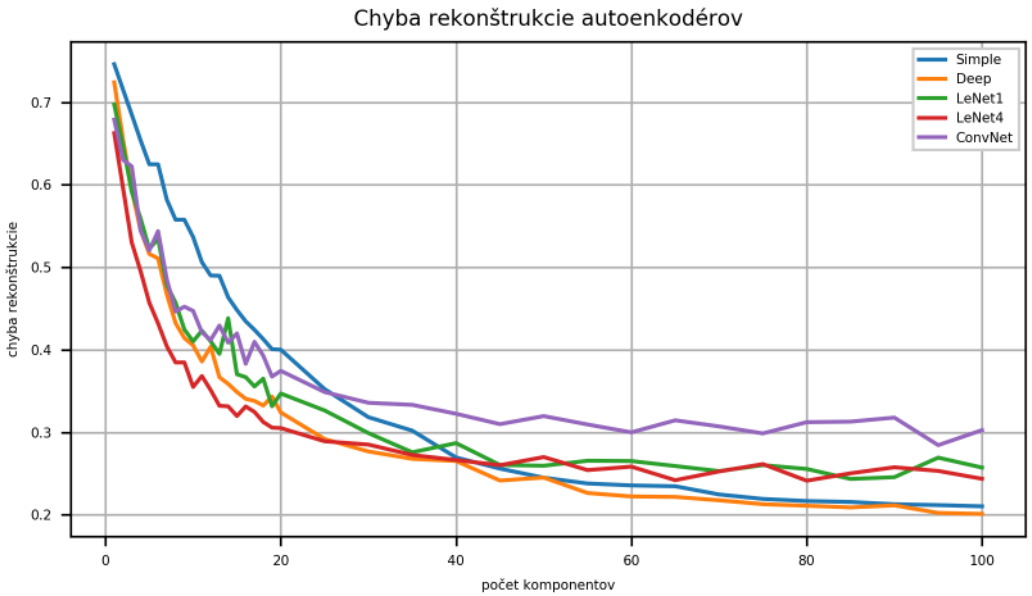
Rozdiel je len v tom, že sa na výstupe namiesto `y_train` a `y_test` používajú vstupné dáta, teda chceme, aby autoenkodér minimalizoval rozdiel medzi vstupom a výstupom. Presnosť autoenkodéra budeme merať pomocou funkcie chyby rekonštrukcie

$$reconstruction\_error = \frac{\sum_{i=1}^{n}\sum_{j=1}^{p}(x_{ij} - r_{ij})^2}{\sum_{i=1}^{n}\sum_{j=1}^{p} x_{ij}^2} \tag{6.1}$$

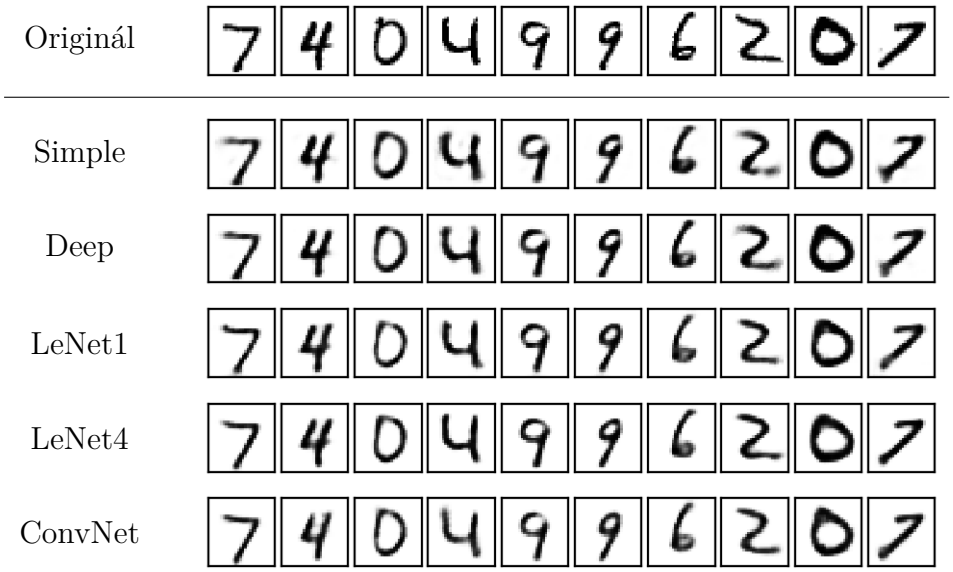
kde x_{ij} sú prvky vstupnej matice a r_{ij} sú prvky výstupnej matice (rekonštruovaných dát), pričom obe matice majú rozmer $n \times p$.

6.3.2 Porovnanie presnosti rekonštrukcie

Pre všetkých päť architektúr sietí z tabuľky 6.1 sme vytvorili model autoenkodéra pridaním dekódovacej časti namiesto výstupnej full-connect vrstvy. Vrstva s veľkosťou N slúži ako výstupná vrstva kodéra, ktorý extrahuje príznaky zo vstupných dát. Pri trénovaní sa však neberie do úvahy výstup kodéra, ale výstup dekodéra, ktorý sme porovnali so vstupom podľa metriky (6.1). Výsledok je na obrázku 6.11.



Obr. 6.11: Chyba rekonštrukcie autoenkodérov



Obr. 6.12: Rekonštrukcia obrázkov testovacej množiny MNIST autoenkodérmi

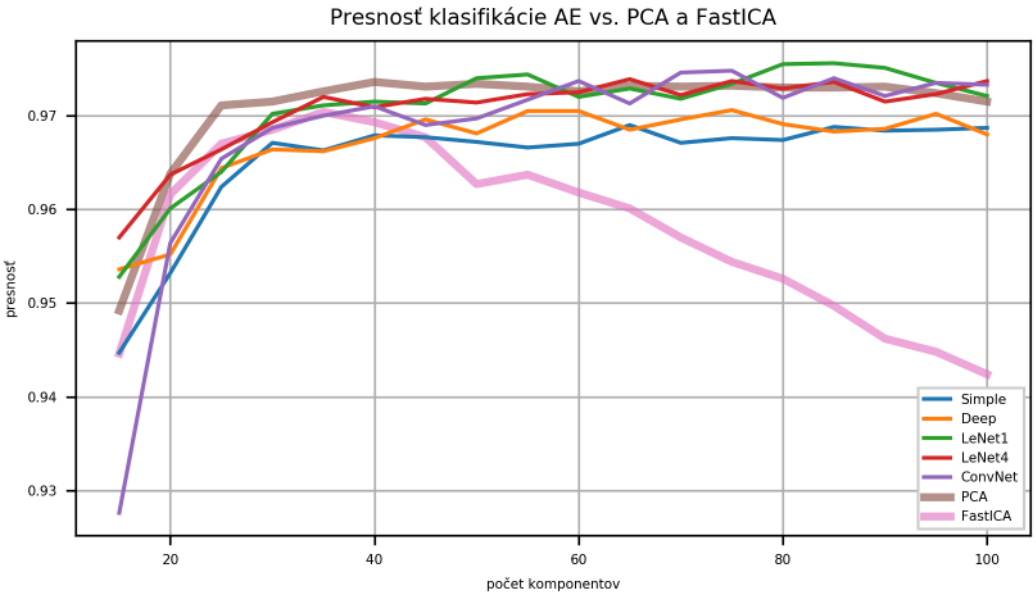
Presnosť rekonštrukcie môžeme demonštrovať aj zobrazením rekonštruovaných obrázkov, ktoré porovnáme s pôvodnými obrázkami. Na obrázku 6.12 vidíme 10 náhodne vybraných obrázkov z testovacej množiny, ktoré sme porovnali s ich rekonštrukciou všetkými typmi porovnávaných autoenkodérov. Veľkosť kódovacej vrstvy bola $N = 32$.

