**Univerzita Hradec Králové**

**Fakulta informatiky a managementu**

BAKALÁŘSKÁ PRÁCE

**2017 Adam Ouhrabka**

**Univerzita Hradec Králové**

**Fakulta informatiky a managementu**

**Katedra Informatiky**

Strojové učení v počítačové grafice

Autor: Adam, Ouhrabka

Aplikovaná informatika

Vedoucí práce: Ing. Bruno Ježek, PhD.

Hradec Králové XXX 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne

Anotace

Annotation

[Anotace 4](#_Toc3122591)

[Annotation 5](#_Toc3122592)

[1 Úvod 8](#_Toc3122593)

[2 Strojové učení v obecných rysech 10](#_Toc3122594)

[2.1 Rozlišení diskriminativních a generativních modelů 11](#_Toc3122595)

[2.2 Metody strojového učení s omezením na neuronové sítě 12](#_Toc3122596)

[2.2.1 Hluboké dopředné neuronové sítě 13](#_Toc3122597)

[2.2.2 Učení neuronových sítí 15](#_Toc3122598)

[2.2.3 Konvoluční neuronové sítě 16](#_Toc3122599)

[2.2.4 Rekurentní neuronové sítě 19](#_Toc3122600)

[2.2.5 General adversarial networks 19](#_Toc3122601)

[3 Procedurální generování 20](#_Toc3122602)

[3.1 Klasifikace PCG algoritmů 21](#_Toc3122603)

[3.2 Tradiční metody procedurálního generování 21](#_Toc3122604)

[3.3 Motivace, účel a vize používání PCG 22](#_Toc3122605)

[4 Procedurální generování prostřednictvím strojového učení 24](#_Toc3122606)

[4.1 Současné experimenty PCGML 25](#_Toc3122607)

[5 Technologie pro implementaci algoritmů strojového učení 27](#_Toc3122608)

[5.1 Python 27](#_Toc3122609)

[5.2 NumPy 27](#_Toc3122610)

[5.3 TensorFlow 28](#_Toc3122611)

[5.4 Keras 29](#_Toc3122612)

[6 Demonstrace metod strojového učení na příkladech generování 30](#_Toc3122613)

[6.1 Příklad hanojských věží 30](#_Toc3122614)

[6.2 Užité technologie 30](#_Toc3122615)

[6.3 Definice cíle a východisek 30](#_Toc3122616)

[6.4 Popis užitého algoritmu 32](#_Toc3122617)

[6.5 Testování hypotetických výstupů algoritmu 32](#_Toc3122618)

[6.5.1 Charakteristika vstupních dat 32](#_Toc3122619)

[6.5.2 Souhrn sledovaných vlastností 34](#_Toc3122620)

[6.5.3 Hypotéza blízké relace 34](#_Toc3122621)

[6.5.4 Charakteristika natrénovaných dat 34](#_Toc3122622)

[6.6 Návrhy pro zlepšení algoritmu 34](#_Toc3122623)

[7 Závěr 35](#_Toc3122624)

# Úvod

Strojové učení a počítačová grafika jsou oblasti, které v posledních dvou dekádách zaznamenávají v rámci technických oborů rapidní vývoj. Co se týká strojového učení lze identifikovat tři příčiny tohoto progresu. Prvně je to invence sofistikovaných modelů a architektur. Za druhé produkce a existence obřích datových sad pro účely trénování a testování. A konečně existence softwarové a hardwarové podpory, která umožňuje relativně jednoduše implementovat komplexní modely a logiku (Abadi et al. nedatováno). Mezi obory strojového učení a počítačové grafiky existuje podstatný průnik. Jedná se například o techniky počítačového vidění a zpracování obrazu. Jádrem teoretické části textu je představení tohoto průniku a specifické podoblasti, jíž lze označit jako „Generování grafického obsahu za pomocí metod strojového učení“ - v literatuře označovanou zkráceně jako PCGML.

Motivací k sestavení tohoto textu byl zájem o počítačovou kreativitu v oblasti grafiky a snaha o poznání metod strojového učení, primárně neuronových sítí.

Úkoly a cíle bakalářské práce jsou definovány následovně. Za prvé bude v obecných rysech představena oblast strojového učení s explikací metod, které jsou využívány v praktické části. Výčet aplikací založených na těchto metodách (zejména na neuronových sítích) je pak striktně omezen na oblast počítačové grafiky.

Druhým cílem je vytvořit přehled postupů a metod, které jsou používány či diskutovány v souvislosti s procedurálním generováním grafického obsahu. Nejprve je tato oblast představena obecně a poté je text zaměřen na techniky procedurálního generování za pomocí strojového učení.

Třetím cílem je prozkoumat dostupné technologie a prostředí, jež jsou vhodné pro implementaci metod strojového učení a jsou využívány v praktické části bakalářské práce.

V rámci praktické části je pak představen návrh a implementace metody, která za využití jednoduchých architektur neuronových sítí řeší procedurální generování distribuci objektů v dvojrozměrné scéně na základě naučených dat. Tato metoda je otestována a vyplynulé výsledky jsou zhodnoceny v závěru, což je posledním zadaným cílem této práce.

# Strojové učení v obecných rysech

Z formálních definic strojového učení je vybráno tradiční znění Toma Mitchella, který definuje strojové učení následujícím způsobem. *„Stroj (počítačový program) se učí plnit třídu zadání T ze zkušenost E a s úspěšností P v tom případě, když se úspěšnost plnění zadaných úkolů T zlepšuje s využitím znalostí nabytých ze* zkušenosti E.“ (Mitchell 1997a). K základním praktickým úlohám, na něž jsou algoritmy strojového učení aplikovány patří klasifikace, predikce, expertní systémy, popřípadě obecná kontrola agenta konkrétního autonomního systému.

Obecně je cílem algoritmů strojového učení aproximace neznámé komplexní funkce (Goodfellow et al. 2016). Omezíme-li se na oblast učení s učitelem, dosahuje se této aproximace s menší nebo větší úspěšností za pomocí předkládání vstupních a výstupních vektorů hledané funkce. Celý aparát tohoto optimizačního procesu je často parametrizovatelný takzvanými hyperparametry, přičemž celková úspěšnost nalezení příslušné funkce je na těchto parametrech závislá (Jordan a Mitchell 2015).

Jednoduché funkce lze v zásadě aproximovat i jinými čistě programově-algoritmickými metodami, to je nicméně často neefektivní popřípadě u komplexnějších funkcí vyloženě nereálné. Následuje výčet hlavních argumentů pro upřednostňování algoritmů strojového učení, namísto implementace jiného procedurálně či obecně programově laděného systému.

Mitchellem definovaná zkušenost *E* je v oblasti strojového učení reprezentována daty. Pro trénovací sadu je důležité to, jaké formy nabývá a v jakém pořadí vstupují jednotlivá data do algoritmu. Především je pak podstatné„*…jak dobře data reprezentují distribuci příkladů, přes které musí být výsledný systém evaluován úspěšností P.“* (Mitchell 1997b). V takto definované trénovací sadě, lze pak za pomocí vhodných algoritmů strojového učení hledat obecné vztahy a korelace, které by tradiční analýze zůstaly skryté, to znamená, že by se jen těžko hledala přiměřená algoritmizovatelná logika. Tak zní první argument. Vzhledem k záměru této práce lze dodat, že tyto skryté vztahy mohou být později užity ke generování nových vzorků při zachování obecné charakteristiky a distribuce trénovací sady.

Mezi další důvody pro preferenci algoritmů strojového učení patří dle Nilssona enormní velikost datové sady, která znemožňuje syntézu do tradičního programu. Dalším argumentem je dynamická proměnlivost dat, na kterou musí být systém schopný v ideálním případě reagovat. A konečně jsou některé úlohy definovatelné pouze pomocí předkládání četných příkladů (Nilsson 1998). Většinu z těchto jedinečných náležitostí lze demonstrovat na současném projektu Alterego, který mapuje neuromuskulární signály kolem úst na omezený slovník pojmů s úspěšností 92% (Kapur et al. 2018). Lze si jen stěží představit, že by tato funkce byla nalezena a prováděna programovou cestou.

