Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

Variační autoenkodér a úlohy pozorování v latentním prostoru

BAKALÁŘSKÁ PRÁCE

Studijní program: Aplikovaná informatika

Autor: Tomáš Faltejsek
Vedoucí práce: Ing. Ondřej Vadinský, Ph.D.
Konzultant práce: full consultant's name (incl. degrees)

Praha, květen 2023

Poděkování

Thanks.

Abstract

Jedním z předních rysů lidské inteligence je intuice a schopnost představovat si nové objekty. Variační autoenkodér je inovací na poli pravděpodobnostních modelů, umožňující architekturu modelů schopných syntézy zcela nových dat s využitím pozorování atributů v latentním prostoru. Teoretická charakteristika a možnosti využití variačního autoenkodéru jsou předmětem této bakalářské práce.

Keywords

keyword, important term, another topic, and another one

Obsah

Seznam obrázků

Note: Add a list of figures if the number of figures in the thesis text exceeds 20. A list of diagrams is applicable only if the author distinguishes between a figure and a diagram. The list of diagrams is included if the number of diagrams exceeds 20. This thesis template does not distinguish between a figure and a diagram.

Seznam tabulek

Note: Add a list of tables if the number of tables used in the thesis exceeds 20.

Seznam použitých zkratek

BCC Blind Carbon Copy
CC Carbon Copy
CERT Computer Emergency Response Team
CSS Cascading Styleheets
DOI Digital Object Identifier
HTML Hypertext Markup Language
REST Representational State Transfer
SOAP Simple Object Access Protocol
URI Uniform Resource Identifier
URL Uniform Resource Locator
XML eXtended Markup Language

Note: Add a list of abbreviations if the number of abbreviations used in the thesis exceeds 20 and the abbreviations used are not common.

Úvod

Introduction is a compulsory part of the bachelor's / diploma thesis. The introduction is an introduction to the topic. It elaborates the chosen topic, briefly puts it into context (there may also be a description of the motivation to write the work) and answers the question why the topic was chosen. It puts the topic into context and justifies its necessity and the topicality of the solution. It contains an explicit goal of the work. The text of the thesis goal is identical with the text that is given in the bachelor's thesis assignment, ie with the text that is given in the InSIS system and which is also given in the Abstract section.

Part of the introduction is also a brief introduction to the process of processing the work (a separate part of the actual text of the work is devoted to the method of processing). The introduction may also include a description of the motivation to write the work.

The introduction to the diploma thesis must be more elaborate - this is stated in more detail in the Requirements of the diploma thesis within the Intranet for FIS students.

Here are some sample chapters that recommend how a bachelor's / master's thesis should be set. They primarily describe the use of the LaTeX template, but general advice will also serve users of other systems well.

1. Východiska variačního autoenkodéru

1.1 Strojové učení

1.1.1 Algoritmus strojového učení

Definici algoritmu strojového učení výstižně shrnuje následující definice (Mitchell, 1997, str. 2):

> „*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*“

Algoritmy strojového učení lze obecně rozdělit na čtyři třídy – učení s učitelem (*supervised learning*), učení bez učitele (*unsupervised learnaing*), kombinaci učení s učitelem a učení bez učitele (*semi-supervised learning*) a posilovaného učení (*reinforcement learning*).

Toto dělení vychází ze zkušenosti E, resp. míry, do jaké má algoritmus strojového učení **povoleno** interagovat s datovou sadou (resp. jakou možnost taková datová sada nabízí). (Goodfellow et al., 2016)

Pro předmět této práce budou blíže představeny pouze oblasti **a učení s učitelem** a **učení bez učitele**.

1.1.2 Učení s učitelem

Zkušenost E algoritmů učení bez učitele vychází z datové sady, která může mít celou řadu vlastností a zároveň je **předem známá klasifikace každého datového bodu do definovaných tříd**. Na základě této asociace je natrénovaný algoritmus schopen provést přiřazení dosud neznámého objektu do jedné z tříd definovaných v trénovací sadě dat.

Učení s učitelem zahrnuje pozorování několika příkladů náhodného vektoru $\mathbf{x}$ a asociované hodnoty (či vektoru) $\mathbf{y}$. Následuje učení se predikovat $\mathbf{y}$ z $\mathbf{x}$, často na základě odhadu $p(\mathbf{y} \mid \mathbf{x})$.

Samotný termín *učení s učitelem* vychází ze situace, kdy je cílová třída $\mathbf{y}$ poskytnuta jakýmsi instruktorem či učitelem, který systému strojového učení ukazuje očekávané chování. (Goodfellow et al., 2016)

1.1.3 Učení bez učitele

Ve vztahu k definici (viz podsekce 1.1.1) lze říci, že zkušenost E algoritmů učení bez učitele vychází z datové sady, která může mít celou řadu vlastností. Cílem trénování algoritmů učení bez učitele je **naučit se užitečné a charakteristické vlastnosti o struktuře vstupní datové sady**. V kontextu hlubokého učení (podsekce 1.1.12) je pak obvyklým cílem algoritmu naučit se celé rozdělení pravděpodobnosti, které generuje původní datovou sadu (ať už explicitně – např. odhad hustoty, či implicitně – např. úlohy syntézy dat a odstranění šumu). Mezi další techniky strojového učení bez učitele se, mimo jiné, řadí shluková analýza.

Obecně lze říct, že učení bez učitele zahrnuje pozorování několika příkladů náhodného vektoru $\mathbf{x}$ na základě kterého se snaží implicitně či explicitně *naučit* rozdělení pravděpodobnosti $p(\mathbf{x})$, případně *užitečné vlastnosti* tohoto pravděpodobnostního rozdělení.

Na rozdíl od podsekce 1.1.2, název *učení bez učitele* napovídá, že v procesu učení není zapojen žádný *instruktor* či *učitel*, a tak systém strojového učení sám musí vyvodit smysl a užitečné vlastnosti předložené datové sady. (Goodfellow et al., 2016)

1.1.4 Perceptron

1.1.5 Hebbovské účení

1.1.6 Umělé neuronové sítě

1.1.7 Universal approximation theorem

1.1.8 Vícevrstvý Perceptron

1.1.9 Dopředná umělá neuronová síť

1.1.10 Gradient descent

1.1.11 Backpropagation

1.1.12 Hluboké učení

1.1.13 Konvoluční sítě

1.1.14 „No free lunch theorem“ pro strojové učení

Dle teorie má algoritmus strojového učení schopnost generalizace i z konečné množiny trénovacích dat. Toto tvrzení je ale v rozporu s elementárními principy logiky – indukce obecných pravidel z omezeného vzorku dat je logicky nevalidní. Chceme-li provést indukci obecného pravidla, které popisuje každý prvek množiny, musíme mít k dispozici informaci o každém prvku z množiny. (Goodfellow et al., 2016)

Pro logické vyvrácení takto naučené generalizace stačí byť jeden vzorek, který je s tímto pravidlem v nesouladu a nebyl součástí trénovací množiny (tzv. *black swan paradox*).

