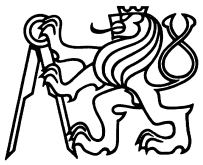
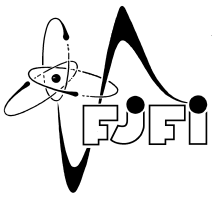


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Generativní modely pro detekci L-H přechodu v plazmatu na tokamaku COMPASS

Generative models for L-H transition detection in COMPASS tokamak plasma

Diplomová práce

Autor: **Bc. Matěj Zorek**

Vedoucí práce: **Ing. Vít Škvára**

Akademický rok: 2019/2020

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Matěj Zorek
Studijní program:	Aplikace přírodních věd
Obor:	Aplikované matematicko-stochastické metody
Název práce (česky):	Generativní modely pro detekci L-H přechodu v plazmatu na tokamaku COMPASS
Název práce (anglicky):	Generative models for L-H transition detection in COMPASS tokamak plasma

Pokyny pro vypracování:

1. Prostudujte dostupnou literaturu zabývající se rekurentními generativními modely (založenými na algoritmu variačního autoencoderu) vhodnými pro použití na časových řadách.
2. Vybrané modely implementujte a porovnejte mezi sebou. Srovnejte latentní prostor natrénovaného autoencoderu s příznaky, extrahovanými z časové řady na základě expertní znalosti.
3. Pro klasifikaci stavů použijte jak latentní reprezentace dat tak i autoencoder navržený přímo pro klasifikaci.
4. Zvažte použití kombinace současných dat a dostupné databáze neolabelovaných dat pro trénink modelů příslušně modifikovaných k tomuto účelu.

Doporučená literatura:

1. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning. MIT Press, 2016.
2. D. P. Kingma, M. Welling, Auto-encoding variational Bayes. arXiv preprint, 2013, arXiv:1312.6114.
3. D. P. Kingma, S. Mohamed, D. J. Rezende, M. Welling, Semi-supervised learning with deep generative models. Advances in neural information processing systems, 2014, 3581-3589.

Jméno a pracoviště vedoucí diplomové práce:
Ing. Vít Škvára
Ústav fyziky plazmatu, AV ČR, Za Slovankou 1782/3, 182 00 Praha 8

Jméno a pracoviště konzultanta:

Datum zadání diplomové práce: 31.10.2019

Datum odevzdání diplomové práce: 4.5.2020

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 15. října 2019

.....
garant oboru

.....
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

.....
děkan

Poděkování:
Chtěl bych zde poděkovat především svému školiteli Ing. Vítu Škvárovi za pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé diplomové práce.

Čestné prohlášení:
Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 29. června 2020 Bc. Matěj Zorek

Název práce:
Generativní modely pro detekci L-H přechodu v plazmatu na tokamaku COMPASS

Autor: Bc. Matěj Zorek

Obor: Aplikované matematicko-stochastické metody

Druh práce: Diplomová práce

Vedoucí práce: Ing. Vít Škvára, Ústav fyziky plazmatu, AV ČR Za Slovankou 1782/3 182 00 Praha 8

Abstrakt: Tato práce se zabývá generativními modely vhodnými ke klasifikaci režimů udržitelnosti plazmatu v tokamaku COMPASS. Mezi použité klasifikační modely se řadí Support Vector Machine, Gradient Tree Boosting a neuronové sítě. Variační autoencodery zde slouží k extrakci nízkodimenzionálních příznaků přímo ze signálů z databáze tokamaku. Nejlepší extraktory a klasifikátory jsou později zkombinovány do semi-supervised variačního autoencoderu a natrénovány pomocí označených i neoznačených dat. Tento postup dosáhl nejlepších výsledků a překonal všechny předchozí modely. Na konci této práce je uvedeno shrnutí a srovnání všech modelů. Implementace probíhala v jazyce Python s využitím knihoven Pytorch a Pyro.

Klíčová slova: hluboké učení, klasifikace, neuronové sítě, semi-supervised variační autoecoder, strojové učení, plazma, tokamak COMPASS, variační autoencoder

Title:
Generative models for L-H transition detection in COMPASS tokamak plasma

Author: Bc. Matěj Zorek

Abstract: This work deals with generative models suitable for classification of plasma sustainability regimes in the COMPASS tokamak. The classification models used include Support Vector Machine, Gradient Tree Boosting and neural networks. Variational autoencoders are used to extract low-dimensional features directly from signals from the tokamak database. The best extractors and classifiers are later combined into a semi-supervised variational autoencoder and trained using labeled and unlabeled data. This procedure achieved the best results and surpassed all previous models. At the end of this work the best models and their results are presented. The implementation is in Python using Pytorch and Pyro libraries.

Key words: classification, COMPASS takamak, deep learning, machine learning, neural networks, semi-supervised variational autoencoder, plasma, variational autoencoder

Obsah

Úvod **9**

1 Popis problematiky **10**
1.1 Plazma a tokamak COMPASS 10
1.2 Existující přístupy ke klasifikaci režimů udržitelnosti plazmatu 11

2 Neuronové sítě **13**
2.1 Základy teorie neuronových sítí 13
2.1.1 Dopředné neuronové sítě 13
2.1.2 Aktivační funkce 15
2.2 Trénování neuronových sítí 16
2.2.1 Stochastic Gradient Descent 17
2.2.2 ADAM 17
2.2.3 Zpětná propagace 18
2.3 Specializované typy vrstev 20
2.3.1 Konvoluční vrstvy 20
2.3.2 Rekurentní vrstvy 22
2.4 Autoencoder 25

3 Variační autoencoder **27**
3.1 Auto-encoding variational Bayes 27
3.2 VAE a neuronové sítě 29
3.2.1 Konvoluční variační autoencoder 31
3.2.2 Rekurentní variační autoencoder 33
3.3 Semi-supervised VAE 34

4 Experimenty a jejich vyhodnocení **38**
4.1 Zdroje a předzpracování dat 38
4.2 Srovnání latentních a ručně vybraných příznaků 40
4.2.1 Nejlepší modely 40
4.2.2 Dosažené výsledky 41
4.2.3 Extrahované příznaky 43
4.3 Klasifikace 45
4.3.1 S původními příznaky 46
4.3.2 Na latentních prostorech 47
4.3.3 S pomocí semi-supervised VAE na H_α 50

Závěr **52**

A Pomocné výpočty **56**

B Doplňující tabulky **58**

Používané značení

x	$\cdots$	Skalární veličina
$\boldsymbol{x}$	$\cdots$	Vektor
x_n	$\cdots$	Prvek na pozici n ve vektoru $\boldsymbol{x}$
$\boldsymbol{x}^{(n)}$	$\cdots$	Vektor $\boldsymbol{x}$ po n-té iteraci
$\boldsymbol{x}^n$	$\cdots$	n-tý vektor resp. vektor v n-té vrstvě neuronové sítě
x_j^n	$\cdots$	j-tá složka n-tého vektoru
$\boldsymbol{X}$	$\cdots$	Matice
$\boldsymbol{X}^{(n)}$	$\cdots$	Matice $\boldsymbol{X}$ po n-té iteraci
$\boldsymbol{X}^n$	$\cdots$	n-tá matice resp. matice v n-té vrstvě neuronové sítě
$\boldsymbol{X}_{i,j}$	$\cdots$	Prvek matice $\boldsymbol{X}$ v i-tém řádku a j-tém sloupci
$\boldsymbol{X}_{i,:}$	$\cdots$	i-tý řádek matice $\boldsymbol{X}$
$\boldsymbol{X}_{:,j}$	$\cdots$	j-tý sloupec matice $\boldsymbol{X}$
$\boldsymbol{X}^T$	$\cdots$	Transpozice matice $\boldsymbol{X}$
$\boldsymbol{I}$	$\cdots$	Diagonální matice
$p(x)$	$\cdots$	Pravděpodobnostní rozdělení spojité veličiny x
$x \sim p(x)$	$\cdots$	Náhodná veličina s rozdělením $p(x)$
$p(x\|z)$	$\cdots$	Podmíněné rozdělení veličiny x za daného z
$p_\theta(x) = p(x\|\theta)$	$\cdots$	Rozdělení veličiny x s parametrem θ (rozdělení veličiny x podmíněné parametrem θ)
$\mathbb{E}_{p(x)}$	$\cdots$	Střední hodnota vzhledem k hustotě pravděpodobnosti $p(x)$
$\mathbb{V}ar_{p(x)}$	$\cdots$	Rozptyl vzhledem k hustotě pravděpodobnosti $p(x)$
$\mathcal{N}(x\|\mu, \sigma^2)$	$\cdots$	Normální rozdělení pro x se střední hodnotou μ a rozptylem σ^2
$\mathcal{N}(\boldsymbol{x}\|\boldsymbol{\mu}, \boldsymbol{\Sigma})$	$\cdots$	Vícerozměrné normální rozdělení pro $\boldsymbol{x}$ se vektorem středních hodnot $\boldsymbol{\mu}$ a kovarianční maticí $\boldsymbol{\Sigma}$
$Q_X(p)$	$\cdots$	p-tý empirický kvantil

Úvod

Cílem této práce je využít předností generativních modelů a neuronových sítí při klasifikaci režimů udržitelnosti plazmatu v tokamaku. Generativní modely slouží k odhadování pravděpodobnostního rozdělení popisujícího proces vzniku cílových dat. Jejich primárním účelem je taková reprezentace tohoto rozdělení, která umožňuje generovat nové vzorky. Díky spojení s neuronovými sítěmi jsou pak současné typy generativních modelů schopny reprezentovat vysokodimenzionální rozdělení patřící například audio signálům či obrazovým datům o vysokém rozlišení.

My budeme generativní modely, konkrétně ty založené na modelu variační autoencoder (VAE), využívat k redukci dimenzionality a extrakci příznaků z dat v databázi tokamaku COMPASS, přičemž využijeme i neoznačená data, která jsou normálně pro klasifikační úlohy nevhodná.

V první kapitole je stručně představena problematika tokamaků a udržení termojaderné fúzní reakce. Následuje shrnutí existujících přístupů k řešení tohoto problému.

Druhá kapitola obsahuje teorii vztahující se k neuronovým sítím. Je zde postupně popsáno z čeho se tyto sítě skládají, co jsou aktivační funkce, jakým způsobem se tyto sítě trénují a jaké typy neuronových sítí existují.

Třetí kapitola je zaměřena na modely typu variační autoencoder. Nejprve je uvedeno obecné odvození jeho spodní meze marginální věrohodnosti, po kterém následuje propojení variačních autoencoderů a neuronových sítí. Dále jsou představeny tři typy VAE, které jsou později použity v této práci. Konec této kapitoly je věnován semi-supervised variačnímu autoencoderu, jehož použití umožňuje rozšíření trénovací množiny dat o neoznačená pozorování.

Na začátku čtvrté kapitoly je popsán proces předzpracování dat z databáze a způsob výběru hyperparametrů modelů. Následuje srovnání ručně vytvořených (expertních) příznaků a příznaků získaných automatickým tréninkem autoencoderů. Kapitola je zakončena porovnáním výsledků jednotlivých klasifikátorů.

Kapitola 1

Popis problematiky

1.1 Plazma a tokamak COMPASS

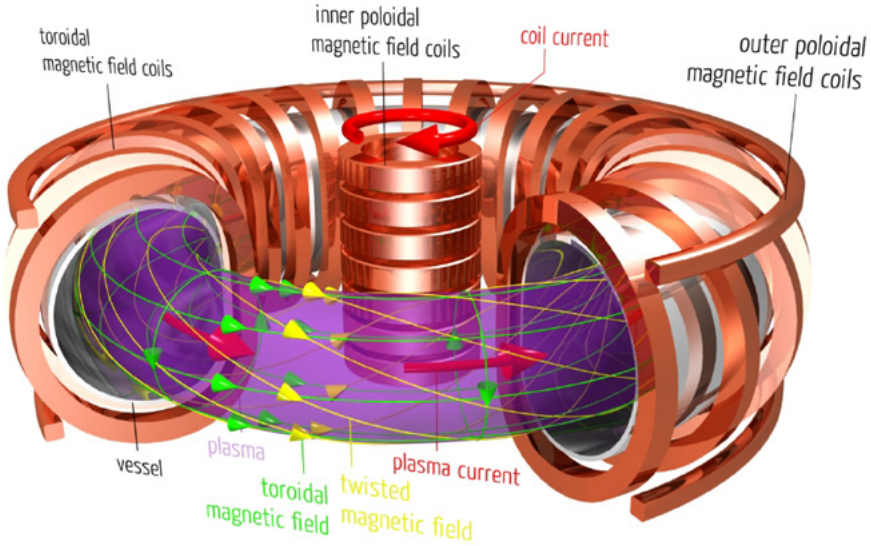
Plazma je jedním ze čtyř základních skupenství. Tvoří ho shluk ionizovaných částic a elektronů. Pravě přítomností a pohybem volných iontů a elektronů vzniká uvnitř plazmatu elektromagnetické pole. Na zemském povrchu se vyskytuje pouze zřídka, nejčastěji ve formě blesků, avšak ve vesmíru tvoří 99% veškeré pozorovatelné hmoty. Skládají se z něj hvězdy včetně Slunce.

V laboratoři vzniká zahříváním a ionizováním malého množství plynu pomocí elektrického proudu nebo radiových vln, které dodávají energii volným elektronům. Následnými srážkami těchto elektronů s atomy se uvolňují další elektrony a tento kaskádový proces postupuje, dokud není plyn dostatečně ionizovaný [1].

V závislosti na teplotě a hustotě dělíme plazmata na částečně ionizované, což jsou například blesky nebo neonové světlo a plně ionizované, které se nachází ve Slunci nebo právě v tokamacích.

Už několik desetiletí se vyvíjejí metody jak plazma, které má v centru teplotu okolo 300 milionů stupňů Celsia, udržet v pozemských podmínkách. Pokud by se podařilo fúzi efektivně a trvale udržet, mohlo by být přebytečné teplo převedeno na elektřinu stejně jako v tepelných elektrárnách. Díky tomu bychom získali téměř nevyčerpatelný zdroj čisté energie, který nezatěžuje životní prostředí. Tokamaky jsou v současné době nejpokročilejším experimentálním zařízením pro studium tohoto problému.

Na rozdíl od klasických jaderných reaktorů, které štěpí těžká jádra uranu ^{235}U na lehčí jádra, v tokamacích dochází ke vzniku těžších jader slučováním těch lehčích. Při termojaderné reakci se často jako zdroj lehkých jader používá deuterium a tricium. Výsledkem této reakce je pak hélium, neutron a uvolněná vazebná energie. Problém nastává ve chvíli, kdy chceme termojadernou reakci udržet. Abychom dosáhli dostatečné doby trvání, je zapotřebí velmi vysokých teplot. Protože při kontaktu plazmatu se stěnou dochází k nechtěnému a prudkému ochlazování, musíme částice plazmatu držet uprostřed toroidní komory, k čemuž slouží silné magnetické pole produkované cívkami (Obr. 1.1).



Obr. 1.1: Průřez komorou tokamaku s vizualizovanými směry magnetických polí [2].

Tokamaků funguje na světě několik, v Praze jsou dokonce dva. Jedním z nich je GOLEM, který je nejmenším a nejstarším fungujícím tokamakem na světě. Druhý je tokamak COMPASS.Ten je umístěn v Praze Ládví na Ústavu fyziky plazmatu Akademie věd České republiky již od roku 2004 [3]. Původně byl umístěn a provozován ve Velké Británii pod UKAEA (UK Atomic Energy Autority) do roku 2002, kdy byl nahrazen tokamakem MAST. Díky svým rozměrům je řazen do kategorie menších tokamaků. I přes svou malou velikost umožňuje dosáhnout režimu vysokého udržení plazmatu, nebo-li H-módu (High-confinement mode) a zároveň odpovídá desetině velikosti tokamaku ITER, v současnosti budovanému ve Francii. Právě díky těmto dvěma vlastnostem je nyní využíván ke studiu specifických jevů, které jsou třeba k hlubšímu pochopení chování plazmatu a jeho následného udržení.

1.2 Existující přístupy ke klasifikaci režimů udržitelnosti plazmatu

V tokamaku COMPASS se plazma nejčastěji nachází v jednom ze dvou základních stavů. Prvním z nich je mód nízkého udržení zvaný L-mód (Low-confinement mode). V tomto stavu se plazma nachází po dosažení standardní magnetické konfigurace na začátku nebo po skončení H-módu. Pokud se plazma nachází v L-módu, je velice náročné udržet termojadernou reakci na delší dobu.

Druhým stavem je H-mód (High-confinement mode), který byl objeven v roce 1982 německým vědcem Fritzem Wagnerem [4]. V tomto stavu je možné lépe kontrolovat chování plazmatu a především jej udržet delší dobu, proto je H-mód také standardním referenčním režimem v současnosti budovaného tokamaku ITER.

Během H-módů se může někdy navíc objevovat třetí stav zvaný ELM (edge-localized mode). ELM označuje dočasné periodicky se opakující nestability, které se vyskytují na okraji plazmatu a narušují průběh H-módu [5].

Znalost toho, v jakém módu se plazma v tokamaku během experimentu nachází, je klíčová. A to jak ta okamžitá, která umožňuje přesnější zpětnovazebné řízení během experimentu, tak i zpětná, tj. získaná ze zaznamenaných experimentálních dat po proběhnutí experimentu. Její korelací s ostatními pozorovanými jevy je možné odhalit další skryté zákonitosti L-H přechodu.

Klasifikací H-módu a L-módu se v současnosti zabývá např. Giuseppe A. Rattá [6], který s použitím patnácti signálů (příznaků) dosáhl přesnosti 98.60% na datech z několika různých tokamaků. Jako klasifikátor využívá Support Vector Machine s jádrem tvořeným radiálně bazickými funkcemi. Volbu hyperparametrů stejně jako výběr vhodných příznaků zde zastřešují evoluční algoritmy.

Jeden z nejnovějších přístupu naopak využívá konvoluční rekurentní neuronové sítě [5]. Původní dva stavy jsou zde rozšířeny o tzv. nerozhodné stavy, kdy se nedá s jistotou říci, zda se jedná o H-mód nebo L-mód. ELMy jsou zde binárně klasifikovány zvlášť, jako doplňující události mimo hlavní klasifikaci. Přesnost modelu je pro tyto tři stavy rovna 96.0%.

V naší předchozí práci [7] jsme klasifikovali H-módy, L-módy a ELMy, jakožto tři separátní stavy. Maximální dosažená přesnost byla rovna 86.78% a jako klasifikátor nám sloužil skrytý Markovův model využívající na míru vytvořené příznaky získávané pouze ze záření H_α naměřeném na experimentech na tokamaku COMPASS.

Záření H_α je specifická červená spektrální čára s vlnovou délkou $656.28nm$, jež vyzařuje vodík obsažený v plazmatu. Amplituda tohoto signálu je závislá na počtu částic, které se ionizují během interakcí plazmatu a stěny komory [4]. Na základě toho lze rozhodnout, zda se plazma nachází v H-módu nebo L-módu, protože v H-módu jsou částice pod větší kontrolou a nedochází tak k tolika interakcím jako v L-módu.

Výše zmíněný přístup jsme později doplnili o strukturovanější výběr příznaků a další klasifikační modely, jimiž jsou Support Vector Machine (SVM) a Gradient Tree Boosting (GB). Původní dataset byl rozšířen o signály H_α pocházející z dalších výstřelů na tokamaku a spolu s tím byl přidán i další klasifikační stav rozdělením ELMu na začátek a konec. S výběrem patnácti nejlepších příznaků dosáhl GB přesnosti 87.90%.

V této práci na tento postup navážeme s tím rozdílem, že se pokusíme nahradit na míru (ručně) vytvořené příznaky novými, které získáme z latentního prostoru variačního autoencoderu. Doufáme, že nelinearita neuronových sítí použitých k tomuto úkolu umožní odkrýt složitější závislosti než v případě expertně postavených signálů a povede k přesnější klasifikaci na datech stejné dimenze. Na příznacích budeme trénovat různé typy klasifikátorů vhodných pro tento účel, včetně neuronových sítí. Využívat budeme opět pouze naměřené záření H_α z databáze tokamaku COMPASS. Později se pokusíme využít i neoznačená pozorování, kterých je v databázi mnohonásobně více, než těch označených a klasifikátor postavený přímo na modelu semi–supervised VAE.

Kapitola 2

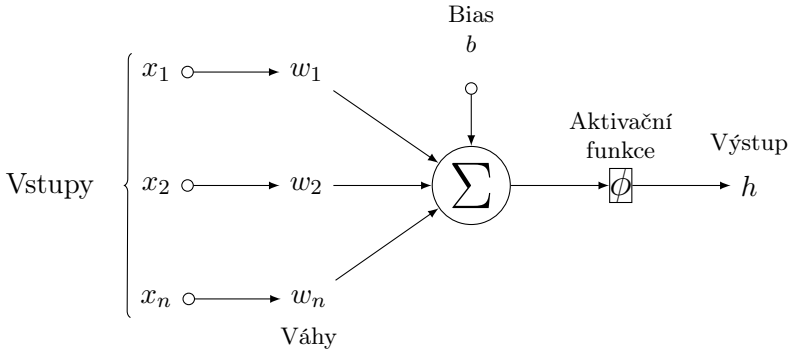
Neuronové sítě

2.1 Základy teorie neuronových sítí

Umělá neuronová síť, známá také pod názvem vícevrstvý perceptron, je matematický model schopný aproximovat složité funkce skrze skládání jednoduchých funkcí dohromady. Jak už název naznačuje, inspirací pro neuronové sítě bylo fungování lidského mozku. Cílem však není perfektně modelovat mozek, ale využít toho, co o něm víme ke zlepšení aproximačních schopností modelu a jeho statistickému zobecnění [8].

2.1.1 Dopředné neuronové sítě

Základním stavebním kamenem těchto sítí je neuron (Obr. 2.1).



Obr. 2.1: Morfologie neuronu.

V jádru je neuron pouze lineární transformace vstupu, na kterou je aplikována aktivační funkce $\phi(x)$, která je nejčastěji nelineární. Matematicky jej můžeme zapsat jako

$$h = \phi(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \phi\Big(\sum_i w_i x_i + b\Big), \tag{2.1}$$

kde h je výstup neuronu, $\boldsymbol{x}$ je vstupní vektor, $\boldsymbol{w}$ jsou váhy jednotlivých prvků vstupního vektoru, b je posun (tzv. bias) a nelineární funkce ϕ se nazývá aktivační funkce. Neuron, nebo-li perceptron je sám o sobě také nejjednodušší verzí neuronové sítě a jeho historie sahá až do padesátých let dvacátého století, kdy byl poprvé použit [9].

Obecně nemusí být výstup h a váhy $\boldsymbol{w}$ pouze skalár a vektor, nýbrž vektor a matice. V případě, že $\boldsymbol{W}$ je matice a $\boldsymbol{h}$ a $\boldsymbol{b}$ jsou vektory, hovoříme místo jednoho neuronu o vrstvě neuronů

$$\boldsymbol{h} = \phi(\boldsymbol{x}|\boldsymbol{W}, \boldsymbol{b}) = \phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{2.2}$$

která bývá v literatuře označována také jako dense vrstva.