## Rozlišení diskriminativních a generativních modelů

Klasicky se algoritmy strojového učení dají odlišit na „učení s učitelem“ a „učení bez učitele“. Nicméně vzhledem ke skutečnosti, že se v průběhu práce pracuje výlučně se  supervised algoritmy, bude užitečnější poukázat na jinou klasifikaci totiž na rozdíl diskriminativních a generativních modelů. Na tento rozdíl mezi nimi lze dobře poukázat v aplikaci na klasifikaci, kde jde obvykle o rozdělení datové distribuce do několika oblastí. Zatímco u diskriminativních modelů jde o namapování vstupů na konkrétní výstup, kterým může být konkrétní třída či skalární hodnota, generativní přístup modeluje pravděpodobnostní relace mezi proměnnými daného modelu (Jebara 1996). Pro N proměnných lze v těchto modelech nalézt úplnou sdruženou pravděpodobnost ve formě p(x1, ... ,xn). Existuje-li vyjádření takové distribuce, lze na základě této formule odvozovat další hodnoty proměnných pomocí bayesovských pravidel. Tomuto procesu se říká inference. Ve vztahu ke klasifikaci je to pak především pravděpodobnost p(y|x), kde y je třída objektu vzhledem k rozdělení x (Ng a Jordan nedatováno). Generativní modely existují většinou ve formě grafických modelů, mezi které patří primárně bayesovské sítě se směrovými relacemi mezi proměnnými a Markovovy modely, popřípadě Markovovy nahodilostní pole (Jebara – google books). Na příkladu klasifikace rastrových obrázků jednociferných čísel demonstruje Revow zásadní praktický důsledek při použití generativních modelů, totiž ten, že najdeme-li generativní model pro daný systém, našli jsme i model, který je schopen generovat nové vzorky (Michael Revow et al. 1996).

Diskriminativní modely se v jádru svého fungovaní snaží aproximovat ideální hranici, která žádoucím způsobem odděluje vícedimenzionální datovou distribuci. Příklad takového oddělení je znázorňen na obrázku 1. Mezi diskriminativní algoritmy patří typicky neuronové sítě, support vector machines, lineární regresní algoritmy a další (Jebara 1996). Primární užití diskriminativních algoritmů lze vidět v klasifikaci popřípadě regresi.



Obrázek demonstrace typického vymezení hranice při klasifikování dat

## Metody strojového učení s omezením na neuronové sítě

V následujících kapitolách budou postupně představeny konkrétní metody strojového učení s omezením na formy neuronových sítí. Nejprve bude popsána standartní architekura umělé neuronové sítě a poté budou představeny tři další modely totiž General Adversarial Networks, Konvoluční neuronové sítě a Rekurentní neuronové sítě. Všechny tyto modely stojí na bázi standartních neuronových sítí a jsou předmětem aktuálního výzkumu a aplikací v posledních letech. Rovněž budou představeny jejich aplikace v oboru počítačové grafiky.

### Hluboké dopředné neuronové sítě

Hluboké neuronové sítě jsou jednou z často používaných metod strojového učení. Je to aparát, který má charakter funkce a lze jej popsat jako *masivně paralelní procesor*. Mnohačetným předkládáním *m* vzorů do neuronové sítě je možné adaptovat funkční parametry jednotlivých neuronů [] tak, že vznikne obecně použitý model pro klasifikaci či predikci dat na základě nového vektoru vstupních hodnot. Tento proces adapatace je chápán jako učení neuronové sítě. Implementace tohoto učení stojí na minimalizaci chybové funkce [], která agreguje rozdíl výstupních hodnot vyprodukovaných neuronovou sítí s datově předepsanými hodnotami pro konkrétní vzor [] (Vladimír Olej a Hájek Petr 2010). Přirozeně se během vývoje oboru strojového učení začalo pracovat i s jinými chybovými funkcemi.

*Rovnice 1 chybová funkce množiny předkládaných vzorů (neuronová síť s dvěmi a více výstupními neurony)*

Přestože existují různé varianty a topologie neuronových sítí, základní principy korelují (Tučková 2003).

Stavebním elementem neuronových sítí je neuron, jež si lze představit jako funkci na jejímž vstupu je vektor hodnot, které vyšly z předcházející vrstvy neuronové sítě (NS) a na jejím výstupu skalární hodnota, jež figuruje jako vstup pro další vrstvy popřípadě jako výstup neuronové sítě. Vstupní vektor je nejprve agregován do skalární hodnoty, která je následně vstupem do aktivační funkce . Ta transformuje vstupní hodnotu do výstupní hodnoty celého neuronu. Parametr přidává prahovou hodnotu neuronu (bias), která přispívá k aktivizaci neuronu.

*Rovnice 2 agregace vstupnich hodnot neuronu*

Konkrétních aktivačních funkcí existuje celá řada a mají zásadní charakter na fungování NS. Aktivační funkce mohou mít za důsledek lineární i nelineární charakter klasifikačních oblastí vyprodukovaných neuronovou sítí. Příkladem nelineární aktivační funkce jako je například sigmoidální funkce (Vladimír Olej a Hájek Petr 2010).

*Rovnice 3 výstupní hodnota neuronu při užití funkce sigmoid*

Aktuálním trendem v oblasti hlubokého učení je však v rámci vnitřních vrstev neuronových sítí jiná funkce, která se označuje jako „rectified linear unit“ zkráceně ReLU, kde (Goodfellow et al. 2016). Funkce neuronu v kontextu strojového učení napodobuje neuron bilogický. Dle Glorota je to však teprve užití této aktivační funkce, která toto napodobení ve velké míře dovršuje. Především diskutuje princip „řídkosti aktivace“, který je identifikován jako pozitivní jev, při kterém v rámci šíření signálu zůstane mnoho neuronových jednotek na svém výstupu nulových. To je i důsledek užití ReLU (Glorot et al. 2011). Aplikace této aktivační funkce má rovněž za důsledek rychlejší trénování (Bhandare et al. 2016).



(goodfellow DL s190)

Podstatnou komponentou neuronové sítě je aktivační funkce poslední vrstvy. Zde záleží jaký charakter výstupu je žádoucí. Obecně se však užívá již prezentovaná funkce sigmoid a její odvozenina funkce softmax . Výstup neuronové sítě klasifikuje vstup do konkrétní proměnné. Je-li tato proměnná diskrétní, to znamená, že představuje například příslušnost vstupních dat do *n* tříd, měla by výstupní vrstva vyjadřovat vektor pravděpodobnosti příslušnosti objektu do těchto tříd. Pro tento konkrétní účel se využívá právě funkce softmax.

Rovnice formální vyjádření funkce softmax

Důsledkem užití této funkce je, že na výstupu neuronové sítě bude vektor *v1*, který bude vyjadřovat pravděpodobnosti pro jednotlivé třídy určené pořadím v rámci vektoru a díky „normalizační“ složce v čitateli bude složkový součet *v1* roven 1.

### Učení neuronových sítí

Podobně jako existuje variabilita chybových funkcí, existují i různé optimalizační (učící) algoritmy. Obecně však jsou však na začátku učení iniciovány všechny váhy v systému jako náhodné nenulové hodnoty. K následné adaptaci vah ve standardních modelech hlubokého učení s učitelem se pak používá nějaká alternativa optimalizačního algoritmu například Stochastic gradient descent ve spojení s algoritmem zpětného šíření chyby - Backpropagation algorithm. Proces spočívá v minimalizaci chybové funkce ve směru gradientu tj. směrem k lokálnímu minimu. To se děje v každé iteraci za pomocí úpravy vah napojených na jednotlivé neurony v jednotlivých vrstvách sítě. Výsledným vztahem pro korekci všech váh v NS je rovnice, kde je keocifient posunu v rámci gradientu, je směr gradientu a je původní hodota příslušné váhy.

*Rovnice 5 korekce vah v BPA*

Ve chvíli, kdy je v rámci jedné iterace průchodu dat neuronovou sítí spočítána hodnota chybové funkce výstupní vrstvy, je nutné zpětně pro všechny neurony skrytých vrstev vypočítat nakolik váhy, které z nich vedou přispívají ke konkrétní hodnotě celkové chyby *E* výstupní vrstvy.

*Rovnice 6 výpočet chyby neuronu*

Při této operaci

Adam - algoritmus

### Konvoluční neuronové sítě

Mezi nejúspěšnější architektury aplikované i v počítačové grafice posledních několika dekád patří mimo GAN také hluboké konvoluční neuronové sítě (CNN), které se v rámci soutěže ILSVRC[[1]](#footnote-1) umísťují na předních příčkách. (Goodfellow et al. 2016).

CNN jsou propojením dvou silných konceptů tj. dopředné neuronové sítě a principu konvoluce. Přestože aplikace z poslední doby užívají CNN i například pro rozpoznání řeči či práci s jazykem. Používají se jak pro obrazová, tak pro časová data. Formálně jsou definovány jako *„...sítě, které namísto maticového násobení užívají alespoň v jedné vrstvě konvoluci.“*. (Goodfellow et al. 2016)

Máme-li dvoudimeznionální pole dat aplikujeme konvoluční operací () následovně:

Rovnice operace konvoluce pro pole I (**dim**= a jádro K (**dim**=

Z formálního vyjádření vyplývá, že se jedná o vzájemné pronásobení matice vzoru *I* a matice jádra *K*, přičemž výsledek tohoto pronásobení je agregován do skalární hodnoty a uložen do výsledné matice *S*. Základní motivací pro provedení této operace je identifikace specifických znaků na vstupních datech. Rovněž je díky využití konvoluce do určité míry zajištěno, že varianty dat jedné třídy budou do neuronové sítě vstupovat jako invariantní co do velikosti, občasných posunů a jiných distorzí (LeCun et al. 1998). Důvodem této skutečnosti je fakt, že se ve fázi učení naleznou taková jádra (tj. jejich váhy), která ve spojení s příslušnými vstupy vyselektují typické znaky, podle kterých bude možné jednoznačně určit třídu. Kromě těchto jader figurují ve výsledné klasifikaci ještě váhy klasických vrstev, které jsou napojeny na výstup konvoluční části modelu.