Oblast strojového učení se tomuto paradoxu z části vyhýbá tím, že pracuje pouze s **pravděpodobnostními pravidly** (oproti zcela určitým pravidlům jako v logické indukci). Algoritmy strojového učení hledají pravidla, která jsou tzv. *probably approximately correct*. (Valiant, 1984)

Ani tento trik však kompletně neřeší představený problém. Zjednodušeně, **"No free lunch theorem"** pro strojové učení tvrdí, že každý klasifikační algoritmus má v průměru, skrze všechny distribuce generující data, **stejnou chybovost** při klasifikování dosud nepozorovaných datových bodů. (Wolpert, 1996) Jinými slovy, **žádný algoritmus strojového učení není univerzálně lepší, než kterýkoliv jiný algoritmus strojového učení**.

Toto tvrzení je pravdivé až při zohlednění *všech možných distribucí generujících data* – tedy v

teoretické rovině. Lze pozorovat, že v praktických aplikacích je možné navrhnout algoritmus, který si v určitých distribucích vede v průměru lépe, než ostatní algoritmy. (Goodfellow et al., 2016)

Cílem této práce je představit Variační autoenkodér – architekturu umělé neuronové sítě, která se těší dobrým výsledkům ve vybraných úlohách učení bez učitele, představených v kapitola 4, ale v důsledku má i své nedostatky (představené v kapitola 3) v ostatních třídách úloh.

1.1.15 Regularizace

„No free lunch theorem“ pro strojové učení (představený v podsekce 1.1.14) implikuje nutnost návrhu algoritmu strojového učení pro konkrétní úlohu, chceme-li aby jeho výkonnost byla vyšší než výkonnost ostatních algoritmů v průměru. Toho lze docílit *zabudováním* určité sady preferencí přímo do algoritmu strojového učení (předpokladem je, že tyto preference jsou v souladu s cílovým problémem, který se algoritmus snaží řešit). (Goodfellow et al., 2016)

Chování a výkonnost algoritmu strojového učení lze ovlivnit zvolením velikosti množiny funkcí (a následně konkrétní podoby jejich identity), které jsou v jeho prostoru hypotéz povoleny[1]. V prostoru hypotéz algoritmu lze rovněž lze vyjádřit preferenci jednoho řešení před druhým. To znamená, že obě takové funkce budou přípustné, ale jedna z nich má preferenci (tedy nepreferovaná funkce bude zvolena když a pouze když je její evaluace vůči trénovací sadě *výrazně* lepší, než preferovaná funkce)[2]. Případně můžeme funkci (nebo množinu funkcí) z prostoru hypotéz vyřadit kompletně (resp. vyjádřit tak nekonečně velkou míru neupřednostnění takové funkce, či množiny funkcí). (Goodfellow et al., 2016)

Obecně můžeme regularizovat model, přičtením *trestu* k jeho ztrátové funkci. Tento trest nazýváme **regularizační prvek** (*regularizer*) a značíme jej Ω.

Regularizaci pak definujeme jako:

> „*Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. Regularization is one of the central concerns of the field of machine learning, rivaled in its importance only by optimization.*“ (Goodfellow et al., 2016)

Formu regularizace je tedy nutné pečlivě zvolit s ohledem na typ úlohy, který má algoritmus za cíl řešit. I autoenkodéry (a variačí autoenkodéry) mohou být navrhnuty k řešení celé řady problémů. Jednotlivé metody regularizace autoenkodérů jsou představeny v kapitola 2 a jejich následné aplikace v kapitola 4.

[1] Například prostor hypotéz lineární regrese je složen z množiny lineárních funkcí jejího vstupu. Tedy lineární regrese zřejmě bude mít problém věrohodně predikovat hodnotu $\sin(x)$ z x (a stejně tak řešit další nelineární problémy).

[2] Například *weight decay*.

1.2 Umělá neuronová síť

Umělá neuronová síť je model strojového učení inspirovaný přírodou. Zdá se být intuitivní, že chceme-li napodobit lidskou inteligenci, měli bychom se pro inspiraci podívat na architekturu lidského mozku. V průběhu času se však konstrukce umělých neuronových sítí začala jejich přírodnímu protějšku podstatně vzdalovat. Řada architektur umělých neuronových sítí tvoří biologicky nerealistický model, byť z této původní myšlenky vychází (stejně tak jako letadla vycházejí z přírodního vzoru létajících ptáků, pohybem svými křídly se ve skutečnosti ve vzduchu neodrážejí). (Geron, 2019)

Představení kompletních principů umělých neuronových sítí není východiskem variačního autoenkodéru, nýbrž svým obsahem pokrývají několik monografií – pro úvod například (Chollet, 2017), (Geron, 2019). V této sekci tedy budou představeny pouze stěžejní techniky využívané autoenkodéry.

1.2.1 Anatomie umělé neuronové sítě

Proces trénování umělé neuronové sítě z pravidla zahrnuje následující objekty (Chollet, 2017):

- *Vrstvy* ze kterých je následně složena *síť* (resp. *model*)
- *Vstupní data* (a případně jejich *cílové třídy*)
- *Ztrátová funkce*, která slouží jako signál zpětné vazby použití pro učení modelu
- *Optimizér*, který modifikuje parametry umělé neuronové sítě (např. váhy)

1.2.2 Perceptron

Jedna z nejjednodušších architektur umělé neuronové sítě (a zároveň *model umělého neuronu*) představena v (Rosenblatt, 1957) inspirována principy Hebbovského učení (Hebb, 1949). Perceptron přichází s důležitým principem **numerických hodnot vstupů, výstupů** (oproti pouhým binárním hodnotám) – tzv *linear treshold unit, LTU* a **vah** mezi jednotlivými neurony. Perceptron byl později kritizován (Minsky a Papert, 1969) za jeho neschopnost řešit triviální problémy (např. *XOR* klasifikace), což eventuálně vedlo ke krátkodobé stagnaci konekcionismu.

Jak se ale ukázalo, některé z těchto limitací lze vyřešit uspořádáním více Perceptronů za sebe do vrstev. (Rumelhart a McClelland, 1987)

Takto uspořádaná umělá neuronová síť se nazývá Vícevrstvý Perceptron.

1.2.3 Vícevrstvý Perceptron

Vícevrstvý Perceptron je tvořen umělou neuronovou sítí, která se skládá z jedné vstupní vrstvy, **jedné nebo více skrytých vrstev** LTU jednotek, a jedné výstupní vrstvy rovněž složené z LTU jednotek. Součástí každé vrstvy (vyjma výstupní) je tzv. **bias** neuron který je **plně propojený** s další vrstvou sítě. (Geron, 2019)

Pokud má umělá neuronová síť dvě a více skrytých vrstev, nazýváme ji **hlubokou neuronouvou sítí**.

Jak již víme, Vícevrstvý Perceptron adresuje limitace Perceptronu (viz podsekce 1.2.2). Článek (Rumelhart a McClelland, 1987) ale představil i další revoluční myšlenku – algoritmus **zpětné propagace** [3] (*backpropagation*).