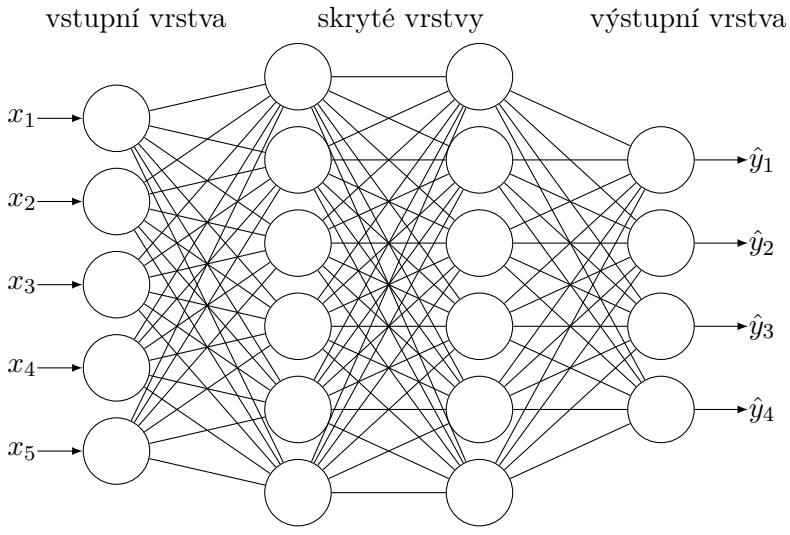
Vícevrstvá neuronová síť $f(\boldsymbol{x})$ vzniká napojováním těchto vrstev za sebe tak, že výstup jedné vrstvy je vstupem druhé (Obr. 2.2)

$$\begin{aligned} f(\boldsymbol{x}|\boldsymbol{\theta}) &= \phi^{(3)}(\phi^{(2)}(\phi^{(1)}(\boldsymbol{x}))) \\ &= \phi^{(3)}\Big(\boldsymbol{W}^3\phi^{(2)}\Big(\boldsymbol{W}^2\phi^{(1)}\Big(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1\Big) + \boldsymbol{b}^2\Big) + \boldsymbol{b}^3\Big), \end{aligned} \tag{2.3}$$

kde $\boldsymbol{\theta}$ je soubor všech vah a posunů v síti f, $\phi^{(1)}$ je první vrstva, $\phi^{(2)}$ je druhá vrstva atd. Poslední vrstvě, v tomto případě třetí $\phi^{(3)}$, se říká výstupní vrstva. Celkový počet vrstev udává hloubku modelu. Neuronové sítě, které mají hloubku větší než tři, jsou pak označovány jako hluboké neuronové sítě. Odtud také plyne označení "hluboké učení" (deep learning) [8].

Díky univerzálnímu aproximačnímu teorému [10] víme, že jednovrstvý perceptron s omezenou, nekonstantní a spojitou aktivační funkcí a konečným počtem neuronů (tj. omezenou šířkou) dokáže aproximovat funkci $f : \mathbb{R}^n \to \mathbb{R}$ na kompaktním podintervalu $\mathbb{R}^n$ s libovolnou přesností. Nedává ale žádné garance pro to, zda a jak je možné parametry takového perceptronu získat. Později [11] bylo toto tvrzení rozšířeno pro vícevrstvé sítě a jiné než aktivační sigmoidální funkce (například ReLU, viz další text). Teoreticky jsou tedy neuronové sítě schopné být univerzálními funkčními aproximátory, prakticky ale jejich použití limitují dostupné techniky tréninku a kvalita a kvantita trénovacích dat.

Během tréninku se snažíme přimět neuronovou síť, aby její výstup co nejvíce odpovídal funkci $f^*(\boldsymbol{x})$, kterou chceme aproximovat s využitím tréninkových vzorků $(\mathbb{X}, \mathbb{Y}) = \{\boldsymbol{x}^n, \boldsymbol{y}^n\}_{n=0}^N$ splňujících $\boldsymbol{y}^n = f^*(\boldsymbol{x}^n)$. Tyto tréninkové vzorky přímo ovlivňují pouze výstupní vrstvu produkující odhad $\hat{\boldsymbol{y}}$. O tom, jak využít ostatní vrstvy k dosažení kýženého výstupu rozhoduje pouze učící algoritmus. Jelikož trénovací data přímo neovlivňují tyto vrstvy a neurčují jak má vypadat jejich výstup, hovoříme o nich jako o skrytých vrstvách (hidden layers).



Obr. 2.2: Vícevrstvá neuronová síť.

2.1.2 Aktivační funkce

Aktivační funkce $\phi(x)$ je funkce, která transformuje lineární kombinaci na výstupu neuronu, viz (2.1). V případě neuronových sítí plní stejný účel jako jádrové funkce u metody podpůrných vektorů (SVM). Díky nim jsou neuronové sítě schopné aproximovat i velmi komplexní funkce. Podmínkou, kterou musí všechny aktivační funkce splňovat, je diferencovatelnost. Tato vlastnost je nezbytná k trénování sítě.

Nejčastěji používanou aktivační funkcí je pravděpodobně rektifikovaná lineární funkce, zvaná také jen jako ReLU

$$\phi(x) = \max\{0, x\}. \tag{2.4}$$

Tato aktivační funkce se velmi snadno optimalizuje díky blízké podobnosti s obyčejnou lineární funkcí $\phi(x) = x$ viz Obr. 2.3 [12]. Mezi její hlavní přednosti dále patří výpočetní jednoduchost a rychlá konvergence. Na rozdíl od sigmoidální funkce a tanh netrpí efektem mizejících gradientů, tzn. že pro velké hodnoty x není gradient blízký nule [13].

Další její dobrou vlastností a současně i nevýhodou je schopnost vrátit skutečnou nulu. Například u konvoluční vrstvy nula naznačuje nepřítomnost naučené reprezentace ve vstupním vektoru (resp. matici), což je žádoucí například pro rozpoznání objektů na obrázcích [8]. Neužitečné se to stává, až při optimalizaci parametrů, kde takto neaktivované neurony mají nulový gradient a neprobíhá proto optimalizace jejich vah.

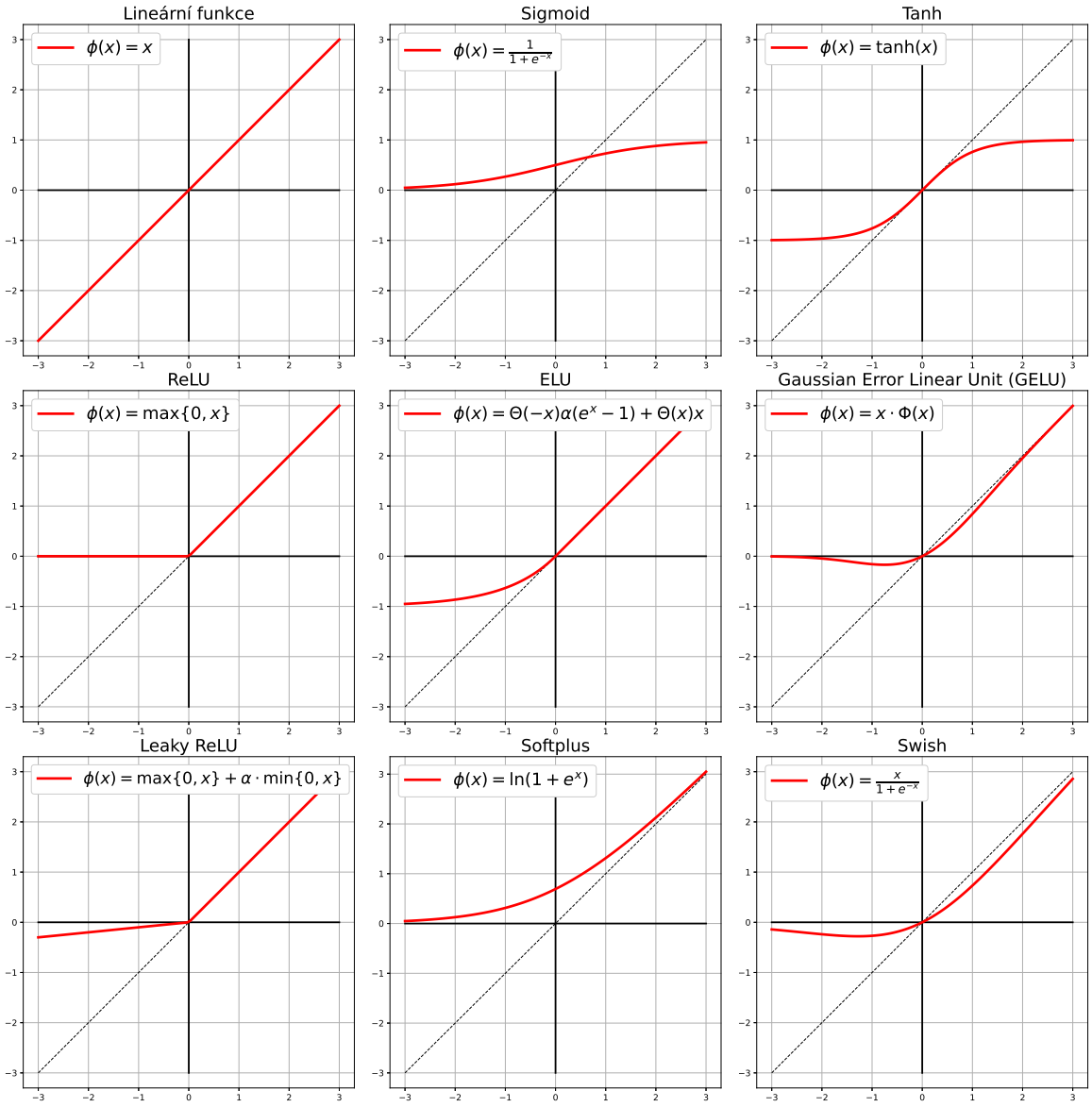
K odstranění tohoto problému byly později vytvořeny její modifikace, mezi které patří třeba Leaky ReLU [14]

$$\phi(x) = \max\{0, x\} + \alpha \cdot \min\{0, x\}, \tag{2.5}$$

kde α je malé číslo v řádu setin, a ELU [15]

$$\phi(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{pro} \quad x \leq 0 \\ x & \text{pro} \quad x > 0, \end{cases} \tag{2.6}$$

která pro záporná čísla místo lineárního využívá exponenciálního poklesu k $-\alpha$.



Obr. 2.3: Často používané aktivační funkce, kde pro Leaky ReLU je $\alpha = 0.1$ a pro ELU je $\alpha = 1$.

2.2 Trénování neuronových sítí

Pro správné natrénování neuronové sítě je třeba nejprve zvolit vhodnou ztrátovou funkci $L(\boldsymbol{\theta})$. Může to být libovolná diferencovatelná funkce zobrazující do $\mathbb{R}$, která udává míru toho, jak se výstup modelu liší od cílových dat. Zadefinujme si tréninkový soubor dat a labelů jako $(\mathbb{X}, \mathbb{Y}) = \big((\boldsymbol{x}^1, \cdots, \boldsymbol{x}^n), (\boldsymbol{y}^1, \cdots, \boldsymbol{y}^n)\big)$. Trénink neuronové sítě pak lze formalizovat jako optimalizační úlohu

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} L(f(\mathbb{X}, \boldsymbol{\theta}), \mathbb{Y}), \tag{2.7}$$

tj. minimalizace ztrátové funkce vůči parametrům $\boldsymbol{\theta}$ při dostupných datech $(\mathbb{X}, \mathbb{Y})$.

Výběr ztrátové funkce závisí především na typu úlohy. Mezi běžně používané funkce jsou

např. Mean Squared Error (MSE)

$$L(\boldsymbol{\theta}) = L(f(\mathbb{X}, \boldsymbol{\theta}), \mathbb{Y}) = \frac{1}{N} \sum_{n=0}^{N} \|f(\boldsymbol{x}^n, \boldsymbol{\theta}) - \boldsymbol{y}^n\|^2 \tag{2.8}$$

pro regresi nebo křížová entropie

$$L(\boldsymbol{\theta}) = L(f(\mathbb{X}, \boldsymbol{\theta}), \mathbb{Y}) = -\sum_{n=0}^{N} \sum_{k=1}^{K} \boldsymbol{y}_k^n \log f_k(\boldsymbol{x}^n, \boldsymbol{\theta}) \tag{2.9}$$

pro klasifikaci do K tříd.

Během tréninku se optimalizační algoritmus snaží ztrátovou funkci minimalizovat vzhledem k vahám $\boldsymbol{\theta}$ a tím lépe namapovat $f(\mathbb{X})$ na $\mathbb{Y}$. Cílem je najít váhy $\boldsymbol{\theta}$ tak, aby hodnota ztrátové funkce $L(\boldsymbol{\theta})$ byla co nejmenší, v ideálním případě nalézt globální minimum. Nicméně ztrátová funkce je díky aktivačním funkcím silně nelineárně závislá na váhách, a proto může mít mnoho lokálních minim, do kterých může optimalizační algoritmus dokonvergovat [16].

Geometricky tak ztrátová funkce vytváří nadplochu v prostoru vah [16]. Díky nelinearitám modelu neuronové sítě nelze řešit úlohu (2.7) přímo jako systém uzavřených rovnic předepisujících hodnoty θ. Místo toho se používají iterativní algoritmy založené na principu gradientního sestupu. Změňme v kroku t aktuální hodnotu parametrů $\boldsymbol{\theta}^{(t)}$ o $\delta\boldsymbol{\theta}$. To vyvolá změnu ztrátové funkce o $\delta L(\boldsymbol{\theta}^{(t)}) \approx (\delta\boldsymbol{\theta})^T \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})$, kde vektor $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})$ míří směrem největšího růstu ztrátové funkce v bodě $\boldsymbol{\theta}^{(t)}$. Díky hladkosti funkce $L(\boldsymbol{\theta})$ vzhledem k $\boldsymbol{\theta}$ se minimum nachází v takovém bodě prostoru, kde gradient zcela vymizí tzn. $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) = 0$. Optimalizační algoritmy pro neuronové sítě hledají toto minimum pomocí drobných kroků proti směru gradientu ztrátové funkce. Jejich typickými zástupci jsou např. SGD a ADAM.

2.2.1 Stochastic Gradient Descent

Nejjednodušší metodou pro optimalizaci neuronových sítí je metoda největšího spádu známá jako Gradient Descent [17]. Tato metoda upravuje váhy pomocí malých kroků ve směru největšího záporného gradientu ztrátové funkce podle vzorce

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)}), \tag{2.10}$$

kde se s každým dalším krokem t posouvá blíže k minimu. Parametr $\eta > 0$ je známý jako míra učení (resp. learning rate) a udává délku jednoho kroku algoritmu.

V tomto případě je ztrátová funkce definovaná vůči celém tréninkovému datasetu. V každé iteraci je pro získání $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})$ tedy potřeba vyhodnotit ztrátu na úplně celém datasetu. Takto definovaný algoritmus je neefektivní a časově velmi náročný, proto se později dočkal vylepšení.

Bylo dokázáno, že místo vyhodnocení ztráty na všech datech stačí jen jejich náhodná podmnožina - minibatch [17]. Podle náhodného výběru této podmnožiny dostal algoritmus název Stochastic Gradient Descent (SGD). Toto vede nejenom ke značnému zrychlení, ale navíc se zvyšuje šance na případný únik z lokálních extrémů a tím pádem také nalezení globálního minima [16]. Obecně ale použití SGD nalezení globálního minima negarantuje.

2.2.2 ADAM

Adam neboli adaptivní momentový optimalizační algoritmus je v současnosti nejpoužívanější algoritmus pro trénování neuronových sítí. Oproti obyčejnému SGD využívá k úpravě vah klouzavý

průměr a rozptyl gradientu. Nepodléhá tak takzvanému "ziggzagingu", tj. efektu toho, že gradient kvůli používání minibatchů v po sobě následujících krocích mění výrazně svůj směr. Při použití klouzavých momentů je pohyb algoritmu v prostoru vah plynulejší. Díky tomu je schopen se častěji vymanit z lokálních extrémů a rychleji konverguje ke globálnímu minimu. Algoritmus tak kombinuje výhody dvou populárních metod, mezi které patří AdaGrad a RMSProp [18].

Mezi jeho další výhody patří adaptivní výpočet míry učení a v neposlední řadě mu nedělají problém řídké gradienty vznikající, např. aplikací rektifikovaných aktivačních funkcí na záporné hodnoty. Úprava vah pak probíhá podle vzorců

$$\boldsymbol{m}^{(t)} = \beta_1 \cdot \boldsymbol{m}^{(t-1)} + (1-\beta_1) \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)}) \tag{2.11}$$

$$\boldsymbol{v}^{(t)} = \beta_2 \cdot \boldsymbol{v}^{(t-1)} + (1-\beta_2) \cdot \left(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})\right)^2 \tag{2.12}$$

$$\hat{\boldsymbol{m}}^{(t)} = \frac{\boldsymbol{m}^{(t)}}{(1-\beta_1^t)} \tag{2.13}$$

$$\hat{\boldsymbol{v}}^{(t)} = \frac{\boldsymbol{v}^{(t)}}{(1-\beta_2^t)} \tag{2.14}$$

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \eta \frac{\hat{\boldsymbol{m}}^{(t)}}{\sqrt{\hat{\boldsymbol{v}}^{(t)}} + \epsilon}, \tag{2.15}$$

$\boldsymbol{m}^{(t)}, \boldsymbol{v}^{(t)}, \hat{\boldsymbol{m}}^{(t)}$ a $\hat{\boldsymbol{v}}^{(t)}$ jsou klouzavý průměr a rozptyl gradientu a jejich normalizované hodnoty, η je míra učení, $\epsilon = 10^{-8}$ brání případnému dělení nulou a parametry $\beta_1, \beta_2 \in [0, 1)$ slouží ke kontrole exponenciálního tlumení klouzavých průměrů pro oba momenty. Stejně jako u SGD probíhá trénink v minibatchích, tzn. že není třeba používat všechna data.

2.2.3 Zpětná propagace

V předchozí sekci jsme si přiblížili jakým způsobem jsou trénovány parametry neuronových sítí pomocí optimalizačních algoritmů. Abychom mohly tyto algoritmy použít potřebujeme nejprve vypočítat gradienty a k tomu slouží zpětná propagace. Zpětná propagace (Backpropagation) je algoritmus sloužící k efektivnímu výpočtu gradientů v neuronové síti pomocí řetězového pravidla [8].

Řetězové pravidlo se používá k výpočtu derivací složených funkcí. Nechť $f : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$ a $y = g(x)$ pak derivaci funkce f podle x lze spočítat jako

$$\frac{df}{dx} = \frac{df}{dy}\frac{dg}{dx}. \tag{2.16}$$

Pro obecné $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ a $\boldsymbol{g} : \mathbb{R}^m \to \mathbb{R}^n$ platí

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{y}) = \left(\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} f(\boldsymbol{y}), \tag{2.17}$$

kde

$$\left(\frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right) = \begin{pmatrix} \nabla_{\boldsymbol{x}} \boldsymbol{g}_1(\boldsymbol{x}) \\ \nabla_{\boldsymbol{x}} \boldsymbol{g}_2(\boldsymbol{x}) \\ \vdots \\ \nabla_{\boldsymbol{x}} \boldsymbol{g}_n(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} \frac{\partial \boldsymbol{g}_1(\boldsymbol{x})}{\partial x_1} & \frac{\partial \boldsymbol{g}_1(\boldsymbol{x})}{\partial x_2} & \dots & \frac{\partial \boldsymbol{g}_1(\boldsymbol{x})}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{g}_n(\boldsymbol{x})}{\partial x_1} & \frac{\partial \boldsymbol{g}_n(\boldsymbol{x})}{\partial x_2} & \dots & \frac{\partial \boldsymbol{g}_n(\boldsymbol{x})}{\partial x_m} \end{pmatrix}$$

je Jakobiho matice.

Kdybychom však počítali derivace pro všechny váhy v modelu jen podle řetězového pravidla, jednalo by se o velmi neefektivní postup, protože bychom museli některé derivace počítat zbytečně mnohokrát znovu, jelikož se objevují při derivacích ve všech vrstvách nad neuronem, ke kterému naleží.

Zpětná propagace tyto derivace počítá mnohem efektivněji pomocí následujících rekurentních vztahů [19]. Naším cílem je získat veličiny $\frac{\partial L(\boldsymbol{\theta})}{\partial W_{ij}^k}, \frac{\partial L(\boldsymbol{\theta})}{\partial b_i^k}$, které můžeme použít pro optimalizaci jednotlivých parametrů sítě. K tomu si nejprve vyjádříme obecný vztah pro chybu na i-tém neuronu v k-té vrstvě, δ_i^k. Nejprve uvažujme neuronovou síť hloubky K a nechť h_i^k je zaktivovaný výstup tohoto neuronu

$$h_i^k = \phi(z_i^k) = \phi\left(\sum_j W_{ij}^k h_j^{k-1} + b_i^k\right) = \phi\left(\boldsymbol{W}_{i,:}^k \boldsymbol{h}^{k-1} + b_i^k\right). \tag{2.18}$$

Pak chybu na výstupní K-té vrstvě označíme jako $\boldsymbol{\delta}^K$

$$\boldsymbol{\delta}^K = \nabla_{\boldsymbol{z}^K} L(\boldsymbol{\theta}) = \left(\frac{\partial L}{\partial h_1^K}, \frac{\partial L}{\partial h_2^K}, \cdots\right) \odot \left(\frac{\partial \phi(\boldsymbol{z}_1^K)}{\partial z_1^K}, \frac{\partial \phi(\boldsymbol{z}_2^K)}{\partial z_2^K}, \cdots\right) = \nabla_{\boldsymbol{h}^K} L(\boldsymbol{\theta}) \odot \frac{\partial \phi(\boldsymbol{z}^K)}{\partial \boldsymbol{z}^K}, \tag{2.19}$$

kde $\odot$ je násobení po prvcích.

Chyba na každé další vrstvě lze pak vypočítat podle rekurentního vztahu pro $1 \leq k < K$

$$\begin{aligned}
\boldsymbol{\delta}^k &= \nabla_{\boldsymbol{z}^k} L(\boldsymbol{\theta}) \\
&= \nabla_{\boldsymbol{h}^k} L(\boldsymbol{\theta}) \odot \frac{\partial \phi(\boldsymbol{z}^k)}{\partial \boldsymbol{z}^k} \\
&= \left(\left(\frac{\partial \boldsymbol{z}^{k+1}}{\partial \boldsymbol{h}^k}\right)^T \nabla_{\boldsymbol{z}^{k+1}} L(\boldsymbol{\theta})\right) \odot \frac{\partial \phi(\boldsymbol{z}^k)}{\partial \boldsymbol{z}^k} \\
&= \left(\left(\boldsymbol{W}^{k+1}\right)^T \boldsymbol{\delta}^{k+1}\right) \odot \frac{\partial \phi(\boldsymbol{z}^k)}{\partial \boldsymbol{z}^k}.
\end{aligned} \tag{2.20}$$

Jakobiho matice ztrátové funkce vzhledem k vahám v k-té vrstvě je pak

$$\frac{\partial L(\boldsymbol{\theta})}{W_{ij}^k} = h_j^{k-1} \delta_i^k \tag{2.21}$$

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{W}^k} = \boldsymbol{h}^{k-1} \left(\boldsymbol{\delta}^k\right)^T \tag{2.22}$$

a její gradient vzhledem k posunům je

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{b}^k} = \nabla_{\boldsymbol{b}^k} L(\boldsymbol{\theta}) = \boldsymbol{\delta}^k. \tag{2.23}$$

Jak již bylo řečeno výše, tento způsob výpočtu gradientů přináší značné výpočetní usnadnění, jelikož se gradienty v každé vrstvě spočítají jen jednou, poté se mohou uložit a použít znovu při výpočtu gradientů vyšších vrstev.