Obrázek konvoluční architekture LaNet-5

V obrázku výše je zobrazena architektura konvoluční sítě, která byla svými autory nazvána jako LeNet-5 (LeCun et al. 1998). Kromě klasických neuronových vrstev, vystupují v těchto architekturách ještě další jednotky v podobě konvolučních a „agregačních“ (subsampling) vrstev.

První z nich aplikuje operaci konvoluce na předchozí vrstvu modelu. Efektivitia těchto architektur spočívá mimo jiné v tom, že se konvuluje hned několikrát. To znamená, že jednotlivá jádra vyústí v procesu učení do navzájem různých konfigurací, což má za důsledek „selekci“ odlišných znaků. Počet konvolučních jader figuruje jako hyperparametr. S řetězovitou aplikací konvoluce se tak uvnitř modelu navyšuje počet jaderných konfigurací a jejich konvolvovaných produktů (feature maps). Zároveň se snižuje velikost matice těchto produktů, což je přirozený důsledek konvoluce kde se výsledná velikost (W = velikost vstupu, K = velikost jádra, P = odsazení, S = velikost kroku).

V  posloupnosti vrstev CNN následuje obvykle za konvoluční vrstvou transformace, která na její výstup aplikuje aktivační funkci, čímž je dosaženo nelinearity výstupu. Třetí v tomto typickém CNN tripletu je takzvaná subsampling či pooling vrstva. Funkce této vrstvy se může lišit. Obecně jde ale o další redukci velikosti konvolučního výstupu na základě okolí. Hodnota výstupu tak může být například maximální numerická hodnota prvku unvitř okolí, průměrná hodnota prvků či další funkce. Účelem této operace je, aby se konvolvovaná reprezentace dat stala ještě více invariantní (Goodfellow et al. 2016). V některých případech je za tímto tripletem přítomna ještě tzv. dropout vrstva, která při šíření dopředného „signálu“ nahodile deaktivuje některé neurony. Dle Krizhevského zamezuje tato technika jevu takzvaného přeučení a obecně přispívá k přesnosti modelu (Krizhevsky et al. 2012).

Konvoluční sítě mají své aplikace především v počítačové grafice, konkrétně počítačovém vidění. Prezentovaný model LaNet-5 byl svými autory už v roce 1998 použit ke strojovému čtení psaného textu. Chyba se v rámci testovací sady pohybovala v řádech desetin procent (LeCun et al. 1998).

Současné aplikace shrnuje ve svém článku Bhandare. Většina aplikací se týká dvourozměrných rastrových obrázků o třech barevných kanálech. CNN byly úspěšně použity pro rozpoznání tváře a klasifikaci objektů obecně. Z implementovaných řešení lze poukazát na klasifikátor ImageNET, jehož dataset obsahuje patnáct milionů obrázků patřících do 22 000 kategorií. Přesnost tohoto modelu byla 62.5%. (Krizhevsky et al. 2012).

Dalším odvětvím aplikace je takzvaný „scene labeling“, kde jde o identifikaci a zaměření objektů ve scéně. Průlomem v této oblasti byla v rámci hlubokého učení architektura CNN navržená Farabetem. Fully convolutional neural networks.

Vývoj CNN nadále spočívá v zacházení s hyperparametry a ladění architektur modelu. Na základě Bhandarova souhrného článku lze konstatovat, že existuje mnoho variant CNN, plnící cíle klasifikace, segmentace, ale i další. (Bhandare et al. 2016).

### Rekurentní neuronové sítě

When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks, presented in chapter (d183)

### General adversarial networks

Jsou specifické architektury neuronových sítí, které patří do rodiny generativních modelů a byly vynalezeny Ianem Goodfellowem v roce 2014. Sám Goodfellow obecně definuje generativní modely jako *„...modely, které se na základě tréninkové distribuce dat tvořených množinou pdata naučí odhad takové distribuce reprezentovat“*. V konkrétních aplikacích mohou takové modely buď plnit pouze onu reprezentativní funkci a nebo mohou na základě vnitřní struktury modelu generovat nové originální vzorky (Goodfellow et al. 2014). Základním principem fungováním je takzvaný princip „maximální pravděpodobnosti“ to znamená „*...nalezení takových parametrů pro model, aby maximalizovali pravděpodobnost trénikových dat*“.

Goodfellow navrhl model skládající se ze dvou komponent. První z nich je diskriminátor, jehož funkcí je binární klasifikace generátorem nagenerovaných vzorků na 1 nebo 0. Druhou komponentou je generátor, jehož úkolem je vytvořit vzorky, které napodobují tréniková data. Generátor v každé iteraci učení ladí své váhy tak, aby transformoval prvotně iniciovaný šum na přijatelnější vzorky, které jsou diskriminátorem znovu klasifikovány a vzniklá chyba se projeví v příští iteraci generování. Obě komponenty tak stojí proti sobě a finální naučení redistribuce vzniká ze vzniklé tenze mezi nimi (Goodfellow 2016).

Většina aplikovaných GAN architektur v sobě zakomponovává konvoluční mechaniku. Hovří se pak o tzv. DCGAN (Goodfellow 2016).

Tyto modely byly kromě generování nových vzorků použity například na obarvení obrázků v infračerveném spektru (Suarez et al. 2017). A obecně  transformaci obrázků do obrázku určité stylizace například transformace fotografie denní scény na scénu noční (Isola et al. 2016).

DOOM generative

# Procedurální generování

V rámci počítačové grafiky existují v podstatě dva způsoby pro definici grafického obsahu modelované scény. Ať už uvažujeme dvojrozměrné, popřípadě trojrozměrné zobrazení scény, tak vlastnosti zobrazovaných objektů této scény musí být vždy předem zaneseny v příslušných datových strukturách, které pak figurují v rámci vizualizačního řetězce. Zachycení geometrie a umístění objektu v souřadnicích scény může být definováno buď staticky tj. „ručním“ modelováním objektů anebo algoritmicky, kdy se vzhled a transformace objektů odvíjejí od sekvence příkazů v rámci daného algoritmu. Podobné rozdělení se týká například i textur a dalších elementů, které scénu dotvářejí. Druhý ze zmíněných přístupu lze označit termínem procedurální generování obsahu (PCG).

Zkratka PCG bývá skloňována především v oblasti počítačových her, kde najdou tyto algoritmy široké uplatnění. Tyto algoritmy jsou však použitelné i v rámci budování parametrizovatelných virtuálních prostředí, simulací, vizualizací, popřípadě v oblasti designu. Nutno zmínit, že PCG se ve své definici neomezuje jen na generování grafického obsahu jako takového, nýbrž se tento pojem používá i pro algoritmické generování jakéhokoliv obsahu jako je například obsah textový, animovaný, popřípadě hudební. Yannakis zahrnuje do oblasti PCG i generování personalizovaného obsahu, které se uplatňuje například v rozhraních pro e-commerce a uživatelských rozhraních obecně (Yannakakis a Togelius 2011).

Univerzální vlastností většiny algoritmů pro PCG je, že ve svém těle pracují s faktorem nahodilosti, který je nezbytný pro zajištění diverzity obsahu. Základní metodou pro naplnění tohoto kritéria je například implementace pseudonáhodných čísel v podobě šumu (HENDRIKX a MEIJER nedatováno). Mezi další podstatné vlastnosti PCG algoritmů patří rychlost, spolehlivost, kontrolovatelnost a uvěřitelnost (Julian et al. 2016). Smith ve svém článků užívá v kontextu PCG výstižný pojem „řízená nahodilost“ (Smith nedatováno). Nahodilost má v PCG efekt spíše na dílčí částí negenerovaného obsahu, ale nikoliv už na celek, který musí být uvěřitelný, popřípadě hratelný. Demonstrováno na příkladu: distribuce jednotlivých listů může být náhodná do té míry, do kdy se objekt jeví stále jako objekt dané třídy, v našem případě jako strom. Termínem „řízená“ je vystihnuto kritérium pro smysluplnost, ze kterého vyplývá další podstatná vlastnost PCG algoritmů, kterou je přítomnost určitých omezení (pravidel, constraints), které v jádře algoritmu zajišťují, že obsah bude uvěřitelný.

## Klasifikace PCG algoritmů

Algoritmy PCG se dělí na parametrizovatelné (assisted) a na ty které ke svému fungování žádnou parametrizaci nepotřebují (non-assisted). I minimální parametrizace jako například definice počtu iterací spadá do skupiny parametrizovatelných PCG algoritmů, které nabízejí designérovy kontrolu nad finální podobou generovaného obsahu. (Carli et al. 2011).