6.3.3 Presnosť klasifikácie metódou najbližšieho suseda

Ukázali sme, ako dokážu autoenkodéry rekonštruovať zakódované dáta. Primárne využitie autoenkodérov je extrakcia príznakov. Na rozdiel od lineárnych metód extrakcie (PCA, ICA a LDA) však autoenkodéry extrahujú príznaky tak, aby ich boli schopné čo najpresnejšie rekonštruovať.

V predchádzajúcej sekcii sme ukázali rekonštrukciu dát na testovacej množine MNIST a porovnali sme modely autoenkodérov podľa chyby, ktorú dosahujú pri rekonštrukcii zakódovaných dát. Teraz porovnáme kvalitu extrahovaných príznakov týchto modelov podľa toho, akú presnosť klasifikácie dosiahne metóda najbližšieho suseda.

Na nasledujúcej strane na obrázku 6.13 je porovnaných všetkých 5 modelov autoenkodérov. Na obrázku sú zobraznené aj presnosti lineárnych metód extrakcie. V metóde najbližšieho suseda sme pri porovnaní použili Euklidovu metriku, pretože s ňou dosiahla pri spoločnej extrakcii PCA a ICA najlepšiu presnosť.



Obr. 6.13: Presnosť klasifikácie extrahovaných príznakov autoenkodérmi

Pri použití konvolučných modelov autoenkodérov dosahuje metóda najbližšieho suseda podobnú presnosť ako pri PCA. Pri modeloch Simple a Deep je presnosť o niečo nižšia.

6.4 Detekcia cudzích vzorov

V predchádzajúcich sekciách sme porovnávali lineárne a nelineárne metódy extrakcie príznakov podľa toho, akú presnosť dosiahne pri klasifikácii metóda najbližšieho suseda. Ukázali sme aj klasifikáciu neurónovými sieťami. V tejto sekcii tieto časti spojíme a pokúsime sa detegovať vzory, ktoré nepatria do priestoru vzorov, z ktorého bola trénovacia množina.

6.4.1 Popis problému

Systémy rozpoznávania vzorov predpokladajú, že vzory, ktoré bude systém rozpoznávať, budú pochádzať z toho istého priestoru vzorov, do ktorého patria vzory trénovacej množiny použitej na natrénovanie systému. Tento predpoklad však v praktických aplikáciách často neplatí, pretože nemáme úplnú kontrolu nad procesom zberu dát [Zad04].

Ak napríklad pracujeme so systémom rozpoznávania druhov zvierat, systém pozná len vopred definovanú množinu druhov, z ktorých vyberie ten, ktorý sa najviac

podobá rozpoznávanému vstupnému obrázku. Tento obrázok však vôbec nemusí obsahovať zviera. Ďalším príkladom je systém rozpoznávania číslic. Ak systém dostane na vstupe obrázok, na ktorom nie je číslica, systém napriek tomu určí pre tento vstup tú číslicu, ktorá sa naň podľa daných kritérií najviac podobá.

Pretože používateľ pri práci so systémom rozpoznávania nevie, ako sa systém rozhoduje, ale vidí len výsledok rozhodnutia, je potrebné vedieť s dostatočnou presnosťou určiť, či rozpoznávaný vstup patrí alebo nepatrí do priestoru vzorov trénovacej množiny.

Detekciu príslušnosti do daného priestoru vzorov budeme testovať na dvoch systémoch. V prvom použijeme klasifikačnú neurónovú sieť, v druhom použijeme neurónovú sieť v spojení s autoenkodérmi.

6.4.2 Detekcia neurónovou sieťou

V klasifikačnej neurónovej sieti používa posledná vrstva zvyčajne softmax ako aktivačnú funkciu, aby sa výsledné hodnoty dali interpretovať ako pravdepodobnosti, s ktorými sieť priradí neznámy vzor do každej z tried. Tieto pravdepodobnosti majú súčet 1 a ich hodnoty sú v intervale $[0, 1]$.

Rozhodnutie o klasifikácii do konkrétnej triedy sa robí tak, že sa vzor zaradí do tej triedy, pre ktorú sieť určila najvyššiu pravdepodobnosť, že vzor je z danej triedy. V skutočnosti ide o podmienenú pravdepodobnosť, teda pravdepodobnosť, že vzor patrí do danej triedy za podmienky, že je z priestoru vzorov trénovacej množiny. Pri detekcii neznámeho vzoru budeme predpokladať, že neurónová sieť nebude vedieť určiť pre vzor jednu konkrétnu triedu, teda že pravdepodobnosti budú mať hodnoty blízko seba. Keďže pravdepodobnosti majú súčet rovný 1, maximálna hodnota z nich bude veľmi malá.

Algoritmus

Algoritmus trénovania siete bude mať takéto kroky:

1. natrénovanie siete vzormi trénovacej množiny,
2. výber dolnej hranice h_0 a hornej hranice h_1, podľa ktorých budeme robiť rozhodnutie o prijatí alebo neprijatí výsledku klasifikácie neurónovou sieťou.

Hranice h_0 a h_1 budú rozdeľovať interval $[0, 1]$ na tri intervaly:

- ak padne maximálna hodnota do intervalu $[0, h_0)$, klasifikáciu do danej triedy zamietneme,
- ak bude hodnota v intervale $[h_0, h_1)$, o klasifikácii nerozhodneme,
- ak bude hodnota v intervale $[h_1, 1]$, klasifikáciu prijmeme.