Algoritmus zpětné propagace lze velmi zjednodušeně interpretovat následovně: Pro každou trénovací instanci je vypočten výstup každého jejího neuronu každé jednotlivé vrstvy. Poté je změřena výstupní chyba celé sítě (například střední kvadratická chyba, MSE) a vypočten podílem, jakým každý neuron poslední skryté vrstvy přispěl k hodnotě každého neuronu výstupní vrstvy. Následně je stejným způsobem měřeno, jak moc byla ovlivněna hodnota jednotlivých neuronů poslední skryté vrstvy hodnotami neuronů předchozí skryté vrstvy – a podobný proces se opakuje než algoritmus dosáhne vstupní vrstvy. (Geron, 2019)

Průchod algoritmu sítí lze intuitivně popsat následovně: Pro každou trénovací instanci algoritmus zpětné propagace nejprve učiní predikci cílové hodnoty – tzv. *forward pass*). Poté změří chybu této predikce. Následně zpětně projde každou vrstvu sítě za účelem změření míry přispění k této chybě každým propojením (a její váhou) – tzv. *reverse pass*. Závěrem algoritmus *lehce* upraví váhy jednotlivých propojení za účelem snížení chyby – tzv. *Gradient Descent step*. (Geron, 2019)

Pro správný chod algoritmu autoři (Rumelhart a McClelland, 1987) do architektury Vícevrstvého Perceptronu zanesli další zásadní změnu – jako tzv. **aktivační funkci** využili logistickou regresi (zcela běžné je využití i dalších aktivačních funkcí, například ReLU).

1.3 Redukce dimenzionality

Do oblasti redukce dimenzionality patří celá řada technik pro práci s vysokodimenzionálními daty. Cílem této oblasti, na rozdíl od regresních problémů, není predikovat hodnotu cílové proměnné – ale porozumět tvaru dat, se kterými pracuje. Typickou úlohou redukce dimenzionality dat je sestrojit **nízkodimenzionální reprezentaci**, která zachytí *většinu významu* původní, vysokodimenzionální, reprezentace. Tento jev nazýváme hledáním **salientních vlastnostní** původní sady dat. (Phillips, 2021)

[3] Algoritmus zpětné propagace byl ve skutečnosti nezávisle objeven více výzkumníky, počínaje (Werbos, 1974). Převzato z (Geron, 2019).

1.3.1 The curse of dimensionality

S rostoucím počtem vstupních proměnných exponenciálně roste počet vzorků dat nutný pro *libovolně přesnou* aproximaci dané funkce. V důsledku je tedy s rostoucí dimenzí vstupních dat značně degradováno chování většiny algoritmů (strojového učení). Tento problém je známý pod termínem **curse of dimensionality**. (Bellman, 1957)

I proto došlo k vzniku oblasti zvané *feature engineering*. Feature engineering je oblast, která se, mimo jiné, zabývá disciplínou selekce vlastností dat, které budou použity při trénování modelu. Pro automatizovanou selekci vlastností existuje celá řada technik. Výběr podprostoru, který *nejlépe* reprezentuje vlastnosti původních dat je NP-těžký kombinatorický problém (*exhaustive search through all the subsets of features*). Tyto techniky dokonce často vyhodnocují každou vstupní proměnnou nezávisle, což může vést ke zkresleným závěrům o jejich významnosti – naopak je běžné, že proměnné začínají vykazovat určitou míru významnosti **až při vzájemném využití**. (Stańczyk a Jain, 2015)

Výše stanovené důvody vedly k emergenci další disciplíny, a to **extrakce vlastností**, která je pro předmět této práce patřičně důležitější.

1.3.2 Extrakce vlastností

Cílem extrakce vlastností *feature extraction* je najít reprezentaci vstupních dat, která je vhodná pro algoritmus strojového učení, který se chystáme využít (jelikož původní reprezentace může být z mnoha důvodů nevhodná – například vysokodimenzionální). Typicky tak musí dojít k redukci dimenzionality vstupních dat. (H. Liu a Motoda, 1998)

K extrakci nových vlastností lze dojít mnoha způsoby. Existují techniky založené na hledání lineárních kombinací původních vstupních vlastností, například Analýza hlavních komponent (*PCA Analýza*) nebo Lineární diskriminační analýza (*LDA Analýza*).

Pro nelineární redukci dimenzionality lze využít technik tzv. manifold learningu.

1.3.3 Analýza hlavních komponent

1.3.4 Manifold learning

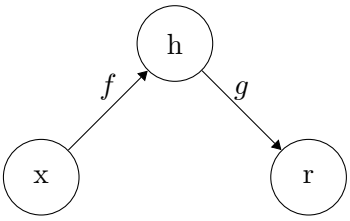
1.4 Pravděpodobnostní modely a inference pomocí variačního Bayese

1.5 Kullback–Lieblerova divergence

1.6 Modely využívající latentních proměnných // Latent Variable Models

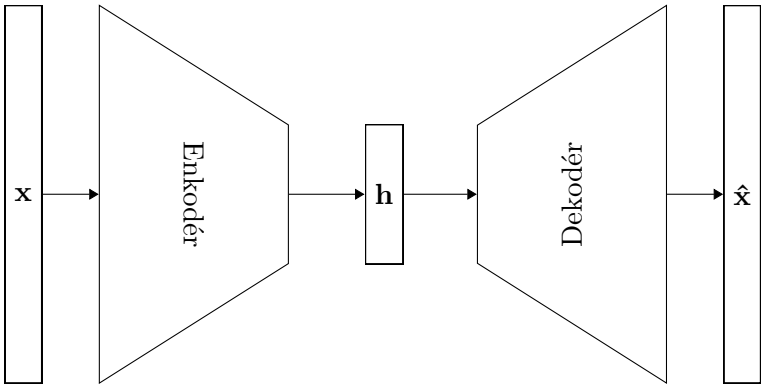
2. Autoenkodér

Autoenkodér je typ umělé neuronové sítě se schopností učit se efektivní reprezentace vstupních dat bez učitele [1]. Umělá neuronová síť Autoenkodéru má symetrickou strukturu a skrytou vrstvu h, která popisuje *kód* použitý pro reprezentaci vstupu. Architekturu Autoenkodéru (viz obrázek 2.1) lze principiálně rozdělit na dvě části – kódovací funkci $h = f(x)$, resp. **enkodér** a dekódovací funkci $r = g(h)$, resp. **dekodér**. Hovoříme tedy o typu umělé neuronové sítě s *enkodér-dekodér* moduly. Výstupem enkodéru je **kód** vstupu h. Výstupem dekodéru je **rekonstrukce** vstupu r. (Goodfellow et al., 2016)



Obrázek 2.1: Obecná struktura Autoenkodéru. Ze vstupu x je enkodérem vytvořen kód h (funkce f). Tento kód je následně dekodérem přetaven na rekonstrukci r (funkce g).

Obecnou strukturu (viz obrázek 2.1) lze reprezentovat dopřednou umělou neuronovou sítí. Jejím cílem je **rekonstruovat vstupní data na výstupní vrstvě** (tzv. *unsupervised learning objective*). Počet vstupů je tak totožný s počtem neuronů ve výstupní vrstvě umělé neuronové sítě (tedy x a r mají stejnou dimenzi). h může mít *menší* či *větší* dimenzi – volba dimenze h se odvíjí od požadovaných vlastností Autoenkodéru. Obecná architektura modulů umělé neuronové sítě Autoenkodéru je zachycena v obrázek 2.2. (Charte et al., 2018)



Obrázek 2.2: Jednotlivé moduly architektury umělé neuronové sítě Autoenkodéru.