Další dělení definující PCG algoritmy se týká toho, zdali má generování zpětnou vazbu na uživatelské interakce nebo takovou vazbu nemá. Popřípadě zdali se obsah generuje při běhu aplikace (online) nebo slouží pouze jako podpůrný nástroj designu při vývoji (offline) (Togelius et al. 2010).

Jedná-li se o algoritmus, který přímo reaguje na interakci s uživatelem a v rámci běhu aplikace dotváří „personalizovaný obsah“, označuje ho Yannakakis zkratkou EDPCG (Experience-driven procedural content generation) (Yannakakis a Togelius 2011).

## Tradiční metody procedurálního generování

Tradiční PCG algoritmy jsou založeny na poznatcích z mnoha oblastí informatiky. Pro generování grafického obsahu se užívají evoluční algoritmy, Lindenmayerovi systémy, fraktály, a další specifickým způsobem aplikované postupy, které pro každý svůj běh negenerují jiný, přesto smysluplný a monotematický obsah. K tomu lze podotknout, že *„Všechny tyto metody mají společné to, že tyto algoritmy, parametry, omezení a cíle, které se podílí na vytváření obsahu jsou v podstatě ručně definované svými tvůrci“* (Summerville et al. 2017)

Stěžejní proměnou PCG algoritmů je podoba reprezentace obecných genotypů, na základě kterých vzniká vždy originální entita (fenotyp). Takové genotypy mohou reprezentovat jak obecné geometrie objektů, tak například obecné vztahy mezi nimi. Tato biologická paralela vychází z často aplikované metody PCG, kterou jsou genetické algoritmy.

## Motivace, účel a vize používání PCG

Klíčovou součástí moderních vizualizací, popřípadě počítačových her jsou namodelované objekty plnící buď kosmetickou či interaktivní funkcí. Nevyužije-li se v rámci produkce PCG metod je všechen obsah modelován ručně. Taková praxe výrazně zvyšuje cenu produktu a prodlužuje dobu jeho vývoje. Na druhé straně je možné potenciál PCG použít pouze tam, kde prvek prvek náhody dotváří funkci herního mechanismu nebo vytváří diverzitu, která nenarušuje účel obsahu (nebrání například v postupu). Generovaný obsah tedy může být na základě tohoto kritéria rozdělen na funkční (necessary) a kosmetický (optional content)(Togelius et al. 2010).

Vize a cíle současného bádaní PCG v grafické oblasti definuje ve svém článku Julian Togelius. Jako první vize je nadneseně představena univerzální metoda na vytvoření komplexních virtuálních světů určitého žánru. I v případě, že by se jednalo pouze o statický grafický obsah takového světa dá se společně s Togeliusem konstatovat, že podobný algoritmus je nerealizovatelný i v rámci vzdálené budoucnosti. Komplexita takového úkolu je enormní. Problémů, které stojí v cestě takovému zadání je hned několik a tvoří primární úkoly v rámci výzkumu PCG.

Generovaný obsah se potýká s originalitou, nedostatečnou diverzitou a smysluplností. To vše jsou znaky kreativity v rámci ručního modelování. Klíčová otázka zní, zdali se v případě PCG dá kreativita algoritmizovat a v případě generování prostřednictvím strojového učení stochasticky modelovat i v rámci komplexních celků. Kritéria výpočetní kreativity, která významě korelují s požadavky na PCG definoval ve své práci Pereira. Jako hlavní znaky vidí: uchování modelu znalostí o dané třídě, schopnost redistribuce znalosti do nových spojení, schopnost kriticky zhodnotit produkt (validace), uvážení kontextu do kterého je tvorba zasazena a užití konvergentních i divergentních postupů při kreaci (Pereira 2008). Všechny tyto podmínky, které Pereira postuluje pro svůj model (Creative general problem solver) jsou realizovatelné a byly realizovány prostřednictvím generativních modelů strojového učení například za použití Generative adversarial neural networks.

Togelius ve svém článku nadále vybízí k syntéze metod, které byly vynalezeny v rámci generování jednotlivých objektů jako například terénu, budov či interiéru, do komplexního parametrizovatelného generátoru. Interakci mezi jednotlivými generátory podtrhuje ve svém článku i Hendrikx (HENDRIKX a MEIJER nedatováno). Výstupy jednotlivých generátorů by tedy měly být vzájemně kompatibilní a ve vztahu, který dodává generovanému celku přidanou hodnotu.

Tradiční metody PCG jsou často aplikovány ad hoc ke konkrétnímu použití. Znovupoužitelnost je realizovatelná pouze principiálně a jen stěží se dají stejné algoritmy použít napříč různými aplikacemi. I to je nedostatek na který Togelius poukazuje (Togelius et al. 2013).

Uvážíme-li Togeliusův nárok na komplexní procedurální generátor, je nutné do něj zahrnout i dynamickou stránku scénu tj. animační. Procedurální animování je další oblast, v rámci které probíhá aktivní výzkum (Togelius et al. 2013)

Jako obecný trend v rámci PCG lze tedy primárně identifikovat redukci genericity generovaného obsahu a směřování k univerzálním navzájem propojeným generátorům obsahů komplexních. K naplnění těchto cílů se výzkumníci v oblasti PCG často uchylují k metodám strojového učení, jež má v návaznosti na současné cíle PCG velký potenciál, neboť umožňuje modelování složitých funkcí a datových distribucí tj. uchování znalosti o modelu a jeho redistribuci do nových celků.

# Procedurální generování prostřednictvím strojového učení

Tradiční PCG přistupuje k tvorbě obsahu s konkrétní referencí vzhledem k cíli. Tato reference však není součástí samotného algoritmu, ale figuruje pouze jako inspirace toho, kdo algoritmus navrhuje. Znamená to, že se v rámci algoritmu hledají pravidla a omezení, která pomohou aproximovat žádoucí obsah. Na rozdíl od tohoto přístupu procedurální generování prostřednictvím strojového učení schované pod zkratkou vezme existující strukturu a na základě zpracování a „pochopení“ daného obsahu vytvoří model, pomocí kterého nageneruje další diverzifikovaný obsah. Tak definuje PCGML Summerville (Summerville et al. 2017).

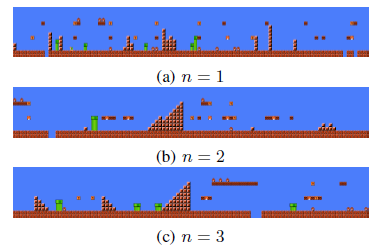
Definice strojového učení od Toma Mitchella uvedená v první kapitole ke je aplikovatelná rovněž pro PCGML, kdy zadáním *T* je generování konkrétní třídy obsahu na základě zkušenosti *E*, která je tvořena množinou ručně modelovaných obsahů (v případě učení s učitelem). Měření úspěšnost *P* může být v této konkrétní aplikaci strojového učení značně komplikované a může v konečné fázi spočívat v subjektivním posouzení omezeného množství nagenerovaného obsahu. Příkladem implementace takové metriky jsou dotazníky vyplněné testery, na jejichž základě byly evaluovány hratelnosti procedurálně nagenerovaných levelů a tedy celého algoritmu. (Roberts a Chen 2015). Pro posouzení většího vzorku dat je třeba implementace algoritmů, jejichž funkcí je evaluace generovaného obsahu na základě příslušných kritérií.

Ne všechny metody strojového učení, které se jinak výborně hodí pro klasifikaci či predikci, jsou vhodné pro generování nového obsahu. Odpovídá tomu i zmíněné rozdělení metod strojového učení na diskriminativní a generativní modely. Z generativních modelů strojového učení vyzdvihuje Summerville jako vhodné zejména, n-gramy, markovy modely, současné generativní modely hlubokého učení (GAN) a specifické architektury rekurentních neuronových sítí (Summerville et al. 2017).

## Současné experimenty PCGML

Následující kapitola prezentuje vybrané užité aplikace a metody v oblasti PCGML. Některé tyto příklady posloužily jako motivace k sepsání této bakalářské práce a vytvoření metody užité v její praktické části. V kontextu strojového učení se všechny tyto aplikace musely po svém způsobu vypořádat s prooblémem malé datové sady a požadavkem na použitelnost nagenerovaných vzorků.

Silným nástrojem pro generování jednoduchého obsahu se ukázala být nejjednoduší varianta markovova řetězce ve spojení s n-gramy. Markovovův řetězec je graf, jehož vrcholy reprezentují uričtý stavový prostor. Mezi vrcholy existují ohodnocené hrany, a ty vyjadřují pravděpodobnost přechodu mezi jednotlivými stavy (Gagnius 2017). N-gram je struktura, která vyjadřuje jaké [*y*] bude následovat v případě, že pole [*x*] délky [*n*] obsahuje nějaké konkrétní hodnoty. Technika tohoto typu se dá dobře využít pro generování jednoduchých levelů viz. obrázek 3. V tomto experimentu byla vstupní data nejprve indexována a rozřezána na sloupce. Poté byl na základě n-gramů vytvořen pravděpodobnostní model, který iterativně predikoval další sloupec podmíněně k [n] posledním sloupcům. (Summerville et al. 2017).



