VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

BAYESOVSKÉ A NEURONOVÉ SÍTĚ

BAYESIAN AND NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. BOHUSLAV HLOŽEK**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. Ing. FRANTIŠEK ZBOŘIL, CSc.**
SUPERVISOR

BRNO 2017

Abstrakt

Tato práce představuje Bayesovskou neuronovou síť na základě modelu Occamovy břitvy. První část práce shrnuje základní poznatky o neuronových sítích a Bayesovo pravidlo. Je vysvětlen princip Occamova ostří a detaily Bayesovské neuronové sítě. Rovněž je představen reálný příklad použití k predikci sesuvu půdy. V druhé části práce je představeno, jak vytvořit Bayesovskou neuronovou síť v jazyce Python. Je ukázán demonstrační program, který na experimentálních datech ukazuje vlastnosti Bayesovských neuronových sítí.

Abstract

This paper introduces Bayesian neural network based on Occams razor. Basic knowledge about neural networks and Bayes rule is summarized in the first part of this paper. Principles of Occams razor and Bayesian neural network are explained. A real case of use is introduced (about predicting landslide). The second part of this paper introduces how to construct Bayesian neural network in Python. Such an application is shown. Typical behaviour of Bayesian neural networks is demonstrated using example data.

Klíčová slova

Bayesovské neuronové sítě, Umělé neuronové sítě, Occamova břitva, Neuron, Přeučení, Bayesovo pravidlo, TensorFlow, Edward, Keras

Keywords

Bayesian neural networks, Artificial neural networks, Occams razor, Neuron, Overfitting, Bayes rule, TensorFlow, Edward, Keras

Citace

HLOŽEK, Bohuslav. *Bayesovské a neuronové sítě*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zbořil František.

Bayesovské a neuronové sítě

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Doc. Ing. Františka Zbořila CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bohuslav Hložek
22. května 2017

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato diplomová práce na téma Bayesovských a neuronových sítí se zabývá propojením těchto dvou světů. Na jedné straně stojí umělé neuronové sítě, které reprezentují model připomínající černou skříňku. Vybraným neuronům je přiložen vstup. Neurony v síti poté provádí vhodné operace a ze sítě se vrátí výsledek.

Oproti tomu Bayesovská pravděpodobnost stojí na známých pravděpodobnostech a závislostech mezi daty, které pracují s neurčitostí. Víme přesně, co děláme. Víme přesně, proč to děláme. Jen je nutné počítat s neurčitostí vstupů i výstupů. To je značný rozdíl oproti neuronové síti, ve které nemusíme chápat, proč síť dělá to, co dělá. To nastává díky přítomnosti parametrů uvnitř sítě, jako jsou například váhy, které se běžnému uživateli nejspíše jeví jako magické konstanty. Výsledek sítě je samozřejmě jednoznačný. Opět rozdíl vůči Bayesovu přístupu.

Přesto je možné tyto dva světy propojit. Uvnitř práce je rozebrána Bayesovská neuronová síť (BNN). Tato síť je v základu umělou neuronovou sítí (NN), ale přidává neurčitost k parametrům sítě. Místo váhy v síti máme pravděpodobnostní rozdělení vah. Místo výstupu máme pravděpodobnostně rozložený výstup. Místo magické činnosti sítě vzniklé některým z učících algoritmů máme Bayesovskou inferenci pracující na principu – čím jednodušší, tím lepší.

Práce je rozdělena do kapitol. V kapitole *Neuronové sítě* shrnuji poznatky o klasickém přístupu k neuronovým sítím. Popsána je topologie, učení i použití sítě. Poté následuje podkapitola o přeučení. Zmínění této nevýhody sítě je důležité, neboť právě Bayesova (popř. Bayesovská, užívají se oba termíny) neuronová síť tento problém do značné míry řeší.

Další kapitola s názvem *Vybrané pojmy z neurčitosti* stručně představuje několik základních pojmů z této oblasti, které práce potřebuje. Zejména se věnuje Bayesovu pravidlu a entropii.

Pravděpodobně nejdůležitější kapitolou v celé práci je kapitola o *Occamově břitvě*. V této kapitole bude rozveden můj předchozí výrok – čím jednodušší, tím lepší. Bude ukázáno, že jednodušší modely jsou pravděpodobněji správnější než ty složité. Jednoduchý model je vhodný pro neuronovou síť. Pravděpodobnosti získáme Bayesovým vzorcem. Tím vzniká propojení mezi světem neurčitosti a světem neuronových sítí.

Následuje kapitola *Bayesovské neuronové sítě*. V této kapitole popisuji, jak síť obecně funguje, včetně jejího matematického modelu. Je ukázána návaznost na Occamovu břitvu i příklad návrhu některých implementací. Představena je implementace podle Neala [14] a metoda založená na Kullback-Leiblerově divergenci.

Kapitola *Využití sítě* představuje skutečný nedávný příklad z Vietnamu, kde klasifikátor pro určení náchylnosti oblasti k sesuvu půdy podával lepší výsledky, když byl implementován jako Bayesovská neuronová síť.

V kapitole *Demonstrace BNN* popisuji vytvoření demonstračního programu, který je schopen řešit zadané regresní úlohy. Řešit umí pomocí standardní umělé neuronové sítě i pomocí Bayesovské neuronové sítě. Rovněž jsou představeny nástroje, které byly k vytvoření programu použity.

Experimenty s tímto programem a jejich interpretace je v kapitole *Experimenty s BNN*. Popisem výsledků těchto experimentů a shrnutí kladů i záporů BNN se zabývá kapitola *Zhodnocení BNN*. Následuje už jen kapitola *Závěr*, která stručně shrnuje dosažené výsledky a nabízí témata pro budoucí výzkum.

Po přečtení práce by měl čtenář pochopit, co je to Bayesovská neuronová síť a v čem se liší od obvyklé umělé neuronové sítě. Čtenář porozumí, na jakých hypotézách tento přístup stojí a informativně bude chápat matematiku, na které jsou Bayesovské neuronové sítě založeny. Čtenář bude schopen vytvořit vlastní sítě v jazyce Python a bude vědět, co od těchto sítí může očekávat.

Kapitola 2

Neuronové sítě

Pokud chceme pochopit činnost Bayesovy neuronové sítě, je zapotřebí chápat samotnou neuronovou síť. Neuronová síť je model strojového učení založený na jeho biologické předloze. Mezi výhody tohoto modelu patří zejména jeho schopnost najít souvislosti mezi daty, o kterých nemáme úplnou informaci. Jelikož výhody Bayesovy sítě budou spočívat zejména v úlohách regrese a klasifikace založené na modelu *MLP* (Multilayer perceptron) učeného pomocí algoritmu *backpropagation*, zaměřím se na tento konkrétní model.

Celkový model se skládá z pospojovaných komponent zvaných neurony. Pro účely klasifikace či regrese je potřeba mít data, jejichž výstupní hodnota je známá. U klasifikace jsou dostupné vzorky dat, o kterých víme, kterou třídu každý z nich reprezentuje. Pro účely regrese obvykle data značí naměřené hodnoty funkce, kterou se snažíme moelem získat. Tato data se rozdělí na tři skupiny [11].

První skupinou jsou trénovací data. Trénovací data se použijí k nastavení řídících parametrů sítě (váhy, biasy, viz. dále) tak, aby model data reprezentoval co nejlépe. Druhou skupinou dat jsou data validační. Ta se používají k testování samotného modelu sítě, jeho neměnných parametrů (angl. *hyperparameters*), jako je například koeficient učení či počet neuronů v jednotlivých vrstvách. Jinými slovy, validační data se používají k porovnání více modelů s odlišně nastavenými hyperparametry. Poslední skupinou jsou data testovací, která již naučenou síť podrobují testu, jestli je schopna správně zařadit vzorek do jeho třídy u klasifikace a odvodit správnou spojitou hodnotu v úloze na regresi.

V následujících podkapitolách bude model neuronové sítě rozebrán podrobněji.

2.1 Neuron

Neuron je základní stavební jednotka neuronové sítě. S okolní sítí komunikuje pomocí svých vstupů $x_1 \dots x_n$ a svého jediného výstupu y. Spoje mezi neurony, po kterých se přenáší výstupní informace jednoho neuronu jako vstupní informace dalšího neuronu, jsou parametrizovány váhami w – právě tyto váhy jsou řídící parametry modelu, které se během učení budou měnit. Úlohou neuronu je transformovat své vstupy na výstup. K tomu využívá dvou funkcí.

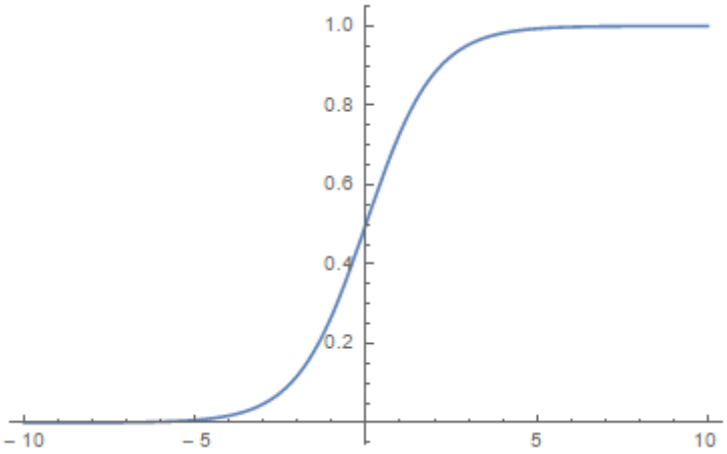
První funkce se nazývá bázovou funkcí. Jejím úkolem je zpracovat všechny vstupy na jedinou hodnotu u. V případě sítě *backpropagation* se využívá lineární bázové funkce:

$$u = \theta + \sum_{i=1}^{n} x_i \cdot w_1$$

Hodnoty x jsou vstupy neuronu (kterých je celkově n), násobené příslušnými vahami w a následné přičtení konstanty θ (angl. *Bias*). *Bias* slouží jako prvotní posun funkce a slouží jako pomocná konstanta pro rychlejší učení (je-li alespoň částečně znám řešený problém a pomocí heuristik předpřipravit neuron na očekávané číslo). Pokud žádné heuristiky nejsou použity, je standardním přístupem zvolit *bias* jako dostatečně malé číslo – generované např. z normálního rozdělení.

Hodnota u bude pomocí druhé funkce transformována na výstup. Této funkci říkáme aktivační funkce. Pro model *backpropagation* se často využívá funkce založená na sigmoidě (obr. 2.1) ve tvaru:

$$y = \frac{1}{1 + \exp(-u)} \tag{2.1}$$



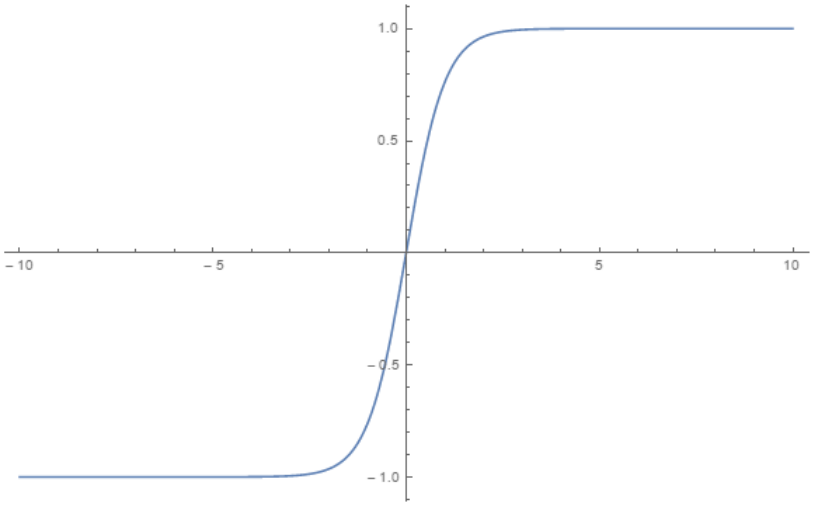
Obrázek 2.1: Sigmoida

Další z možných aktivačních funkcí je funkce založená na hyperbolickém tangentu (obr. 2.2):

$$y = tanh(u) = \frac{e^{2u} - 1}{e^{2u} + 1} \tag{2.2}$$

Odvození vzniklo obdobně jako u tangentu – jen místo *sin* a *cos* je užit *sinh* a *cosh*:

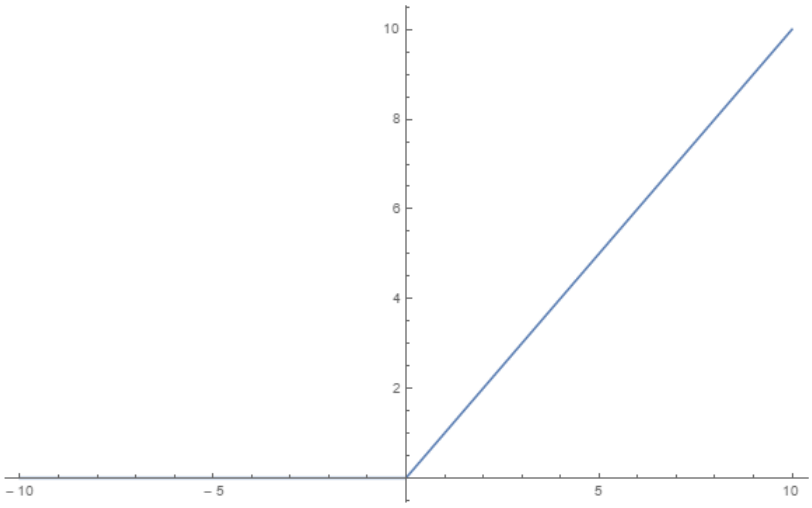
$$\begin{aligned} y &= tanh(u) = \frac{\sinh u}{\cosh u} = \frac{(e^u - e^{-u})/\cancel{2}}{(e^u + e^{-u})/\cancel{2}} = \\ &= \frac{\frac{e^u \cdot e^u}{e^u} - \frac{1}{e^u}}{\frac{e^u \cdot e^u}{e^u} + \frac{1}{e^u}} = \frac{(e^{2u} - 1)/\cancel{e^u}}{(e^{2u} + 1)/\cancel{e^u}} \end{aligned} \tag{2.3}$$



Obrázek 2.2: Hyperbolický tangens

Volba správné aktivační funkce, případně její modifikace (např. posunu v osách), vzhledem k řešené úloze, je důležitým bodem vývoje sítě a může znatelně ovlivnit rychlost i kvalitu učení [9]. V posledních letech se stává čím dál více oblíbenou usměrňovací aktivační funkce (angl. *rectifier*), zejména velmi jednoduchá varianta zvaná *ReLU* (Rectified linear unit, obr. 2.3).

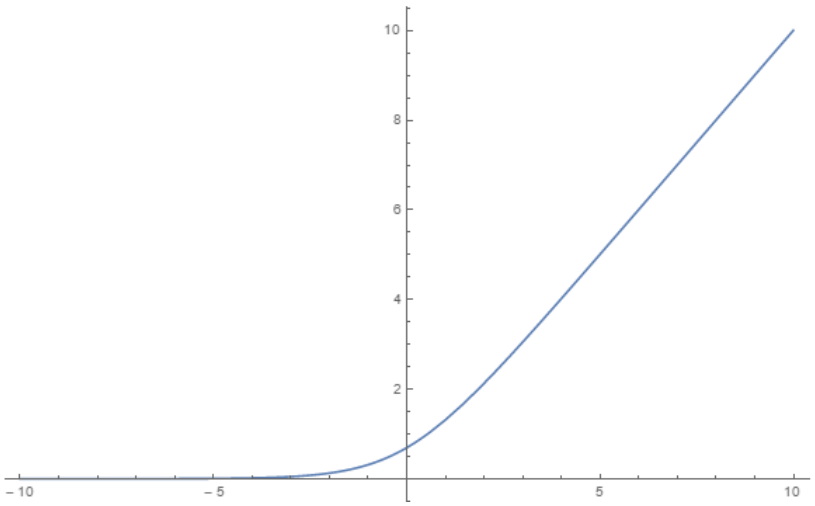
$$y = max(0, x) = \begin{cases} u & \text{pro } u > 0 \\ 0 & \text{pro ostatní případy} \end{cases} \tag{2.4}$$



Obrázek 2.3: Usměrňovací funkce – ReLU

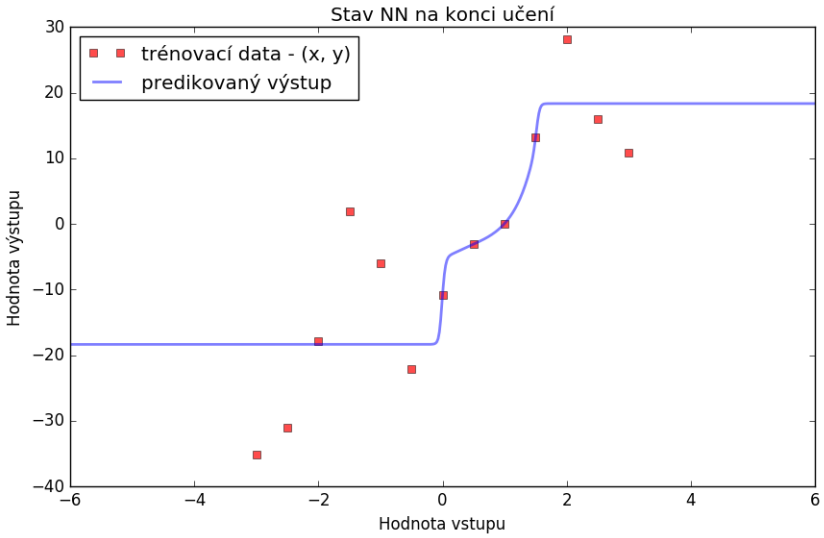
Pro usměrňovací funkci existují i další varianty, např. funkce zvaná *SoftPlus* (obr. 2.4), která je spojitá, a oproti $ReLU$ má derivaci i v bodě nula.

$$y = \ln(1 + e^u) \tag{2.5}$$

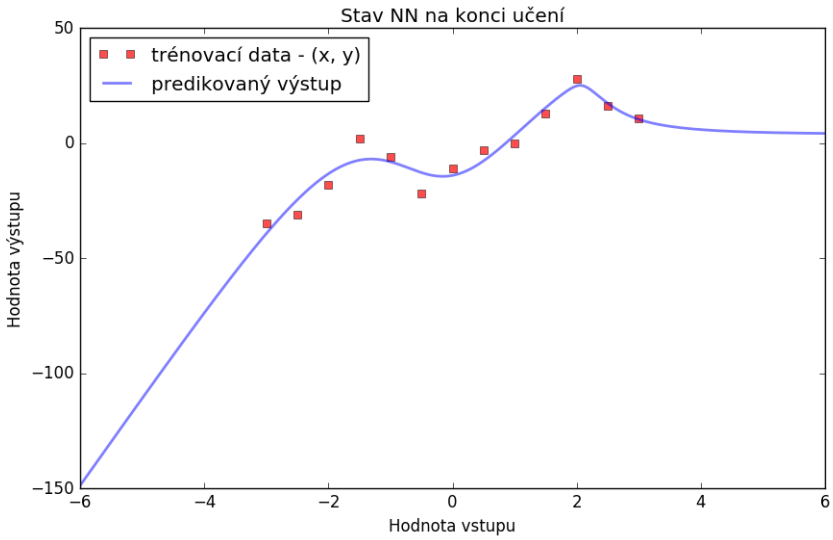


Obrázek 2.4: Usměrňovací funkce – SoftPlus

Abych předvedl, že na správné volbě aktivační funkce skutečně záleží, ukáži regresní úlohu učenou stejně dlouho na stejné topologii, jen s jinou aktivační funkcí. Lze vidět, že aktivační funkce hyperbolického tangentu (obr. 2.5) pro tuto regresní úlohu podává značně horší výsledky něž funkce *SoftPlus* (obr. 2.6)

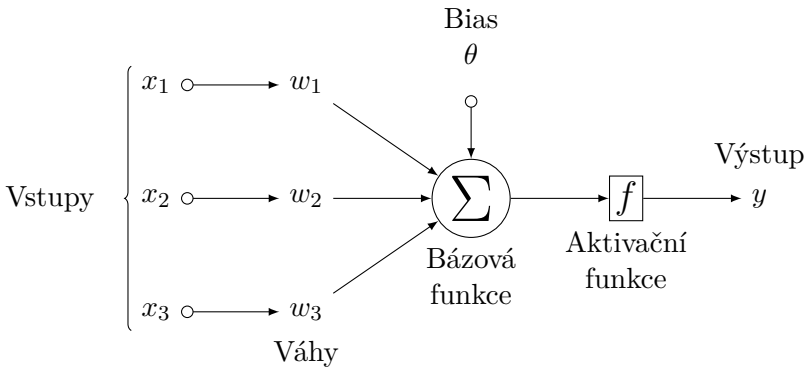


Obrázek 2.5: Ukázka nevhodné aktivační funkce - tanh



Obrázek 2.6: Ukázka vhodné aktivační funkce - softplus

Poznatky o neuronu v síti *MLP* (učené např. metodou *backpropagation*) lze shrnout na následujícím obrázku 2.7.