2.3 Specializované typy vrstev

2.3.1 Konvoluční vrstvy

Konvoluční vrstva je alternativou ke standardní neuronové vrstvě (2.2) vhodné pro zpracování pravidelných struktur, jako jsou časové řady a obrazové vstupy [8]. Odlišností od plně propojených vrstev, které byly představeny v předchozí části, je použití konvoluce namísto obyčejného maticového násobení. Operace konvoluce je pro dvě reálné spojité funkce $f : \mathbb{R} \to \mathbb{R}$ a $g : \mathbb{R} \to \mathbb{R}$ definovaná jako

$$(f * g)(x) = \int_{\mathbb{R}} f(y)g(x - y)dy \tag{2.24}$$

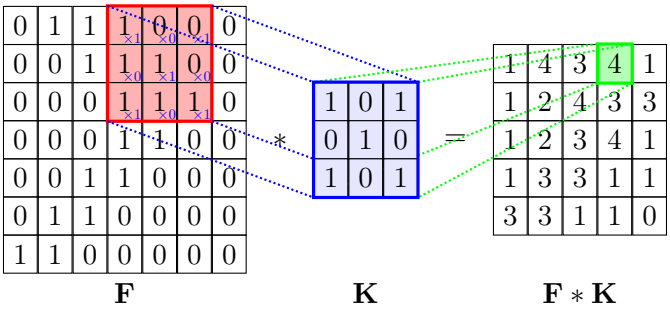
a jedná se o vážené průměrování funkce $f(x)$ funkcí $g(x)$. V diskrétním případě pak nabývá tvaru

$$(f * g)(t) = \sum_{i=-\infty}^{\infty} f(i)g(t - i). \tag{2.25}$$

V praxi se spíš častěji potkáte s její dvourozměrnou verzí

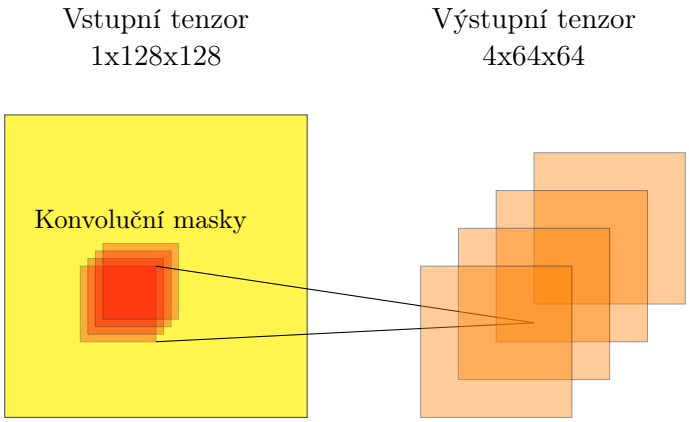
$$(\boldsymbol{F} * \boldsymbol{K})_{i,j} = \sum_m \sum_n \boldsymbol{F}_{i-m,j-n}\boldsymbol{K}_{m,n}, \tag{2.26}$$

kde $\boldsymbol{F}$ je vstupní matice (např. obrázek) a matici $\boldsymbol{K}$ označujeme jako konvoluční masku nebo jádro, které má nejčastěji menší rozměry než $\boldsymbol{F}$ (Obr. 2.4.



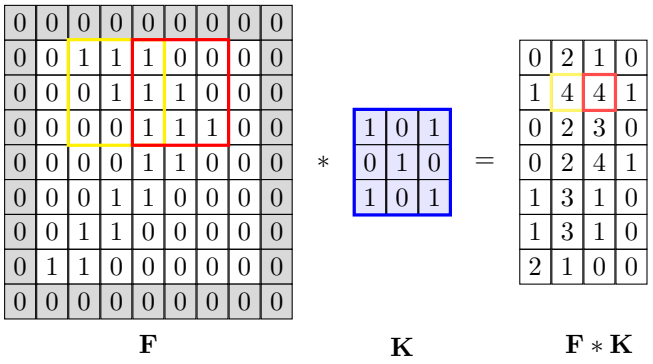
Obr. 2.4: Vizualizace 2D konvoluce s posunem $s = (1, 1)$ tzn. (*1 horizontálně, 1 vertikálně*), bez přesahu $p = 0$ s velikostí masky $f = 3$.

Konvoluční vrstva se skládá ze souboru konvolučních masek $\{\boldsymbol{K}_n\}^N$, jež jsou postupně aplikovány na vstupní matici $\boldsymbol{F}$. Výstupem z ní je vícerozměrný tenzor, jehož hloubka je rovna počtu masek N.



Obr. 2.5: Ilustrace konvoluční vrstvy, kde pomocí 4 konvolučních masek vznikl z tenzoru hloubky jedna jiný tenzor s hloubkou čtyři.

Konvoluční vrstva má kromě trénovatelných parametrů (masek) i několik netrénovatelných, jako jsou třeba délka posunu (strides) a velikost přesahu (padding). Oba tyto parametry nastavuje na počátku uživatel a během tréninku se nijak nemění. Délka posunu slouží k určení toho, o kolik prvků se bude posouvat maska po vstupním tenzoru při výpočtu jednotlivých konvolucí, viz Obr. 2.6. Nejčastěji slouží ke zmenšování výstupu konvoluční operace, které vede k výpočetní úspoře. Velikost přesahu pak udává to, jak se řeší okrajové efekty, tj. jak moc může maska přesáhnout vstupní tenzor. Padding slouží především k prevenci ztráty informace na hranách a postupném zmenšování výstupů při opakované aplikaci konvoluce. Při použití nenulového přesahu se standardně nastavuje vstup nulami po obvodu, viz Obr. 2.6.



Obr. 2.6: Vizualizace konvoluce s posunem $s = (2, 1)$ tzn. (*2 horizontálně, 1 vertikálně*), přesahem $p = 1$ a velikostí masky $f = 3$.

V případě 1D signálu konvoluční vrstva provádí diskrétní jednorozměrnou konvoluci. Pokud $\{y_i\}_{i=1}^{n}$ je vstupní signál délky n, $\{c_j\}_{j=1}^{f}$ je konvoluční jádro délky f, přesah je $p \in \mathbb{N}$ a posun

$s \in \mathbb{N}$, pak je výstupní signál po konvoluci $\{x_k\}_{k=1}^{l}$ daný vztahy

$$l = \lfloor (n - f + 2p)/s \rfloor + 1, \tag{2.27}$$

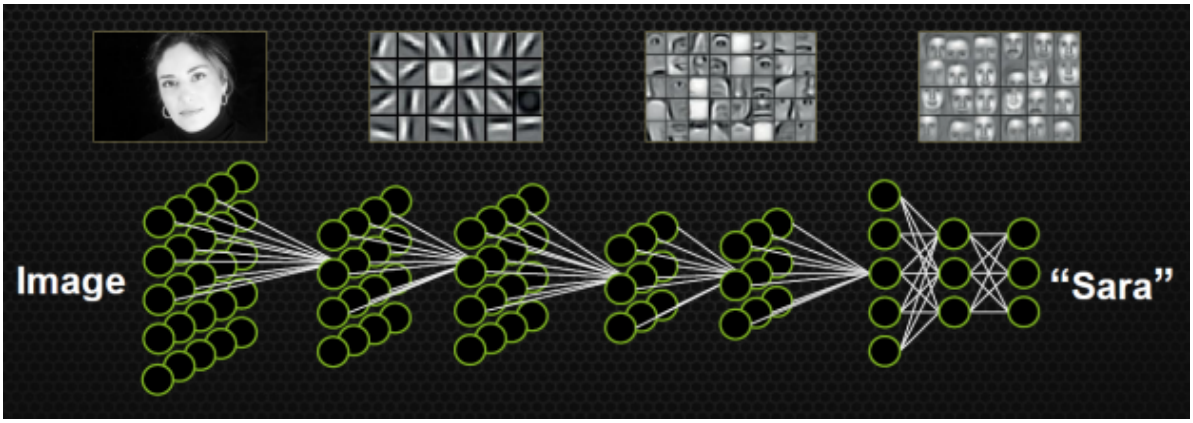
$$x_k = \sum_{j=1}^{f} c_j \tilde{y}_{j+(k-1)s}, \tag{2.28}$$

kde $\tilde{y}$ je vstupní signál y "obalený"z obou stran p nulami.

Hlavní výhodou konvolučních vrstev je mnohem menší počet parametrů, což má za následek menší nároky na paměť a urychluje tak optimalizaci. Tradiční dense vrstvy využívají maticové násobení mezi vstupním vektorem a maticí parametrů, kde pro každý prvek vstupního vektoru a každý neuron je potřeba samostatný parametr. V případě 100-prvkového vstupního vektoru a vrstvou s 64 neurony to dělá 6400 trénovatelných parametrů. Na rozdíl od toho je počet parametrů v konvoluční vrstvě závislý pouze na velikosti a počtu konvolučních masek, které jsou ale zpravidla mnohem menší než je velikost vstupního tenzoru.

Neuronové sítě složené převážně z konvolučních vrstev se nazývají konvoluční neuronové sítě (CNN). Tyto sítě jsou v současnosti neodmyslitelně spjaty s oblastí počítačového vidění. Používají se k detekci a sledování lidí pomocí kamer, uměle vylepšují kvalitu fotografií v mobilech a v rozpoznání objektů na obrázcích překonávají v přesnosti i lidské zástupce [20].

Klíčem úspěchu je rozpoznání a skládání jednoduchých rysů objektů do složitějších a abstraktnějších struktur. Základní představu jak fungují můžeme demonstrovat na klasifikaci lidských tváří Obr. 2.7. První konvoluční vrstvy nejprve detekují různé typy hran na obrázku. Další vrstvy poskládáním těchto hran sestaví složitější tvary, jako je nos, oči, ústa atd. Podobizna obličeje nakonec vznikne kombinací jednotlivých tvarů. Díky translační invarianci konvoluce navíc nezáleží, kde na obrázku se tyto složitější tvary nacházejí, a výsledný příznakový vektor je stejný pro fotografie, na kterých je obličej na jiném místě či trochu jinak vyfocený. Poté lze výsledný příznakový vektor reprezentující tento obličej porovnat reprezentací již známých fotografií a přiřadit mu tak jméno. V případě 1D konvoluce se namísto hran skládají různé sklony, posuny a změny směru obsažené ve vstupní sekvenci.



Obr. 2.7: Skládání příznakových map z konvolučních vrstev za účelem klasifikace lidské tváře [21].

2.3.2 Rekurentní vrstvy

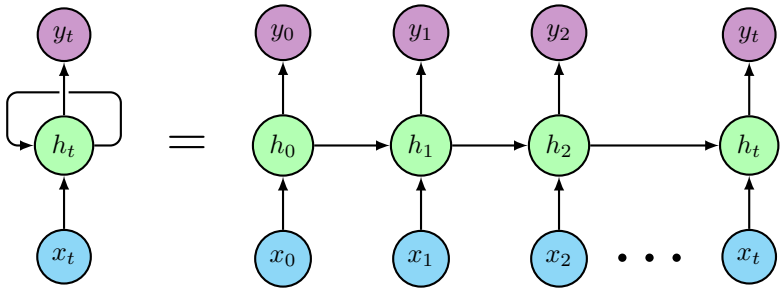
Tradiční dopředné neuronové sítě nemají žádnou "paměť" a přistupují ke každému pozorování jako k izolovanému stavu, který je nezávislý na ostatních. Při práci se sekvenčními daty, kdy

jedno pozorování je obyčejně závislé na těch předchozích (např. časové řady a videa) je žádoucí, aby byl model schopný tyto závislosti zachytit [8].

Řešením tohoto problému jsou rekurentní neuronové vrstvy (resp. sítě), které jsou speciálně navrženy pro zpracování sekvenčních dat. Model rekurentní vrstvy předpokládá závislost aktuálního pozorování na těch předchozích a během tréninku se jí snaží odhalit

$$\boldsymbol{y}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}), \tag{2.29}$$

kde $\boldsymbol{y}_t$ je aktuální stav výstupní veličiny, $\boldsymbol{x}_t$ je aktuální stav vstupu a $\boldsymbol{h}_{t-1}$ je skrytý stav (*paměť*) z předchozího kroku. Na rekurentní vrstvu se můžeme dívat jako na několik kopií jedné vrstvy, kde každá kopie kromě výstupu posílá i další informaci pro své nástupce viz Obr. 2.8.



Obr. 2.8: Ilustrace rekurentní vrstvy rozbalené do takzvaného řetězce.

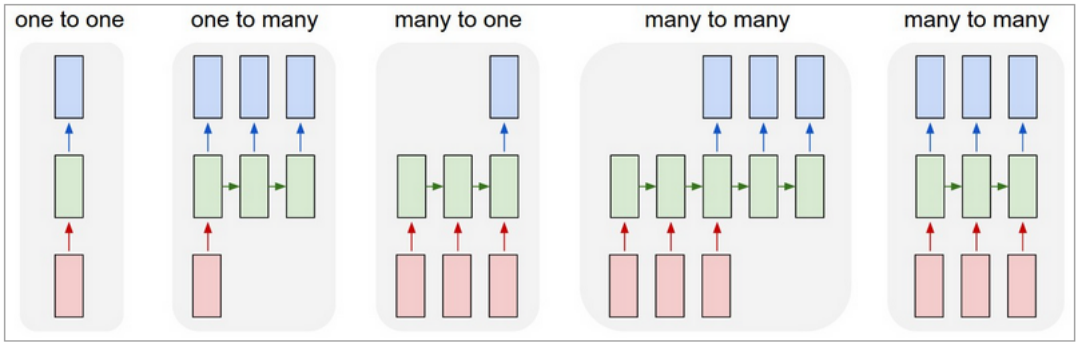
Každá rekurentní vrstva se obecně skládá ze tří obyčejných dense vrstev (2.2) náležející veličinám $\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}$ a $\boldsymbol{y}_t$

$$\boldsymbol{h}_t = \phi_h\big(\boldsymbol{W}_x\boldsymbol{x}_t + \boldsymbol{W}_h\boldsymbol{h}_{t-1} + \boldsymbol{b}_{xh}\big) \tag{2.30}$$

$$\boldsymbol{y}_t = \phi_y\big(\boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y\big) = \phi_y\big(\boldsymbol{W}_y\phi_h\big(\boldsymbol{W}_x\boldsymbol{x}_t + \boldsymbol{W}_h\boldsymbol{h}_{t-1} + \boldsymbol{b}_{xh}\big) + \boldsymbol{b}_y\big). \tag{2.31}$$

Kombinování současného vstupu a předchozího skrytého stavu se zabrání tomu, aby si síť jen zapamatovávala předchozí vstup. Tímto způsobem je díky skrytému stavu přenášena i informace o starších událostech.

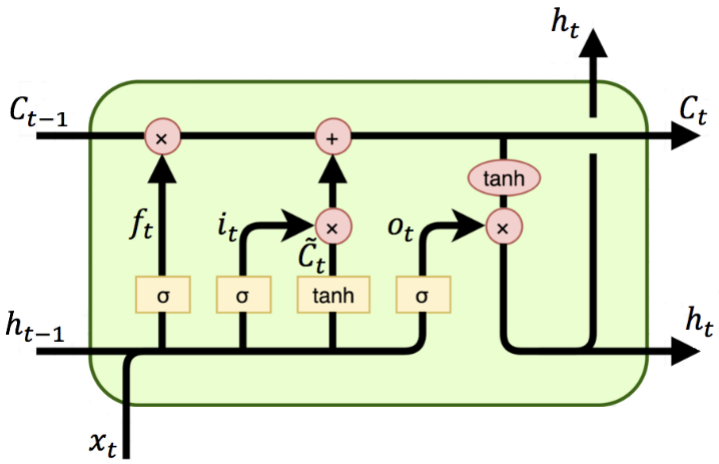
Podle toho, jak se s výstupy a vstupy nakládá, určuje se i způsob využití rekurentní sítě.



Obr. 2.9: Konfigurace rekurentních neuronových sítí [22]. Červené obdélníky symbolizují nenulové vstupní vektory, zelené obdélníky znázorňují rozbalený řetězec rekurentní vrstva a modré značí výstupy využité k dalšímu postupu.

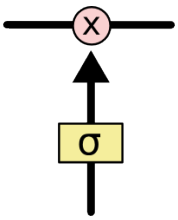
Konfigurace many-to-many je ideální pro seq2seq modely [23], zatímco many-to-one se používá např. ke klasifikaci slovního hodnocení služeb (pozitivní, negativní). V kontextu rekurentních autoencoderů se dá hovořit o spojení konfigurací many-to-one (encoderu) a one-to-many (decoderu).

Jednou speciální verzí rekurentní vrstvy je *Long Short Term Memory* vrstva zkráceně "LSTM", která je schopna zachytit dlouhodobé závislosti. Oproti obyčejným rekurentním vrstvám má velice složitou strukturu skládající se ze čtyř dense vrstev napojených tak, aby si "zapamatovaly" informaci na delší dobu Obr. 2.10.



Obr. 2.10: Struktura LSTM buňky [22].

Tři z těchto vrstev ($\boldsymbol{f}_t, \boldsymbol{i}_t$ a $\boldsymbol{o}_t$) jsou hlavní složkou, tzv. bran, které v LSTM buňce slouží k rozhodnutí, jakou informaci má buňka poslat dál a kterou má zapomenout. Každá taková brána je tvořena jednou dense vrstvou, sigmoidální aktivační funkcí a násobením po prvcích (viz Obr. 2.11).



Obr. 2.11: Ilustrace struktury brány [22].

Díky aktivaci pomocí sigmoidu je výstup brány škálovaný na interval $[0, 1]$ a může tak být chápán jako míra zapomenutí, kde 0 vede k zapomenutí a 1 k poslání celé informace dál. Mate-

maticky můžeme všechny závislosti v LSTM vrstvě popsat pomocí několika rovnic

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \tag{2.32}$$
$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \tag{2.33}$$
$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_C \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C) \tag{2.34}$$
$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{C}}_t \tag{2.35}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \tag{2.36}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t), \tag{2.37}$$
$$\tag{2.38}$$

kde $\boldsymbol{x}_t$ je vstupní vektor příznaků pro t-tý prvek sekvence $\boldsymbol{x}$, $\boldsymbol{h}_t$ je výstup buňky a $\boldsymbol{C}_t$ je stav buňky, jež představuje dlouhodobou informaci získanou z $t-1$ předchozích prvků sekvence $\boldsymbol{x}$. Rovnice (2.34) pak popisuje vztah mezi stavem předchozí buňky $\boldsymbol{C}_{t-1}$ a stavem té současné $\boldsymbol{C}_t$. Vliv (resp. přínos) předchozího stavu je regulován bránou $\boldsymbol{f}_t$, zatímco brána $\boldsymbol{i}_t$ určuje jakou mírou působí na stav současný prvek sekvence $\boldsymbol{x}_t$ spolu s výstupem předchozí buňky $\boldsymbol{h}_{t-1}$. Nyní už je zřejmý původ názvu této vrstvy jež kombinuje dlouhodobou paměť $\boldsymbol{C}_t$ (long term memory) s krátkodobou pamětí $\boldsymbol{h}_{t-1}$ (short term memory). Dimenze obou vektorů je dána počtem neuronů ve vrstvě.

V našem případě doufáme, že $\boldsymbol{h}$ bude nést informaci o krátkodobých změnách ve smyslu přechodů mezi jednotlivými prvky sekvence signálu H_α, zatímco $\boldsymbol{C}$ ponese dlouhodobější informaci o celé sekvenci. Dimenze obou vektorů je dána počtem neuronů ve vrstvě.

V praktickém použití rekurentních neuronových sítí budeme dodržovat tuto konvenci pro vstupní tenzory

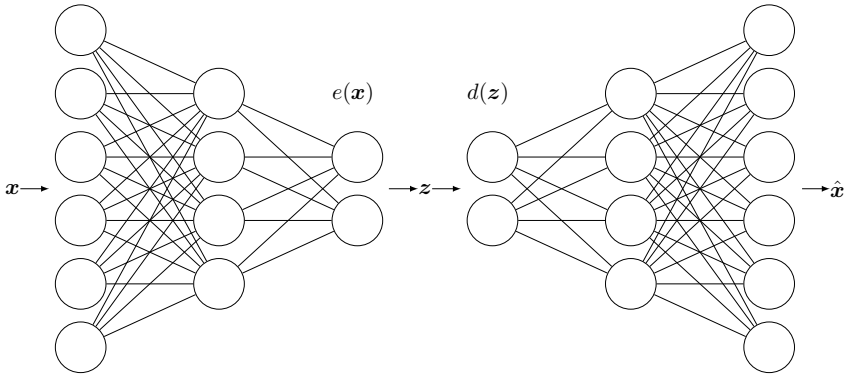
$$\boldsymbol{x} \in \mathbb{R}^{(l,bs,n)}, \tag{2.39}$$

kde l odpovídá délce sekvence, bs je počet vzorků v minibatchi a n je počet příznaků pro jeden prvek. Výstupní tenzor má první dva rozměry stejné, ale místo počtu příznaků představuje nyní n počet neuronů.

LSTM sítě v současnosti dominují v oblasti NLP (natural language processing) při generování textu, strojovém překladu [24], sémantické analýze vět atd. Další uplatněné nacházejí i v předpovídání chování časových řad, generování hudby nebo ovládání robotů. Jejich nevýhodou je relativně náročná optimalizace způsobená velkým množstvím trénovatelných parametrů. To způsobuje pomalejší trénink než u ostatních typů sítí.

2.4 Autoencoder

Autoencodéry jsou klasické dopředné neuronové sítě, které jsou ale trénovány pro zkopírování vstupu na výstup [8]. Prostřední vrstva této sítě obsahuje kódovanou *latentní* reprezentaci vstupu $\boldsymbol{x} \in \mathcal{X}$, kterou budeme značit jako $\boldsymbol{z} \in \mathcal{Z}$, kde prostor $\mathcal{Z}$ označujeme jako *latentní*. První polovina je pak označována jako encoder, tedy zobrazení $e : \mathcal{X} \to \mathcal{Z}$ a druhá polovina jako decoder $d : \mathcal{Z} \to \mathcal{X}$.

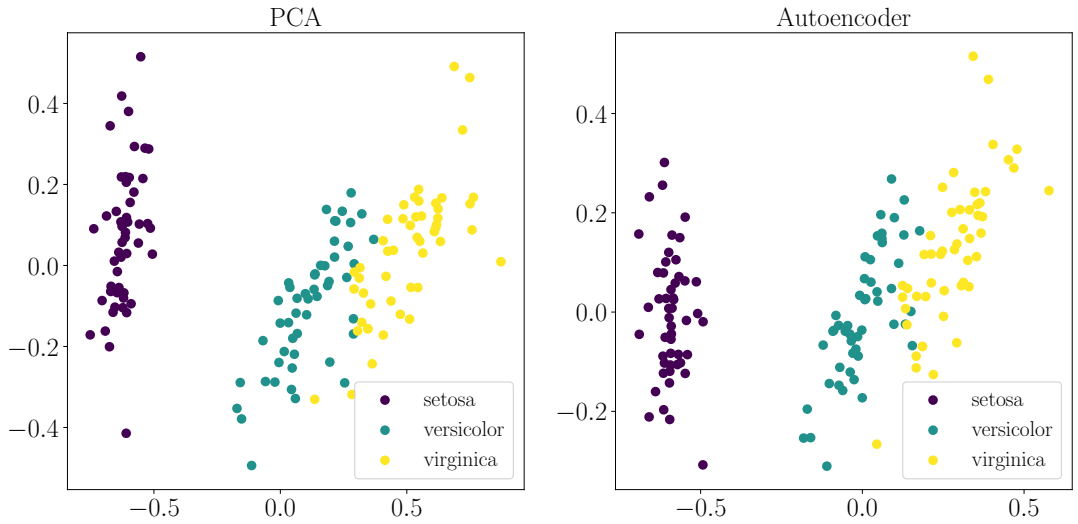


Obr. 2.12: Autoencoder se zúženým latentním prostorem.

Aby se předešlo tomu, že se autoencoder naučí identické zobrazení (což by nebylo příliš užitečné), jsou na něj kladeny různé omezující podmínky, jako třeba zúžení prostřední vrstvy nebo regularizace parametrů. Model je tak nucen rozhodnout, které aspekty vstupu se naučí a které jsou zbytečné. Takto vzniklá aproximace je mnohdy užitečnější než jen kopie.