Obrázek

Výsledné nagenerované levely pro jednotlivá n, vstupní data byla rozdělena na sloupce a za pomocí n-gramů transformována do Markovova řetězce

Komplexnější alternativou pro dosažení obdobného cíle je užití neuronových sítí, a to konkrétně rekuretních architektur. V tomto přístupu se už generování neodehrává na úrovni sloupců nýbrž na úrovni jednotlivých políček 13 kategorií. Kromě těchto políček byla do algoritmu zahrnuta i informace o pohybu hráče uvnitř vstupních dat (levelů). Účelem tohoto kroku byla reálná hratelnost negenerovaných úrovní. Generování levelů probíhalo tedy iterativně v prostoru [nx, ny] od pomyslného [y0 , x0] k [yn , x0] zase zpátky dolů od [yn , x1] k [y0 , x1]. To vše s paměťovou stopou na předtím vygenerovaná pole o délce zhruba 200 polí (Summerville a Mateas 2016).

Jako třetí příklad, který lze označit za výsledek v PCGML, je zde uveden Fisherův pokus o syntézu žánrových scén do modelu schopného tyto scény generovat v četných variacích. Konkrétně se jedná o scénu stolu obklopeném objekty. K učení algoritmu byly použito 130 manuálně vymodelovaných prostředí interiéru, ve kterých se dohromady znovupoužívalo 1723 kategorizovaných objektů. Variace objektů bylo dosaženo implementací takzvaných kontextuálních kategorií. Variování tak neprobíhá pouze na základě příslušnosti objektu k určité kategorii, ale je založené na pozičním kontextu objektu. Druhým modelem pro finální generování je model okurence, který je založen na bayesovksých sítích, a který v podstatě řeší pravděpodobnost možné přítomnosti objektu vzhledem k již přítomným. V rámci modelu okurence byly rovněž implementovány umělá omezení, za účelem zavedení informace o relaci „x je na y“ to znamená „y podpírá x“. Poslední složkou je pozicovací model, který řeší finální umístění a natočení objektu za tímto účelem jsou použity „Gaussian mixture models“ (Fisher et al. 2012).



Obrázek

Autoencodery - opravy --- (Jain et al. nedatováno)

V souvislosti s PCGML lze rovněž zmínit experiment Giacomella a jeho týmu, který s využítím GAN vytvořil model pro generování herních levelů. V tomto případě však byla tréniková data o velikosti 1088 vzorků zakódována jako rastrové obrázky (128x128) půdorysů levelů ze hry DOOM. V rámci každého levelu existuje celkem šest obrázků, z nichž každý repreznetuje jinou informaci tj. například o topologii, výškovém profilu či rozmístění předmětů uvnitř levelu. Autoři vytvořili kromě standartního modelu i podmíněný model, který za pomoci sedmi parametrů parametrizoval nagenerované vzorky.

Podobným způsobem byly GAN architektury implementovány za účelem procedurálního generování terénu (Beckham a Pal 2017).

# Technologie pro implementaci algoritmů strojového učení

Většina používaných knihoven pro účely implementace algoritmů strojového učení je dnes navázána na programovací jazyk Python. Jmenovitě to jsou například knihovny PyTorch, Scikit-learn, Tensorflow a zde užíváný Keras. Vzhledem k jednostrannému zaměření této práce na neuronové sítě budou postupně ve stručnosti představeny ty technologie, se kterými se pracuje v praktické části a které jsou obecně základem pro aplikační práci s neuronovými sítěmi.

## Python

Python je opensource intepretovaný, multiplatformní jazyk s čistou syntaxí evokující pseudokód. Obecně je dnes tento jazyk používán především ve vědecké a technologické komunitě. Důvody pro jeho užívání vystihuje Oliphant: V první řadě je to obrovské množství jak nativních, tak dodatečných knihoven, které řeší komplexní úlohy vyplynulých z vědeckých potřeb. Jmenovitě se jedná například o vizualizační nástroje, knihovny pro statistiku, strojové učení a mnohé další. Za druhé lze jazyk užívat jak prodcedurálním, tak i objektovým způsobem. A konečně Python umožňuje živou interakci s běžícím kódem a má velice silnou odbornou komunitu uživatelů (Oliphant 2007).

## NumPy

NumPy je dle Oliphanta podstatným důvodem akademické oblíbenosti Pythonu (Oliphant 2007). Je to knihovna, která umožňuje za pomocí vyššího jazyku, kterým Python je, efektivně implementovat numerické operace (především na maticích a tensorech obecně). Za touto efektivitou stojí tři složky. Za prvé vektorizace veškerých výpočtů, za pomocí operací implementovaných v jazyce C. Za druhé efektivní správa proměných v paměti a za třetí minimalizace počtu operací (Van Der Walt et al. 2011). V souvislosti se strojovým učením se NumPy využívá při předzpracování dat a je rovněž interní součástí většiny knihoven pro strojové učení.

Základní stukturou (objektem) této knihovny je NumPy pole (ndarray). Primární charakteristikou tohoto pole je, že každý jeho prvek může být nadále vícedimenzionální. Van der Walt konstatuje, že: *„...NumPy pole je pouze příhodný způsob popisu jednoho nebo více bloků paměti za účelem jednoduché manipulace s reprezentovanými čísli.“*. Každá ndarray struktura si drží ukazatel na první byte v poli, datový typ prvků, svůj tvar (shape), počet kroků pro skok na další element (strides) a příznaky zda-li je možné s polem manipulovat.

Důležitým parametrem je především onen počet kroků. Van der Walt tak hovoří o „krokovém paměťovém modelu“, který umožňuje intepretaci paměti na více způsobů bez nutnosti kopírovat data. Toho se využívá především při manipulaci s tvarem (shape) multidimenzionálního pole, kdy dojde pouze k přepočítání potřebných kroků. (Van Der Walt et al. 2011).

## TensorFlow

Tensorflow je knihovna pro podporu strojového učení vyvíjena společností Google. Primárně funguje v Pythonovském kontextu, ale existují její alternativy, popřípadě API pro jiné jazyky. Umožňuje poměrně komplikovanou distribuci výpočtů na externí výpočetní jednotky, popřípadě jejich paralelizaci či naopak synchronizaci. Obecně je tato knihovna uzpůsobena pro experimentální a dobře kontrolovatelný vývoj nejen algoritmů strojového učení. Myšlenka této knihovny je založena na toku operací uspořádaných v grafu. Tyto operace jsou výlučně prováděny na tensorech tj. vstupem i výstupem je tensor (v Pythonu se jedná o ndarray). Intepretováno jinak: Vrcholy tohoto grafu reprezentují matematické operace, které vlastní či mění stav výpočtu. Po hranách pak proudí jednotlivé tensory Kromě jiného obsahuje tensorflow nástroje pro diferenciaci libovolné chybové funkce. (Abadi et al. nedatováno).

Omezíme-li se na modely hlubokého učení, umožňuje Tensorflow v podstatě dva přístupy k implementaci. Odpovídá tomu i rozdělení oficiální dokumentace na „High level APIs“ a „Low Level APIs“. První „High level“ funkcionalita se překrývá se specifikací knihovny Keras a bude pojednána v následující kapitole. Je to právě ono „Low Level API“, které za pomocí nativních struktur umožňuje značnou kontrolu nad budouvanými modely. V praktické části této práce se však s tímto API nepracuje, proto nebude nadále rozepisováno.

Za poslední zmínku z knihovny TensorFlow stojí nástroj TensorBoard, který slouží k vizualizaci jednotlivých fází učení modelů a mnohým dalším vizualizačním a optimalizačním potřebám vývojáře. Umí rovněž vizualizovat celý nadefinovaný komputační graf.

## Keras

Keras je již zmiňované „High level API“ primárně určené pro jednoduché budouvání modelů hlubokého učení. S jeho pomocí lze nadefinovat a parametrizovat většinu základních architektur (CNN, RNN, LSTM, ANN, GAN) a jejich variant pro praktické a aplikační potřeby.

Uživatelská přívětivost je podstatnou zásadou této knihovny, proto je možné nadefinovat funkční modely velice rychle. Keras je nadstavou nad TensorFlow, popřípadě jinými alternativami knihoven strojového učení jako je například Theano. Pro trénování implementovaných modelů lze s výhodou využít grafického hardwaru, kde mohou výpočty na tensorech probíhat paralelně a celý proces je několikanásobně rychlejší.