Výber hraníc budeme robiť pre každú triedu zvlášť s dátami trénovacej množiny. Pri výpočte budeme minimalizovať hodnotu účelovej funkcie

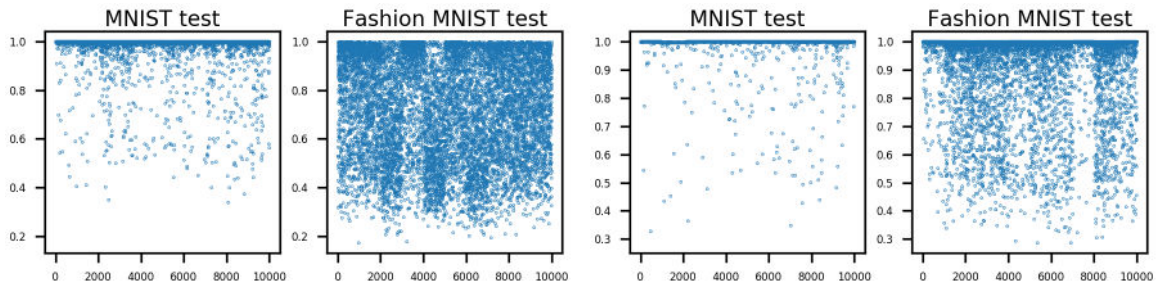
$$\text{minimalizuj} \quad f(h_0, h_1) = n_{wrong} + \frac{n_{neutral}}{w} \tag{6.2}$$

kde n_{wrong} je počet vzorov, ktoré boli správne klasifikované neurónovou sieťou, ale my sme to zamietli sčítaný s počtom vzorov, ktoré boli klasifikované nesprávne a my sme to prijali, $n_{neutral}$ je počet vzorov, pre ktoré sme nevedeli rozhodnúť (bez ohľadu na výstup neurónovej siete) a w je vopred zvolená konštantná váha, ktorú môžeme interpretovať ako mieru preferencie nerozhodnutia o triede pred nesprávnym určením triedy. Hodnoty n_{wrong} a $n_{neutral}$ sa dajú vypočítať z hodnôt hraníc h_0 a h_1.

Pri klasifikácii neznámeho vzoru budeme o správnosti klasifikácie neurónovej siete rozhodovať podľa vypočítaných hraníc. V ideálnom prípade skončí rozhodnutie o klasifikácii neznámeho vzoru nepatriaceho do daného priestoru vzorov neurčito, prípadne zamietnutím.

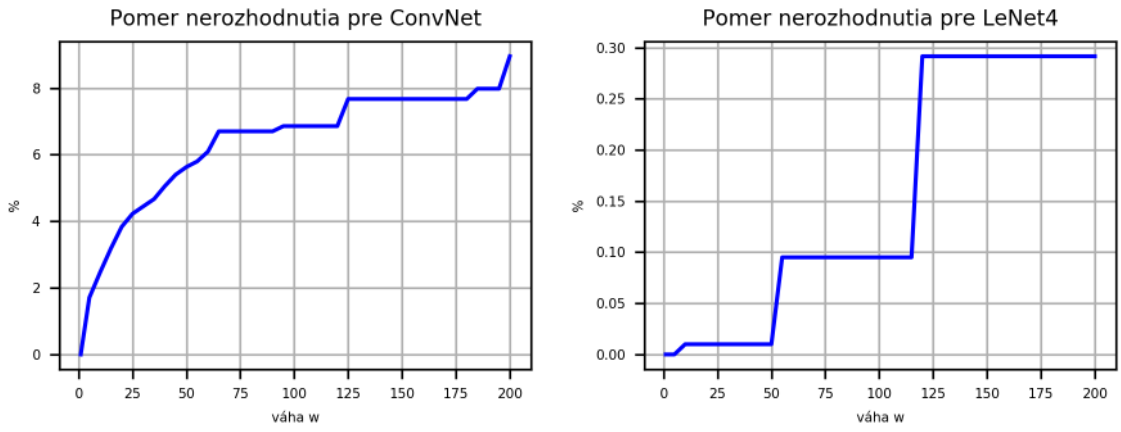
Výsledky

Pri experimentoch sme použili dve konvolučné neurónové siete z tabuľky 6.1 – ConvNet a LeNet4. Siete sme natrénovali s trénovacou množinou MNIST, aby dosahovali čo



(a) ConvNet (b) LeNet4

Obr. 6.14: Hodnoty výstupnej softmax vrstvy neurónových sietí



Obr. 6.15: Pomer nerozhodnutých vzorov sietí pre rôzne hodnoty w

najlepšiu presnosť klasifikácie. V oboch prípadoch sme vypočítali hranice pre každú triedu minimalizáciou účelovej funkcie, pričom sme použili rôzne hodnoty váhy w. Vplyv hodnôt váhy na pomer nerozhodnutých vzorov trénovacej množiny prezentujeme na obrázku 6.15.

Pre testovanie systému sme vyberali váhu w tak, aby pomer nerozhodnutých vzorov neprekročil 8 %. Na základe toho sme vybrali $w = 195$ pre ConvNet a $w = 200$ pre LeNet4. Výpočítané hranice pre tieto váhy sú v tabuľke 6.3.

Model	Hranica	Trieda									
		0	1	2	3	4	5	6	7	8	9
ConvNet **w = 195**	h_0	0.361	0.372	0.375	0.345	0.000	0.402	0.497	0.388	0.315	0.000
	h_1	0.927	0.983	0.978	0.983	0.922	0.987	0.919	0.956	0.989	0.995
LeNet4 **w = 200**	h_0	0.000	0.468	0.000	0.000	0.454	0.454	0.000	0.000	0.000	0.619
	h_1	0.001	0.469	0.001	0.001	0.455	0.965	0.001	0.001	0.999	0.989

Tabuľka 6.3: Vypočítané hranice pre siete podľa zvolenej váhy

Podľa týchto hraníc sme vypočítali akceptačnú maticu klasifikácie oboch neurónových sietí, ktorú uvádzame v tabuľke 6.4. Hodnoty tejto tabuľky predstavujú pre

Model	Správnosť klasifikácie	MNIST test			Fashion MNIST test		
		Prijaté	Nerozhodnuté	Zamietnuté	Prijaté	Nerozhodnuté	Zamietnuté
ConvNet	Správne	92,14 %	6,15 %	0,04 %	0,00 %	0,00 %	0,00 %
	Nesprávne	0,26 %	1,41 %	0,00 %	14,45 %	79,78 %	5,77 %
LeNet4	Správne	97,98 %	0,66 %	0,02 %	0,00 %	0,00 %	0,00 %
	Nesprávne	1,05 %	0,27 %	0,02 %	86,41 %	12,72 %	0,87 %

Tabuľka 6.4: Akceptačná matica sietí pre zvolené váhy

danú testovaciu množinu pomer prijatých, nerozhodnutých a zamietnutých klasifikácii vzorov, pričom rozlišujeme správne a nesprávne klasifikované vzory. Pri Fashion MNISTe uvádzame všetky klasifikácie ako nesprávne, pretože táto množina je z iného priestoru vzorov ako trénovacia množina.