Tradičně se autoencodery používají pro redukci dimenzionality, extrakci příznaků nebo odšumování vstupu [25]. Při redukci dimenzionality nás obvykle vůbec nezajímá výstup decoderu. Namísto toho doufáme, že se při tréninku encoder naučí komprimovat užitečné vlastnosti z dat do nízkodimenzionální latentní reprezentace. Toho se dosahuje snížením počtu neuronů v prostřední vrstvě, což slouží k odfiltrování nepodstatné informace.

V jednoduchém případě, kdy je za ztrátovou funkci zvolena MSE a aktivační funkce jsou pouze lineární, výsledný latentní prostor odpovídá projekci z analýzy hlavních komponent (PCA), viz Obr. 2.13. Encoder se tak naučil provádět neškálovanou PCA, aniž by k tomu byl explicitně naprogramován. Při bližším rozboru obou metod pak můžeme zjistit, že PCA je ve skutečnosti optimálním řešením, ke kterém autoencoder konverguje [8]. Při použití nelineárních aktivačních funkcí může autoencoder aproximovat silnější nelineární zobecněné PCA [26].



Obr. 2.13: Porovnání metody PCA s lineárním autoencoderem na datasetu Iris [27]. Barvy odpovídají jednotlivým třídám. Na levém obrázku jsou vyneseny první dvě hlavní komponenty PCA a na pravém je vynesena projekce do latentního prostoru autoencoderu.

Kapitola 3

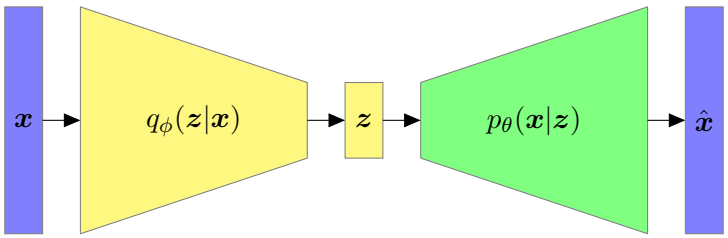
Variační autoencoder

Variační autoencoder, nebo-li VAE, je generativní model, sloužící k hledání pravděpodobnostního rozdělení, které popisuje proces generování cílových dat. Využívá k tomu aproximativní inference a jeho parametry mohou být učeny (resp. trénovány) pomocí algoritmu Auto-encoding variational Bayes [28], který tuto úlohu převádí na obyčejnou gradientní optimalizaci. Od obyčejných autoencoderů se mimo jiné liší tím, že není deterministický, tzn. jeden stejný vstup enkóduje pokaždé jinak.

3.1 Auto-encoding variational Bayes

Myšlenka tohoto algoritmu vychází z předpokladu, že data $\mathbb{X} = (\boldsymbol{x}^1, \cdots, \boldsymbol{x}^N)$ jsou generována nějakým náhodným procesem, jež zahrnuje nepozorované spojité náhodné veličiny $\boldsymbol{z}$. Tento proces tedy nejprve generuje $\boldsymbol{z}^n$ z nějakého apriorního rozdělení $p_{\theta^*}(\boldsymbol{z}) = p(\boldsymbol{z}|\theta^*)$ a následně pak generuje i $\boldsymbol{x}^n$ z tentokráte již podmíněné distribuce $p_{\theta^*}(\boldsymbol{x}|\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z}, \theta^*)$. Dále předpokládáme, že obě pocházejí z parametrických rodin distribucí $p_\theta(\boldsymbol{z})$ a $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ a že jejich hustoty pravděpodobností jsou diferencovatelné podle θ a $\boldsymbol{z}$ skoro všude. Naneštěstí je tento proces skrytý a jeho skutečné parametry θ^* jsou stejně jako latentní proměnné $\boldsymbol{z}$ neznámé [28].

Aby bylo možné tento problém řešit, zavádíme pravděpodobnostní encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ s parametry ϕ, který aproximuje analyticky nevyčíslitelnou skutečnou aposteriorní hustotu pravděpodobnosti $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Pravděpodobnostní je, protože namísto jedné hodnoty $\boldsymbol{z}$ vrací celou distribuci pro všechny možné hodnoty $\boldsymbol{z}$ za daného $\boldsymbol{x}$. Stejně tak je zaveden i decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, který je také pravděpodobnostní, ale jeho výstup poskytuje rozdělení pro $\boldsymbol{x}$ za daného $\boldsymbol{z}$. Spojení encoderu $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ a decoderu $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ nám tak umožňuje simultánní učení parametrů ϕ a θ (Obr. 3.1).



Obr. 3.1: Schéma variačního autoencoderu.

Pro takto definovaný budeme model chtít maximalizovat marginální věrohodnost, jež je rovna součtu marginálních věrohodností pro jednotlivé pozorování

$$\log p_\theta(\mathbb{X}) = \log p_\theta(\boldsymbol{x}^1, \cdots, \boldsymbol{x}^N) = \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}^n). \tag{3.1}$$

Věrohodnost každého pozorování můžeme podle (A.1) a (A.2) přepsat jako

$$\log p_\theta(\boldsymbol{x}^n) = D_{KL}\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)||p_\theta(\boldsymbol{z}|\boldsymbol{x}^n)\Big) + \mathcal{L}(\theta, \phi; \boldsymbol{x}^n). \tag{3.2}$$

První složkou levé strany (3.2) je Kullback–Leiblerova divergence mezi aproximovanou a skutečnou hustotou pravděpodobnosti

$$\begin{aligned} D_{KL}(q(x)||p(x)) &= \int_{\mathbb{R}} q(x) \log\left(\frac{q(x)}{p(x)}\right) \\ &= \mathbb{E}_{q(x)}\Big[-\log p(x) + \log q(x)\Big] \end{aligned} \tag{3.3}$$

Druhý výraz (3.2) odpovídá tzv. spodní mezi marginální věrohodnosti (angl. evidence lower bound - ELBO) a lze jí přepsat jako

$$\begin{aligned} \mathcal{L}(\theta, \phi; \boldsymbol{x}^n) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) + \log p_\theta(\boldsymbol{x}, \boldsymbol{z})\Big] \\ &= -D_{KL}\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)||p_\theta(\boldsymbol{z})\Big) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)}\Big[\log p_\theta(\boldsymbol{x}^n|\boldsymbol{z})\Big], \end{aligned} \tag{3.4}$$

kde člen se střední hodnotou odpovídá očekávané rekonstrukční chybě a D_{KL} se v tomto případě chová jako regularizační člen vynucující podobnost apriorní a aposteriorní hustoty.

Věrohodnost (3.2) nelze vyčíslit přímo, jelikož neznáme hodnotu $p_\theta(\boldsymbol{z}|\boldsymbol{x}^n)$. Můžeme ale využít toho, že hodnota K-L divergence je vždy kladné číslo. Pak z rovnosti (3.2) plyne vztah

$$\log p_\theta(\boldsymbol{x}^n) \geq \mathcal{L}(\theta, \phi; \boldsymbol{x}^n) \implies \log p_\theta(\mathbb{X}) \geq \mathcal{L}(\theta, \phi; \mathbb{X}). \tag{3.5}$$

Díky tomu místo maximalizace věrohodnosti $\log p_\theta(\mathbb{X})$ stačí maximalizovat $\mathcal{L}(\theta, \phi; \mathbb{X})$ resp. minimalizovat $-\mathcal{L}(\theta, \phi; \mathbb{X})$ vzhledem k parametrům θ a ϕ.

Aby bylo možné optimalizovat $\mathcal{L}(\theta, \phi; \mathbb{X})$ pomocí standardních gradientních algoritmů jako je třeba Adam ((2.12)-(2.15)), musíme provést ještě jednu úpravu, kterou je tzv. reparametrizační trik. Tento trik se používá, protože zpětná propagace není schopná spočítat gradienty náhodných veličin (Obr. 3.2).

Proto provedeme reparametrizaci $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$ za pomoci diferencovatelné deterministické transformace $q_\phi(\epsilon, \boldsymbol{x})$ a šumu ϵ tak, že

$$\widetilde{\boldsymbol{z}} = q_\phi(\epsilon, \boldsymbol{x}) \text{ a } \epsilon \sim p(\epsilon). \tag{3.6}$$

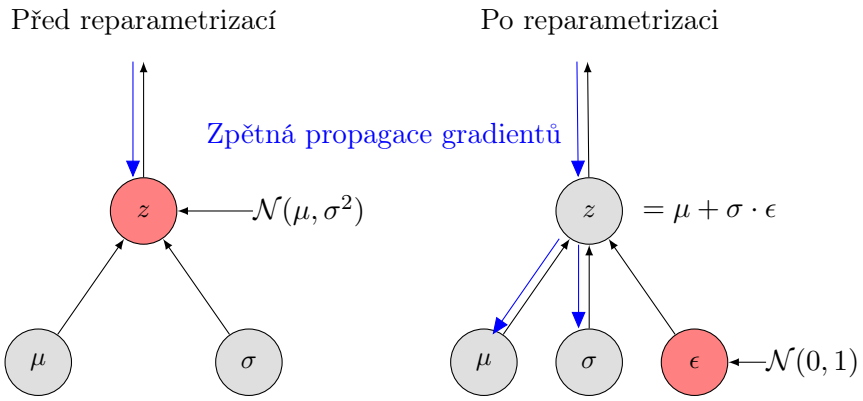
Pro případ jednorozměrného normálního rozdělení $z \sim \mathcal{N}(\mu, \sigma^2)$ by reparametrizace vypadala následovně: $z = \mu + \sigma \cdot \epsilon$ kde $\epsilon \sim \mathcal{N}(0, 1)$ a platí

$$\mathbb{E}_{\mathcal{N}(z|\mu,\sigma^2)}\big[f(z)\big] = \mathbb{E}_{\mathcal{N}(0,1)}\big[f(\mu + \sigma \cdot \epsilon)\big] \simeq \frac{1}{L}\sum_{l=1}^{L} f(\mu + \sigma \cdot \epsilon), \epsilon \sim \mathcal{N}(0, 1). \tag{3.7}$$

Nyní již můžeme napsat finální verzi variační dolní meze, kterou je možné maximalizovat pomocí libovolného gradientního optimalizačního algoritmu

$$\begin{aligned} \mathcal{L}(\theta, \phi; \mathbb{X}) &= \sum_{n=1}^{N} -D_{KL}\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)||p_\theta(\boldsymbol{z})\Big) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)}\Big[\log p_\theta(\boldsymbol{x}^n|\boldsymbol{z}^n)\Big] \\ &= \sum_{n=1}^{N}\Big[-D_{KL}\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)||p_\theta(\boldsymbol{z})\Big) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{x}^n|\boldsymbol{z}^{n,l})\Big], \end{aligned} \tag{3.8}$$

kde $\boldsymbol{z}^{n,l} = q_\phi(\boldsymbol{\epsilon}^l, \boldsymbol{x}^n)$, $\boldsymbol{\epsilon}^l \sim p(\boldsymbol{\epsilon})$, L je počet vzorků a D_{KL} je analyticky vyčíslitelná KL divergence mezi aposteriorním rozdělením enkódovaných data a apriornem.



Obr. 3.2: Diagram průchodu gradientů při zpětné propagaci před a po reparametrizaci. Gradient nelze spočítat pro stochastické (červené) buňky.

3.2 VAE a neuronové sítě

Pro variační autoencoder, jak je definovaný v předchozí sekci, lze použít jako encoder resp. decoder libovolné parametrické funkce, jež splňují dané předpoklady. V současnosti se jako tyto funkce používají především neuronové sítě, jenž jsou z tohoto pohledu nelineárními parametrickými funkcemi, které jsou navíc diferencovatelné a tím pádem se dají trénovat gradientními algoritmy.

Pravděpodobnostní encoder je tedy tvořen neuronovou sítí, jejímž výstupem jsou parametry aproximativního rozdělení $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Vzorky z této distribuce pak slouží jako vstupní vektory pro decoder, který je tvořen další sítí. Cílem decoderu je predikovat parametry rozdělení $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ tak, aby co nejlépe odpovídalo pozorováním $\boldsymbol{x}$. V zájmu jednoduššího hledání hyperparametrů bývá architektura symetrická podle zúžení, jako v případě obyčejného autoencoderu, viz Obr. 2.12.

Pro encodery, které v této práci používáme, budeme předpokládat, že apriorní rozdělení $p_\theta(\boldsymbol{z})$ je rovno centrovanému vícerozměrnému normálnímu rozdělení $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Aproximovaná aposteriorní hustota pak nabývá normálního rozdělení s diagonální kovarianční maticí

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}^n) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}^n, \operatorname{diag}\left((\boldsymbol{\sigma}^n)^2\right)), \tag{3.9}$$

kde $\boldsymbol{\mu}^n = \boldsymbol{\mu}_\phi(\boldsymbol{x}^n)$ a $\boldsymbol{\sigma}^n = \boldsymbol{\sigma}_\phi(\boldsymbol{x}^n)$ jsou vektor středních hodnot a vektor směrodatných odchylek. Právě tyto parametry $\boldsymbol{\mu}^n$ a $\boldsymbol{\sigma}^n$ budou výstupem neuronové sítě, přičemž oba jsou závislé na vstupním pozorování $\boldsymbol{x}^n$ a parametrech sítě $\phi = \{\boldsymbol{W}^1, \boldsymbol{b}^1, \boldsymbol{W}^2, \boldsymbol{b}^2, \cdots\}$. Vzorky náhodné veličiny $\boldsymbol{z}^{n,l} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)$ získáme vektorovou verzí reparametrizace (3.7), tzn.

$$\boldsymbol{z}^{n,l} = q_\phi(\boldsymbol{\epsilon}^l, \boldsymbol{x}^n) = \boldsymbol{\mu}^n + \boldsymbol{\sigma}^n \odot \boldsymbol{\epsilon}^l, \tag{3.10}$$

kde $\boldsymbol{\epsilon}^l \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Kullback–Leiblerova divergence mezi aposteriorní hustotou $q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)$ a apriorní

hustotou $p_\theta(\boldsymbol{z})$ lze dále spočítat analyticky pomocí vzorců (A.3) a (A.4) jako

$$\begin{aligned} D_{KL}\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)||p_\theta(\boldsymbol{z})\Big) &= -\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)}\Big[\log p_\theta(\boldsymbol{z})\Big] + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)}\Big[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}^n)\Big] \\ &= \frac{1}{2}\sum_{j=1}^{J}\Big(\log(2\pi) + (\mu_j^n)^2 + (\sigma_j^n)^2\Big) - \frac{1}{2}\sum_{j=1}^{J}\Big(\log 2\pi + \log(\sigma_j^n)^2 + 1\Big) \\ &= -\frac{1}{2}\sum_{j=1}^{J}\Big(1 + \log(\sigma_j^n)^2 - (\mu_j^n)^2 - (\sigma_j^n)^2\Big), \end{aligned} \tag{3.11}$$

kde J je dimenze latentního prostoru resp. vektoru $\boldsymbol{z}$.

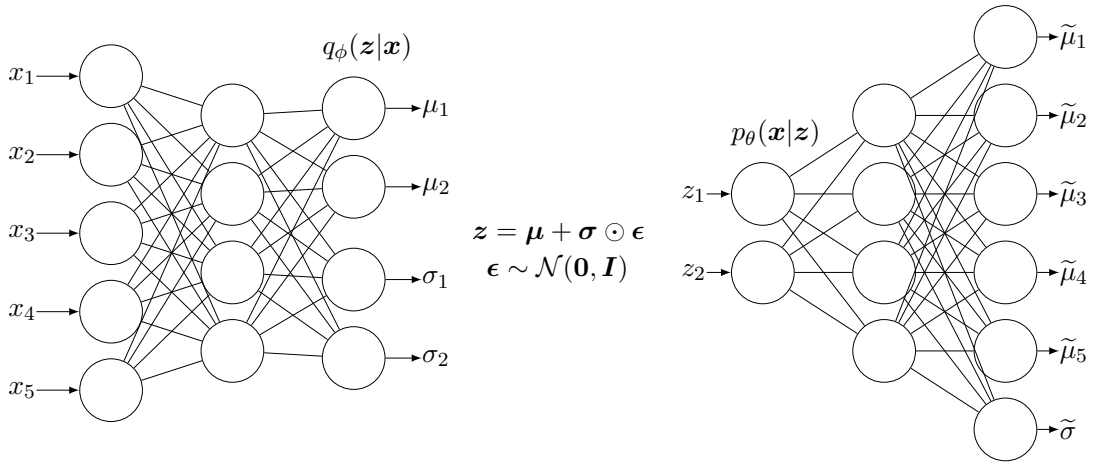
V neposlední řadě je třeba ještě zvolit rozdělení výstupu decoderu kvůli typu rekonstrukční chyby. Při rekonstrukci a generaci obrázku se nejčastěji používá Bernoulliho rozdělení, které pro každý pixel odhaduje pravděpodobnost $p \in [0, 1]$ odpovídající míře jasu tohoto pixelu. Vzhledem k povaze naší úlohy, kdy se snažíme zrekonstruovat signál zakódovaný encoderem, by výstup decoderu měl mít spíše normální rozdělení, neboť se jedná v podstatě o regresní úlohu. V případě, že bychom se spokojili s $\mathcal{N}(\boldsymbol{x}|\widetilde{\boldsymbol{\mu}}^n, \boldsymbol{I})$, které se při regresi obyčejně používá, byla by očekávaná rekonstrukční chyba pro $\boldsymbol{x}^n$ až na konstantu rovna Mean Square Error

$$\frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{x}^n|\boldsymbol{z}^{n,l}) = \frac{1}{L}\sum_{l=1}^{L}\log\mathcal{N}(\boldsymbol{x}^n|\widetilde{\boldsymbol{\mu}}^{n,l}, \boldsymbol{I}) \approx \frac{1}{L}\sum_{l=1}^{L}\|\boldsymbol{x}^n - \widetilde{\boldsymbol{\mu}}^{n,l}\|^2. \tag{3.12}$$

Mnohem lepší však bude nechat decoder generovat výstup s nejednotkovým rozptylem, tzn. $\mathcal{N}(\boldsymbol{x}|\widetilde{\boldsymbol{\mu}}^n, (\widetilde{\sigma}^n)^2\boldsymbol{I})$. Model tak bude kromě odhadů vektoru středních hodnot odhadovat i směrodatnou odchylku σ^n, jež je pro všechny složky vektoru $\widetilde{\boldsymbol{\mu}}^n$ stejná. Hodnota směrodatné odchylky bude tak nepřímo vypovídat o tom, jak moc model věří poskytnuté rekonstrukci $\boldsymbol{x}^n$. Čím menší odchylka bude, tím přesnější by měla být rekonstrukce. Očekávaná rekonstrukční chyba výstupu s tímto rozdělením pak bude

$$\frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{x}^n|\boldsymbol{z}^{n,l}) = -\frac{1}{L}\sum_{l=1}^{L}\Big(\frac{\|\boldsymbol{x}^n - \widetilde{\boldsymbol{\mu}}^{n,l}\|^2}{2\widetilde{\sigma}^{n,l}} + \frac{\widetilde{J}}{2}\log(\widetilde{\sigma}^{n,l})^2 + \frac{\widetilde{J}}{2}\log(2\pi)\Big), \tag{3.13}$$

kde $\widetilde{J}$ je dimenze vektoru $\boldsymbol{x}^n$.

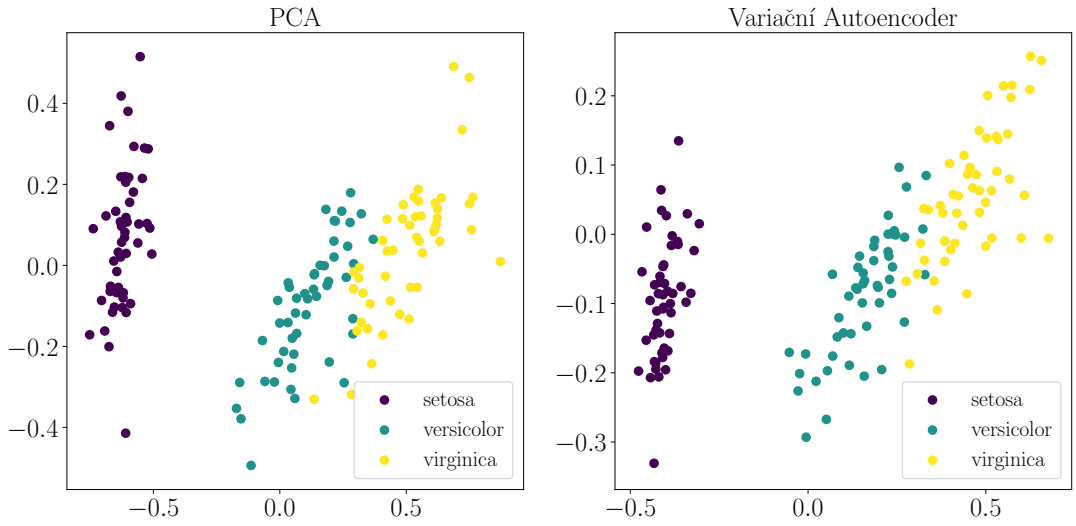


Obr. 3.3: Variační autoencoder s použitím neuronových sítí, kde aproximativní distribuce encoderu je $q_\phi(\boldsymbol{z}|\boldsymbol{x}^n) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$ a rozdělení výstupu decoderu je $p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|\widetilde{\boldsymbol{\mu}}, \widetilde{\sigma}^2\boldsymbol{I})$.

Nyní, když je model kompletní i se všemi rozděleními (Obr. 3.3), můžeme zapsat finální ztrátovou funkci, kterou je třeba minimalizovat

$$\begin{aligned} L(\boldsymbol{\theta}) &= -\mathcal{L}(\theta, \phi; \mathbb{X}) \\ &= -\sum_{n=1}^{N} \left(\frac{1}{2} \sum_{j=1}^{J} \left(1 + \log(\sigma_j^n)^2 - (\mu_j^n)^2 - (\sigma_j^n)^2 \right) \right. \\ &\quad \left. - \frac{1}{L} \sum_{l=1}^{L} \left(\frac{\|\boldsymbol{x}^n - \widetilde{\boldsymbol{\mu}}^{n,l}\|^2}{2\widetilde{\sigma}^{n,l}} + \frac{\widetilde{J}}{2} \log(\widetilde{\sigma}^{n,l})^2 + \frac{\widetilde{J}}{2} \log(2\pi) \right) \right), \end{aligned} \tag{3.14}$$

kde $\boldsymbol{\theta} = \{\theta, \phi\}$ je soubor parametrů obou neuronových sítí (encoder a decoder), přičemž optimalizace θ a ϕ probíhá najednou. Navíc v případě optimalizace pomocí minibatchů, které obsahují více než 100 pozorování, není třeba vzorkovat z decoderu, tzn. můžeme zvolit $L = 1$ [28].



Obr. 3.4: Porovnání metody PCA s lineárním variačním autoencoderem na datasetu Iris [27]. Barvy odpovídají jednotlivým třídám. Na levém obrázku jsou vyneseny první dvě hlavní komponenty PCA a na pravém je vynesena projekce do latentního prostoru variačního autoencoderu.