[1] V literatuře se můžeme zřídka setkat s zařazením autoenkodérů **obecně** do třídy *semi-supervised* algoritmů strojového učení, například (Chollet, 2017, str. 95). Domnívám se, že přesnější formulací je zařazení **obecné** architektury autoenkodérů do třídy algoritmů učení bez učitele. Nutno předeslat (viz podsekce 2.5.1), že v architekturách hlubokých autoenkodérů je běžným jevem tzv. *stacked autoenkodér*. V instancích tohoto případu pak **lze** hovořit o zařazení autoenkodéru do třídy semi-supervizovaného učení. (Bengio et al., 2006), (Ranzato et al., 2007), (Erhan et al., 2010)

Autoenkodér je trénován k rekonstrukci jeho vstupů. Pokud by se Autoenkodér naučil jednoduše určit $x = g(f(x))$ pro každé x, získali bychom *identitu*, která není patřičně užitečná. Proto je při trénování zavedena řada omezení, jejichž účelem je zabránit možnosti naučení Autoenkodéru perfektně kopírovat vstupní data.

2.1 Historický pohled

Vícevrstvý Perceptron podsekce 1.1.8 je univerzálním aproximátorem podsekce 1.1.7 – tedy historicky nalézá uplatnění zejména v klasifikačních úlohách učení s učitelem. Sofistikovaný algoritmus se schopností trénování Vícevrstvého Perceptronu s větším počtem skrytých vrstev stále schází, a to zejména v důsledku problému mizejícího gradientu (*vanishing gradient problem*). Až příchod algoritmu gradientního sestupu podsekce 1.1.10, který adresuje problém mizejícího gradientu v aplikacích s použitím konvolučních sítí podsekce 1.1.13 a úloh učení se bez učitele, značí počátek moderních metod hlubokého učení. V oblasti hlubokého učení podsekce 1.1.12 dochází k emergenci a vývoji řady technik pro řešení úloh učení se bez učitele. V této kapitole je popsána pouze jedna z nich – architektura umělé neuronové sítě založené na *enkodér-dekodér* modulech: Autoenkodér. Autoenkodéry byly poprvé představeny jako způsob pro předtrénování umělých neuronových sítí (formou automatizované extrakce vlastností *feature extraction*). Později Autoenkodéry nalézají uplatnění zejména v úloháh redukce dimenzionality sekce 1.3 či fůzi vlastností (*feature fusion*).

Nedávné teoretické propojení Autoenkodéru a Modelů využívajících latentní proměnných sekce 1.6 však vedlo ke vzniku zcela nové architektury neuronové sítě kombinující charakter redukce dimenzionality Autoenkodéru se statistickými metodami odvozování. To vyneslo Autoenkodéry na popředí v oblasti generativního modelování – této architektuře je věnována kapitola kapitola 3.

Byť Autoenkodéry vznikly v kontextu hlubokého učení, není pravidlem že všechny modely Autoenkodéru obsahují vícero skrytých vrstev. Následuje rozdělení Autoenkodérů dle struktury umělé neuronové sítě.

2.2 Mělký autoenkodér

Mělký autoenkodér (*Shallow autoencoder*), resp. jeho umělá neuronová síť, je sestaven pouze ze tří vrstev – vstupní, kód a výstupní. Je to **nejtriviálnější model autoenkodéru**, jelikož jeho skrytá vrstva (kód) je pouze **jednovrstvá**.

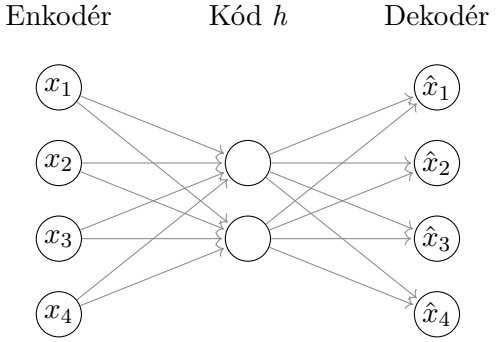
2.3 Autoenkodér s neúplnou skrytou vrstvou

Autoenkodér s neúplnou skrytou vrstvou (*Undercomplete autoencoder*) je Autoenkodér, jehož dimenze kódu (h) je menší, než dimenze vstupu. Tuto skrytou vrstvu h nazýváme **bottleneck**. Bottleneck je způsob, kterým se Autoenkodér s neúplnou skrytou vrstvou učí kompresované reprezentaci znalostí. V důsledku bottleneck vrstvy je Autoenkodér nucen zachytit pouze ty stěžejní vlastnosti trénovacích dat, které následně budou použity pro rekonstrukci.

Trénovací proces Neuplného autoenkodéru je popsán jako minimalizace ztrátové funkce:

$$L(\mathbf{x}, g(f(\mathbf{x}))), \tag{2.1}$$

kde L je ztrátová funkce, penalizující $g(f(\mathbf{x}))$ za *rozdílnost* vůči $\mathbf{x}$ (např. *střední kvadratická chyba*).



Obrázek 2.3: Jednoduchá architektura umělé neuronové sítě Autoenkodéru s neúplnou skrytou vrstvou. Skrytá vrstva představuje *bottleneck*.

Od Analýzy hlavních komponent po Autoenkodér

Máme-li linéární dekodér (Autoenkodér používá pouze linéárni aktivační funkce) a jako ztrátová funkce L je použita *střední kvadratická chyba*, pak se Neuplný autoenkodér naučí stejný *vektorový prostor*, který by byl výsledkem Analýzy hlavních komponentů podsekce 1.3.3. V tomto speciálním případě lze ukázat, že Autoenkodér trénovaný na úloze kompresované reprezentace znalostí jako vedlejší efekt provedl Analýzu hlavních komponentů.

Důležitým důsledkem tohoto jevu je, že **Autoenkodéry** s nelineární kódovací funkcí f a nelineární dekódovací funkcní g **jsou schopny učit se obecnější generalizaci** než u Analýzy hlavních komponent.

Na druhou stranu, má-li Autoenkodér k dispozici příliš mnoho kapacity, může se naučit kopírovat vstupní data na výstupní vrstvu bez extrakce užitečných (charakteristických) vlastností o rozdělení vstupních dat.

Problém s naučením pouhého identického zobrazení

Extrémním případem je teoretický scénář, ve kterém je Autoenkodér složen z kódu (h) o jedné vrstvě a velmi výkonného enkodéru. Takový Autoenkodér by se mohl naučit reprezentovat každý vstup x_i kódem i. Dekodér by se pak tyto indexy mohl naučit mapovat zpátky na hodnoty konkrétních trénovacích vzorků dat. Tento příklad se v praxi běžně nenaskytne, nicméne jasně ilustruje, jak může Autoenkodér při úloze kopírování vstupu na výstupní vrstvu selhat naučit se užitečné vlastnosti o vstupních datech, jsou-li restrikce při učení příliš nízké. Proto je třeba Autoenkodéry regularizovat.

Dále tedy budou představeny přístupy k architekturám Autoenkodérů **s využitím regularizace**.