Pomer správnych rozhodnutí je súčtom prijatých správnych klasifikácií a zamietnutých nesprávnych klasifikácií. Pomer nesprávnych rozhodnutí vyjadruje súčet prijatých nesprávnych klasifikácií a zamietnutých správnych klasifikácií. To znamená, že v prípade Fashion MNIST sa systém so sieťou LeNet4 rozhodol správne v 0,87 %, nerozhodol v 12,72 % a mýlil sa v 86,41 % vzorov. V prípade vzorov MNIST sa tento systém rozhodol správne v 98 %, nerozhodol v 0,93 % a mýlil sa v 1,07 % vzorov.

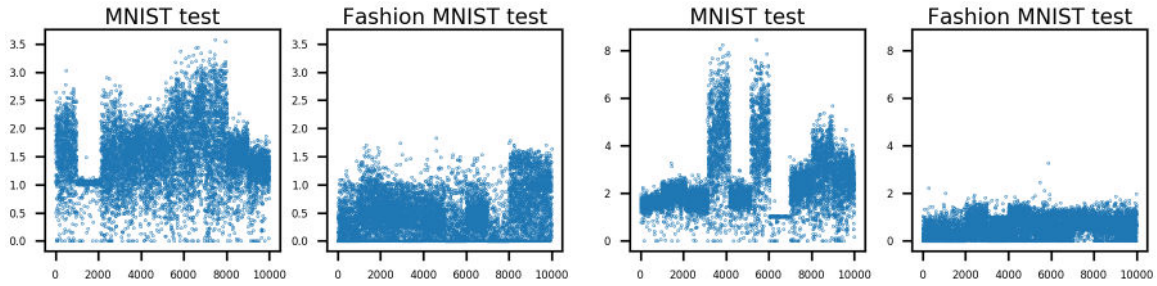
To, že sa sieť LeNet4 pomýlila pri väčšine vzorov Fashion MNISTu, je spôsobené tým, že bola zvolená váha, pre ktorú je pomer nerozhodnutých vzorov veľmi malý. ConvNet sa pri Fashion MNISTe pomýlila pri menšom počte vzorov ako LeNet4 a veľmi veľký počet vzorov označila ako nerozhodnuté.

Problémom pri tomto systéme je aj to, že výstupná vrstva sietí používa aktivačnú funkciu softmax, čo obmedzuje výstupné hodnoty len na interval $[0, 1]$. Skúsme ešte porovnať tieto siete s použitím ReLU ako aktivačnej funkcie výstupnej vrstvy.

Použitie ReLU vo výstupnej vrstve

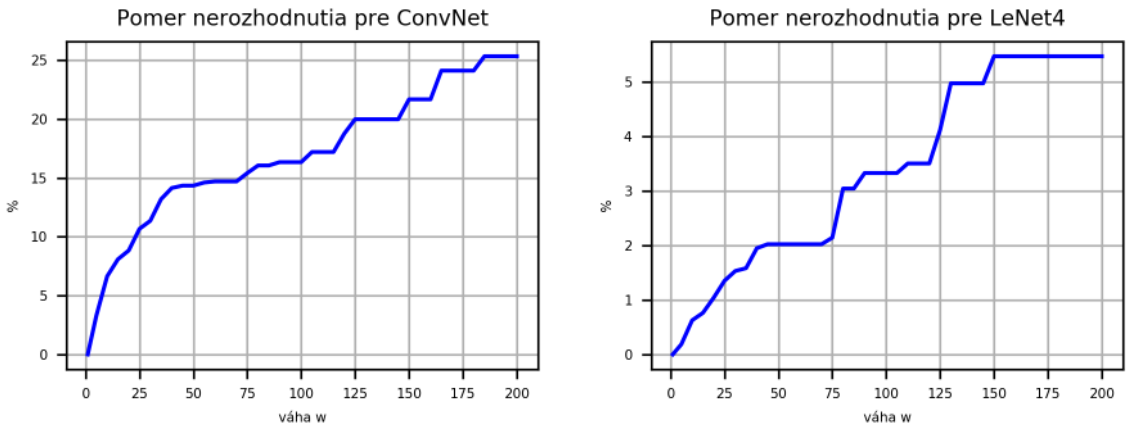
Pri tomto experimente zostane architektúra oboch sietí rovnaká, zmení sa len aktivačná funkcia výstupnej vrstvy. Výstupné hodnoty teda budú v intervale $[0, \infty)$. Trénovanie bude prebiehať rovnako ako v predchádzajúcom experimente.

Obrázok 6.16 ukazuje hodnoty výstupov pri použití ReLU vo výstupnej vrstve.



(a) ConvNet (b) LeNet4

Obr. 6.16: Hodnoty výstupnej ReLU vrstvy neurónových sietí



Obr. 6.17: Pomer nerozhodnutých vzorov sietí s ReLU pre rôzne hodnoty w

Hodnoty výstupov pre MNIST sa posunuli vyššie ako tomu bolo pri použití softmaxu. Hoci sa posunuli aj hodnoty datasetu Fashion MNIST, nedosahujú také hodnoty ako MNIST.

Skúsime teda nájsť hranice pre výstupy trénovacej množiny MNIST. Keďže sú hodnoty väčšie ako 1, aj hranice h_0 a h_1 budú väčšie, takže dostaneme intervaly:

- interval $[0, h_0)$ – pre hodnoty z tohto intervalu klasifikáciu zamietneme,
- interval $[h_0, h_1)$ – pre tieto hodnoty o klasifikácii nerozhodneme,
- interval $[h_1, \infty)$ – pre hodnoty z tohto intervalu klasifikáciu prijmeme.

Hľadanie hraníc bude opäť prebiehať tak, že budeme minimalizovať hodnotu účelovej funkcie z (6.2).

Podľa pomeru nerozhodnutých vzorov na obrázku 6.17 sme pre dosiahnutie menej ako 8 % nerozhodnutých vzorov vybrali váhy $w = 10$ pre ConvNet a $w = 200$ pre LeNet4. Hranice pre tieto váhy sú v tabuľke 6.5.