3.2.1 Konvoluční variační autoencoder

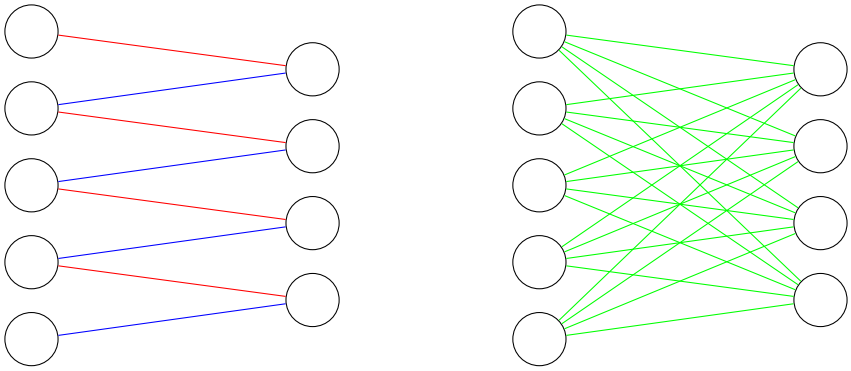
Kromě výše zmíněného standardního VAE s dense vrstvami budeme používat ještě konvoluční (CVAE) a rekurentní (RVAE) variační autoencodery, které by měly být pro práci s časovými řadami mnohem vhodnější.

Konvoluční verze se od původní verze příliš neliší. Namísto obyčejných vrstev jsou zde používány CBN bloky, které se od obyčejné konvoluční vrstvy liší přidáním normalizační vrstvy mezi výstup konvoluce a aktivační funkci. Tato normalizační vrstva známá jako Batch normalization zvyšuje stabilitu, rychlost a kvalitu hlubokých neuronových sítí [29].

Konvoluční encoder je tvořen několika takovýmito bloky napojenými za sebe. Poslední vrstva encoderu je však stejně jako u VAE obyčejná dense vrstva, která příznaky extrahované pomocí konvoluce zkomprimuje do latentního prostoru.

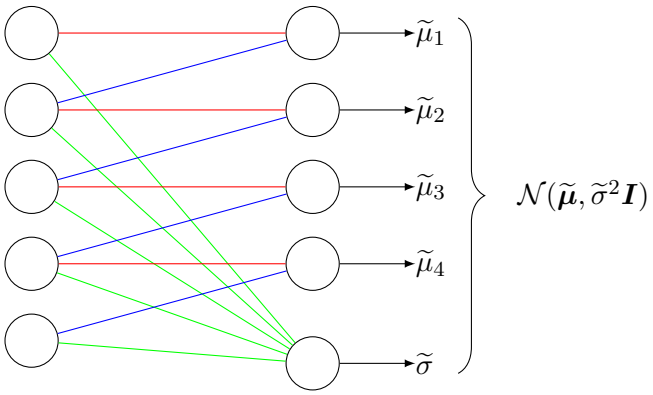
Největší rozdíl je až na výstupu decodéru. Výstupní vrstva standardního VAE se skládá z $\tilde{J}+1$ neuronů, kde prvních $\tilde{J}$ neuronů vrací odhady středních hodnot a poslední neuron odhaduje

směrodatnou odchylku. Stejný postup však nelze použít u konvolučního decodéru, protože každý výstup konvoluční vrstvy nese informaci pouze z okolí velkosti konvolučního jádra, viz Obr. 3.5 a chtít tak odhadnou směrodatnou odchylku pro celý vektor jen na základě malého okolí, není příliš vhodné.



Obr. 3.5: Rozdíl mezi konvoluční a dense vrstvou. Modré a červené spojnice jsou sdílené váhy konvoluční masky s velikostí dva, která se posouvá s krokem jedna po výstupu předchozí konvoluční vrstvy. Zelené spojnice představují váhy jednoho neuronu, přičemž každá z nich představuje unikátní parametr.

Z tohoto důvodu používáme pro konvoluční decodér speciální výstupní vrstvu, která se skládá z konvoluční vrstvy a jednoho obyčejného neuronu sloužícího k odhadu směrodatné odchylky s využitím informace z celého vstupu viz Obr. 3.6.



Obr. 3.6: Ilustrace výstupní vrstvy z konvolučního decoderu. Modré a červené spojnice jsou sdílené váhy konvoluční masky s velikostí dva, která se posouvá s krokem jedna po výstupu předchozí konvoluční vrstvy. Zelené spojnice představují váhy jednoho neuronu, přičemž každá z nich představuje unikátní parametr. Například takto jednoduše definovaná vrstva má 7 trénovatelných parametrů.

Dalším důležitým nastavením konvolučního VAE jsou parametry konvolučních vrstvev, tj. velikosti jádra, délky posunů a přesahů. Výstupní rozměry z konvoluční vrstvy jsou dány předpisem (2.27). Přítomnost dolní celé části může při špatné volbě parametrů způsobovat problémy se zachováním rozměrů v decoderu, které můžeme demonstrovat na jednoduchém příkladě. Uvažujme jednu konvoluční vrstvu s velikostí jádra 3 a posunem 2 bez přesahu. Na vstupní vektor

délky 10 aplikujeme konvoluci a podle (2.27) dostáváme výstupní vektor délky 4. Pokud vzorec (2.27) obrátíme a dosadíme za vstupní velikost číslo 4, tak zjistíme, že vstupní vektor by měl mít délku 9, což se neshoduje se skutečnými rozměry vstupu. Proto je nutné správně nastavit posun, respektive přesah konvoluce při zachování symetrie velikosti jader v mezi enkodérem a dekodérem.

3.2.2 Rekurentní variační autoencoder

Rekurentní variační autoencoder (RVAE) má ze všech námi použitých VAE nejsložitější strukturu. Jeho architektura vychází ze seq2seq modelů sloužících ke strojovému překladu z jednoho jazyka do druhého [23].

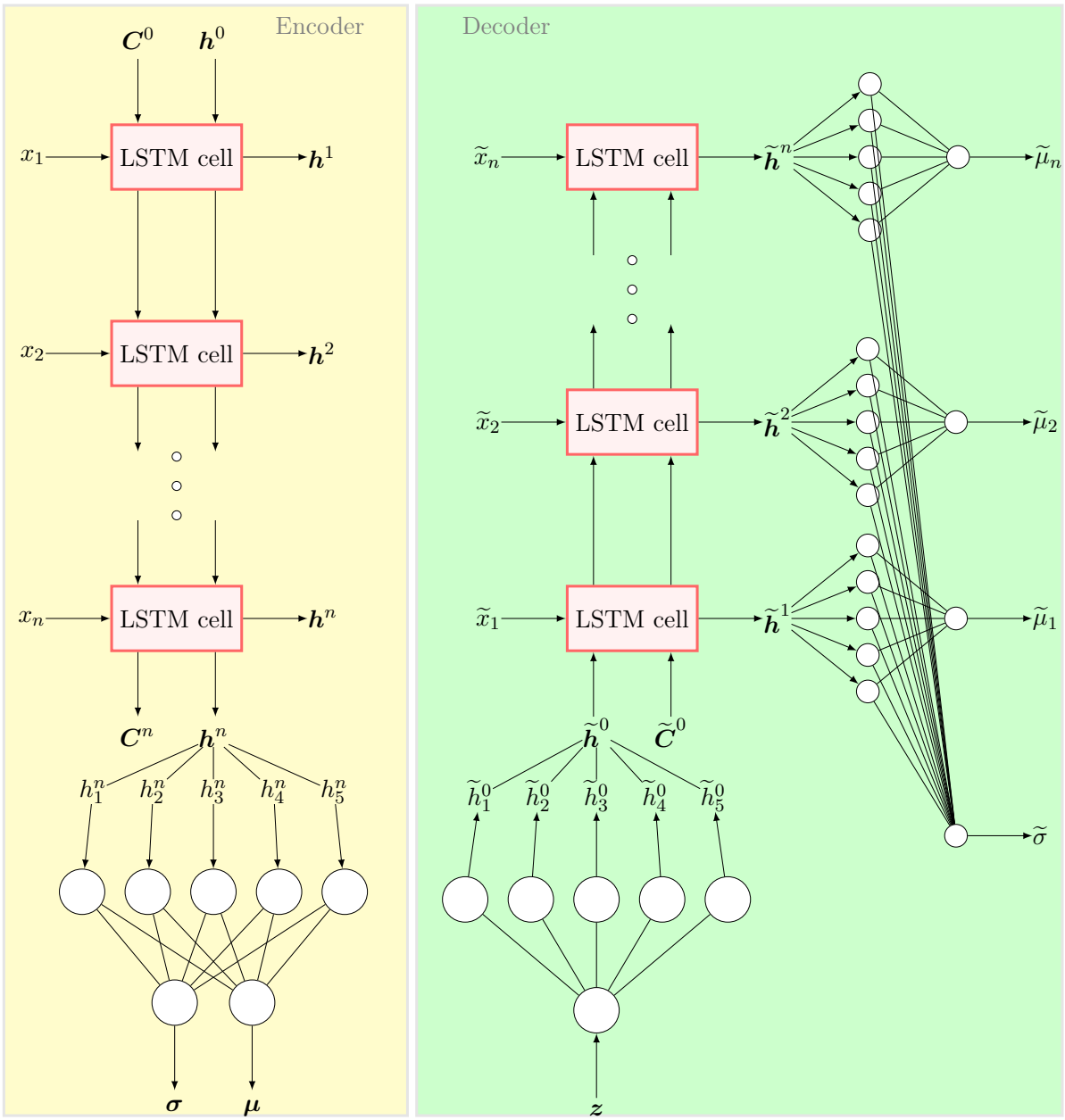
Encoder je tvořen jednou rekurentní vrstvou, která ústí do jednoduché dense vrstvy zakončené reparametrizačním trikem. Tím však podobnost s předchozími VAE končí. U rekurentního encoderu nás totiž spíše než výstup rekurentní vrstvy zajímá její skrytý stav po posledním prvku sekvence. Ten je závislý na všech předchozích prvcích dané sekvence a měl by tak nést informaci o celé sekvenci.

LSTM vrstvy, které používáme v encoderu i decoderu, mají kromě skrytého stavu $\boldsymbol{h}$ (krátkodobé paměti) navíc i tzv. stav buňky $\boldsymbol{C}$ (dlouhodobou paměť). Podle [30] by k extrakci příznaků měly opět stačit pouze $\boldsymbol{h}$, ale vzhledem k povaze úlohy se domníváme, že $\boldsymbol{h}$ bude obsahovat spíše vztahy mezi jednotlivými pozorováními, zatímco $\boldsymbol{C}$ by mělo být schopné poskytnout globálnější pohled na celou sekvenci. Proto vyzkoušíme obě verze. RVAE(h,C) bude označovat rekurentní variační autoencoder využívající $\boldsymbol{C}^n$ i $\boldsymbol{h}^n$ a RVAE(h) bude představovat jednodušší verzi závislou pouze na skrytém stavu $\boldsymbol{h}^n$.

Decoder bude pro obě verze téměř totožný. Rozdíl bude pouze v dvojnásobném počtu výstupů úvodní dense vrstvy. Úkolem této vrstvy je převézt komprimované vzorky $\boldsymbol{z}$ zpět na $\widetilde{\boldsymbol{h}}^0$ a $\widetilde{\boldsymbol{C}}^0$ (resp. jenom $\widetilde{\boldsymbol{h}}^0$), které nyní slouží jako inicializace pro LSTM vrstvu, se stejnými rozměry i parametry jako je v encoderu. V případě RVAE(h) je třeba dlouhodobou paměť $\widetilde{\boldsymbol{C}}^0$ inicializovat alespoň jako nulový vektor.

Pro správné fungování rekurentní vrstvy jsme nuceni dodat vstupní sekvenci, i přestože u decoderu žádnou nemáme, proto budeme na vstup vkládat nulový tenzor $\widetilde{\boldsymbol{x}}$ se stejnými rozměry jak původní sekvence $\boldsymbol{x}$.

Poslední část decoderu je tvořena dvěma dense vrstvami, kde jedna slouží k odhadu směrodatná odchylky výstupu $\widetilde{\sigma}$ na základě všech skrytých stavů $\widetilde{\boldsymbol{h}}^i$, $i \in \{1, \cdots, n\}$ a druhá postupně převádí jednotlivé $\widetilde{\boldsymbol{h}}^i$ na odhady středních hodnot $\widetilde{\mu}_i$ viz Obr. 3.7.



Obr. 3.7: Architektura rekurentního variačního autoencoderu RVAE(h) využívajícího LSTM vrstevy. Dimenze latentního prostoru je ve skutečnosti mnohem větší než jedna a proto je značený vektorem namísto skalární hodnoty. Toto slouží k větší přehlednosti. V případě RVAE(h,C) vstupuje do dense vrstvy v encoderu kromě $\boldsymbol{h}^n$ i $\boldsymbol{C}^n$.

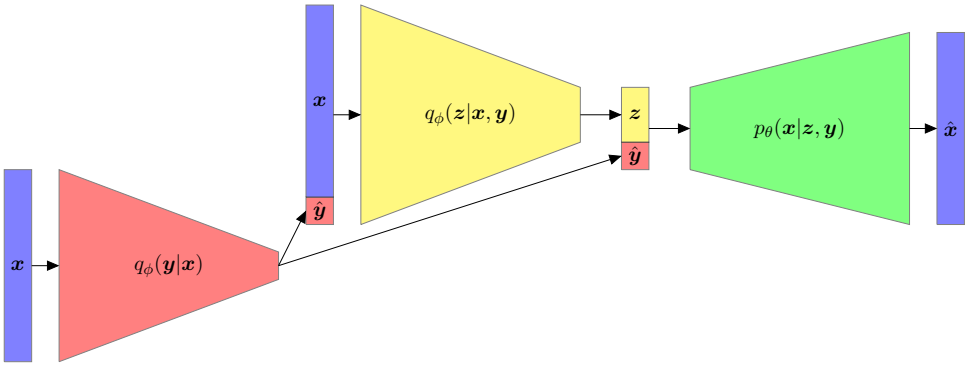
3.3 Semi-supervised VAE

V předchozích sekcích jsme se věnovali tomu jak lze využít variační autoencoder pro redukci dimenzionality a extrakci příznaků. Oba tyto postupy mohou vézt ke zrychlení, resp. zkvalitnění procesu učení sekundárním klasifikačním algoritmem, ale i přes to je následná klasifikace

limitovaná počtem označených dat. Všechny klasifikátory a především ty využívající neuronové sítě potřebují k dosažení uspokojivé přesnosti velké množství označených dat. Pokud je těchto dat málo, dochází k chybám v klasifikaci a během tréninku se také častěji projevují nežádoucí efekty jako je např. přetrénování modelu (overfitting). Někdy je ale k dispozici kromě malé databáze označených dat i další, potenciálně větší množina dat neoznačených, které mají stejné charakteristiky jako ty označené. Lze je nějakým způsobem použít ke zlepšení klasifikace?

Řešením tohoto problému se zabývá podoblast strojového učení známá jako Semi-supervised learning. Semi-supervised learning zahrnuje metody, které využívají neoznačených dat při řešení klasifikačních a regresních problémů. Jednou z těchto metod je i Semi-supervised VAE známý také jako *VAE M2* [31], který využívá vlastností variačního autoencoderu ke zkvalitňování klasifikačního modelu pomocí neoznačených dat.

Podobně jako u obyčejného VAE předpokládáme, že data $\boldsymbol{x}$ jsou generována v závislosti spojité latentní proměnné $\boldsymbol{z}$ a diskrétní latentní proměnné $\boldsymbol{y}$, které představují klasifikační třídy, přičemž obě jsou marginálně nezávislé. Dále předpokládáme, že apriorní rozdělení proměnné $\boldsymbol{z}$ je standardní normální rozdělení $p_\theta(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ a $\boldsymbol{y}$ je rozdělené multinomicky (resp. kategoricky) $p_\theta(\boldsymbol{y}) = \mathrm{Cat}(\boldsymbol{\pi})$, kde $\boldsymbol{\pi}$ je vektor pravděpodobností, jež je pro čtyři třídy apriorně roven $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$.



Obr. 3.8: Vizualizace rozložení semi-supervised variačního autoencoderu.

Dále opět zavedeme aproximativní aposteriorní distribuci – nyní pro obě latentní proměnné najednou. Rozdělení encoderu $q_\phi(\boldsymbol{z}, \boldsymbol{y}|\boldsymbol{x})$ a rozdělení decoderu $p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{y})$. Takto definovaná sdružená distribuce q_ϕ by se obtížně hledala a nevyhovovala by navíc našim potřebám. Proto jí rozdělíme podle vzorce

$$q_\phi(\boldsymbol{z}, \boldsymbol{y}|\boldsymbol{x}) = \frac{q_\phi(\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{x})}{q_\phi(\boldsymbol{x})} \cdot \frac{q_\phi(\boldsymbol{x}, \boldsymbol{y})}{q_\phi(\boldsymbol{x}, \boldsymbol{y})} = q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}) \cdot q_\phi(\boldsymbol{y}|\boldsymbol{x}) \tag{3.15}$$

na dvě složky (Obr. 3.8). Každá latentní proměnná má tak svou vlastní aproximativní distribuci

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}\big(\boldsymbol{\mu}_\phi(\boldsymbol{x}, \boldsymbol{y}), \mathrm{diag}(\boldsymbol{\sigma}^2_\phi(\boldsymbol{x}, \boldsymbol{y}))\big) \tag{3.16}$$

$$q_\phi(\boldsymbol{y}|\boldsymbol{x}) = \mathrm{Cat}(\boldsymbol{\pi}_\phi(\boldsymbol{x})), \tag{3.17}$$

kde $\boldsymbol{\mu}_\phi(\boldsymbol{x}, \boldsymbol{y})$, $\boldsymbol{\sigma}_\phi(\boldsymbol{x}, \boldsymbol{y})$ a $\boldsymbol{\pi}_\phi(\boldsymbol{x})$ jsou vektorové funkce tvořené neuronovými sítěmi.

Pro takto definovaný model budeme opět maximalizovat marginální věrohodnost, která se skládá z maximální věrohodnosti označených dat $\log p_\theta(\boldsymbol{x}, \boldsymbol{y})$ a neoznačených dat $\log p_\theta(\boldsymbol{x})$.

Spodní mez marginální věrohodnosti $\mathcal{L}(\phi, \theta, \boldsymbol{x}, \boldsymbol{y})$ pro označená data získáme modifikací rovnice (3.4), kdy přibude člen příslušný apriornímu rozdělení $\boldsymbol{y}$

$$\begin{aligned}
\mathcal{L}(\phi, \theta, \boldsymbol{x}, \boldsymbol{y}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y}) + \log p_\theta(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\Big] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y}) + \log p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) + \log p_\theta(\boldsymbol{y},\boldsymbol{z})\Big] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y}) + \log p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) + \log p_\theta(\boldsymbol{y}) + \log p_\theta(\boldsymbol{z})\Big] \\
&= -D_{KL}\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})||p_\theta(\boldsymbol{z})\Big) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\Big[\log p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z})\Big] + \log p_\theta(\boldsymbol{y}). \qquad (3.18)
\end{aligned}$$

Pro neoznačená data obdržíme ELBO výpočtem očekávané hodnoty předchozí rovnice vzhledem k aposteriornímu rozdělení $\boldsymbol{y}$

$$\begin{aligned}
\mathcal{L}(\phi, \theta, \boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{y},\boldsymbol{z}|\boldsymbol{x})}\Big[-\log q_\phi(\boldsymbol{y},\boldsymbol{z}|\boldsymbol{x}) + \log p_\theta(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\Big] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{y}|\boldsymbol{x})}\left[\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y}) - \log q_\phi(\boldsymbol{y}|\boldsymbol{x}) + \log p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) + \log p_\theta(\boldsymbol{y},\boldsymbol{z})\Big]\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{y}|\boldsymbol{x})}\left[-\log q_\phi(\boldsymbol{y}|\boldsymbol{x}) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y}) + \log p_\theta(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) + \log p_\theta(\boldsymbol{y},\boldsymbol{z})\Big]\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{y}|\boldsymbol{x})}\Big[-\log q_\phi(\boldsymbol{y}|\boldsymbol{x}) + \mathcal{L}(\phi, \theta, \boldsymbol{x}, \boldsymbol{y})\Big] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{y}|\boldsymbol{x})}\Big[\mathcal{L}(\phi, \theta, \boldsymbol{x}, \boldsymbol{y})\Big] + \mathbb{E}_{q_\phi(\boldsymbol{y}|\boldsymbol{x})}\Big[-\log q_\phi(\boldsymbol{y}|\boldsymbol{x})\Big] \qquad (3.19) \\
&= \sum_{\boldsymbol{y}} q_\phi(\boldsymbol{y}|\boldsymbol{x})\mathcal{L}(\phi, \theta, \boldsymbol{x}, \boldsymbol{y}) + \mathcal{H}(q_\phi(\boldsymbol{y}|\boldsymbol{x})), \qquad (3.20)
\end{aligned}$$

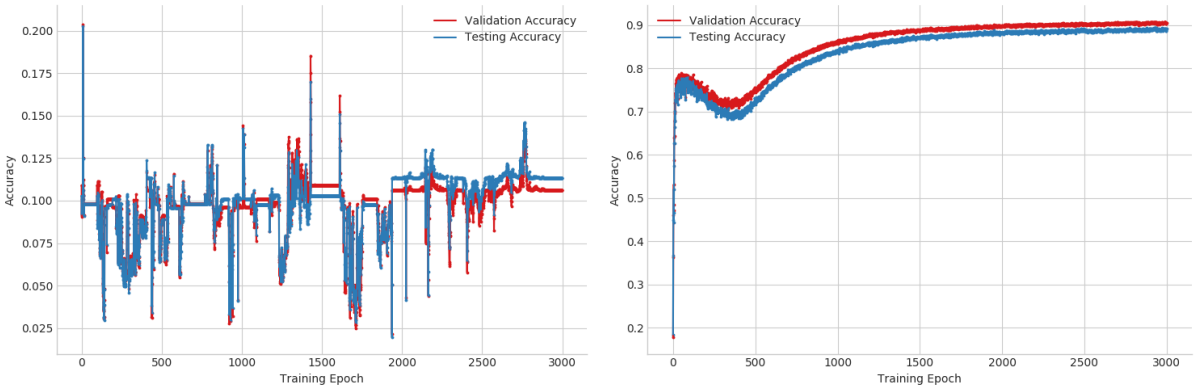
kde $\mathcal{H}(q_\phi(\boldsymbol{y}|\boldsymbol{x}))$ je informační entropie multinomického rozdělení $q_\phi(\boldsymbol{y}|\boldsymbol{x})$.

V praxi se běžně používá spíše verze (3.19), kde se navíc namísto střední hodnoty vezme pouze jeden vzorek a předpokládá se, že v kontextu celého minibatche se to bude chovat jako Monte Carlo vzorkování [28, 32]. Pro naše účely je to ale nevhodné, jelikož je klasifikátor na počátku tréninku velmi nepřesný a může se stát, že během pár úvodních kroků optimalizace kompletně zdiverguje, viz Obr. 3.9a.

Abychom se tomu vyhnuli, použijeme předpis pro střední hodnotou diskrétní náhodné veličiny $\boldsymbol{y}$

$$\mathbb{E}_{p(\boldsymbol{y})}\Big[f(\boldsymbol{y})\Big] = \sum_{\boldsymbol{y}} p(\boldsymbol{y}) f(\boldsymbol{y}). \qquad (3.21)$$

S jeho pomocí tak převedeme (3.19) na (3.20), kde místo jednoho vzorku počítáme vážený průměr přes všechny realizace $\boldsymbol{y}$. Pro každou třídu je tedy třeba vypočítat (3.18) zvlášť. Díky tomu sice roste doba tréninku lineárně s počtem tříd, ale také významně stoupá stabilita tréninku, viz Obr. 3.9.



(a) ELBO pouze s jedním vzorkem (3.19).

(b) ELBO s váženým průměrem (3.20).

Obr. 3.9: Porovnání přesnosti klasifikace pomocí SS VAE s váženým průměrem a bez něj na MNIST datasetu s 3000 označenými vzorky [32]. V případě 3.9b je použita pouze ztrátová funkce (3.22), tzn. bez přidané klasifikační chyby.