Základní definovatelnou strukturou je v Kerasu Model. Ten slouží jako placeholder pro různorodé druhy vrstev. Jednotlivé vrstvy lze do modelu přidávat buď sekvenčně jednu za druhou nebo je v případě komplikovanějších modelů možné definovat souslednost vrstev do grafu. Veškeré vrstvy jsou plně parametrizovatelné a jejich funkce se odvíjí od jejich typu. Keras podporuje standartní plně propojené vrstvy, konvoluční triplety, rekurentní vrstvy a mnohé další. Keras rovněž obsahuje nativní nástroje pro datový preprocessing, škálu callbacků volatelných při trénování, nástroje pro načítání standartních datových sad, vizualizační nástroje a další utility. Pochopitelně jsou přítomné parametrizovatelné funkce pro trénování a evaluaci nadefinovaných modelů.

# Demonstrace metod strojového učení na příkladech generování

Konkrétně specifikovaný cíl této kapitoly a bakalářské práce vůbec je demonstrovat a evaluovat navrženou metodu pro generování scény v diskrétním dvojrozměrném prostoru.

## Příklad hanojských věží

Jako postranní úloha při zpracování bakalářské práce vznikla implementace, která svým principem předjímá postup hlavní aplikace. Ve stručnosti je zde tedy prezentována.

## Užité technologie

Veškeré implementace, které jsou zde demonstrovány byly implementovány v jazyce Python ve verzi 3.6. Argumentem pro užití této technologie je především existence použitých knihoven pro rychlou práci s neuronovými sítěmi (tensorflow a na něj navázaný keras). Tyto nástroje byly popsány v kapitole X. Exemplární příklady [x] a nagenerované výstupy [y] byly pro názornost modelovány a uspořádány v herním enginu Unity, který pro své skripty užívá C#. Vzhledem k přímé nekompatibilitě programovacích jazyků byla všechna data, která bylo třeba zpracovat či exportovat formátována v csv souborech.

## Definice cíle a východisek

Tato kapitola představuje předpoklady, východiska, a užité abstrakce, které byly v rámci prezentované metody aplikovány. Nutno už úvodem podotknout, že navržený algoritmus slouží především jako demonstrace použitelnosti či nepoužitelnosti navrženého principu. Objektivně jsou v závěru práce zkonstatovány jeho ... i nedostatky. Reálná aplikace algoritmu je podmíněna jeho dalším rozpracováním.

Na nejvyšší úrovni jsou vstupní data pro algoritmus definována jako scény *S* = {s1, s2, s3...sn}. Scéna je pro nás diskrétním prostorem n2 možných pozic [x0...xn, z0...zn] uspořádaných v matici M. Na těchto pozicích se mohou vyskytovat objekty. V případě, že je pozice prázdná je tato skutečnost reprezentována nulou.

Rovnice Příklad diskrétního prostoru 32, kde xij > 0 představuje jednu konkrétní třídu objektů

Dohromady vytvářejí tyto objekty určitou množinu *O*. Tato množina vytváří smysluplnost tj. scénický význam *SV* ve dvou aspektech. Za prvé je pro *SV* podstatná samotná přítomnost nebo nepřítomnost objektů ve scéně. Například vyskytuje-li se ve scéně 5 postelí je její význam jiný než vyskytuje-li se v ní 5 krabic s dynamitem. Za druhé je význam scény v menší míře  spoluvytvářen i vzájemnými pozičními (a jinými) relacemi v diskrétním prostoru scény. Ukázáno na příkladu: Nachází-li se ve scéně 5 židlí a 5 stolů, je význam scény jiný, když se všech pět židlí nachází u stolu, než když tomu tak není. Prezentovaný algoritmus řeší především druhý aspekt scénického významu, totiž umístění objektu na nejpravděpodobnější místo vzhledem k aktuálnímu kontextu prostoru. Přítomnost jednotlivých objektových tříd ve scéně je řízena uživatelsky.

Každý objekt přítomný ve scéně sebou nese vektor své pozice, rotace a velikosti. První abstrakcí prezentovaného modelu je, že z těchto tří vlastností objektu je pro zjednodušení úlohy uvažován pouze poziční dvoudimenzionální vektor P = [x, z]. Vektory rotace a velikosti uvažovány nejsou. Kromě geometrických vlastností objektu, je každý objekt kategorizován danou třídou kterou představuje. To umožňuje udržet identitu kategorie přítomných objektů napříč jednotlivými scénami.

Z uvedného již lze názorně definovat nejprve obecný cíl a poté jednotlivé kroky, které povedou k jeho naplnění.

Obecný cíl je definován následovně: Za předpokladu výchozí scény [s0], která je definována množinou objektů [O], umístit uživatelsky vybraný objekt na logickou pozici, která podpoří scénický význam [s0]. Tím vzniká nová scéna [s]. Tento proces bude iterativní, to znamená, že v jednom kroku generovacího algoritmu bude scéna doplněna právě o jeden objekt. Načež se tato nová scéna stane východiskem pro další krok generování.

Rovnice vyjadřuje proces generování

Rovnice 8 naznačuje proces generování výsledné scény [s], kde [s0] odpovídá výchozí scéně, [n] vyjadřuje počet iterací algoritmu a [ox]představuje predikovaný objekt třídy [x].

## Popis užitého algoritmu

Celý algoritmus sestává ze tří hlavních fází. Za prvé je to fáze předzpracování dat. V druhé fázi se natrénují neuronové sítě a v poslední fázi je volána metoda generování. Modelovou situací je dogenerování uživatelsky volených objektů do prostoru libovolného ohraničeného pokoje.

### Charakteristika vstupních dat

Pro všechny testované případy byla užita vstupní datová sada čítající 23 ručně modelovaných příkladů. Scénický význam těchto scén je definovaný jako pokoj. Následující tabulka představuje logiku, se kterou byly předměty do scén umísťovány.

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Interpretace | Typ | Logika umístění |
| 0 | Prázdný prostor | SS | Reprezentuje prázdný **vnitřní** prostor mítnosti. |
| 1 | Stěna | SS | Stěna je vždy umístěna po obvodu čtvercové scény. |
| 2 | Stůl | O | Stůl je umístěn náhodně uvnitř prostoru scény. |
| 3 | Židle | O | Židle je objekt, na kterém sledujeme hned několik hypotéz a platí pro něj několik speciálních pravidel. Tato pravidla jsou rozvedena dále. |
| 4 | Skříň | O | Skříň je umístěna vždy v blízkém kontaktu se stěnou. Sledujeme tedy tuto vlastnost. |
| 5 | Objekt ve volném prostoru | O | Objekt volného prostoru, je abstraktní objekt, který je v rámci scén vždy umístěn do prázdného prostoru. |
| 6 | Okno | O | Objekt umístěný ve zdi. Je dbáno na to aby se před ním nevyskytovaly objekty třídy **4** a **3**. |
| 9 | Prostor za zdí | SS | Abstraktní objekt, který simuluje neznámý prázdný **vnější** prostor. Slouží jako protiklad k objektu **0** a pomáhá při predikci prostorů s nepravidelným tvarem. |

Tabulka názorně popisuje objekty přítomné ve vstupních datech

### Předzpracování dat

Tato fáze je operačně nejnáročnější krokem, ve kterém dochází primárně k vektorizaci dat ze vstupních příkladů a vytvoření dvojic [x], [y], které na konci vytvoří datovou sadu pro neuronové sítě.

Parametry pro tuto fázi jsou čtyři pole. První pole [D] drží data ke všem vstupním příkladům. Druhé pole [C] definuje jádra a jejich tvary, pomocí nichž jsou procházeny vstupní data. Pole [S] drží speciální symboly [SS] viz. Tabulka 1. Jsou to třídy, pro které nebudeme chtít predikovat jejich polohu (například podlaha). Čtvrté pole [O] drží všechny třídy objektů v datech včetně [SS].

1. **For** každé jádro v C **do:**

2. **For** každý příklad v D **do:**

3.V = x\_vektory\_po\_projití\_jádrem\_C()

5. **For** každý vektor v poli V **do:**

6. **For** každý objekt z O, který se nachází ve V **do:**

7.vytvor\_kombinace\_z\_pritomnych\_objektu\_stejne\_tridy()

8. pro\_tyto\_kombinace\_vytvor\_vektory\_x()

9. pro\_tyto\_kombinace\_vytvor\_vektory\_y()

Pseudokód výše vyjadřuje iterativní práci se vstupními daty. Každý příklad je pro každé jádro transformován („rozřezán“) na x vektorů velikosti příslušného jádra. Jádro se pohybuje po vstupní matici zleva doprava směrem dolů vždy o jeden krok, podobně jako konvoluční jádro. S operací konvoluce nemá však tato metoda nic společného - podobnost lze vidět pouze v pohybu po matici.

Ze vzniklých vektorů [V] se už generují dvojice [x] a [y]. Názorně je toto generování představeno na následujícím příkladu, kde generujeme vstupní příklad pro třídu 4 a velikost jádra ().

Jako vstup pro neuronové sítě se obecně hodí normalizované hodnoty. V rámci prezentovaného algoritmu spočívá tato normalizace v zakódování vstupního vektoru [V1]na tvar [x1a]. Kde každý jedničkový příznak v trojici označuje přítomnost objektu dané kategorie. Například - označuje přítomnost objektu třídy jedna na pozici V1[*a*]. Pozice příznaku v rámci trojice se odvíjí od pořadí dané objektové třídy v poli [O] = [0,**1**,4]. Počet prvků v tomto vektoru je dán vztahem P = .