Obrázek 2.7: Neuron

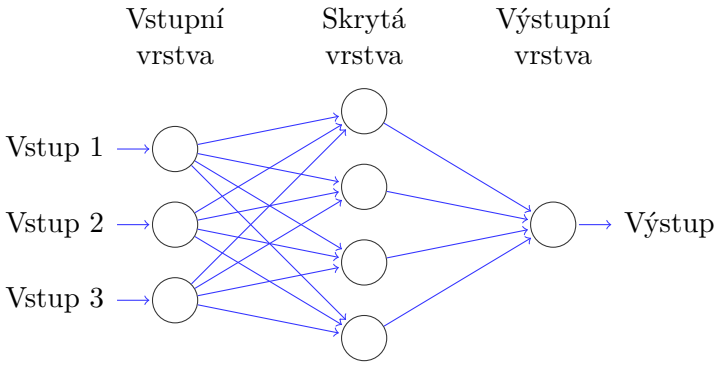
2.2 Síť a její fungování

Neurony v síti jsou uspořádány do vrstev v tzv. dopředné síti. Dopředná síť se vyznačuje všemi následujícími body:

- Neurony jsou organizovány do vrstev, přičemž neurony stejné vrstvy mezi sebou nemají žádné spoje.
- Spoje mezi neurony jsou topologicky orientovány jen v jednom směru.
- Spoje jsou pouze mezi neurony sousedních vrstev.
- Síť má maximální možné množství spojů.

První vrstvu někdy nazýváme *vstupní vrstvou*, neboť vstupy neuronů v první vrstvě odpovídají zakódované vstupní informaci. Poslední vrstvu nazýváme *výstupní vrstvou*. Výstupy neuronů z této vrstvy reprezentují zakódovanou informaci o výstupu sítě. Vrstvy od první do předposlední nazýváme *skryté vrstvy*, které transformují výstupy neuronů předchozí vrstvy na vstup pro neurony následující vrstvy.

Je obvyklé, že všechny neurony v síti mají stejnou bázovou funkci. Oprati tomu se často stává, že aktivační funkce se liší u neuronů výstupní vrstvy a ostatních vrstev. Existují případy, kdy se jako výstup (např. u regrese) bere přímo hodnota vzniklá z bázové funkce, tj. $y = u$.



Obrázek 2.8: Neuronová dopředná síť

Během tvorby sítě je potřeba najít vhodné množství skrytých vrstev a do nich umístit vhodné množství neuronů tak, aby síť byla schopna dostatečně dobře reprezentovat data z trénovací množiny. Síť má vždy jednu výstupní vrstvu a alespoň jednu vrstvu před ní. Pokud by síť mělo jej jednu vrstvu, byla by schopna řešit pouze triviální lineárně separovatelné problémy. To jsou problémy, které lze vyřešit jedinou přímkou v prostoru – např. u binární klasifikace by všichni reprezentanti první třídy musely ležet v jedné polorovině a reprezentanti druhé třídy v druhé polorovině roviny rozdělené danou přímkou.

Učení sítě spočívá ve snaze minimalizovat chybu na trénovacích datech. Vezměme si klasifikační úlohu. Máme zařadit vzorky do příslušných tříd. To se v klasickém podání dělá tak, že do výstupní vrstvy umístíme tolik neuronů, do kolika tříd chceme klasifikovat. Síť funguje tak, že po přiložení vstupu na vstupní vrstvu si počkáme na výsledek výstupní vrstvy. Vstup bude klasifikován do té třídy, jejíž reprezentant (neuron) nabývá extrému ve své hodnotě výstupu.

Snahou je, aby vstupní data byla správně klasifikována namapováním na patřičné reprezentanty výstupní vrstvy. To síť dělá pomocí změn svých vah. Otestujeme vzorek a na základě informace, jak testování dopadlo zpětně měníme váhy předcházejících vrstev. Cílem je, jak již bylo zmíněno, minimalizovat počet chybně zařazených vzorků, tj. výraz

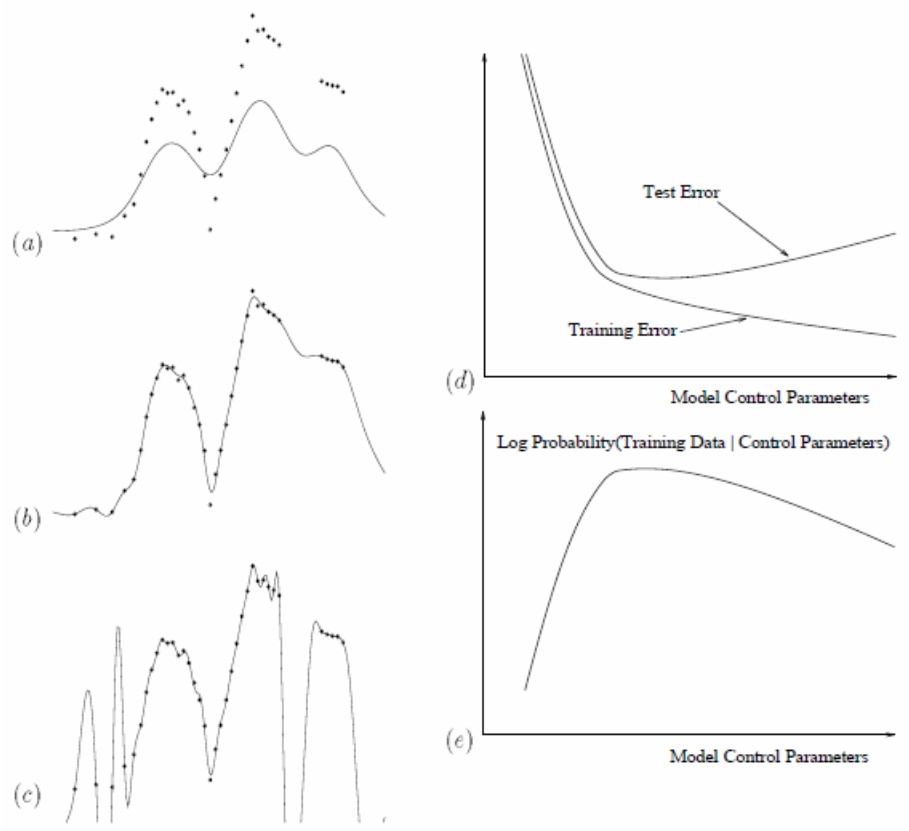
$$E_d = \frac{1}{2} \cdot \sum_{i=1}^{n} (y_i - t_i)^2 \tag{2.6}$$

má být minimální. Číslo n je počet trénovacích vzorků. Hodnota t je třída, do které vzorek patří. Sítí odvozená třída je označena jako y (výsledek klasifikace). Toto samozřejmě není jediný možný přístup. Výraz, který chceme minimalizovat, můžeme volit dle

typu úlohy. U regresní úlohy je obvykle volena minimalizace střední kvadratické odchylky výstupu sítě vůči očekávané hodnotě, což je zobecnění rovnice 2.6. Hodnota y zde značí predikovanou hodnotu a t je očekávaná hodnota (hodnota zde není diskrétní reprezentant třídy, ale hodnota z obecné funkce). Právě způsob učení bude zásadní odlišností Bayesových neuronových sítí.

2.3 Přeučení sítě

Problémem popsaných umělých neuronových sítí je jev zvaný *přeučení* (angl. *overfitting*). Tento jev způsobuje, že síť se naučí trénovací množinu příliš těsně a špatně zobecňuje. Podívejme se na obrázek z práce [10], který problematiku názorně ilustruje.



Obrázek 2.9: Ilustrace přeučení - převzato z [11]

Na obrázku je znázorněna úloha neuronové sítě pro regresi – část (a) až (c). Z jedné vstupní hodnoty (na vodorovné ose) se síť snaží předpovědět jednu výstupní hodnotu (na svislé ose). Body v prostoru znázorňují data trénovací množiny. Křivka znázorňuje chování sítě, přesněji její výstup vzhledem ke vstupu.

V části (a) lze pozorovat situaci, kdy je síť v počáteční fázi učení a zatím nereflektuje trénovací vzorky dostatečně přesně. Po nějaké době učení, kdy síť minimalizuje střední kvadratickou odchylku, se dostaneme do stavu (b). V tomto stavu je síť již mnohem lépe naučená. Nicméně stále pozorujeme body, kterými křivka neprochází,a proto necháme síť učit dále. Dostáváme se do stavu (c). Lze vidět, že křivka sítě relativně přesně interpoluje

body. Problémem je zbylý tvar křivky. Budeme věřit hypotéze, že mezi třetím a čtvrtým bodem je náhlý vzestup a pád křivky? V kapitole o *Occamově břitvě* ukáži, že hypotéza (c) je s velmi vysokou pravděpodobností chybná.

Došlo k přeučení sítě. Pokud se budeme snažit síť naučit trénovací vzorky zcela bezchybně – funkci pro chybu minimalizujeme téměř na 0 – pak tím velmi pravděpodobně zhoršíme schopnosti sítě zobecňovat zadaný problém [14]. To se projeví na velkém počtu chybně zařazených vzorků z testovací množiny. Podotknu, že zařazení testovacích vzorků do množiny trénovacích vzorků by problém nevyřešilo. Síť by se přeučila interpolací přes více bodů a podezřelé chování křivky by nezmizelo. Pokud bychom chtěli, aby se síť vždy s jistotou naučila přesně, potřebovali bychom nekonečně mnoho vzorků. Pak nejspíše máme analytické řešení problému, které vzorky generuje, a síť není vůbec potřeba.

Vidíme, že stav (b) je kvalitnější než stav (c). Je potřeba ve vhodném okamžiku zastavit učení. V části (d) je znázorněno, že vhodný okamžik by byl, když je součet chyby trénovacích dat a chyby testovacích dat minimální. Problémem ovšem je, jak takové místo najít, a jak nastavit hyperparametry sítě tak, aby se chyba minimalizovala. Část obrázku (e) mírně předbíhá k Bayesovskému přístupu k neuronovým sítím. Pro daný model (definovaný svými řídícími parametry a hyperparametry) ukazuje, kdy je hypotéza sítě nejpravděpodobnější. Jelikož jsou části (d) a (e) vhodně umístěny nad sebou, můžeme pozorovat, že maximum se nachází pod místem, kde jsou chyby minimální. Tam učení pro zvolené parametry končí. Následně můžeme zkusit jiné hyperparametry. Více o řešení tohoto problému v kapitole *Bayesovské neuronové sítě*. Více o tom, proč je užita logaritmická funkce, bude objasněno v kapitolce o entropii.

Ukazuje se, že síť má tendenci se přeučit, pokud nastane alespoň jedno z následujících:

- Učení probíhá příliš dlouho a síť se přeučí na datech.
- Není dostatečný počet trénovacích vzorků a síť není schopna odhalit správné závislosti mezi daty.
- Jsou chybně nastaveny hyperparametry sítě. Například příliš mnoho neuronů ve skryté vrstvě často vede k přeučení sítě.

Kapitola 3

Vybrané pojmy z neurčitosti

Další téma, jež je potřeba nastínit k pochopení Bayesových neuronových sítí, je problematika neurčitosti. Zásadním pojmem v neurčitosti je podmíněná pravděpodobnost. Místo toho, abychom použili běžnou statistiku, použijeme model beroucí v úvahu nejen všechny dostupné informace, ale také závislosti mezi nimi.

3.1 Podmíněná pravděpodobnost

Nechť $P(a)$ značí pravděpodobnost náhodného jevu a. Tato pravděpodobnost leží v intervalu $\langle 0; 1\rangle$. Možnost $P(a) = 0$ nazýváme *jevem nemožným*. Hodnotu $P(a) = 1$ pak *jevem jistým*. Podmíněnou pravděpodobnost značíme jako $P(a|b)$. Význam je pravděpodobnost jevu a za předpokladu, že je jev b pravdivý.

Toho se dá využit při zjištění pravděpodobnosti toho, že nastane více jevů zároveň.

$$P(a, b) = P(a|b) \cdot P(b) \tag{3.1}$$

Pokud máme statistický model s úplným rozdělením pravděpodobnosti, pak můžeme dosadit do vzorce a získat podmíněnou pravděpodobnost přímo. Další důležitou větou je [6]:

$$P(a) = \sum_{\forall b} P(a|b) \cdot P(b) \tag{3.2}$$

Tato věta nám říká, že nějaký jev lze získat z podmíněných pravděpodobností tak, že sečteme všechny výskyty podmíněných pravděpodobností vzhledem k jinému jevu. Pokud toto pravidlo zobecníme pro celou teorii (model s parametry) $\mathcal{H}$, dostaneme

$$P(A|\mathcal{H}) = \sum_{\forall B} P(A|B, \mathcal{H}) \cdot P(B|\mathcal{H}) \tag{3.3}$$

3.2 Bayesovo pravidlo

V této části odvodíme důležité Bayesovo pravidlo, které budeme v dalším využívat pro inferenci. Vyjdeme z rovnice (3.1), kterou použijeme dvakrát:

$$P(a, b) = P(a|b) \cdot P(b) \qquad P(b, a) = P(b|a) \cdot P(a)$$

Přítomnost jevů v pravděpodobnosti není závislé na jejich pořadí, a proto platí $P(a,b) = P(b,a)$. Pokud se nám rovnají levé strany, pak se musejí rovnat i pravé strany, tj. $P(a|b) \cdot P(b) = P(b|a) \cdot P(a)$. Po úpravě této rovnice dostaneme základní tvar Bayesova pravidla pro inferenci:

$$P(a|b) = \frac{P(b|a) \cdot P(a)}{P(b)} \tag{3.4}$$

Toto je zásadní rovnice, ze které budu v dalším vycházet, a proto každý z členů této rovnice označím jeho anglickým názvem dle následujícího předpisu:

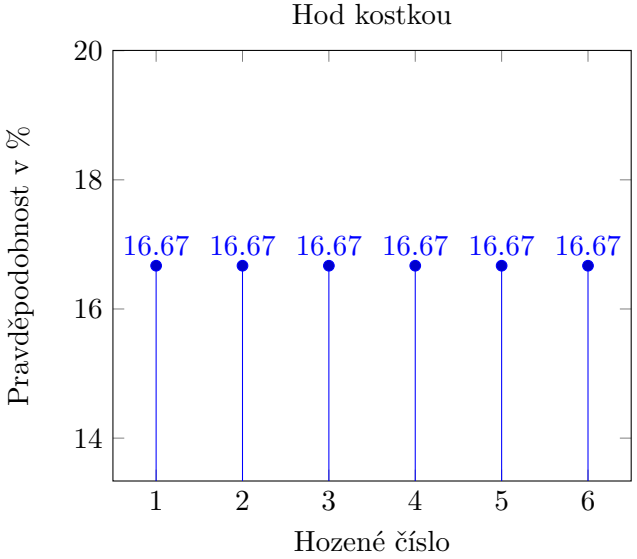
$$Posterior = \frac{Likelihood \cdot Prior}{Evidence} \tag{3.5}$$

V literatuře existuje i zjednodušený zápis ignorující normalizační člen *Evidence*, který nabývá na významu až při srovnání různých teorii. Z historických důvodů se místo značky $=$ užívá symbol α.V tomto případě je tvar zápisu:

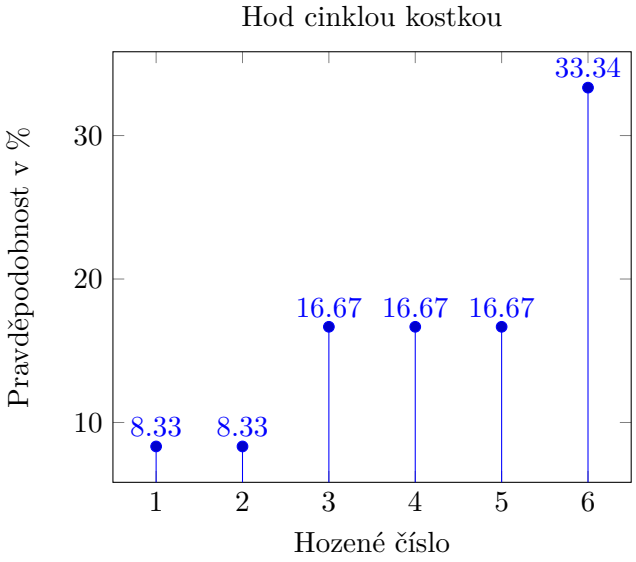
$$Posterior\ \alpha\ Likelihood \cdot Prior \tag{3.6}$$

3.3 Pravděpodobnostní rozdělení a hustota

Již víme, že $P(a)$ značí pravděpodobnost náhodného jevu a. Otázkou ovšem je, jakých všech různých hodnot může jev a nabývat, a jak moc jsou jednotlivé možnosti pravděpodobné. Nejprve se podíváme na diskrétní variantu rozdělení pravděpodobnosti. Předpokládejme, že máme homogenní hrací kostku ve tvaru krychle. Jev a bude označovat výsledek jednoho hodu. Pak existuje právě šest možných výsledků tohoto hodu. Za předpokladu, že je hod náhodný, můžeme říci, že každá možnosti má pravděpodobnost 1/6. Tím jsme popsali *rozdělení pravděpodobnosti* jevu a. Samozřejmě platí, že součet pravděpodobností všech jevů je roven jedné (tj. sto procent).



Pokud by byla použita podvržená kostka, na které padá šestka dvakrát častěji, ovšem jednička a dvojka padá jen s poloviční frekvencí, pak by rozdělení vypadalo následovně:



Toto rozdělení lze rozšířit i na spojité jevy. Mějme generátor, který má za úkol generovat číslo ze zadaného intervalu. Ovšem z některé části tohoto intervalu se čísla mají generovat častěji než z jiné části. Toto postihneme tzv. funkcí hustoty pravděpodobnosti $f(x)$, kdy funkce pro hodnotu x říká, jak pravděpodobné je vygenerovat tuto hodnotu. Máme-li interval obsahující nekonečně mnoho hodnot, pak nastává situace, kdy je statisticky nemožné vygenerovat jedno konkrétní číslo s pravděpodobností větší než nula, a proto nepůjde užit diskrétní přístup.

Jak již bylo naznačeno, bude se počítat pravděpodobnost, že číslo bude vygenerováno z daného pod-intervalu. Tato pravděpodobnost je definována jako plocha pod funkcí hustoty omezená krajními hodnotami intervalu, a to je vzorový případ pro výpočet integrálu. Platí, že

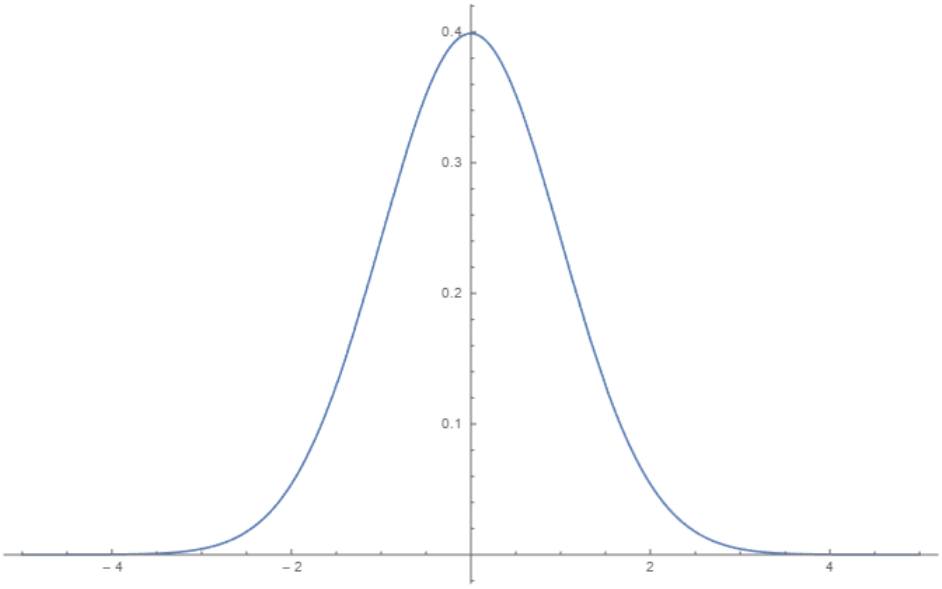
$$P(a \in \langle k; l \rangle) = \int_k^l f(x)dx \tag{3.7}$$

Opět musí platit, že plocha pod celou funkcí hustoty reprezentuje všechny možné výsledky (může generovat právě 100% možných případů), tj.

$$\int_{-\infty}^{\infty} f(x)dx = 1 \tag{3.8}$$

Obrázek 3.1 ukazuje příklad často užívané hustoty rozdělení, zvané jako normální (Gaussovo) rozdělení pravděpodobnosti. Toto rozdělení je určeno svým středem μ (nejpravděpodobnější hodnota v rozdělení) a odchylkou σ (čím vyšší, tím vyšší pravděpodobnost jevů více vzdálených od středu). Normální rozdělení má tvar:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.9}$$



Obrázek 3.1: Normální rozdělení pravděpodobnosti se středem v nule a odchylkou jedna

3.4 Entropie

Entropie je pojem, který definuje míru neurčitosti systému. Nízká entropie značí uspořádání systému, tzn. implicitně víme, jak se systém chová. Vysoká entropie značí vysokou neurčitost. To znamená, že o systému potřebujeme více informací, abychom byli schopni usoudit, jak se systém chová. Entropie je definována v řadě oborů. Nejprve se podívejme na termodynamickou entropii, která intuitivně prezentuje problém.

Mějme uzavřenou nádobu. Vnitřek nádoby je rozdělen na levou a pravou část a v nádobě je dané množství částic plynu. Představme si, že budeme pozorovat jednu konkrétní částici značenou jako x. Bude nás zajímat, jestli je v levé, nebo v pravé části nádoby.

První zkoumaná situace bude, kdy levá a pravá část nádoby bude oddělena pevnou překážkou a všechny částice plynu jsou v levé části. Použijeme naše pozorovací zařízení, které prohlásí, že pozorovaná částice x byla nalezena v levé části nádoby. Otázkou je, jak moc byla tato informace užitečná. Odpovědí je, že vůbec, neboť jsme si tuto informaci mohli odvodit přímo ze stavu. Všechny částice byly v levé části, a proto i pozorovaná částice x tam bude také. Jedná se o deterministický systém. Entropie je nulová, neboť pro vyřešení úlohy nepotřebujeme žádnou informaci.