Model	Hranica	Trieda									
		0	1	2	3	4	5	6	7	8	9
ConvNet **w = 10**	h_0	0.000	0.040	0.004	0.004	0.052	0.004	0.004	0.004	0.072	0.048
	h_1	0.716	0.832	0.344	0.764	0.820	0.784	0.596	0.716	0.844	0.824
LeNet4 **w = 200**	h_0	0.000	0.000	0.010	0.060	0.250	0.220	0.080	0.170	0.070	0.120
	h_1	1.010	1.180	0.990	2.060	1.120	1.410	0.760	1.440	1.470	1.660

Tabuľka 6.5: Vypočítané hranice pre siete s ReLU podľa zvolenej váhy

Model	Správnosť klasifikácie	MNIST test			Fashion MNIST test		
		Prijaté	Nerozhodnuté	Zamietnuté	Prijaté	Nerozhodnuté	Zamietnuté
ConvNet	Správne	92,77 %	4,76 %	0,02 %	0,00 %	0,00 %	0,00 %
	Nesprávne	0,66 %	1,20 %	0,59 %	25,30 %	53,55 %	21,15 %
LeNet4	Správne	93,76 %	4,73 %	0,11 %	0,00 %	0,00 %	0,00 %
	Nesprávne	0,08 %	0,91 %	0,41 %	9,97 %	78,72 %	11,31 %

Tabuľka 6.6: Akceptačná matica sietí s ReLU pre zvolené váhy

Podľa hraníc sme podobne ako pri experimente so softmaxom vypočítali akceptačnú maticu klasifikácie neurónovej siete, ktorá je v tabuľke 6.6.

Z tabuľky vidíme zlepšenie výsledkov siete LeNet4, ktorá sa už nemýli pri takom počte vzorov Fashion MNISTu. Naopak, pri sieti ConvNet nastalo zhoršenie výsledkov, keď je prijatých viac vzorov z Fashion MNISTu.

Výsledky tohto experimentu ukazujú, že pri detekcii toho, či vzor patrí alebo nepatrí do daného priestoru, nie je táto metóda veľmi účinná. Výstup z poslednej vrstvy klasifikačnej neurónovej siete nenesie dostatočnú informáciu na to, aby sa z nej dalo spoľahlivo určiť, či je sieť trénovaná pre daný typ vzoru, ako ukazujú aj obrázky 6.14 a 6.16.

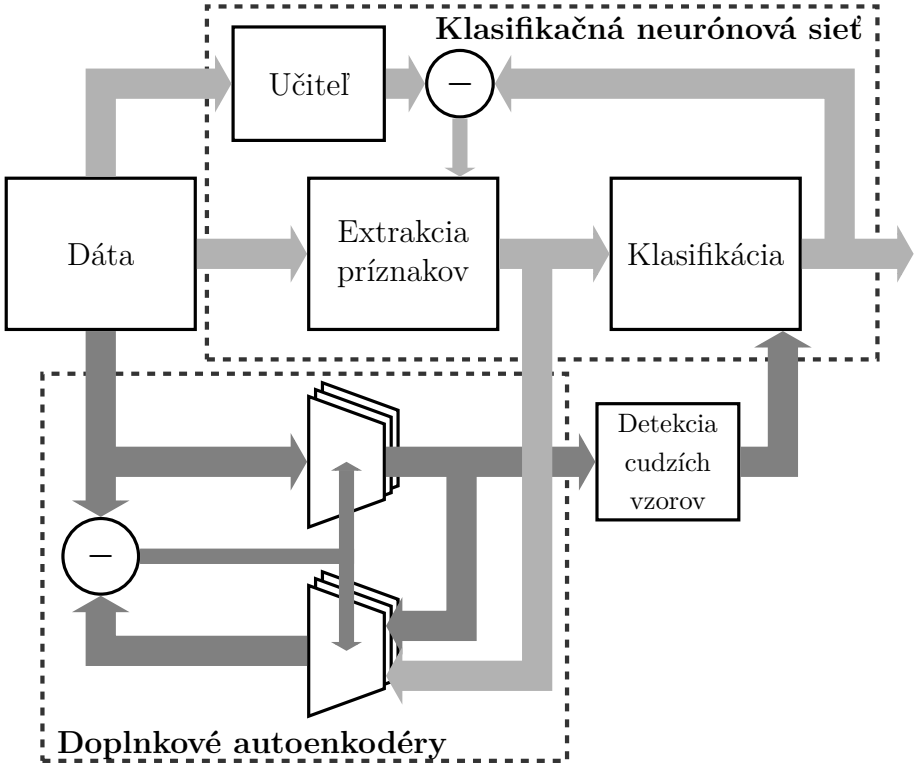
6.4.3 Detekcia autoenkodérmi

V tejto sekcii vyskúšame detegovať cudzie vzory tak, že použijeme globálnu klasifikačnú neurónovú sieť a k nej pridáme pre každú triedu samostatný lokálny autoenkodér. Z klasifikačnej neurónovej siete použijeme výstup predposlednej vrstvy ako doplnkovú informáciu pre autoenkodéry. Schéma tohto systému je znázornená na obrázku 6.18 na nasledujúcej strane.

Algoritmus

Klasifikačnú sieť budeme trénovať štandardne s celou trénovacou množinou. Každý autoenkodér však budeme trénovať len so vzormi trénovacej množiny patriacimi do danej triedy, pričom k trénovaniu budú potrebné aj výstupy predposlednej vrstvy klasifikačnej siete.

Pri testovaní systému budeme klasifikovať vzory testovacej množiny klasifikačnou sieťou. Potom sa pozrieme na doplnkové údaje získané kódovacou časťou prísluš-



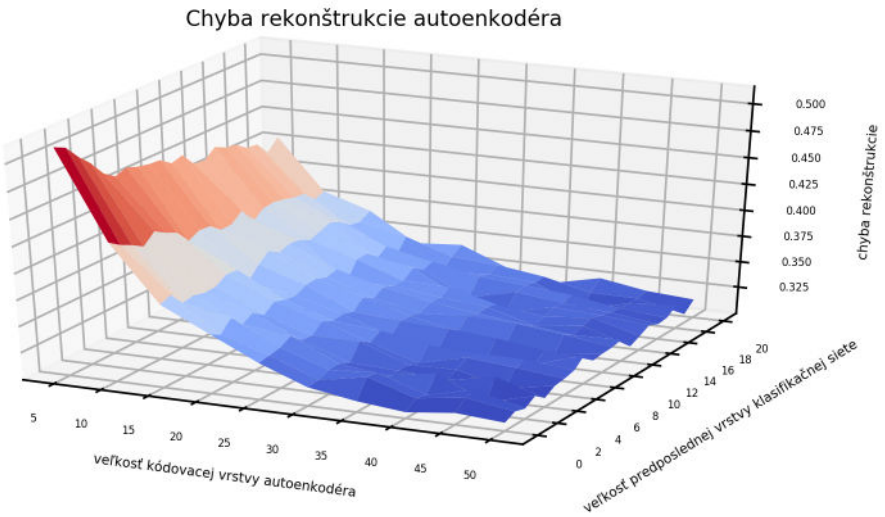
Obr. 6.18: Schéma systému detekcie neznámych vzorov s autoenkodérmi

ného autoenkodéra. Tieto údaje budeme volať chvostom vzoru. Pre rozhodnutie o príslušnosti vzoru do priestoru vzorov použijeme dĺžku jeho chvosta, ktorú vypočítame Euklidovou metrikou.