Ztrátová funkce pro celý dataset je tedy rovna

$$L(\boldsymbol{\theta}) = -\mathcal{L}(\phi, \theta, \mathbb{X}_s, \mathbb{Y}) - \mathcal{L}(\phi, \theta, \mathbb{X}_u) = -\sum_{(\mathbb{X}_s, \mathbb{Y})} \mathcal{L}(\phi, \theta, \boldsymbol{x}, \boldsymbol{y}) - \sum_{\mathbb{X}_u} \mathcal{L}(\phi, \theta, \boldsymbol{x}), \tag{3.22}$$

kde $\mathbb{X}_s$ (resp. $\mathbb{X}_u$) reprezentuje množinu označených (resp. neoznačených) pozorování a $\mathbb{Y}$ je množina labelů.

Distribuce $q_\phi(\boldsymbol{y}|\boldsymbol{x})$ slouží v modelu jako klasifikátor a je tudíž pro nás nejdůležitějším prvkem modelu. Bohužel ve ztrátové funkci (3.22) vystupuje pouze v druhém členu $\mathcal{L}(\phi, \theta, \mathbb{X}_u)$ a je tedy optimalizována jen pro neoznačená data. Abychom docílili toho, že se klasifikátor bude zlepšovat i v případě označených dat, rozšíříme funkci $L(\boldsymbol{\theta})$ o další člen odpovídající marginální věrohodnosti pro proměnnou $\boldsymbol{y}$ přes označená data [31].

Konečná verze ztrátové funkce pro tento model má tvar

$$L(\boldsymbol{\theta}, \alpha) = -\sum_{(\mathbb{X}_s, \mathbb{Y})} \mathcal{L}(\phi, \theta, \boldsymbol{x}, \boldsymbol{y}) - \sum_{\mathbb{X}_u} \mathcal{L}(\phi, \theta, \boldsymbol{x}) + \alpha \cdot \mathbb{E}_{\tilde{p}_s(\boldsymbol{x}, \boldsymbol{y})}\Big[-\log q_\phi(\boldsymbol{y}|\boldsymbol{x})\Big], \tag{3.23}$$

kde $\tilde{p}_s(\boldsymbol{x}, \boldsymbol{y})$ je empirická distribuce označených dat a parametr α slouží ke škálování členů ztrátové funkce. Parametr α budeme podle doporučení z [31] volit $\alpha = 0.1 \cdot N_s$, přičemž N_s je počet označených pozorování.

Škálování faktorem α provádíme protože klasifikační penalizace je vzhledem k rekonstrukční penalizaci příliš malá. Při optimalizaci pomocí označených dat by v opačném případě mohlo dojít k tomu, že gradienty přitékající z rekonstrukční penalizace by měly na optimalizaci větší vliv a nedocházelo by k učení klasifikační části modelu.

Kapitola 4

Experimenty a jejich vyhodnocení

4.1 Zdroje a předzpracování dat

Data použitá při našich experimentech pochází z databáze tokamaku COMPASS, konkrétně se jedná o časové průběhy záření H_α z několika výstřelů. Během každého výstřelu je záření měřeno fotonásobičem se vzorkovací frekvencí 2 MHz. Celý dataset se skládá z dvou částí, jimiž jsou označená data $\mathbb{X}_s$ a neoznačená data $\mathbb{X}_u$. K dispozici je 8 398 010 pozorování (20 výstřelů), k nímž jsou známy příslušné třídy a další databázi neoznačených pozorování, z nichž jsme vybrali dalších 20 výstřelů obsahujících 9 741 500 pozorování.

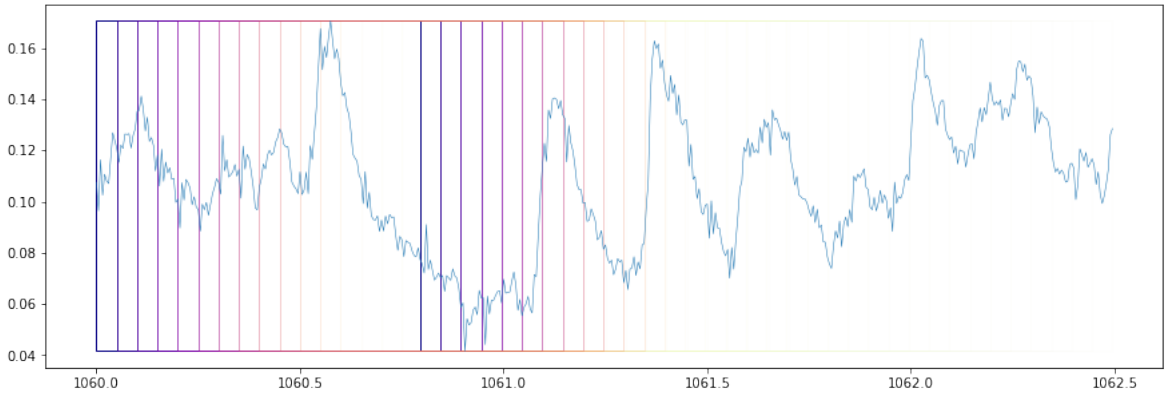
V bakalářské práci a výzkumnému úkolu jsme byli nuceni tento počet označených dat decimovat faktorem 100, tzn. brali jsem z původní uspořádané množiny $\mathbb{X} = \{x_n\}_{n=0}^{N}$ každé sté pozorování $\hat{\mathbb{X}} = \{x_{100\cdot n}\}_{n=0}^{\lfloor \frac{N}{100} \rfloor}$. Motivace k tomuto kroku plynula z požadavku na klasifikaci v reálném čase a schopnosti zpětnovazebné smyčky pracovat s pevně danou frekvencí 0.02 MHz. Navíc osm milionů vzorků by bylo pro modely Gradient Tree Boosting a SVM příliš náročné. Z dat, která po decimaci zbyla, jsme následně počítali charakteristiky derivace, klouzavý průměr, vážený klouzavý průměr atd. Maximální délka použitého okna byla 16 po sobě jdoucích prvků, což bez decimace odpovídá 1600 po sobě jdoucích prvků.

Neuronové sítě jsou na rozdíl od výše zmíněných metod po datech nesmírně hladové a s rostoucím počtem dat roste i jejich kvalita. Bohužel data z fotonásobiče jsou, jako každá jiná měření, zatížena šumem, který však můžeme decimací mírně potlačit. Nyní ale stačí pouze decimační faktor 10. Aby se výsledky této práce daly porovnat s těmi předchozími, rozhodli jsme se navíc využít stejné úseky signálů, na kterých byly počítány původní charakteristiky.

Toho docílíme tak, že z decimovaných výstřelů vytvoříme sekvence délky 160 s posunem 10 podle předpisu

$$\hat{\mathbb{X}} = \left\{(x_{10\cdot n}, x_{10\cdot n+1}), \cdots, x_{10\cdot n+159})\right\}_{n=0}^{\lfloor \frac{N}{10} \rfloor} = \{\boldsymbol{x}^n\}_{n=0}^{\lfloor \frac{N}{10} \rfloor}, \tag{4.1}$$

což odpovídá původním 16 prvkům s posunem 1. Dále je třeba těmto sekvencím přiřadit příslušnou třídu (H-mód, L-mód, ELM_start, ELM_konec). Celá sekvence bude označená stejnou třídou jako její prostřední pozorování, tzn. $(\hat{\mathbb{X}}, \boldsymbol{y}) = \{(x_{10\cdot n}, x_{10\cdot n+1}, \cdots, x_{10\cdot n+159}), y_{10\cdot n+80}\}_{n=0}^{\lfloor \frac{N}{10} \rfloor}$, kde skalárně značené třídy $y_{10\cdot n+80}$ převedeme na vektor $\boldsymbol{y}^n$ pomocí one hot encoding.



Obr. 4.1: Způsob vytváření sekvencí. Posunující okno dlouhé 160 s překryvem 10.

Výsledný dataset se tedy skládá z 83 680 označených sekvencí a 97 400 neoznačených, tzn. $\mathbb{X}_s = (\mathbb{X}, \mathbb{Y}) = \{\boldsymbol{x}^n, \boldsymbol{y}^n\}_{n=0}^{83680}$ a $\mathbb{X}_u = \{\boldsymbol{x}^n\}_{n=0}^{97400}$, kde $\boldsymbol{x}^n \in \mathbb{R}^{160}$ a $\boldsymbol{y}^n \in \left\{\boldsymbol{y} \in \{0,1\}^4 : \sum_{i=1}^{4} y_i = 1\right\}$ one hot encoding jednotlivých tříd.

Celé předzpracování je pak zakončeno přeškálováním celého datasetu pomocí RobustScale [33], který škáluje data podle kvantilového rozsahu (interquantile range: $IQR = Q_X(0.75) - Q_X(0.25)$) a je tudíž robustní vůči outlierům a ELMům. V minulosti se totiž ukázalo, že abnormálně vysoké ELMy mají špatný vliv na škálování datasetu.

Dataset $\mathbb{X}_s$ byl vzhledem k věrohodnému testování modelů rozdělen v poměru 8:2 na tréninkový a testovací část. V případě neuronových sítí jsme tréninkový dataset ještě jednou rozdělili v poměru 8:2, abychom získali validační část, kterou jsme pak používali při hledání hyperparametrů a pro kontrolu overfittingu. U ostatních modelů, které nejsou tak výpočetně náročné, jsme hyperparametry hledali pomocí křížové validace, kde jsme brali v úvahu pět foldů (částí).

Mezi hyperparametry neuronových sítí patří kromě míry učení, velikosti minibatche a typ optimalizačního algoritmu i jejich architektura. Ideální způsob hledání vhodné architektury by byl pravděpodobně Grid Search, kde by mřížku tvořily počty neuronů, počty vrstev a teoreticky i typy vrstev. Tento postup by však kvůli výpočetní náročnosti zabral každému modelu několik dní. Proto jsme nejlepší architekturu hledali heuristickou verzí metody největšího spádu.

Začali jsme s jednoduchou úvodní architekturou, pro kterou jsme provedli několik optimalizačních epoch a vyhodnotili ztrátovou funkci spolu s dalším metrikami na validačním datasetu. Poté jsme vytvořili dvě modifikace této architektury, kdy jedné jsme ubrali neurony a druhé jsme je přidali. Dále opět následovala krátká optimalizace a vyhodnocení. Pokud jedna z modifikovaných sítí překonala v kvalitě tu původní, přijali jsme ji za novou nejlepší a opakovali předchozí krok. Stejným způsobem jsme pak hledali i optimální počet vrstev.

Ve chvíli, kdy jsme našli naši nejlepší architekturu spolu s dalšími hyperparametry, jsme tréninkový a validační dataset opět sloučili do jednoho a natrénovali nejlepší model znovu. Tento model jsme pak konečně vyhodnotili na testovacím datasetu a výsledky uvedli v tabulkách níže.

Všechny metody byly implementovány v jazyce Python s využitím knihoven Pytorch [34], Pyro [35] a Scikit-learn [36]. Výpočty byly prováděny na stroji s grafickou kartou Nvidia GTX 1070 (8GB) a osmijádrovým procesorem AMD FX-8320.

4.2 Srovnání latentních a ručně vybraných příznaků

Jak již víme z předchozí kapitoly, v této práci budeme používat k extrakci příznaků z časových řad tři typy variačních autoecoderů, jimiž jsou standardní, konvoluční a rekurentní VAE. Pro každý z nich jsme pak hledali co nejlepší architekturu. Mírným zjednodušením hledání byla volba decoderů coby zrcadlových kopií encoderů, které se lišily pouze modifikací vstupní a výstupní vrstvy. Díky tomuto zjednodušení se počet případných hyperparametrů zmenšil na polovinu. Latentní prostor má však ve všech případech patnáct dimenzí. Číslo 15 bylo voleno v závislosti na předchozích pracích, kde bylo dosaženo nejlepších výsledků jen s pouhými patnácti příznaky, které byly předem vybrány pomocí metod na redukci dimenzionality a hodnocení užitečnosti příznaků.

4.2.1 Nejlepší modely

Nejlepší architekturou pro standardní VAE se ukázala jednoduchá verze s třívrstvým encoderem využívajícím ELU jako aktivační funkci.

Encoder

Dense(256) + ELU	
Dense(128) + ELU	
($\boldsymbol{\mu}$) Dense(15)	($\boldsymbol{\sigma}$) Dense(15) + Softplus

Decoder

Dense(128) + ELU	
Dense(256) + ELU	
($\widetilde{\boldsymbol{\mu}}$) Dense(160)	($\widetilde{\sigma}$) Dense(1) + Softplus

Tabulka 4.1: Nejlepší architektura pro standardní variační autoencoder (VAE). Dense(n) označuje obyčejnou dense vrstvu popsanou v části 2.1.1 s n výstupy.

Další v pořadí CVAE vyžadoval pro optimálnější rekonstrukci hlubší encoder, který se skládá z tří konvolučních vrstev, u kterých se postupně zvyšuje počet masek a naopak se zmenšuje jejich velikost, následovaných dvěma dense vrstvami. I přes svou hloubku se navíc CVAE trénuje výrazně rychleji než ostatní dva typy, přičemž k dosažení nižších hodnot ztrátové funkce stačí méně epoch, tzn. celý dataset nemusí projít optimalizací tolikrát. Průměrné trvání jedné epochy je pro všechny VAE vyneseno v tabulce 4.2.

model	čas [s]
VAE	17.65
CVAE	8.33
RVAE(h)	19.41
RVAE(h,C)	25.95

Tabulka 4.2: Průměrné časy trvání jedné tréninkové epochy u všech čtyř variačních autoencoderů.

Encoder

Conv1d(16, 6, 2) + ReLU	
Conv1d(32, 4, 2) + BatchNorm + ReLU	
Conv1d(64, 4, 2) + BatchNorm + ReLU	
Dense(256) + ReLU	
($\boldsymbol{\mu}$) Dense(15)	($\boldsymbol{\sigma}$) Dense(15) + Softplus

Decoder

Dense(256) + ReLU	
Dense(1152) + BatchNorm + ReLU	
ConvTransposed1d(32, 4, 2) + BatchNorm + ReLU	
ConvTransposed1d(16, 4, 2) + ReLU	
($\widetilde{\boldsymbol{\mu}}$) ConvTransposed1d(1, 6, 2)	($\widetilde{\sigma}$) Dense(1) + Softplus

Tabulka 4.3: Nejlepší architektura pro konvoluční variační autoencoder (CVAE). Dense(n) označuje dense vrstvu s n výstupy, Conv1d(n,f,p) označuje 1D konvoluční vrstvu s počtem filtrů n, velkostí jádra f a posunem p. ConvTransposed1d je transponovaná verze Conv1d.

Při použití mnohem hlubších encoderů typu ResNet [37] sice dochází k mírnému zlepšení rekonstrukčních vlastností, které jsou ovšem vykoupeny mnohonásobně větším počtem parametrů a delší dobou tréninku. Toto zlepšení však není tak výrazné, proto jsme zůstali mělčí verze, viz Tab. 4.3.

Časově nejnáročnější ze všech modelů je jednoznačně RVAE, i přestože námi vybraný model nemá velké množství skrytých vrstev, viz. Tab. 4.4. Model je jsme volili mělký, protože přidání dalších vrstev doprovázelo výrazné navýšení času potřebnému k optimalizaci, přičemž kvalita modelu se nejen nezvyšovala, ale začínala degradovat. Pro zrychlení výpočtu začínala jeho optimalizace s mírou učení 0.001. Po 300 epochách jsme míru učení snížili na 0.0001, abychom potlačili zvětšující se oscilace hodnot ztrátové funkce.

Encoder

LSTM(128)	
($\boldsymbol{\mu}$) Dense(15)	($\boldsymbol{\sigma}$) Dense(15) + Softplus

Decoder

Dense(128)	
LSTM(128)	
($\widetilde{\boldsymbol{\mu}}$) Dense(160)	($\widetilde{\sigma}$) Dense(1) + Softplus

Tabulka 4.4: Nejlepší architektura pro rekurentní variační autoencoder RVAE(h). Dense(n) (resp. LSTM(n)) označuje dense (resp. LSTM) vrstvu s n výstupy. Architektura modelu RVAE(h,C) má v první dense vrstvě decoderu 256 neuronů namísto 128.

4.2.2 Dosažené výsledky

Finální dosažené hodnoty ztrátové funkce nejlepších modelů vyhodnocené na testovacím datasetu jsou vyneseny v tabulce 4.5.

Typ	hyperparametry modelů	ztráta	počet epoch
VAE	optim: Adam, lr: 0.0001, bs: 100	-216.207	400
CVAE	optim: Adam, lr: 0.0001, bs: 500	**−241.288**	100
RVAE(h)	optim: Adam, lr: 0.001,0.0001, bs: 300	-118.279	400
RVAE(h,C)	optim: Adam, lr: 0.001,0.0001, bs: 200	-139.370	400

Tabulka 4.5: Dosažené hodnoty ztrátové funkce na testovacím datsetu. Mezi hyperparametry optimalizace patří optimalizační algoritmus *optim*, míra učení *lr* a velikost minibatche *bs*. U RVAE jsou uvedeny dvě hodnoty míry učení, protože pro zrychlení optimalizace se byla ta první po 150 epochách snížena.

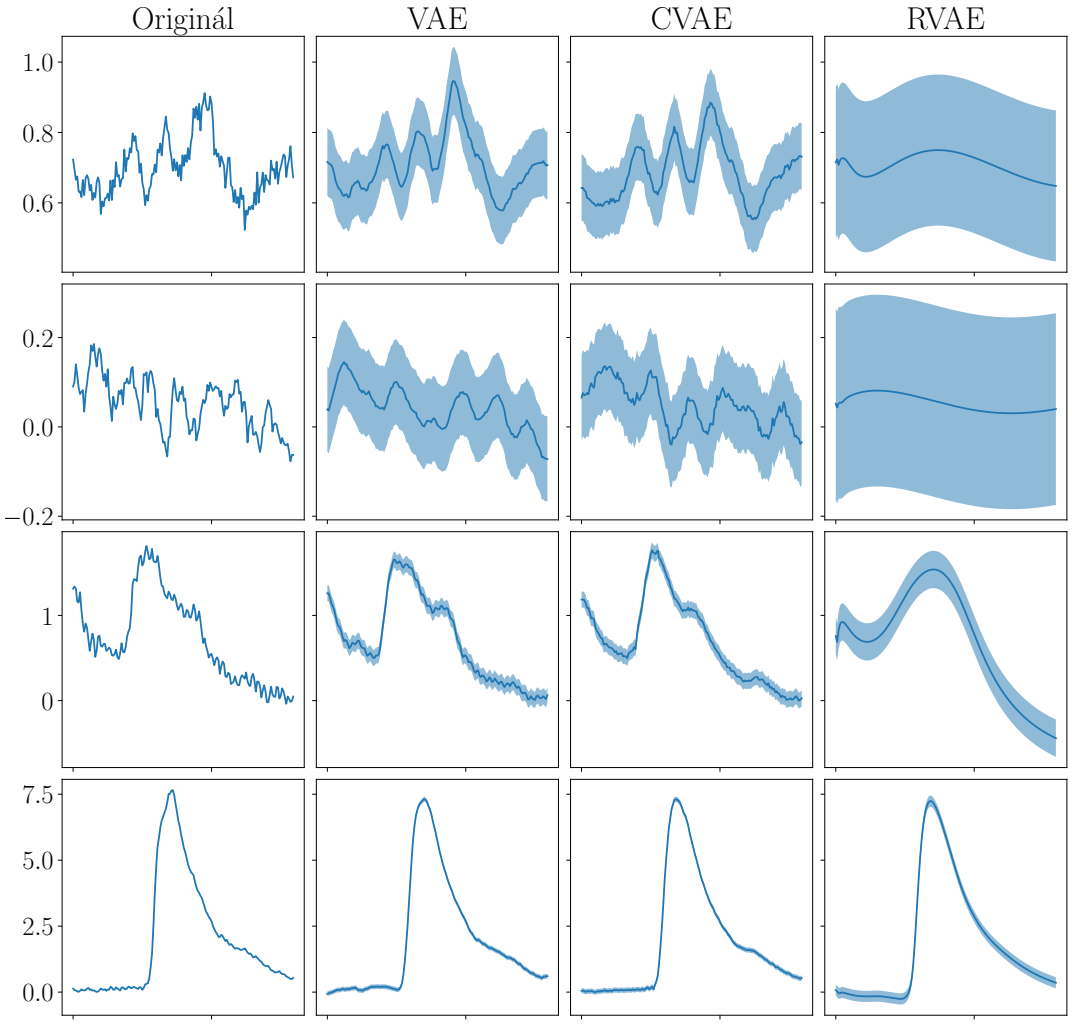
Z tabulky je patrné, že nejnižší ztráty dosáhl konvoluční varaiční autoencoder a dokonce mu k tomu stačilo i nejméně epoch. Jeho optimalizace bylo zastaveno už po 100 epochách, protože nadále nedocházelo k významnějšímu poklesu hodnot ztrátové funkce.

Nejhorší rekonstrukční chyba náleží rekurentnímu variačnímu autoencoderu využívajícího pouze $\boldsymbol{h}$, kterému trvalo 150 epoch, než jeho ztráta vůbec začala klesat a současně ani rychlost poklesu nebyla příliš velká. RVAE(h,C) naproti tomu začalo klesat mnohem rychleji a dosáhlo také nižší ztráty než RVAE(h), ale ani tak kvalitou nekonkuruje VAE a CVAE. Vzhledem k dosažené ztrátě na testovacím datasetu jsem se rozhodli nadále využívat pouze verzi RVAE(h,C) (označovanou pouze jako RVAE).

Standardní variační autoencoder v tomto případě představuje kompromis, neboť nedosahuje kvality konvoluční verze, ale jeho rekonstrukční schopnosti výrazně převyšují oba RVAE. Pro lepší představu toho, jak se jednotlivým typům daří rekonstruovat vstupní sekvence, jsme náhodně vybrali čtyři sekvence, jejichž rekonstrukce jsou zobrazeny na Obr. 4.2.

Při první pohledu na tento obrázek se zdá, že rekonstrukce získaná z VAE se příliš neliší od CVAE. Při bližším zkoumání je ale vidět, že CVAE je opravdu více přesnější, viz rekonstrukce druhé sekvence. U RVAE se navíc potvrdilo, jak již předem napovídala hodnota ztrátové funkce, že má ze všech nejhorší rekonstrukční vlastnosti. Většinu sekvencí odhaduje přibližně střední hodnotou s obrovskou směrodatnou odchylkou.

Jediné případy, které se přesvědčivě naučil rekonstruovat, jsou vysoké peaky příslušné EL-Mům (poslední sekvence na obrázku). Proč se naučil právě tento scénář tak přesně, je pravděpodobně důsledkem největších penalizací. Když se podíváme na škálu osy y na všech vybraných signálech uvidíme, že první tři mají rozdíl mezi maximem a minimem řádech desetin a přibližně oscilují kolem střední hodnoty. Pokud jsou tedy odhadnuty střední hodnotou, jejich příspěvek ke ztrátě není moc velký. Naopak poslední z nich by při stejném typu odhadu zaznamenal obrovskou ztrátu, která převáží chybu u předchozích odhadů a model tak je nucen se nejprve vypořádat s takovými sekvencemi.