V druhé situaci mějme 80% částic v levé části a 20% v pravé části nádoby. Jak dobrá bude v tomto případě informace o tom, kde se nachází částice x? Očekáváme, že částice nejspíše bude v levé části, ale jisti si nejsme. Informace bude hodnotnější než v první situaci. A tato informace bude nejhodnotnější v situaci, kdy žádnou ze situací nemůžeme upřednostnit před jinou, tzn. všechny stavy (částice v levé části, částice v pravé části) jsou stejně pravděpodobně. Tuto hodnotu lze samozřejmě vyjádřit matematicky.

Prvním způsobem vyjádření je *Boltzmannova entropie* definovaná jako:

$$S = k_B \ln W \tag{3.10}$$

V této rovnice je $k_B = 1,38 \cdot 10^{-23}$ Boltzmanova konstanta a W udává, kolik stavů existuje v daném rozložení makro-stavu. Nechť n je počet částic plynu, n_{left} a n_{right} jsou počty částic v levé a pravé části nádoby. Pak nás zajímá, kolika způsoby lze rozdělit n částic do těchto dvou stavů. To se spočítá jako:

$$W = \frac{n!}{n_{left}! \cdot n_{right}!} \tag{3.11}$$

Schválně nebyla zadána hodnota n, aby si čtenář uvědomil nevýhodu tohoto přístupu. Existuje i informatičtější přístup, který vyžaduje pouze relativní pravděpodobnosti jevů, a to je tzv. *Shannonova entropie*, která k vyjádření neurčitosti využívá aditivní logaritmickou funkci. Entropie je pak získána jako:

$$S(X) = -\sum_{n}^{i=1} P(q_i) \cdot \log_2 P(q_i) \tag{3.12}$$

Množina X obsahuje přípustné jevy, v našem případě je $X = \{x_{vlevo}, x_{napravo}\}$. Volba základu logaritmu dva je vhodná, neboť neurčitost je v tomto reprezentována v bitech. Sémantiku lze chápat jako počet ztracených bitů, který bylo potřeba použít k dodatečnému popisu dat. Dodatečně je dodefinováno $log_2 0 \stackrel{\text{def}}{=} 0$.

Teď je možné exaktně vyčíslit entropii v popisovaných situacích.

$$S(\{P(L) = 1, P(R) = 0\}) = -(1 \cdot \log_2 1 + 0 \cdot \log_2 0) = -(0 + 0) = 0$$

$$S(\{P(L) = 0,8, P(R) = 0,2\}) = -(0,8 \cdot \log_2 0,8 + 0,2 \cdot \log_2 0,2) = -(-0.258 - 0.464) = 0,722$$

$$S(\{P(L) = 0,5, P(R) = 0,5\}) = -(0,5 \cdot \log_2 0,5 + 0,5 \cdot \log_2 0,5) = -(-0.5 + -0.5) = 1$$

Jak bylo očekáváno, maximální neuspořádanost nastává v situaci, kdy jsou všechny možnosti stejně pravděpodobné. To vyplývá z klasického problému: zvol čísla x, y tak, aby $x + y = z$ a $x \cdot y$ je maximální (z je libovolná zadaná konstanta).

$$x = z - y$$
$$x \cdot y = (z - y) \cdot y = zy - y^2$$

Hledání maxima vede na derivaci a následné hledání extrému (položení derivace nule).

$$\begin{aligned} \frac{d(zy - y^2)}{dy} &= 0 \\ z - 2y &= 0 \\ y &= z/2 \\ x = z - z/2 &= z/2 \\ x = y &= z/2 \end{aligned}$$

Toto platí pro dvě čísla. Obdobně lze rozšířit pro n čísel na $prom_{lib} = z/n$. V tuto chvíli stačí dosadit za z celou pravděpodobnost jedna a získáme, že maximální entropie je při rovnoměrném rozdělení všech stavů. Entropie bude využita při učení Bayesovské neuronové sítě.

Kapitola 4

Occamova břitva

V této kapitole se zabývám principem zvaným *Occamova břitva* (anglicky *Occams Razor*). Tato hypotéza, pocházející již ze čtrnáctého století, tvrdí, že jednodušší modely jsou pravděpodobnější než modely složitější. V předchozích kapitolách jsme zjistili, že námi zkoumané neuronové sítě mají problémy s přeučením. Tento problém existuje i díky vzniku příliš složitého modelu dané sítě – s příliš mnoho řídícími parametry. Následně jsme si představili Bayesův způsob pro práci s neurčitostí.

Dáme-li tyto informace dohromady, zjistíme, že spočítáním nejpravděpodobnějšího modelu získáme model nejjednodušší (platí-li tato hypotéza), který má rovněž nejmenší náchylnost k přeučení, a proto je pochopení *Occamovy břitvy* zásadní.

4.1 Motivační příklad

Nejprve se podívejme na motivační příklad ke studiu *Occamovy břitvy* [12]. Mějme posloupnosti čísel:

$$-1, 3, 7, 11$$

Naším úkolem je odhadnout, jaké bude následující číslo. Dovolím si prezentovat dvě hypotézy. V obou hypotézách bude x označovat předchozí číslo v posloupnosti. První hypotéza tvrdí:

$$\mathcal{H}_1 : x_{i+1} = x_i + 4$$

Pro získání následujícího čísla posloupnosti vezmi předchozí a přičti k němu čtyři. Hypotéza může být pravdivá, neboť $-1 + 4 = 3$, $3 + 4 = 7$, $7 + 4 = 11$. Podle této hypotézy by následující číslo bylo $11 + 4 = 15$.

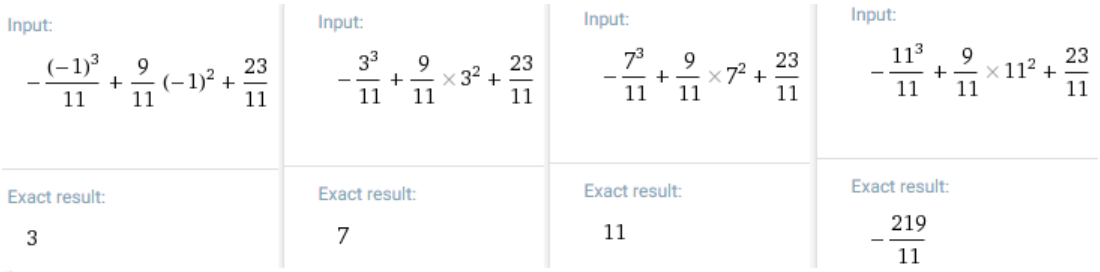
Druhá hypotéza, kterou představím, opět počítá následující číslo z předchozího, a to pomocí předpisu:

$$\mathcal{H}_2 : x_{i+1} = \frac{-1}{11} \cdot x_i^3 + \frac{9}{11} \cdot x_i^2 + \frac{23}{11}$$

Validitu tohoto modelu potvrzují výsledky znázorněné na obrázku 4.1. Dle této hypotézy má být následující číslo $-219/11$.

V tuto chvíli se nabízí otázka. Která z hypotéz $\mathcal{H}_1$ $\mathcal{H}_2$ je pravděpodobnější. Occamova břitva samozřejmě upřednostní první z hypotéz, neboť je jednodušší. Rozeberme si proč.

Occamova břitva využívá k porovnání hypotéz množství dat, které hypotézy mohou nabývat. Porovnání budeme provádět pomocí rovnice vycházející z 3.4 ve tvaru (člen evidence



Obrázek 4.1: Test hypotézy $\mathcal{H}_2$ na *www.wolframalpha.com*

se pokrátí):

$$\frac{P(\mathcal{H}_1|D)}{P(\mathcal{H}_2|D)} = \frac{P(\mathcal{H}_1)}{P(\mathcal{H}_2)} \cdot \frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_2)}$$

Chceme porovnat kvalitu hypotéz v závislosti na datech, která považujeme jako správná (pravdivost zadání prvních čtyř čísel nezpochybňujeme). Toto počítáme na levé straně rovnice. Člen $P(\mathcal{H}_1)/P(\mathcal{H}_2)$ znázorňuje naše počáteční představy o pravděpodobnosti pravdivosti hypotéz. Předpokládejme, že vůči žádné z teorií nemáme předsudky a jejich počáteční pravděpodobnosti nastavíme na stejnou hodnotu. Tím se tento člen zkrátí na 1. Zbývá nám člen:

$$\frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_2)}$$

Tento člen porovnává pravděpodobnost dat za předpokladu, že jsou hypotézy pravdivé. Porovnáme, kolik možných dat může být v hypotézách.$\mathcal{H}_1$ je lineární funkce, která závisí na prvním čísle v sekvenci a na konstantě, kterou budeme přičítat. Abychom se vyhnuli práci s mohutnostmi a nekonečnem, omezme se na celá čísla mezi -50 a 50. První číslo posloupnosti i přičítaná konstanta tak mohou nabývat 101 možností, a proto:

$$P(D|\mathcal{H}_1) = \frac{1}{101} \cdot \frac{1}{101} \doteq 0.0001$$

Hypotéza $\mathcal{H}_2$ se dá vyjádřit pomocí počáteční hodnoty posloupnosti (1/101) a pomocí tří zlomků. Předpokládejme, že zlomek bude ve tvaru, kdy je kladný jmenovatel (znaménko umístíme do čitatele). Zlomek $-1/11$ lze vyjádřit čtyřmi možnostmi, a to:

$$-1/11 \ \vee \ -2/22 \ \vee \ -3/33 \ \vee \ -4/44$$

. Obdobně zlomek 9/11 lze vyjádřit jako $9/11 \ \vee \ 18/22 \ \vee \ 27/33 \ \vee \ 36/44$. S omezením čísel do padesátí lze zlomek 23/11 vyjádřit jen pomocí dvou tvarů, a to $23/11 \ \vee \ 46/22$. Ve výsledku máme:

$$P(D|\mathcal{H}_2) = \frac{1}{101} \cdot \left(\frac{4}{101} \cdot \frac{1}{50}\right) \cdot \left(\frac{4}{101} \cdot \frac{1}{50}\right) \cdot \left(\frac{2}{101} \cdot \frac{1}{50}\right) \doteq 2.5 \cdot 10^{-12}$$

Konečně porovnáme pravděpodobnosti:

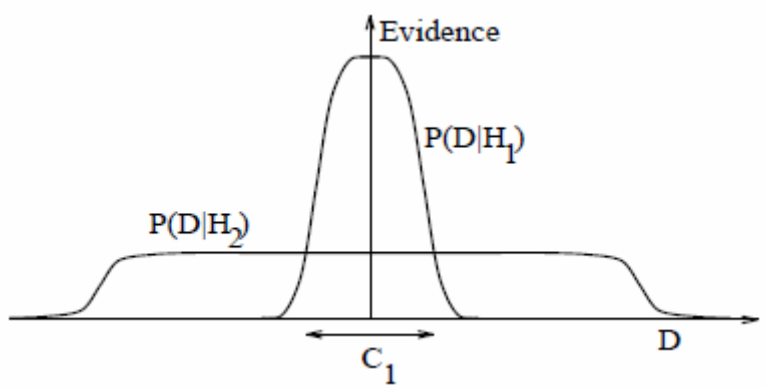
$$\frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_2)} = \frac{10^{-4}}{2.5 \cdot 10^{-12}} = 4 \cdot 10^7$$

A zjistíme, že hypotézu $\mathcal{H}_1$ preferujeme nad hypotézou $\mathcal{H}_2$ v poměru čtyřicet milionů ku jedné. Kdybychom neomezili čísla od -50 do 50, tak by hypotéza $\mathcal{H}_1$ vyhrála ještě drtivěji (až do nekonečně lepší hypotézy při žádném omezení).

4.2 Shrnutí Occamovy břitvy

Occamova břitva funguje velmi dobře ve spojeni s Bayesovou logikou. Ta díky Bayesově vzorci bere v úvahu nejen pravděpodobnost hypotézy na základě prezentovaných dat $P(\mathcal{H}|D)$, ale rovněž bere v úvahu, jak moc je model dobrý na základě přítomných dat $P(D|\mathcal{H})$.

Podívejme se na obrázek 4.2. V něm vidíme hypotézy $\mathcal{H}_1, \mathcal{H}_2$, nad nimiž nemáme žádnou počáteční preferenci. Vodorovná osa představuje prostor dat. $\mathcal{H}_1$ je jednodušší model s méně parametry, a proto jsou jeho možnosti limitované. Pokud ale správně reprezentuje oblast, ve které se skutečně nacházejí data, tj. oblast C_1, pak je vyšší pravděpodobnost, že je model $\mathcal{H}_1$ lepší než komplikovanější model $\mathcal{H}_2$, který by postihoval daleko širší rozložení dat.



Obrázek 4.2: Demonstrace Occamovy břitvy - převzato z [12]

Tento obrázek 4.2 lze chápat i jako aplikaci *No Free Lunch Teoremu* [5]. Tento teorém tvrdí, že každý z efektivních algoritmů strojového učení má stejnou výpočetní náročnost pro množinu všech existujících problémů. Jinými slovy, pokud některý algoritmus podává značně lepší výsledky pro danou třídu úloh než druhý, pak existuje jiná třída úloh, pro kterou je první model značně horší než druhý. Toto je, čeho se snažíme dosáhnout. Vytváříme model tak, aby dobře řešil zadanou úlohu. Funkci hustoty pro $P(D|\mathcal{H}_1)$ adaptujeme na přítomná data tak, aby model stále dobře reprezentoval očekávaná data. To sice může zhoršit výsledky pro jinou sadu úloh, ale nás zajímá, jestli síť se zadanými parametry generuje pro náhodné vstupy data, která poté skutečně přijdou. Neřešíme, jestli naše síť dokáže vyřešit i zcela jinou, irelevantní úlohu.

Kapitola 5

Bayesovské neuronové sítě

5.1 Teoretický základ

V klasickém pojetí tvorby neuronové sítě učení probíhá jako hledání optimálních hodnot vah jednotlivých neuronů daného modelu $\mathcal{H}_i$. Tento model je definován svými hyperparametry, jako je počet neuronů skryté vrstvy a obdobné koeficienty. Hledá se takový vektor vah w, který minimalizuje chybu sítě – model přizpůsobujeme datům.

V bayesovském principu ovšem nebudeme uvažovat jen jedno správné řešení pro w, ale budeme brát v potaz celé oblasti řešení – řídící parametry jako váhy budou náhodné proměnné. Pro každé řešení definujeme jeho pravděpodobnost jako $P(w|D, \mathcal{H}_i)$ určující, s jakou pravděpodobností jsou váhy w správné v modelu $\mathcal{H}_i$ vzhledem ke vstupním datům D.

Bayesovské neuronové sítě jdou s inferencí ještě dál. Jak bylo nastíněno v kapitole o *Occamově břitvě*, porovnávat můžeme modely i mezi sebou, a to na základě *evidence* (připomínám, že *evidence* je v této práci název pro jmenovatele z pravé strany vzorce 3.4) dat. Z principu *Occamovy břitvy* tím získáme jako nejpravděpodobnější model ten nejjednodušší. Nejjednodušší model pak velmi dobře zobecňuje a má mnohem menší problémy s přeučením.

Další výhoda těchto sítí je, že nepotřebují klasickou dělbu dat na trénovací, validační a testovací. Jak plyne z Bayesova vzorce, data a model se budou navzájem korigovat. Naším úkolem bude vyhodnotit modely – tím získáme řídící parametry (vektor vah, popř. i biasů). Následně porovnáme modely vůči sobě, což povede k zisku optimálních hyperparametrů. Rozepišme si úkol inference uvnitř modelu $\mathcal{H}_i$ pomocí Bayesova vzorce:

$$P(w|D, \mathcal{H}_i) = \frac{P(D|w, \mathcal{H}_i) \cdot P(w|\mathcal{H}_i)}{P(D|\mathcal{H}_i)} \tag{5.1}$$

Evidence vzorce 5.1 budeme využívat k porovnávání různých modelů. *Prior* říká, jakých hodnot pravděpodobně bude nabývat vektor vah w v daném modelu. K tomu se často využívá náhodné Gaussovo rozdělení [13]. To znamená, že v rámci jednoho modelu spočítáme *posterior* pomocí členu *likelihood*. Jelikož typické metody pomocí středních chyb a odchylek nejsou v Bayesově teorii průkazné, budeme *posterior* vyčíslovat pomocí Hessovy matice. Každý člen této čtvercové matice má tvar (i, j iterují přes parametry sítě a značí číslo řádku a sloupce):

$$(Hess\ a)_{ij} \equiv \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{5.2}$$

Vyčíslením získáme nejpravděpodobnější vektor vah w_{best}. Vyčíslovaná funkce pro Hessovu matici je:

$$a = -\nabla\nabla log P(w|D, \mathcal{H}_i) \tag{5.3}$$

To znamená, že pomocí gradientů hledáme minimum logaritmicky vzaté pravděpodobnosti vah při daných datech (to nás vrací k části (e) obrázku 2.9). Nezapomínáme, že v klasickém přístupu jsme hledali minimum chybové funkce. Návrh chybové funkce uvedu dále. Rovněž se není třeba obávat složitosti výpočtů. Později zjistíme, že tyto vzorce jsou matematicky korektní, ale místo tohoto můžeme použít Gaussovy aproximace, a ty budou řešeny pomocí metody Monte Carlo s využitím Markova řetězce.

Následně se dostaneme do fáze, kdy srovnáme modely mezi sebou. Postupovat budeme obdobně jako v kapitole o *Occamově břitvě*. Budeme porovnávat *posteriory* modelů vzhledem k přítomným datům (obdoba, jak jsme si v motivačním příkladu omezili data na -50 až 50). *Posterior* spočítáme jako:

$$P(\mathcal{H}_i|D)\ \alpha\ P(D|\mathcal{H}_i) \cdot P(\mathcal{H}_i) \tag{5.4}$$

Prior rovnice 5.4 opět znázorňuje naše počáteční očekávání o kvalitě daného modelu a dovoluje nám explicitně preferovat některý model nad jinými. V našich úvahách jsme vůči modelům indiferentní (je nám jedno, který zvítězí – zajímá nás jen jeho úspěšnost), a proto *Prior* všem modelům nastavíme konstantní.

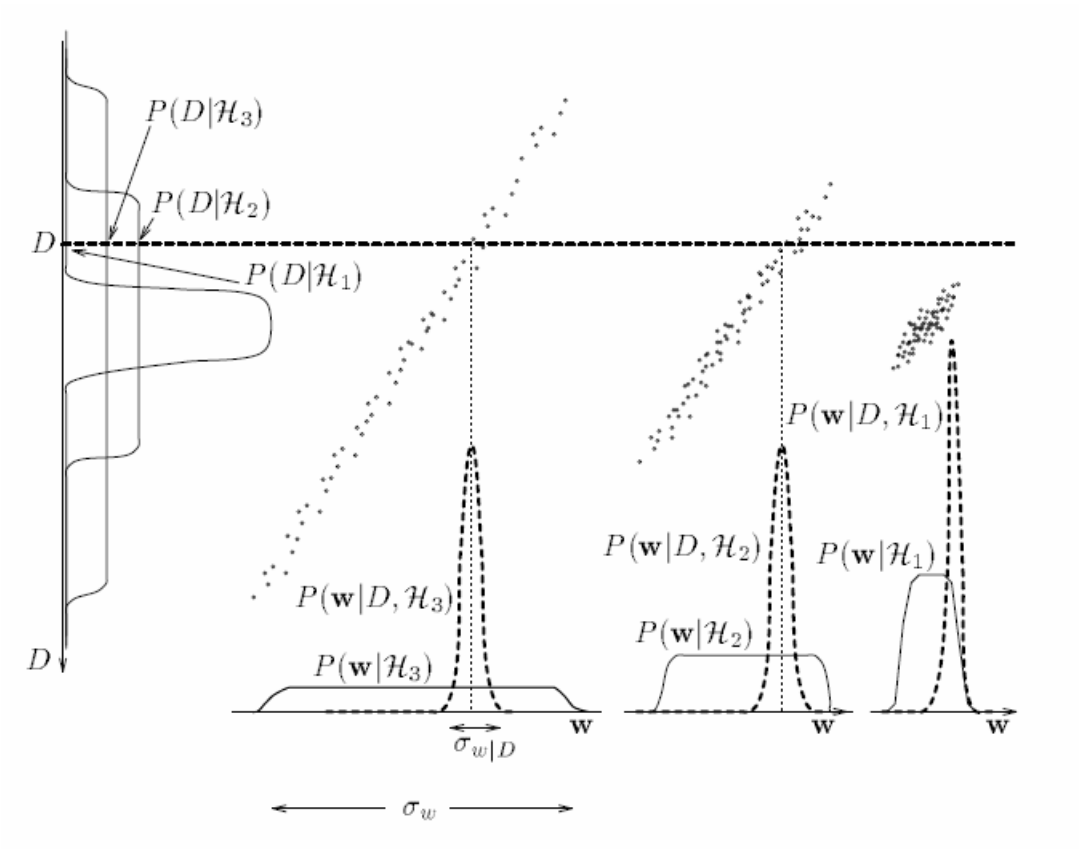
Zbývá nám vyhodnotit *likelihood*. Všimněme si, že *likelihood* ve rovnici 5.4 je roven *evidence* z rovnice 5.1. Opět se jedná o člen, který uvažuje pravděpodobnost dat v modelu dle *Occamovy břitvy*. Obecně se dá spočítat (díky rovnici 5.1) jako:

$$P(D|\mathcal{H}_i) = \int P(D|w, \mathcal{H}_i) \cdot P(w|\mathcal{H}_i) \mathrm{d}w \tag{5.5}$$

Pojďme se důkladněji rozebrat přítomnost *Occamovy břitvy* v porovnávání modelů. Východiskem k následujícím úvahám bude obrázek 5.1.