Veľkosť vrstiev

Veľkosť predposlednej vrstvy klasifikačnej siete a veľkosť kódovacej vrstvy autoenkodérov sme určili pomocou systému s jedným autoenkodérom, pričom sme ako klasifikačnú neurónovú sieť použili ConvNet z tabuľky 6.1. Kódovacia časť autoenkodéra mala architektúru Deep, dekódovacia časť mala vrstvy prislúchajúce kódovacej časti. Obe siete sme trénovali s rôznymi veľkosťami vrstiev. Klasifikačnú sieť sme trénovali tak, aby dosiahla čo najlepšiu klasifikačnú presnosť, autoenkodér tak, aby jeho chyba rekonštrukcie z (6.1) bola minimálna.

Na nasledujúcej strane na obrázku 6.19 je graf chyby rekonštrukcie autoenkodéra pri rôznej veľkosti kódovacej vrstvy autoenkodéra N a veľkosti predposlednej vrstvy klasifikačnej siete M. Hoci chyba rekonštrukcie klesá pri malom N, od $N = 40$ je chyba najmenšia a veľkosť M na ňu nemá vplyv. Preto sme sa rozhodli použiť veľkosť kódo-

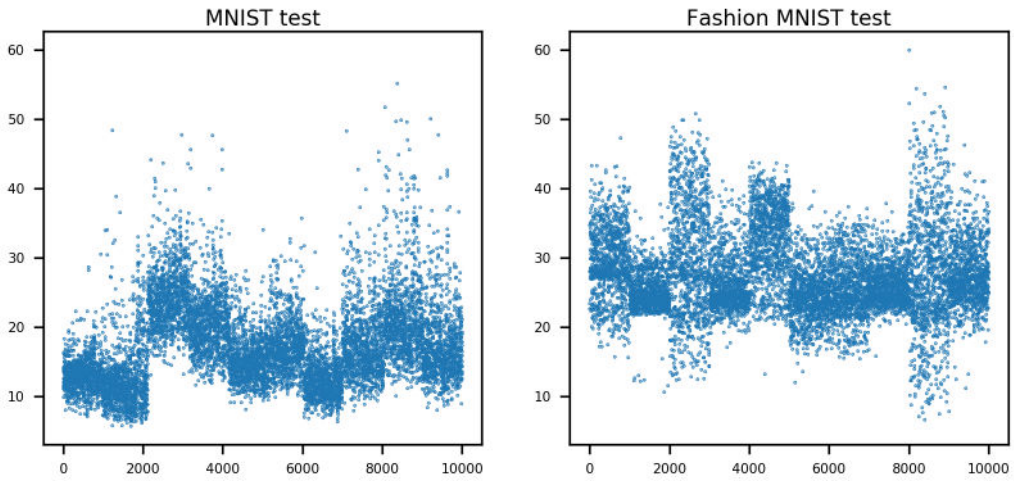


Obr. 6.19: Chyba rekonštrukcie autoenkodéra pre rôzne veľkosti vrstiev

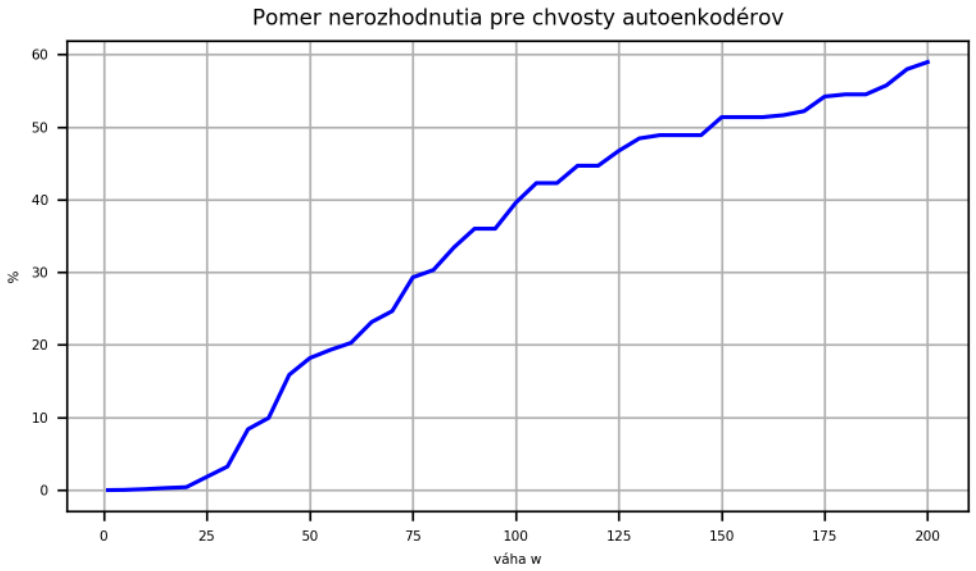
vacej vrstvy $N = 40$. Podľa tabuľky 6.2 sme veľkosť predposlednej vrstvy klasifikačnej siete zvolili $M = 4$, kde sieť ConvNet dosahuje presnosť klasifikácie takmer 98 %.

Výsledky

Všetky lokálne autoenkodéry sme teda natrénovali s veľkosťou kódovacej vrstvy 40. Potom sme vypočítali dĺžky chvostov vzorov trénovacej množiny, podľa ktorých sme určili hranice h_0 a h_1. Pre výpočet dĺžky chvostov Euklidovou metrikou je potrebné, aby boli jednotlivé premenné ortogonálne. Preto sme chvosty ešte upravili pomocou PCA. Na obrázku 6.20 je graf dĺžok chvostov vzorov testovacích množín MNIST a Fashion



Obr. 6.20: Dĺžky chvostov z autoenkodérov po ortogonalizácii



Obr. 6.21: Pomer nerozhodnutých vzorov z autoenkodérov pre rôzne w

MNIST po ortogonalizácii.

Rovnako ako pri predchádzajúcom experimente, aj tu sme určovali hranice h_0 a h_1 tak, aby bola hodnota účelovej funkcie (6.2) minimálna. Hranice v tomto experimente rozdeľujú interval $[0, \infty)$ na intervaly:

- interval $[0, h_0)$, z ktorého hodnoty prijmeme,
- interval $[h_0, h_1)$, o ktorého hodnotách nerozhodneme,
- interval $[h_1, \infty)$, z ktorého hodnoty zamietneme.