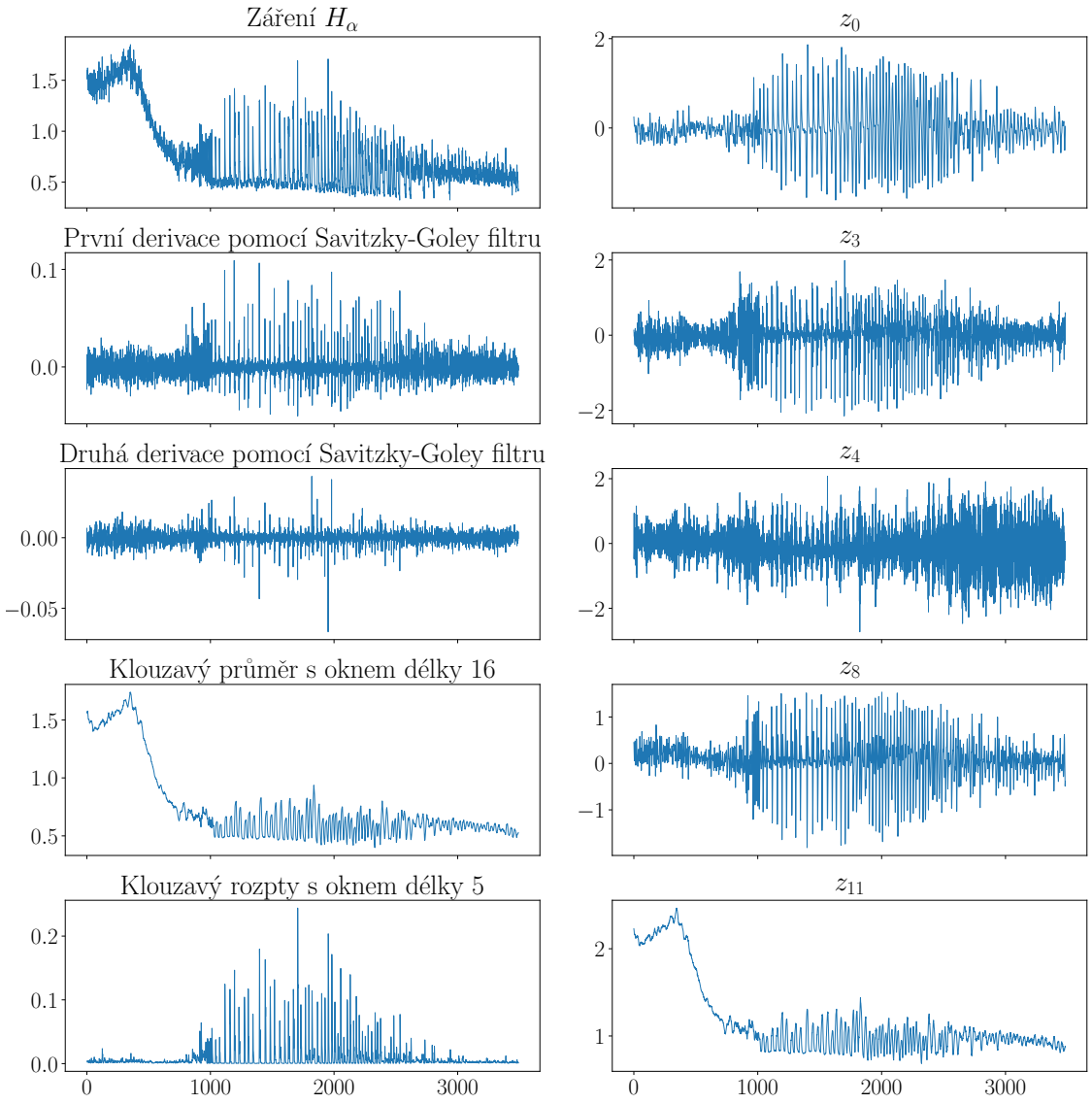


Obr. 4.2: Porovnání rekonstrukčních vlastností jednotlivých variačních autencoderů. Tučná modrá čára je odhadnutá střední hodnota $\widetilde{\boldsymbol{\mu}}$ a světlejší modrý pruh okolo ní odpovídá intervalu $(\widetilde{\boldsymbol{\mu}} - 2 \cdot \widetilde{\sigma}, \widetilde{\boldsymbol{\mu}} + 2 \cdot \widetilde{\sigma})$.

4.2.3 Extrahované příznaky

I když nás rekonstrukční vlastnosti, resp. decoder primárně nezajímají, můžou nám pomoci přibližně odhalit to, jaký význam mají jednotlivé elementy latentního kódu, tzn. extrahovaných příznaků.

V předchozí práci jsme jako příznaky používali charakteristiky signálu jako jsou první a druhá derivace, klouzavý průměr, exponenciálně tlumený klouzavý průměr a klouzavý rozptyl viz Obr. 4.3. Pomocí těchto příznaků jsme se snažili odhalit různé náhlé i pozvolné změny v průběhu signálu H_α, čímž jsme se snažili napodobit způsob, jakým o režimech udržitelnosti plazmatu rozhoduje člověk.

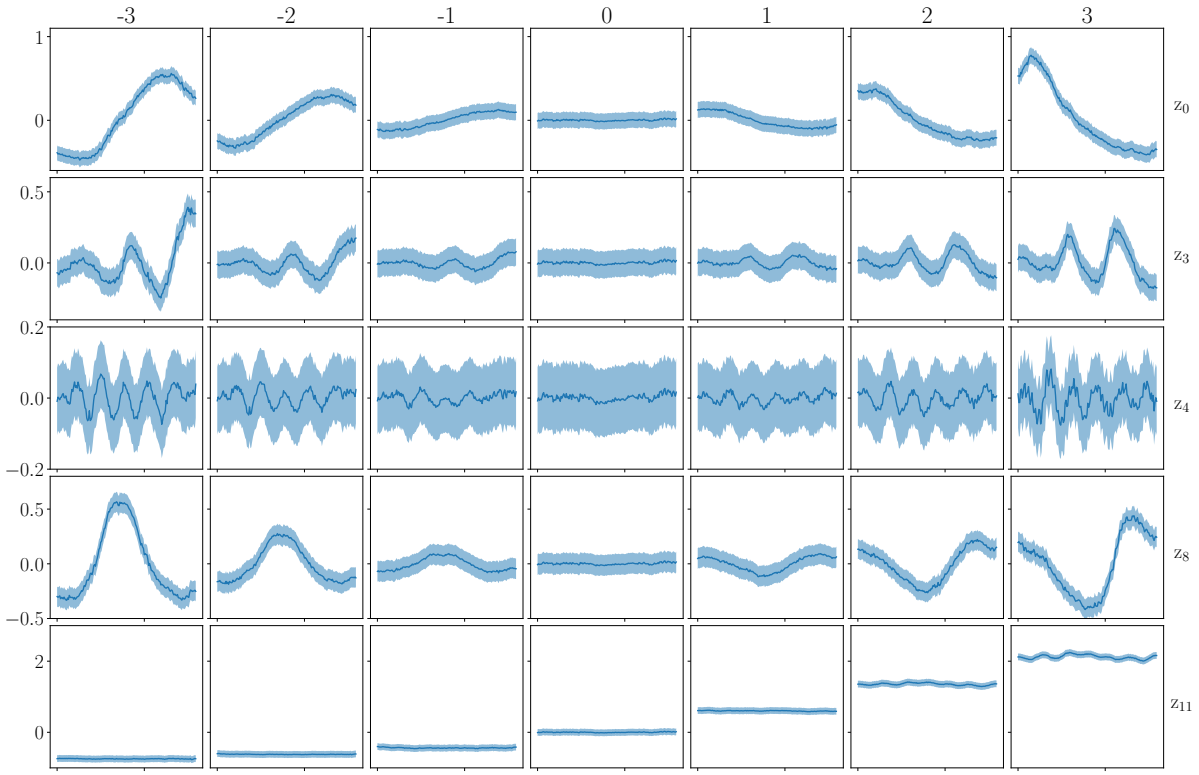


Obr. 4.3: Porovnání několika původních (levý sloupec) a nových (pravý sloupec) příznaků. Nové extrahované příznaky z pravého sloupce pocházejí z latentního prostoru konvolučního variačního autoencoderu.

V současném přístupu, kdy jsou příznaky ze sekvencí extrahovány pomocí encoderu, nedokážeme většinou přesně zjistit, co jednotlivé prvky latentního kódu představují. Výrazná podobnost mezi jedenáctým prvkem latentního kódu CVAE a klouzavého průměru v Obr 4.3 je zajímavý ovšem výjimečný případ, který se u ostatních prvků neopakuje. Přibližnou představu o ostatních můžeme získat při analýze decoderu. Jak už víme z předchozí kapitoly, decoder představuje skutečnou apriorní hustotu pravděpodobnosti pro naměřené sekvence $\boldsymbol{x}$ závislou na latentní reprezentaci $\boldsymbol{z}$, tzn. $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Díky tomu můžeme pozorovat změnu rekonstrukce způsobenou změnou vektoru $\boldsymbol{z}$.

No Obr. 4.4 jsou vyobrazeny změny rekonstrukce při změně pouze jedné hodnoty v latentním kódu. V tomto konkrétním případě jsme použili natrénovaný decoder z modelu CVAE. Výchozí vstupní vektor $\boldsymbol{z}$ je tvořen samými nulami (prostřední sloupec). V řádcích jsou pak zobrazeny

jednotlivé rekonstrukce, kdy jsou za jeden vybraný element latentního kódu dosazeny postupně hodnoty $\{-3, -2, -1,\ 0,\ 1,\ 2,\ 3\}$. Volba těchto hodnot je motivovaná předpokladem, že elementy latentního kódu mají normální rozdělení $\mathcal{N}(0, 1)$. V tomto konkrétním případě je z obrázku patrné, že např. první složka vektoru $\boldsymbol{z}$ ovlivňuje náklon, devátá složka určuje velikost a orientaci peaku uprostřed sekvence a dvanáctá složka posouvá signál po ose y.



Obr. 4.4: Vliv změny latentní proměnné na výstup z decoderu.

Později jsme zkoušeli i variační autoencodery s vyšší dimenzí latentního prostoru, ale zlepšení jejich rekonstrukce bylo zanedbatelné. Docházelo totiž jen k tomu, že se modely začaly zaměřovat na zpřesnění odhadu šumu, kterým jsou naše data zatížená. Na základě toho lze konstatovat, že patnácti rozměrná reprezentace je pro naše sekvence dostačující.

4.3 Klasifikace

Nyní se již zaměříme na klasifikaci jednotlivých režimů udržení plazmatu. Tyto stavy jsou čtyři a jedná se o H-mod, L-mod, začátek ELMu a konec ELMu. Klasifikovat je budeme nejprve s pomocí původních ručně extrahovaných příznaků. Poté využijeme latentní prostor (resp. kód) získaný z variačních autoencoderů. A nakonec provedeme klasifikaci stavů přímo na signálu H_α pomocí neuronové sítě, kterou se navíc pokusíme vylepšit s použitím semi-supervised variačního autoencoder.

4.3.1 S původními příznaky

V předchozí práci jsme ukázali, že k relativně dobré přesnosti klasifikace stačí pouze patnáct příznaků. Tyto příznaky byly vybrány pomocí metody využívající k ohodnocení příznaků tzv. *důležitost příznaků*, která je vedlejším produktem klasifikace pomocí modelů, jež obsahují rozhodovací stromy. Vybrané příznaky jsou samotné záření H_α, 1. a 2. derivace pomocí Savitzky-Golay filtru, klouzavé průměry se zpožděními 400 a 800 μs, exponenciálně tlumené klouzavé průměry zpožděné 200, 300, 400 μs a nakonec klouzavé rozptyly se zpožděními 400, 450, 500, 550, 600, 650 a 800 μs.

Jako klasifikátory jsme použili Support Vector Machine (SVM), Gradient Tree Boosting (GB) a jednoduchou dopřednou neuronovou síť (Tabulka 4.6), jejíž architekturu jsem vybírali stejným postupem jako u variačních autoencoderů. U této neuronové sítě (NN) jsme poprvé museli použít regularizační techniku zvanou dropout, která pro každý minibatch deaktivuje jednotlivé neurony uvnitř neuronové sítě s pravděpodobností p. Výstupní hodnota takovéhoto deaktivovaného neuronu je pak rovna nule. Dropout se využívá k částečnému potlačení efektu přetrénování, kdy neuronová síť dosahuje velmi vysokých přesností na trénovacím datasetu, ale testovacím datasetu je přesnosti výrazně nižší.

Dense(128) + Dropout(0.3) + Sigmoid
Dense(128) + Dropout(0.3) + Sigmoid
Dense(4) + Softmax

Tabulka 4.6: Nejlepší architektura neuronové sítě pro klasifikaci s původními příznaky. Dense(n) označuje dense vrstvu s n výstupy a Dropout(p) označuje dropout regularizaci aplikovanou s pravděpodobností p.

V tabulce 4.7 jsou pak vyneseny průměrné F-míry a přesnosti spočítané na testovacím datasetu pro jednotlivé modely.

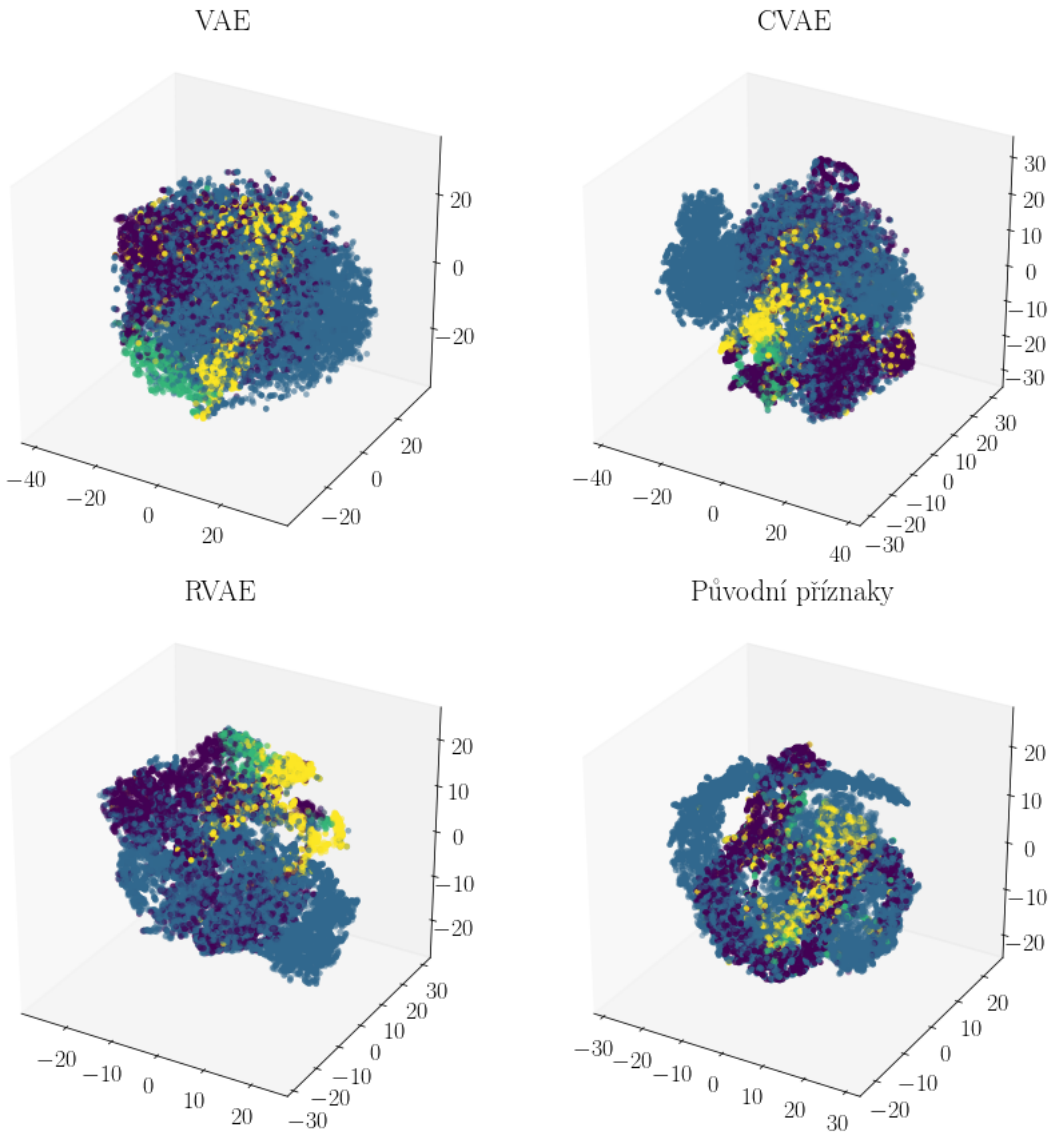
model	hyperparametry klasifikátorů	průměrná F-míra	přesnost
GB	n_st: 1000, lr: 0.1	0.793	0.879
SVM	kernel: RBF, C: 50, γ: 0.07	0.780	0.867
NN	Tabulka 4.6, optim: Adam, lr: 0.001, bs: 500	**0.827**	**0.880**

Tabulka 4.7: Výsledky klasifikace s patnácti původními příznaky. Průměrná F-míra představuje průměr z F-mír jednotlivých stavů. Trénink této neuronové sítě probíhal 200 epoch. Zkratka *n_st* označuje počet stromů , *lr* představuje míru učení (learning rate), *optim* značí optimalizační algoritmus a *bs* je velikost minibatche.

Z tabulky je patrné, že nejlepší průměrná F-míra byla dosažena neuronovou sítí po dvou stech epochách. Druhý v pořadí je Gradient Tree Boosting obsahující 1000 stromů, které byly trénované s mírou učení rovnou 0.1 . Na posledním místě je Support Vector Machine, jehož jádro tvoří radiálně bazické funkce s parametrem $\gamma = 0.07$ a penalizací $C = 50$. Parametry pro GB a SVM byly nalezeny metodou Grid Search (tabulky B.2 a B.1) s využitím křížové validace. Parametry nalezené v předchozí práci zde nešlo použít, protože se jedná o odlišná data.

4.3.2 Na latentních prostorech

Variační autoencodery spadají mezi modely trénované bez učitele (unsupervised learning), neboť při tréninku nepotřebují označená data. Jak jsem ukázali v předchozí podkapitole, VAE modely jsou schopny zredukovat vstupní data na latentní reprezentaci s mnohem nižší dimenzí, ze které jsou schopny zpětně tyto data zrekonstruovat. Na Obr. 4.4 jednotlivé složky latentního kódu charakterizují vlastnosti vstupních dat. Díky tomu můžeme využít část těchto modelů, konkrétně encodery, jako prostředky k redukci dimenzionality vstupních dat (ze 160 dimenzí na 15 dimenzí) a extrakci příznaků. Variační autoencodery jsme měli již předtrénované, proto stačilo hledat jen vhodný klasifikátor.



Obr. 4.5: Projekce latentních prostorů a původních příznaků do 3D s pomocí T-SNE. Barvy odpovídají jednotlivým stavů. Modrá značí L-mód, fialová H-mód, žlutá ELM_start a zelená je ELM_konec.

4.3.2.1 Zafixovaný encoder

Při použití modelů Support Vector Machine a Gradient Tree Boosting jsme si celý dataset předem encodovali stejně, jako bychom to udělali v případě jakéhokoliv jiného způsobu předzpracování. U klasifikace pomocí neuronové sítě se však vyplatí encodovat jednotlivé minibatche během tréninku. Můžeme tak využít vzorkování z aproximativní aposteriorní distribuce $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ tzn. encoderu, které se chová jako regularizace proti přetrénování a není tak třeba do klasifikátoru navíc přidávat např. dropout. Abychom byly schopni encoder takto využít musíme ho přímo napojit na klasifikátor a zafixovat ho, tj. zajistit, aby jeho váhy nepodléhaly další optimalizaci spolu s váhami klasifikátoru. V tuto chvíli je to nutné udělat, jinak by nebylo korektní porovnávat jednotlivé klasifikátory mezi sebou.

Vyzkoušeli jsme kombinace všech typů encoderů a klasifikačních modelů viz tabulka 4.9, přičemž parametry klasifikátorů jsme hledali stejně jako v předchozích případech. Pro všechny tři encodery se jako nejlepší architektura klasifikátoru ukázala třívrstvá neuronová síť, viz tabulka 4.8.

Dense(64) + Sigmoid
Dense(128) + Sigmoid
Dense(4) + Softmax

Tabulka 4.8: Nejlepší architektura neuronové sítě vhodná pro klasifikaci na latentním prostoru. Dense(n) označuje dense vrstvu s n výstupy.

model	hyperparametry klasifikátoru	F_1	přesnost
VAE + GB	n_st: 500, lr: 0.1	0.817	0.857
VAE + SVM	kernel: RBF, C: 10, γ: 0.07	0.803	0.842
VAE + NN (sv)	Tabulka 4.8, optim: Adam, lr: 0.01/0.001, bs:500	0.737	0.777
CVAE + GB	n_st: 1000, lr: 0.1	0.816	0.856
CVAE + SVM	kernel: RBF, C: 1, γ: 0.1	0.807	0.845
CVAE + NN (sv)	Tabulka 4.8, optim: Adam, lr: 0.01/0.001, bs:500	**0.833**	**0.861**
RVAE + GB	n_st: 1000, lr: 0.1	0.793	0.832
RVAE + SVM	kernel: RBF, C: 100, γ: 0.28	0.804	0.838
RVAE + NN (sv)	Tabulka 4.8, optim: Adam, lr: 0.01/0.001, bs:500	0.801	0.828

Tabulka 4.9: Výsledky klasifikace pro všechny model využívající zafixovaný encoder k redukci dimenzionality vstupu. F_1 je průměrná F-míra. Zkratka *n_st* označuje počet stromů, *lr* značí míru učení (learning rate), *optim* značí optimalizační algoritmus, *bs* je velikost minibatche a *sv* představuje "se vzorkováním". Optimalizace byla inicializována s první mírou učení a po N epochách byla snížena na druhou hodnotu pro přesnější konvergenci.

Nejlepším kombinací pro klasifikaci se ukázalo spojení konvolučního encoderu a neuronové sítě, jež dosáhly průměrné F-míry 83.3% a překonaly tím nejlepší výsledek z tabulky 4.7. Na základě toho můžeme konstatovat, že příznaky extrahované variačním autoencoderem jsou minimálně stejně tak dobré jako ty původní, které byly vytvořené ručně, přesně na míru tomuto problému.

Překvapením je, že ve spojení s neuronovou sítí, jako klasifikátorem, překonal rekurentní encoder ten standardní, i přestože měl mnohem horší rekonstrukční schopnosti (viz tabulka 4.5).

Nejkonzistentnější klasifikátor se ukázal SVM, který sice nedosahuje nejlepších výsledků, ale má u všech typů encoderů přibližně stejné hodnoty průměrné F-míry.

4.3.2.2 Nezafixovaný encoder

Nyní jsme se rozhodli jít o krok dál a odfixovat encodery po několika optimalizačních epochách. Klasifikátory by tak díky tomu měly dosahovat lepších výsledků, než se zafixovaným encoderem. Při tomto postupu nelze použít SVM a GB jako klasifikátory, protože nevyužívají gradientní optimalizace, tzn. nepřitékaly by z nich do encoderu gradienty, pomocí kterých by se dále optimalizovaly jeho vnitřní parametry.

Dalším rozdílem oproti předchozímu postupu je opět nutnost přidat dropout jako regularizaci, protože budeme sice dále vzorkovat, ale už nám nadále nebude KL divergence vynucovat podobnost mezi výstupem encoderu a normálním rozdělením. KL divergenci v tomto případě jako regularizaci použít nelze, neboť její hodnoty jsou příliš vysoké a převážila by klasifikační chybu.

Nejlepší architektura klasifikátoru, který pro všechny typy encoderů vyšla stejně, je uvedena v tabulce 4.10 a výsledky klasifikace jsou vyneseny v tabulce 4.11.