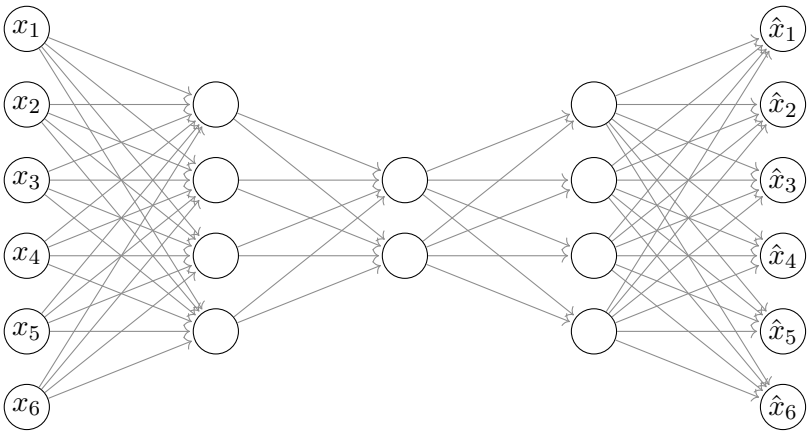
2.4 Autoenkodér s rozšířenou skrytou vrstvou

Autoenkodér s rozšířenou skrytou vrstvou (*Overcomplete Autoencoder*) je Autoenkodér, jehož počet neuronů ve skrýté vrstvě je větší než počet neuronů vstupní (a výstupní) vrstvy.

2.5 Hlubuký autoenkodér

V sekci sekce 2.3 a sekce 2.4 byly představeny Autonkodéry s jednovrstvým enkodérem a s jednovrstvým dekodérem. Existují však i Autoenkodéry, jejichž enkodér a dekodér moduly jsou vícevrstvé, tedy mají hloubku vyšší než jedna.

Hluboký autoenkodér (*Deep* autoenkodér), je Autoenkodér s netriviální hlobkou skryté vrstvy (*kód*) $\boldsymbol{h}$.



Obrázek 2.4: Deep Autoenkodér.

Z netriviální hloubky dopředné umělé neuronové sítě plyne podsekce 1.1.7, který garantuje, že

dopředná umělá neuronová síť s alespoň jednou skrytou vrstvou dokáže aproximovat (*libovolně přesně*) jakoukoliv funkci (za předpokladu dostatečného počtu neuron skryté vrstvy).

Nicméně u mělkých enkodérů, které jsou rovněž dopřednou sítí, neexistuje možnost představit libovolná omezení a regularizační prvky (například řídkost *kódu* $\boldsymbol{h}$ – viz sekce 2.6). A tedy nelze zamezit problému naučení pouhé identity sekce 2.3. Narozdíl od mělkých autoenkoderů sekce 2.2, však mohou Hluboké autoenkodéry (*libovolně přesně*) aproximovat jakékoliv mapování vstupu na kód (*opět s předpokladem dostatečného počtu neuronů skryté vrstvy*).

S netriviální hloubkou se rovněž pojí významná redukce výpočetních nákladů spojených s reprezentací některých funkcí. V důsledku možnosti efektivnější reprezentace takových funkcí dochazí i k zmenšení nároků na velikost množiny trénovacích dat.

2.5.1 Stacked autoenkodér

Běžnou strategií pro trénování Hlubokého autoenkodéru je hladově předtrénovat (*greedy pretraining*) model individuálním natrénováním většího počtu Mělkých autoenkodérů (představených v sekci sekce 2.2), které jsou následně vloženy za sebe. Takto složený Autoenkodér nazýváme Stacked autoenkodér.

2.6 Řídký autoenkodér

Řídká reprezentace dat (*sparsity*) ve strojovém učení znamená, že většina hodnot daného vzorku je nulová. Motivací pro řídkou reprezentaci dat ve strojovém učení je napodobení chování buněk v primární zrakové oblasti (*V1*) mozku savců. Konkrétně schopnosti odhalit a uložit efektivní kódovací strategie pozorovaných vjemů.

Pro sestrojení Řídkého autoenkodérů je tedy nutné představit omezení (regularizační prvek) hodnot aktivací neuronů ve skryté (kódovací) vrstvě $\boldsymbol{h}$ (resp. počtu aktivních neuronů ve skrýte vrstvě).

Řídký autoenkodér (*Sparse Autoecoder*) je Autoenkodér, jehož ztrátová funkce je rozšířena o penalizaci řídkosti kódovací vrstvy $\boldsymbol{h}$ (tzv. *sparsity penalty*) vztahem $\Omega(\boldsymbol{h})$:

$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\boldsymbol{h}), \tag{2.2}$$

kde Ω je **regularizační prvek**, jehož cílem je přiblížit hodnoty aktivací neuronů kódovací vrstvy k cílové hodnotě (a zabránit přeučení). Chceme tak penalizovat neurony kódovací vrstvy, které se aktivují příliš často.

Běžně lze Ω stanovit následovně. Mějme Bernoulliho náhodnou proměnou i modelující aktivace neuronů skrýte (kódovací) vrstvy – můžou tedy nastat dva stavy: neuron skryté vrstvy

je buď aktivován, nebo není aktivován. Pro konkrétní vstup x dostaneme:

$$\hat{p}_i = \frac{1}{|S|} \sum_{x \in S} f_i(x), \tag{2.3}$$

kde $f = (f_1, f_2, \ldots, f_c)$, c je počet neuronů skryté (kódovací) vrstvy a $\hat{p}_i$ je průměrná aktivační hodnota neuronu skryté vrstvy (resp. střední hodnota příslušného Bernoulliho schématu).

Dále mějme p jako cílové rozdělení aktivací. Kullback-Leibnerova divergence mezi náhodnou proměnnou i a p pak udává rozdíl obou rozdělení:

$$KL(p \parallel \hat{p}_i) = p \log \frac{p}{\hat{p}_i} + (1-p) \log \frac{1-p}{1-\hat{p}_i}. \tag{2.4}$$

Výsledný penalizační prvek Ω pro Řídký autoenkodér má tedy následující podobu:

$$\Omega_{\check{\mathrm{R}}AE}(W, b; S) = \sum_{i=1}^{c} KL(p \parallel \hat{p}_i), \tag{2.5}$$

kde průměrná hodnota aktivací $\hat{p}_i$ závisí na parametrech enkodéru a množině trénovacích dat S.

Přičtením tohoto penalizačního prvku ke ztrátové funkci (a následnou minimalizací celkové ztrátové funkce) je Autoenkodér nucen **omezit počet aktivních neuronů v skrýté (kódovací) vrstvě**. V důsledku tohoto omezení pak každý neuron skryté vrstvy reprezentuje nějakou **salientní vlastnost** vstupních dat (a rovněž je zamezeno naučení pouhé identity sekce 2.3).

2.7 Denoising autoenkodér

Denoising autoenkodér je autoenkodér, který na vstupu obdrží poškozená vstupní data a při trénování je jeho předpovědět originální vstup bez poškození, a ten na výstupní vrstvě vrátit.

Autoenkodéry běžně minimalizují funkci ve tvaru:

$$L(\mathbf{x}, g(f(\mathbf{x}))), \tag{2.6}$$

kde L je ztrátová funkce penalizující $g(f(x))$ za odlišnost od $\mathbf{x}$ (např. Euklidovská norma jejich rozdílů). Jak ale bylo ukázáno v sekce 2.3, to umožňuje $f \circ g$ naučit se být pouhou identitu.