Podľa grafu pomeru nerozhodnutých vzorov z obrázku 6.21 sme pre dosiahnutie menej ako 8 % nerozhodnutých vzorov zvolili váhu $w = 30$. Pri tejto váhe sú hodnoty hraníc h_0 a h_1 nasledovné:

Hranica	Trieda									
	0	1	2	3	4	5	6	7	8	9
h_0	20.822	10.600	14.386	13.250	17.036	11.357	19.686	15.143	11.357	10.222
h_1	32.558	31.422	38.237	35.965	34.451	37.480	28.772	29.529	35.587	31.044

Tabuľka 6.7: Vypočítané hranice pre chvosty autoenkodérov podľa zvolenej váhy

Z hodnôt hraníc h_0 a h_1 pre zvolenú váhu sme vypočítali akceptačnú maticu, ktorá je v tabuľke 6.8 na nasledujúcej strane.

Správnosť klasifikácie	MNIST test			Fashion MNIST test		
	Prijaté	Nerozhodnuté	Zamietnuté	Prijaté	Nerozhodnuté	Zamietnuté
Správne	34,77 %	62,85 %	0,71 %	0,00 %	0,00 %	0,00 %
Nesprávne	0,08 %	1,03 %	0,56 %	1,60 %	81,71 %	16,69 %

Tabuľka 6.8: Akceptačná matica chvostov autoenkodérov pre zvolenú váhu

Ako vidíme z tabuľky, systém dosiahol pri Fashion MNISTe veľmi malú chybovosť (len 1,6 %), ale zároveň pri MNISTe nedokázal rozhodnúť až o 63,88 % vzorov. Môže to byť spôsobené tým, že veľkosť kódovacej vrstvy 40 je príliš malá. Pravdepodobnejšie ale je, že informácia z kódovacích častí autoenkodérov nie je dostatočná na to, aby bolo možné odlíšiť cudzí vzor.

6.4.4 Porovnanie

Predstavili sme dva systémy detekcie cudzích vzorov – systém s klasifikačnou neurónovou sieťou a systém, kde okrem globálnej klasifikačnej siete boli lokálne autoenkodéry poskytujúce doplnkovú informáciu pre rozhodnutie o tom, či je vzor z daného priestoru vzorov.

V oboch systémoch sme počítali akceptačnú maticu podľa zvolených hraníc h_0 a h_1. Čísla v tejto matici vyjadrovali pomer prijatých, zamietnutých a nerozhodnutých klasifikácií vzorov testovacích množín MNIST a Fashion MNIST.

Pre porovnanie týchto systémov uvádzame tabuľku správnosti rozhodnutí týchto systémov (tabuľka 6.9 na nasledujúcej strane). V systéme so samostatnou klasifikačnou sieťou sme porovnávali dva varianty – použitie softmax alebo ReLU ako aktivačnej funkcie výstupnej vrstvy. V oboch variantoch sme používali dve architektúry z tabuľky 6.1 – ConvNet a LeNet4. Z každého variantu uvádzame v tabuľke tú architektúru, ktorá dosiahla lepší výsledok.

V systéme s doplnkovými autoenkodérmi sme používali ako architektúru globálnej klasifikačnej siete ConvNet, autoenkodéry používali architektúru Deep.

Tabuľka 6.9 uvádza pomer správnych rozhodnutí, nesprávnych rozhodnutí a nerozhodnutí každého systému. Pomer správnych rozhodnutí je súčtom prijatých správnych klasifikácií a zamietnutých nesprávnych klasifikácií z akceptačných matíc, pomer nesprávnych rozhodnutí je súčtom prijatých nesprávnych klasifikácií a zamietnutých

Systém	MNIST test			Fashion MNIST test		
	Správne rozhodnuté	Nerozhodnuté	Nesprávne rozhodnuté	Správne rozhodnuté	Nerozhodnuté	Nesprávne rozhodnuté
samostatná sieť softmax (ConvNet)	92,14 %	7,56 %	0,30 %	5,77 %	79,78 %	14,45 %
samostatná sieť ReLU (LeNet4)	94,17 %	5,64 %	0,19 %	11,31 %	78,72 %	9,97 %
doplnkové autoenkodéry	35,33 %	63,88 %	0,79 %	16,69 %	81,71 %	1,60 %

Tabuľka 6.9: Porovnanie systémov detekcie cudzích vzorov

správnych klasifikácií. Pomer nerozhodnutých vzorov je súčtom nerozhodnutých správnych a nesprávnych klasifikácií vzorov systémom.

Systém so samostatnou klasifikačnou sieťou dosahuje vysoký pomer správnych rozhodnutí pri datasete MNIST, avšak pri Fashion MNISTe nesprávne rozhodol o 14,45 % vzorov pri použití softmaxu a o 9,97 % pri použití ReLU ako aktivačnej funkcie výstupnej vrstvy. Systém s doplnkovými autoenkodérmi síce bol opatrnejší pri rozhodnutí o klasifikácii Fashion MNISTu (nesprávne rozhodol len v 1,6 % vzorov), zároveň bol ale príliš opatrný pri MNISTe, kde nerozhodol o 63,88 % vzorov.

Na základe týchto výslekov konštatujeme, že ani jeden z týchto systémov nedokázal spoľahlivo rozlíšiť množinu vzorov patriacich do rovnakého priestoru ako trénovacia množina (MNIST) od množiny cudzích vzorov (Fashion MNIST).

7 Záver

Diplomová práca sa zaoberá porovnaním lineárnych a nelineárnych metód extrakcie príznakov s možnosťou spätnej rekonštrukcie vzoru na úlohe rozpoznávania vzorov, pričom ako databáza vzorov bola použitá databáza rukou písaných číslic MNIST. Porovnanie týchto metód spočívalo v porovnaní presnosti klasifikácie vzorov podľa extrahovaných príznakov. Pre klasifikáciu bola použitá metóda najbližšieho suseda. V priebehu riešenia práce bol jej cieľ špecifickejšie vymedzený na vývoj metódy odhaľovania cudzích vzorov, teda vzorov tried, ktoré nie sú obsiahnuté v trénovacej množine.

Práca je členená do 7 kapitol. V prvej kapitole popisujeme súčasný stav systému rozpoznávania vzorov a jeho častí. V kapitole 2 uvádzame ciele práce. V kapitole 3 sa venujeme použitým lineárnym metódam extrakcie príznakov – analýze hlavných komponentov (PCA), analýze nezávislých komponentov (ICA) a lineárnej diskriminačnej analýze (LDA). V kapitole 4 popisujeme neurónové siete a autoenkodéry ako nelineárnu metódu extrakcie príznakov. Kapitola 5 popisuje implementáciu týchto metód v programovacom jazyku Python. Taktiež popisuje databázu vzorov MNIST a databázu Fashion MNIST použitú pri detekcii cudzích vzorov.