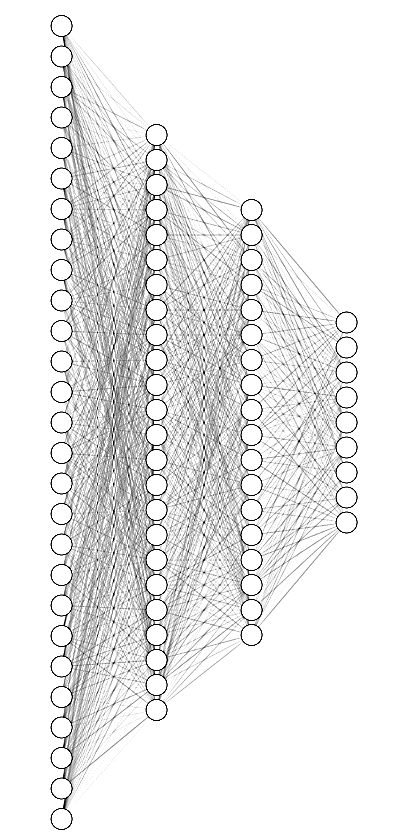
V tomto příkladu generujeme učící dvojice pro třídu 4. Proto musíme vektor [] transformovat na [], kde na pozici objektu 4 vložíme 0. Tím se vytváří vstupní vektor [x], který simuluje nepřítomnost objektu 4 na pozici V1[*f*] a výstupní vektor [y], který na pozici V1[*f*] tento objekt „vkládá“. Tato skutečnost je zvýrazněna tučnou barvou. V tomto smyslu iterujeme přes všechny jederné vektory a vytváříme dvojice [].

Při takovém postupu je přirozeně vygenerováno mnoho redundantních dvojic, neboť vektory vyprodukované jádry se často opakují. Redundace dat pro neuronovou síť je obecně diskutovaným tématem. Článek Davida Medlera ji popisuje jako pozitivní jev (Medler a Dawson nedatováno), lze se však setkat i s opačným názorem. V případě prezentovaných experimentů byly redundance zachovány s hypotézou posílení často opakovaných vzorků. Nicméně v případě větší datové sady by bylo pravděpodobně žádoucí je odstranit.

Tímto byl popsán způsob generování učících dvojic pro jeden vektor jedné třídy jádra (). Ve finálním příkladu se pracuje celkem s pěti jádry [(), (),   
), (),)] a pěti predikovatelnými objekty. Při této kvantitě vzniká na výstupu předzpracovací fáze celkem dvojic csv souborů, které agregují vektory x a y pro jednu konkrétní třídu a konkrétní jádro. Tato data budou v další fázi vstupem pro neuronových sítí.

### Natrénování modelů

Pro natrénování všech modelů byla využita jedna obecná architektura dopředné neuronové sítě. Veškeré modely byly definovány ve frameworku Keras na bázi Tensorflow s podporou učení pomocí grafické karty. Experimentováno bylo s několika různými tvary a parametry. Výsledná topologie NN se v konečné fázi ustálila na dvou skrytých vrstvách. Dimenze vstupní vrstvy odpovídá vztahu P = a je tedy pro každé jádro jiná. Počet neuronů ve srytých vrstvách se směrem k výstupní vrstvě snižuje do pyramidového tvaru s koeficientem 0.9 P pro první skrytou vrstvu a 0.7 P pro druhou. Výstupní vrstva má velikost jádra.



Obrázek Vizualizace neuronové sítě pro jádro , předpokládaje 3 objekty podobně jako v příkladu z kapitoly 6.4.2. Počet neuronů odpovídá L1 = 27, L2 =24, L3=18, L4=9. http://alexlenail.me/NN-SVG/index.html

Aktivační funkce mezi skrytými vrstvami je ReLU. Za předposlední vrstovu je za účelem pravděpodobnostního výstupu implementována funkce softmax.

## Testování hypotetických výstupů algoritmu

Postup měření úspěšnosti navrženého algoritmu je proveden následujícím způsobem. Zaprvé je vytvořena testovací datová sada. Uvnitř této sady jsou dodržována určitá relační pravidla mezi objekty. V případě, že budou tyto vlastnosti nalezeny i v algoritmicky nagenerovaných datech bude potvrzeno, že se algoritmus naučil tuto logiku replikovat.

### Výčet sledovaných vlastností

Pro objekt číslo 3 tj. židli platí následující pravidla. Za prvé platí, že je-li stůl umístěn uprostřed prostoru, jsou kolem něj židle umístěny na způsob X1.Za druhé nachází-li se stůl blízko u stěny, není již ke stěně přidána žádná židle viz X2. Toto pravidlo platí jak ve vertikálním tak horizontálním směru.

Následující příklad scény (S1) představuje prostor, ve kterém jsou zobrazeny a dodrženy všechny sledované vlastnosti.

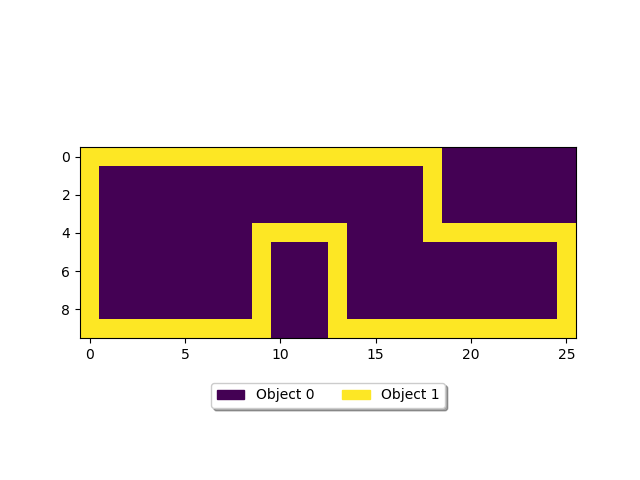
Rovnice příklad vstupního prostoru

Pro přehled je v této kapitole připraven výčet matic, které vyjadřují sledované vlastnosti.

### Jednotkové testování

V první fázi testování bylo ověřováno zda-li dokáže algoritmus replikovat konkrétní vlastnosti v jednoduchém prostoru, který není rušen kontextem ostatních objektů. Tento prostor je definovaný jako matice [M1] velikosti , po jejmíž obvodu se nachází objekt třídy 1 to znamená zeď.

Rovněž bylo ověřováno, zda-li jsou naučené vlastnosti replikovatelné i v jiných prostorech, než na kterých se algoritmu učil. K tomuto účelu byla vytvořen prostor matice [M2].



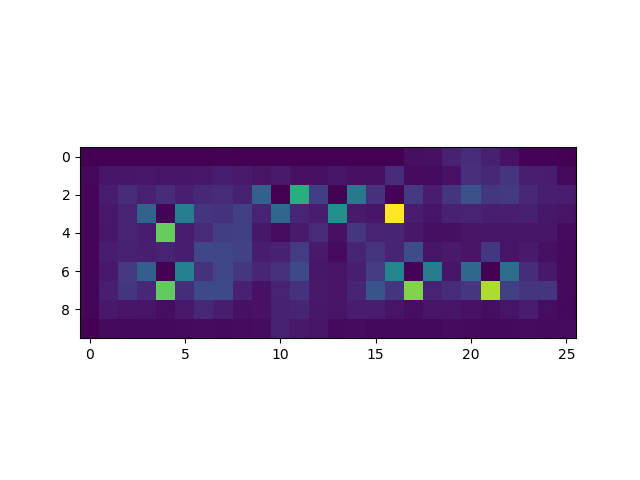
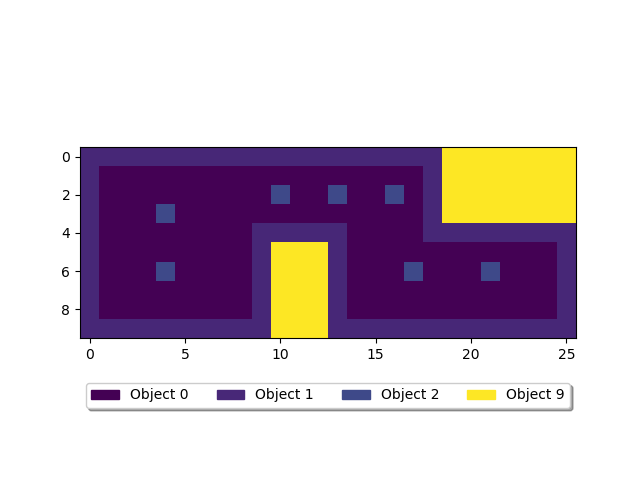
Obrázek vizualizace prostoru M2

#### Hypotéza blízké relace [X1] na [M1]

V rámci testování se potvrdilo, že algoritmus replikuje těsnou relaci objektů tříd 2 a 3. Testování bylo provedeno v prostoru matice [M1]. Ze všech 16 vnitřních pozic doplnil algoritmus v rozsahu čtyř iterací židle kolem stolů po vzoru [X1].Algoritmus zafungoval dobře i v případech, že byl ve výchozím prostoru matice umístěn větší počet stolů. Při testování na deseti scénách, ve kterých byly náhodně umístěny tři stoly (přiměřeně od sebe) obkoružil algoritmus ve všech případech stoly židlemi sledujíce vlastnosti X1 a částečně i X2.