Na obrázku vidíme tři modely (liší se svými hyperparametry) – model $\mathcal{H}_1$, $\mathcal{H}_2$ a $\mathcal{H}_3$. Tyto modely jsou pro jednoduchost v jednodimenzionálním prostoru, kde je jediným řídícím parametrem váha w. Tím se vyhneme práci s maticemi a je možné problém smysluplně zakreslit do dvourozměrného obrázku.

Každá z hypotéz (reprezentované svým modelem) $\mathcal{H}_i$ má počáteční hustotu rozdělení z jinak širokého pásu přípustných hodnot hledané váhy w . Šířku *prioru* $\mathcal{H}_i$ budeme značit σ_{wi} . Hypotéza $\mathcal{H}_1$ má jen málo přípustných hodnot, $\mathcal{H}_3$ je nejobecnější a má jich nejvíce. Zřejmě platí: $\sigma_{w3} > \sigma_{w2} > \sigma_{w1}$. Jelikož žádnou hypotézu nepreferujeme nad jinou, tak pro každou vygenerujeme s Gaussovým rozdělením stejný počet náhodných dat ležících v příslušném rozdělení. Tato data jsou znázorněna jako body a vložíme je do modelu. Z rozložení bodů na svislé ose lze usoudit, že modely produkují vyšší hodnotu dat pro vyšší hodnoty hledané váhy.



Obrázek 5.1: Occamova břitva v Bayesově neuronové síti - převzato z [10]

Tato náhodně generovaná data se aplikují pro výpočet *posterioru*, stejně jako v našich předchozích případech. Získáme křivky $\sigma_{wi|D}$. Střed tohoto rozdělení bude nejpravděpodobnější váha w_{best} pro danou hypotézu. Tím jsme splnili první část úkolu – vyhodnocení jednoho modelu. Dále definujeme *Occamův faktor* jako:

$$OccamFactor = \frac{\sigma_{wi}}{\sigma_{wi|D}} \tag{5.6}$$

Tento faktor nám říká, jak se změní možné hodnoty parametrů (naší jedné váhy w) v závislosti na získaných datech (korespondují se svislou osou). Následně si z Bayesova vzorce na svislou osu vyznačíme $P(D|\mathcal{H}_i)$, tj. pravděpodobnost dat vzhledem k předpokladu platné hypotézy. Data použijeme námi vygenerovaná. Více bodů v oblasti se projeví jako vyšší křivka (funkce hustoty). Tuto část jsme viděli na 4.2, jen pootočenou o devadesát stupňů.

Poslední částí je příchod skutečného vzorku pro srovnání všech modelů. Nechť přijde nový vzorek dat D (na obrázku označen přerušovanou čárou). Ten vidíme jako bod na svislé ose. Nejpravděpodobnější teorii učíme jako maximum z hodnot $P(D|\mathcal{H}_i)$. Vidíme, že $P(D|\mathcal{H}_1)$ v bodě D nabývá hodnoty blízké nule (viz. Gaussovo rozdělení), a proto je model $\mathcal{H}_1$extrémně nepravděpodobný. Zbylé dva modely mají podstatně vyšší pravděpodobnost, a jelikož $P(D|\mathcal{H}_2) > P(D|\mathcal{H}_3)$, jako nejlepší z těchto modelů zvolíme $\mathcal{H}_2$.

5.2 Historická implementace

Nyní se podívejme na jeden konkrétní model Bayesovy neuronové sítě [8]. Původní funkci chyby z kapitoly on neuronových sítích E_d obohatím o další člen, regularizátor. Tento regularizátor bude mít za úkol zvýhodňovat menší hodnoty vah nad většími, neboť velké hodnoty vah velmi silně ovlivňují síť a vedou k přeučení a ke špatnému zobecňování. Cílem bude během učení minimalizovat navrženou chybovou funkci:

$$S(w) = \beta \cdot E_d + \alpha \cdot E_w \tag{5.7}$$

E_d je obvyklé značení pro chybu sítě vůči očekávaným výsledkům. Regulátor E_w je definován jako $\sum_{\forall w} w^2$. Čím vyšší hodnoty vah, tím větší bude tento člen. To způsobí větší hodnotu chybové funkce, což daný model činí méně pravděpodobným, neboť preferujeme nízké hodnoty vah.

Koeficienty α, β jsou hyperparametry, které budeme měnit během inference modelů (tj. během druhé části inference). Váhy w se počítají jako první pomocí následující inference, korespondující s funkcí S z rovnice 5.7:

$$P(w|\alpha, \beta, D) = \frac{P(D|w, \beta) \cdot P(w|\alpha)}{P(D|\alpha, \beta)} \tag{5.8}$$

Pokud uvažujeme Gaussovo rozdělení pravděpodobnosti v datech, pak můžeme jednotlivé členy spočítat jako:

$$P(w|\alpha) = \frac{1}{Z_w(\alpha)} \cdot exp(-\alpha \cdot E_w) \tag{5.9}$$

$$Z_w(\alpha) = \int exp(-\alpha \cdot E_w)\mathrm{d}w \tag{5.10}$$

$$P(D|w, \beta) = \frac{1}{Z_d(\beta)} \cdot exp(-\beta \cdot E_d) \tag{5.11}$$

$$Z_d(\beta) = \int exp(-\beta \cdot E_d)\mathrm{d}D \tag{5.12}$$

Složením těchto poznatků víme, jak provést inferenci pro váhy z rovnice 5.8:

$$P(w|\alpha, \beta, D) = \frac{exp(-\beta \cdot E_d) \cdot exp(-\alpha \cdot E_w)}{Z_s(\alpha, \beta)} \tag{5.13}$$

$$Z_s(\alpha, \beta) = \int exp(-\beta \cdot E_d - \alpha \cdot E_w)\mathrm{d}w \tag{5.14}$$

Máme váhy. V dalším musíme vytvořit lepší model (s kvalitnějšími hyperparametry α, β). Opět je můžeme získat přes integrál. Jelikož ale musíme zároveň integrovat přes všechny hyperparametry, vznikne například:

$$P(w|D) = \frac{1}{P(D)} \cdot \iint P(D|w, \beta) \cdot P(w|\alpha) \cdot P(\beta) \cdot P(\alpha)\mathrm{d}\alpha\mathrm{d}\beta \tag{5.15}$$

Toto je náročný přístup, a proto zvolíme aproximaci:

$$\gamma = \sum_{i=1}^{m} m - \alpha \cdot trace(A^{-1}) \tag{5.16}$$

$$\alpha_{new} = \frac{\gamma}{2 \cdot E_w} \tag{5.17}$$

$$\beta_{new} = \frac{n - \gamma}{2 \cdot E_d} \tag{5.18}$$

Hodnota m udává počet parametrů. Matice A je Hessova matice, *trace* je funkce počítající sumu prvků na hlavní diagonále dané matice. Učení probíhá následovně:

1. Zvol si náhodně počáteční hodnotu hyperparametrů α, β a řídícího parametru pro vektor vah w.
2. Minimalizuj $S(w)$ z rovnice 5.7 nalezením co nejvhodnějších vah (uprav vektor vah w pomocí rovnice 5.8).
3. Spočítej nové γ , s jehož pomocí vyčísli nové hodnoty hyperparametrů α, β.
4. Pokud nebylo dosaženo požadované konvergence, vrať se na první bod.

Pokud máme naučenou síť, pak je připravená k použití. Ze vstupu x chceme pomocí sítě naučené na datech D získat výstup y. To získáme jako integrál přes všechny váhy vzhledem k jejich pravděpodobnosti. Zapsáno formálně:

$$P(y|x, D) = \int P(y|x, w) \cdot p(w|D) \mathrm{d}w \tag{5.19}$$

Dobrou zprávou je, že veškeré integrály se dají počítat pomocí metody Monte Carlo (s využitím Markovova řetězce)[13]. I tak tento způsob představuje velmi náročný algoritmus, ale dobrou zprávou je, že v nedávné době vznikl nový algoritmus, který je značně rychlejší a jednodušší na pochopení. Tento algoritmus, postavený na *Shannonově entropii*, popisuji v následující sekci.

5.3 Kullback-Leiblerova divergence

V současnosti na popularitě nabývá přístup [15] postavený na entropii využívající *Kullback-Leiblerovu* divergenci. Nejprve se ovšem podívejme na pojem *očekávání*. V Bayesových neuronových sítích pracujeme s náhodnými proměnnými, které jsou určeny svojí funkcí hustoty. Otázkou je, jak z této proměnné získat jednu konkrétní hodnotu, a to nejpravděpodobnějšího reprezentanta. Právě toto řeší funkce *očekávání*.

Očekávání se dá chápat jako průměr pro spojitou funkci. Průměr diskrétní funkce je jednoduchý – použije se vážený průměr všech možných stavů náhodné proměnné, kde váhy jsou udány pravděpodobností výskytu dané možnosti (čím pravděpodobnější možnost, tím větší váha), tj.

$$\mu = \sum_{i=1}^{n} q_i p_i \tag{5.20}$$

Kde Q je množina všech možností definovaná jako $Q = \{q_1, q_2, \cdots, q_n\}$ a $p(q_i) = p_i$. Rozšíření pro spojitou variantu probíhá obvyklým způsobem. Místo jednotlivých vzorků budeme integrovat spojitou proměnnou q. Místo jednotlivých pravděpodobností užijeme funkci hustoty $f(q)$. Takto vytvořenou funkci nazveme *očekáváním*.

$$E(Q) = \int_{-\infty}^{\infty} f(q) \cdot q \, dq \tag{5.21}$$

Dále připomeňme *Shannonovu entropii* 3.12, která znázorňovala, kolik v průměru bitů informace je potřeba dodat, abychom popsali neurčitost systému. Ideální situace byla nulová entropie pro deterministické systémy. Nyní uvažme, že kromě daného systému máme i jiný, referenční systém. Kullback-Leiblerova divergence (dále *KL*) říká, jak moc se zkoumaný systém liší od toho referenčního a využívá se právě entropií. Je-li divergence nulová, pak je systém zcela stejný jako náš referenční. Čím více je divergence vzdálená od nuly, tím víc se zkoumaný systém liší od referenčního. KB divergence není symetrická, a proto není metrikou.

Něchť p je zkoumaný systém, q referenční systém a $X = \{x_1, x_2, \cdots, x_n\}$ je množina přípustných stavů systému (daných pravděpodobnostním rozdělením). KL divergenci spočítáme jako rozdíl entropií zkoumaného a referenčního systému.

$$D_{KL}(p \parallel q) = \sum_{i=1}^{n} p(x_i) \cdot (\log_2 p(x_i) - \log_2 q(x_i)) = \sum_{i=1}^{n} p(x_i) \cdot \log_2 \frac{p(x_i)}{q(x_i)} \tag{5.22}$$
$$pozn. \quad D_{KL}(p \parallel q) \neq D_{KL}(q \parallel p)$$

Pokud pracujeme se spojitými veličinami, pak KL divergenci počítáme pomocí funkce hustoty (místo pravděpodobnostního rozdělení):

$$D_{KL}(p \parallel q) = \int_{-\infty}^{\infty} p(x) \cdot \log_2 \frac{p(x)}{q(x)} dx \tag{5.23}$$

S využitím očekávání můžeme konečně přepsat KL divergenci jako

$$D_{KL}(p \parallel q) = E[\, \log_2 p(x) - \log_2 q(x) \,] \tag{5.24}$$

Pokud se vrátíme k příkladu jednoho hodu hrací kostkou a budeme zkoumat chování vůči rovnoměrnému rozdělení, pak je divergence:

$$D_{KL}(Hod\_kostkou \parallel Uniform) = 6 \cdot (\frac{1}{6} \cdot \log_2 \frac{1/6}{1/6}) = 0 \tag{5.25}$$

Z toho vyplývá, že hod kostkou je shodný s rovnoměrným rozdělením. Obecně lze použít libovolný základ pro logaritmy, úlohu hledání extrémů to nijak neovlivní – základ dva pouze opět umožňuje interpretaci výsledku na bity. Pokud bychom zkoumali podvrženou kostku, kde šestka padá dvakrát častěji, ale jednička a dvojka pouze s poloviční frekvencí, vyšlo by:

$$D_{KL}(Hod\_kostkou \parallel Uniform) = 3 \cdot (\tfrac{1}{6} \cdot \log_2 \tfrac{1/6}{1/6}) + 2 \cdot (\tfrac{1}{12} \cdot \log_2 \tfrac{1/12}{1/6}) + \tag{5.26}$$
$$+1 \cdot (\tfrac{1}{3} \cdot \log_2 \tfrac{1/3}{1/6}) = 3 \cdot 0 + 2 \cdot -0,083 + 1 \cdot 0,333 = 0.166$$

V předchozím jsem používal frázi *referenční model* místo *referenčního rozdělení*. To má samozřejmě důvod. Jako referenční model lze užít nejen pravděpodobnostní rozdělení (popř. funkci hustoty), ale i libovolný složitější model. Zejména zajímavou možností pro tuto práci je možnost použití umělé neuronové sítě. Nechť $p(x, z)$ je pravděpodobnost zadaných dat x a řídících parametrů sítě z (obvykle váhy neuronů a biasy). Nechť $q(z; \lambda)$ je pravděpodobnost parametrů sítě (z) vzhledem k parametrům hustoty rozdělení referenčního modelu (λ).

V síti provádíme inferenci proměnných iteračně jako zlepšení odhadů parametrů modelu, ze kterého vyvodíme lepší řídící parametry sítě. Dle algoritmu implementovaného v modulu Edward [16] to provedeme následovně:

$$\lambda^* = arg \min_{\lambda} KL(q(z; \lambda) \parallel p(z \mid x)) \tag{5.27}$$

Po přepsání do formy obsahující očekávání získáme:

$$\lambda^* = arg \min_{\lambda} E_{q(z;\lambda)}[\log q(z; \lambda) - \log p(z \mid x)] \tag{5.28}$$

Jak si můžeme všimnout, v rovnici vystupuje posterior pro řídící parametry sítě (počítaný i v rovnici 5.1)– tzn. váhy po inferenci. Pokud bychom měli tento člen – $\log p(z \mid x)$ – tak bychom nejspíše postupovali od výsledku k zadání. Obvykle ale s výsledkem nezačínáme. Nic nám ovšem nebrání posterior na začátku vygenerovat náhodně a iterativně se ke správnému výsledku přibližovat.

Na pravděpodobnost parametrů půjdeme, jak už několikrát, přes pravděpodobnost dat. Platí, že:

$$\log p(x) = KL(q(z; \lambda) \parallel p(z|x)) + E_{q(z;\lambda)}[\log p(x, z) - \log q(z; \lambda)] \tag{5.29}$$

V jedné iteraci jsou parametry modelu λ konstantní, a proto můžeme zanedbat první člen rovnice (obdobně, jako jsme zanedbávali člen *evidence* v předchozích případech). Zbylý výraz, nazývaný *ELBO – Evidence Lower Bound* – rozepíšeme jako

$$\begin{aligned} ELBO(\lambda) &= E_{q(z;\lambda)}[\log p(x, z) - \log q(z; \lambda)] = \\ &= E_{q(z;\lambda)}[\log p(x, z)] - E_{q(z;\lambda)}[\log q(z; \lambda)] \end{aligned} \tag{5.30}$$

Zamysleme se nad rovnicí 5.30. První člen lze reprezentuje energii systému – schopnost konat práci při zadaných parametrech. Druhý člen udává entropii našeho modelu q. Inference spočívá v maximalizaci prvního člene, což zlepšuje chování na základě pravděpodobnostního modelu zkušebních dat. Rovněž se minimalizuje druhý člen – tím se zabraňuje přeučení, neboť příliš složitý model nebude jednoduché popsat pomocí našich parametrů λ. Čím složitější model, tím víc se blíží výstup našeho modelu k šumu, který má maximální entropii. Lze vidět podobnost s přístupem z rovnice 5.7.

Nové parametry modelu získáme jako:

$$\lambda^* = arg \max_{\lambda} ELBO(\lambda) = E_{q(z;\lambda)}[\log p(x, z) - \log q(z; \lambda)] \tag{5.31}$$

Řešení tohoto problému je opět pomocí aproximace (např. metodou Monte Carlo pro S vzorků):

$$\nabla_\lambda ELBO(\lambda) \approx \tag{5.32}$$
$$\approx \frac{1}{S}\sum_{s=1}^{S}[(\log p(x, z_s) - \log q(z_s;\lambda)\nabla_\lambda \log q(z_s;\lambda)]$$

V této rovnici jsou z_s vzorky očekávání z rozdělení $q(z;\lambda)$. Celkem je S vzorků.

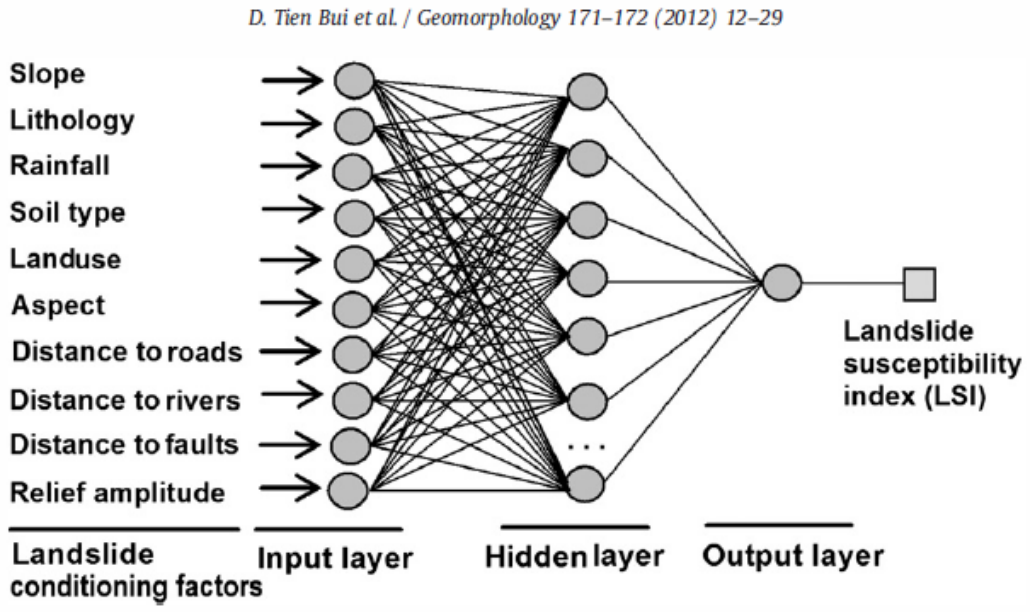
Kapitola 6

Využití sítě

Bayesovské neuronové sítě mají oproti klasickým sítím několik výhod. Lépe zobecňují a jsou méně náchylné k přeučení. V této kapitolce rozeberu práci, která na konkrétním příkladě demonstruje výhody Bayesovy neuronové sítě před klasickou. Práce využívá algoritmu učení podle 5.7.

6.1 Příklad použití

Příklad pochází z práce od *D. Tien Bui* [3], který se zaměřil na klasifikaci oblastí ve Vietnamu, ve kterých hrozí sesuvy půdy. Sesuvy půdy jsou ve provincii Hoa Binh (ta se nachází zhruba uprostřed mezi západní hranicí s Laosem a severovýchodní hranicí s Čínou) častým jevem. Jen za posledních 10 let bylo v oblasti detekováno 118 sesuvů, s průměrným rozsahem 3440m^2.



Obrázek 6.1: Síť navržená v práci [3]

Autoři se rozhodli vytvořit neuronovou síť, která na základě vstupů reprezentujících deset geologických a geografických atributů vrátí ve svém výstupu index, který bude na základě své hodnoty reprezentován jako:

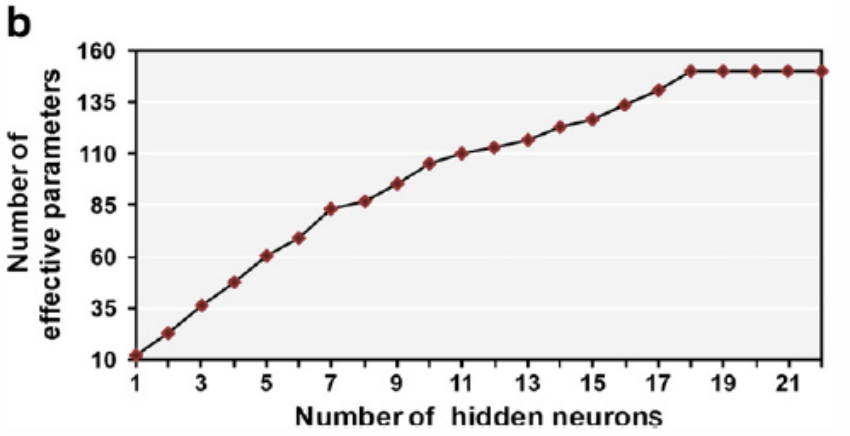
1. V této oblasti je velmi malé riziko sesuvu půdy
2. V této oblasti je malé riziko sesuvu půdy
3. V této oblasti je střední riziko sesuvu půdy
4. V této oblasti je vysoké riziko sesuvu půdy

Jedná se o regresní úlohu, kterou se autoři rozhodli zpracovat pomocí neuronové sítě, viz. obrázek 6.1. Zajímavé je, že se rozhodli pro dvě metody učení a následné klasifikace sítě – jak klasickou metodu *backpropagation*, tak Bayesovu metodu.

Vstupy byly rovnoměrně převedeny na čísla z intervalu $\langle 0.1; 0.9 \rangle$. Například úhrn srážek $470 - 540mm$ znamená, že na vstup neuronu vstupní vrstvy reprezentující srážky (*Rainfall*) přijde číslo 0.37. Celá plocha provincie byla diskretizována na čtvercové oblasti o délce hrany $20m$. Tímto rozdělením bylo získáno mnohem větší množství dat (na čím větší ploše sesuv byl, tím více nových záznamů po přerozdělení vzniklo).