Kapitola 6 sa venuje popisu výsledkov experimentov. V sekcii 6.1 sú porovnané lineárne metódy extrakcie príznakov pri spoločnej extrakcii príznakov z celej množiny vzorov a extrakcii príznakov zo vzorov každej triedy zvlášť. Sekcia 6.2 demonštruje extrakciu príznakov a klasifikáciu neurónovými sieťami. Porovnali sme viacero architektúr sietí a ukázali sme, ako tieto siete klasifikujú vzory a ako vyzerajú extrahované príznaky zo vzorov pomocou týchto sietí. Sekcia 6.3 sa venuje autoenkodérom, ktoré sú špeciálnym typom neurónových sietí. Tu sme porovnali rekonštrukciu vzorov a presnosť klasifikácie vzorov podľa extrahovaných príznakov autoenkodérmi. Posledná sekcia kapitoly 6 prezentuje navrhnuté systémy detekcie cudzích vzorov. Prvý navrhnutý systém

pracuje s výstupmi poslednej vrstvy klasifikačnej neurónovej siete. Druhý systém okrem globálnej klasifikačnej siete používa pre detekciu cudzích vzorov doplnkovú informáciu z kódovacích častí lokálnych autoenkodérov (túto informáciu sme nazvali chvostom). Presnosť detekcie cudzích vzorov bola vyhodnotená na testovacích množinách datasetov MNIST a Fashion MNIST, pričom ani jeden zo systémov sa neukázal ako spoľahlivý pri odlíšení cudzích vzorov.

Odporúčaním pre ďalšiu prácu je zmeniť prístup k určovaniu chvostov v systéme detekcie cudzích vzorov. Ukázalo sa, že určovanie chvostov ako doplnkovej informácie k extrahovaným príznakom je nedostatočné, preto navrhujeme určovať chvosty ako vzdialenosť medzi originálnym vzorom a jeho rekonštrukciou lokálnym autoenkodérom. Tento prístup bude použitý v pripravovanom článku [Kli+].

Bibliografia

[Cho+15] F. Chollet et al. *Keras: The Python Deep Learning library*. 2015. URL: `https://keras.io/` (cit. 01. 04. 2019).

[Bas+17] C. Baskin et al. „Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform“. In: *arXiv preprint arXiv:1708.00052* (2017).

[BS97] A.J. Bell a T.J. Sejnowski. „The “independent components” of natural scenes are edge filters“. In: *Vision research* 37 (1997), s. 3327–3338.

[BH07] M. Berthold a D.J. Hand. *Intelligent Data Analysis: An Introduction. Second Edition*. Springer, 2007. 526 s. ISBN: 978-3-540-43060-5.

[BK88] H. Bourlard a Y. Kamp. „Auto-association by multilayer perceptrons and singular value decomposition“. In: *Biological Cybernetics* 59 (1988), s. 291–294.

[DGG05] K. Delac, M. Grgic a S. Grgic. „Independent comparative study of PCA, ICA, and LDA on the FERET data set“. In: *International Journal of Imaging Systems and Technology* 15.5 (2005), s. 252–260.

[Fis36] R.A. Fisher. „The use of multiple measurements in taxonomic problems“. In: *Annals of Eugenics* 7 (1936), s. 179–188.

[GBC16] I. Goodfellow, Y. Bengio a A. Courville. *Deep Learning*. MIT Press, 2016. 775 s. ISBN: 978-0-262-03561-3.

[HA84] J. Hérault a B. Ans. „Réseaux de neurones à synapses modifiables: Décodage de messages sensoriels composites par une apprentissage non supervisé et permanent“. In: *Comptes Rendus de l'Académie des Sciences Paris* 299 (1984), s. 525–528.

[HJA85] J. Hérault, C. Jutten a B. Ans. „Détection de grandeurs primitives dans un message composite par une architecture de calcul neuromimétique en apprentissage non supervisé“. In: *10 Colloque sur le traitement du signal et des images* (1985), s. 1017–1022.

[HM88] G.E. Hinton a J.L. McClelland. „Learning representations by recirculation“. In: *Neural information processing systems*. 1988, s. 358–366.

[HP18] W. Homenda a W. Pedrycz. *Pattern Recognition: A Quality of Data Perspective*. John Wiley & Sons, 2018. 299 s. ISBN: 978-1-119-30282-7.

[Hot33] H. Hotelling. „Analysis of a complex of statistical variables into principal components“. In: *Journal of Educational Psychology* 24 (1933), s. 417–441.

[HKO01] A. Hyvärinen, J. Karhunen a E. Oja. *Independent Component Analysis*. John Wiley & Sons, 2001. 504 s. ISBN: 978-0-471-40540-5.

[HO00] A. Hyvärinen a E. Oja. „Independent component analysis: algorithms and applications“. In: *Neural networks* 13.4–5 (2000), s. 411–430.

[Ize08] A.J. Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer, 2008. 733 s. ISBN: 978-0-387-78188-4.

[Jam+13] G. James, D. Witten, T. Hastie a R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2013. 426 s. ISBN: 978-1-4614-7137-0.

[Jol02] I.T. Jolliffe. *Principal Component Analysis: Second Edition*. Springer, 2002. 488 s. ISBN: 0-387-95442-2.

[Kli+] M. Klimo, O. Škvarek, M. Mrázik a O. Šuch. „Avoiding a Pattern Recognition Sample Bias“. V príprave.

[LeC87] Y. LeCun. „Modeles connexionistes de l'apprentissage“. Diz. pr. Université de Paris VI, 1987.

[LeC+98] Y. LeCun, L. Bottou, Y. Bengio a P. Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86 (1998), s. 2278–2324.

[LCB98] Y. LeCun, C. Cortes a C.J.C. Burges. *The MNIST database of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (cit. 01. 04. 2019).

[MPH09] L.J.P. van der Maaten, E.O. Postma a H.J. van den Herik. *Dimensionality reduction: A comparative review*. Technical Report TiCC-TR 2009-005. Tilburg University, 2009.

[MP43] W.S. McCulloch a W. Pitts. „A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics* 5 (1943), s. 115–133.

[MP69] M. Minsky a S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969. 258 s.

[NFP12] H.T. Nguyen, K. Franke a S. Petrović. „Reliability in a feature-selection process for intrusion detection". In: *Reliable Knowledge Discovery*. Springer, 2012, s. 203–218.

[Pea01] K. Pearson. „On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2 (1901), s. 559–572.

[Ros58] F. Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65 (1958), s. 386–408.

[Wer74] P. Werbos. „Beyond regression: New tools for prediction and analysis in the behavioral sciences". Diz. pr. Harvard University, 1974.

[XRV17] H. Xiao, K. Rasul a R. Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[Zad04] B. Zadrozny. „Learning and evaluating classifiers under sample selection bias". In: *Proceedings of the 21st International Conference on Machine Learning*. 2004, s. 903–910.

Zoznam príloh

A CD médium

Na priloženom CD médiu sa nachádzajú:

- zdrojové kódy implementovaných programov,
- práca v elektronickej podobe,
- obrázky z práce vo vysokom rozlíšení,
- dáta k obrázkom a tabuľkám.