Dense(128) + Dropout(0.3) + Sigmoid
Dense(128) + Dropout(0.3) + Sigmoid
Dense(4) + Softmax

Tabulka 4.10: Nejlepší architektura neuronové sítě vhodná pro klasifikaci na latentním prostoru. Dense(n) označuje dense vrstvu s n výstupy a Dropout(p) označuje dropout regularizaci aplikovanou s pravděpodobností p.

model	hyperparametry modelů	průměrná F-míra	přesnost
VAE + NN	optim: Adam, lr: 0.01/0.001, bs: 500	0.820	0.856
CVAE + NN	optim: Adam, lr: 0.01/0.001, bs: 500	**0.869**	**0.912**
RVAE + NN	optim: Adam, lr: 0.01/0.001, bs: 500	0.859	0.889

Tabulka 4.11: Výsledky klasifikace při spojení nezafixovaného encoderu a neuronové sítě. Mezi hyperparametry optimalizace patří optimalizační algoritmus *optim*, míra učení *lr* a velikost minibatche *bs*. Dvě hodnoty u míry učení značí, že se nejprve začínalo s vyšší hodnotou pro rychlejší sestup a následně byla míra učení snížena k potlačení oscilací a zpřesnění.

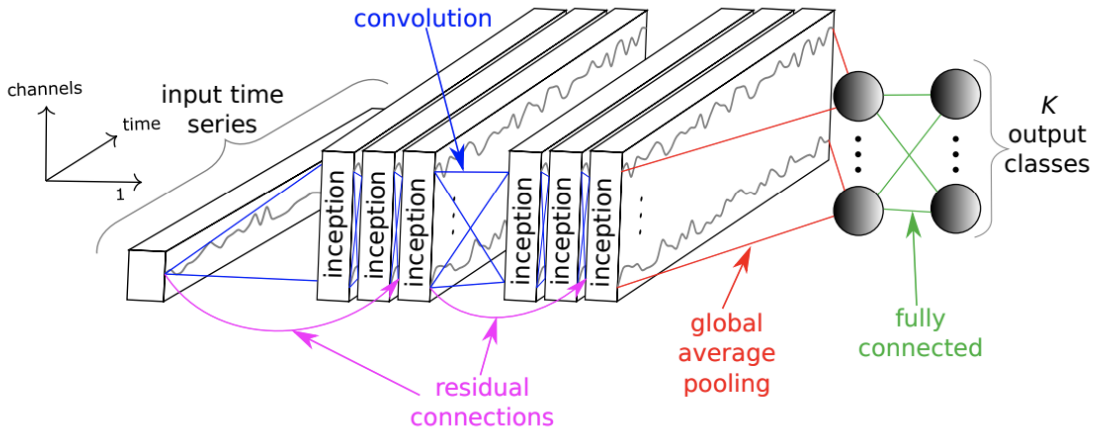
Při porovnání tabulky 4.9 a tabulky 4.11 si můžeme všimnout, že kromě jiného, zde došlo k výraznému zlepšení kvality klasifikátoru využívajícímu standardní encoder. Ten je sice stále nejhorší v porovnání s ostatními, ale jeho průměrná F-míra vzrostla o 8%. Druhý se umístil opět rekurentní encoder, který nyní zaostává za konvolučním encoderem jen o 1%.

Nejlepších výsledků dosáhl opět klasifikátor kombinovaný s konvolučním encoderem. Ten v současnosti překonává v kvalitě, jak původní příznaky tak i zafixovaný encoder. V tuto chvíli však už nelze encoder chápat jako techniku pro předzpracování data, ale spíše jako předtrénování části klasifikační neuronové sítě, která extrahuje příznaky z dat. Stejného principu se využívá, např. u přenosu učení (transfer learning) [38].

4.3.3 S pomocí semi-supervised VAE na H_α

V posledním experimentu jsme se trochu odchýlili od myšlenky prosté extrakce a spojením nezafixovaného encodéru s klasifikátorem nám vznikla hluboká klasifikační neuronová síť, jejíž váhy byly částečně předtrénovány variačním autoencoderem. Proto se o tom dá hovořit jako o klasifikaci přímo na signálu H_α.

Jelikož bylo dosaženo velmi dobrých výsledků, rozhodli jsme se vyzkoušet další dva modely navržené přímo pro klasifikaci časových řad. První z nich je velmi hluboká konvoluční síť InceptionTime [39], která na některých datasetech drží rekordní přesnost.



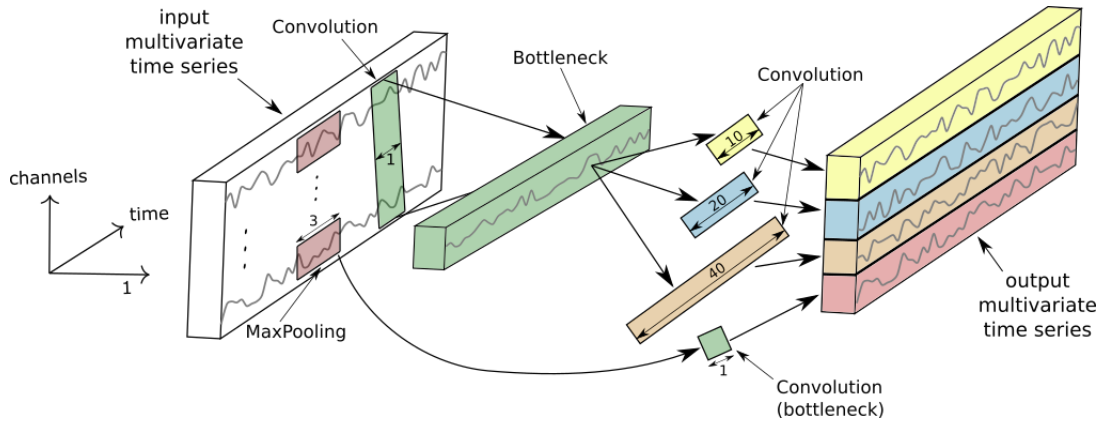
Obr. 4.6: Model InceptionTime [39].

Druhý model je konvoluční-rekurentní neuronová síť (CRNN), jež kombinuje výhody konvolučních a rekurentních neuronových sítí. Její architektura je zaznamenána v tabulce B.3.

model	průměrná F-míra	přesnost
InceptionTime	0.913	0.949
CRNN	0.901	0.939

Tabulka 4.12: Výsledky klasifikace přímo na H_α.

Z tabulky 4.12 je patrné, že obě tyto sítě překonaly předchozí metody. Zdá se tak, že je téměř zbytečné vytvářet příznaky na míru, nebo je jakkoliv z dat extrahovat. To však zase není úplně pravda. Díky tomu, že jsou VAE modely trénované bez učitele, tzn. nepotřebují označená data, můžeme využít mnohem větší množství dat, které máme k dispozici. Proto by měly být příznaky získané pomocí VAE mnohem robustnější než ty, jež si interně extrahují výše zmíněné sítě. Ty byly totiž trénovány pouze na velice limitované množině označených vzorků a proto obsahují několik regularizačních technik, jako je například zúžení (bottleneck) na začátku každé *Inception* buňky, viz Obr. 4.7.



Obr. 4.7: Inception vrstva složená z pěti konvolučních vrstev s různými velikostmi masek. Vstupní vícerozměrný signál je zde redukován pomocí *bottlenecku* (konvoluční vrstvy s jádrem velikosti jedna) na nižší dimenzi rovnu počtu použitých konvolučních masek [39].

My však můžeme generativní model natrénovat i na neoznačených datech s použitím semi-supervised variačního autoencoderu. Tréninkem na označených i neoznačených datech by se měla lehce zlepšit kvalita klasifikátoru, který by měl být navíc i robustnější vůči, do této chvíle, neznámým pozorováním.

Jako encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$ a decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{y})$ byl použit podle dosavadních výsledků nejlepší CVAE, který se od tabulky. 4.3 liší přidaným vstupem $\boldsymbol{y}$. Ten však nevstupuje do encoderu přes konvoluční vrstvy, ale je napojen až na první dense vrstvu.

K optimalizaci těchto nových modelů sloužil stejně jako v předchozích případech Adam s konstantní mírou učení 0.0001. Jeden minibatch byl tvořen 512 vzorky.

model	průměrná F-míra	přesnost
InceptionTime + SSVAE	**0.941**	**0.977**
CRNN + SSVAE	0.903	0.937

Tabulka 4.13: Výsledky klasifikace přímo na H_α s využitím SSVAE.

Při pohledu do tabulky 4.13 je vidět, že průměrná F-míra modelu CRNN se oproti výsledkům z 4.12 zlepšila jen o 0.2% a přesnost mírně klesla na 93.7%. Mnohem zajímavější je výsledek modelu InceptionTime, který se s použitím SSVAE výrazně zlepšil. Přesnost klasifikace tohoto modelu se na testovacím datasetu blíží 98% a průměrná F-míra dosahuje 94.1%, což je zatím nejlepší výsledek, kterého bylo dosaženo.

Závěr

Po stručném popisu problematiky tokamaků a problematiky udržení termojaderné fúzní reakce jsme podrobně prošli základní principy neuronových sítí, jako jsou aktivační funkce, typy vrstev, ztrátové funkce a optimalizační algoritmy, které slouží k jejich trénování. Na to jsme navázali teorií vztahující se k variačnímu autoencoderu, kterou jsme poté zpětně propojili s neuronovými sítěmi. Nakonec jsme zavedli semi-supervised variační autoencoder, který se používá ke zlepšení kvality klasifikace pomocí neoznačených dat.

Cílem této práce bylo využít generativní modely pro klasifikaci režimů udržitelnosti plazmatu v tokamaku COMPASS. Za tímto účelem jsme navrhli tři typy variačních autoencoderů (konvoluční, rekurentní a standardní), jejichž nízkodimenzionální latentní reprezentaci jsme chtěli využít jako příznaky pro navazující klasifikační modely. Porovnáním jejich rekonstrukčních schopností jsme zjistili, že se jako nejlepší jeví konvoluční verze (CVAE), která dosáhla nejnižší ztráty, přičemž jí k tomu stačilo nejméně optimalizačních epoch, viz tabulka 4.5.

Při podrobnější analýze extrahovaných příznaků z tohoto encoderu jsme se dále dozvěděli, že minimálně v jednom případě extrahovala pětivrstvá neuronová síť stejný příznak, který byl v minulosti navržen na míru tomuto klasifikačnímu problému (Obr. 4.3). Následná klasifikace na latentních prostorech se zafixovanými encodery potvrdila, že extrahované příznaky jsou stejně dobré pro klasifikaci, jako ty dříve ručně vytvořené (tabulky. 4.7 a 4.9).

Za účelem zlepšení kvality klasifikace jsme dále odfixovali encodery a nechali klasifikační neuronovou síť, aby je během své optimalizace přizpůsobila pomocí protékajících gradientů. Výsledkem bylo navýšení kvality klasifikace u všech tří encoderů. Nejlepší se opět ukázalo spojení konvolučního encoderu s neuronovou sítí, kde průměrná F-míra dosáhla 86.9% a přesnost 91.2%. Na druhém místě skončil rekurentní encoder s průměrnou F-mírou 85.9% a přesností 88.9%.

Vzhledem k tomu, že nezafixované encodery nelze již nadále považovat za techniku předzpracování, ale spíše jako způsob, jak přetrénovat váhy neuronové sítě, rozhodli jsme se je porovnat s hlubokými klasifikačními neuronovými sítěmi jako jsou InceptionTime a CRNN. Tyto sítě později překonaly všechny předchozí postupy obsahující extrakci příznaků a redukci dimenzionality (tabulka 4.12), přestože byly trénovány jen na označených datech.

Díky úspěchu těchto dvou modelů jsme se rozhodli je zapojit jako klasifikátory do SSVAE a natrénovat je na všech, tzn. i neoznačených datech, kterých je k dispozici mnohem více. Kromě toho, že jsou oba modely nyní robustnější vůči novým neznám pozorováním, se změnila i jejich kvalita. V případě CRNN nebyla změna příliš výrazná. Jeho průměrná F-míra vzrostla jen o 0.2% a přesnost klesla na 93.7%. Mnohem výraznější byla změna u modelu InceptionTime, kdy jeho průměrná F-míra dosáhla 94.1% a přesnost vzrostla až na 97.7%. Toto je nejlepší výsledek, jakého jsme se s využitím H_α kdy dosáhli.

Tento výsledek bohužel nelze přímo porovnat s již existujícími postupy [6, 5], jelikož využívají dodatečné signály, které nemáme k dispozici. Navíc jejich implementace a adaptace přímo na náš dataset by byla nad rámec této práce.

Bibliografie

[1] R.J. Goldston a P.H. Rutherford. *Introduction to Plasma Physics*. CRC Press, 1995. ISBN: 9781439822074. URL: https://books.google.it/books?id=7kM7yEFUGnAC.

[2] Alexandra Vallet. "Hydrodynamic modelling of the shock ignition scheme for inertial confinement fusion". Dis. Lis. 2014.

[3] Radomir Panek et al. "Reinstallation of the COMPASS-D tokamak in IPP ASCR". In: *Czechoslovak Journal of Physics* 56 (říj. 2006), B125–B137. DOI: 10.1007/s10582-006-0188-1.

[4] F Wagner. "A quarter-century of H-mode studies". In: *Plasma Physics and Controlled Fusion* 49.12B (lis. 2007), B1–B33. DOI: 10.1088/0741-3335/49/12b/s01. URL: https://doi.org/10.1088%2F0741-3335%2F49%2F12b%2Fs01.

[5] F. Matos et al. "Classification of tokamak plasma confinement states with convolutional recurrent neural networks". In: *Nuclear Fusion* 60.3 (ún. 2020), s. 036022. DOI: 10.1088/1741-4326/ab6c7a. URL: https://doi.org/10.1088%2F1741-4326%2Fab6c7a.

[6] Giuseppe Rattá a Jesús Vega. "Confinement Regime Identification Using Artificial Intelligence Methods". In: dub. 2015, s. 337–346. ISBN: 978-3-319-17090-9. DOI: 10.1007/978-3-319-17091-6_28.

[7] M. Zorek. "Těžba dat z experimentů na tokamaku COMPASS". In: (2018).

[8] Ian J. Goodfellow, Yoshua Bengio a Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[9] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* (1958), s. 65–386.

[10] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), s. 303–314.

[11] Zhou Lu et al. "The expressive power of neural networks: A view from the width". In: *Advances in neural information processing systems*. 2017, s. 6231–6239.

[12] R. H R Hahnloser et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". English (US). In: *Nature* 408.6815 (pros. 2000). ISSN: 0028-0836. DOI: 10.1038/35050018.

[13] Xavier Glorot, Antoine Bordes a Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, s. 315–323.

[14] Andrew L Maas, Awni Y Hannun a Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Sv. 30. 1. 2013, s. 3.

[15] Djork-Arné Clevert, Thomas Unterthiner a Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[16] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] Jack Kiefer, Jacob Wolfowitz et al. "Stochastic estimation of the maximum of a regression function". In: *The Annals of Mathematical Statistics* 23.3 (1952), s. 462–466.

[18] Diederik P Kingma a Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[19] Michael A Nielsen. *Neural networks and deep learning*. Sv. 2018. Determination press San Francisco, CA, USA: 2015.

[20] Robert Geirhos et al. "Comparing deep neural networks against humans: object recognition when the signal gets weaker". In: *arXiv preprint arXiv:1706.06969* (2017).

[21] *Feature maps example*. Nvidia. URL: `%5Cbibitem%7BCNN%7D[Online]%20%5Curl%7Bhttps://devblogs.nvidia.com/wp-content/uploads/2014/09/nn_example-624x218.png%7D`.

[22] "Understanding LSTM networks". In: *colah's blog* (2015). `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[23] Ilya Sutskever, Oriol Vinyals a Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, s. 3104–3112.

[24] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[25] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of machine learning research* 11.Dec (2010), s. 3371–3408.

[26] Raquel Urtasun a Rich Zemel. *CSC 411: Lecture 14: Principal Component Analysis & Autoencoders*. `https://www.cs.toronto.edu/~urtasun/courses/CSC411/14_pca.pdf`. 2015.

[27] Ronald A Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), s. 179–188.

[28] Diederik P Kingma a Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[29] Sergey Ioffe a Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[30] Otto Fabius a Joost R van Amersfoort. "Variational recurrent auto-encoders". In: *arXiv preprint arXiv:1412.6581* (2014).

[31] Durk P Kingma et al. "Semi-supervised learning with deep generative models". In: *Advances in neural information processing systems*. 2014, s. 3581–3589.

[32] *Pyro Examples*. `https://pyro.ai/examples/ss-vae.html#First-Variant:-Standard-objective-function,-naive-estimator`.

[33] *RobustScaler*. `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`.

[34] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. H. Wallach et al. Curran Associates, Inc., 2019, s. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[35] Eli Bingham et al. "Pyro: Deep Universal Probabilistic Programming". In: *CoRR* abs/1810.09538 (2018). arXiv: `1810.09538`. URL: `http://arxiv.org/abs/1810.09538`.

[36] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), s. 2825–2830.

[37] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, s. 770–778.

[38] Chuanqi Tan et al. "A survey on deep transfer learning". In: *International conference on artificial neural networks*. Springer. 2018, s. 270–279.

[39] Hassan Ismail Fawaz et al. "InceptionTime: Finding AlexNet for Time Series Classification". In: *arXiv preprint arXiv:1909.04939* (2019).

[40] *Jensenova nerovnost*. urlhttps://mathworld.wolfram.com/JensensInequality.html.

Příloha A

Pomocné výpočty

$$
\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log\left(\int_{\mathbb{R}^J} p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})d\boldsymbol{z}\right) \\
&= \log\left(\int_{\mathbb{R}^J} p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}d\boldsymbol{z}\right) \\
&= \log\left(\int_{\mathbb{R}^J} q_\phi(\boldsymbol{z}|\boldsymbol{x})\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}d\boldsymbol{z}\right) \\
&= \log\left(\int_{\mathbb{R}^J} q_\phi(\boldsymbol{z}|\boldsymbol{x})\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}d\boldsymbol{z}\right) \\
&= \log\left(\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]\right) \\
&\geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right)\right] \text{(Jensenova nerovnost [40])} \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) + \log p_\theta(\boldsymbol{x},\boldsymbol{z})\Big] \\
&= \mathcal{L}(\theta,\phi;\boldsymbol{x})
\end{aligned} \tag{A.1}
$$

$$
\begin{aligned}
\log p_\theta(\boldsymbol{x}) - \mathcal{L}(\theta,\phi;\boldsymbol{x}) &= \log p_\theta(\boldsymbol{x}) - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) + \log p_\theta(\boldsymbol{x},\boldsymbol{z})\Big] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[\log p_\theta(\boldsymbol{x}) + \log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p_\theta(\boldsymbol{x},\boldsymbol{z})\Big] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log\left(\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{p_\theta(\boldsymbol{x})}\right)\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p_\theta(\boldsymbol{z}|\boldsymbol{x})\Big] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[\log\left(\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\right)\Big] \\
&= D_{KL}\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\Big)
\end{aligned} \tag{A.2}
$$

$$
\begin{aligned}
\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[\log p_\theta(\boldsymbol{z})\Big] &= \int_{\mathbb{R}^J} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{z}) d\boldsymbol{z} \\
&= \int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \log \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I}) d\boldsymbol{z} \\
&= \int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \log \Big((2\pi)^{-\frac{J}{2}} e^{-\frac{1}{2}\boldsymbol{z}^T\boldsymbol{z}}\Big) d\boldsymbol{z} \\
&= \int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \log \Big((2\pi)^{-\frac{J}{2}} e^{-\frac{1}{2}\sum_{j=1}^J z_j^2}\Big) d\boldsymbol{z} \\
&= -\frac{J}{2}\log 2\pi \int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) d\boldsymbol{z} - \frac{1}{2}\int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \boldsymbol{z}^T\boldsymbol{z} \quad d\boldsymbol{z} \\
&= -\frac{J}{2}\log 2\pi \cdot 1 - \frac{1}{2}\mathbb{E}_{\mathcal{N}\left(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\right)}\Big[\boldsymbol{z}^T\boldsymbol{z}\Big] \\
&= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^J \mathbb{E}_{\mathcal{N}(\mu_j, \sigma_j^2)}\Big[z_j^2\Big] \\
&= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^J \Big(\mu_j^2 + \sigma_j^2\Big) \\
&= -\frac{1}{2}\sum_{j=1}^J \Big(\log 2\pi + \mu_j^2 + \sigma_j^2\Big)
\end{aligned} \tag{A.3}
$$

$$
\begin{aligned}
\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\Big[\log q_\phi(\boldsymbol{z}|\boldsymbol{x})\Big] &= \int_{\mathbb{R}^J} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log q_\phi(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z} \\
&= \int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \log \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) d\boldsymbol{z} \\
&= \int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \log \Big((2\pi)^{-\frac{J}{2}} \prod_{j=1}^J (\sigma_j^{-\frac{1}{2}}) e^{-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{\mu})^T \mathrm{diag}(\boldsymbol{\sigma}^{-1})(\boldsymbol{z}-\boldsymbol{\mu})}\Big) d\boldsymbol{z} \\
&= \int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) \cdot \Big(-\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^J \log \sigma_j\Big) d\boldsymbol{z} \\
&\quad - \frac{1}{2}\int_{\mathcal{R}^J} \mathcal{N}\big(\boldsymbol{z}|\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\big) (\boldsymbol{z}-\boldsymbol{\mu})^T \mathrm{diag}(\boldsymbol{\sigma}^{-1})(\boldsymbol{z}-\boldsymbol{\mu}) d\boldsymbol{z} \\
&= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^J \log \sigma_j^2 - \frac{1}{2}\sum_{j=1}^J \mathbb{E}_{\mathcal{N}\left(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)\right)}\Big[(z_j - \mu_j)^2 \cdot \sigma_j\Big] \\
&= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^J \log \sigma_j^2 - \frac{1}{2}\sum_{j=1}^J \mathbb{E}_{\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{I}\right)}\Big[y_j\Big] \\
&= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_{j=1}^J \log \sigma_j^2 - \frac{1}{2}\sum_{j=1}^J 1 \\
&= -\frac{1}{2}\sum_{j=1}^J \Big(\log 2\pi + \log \sigma_j^2 + 1\Big)
\end{aligned} \tag{A.4}
$$

Příloha B

Doplňující tabulky

hyperparametr	hodnoty					
jádro	RBF	Sigmoid				
C	1	10	25	50	75	100
γ	0.001	0.01	0.1	*scale*		

Tabulka B.1: Hodnoty hyperparametrů pro Grid Search u klasifikátor SVM. 3D mřížka vznikne ze všech možných kombinací těchto tří hyperparametrů. Hodnota *scale* je rovna převrácené hodnotě rozptylu dat ($scale = \frac{1}{\mathbb{V}ar(\mathbb{X})}$).

hyperparametr	hodnoty			
n_stromů	100	500	1000	1500
lr	0.1	0.01		

Tabulka B.2: Hodnoty hyperparametrů pro Grid Search u klasifikátor Gradient Tree Boosting (GB). 2D mřížka vznikne ze všech možných kombinací těchto dvou hyperparametrů. Zkratka *n_stromů* označuje počet stromů a *lr* míru učení.

Inception(32, (5,11,23), 1) + ReLU
LSTM(100)
LSTM(25)
Dense(1)
Dense(4) + Softmax

Tabulka B.3: Architektura konvoluční-rekurentní neuronové sítě. Dense(n) (resp. LSTM(n)) označuje dense (resp. LSTM) vrstvu s n výstupy. Inception(n, f, bn) představuje inception vrstvu, která se skládá ze tří 1D konvolučních vrstev, 1D MaxPoolingu a zúžení, tzb. bottlenecku. Bottleneck je 1D konvoluční vrstva s velikostí filtru 1 a s počtem filtrů bn. Tři další konvoluční vrstvy mají shodně n masek s velikostmi f_1, f_2 a f_3. Výsledný výstup je tvořen složením MaxPoolingu a výstupu tří konvolučních vrstev ($4 \cdot n$). Dense vrstva navazující na poslední LSTM vrstvu má sice výstup tvořený pouze jedním neuronem, ale je aplikovaná na 160 prvků sekvence, tzn výstupem této vrstvy je ve skutečnosti vektor délky 160.