Z toho důvodu Denoising autoenkodér (*Denoising Autoencoder*) minimalizuje funkci:

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))), \tag{2.7}$$

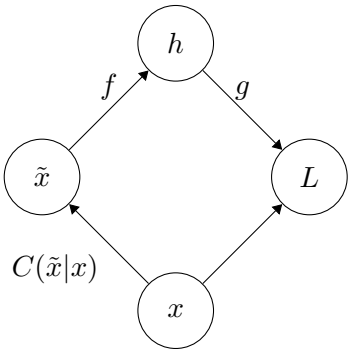
kde $\tilde{\boldsymbol{x}}$ je kopií $\boldsymbol{x}$ která byla úmyslně poškozena procesem $C(\tilde{x}|x)$ (*corruption process*), který reprezentuje podmíněné rozdělení pravděpodobnosti poškozenných vzorků $\tilde{x}$ v závislosti na vzorku vstupních dat x.

Denoising autoenkodér se pak učí **rozdělení rekonstrukce** $p_{reconstruct}(\mathbf{x}|\tilde{\mathbf{x}})$, které je odhadnuto z trénovacích dvojic následovně:

1. Zvolit trénovací vzorek $\boldsymbol{x}$ z množiny trénovacích dat
2. Vygenerovat poškozenou verzi zvoleného vzorku ($\tilde{\boldsymbol{x}}$) procesem C
3. Použít dvojice ($\boldsymbol{x}$, $\tilde{\boldsymbol{x}}$) jako množinu trénovacích dat pro odhadnutí rozdělení rekonstrukce Denoising autoenkodéru $p_{reconstruct}(\mathbf{x}|\tilde{\mathbf{x}}) = p_{decoder}(\boldsymbol{x}|\boldsymbol{h})$, kde $p_{decoder}$ je výstupem funkce dekodéru $g(\boldsymbol{h})$

Při trénování Denoising autoenkodéru jsou funkce f a g nuceny zachytit implicitní strukturu $p_{data}(\boldsymbol{x})$.

Trénovací procedura Denoising autoenkodéru lze schematicky znázornit následovně (viz obrázek 2.5):



Obrázek 2.5: DAE

Denoising autoenkodér se tedy musí naučit toto poškození odstranit a rekonstruovat tak původní vstup (namísto pouhého naučení se identitě).

Denoising Autoenkodéry jsou příkladem hned dvou jevů:

- Emergence užitečných vlastností o vstupních datech jako výsledek minimalizace chyby rekonstrukce
- Schopnosti modelů s vysokou kapacitou/rozšířenou skrytou vrstvou fungovat jako Autoenkodér, **za předpokladu že je jim zabráněno naučit se identické zobrazení vstupních dat**

2.8 Robustní autoenkodér

Robustní autoenkodér (*Robust autoencoder*) představuje další způsob, jakým se vypořádat s *drobným šumem* ve vstupních datech, která má následně rekonstruovat.

Robustní autoenkodéry, narozdíl od Denoising autoenkodéru (viz sekce 2.7), využivájí alternativně definovanou ztřatovou funkci, která je modifikována pro minimalizaci chyby rekonstrukce. Obecně jsou ne-Gaussovským šumem ovlivněny **méně než triviální mělké autoenkodéry** (viz sekce 2.2). Tato alternativní ztrátová funkce je založena na correntropii (*correntropy*, lokalizovaná míra podobnosti), která je v **CITACE** definována následovně:

$$\mathcal{L}_{MCC}(u, v) = -\sum_{k=1}^{d} \mathcal{K}_\sigma(u_k - v_k), \tag{2.8}$$

kde

$$\mathcal{K}_\sigma(\alpha) = \frac{1}{\sqrt{2\pi\sigma}} \exp(-\frac{\alpha^2}{2\sigma^2}), \tag{2.9}$$

a σ je parameter kernelu $\mathcal{K}$.

Correntropie měří hustotu pravdepodobnostni, že dvě *události* jsou si rovny (zde událost může být např.:). Correntropie je **výrazně méně ovlivněna odlehlými hodnotami**, než např. střední kvadratická chyba. (W. Liu et al., 2006) Robustní autoenkodér se snaží tuto míru maximalizovat, což intuitivně vede k **vyšší odolnosti** Robustního autoenkodéru **na ne-Gaussovský šum** přítomný ve vstupních datech.

2.9 Contractive autoenkodér

Přehnaná citlivost na *drobné rozdíly* ve vstupních datech by mohla vést k architektuře Autoenkodéru, která pro velmi podobné vstupy generuje odlišné kódy.

Contractive autonekodér (*CAE*), je Autoenkodér, který je při trénování omezen regularizačním prvkem, který vynucuje aby derivace kódů ve vztahu k jejich vstupu byly co možná nejmenší. Tedy **dva *podobné* vstupy musí mít vzájemně *podobné* kódy**. Přesněji je dosaženo lokální invariance na přípustně malé změny vstupních dat.

Citlivost na *drobné rozdíly* ve vstupních datech lze měřit pomocí Frobeniovy normy $\|\cdot\|_F$ Jacobiho matice enkodéru (J_f):

$$\|J_f(x)\|_F^2 = \sum_{j=1}^{d} \sum_{i=1}^{c} \left(\frac{\partial f_i}{\partial x_j}(x) \right)^2. \tag{2.10}$$

Čím vyšší je tato hodnota, tím více bude kód nestabilní s ohledem na *drobné rozdíly* ve vstupních datech. Z této metriky je následně sestaven **regularizační prvek** který je připočten k hodnotě ztrátové funkce Contractive Autoenkodéru:

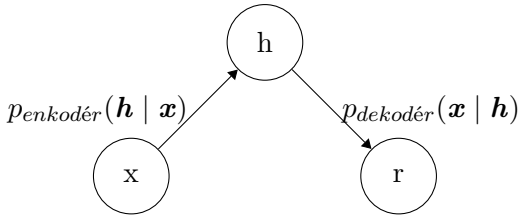
$$\Omega_{CAE}(W, b, S) = \sum_{x \in S} \|J_f(x)\|_F^2. \tag{2.11}$$

Výsledkem je tedy Autoenkodér, jehož dva (lokálně) *podobné* vstupy musejí mít i *podobný* kód. Z Contractive autoenkodéru lze rovněž vzorkovat nové výstupy. Z takto naučeného modelu Autoenkodéru lze generovat nové instance dat: Jakobián (*Jacobiho determinant*) enkodéru je (jako *drobný šum*) přičten ke kódu vstupu. Takto modifikovaný kód je poté dekodérem přetaven na výstup a dostáváme nový vzorek dat.

2.10 Stochastický autoenkodér

Struktura stochastického autoenkodéru se vůči struktuře představené v obrázek 2.1 liší reprezentací enkodér a dekodér modulů. V stochastickém autoenkodéru jsou enkodér a dekodér moduly reprezentovány rozdělením pravděpodobností (nejedná se tedy pouze o funkce, ale moduly zahrnují i určitou míru šumu). Výstup těchto modulů tedy obdržíme **výběrem z příslušného rozdělení pravěpodobnosti**.