Při trénování klasickým přístupem byla data rozdělena na: 70% dat trénovacích, 15% validačních a zbylých 70% dat bylo testovacích. Nejlepší síť obsahovala osmnáct neuronů ve skryté vrstvě. Trénování probíhalo několikrát, vždy s jinými počátečními vahami. Teprve poté se změnil model přidáním dalšího neuronu do skryté vrstvy.

Zajímavé je, že Bayesova neuronová síť skutečně dokáže nalézt nejjednodušší model implicitně. Jak lze vidět na obrázku 6.2, síť se ustálila na 18 skrytých neuronech. Přidání dalšího neuronu vedlo po konvergenci inference k tomu, že další neuron byl v síti ignorován.



Obrázek 6.2: Konvergence parametrů v Bayesově neuronové síti podle práce [3]

Obě sítě samozřejmě zjistily očekávané výsledky, a to že na sesuvu půdy se nejvíce podílí sklon oblasti, typ půdy, úhrn srážek a využití půdy – les, zemědělská půda apod. Otázkou ovšem je, jak moc přesně tyto sítě reagovaly na testovací data. Klasická neuronová síť měla úspěšnost 86,1%, zatímco Bayesova neuronová síť byla úspěšnější, neboť dosahovala přesnosti 90,3%.

Kapitola 7

Demonstrace BNN

Tato kapitola pojednává o demonstračním programu, který byl vyvinut jako produkt k této práci. Program má formát skriptu v programovacím jazyce Python. Důvod výběru tohoto jazyka je objastněn v sekci *Předpoklady pro vývoj*. Popis výsledného skriptu podává sekce *Demonstrační program*.

Následující sekce se zabývají experimenty s programem, které poukazují na některé specifika Bayesovy neuronové sítě a srovnává je s výsledky běžné umělé neuronové sítě.

7.1 Předpoklady pro vývoj

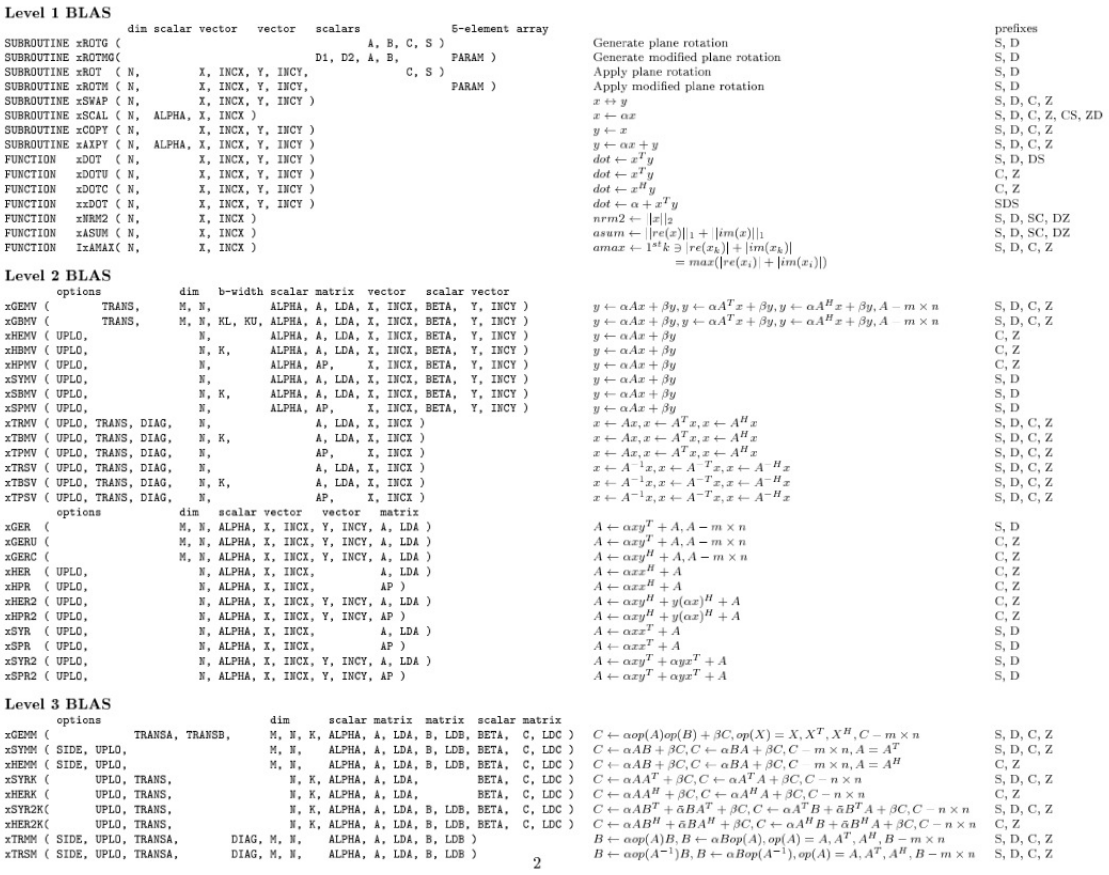
Neuronové sítě jsou jako model strojového učení náročné na prostředky, neboť s rostoucí sítí výrazně narůstá množství operací, které je třeba provést. To znamená, že pro vývoj aplikací pracující se sítěmi je vhodné mít schopnost provádět aritmetické operace efektivně. Zároveň je pro potřeby demonstrace výhodné mít možnost síť vhodně abstrahovat, aby nedocházelo k chybám a bylo možné v rozumném čase sítě konstruovat a ze zdrojového kódu pochopit. Tyto požadavky s využitím vhodných modulů splňuje právě jazyk Python.

Knihovna pro efektivní výpočty, kterou jsem zvolil, se jmenuje *NumPy* [17]. Tento modul při své instalaci využívá zásadních matematických zdrojů, jako jsou *Blas* a *Lapack*. *Blas* (Basic Linear Algebra Subprograms) a *Lapack* (Linear Algebra Package) jsou zdrojové kódy psané v jazyce Fortran, které se snaží co nejefektivnějším způsobem řešit běžné matematické problémy, viz. obr. 7.1

Samozřejmě existují i komerční rozšíření *Blas*, která se snaží nabízené operace dále optimalizovat. Já se spokojil se základní verzí. Nad těmito zdroji vznikne *NumPy*, modul pro matematické výpočty, který se v mnohém snaží napodobit *MatLAB*. Zásadním předpokladem modulu je to, že se pracuje s multidimenzionálními poli.

Nad tímto modulem je postaven modul *TensorFlow* [2]. Jak název napovídá, jedná se o modul pro práci s tenzory – zobecněný více-dimenzionální vektor. Tento modul je schopen přenést vhodné operace z CPU do GPU, což vede k dalšímu zrychlení výpočtů (v době psaní práce *TensorFlow* podporuje pouze čipy NVIDIA®). Modul dále poskytuje operace, které lze s výhodou využít v běžných neuronových sítích.

Posledním krokem je zastřešení těchto operací přímo pod sémantikou neuronových sítí. To zajišťuje modul *keras* [4]. Pomocí tohoto modulu je možné snadno tvořit neuronové sítě dle následujícího diagramu.



Level 1 BLAS

	dim	scalar	vector	vector	scalars	5-element array		prefixes
SUBROUTINE xROTG (					A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE xROTMG(					D1, D2, A, B,	PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (	N,		X, INCX,	Y, INCY,	C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (	N,		X, INCX,	Y, INCY,		PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP (	N,		X, INCX,	Y, INCY)			$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL (	N,	ALPHA,	X, INCX)				$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY (	N,		X, INCX,	Y, INCY)			$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY (	N,	ALPHA,	X, INCX,	Y, INCY)			$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC (	N,		X, INCX,	Y, INCY)			$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT (	N,		X, INCX,	Y, INCY)			$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2 (	N,		X, INCX)				$nrm2 \leftarrow \|x\|_2$	S, D, SC, DZ
FUNCTION xASUM (	N,		X, INCX)				$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$	S, D, SC, DZ
FUNCTION IxAMAX(	N,		X, INCX)				$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

options		dim	b-width	scalar	matrix	vector	scalar	vector		
xGEMV (	TRANS,	M, N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV (	TRANS,	M, N,	KL, KU,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xHEMV (UPLO,		N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV (UPLO,		N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV (UPLO,		N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV (UPLO,		N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV (UPLO,		N,	K,	ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV (UPLO,		N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV (UPLO,	TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV (UPLO,	TRANS, DIAG,	N,	K,		A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV (UPLO,	TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV (UPLO,	TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTBSV (UPLO,	TRANS, DIAG,	N,	K,		A, LDA,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTPSV (UPLO,	TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z

options	dim	scalar	vector	vector	matrix		
xGER (	M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU (	M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC (	M, N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
xHER (UPLO,	N,	ALPHA,	X, INCX,		A, LDA)	$A \leftarrow \alpha xx^H + A$	C, Z
xHPR (UPLO,	N,	ALPHA,	X, INCX,		AP)	$A \leftarrow \alpha xx^H + A$	C, Z
xHER2 (UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2 (UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	AP)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR (UPLO,	N,	ALPHA,	X, INCX,		A, LDA)	$A \leftarrow \alpha xx^T + A$	S, D
xSPR (UPLO,	N,	ALPHA,	X, INCX,		AP)	$A \leftarrow \alpha xx^T + A$	S, D
xSYR2 (UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	A, LDA)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
xSPR2 (UPLO,	N,	ALPHA,	X, INCX,	Y, INCY,	AP)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Level 3 BLAS

options			dim	scalar	matrix	matrix	scalar	matrix		
xGEMM (	TRANSA,	TRANSB,	M, N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha op(A)op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
xSYMM (SIDE, UPLO,			M, N,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S, D, C, Z
xHEMM (SIDE, UPLO,			M, N,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
xSYRK (	UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
xHERK (	UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
xSYR2K(	UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB^T + \bar{\alpha} BA^T + \beta C, C \leftarrow \alpha A^T B + \bar{\alpha} B^T A + \beta C, C - n \times n$	S, D, C, Z
xHER2K(	UPLO,	TRANS,	N, K,	ALPHA,	A, LDA,	B, LDB,	BETA,	C, LDC)	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C, Z
xTRMM (SIDE, UPLO, TRANSA,		DIAG,	M, N,	ALPHA,	A, LDA,	B, LDB)			$B \leftarrow \alpha op(A)B, B \leftarrow \alpha Bop(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
xTRSM (SIDE, UPLO, TRANSA,		DIAG,	M, N,	ALPHA,	A, LDA,	B, LDB)			$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha Bop(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

2

Obrázek 7.1: Operace dostupné v *Blas*, podle http://www.netlib.org/blas/

Nejprve je potřeba definovat síť. Keras neuronovou síť nejčastěji definuje po vrstvách. Tento model se nazývá *Sequential* a očekává, že bude následovat definice jednotlivých vrstev směrem od vstupů k výstupům. V každé vrstvě je možné definovat propojeni s vrstvou předchozí. Nejčastější případ je plné propojení značené *Dense*. Plná hlavička je:

```
keras.layers.core.Dense(units, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=
    None, kernel_constraint=None, bias_constraint=None)
```

Parametr *units* udává počet neuronů ve vrstvě. Velmi důležitý je i parametr *activation* udávající aktivační funkci neuronů v dané vrstvě. Do první vrstvy vždy přidáme i informaci udávající dimenzi vstupních dat (*input_dim*).

Na následujícím příkladu je vytvořena čtyřvrstvá síť. Dimenze vstupů je 3 (např. plat, věk, rodinný stav) a výstup bude jediný (např. schopnost splácet). První vrstvě předáme informaci o třech dimenzích, v poslední (výstupní) vrstvě necháme jediný neuron. Aktivační funkce jsou $relu(x) = max(0, x)$ a $sigmoid(x) = \frac{1}{1+e^{-x}}$.

```
model = Sequential()
model.add(Dense(6, input_dim=3, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Dále sestavíme neuronovou síť pomocí jejího modelu. K tomu slouží klíčové slovo *compile*. Ten požaduje tři parametry. Prvním je typ užitého optimalizátoru (které například mění koeficient učení či se uměle pokouší zabraňovat přeučení). Druhým parametrem je funkce, kterou se síť snaží minimalizovat. Poslední parametr je seznam sbíraných metrik.

```
#ukazky pouziti

# klasifikace do n trid
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# problem regrese pomoci stredni kvadraticke odchylky
model.compile(optimizer='adam',
              loss='mean_squared_error')
```

Vzniklou síť je potřeba naučit na trénovacích datech. K tomu slouží funkce *fit*. Důležité je předání vstupních dat (správné dimenze), očekávaných výstupních dat a počet epoch učení – učení zde probíhá uživatelem stanovený počet iterací.

```
fit(self, x, y, batch_size=32, epochs=10, verbose=1, callbacks=None,
    validation_split=0.0, validation_data=None, shuffle=True,
    class_weight=None, sample_weight=None, initial_epoch=0)

# outcome obsahuje sbirane metriky, jedna se o objekt History
outcome = model.fit(VstupyTrenovacichDat, VystupyTrenovacichDat, epochs
    =1000)
```

Dále se vyhodnotí úspěšnost učení. Je možné použít trénovací data a zjistit, jestli se síť byla schopna všechna naučit. Samozřejmě lze použít i validační data, jsou-li k dispozici. Vyhodnocení probíhá funkcí *evaluate*, která bere vstupy a očekávané výstupy.

```
# outcome necht je napr. loss, accuracy
outcome = model.evaluate(X, Y)
# tisk accuracy
print(model.metrics_names[1], scores[1]))
```

V tuto chvíli je k dispozici naučená analyzovaná síť, kterou lze použít k predikci na nových datech. K tomu slouží funkce *predict*, která očekává vstupy a vrací sítí odvozený výstup.

```
Y = model.predict(X)
```

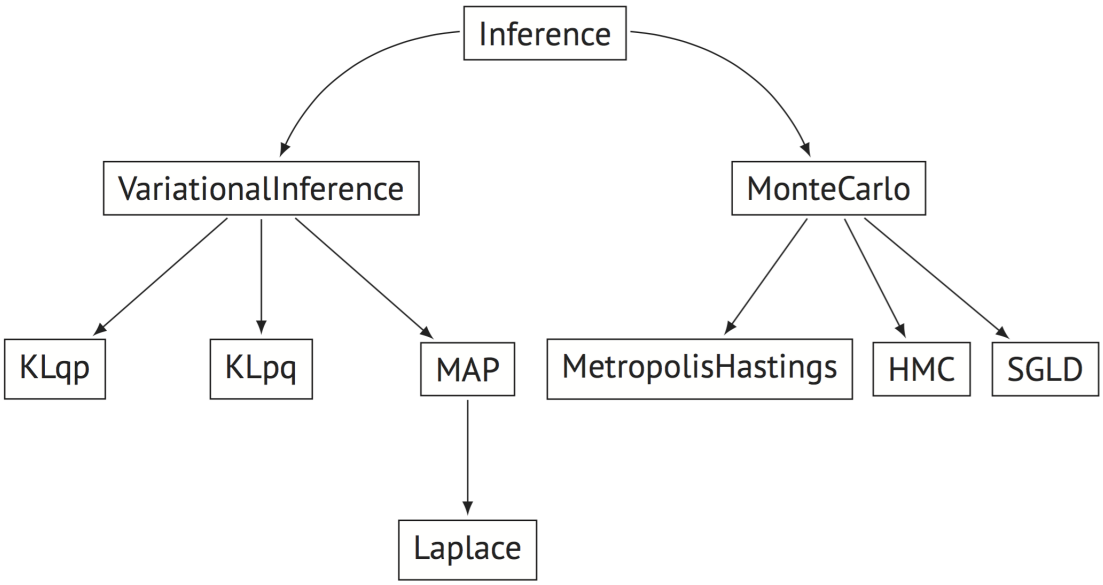
Tímto způsobem lze pomocí modulu *keras* tvořit umělé neuronové sítě. Pro srovnání s Bayesovými neuronovými sítěmi by bylo vhodné opět využívat *TensorFlow*. Dobrou zprávou je, že pro práci s náhodnými jevy (proměnnými) a pro Bayesovskou inferenci rovněž existuje modul postavený nad *TensorFlow*. Tento modul se jmenuje *Edward* [16].

Tento modul řeší problém inference jako rovnici

$$P(z|x) = \frac{P(x,z)}{\int P(x,z)dz} \tag{7.1}$$

X jsou pozorovaná data, z jsou parametry sítě, které nastavujeme. Snaha je dostat co nejpravděpodobnější model. Nejtěžší problém v této rovnici je výpočet integrálu, avšak *edward* poskytuje řadu algoritmů pro řešení (pomocí aproximace) tohoto problému v inferenci, viz. 7.2.

Pokud budeme chtít simulovat Bayesovskou neuronovou síť, pak postupujeme následovně:



Obrázek 7.2: Edward – implementované algoritmy pro inferenci, převzato z: [16]

1. Budeme uvažovat parametry sítě (váhy, biasy) jako náhodné veličiny s počátečním normálním rozdělením pravděpodobnosti
2. Vstupní data nechť jsou ve tvaru $\{(x_n, y_n)\}$, kde $x_n \in \mathbb{R}^F$, kde F je dimenze vstupních dat.
3. Nechť σ^2 je známá odchylka

Pak pravděpodobnost každého vstupu definujeme jako

$$P(y_n|x_n, z, \sigma^2) = Normal(y_n|NN(x_n; z), \sigma^2) \tag{7.2}$$

Symbol z opět značí řídící parametry sítě. NN je model Bayesovské neuronové sítě, který funguje kupříkladu následovně:

$$\forall n : z_n \sim Normal(z_n|0, I) \tag{7.3}$$

$$x_n|z_n \sim Bernoulli(x_n|p = NN(z_n; \theta)) \tag{7.4}$$

Tyto rovnice říkají, že *priory* nastavíme z normálního rozdělení jako náhodnou proměnnou a *likelihood* získáme z modelu sítě s proměnnými parametry z a neměnnými parametry θ.

Tím jsme vytvořili vzájemný vztah mezi daty a modelem. Iterativní volání inference se postará o to, abychom dostávali čím dál více pravděpodobnější model. Důkaz efektivity tohoto přístupu je předložen v práci [15].

Dále uvedu demonstrativní kód pro tvorbu Bayesovské neuronové sítě. Síť definuji manuálně jako:

```
import tensorflow as tf
def neural_network(x, W_0, W_1, W_2, b_0, b_1, b_2):
  h = tf.tanh(tf.matmul(x, W_0) + b_0)
  h = tf.tanh(tf.matmul(h, W_1) + b_1)
  h = tf.matmul(h, W_2) + b_2
  return tf.reshape(h, [-1])
```

Tímto jsem vytvořil třívrstvou neuronovou síť. První dvě vrstvy mají jako aktivační funkci hyperbolický tangens. Poslední vrstva má aktivační funkci $f(x) = x$. Všechny vstupní parametry budou tenzory z *TensorFlow*, a proto mohou nabývat libovolné dimenze (např. *W_1* bude mít takovou dimenzi, kolik je neuronů v příslušné druhé vrstvě). Teď zbývá použít modul *edward* pro přidání neurčitosti do modelu.

Vytvořím *priory* a *posteriory* řídících proměnných sítě. Vstup bude mít jen jednu dimenzi, první dvě vrstvy budou mít pět neuronů, výstupní vrstva bude mít jeden neuron. Zřejmě tvořím model pro jednoduchou funkci, kde jediné y závisí jen na x. Počáteční hodnoty *priorů* jsou z normálního rozdělení se střední hodnotou 0 (v kódu *loc*) a odchylkou jedna (v kódu *scale*).

```
from edward.models import Normal
#priors
#z jednodimenzionalniho vstupu na pet neuronu prvni vrstvy
W_0 = Normal(loc=tf.zeros([1, 5]), scale=tf.ones([1, 5]))
#z peti neuronu prvni vrstvy na pet neuronu druhe vrstvy
W_1 = Normal(loc=tf.zeros([5, 5]), scale=tf.ones([5, 5]))
#z peti neuronu druhe vrstvy na jedinny neuron vystupni vrstvy (
    jednodimenzionalni vystup)
W_2 = Normal(loc=tf.zeros([5, 1]), scale=tf.ones([5, 1]))
#biasy peti neuronu prvni vrstvy
b_0 = Normal(loc=tf.zeros(5), scale=tf.ones(5))
#biasy peti neuronu druhe vrstvy
b_1 = Normal(loc=tf.zeros(5), scale=tf.ones(5))
#biasy jednoho neuronu treti (vystupni) vrstvy
b_2 = Normal(loc=tf.zeros(1), scale=tf.ones(1))
#posteriors
qW_0 = Normal(loc=tf.Variable(tf.random_normal([1, 5])),
              scale=tf.nn.softplus(tf.Variable(tf.random_normal([1, 5])))
                  )
qW_1 = Normal(loc=tf.Variable(tf.random_normal([5, 5])),
              scale=tf.nn.softplus(tf.Variable(tf.random_normal([5, 5])))
                  )
qW_2 = Normal(loc=tf.Variable(tf.random_normal([5, 1])),
              scale=tf.nn.softplus(tf.Variable(tf.random_normal([5, 1])))
                  )
qb_0 = Normal(loc=tf.Variable(tf.random_normal([5])),
              scale=tf.nn.softplus(tf.Variable(tf.random_normal([5]))))
qb_1 = Normal(loc=tf.Variable(tf.random_normal([5])),
              scale=tf.nn.softplus(tf.Variable(tf.random_normal([5]))))
qb_2 = Normal(loc=tf.Variable(tf.random_normal([1])),
              scale=tf.nn.softplus(tf.Variable(tf.random_normal([1]))))
```

Dále zadefinuji, jak vzniká výstup sítě. I výstup bude samozřejmě náhodná proměnná. N značí počet trénovacích vzorků dat.