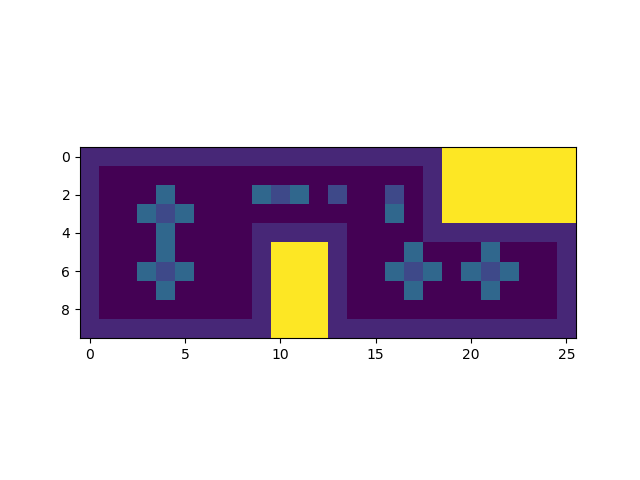
#### Hypotéza vlastností [X1] a [X2]na [M2] pro větší počet objektů

Pro účely tohoto testování byla zvolena demonstrativní situace (Obrázek 5) založená na [M2]. Celkem bylo spuštěno 30 iterací s cílem umístit objekt třídy 2 dle předpokladu [X1] a [X2].V následujícím popisu bude ukázáno a komentováno chování algoritmu.



Obrázek Vlevo vizualizace výchozí situace (iterace 0). Vpravo pravděpodobnosti výskytu objektu kategorie 3 tj, židle

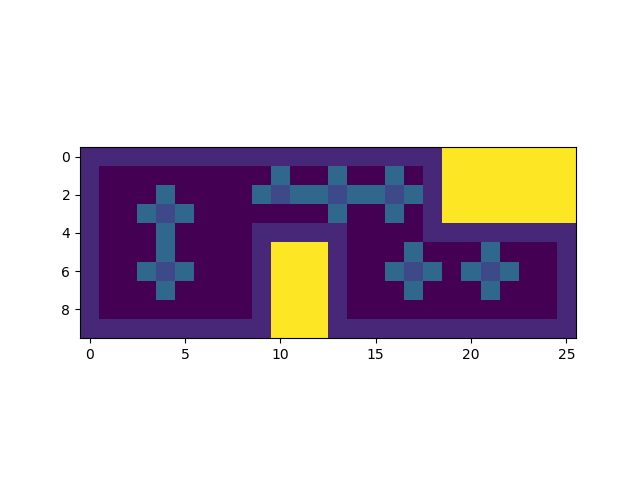
Až do 18 iterace se algoritmus držel pravidel [X1] a [X2] (viz. obrázek 6). V rámci této výchozí situace lze konstatovat, že algoritmus víceméně upřednostňoval neproblematická místa vzhledem k pravidlu [X2] (oranžová oblast). Výjímku tvoří červeně vyznačené místo, které bylo doplněno už v 8. iteraci.





Obrázek Mezistav budování prostoru (iterace 16)

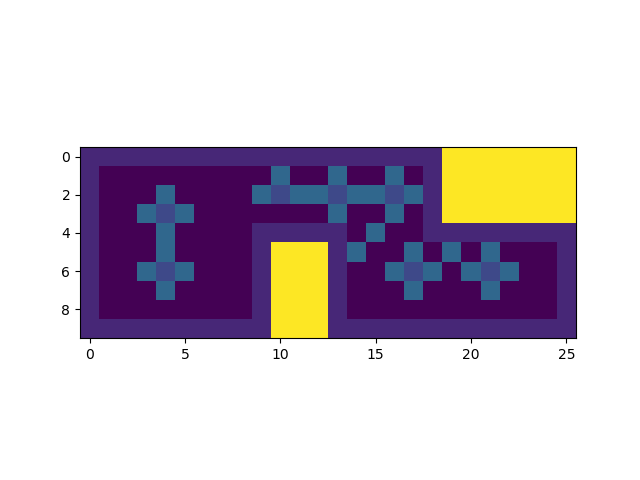
Od devatenácté iterace přestal algoritmus pravidlo [X2] dodržovat a objekty doplňoval pouze na základě pravidla [X1] viz. Obrázek 7.





Obrázek Mezistav budování prostoru (iterace 26)

Konečně je zde představen stav po 30. iteraci (Obrázek 8). V červeně označeném místě bylo algoritmem sice dodrženo pravidlo [X2], nicméně vzhledem k tomu, že židle musely být někam umístěny, byly na úkor tohoto místa umístěny nelogicky do prostoru vyznačeném zelenou barvou.



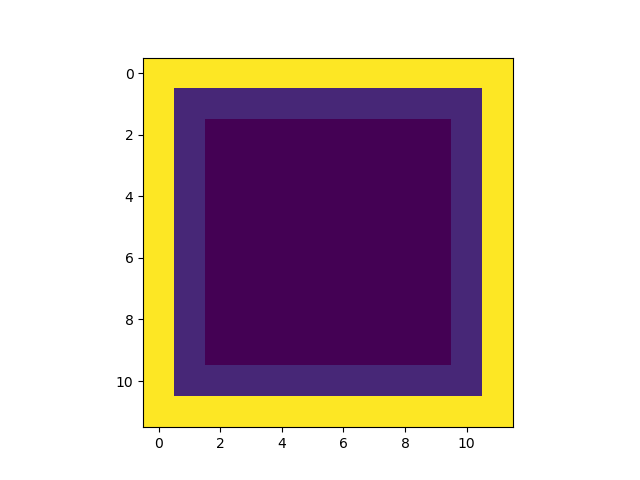
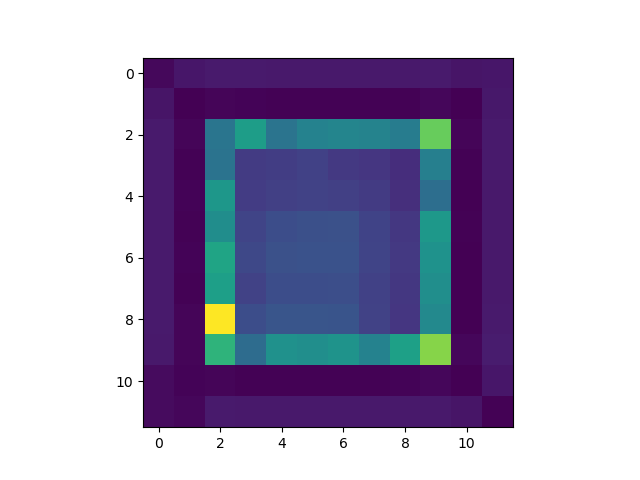


#### Hypotéza speciální vlastnosti [X2a] a [X2b]na [M1]

Pro test vlastnosti [X2] byly stoly umístěny objednu pozici od obvodového zdiva. Vznikl tak čtverec, který je na [M1] vytyčen pozicemi [3,3], [3,8], [8,3], [8,8]. Celkem vzniklo v [M1] dvacet testovacích pozic. V rozsahu tří iterací umístil algoritmus židle kolem stolu po vzoru [X2] ve všech případech. Při čtvrté iteraci došlo obvykle k tomu, že nejpravděpodobnější pozice byla dle pravidla [X1] predikována ke zdi. Pouze v šesti případech z celkových dvaceti umístil algoritmus čtvrtou židli do volného prostoru. V ostatních případech doplnil vlastnost [X2]vlastností [X1], což se dá při čtvrté itraci v podstatě identifikovat jako pozitivní jev.

#### Hypotéza vlastnosti [X3] na [M1]

Tato hypotéza předpokládá, že se objekt třídy 4, to znamená skříň bude stavět ke stěně



### Integrační testování

Z oblasti tradičního softwarového vývoje je převzat pojem integračního testu. Ten je zde používán ve smyslu otestování, jak dobře jsou sledované vlastnosti replikovatelné v případě, že do generovaného prostoru vstupuje větší variabilita objektů. Tyto testy jsou prezentovány na [M2], protože je prostorově zajímavější.

## Návrhy pro zlepšení algoritmu

Spojitost, konvoluce? Specifické žánry? Natrénování na různých prostorech. Našěptávač pozice.

# Závěr

1. Large Scale Visual Recognition Challenge - soutěž v oblasti rozpoznání objektu. [↑](#footnote-ref-1)