Mějme skrytou vrstvu (*kód*) $\boldsymbol{h}$. Obecně má pro enkodér toto rozdělení podobu $p_{enkodér}(\boldsymbol{h} \mid \boldsymbol{x})$ a pro dekodér $p_{dekodér}(\boldsymbol{x} \mid \boldsymbol{h})$. Modifikací základní struktury autoenkodéru (obrázek 2.1) tedy dostáváme:



Obrázek 2.6: Obecná struktura Autoenkodéru. Ze vstupu x je enkodérem vytvořen kód h (funkce f). Tento kód je následně dekodérem přetaven na rekonstrukci r (funkce g).

V tradiční dopředné umělé neuronové síti podsekce 1.1.9 je běžnou strategií pro návrh výstupní vrstvy definování (*výstupního*) rozdělení pravděpodobnosti $p(\boldsymbol{y} \mid \boldsymbol{x})$. Pro návrh ztrátové funkce pak minimalizace záporného logaritmu věrohodnosti $-\log p(\boldsymbol{y} \mid \boldsymbol{x})$. V takové architektuře je $\boldsymbol{x}$ vektor vstupních dat a $\boldsymbol{y}$ vektor cílových proměnných, které se umělá neuronová síť snaží předpovědět (např. štítků jednotlivých tříd).

S architekturou autoenkodérů se však pojí jeden zásadní rozdíl. V autoenkodéru **je $\boldsymbol{x}$ jak vstupní, tak cílová proměnná**.

Dekodér pak lze interpretovat jako modul poskytující podmíněné rozdělení $p_{dekodér}(\boldsymbol{x} \mid \boldsymbol{h})$. Autoenkodér lze trénovat minimalizací $-\log p_{dekodér}(\boldsymbol{x} \mid \boldsymbol{h})$. Konkrétní podoba ztrátové funkce se odvíjí od požadovaných vlastností autoenkodéru a od přesné podoby modulu dekodéru (pokud hodnoty $\boldsymbol{x} \in \mathbb{R}$, pak jsou pro parametrizaci normálního rozdělení použity linéarní výstupní jednotky, tedy $-\log p_{dekodér}(\boldsymbol{x} \mid \boldsymbol{h})$ vrací *střední kvadtratickou chybu*).

Stochastický autoenkodér tedy **generalizuje kódovací funkci** $f(x)$ na **kódovací rozdělení pravděpodobnosti** $p_{enkodér}(\boldsymbol{h} \mid \boldsymbol{x})$.

Enkodér a dekodér moduly stochastického autoenkodéru tedy lze považovat za **modely využívající latentní proměnné** (*latent variable models*, viz sekce 1.6). Model využívající latentní proměnné značíme $p_{model}(\boldsymbol{h}, \boldsymbol{x})$.

Stochastický enkodér je pak definován následovně:

$$p_{enkodér}(\boldsymbol{h} \mid \boldsymbol{x}) = p_{model}(\boldsymbol{h}, \boldsymbol{x}) \tag{2.12}$$

a **stochastický dekodér** následovně:

$$p_{dekodér}(\boldsymbol{x} \mid \boldsymbol{h}) = p_{model}(\boldsymbol{x}, \boldsymbol{h}) \tag{2.13}$$

Výběr nových dat z rozdělení pravděpodobnosti autoenkodéru je detailněji představen v kaptiole kapitola 3.

2.11 Ostatní typy autoenkoderů

Na poli výzkumu strojového se autoenkodéry těší velkému úspěchu. Přirozeně tak existuje celá řada architektur a modifikací, které slouží k specifickým účelům. V této kapitole byly představeny pouze ty architektury autoenkodéru, na kterých bude později stavět kapitola kapitola 4.

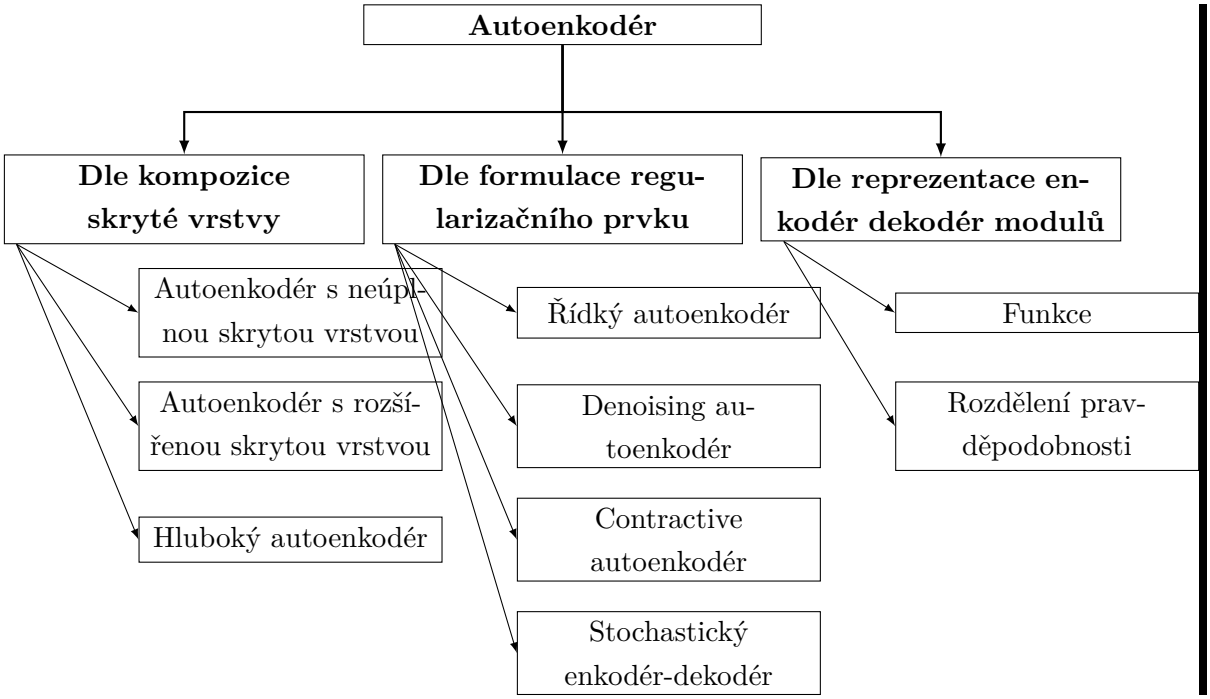
V této sekci budou stučně shrnuty *ostatní* typy autoenkodérů, které nejsou pro předmět nadcházejících kapitol stěžejní, ale obecně se jim dostává vysoké míry využití.

2.11.1 Adversariální autoenkodér

Adversariální autoenkodér (*Adversarial autoencoder*) přináší koncept Generativních Adversariálních sítí **CITACE** na pole autoenkodérů. V adversariálním autoenkodéru je kód ($\boldsymbol{h}$) modelován uložením předchozího statistického rozdělení (*prior*). Následně je natrénován *běžný* autoenkodér, současně se *diskriminativní* síť snaží odlišit kódy modelu od výběrů dat z přechozího statistického rozdělení. Jelikož generátor (v tomto případě enkodér) je trénován k *přelstění* diskriminátoru, kódy mají tendenci následovat uložené rozdělení pravděpodobnosti. Tím pádem, z Adversariálního autoenkodéru lze **provádět výběr zcela nových dat** (*generovat nové vzorky*).