```
#nacti trenovaci data do x_train (tensor vstupu) a do y_train (tensor
    ocekavanych vystupu) - v tomto pripade jednodimenzionalni tensor (=
    vektor) s N prvky
x = x_train
y = Normal(loc=neural_network(x, W_0, W_1, W_2, b_0, b_1, b_2),
           scale=0.1 * tf.ones(N))
```

Učení probíhá jako posloupnost inferencí dle algoritmu Kullback–Leibler. Inference hledá nejpravděpodobnější model. Řídící parametry sítě jsou v prvním argumentu jako *prior* :*posterior*, druhý argument je seznam cílových dat ve formátu *náhodná proměnná*:*ideální hodnota*. Metoda *KLqp* vybere nejvhodnější metodu řešení (viz. obrázek 7.2 – nutno apro-

ximovat integrály) pro počítání algoritmem Kullback-Leibler. Počet průchodů algoritmem definuji v hodnotě *n_iter*.

```
inference = ed.KLqp({W_0: qW_0, b_0: qb_0,
                     W_1: qW_1, b_1: qb_1,
                     W_2: qW_2, b_2: qb_2}, data={y: y_train})
inference.run(n_iter=2000)
```

Tímto vznikly modely pro běžnou i pro Bayesovskou neuronovou síť. Jelikož jsou oba modely postavené nad stejným modulem *TensorFlow*, je možné tyto modely věrohodně porovnávat.

Poslední modul, který je potřeba, je modul *matplotlib* [7]. Tento modul slouží ke kreslení grafů a funkcí. Využiji ho pro vizualizaci výsledků jednotlivých experimentů.

7.2 Demonstrační program

Program vznikl v jazyce Python ve verzi 3.5 dostupného v distribuci *Anaconda* 4.2.0 [1]. Užitý modul *NumPy* byl ve verzi 1.12.1, modul *TensorFlow* ve verzi 1.1.0, *Keras* měl verzi 2.0.4 a Edward verzi 1.3.1. Verze Pythonu 3.5 byla zvolena z toho důvodu, že ve verzi 3.6 v době psaní práce není stabilně funkční modul *TensorFlow* a pro verze 3.4 a starší naopak chybí podpora modulu *Edward*.

Následující seznam popisuje přepínače programu. Žádné přepínače nejsou povinné, neboť každý z nich má implicitní hodnotu, která se použije, není-li přepínač zadán.

- přepínač `--task` (popř. `-t`) slouží pro specifikaci úkolu, který bude řešen. Hodnota *bayes* je implicitní, a proběhne demonstrace BNN na zvolených datech. Hodnota *classic* použije NN. Hodnota *both* zavolá pro porovnání jak BNN, tak i NN. Hodnota *gen* nebo *dat* (libovolná z těchto dvou) nastaví program, aby pouze vygeneroval data a uložil je do souboru.
- přepínač `--experiment` (popř. `-e`) slouží pro volbu vstupních dat. V rámci programu je k dispozici několik předpřipravených sad dat.
  - hodnota *gen*, popř. *dat* bere za vstupní data ta, která byla dříve vygenerována do souboru *bnndata.dat*.
  - *cos* slouží pro volbu dat generovaných goniometrickou funkcí cosinus
  - *lin* generuje lineárně závislá data
  - *noi* vytváří konstantní funkci (předpokládá se, že bude nastaveno zašumění dat)
  - *log* demonstruje data pro přirozený logaritmus
  - *pol*, *det* a *man* ukazuje funkci pro netriviální polynomy – tyto tři vstupy se liší způsobem zadání dat a mohou být uživateli návodem, jak zadávat data vlastní
- přepínač `--layers` (popř. `-l`) slouží pro udání počtu vrstev tvořené sítě. Minimální počet vrstev, které lze zadat, jsou dvě.
- přepínač `--neuronsperlayer` (popř. `-npl`) udává počet neuronů ve vrstvách sítě. Výjimkou je vrstva poslední (výstupní), která se nastavuje podle počtu dimenzí výstupu – v případě demonstračních úloh je výstupní dimenze jediná.

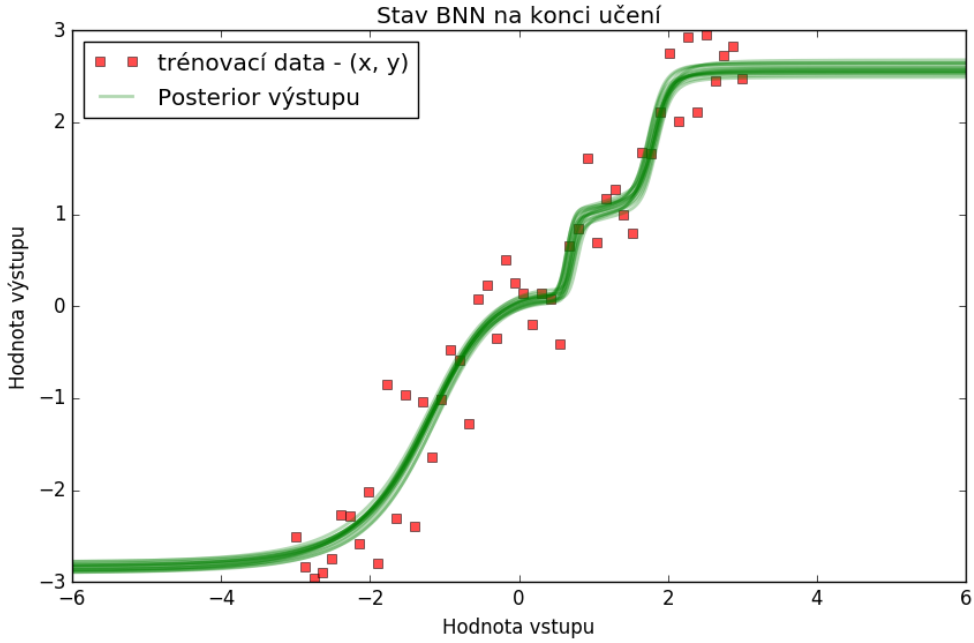
- přepínač `--inputs` (popř. `-i`) udává počet vzorků trénovacích dat. Tento parametr je bezpředmětný pro variantu, kde jsou data získána ze souboru. V ostatních případech je minální počet vzorků nastaven na tři.
- přepínač `--noise` (popř. `-n`) přidává ke vstupním datům šum. Šum je generován náhodně z normálního rozdělení se středem v nule. Hodnota tohoto přepínače udává odchylku (je očekáváno nezáporné reálné číslo).
- přepínač `--generations` (popř. `-g`) udává kladné celé číslo. Toto číslo říká počet iterací učení sítě – 1 iterace značí jeden průchod iteračním algoritmem pro učení. Pro předpřipravené úlohy s implicitně nastavenou sítí doporučuji udat počet generací alespoň v řádu jednotek tisíců – to platí jak pro NN, tak i pro BNN.
- přepínač `--outputdata` (popř. `-o1`) neočekává žádnou hodnotu. Jedná je o přepínač typu `Bool`. Jeho udáním program na začátku činnosti vykreslí data.
- přepínač `--outputinit` (popř. `-o2`) je obdobný jako přepínač `-o1`. Tento přepínač ovšem vykreslí stav sítě před tím, než je zahájeno učení.
- přepínač `--bayesmean` (popř. `-bm`) je opět přepínač bez parametru a říká, že výsledkem BNN nebudou vzorky řešení, ale očekávání.
- přepínač `--activation` (popř. `-a`) slouží pro volbu aktivační funkce BNN a NN. Mezi podporované volby patří *relu*, *sigmoid*, *softplus* a *tanh*. ReLU je implicitní hodnota.

Na obrázku 7.3 lze spatřit příklad použití programu na značně zašuměných datech generovaných lineární funkcí. Ukázka výstupu je na obrázku 7.4. V příkladu je zvolena možnost použití obou sítí. Výstup je z části Bayesovské neuronové sítě, kde jsou váhy sítě považovány jako náhodné proměnné, a proto bylo pro potřeby znázornění výsledku vzorkováno šestnáct sad těchto vah. Pravděpodobnost vzorkování váhy odpovídá pravděpodobnosti dané hodnoty váhy (tzn. pokud síť nějakou oblast řešení považuje za velmi pravděpodobnou, bude z této oblasti velmi pravděpodobně i vzorek).

Na výstupu pak lze spatřit chování sítě podle těchto šestnácti instancí sítě. Hodnotu šestnáct lze přenastavit v programu modifikací hodnoty *self.bnndrawsamples*.

```
C:\showcase>python bnnxhloze01.py -t both -e lin -n 0.5 -g 1500 -l 3 -npl 4 -a t
anh

xhloze01: BNN/NN demo - 2017 - VUT FIT

TASK: Both BNN and NN

Edward learing started
1500/1500 [100%] ██████████████████████████████ Elapsed: 6s | Loss: 615.994
Edward learing completed
Keras learing started
Keras learing completed
50/50 [==============================] - 0s
MSE BNN:  3.36021  t(H:M:S)= 0:00:06.249435
MSE NN:  0.23347979784  t(H:M:S)= 0:00:01.544086