2.12 Taxonomie autoenkodérů

Bylo představeno několik tříd Autoenkoderů (rozdělení dle navržení ANN sítě, rozdělení dle regularizačního prvku) a následně popsáno několik dalších druhú AE. Zde je jejich (nevyčerpávající) taxonomie.



Obrázek 2.7: Autoenkodéry rozděleny dle charakteristik zpracování kódovací vrstvy

2.13 Využití Autoenkodéru

- Mapování vysokorozměrných dat do 2D pro vizualizaci
- Učení se abstraktních vlastností o vstupních datech bez učitele, pro následné využití v supervizovaných úlohách
- Komprese

3. Variační autoenkodér

(Kingma a Welling, 2013)

3.1 Evidence Lower Bound

3.2 Reparametrizační trik

3.3 Formalizace

3.4 Model umělé neuronové sítě

3.5 Nedostatky a omezení

3.6 Rozšíření a aktuální stav poznání

3.7 Pozorování v latentním prostoru

4. Úlohy pozorování v latentním prostoru

4.1 Generativní modelování obrazových dat

4.2 Rekonstrukce obrazových dat

4.3 Interpolace vět

4.4 Detekce anomálií

4.5 Syntéza tabulárních dat

4.6 Komprese

5. Experimenty s modelem variačního autoenkodéru

5.1 Generativní modelování obrazových dat

5.1.1 Vymezení problémové oblasti

5.1.2 Datová sada a předzpracování

5.1.3 Nastavení experimentu

5.1.4 Návrh modelu

5.1.5 Evaluace

5.1.6 Diskuze

5.2 Interpolace vět

Závěr

The conclusion is a mandatory part of the bachelor's / diploma thesis. It contains a summary of the work and comments on the degree of fulfillment of the goal, which was set in the work, or summarizes the answers to the questions that were asked in the introduction.

The conclusion to the diploma thesis must be more elaborate - this is stated in more detail in the Requirements of the diploma thesis within the Intranet for FIS students.

The conclusion is perceived as a chapter, which begins on a separate page and is called the conclusion. The name Conclusion is not numbered. The text of the conclusion itself is divided into paragraphs.

Bibliografie

BELLMAN, Richard, 1957. *Dynamic Programming*. Princeton University Press. ISBN 9780691079516. Google-Books-ID: wdtoPwAACAAJ.

BENGIO, Yoshua, LAMBLIN, Pascal, POPOVICI, Dan a LAROCHELLE, Hugo, 2006. Greedy Layer-Wise Training of Deep Networks. In: *Advances in Neural Information Processing Systems* [online]. MIT Press. Sv. 19 [cit. 2023-03-06]. Dostupné z: https://proceedings.neurips.cc/paper/2006/hash/5da713a690c067105aeb2fae32403405-Abstract.html.

ERHAN, Dumitru, BENGIO, Yoshua, COURVILLE, Aaron, MANZAGOL, Pierre-Antoine, VINCENT, Pascal a BENGIO, Samy, 2010. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research* [online]. Roč. 11, č. 19, s. 625–660 [cit. 2023-03-06]. ISSN 1533-7928. Dostupné z: http://jmlr.org/papers/v11/erhan10a.html.

GERON, Aurelien, 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O'Reilly Media, Inc. ISBN 9781492032649.

GOODFELLOW, Ian, BENGIO, Yoshua a COURVILLE, Aaron, 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

HEBB, Donald O., 1949. *The organization of behavior: A neuropsychological theory* [Hardcover]. New York: Wiley. ISBN 0-8058-4300-0.

CHARTE, David, CHARTE, Francisco, GARCÍA, Salvador, JESUS, María J. del a HERRERA, Francisco, 2018. A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines. *Information Fusion* [online]. Roč. 44, s. 78–96 [cit. 2023-03-04]. ISSN 15662535. Dostupné z DOI: 10.1016/j.inffus.2017.12.007. arXiv:1801.01586 [cs].

CHOLLET, François, 2017. *Deep Learning with Python*. Manning. ISBN 9781617294433.

KINGMA, Diederik P. a WELLING, Max, 2013. *Auto-Encoding Variational Bayes* [online]. 2013-12 [cit. 2023-03-04]. Tech. zpr. arXiv. Dostupné z DOI: 10.48550/arXiv.1312.6114. arXiv:1312.6114 [cs, stat] version: 1 type: article.

LIU, Huan a MOTODA, Hiroshi (ed.), 1998. *Feature Extraction, Construction and Selection* [online]. Boston, MA: Springer US [cit. 2023-03-05]. ISBN 9781461376224 9781461557258. Dostupné z DOI: 10.1007/978-1-4615-5725-8.

LIU, Weifeng, POKHAREL, P.P. a PRINCIPE, J.C., 2006.
Correntropy: A Localized Similarity Measure. In:
The 2006 IEEE International Joint Conference on Neural Network Proceedings,
s. 4919–4924. Dostupné z DOI: 10.1109/IJCNN.2006.247192.

MINSKY, M. a PAPERT, S., 1969. *Perceptrons*. Cambridge, MA: MIT Press.

MITCHELL, Tom M., 1997. *Machine Learning*. New York: McGraw-Hill.
ISBN 978-0-07-042807-2.

PHILLIPS, Jeff M., 2021. *Mathematical Foundations for Data Analysis*.
Springer International Publishing. ISBN 9783030623401.
Google-Books-ID: AUDYzQEACAAJ.

RANZATO, M.A., HUANG, Fu, BOUREAU, Y-Lan a LECUN, Yann, 2007. Unsupervised
Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In:
s. 1–8. Dostupné z DOI: 10.1109/CVPR.2007.383157.

ROSENBLATT, F., 1957. *The perceptron - A perceiving and recognizing automaton*.
Ithaca, New York, 1957-01. Tech. zpr., 85-460-1. Cornell Aeronautical Laboratory.

RUMELHART, David E. a MCCLELLAND, James L., 1987.
Learning Internal Representations by Error Propagation. In: *Parallel Distributed
Processing: Explorations in the Microstructure of Cognition: Foundations*, s. 318–362.

STAŃCZYK, Urszula a JAIN, Lakhmi C. (ed.), 2015.
Feature Selection for Data and Pattern Recognition [online].
Berlin, Heidelberg: Springer Berlin Heidelberg [cit. 2023-03-05].
Studies in Computational Intelligence. ISBN 9783662456194 9783662456200.
Dostupné z DOI: 10.1007/978-3-662-45620-0.

VALIANT, L. G., 1984. A theory of the learnable. *Communications of the ACM* [online].
Roč. 27, č. 11, s. 1134–1142 [cit. 2023-03-08]. ISSN 0001-0782.
Dostupné z DOI: 10.1145/1968.1972.

WERBOS, P. J., 1974.
Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.
Dis. pr. Harvard University.

WOLPERT, David H., 1996.
The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*.
Roč. 8, č. 7, s. 1341–1390. ISSN 0899-7667.
Dostupné z DOI: 10.1162/neco.1996.8.7.1341.

Přílohy

A. Zdrojové kódy modelů