C:\showcase>
```

Obrázek 7.3: Ukázka použití programu



Obrázek 7.4: Ukázka výstupu z programu zadaného v obr. 7.3

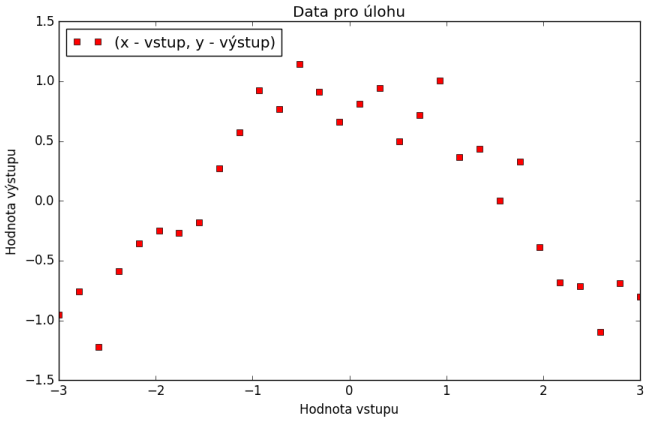
Kapitola 8

Experimenty s BNN

V této kapitole budu experimentovat s demonstračním programem na předpřipravených datech. Mým cílem je ukázat chování BNN a srovnat jej s tradičním přístupem NN.

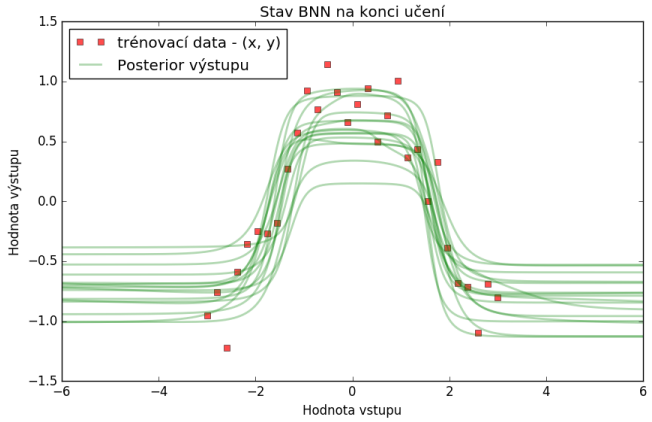
8.1 Pozorování procesu učení

Pro první experiment jsem si vygeneroval data odpovídající zašuměné funkci cosinus (obr. 8.1). Vygenerováno bylo třicet vzorků dat $(x, y) : y = \cos x + noise$. Sítě budou řešit regresní úlohu. Pokusí se zjistit, jaká funkce co nejlépe popisuje vygenerovaná data. Připomínám, že každá ze sítí si pod slovy *co nejlépe* představuje něco jiného.

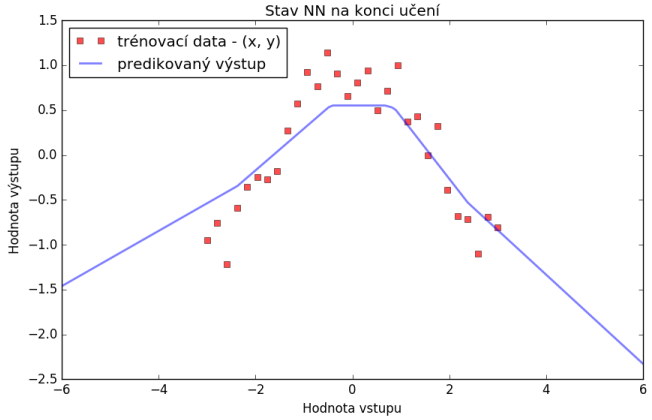


Obrázek 8.1: Vygenerovaná data

Nejprve budu pozorovat chování na implicitní síti, tj. síti se třemi vrstvami a pěti neurony v každé vnitřní vrstvě (značeno jako síť *3-5*). Porovnávat budu BNN s NN, jejíž aktivační funkce jsou *tanh* a *ReLU*. Jako první ukáži stav sítí po 250 průchodech algoritmem. Toto lze spatřit na obrázcích 8.2 a 8.3. Vidíme, že Bayesovská neuronová síť má stále velmi velký rozsah přípustných hodnot pro své váhy a výsledek je značně neurčitý, a to i v oblastech se zadanými daty. Toto je dobrým signálem k tomu, že učení ještě neskončilo. Umělá neuronová síť s neurčitostí nepracuje, a proto je k dispozici jediná křivka popisující chování sítě.



Obrázek 8.2: stav po 250 průchodech, síť 3-5, tanh

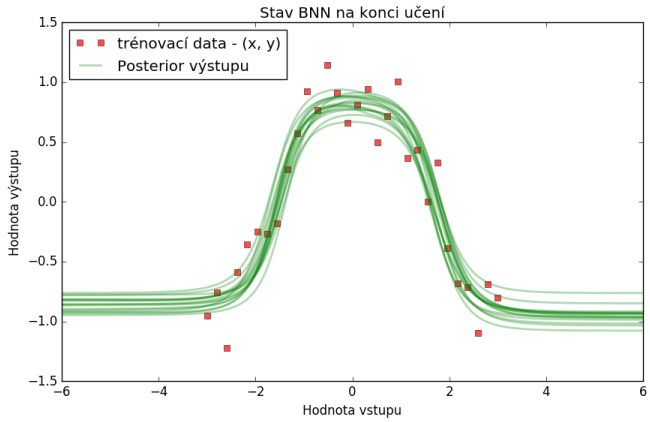


Obrázek 8.3: stav po 250 průchodech, síť 3-5, ReLU

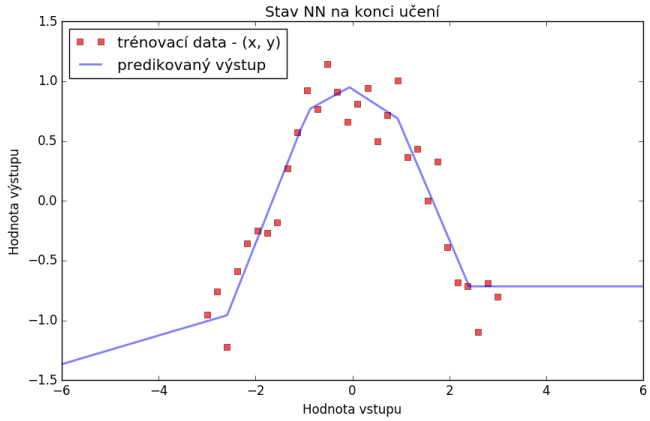
Dále ukáži stav po 1000 průchodech na obrázcích 8.4 a 8.5. Stav po 4000 průchodech je na obr. 8.6 a 8.7. Pro srovnání u ukázáno i řešení NN se 4000 průchody, pokud je jako aktivační funkce zvolena funkce SoftPlus (obr. 8.8). Z prezentovaných obrázků vyplývá princip učení. NN začíná s parametry zcela nereflektujícími data a postupně zlepšuje odhad výstupu minimalizací střední kvadratické odchylky. Oproti tomu BNN začíná ve stavu, kdy na počátku hustota výstupu postihuje všechny možné funkce (reprezentovatelné sítí), ale každou z nich jen s malou pravděpodobnosti. Během procesu učení pak zvyšuje pravděpodobnost rozdělení parametrů tak, aby síť s pravděpodobnými parametry dávala dobré výsledky vzhledem k zadaným datům. Stav z počáteční fáze učení lze vidět na obrázcích z počátku učení 8.9 a 8.10 (30 průchodů). NN začínají z *ničeho* a snaží se najít řešení zesložitěním modelu. BNN začínají ze *všeho* a řešení hledají jako zjednodušení modelu.

Pokud bych chtěl po BNN konkrétní výsledek místo funkce hustoty, pak můžu všechny náhodné proměnné – řídící parametry sítě – nahradit jejich očekáváním, jak je ukázáno na obrázku 8.11. Pokud jsou všechny náhodné proměnné reprezentující řídící parametry de-

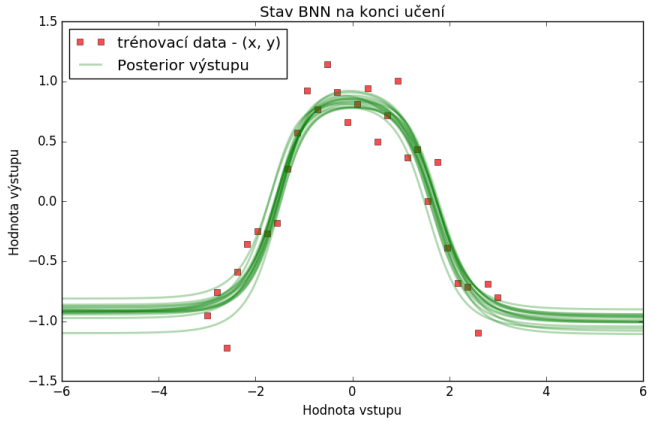
terministické, bude i výstupní náhodná proměnná deterministická (100% pravděpodobnost pro jednu hodnotu, 0% jinde).



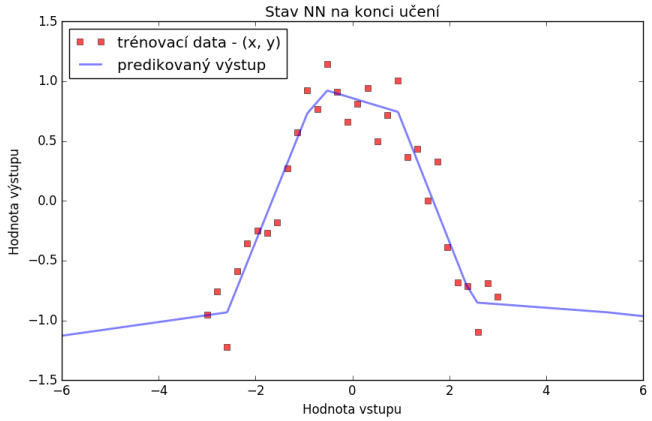
Obrázek 8.4: stav po 1000 průchodech, síť 3-5, tanh



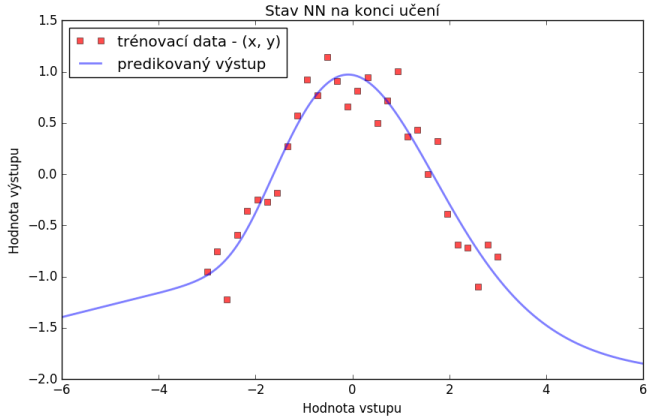
Obrázek 8.5: stav po 1000 průchodech, síť 3-5, ReLU



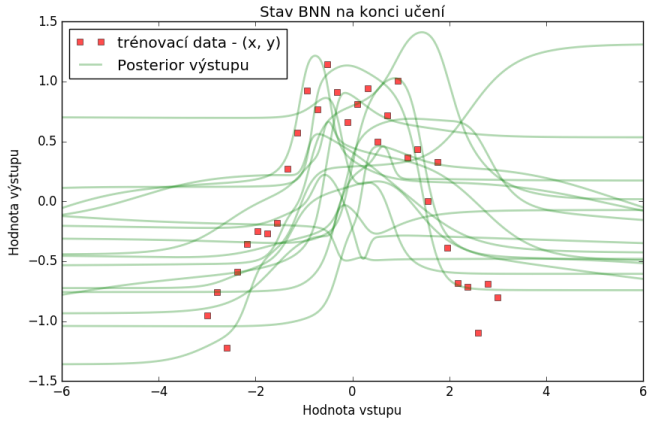
Obrázek 8.6: stav po 4000 průchodech, síť 3-5, tanh



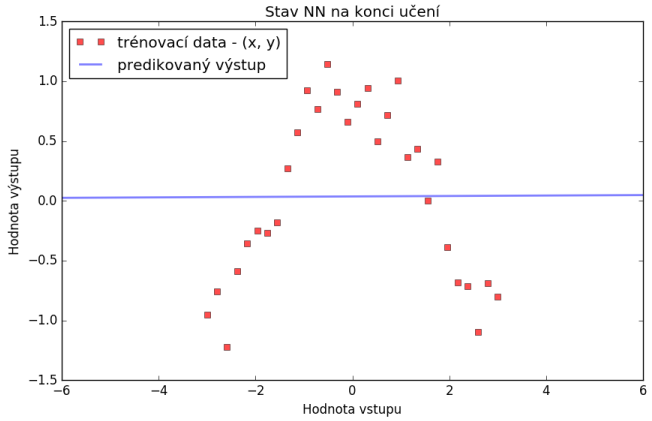
Obrázek 8.7: stav po 4000 průchodech, síť 3-5, ReLU



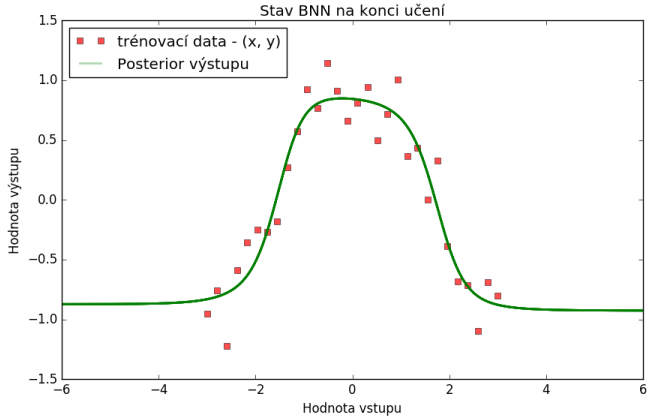
Obrázek 8.8: stav po 4000 průchodech, síť 3-5, SoftPlus



Obrázek 8.9: stav po 30 průchodech, síť 3-5, tanh



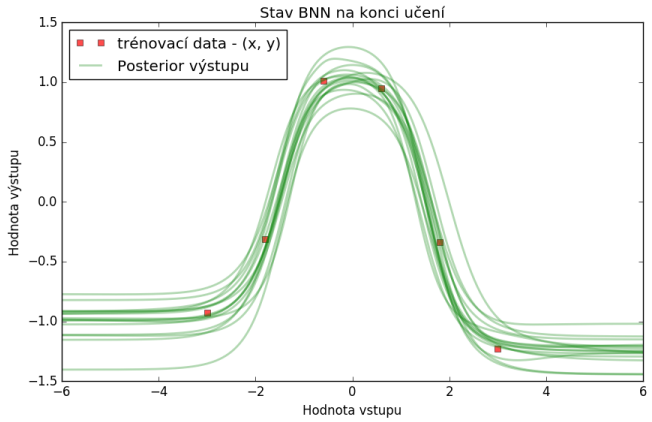
Obrázek 8.10: stav po 30 průchodech, síť 3-5, ReLu, SoftPlus i Tanh



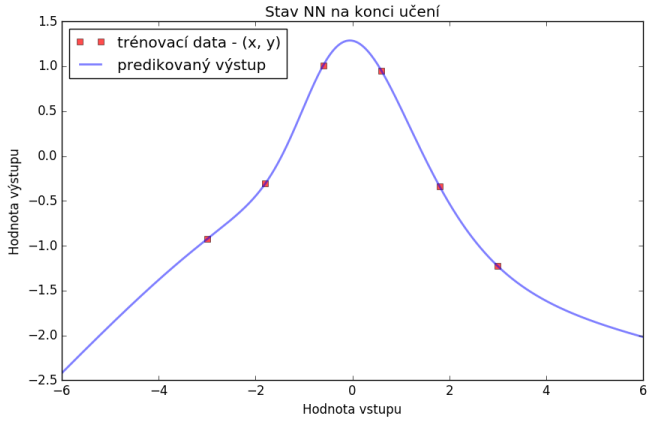
Obrázek 8.11: stav po 4000 průchodech, síť 3-5, tanh, Očekávání

8.2 Efektivita a problémy sítí

Dále budu zkoumat chování sítě v mezních situacích. První zkoumanou situací bude stav, kdy je nedostatek dat – toto je znázorněno na obrázcích 8.12 a 8.13. Jak lze vidět, tak i po velmi dlouhé době učení zůstává síť ve stavu, kdy jsou řídící parametry neurčité. Toto se nezmění ani po delší době, neboť s malým množstvím dat si síť není jistá, jak by měl výstup vypadat. Oproti tomu NN pouze minimalizuje střední kvadratickou odchylku a dosáhla chyby pouze okolo $3 \cdot 10^{-10}$, což znamená, že data byla aproximována velmi přesně. Nedostatek dat se projeví rozptýlenější funkcí hustoty řídících proměnných.



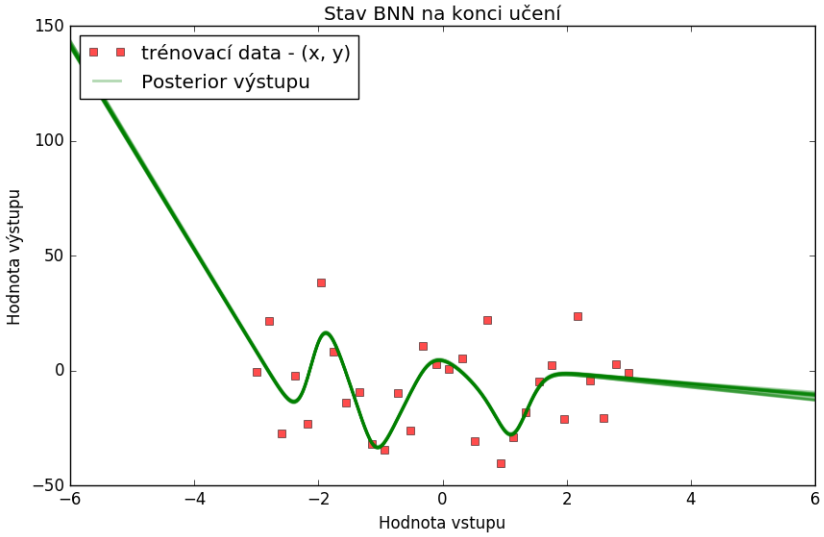
Obrázek 8.12: stav po 8000 průchodech, síť 3-5, tanh



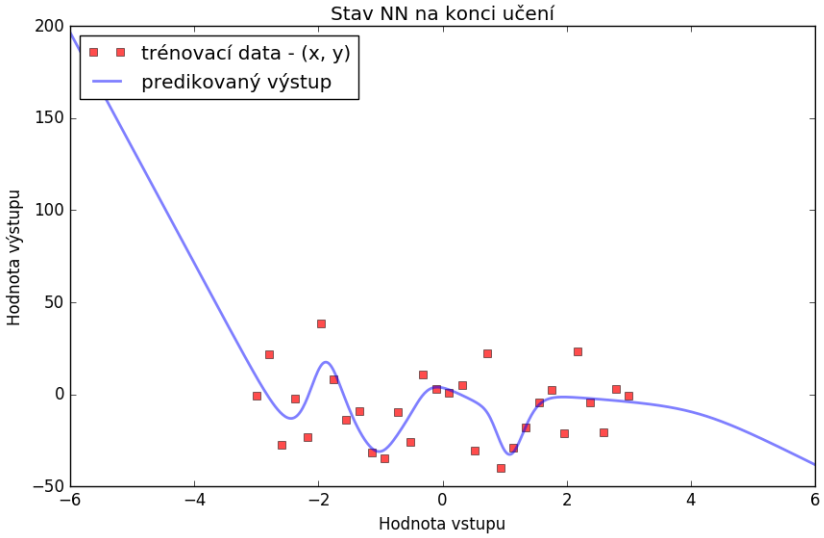
Obrázek 8.13: stav po 8000 průchodech, síť 3-5, SoftPlus

Další situace, kterou je zajímavé prozkoumat, je funkce založená na zcela náhodných datech – šumu. Jak bylo řečeno v rovnici 5.30, učení spoléhá na minimalizaci entropie. Šum má maximální entropii, neboť vše z generovaného intervalu je stejně pravděpodobné. Pokud jsou ovšem samotná data šumem, jak se bude BNN chovat? Jak lze vidět na obrázcích 8.14, 8.15, 8.16 a 8.17, tak BNN se neurčitě chová velmi málo, a to jen na okrajích. Bez využití entropie BNN degradovala na NN – neboť pro stejné modely vznikly obdobné výsledky.

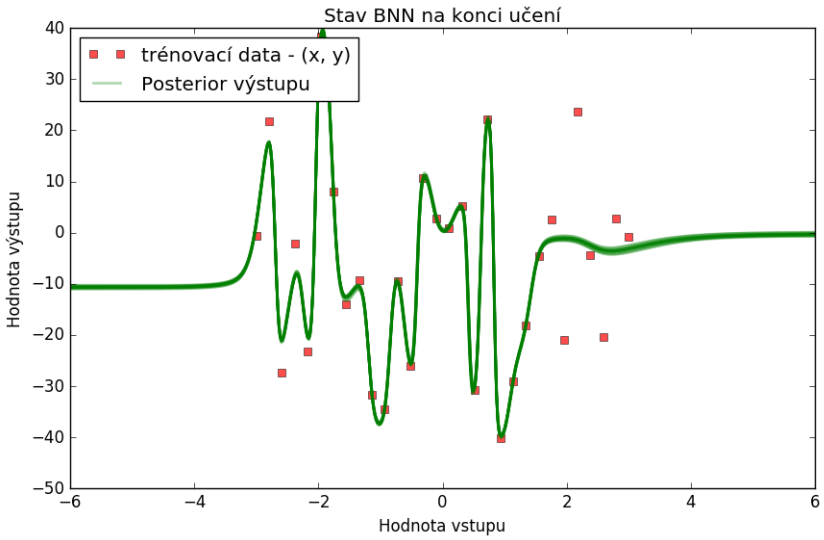
Testováno bylo na dvouvrstvém modelu se čtyřiceti neurony ve skryté vrstvě. Prezentovány jsou dvě různé aktivační funkce.



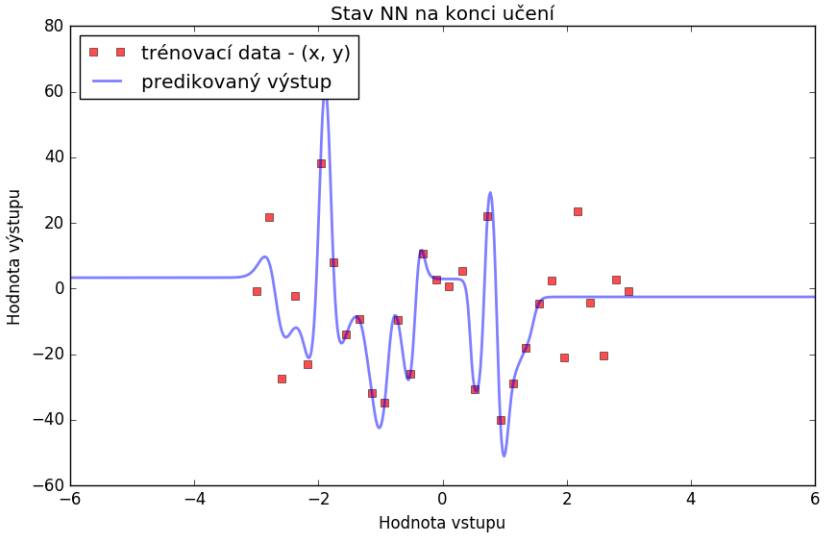
Obrázek 8.14: stav po 25000 průchodech, síť 2-40, SoftPlus



Obrázek 8.15: stav po 25000 průchodech, síť 2-40, Softplus

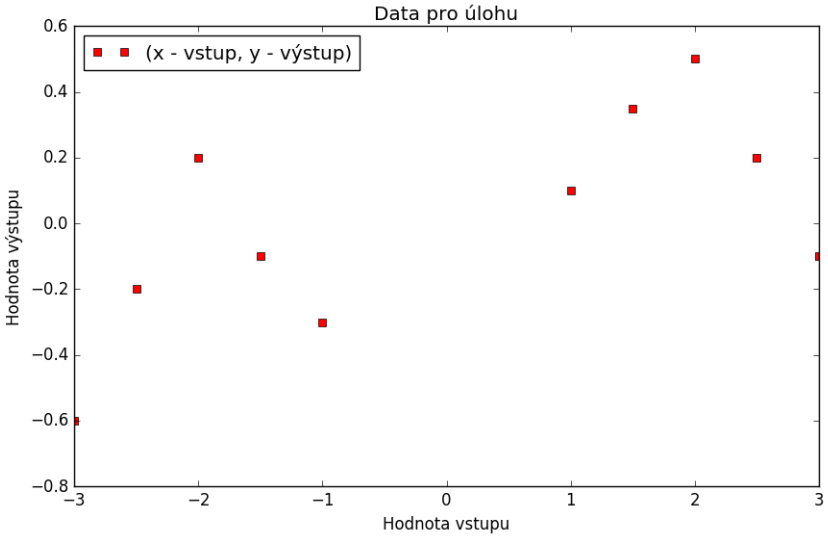


Obrázek 8.16: stav po 25000 průchodech, síť 2-40, tanh

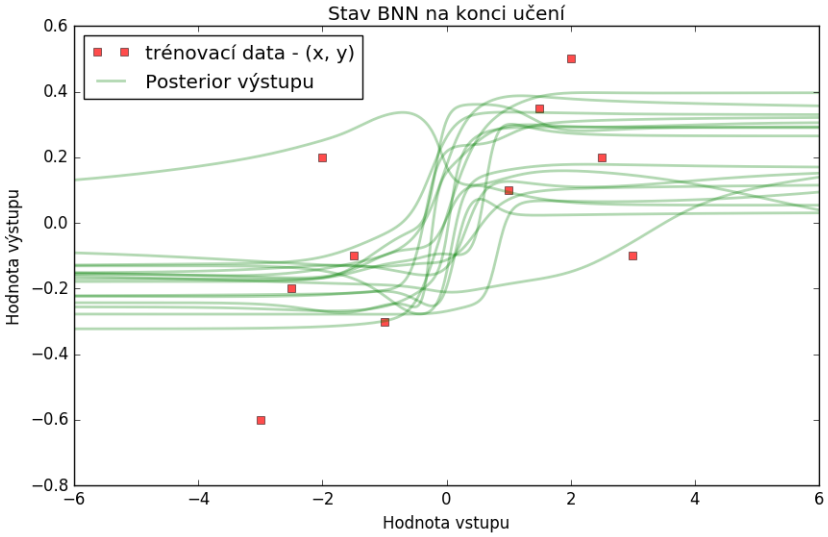


Obrázek 8.17: stav po 25000 průchodech, síť 2-40, tanh

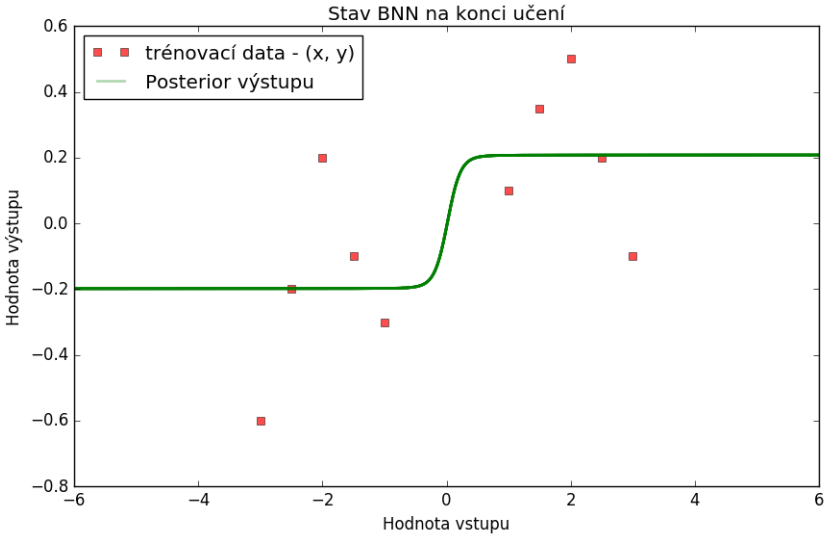
Dále se zaměřím na problematiku přeučení. Mějme málo dat (viz. obr. 8.18). NN při dostatečně dlouhém učení opět minimalizuje střední kvadratickou odchylku, ovšem výsledný deterministický model nemusí vůbec odpovídat skutečnosti (obr. 8.21). Oproti tomu BNN problém příliš generalizuje (obr. 8.19). Pokud si vykreslím očekávání tohoto modelu (obr. 8.20), tak pozoruji, že si síť myslí, že se jedná o dva shluky a blíže je v dohledné době neanalyzuje.



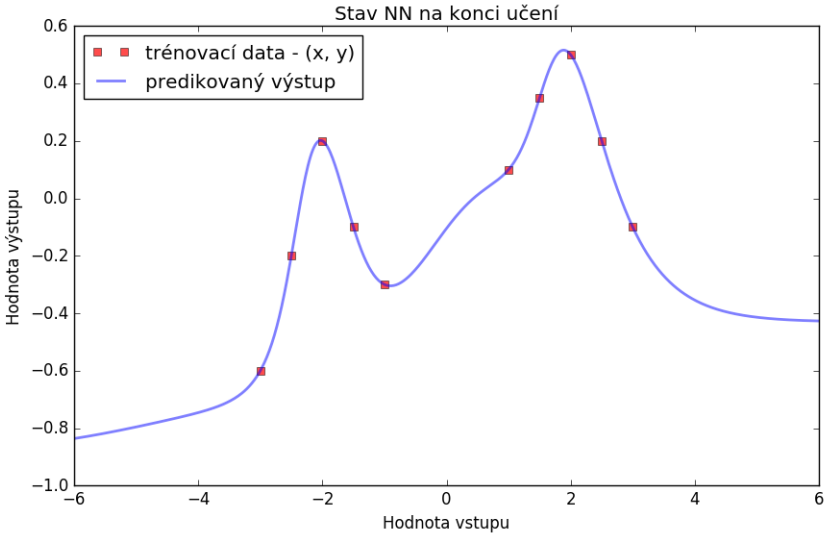
Obrázek 8.18: Vygenerovaná Data – okolo nuly chybí



Obrázek 8.19: stav po 5000 průchodech, síť 3-7, tanh

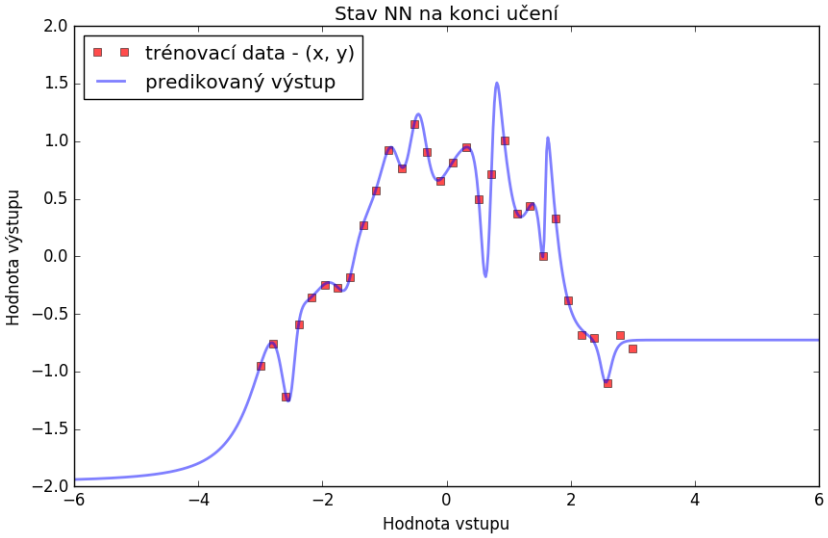


Obrázek 8.20: stav po 5000 průchodech, síť 3-7, tanh, očekávání

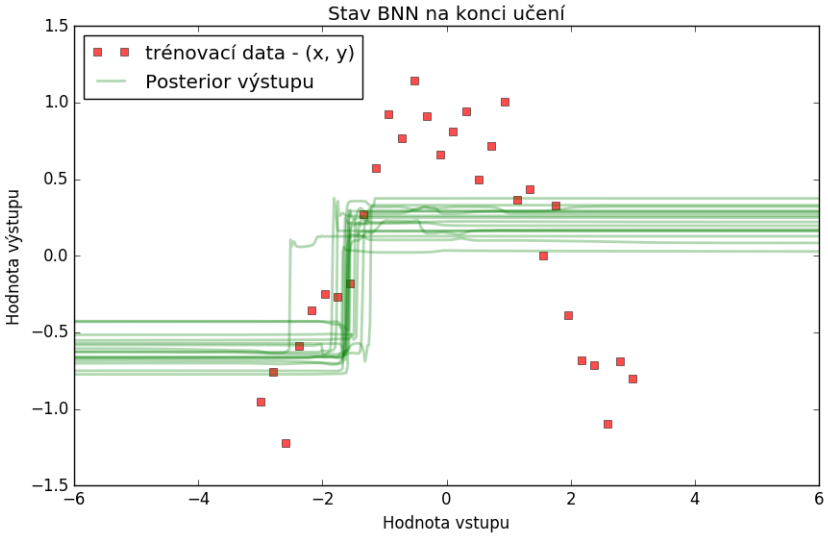


Obrázek 8.21: stav po 5000 průchodech, síť 3-7, tanh

Při zkoumání přeučení kvůli příliš složitému modelu (velké množství vrstev s mnoha neurony), lze zmíněné chování rovněž pozorovat (obr. 8.22 a 8.23). Z toho plyne, že BNN sice netrpí problémem s přeučením, ale problémem s nadměrnou generalizací.



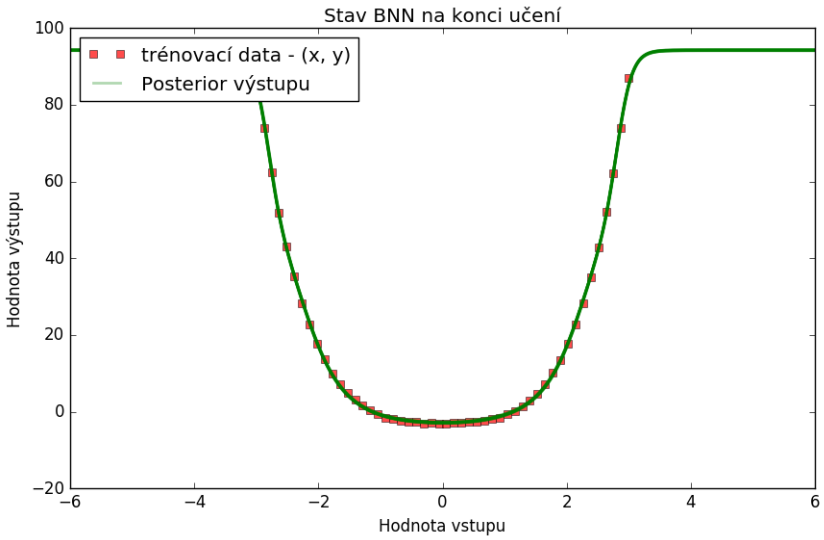
Obrázek 8.22: stav po 20000 průchodech, síť 8-10, tanh



Obrázek 8.23: stav po 20000 průchodech, síť 8-10, tanh

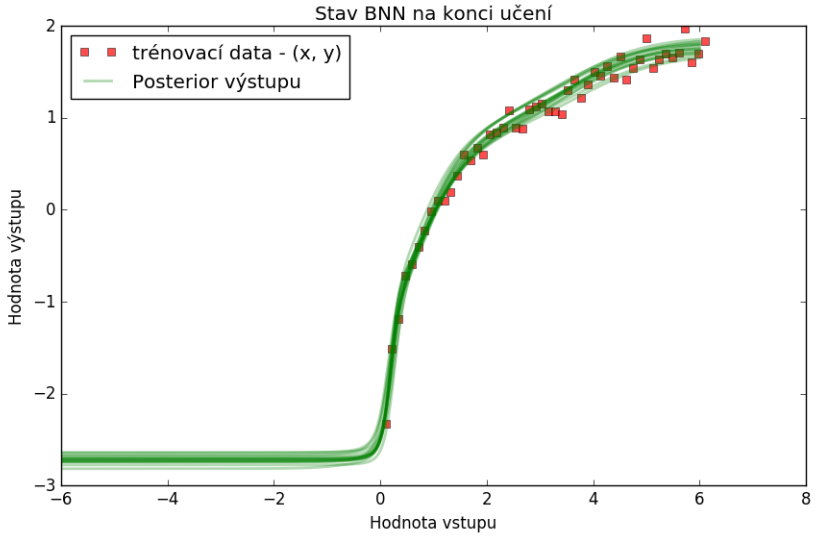
8.3 Další příklady

V této části předvedu několik dalších experimentů s BNN. Obrázek 8.24 ukazuje, že je-li zkoumaná funkce dostatečně jednoduchá a dat dostatek, pak BNN velmi přesně tuto funkci aproximuje. Neurčitost je v tomto případu minimální, neboť si je síť velmi jista, o jakou funkci se jedná.



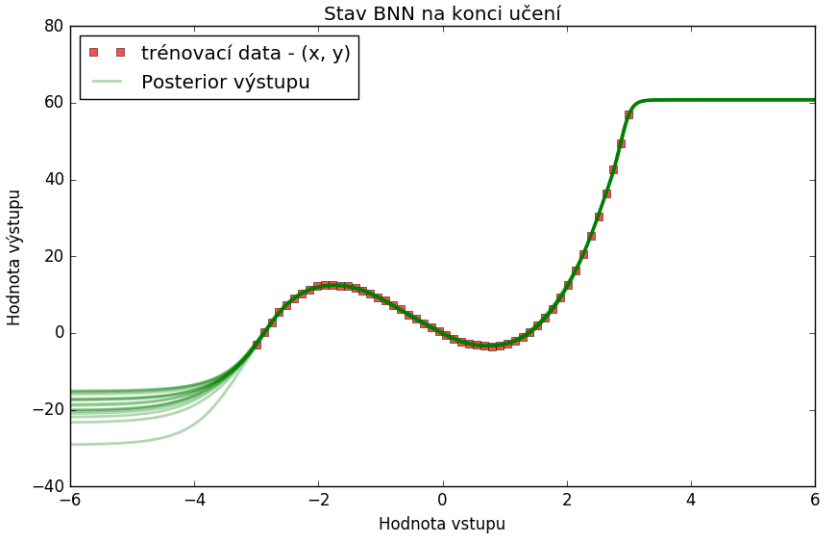
Obrázek 8.24: stav po 5000 průchodech, síť 3-5, tanh

Další obrázek 8.25 ukazuje zašuměnou logaritmickou funkci. Síť i po učení stále vykazuje nedeterminismus v oblasti s nakupenými daty – hustota vhodně pokrývá data, což může být žádoucí chování.

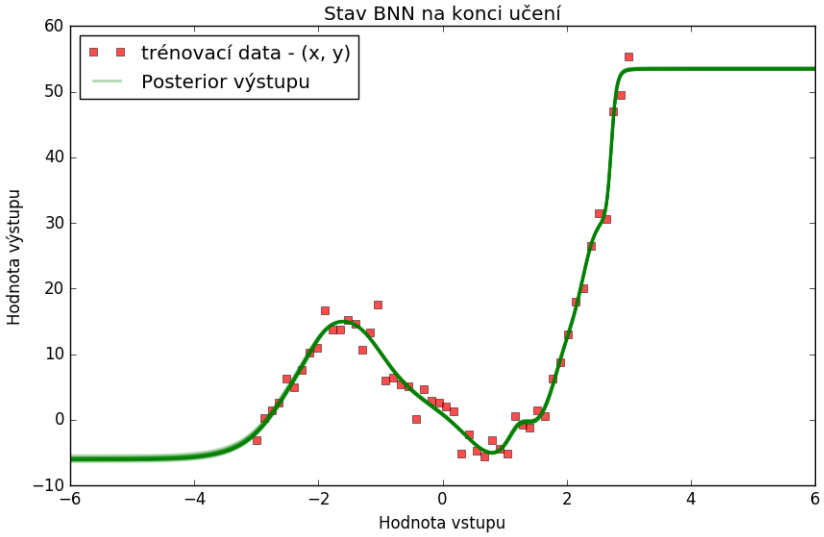


Obrázek 8.25: stav po 5000 průchodech, síť 3-5, tanh

Obrázky 8.26 a 8.27 pracují s předdefinovaným polynomem s minimálním šumem a s velkým šumem v datech. S malým šumem lze pozorovat neurčitost v levé části grafu, kde síť nemá data, a proto je zde větší neurčitost. Přidáním šumu zvětšíme entropii a zhoršíme učení.



Obrázek 8.26: stav po 5000 průchodech, síť 3-5, tanh



Obrázek 8.27: stav po 5000 průchodech, síť 3-5, tanh, přidán značný šum

Kapitola 9

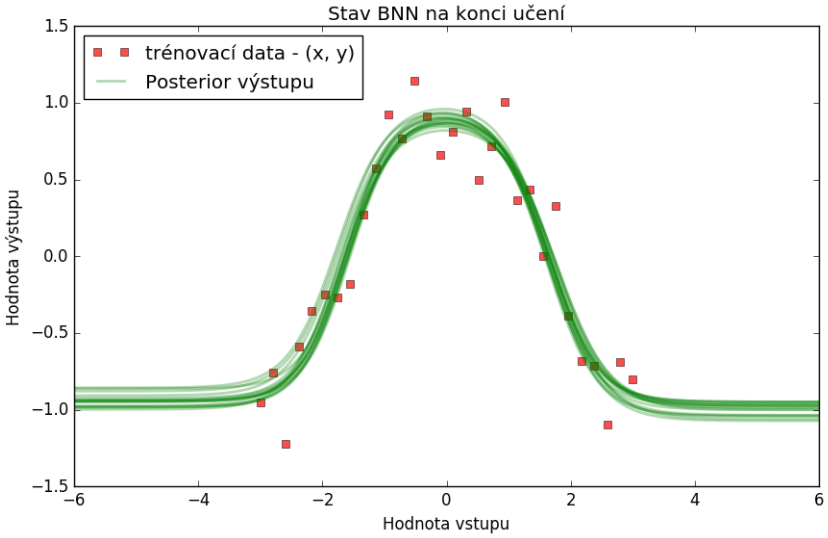
Zhodnocení BNN

V této části práce je shrnutí Bayesovské neuronové sítě. Tato síť v principu funguje obdobně jako umělá neuronová síť. Rozdíl je v tom, že řídící parametry jsou náhodné proměnné a učení probíhá inferencí. V případě této práce byl použit algoritmus postavený na Kullback-Leiblerovy divergenci. Výsledek, reprezentovaný jako náhodná proměnná, může být deterministicky interpretován svým očekáváním.

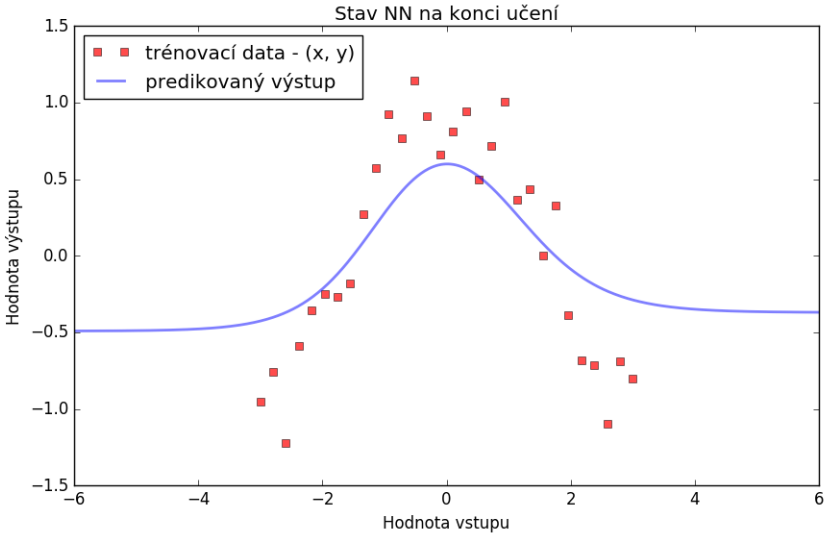
Z experimentů vyplývá, že BNN je fungující alternativou k NN. Učení ovšem probíhá z opačných počátečních stavů. NN začíná z ničeho (pokud jsou počáteční parametry nastaveny na nulu, popř. hodnoty blízko nule) a postupně zlepšuje svůj výstup tak, aby odpovídal trénovacím datům. BNN začíná ze stavu, kdy síť reprezentuje všechny funkce, které je schopna popsat. Postupně pak omezuje tuto množinu tak, dokud nezbudou jen ty, které *dostatečně dobře* reprezentují trénovací data.

U BNN není třeba rozdělovat data na trénovací a validační, neboť nehrozí přeučení na tato data – naopak upřednostníme pravděpodobné modely a pomůžeme procesu učení. BNN mají ovšem problém opačný k přeučení. Místo přeučení může dojít k nepřiměřenému zobecnění, kdy výsledek degraduje na konstantní (popř. po částech konstantní) funkci. Toto nastává zejména u příliš složitých sítí. V takovém případě je vhodné provést *pruning* sítě (její zjednodušení odstraněním vrstvy či neuronu).

Z hlediska rychlosti lze říci, že BNN byla v průměru dvakrát pomalejší než NN při stejném počtu průchodů algoritmem. Nicméně algoritmy jsou natolik odlišné, že toto není nujak průkazné o efektivitě. Na obrázcích 9.1 a 9.2 lze vidět, že BNN konverguje k řešení značně rychleji než NN. Pokud jde o metriku střední kvadratické odchylky (*MSE*), tak NN vždy vítězí. To je očekávané, neboť NN minimalizuje právě *MSE*. BNN inference pracuje s maximalizací pravděpodobnosti modelu sítě a minimalizací entropie reprezentujícího rozdělení (funkci hustoty) vůči datům. Z toho rovněž vyplývá, že síť má problémy s reprezentací náhodného šumu. V tomto případě se BNN chová obdobně jako NN. Za zmínku stojí, že BNN v implementovaném programu potřebuje inicializaci značně rozsáhlého modulu, což znamená, že učení BNN bude vždy trvat o konstantu déle než u NN. V mém případě se jednalo o tři a půl sekundy. Je na uživateli zhodnotit, který přístup ve své úloze potřebuje. Vhodné je zkusit úlohu vyřešit co nejvíce způsoby a výsledky vůči sobě porovnat. V ideálním případě se různé modely shodnou na výsledku.



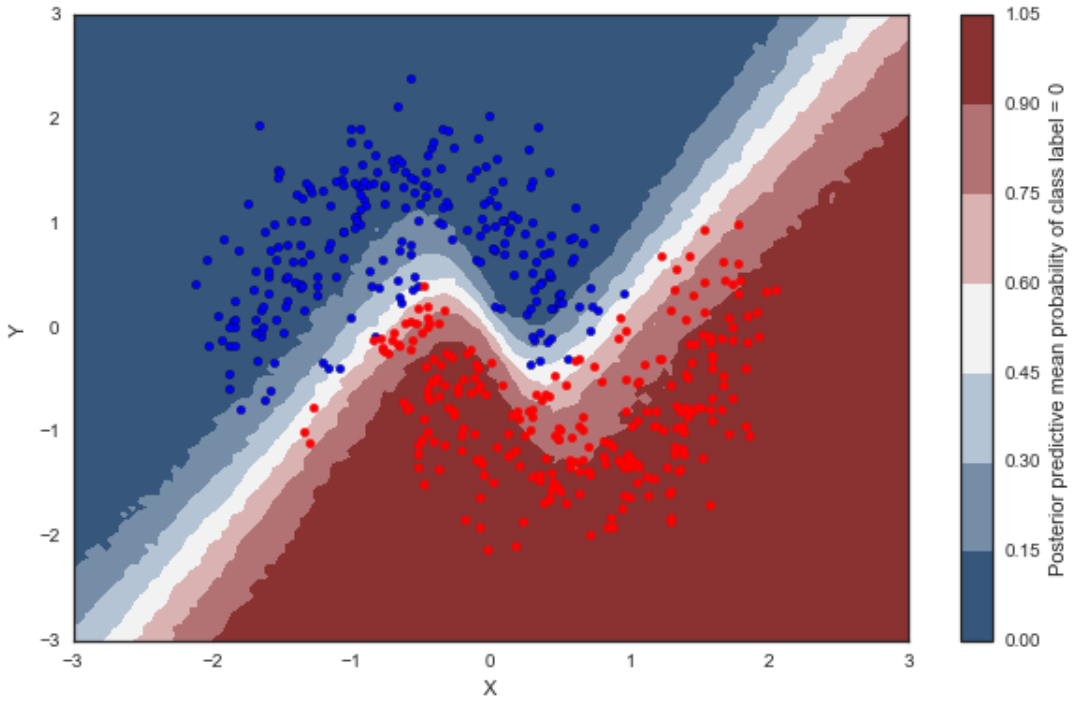
Obrázek 9.1: stav po 500 průchodech, síť 2-2, tanh



Obrázek 9.2: stav po 500 průchodech, síť 2-2, tanh

BNN samozřejmě umí řešit i jiné problémy, než je demonstrovaná regrese. Obrázek 9.3 převzatý z [18] demonstruje BNN na binární klasifikační úloze – tj. zařazování multidimenzionálních vstupní data do dvou tříd. Vstup je v příkladu dvoudimenzionální, neboť tento problém lze vhodně zakreslit do roviny. Červené body reprezentují trénovací vzorky z první třídy. Modré body reprezentují vzorky z druhé třídy. Pozadí udává, které oblasti dat budou klasifikovány do které třídy. U neurčitých výsledků lze respektovat neurčitost a vybrat třídu náhodně, ovšem s respektem k rozložení pravděpodobnosti (pokud je na daném místě vzo-

rek z první třídy na 60%, pak ho rovněž s 60% pravděpodobností do první třídy zařadím). Jiným přístupem je klasifikovat vzorek vždy do nejpravděpodobnější třídy.



Obrázek 9.3: ukázka klasifikace - převzato z [18]

Kapitola 10

Závěr

Práce představila Bayesovskou neuronovou síť a ukázala její výhody i nevýhody. Výhodou sítě je zejména její větší autonomie, kdy se až do konvergence navzájem korigují řídící parametry sítě (váhy) s parametry reprezentující očekávané rozložení dat.

Jelikož je učení založeno na Bayesovské inferenci vycházející z Bayesova vzorce, získaný model respektuje velmi důležitý princip – *Occamovu břitvu*. Ta za nejpravděpodobnější hypotézu (popř. teorii, model) prohlašuje tu nejjednodušší.

Jednodušší modely (hlavně ty s relativně nízkými absolutními hodnotami vah) pak lépe zobecňují a jsou méně náchylné k přeučení. Efektivita takové sítě byla prezentována na praktickém příkladu.

Následně byl vytvořen demonstrační program, který dokáže provádět regresi pomocí neuronových sítí – samozřejmě i pomocí Bayesovské. Experimentálně bylo předvedeno, jak síť funguje.

Bayesovská neuronová síť se dá použít nejen jako alternativa k deterministické variantě, ale také jako její doplněk. Požadovaná úloha se vyřeší oběma způsoby a výsledné modely se porovnají. Pokud se výrazně liší (NN je přeučená a BNN je příliš zobecněná), pak z toho vyplývá, že byla zvolena nevhodná topologie sítě.

Do budoucna je možné zaměřit se na automatizované hledání vhodné topologie (místo přístupu hrubou silou). Bylo by zajímavé zkoumat, jestli lze topologii vhodně zakódovat do řídících parametrů a dynamicky měnit topologii sítě během učení.

Dalším možným tématem je vyhledat a porovnat všechny dostupné algoritmy učení, neboť zde jsou prezentovány pouze dva. *Nealův* algoritmus, neboť byl použit v ukázce práce z Vietnamu, a algoritmus využívající Kullback-Leiblerovu divergenci, neboť pomocí tohoto algoritmu jsem vytvořit demonstrační program.

Jiná možnosti navázání na práci je specializace na chování hlubokých neuronových sítí s využitím neurčitosti. V tomto případě doporučuji začít prozkoumáním modulu *PyMC3* pro Python, neboť v tom se tato problematika od nedávna vyvíjí. Alternativně lze zkoumat jiné třídy problémů, než je regrese. Kupříkladu by bylo možné prozkoumat možnosti BNN v oblasti učení bez učitele (shlukování).

Rovněž by se šlo zaměřit na problematiku dolování informací z dat a prozkoumat, jak si Bayesovské neuronové sítě stojí vůči zavedeným populárním algoritmům (např. *SVM*).

Shrnu tím, že tato práce je pouze úvodem do problematiky Bayesových neuronových sítí a možnosti dalšího průzkumu problematiky jsou dalekosáhlé.

Literatura

[1] Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Continuum Analytics. web, Listopad 2016.
URL https://continuum.io

[2] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. online, 2015, software available from tensorflow.org.
URL http://tensorflow.org/

[3] Bui, D. T.; Pradhan, B.; Lofman, O.; aj.: *Landslide susceptibility assessment in the Hoa Binh province of Vietnam: A comparison of the Levenberg–Marquardt and Bayesian regularized neural networks.* Geomorphology, , č. 171-172, 2012: s. 12–29.

[4] *Chollet, F.: keras. online, 2015.*
URL https://github.com/fchollet/keras

[5] *Forster, M.: Notice: No Free Lunches for Anyone, Bayesians Included. online, 2009.*
URL http://philosophy.wisc.edu/forster/papers/Krakow.pdf

[6] *Holst, A.: The Use of a Bayesian Neural Network Model for Classification Tasks. Royal Institute of Technology, S-100 44 Stockholm, Sweden, 1997, ISBN 91-7153-645-0.*

[7] *Hunter, J. D.: Matplotlib: A 2D graphics environment.* Computing In Science & Engineering, *ročník 9, č. 3, 2007: s. 90–95, doi:10.1109/MCSE.2007.55.*

[8] *Lampinen, J.; Vehtari, A.: Bayesian Approach for Neural Networks – Review and Case Studies.* Neural Networks, , č. 14, 2001: s. 7–24.

[9] *LeCun, Y.; Bengio, Y.; Hinton, G.: Deep learning.* Nature, , č. 521, 2015: s. 436–444, *doi:10.1038/nature14539.*

[10] *MacKay, D. J. C.: Bayesian Methods for Neural Networks: Theory and Applications. online, 1995.*
URL http://www.inference.eng.cam.ac.uk/mackay/cpi4.pdf

[11] *MacKay, D. J. C.: Modelling Phase Transformations in Steel: Bayesian Non-Linear Modelling with Neural Networks. online, 1995.*
URL http://www.inference.eng.cam.ac.uk/mackay/cpi_short.pdf

[12] *MacKay, D. J. C.: Probable Networks and Plausible Predictions - A Review of Practical Bayesian Methods for Supervised Neural Networks. online, 1995.*
URL http://www.inference.eng.cam.ac.uk/mackay/network.pdf

[13] Neal, R. M.: Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method. Technická Zpráva CRG-TR-92-1, 1992.

[14] Neal, R. M.: Bayesian Learning for Neural Networks. Dizertační práce, University of Toronto, Department of Computer Science, 1995.

[15] Tran, D.; Hoffman, M. D.; Saurous, R. A.; aj.: Deep probabilistic programming. In International Conference on Learning Representations*, 2017.*

[16] Tran, D.; Kucukelbir, A.; Dieng, A. B.; aj.: Edward: A library for probabilistic modeling, inference, and criticism. arXiv preprint arXiv:1610.09787*, 2016.*

[17] van der Walt, S.; Colbert, S. C.; Varoquaux, G.: The NumPy Array: A Structure for Efficient Numerical Computation. Computing In Science & Engineering*, , č. 13, 2011: s. 22–30, doi:10.1109/MCSE.2011.37.*

[18] Wiecki, T.: Bayesian Deep Learning. online, 2016.
URL http://twiecki.github.io/blog/2016/06/01/bayesian-deep-learning/